Secure Computation

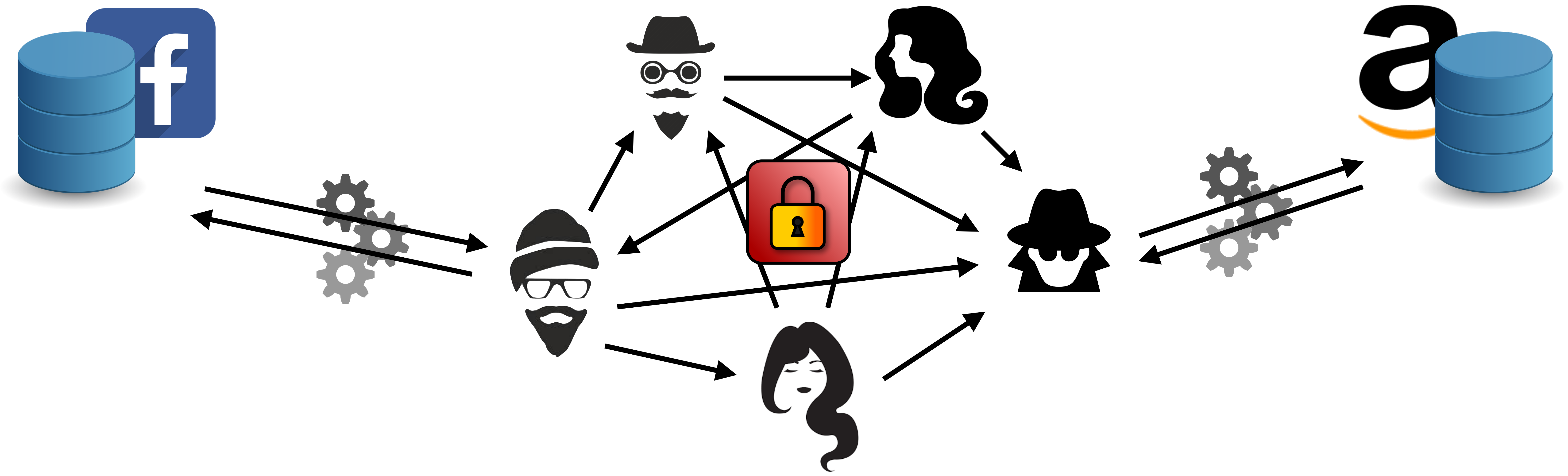
In this course, we will introduce *secure computation*, an active research area in cryptography that aims at protecting private data even when they are used in computations.

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

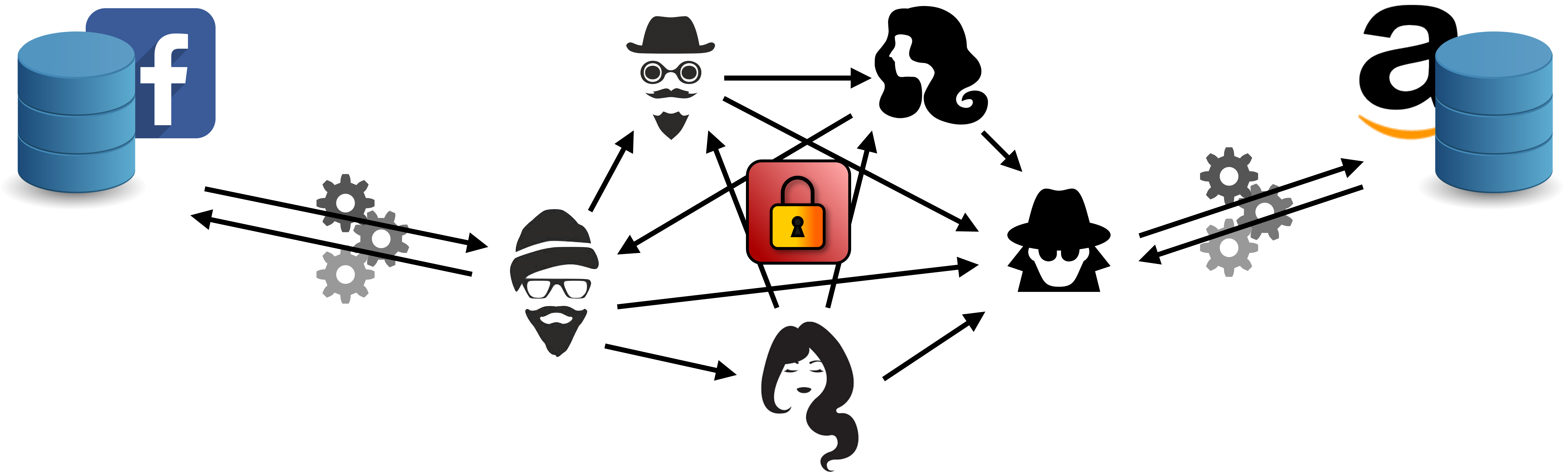
couteau@irif.fr

Secure Computation



Classical cryptography: protecting communications. However, data are not only *exchanged*: they are often *used in computations.*

Secure Computation



Classical cryptography: protecting communications. However, data are not only *exchanged*: they are often *used in computations.*

Is it possible to protect data privacy even when it’s used in computations?

Secure Computation - Examples

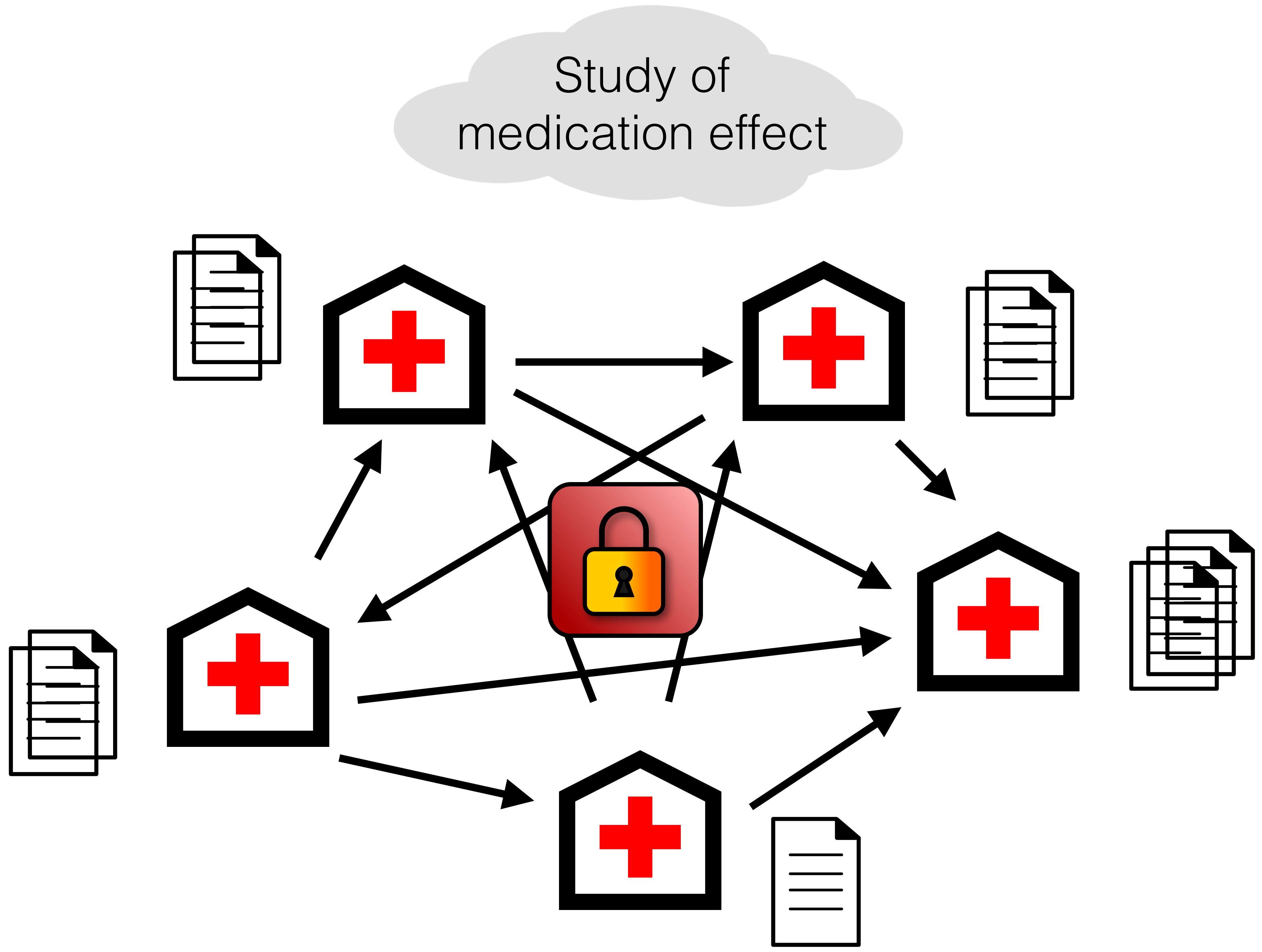
Scenarios

Hospitals hold private data of cancer patients. Access to this data would benefit to cancer research (statistics, machine learning to develop treatments, etc). The data can legally and morally not be shared.

Is it possible to compute statistics on the joint data held by hospitals, without seeing the data?

We want two properties:

- **Correctness:** everyone learns the result of the computation
- **Privacy:** nothing more than the result is learned



Model (more on that later):

- Point-to-point secure, authenticated network
- Polytime, probabilistic, interactive algorithms

Secure Computation - Examples

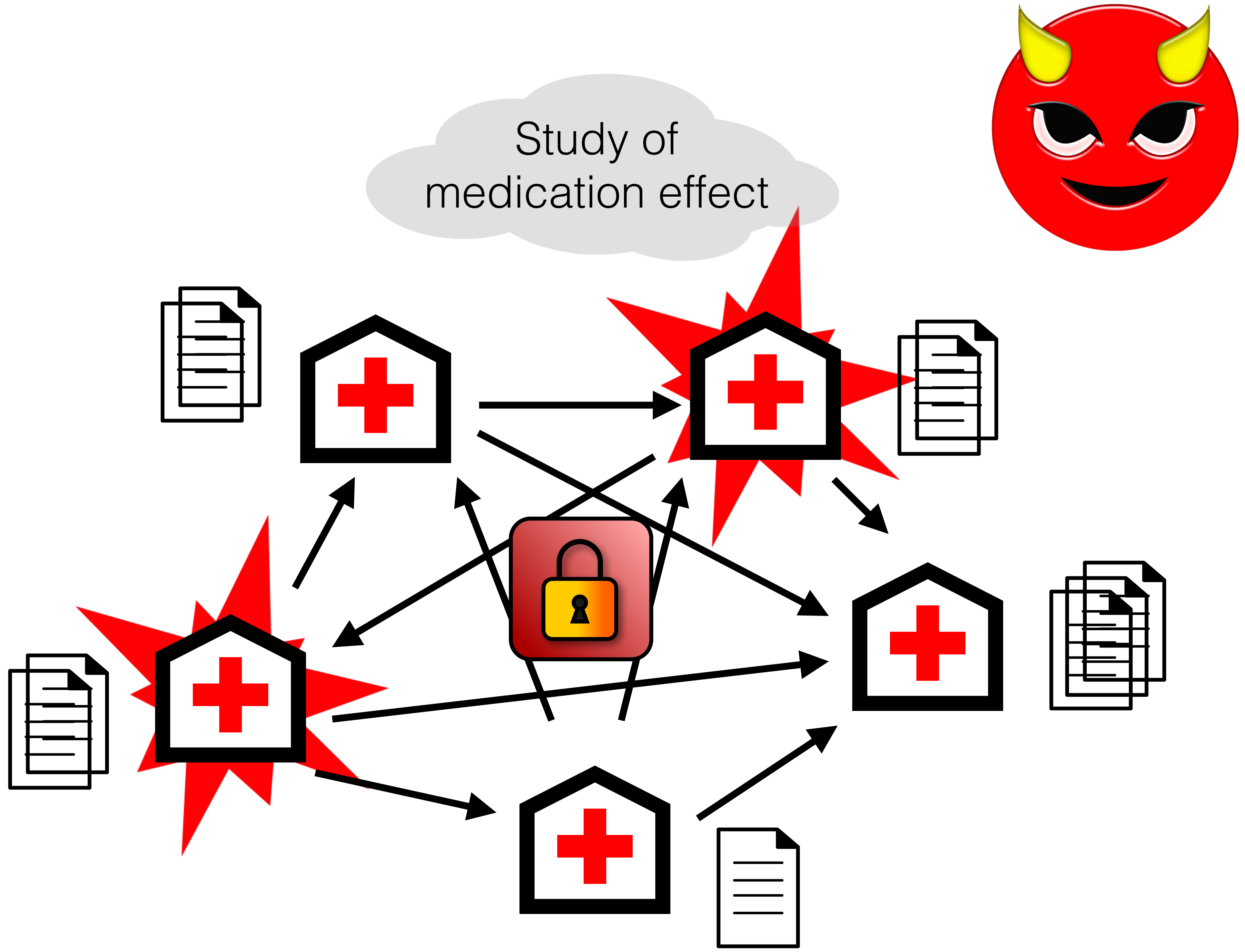
Scenarios

Hospitals hold private data of cancer patients. Access to this data would benefit to cancer research (statistics, machine learning to develop treatments, etc). The data can legally and morally not be shared.

Is it possible to compute statistics on the joint data held by hospitals, without seeing the data?

We want two properties:

- **Correctness:** everyone learns the result of the computation
- **Privacy:** nothing more than the result is learned

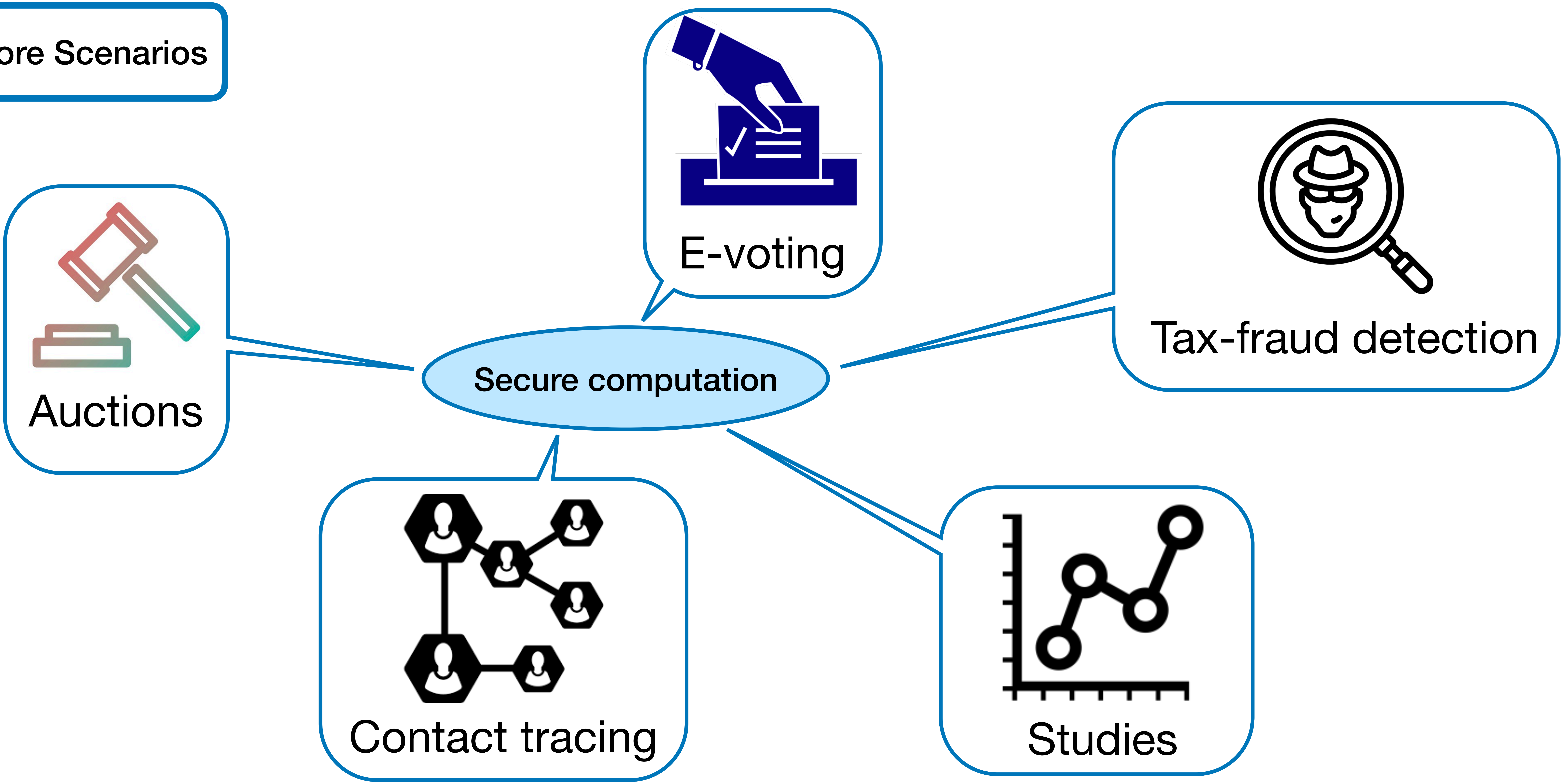


Model (more on that later):

- Point-to-point secure, authenticated network
- Polytime, probabilistic, interactive algorithms
- An adversary can corrupt (control) a subset of the parties

Secure Computation - Examples

More Scenarios



Secure computation

E-voting

Tax-fraud detection

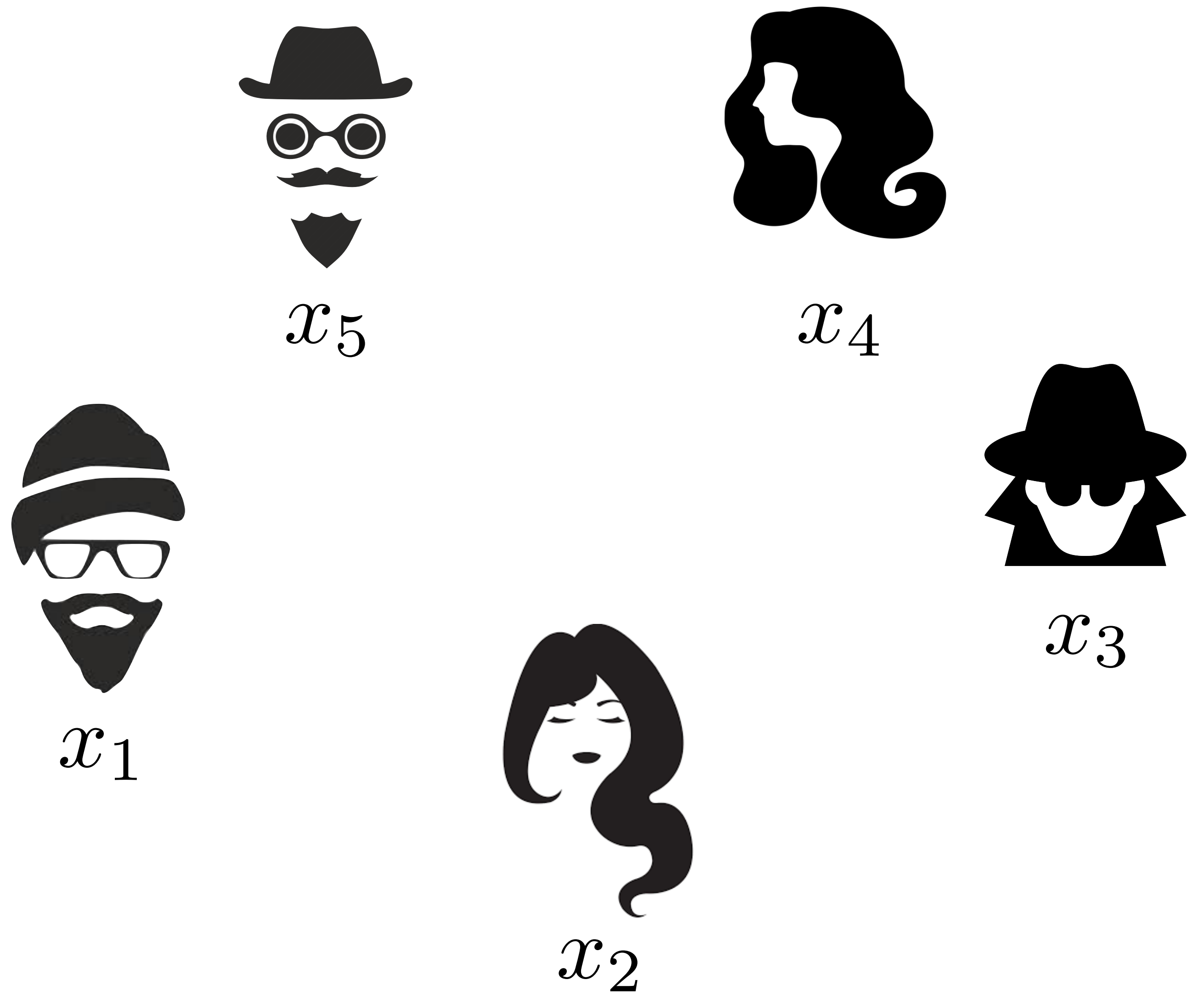
Auctions

Contact tracing

Studies

Warm-up Exercise

The problem

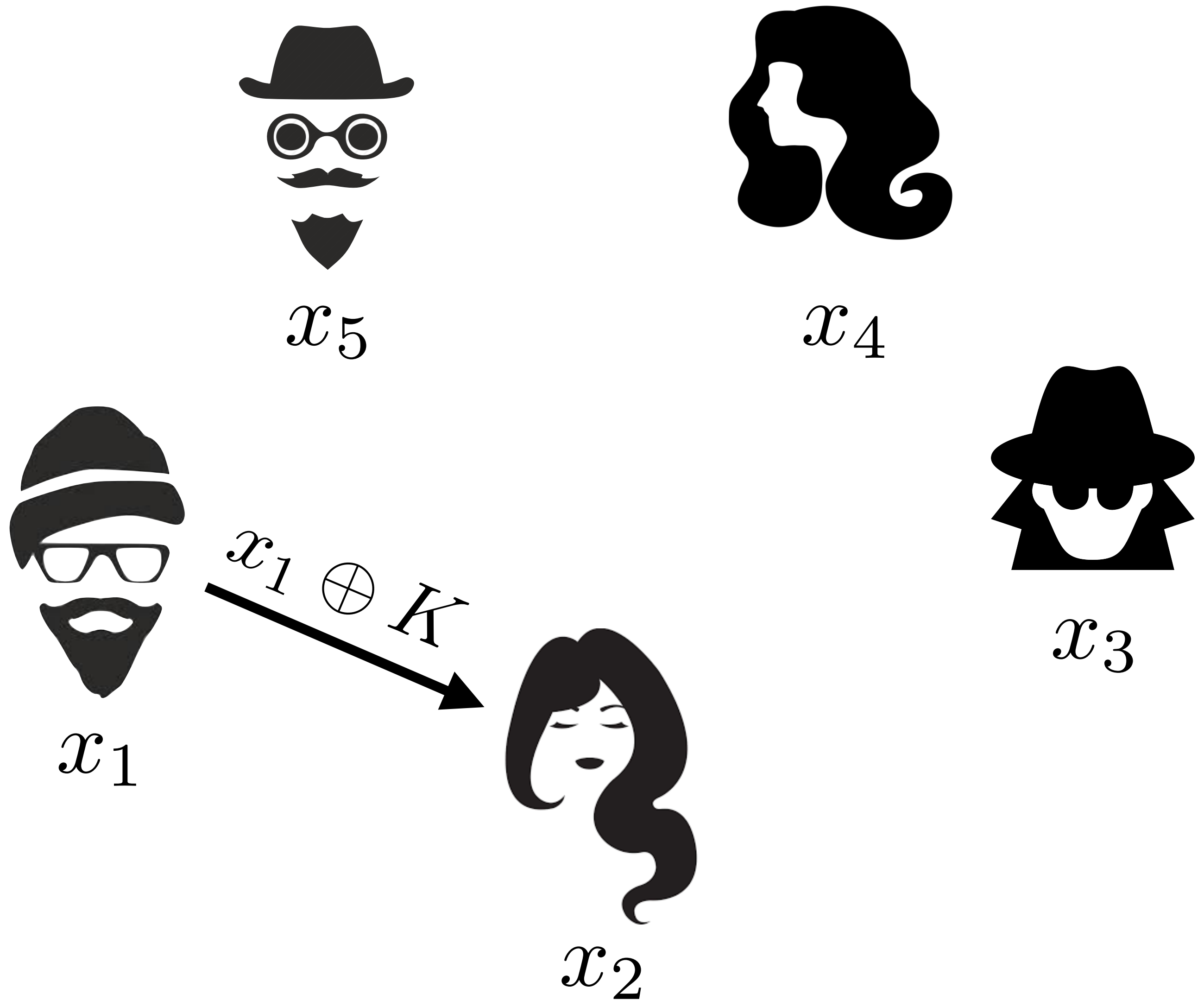


- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

Assumption: the players behave honestly. They can interact through secure and authenticated channels.

Warm-up Exercise

The problem



- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\oplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

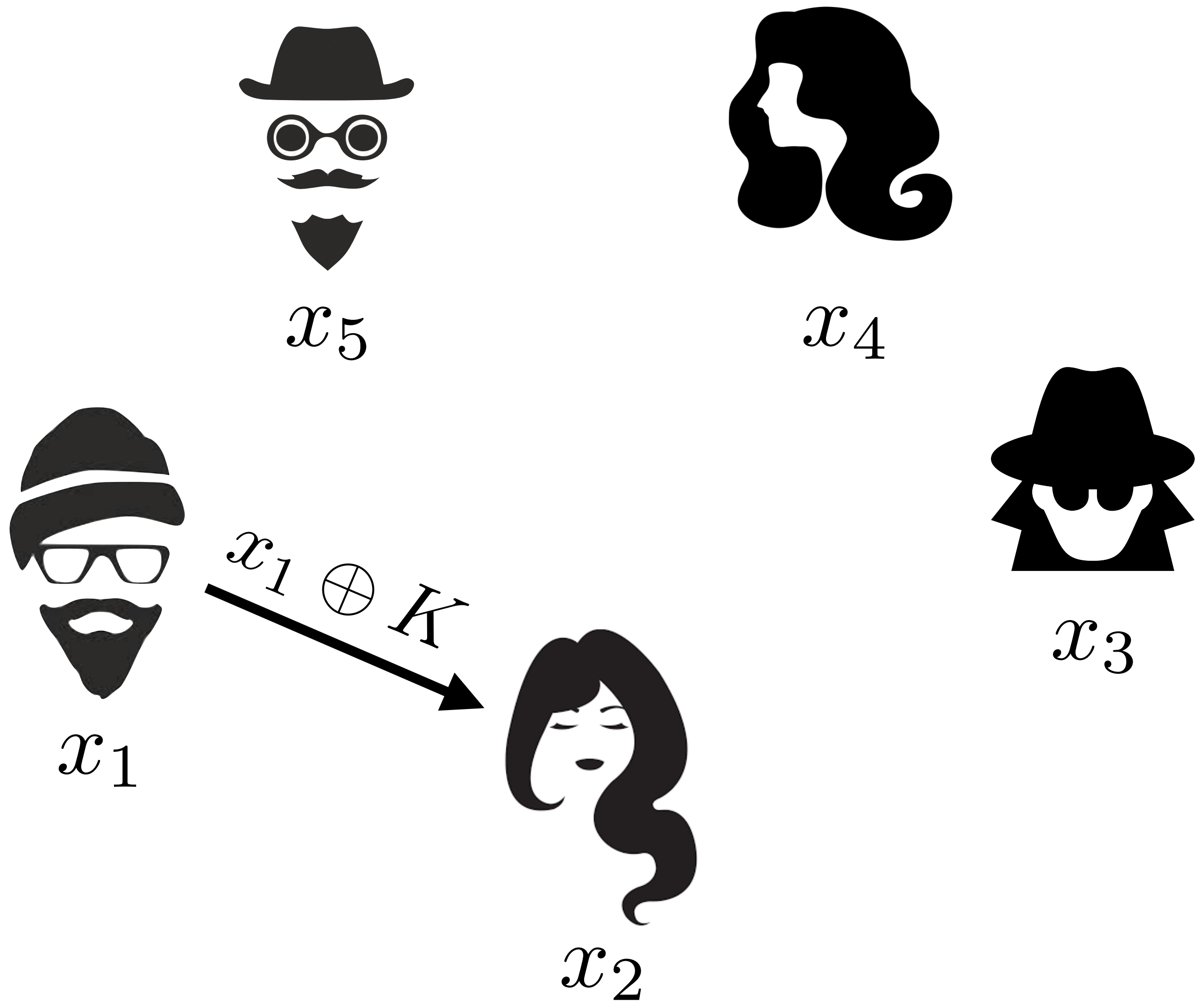
Assumption: the players behave honestly. They can interact through secure and authenticated channels.

Solution

- generates a random 128-bit string K and sends $x_1 \oplus K$ to

Warm-up Exercise

The problem



- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\oplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

Assumption: the players behave honestly. They can interact through secure and authenticated channels.

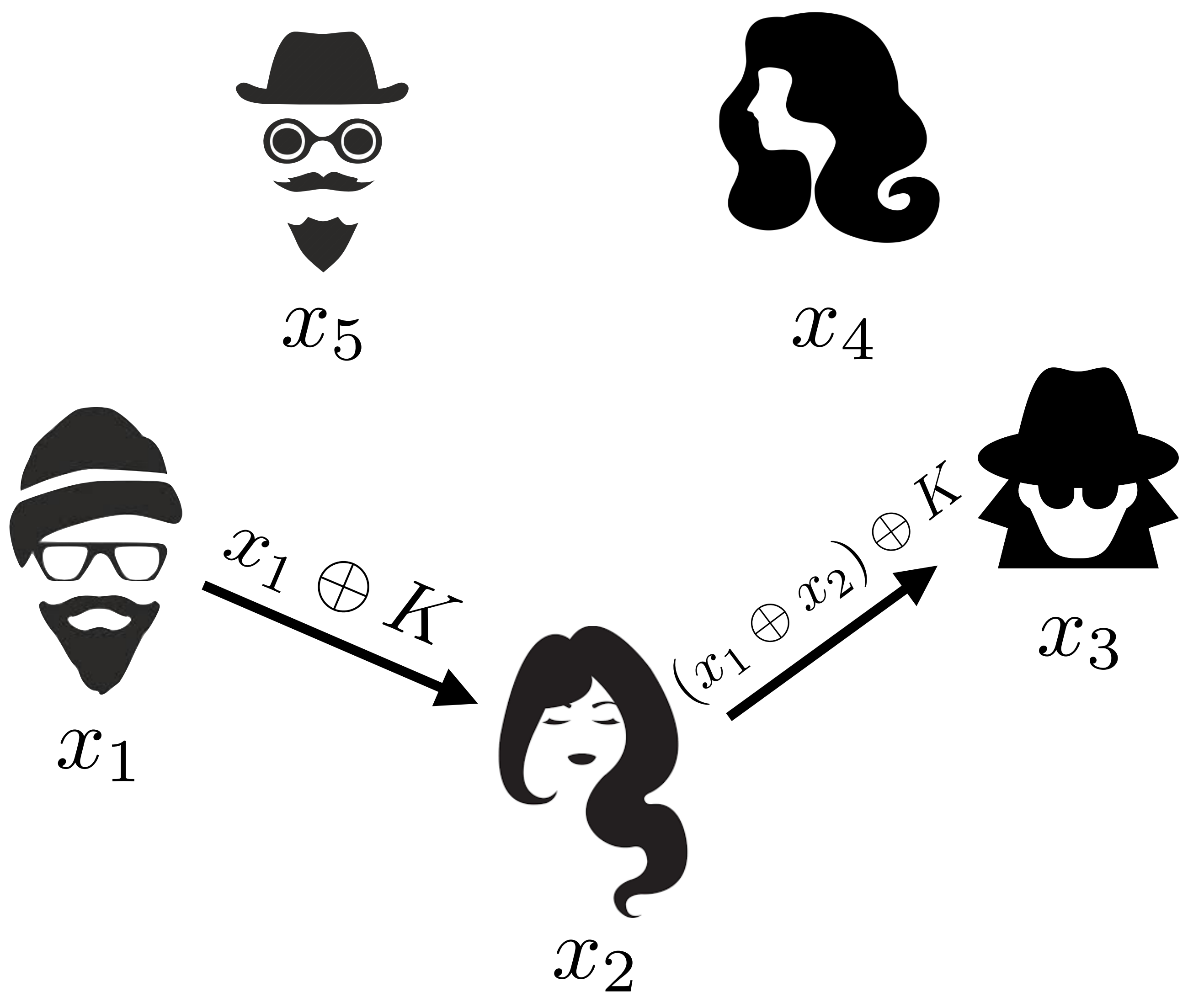
Solution

- generates a random 128-bit string K and sends $\boxed{x_1 \oplus K}$ to

This should look familiar: it's a one-time pad!
Hence, it leaks no information about x_1.

Warm-up Exercise

The problem



- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

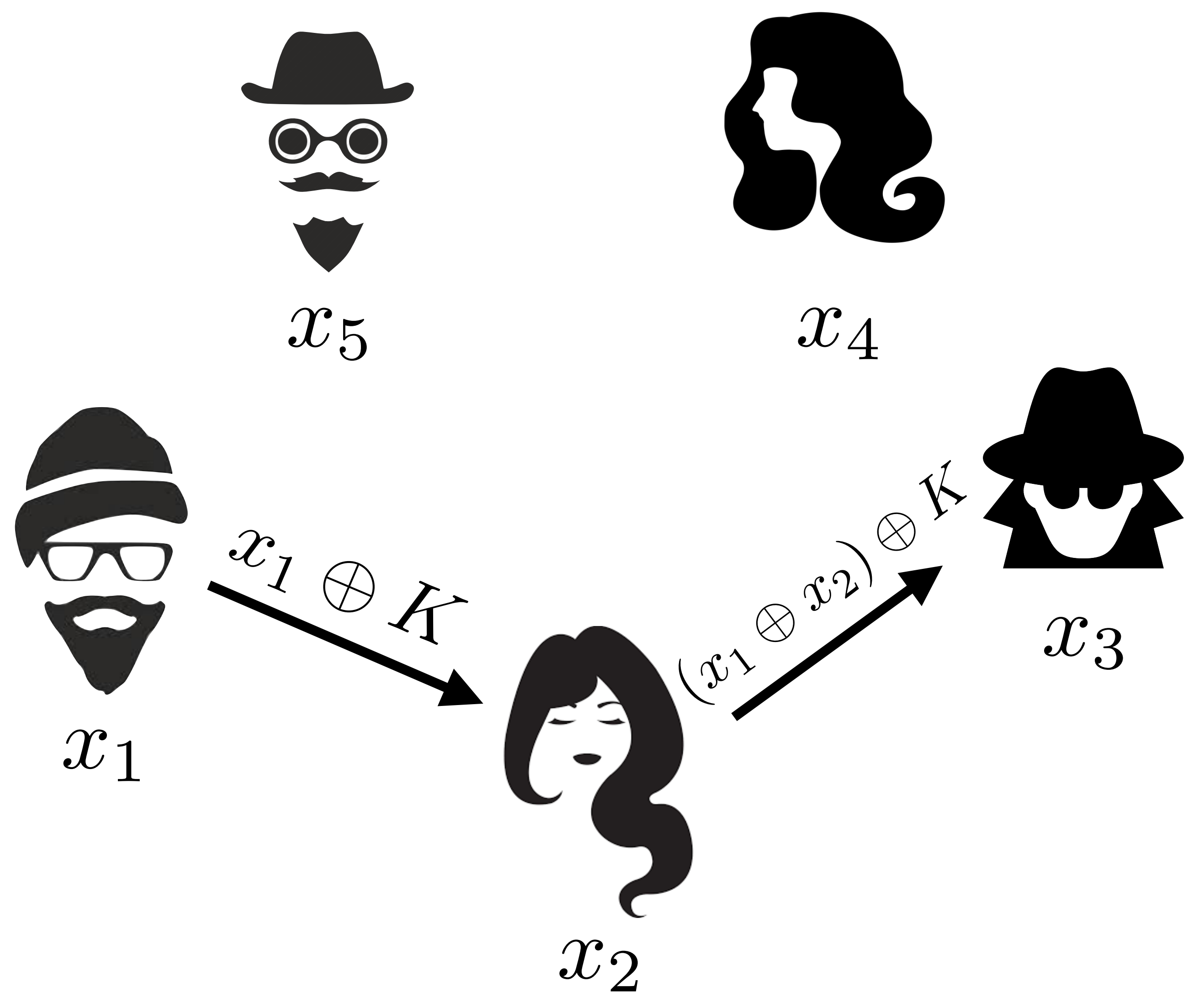
Assumption: the players behave honestly. They can interact through secure and authenticated channels.

Solution

- generates a random 128-bit string K and sends $x_1 \oplus K$ to
- computes and sends $(x_1 \oplus K) \oplus x_2 = (x_1 \oplus x_2) \oplus K$ to

Warm-up Exercise

The problem



- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

Assumption: the players behave honestly. They can interact through secure and authenticated channels.

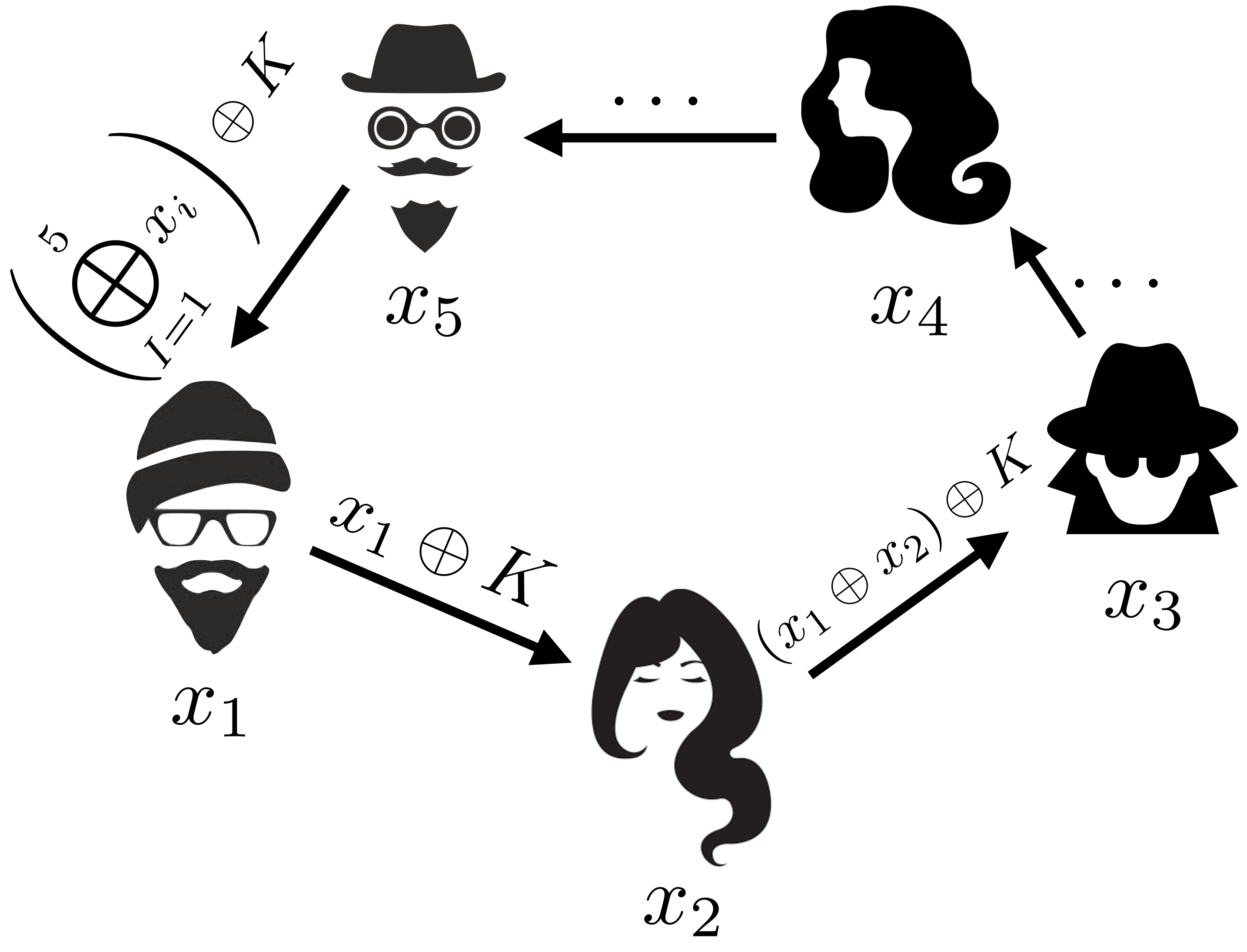
Solution

- generates a random 128-bit string K and sends $x_1 \oplus K$ to
- computes and sends $(x_1 \oplus K) \oplus x_2 = \boxed{(x_1 \oplus x_2) \oplus K}$ to

Still a one-time pad

Warm-up Exercise

The problem



$\left(\bigoplus_{i=1}^{5} x_i\right) \oplus K$
x_5
$\cdots$
x_4
$\cdots$
x_1
$x_1 \oplus K$
$(x_1 \oplus x_2) \oplus K$
x_3
x_2

- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

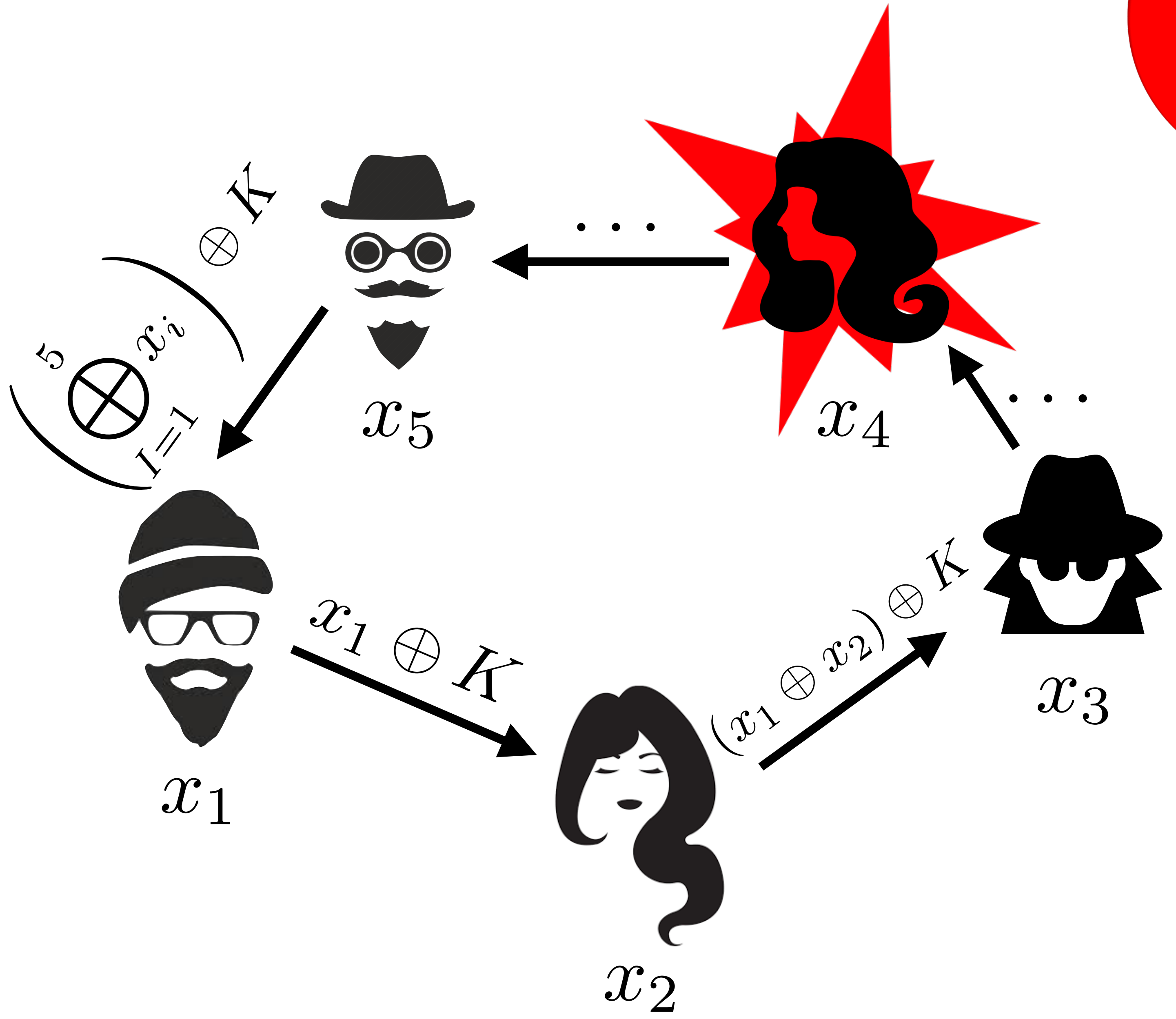
Assumption: the players behave honestly. They can interact through secure and authenticated channels.

Solution

- 🧔 generates a random 128-bit string K and sends $x_1 \oplus K$ to 👩
- 👩 computes and sends $(x_1 \oplus K) \oplus x_2 = (x_1 \oplus x_2) \oplus K$ to 🕵
- and so on… Until 🧔 gets back $(x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5) \oplus K$, removes K, and sends the result.

Warm-up Exercise

The problem



x_5
x_4
x_1
x_2
x_3
$x_1 \oplus K$
$(x_1 \oplus x_2) \oplus K$
$\left(\bigoplus_{i=1}^{5} x_i\right) \oplus K$
$\cdots$
$\cdots$

- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \bigoplus x_2 \bigoplus x_3 \bigoplus x_4 \bigoplus x_5$

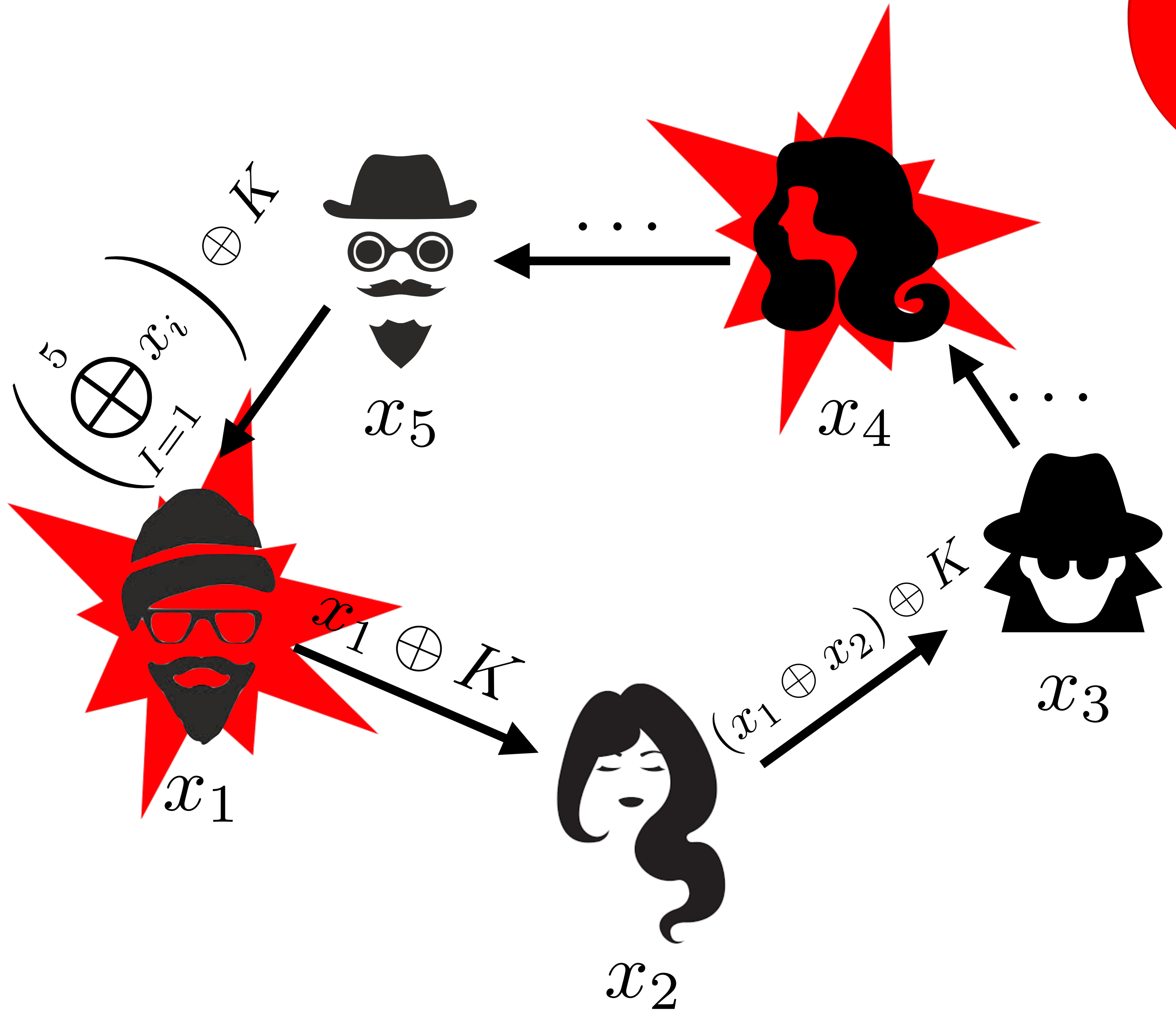
Assumption: the players behave honestly. They can interact through secure and authenticated channels.

Security

If the adversary corrupts a single party, we are fine: it sees only something masked with K

Warm-up Exercise

The problem



x_5
x_4
x_3
x_2
x_1
$x_1 \oplus K$
$(x_1 \oplus x_2) \oplus K$
$\cdots$
$\cdots$
$\left(\bigoplus_{i=1}^{5} x_i\right) \oplus K$

- Five players with respective inputs $x_1, x_2, \cdots, x_5 \in \{0,1\}^{128}$
- Goal: computing the bitwise-XOR (denoted $\bigoplus$) of all inputs: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$

Assumption: the players behave honestly. They can interact through secure and authenticated channels.

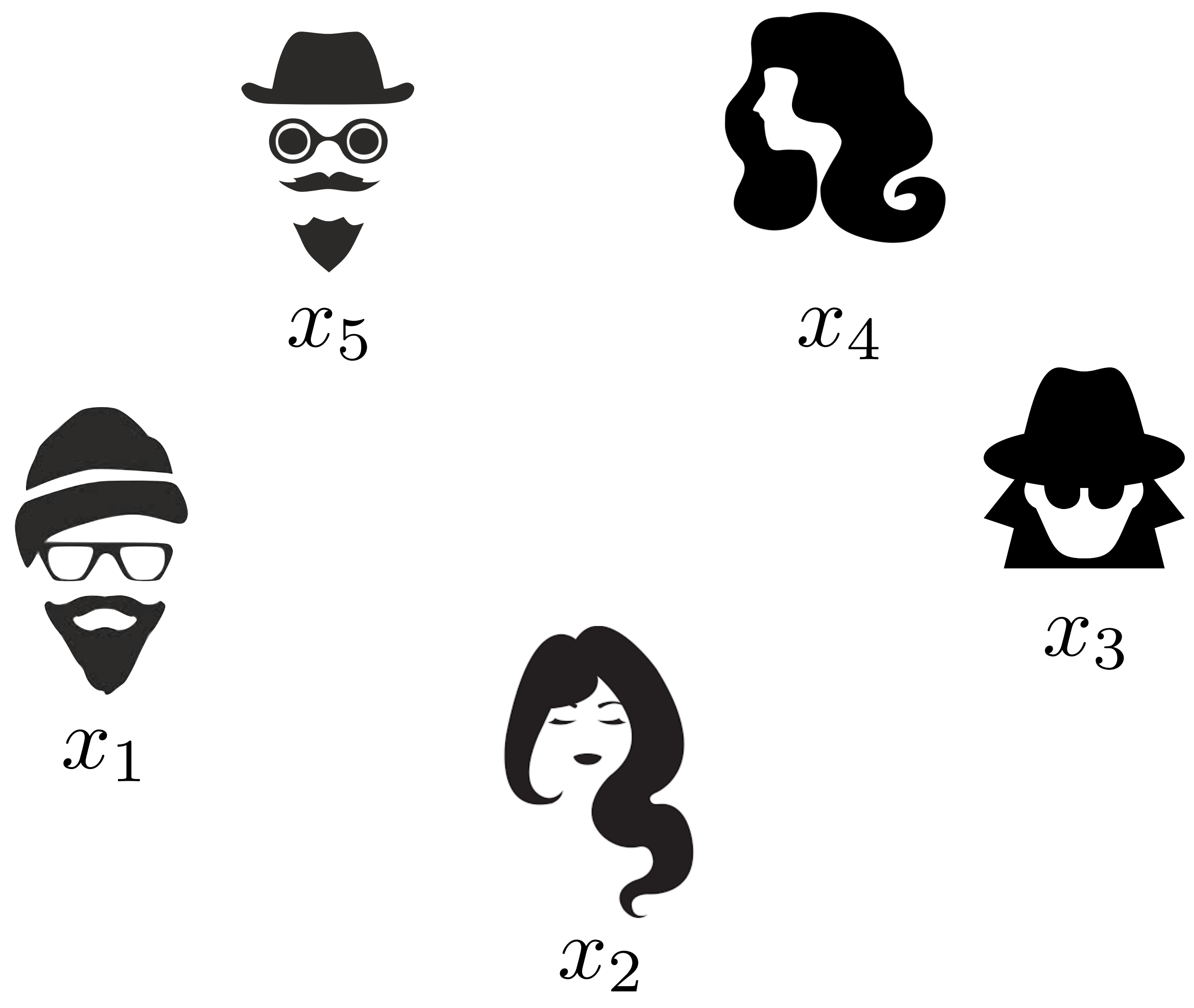
Security

If the adversary corrupts a single party, we are fine: it sees only something masked with K

In case of two corruptions, we are in trouble: the adversary can learn K!

The Model

Real World

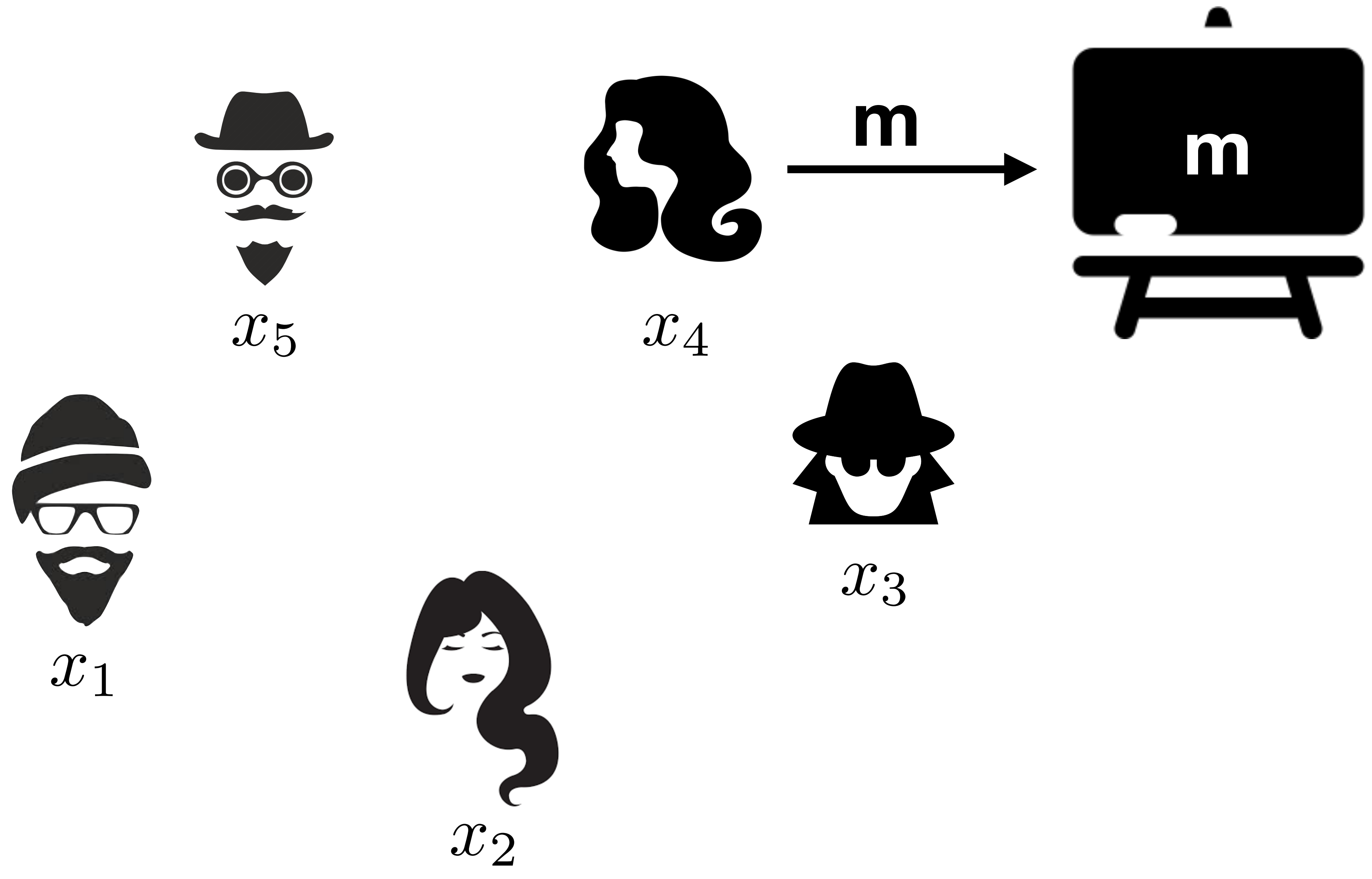


Goal

- 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

The Model

Real World



Network model

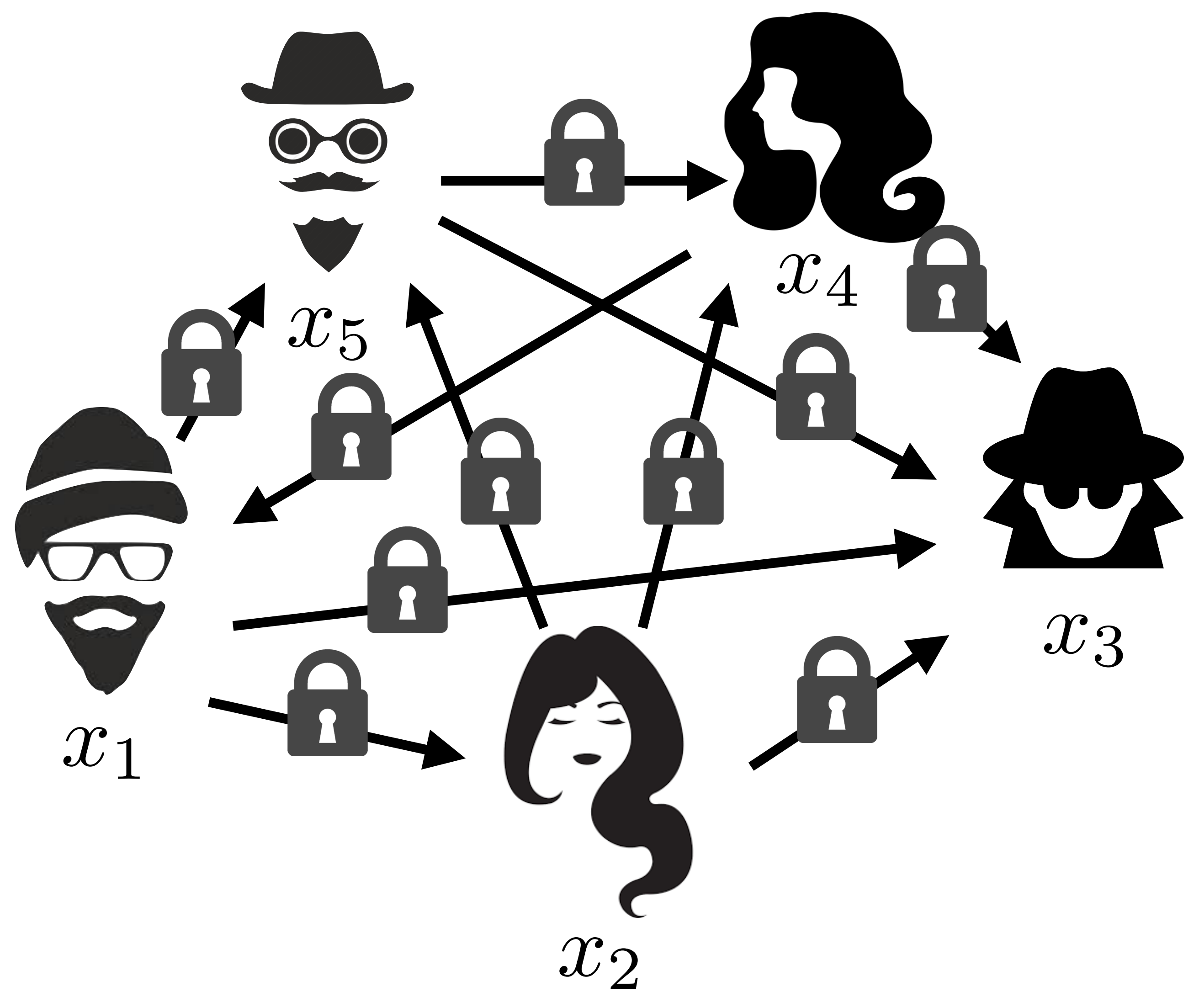
- Fully authenticated network (with signatures)
- Two communication models

1. The broadcast (or blackboard) network

Each party with message **m** can write it on a public blackboard. Everyone can see what is written on the board. All messages are authenticated.

The Model

Real World



Network model

- Fully authenticated network (with signatures)
- Two communication models

1. The broadcast (or blackboard) network

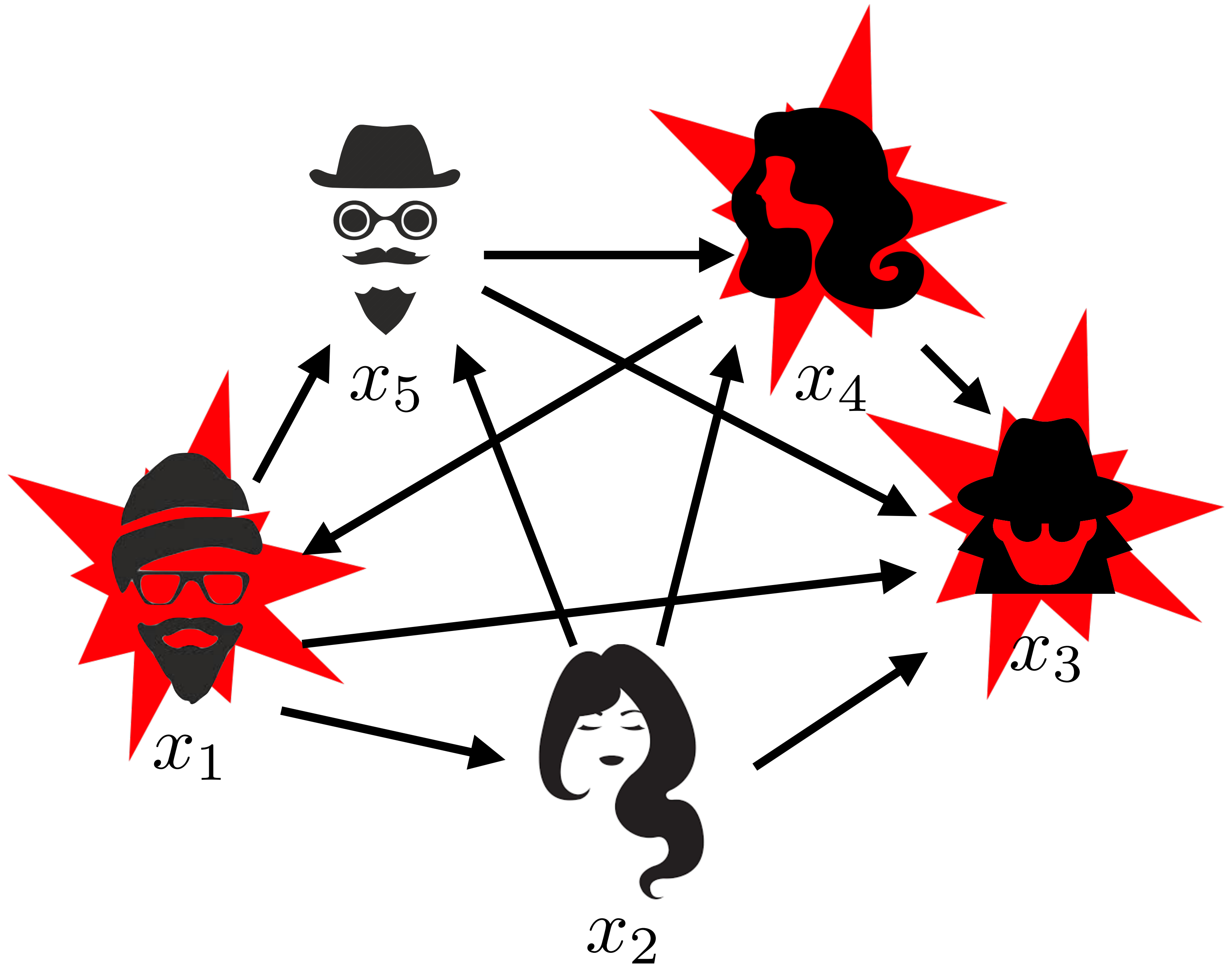
   Each party with message **m** can write it on a public blackboard. Everyone can see what is written on the board. All messages are authenticated.

2. The point-to-point network

   All parties are connected through a complete point-to-point network. Each channel is perfectly authenticated *and* private.

The Model

Real World



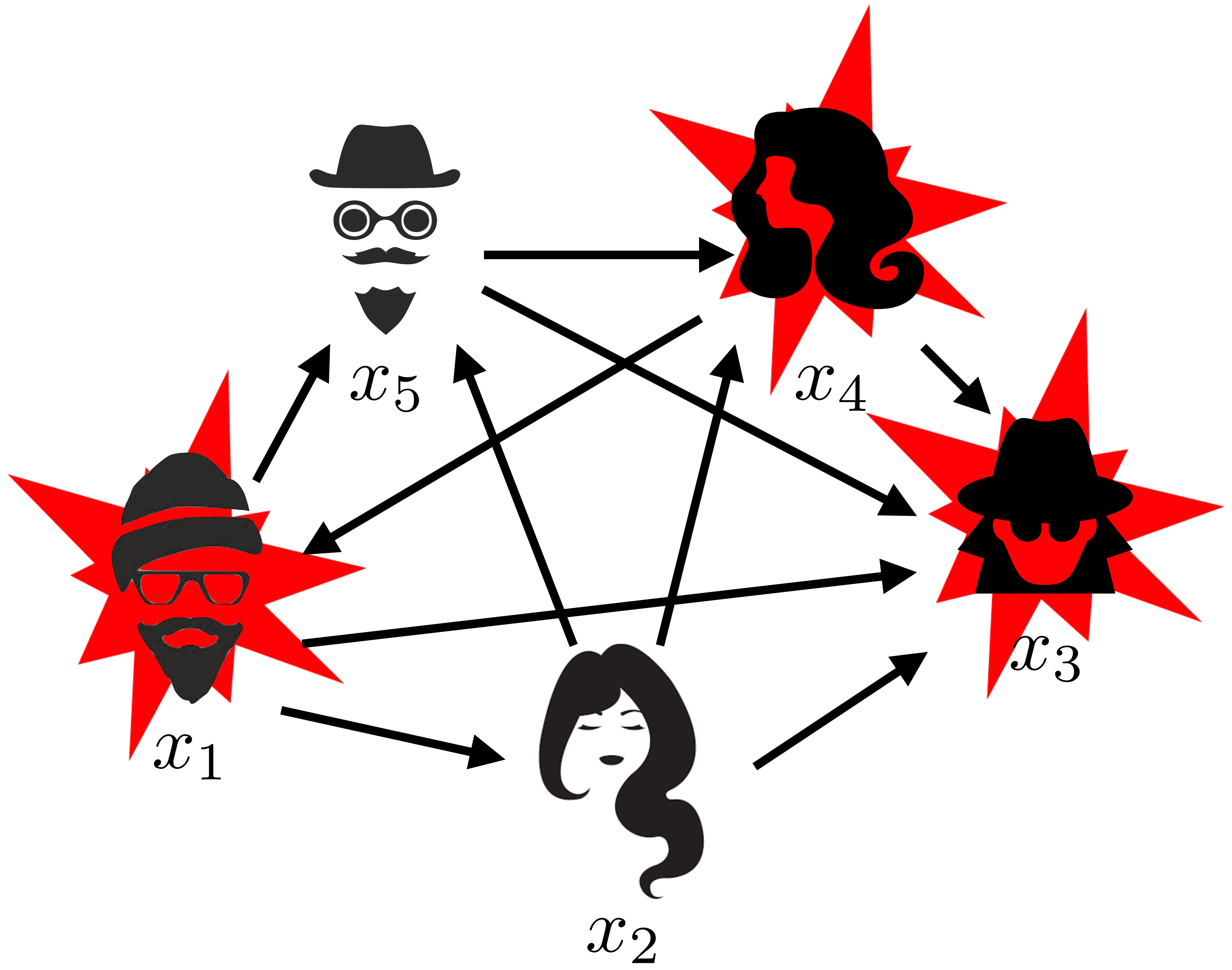
Adversarial model

- An adversary can corrupt a subset of the players

The adversary sees everything a corrupted player sees:
its private input, and all messages it sends or receive.

The Model

Real World



Adversarial model

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

1. Honest-but-curious corruption

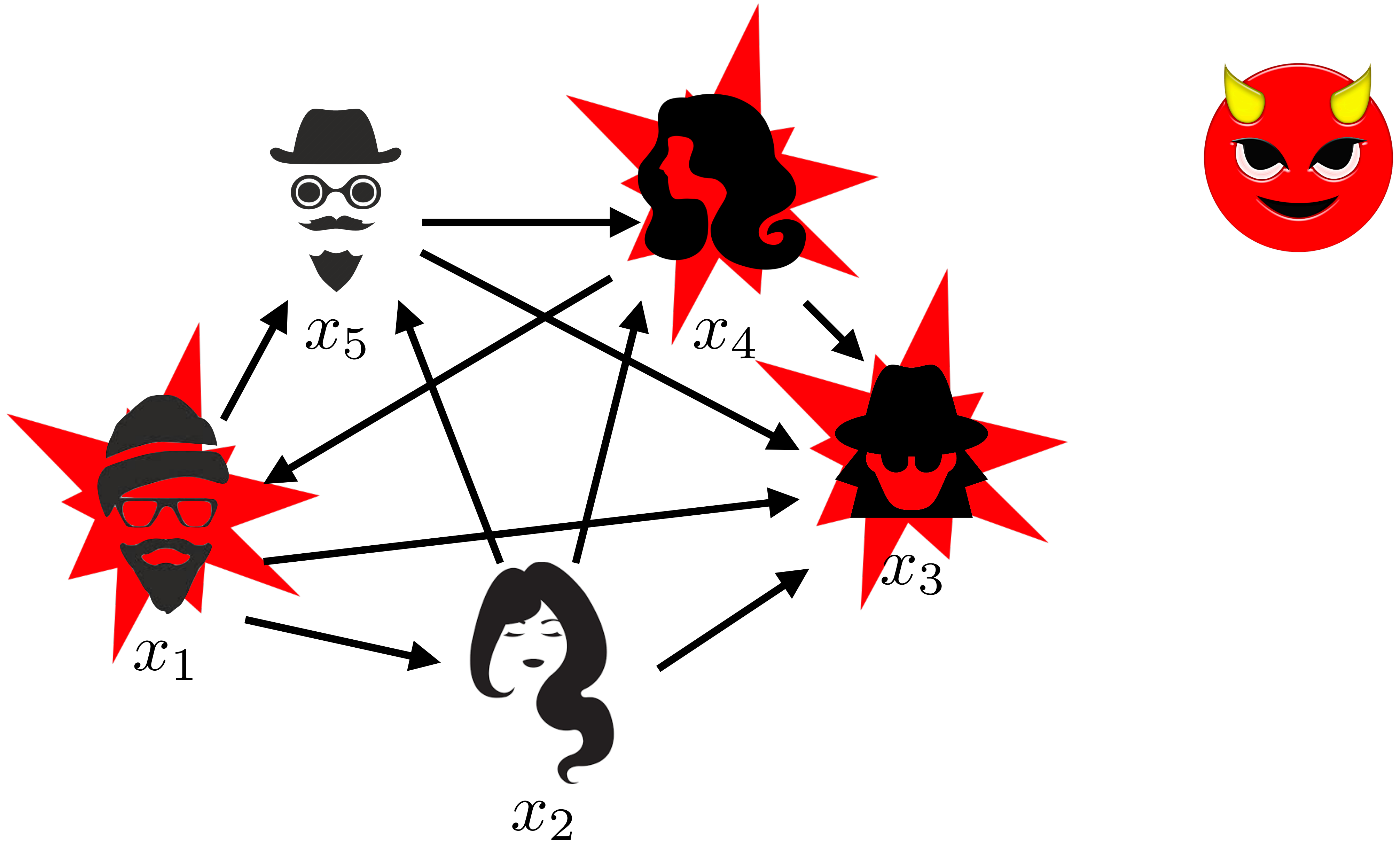
   The corrupted parties follow the specification of the protocol. The adversary is passive: he tries to retrieve private information by observing the transcript.

2. Malicious corruption

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

The Model

Real World



Adversarial model

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

1. Honest-but-curious corruption

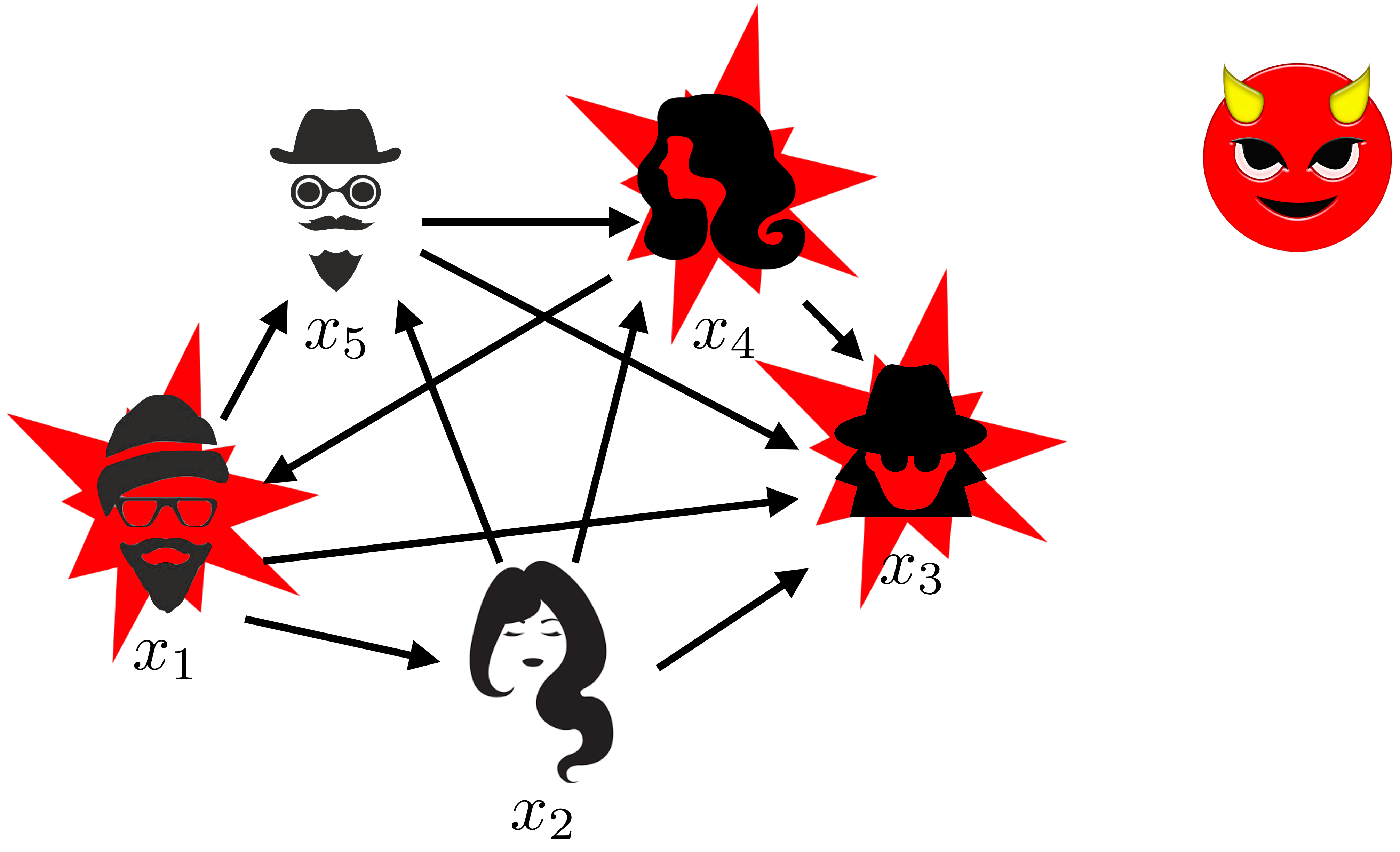
The corrupted parties follow the specification of the protocol. The adversary is passive: he tries to retrieve private information by observing the transcript.

2. Malicious corruption

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

The Model

Real World



Adversarial model

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

1. Honest majority

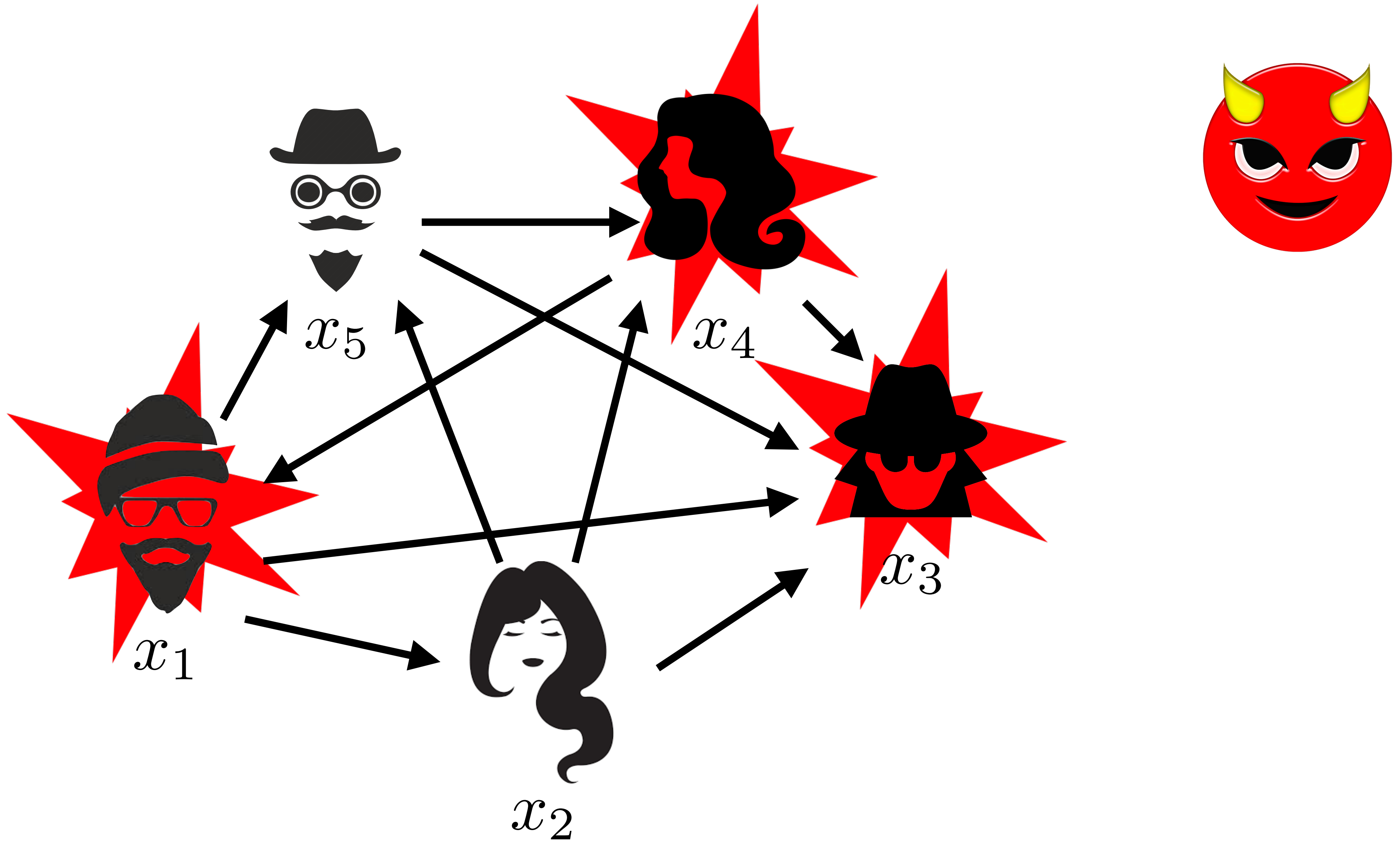
   The adversary can simultaneously corrupt only a strict minority of the players.

2. Dishonest majority

   The adversary can corrupt all-but-one players.

The Model

Real World



Adversarial model

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

1. Honest majority

   The adversary can simultaneously corrupt only a strict minority of the players.

2. Dishonest majority

   The adversary can corrupt all-but-one players.

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability.
If you are not, please say so!

Quick recap:

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability.
If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ $\quad$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability. If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$

Support = $\{0,1\}^\lambda$

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability. If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$

Support = $\{0,1\}^\lambda$

$$\mathcal{D}_0 \approx \mathcal{D}_1 \iff \forall \text{PPT} \mathcal{A}, \forall \text{ large enough } \lambda \in \mathbb{N},$$

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability. If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ $\quad$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$

Support = $\{0,1\}^\lambda$

$$\mathcal{D}_0 \approx \mathcal{D}_1 \iff \forall \text{PPT} \mathcal{A}, \forall \text{ large enough } \lambda \in \mathbb{N},$$

Probabilistic polynomial time Turing machine

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability.
If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$

$\exists N \in \mathbb{N}, \forall \lambda > N \cdots$

Support = $\{0,1\}^{\lambda}$

$$\mathcal{D}_0 \approx \mathcal{D}_1 \iff \forall \text{PPT} \mathcal{A}, \forall \text{ large enough } \lambda \in \mathbb{N},$$

Probabilistic polynomial time Turing machine

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability. If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ $\quad \mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$

$\exists N \in \mathbb{N}, \forall \lambda > N \cdots$

Support = $\{0,1\}^\lambda$

$$\mathcal{D}_0 \approx \mathcal{D}_1 \iff \forall \mathsf{PPT} \mathcal{A}, \forall \text{ large enough } \lambda \in \mathbb{N},$$

Probabilistic polynomial time Turing machine

$$|\Pr[x \leftarrow \mathcal{D}_{0,\lambda} \; : \; \mathcal{A}(x) = 1] - \Pr[x \leftarrow \mathcal{D}_{1,\lambda} \; : \; \mathcal{A}(x) = 1]| = \mathsf{negl}(\lambda)$$

The Model - Defining Security

Computational Indistinguishability

You should be familiar with the notion of computational indistinguishability. If you are not, please say so!

Quick recap: $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$

$\exists N \in \mathbb{N}, \forall \lambda > N \cdots$

Support = $\{0,1\}^{\lambda}$

$$\mathcal{D}_0 \approx \mathcal{D}_1 \iff \forall \mathsf{PPT}\, \mathcal{A}, \forall \text{ large enough } \lambda \in \mathbb{N},$$

Probabilistic polynomial time Turing machine

$$|\Pr[x \leftarrow \mathcal{D}_{0,\lambda} \ : \ \mathcal{A}(x) = 1] - \Pr[x \leftarrow \mathcal{D}_{1,\lambda} \ : \ \mathcal{A}(x) = 1]| = \mathsf{negl}(\lambda)$$

$\forall c \in \mathbb{N}, \forall \text{ large enough } \lambda \in \mathbb{N}, \mathsf{negl}(\lambda) < 1/\lambda^c$

The Model - Defining Security

Computational Indistinguishability - Example

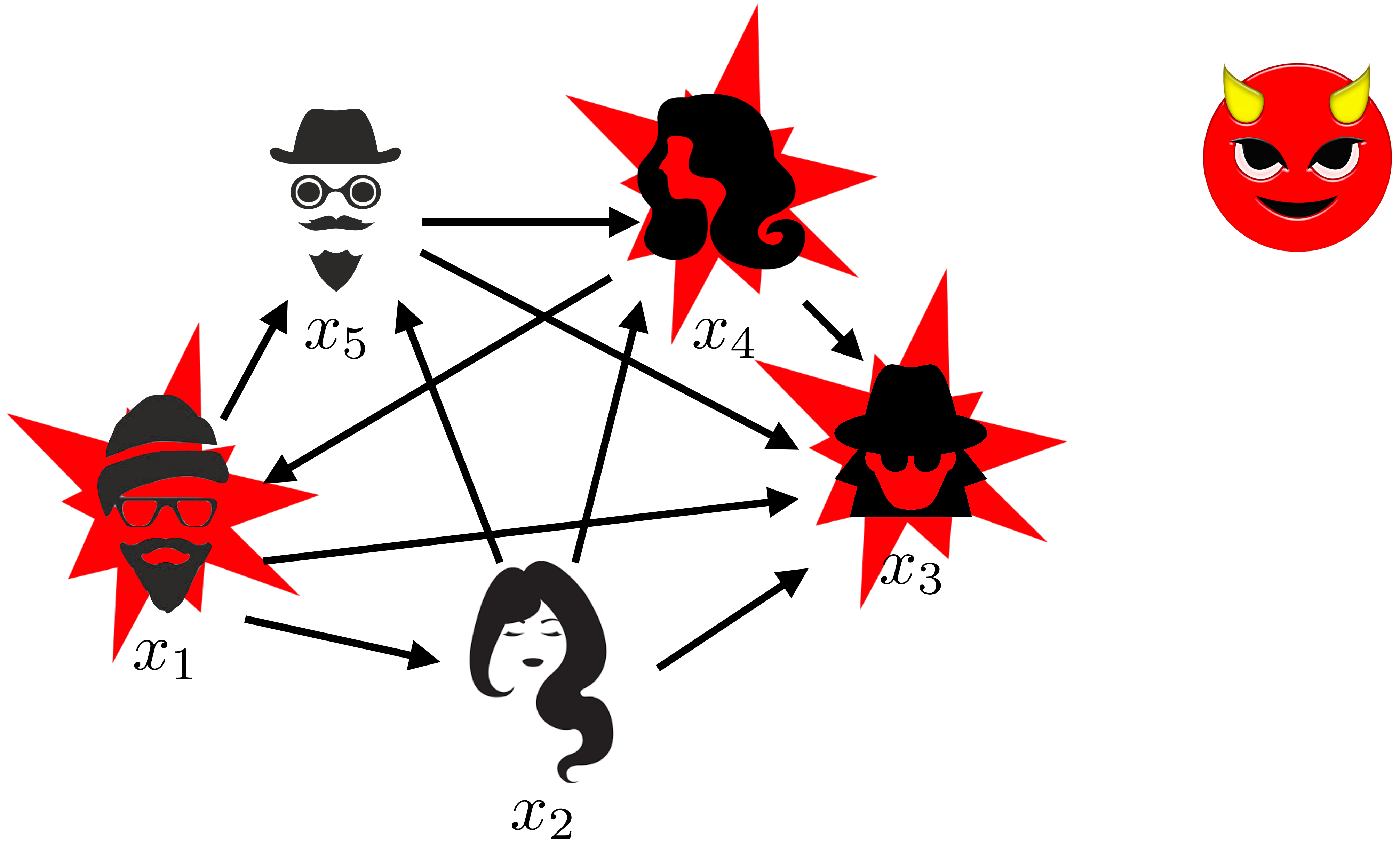
$$(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$$

$$\boxed{\{(g^a, g^b, g^{ab}) \mid (a,b) \leftarrow \mathbb{Z}_p^2\} \approx \{(g^a, g^b, g^c) \mid (a,b,c) \leftarrow \mathbb{Z}_p^3\}}$$

(this is the Decisional Diffie-Hellman assumption)

The Model - Defining Security

Real World

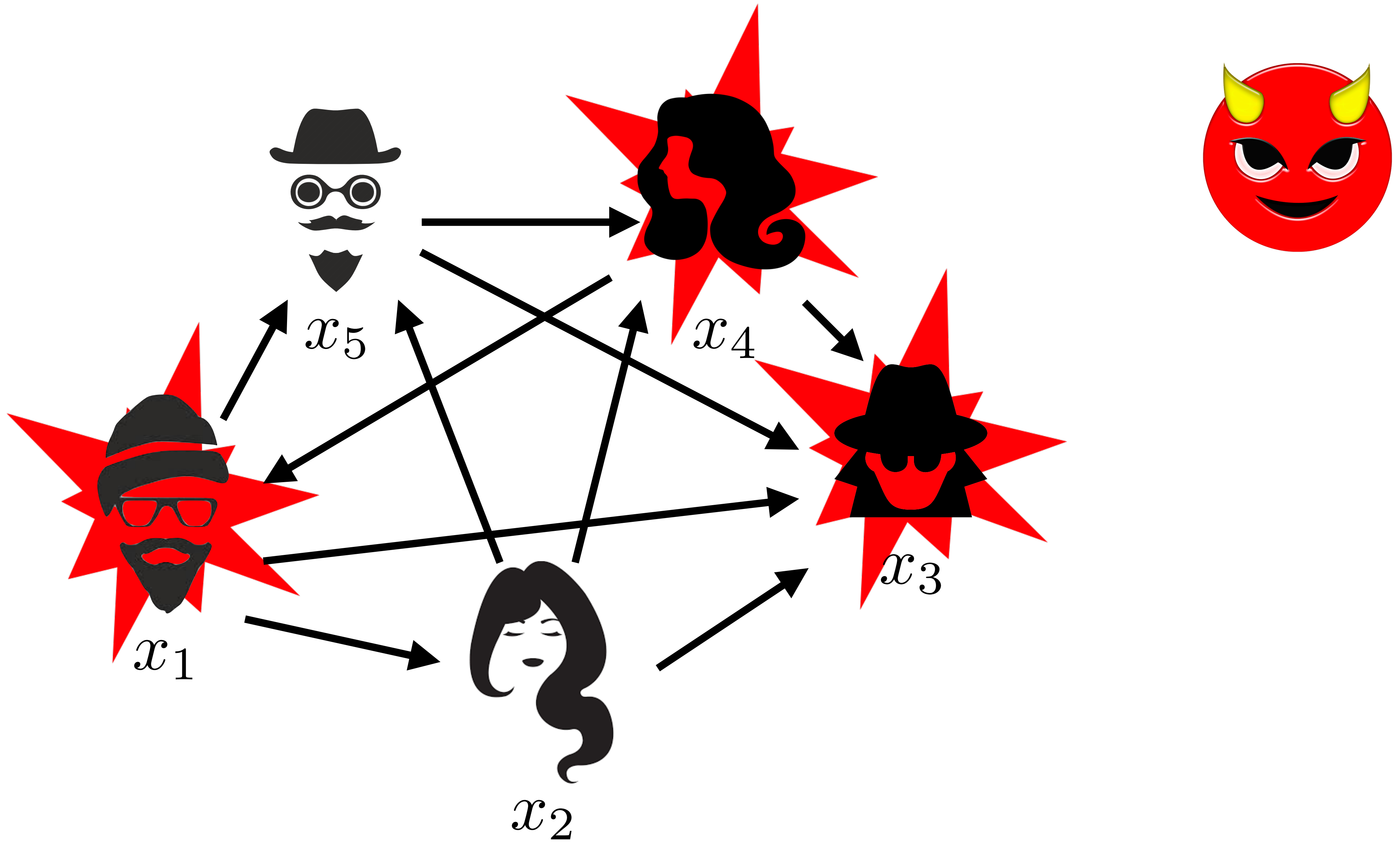


Real Behavior

- Players interact through [network model]
- Some players are corrupted in [corruption model]

The Model - Defining Security

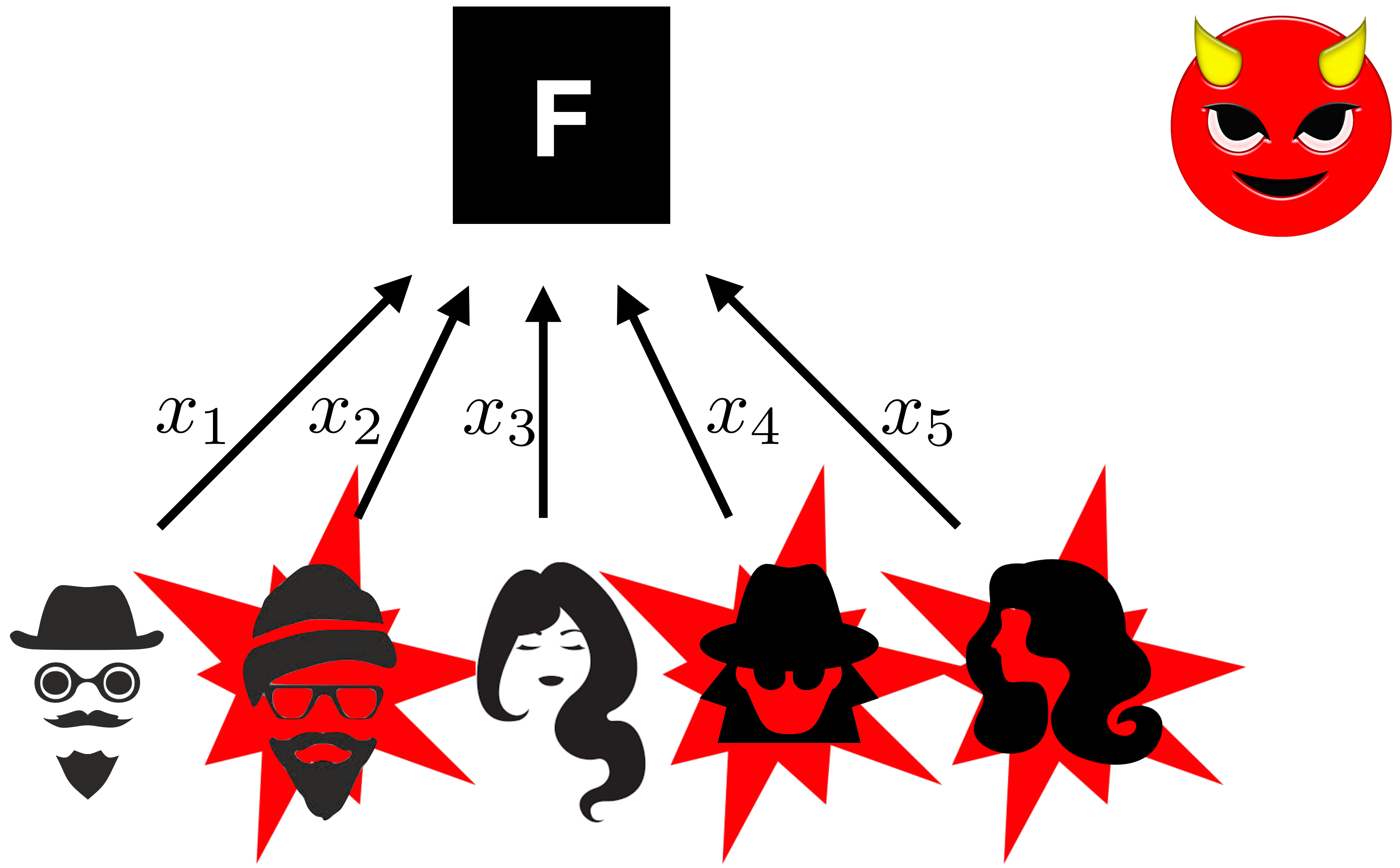
Real World



Real Behavior

- Players interact through [network model]
- Some players are corrupted in [corruption model]

Ideal World

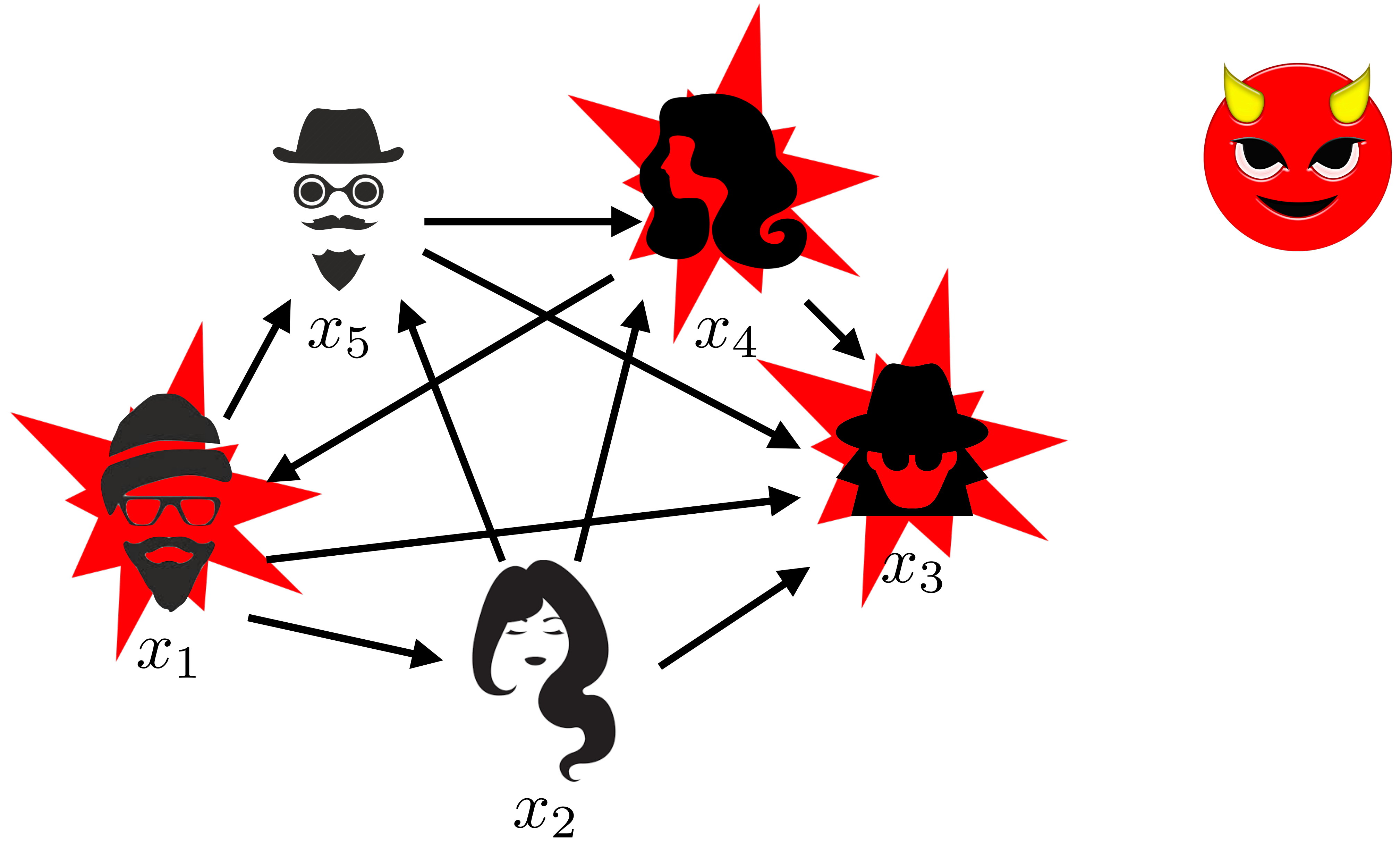


Ideal Behavior

- All parties send their input to a trusted party **F** through perfectly secure authenticated channels

The Model - Defining Security

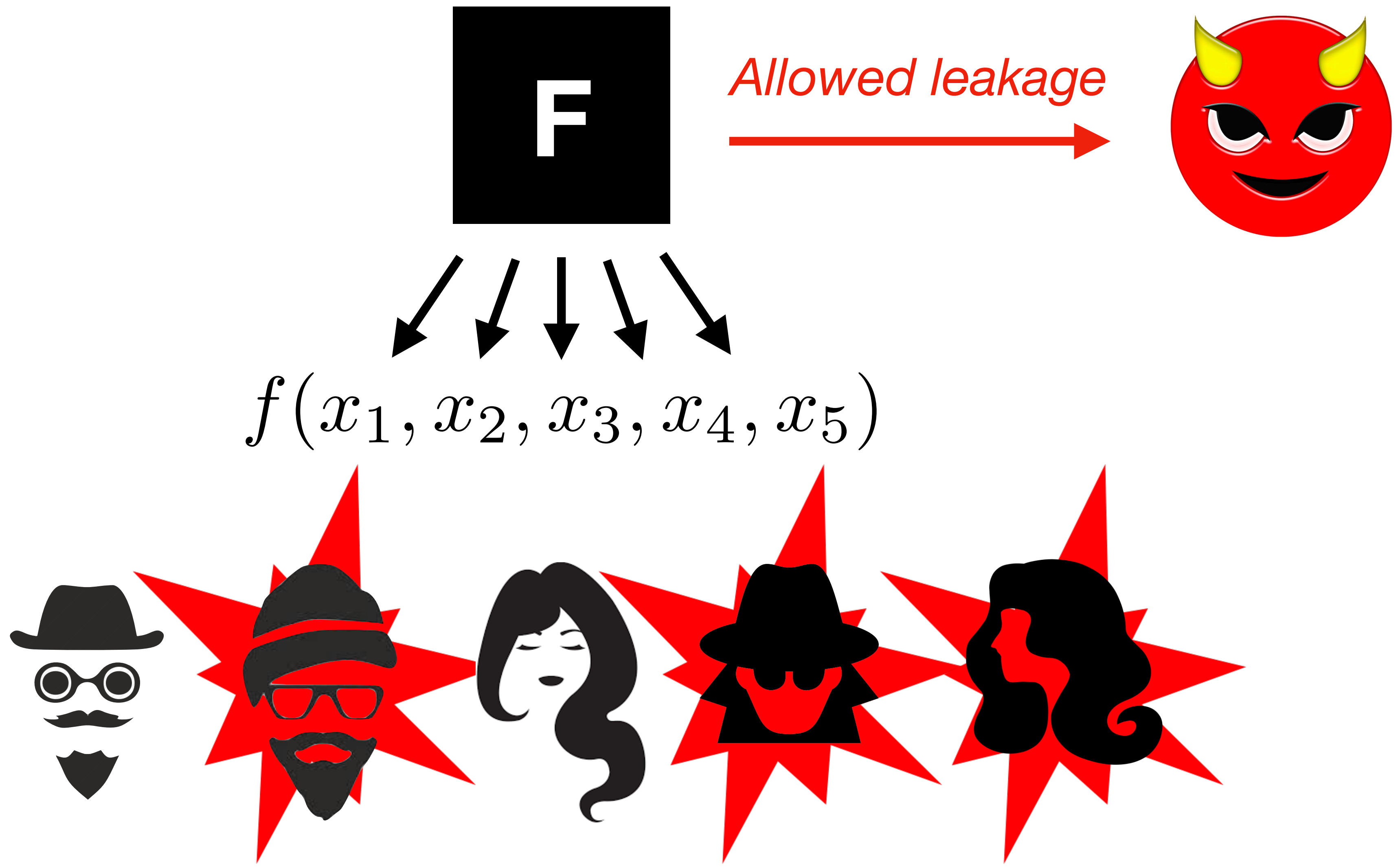
Real World



Real Behavior

- Players interact through [network model]
- Some players are corrupted in [corruption model]

Ideal World



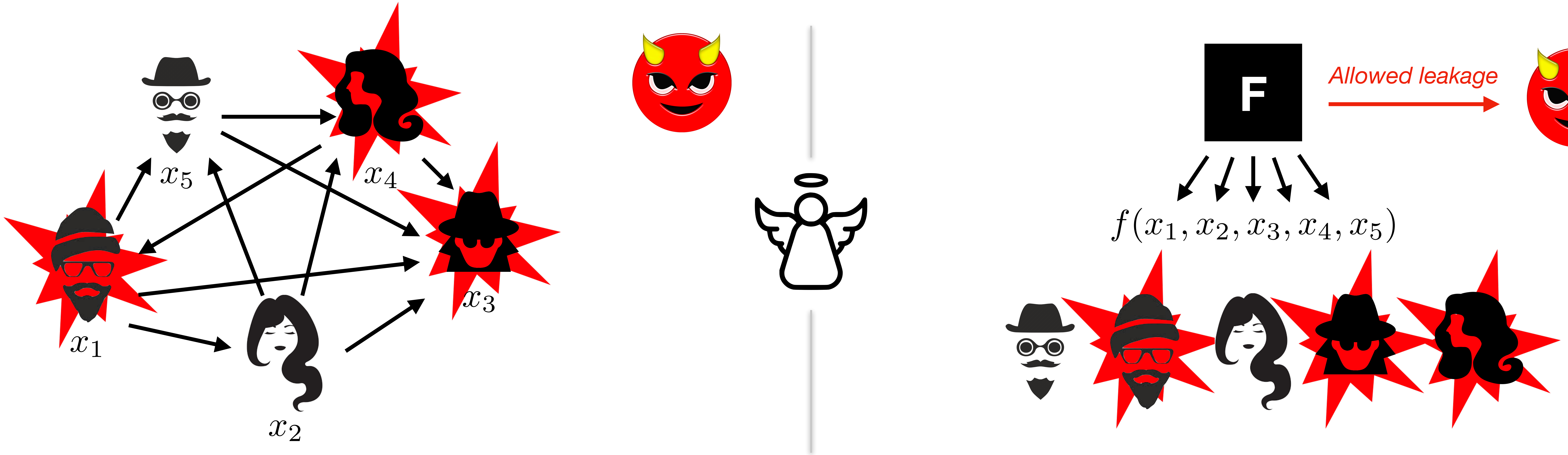
Ideal Behavior

- All parties send their input to a trusted party **F** through perfectly secure authenticated channels
- **F** computes the output and reveals the result
- The adversary only gets some allowed leakage (+ input/output of corrupted parties)

The Model - Defining Security

Real World

Ideal World



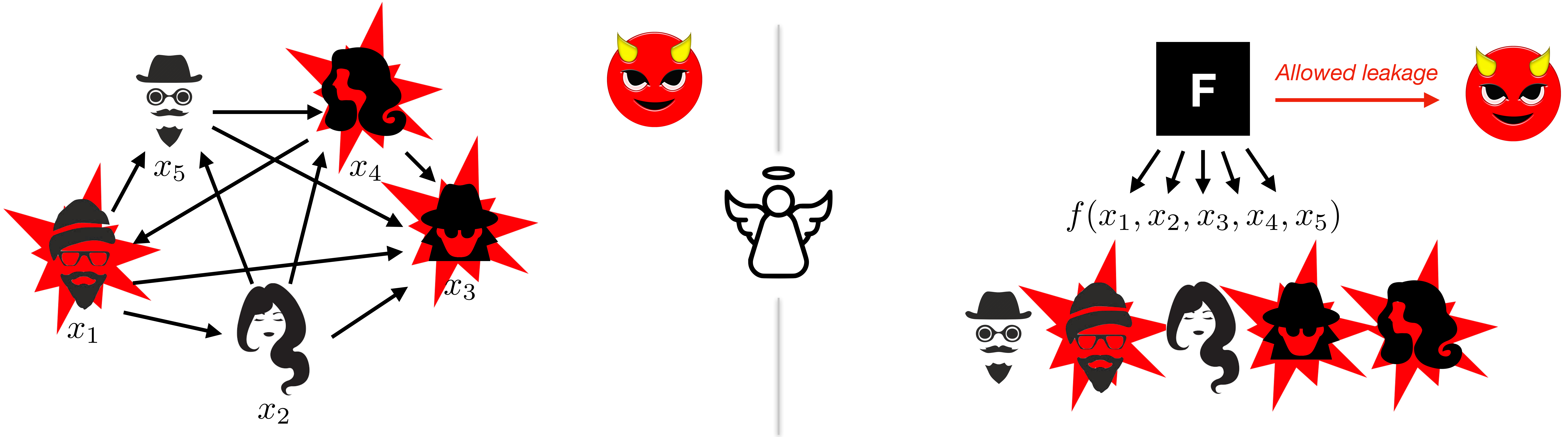
Simulation

Core idea: we construct a *simulator* which fools the adversary into believing he is playing the real world protocol, while making him effectively play the ideal world protocol. Then, we prove that *no adversary* can distinguish the simulated protocol from the real protocol.

The Model - Defining Security

Real World

Ideal World



Simulation

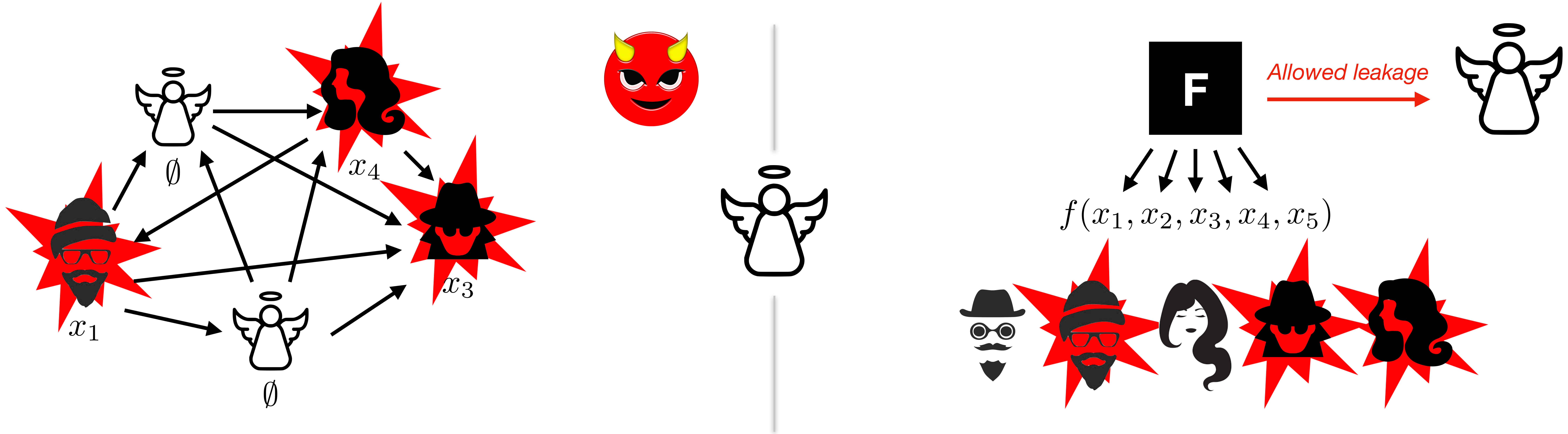
Core idea: we construct a *simulator* which fools the adversary into believing he is playing the real world protocol, while making him effectively play the ideal world protocol. Then, we prove that *no adversary* can distinguish the simulated protocol from the real protocol.

Take the time to convince yourself that this guarantees that the real protocol is as secure as the ideal functionality. If you cannot convince yourself, please ask.

The Model - Defining Security

Real World

Ideal World



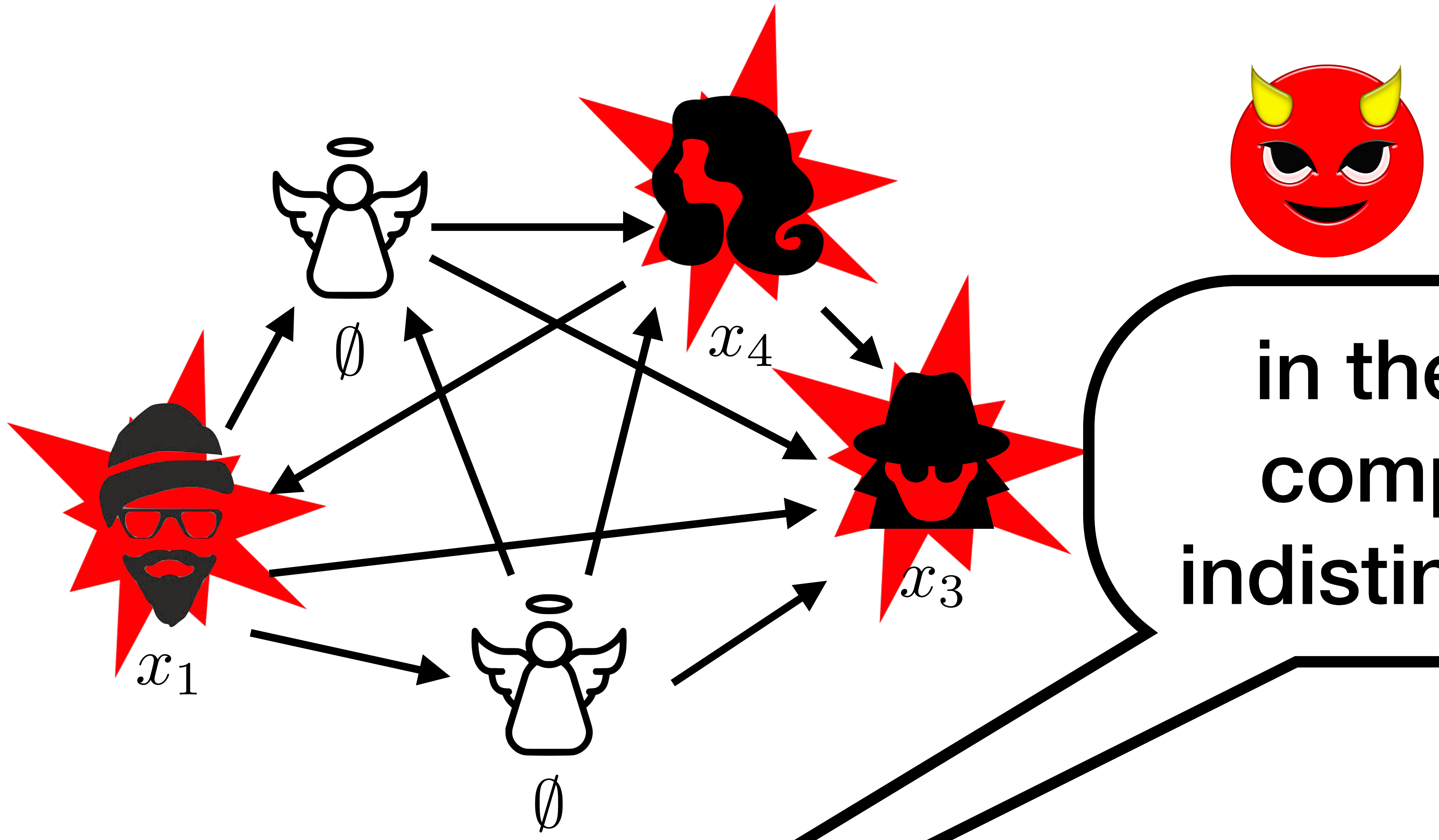
Simulation

Core idea: we construct a *simulator* which fools the adversary into believing he is playing the real world protocol, while making him effectively play the ideal world protocol. Then, we prove that *no adversary* can distinguish the simulated protocol from the real protocol.

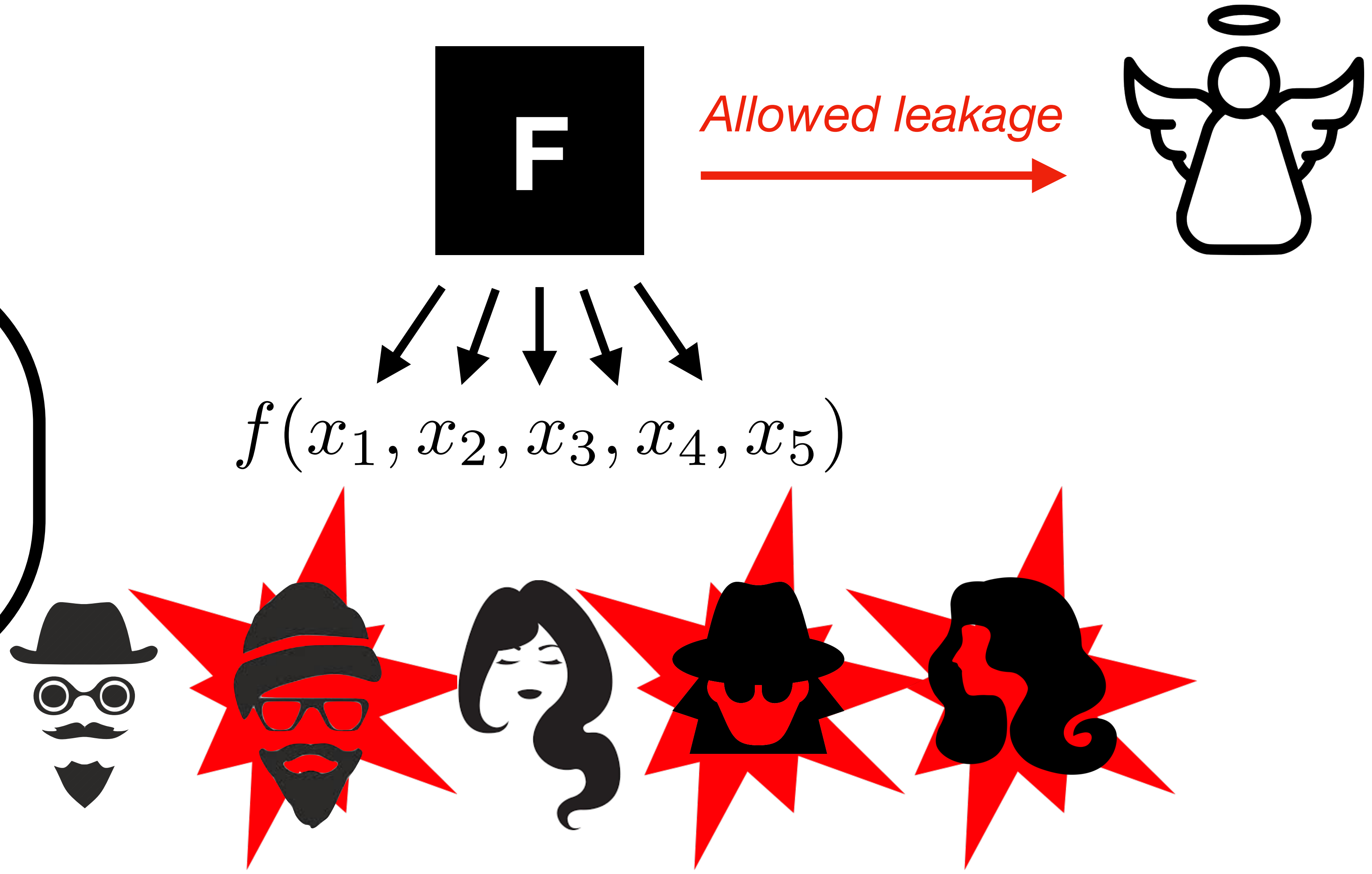
emulates the *honest parties* in the real world, and the *adversary* in the ideal world.

The Model - Defining Security

Real World



Ideal World



in the sense of computational indistinguishability

Simulation

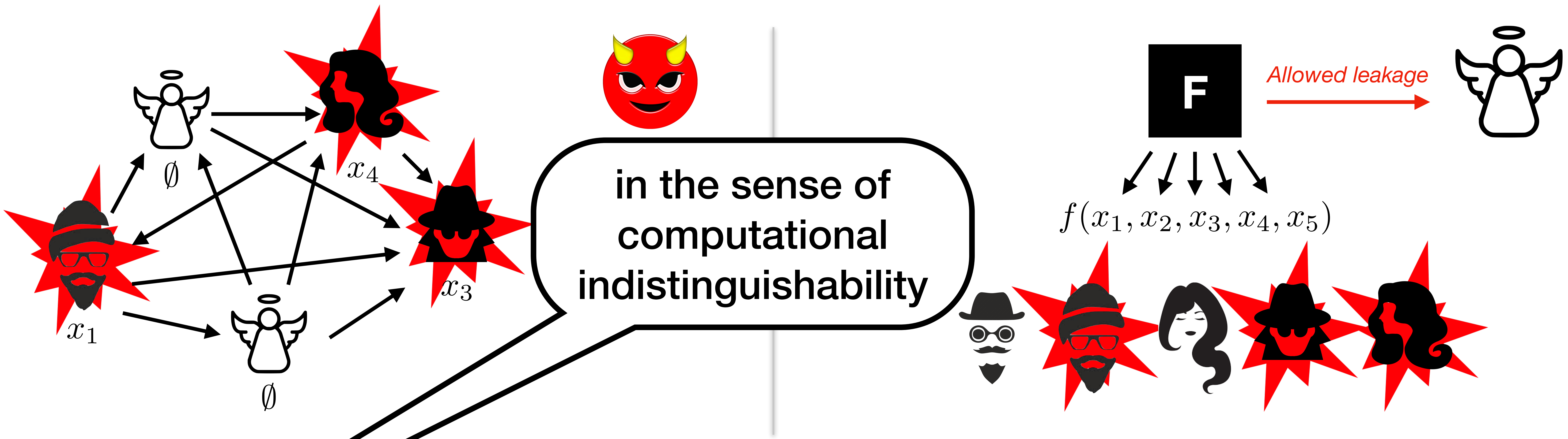
Core idea: we construct a *simulator* which fools the adversary into believing he is playing the real world protocol, while making him effectively play the ideal world protocol. Then, we prove that *no adversary* can distinguish the simulated protocol from the real protocol.

👼 emulates the *honest parties* in the real world, and the *adversary* in the ideal world.

The Model - Defining Security

Real World

Ideal World



in the sense of computational indistinguishability

Simulation

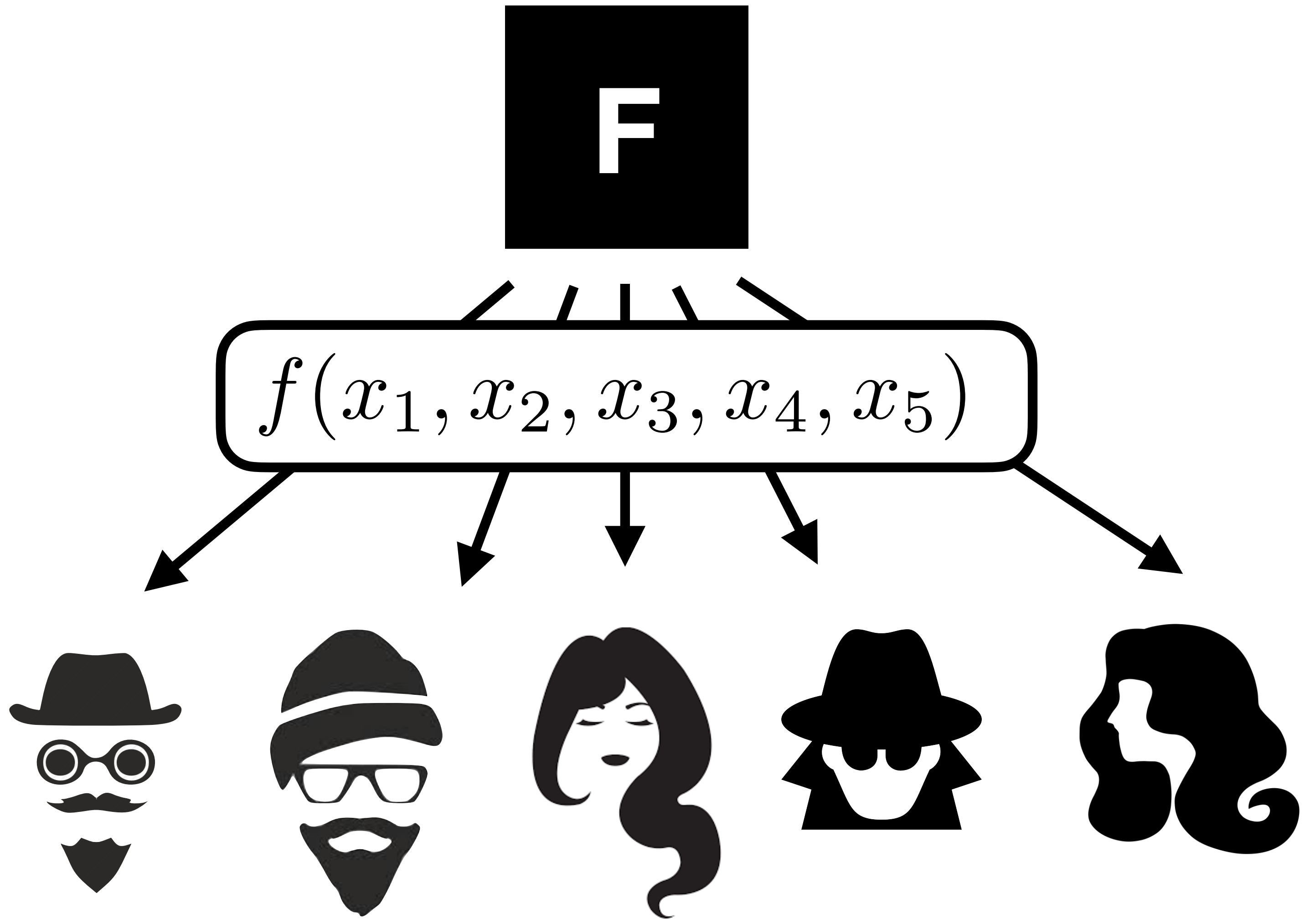
Core idea: we construct a *simulator* which fools the adversary into believing he is playing the real world protocol, while making him effectively play the ideal world protocol. Then, we prove that *no adversary* can distinguish the simulated protocol from the real protocol.

The **distributions** of the adversary's **view** in the real world and in the simulated world are **computationally indistinguishable**

Exercise 2

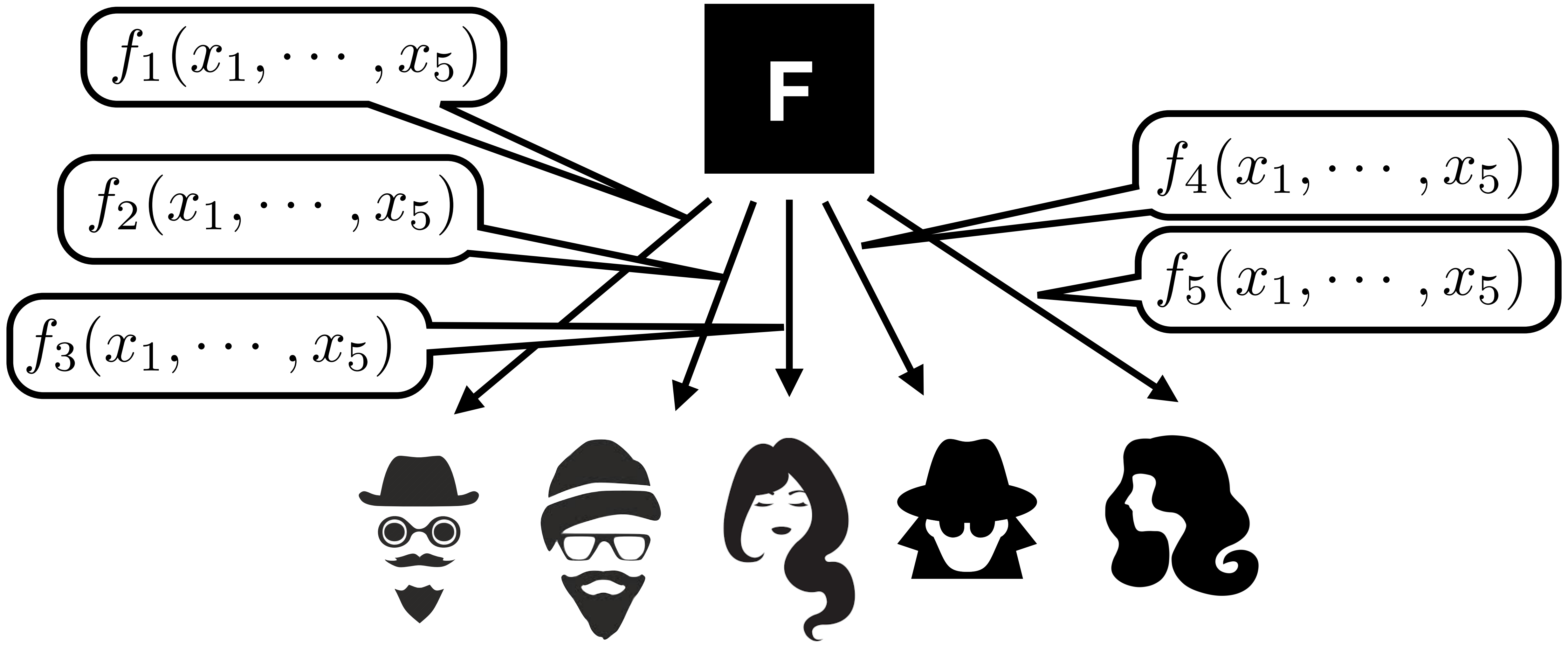
Same Output

Model: all parties receive the same output



Independent Outputs

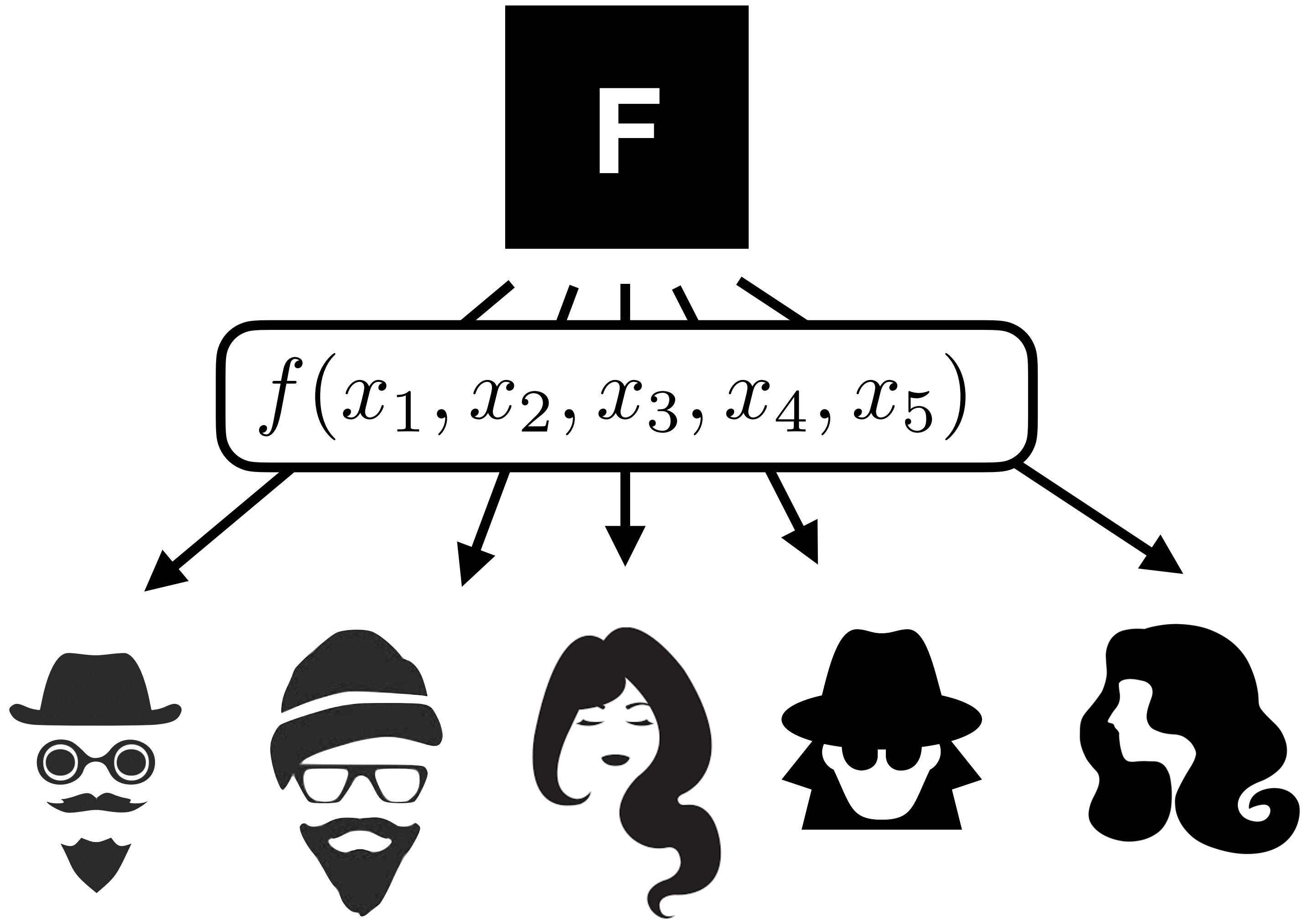
We can also consider a more general model, where each party gets a specific output (this also captures the case where not all parties should get the output).



Exercise 2

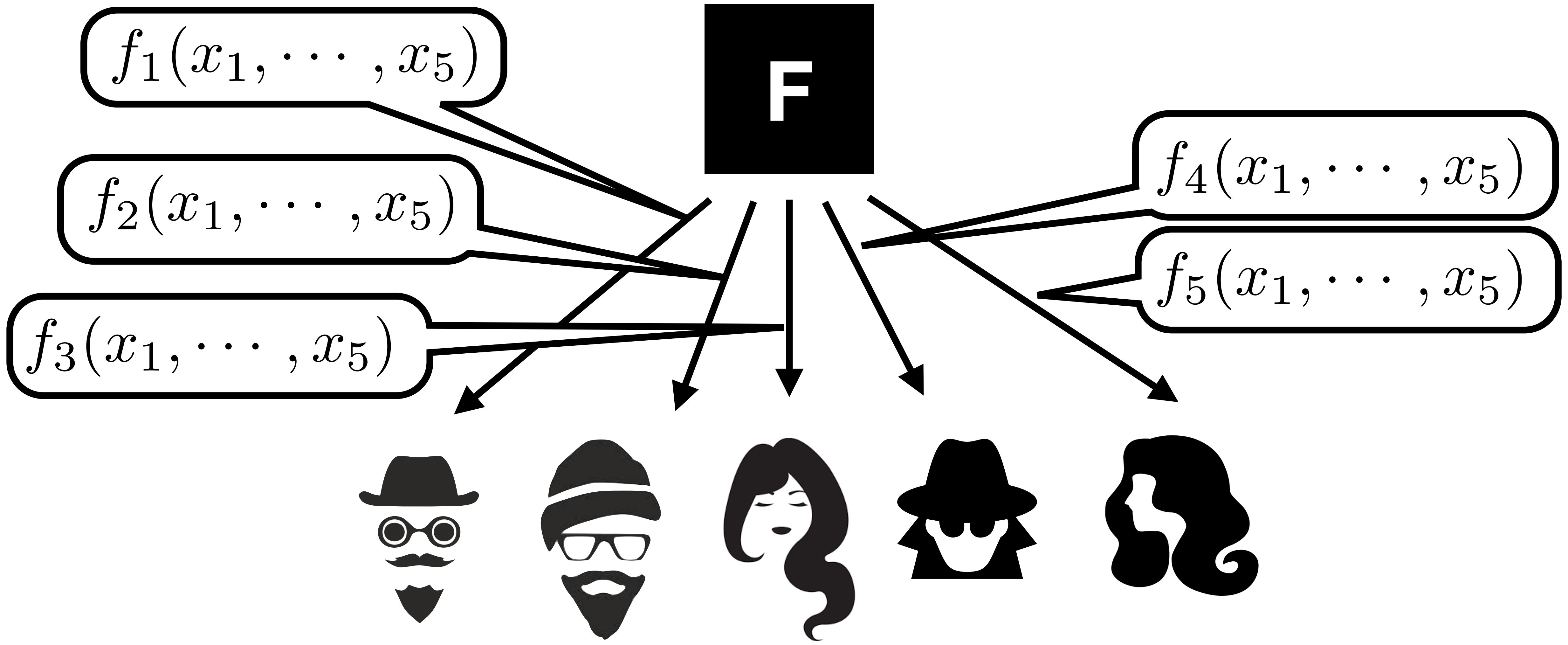
Same Output

Model: all parties receive the same output



Independent Outputs

We can also consider a more general model, where each party gets a specific output (this also captures the case where not all parties should get the output).



Q: show that a general solution for the same output scenario implies a general solution for the independent outputs scenario.

Exercise 2 - Solution

Suppose that for all functionality $f : X_1 \times X_2 \times X_3 \times X_4 \times X_5 \mapsto \{0,1\}^*$, there is a secure protocol where all parties get the same output. Let $(x_1, x_2, x_3, x_4, x_5)$ be the parties' inputs, and let $(f_1, f_2, f_3, f_4, f_5)$ be the functions computing the independent outputs each party wants.

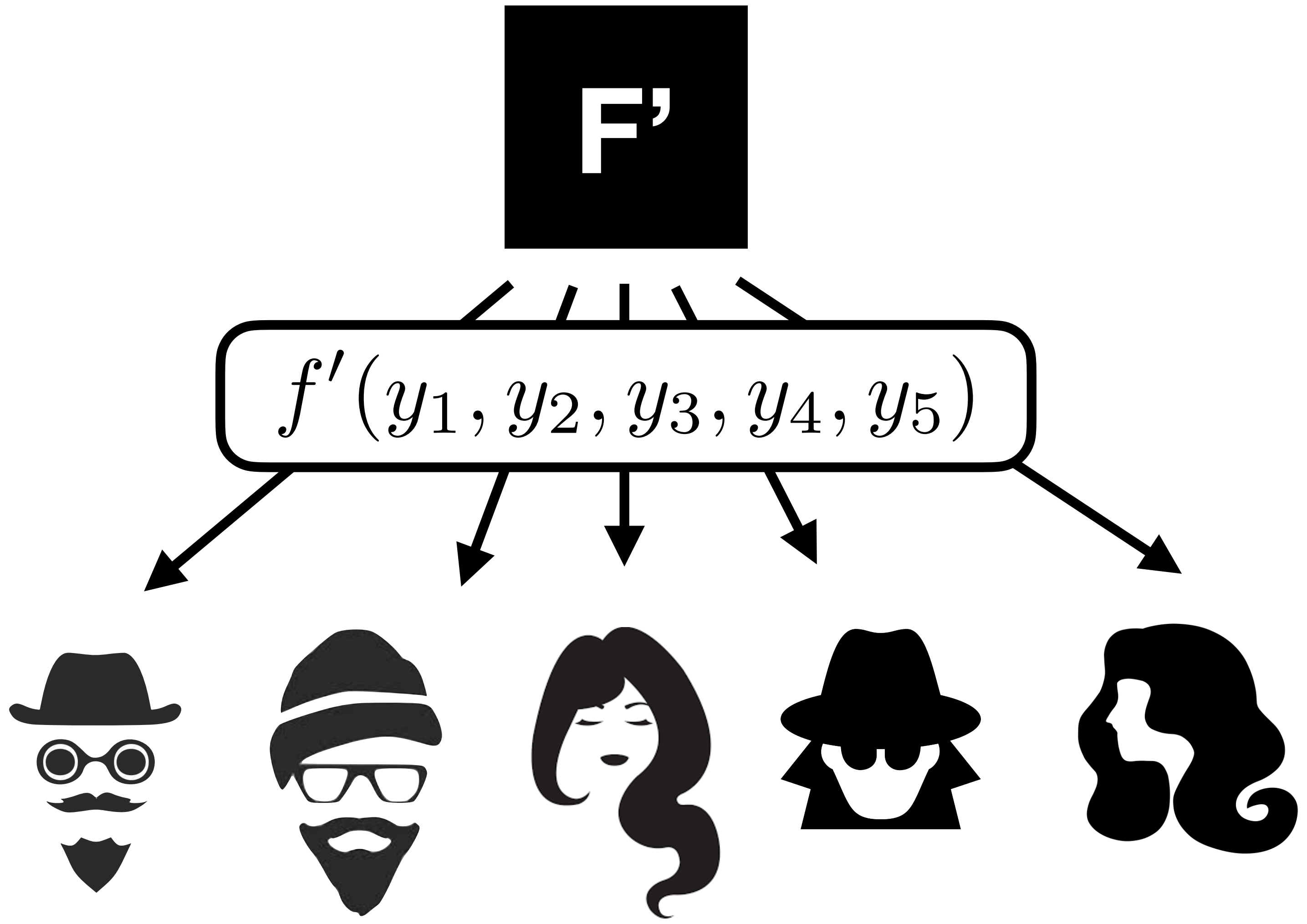
Exercise 2 - Solution

Suppose that for all functionality $f : X_1 \times X_2 \times X_3 \times X_4 \times X_5 \mapsto \{0,1\}^*$, there is a secure protocol where all parties get the same output. Let $(x_1, x_2, x_3, x_4, x_5)$ be the parties' inputs, and let $(f_1, f_2, f_3, f_4, f_5)$ be the functions computing the independent outputs each party wants.

Define the following *single output* 5-party functionality **F'** which securely computes:

$$\boxed{f' : ((x_1, r_1), \cdots, (x_5, r_5)) \mapsto (r_1 \oplus f_1(x_1, \cdots, x_5), \cdots, r_5 \oplus f_5(x_1, \cdots, x_5))}$$

Reduction: each party with input x_i picks a uniformly random r_i . All parties emulate **F'**.
Security: follows from the fact that r_i perfectly masks $f_i(x_1, \cdots, x_5)$.



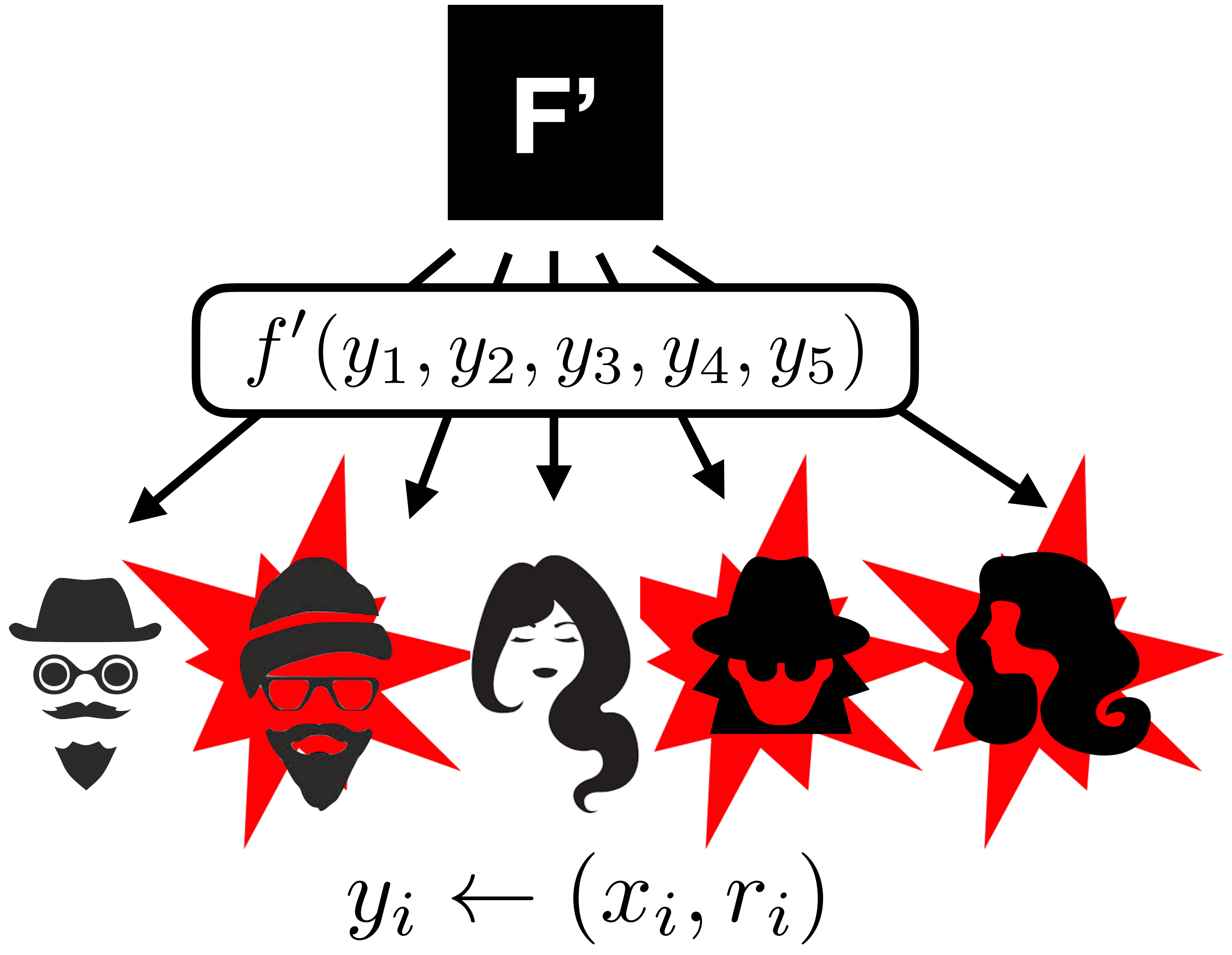
F'

$f'(y_1, y_2, y_3, y_4, y_5)$

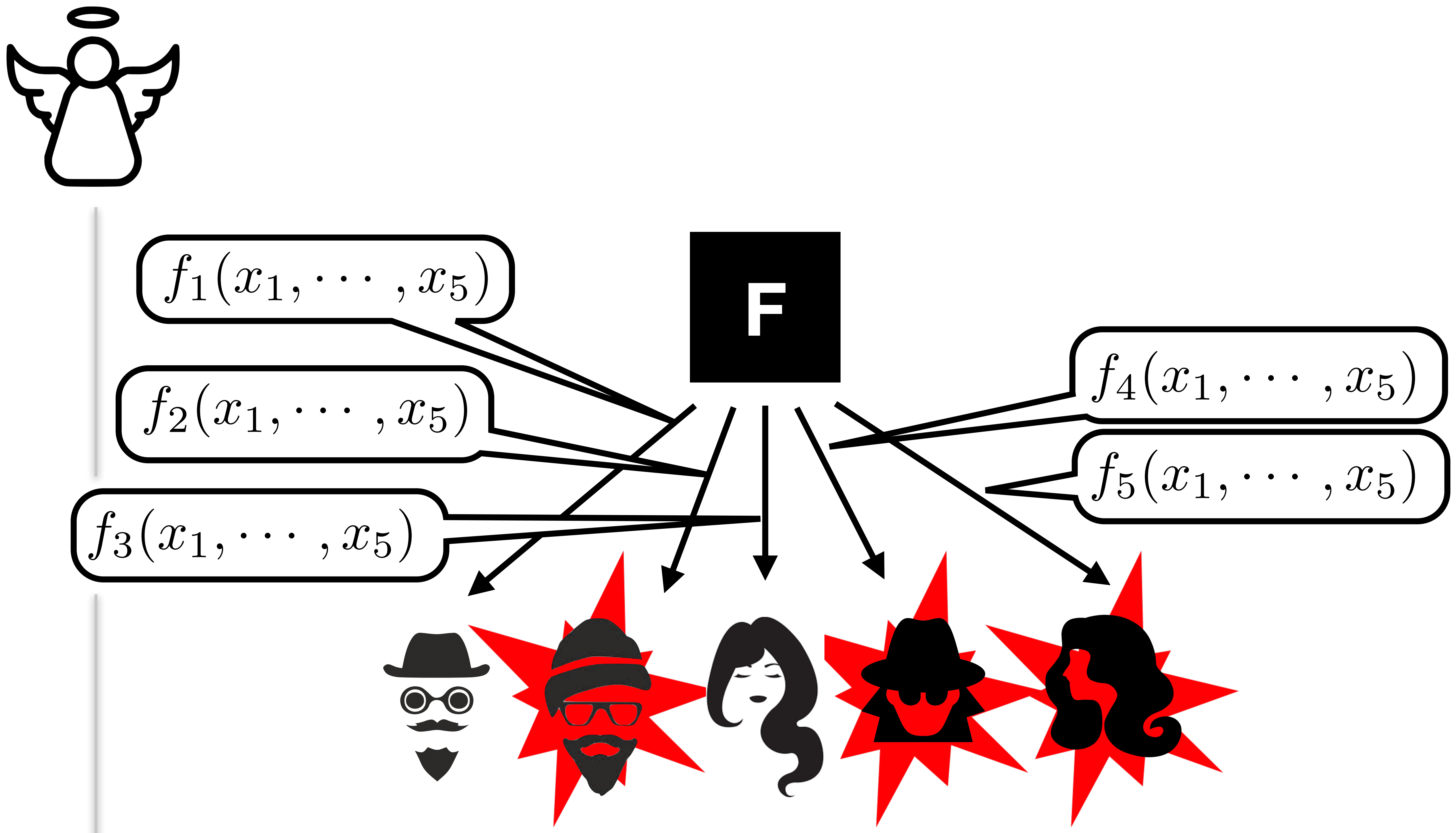
$y_i \leftarrow (x_i, r_i)$

Exercise 2 - Solution

Real World

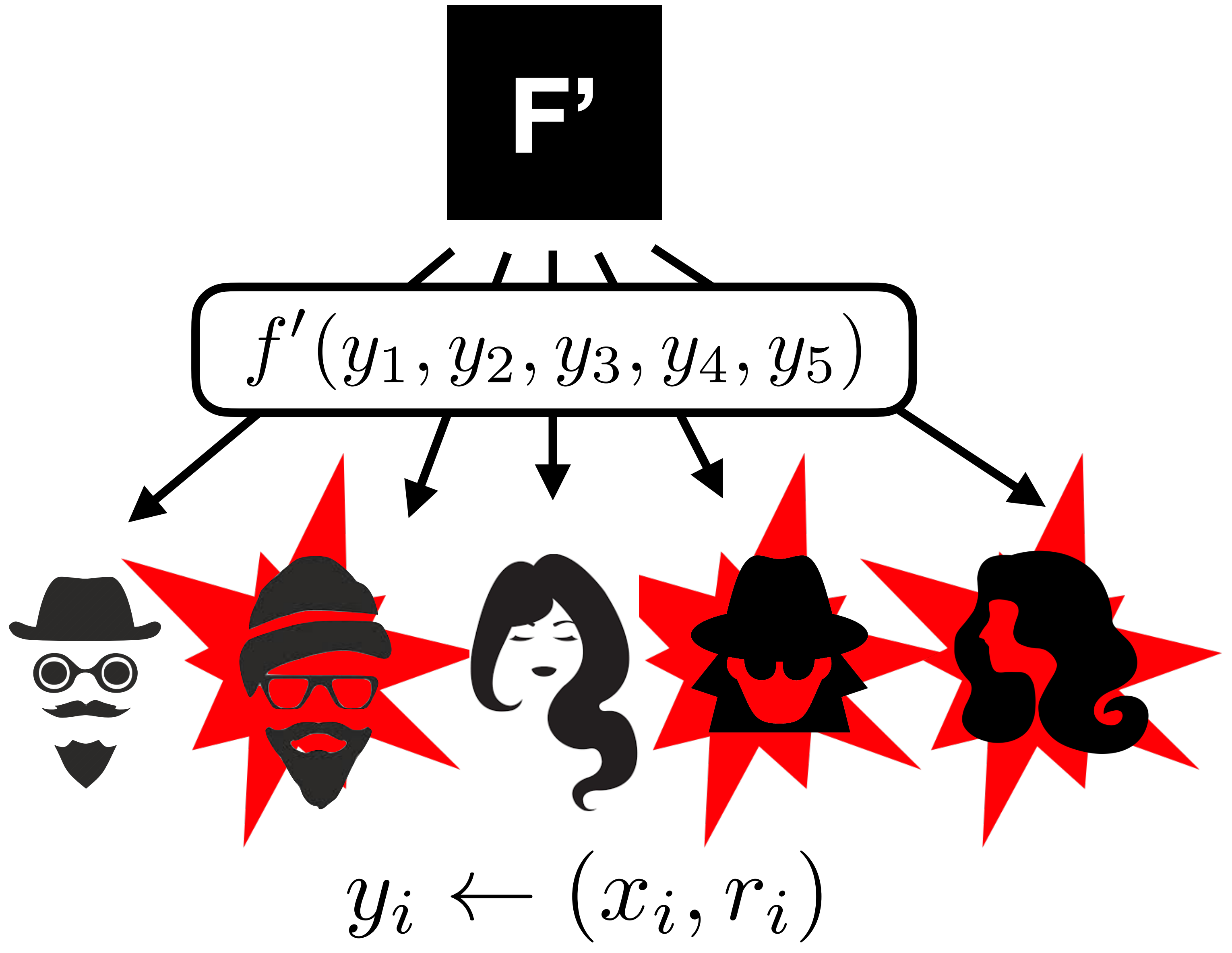


Ideal World



Exercise 2 - Solution

Real World

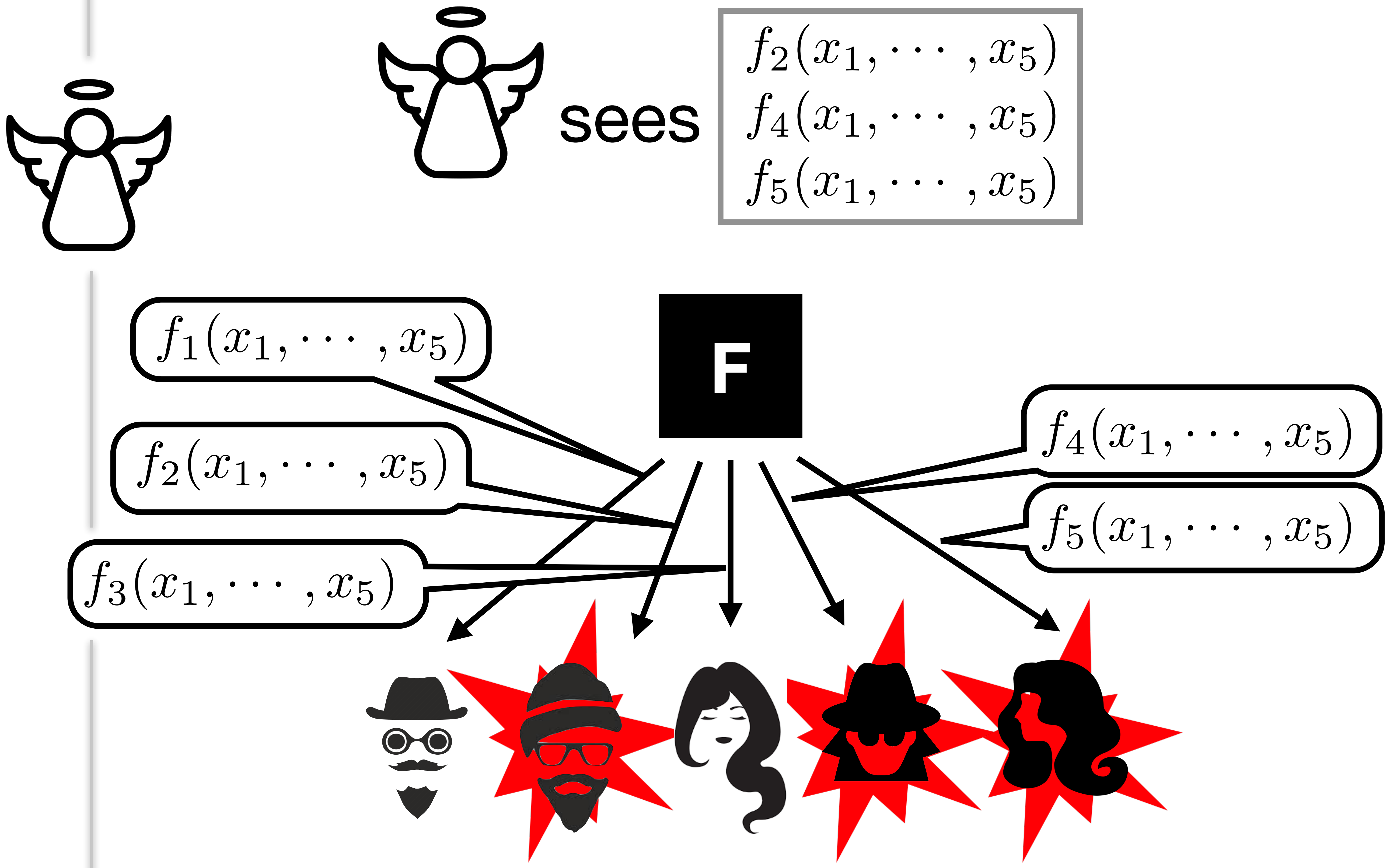


F'

$f'(y_1, y_2, y_3, y_4, y_5)$

$y_i \leftarrow (x_i, r_i)$

Ideal World



sees

$f_2(x_1, \cdots, x_5)$
$f_4(x_1, \cdots, x_5)$
$f_5(x_1, \cdots, x_5)$

F

$f_1(x_1, \cdots, x_5)$

$f_2(x_1, \cdots, x_5)$

$f_3(x_1, \cdots, x_5)$

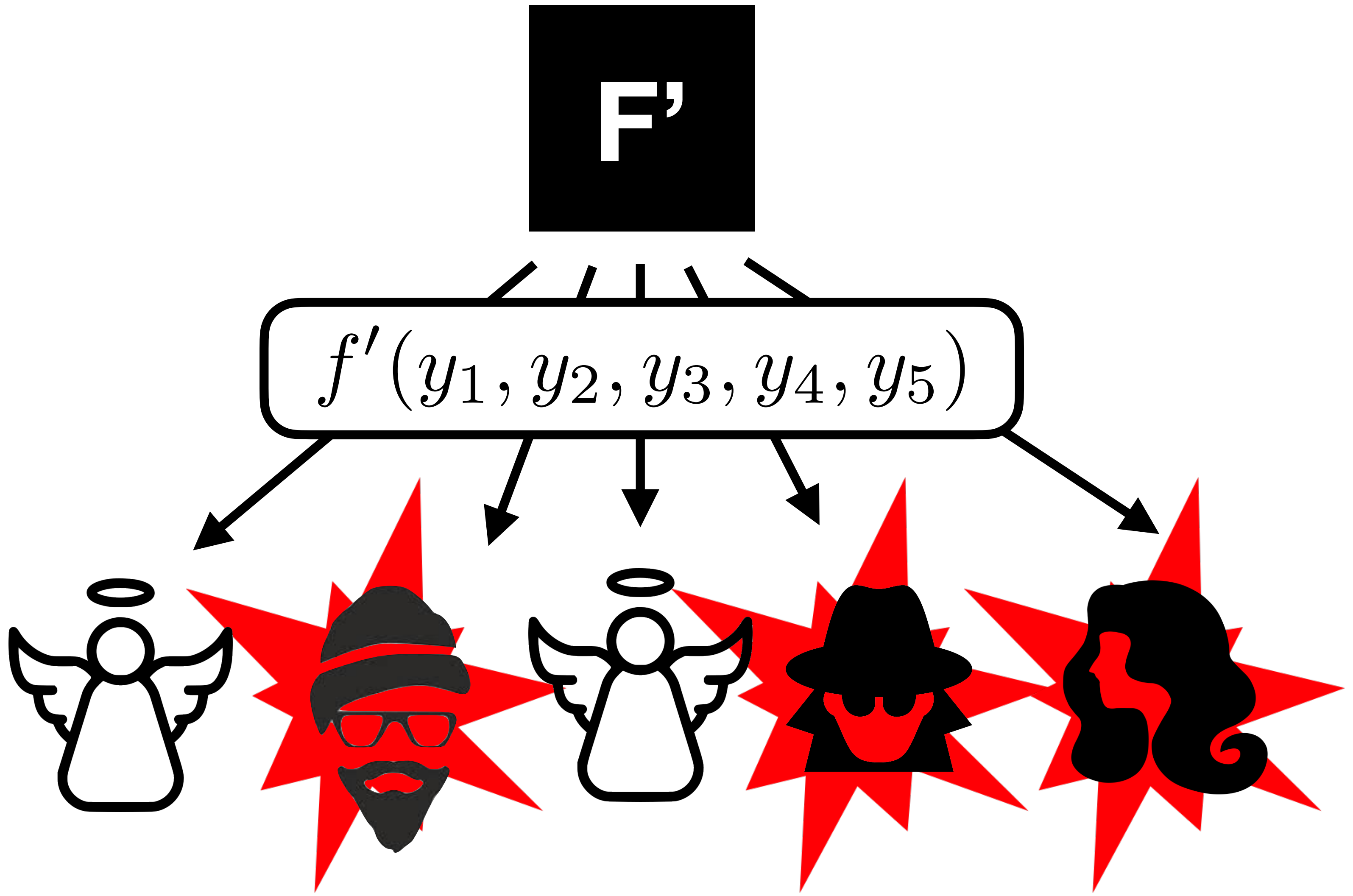
$f_4(x_1, \cdots, x_5)$

$f_5(x_1, \cdots, x_5)$

Exercise 2 - Solution

Real World

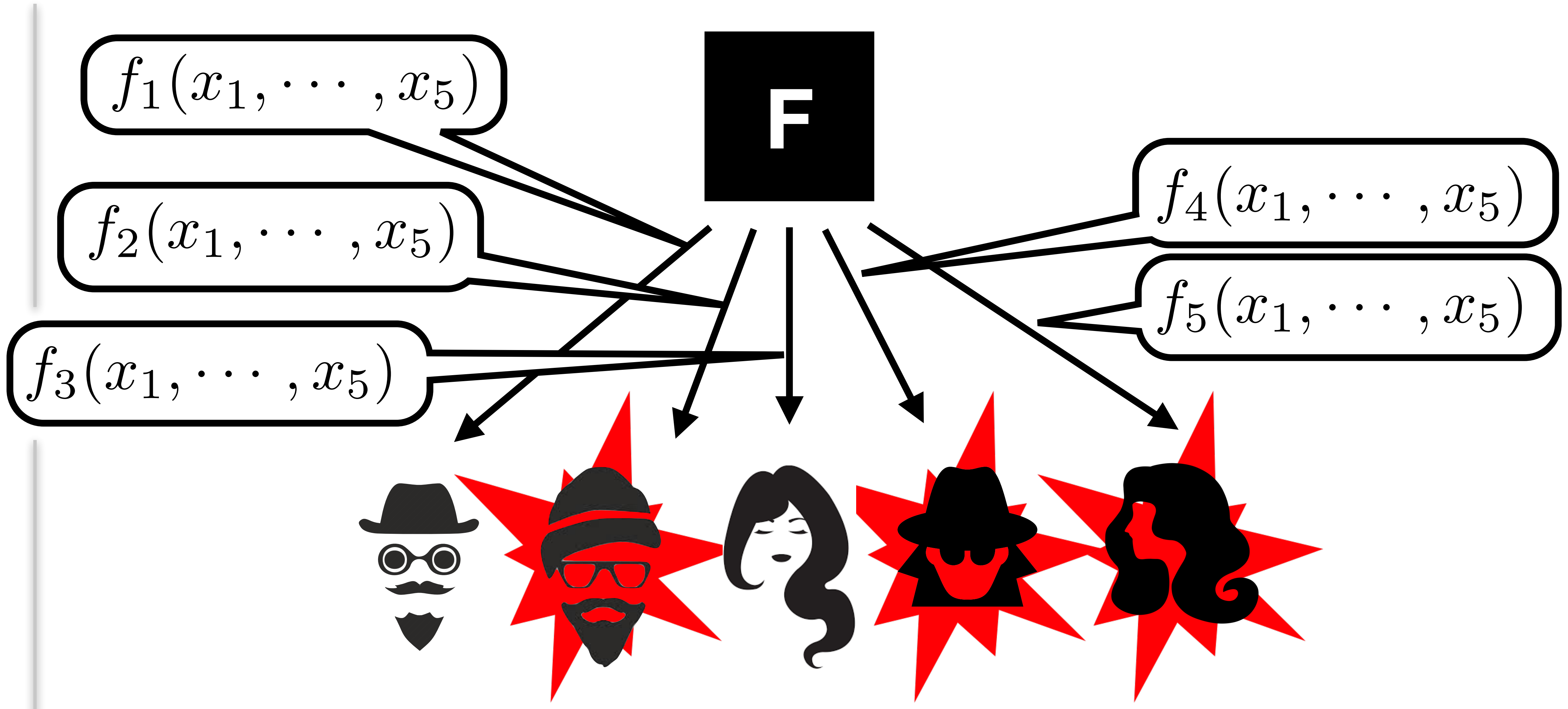
Nothing to emulate? The adversary sees nothing of what the emulated parties send…



$y_i \leftarrow (x_i, r_i)$

Ideal World

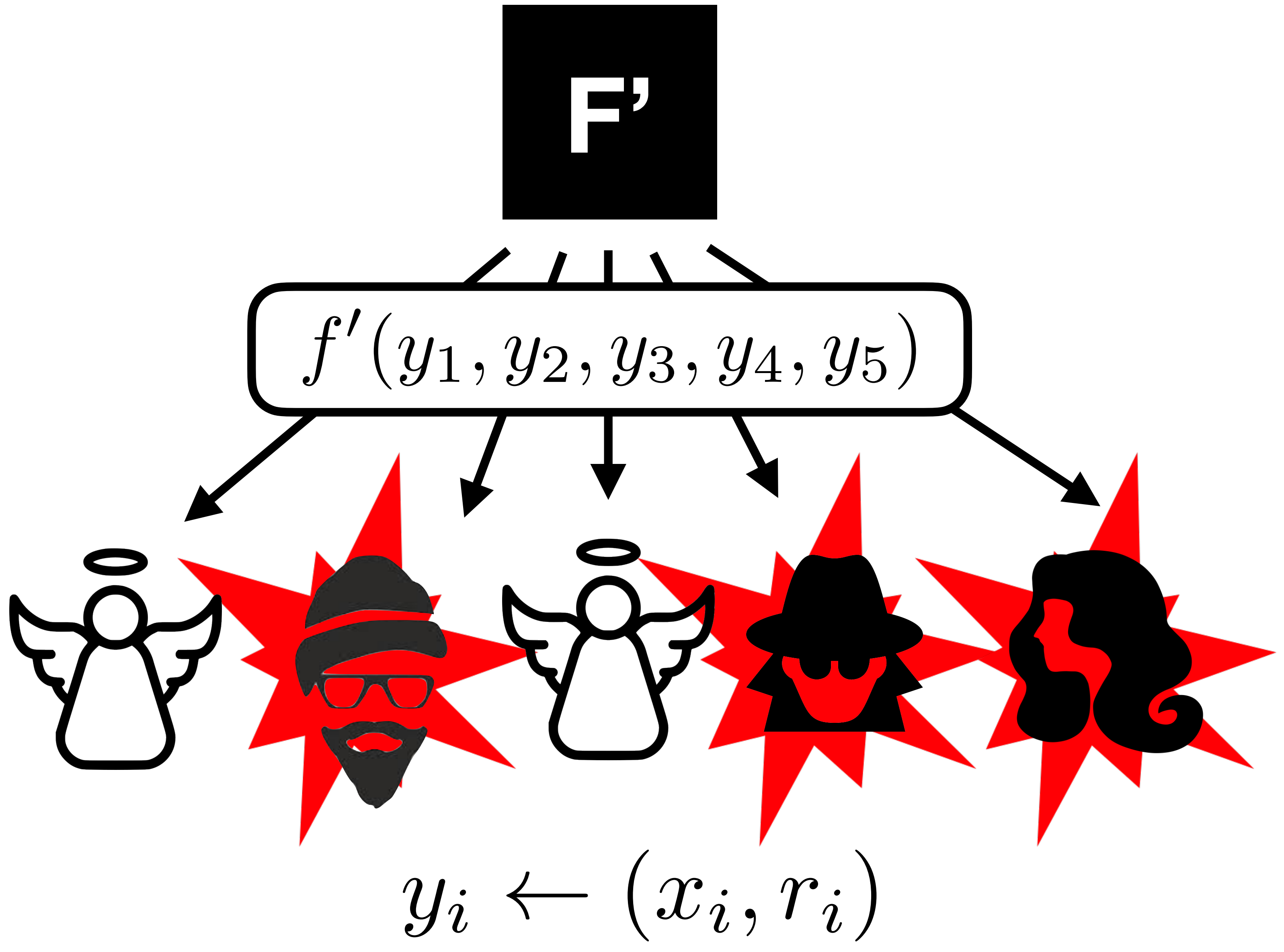
sees
$f_2(x_1, \cdots, x_5)$
$f_4(x_1, \cdots, x_5)$
$f_5(x_1, \cdots, x_5)$



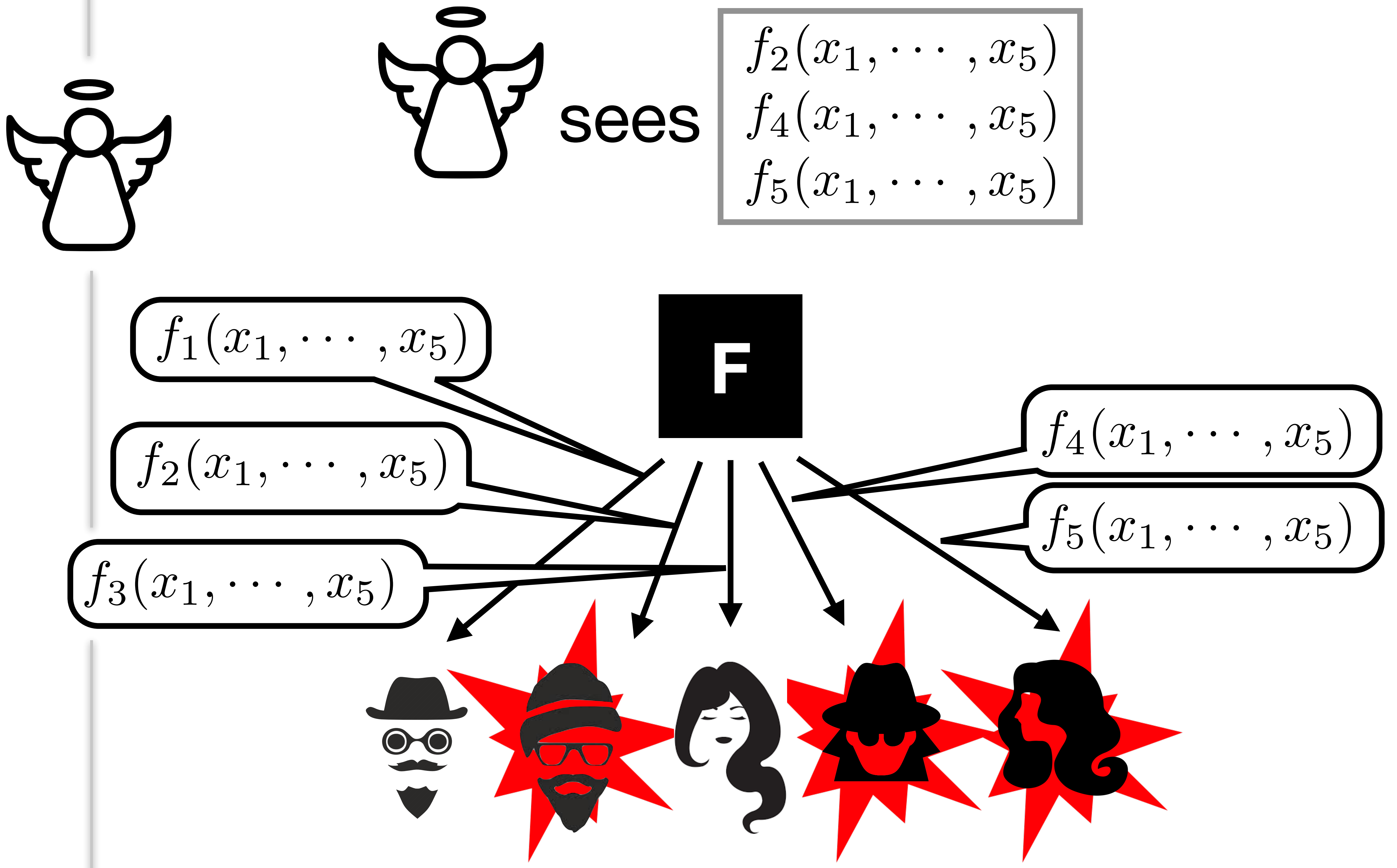
Exercise 2 - Solution

Real World

Different scenario: there is also a functionality in the real world! We call this « hybrid world ».



Ideal World

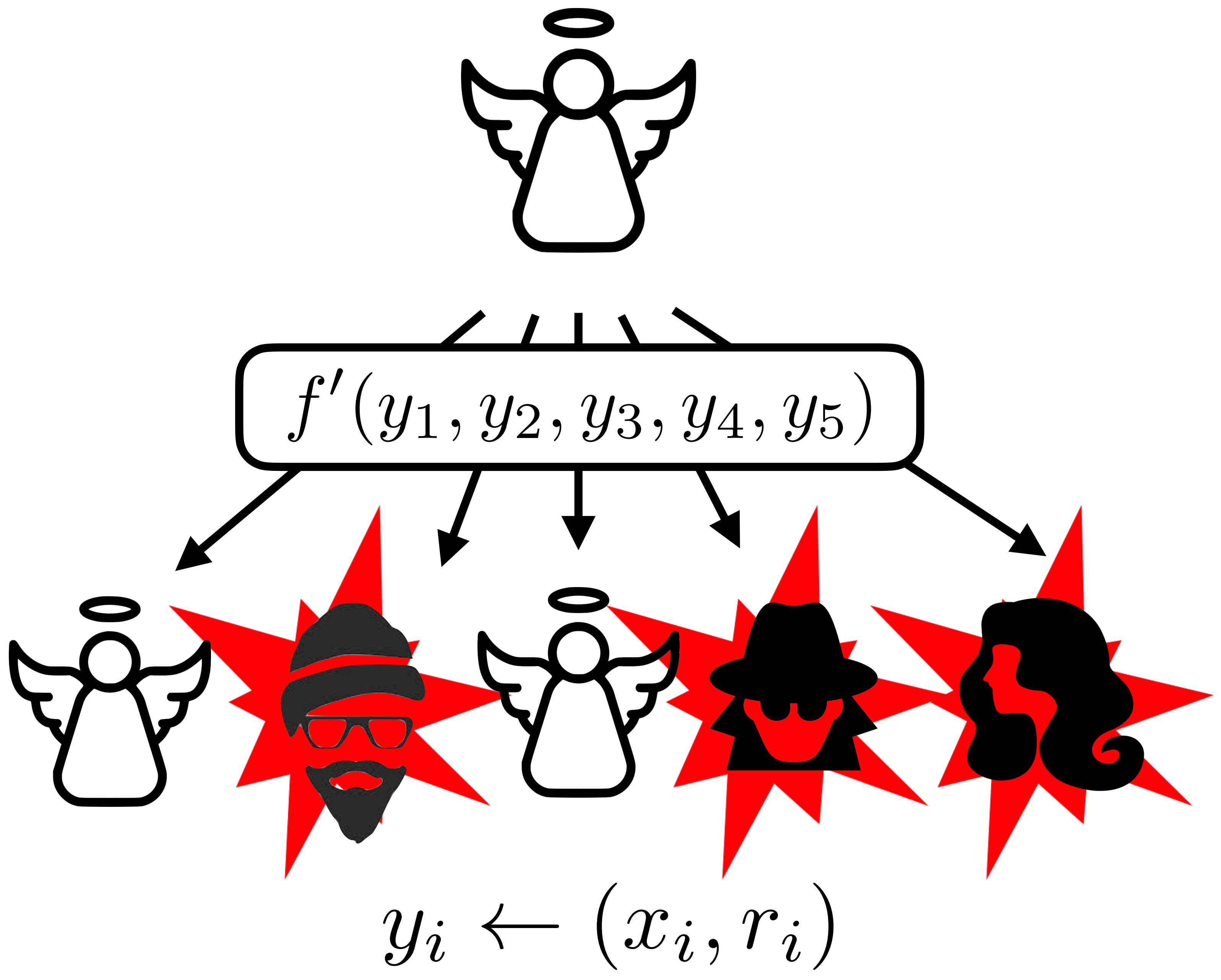


Exercise 2 - Solution

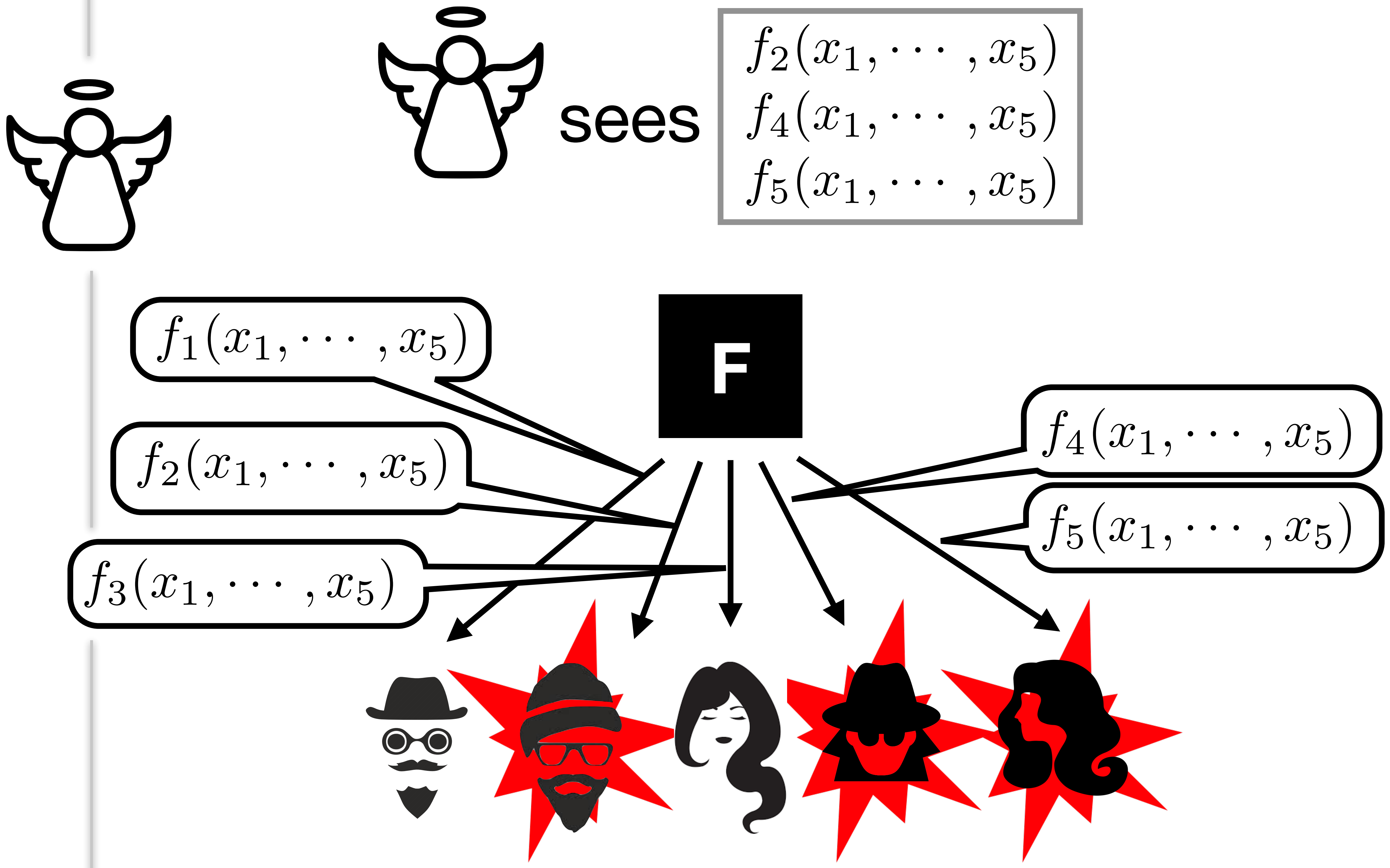
Real World

Different scenario: there is also a functionality in the real world! We call this « hybrid world ».

-> The simulator emulates **F'**.



Ideal World

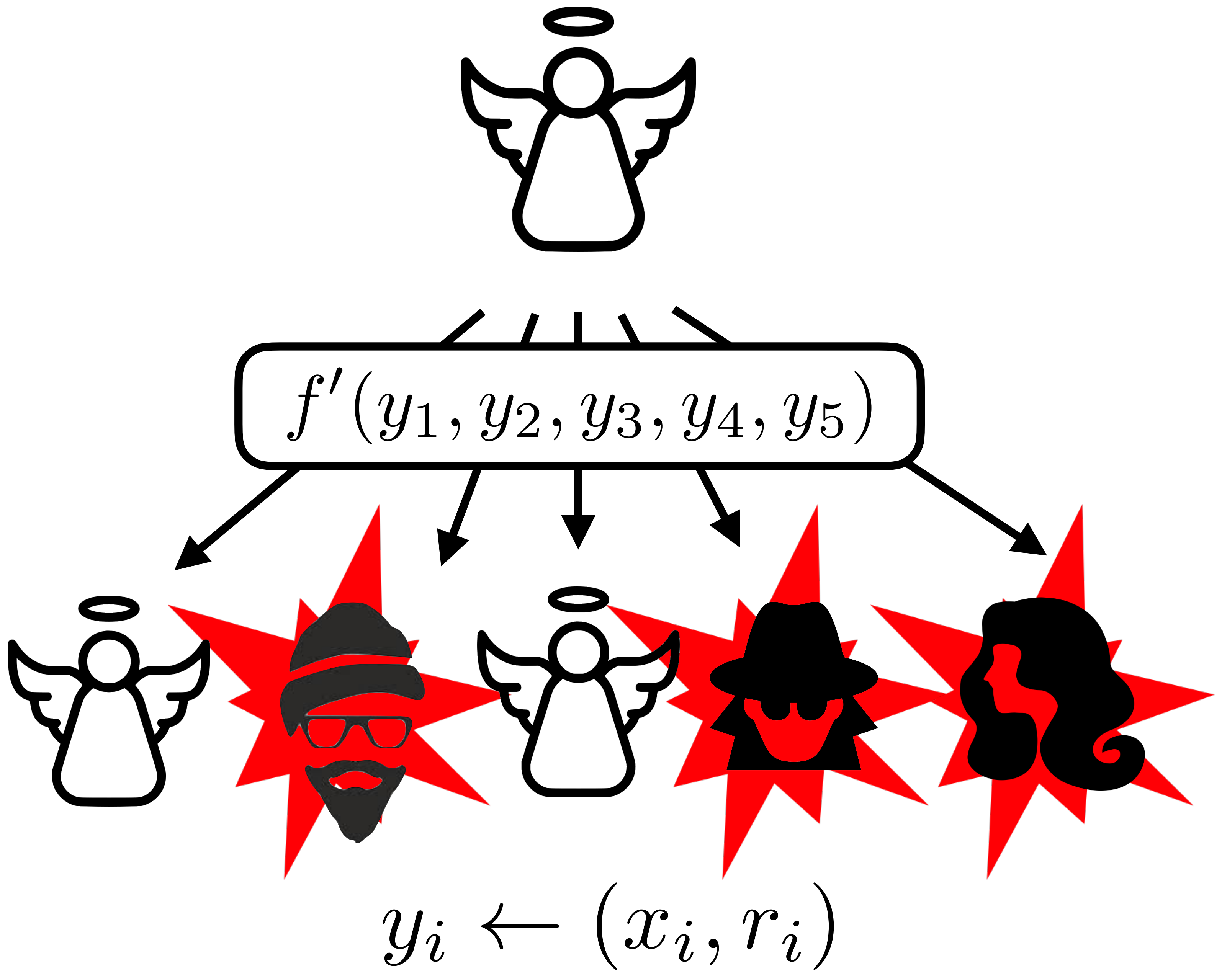


Exercise 2 - Solution

Real World

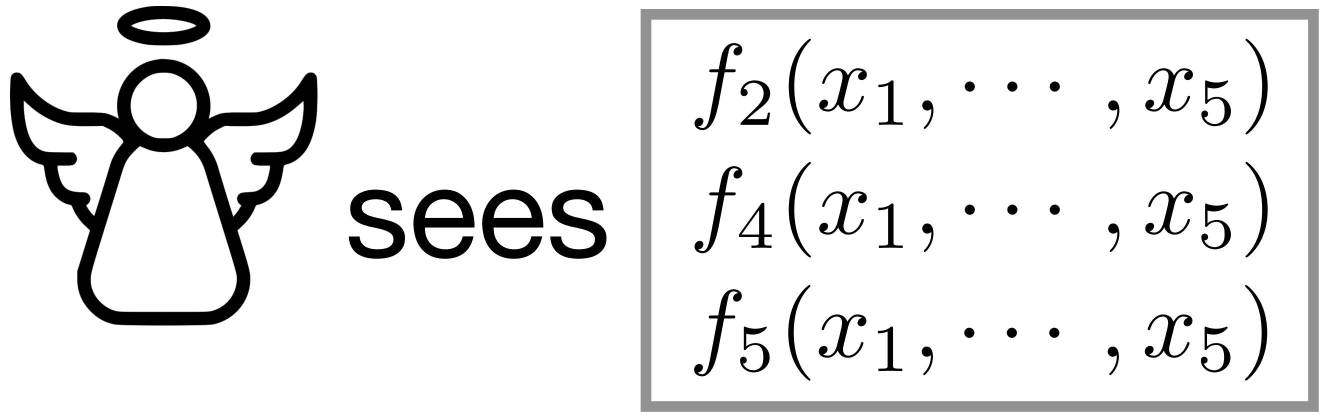
Different scenario: there is also a functionality in the real world! We call this « hybrid world ».

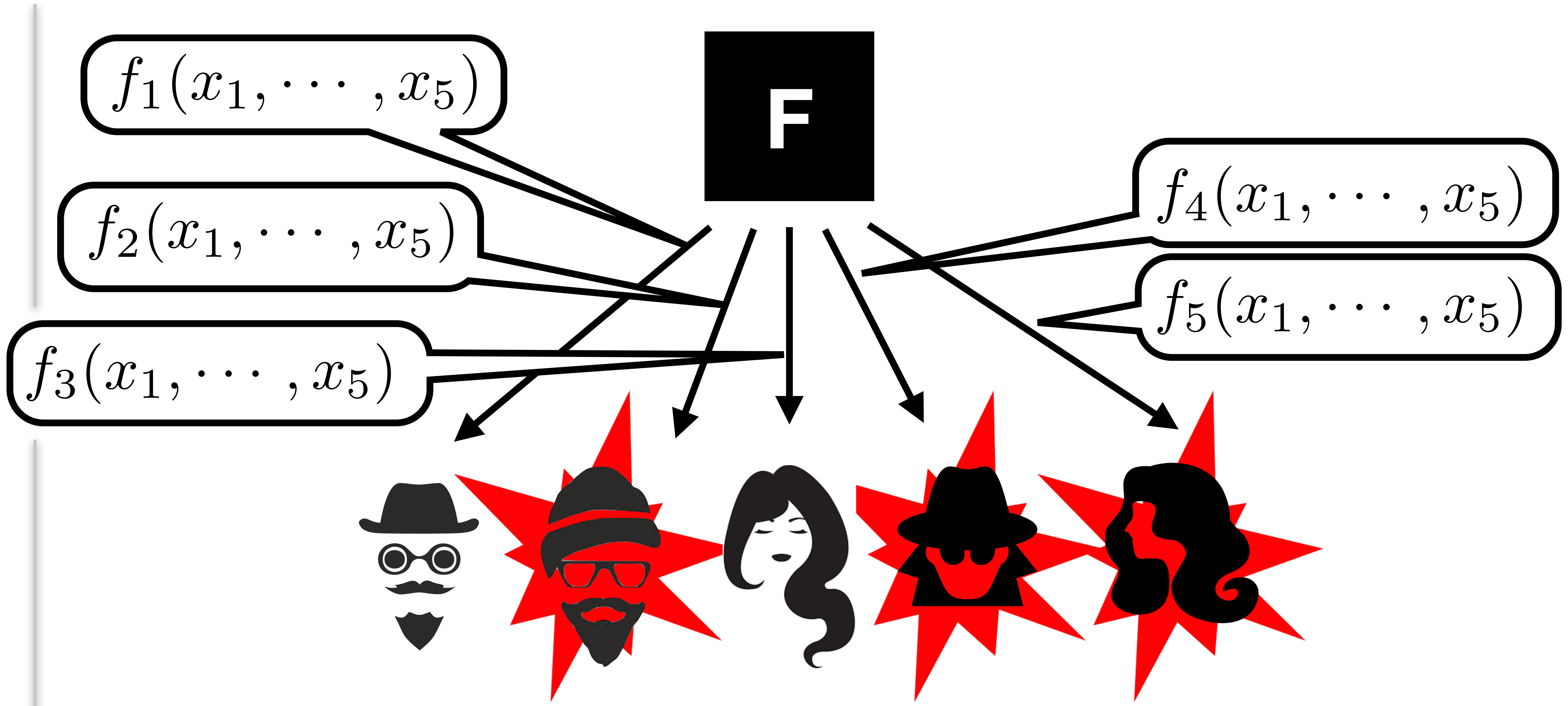
-> The simulator emulates **F'**.



Output: (random, $f_2(x_1, \cdots, x_5)$, random, …)

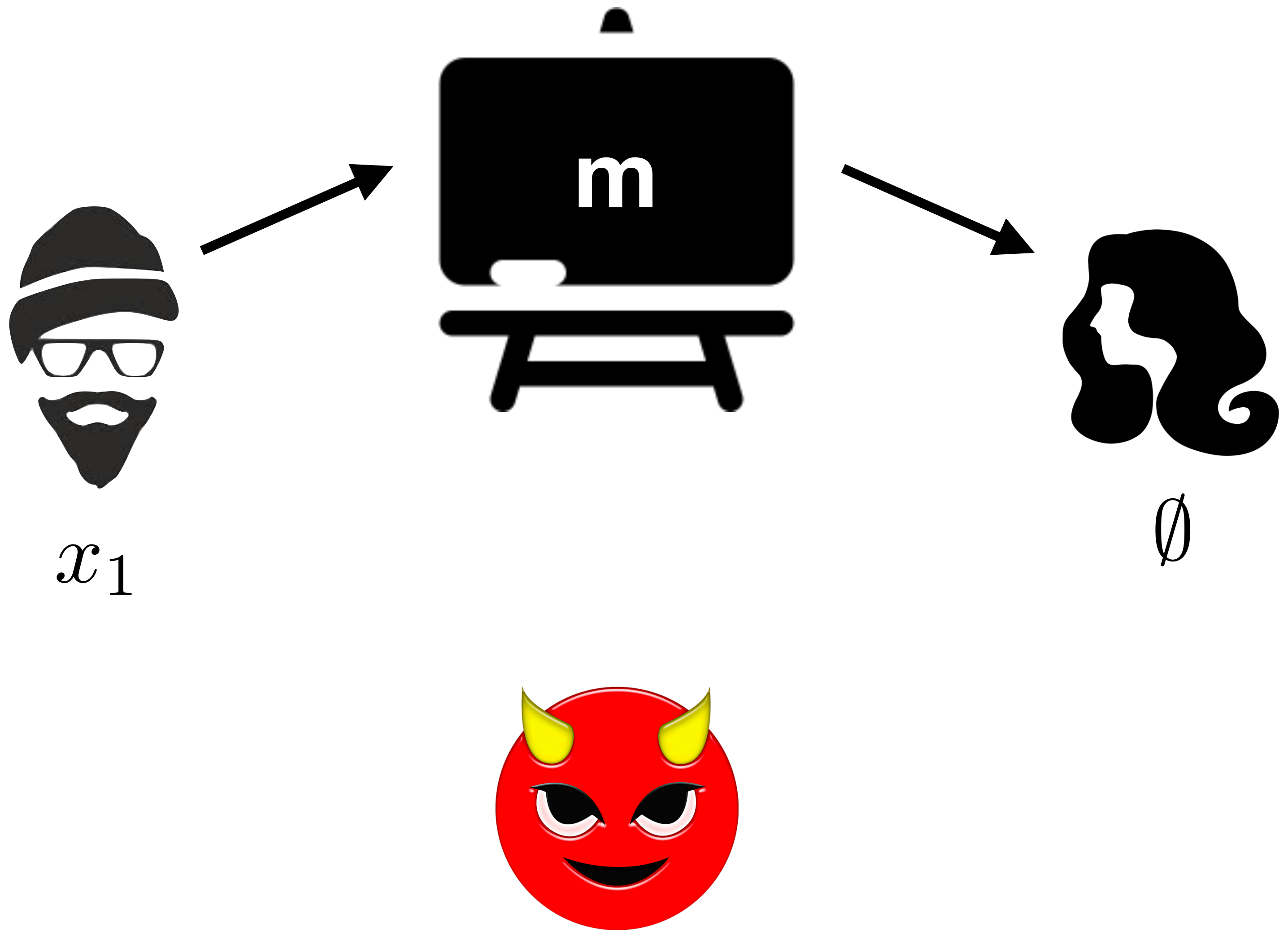
Ideal World



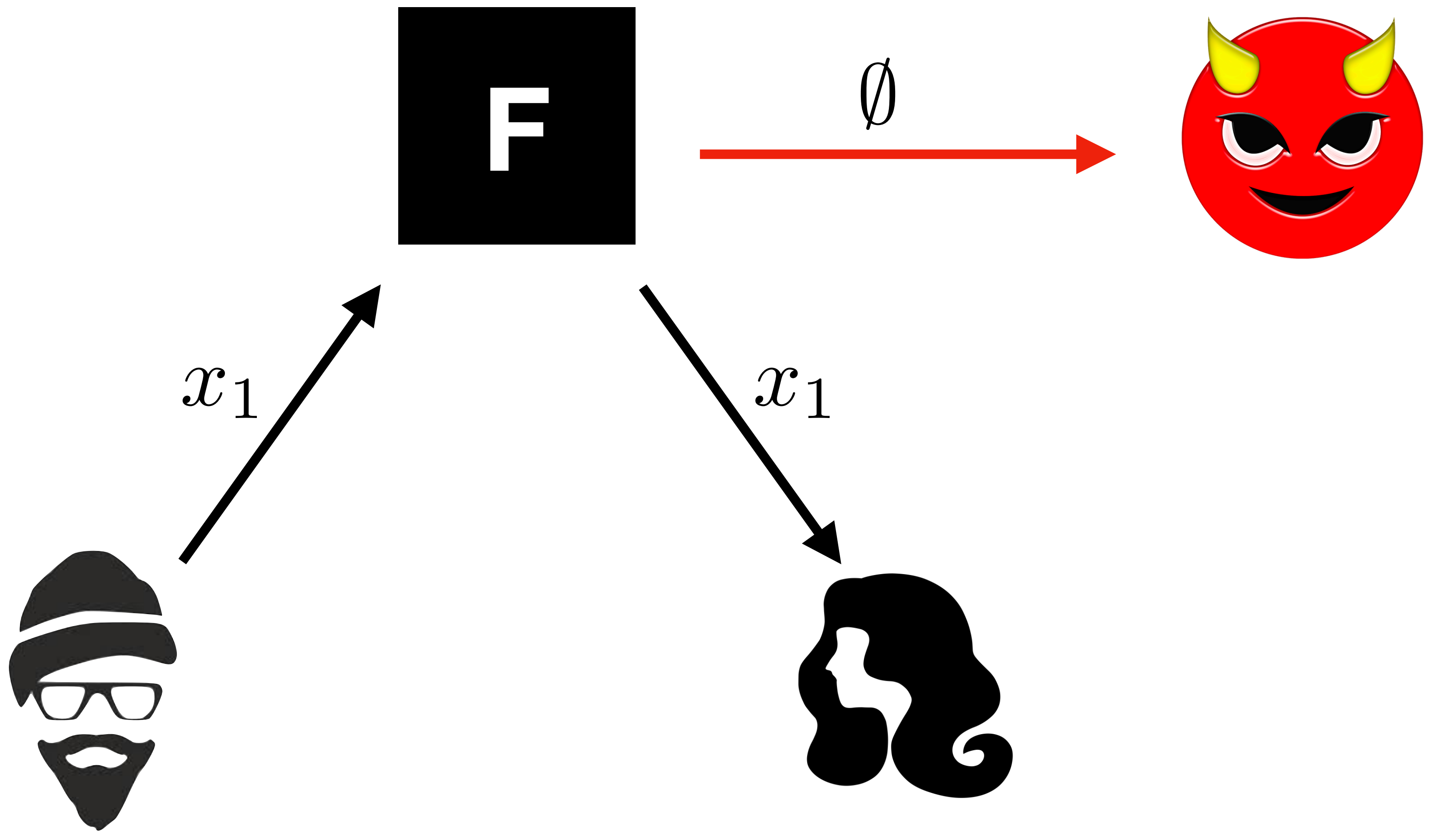


Simple Example: Secure Communication

Real World



Ideal World



Real Behavior

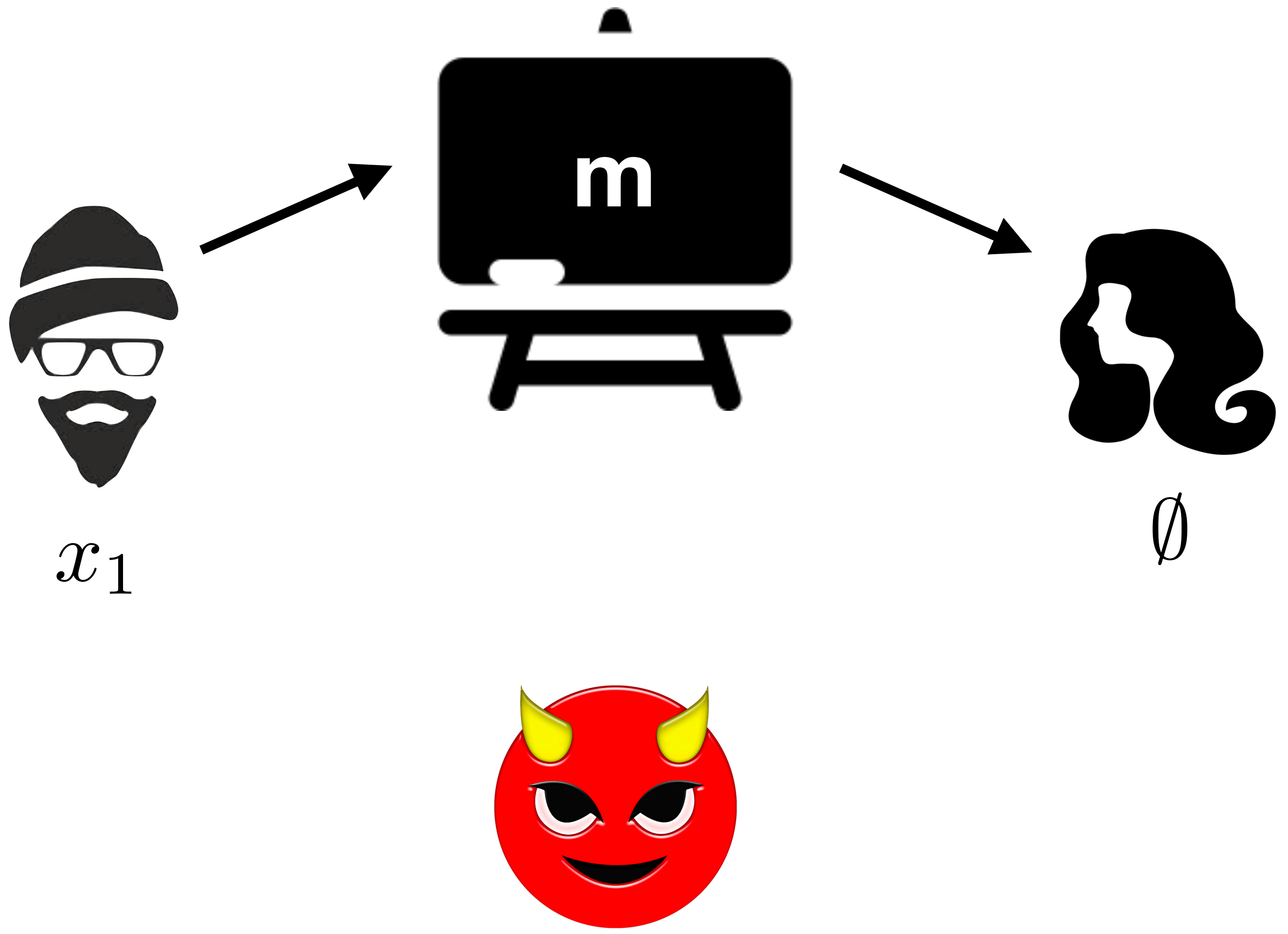
- Blackboard model (public communication)
- No corruption (but the adversary sees the board)

Ideal Behavior

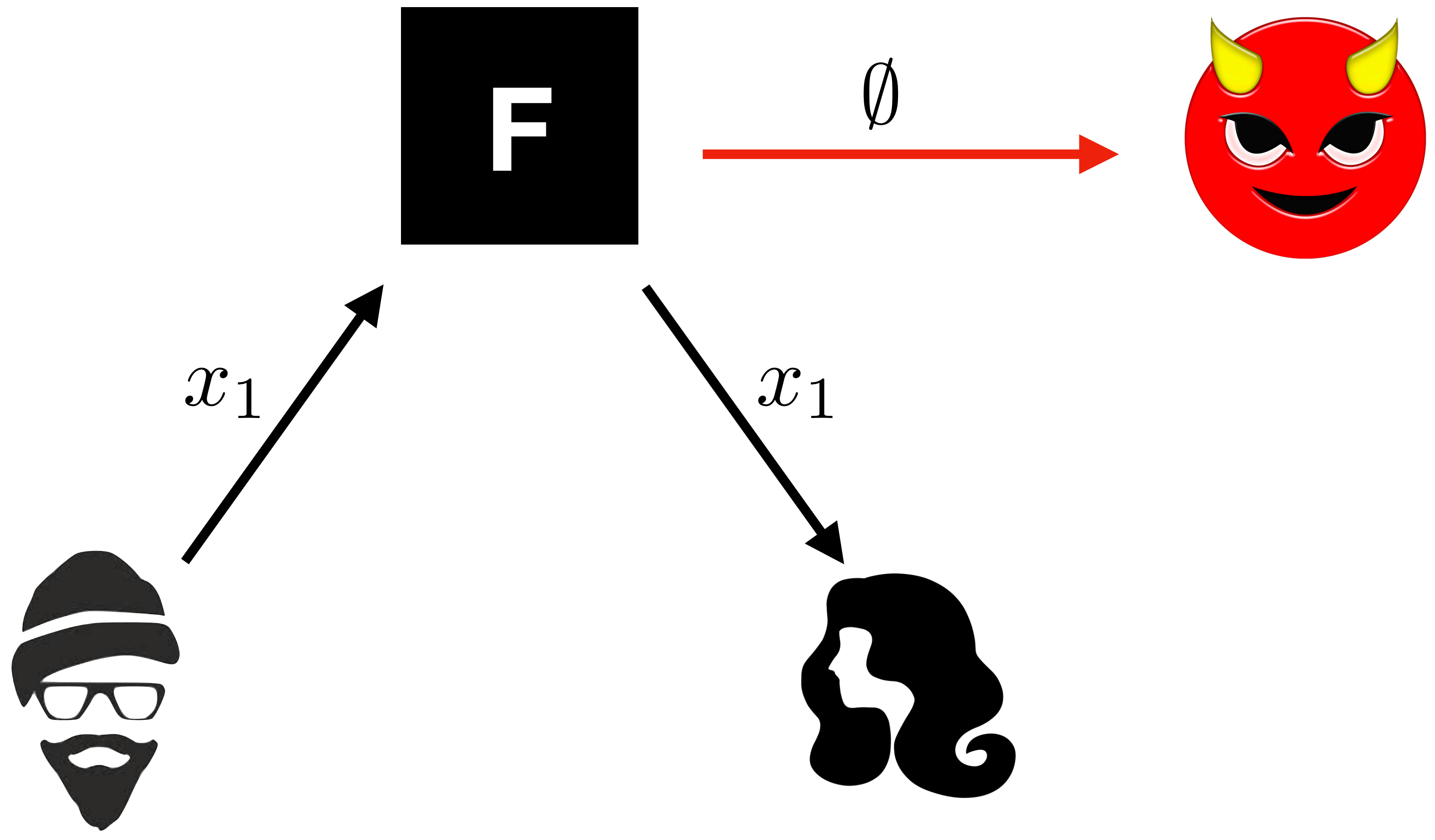
- The sender sends his input to **F**
- **F** sends it to the receiver; no leakage.

Simple Example: Secure Communication

Real World



Ideal World



Any idea how to do that?

Real Behavior

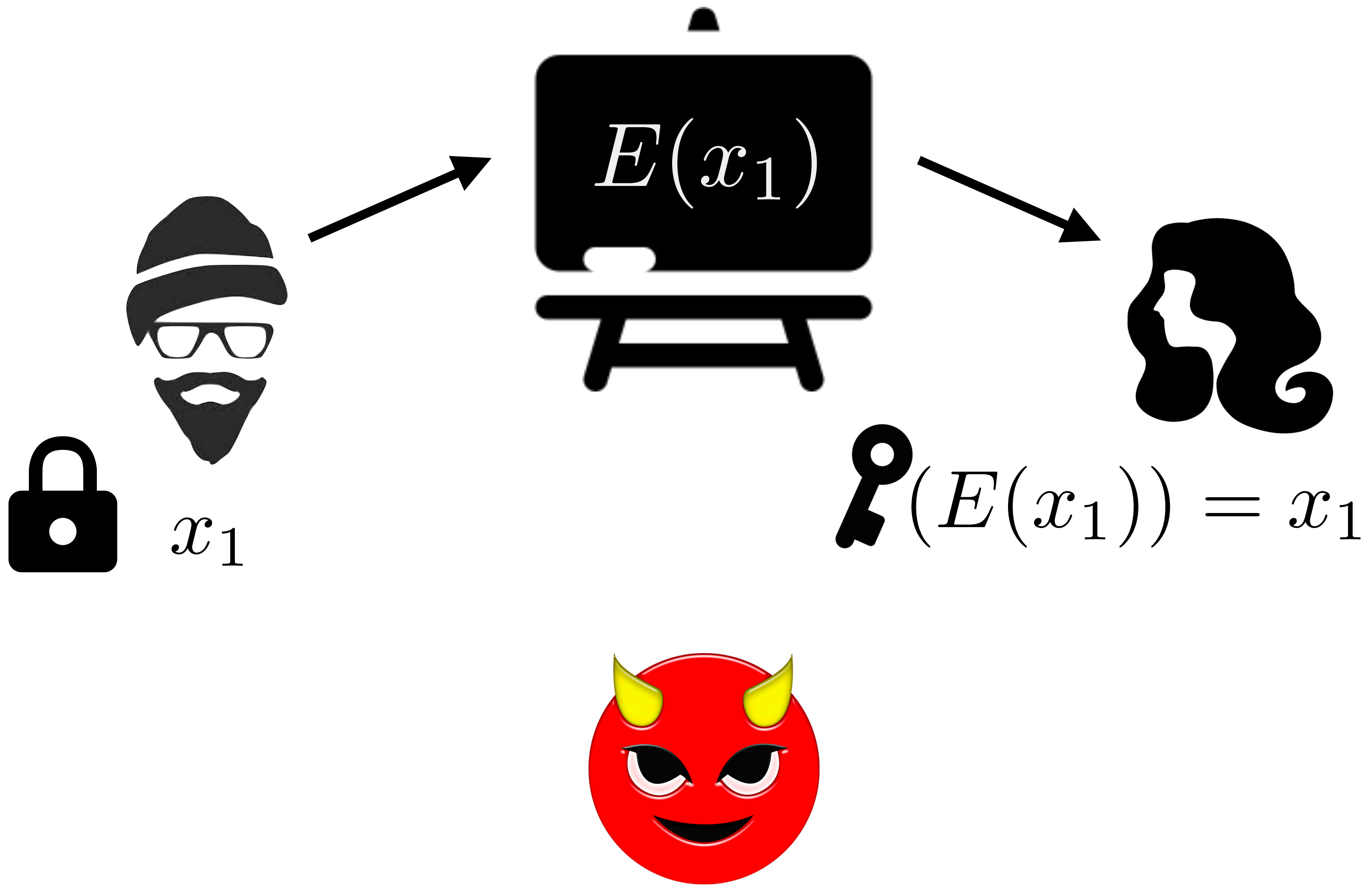
- Blackboard model (public communication)
- No corruption (but the adversary sees the board)

Ideal Behavior

- The sender sends his input to **F**
- **F** sends it to the receiver; no leakage.

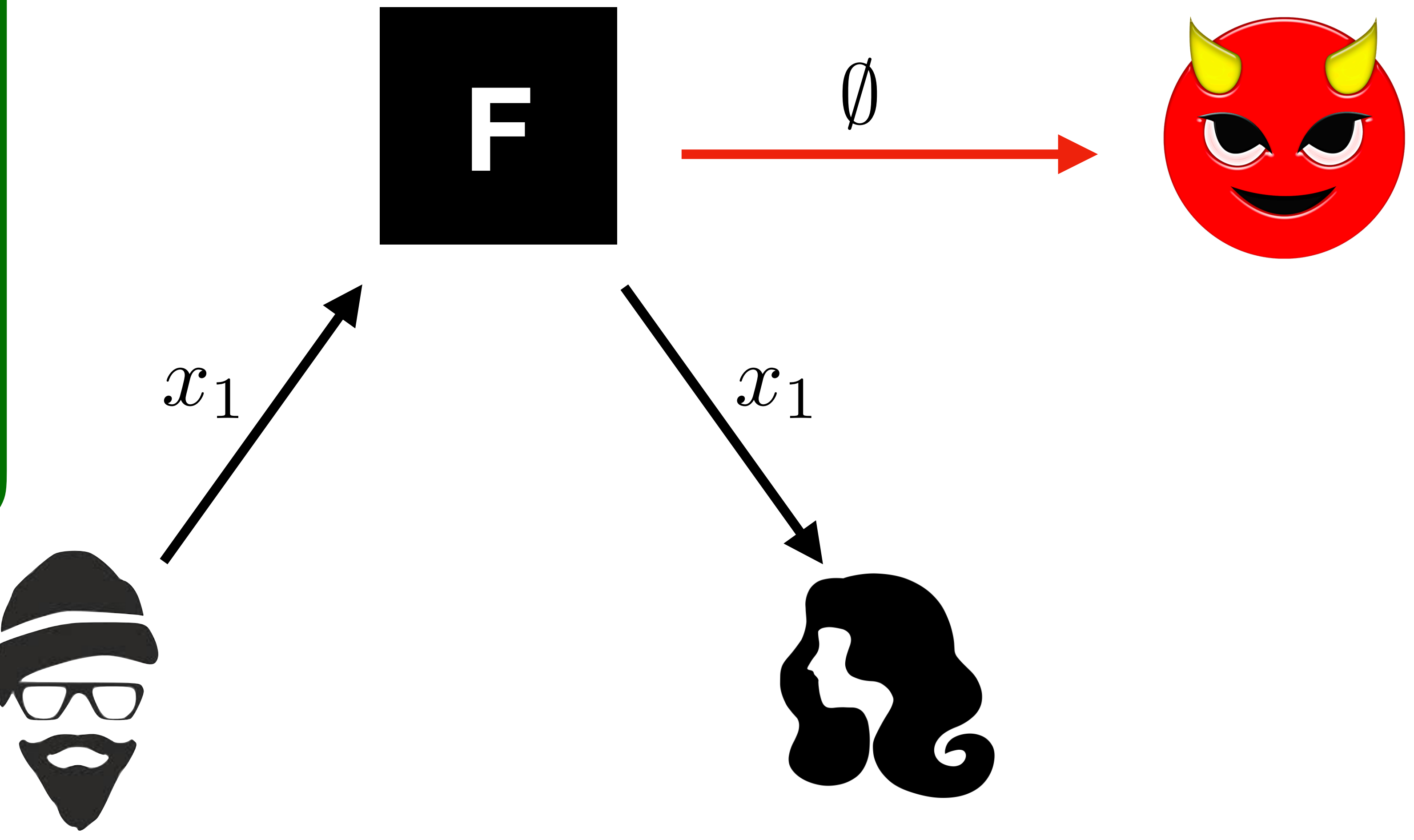
Simple Example: Secure Communication

Real World



E = IND-CPA secure encryption scheme. Sender message: input encrypted with the public key of the receiver.

Ideal World



Real Behavior

- Blackboard model (public communication)
- No corruption (but the adversary sees the board)

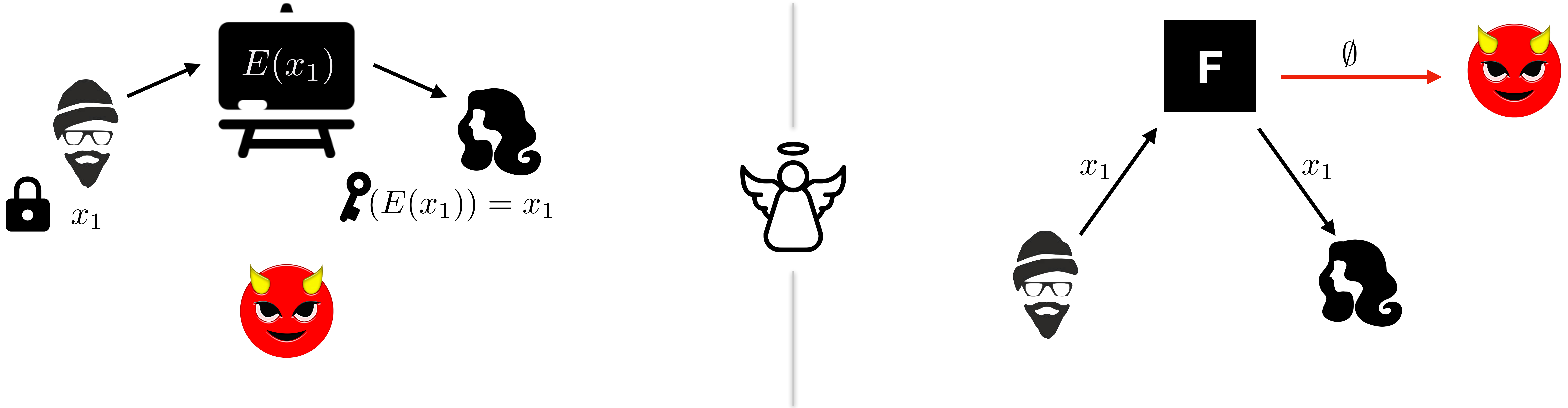
Ideal Behavior

- The sender sends his input to **F**
- **F** sends it to the receiver; no leakage.

Simple Example: Secure Communication

Real World

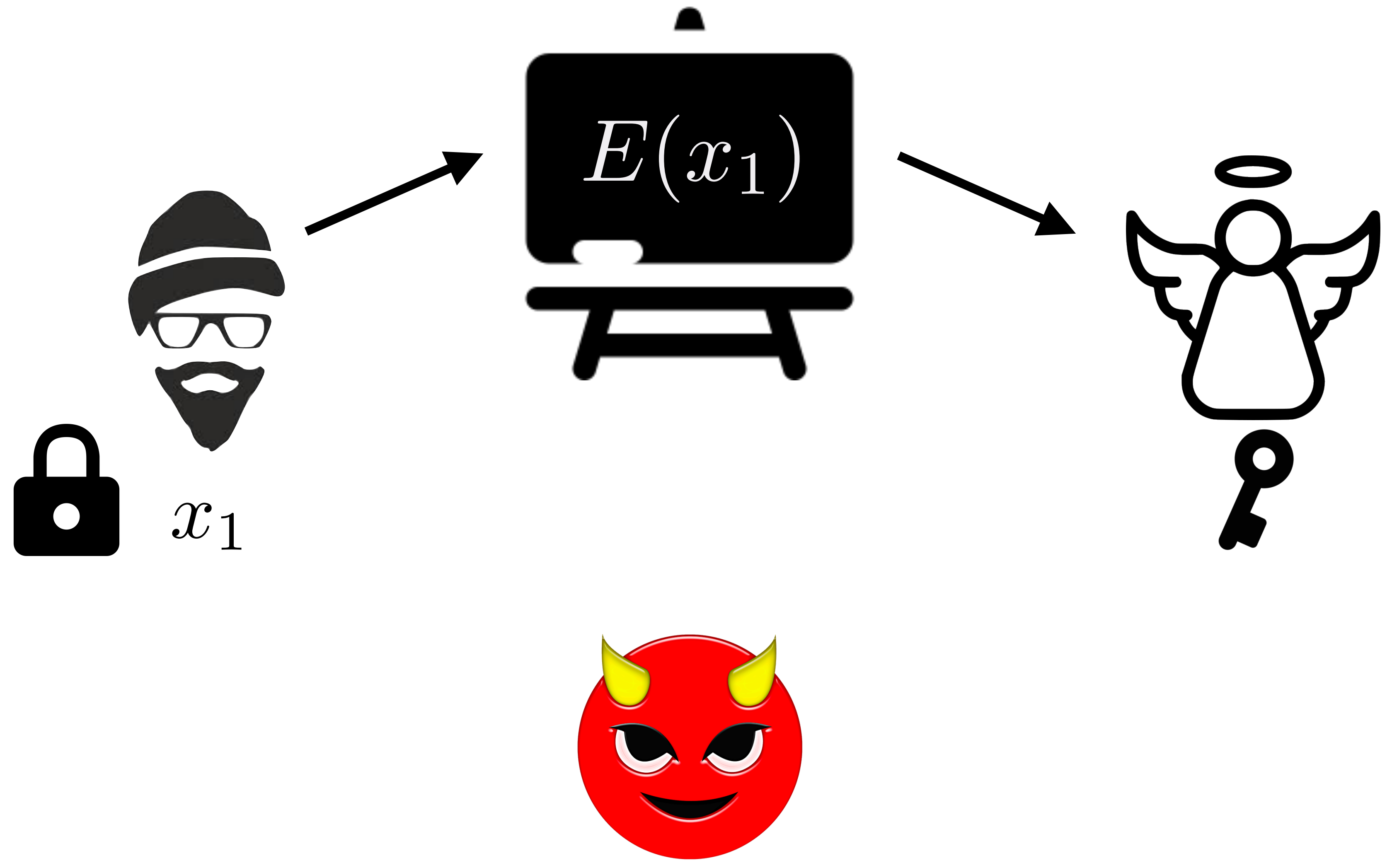
Ideal World



Simulation

Simple Example: Secure Communication

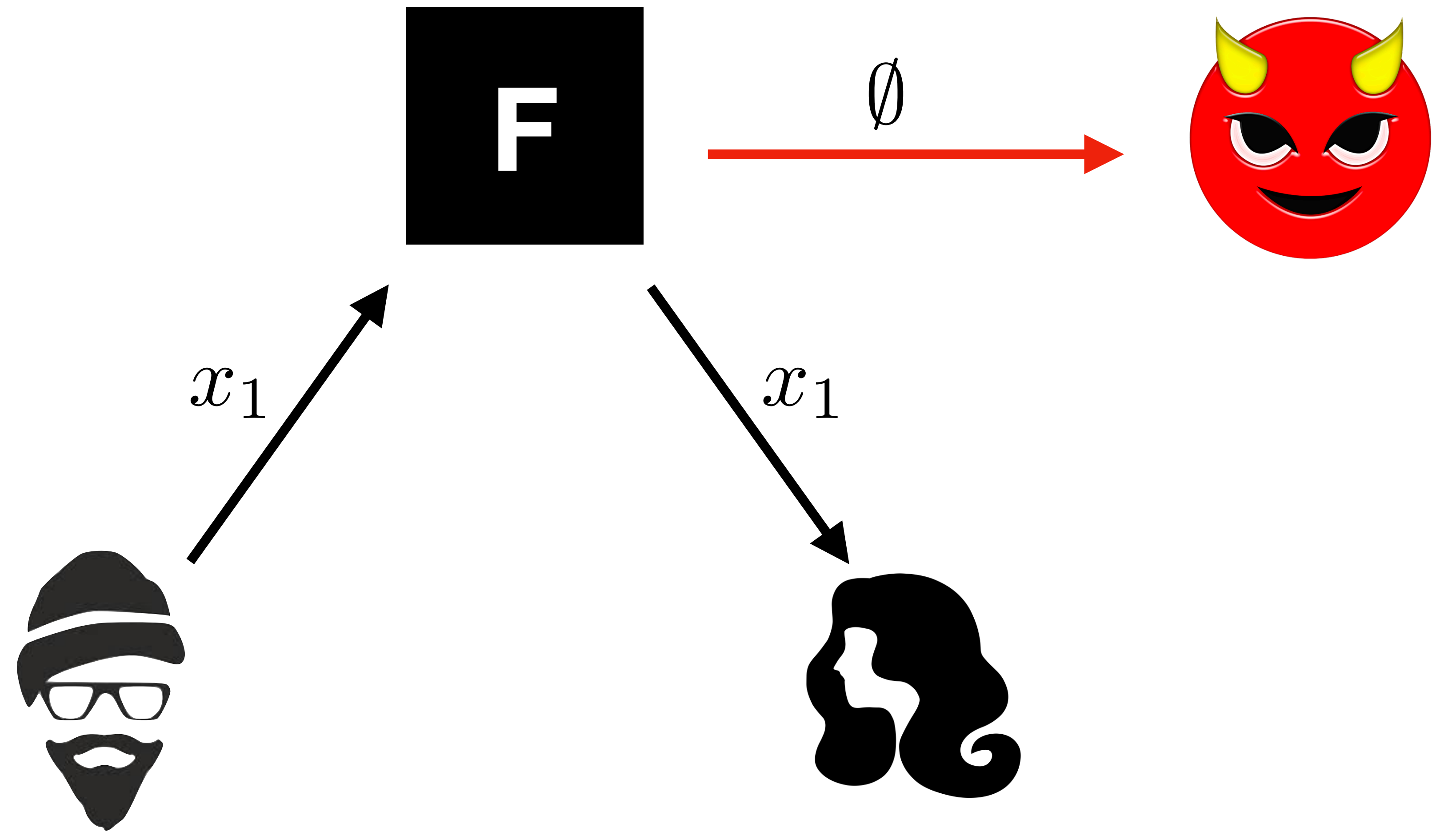
Real World



Real Behavior

Simulation

Ideal World



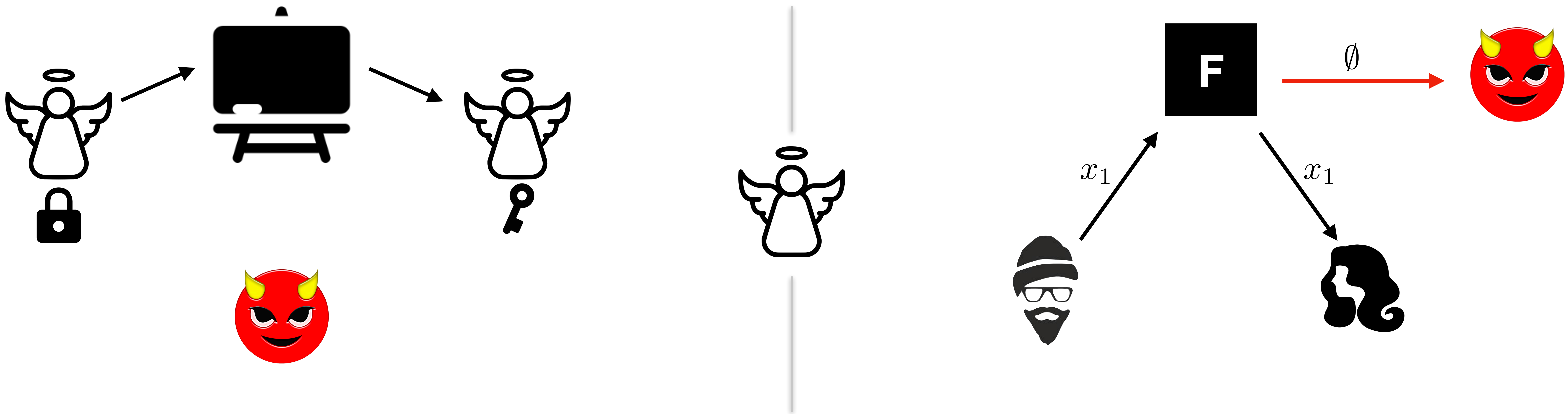
Ideal Behavior

Simulating 👩 is easy: 👼 generates a public key 🔒 honestly.

Simple Example: Secure Communication

Real World

Ideal World



Real Behavior

Simulation

Ideal Behavior

Simulating 👩 is easy: 😇 generates a public key 🔒 honestly.

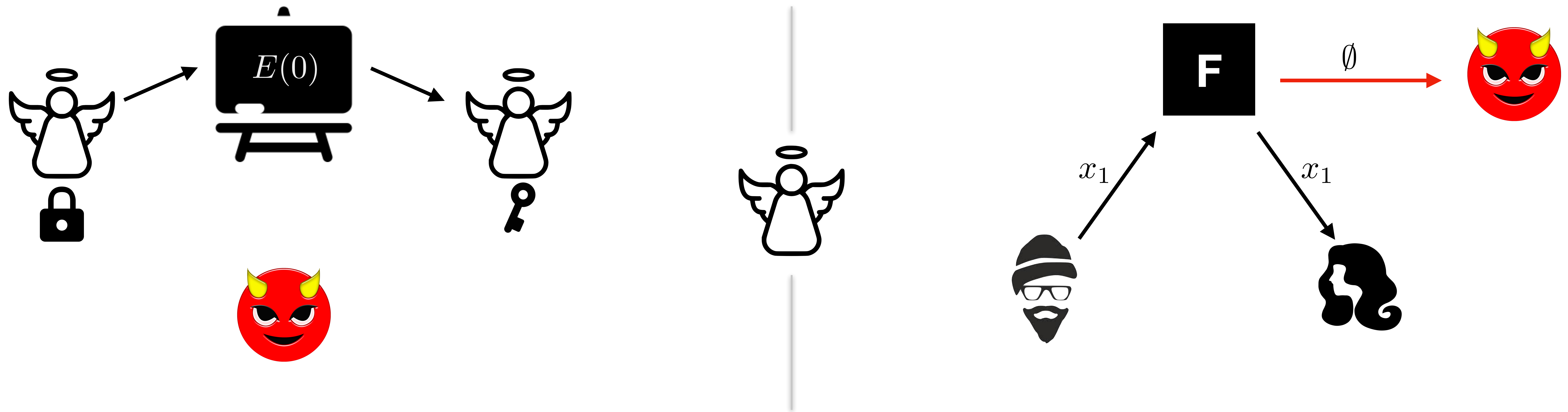
Simulating 🧔 is harder, since 😇 does not know x_1.

What does the simulator write on the board?

Simple Example: Secure Communication

Real World

Ideal World



Real Behavior

Simulation

Ideal Behavior

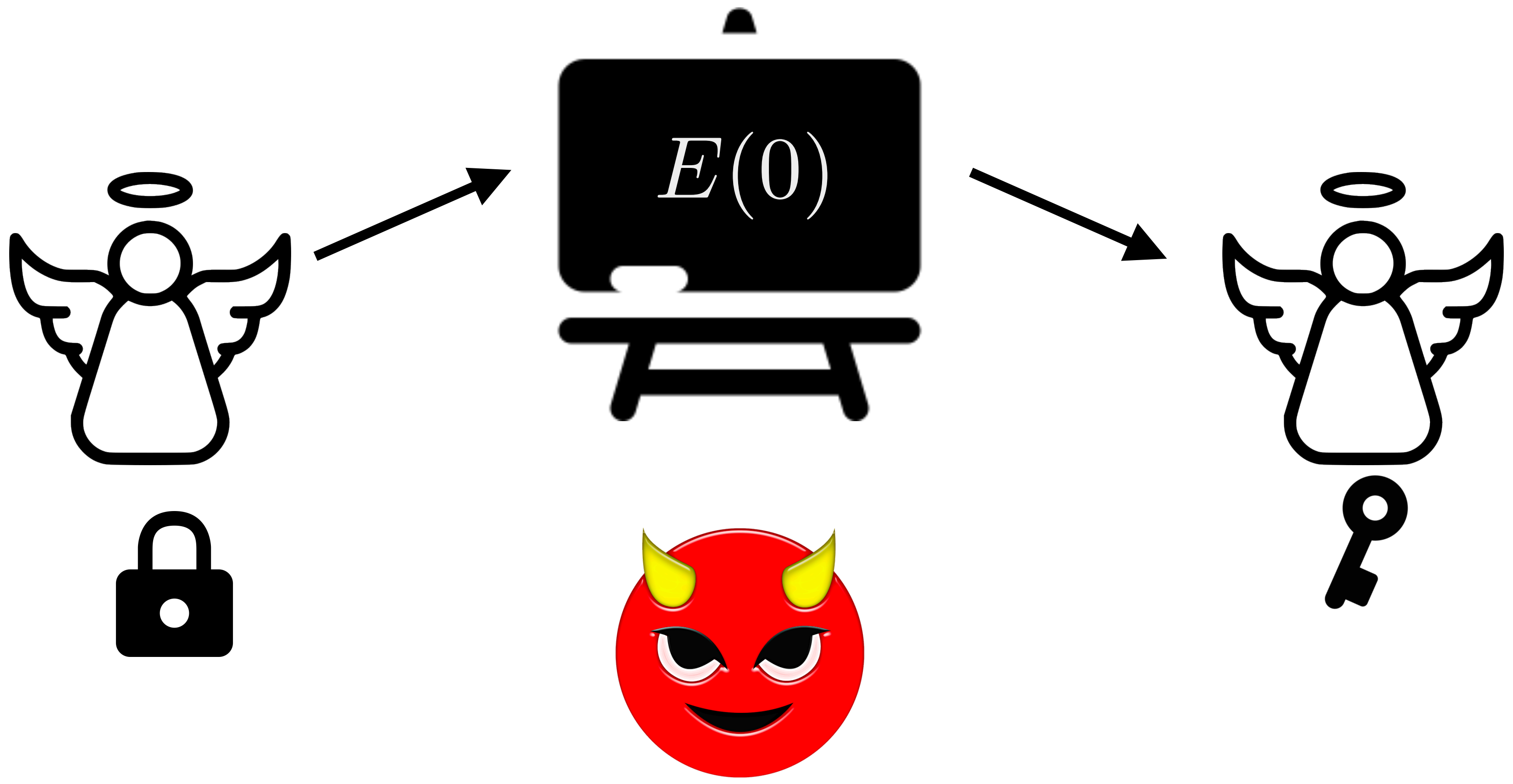
Simulating 👩 is easy: 😇 generates a public key 🔒 honestly.

Simulating 🧔 is harder, since 😇 does not know x_1.

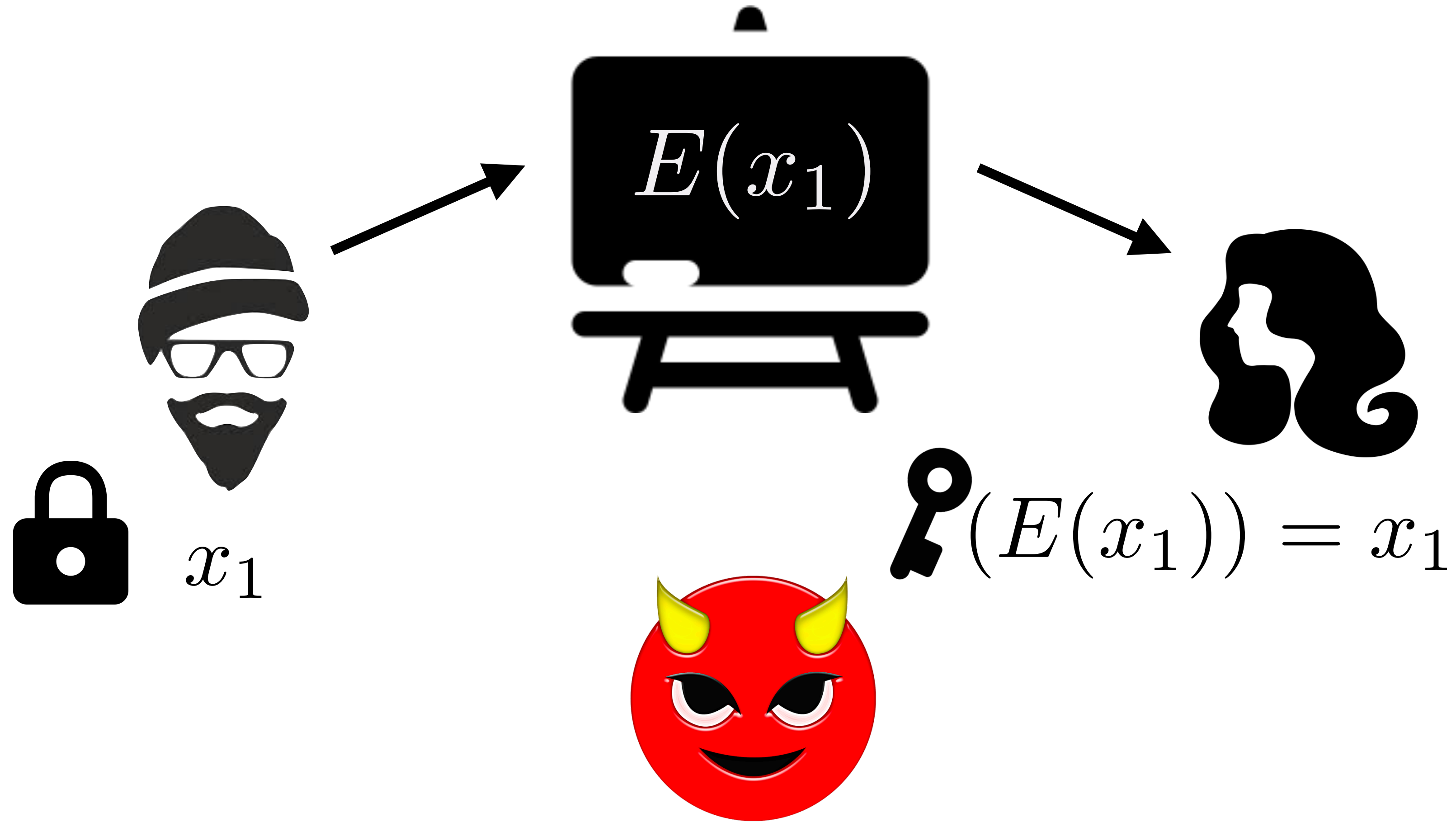
Solution: encrypt an arbitrary value instead!

Simple Example: Secure Communication

Simulation



Reality



Simulation

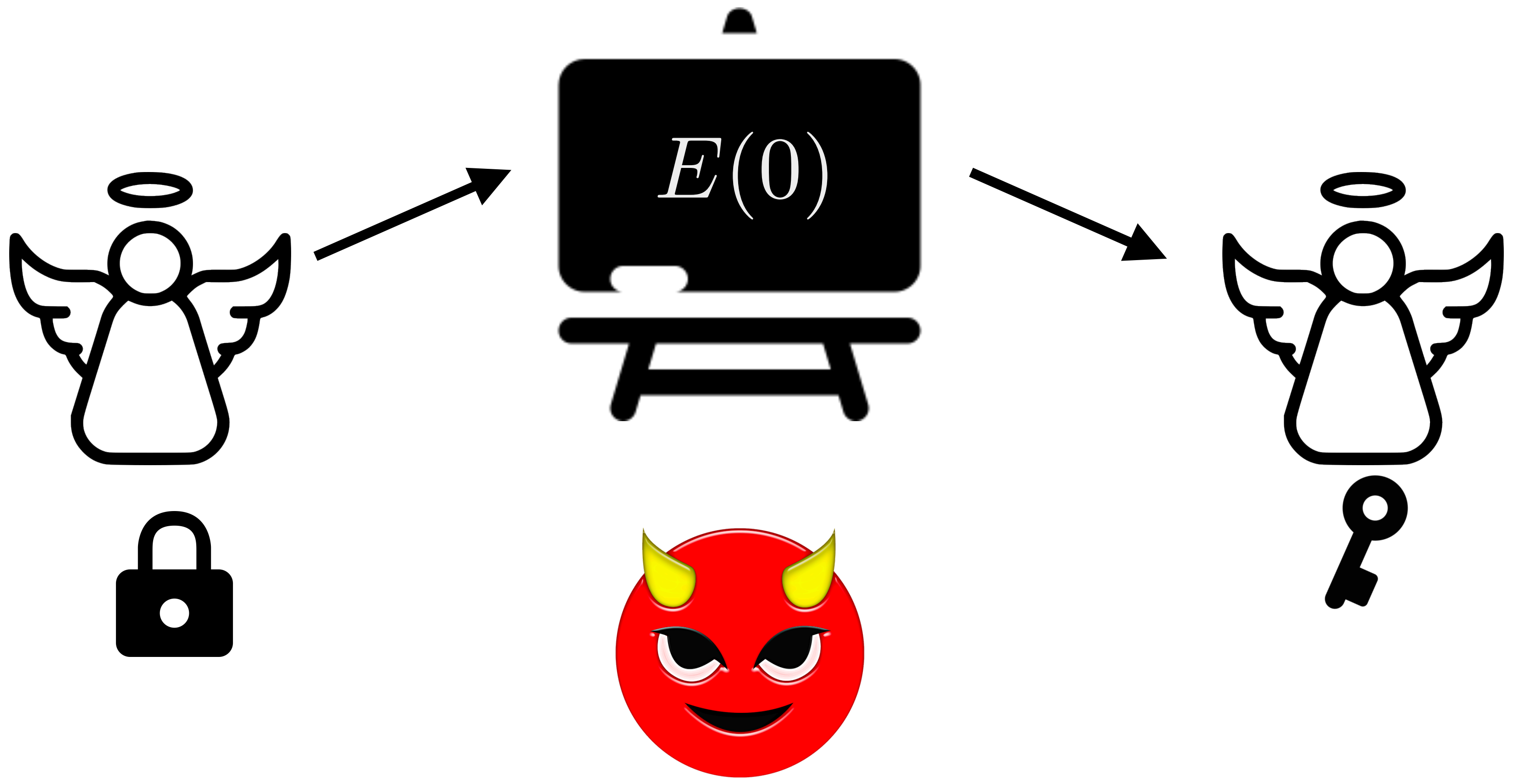
Simulating 👩 is easy: 👼 generates a public key 🔒 honestly.

Simulating 🧔 is harder, since 👼 does not know x_1.

Solution: encrypt an arbitrary value instead!

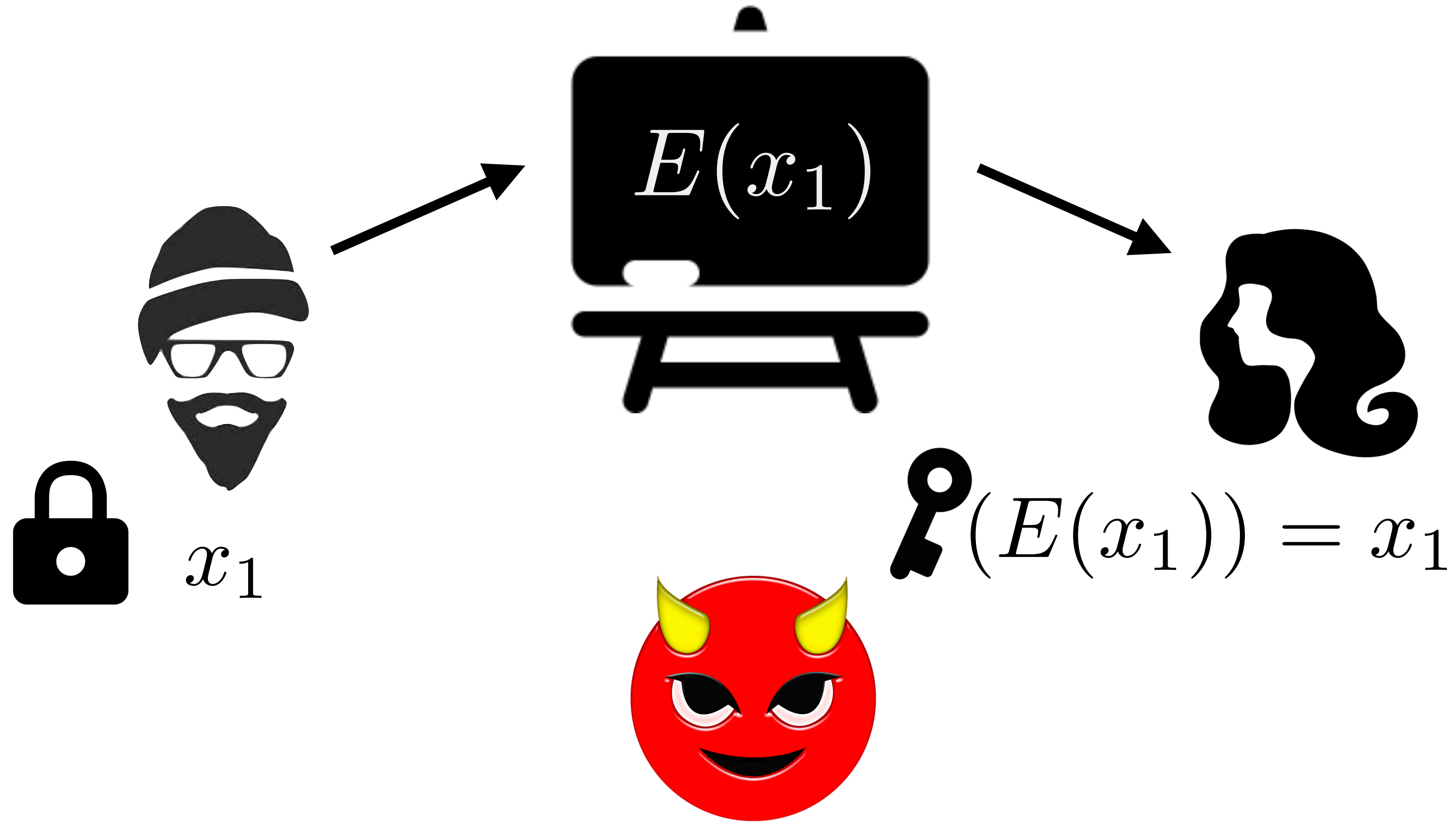
Simple Example: Secure Communication

Simulation



View: 🔒 , $E(0)$

Reality



Simulation

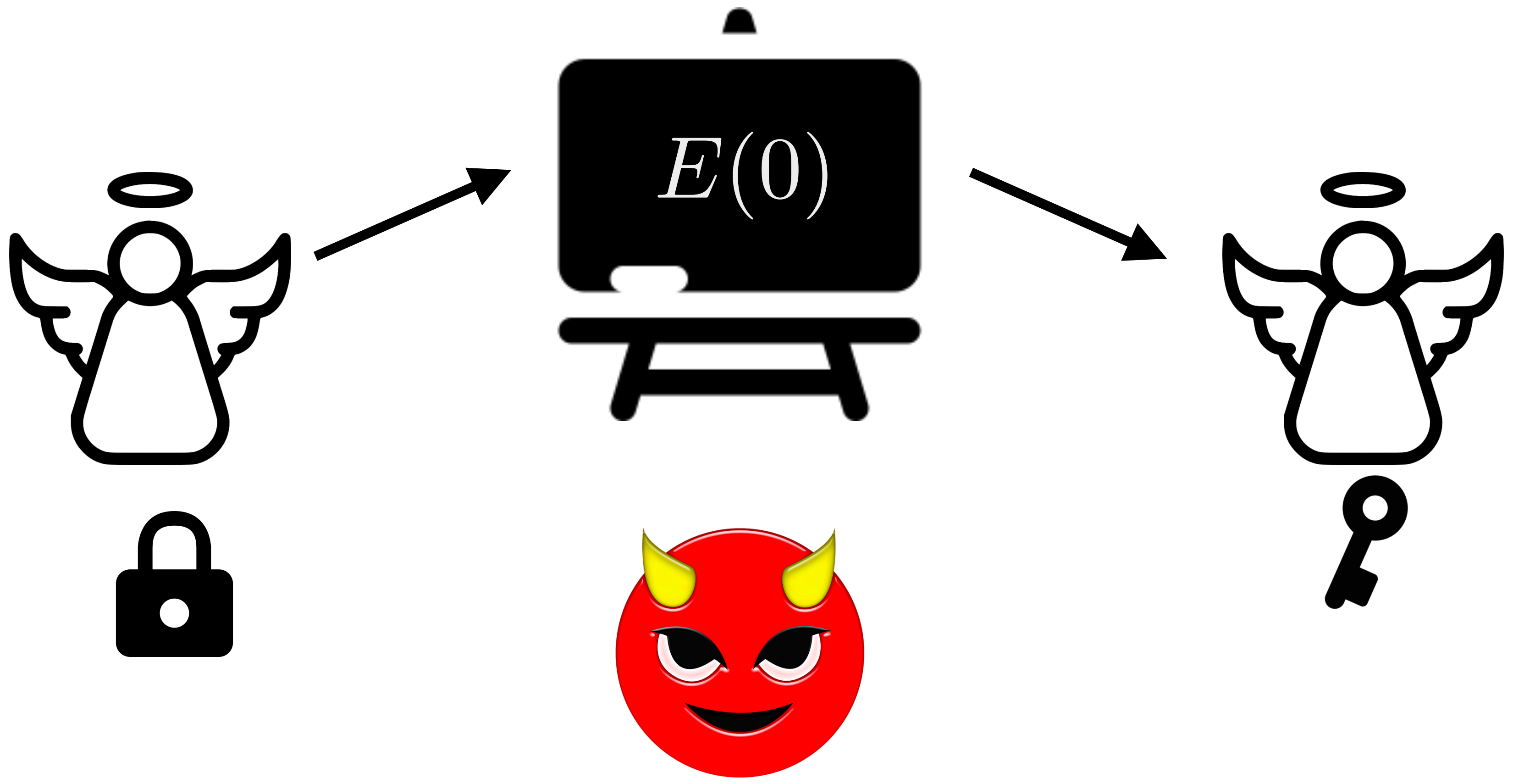
Simulating 👩 is easy: 😇 generates a public key 🔒 honestly.

Simulating 🧔 is harder, since 😇 does not know x_1.

Solution: encrypt an arbitrary value instead!

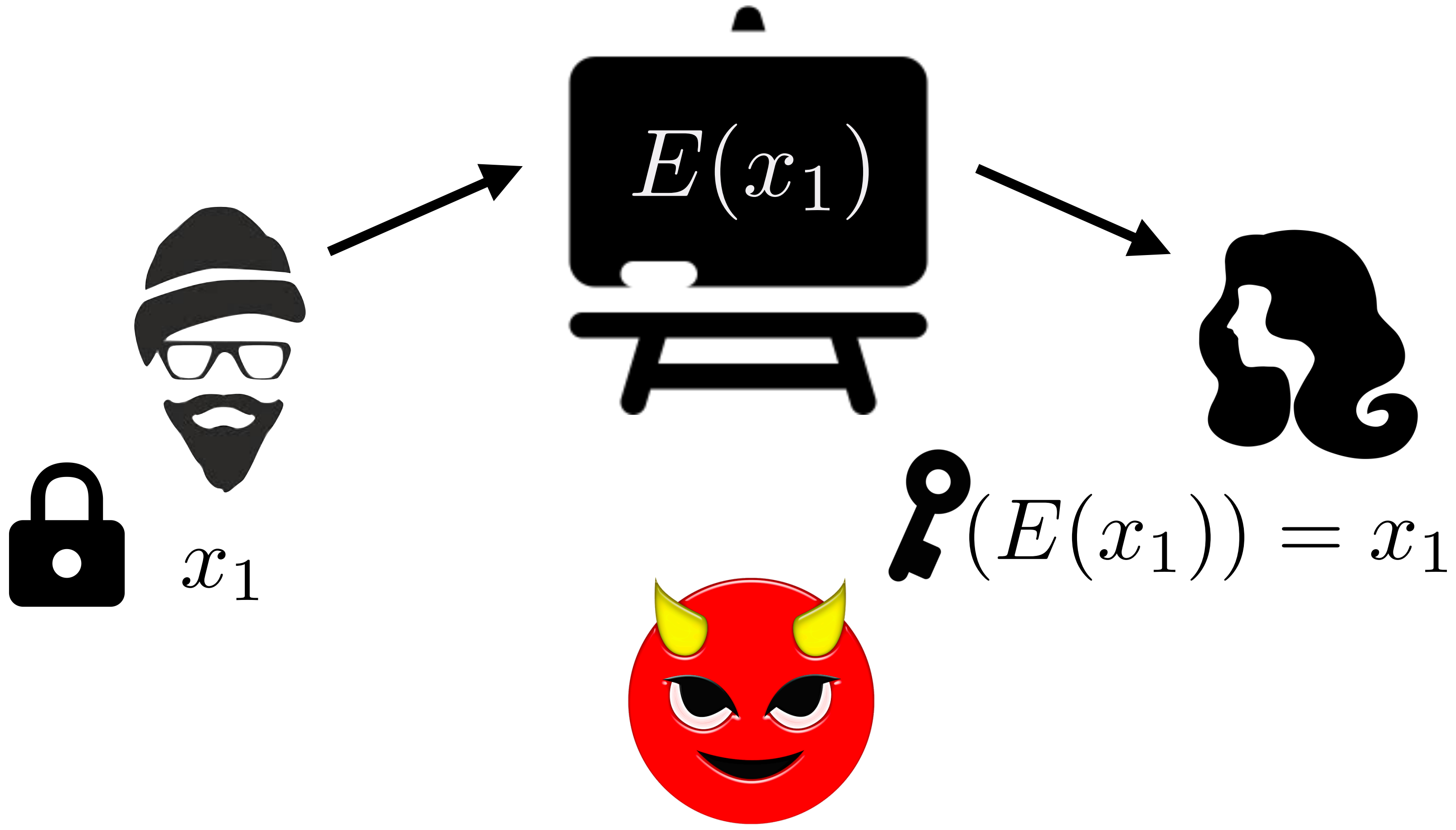
Simple Example: Secure Communication

Simulation



View: 🔒 , $E(0)$

Reality



View: 🔒 , $E(x_1)$

Simulation

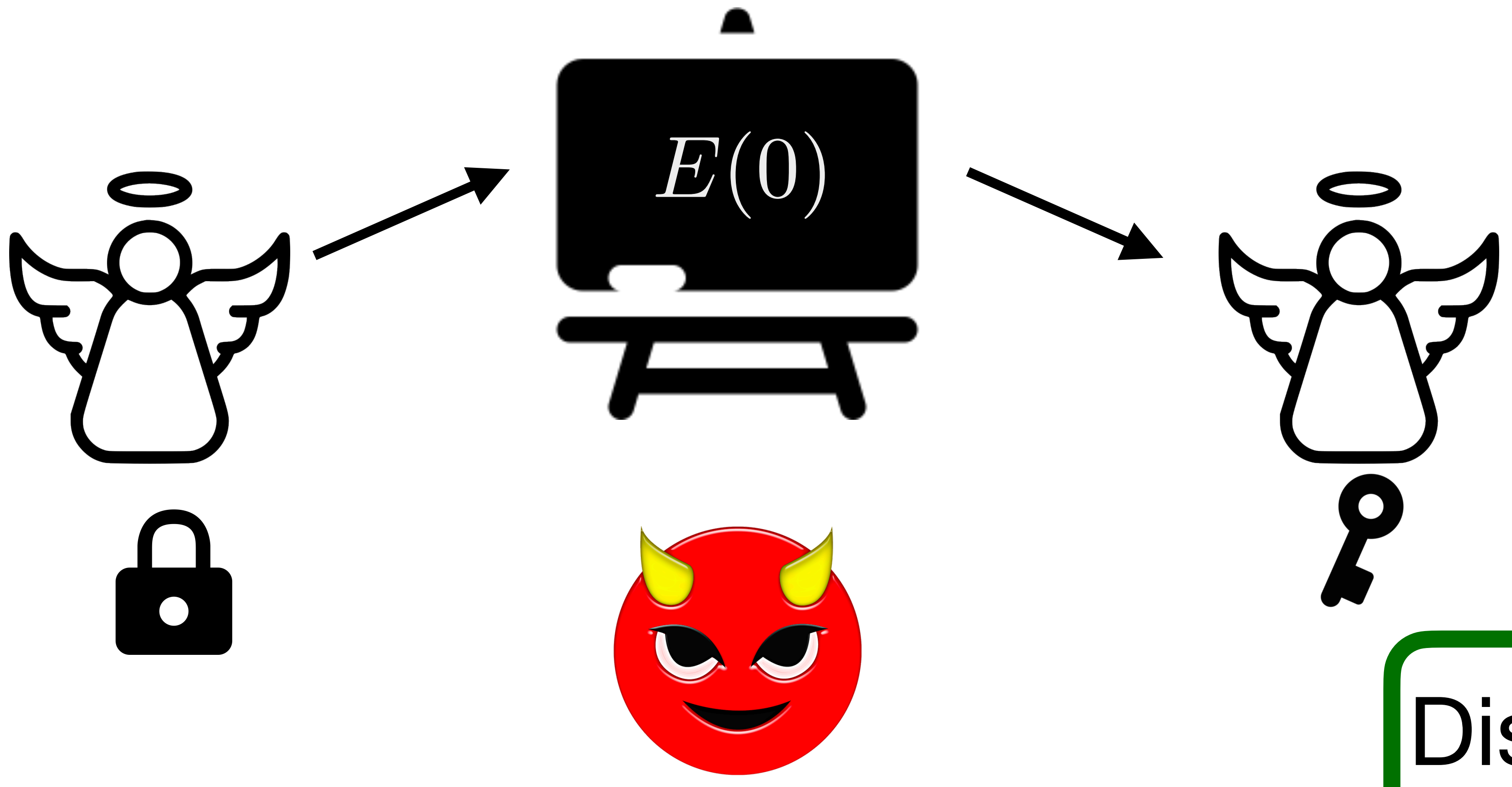
Simulating 👩 is easy: 👼 generates a public key 🔒 honestly.

Simulating 🧔 is harder, since 👼 does not know x_1.

Solution: encrypt an arbitrary value instead!

Simple Example: Secure Communication

Simulation



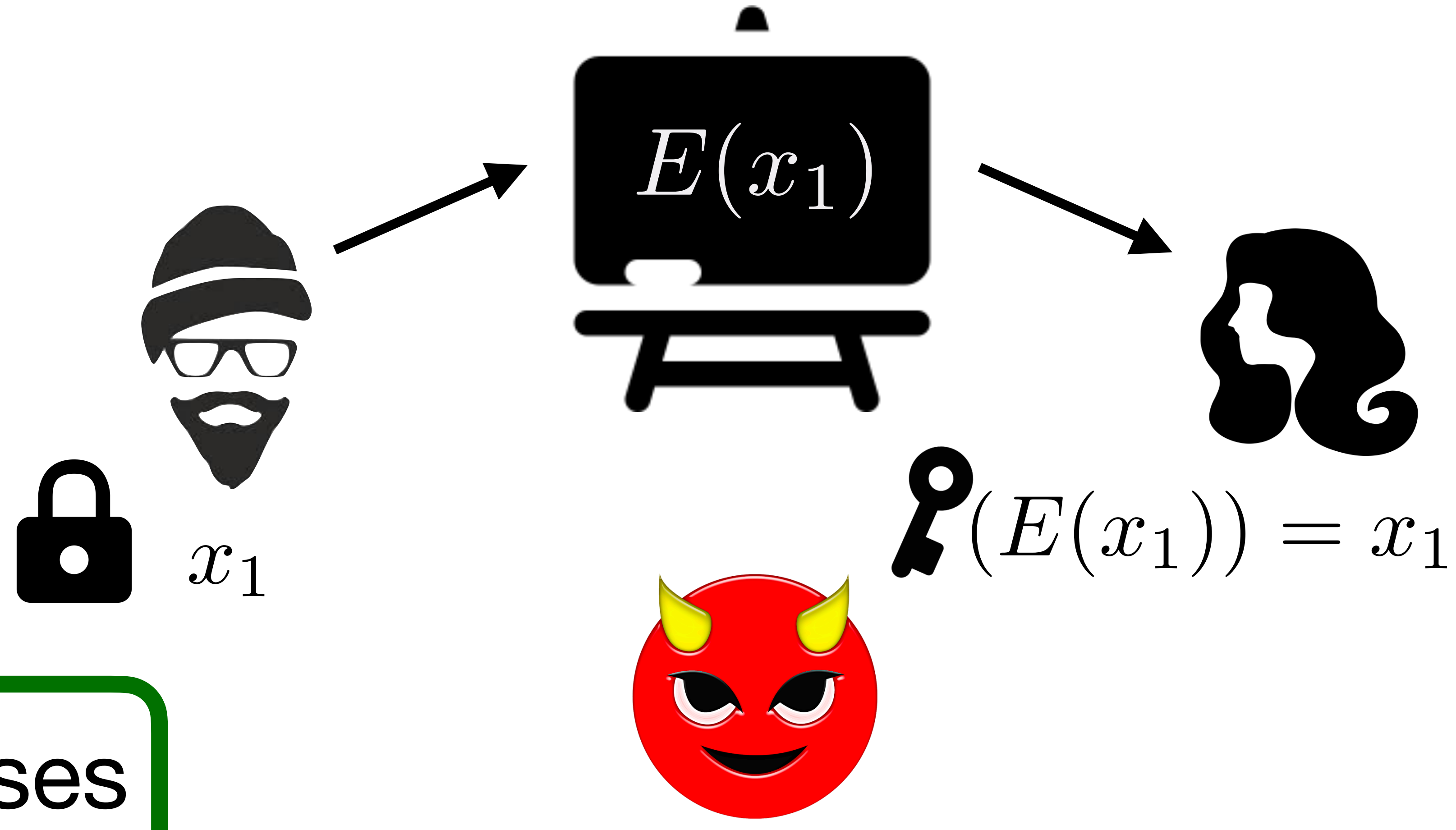
View: 🔒 , $E(0)$

Distinguishing the two cases

$\Longleftrightarrow$

Breaking the IND-CPA of E

Reality



View: 🔒 , $E(x_1)$

Simulation

Simulating 👩 is easy: 👼 generates a public key 🔒 honestly.

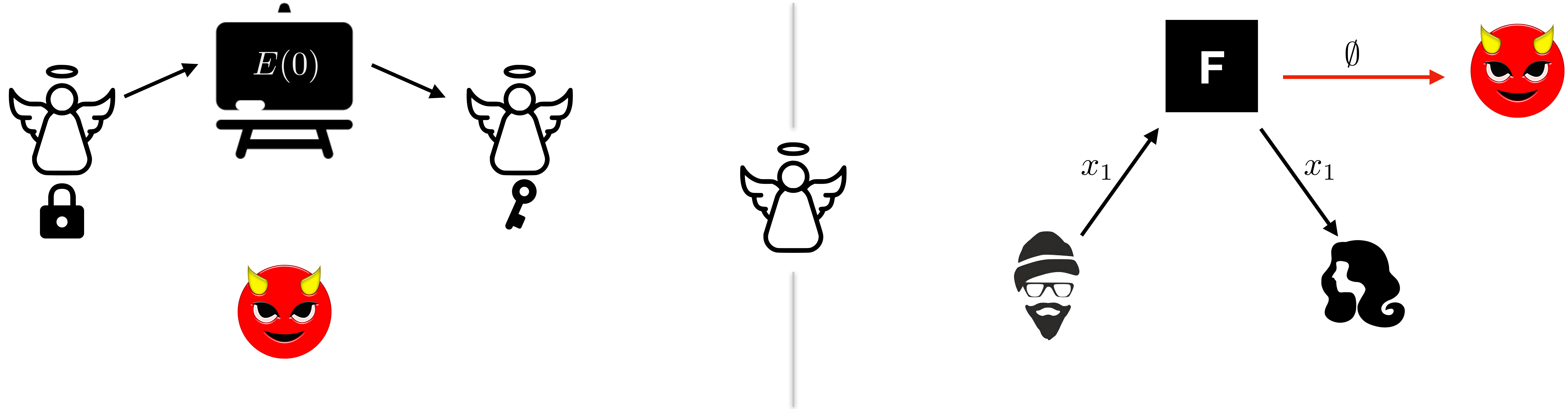
Simulating 🧔 is harder, since 👼 does not know x_1.

Solution: encrypt an arbitrary value instead!

Simple Example: Secure Communication

Real World

Ideal World

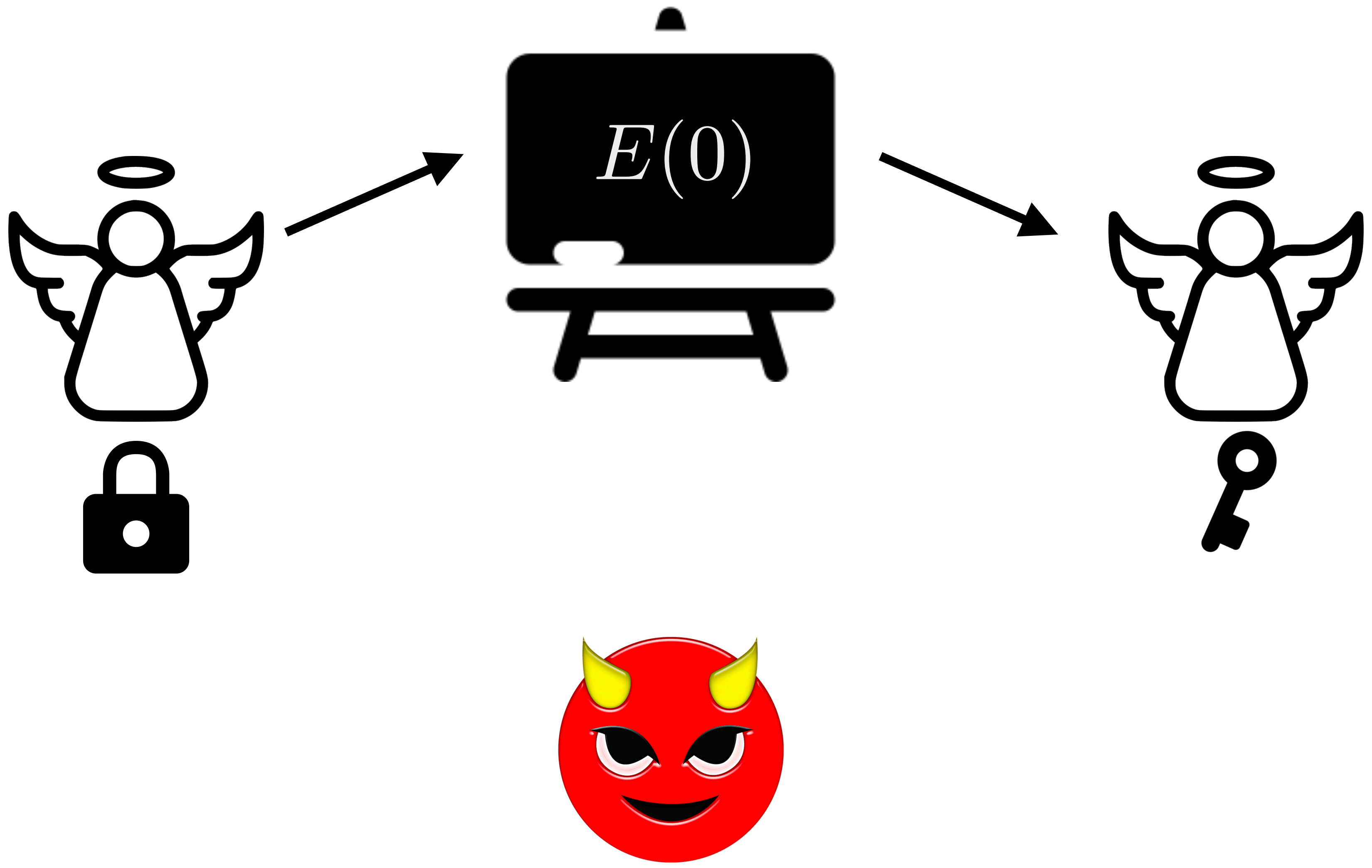


Simulation

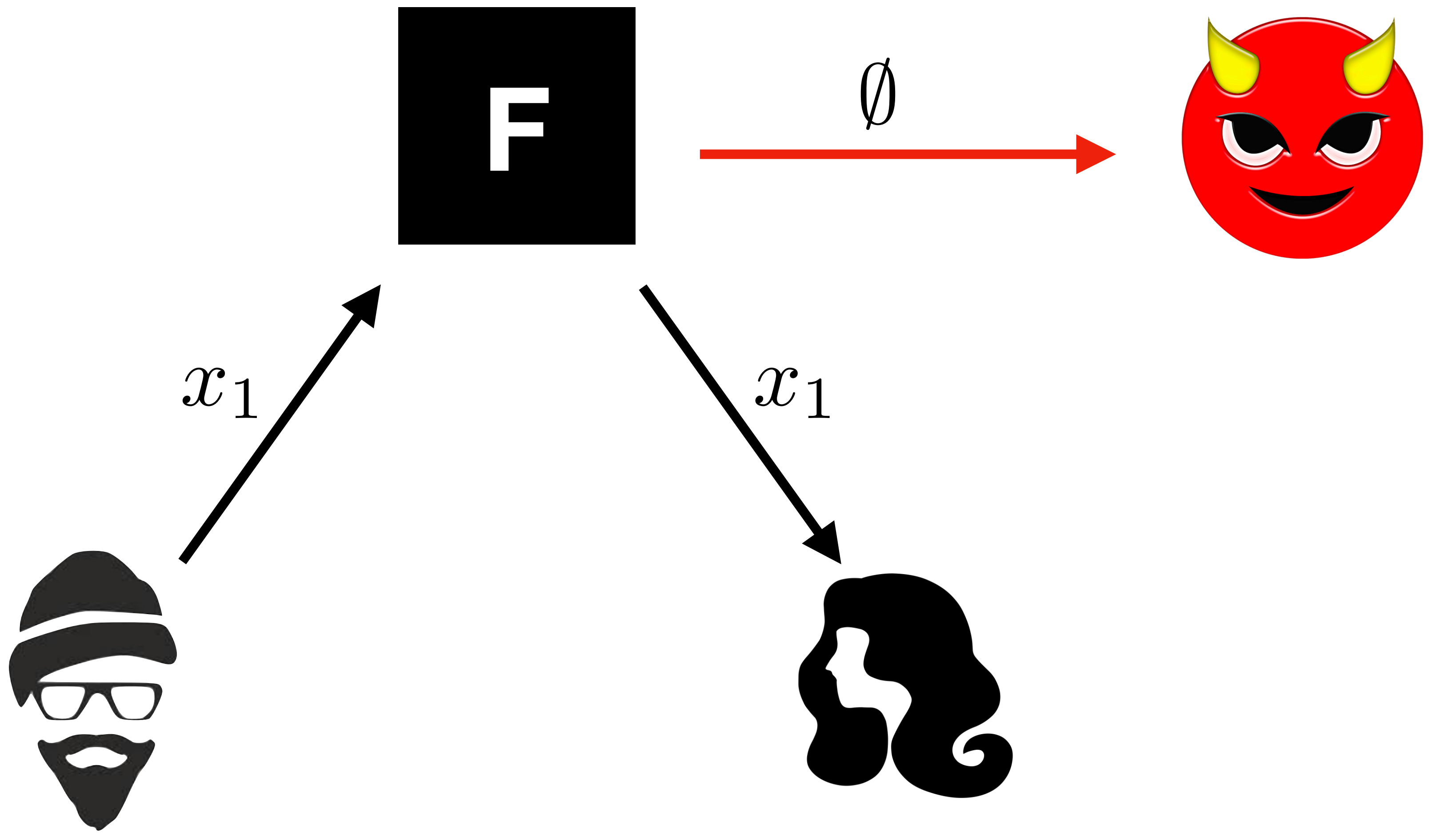
The simulation is indistinguishable from the real protocol if E is IND-CPA secure, hence the protocol securely *emulates* the ideal functionality F under the assumption that E is IND-CPA secure.

Simple Example: Secure Communication

Real World



Ideal World



Really?

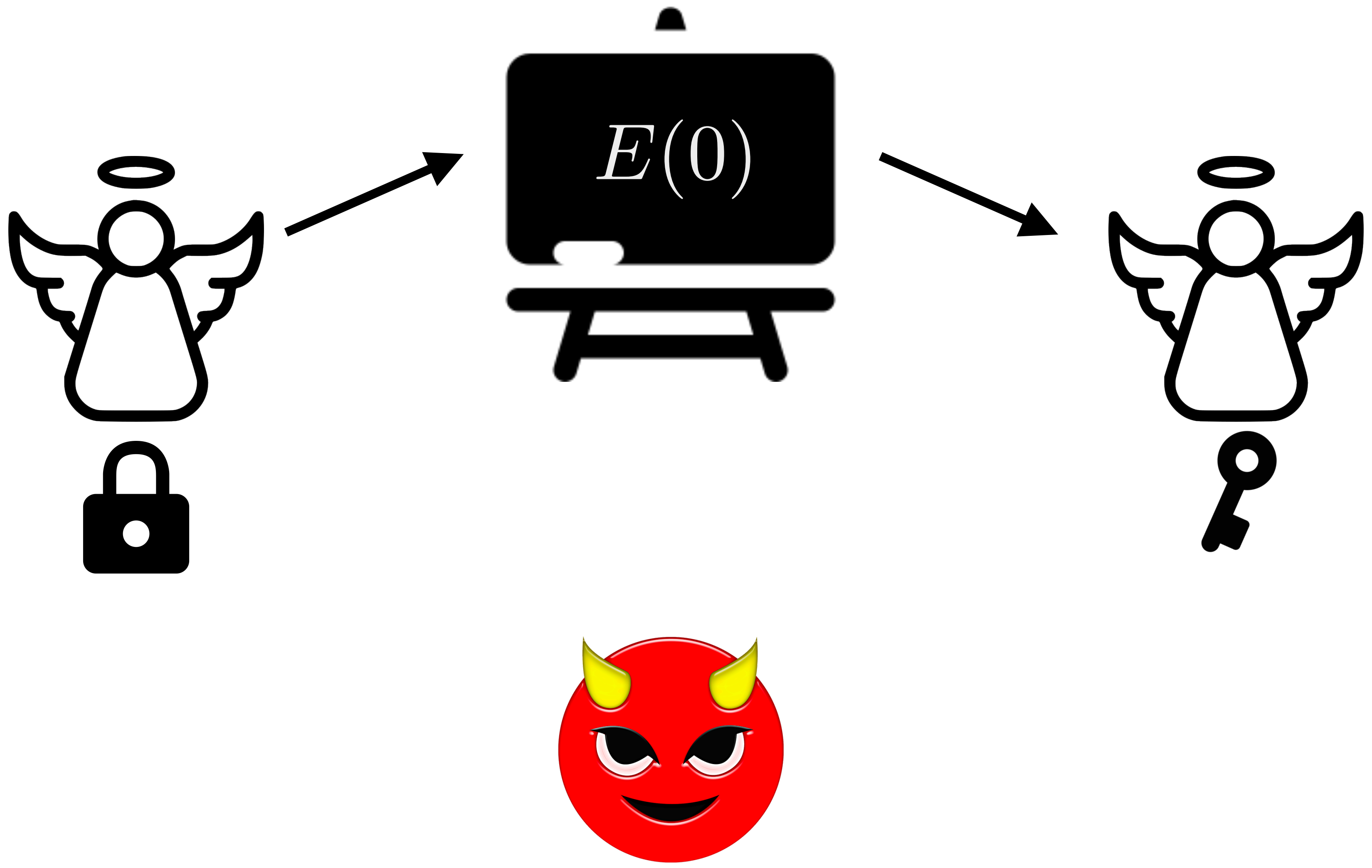
Simulation

View: 🔒 , $E(0)$ $\approx$ **View:** 🔒 , $E(x_1)$

⚠ **Cannot work: the plaintexts do not have the same size!**

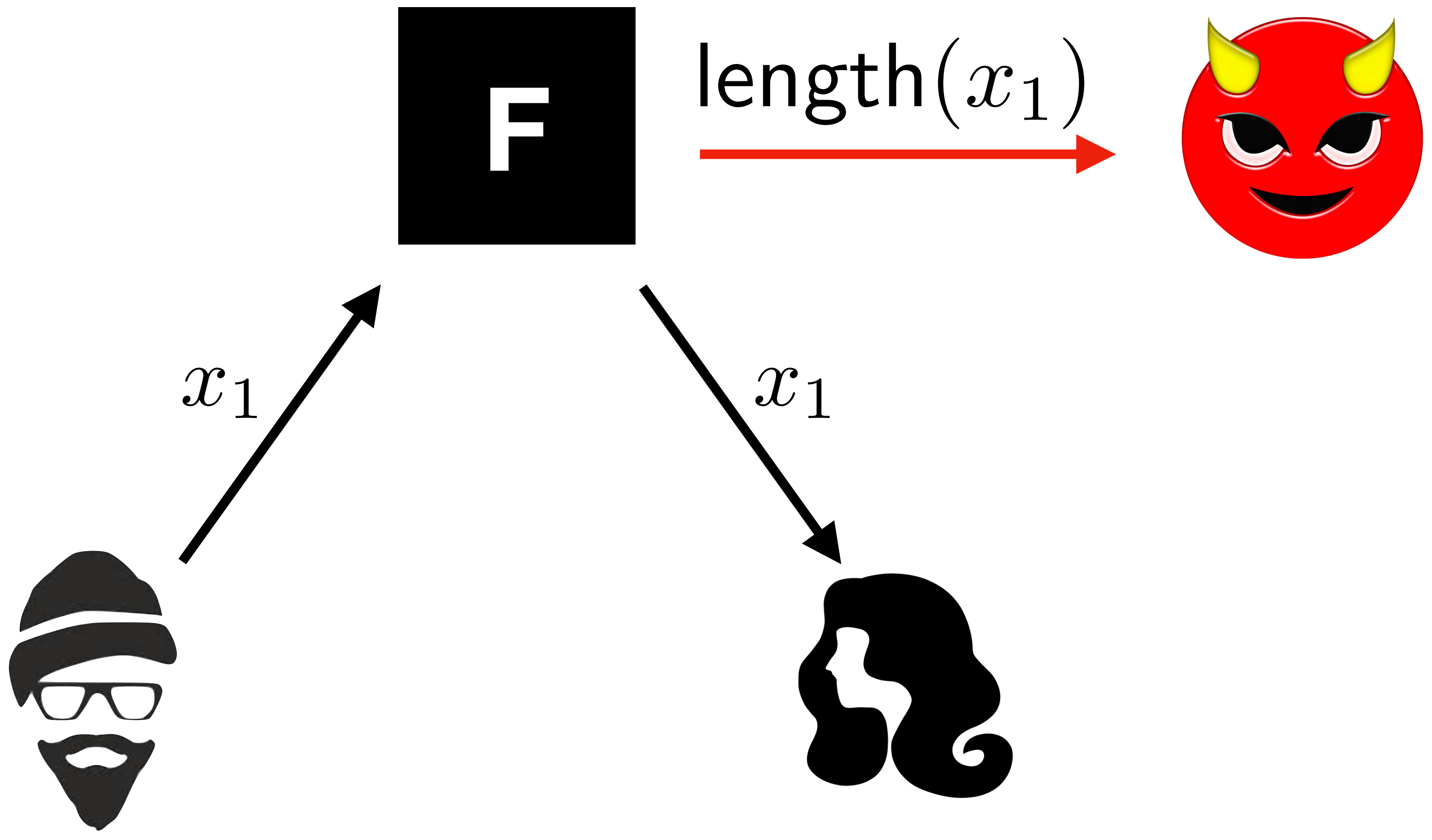
Simple Example: Secure Communication

Real World



Solution:
- Allow this leakage

Ideal World



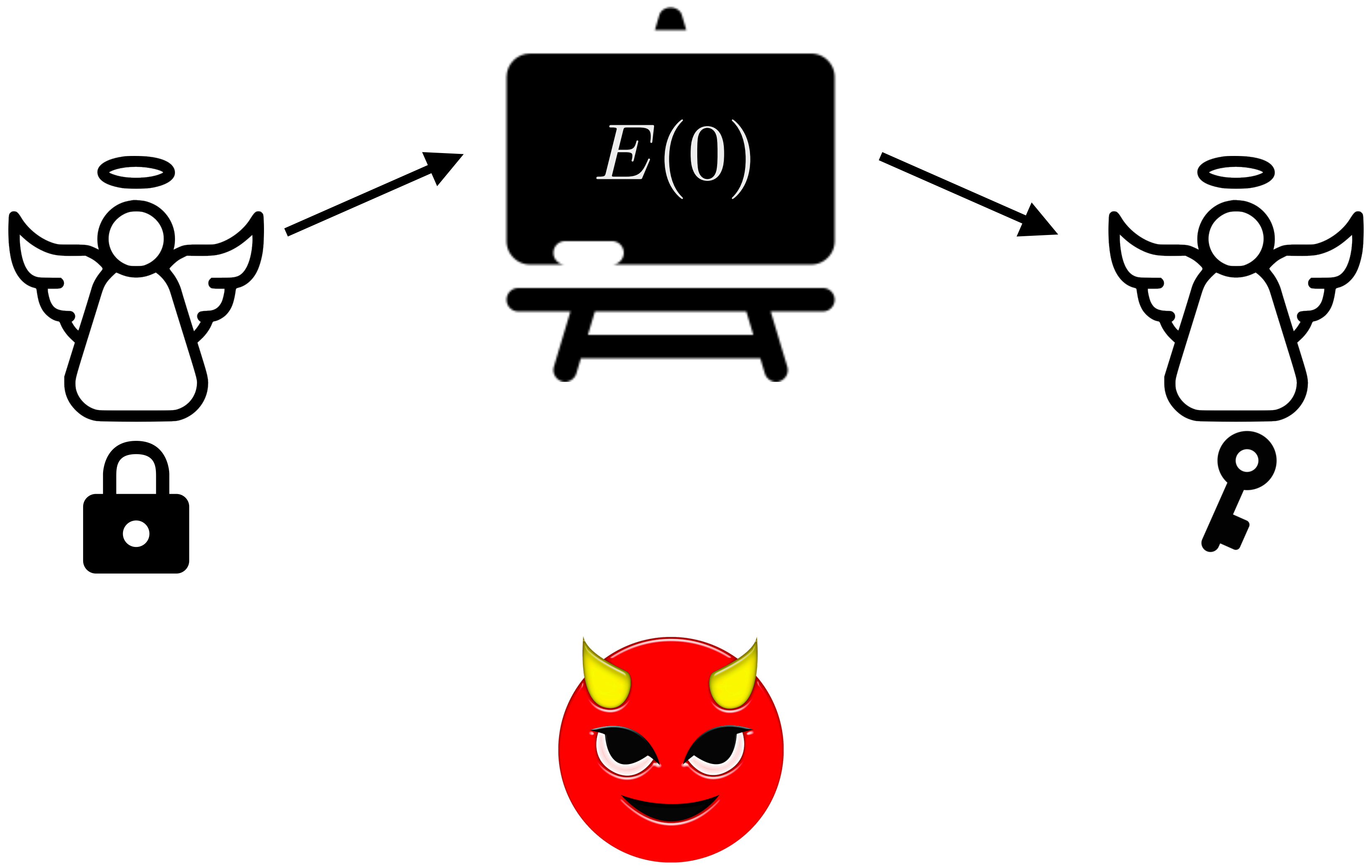
Simulation

View: 🔒 , $E(0)$ $\approx$ **View:** 🔒 , $E(x_1)$

⚠ **Cannot work: the plaintexts do not have the same size!**

Simple Example: Secure Communication

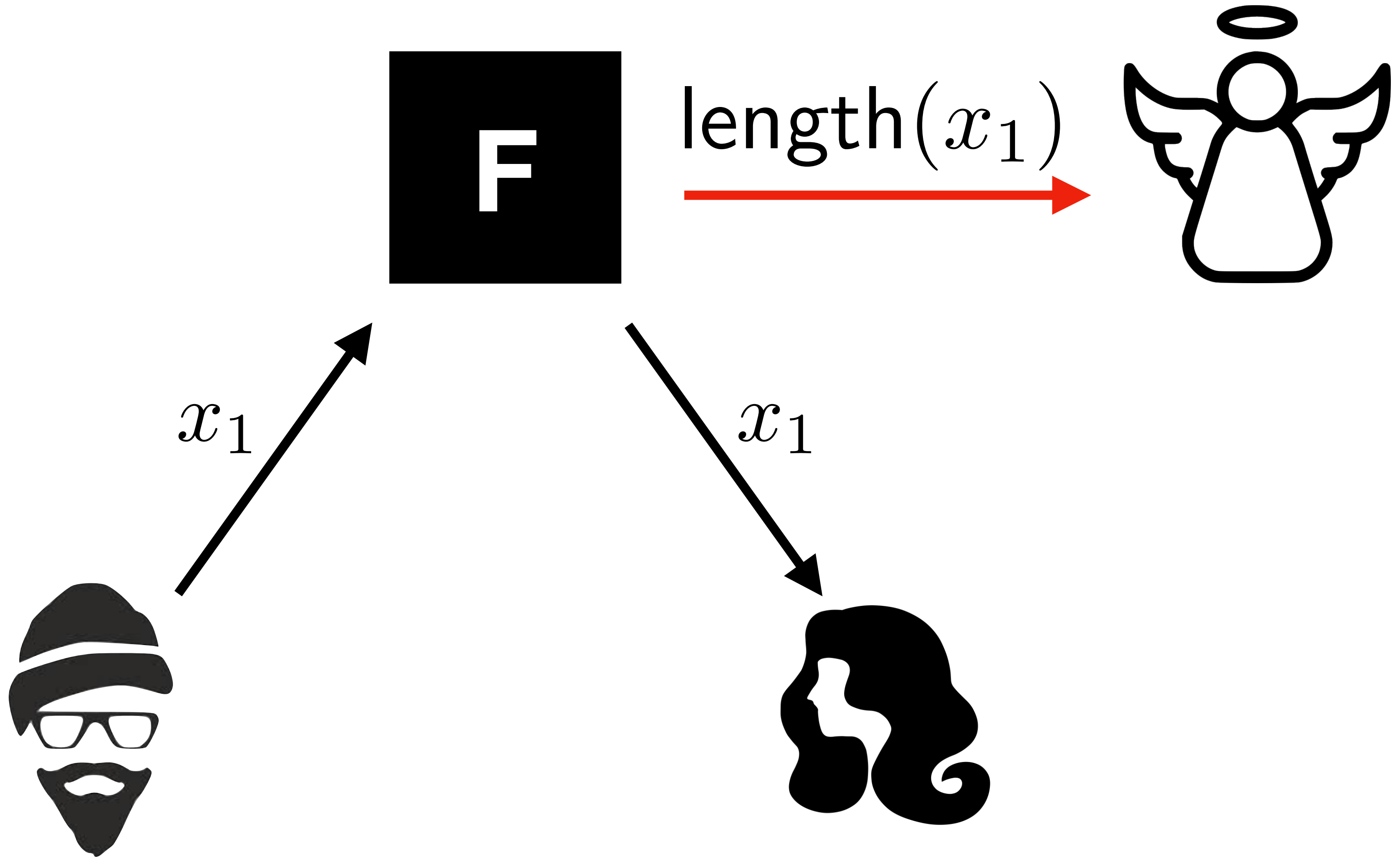
Real World



Solution:

- Allow this leakage
- Emulate the ideal adversary

Ideal World



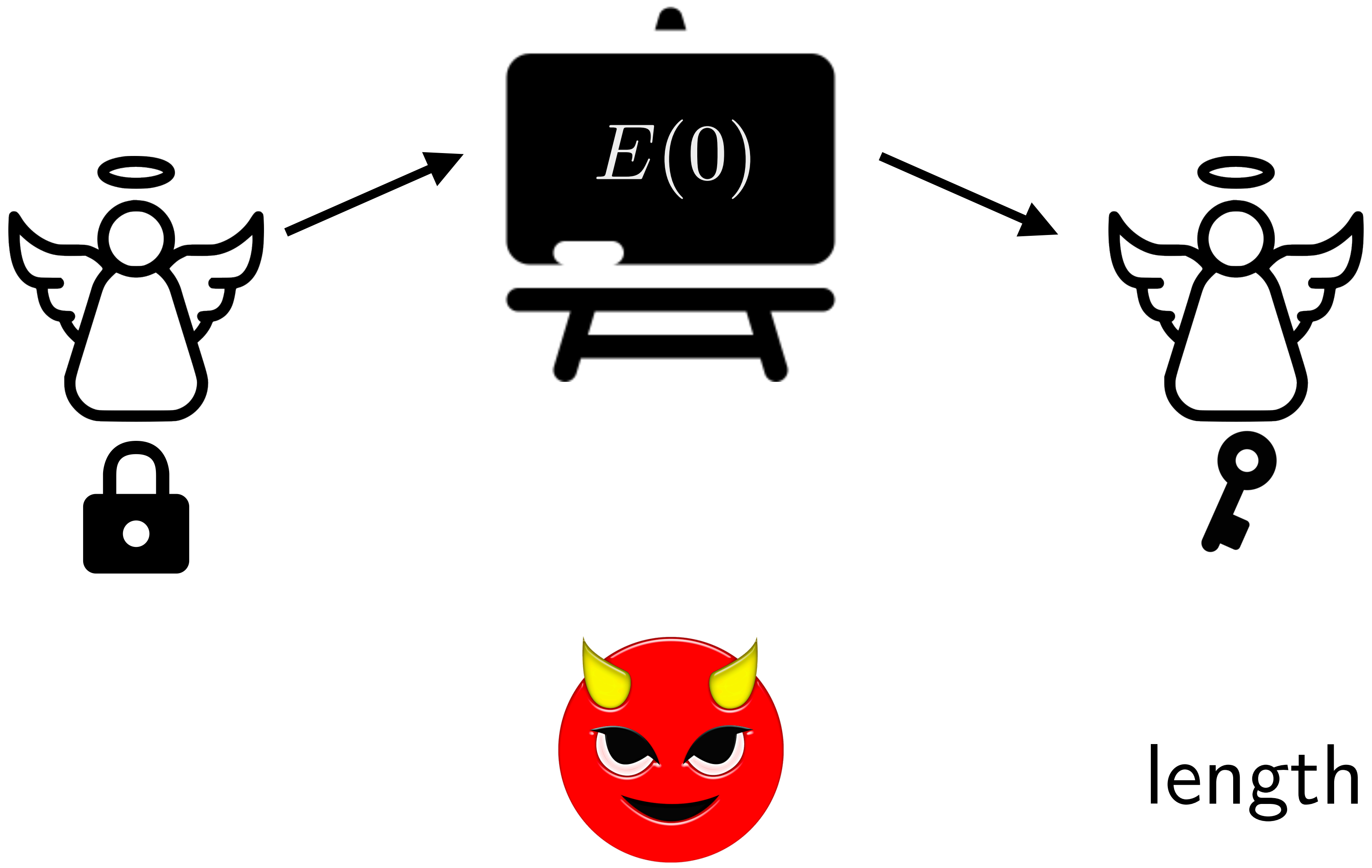
Simulation

View: 🔒 $, E(0)$ $\approx$ **View:** 🔒 $, E(x_1)$

⚠ **Cannot work: the plaintexts do not have the same size!**

Simple Example: Secure Communication

Real World

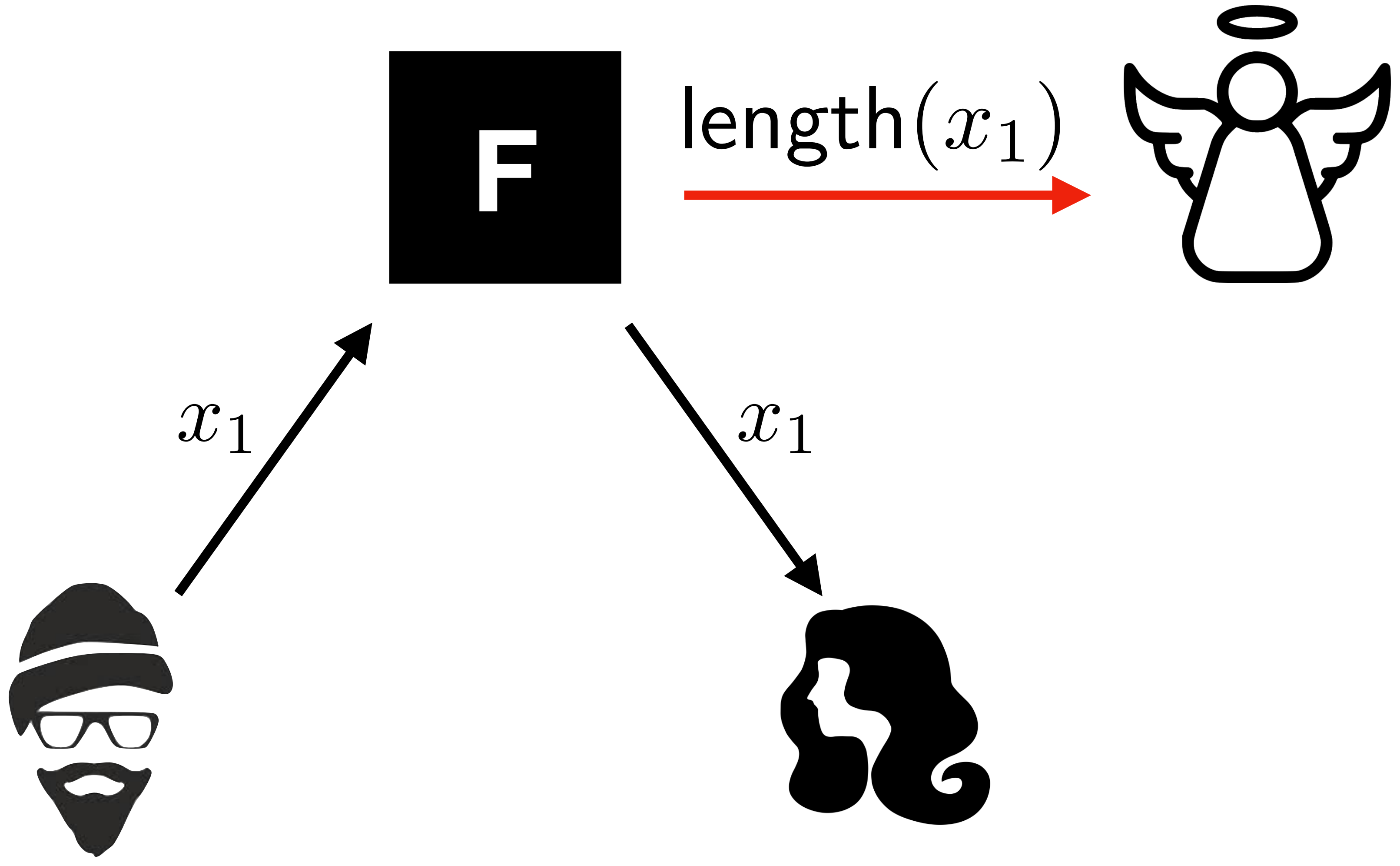


length(x_1) zeroes

Solution:

- Allow this leakage
- Emulate the ideal adversary
- Use E(0…0)

Ideal World



Simulation

View: 🔒 $, E(0 \cdots 0)$ $\approx$ **View:** 🔒 $, E(x_1)$

👍 **Problem solved**

Simple Example II: Oblivious Transfer

Sender

(s_0, s_1)

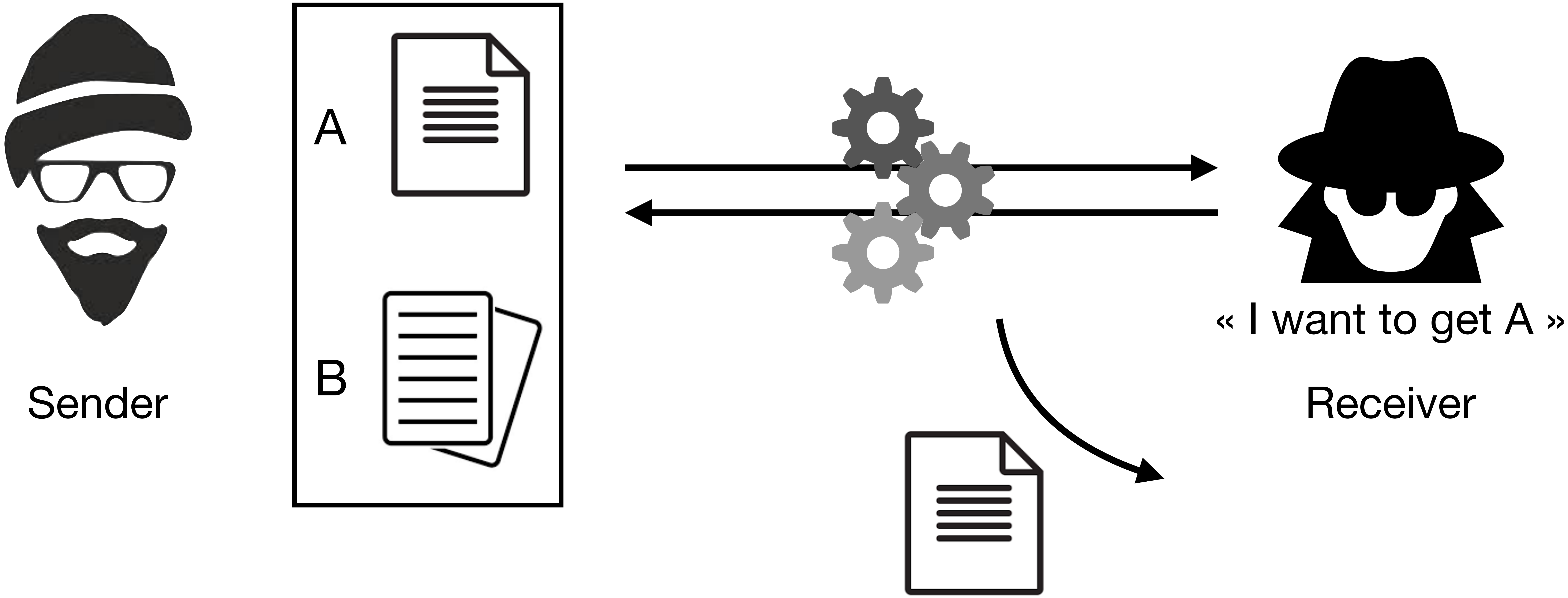
Goal :

- The receiver learns s_b
- The sender learns nothing about b
- The receiver learns nothing about s_{1-b}

Receiver

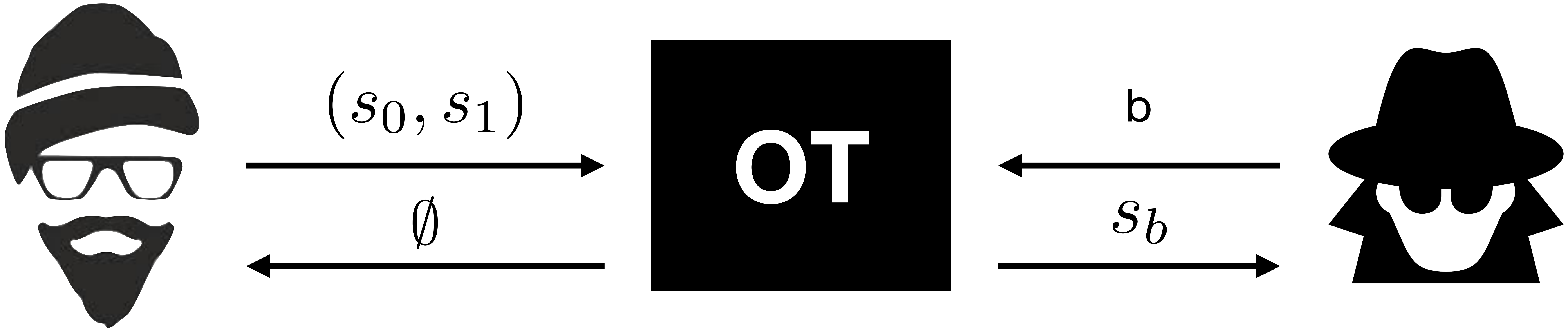
Selection bit b

Simple Example II: Oblivious Transfer



A (minimalistic) version of *symmetrically private* download from a database held by a server: the client wants to retrieve an item (but does not want to reveal which one), and the server wants to keep all other items private.

Simple Example II: Oblivious Transfer



(s_0, s_1)

$\emptyset$

OT

b

s_b

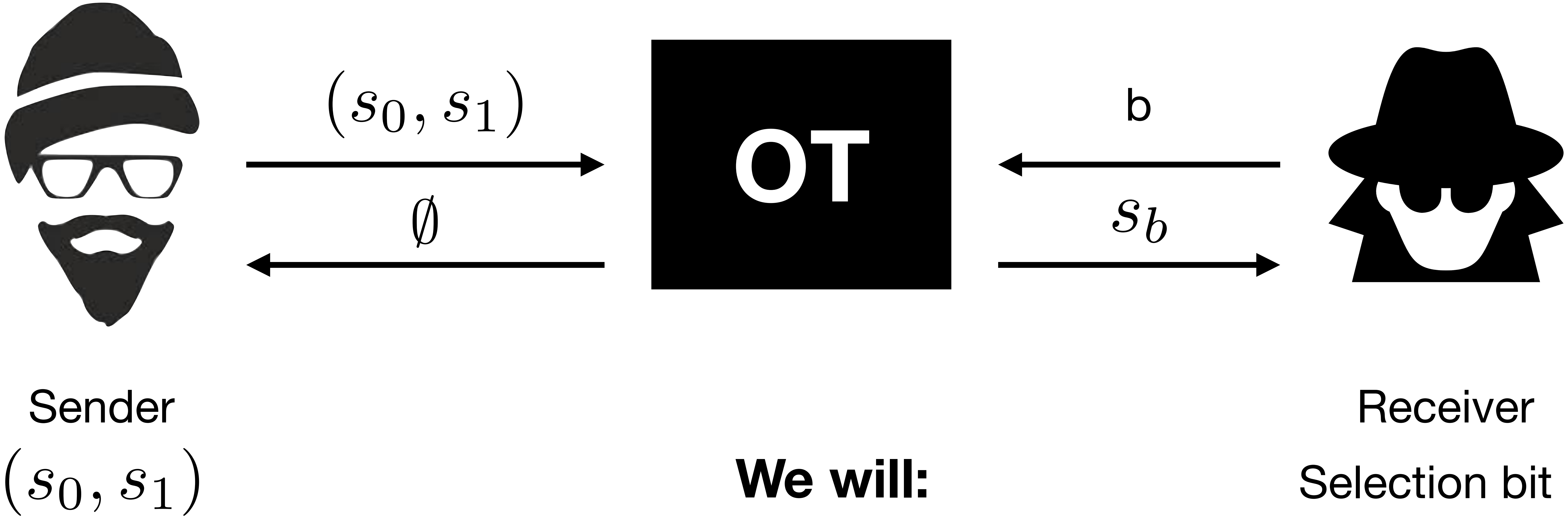
Sender

(s_0, s_1)

Receiver

Selection bit b

Simple Example II: Oblivious Transfer



We will:

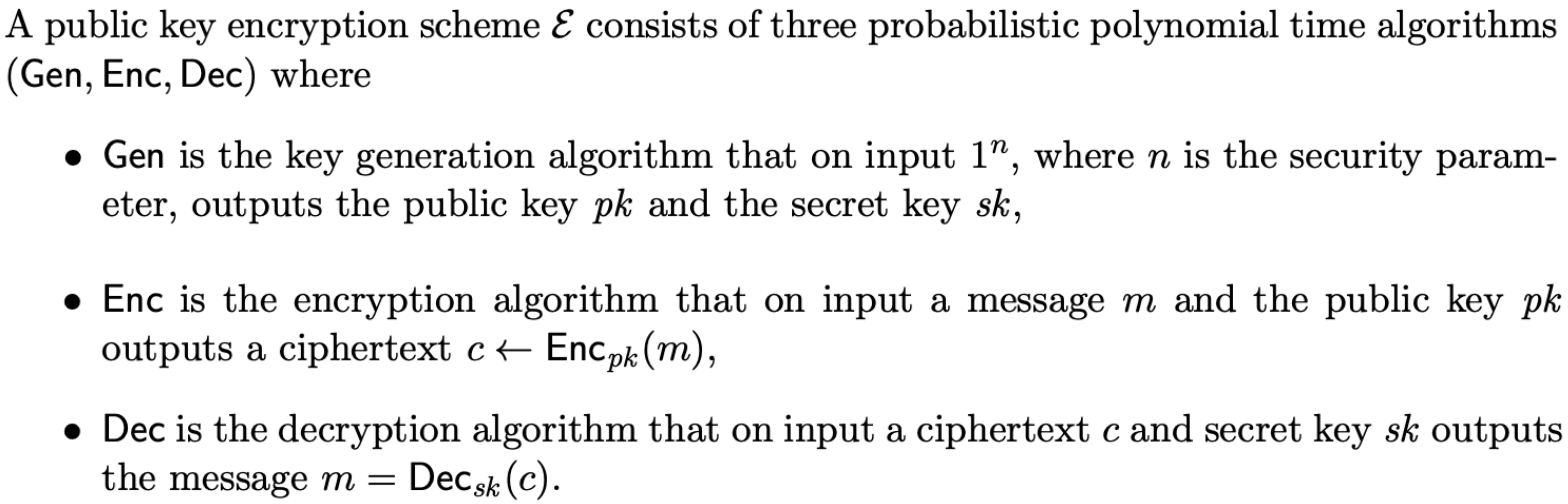
- Provide a full construction of OT, starting from an IND-CPA encryption scheme satisfying additional special properties
- Formally prove that the construction is secure.

The following closely follows the lecture notes of Jonathan Kat:

https://www.cs.umd.edu/~jkatz/gradcrypto2/f13/lecture3.pdf

Simple Example II: Oblivious Transfer

Reminder:

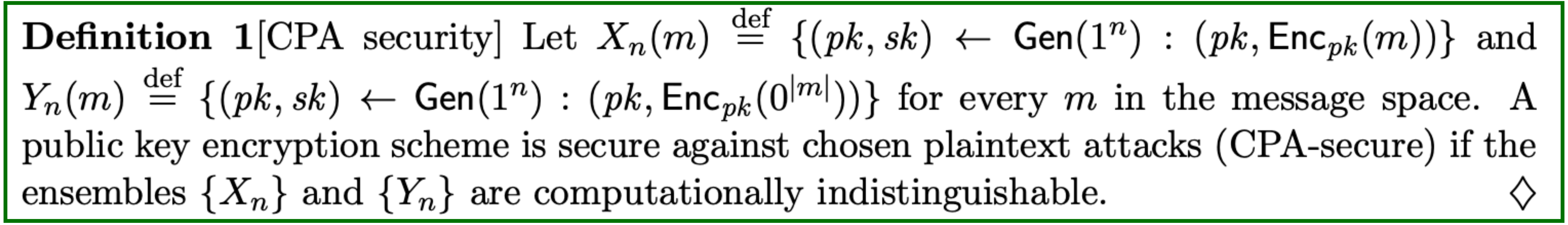


A public key encryption scheme $\mathcal{E}$ consists of three probabilistic polynomial time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where

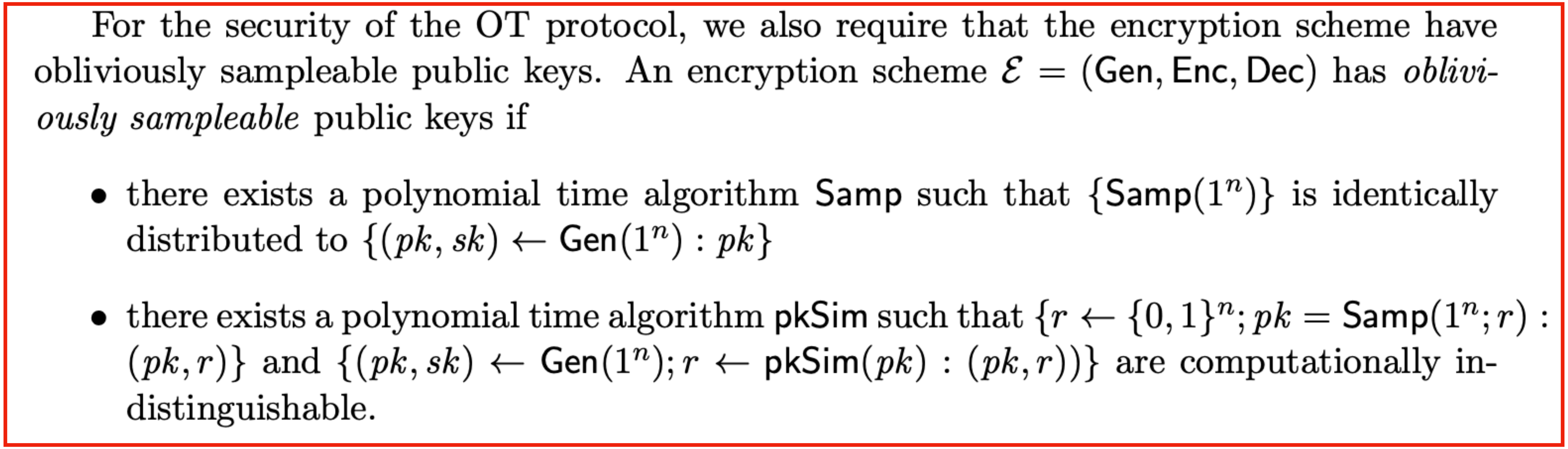
- Gen is the key generation algorithm that on input 1^n, where n is the security parameter, outputs the public key pk and the secret key sk,
- Enc is the encryption algorithm that on input a message m and the public key pk outputs a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$,
- Dec is the decryption algorithm that on input a ciphertext c and secret key sk outputs the message $m = \mathsf{Dec}_{sk}(c)$.

Simple Example II: Oblivious Transfer

Standard (though you might have seen another - equivalent - formulation)



Definition 1[CPA security] Let $X_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(m))\}$ and $Y_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(0^{|m|}))\}$ for every m in the message space. A public key encryption scheme is secure against chosen plaintext attacks (CPA-secure) if the ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable. $\diamondsuit$



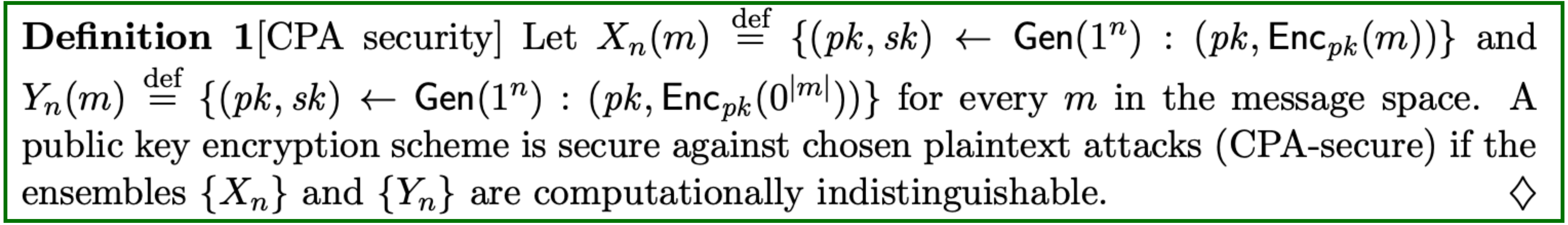
For the security of the OT protocol, we also require that the encryption scheme have obliviously sampleable public keys. An encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *obliviously sampleable* public keys if

- there exists a polynomial time algorithm Samp such that $\{\mathsf{Samp}(1^n)\}$ is identically distributed to $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : pk\}$
- there exists a polynomial time algorithm pkSim such that $\{r \leftarrow \{0,1\}^n; pk = \mathsf{Samp}(1^n; r) : (pk, r)\}$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \mathsf{pkSim}(pk) : (pk, r))\}$ are computationally indistinguishable.

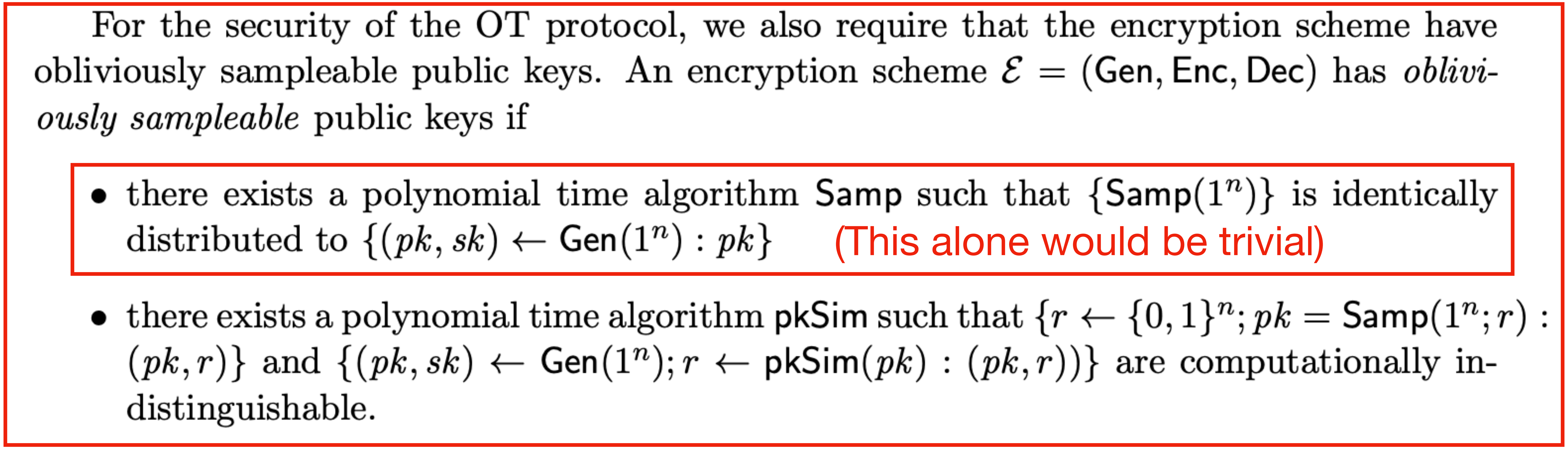
New

Simple Example II: Oblivious Transfer

Standard (though you might have seen another - equivalent - formulation)



Definition 1[CPA security] Let $X_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(m))\}$ and $Y_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(0^{|m|}))\}$ for every m in the message space. A public key encryption scheme is secure against chosen plaintext attacks (CPA-secure) if the ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable. ◊



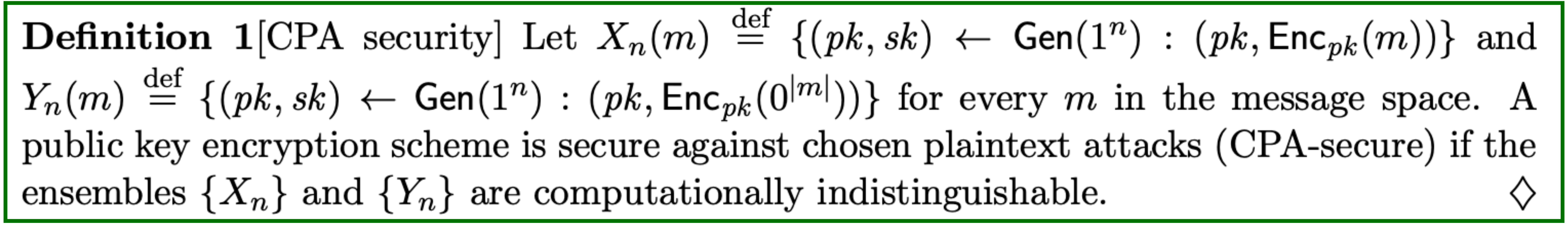
For the security of the OT protocol, we also require that the encryption scheme have obliviously sampleable public keys. An encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *obliviously sampleable* public keys if

- there exists a polynomial time algorithm Samp such that $\{\mathsf{Samp}(1^n)\}$ is identically distributed to $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : pk\}$ (This alone would be trivial)
- there exists a polynomial time algorithm pkSim such that $\{r \leftarrow \{0,1\}^n; pk = \mathsf{Samp}(1^n; r) : (pk, r)\}$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \mathsf{pkSim}(pk) : (pk, r))\}$ are computationally indistinguishable.

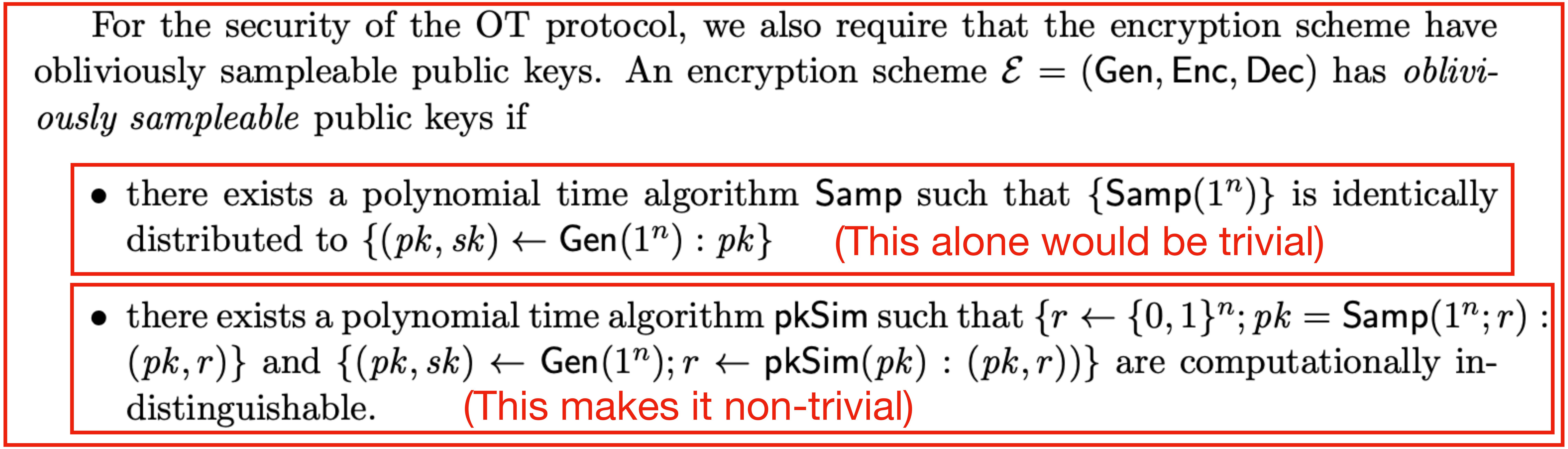
New

Simple Example II: Oblivious Transfer

Standard (though you might have seen another - equivalent - formulation)



Definition 1[CPA security] Let $X_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(m))\}$ and $Y_n(m) \overset{\text{def}}{=} \{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : (pk, \mathsf{Enc}_{pk}(0^{|m|}))\}$ for every m in the message space. A public key encryption scheme is secure against chosen plaintext attacks (CPA-secure) if the ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable. $\diamond$



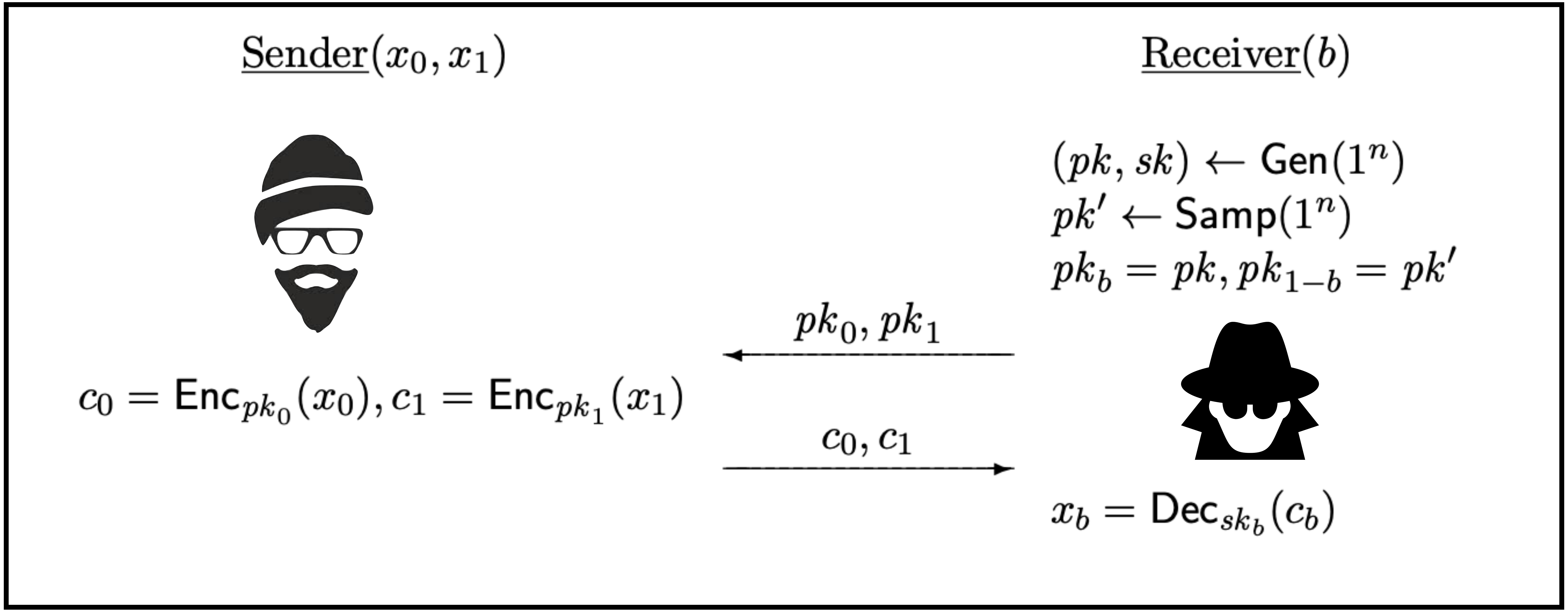
For the security of the OT protocol, we also require that the encryption scheme have obliviously sampleable public keys. An encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has *obliviously sampleable* public keys if

- there exists a polynomial time algorithm Samp such that $\{\mathsf{Samp}(1^n)\}$ is identically distributed to $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : pk\}$ (This alone would be trivial)
- there exists a polynomial time algorithm pkSim such that $\{r \leftarrow \{0,1\}^n; pk = \mathsf{Samp}(1^n; r) : (pk, r)\}$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \mathsf{pkSim}(pk) : (pk, r))\}$ are computationally indistinguishable. (This makes it non-trivial)

New

Simple Example II: Oblivious Transfer

OT Protocol



$\underline{\text{Sender}}(x_0, x_1)$

$\underline{\text{Receiver}}(b)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$

$pk' \leftarrow \mathsf{Samp}(1^n)$

$pk_b = pk, pk_{1-b} = pk'$

$\xleftarrow{pk_0, pk_1}$

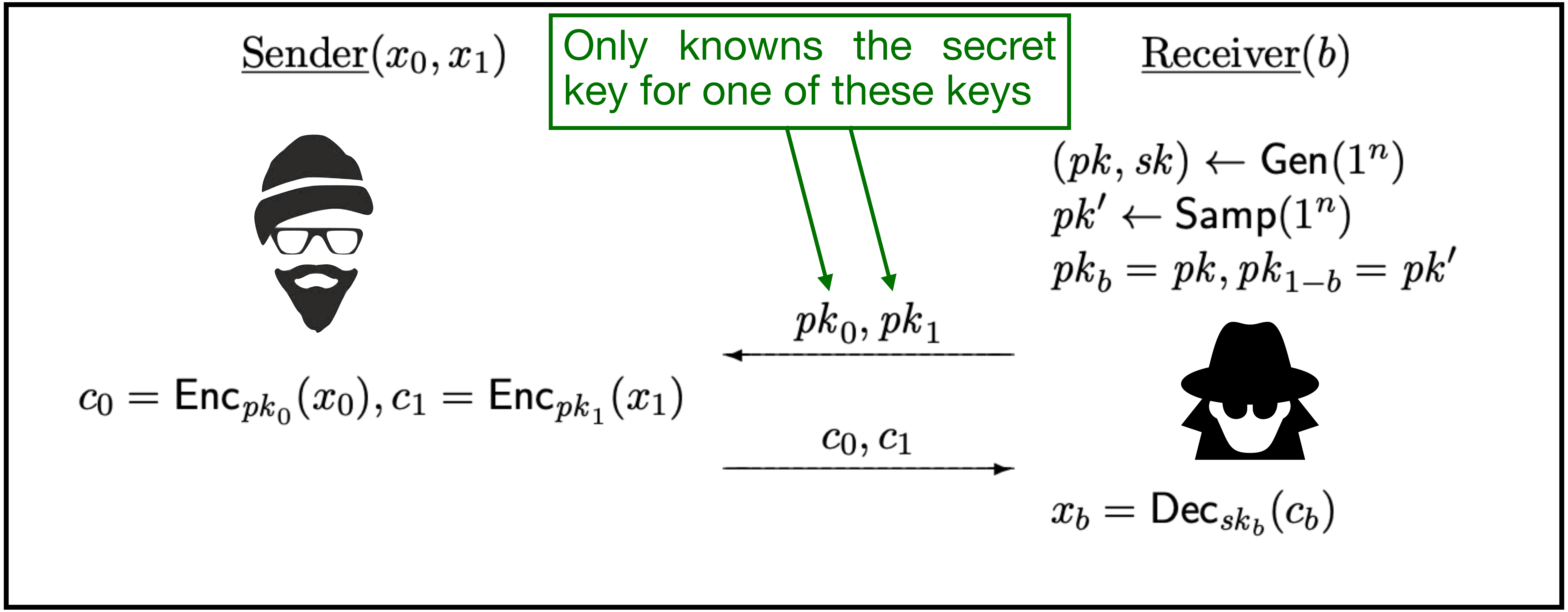
$c_0 = \mathsf{Enc}_{pk_0}(x_0), c_1 = \mathsf{Enc}_{pk_1}(x_1)$

$\xrightarrow{c_0, c_1}$

$x_b = \mathsf{Dec}_{sk_b}(c_b)$

Simple Example II: Oblivious Transfer

OT Protocol



Only knowns the secret key for one of these keys

$\underline{\text{Sender}}(x_0, x_1)$

$\underline{\text{Receiver}}(b)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
$pk' \leftarrow \mathsf{Samp}(1^n)$
$pk_b = pk, pk_{1-b} = pk'$

$$pk_0, pk_1$$

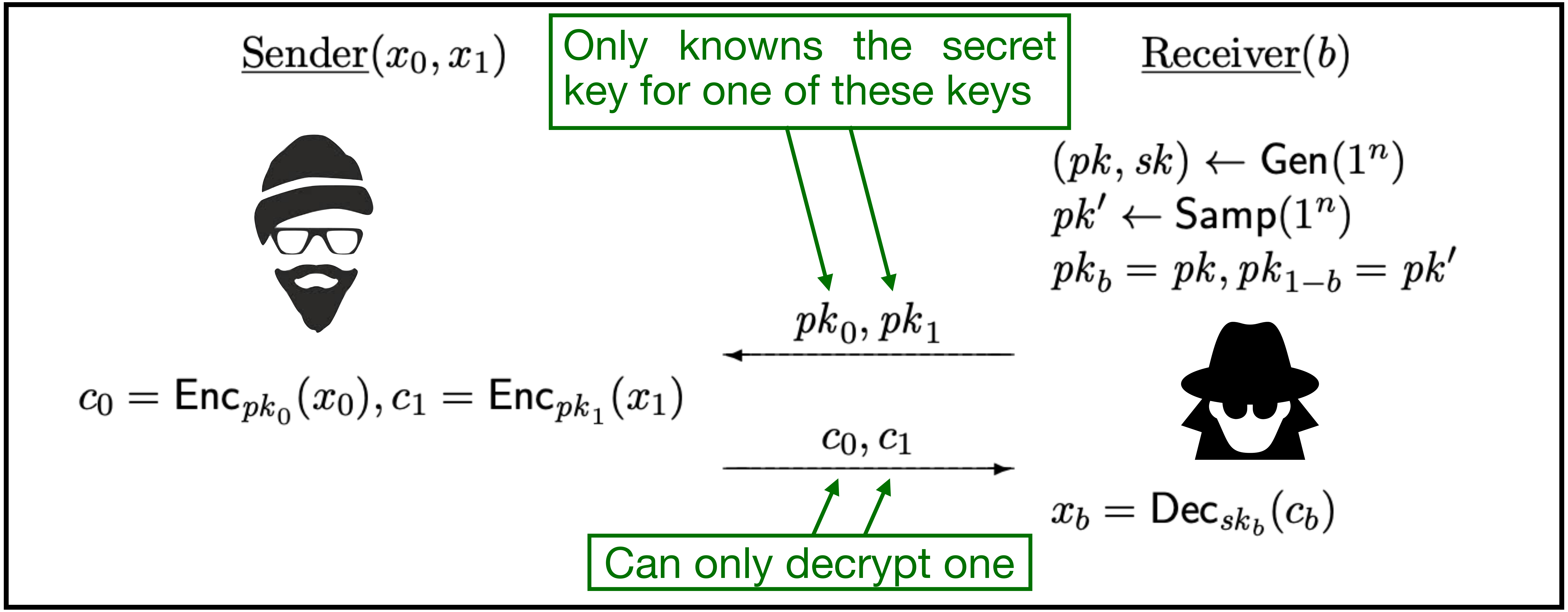
$c_0 = \mathsf{Enc}_{pk_0}(x_0), c_1 = \mathsf{Enc}_{pk_1}(x_1)$

$$c_0, c_1$$

$x_b = \mathsf{Dec}_{sk_b}(c_b)$

Simple Example II: Oblivious Transfer

OT Protocol



$\underline{\text{Sender}}(x_0, x_1)$

Only knowns the secret key for one of these keys

$\underline{\text{Receiver}}(b)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$

$pk' \leftarrow \mathsf{Samp}(1^n)$

$pk_b = pk, pk_{1-b} = pk'$

pk_0, pk_1

$c_0 = \mathsf{Enc}_{pk_0}(x_0), c_1 = \mathsf{Enc}_{pk_1}(x_1)$

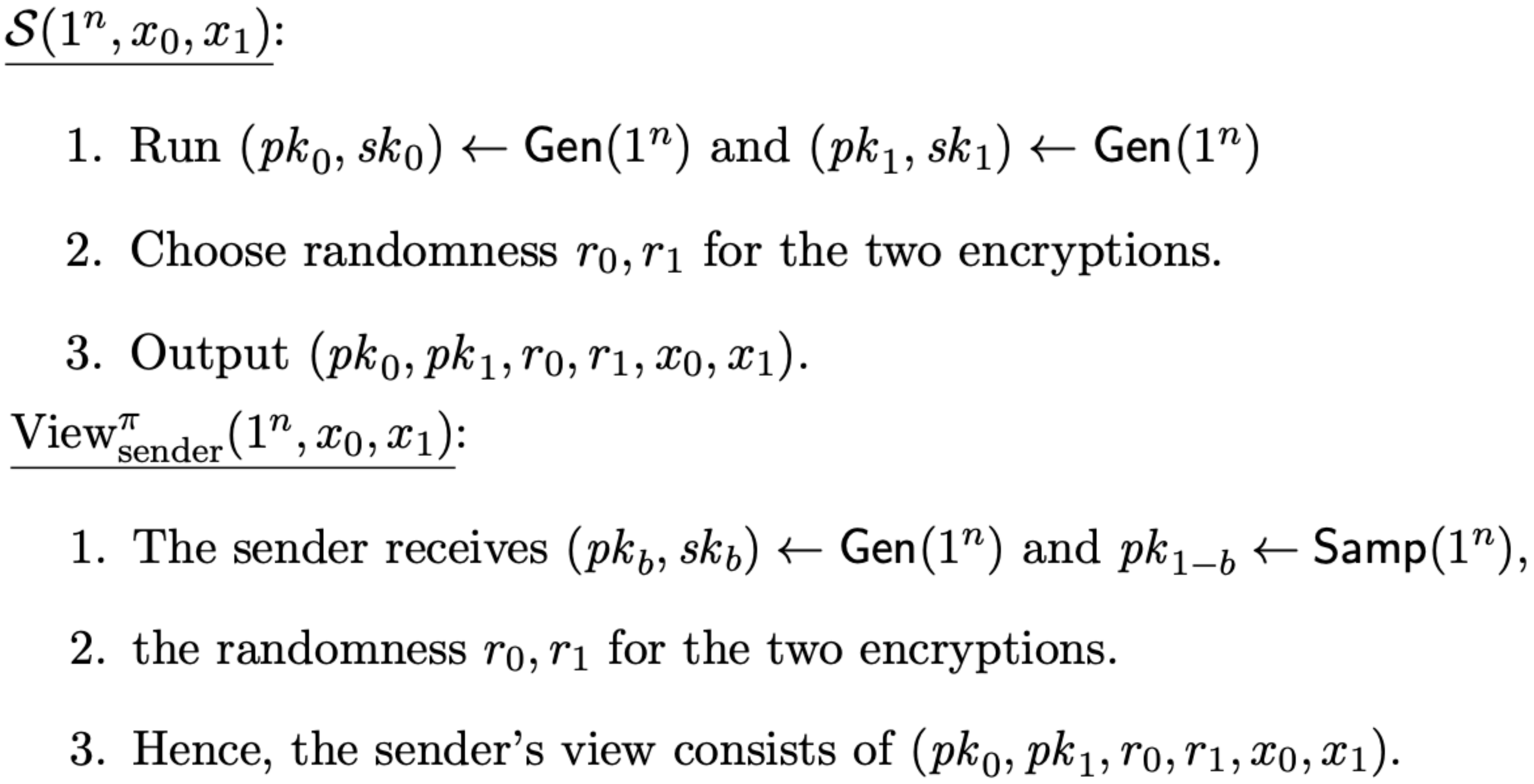
c_0, c_1

$x_b = \mathsf{Dec}_{sk_b}(c_b)$

Can only decrypt one

Oblivious Transfer - Security Analysis

Security Against the Sender



$\underline{\mathcal{S}(1^n, x_0, x_1)}$:

1. Run $(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^n)$ and $(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^n)$
2. Choose randomness r_0, r_1 for the two encryptions.
3. Output $(pk_0, pk_1, r_0, r_1, x_0, x_1)$.

$\underline{\mathrm{View}^{\pi}_{\mathrm{sender}}(1^n, x_0, x_1)}$:

1. The sender receives $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n)$ and $pk_{1-b} \leftarrow \mathsf{Samp}(1^n)$,
2. the randomness r_0, r_1 for the two encryptions.
3. Hence, the sender's view consists of $(pk_0, pk_1, r_0, r_1, x_0, x_1)$.

Oblivious Transfer - Security Analysis

Security Against the Sender

$\underline{\mathcal{S}(1^n, x_0, x_1)}$:

1. Run $(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^n)$ and $(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^n)$
2. Choose randomness r_0, r_1 for the two encryptions.
3. Output $(pk_0, pk_1, r_0, r_1, x_0, x_1)$.

$\approx$

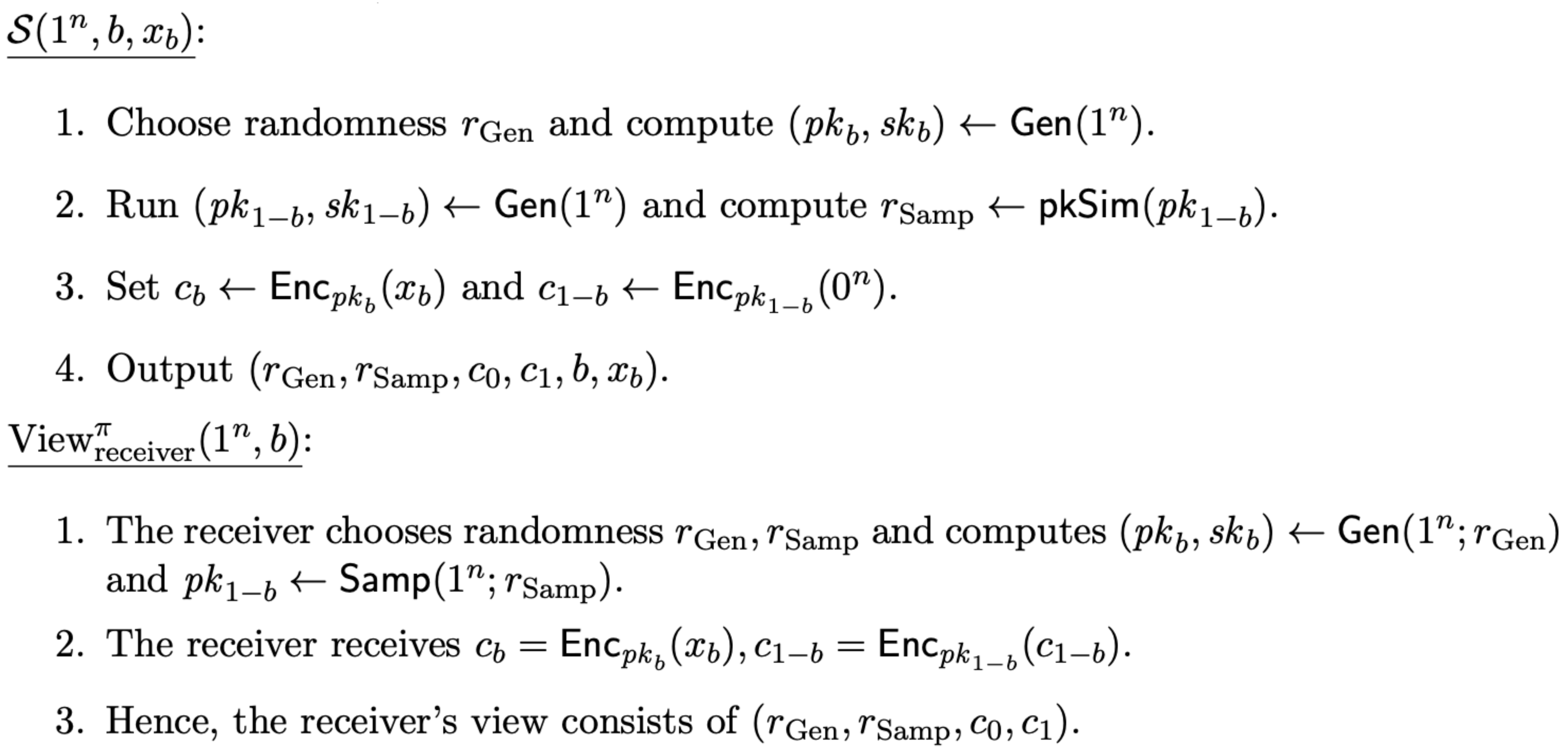
$\underline{\mathrm{View}^{\pi}_{\mathrm{sender}}(1^n, x_0, x_1)}$:

1. The sender receives $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n)$ and $pk_{1-b} \leftarrow \mathsf{Samp}(1^n)$,
2. the randomness r_0, r_1 for the two encryptions.
3. Hence, the sender's view consists of $(pk_0, pk_1, r_0, r_1, x_0, x_1)$.

$\mathsf{Samp}(1^n)$ and $\mathsf{Gen}(1^n)$ are identically distributed.

Oblivious Transfer - Security Analysis

Security Against the Receiver



$\underline{\mathcal{S}(1^n, b, x_b)}$:

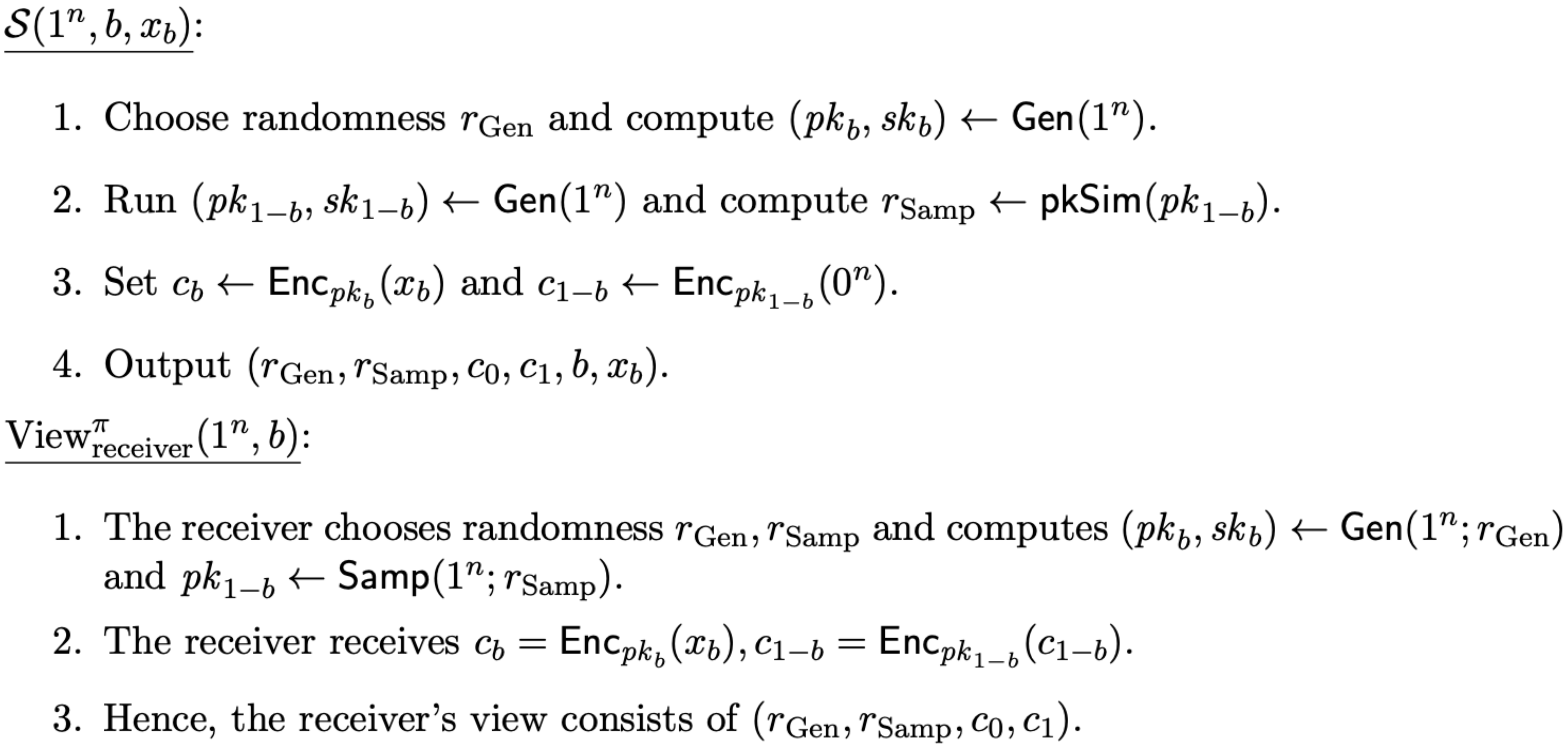
1. Choose randomness r_{Gen} and compute $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n)$.
2. Run $(pk_{1-b}, sk_{1-b}) \leftarrow \mathsf{Gen}(1^n)$ and compute $r_{\text{Samp}} \leftarrow \mathsf{pkSim}(pk_{1-b})$.
3. Set $c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b)$ and $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(0^n)$.
4. Output $(r_{\text{Gen}}, r_{\text{Samp}}, c_0, c_1, b, x_b)$.

$\underline{\text{View}^{\pi}_{\text{receiver}}(1^n, b)}$:

1. The receiver chooses randomness $r_{\text{Gen}}, r_{\text{Samp}}$ and computes $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n; r_{\text{Gen}})$ and $pk_{1-b} \leftarrow \mathsf{Samp}(1^n; r_{\text{Samp}})$.
2. The receiver receives $c_b = \mathsf{Enc}_{pk_b}(x_b), c_{1-b} = \mathsf{Enc}_{pk_{1-b}}(c_{1-b})$.
3. Hence, the receiver's view consists of $(r_{\text{Gen}}, r_{\text{Samp}}, c_0, c_1)$.

Oblivious Transfer - Security Analysis

Security Against the Receiver



$\underline{\mathcal{S}(1^n, b, x_b)}$:

1. Choose randomness r_{Gen} and compute $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n)$.
2. Run $(pk_{1-b}, sk_{1-b}) \leftarrow \mathsf{Gen}(1^n)$ and compute $r_{\text{Samp}} \leftarrow \mathsf{pkSim}(pk_{1-b})$.
3. Set $c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b)$ and $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(0^n)$.
4. Output $(r_{\text{Gen}}, r_{\text{Samp}}, c_0, c_1, b, x_b)$.

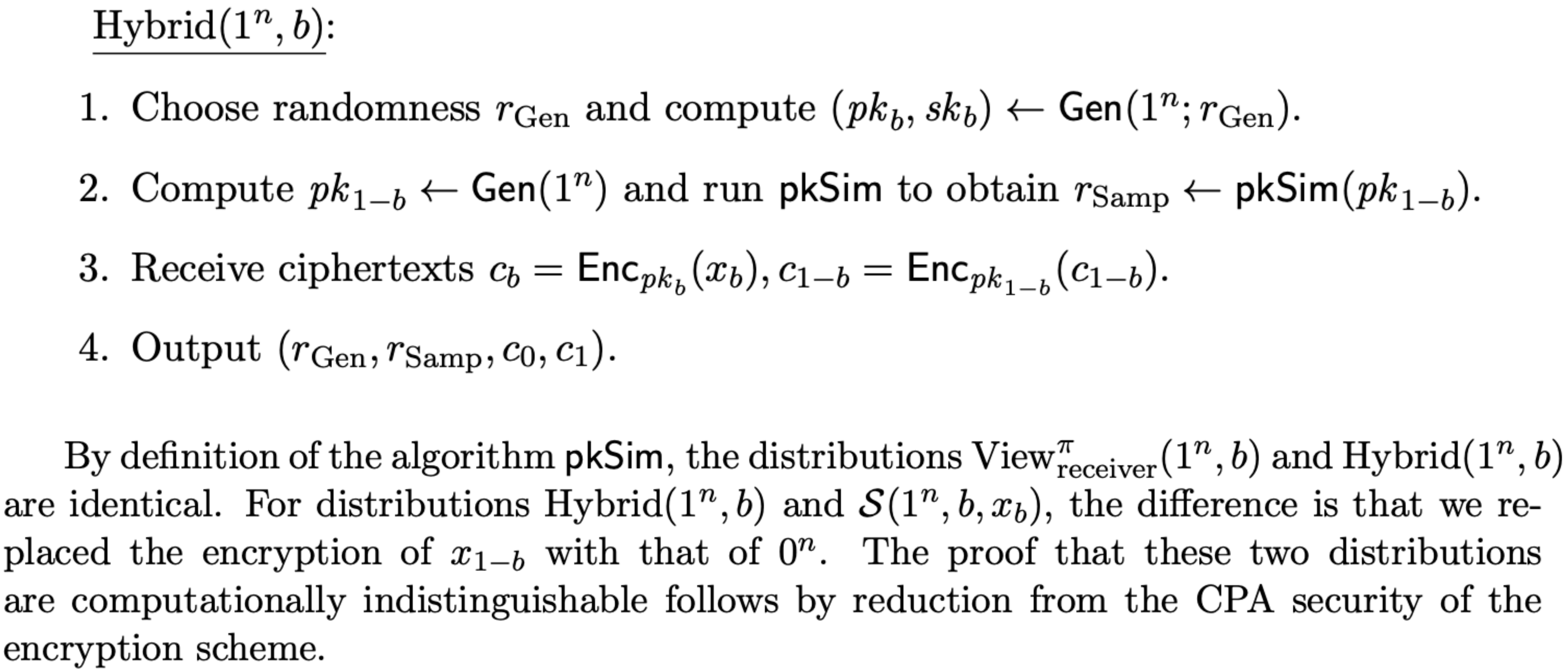
$\approx$

$\underline{\text{View}^{\pi}_{\text{receiver}}(1^n, b)}$:

1. The receiver chooses randomness $r_{\text{Gen}}, r_{\text{Samp}}$ and computes $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n; r_{\text{Gen}})$ and $pk_{1-b} \leftarrow \mathsf{Samp}(1^n; r_{\text{Samp}})$.
2. The receiver receives $c_b = \mathsf{Enc}_{pk_b}(x_b), c_{1-b} = \mathsf{Enc}_{pk_{1-b}}(c_{1-b})$.
3. Hence, the receiver's view consists of $(r_{\text{Gen}}, r_{\text{Samp}}, c_0, c_1)$.

Oblivious Transfer - Security Analysis

Security Against the Receiver



Hybrid$(1^n, b)$:

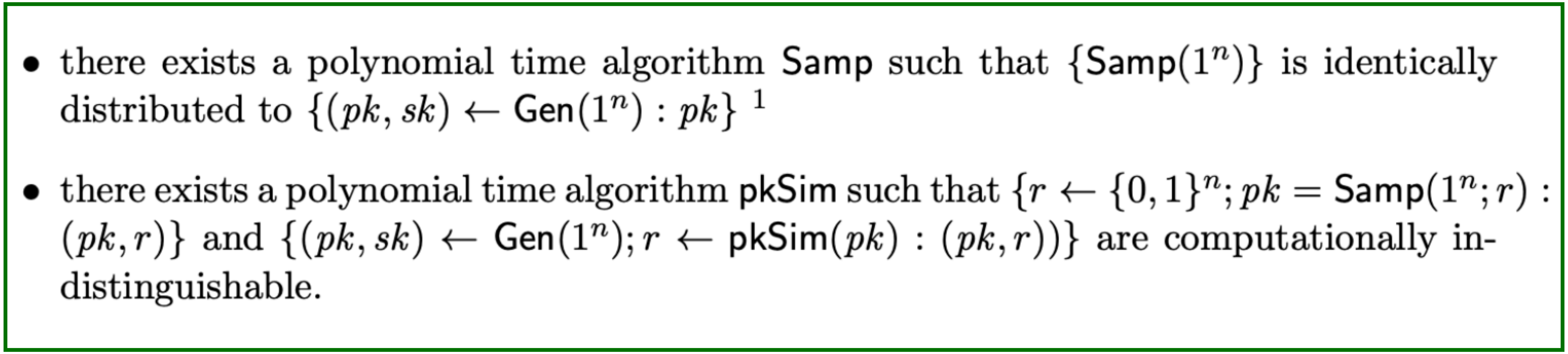
1. Choose randomness r_{Gen} and compute $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^n; r_{\text{Gen}})$.
2. Compute $pk_{1-b} \leftarrow \mathsf{Gen}(1^n)$ and run pkSim to obtain $r_{\text{Samp}} \leftarrow \mathsf{pkSim}(pk_{1-b})$.
3. Receive ciphertexts $c_b = \mathsf{Enc}_{pk_b}(x_b), c_{1-b} = \mathsf{Enc}_{pk_{1-b}}(c_{1-b})$.
4. Output $(r_{\text{Gen}}, r_{\text{Samp}}, c_0, c_1)$.

By definition of the algorithm pkSim, the distributions $\text{View}^{\pi}_{\text{receiver}}(1^n, b)$ and Hybrid$(1^n, b)$ are identical. For distributions Hybrid$(1^n, b)$ and $\mathcal{S}(1^n, b, x_b)$, the difference is that we replaced the encryption of x_{1-b} with that of 0^n. The proof that these two distributions are computationally indistinguishable follows by reduction from the CPA security of the encryption scheme.

Home Exercise

Prove that ElGamal satisfies the *obliviously samplable keys* requirement

Reminder:



- there exists a polynomial time algorithm Samp such that $\{\mathsf{Samp}(1^n)\}$ is identically distributed to $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n) : pk\}$ [1]
- there exists a polynomial time algorithm pkSim such that $\{r \leftarrow \{0,1\}^n; pk = \mathsf{Samp}(1^n; r) : (pk, r)\}$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^n); r \leftarrow \mathsf{pkSim}(pk) : (pk, r))\}$ are computationally indistinguishable.

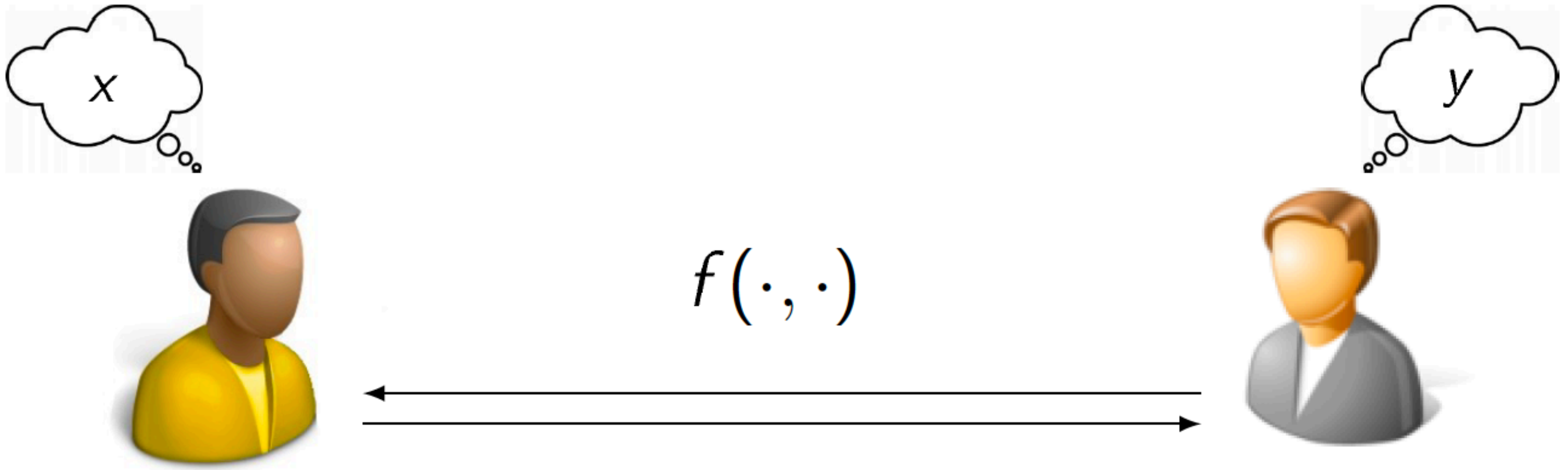
Two-Party Secure Computation for *All* Functions

Until now, we only addressed special cases of secure computation, for very specific, restricted (two party) functionalities: *secure communication* and *oblivious transfer*.

However, a beautiful result of Yao, from 1986, showed that the existence of (private-key) encryption schemes, together with a protocol for oblivious transfer, as we just constructed, suffices to securely compute *all functions* in the two-party setting.

In the following, we will prove this result.

Two-Party Secure Computation for *All* Functions



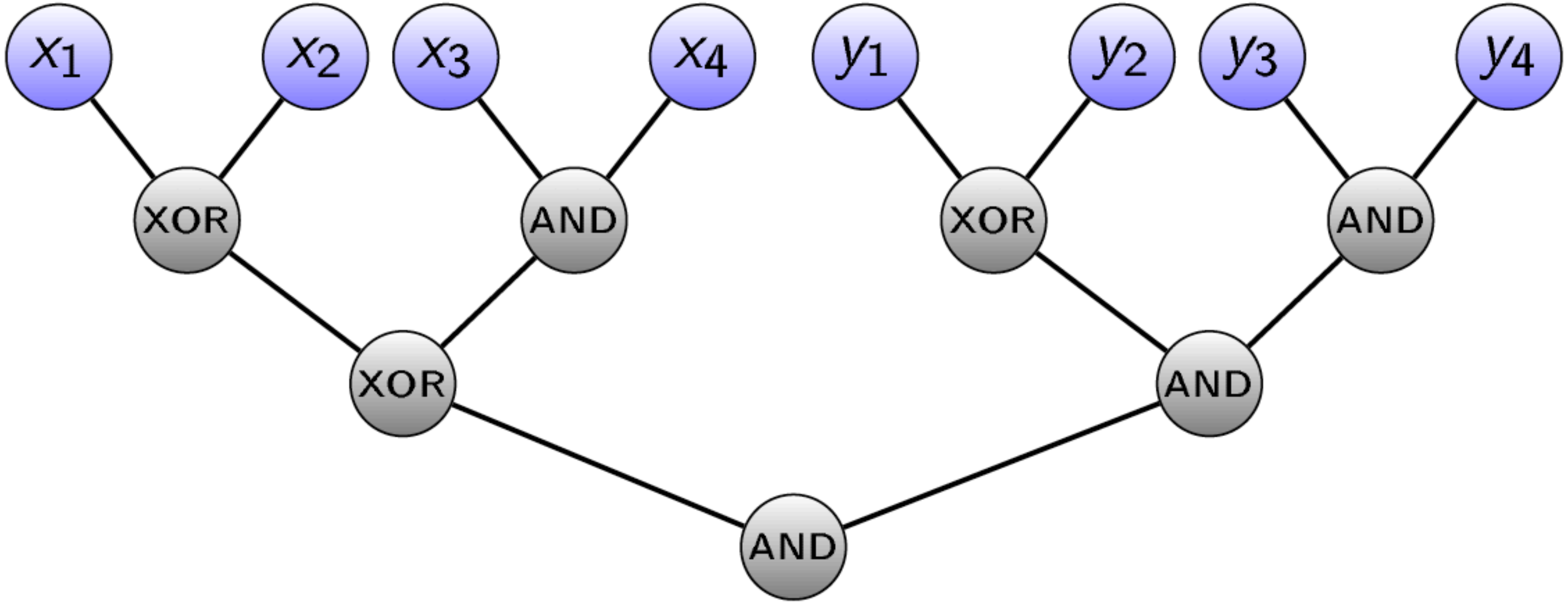
Goal: $f(x, y)$

Idea: represent f as a boolean circuit

Building Block I: Boolean Circuits

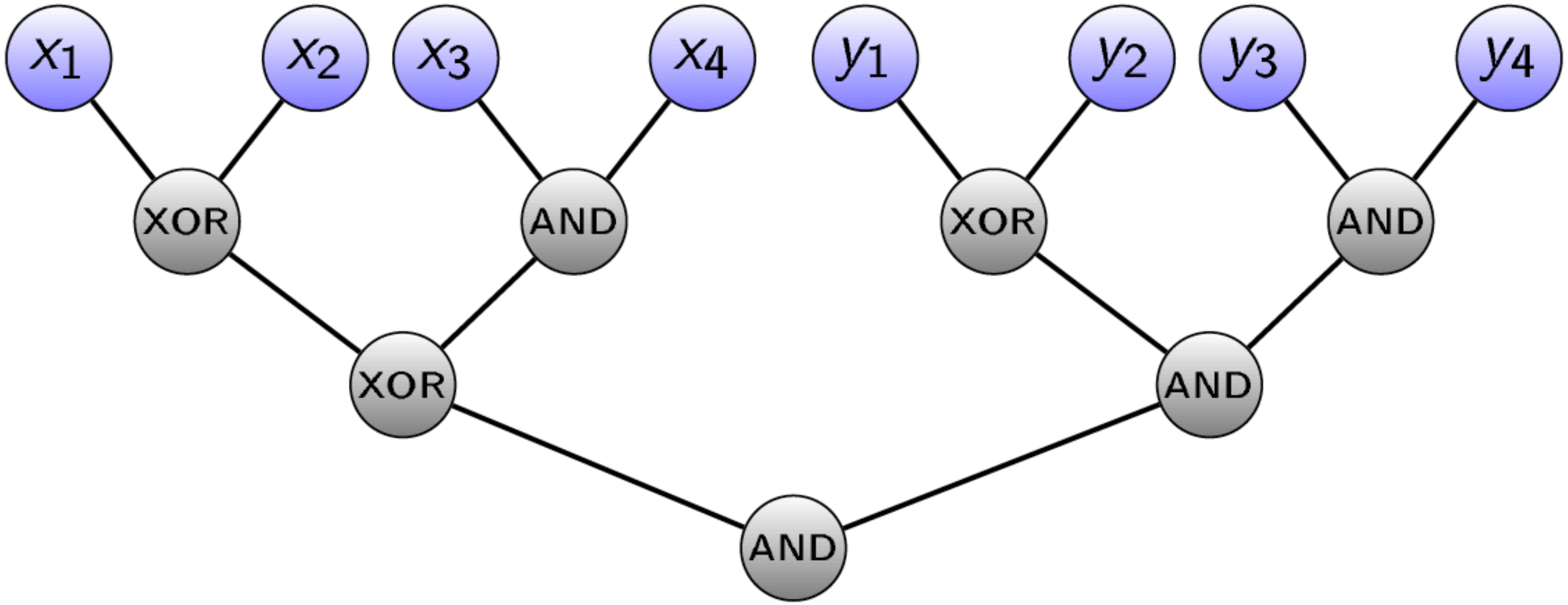
Claim: any polytime-computable function can be computed by a poly size boolean circuit over the {XOR, AND} bases.

Proof: that's how your computer does it.



Building Block I: Boolean Circuits

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



Building Block II: Symmetric Encryption

We let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with the following properties:

- KeyGen generates a key K
- $\mathsf{Enc}_K(m) \rightarrow c$ generates a random encryption of the plaintext m
- $\mathsf{Dec}_K(c)$ returns m if $c = \mathsf{Enc}_K(m)$
- A decryption of a ciphertext c with a *wrong* key K' returns « error » (hence, it reveals that a wrong key was used)

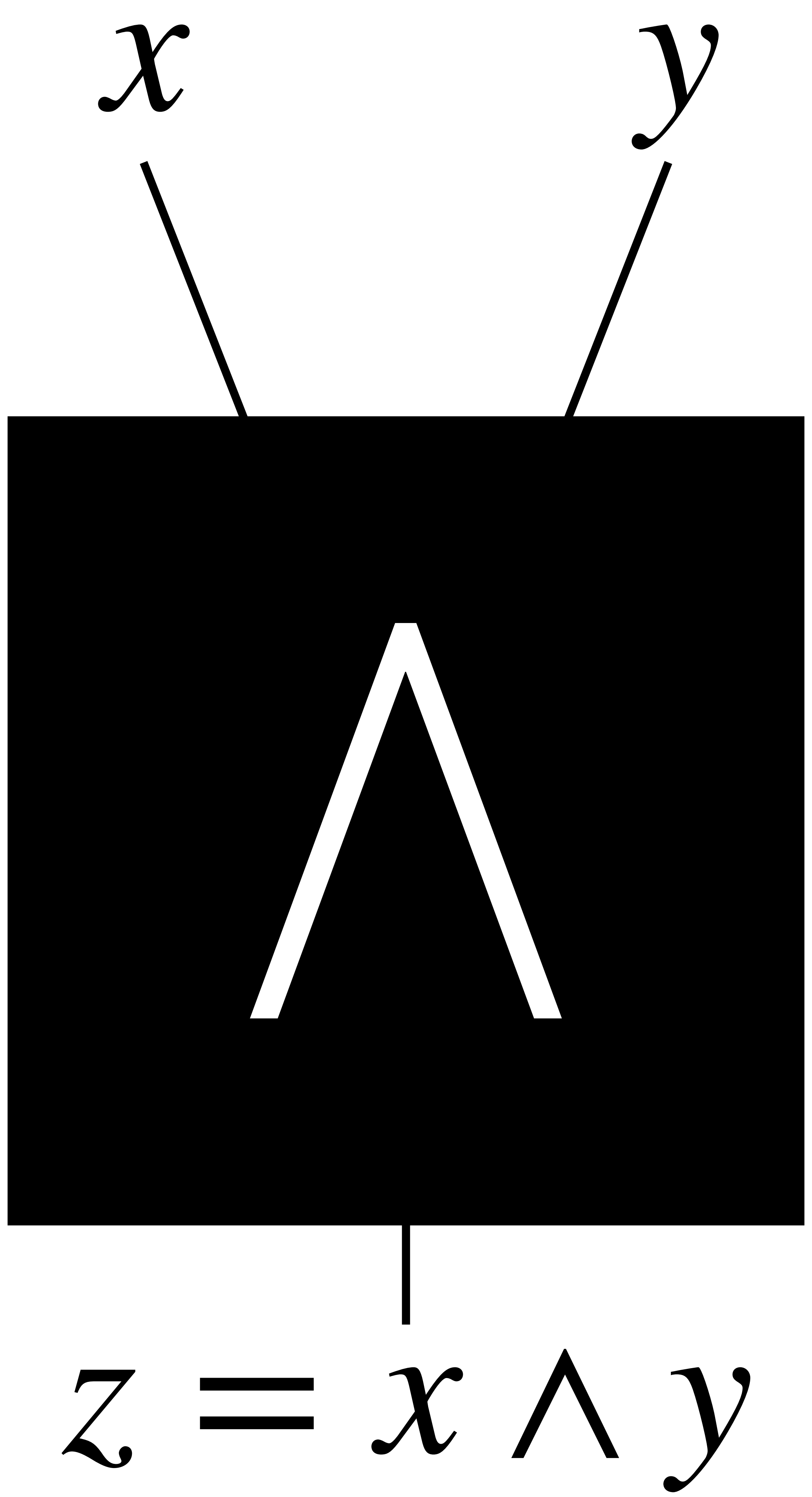
Building Block II: Symmetric Encryption

We let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with the following properties:

- KeyGen generates a key K
- $\mathsf{Enc}_K(m) \rightarrow c$ generates a random encryption of the plaintext m
- $\mathsf{Dec}_K(c)$ returns m if $c = \mathsf{Enc}_K(m)$
- A decryption of a ciphertext c with a *wrong* key K' returns « error » (hence, it reveals that a wrong key was used)

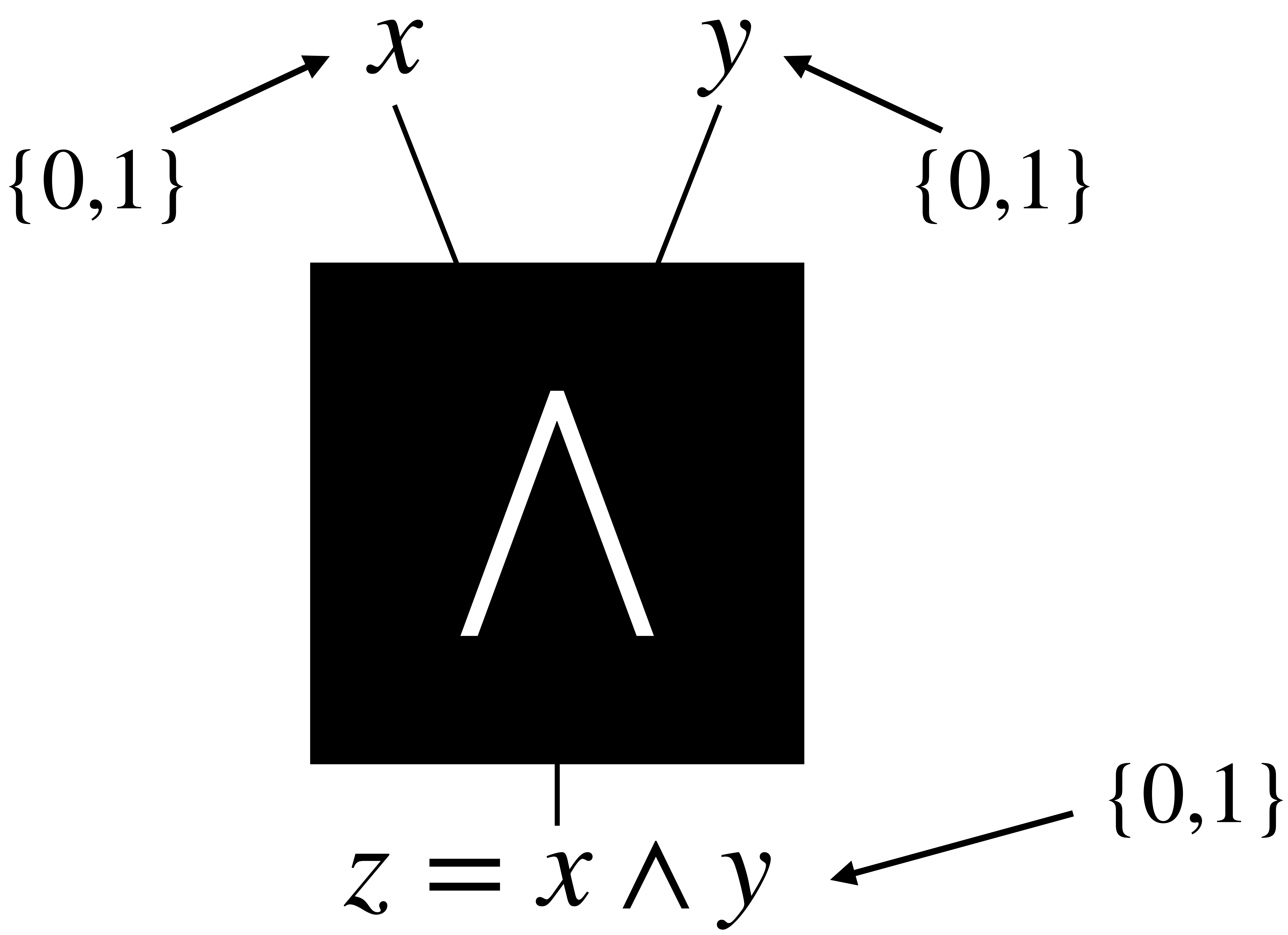
Building Block II: Symmetric Encryption

We will use this encryption scheme to « encrypt » logical gates.



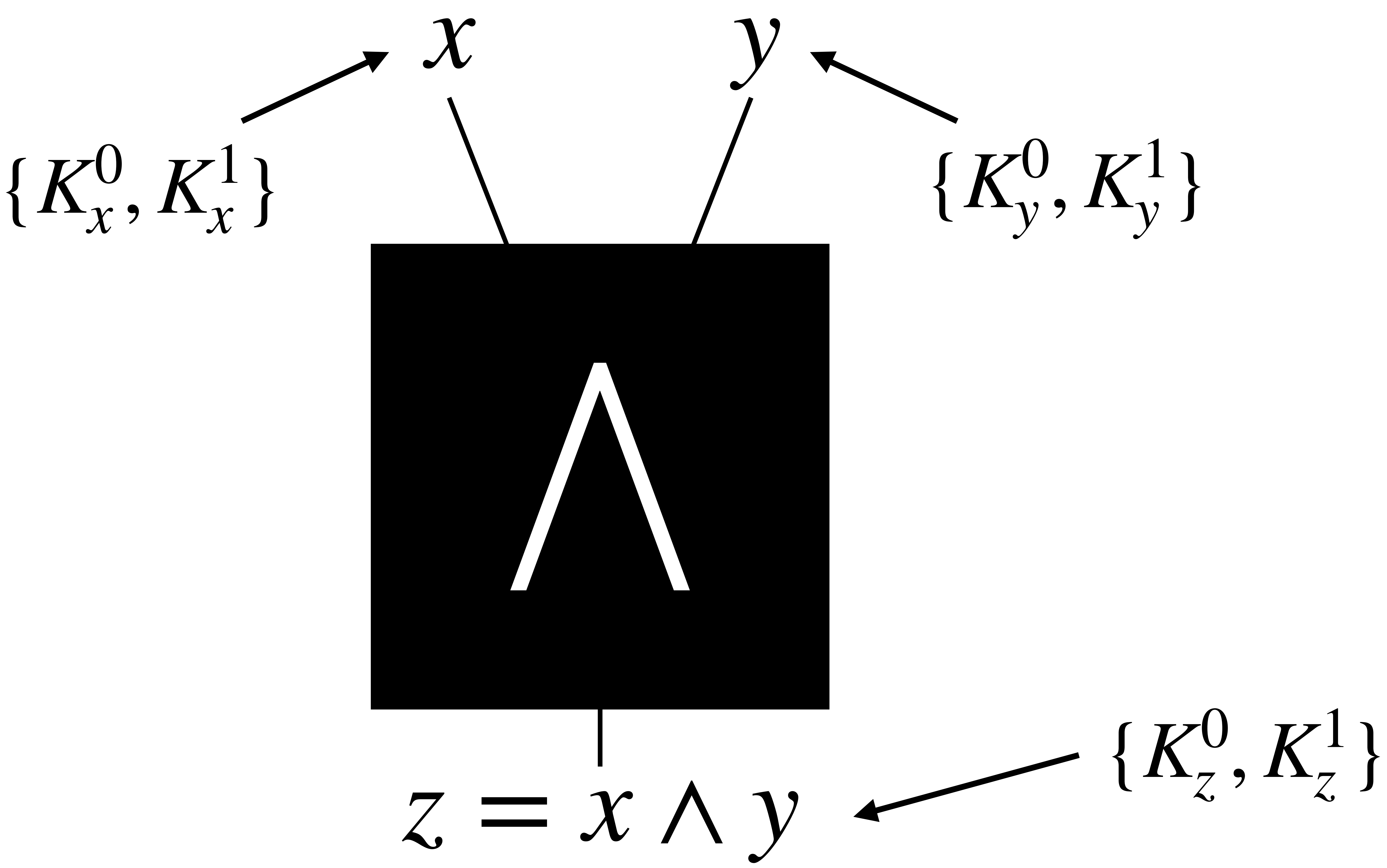
Building Block II: Symmetric Encryption

The inputs and outputs are bits, but we will « represent » them using random keys, to hide their true value:



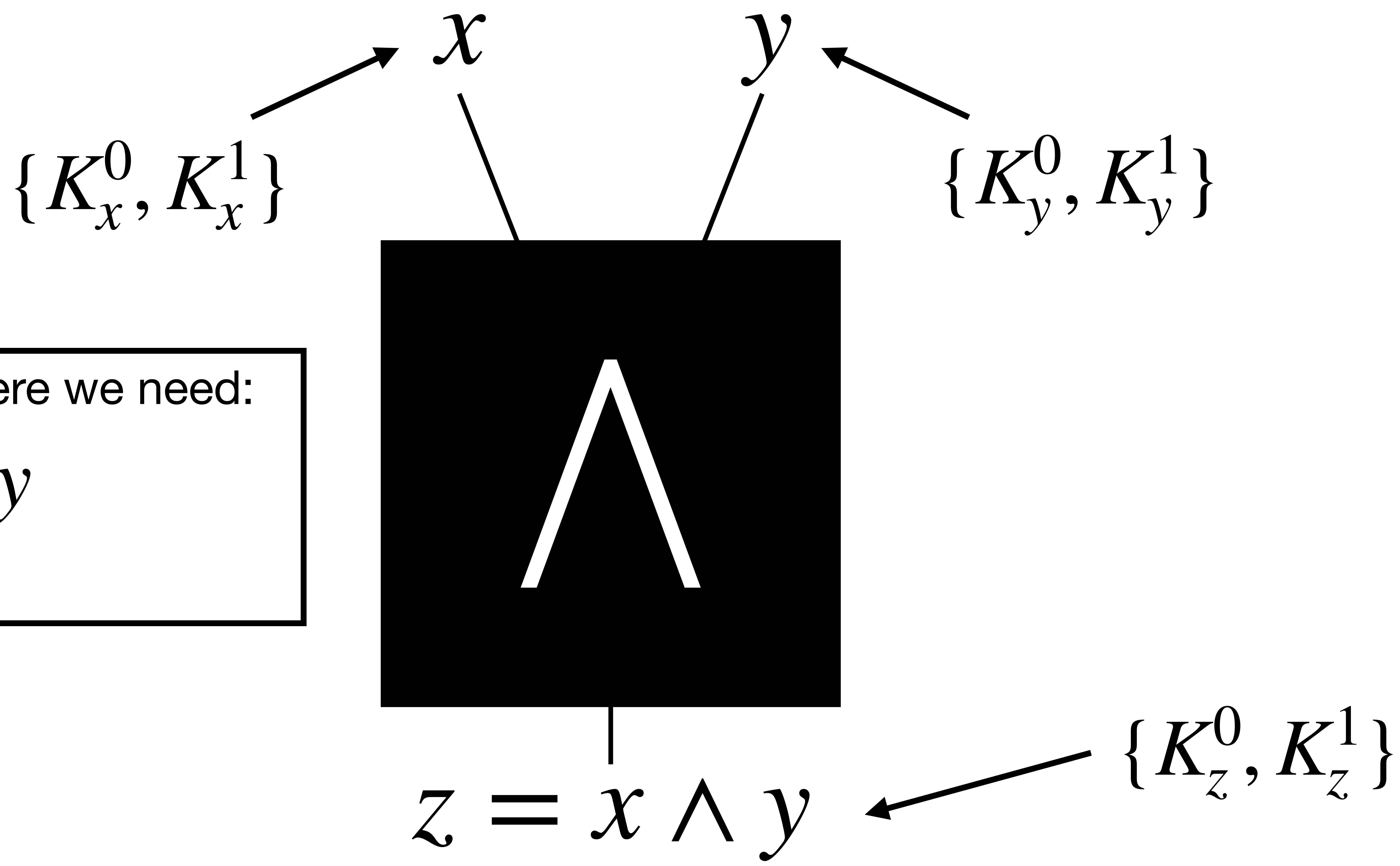
Building Block II: Symmetric Encryption

The inputs and outputs are bits, but we will « represent » them using random keys, to hide their true value:



Building Block II: Symmetric Encryption

The inputs and outputs are bits, but we will « represent » them using random keys, to hide their true value:



x

y

$\{K_x^0, K_x^1\}$

$\{K_y^0, K_y^1\}$

$\wedge$

$z = x \wedge y$

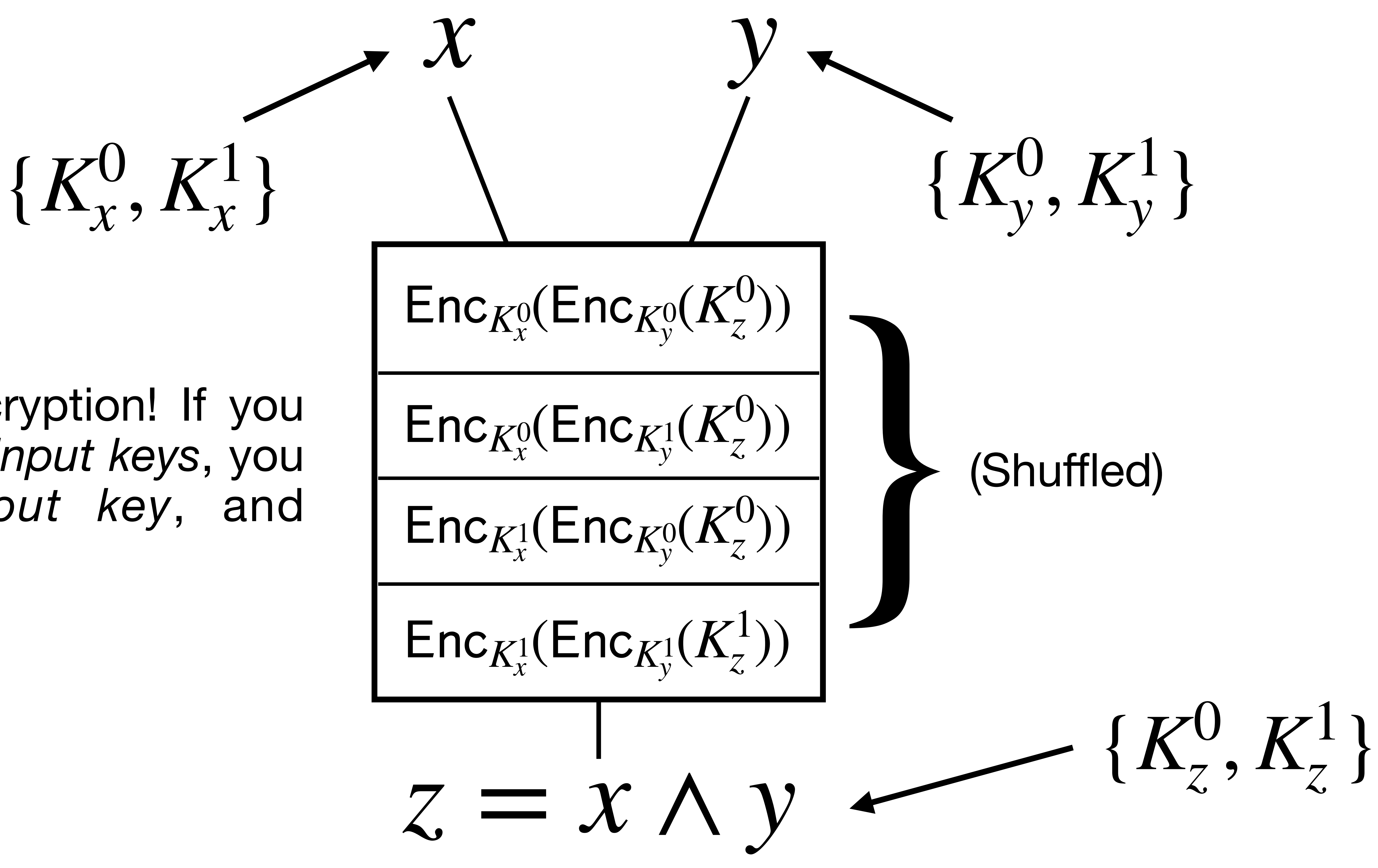
$\{K_z^0, K_z^1\}$

To stay consistant, here we need:

$$K_z^{x \wedge y}$$

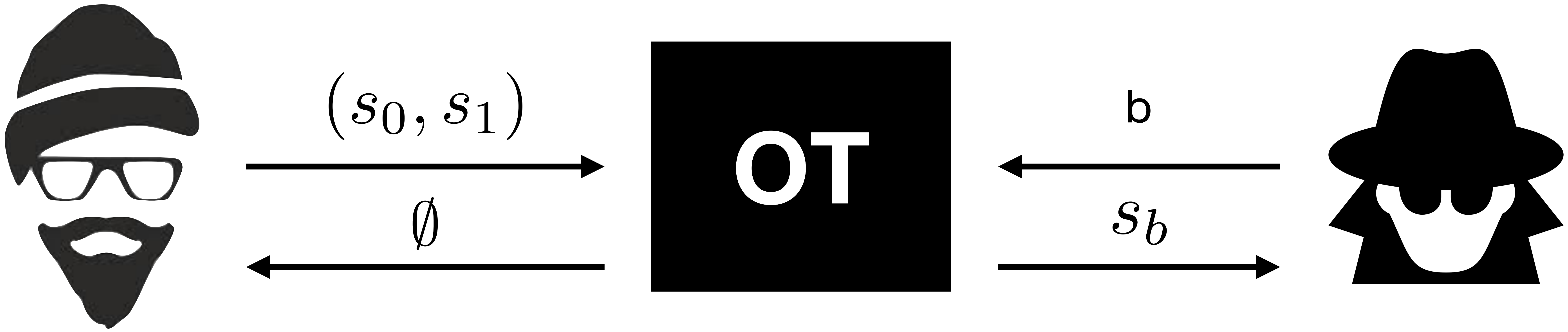
Building Block II: Symmetric Encryption

The inputs and outputs are bits, but we will « represent » them using random keys, to hide their true value:


x
y
$\{K_x^0, K_x^1\}$
$\{K_y^0, K_y^1\}$
$\mathsf{Enc}_{K_x^0}(\mathsf{Enc}_{K_y^0}(K_z^0))$
$\mathsf{Enc}_{K_x^0}(\mathsf{Enc}_{K_y^1}(K_z^0))$
$\mathsf{Enc}_{K_x^1}(\mathsf{Enc}_{K_y^0}(K_z^0))$
$\mathsf{Enc}_{K_x^1}(\mathsf{Enc}_{K_y^1}(K_z^1))$
(Shuffled)
$z = x \wedge y$
$\{K_z^0, K_z^1\}$

Key idea: double encryption! If you have exactly the right *input keys*, you learn the right *output key*, and nothing more.

Building Block III: Oblivious Transfer



(s_0, s_1)

$\emptyset$

OT

b

s_b

Sender

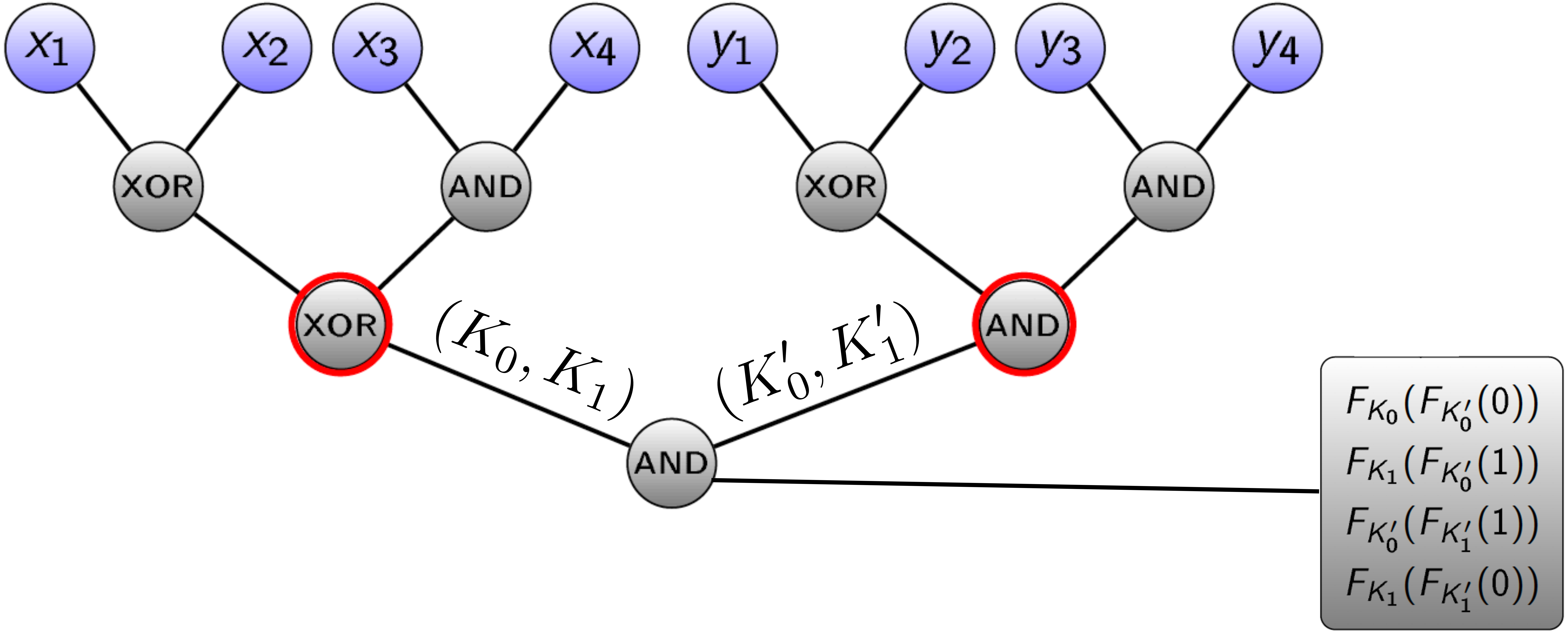
(s_0, s_1)

Receiver

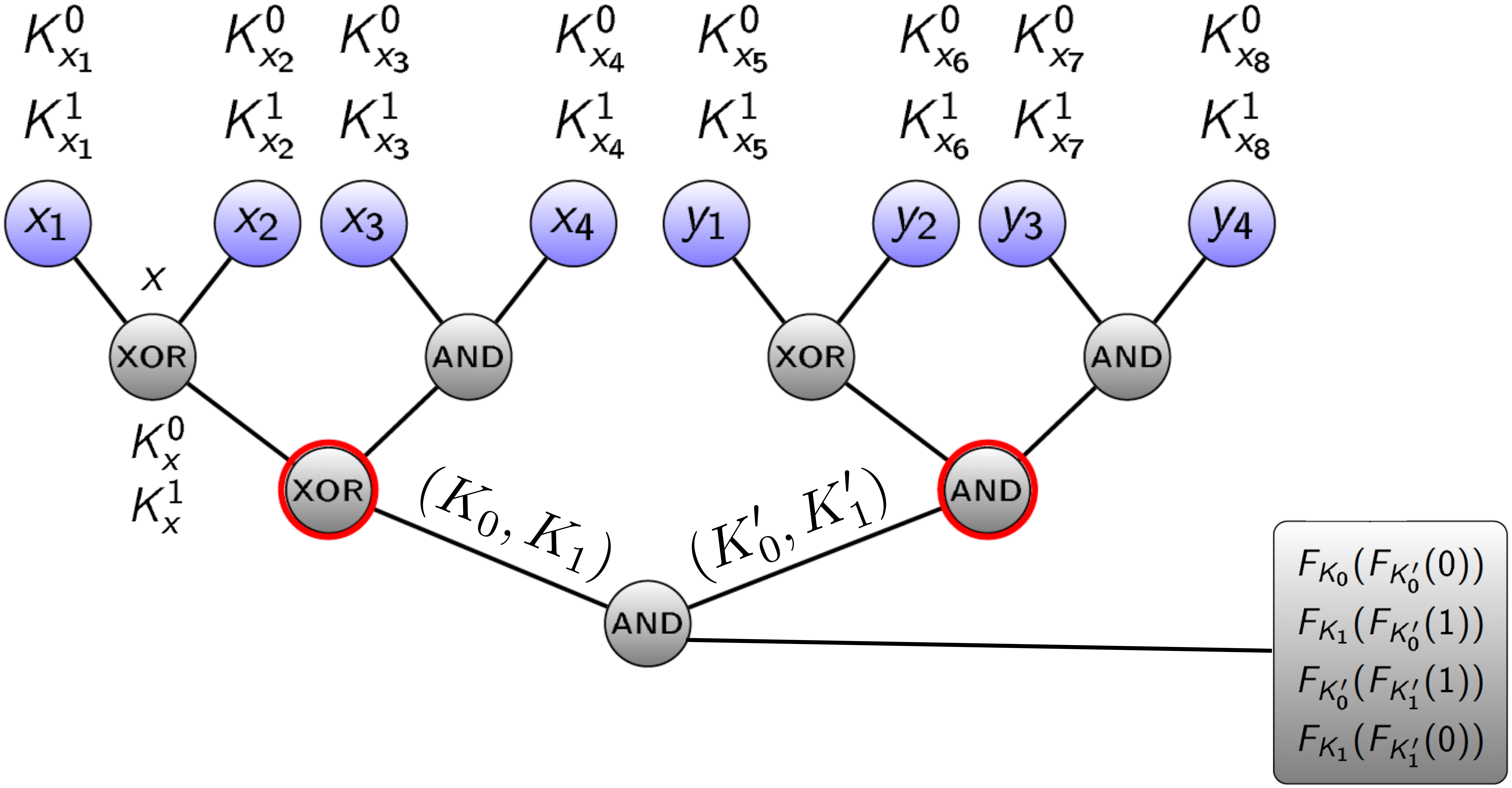
Selection bit b

Garbled Circuits

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



Garbled Circuits



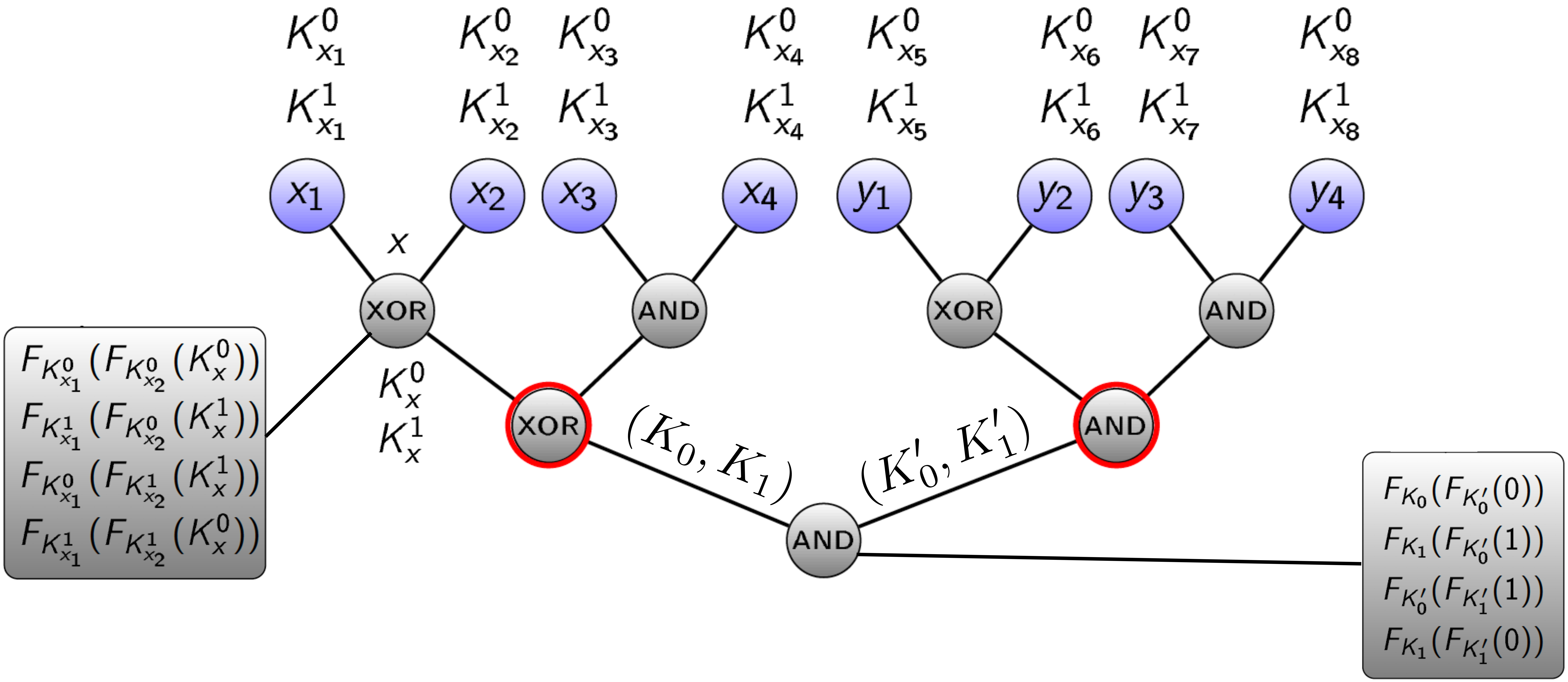
$F_{K_0}(F_{K'_0}(0))$

$F_{K_1}(F_{K'_0}(1))$

$F_{K'_0}(F_{K'_1}(1))$

$F_{K_1}(F_{K'_1}(0))$

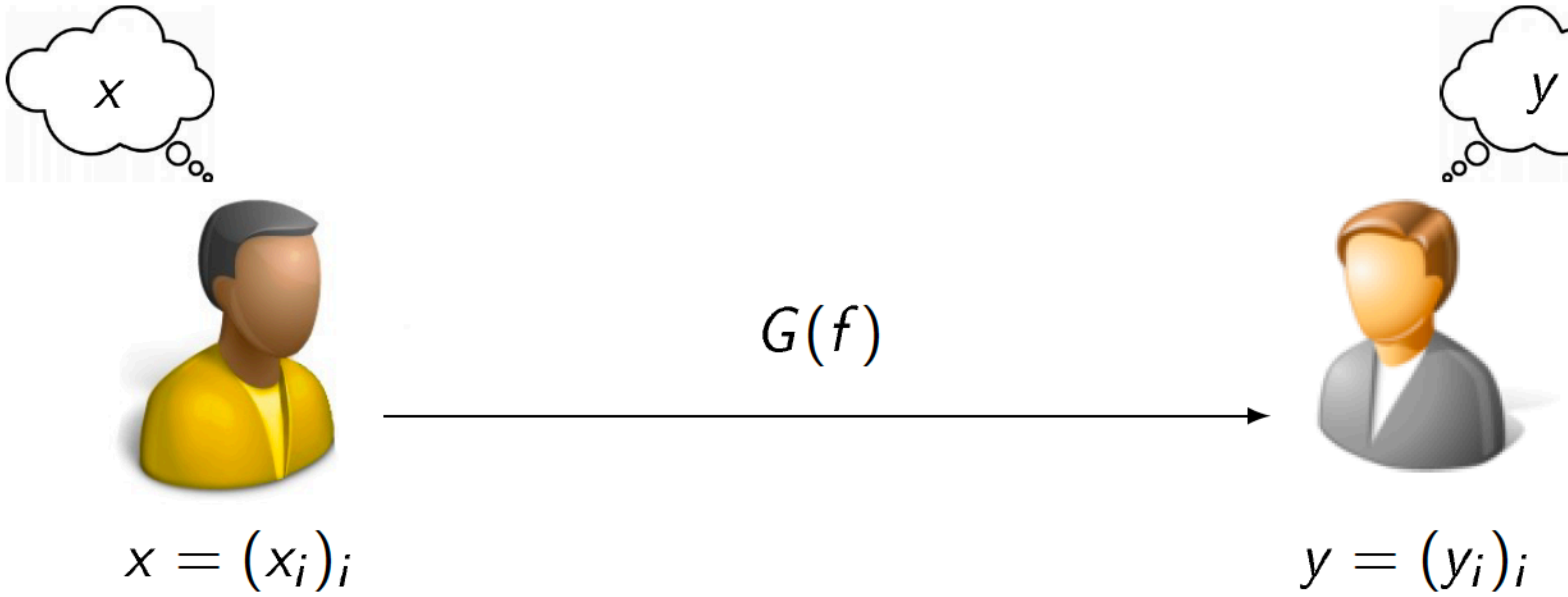
Garbled Circuits



$F_{K^0_{x_1}}(F_{K^0_{x_2}}(K^0_x))$
$F_{K^1_{x_1}}(F_{K^0_{x_2}}(K^1_x))$
$F_{K^0_{x_1}}(F_{K^1_{x_2}}(K^1_x))$
$F_{K^1_{x_1}}(F_{K^1_{x_2}}(K^0_x))$

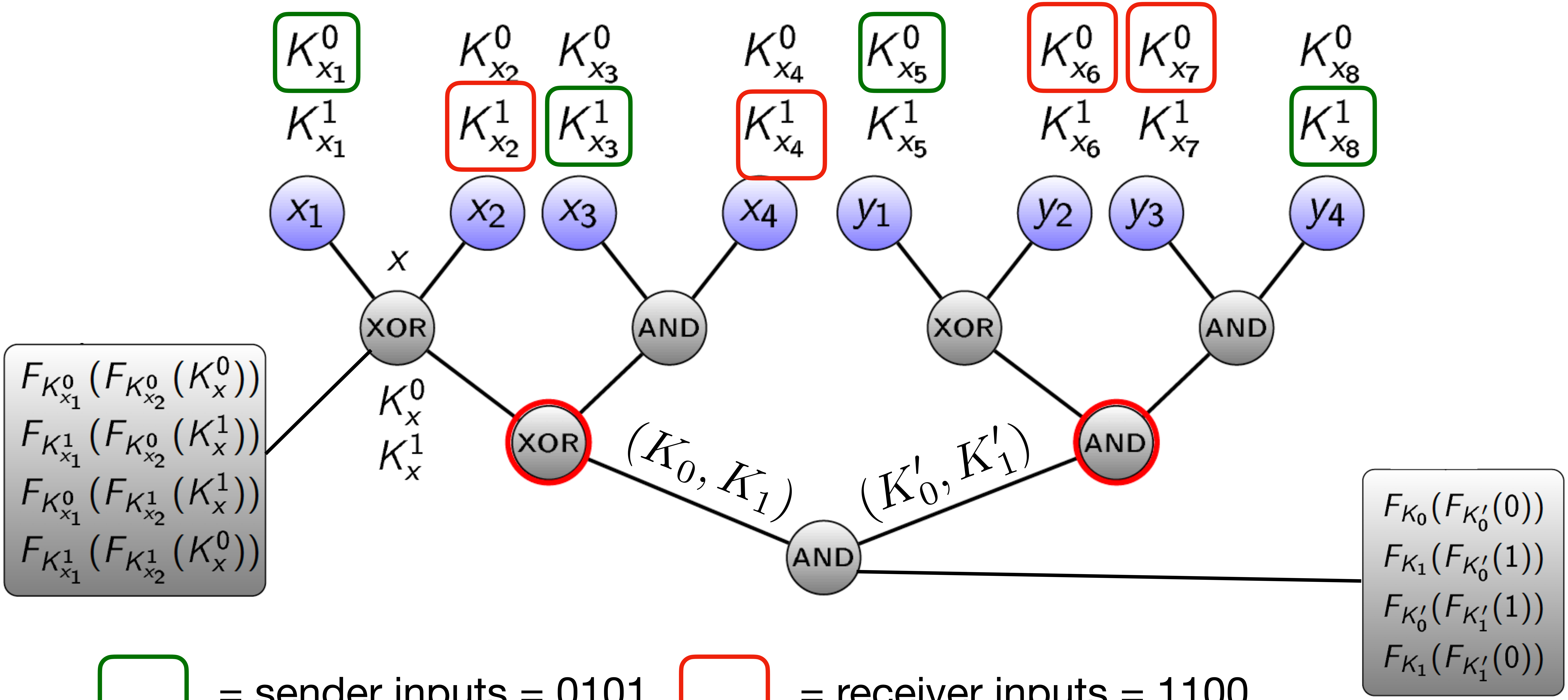
$F_{K_0}(F_{K'_0}(0))$
$F_{K_1}(F_{K'_0}(1))$
$F_{K'_0}(F_{K'_1}(1))$
$F_{K_1}(F_{K'_1}(0))$

Two-Party Secure Computation for All Functions



It remains to find a way to transmit exactly the appropriate input keys (and nothing more)

Garbled Circuits

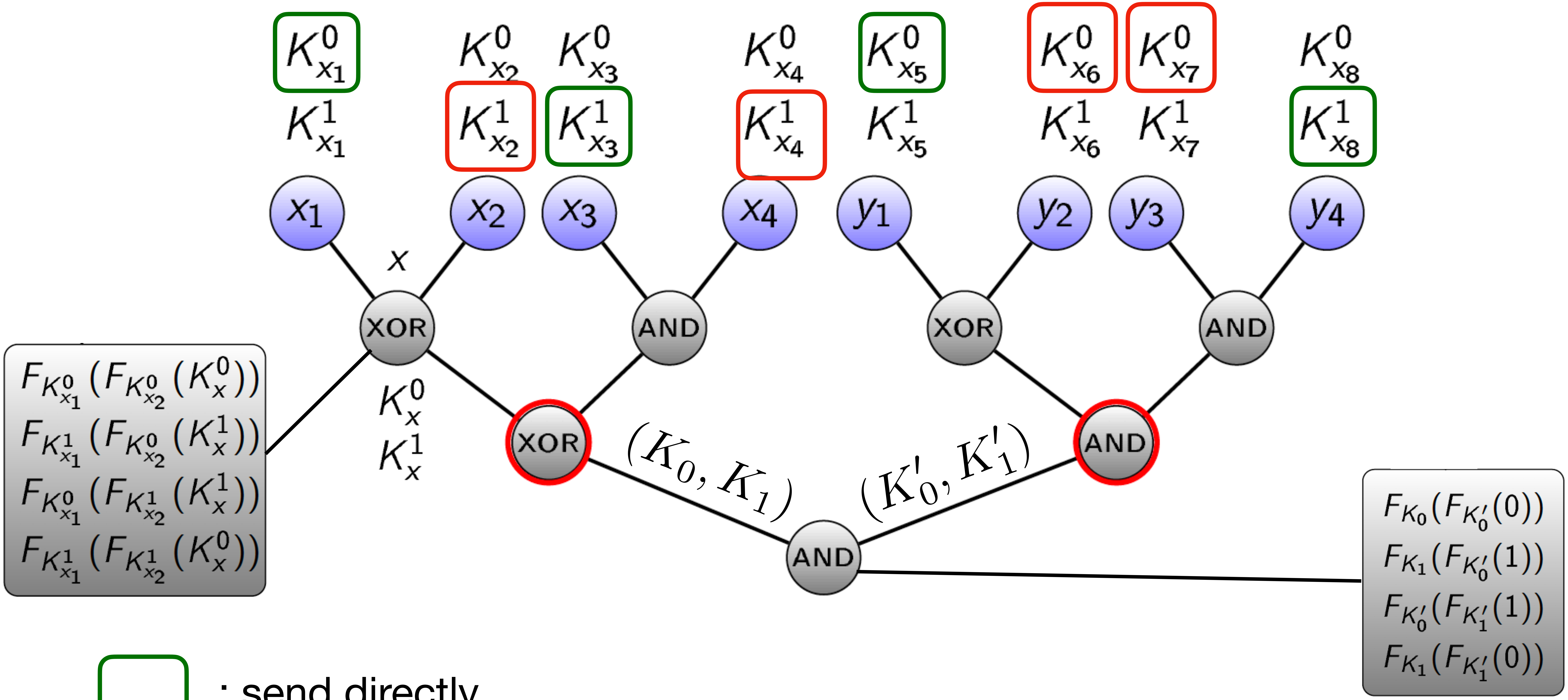


$$F_{K^0_{x_1}}(F_{K^0_{x_2}}(K^0_x))$$
$$F_{K^1_{x_1}}(F_{K^0_{x_2}}(K^1_x))$$
$$F_{K^0_{x_1}}(F_{K^1_{x_2}}(K^1_x))$$
$$F_{K^1_{x_1}}(F_{K^1_{x_2}}(K^0_x))$$

$$F_{K_0}(F_{K'_0}(0))$$
$$F_{K_1}(F_{K'_0}(1))$$
$$F_{K'_0}(F_{K'_1}(1))$$
$$F_{K_1}(F_{K'_1}(0))$$

🟩 = sender inputs = 0101 🟥 = receiver inputs = 1100

Garbled Circuits

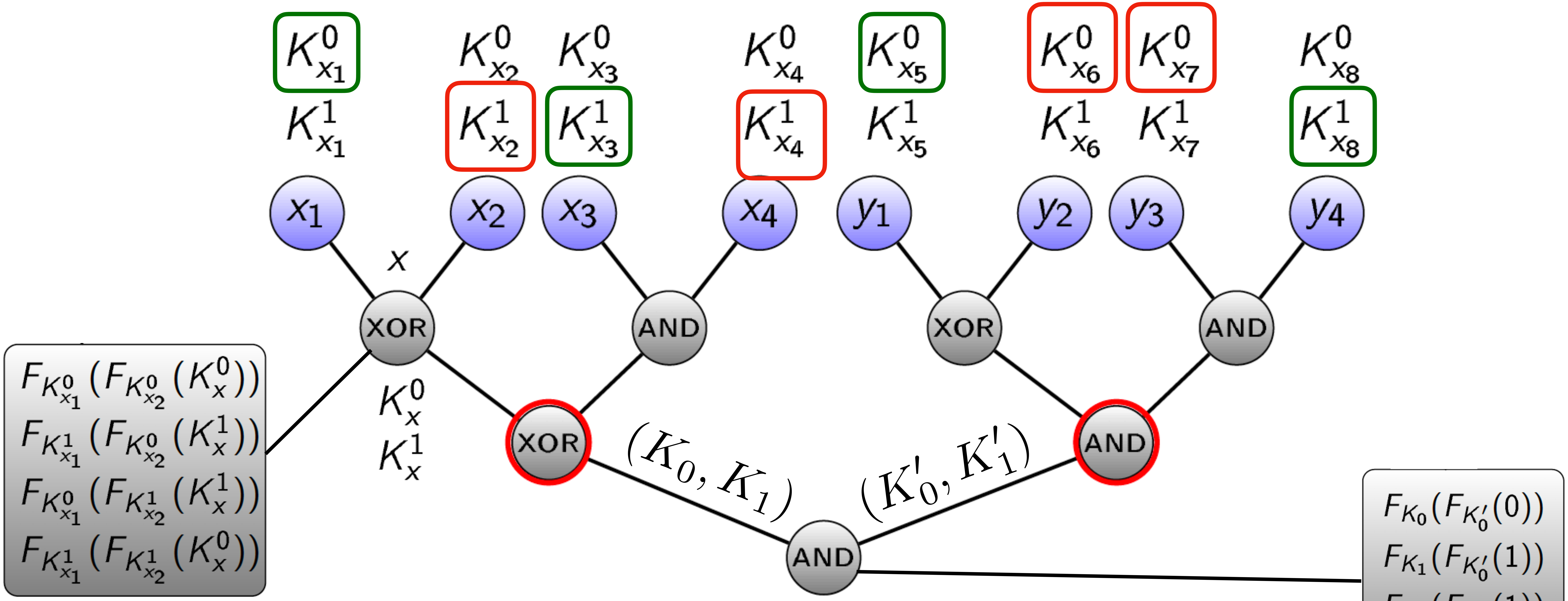


$F_{K_{x_1}^0}(F_{K_{x_2}^0}(K_x^0))$
$F_{K_{x_1}^1}(F_{K_{x_2}^0}(K_x^1))$
$F_{K_{x_1}^0}(F_{K_{x_2}^1}(K_x^1))$
$F_{K_{x_1}^1}(F_{K_{x_2}^1}(K_x^0))$

$F_{K_0}(F_{K_0'}(0))$
$F_{K_1}(F_{K_0'}(1))$
$F_{K_0'}(F_{K_1'}(1))$
$F_{K_1}(F_{K_1'}(0))$

□ : send directly

Garbled Circuits



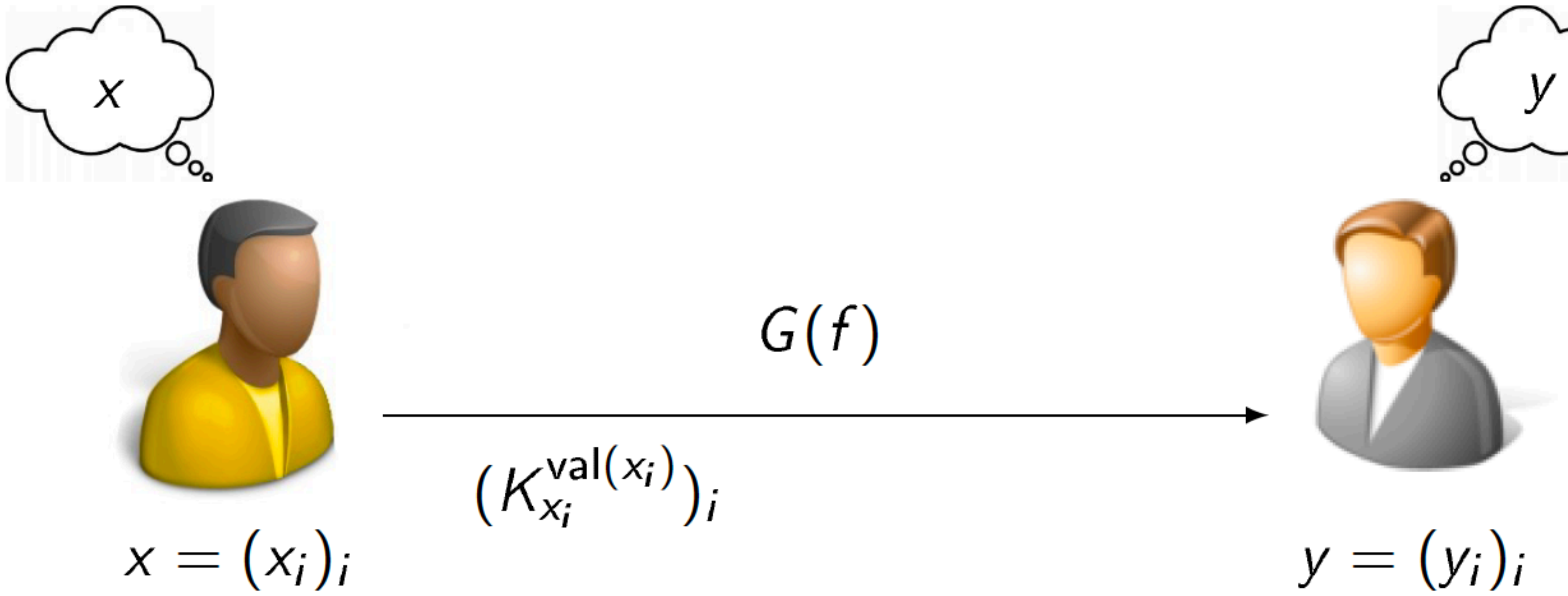
$$F_{K_{x_1}^0}(F_{K_{x_2}^0}(K_x^0))$$
$$F_{K_{x_1}^1}(F_{K_{x_2}^0}(K_x^1))$$
$$F_{K_{x_1}^0}(F_{K_{x_2}^1}(K_x^1))$$
$$F_{K_{x_1}^1}(F_{K_{x_2}^1}(K_x^0))$$

$$F_{K_0}(F_{K'_0}(0))$$
$$F_{K_1}(F_{K'_0}(1))$$
$$F_{K'_0}(F_{K'_1}(1))$$
$$F_{K_1}(F_{K'_1}(0))$$

🟩 : send directly

🟥 : use oblivious transfer

Two-Party Secure Computation for All Functions



x

y

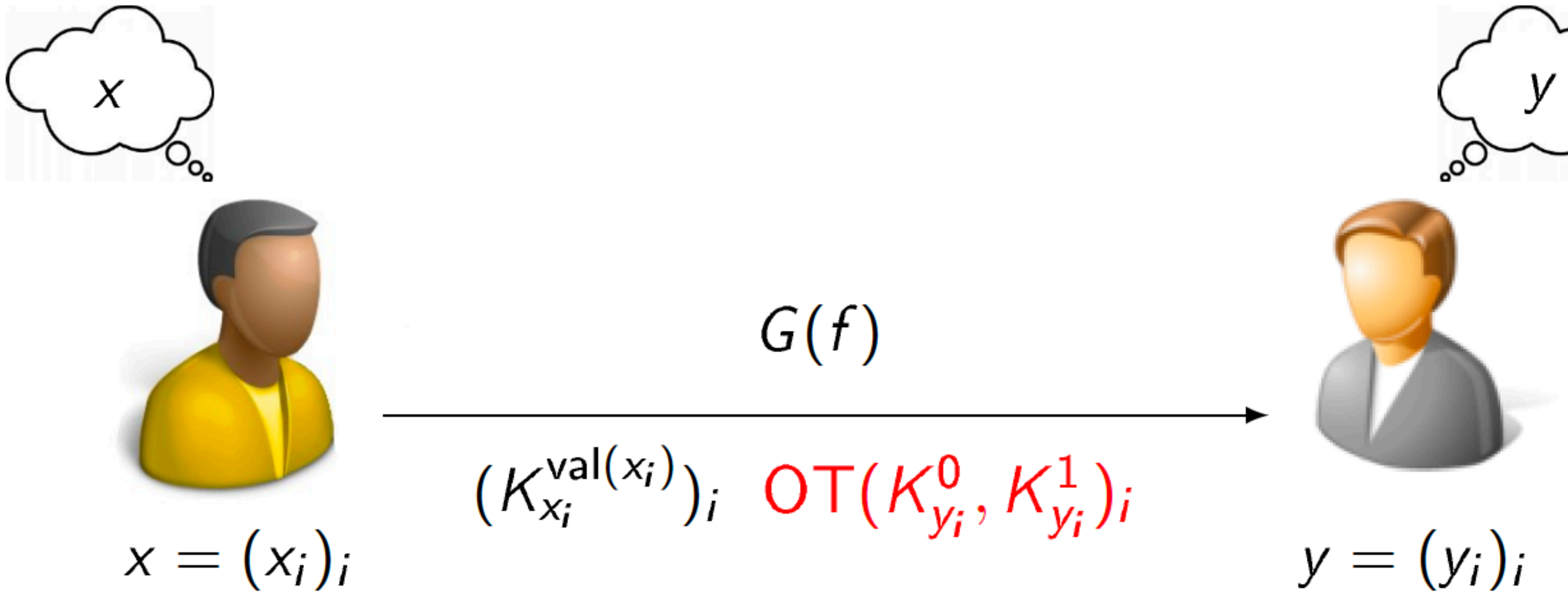
$G(f)$

$(K_{x_i}^{\mathsf{val}(x_i)})_i$

$x = (x_i)_i$

$y = (y_i)_i$

Two-Party Secure Computation for All Functions



x

y

$G(f)$

$(K_{x_i}^{\mathsf{val}(x_i)})_i$ $\mathsf{OT}(K_{y_i}^0, K_{y_i}^1)_i$

$x = (x_i)_i$

$y = (y_i)_i$

That’s all for today!

If you have any question after the lesson:

couteau@irif.fr