Real-time Thread Isolation and Trusted Execution on Embedded RISC-V

Samuel Lindemer, Gustav Midéus, Shahid Raza
RISE Cybersecurity Unit
RISE Research Institutes of Sweden
firstname.lastname@ri.se

***Abstract*—The Internet of Things paradigm has led to an increasing demand for low-power single-core embedded devices with hardware-enforced trusted execution environments (TEE). The preeminent solution in this space is Arm TrustZone, which provides four CPU execution states, each with its own memory and instruction permissions. In 2019, physical memory protection (PMP) instructions were ratified in the RISC-V ISA, which offers enhancements similar to an Arm MPU, but does not enable the creation of TEE frameworks similar to TrustZone. In this paper, we discuss a limitation in the PMP specification which precludes the design of such a TEE framework on RISC-V without resorting to a non-standard hardware modification. We propose a simple modification to the PMP specification which would resolve this limitation without the addition of additional registers. We discuss our early ongoing work in implementing a prototype of this hardware extension and integrating it with the Zephyr real-time operating system (RTOS).**

***Index Terms*—embedded, security, IoT, RISC-V, TEE**

I. Introduction

Traditional operating systems prevent applications from accessing physical memory outside their addresses spaces through a combination of software virtualization and a memory management unit (MMU) for address translation and process isolation. Highly constrained embedded devices (e.g., battery-powered nodes with up to 32 KiB RAM) do not have an MMU – all application code addresses physical memory directly. Consequently, a successful code reuse attack (CRA) could give an adversary complete control of system peripherals and memory. Connecting such devices to a network would be ill advised.

Constrained devices can employ a *memory protection unit* (MPU) – essentially a simplified MMU without address translation – to achieve hardware-enabled memory isolation of unprivileged software. An MPU requires the existence of at least two hardware-enforced CPU privilege levels. In privileged mode, the CPU can modify the memory access permissions for unprivileged mode, and is thus usually reserved for the embedded operating system. On each context switch, the CPU reconfigures the MPU to allow access only to the stack and heap required by the next thread.

The Arm Cortex-M processor core series, introduced in 2004, has since become a market leader for low-power networked embedded devices, as it was designed from the outset for strong RTOS support and an MPU hardware extension. Arm later introduced TrustZone for Cortex-M, a set of hardware extensions which enable the construction of trusted execution environments (TEE). TrustZone creates two virtual processors known as the *secure world* and the *non-secure world* (see Figure 2). Each world has two privilege levels enforced by its own MPU.

The open source RISC-V ISA began development in 2010 and became frozen in 2019, allowing developers and researchers to work on a stable platform. RISC-V differentiates itself from existing instruction sets, such as Arm and x86, through its extensibility. Bare-metal embedded devices with a single privilege level can be built with only the core ISA, but a powerful multi-core machine can also be constructed using the optional extensions. Several FPGA-ready open source RISC-V implementations have been developed, such as the Rocket Chip [1] and LiteX SoC with VexRiscv core [2]. These projects have greatly improved RISC-V's accessibility as a platform for research in computer architecture.

II. Related Work

Since the RISC-V ISA was frozen quite recently, some of the existing TEE research is no longer consistent with the latest specification. In 2017, Costan et al. gave a detailed summary of the inherent challenges in verifying security claims made about Intel SGX, the premier TEE solution for cloud environments [3]. The authors advocate for simplicity above all else in TEE design, as the number of vulnerabilities tend to scale with the size of the trusted computing base (TCB). The latter half of the same work introduced the Sanctum TEE architecture for RISC-V [4] [5], which is heavily influenced by SGX.

In 2019, two other TEE frameworks for RISC-V were proposed, TIMBER-V [6] and Keystone [7]. TIMBER-V is intended for low-power embedded systems with real-time scheduling requirements. This architecture is built on several hardware extensions (tagged memory, tag-aware instructions, new CPU registers and a custom MPU) in order to create two *domains*, similar in principle to Arm TrustZone, and does not utilize RISC-V's recently added *physical memory protection* (PMP) instructions. Keystone, in contrast, is designed for less constrained machines with a traditional OS and does not diverge from the latest ratified RISC-V ISA.

All three of these TEE frameworks for RISC-V require three CPU privilege levels. The base ISA has only one level, *machine mode*, which is intended for bootloaders and bare metal code. Two more levels are currently ratified as extensions, *supervisor mode* and *user mode*, which are intended for running a traditional operating system with MMU support and its applications. Sanctum and Keystone utilize M-mode to run a *security monitor* (SM), which provides context switching between an arbitrary number of enclaves running in the lower privilege levels.

Low-power RISC-V platforms only implement M- and U-mode, which poses an additional challenge for TEE development. This configuration is explicitly recommended in the RISC-V privileged ISA specification for embedded devices [8]. Hex-Five MultiZone[1] is a proprietary TEE solution for these types of devices, which is also built on an M-mode SM, but the enclaves run solely in U-mode [9].

III. Real-time Thread Isolation on RISC-V

A single-core 32-bit RISC-V machine with the U-mode extension and PMP hardware bears many functional similarities to an ARMv8-M machine running solely in the non-secure world. An RTOS can utilize the memory protection features on either architecture by setting the bounds of accessible memory to the top of the stack and bottom of the heap of the executing thread on every context switch [10] (see Figure 1).

A. *RTOS support for RISC-V*

Despite the apparent ease of implementing PMP support on an OS that already supports an Arm MPU, few embedded operating systems currently do. The embedded operating systems listed below, with the exception of FreeRTOS, currently support thread isolation with the ARMv8-M MPU in their upstream repositories but not RISC-V PMP.

Operating system	RV32I support	RV64I support	Real-time	License
FreeRTOS[2]	✓	✗	✓	MIT
Mynewt[3]	✓	✗	✓	Apache 2.0
NuttX[4]	✓	✗	✓	BSD
Pharos[5]	✗	✓	✓	Apache 2.0
RIOT[6]	✓	✗	✓	LGPL 2.1
Zephyr[7]	✓	✓	✓	Apache 2.0

One area of our ongoing work is implementing PMP-enforced *userspace* support to Zephyr, a real-time operating system backed by The Linux Foundation. This RTOS supports a handful of RISC-V platforms (e.g., SiFive HiFive1 and LiteX VexRiscv). However, all threads currently run in M-mode alongside the kernel.

B. *Porting from Arm to RISC-V*

1) Privilege levels: Both ARMv8-M and embedded RV32I have two privilege levels. Machines boot directly into the highest privilege level which has, by default, access to all registers, including those for configuring the memory protection rules. Programs running in an unprivileged state often need to make system calls that can only be handled in a privileged CPU state (e.g., a thread jumping to the RTOS). This is done on ARMv8-M by raising a special exception with the *supervisor call* `svc` instruction [11]. On RISC-V, there is an equivalent *environment call* `ecall` instruction [12].

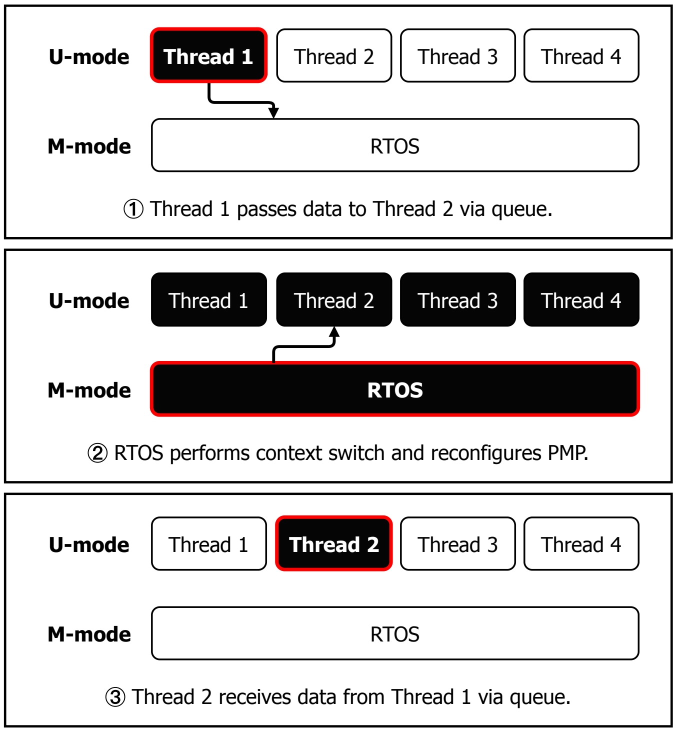

U-mode
Thread 1
Thread 2
Thread 3
Thread 4
M-mode
RTOS
① Thread 1 passes data to Thread 2 via queue.
U-mode
Thread 1
Thread 2
Thread 3
Thread 4
M-mode
RTOS
② RTOS performs context switch and reconfigures PMP.
U-mode
Thread 1
Thread 2
Thread 3
Thread 4
M-mode
RTOS
③ Thread 2 receives data from Thread 1 via queue.

Fig. 1. Overview of RTOS thread isolation on RV32I with PMP. (A red border indicates the running program; a black fill indicates the address space currently accessible to that program.)

2) Memory faults: When an illegal memory access occurs in an unprivileged CPU state, a fault is triggered and handled in privileged mode. On ARMv8-M, this entails setting the *memory management fault address* `MMFAR` register with the location of the illegal access and the *memory management status* `MMFSR` register to indicate whether the fault resulted from an attempted load or store instruction, or an instruction fetch. On RV32I, the *machine trap value* `mtval` register is set to the address of the illegal memory access and the *machine cause* `mcause` register is set to indicate whether the fault resulted from an illegal load, store or instruction access.

3) Managing regions: The memory protection rules dictate which portions of RAM, ROM and MMIO are readable, writable and/or executable from unprivileged execution modes. It is also possible to protect regions from privileged accesses until the machine is reset. ARMv8-M supports eight MPU regions; RV32I supports up to sixteen. Region management is quite similar from the programmer's perspective on either architecture.

On ARMv8-M, the permissions and size of a memory region are declared with a write to the region number `MPU_RNR`

[1]https://hex-five.com/first-secure-iot-stack-riscv/
[2]https://www.freertos.org/index.html
[3]http://mynewt.apache.org/
[4]https://nuttx.org/
[5]https://sourceforge.net/p/rtospharos/wiki/Home/
[6]https://www.riot-os.org/
[7]https://www.zephyrproject.org/

register, which selects a region to be configured on the subsequent writes to the base address `MPU_RBAR` and limit address `MPU_RLAR` registers [13]. These latter writes determine the permissions applied to the selected region.

On RV32I, four PMP configuration registers, `pmpcfg0` to `pmpcfg3`, declare the permissions and sizes of all sixteen memory regions. These correspond to a set of PMP address registers, `pmpaddr0` to `pmpaddr15`, which declare the location of each region. PMP regions are prioritized, which means the lowest-numbered PMP entry matching the address of a memory access determines success. If the lock bit is set in a `pmpcfg*` register, the permission settings are applied to all privilege levels until system reset. PMP registers are *only* accessible in M-mode.

IV. Real-time Enclaves on RISC-V

Our ultimate goal is a standard RISC-V compliant architecture that supports both thread isolation *and* trusted execution environments without sacrificing real-time scheduling guarantees. This is currently possible on ARMv8-M TrustZone platforms (see Figure 2). The *worlds* of TrustZone are orthogonal to the privilege levels enforced by the MPUs. A single bit is added to every system memory transaction to indicate whether it originated from a secure or non-secure CPU state, and various SoC components may also be TrustZone-aware simply by reading this bit.

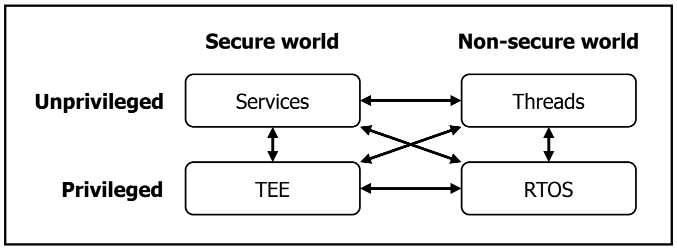
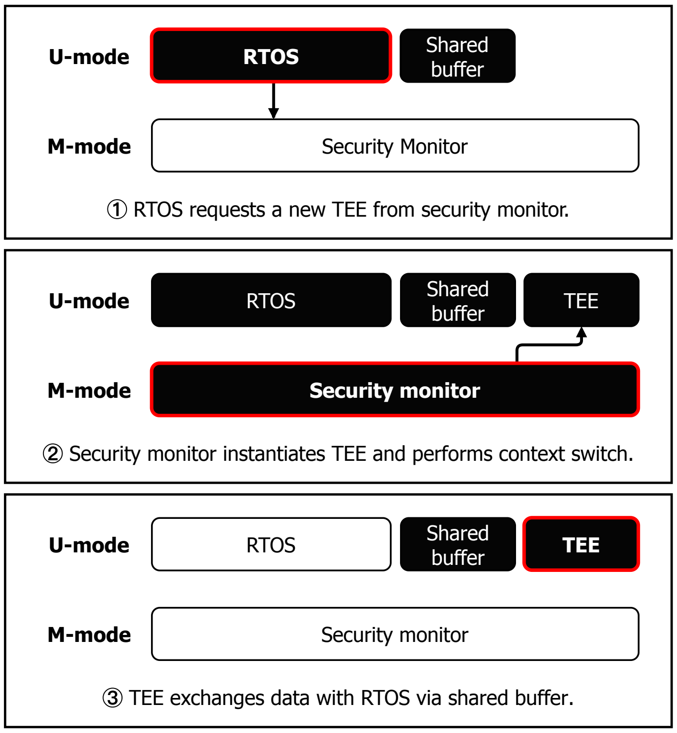


Fig. 2. Recommended TrustZone configuration for an RTOS running in the non-secure world with threads requiring access to secure world APIs. Note that direct transitions between any of the four processor states is possible in ARMv8-M. Context switching from secure world threads is handled by the CMIS-RTOS v2 API extension [14], as the RTOS cannot directly access secure world stack memory.

The recommended RISC-V configuration for secure embedded systems, according to the privileged ISA specification, has only two privilege levels. This is sufficient for thread isolation (see Figure 1) *or* complete isolation of programs with a security monitor (see Figure 3), but not both. The latter approach places the RTOS in U-mode, which precludes the use of PMP to protect and isolate thread memory.

A. Proposed solution

The solution we are currently developing requires minor alterations to the RISC-V PMP specification. In order to instantiate an arbitrary number of mutually isolated execution environments with two privilege levels, a third privilege level running a security monitor is required (see Figure 4). This is, indeed, the approach employed by Keystone for TEEs on


U-mode
M-mode
RTOS
Shared buffer
Security Monitor
① RTOS requests a new TEE from security monitor.
TEE
Security monitor
② Security monitor instantiates TEE and performs context switch.
③ TEE exchanges data with RTOS via shared buffer.
Fig. 3. Simple TEE architecture on the unmodified RISC-V ISA with only two privilege levels. This approach precludes the use of PMP to isolate RTOS threads, as depicted in Figure 1.

larger machines [7]. However, this approach cannot be directly translated to constrained platforms because RISC-V's S-mode uses an MMU to isolate U-mode software.

It is uncommon to implement an MMU on highly constrained low-power chips due to energy and cost considerations. There are no CPUs in the Arm Cortex-M lineup with an MMU, for example. Some embedded operating systems, Zephyr included, support MMU hardware, but only in *thread protected mode*. This configuration applies an identity page table. In other words, the MMU works as an MPU with unlimited regions by disabling address translation. Underutilizing hardware in this way is a usually poor design trade-off.

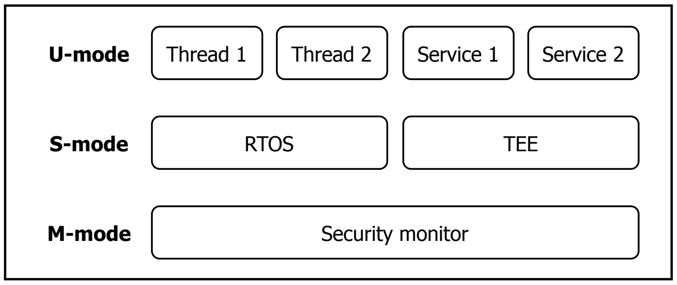


Fig. 4. Three privilege levels are required to achieve both RTOS thread isolation and enclave separation. The latter can be achieved with existing PMP functionality; the former is possible with our proposed addition to the PMP specification.

We propose a relatively simple alteration to the current PMP specification which would allow some PMP registers to be configured in S-mode, and for their restrictions to apply only to U-mode. This would be achieved with a simple flag added to the `pmpcfg*` registers, as shown in Figure 5. This modification would be sufficient to create a TEE framework such as that shown in Figure 4.

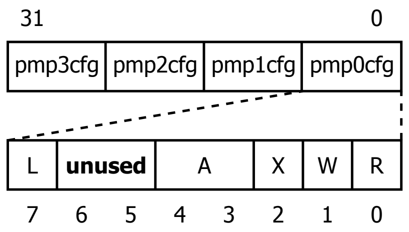


Fig. 5. The current specification of the `pmpcfg*` registers leaves two bits per region unused. By re-purposing one of these bits as a flag to enable S-mode configuration, our desired configuration would be achieved without adding new registers to the ISA.

In order to maintain real-time scheduling behavior in the proposed architecture, security monitor implementations must perform context switches in a deterministic amount of time. Since enclave memory bounds are modifiable only in M-mode, all API calls from one enclave to another must be facilitated by the security monitor. Direct API calls between enclaves would require additional hardware modifications, so this is ultimately a design trade-off. However, we argue that our approach is justified as it maintains a simple programming model for developers which will, in turn, minimize the prevalence of software vulnerabilities.

We are in the process of implementing our PMP modification for the VexRiscv core[8] and LiteX SoC running on an Arty A7 FPGA. We are currently developing our own security monitor suitable for constrained platforms with real-time requirements and an accompanying API for Zephyr RTOS running as untrusted S-mode software.

B. Related proposals

At a recent RISC-V summit, a representative of Huawei presented a modified PMP scheme for embedded security [15]. In this architecture, only M- and U-mode are implemented, but an additional set of PMP registers accessible only to U-mode has been added. This allows a U-mode program to block M-mode access to its memory space, which offers protection against an adversary in control of M-mode software. Our trust assumptions differ from Huawei, as we consider M-mode software to be trusted. Our modification to the PMP specification would not require additional registers, and M-mode would be allowed to override S-mode PMP settings.

SiFive's recently-announced WorldGuard hardware extensions promise to create TEEs on RISC-V chips with two privilege modes [16]. Based on the information currently available, WorldGuard appears to require all untrusted software to run in U-mode. This would preclude the use of PMP for thread isolation (see Figure 3).

[8]https://github.com/SpinalHDL/VexRiscv

V. Conclusion and Future Work

TEE framework development on RISC-V is an active research area with a handful of competing proposals emerging annually. We propose an architecture for low-power single-core RISC-V TEEs requiring one simple modification to the ISA, namely the addition of a single-bit flag in place of an unused bit in the PMP configuration registers. This enables simultaneous thread isolation and TEE separation on devices with a flat address space (i.e., without an MMU). Our work regarding implementation, performance evaluation and API support for Zephyr RTOS is ongoing.

VI. Acknowledgement

This research has been supported by the EU H2020 CONCORDIA project (grant agreement No 830927).

References

[1] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[2] F. Kermarrec, S. Bourdeauducq, J.-C. Le Lann, and H. Badier, "LiteX: an open-source SoC builder and library based on Migen Python DSL," 03 2019.

[3] V. Costan, I. Lebedev, and S. Devadas, "Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture," *Foundations and Trends® in Electronic Design Automation*, vol. 11, no. 1-2, pp. 1–248, 2017. [Online]. Available: https://doi.org/10.1561/1000000051

[4] ——, "Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture," *Foundations and Trends® in Electronic Design Automation*, vol. 11, no. 3, pp. 249–361, 2017. [Online]. Available: https://doi.org/10.1561/1000000052

[5] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation," in *USENIX Security Symposium*, 2016.

[6] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V," in *NDSS*, 2019.

[7] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanovic, "Keystone: A Framework for Architecting TEEs," *CoRR*, vol. abs/1907.10119, 2019. [Online]. Available: http://arxiv.org/abs/1907.10119

[8] A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20190608-Priv-MSU-Ratified*, RISC-V Foundation Std., June 2019.

[9] "MultiZone™ Security Secure IoT Stack Document Version Rev 1.0.2," Hex Five Security, Inc., Tech. Rep., September 2019.

[10] J. Labrosse. (2018, Deember) Using the RISC-V PMP with an Embedded RTOS to Achieve Process Separation and Isolation. Silicon Labs, Inc. [Online]. Available: https://youtu.be/upkZZldpljA

[11] *ARMv8-M Architecture Reference Manual*, Arm Limited, 110 Fulbourn Road Cambridge, England CB1 9NJ, June 2017.

[12] A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213*, RISC-V Foundation Std., December 2019.

[13] *ARMv8-M Memory Protection Unit*, Arm Limited, 110 Fulbourn Road Cambridge, England CB1 9NJ, February 2017.

[14] *RTOS design considerations Version 2.0*, Arm Limited, 110 Fulbourn Road Cambridge, England CB1 9NJ, February 2017.

[15] T. Kurd. (2019, December) Architectural Extensions for a RISC V Processor for Embedded Security. Huawei Technologies Co., Ltd. [Online]. Available: https://youtu.be/tiOs8DlkkIk

[16] B. Wheeler, "SiFive Secures RISC-V," Tech. Rep., November 2019.