Physical Memory Please: Practical Memory-Aliasing Attacks on RISC-V PMP

Antonis Louka[1], Jesse De Meulemeester[2], Steven Keuchel[1], Ingrid Verbauwhede[2], and Jo Van Bulck[1]

[1]*DistriNet, KU Leuven,* [2]*COSIC, KU Leuven*

Abstract

Recent years have seen a surge in security architectures built on the open RISC-V instruction set architecture. A key enabler of this trend has been the standardized Physical Memory Protection (PMP) extension, safeguarding critical firmware and forming the foundation for, amongst others, versatile Trusted Execution Environments (TEEs). However, while production TEEs on popular x86 and Arm platforms have undergone extensive security vetting, emerging RISC-V TEEs have received far less scrutiny.

This paper studies the impact of recent memory-aliasing attacks, originally demonstrated on x86, within the RISC-V ecosystem. We show that a practical memory-aliasing setup can fully bypass PMP-based isolation, enabling arbitrary read and write access to protected memory regions. Using this primitive, we demonstrate end-to-end attacks on the popular RISC-V Keystone TEE, achieving full enclave memory disclosure and, ultimately, completely undermining remote attestation guarantees by extracting the long-term platform measurement key. Leveraging open-source RISC-V firmware, we further develop a practical mitigation that detects rogue DIMM configurations at boot time, effectively preventing software-based memory-aliasing attacks. Our findings nuance the trust in PMP-based isolation and highlight how microarchitectural attack vectors from established architectures like x86 can translate to emerging RISC-V settings.

1 Introduction

RISC-V is an emerging open-source Instruction Set Architecture (ISA) that is gaining increasing traction in both academia and industry. While early RISC-V deployments primarily targeted resource-constrained embedded systems, recent developments have led to the adoption of high-performance commercial RISC-V cores across the entire computing spectrum, including laptops [55, 65], workstations [42, 43], and servers [52, 54].

As a modern ISA, RISC-V includes standardized support for privilege levels (machine, supervisor, user) and robust memory protection mechanisms at both the physical and virtual levels. A cornerstone feature in this respect is *Physical Memory Protection (PMP)* [1], a flexible hardware primitive that enables machine mode to define access-control policies over contiguous physical address ranges. Operating independently of the operating system or hypervisor's virtual memory protections, PMP is instrumental in safeguarding high-privilege memory regions such as firmware. For example, the key OpenSBI firmware leverages PMP to protect its own memory and critical memory-mapped I/O devices from potentially buggy or malicious operating system software. Furthermore, due to its flexibility and standardized availability across RISC-V platforms, PMP has become a foundational building block for a wide range of *Trusted Execution Environments (TEEs)* on RISC-V [6, 7, 12, 19, 31, 33, 36, 38, 47, 64]. These TEEs typically rely on a trusted security monitor running in machine mode to configure PMP entries that isolate enclaves from the rest of the system, including the privileged operating system. Among the various RISC-V TEE implementations, Keystone [36] stands out as a popular base platform due to its open-source nature, extensibility, and modular architecture. Notably, Keystone is the only RISC-V project recognized under the Linux Foundation's Confidential Computing Consortium [10].

While TEEs have been extensively deployed and scrutinized on other architectures, such as Intel Software Guard Extensions (SGX), AMD Secure Encrypted Virtualization (SEV), and Arm TrustZone, security architectures for production RISC-V cores have only received limited architectural [62, 63] and microarchitectural [20, 29, 41] scrutiny. Recognizing PMP's pivotal role as a standardized hardware primitive for enclave and firmware isolation, formal verification efforts have focused on its architectural correctness [8], or the security guarantees that PMP provides to software [23]. However, to our knowledge, no prior work has examined PMP's resilience against microarchitectural attacks.

In this paper, we investigate the feasibility and impact of memory aliasing attacks [14, 44], recently disclosed in the context of x86-based TEEs, on the RISC-V PMP access control

mechanism. Specifically, we examine the *BadRAM* [14] primitive, a novel and low-cost microarchitectural attack that leverages limited one-time physical access to configure malicious Dynamic Random-Access Memory (DRAM) topologies. By deliberately modifying DRAM size metadata reported by individual Dual In-line Memory Modules (DIMMs) during system boot, BadRAM attackers introduce aliases in the CPU's physical address space that resolve to the same physical location on the DIMM. This aliasing effect enables bypassing of CPU-level physical access controls, ultimately compromising the confidentiality and integrity of encrypted enclave memory. While opaque firmware-level mitigations have since been deployed on Intel SGX [30] and AMD SEV [3] platforms, the applicability of BadRAM to RISC-V systems remains unexplored. This work addresses that gap by adapting BadRAM to RISC-V and evaluating its impact on PMP-based isolation and potential mitigation strategies.

Target Platform. In line with BadRAM attacks on Intel and AMD x86 platforms, we target RISC-V motherboards equipped with discrete (i.e., pluggable) DIMMs, which are becoming increasingly common on mature, higher-end RISC-V systems [15, 42, 43]. For our experiments, we use the Milk-V Pioneer board [42], featuring a high-performance 64-core Sophon SG2042 server-class CPU and four standard DDR4 DIMM slots.

We extend the Keystone framework to run natively on this platform, including support for secure boot and remote attestation. The Milk-V port developed in this work represents the first deployment of Keystone on a silicon board featuring an advanced out-of-order RISC-V CPU, enabling future microarchitectural research on RISC-V TEEs beyond the memory aliasing analysis presented in this paper.

Main Findings. We then adapt the BadRAM primitive to RISC-V and demonstrate that memory aliasing can trivially bypass PMP-based isolation after reverse-engineering the physical-to-DRAM address mapping function. By modifying the untrusted Linux memory allocator, we silently allocate aliased memory regions to Keystone enclaves, enabling arbitrary reads and writes of confidential enclave data. Critically, in an end-to-end attack on Keystone's remote attestation primitive, we leak the platform measurement key from protected machine-mode memory. This allows us to forge arbitrary attestation reports and, ultimately, dismantle trust in the Keystone RISC-V ecosystem.

To help recover trust, we investigate practical firmware countermeasures to detect BadRAM at boot time. Specifically, we extend the zero-state bootloader firmware for the Sophon SG2042 CPU with two mitigation strategies: a generic alias-scanning approach, and a highly optimized variant that leverages prior reverse-engineering insights into Sophon's proprietary DRAM address function [41] to minimize the amount of required memory accesses. Our work presents the first open-source and inspectable alternative to the opaque, proprietary firmware mitigations deployed on Intel [30] and AMD [3] platforms. We experimentally evaluate both mitigation techniques and find that the optimized version can reliably detect malicious memory configurations with negligible boot-time overhead.

Contributions. In summary, our main contributions are:

- We detail the modifications needed for native hardware support of the Keystone framework on the Milk-V Pioneer board, which features Sophgo's SG2024 SoC — a RISC-V platform with an out-of-order CPU design.
- We port a memory aliasing primitive (BadRAM) to RISC-V and demonstrate the feasibility of these attacks on RISC-V by breaking PMP and leaking protected enclave-allocated memory.
- We evaluate the impact of BadRAM attacks by breaking remote attestation checks on Keystone enclaves and leaking the Security Monitor (SM)'s secret key.
- We propose a mitigation strategy to defend against BadRAM-RISC-V attacks and discuss limitations and countermeasures.

Ethics and Open Science. To support reproducibility of our work and facilitate future research on the security analysis of RISC-V TEEs, we open-source our experimental setup, attack implementations, and mitigation prototypes at `https://github.com/dnet-tee/PMPlease`. This includes our Keystone port for the Milk-V Pioneer platform, untrusted Linux kernel attack code, firmware-level mitigations, and evaluation scenarios.

All experiments were conducted on our own local machines without involving any personal data. While Keystone explicitly aims to become a production-ready platform [34], we are not currently aware of any real-world production deployments that would necessitate responsible disclosure.

Paper Outline. The paper is organized as follows. § 2 outlines key RISC-V features and memory interface attacks. § 3 presents the problem statement, threat model, and setup, while § 4 reviews related work on RISC-V TEEs. § 5 details the Keystone port to the Milk-V Pioneer board with secure boot, and § 6 evaluates BadRAM-induced physical memory aliasing attacks on enclaves. Mitigations are discussed in § 7, and § 8 concludes the paper.

2 Background and Related Work

This section introduces the necessary background on the RISC-V ISA and boot process, as well as prior work on mem-

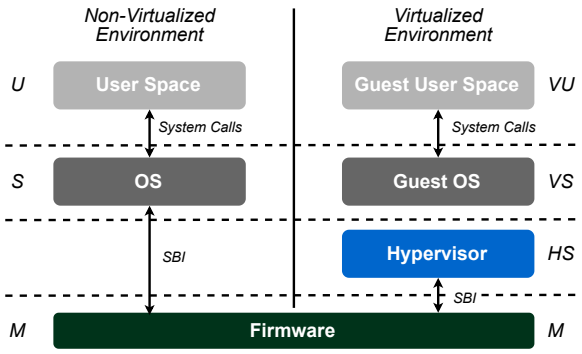

Figure 1: RISC-V processor privilege levels with (right) and without (left) virtualization support.

ory interface attacks on Intel and AMD x86 platforms.

2.1 RISC-V

RISC-V is an open-standard ISA designed with modularity and extensibility in mind. Its open nature has made it an attractive platform for both academic research and industrial development. Unlike proprietary ISAs such as x86 and Arm, RISC-V is not controlled by a single vendor but is governed by an open consortium comprising both industry and academic partners. The architecture follows a modular design philosophy, consisting of a minimal base ISA that can be extended with a rich set of optional extensions to support advanced CPU features. The RISC-V consortium oversees the standardization process, which encompasses defining extensions and maintaining compatibility across implementations.

RISC-V Privilege Levels. To support modern operating systems, the RISC-V privileged architecture [1] defines a common privilege hierarchy comprising three main modes: machine *(M)*, supervisor *(S)*, and user *(U)*, as visualized in Fig. 1 (left). Regular user applications are intended to run in U-mode, isolated by an Operating System (OS) running in S-mode. M-mode holds the highest privilege level and has unrestricted access to all system resources. M-mode is intended for low-level firmware that can provide platform-specific functionality to a general-purpose operating system running in S-mode through a standardized Supervisor Binary Interface (SBI) specification [2]. The OpenSBI [26] project provides an open-source reference implementation of M-mode SBI firmware for popular RISC-V platforms and has, furthermore, served as a basis for custom security architectures (cf. Section 4).

M-mode operates exclusively on physical addresses, whereas U- and S-modes access virtual addresses that are translated to physical addresses via a page-table data structure set up by the OS. Importantly, M-mode can impose additional read-write-execute access restrictions on physical addresses as discussed in the next subsection.

A recent addition to the RISC-V execution model is the hypervisor mode (HS-mode) [49], a privileged mode designed to virtualize S-mode and U-mode, thereby enabling efficient OS hosting on top of type-1 or type-2 hypervisors. Analogous to x86 virtualization, HS-mode introduces an additional address translation stage, from guest-physical addresses to supervisor-physical addresses, to support efficient virtualization of guest OSs. We note that HS-mode has only recently been ratified and is not yet widely available in commercial RISC-V hardware, including the Milk-V Pioneer board used in our evaluation (cf. Section 3.3).

Physical Memory Protection (PMP). To enforce isolation between privilege modes, RISC-V provides a hardware mechanism known as Physical Memory Protection (PMP) [1]. PMP allows software executing in M-mode to configure access-control policies over specific physical memory regions. PMP is controlled by a set of Control and Status Registers (CSRs) that are configurable only from M-mode and restrict access to contiguous physical memory regions from lower-privilege modes, i.e., HS-mode, S-mode, and U-mode. Access control is transparently enforced by the hardware at every memory access. PMP entries are statically prioritized and each entry is defined by a `pmpaddr` register and an 8-bit subset of a `pmpcfg` register, where the former is used to derive a contiguous physical address range, and the latter defines an addressing mode and permission bits (read, write, execute) for that region. Each RISC-V core can have up to 64 PMP entries, albeit in practice, fewer are implemented. The SG2042 CPU [58] implements 8 entries with a 2048-byte granularity.

Reflecting its critical role in enforcing security and isolation within the RISC-V ecosystem, several recent extensions build upon and further extend PMP. The ratified SMEPMP [32] extension allows M-mode to voluntarily restrict its own access to designated PMP regions, helping to mitigate confused-deputy vulnerabilities. Furthermore, IOPMP [21] (under development) extends PMP's isolation guarantees to untrusted DMA peripherals. Finally, SPMP [17] (also under development) introduces additional PMP registers that S-mode can configure to isolate U-mode tasks on low-end IoT cores lacking virtual memory support.

An exemplary use case for PMP is isolating M-mode firmware code and data from a potentially buggy or malicious operating system. This is how PMP is employed in the OpenSBI project, which additionally supports isolating system-level partitions into dedicated domains [56]. As an ISA-level primitive for configurable memory access control, PMP also serves as the foundation for RISC-V TEEs (cf. Section 4). Frameworks such as Keystone [36] leverage PMP to carve out isolated enclave regions that are inaccessible to an untrusted operating system or hypervisor. This typically involves implementing a security monitor software layer run-

ning in M-mode, which constitutes the platform's Trusted Computing Base (TCB) and is responsible for configuring PMP entries and mediating context switches to and from enclaves.

RISC-V Boot Phases. The RISC-V boot process consists of multiple stages that transition through different privilege levels [48]. The SG2042 Milk-V board [58], used in this work, includes two CPU subsystems: *(i)* the main 64-core RISC-V CPU; and *(ii)* an Arm-based System Co-Processor (SCP).

During system startup, execution begins on the SCP subsystem, which performs platform initialization. This includes configuring the PCIe topology and controllers, initializing DRAM by reading the Serial Presence Detect (SPD) chip via the I2C bus, setting up the on-chip mesh interconnect, and loading the main RISC-V CPU firmware, referred to as the Zero Stage Boot Loader (ZSBL), from either the SPI flash chip or an SD card [58]. After completing these steps, the SCP releases all 64 RISC-V cores, which then begin execution from the ZSBL.

The ZSBL, running in M-mode, performs essential hardware initialization, sets up the execution stack, and loads the next boot stage from persistent storage into memory. Control is then transferred to OpenSBI, which also operates in M-mode as system firmware, providing a standardized runtime environment and service interface between machine mode and the operating system executing in supervisor mode. Once system initialization is complete, OpenSBI hands off control to software executing in S-mode, typically a bootloader or directly the Linux kernel, which eventually transitions to U-mode for user-level execution.

2.2 Memory Interface Attacks

Confidential computing, enabled by hardware-level TEEs, aims to securely offload sensitive computations to untrusted remote platforms such as cloud providers. This threat model inherently includes physical datacenter access by rogue employees or local law enforcement. To mitigate such threats, production-grade TEEs commonly employ transparent memory encryption, protecting against straightforward attacks such as cold-boot extraction [66]. However, recent studies [9, 13, 14, 53] have demonstrated that even adversaries with limited physical access and modest resources can successfully compromise encrypted memory in production x86 TEEs from Intel and AMD. These attacks generally fall into two categories: *(i)* passive side-channel attacks on the CPU–DRAM interface using interposers; and *(ii)* active memory aliasing attacks that manipulate the physical address mapping between the CPU and DRAM.

Passive DRAM Interposers. The MemBuster attack [35] targets the (now deprecated) Intel Client SGX platform. Although Client SGX's memory encryption engine [22] provided strong cryptographic protection for memory confidentiality, integrity, and freshness, the address bus remained necessarily unencrypted. MemBuster exploits this weakness by using a commercial DDR4 interposer, costing approximately $170,000, to capture address-bus traffic and recover secrets from non-constant-time code execution.

Follow-up work by Seto et al. [53] demonstrated that such DDR4 interposer attacks can be reproduced with inexpensive, second-hand hardware costing under $1,000 by lowering DRAM speed via SPD DIMM configuration. Their WireTap attack targets Intel's newer Scalable SGX implementation for server platforms. Notably, Scalable SGX replaced Client SGX's strong memory encryption with a weaker, deterministic scheme that provides only confidentiality (without integrity or freshness) to support much larger protected memory regions, i.e., on the order of terabytes rather than megabytes [30]. As a result of this deterministic design, WireTap can exploit powerful ciphertext side channels [37] to fully compromise SGX protections. The WireTap approach has since been extended to DDR5 platforms by Chuang et al. [9].

Memory Aliasing. The BadRAM primitive, introduced by De Meulemeester et al. [14], is a practical, low-cost method for introducing physical-address aliasing on DDR4 and DDR5 memory modules. BadRAM assumes an adversary capable of modifying and reprogramming the SPD EEPROM of a connected DIMM. Such tampering can be achieved either through brief physical access or remotely via software control, e.g., by manipulating the x86 platform initialization BIOS logic or using the standard `i2c-tools` Linux utility when the DIMM's SPD is shipped unlocked. By adjusting the SPD metadata size field to report a capacity twice that of the actual physical DRAM. BadRAM causes the system to establish distinct CPU-physical address mappings that silently resolve to the same DRAM locations. This aliasing effect allows effective bypass of the CPU's physical address access-control restrictions. In practice, BadRAM has been demonstrated on AMD SEV-SNP platforms, where an attacker can corrupt or replay ciphertexts, exfiltrate secrets, and forge end-to-end attacks that undermine attestation guarantees. The issue has since been mitigated on Intel [30] and AMD [3] platforms through opaque firmware updates that attempt to detect aliasing during the boot process.

In follow-up work, De Meulemeester et al. developed BatteringRAM [13], which bypasses the boot-time firmware mitigations introduced for BadRAM using a custom, low-cost hardware interposer. This interposer requires physical access to install between the CPU and the DIMM, similar to the aforementioned WireTap [53] attack, and can be dynamically activated at runtime to induce transparent memory aliasing. By remaining disabled during the boot phase, BatteringRAM trivially circumvents firmware-based alias detection, enabling ciphertext replay and breaking both Intel Scalable SGX and AMD SEV-SNP on fully up-to-date platforms.

Ultimately, BatteringRAM and WireTap expose fundamental security-performance trade-offs in modern confidential-computing architectures that rely on scalable memory encryption without freshness protection. We discuss the implications of these physical-access attacks for the RISC-V ecosystem and our firmware mitigation in Section 7.

3 Problem Statement and Threat Model

3.1 Problem Statement

RISC-V's PMP mechanism provides hardware-enforced isolation for designated physical memory regions. On every memory access, the hardware performs a privilege check and blocks any physical memory accesses lacking the required permissions. This mechanism forms the foundation for RISC-V TEE frameworks, such as Keystone, which leverage PMP as a core architectural feature to isolate enclave memory from untrusted software.

However, PMP guarantees protection only for explicitly defined physical addresses. If the physical-to-DRAM address mapping can be manipulated or aliased at the DRAM level, PMP may protect only the nominal addresses while leaving their aliased counterparts exposed to adversarial access. In this paper, we examine the robustness of RISC-V's PMP mechanism against such memory aliasing attacks, specifically the BadRAM attack primitive, previously demonstrated only on Intel and AMD x86 architectures [14]. Our evaluation focuses on the Keystone framework [36], one of the most mature and flexible RISC-V enclave systems, though our findings generalize to any TEE design or firmware isolation guarantee that relies on PMP for physical memory isolation.

3.2 Threat Model

At the software level, we assume the standard TEE threat model, where all software components outside the enclave and security monitor are considered attacker-controlled. Concretely, we assume the adversary has elevated privileges on the target machine, *i.e.*, has full control over S-mode, including the Linux kernel. We assume all code running in M-mode, including OpenSBI, Keystone's security monitor, and any platform-specific ZSBL, is trusted and free of exploitable vulnerabilities. Notably, as our case-study attacks (cf. Section 6) target Keystone's generic attestation and enclave-loading procedures, the adversary need not have access to the source or binary of specific application enclaves.

At the hardware level, we target RISC-V boards that use discrete, non-soldered DIMMs, such as the Milk-V Pioneer [42], Milk-V Titan [43], and DeepComputing Station-V [15] platforms. As RISC-V platforms mature and gain more share in server and workstation markets, such configurations will expectedly become the norm. Following the threat model of prior BadRAM attacks on Intel and AMD x86 platforms,

Figure 2: Experimental setup showing off-the-shelf Milk-V Pioneer motherboard, including Sophon SG2042 RISC-V CPU (top, under fan) and single DDR4 DIMM (bottom) with unlocked SPD.

we assume at least one of the following capabilities: *(i)* the SPD EEPROM(s) on one or more DIMMs are not locked (a behavior observed in multiple off-the-shelf DIMM vendors [14]), allowing remote reprogramming via standard utilities; or *(ii)* the adversary can manipulate the (potentially untrusted) platform-initialization firmware that configures the memory controller; or *(iii)* the adversary gains one-time physical access to a memory module (e.g., during a data-center maintenance window or via supply-chain compromise) and can unlock or modify the DIMM's SPD contents.

Finally, at the TEE level, Keystone does not include memory encryption by default. Several proposals add memory-encryption support for Keystone, but they either demand substantial hardware changes [67] or rely on a secure on-chip scratchpad region [5] that is unavailable on our Milk-V platform. Consequently, our case-study attacks assume a standard Keystone deployment without memory encryption. While unencrypted memory buses may permit direct physical snooping, such attacks require physical access, and specialized equipment. In contrast, on platforms where the SPD is writable from software, BadRAM aliasing attacks can be introduced entirely by a privileged software adversary. Even in the absence of memory encryption, physical memory isolation mechanisms such as PMP critically depend on the correctness of the underlying memory topology and can be subverted by attackers that never directly interact with the memory bus.

If scalable memory encryption is present, our aliasing primitive would still enable ciphertext replay and corruption, similar to prior x86 attacks [13, 14]. Even for the strongest level of memory encryption, guaranteeing confidentiality, integrity, and freshness, aliasing can facilitate low-cost side-channel analysis to infer memory write events, similar in spirit to MemBuster [35], but at a fraction of its cost.

3.3 Experimental Setup

All experiments in this paper were conducted on an off-the-shelf Milk-V Pioneer board [58], depicted in Fig. 2. This board is powered by a high-end Sophon SG2042 CPU featuring 64 C920 out-of-order RISC-V cores at 2 GHz and supporting up to 128 GB of DDR4 memory. We equipped our system with a single 32 GB DDR4 memory module (MTA18ASF4G72PDZ-3G2E1).

We base ourselves on the chip vendor's (Sophgo Technologies) open-source ZSBL [61], which we modified to include support for secure boot and our mitigations, as discussed in Sections 5 and 7. Furthermore, we modified Sophgo's fork of OpenSBI v1.2 [60] to include support for the Keystone security monitor, which we also detail in Section 5.

Introducing Memory Aliases From Software. We leverage the BadRAM attack primitive to introduce aliases in the processor's physical memory view. By modifying the DIMM's SPD configuration, an adversary can create a discrepancy between the reported and true size of the installed DIMM.

While BadRAM originally required one-time physical access to the victim DIMM, we note that the System Management Bus (SMBus) to which the SPD is connected is fully exposed to software on our platform. Furthermore, unlike Intel and AMD systems, we note that the Pioneer board does not include any protections against writes to the SPD. As a result, a privileged software adversary can issue commands to an unlocked SPD chip, mounting the BadRAM attack entirely from software.

To simulate this scenario, we manually unlocked the DIMM in our system and verified that we could overwrite the SPD configuration purely from software. We mounted the BadRAM attack by doubling the reported memory size to 64 GB, which took effect after a system reboot. While we experimentally verified this technique on the Pioneer board, other RISC-V boards may not expose the SMBus in the same way, or may block writes to the SPD entirely. Note that, though required by JEDEC [27, 28], in practice, off-the-shelf memory modules do not always properly lock the base configuration blocks of the SPD [14, 39]. Furthermore, the SPD may be unlocked or modified as part of a supply-chain attack.

4 RISC-V Security Architectures

In this section, we review key efforts implementing TEEs on RISC-V and highlight the central role of PMP in enabling enclave creation and isolation. We then focus on the Keystone framework, which serves as the target of the memory aliasing attacks demonstrated in this work.

4.1 RISC-V TEEs

When developing a TEE, designers often face the dilemma of either relying on existing complex architectures, with their large and difficult-to-verify TCBs, or creating a custom, minimal design. A notable approach to the latter is found in monitor-based TEEs, which employ a thin, trusted software layer to enforce isolation and maintain a small TCB. Costan et al. introduced Sanctum [12], a RISC-V TEE that provides strong, formally provable isolation guarantees for concurrently executing software, conceptually similar to Intel SGX. Although Sanctum requires several hardware modifications to support user-space enclaves, it demonstrates the feasibility of the trusted monitor paradigm.

On RISC-V, several research projects have proposed frameworks that leverage PMP, and possibly architecture-specific hardware abstractions, coupled with a trusted security monitor layer in M-mode to enable isolated TEEs. Weiser et al. present TIMBER-V [64], targeting embedded and Internet of Things (IoT) devices through a tagged-memory architecture that enhances flexibility and efficiency in isolation for embedded systems. Their design reduces memory fragmentation, enables dynamic reuse of untrusted memory, and introduces new concepts for secure memory sharing across protection domains. Bahmani et al. propose CURE [6], a RISC-V enclave architecture that offers support for multiple enclave types — (i) sub-space, (ii) user-space, and (iii) self-contained — allowing for tuning according to application requirements. Lee et al. introduce Cerberus [33], a formally verified framework for secure and efficient memory sharing between enclaves. Cerberus enables enclaves to share memory securely by enforcing immutability and is implemented atop the existing *Keystone* framework, extending its isolation guarantees to multi-enclave communication scenarios.

Similar to AMD SEV, Intel TDX, or Arm CCA, there are multiple research projects developing RISC-V hypervisor-based TEEs. Ozga et al. [46] introduce Assured Confidential Execution (ACE), an open-source confidential computing technology targeting embedded RISC-V systems. ACE defines a set of design principles and methodologies that can be universally used as a basis for designing firmware that requires verification. Ozga et al. [45] also provide a processor-independent confidential computing architecture that can be applied to specific hardware for which the authors implemented the Security Monitor.

Influenced by ACE, Sahita et al. propose CoVE [50], an ISA and non-ISA extension for creating TEE virtual machines (TVMs) on RISC-V. This architecture depends on the TEE Security Manager (TSM) that operates in M-mode. TSM facilitates transitions between confidential and non-confidential operations, provides memory management functionality, and ensures proper isolation and security. CoVE utilizes two separate Application Binary Interfaces, one for allowing TVMs to communicate with the TSM and another for the hypervi-

sor to invoke management functions from the TSM driver, respectively. To achieve privileged operation modes for host-software while providing separation between components existing inside and outside the TCB, CoVE proposes the concept of *Confidential-mode qualifier*. This qualifier state is kept minimally, utilizing one bit in each hart, and is essentially propagated to PMP, MMU lookups, and SoC.

4.2 Keystone Framework

The Keystone framework is an open-source project that facilitates the flexible, extensible creation of TEEs on RISC-V platforms, allowing designers to tailor the TCB to specific security and performance requirements. Its primary objective is to provide an adaptable software-based design that avoids the rigidity often imposed by hardware-bound implementations. Keystone leverages RISC-V's hardware abstractions, particularly PMP to establish isolated memory regions in which enclaves can execute securely, even in the presence of an untrusted OS.

Keystone Operation Modes. Keystone is organized around three main components: *(i)* the SM, which operates in M–mode; *(ii)* the Enclave Runtime (RT) component running in S–mode; and *(iii)* one or more Enclave Applications (EApps) executing in U-mode. Keystone extends the OpenSBI firmware to include the SM, the only part of the framework that executes with machine-level privileges. Figure 3 illustrates how these components are organized across the RISC-V privilege-level hierarchy.

Security Monitor. The SM establishes the system's security boundaries without requiring additional resource management. Based on the Keystone manual, the SM exposes functionality callable by the enclaves and the OS through the SBI. The SM is responsible for enforcing access control over the memory regions assigned to enclaves through the PMP mechanism. It also manages remote attestation, synchronizes PMP entries, handles enclave threading, and implements basic side-channel mitigations. During initialization, the SM explicitly configures the first PMP entry to protect its own memory region, accessible exclusively by the SM, and the last PMP entry to define the untrusted memory region, which typically spans the entire DRAM and is made accessible to the operating system during boot. The remaining PMP entries are dynamically allocated, with one entry consumed per active enclave.

Enclave Memory Isolation. Upon enclave creation, the SM assigns a dedicated PMP entry to the enclave to protect its memory from U- and S-mode software. Keystone also allows the OS to allocate contiguous memory regions within the untrusted address space, which serve as a communication buffer between the enclave and the operating system (referred to as untrusted shared buffer). When the enclave terminates, the SM resets the corresponding PMP permission bits, clears the enclave's state, and releases the PMP entry for reuse by subsequent enclaves.

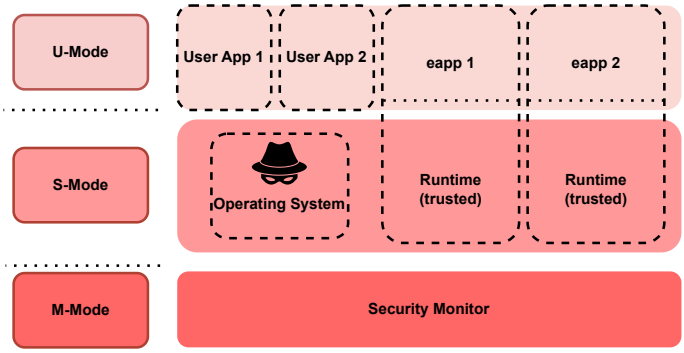

Figure 3: Mental model of Keystone framework illustrating privilege hierarchy and component separation.

Runtime and Untrusted Host. The RT component manages EApps and provides essential enclave services such as in-enclave virtual memory management, system call handling, standard libc support, and communication with the SM through the SBI interface. In Keystone's design, each enclave executes within its own isolated physical memory region, accompanied by a dedicated trusted S-mode RT and an untrusted host utility process running in U-mode. The host process is responsible for deploying the RT and EApp, initiating enclave creation, and exposing edge-call interfaces for communication between the enclave and the host.

5 Porting Keystone to Milk-V Pioneer

The Keystone framework was initially developed and evaluated on the silicon HiFive Unleashed FU540 SoC, as well as various FPGA-based prototypes and the QEMU software emulator platform. In this work, we for the first time port Keystone to run natively on a high-end desktop-class platform, the Milk-V Pioneer (SG2042) board [58]. In the remainder of this section, we describe the OpenSBI firmware, Linux kernel, and SM modifications required to run Keystone-based enclaves on this platform.

Buildroot Modifications. Running Keystone on the Milk-V Pioneer board required extensive modifications to both the framework and its build process. Keystone relies on Buildroot [16] to efficiently cross-compile all required components for RISC-V hardware. We modified the Buildroot configuration to integrate vendor-provided forks of OpenSBI [60]

and the Linux kernel [59], maintained by RevyOS (a custom Debian-based distribution optimized for the SG2042 ecosystem).

At the kernel level, we introduced a custom Linux configuration derived from the RevyOS setup to ensure the correct operation of Keystone enclaves on the Pioneer board. The most critical change was enabling Contiguous Memory Allocation (CMA) support to allow large enclave instantiation, since the kernel's default maximum page order of 10 restricts the number of contiguously allocatable pages [40]. CMA enables dynamic allocation of large contiguous memory regions at runtime, which Keystone requires when the untrusted kernel donates contiguous physical memory for enclave creation. Without CMA support, larger enclaves, e.g., the attestation test case included in the official repository, cannot be launched.

OpenSBI Modifications. In the current upstream Keystone repository, the critical SM component is based on OpenSBI v1.1, which is incompatible with the Milk-V platform. The Keystone OpenSBI fork overrides several predefined platform definitions with Keystone-specific settings and introduces invocation hooks for the SM module during each platform initialization step.

To enable support for the Milk-V Pioneer board, we redefined the platform interfaces within the SM using definitions from newer OpenSBI releases and reapplied Keystone's custom modifications as needed. Because certain features that Keystone depends on were deprecated in newer OpenSBI versions, we adopted the SoC vendor's (Sophgo) development branch based on OpenSBI v1.2 [60]. This version provides the required functionality, is fully supported on the Milk-V board, and integrates cleanly without invasive source-level changes to Keystone.

Adding Secure Boot on Milk-V Pioneer. To ensure stable integration between the Pioneer board and the Keystone framework, and to provide a consistent and complete configuration of the framework on native hardware, we implement a secure boot functionality within the ZSBL. This addition establishes a complete chain-of-trust from the early firmware stage onwards, which validates when enclaves request attestation evidence from the SM. Specifically, we extend Sophgo's ZSBL [61] with a secure boot functionality invoked before transferring control to the OpenSBI firmware. The caller of the secure boot process is responsible for sanitizing the core state and memory. During this process, the bootloader utilizes fixed, predefined device keys to derive version-specific keys for the SM using the `ed25519` cryptographic algorithm.

We generate for the SM specifically the following components based on how Keystone enables secure boot for QEMU emulated platforms [36]: a hash value (SM_{hash}) computed by hashing the OpenSBI binary blob (`fw_dynamic.bin`), a public-private key pair (SM_{PubK} and SM_{PrivK}, respectively), derived using the SM_{hash} and the device private key, and a signature (SM_{sig}) computed over the tuple (SM_{hash}, SM_{PubK}) signed by the device private key. These values are then passed to the SM through predefined linker variables declared in the ZSBL linker script. In our prototype implementation, we utilize predefined dummy device keys to create a proof-of-concept chain-of-trust, but realistically, in a production environment, the keys should be provisioned by the hardware vendor, e.g., via a TPM or hardware key-storage facility.

Note that the firmware code executing on SG2042's system coprocessor, which performs platform initialization before the ZSBL runs, is based on Arm Trusted Firmware-A, which also includes secure boot functionality. In theory, it should be possible to start the chain-of-trust from that stage. However, the specific firmware code is closed source, preventing us from modifying it in order to establish the chain-of-trust or inspecting its implementation to ensure security. Consequently, we opted to establish our chain-of-trust from ZSBL, which is fully open source.

Reproducibility. To facilitate the reproducibility of our experiments and encourage future research on Keystone-based microarchitectural attacks and defenses targeting high-end out-of-order silicon RISC-V processors, we provide publicly available forks of all relevant components. These forks incorporate our modifications to enable native Keystone support on the Milk-V Pioneer board, including the adapted ZSBL. To streamline the development workflow, we also provide a Dockerfile that automatically builds all required firmware components (ZSBL, OpenSBI, Linux kernel images, device trees, etc.) and composes a firmware image that can be easily flashed to an SD card used to boot the real hardware. With these modifications, we achieve a fully functional deployment of the Keystone framework on the Milk-V Pioneer platform. Our port supports enclave creation, attestation verification, and large enclave instantiation via CMA, demonstrating end-to-end functionality with secure boot and remote attestation.

6 Evaluation

In this section, we demonstrate the practical impact of memory aliasing attacks on RISC-V enclaves with two concrete attack scenarios: (1) leaking arbitrary enclave memory, and (2) extracting the SM private secret key (SM_{PrivK}). We demonstrate the feasibility of these primitives by mounting an end-to-end attack against Keystone's enclave remote attestation process.

6.1 Leaking Arbitrary Enclave Memory

We first demonstrate how BadRAM-induced physical memory aliases undermine the isolation guarantees provided by PMP, allowing a malicious host with root privileges to read arbitrary enclave memory. By introducing memory aliases,

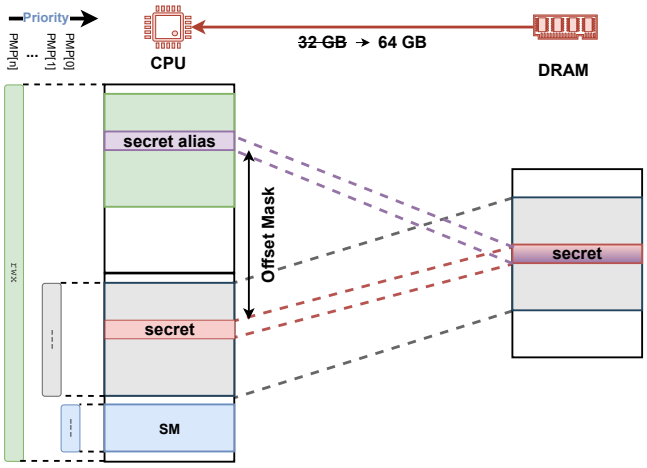

Figure 4: Conceptual illustration of the BadRAM aliasing attack on a RISC-V system running Keystone enclaves protected by PMP. Each PMP entry is defined by `pmpcfg` and `pmpaddr` registers; for simplicity, entries are denoted as PMP[i], where Keystone uses PMP[0], and PMP[n] entries for the SM and untrusted OS memory respectively.

the attacker creates a scenario where two physical addresses map to the same location in DRAM (cf. Figure 4). While the SM configures PMP entries to protect the enclave's memory regions, this protection does not cover the aliased memory regions. By inferring the addresses that correspond to PMP-protected memory regions, the attacker can perform arbitrary reads and writes to the protected memory ranges, violating isolation guarantees provided by PMP and enabling access to secret enclave data.

We decompose our attack into two phases: (i) alias discovery, where we locate the aliases and compute the corresponding address mask, and (ii) data exfiltration, in which we use the discovered alias mask to compute and read from the aliased address range from the untrusted host context to recover enclave contents.

Finding Memory Aliases. The mapping between the original and aliased memory locations depends on the DIMM topology and the physical-to-DRAM memory mapping implemented by the memory controller. As this mapping is typically unavailable to an attacker, the alias mask, *i.e.*, the bit pattern that maps original to aliased addresses, must be reverse engineered. Because the physical-to-DRAM address mapping is static for a given memory configuration, its calculation needs to be performed only once.

We port the simple alias discovery tool proposed by De Meulemeester et al. [14] to RISC-V. Their approach writes unique values to memory and scans for the corresponding changes at other addresses, thereby identifying address pairs that map to the same DRAM location. This simple technique is effective in discovering the alias mask as the Milk-V Pioneer platform does not implement memory scrambling. Using this method, we discover that, on our machine, the alias mask is `0x800000000`, *i.e.*, the ghost bit corresponds to the most significant physical address bit. This result matches previous reverse engineering efforts on the SG2042 that found the physical-to-DRAM mapping to be a simple one-to-one mapping [41].

Leaking Enclave Memory. With the calculated alias mask, we can easily leak arbitrary PMP-protected memory. From the untrusted OS, we compile and insert a modified Keystone driver (the untrusted kernel module responsible for interfacing with the trusted SM) that prints the base address and the size of the allocated private memory of the target enclave. We then calculate the aliased enclave base address as `base_addr` $\oplus$ `alias_mask`. This address range maps to the same location in DRAM, but falls outside the PMP-protected memory region, and thus does not have any memory isolation policies set. The untrusted host can, therefore, access this aliased range without restrictions, leaking the entire enclave memory. As our platform does not employ memory encryption, this yields arbitrary plaintext access to enclave secrets. Note that access to the original, base enclave addresses are still blocked by PMP.

6.2 End-to-end Attestation Attack

We now present an end-to-end attack that completely undermines Keystone's remote attestation. We first show how the attacker can extract the SM private key using the BadRAM primitive. We then demonstrate how the attacker can use this key to forge arbitrary attestation reports, marking malicious enclaves as trustworthy.

Setup. Remote attestation is performed by a trusted verifier, $\mathcal{V}$, whose goal is to determine the authenticity of an enclave. For our experiments, we implement a canonical attestation example using the Keystone SDK and a custom remote verifier. In our setup, the verifier $\mathcal{V}$ generates a random nonce and transmits it to the untrusted host $\mathcal{H}$, which forwards the nonce to the enclave. The enclave then requests an attestation report from the SM, supplying the nonce. The SM then assembles and signs the report with its private key, and returns it to the enclave through a shared buffer. The host relays the report to $\mathcal{V}$, which verifies the report. If all checks succeed, the verifier accepts the enclave as trusted.

6.2.1 Leaking the SM Private Key

Based on Keystone's specification, the attestation report contains fields characterizing both the enclave and the SM. To attest the enclave, the report carries the enclave package hash (covering the loader, runtime module, and eapp binary), the

verifier-supplied nonce and its length, and a digital signature over the tuple ⟨enclave_hash, nonce, nonce_length⟩ computed and signed with SM_{PrivK}. To attest the SM, the report additionally includes the $\langle SM_{hash}, SM_{sig}, SM_{PubK}\rangle$ and the device public key Dev_{PubK}. The verifier recomputes the expected measurements for both the SM and the enclave, compares them to the received values, and accepts the enclave only if all checks succeed. We use this workflow to validate our Keystone port on the Milk-V board and refer to the baseline run as the *GoodRAM* attestation scenario.

In the *BadRAM* scenario, where physical aliasing is present, we implement a malicious host to extract the SM_{PrivK} from the untrusted OS environment. The attack uses the publicly available SM metadata contained in a valid attestation report $\langle SM_{hash}, SM_{sig}, SM_{PubK}\rangle$ as anchors to locate the page holding the SM_{PrivK}. Concretely, the host scans DRAM page-by-page via the BadRAM primitive and tests each page for the presence of the SM metadata. When a page containing the expected $\langle SM_{hash}, SM_{sig}, SM_{PubK}\rangle$ tuple is found, the SM_{PrivK}, located 64 bytes before the public key, can be extracted. The scan time depends on the installed memory size, but the procedure is a one-time cost; once extracted, the SM_{PrivK} can be used for multiple malicious attacks (forged reports, backdoored enclaves, etc.).

6.2.2 Forging Attestation Reports

We now use the extracted SM private key (SM_{PrivK}) to mount an end-to-end attack on Keystone's attestation pipeline. The goal is to forge a new attestation report such that a malicious enclave (*Menc*) passes the remote verifier's checks and is accepted as benign by $\mathcal{V}$.

We use the exfiltrated SM_{PrivK} to synthesize a valid attestation report from the host $\mathcal{H}$, and deliver it to the remote verifier. Recall that a Keystone report contains the enclave hash, the verifier-supplied nonce and its length, and a signature over these fields. The malicious host has access to the nonce and nonce length and can assemble the remaining fields. It obtains the expected enclave hash (by using the benign enclave image) and populates the report accordingly. Finally, the host computes a signature over the tuple ⟨enclave_hash, nonce, nonce_length⟩ using the recovered SM_{PrivK}. The forged report therefore satisfies the verifier's checks and causes the malicious enclave (*Menc*) to be accepted as benign, breaking trust in Keystone's remote attestation.

An overview of the attestation attack is shown in Figure 5. In *Phase 0*, the ZSBL provisions an SM public-private key pair, clears the device private key from RAM, and hands control to OpenSBI. During the OpenSBI boot stage, the SM initializes its runtime using the provisioned keys (see §2.1). In *Phase 1*, we execute a benign attestation, the remote verifier $\mathcal{V}$ generates a nonce and delivers it to the host utility $\mathcal{H}$, which forwards it to the enclave. The enclave requests an attestation report from the SM; the SM assembles and signs the report,

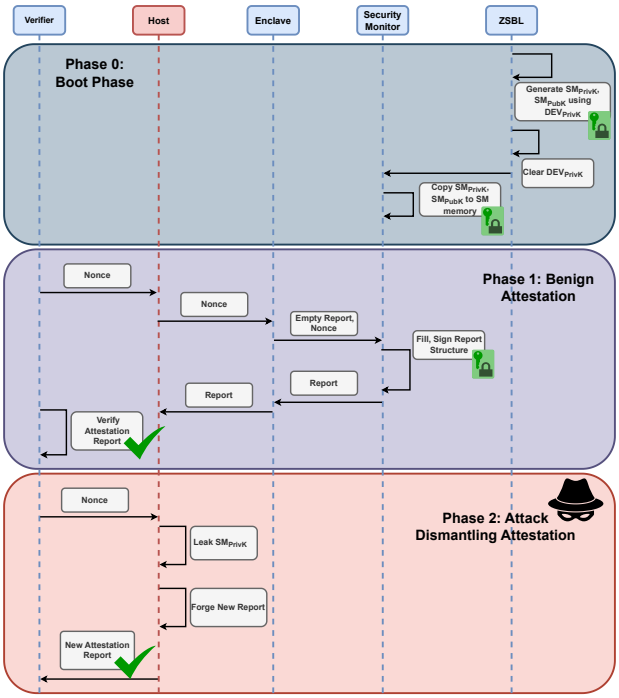

Figure 5: Attestation workflow for Keystone enclaves. Phase 0 (boot): ZSBL generates SM keys. Phase 1 (benign): enclave receives random *nonce* and asks for a signed remote attestation report. Phase 2 (attack): a malicious host leaks and uses the SM private key to forge reports undermining attestation.

and the enclave places the signed report into the untrusted shared buffer. The host $\mathcal{H}$ relays the report to the remote verifier, which validates report fields and signatures.

During the attack (Phase 2), the untrusted host utility $\mathcal{H}$ leaks SM_{PrivK} and uses it to forge an attestation report after obtaining the verifier-supplied nonce. The benign SM generated report contains the device public key (Dev_{PubK}), the SM report fields $\langle SM_{hash}, SM_{PubK}, SM_{sig}\rangle$ provisioned at boot, and the enclave report fields ⟨Enc_hash, Enc_sig, Nonce, Nonce_len⟩. Here, Enc_hash is computed as the hash of the Keystone package components (loader, runtime, and eapp), and Enc_sig is the signature of Enc_hash under the SM private key, e.g. $Enc\_sig = Sign(Enc\_hash, SM_{PrivK})$. Once SM_{PrivK} is recovered, the untrusted host $\mathcal{H}$ can recompute and sign arbitrary enclave reports with the correct nonce and length; the forged report therefore passes the verifier's checks and undermines Keystone's remote attestation.

7 Mitigations

In this section, we discuss mitigation strategies for RISC-V platforms against BadRAM attacks, and DRAM attacks in

Algorithm 1 Linear Boot-Time Alias Detection.

```
1:  for each DIMM do
2:      for i = DIMM.base to DIMM.size step 4096 do
3:          write(i, i)
4:      end for
5:      for i = DIMM.base to DIMM.size step 4096 do
6:          if read(i) ≠ i then
7:              return Alias detected
8:          end if
9:      end for
10: end for
11: return No aliases detected
```

general. The root cause exploited by BadRAM is the system's blind trust in the memory controller configuration, which trusts the topology information reported by the DIMM's SPD chip. To overcome this blind trust, we propose and implement two boot-time alias detection mechanisms, and discuss other mitigations to enhance RISC-V's security in the presence of memory aliasing attacks.

7.1 Boot-Time Alias Checking

A practical and robust defense against static memory aliasing attacks like BadRAM is to validate the physical memory layout at boot-time, before initializing any security-critical components. This ensures that the memory controller was configured correctly and no static aliases are present. Since any subsequent changes to the SPD will only take effect on the next boot, this effectively mitigates (software-based) memory-aliasing attacks. This mitigation is already adopted in practice to protect commercial TEEs, with Intel's opaque `MCHECK` and Alias Checking Trusted Module (ACTM) [30] and AMD's `ALIAS_CHECK` [3]. However, the specific algorithms and implementation details pertaining to these proprietary solutions remain undocumented, with no open-source alias-checking mitigation to date.

Attacker Model. The boot-time alias-checking mitigation described in this subsection targets adversaries capable of manipulating memory configuration metadata prior to boot, such as SPD contents. It effectively detects static memory aliasing attacks like BadRAM, including scenarios where aliases are introduced entirely by a privileged software adversary. Attacks that induce aliases at runtime or through physical manipulation of the memory bus, such as BatteringRAM [13] or Wiretap [53], require hardware-assisted defenses, discussed in Section 7.2.

7.1.1 Linear Alias Check

A straightforward, platform-agnostic approach to detect aliases is to perform a linear scan of the entire addressable DRAM space for each DIMM. This procedure is summarized in Algorithm 1. A first pass iterates over all reported memory regions page by page (i.e., 4 kB) and writes a unique, address-derived magic value, such as the page's base address, into the first 8 bytes of each page. A subsequent verification pass reads back these values and compares them against the expected ones. Any mismatch indicates that a value was overwritten via an alias address, prompting the boot process to abort.

Although conceptually straightforward, this naive implementation exhibits a time complexity of $\mathcal{O}(d \cdot N)$, where d is the number of DIMMs and N the reported size per module. Furthermore, it is susceptible to memory corruption in the presence of aliases, since it cannot prevent writes from overwriting memory regions already in use (e.g., by the ZSBL itself) through their aliased counterparts.

Algorithm 2 Optimized Boot-Time Alias Detection.

```
1:  for each DIMM do
2:      A ← arbitrary DRAM address (e.g., DIMM.base)
3:      p ← random 64-bit marker value
4:      write(A, p)
5:      flush(A)
6:      ▷ Iterate over higher-order address bits > 64-byte cacheline
7:      for i = 6 to log2(DIMM.size) do
8:          B ← A ⊕ (1 ≪ i)
9:          m ← read(B)
10:         if m = p then
11:             return Alias detected
12:         end if
13:     end for
14: end for
15: return No aliases detected
```

7.1.2 Optimized Alias Check

Instead of naively checking every location, it is possible to optimize the algorithm by leveraging insight into how BadRAM introduces aliases at the microarchitectural level. The attack works by tricking the memory controller into driving unused DRAM address lines. Consequently, alias pairs are therefore not random, but only differ in these unused address lines.

Using platform-specific knowledge of the physical-to-DRAM address mapping, it is possible to construct an efficient alias detection algorithm, shown in Algorithm 2. Instead of testing every page, the algorithm only needs to test for aliases along every logical DRAM address bit. For a given base address $\mathcal{A}$, its potential alias $\mathcal{B}$ can be computed by flipping the respective address bit, as shown on line 8. If a value written to $\mathcal{A}$ is visible from $\mathcal{B}$, then these two addresses are aliased. This reduces the time complexity to $\mathcal{O}(d \log(N))$.

7.1.3 Implementation and Evaluation

We validate the feasibility of our boot-time detection by implementing both the linear and optimized alias checking routines within the ZSBL of the SG2042 platform. We modify the ZSBL to perform the alias checking before platform initialization and abort the boot process if memory aliasing is detected in the machine. A secondary option is to allow booting with secure boot disabled (default for our implementation), which would cause all subsequent enclave attestation requests to fail.

Optimized Alias Check. Implementing the optimized alias check requires knowledge of the physical-to-DRAM address mapping. As the SG2042's memory controller is not open-source, we leverage prior work that reverse-engineered its mapping in the context of RowHammer [41]. Marazzi et al. find that the SG2042 employs a straightforward linear mapping instead of using more complicated XOR-based functions as often observed in Intel and AMD processors. Using this knowledge, we implement Algorithm 2. While this optimization is hardware-specific, our linear alias-check variant remains fully memory-agnostic and can operate without knowledge of the underlying mapping, ensuring broader applicability across RISC-V platforms. We encourage RISC-V vendors to incorporate alias-checking mechanisms during the early boot phase, similar to alias-checking implemented by Intel and AMD.

Performance. We measure the performance impact of our mitigation by comparing the cycle count of the alias checking routines and secure boot to the total time for the default, vendor-provided ZSBL stage. The results for three different memory configurations are shown in Table 1.

We verified that both algorithms successfully report the presence or absence of memory aliases. Our results show that the optimized alias scanning is more than five orders of magnitude faster than the linear scan in the tested memory configurations, incurring a negligible overhead compared to the entire ZSBL stage (<0.0001 %).

Note that the linear alias scan performs an early exit upon alias detection, explaining the roughly equal number of cycles for the BadRAM configuration and the corresponding GoodRAM configuration.

7.1.4 Discussion

Fixed Memory Topologies. While our evaluation focuses on RISC-V platforms with pluggable DRAM modules that rely on SPD data, many embedded and IoT-class RISC-V systems use fixed device trees or hardcoded DRAM parameters,without using SPD chips. Similar behavior exists on x86 systems with soldered memory, where SPD data is emulated or cached in firmware. Although such designs limit direct SPD-based aliasing, they retain the same trust assumption; firmware must accurately report the memory topology, thus a compromised or tampered firmware could misreport these parameters to induce comparable aliasing effects.

TCB Recovery. Even with the proposed mitigations, recovering the TCB after a potential compromise remains an important consideration. Once a trusted component is breached, the integrity of the entire TCB can no longer be assumed. Restoring trust requires re-establishing the root of trust and re-measuring critical firmware components such as the ZSBL, OpenSBI, and Keystone's SM.

RISC-V currently lacks a standardized recovery mechanism; however, concepts such as authenticated firmware updates and re-keying could support dynamic re-establishment of trust. Intel's approach [4, 11] provides a useful reference: by incorporating dedicated Secure Version Numbers (SVNs), which are incremented whenever critical firmware is updated, into key derivations, attestation keys can be automatically regenerated without user intervention.

We propose to implement a similar mechanism for RISC-V Keystone at the ZSBL level. By integrating a versioning counter, analogous to Intel's SVN, into the (SM_{PubK}, SM_{PrivK}) keypair generation process, fresh attestation keys for the SM can be automatically regenerated, enabling trust recovery following critical ZSBL root-of-trust updates.

7.2 Hardware Mitigations

Boot-time alias checks provide a robust and low-cost defense against static aliasing attacks such as BadRAM, which may be exploited by software adversaries when the SPD is shipped unlocked. However, these checks are limited to the boot phase: any aliases introduced at runtime, for example, through physical tampering with the memory bus [13], remain undetected. Defending against these more powerful, inherently physical-access adversaries requires more fundamental, hardware-assisted mitigations.

Strong Memory Encryption. When the external DRAM memory is considered untrusted, the memory controller may transparently encrypt the stored data to provide confidentiality, integrity, and/or freshness. While some commercial TEEs have adopted scalable memory encryption designs, opting for larger protected memory regions rather than integrity or freshness, robust mitigations require strong memory encryption to mitigate memory aliasing attacks [13, 14].

One example is Intel's Memory Encryption Engine (MEE) [22], which is designed to mitigate any physical tampering with the memory bus. However, the trade-off for these robust protections is significant, both in terms of performance from integrity and freshness checks, and storage-wise due to message authentication codes and the integrity tree. Recent academic works [18, 25, 51, 57] provide improved solutions, but haven't reached adoption by commercial TEEs yet.

Table 1: Measured execution time of alias checking and secure boot routines under different DRAM configurations. In the 64 GB *BadRAM* configuration, SPD tampering is detected and secure boot does not execute, as indicated by (✗).

Setup	*GoodRAM 1×32 GB*		*GoodRAM 2×32 GB*		*BadRAM 1×64 GB*	
	CPU Cycles	**Time (s)**	**CPU Cycles**	**Time (s)**	**CPU Cycles**	**Time (s)**
Vendor ZSBL	39,762,132,701	19.881066351	39,768,604,853	19.884302427	39,762,132,701	19.884302427
Linear alias check	1,500,525,276	0.750262638	3,160,186,798	1.580093399	1,507,207,189	0.753603595
Opt. alias check	11,051	0.000005525	20,495	0.000010247	11,374	0.000005687
Secure boot	112,175,761	0.05608788	112,886,671	0.056443335	(✗)	(✗)
Total (opt)	39,874,319,513	19.937159756	39,881,512,019	19.940756009	39,874,319,836	19.940395994

Extended PMP. Moolman et al. propose a solution which augments the existing PMP model with system-wide secure memory management capabilities [67]. Their proposal, Extended PMP (ePMP), extends the memory controller hardware with a new structure containing information from the PMP tables of each core. This allows for the controller to maintain a unified view of the protected ePMP regions. Their design relies upon a dedicated, hardware-level MEE, constructing per-region integrity trees whose roots are stored in secure on-chip memory. While such an approach requires substantial architectural changes, it effectively bridges the gap between software-only isolation (as used in Keystone) and hardware-backed confidentiality and integrity, mitigating memory aliasing attacks such as BadRAM.

Scratchpads. A mitigation that is already available on some RISC-V platforms is the use of dedicated scratchpad memory regions, which are address regions that are not backed by DRAM For example, the *Sifive HiFive FU540* provides a scratchpad region that allocates data directly into the cache (L2 Zero Device) [24]. By configuring the *WayEnable* register, software can reserve specific cache ways for this purpose.

This mechanism can be leveraged by TEEs like Keystone, by allowing an enclave to exist only in cache. Keystone already supports utilizing scratchpad regions if the processor supports it. The scratchpad address region can be then used by the SM for enclave allocation.

Components allocated to the scratchpad regions are by definition not affected by memory alias attacks such as BadRAM, as the data never resides in the DRAM. However, scratchpads are only available on select platforms and are not standardized for RISC-V, whereas TEEs like Keystone are designed to support all platforms, with or without scratchpads. Furthermore, scratchpads severely restrict enclave resource management, constraining both the size and the number of concurrent enclaves to the available scratchpad capacity. Andrade et al. [5] explored oversubscribing a limited on-chip scratchpad using enclave self-paging and software-based memory encryption, including integrity and freshness guarantees, albeit at the cost of considerable performance overhead.

8 Conclusion

In this paper, we demonstrated the feasibility of physical memory aliasing attacks on RISC-V platforms, showing that they can break access-control guarantees enforced by the critical PMP memory protection mechanism. We further showed that such aliasing can be practically exploited to leak protected enclave memory and compromise Keystone's remote attestation by extracting the SM private key. Our experiments were conducted on off-the-shelf hardware, the Milk-V Pioneer RISC-V board, after porting Keystone to run natively with added secure boot functionality.

We also proposed low-cost firmware-level mitigations capable of detecting BadRAM-style aliasing during boot and discuss their limitations alongside other potential defenses. Our findings highlight that RISC-V TEEs relying solely on PMP require dedicated aliasing countermeasures and preferably strong memory encryption to protect against physical adversaries. In a wider perspective, our work illustrates how subtle microarchitectural attack vectors, long studied in established architectures such as x86, can also manifest in emerging RISC-V platforms.

Acknowledgements

This work was supported by the Research Fund KU Leuven, and the Cybersecurity Research Program Flanders via grant VOEWICS02. Additionally, this work was funded by the European Commission through Horizon 2020 (ERC #101020005 BELFORT). Jesse De Meulemeester is also funded by an FWO fellowship (11PFE24N). Steven Keuchel is funded by the European Union via a European Research Council (ERC) Starting Grant (UniversalContracts; 101040088), and by the Air Force Office of Scientific Research under award number FA9550-21-1-0054.

References

[1] The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20250508. Accessed: 2025-

10-30. https://docs.riscv.org/reference/isa/_attachments/riscv-privileged.pdf.

[2] The RISC-V Supervisor Binary Interface Specification, Version 3.0. Accessed: 2025-10-30. https://docs.riscv.org/reference/platform-software/sbi/_attachments/riscv-sbi.pdf.

[3] AMD. Undermining integrity features of SEV-SNP with memory aliasing. https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3015.html, December 2024.

[4] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, volume 13, 2013.

[5] Gui Andrade, Dayeol Lee, David Kohlbrenner, Krste Asanovic, and Dawn Song. Software-based off-chip memory protection for risc-v trusted execution environments. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.

[6] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. CURE: A security architecture with CUstomizable and resilient enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1073–1090. USENIX Association, August 2021.

[7] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Zhang Sizhuo, Arvind Arvind, and Srinivas Devadas. Mi6: Secure enclaves in a speculative out-of-order processor. pages 42–56, 10 2019.

[8] Kevin Cheang, Cameron Rasmussen, Dayeol Lee, David W Kohlbrenner, Krste Asanović, and Sanjit A Seshia. Verifying risc-v physical memory protection. *arXiv preprint arXiv:2211.02179*, 2022.

[9] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. Tee.fail: Breaking trusted execution environments via ddr5 memory bus interposition. In *47th IEEE Symposium on Security and Privacy (IEEE S&P '26)*. IEEE Computer Society, 2026.

[10] Confidential Computing Consortium. Current projects. https://confidentialcomputing.io/projects/current-projects/, 2025. Accessed: 2025-10-22.

[11] Intel Corporation. Tcb recovery flow — developer guide. https://www.intel.com/content/www/us/en/docs/dynamic-application-loader/developer-guide/1-0/tcb-recovery-flow.html, 2023. Accessed: 2025-11-04.

[12] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 857–874, Austin, TX, August 2016. USENIX Association.

[13] Jesse De Meulemeester, David Oswald, Ingrid Verbauwhede, and Jo Van Bulck. Battering RAM: Low-cost interposer attacks on confidential computing via dynamic memory aliasing. In *47th IEEE Symposium on Security and Privacy (S&P)*, May 2026.

[14] Jesse De Meulemeester, Luca Wilke, David Oswald, Thomas Eisenbarth, Ingrid Verbauwhede, and Jo Van Bulck. BadRAM: Practical memory aliasing attacks on trusted execution environments. In *46th IEEE Symposium on Security and Privacy (S&P)*, May 2025.

[15] DeepComputing. Deepcomputing stationv d300. https://deepcomputing.io/product/workstation-stationv/, 2025. Accessed: 2025-10-30.

[16] Buildroot Developers. Buildroot — making embedded linux easy. https://buildroot.org/, 2025. Accessed: 2025-10-21.

[17] Dong Du and the RISC-V SPMP Task Group. Risc-v s-mode physical memory protection (spmp) specification, version 0.8.0. Technical report, RISC-V International, 2022. Accessed: 2025-10-29.

[18] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Scalable memory protection in the PENGLAI enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 275–294, 2021.

[19] Cesare Garlati and Sandro Pinto. Secure iot firmware for risc-v processors. *Embbedded world*, 2021, 2021.

[20] Lukas Gerlach, Daniel Weber, Ruiyi Zhang, and Michael Schwarz. A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs. In *S&P*, 2023.

[21] RISC-V IOPMP Task Group. Risc-v iopmp architecture specification. Technical report, RISC-V International, 2025. Accessed: 2025-10-29.

[22] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016.

[23] Sander Huyghebaert, Steven Keuchel, Coen De Roover, and Dominique Devriese. Formalizing, verifying and applying isa security guarantees as universal contracts. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2083–2097, 2023.

[24] SiFive Inc. *SiFive FU540-C000 Manual*. Accessed: 2025-10-24.

[25] Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, and Naofumi Homma. ELM: A low-latency and scalable memory encryption scheme. *IEEE Transactions on Information Forensics and Security*, 17:2628–2643, 2022.

[26] RISC-V International. Anup patel western digital. https://riscv.org/wp-content/uploads/2024/12/13.30-RISCV_OpenSBI_Deep_Dive_v5.pdf, 2024. Accessed: 2025-10-14.

[27] JEDEC. Definitions of the EE1004-v 4 kbit serial presence detect (SPD) EEPROM and TSE2004av 4 kbit SPD EEPROM with temperature sensor (TS) for memory module applications. Standard 21-C, Section 4.1.6, JEDEC, May 2022.

[28] JEDEC. SPD5118 hub and serial presence detect device standard. Standard JESD300-5B.01, JEDEC, May 2023.

[29] Yu Jin, Minghong Sun, Dongsheng Wang, Pengfei Qiu, Yinqian Zhang, and Shuwen Deng. Ghostcache: Timer- and counter-free cache attacks exploiting weak coherence on risc-v and arm chips. 2025.

[30] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlata. Supporting Intel SGX on multi-package platforms. https://arxiv.org/abs/2507.08190, 2025.

[31] Henrik A Karlsson. Minimal partitioning kernel with time protection and predictability. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 234–241. IEEE, 2024.

[32] Nick Kossifidis, Joe Xie, Bill Huffman, Allen Baum, Greg Favor, Tariq Kurd, and Fumio Arakawa. Pmp enhancements for memory access and execution prevention on machine mode (smepmp). Technical report, RISC-V International, 2025. Accessed: 2025-10-29.

[33] Dayeol Lee, Kevin Cheang, Alexander Thomas, Catherine Lu, Pranav Gaddamadugu, Anjo Vahldiek-Oberwagner, Mona Vij, Dawn Song, Sanjit A. Seshia, and Krste Asanovic. Cerberus: A formal approach to secure and efficient enclave memory sharing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 1871—1885, New York, NY, USA, 2022. Association for Computing Machinery.

[34] Dayeol Lee and contributors. https://github.com/keystone-enclave/keystone, 2025. Accessed: 2025-10-21.

[35] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia-che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In *29th USENIX Security Symposium*, pages 487–504, 2020.

[36] 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.

[37] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In *30th USENIX Security Symposium*, pages 717–732, 2021.

[38] Samuel Lindemer, Gustav Midéus, and Shahid Raza. Real-time thread isolation and trusted execution on embedded risc-v. In *Proceedings of the International Workshop on Secure RISC-V Architecture Design Exploration (SECRISC-V)*, 2020.

[39] Zhiye Liu. Gigabyte motherboard firmware update: Saving your DDR5 RAM from corruption. https://www.tomshardware.com/newsgigabyte-motherboard-firmware-update-saving-your-ddr5-ram-from-corruption, September 2023.

[40] The Linux Kernel Source Maintainers. include/linux/mmzone.h — memory zone definitions. https://elixir.bootlin.com/linux/v6.10.10/source/include/linux/mmzone.h#L30, 2025. Accessed: 2025-10-22.

[41] Michele Marazzi and Kaveh Razavi. Risc-h: Rowhammer attacks on risc-v. In *4th Workshop on DRAM Security (DRAMSec) co-located with ISCA 2024*, 2024.

[42] Milk-V. Milk-v pioneer board specifications. https://milkv.io/pioneer, 2025. Accessed: 2025-10-22.

[43] Milk-V. Milk-v titan board specifications. https://milkv.io/titan, 2025. Accessed: 2025-10-22.

[44] Mitre. CWE-1257: Improper access control applied to mirrored or aliased memory regions. https://cwe.mitre.org/data/definitions/1257.html, 2020.

[45] Wojciech Ozga. Towards a formally verified security monitor for vm-based confidential computing. In *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '23, page 73–81, New York, NY, USA, 2023. Association for Computing Machinery.

[46] Wojciech Ozga. Ace: Confidential computing for embedded risc-v systems. 2025.

[47] Shangjie Pan, Yinghao Yang, Xuanyao Peng, Xiquan Zhao, Dong Du, Hang Lu, Yubin Xia, and Xiaowei Li. Layertee: Decoupled memory protection for scalable multi-layer communication on risc-v. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[48] Uros Popovic. Risc-v sbi and the full boot process. https://popovicu.com/posts/risc-v-sbi-and-full-boot-process/, 2024. Accessed: 2025-10-14.

[49] RISC-V. Hypervisor extension, version 1.0. https://five-embeddev.github.io/riscv-docs-html/riscv-priv-isa-manual/latest-adoc/hypervisor.html, 2025. Accessed: 2025-10-22.

[50] Ravi Sahita, Vedvyas Shanbhogue, Andrew Bresticker, Atul Khare, Atish Patra, Samuel Ortiz, Dylan Reid, and Rajnesh Kanwal. Cove: Towards confidential computing on risc-v platforms. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, CF '23, page 315–321, New York, NY, USA, 2023. Association for Computing Machinery.

[51] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, José A. Joao, and Moinuddin K. Qureshi. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 416–427, 2018.

[52] Scaleway. The world's first risc-v servers available in the cloud. https://labs.scaleway.com/en/em-rv1/, 2024. Accessed: 2025-11-03.

[53] Alex Seto, Oytun Kuday Duran, Samy Amer, Jalen Chuang, Stephan van Schaik, Daniel Genkin, and Christina Garman. Wiretap: Breaking server sgx via dram bus interposition. In *2025 SIGSAC Conference on Computer and Communications Security (CCS '25)*. Association for Computing Machinery, 2025.

[54] Agam Shah. China deploys massive risc-v server in commercial cloud. https://www.hpcwire.com/2023/11/08/china-deploys-massive-risc-v-server-in-commercial-cloud/, 2023. Accessed: 2025-11-03.

[55] Sipeed. Lichee console 4a. https://sipeed.com/licheepi4a, 2023. Accessed: 2025-11-03.

[56] RISC-V Software Source. Opensbi: Domain-support documentation. https://github.com/riscv-software-src/opensbi/blob/master/docs/domain_support.md. Accessed: 2025-10-29.

[57] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 665–678. ACM, 2018.

[58] Milk-V Pioneer Team. *SG2042 Technical Reference Manual*. Accessed: 2025-10-21.

[59] RevyOS Team. sg2042-vendor-kernel: Linux kernel stable tree for sg2042. https://github.com/revyos/sg2042-vendor-kernel, 2025. Accessed: 2025-10-21.

[60] Sophgo Technologies. opensbi: Risc-v open source supervisor binary interface. https://github.com/sophgo/opensbi, 2025. Accessed: 2025-10-21.

[61] Sophgo Technologies. zsbl: Sophgo risc-v zero stage boot loader. https://github.com/sophgo/zsbl, 2025. Accessed: 2025-10-21.

[62] Fabian Thomas, Eric García Arribas, Lorenz Hetterich, Daniel Weber, Lukas Gerlach, Ruiyi Zhang, and Michael Schwarz. RISCover: Automatic Discovery of User-exploitable Architectural Security Vulnerabilities in Closed-Source RISC-V CPUs. In *CCS*, 2025.

[63] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *26th ACM Conference on Computer and Communications Security (CCS)*, pages 1741–1758, November 2019.

[64] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v. In *26th NDSS Symposium*, 01 2019.

[65] Xcalibyte. Roma laptop pre-order. https://xcalibyte.com.cn/en/romapreorder/, 2022. Accessed: 2025-11-03.

[66] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd M. Austin. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 313–324, 2017.

[67] Tamara Silbergleit Lehman Zach Moolman. Extending risc-v keystone to include efficient secure memory. 2024.