Separating Broadcast *from* Cheater Identification

Yashvanth Kondi

SILENCE LABORATORIES

Divya Ravi

UNIVERSITY OF AMSTERDAM

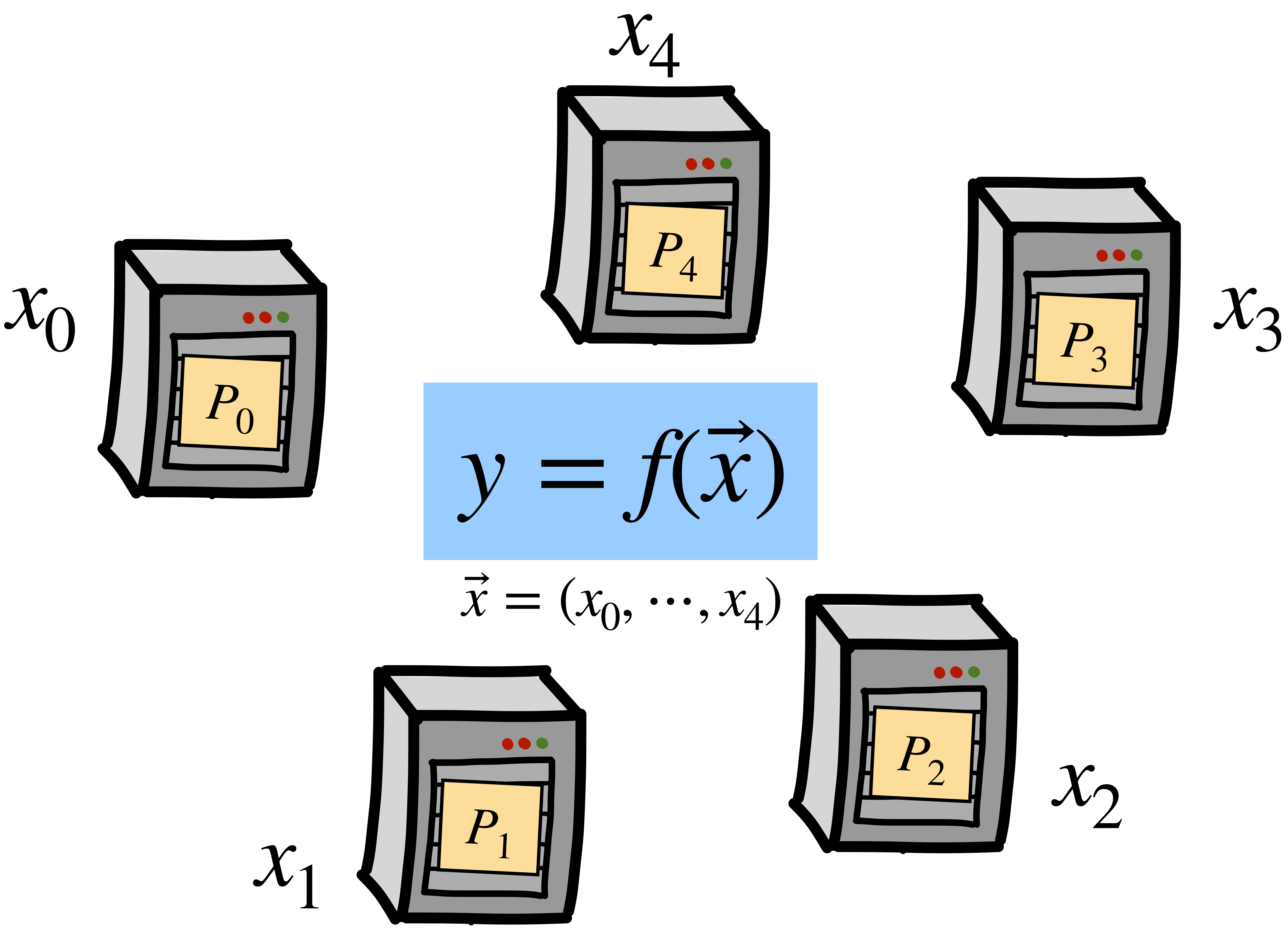
This Talk

- Introduction to Secure Multiparty Computation (MPC) with Identifiable Abort (IA)

- Problem: Most known IA protocols employ broadcast (BC), which is expensive. Is this cost inherent?

- Our results:
  - **Formulate BC-IA** by teasing out the exact requirements on BC in IA setting
  - **Impossibility** in the dishonest majority setting
  - **Simple 2-round BC-IA** in honest majority setting
  - **General compiler**: MPC-IA using $r \times \text{BC} \rightarrow (r+1) \times \text{BC-IA} \rightarrow 2(r+1)$ p2p
  - **Concrete real-world application**: threshold ECDSA signing

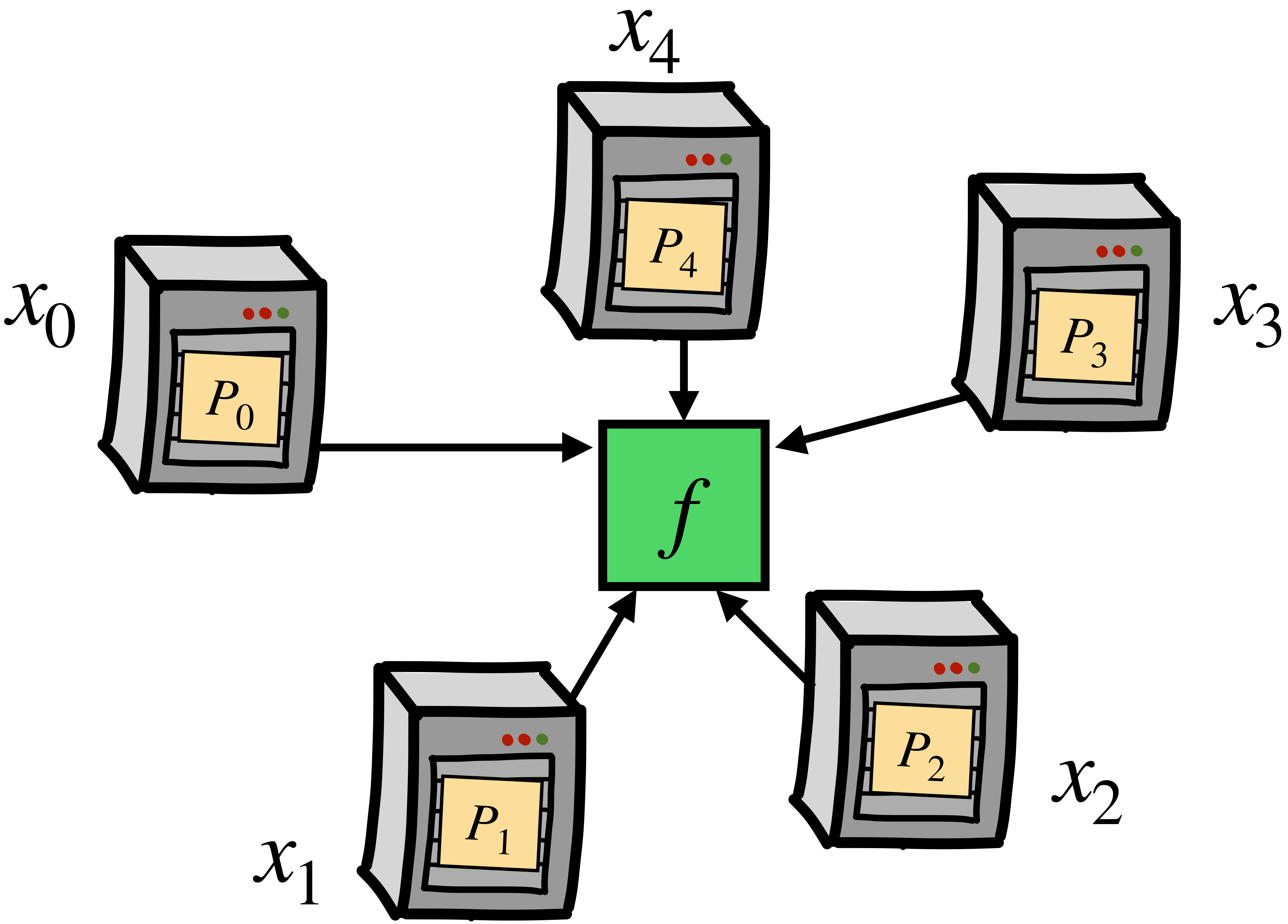
Secure Multiparty Computation (MPC)



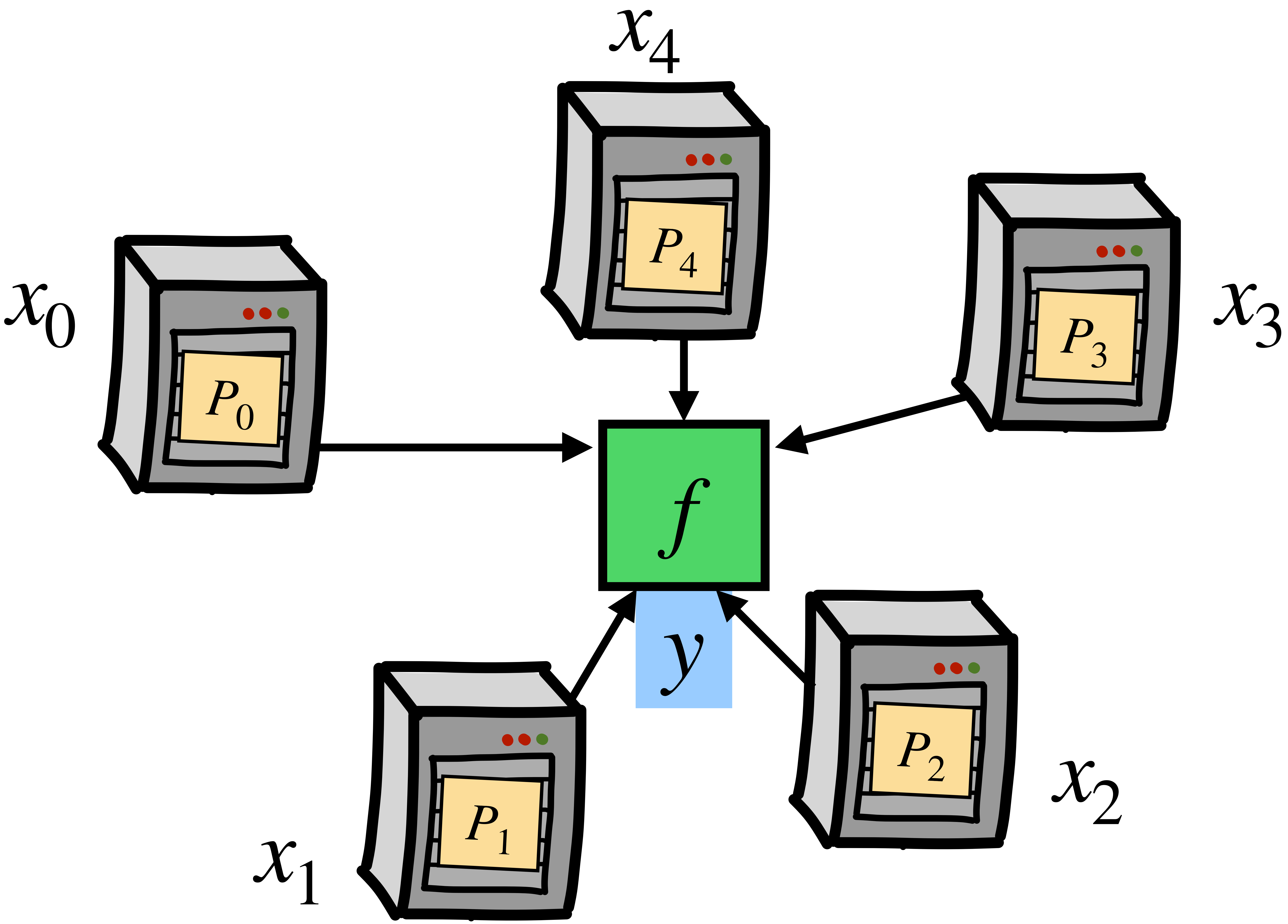
$$y = f(\vec{x})$$

$\vec{x} = (x_0, \cdots, x_4)$

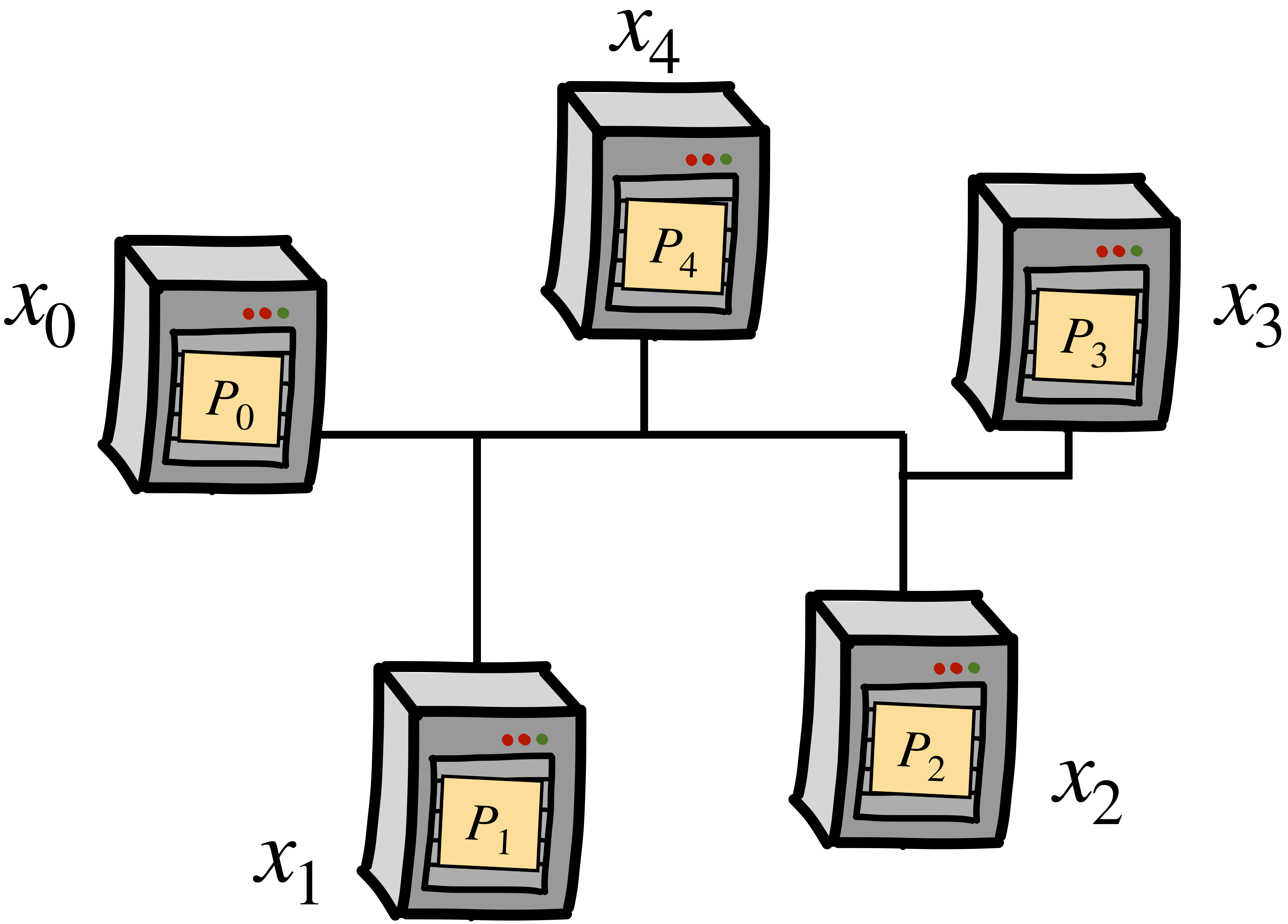
Secure Multiparty Computation (MPC)



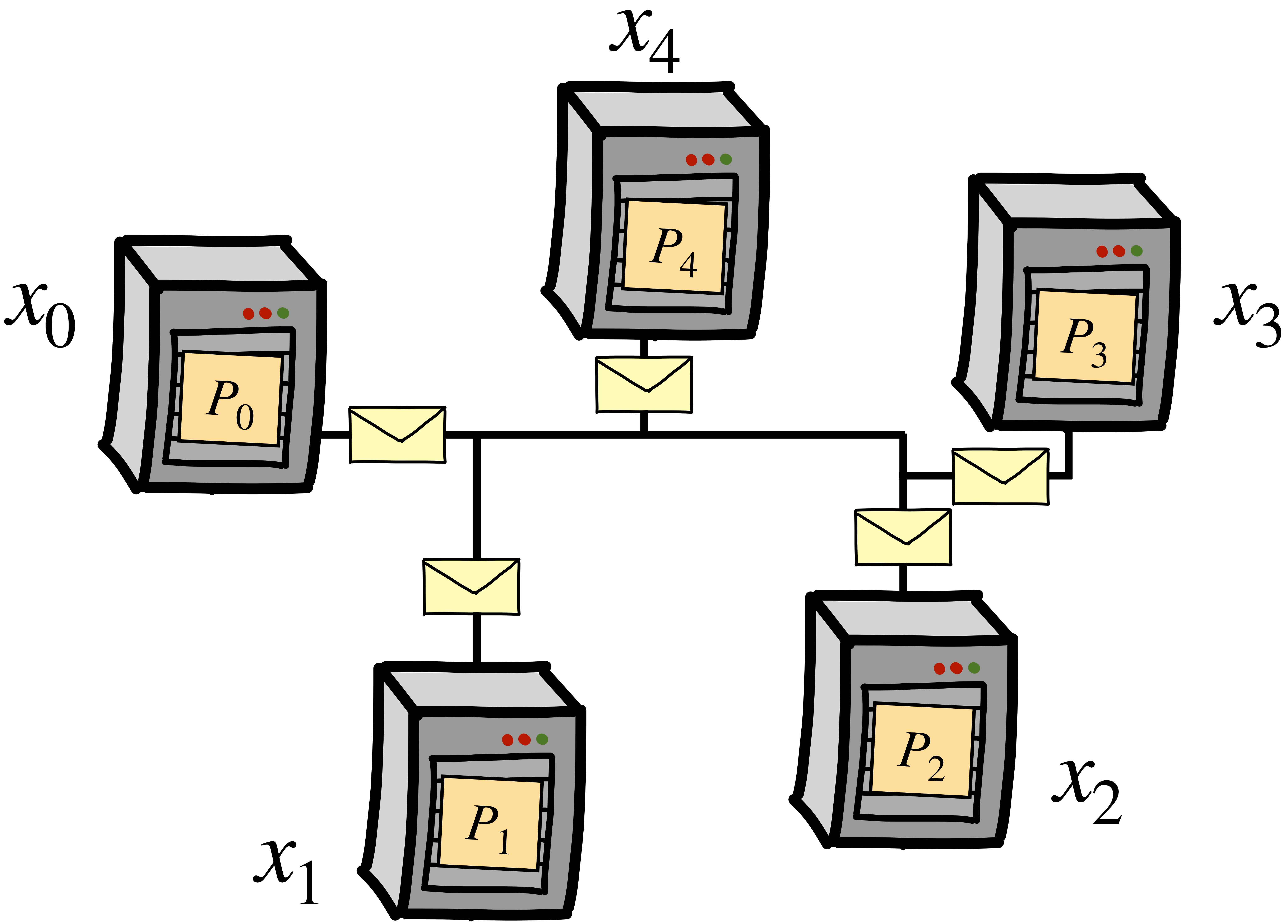
Secure Multiparty Computation (MPC)



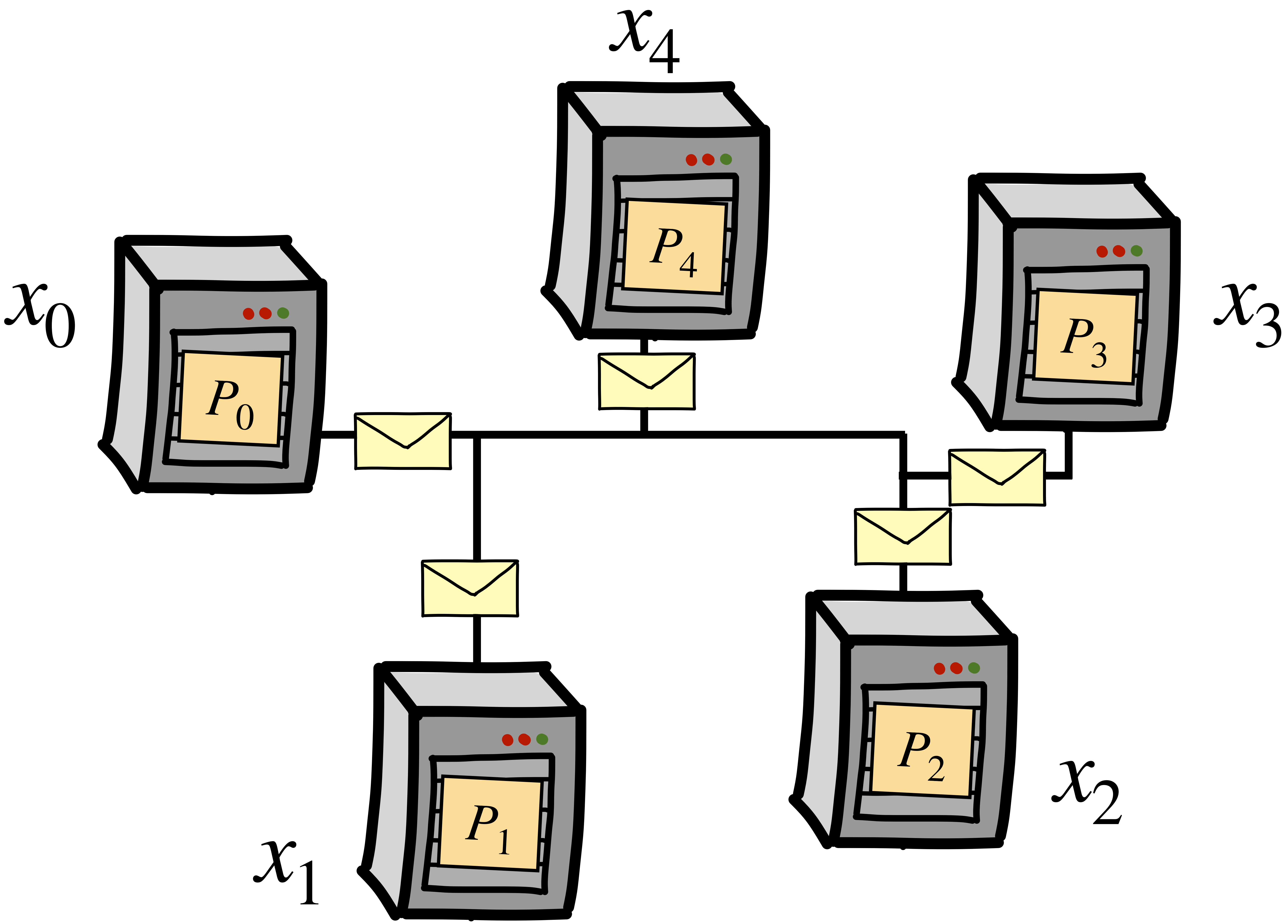
Secure Multiparty Computation (MPC)



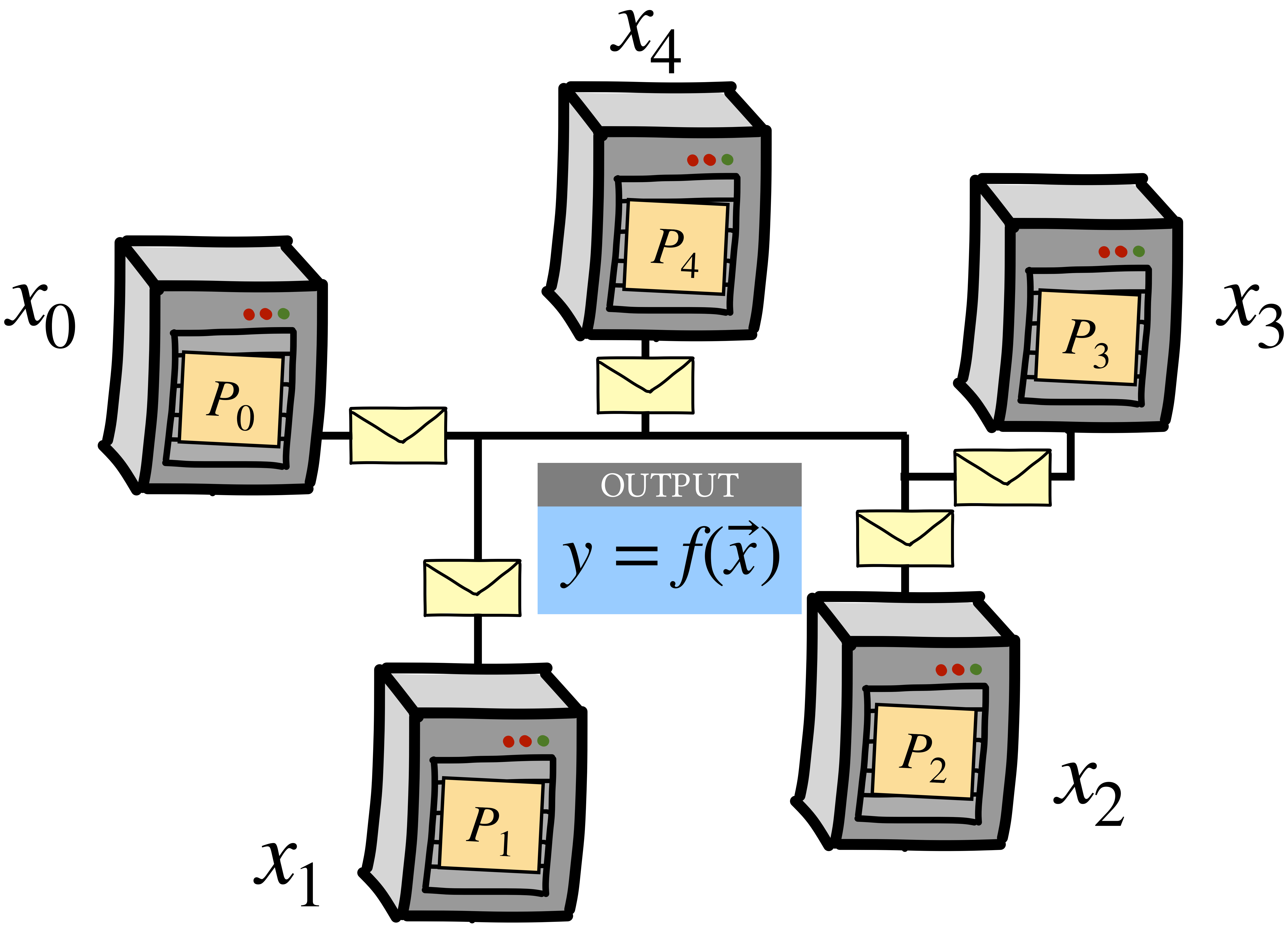
Secure Multiparty Computation (MPC)



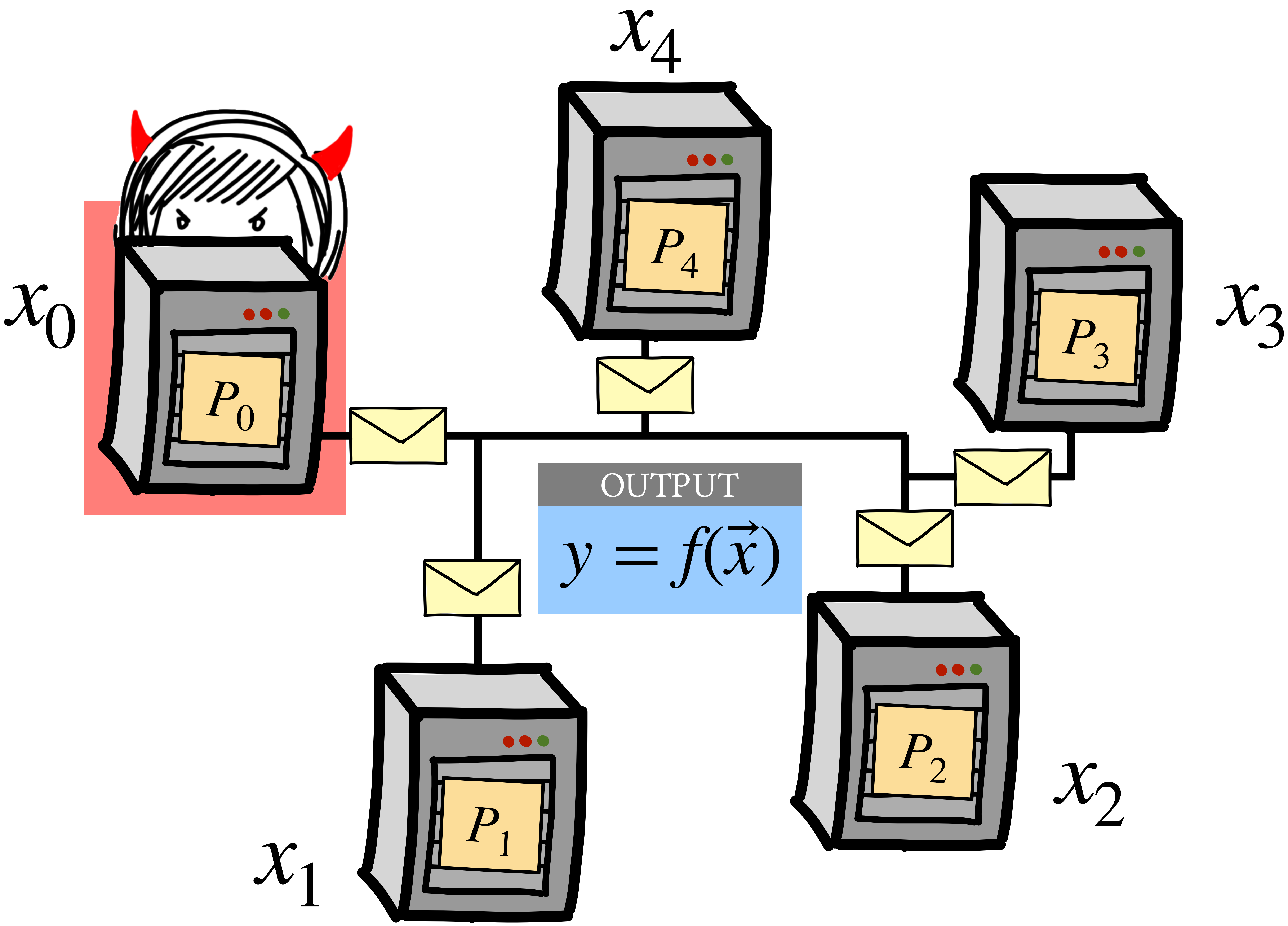
Secure Multiparty Computation (MPC)



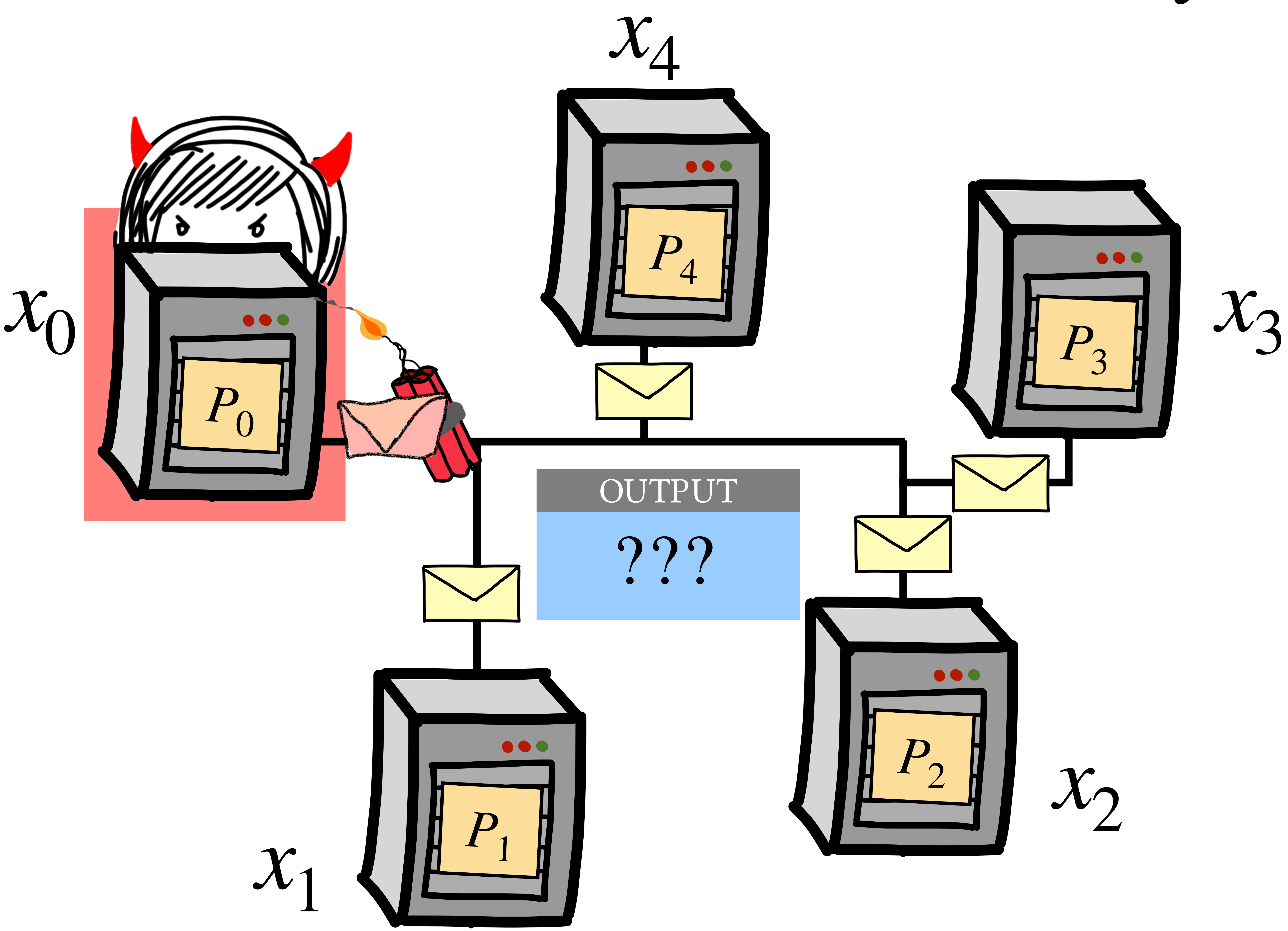
Secure Multiparty Computation (MPC)



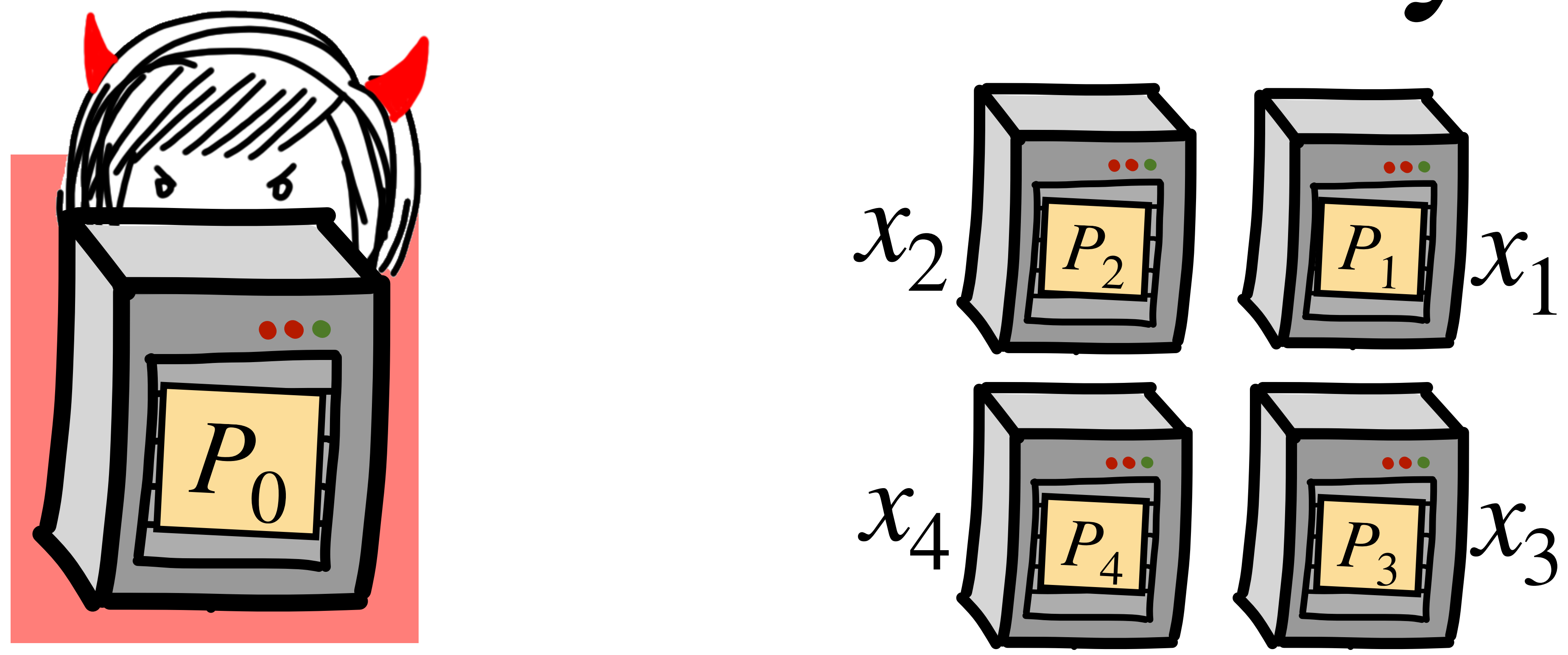
MPC: Active Security



MPC: Active Security



Grades of Active Security

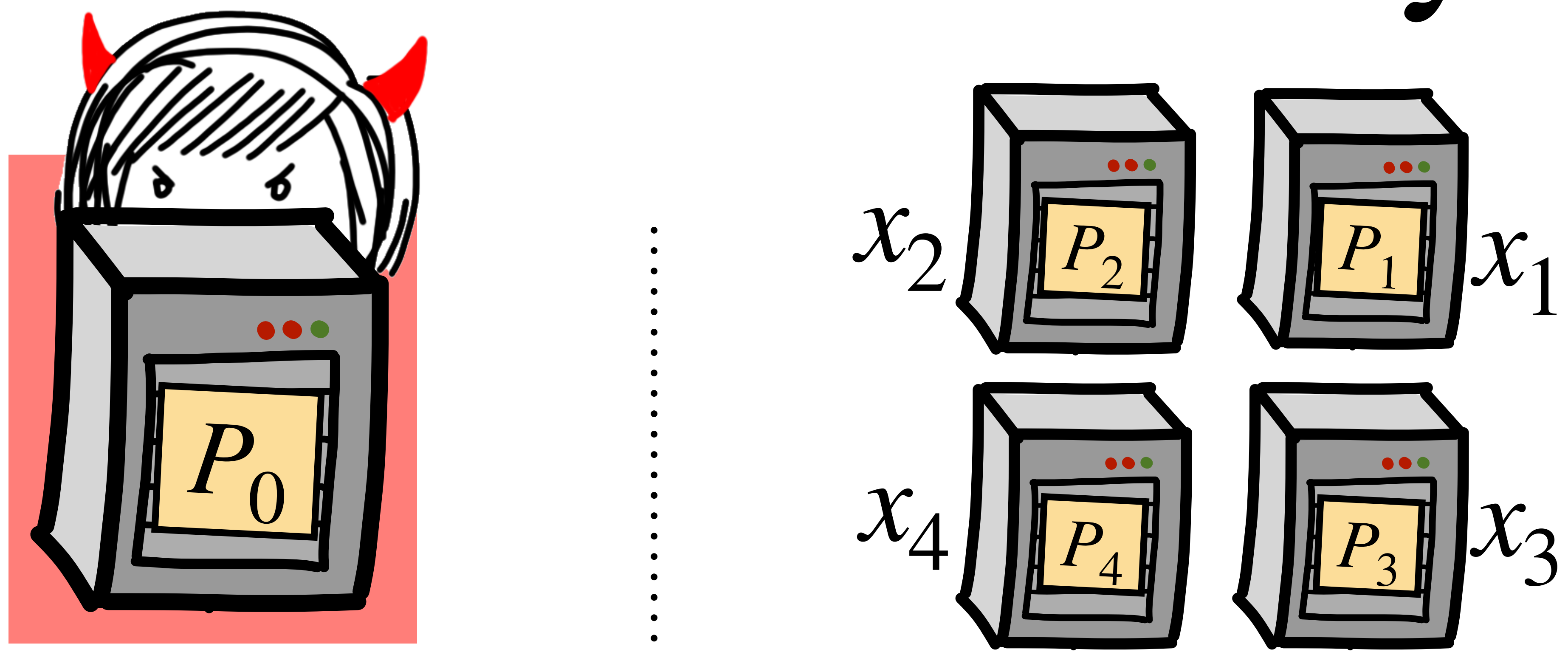


Security with Abort

Fairness

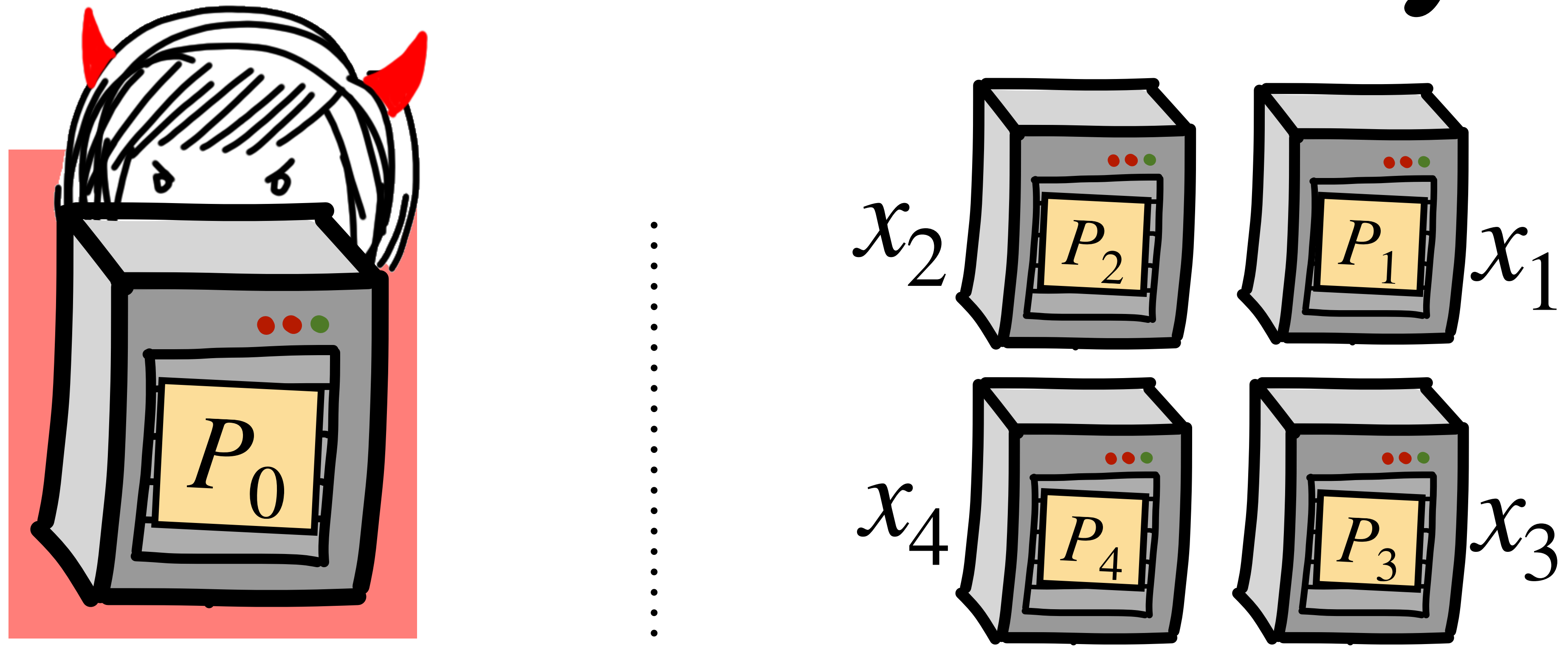
Guaranteed Output

Grades of Active Security



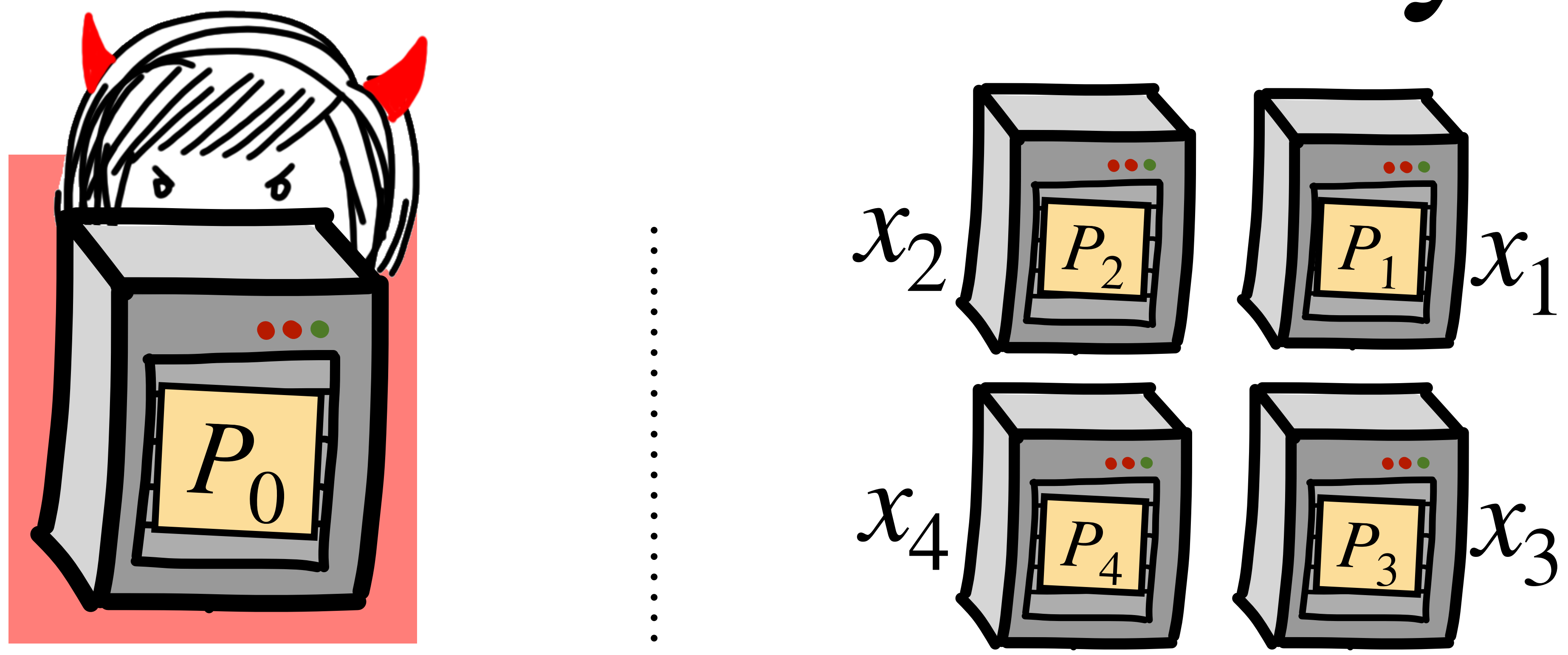
	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security



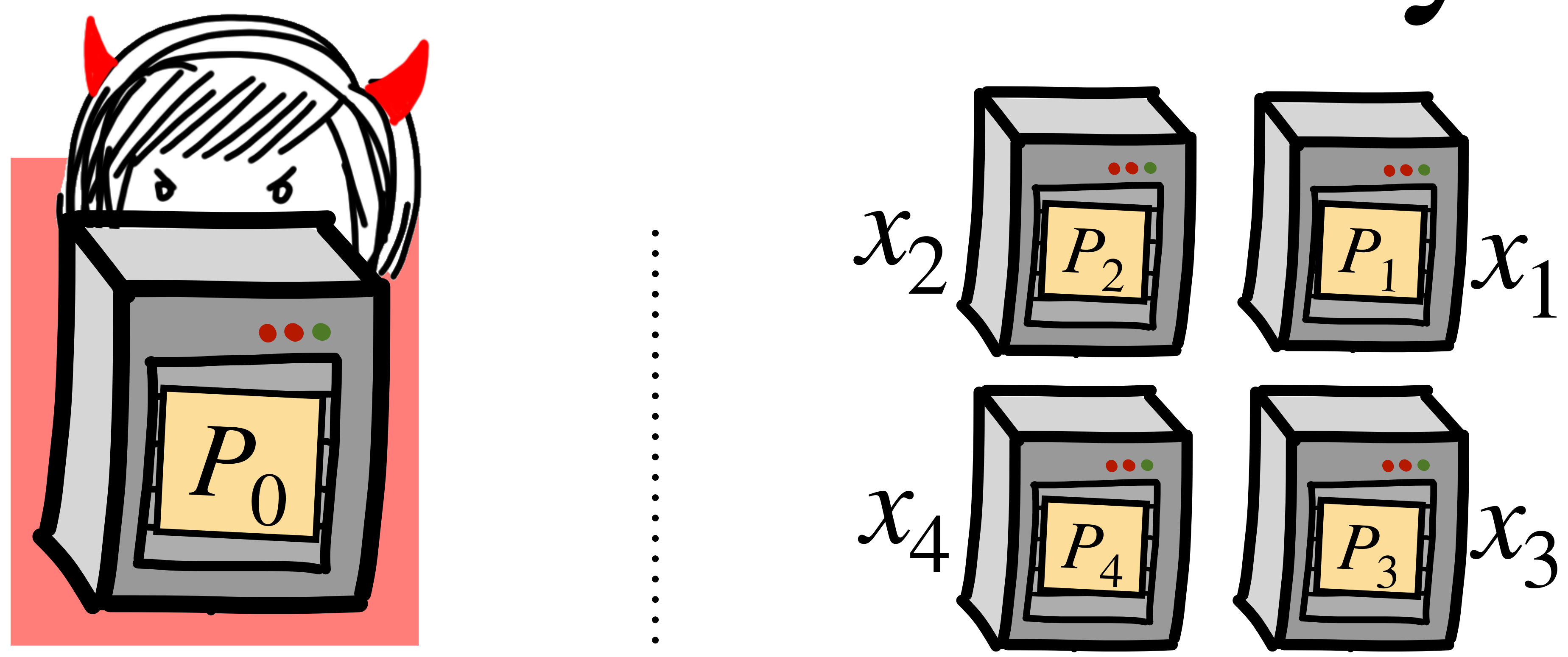
	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security



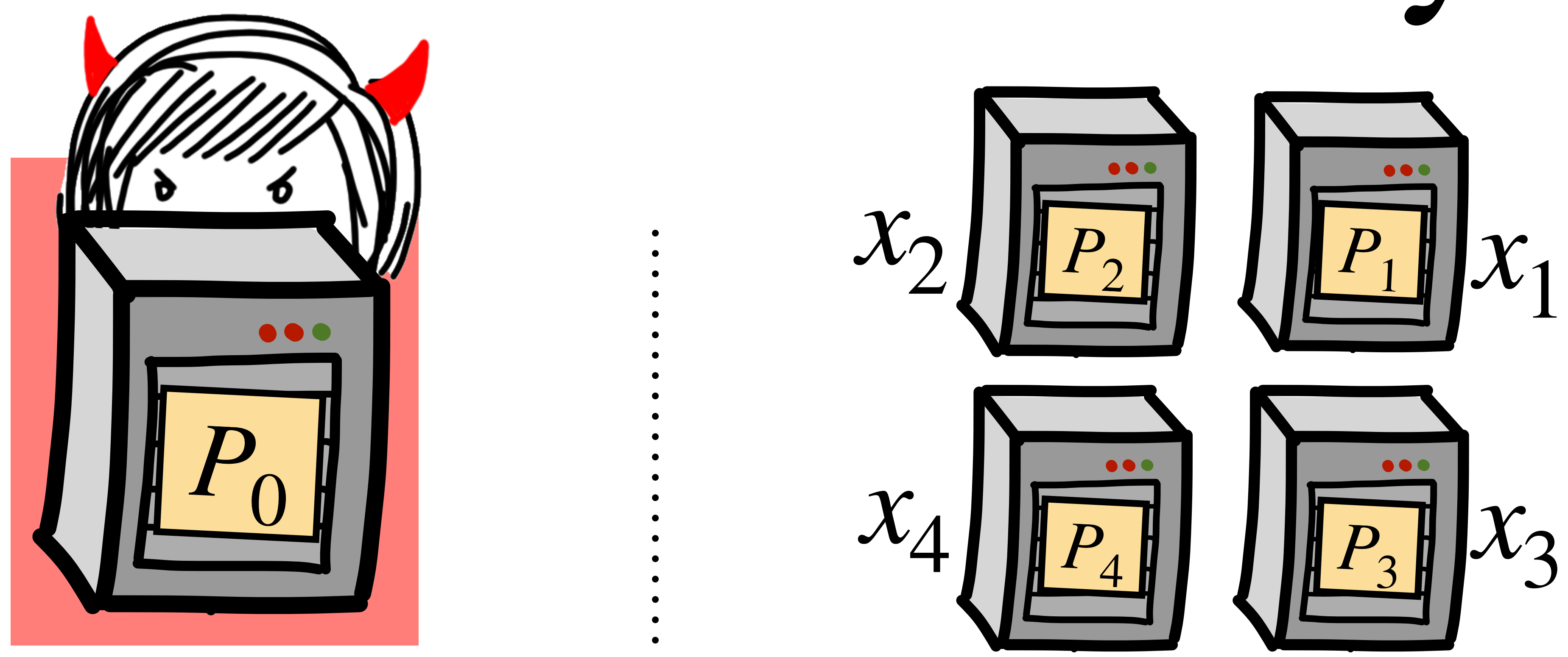
	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security



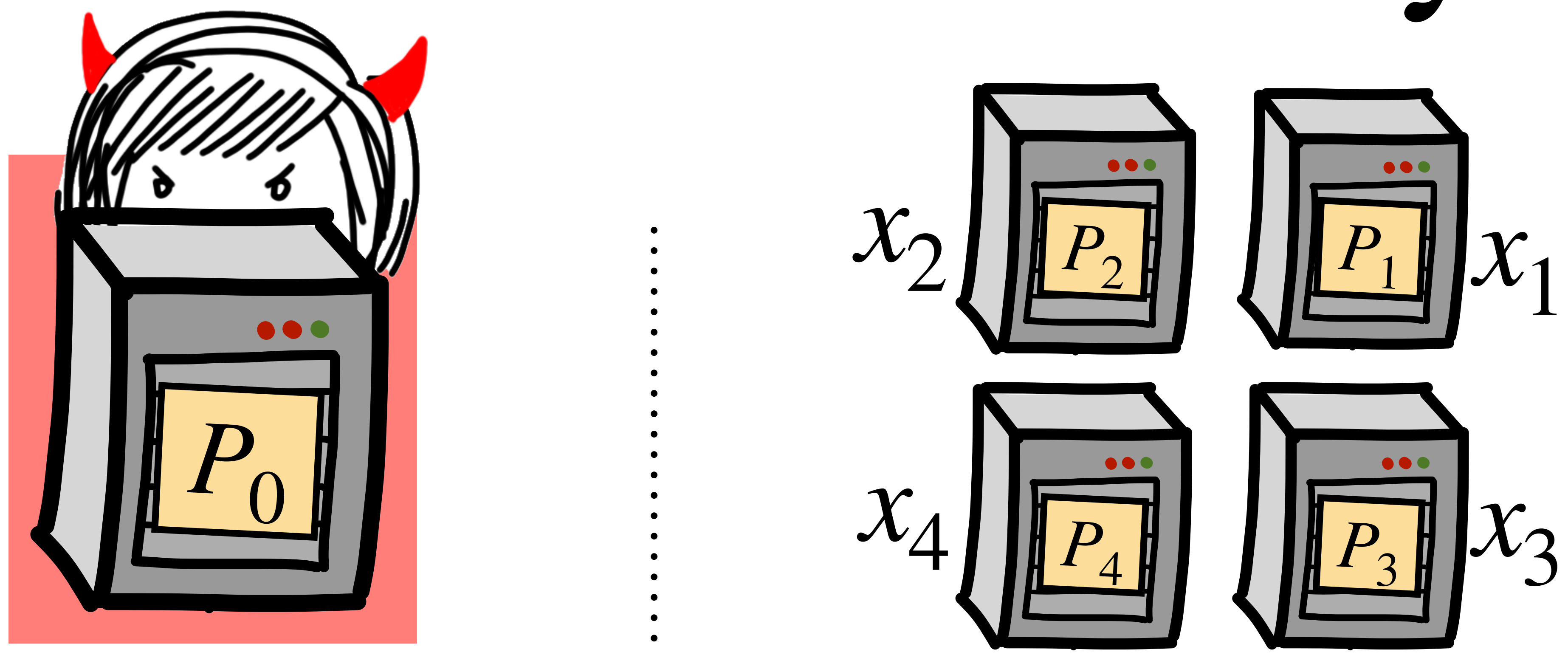
	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security



	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output DoS-resistant	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security

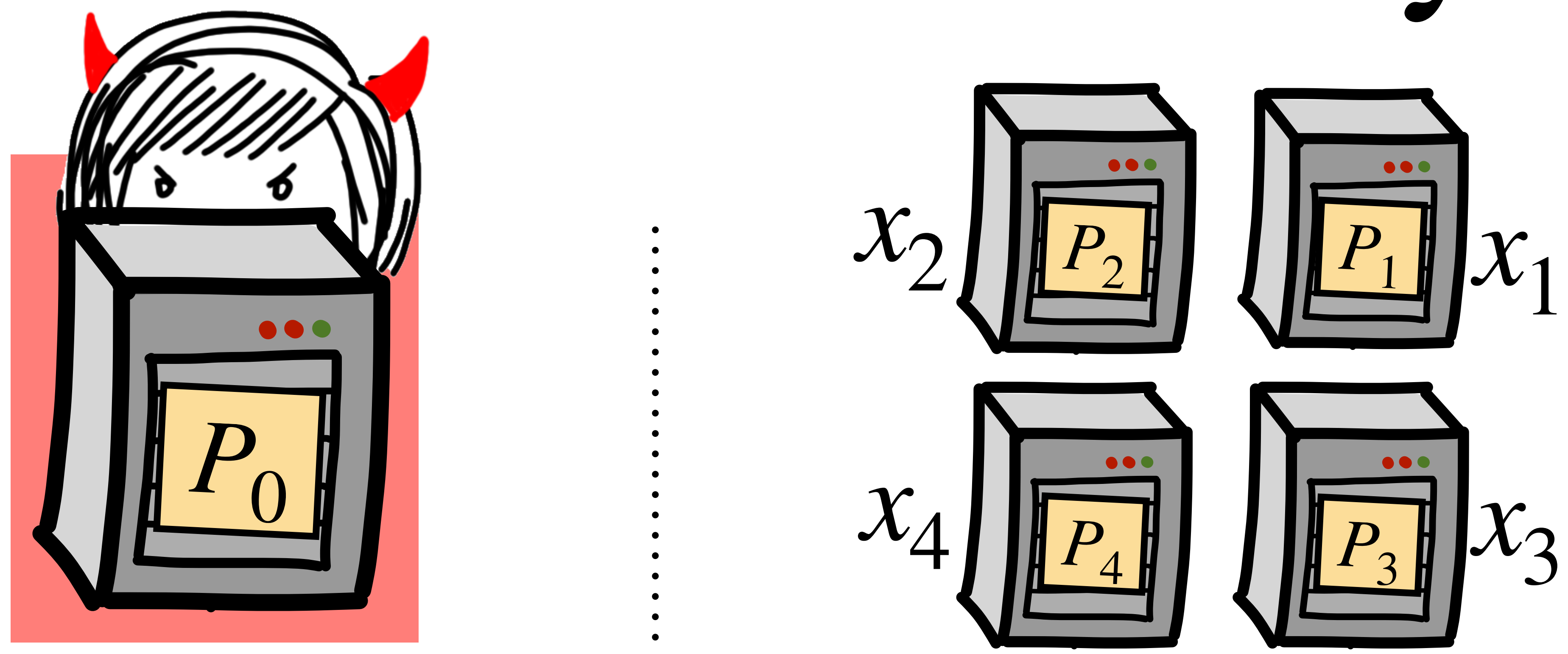


Privacy

x_1, x_2, x_3, x_4 always protected

	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output DoS-resistant	$y = f(\vec{x})$	$y = f(\vec{x})$

Grades of Active Security



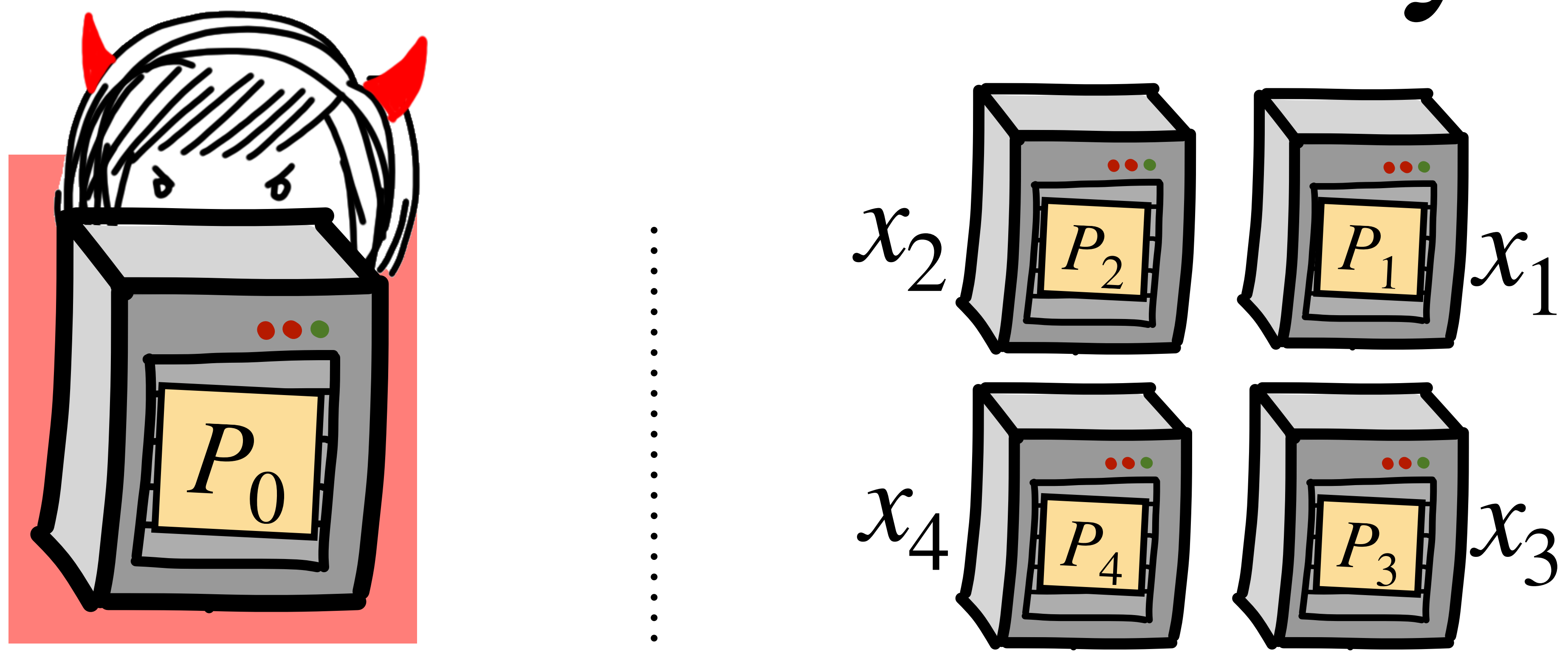
	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Identifiable Abort	$y = f(\vec{x})$	! CHEATER FOUND ! P_0 = 😈
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output	$y = f(\vec{x})$	$y = f(\vec{x})$

Privacy

x_1, x_2, x_3, x_4 always protected

DoS-resistant

Grades of Active Security



	OUTPUT	
Security with Abort	$y = f(\vec{x})$	$\perp$
Identifiable Abort	$y = f(\vec{x})$	! CHEATER FOUND ! P_0 =
Fairness	Case 1 : $y = f(\vec{x})$	$y = f(\vec{x})$
	Case 2 : $\perp$	$\perp$
Guaranteed Output DoS-resistant	$y = f(\vec{x})$	$y = f(\vec{x})$

Privacy

x_1, x_2, x_3, x_4 always protected

What's the Tradeoff?

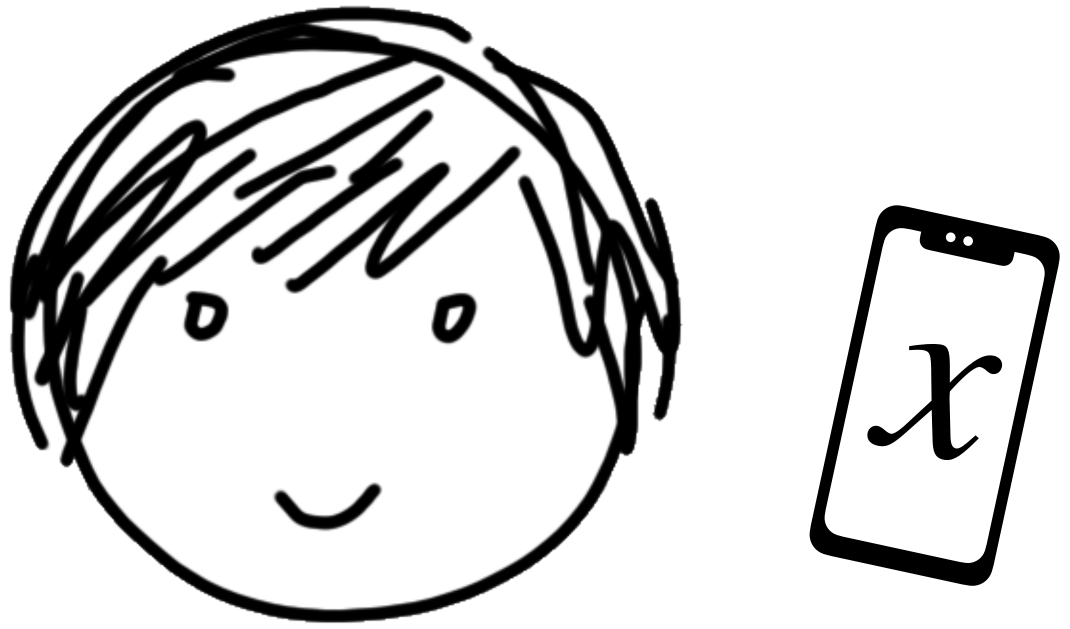
- Security with Abort and Identifiable Abort are feasible (under standard cryptographic assumptions) even if only one party is honest [GMW87]
  a.k.a. $t < n$ setting

- Fairness and Guaranteed Output for general functions are only feasible when a majority of parties are honest [Cleve86]

- For the same corruption threshold, *known constructions* for stronger security typically incur a substantial penalty in complexity/performance
  (not a tight statement)

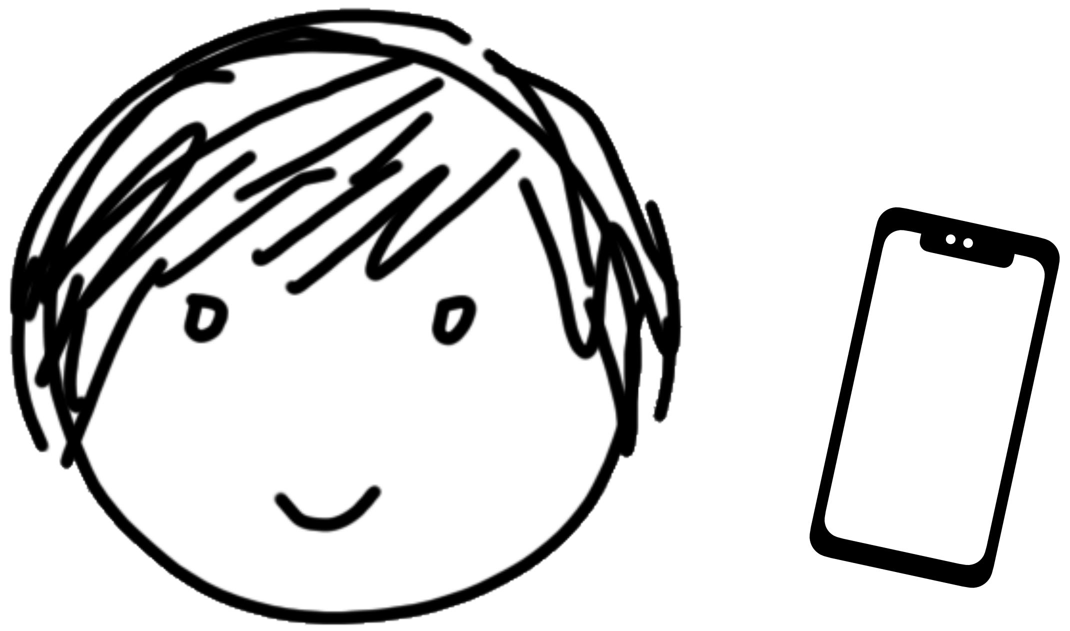
- IA typically studied as a "compromise" when GOD is infeasible

Guaranteed Output *vs.* Identifiable Abort



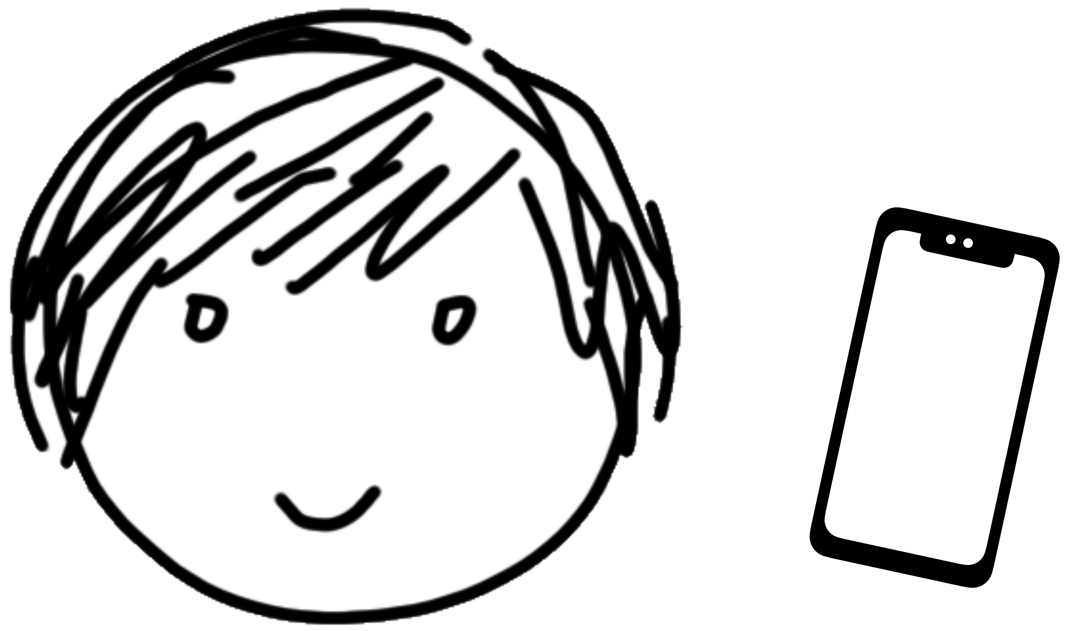
Guaranteed Output *vs.* Identifiable Abort

$\mathsf{SecretShare}(x) \mapsto$

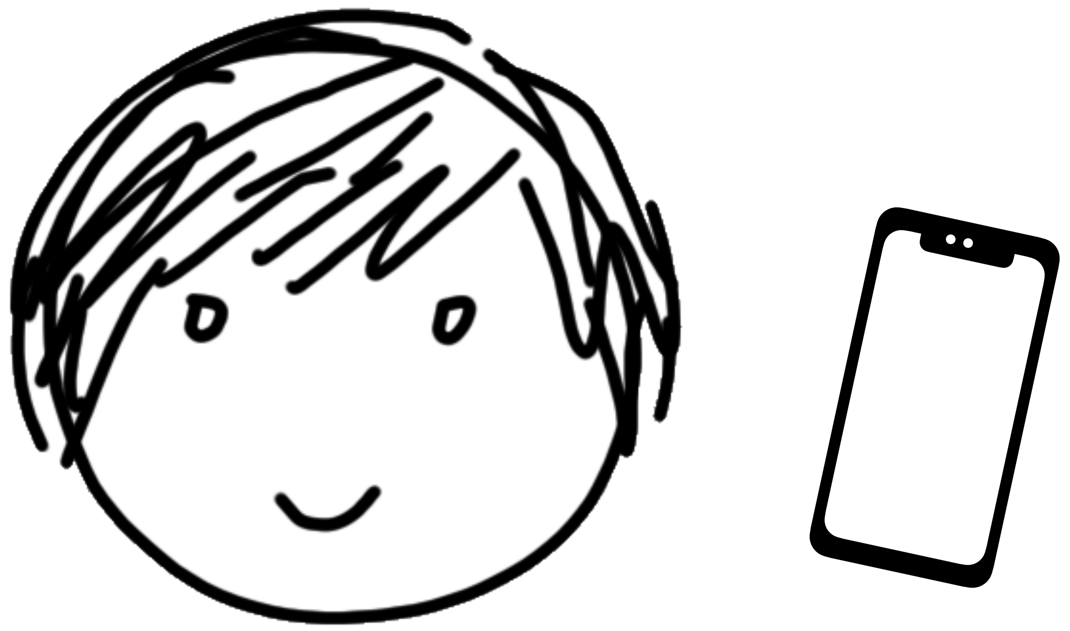


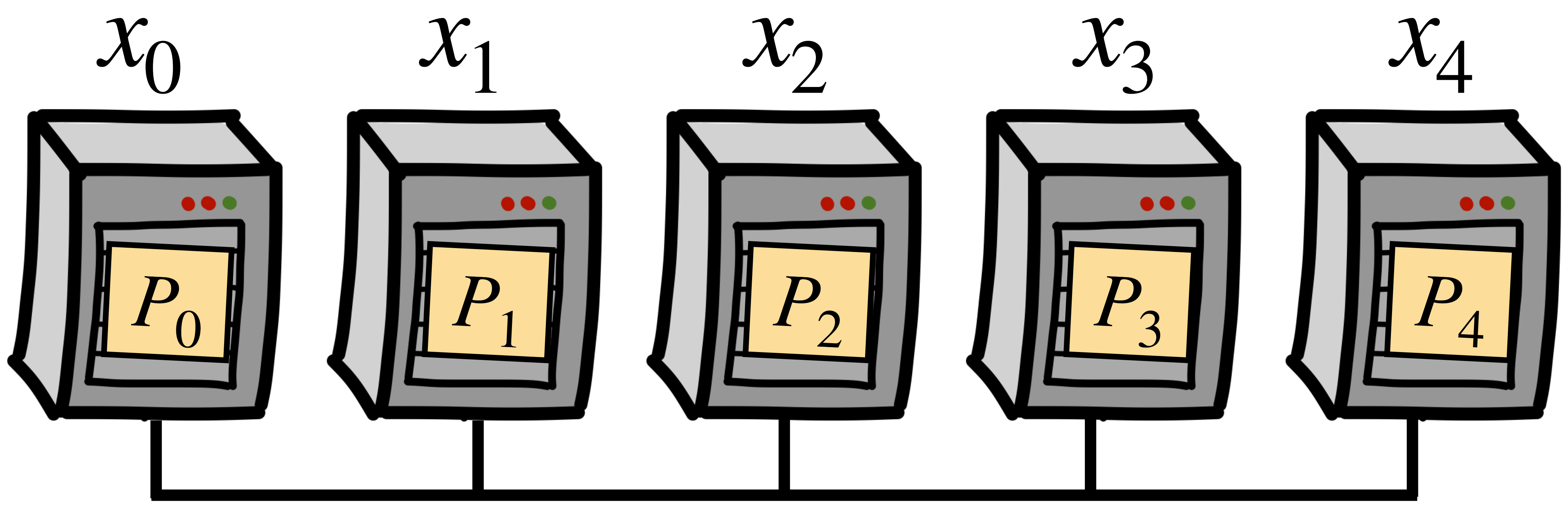
Guaranteed Output *vs.* Identifiable Abort

$\mathsf{SecretShare}(x) \mapsto \quad x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4$

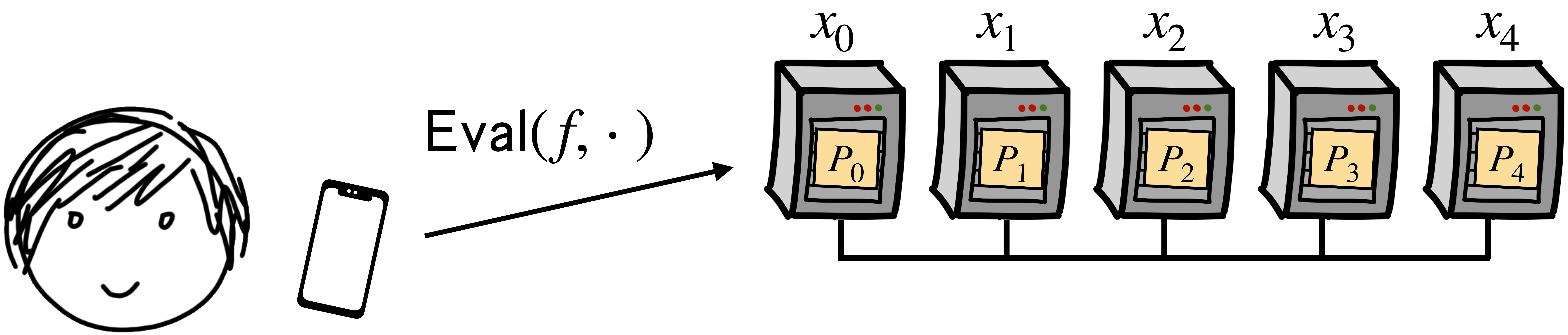


Guaranteed Output *vs.* Identifiable Abort

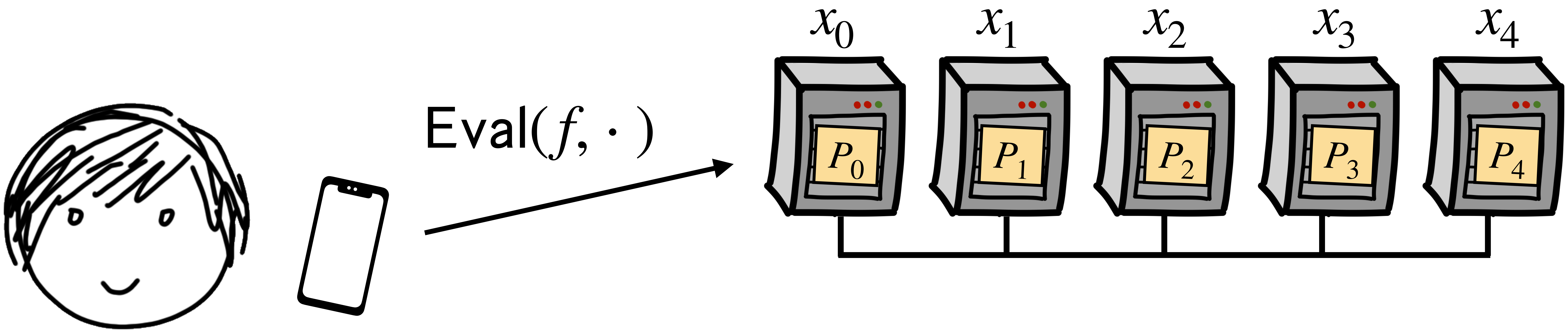




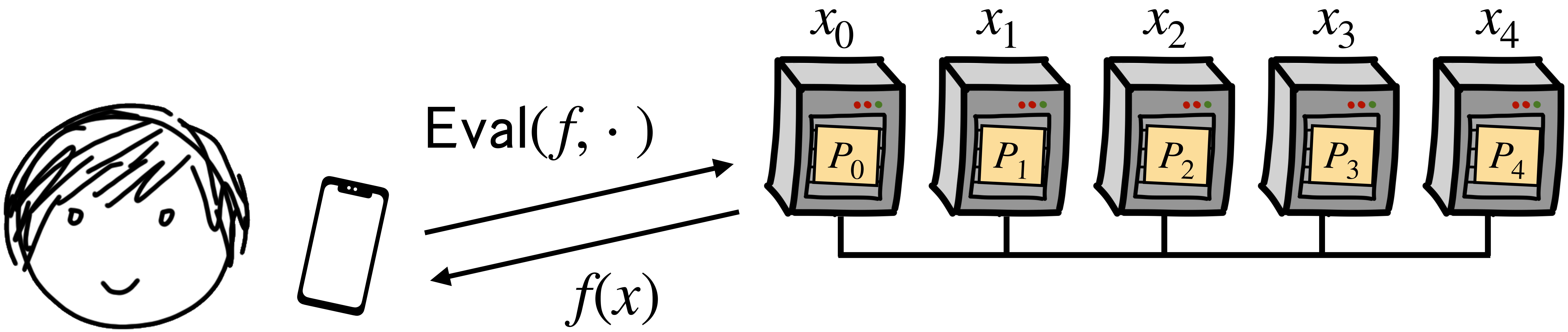
Guaranteed Output *vs.* Identifiable Abort



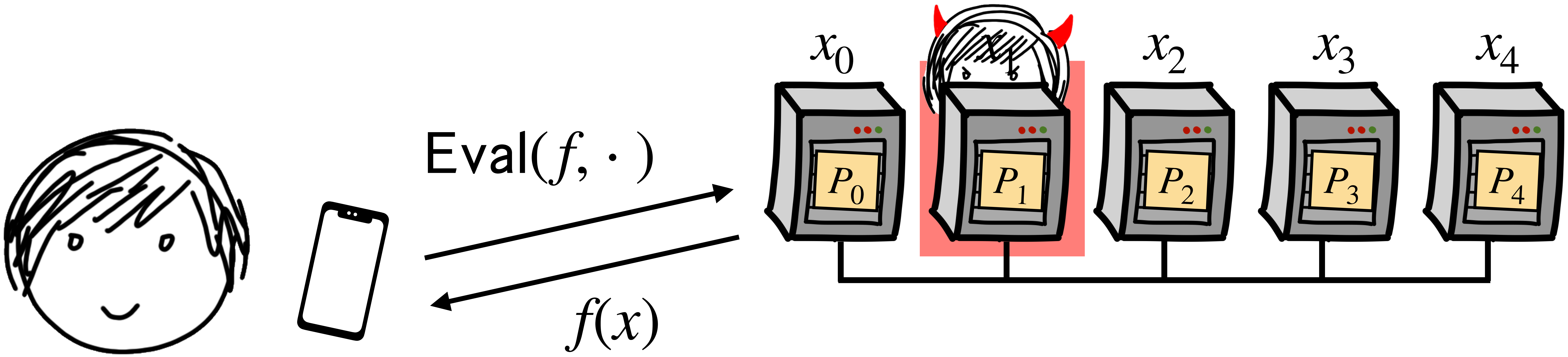
Guaranteed Output *vs.* Identifiable Abort



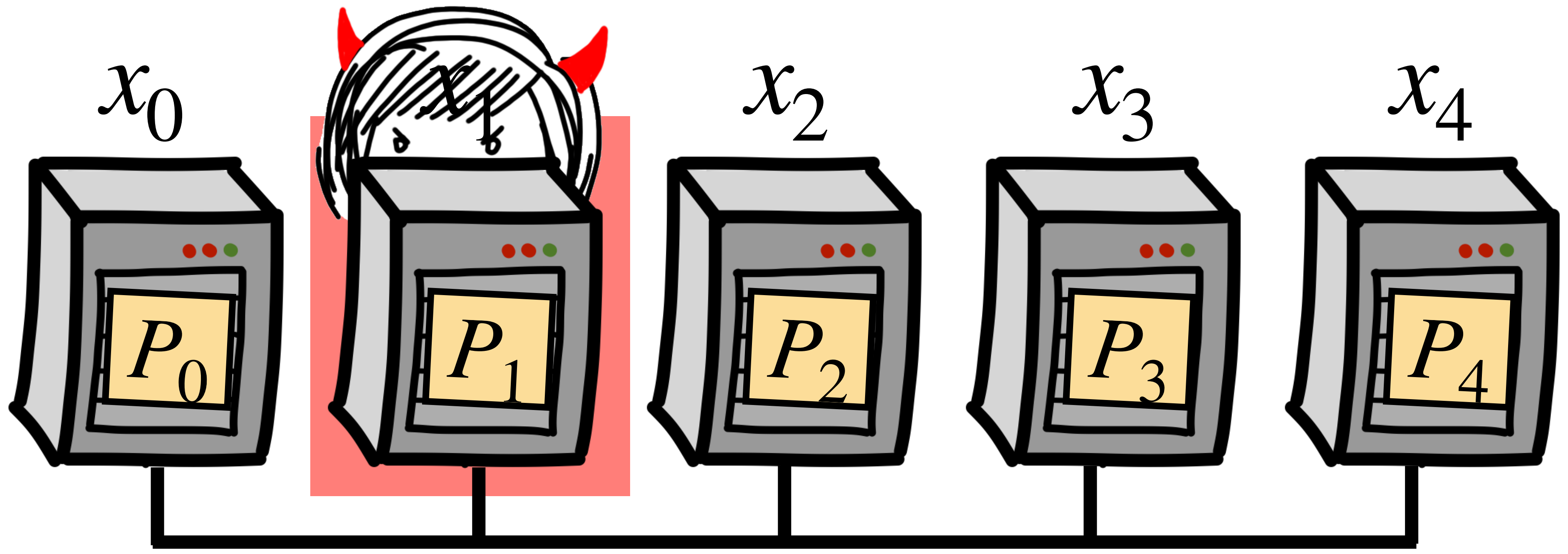
Guaranteed Output *vs.* Identifiable Abort



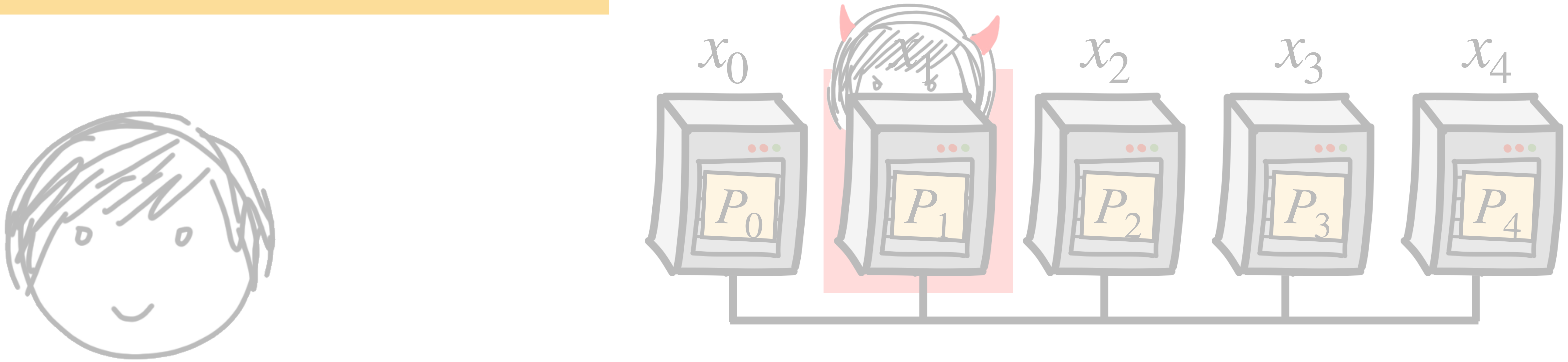
Guaranteed Output *vs.* Identifiable Abort



Guaranteed Output *vs.* Identifiable Abort

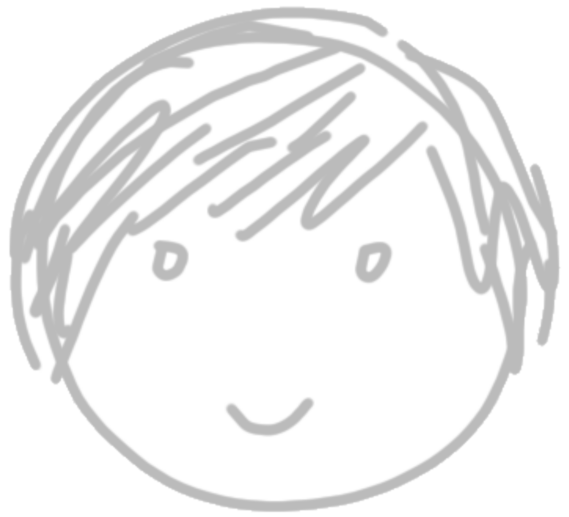


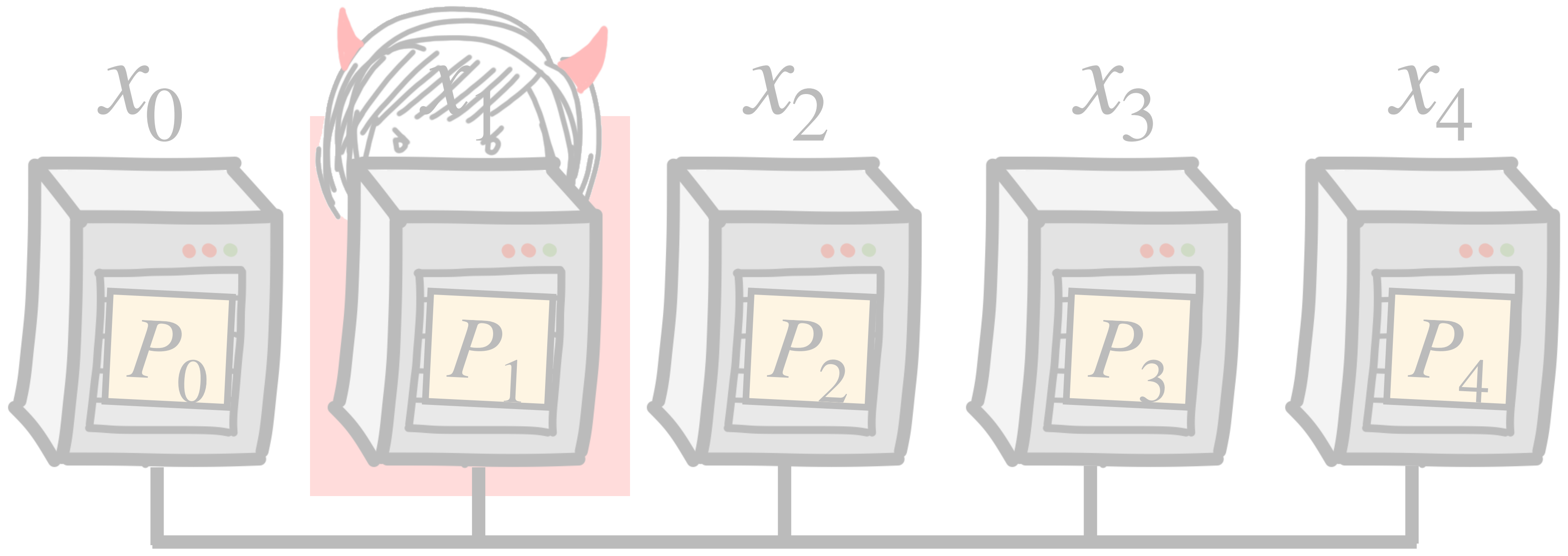
Guaranteed Output *vs.* Identifiable Abort

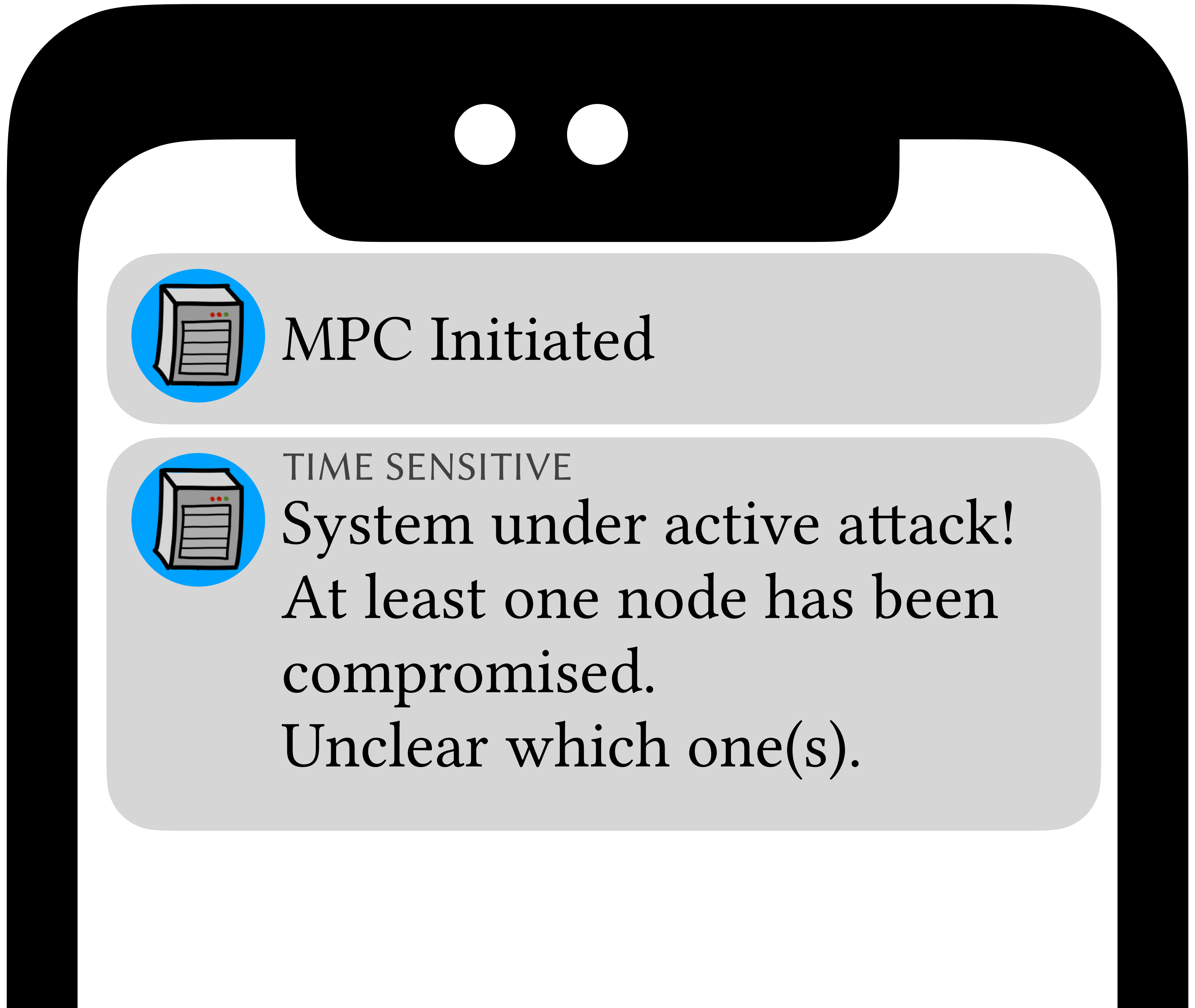


MPC Initiated

Guaranteed Output *vs.* Identifiable Abort



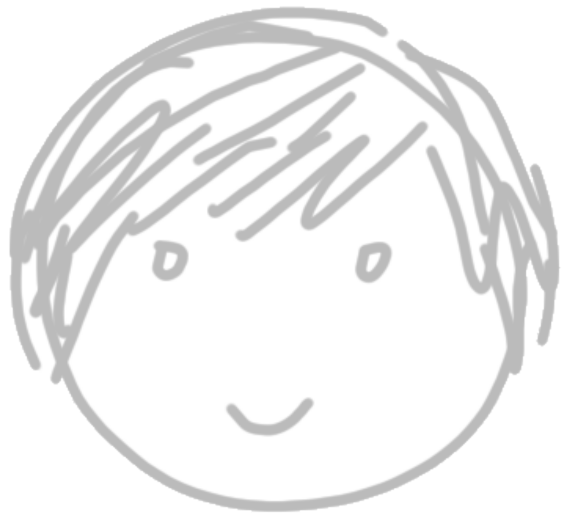


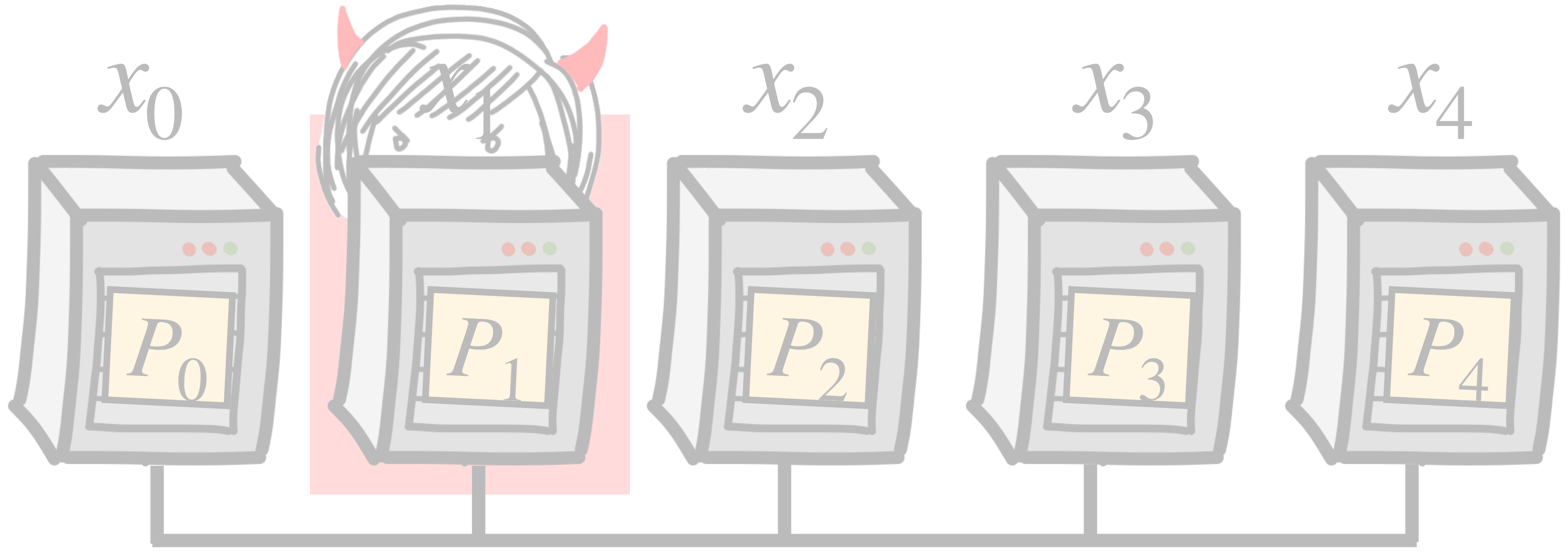


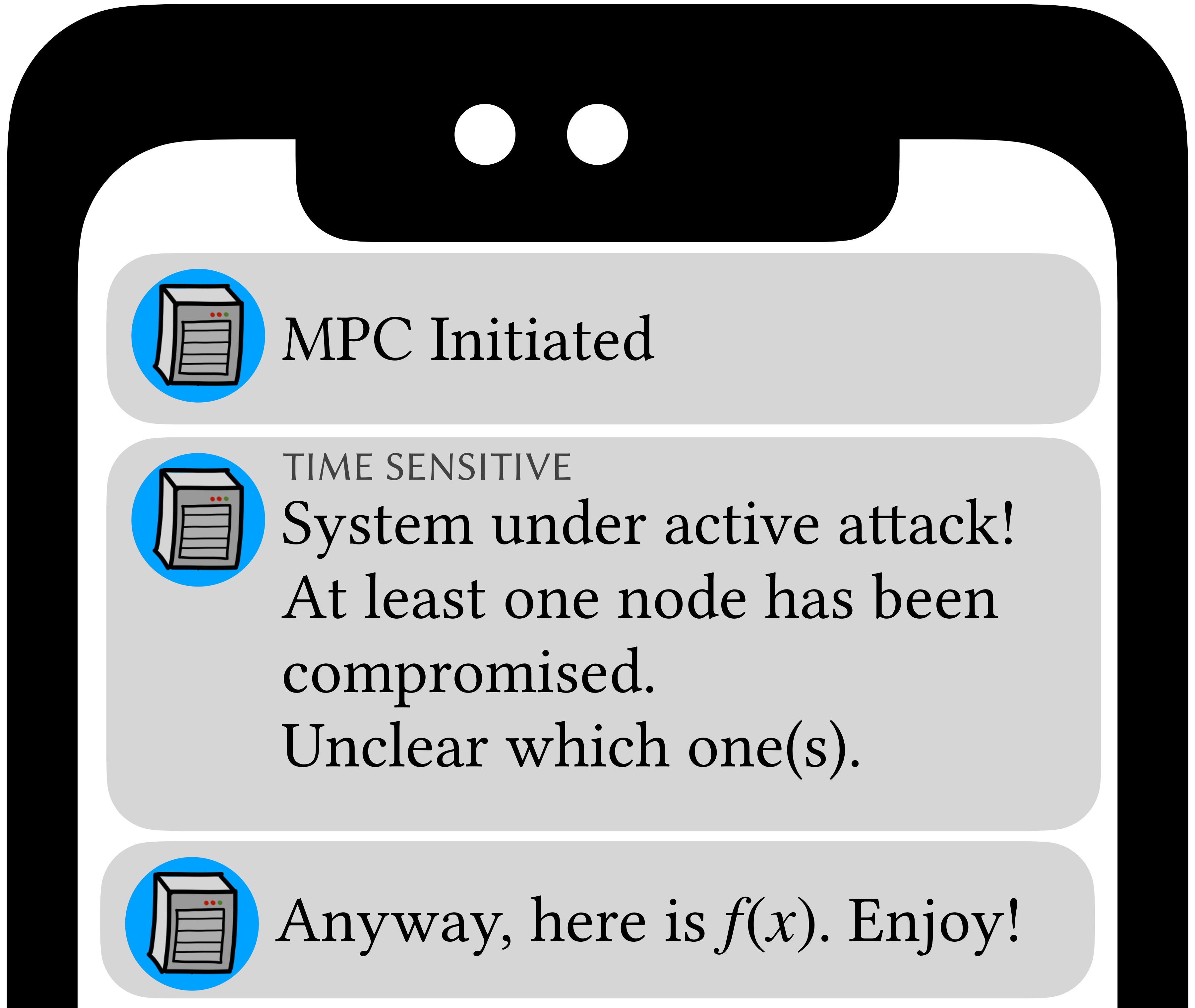
MPC Initiated

TIME SENSITIVE
System under active attack!
At least one node has been compromised.
Unclear which one(s).

Guaranteed Output *vs.* Identifiable Abort





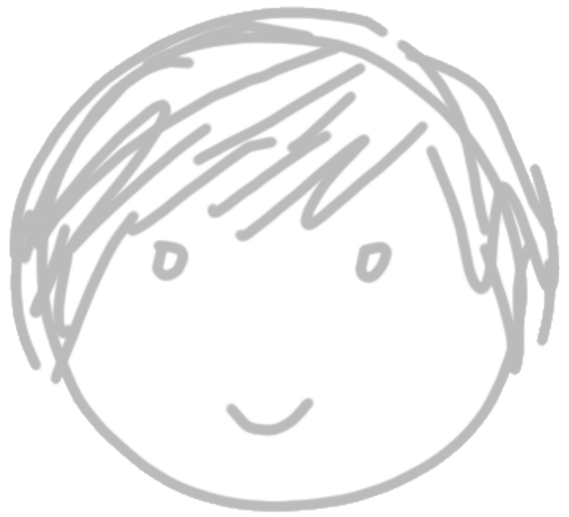


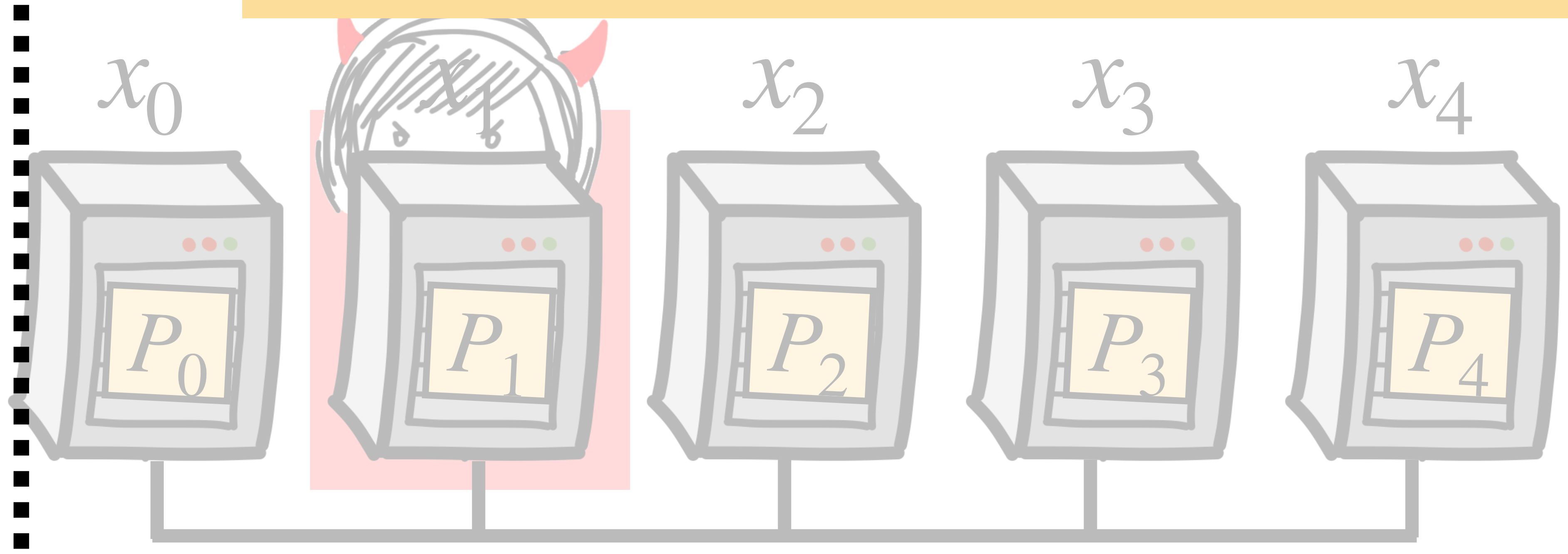
MPC Initiated

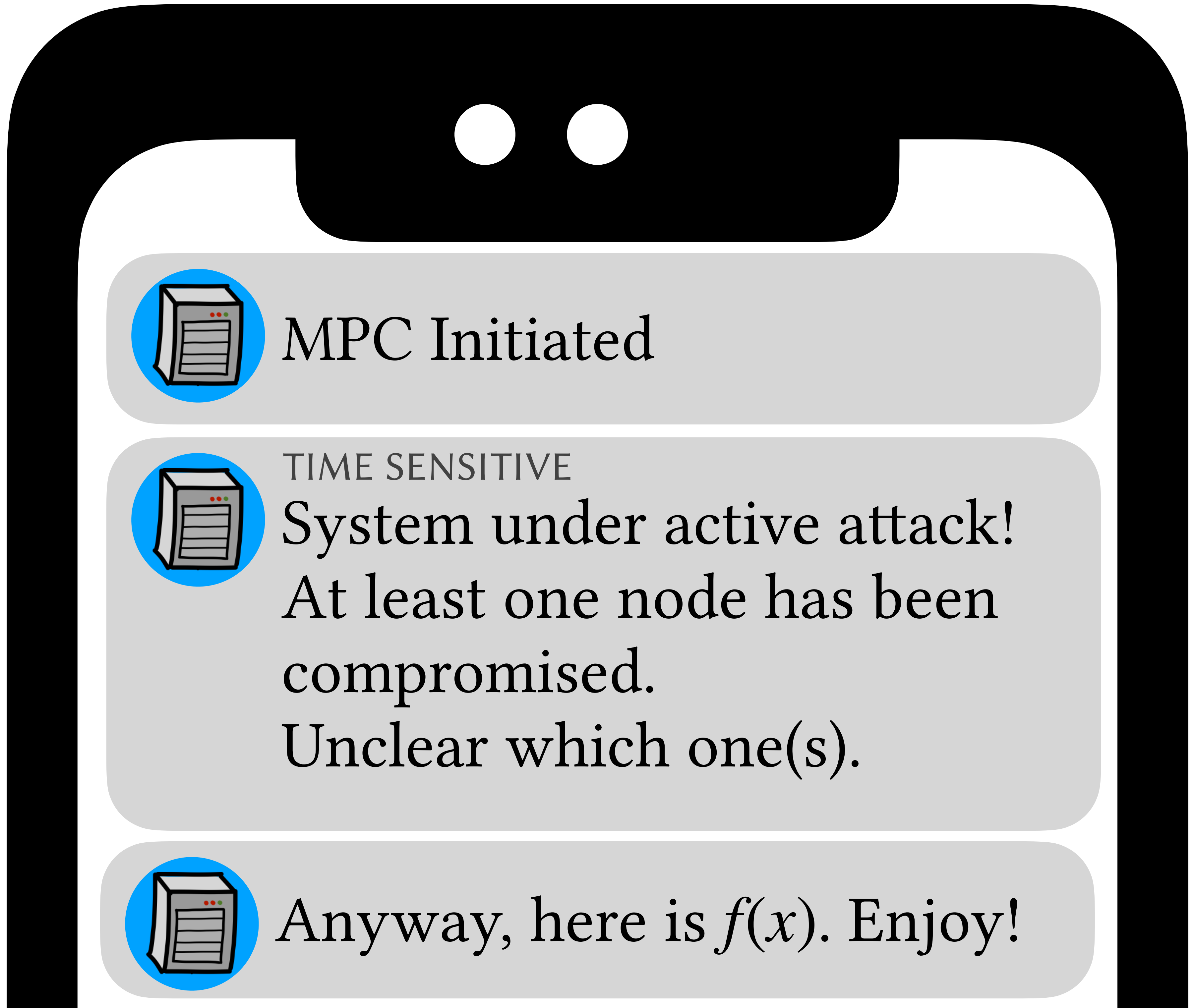
TIME SENSITIVE
System under active attack!
At least one node has been compromised.
Unclear which one(s).

Anyway, here is $f(x)$. Enjoy!

Guaranteed Output *vs.* Identifiable Abort



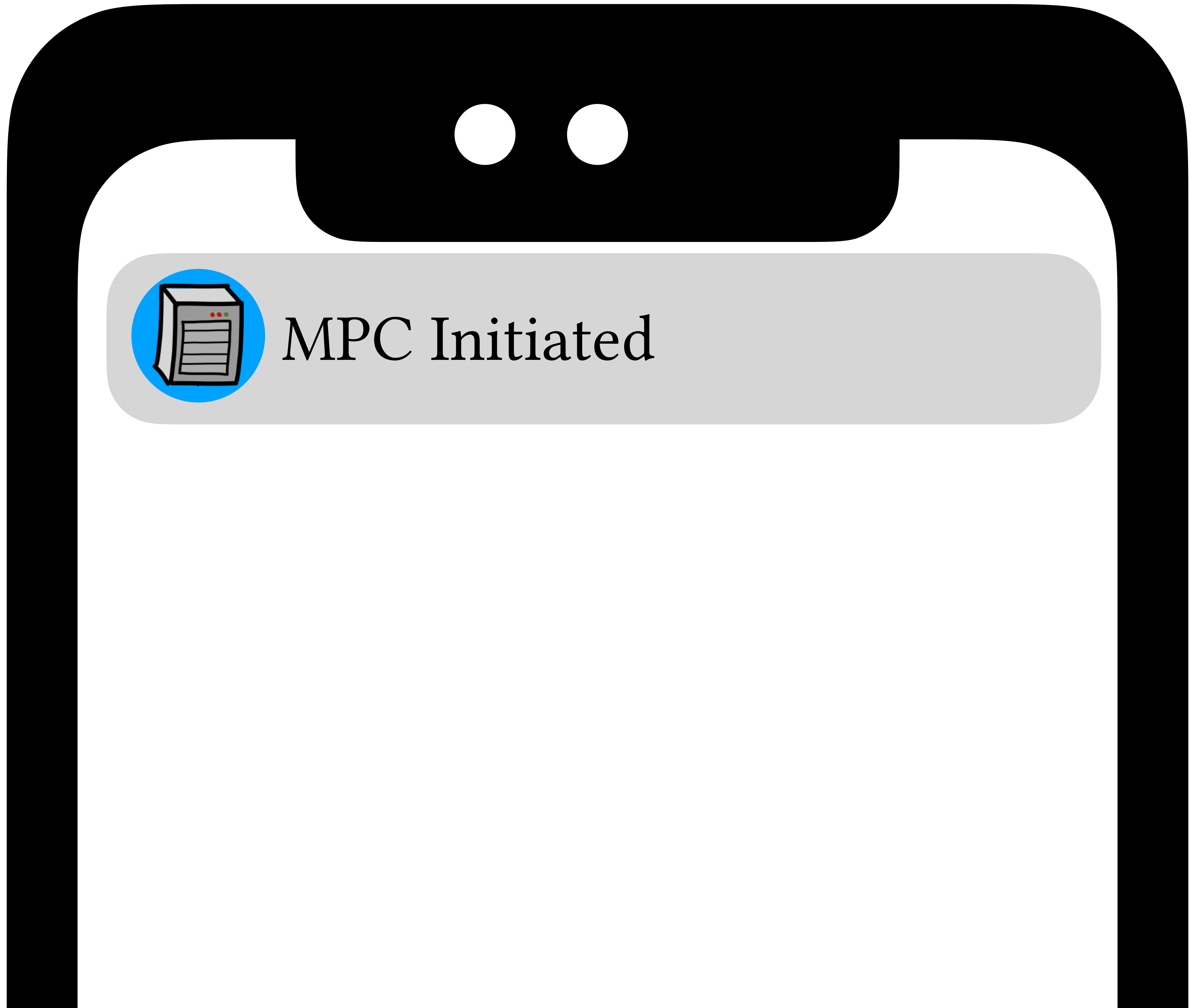




MPC Initiated

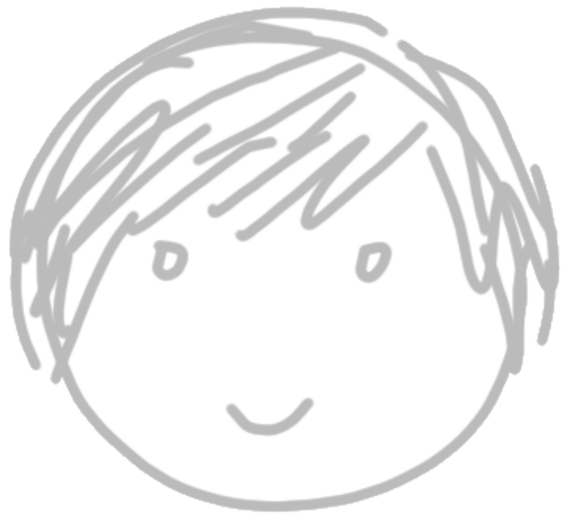
TIME SENSITIVE
System under active attack!
At least one node has been compromised.
Unclear which one(s).

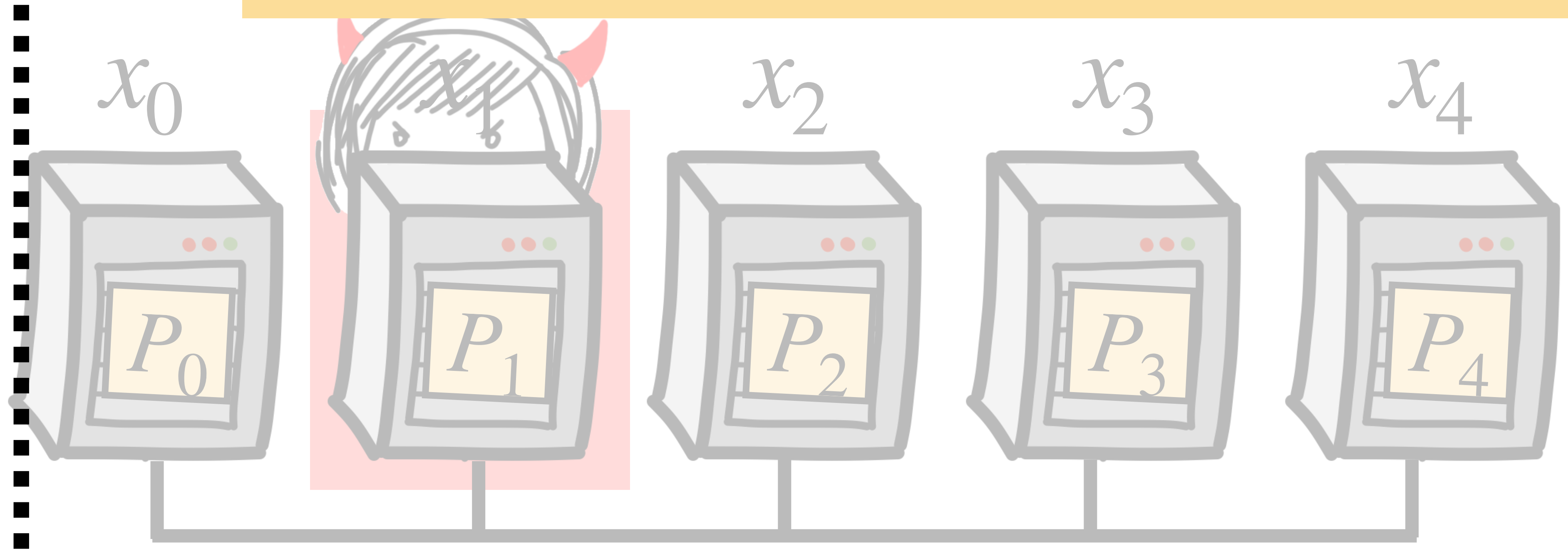
Anyway, here is $f(x)$. Enjoy!

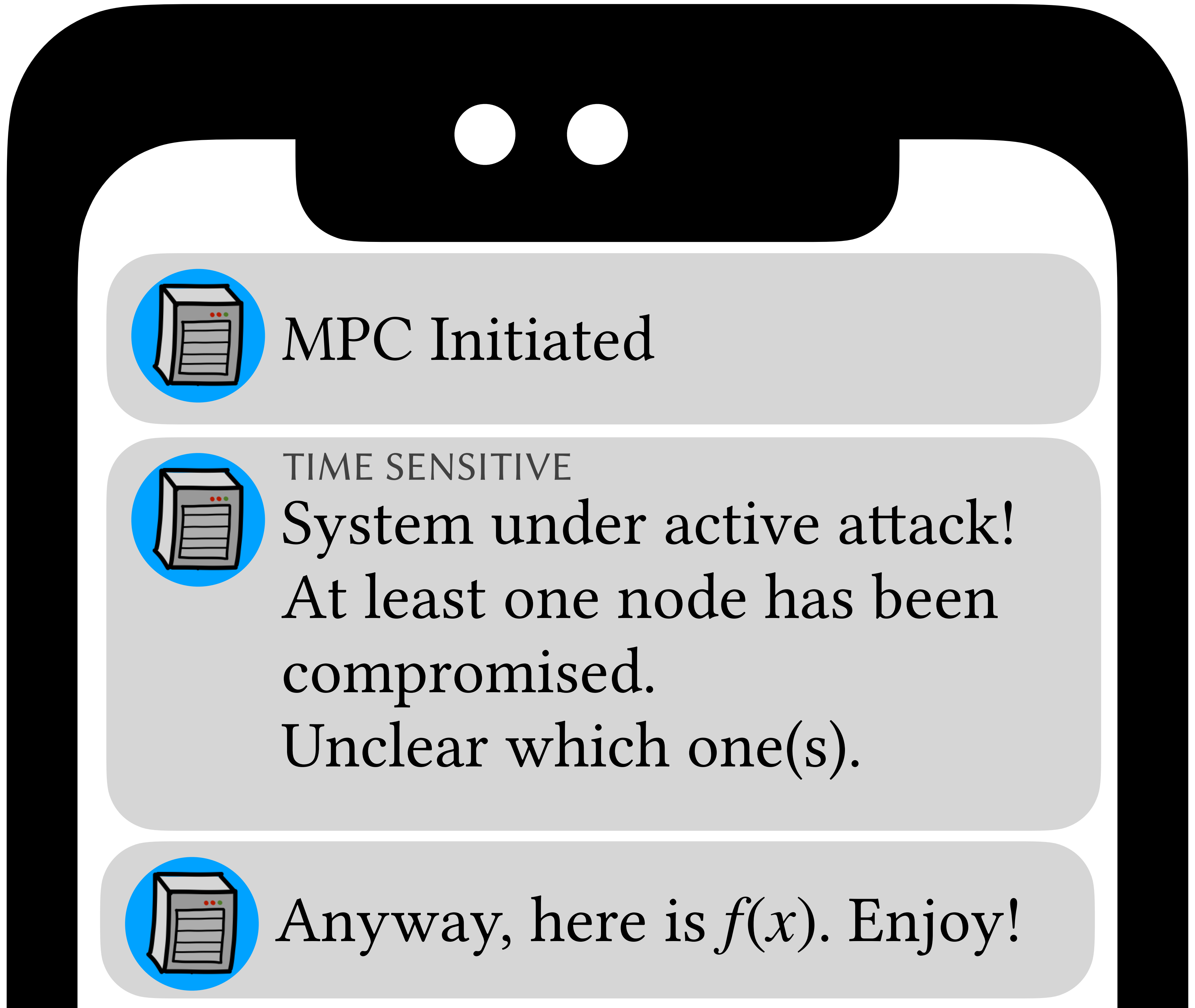


MPC Initiated

Guaranteed Output *vs.* Identifiable Abort



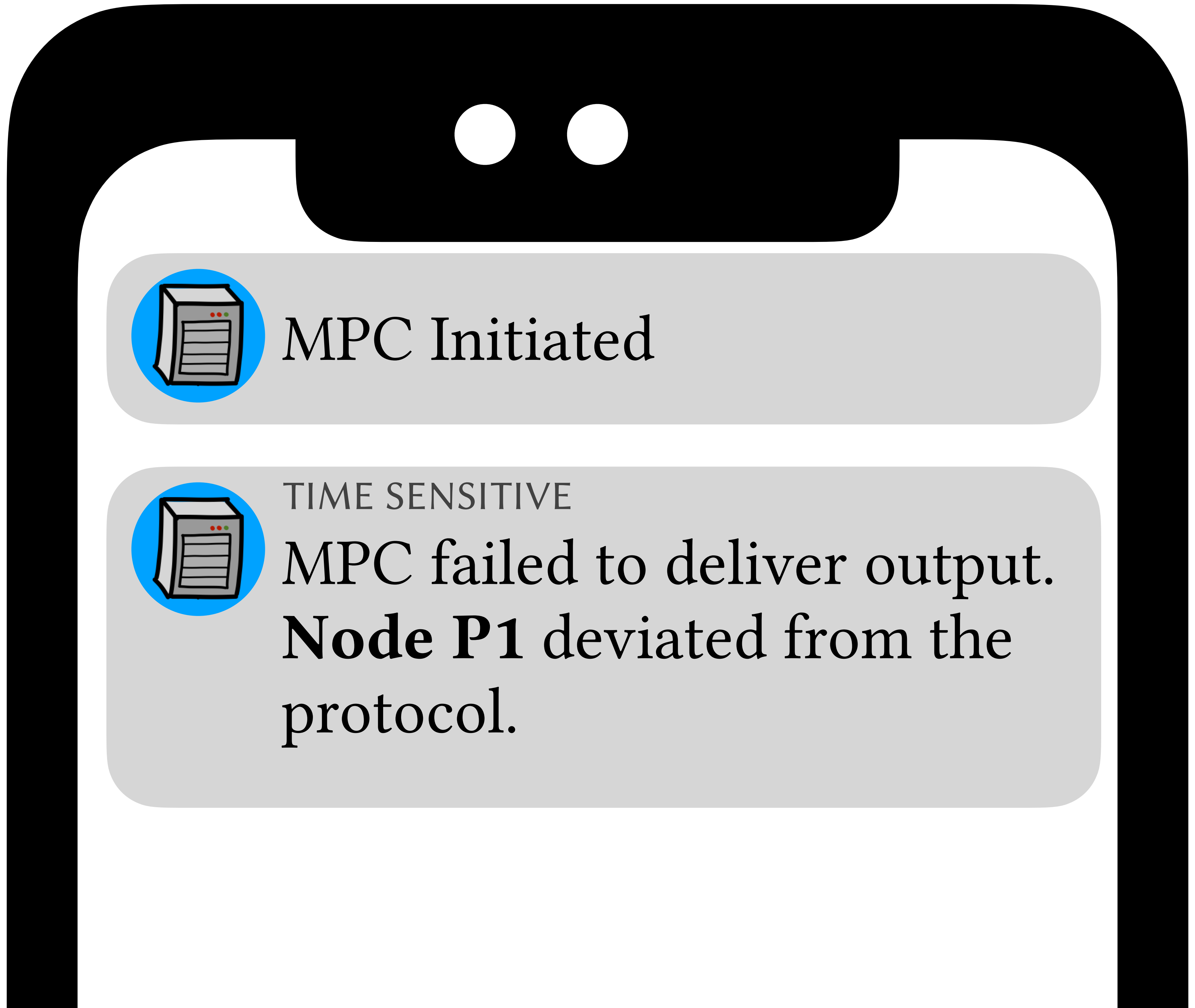




MPC Initiated

TIME SENSITIVE
System under active attack!
At least one node has been compromised.
Unclear which one(s).

Anyway, here is $f(x)$. Enjoy!



MPC Initiated

TIME SENSITIVE
MPC failed to deliver output.
Node P1 deviated from the protocol.

Practical Application: Re-staking

- Re-staking TLDR:
  - Operators buy into the protocol (service/AVS) with “re-staked” assets
  - In case of malicious behaviour, this stake can be “slashed”
  - Economic security: protocol deviations are disincentivized

- Identifiable Abort is a natural fit for this setting
  - Cheating parties can be identified and slashed
  - DoS resistant MPC via economic incentives

- **Hope**: complexity of IA closer to Sec w. Abort than Guaranteed Output Delivery

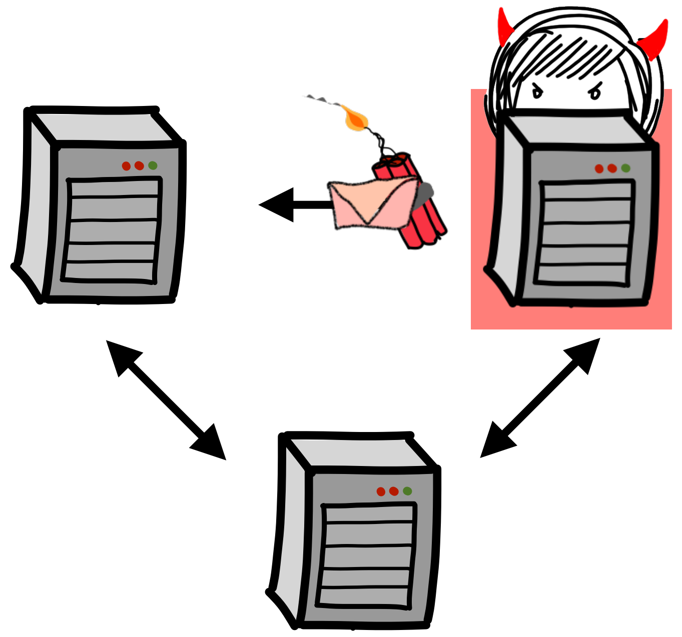
Identification Mechanisms

- Cheater *could* be found through out of band methods.

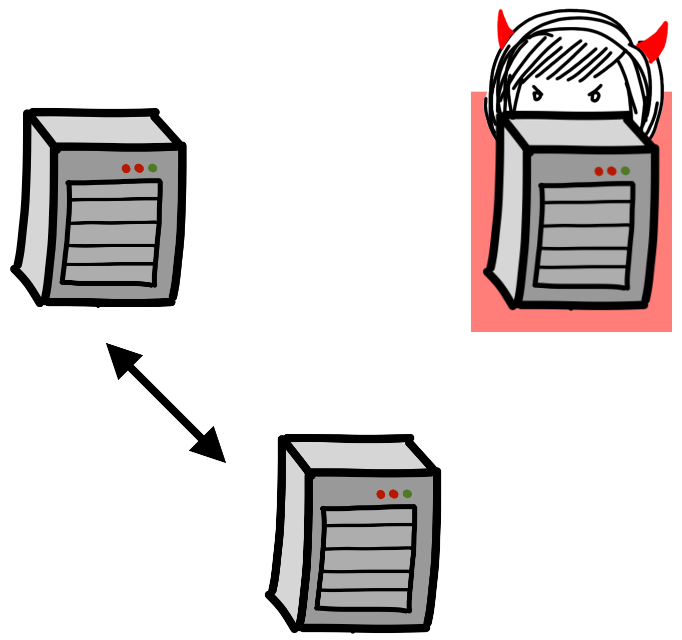
- We want **certifiable** protocol mechanism to identify who crashed the protocol
  ⇒ each party either gets output, or identity of cheating party + cert. of cheat

  Note: no consensus on identity

- Two ways to crash protocol:



1. Malformed protocol message

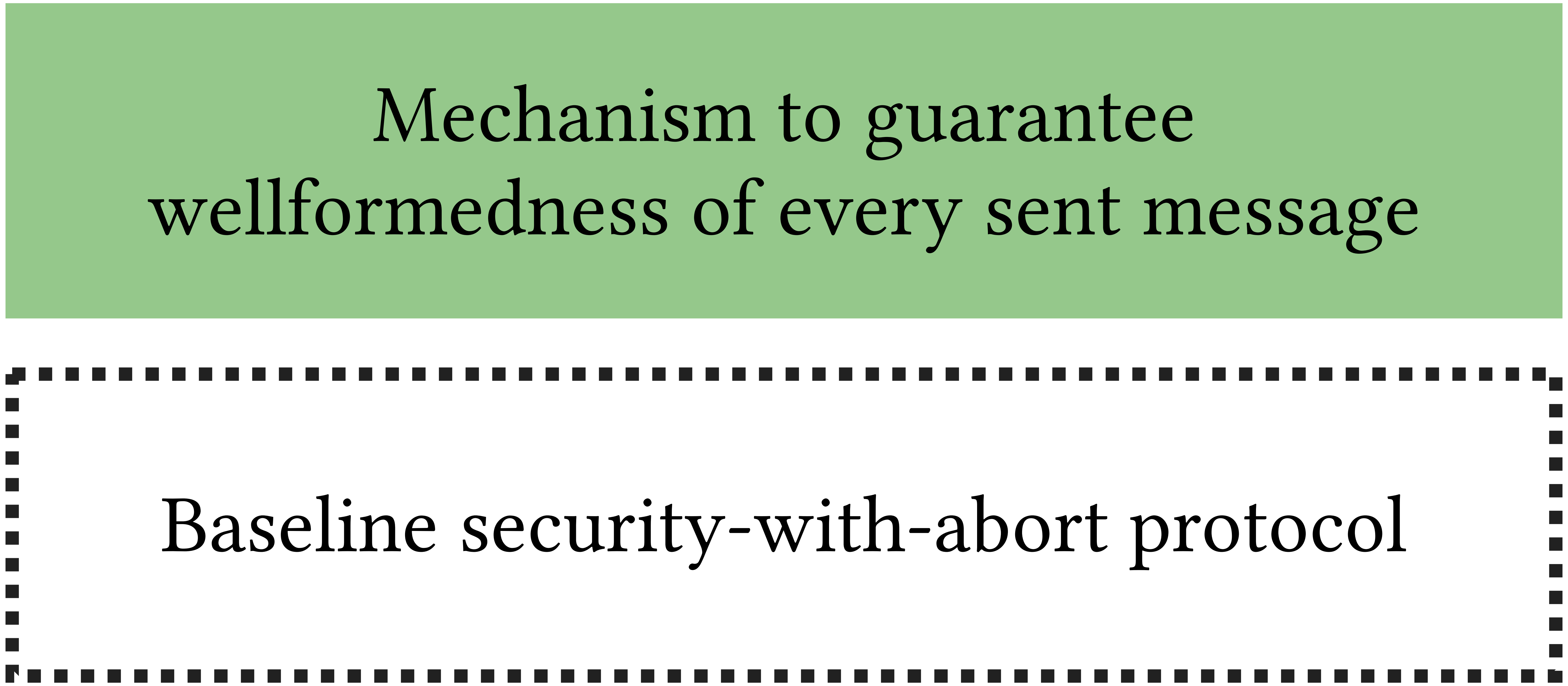


2. No message at all

Anatomy of MPC-IA

Baseline security-with-abort protocol

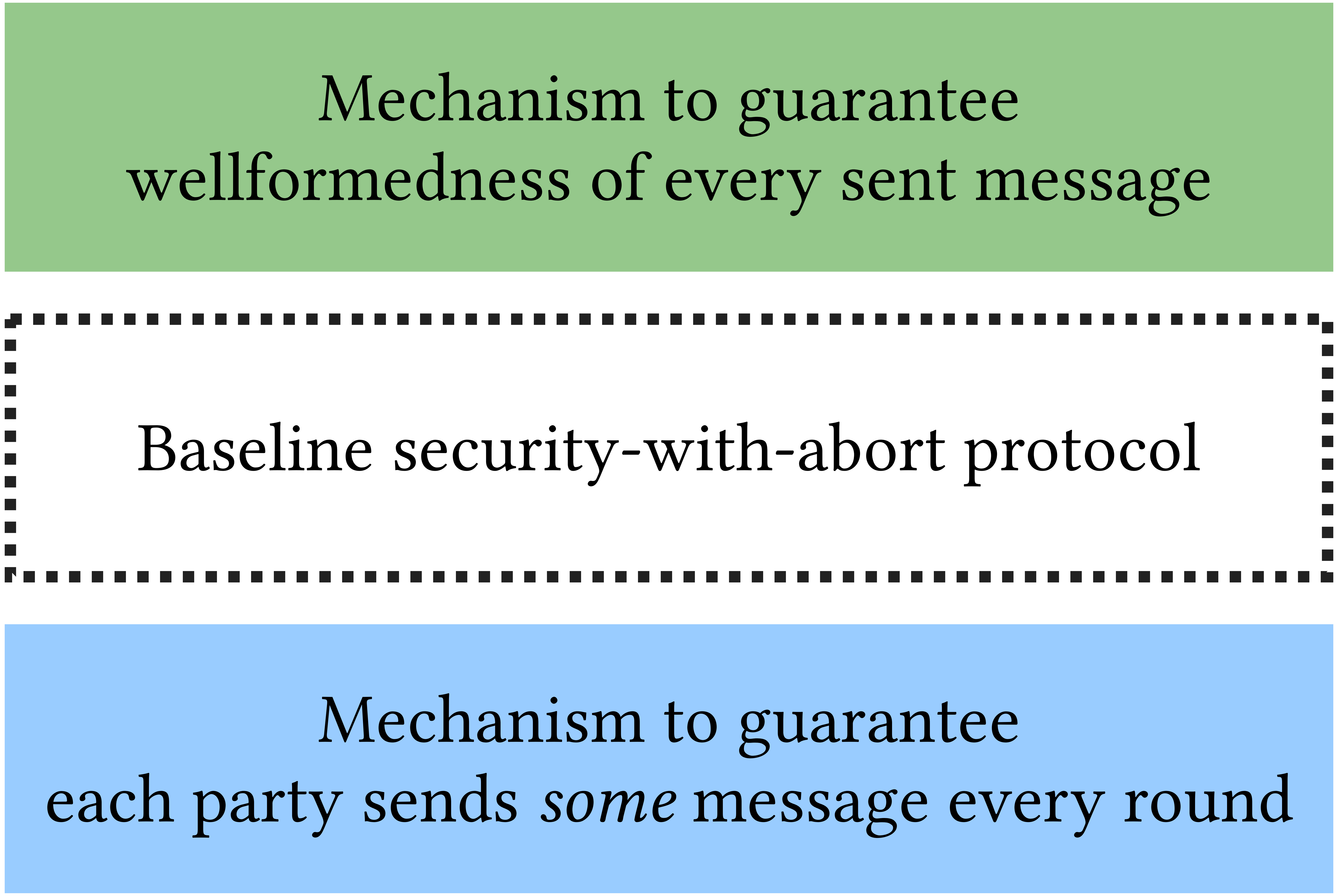
Anatomy of MPC-IA



Mechanism to guarantee wellformedness of every sent message

Baseline security-with-abort protocol

Anatomy of MPC-IA

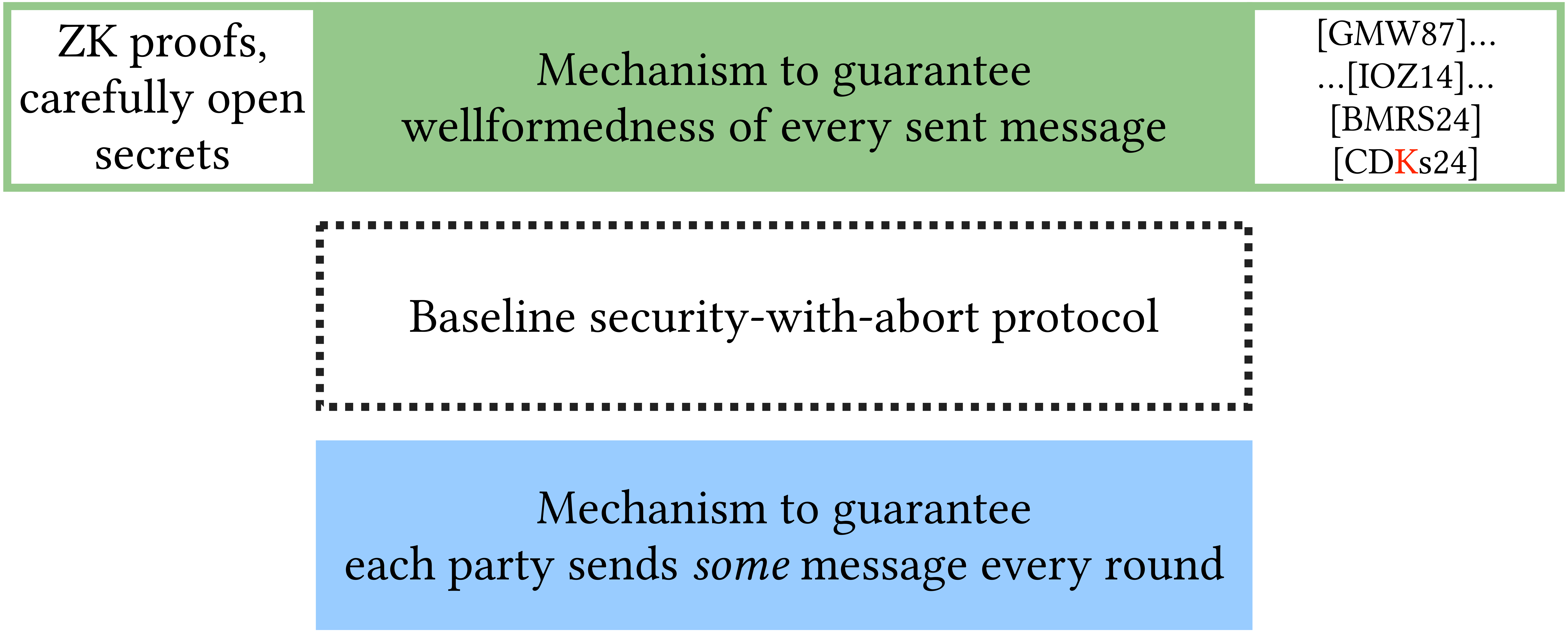


Mechanism to guarantee wellformedness of every sent message

Baseline security-with-abort protocol

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

Anatomy of MPC-IA



ZK proofs, carefully open secrets

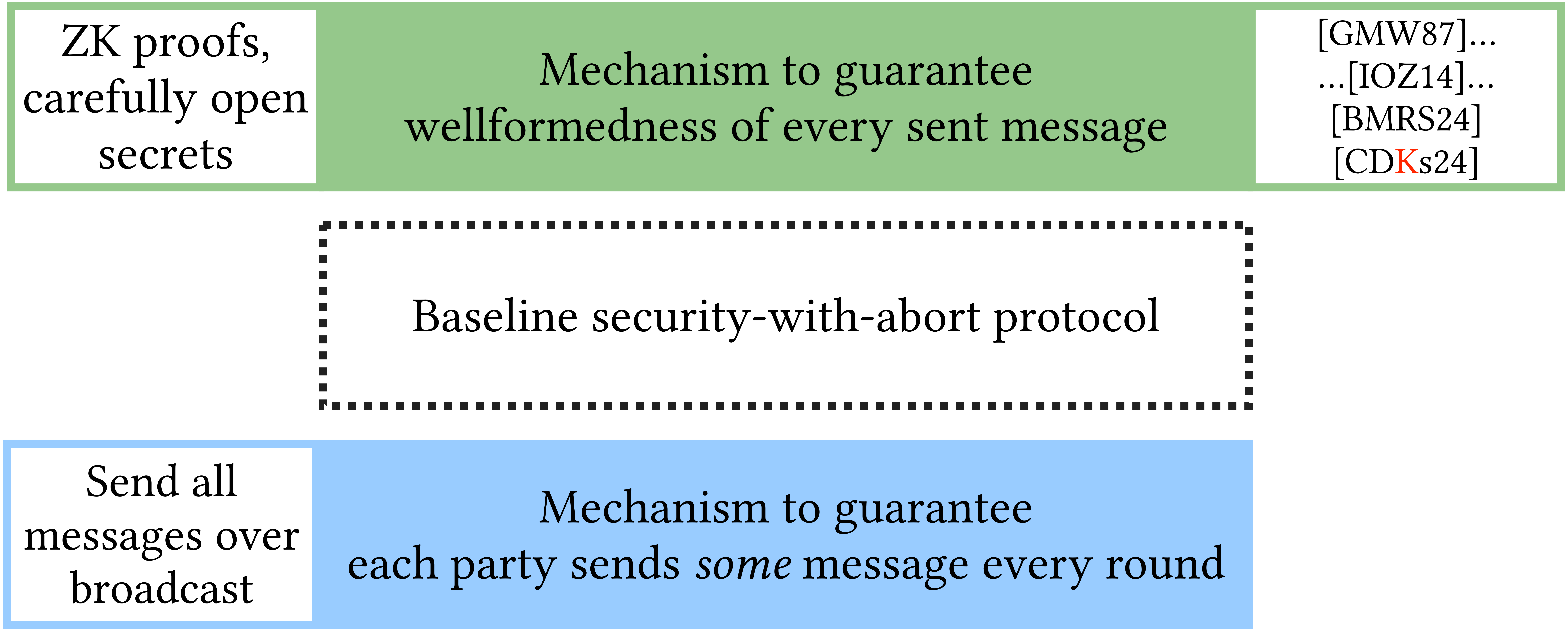
Mechanism to guarantee wellformedness of every sent message

[GMW87]…
…[IOZ14]…
[BMRS24]
[CDKs24]

Baseline security-with-abort protocol

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

Anatomy of MPC-IA



ZK proofs, carefully open secrets

Mechanism to guarantee wellformedness of every sent message

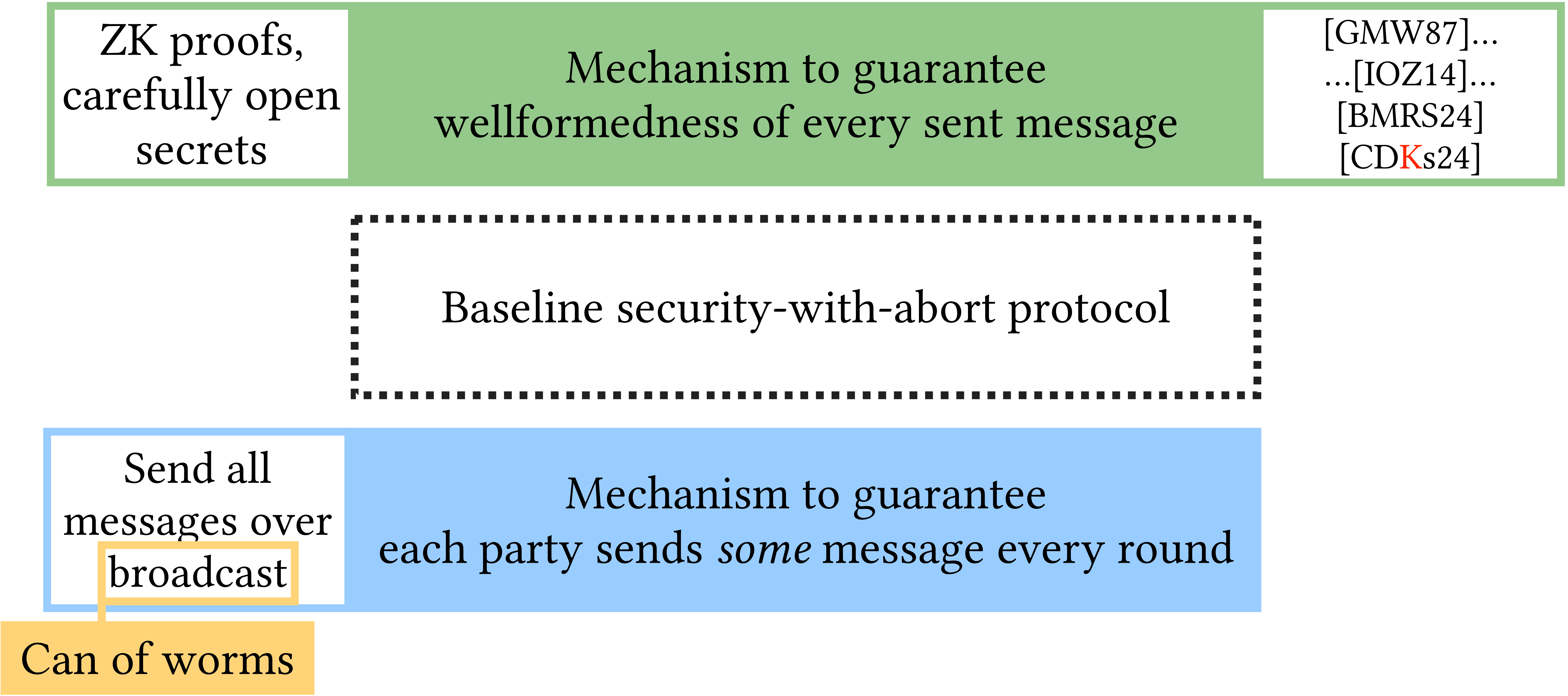
[GMW87]…
…[IOZ14]…
[BMRS24]
[CDKs24]

Baseline security-with-abort protocol

Send all messages over broadcast

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

Anatomy of MPC-IA



ZK proofs, carefully open secrets

Mechanism to guarantee wellformedness of every sent message

[GMW87]…
…[IOZ14]…
[BMRS24]
[CDKs24]

Baseline security-with-abort protocol

Send all messages over broadcast

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

Can of worms

“Broadcast”?

- Engineering Anecdata:
  “Do I really need to implement broadcast?”
  “*yes*”
  “Is it just for some theoretical proof nonsense?”
  “*no, it’s to catch parties that don’t send messages for example*”
  “That seems unnecessary, I can just <insert heuristic>”

- In some settings [Lin22]: coordinator routes all messages
  ⇒ reasonable in sec. w. abort. setting, very strong assumption for IA

- Other settings [GMPS21, GKM+22, ZYP23]: use a blockchain
  ⇒ expensive, slow, introduces external dependencies

Broadcast Protocols

- [Cohen Lindell 14] MPC-IA implies broadcast: compute $\mathcal{F}_{\mathsf{PKI}}$ with IA

- Assuming PKI (+synchrony), broadcast is *feasible* [Dolev Strong 83]
  …but **round complexity is an issue**: $O(t)$ deterministic, or expected $O(1)$ randomized with large constants
  [Katz Koo 06][Abraham Devadas Dolev Nayak Ren 19]

- This is straightforward in the security with abort setting, via simple echo broadcast [Goldwasser Lindell 02]

- Can we construct a simple instantiation of BC as suitable for IA?
  Goal: MPC-IA protocols that are easy to deploy over p2p channels

BC-IA Properties

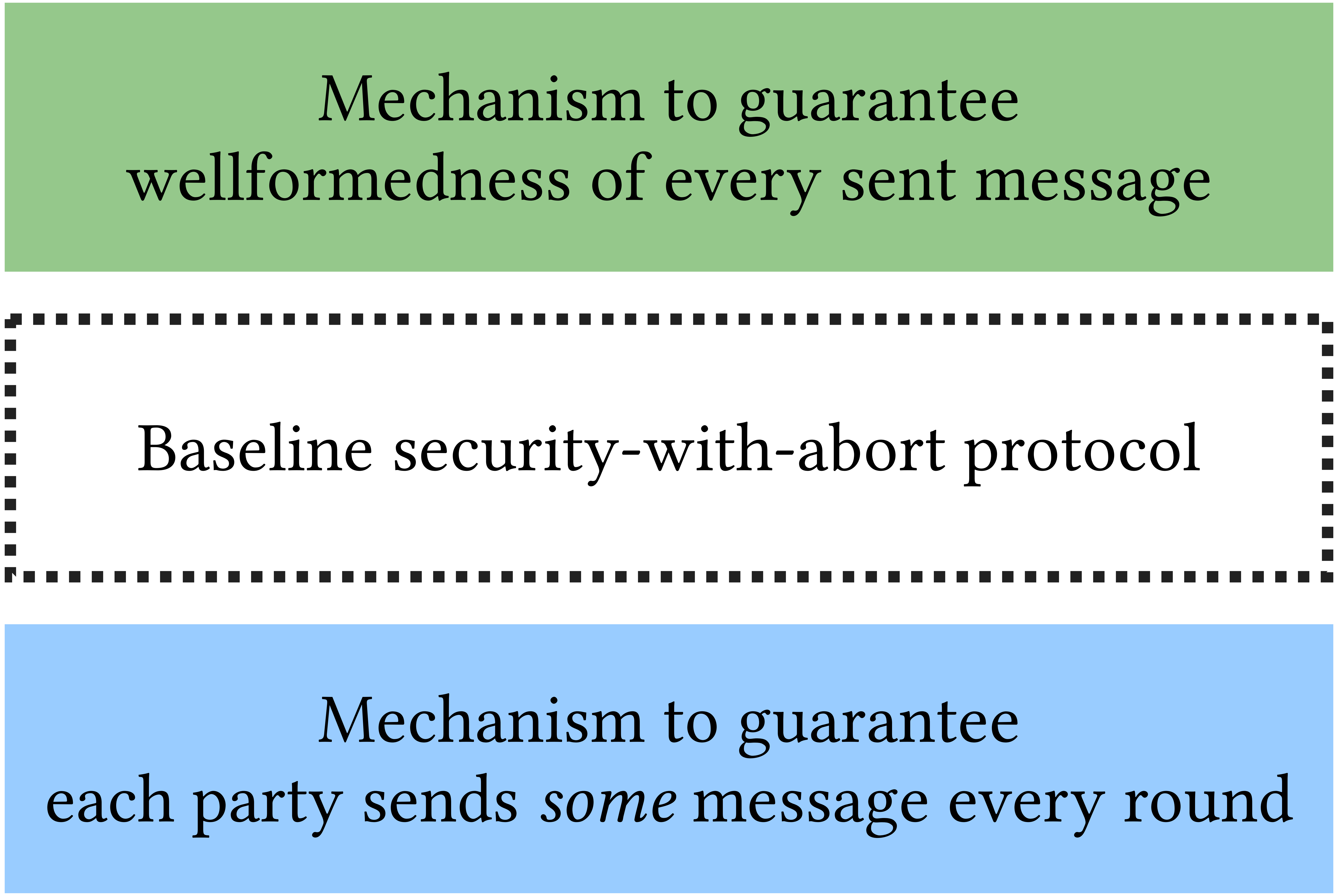
- **Consistency**: All honest parties that output a valid (dealer signed) message will be in agreement

- If the sender is corrupt, an honest party alternatively 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 misbehaviour vs. potential network fault—different penalties

- **Defamation-freeness**: Honest party can’t be framed with Ω or ω

Anatomy of MPC-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-IA

Mechanism to guarantee wellformedness of every sent message

Baseline security-with-abort protocol

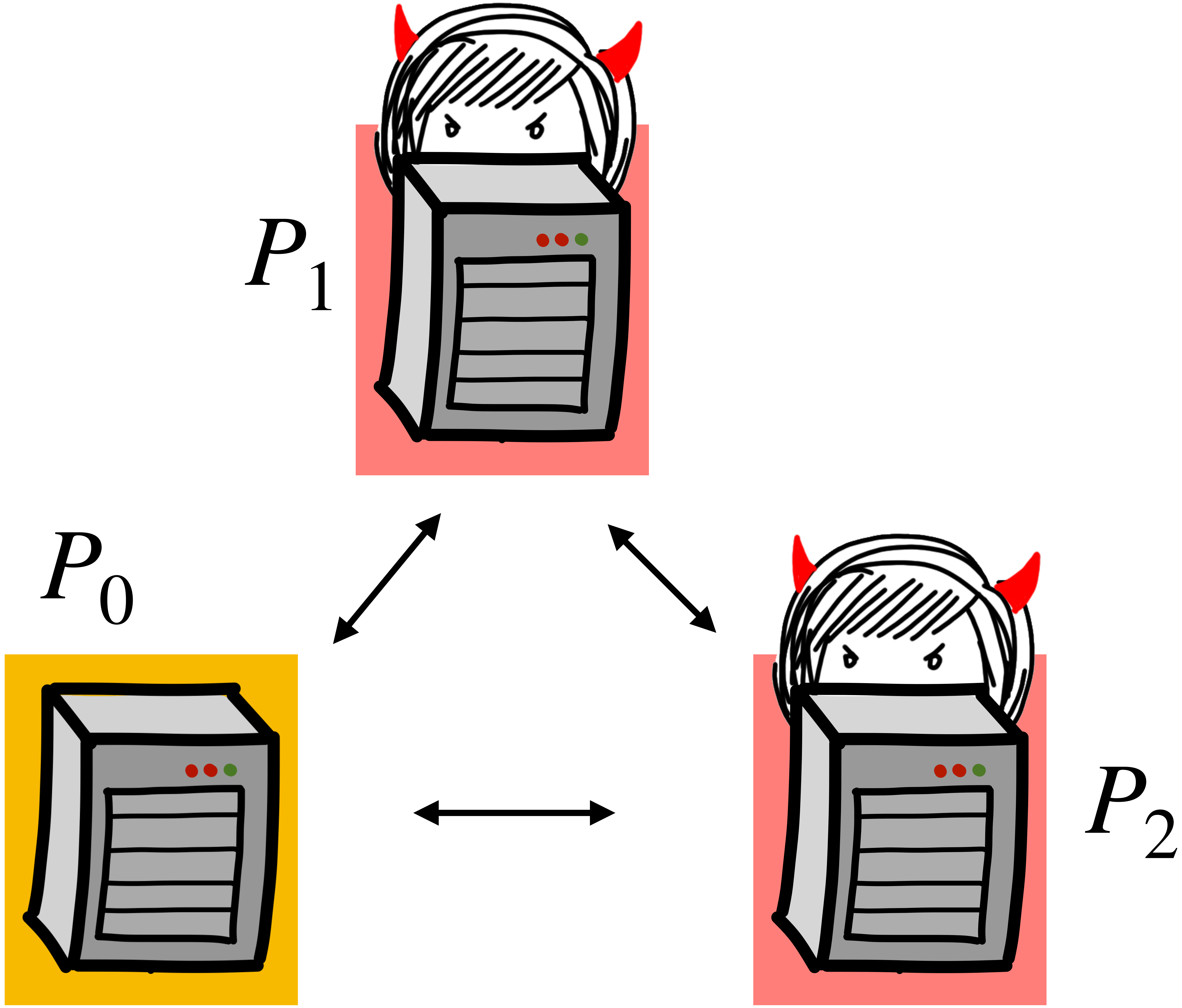
Mechanism to guarantee each party sends *some* message every round

This work: define “Broadcast-IA”

- Impossible w. dishonest majority
- 2-round honest-majority protocol

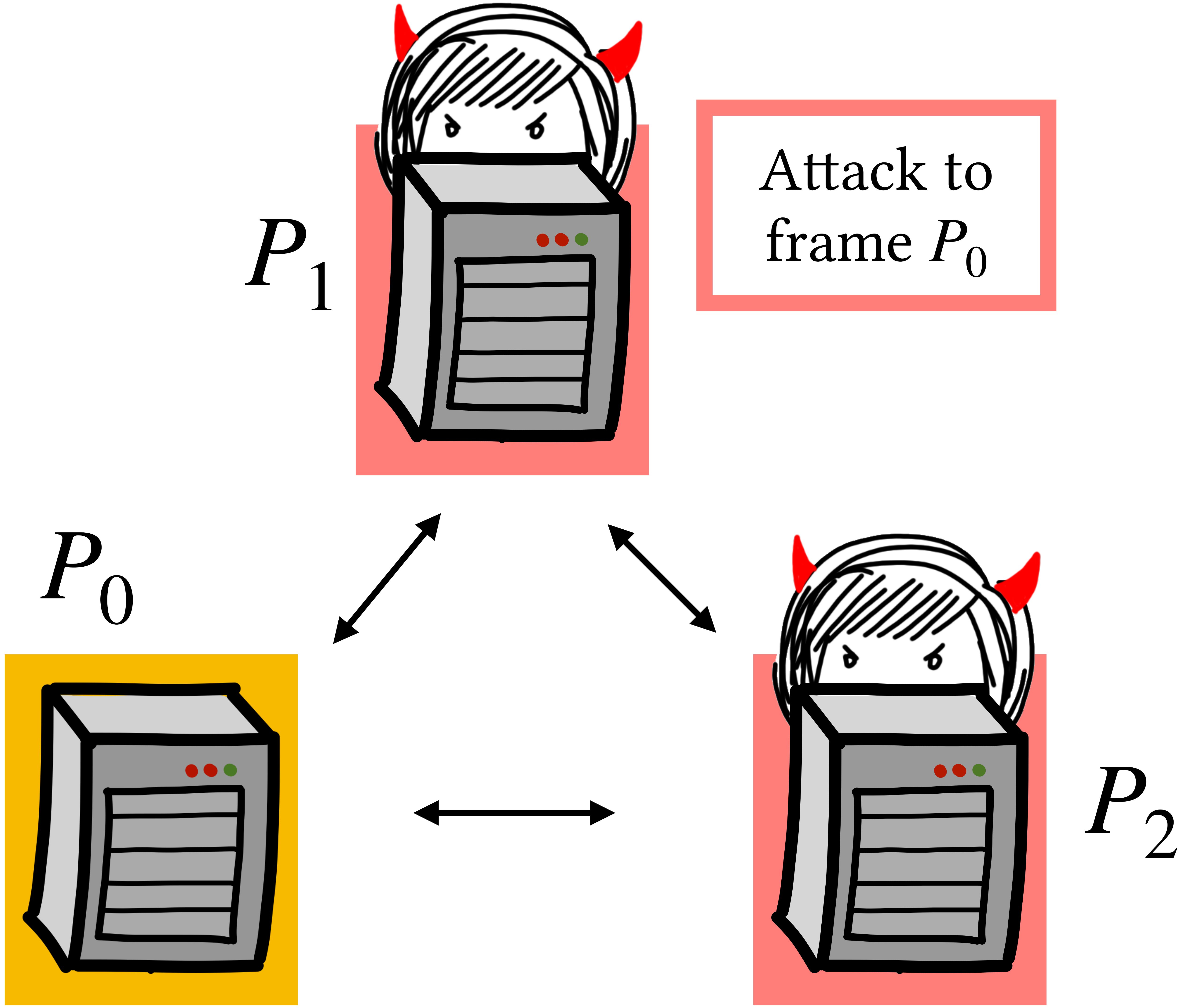
Broadcast-IA is Impossible with Dishonest Majority

[This work]



Broadcast-IA is Impossible with Dishonest Majority

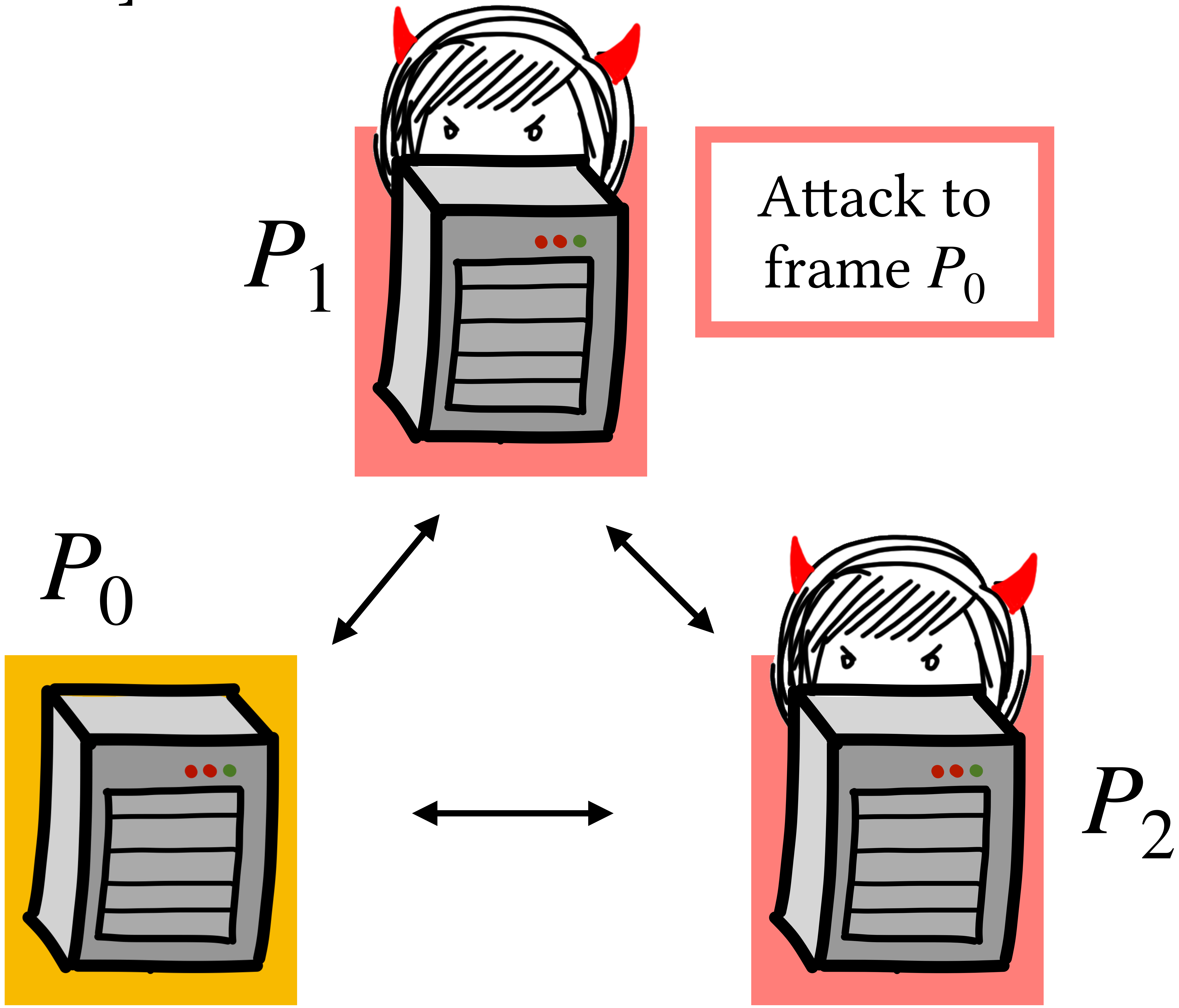
[This work]



Attack to frame P_0

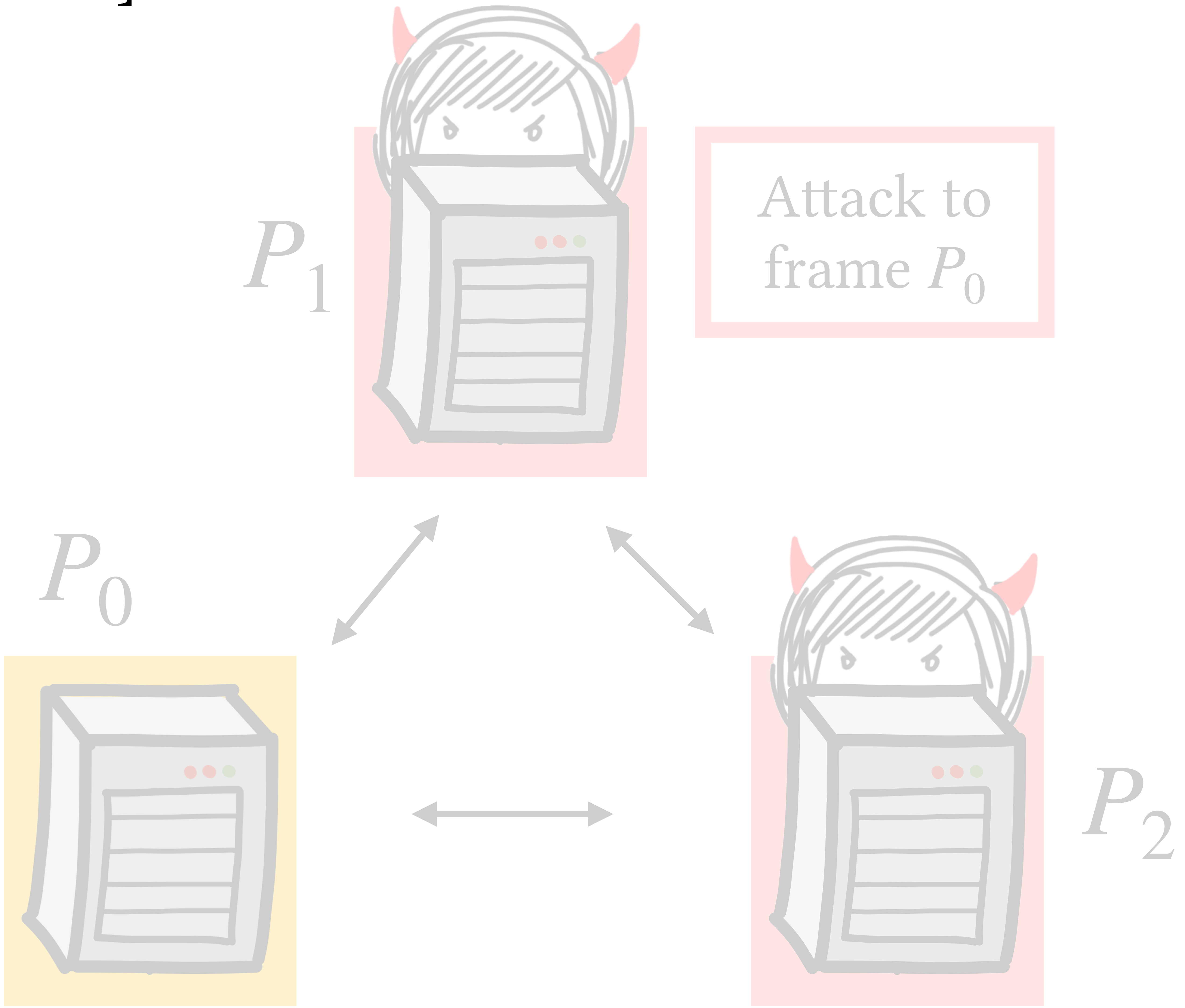
Broadcast-IA is Impossible with Dishonest Majority

[This work]



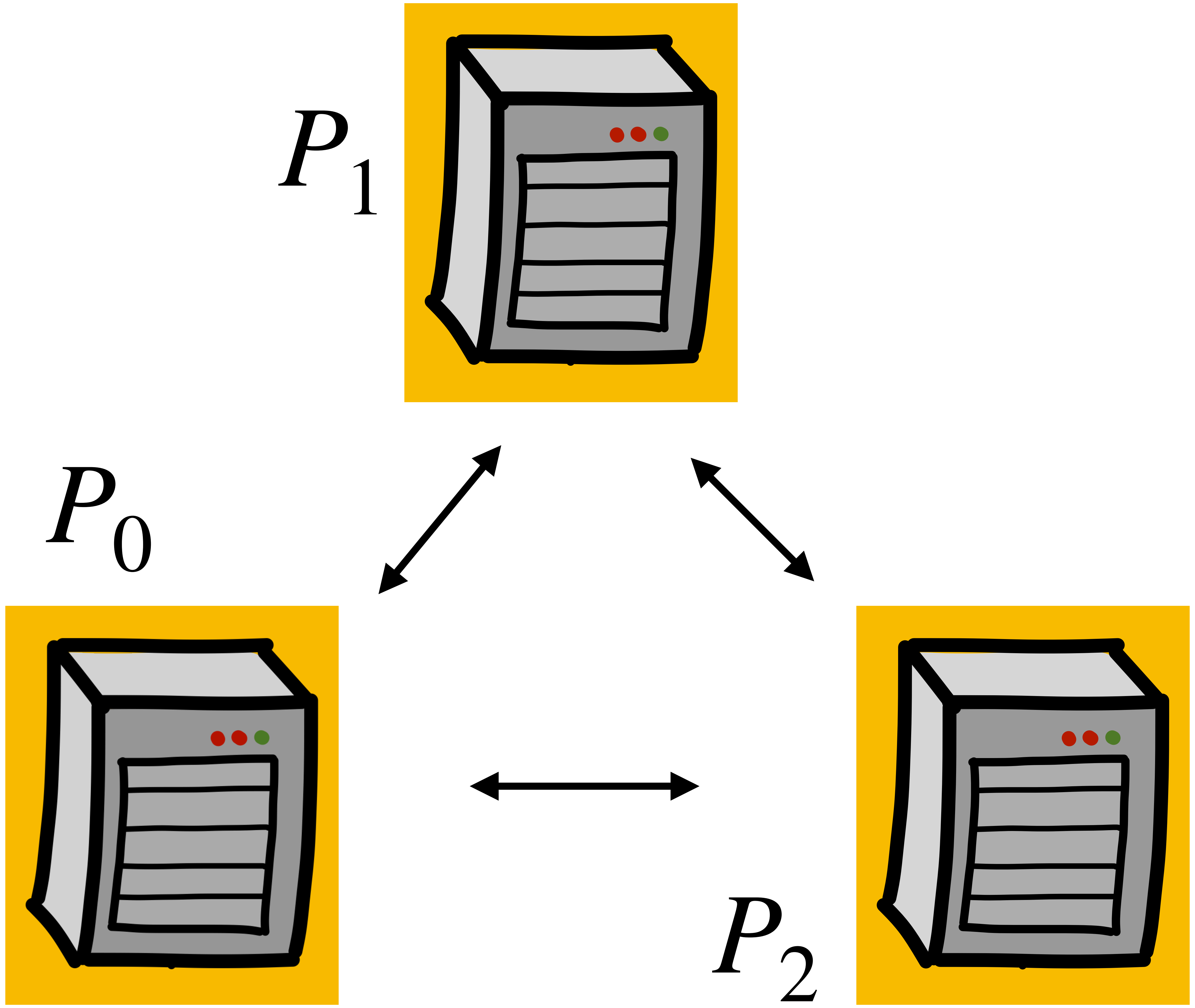
Broadcast-IA is Impossible with Dishonest Majority

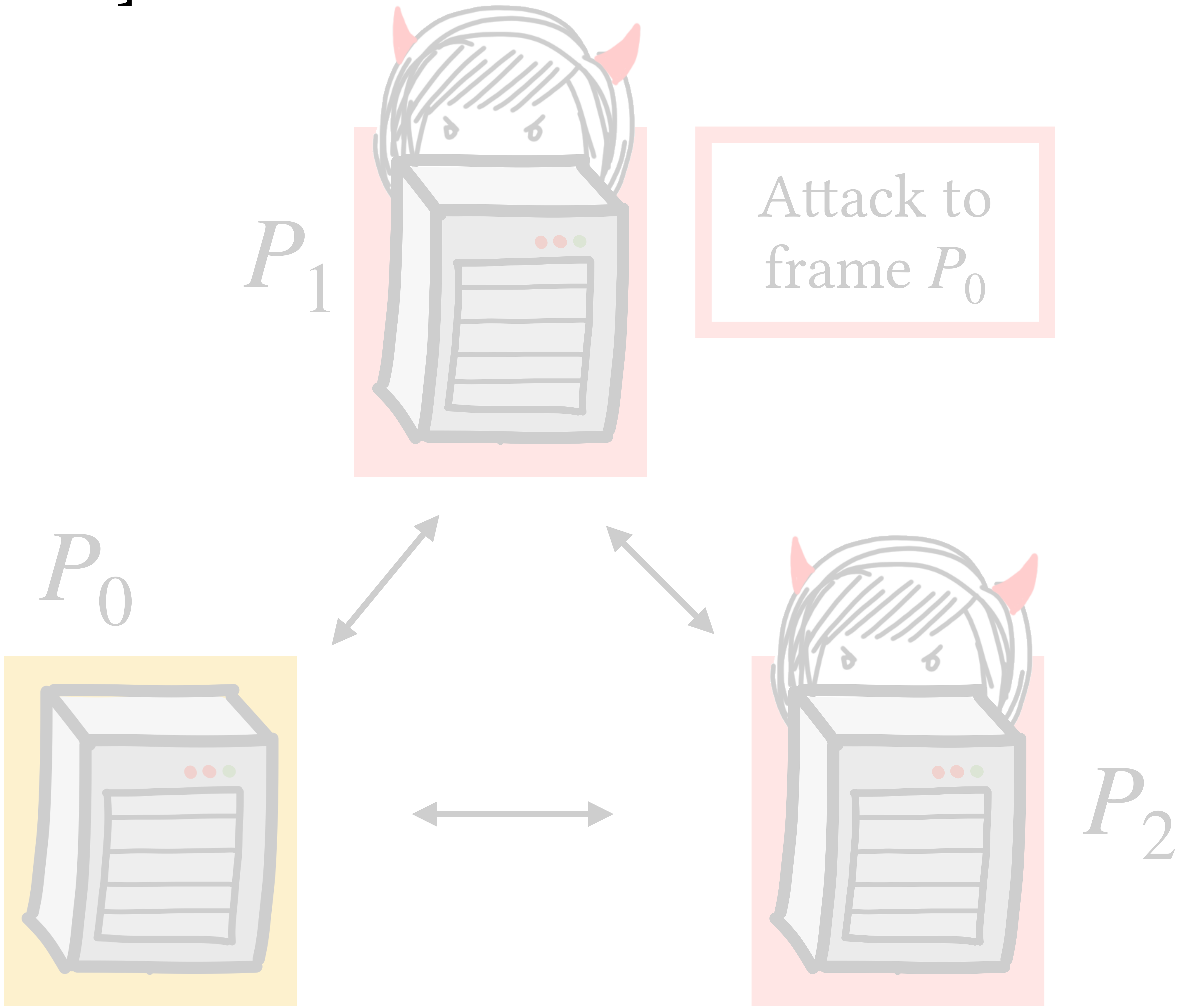
[This work]



Broadcast-IA is Impossible with Dishonest Majority

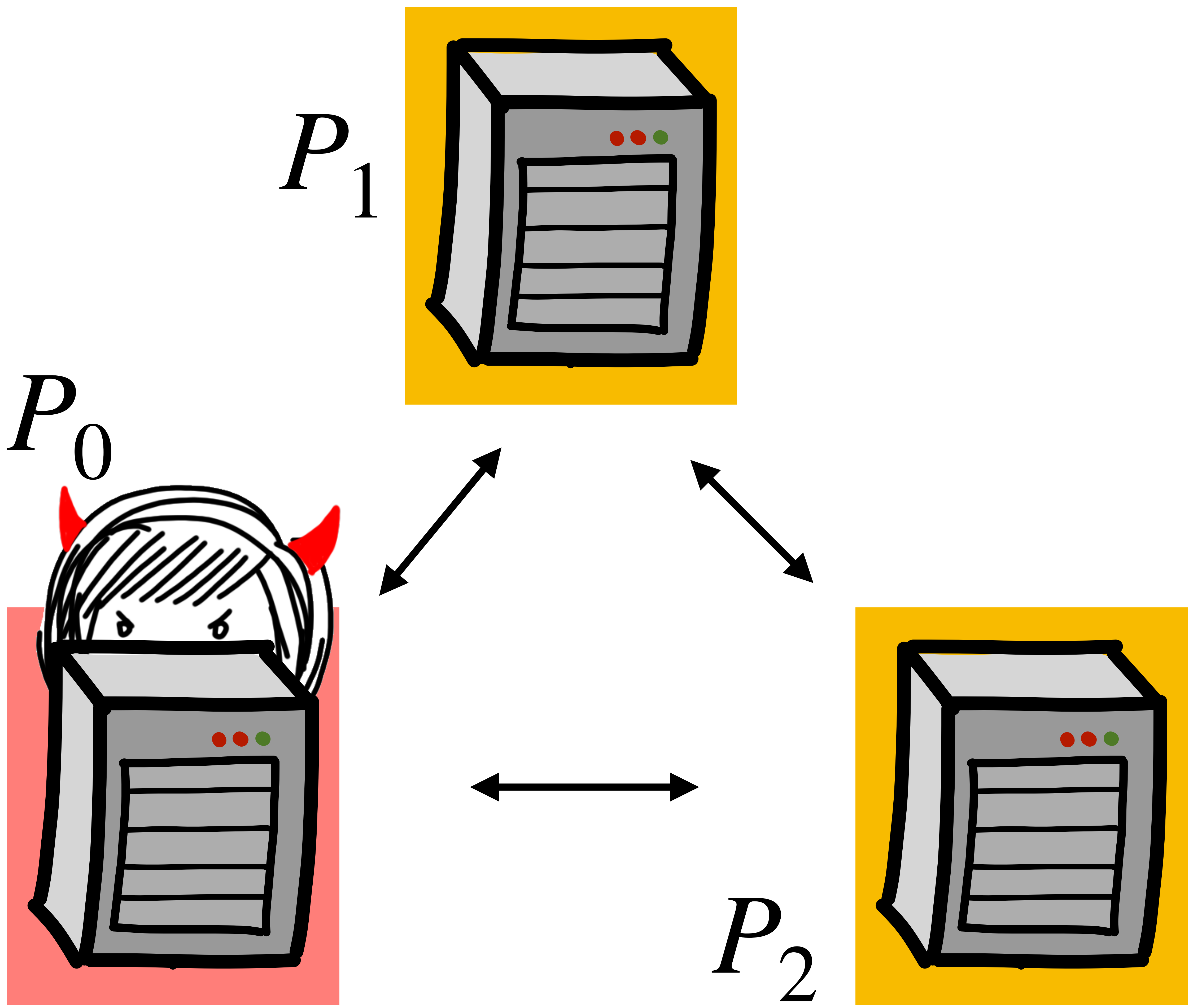
[This work]

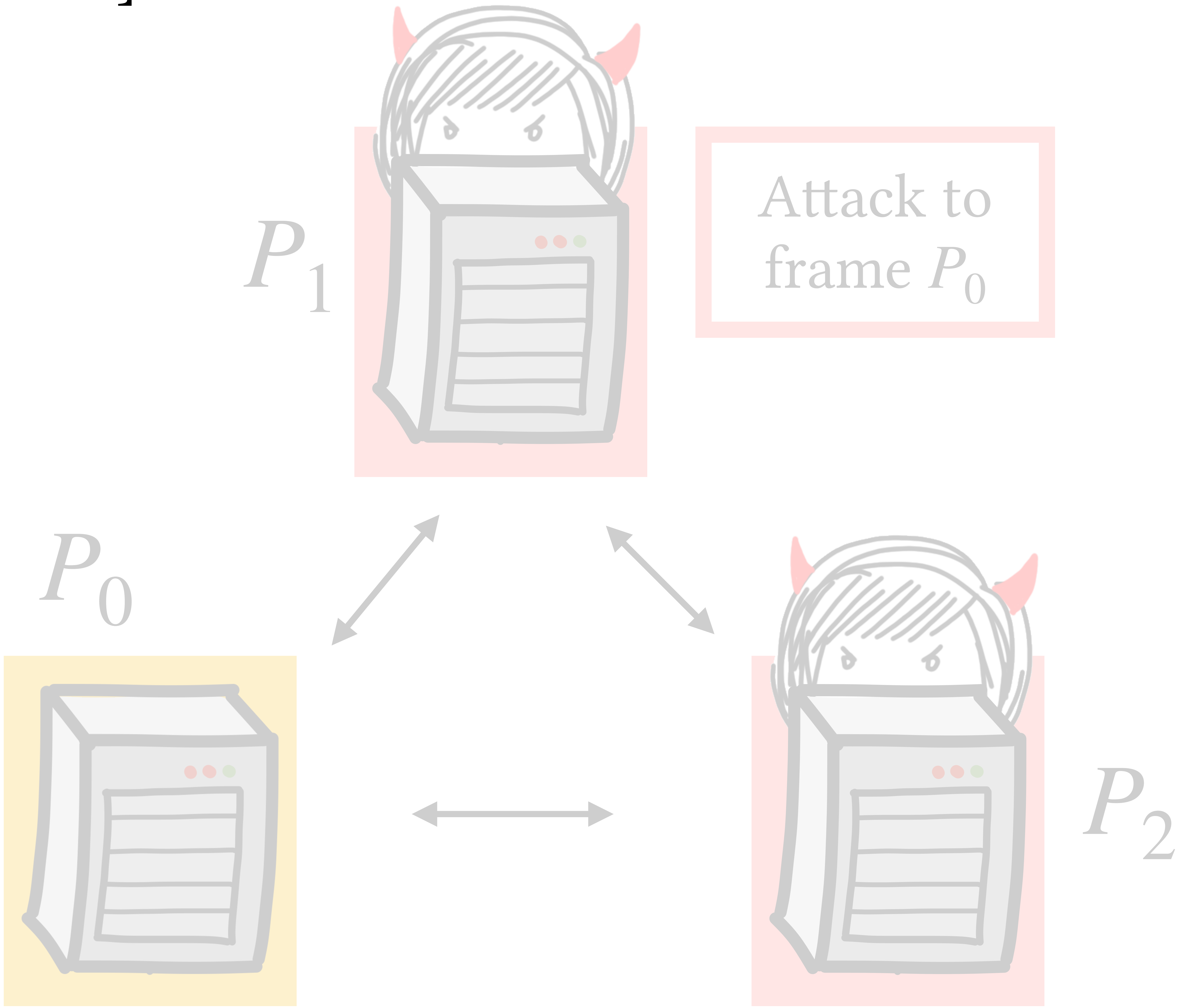




Broadcast-IA is Impossible with Dishonest Majority

[This work]





Broadcast-IA is Impossible with Dishonest Majority

[This work]

P_1
P_0
P_2
P_1
Attack to frame P_0
P_0
P_2

Broadcast-IA is Impossible with Dishonest Majority

[This work]

P_1
P_0
P_2
P_1
P_0
P_2

OUTPUT

ω: "P_0 offline"

Attack to frame P_0

Broadcast-IA is Impossible with Dishonest Majority

[This work]

P_1
P_0
P_2
P_1
P_0
P_2

OUTPUT

ω: “P_0 offline”

Attack to frame P_0

Broadcast-IA is Impossible with Dishonest Majority

[This work]

OUTPUT

ω: “P_0 offline”

Attack to frame P_0

P_1
P_0
P_2
P_1
P_0
P_2

Broadcast-IA is Impossible with Dishonest Majority

[This work]

P_1
P_0
P_2
P_1
P_0
P_2

OUTPUT

ω: "P_0 offline"

Attack to frame P_0

Broadcast-IA is Impossible with Dishonest Majority

[This work]

P_1
P_0
P_2
P_1
P_0
P_2

OUTPUT

ω: "P_0 offline"

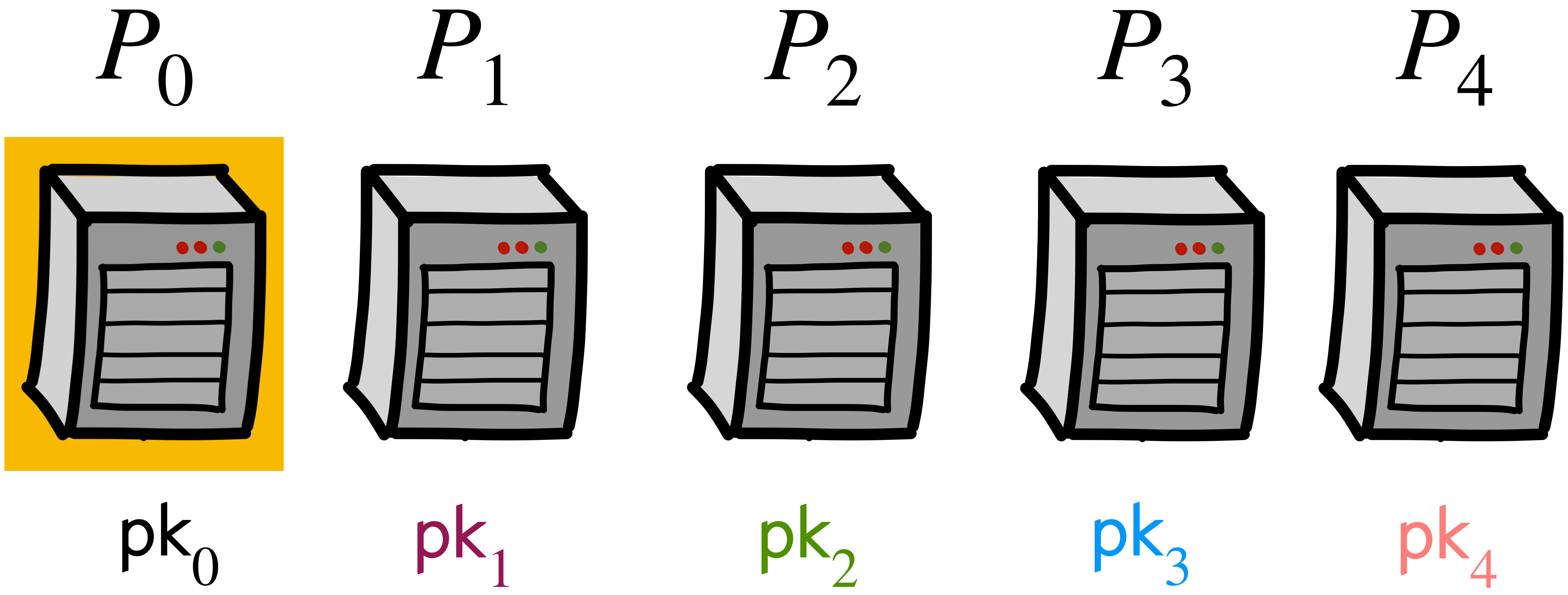
Attack to frame P_0

OUTPUT

ω: "P_0 offline"

Broadcast-IA with Honest Majority

[This work]



P_0 P_1 P_2 P_3 P_4

pk_0 pk_1 pk_2 pk_3 pk_4

P_0 wishes to broadcast m

Broadcast-IA with Honest Majority

[This work]

Round 1

Round 2

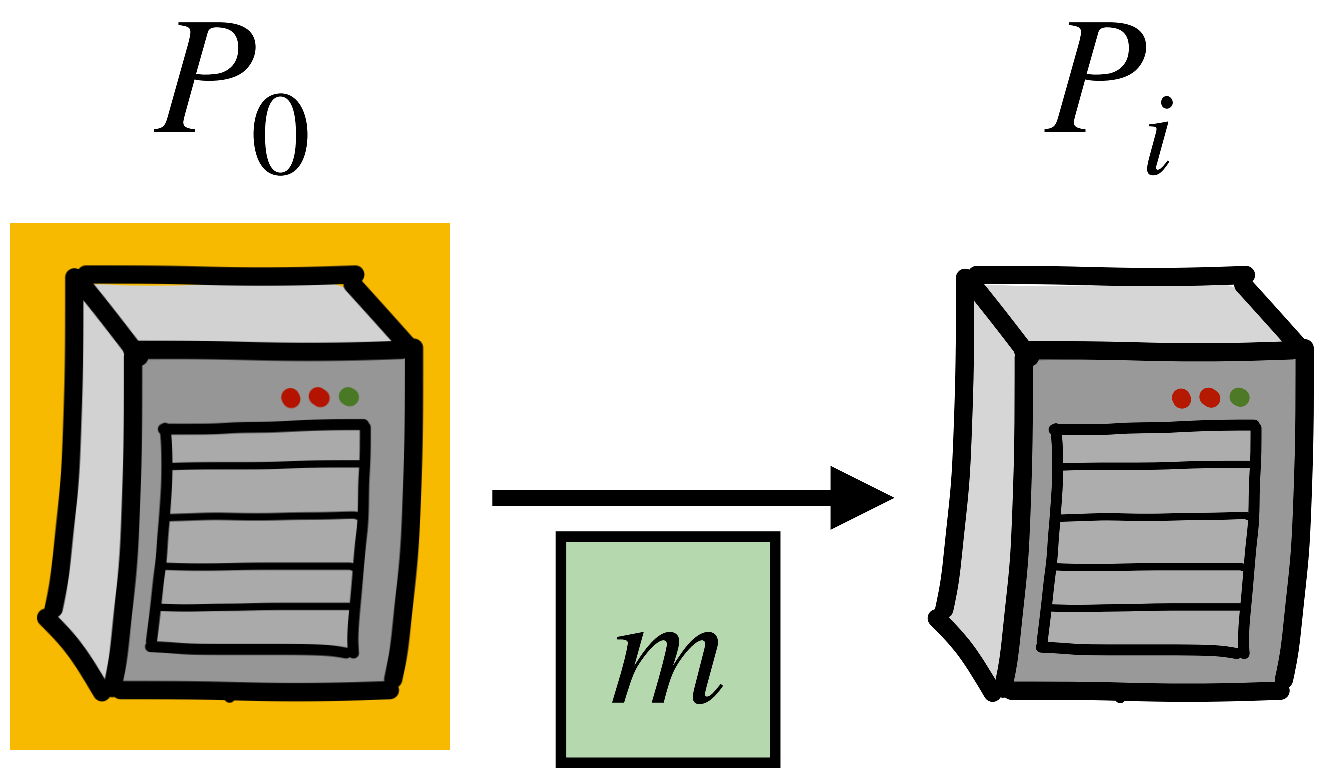
Output

Broadcast-IA with Honest Majority

[This work]

Round 1

Sign m,
Send to all



Round 2

Output

Broadcast-IA with Honest Majority

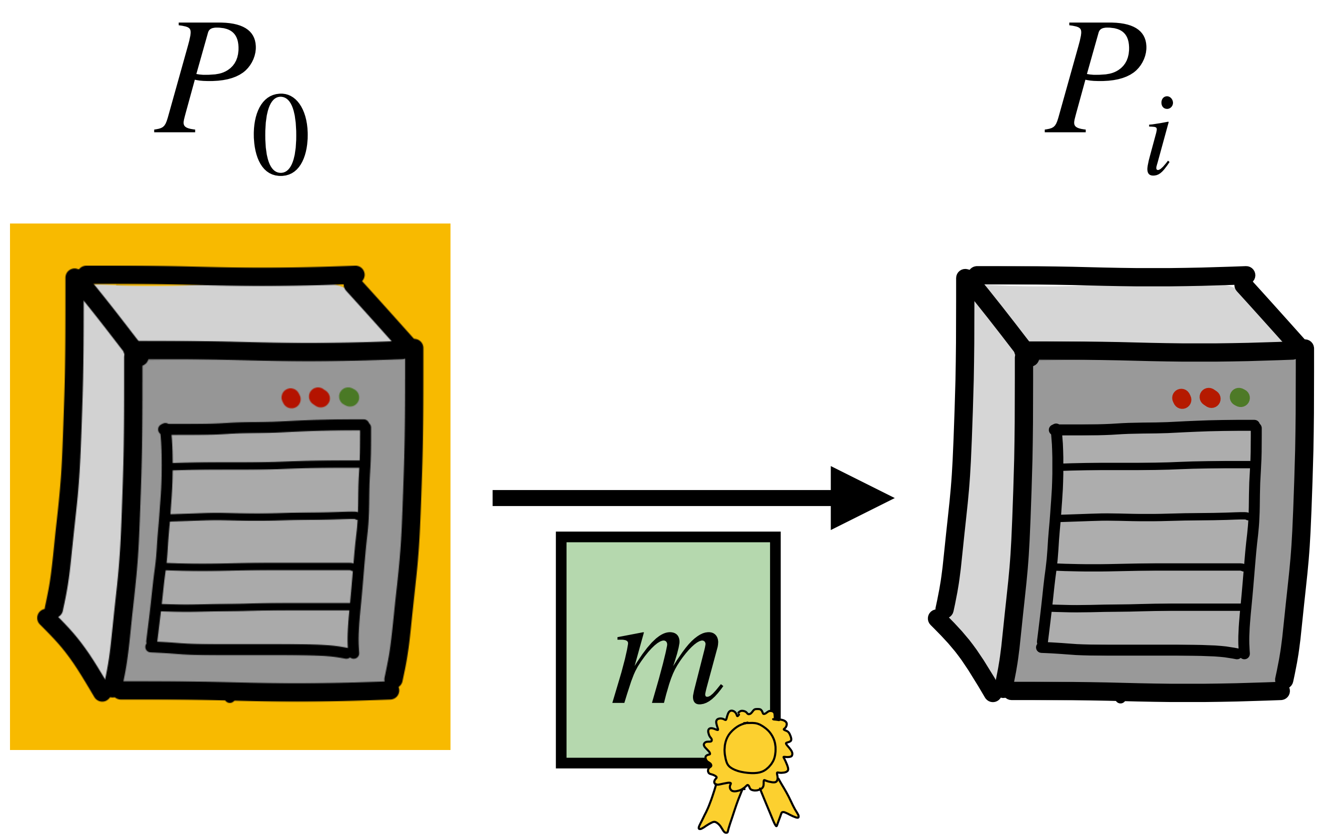
[This work]

Round 1

Sign m,
Send to all

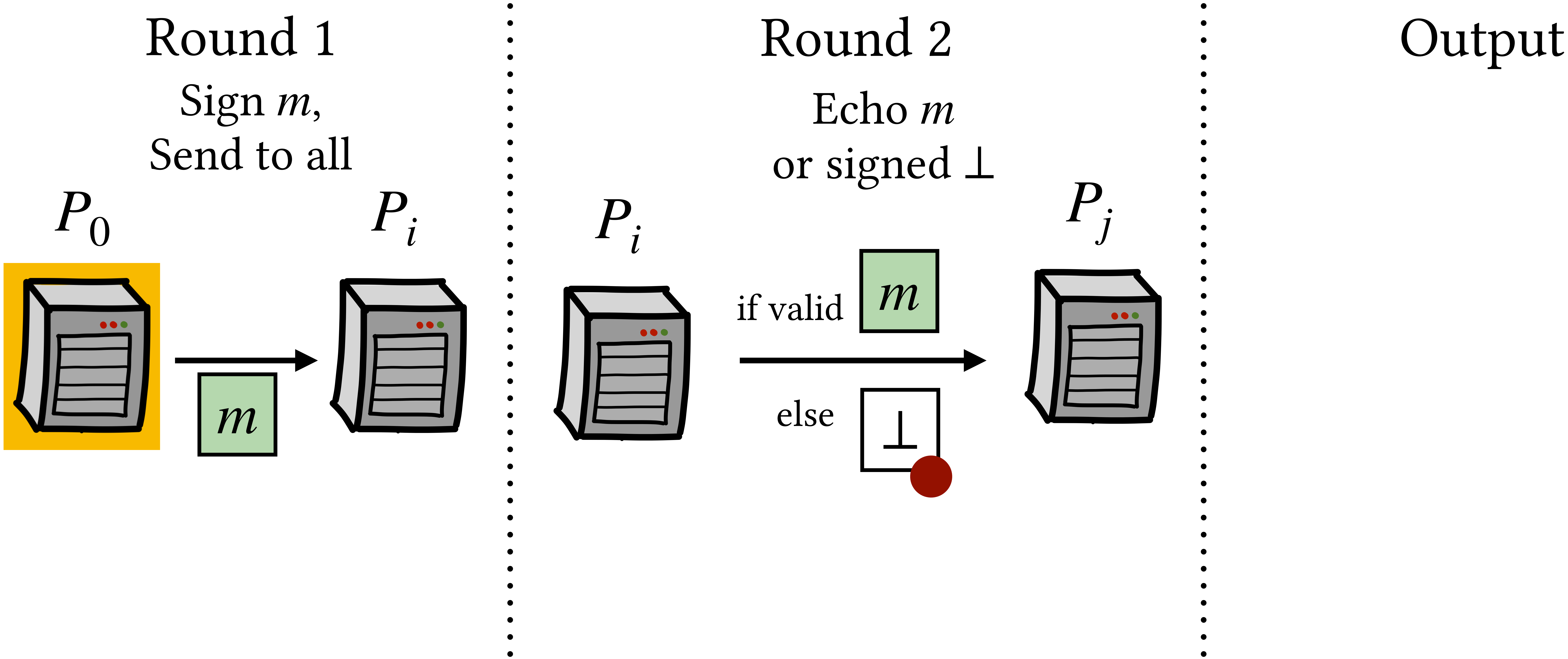
Round 2

Output



Broadcast-IA with Honest Majority

[This work]



Round 1

Sign m,
Send to all

P_0 P_i

m

Round 2

Echo m
or signed $\perp$

P_i P_j

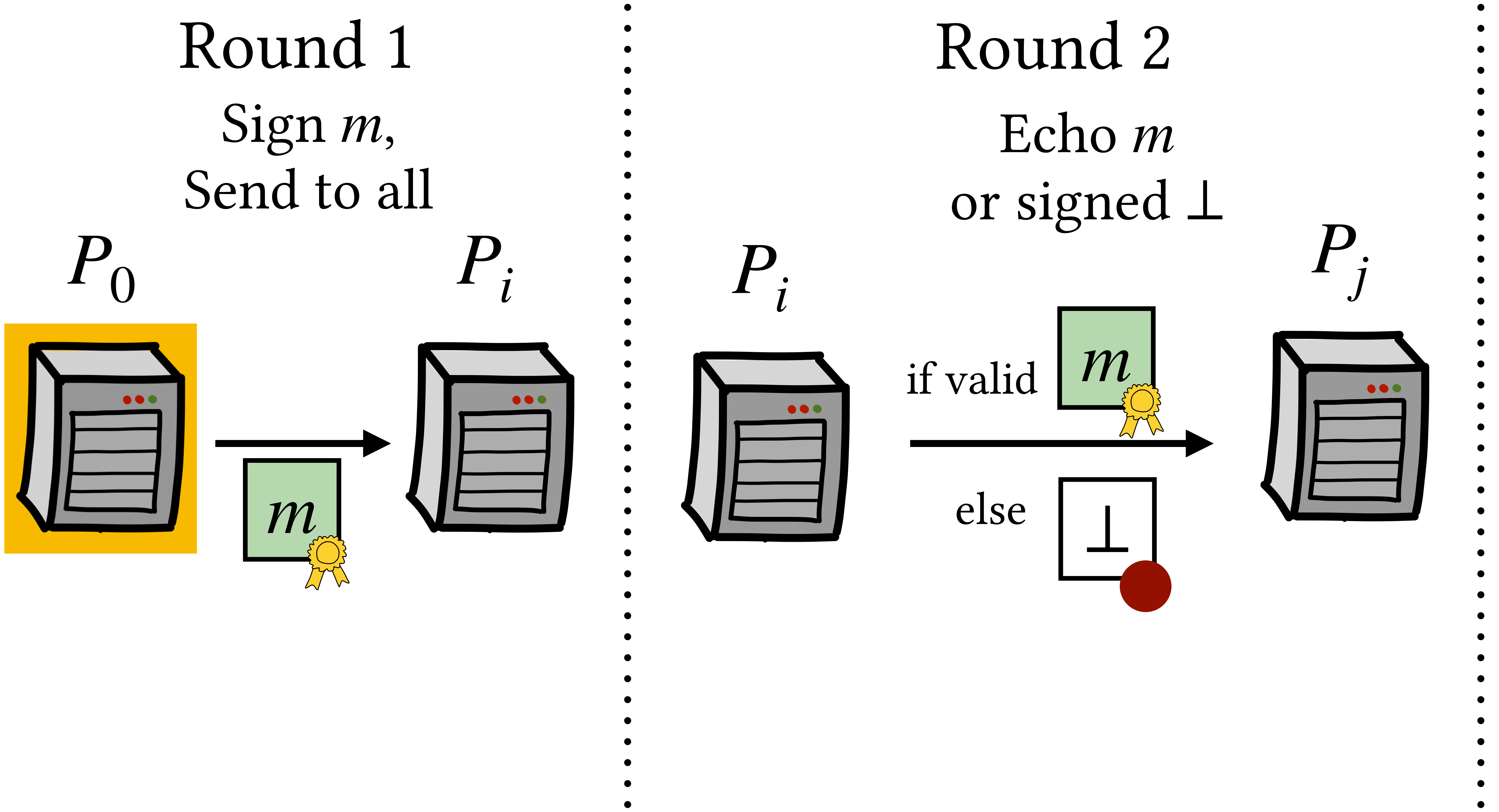
if valid m

else $\perp$

Output

Broadcast-IA with Honest Majority

[This work]



Round 1

Sign m,
Send to all

P_0 $\xrightarrow{m}$ P_i

Round 2

Echo m
or signed $\perp$

P_i $\longrightarrow$ P_j

if valid m

else $\perp$

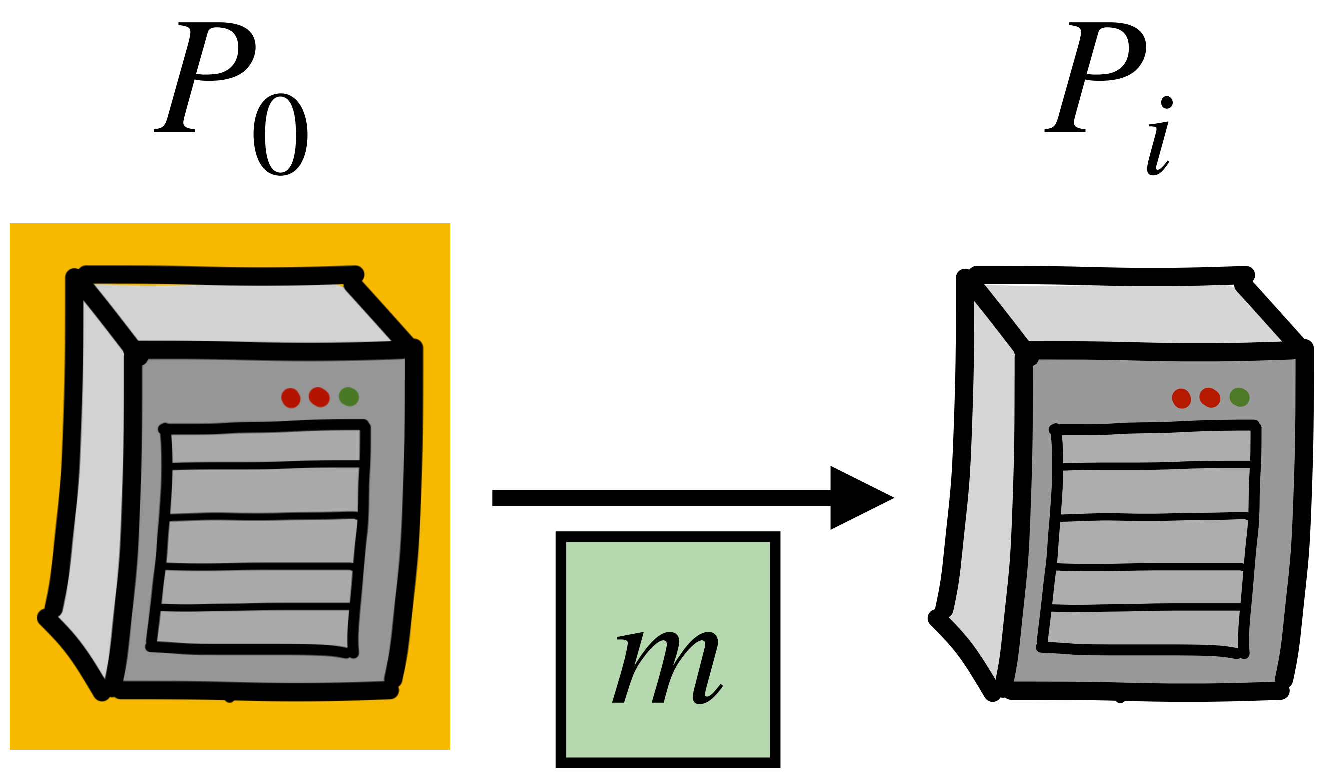
Output

Broadcast-IA with Honest Majority

[This work]

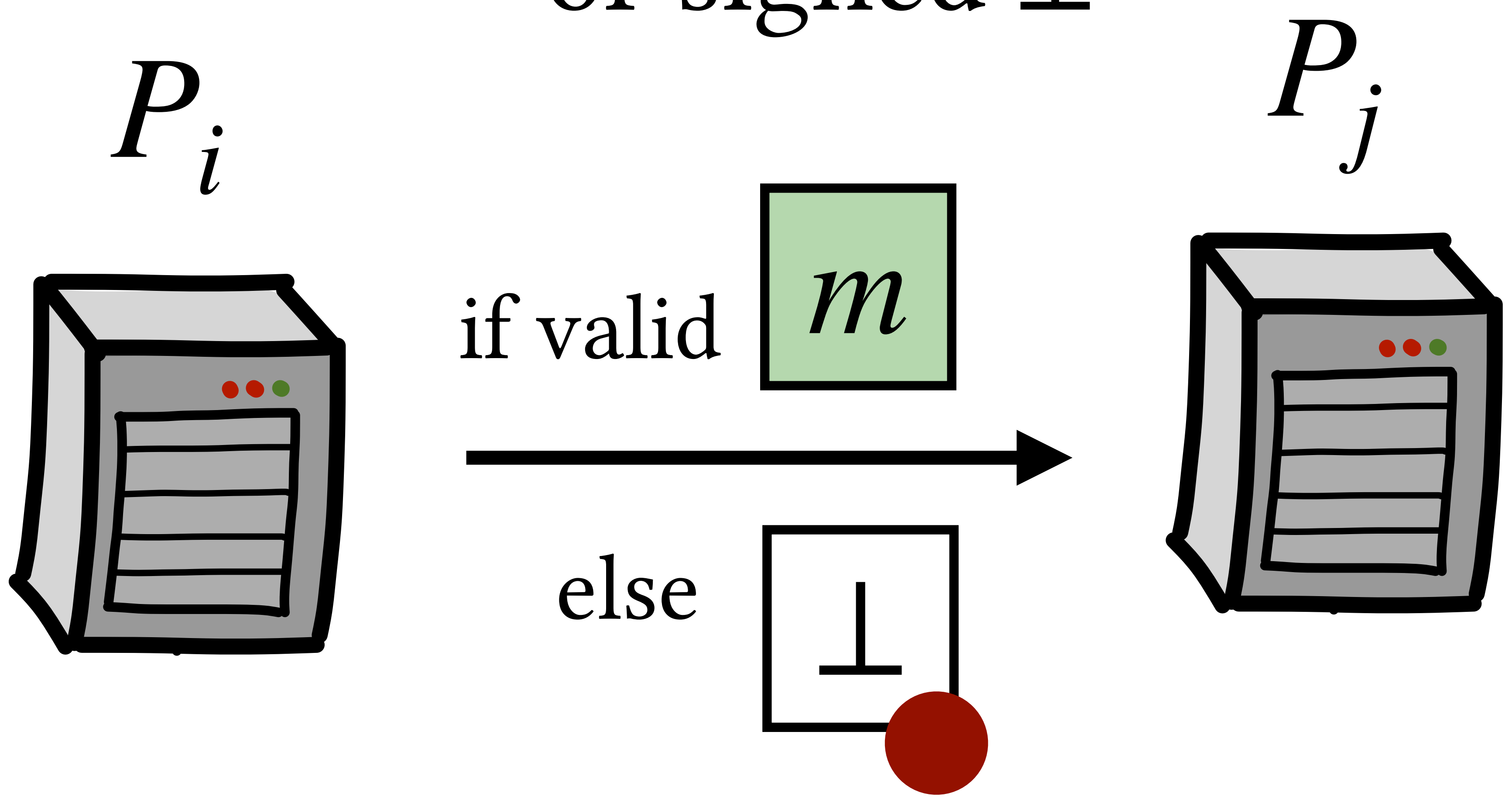
Round 1

Sign m,
Send to all



Round 2

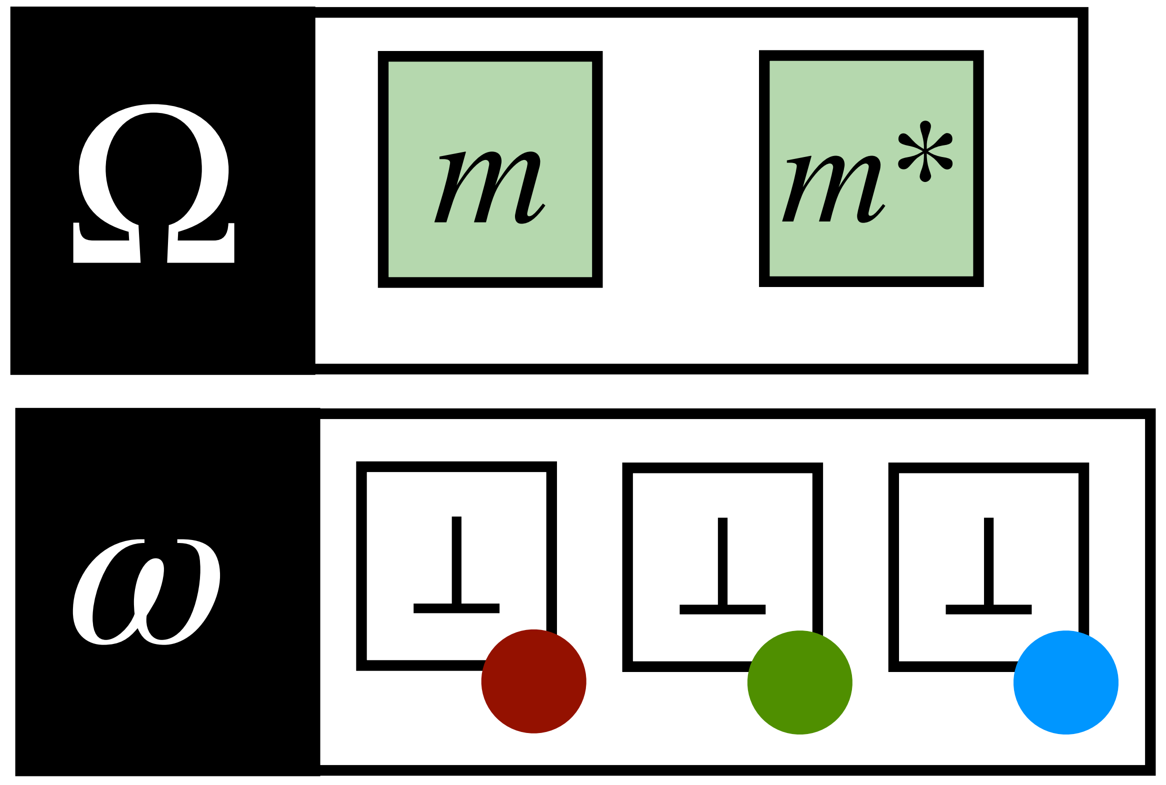
Echo m
or signed $\perp$



Output

Each P_i :

1. Check for potential certificates of cheating:



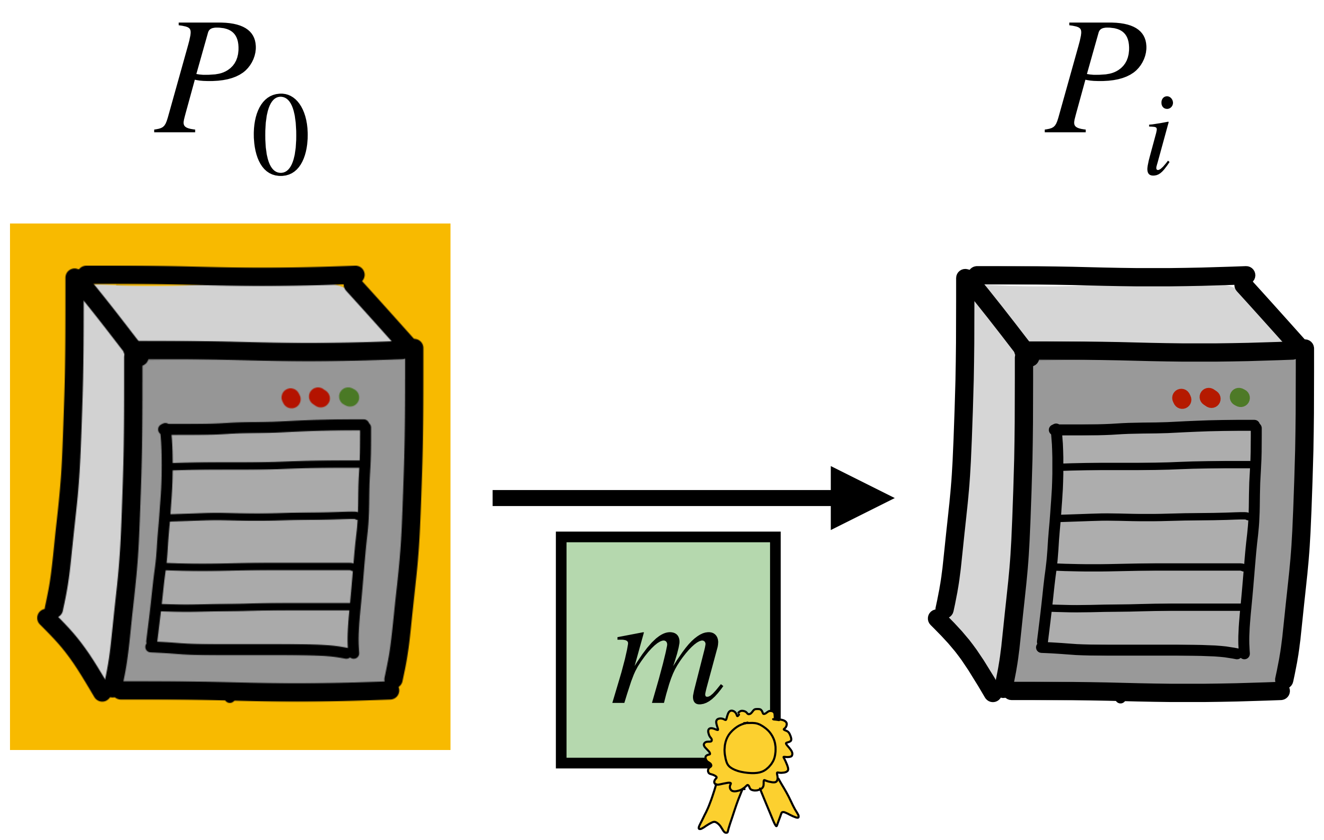
2. If no Ω, ω found, output m

Broadcast-IA with Honest Majority

[This work]

Round 1

Sign m,
Send to all



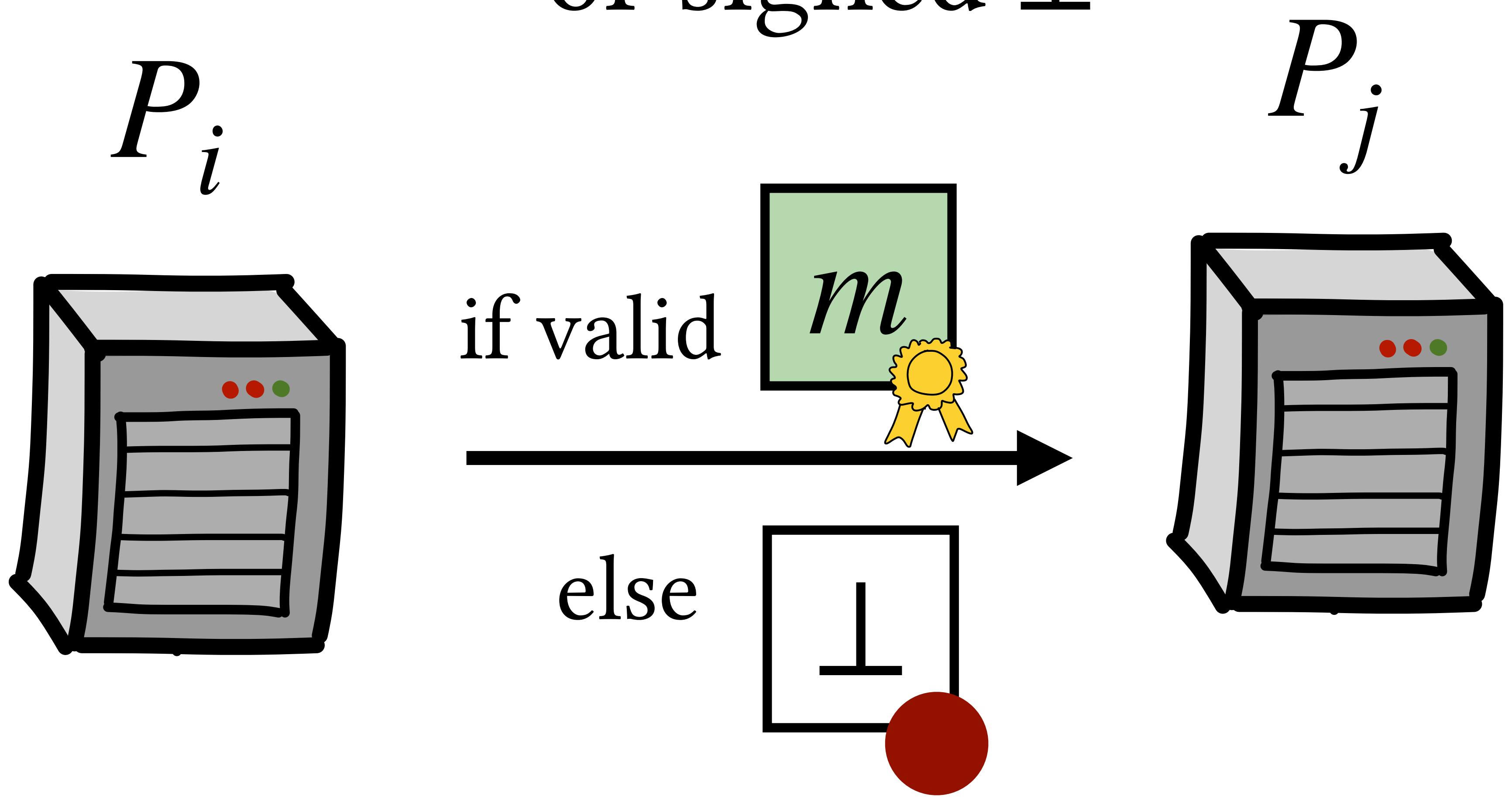
P_0

P_i

m

Round 2

Echo m
or signed $\perp$



P_i

P_j

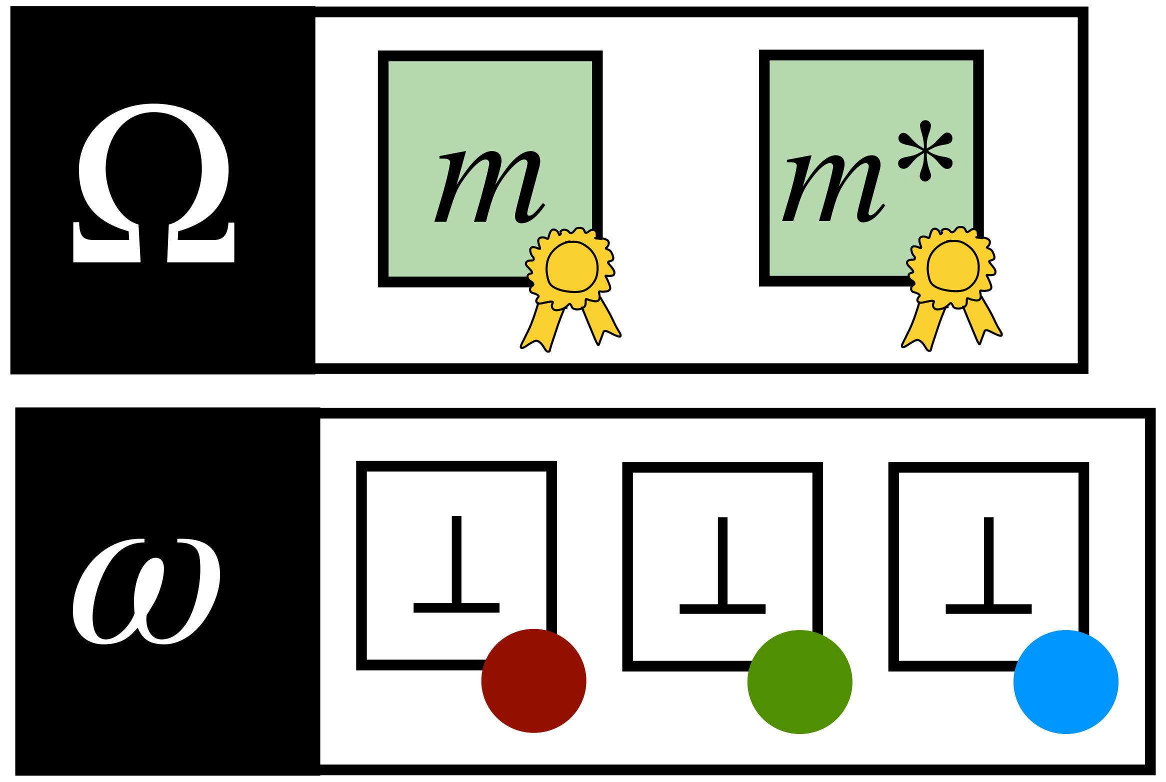
if valid m

else $\perp$

Output

Each P_i :

1. Check for potential certificates of cheating:



Ω m m^*

ω $\perp$ $\perp$ $\perp$

2. If no Ω, ω found, output m

Broadcast-IA: Analysis

- **Honest** P_0: Complete, defamation-free
  - No Ω: Will not sending conflicting m, m^*
  - No ω: At most t corrupt parties will echo $\perp$ ⇒ not enough sigs

- **Corrupt** P_0: Consistent
  - If any honest parties receive m, m^* ⇒ yields Ω
  - If m withheld from *all* honest parties ⇒ yields ω
  - Send m to any honest party ⇒ m committed as output

- **Notes on output** m:

  1. Accompanied by sig(m) from P_0: proves P_0 sent m to P_i
  2. P_i producing sig(m) DOES NOT prove that some P_j also output m

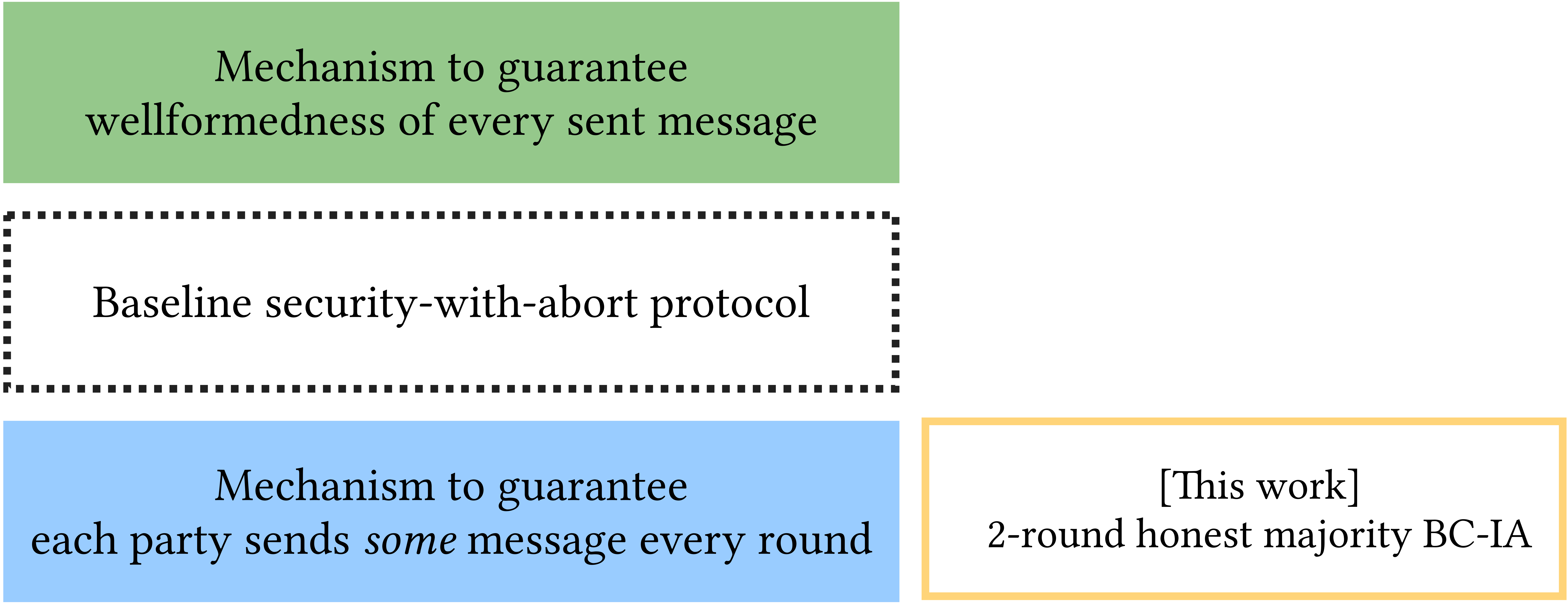
Synchrony

- Protocol assumes a well-defined network time-out (i.e. synchrony)

- Inherent: Identifiable Abort not well-defined in p2p asynchronous setting
  - Honest parties w. bad network indistinguishable from corrupt

- Important to reason about what happens when network goes bad:
  - Honest parties may be certified non-responsive (ω)
    $\Rightarrow$ Very bad idea to take drastic action based on non-responsiveness alone
  - Liveness may be violated
  - Cheat (Ω) remains attributable to corrupt parties only
    $\Rightarrow$ Higher level protocols can still maintain safety/privacy of secrets

Anatomy of MPC-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]
2-round honest majority BC-IA

Anatomy of MPC-IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

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

Informal Theorem

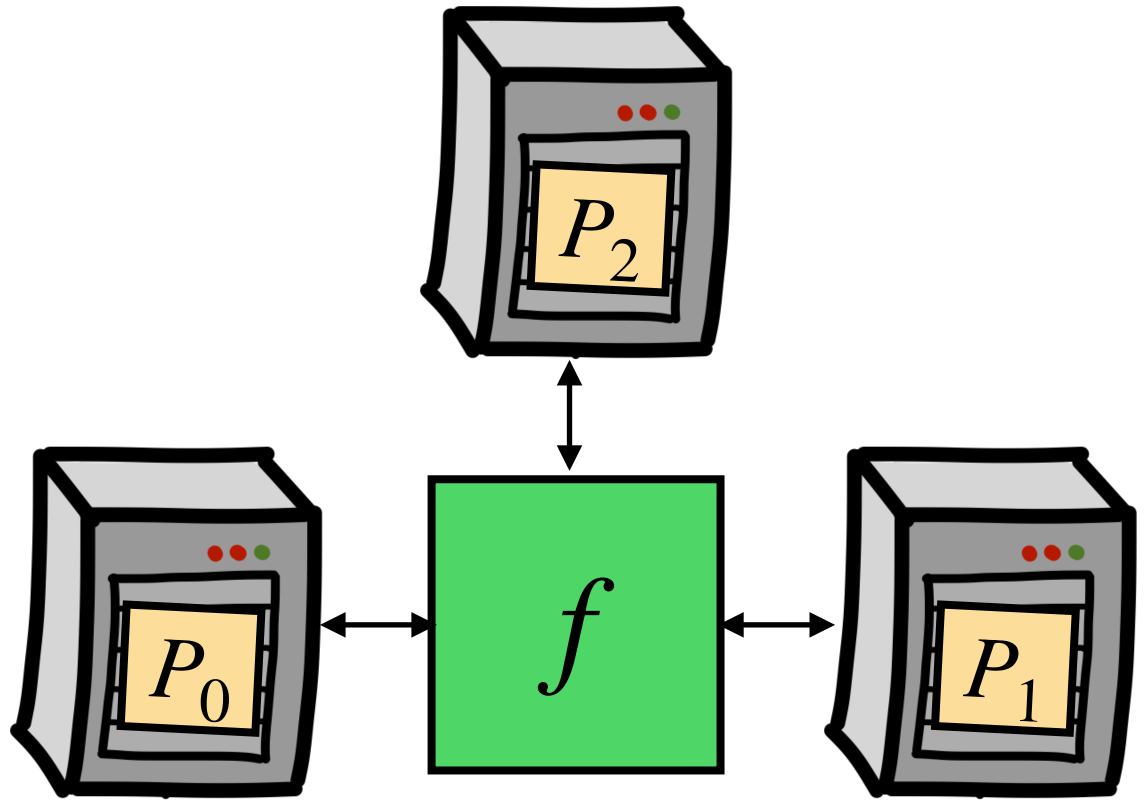
If Π^{BC} is a protocol that realizes $\mathcal{F}^{f}_{\mathsf{IA}}$ using r Ideal Broadcasts, then $\Pi^{\mathsf{BC\text{-}IA}}$ realizes $\mathcal{F}^{f}_{\mathsf{IA}*}$ using $(r+1)$ BC-IA instances

$\Rightarrow 2(r+1)$ p2p rounds

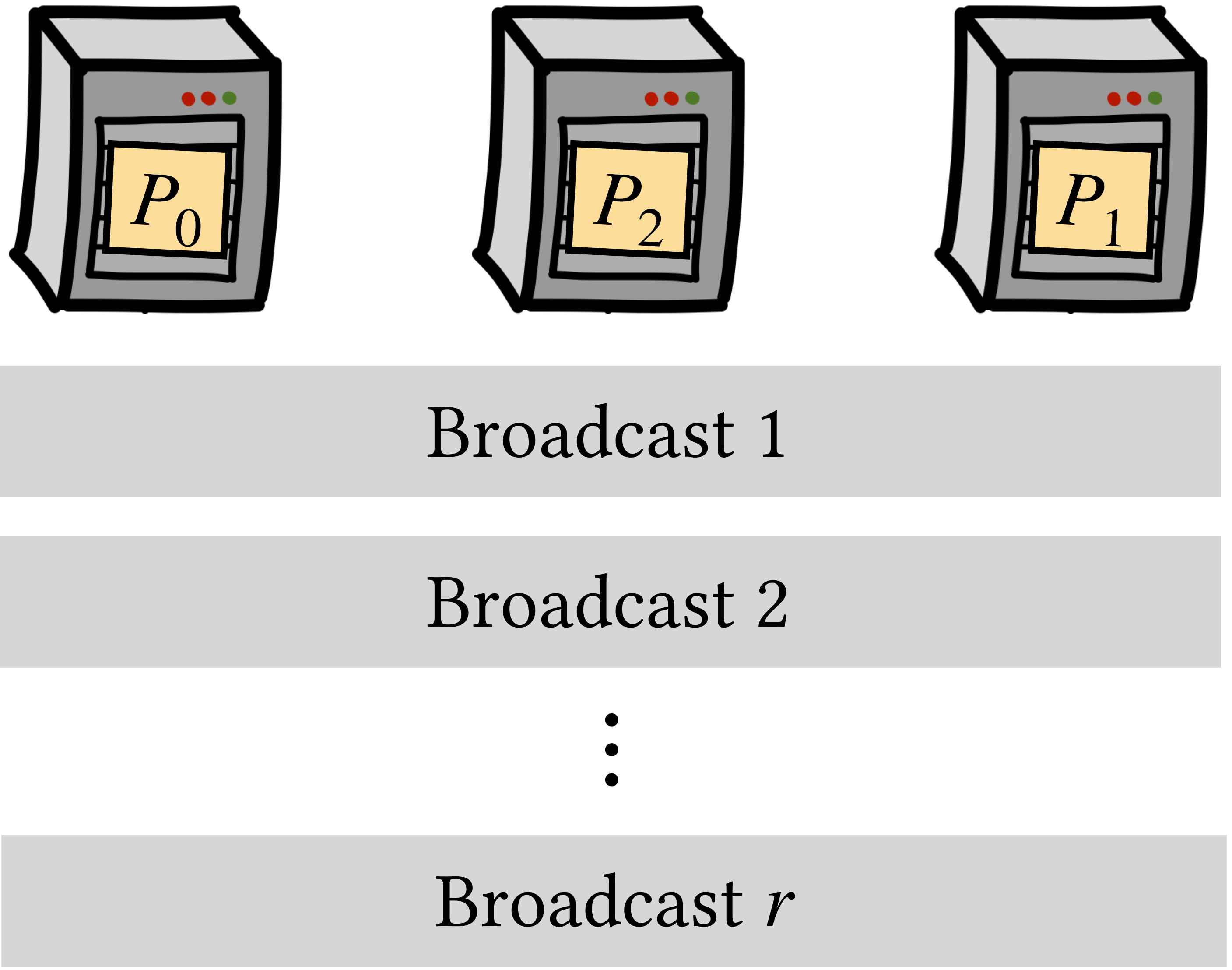
[This work]
2-round honest majority BC-IA

Our Compiler

Ideal



Real



Broadcast 1

Broadcast 2

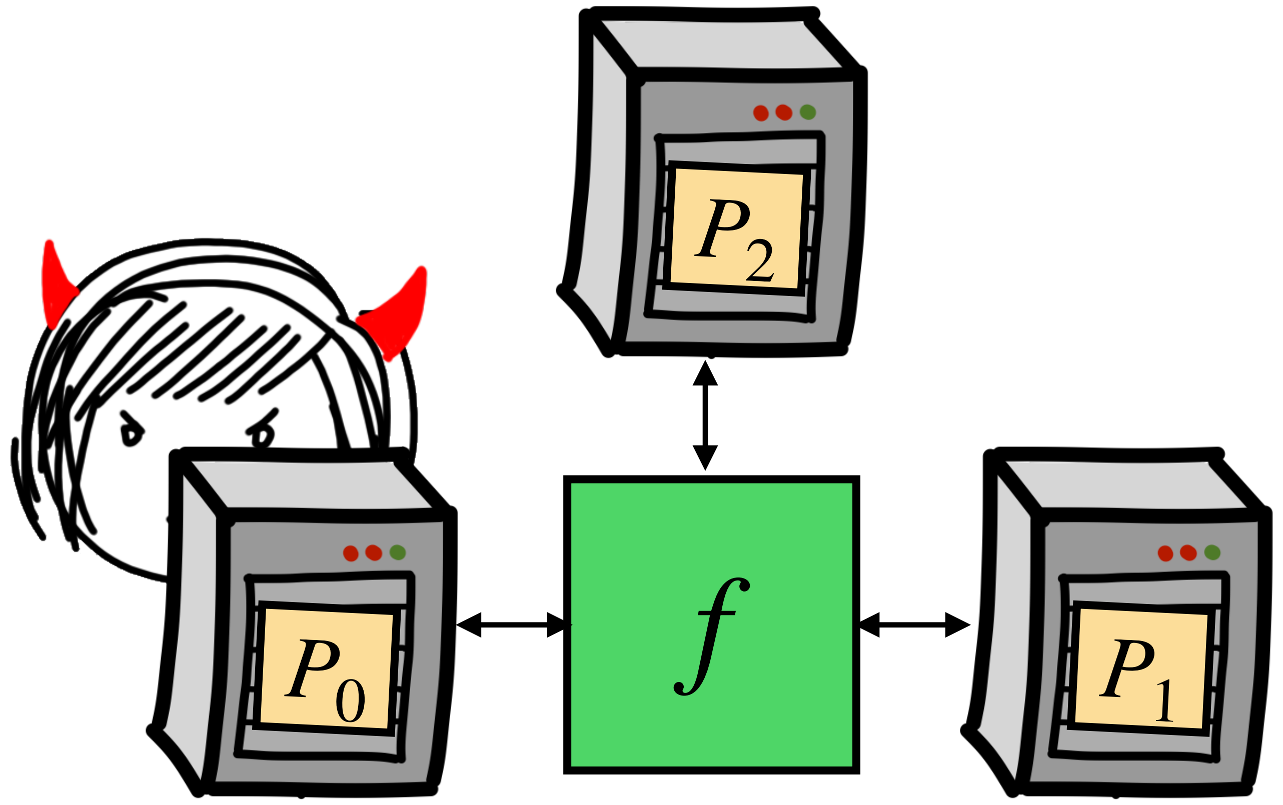
⋮

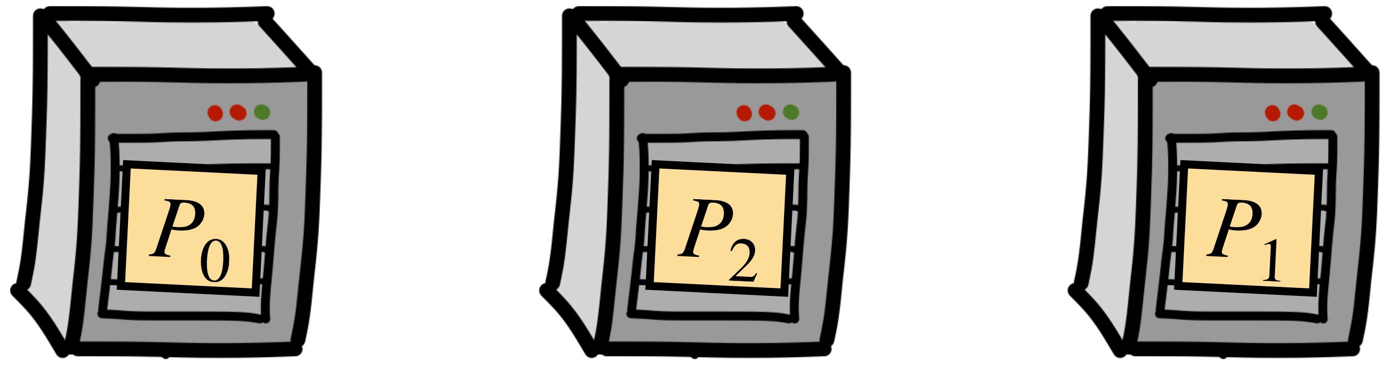
Broadcast r

Π^{BC}

Our Compiler

Ideal





Real

Π^{BC}

Broadcast 1

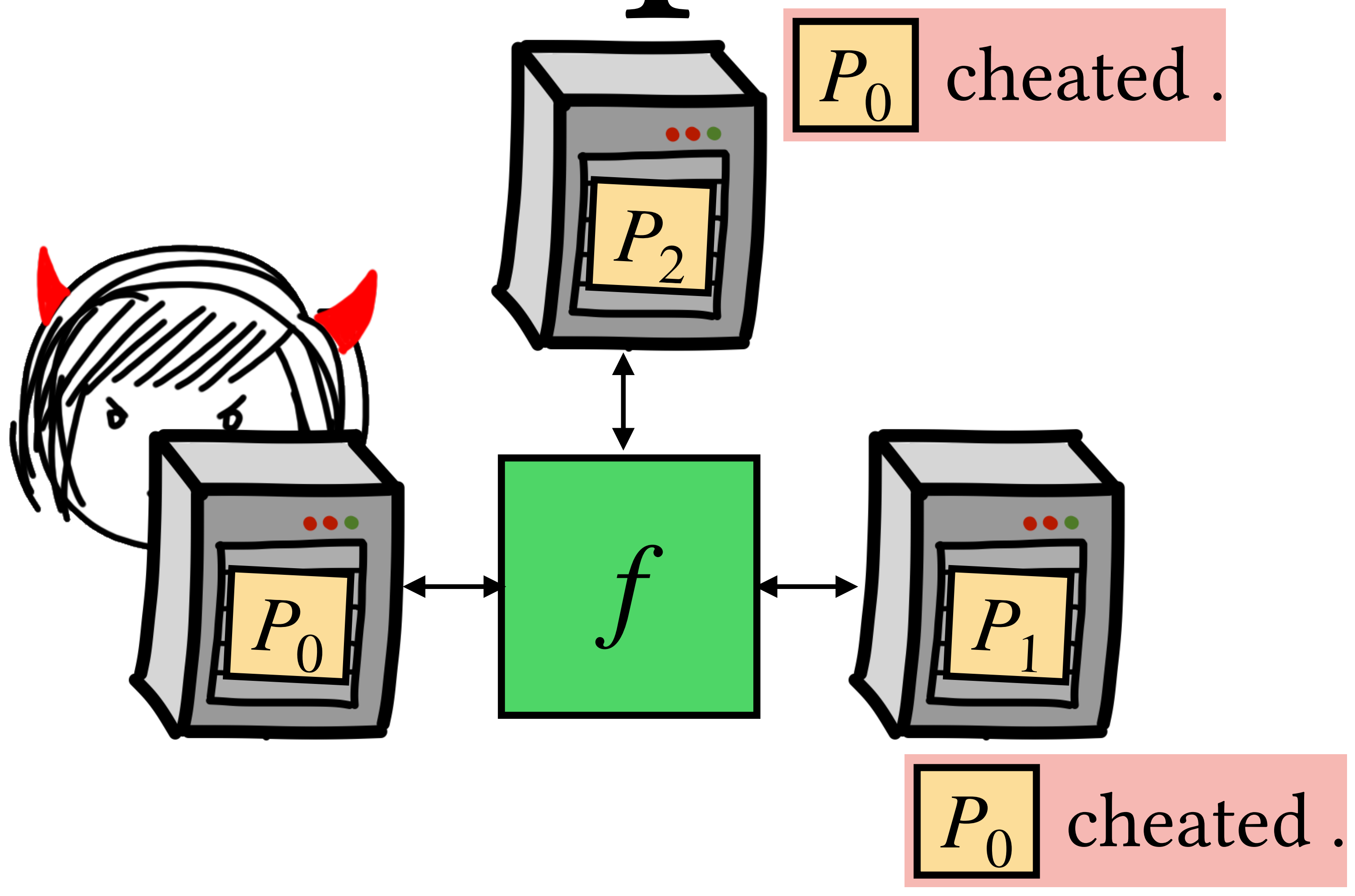
Broadcast 2

⋮

Broadcast r

Our Compiler

Ideal

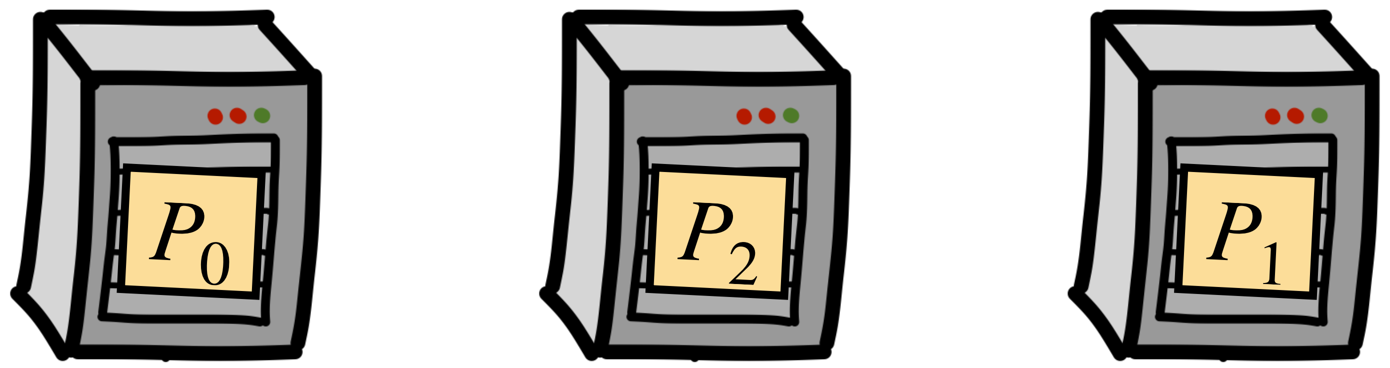


P_0 cheated .

P_0 cheated .

Real

Π^{BC}



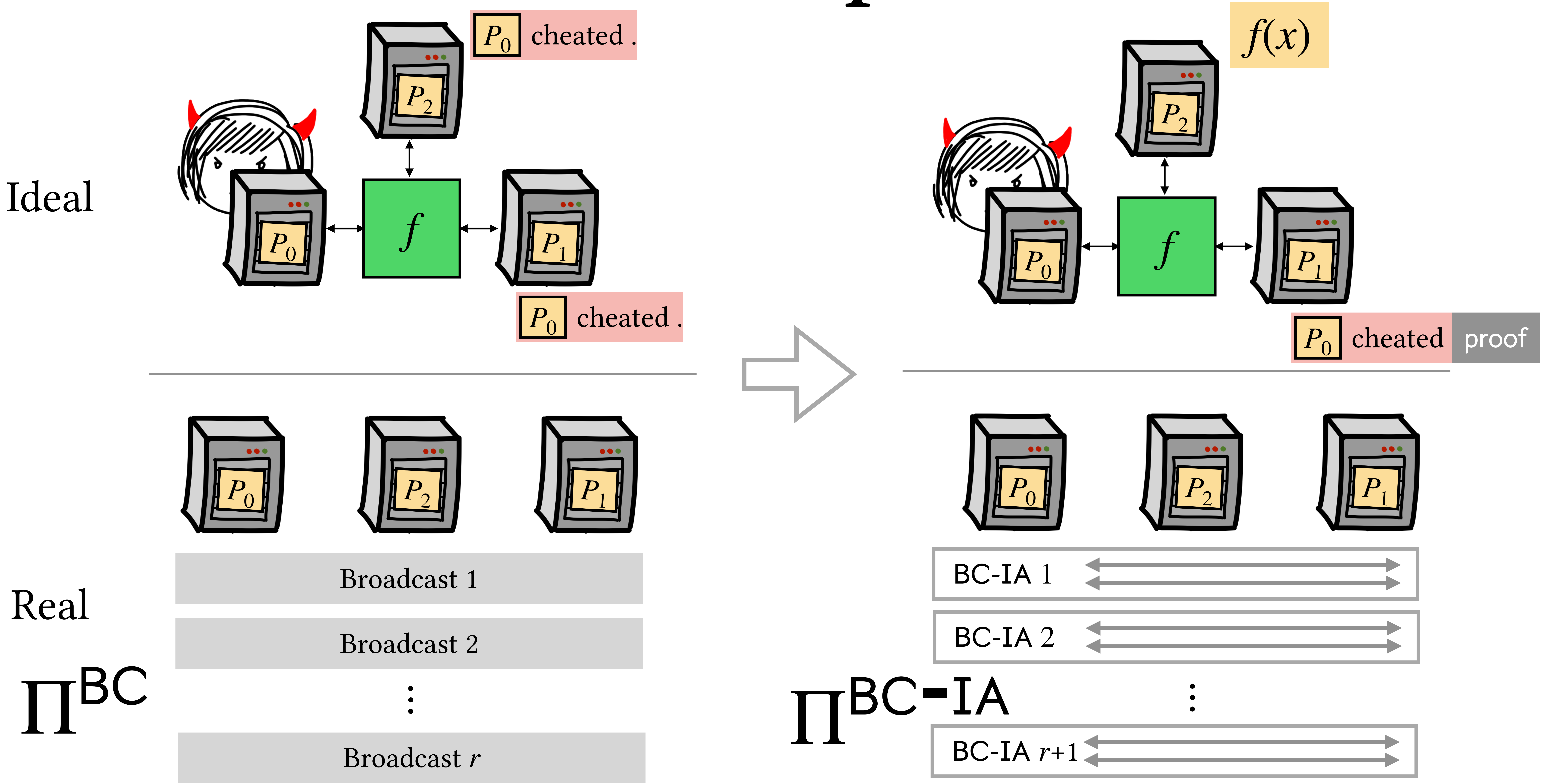
Broadcast 1

Broadcast 2

⋮

Broadcast r

Our Compiler



Ideal

P_0 cheated .

P_2

P_0 f P_1

P_0 cheated .

Real

Π^{BC}

P_0 P_2 P_1

Broadcast 1

Broadcast 2

⋮

Broadcast r

$f(x)$

P_2

P_0 f P_1

P_0 cheated proof

P_0 P_2 P_1

BC-IA 1

BC-IA 2

⋮

BC-IA r+1

$\Pi^{\mathsf{BC\text{-}IA}}$

Anatomy of MPC-IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

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

Which Π^{BC} to plug in?

(Increasingly) well studied in the dishonest majority ($t < n$) setting

[Ishai Ostrovsky Zikas 14][Baum Orsini Scholl Soria-Vazquez 20][Cohen Doerner K shelat 24][Baum Melissaris Rachuri Scholl 24]

[This work]
2-round honest majority BC-IA

Anatomy of MPC-IA

Mechanism to guarantee wellformedness of every sent message

Baseline security-with-abort protocol

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

Which Π^{BC} to plug in?

(Increasingly) well studied in the dishonest majority ($t < n$) setting

[Ishai Ostrovsky Zikas 14][Baum Orsini Scholl Soria-Vazquez 20][Cohen Doerner K shelat 24][Baum Melissaris Rachuri Scholl 24]

[This work]

2-round honest majority BC-IA

inherent

Anatomy of MPC-IA

Mechanism to guarantee
wellformedness of every sent message

Baseline security-with-abort protocol

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

Which Π^{BC} to plug in?

(Increasingly) well studied in the dishonest majority ($t < n$) setting

[Ishai Ostrovsky Zikas 14][Baum Orsini Scholl Soria-Vazquez 20][Cohen Doerner K shelat 24][Baum Melissaris Rachuri Scholl 24]

Understudied in $t < n/2$ setting

[This work]

2-round honest majority BC-IA

inherent

Real-World Application: Threshold ECDSA

This work: Instantiate ECDSA-IA

Mechanism to guarantee
wellformedness of every sent message

Light ZK proofs in $\mathbb{G}$
+ verifiable complaints

Baseline security-with-abort protocol

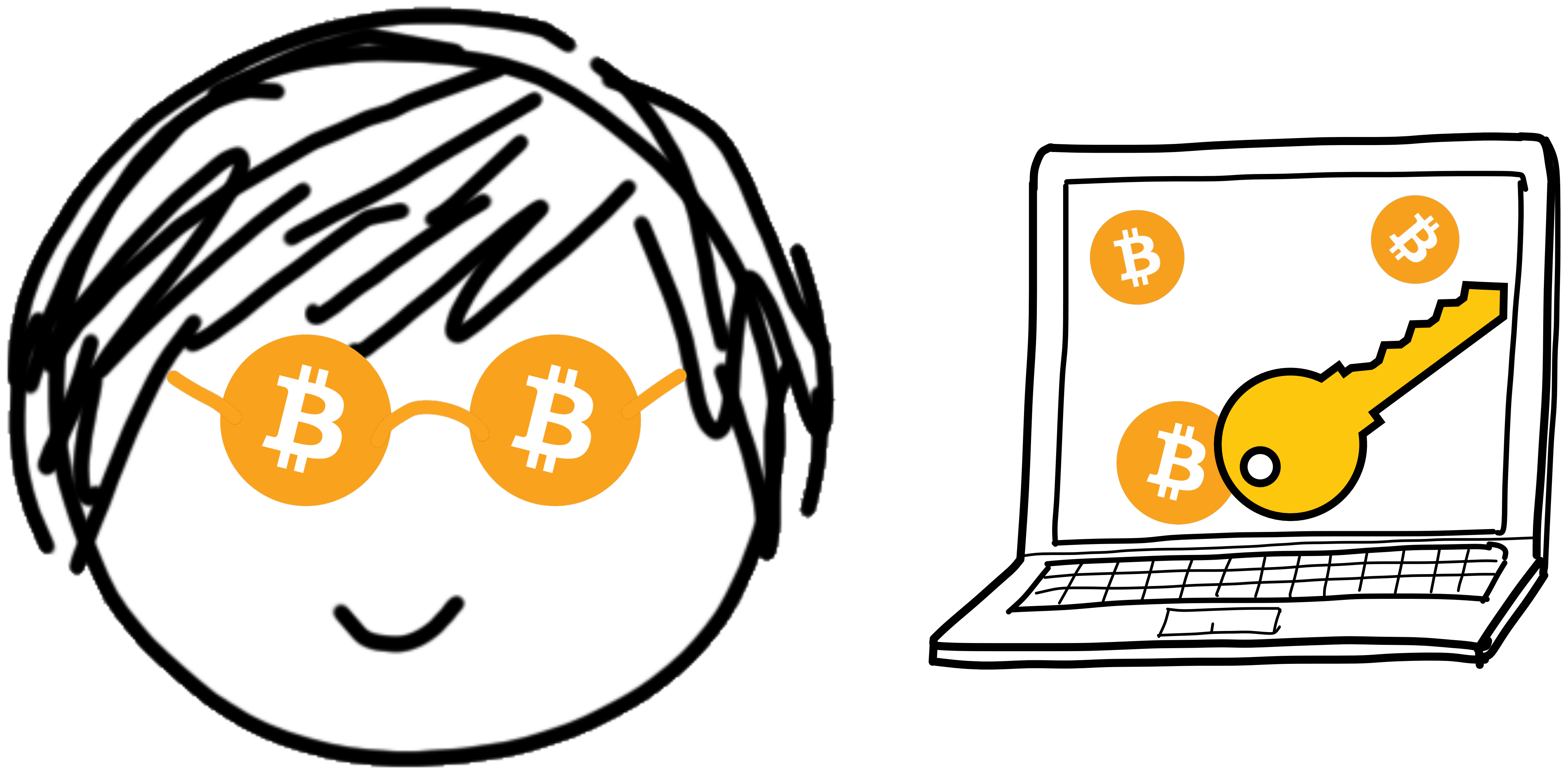
3-BC-round honest-majority
ECDSA signing à la [DKLs23]

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

[This work]
2-round honest majority BC-IA

inherent

Threshold Signing



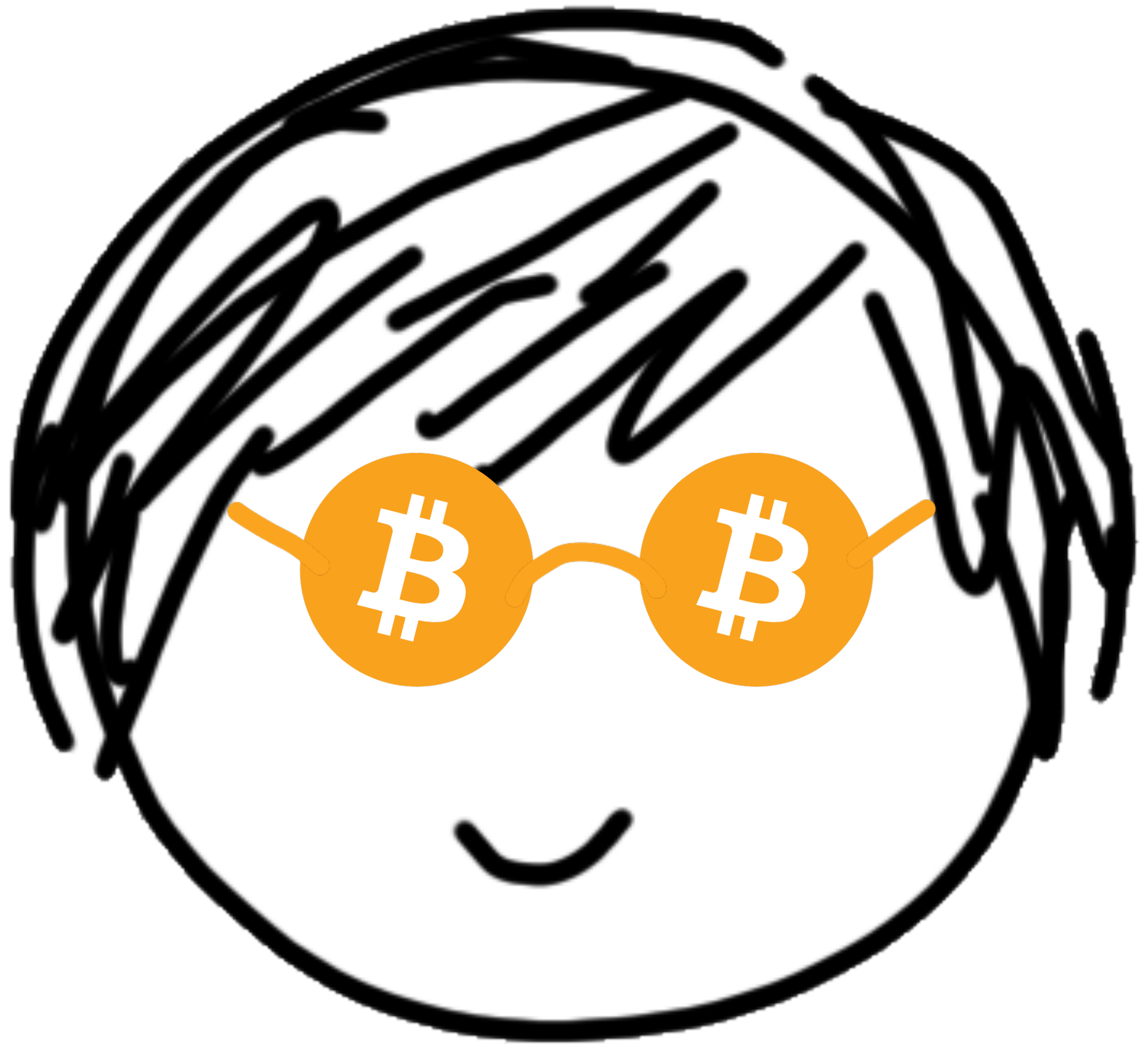
Spend ₿ by signing transactions

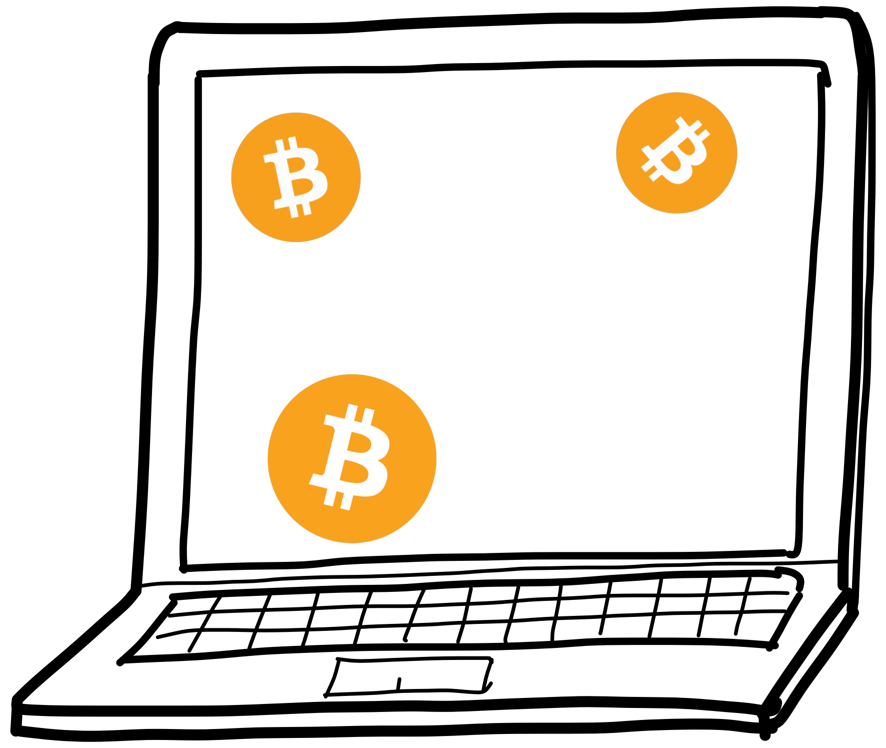
Signing key stored on laptop

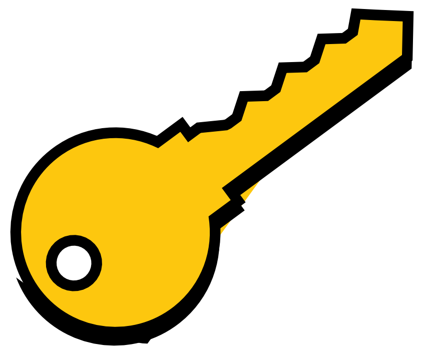
Laptop hacked ⇒ funds gone

Threshold Signing

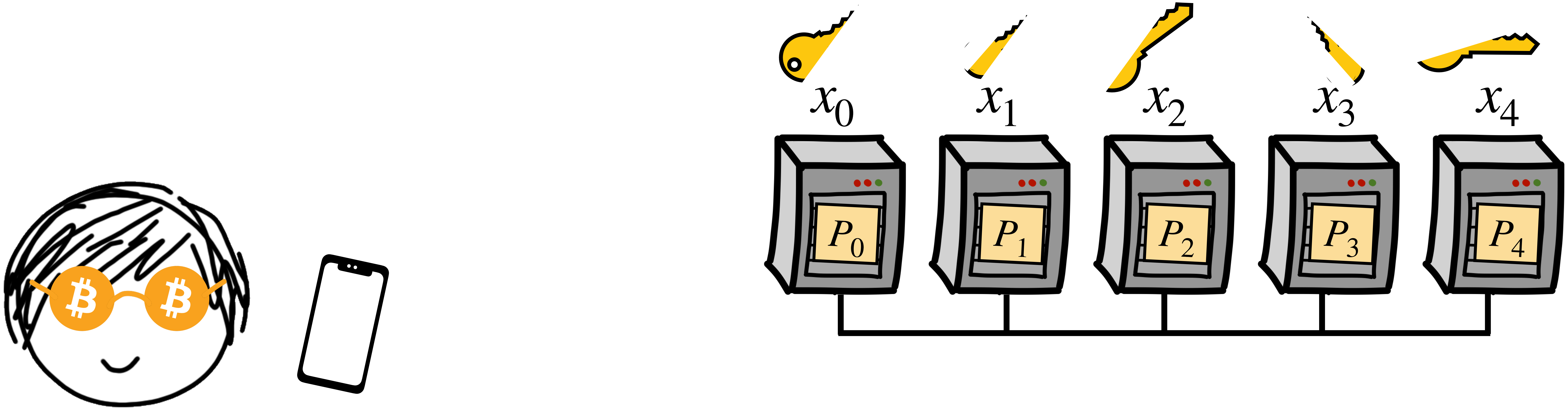
Threshold Signing



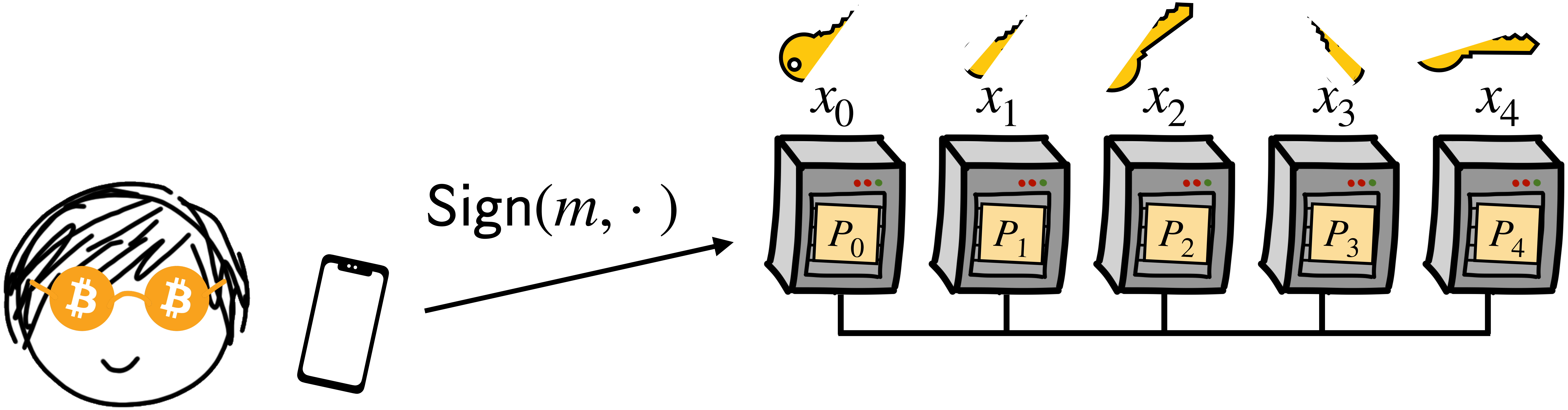




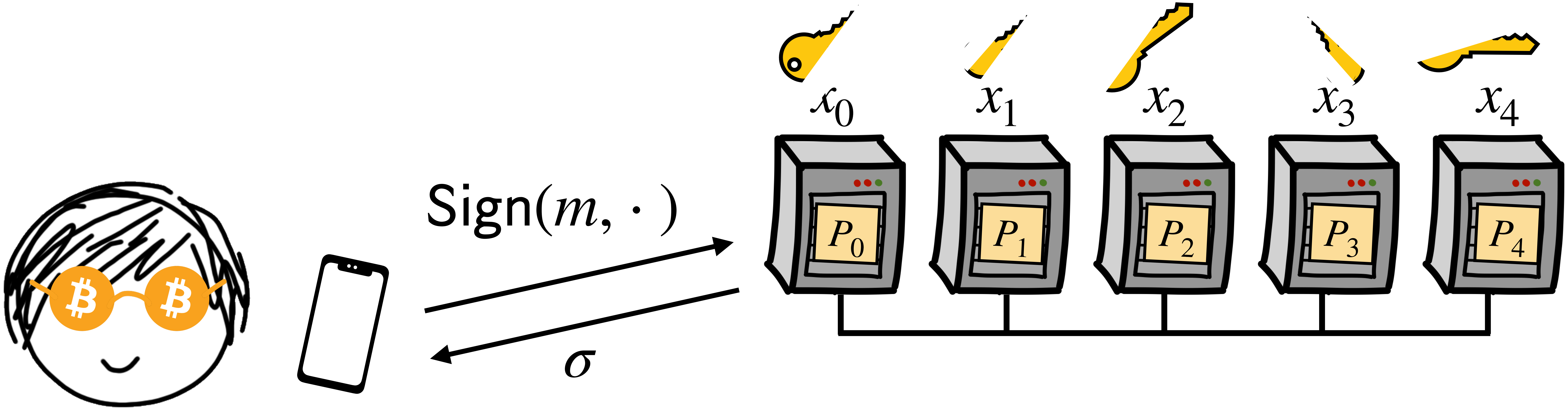
Threshold Signing



Threshold Signing



Threshold Signing



Distributed Risk: Attacker will need to compromise multiple devices

ECDSA

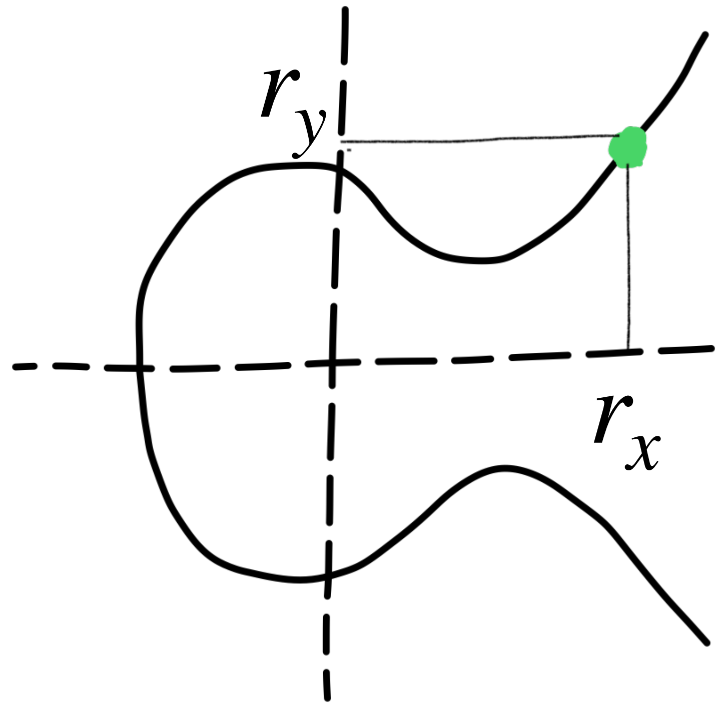
- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
- Devised by Scott Vanstone in 1992, standardised by NIST
- Widespread adoption across the internet
- Natural target for threshold signing

Threshold ECDSA: Structure

$$\begin{aligned}
&\mathrm{ECDSASign}(\mathsf{sk}, m):\\
&k \leftarrow \mathbb{Z}_q\\
&R = k \cdot G\\
&e = H(m)\\
&s = \frac{e + \mathsf{sk} \cdot r_x}{k}\\
&\text{output } \sigma = (s, R)
\end{aligned}$$

Threshold ECDSA: Structure

$R = (r_x, r_y)$



$\mathrm{ECDSASign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

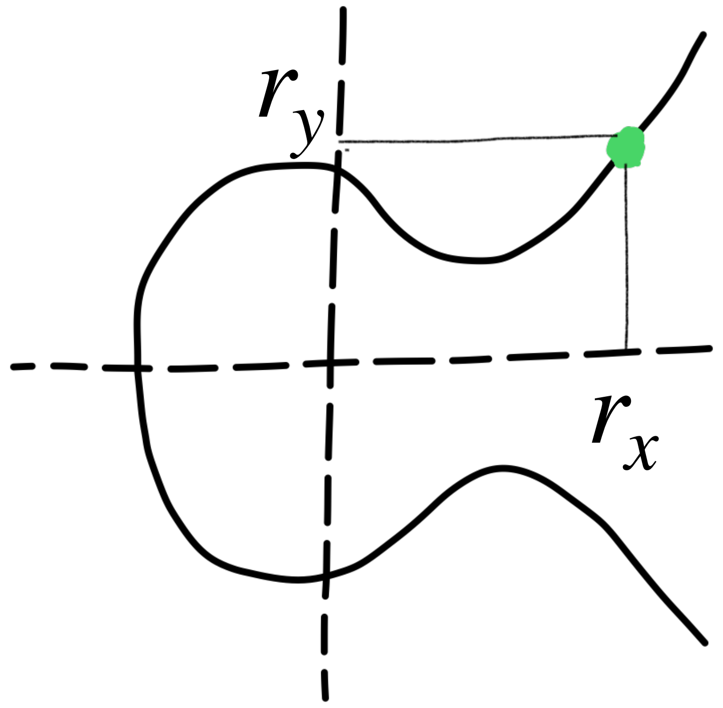
$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

r_x → x-coordinate of ***R*** (not secret)

output $\sigma = (s, R)$

Threshold ECDSA: Structure

$R = (r_x, r_y)$



$\mathrm{ECDSASign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

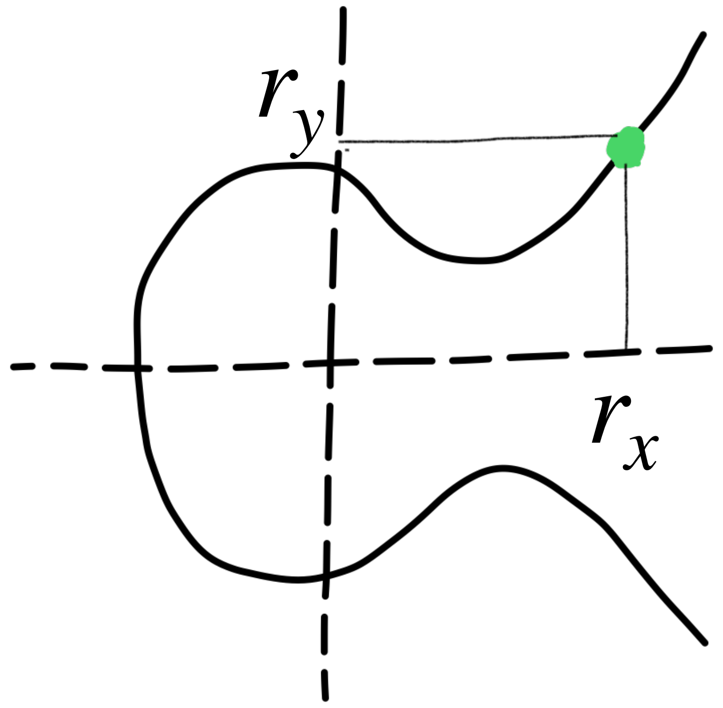
$$\boxed{s = \frac{e + \mathsf{sk} \cdot r_x}{k}}$$

x-coordinate of ***R*** (not secret)

output $\sigma = (s, R)$

Threshold ECDSA: Structure

$R = (r_x, r_y)$



$\mathsf{ECDSASign}(\mathsf{sk}, m)$:

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

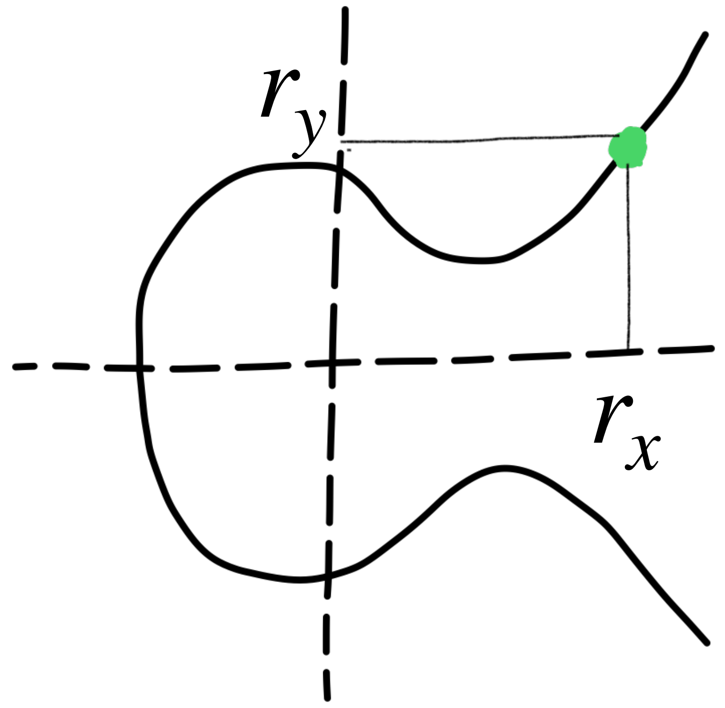
output $\sigma = (s, R)$

Multiplication of secret values

x-coordinate of $\boldsymbol{R}$ (not secret)

Threshold ECDSA: Structure

$R = (r_x, r_y)$



$\mathrm{ECDSASign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \boxed{\mathsf{sk}} \cdot r_x}{\boxed{k}}$$

output $\sigma = (s, R)$

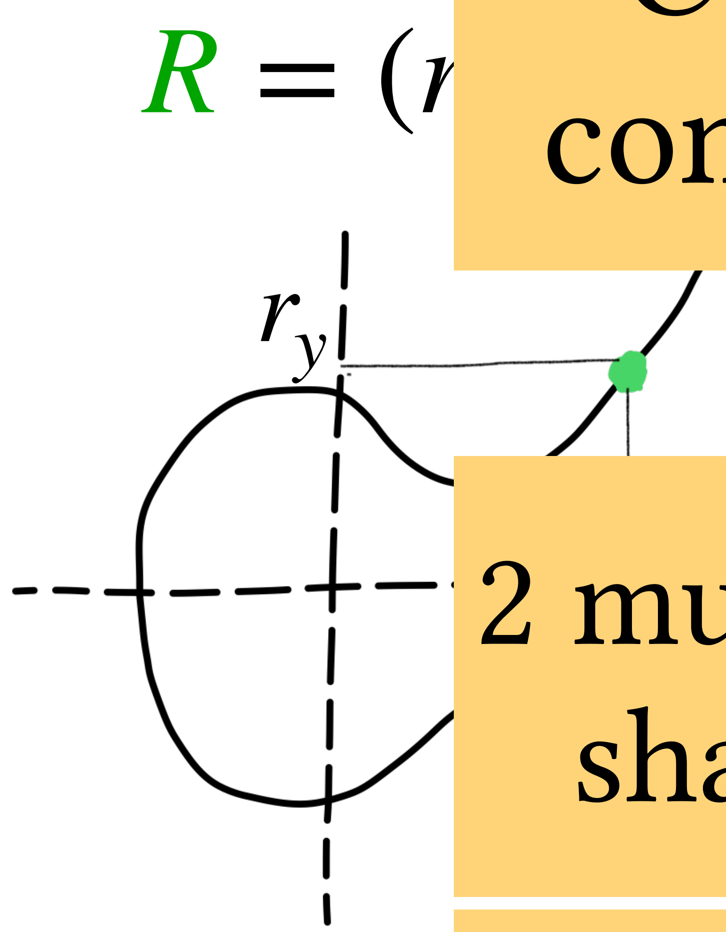
Multiplication of secret values

x-coordinate of ***R*** (not secret)

Division (Modular inverse)

Threshold ECDSA: Structure

ECDSASign(sk, m) :



Coin tossing: commit+release

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

2 mults: Ver. Secret sharing + NIZK

$$s = \frac{e+\mathsf{sk} \cdot r_x}{k}$$

Multiplication of secret values

x-coordinate of $\boldsymbol{R}$ (not secret)

Division (Modular inverse)

b'cast shares

output $\sigma = (s, R)$

Overall: 3 BC-IA rounds ⇒ 6 p2p rounds

ECDSA-IA: Efficiency

- Envisioned mode of operation:
  - Run [DKLs23] (sec w. abort) by default
  - Fall back to this protocol if too many aborts observed

- Worst case execution path most relevant to measuring efficiency
  - $(t, n) = (10{,}21)$: ~500ms compute time on standard hardware
    <u>Relative to dishonest majority</u>
    noticeably slower than (s.w.a.) OT-based ECDSA [DKLs23]
    order of magnitude faster than Paillier-based ECDSA-IA [CGGMP20]

- Actual worst-case performance depends on network conditions
  - Up to 6 × Network Timeout

In Conclusion

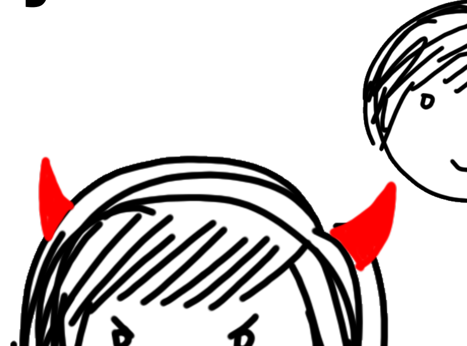
- Identifiable Abort can offer meaningful DoS-resistance
  (sometimes more desirable than Guaranteed Output)
  - IA requires some form of broadcast (tricky to instantiate)

- We define Broadcast-IA to certify cheaters: silent parties and protocol deviations
  - Prove *impossible* w. dishonest majority
  - 2-round $t < n/2$ construction over p2p channels (synchrony + PKI)

- Use this tool to instantiate Threshold ECDSA-IA over p2p channels
  - Simpler, more efficient than Guaranteed Output
  - <u>Ongoing research</u>: General Secure Function Evaluation with IA

Thanks!

Thanks **Eysa Lee** for



eprint coming soon, **(pre)preprint on ykondi.net**

Signing from ECDSA Tuples

[Abram Nof Orlandi Scholl Shlomovits 22]

$$[\mathsf{sk}] \quad [k] \quad [\phi] \quad [\phi \cdot k] \quad [\phi \cdot \mathsf{sk}]$$

Round 1

Round 2

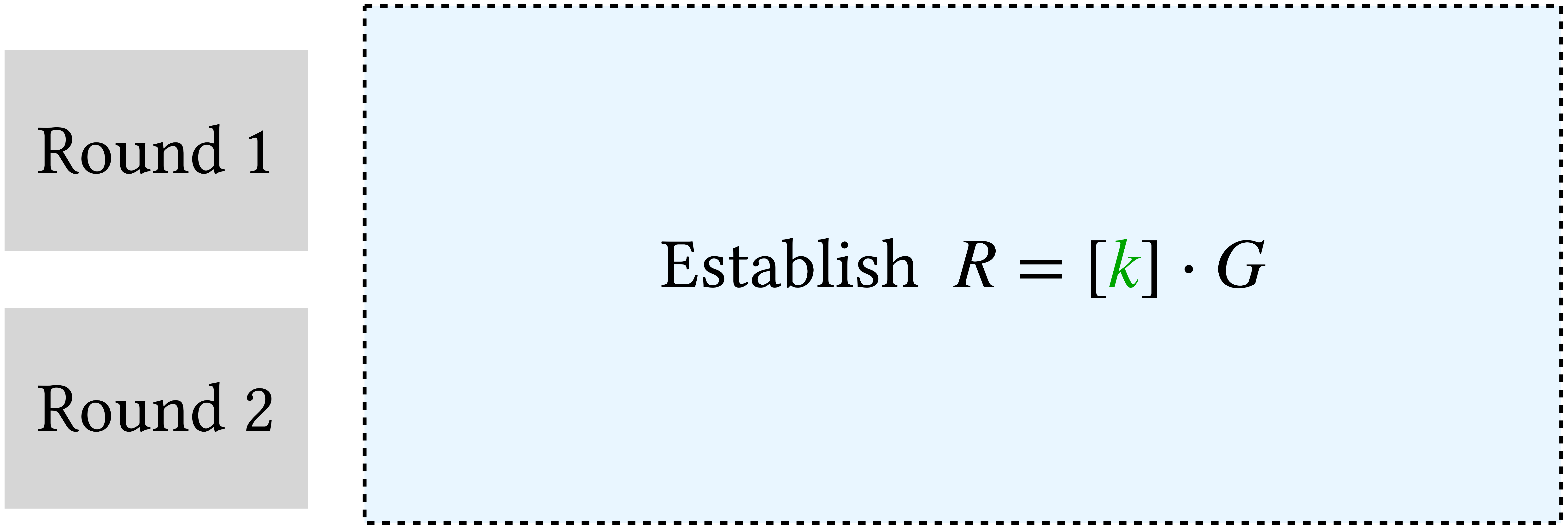
Establish $R = [k] \cdot G$

Round 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$
and $\beta = [\phi \cdot k]$

Output $(s = \alpha / \beta, R)$

Sampling ECDSA Tuples



$$[\mathsf{sk}] \quad [k] \quad [\phi] \quad [\phi \cdot k] \quad [\phi \cdot \mathsf{sk}]$$

Sampling ECDSA Tuples

Random string: $G_1, G_2 \in \mathbb{G}$ unknown DLog

BC-IA 1

Pedersen VSS: public deg-t poly $\mathsf{C} \in \mathbb{G}[X]$

Each P_i (should) hold $f(i), h(i) \in \mathbb{Z}_q$ s.t.

$$f(i)G_1 + h(i)G_2 = C(i)$$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$$[\mathsf{sk}] \quad [k] \quad [\phi] \quad [\phi \cdot k] \quad [\phi \cdot \mathsf{sk}]$$

Sampling ECDSA Tuples

Random string: $G_1, G_2 \in \mathbb{G}$ unknown DLog

BC-IA 1

Pedersen VSS: public deg-t poly $\mathsf{C} \in \mathbb{G}[X]$
Each P_i (should) hold $f(i), h(i) \in \mathbb{Z}_q$ s.t.

$$f(i)G_1 + h(i)G_2 = C(i)$$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1, H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

Local mult + rerandomize

$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$ DKG

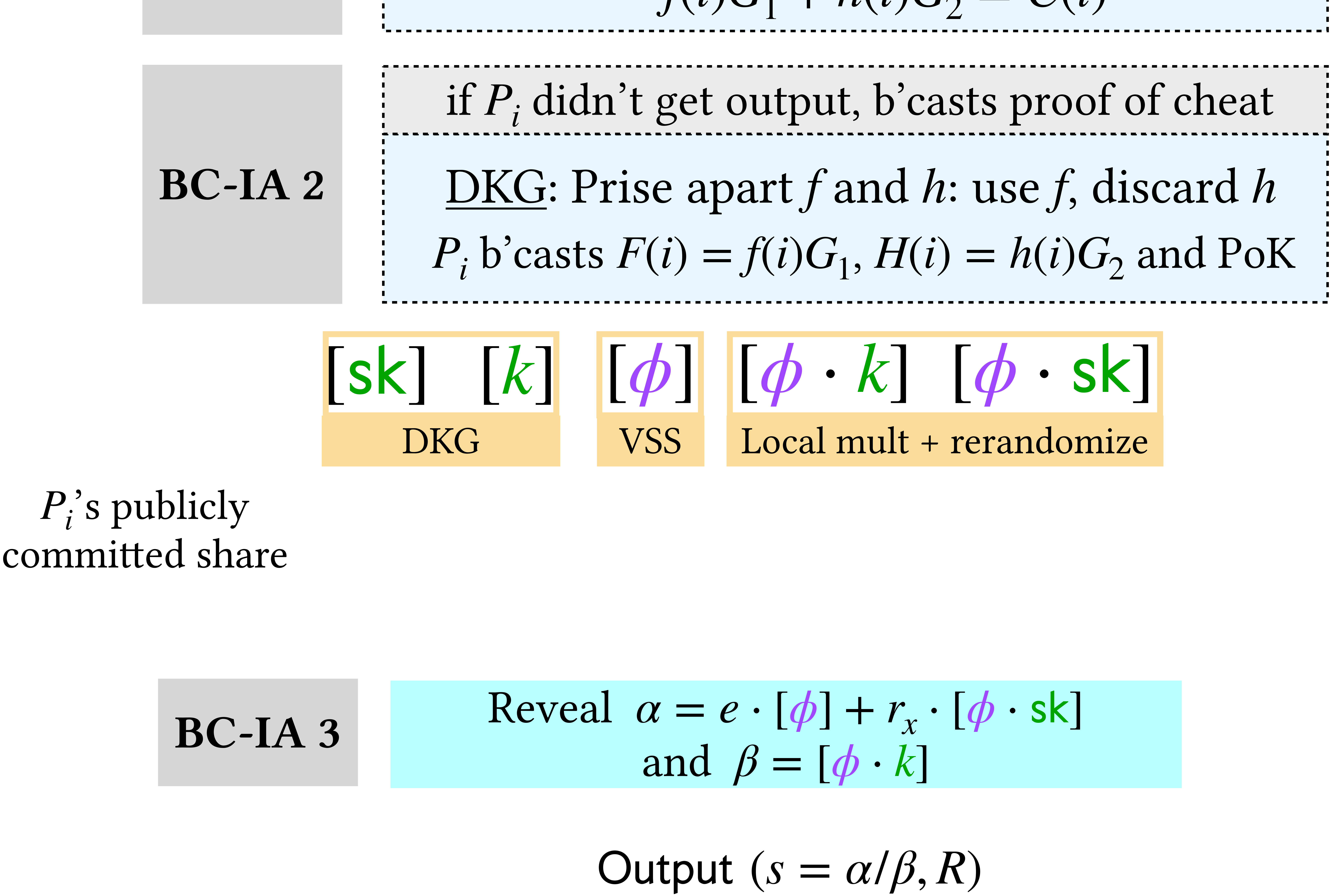
$[\phi]$ VSS

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$ Local mult + rerandomize

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$
and $\beta = [\phi \cdot k]$

Output $(s = \alpha/\beta, R)$



$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

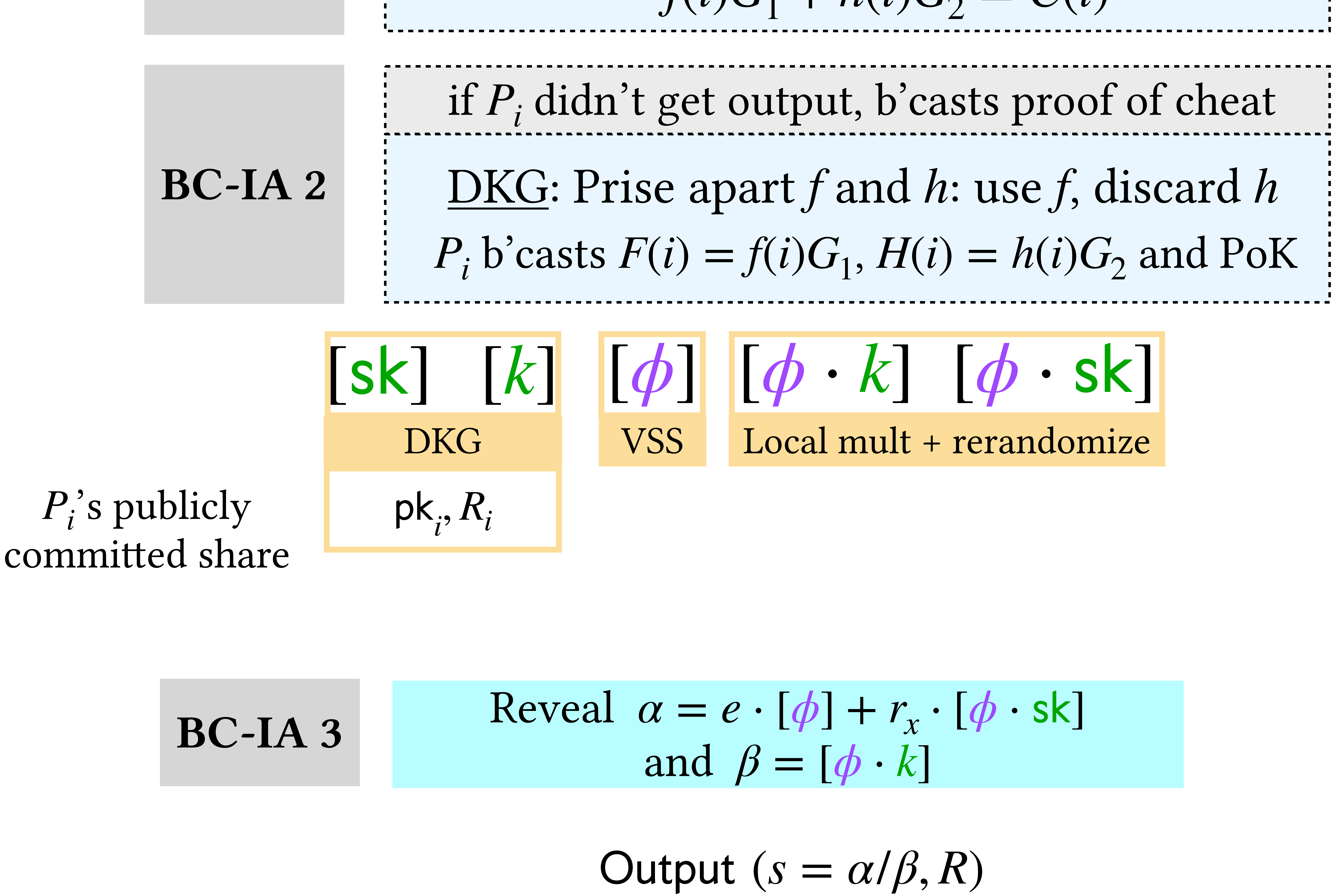
Local mult + rerandomize

P_i's publicly committed share

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$ and $\beta = [\phi \cdot k]$

Output $(s = \alpha/\beta, R)$



$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

Local mult + rerandomize

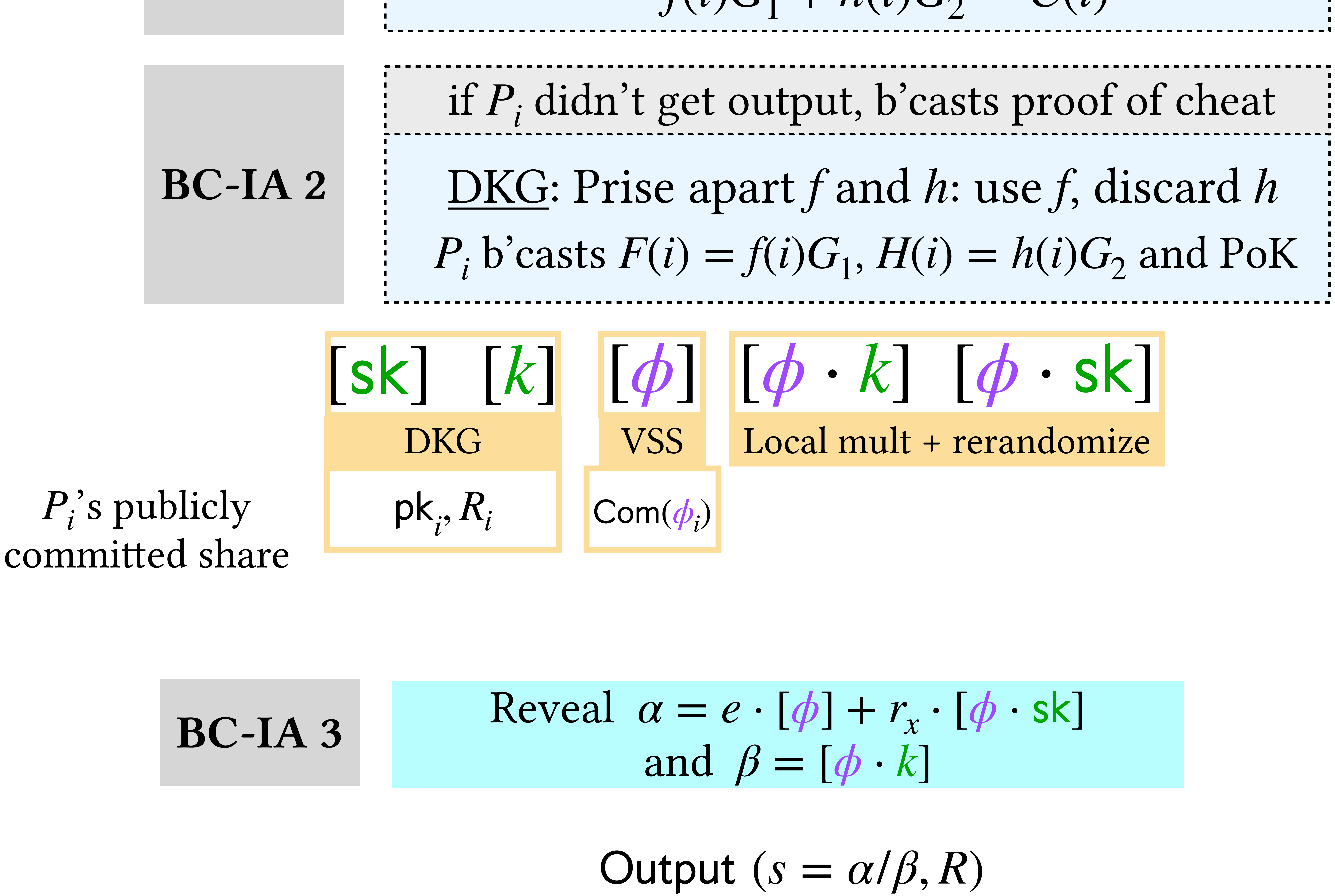
P_i's publicly committed share

pk_i, R_i

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$ and $\beta = [\phi \cdot k]$

Output $(s = \alpha/\beta, R)$



$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$

DKG

$[\phi]$

VSS

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

Local mult + rerandomize

P_i's publicly committed share

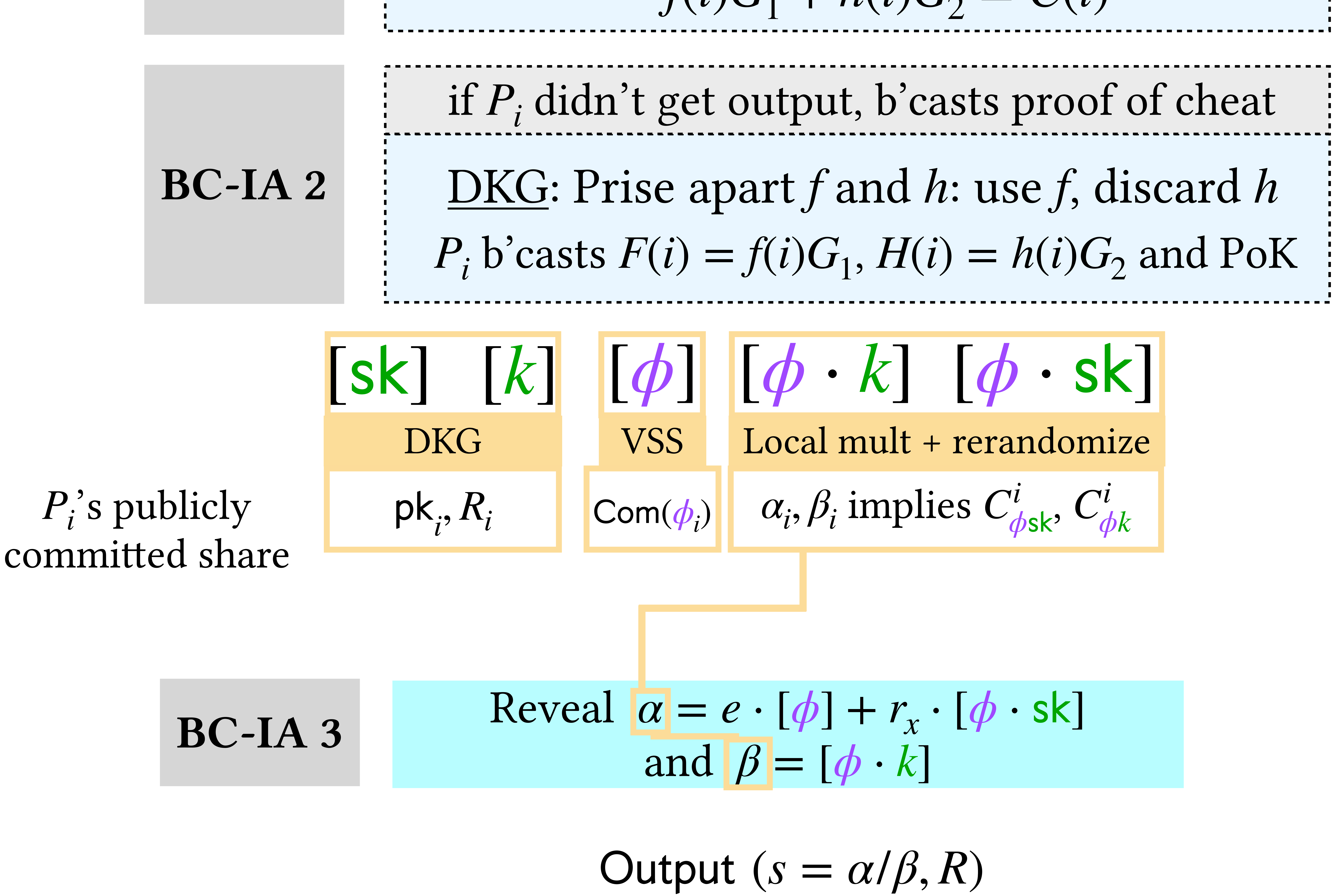
pk_i, R_i

$\mathsf{Com}(\phi_i)$

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$
and $\beta = [\phi \cdot k]$

Output $(s = \alpha/\beta, R)$



$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$ | $[\phi]$ | $[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

DKG | VSS | Local mult + rerandomize

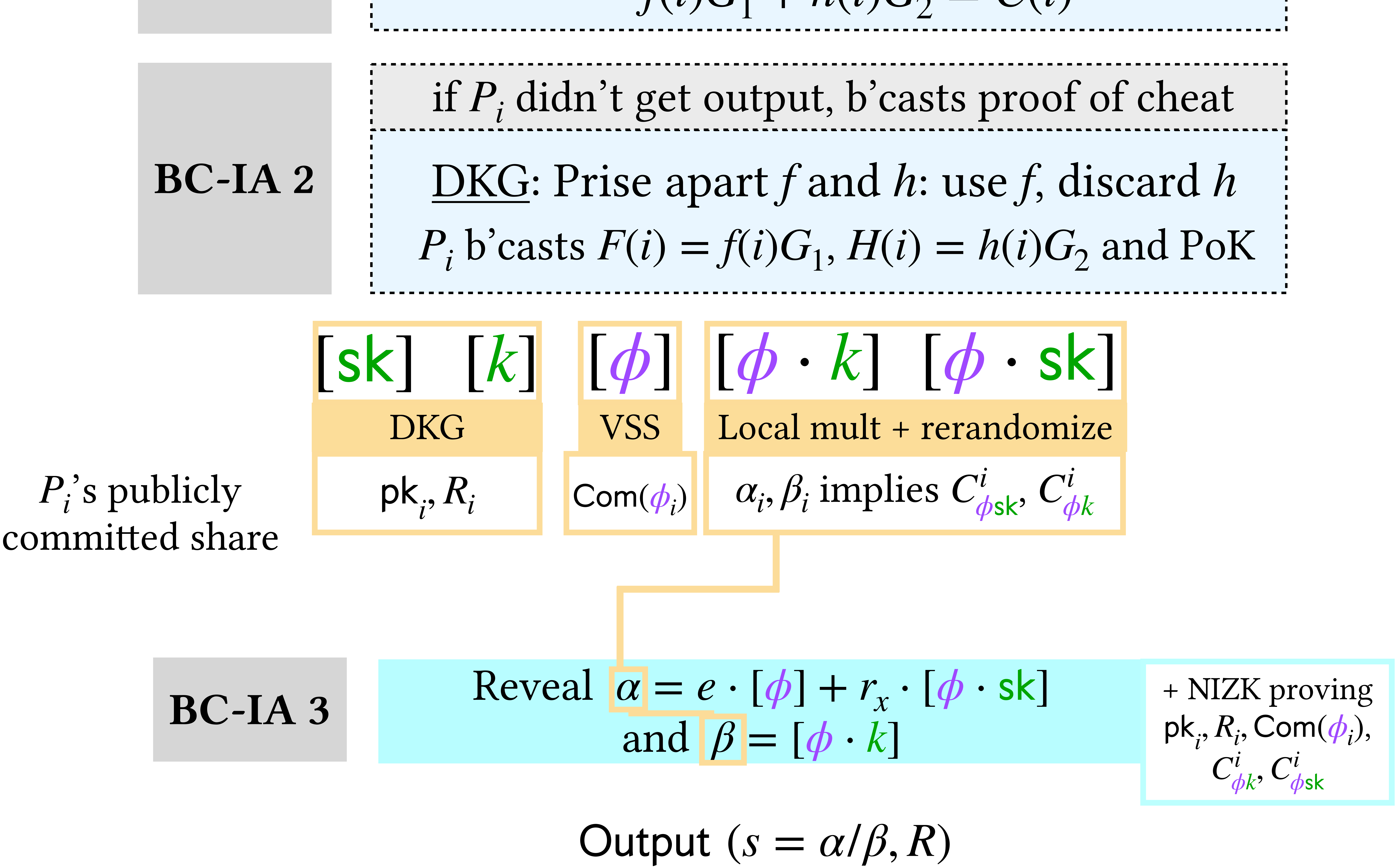
P_i's publicly committed share

pk_i, R_i | $\mathsf{Com}(\phi_i)$ | α_i, β_i implies $C^i_{\phi\mathsf{sk}}, C^i_{\phi k}$

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$
and $\beta = [\phi \cdot k]$

Output $(s = \alpha/\beta, R)$



$f(i)G_1 + h(i)G_2 = C(i)$

BC-IA 2

if P_i didn't get output, b'casts proof of cheat

DKG: Prise apart f and h: use f, discard h

P_i b'casts $F(i) = f(i)G_1$, $H(i) = h(i)G_2$ and PoK

$[\mathsf{sk}]$ $[k]$

DKG

pk_i, R_i

$[\phi]$

VSS

$\mathsf{Com}(\phi_i)$

$[\phi \cdot k]$ $[\phi \cdot \mathsf{sk}]$

Local mult + rerandomize

α_i, β_i implies $C^i_{\phi\mathsf{sk}}, C^i_{\phi k}$

P_i's publicly committed share

BC-IA 3

Reveal $\alpha = e \cdot [\phi] + r_x \cdot [\phi \cdot \mathsf{sk}]$ and $\beta = [\phi \cdot k]$

+ NIZK proving $\mathsf{pk}_i, R_i, \mathsf{Com}(\phi_i)$, $C^i_{\phi k}, C^i_{\phi \mathsf{sk}}$

Output $(s = \alpha/\beta, R)$