



European Research Council
Established by the European Commission

Central European Conference on Cryptology
Budapest, June 19, 2025

MPC-hardness and applications

Stefan Dziembowski



This talk is based on

- J. Bormet, S. D., S. Faust, T. Lazurej, and M. Mielniczuk: **Strong Secret Sharing with Snitching**, CRYPTO 2025
- S. D., S. Faust, T. Lazurej, and M. Mielniczuk: **Secret Sharing with Snitching**, ACM CCS 2024
- S. D., S. Faust, and T. Lazurej: **Individual Cryptography**, CRYPTO 2023

Independent work:

- M. Kelkar, K. Babel, P. Daian, J. Austgen, V. Buterin, A. Juels: **Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets**. ACM CCS 2024

Plan

1. Introduction to different models in distributed cryptography
2. Secret Sharing with Snitching
3. Other applications of MPC-hardness
4. Extensions and research problems



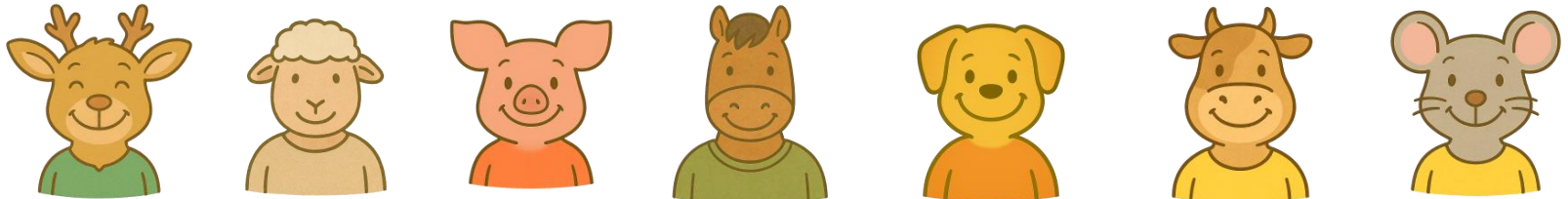
Distributed cryptography

Protocols between a **group of parties** that want to achieve a

common goal

even though they

don't trust each other.



Two approaches

- **classical cryptographic approach**
(MPCs, consensus, broadcast,...)
 - an active area since the **1980s**.



slightly different models

- **blockchain** – introduced in 2008 by an anonymous author

MPC = Multiparty Computations

f – publicly known function

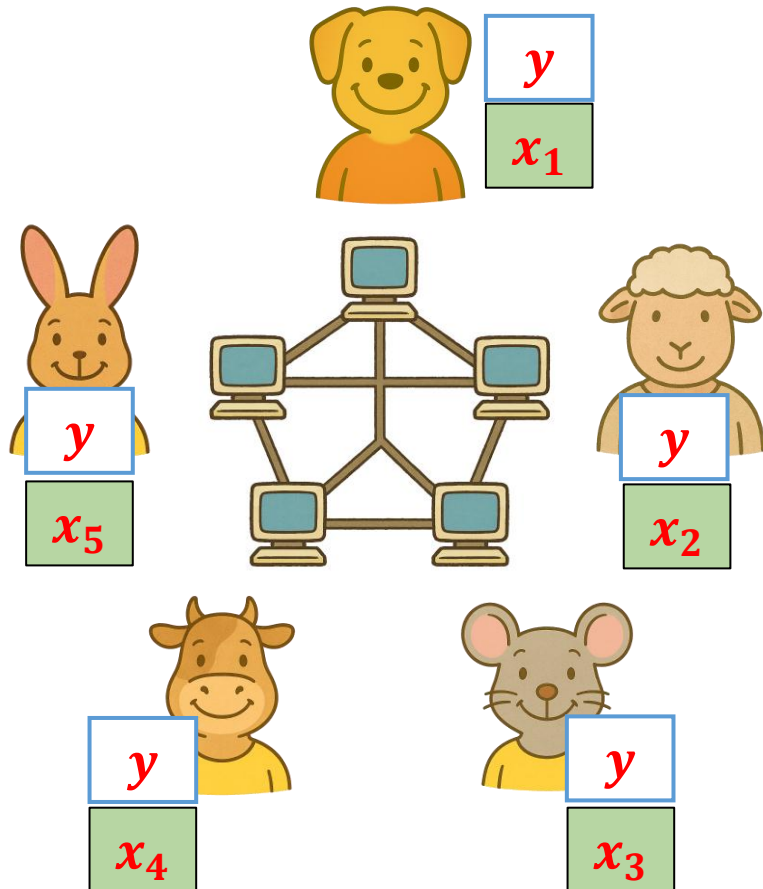
Protocols that allow a group of parties to **securely** evaluate

$$y := f(x_1, \dots, x_n)$$

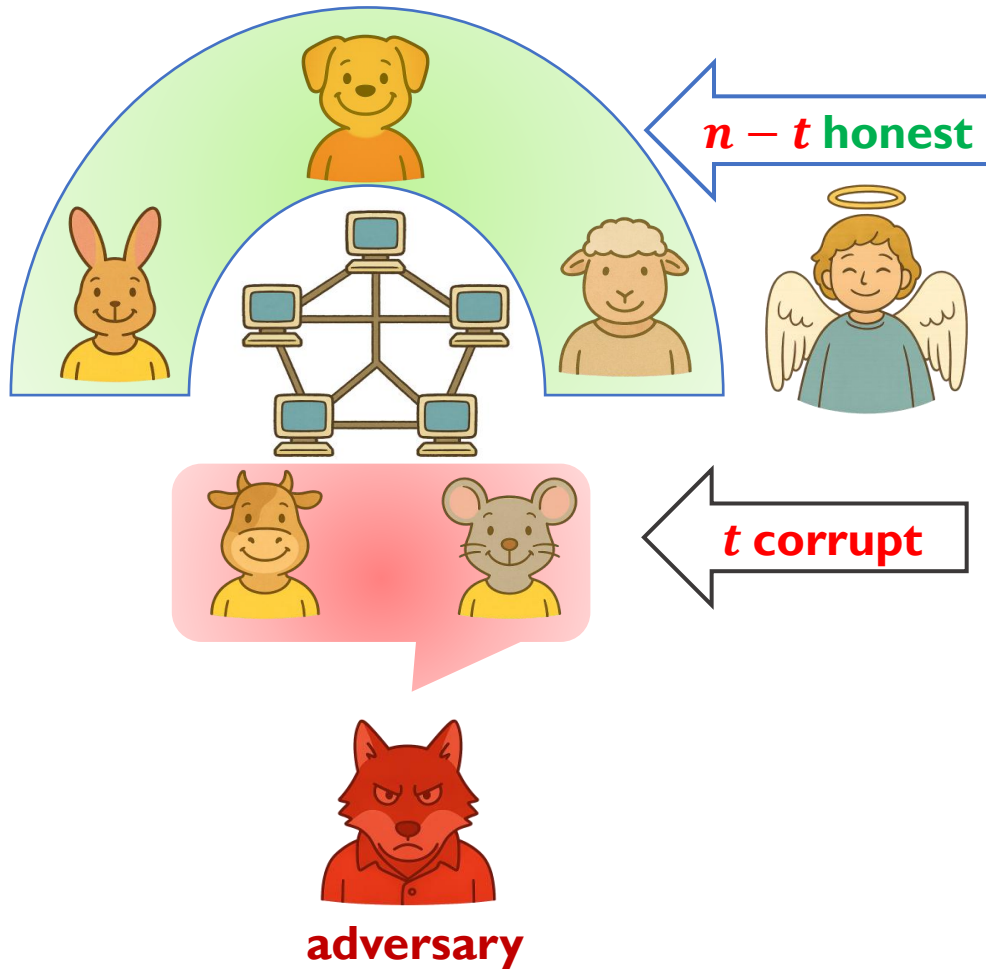
E.g., voting ($x_i \in \{0, 1\} \subseteq \mathbf{N}$):

$$f(x_1, \dots, x_n) = x_1 + \dots + x_n$$

Can be generalized to reactive functionalities (e.g., auctions).



Typical settings



n - number of parties

The protocol is attacked by the **adversary** who can corrupt up to

$$t < n$$

parties.

Common assumptions:

“honest majority”:

$$t < n/2$$

“honest **super**majority”:

$$t < n/3$$

“Secure evaluation”?

Typical requirements:

correctness – the output is correct

The only way the adversary can **influence the output** is to **manipulate the inputs of the corrupt parties**.

privacy – the inputs remain private

The adversary does **not learn** more than she can infer from the **input and output of corrupt parties**.

State of the art

maximal corruption threshold **depends on the settings** (computational/unconditional security, etc.)

In theory, **every function f can be “compiled” into an MPC protocol.**

Caveat: Despite enormous progress, MPCs remain **relatively inefficient.**

orders of magnitude less efficient than computing f
“in plain”

Blockchains

≈ protocols for constructing “distributed ledgers”

Typically, also based on “(super)honest majority” assumptions, measured in terms of:

- **computing power** (PoW-blockchains)
- **financial resources** (PoStake-blockchains)
- **disk space** (PoSpace-blockchains)
- ...

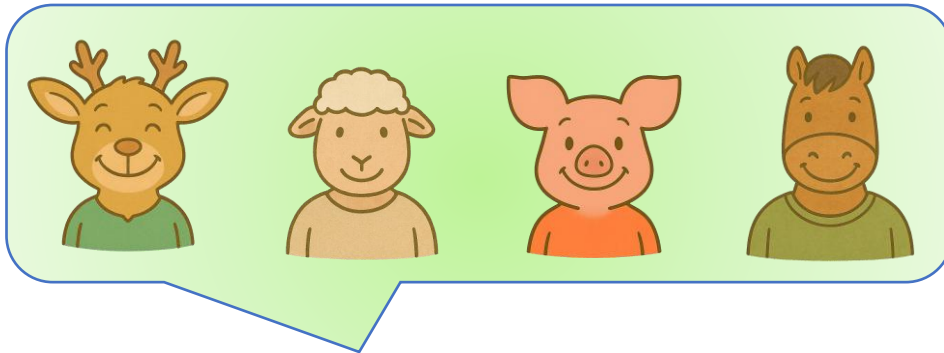
Turns out:

Blockchain literature usually interprets “honest majority”
differently from the MPC literature.

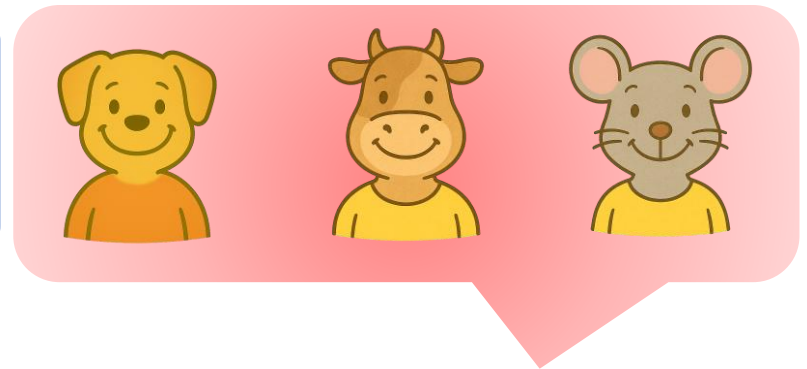
“Honest behavior” in MPC

n – number of parties

t – corruption threshold



honest



$\leq t$
corrupt
parties

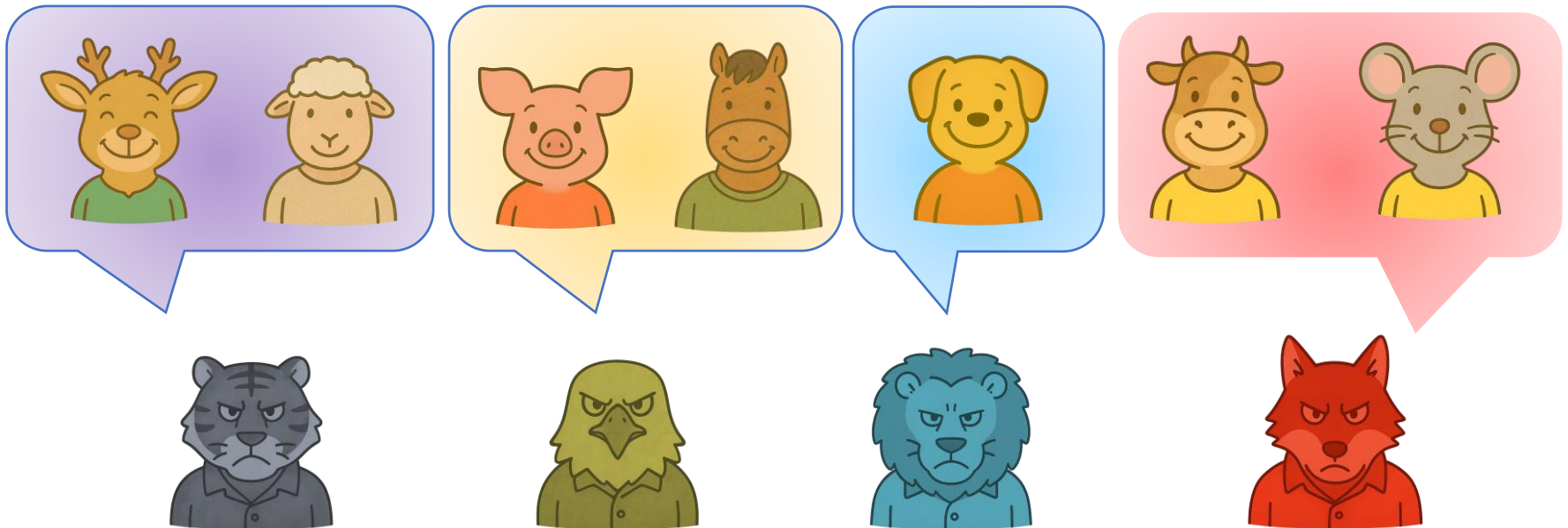


In blockchain

n – number of parties

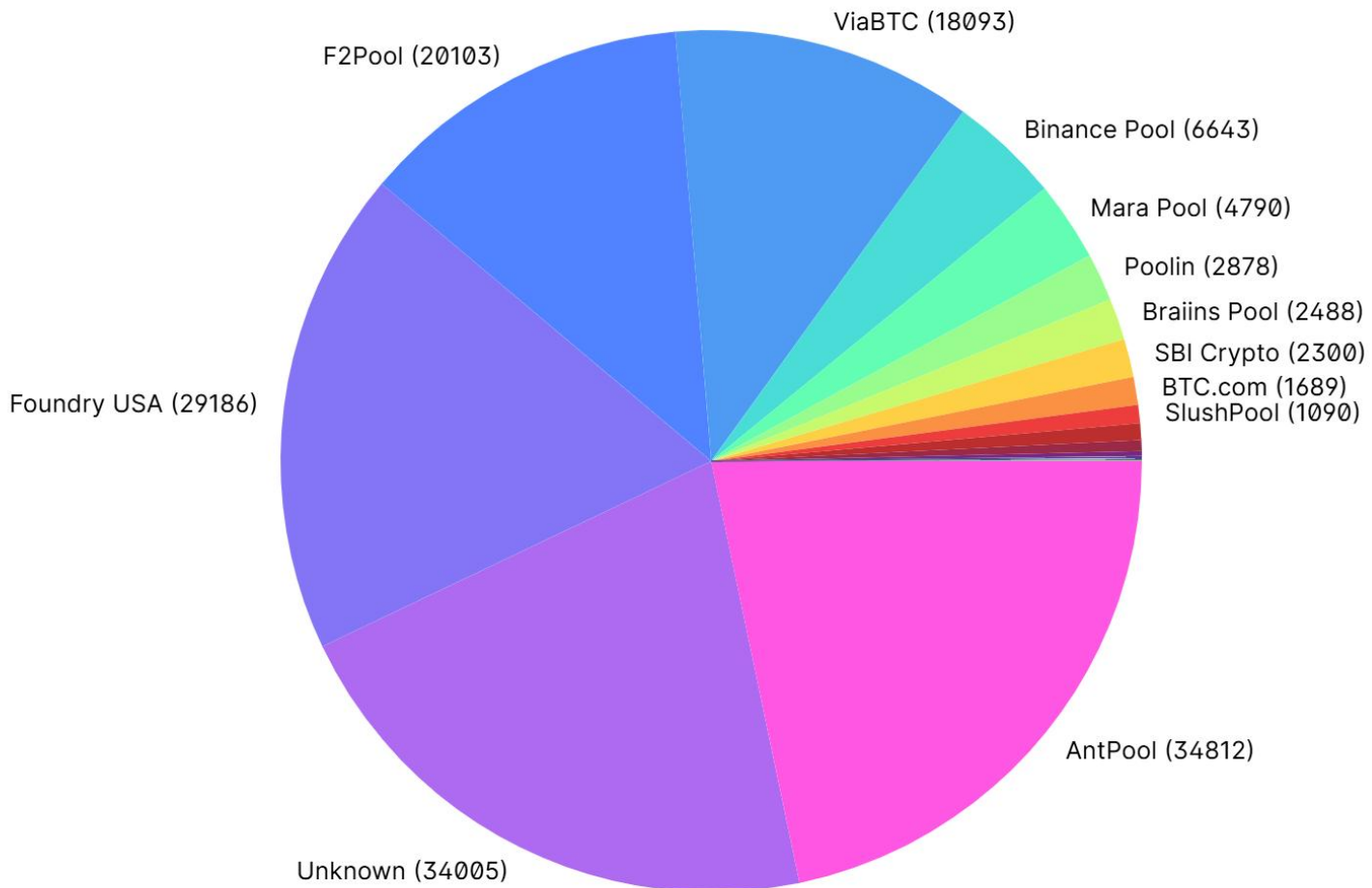
t – corruption threshold

There are **multiple adversaries**



Each adversary can corrupt $< t$ parties.
They are selfish and compete.

Example: Bitcoin's blockchain



In blockchain all parties are corrupt, but by different adversaries.



Question: Can we adapt “traditional MPCs” to these settings?

Correctness – can be addressed, e.g., by using an **incentive mechanism and punishments** to ensure that the output is computed correctly.



For general MPCs, see, e.g., [ADMM14] and the follow up work

(out of scope of this talk)

Secrecy – less clear...

our topic today

Plan

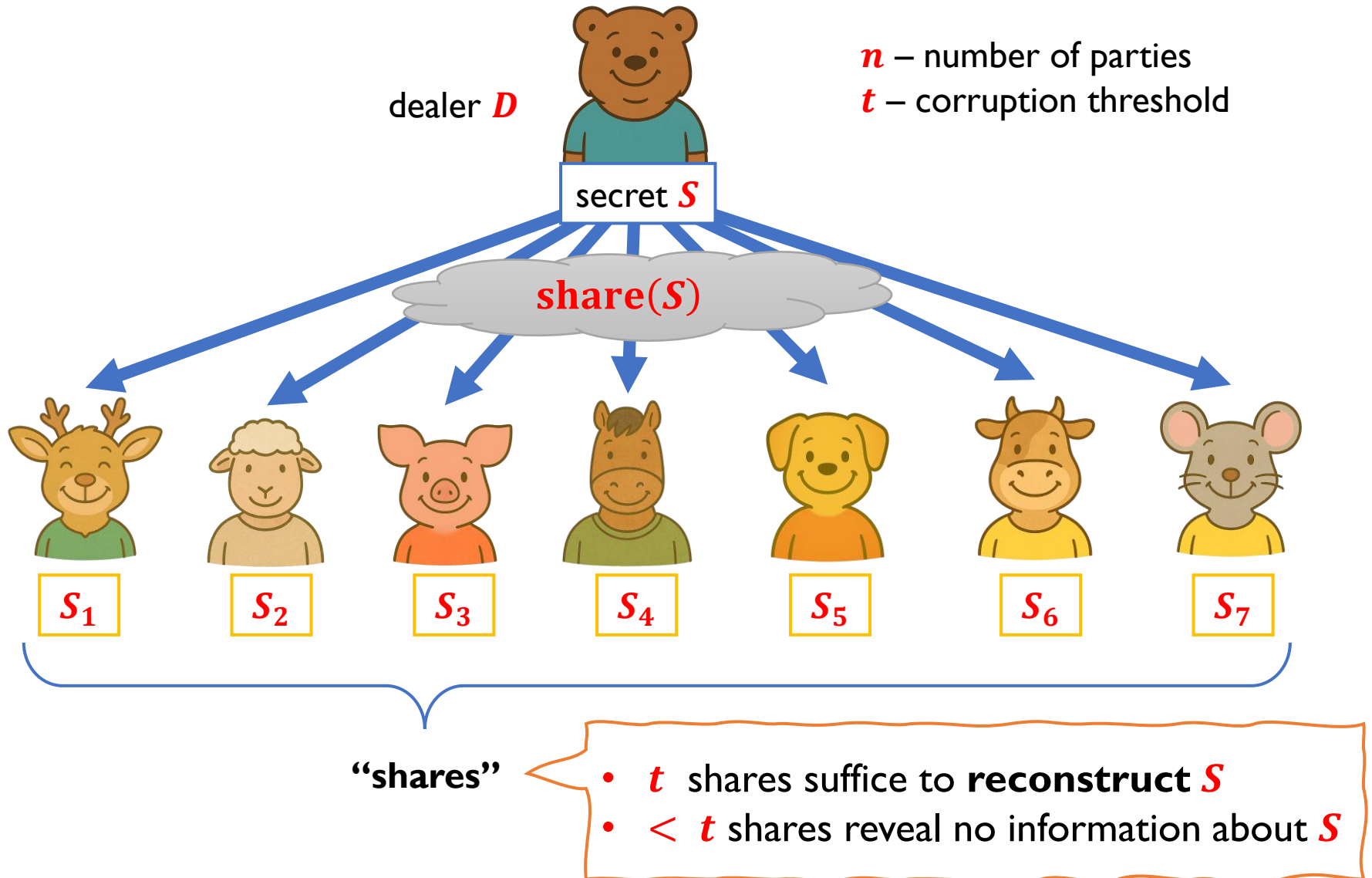
1. Introduction to different models in distributed cryptography 
2. Secret Sharing with Snitching 
3. Other applications of MPC-hardness
4. Extensions and research problems

Case study: t -out-of- n secret sharing

Secret Sharing [Shamir'79]: a protocol that permits a **dealer** to share a secret S among a group of n parties, so that:

- any set of t parties can learn S
- no set of $< t$ parties can get any information about S .

Pictorially

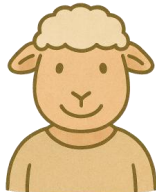


Example $t = 4$

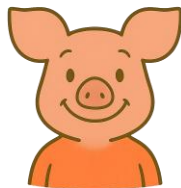
know S



S_1



S_2



S_3



S_4



S_5



S_6

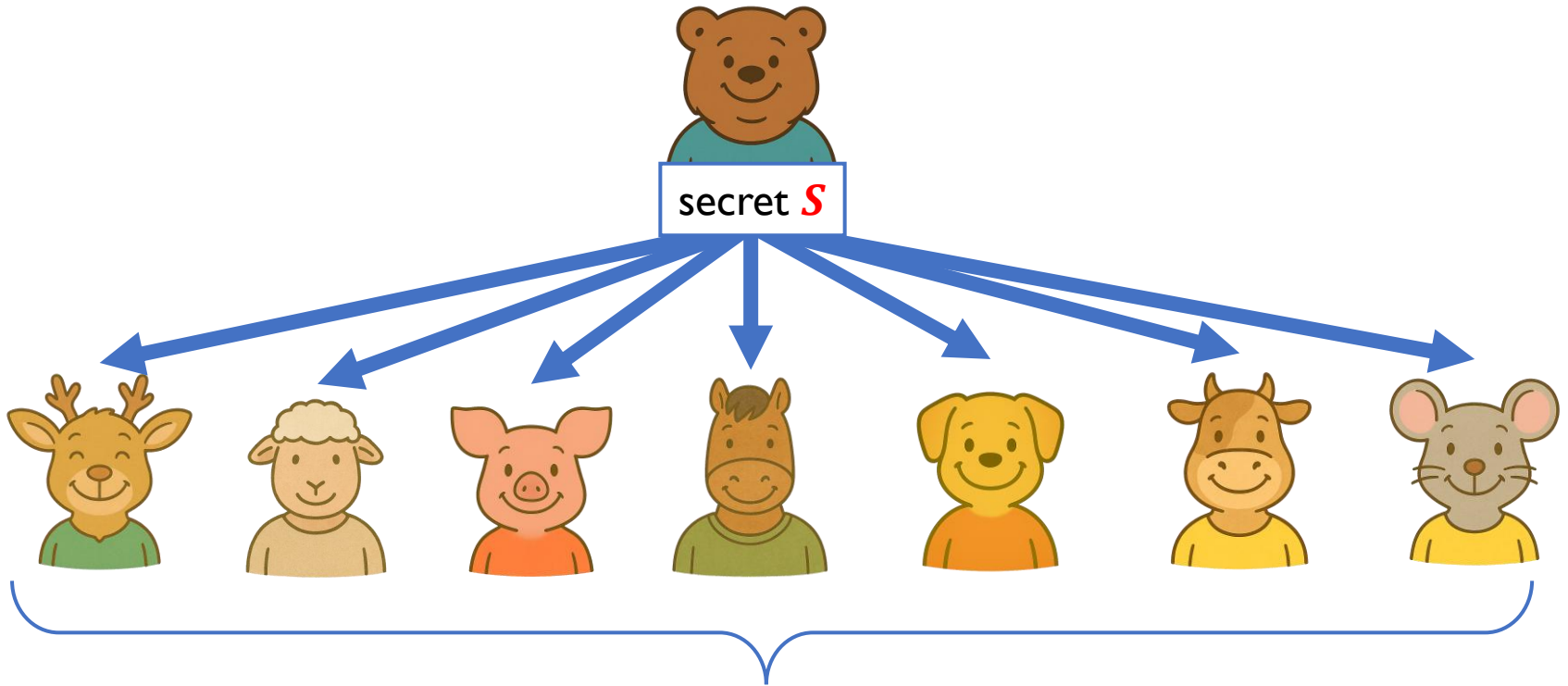


S_7

doesn't know S



Example of use



They should reconstruct S only if a **certain event happens**.

Call an earlier reconstruction **“illegal”**.

How to implement it?

m – some parameter

Suppose $S \in \{0, 1\}^m$

call it “xor secret sharing”

n -out-of- n secret sharing:

$$\text{share}(S) := (S_1, \dots, S_n)$$

Where S_1, \dots, S_n are random such that

$$S_1 \oplus \dots \oplus S_n = S$$

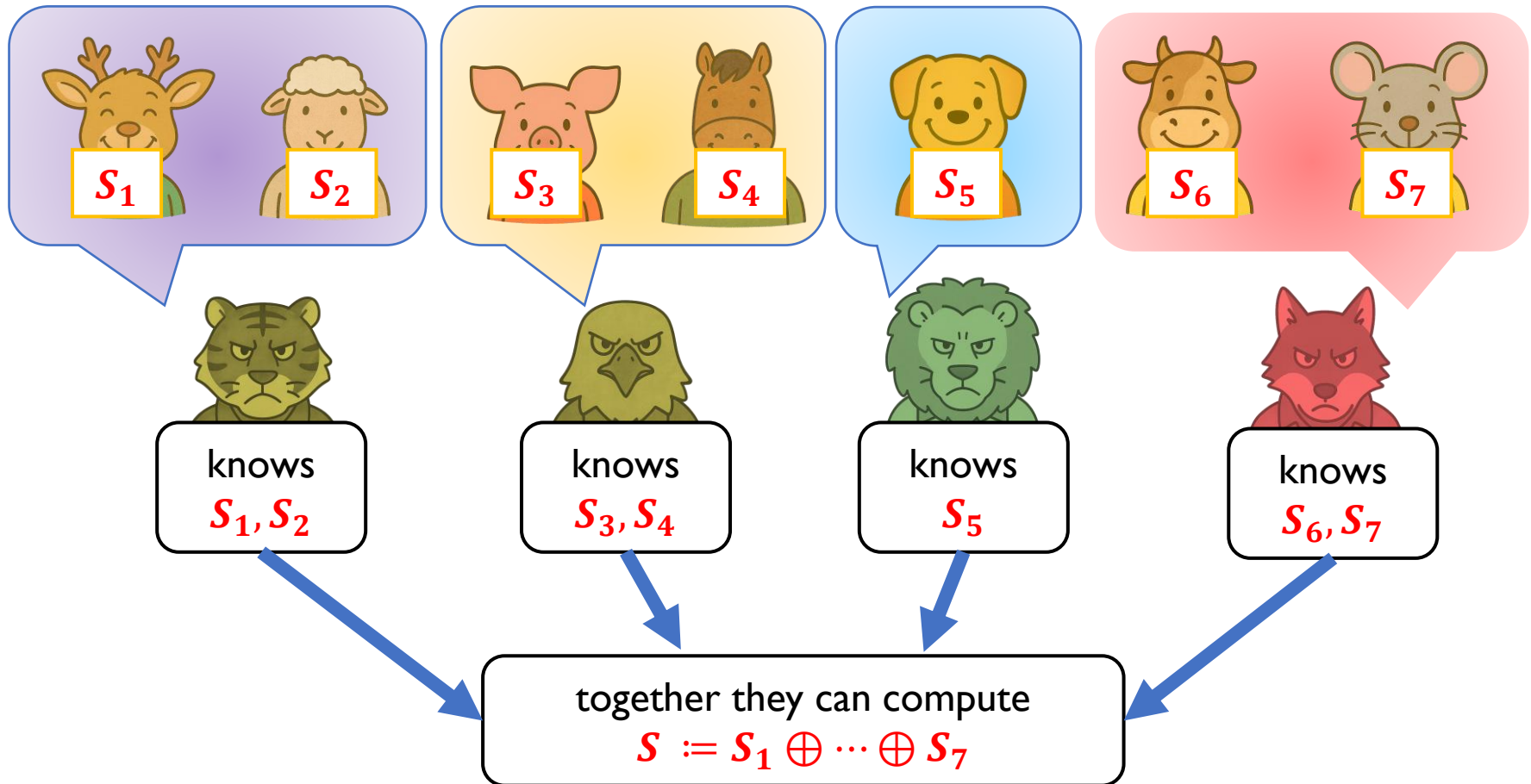
To reconstruct just xor the S_i 's.

string xor
operation

t -out-of- n secret sharing: see [Shamir'79]

Problem

Learning a secret is a “passive attack” and can be done entirely “**outside the protocol**”.



Preventing this looks impossible...



Our idea

Since illegal reconstruction cannot be prevented, at least make it **provable** and **punishable**.

In other words, create a mechanism for **economically disincentivizing illegal reconstruction**.

Recall that the **adversaries are competing**.

So: create a system for **reporting “illegal reconstruction” to some judge**.



The judge



An entity to whom the parties can **snitch** if someone was collaborating with them to illegally reconstruct **S**.

Practical example: smart contracts

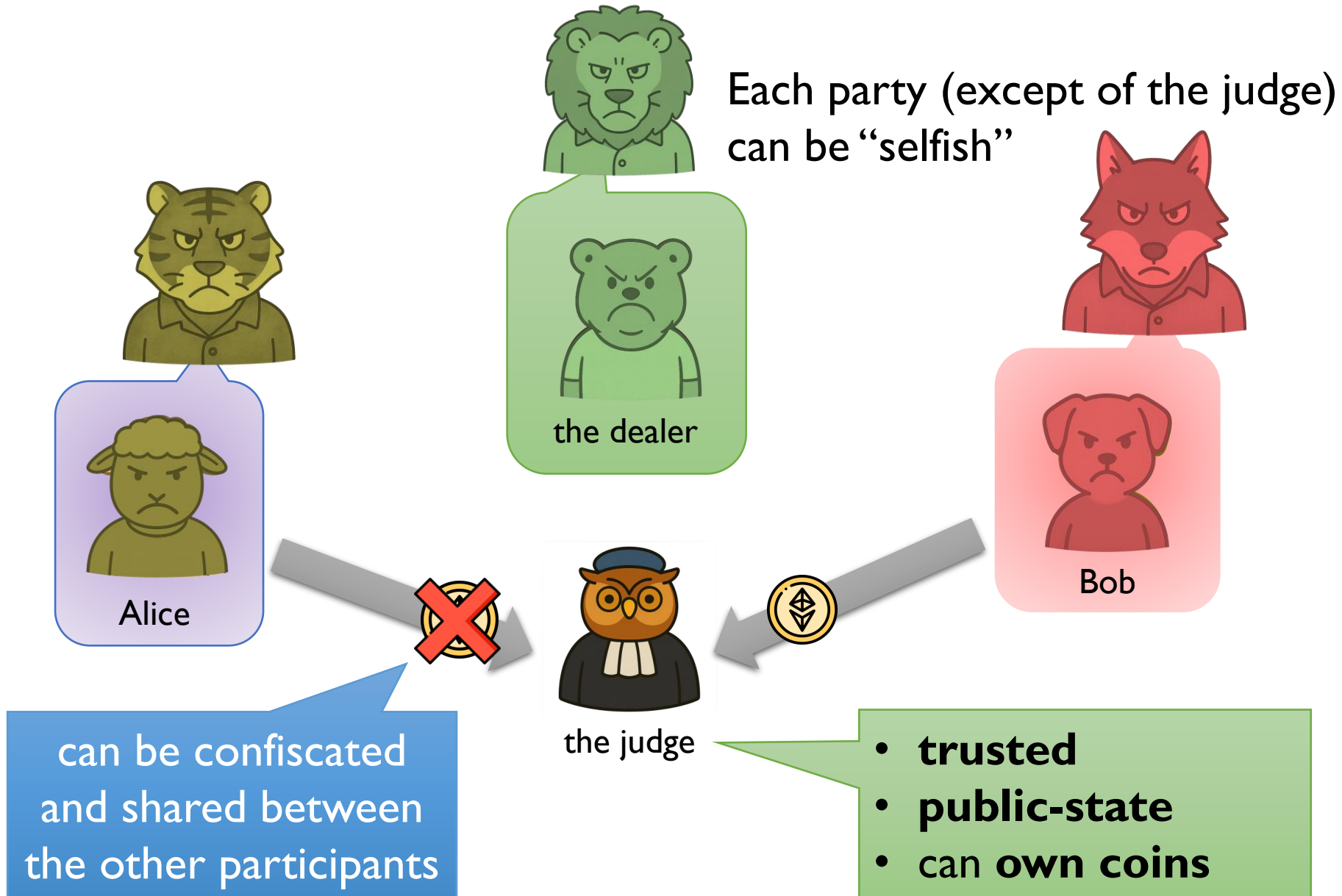


available, e.g.,
on Ethereum

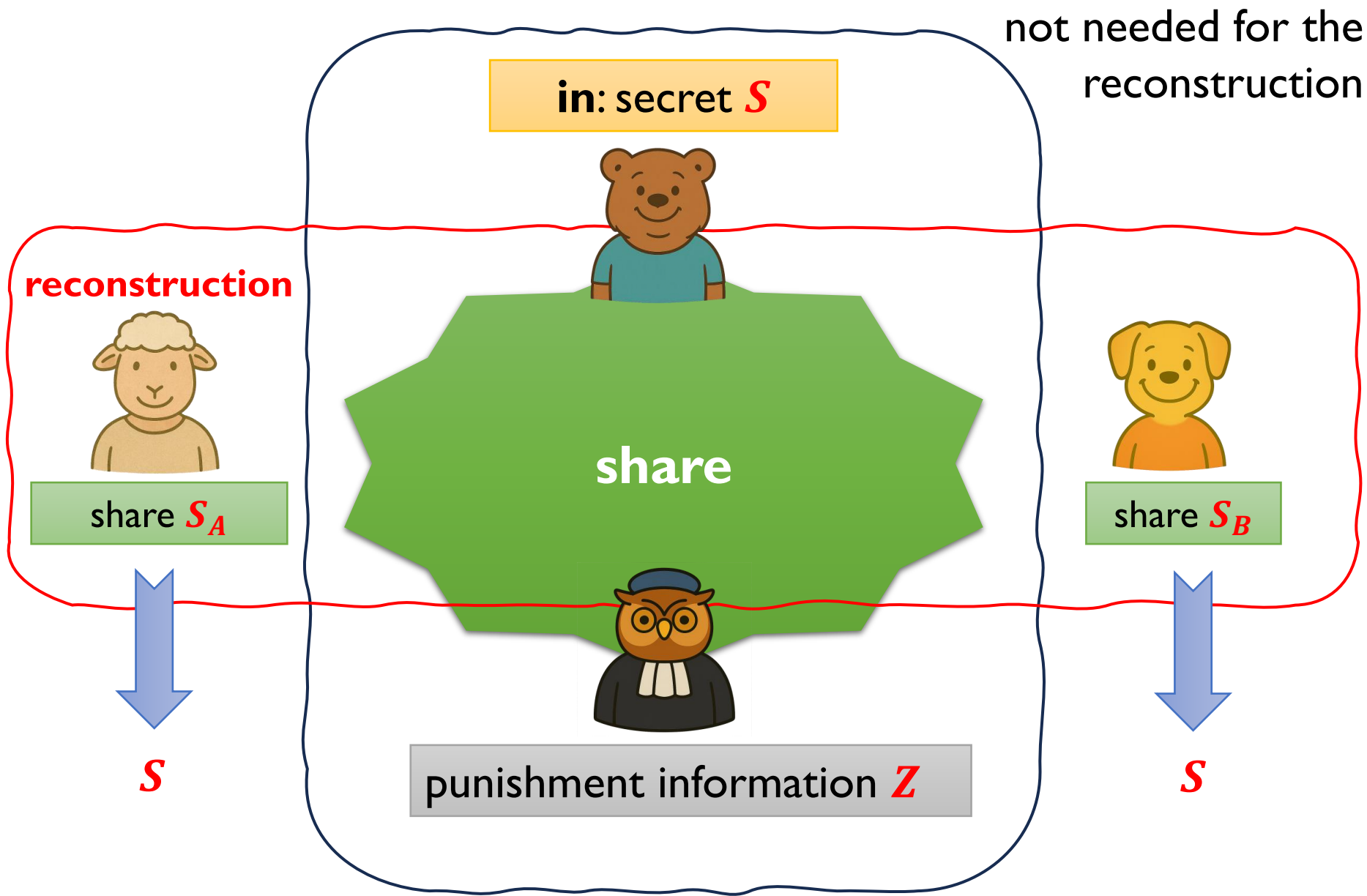
Agreements on blockchain that are:

- **enforceable** by the blockchain mechanics,
- **public-state**
- can **own coins**

Consider **2-out-of-2** secret sharing

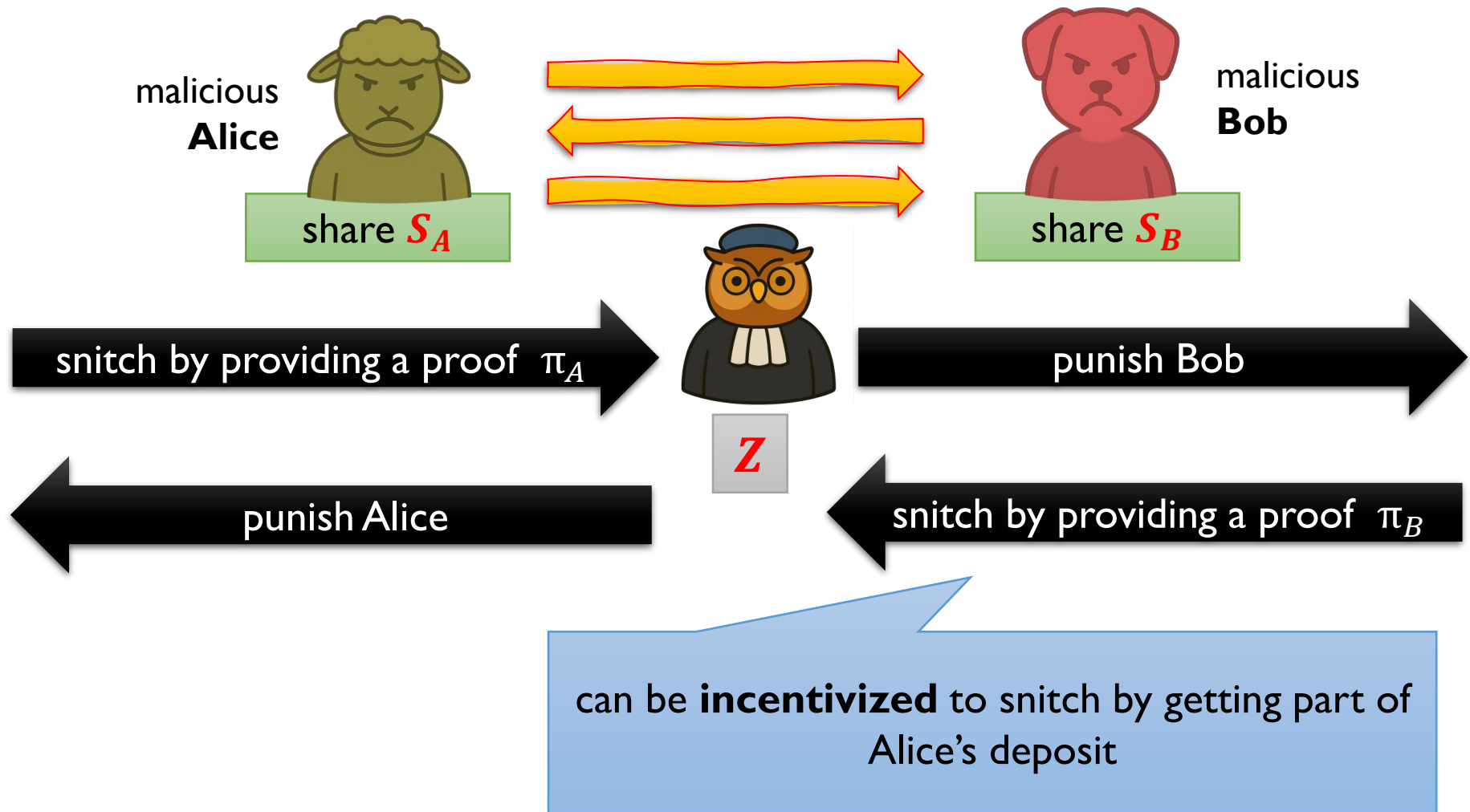


Sharing and reconstructing



Snitching

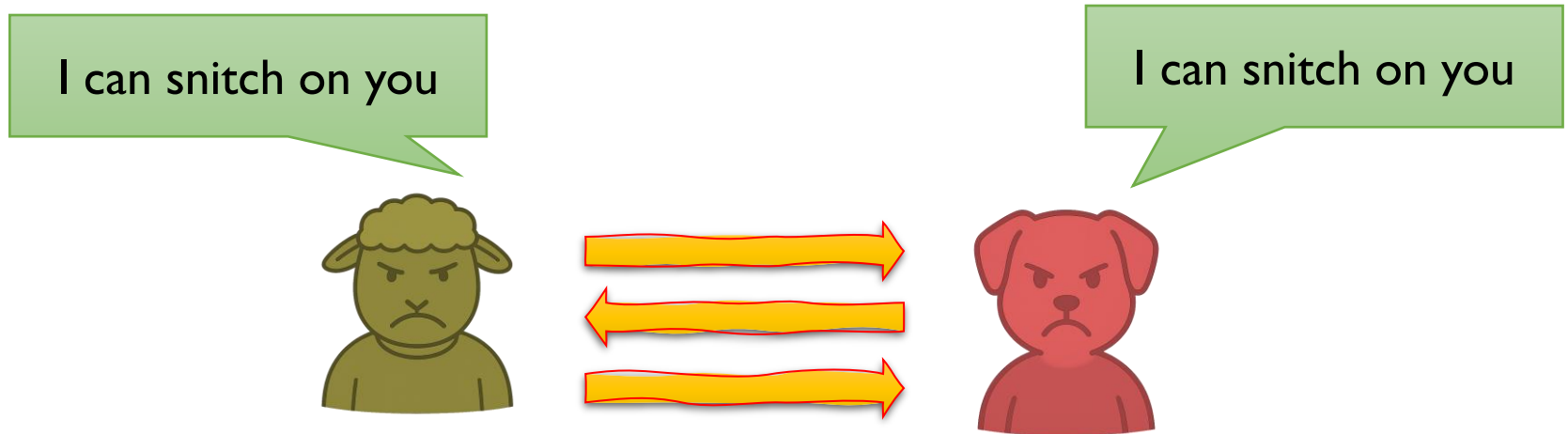
Suppose **Alice** and **Bob** illegally **reconstruct** the secret.



Note

We are interested in deterring “**illegal reconstruction**”.

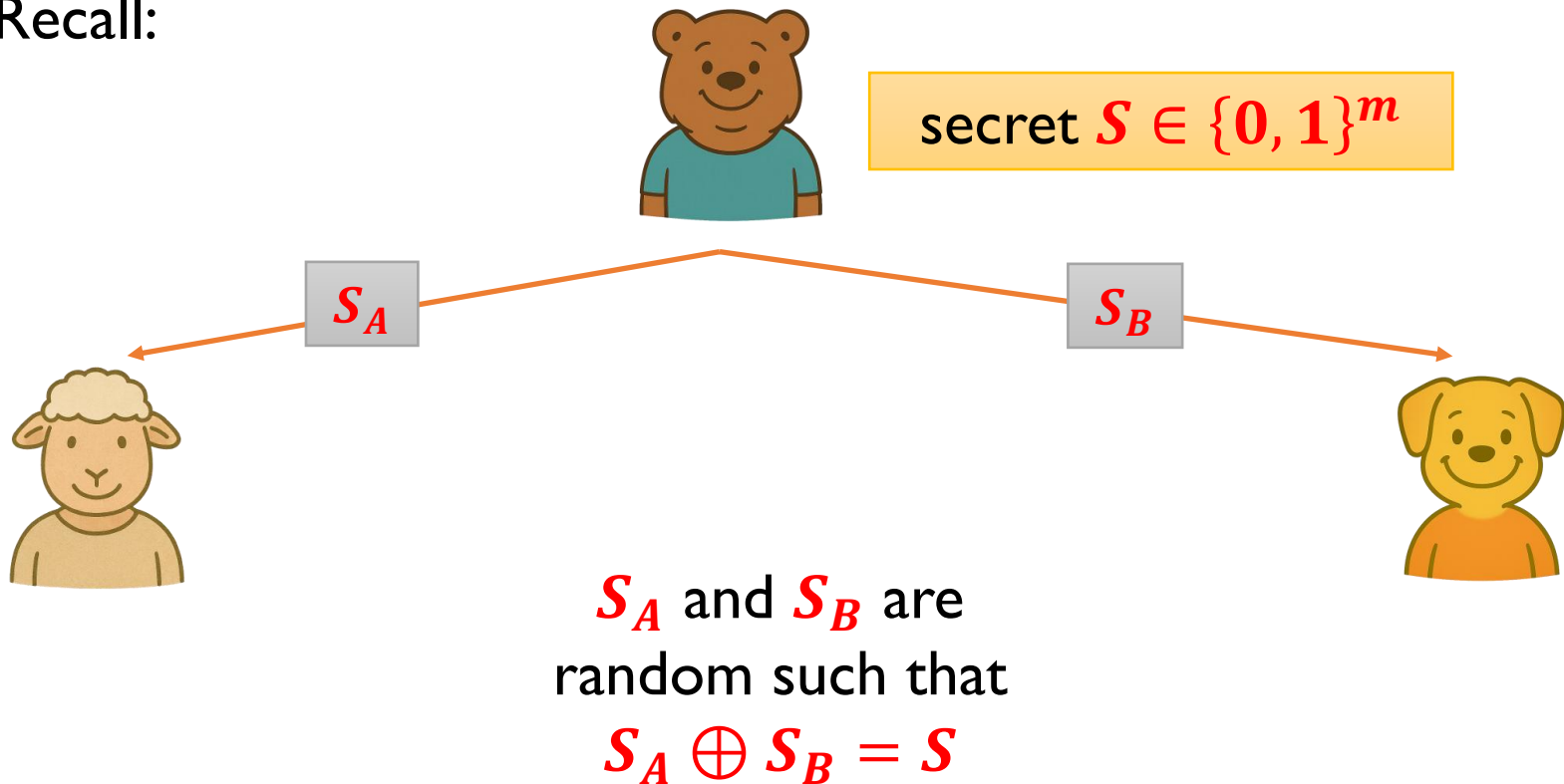
So, if they can **both** snitch on each other, it's even better.



Construction?

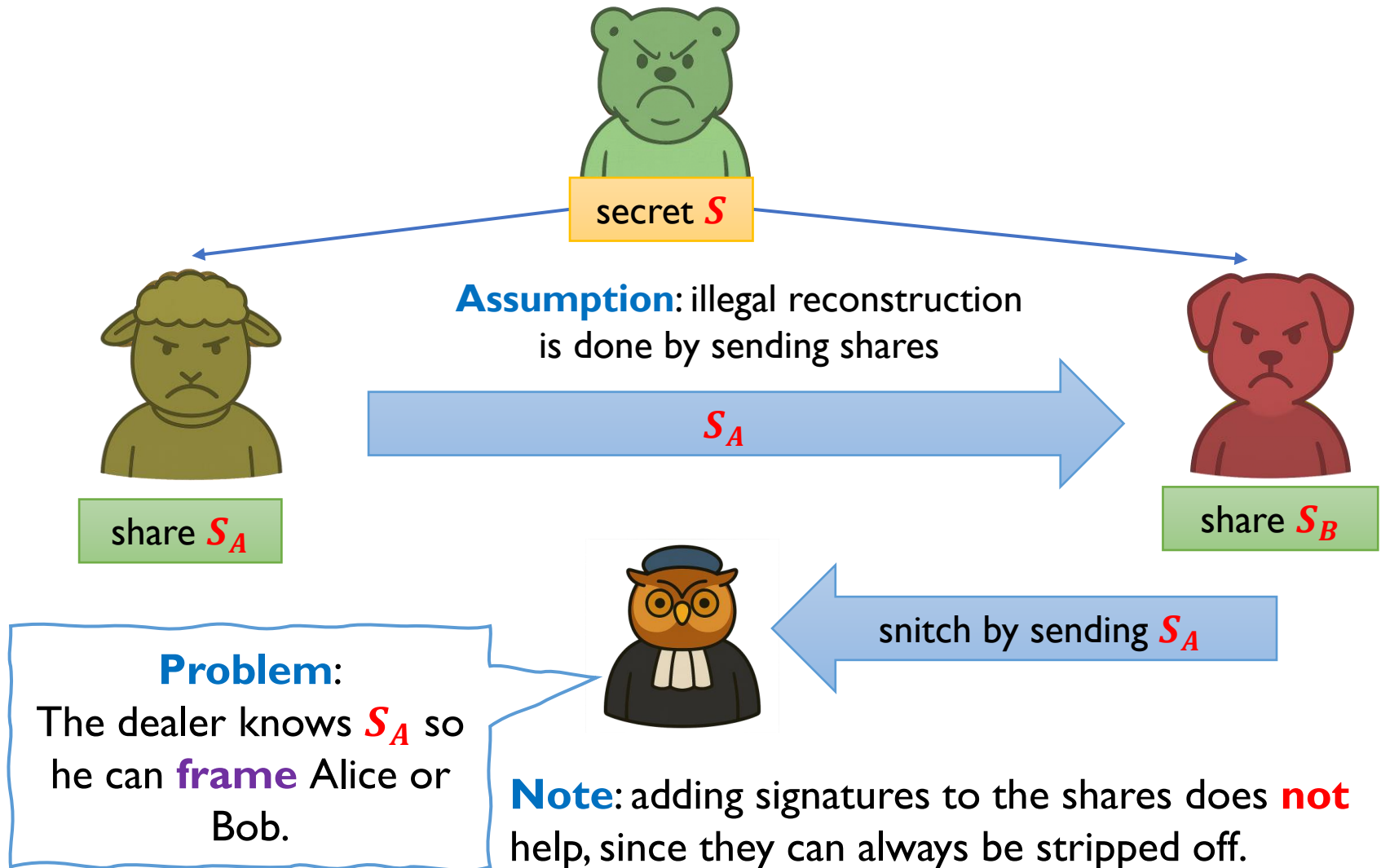
Secret Sharing **without** Snitching

Recall:



A straw-man proposal

Use the standard secret sharing

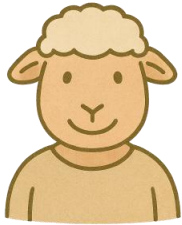


A better idea

f – a one-way function



secret S



share \hat{S}_A



share \hat{S}_B

Compute in MPC:

1. sample Y – a random string
2. let $\hat{S} := (S || Y)$
3. compute \hat{S}_A, \hat{S}_B – xor sharing of \hat{S}
4. compute $Z := f(Y)$

xor shares of \hat{S}



punishment information

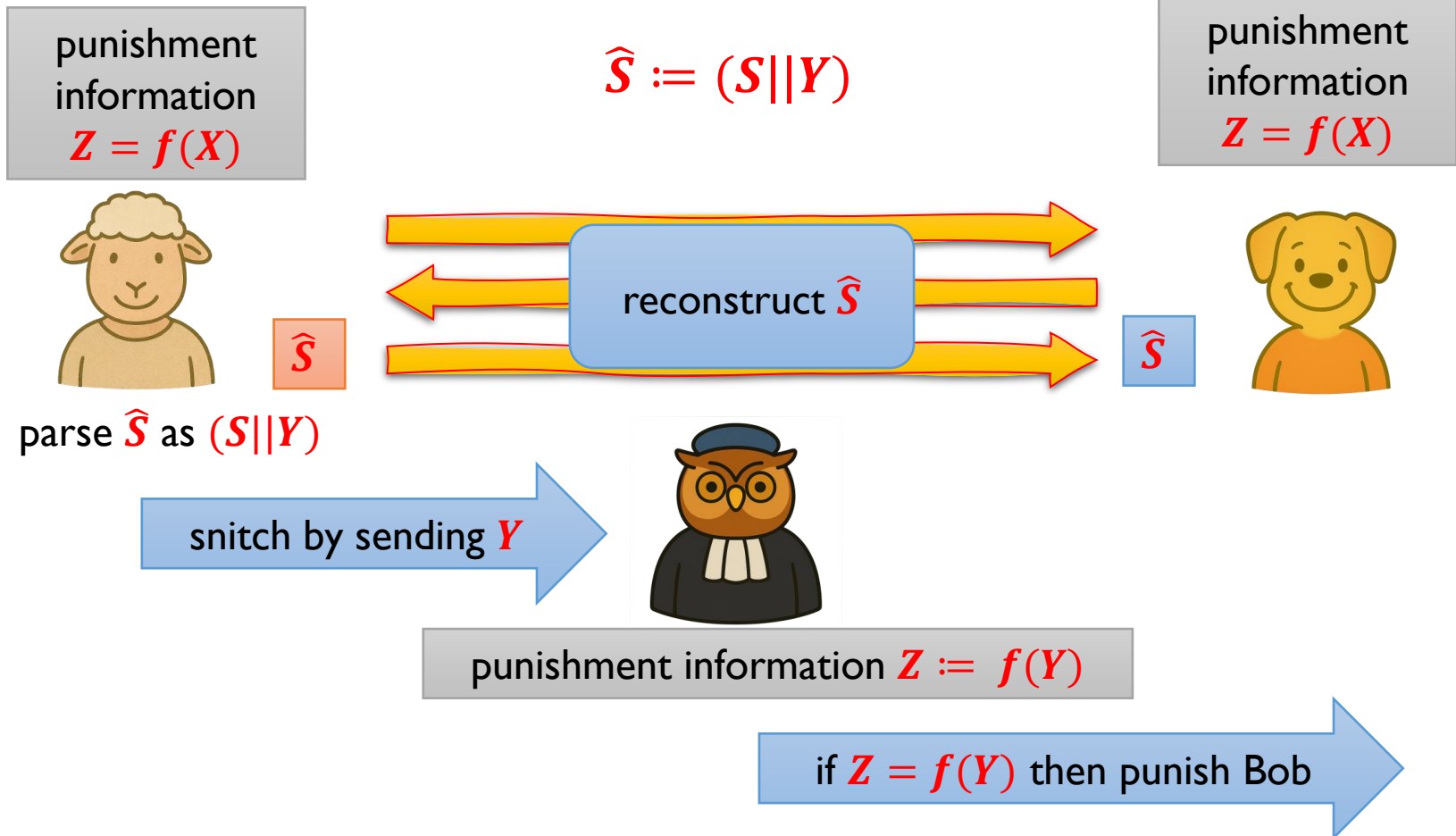
$$Z := f(Y)$$

Snitching and punishment

f – a one-way function

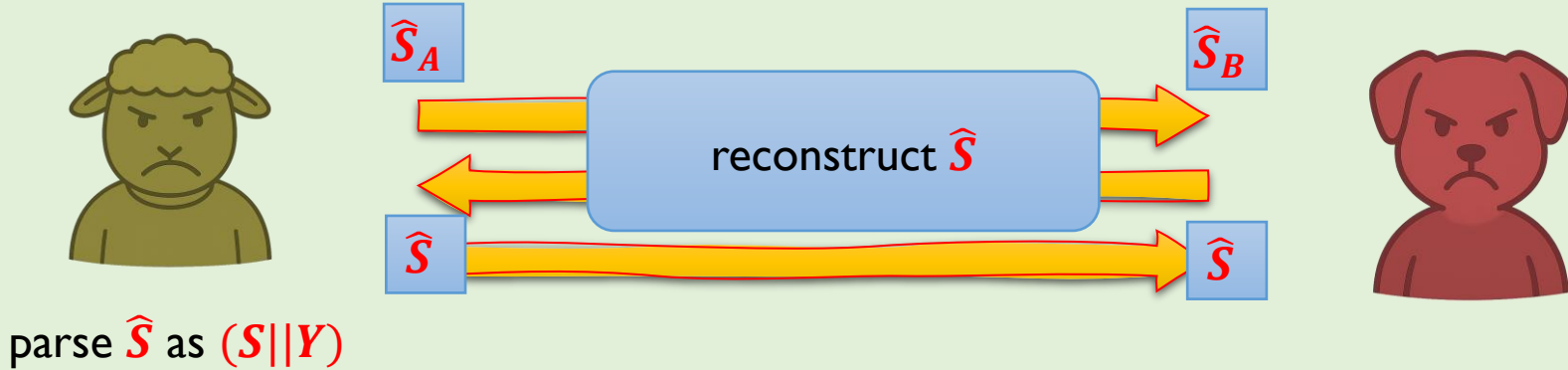
secret S

Y – a random string

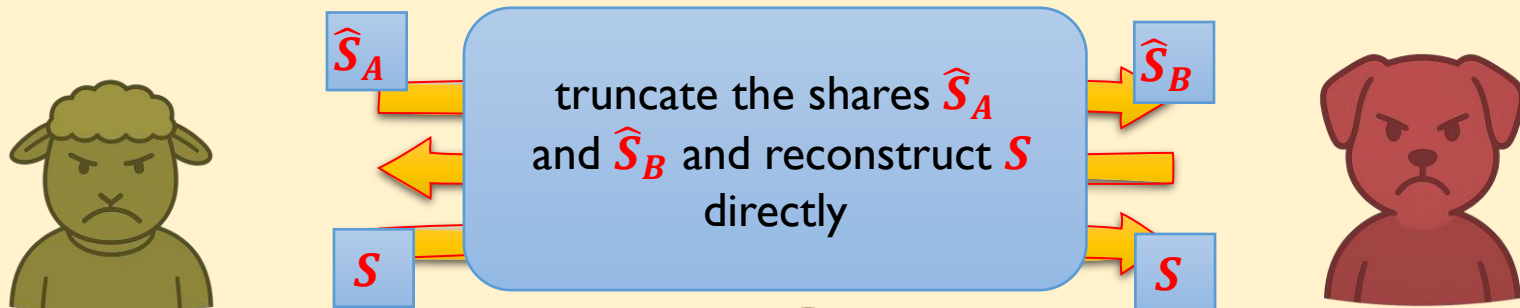


Problem

Instead of doing this:



They can do this:

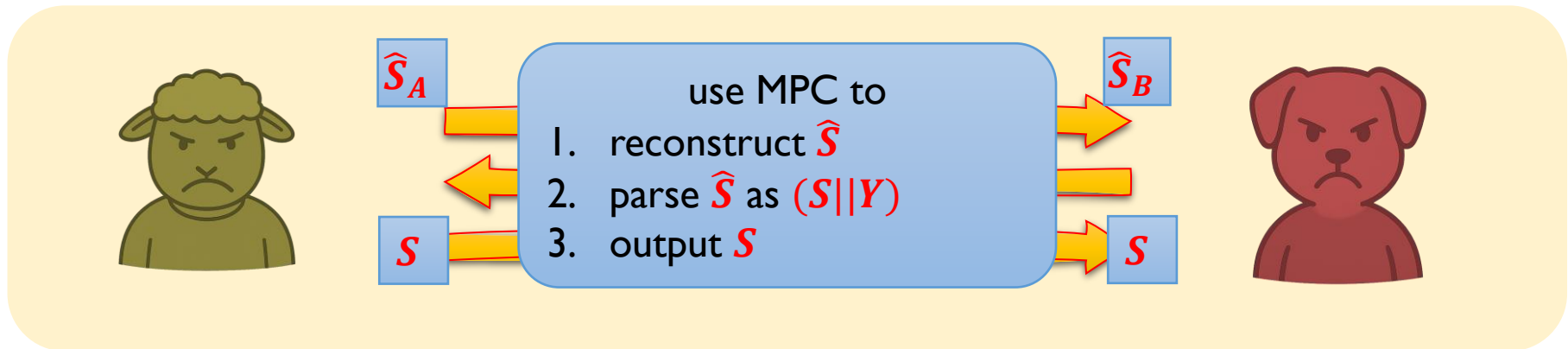


So: snitching is prevented!



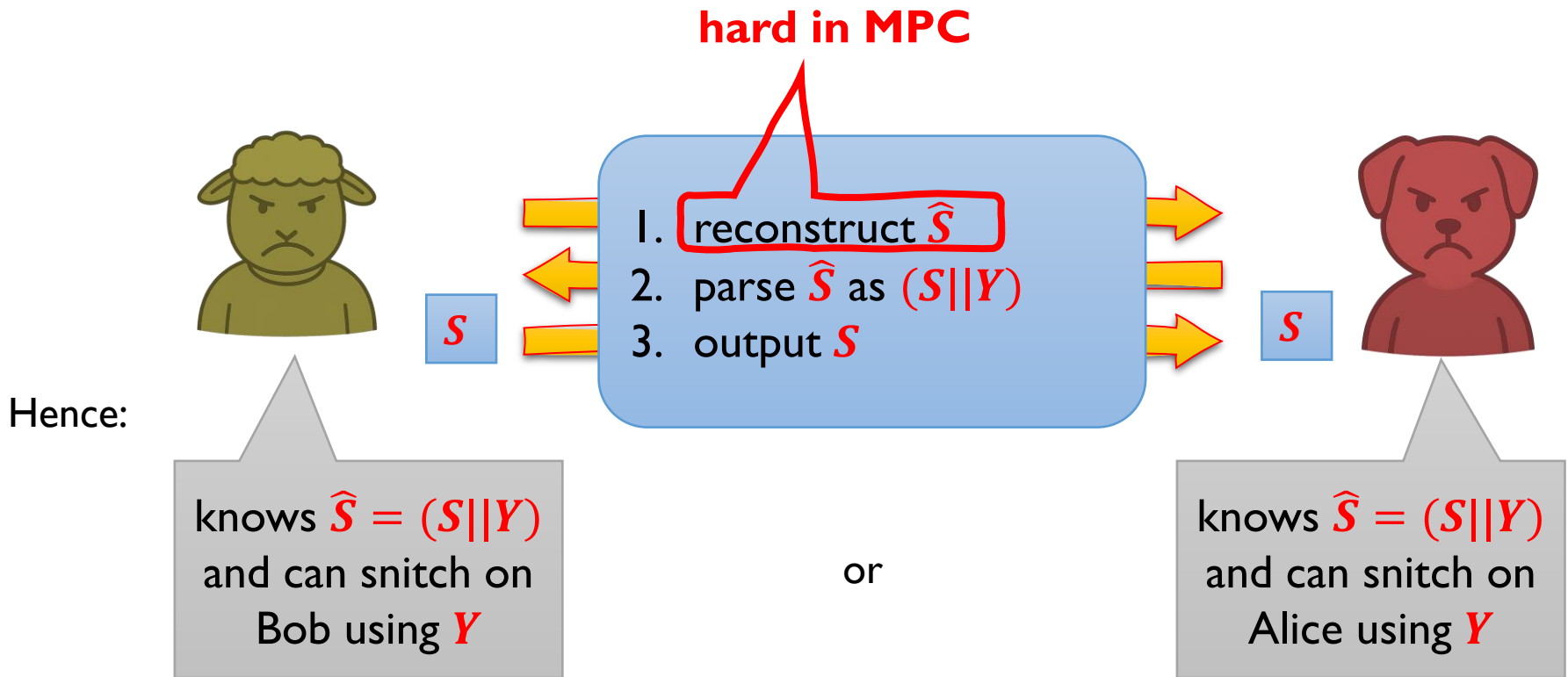
More generally

Even if they use some other secret sharing, they can do the following:



Moral

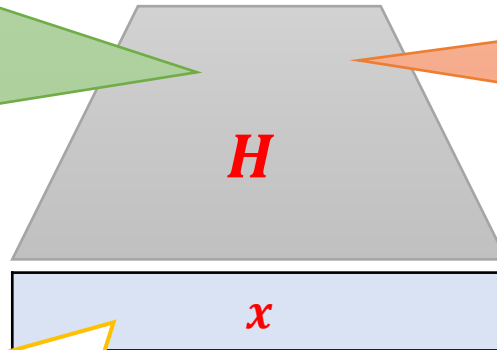
We need Secret Sharing, in which reconstruction is “**hard in MPC**”.



How to model it?

Main building block: hash functions

to avoid attacks that **exploit the structure of H** we only use **fixed-input length** functions.



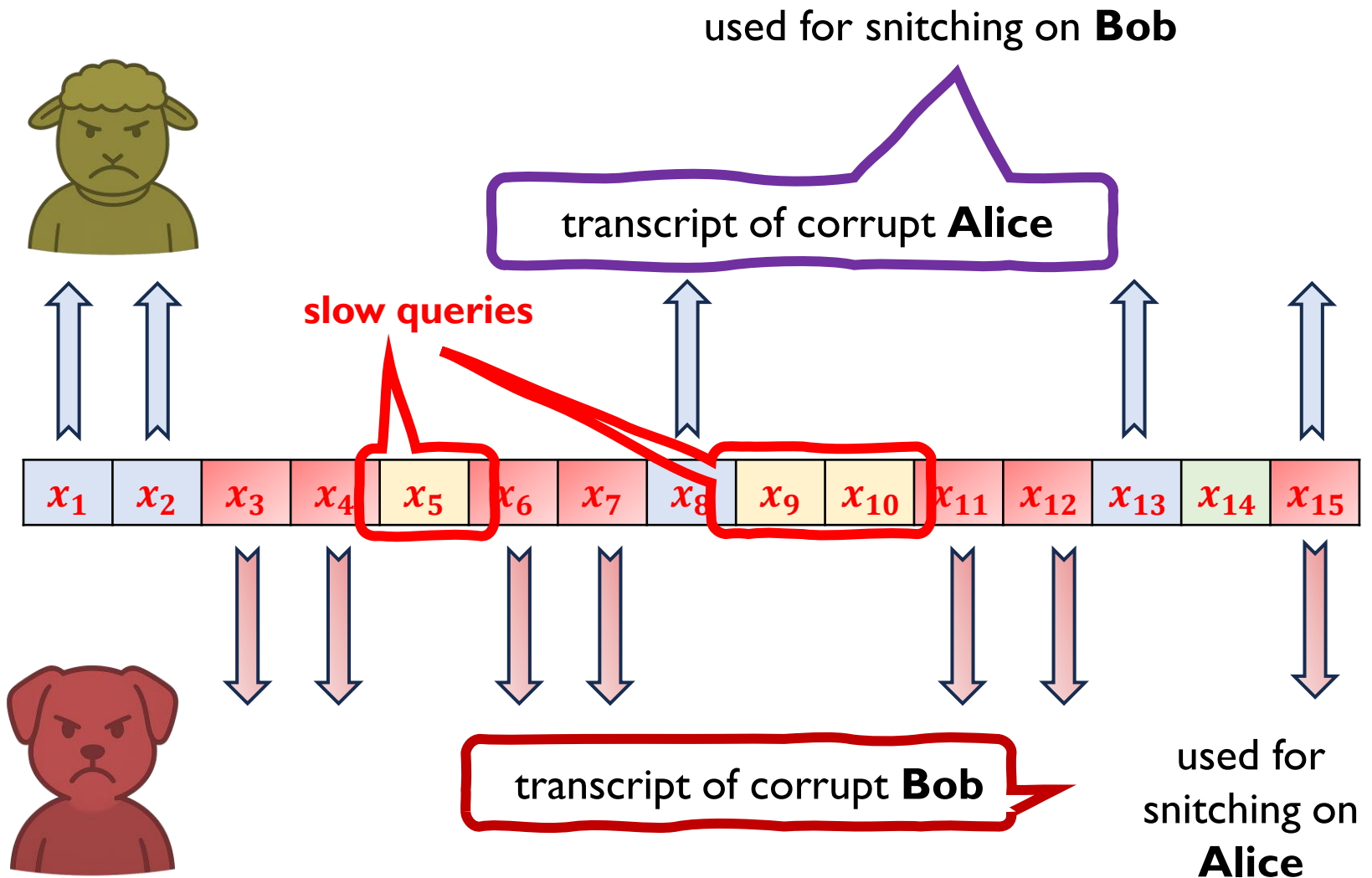
modeled as a random oracle

Two types of queries:

- **slow** (computed in MPC) – the hash input is **not known to any individual party**
- **fast** – the input needs to be **known to some individual party**

The budget for slow queries is bounded.

More formally



Generalizes easily to multiparty settings

Our construction

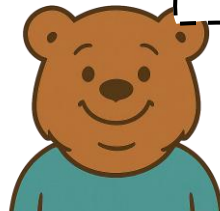
Main idea: to **reconstruct the secret**, the parties need to compute a **massive number** of hashes ***H***.

More precisely:

1. **Sharing** requires a small number of MPC computations of ***H***.
2. **Reconstructing** requires inverting ***H*** by brute-forcing it.

Sharing (simplified)

k – security parameter
 d – “moderate hardness” parameter
 H, G – hash functions



secret S



$$X_A \leftarrow \{0, 1\}^k$$



compute in MPC:

1. sample $Y \leftarrow \{0, 1\}^d$
2. let $Z := H(X_A \oplus X_B || Y)$

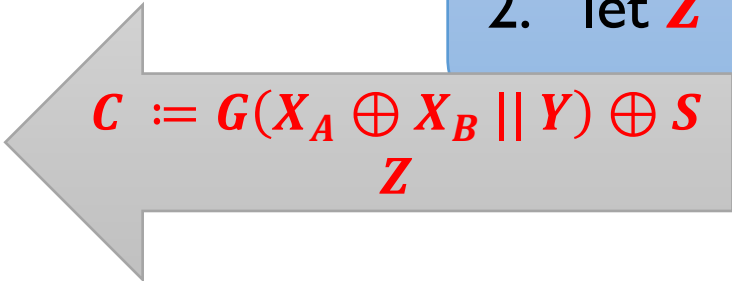


$$X_B \leftarrow \{0, 1\}^k$$



$$C := G(X_A \oplus X_B || Y) \oplus S$$

Z

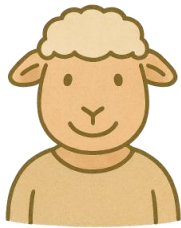


Z

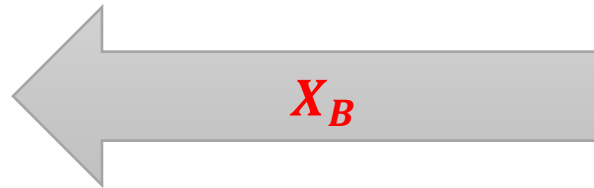


Reconstruction

$$\begin{aligned} Z &:= H(X_A \oplus X_B || Y) \\ C &:= G(X_A \oplus X_B || Y) \oplus S \end{aligned}$$



X_A
 C



X_B

1. compute $X := X_A \oplus X_B$
2. find Y such that $Z := H(X || Y)$
3. output $S := G(X || Y) \oplus C$

brute force

Snitching

1. compute $X := X_A \oplus X_B$

2. find Y such that $Z := H(X || Y)$

brute force

3. output $S := G(Y) \oplus C$



X and Y such that
 $H(X || Y) = Z$



Z

Full version

We need k variables Y instead of one.

We generalize the definition to t -out-of- n SSS

We use 2-out-of-2 SSS to build t -out-of- n SSS

For the details: see our ACM CCS'24 paper.

An example of an application

“Maximal Extractable Value”

MEV – protection

Instead of posting a transaction ***S*** directly on a blockchain, a user **secret-shares it with some consortium**.

The consortium **reconstructs *S* after some time has passed**.

Note: no long-term protection is needed.

Alternative solutions

Collusion-free protocols make physical assumptions that prevent communication between the parties.

Traceable secret sharing (based on **traitor tracing** techniques) – assume that reconstruction is done by physical boxes that are given to the adversary.

(see our ACM CCS'24 paper for more on this related work).

Bonus question



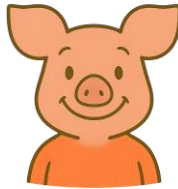
What if the adversaries can also use a **judge whom they all fully trust**?

For example, they can deploy their **own smart contract** on the blockchain.

Our new notion: **Strong Secret Sharing with Snitching**

Strong Secret Sharing with Snitching

honest



honest parties

judge

adversarial

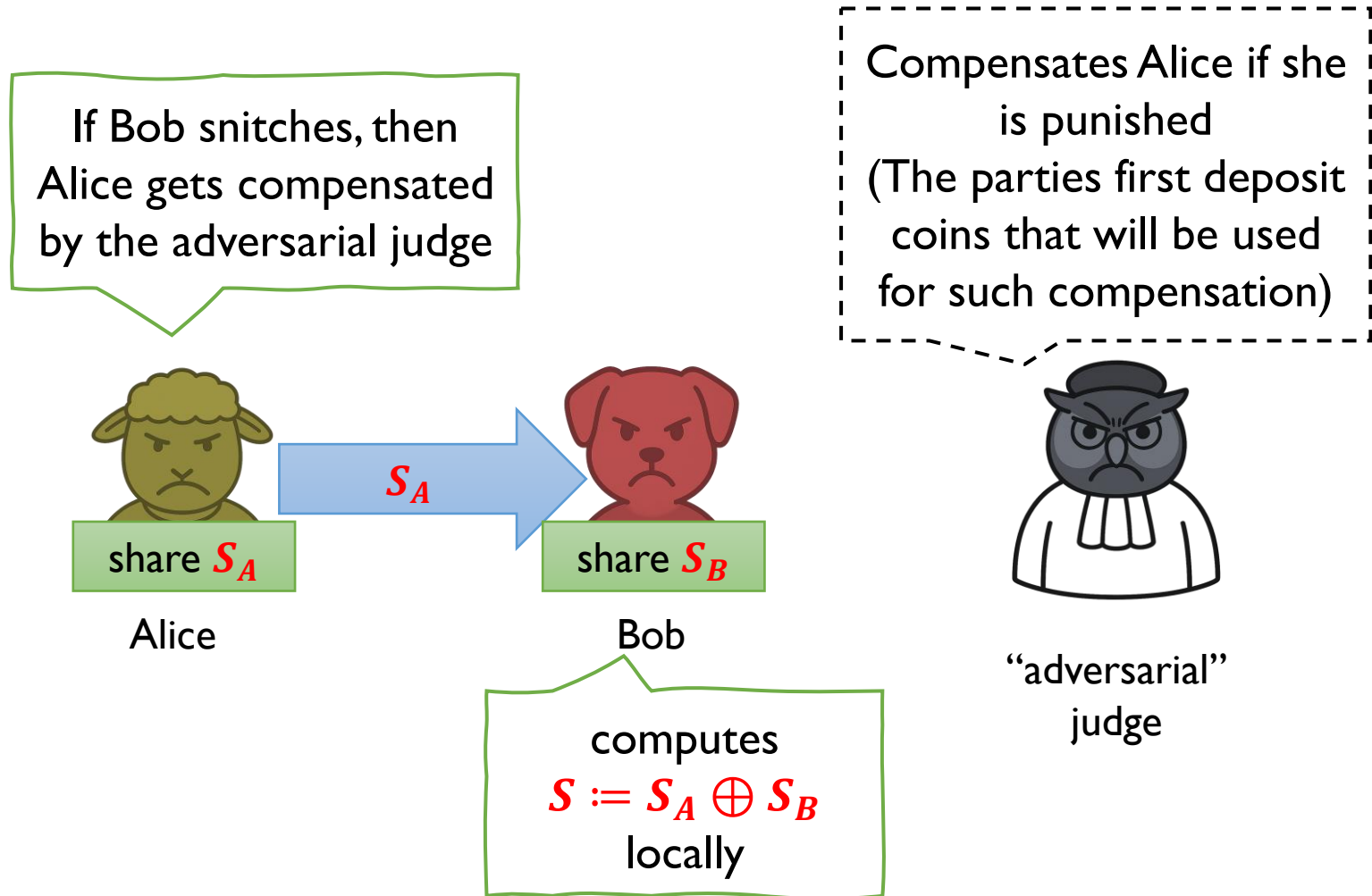


adversaries

“adversarial”
judge

Example of an attack: insurance

Consider again **2-out-of-2** secret sharing.






Our contribution

1. A new model that captures such attacks.
2. A construction that is secure in this model.

based on the idea of **self-snitching**
“everybody can punish itself”

(requires the use of **Non-Interactive Zero Knowledge** to make the self-snitching undistinguishable from the real snitching)

Plan

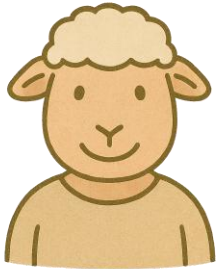
1. Introduction to different models in distributed cryptography 
2. Secret Sharing with Snitching 
3. Other applications of MPC-hardness 
4. Extensions and research problems

(Zero-Knowledge) Proofs of Individual Knowledge (PIK)

[D., et al., CRYPTO 2023]

a similar notion: **Proofs of Complete Knowledge**

[Kelkar, et al., ACM CCS 2024]



prover proves that a
message M is stored
entirely on a single
machine



verifier verifies
this claim

knows M

or

(in the ZK variant)
has some information
about M

Applications of PK

- preventing **account sharing**
- **deniable messaging**
- preventing **vote selling** in online voting

see: **[D., et al., CRYPTO 2023, Kelkar, et al., ACM CCS 2024]**

Another recent work in this model

J. Hsin-yu Chiang, B. David, T. Kasper Frederiksen, A. Mondal, E. Yeniaras:

Detecting Rogue Decryption in (Threshold) Encryption via Self-Incriminating Proofs

ePrint 2024

Plan

1. Introduction to different models in distributed cryptography
2. Secret Sharing with Snitching
3. Other applications of MPC-hardness
4. Extensions and research problems



Extensions

- Weaker modeling of fast queries
- Making it compatible with Bitcoin mining rigs (hope: better security against TEEs)
- Building protocols on top of SSS

Research question:

1. Study MPC-hard primitives
2. Combine individual crypto it with timed crypto

Thanks!

