SGX and cryptocurrencies

Warren He
Mitar Milutinovic @mitar_m
Dawn Song @dawnsongtweets

Overview

Goal: Improve blockchain technologies using SGX, a hardware trusted computing platform.

- SGX Overview
- Consensus
- Smart Contracts
- Issues
- Summary

Lightning talk: automatic analysis and proof of correctness of smart contracts

SGX Overview

SGX

Key parts:

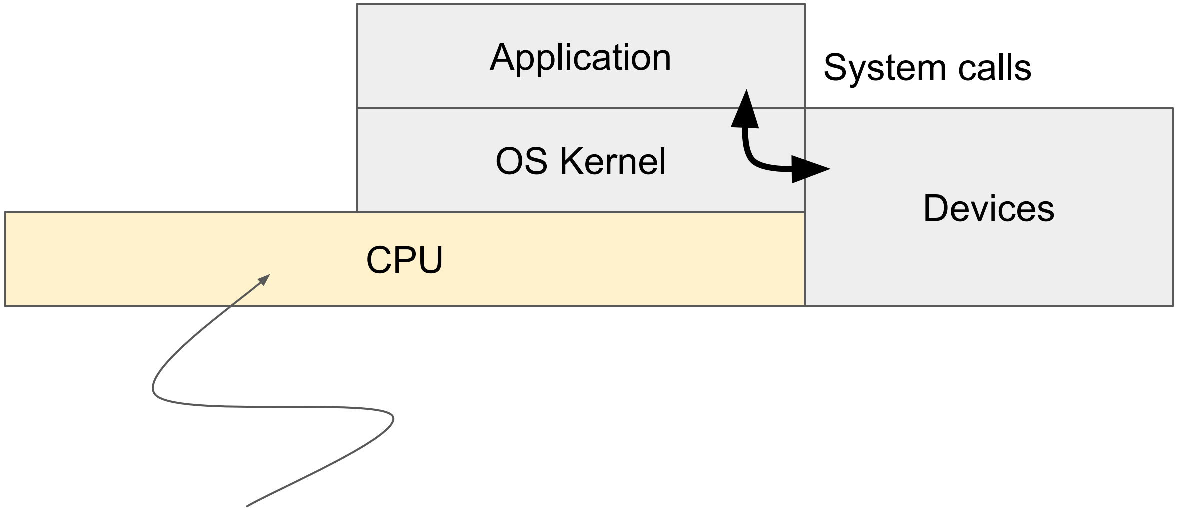
- Isolation
- Attestation
- Platform services

SGX

Key parts:

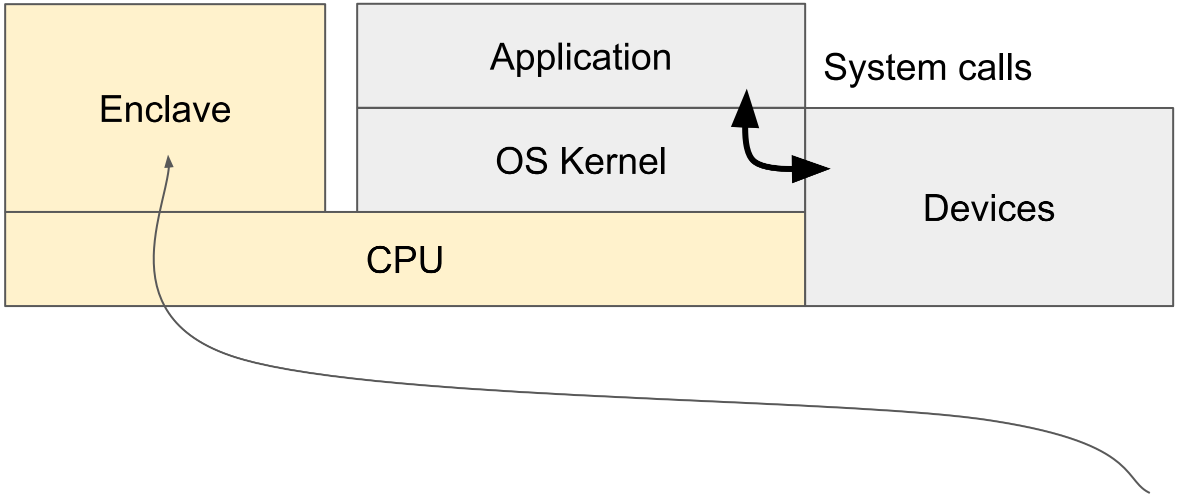
- **Isolation**
- Attestation
- Platform services

Intel SGX (Software Guard Extensions)



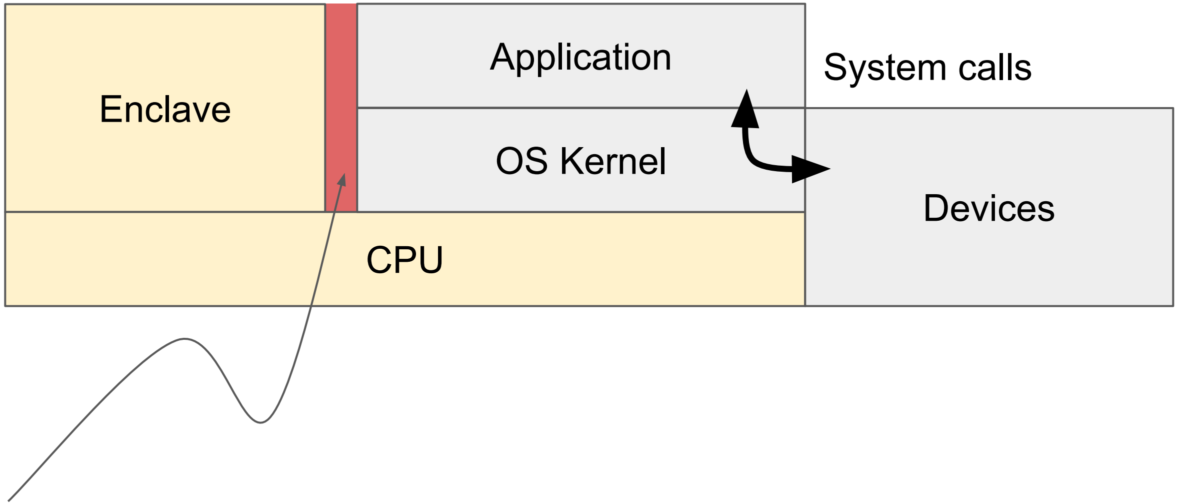
A new set of CPU **instructions** available on Intel Skylake microarchitecture.

Intel SGX (Software Guard Extensions)



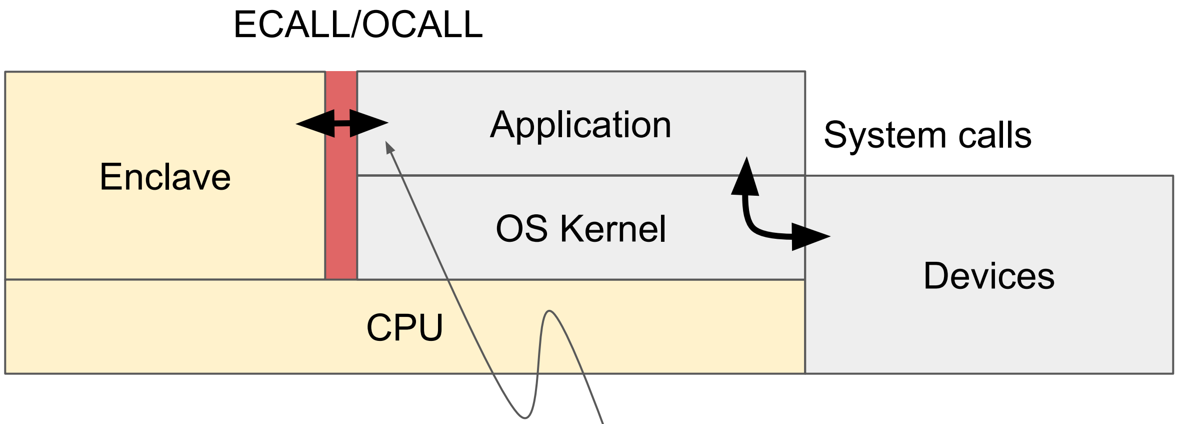
Run code in a hardware-protected container, called an **enclave**.

Intel SGX (Software Guard Extensions)



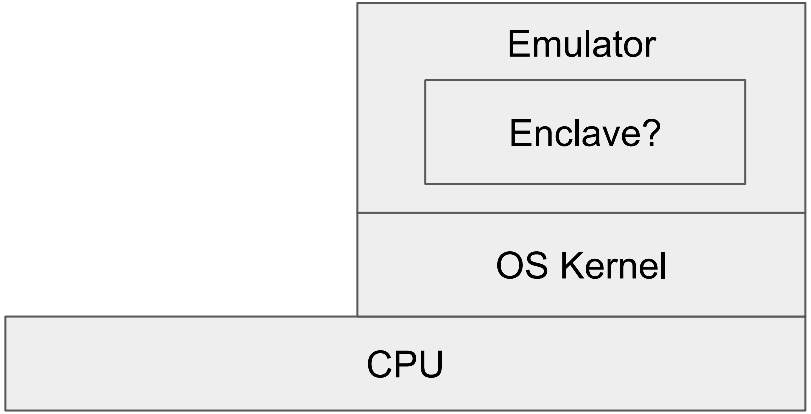
Isolated from other software, even the operating system kernel.

Intel SGX (Software Guard Extensions)



Pure computation, plus the enclave can **talk** to the application that embeds it.

Intel SGX (Software Guard Extensions)



How does the code know that it's really running in an SGX enclave?

SGX

Key parts:

- Isolation
- **Attestation**
- Platform services

SGX Remote Attestation

If the CPU had a key, could it sign something?

The CPU can’t prove anything to the enclave.

But the CPU can prove something to someone else.

SGX Remote Attestation

The attestation protocol proves that a **specific piece of code** ran on **suitable hardware**, producing a **specific result**.

The proof is a signed statement (by the CPU’s key), called a **quote**

You can contact Intel’s server to verify the quote

SGX

Key parts:

- Isolation
- Attestation
- **Platform services**

Intel SGX Platform Services

Augments SGX instructions with Intel-provided closed-source components:

- Set up the CPU to create quotes
  - Provisioning enclave
  - Launch enclave
  - Quoting enclave
- Platform service enclaves
  - Monotonic counters
  - Trusted relative time

How can SGX help with blockchains, cryptocurrencies, and smart contracts?

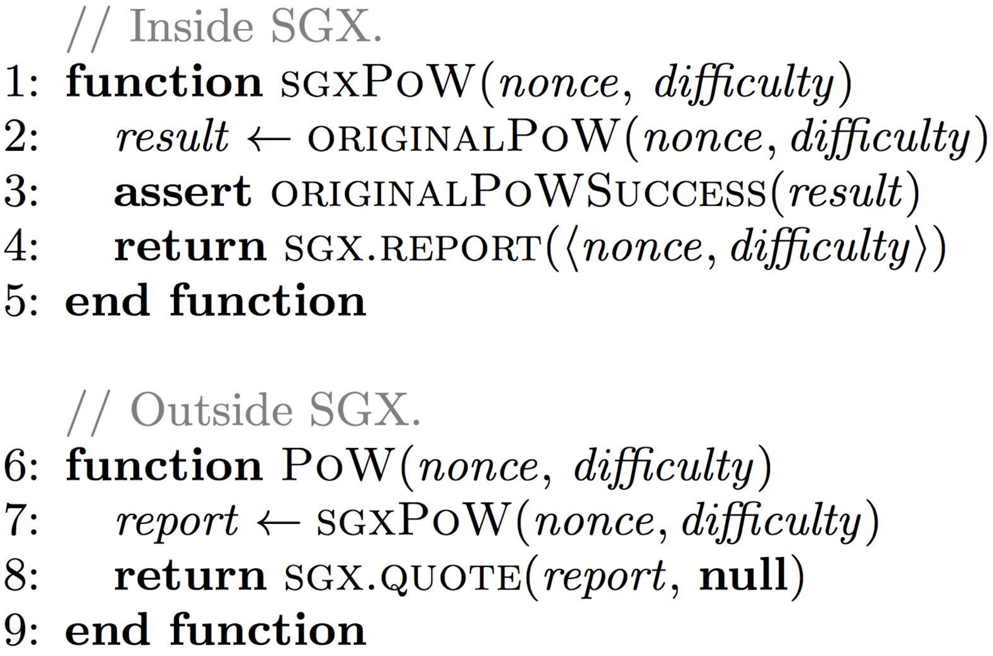
Consensus

SGX and proof of work 💪

Run existing proof of work schemes inside enclave

Create a quote for results

Verify by validating quote



```
// Inside SGX.
function SGXPoW($nonce$, $difficulty$)
    $result$ ← ORIGINALPoW($nonce$, $difficulty$)
    assert ORIGINALPoWSuccess($result$)
    return SGX.REPORT($\langle nonce, difficulty \rangle$)
end function

// Outside SGX.
function PoW($nonce$, $difficulty$)
    $report$ ← SGXPoW($nonce$, $difficulty$)
    return SGX.QUOTE($report$, null)
end function
```

SGX and proof of work 💪

Sidesteps the ASIC vs. non-ASIC debate

Democratizes mining

SGX and proof of work 💪

Wrap other kinds of work. Even useful work?

Doesn’t need efficient proof algorithm

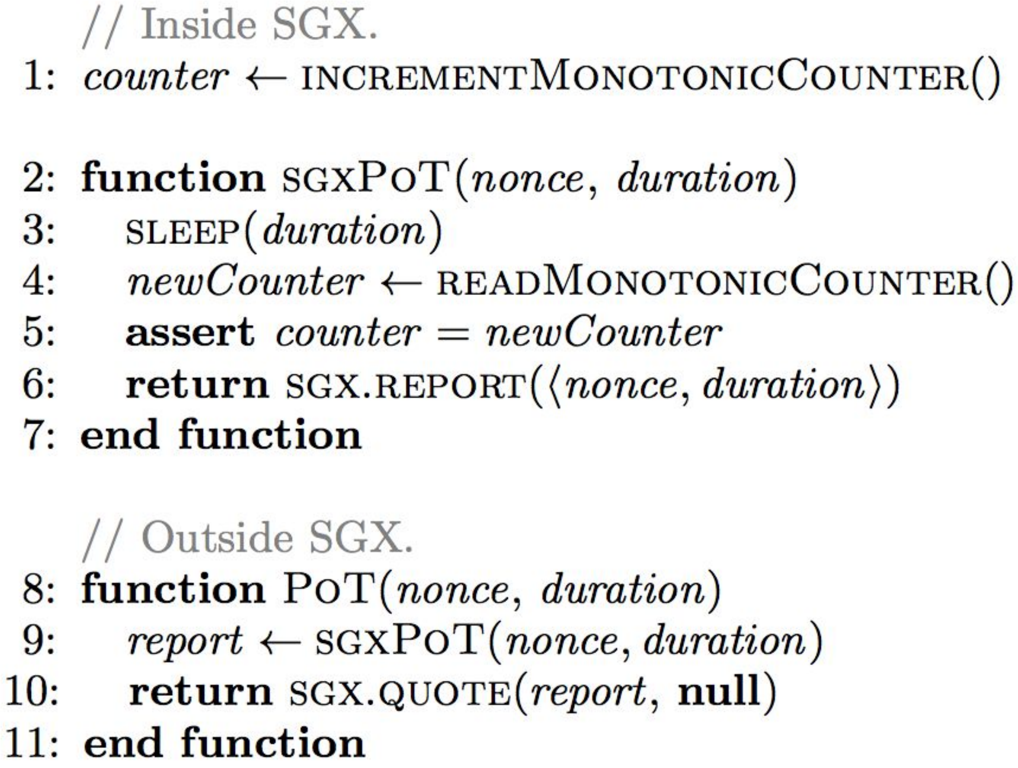
Security depends on SGX

Proof of time ⌛

Proof of work schemes are energy inefficient.

We can use SGX to simulate proof of work on input X

- Figure out how long the work on X would take
- Wait for that long; **don't do any computation**
- Return a quote to prove that you waited for X

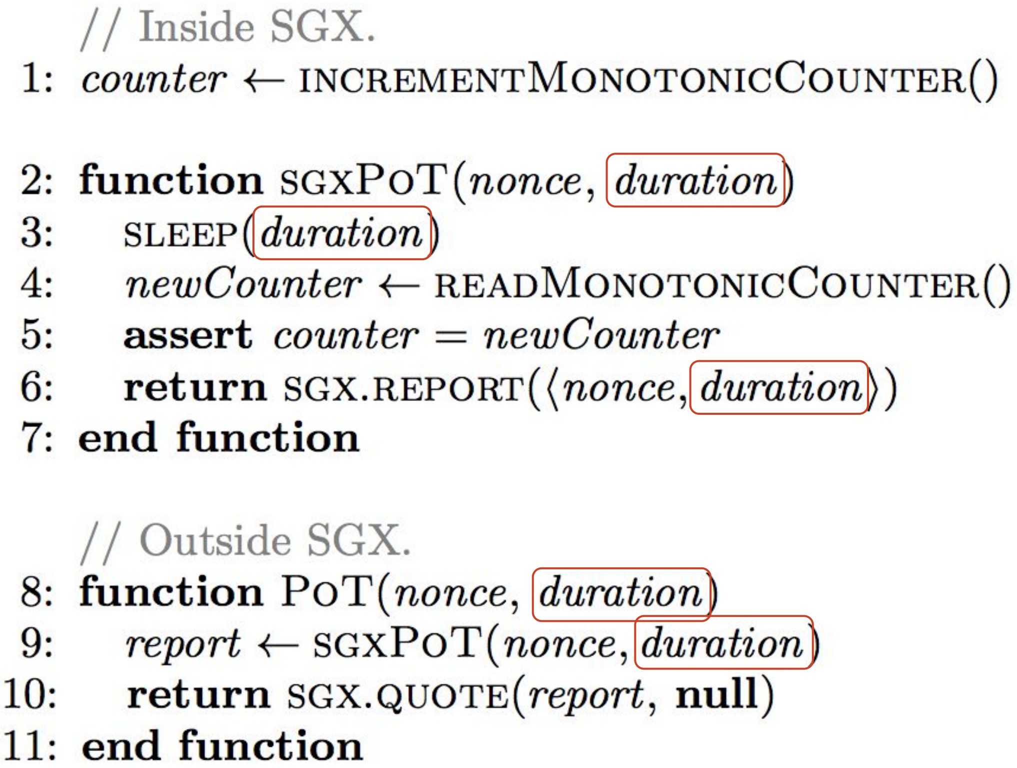


```
// Inside SGX.
$counter \leftarrow$ INCREMENTMONOTONICCOUNTER()

function SGXPOT($nonce$, $duration$)
    SLEEP($duration$)
    $newCounter \leftarrow$ READMONOTONICCOUNTER()
    assert $counter = newCounter$
    return SGX.REPORT($\langle nonce, duration \rangle$)
end function

// Outside SGX.
function POT($nonce$, $duration$)
    $report \leftarrow$ SGXPOT($nonce$, $duration$)
    return SGX.QUOTE($report$, null)
end function
```

Proof of time



```
// Inside SGX.
$counter \leftarrow$ incrementMonotonicCounter()

function sgxPoT($nonce$, $\boxed{duration}$)
    sleep($\boxed{duration}$)
    $newCounter \leftarrow$ readMonotonicCounter()
    assert $counter = newCounter$
    return sgx.report($\langle nonce, \boxed{duration} \rangle$)
end function

// Outside SGX.
function PoT($nonce$, $\boxed{duration}$)
    $report \leftarrow$ sgxPoT($nonce$, $\boxed{duration}$)
    return sgx.quote($report$, null)
end function
```

Proof of time ⌛ - Intel Distributed Ledger

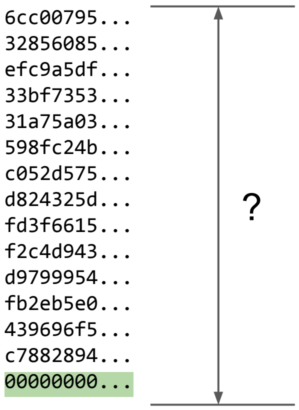
Intel’s own distributed ledger project (Sawtooth Lake) waits a random amount of time in an enclave. Time waited is similar to Bitcoin.

```
6cc00795...
32856085...
efc9a5df...
33bf7353...
31a75a03...
598fc24b...
c052d575...
d824325d...
fd3f6615...
f2c4d943...
d9799954...
fb2eb5e0...
439696f5...
c7882894...
00000000...
```

https://github.com/intelledger

Proof of time ⌛ - Intel Distributed Ledger

Intel’s own distributed ledger project (Sawtooth Lake) waits a random amount of time in an enclave. Time waited is similar to Bitcoin.

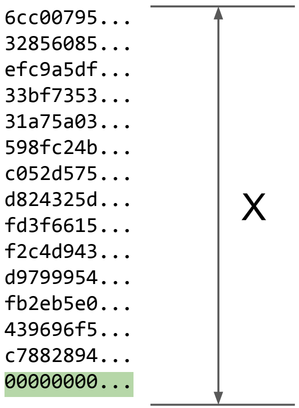


```
6cc00795...
32856085...
efc9a5df...
33bf7353...
31a75a03...
598fc24b...
c052d575...
d824325d...
fd3f6615...
f2c4d943...
d9799954...
fb2eb5e0...
439696f5...
c7882894...
00000000...
```

https://github.com/intelledger

Proof of time ⌛ - Intel Distributed Ledger

Intel’s own distributed ledger project (Sawtooth Lake) waits a random amount of time in an enclave. Time waited is similar to Bitcoin.

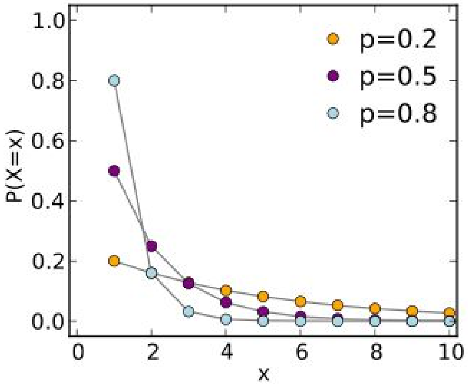


```
6cc00795...
32856085...
efc9a5df...
33bf7353...
31a75a03...
598fc24b...
c052d575...
d824325d...
fd3f6615...
f2c4d943...
d9799954...
fb2eb5e0...
439696f5...
c7882894...
00000000...
```

X

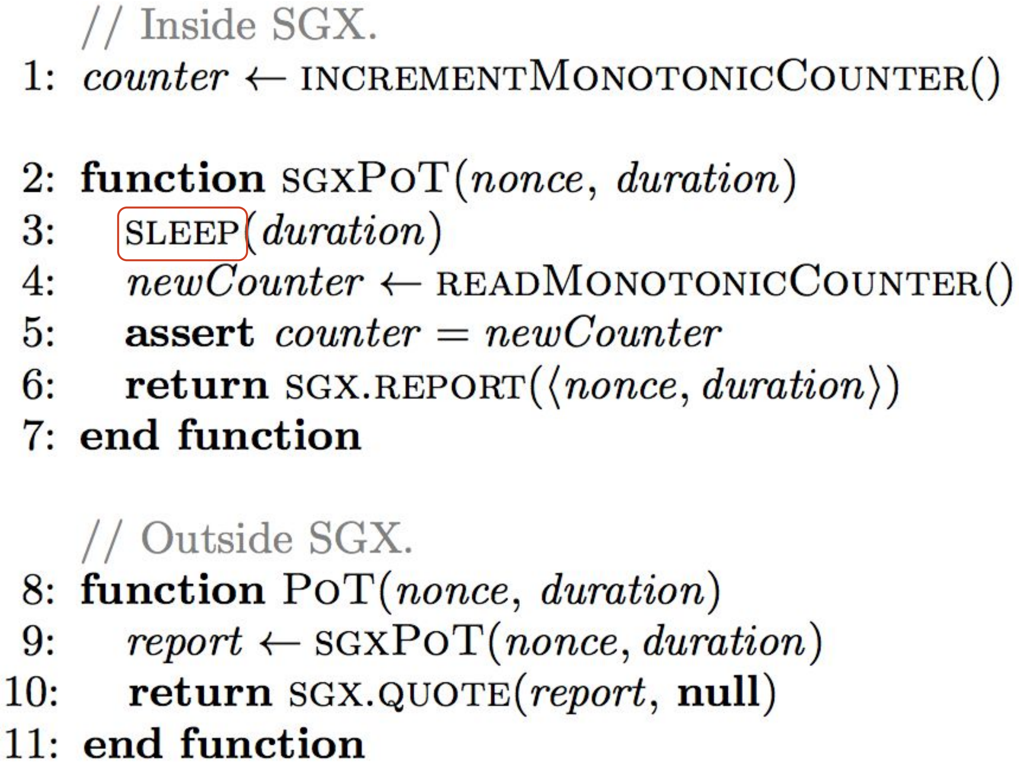
~ geometric distribution

$$\Pr[X=x]=(1-p)^{k-1}p$$



https://github.com/intelledger

Proof of time ⌛

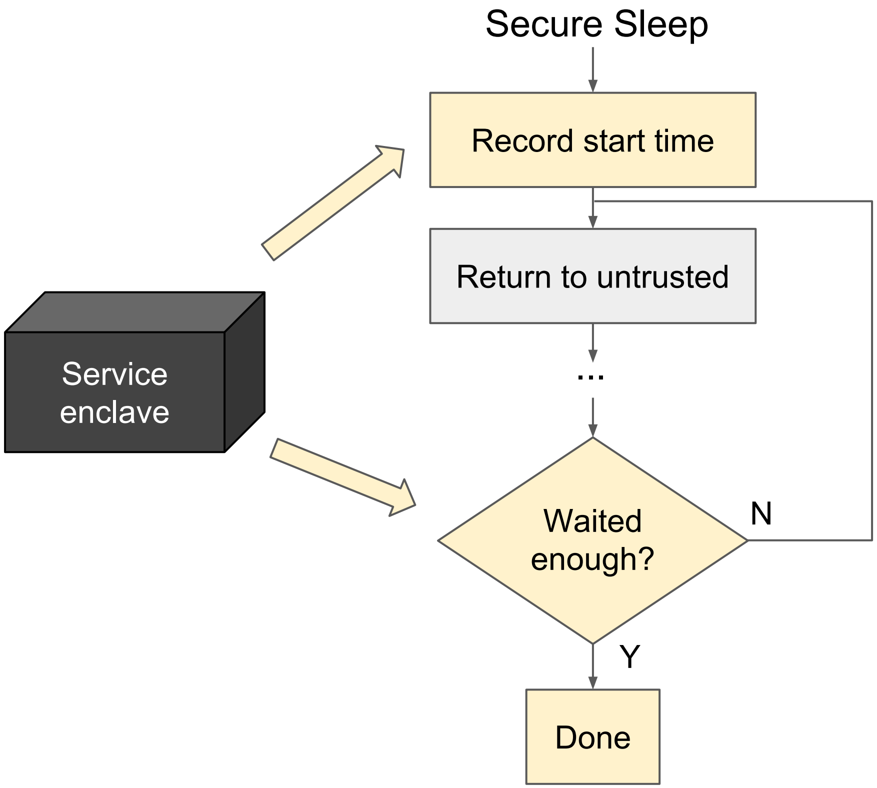


```
// Inside SGX.
$counter \leftarrow$ INCREMENTMONOTONICCOUNTER()

function SGXPOT($nonce$, $duration$)
    SLEEP($duration$)
    $newCounter \leftarrow$ READMONOTONICCOUNTER()
    assert $counter = newCounter$
    return SGX.REPORT($\langle nonce, duration \rangle$)
end function

// Outside SGX.
function POT($nonce$, $duration$)
    $report \leftarrow$ SGXPOT($nonce$, $duration$)
    return SGX.QUOTE($report$, null)
end function
```

Proof of time ⌛ - Implementation



Secure Sleep

Record start time

Return to untrusted

...

Waited enough?

N

Y

Done

Service enclave

Proof of time ⌛ - Implementation

Key challenge: prevent parallel execution

- Doesn't use all CPU resources
- How can enclave instances know about each other?

Proof of time ⌛ - Implementation

Key challenge: prevent parallel execution

Solution: counters

- (during node setup) create a monotonic counter
  ```
  sgx_create_monotonic_counter(*counter_uuid, *value)
  ```
- increment a monotonic counter when you start
  ```
  sgx_increment_monotonic_counter(*counter_uuid, *value)
  ```
- sleep
- check that it’s still the same
  ```
  sgx_read_monotonic_counter(*counter_uuid, *value)
  ```

Proof of time ⌛ - Implementation

Key challenge: prevent parallel execution

Solution: counters

- (during node setup) create a monotonic counter
  ```
  sgx_create_monotonic_counter(*counter_uuid, *value)
  ```
- increment a monotonic counter when you start
  ```
  sgx_increment_monotonic_counter(*counter_uuid, *value)
  ```
- sleep
- check that it’s still the same
  ```
  sgx_read_monotonic_counter(*counter_uuid, *value)
  ```

Proof of time ⌛ - Implementation

Key challenge: what was our monotonic counter?

- Communication must pass through untrusted application
- Storage must pass through untrusted application

Proof of time ⌛ - Implementation

Key challenge: which monotonic counter?

Solution: all of them

> **SGX_ERROR_MC_OVER_QUOTA**
>
> The enclave has reached the quota(256) of Monotonic Counters it can maintain

https://software.intel.com/sites/default/files/managed/d5/e7/Intel-SGX-SDK-Users-Guide-for-Windows-OS.pdf

Proof of time ⌛ - Implementation

Key challenge: which monotonic counter?

Solution: all of them

- create 256 monotonic counters
- sleep
- make sure you still have all 256

Proof of time ⌛ - Compromised CPUs

Big incentive to compromise individual CPUs

You can mine way faster than the rest of the network

Intel manages a revocation list of known compromised CPUs

Proof of time ⌛

Desirable properties:

- ASICs provide no advantage
- No wasted energy

But CPU compromise is still an issue

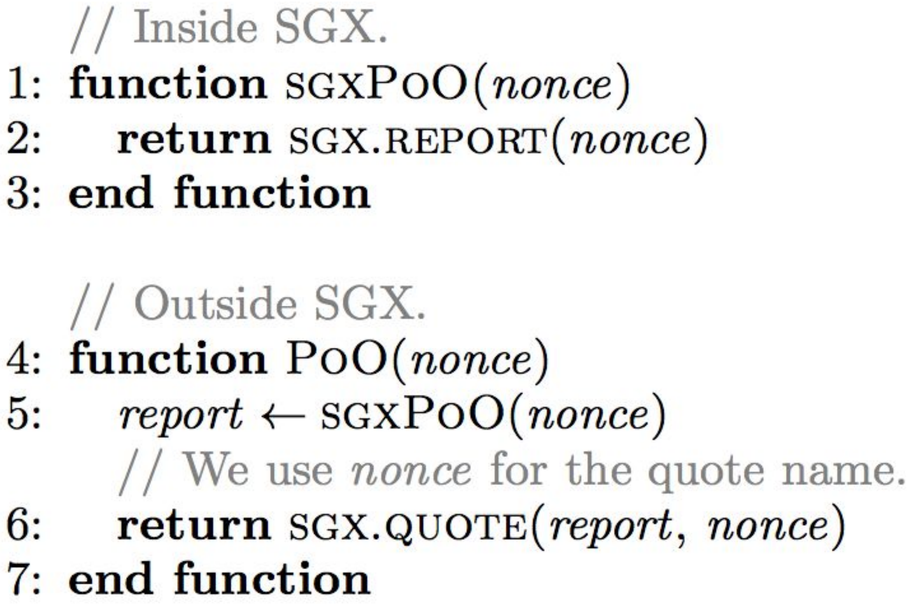
Reduces mining to ownership of SGX CPUs

Proof of ownership

Just count number of CPUs “voting” for a new block

The votes (SGX quotes) are privacy preserving

Intel’s Enhanced Privacy ID (EPID) algorithm can determine whether two quotes with same **name** came from the same CPU or not



```
    // Inside SGX.
function SGXPoO(nonce)
    return SGX.REPORT(nonce)
end function

    // Outside SGX.
function PoO(nonce)
    report ← SGXPoO(nonce)
    // We use nonce for the quote name.
    return SGX.QUOTE(report, nonce)
end function
```

Proof of ownership

Scalability: Name Base Mode

With Name Base Mode, the scheme implementer must ensure a particular name is not used too much.

https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services

Proof of ownership

Scalability: network messages

Every node votes on each block

That’s a lot of votes

Consensus Overview

	ASIC resistant	Energy efficient	Time efficient	Scalable
Bitcoin	no	no	no	yes
SGX proof of work	yes	no	no	yes
Proof of time	yes	yes	no	yes
Proof of ownership	yes	yes	yes	no

Working on combining these to compensate for individual schemes’ shortcomings

Smart Contracts

Smart Contracts

One node executes the contract in an enclave

Create a quote with the result

Disseminate the quote

Easily combine confidentiality and auditability

Smart Contracts

Only one node has to execute the contract

Others just verify the quote

Non-deterministic contract code

Reduces smart contracts to availability

Issues

Issue #1
Unclear licensing and terms of use for SGX by Intel

SGX is being shipped in hardware, but to launch an enclave, it has to be authorized by Intel’s launch enclave.

It is unclear how will launch enclave decide that, probably based on a business partnership with Intel.

Intel might terminate previously given authorization to launch at their discretion.

Issue #2
Centralized remote attestation service

To do a remote attestation, you have to contact Intel’s cloud service.*

This allows them to verify quotes against compromised CPUs and other revocation lists.

A 3rd party (decentralized?) alternative might be possible to be implemented.

But would they allow such 3rd party enclave to run?

*https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services

Issue #3
Disabled by default

A BIOS setting.

Not a problem for miners, but what about non-technical end-users?

Mobile (thin) devices don't even have SGX.

Summary

Promising new primitives. More work needed to create a robust, tamper-proof solution.

Already a nice match to augment permissioned and centralized cryptocurrencies and give additional trust anchor to simplify and optimize the rest of the stack.

Unclear if suitable for decentralized cryptocurrencies: an open ecosystem around SGX would help alleviate concerns.

Towards Automation of Correctness Proofs of Smart Contracts

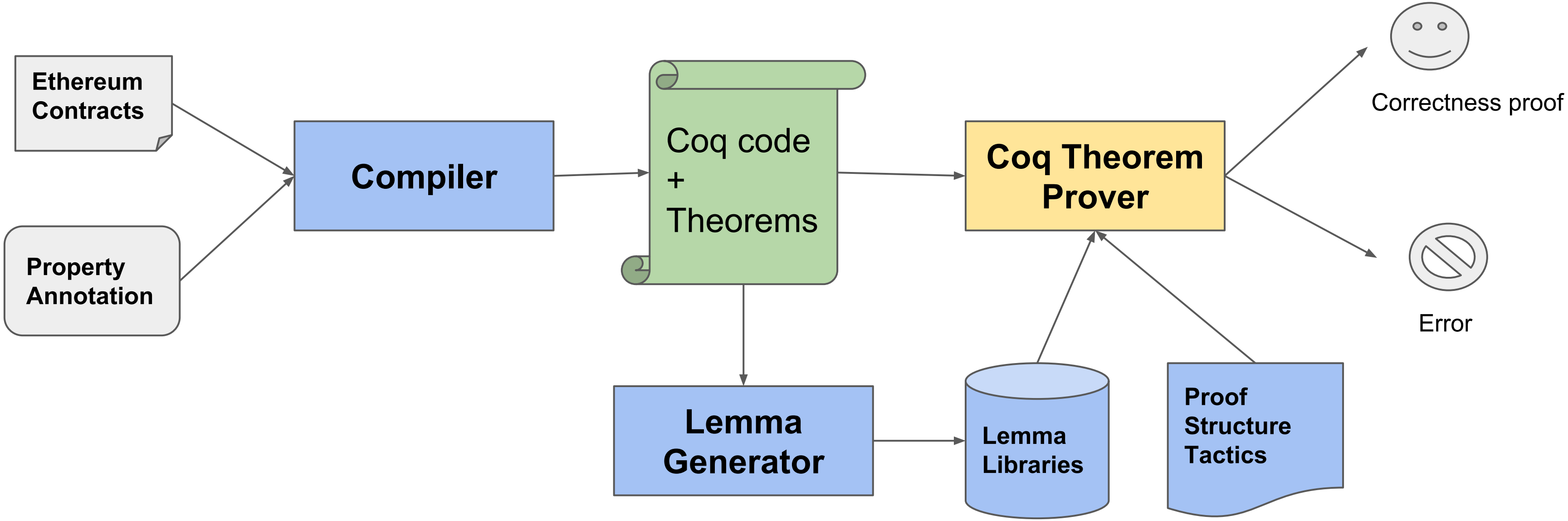
Dawn Song

Joint work with Aymeric Fromherz

Smart Contract is Hard to Get Right

- Smart contract can be complex & subtle
- Corner cases may not be handled properly
  - E.g., leaking money in certain cases [Shi et al.]
- No tools to help analyze & prove correctness of smart contracts

Automatic Correctness Proof via Coq



Example: Preservation Property

- Certain property of global state stay constant over state changes
  - Banking system: Total money across different accounts stay constant at any point
  - Auction system: there is one highest bidder in the system at any point
  - Voting system: total counts (votes + non-votes) stay the same in the system at any point

- Automatic proof
  - Proof structure tactics for preservation property
  - Automatic generation of lemma libraries

- Proofs & errors found in real-world ethereum contracts

Conclusion

- Smart contract is hard to get right

- First step towards automatic analysis and proof of correctness of smart contract

- Lots more to do