Giving State to the Stateless:
Augmenting Trustworthy Computation with Ledgers

Gabriel Kaptchuk, Matthew Green, and Ian Miers
BEWARE!
HERE BE BLOCKCHAINS!
BEWARE!
HERE BE BLOCKCHAINS!
Beware!
Here be SGX!
Now that we have SGX, all security problems are trivial. Security research is officially over!
BEWARE!
HERE BE SGX!

SPECTRE

Foreshadow

MELTDOWN
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

Is the password “1234”?

Decryption Failure! 9 more attempts!

$\text{Is the password “1234”?}$

$\text{9 Guesses}$

$\text{9 Guesses}$

$\text{KDF}_k (“1234”) \Rightarrow \text{Secret Key}$
Rewind Protection

Is the password “1234” or “1235”?  

Decryption Failure! 9 more attempts!

Is the password “1234”?  

Is the password “1235”?

$KDF_K(“1234”) = 10$ Guesses

$KDF_K(“1235”) = 10$ Guesses

$KDF_K(“1234”) = 9$ Guesses

$KDF_K(“1235”) = 9$ Guesses
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
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

Ledger

User

Host

Enclave
Model

Ledger

User

Host

Enclave
Model
Model

Ledger

User

Host

Enclave

(No NVRAM)
(No Counters)
(No RNG)
Model
Model

Ledger

Data \( \sigma \)

User

Input \( \rightarrow \) Output

Host

State, Inputs \( \rightarrow \) New State, Output

Enclave

(No NVRAM)
(No Counters)
(No RNG)
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
Transaction Hash
Previous Tx Hash
C

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

User

Host

Enclave
Verify(\text{C}, \sigma )

Verify( C = \text{Com}(\text{10}, "1234"; r) )
Verify( $C$, $\sigma$ )
Verify( $C = \text{Com}(100, "1234"; r)$ )
Verify(\( C \), \( \sigma \))
Verify(\( C = \text{Com}(10, "1234"; r) \))
\( 10 = \text{Decrypt}(10, \text{PRF}_{sk}(\text{Previous Tx Hash})) \)

Derive encryption key for next state

Derive decryption key for previous state

Transaction Hash
Previous Tx Hash
\( C \)
Verify(10, 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

Derive decryption key for previous state

\[
\text{Verify}(\Box, \sigma) \\
\text{Verify}(C = \text{Com}(10, "1234"; r)) \\
\begin{align*}
10 &= \text{Decrypt}(10, \text{PRF}_{sk}(\text{Previous Tx Hash})) \\
9 &= \text{Program}(10, "1234"; \text{PRF}_{sk}(\text{"rand" || Previous Tx Hash})) \\
9 &= \text{Encrypt}(9, \text{PRF}_{sk}(\text{Current Tx Hash}))
\end{align*}
\]
Derive encryption key for next state

Derive decryption key for previous state

\[ \text{Verify}(\text{Com}(10, \text{"1234"}; r) ) \]

\[ 10 = \text{Decrypt}(10, \text{PRF}_{sk} (\text{Previous Tx Hash}) ) \]

\[ 9 = \text{Program}(10, \text{"1234"}; \text{PRF}_{sk} (\text{"rand" || Previous Tx Hash})) \]

\[ 9 = \text{Encrypt}(9, \text{PRF}_{sk} (\text{Current Tx Hash}) ) \]
User

Host

Enclave

Transaction Hash
Previous Tx Hash
C

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

Enclave:

KDF

KDF = KDFk("1234")

10, "1234", C, r, σ

User:

"1234"

Decryption Failure, 9 more

0 Guesses Left
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

Show me a valid cryptocurrency payment to my address and I’ll give you the key!
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 — let’s do more than just currency-related research.

- Keeping state is a difficult problem with wide-ranging applications.
Thank You!
Bonus Slides