

Threshold ECDSA with Identifiable Abort: *the case for Honest Majority*

Yashvanth Kondi

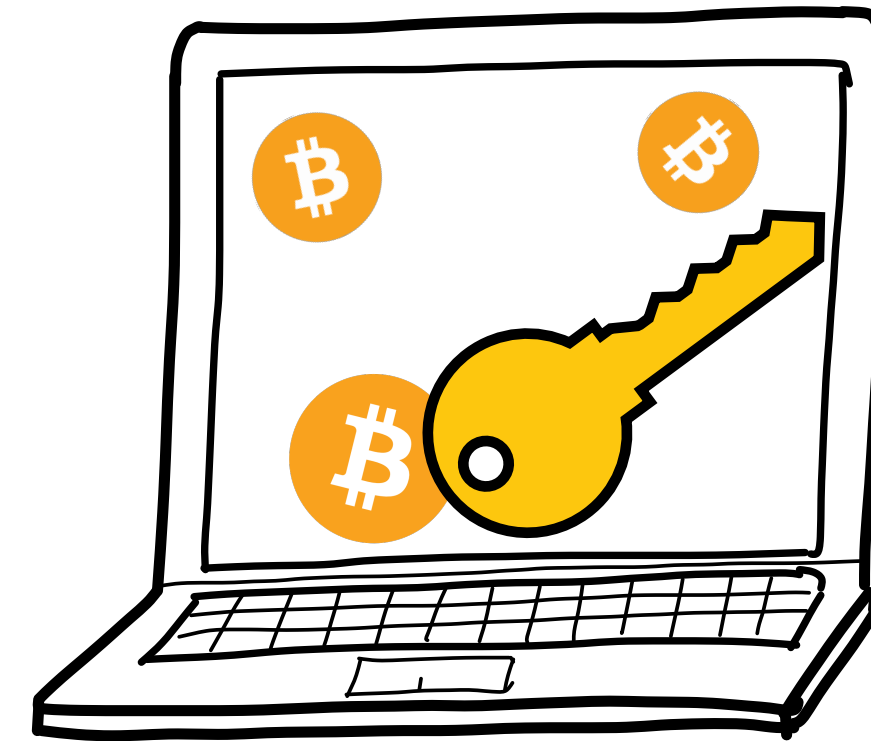
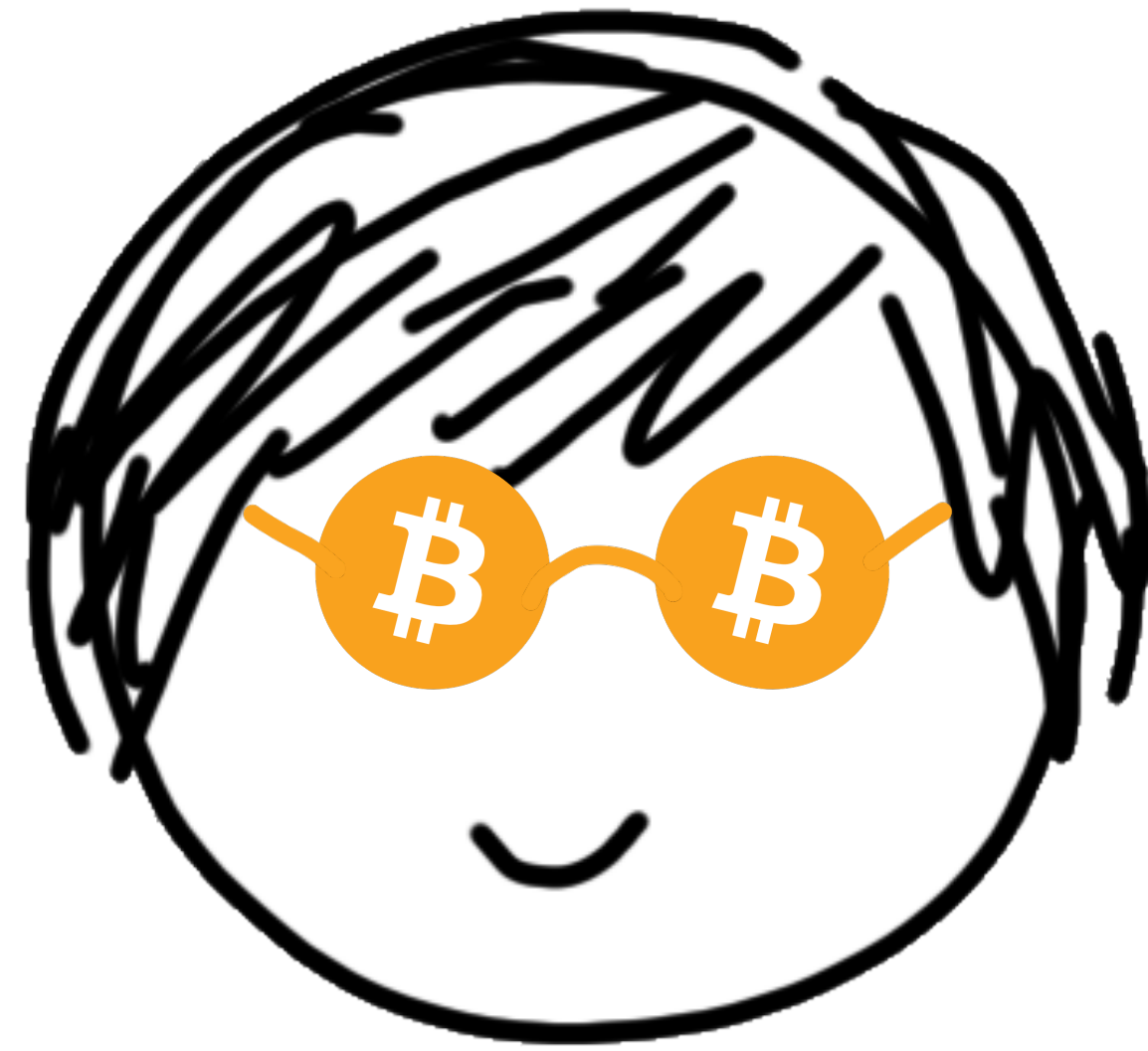
S:LENCE
LABORATORIES

Divya Ravi

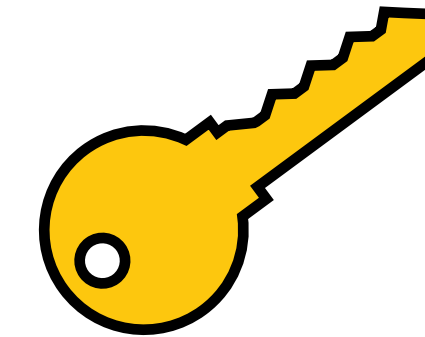
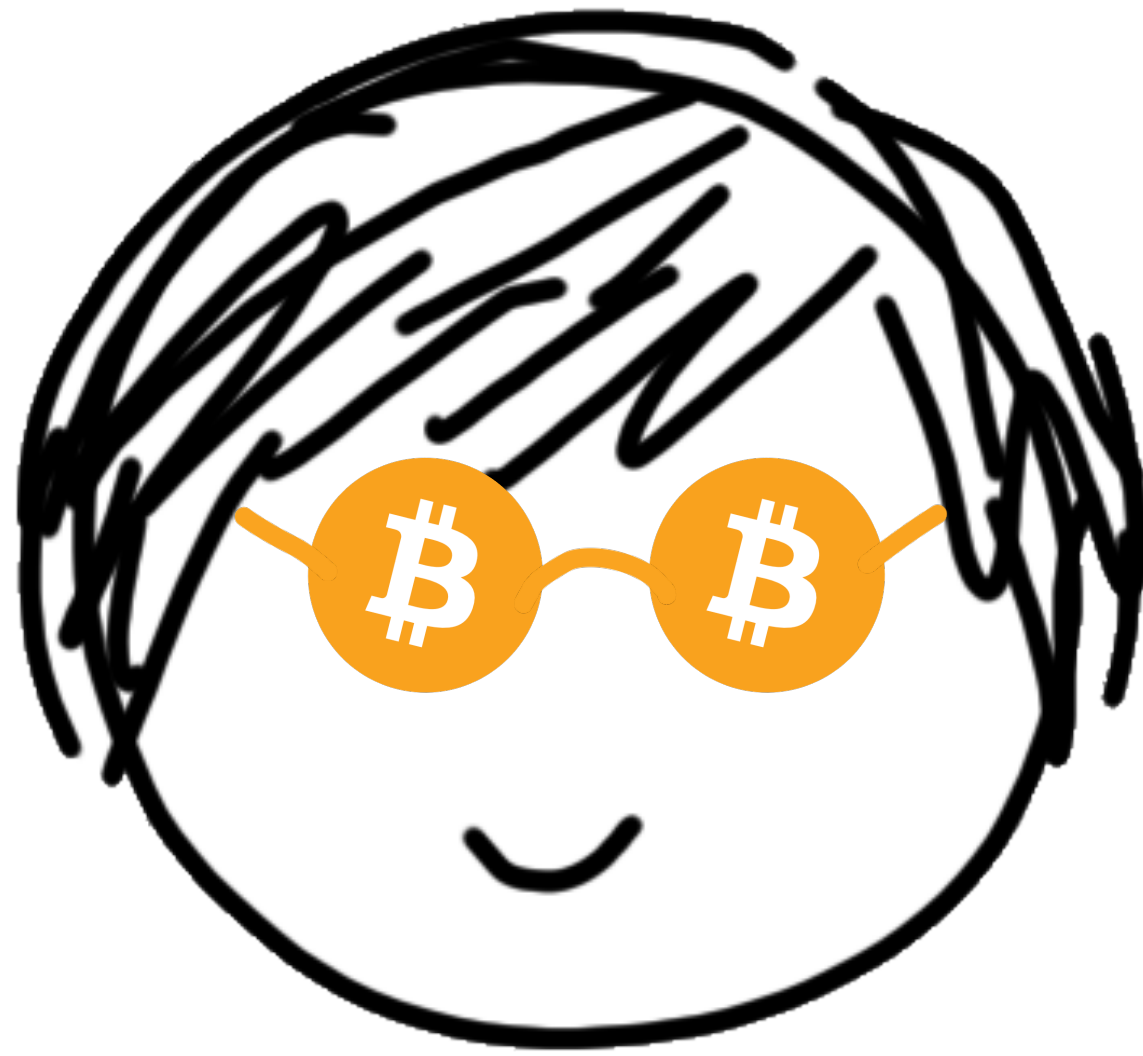


UNIVERSITY
OF AMSTERDAM

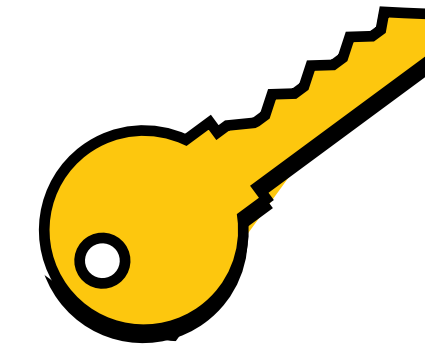
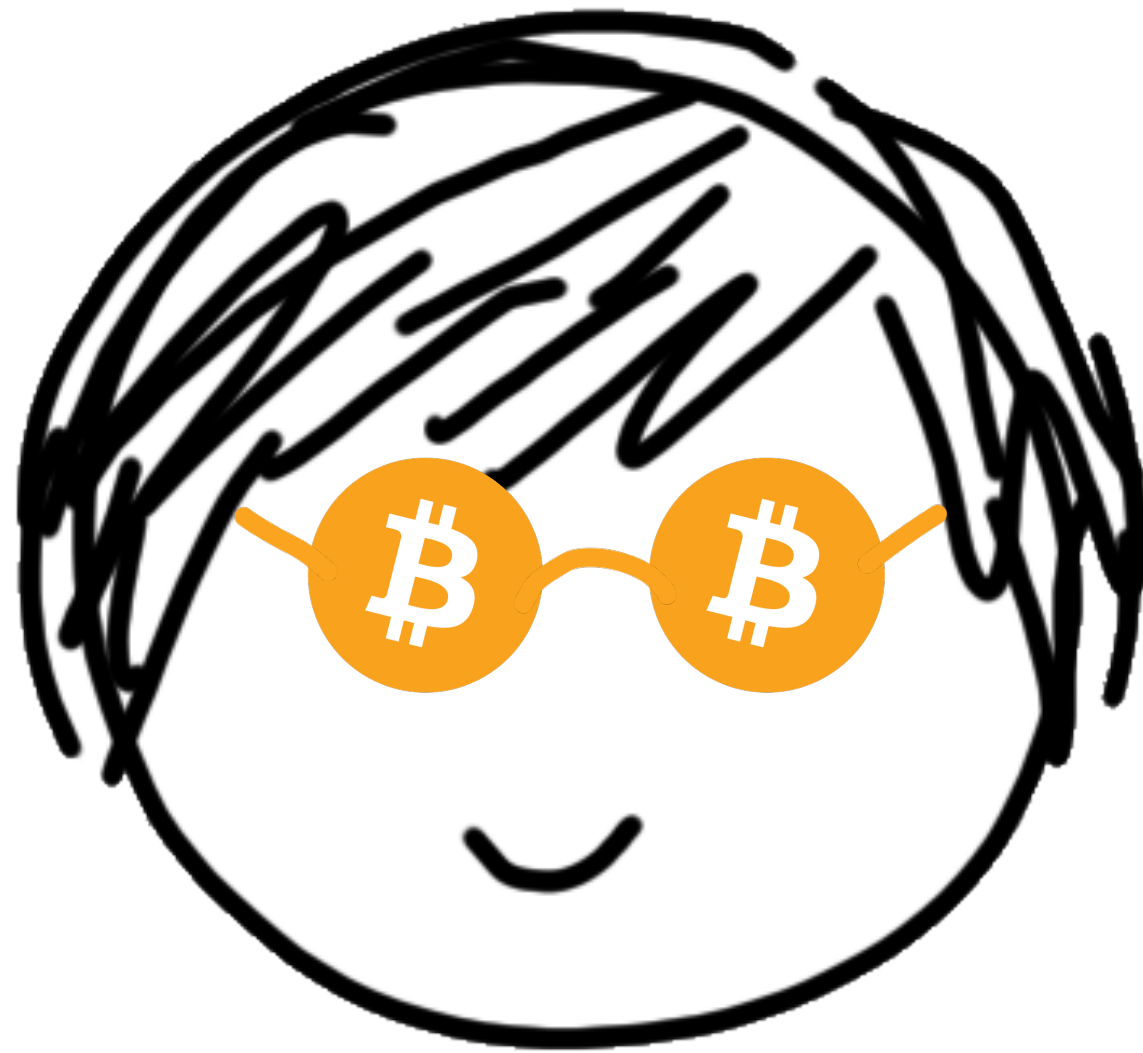
Threshold Signing



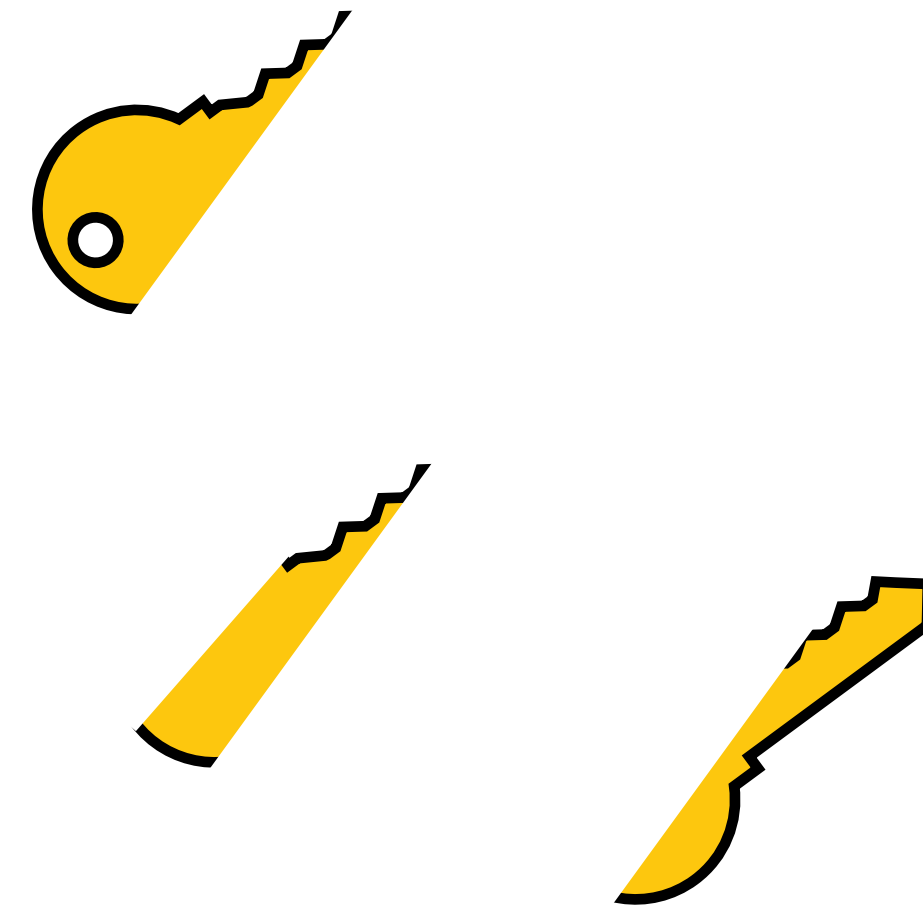
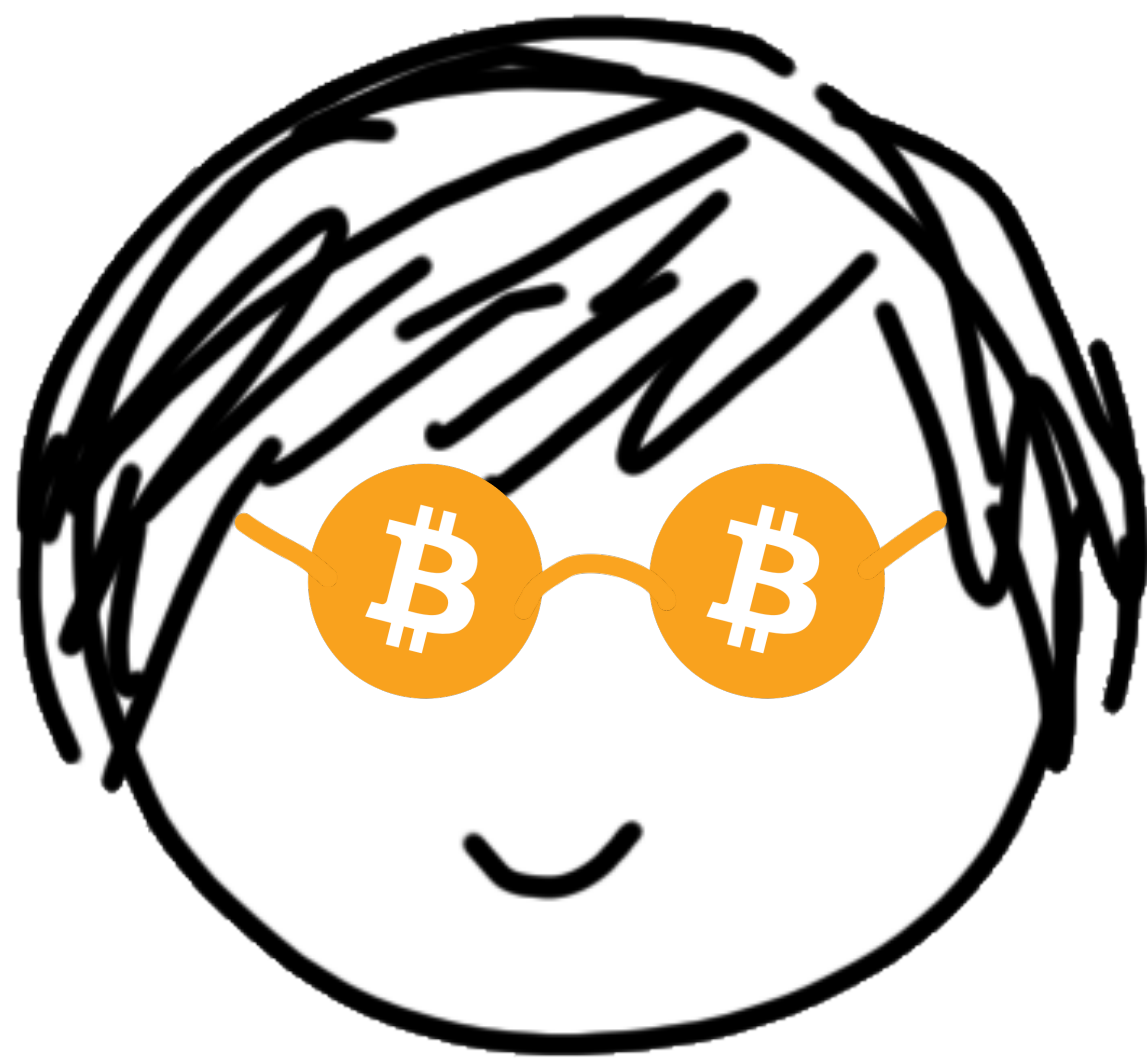
Threshold Signing



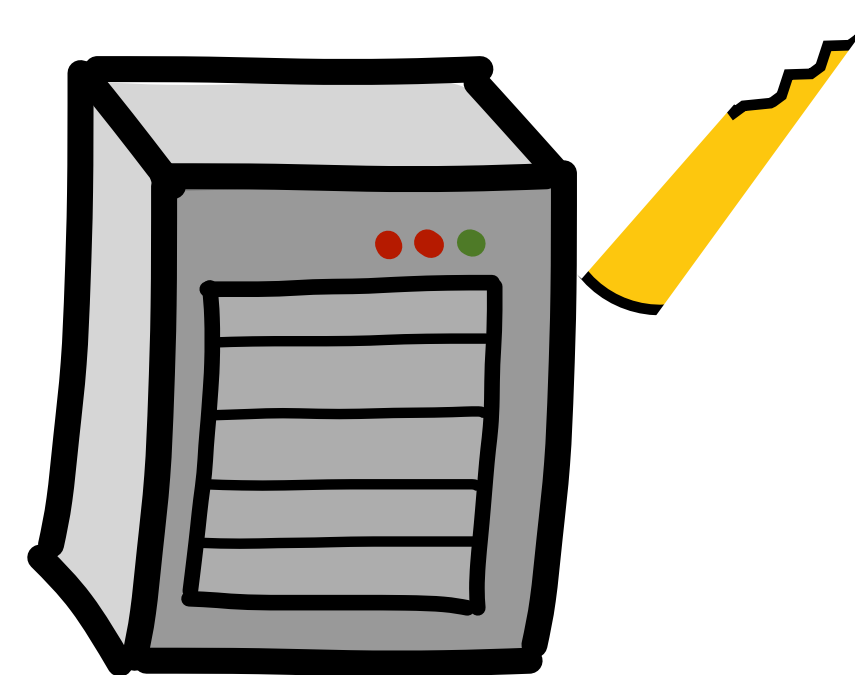
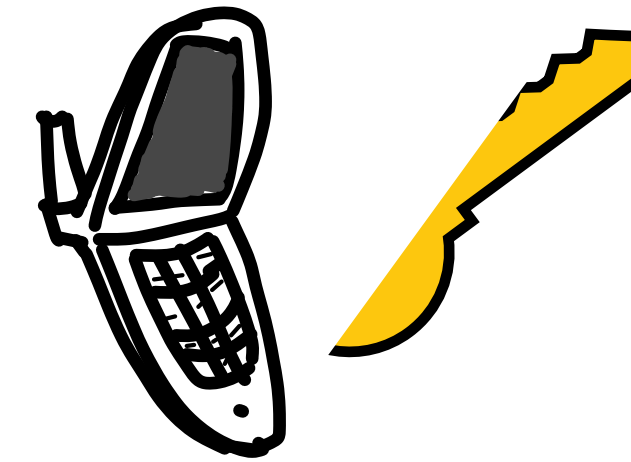
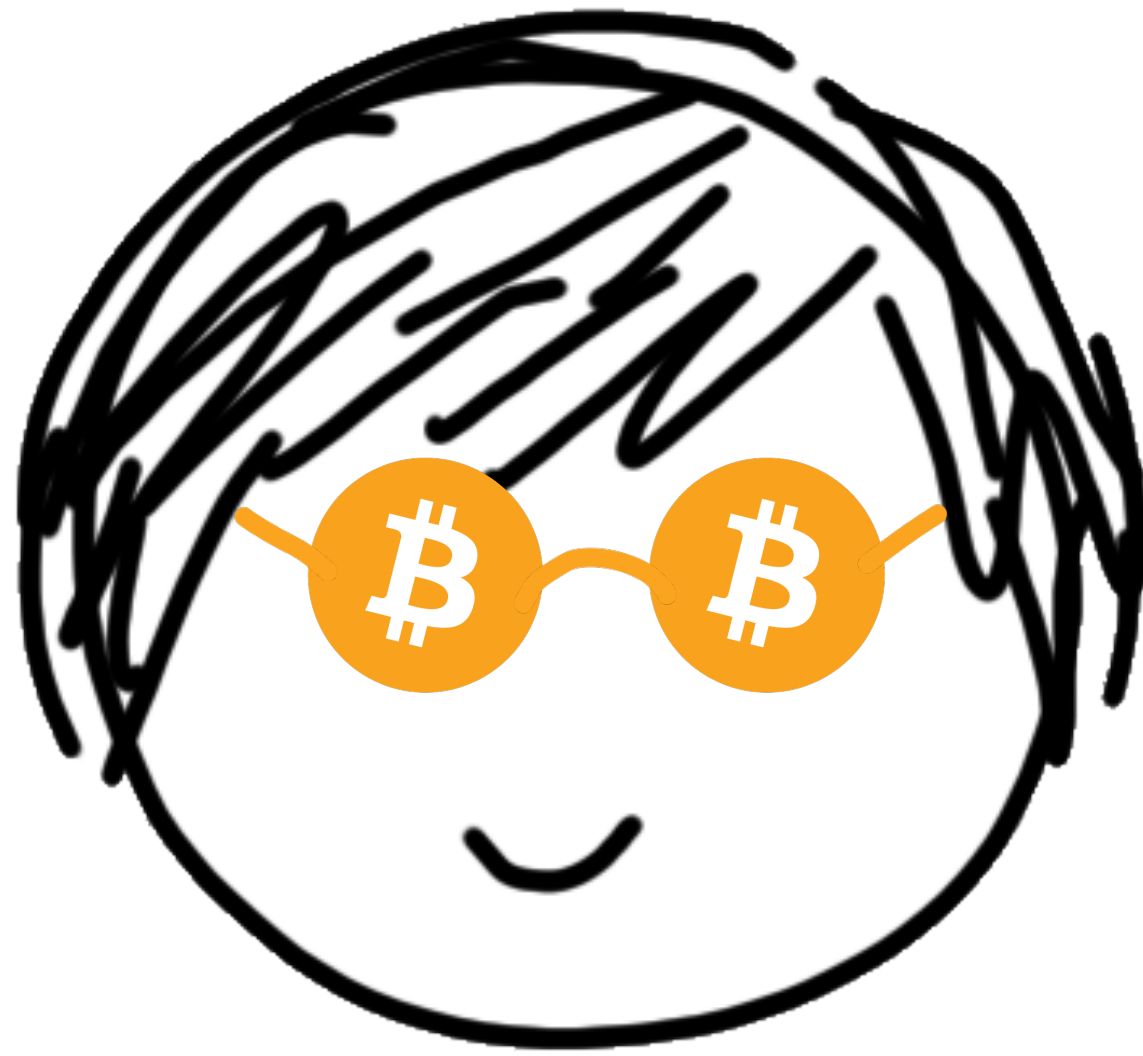
Threshold Signing



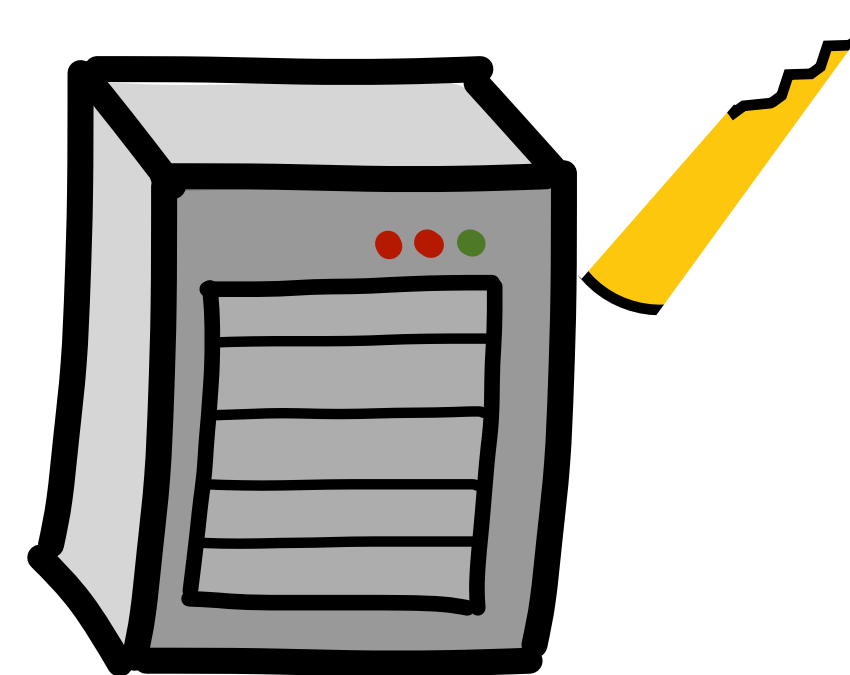
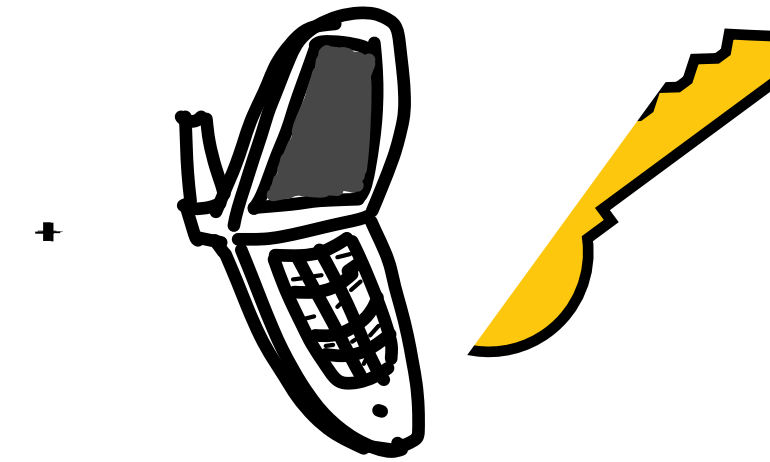
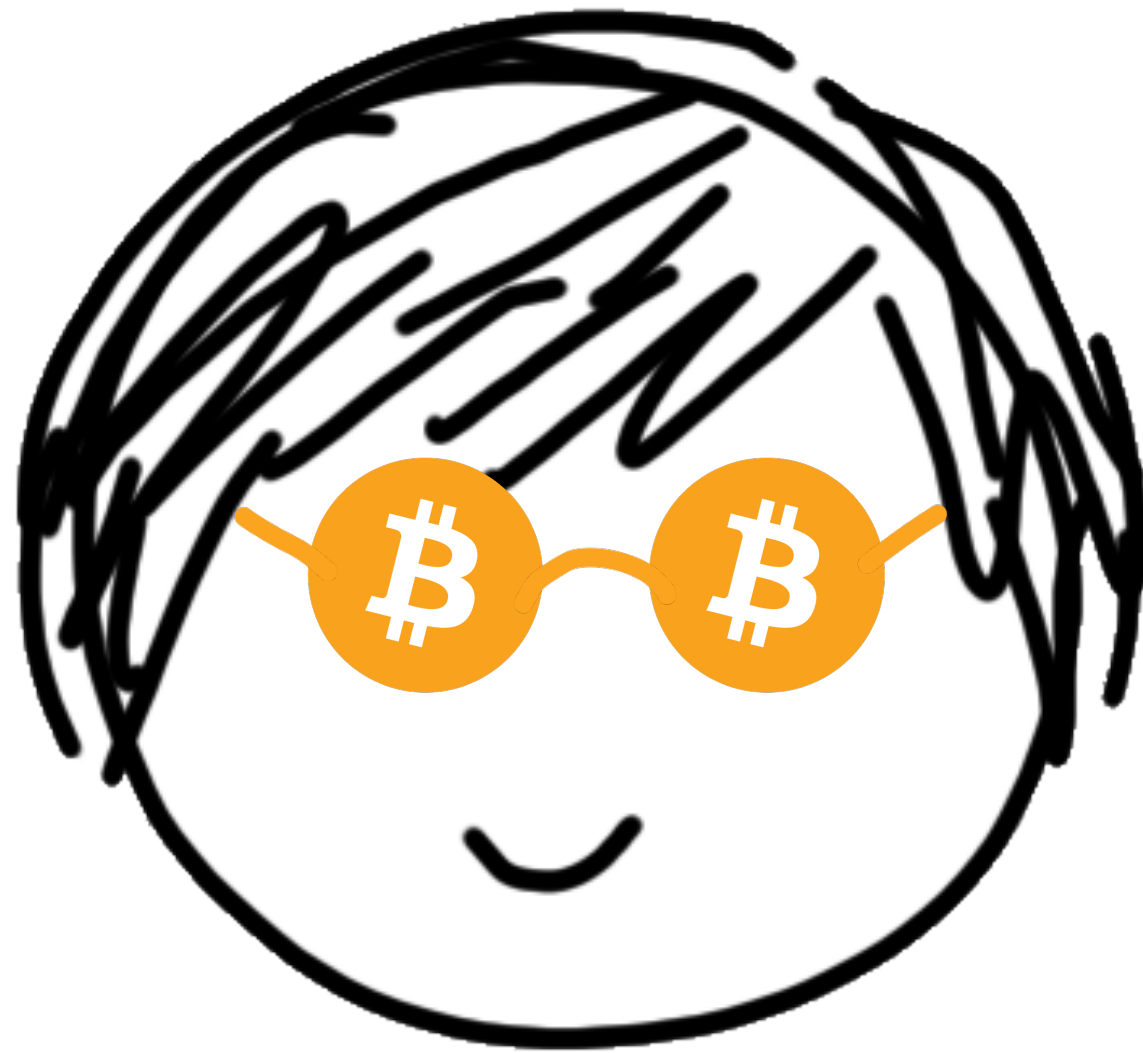
Threshold Signing



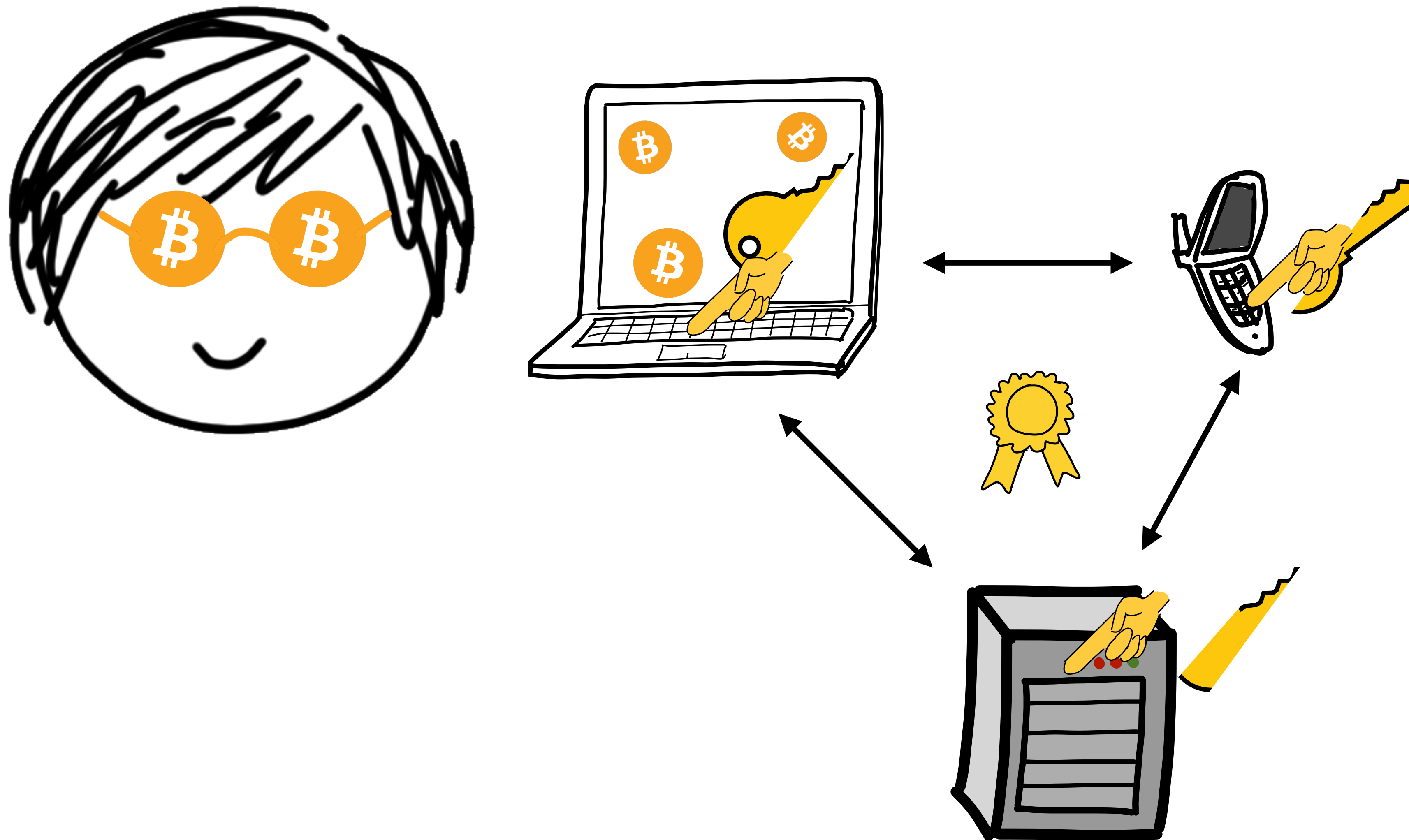
Threshold Signing



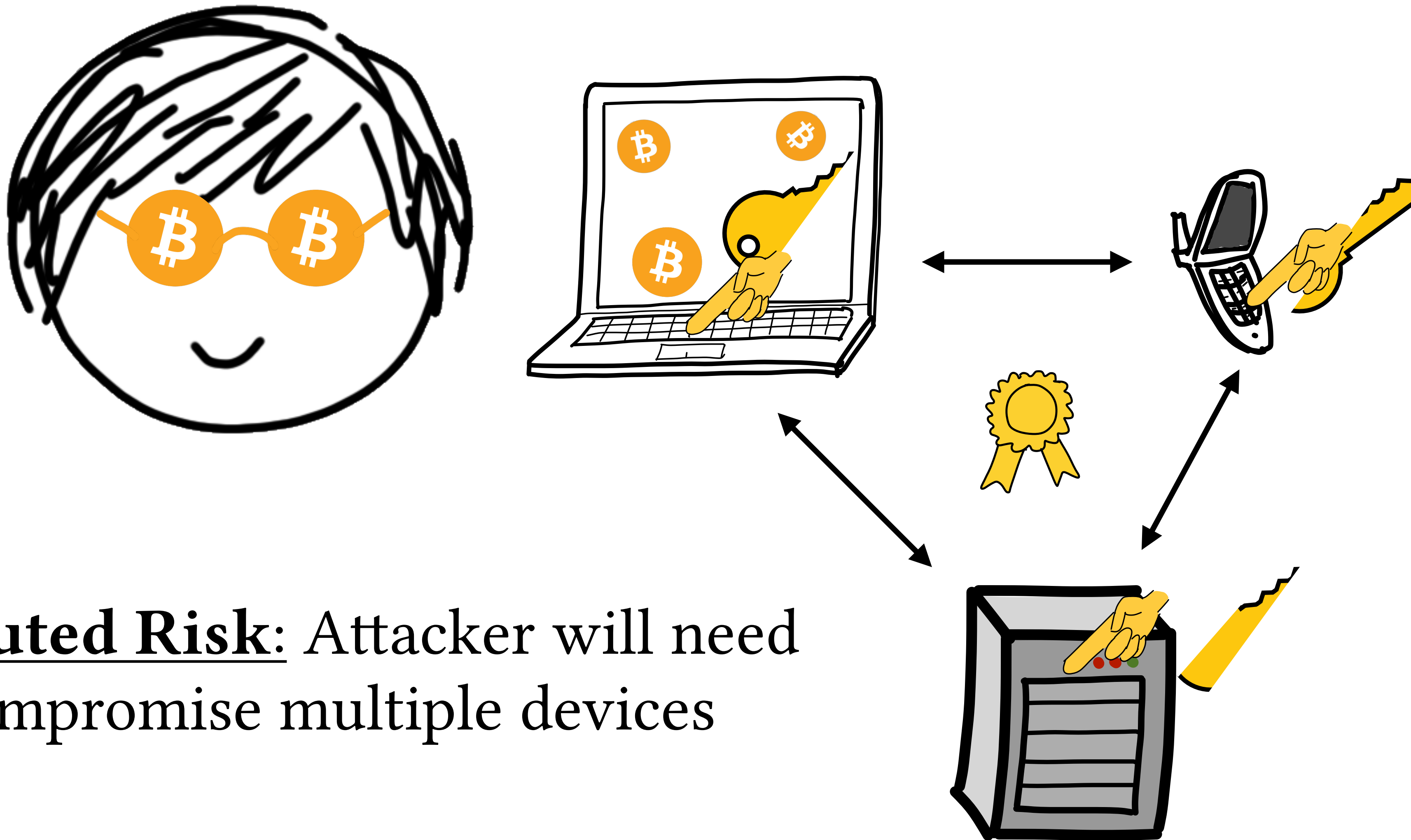
Threshold Signing



Threshold Signing



Threshold Signing



Distributed Risk: Attacker will need to compromise multiple devices

This Talk

This Talk

Setting

This Talk

Setting

Identifiable Abort: What, why, and how

This Talk

Setting

Identifiable Abort: What, why, and how

This work: Broadcast for Identifiable Abort

This Talk

Setting

Identifiable Abort: What, why, and how

This work: Broadcast for Identifiable Abort

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

This work: Broadcast for Identifiable Abort

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

This work: Broadcast for Identifiable Abort

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

Anatomy of an ECDSA-IA protocol

This work: Broadcast for Identifiable Abort

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

Anatomy of an ECDSA-IA protocol

This work: Broadcast for Identifiable Abort

Existing works assume secure broadcast (hard/expensive)

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

Anatomy of an ECDSA-IA protocol

This work: Broadcast for Identifiable Abort

Existing works assume secure broadcast (hard/expensive)

We define **Broadcast-IA**: impossible for $t < n$, simple for $t < n/2$

ECDSA-IA

This Talk

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

Anatomy of an ECDSA-IA protocol

This work: Broadcast for Identifiable Abort

Existing works assume secure broadcast (hard/expensive)

We define **Broadcast-IA**: impossible for $t < n$, simple for $t < n/2$

ECDSA-IA

Clean honest majority ECDSA-IA protocol

This Talk

The case for honest majority

- Many settings have a *global* honest majority anyway
- HM is necessary for fundamental IA building block
 - when using p2p channels only
- Clean ECDSA protocol
 - MPC is easier with HM (no OT/Paillier necessary)

Setting

Running example: network of nodes managing shares

Identifiable Abort: What, why, and how

Mitigating Denial of Service attacks

Anatomy of an ECDSA-IA protocol

This work: Broadcast for Identifiable Abort

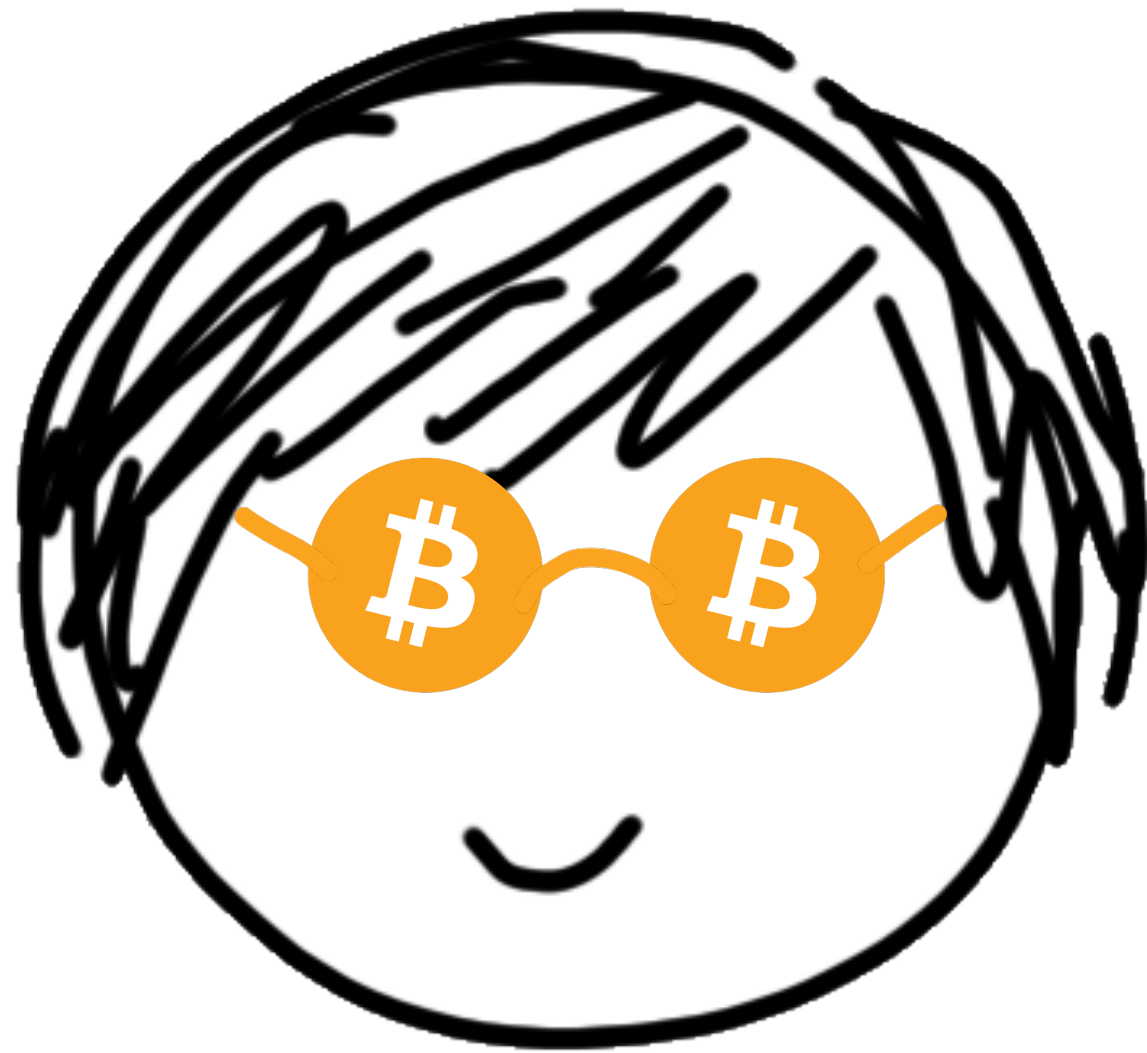
Existing works assume secure broadcast (hard/expensive)

We define **Broadcast-IA**: impossible for $t < n$, simple for $t < n/2$

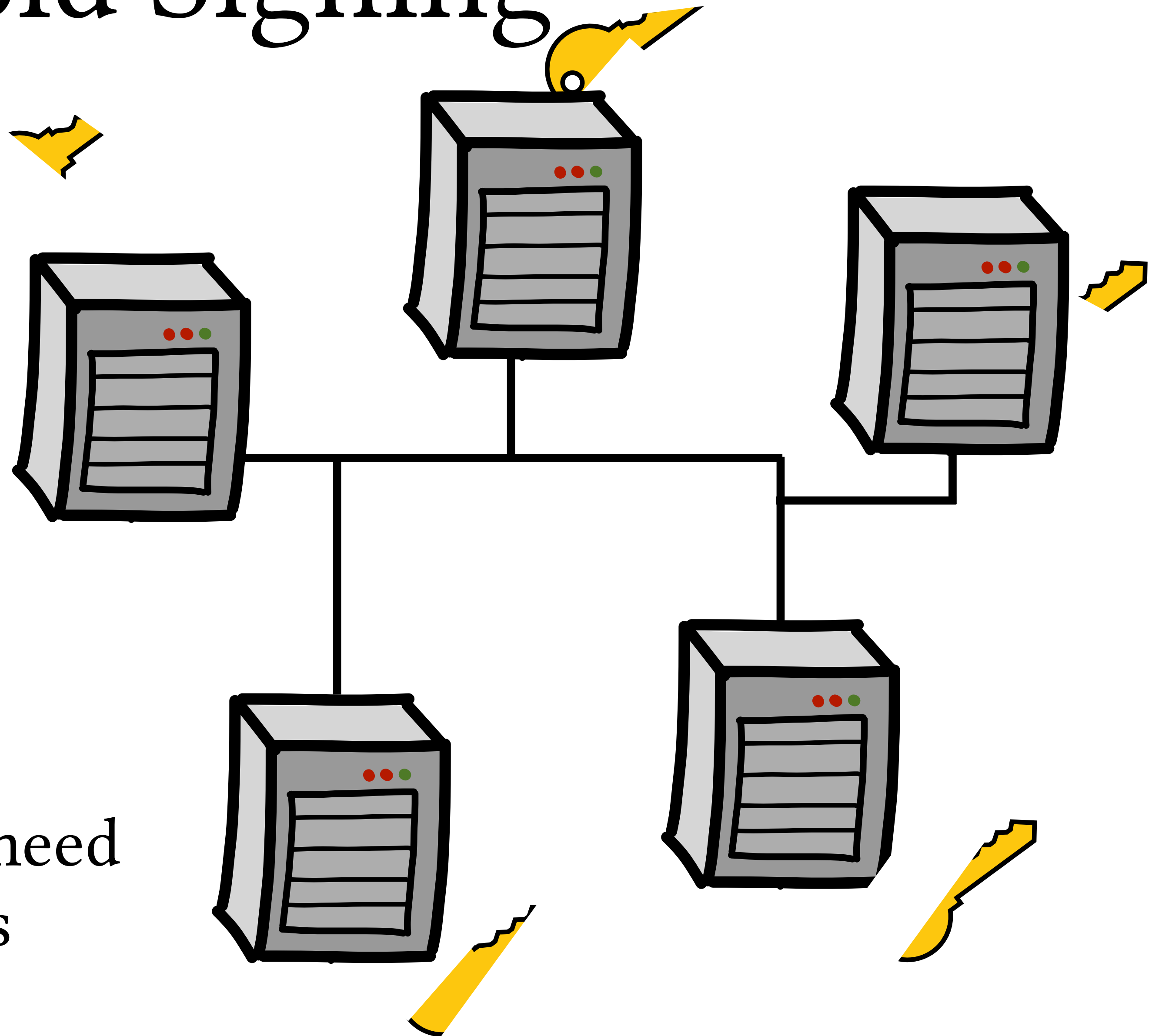
ECDSA-IA

Clean honest majority ECDSA-IA protocol

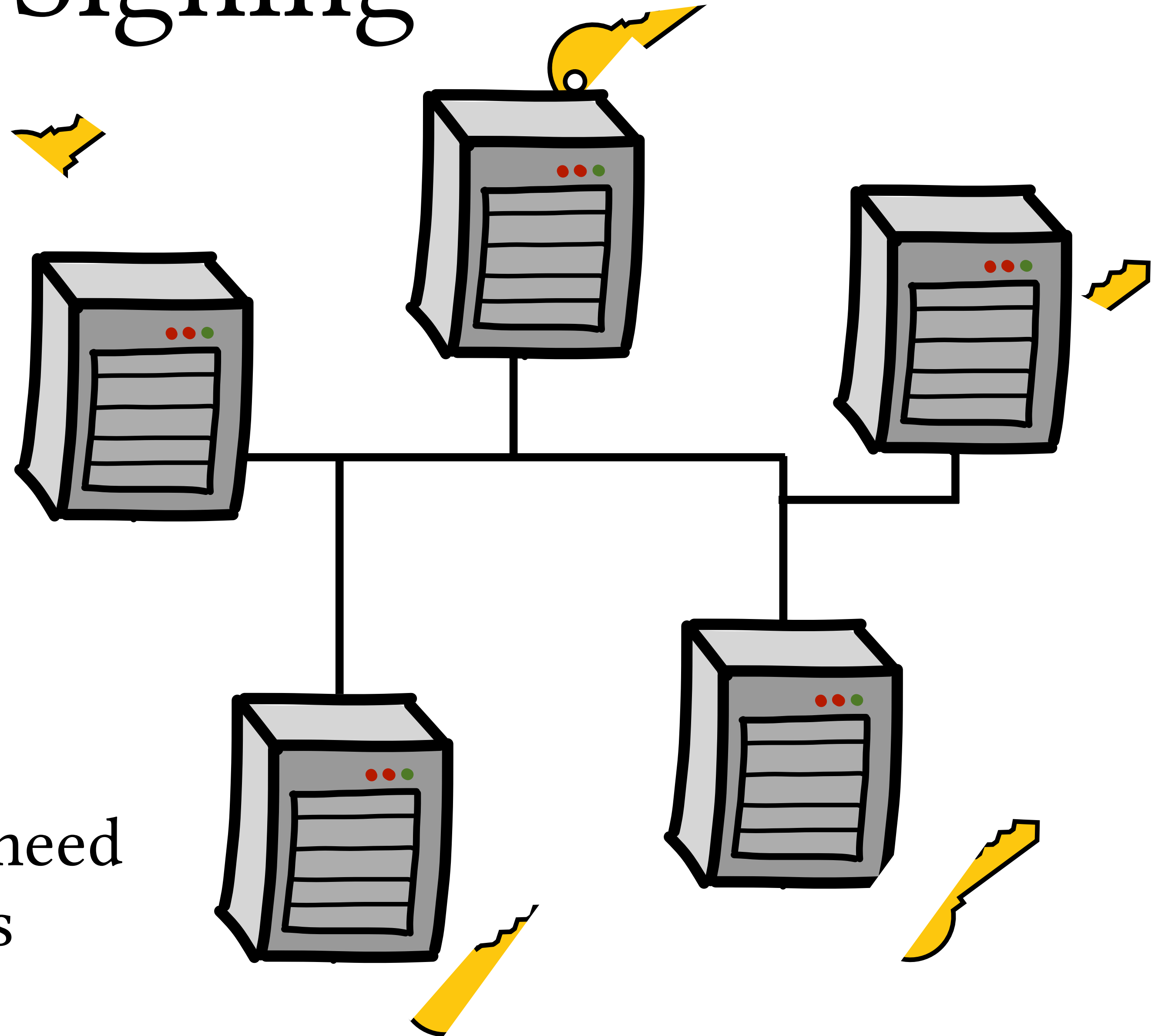
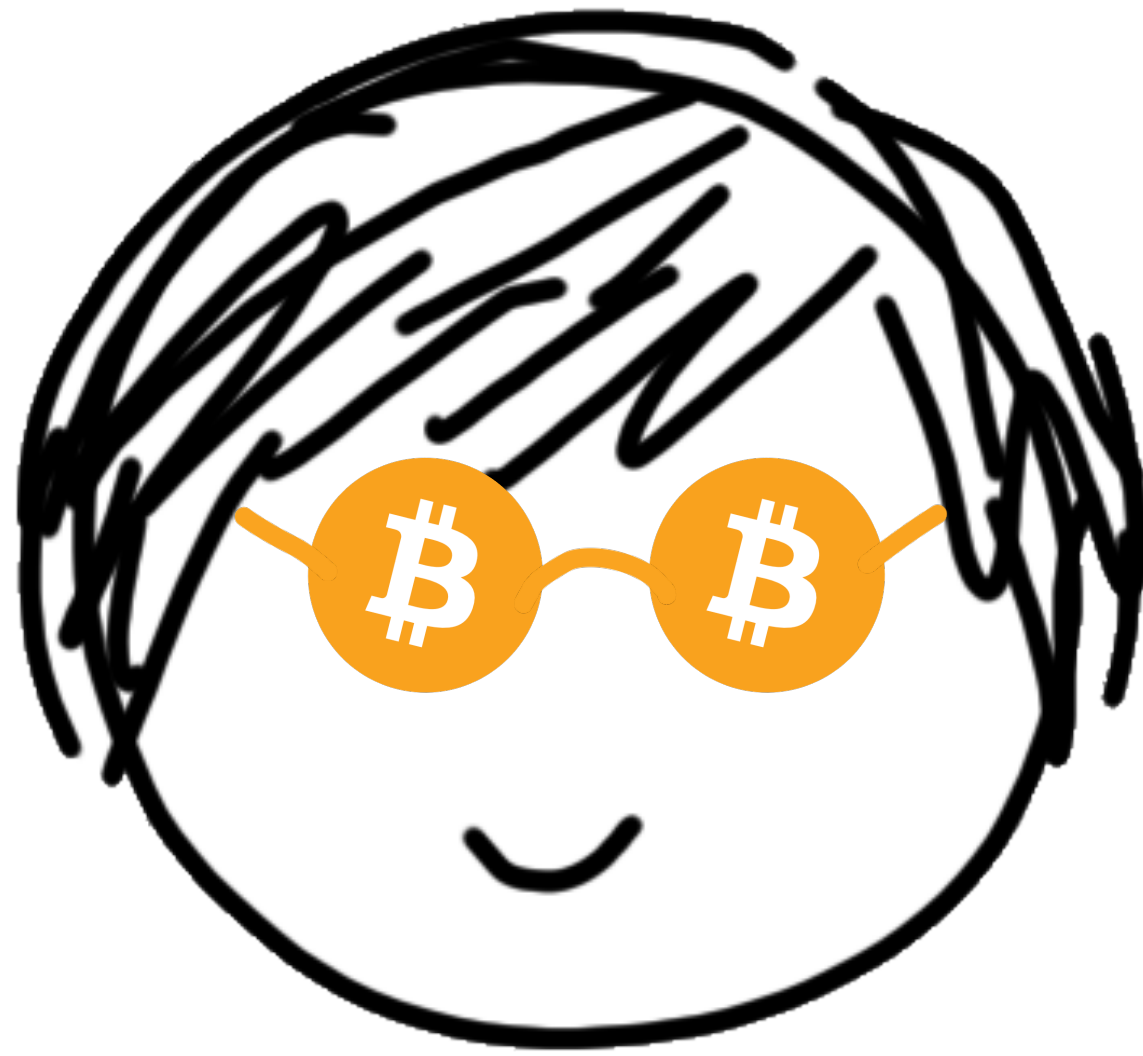
Threshold Signing



Distributed Risk: Attacker will need to compromise multiple nodes

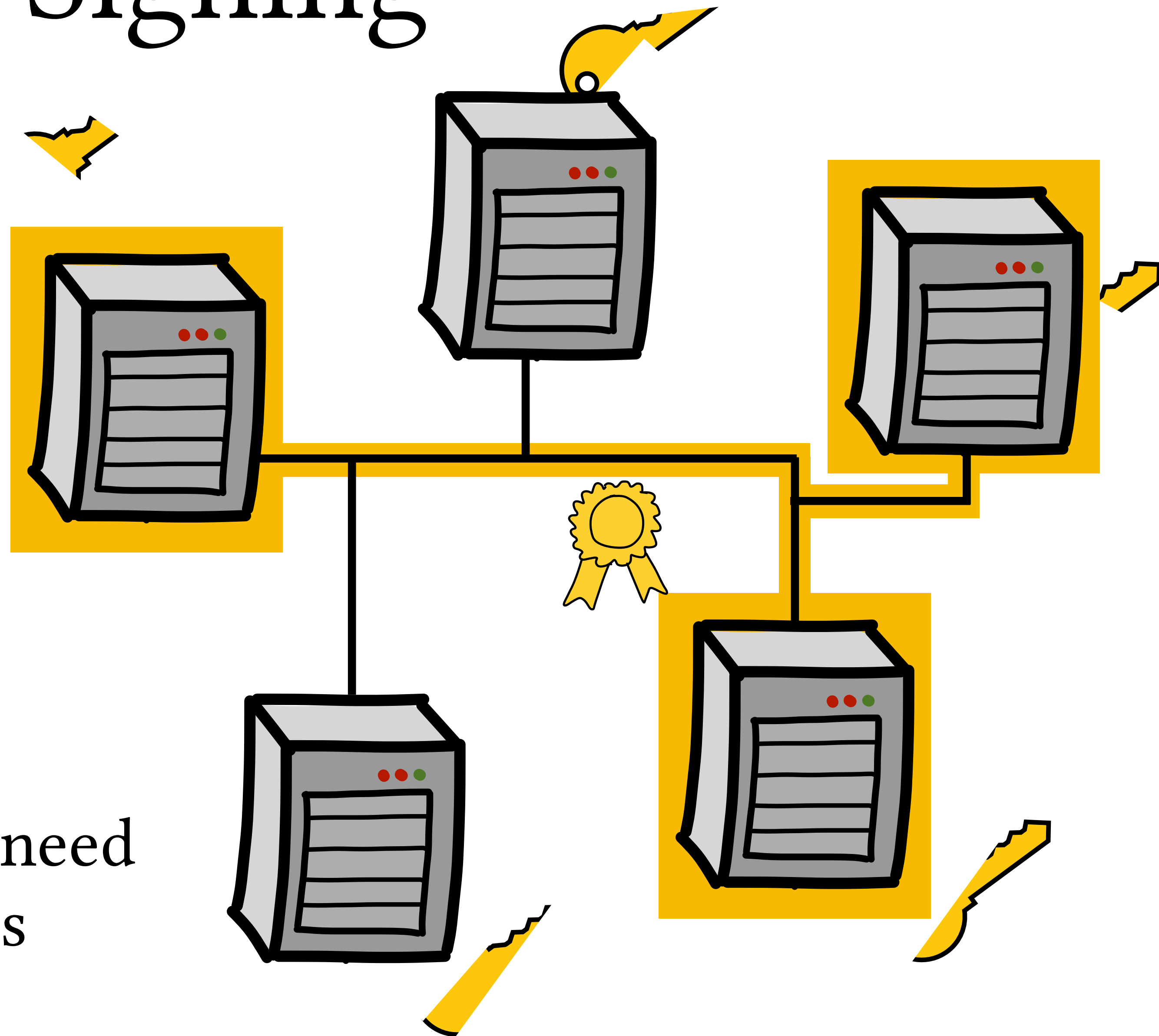
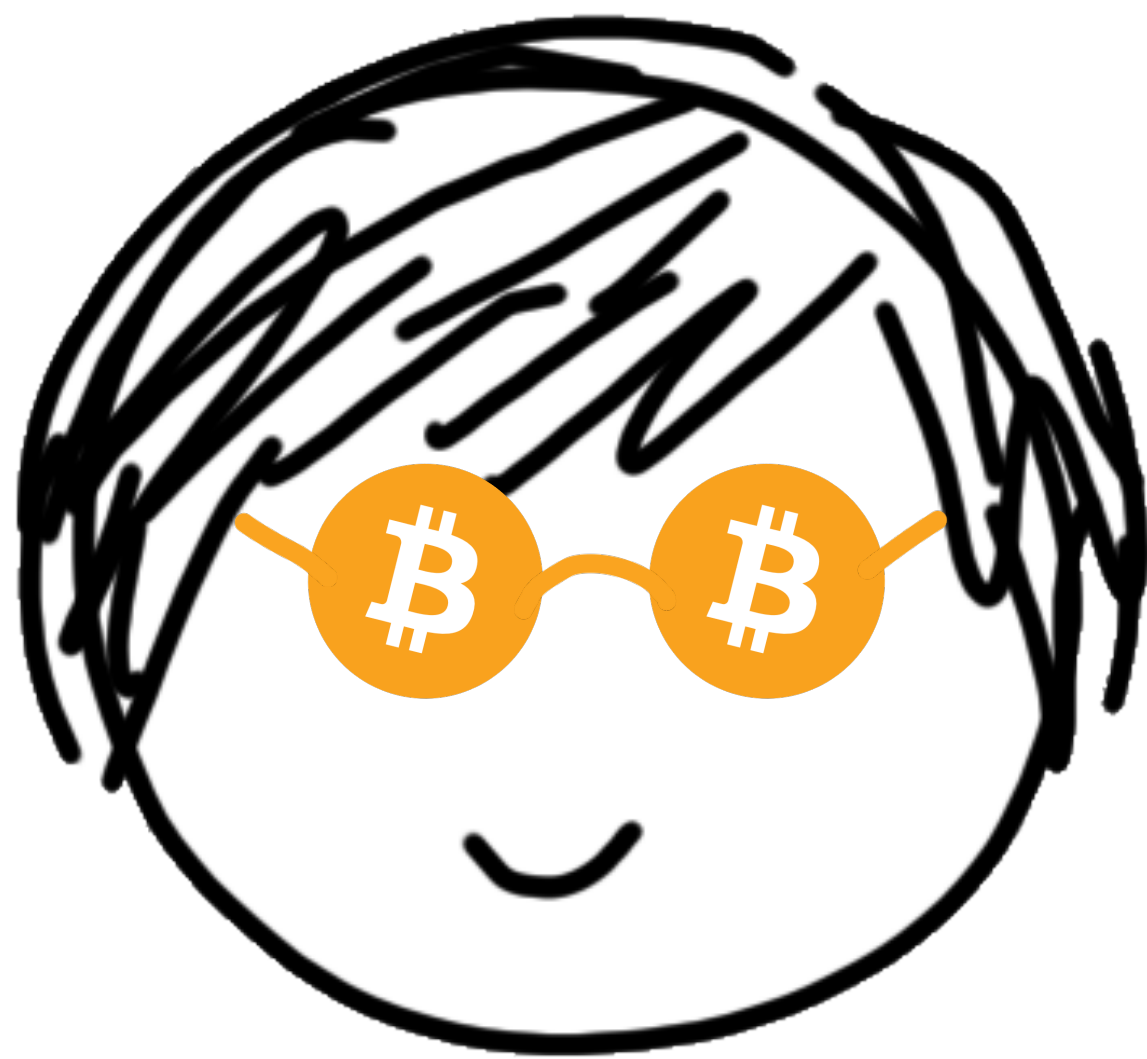


$(3,n)$ Signing



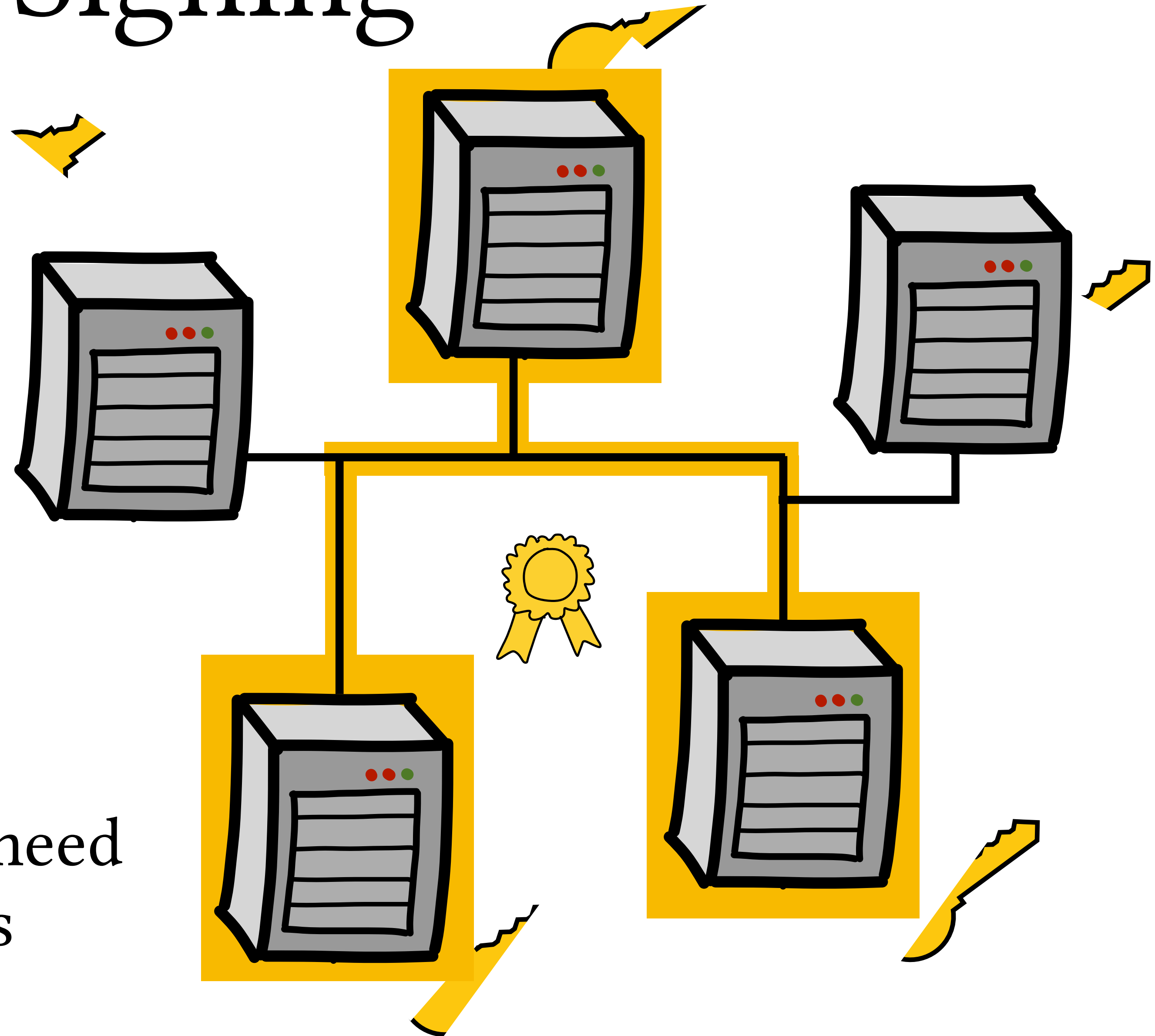
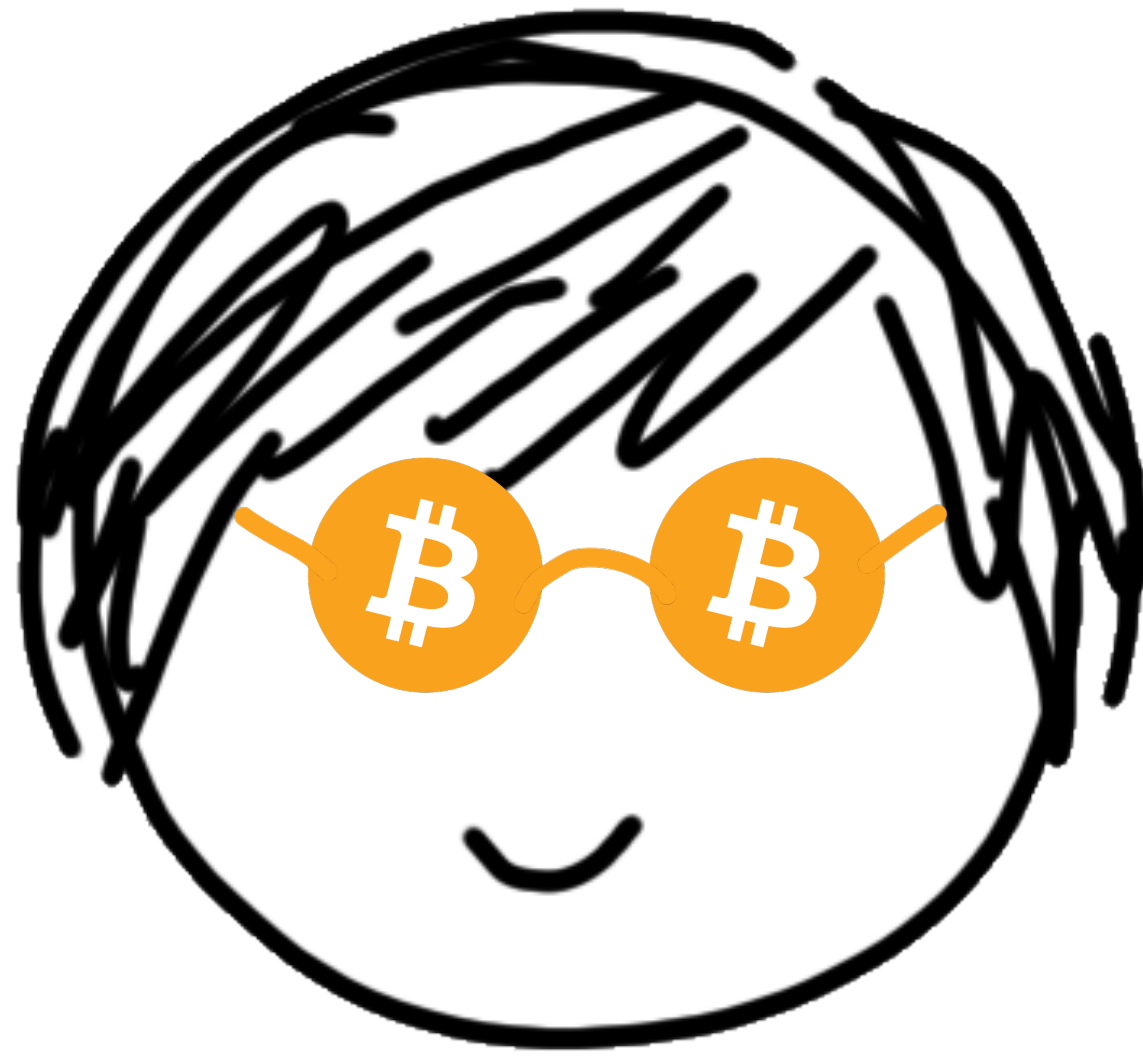
Distributed Risk: Attacker will need to compromise multiple nodes

$(3,n)$ Signing



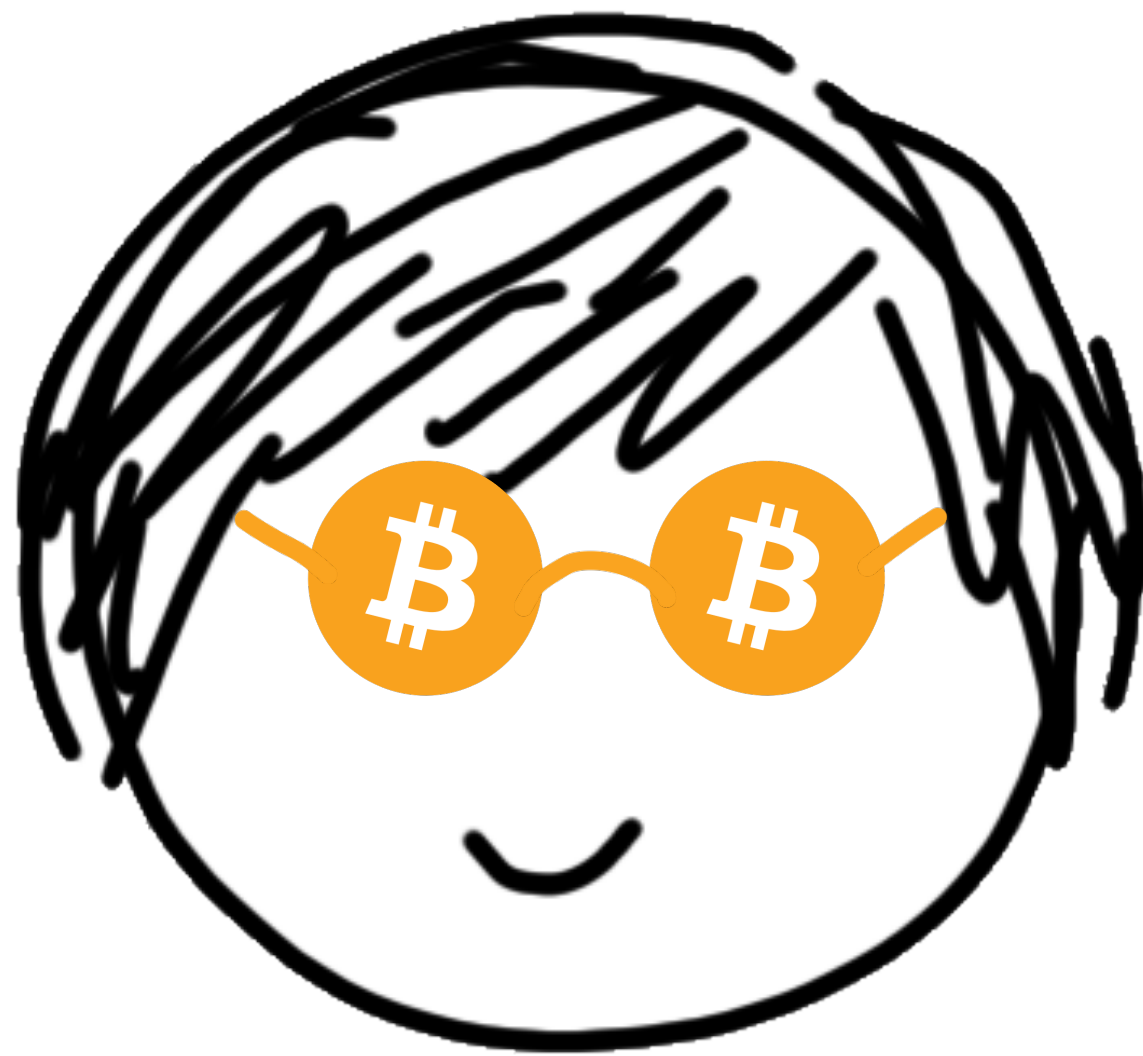
Distributed Risk: Attacker will need
to compromise multiple nodes

$(3,n)$ Signing

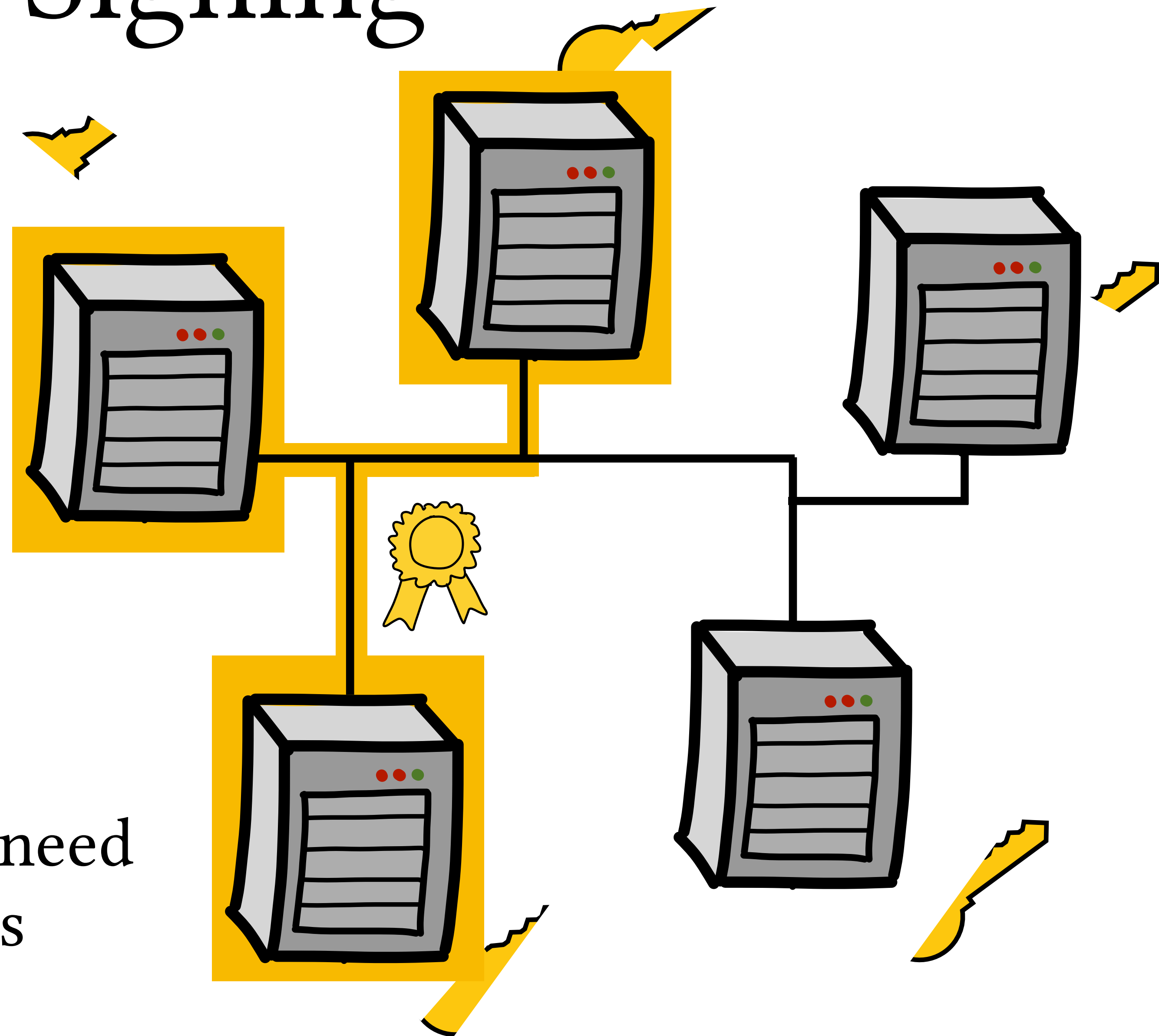


Distributed Risk: Attacker will need to compromise multiple nodes

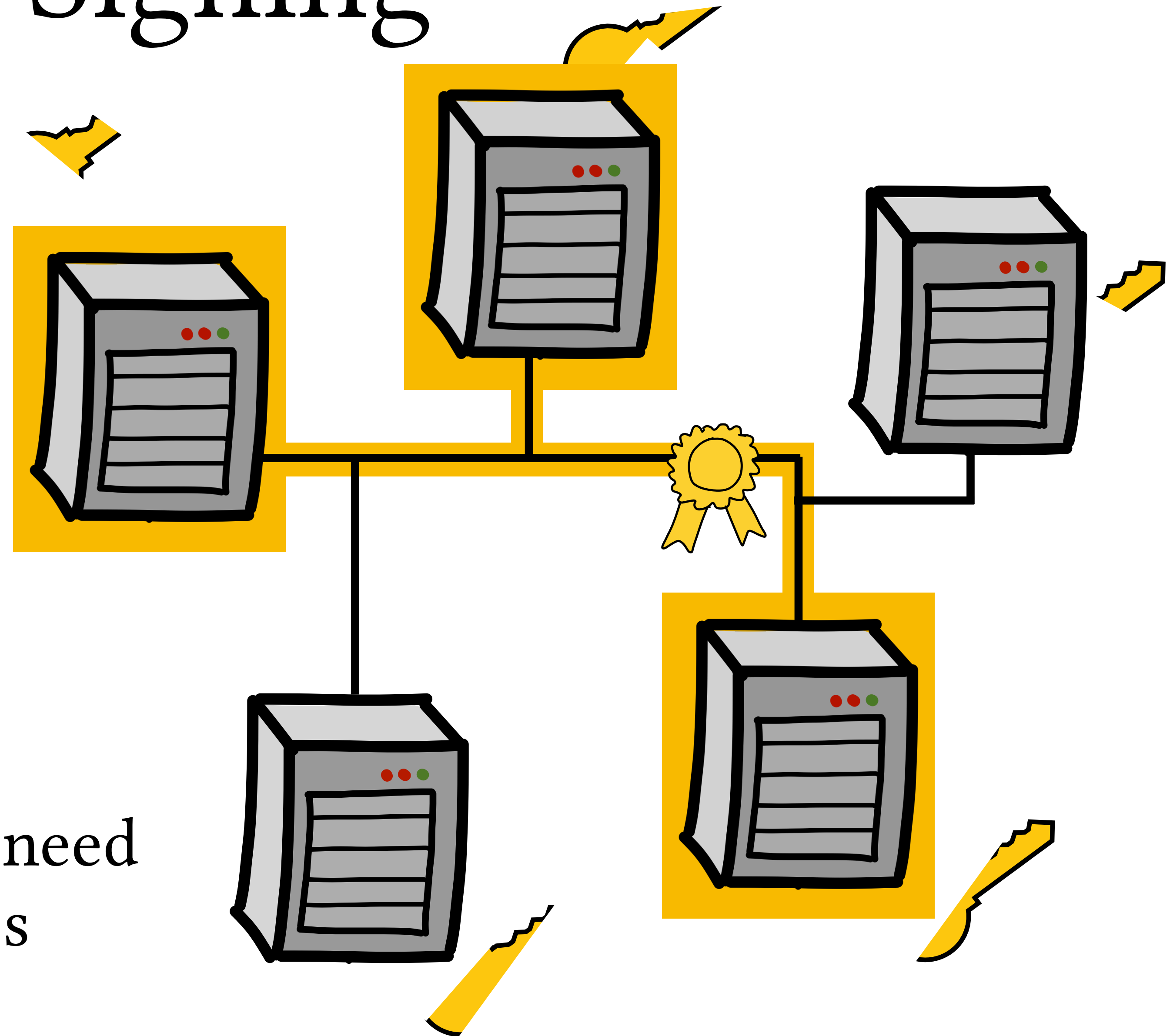
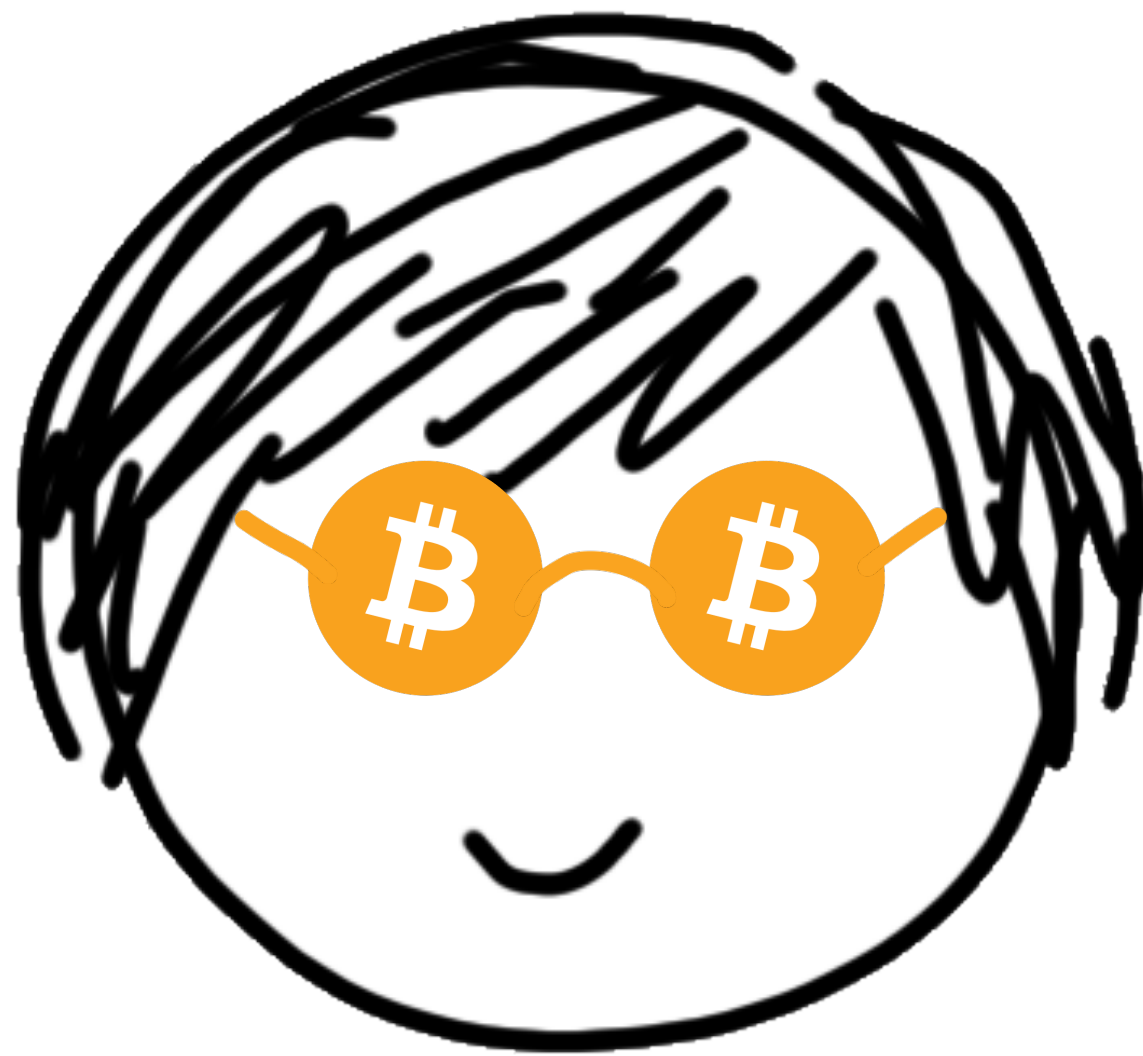
$(3,n)$ Signing



Distributed Risk: Attacker will need to compromise multiple nodes

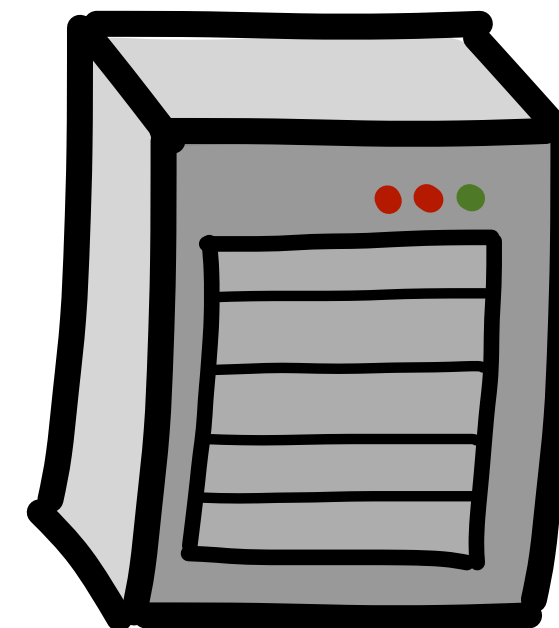
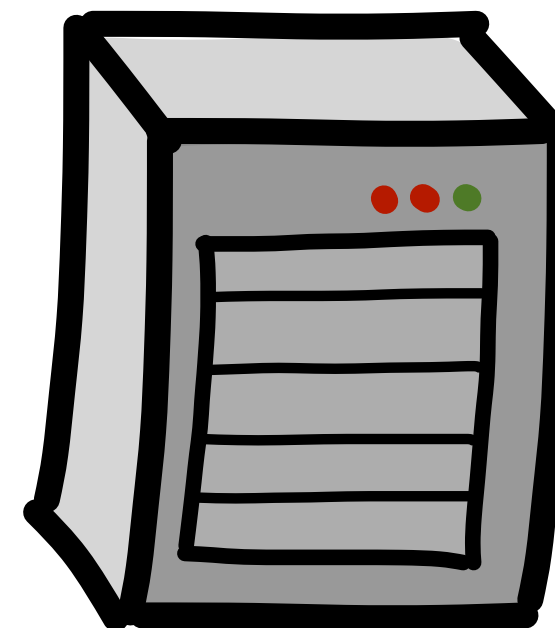
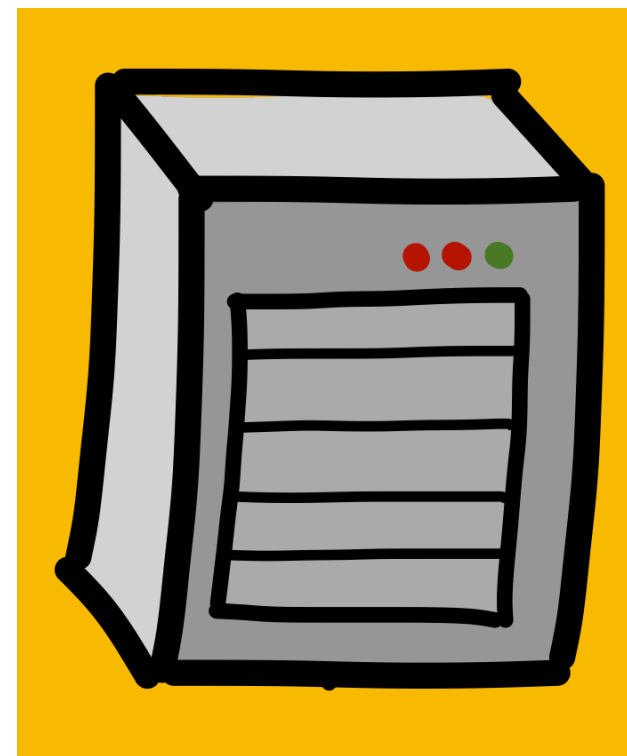
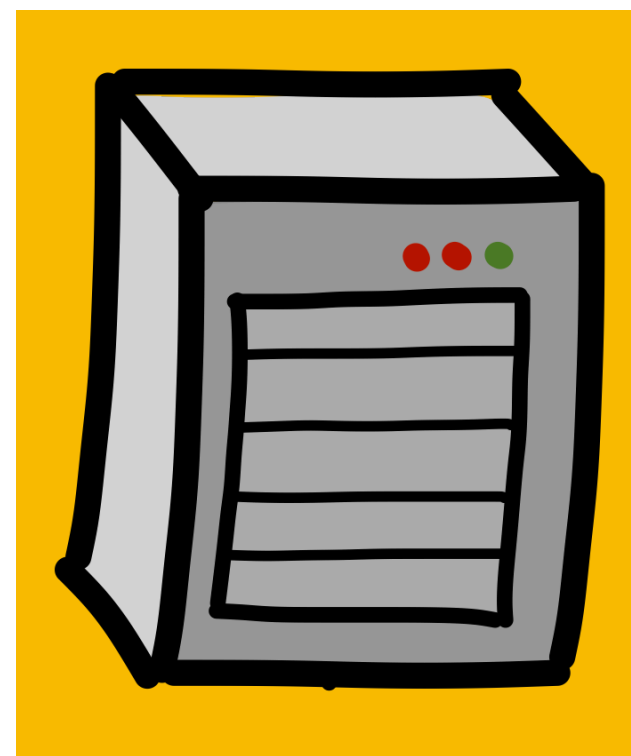
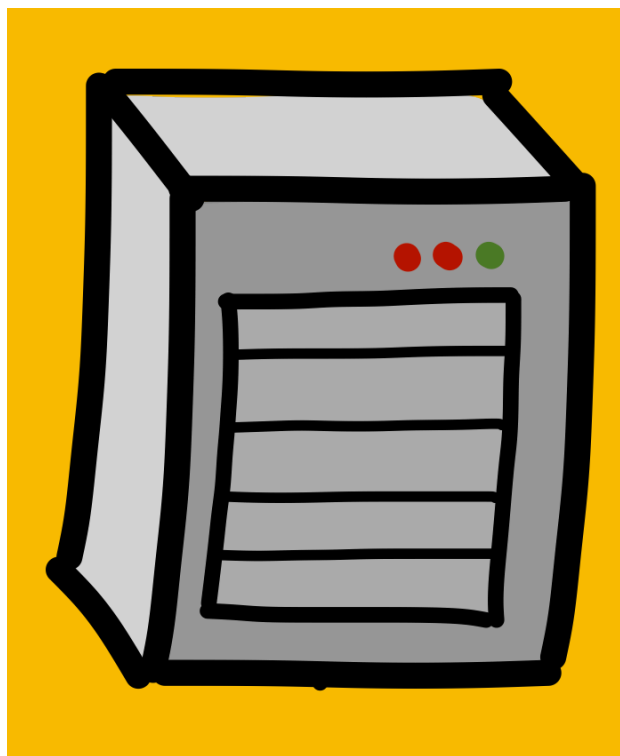


$(3,n)$ Signing



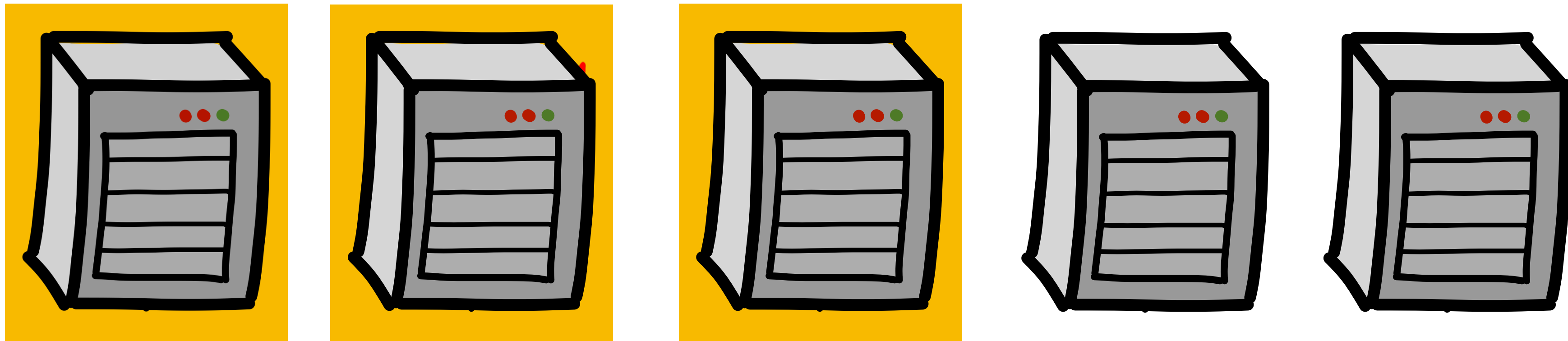
Distributed Risk: Attacker will need to compromise multiple nodes

$(3,n)$ Signing



$(3, n)$ Signing

Best Possible Security: Protection against 2 corruptions



$(3,n)$ Signing

Best Possible Security: Protection against 2 corruptions



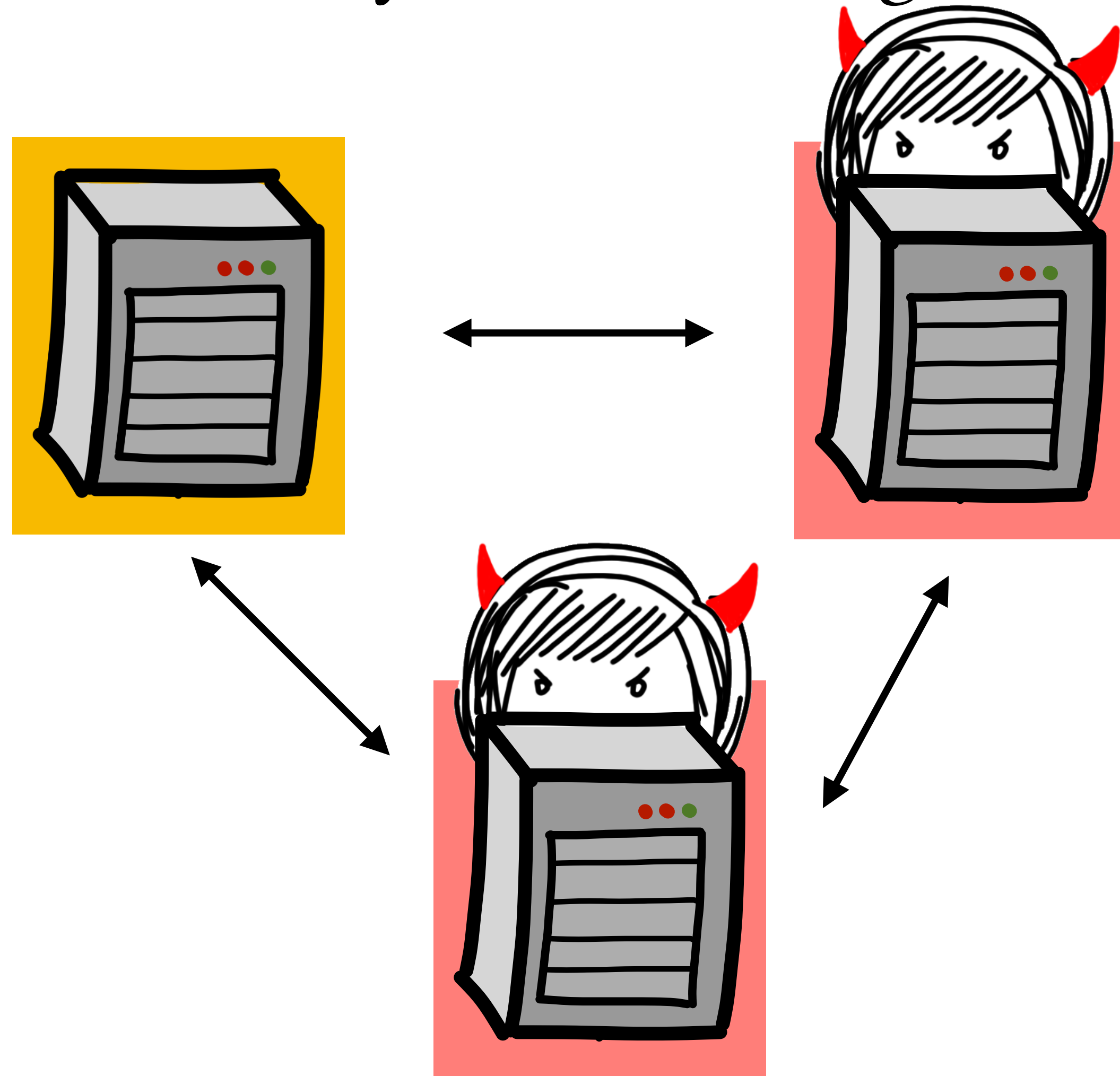
...out of five parties

“Global” honest majority

Necessary to retrieve  in case of a fault

$(3,n)$ Signing

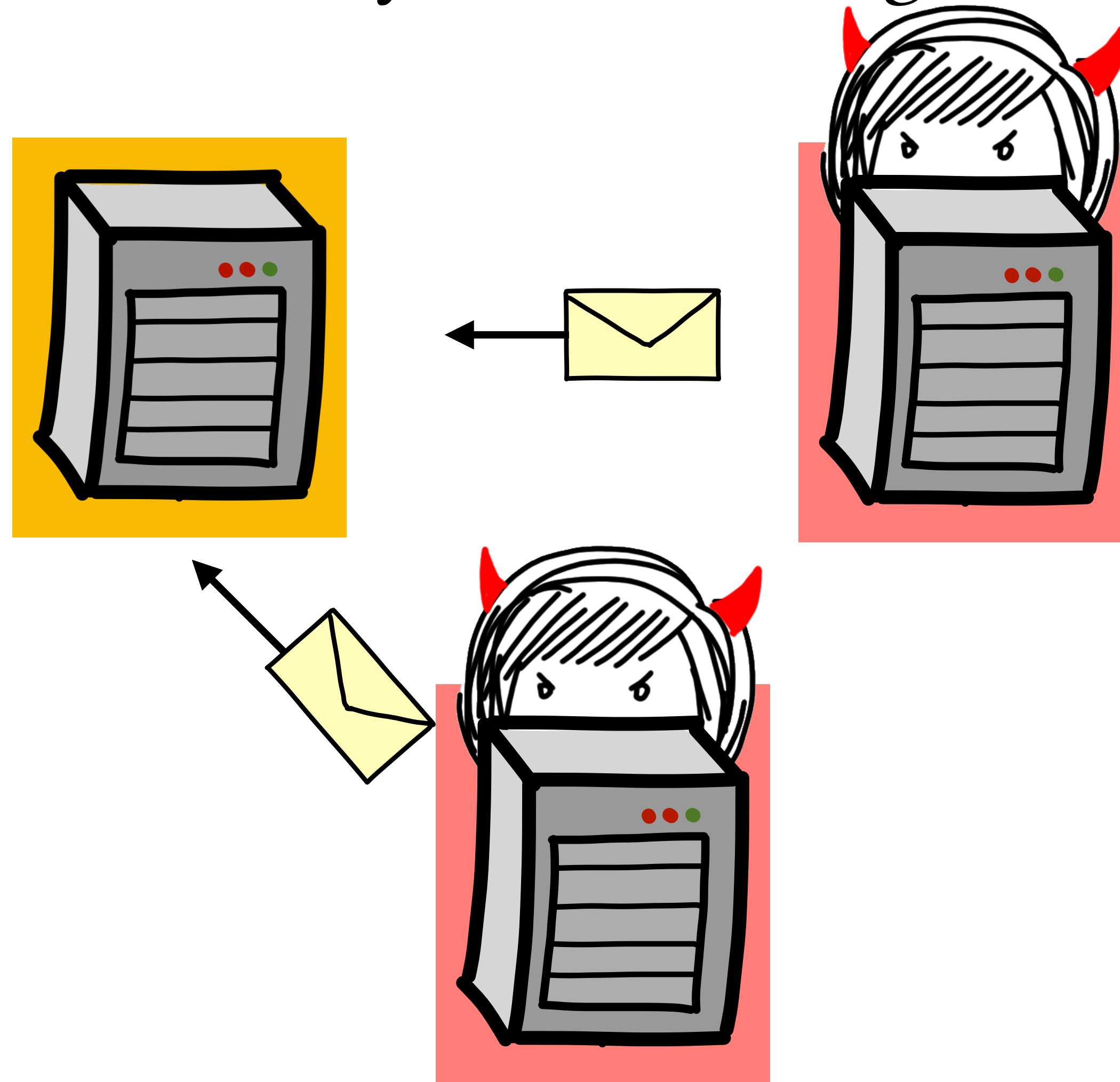
Best Possible Security: Protection against 2 corruptions



Threshold signing via
“Dishonest majority”
MPC protocol

$(3,n)$ Signing

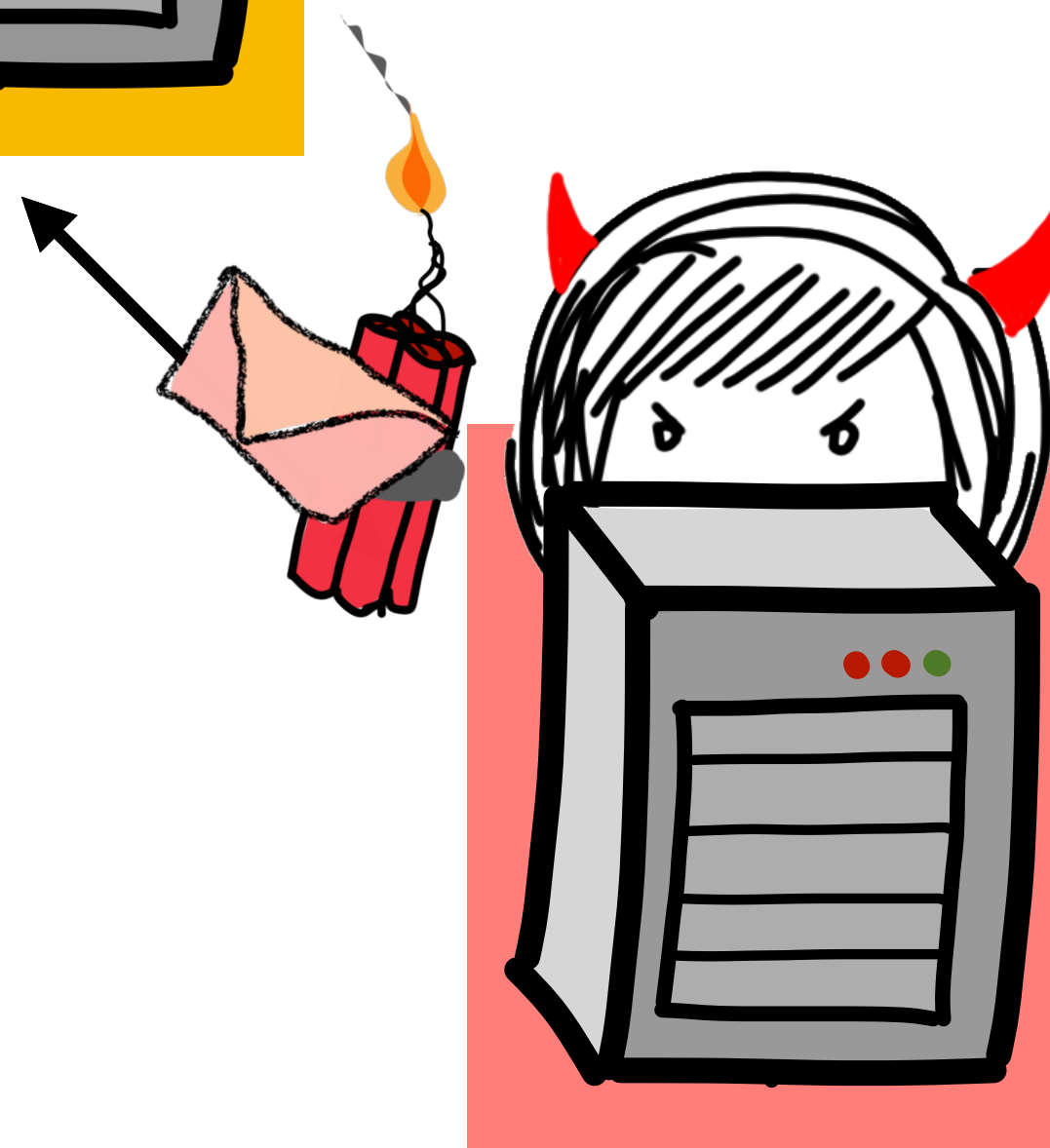
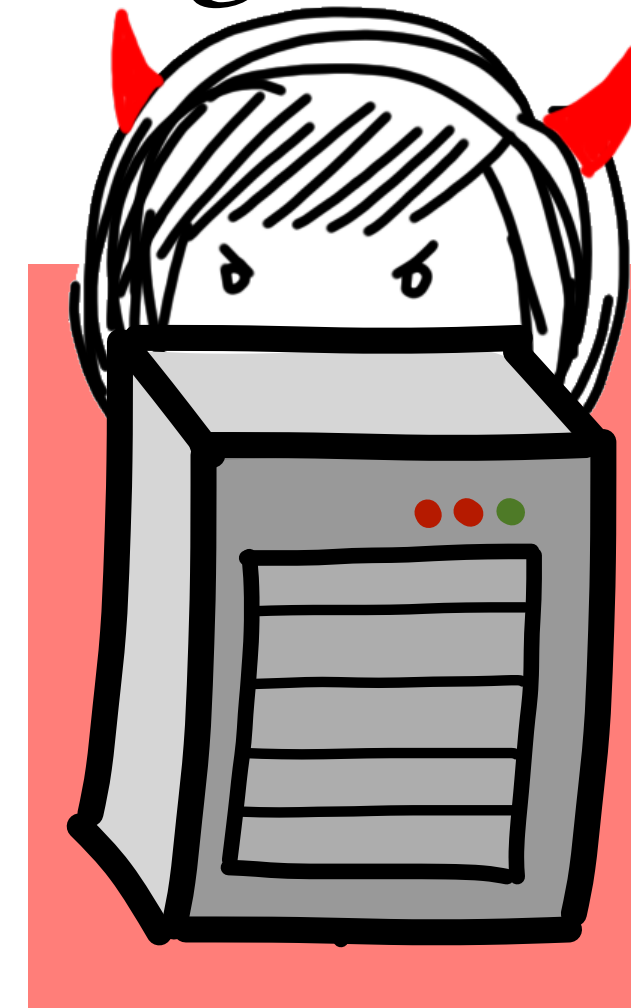
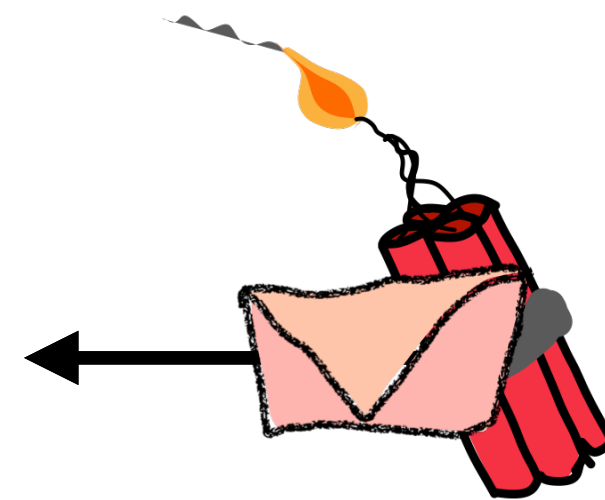
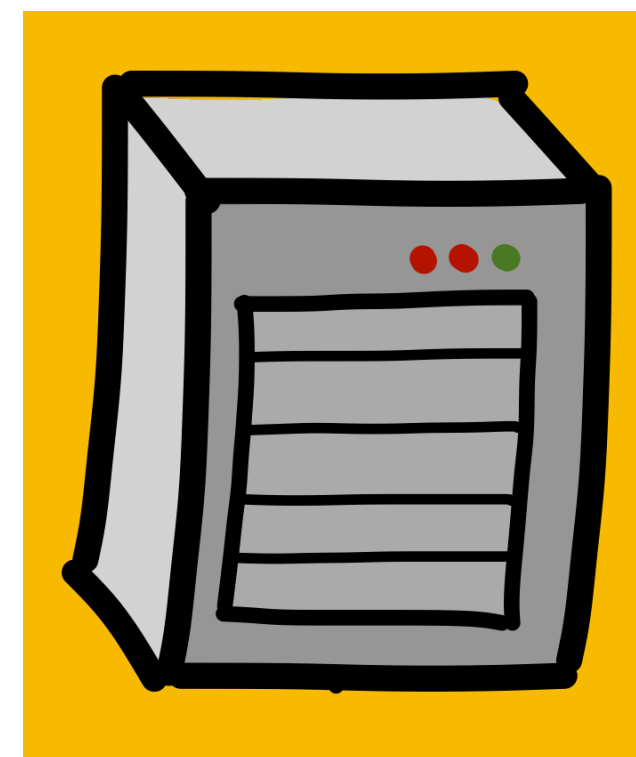
Best Possible Security: Protection against 2 corruptions



Threshold signing via
“Dishonest majority”
MPC protocol

$(3,n)$ Signing

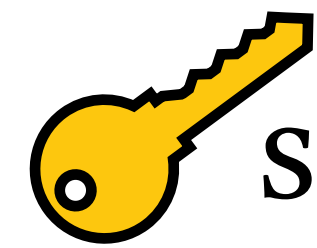
Best Possible Security: Protection against 2 corruptions



Threshold signing via
“Dishonest majority”
MPC protocol

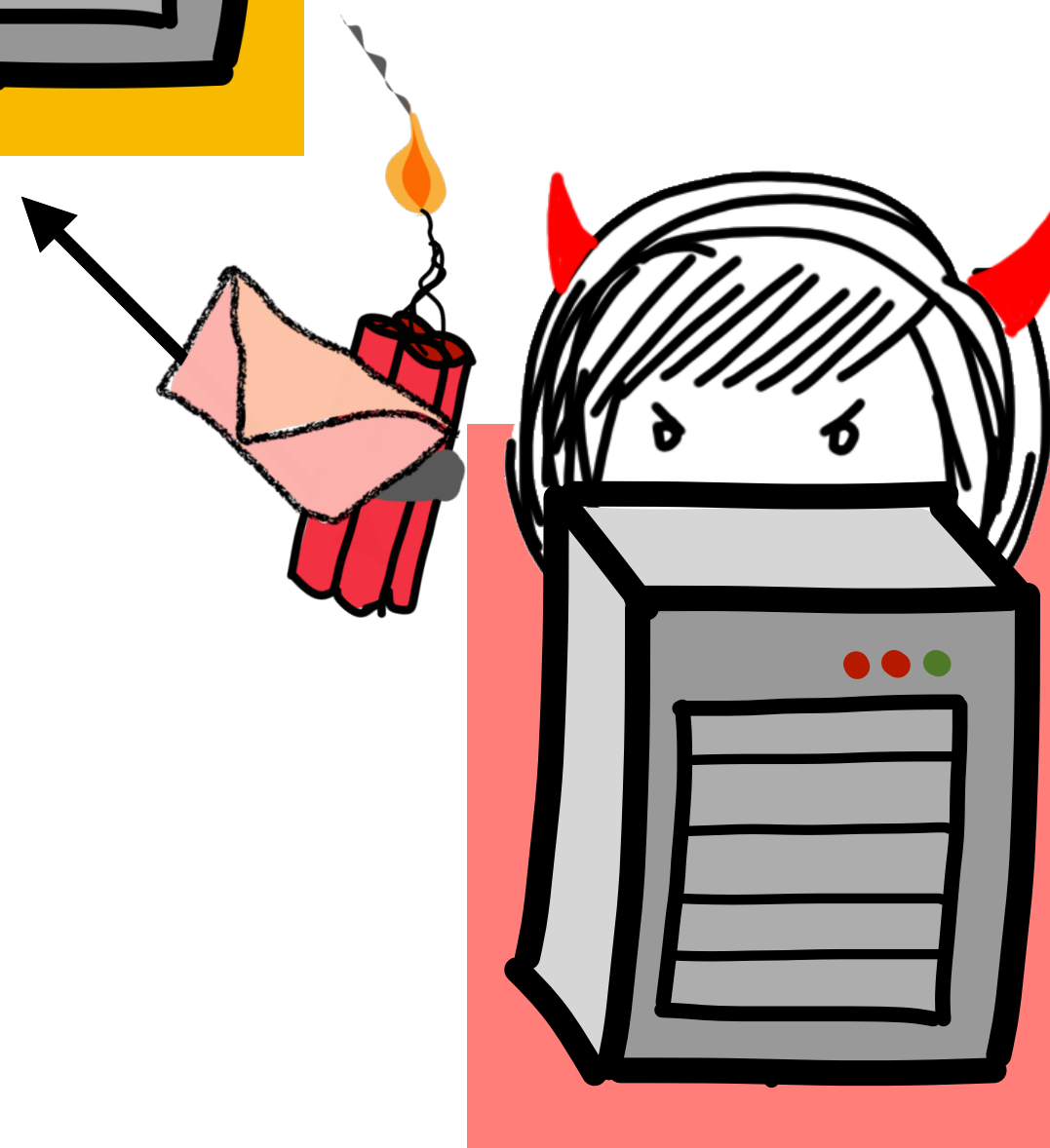
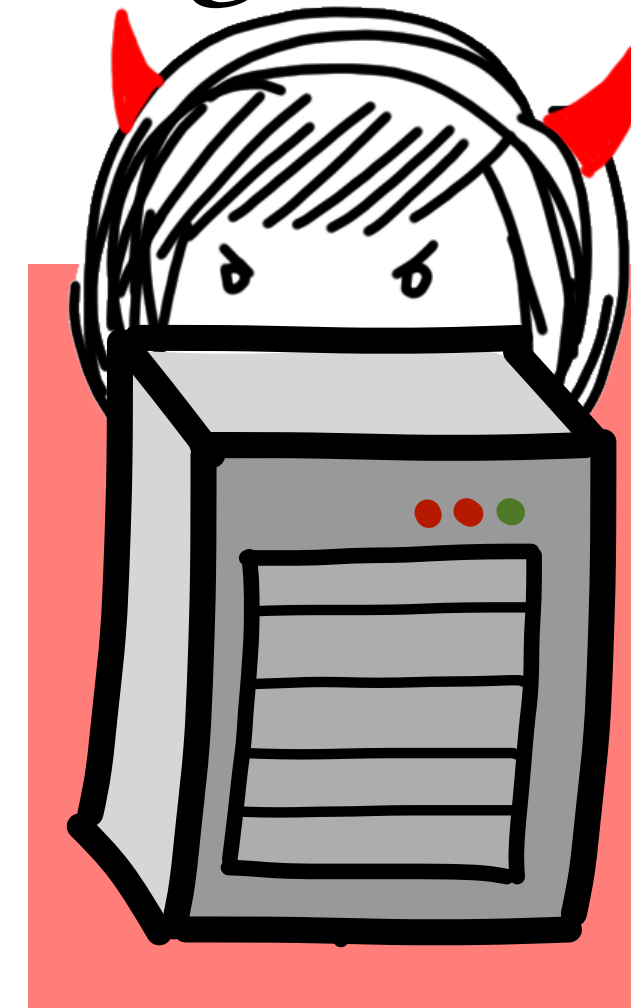
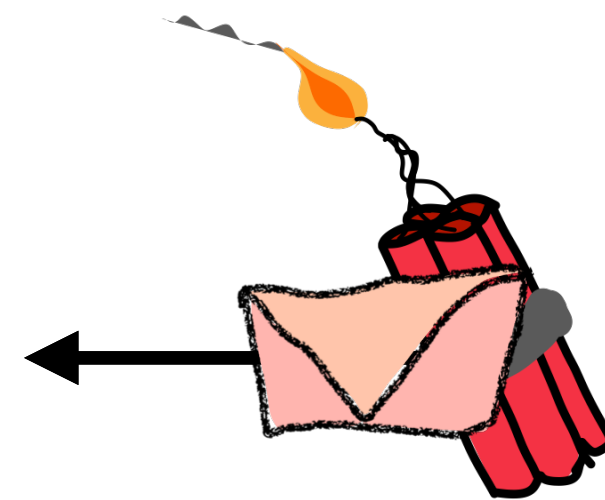
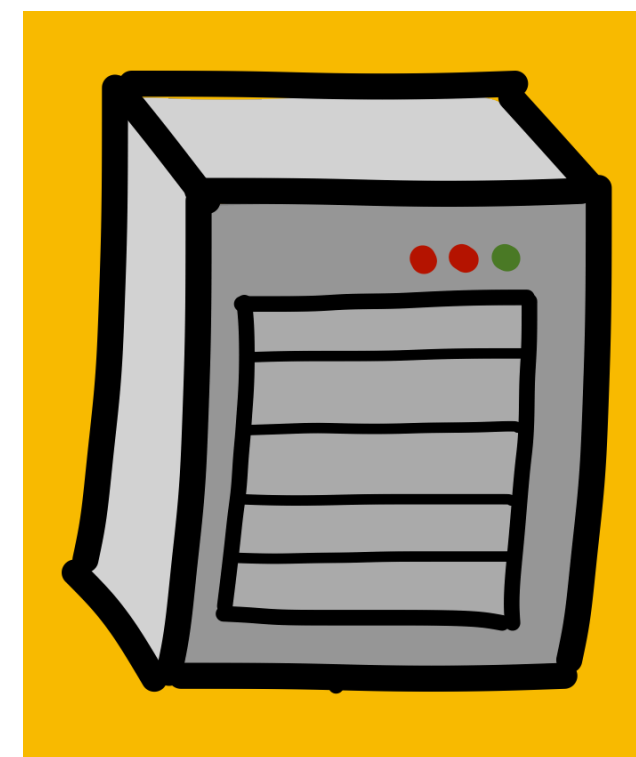
$(3,n)$ Signing

Best Possible Security: Protection against 2 corruptions



still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”



Threshold signing via
“Dishonest majority”
MPC protocol

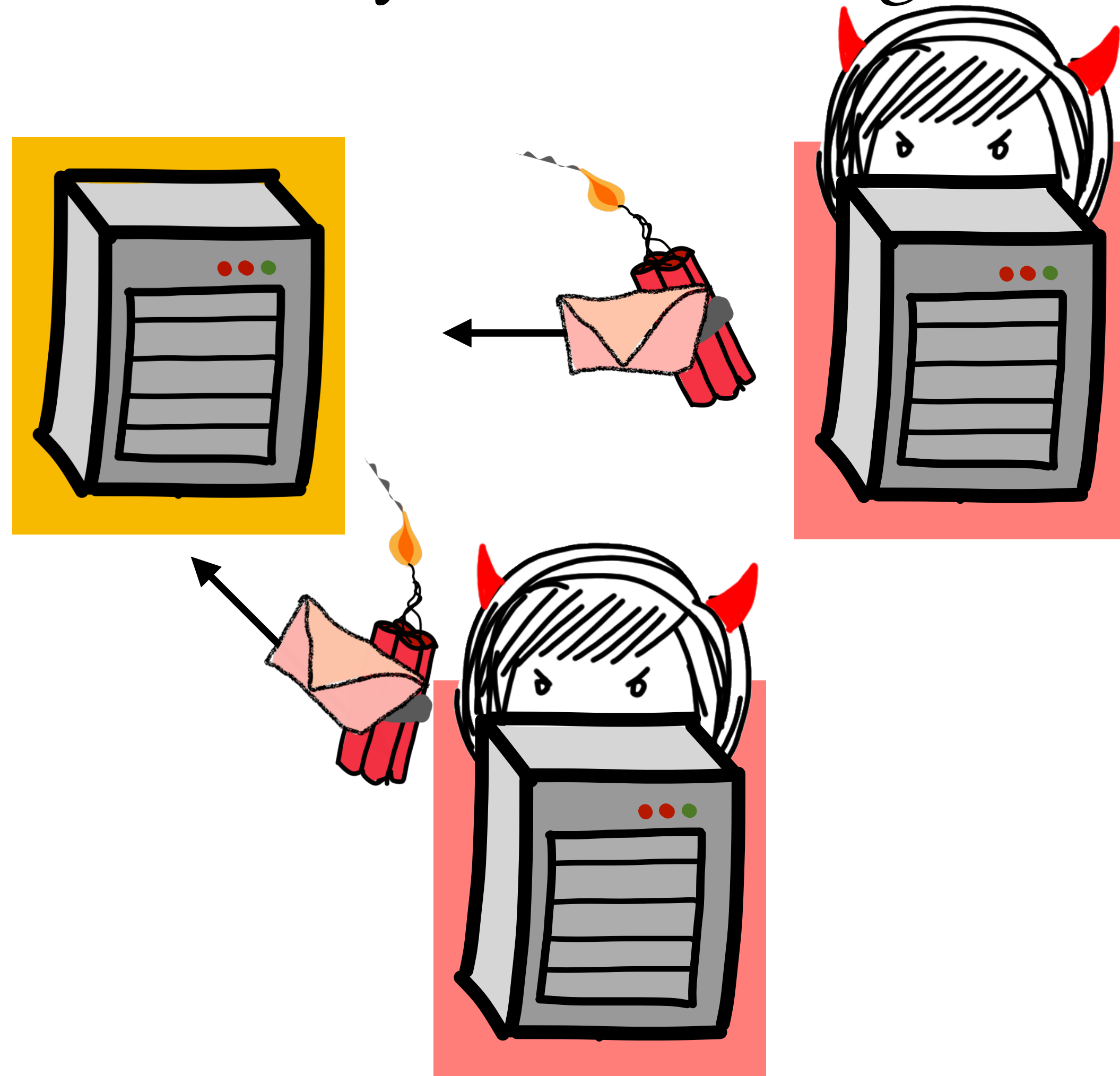
$(3,n)$ Signing

Best Possible Security: Protection against 2 corruptions

 still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



Threshold signing via
“Dishonest majority”
MPC protocol

$(3,n)$ Signing

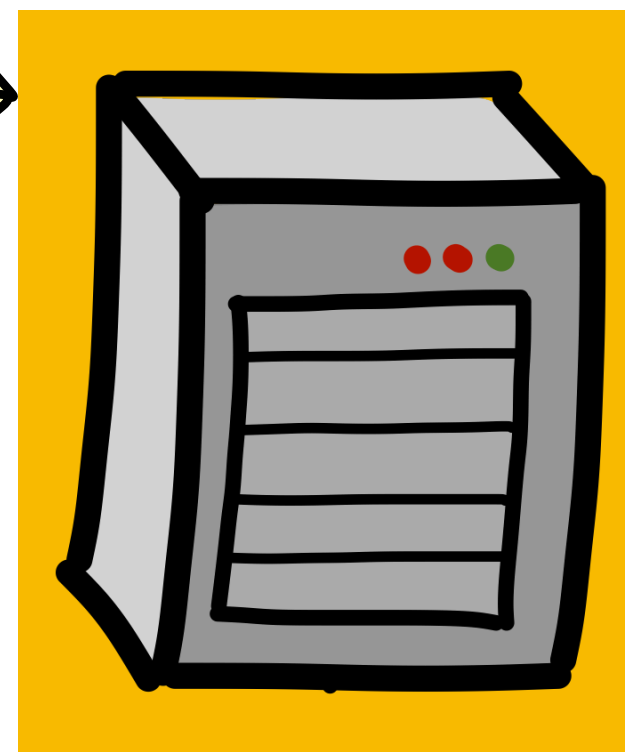
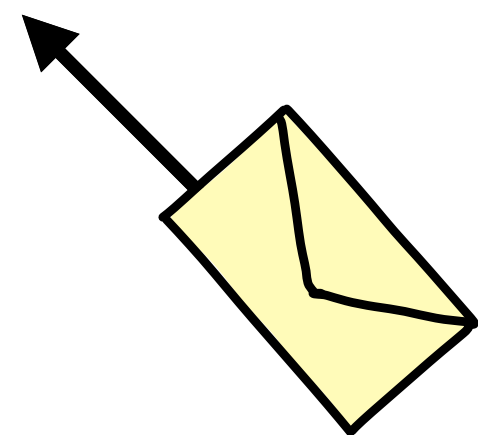
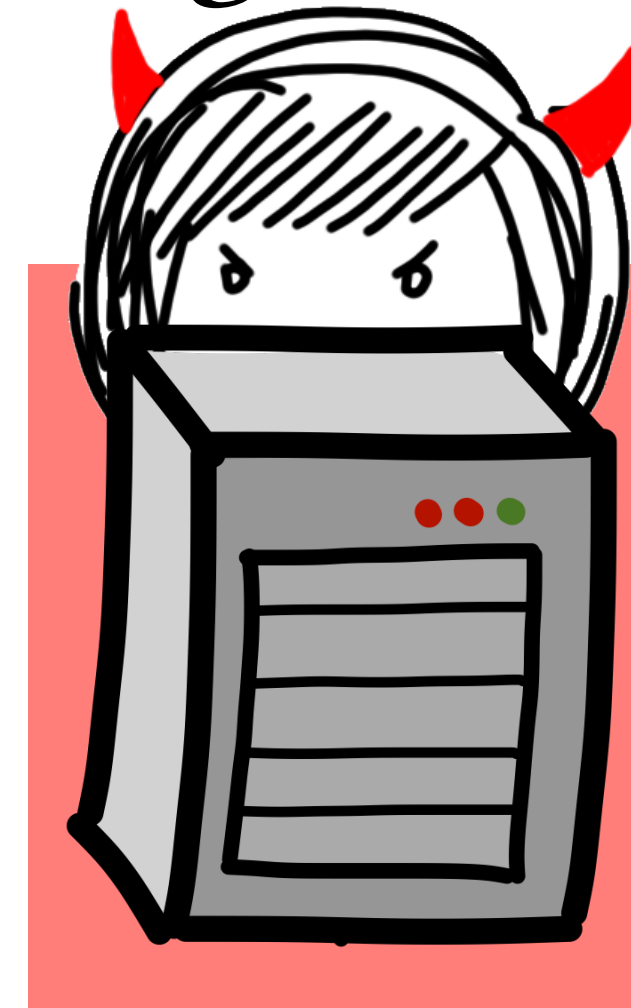
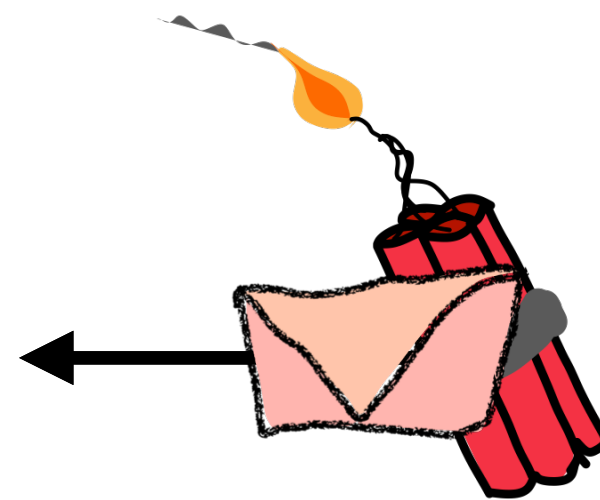
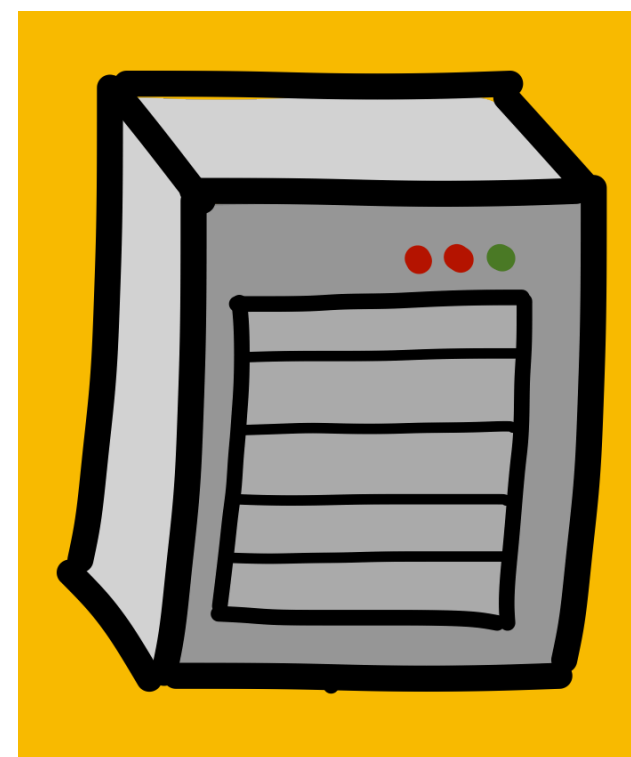
Best Possible Security: Protection against 2 corruptions



still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



$(3,n)$ Signing

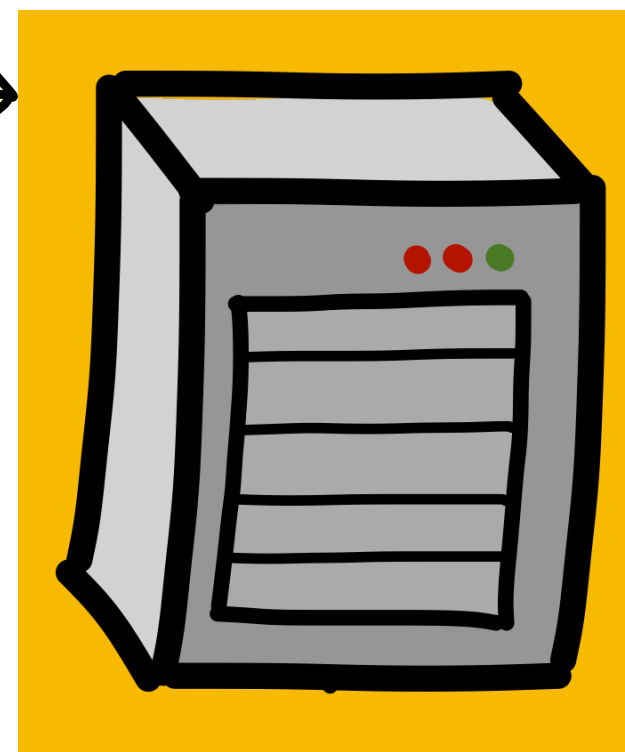
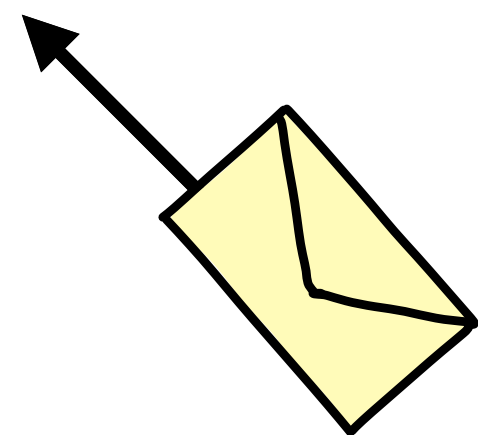
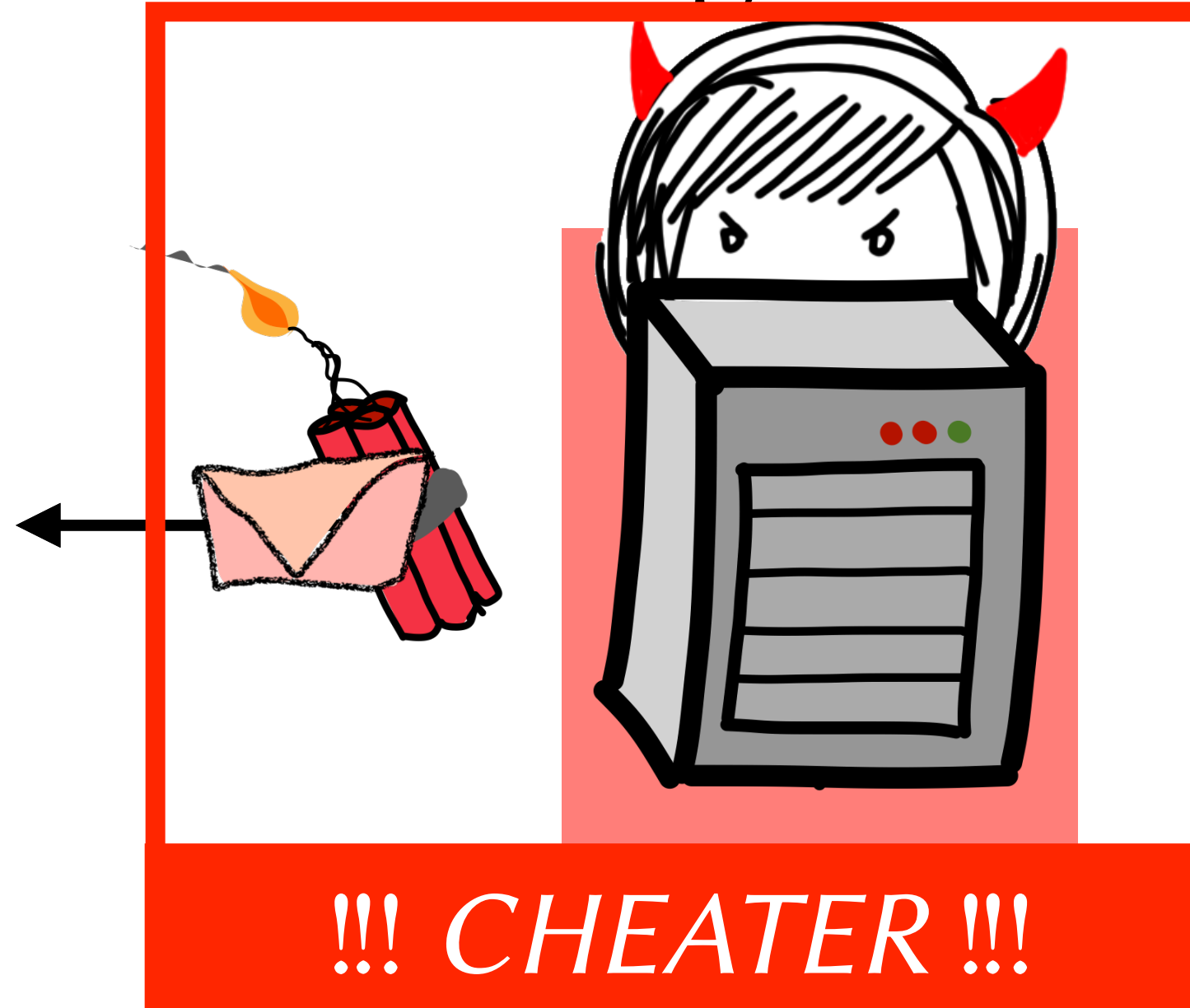
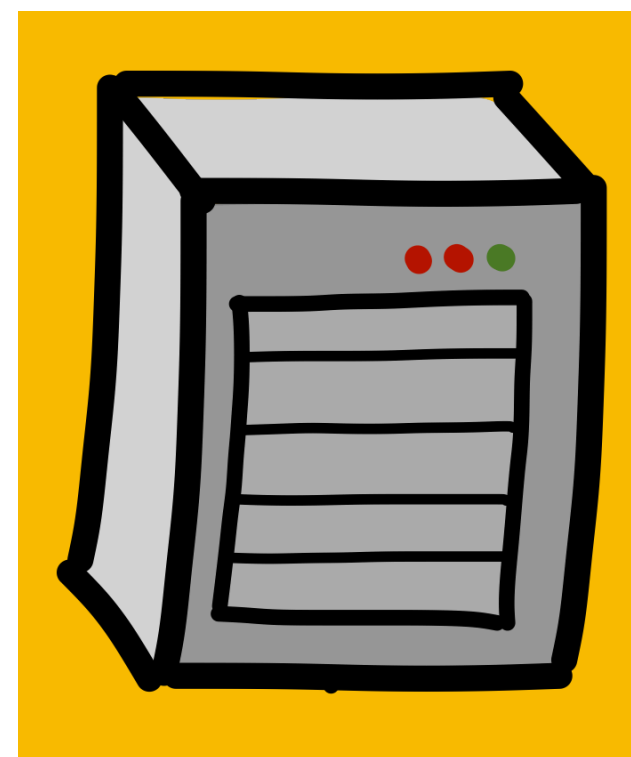
Best Possible Security: Protection against 2 corruptions



still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



$(3,n)$ Signing

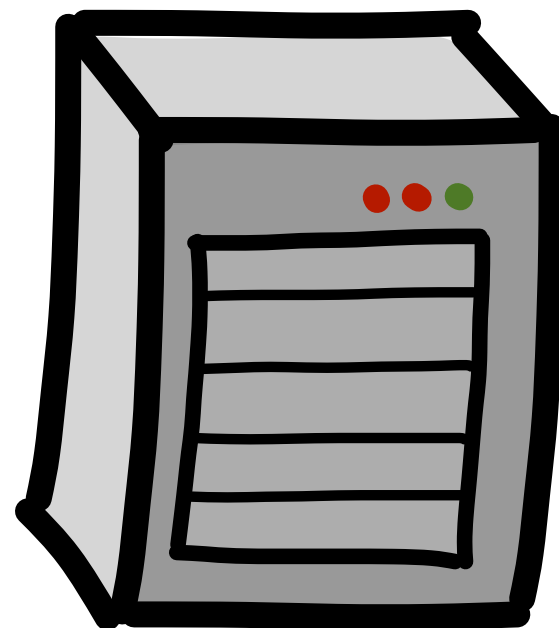
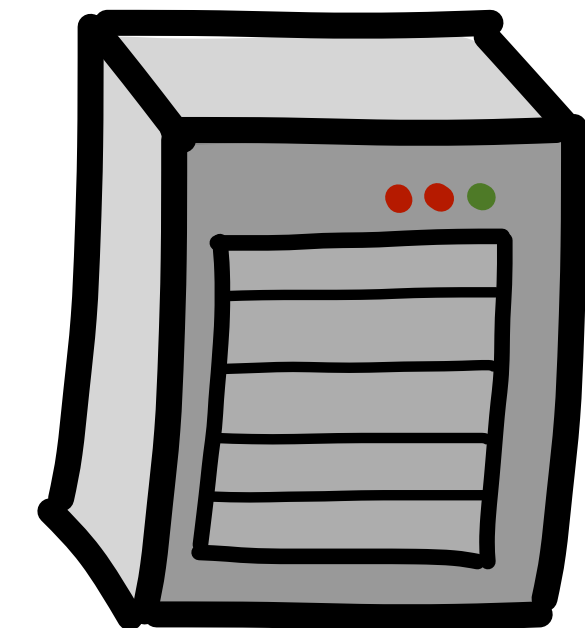
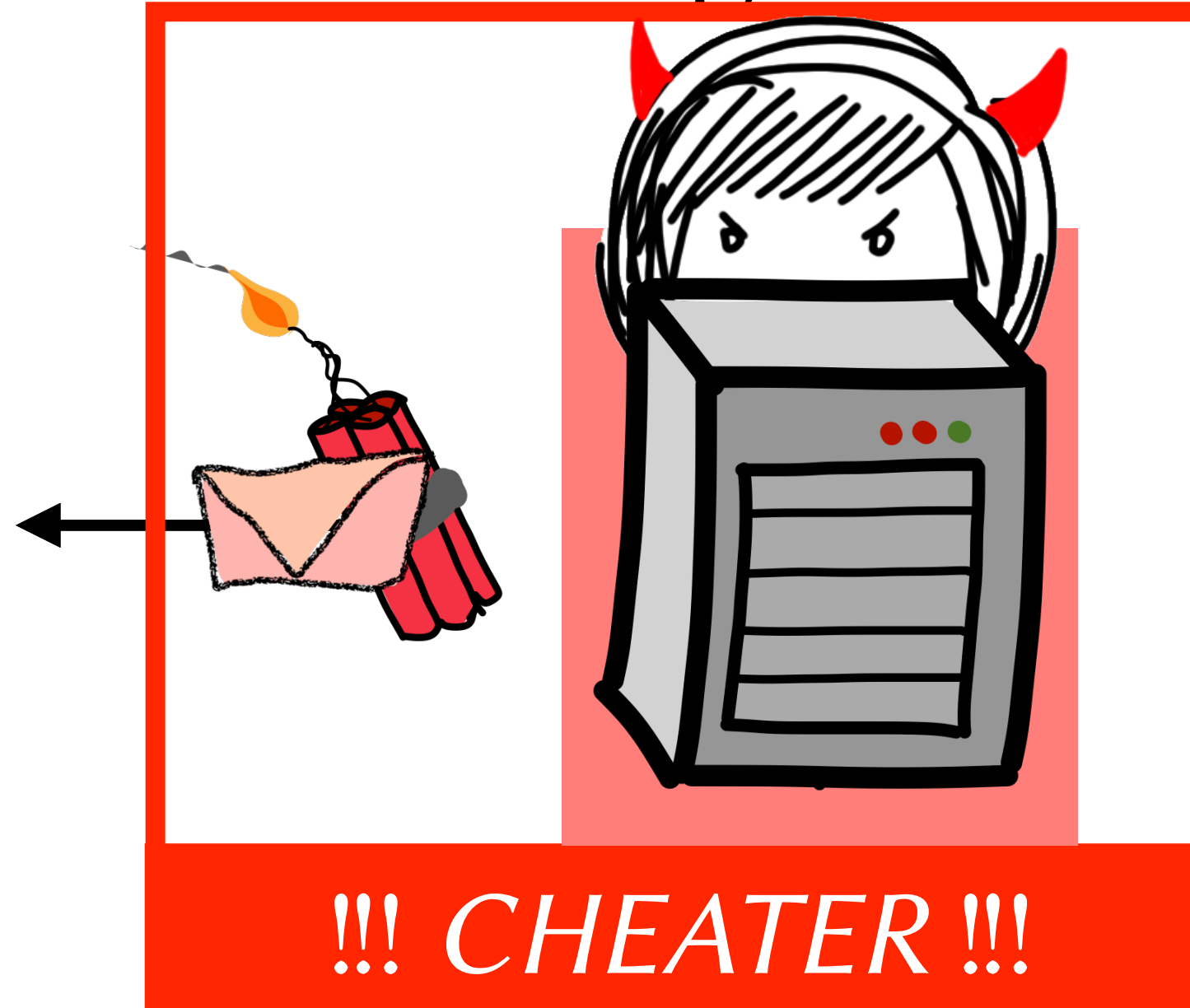
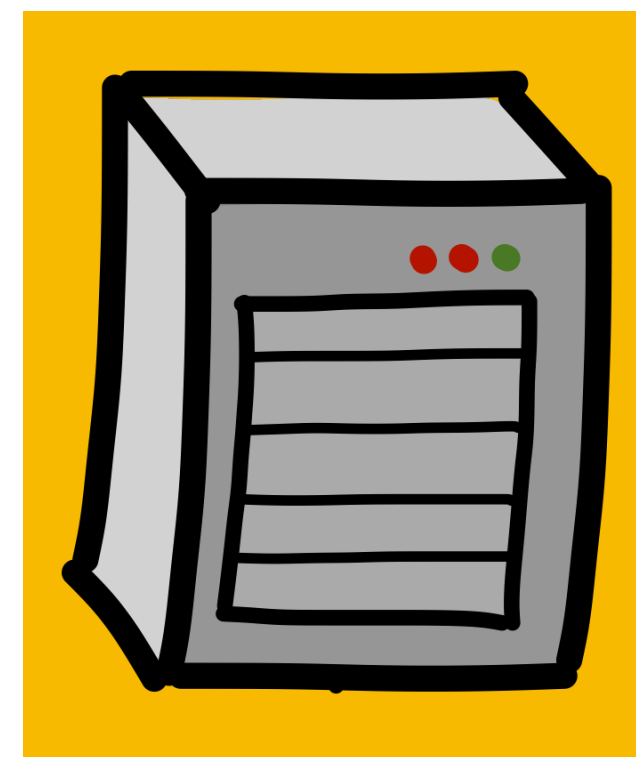
Best Possible Security: Protection against 2 corruptions



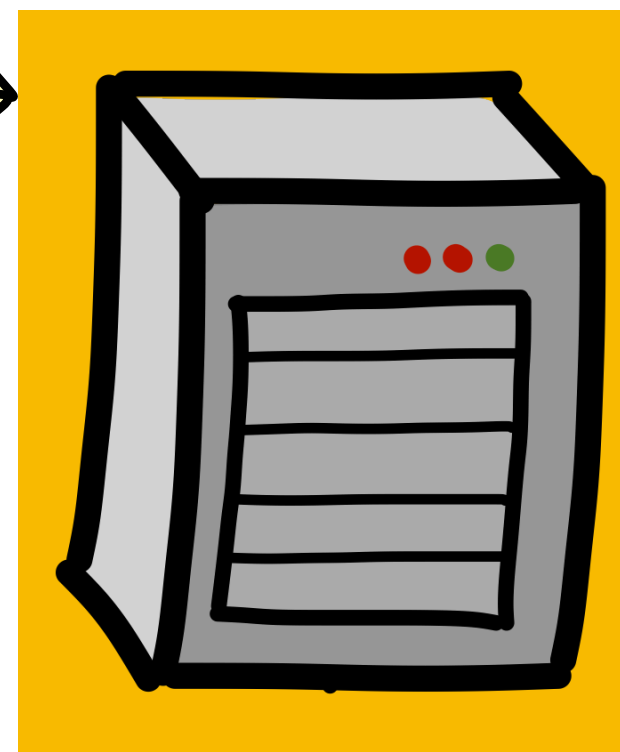
still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



“Global”
honest majority



$(3,n)$ Signing

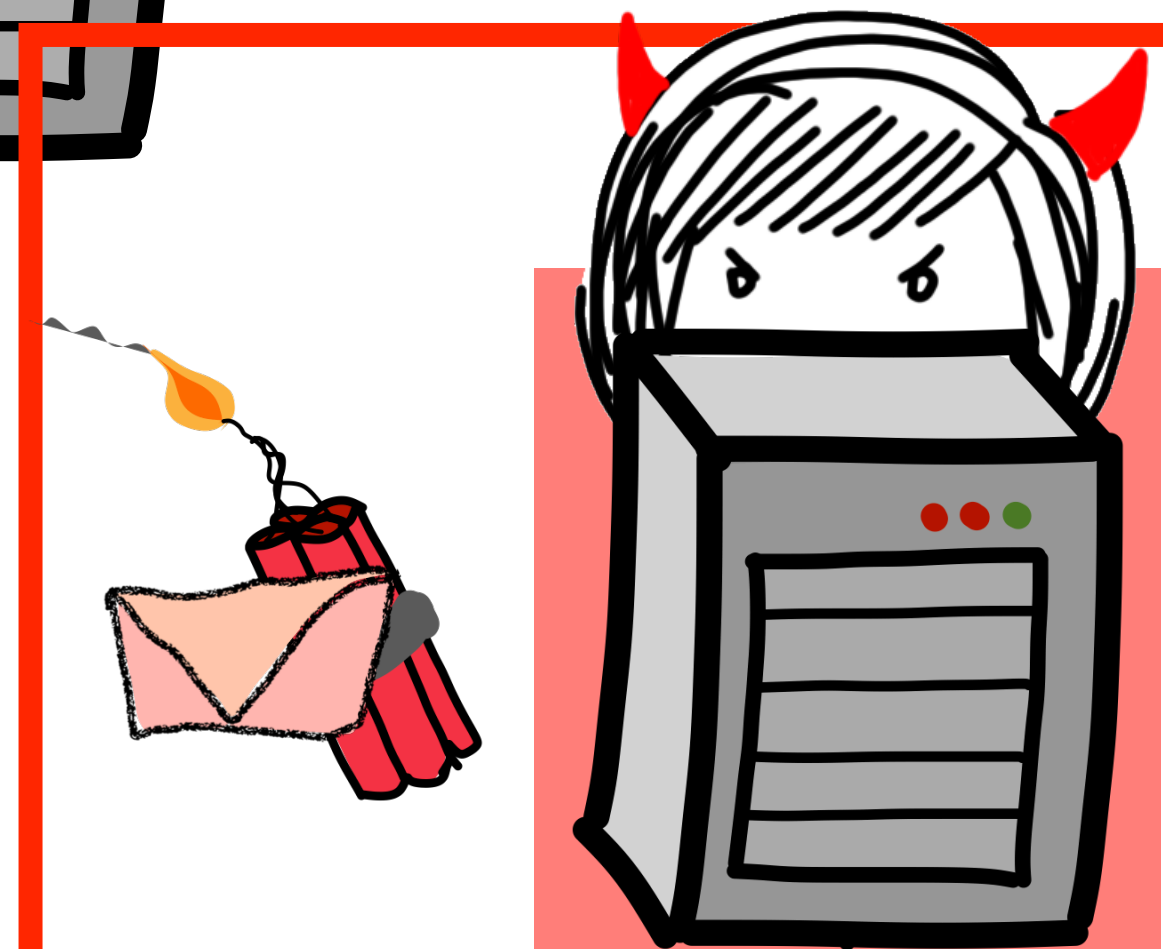
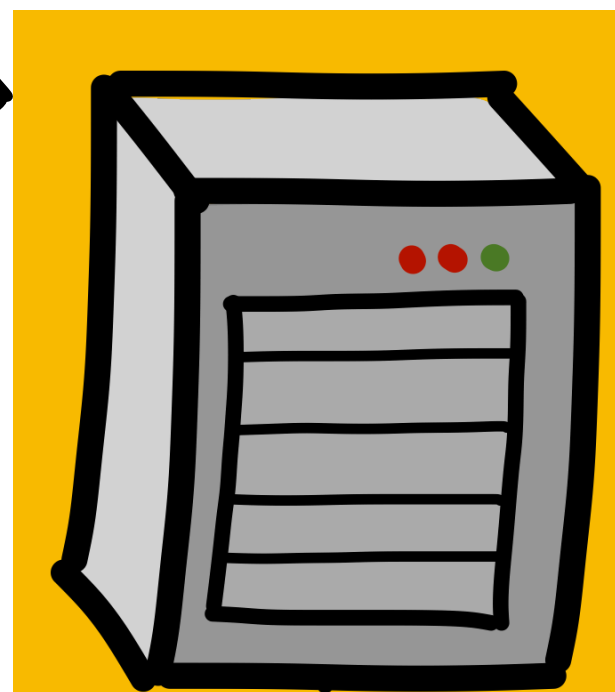
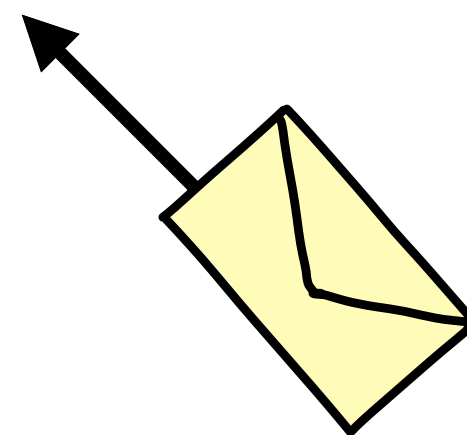
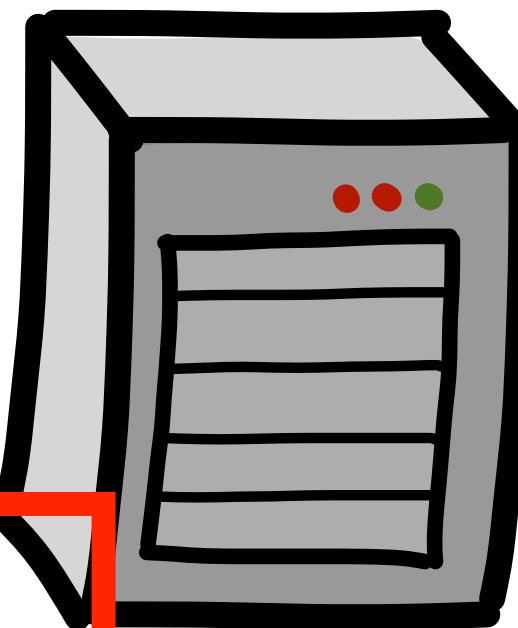
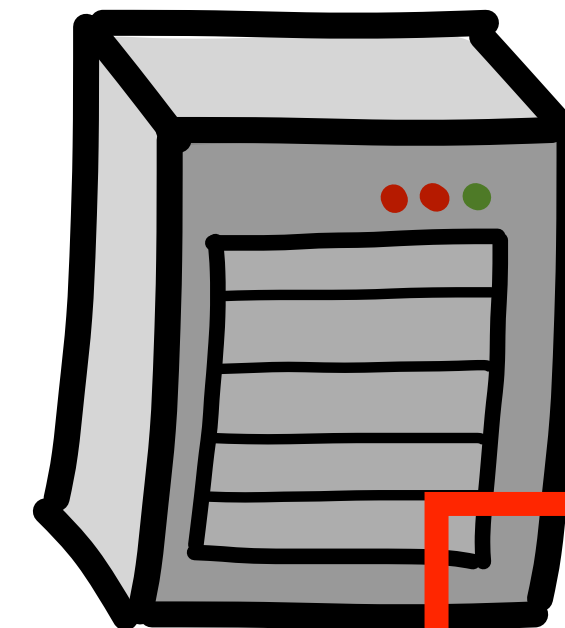
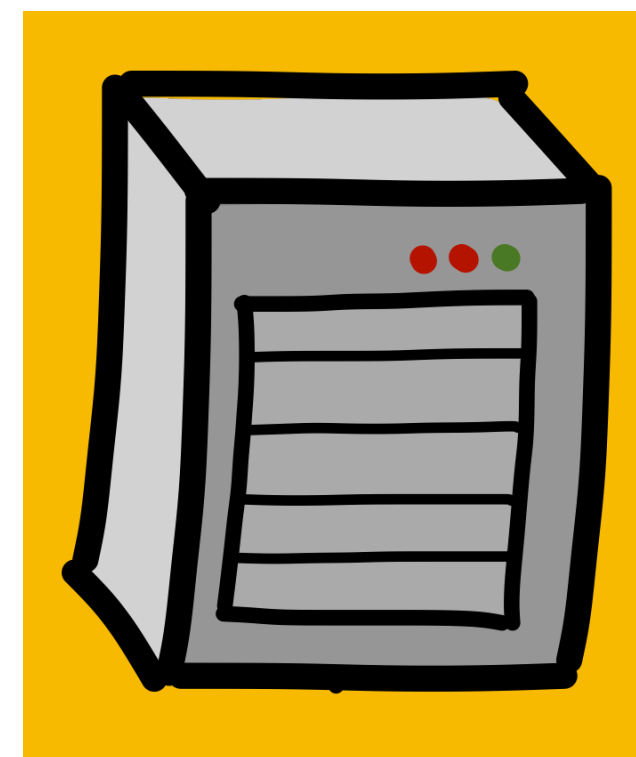
Best Possible Security: Protection against 2 corruptions



still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



!!! CHEATER !!!

$(3,n)$ Signing

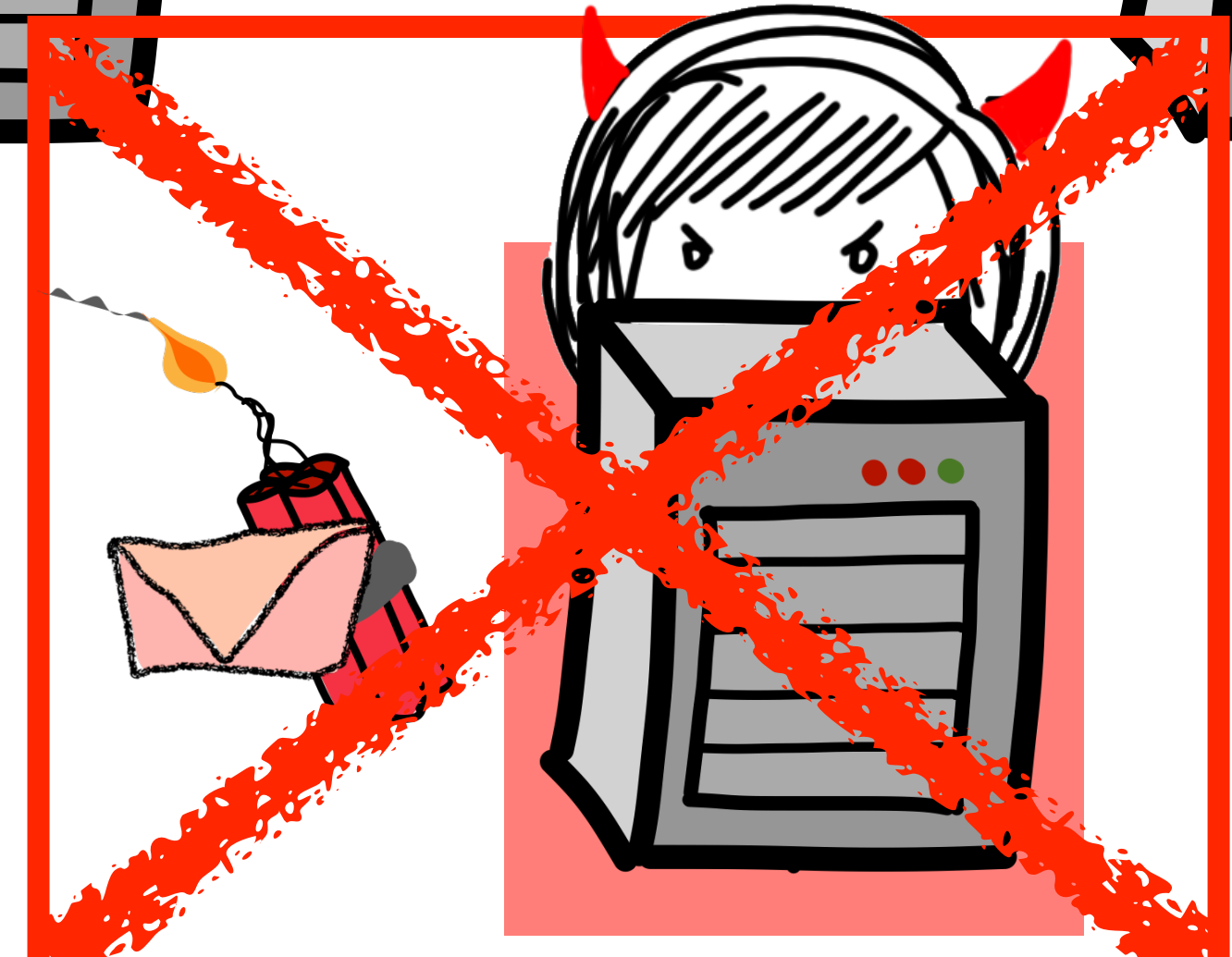
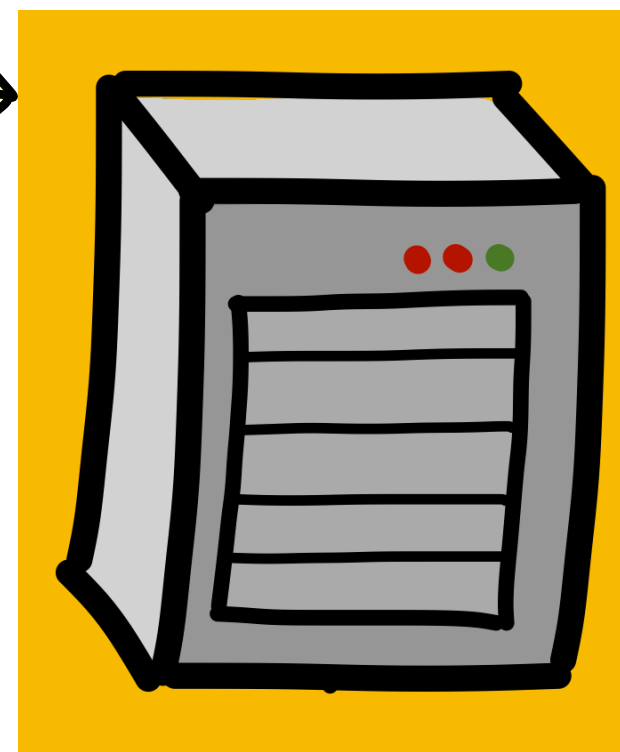
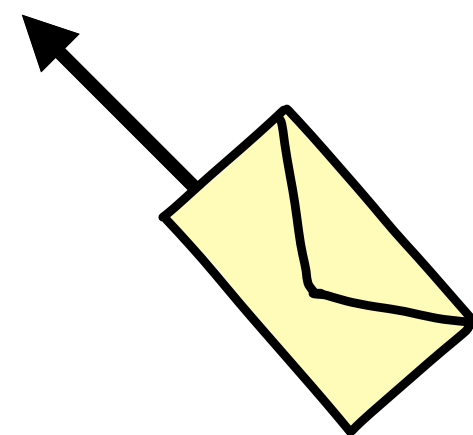
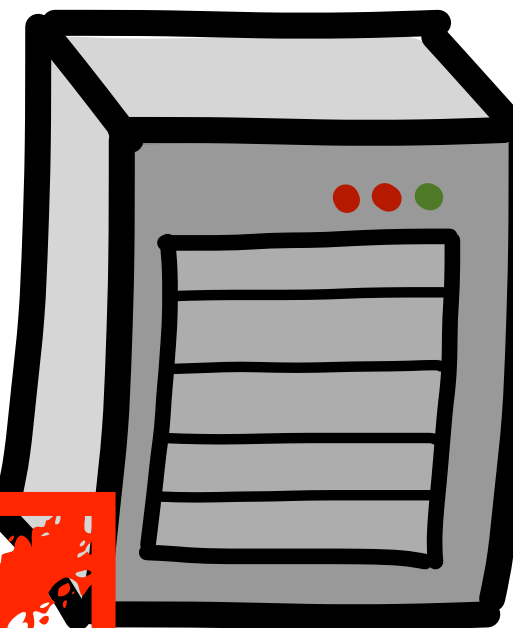
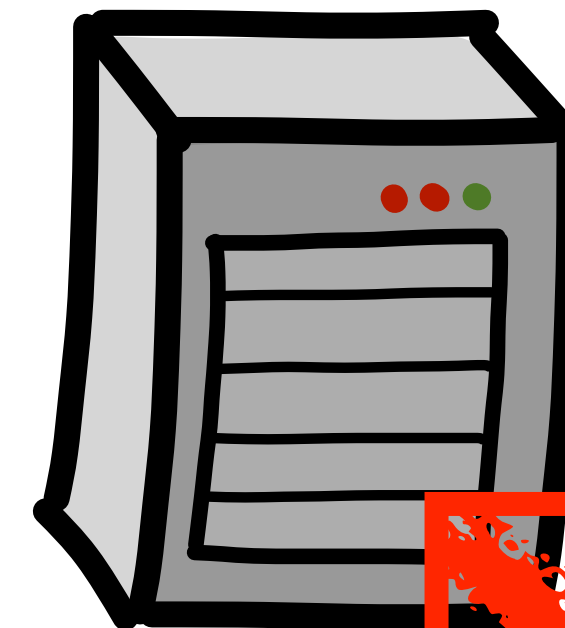
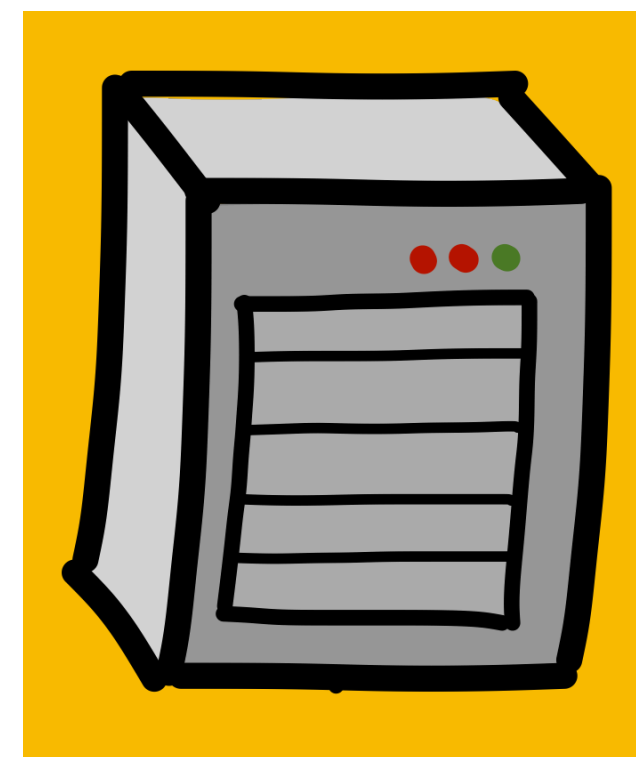
Best Possible Security: Protection against 2 corruptions



still safe!

But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



!!! CHEATER !!!

$(3,n)$ Signing

Best Possible Security: Protection against 2 corruptions



still safe!

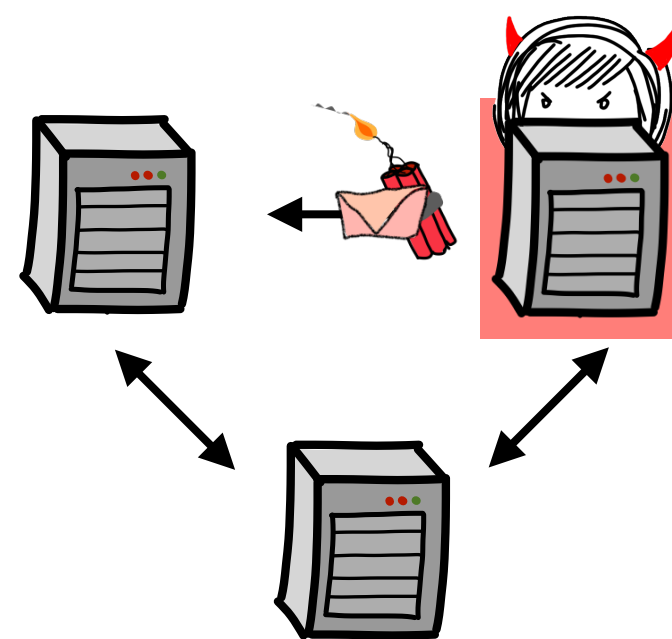
But, MPC fails
→ no sig (DoS)
“security w. abort”

Folklore remedy:
Identifiable Abort



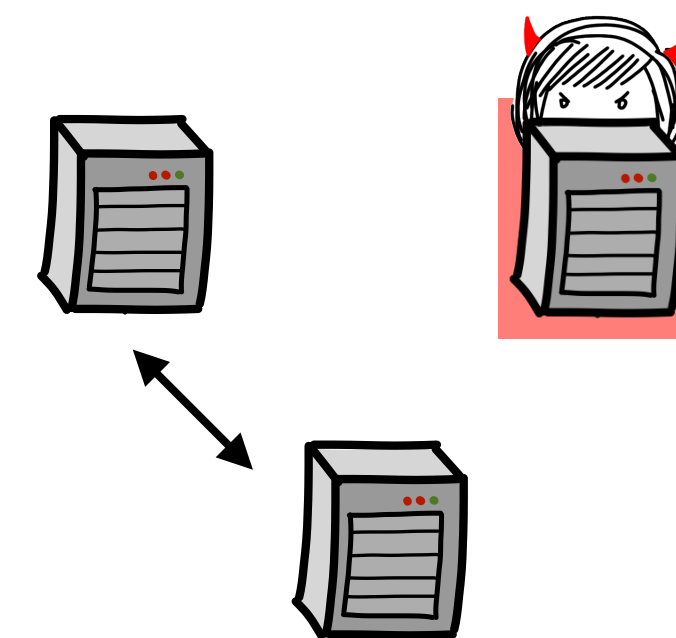
Recipe for Identifiable Abort

- Cheater *could* be found through out of band methods.
- We want **certifiable** protocol mechanism to identify who crashed the protocol
⇒ each party either gets output, or identity of cheating party + cert. of cheat
- Two ways to crash protocol:



1. Malformed protocol message


⋮



2. No message at all

Anatomy of MPC-ECDSA w. IA

Anatomy of MPC-ECDSA w. IA



Baseline security-with-abort protocol

Anatomy of MPC-ECDSA w. IA

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

Anatomy of MPC-ECDSA w. IA

Standard
mitigation:
ZK proofs

[GMW87],
[CGGMP20]

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

Anatomy of MPC-ECDSA w. IA

Standard
mitigation:
ZK proofs

[GMW87],
[CGGMP20]

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Existing works:
all messages
over broadcast

Mechanism to guarantee
each party sends *some* message every round

Anatomy of MPC-ECDSA w. IA

Standard
mitigation:
ZK proofs

[GMW87],
[CGGMP20]

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Existing works:
all messages
over broadcast

Mechanism to guarantee
each party sends *some* message every round

external trust assumptions, can be expensive

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

This work: define “Broadcast-IA”

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

This work: define “Broadcast-IA”

1. Impossible w. dishonest majority
2. Simple honest-majority protocol

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

Mechanism to guarantee
each party sends *some* message every round

Simple honest-majority protocol
[DKLs23]

This work: define “Broadcast-IA”

1. Impossible w. dishonest majority
2. Simple honest-majority protocol

Anatomy of MPC-ECDSA w. IA

Mechanism to guarantee
wellformedness of every sent message

Light (Schnorr-like) ZK proofs
+ verifiable complaints

Baseline security-with-abort protocol

Simple honest-majority protocol
[DKLs23]

Mechanism to guarantee
each party sends *some* message every round

This work: define “Broadcast-IA”

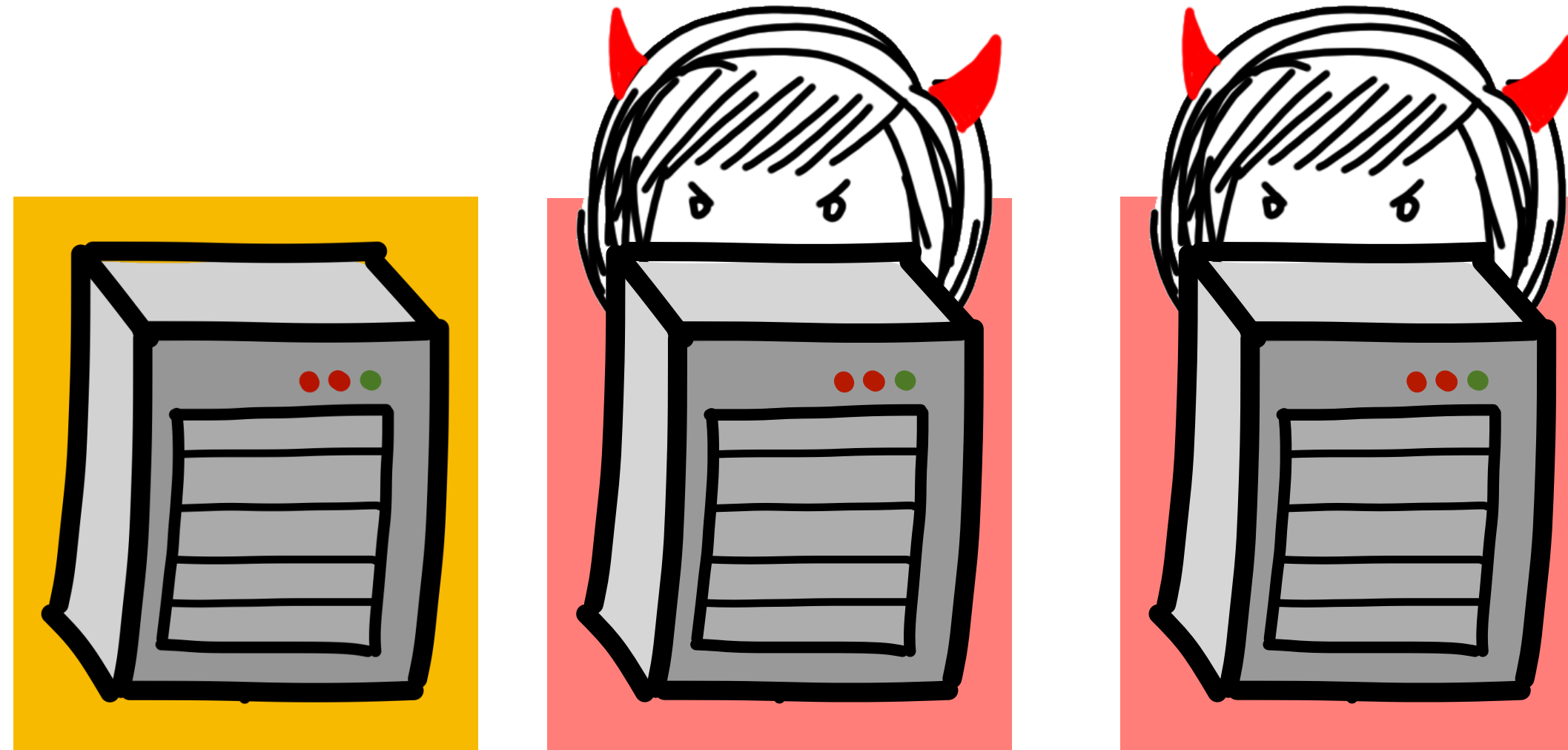
1. Impossible w. dishonest majority
2. Simple honest-majority protocol

Broadcast and (Identifiable) Abort

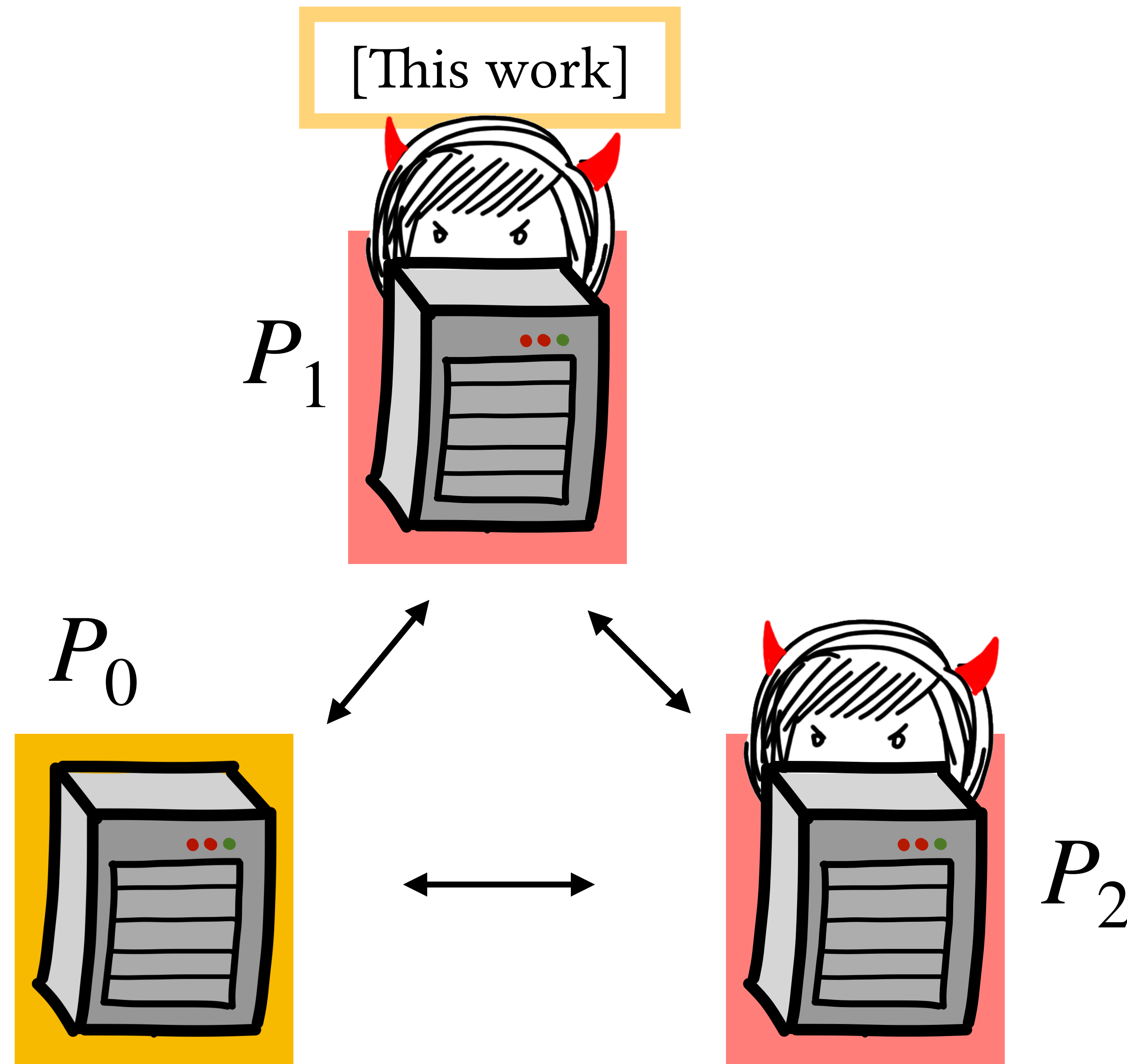
- Basic broadcast guarantee, **Consistency**: Malicious sender can't trick honest receivers into accepting conflicting messages m, m^*
- In the security with abort setting, consistency is trivial via simple echoing [GL05]
- In our IA setting, if the sender cheats, each honest party obtains a certificate:
 - (An attempt to) violate consistency, yields a **certificate of cheating** Ω
 - If the sender sends nothing, yields a **certificate of non-responsiveness** ω
- Ω vs. ω : Definite corruption vs. potential network fault—different penalties

Broadcast-IA is Impossible with Dishonest Majority

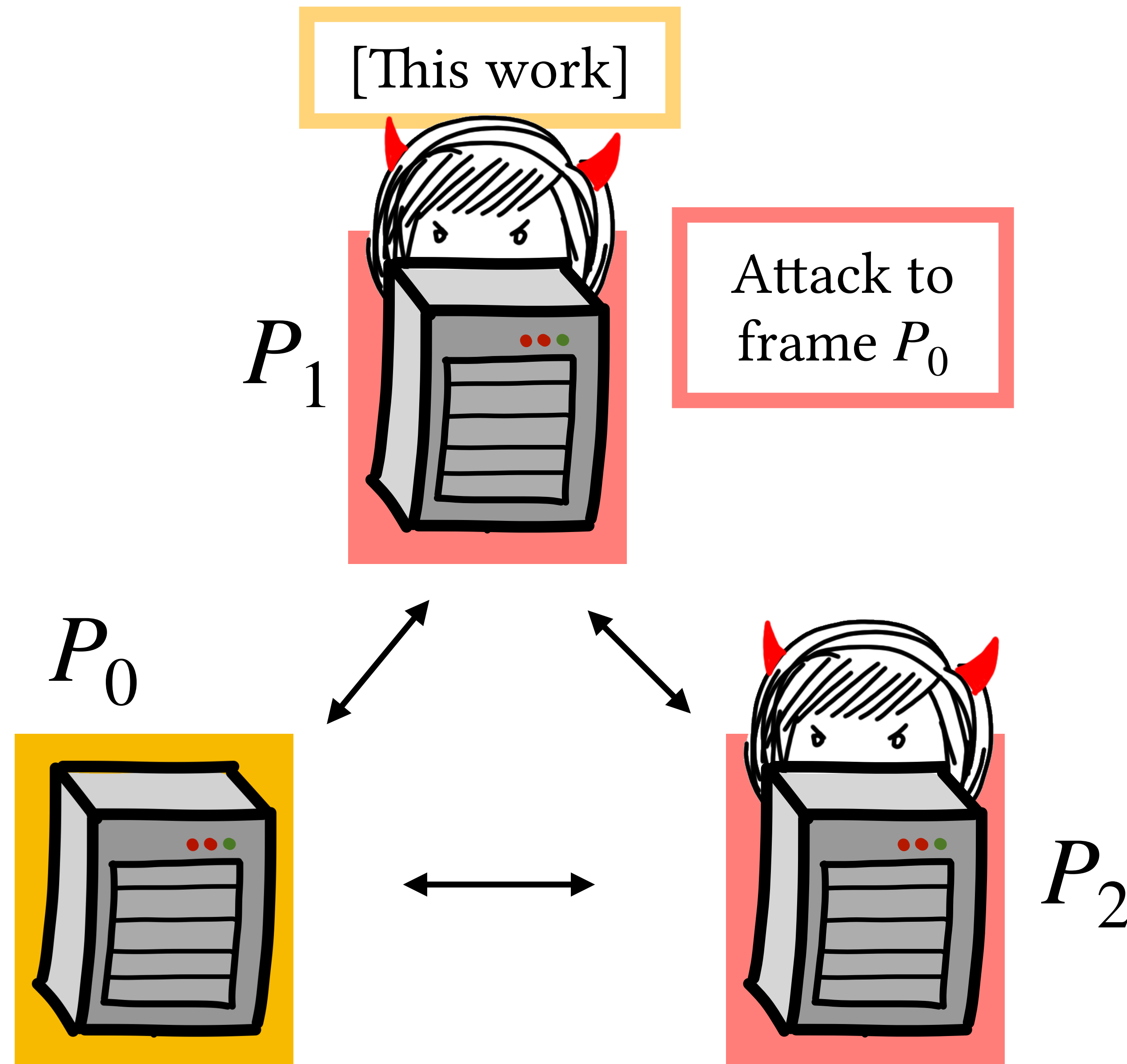
[This work]



Broadcast-IA is Impossible with Dishonest Majority

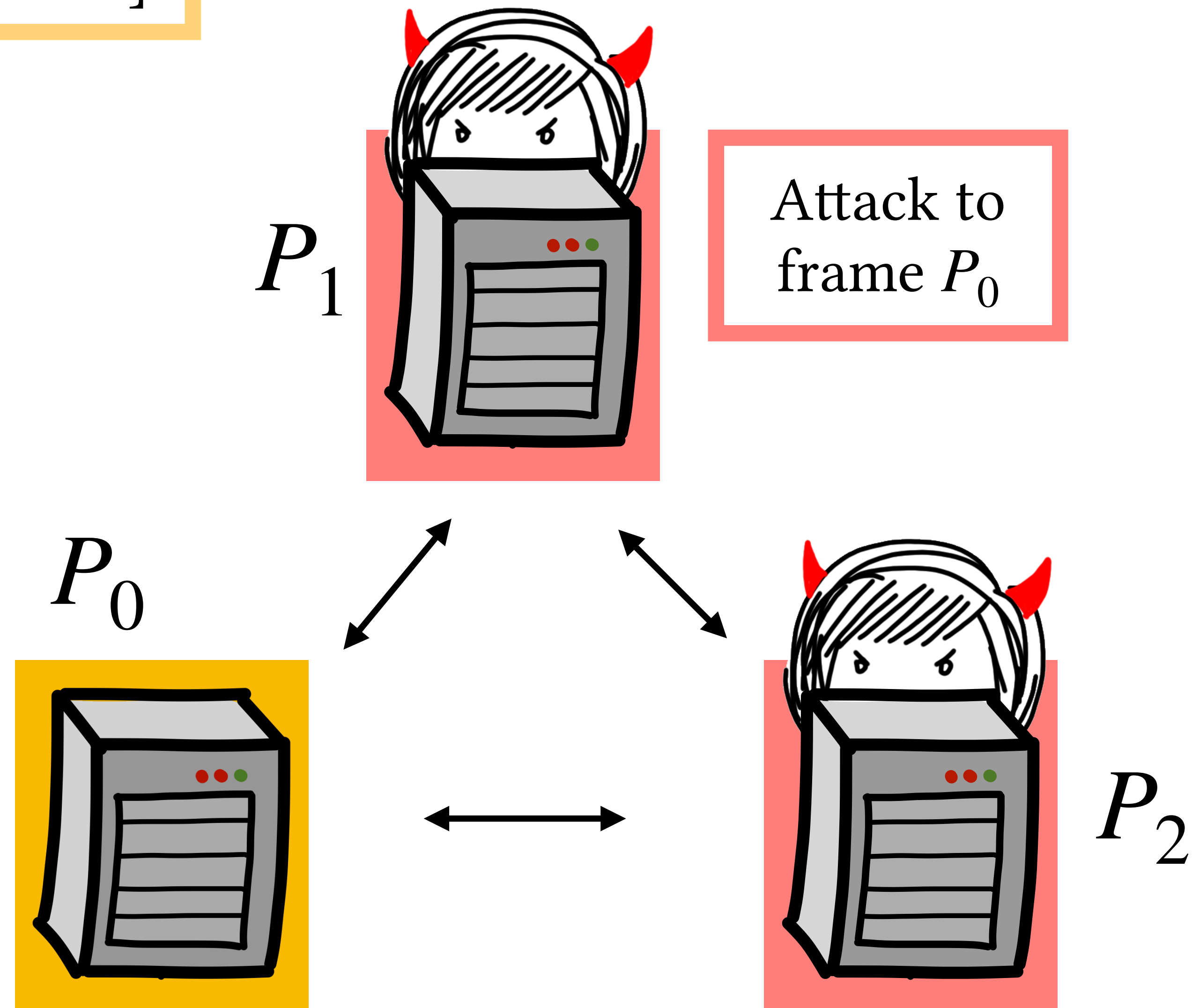


Broadcast-IA is Impossible with Dishonest Majority



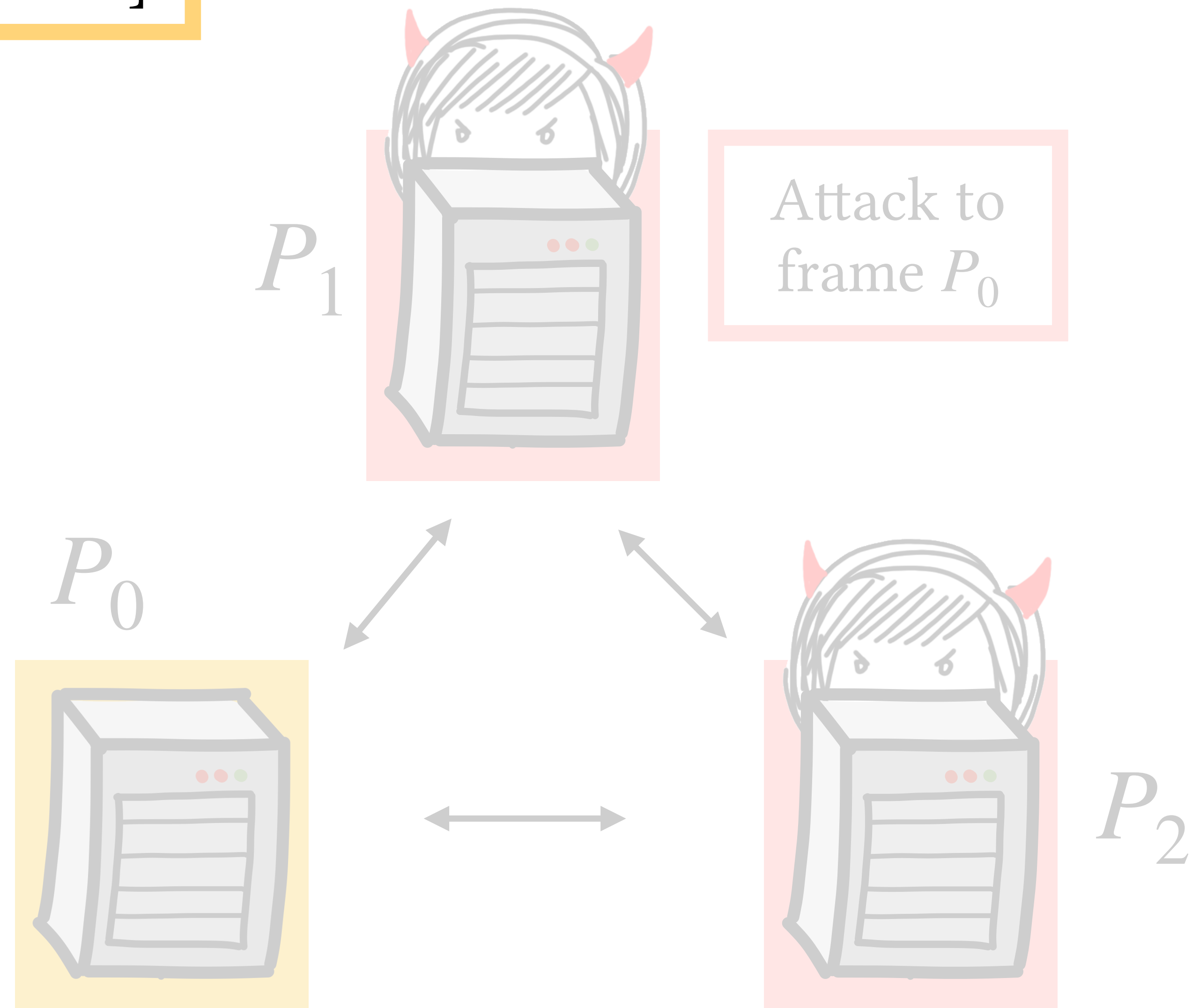
Broadcast-IA is Impossible with Dishonest Majority

[This work]



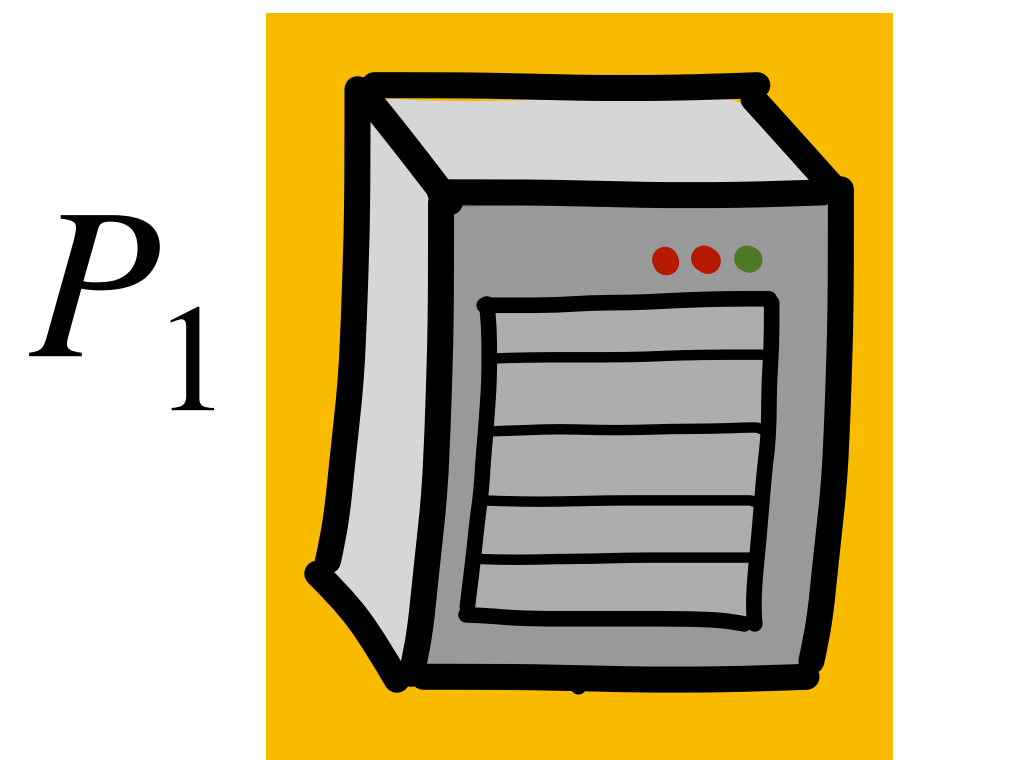
Broadcast-IA is Impossible with Dishonest Majority

[This work]

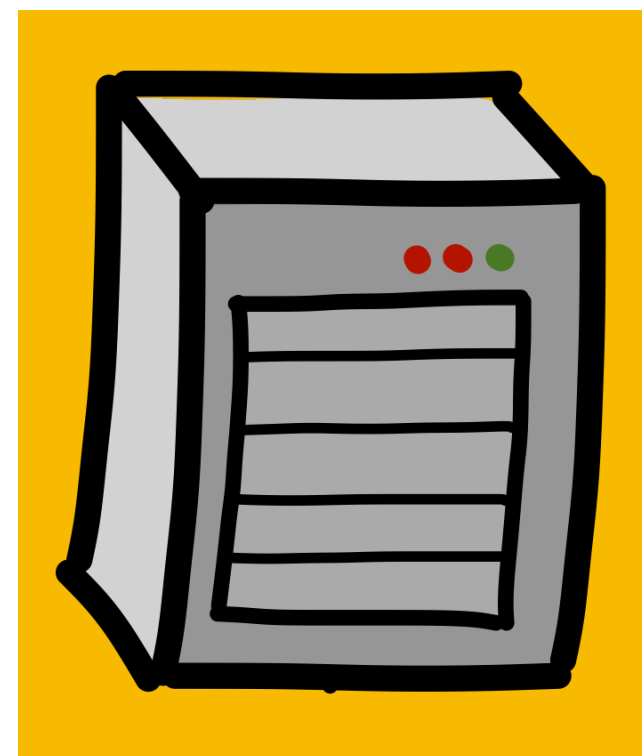


Broadcast-IA is Impossible with Dishonest Majority

[This work]



P_0



P_2



Attack to
frame P_0

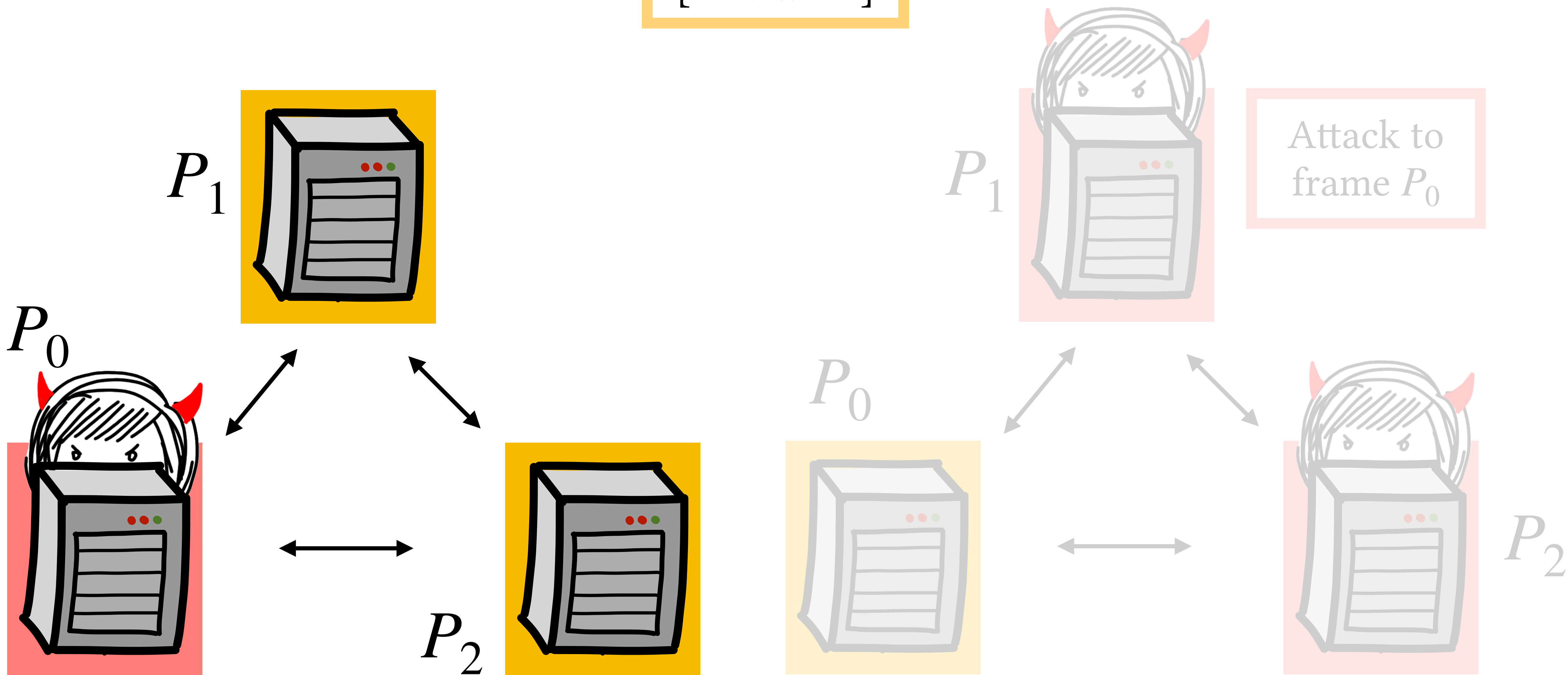
P_0



P_2

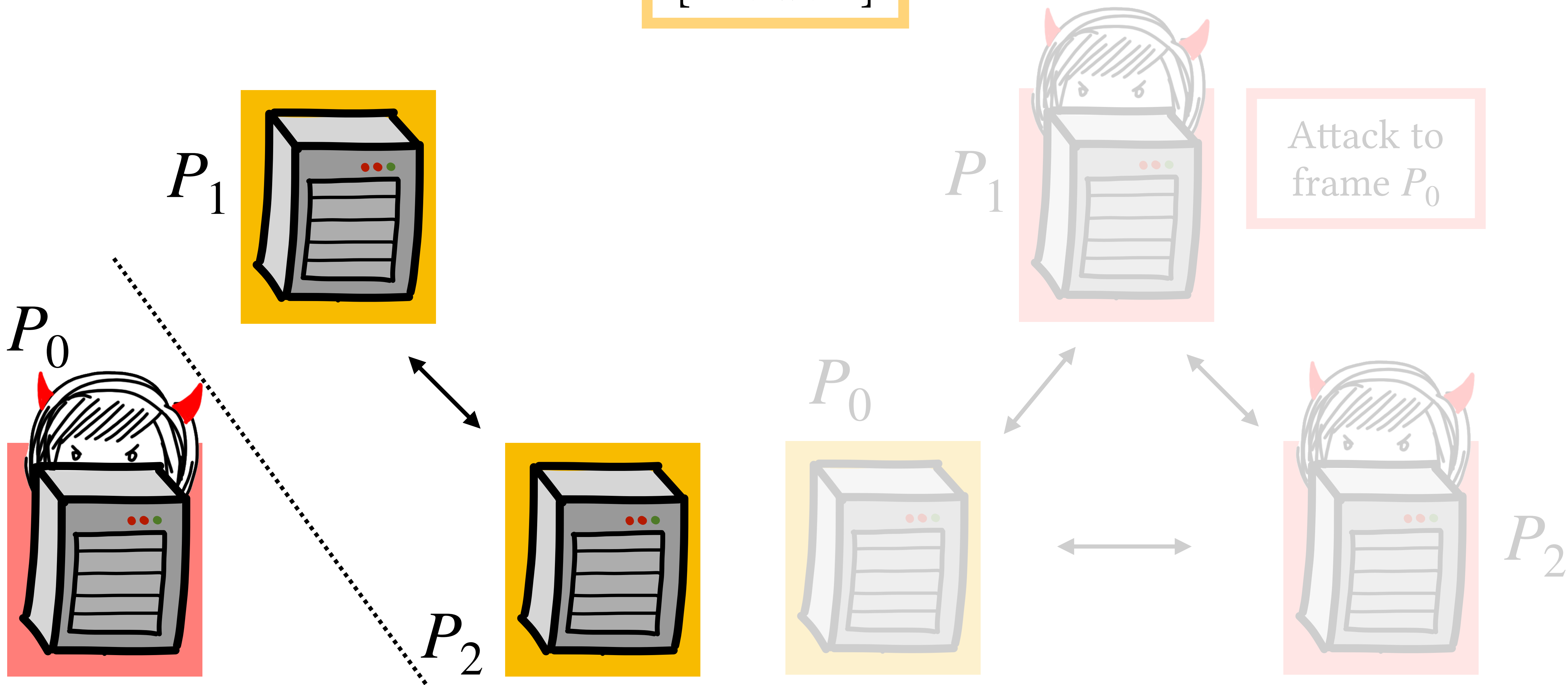
Broadcast-IA is Impossible with Dishonest Majority

[This work]



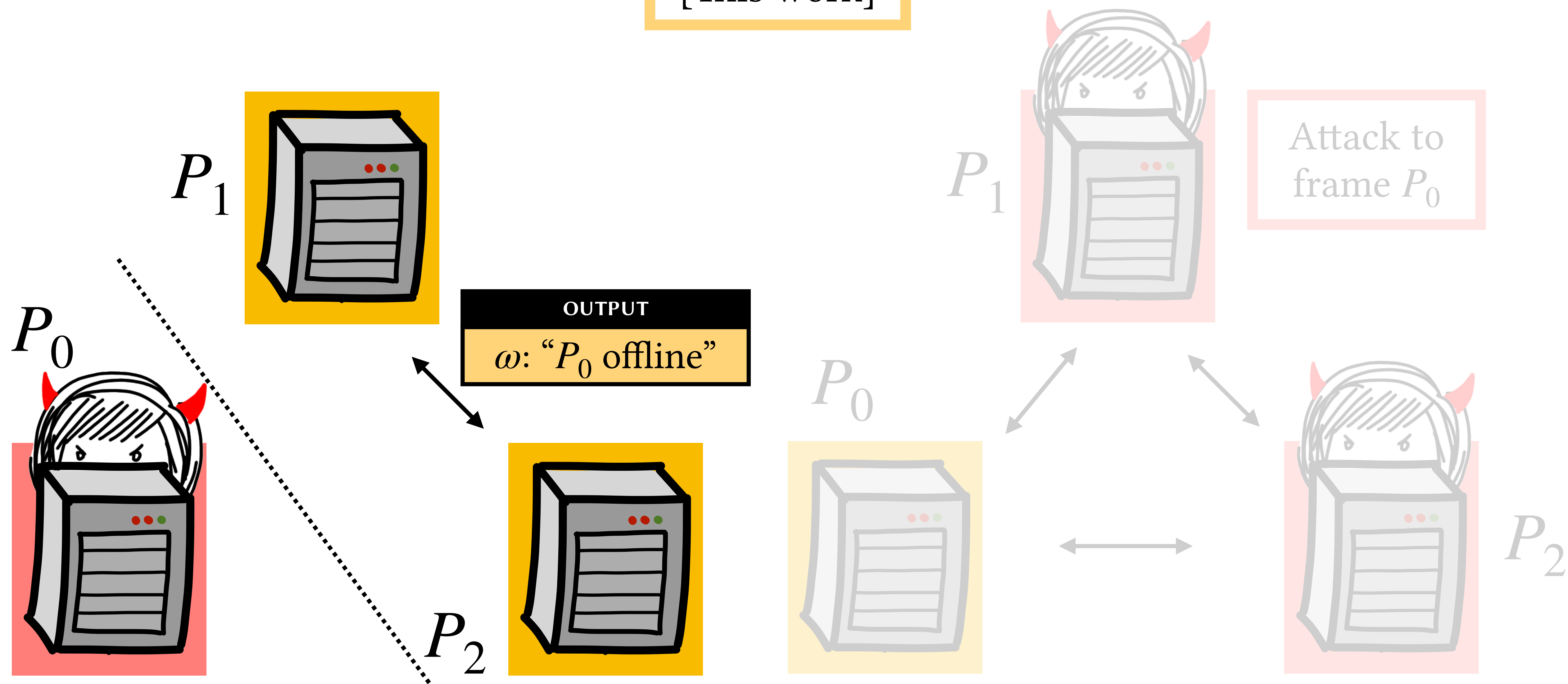
Broadcast-IA is Impossible with Dishonest Majority

[This work]



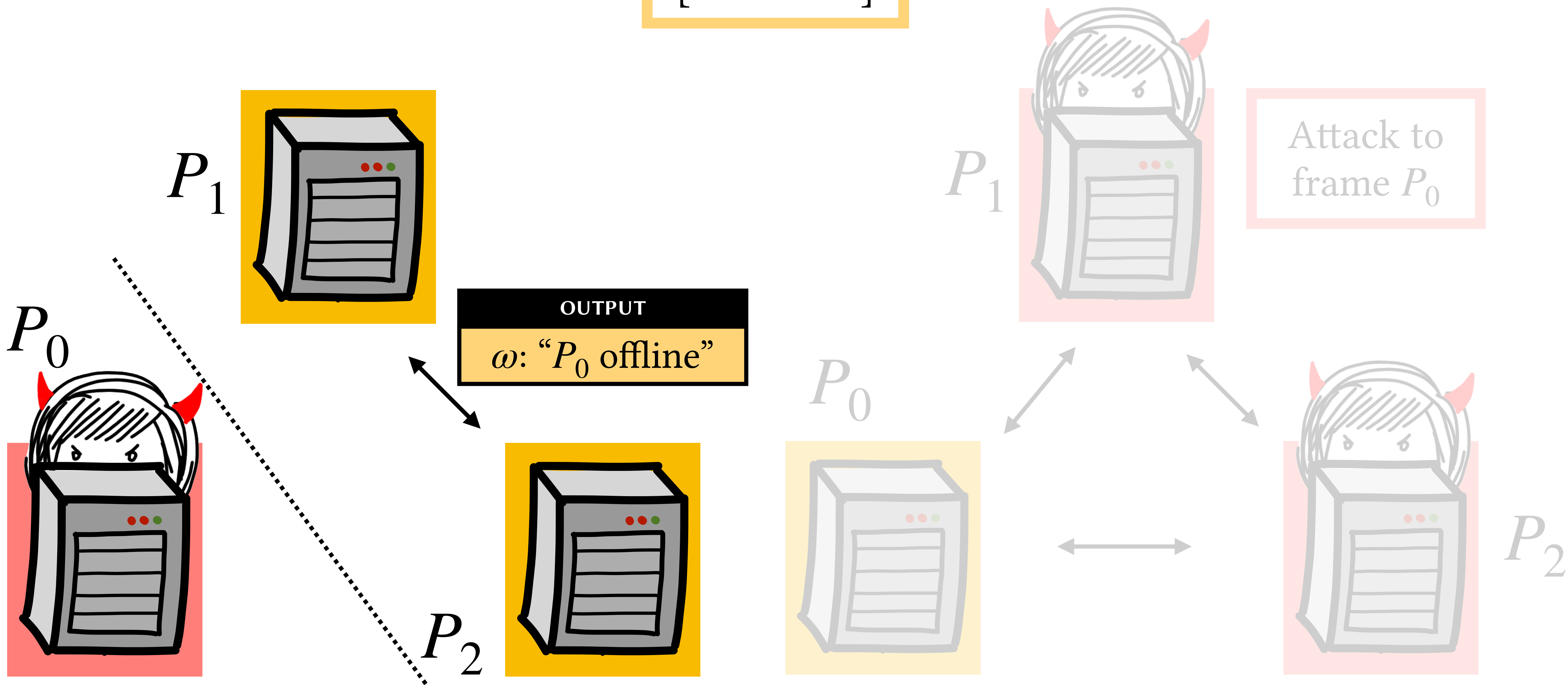
Broadcast-IA is Impossible with Dishonest Majority

[This work]



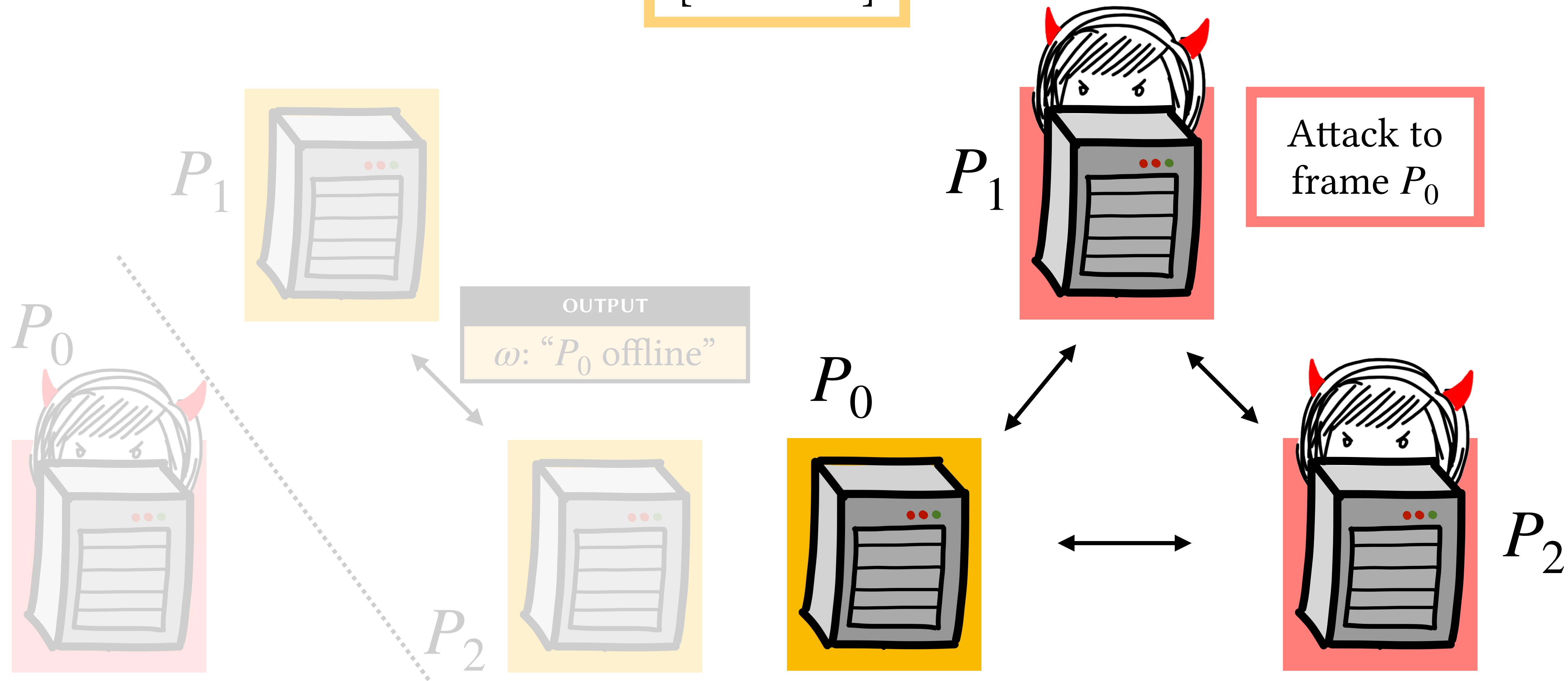
Broadcast-IA is Impossible with Dishonest Majority

[This work]



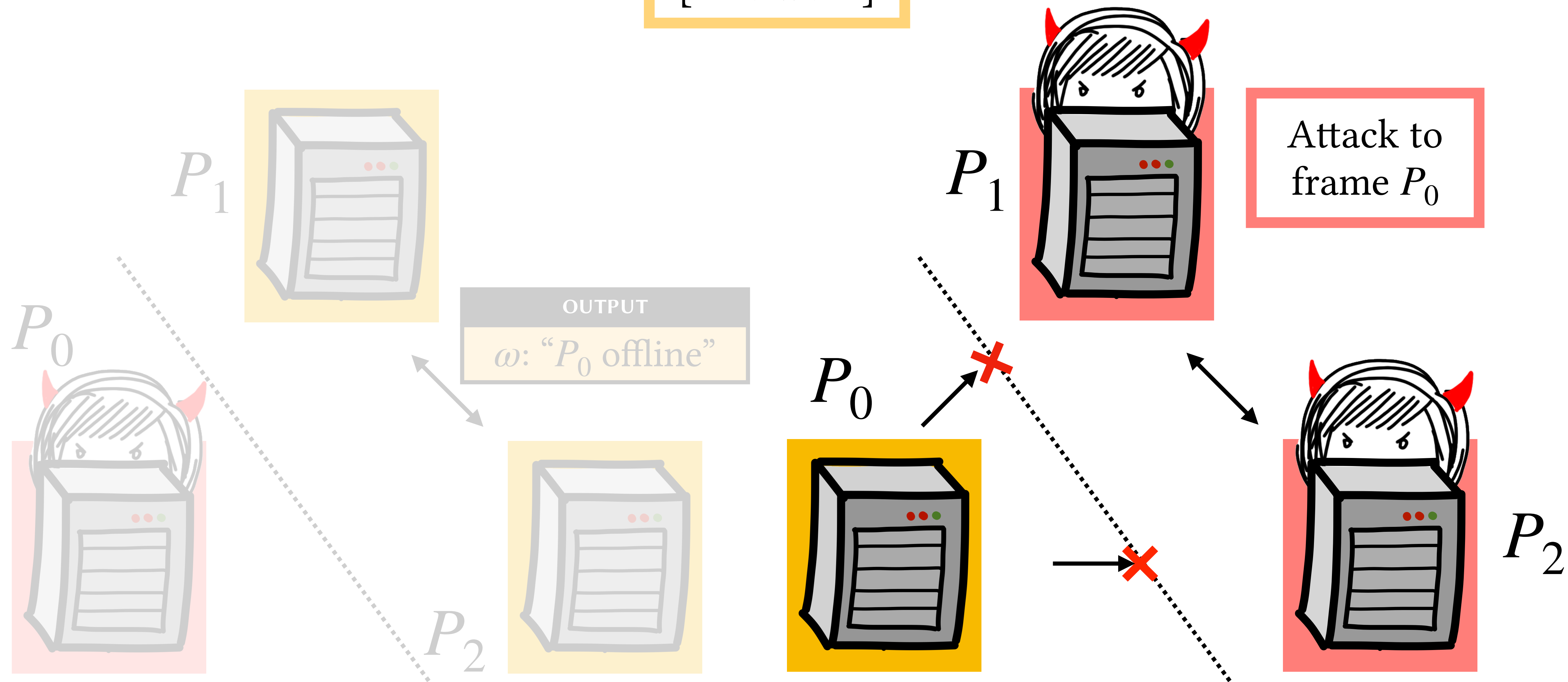
Broadcast-IA is Impossible with Dishonest Majority

[This work]



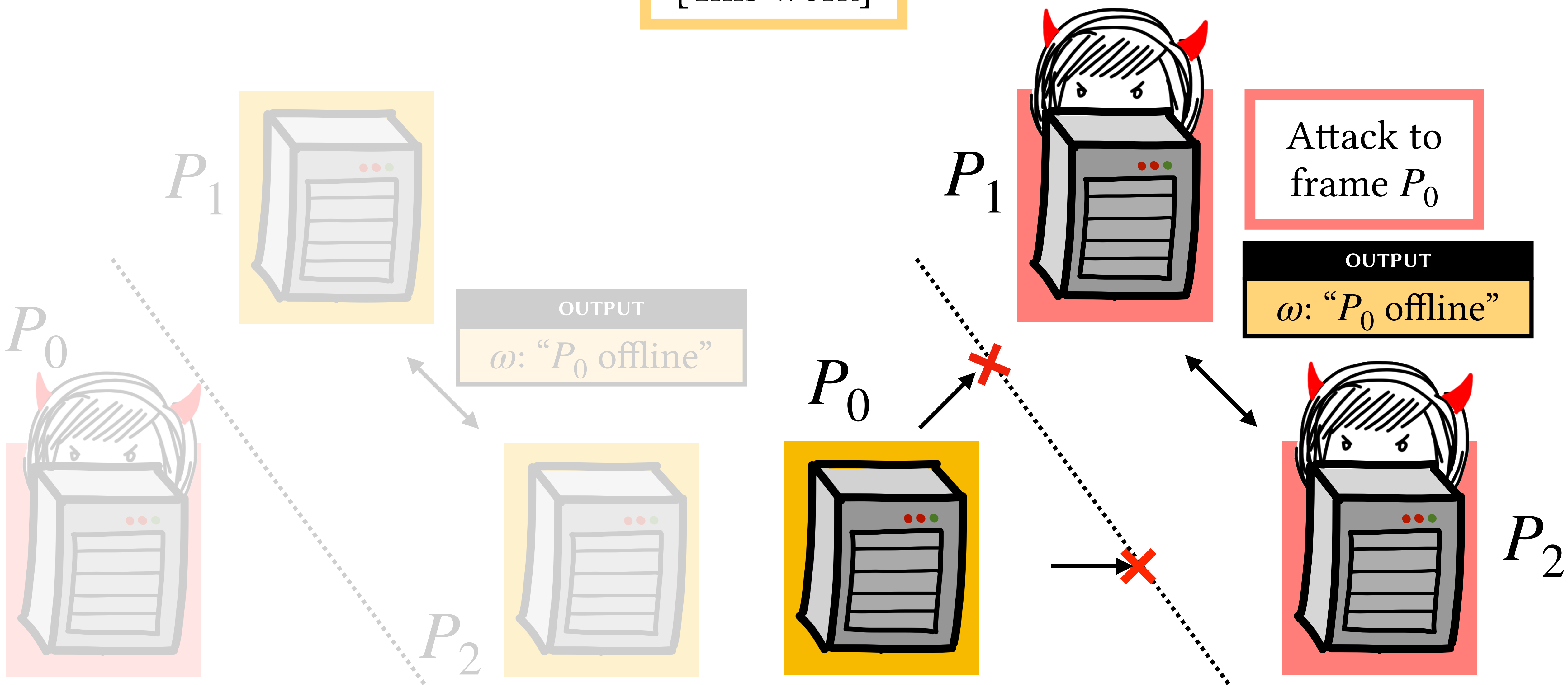
Broadcast-IA is Impossible with Dishonest Majority

[This work]



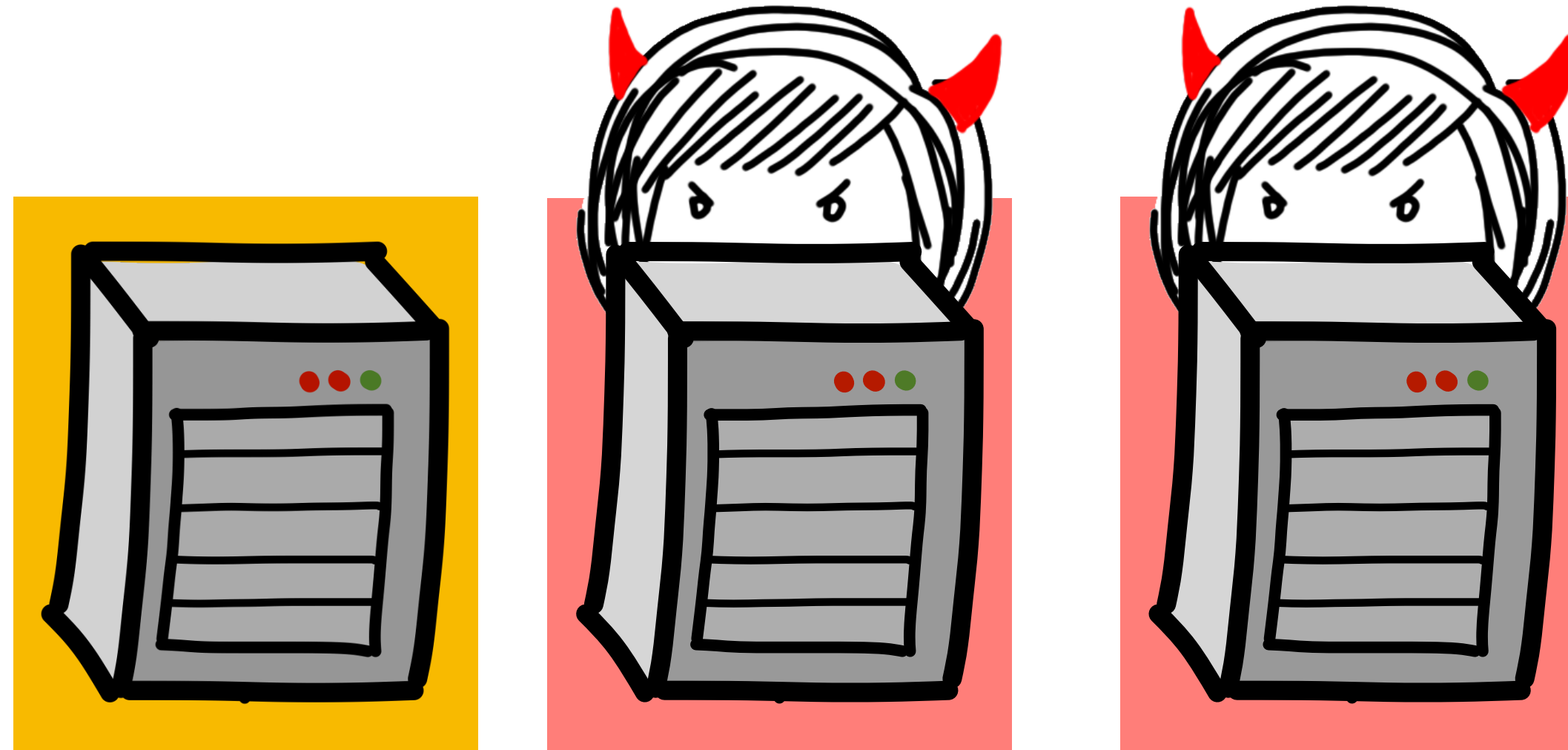
Broadcast-IA is Impossible with Dishonest Majority

[This work]



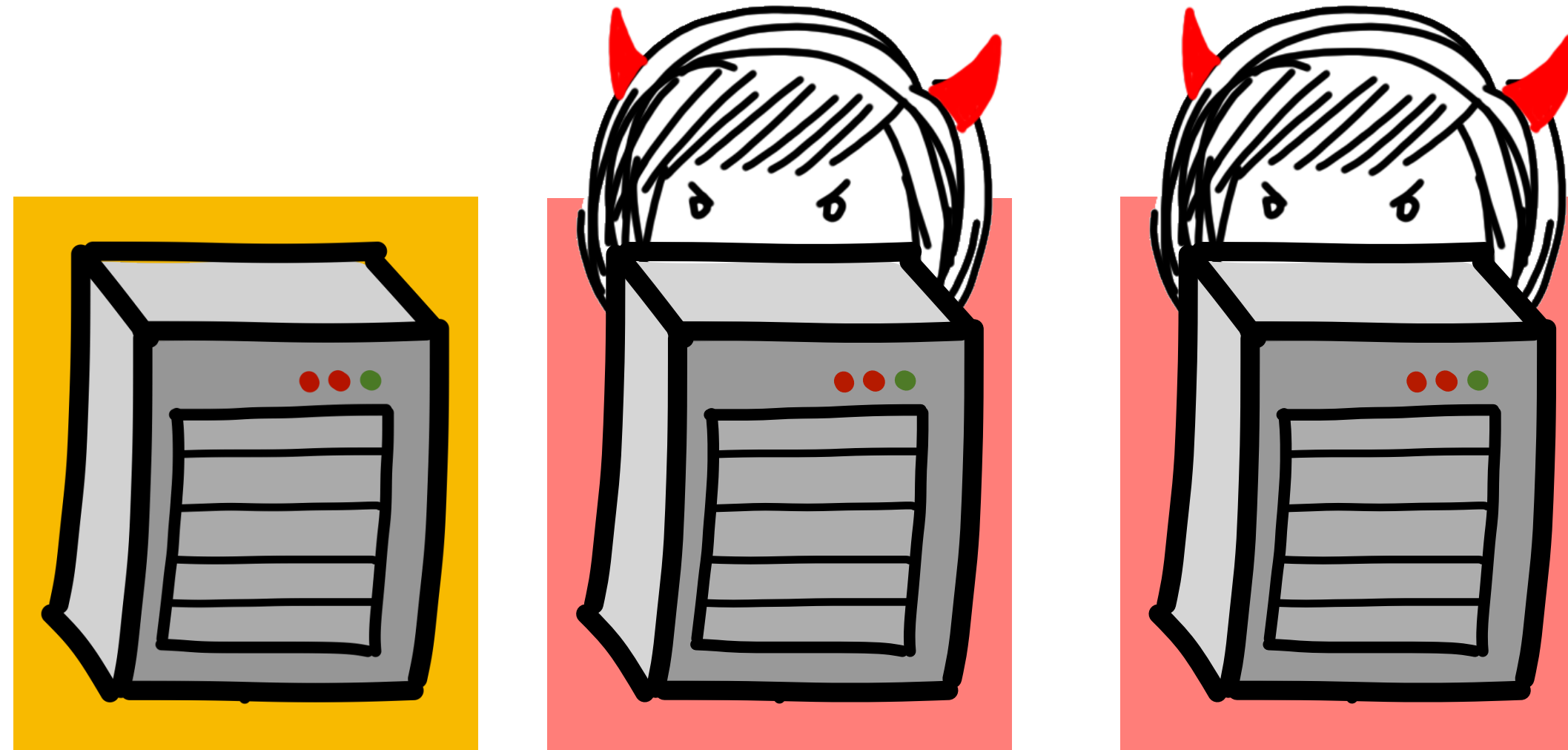
Broadcast-IA is Impossible with Dishonest Majority

[This work]



Broadcast-IA is Impossible with Dishonest Majority

[This work]



Broadcast-IA with Honest Majority

[This work]

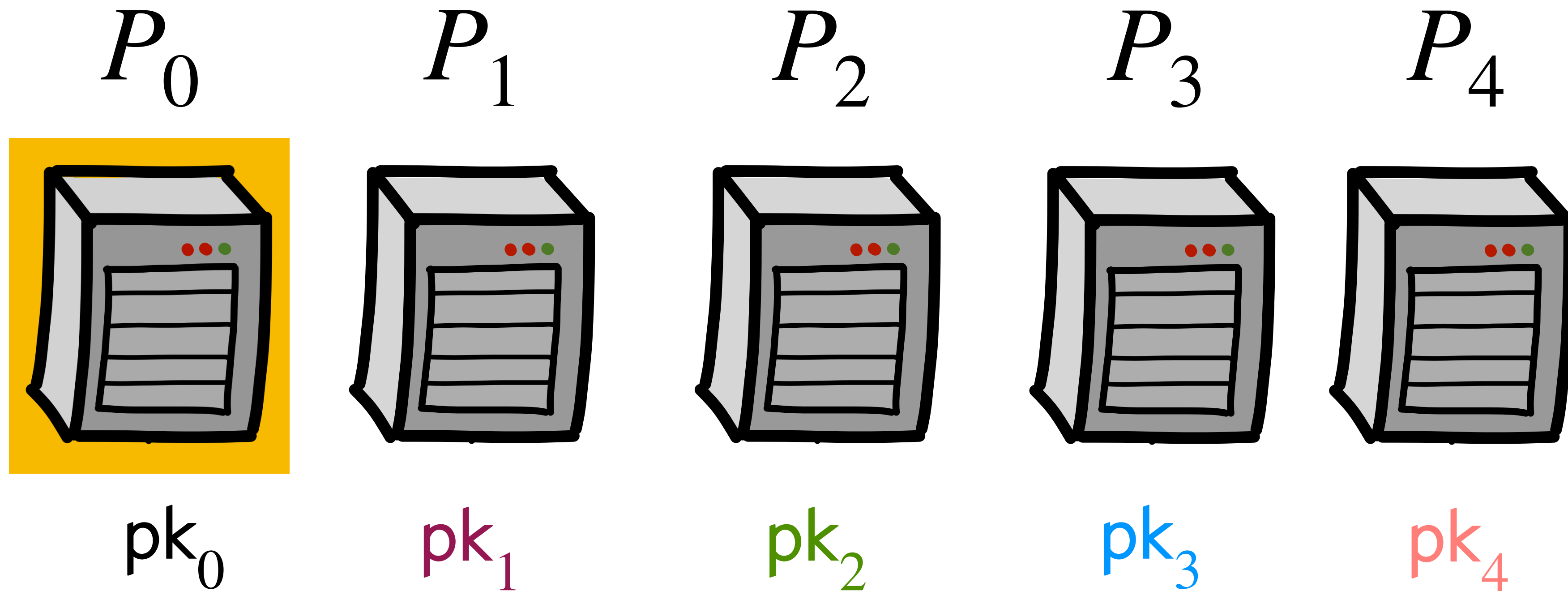


Recall: Global honest majority

Use it proactively

Broadcast-IA with Honest Majority

[This work]



P_0 wishes to broadcast m

Broadcast-IA with Honest Majority

[This work]

Round 1

⋮

Round 2

⋮

Output

Broadcast-IA with Honest Majority

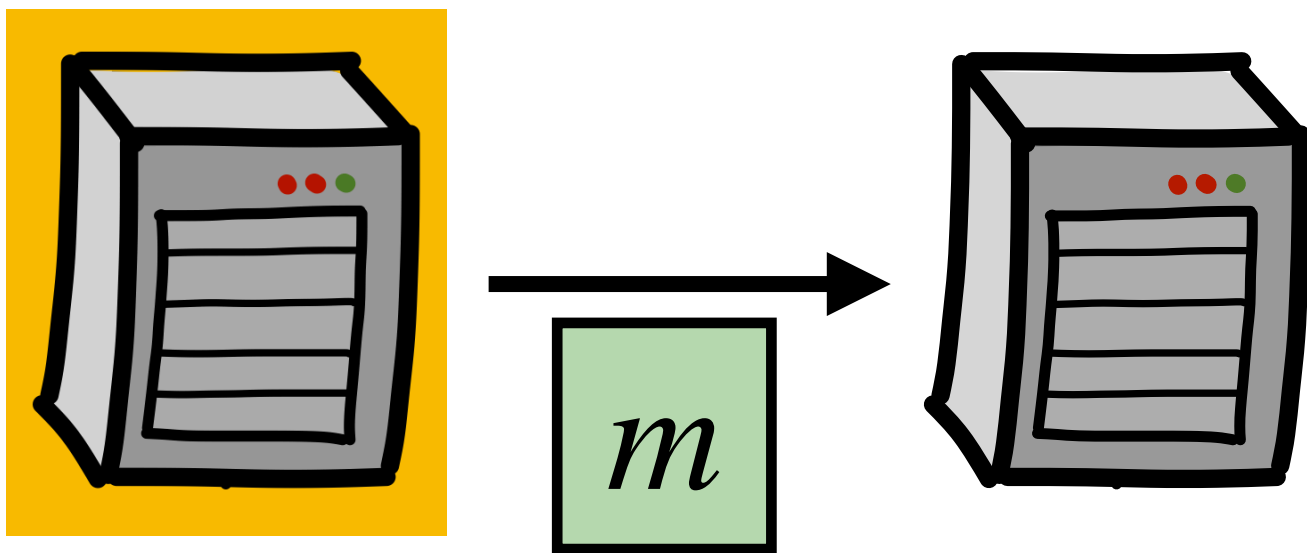
[This work]

Round 1

Sign m ,
Send to all

P_0

P_i



Round 2

Output

Broadcast-IA with Honest Majority

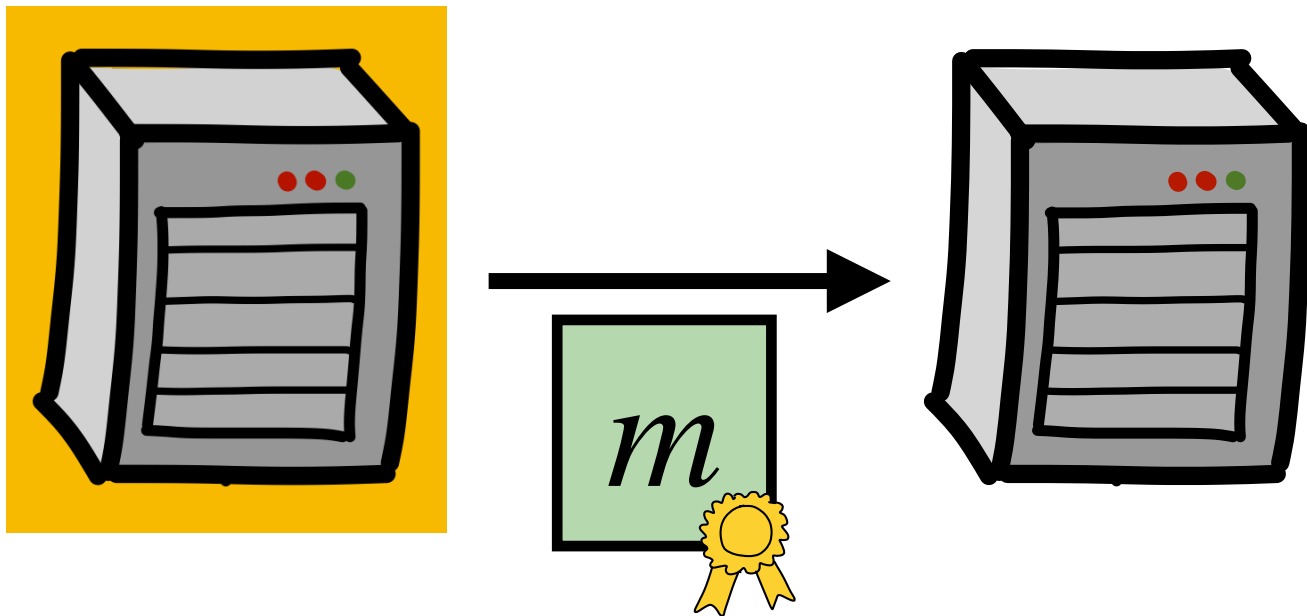
[This work]

Round 1

Sign m ,
Send to all

P_0

P_i

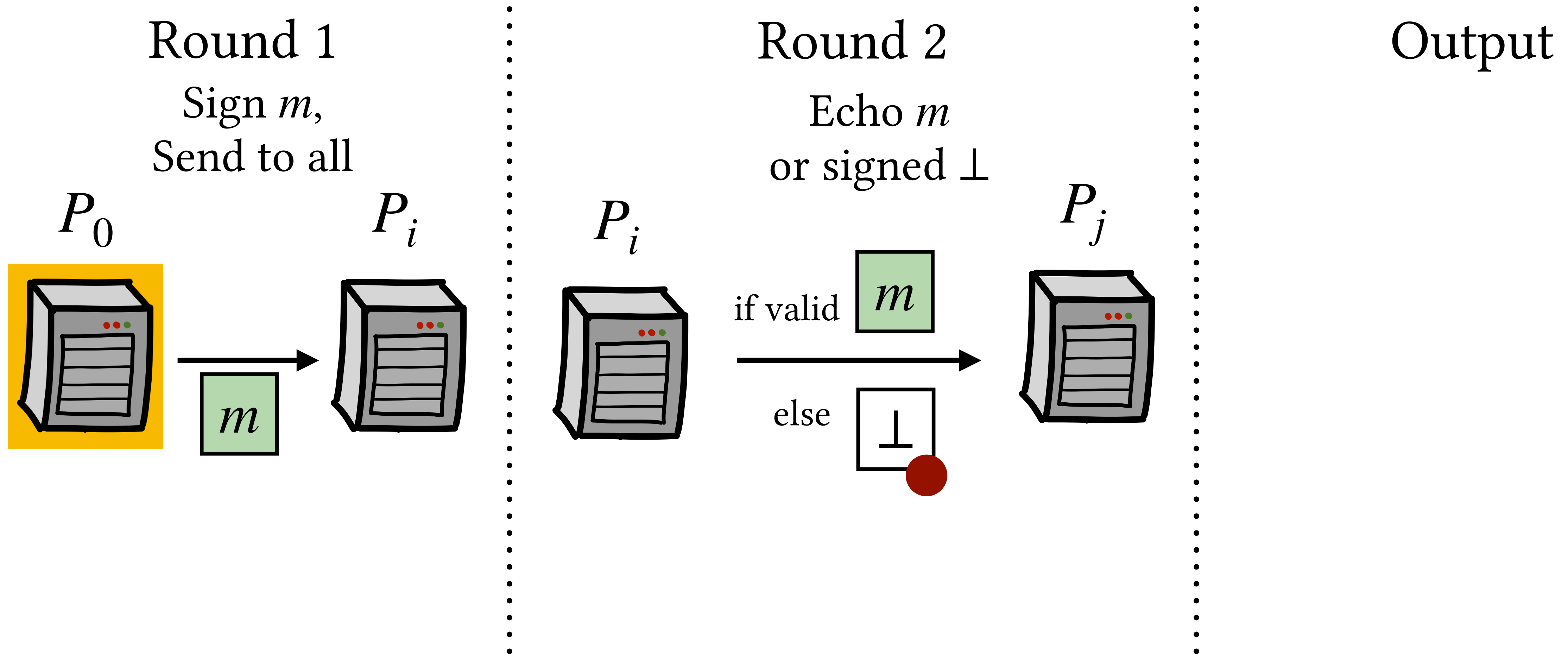


Round 2

Output

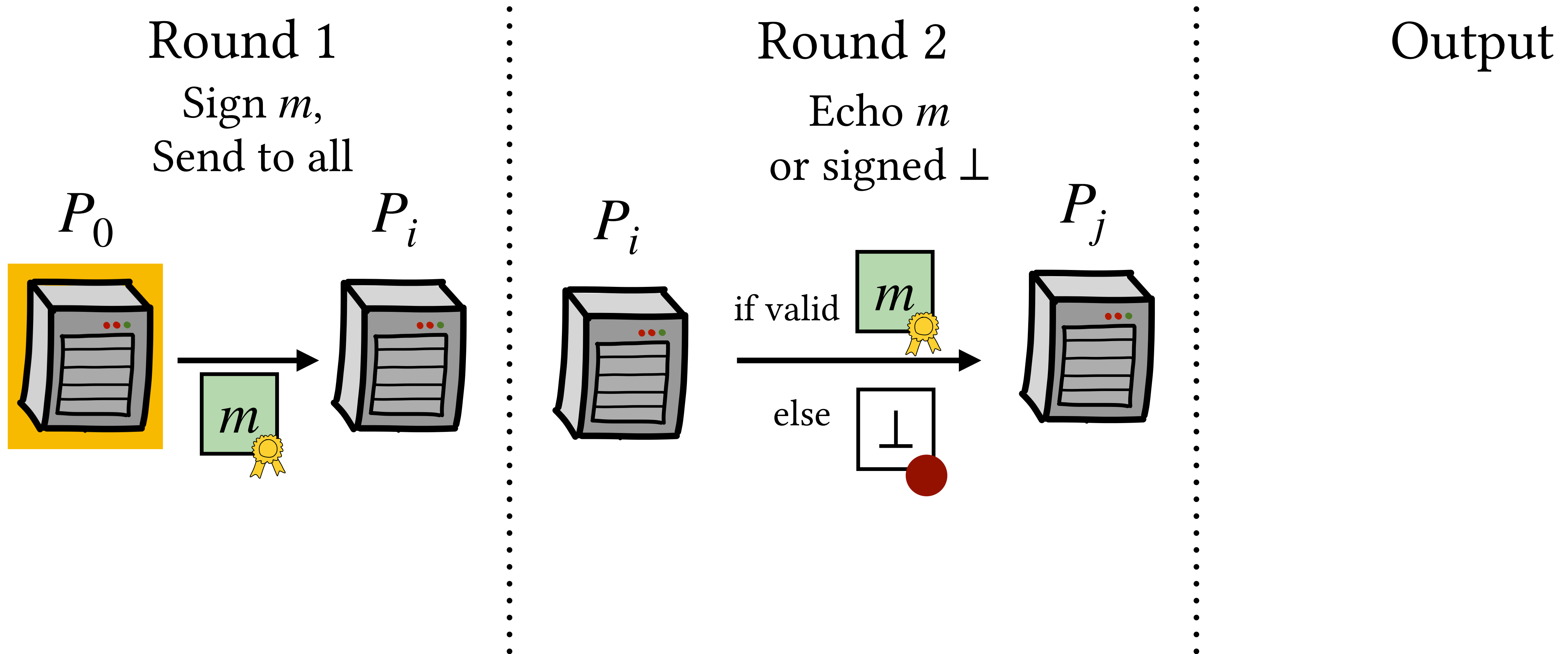
Broadcast-IA with Honest Majority

[This work]



Broadcast-IA with Honest Majority

[This work]

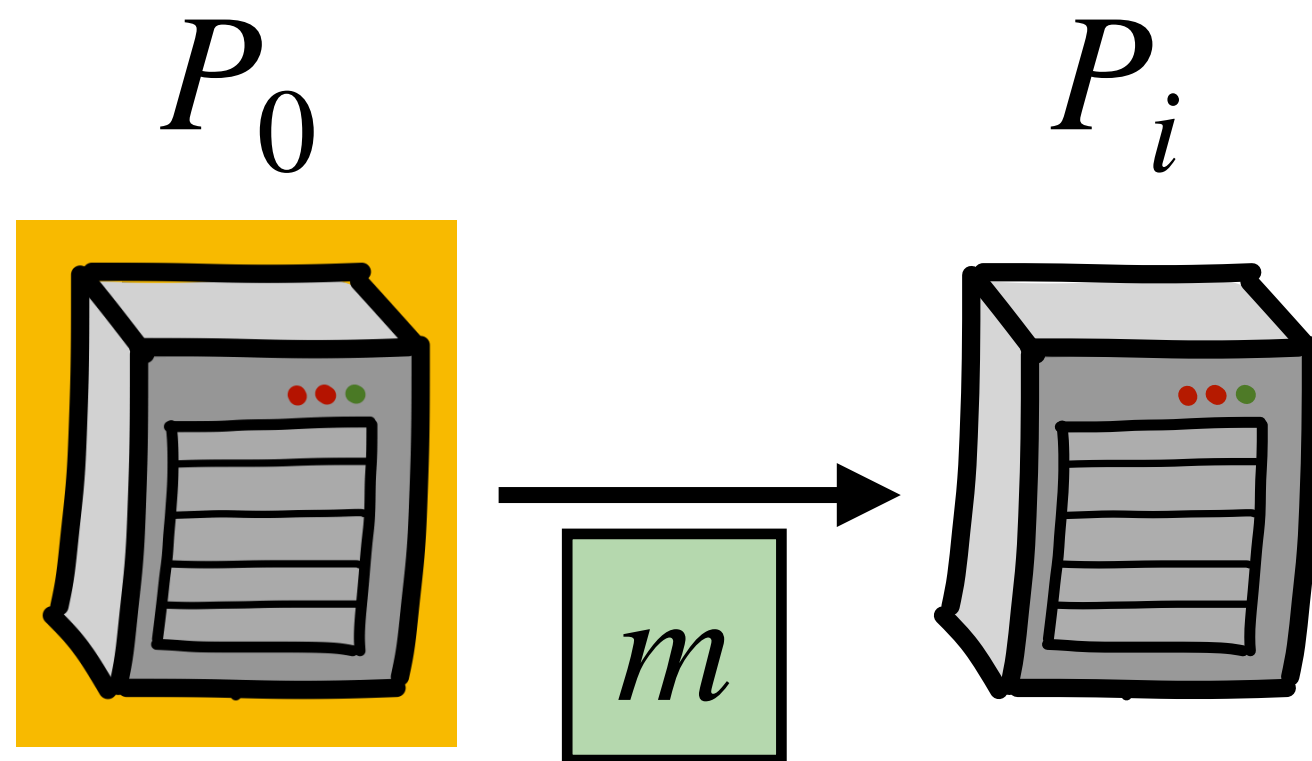


Broadcast-IA with Honest Majority

[This work]

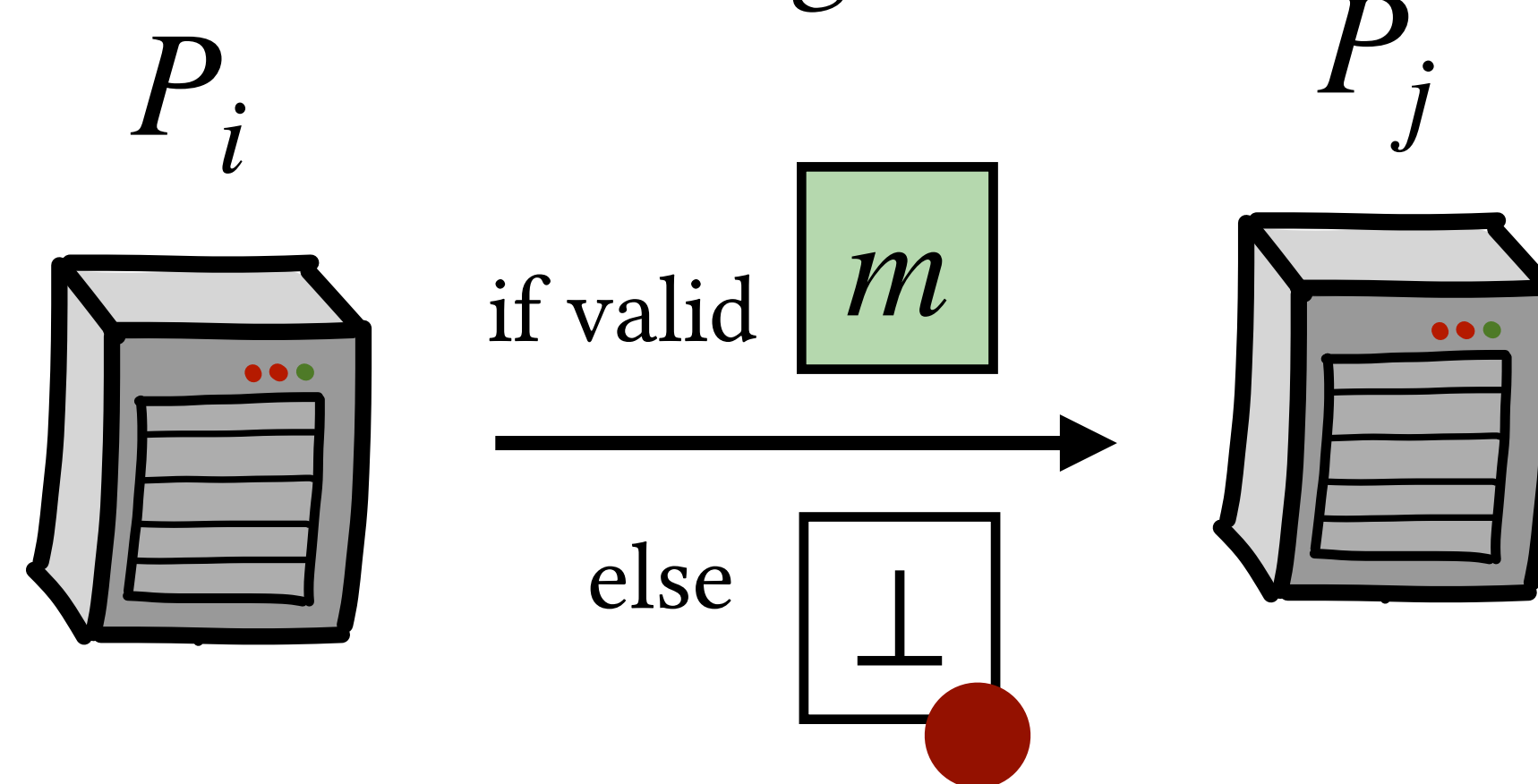
Round 1

Sign m ,
Send to all



Round 2

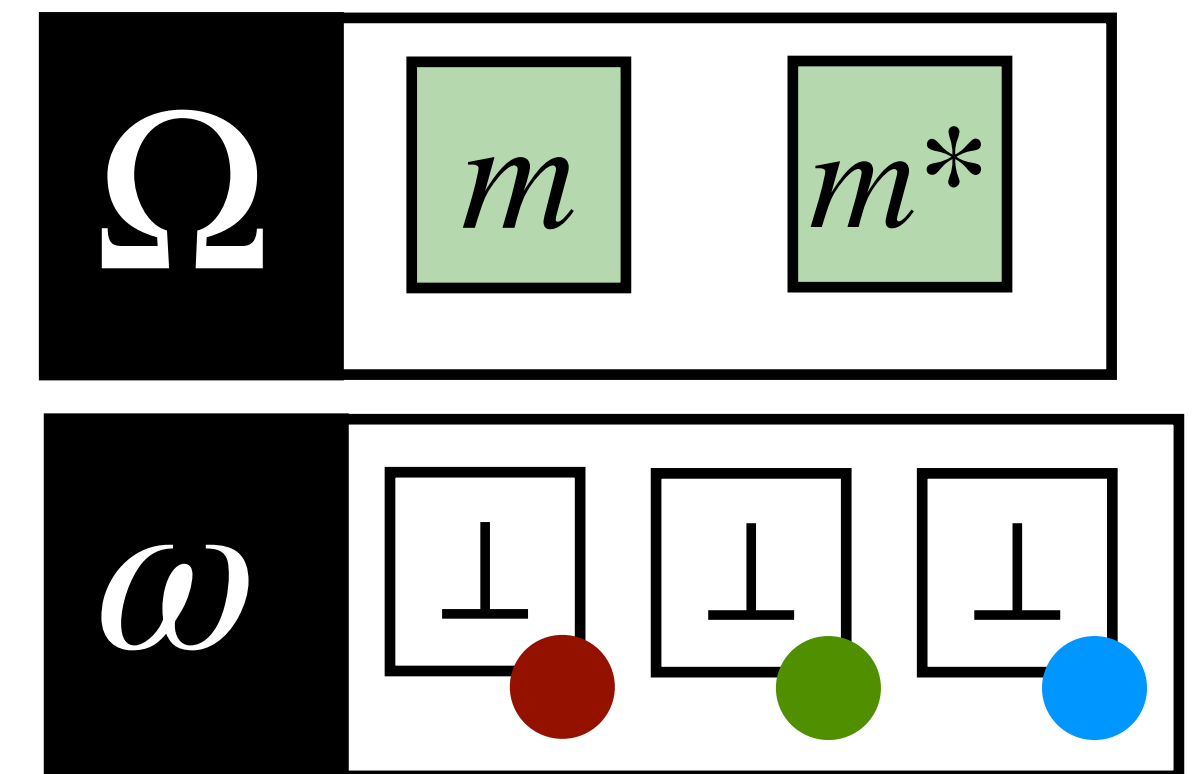
Echo m
or signed \perp



Output

Each P_i (server icon):

1. Check for potential certificates of cheating:



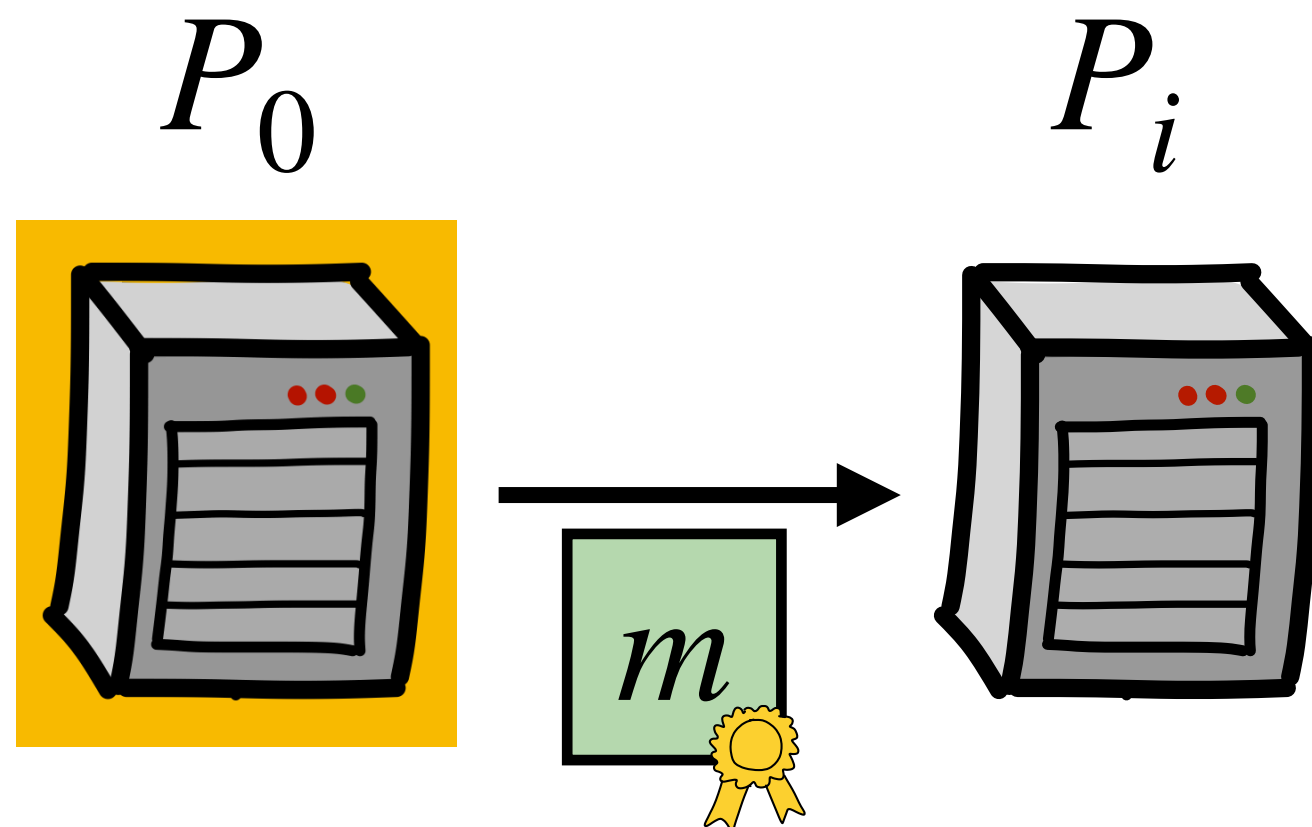
2. If no Ω, ω found,
output m (green box with a gold medal icon)

Broadcast-IA with Honest Majority

[This work]

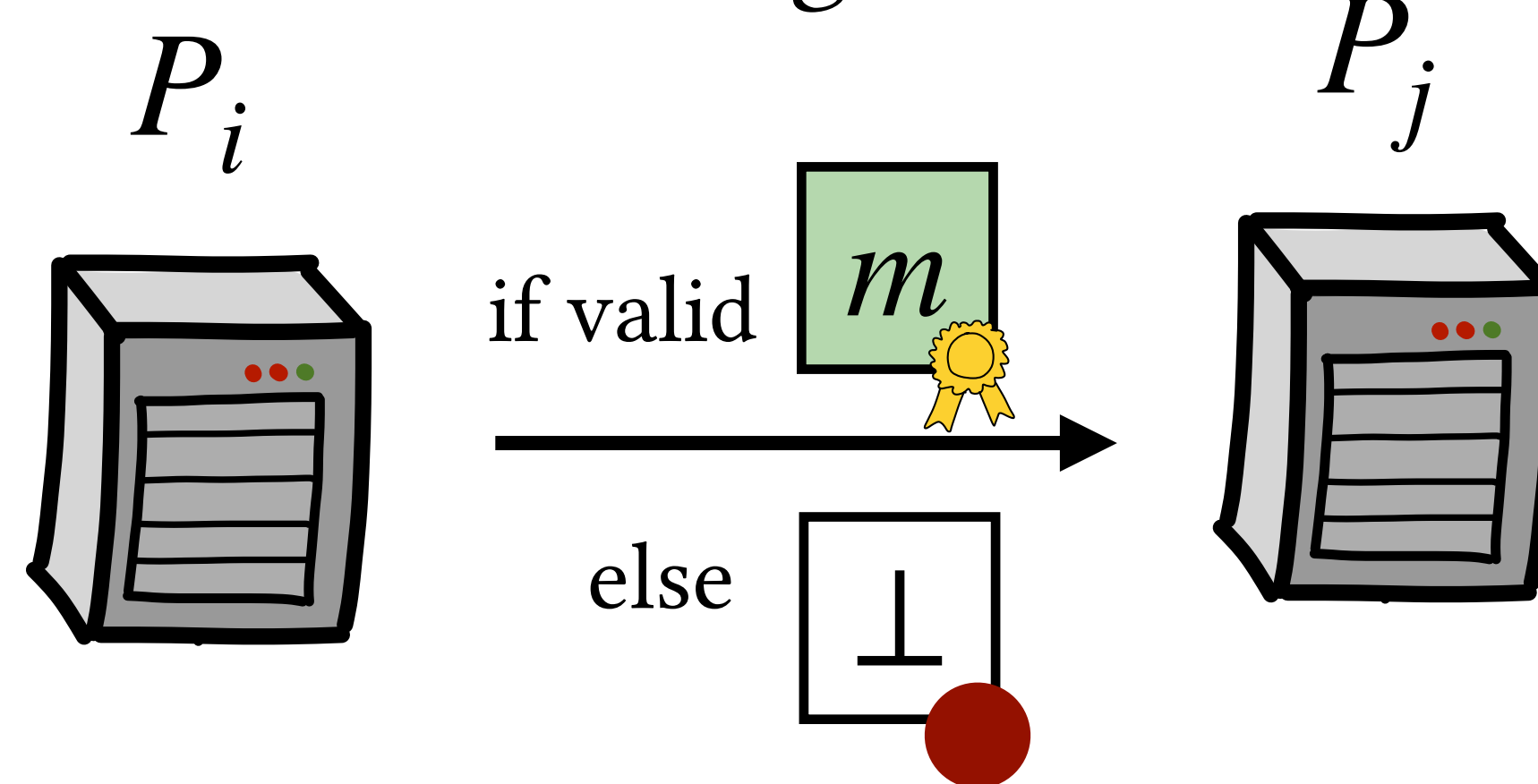
Round 1

Sign m ,
Send to all



Round 2

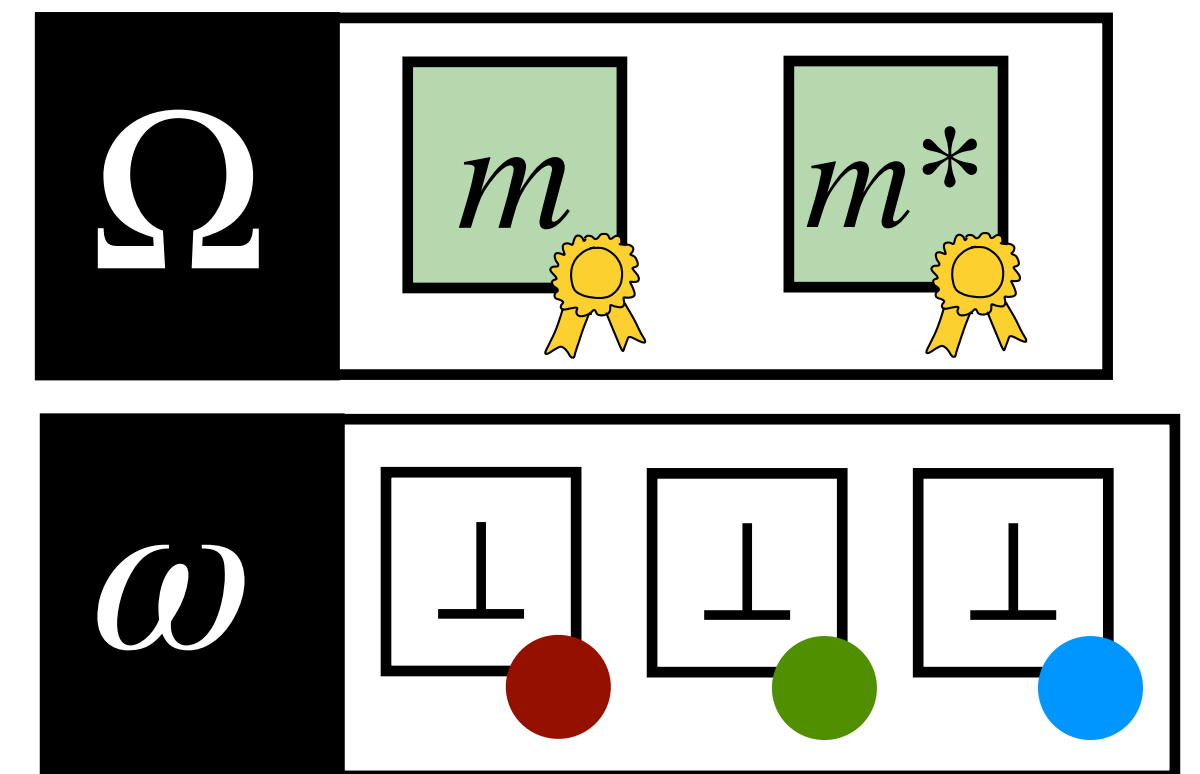
Echo m
or signed \perp



Output

Each P_i (server icon):

1. Check for potential certificates of cheating:



2. If no Ω , ω found,
output m (green box with yellow ribbon)

Broadcast-IA: Analysis

- **Honest P_0 :**
 - No Ω : Will not sending conflicting m, m^*
 - No ω : At most 2 corrupt parties will echo $\perp \Rightarrow$ not enough sigs
- **Corrupt P_0 :**
 - If any honest parties receive $m, m^* \Rightarrow$ yields Ω
 - If m withheld from *all* honest parties \Rightarrow yields ω

Therefore, each honest party outputs either Ω, ω , or consistent m
- Notes on output m :
 1. Accompanied by $\text{sig}(m)$ from P_0 : proves P_0 sent m to P_i
 2. P_i producing $\text{sig}(m)$ DOES NOT prove that some P_j also output m

Building ECDSA-IA

- Baseline ECDSA protocol: Honest Majority variant of [DKLs23]
 - hm-[DKLs23]: One broadcast round on top of VSS + DKG
 - This work: one broadcast + Schnorr-like NIZK, on top of VSS-IA + DKG-IA

Building ECDSA-IA

- Baseline ECDSA protocol: Honest Majority variant of [DKLs23]
 - hm-[DKLs23]: One broadcast round on top of VSS + DKG
 - This work: one broadcast + Schnorr-like NIZK, on top of VSS-IA + DKG-IA
- Message consistency layer

Building ECDSA-IA

- Baseline ECDSA protocol: Honest Majority variant of [DKLs23]
 - hm-[DKLs23]: One broadcast round on top of VSS + DKG
 - This work: one broadcast + Schnorr-like NIZK, on top of VSS-IA + DKG-IA
- **VSS-IA**: Pedersen-style VSS over broadcast.
 - Success: Samples a Pedersen commit of secret uniform value
 - Fail: Only in case of malformed ciphertext $P_i \rightarrow P_j$. Then P_j computes Ω as an opening to that ciphertext.
- **DKG-IA**: Run VSS-IA, unmask Pedersen commitment (w. Schnorr NIZK)
- Overall, 3 broadcast-IA rounds, no Paillier/OT in honest-majority setting

Message consistency layer

In Conclusion

- Dishonest majority protocols are inherently DoS-susceptible
 - Can get around this with secure broadcast \Rightarrow extra assumptions
- We define Broadcast-IA to detect cheaters: silent parties and protocol deviations
 - Provably impossible w. dishonest majority
 - Simple construction over p2p channels w. honest majority
- We build VSS-IA \rightarrow DKG-IA \rightarrow ECDSA-IA with simple honest majority protocols
 - Leverage global honest majority
 - Orders of magnitude lighter than dishonest majority
- **Forthcoming:** Benchmarks, full paper

Thanks!