Keystone
An Open Framework for Architecting Trusted Execution Environments

Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, Dawn Song

Presented by Miles Dai
6.888, Fall 2020
Motivation

Trusted Execution Environments are rigid and uncustomizable.

Existing solutions inherit the underlying design limitations:
• Intel SGX: large software stack
• AMD SEV: large TCB
• ARM TrustZone: not enough domains
Overview
Overview – **Trusted Hardware**

- **Hardware-restricted physical memory access (PMP)**
- **Source of randomness**
- **Root of trust**
Overview – Security Monitor

- Enforce memory isolation
- Implements enclave lifecycle
- Interrupts and Exceptions
- TEE Primitives
Overview – **Runtime**

- Virtual memory management
- Communication outside the enclave (syscalls, IPC, etc.)
- Multithreading
Threat Model

4 Identified Attacker Models
• Physical
• Software
• Side-channel (cache, timing, control)
• Denial-of-Service

What’s Trusted?
• Trusted PMP spec and hardware implementation
• Trust SM, RT, and eapps (after verification)
Threat Model

What’s not covered (natively)?
• Denial-of-Service: The OS can DoS enclaves
• Speculative Execution
• Timing SC*
• Off-chip component SC*
• Non-interference for SM API (SBI)

*Keystone offloads protections for “non-traditional” attacks to RT and SM implementation as well as hardware protections.
Discussion

What are some strengths and weaknesses of Keystone?
Strengths

• Enclave feature and size flexibility
• Defends against entire classes of attacks
• Open source
• Portability: many design features are hardware-agnostic
• Compartmentalization
  • The SM is minimal enough to be formally verified
  • Smaller runtimes may be easier to implement correctly than one large kernel
Weaknesses

• Limited PMP registers (RISC-V currently supports 16)
• TCB comparison with LoC is a bit sketchy
• Kernels all the way down...
  • There are many assumptions made about correct implementation and design of the RT
  • “We assume that the SM, RT, and eapp are bug-free”
  • In practice, would the RTs eventually become bloated and simply evolve into small kernels?
• Communication into and out of the container takes a big performance hit
Physical Memory Protection (PMP)
Physical Memory Protection (PMP)

![Figure 3.27: PMP configuration register format.](image)

<table>
<thead>
<tr>
<th>A</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
<td>Null region (disabled)</td>
</tr>
<tr>
<td>1</td>
<td>TOR</td>
<td>Top of range</td>
</tr>
<tr>
<td>2</td>
<td>NA4</td>
<td>Naturally aligned four-byte region</td>
</tr>
<tr>
<td>3</td>
<td>NAPOT</td>
<td>Naturally aligned power-of-two region, ≥8 bytes</td>
</tr>
</tbody>
</table>

Table 3.8: Encoding of A field in PMP configuration registers.
System Initialization

Priority

- pmp0
- pmp1
- pmp2
- ...
- pmpN

DRAM

- U/S Accessible
- Not Accessible
After the SM boots, control is transferred to the OS. How does the OS boot change the PMP configurations?
By default, the OS can access all memory, but notice that pmp0 takes precedence and prevents the OS from accessing the SM region.
Enclave Lifecycle

Creation
- Measure enclave memory
- Validates OS-initialized page table

Execution
- Starts execution at a predefined enclave entry point

Destruction
- Clear enclave memory region, return memory to OS
- SM cleans and frees all enclave resources
Enclave Creation

Priority

<table>
<thead>
<tr>
<th>pmp0</th>
<th>pmp1</th>
<th>pmp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
</tbody>
</table>

... pmpN

PMPN

111

DRAM

<table>
<thead>
<tr>
<th>SM</th>
<th>Enclave 1</th>
<th>Enclave 2</th>
</tr>
</thead>
</table>

U/S Accessible

Not Accessible
Enclave Entry

Discussion Question: What actions should be taken to enter Enclave 2?
Enclave Entry

1. Access to Enclave 2 memory is granted
2. Access to memory outside Enclave 2 is restricted
3. Access is permitted for an untrusted shared buffer if requested by the OS
Enclave Entry with Untrusted Shared Buffer

pmpN is used to allow access to an untrusted shared buffer for communication across the enclave boundary.
Security Monitor

Responsibilities:
• Setting PMP registers
• Validate enclave memory allocation and OS-provided page table
• Measures enclave in virtual memory
• Synchronizes PMP bits across cores during enclave creation

What the SM does NOT do:
• Memory allocation
• Page table setup
Security Monitor – TEE Primitives

• Secure Boot
• Secure randomness
• Remote Attestation
• Platform-specific extensions (e.g. protections from physical attackers)
  • Secure On-Chip Memory
  • Cache Partitioning
  • Dynamic Resizing
Runtime

• Supervisor capability allows for kernel-like behavior
• Memory Management
  • Virtual address space is statically mapped by default
  • RT extensions can add flexibility (e.g. support for unmapped physical memory, page swapping, page encryption/integrity protection)
• Interface with non-enclave memory: edge calls
• Multi-threading (theoretically)
Security Analysis – Protection of the Enclave

- Direct enclave memory access is protected by PMP.
- **Mapping attacks:** Page tables are located within the enclave and are managed by the (trusted) RT.
- **Syscall tampering:** RT modules can defend against Iago attacks
- **Side Channels:** enclaves share no state with the host OS.
Security Analysis – Protection of the OS

• RTs can now attack the OS since they all operate in S-Mode!
• RTs cannot access memory or modify page tables outside the enclave.
• SM performs a **complete context switch**
• Machine timer prevents DoS attack from an enclave
Security Analysis – Protection of the SM

• PMP does the heavy lifting again: access to the SM memory is disabled by the bootloader
• The SBI must be narrow
• A minimal SM allows for formal verification
Performance

- Enclave-management is dominated by initial validation and measurement.
- Multi-core PMP synchronization during enclave creation may not be scalable.
- Moving data across the boundary is slow
Discussion Questions – Enclave Design

• If everything is getting simplified, the attack surface is smaller, etc. the original complexity needs to go somewhere? Where is it?

• Why did Intel SGX and AMD TrustZone decide to go with a static enclave design when a flexible and adjustable design such as Keystone is possible?

• It seems as though we've given up on managing virtual memory outside of the enclave- is there any hope left for alternative solutions?
Discussion Questions - Application

• Is this practical given that enclave applications have to be Keystone-native, have RT support, or be partitioned applications?

• Does Keystone actually fill a necessary gap in what is currently available? Are there a significant number of programs that really need features that aren't available with plain SGX?

• Are there any security vulnerabilities introduced if a non-expert enclave programmer doesn't specify the TEE design correctly?

• Is there a motivation for a manufacturer like Intel to move to such an open source framework?