Keystone Enclave
An Open-Source Secure Enclave for RISC-V

Dayeol Lee¹,², David Kohlbrenner, Kevin Cheang¹, Cameron Rasmussen¹, Kevin Laeufer¹, Ian Fang, Akash Khosla, Chia-Che Tsai², Sanjit Seshia¹, Dawn Song²,³, and Krste Asanovic¹,²

University of California, Berkeley ※

Collaborators: Ilia Lebedev⁴, and Srinivas Devadas⁴

※All authors are affiliated with the UCB
What is a Secure Enclave?

- Trustworthy Hardware
- Applications
- User Program and Data
- OS
- Integrity
- Confidentiality
- Enclave contents
- Remote Attestation
Secure Enclave as a Cornerstone Security Primitive

● Strong security capabilities
  ○ Authenticate itself (device)
  ○ Authenticate software
  ○ Guarantee the integrity and privacy of remote execution

● A cornerstone for building new security applications
  ○ Confidential computing in the cloud (e.g., machine learning)
  ○ Secure IoT sensor network
Why do we need an Open-Source Enclave?

- Existing enclave systems are proprietary and difficult to experiment with
  - Closed-source commercial hardware (e.g., Intel SGX, ARM TrustZone)
  - Lack of good research infrastructure
- A Lot of Challenges for Enclaves
  - Hardware vulnerabilities: Intel SGX - ForeShadow (USENIX’18), AMD SEV - SEVered (EuroSec’18)
  - Side channel attacks and physical attacks
  - Important questions: do patches really fix the problem? Are there any other issues?

Open Source Design
- Provides transparency & enables high assurance
- Builds a community to help people work on the same problems
Keystone Enclave
Keystone: Open Framework for Secure Enclaves

● The First Full-Stack Open-Source Enclave for Minimal Requirements
  ○ Root of trust, security monitor, device driver, SDK, …
  ○ Memory isolation, secure bootstrapping, remote attestation, …

● Memory Isolation only with Standard RISC-V Primitives
  ○ RISC-V Privileged ISA (U-, S-, and M-mode support)
  ○ Physical Memory Protection (PMP)
  ○ Demonstrate in unmodified processors

● Open Framework: Built Modular & Portable for Easy Extension
  ○ Platform-agnostic isolated execution environment
  ○ Platform-specific threat models (cross-core side channels, untrusted external memory, etc)
  ○ Use various entropy sources/roots of trust in different platforms

github.com/keystone-enclave
Earlier Work: Sanctum

● The First Enclave Design in RISC-V ISA
  ○ V. Costan et al., USENIX Security ’16
  ○ Proof of concept in C++
    (https://github.com/pwnall/sanctum)

● Non-standard Hardware Extension
  ○ PMP was introduced in 2017 (RISC-V Priv. v1.10)

● Keystone and Sanctum
  ○ Keystone was built from scratch
  ○ Keystone shares many good practices from prior experiences of Sanctum
  ○ The primary goal of Keystone is to make an open end-to-end framework
What Hardware Do We Need?

- RISC-V Physical Memory Protection (PMP)
- RISC-V U-, S-, and M-mode
- (RISC-V) Device Gasket PMP (i.e., iopmp)
- An Entropy Source available at boot
- Root of Trust (preferably a crypto engine)
  - Measuring & signing the security monitor
  - Platform key store
- If untrusted/external DRAM – memory encryption/integrity engine (not implemented yet)
Overview of Keystone

- Manages enclaves and PMP entries
- Multicore PMP synchronization
- Remote attestation

Keystone Security Monitor (SM)

- Stored in tamper-proof hardware
- Zeroth-stage bootloader (ZSBL)
- Tamper-proof platform key store (preferably a crypto engine)

Silicon Root of Trust

- Untrusted app hosting an enclave
- Untrusted device driver
- Allocates contiguous memory
- Provides the interface to user

Host Application

- ioctl()

Operating System

- Untrusted device driver
- Allocates contiguous memory
- Provides the interface to user

Enclave Application

- The application to execute in the enclave
- Syscalls, traps,…

Enclave Runtime

- A part of the enclave running in S-mode

Trusted, Isolated

Untrusted

U-mode

S-mode

M-mode

Enclave

Trusted, Isolated

Untrusted

U-mode

S-mode

M-mode

ioctl()
Keystone Overview (Simplified)

Remote Machine

- Host Application
- Enclave Application
- Host OS
- Enclave Runtime
- Keystone Security Monitor
- Root of Trust
- PMP

You

Untrusted Network

measures
signs

measures
signs

controls

Network
Keystone Overview (Simplified)

How does PMP work?
Memory Isolation with RISC-V PMP

- **Physical Memory Protection (PMP)**
  - Special registers to control permissions of U- and S-mode accesses to a specified memory region
  - # of PMP entries can vary (e.g., default Rocket has 8)
  - Statically prioritized by the order of entry indices
  - Whitelist-based
  - Dynamically configurable by M-mode
  - Addressing modes: NAPOT (>= 4-bytes), Base/Bound

- **How Keystone uses PMP**
  - Top/bottom PMP entries are reserved for SM/OS
  - 1 PMP entry for each “active” enclave
  - NAPOT > 4KB (fragmentation / Linux buddy allocation)
Isolation via Switching PMP Permission Bits

PMP entries

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

Priority

address range

rwx permissions

S/U accessibility

- accessible
- not accessible

SM

OS

DRAM(0x80000000-)

SM Boots

OS Boots

000

111
Creating an Isolated Enclave

OS allocates a contiguous chunk of memory using `__get_free_pages()` and initializes the free pages with the enclave page table, and the enclave program (runtime + enclave application)
Creating an Isolated Enclave

SM sets PMP entry and finalizes the enclave hash

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

S/U accessibility

- accessible
- not accessible

Enclave 1 Memory

Enclave 2 Memory

OS

SM

DRAM(0x80000000-)

PMP entries

000

000

000

111
Creating an Isolated Enclave

OS can ask SM to create as many enclaves as the number of remaining PMP entries

<table>
<thead>
<tr>
<th>PMP entries</th>
<th>S/U accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmp0</td>
<td>000</td>
</tr>
<tr>
<td>pmp1</td>
<td>000</td>
</tr>
<tr>
<td>pmp2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>pmpN</td>
<td>111</td>
</tr>
</tbody>
</table>

- SM (accessible)
- Enclave 1 Memory (accessible)
- Enclave 2 Memory (not accessible)
- OS (accessible)
Executing an Enclave

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

S/U accessibility
- accessible
- not accessible

PMP entries

DRAM(0x80000000-)

OS

Enclave 1
Memory

Enclave 2
Memory

SM

Free pages

PT

RT

ELF

0x0

111

000

000
Executing an Enclave

SM flips the PMP permission bits of pmp2 and pmpN to execute Enclave 2

```
<table>
<thead>
<tr>
<th>pmp0</th>
<th>pmp1</th>
<th>pmp2</th>
<th>...</th>
<th>pmpN</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>
```

SM flips the PMP permission bits of pmp2 and pmpN to execute Enclave 2.
SM flips the PMP permission bits of pmp2 and pmpN to execute Enclave 2.

SM: System Management
Enclave 1 Memory
OS
Enclave 2 Memory

DRAM(0x80000000-)

PMP entries:
- pmp0
- pmp1
- pmp2
-...
- pmpN

S/U accessibility:
- accessible
- not accessible
The enclave can only exit by an SM SBI call. The SM flips the permissions before entering the untrusted context.

- **S/U accessibility**
  - SM: accessible
  - Enclave 1 Memory: accessible
  - OS: not accessible
  - Enclave 2 Memory: not accessible

- **PMP entries**
  - pmp0: 000
  - pmp1: 000
  - pmp2: not accessible
  - ...: not accessible
  - pmpN: 111

- **DRAM** (0x80000000-)

Enclave 1 and Enclave 2 Memory are not accessible to other enclaves and are isolated from the OS.
Destroying an Enclave

PMP entries:
- pmp0
- pmp1
- pmp2
- ...
- pmpN

S/U accessibility:
- accessible
- not accessible

Memory segments:
- SM
- Enclave 1 Memory
- OS
- DRAM (0x80000000-)

Not accessible

PMP entries accessible

Panorama Memory Protection (PMP) is a technique used to control access to memory regions in a computer system. It allows the system to protect certain parts of memory from being accessed by unauthorized processes. In the context of an enclave, PMP entries can be configured to ensure that only specific parts of the memory are accessible to the enclave, enhancing security and privacy. The diagram illustrates how PMP entries are used to assign different levels of access to different segments of memory, such as SM (Software Memory), Enclave Memory, and the OS memory space. The DRAM region is also shown, indicating the starting address of the DRAM space in hexadecimal format.
Untrusted Shared Buffer

The OS can allocate a shared buffer in OS memory.
The SM uses the last PMP entry to allow the enclave to access the buffer.
Keystone Overview Revisited

Remote Machine

- Host Application
- Enclave Application
- Host OS
- Enclave Runtime
- Keystone Security Monitor
- Root of Trust
- PMP

What is a Runtime?

Untrusted Network

You

Measures

Signs

Root of Trust

PMP

Controls

Measures

Signs

Untrusted Network
S-Mode Enclave Runtime

- Provides Kernel-like Functionality
  - Syscalls, traps
  - thread and page table management

- Useful Layer of Abstraction
  - Least privilege of U-mode code
  - Additional functionality without complicating the SM
  - SM < 2K LoC + 5K LoC crypto lib.

- Reusability
  - Compatible with multiple user programs
  - Can act as a shield system (e.g., Haven, Graphene) in SGX
Keystone Overview Revisited

Remote Machine

- Host Application
- Enclave Application
- Host OS
- Enclave Runtime
- Keystone Security Monitor
- Root of Trust

You

Untrusted Network

measures signs

How to implement?
Silicon Root of Trust

- Tamper-proof hardware that cryptographically hashes the security monitor, provisions an attestation key, and signs them with device’s secret key.

- Various ways to implement the root of trust
  - Various entropy sources, various platform key store, and implementation of the crypto engine

- Keystone uses Sanctum’s root of trust which uses ECDSA and SHA-3
Keystone Overview Revisited

How does the enclave authenticate itself and create a secure channel?
Remote Attestation

- SM measures the enclave upon enclave creation
- Enclave may bind a key to the enclave report
- SM signs the enclave report and hands it (+ SM report) to the user
Project Status

● Testable in Various Platforms
  ○ Latest RISC-V QEMU: functionality test, development
  ○ Latest FireSim (v1.4.0): performance analysis, hardware modification
  ○ SiFive Unleashed: runs on a real quadcore in-order processor!

● Ongoing Efforts
  ○ Formal verification of PMP-based security monitor
  ○ Mitigating cache side-channel attacks using platform features

● Contributions Needed!
  ○ Building software stack: more use cases, libraries, edge compiler, …
  ○ Adding software/hardware extensions
    e.g., demand paging, memory encryption/integrity, multithreading, CMA integration, …
Project Links

● Deployment:
  ○ QEMU: https://github.com/keystone-enclave/keystone
  ○ FireSim: https://github.com/keystone-enclave/keystone-firesim
  ○ SiFive Unleashed: https://github.com/keystone-enclave/keystone-hifive-unleashed

● Keystone Repository:
  ○ Keystone-SDK: https://github.com/keystone-enclave/keystone-sdk
  ○ Device Driver: https://github.com/keystone-enclave/riscv-linux
  ○ Security Monitor: https://github.com/keystone-enclave/riscv-pk
  ○ A Simple Runtime: https://github.com/keystone-enclave/keystone-runtime
  ○ Demo: https://github.com/keystone-enclave/keystone-demo

● Documentation (more coming):
  ○ Website/Blog: https://keystone-enclave.org
  ○ Development Docs: https://docs.keystone-enclave.org
Demo
A Remote Enclave with Secure Channel

- SiFive Unleashed board + simulated non-standard hardware
  - Root of trust: Modified FU540 FSBL with hard-coded device key
- Successfully ported libsodium for ECDH Key Exchange

```
x86_64

Trusted First Party

verify(report);
establish_channel();
send/receive data;
```

Remote machine (SiFive Unleashed Board)

```
Untrusted Host

Shared Buffer

report w/ DH

encrypted data

report w/ DH

encrypted data

Enclave

WordCount

Data

libsodium

attestation

Security Monitor
```
Conclusion

Keystone: an Open-Source Full-Stack Enclave for RISC-V
- Runs on standard RISC-V cores
- Modular design for better extensibility & portability

Use Cases
- Secure hardware research (e.g., LLC side-channel defense w/ way partitioning + PMP)
- Building secure systems (e.g., Secure IoT network)

Opens up Research Opportunities around Hardware Security
- Formal Verification of PMP and Security Monitor Implementation
- Performance Analysis
- Defending Side Channels & Physical Attacks
- Multi-level Security (MLS) for Sensitive Data Analytics
Thank You!

Dayeol Lee (dayeol@Berkeley.edu)
David Kohlbrenner (dkohlbre@berkeley.edu)
Forum (keystone-enclave@googlegroups.com)