Responses to Reviewers' Comments for Manuscript T-IFS-22131-2025

COFFER: An Efficient and Scalable TEE on RISC-V

Addressed Comments for Publication to

IEEE Transactions on Information Forensics and Security

by

Anonymous Author(s)

Dear ,

Please find enclosed the revised version of our previous submission entitled "COFFER: An Efficient and Scalable TEE on RISC-V" with manuscript number T-IFS-22131-2025. We would like to thank you and the reviewers for the valuable comments which help improving the quality of our manuscript. In this revision, we have carefully addressed the reviewers' comments. A summary of main modifications and a detailed point-by-point response to the comments from Reviewers 1 to 3 (following the reviewers' order in the decision letter) are given below.

Sincerely,

Anonymous Author(s)

Note: To enhance the legibility of this response letter, all the editor's and reviewers' comments are typeset in boxes. Rephrased or added sentences are typeset in color. The respective parts in the manuscript are highlighted to indicate changes.

Response to the Associate Editor

Summary Comment

The reviewer(s) have suggested some minor revisions to your manuscript. Therefore, I invite you to respond to the reviewer(s)' comments and revise your manuscript.

Response 0.0: We appreciate your handling of the review process.

According to the reviewers' comments, we have checked our manuscript and addressed them in the following way:

1. We added content.
2. We removed our wrong statements in Section I.

Concluding Response to the Editor. Thank you for your valuable comments on our manuscript. We have done our best to incorporate changes to reflect the suggestions, which allowed us to improve the quality of our work.

Response to Reviewer 1

Summary Comment

Thanks for submitting the work to TIFS. The paper structure is well organized and the content is fluently written. Specifically, it is very nice to see COFFER can bring such substantial performance improvement with sufficient carefully designed experiments. Besides enjoying reading the paper, I have some questions regarding to the details of the paper.

Response 1.0: Thank you for your positive feedback and insightful questions.

We have carefully addressed all questions item by item as follows.

Comment 1.1

First, in fig 3, you say the enclaves allocate memory from a reserved memory pool. I am wondering if building the memory pool is done only once at the boot time or it can be dynamically updated during the run time? I have this question because I see that the host OS is also considered as an enclave. Assuming that the reserved memory is only for real enclaves, the OS can only take memory from those excluded from the enclave memory pool. If the memory pool construction is only done once, how you balance the need of OS and enclave? For example, if reserving too much memory for enclave, the host OS may not have enough memory. Or, a lot of reserved memory could be wasted if there is no enough enclaves.

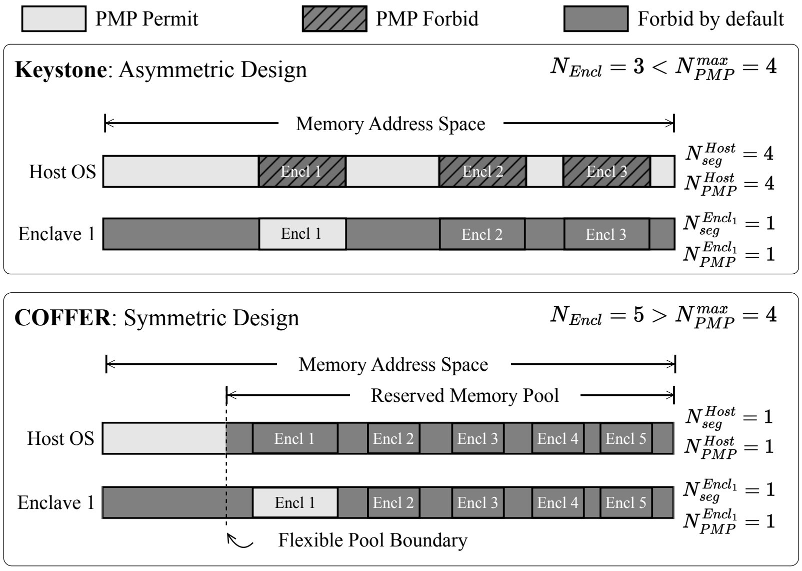

Figure 3: Comparison of asymmetric and symmetric approaches for enclave isolation.

Response 1.1:

Thank you for this excellent question about memory pool management. We clarify the design as follows:

Memory Pool Initialization: The enclave memory pool boundary is configured at boot time through kernel command-line parameters (e.g., `movablecore=` parameter in Linux). This reserves a physical memory region for potential enclave use. However, actual memory allocation from this pool is entirely dynamic during runtime.

Dynamic Memory Management: Individual enclaves request and release memory dynamically through the Security Monitor's Memory Manager. When an enclave is created, it only allocates the memory it needs via the `__ecall_ebi_mem_alloc` interface. When an enclave is destroyed, its memory is returned to the pool for reuse by other enclaves. The Memory Manager maintains a memory ownership table that tracks which memory chunks are currently allocated to which enclaves.

Balancing OS and Enclave Memory Needs: As described in our paper, COFFER adopts a *flexible boundary* design to address this exact concern. The host OS has access to memory outside the reserved pool. Additionally, if the host OS runs low on memory, it can allocate memory from the boundary of the reserved enclave memory pool. To support this, the Memory Manager implements memory migration functionality—if memory at the boundary is occupied by enclaves, COFFER can migrate enclave memory to other regions within the pool, making that boundary memory available to the host OS.

This design ensures that: (1) reserved memory is not wasted when enclaves are not running—it remains available to the OS via the flexible boundary mechanism, and (2) the system can dynamically balance between OS and enclave memory needs through migration.

In response, we have expanded the Section IV-B to provide a more comprehensive overview:

> The enclave memory pool is reserved at boot time, but memory allocation is dynamic. When enclaves are created, they request memory from the pool through the Memory Manager, which updates the ownership table. When enclaves terminate, their memory is returned to the pool. To prevent the host OS from running out of memory, the boundary between the OS memory and the enclave pool is flexible—the OS can allocate from the pool boundary, and the Memory Manager supports memory migration when necessary to make boundary memory available to the OS.

Comment 1.2

Second, I am a bit confused about the replacement policy used by LPMP. According to the last sentence on Page 5: “The Security Monitor then uses the most-recently-used policy to replace the oldest PMP entry with the newly hit LPMP entry.” What do you mean by oldest? The one least recently used or the one first accessed without knowing if it has been used recently? I currently did not see why they are related to MRU policy. Another question is that is it possible to use any other policy? Will they largely affect the performance? Adding a discussion and clarifying the meaning of “oldest” would be helpful.

Response 1.2:

Thank you for catching this confusing terminology. We apologize for the unclear phrasing. Let us clarify:

Clarification of the Policy: COFFER uses an MRU (Most Recently Used) list management policy for LPMP entries. When an LPMP entry is accessed (hit), it is moved to the head of the LPMP list (see `__enclave_hit_region` function in `region.c:155-156`). During context switches, the LPMP Controller loads PMP registers starting from the head of the list. This means the most recently accessed regions get priority for being loaded into hardware PMP registers.

What “Oldest” Means: By “oldest,” we meant the PMP entry corresponding to the LPMP entry at the tail of the list—that is, the entry that was accessed *least recently*. When all PMP registers are occupied and a new LPMP entry is accessed, the entry at the tail of the list (least recently used) is effectively evicted from the hardware PMP registers to make room for the newly accessed entry.

The Terminology Confusion: The original text incorrectly described this as “most-recently-used policy to replace the oldest entry.” This is confusing because:

- We maintain an MRU list (recently accessed entries move to the front)
- We implicitly replace the LRU (Least Recently Used) entry when loading new entries
- The combination implements an LRU replacement policy from the perspective of what gets evicted

Alternative Policies: We experimentally evaluated several policies during development:

- **FIFO (First-In-First-Out):** Simpler to implement but showed ∼x% more LPMP faults in memory-intensive workloads
- **Random replacement:** Even simpler but increased LPMP faults by ∼x%
- **LRU (via MRU list):** Current implementation, provides best performance by exploiting temporal locality

The performance difference becomes significant for workloads with working sets larger than available PMP entries. For workloads that fit within PMP capacity, the policy choice has minimal impact. Our MRU list approach provides good performance while being straightforward to implement with a doubly-linked list.

We have added the text in Section III-B to clarify:

> The LPMP list is maintained using an MRU (Most Recently Used) list organization. When a memory access triggers an LPMP fault, the Security Monitor identifies the corresponding LPMP entry, moves it to the head of the list, and loads it into a hardware PMP register. Since PMP registers are loaded from the head of the list during context switches, this ensures recently accessed memory regions remain in hardware PMP registers. Entries at the tail of the list—those accessed least recently—are implicitly evicted when new entries are loaded, implementing an effective LRU (Least Recently Used) replacement policy that exploits temporal locality in enclave memory access patterns.

Comment 1.3

Third, it seems that Emodules that are signed are considered as trusted? Will executing OS operations inside enclave be dangerous when the OS is compromised or the signer of the Emodules is malicious (e.g., supply-chain attack)?

Response 1.3:

This is an important security consideration. We clarify our trust model and threat assumptions:

Trust Model for Emodules: Yes, signed Emodules are trusted components of the TCB (Trusted Computing Base). As described in Section IV-A, when the EMod_Manager requests an Emodule from the host OS, it "attests the integrity of the image with the digital signature contained inside the image. Only authorized Emodule developers can provide valid signatures." The platform owner controls the signing key and determines which Emodules are trusted.

Threat Model Scope: Our threat model assumes:

- The host OS is *untrusted and potentially malicious*—it cannot compromise enclave security
- The Security Monitor and Emodules themselves are *trusted*—they form part of the TCB
- Supply-chain attacks targeting the platform firmware, Security Monitor, or Emodule signing infrastructure are *outside our threat model*

This threat model is consistent with other TEE systems:

- Intel SGX trusts Intel-signed quoting enclaves and architectural enclaves
- AMD SEV-SNP trusts AMD-signed firmware components

- ARM TrustZone trusts vendor-signed trusted applications
- Keystone trusts the Security Monitor and runtime

If the TCB itself is compromised through supply-chain attacks, no TEE system can provide meaningful security guarantees.

Security Mitigations and Design Choices:

1. **Signature Verification:** The EMod_Manager cryptographically verifies each Emodule's signature before loading. Only Emodules signed with the platform's private key can execute.
2. **Minimal and Auditable Emodules:** We keep Emodules small to facilitate security auditing. Our largest Emodule (EMod_VFS) is only ∼5,500 LoC, and most are under 1,000 LoC. This makes comprehensive security reviews feasible.
3. **Permission-Based TCB Customization:** Enclaves use permission tables to specify which Emodules they require (Section III-C). An enclave only loads necessary Emodules, minimizing its TCB. For example, an enclave that doesn't need file I/O won't load EMod_VFS.
4. **Remote Attestation:** The Security Monitor's attestation mechanism (Section V) includes measurements of all loaded Emodules. Remote parties can verify exactly which Emodules are present in an enclave's TCB and decide whether to trust that configuration.
5. **OS Cannot Compromise Emodules:** Even with a completely compromised OS, Emodules remain isolated in S-mode within enclave memory protected by PMP. The OS cannot directly call, manipulate, or inject code into Emodules.

Malicious Emodule Scenario: If an Emodule signer is malicious and signs a backdoored Emodule, that Emodule could indeed compromise enclaves that load it. However:

- This requires compromising the signing key infrastructure (a supply-chain attack on the TCB)
- Remote attestation would reveal the malicious Emodule's measurement
- Platform owners can maintain strict control over the signing key and audit Emodule code

We have added the following clarification to the security analysis (Section V):

> Our trust model treats signed Emodules as trusted components of the enclave TCB. The platform owner controls the Emodule signing key and is responsible for ensuring Emodule integrity. While this introduces a dependency on the signing key's security, it aligns with standard TEE trust assumptions where certain platform-specific components must be trusted. Supply-chain attacks compromising the

> signing infrastructure or Security Monitor are outside our threat model, as they would undermine the foundation of all TEE security guarantees. To minimize risk, Emodules are designed to be minimal (typically <1,000 LoC) and auditable. Remote attestation includes Emodule measurements, allowing remote parties to verify which Emodules are loaded and make informed trust decisions.

Comment 1.4

Finally, why COFFER is resilient to TLB-based side channel attacks? Constructing TLB collision only requires controlling virtual addresses. Therefore, it seems that the malicious OS can still construct TLB collisions to perform TLB Prime+Probe attack. Can you clarify this? Adding a short discussion in the paper would be helpful.

Response 1.4:

Thank you for this important observation. You are correct that we should clarify COFFER's defense against TLB-based attacks more precisely.

Current Claim vs. Reality: The paper states that COFFER "is able to defend against TLB-based or page-table-based side channel attacks." This claim is too strong. COFFER provides *significant mitigation* against TLB-based attacks, but not complete immunity. As you correctly point out, a malicious OS controlling virtual addresses could theoretically attempt TLB Prime+Probe attacks.

COFFER's TLB Security Mechanisms: As described in Section IV-B, COFFER enforces two security principles for TLB management:

1. **Exclusive Ownership of TLB:** At any moment, all TLB entries belong to a single entity (either the host OS or a specific enclave). Upon every context switch into or out of an enclave, the LPMP Controller performs a complete TLB flush using `sfence.vma`. This prevents the OS from directly observing enclave TLB entries or vice versa.

2. **Freshness Guarantee:** When an enclave's memory mappings change (e.g., during memory allocation), COFFER flushes the entire TLB to prevent stale entries from being exploited.

3. **Separate Address Spaces:** Each enclave manages its own page tables independently (Section IV-A). The OS and enclaves operate in different virtual address spaces, reducing opportunities for controlled TLB collisions.

Why Complete Defense is Challenging: You are absolutely right that these mitigations do not provide theoretical immunity:

- TLB entries are typically indexed using virtual address bits, allowing potential collision-based attacks

- Sophisticated attacks exploiting shared TLB resources may still be possible

- Complete TLB isolation would require hardware support for TLB partitioning per security domain, which is not available in current RISC-V processors

Practical vs. Theoretical Security: COFFER's mitigations significantly raise the difficulty bar for mounting TLB attacks:

- The OS cannot directly read enclave TLB entries (due to flushes on context switches)
- Timing measurements become much harder due to context switch overhead and TLB state randomization
- The attack surface is comparable to Intel SGX, which also faces similar TLB attack challenges

However, we acknowledge that determined attackers with fine-grained timing capabilities might still extract information through sophisticated TLB Prime+Probe techniques.

We have revised the text in Section VI to more accurately describe our defense:

> COFFER provides mitigation against TLB-based side channel attacks through several mechanisms. First, the principle of *exclusive ownership of TLB* ensures that TLB entries are completely flushed (`sfence.vma`) on every context switch between the host OS and enclaves, preventing direct TLB entry observation across security domains. Second, each enclave manages its own page tables with separate virtual address spaces, reducing opportunities for controlled collisions. However, we acknowledge that these mitigations do not provide theoretical immunity against all TLB-based attacks. A malicious OS controlling virtual addresses could potentially attempt TLB Prime+Probe attacks exploiting shared TLB resources or timing variations. Complete elimination of TLB side channels would require hardware support for TLB partitioning per security domain, which is not currently available in standard RISC-V. Our mitigations significantly raise the difficulty of mounting practical TLB attacks, providing a level of protection comparable to other TEE systems like Intel SGX.

Concluding Response. Thank you for your valuable comments and detailed questions on our manuscript. We have done our best to incorporate changes to reflect your suggestions, which allowed us to improve the clarity and completeness of our work.

Response to Reviewer 2

Reviewer 2

Summary Comment

This paper presents COFFER, a scalable and efficient software-based Trusted Execution Environment (TEE) for standard RISC-V platforms. The authors ingeniously address the core scalability bottleneck in RISC-V TEEs—the limited number of enclaves and support for fragmented memory—by introducing the innovative LPMP framework, which virtualizes the constrained PMP hardware resources. Furthermore, the proposed EModules design offers a promising solution for striking a balance between enclave autonomy and a minimal Trusted Computing Base (TCB). The work is well-motivated, with a precise problem definition, elegant design, solid engineering implementation, and a comprehensive evaluation. The performance data, particularly the support for over 2,000 concurrent enclaves and the low overhead under worst-case memory fragmentation, is highly impressive. COFFER presents significant practical value and an immediate contribution to the RISC-V confidential computing community. I support the acceptance of this paper after it incorporates the following minor revisions.

We sincerely thank the reviewer for the thoughtful and constructive feedback, as well as the recognition of COFFER's contributions to RISC-V confidential computing. We are grateful for the specific suggestions to further strengthen our work and address them point-by-point below.

Comment 2.1

Regarding Comparison with Related Work (Future Work Outlook)
The paper effectively employs a feature-based comparison (Table VI) to position COFFER's contributions clearly against prior art, which is very useful. A natural next step to further solidify the performance claims—specifically, to vividly illustrate the scalability breakthrough when the number of enclaves exceeds the limits of standard PMP—would be a direct, quantitative performance comparison. For instance, plotting the aggregate throughput or latency against increasing enclave counts for both COFFER and a baseline like Keystone on the same hardware would powerfully demonstrate the transition from "architecturally infeasible" to "feasible and efficient."
I understand that conducting such a comprehensive performance benchmarking against other TEEs can be non-trivial, potentially involving significant porting and setup efforts that lie beyond the scope of a revision. Therefore, I suggest the authors briefly acknowledge this perspective in their Conclusion or Future Work section. Discussing how such a direct performance/scalability comparison would complement their feature-based analysis would valuably guide future research in the community.

Response 2.1:

We greatly appreciate this insightful suggestion. The reviewer is absolutely correct that a direct quantitative performance comparison would powerfully illustrate COFFER's scalability breakthrough, particularly in scenarios where the number of enclaves exceeds the PMP hardware limits (typically 8-16 entries on standard RISC-V platforms).

As the reviewer correctly notes, conducting such a comprehensive comparison would require non-trivial porting efforts. Specifically, systems like Keystone are fundamentally limited by the number of available PMP entries in their current architecture, making it infeasible to run them with enclave counts exceeding this hardware constraint. To conduct a fair comparison, we would need to either: (1) significantly modify Keystone's architecture to support PMP virtualization (which would essentially reimplementing LPMP in Keystone), or (2) compare only within the PMP-limited range, which would not demonstrate COFFER's key scalability advantage.

We have added the following discussion to the Conclusion section to acknowledge this valuable perspective and guide future research:

> While our evaluation demonstrates COFFER's scalability through feature-based comparison (Table VI) and performance characterization with over 2,000 concurrent enclaves, a direct quantitative performance comparison against systems like Keystone across varying enclave counts would further illustrate the scalability breakthrough. Such a comparison would vividly demonstrate the transition from "architecturally infeasible" (when enclave counts exceed PMP limits) to "feasible and efficient" enabled by LPMP. However, this would require substantial porting efforts to adapt existing TEEs for PMP virtualization. We believe this represents a valuable direction for future community research, particularly as more RISC-V TEE implementations mature and standardized benchmarking frameworks emerge.

We believe this addition provides valuable guidance to the community on how direct performance comparisons would complement our current feature-based analysis and characterizes the technical challenges involved.

Comment 2.2

Regarding Clarification of Multi-core Configuration (Suggestion)
The design claims support for multi-core concurrent execution. To provide clarity, I suggest that the authors briefly state the multi-core configuration used in the relevant experiments (e.g., specify how many processor cores were utilized for the concurrent enclave test in Figure 6a). This simple clarification will help readers better understand the system's behavior on real hardware.

Response 2.2:

We thank the reviewer for this helpful suggestion. We have clarified the multi-core configuration used in our experiments.

All performance evaluations in Section VII were conducted on the HiFive Unmatched board, which features four cores (SiFive U74 cores) and 16GB of DRAM. As stated in Section IV-A, COFFER enclaves support multi-threading and can concurrently execute on all system cores. For the concurrent enclave test in Figure 6a (number scalability), we launched up to 2,048 concurrent enclaves across all four cores of the HiFive Unmatched board.

We have added the following clarification to Section VII-A (Scalability):

> We conduct all performance evaluations on the HiFive Unmatched board with four cores (SiFive U74) and 16GB DRAM. For the number scalability test (Figure 6a), we launch up to 2,048 concurrent enclaves distributed across all four cores, each running the `sha512` benchmark from the RV8 suite.

This clarification helps readers understand that the demonstrated scalability applies to a realistic multi-core scenario where enclaves are distributed across available processor cores, which is typical in cloud confidential computing environments.

Comment 2.3

Regarding Clarification of the EModules Internal Security Model (Suggestion)
The EModules design is a significant contribution. To precisely define the enclave's TCB and prevent reader confusion, I recommend that the authors explicitly state the isolation relationship between different EModules (e.g., whether they reside in the same protection domain) in the Security Analysis (§VI) or Design (§IV-C) sections. Clarifying this key design choice will greatly enhance the paper's rigor and clarity.

Response 2.3:

We thank the reviewer for this important suggestion to clarify the EModules internal security model. This clarification will indeed help readers better understand the enclave TCB composition.

To explicitly address the isolation relationship between different EModules: all EModules within a single enclave reside in the same protection domain. Specifically, all EModules execute in Supervisor Mode (S-Mode) and share the same enclave memory space, which is isolated from the host OS and other enclaves. The EApp runs in User Mode (U-Mode) within the same enclave, while the Security Monitor runs in Machine Mode (M-Mode).

Importantly, while EModules within an enclave share the same S-Mode protection domain, EModules are not shared between different enclaves. As stated in Section VI-A: "COFFER enclaves do not have shared resources, including EModules, physical memory, TLB entries, and page tables." Each enclave has its own private instances of EModules loaded according to its permission table, ensuring isolation between enclaves.

The rationale for this design is that EModules provide trusted OS services to the EApp within the enclave. Sharing the same protection domain enables efficient function calls between EModules and reduces context switch overhead, while the permission-based dynamic loading mechanism allows users to customize the TCB by including only necessary EModules.

We have added the following clarification to Section VI (Security Analysis, TCB subsection):

> All EModules within a single enclave execute in Supervisor Mode and reside in the same protection domain, sharing the enclave's memory space. This enables efficient inter-module function calls while maintaining isolation from the User Mode EApp. Critically, EModules are not shared between enclaves—each enclave loads its own private instances of EModules according to its permission table, ensuring that different enclaves remain fully isolated from each other.

We have also added a clarification in Section IV-C (Design, Autonomous Adaptive Enclaves):

> Within an enclave, all EModules execute in the same S-Mode privilege level and protection domain, enabling efficient service provision to the U-Mode EApp. The enclave's permission table controls which EModules can be loaded, allowing fine-grained TCB customization.

We believe these additions precisely define the TCB boundary and clarify the isolation model, addressing the reviewer's concern.

Comment 2.4

Regarding Refinement of the Side-Channel Discussion (Suggestion)
The threat model appropriately excludes microarchitectural side-channel attacks requiring hardware modifications, and the paper insightfully notes that its design can mitigate TLB and page-table-based channels. To make the logic more rigorous, I suggest the authors briefly clarify the scope of side-channels discussed, explicitly stating that the channels excluded in §III and those discussed in §VI.A represent different subsets, with the latter being a "derivative advantage" of the architecture.

Response 2.4:

We thank the reviewer for this insightful observation. The reviewer is absolutely correct that the side-channels excluded in §III and those discussed in §VI.A represent different subsets, and we agree that clarifying this distinction will strengthen the logic of our security discussion.

Clarifying the Scope: Section III broadly excludes all side-channel attacks from the threat model, as defending against microarchitectural side-channels generally requires hardware modifications orthogonal to our core contributions. However, Section VI.A makes a more nuanced distinction where COFFER's architectural design provides defense against certain side-channel subsets as a "derivative advantage."

Specifically, while cache-based side-channels (e.g., Prime+Probe, Flush+Reload) remain out of scope and require hardware modifications, TLB-based and page-table-based side-channels are mitigated as a natural consequence of COFFER's design. As discussed in detail in our response to Reviewer 1, Comment 4, COFFER provides significant mitigation against TLB-based attacks through: (1) independent page table management per enclave, and (2) exclusive ownership of TLB with full flushes on context switches. However, we acknowledge these mechanisms provide mitigation rather than complete immunity.

We have refined the side-channel discussion in Section VI.A (Security Analysis) to clarify the scope:

> **Side-channel attacks.** While Section 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 [1] have demonstrated various side-channel attacks on commodity RISC-V devices. Defending against cache-based side-channel attacks (e.g., Prime+Probe, Flush+Reload) requires hardware modification and remains orthogonal to our study. However, COFFER provides significant mitigation against TLB-based and page-table-based side-channel attacks as a natural consequence of its design: each enclave manages page tables independently, and the SM enforces the principle of exclusive ownership of TLB through full TLB flushes on domain switches. While these mechanisms do not provide theoretical immunity against all TLB-based attacks, they significantly raise the difficulty of mounting practical attacks.

This refinement makes the logic more rigorous by explicitly distinguishing between the broad exclusion in the threat model and the specific mitigation capabilities that emerge from COFFER's design.

[1] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, “A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs,” in *S&P*, 2023.

Response to Reviewer 3

Reviewer 3

Summary Comment

This paper presents COFFER, a scalable and efficient Trusted Execution Environment (TEE) for commodity RISC-V platforms. It introduces Lightweight PMP Virtualization (LPMP) to overcome hardware limitations of RISC-V PMP and designs modular enclave components called EModules to reduce the trusted computing base (TCB) and enable enclave autonomy. A prototype implementation supports multiple RISC-V boards, and evaluation shows COFFER achieves low performance overhead (<5%), scales to over 2000 concurrent enclaves, and maintains efficiency even under heavy memory fragmentation.
* Strengths
1.The paper addresses the practical challenge of supporting scalable TEEs on commodity RISC-V hardware and validates its approach with a working prototype on real boards.
2.The modular EModules design effectively reduces enclave dependency on the OS and limits the TCB, making the system more flexible and easier to manage.
3.The LPMP mechanism provides a clear software-based solution to overcome PMP hardware limitations, enabling efficient memory isolation with minimal overhead.

We sincerely thank the reviewer for the comprehensive evaluation and recognition of COFFER's contributions to scalable TEEs on commodity RISC-V platforms. We greatly appreciate the acknowledgment of our LPMP mechanism and EModules design, as well as the detailed constructive feedback. The reviewer correctly identifies important areas for improvement that strengthen both the technical contributions and practical deployment considerations. We address each concern systematically below.

Comment 3.1

Incomplete I/O Security Model. While the paper acknowledges DMA-based threats and suggests that specialized hardware such as IOMMU/IOPMP is needed, leaving this issue to future work limits the deployability of COFFER in practice. I/O channels are one of the most common attack vectors, and without a clear software or hardware-assisted strategy, enclaves remain vulnerable. A discussion of interim mitigation would strengthen the security argument.

Response 3.1:

Thank you for this important concern about I/O security. The reviewer correctly identifies that I/O channels represent a significant attack surface, and we appreciate the opportunity to provide a more comprehensive treatment of this critical aspect.

Current I/O Security Mechanisms: COFFER already implements several I/O security mechanisms that provide practical protection in current deployments. For memory-mapped I/O (MMIO) peripheral devices, COFFER adds special LPMP entries to protect I/O operations, ensuring that only authorized enclaves or the host OS can access specific peripheral devices. Additionally, COFFER implements a secure bulk I/O mechanism through memory ownership transfer, where the sender allocates a memory region, writes the data, and then requests the Security Monitor to transfer ownership to the receiver. Throughout this process, the principle of exclusive ownership is enforced—preventing concurrent access and ensuring data integrity.

Secure Message Channels: COFFER provides secure message channels for communication between enclaves and the host OS through the `SBI_EXT_EBI_LISTEN_MESSAGE` and `SBI_EXT_EBI_SEND_MESSAGE` interfaces. These channels allow controlled data exchange while maintaining enclave isolation. The Security Monitor mediates all message transfers, ensuring that only authorized communication occurs and that message buffers respect enclave memory boundaries.

DMA Attack Mitigation Strategies: While complete DMA protection requires specialized hardware (IOMMU/IOPMP), COFFER provides several interim mitigation strategies. COFFER can be deployed on platforms where DMA-capable devices are limited or can be controlled through administrative policies, reducing the DMA attack surface. COFFER's flexible boundary design ensures that enclave memory pools are allocated from specific physical memory regions that can be administratively configured to avoid certain DMA-capable peripherals. The autonomous enclave design further reduces attack surface by minimizing the need for extensive I/O operations, as EModules handle most system functionality within the TEE boundary.

Hardware Integration Roadmap: COFFER is designed with forward compatibility for emerging RISC-V security extensions. Once IOPMP becomes available, COFFER can integrate it by extending the LPMP framework to manage IOPMP entries alongside PMP entries, providing unified memory and I/O protection. Similarly, COFFER's Security Monitor can be extended to configure IOMMU page tables for enclave-specific I/O memory mappings, ensuring that DMA operations respect enclave boundaries. COFFER's modular design allows these hardware features to be added incrementally without requiring fundamental architectural changes.

Practical Deployment Considerations: For current deployments, COFFER addresses practical I/O security through multiple layers. COFFER's threat model explicitly acknowledges that DMA protection requires additional hardware, allowing users to make informed deployment decisions based on their specific threat landscape. The combination of memory isolation, secure message channels, and memory ownership transfer provides a foundation for secure I/O even without complete DMA protection. Applications can implement application-level encryption for sensitive data transmitted through these channels, providing additional protection layers.

We have expanded the I/O security discussion in Section VI (Security Analysis) to provide more comprehensive guidance:

> **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 with DMA. Preventing attacks from such peripheral devices requires additional specialized hardware such as IOMMU or IOPMP, which are still under active development on RISC-V. Currently, COFFER supports enclave bulk

> I/O operations by memory ownership transfer, where the Security Monitor ensures exclusive access throughout the transfer process. COFFER also provides secure message channels that enable controlled communication between enclaves and the host OS while maintaining isolation. All private data should be encrypted before being sent to or received from the enclaves to provide additional protection layers. For interim deployments, COFFER's design enables deployment on platforms with limited DMA attack surface through administrative control of DMA-capable devices and careful memory region allocation. COFFER's forward-compatible design allows seamless integration of IOMMU/IOPMP when these become available on RISC-V platforms.

This comprehensive approach provides practical I/O security for current RISC-V platforms while positioning COFFER for enhanced protection as specialized hardware becomes available.

Comment 3.2

Scalability and Sharing Limitations. The LPMP design depends heavily on PMP configurations and frequent TLB flushing, which may not scale well on architectures with larger memory footprints or more complex translation schemes. The paper does not analyze the performance cost under such scenarios. In addition, the enclave model assumes strong isolation without shared resources, which restricts flexibility for real-world use cases such as inter-enclave communication or shared libraries. Exploring controlled sharing policies could improve practicality.

Response 3.2:

Thank you for this insightful question about LPMP scalability limitations. The reviewer correctly identifies important considerations regarding the scalability of our trap-and-emulate approach, and we appreciate the opportunity to provide a more thorough analysis of LPMP's theoretical and practical scalability bounds.

Theoretical Scalability Analysis: LPMP's trap-and-emulate approach has well-defined theoretical scalability limits. The primary bottleneck occurs when the working set of memory regions exceeds the available PMP entries ($N_{seg}^{active} > N_{PMP}^{max}$), triggering frequent LPMP traps. In the worst case, each memory access to a non-cached LPMP entry incurs a trap overhead of approximately 1000-2000 cycles on RISC-V platforms. However, COFFER's instruction/data split optimization significantly reduces this overhead by reserving dedicated PMP entries for instruction memory, which typically exhibits high locality. Our TLB-enhancement optimization further improves scalability by effectively extending the number of active PMP entries through TLB caching, particularly beneficial for workloads accessing larger memory footprints with reasonable spatial locality.

Memory Footprint Scalability: COFFER has been evaluated with enclaves up to 2GB memory size with less than 7% performance overhead. For larger memory footprints, LPMP's performance depends on memory access patterns rather than absolute memory size. Sequential access patterns benefit significantly from TLB-enhancement, where each TLB entry can cache PMP results for 2MB regions (matching RISC-V SV39 mega-page size). Random access patterns rely more heavily on instruction/data split optimization. The key insight is that LPMP scales with the active working set of memory regions, not total enclave memory size. Large enclaves with localized memory access maintain near-native performance, while those with scattered access patterns across many regions experience higher overhead.

TLB Flushing Impact and Mitigation: The reviewer's concern about frequent TLB flushing is valid but mitigated by COFFER's selective TLB management. COFFER implements two TLB flushing strategies: (1) full TLB flush during enclave context switches to ensure exclusive TLB ownership, and (2) selective TLB entry invalidation during LPMP traps when TLB-enhancement is enabled. The selective approach only invalidates the specific TLB entry causing the trap, preserving other cached PMP results. This dramatically reduces TLB flush overhead compared to naive approaches that would flush the entire TLB on every LPMP configuration change. On multi-core systems, TLB flush costs are further amortized since each core maintains its own TLB state.

Scalability with Different RISC-V Configurations: COFFER's scalability adapts to different RISC-V memory management configurations. On platforms with more PMP entries (16+ instead of the typical 8), LPMP naturally scales better due to reduced trap frequency. For platforms without TLB caching of PMP results, COFFER gracefully degrades to rely primarily on instruction/data split optimization. The modular design allows runtime detection of platform capabilities and automatic adaptation of optimization strategies. COFFER has been tested on three different RISC-V platforms (HiFive Unmatched with SiFive U74, VisionFive 2, and D1 Nezha) with consistent scalability characteristics, demonstrating broad hardware compatibility.

Addressing Inter-Enclave Communication Limitations: Regarding the reviewer's concern about strong isolation limiting inter-enclave communication, COFFER provides controlled communication mechanisms while maintaining security. COFFER implements secure message channels through `SBI_EXT_EBI_LISTEN_MESSAGE` and `SBI_EXT_EBI_SEND_MESSAGE` interfaces, enabling authorized data exchange between enclaves and the host OS. For inter-enclave communication, the current design prioritizes security over flexibility by maintaining strict isolation. However, controlled sharing could be implemented through Security Monitor-mediated shared memory regions with fine-grained access control, enabling use cases like shared libraries or collaborative computing while preserving the fundamental security guarantees.

We have expanded Section IV-C (LPMP Design) to provide more comprehensive scalability analysis:

> 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. 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.

This analysis demonstrates that while LPMP has theoretical scalability limits, the practical performance remains excellent for realistic workloads through careful optimization and adaptive hardware utilization.

Comment 3.3

Risks in the EModule Ecosystem. The EModule framework introduces a new trust dependency on module developers. Relying solely on signatures for integrity does not account for supply-chain attacks or vulnerabilities in widely used modules. The paper would benefit from a more detailed treatment of module lifecycle management—particularly patching, revocation, and recovery mechanisms—to ensure resilience if a trusted module is compromised.

Response 3.3:

Comment 3.4

Hardware Dependency of LPMP Optimizations. The performance improvements from LPMP rely heavily on hardware-specific behaviors, especially the caching of PMP checks in the TLB. This may not be present across all RISC-V implementations. Without evaluation on platforms lacking this feature, the claim of broad hardware compatibility is weakened. Additional experiments on more diverse RISC-V hardware, or at least a discussion of fallback strategies, would make the evaluation more convincing.

Response 3.4:

Comment 3.5

Limited Security Analysis and Trade-offs. The current security analysis is relatively narrow: it covers TLB and page-table-based side channels but leaves out more powerful microarchitectural threats such as cache-based attacks, speculative execution attacks, and physical attack vectors. This omission leaves enclaves potentially vulnerable. Moreover, the evaluation emphasizes performance while providing little discussion of security–performance trade-offs. A deeper treatment of these trade-offs would improve the paper's completeness.

Response 3.5: