Reasoning about enclaved execution and attestation in Cerise

Thomas Van Strydonck
Dominique Devriese

KU Leuven
Enclaves are like black boxes in memory
Attestation: authentication makes enclaves useful

What's in the box?!

Widevine is running
Attestation: authentication makes enclaves useful

What’s in the box?!
How do we **formalize** the security guarantees obtained from attestation?
Concretely: **flexible** enclaved execution system

**CHERI-TrEE**

CHERI-TrEE: Flexible enclaves on capability machines

Thomas Van Strydonck*, Job Noorman*, Jennifer Jackson¹, Leonardo Alves Dias¹
Robin Vanderstraeten¹, David Oswald¹, Frank Piessens¹, Dominique Devriese*

*KU Leuven ¹University of Birmingham ²Vrije Universiteit Brussel

Abstract—This paper studies the integration of two successful hardware-supported security mechanisms: capabilities and enclaved execution. **Capabilities** are a powerful and flexible security mechanism for implementing fine-grained memory access control and compartmentalizing untrusted or buggy software components. **Capability machines** implement the concept of capabilities at the machine code level: they provide hardware support for capabilities by defining an instruction set architecture (ISA) that provides access to system memory only through memory capabilities, a kind of hardware-supported fat pointers. The ISA is designed to ensure that software can only create capabilities that represent a subset of the authority that the software already holds. Hence, capabilities are a secure basis for implementing memory access control and isolation. Next to memory capabilities, capability machines can support a wide variety of other kinds of capabilities, including, for example, object capabilities that can control access to software defined objects, or sealing capabilities that can symbolically encrypt or decrypt other capabilities. Capability machines have a long history [1], but have gained significant momentum over the last decade with, for instance, the development of the CHERI system [2].
How do we formalize the security guarantees obtained from local attestation?

Cerise
Overview

- Building Blocks of Enclaved Execution
- Formal Reasoning
- Status & Discussion
Overview

- Building Blocks of Enclaved Execution
- Formal Reasoning
- Status & Discussion
1. Exclusive access

At least during initialization
2. Controlled invocation

Entry points
3. Secure communication

Efficient
4. Attestation

- Authentication: identity $\rightarrow$ Hash
- Secure look-up (remote: reuse)
Overview

- Building Blocks of Enclaved Execution
- Formal Reasoning [WIP]
- Status & Discussion
Running example

$n : \mathbb{N}$

\[ \lambda n. \text{assert-even}(n) \]

\[ \lambda n. 2 * n \]

Protocol: “output even numbers”
Building blocks of enclaved execution

1. Exclusive access
2. Controlled invocation
3. Local secure communication
4. Local attestation

Capabilities → Cerise
Issue: pointers grant unrestricted access to memory
Hardware capabilities restrict authority
Hardware capabilities restrict authority
Hardware capabilities are unforgeable

- tag
- base
- end
- addr
- perm

R/W/X/...
Spatial memory safety: protection against adversary
Enter capabilities implement compartmentalization
Cerise: logical relations to reason about untrusted code

Universal contract: any code is safe to execute ($\mathcal{E}$)
Cerise: Logical relations to reason about untrusted code

\[ \forall(w) \begin{cases} 
\forall(z) \\
\forall(E, b, e, a) \\
\forall(RW/RWX, b, e, -) \\
\forall(RO/RX, b, e, -) 
\end{cases} \]
Building blocks of enclaved execution

1. Exclusive access
2. Controlled invocation
3. Local secure communication
4. Local attestation

Capabilities → Cerise
1. **Establishing exclusive access**

\[ \forall (RW/RWX, b, e, -) \triangleq \exists w, (a \leftrightarrow w) \ast \forall (w) \]

*Cerise lacks revocation...*
Establishing exclusive access: memory sweep

C_{enclave}

PCC

r_1

r_2

...
Logical layer to reason about memory sweep

Logical Memory: $L\text{Addr} \rightarrow L\text{Word}$
Logical Registers: $\text{Reg} \rightarrow L\text{Word}$

“Memory reachable from Regs and LRegs is isomorphic”

Physical Memory: $\text{Addr} \rightarrow \text{Word}$
Physical Registers: $\text{Reg} \rightarrow \text{Word}$

Chung-Kil Hur    Derek Dreyer    Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany
E-mail: {gil, dreyer, viktor}@mpi-sws.org
Logical layer to reason about memory sweep

**Logical Memory**: \( \text{LAddr} \rightarrow \text{LWord} 

**Logical Registers**: \( \text{Reg} \rightarrow \text{LWord} \)

“Memory reachable from Regs and LRegs is isomorphic”

\[
\forall (\text{RW/RWX, } b, e, -, v) \triangleq \bigstar_{a \in [b,e]} \exists w, (a, v \mapsto_{\text{log}} w) \ast \forall (w)
\]

Sweep succeeds? Mint \( a, v \mapsto_{\text{log}} w \) + rewrite in LRegs
Building blocks of enclaved execution

1. Exclusive access
2. Controlled invocation
3. Local secure communication
4. Local attestation

Sealed capabilities
Sealed capabilities implement symbolic crypto

- Encrypt _enc
- Sign _sign

Unseal(_enc)  Seal(_enc)

$C_{\text{enclave}}$
Attach a protocol to each sealing key

Logical Relations for Encryption
(Extended Abstract)

Eijiro Sumii
University of Tokyo
sumii@saul.cia.upenn.edu

Benjamin C. Pierce
University of Pennsylvania
bcpierce@cis.upenn.edu

Abstract

The theory of relational parametricity and its logical relations proof technique are powerful tools for reasoning about information hiding in the polymorphic λ-calculus. We investigate the application of these tools in the security domain by defining a cryptographic λ-calculus—an extension of the standard simply typed λ-calculus with primitives for encryption, decryption, and key generation—and introducing logical relations for this calculus that can be used to prove behavioral equivalences between programs that rely on encryption.

We illustrate the framework by encoding some simple security protocols, including the Needham-Schroeder public-key protocol. We give a natural account of the well-known attack on the original protocol and a straightforward proof that the improved variant of the protocol is secure.

programming languages—the concept of relational parametricity [23] and its accompanying logical relations proof method—in the domain of security protocols.

We begin by defining a cryptographic λ-calculus, an extension of the ordinary simply typed λ-calculus with primitives for encryption, decryption, and key generation. (One can imagine a large family of different cryptographic λ-calculi, each based on a different set of encryption primitives. For the present study, we use the simplest member of this family—the one where the primitives are assumed to provide perfect shared-key encryption.) This calculus offers a suitable mix of structures for our investigation: encryption primitives, since our goal is to reason about programs from the security domain, together with the type structure on which logical relations are built. We now proceed in three steps:

1. We show how some simple security protocols can

\[ \lambda n. 2 * n \]

\[ P_{sign}(c : Cap) = \exists n. c \mapsto 2 * n \]
Resource algebra for protocols

\[ \text{CanAlloc } k \quad =\ast \quad \text{Pred } k \ P \]

\[ \text{Pred } k \ P \ -\ast \text{Pred } k \ P' \ -\ast \ (\forall \ x. \triangleright (P \ x \equiv P' \ x)) \]
Reasoning about sealed capabilities

∀(sealed(w,k)) =
( ∃ P. Persistent P * Pred k P * ▷ P w)
How does client *know* that $P_{\text{sign}}$ is used?

\[ \lambda n. \text{assert\_even}(n) \]

\[ \text{assert } \mapsto \text{True} \]

\[ P_{\text{sign}}(c : \text{Cap}) = \exists n. c \mapsto 2 \times n \]
Building blocks of enclaved execution

1. Exclusive access
2. Controlled invocation
3. Local secure communication
4. Local attestation
Operational aspects of attestation

\[ l = \text{hash}(C_{\text{enclave}}) \]

TCB

\[ l \quad \text{enc}, \text{sign} \]

\[ \text{Id} \]
Reasoning about attestation: establish $P_{\text{sign}}$

- **Identity(DE) = l** (*Collision-free*)
- **Lookup(ksign) = l**
- **Pred(ksign,$P_{\text{sign}}$)**
- **$\forall$(sealed(w,ksign)) = ( $\exists$ P. Persistent P $\ast$ Pred ksign P $\ast$ $\triangleright$ P w)**

**custom_Psign :**

- Identity $\Rightarrow$ (Word $\rightarrow$ iProp)
- **custom_Psign(l) = $P_{\text{sign}}$**

(...)$\forall$ enclaves. (Lookup(k) = l $\ast$
- custom_Psign(l) = P) $\ast$ Pred(k,P)
Reasoning about attestation: initialize enclave

\[ I_{\text{new}} \in \text{dom}(\text{custom}_P\text{sign}) \]

\[ P_{\text{sign}} = \text{custom}_P\text{sign}(I) \]

\[ P_{\text{sign}} = \top \]

\[ c : \text{Cap} \mapsto \exists n. c \mapsto 2 \times n \]

\[ \text{assert}_{\text{even}}(2 \times n) \mapsto \text{True} \]

\[ \text{assert}_{\text{even}}(2 \times n) \]

\[ P_{\text{sign}} = \text{custom}_P\text{sign}(I) \]

\[ P_{\text{sign}} = \top \]

\[ \forall \text{enclaves}. (\text{Lookup}(k) = I \ast \text{custom}_P\text{sign}(I) = P) \ast \text{Pred}(k, P) \]

\[ \text{custom}_P\text{sign} : \text{Identity} \rightarrow (\text{Word} \rightarrow \text{iProp}) \]

\[ \text{custom}_P\text{sign}(I) = P_{\text{sign}} \]
Overview

- Building Blocks of Enclaved Execution
- Formal Reasoning
- Status & Discussion
Status of reasoning about enclaved execution

1. Exclusive access  WIP (Memory Sweep)
2. Controlled invocation  ✔
3. Local attestation  WIP
4. Local secure communication  ✔
Future work

•  ←←Examples→→
  ○  Multiple enclaves/protocols, caller attestation
  ○  Binary Cerise: confidentiality
  ○  Verify interrupts

•  Remote case (Asserts)
  ○  Probabilistic model?
  ○  Hash function

•  Generalize?
  ○  Keystone
  ○  Sancus