COFFER: An Efficient and Scalable TEE on RISC-V

Mingde Ren, Jiatong Chen, Ziquan Wang, *Student Member, IEEE*, Fengwei Zhang, *Senior Member, IEEE*, Zhenyu Ning, and Heming Cui, *Member, IEEE*

***Abstract*—*Trusted Execution Environment* (TEE) is a primary means for confidential computing. However, at the moment the RISC-V platform is limited for confidential computing because current RISC-V TEEs either lack scalability or compatibility. The reason for this dilemma in scalability and compatibility is that the standard isolation primitive on RISC-V, *Physical Memory Protection* (PMP), is not scalable. Meanwhile, previous enclave designs depend on the *Rich Execution Environment* (REE) for OS functionalities, which increases domain switch frequency and enlarges the attack surface of the TEE. In this work, we propose COFFER, a scalable and efficient software-based TEE for the standard RISC-V platform. COFFER includes two core techniques: *Logical PMP* (LPMP) and *Enclave Module*s (EModules) to address the issues mentioned above. LPMP is a secure and efficient framework for PMP virtualization. It provides both scalability and hardware compatibility for COFFER. EModules are dynamically assembled lightweight libraries to provide enclaves with OS functionalities. The EModules provide COFFER with software compatibility and reduce the *Trusted Computing Base* (TCB) size of the enclaves. We implement and evaluate COFFER on commercially available RISC-V devices. The evaluation results show that COFFER can support 2,000+ concurrent enclaves with negligible performance overhead. Particularly, LPMP supports enclave execution under heavy memory fragmentation with little performance overhead.**

***Index Terms*—Trusted execution environment, enclave , RISC-V, confidential computing, memory isolation, scalability.**

I. INTRODUCTION

TRUSTED Execution Environment (TEE), as a primary means of confidential computing, offers a practical solution for user data protection. TEEs provide isolated domains called enclaves which can protect user data under a potentially compromised host. Examples including Intel SGX [1], Intel TDX [2] and AMD SEV [3] have already gained popularity in the industry [4], [5], [6]. Additionally, Arm has recently announced CCA [7], which is expected to appear in the market soon.

RISC-V is another candidate platform for confidential computing. While currently RISC-V devices are primarily deployed as endpoint IoT devices, there are also chip designs targeting at high-performance computation for cloud servers [8]. Additionally, RISC-V could be more favored because it is more cost-effective than x86 or Arm [9]. However, at the moment, the standard RISC-V platform is limited for confidential computing. This is because existing TEE proposals for the standard RISC-V platform are not scalable, failing to meet the requirements for some scenarios such as cloud computing. For example, Keystone [10] is a software-based TEE for standard RISC-V hardware. Nevertheless, it can only support ≤ 16 enclaves, with each enclave allowed to have only ≤ 16 contiguous memory regions. Meanwhile, Composite Enclaves [11], a RISC-V TEE built on top of Keystone targeting at specialized hardware integration, also faces the same scalability limitation.

To overcome the scalability limitation, Penglai [12] introduces permission tables and modifies the MMU for enclave memory isolation. TIMBER-V [13] avoids limitations on the number of enclaves by utilizing memory tagging as the isolation mechanism. However, such RISC-V TEEs provide scalability at the cost of sacrificing hardware compatibility. Their dependency on customized hardware prevents them from being deployable on standard RISC-V platforms. The RISC-V community has also been discussing security extensions such as CoVE [14] and PMP Table [15]. Although these proposals do not have scalability limitations, they are still under development and yet to be merged into the mainline of RISC-V standard specifications [16], [17], [18]. Moreover, these security extensions are only optional features on RISC-V and thus they might not be as generic as PMP, which is available on the majority of RISC-V platforms (see Section II-A). Therefore, depending on these extensions may limit system hardware compatibility as well. This dilemma between scalability and compatibility poses an essential question: **How to design a scalable and efficient TEE with high compatibility for the RISC-V platform?**

Designing such a TEE involves three challenges. The first is to support a large number of enclaves using PMP, the standard memory isolation primitive on RISC-V. PMP includes a set of *Control and Status Register*s (CSRs) to specify memory segments and the corresponding permissions. However, the number of memory segments that can be covered

Received 21 May 2025; revised 13 November 2025; accepted 10 December 2025. Date of publication 25 December 2025; date of current version 29 January 2026. This work was supported in part by the National Natural Science Foundation of China under Grant U2541211, Grant 62372218, and Grant U24A6009. The associate editor coordinating the review of this article and approving it for publication was Dr. Jason Xue. *(Mingde Ren and Jiatong Chen contributed equally to this work.) (Corresponding authors: Zhenyu Ning; Fengwei Zhang.)*

Mingde Ren is with the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology (SUSTech), Shenzhen 518055, China, and also with the Department of Computer Science, The University of Hong Kong, Hong Kong, China (e-mail: mingde-ren@outlook.com).

Jiatong Chen and Ziquan Wang are with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen 518055, China (e-mail: chenjt2023@mail.sustech.edu.cn; wangzq2024@mail.sustech.edu.cn).

Fengwei Zhang is with the Department of Computer Science and Engineering and the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology (SUSTech), Shenzhen 518055, China (e-mail: zhangfw@sustech.edu.cn).

Zhenyu Ning is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: zning@hnu.edu.cn).

Heming Cui is with the Department of Computer Science, The University of Hong Kong, Hong Kong, China (e-mail: heming@cs.hku.hk).

Digital Object Identifier 10.1109/TIFS.2025.3648190

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

by PMP is limited by hardware resources. This issue of PMP also presents the second challenge: how to support highly fragmented enclave memory regions. With memory virtualization, physical memory allocation in modern OSes tends to be fragmented. Enforcing contiguous memory allocation may downgrade system performance and lower resource utilization [19]. On the other hand, if an enclave has highly fragmented physical memory, then PMP cannot protect all memory regions of the enclave, which is unacceptable for a TEE. Additionally, enclave execution efficiency is important in confidential computing. However, current enclave designs usually depend on REE for OS functionalities to reduce TCB complexity [1], [12], [20], [21]. Such dependency leads to frequent and costly domain switches between the enclaves and REE, downgrading system performance. Moreover, the dependency on REE undermines the autonomy of the enclaves and expands the attack surface of TEE, making it potentially vulnerable [22], [23], [24], [25]. Therefore, the third challenge is to remove critical enclave functional dependencies on REE while keeping the enclave TCB size as small as possible.

In this paper, we propose COFFER, a scalable and efficient software-based TEE on RISC-V. COFFER dissolves the first two challenges with LPMP, an efficient and secure framework for RISC-V PMP virtualization. For the first challenge, LPMP adopts a symmetric host/enclave treatment to increase the enclave amount. For the second challenge, LPMP virtualizes PMP entries to support fragmented physical memory. It uses a trap-and-emulate method for enclave memory access. In light of modern cache designs, LPMP uses the *Most Recently Used* (MRU) policy with instruction/data split for PMP management, which significantly reduces trap times and improves enclave performance. Moreover, we observe that on many commercially available RISC-V platforms [26], [27], the PMP checking results are cached in the TLB (see Section II-B). On such platforms, LPMP carefully maintains the TLB state to further improve enclave performance without undermining security. For the third challenge, COFFER equips each enclave with a specialized set of EModules to eliminate dependencies on REE. EModules are modularized lightweight libraries inspired by LibOSes [28], [29], [30], [31], [32], [33], [34], [35], [36], [37]. They enable COFFER enclaves to strike a balance between functionalities and the TCB size. Besides, the EModules can also improve enclave performance since it reduces domain switch frequency.

We have implemented and evaluated COFFER on off-the-shelf commodity RISC-V devices. Our evaluation results show that the LPMP framework can support concurrent execution of 2,000+ enclaves. Furthermore, LPMP enables COFFER enclaves to execute under heavy memory fragmentation with $\leq 3\%$ overhead. We have also evaluated COFFER on real-world applications. The result shows that COFFER enclaves incur $4\% \sim 15\%$ performance overhead for SQLite3, and $\leq 5\%$ for DarkNet.

In addition, we also implement LPMP as a software extension (in ~700 LoCs) that can be integrated into existing RISC-V TEE such as Keystone [10] to break their enclave number limitation. We have submitted our LPMP implementation to the Keystone project.

We make the following contributions in this paper:

- We present COFFER, a scalable and efficient software-based TEE on standard RISC-V platforms.
- We propose LPMP, a secure and efficient framework for PMP virtualization. It manages PMP with the MRU policy and instruction/data split. On certain devices, LPMP provides further performance improvement with TLB caching.
- We provide EModules for COFFER enclaves to remove dependencies on REE and strike a balance between functionalities and the TCB size.
- We implemented and evaluated COFFER on HiFive Unmatched [26] and VisionFive 2 [27]. Our evaluation results show that LPMP can support a large number of enclaves with negligible performance overhead. LPMP also supports enclave execution under heavily fragmented physical memory with little performance degradation. We will open-source and provide continuous maintenance to our implementation of COFFER.

TABLE I
AVAILABILITY OF SECURITY EXTENSIONS ON COMMON RISC-V PLATFORMS

Security Extension	HiFive Unmatched	VisionFive 2	D1 Nezha	NOEL-V	Rocket-Chip	BOOM
PMP	✓	✓	✓	✓	✓	✓
Smepmp	✗	✗	✗	✓	❍	❍
Sspmp	✗	✗	✗	✗	❍	❍
PMP Table	✗	✗	✗	✗	❍	❍
CoVE	✗	✗	✗	✗	-	-

✓: Available, ✗: Unavailable, -: Unknown,
❍: Implemented but not officially supported.

II. MOTIVATIONS

A. Availability of RISC-V Security Extensions

RISC-V is an open and free *Instruction Set Architecture* (ISA) introduced in 2014 [38]. Any individual or organization can propose new extensions for the RISC-V platform. To date, there are 28 extensions ratified by the RISC-V community [39] and 50+ extensions under development [18]. Among them there are 5 security extensions focused on memory isolation. We list the availability of these extensions on common RISC-V platforms in Table I. We find that PMP is the only universally supported security extension among common RISC-V platforms and that **PMP is the only security extension available on off-the-shelf commodity RISC-V devices.**

Moreover, despite the variety of RISC-V security extensions, they are proposed by distinguished groups. Consequently, they may potentially be incompatible with each other. Dependency on non-universally compatible extensions may weaken system practicality. Therefore, to provide high compatibility with RISC-V commodity devices, we must build the TEE based on PMP only.

B. RISC-V PMP Features

Although PMP is universally available, circuit implementation details may differ among different RISC-V platforms. To ensure high system compatibility, we must only utilize the common features on these platforms. Table II summarizes PMP-related hardware details on common RISC-V platforms. Note that although the latest version of RISC-V specification [17] allows for up to 64 PMP entries, **most commercially available RISC-V devices only support 8 PMP entries.**

TABLE II
PMP-RELATED HARDWARE DETAILS ON COMMON RISC-V PLATFORMS

Hardware Detail	HiFive Unmatched	VisionFive 2	D1 Nezha	NOEL-V	Rocket-Chip	BOOM
PMP Number (N_{PMP}^{max})	8	8	8	8/10	16	8
PMP Granularity	4KB-Page (U7 Core) 64-Byte (S7 Core†)	4KB-Page (U7 Core) 64-Byte (S7 Core)	4KB-Page	4KB-Page	4-Byte (Configurable)	4-Byte (Configurable)
TLB-Cached PMP Checking	✓	✓	✗	Unknown	✓/✗††	✓/✗††
TLB Details	L1: Fully-assoc, 40-entry Inst/Data L2: Direct-map, 512-entry Hybrid	L1: Fully-assoc, 40-entry Inst/Data L2: Direct-map, 512-entry Hybrid	L1: Fully-assoc, 10-entry Inst/Data L2: 2-way Set-assoc, 256-entry Hybrid	L1: Fully-assoc, 16-entry Inst/Data L2: Fully-assoc, 16-entry Hybrid	Fully-assoc, 32-entry Inst/Data	L1: Fully-assoc, 32-entry Inst/Data L2: Direct-map, 512-entry Hybrid

†: There is no MMU on S7 core.
††: PMP checking results are cached in the TLB if the PMP granularity is configured to ≥ 4KB-Page.

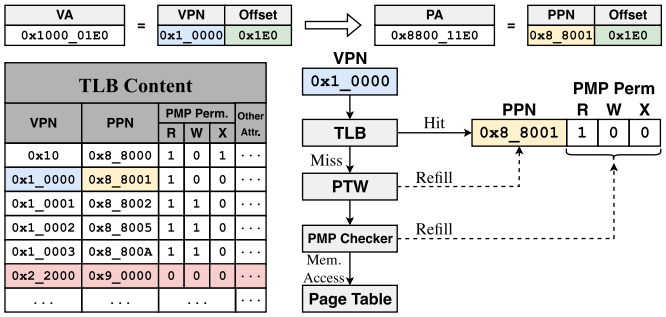


Fig. 1. TLB-cached PMP checking results. *PMP Perm.* means PMP permissions. *Other Attr.* means other attributes. *PTW* means page table walker. TLB contains PMP checking results in addition to address translation results, including entries with prohibitive PMP permissions (shown in red color). PMP checker is only used on TLB misses.

The PMP specification defines three address-matching modes: TOR, NAPOT and NA4. In TOR mode, two PMP entries are used to specify a memory segment. In NAPOT mode, a single PMP entry can cover a memory segment but the segment must be naturally power-of-two aligned. NA4 can be regarded as a special case of NAPOT where the segment size must be 4 bytes, which is the finest granularity standard PMP provides. However, requiring naturally aligned memory regions poses extra restrictions to the system memory allocator. Therefore, in this paper we mainly focus on the TOR mode, which is more generic but harder to scale up. Nevertheless, all methods in this paper can be directly applied to NAPOT/NA4 mode as well. Moreover, as shown in Table II, on commercially available RISC-V processors with address translation support, the PMP granularity is only 4KB-Page. This is not surprising because adopting page-size granularity avoids circuit complexity.

Additionally, we find that on many RISC-V platforms, PMP checking is only triggered on a TLB miss when the MMU is enabled. For addresses that are already present in the TLB, PMP checking is skipped and previous checking results are used (see Figure 1). The official specification (§3.7.2 in [17]) suggests that on these platforms an entire TLB flush should be performed after each PMP configuration update to make the new configuration take effect. However, we find that this suggestion can be relaxed to enlarge the amount of effective PMP entries with TLB capacity. As a result, **TLB can significantly improve the scalability of PMP.**

Despite the performance benefits, careless utilization of hardware caching units can be dangerous [40], [41]. We show in §IV-B that utilizing TLB to cache PMP checking results does not undermine strong memory isolation, as long as the TLB status is carefully maintained.

C. Enclave TCB Reduction

Having a small TCB is one of the fundamental requirements for a TEE. Current works mainly use three different approaches for enclave TCB reduction: (1) delegate part of functionalities to the host kernel [1], [10], [11], (2) provide only a diminutive set of functionalities [42], [43], [44], [45], and (3) use modularized designs to customize a minimal TCB for each *Enclave Application* (EApp) [28], [30], [33], [36], [37], [46].

The first approach reduces the TCB size at the expense of sacrificing enclave autonomy [1], [10], [11]. An enclave is said to be autonomous if it has zero functional dependency on REE. Non-autonomous enclaves expose larger attack surfaces because they require extra REE-TEE interfaces for functional dependencies such as delegated system call handling, I/O operations, etc. Previous works [23] have demonstrated the exploitability of such attack surfaces based on Iago attacks [22], which are rooted in the insecure nature of the system call *Application Binary Interface* (ABI). On the other hand, autonomous enclaves are impervious to such threats since they do not expose these attack surfaces.

The second approach reduces TCB in a detrimental manner. TEEs taking this approach typically use micro-kernel-based enclave runtimes to provide core functionalities [42], [43], [44], [45]. However, such diminutive runtimes are usually too restricted to host general EApps. Consequently, the EApps must either relinquish certain capabilities or re-implement the absent functionalities themselves. The former case makes TEE less potent in practice while the latter reintroduces the trimmed TCB, rendering preceding efforts in vain.

The third approach avoids the shortages of the first two approaches. Previous works have taken the third approach based on LibOSes or unikernels [28], [30], [33], [36], [37], [46]. To achieve TCB reduction, one choice is static trimming, which starts with full functionalities and removes unused ones at build time. However, in scenarios such as serverless computing, the requirements of user-provided functions are typically unknown at enclave build time. Another choice is dynamic configuration, which lets the user defines the permissions of the enclave at enclave launch time. Under this approach, enclaves with the same EApp may have different

TCBs because of different user-defined permission tables. In COFFER, we take this approach to strike a balance between functionalities and TCB sizes.

III. THREAT MODEL

We assume a privileged attacker in our threat model. The attacker can have full control of the OS kernel and may attempt to undermine the confidentiality of the enclaves by trying to access the enclaves' memory. Besides, the attacker may try to manipulate the binary files of the EApps and EModules. The attacker may also manipulate all system call return values to attempt an Iago attack. On the other hand, we assume that all the platform hardware is correctly implemented and trusted and the platform hardware includes standard RISC-V PMP. Additionally, preventing DMA attacks requires specialized hardware such as IOMMU [47] and IOPMP [48]. Therefore, these proposals are orthogonal to our study. Finally, physical attacks, side-channel attacks and *Denial-of-service* (DoS) attacks are out of the scope of this work.

IV. DESIGN

COFFER aims to serve as a scalable and efficient TEE for standard commodity RISC-V platforms. It enables such devices to host cloud confidential computing. To this end, we design COFFER with the following goals in mind:

- G1. **Security**. COFFER should isolate enclaves' execution throughout their lifecycles. Besides, enclaves in COFFER should be autonomous.[1] Finally, COFFER's TCB size should be minimized.
- G2. **Scalability**. COFFER should support two types of scalability: *number scalability* and *memory scalability*. Number scalability demands COFFER to be able to host a large number of concurrent enclaves. Memory scalability demands that COFFER should not set additional limitations on enclave memory size or number of memory segments.
- G3. **Compatibility**. COFFER should provide both high *hardware compatibility* and high *software compatibility*. High hardware compatibility means that COFFER should be deployable on any RISC-V device that satisfies the standard specification [16], [17]. High software compatibility means that COFFER should support running general ELF files as the *Enclave Application*s (EApps).
- G4. **Efficiency**. COFFER should not introduce high performance overhead to the EApps. The system should be efficient even under heavy memory fragmentation.

A. Overview

The system overview of COFFER is illustrated in Figure 2. The trusted components are shown in grey areas. They are comprised of the enclaves, the *Security Monitor (SM)* and the hardware. An enclave consists of an EApp and several *Enclave Module*s (EModules). The EApp is the sensitive application to be protected. The EModules provide OS functionalities to the EApp, including system call handling, I/O operations, etc. A special EModule, the `EMod_Manager` (EMM in Figure 2),

[1]We say an enclave to be autonomous if it does not have critical functional dependencies on REE.

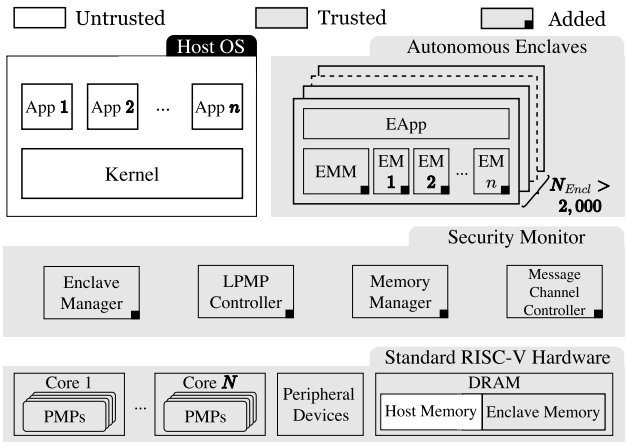


Fig. 2. Architecture overview of COFFER.

is compulsory for every enclave. It loads and manages all other EModules (EM_1 ~EM_n in Figure 2) of the enclave. The `EMod_Manager` loads these EModules on demand at run-time. We describe EModules in detail in §IV-C. COFFER enclaves support multi-threading and can concurrently execute on all system cores. The Security Monitor manages the enclaves. It is comprised of four components: the *Enclave Manager*, the *Memory Manager*, the *LPMP Controller*, and the *Message Channel Controller*. The Enclave Manager manages the lifecycles of the enclaves. The Memory Manager provides support for enclave dynamic memory allocation and enclave memory migration. The LPMP Controller manages LPMP contexts of the enclaves. It is the core Security Monitor component COFFER uses to achieve scalability. Finally, the Message Channel Controller establishes secure message channels for communications between REE and TEE.

The Enclave Manager and LPMP Controller provide isolation for the enclaves, as required by **G1**. The other two requirements of **G1**, autonomy and minimized TCB for enclaves, are achieved by the design of EModules. The LPMP Controller is the core component for **G2**. The Enclave Manager uses it to achieve number scalability and the Memory Manager uses it to achieve memory scalability. Meanwhile, COFFER is a software-based TEE. The only hardware primitive it relies on is PMP provided by standard RISC-V hardware. Therefore, COFFER has high hardware compatibility, as required by **G3**. Moreover, the EModules provide rich OS functionalities to support running general ELF files as EApps, which endows COFFER with high software compatibility, complementing **G3**. Finally, the LPMP Controller provides scalable enclave isolation with little performance overhead, enabling COFFER to achieve **G4**.

B. Scalable and Efficient Memory Isolation

LPMP is a scalable and efficient software framework for PMP virtualization. LPMP enables COFFER to provide both enclave number scalability and enclave memory scalability. For enclave number scalability, LPMP adopts a flexible TEE/REE memory partition and treats the host OS and the enclaves symmetrically. To achieve enclave memory scalability, LPMP virtualizes hardware PMP entries for each enclave. However, a simple trap-and-emulate virtualization for PMP is inefficient because traps are triggered at a high frequency.

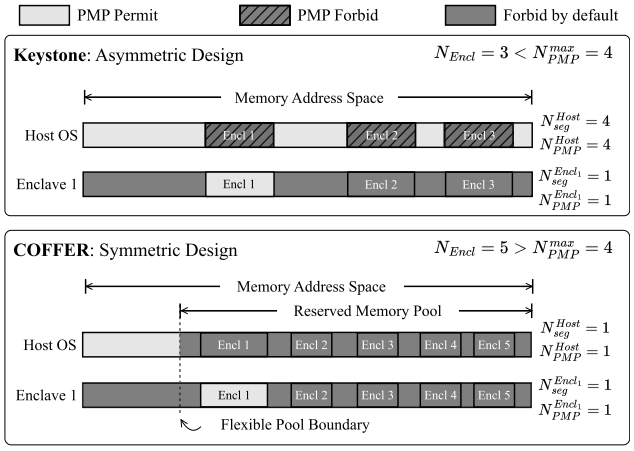


Fig. 3. Comparison of asymmetric and symmetric approaches for enclave isolation. In the asymmetric approach, the host OS and the enclaves are treated differently. In the host OS's view, the PMP entry of the least priority is used to allow access to all physical addresses. Other PMP entries are used to forbid access to enclaves' memory. The required number of PMP entries grows linearly with the number of enclaves. In the symmetric approach, the host OS is treated as a special enclave. The host OS's memory is consecutive and the enclaves allocate memory from a reserved memory pool. The boundary of the reserved memory pool is flexible. The required number of PMP entries for both the host OS and the enclaves is always 1.

In light of modern cache designs, we take an instruction/data split approach for LPMP management to ensure system efficiency. On certain RISC-V platforms, LPMP leverages the TLB for further performance improvement.

1) Symmetric Host/Enclave Treatment: Some previous RISC-V TEEs [10], [11] have used PMP for enclave memory isolation but failed to provide number scalability. They attribute the reason to the limited PMP hardware resources. Nevertheless, we observe that these TEEs treat the host OS and the enclave differently (asymmetric design in Figure 3) and the PMP resources actually exhaust in the host OS's view. For example, during execution of the host OS, Keystone uses the PMP entry with the least priority to allow memory accesses at all physical addresses. Then it uses several PMP entries with higher priority to prevent the host OS from accessing enclaves' memory. Consequently, the number of PMP entries required increases with the number of enclaves, which poses a limitation on the number of enclaves ($N_{PMP}^{Host} = N_{Encl} + 1 \leq N_{PMP}^{max}$).

However, in each enclave's view, only one PMP entry is required to allow accesses to its own memory. Hence, if we treat the host OS as a special enclave, which also only require one PMP entry, then the limitation in the number of enclaves is removed. To this end, LPMP introduces a flexible TEE/REE memory partitioning: a reserved memory pool is dedicated for enclave memory allocation and the host OS's memory is a contiguous region (symmetric design in Figure 3). In practice, the enclave memory pool is reserved at boot time, while actual allocations from this pool are fully dynamic at runtime. When enclaves are created, they request memory from the pool through the Memory Manager, which updates an ownership table to track which memory chunks belong to which enclave. When enclaves terminate, their memory is returned to the pool for reuse. To prevent the host OS from getting out of memory, we adopt a flexible boundary for the memory pool. The host OS can allocate memory from the boundary of the reserved memory pool. The flexible boundary ensures that if the host

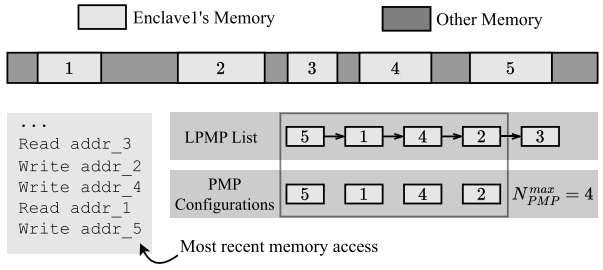


Fig. 4. Illustration of LPMP. When $N_{seg}^{Encl_i} > N_{PMP}^{max}$, the LPMP Controller in the Security Monitor loads the first N_{PMP}^{max} entries in the LPMP list as real PMP configurations. The LPMP list is arranged with the most-recently-used policy.

OS experiences memory pressure, it can allocate from the edge of the reserved enclave pool; if that boundary region is occupied by enclaves, the Memory Manager migrates enclave memory to other regions within the pool to make boundary memory available to the host OS.

In the symmetric treatment of the host OS and enclaves, the number of PMP entries required is always 1, which does not grow as the number of enclaves increases. As a consequence, COFFER is capable of supporting an unlimited number of enclaves in theory.

2) Virtualizing PMP Entries: Besides enclave number scalability, **G2** also demands COFFER to support enclave memory scalability, i.e., memory size and the number of memory segments should not be limited for the enclaves. However, the previously mentioned method does not support dynamic enclave memory allocation. Since the PMP hardware resources are not exhausted in the previous method, it can be trivially extended to support limited dynamic enclave memory allocation. Nevertheless, the number of memory segments an enclave can own is limited by the number of PMP entries ($N_{seg}^{Encl_i} \leq N_{PMP}^{max}$), which does not satisfy **G2**.

To overcome this limitation, LPMP provides virtualized PMP entries (which we call LPMP entries in the following) for each enclave. An LPMP entry contains the same information as a real PMP entry. However, the number of LPMP entries an enclave can possess is not limited. The LPMP Controller in the Security Monitor maintains a dynamic list of LPMP entries for each enclave and takes a trap-and-emulate approach to make every LPMP entry take effect (shown in Figure 4). During context switches into the enclave, the LPMP Controller loads N_{PMP}^{max} LPMP entries into real PMP CSRs. Memory accesses to addresses covered by other LPMP entries trigger traps into the Security Monitor. The LPMP Controller maintains the LPMP list in a most recently used order: when an LPMP entry is accessed, it is moved to the head of the list and loaded into a hardware PMP register. Entries at the tail of the list, those accessed least recently, are implicitly evicted from hardware PMP registers when new entries are loaded. This effectively implements an LRU replacement policy for what gets evicted, leveraging temporal locality in enclave memory access patterns. It is possible to implement some other policies like FIFO or Random policies, but they would likely perform worse for memory-intensive workloads due to their inability to exploit temporal locality. This way, we are effectively using PMP hardware resources as a cache of the LPMP entry list. Since the number of LPMP entries per enclave is not

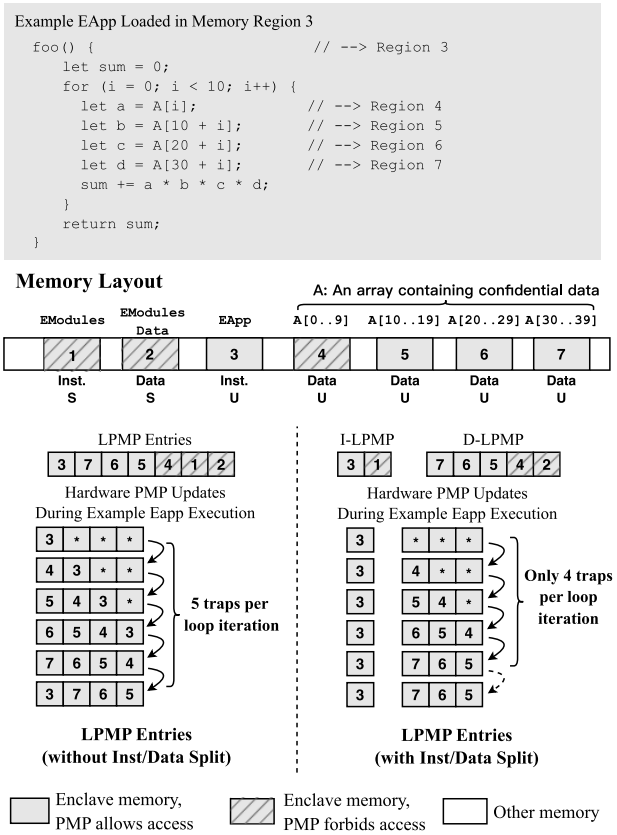


Fig. 5. Illustration of instruction/data split. Suppose an enclave with 7 memory regions is running an EApp with function `foo()`. 4 memory regions are used to store an object array `A`. Without instruction/data split, the LPMP entry for Region 3 is unloaded and reloaded in each loop iteration. With instruction/data split, the LPMP entry for Region 3 always takes effect and trap frequency is reduced.

limited, an enclave may own a theoretically unlimited amount of memory segments. Therefore, LPMP enables COFFER to complement **G2**.

3) Instruction/Data Split: Though the trap-and-emulate virtualization of PMP provides scalability, this approach can incur unacceptable performance overhead due to frequent LPMP traps. Such inefficiency conflicts with our **G4**. Nevertheless, we notice that a fair amount of traps are triggered by instruction memory accesses. As illustrated in Figure 5, the LPMP entry for instruction memory accesses is kicked out of real PMP hardware for every N_{PMP}^{max} data accesses. As a result, LPMP traps for instruction memory accesses are frequently triggered even if enclave instruction memory is highly localized and the corresponding LPMP entry is supposed to take effect constantly. To address this issue, we borrow the idea of instruction/data split from modern cache designs. To this end, the LPMP Controller reserves one or more PMP CSRs dedicated for enclave instruction memory and applies the most-recently-used policy for instruction and data PMP CSRs separately. With the instruction/data split approach, we significantly improve COFFER enclave execution efficiency.

4) TLB-Enhanced LPMP: In addition to the instruction/data split optimization which is universal for all standard RISC-V platforms, we propose another optimization for certain RISC-V processors. In §II-B, we mention that on some RISC-V processors, the PMP checking results are cached in the TLB when the MMU is on. We leverage this feature to further optimize LPMP efficiency. The theoretical scalability of LPMP depends on the active working set of memory regions rather than total enclave memory size. When the number of active memory segments exceeds available PMP entries ($N_{seg}^{active} > N_{PMP}^{max}$), trap frequency increases proportionally. However, the instruction/data split optimization reserves dedicated PMP entries for instruction memory, significantly reducing traps for code with high spatial locality. TLB-enhancement further extends effective PMP capacity by caching results for 2MB regions, enabling COFFER to maintain near-native performance even for large memory footprints with reasonable access locality. During an LPMP trap, the LPMP Controller loads the entry for the most recently hit memory region into real PMP configuration. The standard RISC-V PMP specification (§3.7.2 in [17]) suggests a full TLB flush after each PMP configuration update to ensure the new configurations take effect. This suggestion guarantees that PMP checking results cached in the TLB are always consistent with real PMP configurations. However, this consistency can be relaxed in Coffer. This is because we only need the TLB-cached PMP checking results to be consistent with the enclave's LPMP list instead of real PMP configurations. To this end, on each LPMP trap, the LPMP Controller only flushes the most recently filled TLB entry with the prohibitive PMP checking result that triggers the trap. All other entries remain in the TLB. The selective TLB invalidation strategy minimizes flush overhead by preserving cached PMP results for non-conflicting memory regions, making LPMP practical even under memory-intensive workloads. As a result, previous PMP checking results continue to take effect even if the corresponding LPMP entries are not loaded in real PMP CSRs. This way, we effectively increase the number of PMP entries using TLB capacity. To fully utilize this optimization, we perform enclave memory allocation at 2MB granularity, the size of mega-pages in the RISC-V SV39 address translation standard. This allows for TLB-cached LPMP entries to cover larger enclave memory regions.

Nevertheless, frivolous utilization of caching units can be catastrophic [40], [41]. We need to ensure our utilization of TLB does not undermine the security of COFFER enclaves. To this end, we propose two security principles for TLB maintenance: (1) *exclusive ownership of TLB* and (2) *freshness guarantee*. The principle of exclusive ownership of TLB demands that at any moment all TLB entries must belong to a single entity, either the host OS or a particular enclave. To enforce this principle, the LPMP Controller performs an entire TLB flush upon switching into or out of an enclave. The principle of freshness guarantee demands that TLB-cached PMP checking results must always be up-to-date, consistent with the enclave's LPMP list. When an enclave's LPMP list is modified (for example, during enclave memory allocation), the LPMP Controller also flushes the entire TLB to guarantee freshness of the entries. This avoids stale LPMP entries to be cached in the TLB. The two principles ensure security for the TLB-enhanced LPMP, which significantly improves the efficiency of COFFER enclaves under heavy memory fragmentation.

TABLE III
DEFINITION OF SYMBOLS

N_{Encl}	Number of concurrent enclaves
N_{PMP}^{max}	Number of maximum PMP entries per core
$N_{seg}^{Host/Encl_i}$	Number of memory segments of Host/Enclave$_i$
$N_{PMP}^{Host/Encl_i}$	Required number of PMP entries of Host/Enclave$_i$

C. Autonomous Adaptive Enclaves

To meet **G1**, COFFER enclaves do not rely on the host OS for critical OS functionalities. For this purpose, COFFER equips each enclave with a set of EModules to provide OS functionalities. However, providing all OS functionalities in the enclave leads to TCB bloating. To address this problem, COFFER introduces a user-defined EModule permission table to customize the TCB for each enclave.

1) EModule: An EModule is a modularized software entity that provides OS services, including system call handling and page mapping, to the enclave. An EModule may also provide basic functionalities (such as memory allocation) to other EModules. Each EModule focuses on a single task and is as small as possible. A special EModule, `EMod_Manager`, is compulsory for every enclave. It provides the most fundamental functionalities such as enclave boot-up, page mapping, simple system call handling, and loading other EModules. With EModules, enclaves can provide all functionalities required by the EApp on their own, without having critical functional dependency over REE, satisfying the requirement for autonomy. Besides, handling OS operations inside the enclaves reduces domain switch frequency, improving overall system performance.

2) Dynamically Assembled Enclave: COFFER uses EModules to dynamically assemble an adaptive enclave runtime. When an EApp invokes certain OS functionality provided by an EModule, the `EMod_Manager` verifies and loads the EModule on demand. The dynamic loading approach avoids unused components being included in the enclave and reduces memory consumption. In addition, there is a user-defined EModule permission table for each enclave. When an EModule is invoked, the `EMod_Manager` checks whether the enclave has permission to load the EModule. If the EApp does not have permission to the EModule, then the `EMod_Manager` aborts enclave execution and exits immediately. To guarantee the integrity of the permission table, it is combined with the EApp for the enclave digest report. This way, the user can customize TCBs of the enclaves. Even for enclaves with the same EApp, the TCBs can differ because of different permission configurations.

3) Autonomous Enclave: Enclaves in COFFER do not have critical functional dependencies on the host OS. The EModules handle all system calls and interrupts for the enclaves. This design largely reduces the attack surface of the enclaves and prevents potential threats such as Iago attacks [22]. However, although COFFER enclaves do not proxy system calls to the host, they may still leverage the host OS to improve I/O efficiency. For example, the Security Monitor provides a special interface for bulk data transfer between the host OS and the enclaves. To use this interface, the sender first writes data to a newly allocated memory region and then tells the Security Monitor to change the ownership of the region to the receiver. Throughout the entire process, the principle of exclusive ownership is enforced. This way, COFFER excludes I/O drivers (which takes a large portion of OS kernels) from the TCB of the enclaves and enables efficient I/O for the enclaves. Note that as mentioned in §III, preventing DMA attacks is orthogonal to our study.

V. IMPLEMENTATION

In this section, we describe how we implement the prototype of COFFER. We implement the Security Monitor prototype on top of OpenSBI [49]. Besides, we implement six minimal EModules: `EMod_Manager`, `EMod_Alloc`, `EMod_Futex`, `EMod_UART`, and `EMod_VFS`. Our COFFER prototype supports HiFive Unmatched board [26], VisionFive 2 board [27] and D1 Nezha board [50].

A. EModules

EModules are dynamically loadable functioning modules which provide critical low-level functionalities to the enclaves. They are supposed to be minimal, flexible, and self-contained. As a result, the enclaves do not have to rely on REE for these functionalities and can be autonomous. Only `EMod_Manager` is compulsory for every enclave. Other EModules are loaded on demand. We port all EModules except `EMod_Manager` from production-ready libraries to ease the engineering effort and ensure functional correctness.

1) EMod_Manager: The `EMod_Manager` initializes the enclave, manages other EModules and handles a few simple system calls. During enclave initialization, the Security Monitor loads the binary of the `EMod_Manager` to the primary memory partition of the enclave and then transfers the control flow to the `EMod_Manager` to boot the enclave. The EApp is loaded after enclave initialization. After loading the EApp, the `EMod_Manager` calculates a hash digest of the enclave and can output the digest with secure device operations. During EApp execution, when the EApp invokes a system call, the `EMod_Manager` dispatches it to the corresponding EModule. If the EModule is not loaded at the moment, the `EMod_Manager` requests the EModule image from the host OS via secure message channels. After receiving the image, it first attests the integrity of the image with the digital signature contained inside the image. Only authorized EModule developers can provide valid signatures.

2) Other EModules: In our prototype, we implement four other EModules: `EMod_Alloc` for enclave heap memory allocation, `EMod_Futex` for futex support, `EMod_UART` as UART device driver, and `EMod_VFS` for RAMFS support. We port `EMod_Alloc` from the musl libc allocator [51], `EMod_Futex` and `EMod_UART` from the Linux kernel. For `EMod_VFS`, we port from Unikraft's `vfscore` library [30]. These EModules together with the `EMod_Manager` provide 50+ most commonly used system calls.

B. Security Monitor

We implement COFFER's Security Monitor based on OpenSBI v0.9 [49]. There are four components in the Security Monitor: the Enclave Manager, the Memory Manager, the LPMP Controller, and the Message Channel Controller.

TABLE IV
COFFER SECURITY MONITOR ABI. E(H) MEANS THE ABI CAN ONLY BE INVOKED BY THE ENCLAVES (THE HOST OS). E/H MEANS THE ABI IS AVAILABLE FOR BOTH PARTIES

ABI Name	Caller	Functionality
`create_encl`	H	Create and initialize enclave
`enter_encl`	H	Load EApp into enclave
`pause_encl`	E	Pause the enclave
`resume_encl`	H	Resume paused enclave
`exit_encl`	E/H	Exit and clean up enclave
`attest_encl`	H	Attest enclave integrity
`mem_alloc`	E	Allocate memory
`mem_transfer`	E/H	Transfer memory ownership
`channel_esta`	E/H	Establish message channel
`channel_send`	E/H	Send message to channel
`channel_stop`	E/H	Terminate message channel

Table IV shows the `ecall` ABIs that the Security Monitor provides. The first six ABI calls are provided by the Enclave Manager. The allocation call is handled by the Memory Manager. Three are provided by the Message Channel Controller.

1) Enclave Manager: The enclave manager handles six `ecalls` for creating, entering, exiting, pausing, resuming, and attesting enclaves. It maintains the contexts of all enclave threads. We modify the `medeleg` CSR to let the Enclave Manager to intercept all `ecalls` from both U-Mode and S-Mode and redirect the host system calls back to the OS kernel.

2) Memory Manager: The memory manager maintains the reserved enclave memory pool and takes care of enclave memory allocation and migration. We split the memory pool into chunks of 2MB to ease memory management. 2MB is also the size of mega-pages in RISC-V's SV39 standard [17]. There is a memory ownership table inside the Memory Manager. During the allocation, the memory manager updates the memory ownership table and then informs the LPMP Controller to update the LPMP context. If there is no contiguous memory region for the required size, the Memory Manager asks the enclave to separate the allocation into multiple requests. The Memory Manager also supports memory migration to adjust the flexible TEE/REE memory pool boundary. During the memory migration, the Memory Manager copies the memory and updates relevant contexts, including CSRs and page tables, and then it cleans up the original memory region and updates the memory ownership table.

3) LPMP Controller: The LPMP Controller maintains the LPMP entry list for each enclave. The list contains addresses and sizes of all memory segments owned by the enclave. All list entries are allocated from a static array whose size equals the number of chunks in the memory pool. During context switches, the LPMP Controller fills up PMP CSRs with entries in the LPMP list one by one. It uses NAPOT mode for aligned addresses and TOR mode for misaligned addresses. Each TOR region consumes two PMP CSRs. On an LPMP fault, the LPMP Controller checks whether the accessed address belongs to the enclave. If so, the LPMP Controller brings the hit region to the head of the list and updates the PMP configurations accordingly. To enable instruction/data split, the LPMP Controller reserves at least one PMP entry for enclave instructions. After each PMP configuration update, the LPMP Controller flushes the TLB entry using instruction, `sfence.vma a0, zero`, where `a0` stores the virtual address stored in CSR `mtval`. This way, the TLB is leveraged to effectively enlarge the capacity of PMP. To ensure strong isolation, the LPMP Controller enforces the two principles introduced in §IV-B: exclusive ownership of TLB and freshness guarantee. To this end, upon each domain switch or when the LPMP list is updated, the LPMP Controller performs an entire TLB flush using the instruction `sfence.vma`.

4) Message Channel Controller: Message channel is a utility to support host-enclave communication. There are 3 `ecalls` for message channels: `channel_listen`, `channel_send` and `channel_stop`. The `channel_listen` call have three parameters: the sender's enclave ID, the address of the receiver buffer, and the maximum buffer length. The `channel_send` call also has three parameters: the receiver's enclave ID, the address of the message, and the length of the message. To send a message over a channel, the receiver must first start listening in advance. When the sender invokes `channel_send`, the Message Channel Controller (MCC) first verifies the ownership of the physical memory of the message source according to the ownership table inside the Memory Manager. If the message is valid, then the Message Channel Controller changes the `mstatus` CSR to make the core fetches instructions in physical addresses but reads/stores data in virtual addresses and performs memory copying.

VI. SECURITY ANALYSIS

A. Protection Against Different Attacks

1) Protection Against Malicious OS Kernel: A malicious OS kernel may launch attacks towards COFFER enclaves and the Security Monitor. It may attempt to access memory owned by the enclaves or the SM. We prevent this by enforcing physical memory isolation based on LPMP(§§ IV-B), which forbids the host OS from accessing the memory of other entities. Since COFFER enclaves are autonomous (§§ IV-C), the enclaves have minimal functional dependencies over the host OS, which avoids attacks based on OS operations such as system call handling [22], [23] and page mapping [24]. The malicious kernel may also attempt to replace or inject code to EApp/EModule binaries. However, the Security Monitor provides an attestation `ecall` to verify enclave integrity, which is common in TEE designs such as Keystone [10].

2) Protection Against Compromised Enclave: Though the integrity of EApps is verified through attestation, the EApps may contain vulnerabilities and can be compromised [52]. Therefore, we also prevent potential attacks from enclaves. First, the strong isolation provided by LPMP prevents the enclaves from accessing each other's memory. Second, we do not allow the enclaves to access the host OS's memory. All communication between REE and TEE must be performed using the secure message channel provided by the Security Monitor. Lastly, the COFFER enclaves do not have shared resources, including EModules, physical memory, TLB entries, and page tables, This avoids potential data leakage from the shared resources.

3) I/O Security: For MMIO peripheral devices, COFFER can add special LPMP entries for the host/enclaves to protect the I/O operations. However, there are also peripheral devices

TABLE V
CODE SIZES OF PRIVILEGED COMPONENTS IN COFFER

EModule	LoC
EMod_Manager	4,430
EMod_Alloc	595
EMod_Futex	539
EMod_UART	898
EMod_VFS	5,490

(a) LoC counting of EModules.

Security Monitor	LoC
OpenSBI	14,830
Enclave Manager	1,119
Memory Manager	735
LPMP Controller	191
Message Channel	176

(b) LoC counting of the SM.

with DMA. Preventing attacks from such peripheral devices requires additional specialized hardware such as IOMMU [47] or IOPMP [48]. IOPMP is still under active development stage, and several RISC-V vendors have announced plans to implement it in future SoCs. IOMMU has already been ratified in non-ISA specifications, but currently only a few commodity RISC-V SoCs support it. It is feasible to retrofit these features into COFFER without significant architectural changes. COFFER is designed with forward compatibility for emerging RISC-V security extensions. Currently, COFFER supports enclave bulk I/O operations by memory ownership transfer (Section IV-C). All private data should be encrypted before being sent to or received from the enclaves.

4) Side-Channel Attacks: While §III excludes side-channel attacks from our threat model in general, we note that COFFER's architectural design provides defense against certain side-channel subsets as a derivative advantage. Previous works [41] have demonstrated various side-channel attacks on commodity RISC-V devices. Defending against cache-based side-channel attacks requires hardware modification and remains orthogonal to our study. However, COFFER provides strong mitigation against TLB-based or page-table-based side-channel attacks as a natural consequence of its design: each enclave manages page tables independently, and the Security Monitor enforces the principle of exclusive ownership of TLB through full TLB flushes on domain switches.

B. TCB

The TCB of an enclave in COFFER contains an User Mode EApp, one or more Supervisor Mode EModules and the Machine Mode Security Monitor. All EModules within a single enclave execute in the same S-mode protection domain and are not shared across enclaves; each enclave assembles its own private instances according to its permission table. The EApp is a general Linux ELF file so its TCB reduction is left to the EApp developers. Here we focus on the enclave components running in the privileged modes. Table V shows the code sizes of the privileged components in COFFER. Table V(a) lists the *Lines of Code* (LoC) of each EModule. Note that the most complex EModule is the `EMod_VFS`, which contains 5,490 LoCs. However, this size is comparable to the file-system related part in production-level enclave LibOSes such as Graphene-SGX [29]. Table V(b) shows the detailed TCB breakdown of the Security Monitor in COFFER. The `EMod_Manager` and the Security Monitor are compulsory for every enclave. They consist of 21,473 LoC in total. An EApp can only load the EModules it has access permission to. This may significantly reduce enclave runtime TCB sizes in COFFER.

1) EModule Ecosystem Security: COFFER's current implementation provides signature-based integrity verification for EModules, where the `EMod_Manager` performs ECDSA signature verification before loading any module. The platform owner controls the signing key, establishing a clear chain of trust similar to standard TEE practice. Minimal module sizes facilitate auditing, and on-demand lazy loading let enclaves load only necessary modules, reducing attack surface. While the current prototype focuses on foundational signing and verification, comprehensive lifecycle management features such as automated version tracking, formal revocation mechanisms, and runtime patching represent important directions for future work. Supply-chain attacks targeting the signing infrastructure remain an open research question affecting all TEE systems.

2) Security–Performance Tradeoffs: COFFER embodies several deliberate security performance tradeoffs. The fundamental LPMP design trades modest trap-handling overhead for comprehensive fine-grained memory isolation without hardware modification. TLB-enhancement optimization leverages TLB caching for performance gains while has two TLB flushing strategies which may have different secrecy. The autonomous enclave design with EModules trades increased enclave TCB size for substantially reduced attack surface by eliminating OS dependencies that enable Iago and controlled-channel attacks.

VII. EVALUATION

A. Research Questions

In this section, we evaluate the prototype of COFFER based on our implementation (§V). For data consistency, we conduct all performance evaluations on the same HiFive Unmatched board [26] with four cores SiFive U74 and 16GB DRAM. All results are obtained by averaging over 30 repeated experiments. We consider five research questions in our evaluation:

RQ1. How is COFFER's scalability?
RQ2. How is COFFER's compatibility?
RQ3. How efficiently do COFFER enclaves execute under highly fragmented physical memory?
RQ4. How much overhead is incurred on the benchmarks?
RQ5. How does COFFER compare to the state-of-the-art RISC-V TEEs?

B. Scalability

For **RQ1**, we evaluate COFFER's scalability from two aspects: number scalability and memory scalability.

1) Number Scalability: To evaluate COFFER's number scalability, we observe how the performance overhead changes when the number of concurrent enclaves scales up. We launch up to 2,048 concurrent enclaves distributed across all four cores, each running the `sha512` benchmark from the RV8 benchmark suite [53]. We measure the average enclave execution time and the execution time of the slowest enclave and compare with the single enclave execution time. The result is shown in Figure 6(a). From the results we conclude that the enclave performance overhead does not scale with the number of concurrent enclaves, and the overhead is within 1% even for 2,000+ concurrent enclaves.

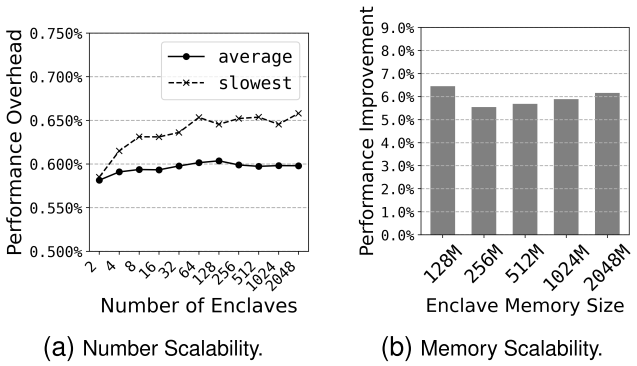


Fig. 6. COFFER scalability. (a) For number scalability, we run varying numbers of enclaves and compare them to single enclave execution. The overhead does not scale as the number of enclaves increases. (b) For memory scalability, we run enclaves with different memory sizes and compare the execution time to Linux processes. The overhead is within 7% with enclave memory sizes up to 2GB.

2) Memory Scalability: To evaluate COFFER's memory scalability, we run enclaves of different memory sizes and compare the execution time with Linux. We run the `memrate` worker of `stress-ng` [54] in the enclave for this purpose. We make the `stress-ng` worker allocate memory buffers of different sizes, from 128MiB to 2GB. The memory access rate is set to 2,000 reads and 1,000 writes per second. The results are shown in Figure 6(b). It can be observed that the enclaves incur around 6% performance overhead compared to Linux. The performance difference may originate from the prototype memory allocator EModule `EMod_Alloc`, which is designed to be lightweight but not optimized to be cache-friendly. A more sophisticated memory allocator can be used to improve the performance of the enclaves.

C. Compatibility

For **RQ2**, we evaluate COFFER's compatibility from both hardware and software perspectives.

1) Hardware Compatibility: COFFER is a software-based TEE. The hardware primitive it relies on for memory isolation is PMP, which is defined in the RISC-V standard specification and supported by all RISC-V compliant processors. Therefore, COFFER can be deployed on any standard RISC-V devices, including off-the-shelf commodity boards. We have tested COFFER on different RISC-V platforms: the SiFive Unmatched [26], the VisionFive 2 [27], and the XiangShan (Nanhu) [55] on Prodigy S7-19P Logic System. While not every hardware feature we used in LPMP optimization such as caching PMP checks results in the TLB is available in all these three platforms, COFFER's behavior on these three platforms is consistent during our experiments. The results are shown in Figure 7.

2) Software Compatibility: COFFER supports statically linked ELF binaries as EApps. Currently, ELF binaries compiled with `musl-libc` [51] and `newlib` [56] are supported. In our prototype, we support 50+ most commonly used system calls, which are already enough for COFFER to host a wide range of real-world applications as the EApps, including Redis, SQLite3, DarkNet, zstd, etc. Besides, since COFFER tries to remove the essential dependencies on the host OS, it is theoretically compatible with any host environment.

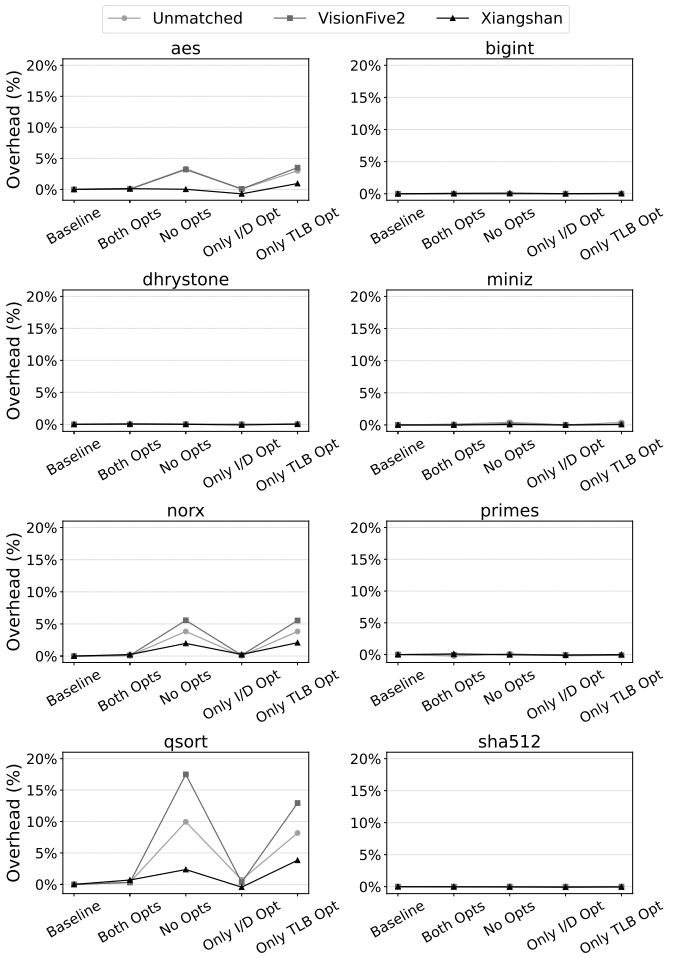


Fig. 7. COFFER compatibility. Performance overheads on RV8 benchmarks with different optimization settings of COFFER on different RISC-V platforms (Hifive Unmatched, StarFive VisionFive 2, and XiangShan Nanhu).

D. Efficiency Under Memory Fragmentation

To evaluate COFFER enclaves performance under highly fragmented physical memory (**RQ3**), we perform a worst-case simulation: all physical pages of the enclave are not adjacent to each other. Besides, we assume that TOR is the only available PMP address matching mode. We run RV8 benchmarks with large memory consumption and the `vm` worker of `stress-ng` in enclaves under different settings: (1) no PMP protection as the performance baseline, (2) LPMP without optimization, (3) LPMP with instruction/data split, (4) LPMP with TLB-enhancement, and (5) LPMP with both optimizations. For the `vm` worker we choose two different memory access patterns: `vm-write64` for sequential memory access and `vm-swap` for random memory access. The experiment results are shown in Figure 8. Results for LPMP with no optimization are not shown for RV8 benchmarks and `stress-ng vm-swap`. This is because enclaves under these configurations cannot finish execution in a reasonable time (these enclaves cannot finish within an hour while enclaves under the baseline configuration take only <10 seconds). From the experiment results we find the TLB-enhancement optimization is much better than the instruction/data split for RV8 benchmarks. However, for `stress-ng` workers, instruction/data split is much better. The difference originates

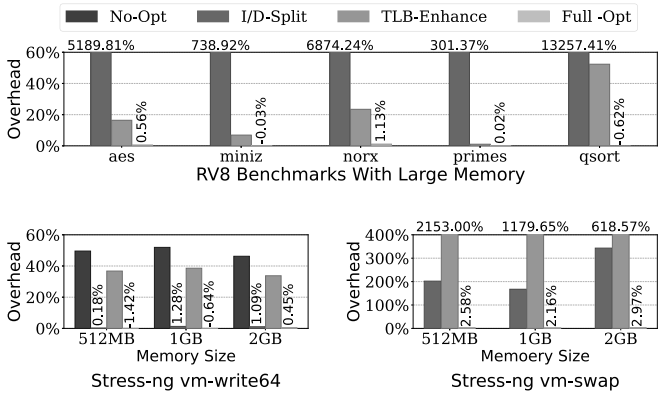


Fig. 8. LPMP overhead under worst-case memory fragmentation. Results for LPMP with no optimization are not shown for RV8 benchmarks and `stress-ng vm-swap` as enclaves under these configurations cannot finish execution in a reasonable time.

from that the `stress-ng vm` workers only involve memory access while the RV8 benchmarks contain more computation instructions. In all experiments, LPMP with both optimizations incur only ≤ 3% overhead compared to the baseline. Therefore, the two optimizations together enable COFFER enclaves to execute at nearly-native speed even under heavy physical memory fragmentation.

E. General Performance

To answer **RQ4**, we tested COFFER on both benchmark suites and real-world applications.

1) Benchmark Suites: We evaluate the general performance overhead of COFFER using the RV8 benchmarks [53]. All benchmarks are statically compiled with `newlib` [56]. We run the same ELF binaries both as EApps and as Linux processes. And we measure the time the enclaves and Linux processes take to finish. Figure 9(a) shows the results. For all benchmarks, the performance overhead is within 5% for enclaves in COFFER compared to Linux. For benchmarks with larger memory usages, such as `qsort`, `aes` and `norx`, zeroing enclave memory on cleaning up incurs additional overhead. For `primes` and `qsort`, we observe a shorter execution time for enclaves than Linux. This is because the COFFER enclaves in our prototype switch back to Linux once per 100 timer interrupts to reuse the Linux scheduler. This implementation triggers fewer task switches than Linux processes and therefore shortens the execution time. However, this should not be considered as a performance advantage of COFFER.

2) Real-World Applications: We evaluate COFFER on two real-world applications: DarkNet [57] and SQLite3 [58], to simulate confidential neural network inferences and confidential databases. We run them as EApps and compare the performance to Linux processes. The results are shown in Figure 9(b). For secure neural network inferences, COFFER enclaves have comparable performance to Linux processes. For secure databases, COFFER enclaves running write-intensive benchmarks have higher performance overhead. This is because the EModule `EMod_VFS` is a lightweight *RAM File System* (RAMFS) implementation. A more sophisticated file system EModule can be used to improve performance. COFFER enclaves hosting read-intensive benchmarks are more efficient and only incur less than 5% performance overhead.

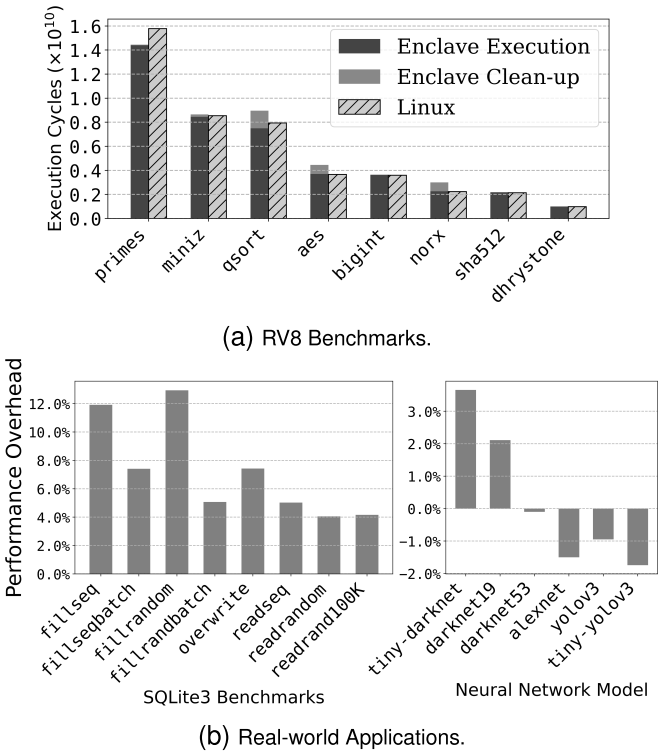


Fig. 9. Performance Comparison. We compare the performance of COFFER enclaves to Linux processes on the RV8 benchmarks and real-world applications. COFFER enclaves incur less than 5% performance overhead on RV8 benchmarks and neural network inferences, less than 15% overhead for SQLite3 databases.

F. Comparison

For **RQ5**, We compare COFFER to the state-of-the-art RISC-V TEEs. The results are shown in Table VI. Our comparison shows that COFFER is the only RISC-V TEE that with both high scalability and compatibility. Besides, COFFER is also the only RISC-V TEE that provides autonomous adaptive enclaves. Many RISC-V TEEs depend on customized hardware so it is hard to conduct fair performance comparisons with them directly. Other software-based TEEs currently do not support our device. Therefore, we compare COFFER's performance with other RISC-V TEEs based on their published results on RV8 benchmarks, the most commonly used RISC-V benchmark suite. A direct quantitative performance comparison for workloads with highly fragmented memory would require substantial modifications to baseline systems and is highly duplicated to our works. Existing software-based TEEs like Keystone cannot support enclaves with such fragmented memory layouts. Our feature-based comparison and worst-case fragmentation experiments in §VII demonstrate that COFFER is the only evaluated system capable of executing workloads under extreme memory fragmentation (achieving ≤ 3% overhead), representing a qualitative breakthrough from architecturally infeasible to feasible and efficient. There are no data for Sanctum, TIMBER-V, and Composite Enclaves. This is because Sanctum and TIMBER-V are proposed before the RV8, and Composite Enclaves focus on device operations and are not evaluated on RV8. We can see from publicly available data that COFFER has comparable performance with other RISC-V TEEs.

TABLE VI

COMPARISON WITH THE STATE-OF-THE-ART RISC-V TEES. N_{PMP}^{max} MEANS THE MAXIMUM AVAILABLE NUMBER OF PMP ENTRIES.N_{Reg}^{max} MEANS THE MAXIMUM AVAILABLE NUMBER OF DRAM REGIONSPROPOSED IN SANCTUM. GENERAL BINARY MEANS ENCLAVES CAN HOST UNMODIFIED BINARIESAS EAPPS. DEDICATED SDK MEANS ENCLAVES CAN ONLY SUPPORT EAPPSDEVELOPED WITH A SPECIAL SDK.OVERHEAD ON RV8 BENCHMARKS ARE FROM THE ORIGINAL PAPERS. COFFER IS THE ONLY RISC-V TEE WITH HIGH SCALABILITY,HIGH COMPATIBILITY, AND AUTONOMOUS ADAPTIVE ENCLAVES

Name	Enclave Number	Enclave Memory Segment Number	Hardware Requirement	EApp Compatibility	Enclave Autonomy	Adaptive TCB	Overhead on Benchmarks (RV8)
Keystone	$\leq N_{PMP}^{max}$	$\leq N_{PMP}^{max}$	RISC-V Standard	General Binary	✗	✗	$< 1\%$
CURE	13	1	Customized	General Binary	✓/ ✗	✗	$< 10\%$
Sanctum	$\leq N_{Reg}^{max}$	1	Customized	Dedicated SDK	✗	✗	No data
TIMBER-V	No Limitation	No Limitation	Customized	Dedicated SDK	✗	✗	No data
Penglai	No Limitation (Tested for $1,000+$)	No Limitation	Customized	POSIX-compatible	✗	✗	$< 5\%$
Composite Enclaves	$\leq N_{PMP}^{max}$	$\leq N_{PMP}^{max}$	Specialized Peripheral	General Binary	✗	✗	No data
COFFER	No Limitation (Tested for $2,000+$)	No Limitation	RISC-V Standard	General Binary	✓	✓	$< 5\%$

VIII. RELATED WORKS

A. VM-Based TEEs

VM-based TEEs are becoming more popular in the industry, including AMD SEV [3], Intel TDX [2], and Arm CCA [7]. Such TEEs are also autonomous by design. However, compared to VM-based TEEs, COFFER enclaves have significantly smaller TCBs due to the EModules design. In addition, VM-based TEEs are not primary options on commercially available RISC-V platforms because of the lack of H-extension support [17]. Nevertheless, LPMP in COFFER can also be used to support VM-based enclaves on RISC-V since the LPMP list provides similar functionalities that resemble the permission tables in VM-based TEEs.

B. RISC-V Security Extensions

There are several security extensions proposed in the RISC-V community. CoVE [14] is a design for confidential VMs on RISC-V. HPMP [15] is a proposal to support table-based memory protection on RISC-V. IOMMU [47] and IOPMP [48] are specialized hardware for I/O security. Smepmp [59] is a proposal to enhance the PMP mechanism. Sspmp [60] is proposed as S-mode PMP. However, these proposals are still under development and have not been merged into the mainline of RISC-V specifications [16], [17], [18]. Therefore, PMP is the most generic security feature on RISC-V. We expect this situation to last in the near future.

C. Other RISC-V TEEs

Keystone [10] is a software-based modularized TEE framework for RISC-V. CURE [61] provides different types of enclaves. Sanctum [20] is an SGX-style TEE on the RISC-V platform. TIMBER-V [13] is a RISC-V TEE that provides isolated enclaves. Penglai [12] is a RISC-V TEE targeted at scalable cloud scenarios. Composite Enclaves [11] is a RISC-V TEE built on top of Keystone for specialized peripheral devices. Compared to these RISC-V TEEs, COFFER is the only one that does not have limitation on the number of enclaves or enclave memory segments but deployable on standard RISC-V platforms.

D. TEEs on Other Architectures

Intel SGX [1] uses memory encryption to protect the confidentiality of the enclaves. GrapheneSGX [29], Occlum [46] and CubicleOS [36] are built on top of SGX, using LibOSes to support general applications. HyperEnclave [62] runs SGX programs on AMD servers with a Rust monitor. On the Arm platform, TrustZone [63] provides an isolated domain called the *secure world*. Several Arm TEEs [64], [65], [66] use virtualization-based isolation with TrustZone to provide more isolated domains as enclaves. Sanctuary [67] provides SGX-style enclaves by isolating them with Arm TZASC. Arm has also announced CCA, a new TEE for confidential VMs. Shelter [21] provides user-space enclaves based on Arm CCA.

IX. CONCLUSION

In this paper, we propose COFFER, which is a scalable and efficient TEE for commodity RISC-V platforms. COFFER resolves the dilemma between scalability and compatibility in RISC-V TEEs. It introduces a software framework LPMP for efficient and secure PMP virtualization. Besides, it uses EModules to provide efficiency and enclave autonomy. We implement a prototype of COFFER and our evaluation results show that COFFER can support $2,000+$ concurrent enclaves with minor performance overhead and the performance overhead is low even under heavy physical memory fragmentation. To the best of our knowledge, COFFER is the first RISC-V TEE that can provide scalability on commodity RISC-V devices.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and COMPASS members for their insightful comments.

REFERENCES

[1] (2015). *Product Change Notification 114074-00*. [Online]. Available: https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf

[2] Intel Corporation. (2014). *Intel Trust Domain Extensions*. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html

[3] T. W. D. Kaplan and J. Powell. (2016). *Amd Memory Encryption*. [Online]. Available: https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[4] (2021). *Google Cloud*. [Online]. Available: https://www.wired.com/story/google-cloud-confidential-virtual-machines/

[5] (2023). *Aws Ec2 With Amd SEV-SNP*. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html

[6] (2021). *Microsoft Azure*. [Online]. Available: https://mspoweruser.com/microsoft-cloud-provider-confidential-virtual-machines/

[7] (2021). *Arm Confidential Compute Architecture*. [Online]. Available: https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[8] Y. Xu et al., "Towards developing high performance RISC-V processors using agile methodology," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 1178–1199.

[9] RISC-V International. (2023). *Frequently Asked Questions (FAQ)*. [Online]. Available: https://riscv.org/about/faq/

[10] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst.*, New York, NY, USA, Apr. 2020, pp. 1–16, doi: 10.1145/3342195.3387532.

[11] M. Schneider, A. Dhar, I. Puddu, K. Kostiainen, and S. Capkun, "Composite enclaves: Towards disaggregated trusted execution," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 630–656, Nov. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9309

[12] E. Feng et al., "Scalable memory protection in the PENGLAI enclave," in *Proc. 15th USENIX Symp. Operating Syst. Design Implement.*, Jul. 2021, pp. 275–294. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/feng

[13] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, Feb. 2019, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/timber-v-tag-isolated-memory-bringing-fine-grained-enclaves-to-risc-v/

[14] R. Sahita et al., "CoVE: Towards confidential computing on RISC-V platforms," 2023, *arXiv:2304.06167*.

[15] D. Du, B. Yang, Y. Xia, and H. Chen, "Accelerating extra dimensional page walks for confidential computing," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Oct. 2023, pp. 654–669, doi: 10.1145/3613424.3614293.

[16] K. A. Andrew Waterman. (2019). *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[17] (2023). *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/riscv-isa-release-056b6ff-2023-10-02/riscv-privileged.pdf

[18] (2023). *RISC-V Specification Status*. [Online]. Available: https://wiki.riscv.org/display/HOME/Specification+Status

[19] A. Saha and S. Jindal, "EMARS: Efficient management and allocation of resources in serverless," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 827–830.

[20] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 857–874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[21] Y. Zhang et al., "SHELTER: Extending arm CCA with isolation in user space," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 6257–6274. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-yiming

[22] S. Checkoway and H. Shacham, "Iago attacks: Why the system call API is a bad untrusted RPC interface," in *Proc. 18th Int. Conf. Architectural support Program. Lang. operating Syst.*, New York, NY, USA, Mar. 2013, pp. 253–264, doi: 10.1145/2451116.2451145.

[23] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, Nov. 2019, pp. 1741–1758, doi: 10.1145/3319535.3363206.

[24] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 640–656.

[25] K. Shang, W. He, and S. Zhang, "Review on security defense technology research in edge computing environment," *Chin. J. Electron.*, vol. 33, no. 1, pp. 1–18, Jan. 2024. [Online]. Available: https://cje.ejournal.org.cn/en/article/doi/10.23919/cje.2022.00.170

[26] (2021). *Hifive Unmatched*. [Online]. Available: https://www.sifive.com/boards/hifive-unmatched

[27] (2023). *Vision Five 2*. [Online]. Available: https://doc-en.rvspace.org/Doc_Center/visionfive_2.html

[28] K. Boos, N. Liyanage, R. Ijaz, and L. Zhong, "Theseus: An experiment in operating system structure and state management," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI 20)*, Nov. 2020, pp. 1–19. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/boos

[29] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 645–658. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[30] S. Kuenzer et al., "Unikraft: Fast, specialized unikernels the easy way," in *Proc. 16th Eur. Conf. Comput. Syst.*, New York, NY, USA, Apr. 2021, pp. 376–394, doi: 10.1145/3447786.3456248.

[31] H. Lefeuvre et al., "FlexOS: Towards flexible OS isolation," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, Feb. 2022, pp. 467–482, doi: 10.1145/3503222.3507759.

[32] A. Madhavapeddy et al., "Unikernels: Library operating systems for the cloud," in *Proc. 18th Int. Conf. Architectural support Program. Lang. operating Syst.*, New York, NY, USA, Mar. 2013, pp. 461–472, doi: 10.1145/2451116.2451167.

[33] A. K. M. F. Mehrab, R. Nikolaev, and B. Ravindran, "Kite: Lightweight critical service domains," in *Proc. 17th Eur. Conf. Comput. Syst.*, New York, NY, USA, Mar. 2022, pp. 384–401, doi: 10.1145/3492321.3519586.

[34] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library OS from the top down," in *Proc. 16th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2011, pp. 291–304, doi: 10.1145/1950365.1950399.

[35] A. Raza et al., "Unikernel Linux (UKL)," in *Proc. 18th Eur. Conf. Comput. Syst.*, New York, NY, USA, May 2023, pp. 590–605, doi: 10.1145/3552326.3587458.

[36] V. A. Sartakov, L. Vilanova, and P. Pietzuch, "CubicleOS: A library OS with software componentisation for practical isolation," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, Apr. 2021, pp. 546–558, doi: 10.1145/3445814.3446731.

[37] D. Schatzberg, J. Cadden, H. Dong, O. Krieger, and J. Appavoo, "EbbRT: A framework for building per-application library operating systems," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI 16)*, Nov. 2016, pp. 671–688. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/schatzberg

[38] (2014). *Instruction Sets Should Be Free: The Case for RISC-V*. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.pdf

[39] (2023). *RISC-V Recently Ratified Extensions*. [Online]. Available: https://wiki.riscv.org/display/HOME/Recently+Ratified+Extensions

[40] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *Proc. 27th USENIX Conf. Secur. Symp.*, 2018, pp. 955–972. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/gras

[41] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, "A security RISC: Microarchitectural attacks on hardware RISC-V CPUs," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 2321–2338.

[42] (2019). *Linaro Limited: Open Portable Trusted Execution Environment*. [Online]. Available: https://www.trustedfirmware.org/projects/op-tee/

[43] (2021). *Qualcomm TEE*. [Online]. Available: https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp4078.pdf

[44] (2019). *Trustonic TEE*. [Online]. Available: https://www.trustonic.com/technical-articles/what-is-a-trusted-execution-environment-tee/

[45] (2023). *Huawei TEE*. [Online]. Available: https://consumer.huawei.com/au/sustainability/privacy/

[46] Y. Shen et al., "Occlum: Secure and efficient multitasking inside a single enclave of Intel SGX," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, 2020, pp. 955–970, doi: 10.1145/3373376.3378469.

[47] (2023). *RISC-V LOMMU*. [Online]. Available: https://github.com/riscv-non-isa/riscv-iommu
[48] (2023). *RISC-V LOPMP*. [Online]. Available: https://github.com/riscv-non-isa/iopmp-spec
[49] (2019). *OpenSBI*. [Online]. Available: https://github.com/riscv-software-src/opensbi
[50] (2021). *Allwinner D1 Nezha Board*. [Online]. Available: https://d1.docs.aw-ol.com/en/d1_dev/
[51] (2023). *Musl Libc*. [Online]. Available: https://musl.libc.org/
[52] J. Lee et al., "Hacking in darkness: Return-oriented programming against secure enclaves," in *Proc. 26th USENIX Secur. Symp. (USENIX Secur. 17)*, Aug. 2017, pp. 523–539. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk
[53] (2019). *RV8 Benchmark Suite*. [Online]. Available: https://github.com/michaeljclark/rv8-bench
[54] (2023). *Stress-NG*. [Online]. Available: https://github.com/ColinIanKing/stress-ng
[55] K. Wang et al., "XiangShan: An open-source project for high-performance RISC-V processors meeting industrial-grade standards," in *Proc. IEEE Hot Chips 36 Symp. (HCS)*, Aug. 2024, pp. 1–25.
[56] (2023). *Newlib*. [Online]. Available: https://sourceware.org/newlib/
[57] J. Redmon. (2013). *Darknet: Open Source Neural Networks in C*. [Online]. Available: http://pjreddie.com/darknet/
[58] (2022). *Sqlite3*. [Online]. Available: https://www.sqlite.org/
[59] N. Kossifidis et al. (2021). *Smepmp: Pmp Enhancements for Memory Access and Execution Prevention on Machine Mode*. [Online]. Available: https://github.com/riscv/riscv-tee/blob/main/Smepmp/Smepmp.pdf
[60] (2023). *Sspmp: RISC-V S-Mode Physical Memory Protection*. [Online]. Available: https://github.com/riscv/riscv-spmp/blob/main/rv-spmp-spec.pdf
[61] R. Bahmani et al., "CURE: A security architecture with customizable and resilient enclaves," in *Proc. 30th USENIX Secur. Symp.*, Sep. 2021, pp. 1073–1090. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani
[62] Y. Jia et al., "HyperEnclave: An open and cross-platform trusted execution environment," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 437–454.
[63] (2008). *Security Technology: Building a Secure System Using TrustZone Technology*. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29- GENC009492C_trustzone_security_whitepaper.pdf
[64] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "vTZ: Virtualizing armtrustzone," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 541–556.
[65] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "TwinVisor: hardware-isolated confidential virtual machines for ARM," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, Oct. 2021, pp. 638–654, doi: 10.1145/3477132.3483554.
[66] W. Li, Y. Xia, L. Lu, H. Chen, and B. Zang, "TEEv: Virtualizing trusted execution environments on mobile platforms," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, Apr. 2019, pp. 2–16.
[67] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Sanctuary: Arming TrustZone with user-space enclaves," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Feb. 2019, pp. 1–15. [Online]. Available: https://www.ndsssymposium.org/wp-content/uploads/2019/02/ndss2019_01A-1_Brasser_paper.pdf

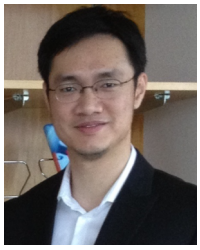
Mingde Ren received the bachelor's degree in physics from the Southern University of Science and Technology (SUSTech). He is currently pursuing the joint Ph.D. degree with SUSTech and The Hong Kong University. His research interests include trusted execution environment, confidential computing, and applied cryptography.

Jiatong Chen received the bachelor's degree in communication engineering from Jilin University. He is currently pursuing the master's degree with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech). His research interests include system security and confidential computing, with a focus on trusted execution environment and hardware-assisted security.

Ziquan Wang (Student Member, IEEE) is currently pursuing the master's degree with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech). His research interests include system security and heterogeneous computing.

Fengwei Zhang (Senior Member, IEEE) is an Associate Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech). Before joining SUSTech, he spent four years as an Assistant Professor with the Department of Computer Science, Wayne State University. His primary research interests are in the areas of systems security, with a focus on trustworthy execution, hardware-assisted security, debugging transparency, and plausible deniability encryption.

Zhenyu Ning received the Ph.D. degree from the Department of Computer Science, Wayne State University, in 2020. He is currently an Associate Professor with the College of Computer Science and Electronic Engineering, Hunan University. His research interests are in different areas of security and privacy, including system security, mobile security, IoT security, transportation security, trusted execution environment, and hardware-assisted security.



Heming Cui (Member, IEEE) is an Associate Professor with the Department of Computer Science, The University of Hong Kong (HKU). His research interests include operating systems, programming languages, distributed systems, and cloud computing, with a particular focus on building software infrastructures and tools to improve reliability and security of real-world software.