Giving State to the Stateless:

Augmenting Trustworthy Computation with Ledgers

Gabriel Kaptchuk, Matthew Green, and Ian Miers

Johns Hopkins University

JOHNS HOPKINS
UNIVERSITY

BEWARE!

HERE BE BLOCKCHAINS!

JOHNS HOPKINS
UNIVERSITY

BEWARE!

HERE BE SGX!

JOHNS HOPKINS
UNIVERSITY

JOHNS HOPKINS
UNIVERSITY

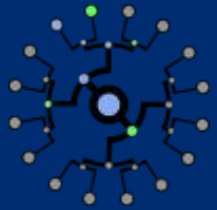
BEWARE!

HERE BE SGX!

JOHNS HOPKINS
UNIVERSITY

Why care about the bogeymen?

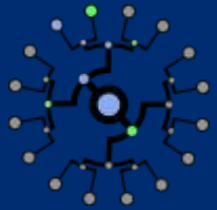
- Ledgers exist *in practice* and they aren't going away
  - Blockchains
  - Google Certificate Transparency Log



- Trusted Execution Environments exist *in practice*

Why care about the bogeymen?

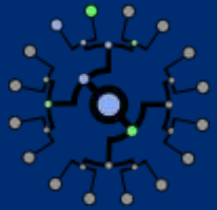
- Ledgers exist *in practice* and they aren't going away
  - Blockchains
  - Google Certificate Transparency Log



- Trusted Execution Environments exist *in practice* … kinda?

Why care about the bogeymen?

- Ledgers exist *in practice* and they aren't going away
  - Blockchains
  - Google Certificate Transparency Log



- Trusted Execution Environments exist *in practice* … kinda?
  - Intel SGX and ARM Trustzone
  - Software only obfuscation
  - FPGA style hardware with burned keys

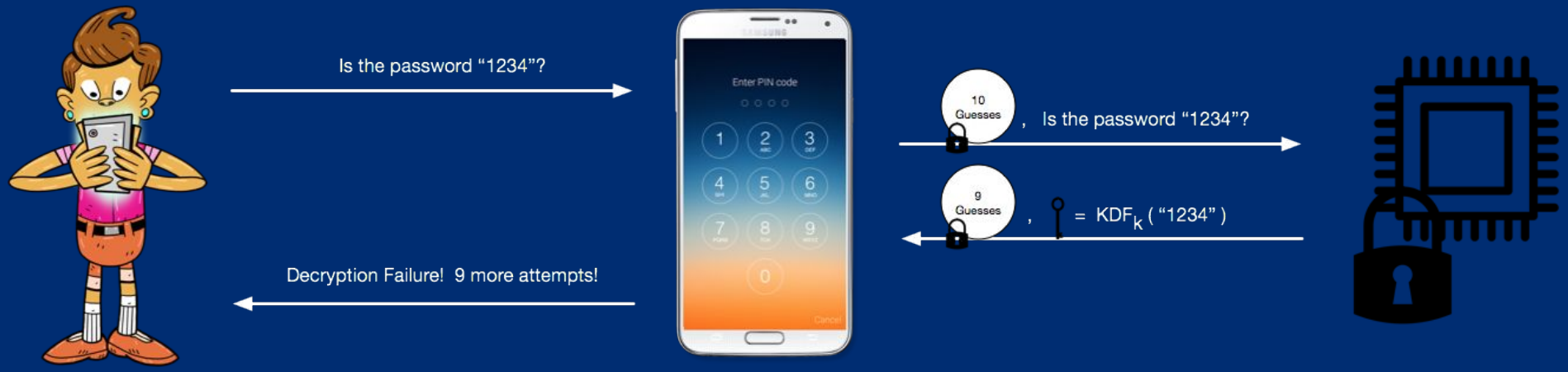
How can TEEs augment ledgers?
vs
How can ledgers augment TEEs?

Rewind Protection

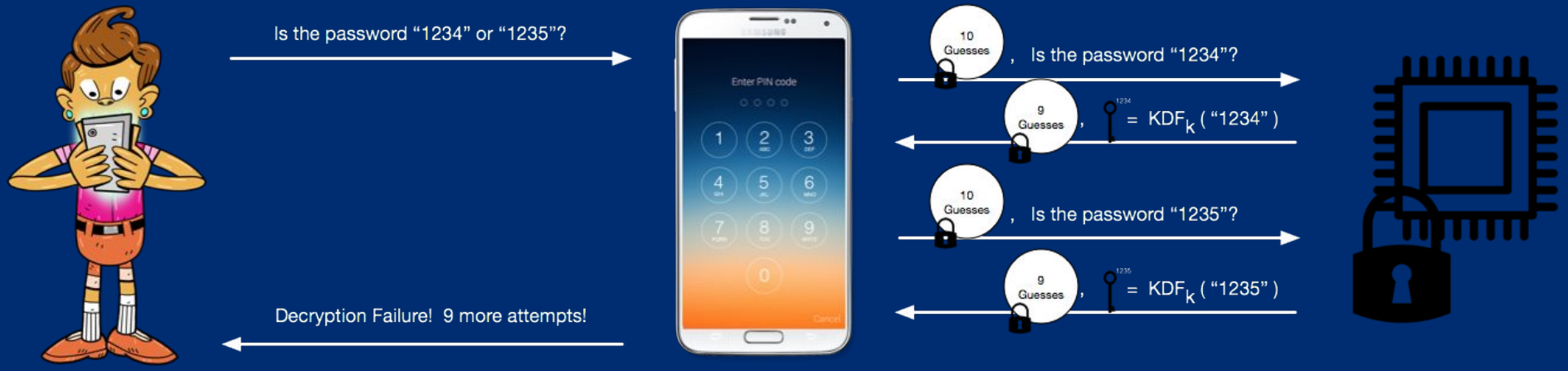
Is the password “1234”?

Nope! You have 9 more attempts!

Rewind Protection



Rewind Protection



Rewind Protection

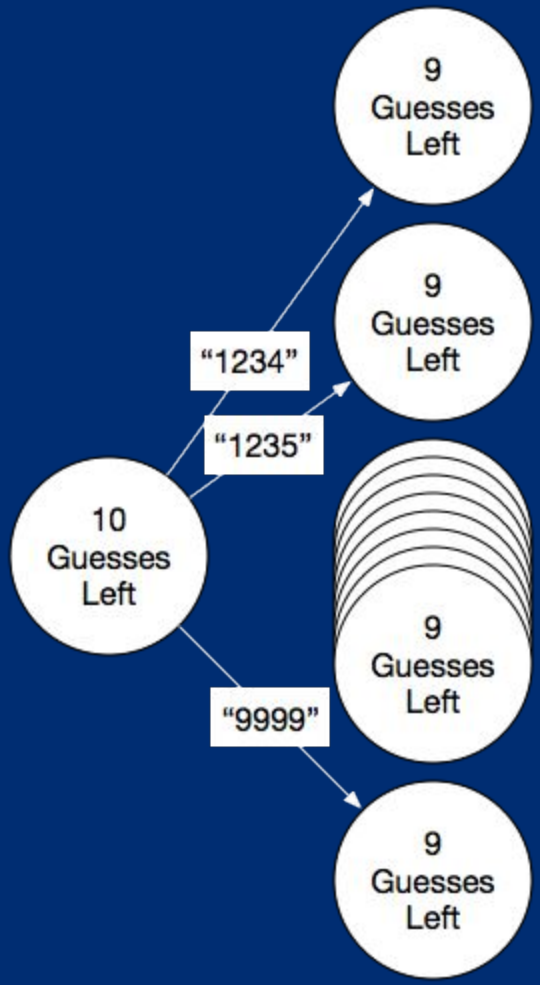
- Hardware based TEE's require NVRAM for protection
  - Scale poorly, expensive, and require special considerations for power fluctuations
  - Prior Work: Memoir [PLDMM11]

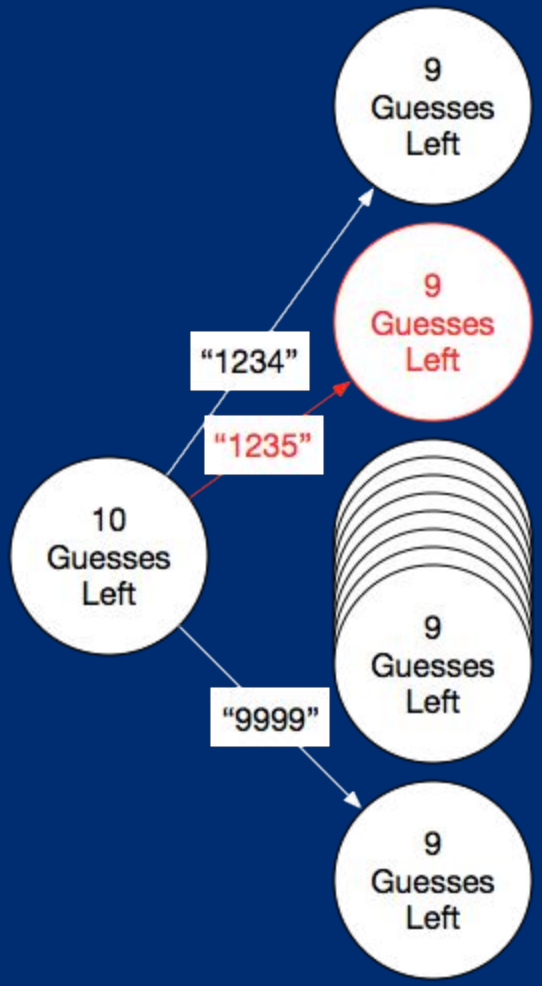
- Software only obfuscation can't get hardware-back protections
  - Prior Work: Goyal and Goyal [GoyGoy17] get one time programs from Ledgers + Obfuscation

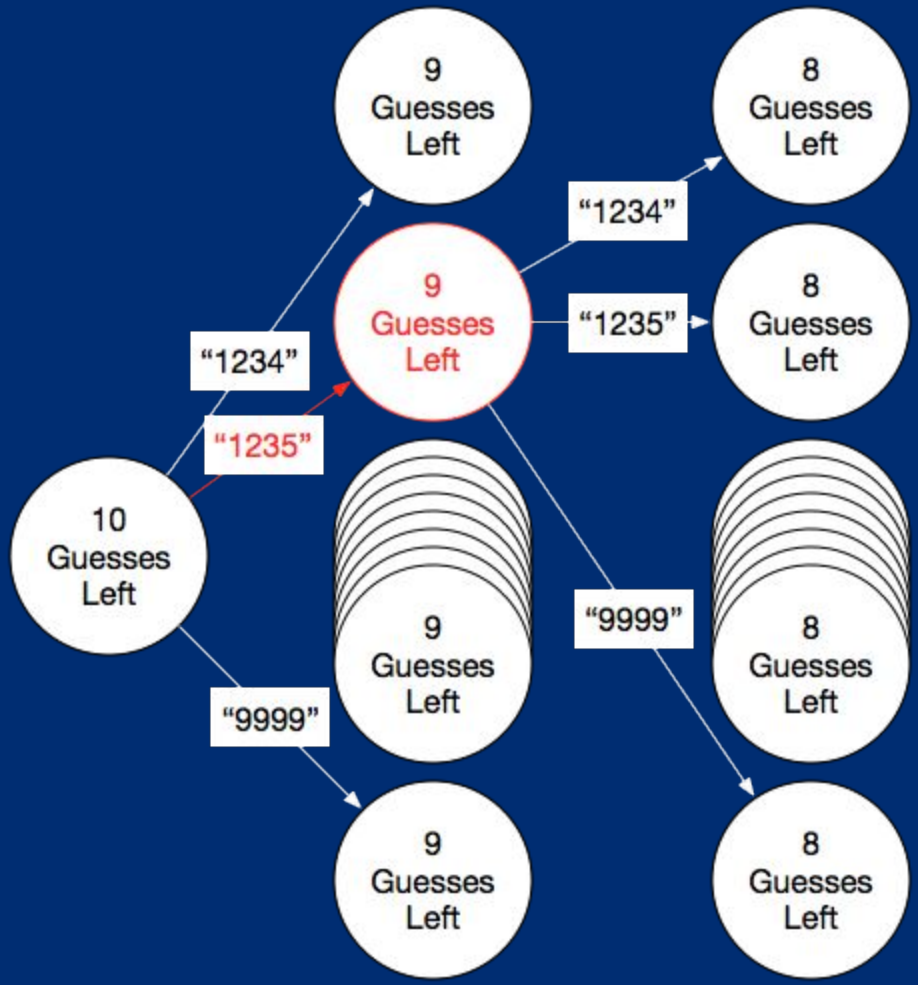
- This problem is *real*

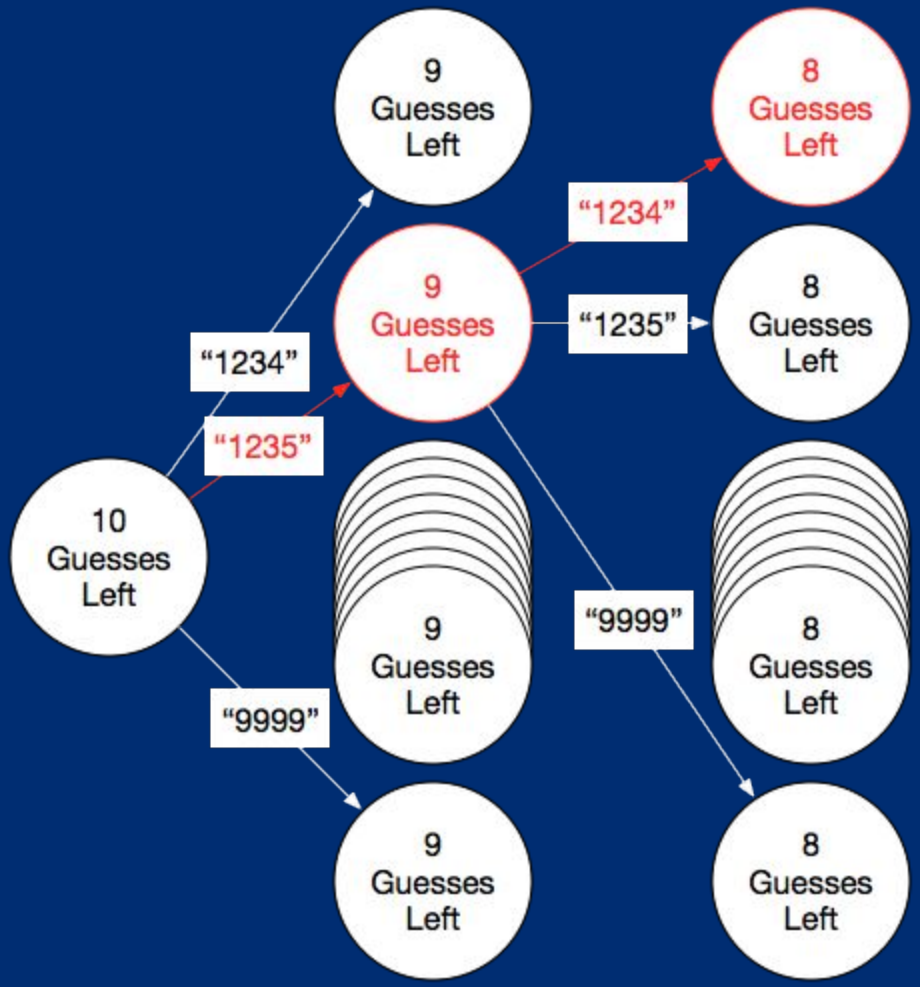
JOHNS HOPKINS UNIVERSITY

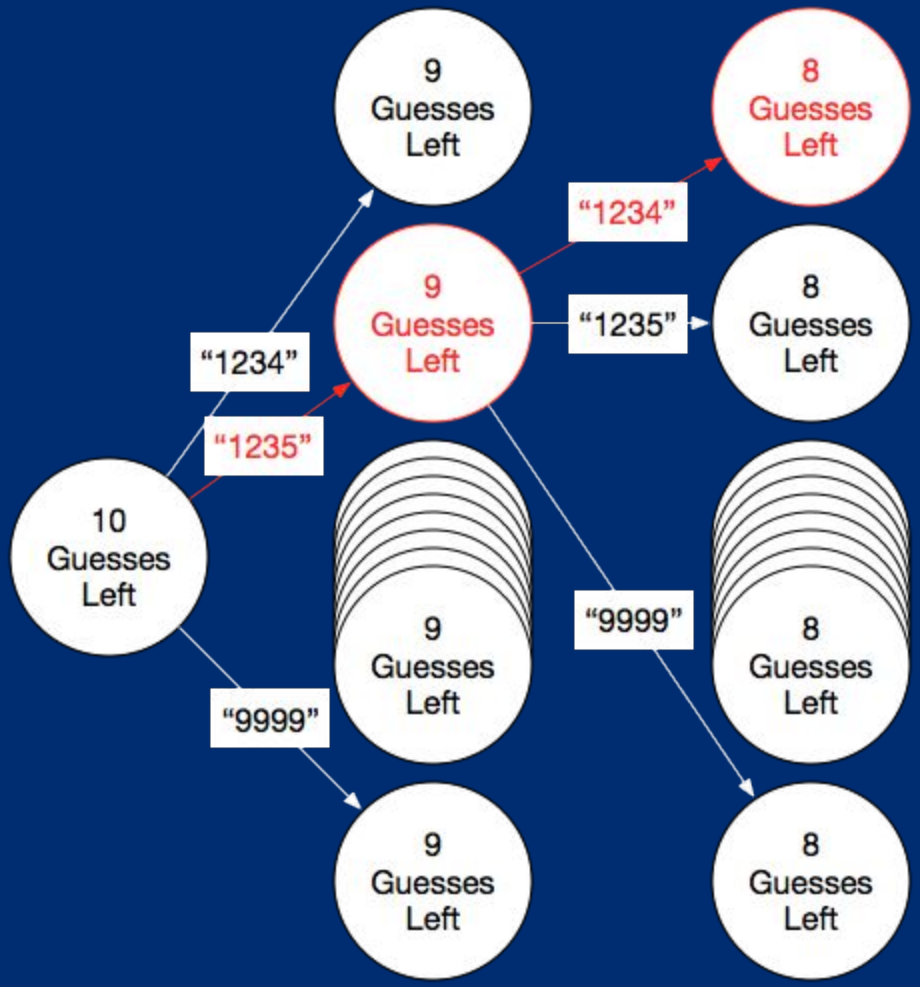
10
Guesses
Left





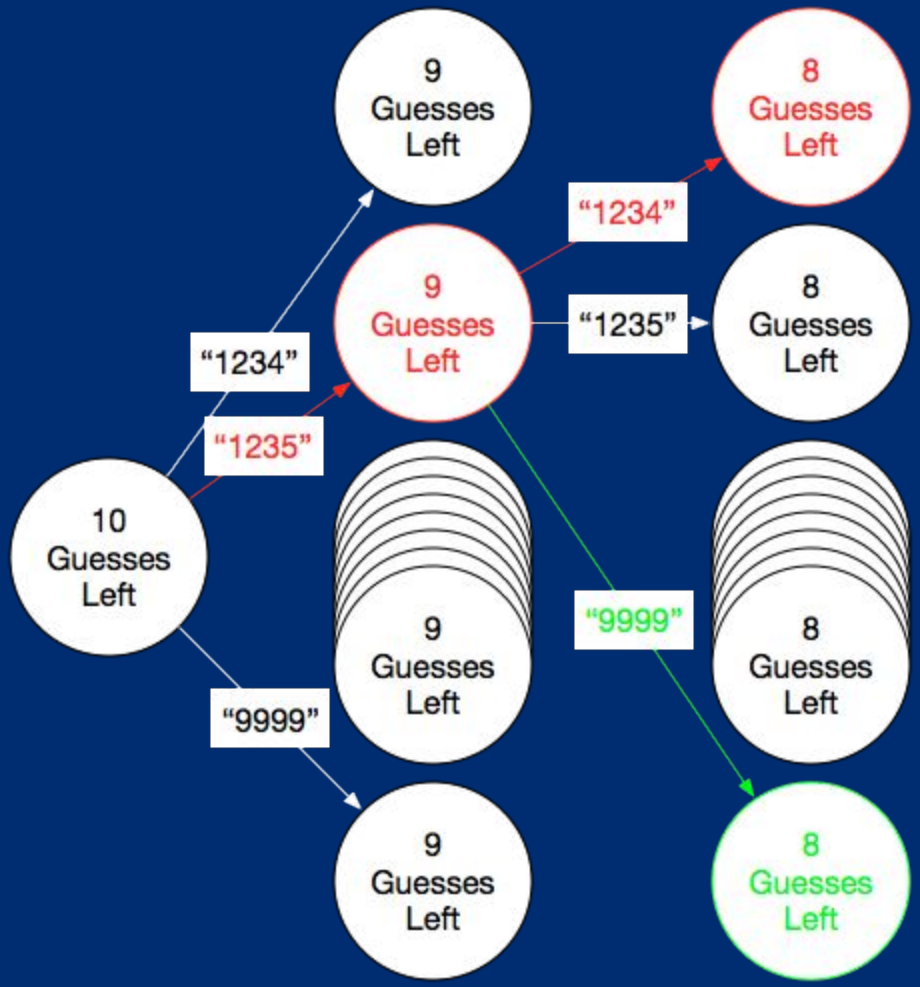






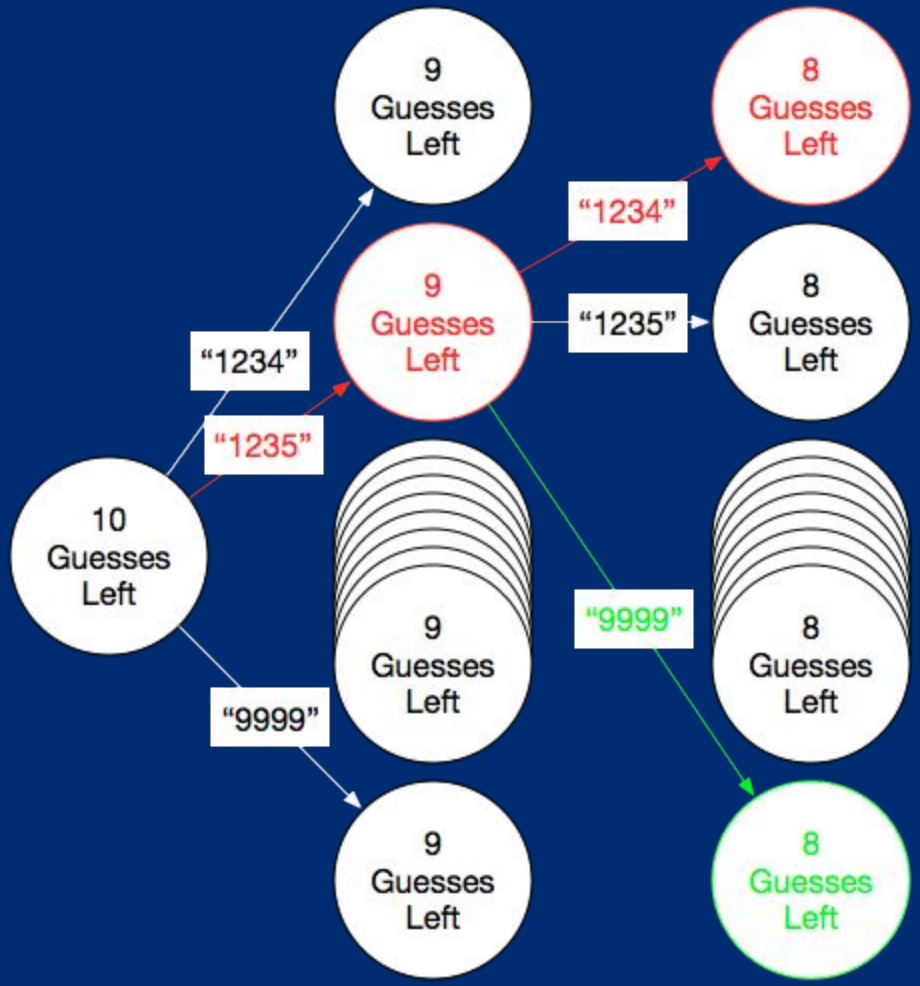
Repeated Execution

- Re-execution of a path doesn't cause a vulnerability
  - Derive the same key repeatedly
  - Starting again generates new master key



Repeated Execution

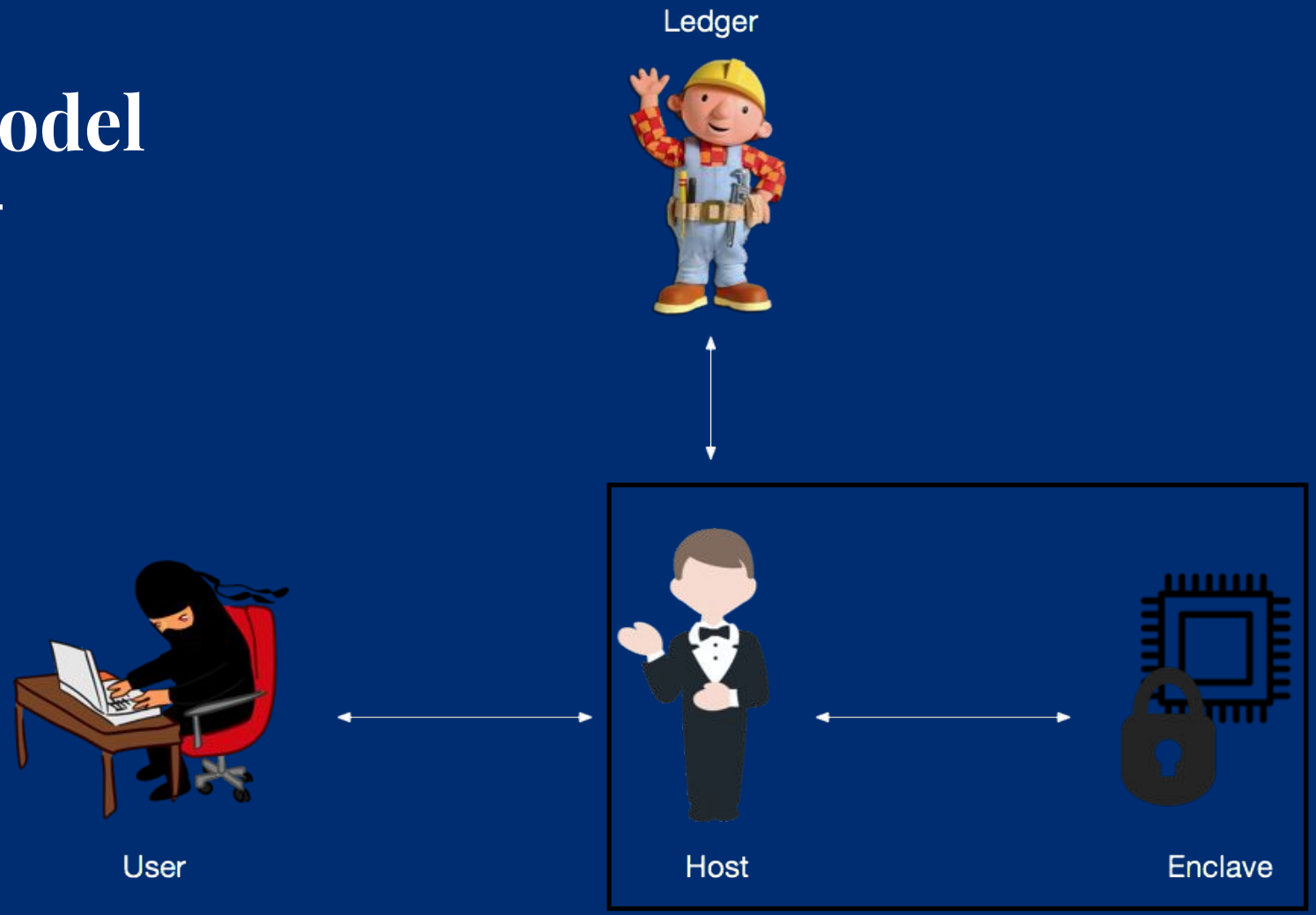
- Re-execution of a path doesn't cause a vulnerability
  - Derive the same key repeatedly
  - Starting again generates new master key
- Forking is dangerous
  - Running new inputs on old state
  - Running old steps with new randomness



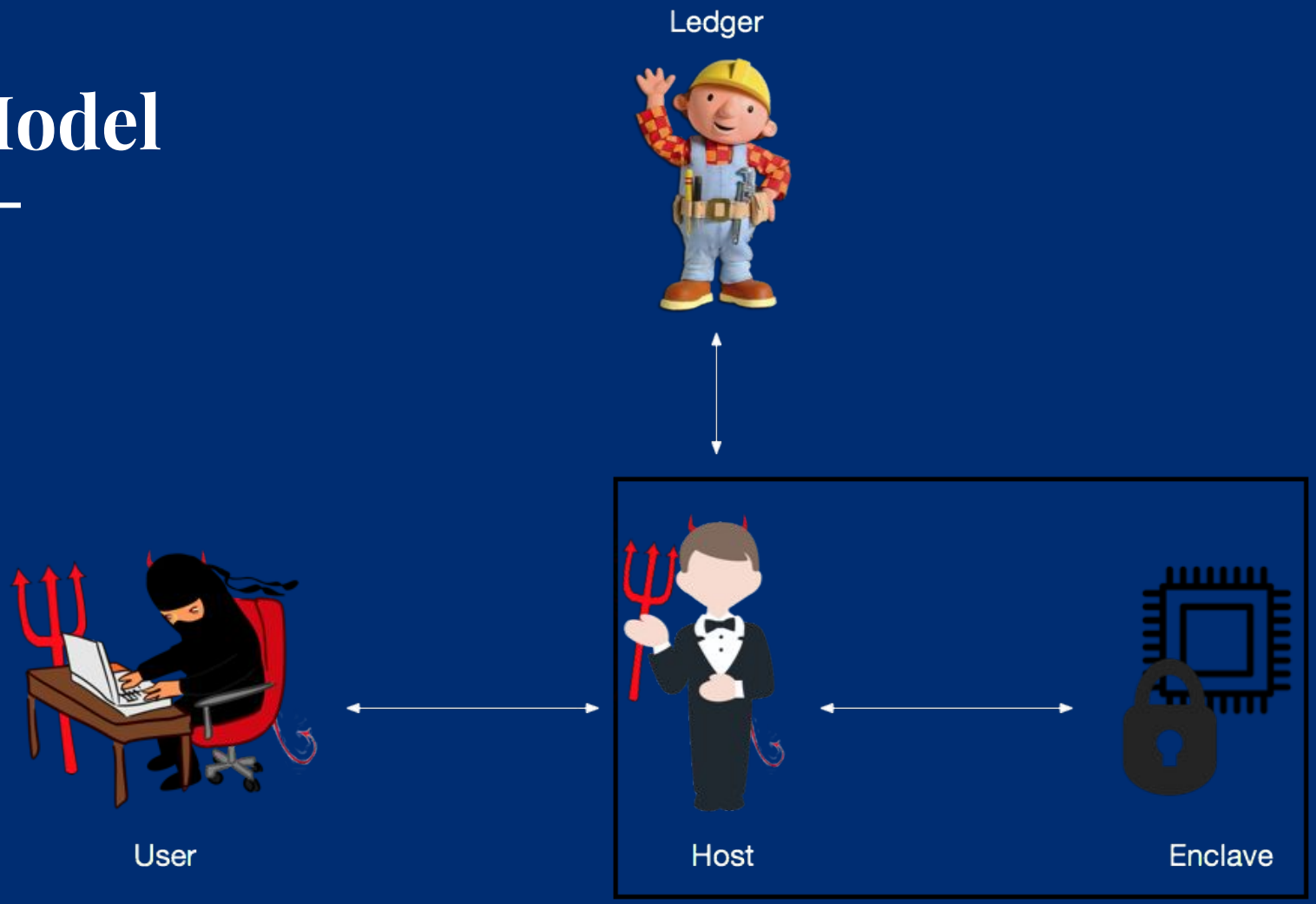
Repeated Execution

- Re-execution of a path doesn't cause a vulnerability
  - Derive the same key repeatedly
  - Starting again generates new master key
- Forking is dangerous
  - Running new inputs on old state
  - Running old steps with new randomness
- Strategy: bind program execution to something linear

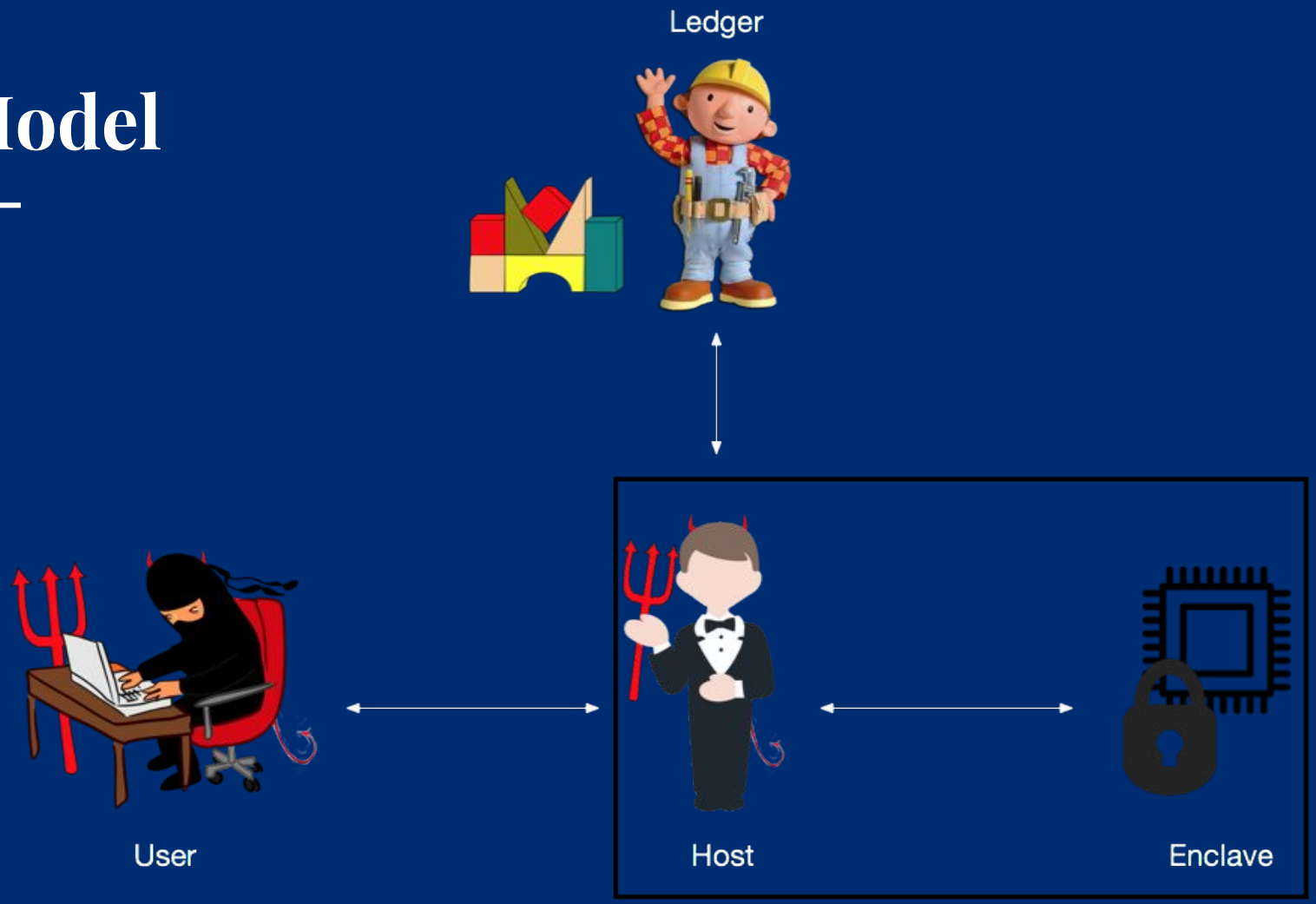
Model



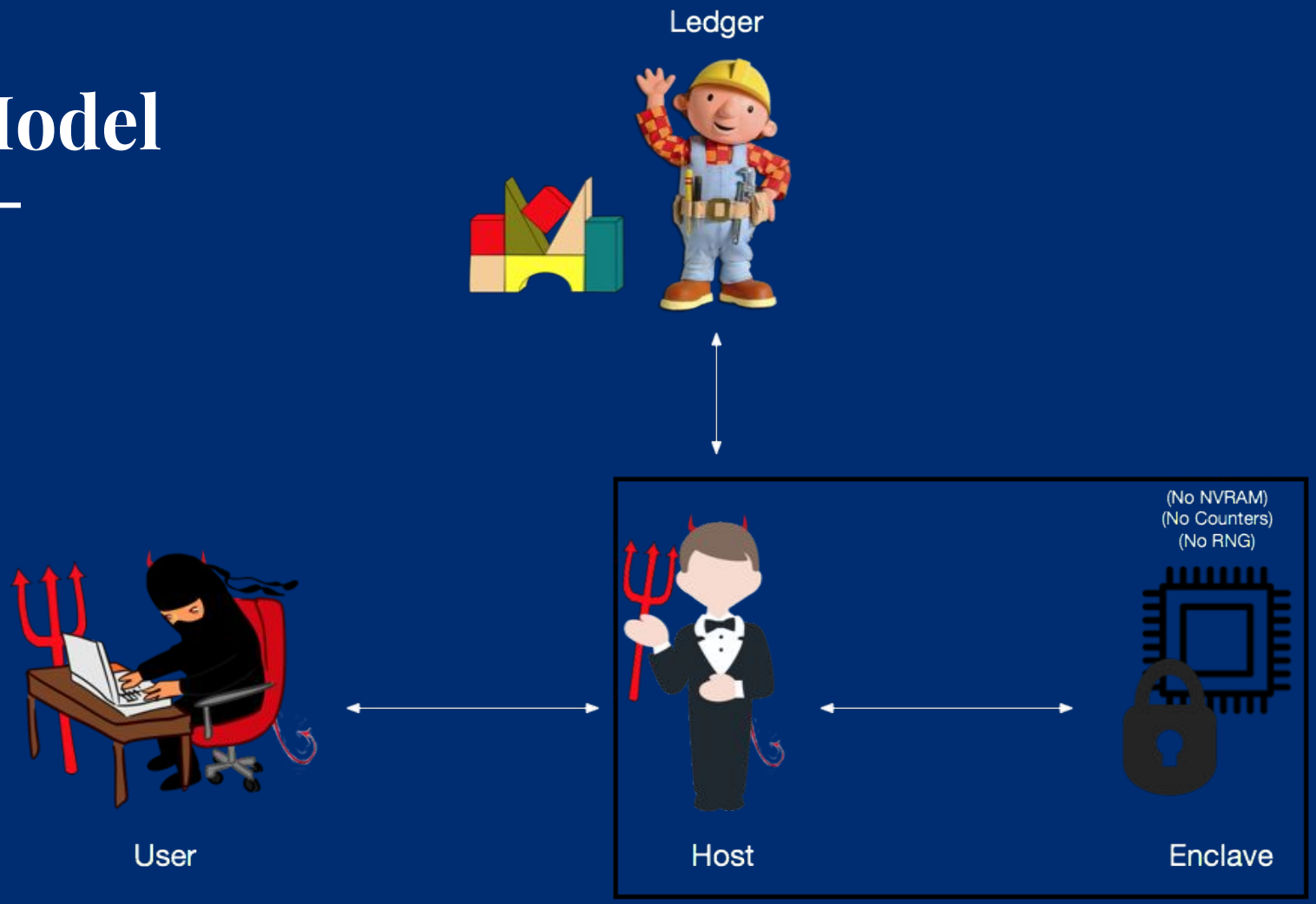
Model



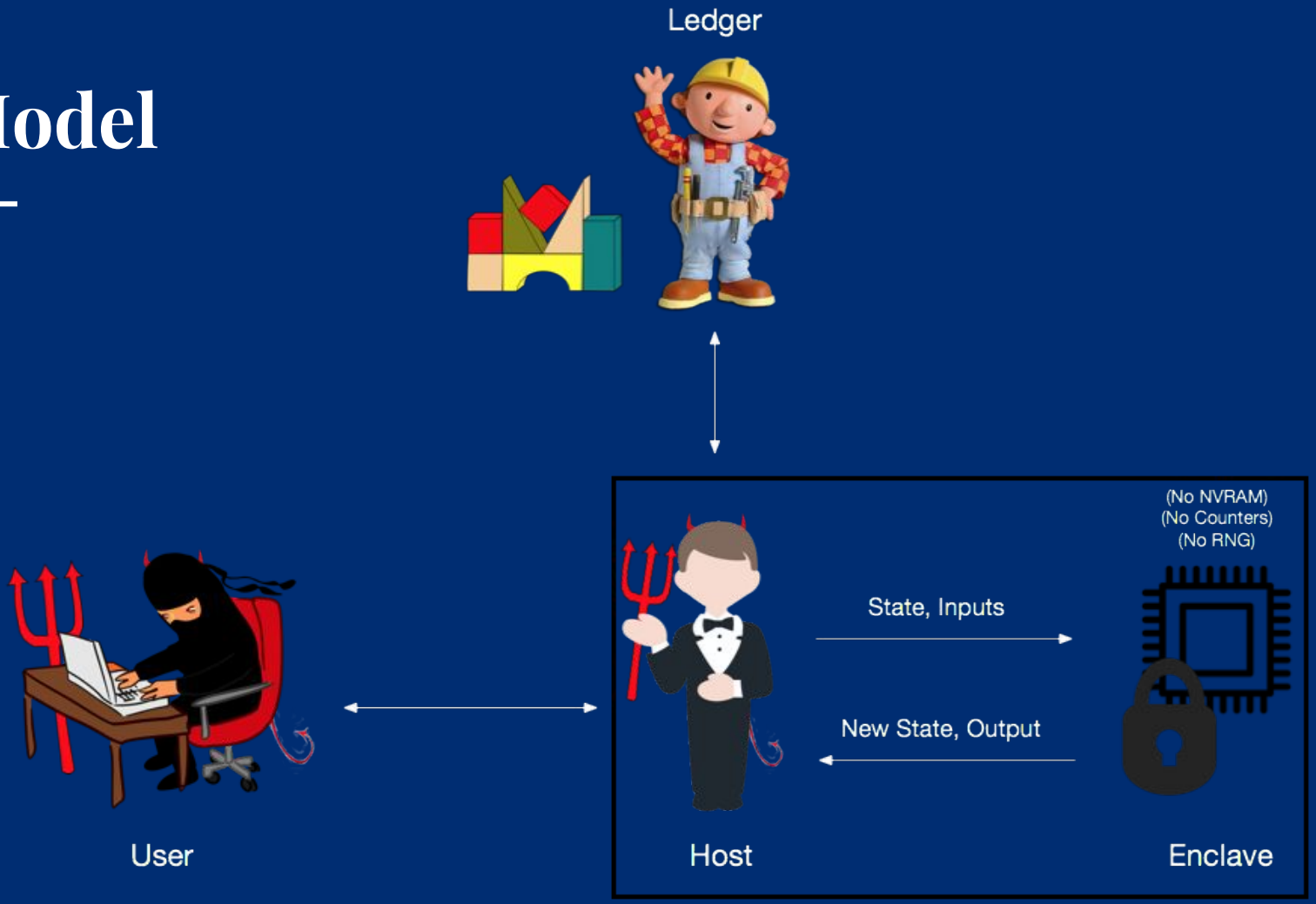
Model



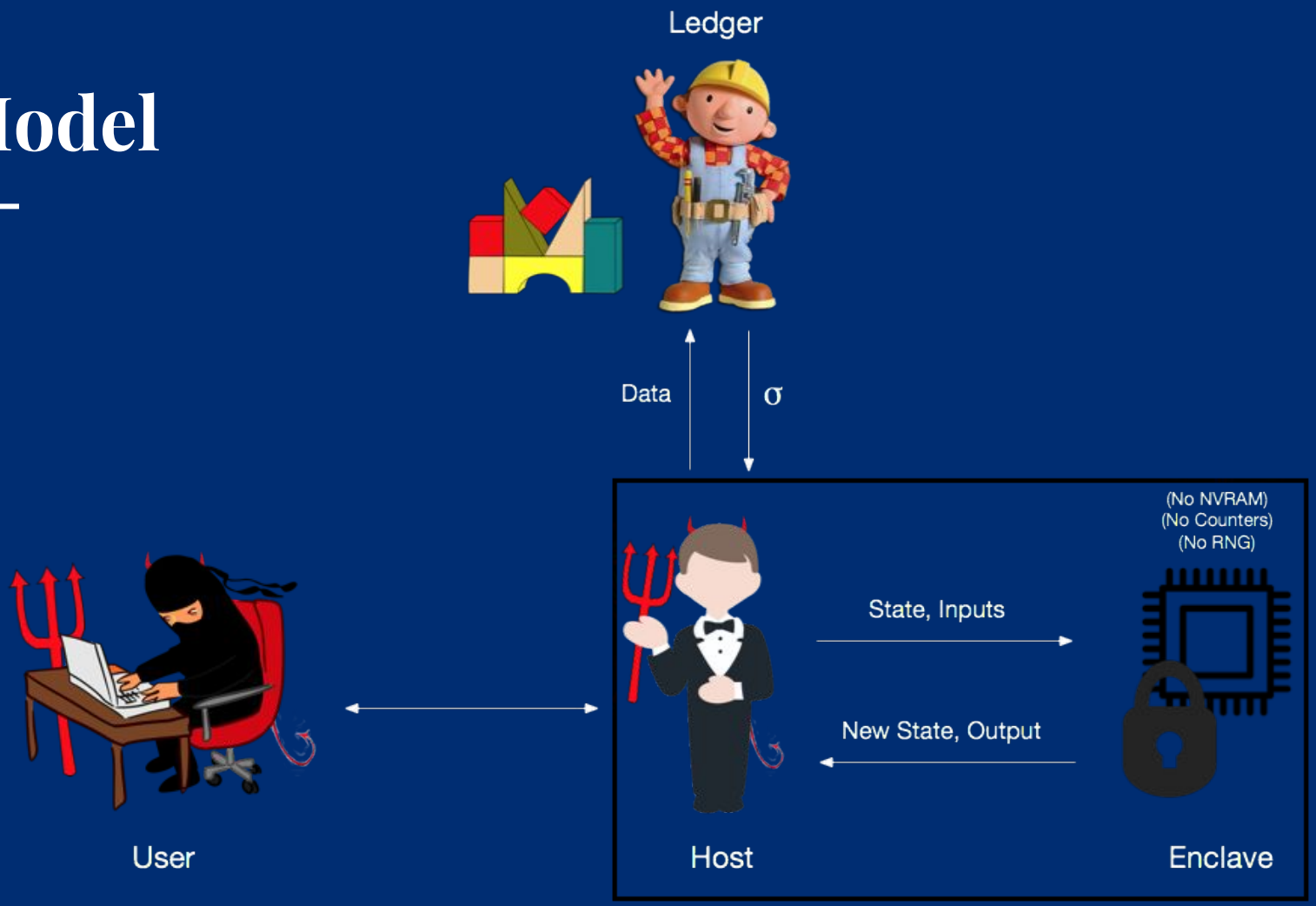
Model



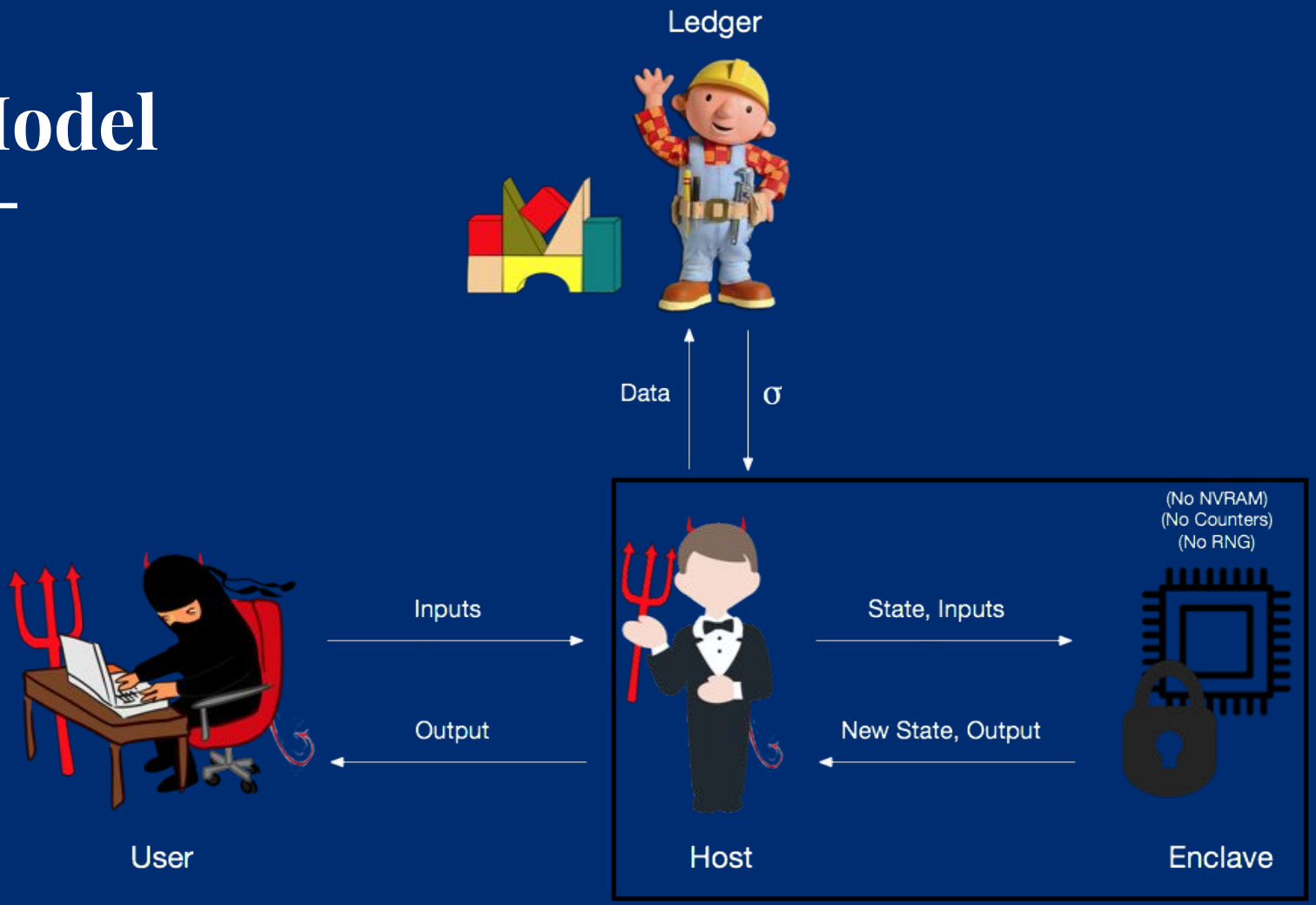
Model


Ledger
(No NVRAM)
(No Counters)
(No RNG)
State, Inputs
New State, Output
User
Host
Enclave

Model

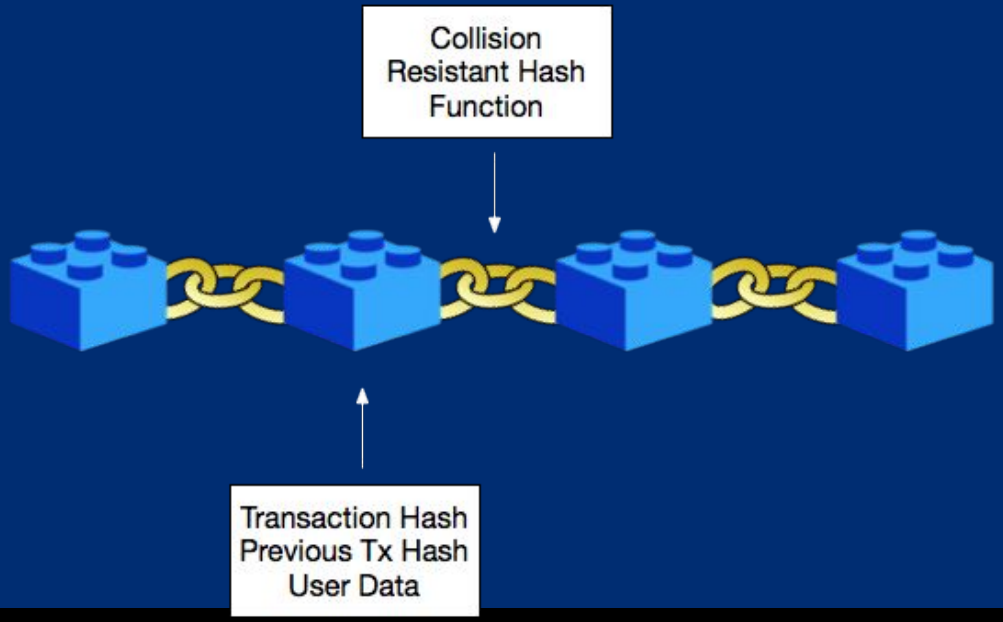

Ledger
Data
σ
(No NVRAM)
(No Counters)
(No RNG)
State, Inputs
New State, Output
User
Host
Enclave

Model


Ledger
Data
σ
(No NVRAM)
(No Counters)
(No RNG)
Inputs
State, Inputs
Output
New State, Output
User
Host
Enclave

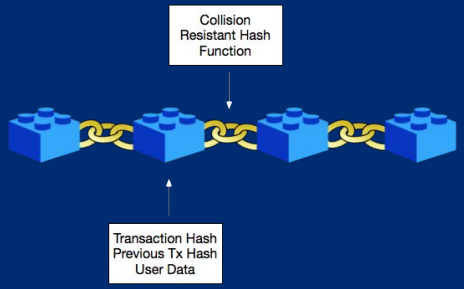
Ledger Requirements

- Creates hash chains of transactions
  - Similar to transaction in bitcoin, ethereum, etc...



Ledger Requirements

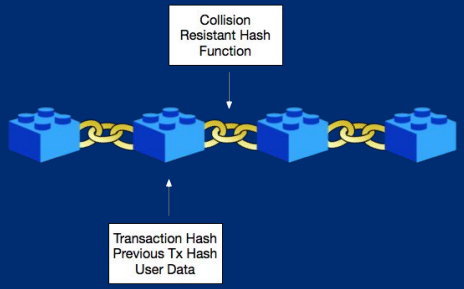
- Creates hash chains of transactions
  - Similar to transaction in bitcoin, ethereum, etc…



- Publicly verifiable proof of publication and public access
  - Digital signatures for computational security
  - Proof of work for economic security

Ledger Requirements

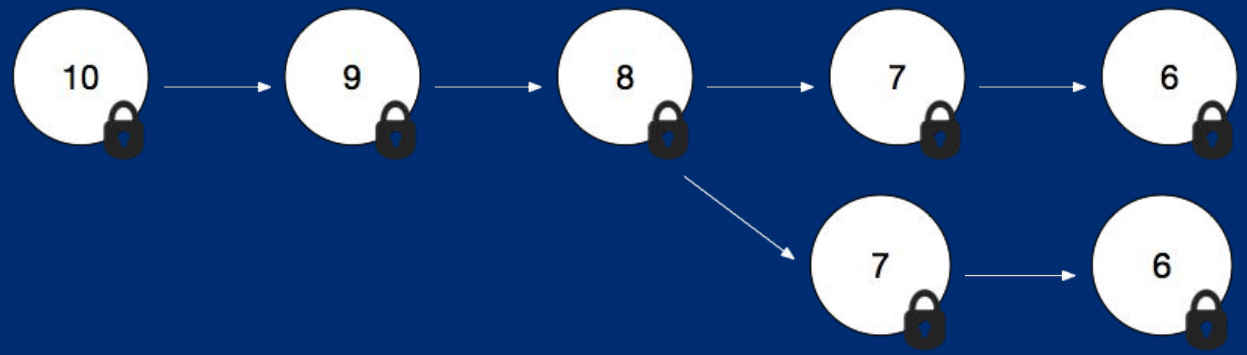
- Creates hash chains of transactions
  - Similar to transaction in bitcoin, ethereum, etc…



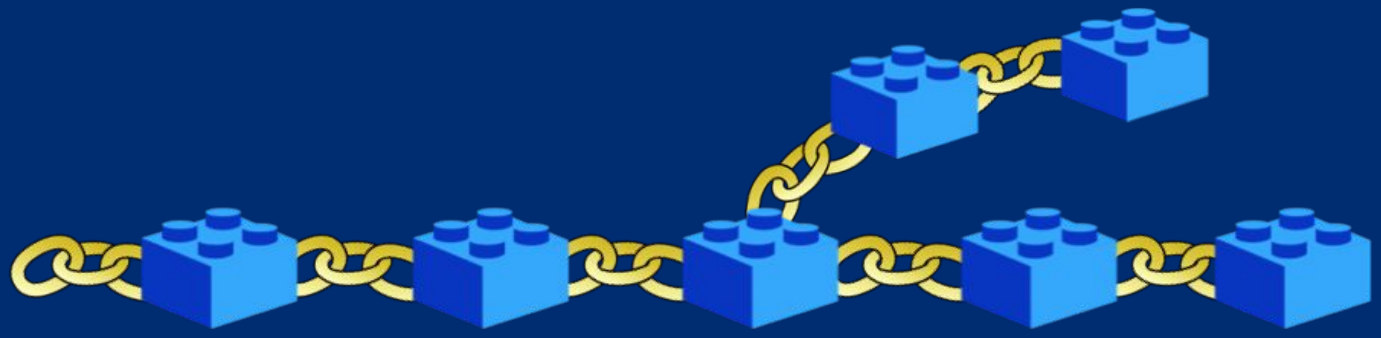
- Publicly verifiable proof of publication and public access
  - Digital signatures for computational security
  - Proof of work for economic security

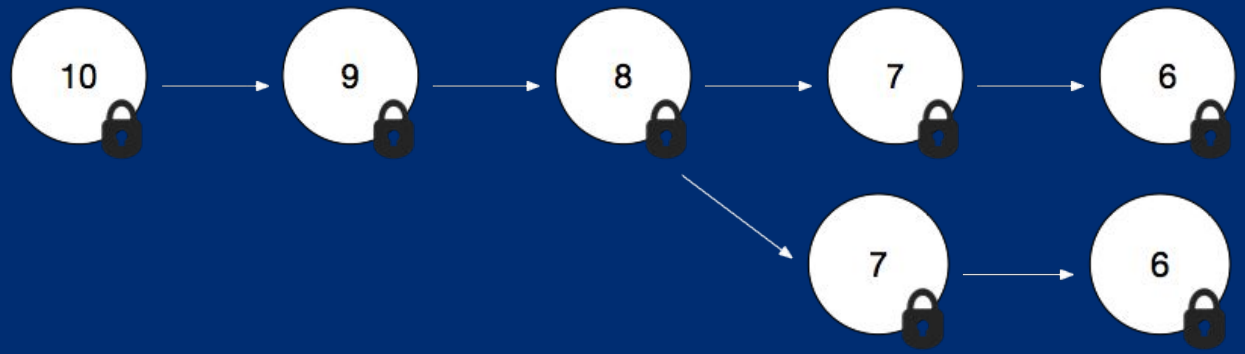
- Simplifying assumption: Single user ledgers

JOHNS HOPKINS
UNIVERSITY

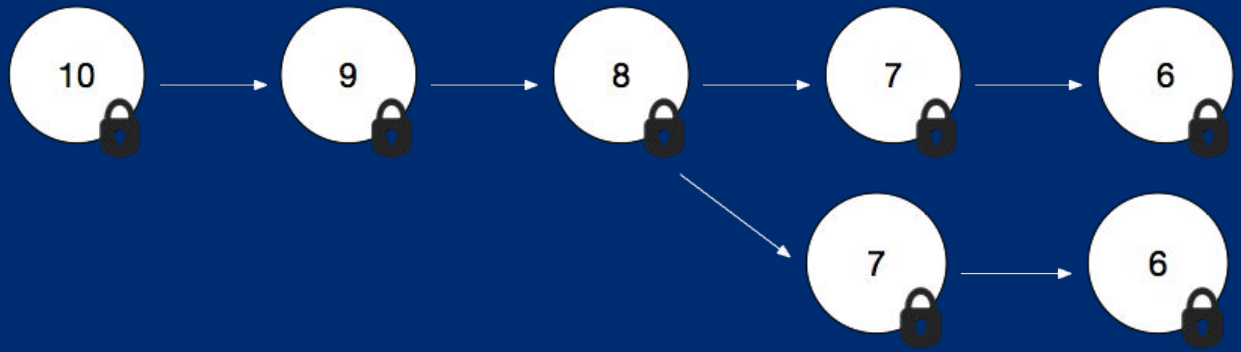


JOHNS HOPKINS
UNIVERSITY

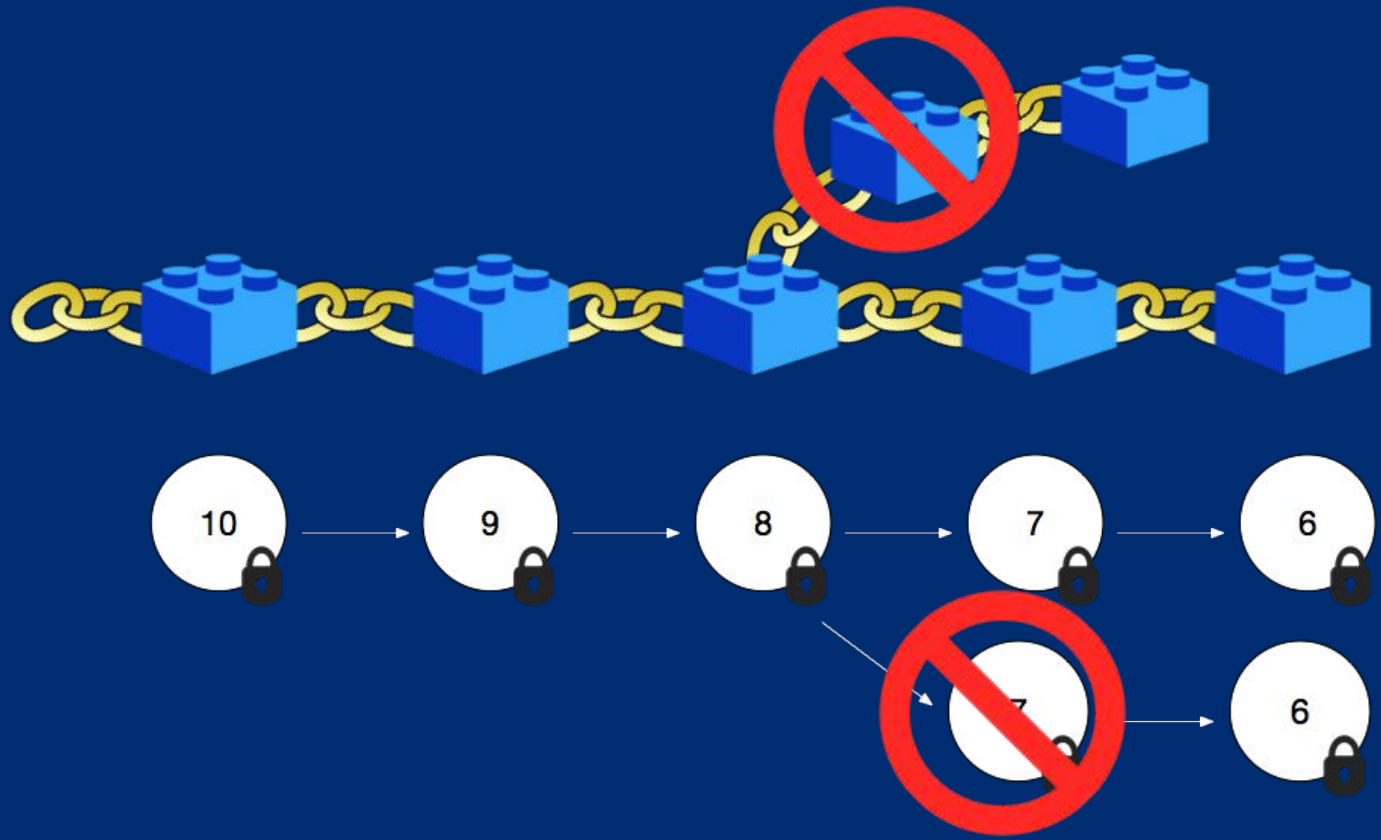




JOHNS HOPKINS
UNIVERSITY



JOHNS HOPKINS
UNIVERSITY



JOHNS HOPKINS
UNIVERSITY

Ledger

User

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

User

"1234"

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

C = Com(10 , “1234”; r)

User

“1234”

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

C = Com(10 , “1234”; r)

User

“1234”

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

C = Com(10 , "1234"; r)

, σ

User

"1234"

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

C = Com(10 , "1234"; r)

, σ

User

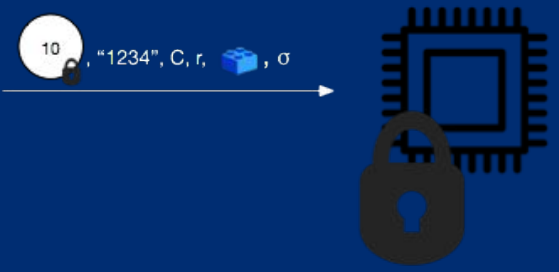
"1234"

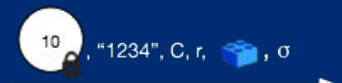
Host

10 , "1234", C, r, , σ

Enclave

JOHNS HOPKINS
UNIVERSITY


10
, “1234”, C, r,
, σ



Verify(, σ)

Verify(C = Com(10 , “1234”; r))

Derive encryption key for next state

Transaction Hash
Previous Tx Hash
C

Derive decryption key for previous state

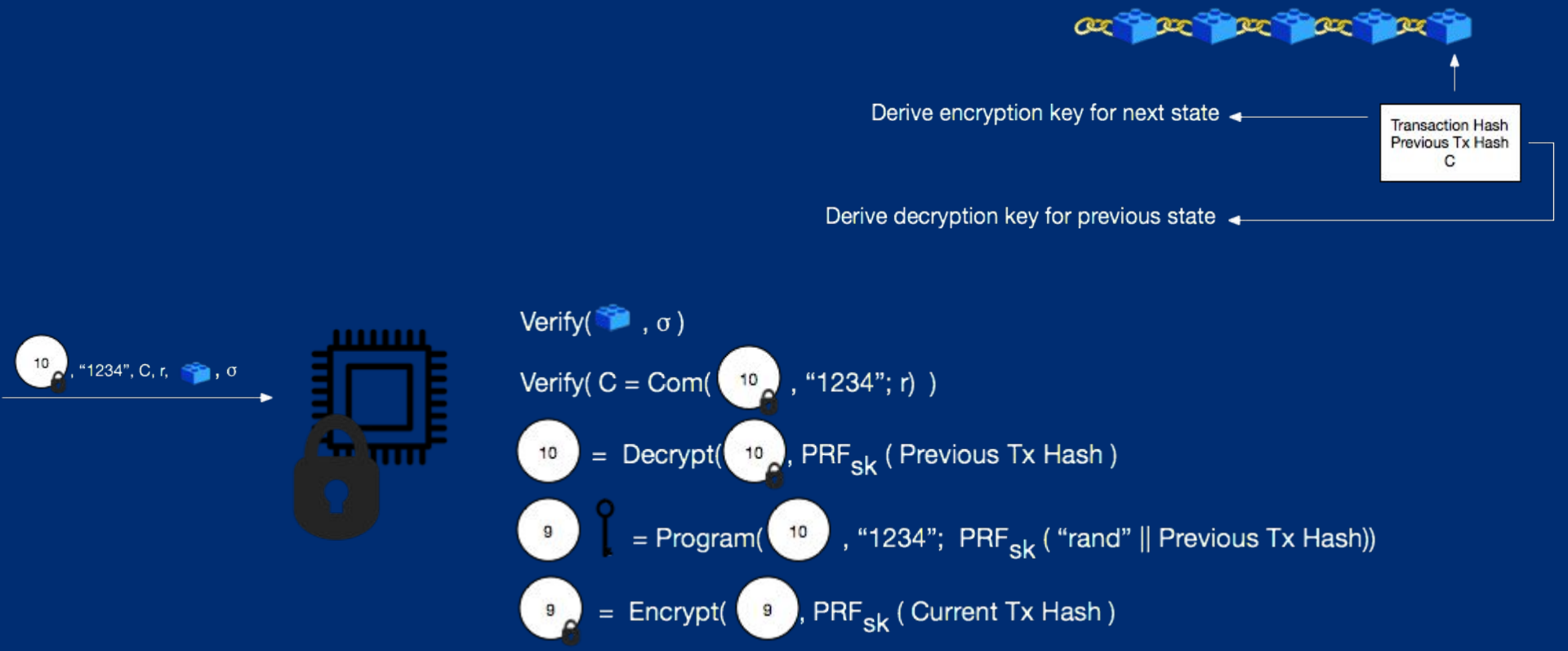
10 , "1234", C, r, , σ

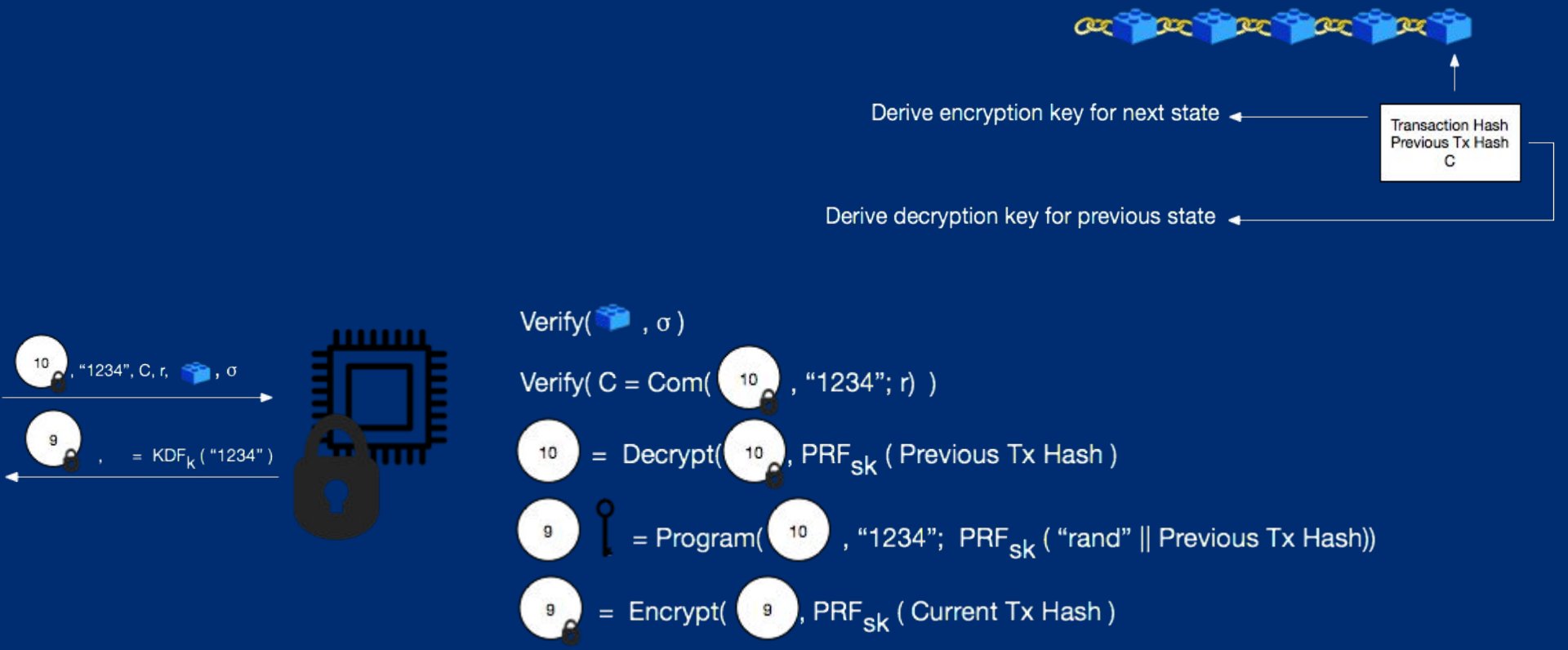
Verify(, σ)

Verify(C = Com(10 , "1234"; r))

10 = Decrypt(10 , PRF_{sk} (Previous Tx Hash)


Derive encryption key for next state
Transaction Hash
Previous Tx Hash
C
Derive decryption key for previous state
10 , "1234", C, r, , σ
Verify(, σ)
Verify(C = Com(10 , "1234"; r))
10 = Decrypt(10 , PRF$_{sk}$ (Previous Tx Hash)
9 = Program(10 , "1234"; PRF$_{sk}$ ("rand" || Previous Tx Hash))




Derive encryption key for next state
Transaction Hash
Previous Tx Hash
C
Derive decryption key for previous state
10 , "1234", C, r, , σ
9 , = KDF_k ("1234")
Verify(, σ)
Verify(C = Com(10 , "1234"; r))
10 = Decrypt(10 , PRF_sk (Previous Tx Hash)
9 = Program(10 , "1234"; PRF_sk ("rand" || Previous Tx Hash))
9 = Encrypt(9 , PRF_sk (Current Tx Hash)

Ledger

C = Com(10 , "1234"; r)

, σ

"1234"

10 , "1234", C, r, , σ

9 , = KDF_k ("1234")

User

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Ledger

C = Com(10 , "1234"; r)

, σ

"1234"

Decryption Failure, 9 more

User

10 , "1234", C, r, , σ

9 , = KDF_k ("1234")

Host

Enclave

JOHNS HOPKINS
UNIVERSITY

Protocol Extensions

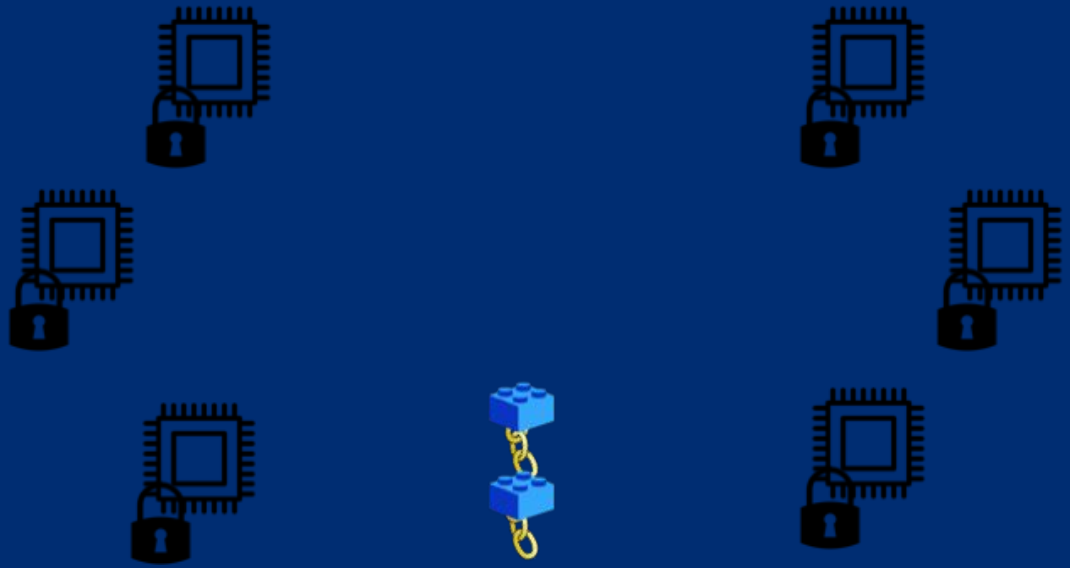
- We have managed to condition execution on ledger postings

- Extension #1: Programs can require public posting
  - *E.g.* Error reporting, guaranteed logging

- Extension #2: One Time Programs
  - Swept under the rug: so far we have secure *multi-execution programs*
  - Derive *unique* valid hash chain from program code

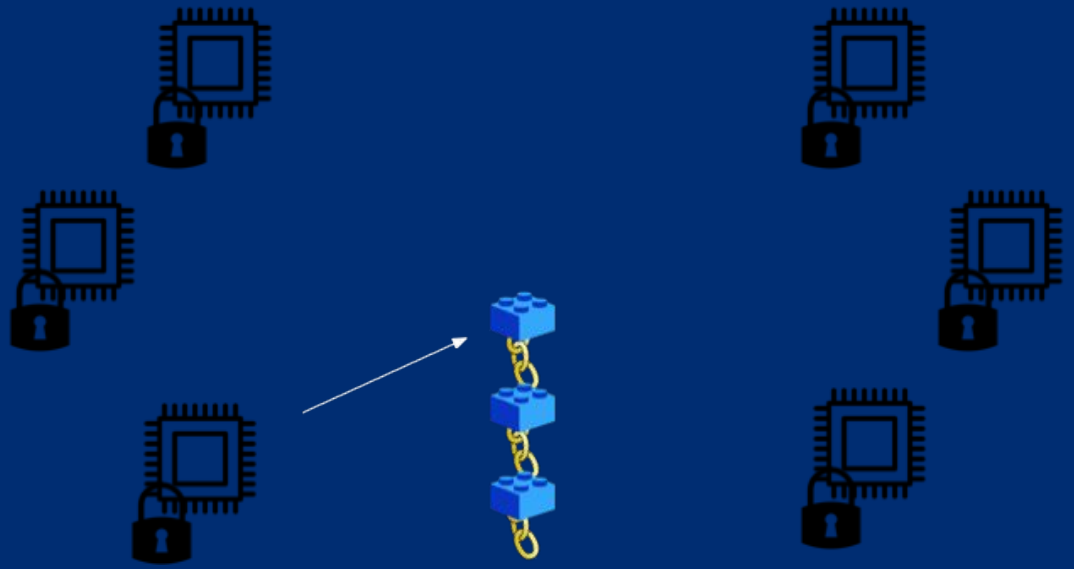
Additional Applications

- Smart contracts computing on private data
  - Concurrent work with Intel's Private Data Objects
  - Later follow-up work in the same area [Eikiden]



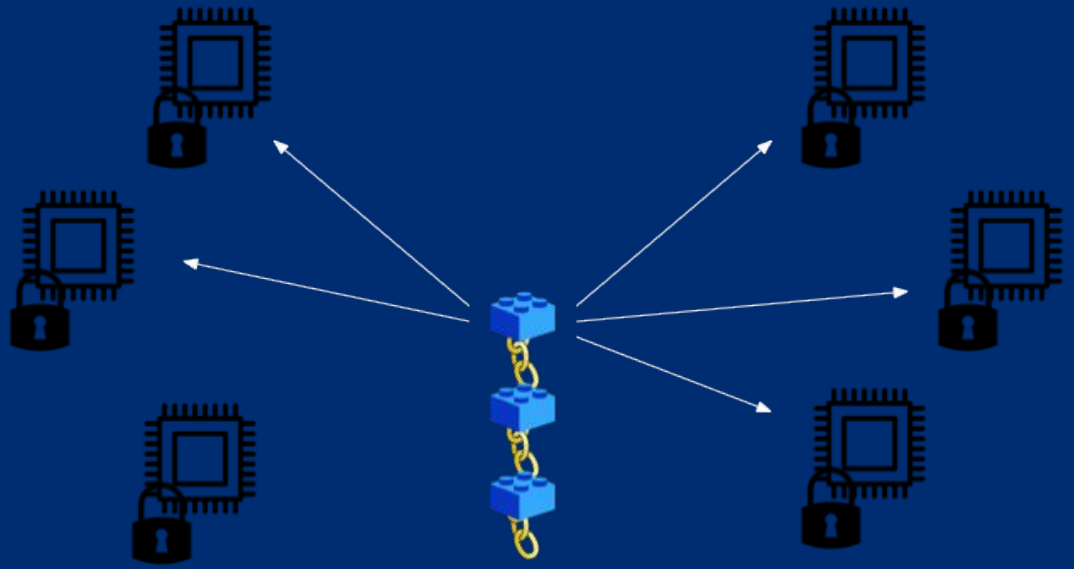
Additional Applications

- Smart contracts computing on private data
  - Concurrent work with Intel's Private Data Objects
  - Later follow-up work in the same area [Eikiden]



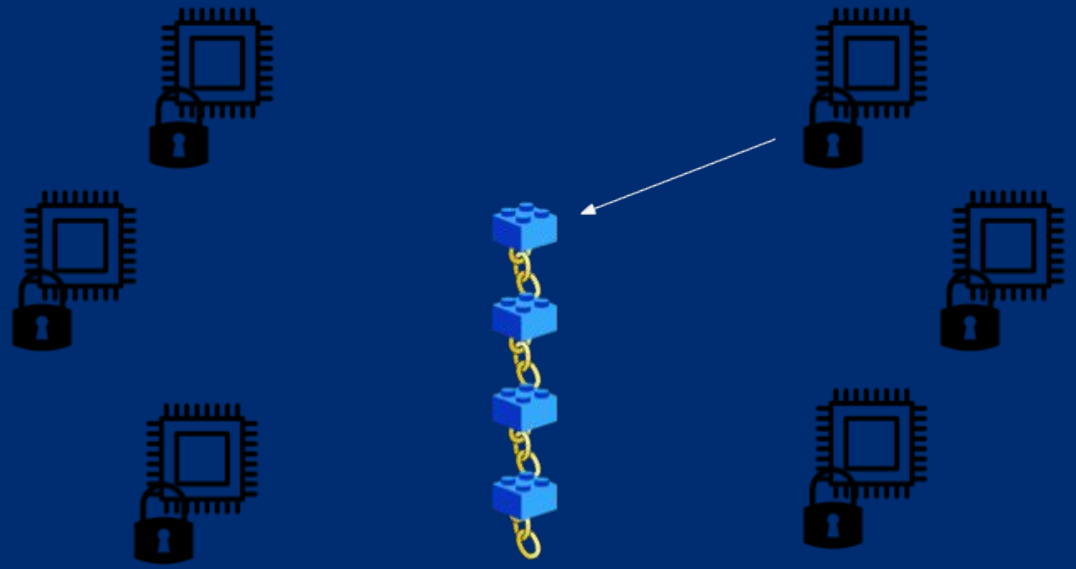
Additional Applications

- Smart contracts computing on private data
  - Concurrent work with Intel's Private Data Objects
  - Later follow-up work in the same area [Eikiden]



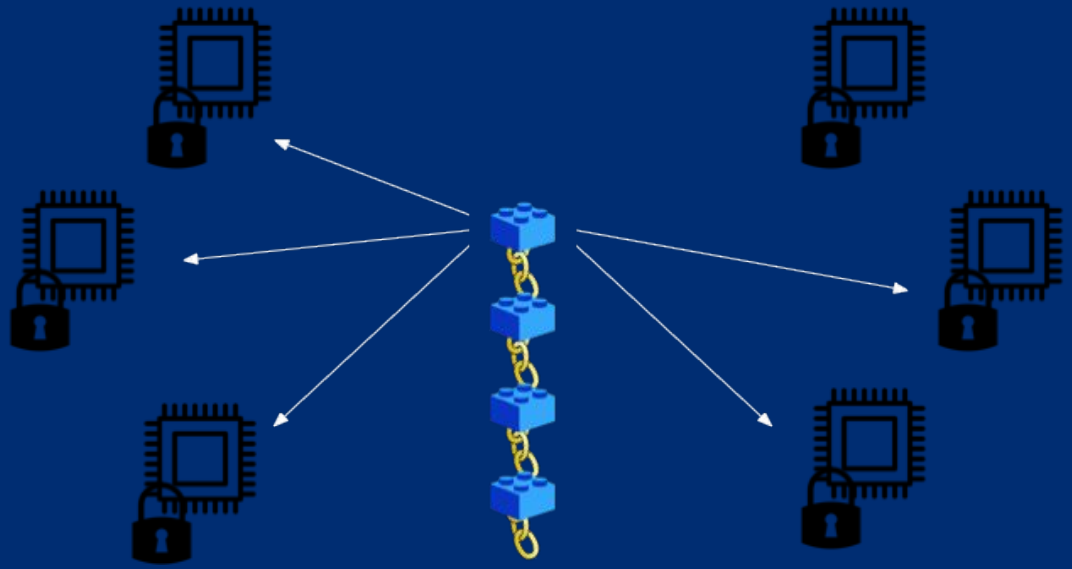
Additional Applications

- Smart contracts computing on private data
  - Concurrent work with Intel's Private Data Objects
  - Later follow-up work in the same area [Eikiden]



Additional Applications

- Smart contracts computing on private data
  - Concurrent work with Intel's Private Data Objects
  - Later work in the same area [Ekiden]



Additional Applications

- Autonomous Ransomware
  - Inevitable outcome of malicious trusted execution environments
  - Eliminates the need for command and control systems

Conclusions

- We create a novel protocol that provides trustworthy state for TEE's by binding state to an append-only ledger
- Ledgers are here to stay — lets do more than just currency-related research
- Keeping state is a difficult problem with wide ranging applications

Thank You!

Gabriel Kaptchuk
kaptchuk.com
gabe@kaptchuk.com

JOHNS HOPKINS
UNIVERSITY

Bonus Slides