Towards End-to-End Verified TEEs via Verified Interface Conformance and Certified Compilers

Farzaneh Derakhshan  
(fderakhs@andrew.cmu.edu)  
Joint work with Zichao Zhang, Amit Vasudevan, and Limin Jia
Trusted Execution Environments (TEE)

Non secure world

Rich execution environment

Untrusted App1

Untrusted App2

Normal OS

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

Shared memory

Core 1

Core 2

Control registers

cp

IDT

Page table

I/O devices

Hardware
Trusted Execution Environments (TEE)

Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.
Trusted Execution Environments (TEE)

Goal:
• Run multiple mutually distrusting programs simultaneously on shared hardware.

Application
• Cloud computing
• Secure banking

Hardware
Shared memory
I/O devices
Page table
Control registers
pc
IDT

Core 1
Core 2

Non secure world
Rich execution environment
Untrusted App1
Untrusted App2
Normal OS

Secure world
Trusted execution environment
Trusted App1
Trusted App2
Trusted OS

• Run multiple mutually distrusting programs simultaneously on shared hardware.

Application
• Cloud computing
• Secure banking
Trusted Execution Environments (TEE)

Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm
Trusted Execution Environments (TEE)

Goal:
- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application
- Cloud computing
- Secure banking

Example Trusted OS
- Hypervisors
- Trusty for Android
- OP-TEE for Arm
Trusted Execution Environments (TEE)

Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application

- Cloud computing
- Secure banking

Example Trusted OS

- Hypervisors
- Trusty for Android
- OP-TEE for Arm
Trusted Execution Environments (TEE)

Goal:
- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application
- Cloud computing
- Secure banking

Example Trusted OS
- Hypervisors
- Trusty for Android
- OP-TEE for Arm
Trusted Execution Environments (TEE)

Goal:

- Run multiple mutually distrusting programs simultaneously on shared hardware.

Application

- Cloud computing
- Secure banking

Example Trusted OS
- Hypervisors
- Trusty for Android
- OP-TEE for Arm
**Trusted Execution Environments (TEE)**

**Goal:**
- Run multiple mutually distrusting programs simultaneously on shared hardware.

**Application**
- Cloud computing
- Secure banking

**Example Trusted OS**
- Hypervisors
- Trusty for Android
- OP-TEE for Arm

**Subversion of a TEE means the attacker takes full-control over the entire platform!**
TEE formal verification removes many of the vulnerabilities

- Full functional correctness

- Examples: Ironclad apps, sel4, mCertiKOS

- Advantages: Strong guarantee

- Disadvantages: Not developer friendly, Not update friendly, High cost of verification (time and dollar!)

- Specify security properties in lieu of full-functional correctness

- Examples: XMHF, uberXMHF,

- Advantages: Development friendly, use source-level automated verification tools,

- Disadvantages: lack of guarantees on the compiled code.
TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

• **Examples:** *Ironclad apps, sel4, mCertiKOS*
TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

• **Examples:** Ironclad apps, sel4, mCertiKOS
• **Advantages:** Strong guarantee
TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** Ironclad apps, sel4, mCertiKOS
- **Advantages:** Strong guarantee
- **Disadvantages:** Not developer friendly, Not update friendly, High cost of verification (time and dollar!)
TEE formal verification removes many of the vulnerabilities

➡ Full functional correctness

- **Examples:** Ironclad apps, sel4, mCertiKOS
- **Advantages:** Strong guarantee
- **Disadvantages:** Not developer friendly, Not update friendly, High cost of verification (time and dollar!)

➡ Specific security properties in lieu of full-functional correctness
TEE formal verification removes many of the vulnerabilities

 ➪ Full functional correctness

- **Examples:** Ironclad apps, sel4, mCertiKOS
- **Advantages:** Strong guarantee
- **Disadvantages:** Not developer friendly, Not update friendly, High cost of verification (time and dollar!)

 ➪ Specific security properties in lieu of full-functional correctness

- **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki

TEE formal verification removes many of the vulnerabilities

- Full functional correctness

  - **Examples:** Ironclad apps, sel4, mCertiKOS
  - **Advantages:** Strong guarantee
  - **Disadvantages:** Not developer friendly, Not update friendly, High cost of verification (time and dollar!)

- Specific security properties in lieu of full-functional correctness

  - **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki
  - **Advantages:** Development friendly, use source-level automated verification tools

TEE formal verification removes many of the vulnerabilities

- Full functional correctness
  
  - **Examples:** Ironclad apps, sel4, mCertiKOS
  - **Advantages:** Strong guarantee
  - **Disadvantages:** Not developer friendly, Not update friendly, High cost of verification (time and dollar!)

- Specific security properties in lieu of full-functional correctness
  
  - **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki
  - **Advantages:** Development friendly, use source-level automated verification tools
  - **Disadvantages:** Weaker guarantee

Specific security properties in lieu of full-functional correctness

- **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki
- **Advantages:** Development friendly, use source-level automated verification tools
- **Disadvantages:** Weaker guarantee
Specific security properties in lieu of full-functional correctness

- **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki
- **Advantages:** Development friendly, use source-level automated verification tools
- **Disadvantages:** Weaker guarantee

Prior approaches lack guarantees on the compiled code
Specific security properties in lieu of full-functional correctness

- **Examples:** XMHF, uberXMHF, Security Microvisor, Contiki
- **Advantages:** Development friendly, use source-level automated verification tools
- **Disadvantages:** Weaker guarantee

Prior approaches lack guarantees on the compiled code

Our approach - Compartmentalization and certified compilers to aid verification:

- Compartments as units for verification and compilation.
- Allows us to bring the security properties down to the compiled code.
Compartments schema

Non secure world

- Rich execution environment
  - Untrusted App1
  - Untrusted App2
  - Normal OS

Secure world

- Trusted execution environment
  - Trusted App1
  - Trusted App2
  - Trusted OS

Shared memory

Hardware

- Control registers: pc, IDT, Page table
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

Shared memory

Core 1

Core 2

Control registers

pc
IDT
Page table

Hardware
Compartments schema

Secure world

Trusted execution environment

- Trusted App1
- Trusted App2
- Trusted OS

Shared memory

Core 1  Core 2

Hardware

Control registers

pc  IDT  Page table
Compartment schema

Secure world

Trusted execution environment

- Trusted App1
- Trusted App2
- Trusted OS

Shared memory

Control registers
- pc
- IDT
- Page table

Hardware
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

uberobject

Internal functions

Public interface

F1

F2
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Exclusive resources

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

Hardware components

Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware
Compartments schema

Secure world

Trusted execution environment
- Trusted App1
- Trusted App2
- Trusted OS

uberobject
- F1
- F2

Internal functions
Public interface

Shared memory

Hardware
- Core 1
- Core 2
- Control registers: pc, IDT, Page table

Exclusive resources
Page table
Compartments schema

Secure world

Trusted execution environment

Trusted App1
Trusted App2
Trusted OS

uberobject

Internal functions

Public interface

F1
F2

Exclusive resources

Page table

Shared memory

Core 1
Core 2

Control registers
pc IDT Page table

Hardware

Pre-condition P

Public interface
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

Internal functions

Public interface

F1

F2

Exclusive resources

Page table

Shared memory

Core 1

Core 2

Control registers

pc

IDT

Page table

Hardware

Pre-condition

P

Post-condition

Q

Public interface

Core 2

Core 1

Hardware

Shared memory

Control registers

pc

IDT

Page table

Hardware
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Shared memory

Control registers

Hardware

Core 1

Core 2

Shared memory

Control registers

Hardware

Page table

Pre-condition

Post-condition

Public interface

Exclusive resources

Page table

Compartments schema
Compartments schema

Secure world

Trusted execution environment

Trusted App1

Trusted App2

Trusted OS

uberobject

F1

F2

Internal functions

Public interface

Exclusive resources

Page table

Core 1

Core 2

Control registers

Shared memory

Hardware

pre-condition

Post-condition

Pre-condition

Post-condition

P

P'

Q

Q'

PC

IDT

Page table
Compartment schema

Secure world

Trusted execution environment
- Trusted App1
- Trusted App2
- Trusted OS

uberobject

Internal functions
- F1
- F2

Public interface

Shared memory

Core 1
Core 2

Control registers
- pc
- IDT
- Page table

Hardware

Exclusive resources

Page table

Public interface

Pre-condition
- P
- Q

Post-condition
- P'
- Q'

The last bit of page table flag is set to 1.
Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

uberobject 1
Compartment as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

uberobject 1

uberobject 2

The secure monitor bit is 1 (after function return)

The last bit of page table flag is set to 1 (after function return)

uberobject 1

uberobject 2
Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)

Source-level compartments

uberobject 1

uberobject 2

Sequential verification tool
Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

uberobject 1

F1
F2

Internal function

Public interface

Sequential verification tool

Compiler

uberobject 2

G1
G2

Internal function

Public interface

Sequential verification tool

Compiler

The secure monitor bit is 1 (after function return)
Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

uberobject 1

Sequential verification tool

Compiler

assembly code

Target-level compartments

uberobject 2

Sequential verification tool

Compiler

assembly code

The secure monitor bit is 1 (after function return)
Compartment as units of verification and compilation

Source-level compartments

The last bit of page table flag is set to 1 (after function return)

uberobject 1

Sequential verification tool

Compiler

assembly code

Public interface

Internal function

uberobject 2

The secure monitor bit is 1 (after function return)

Sequential verification tool

Compiler

assembly code

Public interface

Internal function

Target-level compartments

The last bit of page table flag is set to 1 (after function return)

uberobject 1

Sequential verification tool

Compiler

assembly code

Public interface

Internal function

uberobject 2

The secure monitor bit is 1 (after function return)

Sequential verification tool

Compiler

assembly code

Public interface

Internal function

Core 1

Core 2
Compartments as units of verification and compilation

- Source-level compartments
  - uberobject 1
    - F1
    - F2
    - Internal function
    - Public interface
    - Sequential verification tool
    - Compiler
    - F1
    - F2
    - assembly code
    - Public interface
  - uberobject 2
    - G1
    - G2
    - Internal function
    - Public interface
    - Sequential verification tool
    - Compiler
    - G1
    - G2
    - assembly code
    - Public interface

- Target-level compartments

- The last bit of page table flag is set to 1 (after function return)
- The secure monitor bit is 1 (after function return)

Both properties hold in any concurrent execution

- Core 1
- Core 2
Compartments as units of verification and compilation

The last bit of page table flag is set to 1 (after function return)

Source-level compartments

uberobject 1

Sequential verification tool

Compiler

assembly code

Target-level compartments

uberobject 2

The secure monitor bit is 1 (after function return)

Both properties hold in any concurrent execution

Core 1

Core 2
Outline

• **Concurrent execution** - an example

• Verify source-level guarantees

• Preserve target-level guarantees

• Using off-the-shelf tools

• Case studies

• Related work
Concurrent execution

uberobject 1

F1
F2

Public interface

Internal functions

uberobject 2

G1
G2

Public interface

Internal functions

Exclusive memory

\textit{uobj}_1.M

Exclusive memory

\textit{uobj}_2.M
Concurrent execution

uberobject 1

F1
F2

Internal functions

Public interface

uberobject 2

G1
G2

Internal functions

Public interface

Exclusive memory

\[ uobj_1 \cdot M \]

Exclusive memory

\[ uobj_2 \cdot M \]

Core 1

Core 2
Concurrent execution

uberobject 1

F1

Internal functions

F2

Public interface

uberobject 2

G1

Internal functions

G2

Public interface

Exclusive memory

\( uobj_1 \cdot M \)

Core 1

\( uobj_1 \cdot F1 \)

Core 2

\( uobj_2 \cdot G2 \)

Exclusive memory

\( uobj_2 \cdot M \)
Concurrent execution

uberobject 1

F1  Internal functions
F2  Public interface

uberobject 2

G1  Internal functions
G2  Public interface

Exclusive memory  $uobj_1.M$

Exclusive memory  $uobj_2.M$

Call

$uobj_2.G1$

$uobj_1.F1$

$uobj_2.G2$
Concurrent execution

uberobject 1

uberobject 2

Exclusive memory $uobj_1.M$

Exclusive memory $uobj_2.M$

Core 1

Core 2

Call

$uobj_1.F1$

$uobj_2.G1$
Concurrent execution

uberobject 1

F1

F2

Internal functions

Public interface

uberobject 2

G1

G2

Internal functions

Public interface

Exclusive memory

$uobj_1 \cdot M$

$uobj_2 \cdot M$

Core 1

$uobj_2 \cdot G1$

Core 2

Internal functions

Concurrent execution
Concurrent execution

uberobject 1

F1

F2

Internal functions

Public interface

uberobject 2

G1

G2

Internal functions

Public interface

Exclusive memory $uobj_1.M$

Exclusive memory $uobj_2.M$

Core 1

$uobj_2.G1$

Core 2

Internal functions

uberobject 1

uberobject 2
Concurrent execution

Uberobject 1

- F1
- F2
- Internal functions
- Public interface

Uberobject 2

- G1
- G2
- Internal functions
- Public interface

Exclusive memory
- $uobj_1 \cdot M$

Exclusive memory
- $uobj_2 \cdot M$
Concurrent execution

uberobject 1

uberobject 2

Public interface

Public interface

Internal functions

Internal functions

Exclusive memory $uobj_1.M$

Exclusive memory $uobj_2.M$

Core 1

Core 2

$uobj_1.F1$
Concurrent execution

Uberobject 1
- Internal functions
- Public interface
- Exclusive memory $uobj_1.M$
- $uobj_1.F1$

Uberobject 2
- Internal functions
- Public interface
- Exclusive memory $uobj_2.M$

Core 1
Core 2
Concurrent execution

uberobject 1

F1
Internal functions
Public interface
F2

uberobject 2

G1
Internal functions
Public interface
G2

Exclusive memory

\textit{uobj}_1\cdot M

\textit{uobj}_2\cdot M

Core 1

Core 2
Source-level guarantees via verification of each compartment
— Respecting the interface —
Requirement from a source-level compartment—Respecting the interface
Requirement from a source-level compartment—Respecting the interface

\[ uobj_1.F1 \]

\[ uobj_2.G2 \]
Requirement from a source-level compartment—Respecting the interface

**Guarantee:** Any internal step of *this uberobject* can only read from/write to its own exclusive memory.
Requirement from a source-level compartment—Respecting the interface

**Guarantee:** Any internal step of *this uberobject* can only read from/write to its own exclusive memory.

**Rely:** Any internal step of *other uberobjects* will never read from/write to this uberobject’s exclusive memory.
Requirement from a source-level compartment—Respecting the interface

**Guarantee:**

Internal steps

\[ \sigma \xrightarrow{\delta} \sigma' \]

\[ \sigma_{\text{rest}} \quad uobj_1.M \]

\[ \delta \quad \tau \]

\[ \delta \]

\[ \sigma_{\text{rest}} \quad uobj_1.M \]

\[ \sigma \quad uobj_1.F1 \]

\[ uobj_2.G2 \]
Requirement from a source-level compartment—Respecting the interface

**Guarantee:**

Internal steps

**Rely:**

Concurrent steps

\[
\sigma \\
\sigma \text{ rest} \quad uobj_1 \cdot M \\
\delta \\
\tau
\]

\[
\sigma' \\
\sigma' \text{ rest} \quad uobj_1 \cdot M \\
\delta
\]

\[
\sigma \text{ rest} \quad uobj_1 \cdot M \\
\delta \\
\tau
\]

\[
\sigma' \text{ rest} \quad uobj_1 \cdot M \\
\delta
\]

\[
\text{uobj}_1 \cdot F1 \\
\ldots
\]

\[
\text{uobj}_2 \cdot G2 \\
\ldots
\]
Verifying Pre and Post conditions—Respecting the interface

** Guarantee:  
1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.  
2. When a function in *this uberobject terminates*, its post-condition holds.
Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.
Guarantee:

1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.
Guarantee:
1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.
Verifying Pre and Post conditions—Respecting the interface

**Guarantee:**
1. If *this object calls* other uberobject’s public interfaces, it will satisfy their pre-condition.

2. When a function in *this uberobject terminates*, its post-condition holds.
Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If other objects call this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in other uberobjects terminate, their post-conditions hold.
Verifying Pre and Post conditions—Respecting the interface

Rely:
1. If *other objects call* this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in *other uberobjects terminate*, their post-conditions hold.
Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If other objects call this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in other uberobjects terminate, their post-conditions hold.
Verifying Pre and Post conditions—Respecting the interface

Rely:

1. If other objects call this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in other uberobjects terminate, their post-conditions hold.
Verifying Pre and Post conditions—Respecting the interface

Rely:
1. If other objects call this uberobject’s public interface, they will satisfy this uberobject’s pre-condition.

2. When functions in other uberobjects terminate, their post-conditions hold.
If each uberobject in a system respects the interface, then:

- In any concurrent run, the **pre-conditions upon the call** and the **post-condition upon return** hold for all functions.

- Any concurrent execution is **data race free**, i.e., no two threads access a location concurrently when at least one of the accesses is a write.
Target-level guarantees via certified compilers — Preserving the interface —
Requirement from a compiler—Preserving the interface

Source-level uberobject

Exclusive memory (Source-level) $uobj_1.M_s$

Public interface

F1

F2

Internal functions

Compiler

Target-level uberobject

Exclusive memory (Target-level) $uobj_1.M_t$

Public interface

F1

F2

assembly code
Requirement from a compiler—Preserving the interface

- Memory transformation function:

- Code transformation function:
Requirement from a compiler—Preserving the interface

- Memory transformation function:
  - **Well-defined:** Total and injective on heap locations, and map source-level heap locations to target-level heap locations.

- Code transformation function:
Requirement from a compiler—Preserving the interface

- **Memory transformation function:**
  - **Well-defined:** Total and injective on heap locations, and map source-level heap locations to target-level heap locations.

- **Code transformation function:**

![Diagram showing the transformation of source-level and target-level uberobjects through a compiler.](image-url)
Requirement from a compiler—Preserving the interface

- Memory transformation function:
  - **Well-defined:** Total and injective on heap locations, and map source-level heap locations to target-level heap locations.

- Code transformation function:
  - **Interface-preserving:** If an uobj respects the interface at the source level, then its compiled version respects the interface at the target level.
Requirement from a compiler—Preserving the interface

- Memory transformation function:
  - **Well-defined:** Total and injective on heap locations, and map source-level heap locations to target-level heap locations.

- Code transformation function:
  - **Interface-preserving:** If an uobj respects the interface at the source level, then its compiled version respects the interface at the target level.
If each source-level uberobject in a system respects the interface and all compilers are interface-preserving, then

**In any concurrent run at the target-level**, the security properties hold:

*All functions satisfy their post-conditions upon return.*
CAS-CompCert is an interface-preserving compiler.

(PLDI'2019)
Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1. The secure monitor bit is 1.

Source-level compartments

F1
F2
Internal functions: C+CASM
Public interface

G1
G2
Internal functions: C+CASM
Public interface

uberobject 1
uberobject 2
Sequential verification tool (Frama-C)

CASCompCert

Target-level compartments

F1
F2
assembly code
Public interface

G1
G2
assembly code
Public interface

Both properties hold in any concurrent execution

Core 1
Core 2
Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

Source-level compartments

- **F1**
  - Public interface
  - Internal functions: C+CASM

- **F2**
  - Public interface

uberobject 1

Sequential verification tool (Frama-C)

CASCompCert

uberobject 2

- **G1**
  - Public interface
  - Internal functions: C+CASM

- **G2**
  - Public interface

Both properties hold in any concurrent execution

Target-level compartments

- **F1**
  - Public interface
  - assembly code

- **F2**
  - Public interface
  - assembly code

- **G1**
  - Public interface
  - assembly code

- **G2**
  - Public interface
  - assembly code

uberobject 1

uberobject 2

A1: DSL semantics accurately reflect the assembly semantics
Our proposed tool-chain and its assumptions

Source-level compartments

- **F1**
- **G1**

Internal functions: C+CASM

**F2**

Public interface

uberobject 1

Sequential verification tool (Frama-C)

CASCompCert

uberobject 2

Sequential verification tool (Frama-C)

CASCompCert

Target-level compartments

- **F1**
- **G1**

assembly code

**F2**

Public interface

**G2**

assembly code

Public interface

**Core 1**

**Core 2**

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

Both properties hold in any concurrent execution
Our proposed tool-chain and its assumptions

The last bit of page table flag is set to 1.

The secure monitor bit is 1.

Source-level compartments

Sequential verification tool (Frama-C)

Internal functions: C+CASM

Public interface

F1

F2

uberobject 1

uberobject 2

A1: DSL semantics accurately reflect the assembly semantics

A2: C verifier's logic is sound, it only verifies correct predicates

A3: C semantics used by the C analysis tool and the CASCompCert compiler agree.

Target-level compartments

Sequential verification tool (Frama-C)

Internal functions: C+CASM

Public interface

G1

G2

assembly code

assembly code

F1

F2

G1

G2

F2

G1

G2

Both properties hold in any concurrent execution

Core 1

Core 2
Case studies

➡ UberXMHF TEE: Open source microhypervisor TEE (x86 32-bit hardware)

• An execution environment for an untrusted OS

• Verify the security property of guest memory separation: page table permissions bit is set correctly.

➡ Trustzone TEE: A light-weight open-source Trustzone TEE (ARM 32-bit)

• An execution environment for a simple guest OS running at the highest privilege level

• Verify correct setup to get guest memory separation: the secure monitor mode is set correctly.
Related work

➡ Verified TEEs
   • Sel4 - S&P’2013
   • CertiKOS - USENIX OSDI’2016
   • XMHF - S&P ‘2013
   • uberXMHF - USENIX Security ’2016
   • Security Microvisor - TDSCM ‘2019
   • Contiki - DDECS ‘2015

➡ Certified compilers:
   • CASCompCert - PLDI’2019 , ...

➡ Compartmentalization:
   • Secure Compartmentalizing compilation (SCC) - CSF’2016
   • Robustly Safe Compartmentalizing Compilation (RSCC) - CCS’2018
   • CHERI compartmentalization - SP ‘2015
Conclusion

➡ Summary:

• Compartmentalization for implementing TEEs enables us to:
  • achieve compositional verification results at the source level, and
  • leverage certified compilers to preserve the guarantees at the target level.

• Two case studies

➡ What else is in the paper?

• DSL semantics for assembly
• Interrupts
• Noninterference
Our proposed tool-chain and its assumptions

**The last bit of page table flag is set to 1 (after function return)**

Source-level compartments

- **F1**
  - Internal functions: C+CASM
  - **G1**
    - Internal functions: C+CASM
  - **G2**

Public interface

Sequential verification tool (Frama-C)

CASCompCert

assembly code

**uberobject 1**

- **F1**
- **F2**

A1: DSL semantics accurately reflect the assembly semantics

**uberobject 2**

- **G1**
- **G2**

A2: C verifier’s logic is sound, it only verifies correct predicates

**Target-level compartments**

- **Core 1**
- **Core 2**

Both properties hold in any concurrent execution

A3: C semantics used by the C analysis tool and the CASCompCert compiler agree.

The secure monitor bit is 1 (after function return)

- **Core 1**
- **Core 2**

The last bit of page table flag is set to 1 (after function return)

The secure monitor bit is 1 (after function return)