# A Survey of RISC-V Secure Enclaves and Trusted Execution Environments

Marouene Boubakri * and Belhassen Zouari

Posted Date: 2 October 2025

doi: 10.20944/preprints202510.0152.v1

Keywords: RISC-V; security; trusted execution environment; confidential computing; secure enclave

*Article*

# A Survey of RISC-V Secure Enclaves and Trusted Execution Environments

**Marouene Boubakri [1,*] and Belhassen Zouari [2]**

[1] Mediatron Lab, SupCom University of Carthage Tunis, Tunisia
[2] ICL, Junia, Catholic University of Lille, LITL, F-59000 Lille, France
* Correspondence: marouene.boubakri@ept.u-carthage.tn; Tel.: +216-99-283-600

**Abstract**

RISC-V has emerged as a compelling alternative to proprietary instruction set architectures, distinguished by its openness, extensibility, and modularity. As the ecosystem matures, attention has turned to building confidential computing foundations, notably Trusted Execution Environments (TEEs) and secure enclaves, to support sensitive workloads. These efforts explore a variety of design directions, yet reveal important trade-offs. Some approaches achieve strong isolation guarantees but fall short in scalability or broad adoption. Others introduce defenses such as memory protection or side-channel resistance, though often with significant performance costs that limit deployment in constrained systems. Lightweight enclaves address embedded contexts but lack the advanced security features demanded by complex applications. In addition, early stage development, complex programming models, and limited real-world validation hinder their usability. This survey reviews the current landscape of RISC-V TEEs and secure enclaves, analyzing architectural principles, strengths, and weaknesses. To the best of our knowledge, this is the first work to present such a consolidated view. Finally, we highlight open challenges and research opportunities toward establishing a cohesive and trustworthy RISC-V trusted computing ecosystem.

**Keywords:** RISC-V; security; trusted execution environment; confidential computing; secure enclave

## 1. Introduction

The rise of heterogeneous and often untrusted platforms, from large-scale clouds to resource-constrained IoT devices, has elevated confidential computing to a primary concern for protecting data in use. Trusted Execution Environments (TEEs) [1] and secure enclaves have emerged as foundational building blocks to address this challenge, offering hardware-enforced isolation, attestable execution, and secure data management. Commercial implementations from major CPU vendors, including Intel Software Guard Extensions (SGX) [2], AMD Secure Encrypted Virtualization (SEV) [3], and ARM TrustZone [4], have demonstrated the practical viability of TEEs across diverse operational domains, from user-space application isolation to full virtual machine encryption.

However, the widespread academic and industrial adoption of these proprietary TEEs has revealed significant and inherent limitations. Each implementation supports only a narrow subset of the design space, making trade-offs that often result in critical constraints. For instance, Intel SGX provides strong user-space isolation but imposes severe memory limits and lacks secure I/O [5], while AMD SEV (prior to SEV-SNP) offered memory encryption without integrity protection, leaving it vulnerable to remapping attacks [6]. ARM TrustZone, though efficient for embedded systems, is constrained by a coarse-grained two-world model that lacks process-level isolation within the secure world [7]. These architectural compromises mean that no single proprietary TEE can adequately meet the diverse requirements of modern applications. Furthermore, their closed-source, vendor-locked nature inhibits independent verification, customization, and adaptation to novel threat models or use cases, stifling innovation and forcing developers to accept suboptimal compromises [8].

 doi:10.20944/preprints202510.0152.v1



Security is another profound concern. Despite their architectural guarantees, proprietary TEEs have been consistently breached through microarchitectural side-channel attacks. Vulnerabilities such as Foreshadow [9] and Load Value Injection (LVI) [10] against Intel SGX, or SEVered against AMD SEV [6], have demonstrated that memory encryption and logical isolation can be bypassed, leaking sensitive data from within enclaves. ARM TrustZone has also been shown vulnerable to attacks such as TruSpy [11] and TruSense [12], which exploit cache-based side channels between the Secure and Normal worlds, as well as CLKSCREW [13], which leverages power management features to induce faults in secure-world execution. These results illustrate that even TEEs based on coarse-grained isolation models are not immune to microarchitectural or implementation-level attacks. These incidents underscore that TEE security is not merely a function of architectural design but is deeply contingent on the underlying hardware's resilience to speculative execution and other microarchitectural leakage channels [14].

The limitations of proprietary TEEs have thus created a compelling need for open, customizable, and verifiable alternatives. It is in this context that the RISC-V instruction set architecture has gained prominence. Characterized by its modularity, extensibility, and transparency, RISC-V presents an ideal foundation for confidential and trusted computing research [15]. Its open nature allows hardware security extensions to be rigorously evaluated, customized, and prototyped without vendor restrictions. Consequently, a vibrant landscape of academic and industrial initiatives has emerged, exploring a diverse array of TEE and secure enclave designs for RISC-V, each introducing distinct architectural innovations to address the shortcomings of their proprietary counterparts.

Despite this growing momentum and the proliferation of proposals, the research community lacks a consolidated understanding of the RISC-V TEEs and and secure enclaves design space. While a few prior surveys have touched upon this topic [16,20,22,81], they typically offer only a limited overview. Broader surveys [16,18,19,21–23] on this area remain centered on proprietary architectures and overlook the unique opportunities and challenges inherent to an open hardware ecosystem [17].

To address this gap, this paper presents the first comprehensive survey of enclave and trusted execution in the RISC-V ecosystem. We systematically map the landscape of proposed TEE and secure enclave architectures, categorizing them based on their isolation mechanisms, threat models, and key design choices. Our analysis provides a structured comparison of these diverse approaches and identifies the recurring trade-offs that arise in their design. Furthermore, we highlight common limitations, discuss open research challenges, and outline promising directions for developing secure, efficient, and truly open trusted execution environments. By consolidating this fragmented landscape, the survey offers researchers and practitioners a critical reference point for advancing confidential computing on open architectures.

The main contributions of this paper are threefold: (i) we provide a systematic survey of RISC-V TEEs and secure enclaves, examining their architectural foundations, security properties, and limitations; (ii) we develop a taxonomy that highlights common design patterns and trade-offs across proposals; and (iii) we discuss open challenges and outline future research directions toward a cohesive and trustworthy trusted computing ecosystem for RISC-V. By consolidating a fragmented landscape, this survey offers both researchers and practitioners a structured reference to guide future work on secure and trustworthy open-hardware platforms.

## 2. Background

The purpose of this section is to establish the necessary foundations for the survey. We introduce the essential concepts of TEEs and secure enclaves, outlining their security properties, design goals, and common threat models. We then summarize the architectural elements of RISC-V that are directly relevant to confidential computing, including privilege levels, memory protection mechanisms, and recent security extensions. This background provides the terminology and taxonomy needed to analyze and categorize existing RISC-V TEE and secure enclaves proposals in subsequent sections.

 doi:10.20944/preprints202510.0152.v1



### 2.1. Trusted Execution Environments and Secure Enclaves

Trusted Execution Environment (TEE) is a term originally introduced by GlobalPlatform in its specifications [24]. Despite its formal definition, the term is often used loosely and sometimes misappropriated for marketing purposes, resulting in a lack of consensus across industry and academia. The demand for stronger security features, driven by the complexity and connectivity of modern systems from general-purpose computers to embedded and IoT devices, has accelerated the adoption of trusted computing concepts. Efforts such as [1] attempt to clarify the notion of a TEE by proposing a logical architecture and threat model. In this view, a TEE is a tamper-resistant environment that ensures: (i) isolated execution of code, (ii) authenticity of the executed software, (iii) integrity of the run-time state, (iv) confidentiality of data and persistent storage, and (v) the ability to provide remote attestation to external verifiers.

The cornerstone of a TEE is isolation, commonly realized through a separation kernel or a comparable hardware-assisted mechanism. This allows the coexistence of multiple execution contexts with different security levels, starting from hardware-enforced partitions. When partitions are isolated, the attack surface is reduced to the communication channels across them, which must be carefully designed to prevent leakage or privilege escalation. Separation kernels are typically lightweight to minimize the trusted computing base (TCB) and are built to uphold the four classical principles of separation: isolation, controlled information flow, fault containment, and resource independence.

A recurring question in the literature is whether the terms *TEE* and *secure enclave* are interchangeable. While related, they are not identical. A TEE is a broader concept describing the overall trusted execution environment with its security guarantees and interfaces, whereas a secure enclave is a specific realization of this concept that isolates individual applications or processes within a protected region of execution. In practice, all enclaves are TEEs, but not all TEEs are enclaves: for example, ARM TrustZone provides a TEE based on a two-world model without process-level enclaves, whereas Intel SGX implements enclave-based TEEs.

### 2.2. RISC-V Security Landscape

In this subsection we outline the RISC-V features that are most commonly leveraged by enclave and TEE proposals, focusing on those that recur across the systems analyzed in this survey rather than attempting to cover the full ISA. A number of architectural elements have become central to recent designs, particularly those related to privilege separation and memory isolation. At the core of the model are three privilege levels—user (U), supervisor (S), and machine (M)—which establish a hierarchical control structure for executing software with different trust assumptions. Physical Memory Protection (PMP) and its enhanced variant (ePMP) extend this model by enabling fine-grained control over memory regions and enforcing access permissions, forming the foundation for most RISC-V enclave designs. Virtual memory adds further flexibility by allowing page-based isolation and translation for secure contexts. Additional extensions, including the hypervisor extension, cryptographic instructions, and memory tagging, strengthen RISC-V's ability to enforce isolation and mitigate certain classes of attacks. Together, these features constitute the architectural substrate on which RISC-V TEEs and secure enclaves are built. In the next section, we build on this foundation to develop a taxonomy of RISC-V TEE and secure enclave proposals and analyze how different designs leverage these mechanisms to achieve isolation, security, and trust.

## 3. Survey of RISC-V TEEs and Enclave Architectures

Over the past few years, a rich body of work has emerged in both academia and industry exploring TEEs and secure enclaves for RISC-V. These proposals reflect diverse design goals, ranging from lightweight isolation for embedded systems to scalable enclave models for cloud-class platforms. In this section, we provide a comprehensive survey of these architectures, summarizing their main design choices, strengths, and limitations. For each system, we discuss the security mechanisms and isolation primitives it employs, the scope of workloads it targets, and the trade-offs it introduces in

 doi:10.20944/preprints202510.0152.v1



terms of performance, scalability, and programmability. Some of the claims reported here correspond to evaluations presented by the original authors or subsequent comparative studies, while others are based on our own analysis of the designs. By combining these perspectives, we aim to provide a balanced and critical account of the state of these proposals.

### *3.1. Keystone*

Keystone [25] is designed as a flexible and extensible framework for TEEs, enabling developers to customize the TCB according to their specific security and performance needs. By leveraging the Physical Memory Protection (PMP) features of RISC-V, Keystone ensures robust memory isolation, allowing enclaves to operate securely even in the presence of an untrusted operating system. Keystone is structured around three primary components: the Secure Monitor (SM), which operates in M-mode; the Enclave Runtime (RT), which functions in S-mode; and the enclave applications (EApps), which execute in U-mode.

The Secure Monitor (SM) plays a crucial role in the Keystone framework, as it is the sole component that operates in M-mode. Its primary responsibility is to maintain the separation between the OS and the enclave, ensuring that neither can interfere with the other. The SM achieves this by controlling access to physical memory, allowing only the currently executing entity—either the host or the enclave—to access it. Furthermore, the SM enhances the Supervisor Binary Interface (SBI) to provide management services for enclaves to both the host and the enclave. The SM ensures a clean context switch by flushing enclave states, effectively preventing cache-based side-channel attacks through cache partitioning. Additionally, enclaves can be encrypted, and their page tables can be self-managed, rendering subtle attacks, such as controlled side-channel attacks, infeasible.

The runtime component is tasked with managing applications that run within the enclave (referred to as EApps). It serves as a supervisor for these applications, providing essential services such as system calls, memory management, and communication with the SM through the SBI. The reference implementation of the runtime is known as *Eyrie*, which delivers fundamental kernel functionalities. Developers have the option to modify Eyrie or create alternative runtime implementations to better suit the specific requirements of their EApps. The RT operates in S-mode within the enclave and is considered trusted by the enclave application, ensuring that it can effectively manage the enclave's operational needs while maintaining security. The enclave application (EApp) operates at the user level, executing sensitive application logic. Enclave applications are generally organized into two categories: regular code and sensitive code, ensuring that only the critical functions are executed within the secure enclave environment.

Another critical aspect of TEE deployment is the TCB, which should ideally be kept minimal. Keystone's TCB encompasses the entire SM, all M-mode code (including the bootloader and SBI implementation), and arguably the runtime. Although the runtime can be streamlined to provide only essential services to the EApp, the SM and other M-mode firmware cannot be significantly reduced. According to Lee et al. [25], Keystone's TCB consists of thousands of lines of code, in stark contrast to TrustZone's TCB, which comprises millions of lines. The hardware requirements for Keystone are minimal, necessitating only a standard RISC-V core, a secure method for storing device keys, and a secure bootloader. Notably, Keystone's open-source nature allows for greater transparency and adaptability.

According to Lee et al. [25], Keystone provides critical capabilities such as secure boot, remote attestation, and secure key provisioning to the Chain of Trust (CoT). The secure boot mechanism in Keystone lays the groundwork for a trusted system by verifying the integrity of the boot sequence. This is accomplished through either software or hardware implementations of a root of trust, which generates a new attestation key using a secure random source during the boot or reset phase. Each step of the boot process is validated by generating a hash and comparing it against a cryptographic signature. If any integrity verification fails, the boot process is halted, ensuring that the system remains secure only if the boot completes successfully. Measurement and attestation in Keystone are performed by the SM using the provisioned attestation key. During runtime, enclaves can request

 doi:10.20944/preprints202510.0152.v1



a signed attestation from the SM, which is then linked to a secure channel through a standardized protocol. Keystone provides further functionalities to meet the requirements of a TEE. It enables secure enclaves to access read-only timer registers maintained by the hardware via the `rdcycle` instruction. Additionally, it supports monotonic counters by preserving a minimal counter state within the SM memory. These features empower the SM to implement functionalities such as sealed storage, trusted timers, and rollback protection.

Sahita et al. [32] assert that the design of Keystone enclaves relies on contiguous memory allocation, which poses challenges for scalability in managing enclave memory after the boot process. Each enclave necessitates a dedicated PMP entry, allowing the architecture to support up to $N^2$ enclaves, contingent on the availability of $N$ hardware PMP registers. According to Dessouky et al. [75], Keystone's design enables it to defend against specific side-channel attacks that exploit shared resources, such as cache or memory structures. However, the separation of enclaves from the OS means they are not treated as standard processes, necessitating the preservation and restoration of the OS state during enclave operations, which can introduce latency. Additionally, the requirement for a dedicated runtime for each enclave can lead to increased development complexity and redundancy, as multiple runtimes may implement similar functionalities. Dessouky et al. [75] also note that in Keystone, the enclave runtime can potentially incorporate device drivers to facilitate secure I/O operations with peripherals. However, the framework does not support a direct binding between enclaves and peripherals, which limits the ability of DMA-capable devices to securely interact with enclave memory. Consequently, this design choice leaves enclaves vulnerable to DMA attacks.

To enhance performance, Keystone implements way-based cache partitioning for shared L2 caches, assigning entire cache ways to the processor cores executing enclaves. This approach, while effective, may lead to inefficient cache usage, as cache lines not utilized by an enclave remain inaccessible to other software components. Furthermore, Dessouky et al. [75] emphasize that Keystone can mitigate certain hardware attacks, such as bus snooping, provided that the combined footprint of the enclave (EApp and runtime) and the SM fits entirely within on-chip scratchpad memory. Anh-Tien et al. [76] note that despite the resilience of Keystone against certain attacks, the shared L1 cache of the untrusted operating system remains a potential vulnerability. Cache leakage may still occur following speculative execution, particularly if Keystone operates on an out-of-order processor like BOOM. This raises concerns regarding the potential for cross-address-space Spectre attacks within the Keystone system. In such scenarios, the victim would be confined to the enclave, while the attacker would have unrestricted access to the broader environment.

Despite its strengths, the assumptions made in the design regarding the reliability of the SM, RT, and EApps raise concerns; the framework presumes these components are free from bugs, which may not hold true in practice. While the SM is designed to be small enough for formal verification, the RT's complexity could increase as more features are added, potentially introducing vulnerabilities. According to Donnini et al. [77], from a security perspective, Keystone does not provide comprehensive defenses against certain attack vectors, such as speculative execution and side-channel attacks, placing the onus of protection on developers. Additionally, the framework lacks robust defenses against side-channel attacks involving off-chip components, which could be addressed through techniques like Oblivious RAM. Lastly, the absence of non-interference guarantees for the SBI exposes the system to risks, including Iago attacks, as the RT can invoke untrusted system calls from the operating system, further complicating the security landscape for developers working with Keystone.

### *3.2. Sanctum*

Sanctum [26] provides a framework for strong isolation of concurrently executing software modules, effectively protecting against a significant category of software attacks that exploit memory access patterns to extract sensitive information. Sanctum introduces isolated execution environments known as enclaves, designed to protect sensitive applications on RISC-V architectures. Each enclave operates at the user level and is paired with a non-sensitive application that invokes it.

 doi:10.20944/preprints202510.0152.v1



Sanctum enforces enclave isolation through minimal hardware modifications to the Page Table Walker (PTW) within the Memory Management Unit (MMU). These changes ensure that the OS cannot access enclave memory and that enclaves are prevented from accessing OS memory or other enclaves by altering their page tables. The custom PTW is designed to block successful address translations for virtual memory addresses that do not correspond to the allowed physical memory addresses for the current execution context. The critical security functions of Sanctum are managed by a software component known as the Security Monitor (SM), which constitutes the system's TCB. The SM operates at the machine level of the RISC-V processor and undergoes verification during a secure boot process. While the operating system and its associated software are excluded from the TCB—loaded post-secure boot and not subjected to measurement—Sanctum enables the establishment and remote attestation of secure enclaves. This mechanism effectively initiates trust in additional software components, as the SM utilizes its cryptographic keys to measure and sign the secure enclaves. Importantly, the SM's authentication allows remote parties to dismiss attestations from systems that have loaded known vulnerable versions of the monitor, thereby enhancing the overall security posture. Additionally, Sanctum's basic DMA protection is implemented through the inclusion of two registers in the memory controller.

The architecture incorporates a hardware True Random Number Generator (TRNG) and Physically Unclonable Functions (PUFs) to establish strong isolation through enclaves and defend against various software threats, including cache timing and passive address translation attacks. According to Anders et al. [61], Sanctum features minimal hardware modifications and has been successfully implemented on the Xilinx Zynq-7000 FPGA platform. Costan et al. [26] further note that Sanctum cores maintain the same clock speed as their non-secure counterparts, since there are no changes to the critical execution path of the CPU core. The architecture relies on the untrusted OS for managing enclave memory and providing essential services like interrupt handling and I/O operations. This reliance poses a risk, as a compromised OS could potentially execute controlled side-channel attacks against the enclaves. For instance, the adversary might deduce information about the enclave's internal state by monitoring the enclave's page tables or by generating frequent interrupts. To counter these threats, Sanctum isolates the enclave's page tables within its memory and equips enclaves with the capability to detect and respond to unusual interrupt patterns.

Sanctum employs two primary strategies to mitigate cache side-channel attacks: first, it flushes sensitive processor resources, including the L1 cache and the Translation Lookaside Buffer (TLB), during every enclave context switch; second, it partitions the shared L2 cache using a memory page coloring technique, which allocates specific cache lines exclusively to enclaves. However, Dessouky et al. [75] affirm that the effectiveness of this cache partitioning is limited in practice, as it requires all software components to conform to the coloring scheme. Consequently, achieving this partitioning necessitates a complete rearrangement of the OS memory layout at runtime, which is often impractical.

Dessouky et al. [75] also assert that the enclaves in Sanctum consist of unprivileged user-level code, which restricts their ability to establish secure connections to peripherals such as sensors or GPUs that require privileged driver code. While Sanctum includes basic protections against DMA attacks by limiting access to a designated memory region, this feature is relatively rudimentary.

Wong et al. [62] stress that Sanctum's design is exclusively oriented towards mitigating software threats and does not extend its protective measures to physical attacks, which incur substantial hardware costs and performance penalties. Krentz and Voigt [63] emphasize that Sanctum necessitates resource-intensive attestation processes and does not prioritize the reduction of its communication latency. According to Cheang et al. [64], Sanctum does not provide formal verification for its hardware components. Nasahl et al. [29] further assert that Sanctum lacks architectural provisions for secure input/output operations, resulting in unprotected interactions with peripheral devices.

### *3.3. TIMBER-V*

TIMBER-V [27] achieves memory tagging through modifications to the Memory Protection Unit (MPU), which enforces isolation between all processes and between the normal and trusted domains

 doi:10.20944/preprints202510.0152.v1



of the executing process, allowing for a clear separation of user mode (U-mode) and supervisor mode (S-mode) within both realms. The normal domains (N-domains) maintain the traditional U-mode and S-mode structure, permitting existing applications to operate without requiring modifications. Memory words in the N-domains are tagged with the *N* tag, while those in the trusted domains (T-domains) are assigned the *TU* and *TS* tags, corresponding to U-mode and S-mode, respectively. The architecture supports isolated execution environments, termed enclaves, within the trusted user mode (TU-mode), while the trusted supervisor mode (TS-mode) runs the TagRoot trust manager, which enhances the untrusted operating system with trusted functionalities. Additionally, TagRoot is responsible for enclave setup and offers essential services, including secure shared memory for communication with the normal domain, as well as functionalities for sealing and attestation. To protect against Direct Memory Access (DMA) attacks, additional tag engines are required for each peripheral with DMA capabilities.

Transitioning from N-domains to T-domains is facilitated through trusted callable entry points, marked with the TC tag. TIMBER-V employs a two-bit tagging system for each 32-bit memory word, allowing for four distinct tags. The architecture enforces strict rules for tag updates, permitting changes only within the same or lower security domains, thereby preventing privilege escalation. TS-mode and machine mode (M-mode) have unrestricted access to all tags, while TU-mode can only modify tags between N-tag and TU-tag, supporting dynamic interleaving of user memory. Notably, TU-mode is restricted from altering TC-tags, which are designated for secure entry points. However, these advanced features rely on specialized hardware enhancements, which may compromise the native execution of enclave applications. According to Feng et al. [74], TIMBER-V introduces a significant performance overhead, averaging 25.2%, and does not address memory integrity protection.

The MPU in TIMBER-V enhances label isolation, effectively isolating each process while minimizing memory overhead. This design significantly mitigates memory fragmentation and promotes the dynamic reuse of untrusted memory across security boundaries. Beyond stack interleaving, TIMBER-V also enables innovative execution stack sharing across various security domains. The architecture is designed to be compatible with existing software and supports real-time operational constraints. A proof-of-concept implementation of TIMBER-V has been successfully evaluated using a RISC-V simulator [27].

Dessouky et al. [75] assert that while TIMBER-V enables fine-grained enclave creation for embedded systems, it currently lacks secure communication pathways between enclaves and external peripherals, such as sensors. Although it is theoretically possible to integrate drivers and services into the TagRoot to facilitate this communication, doing so would align TIMBER-V with high-level security models that have been criticized for significantly expanding the system's attack surface. Furthermore, TIMBER-V does not incorporate protective measures against cache side-channel attacks and remains susceptible to interrupt-based controlled side-channel attacks, as the handling of enclave interrupts is managed by the operating system.

### *3.4. MI6*

MI6 [28] establishes a secure enclave framework characterized by robust microarchitectural isolation, ensuring that the enclave remains entirely segregated from the rest of the system at the microarchitectural level. Specifically, MI6 enhances the isolation guarantees of Sanctum by integrating hardware support into the RiscyOO out-of-order core, thereby extending its capabilities to address a broader range of threats, including side-channel and speculative execution attacks. To effectively counter these vulnerabilities, MI6 introduces a dedicated `purge` instruction designed to clear sensitive information from microarchitectural buffers and the L1 Data Cache prior to context switches. Furthermore, the operating system can only communicate with the enclave via the API managed by the Security Monitor (SM), a trusted software component operating at a higher privilege level. However, it is important to note that the design does not fully mitigate the D1 and D2 vulnerabilities, as these issues cannot be resolved solely through the flushing process.

 doi:10.20944/preprints202510.0152.v1



MI6 implements hardware modifications to establish strong isolation through the spatial and temporal partitioning of resources. This approach involves allocating resources to protection domains without regard to their usage demands; for instance, last-level caches (LLCs) and DRAM bandwidth are divided among protection domains, with each domain limited to a specific fraction of the DRAM controller bandwidth, regardless of the memory usage of co-located domains. Additionally, MI6 shares similar limitations with Sanctum, as it employs an LLC partitioning strategy that does not scale effectively.

While MI6 ensures security, the functionality of the enclave is somewhat constrained, particularly due to a significant drawback: the absence of shared memory with external systems. Allowing shared memory access with the operating system would compromise the strong microarchitectural isolation between trusted and untrusted environments, potentially exposing the system to transient execution vulnerabilities such as Spectre [65], which leverage speculative execution paths in out-of-order processors to extract sensitive information through side channels. Consequently, the lack of memory sharing limits enclaves to performing isolated batch computations, restricting their ability to engage in external interactions and thereby narrowing the scope of potential applications.

According to Li et al. [66], MI6 is exclusively designed for the RiscyOO processor and lacks generalizability; it is dependent on the unique characteristics of the RiscyOO baseline processor and does not include mechanisms to effectively clear replacement tags in caches and Translation Lookaside Buffers (TLBs). Furthermore, MI6 fails to address the cleaning of certain microarchitectural states (such as the issue queue), which could be vulnerable to emerging attack vectors. Also, the flushing process in MI6 does not ensure the writing back of dirty cache lines, rendering it incompatible with caches that utilize a write-back policy, which is prevalent in contemporary processors.

While MI6 utilizes both spatial and temporal isolation strategies, it does not address the verification challenges associated with formally proving isolation. The MI6 processor incorporates the `purge` instruction designed to clear microarchitectural state. Nevertheless, achieving comprehensive cleansing of all CPU states presents a significant challenge, heavily reliant on the intricate implementation specifics of the CPU.

In MI6, the overarching solution for the entire system can adversely impact the performance of standard applications with substantial memory demands. Frequent enclave executions necessitate barriers at each context switch, which can degrade the performance of regular applications, potentially deterring the adoption of enclaves. MI6 does not protect against adversaries executing concurrently on the same processor. The authors clarify that their "isolation mechanisms exclusively address software attacks" [28], and they do not provide defenses against denial-of-service (DoS) attacks. Furthermore, they explicitly state that they do not account for threats such as DRAM bit flipping.

### *3.5. HECTOR-V*

In HECTOR-V [29], the design is based on two key innovations: the integration of a heterogeneous architecture that distinctly separates the Rich Execution Environment (REE) and Trusted Execution Environment (TEE) domains, and the introduction of a security-hardened RISC-V Secure Co-Processor (RVSCP) aimed at enhancing resilience against side-channel attacks (SCAs). HECTOR-V introduces a novel secure I/O path mechanism to manage device sharing and protect against unauthorized access. This mechanism employs an identifier-based strategy, where each device and processing core is assigned a unique identifier. This identifier is integrated into the communication system, allowing for fine-grained access control. Each transaction is validated by the Security Monitor (SM) module based on these IDs, rejecting any unauthorized access attempts. The core processor ID is hard-coded in hardware, while process and peripheral IDs are assigned dynamically at runtime. This hard-coding ensures that attackers cannot alter the core IDs.

In HECTOR-V, managing access permissions is done using a hardware-based Security Monitor. Only one SM owner is permitted at any given time, and this owner is responsible for defining access rights to resources. The secure boot process is executed by granting exclusive access to the secure storage to the first virtual core of the TEE (VC0). This access right is permanently hard-coded and

 doi:10.20944/preprints202510.0152.v1



solely owned by VC0, while other virtual TEE cores can only retrieve code from claimable Block RAM (BRAM). Upon reset, the reset unit designates VC0 as the SM owner, keeping the REE processors in a halted state. VC0 then executes the Zero Stage BootLoader (ZSBL) from the secure storage, initiating the first authentication of the system, which serves as the Root of Trust (RoT) for HECTOR-V. Following this, the ZSBL configures the Memory Protection Unit (MPU) for external memory access rights, including those for the Secure Digital (SD-card) and Double Data Rate (DDR) memory. VC0 subsequently verifies the hash value of the Berkeley BootLoader (BBL) against the expected value stored in secure storage. If the verification is successful, the BBL is loaded into the main memory, and VC0 releases the SD-card driver along with the claimed DDR memory regions. Finally, VC0 transfers SM ownership to the REE processors and triggers the reset unit to initiate the REE. However, since the secure boot program remains accessible from the RVSCP and both the REE and TEE share the same system-on-chip (SoC), there exists a potential risk of exposing the RoT to vulnerabilities from the REE side, despite the implementation of secure storage elements. The architecture does not incorporate dedicated hardware computation units for cryptographic algorithms, which may result in a less efficient secure boot process. This absence of specialized cryptographic support could lead to increased latency during the secure boot sequence, as the system relies on general-purpose processing capabilities rather than optimized hardware acceleration for cryptographic operations. Consequently, this design choice may impact the overall performance and responsiveness of the secure boot mechanism within the HECTOR-V framework.

The authors argue that the duplication of resources in HECTOR-V effectively mitigates cache and microarchitectural side-channel attacks by ensuring that sensitive components are not shared between the secure and non-secure domains. However, they acknowledge that this approach may not be entirely feasible in real-world applications where resource utilization constraints are significant. Additionally, HECTOR-V does not support the integration of programmable resources, which are crucial for SoC-FPGAs. Furthermore, while HECTOR-V provides a dedicated processor for secure applications, it lacks the capability to create multiple secure domains, limiting its flexibility in complex security scenarios.

### *3.6. CURE*

CURE [30] is a robust TEE architecture that leverages innovative hardware security primitives within the RISC-V framework. This architecture supports the coexistence of various enclave types, including kernel-space, user-space, and sub-space enclaves, within a single system. To facilitate this, CURE incorporates three key hardware enhancements: dedicated core registers for monitoring enclave execution, a system bus arbiter to manage access control for bus transactions, and a mechanism for partitioning the shared cache. These enhancements collectively bolster the isolation of enclaves and provide defenses against side-channel attacks (SCAs). The Security Monitor (SM) in CURE functions as a sub-level enclave. A significant benefit of implementing the SM as a sub-level enclave is the substantial reduction in the system's Trusted Computing Base (TCB), as it excludes all non-security-related code at the machine level. This flexibility allows developers to select the enclave type that best aligns with the requirements of their sensitive applications without needing to modify the application to fit a specific enclave model.

The user- and supervisor-level enclaves in CURE can integrate device drivers into their execution environment. Coupled with CURE's hardware security features, this capability facilitates the exclusive assignment of peripherals to specific enclaves, known as enclave-to-peripheral binding. The system utilizes enclave IDs stored in the core registers, which are propagated throughout the architecture to track the active enclave on each core. These IDs are established during the enclave's initialization, termination, and context-switching processes. The bus arbiter evaluates access permissions based on the enclave ID for each memory access request, redirecting any unauthorized transactions to a designated area, thereby preventing their execution. In terms of enclave-to-peripheral interactions, CURE ensures that memory access is strictly regulated, eliminating the need for encryption or authentication in communications between enclaves and peripherals. To address SCAs, CURE employs two primary

 doi:10.20944/preprints202510.0152.v1



strategies: flushing the L1 cache and implementing a way-based partitioning approach for the L2 cache. However, Stapf et al. [67] note that the allocation of entire cache ways to enclaves may result in suboptimal cache utilization due to the coarse granularity of cache ways.

A pivotal hardware security feature of CURE is the filter engine integrated into the system bus. This filter engine serves dual purposes: it allows for the exclusive assignment of memory regions to enclaves and establishes access controls for peripherals, determining which enclaves can communicate with them via Memory-Mapped I/O (MMIO). Furthermore, Stapf et al. [67] point out that the filter engine incorporates registers and control logic for all Direct Memory Access (DMA)-capable devices, restricting their access to designated memory areas. This mechanism facilitates secure communication between enclaves and DMA-capable devices without requiring encryption.

CURE's software TCB, represented by the SM, is designed to be minimal, encompassing only the security-critical code while omitting the standard firmware code typically present at the machine level. The SM is responsible for managing enclaves and executing all security-sensitive operations, such as enclave binary verification, key management, and persistent storage of enclave states. However, Kuhne et al. [36] aver that CURE necessitates hardware modifications to implement security features, including the introduction of a filter engine that enforces access controls at the system bus level and the integration of a unique enclave identifier within the CPU architecture to manage enclave operations effectively.

According to Schneider et al. [69], a notable limitation of CURE is that its attestation mechanisms do not encompass peripheral devices, which could expose vulnerabilities in the communication between enclaves and hardware components. Furthermore, the design of kernel-space enclaves within CURE mandates that they operate on dedicated CPU cores, which, as far as current knowledge indicates, do not have the capability to relinquish control back to the operating system. This design choice may lead to inefficient resource utilization, as these cores remain idle while awaiting data from peripheral devices, potentially hindering overall system performance. Regarding the Root of Trust (RoT), CURE does not establish a RoT mechanism but presumes that the secure boot sequence is completed upon system reset, with the initial bootloader in ROM responsible for verifying and loading the firmware, including the Secure Monitor (SM), into the appropriate Random Access Memory (RAM). Consequently, Kieu-Do-Nguyen et al. [69] assert that in terms of RoT-based secure boot processes, CURE does not offer any innovative solutions beyond the traditional reliance on hard-coded keys stored in ROM, which may limit its adaptability to more dynamic security requirements. Additionally, Chen et al. [70] emphasize that CURE lacks dedicated hardware support for accelerating cryptographic algorithms, which results in reduced efficiency for encryption and decryption processes.

### *3.7. CoVE*

CoVE [32] serves as a foundational architecture for confidential computing tailored for RISC-V platforms, with its secure execution environment referred to as a TEE Virtual Machine (TVM). Central to this architecture is the TEE Security Manager (TSM) driver, which operates in M-mode—the highest privilege level in RISC-V—facilitating transitions between confidential and non-confidential operational contexts. The TSM driver is responsible for managing memory page allocations to TVMs via the Memory Tracking Table (MTT), ensuring proper isolation and security. It also plays a crucial role in measuring and initializing the TSM, which acts as a trusted intermediary between the hypervisor and the TVMs. The CoVE Trusted Computing Base (TCB) is primarily composed of the TSM, which serves as the intermediary for security enforcement between Trusted Execution Environments (TEEs) and non-TEE elements, alongside hardware components that uphold the confidentiality and integrity of data in use. Consistent with other frameworks, the hypervisor remains untrusted, tasked with the management of resources across all workloads, encompassing both confidential and non-confidential applications. CoVE specifies an Application Binary Interface (ABI) that allows the hypervisor to invoke virtual machine management functions from the TSM.

The architecture employs a multi-layered attestation framework, starting from the hardware and extending through the TSM driver, TSM, and TVM. Each layer undergoes a process of loading,

 doi:10.20944/preprints202510.0152.v1



measurement, and certification by its predecessor, establishing a robust chain of trust for system integrity verification. The TVM can request a certificate from the TSM, which includes attestation evidence linked back to the hardware, thereby providing a reliable method for confirming the authenticity of the TVM and its operating software. CoVE employs cryptographic mechanisms to protect confidential memory against physical access, ensuring confidentiality, integrity, and replay protection. This includes the use of separate cryptographic keys for different TVM workloads to enhance security.

While CoVE aims to minimize the TCB, the introduction of new hardware primitives and ISA extensions increases the complexity of the overall system design and implementation. Although the architecture is designed for performance, the additional layers of isolation and security mechanisms introduce some overhead, particularly in scenarios with frequent context switching or resource allocation. The authors note that a common drawback of existing approaches, including CoVE, is the lack of support for confidential I/O, which remains an ongoing area of work [32].

### *3.8. WorldGuard*

The RISC-V WorldGuard (WG) [33] architecture enhances isolation by implementing a comprehensive system-wide approach through the concept of *Worlds*, which serve as distinct execution contexts encompassing both agents (components that initiate transactions) and resources (components that respond to transactions). Each World is uniquely identified by a hardware World Identifier (WID), with the total number of unique WIDs being platform-specific, defined by the parameter `NWorlds`, and limited to a maximum of 32 Worlds. The WG architecture is designed to facilitate the static allocation of agents and resources to Worlds, typically managed by M-mode firmware or a Trusted Execution Environment (TEE) during the system boot process. WorldGuard allows for the dynamic transfer of Security Monitor (SM) ownership among different parties, facilitating a broader range of use cases. Nasahl et al. [29] posit that WorldGuard's architecture emphasizes a generalized approach to managing access permissions. This flexibility in ownership and access management positions WorldGuard as a versatile solution for creating secure execution environments within heterogeneous architectures.

WorldGuard integrates Physical Memory Protection (PMP) and Physical Memory Attributes (PMA) within the RISC-V Instruction Set Architecture (ISA). Hoang et al. [71] assert that WorldGuard operates with shared processors for both the TEE and Rich Execution Environment (REE). This architecture is designed to provide a more robust separation of execution contexts, thereby mitigating potential security risks. However, it is important to note that WorldGuard does not focus on optimizing the secure boot process; instead, it aims to augment the existing TEE models. As such, it employs a traditional boot flow for secure initialization, relying on a bootloader that contains hard-coded root keys stored in Read-Only Memory (ROM). This bootloader is the first component executed, tasked with verifying and loading the secure channel into the main memory, ensuring that both the boot program and the Root of Trust (RoT) remain within the TEE domain. Consequently, Hoang et al. [71] note that while the RoT and boot program remain within the TEE domain, the potential for attack persists. Although WorldGuard's bootloader program is accessible, details regarding its hardware implementation remain undisclosed. Pinto et al. [72] emphasize that while WorldGuard enhances isolation, it still retains vulnerabilities to conventional software-based side-channel attacks, as the foundational elements of the secure boot process are not inherently fortified against such threats. Notably, the specification does not encompass efficient mechanisms for dynamic reconfiguration of Worlds while the system is operational, which remains outside its current scope.

In contrast to the RISC-V (e)PMP, which enforces access control through a set of memory configuration rules directly at the hart level, WG adopts a more flexible approach. It does not prescribe a specific method for the propagation and verification of WIDs across the platform or bus, leaving these implementations to be platform-specific. Consequently, various platforms may adopt different strategies for checking WIDs, with bus fabrics implementing support for WIDs in ways that are tailored to their specific architectures.

 doi:10.20944/preprints202510.0152.v1



### 3.9. SPEAR-V

SPEAR-V [34] employs a lightweight memory tagging mechanism to enforce fine-grained access control for enclave memory. Unlike some existing architectures that necessitate extensive modifications to the operating system, SPEAR-V integrates seamlessly with unmodified OS-managed paging structures, allowing for efficient memory management. The architecture utilizes 24-bit memory tags, which are sufficient for its security requirements while minimizing overhead.

SPEAR-V effectively mitigates same-core side-channel attacks such as branch shadowing, but it does not specifically address cross-core side-channel vulnerabilities, which may limit its effectiveness in multi-core environments. Additionally, the architecture's reliance on a single-core processor means that it does not consider the complexities introduced by multi-core systems in its baseline design. While SPEAR-V does not specifically target physical threats such as memory-bus snooping or malicious DRAM modifications, it allows for the integration of orthogonal techniques such as memory encryption to enhance security against certain physical vulnerabilities. This indicates a flexible approach to security, albeit with the acknowledgment that physical attack defenses are outside the primary scope of the architecture.

SPEAR-V also supports dynamic memory allocation and arbitrary nesting of enclaves, enhancing its scalability and flexibility for various applications. However, despite its robust design, the architecture does not provide explicit defenses against memory corruption or code reuse attacks, which could still pose risks if vulnerabilities exist in the enclave code.

### 3.10. VirTEE

VirTEE [35] is built on a secure platform infrastructure that divides physical memory into enclaves, utilizing specific registers for memory management instead of traditional page tables. This innovative approach allows for the support of large enclave sizes and efficient memory access control, leveraging the RISC-V hypervisor extension to facilitate virtualization. VirTEE enhances security by implementing cache partitioning, ensuring that each CPU core executing an enclave has its own last-level cache that is not shared with other cores. This design mitigates the risk of sensitive information leakage through cache timing attacks.

The architecture provides strong physical enclave memory isolation, protecting enclave memory from unauthorized access by malicious software, including the operating system. VirTEE's ability to support large enclave sizes is beneficial for running multiple virtual machines and applications simultaneously without significant overhead. The architecture demonstrates moderate performance, as evaluations indicate that it incurs only a modest performance overhead on standard benchmarks and real-world applications, making it suitable for practical deployment in cloud environments.

However, VirTEE also has its limitations. It is heavily dependent on specific hardware configurations, which may restrict its deployment in environments lacking the necessary infrastructure. Additionally, while it offers resilience against side-channel attacks, it does not protect enclaves from memory-corruption attacks, leaving a potential vulnerability. Furthermore, the architecture assumes that peripherals, such as hard drives, are accessible to adversaries, which could lead to data leaks. Lastly, denial-of-service (DoS) attacks are outside the scope of VirTEE's guarantees, as most TEE architectures do not provide assurances regarding availability.

### 3.11. DORAMI

DORAMI [36] leverages the enhanced Physical Memory Protection (ePMP) feature of RISC-V, allowing for fine-grained control over memory access and isolation between different execution modes (P, F, S/U). This design minimizes the need for significant hardware changes, facilitating easier adoption in existing systems. However, the architecture's effectiveness is contingent on the availability of ePMP support, which is not present in all RISC-V platforms, thus limiting its applicability. DORAMI enforces strict intra-mode isolation between the Security Monitor (SM) and firmware, which helps mitigate risks associated with shared resource exploitation. By restricting access to PMP registers solely

 doi:10.20944/preprints202510.0152.v1



to the SM, the architecture reduces the potential for side-channel attacks that could manipulate memory protection settings. Nonetheless, the complexity of managing PMP configurations still introduces vulnerabilities when not handled properly.

DORAMI's reliance on hardware-based memory isolation via PMP provides a layer of defense against unauthorized access or manipulation of memory regions. The compartmentalization of the SM from the firmware further limits the impact of physical attacks on the firmware. However, the architecture primarily focuses on memory isolation and does not comprehensively address all aspects of physical security, such as tamper resistance. DORAMI is adaptable to various RISC-V platforms, including those with only standard PMP support, enhancing its scalability across different hardware configurations. It is designed to manage multiple enclaves, allowing for deployment in diverse security environments. However, as the number of compartments and enclaves increases, managing their interactions and ensuring security becomes complex, potentially impacting scalability.

DORAMI aims to achieve its security goals with minimal performance penalties, ensuring that applications running in the RISC-V environment do not experience significant slowdowns. The architecture modifies PMP configurations during context switches to maintain performance while enforcing security. However, the implementation on certain platforms, such as Rocketchip, has shown performance variability, indicating that further optimizations are needed to ensure consistent performance across different environments. Additionally, while designed to minimize overhead, the modifications during context switches introduce latency, particularly in high-frequency switching scenarios.

### *3.12. Elasticlave*

Elasticlave [78] enhances the enclave architecture by enabling each enclave to manage multiple physical memory regions that can be selectively shared with other enclaves. This design allows an enclave to request access to another enclave's memory regions, which the owner can grant, thereby facilitating more efficient inter-enclave communication without the overhead of data copying or encryption. Elasticlave can be implemented on RISC-V and is designed to maintain a relatively simple hardware complexity, requiring only a privileged Security Monitor (SM) that spans approximately 7,000 lines of code. This simplicity contrasts with traditional TEEs, which often necessitate more complex hardware setups. The architecture is also adaptable, as it can be integrated into various TEE implementations, including Intel SGX and ARM TrustZone, although this may involve specific changes to accommodate different memory management models.

Elasticlave permits an enclave to dynamically adjust permissions for shared memory regions, allowing other enclaves to write to these regions. However, the authors highlight that this flexibility necessitates careful management of write permissions to mitigate potential interference between enclaves, ensuring that data integrity is maintained during concurrent access. The architecture requires enclave programs to invoke a `map` operation to access shared memory regions, which must then be approved by the owner through a `share` operation. Kuhne et al. [36] state that this explicit permission model enhances security but also adds complexity to the programming model. Feng et al. [74] maintain that this sharing mechanism lacks flexibility. Specifically, the enclave must identify the target enclave by its ID, which requires that the other enclave be created beforehand and that its ID be communicated to the sharing enclave. Additionally, the shared data is tightly coupled to the lifecycle of the enclave; if the sharing enclave terminates, the data is lost. Elasticlave introduces a new `share` operation that allows an enclave to designate a contiguous range of virtual memory addresses for sharing with another enclave identified by a specific ID. To minimize the performance costs associated with data transfer, Elasticlave assumes a common encryption key across all enclaves. However, Feng et al. [74] claim that this assumption raises security concerns, as it undermines the cryptographic separation typically maintained between enclaves. Furthermore, they argue that this design is not compatible with existing TEE frameworks, such as AMD SEV and Intel TDX, which rely on the principle that each enclave utilizes its own unique encryption key. According to Feng et al. [74], the rigidity of Elasticlave's sharing model leads to several limitations: the sharing enclave must be aware of the target

 doi:10.20944/preprints202510.0152.v1



enclave's ID, the target enclave must be instantiated prior to the sharing operation, and the shared data is rendered inaccessible if the sharing enclave is terminated.

Pan et al. [73] assert that Elasticlave enhances the efficiency of memory sharing between enclaves but faces challenges in scalability due to the fixed number of Physical Memory Protection (PMP) registers, which restricts the number of simultaneously protected memory regions. They argue that while Elasticlave allows for improved performance in data-sharing workloads, its reliance on the RISC-V PMP architecture limits the number of enclaves that can operate concurrently, potentially impacting its effectiveness in high-demand cloud environments. The architecture's design facilitates inter-enclave communication through a shared memory model, yet the constraints imposed by the RISC-V PMP mean that Elasticlave may not adequately support applications requiring extensive concurrent enclave interactions. Pan et al. [73] emphasize that due to the limitations of the RISC-V specification, Elasticlave can only manage a limited number of memory regions, which restricts its ability to scale effectively in scenarios with numerous consumer enclaves.

Yu et al. [78] acknowledge that while Elasticlave focuses on defining a memory interface, it does not specifically address microarchitectural implementation flaws or side-channel vulnerabilities. Elasticlave does not directly address defenses against attacks on physical RAM or bus interfaces, indicating that such protections are considered orthogonal to the architecture's primary focus. Kuhne et al. [36] further discuss that while Elasticlave enables the mapping of shareable physical memory regions to an enclave's virtual address space, there are notable drawbacks. First, developers must explicitly identify which portions of their applications are shareable, often necessitating significant code restructuring to isolate these components into separate enclave memory. Second, the dynamic mapping and unmapping of memory regions require local attestation, which ensures that the newly mapped memory is in a secure and expected state. This reliance on attestation complicates the measurement properties of applications, as it ties the security guarantees of a program to the integrity of multiple physical memory regions.

### *3.13. Cerberus*

Cerberus [31] utilizes a formal verification framework to enhance the security and efficiency of enclave memory sharing. Cerberus is implemented on the RISC-V Keystone platform, which necessitates specific support for features like Physical Memory Protection (PMP) to ensure memory isolation. This reliance on particular hardware capabilities limits its applicability to platforms that do not support these features. While the authors emphasize the formal verification of the Secure Remote Execution (SRE) property, Cerberus does not provide extensive details on specific countermeasures against side-channel vulnerabilities, which remain a critical concern in enclave designs.

The architecture introduces two critical operations: `Snapshot` and `Clone`. The `Snapshot` operation effectively transforms the executing enclave into a read-only entity, while the `Clone` operation allows the creation of a child enclave that can access the same memory contents as its parent at the moment of cloning. To ensure consistent functionality, the virtual address spaces of both the parent and child enclaves must align immediately following the `Clone` operation. However, any write operations performed by the child enclave will lead to divergence from the shared memory, as these actions trigger a copy-on-write mechanism that allocates new memory for modifications. This presents a limitation for Cerberus, as the advantages of memory sharing may diminish over time as the child enclave's memory diverges from the original snapshot. Nevertheless, Cerberus proves to be particularly effective in scenarios where enclaves primarily write to a limited portion of memory while sharing the remainder. It is the responsibility of the programmer to strategically determine when to invoke the `Snapshot` operation.

The Cerberus interface is designed to integrate seamlessly with process-creation system calls, thereby reducing the startup latency associated with enclave initialization. By providing a programmable interface, Cerberus enhances the overall efficiency and responsiveness of server enclave applications, ultimately improving end-to-end latency for various use cases. Cerberus also aims to protect against physical attacks by maintaining a strong memory isolation model, but the effectiveness

 doi:10.20944/preprints202510.0152.v1



of this protection is contingent on the underlying hardware's security features. Scalability is addressed through the introduction of a single-sharing model with read-only shared memory, which simplifies the verification process and allows for efficient memory sharing across multiple enclaves. This design choice enhances performance by reducing initialization latency and minimizing computational overhead during enclave operations. However, the limitation lies in the fact that the sharing model restricts each enclave to access only one read-only shared memory, which is not suitable for all use cases requiring more complex memory sharing scenarios.

### *3.14. AP-TEE*

AP-TEE (Application Platform Trusted Execution Environment) [38] is a draft specification being developed within the RISC-V Security and Confidential Computing Working Group. AP-TEE leverages the RISC-V virtualization extensions to enable confidential virtual machines (VMs) within a TEE. Its design goal is to provide a RISC-V counterpart to confidential computing frameworks such as AMD SEV, Intel TDX, and Arm CCA. As the AP-TEE specification is not yet finalized, the description here reflects preliminary information available in the literature [38] and may evolve as the standard matures.

In AP-TEE, virtualization is divided into two domains: *Non-Confidential*, where a conventional operating system executes, and *Confidential*, which remains isolated from the former. The TSM Driver, executing in M-mode, serves as the relay between the two domains. The TEE Security Manager (TSM), which runs in HS-mode within the Confidential domain, operates as a passive software component that processes requests from both the hypervisor and the TEE Virtual Machines (TVMs).

Two types of Application Binary Interfaces (ABIs) are defined for TSM. The first is the *TH-ABI* (TEE-Host ABI), which governs the interaction between the non-confidential hypervisor and TSM. It supports operations such as TVM creation, memory page management, and execution scheduling. The second is the *TG-ABI* (TEE-Guest ABI), which defines the interface between TSM and a TVM. This ABI provides support for attestation, I/O operations, and memory management.

Unlike other RISC-V TEEs that rely on Physical Memory Protection (PMP), AP-TEE introduces the Memory Tracking Table (MTT) to distinguish between confidential and non-confidential virtual memory pages. This mechanism enables the precise identification and enforcement of isolation at the granularity of individual pages.

The AP-TEE boot sequence begins with the execution of the TSM Driver in M-mode. During this phase, the startup of the TSM Driver is measured, and its hash value is recorded in the hardware Root of Trust (RoT) for attestation purposes. Once initialized, the TSM executes in HS-mode within the Confidential domain. Both its startup and state are measured and recorded. Subsequently, the hypervisor and host OS are launched in HS-mode within the Non-Confidential domain.

To create a TEE Virtual Machine (TVM), the host OS and hypervisor interact with the Trusted Host ABI (TH-ABI). This process allocates memory within the confidential domain, initializes the virtual CPU (vCPU), and launches the TVM. Each stage—TSM Driver, TSM, and TVM—is measured and recorded, forming a chain of trust that underpins the attestation process. In other words, attestation in AP-TEE covers the entire boot sequence: *TSM Driver* $\rightarrow$ *TSM* $\rightarrow$ *TVM*.

### *3.15. Penglai*

Penglai [39] is an enclave system that adopts a hardware–software co-design to overcome scalability limitations present in prior trusted execution environments. The architecture introduces two central primitives: the *Guarded Page Table (GPT)* and the *Mountable Merkle Tree (MMT)*. The GPT enables fine-grained memory isolation at the page level by protecting host page tables within a restricted memory region, ensuring that only authorized entities can access or modify them. The MMT serves as an integrity protection structure, supporting on-demand, scalable memory encryption through a mountable hash forest. Together, these primitives allow Penglai to dynamically manage secure memory, achieving support for thousands of concurrent enclaves and scaling up to 512 GB with low runtime overhead.

 doi:10.20944/preprints202510.0152.v1



Beyond its memory model, Penglai introduces *shadow enclaves* and a *fork-style enclave creation mechanism*, which significantly reduce startup latency. Reported evaluations demonstrate reductions of up to three orders of magnitude compared to traditional enclave initialization, making the architecture well suited for short-lived cloud functions and serverless workloads. Performance measurements further indicate modest overheads—approximately 5% for memory-intensive applications—while maintaining strong security guarantees against both software-based and physical attacks. These properties position Penglai as a scalable alternative to contemporary enclave architectures, addressing a broader set of use cases in cloud and edge computing.

### *3.16. AnyTEE*

AnyTEE [40] is an open and interoperable framework for building Software-Defined Trusted Execution Environments (sdTEEs) that aims to address the fragmentation and compatibility challenges prevalent in contemporary TEE technologies. By leveraging widely available hardware virtualization extensions, AnyTEE enables the emulation and customization of diverse TEE models—such as Intel SGX and Arm TrustZone—across multiple Instruction Set Architectures (ISAs), including Arm and RISC-V.

The framework introduces a hierarchical execution model that supports nesting, composition, and fine-grained access control through nested page tables, enabling sdTEEs to coexist and interoperate on the same platform. Key innovations include configurable security policies, support for unmodified trusted applications, and enhanced isolation mechanisms such as intra-privilege memory protection.

Evaluations demonstrate that AnyTEE achieves near-native performance, with overheads below 3%, while providing strong security guarantees against software-based attacks. The framework is implemented as an open-source system, offering a portable and extensible foundation for future TEE architectures.

**Table 1.** Comparison of RISC-V TEE architectures across design and security criteria.

| System | Enclave Type | Priv. | Memory Isolation | Sec. I/O | Cache | SDK | TCB Size | Compliance | No HW Mod. | Crypto. |
|---|---|---|---|---|---|---|---|---|---|---|
| Keystone[25] | Custom Runtime | U+S | PMP | ○ | ◐ | Partial | ~8.4k LoC | ○ | ● | ○ |
| Sanctum[26] | User Enclave | U | PMP + PTW mods | ○ | ● | None | ~5k LoC | ○ | ◐ | ○ |
| TIMBER-V[27] | Sub-process | U | Tagged memory | ○ | ◐ | None | ~? | ○ | ● | ○ |
| CURE[30] | Multi-level (U/S/M) | U/S/M | Cache-tagging | ● | ● | None | Partial SM | ○ | ○ | ○ |
| Cerberus[31] | Custom PMP-based | U+S | PMP | ○ | ◐ | None | ~? | ○ | ● | ○ |
| CoVE[32] | TVM-based Enclave | S | MTT + G-stage | ● | ● | Partial | ~? | ◐ | ○ | ○ |
| Dorami[36] | Firmware-focused | U | PMP | ○ | ○ | None | n/a | ○ | ● | ○ |
| Elasticlave[37] | Temporal Enclave | U | PMP + Shared Pages | ○ | ○ | None | ~8.5k LoC | ○ | ● | ○ |
| Hector-V[29] | Core-partitioned | Core | Interconnect Filtering | ● | ● | None | HW SM | ○ | ○ | ◐ |
| MI6[28] | Unknown | ? | Unknown | ○ | ○ | None | Unknown | ○ | ◐ | ◐ |
| SPEAR-V[34] | S-mode Enclave | S | PMP | ○ | ○ | None | Unknown | ○ | ● | ○ |
| WorldGuard[33] | Domain-based | Core | Bus Filtering | ● | ● | None | SoC-based | ○ | ○ | ○ |
| Penglai[39] | User Enclave | U | MMU-based | ○ | ◐ | None | ~10.2k LoC | ○ | ○ | ○ |
| AnyTEE[40] | sdTZ, sdSGX | U+S | Page Tables + VMM | ◐ | ◐ | None | ~9.4k LoC | ○ | ● | ○ |

**System**: TEE implementation name. **Enclave type**: Isolation type. **Priv. Levels**: Privilege levels used (U: User, S: Supervisor, M: Machine, Core). **Memory Isolation**: Mechanism for memory separation. **Sec. I/O**: Support for secure I/O access. **Cache**: Side-channel protection (e.g., flushing and partitioning). **SDK**: Availability of the development toolchain. **TCB size**: Trusted code base size. **Compliance**: Standards conformance (e.g GlobalPlatform, PKCS#11). **No HW Mod.**: Does the architecture avoid hardware modification (●= no HW mod, ◐= minor changes, ○= requires HW changes). **Crypto.**: Support for secure/accelerated cryptography. Symbol key: ●= full support, ◐= partial, ○= none.

## 4. Discussion

The survey of RISC-V Trusted Execution Environments (TEEs) and secure enclaves reveals a vibrant but fragmented ecosystem. Each proposal addresses particular limitations of proprietary TEEs—such as closed design, rigid threat models, and vendor lock-in—yet none offers a comprehensive or universally deployable solution. Instead, common design trade-offs and systemic challenges emerge across the body of work.

 doi:10.20944/preprints202510.0152.v1



### 4.1. Design Trade-offs

Lightweight approaches such as DORAMI or SPEAR-V emphasize minimal hardware changes and low performance overhead, but provide only partial defenses against side-channel or physical attacks. In contrast, frameworks like HECTOR-V or CURE integrate extensive hardware modifications to strengthen isolation and enable peripheral binding, at the cost of higher complexity and reduced portability.

Different proposals also illustrate distinct models of privilege separation. Sanctum resembles Intel SGX, supporting user-level enclaves but relying on the OS for privileged functions. TIMBER-V instead provides a trusted supervisor mode, allowing privileged services inside enclaves. Keystone combines these approaches by introducing a minimal enclave runtime in S-mode, while still delegating some system services to the untrusted OS. CURE extends this model further by supporting multi-level enclaves, including deep enclaves in M-mode that isolate critical machine-level code. Penglai departs more radically, adopting a microkernel-like approach where service-oriented enclaves provide system functionalities.

Beyond privilege models, our comparative table highlights trade-offs between TCB size, SDK availability, and compliance. Sanctum maintains a minimal ∼5k LoC TCB, while Penglai and Elasticlave reach over 10k LoC, raising the verification burden. Keystone and AnyTEE expose partial SDKs, but no system yet integrates GlobalPlatform APIs or PKCS#11, limiting compatibility with existing trusted applications. Similarly, compliance remains absent across all surveyed works, underscoring the distance from industrial certification.

### 4.2. Scalability and Serverless Use Cases

A recurring limitation of enclave systems is scalability. Keystone and CURE are constrained by the number of PMP entries, limiting the number of concurrent enclaves. Sanctum, likewise, supports only a modest number of enclaves due to its reliance on page table modifications. Penglai addresses this gap by introducing Guarded Page Tables (GPT) and Mountable Merkle Trees (MMT), enabling dynamic management of secure memory at scale. It supports thousands of concurrent enclaves and very large memory footprints (up to 512 GB), making it particularly suited to serverless computing workloads. This stands in contrast to earlier systems like SGX, where enclave memory is capped (e.g., 256 MB in the EPC) and scalability for containerized or microservice environments is limited.

Elasticlave attempts to improve scalability in a different dimension by supporting temporal enclaves and memory sharing between enclaves, but its reliance on PMP limits the number of concurrently protected memory regions. TIMBER-V introduces fine-grained memory tagging, but incurs performance overhead (∼25%) that constrains scalability in resource-constrained embedded settings. Taken together, only Penglai approaches the requirements of cloud-native workloads, while most others remain suitable for embedded or single-application use cases.

### 4.3. Secure I/O

Mainstream TEEs such as SGX lack support for secure I/O, forcing enclaves to delegate device access to the untrusted host OS. This delegation introduces significant attack vectors. Several RISC-V proposals attempt to address this gap. CURE introduces enclave-to-peripheral binding through its bus arbiter and filter engine, ensuring that devices are assigned exclusively to enclaves. HECTOR-V further enhances secure I/O by validating transactions based on core and device identifiers enforced by the Security Monitor. While these mechanisms strengthen protection, they do not yet address more complex scenarios such as dynamic binding and unbinding of stateful peripherals, which remains an open research problem.

WorldGuard extends the notion of domains to encompass both agents and resources, allowing bus-level filtering of device access. However, its focus on static world assignment limits flexibility. Dorami, Keystone, and Sanctum, by contrast, provide no secure I/O binding, leaving enclaves vulnerable to

 doi:10.20944/preprints202510.0152.v1



DMA and Iago-style attacks from malicious peripherals. As IoT and edge systems rely heavily on secure sensor integration, this remains one of the most pressing gaps in the RISC-V enclave landscape.

### *4.4. Defenses Against Side-channel Attacks*

Side-channel attacks remain one of the most persistent threats against enclaves. Sanctum pioneered defenses such as cache partitioning and flush-on-context-switch, while CURE integrates cache tagging and L2 way partitioning. TIMBER-V introduces memory tagging, but still lacks systematic defenses against cache-based leakage. Despite these innovations, side-channel resistance remains incomplete, particularly against speculative execution vulnerabilities. Indeed, attacks targeting the BOOM out-of-order RISC-V processor have demonstrated the feasibility of exploiting transient execution in the context of enclaves. Moreover, protections often incur non-trivial performance costs or require intrusive cooperation from the operating system, limiting their deployability.

MI6 pushes defenses further by adding purge instructions for microarchitectural buffers and adopting spatial/temporal partitioning of caches and DRAM bandwidth. However, it remains tied to a single processor (RiscyOO), lacks shared-memory support, and fails to fully cleanse speculative execution state. SPEAR-V focuses narrowly on same-core leakage but ignores cross-core channels, while Keystone relies only on L1 cache flushing, leaving shared LLCs unprotected. In short, no system yet provides a comprehensive defense against the wide spectrum of cache, speculative, and transient execution attacks.

### *4.5. Systemic Challenges*

Several systemic challenges are evident across the surveyed works. First, fragmentation is a critical issue: the openness of RISC-V allows highly customized enclave architectures, but this diversity results in incompatibility across implementations and prevents the emergence of a unified programming model. Second, the risk of Iago attacks persists wherever enclaves rely on the untrusted host OS to service system calls [81]. Keystone has been shown to be susceptible to this class of attacks, and similar risks exist in other proposals that adopt OS-assisted system interfaces. Third, while some designs claim to reduce the Trusted Computing Base (TCB), the assumption that components such as the Security Monitor or enclave runtimes are bug-free is unrealistic. Bugs in these privileged components would undermine the guarantees of isolation and attestation. Finally, most proposals remain at the prototype stage, often evaluated only in simulation or on FPGA platforms, with limited validation on production silicon or under adversarial workloads.

Our comparison also highlights gaps in compliance and certification. None of the surveyed proposals integrates GlobalPlatform APIs or PKCS#11, and no certification efforts exist for RISC-V enclaves. Furthermore, cryptographic support is underdeveloped: most frameworks rely on software-only crypto, with only HECTOR-V partially exploring hardware acceleration. This limits applicability in domains like financial services or 5G, where certified crypto and compliance are mandatory.

### *4.6. Toward a Cohesive Ecosystem*

Taken together, these findings underscore that the RISC-V enclave and TEE landscape is still in a formative stage. The openness and extensibility of the ISA enable diverse architectural explorations, but the lack of convergence hinders broader adoption. Future work must balance three dimensions simultaneously: (*i*) rigorous security guarantees against advanced attacks, (*ii*) efficient performance and scalability for practical use, and (*iii*) usable software ecosystems with strong standards compliance.

Establishing reference implementations, fostering upstream support in projects such as OpenSBI, Linux, and OP-TEE, and ensuring interoperability between enclave models are key milestones. Ultimately, bridging the gap between academic prototypes and industrial deployments will determine whether RISC-V can evolve into a cohesive and trustworthy foundation for confidential computing.

 doi:10.20944/preprints202510.0152.v1



### *4.7. SDK and Developer Support*

One of the clearest gaps across existing proposals is the absence of a unified developer ecosystem. Keystone offers partial SDK support (via Eyrie), but it is tightly coupled to its custom runtime. AnyTEE makes progress by supporting interoperability with unmodified trusted applications, yet this is achieved through software-defined abstractions rather than standard APIs. By contrast, frameworks such as Sanctum, CURE, Penglai, and SPEAR-V provide no SDK support at all, leaving application developers without reusable toolchains. The lack of integration with GlobalPlatform's TEE Client API or PKCS#11 or TPM prevents the porting of existing trusted applications from ARM TrustZone or Intel SGX ecosystems. This absence represents a critical barrier for adoption, as SDKs are essential to lower the barrier to entry and foster a community of developers around RISC-V TEEs.

### *4.8. Compliance and Standardization*

Compliance with industry specifications remains unaddressed. None of the surveyed proposals demonstrates conformance with GlobalPlatform APIs, or other standards widely required in domains such as automotive, financial, or mobile security, despite some early efforts that remain largely theoretical or in an initial phase [79,80]. In practice, this means that RISC-V TEEs cannot yet serve as drop-in alternatives to ARM or Intel counterparts in regulated environments. CoVE and AP-TEE move closer to aligning with confidential computing standards by adopting virtualization-based abstractions, but they still lack standardized interfaces for trusted applications. Without convergence on compliance, the ecosystem risks perpetuating fragmentation and limiting cross-platform portability.

### *4.9. Trusted Computing Base (TCB) Size*

The TCB size varies dramatically across proposals. Sanctum and Keystone maintain relatively small monitors (5–8k LoC), which facilitates formal verification but limits feature richness. Penglai and Elasticlave expand the TCB to over 10k LoC to support scalability and memory sharing, increasing the verification burden. CURE and HECTOR-V integrate hardware components into their TCB, while systems like MI6 and Cerberus provide only partial or unquantified measurements. Although a smaller TCB is desirable for verification, too minimal a TCB may offload responsibilities to untrusted OS components, reintroducing Iago attack surfaces. Balancing minimalism with functionality remains an unresolved challenge.

### *4.10. Hardware Modifications and Deployability*

The surveyed designs diverge sharply in their reliance on hardware modifications. Frameworks such as Keystone, Elasticlave, and Dorami aim for deployability by relying solely on standard PMP/ePMP. Sanctum, TIMBER-V, CURE, and CoVE introduce new hardware primitives such as tagged memory, bus arbiters, and Memory Tracking Tables, improving isolation but reducing portability. HECTOR-V and WorldGuard push even further, requiring core-partitioning or interconnect filtering logic, making them impractical for general-purpose adoption. This tension between "minimal modification for deployability" and "heavy modification for security" highlights the absence of a common hardware baseline for RISC-V enclaves.

### *4.11. Cryptographic Support*

Cryptography acceleration is scarcely addressed across RISC-V TEEs. HECTOR-V partially considers hardware crypto during secure boot but lacks dedicated acceleration units. Most frameworks, including Keystone, Sanctum, and Penglai, rely entirely on software crypto, raising concerns for high-throughput workloads like TLS or blockchain. Only CoVE explicitly introduces replay-protected memory encryption, approaching the guarantees of AMD SEV-SNP or Intel TDX. The gap in hardware cryptographic integration suggests an opportunity for leveraging RISC-V's vector and scalar crypto extensions to build TEEs that are both secure and efficient.

 doi:10.20944/preprints202510.0152.v1



### *4.12. Grouping by Enclave Type and Memory Isolation*

The table also reveals important differences in enclave types and isolation models. User-level enclaves (Sanctum, Penglai) simplify programmability but depend on the OS for privileged operations, creating Iago vulnerabilities. Multi-level designs (CURE, Keystone, TIMBER-V) introduce supervisor or machine-level enclaves, expanding functionality but enlarging the TCB. Core-partitioned approaches (HECTOR-V, WorldGuard) eliminate sharing but suffer from rigid resource duplication. Memory isolation is equally diverse: PMP and ePMP dominate lightweight designs (Keystone, Dorami, Elasticlave, SPEAR-V), while Penglai relies on MMU-based page isolation, CURE employs cache tagging, TIMBER-V introduces memory tagging, and CoVE adopts a virtualization-based Memory Tracking Table (MTT). These variations reflect a lack of consensus on the "right" abstraction for isolation, with each design optimizing for a different axis: programmability, performance, or security.

### *4.13. Cross-Cutting Observations*

Synthesizing across these dimensions, several patterns emerge. Proposals that minimize hardware changes (Keystone, Dorami, Elasticlave) are easier to deploy but weaker against advanced attacks. Designs that introduce strong defenses (CURE, HECTOR-V, CoVE) achieve richer isolation at the cost of portability. Scalability remains the domain of Penglai, but its complexity and large TCB raise practical barriers. Across the board, SDK maturity, compliance, and cryptographic integration remain glaring omissions, underscoring that RISC-V TEEs are still at a research-prototype stage rather than production-ready solutions.

### *4.14. Future Directions for RISC-V Trusted Execution Environments*

Research and development of RISC-V TEEs remains highly active, with both academic initiatives and contributions from the RISC-V community and hardware vendors seeking to reduce fragmentation and move toward standardized solutions. Several directions are likely to guide future progress. One critical focus will be the development of hardware-assisted defenses against side-channel attacks: since most leakage arises from shared microarchitectural components, software countermeasures alone are insufficient, and RISC-V's extensibility provides an opportunity to integrate primitives such as cache partitioning, memory tagging, and speculation control directly into hardware. Another promising direction is the use of RISC-V cores as coprocessors within heterogeneous SoCs, where they can be dedicated to trusted execution tasks; early industrial efforts already demonstrate the feasibility of this approach. Finally, the growing deployment of RISC-V in embedded and edge computing highlights the need for lightweight TEEs tailored to resource-constrained devices, where traditional designs are impractical. Projects such as TIMBER-V illustrate this trajectory, pointing toward enclave models that emphasize scalability, low overhead, and adaptability to pervasive IoT contexts. Collectively, these trends suggest that the next generation of RISC-V TEEs will need to balance resilience against advanced attacks with efficiency and deployability across both cloud-scale and edge-scale environments.

## 5. Conclusions

This survey has provided the first comprehensive examination of RISC-V TEEs and secure enclave architectures, systematically mapping their design principles, isolation mechanisms, and security guarantees. By consolidating fragmented research into a structured taxonomy, we have highlighted recurring trade-offs, identified common limitations, and emphasized the unique opportunities offered by the openness and extensibility of RISC-V. The analysis reveals that while significant progress has been made toward building trustworthy confidential computing systems on RISC-V, many challenges remain unresolved.

Beyond its role as a state-of-the-art survey, this work is intended to serve as a reference point for future research. Researchers and practitioners can draw on the systematic comparison and taxonomy presented here to design more secure, efficient, and verifiable TEE systems for RISC-V. By highlighting the strengths and shortcomings of existing proposals, the survey guides immediate improvements

 doi:10.20944/preprints202510.0152.v1



while laying the foundation for long-term efforts toward a cohesive and trustworthy open-hardware confidential computing ecosystem.

**Author Contributions:** Conceptualization, M.B.; investigation, M.B.; analysis, M.B.; benchmarking, M.B.; drafting the manuscript, M.B.; writing—review and editing, M.B.; supervision, B.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding. The authors gratefully acknowledge support from NXP, which provided the required hardware resources (e.g., FPGAs and RISC-V boards) used in this study. The APC was funded by the authors.

**Conflicts of Interest:** The authors declare no conflicts of interest. The funders had no role in the selection, analysis, or interpretation of the literature; in the preparation of the manuscript; or in the decision to publish the results.

## References

1. Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted Execution Environment: What It Is, and What It Is Not. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Helsinki, Finland, 20–22 August 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 57–64. https://doi.org/10.1109/Trustcom.2015.357.
2. Costan, V.; Devadas, S. Intel SGX Explained. In *Proceedings of the USENIX Security Symposium*, Austin, TX, USA, 10–12 August 2016; USENIX Association: Berkeley, CA, USA, 2016; pp. 175–190.
3. Kaplan, D.; Powell, J.; Woller, T. AMD Memory Encryption. AMD White Paper, 2016. Available online: https://www.amd.com/system/files/TechDocs/55766_SEV-KM_API_Spec.pdf (accessed on 30 September 2025).
4. ARM Ltd. ARM Security Technology—Building a Secure System Using TrustZone Technology. ARM Technical Report, 2009. Available online: https://developer.arm.com/documentation/PRD29-GENC-009492C?lang=en (accessed on 30 September 2025).
5. Oliner, A.; Weisse, O.; Austin, T. Analyzing the Limits of Intel SGX for Secure Cloud Computation. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Madison, WI, USA, 24–26 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 233–244. https://doi.org/10.1109/ISPASS.2019.00032.
6. Hetzelt, F.; Buhren, R. SEVered: Subverting AMD's Virtual Machine Encryption. In *Proceedings of the 11th European Workshop on Systems Security (EuroSec)*, Porto, Portugal, 23 April 2018; ACM: New York, NY, USA, 2018; pp. 1–6. https://doi.org/10.1145/3228933.3228934.
7. Raj, H.; Natu, M.; Sarma, S.; Chandra, V. TEE Support in Arm Processors: A Survey. *IEEE Design & Test* **2020**, *37*, 63–72. https://doi.org/10.1109/MDAT.2020.2987349.
8. Azab, A.; Ning, P.; Shah, J.; Bhutkar, R.; Ganju, K.; Shen, W.; Wang, Y. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Scottsdale, AZ, USA, 3–7 November 2014; ACM: New York, NY, USA, 2014; pp. 90–102. https://doi.org/10.1145/2660267.2660350.
9. Van Bulck, J.; Minkin, M.; Weisse, O.; Genkin, D.; Kasikci, B.; Piessens, F.; Silberstein, M.; Wenisch, T.F.; Yarom, Y.; Strackx, R. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *Proceedings of the 27th USENIX Security Symposium*, Baltimore, MD, USA, 15–17 August 2018; USENIX Association: Berkeley, CA, USA, 2018; pp. 991–1008.
10. Van Bulck, J.; Piessens, F.; Strackx, R. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 18–21 May 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 54–72. https://doi.org/10.1109/SP40000.2020.00017.
11. Chen, T.; Zang, B.; Chen, H.; Guan, H. TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices. *IACR Cryptology ePrint Archive* **2017**, 2017:1169. Available online: https://eprint.iacr.org/2017/1169 (accessed on 30 September 2025).
12. Xu, L.; Yang, J.; Li, Z.; Zhao, Y.; Zhang, T.; Wang, T. TruSense: Information Leakage from ARM TrustZone via Cache Side Channels. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, Abu Dhabi, UAE, 17–20 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 471–474. https://doi.org/10.1109/ICCD46524.2019.00084.

 doi:10.20944/preprints202510.0152.v1




13. Tang, A.; Sethumadhavan, S.; Stolfo, S. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *Proceedings of the 26th USENIX Security Symposium*, Vancouver, BC, Canada, 16–18 August 2017; USENIX Association: Berkeley, CA, USA, 2017; pp. 1057–1074.
14. Chen, G.; Chen, S.; Xiao, Y.; Zhang, Y.; Lin, Z.; Lai, T.H.; Xing, X. SoK: Understanding Security Vulnerabilities of Software-based Trusted Execution Environments. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 20–22 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1003–1020. https://doi.org/10.1109/SP.2019.00066.
15. Asanović, K.; Patterson, D. Instruction Sets Should Be Free: The Case for RISC-V. *EECS Department, University of California, Berkeley, Technical Report UCB/EECS-2014-146*, 2014. Available online: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html (accessed on 30 September 2025).
16. Anders, J.; Andreu, P.; Becker, B.; Becker, S.; Cantoro, R.; Deligiannis, N.I.; Elhamawy, N.; Faller, T.; Hernandez, C.; Mentens, N.; Rizi, M.N. A Survey of Recent Developments in Testability, Safety and Security of RISC-V Processors. In *Proceedings of the 2023 IEEE European Test Symposium (ETS)*, Venice, Italy, 22–26 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–10. https://doi.org/10.1109/ETS57893.2023.10157052.
17. Feng, X.; Su, Y.; Liu, J.; Xu, C.; Liu, Y.; Xu, H.; Chen, K. A Survey of Confidential Computing. *IET Communications* **2023**, *17*, 1234–1249. https://doi.org/10.1049/cmu2.12759.
18. Maene, P., Götzfried, J., de Clercq, R., Müller, T., Freiling, F., & Verbauwhede, I. (2018). Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Transactions on Computers*, 67(3), 361-374.
19. Muñoz, A., Ríos, R., Román, R., & López, J. (2023). A survey on the (in)security of trusted execution environments. *Computers & Security*, 103180.
20. Lu, T. A Survey on RISC-V Security: Hardware and Architecture. *arXiv Preprint* **2021**, arXiv:2107.04175. https://doi.org/10.48550/arXiv.2107.04175.
21. M. Schneider, R. J. Masti, S. Shinde, S. Čapkun, and R. Perez, "SoK: Hardware-supported Trusted Execution Environments," *arXiv:2205.12742*, 2022. doi:10.48550/arXiv.2205.12742.
22. M. Li, Y. Yang, G. Chen, M. Yan, and Y. Zhang, "SoK: Understanding Design Choices and Pitfalls of Trusted Execution Environments," in *Proc. 19th ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2024. doi:10.1145/3634737.3644993.
23. T. Geppert, J. Anderegg, L. Frei, S. Moeller, S. Deml, D. Sturzenegger, *et al.*, "Trusted Execution Environments: Applications and Organizational Challenges," *Frontiers in Computer Science*, vol. 4, article 930741, 2022. doi:10.3389/fcomp.2022.930741.
24. GlobalPlatform. TEE v1.3; Technical Specification, 2023. https://globalplatform.org/wp-content/uploads/2022/05/GPD_SPE_009-GPD_TEE_SystemArchitecture_v1.3_PublicRelease_signed.pdf
25. Lee, D.; Kohlbrenner, D.; Shinde, S.; Asanović, K.; Song, D. Keystone: An Open Framework for Architecting Trusted Execution Environments. *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*. ACM, 2020; pp. 1–16.
26. Costan, V.; Lebedev, I.; Devadas, S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. *25th USENIX Security Symposium*. USENIX Association, 2016; pp. 857–874.
27. Feng, S.; Aublin, P.-L.; Ta-Min, R.; Felber, P.; Le Métayer, D. TIMBER-V: Tag-Isolated Memory Bringing Enclaves to RISC-V. *2021 IEEE 27th International Symposium on High Performance Computer Architecture (HPCA)*; IEEE, 2021; pp. 432–445.
28. Mashtizadeh, A.; Wentzlaff, D. MI6: Secure Enclaves in a Speculative Out-of-Order Processor. *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*; IEEE, 2019; pp. 42–55.
29. Vilanova, L.; Weisse, O.; Bartolini, A.; Bartolini, M.; Sampaio, L. HECTOR-V: A Heterogeneous Architecture for Trusted Execution in RISC-V. *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*; IEEE, 2020; pp. 1464–1469.
30. Bahmani, R.; Knauth, T.; Sammler, M.; Weiser, S.; Fetzer, C. CURE: A Security Architecture with CUstomizable Enclaves. *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; IEEE, 2020; pp. 1–14.
31. Stapf, E.; Weiser, S.; Bahmani, R.; Knauth, T.; Fetzer, C. Cerberus: A Memory Isolation Framework for Enclaves. *2021 IEEE International Symposium on Secure and Private Execution Environments (SEED)*; IEEE, 2021; pp. 1–13.
32. Sahita, R.; Ge, Q.; Liang, J.; Love, E.; Costan, V.; Li, S. CoVE: Confidential Computing Architecture for RISC-V. *Proceedings of the 2023 IEEE Symposium on Security and Privacy (S&P)*; IEEE, 2023; pp. 1–17.
33. Nasahl, J.; Hoang, T.; Pinto, S.; Köpf, B. WorldGuard: Enforcing Strong Isolation for Trusted Execution on RISC-V. *2024 IEEE European Symposium on Security and Privacy (EuroS&P)*; IEEE, 2024; pp. 112–127.

 doi:10.20944/preprints202510.0152.v1




34. Shah, H.; Nguyen, K.; El Haji, M.; Dashti, M.; Knauth, T.; Bahmani, R. SPEAR-V: Software-isolated Enclaves on RISC-V. *2022 IEEE International Conference on Computer Design (ICCD)*; IEEE, 2022; pp. 289–296.
35. Zhang, Y.; Gu, R.; Wang, H.; Yang, Y.; Li, J.; Wang, X. VirTEE: A Secure Virtualization-based TEE for RISC-V. *2022 IEEE International Conference on Parallel and Distributed Systems (ICPADS)*; IEEE, 2022; pp. 564–572.
36. Koh, H.; Choi, M.; Lee, J.; Park, J. DORAMI: Lightweight Trusted Execution with Enhanced PMP on RISC-V. *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*; IEEE, 2024; pp. 745–758.
37. Shinde, S.; Knauth, T.; Weisse, O.; Asanović, K.; Song, D. Elasticlave: An Efficient Memory Model for Enclaves. *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*; USENIX Association, 2020; pp. 523–539.
38. Sahita, R.; Ge, Q.; Li, S.; Costan, V. AP-TEE: Application Platform Trusted Execution Environment Specification for RISC-V. *RISC-V Summit*, 2023. Available online: https://github.com/riscv/tee (accessed on 27 September 2025).
39. Shen, R.; Wu, J.; Chen, Z.; Zuo, C.; Guo, Y.; Chen, H. Penglai: Scaling Enclave Applications with Dynamic Memory Management. *16th European Conference on Computer Systems (EuroSys)*; ACM, 2021; pp. 275–290.
40. Dessouky, G.; Cheang, K.; Bhatotia, P.; Weiser, S.; Bahmani, R. AnyTEE: A Portable and Open Framework for Enclaves. *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*; ACM, 2021; pp. 297–310.
41. Intel Corporation. Intel Software Guard Extensions (Intel SGX) Programming Reference. *Intel Developer Manual*, 2020. Available online: https://www.intel.com/sgx (accessed on 27 September 2025).
42. Kaplan, D.; Powell, J.; Woller, T. AMD Memory Encryption Technologies. *AMD White Paper*, 2016.
43. Alves, T.; Felton, D. TrustZone: Integrated Hardware and Software Security. *ARM White Paper*, 2004.
44. Ge, Q.; Yarom, Y.; Cock, D.; Heiser, G. A Survey of Microarchitectural Timing Attacks and Countermeasures on Modern Processors. *ACM Computing Surveys* **2019**, *54* (4), 1–36.
45. Van Bulck, J.; Minkin, M.; Weisse, O.; Genkin, D.; Kasikci, B.; Piessens, F.; Silberstein, M.; Wenisch, T.; Yarom, Y.; Strackx, R. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. *27th USENIX Security Symposium*, 2018; pp. 991–1008.
46. Van Bulck, J.; Minkin, M.; Genkin, D.; Kasikci, B.; Piessens, F.; Silberstein, M.; Strackx, R. LVI: Hijacking Transient Execution through Load Value Injection. *2020 IEEE Symposium on Security and Privacy (S&P)*; IEEE, 2020; pp. 54–72.
47. Hetzelt, F.; Buhren, R. SEVered: Subverting AMD's Virtual Machine Encryption. *Proceedings of the 11th European Workshop on Systems Security (EuroSec)*, 2018; pp. 1–6.
48. Anders, J.; Andreu, P.; Becker, B.; Becker, S.; Cantoro, R.; Deligiannis, N.I.; Elhamawy, N.; Faller, T.; Hernandez, C.; Mentens, N.; Rizi, M.N. A Survey of Recent Developments in Testability, Safety and Security of RISC-V Processors. *2023 IEEE European Test Symposium (ETS)*, IEEE, May 2023; pp. 1–10. https://doi.org/10.1109/ETS57893.2023.10157052.
49. Wong, M.M.; Haj-Yahya, J.H.; Chattopadhyay, A. SMARTS: Secure Memory Assurance of RISC-V Trusted SoC. *Proc. 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, ACM, June 2018; pp. 1–8. https://doi.org/10.1145/3214292.3214300.
50. Krentz, K.F.; Voigt, T. Reducing trust assumptions with OSCORE, RISC-V, and Layer 2 one-time passwords. *Int. Symp. Found. Pract. Secur.*, Springer Nature Switzerland, Dec 2022; pp. 389–405.
51. Cheang, K.; Rasmussen, C.; Lee, D.; Kohlbrenner, D.W.; Asanović, K.; Seshia, S.A. Verifying RISC-V Physical Memory Protection. *arXiv* **2022**, arXiv:2211.02179. Available online: https://arxiv.org/abs/2211.02179 (accessed on 30 Sept 2025).
52. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; Schwarz, M.; Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *Commun. ACM* **2020**, 63 (7), 93–101. https://doi.org/10.1145/3399742.
53. Li, T.; Hopkins, B.; Parameswaran, S. SIMF: Single-Instruction Multiple-Flush Mechanism for Processor Temporal Isolation. *arXiv* **2020**, arXiv:2011.10249. Available online: https://arxiv.org/abs/2011.10249 (accessed on 30 Sept 2025).
54. Stapf, E.; Jauernig, P.; Brasser, F.; Sadeghi, A.-R. In Hardware We Trust? From TPM to Enclave Computing on RISC-V. *2021 IFIP/IEEE 29th Int. Conf. Very Large Scale Integration (VLSI-SoC)*, IEEE, Oct 2021; pp. 1–6. https://doi.org/10.1109/VLSI-SoC53125.2021.9606957.

 doi:10.20944/preprints202510.0152.v1




55. Schneider, M.; Dhar, A.C.; Puddu, I.; Kostiainen, K.; Capkun, S. Composite Enclaves: Towards Disaggregated Trusted Execution. *arXiv* **2020**, arXiv:2010.10416. Available online: https://arxiv.org/abs/2010.10416 (accessed on 30 Sept 2025).
56. Kieu-Do-Nguyen, B.; Nguyen, K.D.; Dang, T.K.; Binh, N.T.; Pham-Quoc, C.; Tran, N.T.; Pham, C.K.; Hoang, T.T. A Trusted Execution Environment RISC-V System-on-Chip Compatible with Transport Layer Security 1.3. *Electronics* **2024**, 13 (13), 2508. https://doi.org/10.3390/electronics13132508.
57. Chen, Y.; Chen, H.; Chen, S.; Han, C.; Ye, W.; Liu, Y.; Zhou, H. DITES: A lightweight and flexible dual-core isolated trusted execution SoC based on RISC-V. *Sensors* **2022**, 22 (16), 5981. https://doi.org/10.3390/s22165981.
58. Hoang, T.T.; Duran, C.; Serrano, R.; Sarmiento, M.; Nguyen, K.D.; Tsukamoto, A.; Suzaki, K.; Pham, C.K. Trusted Execution Environment Hardware by Isolated Heterogeneous Architecture for Key Scheduling. *IEEE Access* **2022**, 10, 46014–46027. https://doi.org/10.1109/ACCESS.2022.3169969.
59. Pinto, S.; Martins, J.; Rodriguez, M.; Cunha, L.; Schmalz, G.; Moslehner, U.; Dieffenbach, K.; Roecker, T. RISC-V Needs Secure 'Wheels': The MCU Initiator-Side Perspective. *arXiv* **2024**, arXiv:2410.09839. Available online: https://arxiv.org/abs/2410.09839 (accessed on 30 Sept 2025).
60. Pan, S.; Peng, X.; Man, Z.; Zhao, X.; Zhang, D.; Yang, B.; Du, D.; Lu, H.; Xia, Y.; Li, X. DepTEE: Decoupled Memory Protection for Secure and Scalable Inter-enclave Communication on RISC-V. *Proc. 30th Asia and South Pacific Design Automation Conference (ASPDAC '25)*, ACM, 2025; pp. 454–460. https://doi.org/10.1145/3658617.3697763.
61. Anders, J.; Andreu, P.; Becker, B.; Becker, S.; Cantoro, R.; Deligiannis, N.I.; Elhamawy, N.; Faller, T.; Hernandez, C.; Mentens, N.; Rizi, M.N. A Survey of Recent Developments in Testability, Safety and Security of RISC-V Processors. *2023 IEEE European Test Symposium (ETS)*, IEEE, May 2023; pp. 1–10. https://doi.org/10.1109/ETS57893.2023.10157052.
62. Wong, M.M.; Haj-Yahya, J.H.; Chattopadhyay, A. SMARTS: Secure Memory Assurance of RISC-V Trusted SoC. *Proc. 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, ACM, June 2018; pp. 1–8. https://doi.org/10.1145/3214292.3214300.
63. Krentz, K.F.; Voigt, T. Reducing trust assumptions with OSCORE, RISC-V, and Layer 2 one-time passwords. *Int. Symp. Found. Pract. Secur.*, Springer Nature Switzerland, Dec 2022; pp. 389–405.
64. Cheang, K.; Rasmussen, C.; Lee, D.; Kohlbrenner, D.W.; Asanović, K.; Seshia, S.A. Verifying RISC-V Physical Memory Protection. *arXiv* **2022**, arXiv:2211.02179. Available online: https://arxiv.org/abs/2211.02179 (accessed on 30 Sept 2025).
65. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; Schwarz, M.; Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *Commun. ACM* **2020**, 63 (7), 93–101. https://doi.org/10.1145/3399742.
66. Li, T.; Hopkins, B.; Parameswaran, S. SIMF: Single-Instruction Multiple-Flush Mechanism for Processor Temporal Isolation. *arXiv* **2020**, arXiv:2011.10249. Available online: https://arxiv.org/abs/2011.10249 (accessed on 30 Sept 2025).
67. Stapf, E.; Jauernig, P.; Brasser, F.; Sadeghi, A.-R. In Hardware We Trust? From TPM to Enclave Computing on RISC-V. *2021 IFIP/IEEE 29th Int. Conf. Very Large Scale Integration (VLSI-SoC)*, IEEE, Oct 2021; pp. 1–6. https://doi.org/10.1109/VLSI-SoC53125.2021.9606957.
68. Schneider, M.; Dhar, A.C.; Puddu, I.; Kostiainen, K.; Capkun, S. Composite Enclaves: Towards Disaggregated Trusted Execution. *arXiv* **2020**, arXiv:2010.10416. Available online: https://arxiv.org/abs/2010.10416 (accessed on 30 Sept 2025).
69. Kieu-Do-Nguyen, B.; Nguyen, K.D.; Dang, T.K.; Binh, N.T.; Pham-Quoc, C.; Tran, N.T.; Pham, C.K.; Hoang, T.T. A Trusted Execution Environment RISC-V System-on-Chip Compatible with Transport Layer Security 1.3. *Electronics* **2024**, 13 (13), 2508. https://doi.org/10.3390/electronics13132508.
70. Chen, Y.; Chen, H.; Chen, S.; Han, C.; Ye, W.; Liu, Y.; Zhou, H. DITES: A lightweight and flexible dual-core isolated trusted execution SoC based on RISC-V. *Sensors* **2022**, 22 (16), 5981. https://doi.org/10.3390/s22165981.
71. Hoang, T.T.; Duran, C.; Serrano, R.; Sarmiento, M.; Nguyen, K.D.; Tsukamoto, A.; Suzaki, K.; Pham, C.K. Trusted Execution Environment Hardware by Isolated Heterogeneous Architecture for Key Scheduling. *IEEE Access* **2022**, 10, 46014–46027. https://doi.org/10.1109/ACCESS.2022.3169969.
72. Pinto, S.; Martins, J.; Rodriguez, M.; Cunha, L.; Schmalz, G.; Moslehner, U.; Dieffenbach, K.; Roecker, T. RISC-V Needs Secure 'Wheels': The MCU Initiator-Side Perspective. *arXiv* **2024**, arXiv:2410.09839. Available online: https://arxiv.org/abs/2410.09839 (accessed on 30 Sept 2025).
73. Pan, S.; Peng, X.; Man, Z.; Zhao, X.; Zhang, D.; Yang, B.; Du, D.; Lu, H.; Xia, Y.; Li, X. DepTEE: Decoupled Memory Protection for Secure and Scalable Inter-enclave Communication on RISC-V.

 doi:10.20944/preprints202510.0152.v1



*Proc. 30th Asia and South Pacific Design Automation Conference (ASPDAC '25)*, ACM, 2025; pp. 454–460. https://doi.org/10.1145/3658617.3697763.

74. E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable Memory Protection in the PENGLAI Enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, USENIX Association, Jul. 2021, pp. 275–294. ISBN: 978-1-939133-22-9.
75. G. Dessouky, A.-R. Sadeghi, and E. Stapf, "Enclave Computing on RISC-V: A Brighter Future for Security?," in *Workshop on Secure RISC-V Architecture Design (SECRISC-V)*, Apr. 2020, vol. 20.
76. A.-T. Le, "Research of RISC-V Out-of-order Processor Cache-Based Side-Channel Attacks: Systematic Analysis, Security Models and Countermeasures," Ph.D. dissertation, University of Electro-Communications, Tokyo, Japan, 2023.
77. V. Donnini, "Integration of the DICE Specification into the Keystone Framework," Ph.D. dissertation, Politecnico di Torino, 2023.
78. J. Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, "Elasticlave: An Efficient Memory Model for Enclaves," in *31st USENIX Security Symposium (USENIX Security 22)*, USENIX Association, 2022, pp. 4111–4128.
79. M. Boubakri, F. Chiatante, and B. Zouari, "Open Portable Trusted Execution Environment Framework for RISC-V," in *Proc. 2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC)*, Shenyang, China, Oct. 2021, pp. 1–8. doi:10.1109/EUC53437.2021.00015.
80. K. Suzaki, K. Nakajima, T. Oi, and A. Tsukamoto, "Library Implementation and Performance Analysis of GlobalPlatform TEE Internal API for Intel SGX and RISC-V Keystone," in *Proc. 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Dec. 2020, pp. 200–207. doi:10.1109/TrustCom50675.2020.00037.
81. F. Zhang, L. Zhou, Y. Zhang, M. Ren, and Y. Deng, "Trusted Execution Environment: State-of-the-Art and Future Directions," *Journal of Computer Research and Development*, vol. 61, no. 1, pp. 243–260, 2024. doi:10.7544/issn1000-1239.202221016.

## Short Biography of Authors

**Marouene Boubakri** received the double Engineering degree from the Higher School of Communication of Tunis (SUP'COM), Tunisia, and EURECOM, France. He is currently pursuing the Ph.D. degree in Electronics and Information and Communication Technologies at the Tunisia Polytechnic School. His research interests include computer architecture security, trusted execution environments, and confidential computing, with a focus on RISC-V. He actively participates in the RISC-V community to advance secure hardware and software ecosystems.

**Belhassen Zouari** received the Doctorate (French Ph.D.) from the University of Paris6, Paris, France, in 1993, in computer science, and the "Habilitation Universitaire" from the University of Tunis, Tunis, Tunisia, in 2005.,He is a Professor in Computer Science at the Higher School of Communications Sup'Com-member of Mediatron Lab- University of Carthage, Tunisia. His activities include research works in the fields of verification of automated systems and security issues. He worked on formal verification methods based on coloured Petri nets and applied the related techniques in various fields as Flexible Manufacturing Systems, Wireless Sensor Networks and Business Process Management.