Bridging the Interoperability Gaps Among Trusted Architectures in MCUs

Sandro Pinto[1], Luís Cunha[1], Daniel Oliveira[1], Michele Grisafi[2], Emanuele Beozzo[2], and Bruno Crispo[2]

[1] University of Minho
sandro.pinto@algoritmi.uminho.pt,id11207@alunos.uminho.pt,
daniel.oliveira@dei.uminho.pt
[2] DISI - University of Trento
{michele.grisafi, emanuele.beozzo, bruno.crispo}@unitn.it

Abstract. The increasing adoption of IoT devices highlights the urgent need for robust security solutions, particularly on low-end MCUs. However, these platforms typically lack the advanced security mechanisms found in higher-end processors, making them highly vulnerable to attacks. This paper presents an interoperable solution that enables Trusted Execution Environments (TEEs) on low-end MCUs, leveraging platform-specific hardware security primitives available in both ARMv7-M and RISC-V architectures. By complementing these primitives with software-based isolation techniques, the proposed solution achieves strong domain isolation even in resource-constrained devices. Additionally, the adoption of the GlobalPlatform API provides a common abstraction layer that decouples application development from platform-specific details, allowing trusted applications to run seamlessly across different MCU platforms. In the evaluation, we assessed the solutions through performance microbenchmarks and real-world applications, demonstrating their feasibility and the trade-offs between security and performance across different workloads.

Keywords: Trusted Execution Environment · TEE · IoT Security · Armv7-M · RISC-V · Embedded systems · Interoperability.

1 Introduction

Microcontroller Units (MCUs) have become integral to the rapid expansion of the Internet of Things (IoT) as they continue to permeate sectors such as healthcare, manufacturing, and smart cities, serving as the computational backbone for a multitude of connected devices. These compact integrated circuits, which combine processors, memory, and input/output peripherals, are embedded within devices to manage specific functions, enabling seamless data exchange and automation. The global IoT microcontroller market was valued at USD 5.55 billion in 2024 and is projected to grow at a compound annual growth rate (CAGR) of 16.3% from 2025 to 2030, reflecting their critical role in IoT applications.

Low-end microcontrollers, are experiencing significant popularity due to their cost-effectiveness and suitability for simple applications. This growth is largely driven by their integration into consumer electronics, automotive systems, and industrial automation. ARM Cortex-M3 and M4 microcontrollers, for example, have gained substantial popularity due to their balance of performance and efficiency, and the Cortex-M4 processor, with its enhanced Digital Signal Processing (DSP) capabilities, has become a preferred choice for applications requiring efficient motor control and complex algorithm processing. Despite their pivotal role in the IoT ecosystem, MCUs present unresolved security challenges that could critically impact the integrity and safety of connected systems. Haoqi et al. [27] were able to bypass security defenses in embedded systems by misusing the ARM Cortex-M's Flash Patch and Breakpoint (FPB) unit. Firmware vulnerabilities further exacerbate the security risks associated with MCUs. An empirical study, analyzing IoT firmware identified 29 distinct Common Weakness Enumerations (CWEs) in code snippets shared across developer platforms, indicating prevalent weaknesses that could be exploited in real-world scenarios. Moreover, the infamous Mirai malware exemplifies how compromised IoT devices, often leveraging MCU vulnerabilities, can be co-opted into botnets to launch large-scale cyberattacks. These incidents underscore the necessity for robust security measures in MCU design and implementation to safeguard the expanding IoT landscape. A key aspect of designing security mechanisms to counter such attacks is the ability to equip MCUs with a Trusted Execution Environment (TEE). TEEs serve as the foundation of security across a wide range of computing devices, spanning from high-performance application processor units (APUs) to, more recently, resource-constrained MCUs used in IoT applications [25]. Although TEEs, such as Intel SGX and Arm TrustZone, have been widely deployed in APUs to secure mobile/embedded devices for over two decades, MCUs remain largely behind. In most cases, MCUs either lack critical security hardware primitives or fail to leverage them effectively when present [28].

For instance, MCUs lack virtual memory support via a Memory Management Unit (MMU) and, as a result, cannot run full-fledged operating systems like Linux. Instead, MCUs typically rely on a Memory Protection Unit (MPU), which enforces fine-grained access control over memory regions based on a predefined set of entries [15]. Each entry specifies a memory region's base address, bounds, and access permissions (read, write, and/or execute). Despite the inclusion of MPUs, MCUs' security capabilities remain insufficient against the evolving landscape of IoT threats, as MPUs lack the flexibility and robustness needed to counter sophisticated attacks [26]. As IoT devices become more interconnected and exposed to diverse attack vectors, relying solely on MPUs for security leaves critical vulnerabilities unaddressed [12]. Hardware security primitives are essential for ensuring strong isolation at both the core and platform levels, providing fundamental support for TEE implementations [22]. However, these primitives are highly platform-dependent and vary significantly across vendors. To address this issue, this paper presents an architectural solution to strengthen the security of MCUs while ensuring interoperability between different MCU vendors and

platforms. This is achieved by designing and implementing TEEs that can also run on low-end MCUs, leveraging the hardware security primitives (e.g., MPU) provided natively by each platform, while complementing them with software-based mechanisms to enable full TEE functionality. Furthermore, the presented TEEs offer a common abstraction layer through the standardized GlobalPlatform APIs [1, 2], enabling developers to create Trusted Applications (TAs) that can seamlessly execute across different MCU platforms. The paper implements this approach on two of the most currently relevant MCU architectures, ARMv7-M and RISC-V. The main contributions of this paper are as follows:

- A comprehensive analysis of hardware security primitives in ARMv7-M and RISC-V MCU profiles.
- The design of a multi-platform TEE architecture compliant with the GlobalPlatform API, tailored for low-end MCUs.
- Two open-source implementations of the proposed TEE: one for ARMv7-M [14] and another for RISC-V (M+U) [21].
- An extensive evaluation of both implementations through microbenchmarks and real-world applications.

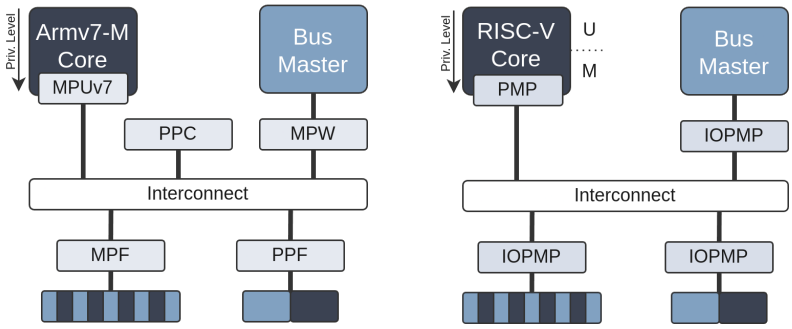
The remainder of this paper is structured as follows. Section 2 presents background information on the target architectures and the GlobalPlatform specifications. Section 3 details the design and implementation of our TEE, followed by a performance evaluation in Section 4. Section 5 reviews related work on low-end MCU-based TEEs. Finally, Section 6 concludes the paper.

2 Background

2.1 Hardware Security Primitives Across Architectures

The architectural models in Figure 1 are designed based on the capabilities of the underlying hardware primitives. Our analysis focuses on hardware features available in ARMv7-M and RISC-V platforms.

ARMv7-M. At the core level, ARM defines various types of security controllers, some shared between processor families with slight variations. One such controller is the MPU. In ARMv7-M, the MPU (MPUv7) can enforce memory protection on fixed, Naturally Aligned Power-of-Two (NAPOT) region sizes. However, the MPU only enforces restrictions at the core-level. To prevent other bus masters from bypassing these policies, e.g., DMAs and secondary cores, vendors typically define several platform-specific controllers placed between memory and peripherals. It is important to note that the deployment of these primitives varies by vendor, and some platforms may not implement them at all. The Platform Protection Controller (PPC) manages registers that configure system resource permissions, including bus master security attributes and peripheral privilege levels. The Master Protection Wrappers (MPW) sit between legacy or permission-unaware masters and the system, classifying transactions



(a) ARMv7-M. (b) RISC-V (M+U).

Fig. 1: Hardware security primitives for the different architectures.

as secure/non-secure or privileged/non-privileged, typically under PPC control. Similarly, Peripheral Protection Firewalls (PPF) and Memory Protection Firewalls (MPF) act as MPUs for protected slaves, filtering transactions based on their configurations.

RISC-V. The RISC-V privilege specification defines multiple privilege levels and the optional Physical Memory Protection (PMP) mechanism. A hart's privilege level determines what the running software can execute, enforcing isolation within the software stack and triggering exceptions for unauthorized operations. The closest configurations to an MCU in RISC-V are those with only one or two privilege levels, i.e., Machine (M) mode only or both M and User (U) mode (M+U). If implemented, the PMP applies access checks to all memory transactions, with configurations managed by the highest privilege level. It offers greater flexibility, supporting both top-of-the-range memory region definitions and NAPOT encoding (akin to ARMv7-M), with regions as small as 4 bytes. Similar to ARM's MPU, the PMP enforces checks only at the core level. To address this limitation, the RISC-V community has been developing the I/O Physical Memory Protection (IOPMP) specification to regulate device access to memory by verifying permissions on non-regulated transactions. While open implementations of these bus-level controllers exist [9], the specification remains unratified, and commercial solutions are still limited.

2.2 TEEs and Specifications

A TEE functions as a secure, isolated domain within a system, specifically designed to protect sensitive operations, as depicted in Figure 2. TEEs typically include a kernel operating at the highest privilege level and several secure applications, i.e., TAs, which enable modularization and isolation of different services.

However, the rapid evolution of these systems has led to significant fragmentation and heterogeneity in embedded device software stacks. The choice of TEE in a device's security backend is largely driven by the Original Equipment

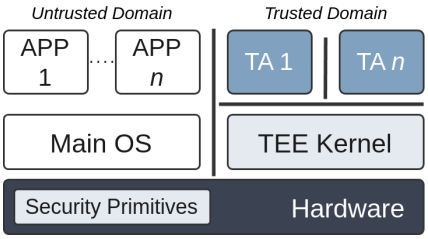


Fig. 2: TEE general architecture.

Manufacturer (OEM) and chipset. Since each TEE may have its own unique API, developing TAs that run across multiple architectures becomes considerably more challenging.

Global Platform. To counteract this fragmentation and ensure the compatibility of TAs across different TEE implementations, led to the widespread adoption of GlobalPlatform (GP) standards. GP's suite of specifications defines various aspects of TEE architecture all the way from internal operations to functional interfaces. The TEE Client API [1] specifies how untrusted applications interact with security services provided by TAs. Meanwhile, the TEE Internal Core API [2] defines the services available to TAs within the TEE.

3 A cross-platform TEE

In this paper, we propose a cross-platform TEE architecture for low-end MCUs. The design follows the GP standard and supports multiple domains: the TEE kernel, TAs, and the Rich Execution Environment (REE). While the TEE kernel and TAs manage security-critical operations (e.g., cryptographic and security services), the REE can host either a bare-metal application or an RTOS instance.

To ensure the security of the system, we adopt a zero-trust model, where each software component is isolated and only the TEE kernel is inherently trusted. This model guarantees that security services can operate independently without being compromised by the REE. Additionally, by isolating TAs from both the REE and the TEE kernel, potential vulnerabilities within TAs cannot compromise the overall system. To enforce isolation and prevent the various domains from breaking this isolation, the TEE kernel ensures that each domain runs unprivileged with its own dedicated memory resources.

Traditional TEEs rely on complex hardware-based security primitives for domain isolation. For example, TrustZone introduces additional privilege levels (enclosed in the Secure World enclave), while MMUs enforce fine-grained access control policies. In contrast, typical MCUs offer only basic security mechanisms, such as two privilege levels and an MPU or Physical Memory Protection (PMP). Although these features enable hardware-based resource isolation, they have limitations that result in coarse-grained separation. To address these limitations, we enhance these architectural features with software-based isolation

techniques, ensuring strong separation between software components. Additionally, by extending the existing hardware features, our TEE maintains a small Trusted Computing Base (TCB), reducing the overall system's attack surface.

To control the interactions between the various domains and, at the same time, bolster the interoperability of both TAs and Client Applications (CAs)[3] we adopt the GP APIs. These regulate the communication between TAs, CAs, and the TEE kernel, while abstracting the underlying architecture, enabling a unified interface for developing TAs and CAs. In particular, this abstraction decouples application development from platform-specific details, allowing the same application to be seamlessly executed across different MCU architectures without requiring modifications. GP specifies two types of APIs: Core and Client. The Core API is used for communication between TAs and the TEE kernel, enabling TAs to request services from the kernel. In contrast, the Client API allows the CA to request services from either the TA or the TEE. While our TEE is fully compliant with the Client API, we only support a subset of the Core API. The original GP Core API is a rich set of interfaces to support many different use-cases and services, most of which might be superfluous on MCUs, where only a subset of these API could suffice. This set could be defined based on the required functionalities, thus reducing the TCB and preserving valuable resources. In particular, we support the Operation, Memory and Object Management functions, the Secure Storage functions, the Random Value and Key generation functions, the BigInt functions, and the Symmetric, Asymmetric, MAC and Message Digest functions.

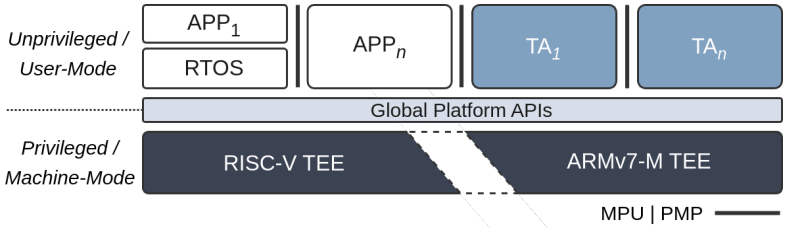


Fig. 3: ARMv7-M and RISC-V (M+U) TEE architecture.

As depicted in Figure 3, our implementation targets two distinct architectures: ARMv7-M and RISC-V (M + U). At its core, our approach relies on two key techniques: *privilege reduction* and *memory isolation*, both essential for isolating different domains. While these architectures share similar features, the implementation must be customized for each.

[3] These applications reside in the REE and interact with the TA and the TEE.

3.1 ARMv7-M Implementation

The ARMv7-M architecture supports an MPU with a limited number of NAPOT regions, thus making the isolation of multiple domains challenging. While the MPU can be reconfigured at runtime to virtually support an unlimited number of domains, this approach would introduce excessive overhead, making the TEE impractical. To limit the overhead and still enable multiple domains, our implementation supports up to two different TAs. Furthermore, we extend the privileges of the TAs over the REE, allowing them to communicate without dedicated shared regions. Figure 4 shows the ARMv7-M ACP.

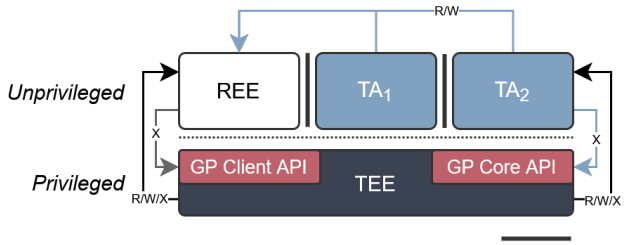


Fig. 4: Access Control Policy for extra-domain memory accesses.

Privilege reduction. A fundamental prerequisite for strong isolation between the REE and TEE is privilege separation. On ARMv7-M, code executes in either privileged or unprivileged mode. Unprivileged mode is restricted by the MPU's access policy and has limited access to critical system resources, such as the Private Peripheral Bus (PPB). The PPB contains essential registers that control the entire board, including MPU configurations. In most cases, privilege reduction is straightforward, as unprivileged code cannot regain privileges except through controlled entry points (e.g., SVC calls). However, two challenges arise: (i) the REE may need privileged access to configure parts of the board (e.g., interrupts) via the PPB, and (ii) interrupt service routines (ISRs) inherently execute with privileges. Luckily, most accesses to the PPB generate an exception (trap) when executed without privileges. To manage controlled access to the PPB[4], whenever the REE triggers such an exception the TEE invokes a PPB Access Recovery routine. This routine verifies the REE operation's legitimacy and, if valid, executes it within the privileged TEE.

Memory isolation. While privilege separation regulates access to hardware configuration, memory isolation must be explicitly enforced using the MPU. Unlike an MMU, the MPU operates on physical addresses and is unaware of the running software. Given the limited number of MPU regions, these must be shared among all software components while maintaining the ACP illustrated in

[4] Only non-critical registers are accessible.

Figure 4. Dynamic MPU reconfiguration is costly, whereas a static configuration alone cannot fully implement our ACP. To balance security and performance, the TEE OS enforces a static region layout while allowing dynamic sub-region configurations.

TEE Execution. The TEE is the first component to execute on boot, running with full privileges. After configuring the device, it bootstraps the REE by lowering its privileges and ensuring it remains unprivileged throughout execution. For handling interrupts, the TEE employs a privilege reduction sequence consisting of an *exception catcher*, *exception simulator*, and *exception return*. Specifically, the TEE replaces each ISR with a custom handler (exception catcher) that fabricates a return frame pointing to the exception simulator. This component then drops privileges and prepares the execution of the original user-defined ISR in unprivileged mode. Finally, when the ISR returns to the exception return, it restores the system to its original state.

Software instrumentation. Although most privileged operations on ARMv7-M trigger a fault when executed without the necessary privileges, some do not. For instance, system instructions like `CPS`, `MSR`, and `MRS`, which are essential for device configuration, fail silently when executed in unprivileged mode. Since our approach runs the entire application unprivileged, these instructions must be trapped and emulated. To address this, we instrument the code to explicitly trigger a trap whenever such instructions are used.

GlobalPlatform Compliance. The GP APIs are split into Core and Client APIs, both of which signal a context switch between TA-OS and CA-TA, respectively. Since both TAs and the CA are executed without privileges, every call to a Core or Client API from their side is implemented through an `SVC` call. This will trigger a context switch between domains, and allow the TEE kernel to configure the MPU and re-direct the execution to the proper domain.

3.2 RISC-V Implementation

In the RISC-V ecosystem, the lack of standardized TEE architectures results in fragmentation, with proprietary solutions lacking interoperability and clear security guarantees. While the different privilege levels and the use of a PMP allow access control, their limitations require software-based enhancements. Therefore, similarly to the solution presented for ARM, we augment these hardware features with software primitives to improve isolation guarantees and security.

Privilege Reduction. The multiple privilege levels defined in the RISC-V privileged specification enable the physical separation of software stack layers. Additionally, the PMP enforces access control by restricting lower privilege levels from accessing system resources. This hardware controller is configured via CSRs, accessible only in M-Mode, allowing M-Mode software to regulate resource access for the entire software stack. Considering these features, and to mitigate privilege escalation, the TEE kernel operates at the highest privilege level, while

every other application and service is de-privileged and runs on the lower privilege levels. Additionally, each non-secure domain must be restricted to only the essential resources, such as specific devices and system services.

Memory Isolation. Similar to ARM's MPU, the number of PMP entries constrains the number of regions that can be protected. This limited number of entries must be shared among all execution domains. While reconfiguring PMP entries at each scheduling point increases domain-switching overhead, it enhances system flexibility by enabling more domains and more protected regions per domain. Alternatively, entries can be pre-configured and toggled as needed for constrained real-time scenarios. Although this solution would reduce the performance overhead,[5] it could also negatively impact system scalability, for instance limiting the number of available TAs. This especially on systems with a limited number of PMP entries, e.g., low-power devices, and with domains that do not share memory regions.

TEE Execution. During initialization, the TEE boot agent initializes essential CPU- and platform-specific hardware components. It then loads each software component, i.e., TEE kernel and domains's binaries, into their designated memory regions. Finally, the boot process concludes by configuring the PMP to enforce memory access policies and jumping to the first domain. Scheduling is managed using the *mtimer* with a round-robin policy. At each scheduling point, four key actions occur: (i) saving the current domain's context, (ii) selecting the next domain for execution, (iii) reconfiguring the PMP to grant access to the new domain's memory regions, and (iv) restoring the new domain's context. Additionally, a domain can issue an `ecall` to pass execution arguments and request a specific domain, enabling standardized APIs such as the GP.

GlobalPlatform Compliance. In RISC-V, the `ecall` instruction enables a domain to request services from the execution environment by triggering the exception handler, which processes the request. As Figure 5 depicts, integrating this mechanism with the GP API allows a domain to invoke specific TEE services, which are interpreted and handled within TEE, eventually directing execution to the domain where the TA runs. The runtime environment ensures compliance with GP by providing the necessary APIs for the TA. Additionally, moving these APIs to a de-privileged part of the system, helps minimize the TEE's Trusted Computing Base (TCB), effectively reducing its attack surface.

4 Evaluation

In this section, we present the performance characteristics of the proposed TEE solutions. The evaluation considers both low-level microbenchmarks, such as domain switch time and TA-based API execution, as well as real-world application

[5] In general, it could reduce the number of instructions required during a context switch by approximately $2 \times$ *Number of PMP Regions*

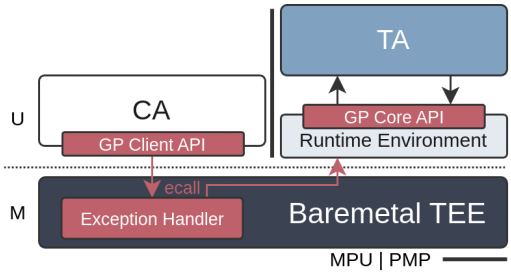


Fig. 5: Ecall instruction's execution flow.

workloads, including a smart lock and a Bitcoin wallet. By using the same workloads on these distinct architectures, we aim to highlight the interoperability the presented solutions offer when designing secure workloads.

The ARMv7-M solution was evaluated on a B-L475E-IOT01A board from STMicroelectronics, running a Cortex-M4 core. The board is equipped with 1MB of Flash memory and 128KB of SRAM. The RISC-V solution was evaluated on a Digilent ARTY7 35T FPGA, running the E300 SoC. The E300 features an RV32IMAC core with a 16KB L1 instruction cache and a 16KB data SRAM. Additionally, it supports only M- and U-mode.

4.1 Microbenchmarks

In the presented microbenchmarks, we analyzed the performance overhead associated with interactions between the executing domains. Specifically, we measured the time required for domain switches and the execution of TA-based APIs to assess their impact on overall system performance. The performance metrics are presented in clock cycles, which provides a more reliable basis for cross-platform comparison and evaluation. In RISC-V, this measurement was obtained using the *cycle* CSR, while in ARM, it was collected using a logical analyzer. The results are demonstrated in Table 1.

Table 1: Performance comparison of domain switch times and TA-based APIs (in clock cycles).

	Domain Switch Time		**TA-based APIs**		
	$u.d. \rightarrow t.d.$	$t.d \rightarrow u.d.$	Open Session	Invoke Command	Close Session
ARM	1746	892	2300	3234	1962
RISC-V	446	479	6880	6782	1082

Domain switch time. In this experiment, we measured the time required to transition from one domain to another by measuring the clock cycles between the last instruction of the calling domain and the first instruction of the target domain. On the ARMv7-M solution, a context switch requires a backup of all registers and the handling of the input and output parameters. In particular, during a switch to the TEE, the input parameters must be converted into a TA format. Furthermore, the MPU sub-regions must be configured to maintain the isolation between the two worlds.

Similarly, the primary bottleneck in the RISC-V solution is the overhead associated with saving and restoring the full execution context. During a domain switch, all 32 general-purpose registers (apart from a0) must be stored and reloaded, a process that is particularly time-consuming as these values must be written to and retrieved from memory. Additionally, the reported cycle count includes a full reconfiguration of the PMP regions. In time-sensitive applications, using statically allocated PMP regions with dynamic switching, i.e., simply toggling the enabled regions, could reduce the number of instructions required for PMP reconfiguration. As the used SoC features only 8 regions, common for low-power implementations, the number of instructions could be reduced by 16 (around 16 cycles for this SoC) which would represent a reduction of $\simeq 7\%$.

TA-based APIs. The typical communication model in TEEs involves CAs interacting with TAs via a remote procedure call (RPC) mechanism [7]. This model is session-based and generally includes three key operations: (i) establishing a session with the TA, (ii) invoking functions within the TA by sending commands, and (iii) terminating the session once the TA operation has finished. To evaluate these APIs, we measured the number of clock cycles between the CA invoking the API and the execution of the first instruction of the TA's corresponding handler function. In both implementations, the main performance bottleneck for these APIs stems from the data preparation required by the runtime environment before transferring execution to the TA. While this ensures compliance with the GP API, it introduces additional overhead. This impact is particularly evident in the significant performance gap between the Close Session API and the other two, as it involves the least amount of data transfer between domains.

4.2 Real-world Applications

We evaluate two real-world applications implemented as TAs: a smart lock application and a Bitcoin Wallet. The smart lock is based on a reference implementation of a trusted keypad integrated into the Vulcan system [29]. The Bitcoin Wallet is based on an open-source implementation that supports six commands, ranging from master key generation to transaction signing. The experiment aims to compare the execution time of the applications running in a bare-metal setup, i.e., everything executes within the same domain, against a setup where security services are isolated within a TA. This comparison allows us to assess the performance degradation of the proposed solutions in real-world applications. The results are presented in Table 2.

Table 2: Execution time comparison for smart lock and Bitcoin wallet operations in single and two-domain setups on ARM and RISC-V platforms (in clock cycles).

	Smart Lock		**Bitcoin Wallet**	
	Single Domain	*Two Domains*	*Single Domain*	*Two Domains*
ARM	574.4K	574.6K **(1.003x)**	84M	93M **(1.10x)**
RISC-V	137K	374K **(2.73x)**	373M	374M **(1.003x)**

Smart Lock. The integration of the smart lock as a TA follows the communication protocol specified by the GP APIs. This means that the application initiates a session with the smart lock, sends a command to enter the pin code, and subsequently closes the session. This sequence effectively tests all of the API calls. The results for ARMv7-M show a negligible overhead of 1.003x when using the proposed TEE, compared to the bare-metal application running within the same domain. In the RISC-V implementation the overhead increases to 2.73x.

Bitcoin Wallet. The integration of the Bitcoin Wallet as a TA also follows the communication protocol defined by the GP APIs. In this case, the application initiates a session with the wallet and sends a command to generate a new mnemonic and corresponding masterkey. The session is then closed after the operations are completed. As demonstrated by the results, as neither the ARMv7-M nor the RISC-V platforms feature a cryptographic hardware acceleration primitives, it requires a lot of clock cycles to complete its operations. Both TEE introduce small overhead, with the ARMv7-M version adding 1.10x overhead and the RISC-V a negligible 1.003x slowdown.

5 Related Work

TEEs have been extensively studied in the literature across both high-end systems and low-end MCU devices. In the latter category, research efforts have focused on extending existing security features, addressing their limitations, or designing TEEs from scratch.

Low-end MPU-less devices. Among the broad spectrum of MCUs, some low-end devices lack even the most basic security features. For example, many MCUs from the MSP430 and AVR architectures, as well as certain ARMv6-M devices, lack an MPU or any other hardware module for enforcing memory isolation. To address this gap, several solutions have proposed hardware modifications to create either hardware-assisted [10, 17, 6, 20] or purely hardware-based [20] TEEs. While effective, these approaches are often impractical in real-world scenarios, as they require costly replacement and redeployment of existing insecure devices.

As a more cost-effective alternative, software-based TEEs have been introduced to establish a Root of Trust without relying on dedicated security hard-

ware. Two notable examples are SμV [3], designed for the Modified Harvard architecture, and PISTIS [13], tailored for the Von Neumann architecture. While PISTIS provides a fully functional yet minimal TEE, it does not support GP APIs and imposes significant overhead.

ARMv7-M. Only a few works have attempted to introduce a fully-fledged TEE for the ARMv7-M architecture. Hermes [16] is a lightweight hypervisor that supports the isolation of multiple execution domains, all running unprivileged over a thin runtime software layer. While Hermes can support TA-like services, its primary focus is on optimized interrupt handling rather than strict security guarantees. On the other hand, MultiZone [24] is a proprietary TEE that prioritizes security. It enables the creation of multiple static user-defined enclaves, referred to as zones, each assigned a dedicated set of isolated resources (e.g., memory, peripherals). MultiZone leverages the MPU as its primary isolation mechanism, with a lightweight runtime manager orchestrating interactions between zones. Although these approaches are capable of providing strong security guarantees, they provide a non-standard architecture that is neither compatible with common TAs nor with the GP APIs.

RISC-V. The emerging RISC-V standard defines two main processor profiles: the Application profile (RVA), designed for high-end processors with an MMU, and the MCU profile (RVM), used in low-end SoCs that lack an MMU. This profile also often provides only up to two privilege levels (M + U). Notably, neither profile includes a built-in hardware-based TEE, prompting researchers to explore both hardware and software solutions. Several studies have proposed hardware extensions to introduce TEE support in RISC-V, either targeting the Application profile [4, 19, 23, 8] or the MCU profile [30]. However, as mentioned previously, deploying hardware modifications is often impractical.

Boubakri et al. [5] ported the open-source OP-TEE architecture to RISC-V, enabling a GP-compliant TEE. Unlike our work, their approach relies on an MMU for access policy enforcement, thus being compatible only with the RVA profile. A more flexible approach is Keystone [18], which dynamically configures the PMP at run-time to isolate multiple security enclaves spanning over two privilege levels, User and Supervisor. Similarly to our work, Keystone introduces a privileged run-time module (executed at M) to handle the configuration of the device and the communication between domains. Although Keystone is not explicitly tailored for MCUs, its isolation primitive is based on the PMP rather than the MMU, thus being compatible with low-end devices. Following this approach, Penglai [11] is an open-source TEE that supports both the RVA and RVM profiles, focusing on the deployment of a large quantity of enclaves. A similar approach is used by MultiZone [24], which was originally created to support RISC-V MCUs, offering multiple secure enclaves using the PMP. Interestingly, MultiZone's hardware requirements align with our work, but, like the aforementioned TEEs, it does not conform with GP specifications, requiring the development of custom TAs and security services.

6 Conclusion

This work presented two solutions for enabling TEEs on low-end MCUs, addressing both security and interoperability challenges in IoT devices. By leveraging hardware security primitives and enhancing them with software-based mechanisms, our approach ensures strong isolation guarantees across different MCU architectures. Furthermore, by conforming to the GlobalPlatform API, we demonstrated how TAs can be seamlessly deployed across multiple platforms without modification. Our evaluation confirmed the feasibility of this approach on both ARM and RISC-V architectures, highlighting the performance trade-offs and the impact of the proposed solutions on real-world applications.

Acknowledgments. Funded by the European Union under Horizon Europe Programme - Grant Agreement 101070537 — CrossCon. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Climate, CINEA. Neither the European Union nor the granting authority can be held responsible for them.

References

1. TEE Client API Specification v1.0. Technical report, Global Platform (2010)
2. TEE Internal Core Specification v1.2.1. Technical report, Global Platform (2019)
3. Ammar, M., Crispo, B., Jacobs, B., Hughes, D., Daniels, W.: SμV—The Security MicroVisor: A Formally-Verified Software-Based Security Architecture for the Internet of Things. IEEE Transactions on Dependable and Secure Computing (2019)
4. Bahmani, R., Brasser, F., Dessouky, G., Jauernig, P., Klimmek, M., Sadeghi, A.R., Stapf, E.: CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In: 30th USENIX Security Symposium (USENIX Security 21) (2021)
5. Boubakri, M., Chiatante, F., Zouari, B.: Open Portable Trusted Execution Environment framework for RISC-V. In: 2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC) (2021)
6. Brasser, F., El Mahjoub, B., Sadeghi, A.R., Wachsmann, C., Koeberl, P.: TyTAN: Tiny Trust Anchor for Tiny Devices. In: Proceedings of the 52nd annual design automation conference. pp. 1–6 (2015)
7. Cerdeira, D., Santos, N., Fonseca, P., Pinto, S.: SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In: 2020 IEEE Symposium on Security and Privacy (SP) (2020)
8. Costan, V., Lebedev, I., Devadas, S.: Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In: 25th USENIX Security Symposium (USENIX Security 16) (2016)
9. Cunha, L., Marques, F., Rodríguez, M., Gomes, T., Sá, B., Pinto, S.: Open-source RISC-V Input/Output Physical Memory Protection (IOPMP) IP. In: RISC-V Summit Europe 2024 (2024)
10. Eldefrawy, K., Tsudik, G., Francillon, A., Perito, D.: SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In: Ndss. vol. 12, pp. 1–15 (2012)
11. Feng, E., Lu, X., Du, D., Yang, B., Jiang, X., Xia, Y., Zang, B., Chen, H.: Scalable Memory Protection in the PENGLAI Enclave. In: 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). pp. 275–294 (2021)

12. Grisafi, M., Ammar, M., Crispo, B.: On the (in)security of Memory Protection Units : A Cautionary Note. In: 2022 IEEE International Conference on Cyber Security and Resilience (CSR) (2022)
13. Grisafi, M., Ammar, M., Roveri, M., Crispo, B.: PISTIS: Trusted Computing Architecture for Low-end Embedded Systems. In: 31st USENIX Security Symposium (USENIX Security 22) (2022)
14. Grisafi, M., Beozzo, E., Crispo, B.: ARMv7-M TEE respository, https://github.com/crosscon/baremetal-tee/tree/main/MPU-version
15. Hardin, T., Scott, R., Proctor, P., Hester, J., Sorber, J., Kotz, D.: Application Memory Isolation on Ultra-Low-Power MCUs. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18) (2018)
16. Klingensmith, N., Banerjee, S.: Hermes: A Real Time Hypervisor for Mobile and IoT Systems. In: Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications (2018)
17. Koeberl, P., Schulz, S., Sadeghi, A.R., Varadharajan, V.: TrustLite: A Security Architecture for Tiny Embedded Devices. In: Proceedings of the Ninth European Conference on Computer Systems. pp. 1–14 (2014)
18. Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K., Song, D.: Keystone: An Open Framework for Architecting TEEs. In: Proceedings of the Fifteenth European Conference on Computer Systems (2020)
19. Nasahl, P., Schilling, R., Werner, M., Mangard, S.: HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (2021)
20. Noorman, J., Agten, P., Daniels, W., Strackx, R., Van Herrewege, A., Huygens, C., Preneel, B., Verbauwhede, I., Piessens, F.: Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base. In: 22nd USENIX Security Symposium (USENIX Security 13). pp. 479–498 (2013)
21. Oliveira, D., Cunha, L., Pinto, S.: RISC-V TEE respository, https://github.com/crosscon/baremetal-tee/tree/riscv
22. Oliveira, D., Gomes, T., Pinto, S.: uTango: An Open-Source TEE for IoT Devices. IEEE Access (2022)
23. Pan, S., Peng, X., Man, Z., Zhao, X., Zhang, D., Yang, B., Du, D., Lu, H., Xia, Y., Li, X.: Dep-TEE: Decoupled Memory Protection for Secure and Scalable Inter-enclave Communication on RISC-V (2025)
24. Pinto, S., Garlati, C.: Multi Zone Security for Arm Cortex-M Devices. In: Embedded World Conference (2020)
25. Pinto, S., Matjaz, B.: A Novel Trusted Execution Environment for Next-Generation RISC-V MCUs. In: Embedded World Conference (2024)
26. Rodrigues, C., Oliveira, D., Pinto, S.: BUSted!!! Microarchitectural Side-Channel Attacks on the MCU Bus Interconnect. In: 2024 IEEE Symposium on Security and Privacy (SP) (2024)
27. Shan, H., Sullivan, D., Arias, O.: When Memory Mappings Attack: On the (Mis) use of the ARM Cortex-M FPB Unit. In: 2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST) (2023)
28. Tan, X., Ma, Z., Pinto, S., Guan, L., Zhang, N., Xu, J., Lin, Z., Hu, H., Zhao, Z.: SoK: Where's the "up"?! A Comprehensive (bottom-up) Study on the Security of Arm Cortex-M Systems. In: 18th USENIX WOOT Conference on Offensive Technologies (WOOT 24) (2024)

29. Van Bulck, J., Mühlberg, J.T., Piessens, F.: VulCAN: Efficient Component Authentication and Software Isolation for Automotive Control Networks. In: Proceedings of the 33rd Annual Computer Security Applications Conference (2017)
30. Weiser, S., Werner, M., Brasser, F., Malenko, M., Mangard, S., Sadeghi, A.R.: TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V. In: Proceedings 2019-Network and Distributed System Security Symposium (NDSS) (2019)