Sealed-Glass Proofs

IEEE EuroS&P, 2017
April 26, 2017

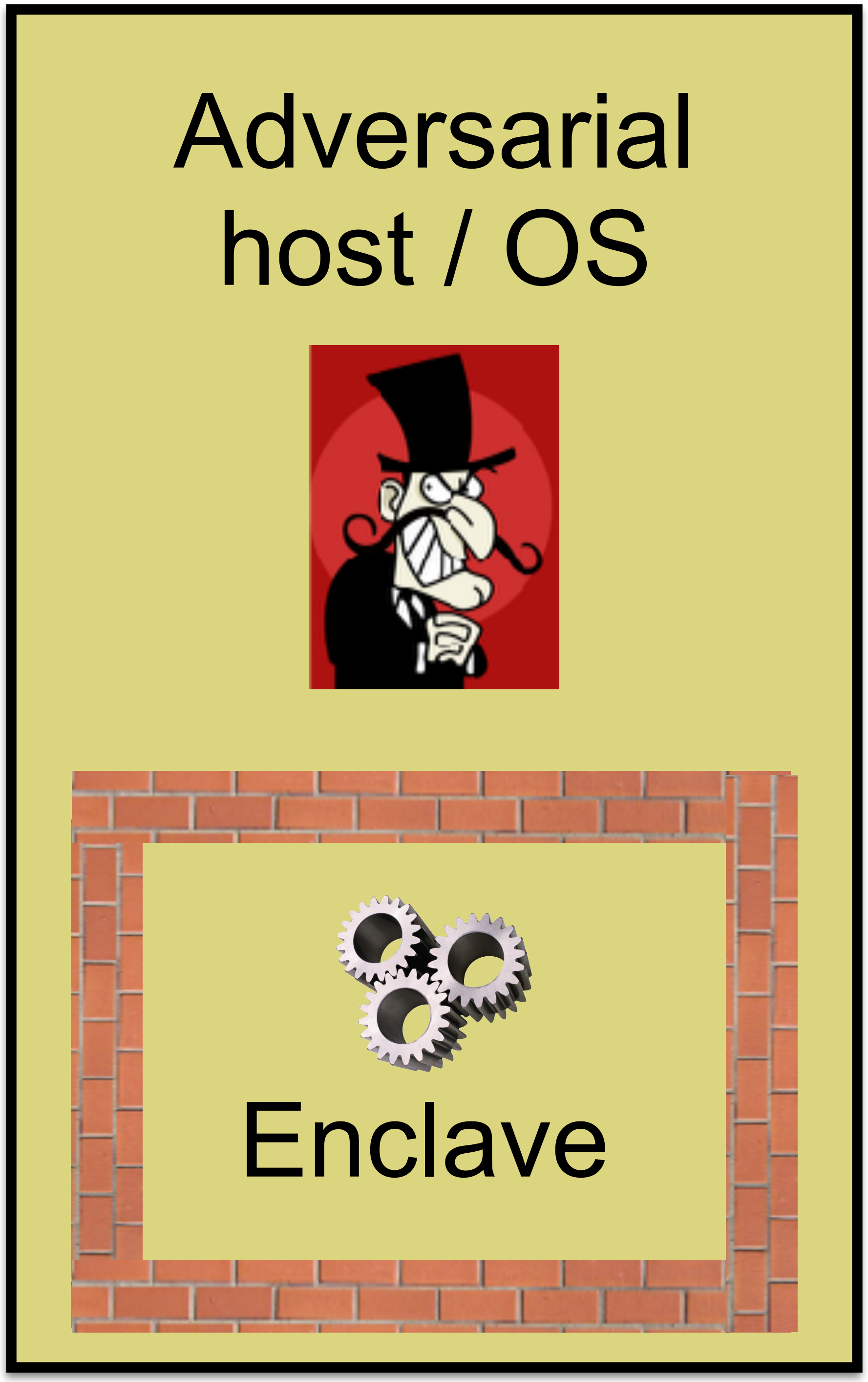
Florian Tramèr, Fan Zhang, Huang Lin,
Jean-Pierre Hubaux, Ari Juels, and Elaine Shi.

Attested Execution

properties, threat model, use-cases

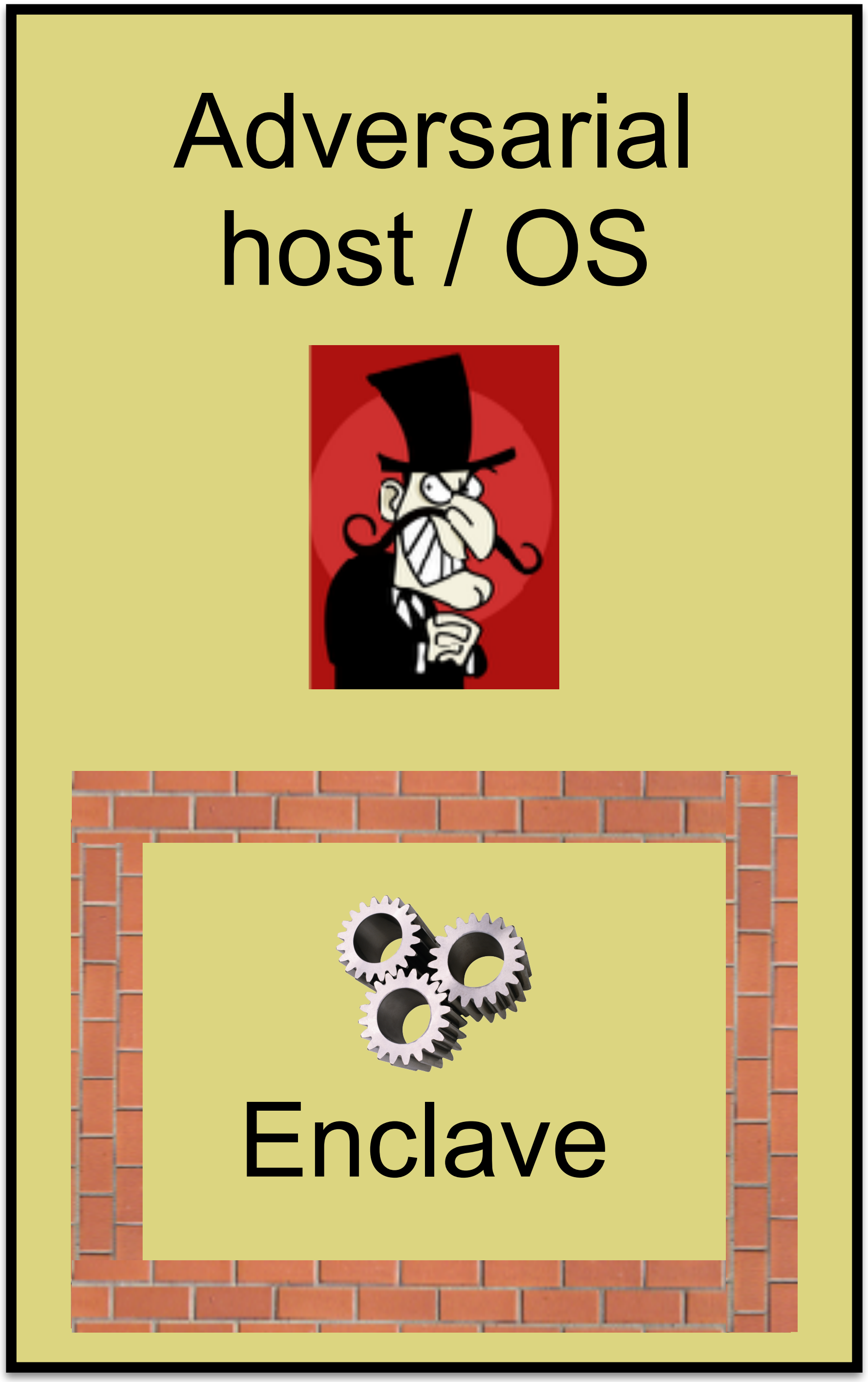
Attested Execution
properties, threat model, use-cases

- Isolated execution environment on untrustworthy host



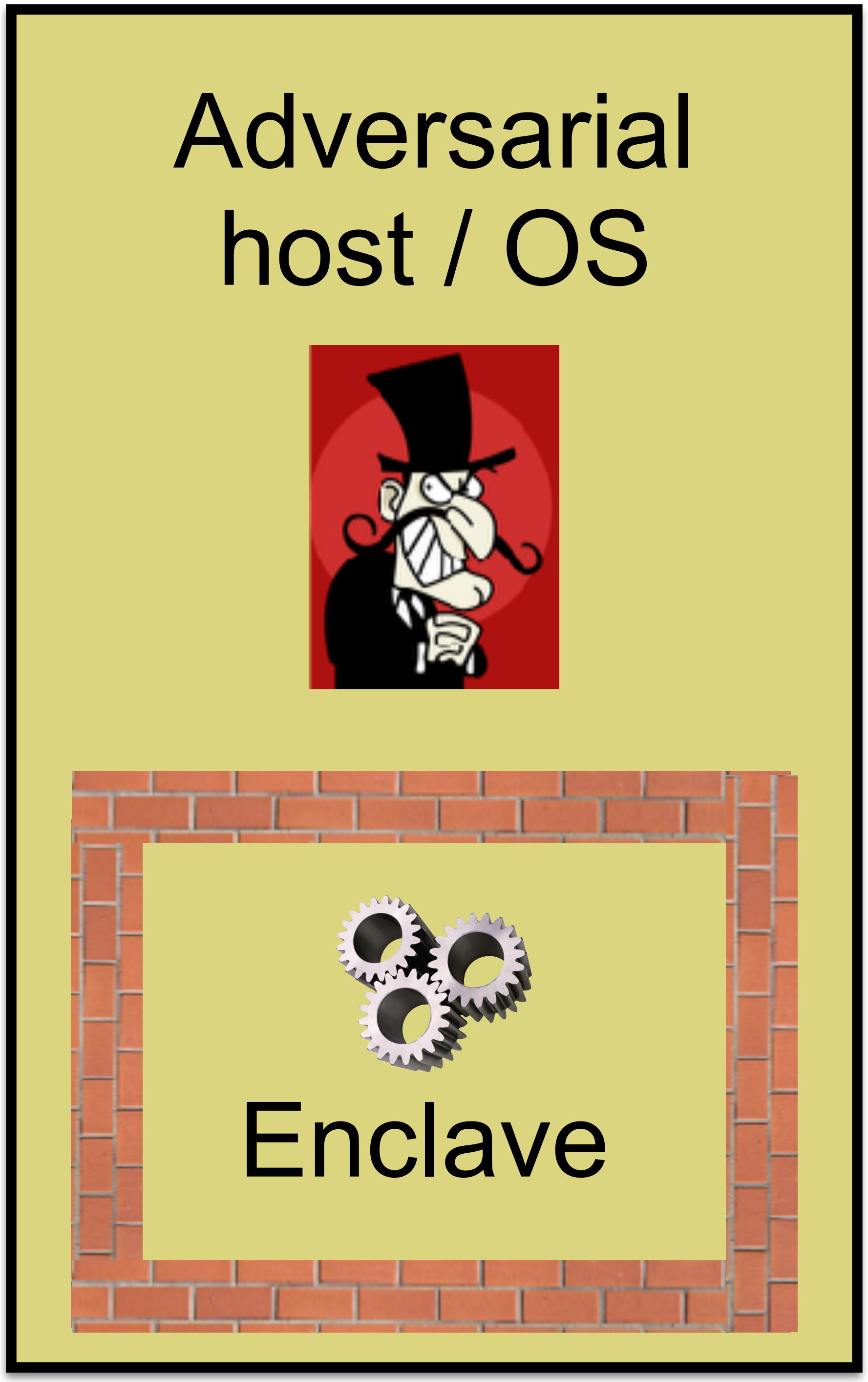
Attested Execution
properties, threat model, use-cases

- Isolated execution environment on untrustworthy host
  - Confidentiality



Attested Execution
properties, threat model, use-cases

- Isolated execution environment on untrustworthy host
  - Confidentiality
  - Integrity

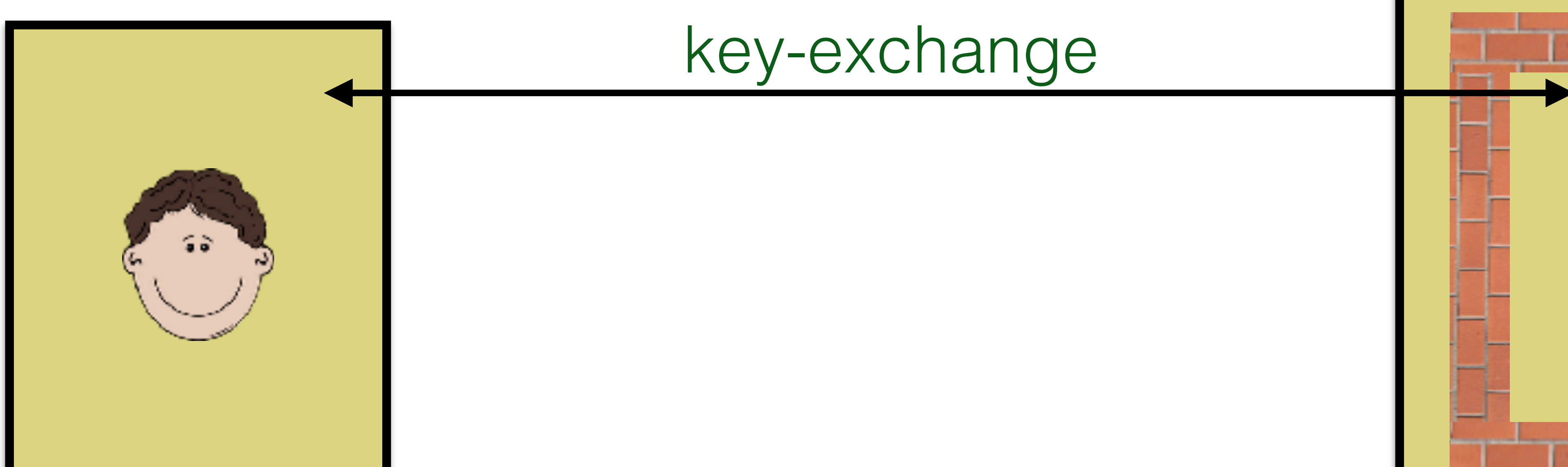


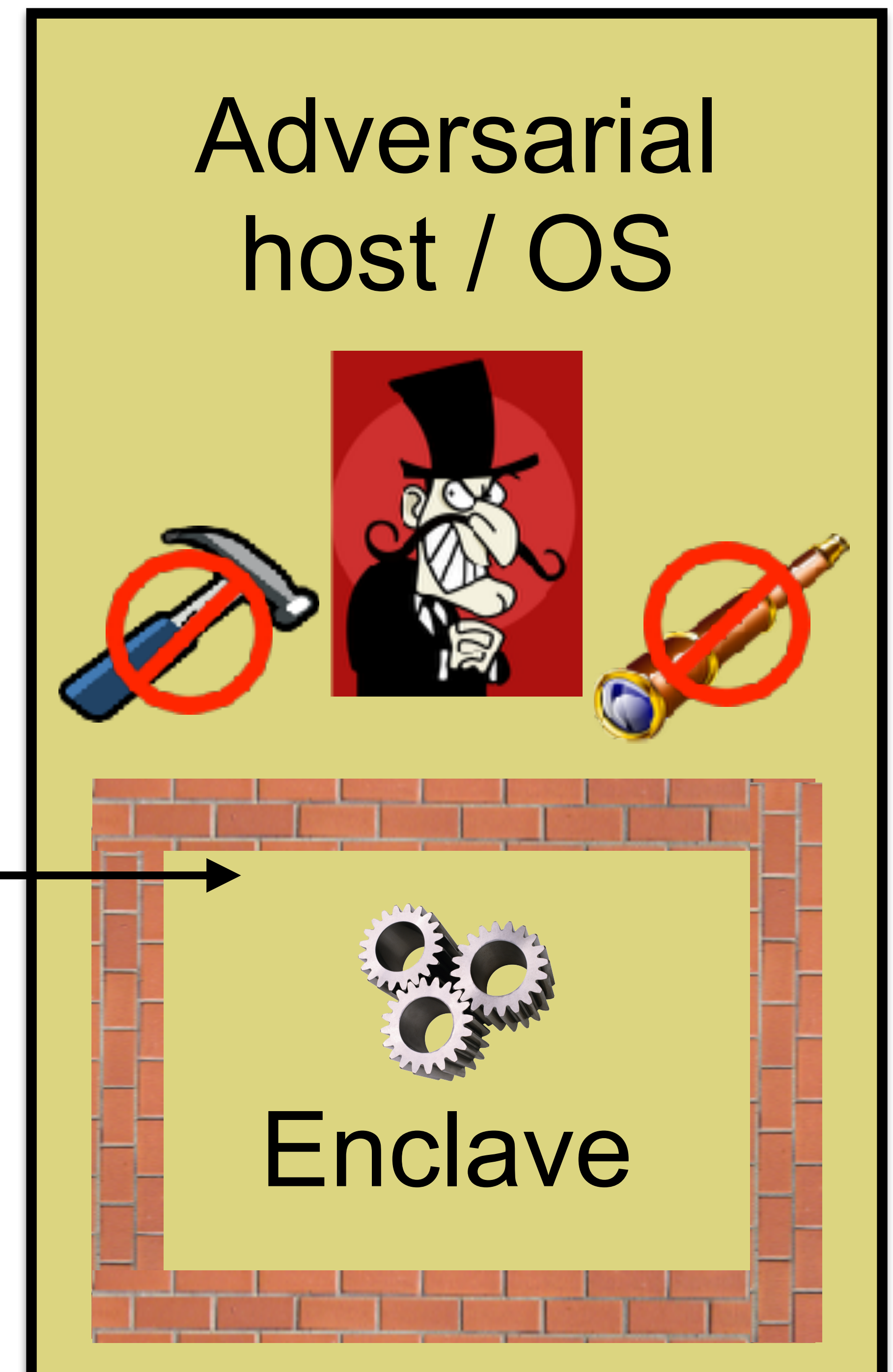
Attested Execution
properties, threat model, use-cases

- Isolated execution environment on untrustworthy host
    - Confidentiality
    - Integrity
    - Authenticity

Attested Execution
properties, threat model, use-cases

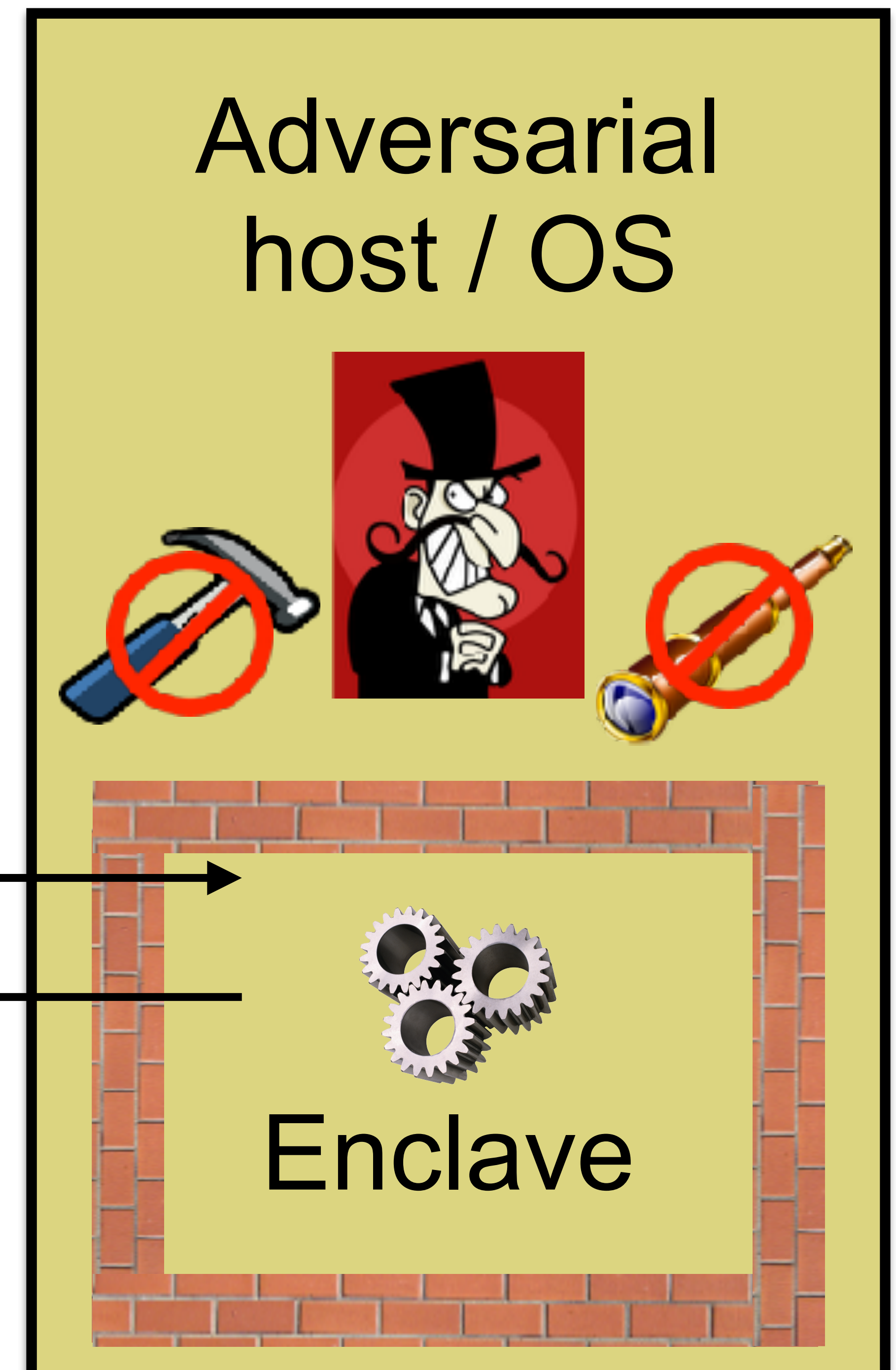
- Isolated execution environment on untrustworthy host
  - Confidentiality
  - Integrity
  - Authenticity

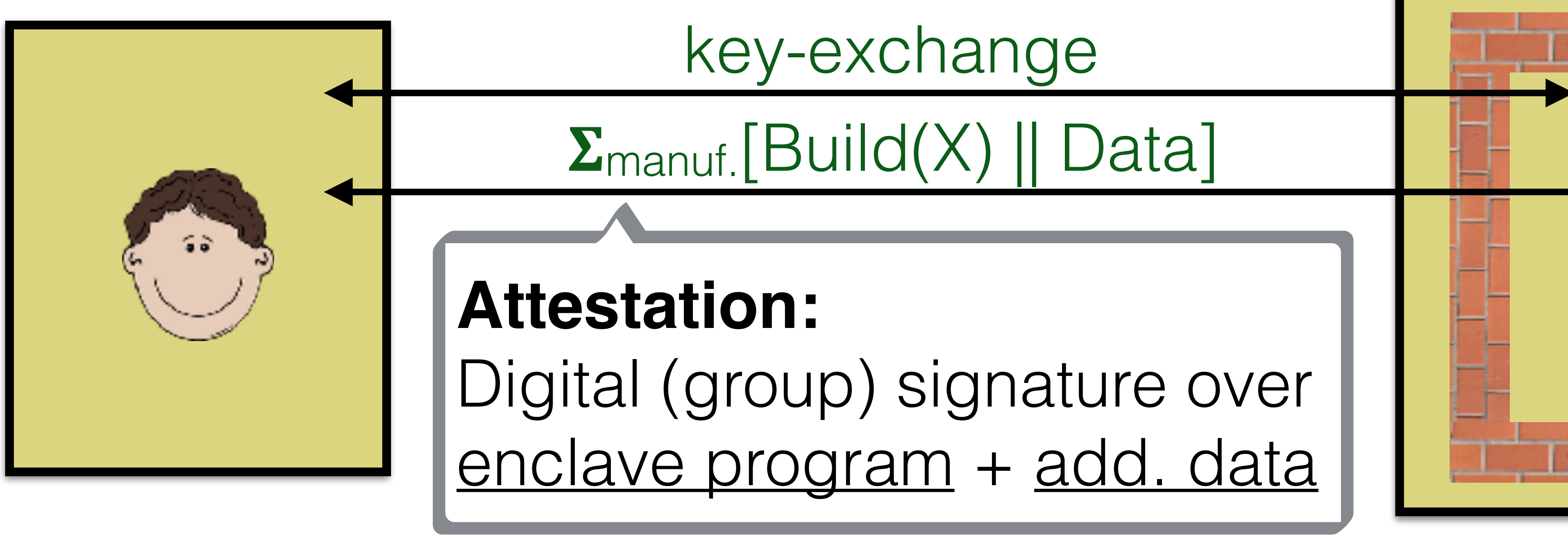




Attested Execution
properties, threat model, use-cases

- Isolated execution environment on untrustworthy host
  - Confidentiality
  - Integrity
  - Authenticity

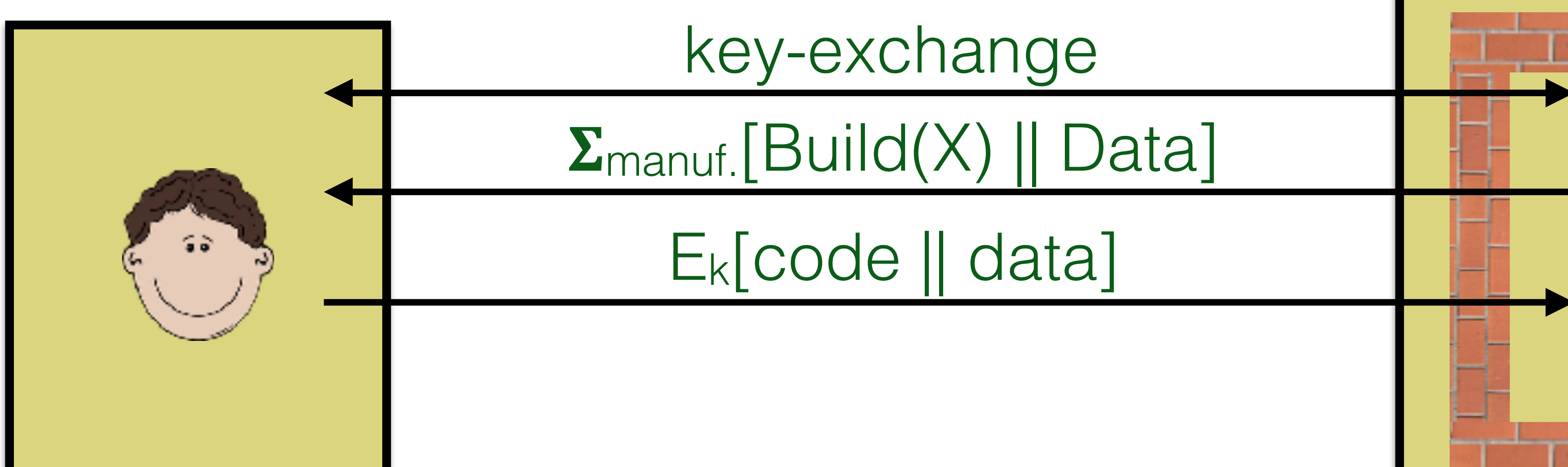


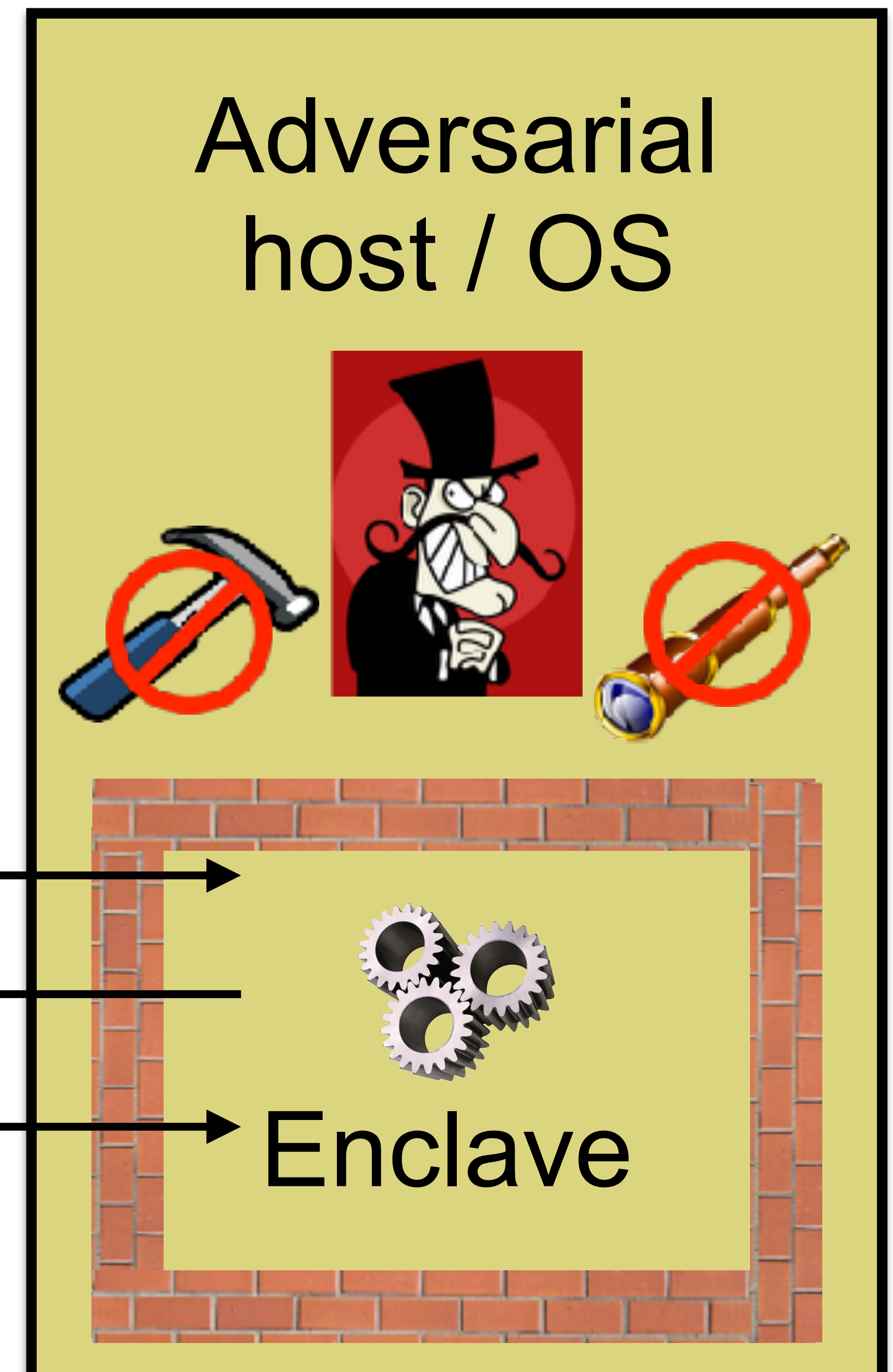


Attested Execution

properties, threat model, use-cases

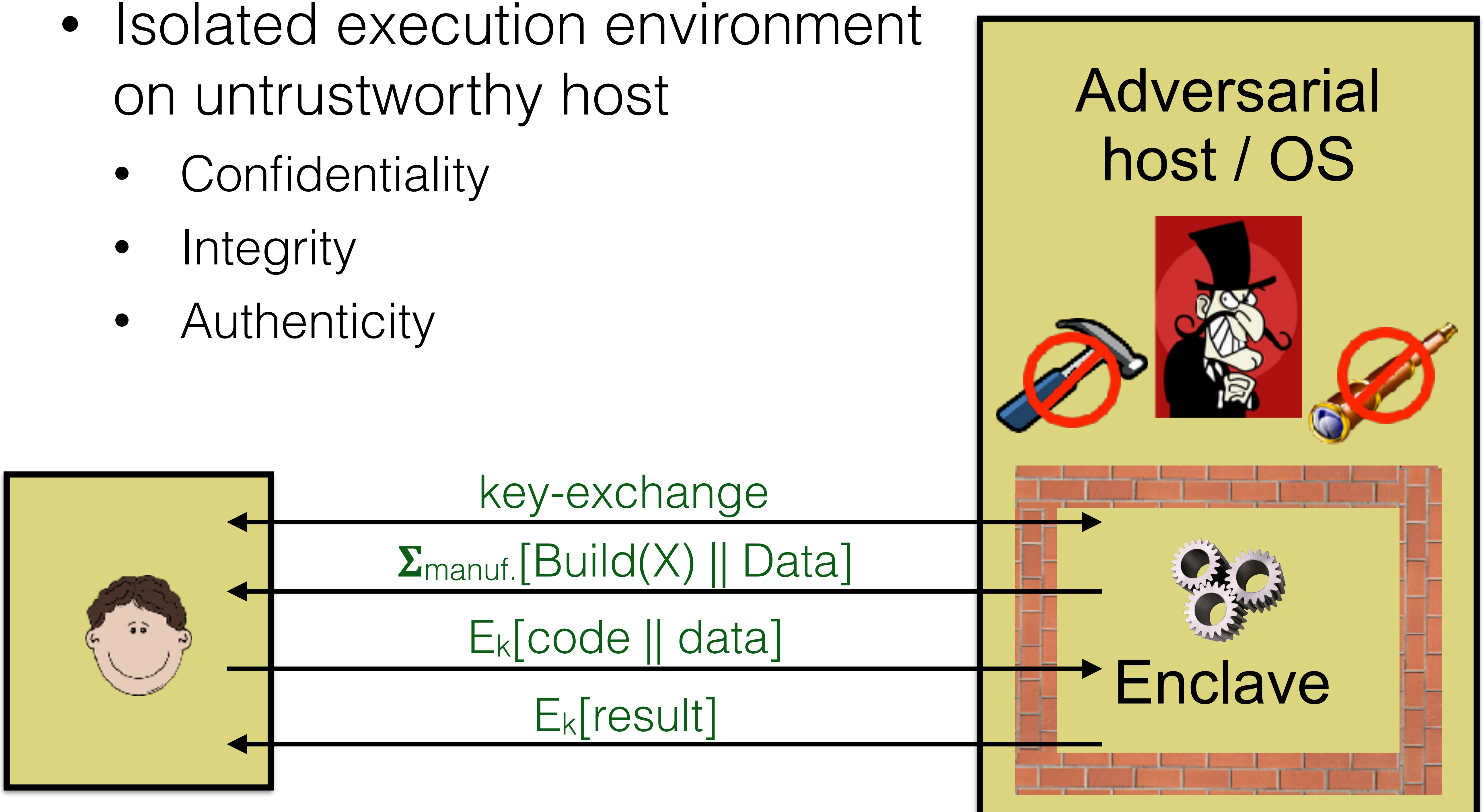
- Isolated execution environment on untrustworthy host
  - Confidentiality
  - Integrity
  - Authenticity





Attested Execution
properties, threat model, use-cases

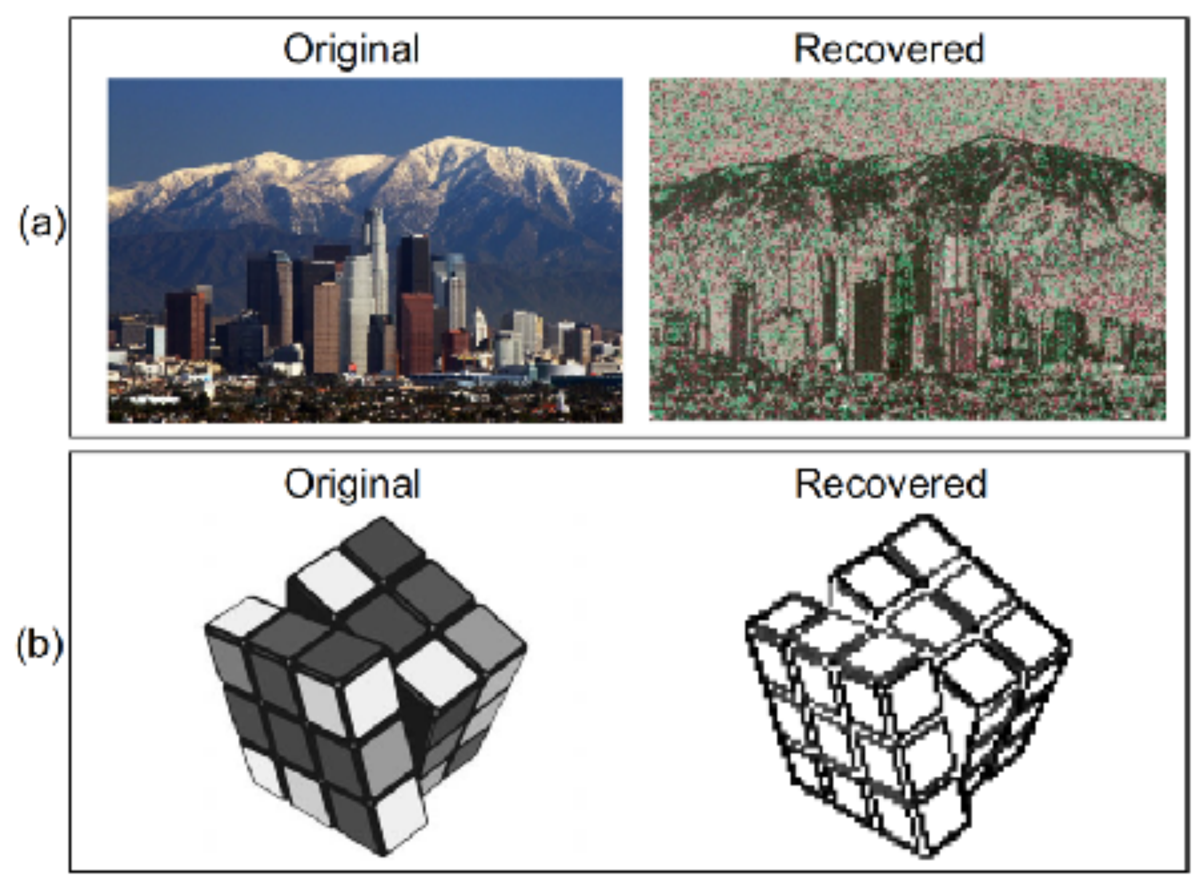
- Isolated execution environment on untrustworthy host
  - Confidentiality
  - Integrity
  - Authenticity



Isolation is imperfect

Isolation is imperfect

- E.g., SGX page faults can be induced and seen by OS
  - Leaks memory access patterns
- Many recent papers about cache side channels



libjpeg attack from Y. Xu, W. Cui, and M. Peinado, *"Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems"*, IEEE S&P, 2015

Solutions?

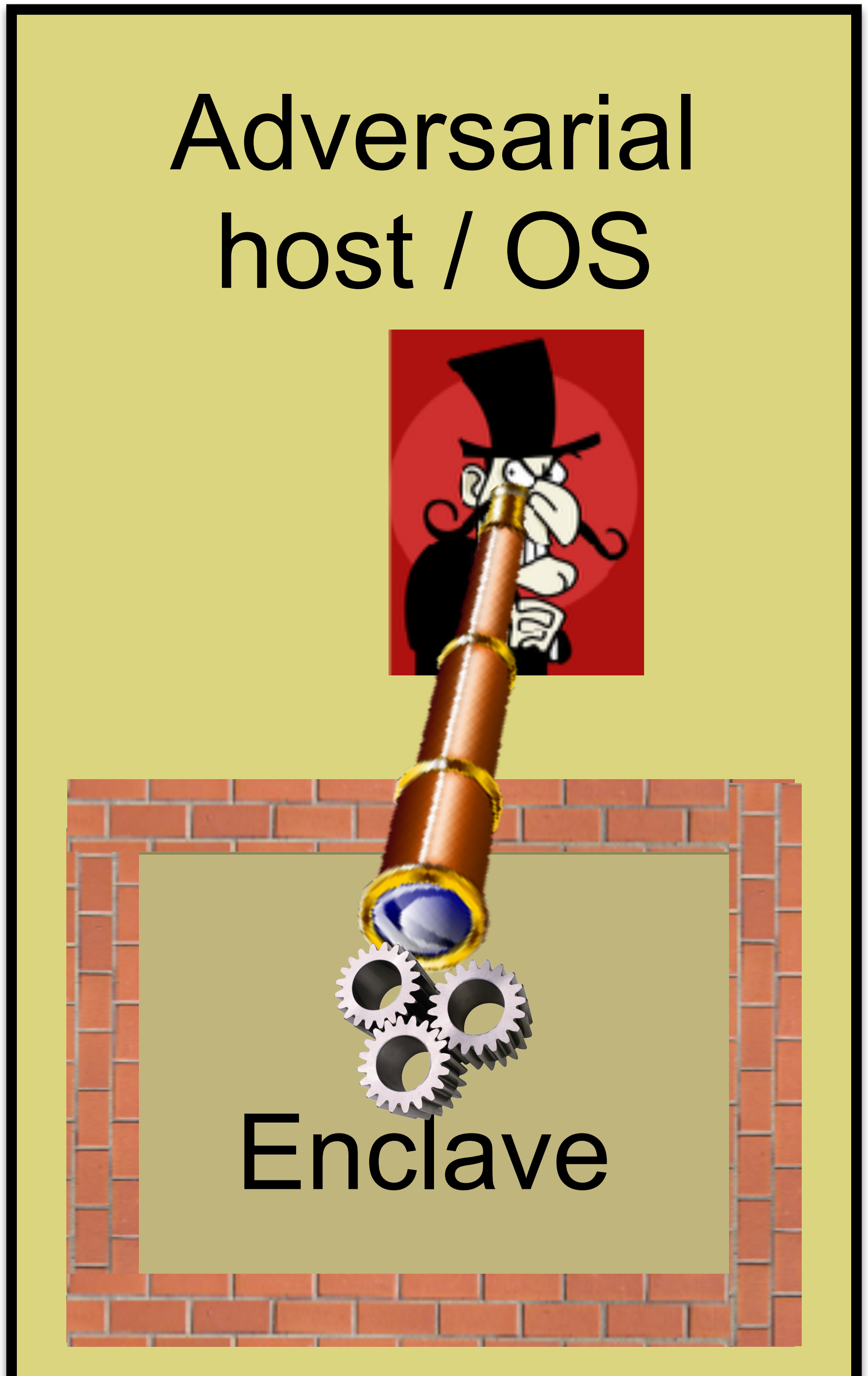
Solutions?

- Side channels “out-of-scope”
- Oblivious Data Structures
- ORAM
- What if leakage doesn’t matter?

New adversarial model

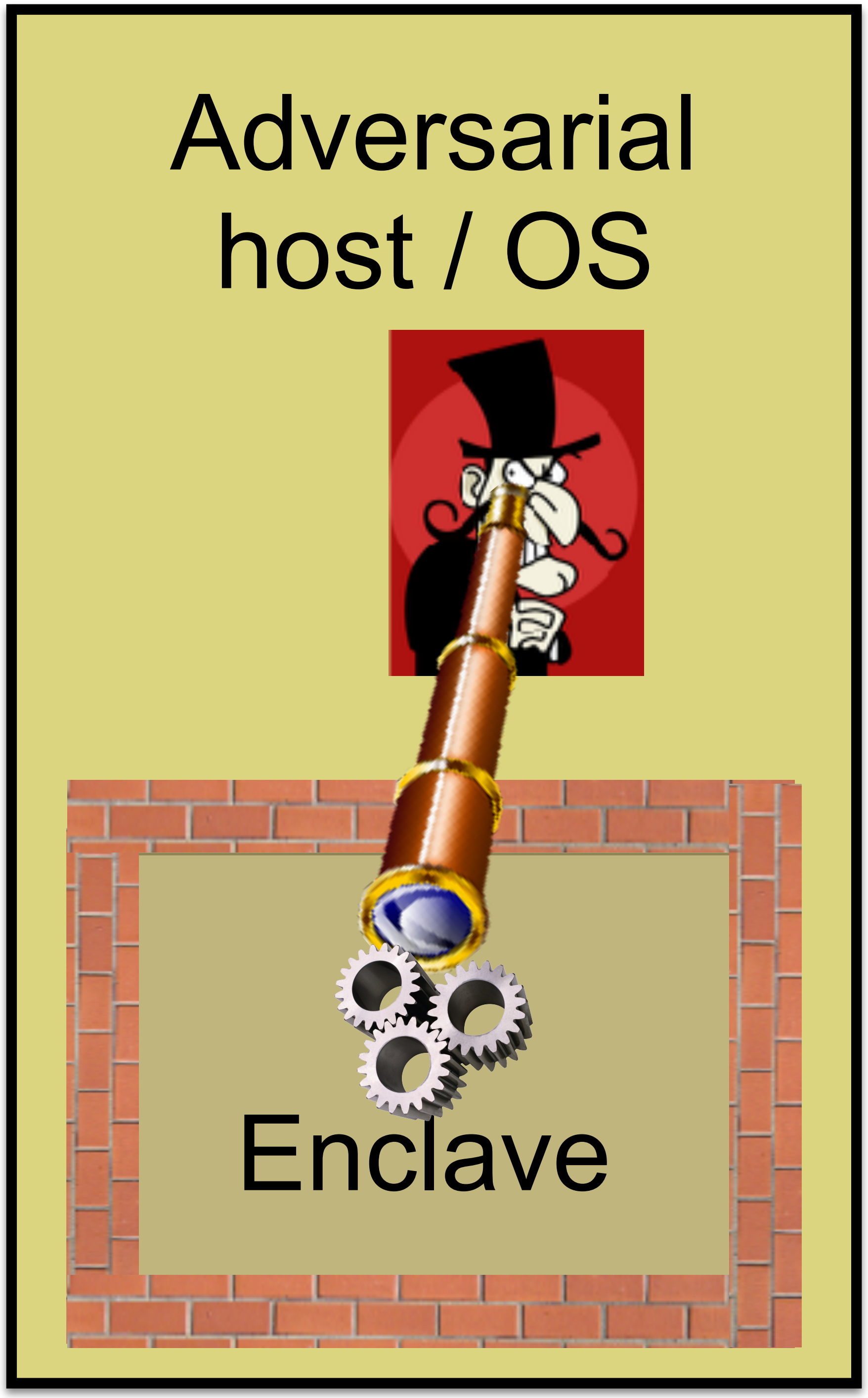
New adversarial model

- Model user program execution in SGX as ***Transparent***



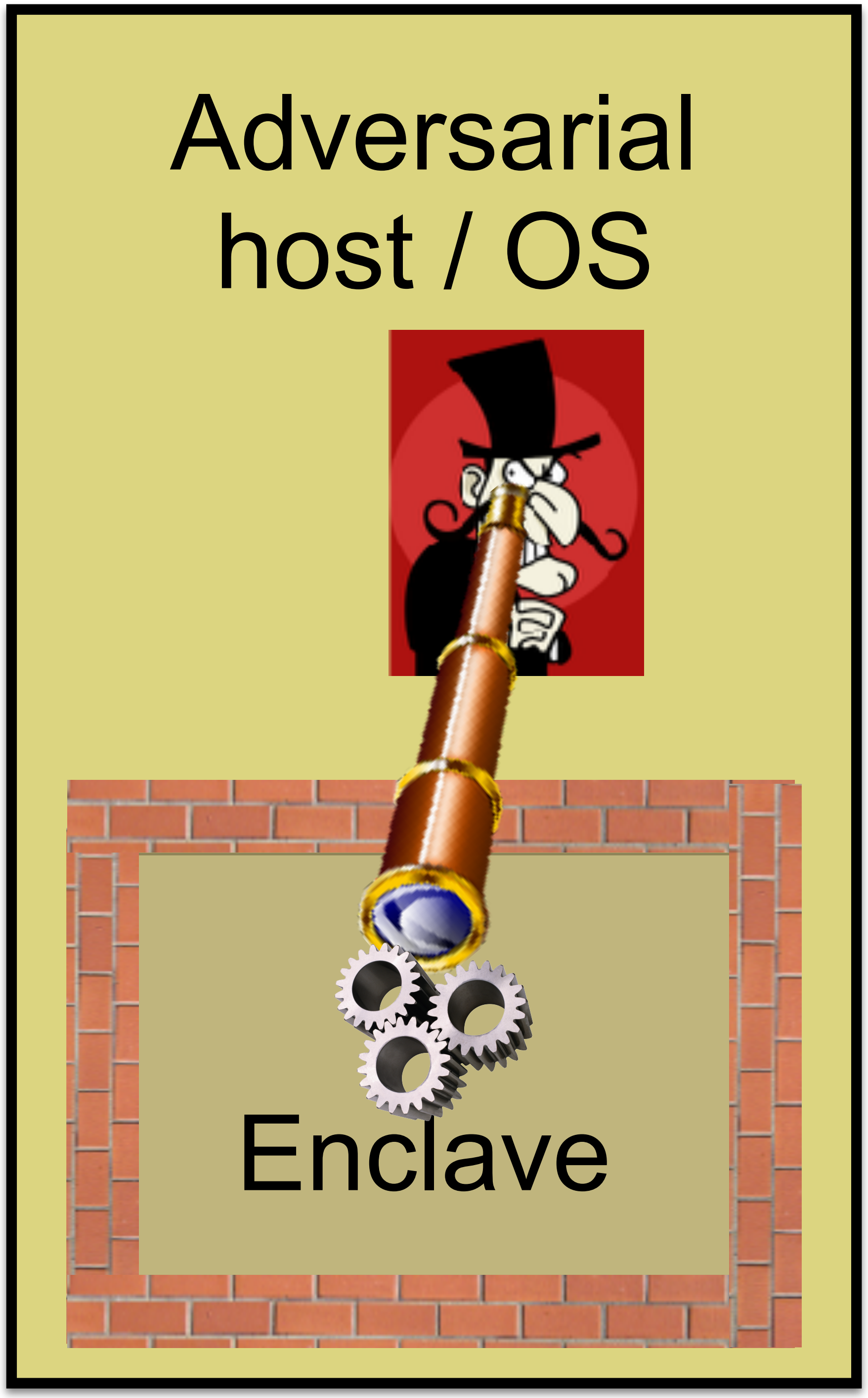
New adversarial model

- Model user program execution in SGX as ***Transparent***
- We assume *unbounded leakage* of program execution to host



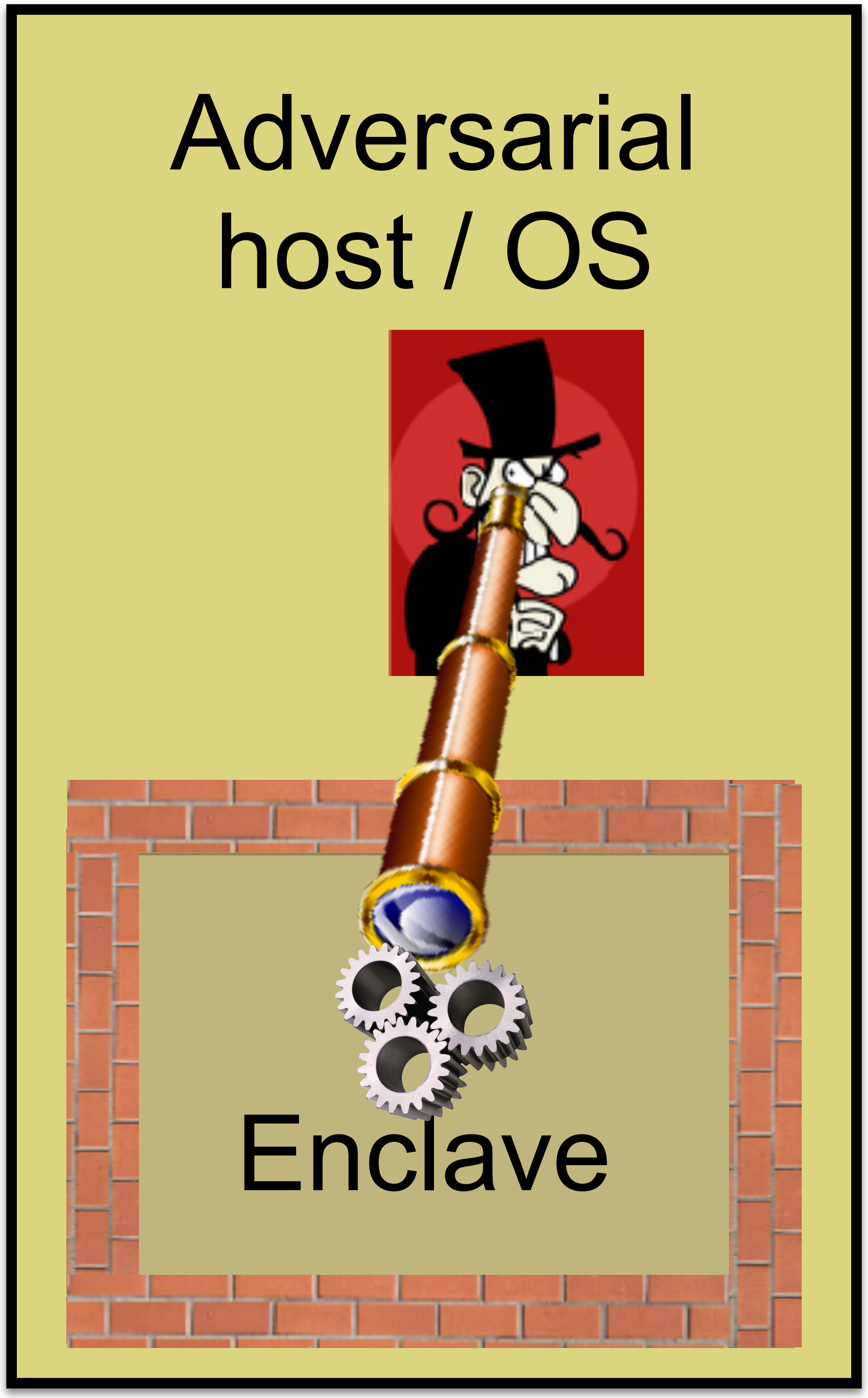
New adversarial model

- Model user program execution in SGX as ***Transparent***
- We assume *unbounded leakage* of program execution to host
- But *correct* execution and attestation



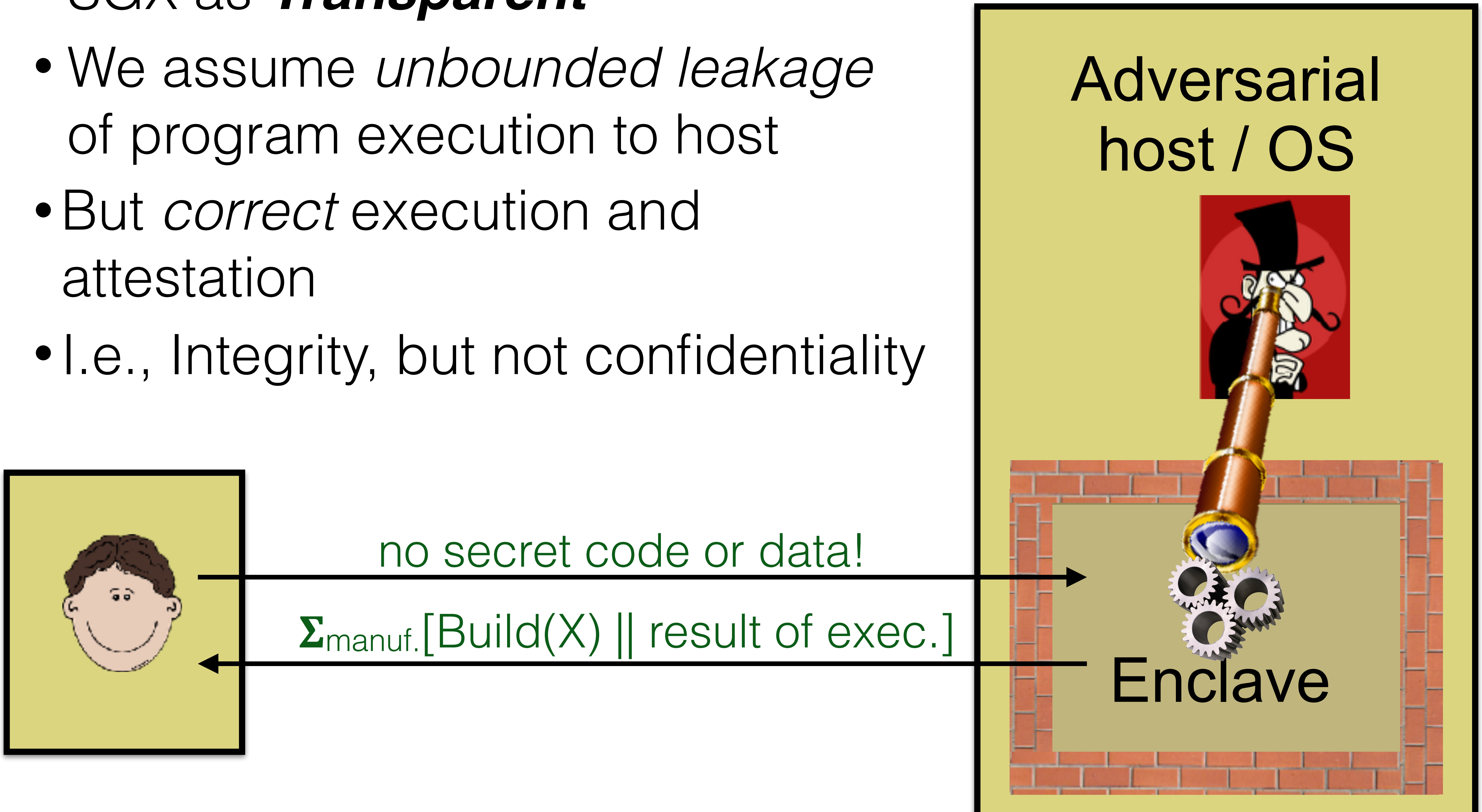
New adversarial model

- Model user program execution in SGX as ***Transparent***
- We assume *unbounded leakage* of program execution to host
- But *correct* execution and attestation
- I.e., Integrity, but not confidentiality



New adversarial model

- Model user program execution in SGX as ***Transparent***
- We assume *unbounded leakage* of program execution to host
- But *correct* execution and attestation
- I.e., Integrity, but not confidentiality



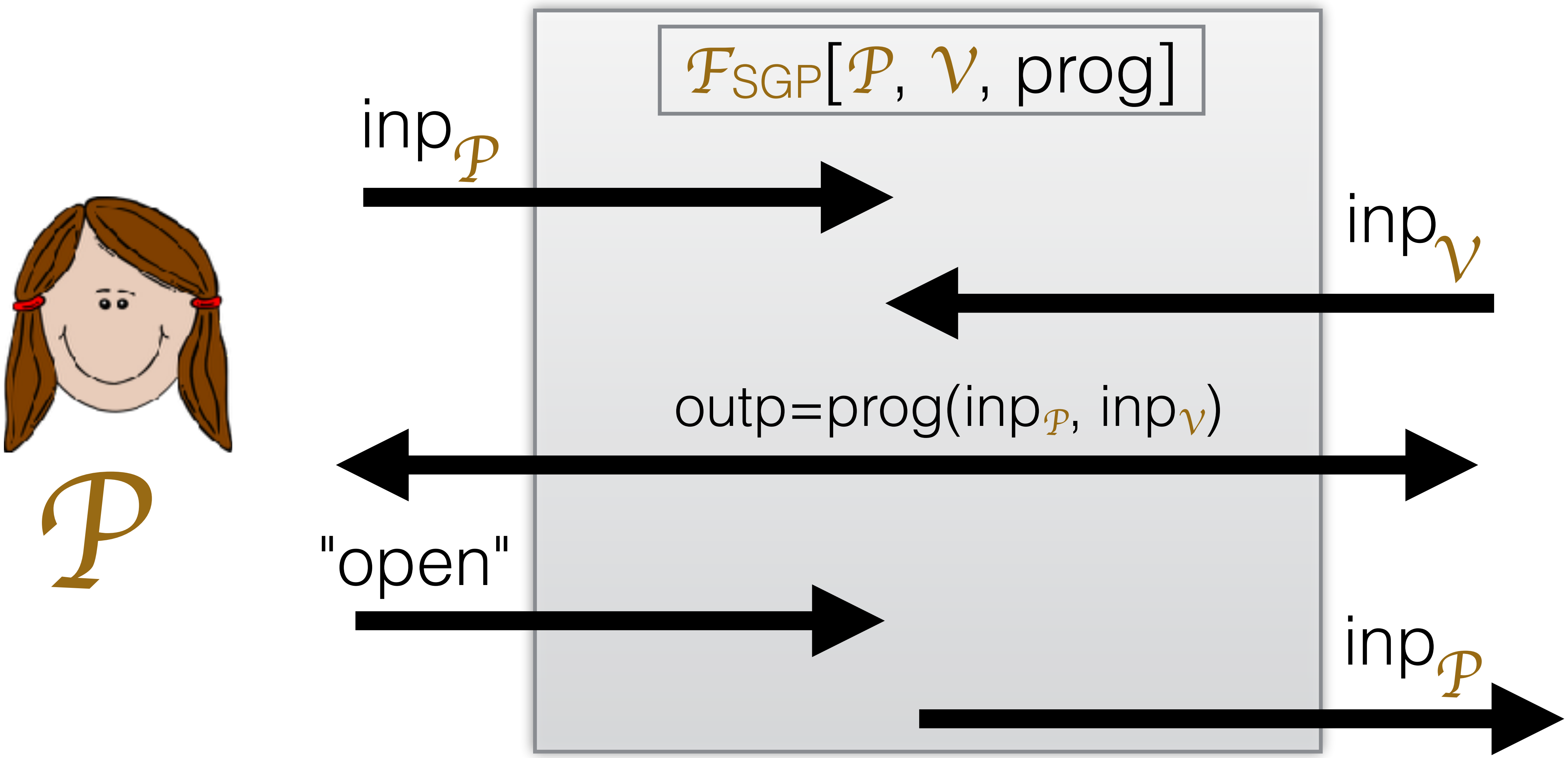
Sealed-Glass Proof (SGP)

Sealed-Glass Proof (SGP)

- **Key observation:** In many interactive proofs, prover holds secrets, so *information leakage on prover device doesn't hurt*

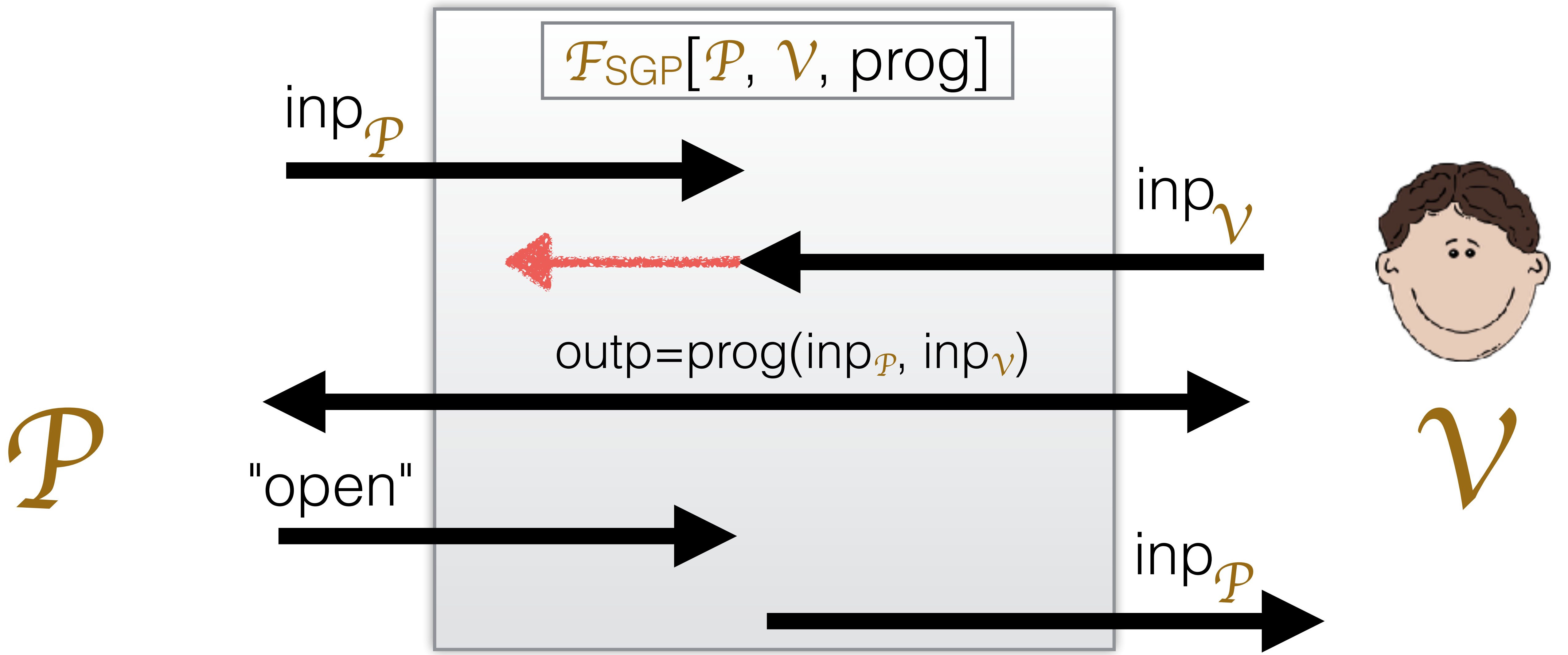
Sealed-Glass Proof (SGP)

- **Key observation:** In many interactive proofs, prover holds secrets, so *information leakage on prover device doesn't hurt*



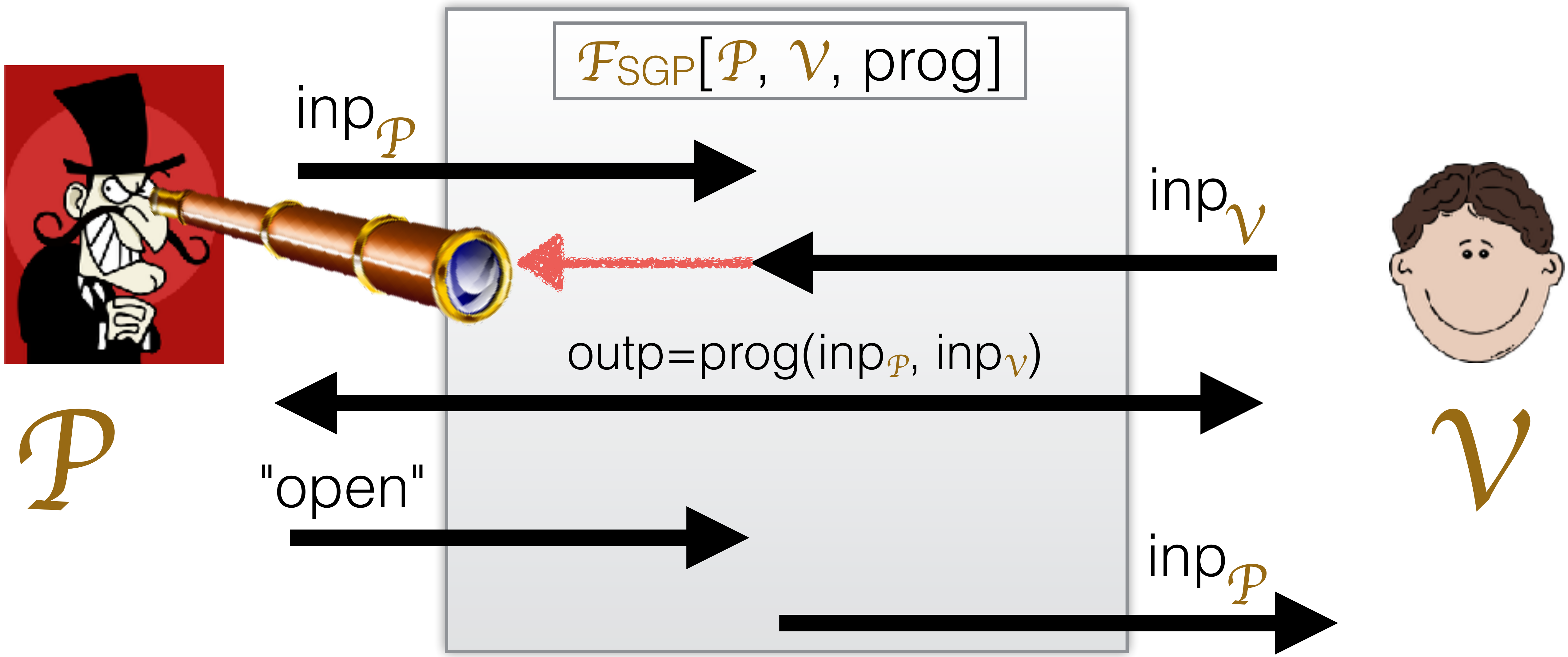
Why "sealed glass" ?

- **Model:** $\mathcal{P}$ can observe but can't modify once $\text{inp}_{\mathcal{P}}$ "committed"

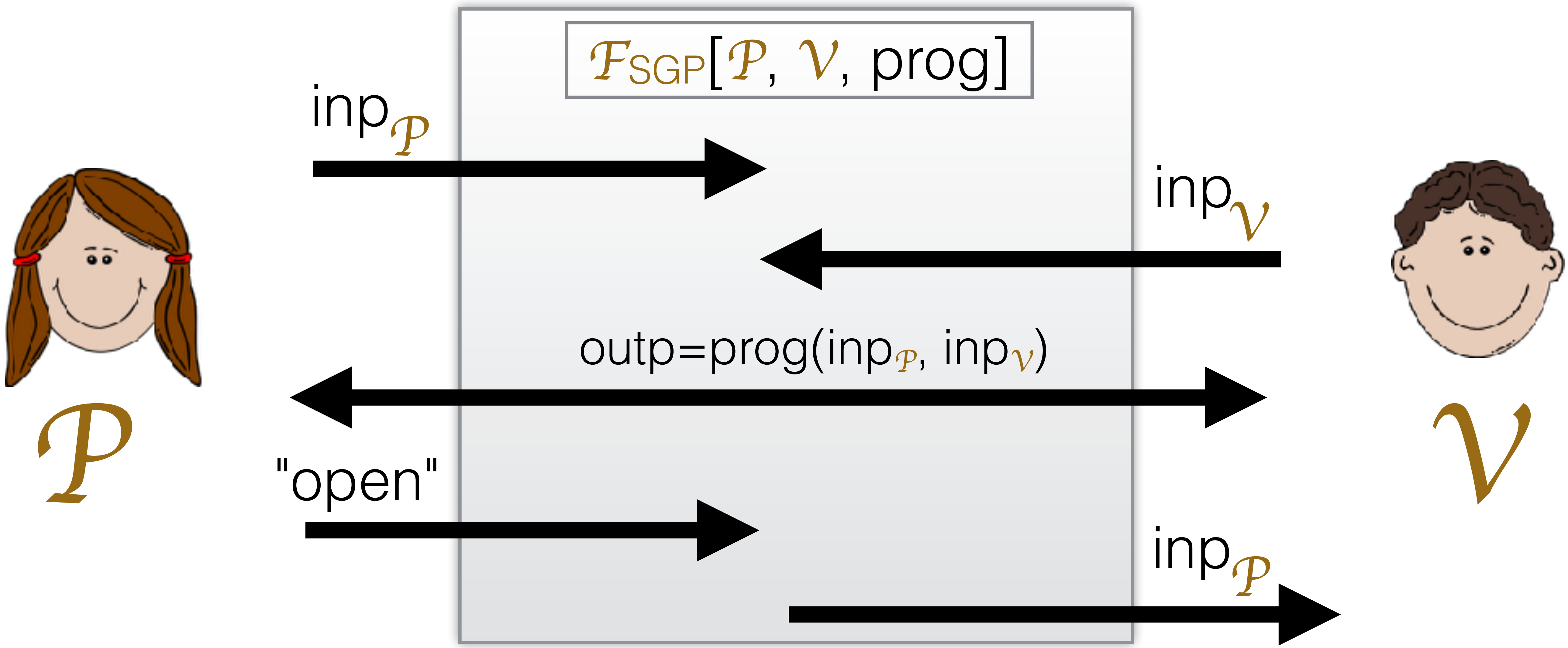


Why "sealed glass" ?

- **Model:** $\mathcal{P}$ can observe but can't modify once $\text{inp}_{\mathcal{P}}$ "committed"

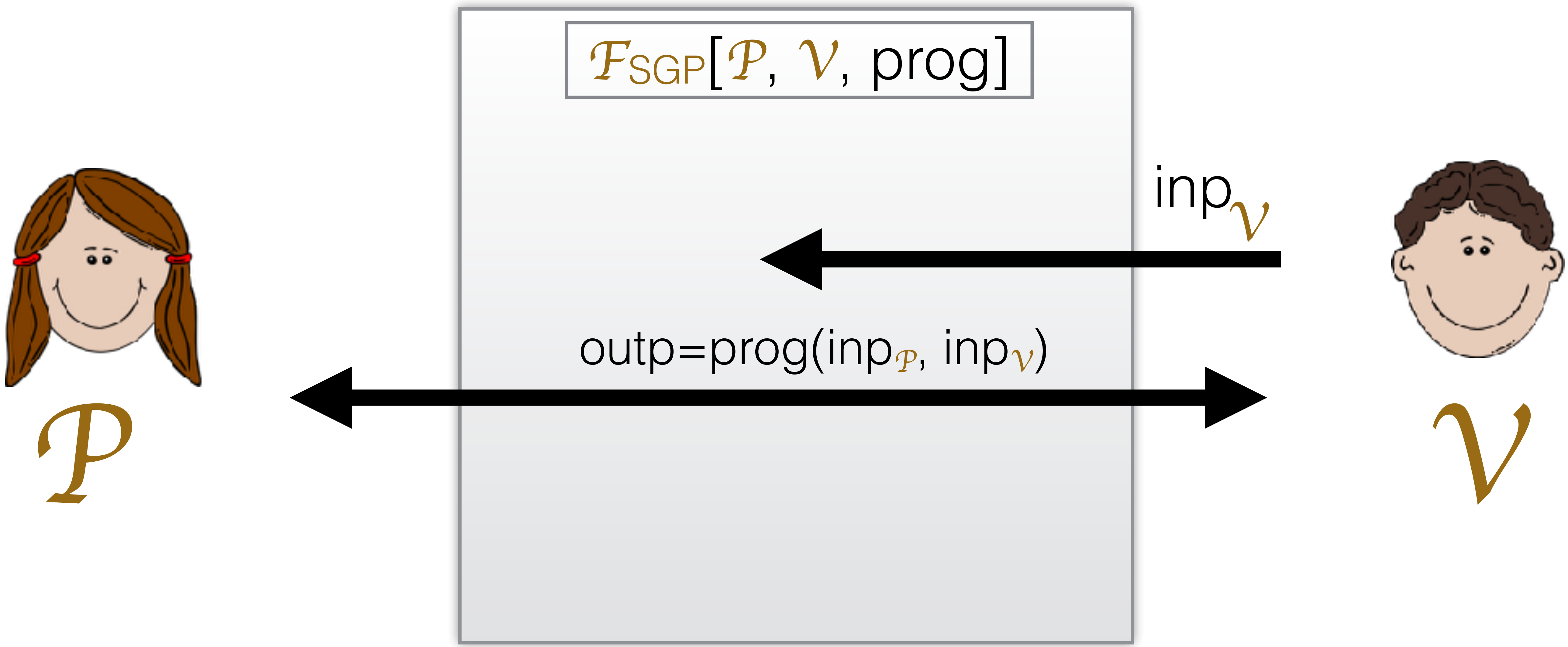


SGP generalizes…



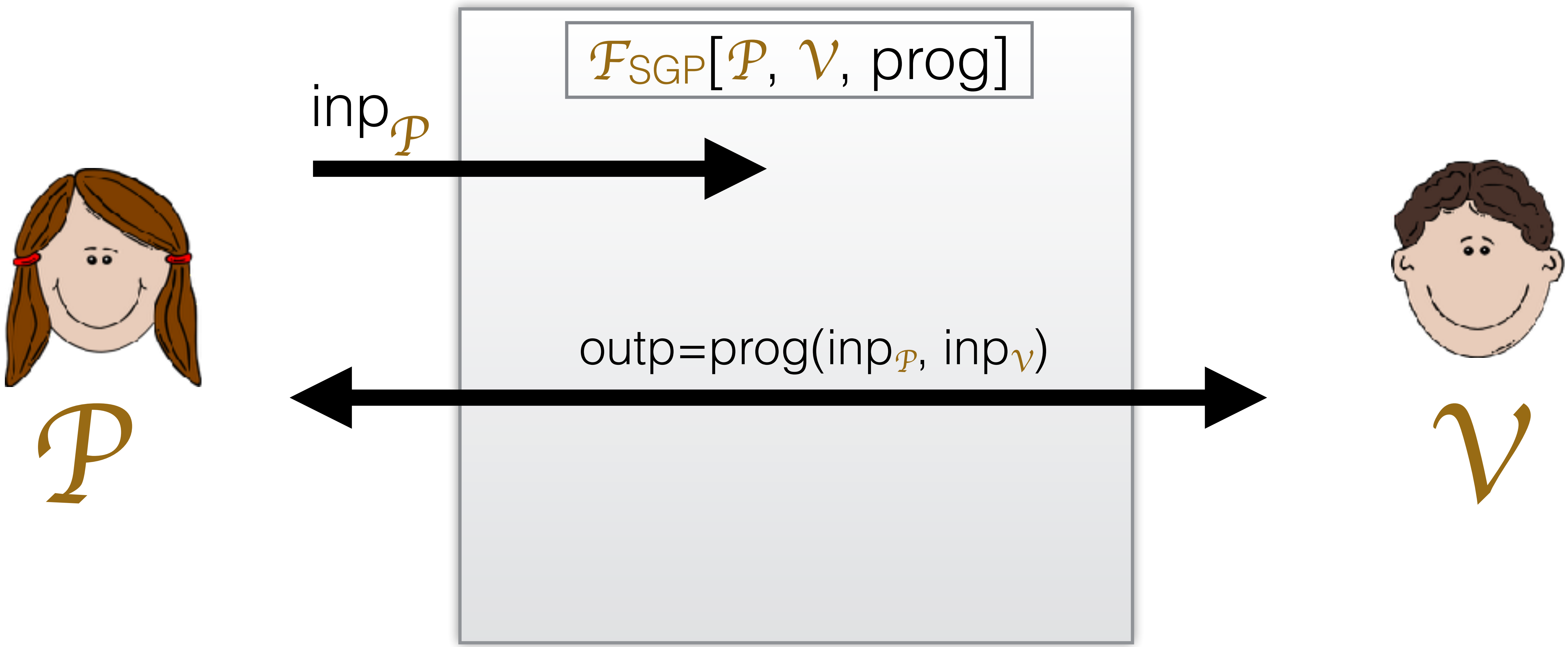
SGP generalizes…

- Verifiable Computing



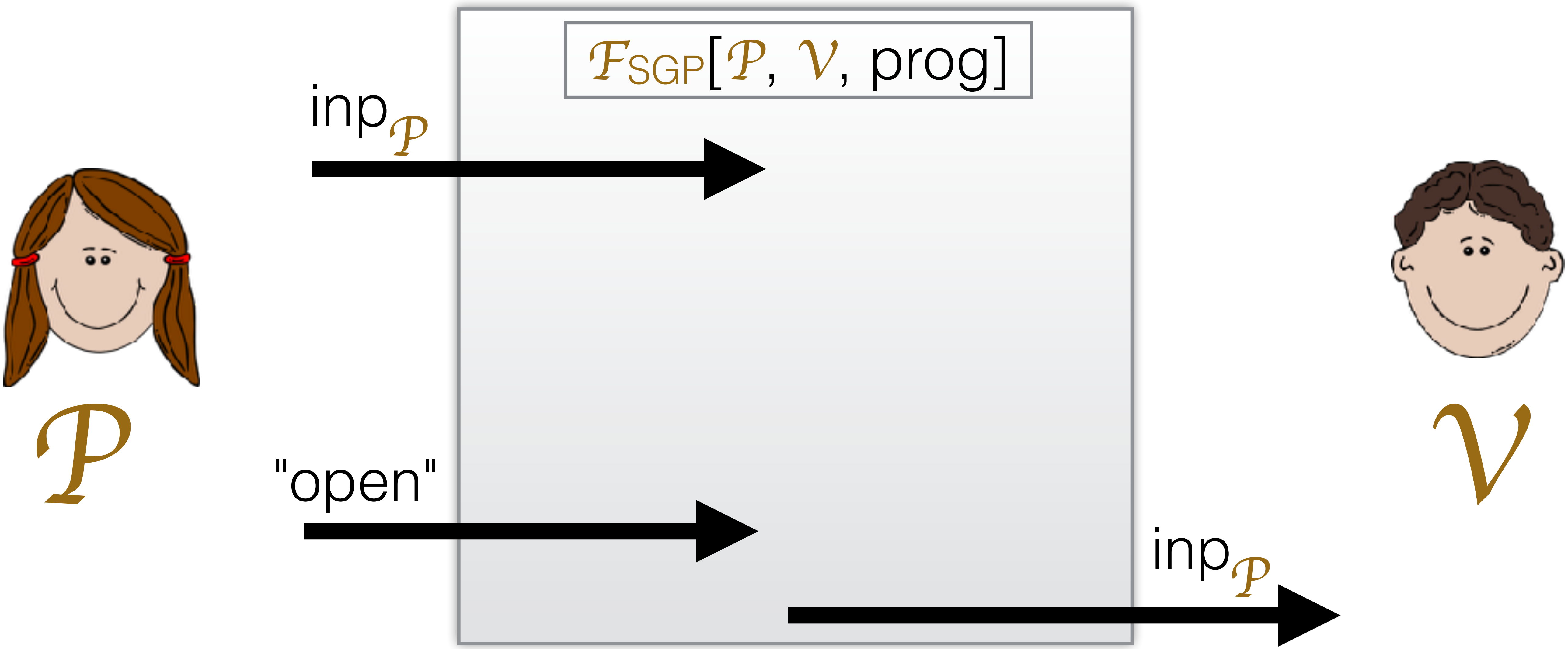
SGP generalizes…

- Verifiable Computing
- ZK proofs



SGP generalizes…

- Verifiable Computing
- ZK proofs
- Commitments, etc.



Application:
Fair bug bounty system

exploit

$Reward
software **S**

seller

buyer

Application:
Fair bug bounty system

Application: Bug bounty

seller

buyer

Application: Bug bounty

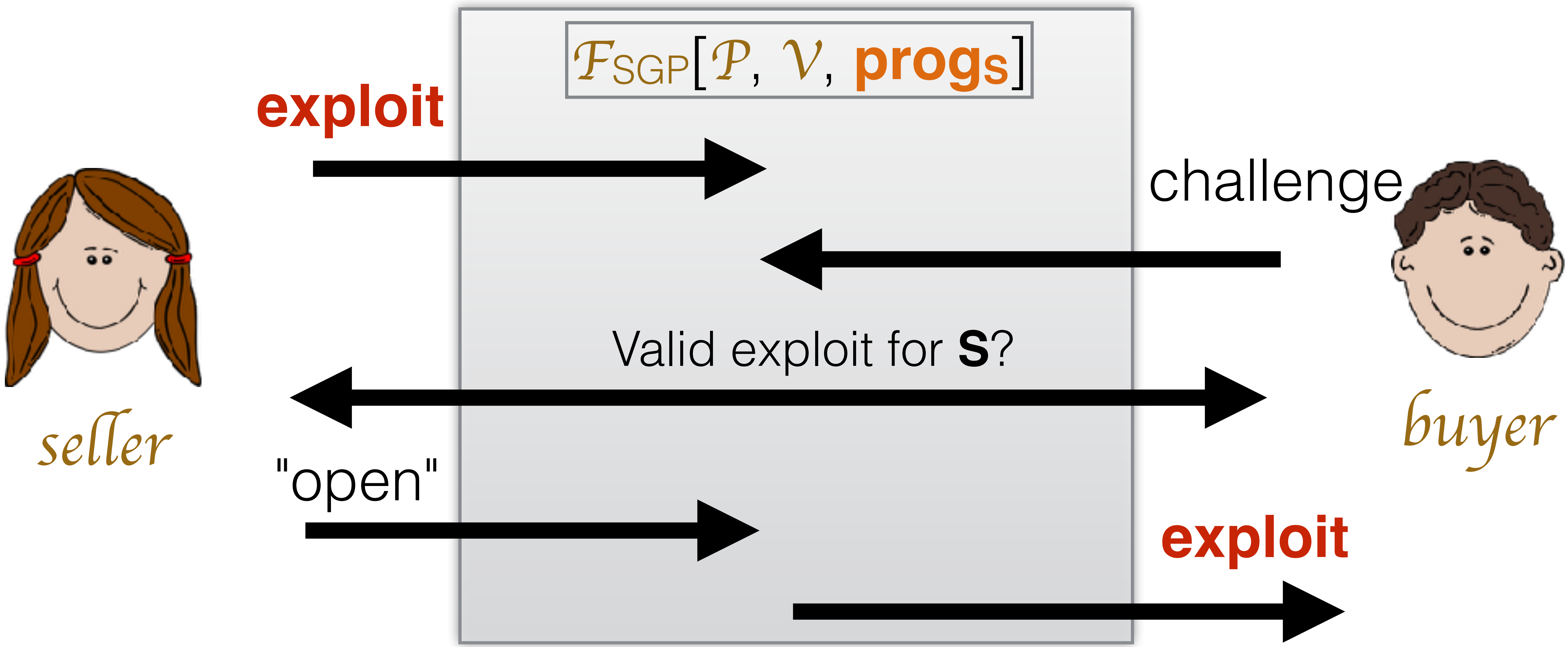
- **prog**$_S$(**exploit**) = "true"
  iff exploit compromises software **S**
  - E.g., SQL injection attack

seller

buyer

Application: Bug bounty

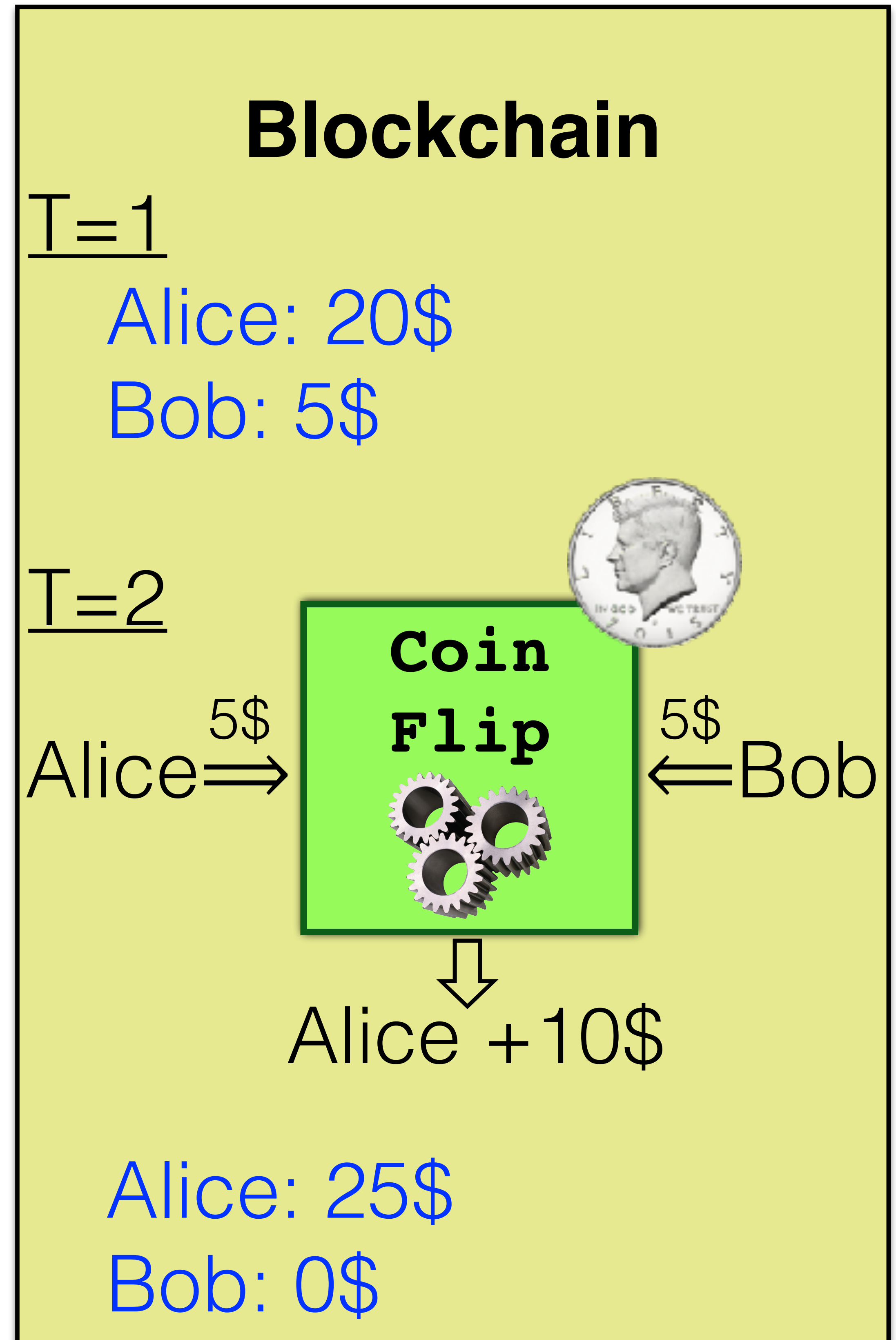
- $\mathbf{prog_S}(\mathbf{exploit})$ = "true" iff exploit compromises software **S**
  - E.g., SQL injection attack



Interlude: Smart-Contracts

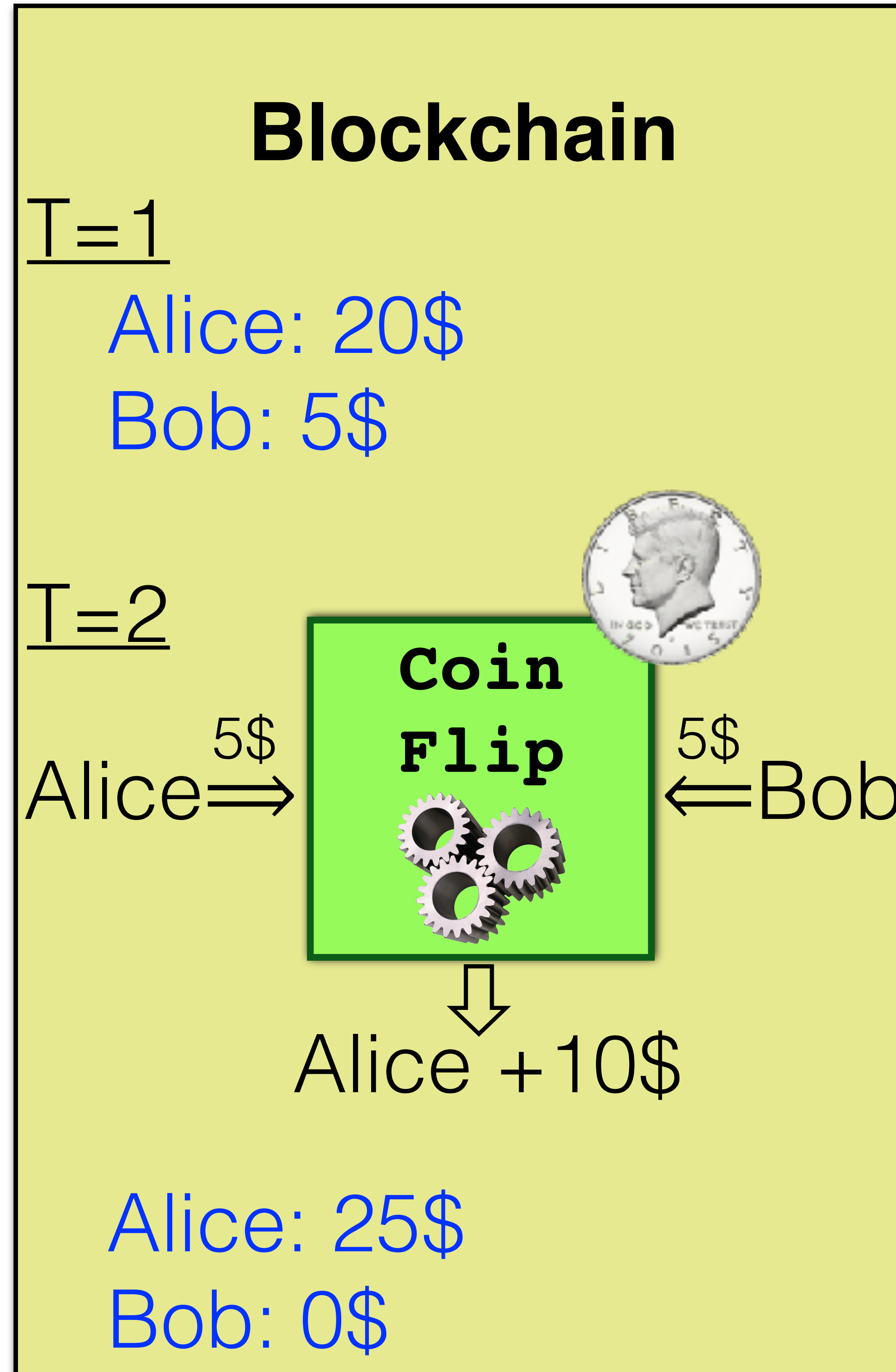
Interlude: Smart-Contracts

- Code executed on blockchain



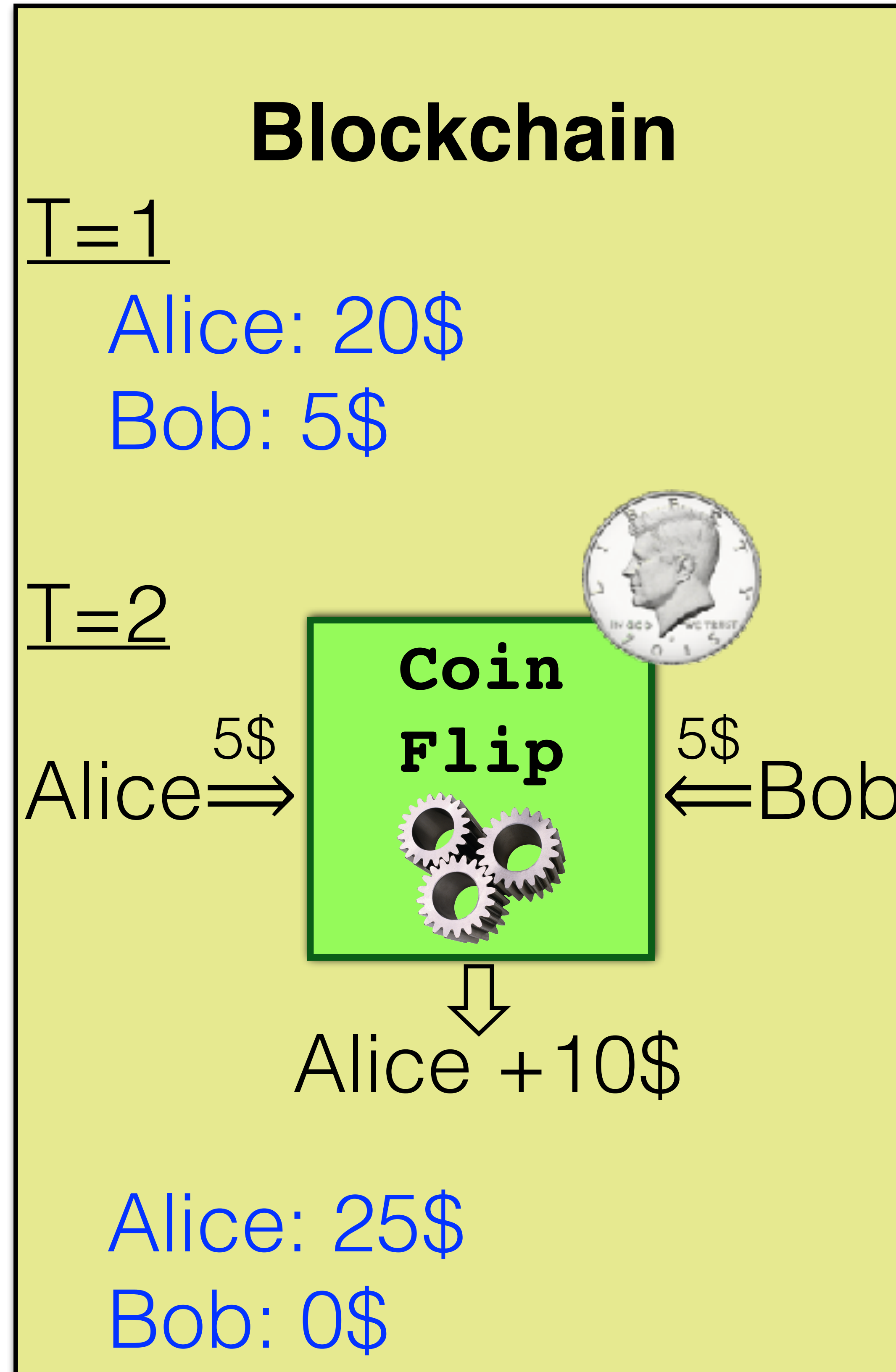
Interlude: Smart-Contracts

- Code executed on blockchain



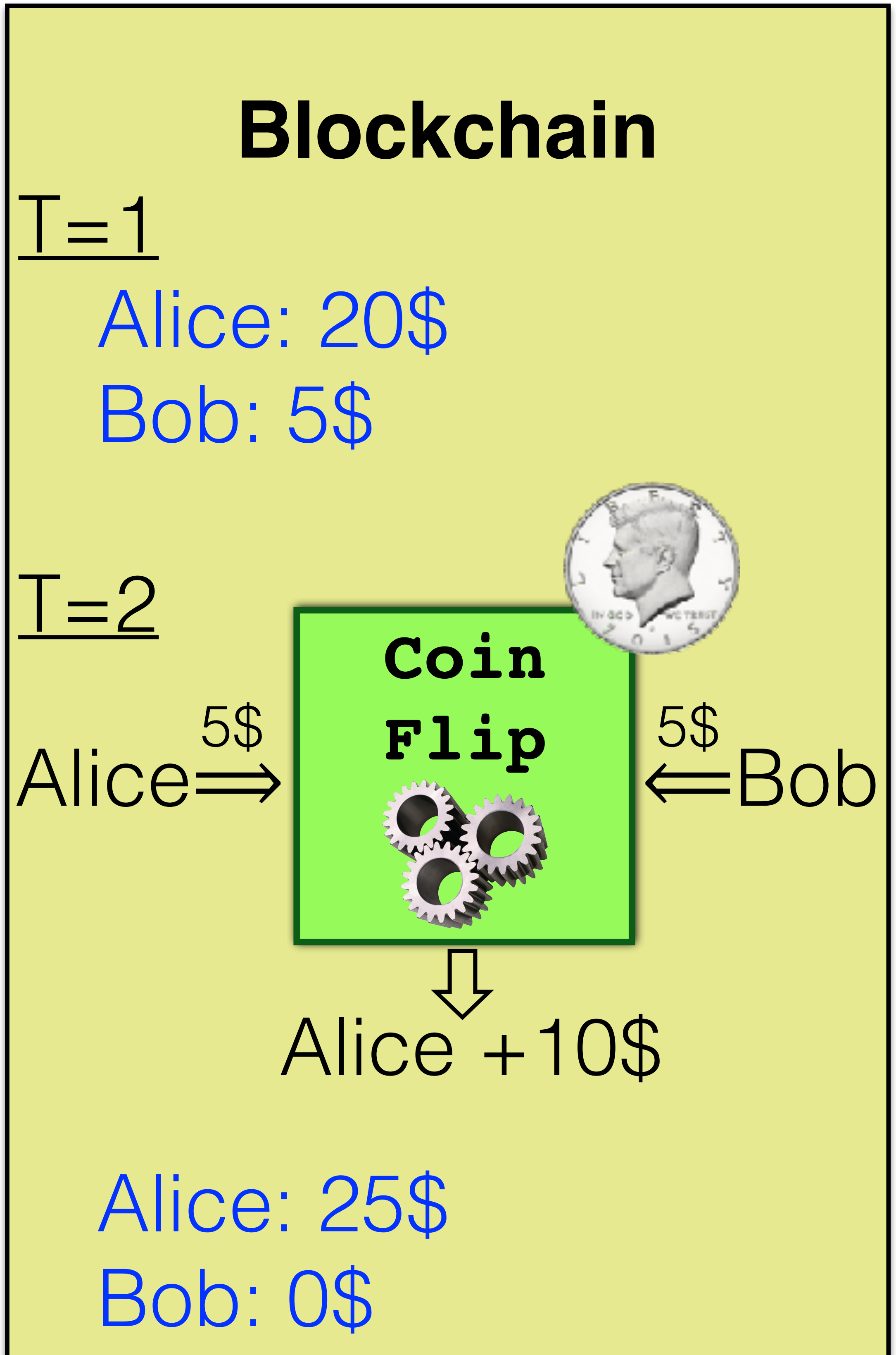
Interlude: Smart-Contracts

- Code executed on blockchain
- Scripted in **Turing-complete language** (e.g. Ethereum)



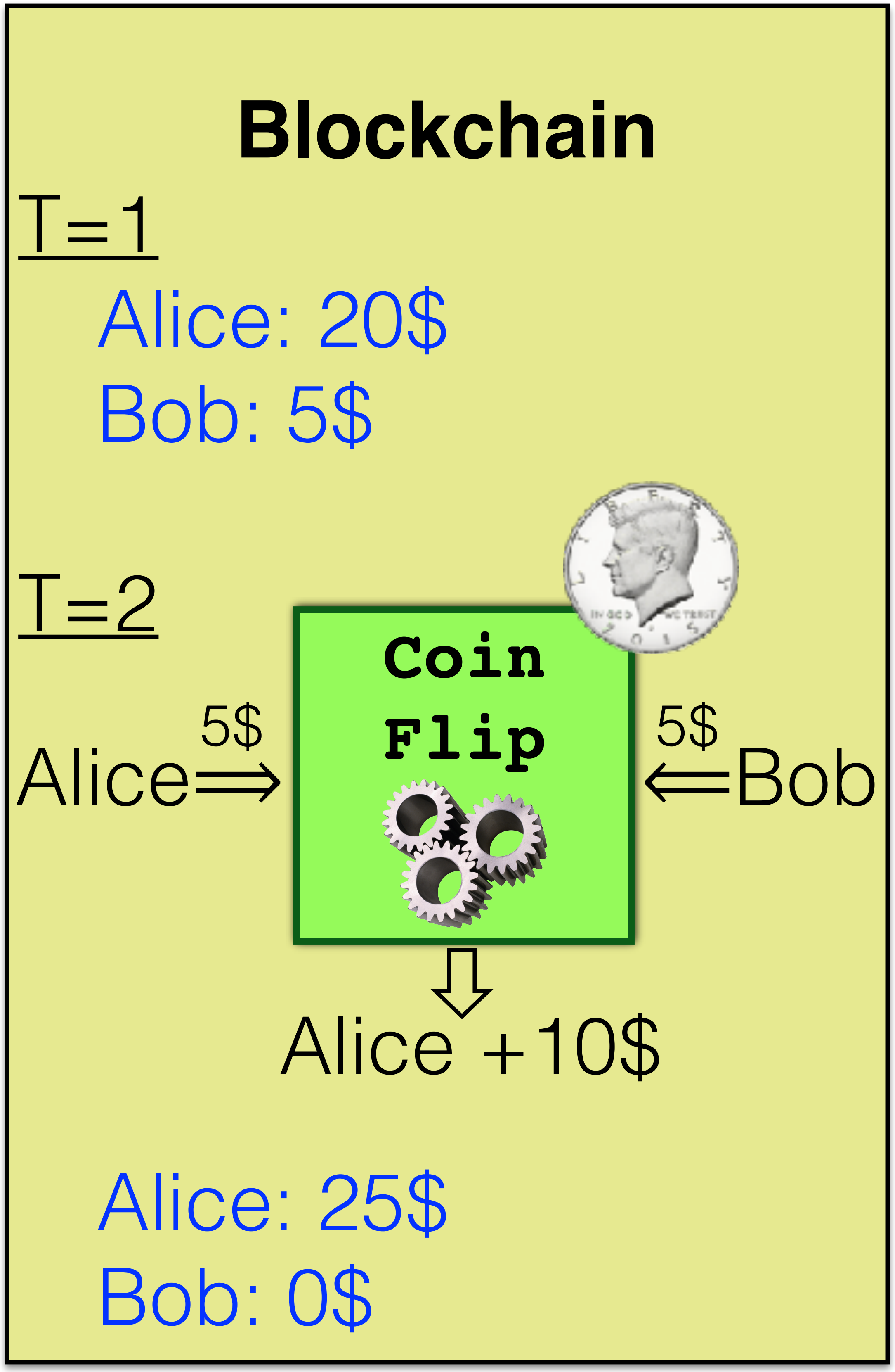
Interlude: Smart-Contracts

- Code executed on blockchain
- Scripted in **Turing-complete language** (e.g. Ethereum)



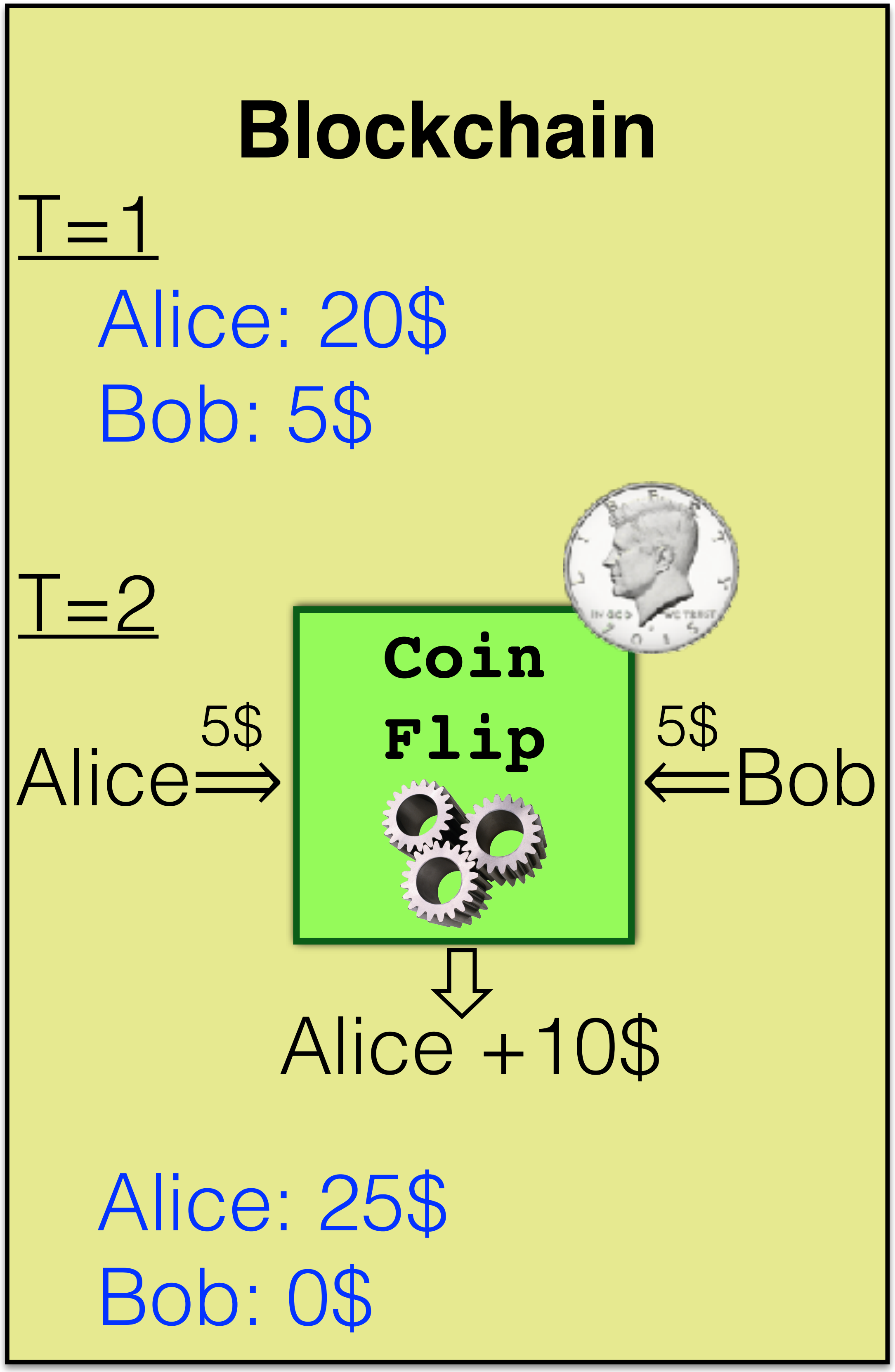
Interlude: Smart-Contracts

- Code executed on blockchain
- Scripted in **Turing-complete language** (e.g. Ethereum)
- Operates on blockchain state
  - Money
  - Local persistent storage



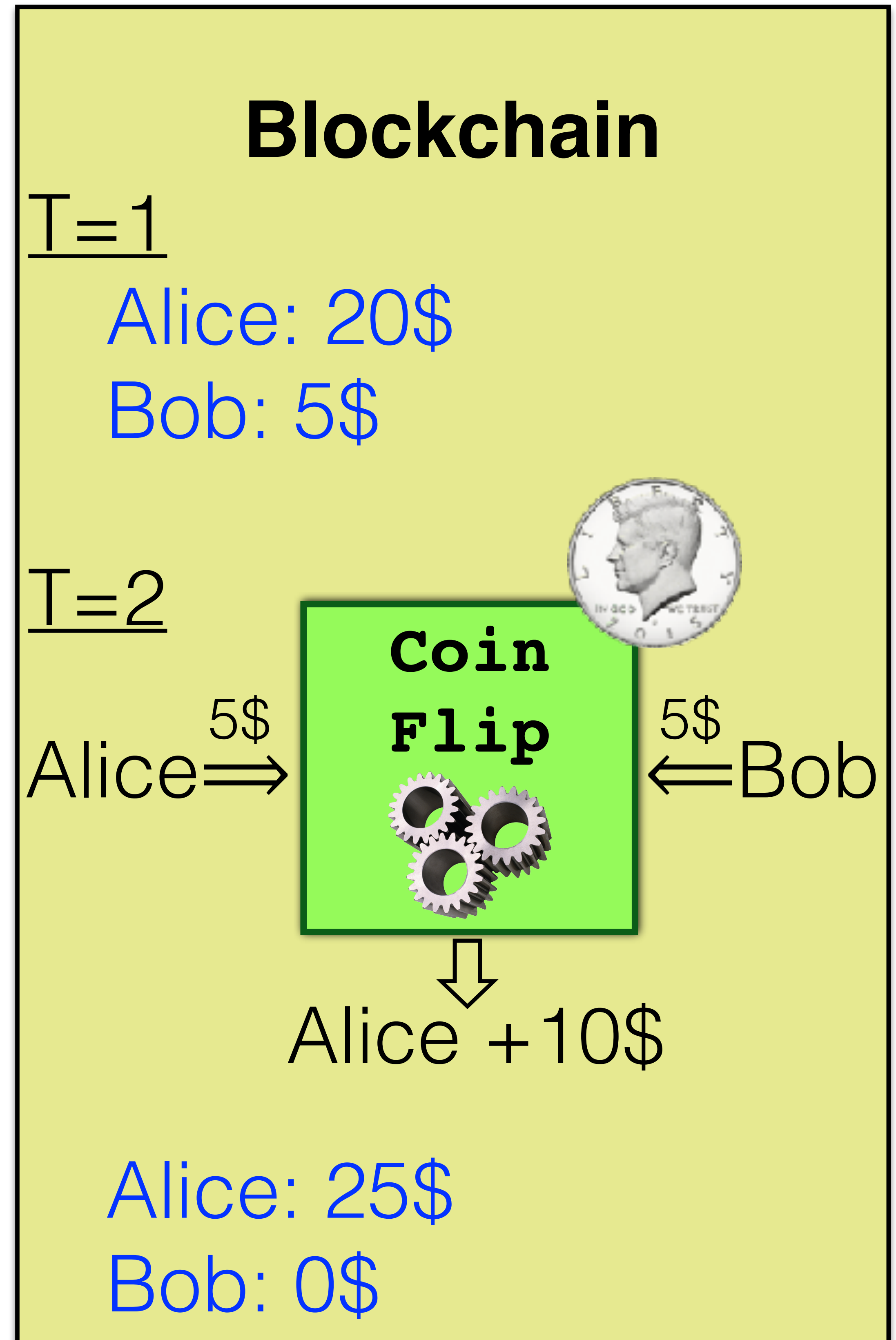
Interlude: Smart-Contracts

- Code executed on blockchain
- Scripted in **Turing-complete language** (e.g. Ethereum)
- Operates on blockchain state
  - Money
  - Local persistent storage



Interlude: Smart-Contracts

- Code executed on blockchain
- Scripted in **Turing-complete language** (e.g. Ethereum)
- Operates on blockchain state
  - Money
  - Local persistent storage
- Contract state is publicly visible

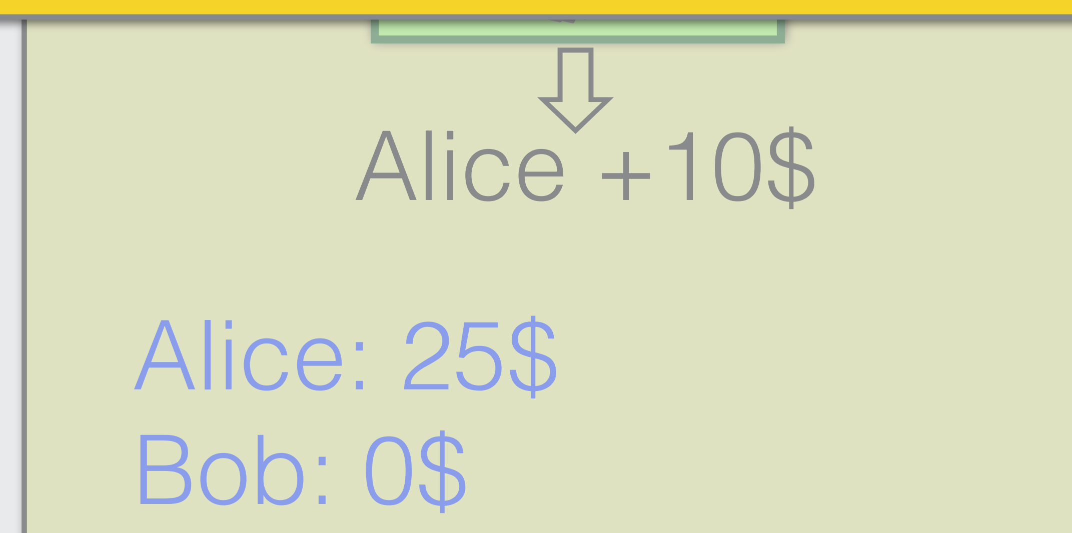


Interlude: Smart-Contracts

- Code executed on blockchain

Abstraction: Smart contract simulates ***trusted third party*** *with* ***public state***.

  - Callable by user accounts
  - Callable by other contracts
- State is publicly visible



End-to-end bug-bounty system

$\mathcal{F}_{\mathrm{SGP}}[\mathcal{P}, \mathcal{V}, \quad]$

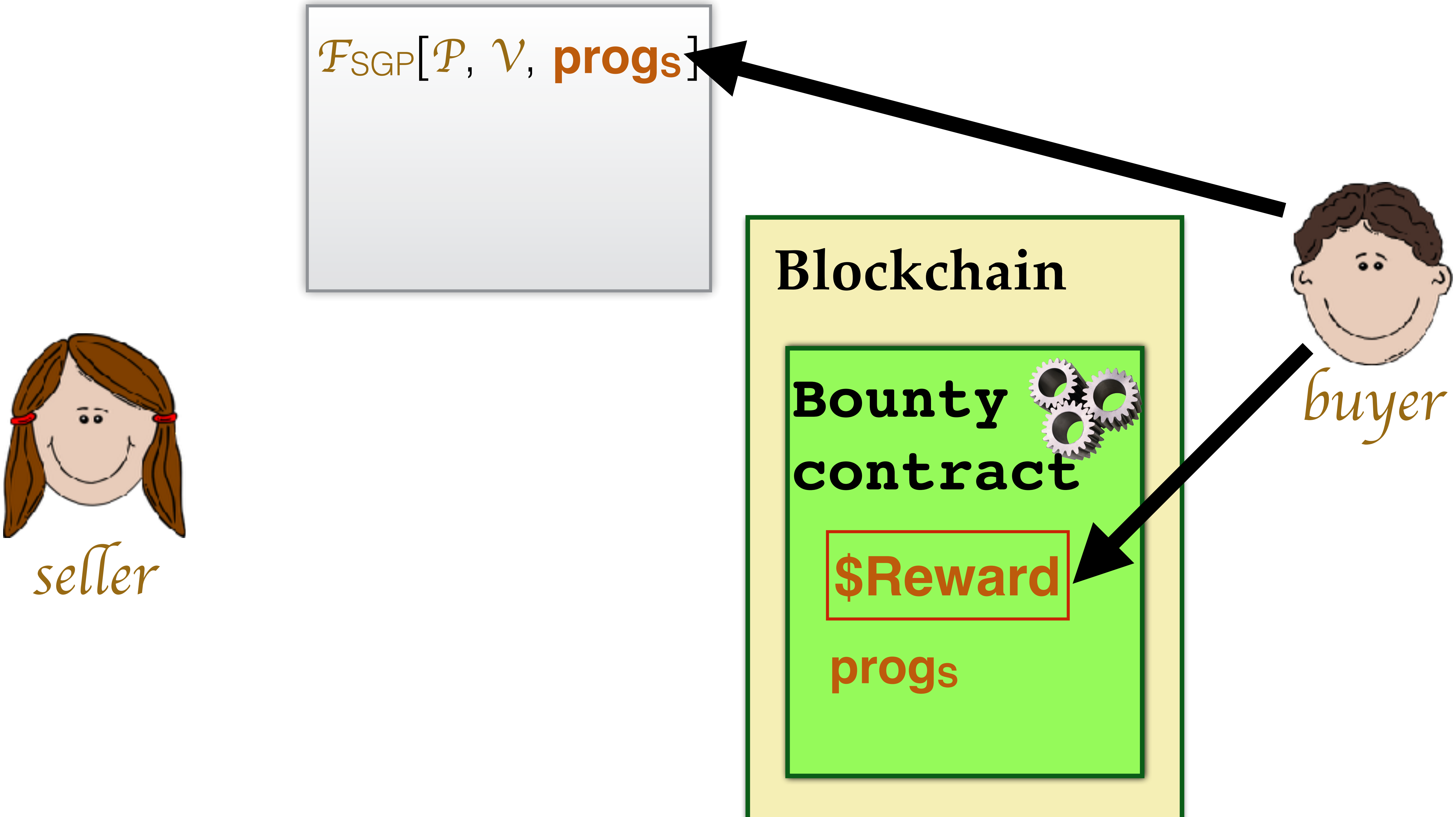
Blockchain

Bounty contract

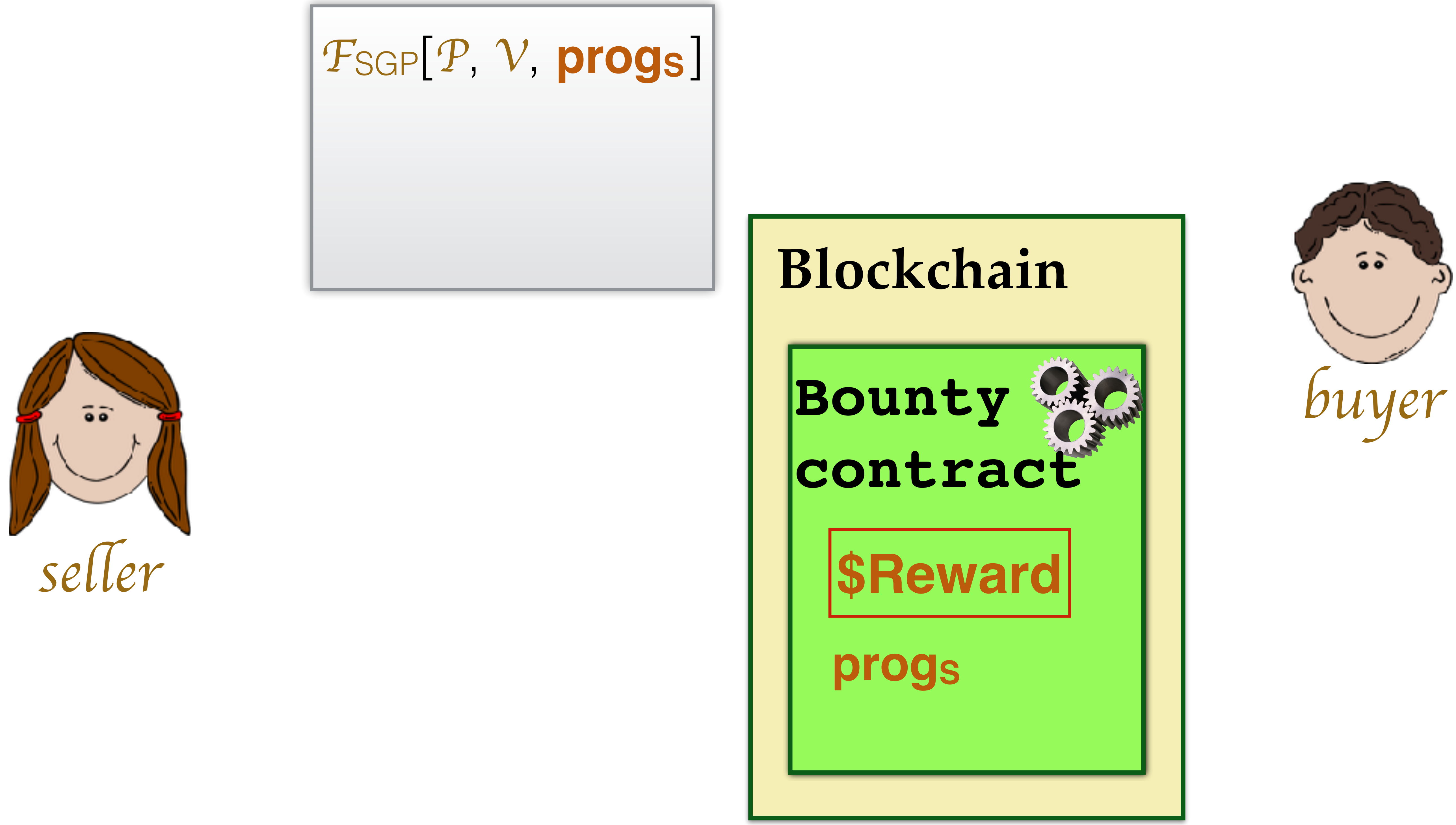
buyer

seller

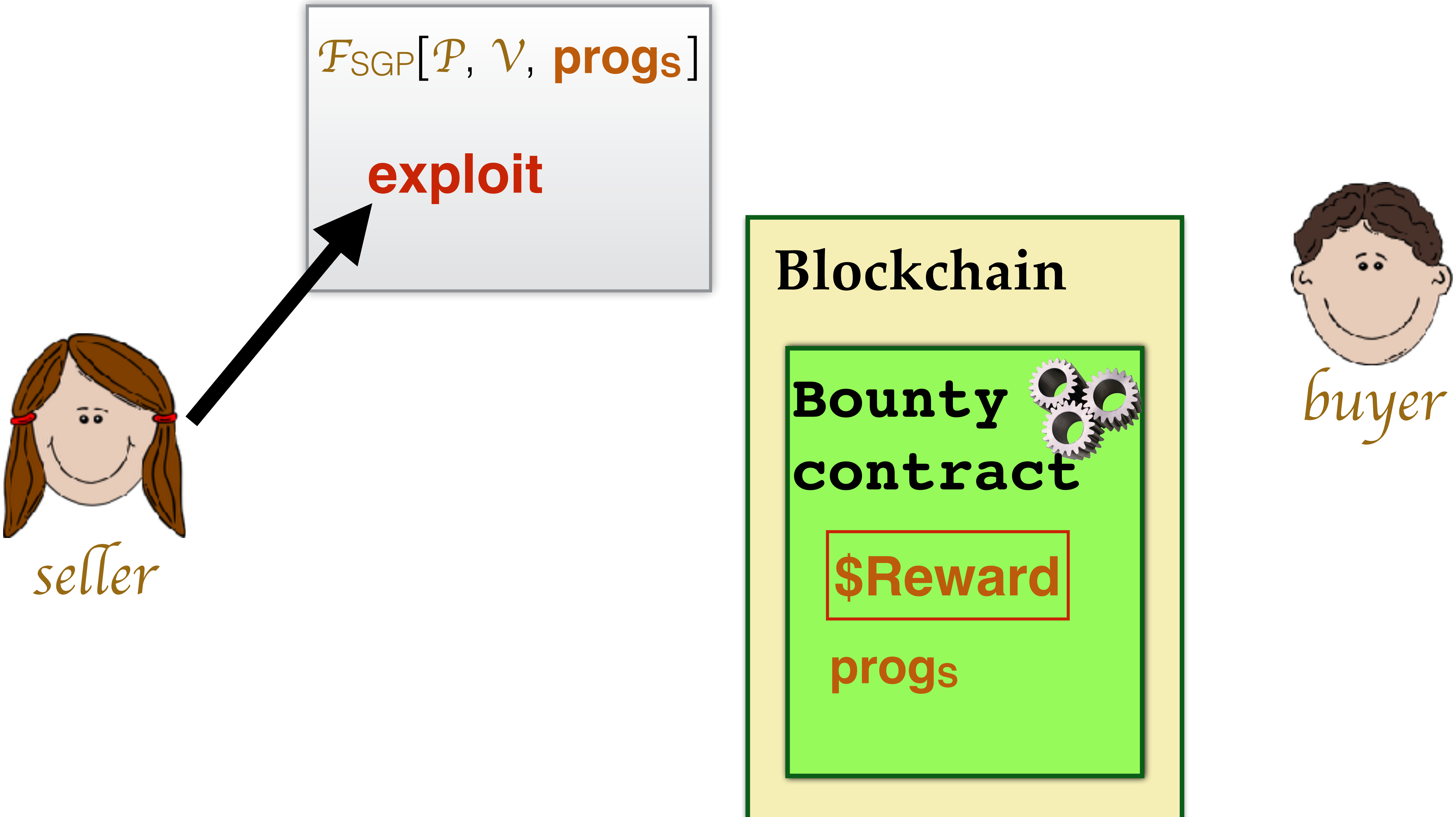
End-to-end bug-bounty system



End-to-end bug-bounty system



End-to-end bug-bounty system



End-to-end bug-bounty system

$\mathcal{F}_{\mathrm{SGP}}[\mathcal{P}, \mathcal{V}, \textbf{progs}]$

exploit

seller

Blockchain

Bounty contract

$Reward

progs

buyer

End-to-end bug-bounty system

$\mathcal{F}_{\mathrm{SGP}}[\mathcal{P}, \mathcal{V}, \textbf{progs}]$

exploit ✔

seller

Blockchain

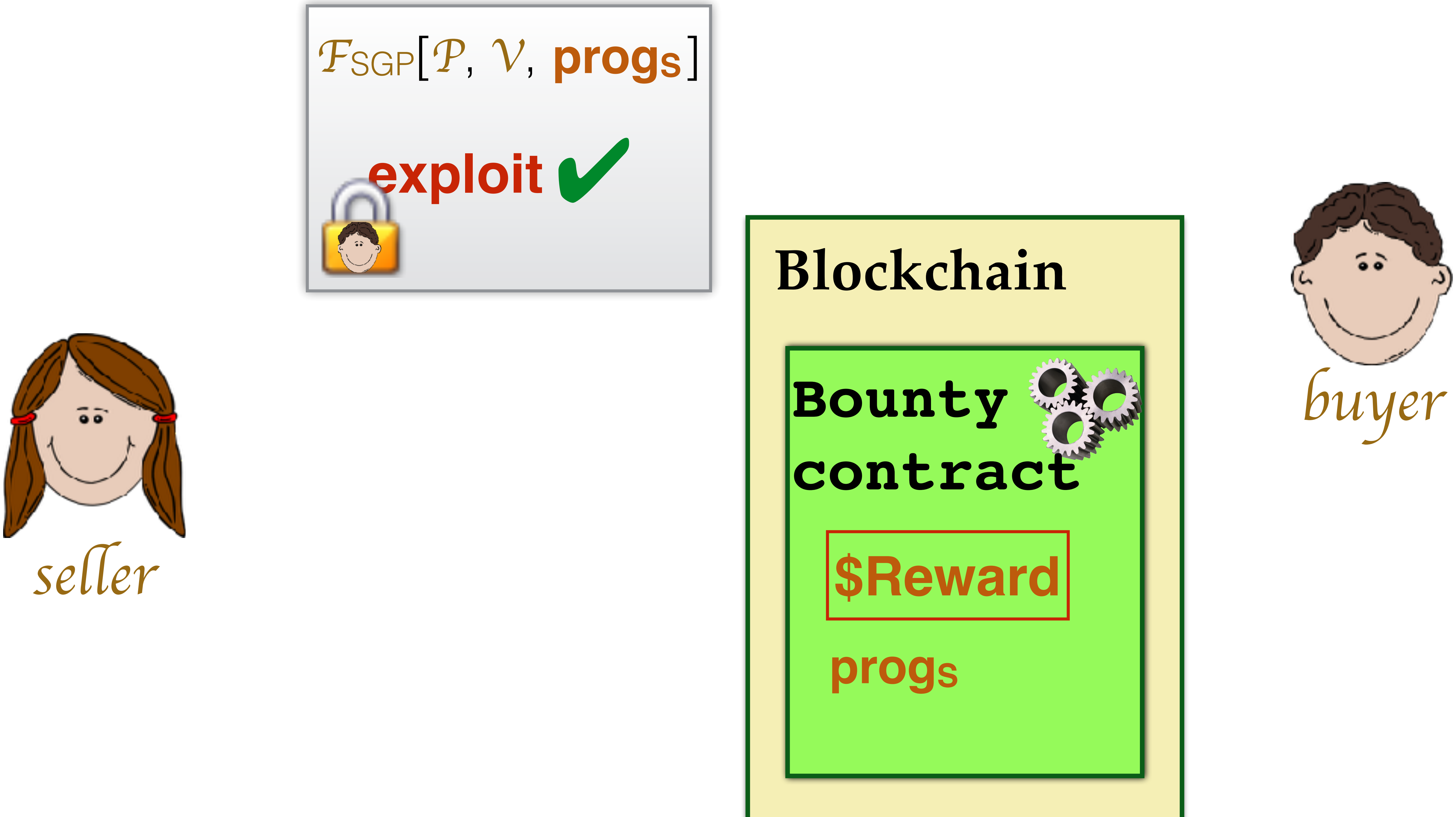
Bounty contract

$Reward

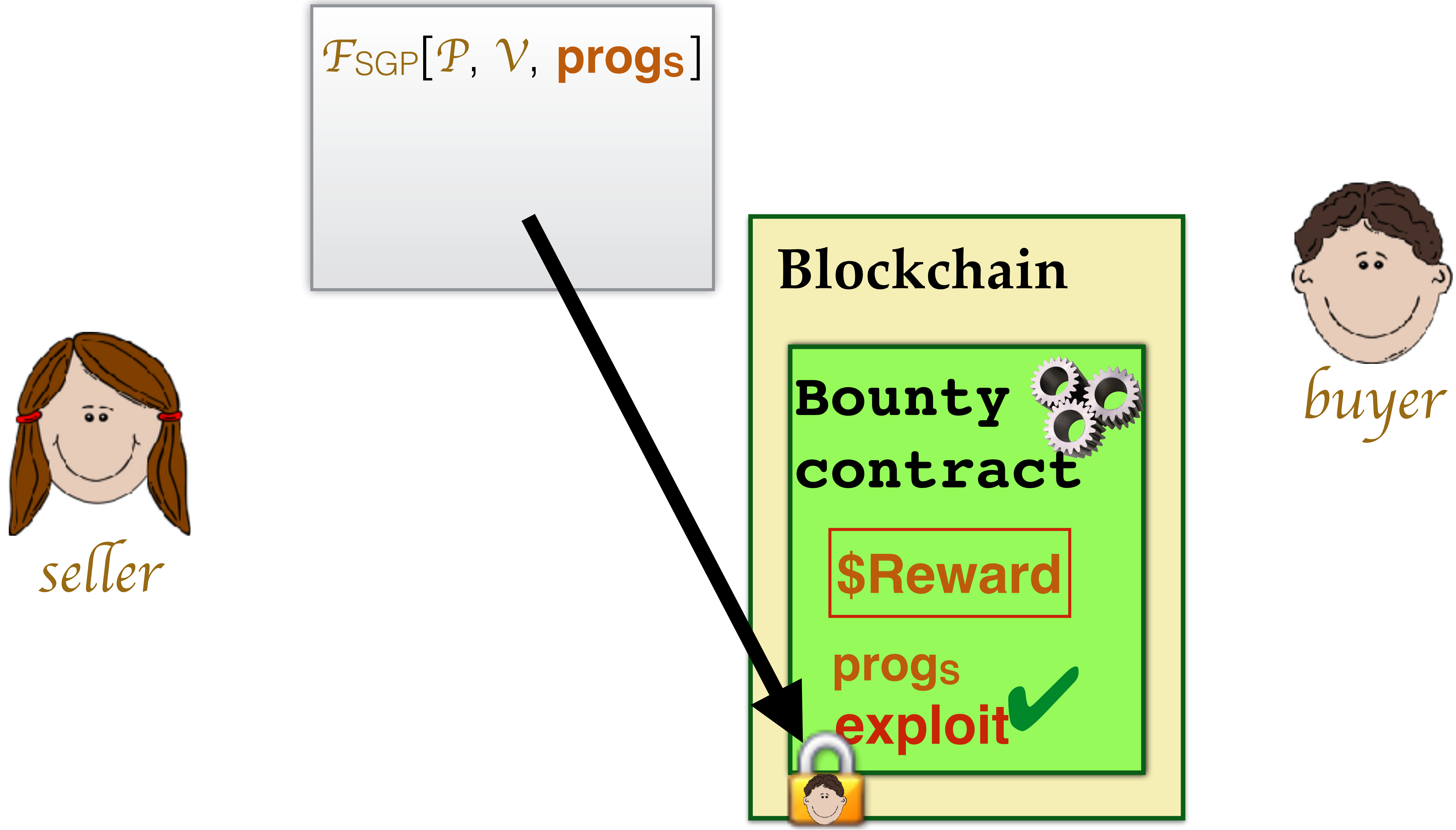
progs

buyer

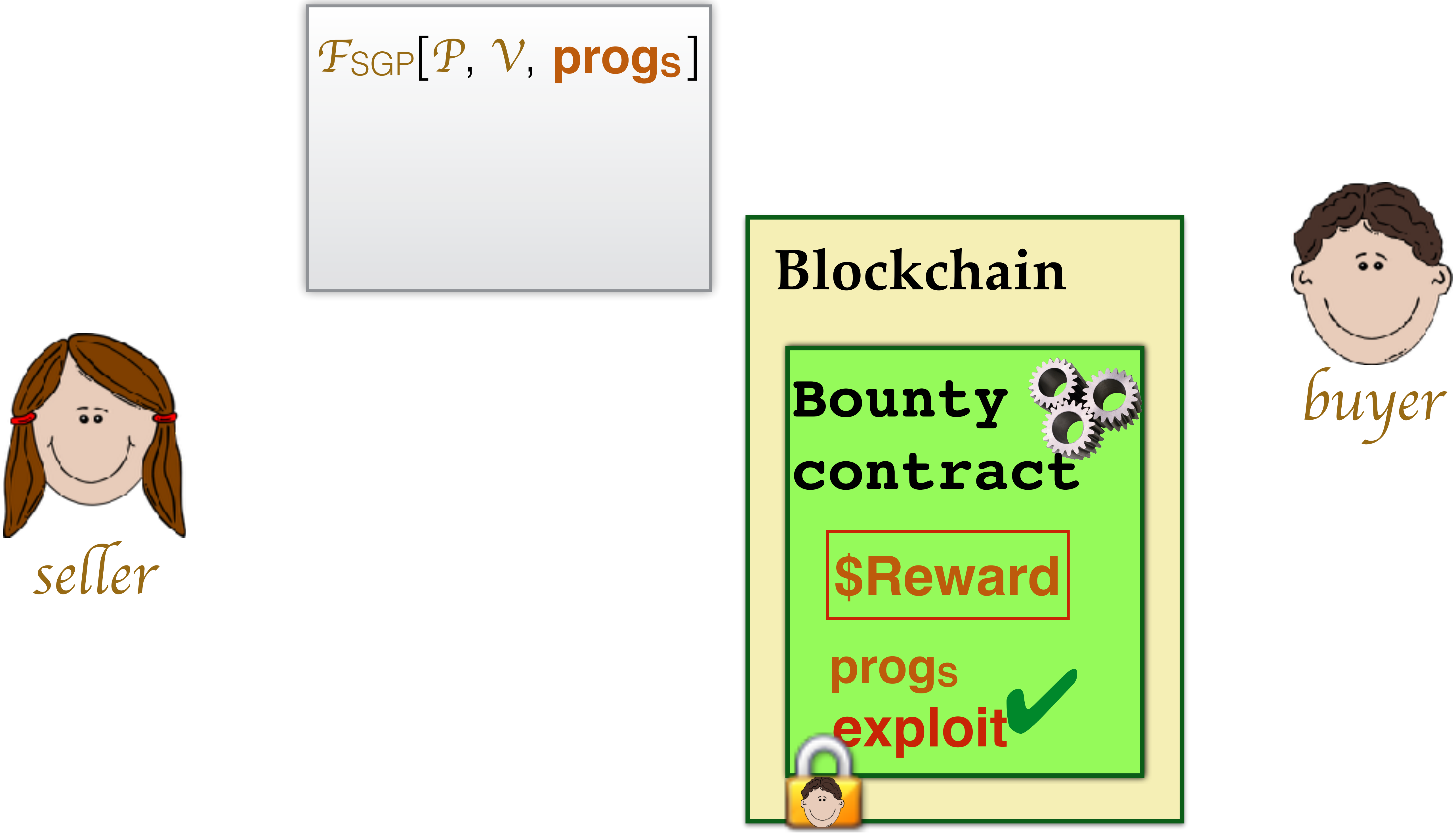
End-to-end bug-bounty system



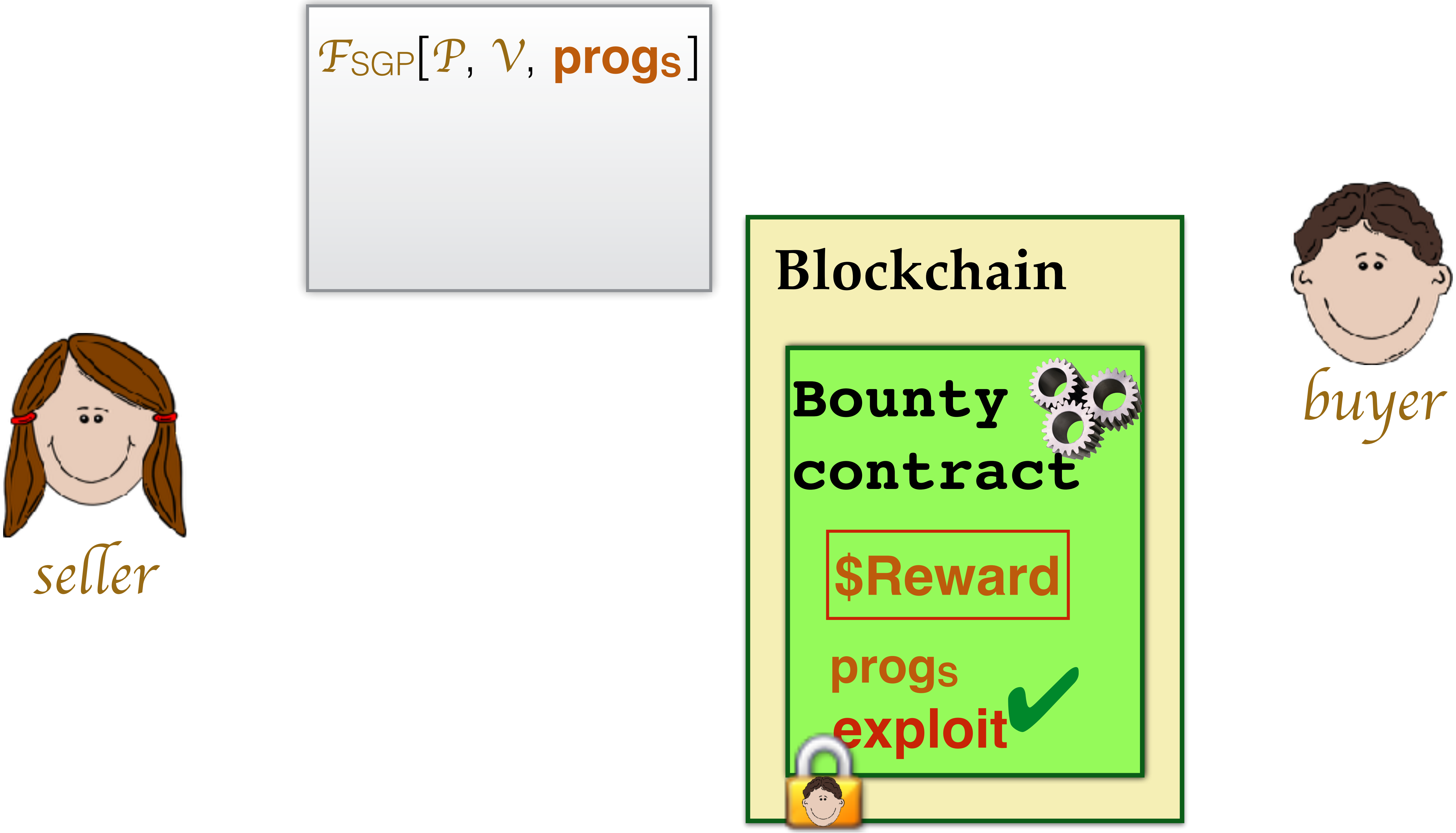
End-to-end bug-bounty system



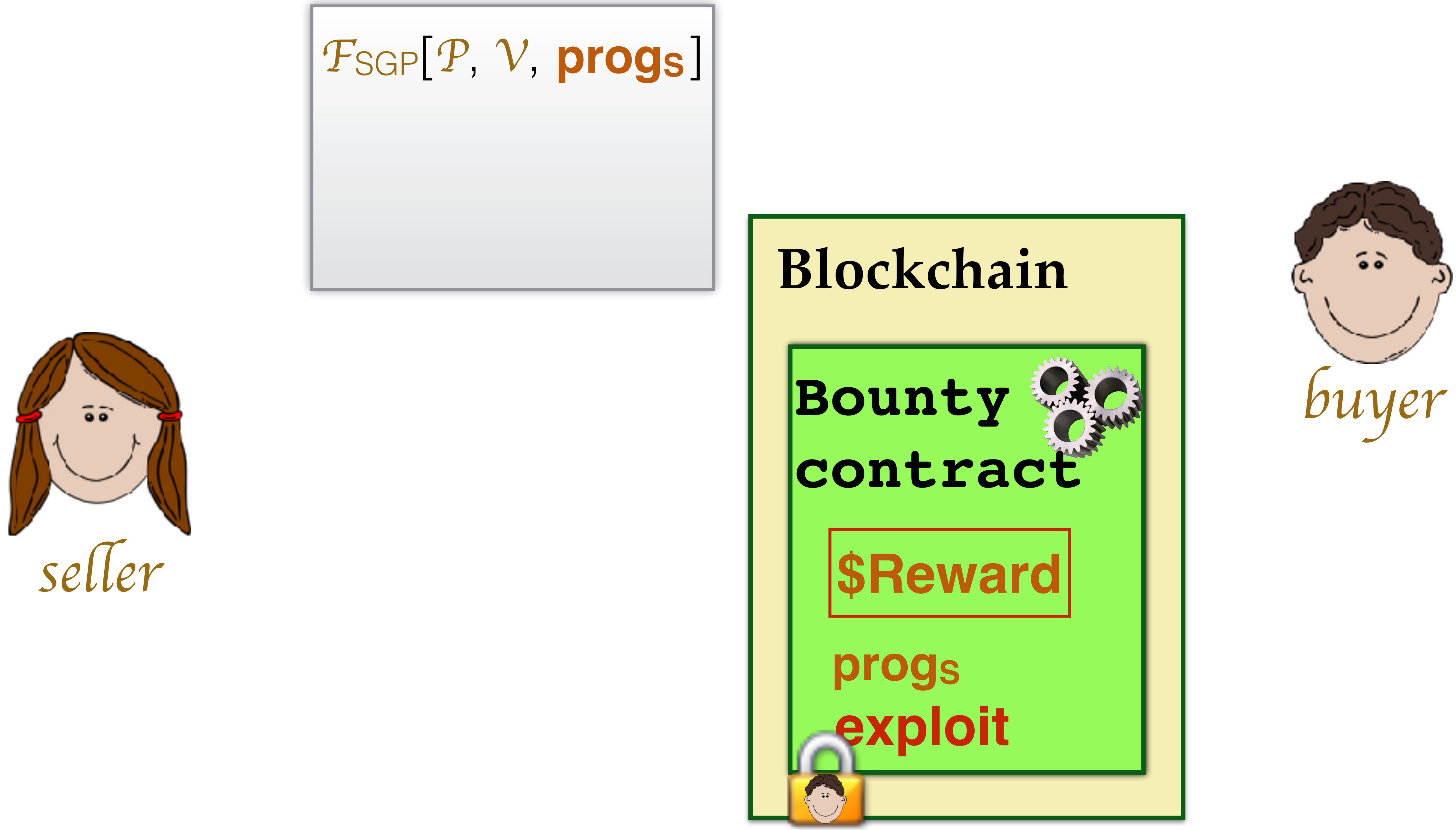
End-to-end bug-bounty system



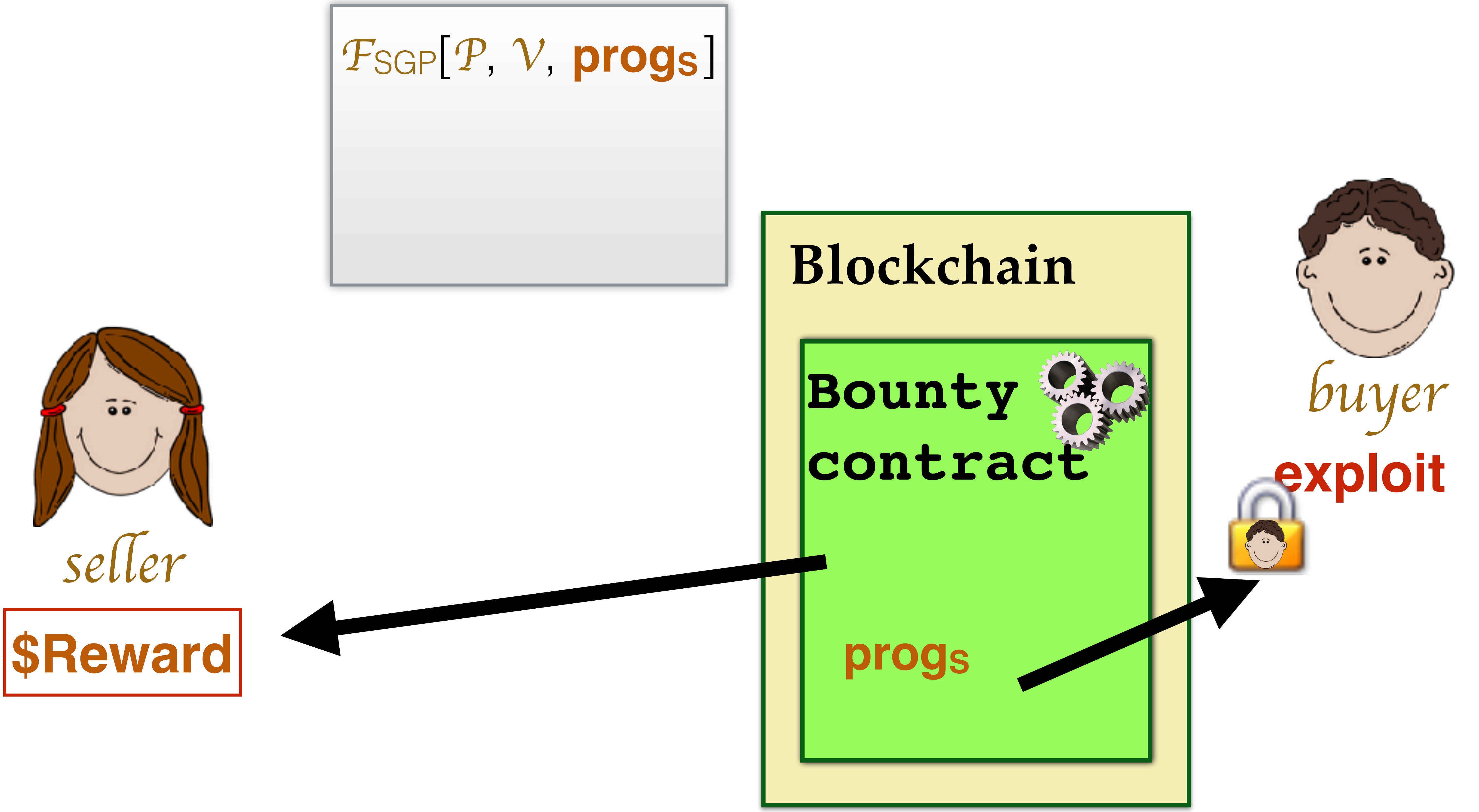
End-to-end bug-bounty system



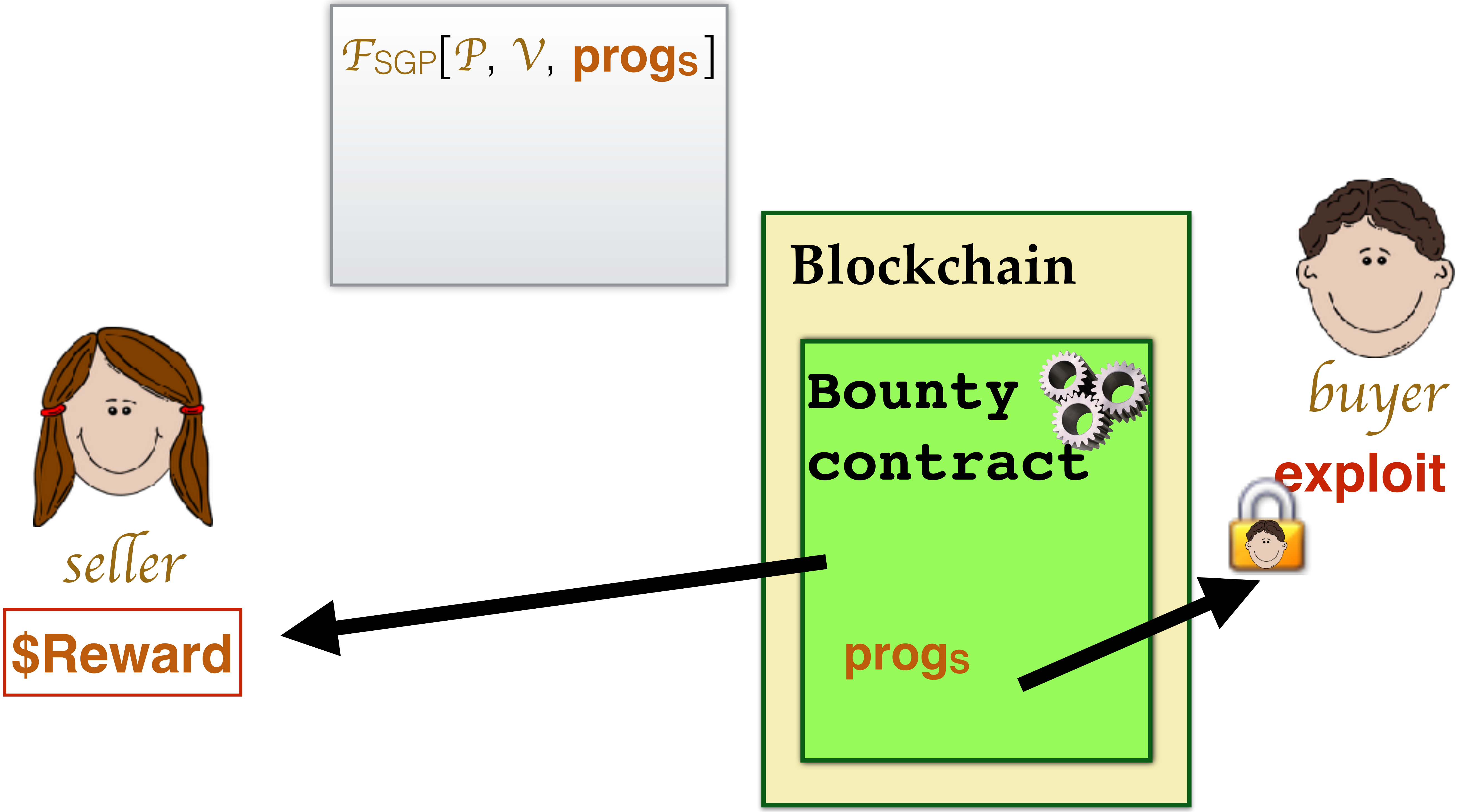
End-to-end bug-bounty system



End-to-end bug-bounty system



End-to-end bug-bounty system



- Fair exchange: **$Reward** for **exploit** against **S**

End-to-end bug-bounty system

End-to-end bug-bounty system

Properties:

1. **Fair exchange: $Reward** iff delivered **exploit**

End-to-end bug-bounty system

Properties:

1. **Fair exchange:** **$Reward** iff delivered **exploit**
2. **Confidentiality:** **exploit** encrypted under public key of *buyer*

End-to-end bug-bounty system

Properties:

1. **Fair exchange:** **$Reward** iff delivered **exploit**
2. **Confidentiality:** **exploit** encrypted under public key of *buyer*
3. **Guaranteed payment*:** *buyer* will pay at least one valid *seller* before specified deadline
   → Prevents bug-bounty competition from being unfairly terminated

**ZK-snark-based Bitcoin systems can't achieve this one*

Marketplaces for what kinds of bugs?

Marketplaces for what kinds of bugs?

- In principle, for any system executable / simulatable in enclave

Marketplaces for what kinds of bugs?

- In principle, for any system executable / simulatable in enclave

- In paper:
  - SQL injection attacks

Marketplaces for what kinds of bugs?

- In principle, for any system executable / simulatable in enclave
- In paper:
  - SQL injection attacks
    - Facebook Proxygen library fronting SQLite
  - Certificate Validation Logic conflicts (“Frankencerts”)

Marketplaces for what kinds of bugs?

- In principle, for any system executable / simulatable in enclave
- In paper:
  - SQL injection attacks
    - Facebook Proxygen library fronting SQLite
  - Certificate Validation Logic conflicts ("Frankencerts")
    - OpenSSL and mbedTLS
  - MITM attacks against TLS handshakes
    - Simulation environment in which exploit attacks simulated handshake between server and honest user
    - (Assuming SGX v2)

Summary

Summary

- **Transparent enclave execution (TEE)**
  - Lots of fun things can be done without confidentiality!
  - Natural extensions to allow for some functionalities to remain hidden from host (e.g., crypto primitives)

Summary

- **Transparent enclave execution (TEE)**
  - Lots of fun things can be done without confidentiality!
  - Natural extensions to allow for some functionalities to remain hidden from host (e.g., crypto primitives)
- **Combining SGX with smart-contracts**
  - Can provide guarantees (e.g. fair-exchange) not achievable with “traditional” crypto
  - Difficult to get right! Both formally and in practice

Summary

- **Transparent enclave execution (TEE)**
  - Lots of fun things can be done without confidentiality!
  - Natural extensions to allow for some functionalities to remain hidden from host (e.g., crypto primitives)
- **Combining SGX with smart-contracts**
  - Can provide guarantees (e.g. fair-exchange) not achievable with “traditional” crypto
  - Difficult to get right! Both formally and in practice

Sealed Glass Proofs

https://eprint.iacr.org/2016/635

Formal Abstractions for Attested Execution Secure Processors

https://eprint.iacr.org/2016/1027