Formal Abstractions for Attested Execution Secure Processors

Eurocrypt

May 1st, 2017

Rafael Pass, Elaine Shi, Florian Tramèr

Trusted hardware:

Different communities, different world views

Trusted hardware:

Different communities, different world views

Crypto

Architecture + Systems & Security

Trusted hardware:
Different communities, different world views

Crypto

Architecture + Systems & Security

- "Minimal" trusted hardware to circumvent theoretical impossibilities
- Little concern about practical performance

Trusted hardware:
Different communities, different world views

Crypto

- "Minimal" trusted hardware to circumvent theoretical impossibilities
- Little concern about practical performance

Architecture + Systems & Security

- Trusted execution of "general-purpose" user-defined progs
- Cost-effectiveness, reusability, expressivity

Architecture community converged on “attested execution”

Bastion

GhostRider

Iso-X

Ascend

Sanctum

Aegis

XOM

Phantom

Academia

TPM

Intel® SGX

TrustZone®
Security Foundation by ARM®

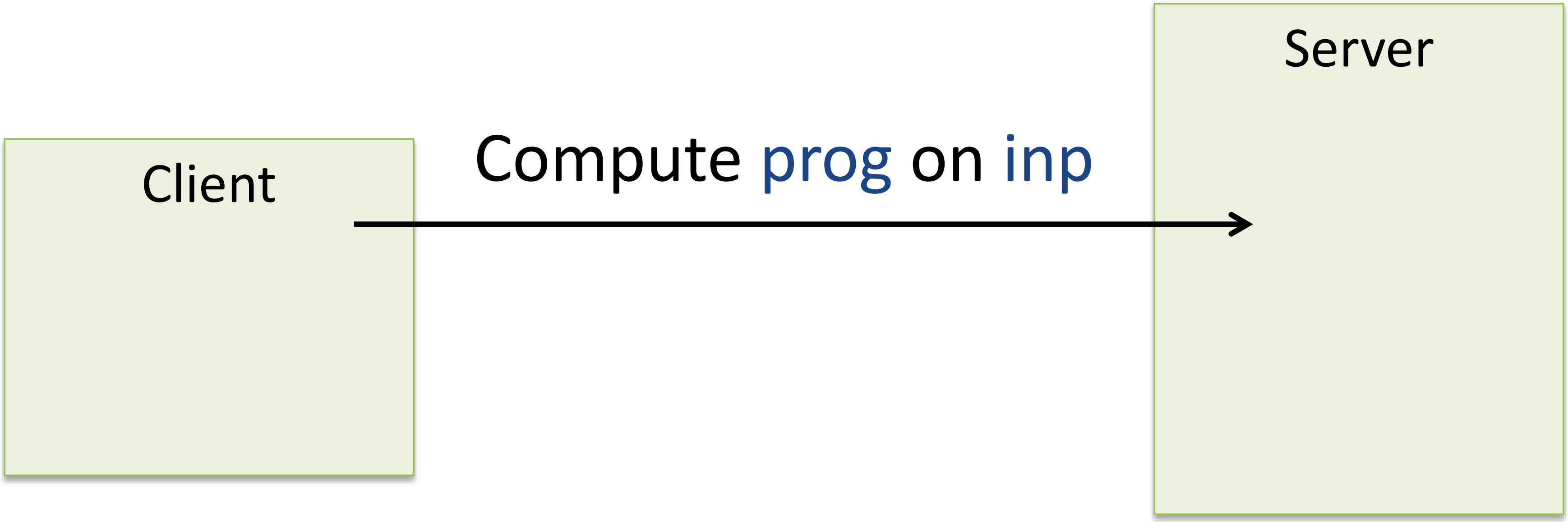
Industry

Architecture community converged on "attested execution"

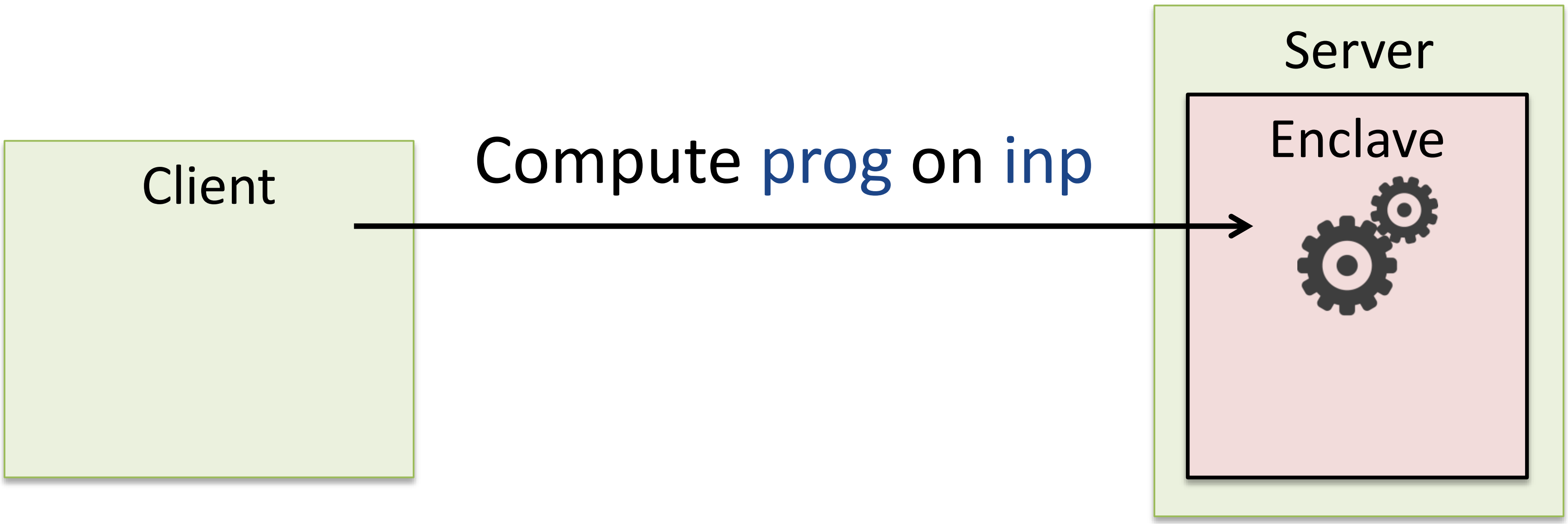
What is "attested execution" ?

What can it (not) express?

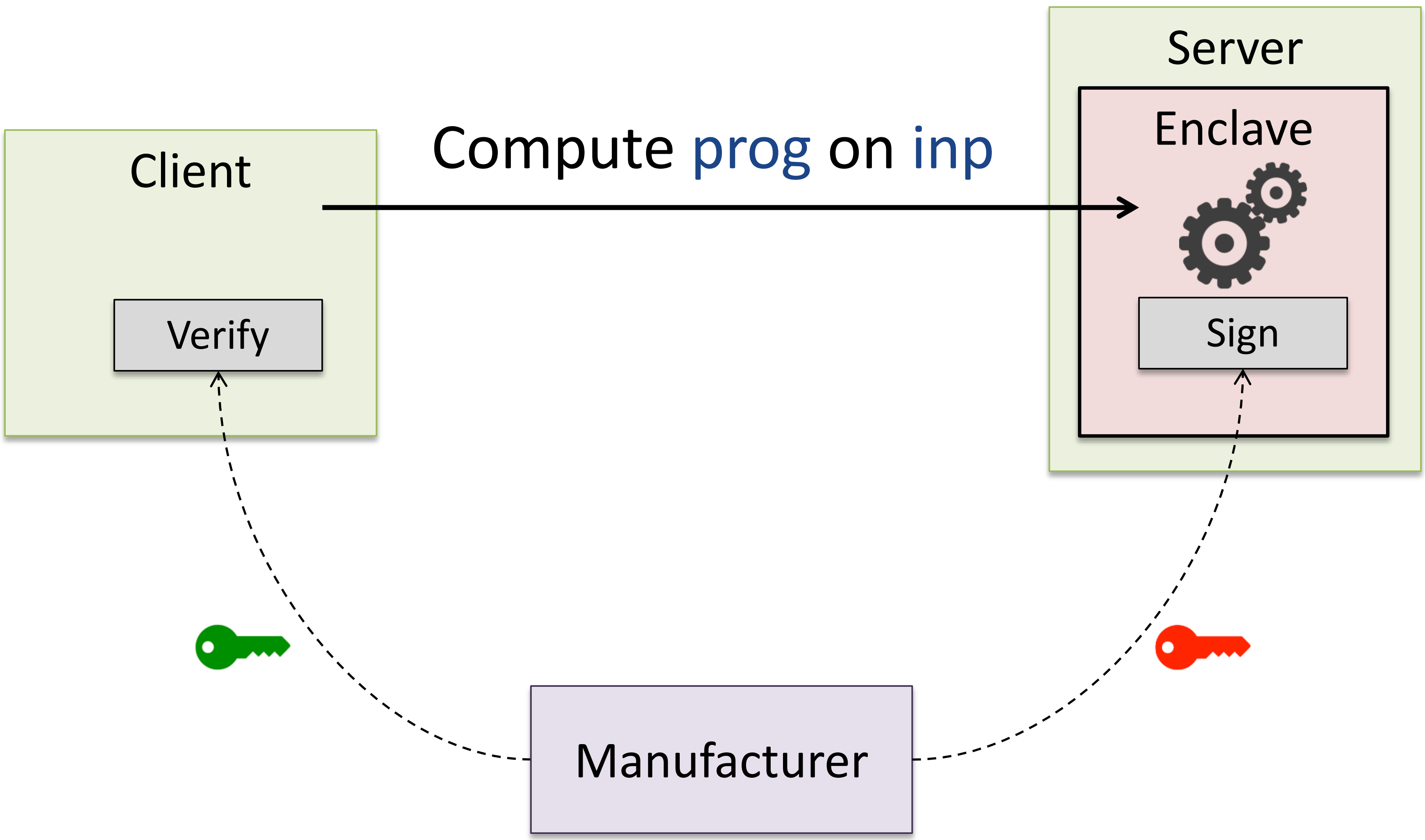
Attested Execution



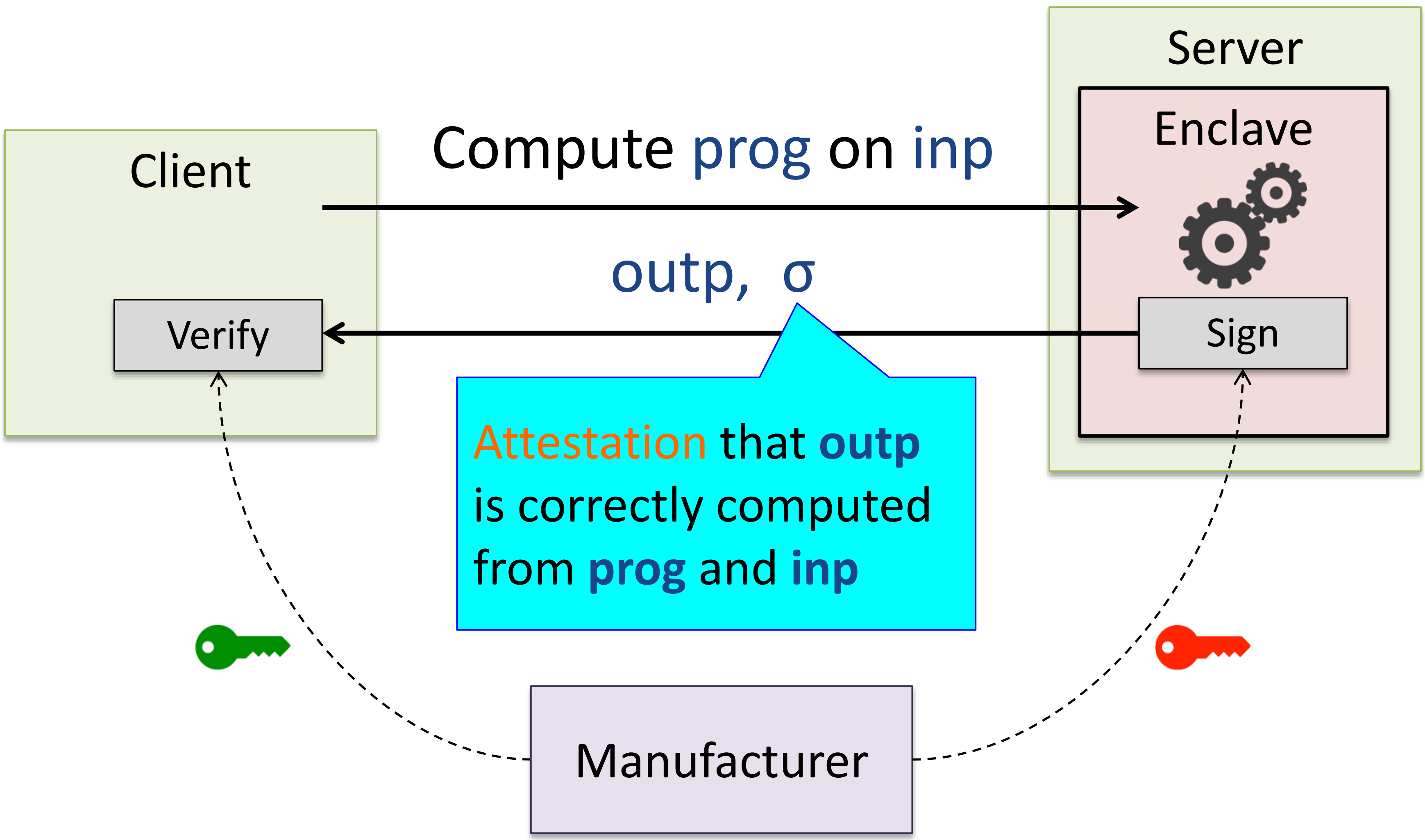
Attested Execution



Attested Execution



Attested Execution



Why Ideal Abstractions?

Why Ideal Abstractions?

- Formal security proofs for implementations from precise abstractions and security models

Why Ideal Abstractions?

- Formal security proofs for implementations from precise abstractions and security models

- **Ultimate Goal:** Formally verified processor implementing this formal abstraction

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

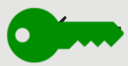
$$\boldsymbol{\mathcal{G}_{\text{att}}[\Sigma, \text{reg}]}$$

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\boldsymbol{\mathcal{G}_{\mathrm{att}}[\Sigma, \mathrm{reg}]}$

init():   $\leftarrow \Sigma.\mathrm{KeyGen}(1^{\lambda})$

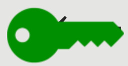
getpk() from P: send  to P

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{att}[\Sigma, \mathrm{reg}]$

init():   $\leftarrow \Sigma.\mathrm{KeyGen}(1^{\lambda})$

getpk() from P: send  to P

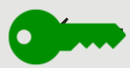
install(prog, sid) from P ∈ reg:

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{\mathrm{att}}[\Sigma, \mathrm{reg}]$

init():   Σ.KeyGen(1^λ)

getpk() from P: send  to P

install(prog, sid) from P ∈ reg:

enclave id (nonce)

enclave memory

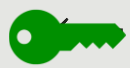
(eid, P)	(sid, prog, M)
…	…

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{\text{att}}[\Sigma, \text{reg}]$

init():   Σ.KeyGen(1^λ)

getpk() from P: send  to P

install(prog, sid) from P ∈ reg:

resume(eid, inp) from P ∈ reg:

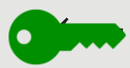
enclave id (nonce)	enclave memory
(eid, P)	(sid, prog, M)
…	…

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{\mathrm{att}}[\Sigma, \mathrm{reg}]$

init():   $\Sigma.\mathrm{KeyGen}(1^\lambda)$

getpk() from P: send  to P

install(prog, sid) from P ∈ reg:

resume(eid, inp) from P ∈ reg:

(out, M') = prog(inp, M)

enclave id (nonce) | enclave memory

(eid, P)	(sid, prog, M)
…	…

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{att}[\Sigma, \text{reg}]$

init(): $\leftarrow$ Σ.KeyGen(1^{λ})

getpk() from P: send to P

install(prog, sid) from P ∈ reg:

enclave id (nonce)

enclave memory

(eid, P)	(sid, prog, M′)
…	…

resume(eid, inp) from P ∈ reg:

(out, M') = *prog(inp, M)*

Formal Model

Signature scheme

Registry of all platforms with trusted hardware

$\mathcal{G}_{att}[\Sigma, \text{reg}]$

init(): Σ.KeyGen(1^λ)

getpk() from P: send to P

enclave id (nonce)

enclave memory

install(prog, sid) from P ∈ reg:

(eid, P)	(sid, prog, M′)
…	…

resume(eid, inp) from P ∈ reg:

(out, M′) = prog(inp, M)

σ = Σ.Sign(, eid, sid, prog, out)

send (out, σ) *to P*

Composability with Global State

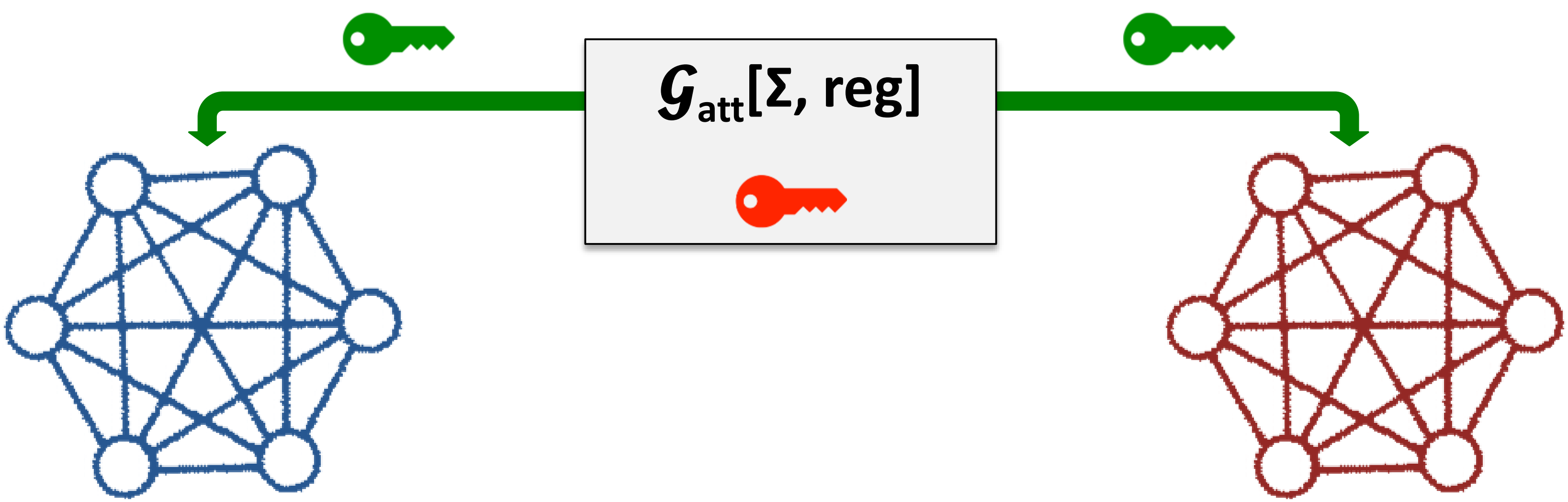
Composability with Global State

Model $\mathcal{G}_{\text{att}}$ as *global* ideal functionality [CDPW'07]

Composability with Global State

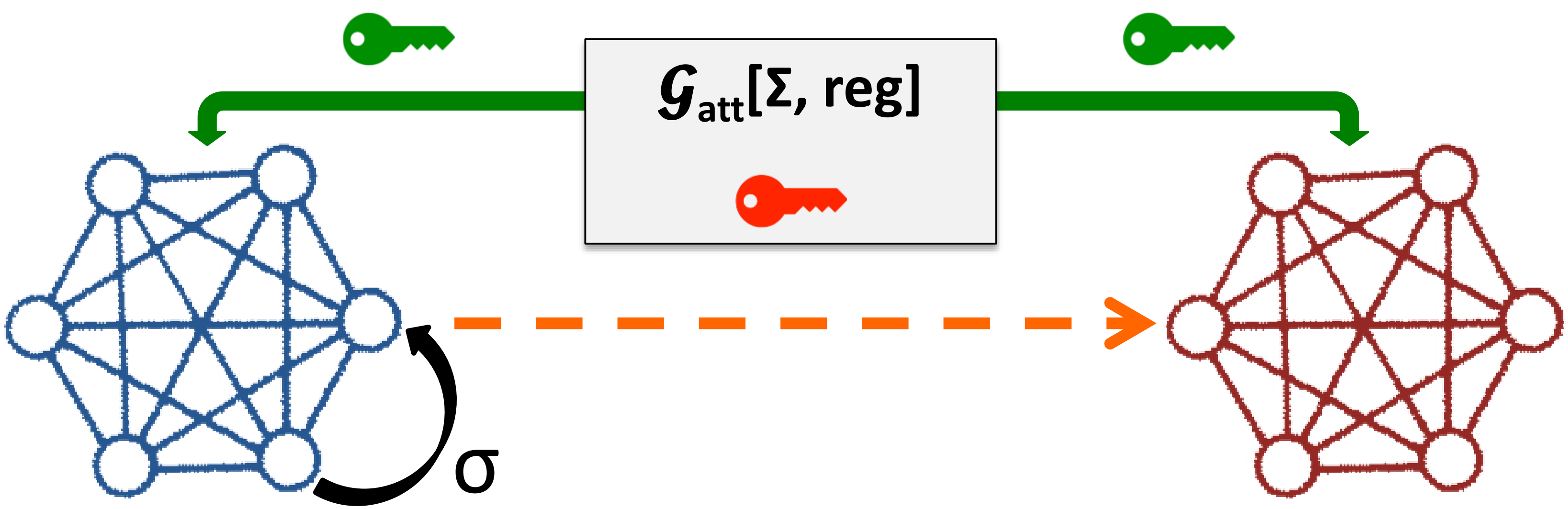
Model $\mathcal{G}_{\text{att}}$ as *global* ideal functionality [CDPW'07]

Attestation key is ***shared*** across protocols



Composability with Global State

Model $\mathcal{G}_{att}$ as *global* ideal functionality [CDPW'07]

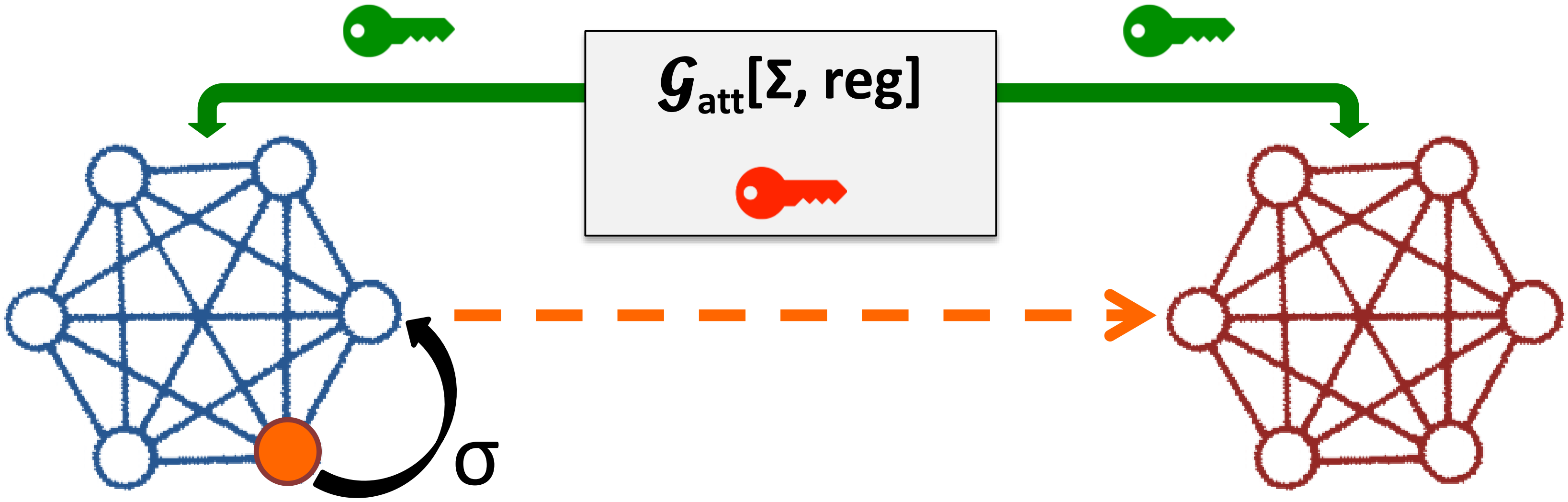


Composability with Global State

Model $\mathcal{G}_{\text{att}}$ as *global* ideal functionality [CDPW'07]

Example of concrete security issue:

Non-deniability for parties in **reg**



The more interesting question

What is “attested execution” ?

What can it **(not)** express?

The good

Powerful Abstraction!

The good

Powerful Abstraction!

$\mathcal{G}_{\text{att}} \rightarrow$ "Stateful Obfuscation"

Impossible even with stateless tokens and cryptographic obfuscation

The good

Powerful Abstraction!

$\mathcal{G}_{\text{att}}$ ➔ "Stateful Obfuscation"

Impossible even with stateless tokens and cryptographic obfuscation

The surprise

UC-Secure MPC?

The good

The surprise

Powerful Abstraction!

$\mathcal{G}_{\mathrm{att}} \rightarrow$ "Stateful Obfuscation"

Impossible even with stateless tokens and cryptographic obfuscation

UC-Secure MPC?

☑ It's Complicated

The good

Powerful Abstraction!

G_{att} → "Stateful Obfuscation"

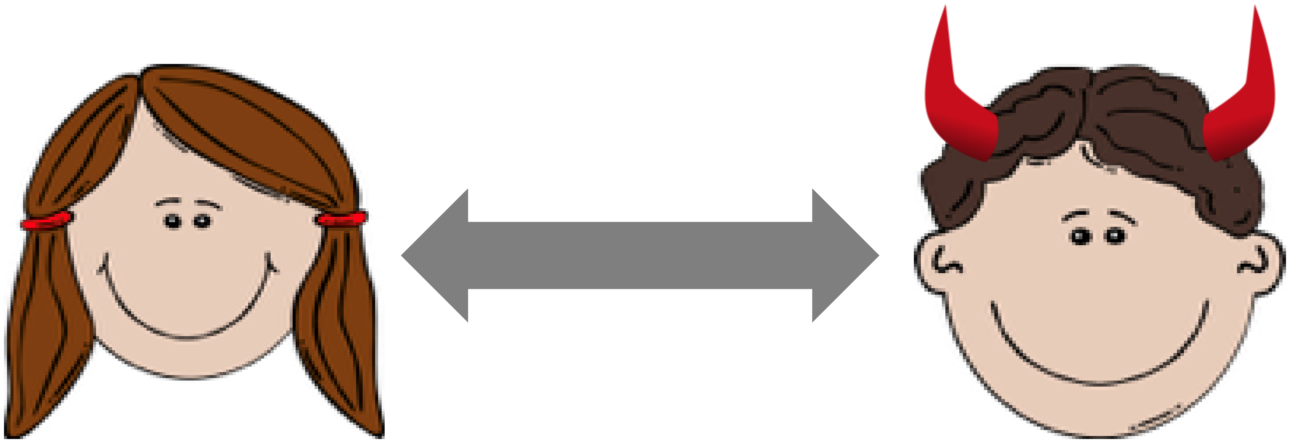
Impossible even with stateless tokens and cryptographic obfuscation

The surprise

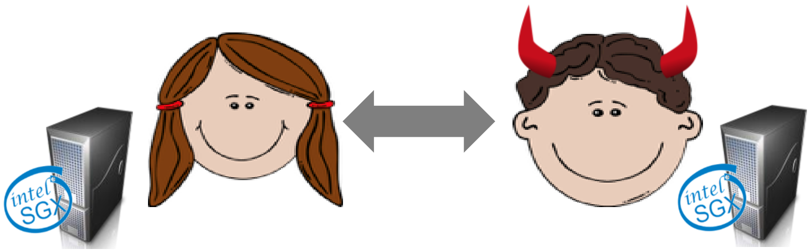
UC-Secure MPC?

☑ It's Complicated

Consider 2PC

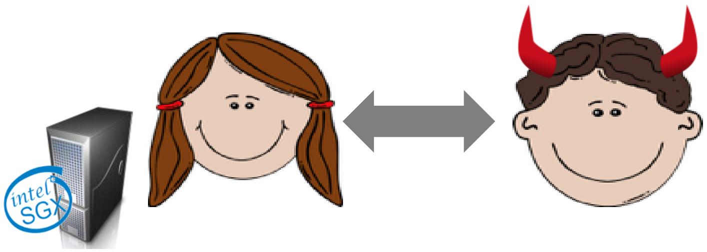


Consider 2PC



UC-secure 2PC possible if both parties have trusted hardware

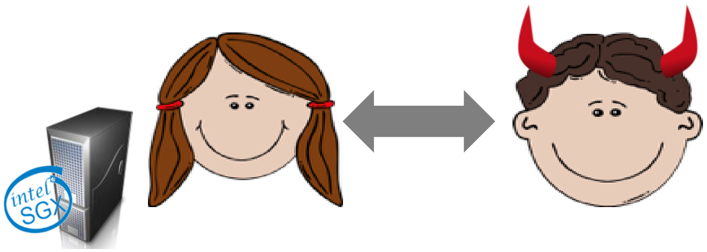
Consider 2PC



UC-secure 2PC possible if both parties have trusted hardware

Impossible if only one party has trusted hardware!

Consider 2PC



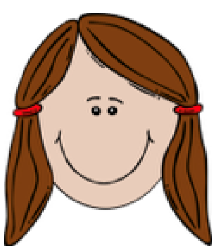
This is counter-intuitive.

Impossible if only one party has trusted hardware!

Issue: non-deniability

Issue: non-deniability

Convinced that
some honest party in the registry
participated in the protocol

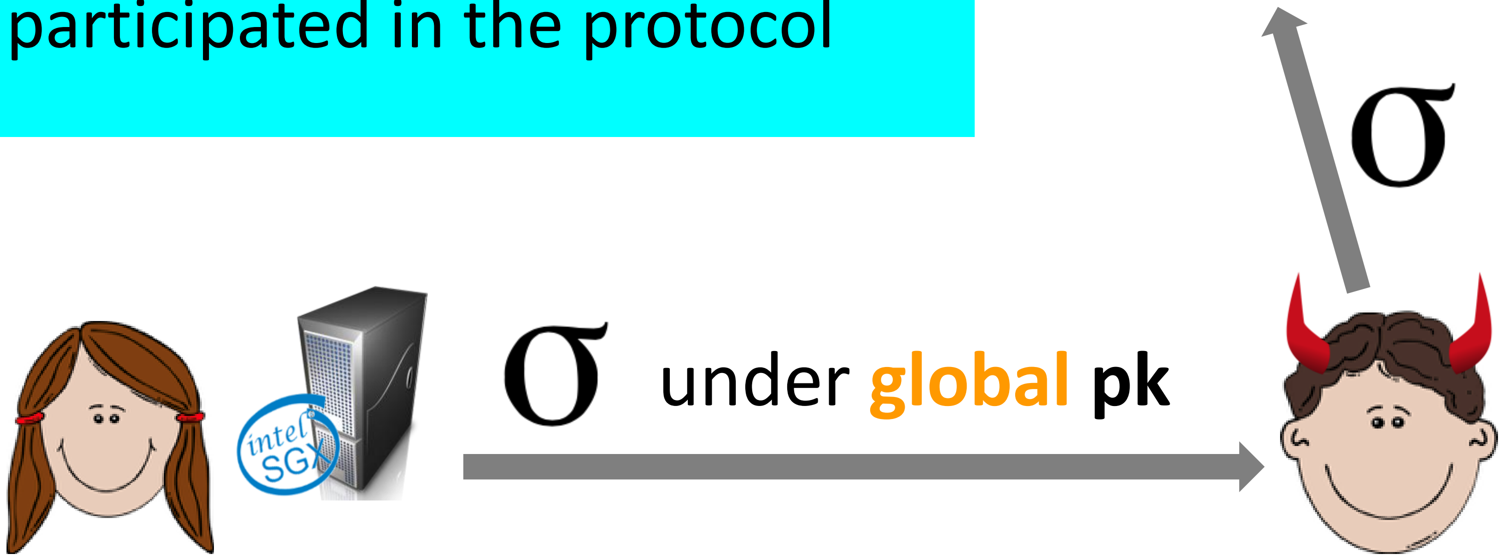


σ under **global pk**

σ

Non-issue if all nodes have trusted hardware
or if pk isn't global

Convinced that
some honest party in the registry
participated in the protocol



What if we **really really** want to use a single trusted processor?

What if we **really really** want to use a single trusted processor?

Extra setup assumption: Augmented CRS

What if we **really really** want to use a single trusted processor?

Extra setup assumption: Augmented CRS

UC-Secure MPC with *O(1) crypto operations*

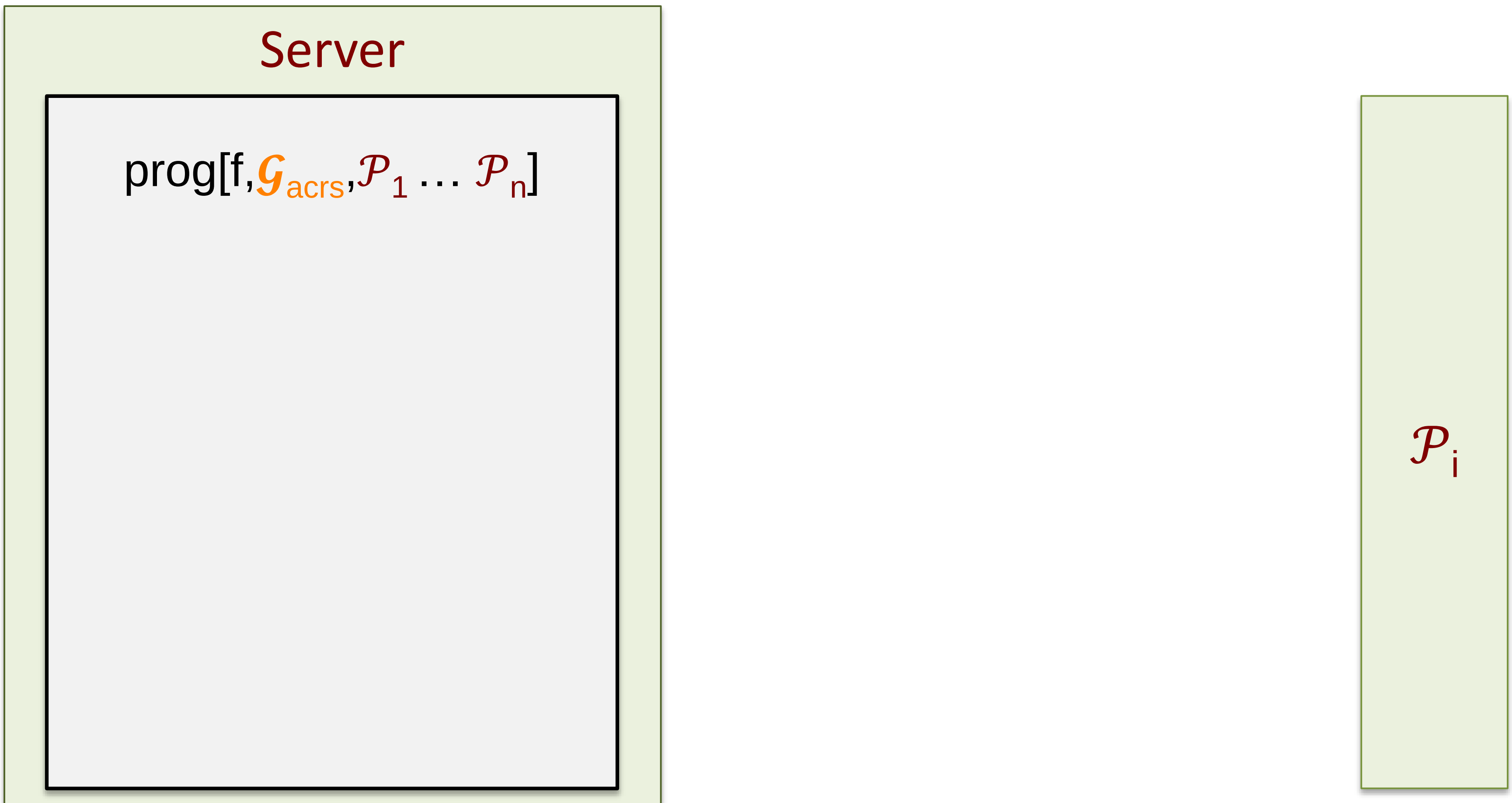
What if we **really really** want to use a single trusted processor?

Extra setup assumption: Augmented CRS

UC-Secure MPC with *O(1) crypto operations*

Backdoor enclave program: allow simulator to *extract inputs* and *program the outputs* for corrupt parties

What if we **really really** want to use a single trusted processor?



What if we **really really** want to use a single trusted processor?

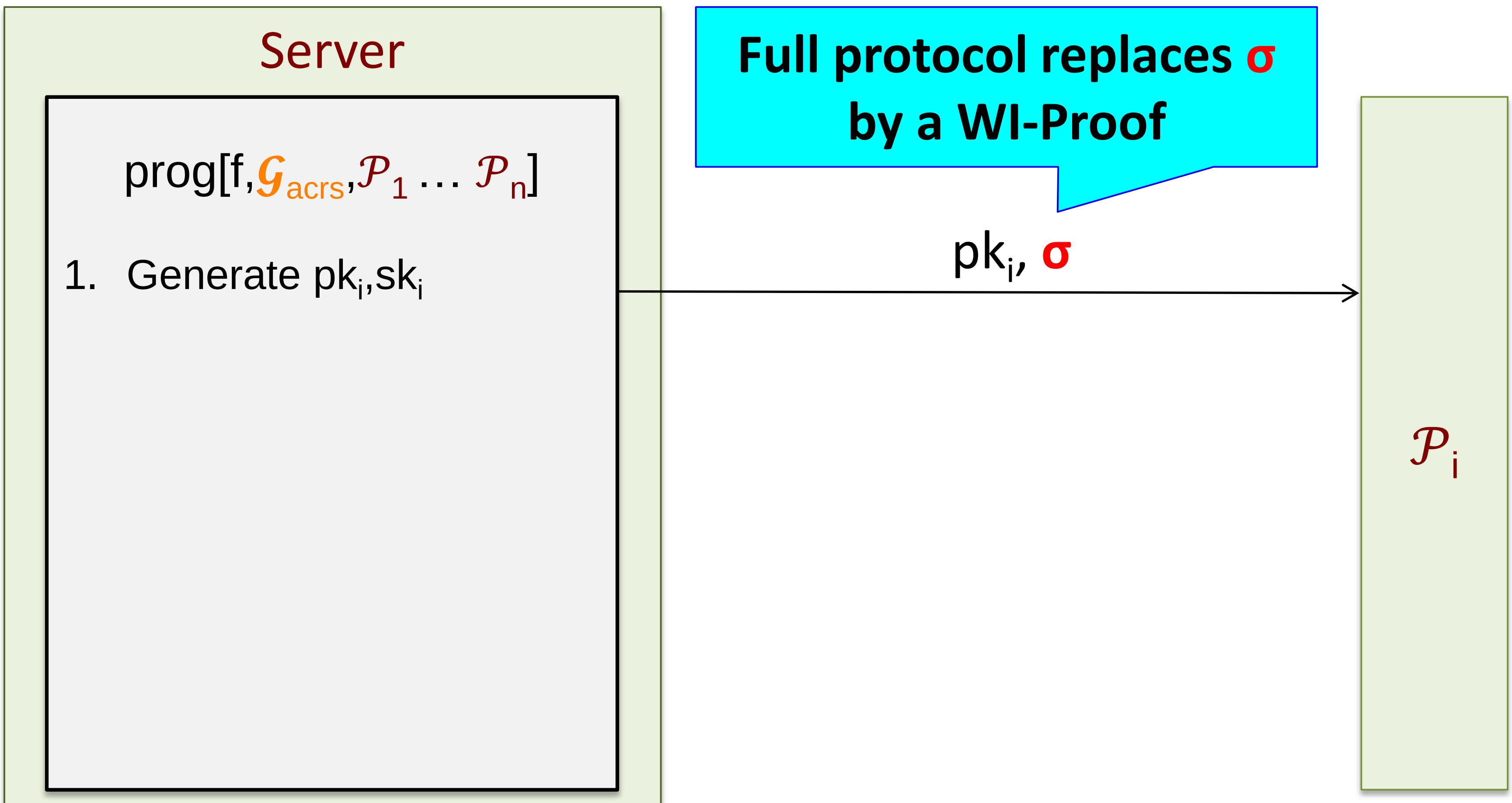
Server

prog[f,$\mathcal{G}_{acrs}$,$\mathcal{P}_1$ … $\mathcal{P}_n$]

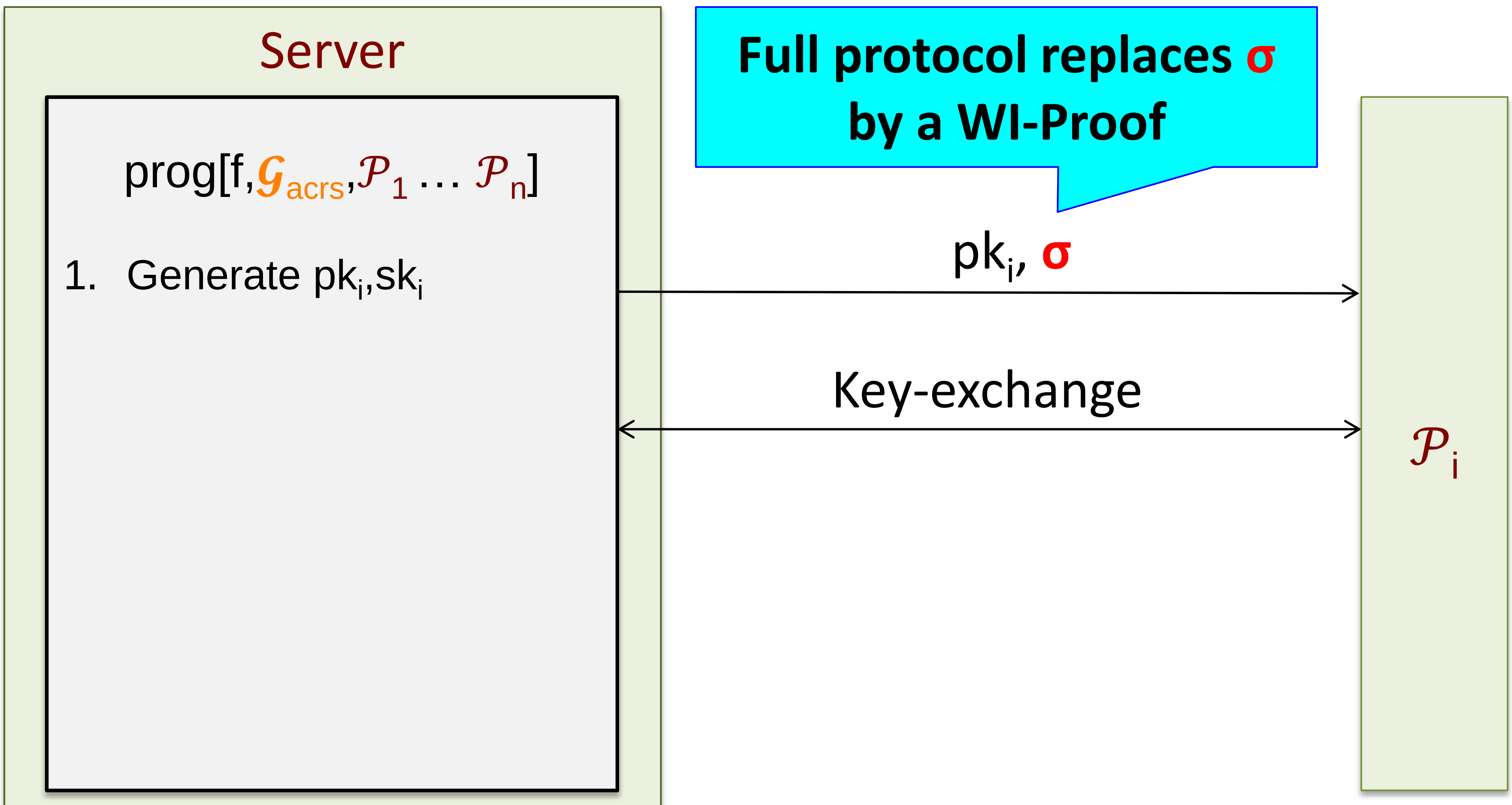
1. Generate pk_i,sk_i

$\mathcal{P}_i$

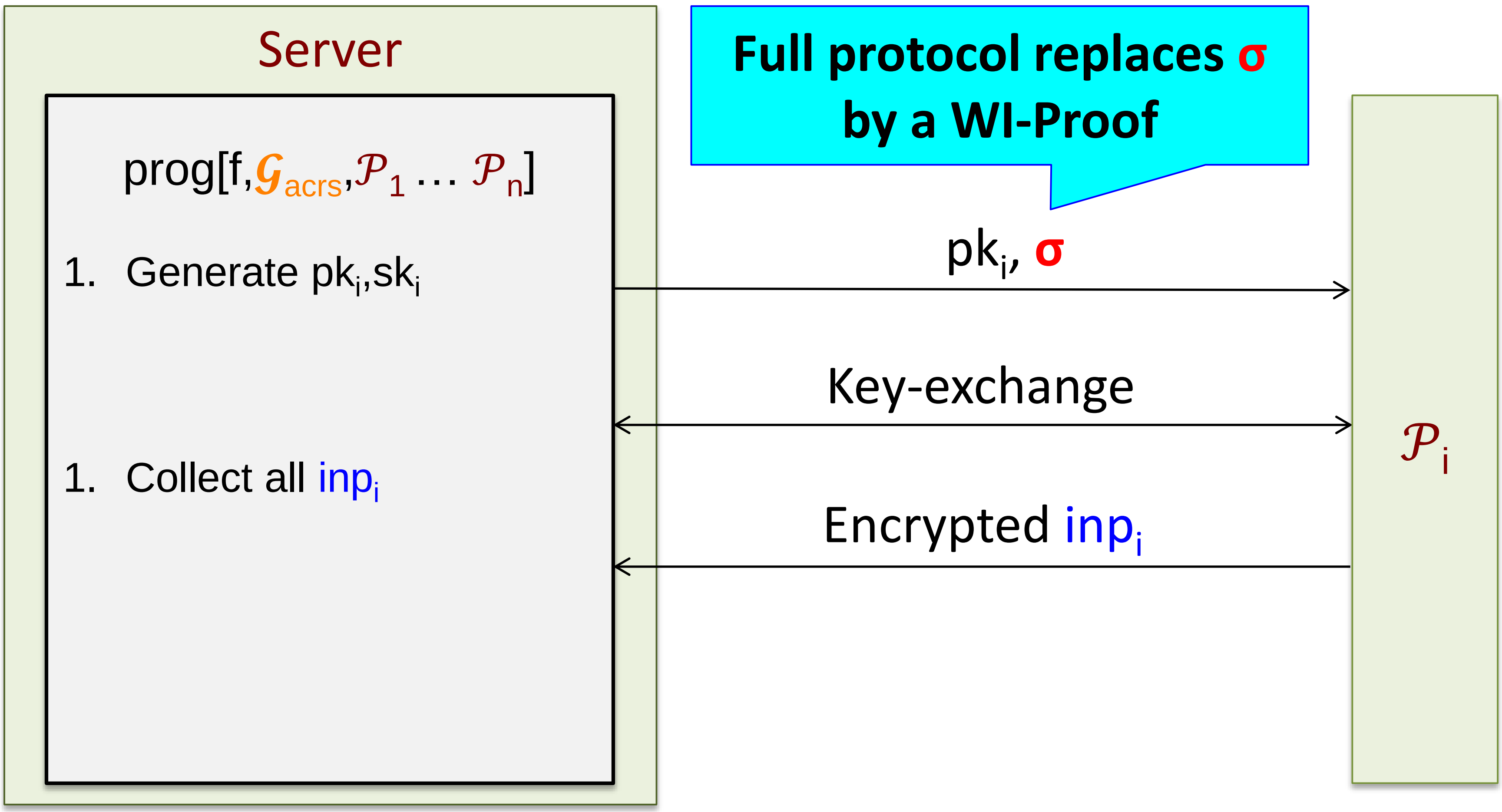
What if we **really really** want to use a single trusted processor?



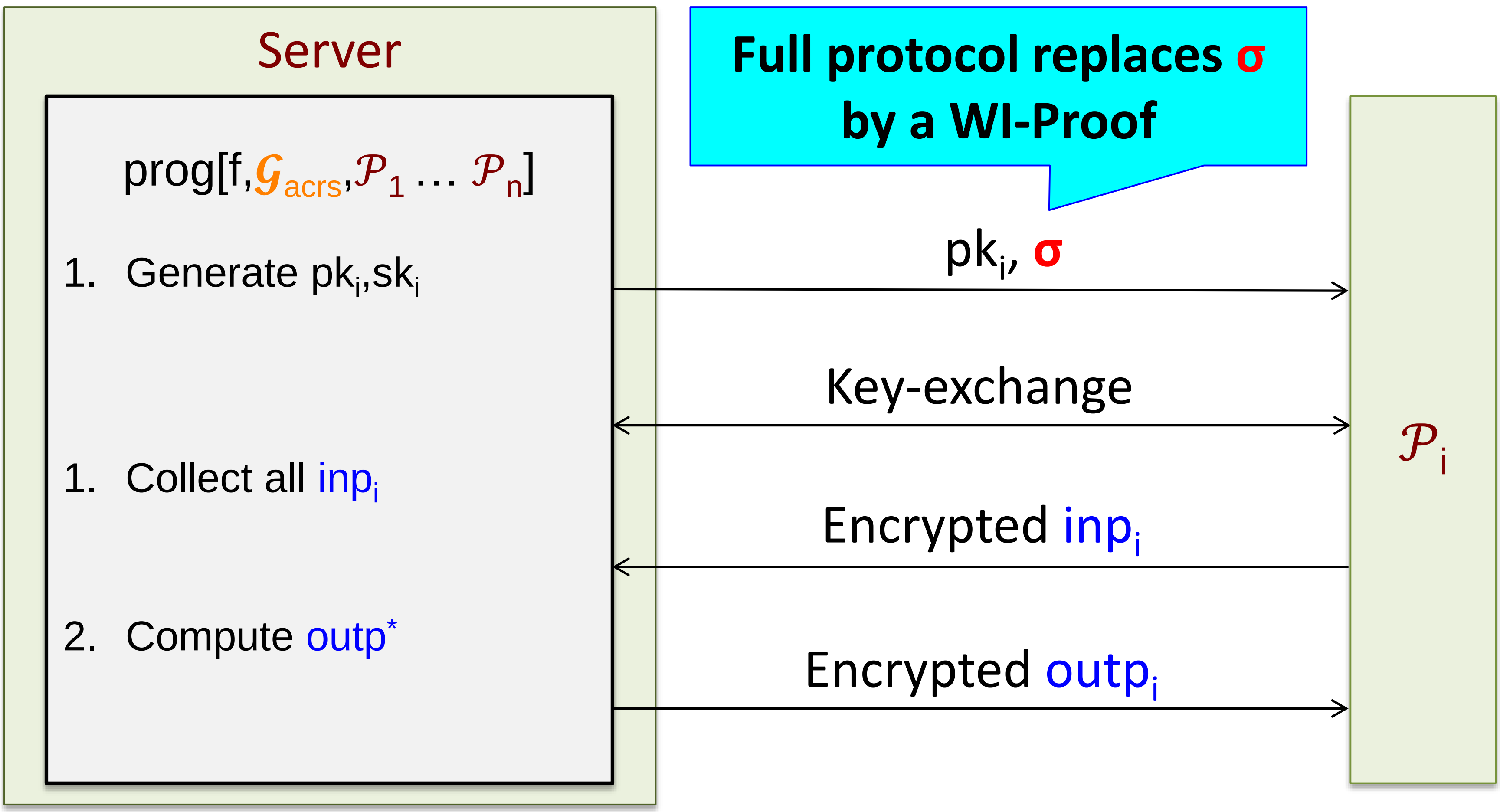
What if we **really really** want to use a single trusted processor?



What if we **really really** want to use a single trusted processor?



What if we **really really** want to use a single trusted processor?



What if we **really really** want to use a single trusted processor?

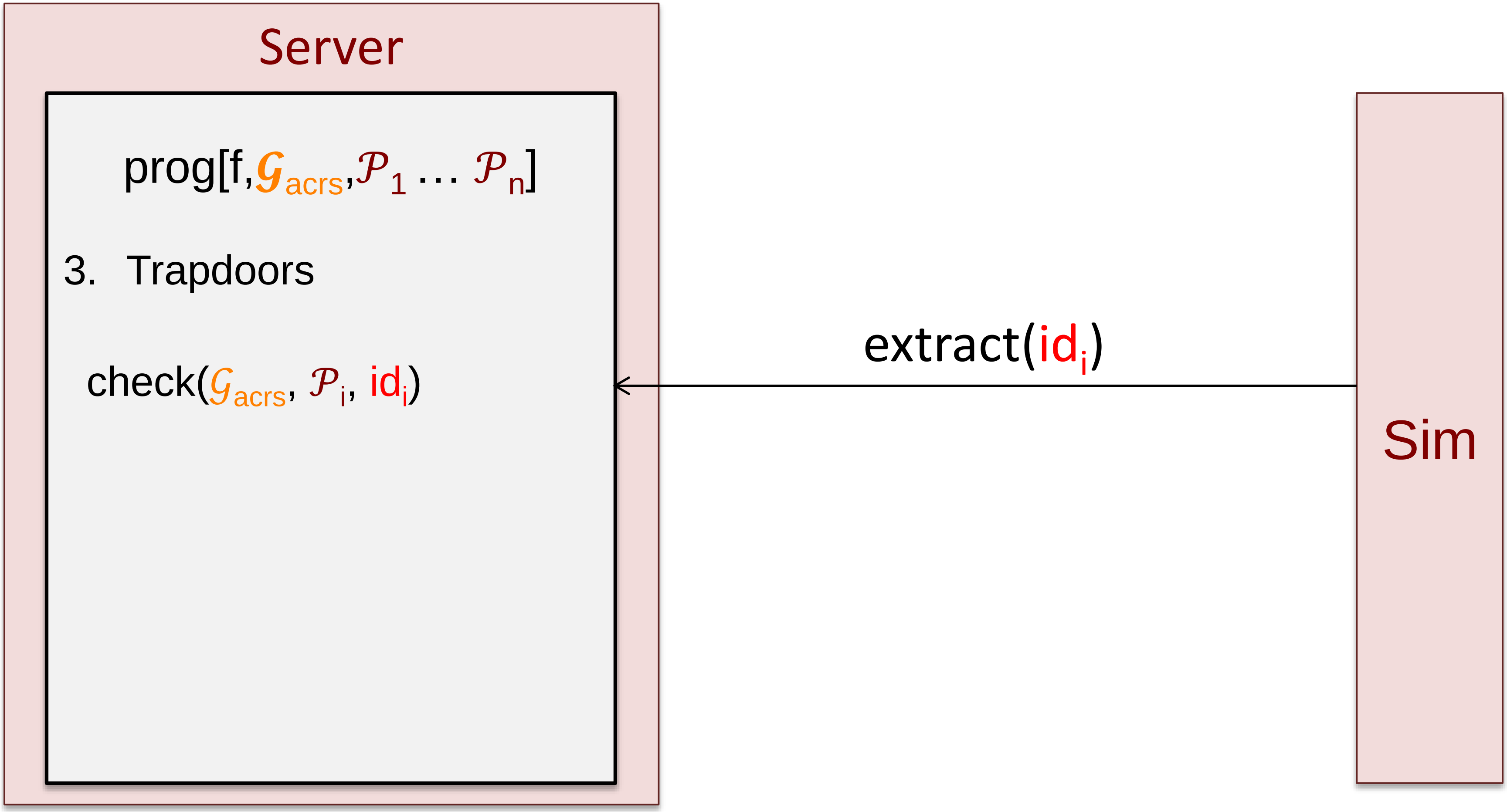
Server

prog[f, $\mathcal{G}_{acrs}$, $\mathcal{P}_1$ … $\mathcal{P}_n$]

3. Trapdoors

Sim

What if we **really really** want to use a single trusted processor?



What if we **really really** want to use a single trusted processor?

Server

prog[f, $\mathcal{G}_{acrs}$, $\mathcal{P}_1 \ldots \mathcal{P}_n$]

3. Trapdoors

check($\mathcal{G}_{acrs}$, $\mathcal{P}_i$, id_i)

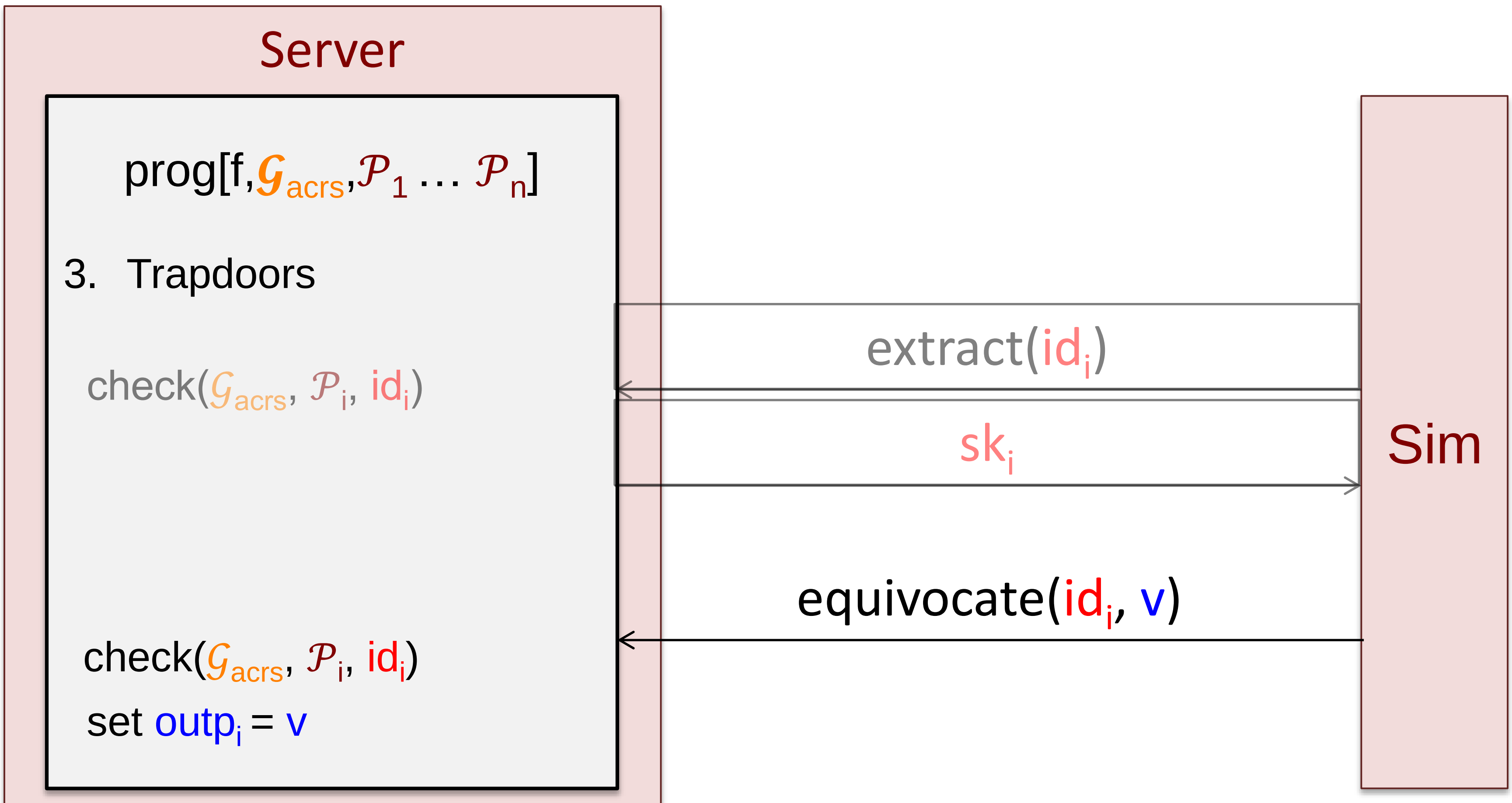
Sim can recover inp$_i$

extract(id_i)

sk_i

Sim

What if we **really really** want to use a single trusted processor?



Fair 2PC

- Fairness impossible for general functionalities in plain model [Cleve86]

Fair 2PC

Can trusted hardware help with fairness?

- Fairness impossible for general functionalities in plain model [Cleve86]

Fair 2PC

UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

- Fair 2PC possible if both parties have clock-aware secure processors

UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

- Fair 2PC possible if both parties have clock-aware secure processors

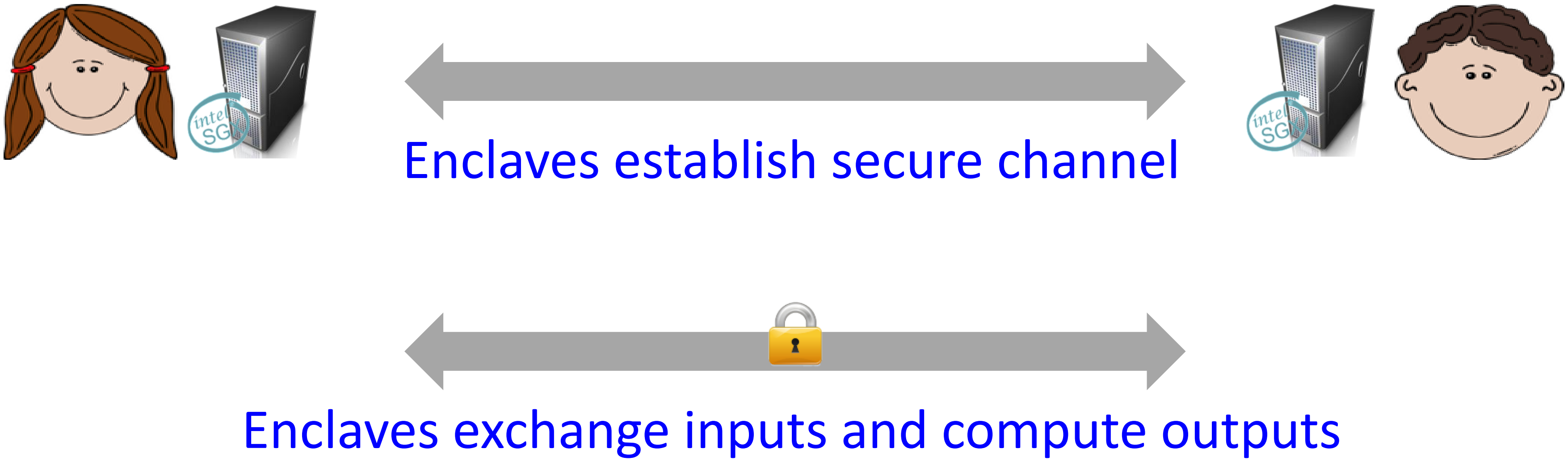
- Fair coin-tossing possible if one party has clock-aware secure processors (+ ACRS)

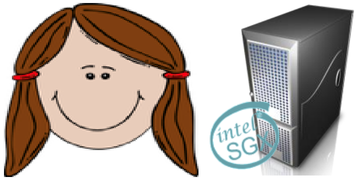
UC-Secure Fair 2PC

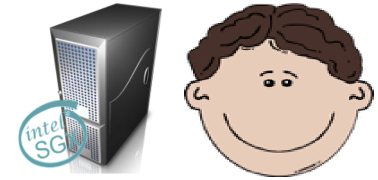
Enhanced model: Clock-aware secure processor

- Fair 2PC possible if both parties have clock-aware secure processors
- Fair coin-tossing possible if one party has clock-aware secure processors (+ ACRS)

Enclaves establish secure channel


Enclaves establish secure channel
Enclaves exchange inputs and compute outputs
intel SGX
intel SGX



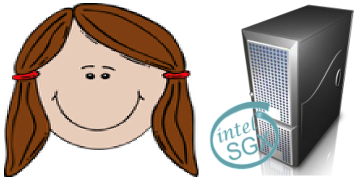


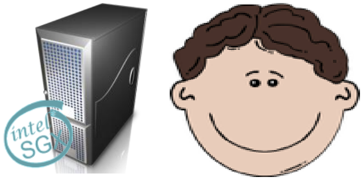
Enclaves establish secure channel

Enclaves exchange inputs and compute outputs

“Will release to Alice in 2^{λ} time”

“Will release to Bob in 2^{λ} time”





Enclaves establish secure channel

Enclaves exchange inputs and compute outputs

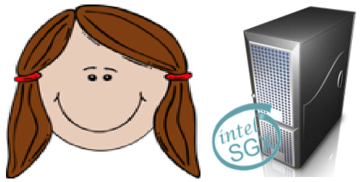
"Will release to Alice in 2^{λ} time"

"Will release to Bob in 2^{λ} time"

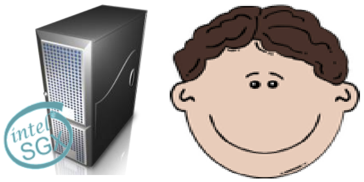
"Will release to Alice in $2^{\lambda-1}$ time"

"Will release to Bob in $2^{\lambda-1}$ time"

…



Enclaves establish secure channel



If Alice learns result at time $t < 2^{\lambda}$,

Bob will learn it at the latest by time $2t$

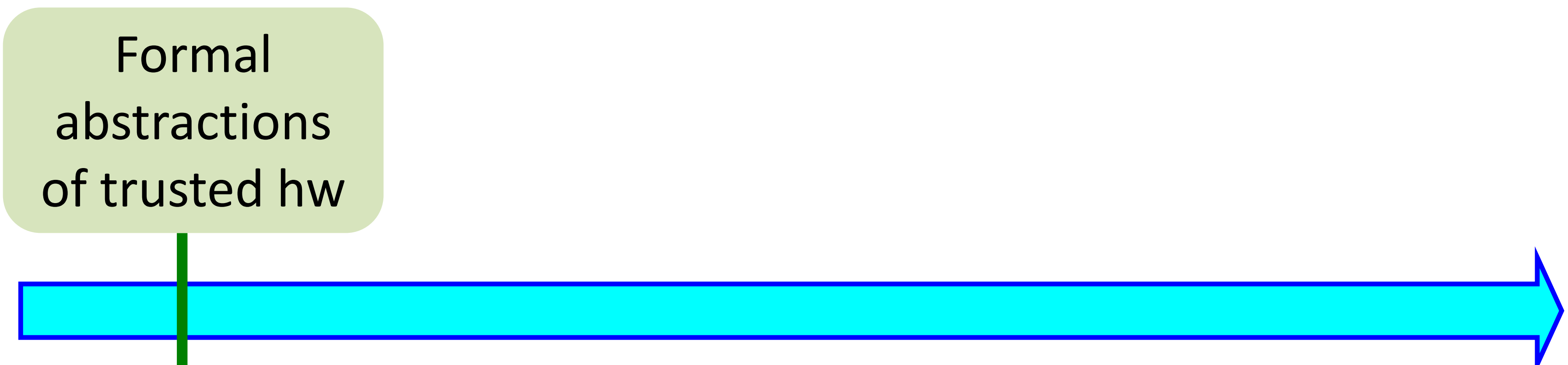
+ no "wasted" computation!

"Will release to Alice in $2^{\lambda-1}$ time"

"Will release to Bob in $2^{\lambda-1}$ time"

…

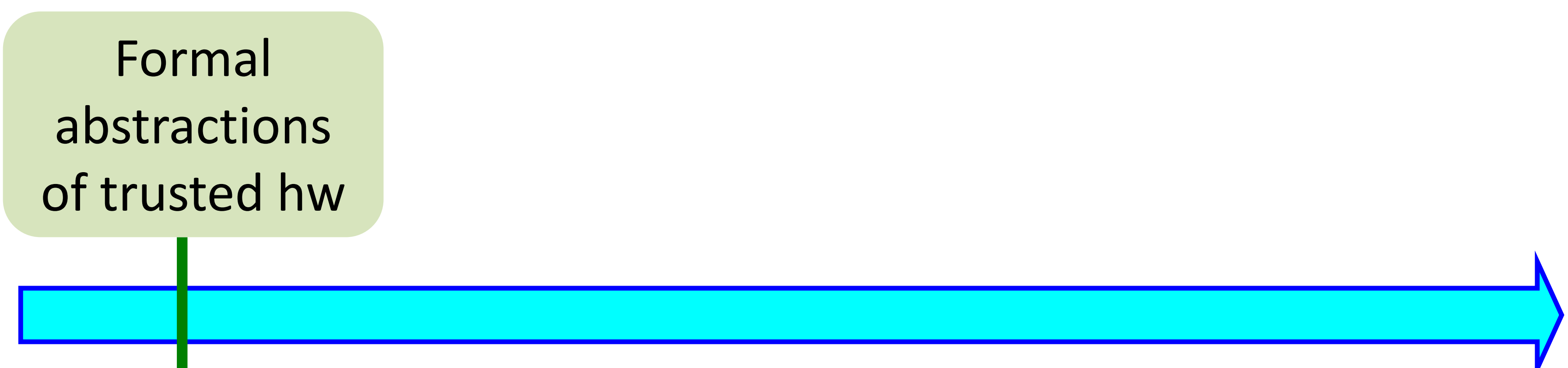
What next?



What next?

Attested execution is a powerful assumption

$\Longrightarrow$ Stateful Obfuscation, Efficient MPC, Fair 2PC



What next?

Attested execution is a powerful assumption

$\Longrightarrow$ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless *all parties have trusted hardware*

$\Longrightarrow$ Non-deniability, Extra setup assumptions

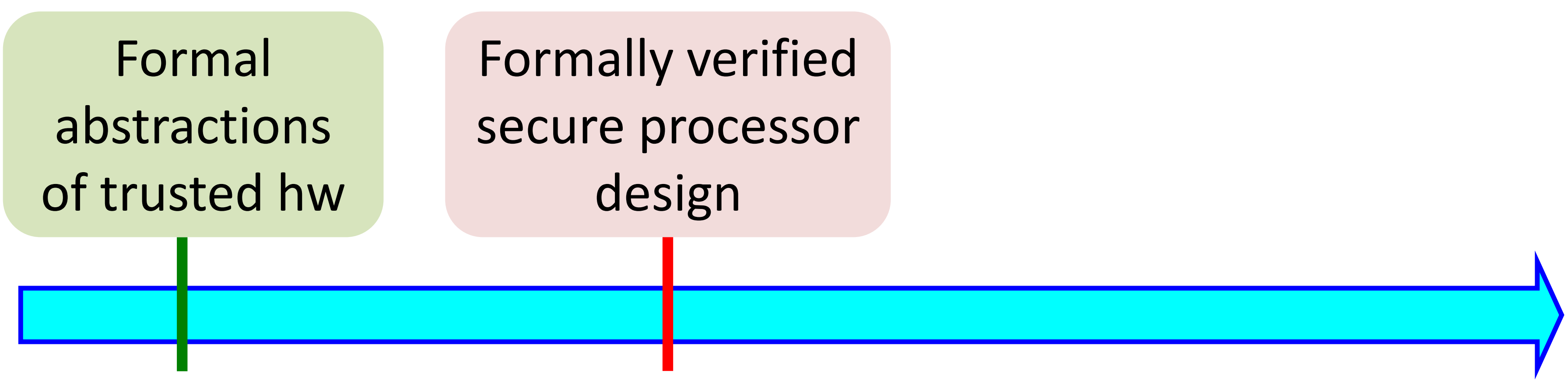
What next?

Attested execution is a powerful assumption

⟹ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless *all parties have trusted hardware*

⟹ Non-deniability, Extra setup assumptions



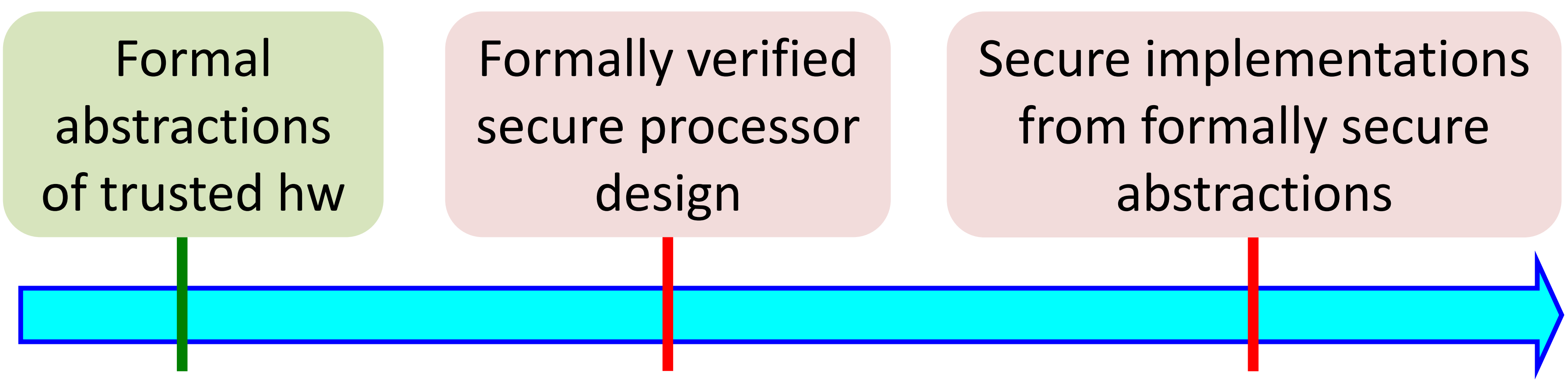
What next?

Attested execution is a powerful assumption

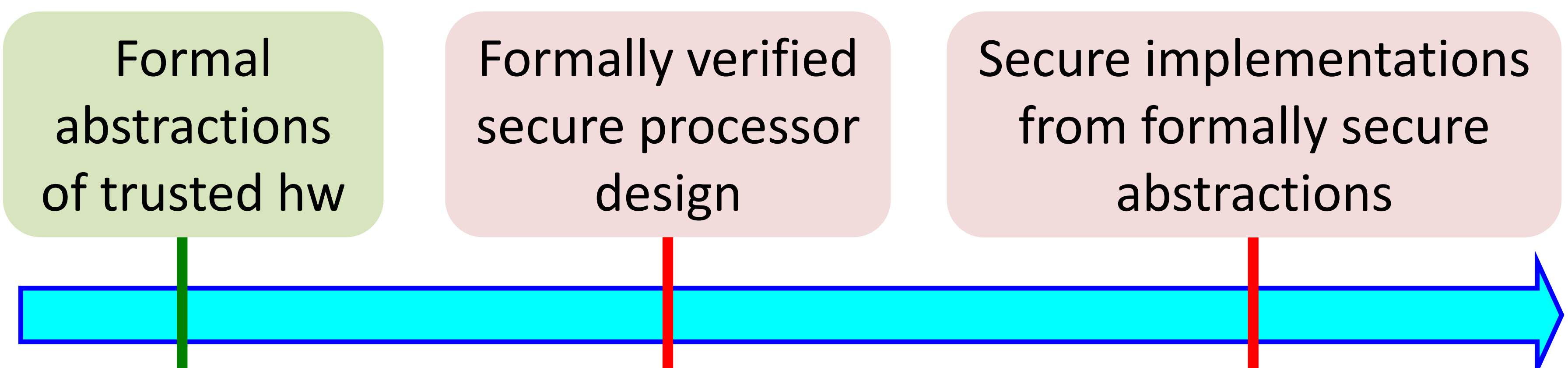
⟹ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless *all parties have trusted hardware*

⟹ Non-deniability, Extra setup assumptions



Thank You


Formal abstractions of trusted hw
Formally verified secure processor design
Secure implementations from formally secure abstractions