Towards Formal Verification of State Continuity for Enclave Programs

Mohit Kumar Jangid
The Ohio State University

Guoxing Chen
Shanghai Jiao Tong University

Yinqian Zhang
Southern University of Science and Technology

Zhiqiang Lin
The Ohio State University

Outline

- Background
- Motivation
- Our approach
- Case study - Sawtooth
- Conclusion

Outline

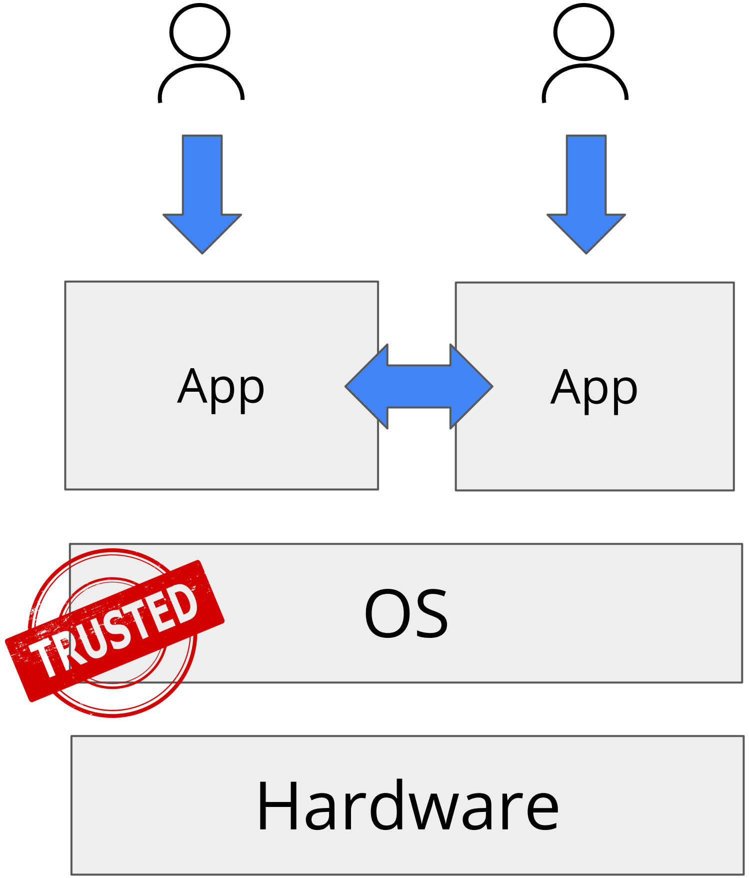
- **Background**
- Motivation
- Our approach
- Case study - Sawtooth
- Conclusion

Operating system protects user data by

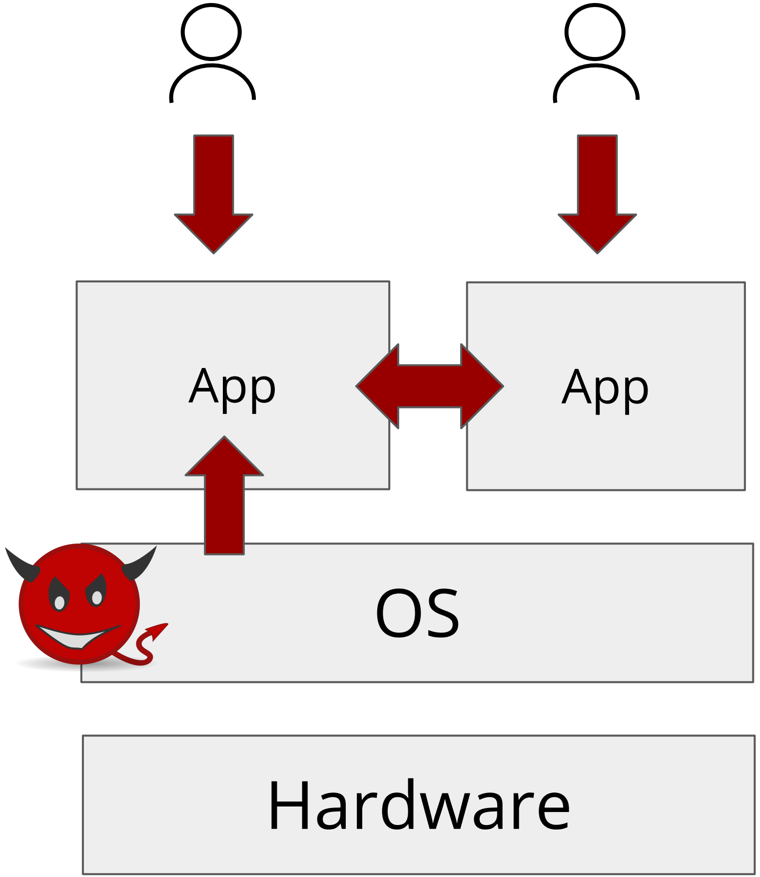
- Process isolation
- Access control (privileged access to devices)

Assumption

OS is trusted

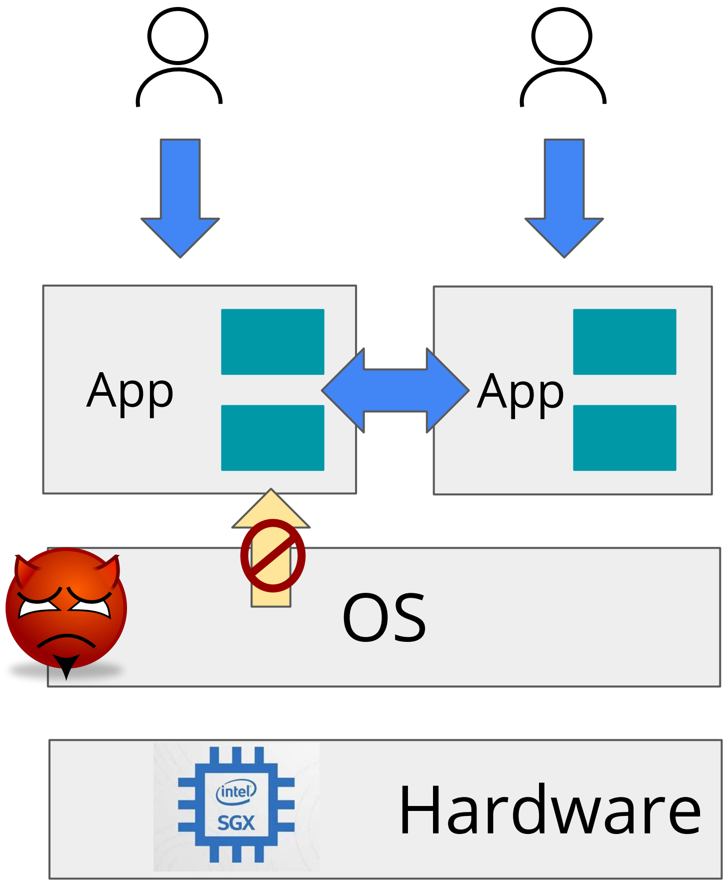


Is it possible to protect the user data when OS is compromised?



Intel SGX

- Trusted hardware solution to protect confidentiality and integrity of the runtime code and data.
- App is divided into trusted and untrusted code section.
- Hardware encrypted trusted code run inside the protected memory regions (enclaves).

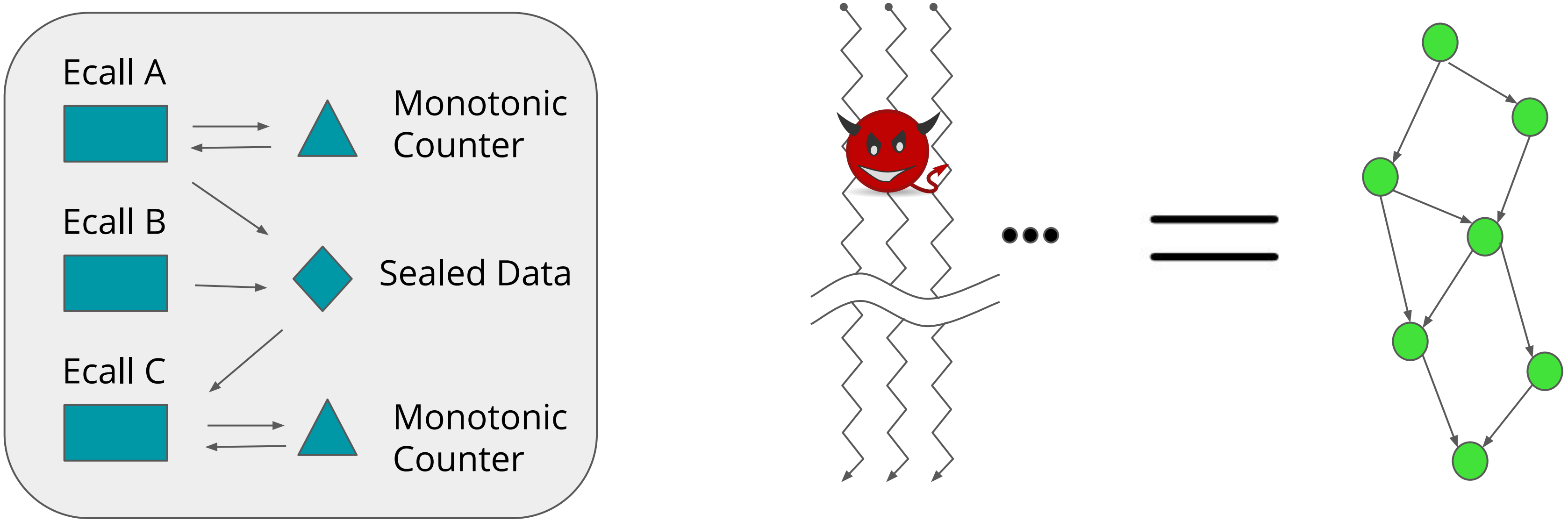


State Continuity

- Classic definition: protected module must resume from the same execution state after TCB (Trusted Computing Base) interrupts
- New TCB modules in SGX context:
  - Enclave memory (local/global variables)
  - Non-volatile memory (monotonic counters)
  - Persistent storage (sealed data)
- New threat model in SGX context:
  - Controls the privileged code (OS and application code)
  - Arbitrary thread and process instantiation
  - Permute, reorder enclave calls
  - Access to ecall or ocall arguments and returns
  - Replay, modify of data in untrusted code

State Continuity for Enclave Programs

- Enclave program states always executes on the expected TCB state under the SGX threat model and TCB interrupts



Enclave Program   All Enclave Executions   Expected TCB States

Example SGX Application -- Sawtooth

- Permissioned Blockchain Framework
- Consensus algorithm: Proof-of-Elapsed-Time (PoET)
- Leverages Intel SGX for fair node participation
- Each node workflow
  - Signup and register into the blockchain network
  - Participate in the block leader election

Sawtooth Block Leader Election

Ecall E1

Election 1 Create Reference Objects	• Generate random wait duration • Create reference monotonic counter (**MC_ref**) • Seal the duration and MC_ref

Wait random duration

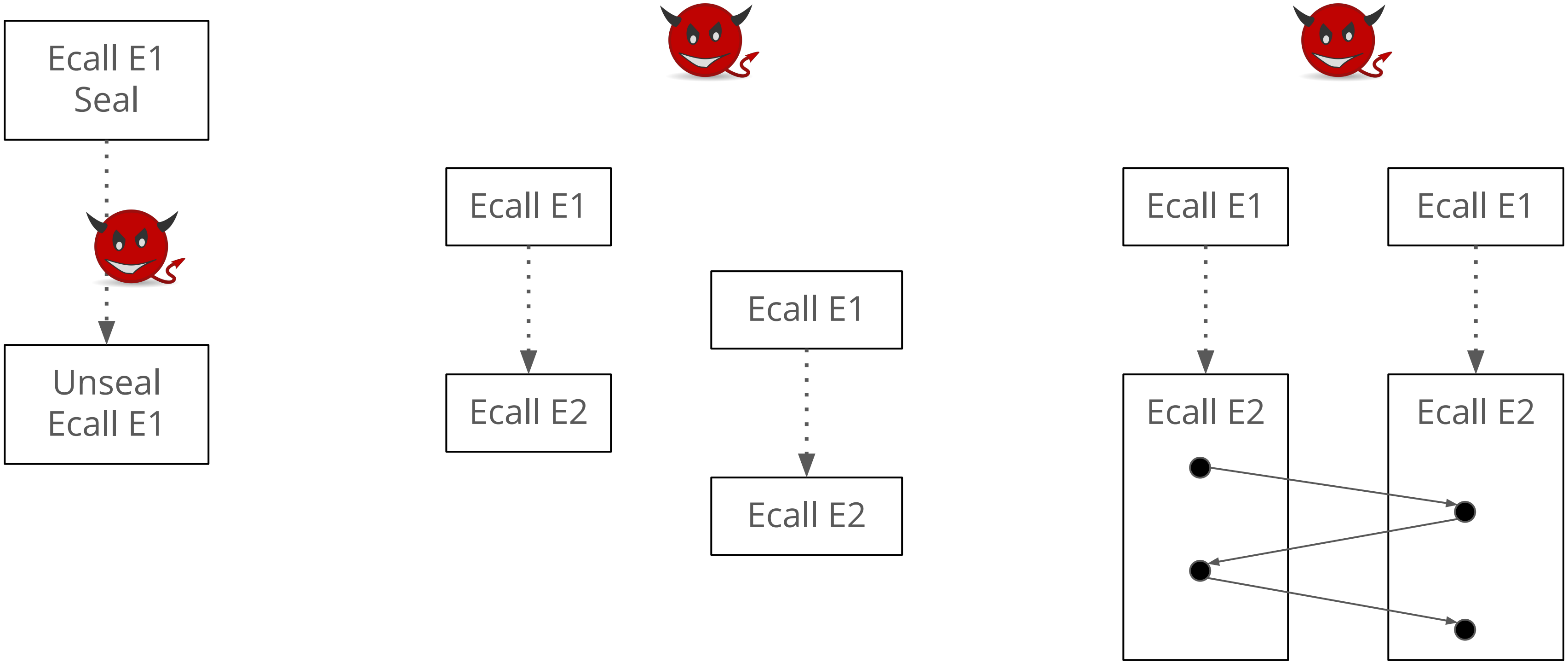
Ecall E2

Election 2 Verify Proof of Elapsed Time	• Unseal and verify the sealed object • Verify elapsed time • Compare **MC_ref** X X X • PoETCertificate **Monotonic Counter ++**

Sawtooth Expected TCB States

1. Monotonic Counter Value < MC_Ref	~~**PoETCertificate**~~
2. Monotonic Counter Value = MC_Ref	**PoETCertificate**
3. Monotonic Counter Value > MC_Ref	**Abort**

What Could Go Wrong?



Outline

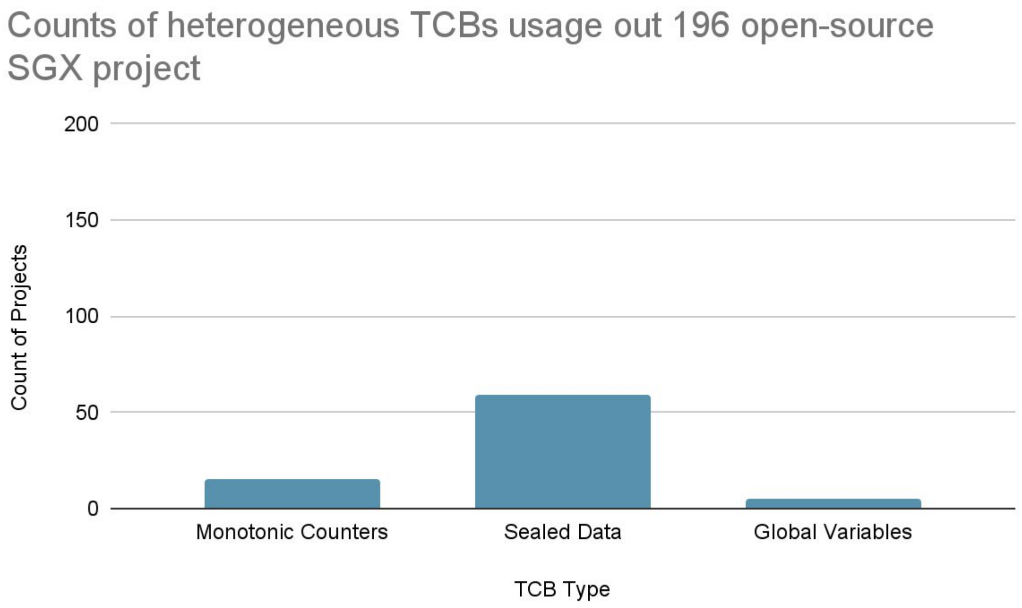
- Background
- **Motivation**
- Our approach
- Case study - Sawtooth
- Conclusion

Maintaining State Continuity is Important

State continuity TCB modules are prevalent in many open SGX applications.

196 open source SGX applications

- 59 -- Sealing
- 15 -- Monotonic Counters
- 05 -- Global variables.



The Research Problem

State continuity properties are difficult to verify in the SGX environment. Why?

Manual efforts is tedious and error prone

1. Clearly understand trusted & untrusted boundary
2. Correct coordination of heterogeneous TCB modules
3. Carefully apply thread synchronization and locks

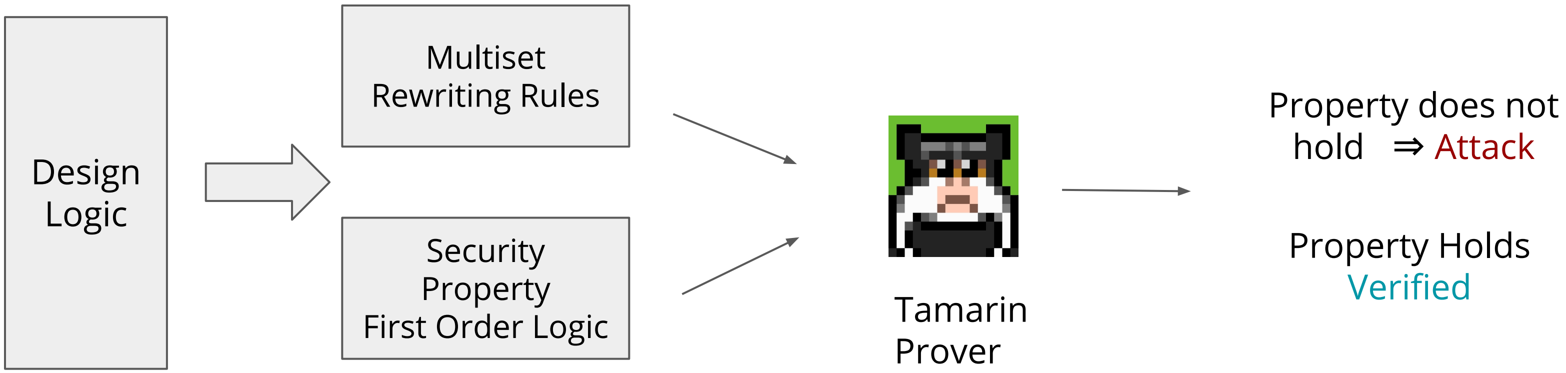
Is there a systematic approach to verify state continuity ?

Outline

- Background
- Motivation
- **Our approach**
- Case study - Sawtooth
- Conclusion

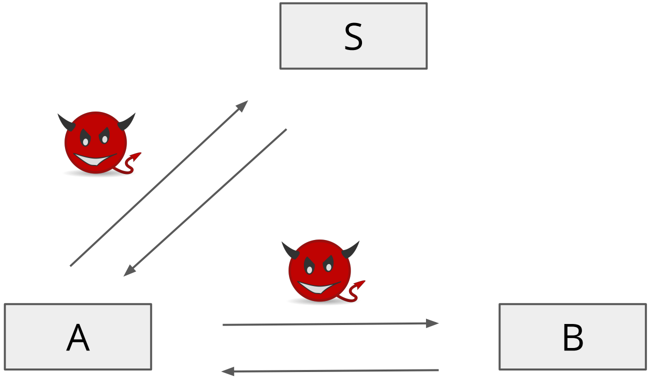
Our Approach

- Use Symbolic Verification Tool -- Tamarin, to verify state continuity property

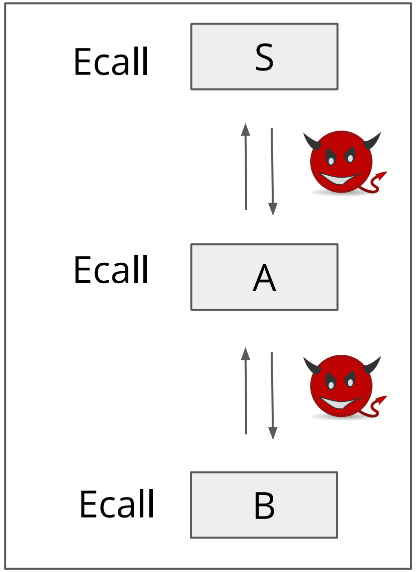


Key Observation

Cryptographic Protocols and SGX Environment share common features

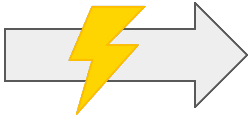


Key Exchange Protocols



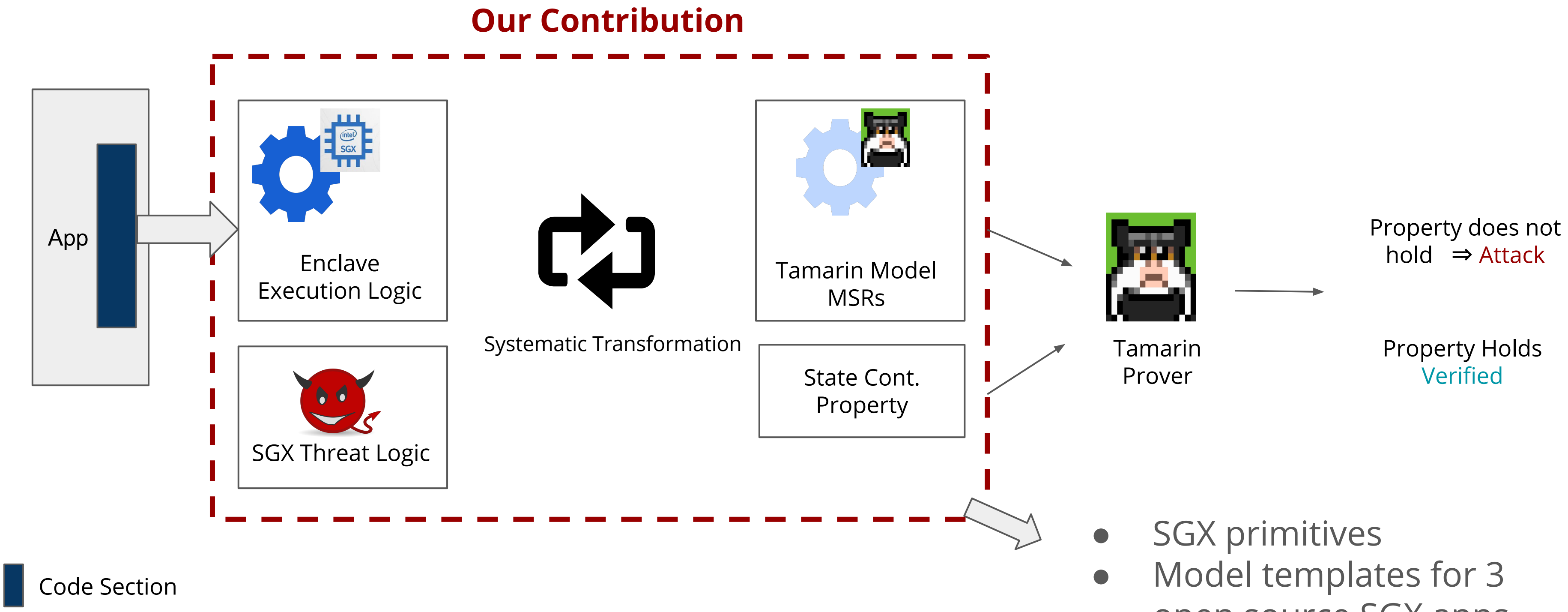
SGX Environment

Tamarin MSR and query language



State Cont. Properties
SGX Thread Model

Our Approach



- SGX primitives
- Model templates for 3 open source SGX apps

Transformation icon cite: https://icon-library.com/icon/transformation-icon-3.html

Model Primitives used in our work

SGX primitives	Programming primitives
1. Enclave threads 2. Association network of SGX entities 3. Monotonic counters 4. Local/Global variables 5. SGX threat model 6. Key derivation 7. Sealing	1. Locks 2. Loops 3. Branching 4. Database (Read only)

SGX Threat Model

SGX Threat Model Construction	**Realized by**
Thread and process instantiation	Using the thread policy based on the ecall facts F_{ecall} in the first enclave thread rule and binding ecall sequences of rules using thread facts F_{thread}
Permute or reorder ecalls	Modeling the first enclave thread rule open to executability without order dependencies of timepoints and facts
Pause enclave execution at instruction level	Modeling instructions in individual rules and utilizing atomic rule executability
Read access to ecall returns; Read/Modify access to ecall or ocall arguments and returns	Arguments and returns pass through public channel
Replay, modify of sealing, ecall or arguments and returns	Public channel use in combination Tamarin's inbuilt Dolev Yao adversary capabilities

Outline

- Background
- Motivation
- Our approach
- **Sawtooth -- Tamarin Model**
- Conclusion

Recall -- Sawtooth

Each node workflow

1. Signup and register into the blockchain network
2. Election Ecall 1
3. Election Ecall 2

Recall -- Sawtooth Block Leader Election

Ecall E1

Election 1 Create Reference Objects	● Generate random wait duration ● Create reference monotonic counter (**MC_ref**) ● Seal the duration and MC_ref

●
●
● Wait random duration

Ecall E2

Election 2 Verify Proof of Elapsed Time	● Unseal and verify the sealed object ● Verify elapsed time ● Compare **MC_ref** ● PoETCertificate **Monotonic Counter ++**

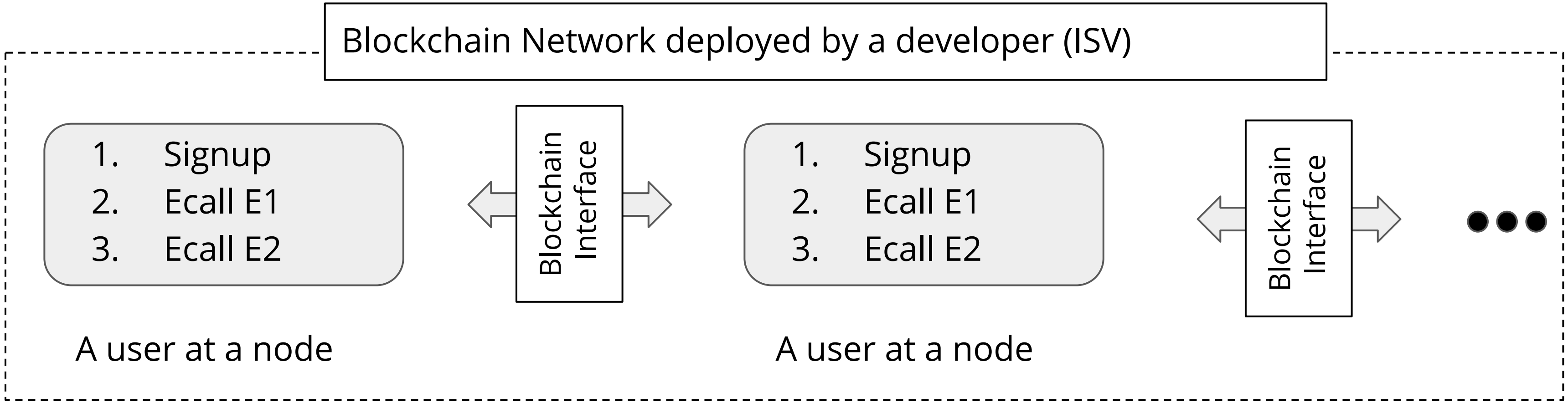
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter

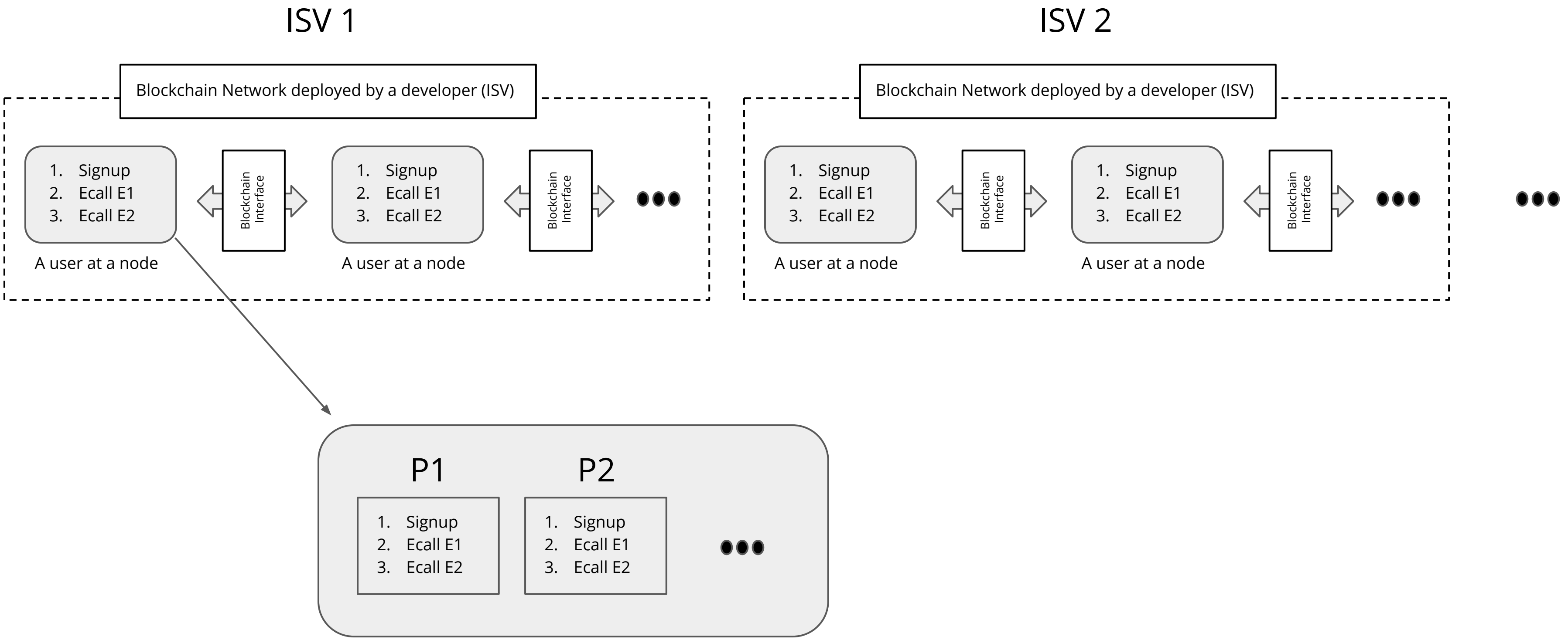
Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - **SGX entities -- ISV, User, Nodes, Processes**
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter

SGX Entities



SGX Entities

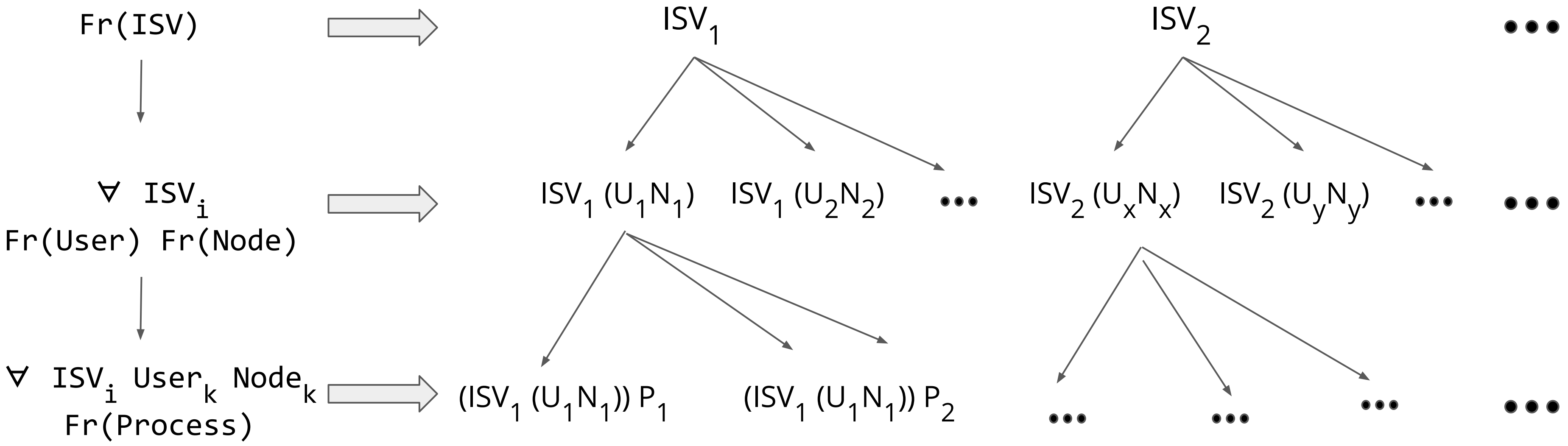


Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - **Entity association network**
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter

Entity Association Network

- Tamarin Fr(*) ***Fact*** produces unique variables
- Tamarin ***Rules*** can in instantiated unbounded times
- Variables can be passed on through ***Rules*** using Tamarin ***Facts***

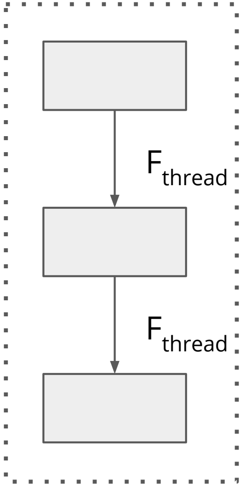


Tamarin Model for Sawtooth

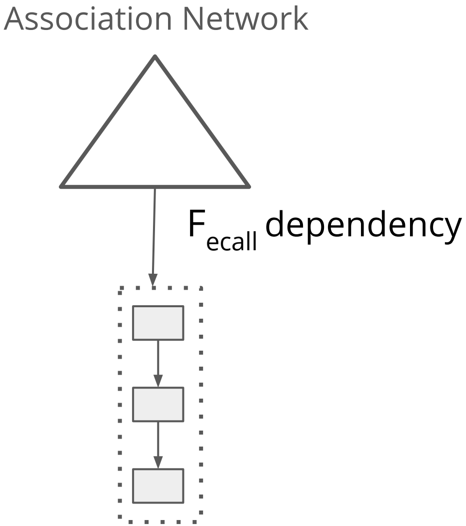
- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - **Enclave threads**
  - Sealed sign-up information
  - Monotonic Counter

Enclave Thread Construction

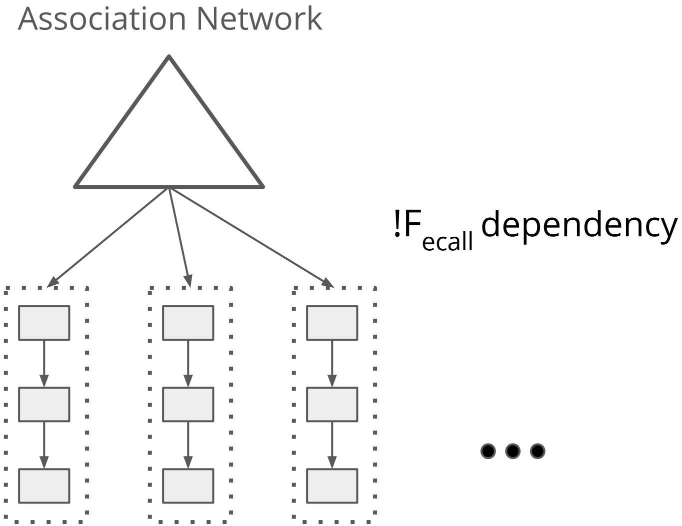
- ***Linear Fact (F)*** can be consumed only once.
- ***Persistent Fact (!F)*** can be consumed unbounded times.
- Linear and persistent **Fact dependencies** allows configuration of single and multiple thread



Enclave Thread



Single Thread



Multiple Threads

Tamarin Model for Sawtooth

- What components of the workflow do we need?
  - SGX entities -- ISV, User, Nodes, Processes
  - Entity association network
  - Enclave threads
  - Sealed sign-up information
  - Monotonic Counter

State Continuity Property

Fair election participation of each node in the blockchain requires that a node must not generate two certificates with same MC_ref

First Order Logic Query

```
All
PoETCertificate ( node , MC_ref ) @t1 &
PoETCertificate ( node , MC_ref ) @t2
        == > # t1 =# t2
```

Verification

Apply patch in the model

Sawtooth Model
Multiset Rewriting Rul

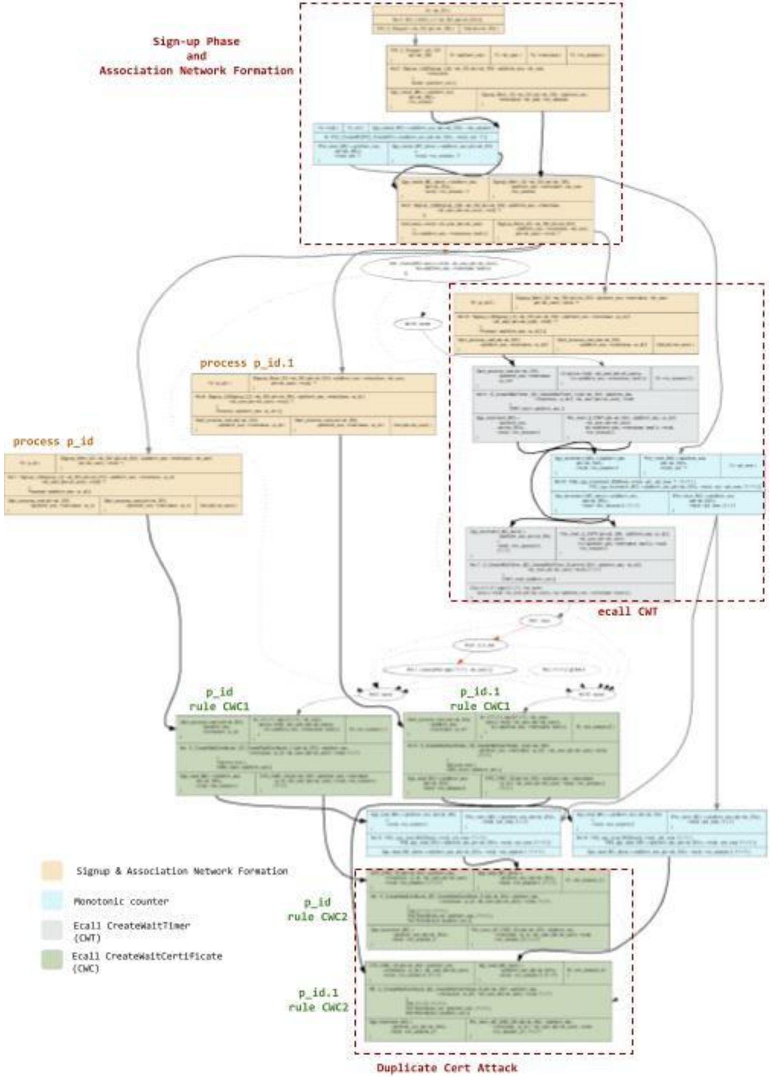
State Cont. Property
First Order Logic

Tamarin Prover

Property does not hold
⇒ Attack

Property Holds
Verified

Sawtooth Attack Trace



Summary of Case Studies

App	Attack Discovery Time	Verification Time	# Rules	Model LOC
Sawtooth	1m 18s	25s	11	300
Heartbeat	7s	2h 4m 7s	11	250
BI-SGX	36s	37s	18	450

Conclusion

- First attempt towards using symbolic verification tools to verify the state continuity for SGX enclave programs.

- We demonstrate our approach using three open-source SGX applications, resulting into reusable SGX primitives and model templates.

- Tamarin Prover can effectively model SGX-specific semantics and operations; and state continuity properties.

- Our Tamarin code is released at Github: https://github.com/OSUSecLab/SGX-Enclave-Formal-Verification.

Thank you

Mohit Kumar Jangid
The Ohio State University

Guoxing Chen
Shanghai Jiao Tong University

Yinqian Zhang
Southern University of Science and Technology

Zhiqiang Lin
The Ohio State University

Follow up questions at jangid.6@osu.edu.