CS4501 Cryptographic Protocols
Lecture 3: Simulation, Communication

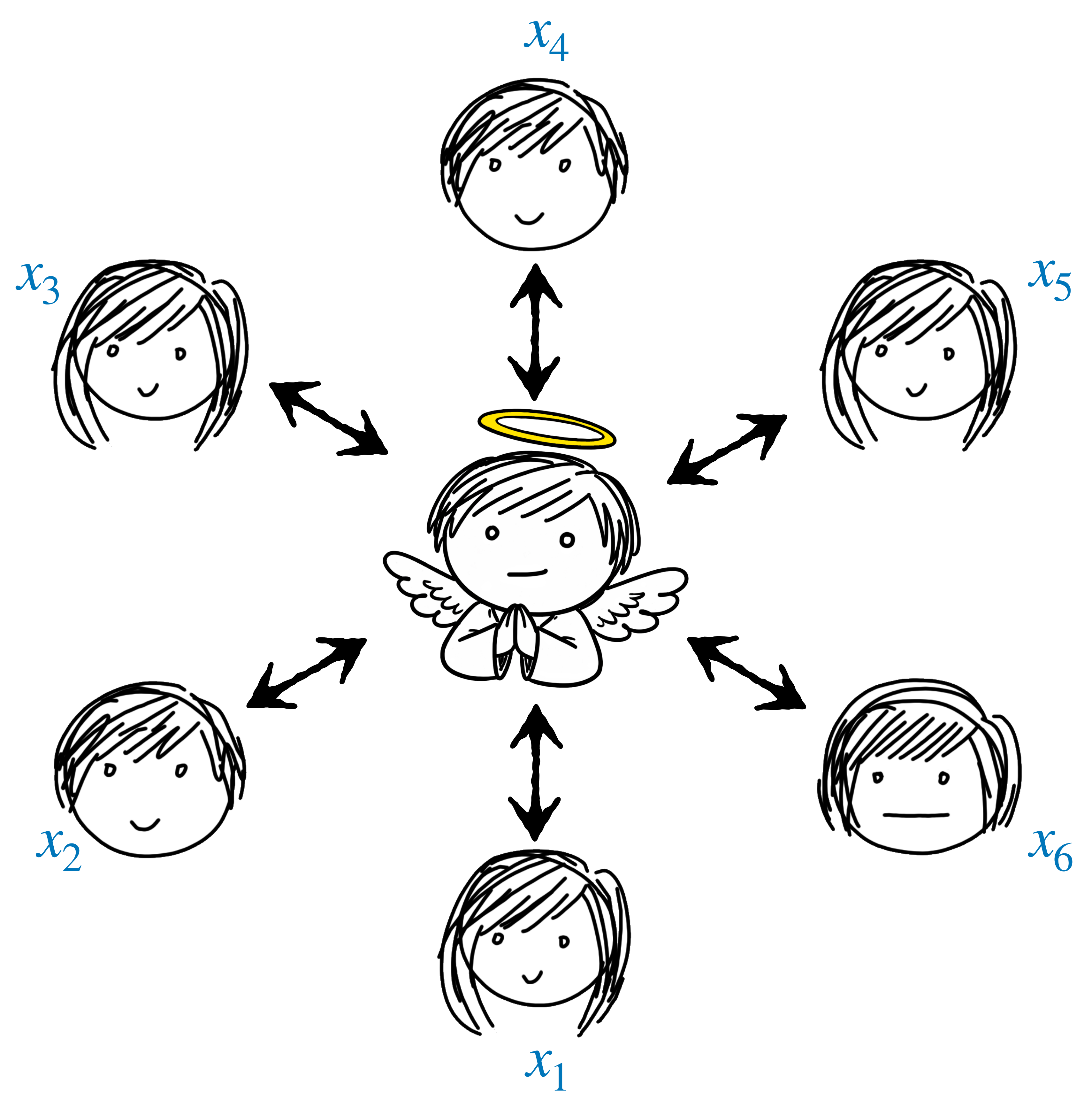
https://jackdoerner.net/teaching/#2026/Spring/CS4501

The Ideal World

1. Each P_i sends its input x_i to the ideal functionality ($\mathscr{F}$) which is a trusted third party.

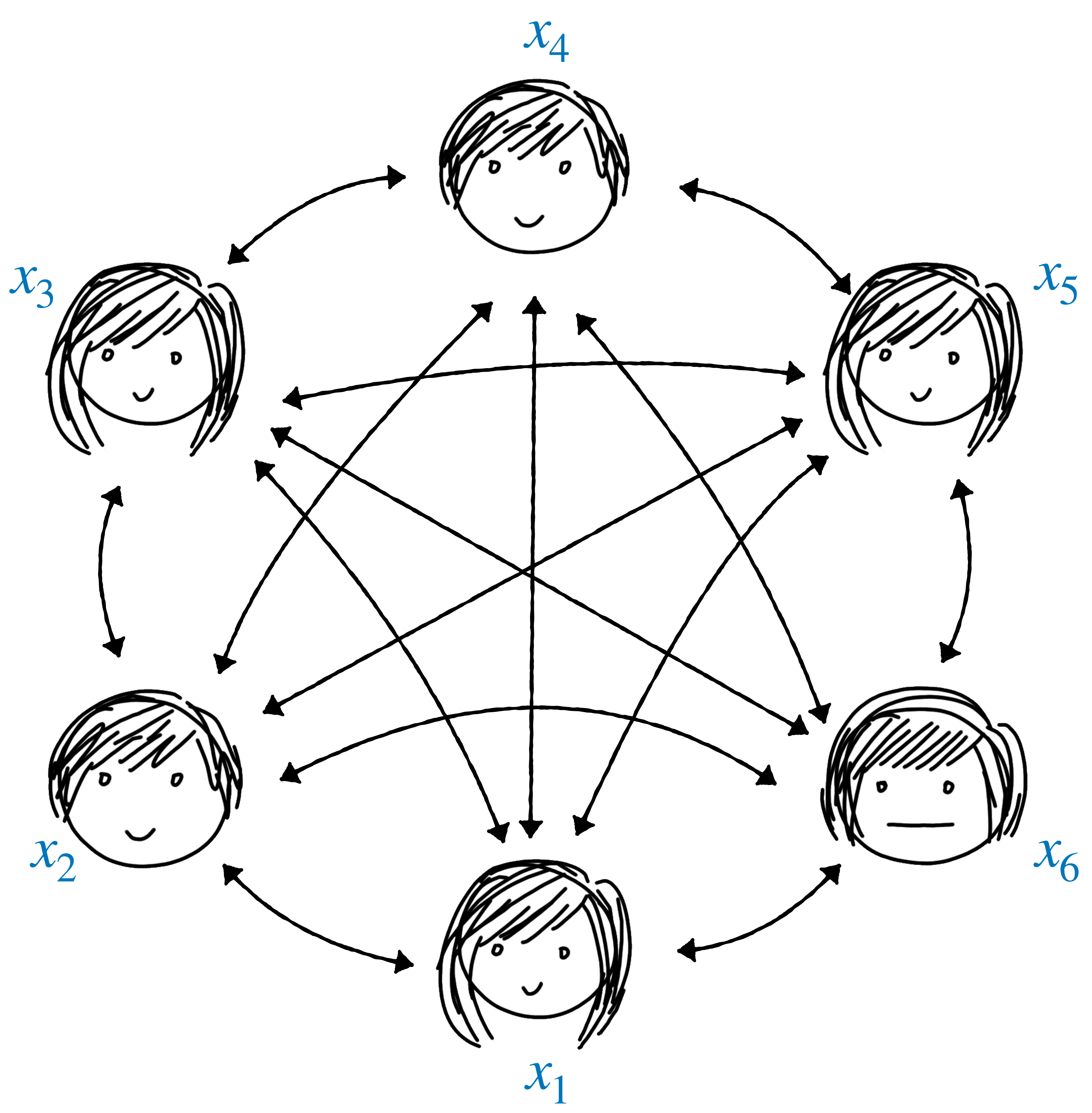
2. $\mathscr{F}$ computes $y = f(x_1, \ldots, x_n)$.

3. $\mathscr{F}$ sends y to every party, and they output it.

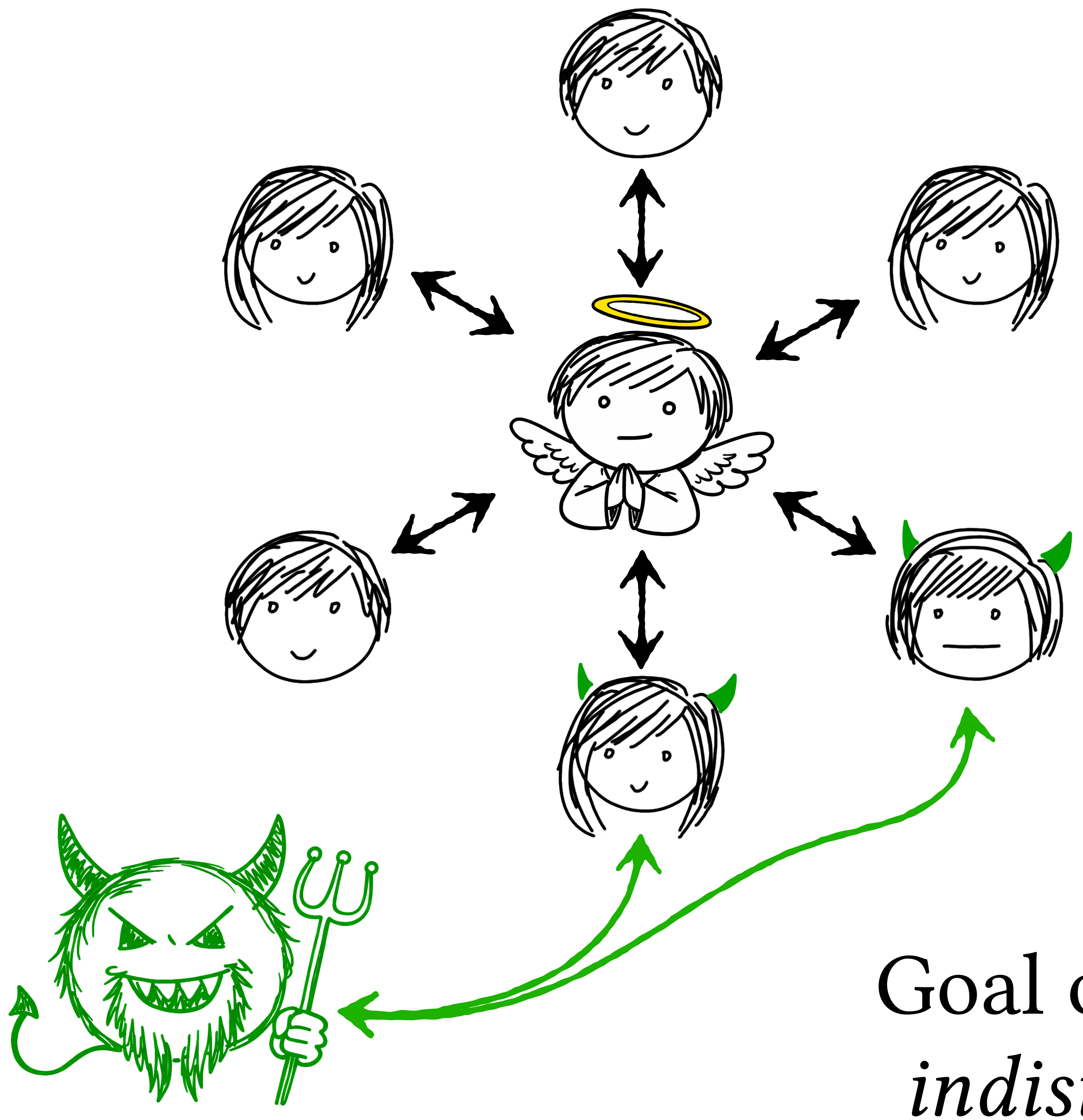


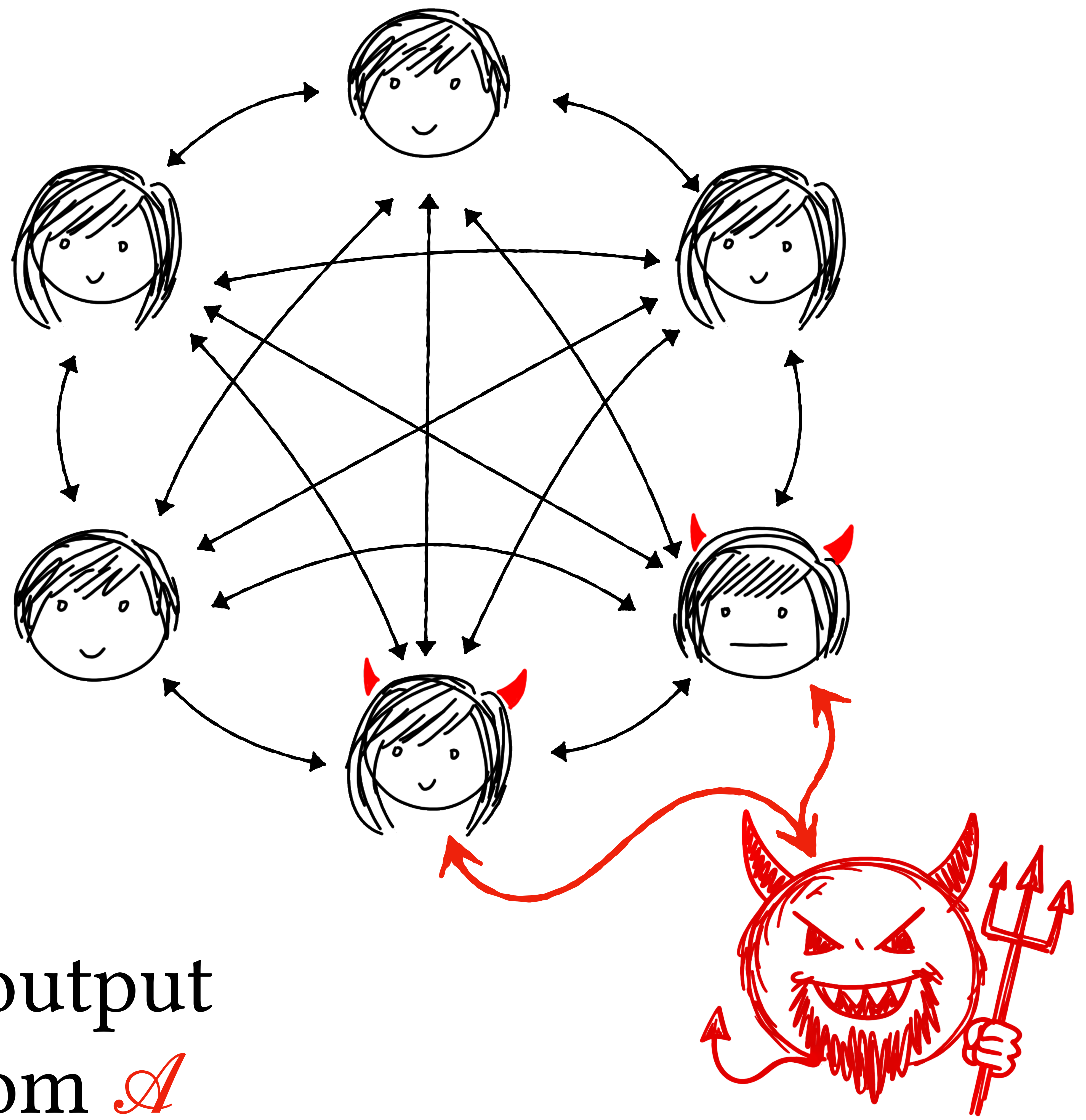
The Real World

1. The parties $(P_1, \ldots, P_n)$ run a protocol π on inputs $(x_1, \ldots, x_n)$
2. When π terminates, the parties output y.



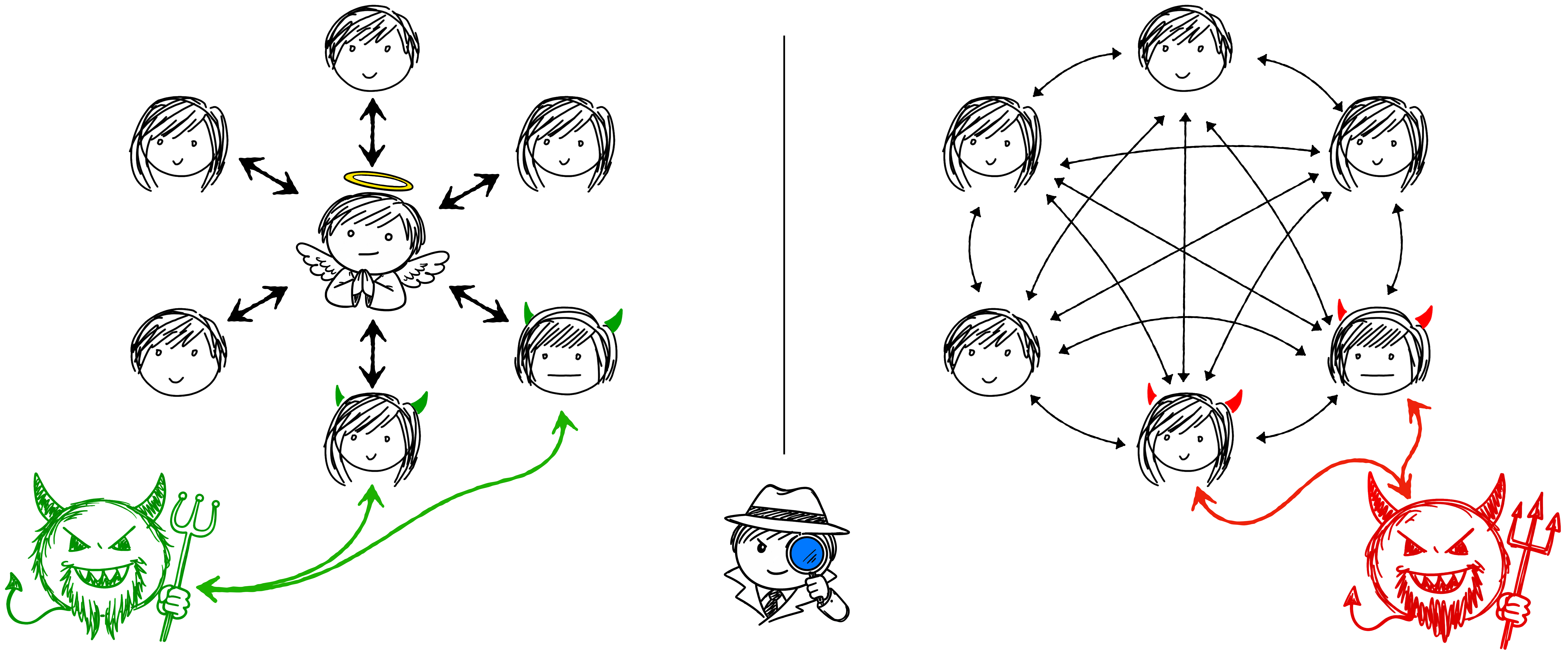
For every real adversary, and ideal adversary





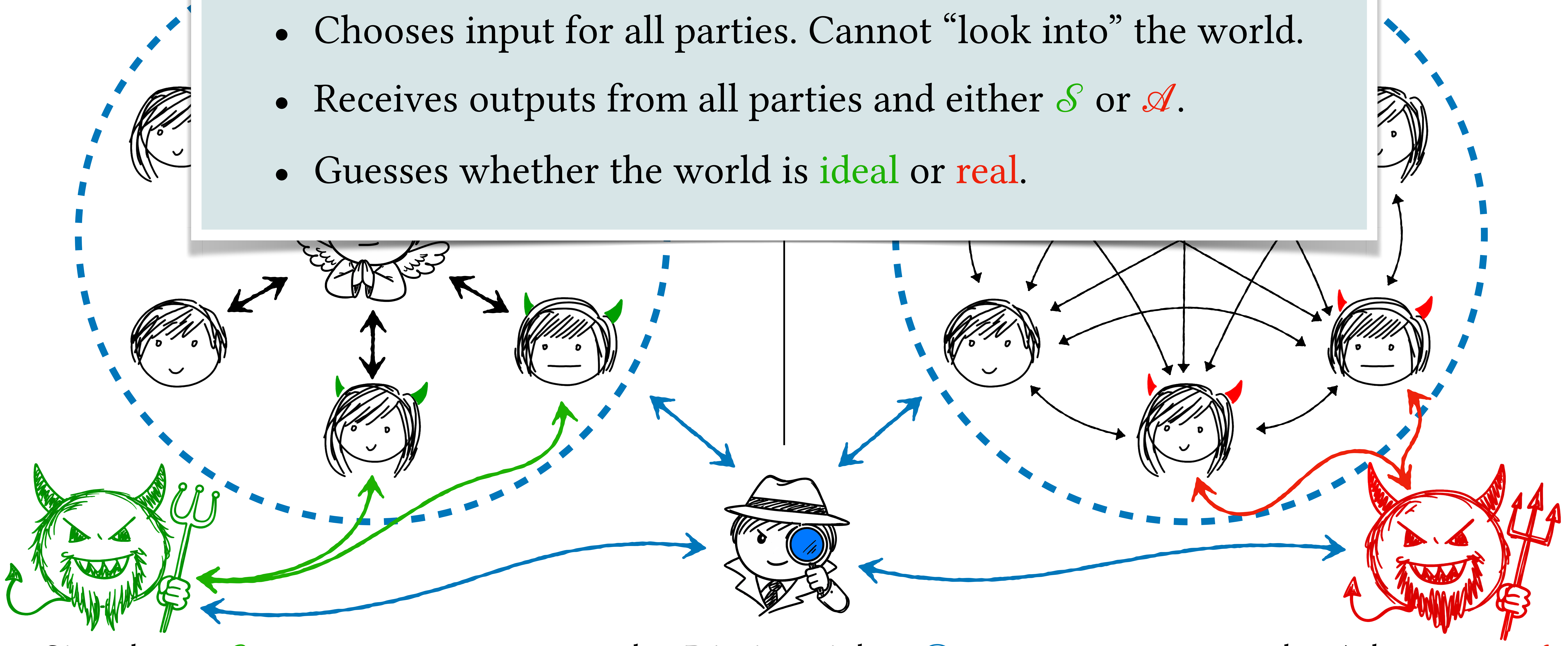
Goal of $\mathcal{S}$: produce output *indistinguishable* from $\mathcal{A}$

Is $\mathcal{S}$ is Indistinguishable from $\mathcal{A}$? Who will Judge?



the Simulator $\mathcal{S}$ the Distinguisher $\mathcal{D}$ the Adversary $\mathcal{A}$

Is $\mathscr{S}$ is … dge?

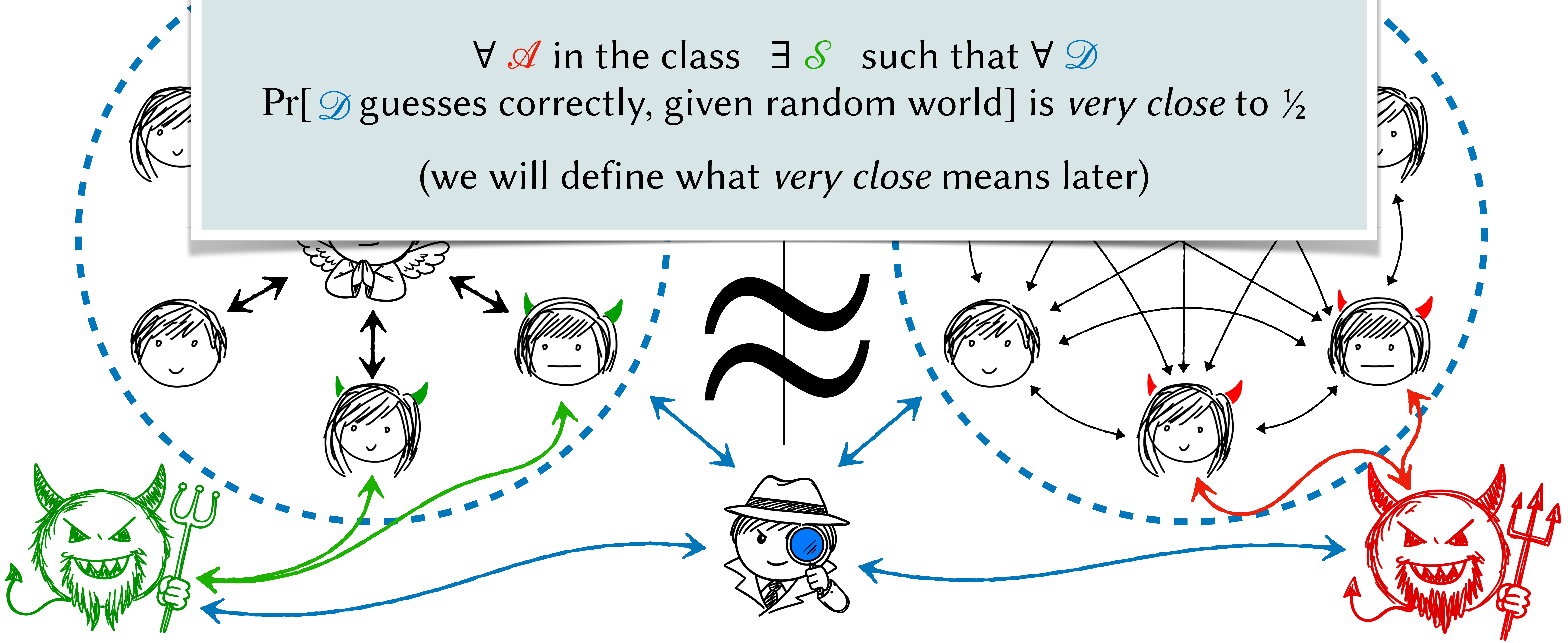


the Distinguisher $\mathscr{D}$

- Interacts with one of the worlds and attempts to determine *which* one by running an experiment.
- Chooses input for all parties. Cannot "look into" the world.
- Receives outputs from all parties and either $\mathscr{S}$ or $\mathscr{A}$.
- Guesses whether the world is ideal or real.

the Simulator $\mathscr{S}$ the Distinguisher $\mathscr{D}$ the Adversary $\mathscr{A}$

Is $\mathcal{S}$ is dge?



the Simulator $\mathcal{S}$
the Distinguisher $\mathcal{D}$
the Adversary $\mathcal{A}$

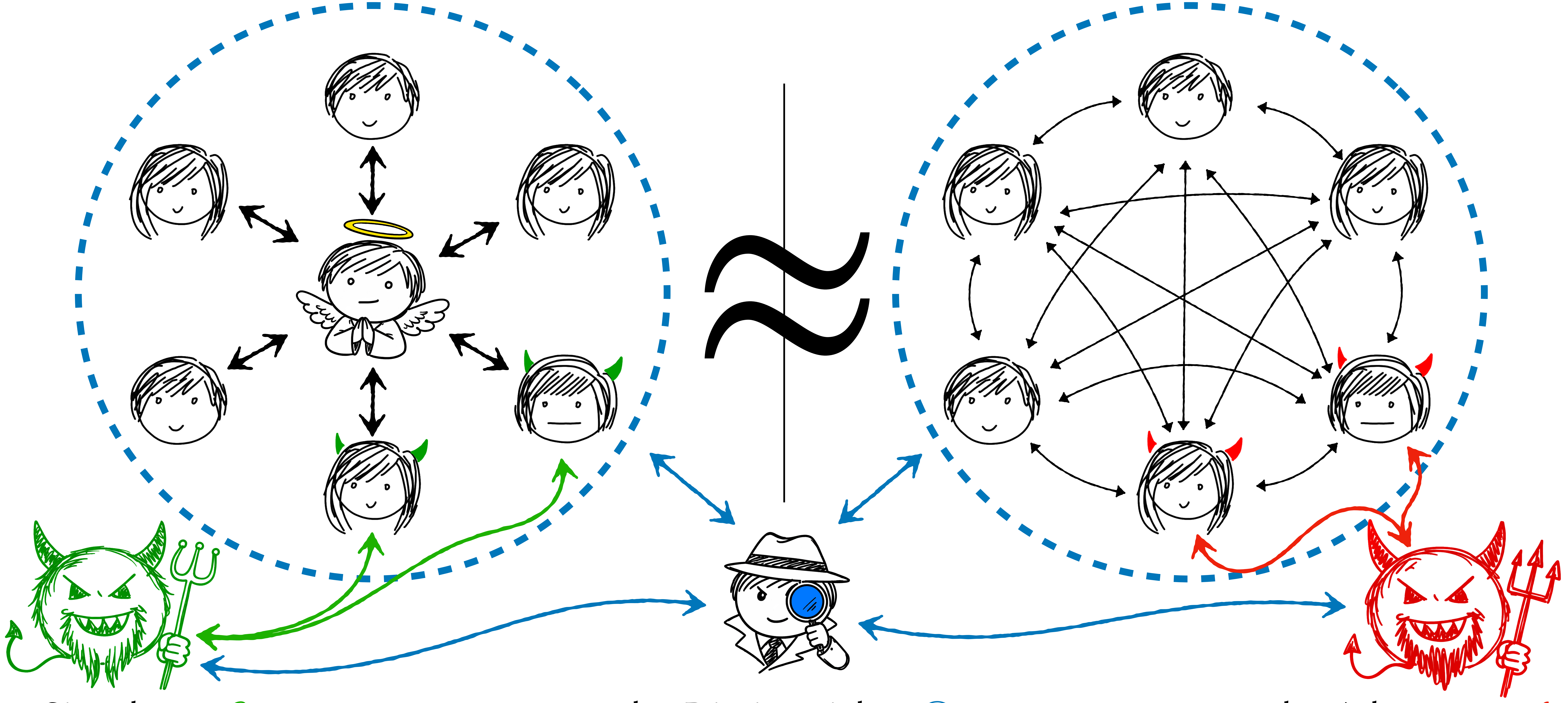
Finally, How to Define Simulation-Based Security!

Definition 1. The protocol π *realizes* the ideal functionality $\mathcal{F}$ in the presence of some class of adversaries if:

$\forall\ \mathcal{A}$ in the class $\exists\ \mathcal{S}$ such that $\forall\ \mathcal{D}$
Pr[$\mathcal{D}$ guesses correctly, given random world] is *very close* to ½

(we will define what *very close* means later)

The Simulation-Based Security Paradigm



the Simulator $\mathscr{S}$ the Distinguisher $\mathscr{D}$ the Adversary $\mathscr{A}$

Specifying the Details

We've met the players, but in order to understand what we're achieving, we must know more about them.

- **Functionality:** what do we want to compute, and with what IO behavior? A functionality can also capture *vulnerabilities* by directly taking inputs from or leaking information to the simulator $\mathscr{S}$.
- **Adversarial Model:** what kinds of behavior and interaction with the system do we want to protect against? E.g. who can be corrupted?
- **Security Type:** how strong should our protection be? How much computing power does $\mathscr{D}$ have? How much better than random is the guesswork of $\mathscr{D}$ allowed to be?
- **Network Model:** Who is connected to who in the real world? Are those connections private? Do the parties have a shared clock? Is there a way to *broadcast* reliably?

Facets of Adversarial Models

Behavior:

- *Semi-honest* (a.k.a. *Passive*): corrupted parties follow the protocol honestly, but share their internal state with $\mathscr{A}$, who tries to learn more than is allowed.
- *Malicious* (a.k.a. *Active*): corrupted parties can deviate from the protocol instructions in arbitrary ways. $\mathscr{A}$ coordinates their actions.

Adversarial Power:

- *Unbounded* (i.e. "all-powerful"): $\mathscr{A}$ has unlimited computing power. Can break any cryptographic assumption. We can still achieve security using information theory!
- *Computationally Bounded*: $\mathscr{A}$ runs in Probabilistic Polynomial Time (PPT).

Behavior:

- *Semi-honest* (a.k.a. *Passive*): corrupted parties follow the protocol honestly, but share their internal state with $\mathscr{A}$, who tries to learn more than is allowed.
- *Malicious* (a.k.a. *Active*): corrupted parties can deviate from the protocol instructions in arbitrary ways. $\mathscr{A}$ coordinates their actions.

Adversarial Power:

- *Unbounded* (i.e. "all-powerful"): $\mathscr{A}$ has unlimited computing power. Can break any cryptographic assumption. We can still achieve security using information theory!
- *Computationally Bounded*: $\mathscr{A}$ runs in Probabilistic Polynomial Time (PPT).

Corruption Strategy:

- *Static*: corruptions are determined at the beginning of the experiment. Honest parties always stay honest.
- *Adaptive*: $\mathscr{A}$ can dynamically corrupt parties during the protocol (security is very hard to achieve in this setting).

We will start here

Security Types

Perfect:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be identically distributed from the perspective of $\mathscr{D}$.
- $\mathscr{D}$ must be able to do no better than a random guess.

We will start here

Statistical:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be *statistically indistinguishable.* (Their *statistical distance* must be *negligible* relative to the *security parameter*)

Perfect:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be identically distributed from the perspective of $\mathscr{D}$.
- $\mathscr{D}$ must be able to do no better than a random guess.

Statistical:

- $\mathscr{D}$ and $\mathscr{A}$ have unbounded computational power.
- The real and ideal experiments must be *statistically indistinguishable*. (Their *statistical distance* must be *negligible* relative to the *security parameter*)

Computational:

- $\mathscr{D}$ and $\mathscr{A}$ are *efficient*. They run in PPT.
- We can make *cryptographic* assumptions. That is, we can assume certain computational problems can't be solved by $\mathscr{D}$ or $\mathscr{A}$.
- The real and ideal experiments must be *computationally indistinguishable*. This means that the outputs of $\mathscr{D}$ are *statistically close* when it interacts with the real and ideal experiments, even though the experiments themselves might have *statistically far* distributions.

Simulation-Based Security by Example

Example: n-Party Sum

- Let M be some positive integer fixed a priori.
- Each P_i has private input $x_i \le M/n$.
- They wish to compute $y = \sum_{i \in [n]} x_i$.

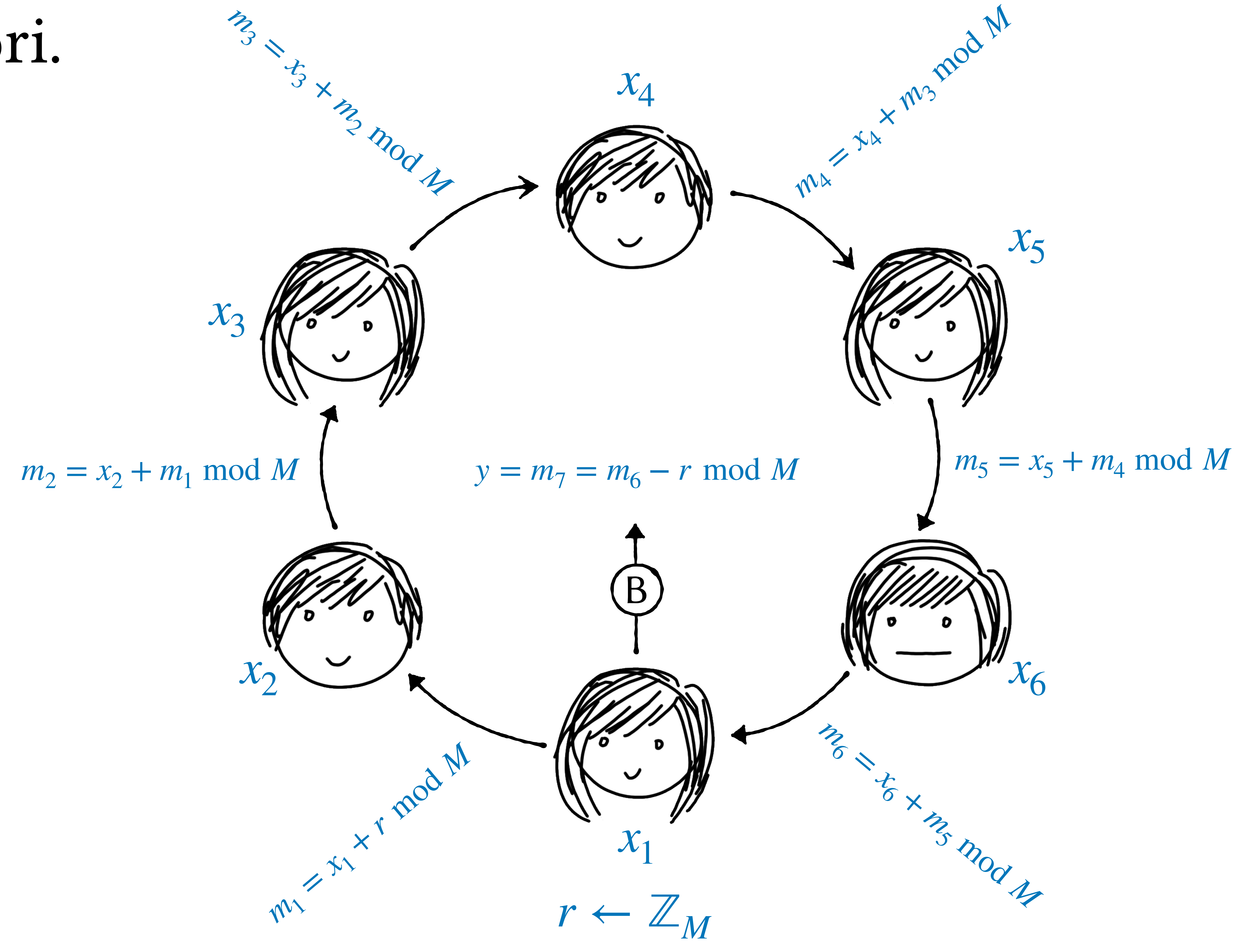
Notation

$[n] = \{1, \dots, n\}$ $[n, m] = \{n, \dots, m\}$

$\mathbb{Z}_M$ is the integers modulo M.
For our purposes, $\mathbb{Z}_M = [0, M-1]$.

$x \leftarrow D$ where D is a distribution means x is a random var distributed according to D

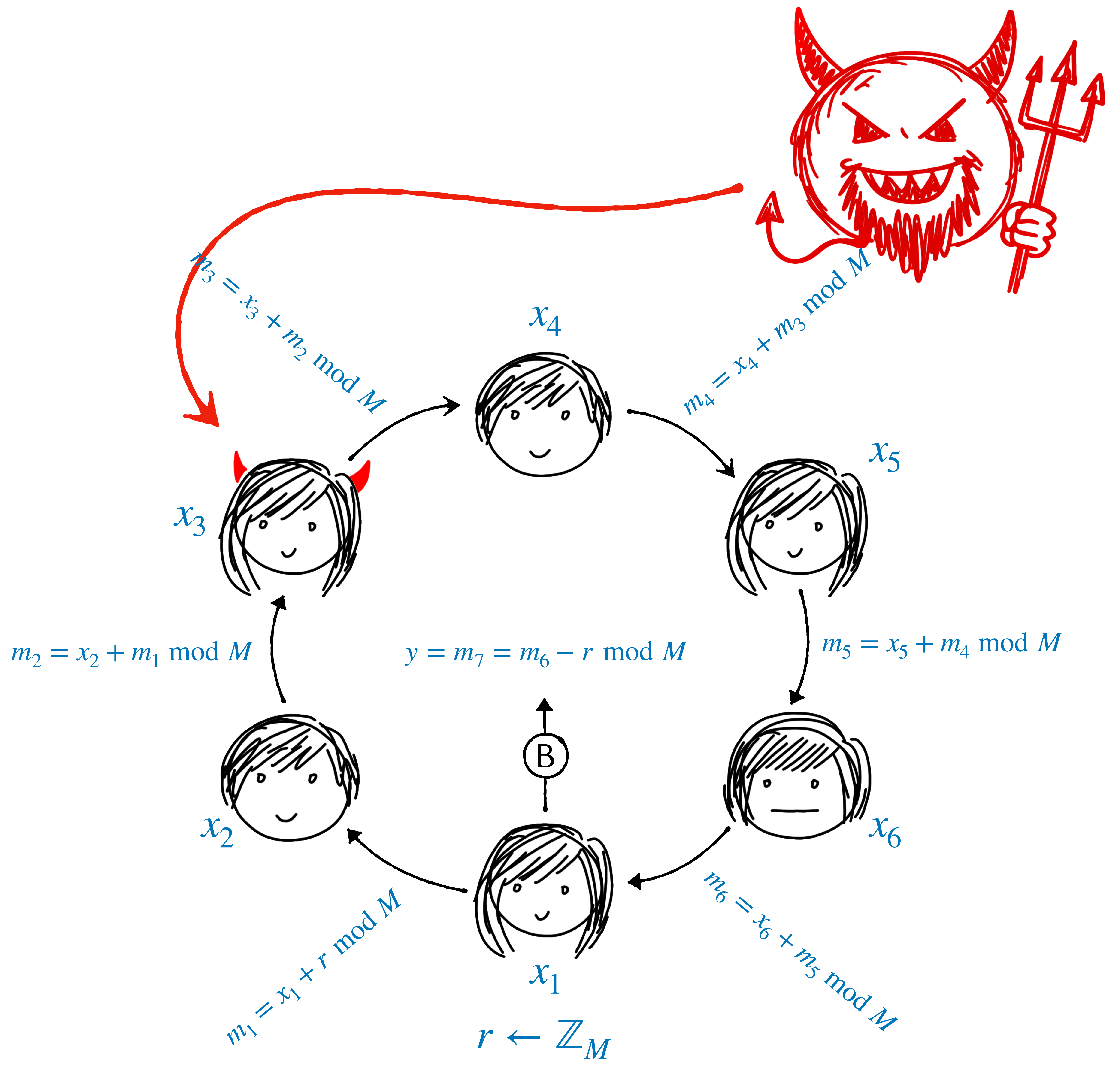
$x \leftarrow S$ where S is a set means x is sampled from the uniform distribution over S



$m_3 = x_3 + m_2 \bmod M$
x_4
$m_4 = x_4 + m_3 \bmod M$
x_3
x_5
$m_2 = x_2 + m_1 \bmod M$
$y = m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$
B
x_2
x_6
$m_1 = x_1 + r \bmod M$
x_1
$m_6 = x_6 + m_5 \bmod M$
$r \leftarrow \mathbb{Z}_M$

Example: n-Party Sum

- *Semi-honest* $\mathscr{A}$ *statically* corrupts at most one party (WLOG P_3)
- $\mathscr{A}$ learns $m_2 = x_2 + x_1 + r \bmod M$
- Last time we talked about the property of **privacy**.
- $\forall i \in \mathbb{Z}_M \;\; \Pr[m_2 = i] = 1/M$
- In other words, the distribution of m_2 is independent of x_1 and x_2.
- $\mathscr{A}$ could sample $m_2 \leftarrow \mathbb{Z}_M$ on its own.



$m_3 = x_3 + m_2 \bmod M$
x_4
$m_4 = x_4 + m_3 \bmod M$
x_3
x_5
$m_2 = x_2 + m_1 \bmod M$
$y = m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$
B
x_2
x_6
$m_1 = x_1 + r \bmod M$
x_1
$m_6 = x_6 + m_5 \bmod M$
$r \leftarrow \mathbb{Z}_M$

Example: n-Party Sum

- *Semi-honest* $\mathscr{A}$ *statically* corrupts at most one party (WLOG P_3)

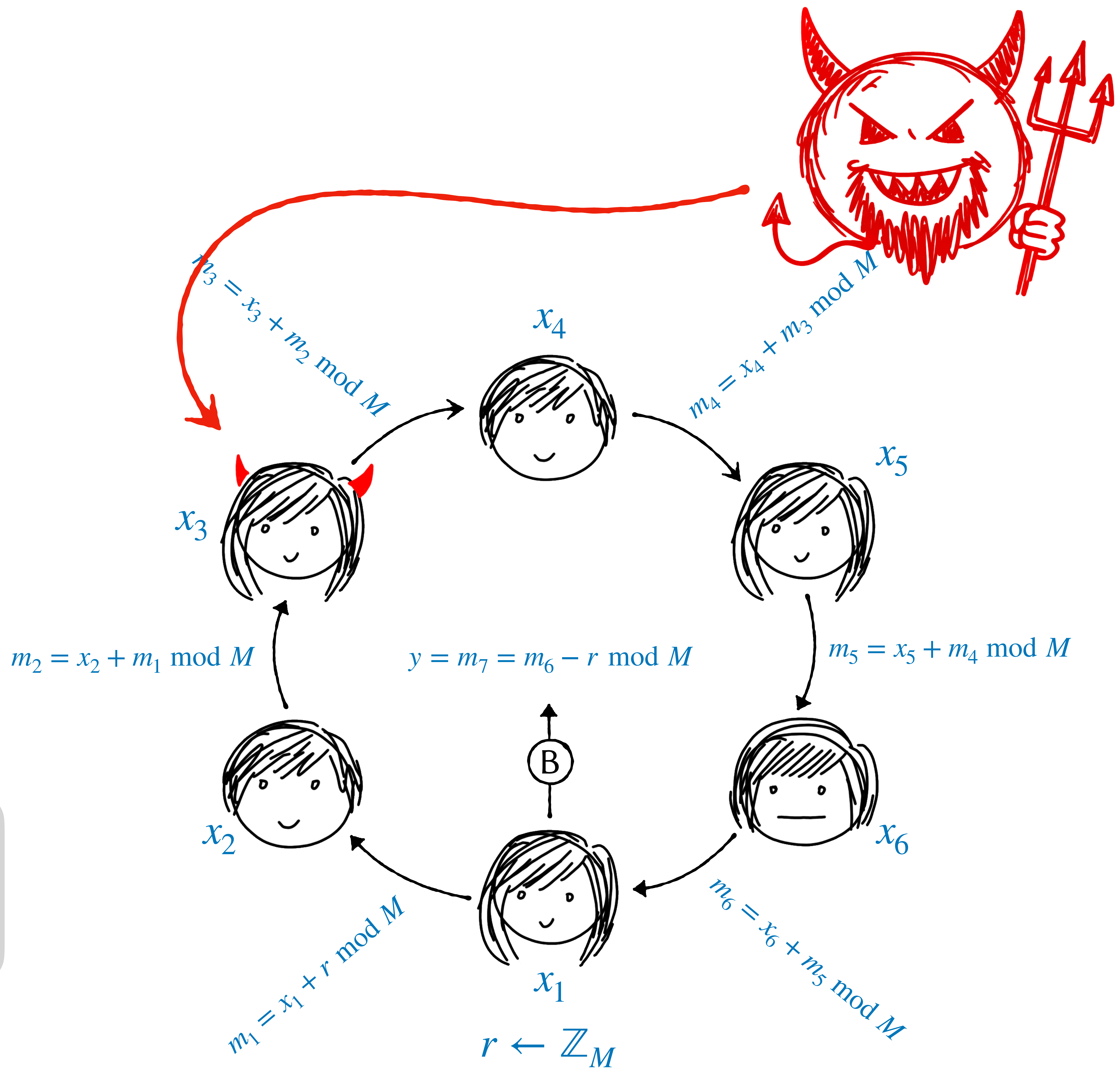
- $\mathscr{A}$ learns $m_2 = x_2 + x_1 + r \bmod M$

- Now we have introduced the *real-ideal model.* We want to prove that for every such $\mathscr{A}$ there is a simulator $\mathscr{S}$.

$\forall$ $\mathscr{A}$ in the class $\exists$ $\mathscr{S}$ such that $\forall$ $\mathscr{D}$
Pr[$\mathscr{D}$ guesses correctly] is *very close* to ½

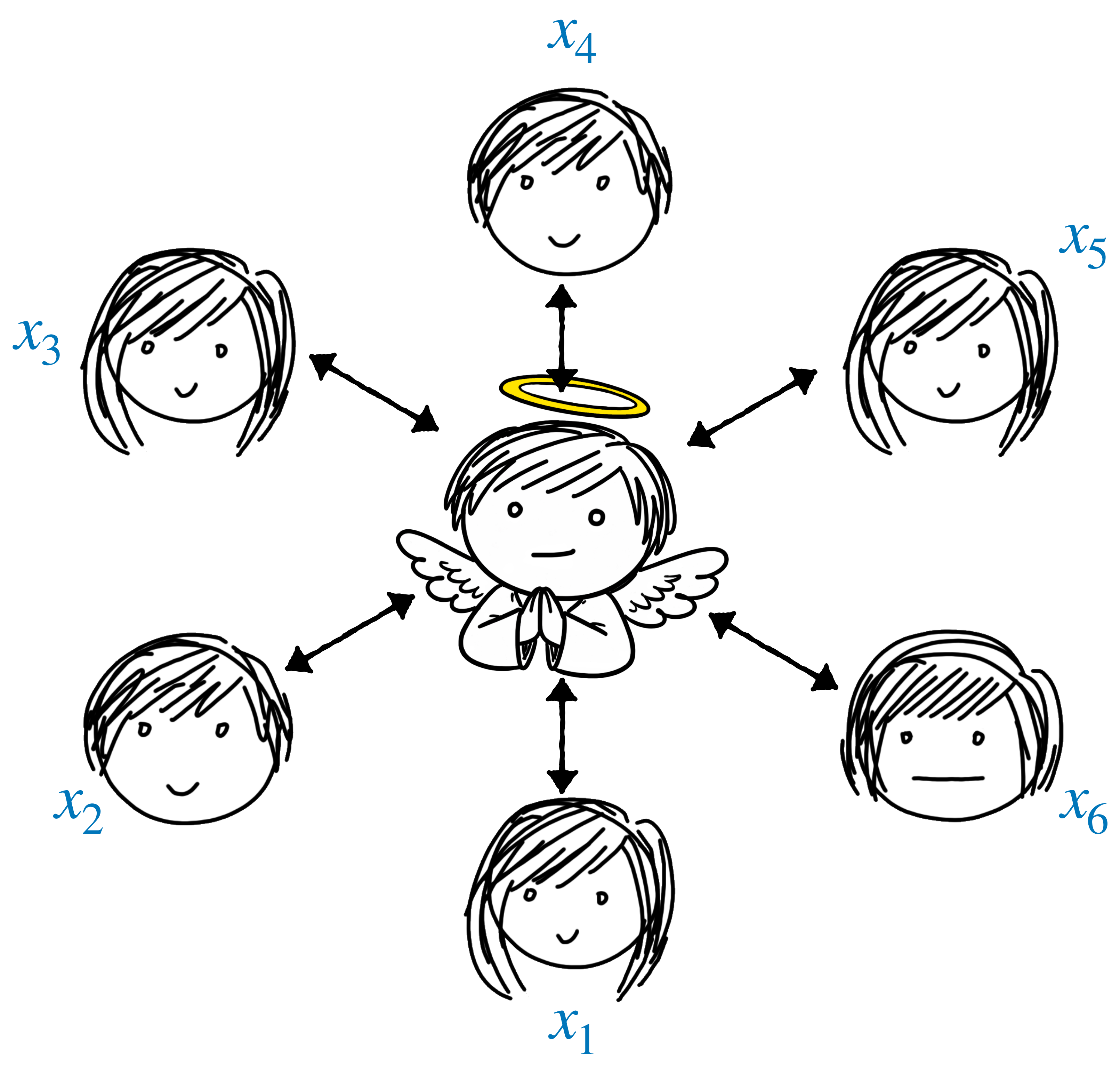
Semi-honest, statically corrupts ≤ 1

- We know what the real-world experiment looks like. What's missing from our picture of the ideal world?


$m_3 = x_3 + m_2 \bmod M$
x_4
$m_4 = x_4 + m_3 \bmod M$
x_3
x_5
$m_2 = x_2 + m_1 \bmod M$
$y = m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$
B
x_2
x_6
$m_1 = x_1 + r \bmod M$
x_1
$m_6 = x_6 + m_5 \bmod M$
$r \leftarrow \mathbb{Z}_M$

Example: n-Party Sum

- **Missing Piece #1**: what functionality do we want this protocol to realize?
  - Let M be some positive integer
  - Each P_i has private input $x_i \le M/n$
  - They wish to compute $y = \sum_{i\in[n]} x_i$



Example: n-Party Sum

- **Missing Piece #1:** what functionality do we want this protocol to realize?

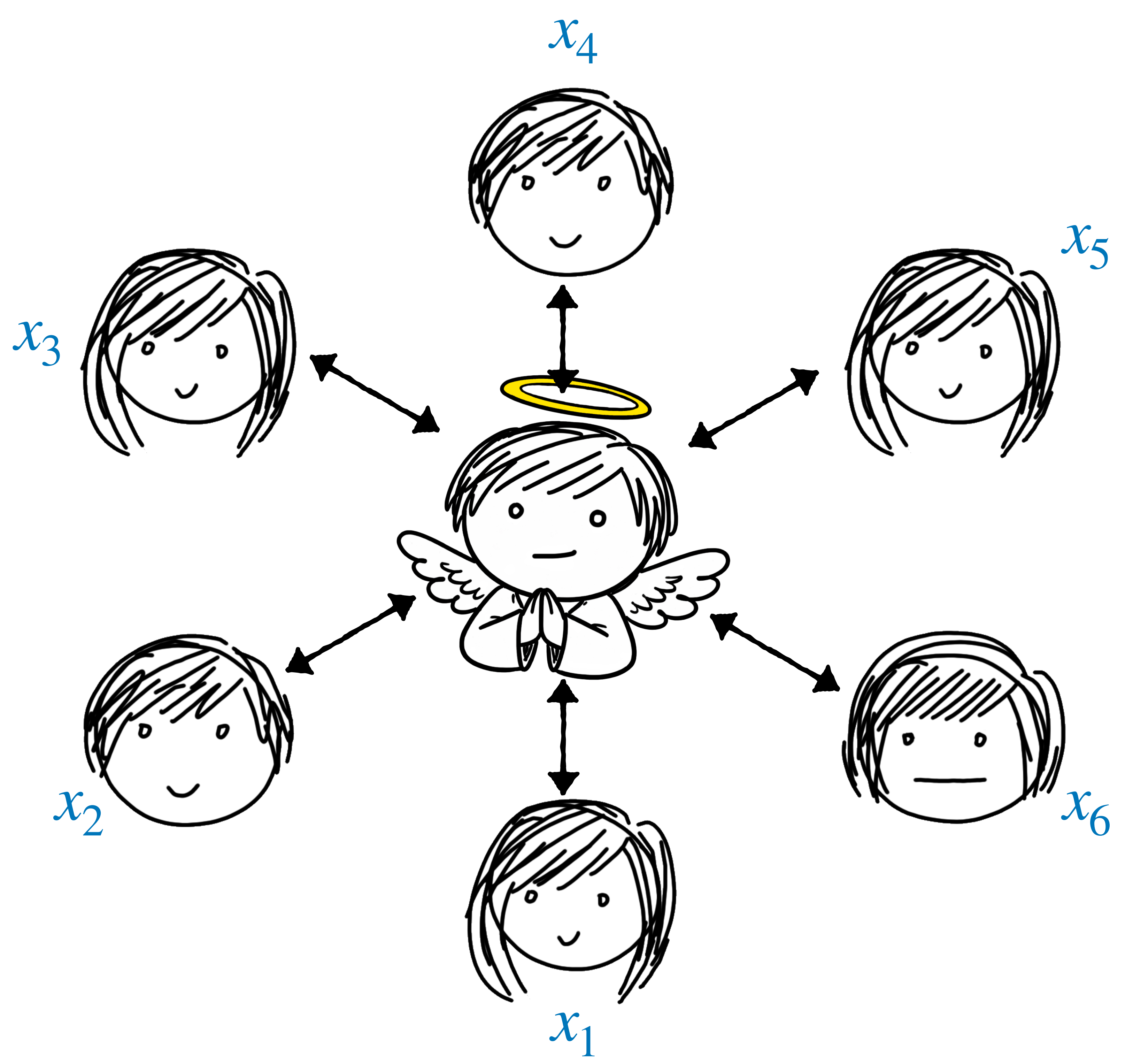
"name"

global parameters
(not party inputs)
omitted when clear

Functionality $\mathcal{F}_{\text{sum}}(n, M)$

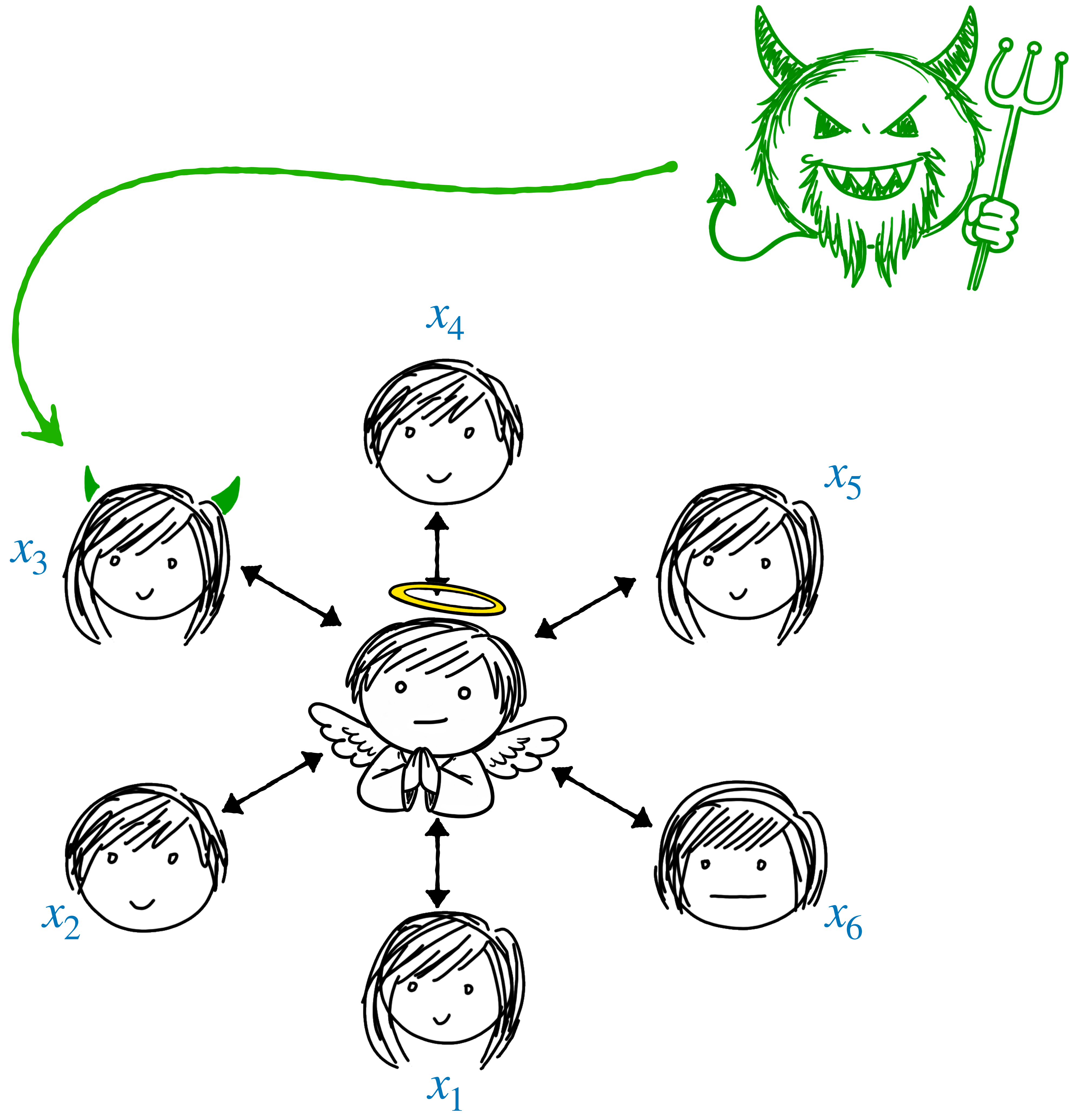
1. Receive $x_i \in \mathbb{N}$ such that $x_i \leq M/n$ from P_i for every $i \in [n]$.
2. Compute $y = \sum_{i \in [n]} x_i$.
3. Send y to P_i for every $i \in [n]$.

can communicate like a party
(later we will see that there is more nuance)



Example: n-Party Sum

- **Missing Piece #2:** how can we construct a simulator $\mathscr{S}$ that mimics he output of $\mathscr{A}$ for *any* $\mathscr{A}$ (semi-honest, one static corruption).
- Remember our *quantifier order*. Since we insisted $\forall\ \mathscr{A}\ \exists\ \mathscr{S}$, we can define an $\mathscr{S}$ that depends upon the code of $\mathscr{A}$.
- These are all computer programs, so $\mathscr{S}$ can run $\mathscr{A}$ "in its head" (i.e. as a subroutine)!
- If $\mathscr{S}$ can *trick* $\mathscr{A}$ into thinking it's in the real-world experiment, then $\mathscr{A}$ will output exactly what it would output in the real world. $\mathscr{S}$ can forward $\mathscr{A}$'s output to $\mathscr{D}$.

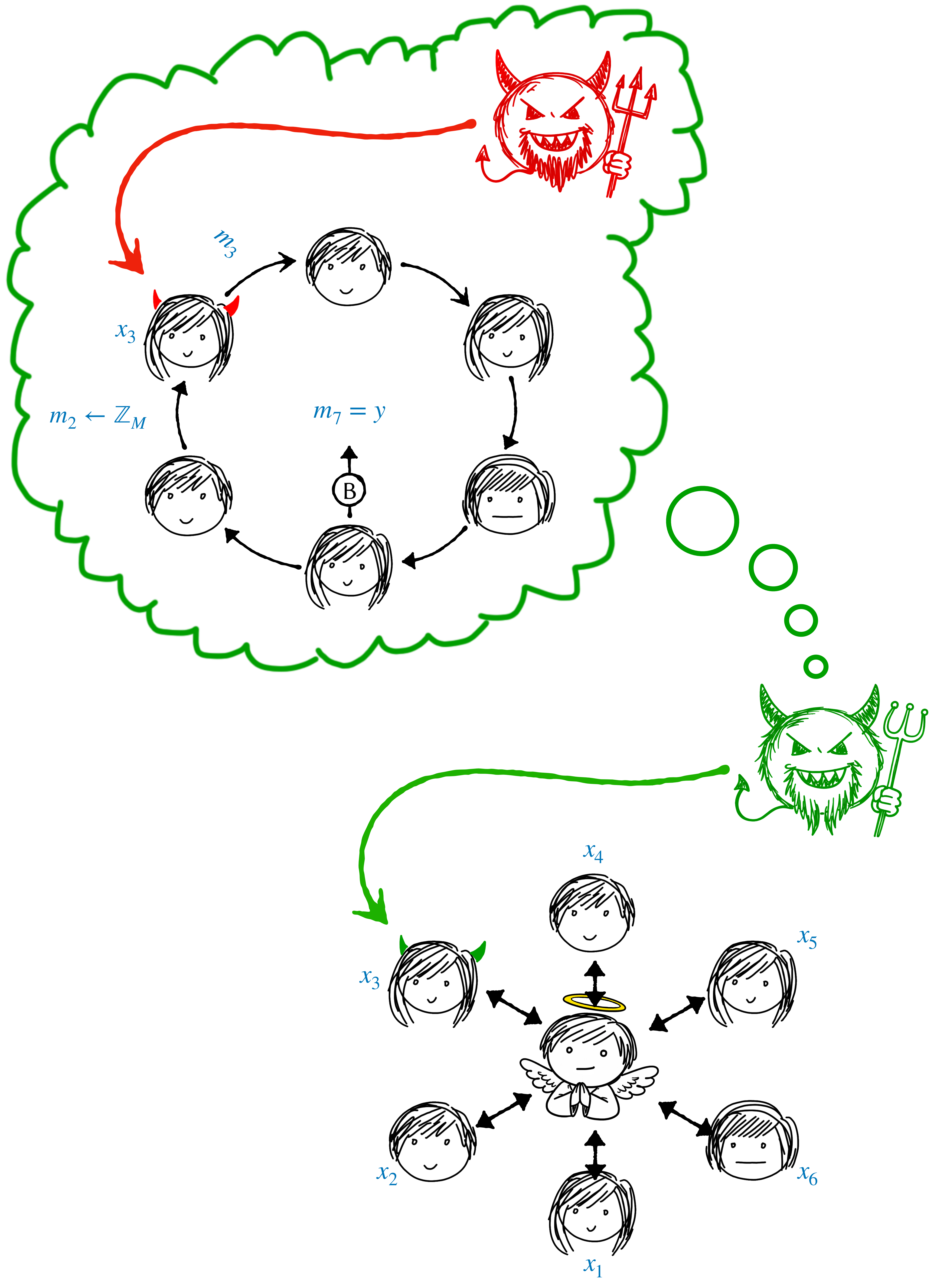


Example: n-Party Sum

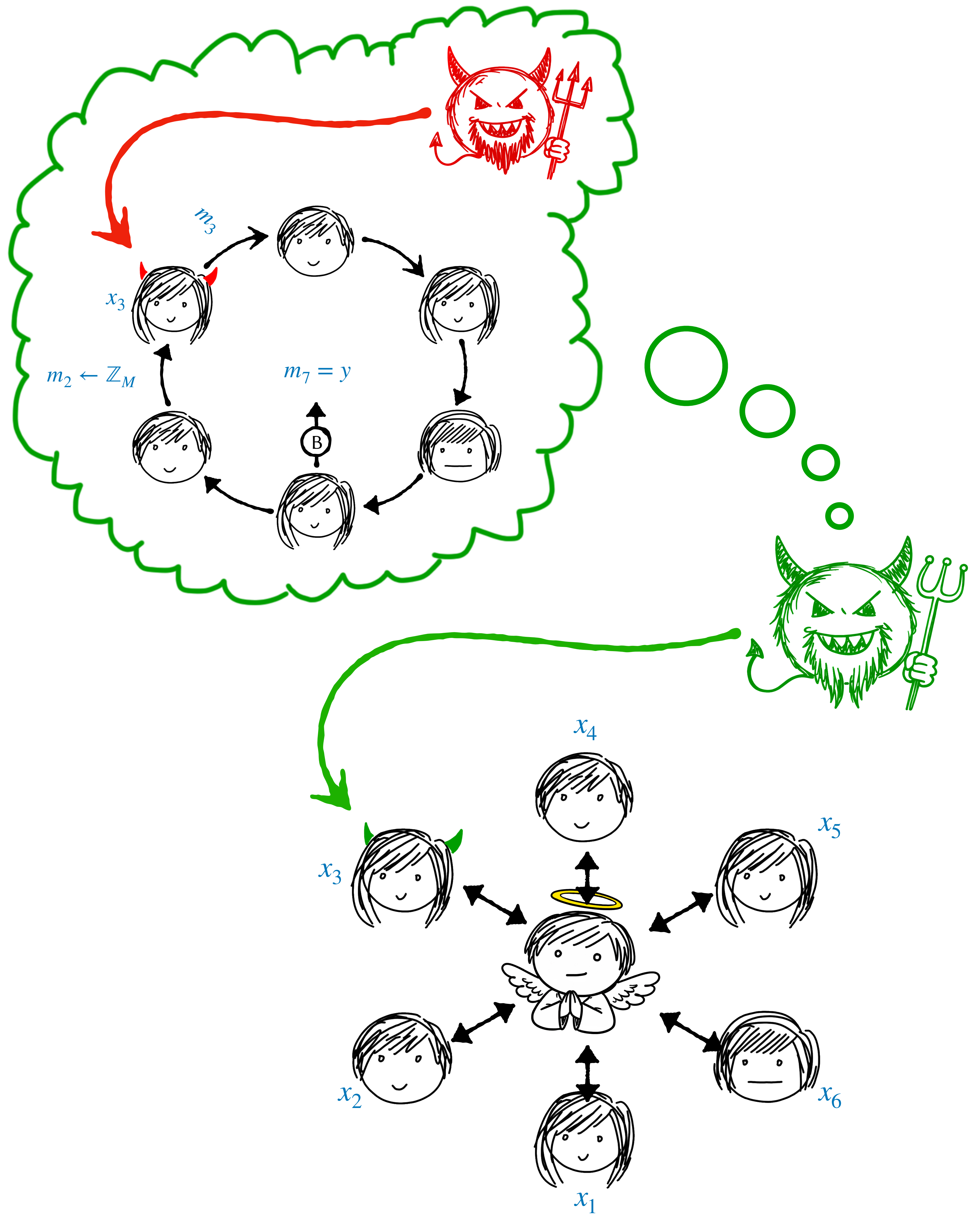
- **Missing Piece #2:** how can we construct a simulator $\mathscr{S}$ that mimics he output of $\mathscr{A}$ for *any* $\mathscr{A}$ (semi-honest, one static corruption).

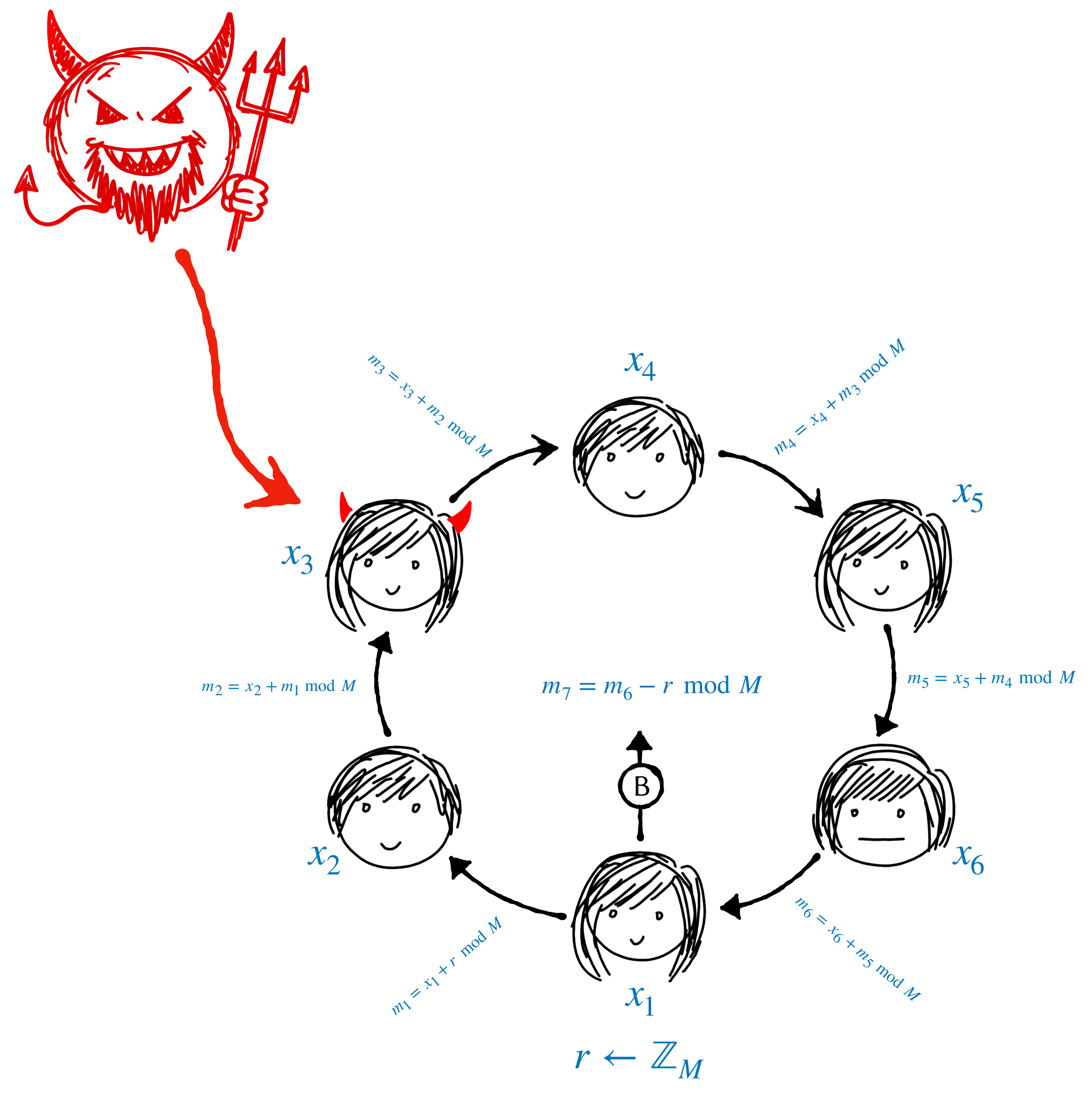
Simulator $\mathscr{S}(n)$

1. Emulate $\mathscr{A}$ in an interaction with the protocol.
2. When $\mathscr{A}$ tries to corrupt emulated P_i, corrupt P_i in the ideal world and learn its input x_i. Make x_i the input of the emulated P_i before corruption occurs.
3. Send x_i to $\mathscr{F}_{\text{sum}}$ on behalf of P_i.
4. Receive y from $\mathscr{F}_{\text{sum}}$ on behalf of P_i.
5. Use x_i and y to continue emulating the real world.
6. When $\mathscr{A}$ halts, output what it does.

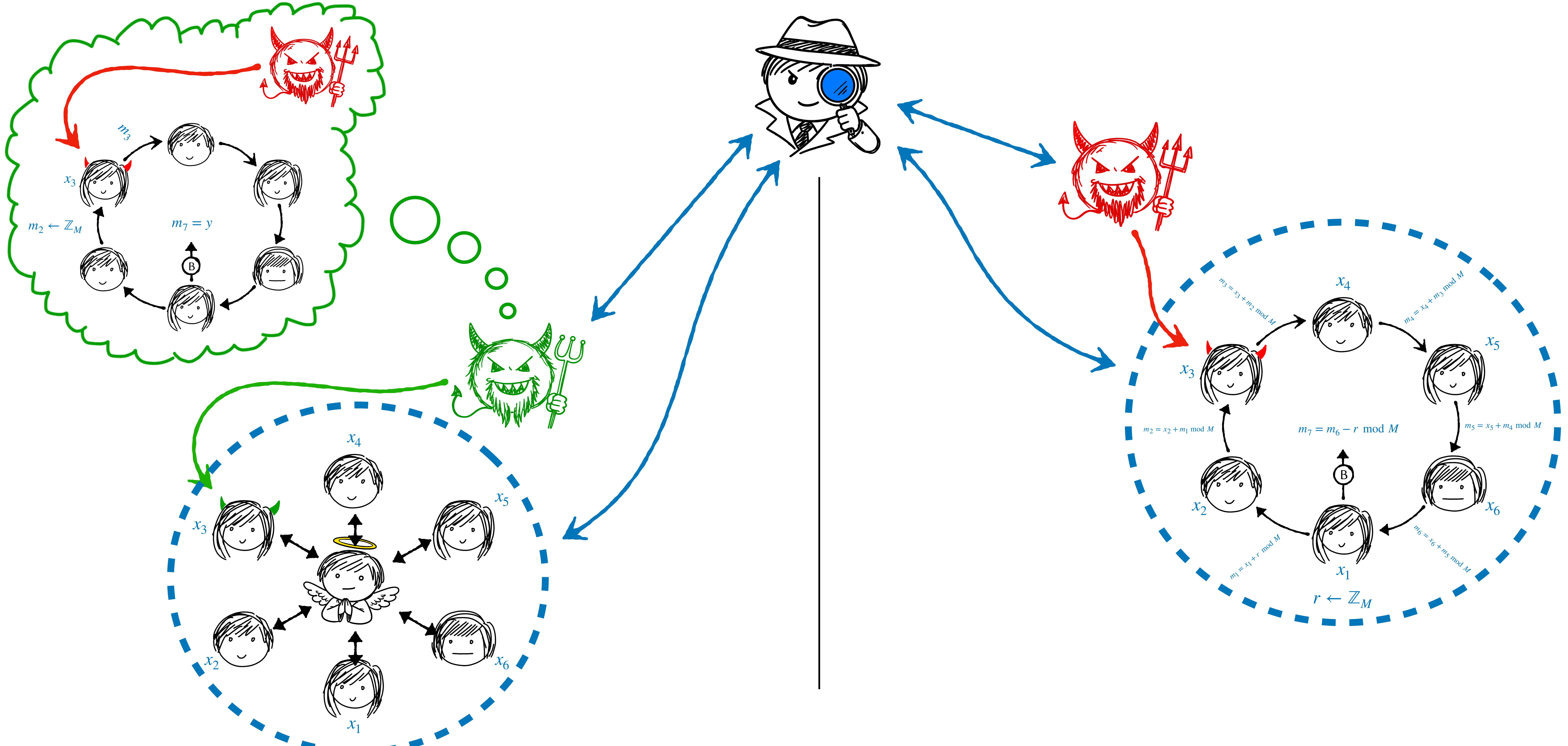


Example: n-Party Sum, Ideal vs Real Worlds

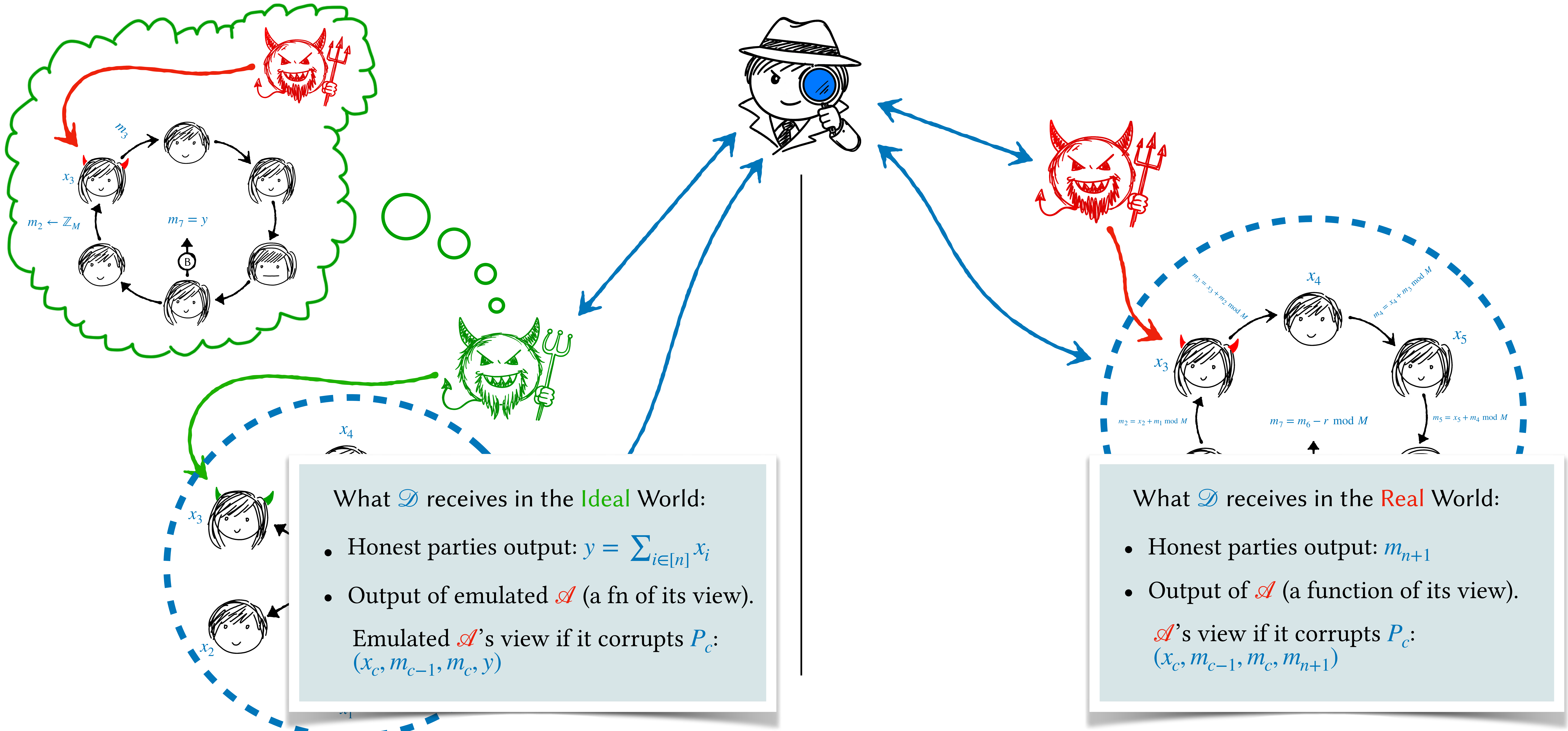



x_4
$m_3 = x_3 + m_2 \bmod M$
$m_4 = x_4 + m_3 \bmod M$
x_3
x_5
$m_2 = x_2 + m_1 \bmod M$
$m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$
B
x_2
x_6
$m_1 = x_1 + r \bmod M$
$m_6 = x_6 + m_5 \bmod M$
x_1
$r \leftarrow \mathbb{Z}_M$

Example: *n*-Party Sum, Ideal vs Real Worlds


m_3
x_3
$m_2 \leftarrow \mathbb{Z}_M$
$m_7 = y$
B
x_4
x_5
x_3
x_2
x_6
x_1
x_4
$m_3 = x_3 + m_2 \bmod M$
$m_4 = x_4 + m_3 \bmod M$
x_5
x_3
$m_2 = x_2 + m_1 \bmod M$
$m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$
B
x_2
x_6
$m_1 = x_1 + r \bmod M$
$m_6 = x_6 + m_5 \bmod M$
x_1
$r \leftarrow \mathbb{Z}_M$

Example: n-Party Sum, Ideal vs Real Worlds



m_3
x_3
$m_2 \leftarrow \mathbb{Z}_M$
$m_7 = y$
B
x_4
x_3
x_2
x_4
x_3
x_5
$m_3 = x_3 + m_2 \bmod M$
$m_4 = x_4 + m_3 \bmod M$
$m_2 = x_2 + m_1 \bmod M$
$m_7 = m_6 - r \bmod M$
$m_5 = x_5 + m_4 \bmod M$

What $\mathscr{D}$ receives in the Ideal World:

- Honest parties output: $y = \sum_{i \in [n]} x_i$
- Output of emulated $\mathscr{A}$ (a fn of its view).

Emulated $\mathscr{A}$'s view if it corrupts P_c: (x_c, m_{c-1}, m_c, y)

What $\mathscr{D}$ receives in the Real World:

- Honest parties output: m_{n+1}
- Output of $\mathscr{A}$ (a function of its view).

$\mathscr{A}$'s view if it corrupts P_c: $(x_c, m_{c-1}, m_c, m_{n+1})$

Example: n-Party Sum, Ideal vs Real Worlds

What $\mathscr{D}$ receives in the Ideal World:

- Honest parties output: $y = \sum_{i\in[n]} x_i$
- Output of emulated $\mathscr{A}$ (a fn of its view).

  Emulated $\mathscr{A}$'s view if it corrupts P_c: (x_c, m_{c-1}, m_c, y)

What $\mathscr{D}$ receives in the Real World:

- Honest parties output: m_{n+1}
- Output of $\mathscr{A}$ (a function of its view).

  $\mathscr{A}$'s view if it corrupts P_c: $(x_c, m_{c-1}, m_c, m_{n+1})$

- Since $(x_1, \ldots, x_n)$ are determined by $\mathscr{D}$, they are distributed identically in both worlds.
- m_{c-1} is uniformly distributed in both worlds.
- m_c is deterministic given m_{c-1} and x_c and thus identically distributed in both worlds.
- By the *correctness* property of the protocol, $m_{n+1} = \sum_{i\in[n]} x_i$ in the real world, which is identically distributed to ideal-world y.
- Therefore the view of $\mathscr{A}$ is identically distributed in the two worlds, and so is its output.
- Therefore the view of $\mathscr{D}$ is identically distributed in the two worlds.
- Therefore,

$$\Pr\left[\begin{array}{l}\mathscr{D}\text{ guesses correctly}\\ \text{given random world}\end{array}\right] = \boxed{\frac{1}{2}}$$

- This holds $\forall\, \mathscr{A}\ \forall\, \mathscr{D}$, regardless of computation power, so we have...

Theorem 1. *The sum protocol* perfectly *realizes $\mathscr{F}_{\text{sum}}$ in the presence of a semi-honest adversary that statically corrupts up to one party.*

Take a big breath. Let it sink in.
This will be easy for you by the End.
Let’s generalize what we just saw.

Generalizing the Functionality

Functionality $\mathcal{F}_{\mathsf{sum}}(n, M)$

1. Receive $x_i \in \mathbb{N}$ such that $x_i \leq M/n$ from P_i for every $i \in [n]$.
2. Compute $y = \sum_{i\in[n]} x_i$.
3. Send y to P_i for every $i \in [n]$.

- Functionalities can contain any arbitrary code. They can react dynamically to new instructions or data from parties while their computations are in progress, keep persistent state between invocations, or even communicate with $\mathcal{S}$ directly.
- However, in this class we care about the above pattern: receive inputs from everyone, compute a function, deliver outputs to everyone. This is called *Secure Function Evaluation* (SFE).

- Rather than specify every instance of this pattern individually, we will give a general SFE functionality. This is the main functionality we will care about!
- Suppose we have n parties, and every P_i for $i \in [n]$ has an input in the set $\mathcal{X}_i$ and expects an output in the set $\mathcal{Y}_i$.
- An n-ary function f maps n inputs to n outputs. If the i^{th} input is in $\mathcal{X}_i$ and the i^{th} output is in $\mathcal{Y}_i$, we denote this with $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$.

Functionality $\mathcal{F}_{\mathsf{SFE}}(n, f, \mathcal{X}_1, \ldots, \mathcal{X}_n)$

1. Receive $x_i \in \mathcal{X}_i$ from P_i for every $i \in [n]$.
2. Compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
3. Send y_i to P_i for every $i \in [n]$.

Generalizing the Functionality

- Rather than specify every instance of this pattern individually, we will give a general SFE functionality. This is the main functionality we will care about!
- Suppose we have n parties, and every P_i for $i \in [n]$ has an input in the set $\mathcal{X}_i$ and expects an output in the set $\mathcal{Y}_i$.
- An n-*ary* function f maps n inputs to n outputs. If the ith input is in $\mathcal{X}_i$ and the ith output is in $\mathcal{Y}_i$, we denote this with $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$.

Functionality $\mathcal{F}_{\mathsf{SFE}}(n, f, \mathcal{X}_1, \ldots, \mathcal{X}_n)$

1. Receive $x_i \in \mathcal{X}_i$ from P_i for every $i \in [n]$.
2. Compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
3. Send y_i to P_i for every $i \in [n]$.

- Notice that $\mathcal{F}_{\mathsf{SFE}}$ guarantees correctness, privacy, input independence, and output delivery!
- We will often say that a protocol π *securely computes* or *securely evaluates* $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$. This usually means that π *realizes* $\mathcal{F}_{\mathsf{SFE}}(n, f, \mathcal{X}_1, \ldots, \mathcal{X}_n)$.
- **More Notation:** The set of bit-strings of length m is denoted $\{0,1\}^m$, and the set of bit-strings of any length is $\{0,1\}^*$. The empty set is $\emptyset$, the empty string is λ, and an n-ary function with unrestricted domain and range is denoted $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$
- **Note:** many of your homework problems will be of the form *"Construct a protocol that securely computes f and prove that it is secure."* You have now seen enough to start solving these kinds of problems!

Generalizing the Functionality

For Example, we just proved that the sum protocol securely computes $f(x_1, \ldots, x_n) = (y, \ldots, y)$ where each $\mathcal{X}_i = \left[0, \lfloor M/n \rfloor + 1\right]$ and $y = \sum_{i \in [n]} x_i$.

More examples of functions that we can imagine security computing:

- Boolean AND: $f(x_1, \ldots, x_n) = (y, \ldots, y)$ where each $\mathcal{X}_i = \{0,1\}$ and $y = x_1 \wedge \ldots \wedge x_n$.
- Boolean XOR: $f(x_1, \ldots, x_n) = (y, \ldots, y)$ where each $\mathcal{X}_i = \{0,1\}$ and $y = x_1 \oplus \ldots \oplus x_n$.
- Asymmetric functions, e.g. $f(x_1, \ldots, x_n) = (y_1, y_2, \lambda, \ldots, \lambda)$ where each $\mathcal{X}_i = \{0,1\}$ and $y_1 = x_1 \oplus x_2$ and $y_2 = x_1 \wedge x_3$. It's important which party supplies which input!
- $f(S_1, S_2) = (S_1 \cap S_2, |S_1 \cap S_2|)$ where $\mathcal{X}_1 = \mathcal{X}_2 = \mathcal{U}$ and $\mathcal{U}$ is some *universe*.
  Q:What is the "task name" for this function? We've seen it before!

Generalizing the Functionality

Functionalities can also be randomized!

In the context of Secure Function Evaluation, this implies that there is an $(n+1)^{\text{st}}$ input to f that is uniformly distributed over $\{0,1\}^*$. This input is used as a random tape (i.e. a sequence of random coin flips) by the code of f.

Usually this extra input is implicit, but sometimes it will be important to specify exactly which random tape was used. In this case we will write $f(x_1, \ldots, x_n; r)$ for random tape r.

More examples of *randomized functions* that we can imagine security computing:

- Coin Tossing: $f(\lambda, \ldots, \lambda) = (b, \ldots, b)$ where $b \leftarrow \{0,1\}$.
- Leader Election: $f(\lambda, \ldots, \lambda) = (i, \ldots, i)$ where $i \leftarrow [n]$.
- Blind Encryption: $f(m, \mathsf{sk}) = (\mathsf{Enc}_{\mathsf{sk}}(m), \lambda)$ where Enc is an encryption function.

Generalizing the View of $\mathscr{A}$

When $\mathscr{A}$ is *semi-honest* the corrupted parties follow the protocol instructions.

Without loss of generality, we can assume that $\mathscr{A}$ always outputs *everything* it sees in the protocol. **Q: why is this?**

So, what does $\mathscr{A}$ see in the execution of an arbitrary protocol π?

For every $i \in [n]$, let view_i denote the view of P_i, which contains

- The input x_i of P_i.
- The random coins of P_i, denoted r_i.
- The messages received by P_i during the protocol.

Note that:

- Messages sent by P_i are a (deterministic) function of its view at the time of sending.
- The output of P_i is a function of its view at the end of the protocol.

Generalizing the View of $\mathscr{A}$

Suppose that $\mathscr{A}$ statically semi-honestly corrupts up to t parties. Let $I = \{i_1, \ldots, i_t\}$ index the set of corrupted parties, and let $\mathsf{view}_I = \{\mathsf{view}_{i_1}, \ldots, \mathsf{view}_{i_t}\}$ be the views of those parties.

Typically, view_I represents single, specific execution of the protocol involving a specific, fixed set of random tapes for all participants. We denote the corresponding random variable (in which the random coins are uniformly distributed and all other values are derived from them as expected) using VIEW_I.

Similarly, we use output_i to denote the output of P_i when the protocol ends, and $\mathsf{output}_\pi = \{\mathsf{output}_1, \ldots, \mathsf{output}_n\}$ to denote the outputs of all parties. OUTPUT_π, then, is the corresponding random variable.

A Simplified Definition for Semi-Honest $\mathscr{A}$

Definition 2. Perfect Semi-Honest Secure Function Evaluation (Simplified).

Let $n, t \in \mathbb{N}$ such that $t < n$, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be an n-ary function, and let π be a n-party protocol.

We say that π *perfectly securely computes* f in the presence of a semi-honest adversary that statically corrupts up to t parties if there exists a *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every input vector $\vec{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ it holds that

$$\left(\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right), f(\vec{x})\right) \equiv (\mathsf{VIEW}_I, \mathsf{OUTPUT}_\pi)$$

Simulated view of corrupt parties. Computed using corrupt party inputs and outputs

Real view of corrupt parties in the protocol.

A Simplified Definition for Semi-Honest $\mathscr{A}$

Definition 2. Perfect Semi-Honest Secure Function Evaluation (Simplified).

Let $n, t \in \mathbb{N}$ such that $t < n$, let $f : \mathscr{X}_1 \times \ldots \times \mathscr{X}_n \to \mathscr{Y}_1 \times \ldots \times \mathscr{Y}_n$ be an n-ary function, and let π be a n-party protocol.

We say that π *perfectly securely computes* f in the presence of a semi-honest adversary that statically corrupts up to t parties if there exists a *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every input vector $\vec{x} \in \mathscr{X}_1 \times \ldots \times \mathscr{X}_n$ it holds that

$$\left(\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right), f(\vec{x})\right) \equiv (\mathsf{VIEW}_I, \mathsf{OUTPUT}_\pi)$$

Ideal computation

Output of all parties in real protocol

A Simplified Definition for Semi-Honest $\mathscr{A}$

Definition 2. Perfect Semi-Honest Secure Function Evaluation (Simplified).

Let $n, t \in \mathbb{N}$ such that $t < n$, let $f : \mathscr{X}_1 \times \ldots \times \mathscr{X}_n \to \mathscr{Y}_1 \times \ldots \times \mathscr{Y}_n$ be an n-ary function, and let π be a n-party protocol.

We say that π *perfectly securely computes* f in the presence of a semi-honest adversary that statically corrupts up to t parties if there exists a *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every input vector $\vec{x} \in \mathscr{X}_1 \times \ldots \times \mathscr{X}_n$ it holds that

$$\left(\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right), f(\vec{x})\right) \underset{\substack{\uparrow \\ \text{Identically Distributed}}}{\equiv} (\mathsf{VIEW}_I, \mathsf{OUTPUT}_\pi)$$

A Simplified Definition for Semi-Honest $\mathscr{A}$

Definition 2. Perfect Semi-Honest Secure Function Evaluation (Simplified).

Let $n, t \in \mathbb{N}$ such that $t < n$, let $f : \mathscr{X}_1 \times \ldots \times \mathscr{X}_n \to \mathscr{Y}_1 \times \ldots \times \mathscr{Y}_n$ be an n-ary function, and let π be a n-party protocol.

We say that π *perfectly securely computes* f in the presence of a semi-honest adversary that statically corrupts up to t parties if there exists a *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every input vector $\vec{x} \in \mathscr{X}_1 \times \ldots \times \mathscr{X}_n$ it holds that

$$\left(\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right), f(\vec{x})\right) \equiv (\mathsf{VIEW}_I, \mathsf{OUTPUT}_\pi)$$

Sanity Check: if Definition 2 holds, can we build $\mathscr{S}$ so that the full security definition holds too with respect to $\mathscr{F}_{\mathsf{SFE}}$?

An Even Simpler Def for Deterministic f!

Definition 3. Perfect Semi-Honest Secure *Deterministic* Function Evaluation.

Let $n, t \in \mathbb{N}$ such that $t < n$, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be an n-ary function, **(which does not use any randomness)** and let π be a n-party protocol.

We say that π *perfectly securely computes* f in the presence of a semi-honest adversary that statically corrupts up to t parties if

1. For every input vector $\vec{x} \in \mathcal{X}_1 \times \ldots \mathcal{X}_n$ it holds that $f(\vec{x}) \equiv \mathsf{OUTPUT}_\pi$.
2. There exists a *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every input vector $\vec{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ it holds that

$$\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right) \equiv \mathsf{VIEW}_I$$

Why only for Deterministic f?

Consider the randomized 2-ary function $f : (\lambda, \lambda) \to (b, \lambda)$ where $b \leftarrow \{0,1\}$ along with the following 2-party protocol π:

P_1 samples $b \leftarrow \{0,1\}$ uniformly, sends b to P_2, and outputs b. P_2 outputs nothing.

Claim: *π securely computes f per Definition 3.*

Proof Sketch.

1. $\mathsf{OUTPUT}_\pi = (b, \lambda)$ where $b \leftarrow \{0,1\}$, thus $f(\lambda, \lambda) \equiv \mathsf{OUTPUT}_\pi$
2. Let Sim be defined such that:
   - If is P_1 corrupted, Sim sets the random tape of P_1 such that $b \leftarrow \{0,1\}$.
   - If is P_2 corrupted, Sim sets the received message of P_2 to be $b \leftarrow \{0,1\}$.

   In both cases, $\mathsf{Sim}\left(I, \vec{x}_I, f_I(\vec{x})\right) \equiv \mathsf{VIEW}_I$ ∎

Why only for Deterministic f?

Consider the randomized 2-ary function $f : (\lambda, \lambda) \rightarrow (b, \lambda)$ where $b \leftarrow \{0,1\}$ along with the following 2-party protocol π:

P_1 samples $b \leftarrow \{0,1\}$ uniformly, sends b to P_2, and outputs b. P_2 outputs nothing.

But Wait! This protocol *shouldn't* be secure! It leaks b when f doesn't specify to do so!

Under Definition 2, we consider the *joint distribution* of all outputs and corrupt views, which means that the view produced by Sim when P_2 is corrupt has to be *consistent* with the output of P_1.

But when P_2 is corrupt, the output of P_1 is randomized and Sim doesn't learn it, so the best-case scenario is that Sim has an error probability of ½.

Security definitions are subtle!
Definitions 2 and 3 are equivalent *only* if f is deterministic.

How Do Parties Communicate in the Real World?

Secure Communication

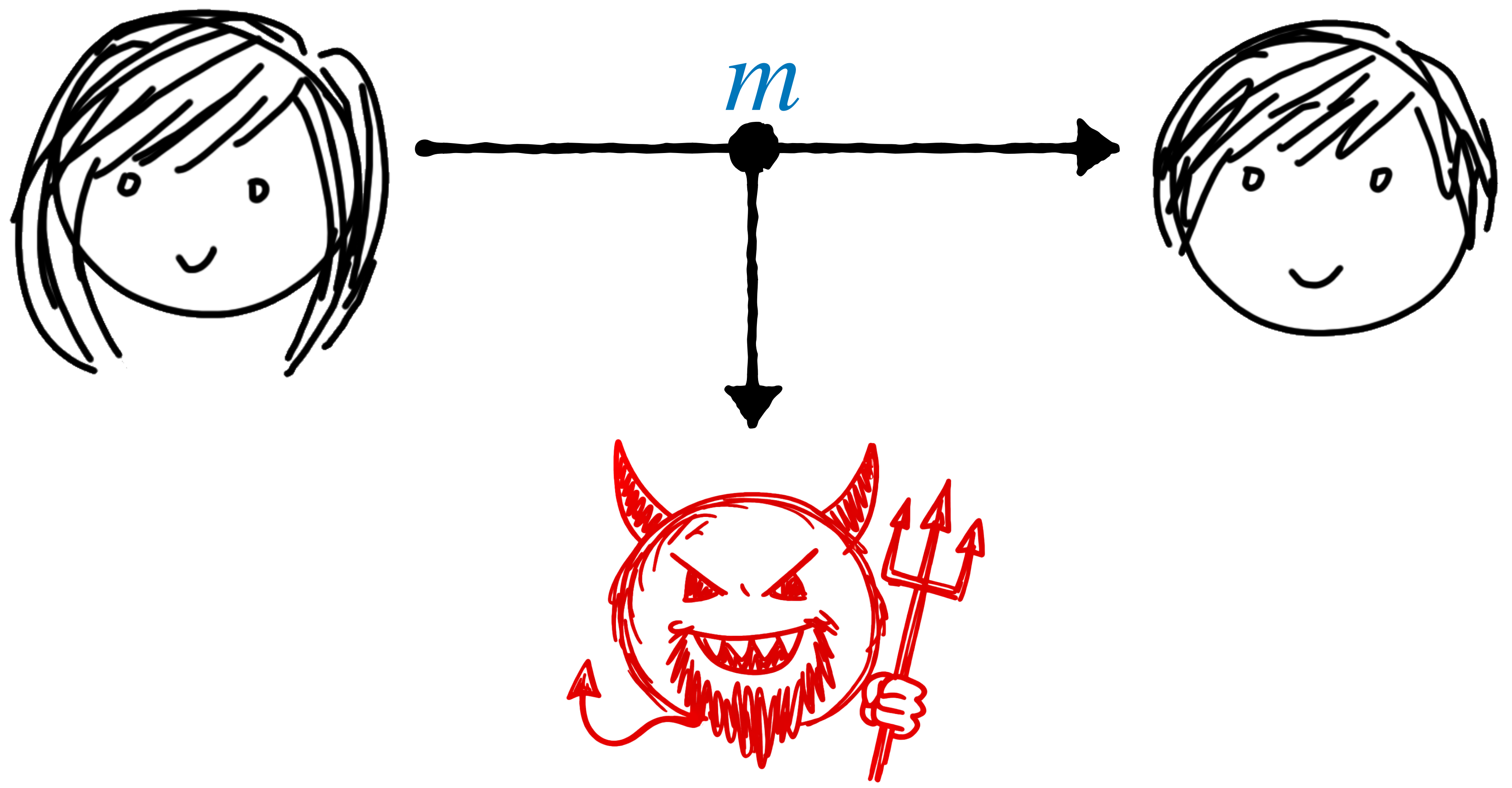
Suppose Alice wishes to send a message to Bob. We say she uses a *channel*.

In the real world their message must traverse a network of *many* computers that they do not own or trust. It is likely that they do not trust or even know the person to whom each node belongs.

We model this by assuming that $\mathscr{A}$ controls all real-world communication by default. $\mathscr{A}$ is free to read and alter communication between honest parties.

Unsurprisingly, not much can be achieved in this *insecure channel* model.

To help us make progress, we will introduce two more-powerful channel types.



Secure Communication

Authenticated Channels:

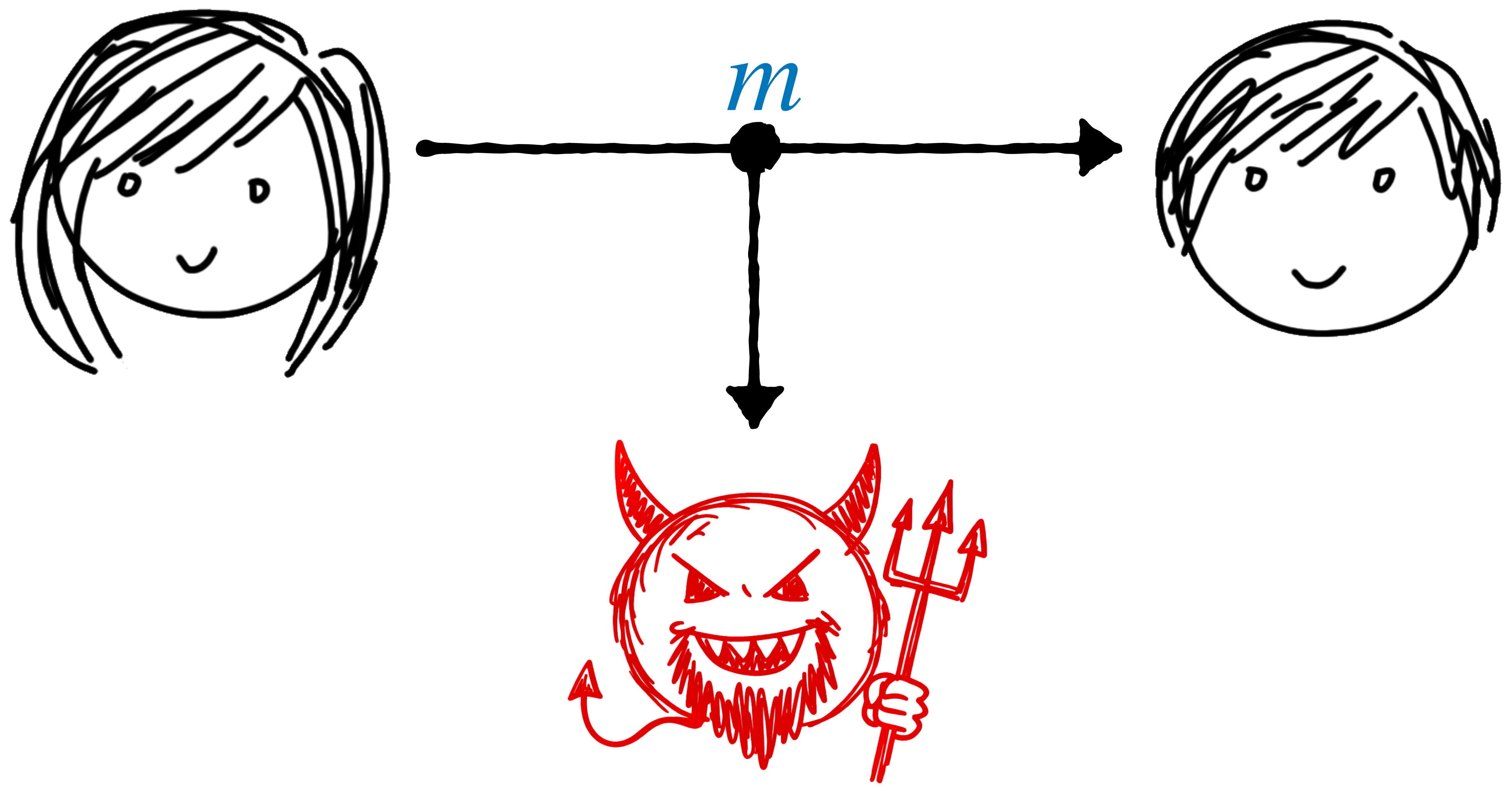
These channels allow the adversary to learn the *content* of the message, but not to alter it (including by silently dropping any part of it or injecting something new).

- If $\mathscr{A}$ is semi-honest, this comes *for free* because $\mathscr{A}$ follows the (network) protocol!
- If $\mathscr{A}$ is malicious, we can do more work to build an authenticated channel. We need a way to determine if $\mathscr{A}$ has altered messages on the wire. We'll come back to this!

Secure Channels:

These channels are authenticated and *private.* They allow $\mathscr{A}$ to learn only (an upper bound on) the length of m.

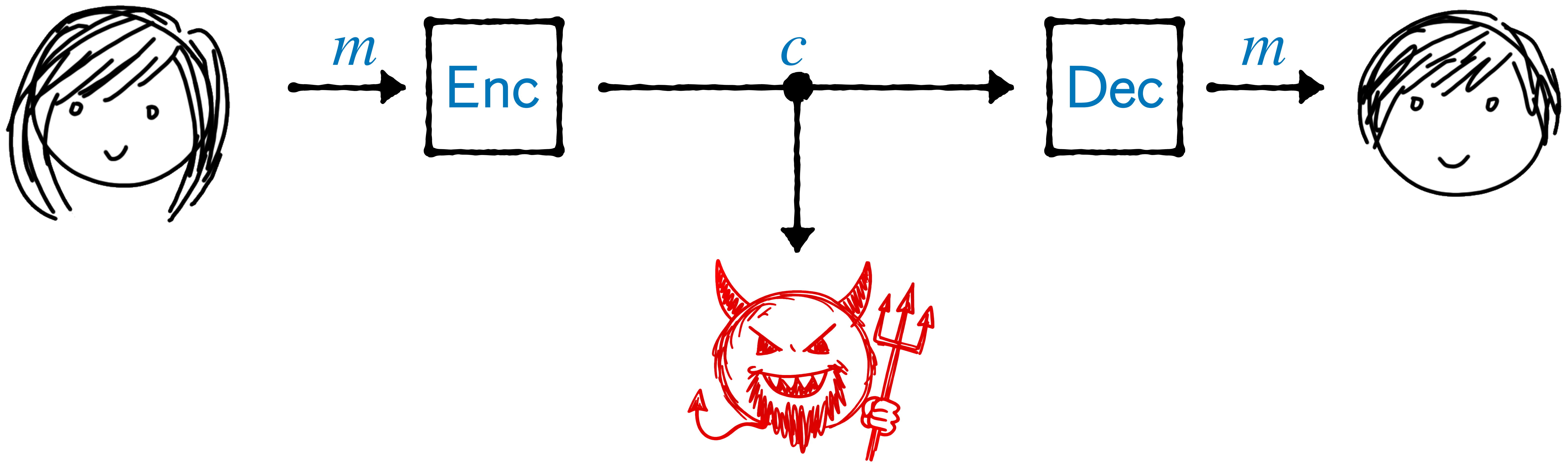
How to add privacy to an authenticated channel?



Symmetric Encryption

We begin by giving Alice and Bob matching encryption and decryption algorithms, Enc and Dec respectively, which can transform the message m into a ciphertext c. We don't want $\mathscr{A}$ to be able to decrypt, so Bob's decryption capability must be tied to a secret that only he and Alice know.

Q: Is it reasonable for this secret to be the algorithm Dec?

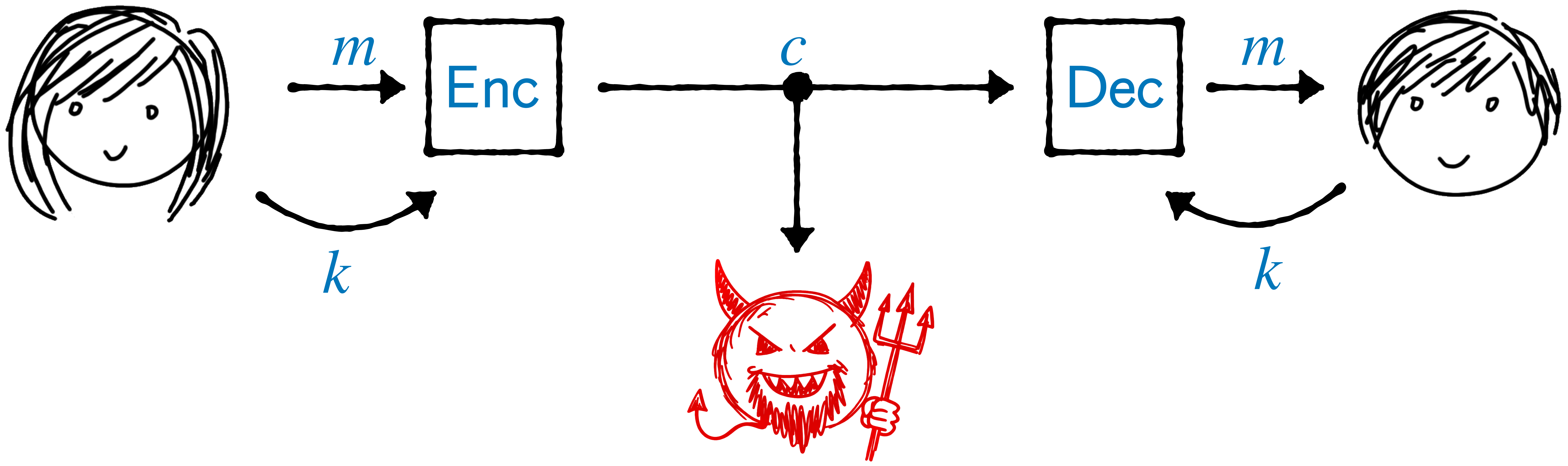


Kerckhoff's Principle

Phrased by Claude Shannon as: *"the enemy knows the system".*

We should assume all *algorithm descriptions* are public. We can only keep some small piece of agreed-upon *data* secret. We call this the *secret key* k.

Q: Suppose Alice and Bob use a fixed value of k. What happens when we quantify over all adversaries $\mathcal{A}$? How can we get around this problem?



Symmetric-Key Encryption (SKE)

Definition 4. Syntax for SKE

For $\kappa \in \mathbb{N}$, let $\mathscr{M}_\kappa$ be a message space, $\mathscr{K}_\kappa$ be a key space, and $\mathscr{C}_\kappa$ be a ciphertext space. A *Symmetric-Key Encryption Scheme* is a trio of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- Given security parameter 1^κ, randomized key-generation algorithm $k \leftarrow \mathsf{Gen}(1^\kappa)$ outputs some secret key $k \in \mathscr{K}_\kappa$.

Unary representation, i.e. κ ones.
Important in the second half of the course...

Symmetric-Key Encryption (SKE)

Definition 4. Syntax for SKE

For $\kappa \in \mathbb{N}$, let $\mathscr{M}_\kappa$ be a message space, $\mathscr{K}_\kappa$ be a key space, and $\mathscr{C}_\kappa$ be a ciphertext space. A *Symmetric-Key Encryption Scheme* is a trio of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

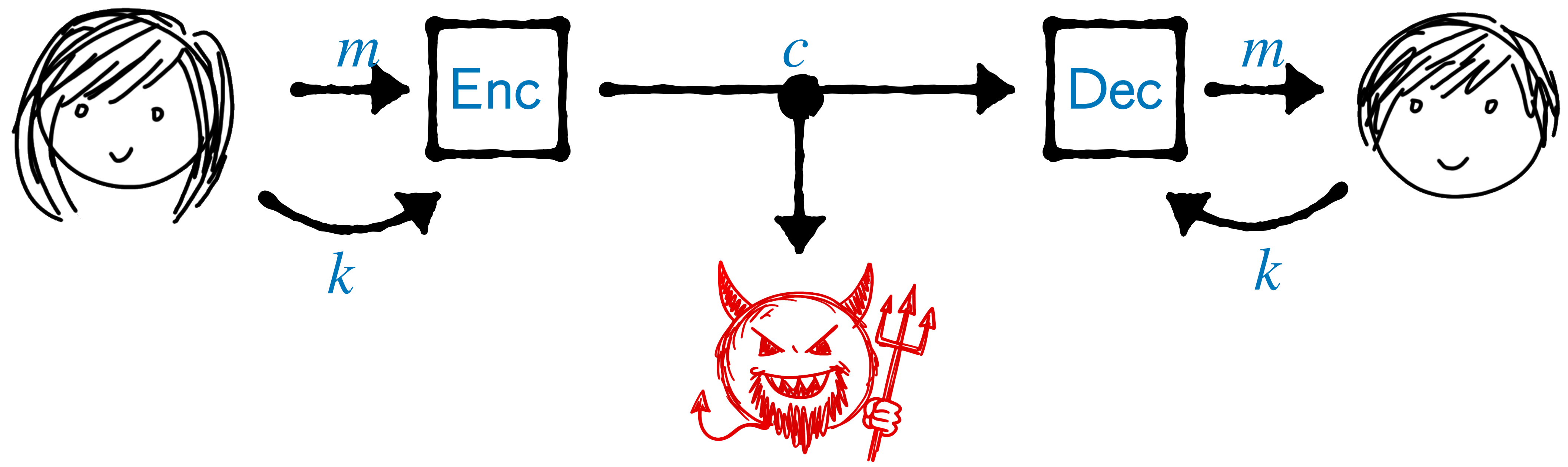
- Given security parameter 1^κ, randomized key-generation algorithm $k \leftarrow \mathsf{Gen}(1^\kappa)$ outputs some secret key $k \in \mathscr{K}_\kappa$.
- The encryption algorithm $c \leftarrow \mathsf{Enc}_k(m)$ outputs a (possibly randomized) ciphertext $c \in \mathscr{C}_\kappa$ given a message $m \in \mathscr{M}_\kappa$ and key $k \in \mathscr{K}_\kappa$.
- The decryption algorithm $m = \mathsf{Dec}_k(c)$ outputs a deterministic message $m \in \mathscr{M}_\kappa$ given a ciphertext $c \in \mathscr{C}_\kappa$ and key $k \in \mathscr{K}_\kappa$.

Definition 5. Correctness for SKE

$$\forall \kappa \in \mathbb{N}\ \forall k \in \mathscr{K}_\kappa\ \forall m \in \mathscr{M}_\kappa,\ \mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$$

Security Definitions are Subtle

Earlier when we drew this picture, there was only one secret value. What was it?



Idea 1. $\mathscr{A}$ cannot learn k.

Problem. $\mathsf{Enc}_k(m) = m$ is a valid encryption scheme under this definition.

Idea 2. $\mathscr{A}$ cannot learn m.

Problem. What if $\mathscr{A}$ learns *half* of m.

Security Definitions are Subtle

Idea 1. $\mathscr{A}$ cannot learn k.

Problem. $\mathsf{Enc}_k(m) = m$ is a valid encryption scheme under this definition.

Idea 2. $\mathscr{A}$ cannot learn m.

Problem. What if $\mathscr{A}$ learns *half* of m.

Idea 3. $\mathscr{A}$ cannot learn any *bit* of m.

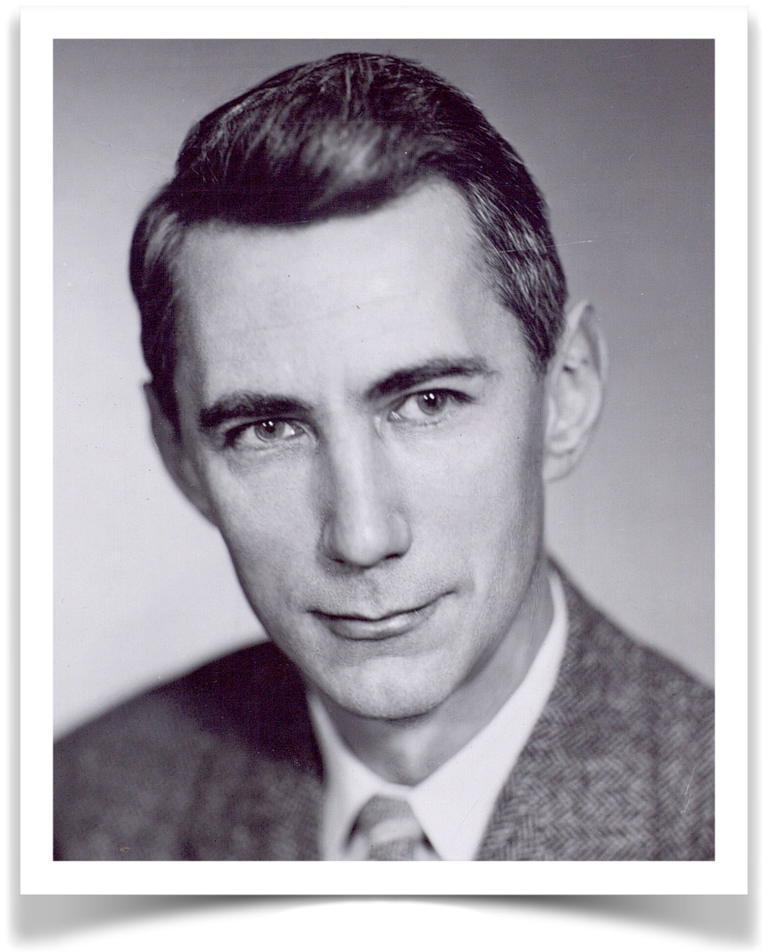
Problem. What if $\mathscr{A}$ learns $f(m)$ where f does not leak any individual bit (e.g. parity).

Idea 4. $\mathscr{A}$ cannot learn any *function* of m.

Problem. This is unachievable! What if $\mathscr{A}$ already knows a function of m?
(e.g. knows that c encrypts a day of the week. The last letter of m is always y).

Idea 5. $\mathscr{A}$ cannot learn any *new information* about m.

And now it's time to talk about Claude Shannon



Idea 5. $\mathscr{A}$ cannot learn any *new information* about m.

What does this mean?

CS4501 Cryptographic Protocols
Lecture 3: Simulation, Communication

https://jackdoerner.net/teaching/#2026/Spring/CS4501