École Polytechnique Fédérale de Lausanne

Master Semester Project

Porting the Keystone Security Monitor to Miralis

Frederic Khayat

Professor: Prof. Edouard Bugnion
Supervisor: Charly Castes

January 10, 2025

Contents

Abstract
This report outlines the integration of Keystone, an open-source Trusted Execution Environment (TEE) framework, with Miralis, a firmware virtualizer. This integration addresses a critical challenge in modern systems: the tension between running security-sensitive code and firmware at the highest privilege level.

Keystone traditionally requires full trust in the entire firmware codebase, which introduces potential security vulnerabilities. To mitigate this risk, we propose porting Keystone to function as a Miralis security policy. This approach removes the need to trust firmware, thereby reducing the trusted computing base (TCB). We present a working implementation in Rust that supports key Keystone features such as enclave creation, execution, and memory protection. Our work demonstrates the feasibility of running Keystone alongside virtualized firmware, offering a more secure foundation for RISC-V-based trusted execution environments.

1 Introduction

The increasing adoption of RISC-V, an open and extensible instruction set architecture (ISA), has driven significant interest in building secure and customizable systems. This report studies how Keystone, an open-source framework for creating customizable TEEs on RISC-V platforms, can be integrated into Miralis, a system designed to minimize the TCB.

In RISC-V systems, multiple components require the highest privilege level to perform their functions. Firmware, for instance, operates at this level to execute various privileged instructions critical for system functionality. At the same time, security-related software like Keystone depends on these privileges to provide security-enforcing features. However, this dual requirement creates a conflict: while security software leverages high privileges for enforcing security, the firmware's extensive scope increases the TCB, which conflicts with the goal of minimizing the attack surface. To address this limitation, Miralis is designed to separate the non-security components of the firmware from the TCB by running them with lower privileges. This separation reduces the TCB size while preserving the full feature set of the firmware.

The report begins with an overview of the foundational components, highlighting their individual roles in the system. We then delve into the technical details of porting Keystone to Miralis, focusing on how

the security monitor and Physical Memory Protection (PMP) are adapted to work with Miralis.

2 Background

This section starts by introducing key RISC-V concepts to build a solid foundation. It then covers **OpenSBI**, the core firmware, followed by **Keystone**, a security monitor built on OpenSBI, and **Miralis**, which enhances the system by minimizing the trusted computing base. We conclude the section with an overview of the threat model.

2.1 RISC-V concepts

RISC-V is an open-source Instruction Set Architecture (ISA) that follows the reduced instruction set computing principles, offering a modular platform for processor design without proprietary restrictions.

It implements a hierarchical privilege model with three main modes of execution 1:

- **Machine Mode (M-mode)** is the most privileged level with full hardware control and is mandatory in all implementations, it is the level at which the firmware runs
- **Supervisor Mode (S-mode)** provides support for operating systems through features like virtual memory management
- **User Mode (U-mode)** is the least privileged level where application code runs

Transitions between privilege modes occur through a trapping mechanism. For instance, the ecall instruction allows a lower-privileged mode to request services from a higher-privileged mode, while the corresponding mret, sret, and uret instructions handle returns to lower privilege levels. Additionally, hardware events such as timer interrupts or memory access violations can trigger transitions to higher privilege modes where the appropriate handler code is executed.

Code run in M-mode is usually inherently trusted and therefore part of the TCB. It has low-level access to the machine implementation and can be used to manage Physical Memory Protection (PMP), a mechanism that allows defining memory regions with specific access permissions. The PMP process will be detailed later in subsection 3.2.

2.2 OpenSBI

OpenSBI[2] (Open Source Supervisor Binary Interface) is a firmware that bridges the hardware and S-mode software (e.g., operating systems), as shown in Figure 1. It implements the RISC-V SBI specification[4] and is designed for extensibility, allowing adaptation to specific hardware configurations. OpenSBI can emulate unsupported instructions, handle hardware-level initialization, and act as an intermediary to perform privileged operations in M-mode.

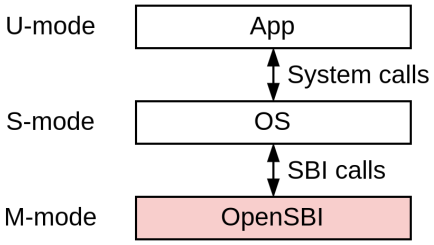

Figure 1: Typical RISC-V system running with an OS and OpenSBI

2.3 Keystone

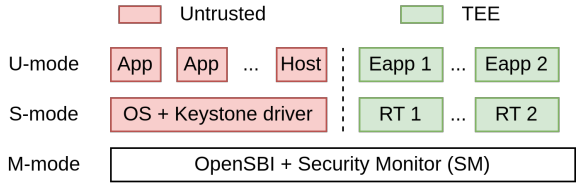

Figure 2: Keystone system with host processes, untrusted OS, security monitor, and multiple enclaves (each with a runtime and an Eapp)

Keystone[5] is an open-source project designed to create customizable TEEs called enclaves on RISC-V systems. It leverages physical memory protection (PMP) registers to secure segments of physical memory. The system involves several components, as depicted in Figure 2:

- **Host:** A U-mode application that creates and manages enclaves.
- **Keystone Driver:** A kernel module ensuring communication between the host and the security monitor. It also coordinates with the OS to allocate memory for the enclave
- **Security Monitor (SM):** An M-mode OpenSBI extension that provides functions for enclave operations such as creation, execution, and attestation.
- **Enclave:** A protected region of the physical memory, typically comprising a Runtime (RT) and an Enclave Application (Eapp).
- **Runtime (RT):** An S-mode executable within the enclave. It communicates with the SM, mediates host communication via shared memory, and provides essential S-mode functionalities for the Eapp (e.g., virtual memory). It acts as a minimal secure OS for the Eapp.
- **Enclave application (Eapp):** A U-mode executable within the enclave that communicates directly with the RT.

For example, to create an enclave with a custom Eapp and Runtime (RT), the user must first instantiate a host application to manage the enclave. The host is responsible for specifying parameters such as the size of the enclave memory region, which it communicates to the Keystone driver. The driver then allocates the required memory, loads the RT and Eapp into it, and notifies the Security Monitor (SM). The SM secures the enclave memory using RISC-V Physical Memory Protection (PMP) and can now run the enclave by switching contexts.

Although multiple components are involved in this process, developers typically only need to implement the Host and the Eapp. The Keystone driver, SM, and RTs are reusable, simplifying the development workflow.

Another advantage of Keystone is its ability to be easily extended with plugins to provide additional security guarantees. For example, the Keystone developers have proposed several plugins designed to enhance security. One such plugin is a secure On-Chip memory extension, which safeguards against physical attackers with access to the DRAM by ensuring that the code or data never leaves the chip package. Another proposal is a Cache partitioning extension, which mitigates the risk of cache side-channel attacks.

2.4 Miralis

While implementing Keystone as an OpenSBI extension is feasible, it results in a dual role for the firmware. It must address both security-critical functions (e.g., enabling TEEs) and vendor-specific hard-

ware management. This dual responsibility creates a fundamental conflict: while minimizing M-mode software is desirable to reduce the Trusted Computing Base (TCB), the inclusion of extensive firmware features inevitably expands the TCB.

Miralis[1] resolves this issue by limiting M-mode execution to security-critical functions and running all other firmware code in U-mode. Firmware running in U-mode, commonly known as "virtualized firmware", relies on Miralis to handle privileged instructions through a trap mechanism. When the firmware executes a privileged instruction, Miralis intercepts the trap and, by default, executes the instruction on behalf of the firmware, effectively granting it near-complete access to the machine. However, Miralis allows for the installation of custom **security policies** that impose restrictions on what the firmware is permitted to do. One example of a policy would be one that protects the operating system's memory from firmware access.

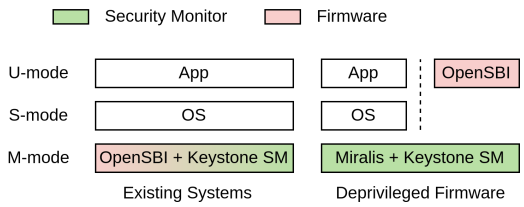

Figure 3: Comparison between running Keystone on the existing OpenSBI firmware vs running Keystone alongside Miralis

This approach offers several significant advantages:

- Compatibility: Miralis does not require modifications to existing firmware, enabling integration with current components without altering their implementation.
- Flexibility: Instead of enforcing a single universal security policy, Miralis supports the creation and deployment of diverse, custom policies tailored to specific needs. This adaptability facilitates the development of solutions for a wide range of challenges.

2.5 Threat Model

The Keystone Security Monitor trusts the PMP specification as well as the PMP and RISC-V hardware implementation to be bug-free. It also trusts that the firmware will not interfere with its operations.

If Miralis is used, the Keystone SM will no longer have to trust the firmware as it will be virtualized. However, Keystone will now have to trust Miralis instead. We argue that trusting Miralis is safer than trusting the firmware for the following reasons:

- The size of Miralis is smaller than most firmware, which reduces the TCB.
- The code of Miralis is open-source, while the code of the firmware might not be.
- Miralis is designed with modern security practices in mind (for instance, it was written in Rust, a relatively safe programming language).

Finally, other threat models such as side-channel and physical attacks are out of scope but can be mitigated with additional hardening.

3 Porting Keystone to Miralis

We will now explain how Keystone can be ported to Miralis. As specified before, Keystone is a large project consisting of multiple components, such as the Runtime, Security Monitor, and Eapp. Among

these, only the Security Monitor operates in M-mode, making it the sole component that needs to be adapted for Miralis. All other components can remain unchanged and reused without modification.

3.1 Trap Handling

The Keystone Security Monitor currently functions as an OpenSBI extension. To understand the implications of this setup, we first review how the SBI extension system operates.

SBI defines a set of standardized functions that the S-mode can invoke. These functions are organized into extensions, where each extension groups related functionality. Extensions enable modularity, allowing different features to be added or omitted based on the hardware or software requirements. These extensions can either be standardized by the RISC-V Foundation, addressing common needs like timer management, or be vendor-specific like the Keystone extension. In either case, SBI functions are invoked using the 'ecall' instruction, with parameters passed through registers:

- **Register a7:** Contains the extension ID: a unique identifier used to distinguish between all the available extensions (ex: 0x08424b45 is the extension ID of Keystone).
- **Register a6:** Contains the ID of the specific function to call from the set of functions offered by the extension (ex: 2001 is the ID of the Keystone function used to create an enclave).
- **Registers (a0, ... a5):** Hold the function's arguments.

When the Security Monitor registers its extension ID with SBI, any ecall matching this ID is routed to the Keystone security monitor, as shown in Figures 4 and 5.

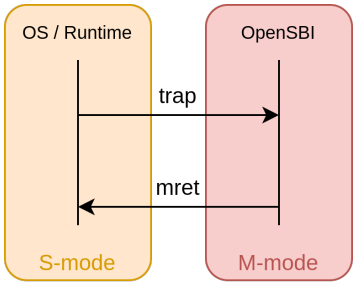

Figure 4: Standard SBI call

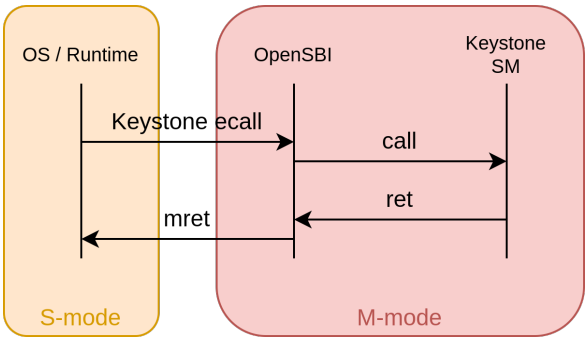

Figure 5: Keystone SBI call

However, porting Keystone to Miralis requires changing this protocol. Since OpenSBI operates in U-mode under Miralis, Keystone should no longer be implemented as an OpenSBI extension, as the goal is to exclude OpenSBI from the TCB. Instead, Keystone must operate in M-mode as a **Miralis security policy**. In this setup, Miralis intercepts ecall instructions and determines their handling:

- If an ecall does not match Keystone's extension ID, the regular Miralis trap handling flow is followed (Figure 6).
- For Keystone-specific ecalls, Miralis invokes Keystone's security policy, allowing it to execute enclave-related operations securely without interacting with OpenSBI (Figure 7).

This approach ensures that Keystone's enclave management operates securely in M-mode without relying on U-mode firmware.

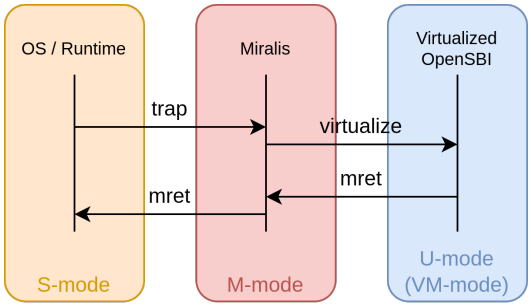

Figure 6: Miralis trap handling if the trap is not an ecall or if the extension ID does not correspond to Keystone

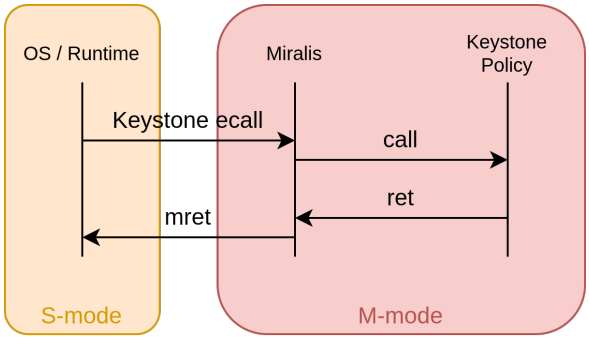

Figure 7: Miralis executing the Keystone policy when the trap is an ecall whose extension ID corresponds to Keystone

3.2 PMP

Keystone leverages the RISC-V physical memory protection (PMP) mechanism to secure and protect enclaves. The PMP defines a finite number of PMP regions that can be individually configured to enforce access permissions (Read, Write, and Execute) to a range of addresses in memory. Each PMP region is configurable using config (pmpcfg) and address (pmpaddr) registers, whose functionality is detailed in the RISC-V specification[3].

However, to understand how the Keystone SM utilizes PMP, we need to know several properties of RISC-V PMP:

1. **Prioritized by Index:** PMP entries are statically prioritized by their index, with a check stopping at the highest priority matching. Indices run from 0..N (where N is platform defined), with 0 having the highest priority, and N having the lowest. Thus, the access permissions to a physical address should be of the lowest-index PMP entry among the matched ones.
2. **Default Deny:** If no PMP entry matches with an address, the memory access will be rejected by default.
3. **Dynamically Configurable:** M-mode can write to PMP CSRs during runtime to define PMP entries dynamically.

At the very beginning of the boot process, physical memory is not accessible by U- or S-modes because of property 2 (Listing 1).

During its initialization, Miralis configures PMP so that the first PMP entry denies all access to Miralis' memory, while the last PMP entry grants full access to the rest of the memory, allowing the OS to access the rest of the memory and start booting. Because of property 1, the net result will be as shown in Listing 2.

When the Keystone policy creates an enclave, it will assign a PMP entry to protect the enclave's memory from other U-/S-mode software (Listing 3).

When the Keystone policy executes the enclave, it flips the permission bits of the enclave's PMP entry and the last PMP entry as shown in Listing 4. This will allow the enclave to execute in U-/S- mode while protecting the rest of the physical memory. In addition, Keystone allows the allocation of an additional contiguous memory region in the OS memory space to enable communication between the enclave and the OS. The contiguous memory region is called the untrusted shared buffer.

If the enclave stops executing and cedes control to either the OS or OpenSBI (for

example after a timer interrupt), the Keystone policy will flip back the permission bits to make the enclave inaccessible.

Note: If the system supports fewer than 16 PMP registers, the OS and firmware memory cannot be protected during enclave execution, as the limited number of PMP registers is insufficient to guard all components simultaneously.

```
We denote the i'th PMP entries as pmp[i].
-: inaccessible (NO_PERM), =: accessible (ALL_PERM)

pmp[0:N]     |                                          | : undefined
net result   |------------------------------------------|
```

Listing 1: PMP protection at boot time

```
 -: inaccessible (NO_PERM), =: accessible (ALL_PERM)

pmp[0]       |-------|                                  | : Miralis
pmp[others]  |                                          | : undefined
pmp[N]       |==========================================| : OS memory
net result   |-------|==================================|
```

Listing 2: PMP protection after Miralis initialization

```
-: inaccessible (NO_PERM), =: accessible (ALL_PERM)

pmp[0]       |-------|                                  | : Miralis
pmp[1]       |               |---------|                | : Enclave
pmp[others]  |                                          | : undefined
pmp[N]       |==========================================| : OS memory
net result   |-------|======|---------|================|
```

Listing 3: PMP protection when the enclave exists but is not running

```
-: inaccessible (NO_PERM), =: accessible (ALL_PERM)

pmp[0]       |-------|                                  | : Miralis
pmp[1]       |               |=========|                | : Enclave
pmp[others]  |                                          | : undefined
pmp[N]       |                                    |==|  | : Buffer
net result   |-------|======|=========|-------|==|----|
```

Listing 4: PMP protection while running the enclave

Note that while the virtualized firmware can define its own PMP regions, Miralis ensures these regions are assigned a lower priority than those used by Keystone. This guarantees that the enclaves remain securely protected, as the higher-priority

PMP regions defined by Keystone will take precedence over the firmware's regions.

With this protection mechanism, only Miralis itself can access the enclave. All other executables, including the operating system, virtualized firmware, and user-space applications, are unable to access the enclave's memory as it is protected via the PMP registers. The PMP registers themselves remain protected, as only M-mode software (i.e. Miralis) can modify them.

4 Practical implementation

In this section, we describe the practical work involved in porting Keystone to Miralis.

4.1 Keystone Security Policy in Rust

Miralis is implemented in Rust, a modern programming language designed with safety in mind. Consequently, the Keystone Security Monitor was rewritten in Rust to align with Miralis' architecture and leverage the language's safety over C, the language of the original implementation.

The ported Keystone implementation on Miralis supports the following features:

4.1.1 Supported Features

- Enclave creation, destruction, startup, pausing, and resumption.
- Enclave protection via RISC-V physical memory protection.

4.1.2 Unsupported Features

- **Multi-threading:** The implementation is not thread-safe when manipulating enclaves in parallel.
- **Data sealing & attestation:** Not implemented.
- **Platform-specific plugins:** The FU540-specific cache partitioning plugin to mitigate cache side-channel attacks is not included.
- **OS memory protection during enclave execution:** Not implemented due to the limited number of PMP registers available.

4.2 Automatic Build System

To facilitate the usage of Keystone with Miralis, an automatic build system was created using GitHub Actions, Makefiles, and patch files. This system automates the building and preparation of all required components, making them ready for execution on Miralis.

The components outputted by the build process are:

- A Linux image compatible with the Keystone driver.
- An OpenSBI binary that jumps to that Linux image.
- A modified version of the iozone benchmark that can execute from inside an enclave.
- A file system containing examples of Keystone hosts, runtimes, and Eapps, including the ones used to run the iozone benchmark.

4.3 Testing

Testing was conducted to verify the functionality of the Keystone security policy. An automatic test suite was implemented, to validate enclave operations such as creation, destruction, startup, pausing, resuming, and memory protection via PMP.

Additionally, the original Keystone test-suite, comprising eight tests, was reused to further validate the implementation. All tests passed except for the following ones:

- `fib-bench`: Fails because it uses the "rdcycle" instruction, which causes OpenSBI to panic.
- `data-sealing` and `attestation` tests: Fail as these features are not yet implemented.

In total, over 1000 lines of code (LOC) were developed across various components. The Keystone security policy itself consisted of 450 lines of Rust code. The automatic build system and patch files contributed 260 lines of code, while the test suite was composed of 152 lines of code. Additionally, around 150 lines of code were written for supporting utilities (such as supporting the Sstc RISC-V extension).

5 Conclusion

In this report, we successfully demonstrated the integration of the Keystone framework into Miralis. Keystone provides the essential functionality for creating and managing secure enclaves. Miralis, in turn, enhances this setup by minimizing the TCB, ensuring that firmware code is isolated from the enclaves and cannot interfere with their operations. This combination offers a powerful and secure platform that leverages the strengths of both Keystone's enclave management and Miralis's firmware virtualization.

6 Future Work

There are several promising directions for further development. One immediate step is running enclaves with Miralis on physical hardware, which would allow for benchmarking the system and evaluating its performance impact. Implementing features such as enclave attestation and data sealing is another key area, as these would significantly extend Keystone's functionality and make it more versatile in real-world applications. Customizing the system to protect against specific threat models like side-channel attacks is also an important aspect. These efforts would refine the integration of Keystone with Miralis and unlock its full potential as a secure platform.

References

[1] Charly Castes, Neelu S. Kalani, Sofia Saltovskaia, Noé Terrier, Abel Vexina Wilkinson, and Edouard Bugnion. Kicking the firmware out of the tcb with the miralis virtual firmware monitor. In *Proceedings of the 2nd Workshop on Kernel Isolation, Safety and Verification*, KISV '24, page 8–15, New York, NY, USA, 2024. Association for Computing Machinery.

[2] OpenSBI Community. Opensbi: Open source implementation of the risc-v supervisor binary interface, 2025.

[3] RISC-V Foundation. The risc-v instruction set manual: Volume ii: Privileged architecture, 2024. Version 20240411.

[4] RISC-V Foundation. The risc-v sbi specification, 2024. Version 2.0.

[5] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.