ZION: A Practical Confidential Virtual Machine Architecture on Commodity RISC-V Processors

Jie Wang*, Juan Wang*†, Yinqian Zhang‡
*Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education,
School of Cyber Science and Engineering, Wuhan University, Wuhan, China
†Zhongguancun Laboratory
‡Southern University of Science and Technology
{iwangjye, jwang}@whu.edu.cn, yinqianz@acm.org

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11133121

***Abstract*—Trusted Execution Environments (TEEs) provide robust hardware-based isolation to mitigate data breaches and privacy risks. Confidential Virtual Machines (confidential VMs or CVMs) extend these capabilities by using VMs as their execution abstraction, offering superior compatibility over process-based TEEs like Intel SGX. The rising demand for Confidential VMs has spurred innovations from major chip manufacturers, such as AMD SEV, Intel TDX, and Arm CCA, and their integration into leading cloud platforms, including AWS, Azure, and Google Cloud. On the RISC-V platform, however, existing TEE architectures rely on process-level abstractions or custom hardware, leading to limited compatibility and scalability.**

This paper presents ZION, a confidential VM architecture for commodity RISC-V hardware that operates without custom extensions. ZION ensures security, flexibility, and efficiency through a short-path *CVM mode* and a secure vCPU mechanism for protecting and efficiently updating vCPU states, enhancing context-switching performance. It combines Physical Memory Protection (PMP) with paging for scalable memory isolation, employs a hierarchical memory structure for efficient management, and introduces a split-page-table-based mechanism for secure memory sharing with virtio devices. Evaluations show ZION achieves under 5% overhead in real-world applications, demonstrating its practicality.

***Index Terms*—Confidential VM, RISC-V, Trusted Execution Environment**

I. INTRODUCTION

Trusted Execution Environments (TEEs) provide robust hardware-based isolation, making them increasingly crucial in addressing the rising incidents of data breaches and privacy violations [1], [2]. Among TEE technologies, Confidential Virtual Machines (confidential VMs or CVMs) stand out by using VMs as their execution abstraction. This approach offers greater compatibility and practicality compared to process-based TEEs like Intel SGX [3].

Driven by these advantages, confidential VM technology has seen significant development and adoption. Leading chip manufacturers have introduced their own confidential VM architectures, such as AMD SEV [4], Intel TDX [5], and Arm CCA [6]. Similarly, major cloud providers, such as Amazon AWS [7], Microsoft Azure [8], and Google Cloud [9], have integrated confidential VMs to offer secure and privacy-focused services for tenants with confidential workloads.

Confidential VMs on RISC-V are also gaining increasing attention and adoption, driven by the expanding role of RISC-V across diverse computing domains. Initially, RISC-V established its presence in low-performance Internet of Things (IoT) devices, leveraging its open-source architecture and flexibility to support a wide range of applications. However, RISC-V [10] is now transcending its traditional focus, making notable progress in high-performance cloud computing environments that demand robust, secure, and efficient solutions. This evolution is fueling a growing need for confidential VM technology, as organizations prioritize data privacy and secure computation in these advanced settings.

However, practical confidential VM architectures are still lacking on the current RISC-V platform. Sanctum [11], Keystone [12], and Penglai [13], the prominent RISC-V TEE architectures, all use process-level abstractions, necessitating modifications to user applications to be compatible with their TEEs. In contrast, confidential VM architectures designed to run on the RISC-V platform face issues with hardware compatibility. For example, CURE [14] and VirTEE [15] offer VM enclaves capable of running confidential VMs via hardware extensions. However, their reliance on custom hardware introduces hardware compatibility issues—they cannot run on commodity RISC-V processors. Another initiative, the confidential VM extension (CoVE) [16] proposed by RISC-V TEE TaskGroup (AP-TEE TG) [17], is still under development [18]. Considering the long delay from design to actual adoption of hardware security primitives, the availability of CoVE in the near future remains uncertain.

In addition to hardware compatibility, the current confidential VM architecture of the RISC-V platform also has deficiencies in flexibility and scalability. In terms of flexibility, CURE [14] and VirTEE [15] implement region-based memory isolation, requiring the physical addresses of the memory isolation area to be continuous. This design struggles to meet the flexible memory isolation needs of confidential VM, making dynamic expansion difficult and causing significant memory fragmentation. In terms of scalability, due to the

Corresponding author: Juan Wang

This work was supported in part by the National Key R&D Program of China under Grant No. 2023YFB4503902, the Smart Grid-National Science and Technology Major Project (No. 2024ZD0803000), the Major Program (JD) of Hubei Province (No. 2023BAA027), and the Key R&D Projects in Hubei Province under Grant No. 2023BAB165.

resource limitations of the security primitives of custom hardware extensions, CURE and VirTEE can only support 13 concurrent VM enclaves, which is difficult to meet the large-scale concurrency requirements of cloud platforms. These challenges have led us to pose the following research question: *Can we design a secure, efficient, and flexible confidential VM architecture using the commodity RISC-V processors?*

In this paper, we present ZION, a confidential VM architecture built on the commodity hardware of the RISC-V platform. ZION introduces several innovative designs to ensure security, flexibility, and efficiency. It introduces a short-path *CVM mode* and combines a secure vCPU with a shared vCPU interaction mechanism, enabling secure and efficient protection and updates of vCPU states. This ensures both the security and performance of context switching between confidential VMs. For memory isolation, ZION combines the Physical Memory Protection (PMP) hardware isolation primitive with paging to create a flexible and scalable memory isolation mechanism. Additionally, ZION incorporates a hierarchical memory structure and allocation strategy, significantly improving memory management efficiency for confidential VMs. Furthermore, ZION introduces a secure and efficient memory sharing mechanism based on split page tables, enabling support for virtio devices of confidential VMs. Evaluations show that ZION incurs less than 5% performance overhead in most real-world applications.

In sum, this paper makes the following contributions:

- We design a practical confidential VM architecture atop commodity RISC-V processors. This design does not require any hardware security extensions, ensuring compatibility with existing RISC-V hardware.
- We propose a series of optimizations for confidential VMs, including isolation mode, vCPU state protection and updates, memory isolation and management, and memory sharing, which ensure the flexibility, security, and efficiency of the ZION architecture.
- We implement a prototype of ZION and conduct comprehensive performance evaluation, which show that the overhead introduced by ZION is under 5% for most real-world applications.

II. BACKGROUND

RISC-V is an open, patent-unencumbered reduced instruction set computing (RISC) architecture [19]. RISC-V processors can be customized, implemented, and utilized by any entity. In addition to its clean, modular design, RISC-V includes specific security features, such as:

- **Physical Memory Protection:** PMP serves as a physical memory isolation mechanism, allowing software in machine mode (M mode) to control the memory access permissions of software in lower privilege modes. Each hardware thread has a set of PMP entries that define the address, size, and access permissions for multiple memory regions.
- **Trap Delegation:** By default, all traps (including interrupts and exceptions) on the RISC-V platform are handled in M mode. To improve the efficiency of trap handling, RISC-V introduces a trap delegation mechanism, which allows traps to be directly handled in lower privilege modes.
- **Hypervisor Extension:** To enable virtualization, RISC-V introduces the hypervisor extension, which extends supervisor mode to hypervisor-extended supervisor mode (HS mode). This extension also adds virtual supervisor (VS) and virtual user (VU) modes to support VMs. Hypervisor extension introduces new control and status registers and instructions to facilitate the execution of VMs and assist the hypervisor in managing them. Additionally, RISC-V supports memory virtualization through two-stage address translation in its virtualization modes.

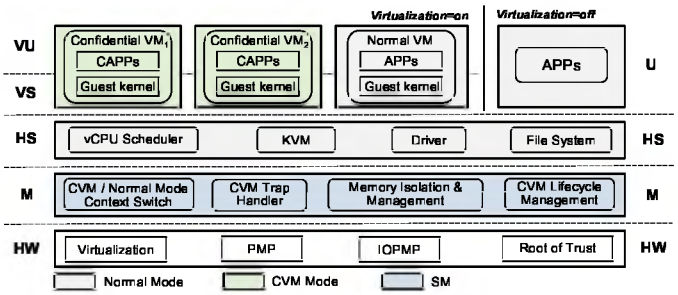

Fig. 1: The architecture of ZION.

III. OVERVIEW

A. System Architecture

The system architecture of ZION, as shown in Figure 1, introduces two new modes: *Normal mode* and *CVM mode*. Mode switching between the two is managed by the Secure Monitor (SM). *Normal mode* is responsible for running the non-trusted components, including the hypervisor, upper-layer applications, and normal VMs. The hypervisor operates in HS mode, which encompasses the scheduler, device drivers, file system, and the KVM. Meanwhile, QEMU runs in U mode, providing simulated device support for confidential VMs.

ZION is dedicated to running confidential VMs and operates in virtualized environments. The kernel of the confidential VM runs in VS mode, while its applications run in VU mode. The implementation and security of ZION are ensured by the SM, which operates at the highest privilege level (the M mode) and serves as the Trusted Computing Base (TCB) for the system. The SM uses hardware security primitives (e.g., PMP, IOPMP, trap delegation) to construct ZION, providing secure isolation and protection for the vCPUs and memory of confidential VMs. In addition, the SM handles memory management in ZION, including allocation, reclamation, and secure memory sharing. To facilitate collaboration between *Normal mode* and *CVM mode*, SM offers different Environment Call (ECALL) interfaces. These interfaces allow the hypervisor in *Normal mode* to control the lifecycle of confidential VMs, including initialization, operation, and suspension. Additionally, confidential VMs in ZION can use these ECALL interfaces to access functions such as retrieving measurement reports and platform random numbers.

B. Threat Model

ZION's threat model assumes the trustworthiness of underlying hardware mechanisms, including virtualization, trap delegation, PMP, and IOPMP. The security model assumes the SM is reliable and correctly implements the security mechanisms described in this paper. However, ZION does not trust privileged software beyond the SM and assumes the hypervisor could be fully compromised, including critical functions like vCPU scheduling and memory management. It is also assumed that the confidential VM does not intentionally leak sensitive data or code.

Physical attacks, such as bus snooping [20], cold boot attacks [21], and fault injection [22], [23], are beyond the scope of our research, as they require additional hardware support (e.g., a memory encryption engine) for mitigation. Similarly, cache side-channel attacks [24] and transient execution attacks [25], [26] fall outside our threat model, though existing defense mechanisms [27], [28] can be integrated into ZION's design. Like other TEE architectures, ZION does not address denial-of-service attacks.

IV. ZION DESIGN

A. Short-path CVM Mode

Most current confidential VM architectures, such as ARM CCA [6], TwinVisor [29], and CoVE [16], follow a similar model. These designs rely on the monitor to handle CPU security state transitions and introduce a thin secure hypervisor through horizontal isolation technologies, such as the TEE Security Manager (TSM) in CoVE. The secure hypervisor, which is much smaller than the normal hypervisor, provides essential security functions for confidential VMs. However, this approach increases the context switching path, resulting in performance degradation.

In its isolation mode, ZION adopts a design based on a virtual security processor, which divides the CPU into *Normal mode* and protected *CVM mode* using time-sharing multiplexing techniques. Unlike other designs, ZION centralizes execution state switching and security management in the SM, allowing the switch between *CVM mode* and *Normal mode* to require only a single privilege level switch. This approach reduces the overhead caused by multiple privilege level switches, as seen in traditional confidential VM architectures.

While ZION avoids introducing a secure hypervisor, this raises the risk that traps from confidential VMs could be captured by the normal hypervisor, potentially leading to security issues like leakage of sensitive register states. To mitigate this risk, ZION implements a trap delegation control mechanism that transfers trap handling rights to the SM. The delegation criteria are as follows: 1) Traps that can be processed by the confidential VM are delegated to VS mode and handled by the confidential VM kernel; 2) Traps that cannot be processed directly are sent to the SM for handling. This design improves context switching efficiency while maintaining the security of the confidential VM.

B. vCPU State Protection and Update

Unlike normal VMs, where the vCPU state is entirely controlled by the hypervisor, the vCPU state of confidential VMs must be protected from tampering or theft by untrusted hypervisors for security reasons. However, certain confidential VM functionalities, such as MMIO, still require assistance from the hypervisor. The challenge lies in ensuring the security of the confidential VM's vCPU state while facilitating efficient state transfer between the confidential VM and the hypervisor.

To address this, ZION introduces a secure and efficient vCPU state protection and update mechanism, combining secure and shared vCPUs. ZION allocates memory within the SM to create a secure vCPU structure. This structure stores the general-purpose and control register states of each vCPU of the confidential VM, ensuring the security of the vCPU state. To accelerate state transfer, ZION establishes a shared vCPU structure in the hypervisor, which is used for fast exchange of vCPU states between the hypervisor and the SM.

During the transition between *CVM mode* and *Normal mode*, the SM stores the register states in either secure vCPU or shared vCPU based on the type of exception or interrupt that causes the confidential VM to exit. For example, if a load instruction triggers a confidential VM exit, the SM stores the trap-related register states, such as *htinst*, in the shared vCPU to allow the hypervisor to parse the exception and retrieve the target register of the load instruction. Other register states are stored in the secure vCPU to maintain isolation and protection.

After the hypervisor handles the exception, it writes the required values into the shared vCPU registers. When the confidential VM is resumed, the SM not only restores the register states from the secure vCPU but also retrieves necessary values from the shared vCPU. To protect against TOCTOU (Time-of-Check to Time-of-Use) attacks [30], ZION adopts a Check-after-Load mechanism from TwinVisor, performing security checks after reading the shared vCPU register values to prevent tampering by the malicious hypervisor [29].

C. Flexible and Scalable Memory Isolation

A straightforward approach to memory isolation is to allocate contiguous physical memory directly to confidential VMs and utilize the PMP to isolate this memory region. While this method ensures memory isolation, it presents two key challenges: first, the limited number of PMP entries (typically 16) restricts the number of concurrent confidential VMs; second, this memory pre-allocation strategy requires resources to be allocated in advance based on the expected memory size when creating the confidential VM. This approach not only weakens the cloud platform's ability to overcommit memory, but also leads to memory fragmentation, while limiting the confidential VM's ability to dynamically expand memory.

To enhance the flexibility and scalability of memory isolation, ZION combines PMP with paging. This approach leverages PMP to enforce memory isolation between *CVM mode* and *Normal mode*. Initially, the SM configures PMP to create a secure memory pool within normal memory, designated for confidential VMs. Before transitioning to *CVM mode*, the SM

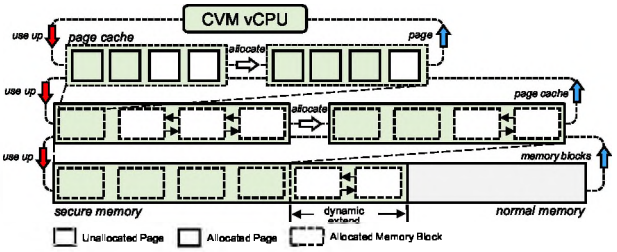

Fig. 2: Hierarchical memory allocation for confidential VMs.

updates the PMP configuration register (*pmpcfg*) to grant the confidential VM access to this secure memory. During hypervisor operation, the SM disables access to the secure memory pool, ensuring that untrusted hypervisors cannot modify or corrupt the protected memory.

For memory isolation between confidential VMs, ZION relies on Stage-2 page table isolation. During the creation and page fault handling of a confidential VM, the SM ensures that the memory allocated to the confidential VM is not shared with other confidential VMs. Only the SM has authority over the page table of the confidential VM, ensuring that the secure memory where the confidential VM's page table is stored is not mapped into the address space of other confidential VMs. As a result, even if a confidential VM is compromised, it cannot access or modify the memory of other confidential VMs due to the restrictions enforced by the page table.

To safeguard against potential Direct Memory Access (DMA) attacks from malicious peripherals, the secure memory area is also protected by the IOPMP [31], which controls peripheral access in addition to the CPU-level protection provided by PMP. Furthermore, to prevent untrusted hypervisors from launching page table-based attacks, such as side-channel or controlled-channel attacks [32], the SM configures page tables for confidential VMs within the secure memory pool. This prevents the hypervisor from tampering with or interfering with the confidential VM's page table.

D. Hierarchical Memory Management

To efficiently manage the secure memory pool of *CVM mode*, ZION introduces a hierarchical secure memory structure and employs targeted memory allocation strategies at various levels. When a privileged user registers contiguous physical memory with the SM, the SM divides the memory segment into smaller secure memory blocks (default size of 256KB) and links these blocks in a bidirectional circular linked list. Each secure memory block consists of several contiguous memory pages and contains pointers to the previous and next blocks. The blocks are ordered by address, with allocation prioritized from the head of the linked list.

Figure 2 illustrates the memory allocation process for *CVM mode*, which is divided into three stages. In the first stage, the SM attempts to allocate a memory page directly from the page cache associated with the vCPU of the confidential VM in which the page fault occurs. If free memory pages are available in the page cache, one is allocated directly; otherwise, it enters the second stage of allocation. In this stage, the SM allocates a new secure memory block from the head of the linked list and associates it with the vCPU experiencing the page fault. This new block serves as the page cache for that vCPU, ensuring each vCPU has an independent cache. In the third stage, when the secure memory pool nears depletion, the SM requests the hypervisor to expand the secure memory pool. The expansion strategy includes applying for a new contiguous physical memory area to be registered as a secure memory pool or expanding an existing memory pool.

This hierarchical memory allocation model significantly enhances memory management efficiency for confidential VMs. In the first stage, the availability of page cache allows most allocation requests to be fulfilled quickly, with the vCPU-specific cache supporting efficient memory allocation while avoiding the performance loss from concurrent locking. In the second stage, secure memory blocks are efficiently managed through the linked list, allowing allocation with a time complexity of $O(1)$. Since the first two stages are entirely controlled by the SM, they avoid the performance overhead associated with switching to *Normal mode*. Although the third phase requires the assistance of the hypervisor and introduces operations including memory expansion, mode switching, and TLB flushing, its impact on overall performance is minimal since most allocation requests can be handled in the earlier stages.

E. Split Page Table-Based Memory Sharing

To prevent the hypervisor from tampering with the confidential VM's page table, we assign page table management to the SM by default. However, this approach incurs performance overhead when supporting shared memory.

In the unoptimized shared memory establishment process, the hypervisor first allocates memory and creates a mapping, then synchronizes this mapping with the SM. After the SM verifies the request, it maps the confidential VM's address space to the physical memory, establishing the shared memory. However, this synchronization introduces performance overhead and consistency issues, particularly during page swap operations in *Normal mode*. If synchronization is delayed, it could threaten system availability and security. On the other hand, implementing a synchronization mechanism would increase system complexity and degrade performance.

To address these issues, we propose a shared memory scheme based on a split page table design. The Guest Physical Address (GPA) space of the confidential VM is divided into two regions: a private address space, secured by the SM for storing confidential code and data, and a shared address space, managed by the hypervisor for memory sharing. The shared address space maps directly to normal memory. The confidential VM's root page table includes entries for both secure and shared page tables, enabling access to memory from either. In contrast, the hypervisor's root page table contains only shared page table entries, restricting its access to normal memory and preventing modifications to the private memory mappings of the confidential VM. This design ensures that the hypervisor cannot use the shared page table it controls

to access the secure memory of the confidential VM, thereby maintaining memory security.

Since the hypervisor directly manages the shared page table, updates to shared memory can bypass the SM, eliminating the need for complex synchronization and reducing performance overhead. The update process affects only the shared address space, streamlining the operation and minimizing synchronization burdens. As a result, the split page table-based approach enhances both the efficiency and security of shared memory management while resolving related performance challenges.

V. Evaluation

A. Implementation & Experimental Setup

We integrated the SM with OpenSBI and made modifications to support our design. We also adapted the host's KVM module for compatibility and added the ECALL interfaces for calling SM. Additionally, we made minor adjustments to the confidential VM kernel to support shared memory requests.

To evaluate the performance of Zion, we conducted experiments on the Genesys2 FPGA development board, which simulated four 64-bit Rocket cores with hypervisor extensions, running at 100 MHz and equipped with 1 GB of memory. The host ran Linux (kernel version 5.19.16), and the virtualization stack used customized QEMU and KVM. The OS in the confidential VM matched that of the host. To support virtio devices, the confidential VM enabled SWIOTLB support and configured its buffer as shared memory.

B. CVM Mode Switching Performance

1) Optimization brought by shared vCPU mechanism: We evaluated the context switching time overhead of confidential VMs with and without shared vCPU state update support. The CVM entry and exit, triggered by MMIO exceptions, were tested 200 times. For CVM entry, the switching time without shared vCPU support was 5,293 cycles, while with shared vCPU support, it was reduced to 4,191 cycles, reflecting a performance improvement of 20.8%.

A similar improvement was observed in the CVM exit test. The switching time without shared vCPU support was 3,267 cycles, compared to 2,524 cycles with shared vCPU support, resulting in a performance improvement of 22.74%. These results demonstrate that Zion's shared vCPU state update mechanism significantly enhances context switching performance in *CVM mode*.

2) Optimization brought by short-path CVM mode: To assess the performance impact of Zion 's short-path design on *CVM mode* switching, we developed a simple secure hypervisor in the host system. This setup allowed us to compare the performance of long-path and short-path *CVM mode* context switching. The long-path *CVM mode* introduces an additional privilege layer conversion during context switching. At CVM entry, the execution flow jumps from the host to the SM, then to the secure hypervisor, and finally to the confidential VM. During CVM exit, the flow moves first from the *CVM mode* to the secure hypervisor, then from the hypervisor to the SM, and finally back to the host.

TABLE I: Execution cycles for RV8 Benchmarks in various environments. The baseline is indicated by *, and all values are in 10^9 cycles.

Benchmark	Normal VM*	Confidential VM (%)	
aes	6.312	6.498	(+2.95)
bigint	8.965	9.210	(+2.73)
dhrystone	4.144	4.264	(+2.90)
miniz	25.412	25.900	(+1.92)
norx	3.905	4.014	(+2.79)
primes	19.002	19.347	(+1.81)
qsort	2.148	2.205	(+2.65)
sha512	3.947	4.063	(+2.93)
Average	-	-	+2.59

We tested CVM entry and CVM exit triggered by a timer interrupt 200 times under both the long-path and Zion 's short-path context switching. Since the timer interrupt does not involve vCPU state updates, this test isolates the impact of context switching alone. The results show that Zion 's short-path design significantly improves context switch performance. For CVM entry, the long-path *CVM mode* switching time is 7,282 cycles, while Zion 's short-path mode takes only 4,028 cycles, resulting in a 44.7% performance improvement. For CVM exit, the long-path switching time is 5,384 cycles, whereas Zion 's short-path mode takes just 2,406 cycles, yielding a 55.3% improvement.

It is important to note that in our test, the secure hypervisor was not fully isolated. As such, this test only partially reflects the performance penalty of the long-path mode. Once security measures such as microarchitecture state clearing are added to the secure hypervisor, these operations will likely have a more significant impact on the overall performance of the confidential VM.

C. Stage-2 Page Fault Handling Performance

In this section, we evaluated the page fault processing performance of the normal VM and the confidential VM. After both VMs were started, we ran a program that allocated continuous physical memory and performed write operations. For the normal VM, we directly recorded the stage-2 page fault processing time in KVM. For the confidential VM, we recorded the page fault processing time at different memory allocation stages in the SM.

The test results show that the average page fault processing time for the normal VM is 39,607 cycles. For the confidential VM, the processing time for page faults triggering only the first-stage memory allocation is 31,103 cycles. When the second-stage memory allocation is triggered, the processing time increases slightly to 34,729 cycles. However, when the third-stage memory allocation is triggered, the page fault processing time increases significantly to 57,152 cycles.

Since most page fault requests are handled during the first two stages, the average page fault processing time for the confidential VM is 31,449 cycles, which is only slightly higher than the first-stage allocation time. These results align with our design analysis and demonstrate the efficiency of Zion 's hierarchical memory management.

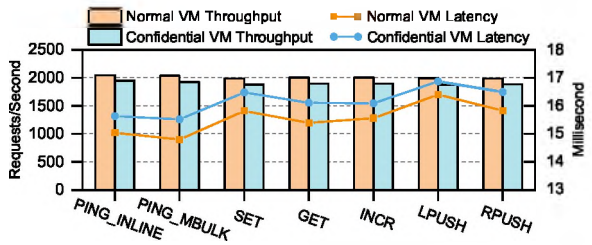

Fig. 3: Comparison of throughput and latency for various operations between normal VMs and confidential VMs.

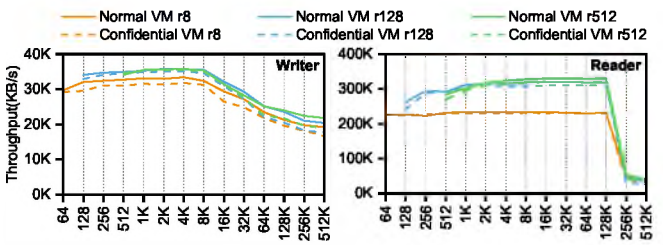

Fig. 4: Throughput performance for read and write operations at varying file sizes in normal and confidential VMs.

D. Macrobenchmarks

In this section, we evaluate confidential VM performance on real-world applications using CPU-intensive (RV8, Coremark), memory-intensive (Redis), and I/O-intensive (IOZone) workloads. Both normal and confidential VMs were configured with one vCPU, 256MB memory, and SWIOTLB enabled.

RV8 Benchmarks. The RV8 benchmarks include CPU-intensive programs such as encryption and decryption, mathematical operations, and sorting. We ran each program in the RV8 benchmark suite 20 times and calculated the average value for each. As shown in Table I, compared to normal VMs, the overheads of confidential VMs remained within 3%, with the average overhead measured at 2.59%.

CoreMark. CoreMark is another CPU-intensive benchmark. We ran it 20 times to obtain the average score. The test results indicate that the average score on the normal VM is 2,047.6, while the confidential VM scored 1,992.3, reflecting a 2.77% performance drop compared to the normal VM. This result aligns closely with the findings from our RV8 benchmark tests.

Redis Benchmark. The Redis benchmark tests the performance of the Redis in-memory database, which is itself a memory-intensive test. For this test, we conducted ten rounds of testing, executing 10,000 requests in each round. The test results are presented in Figure 3. The overhead of confidential VMs remains within a reasonable range when compared to normal VMs. In terms of throughput, confidential VMs show an average decrease of 5.3% compared to normal VMs. Regarding latency, confidential VMs experience an average increase of 4%.

IOZone. In this test, we compared continuous file read/write performance between a confidential VM and a normal VM across various file sizes (64KB–512MB) and record sizes (8KB, 128KB, 512KB). As shown in Figure 4, both write and read throughput are lower when the record size is small. For smaller files, the performance difference is minimal (under 5%), but as file sizes grow, the confidential VM's overhead increases, reaching up to 20% due to frequent I/O exits.

VI. Related Work

Sanctum [11], Keystone [12], Penglai [13], TIMBER-V [33], and SPEAR-V [34] are notable TEE implementations on RISC-V, all of which adopt the process model as their isolation abstraction. While effective, these process-based TEEs require significant porting efforts to support legacy applications, as they cannot run them directly. In contrast, ZION, a VM-based TEE, enables the execution of unmodified applications, offering greater practicality and ease of deployment.

Several VM-based TEEs also exist on the RISC-V platform, including CURE [14], VirTEE [15], and CoVE [16]. CURE supports multiple TEE abstractions, including user-space, kernel-space, sub-space, and VM enclaves, with the VM enclave enabling confidential VMs. Building on CURE, VirTEE introduces features such as live migration and secure I/O. However, both CoVE and VirTEE rely on hardware modifications, including enhancements to the CPU core and bus arbiter. These hardware dependencies limit their scalability, flexibility, and the number of parallel enclaves they can support. ZION, by comparison, operates without hardware modifications, offering greater scalability and flexibility, particularly in memory isolation and management. CoVE, developed by the AP-TEE TG [17], is a proposed confidential VM extension. It introduces a Memory Tracking Table (MMT) for controlling physical memory access and a lightweight secure hypervisor, the TSM, to isolate confidential VMs and provide essential security services. However, CoVE's specifications are still under development, and compatible hardware platforms are not expected to be widely available in the near term. In comparison, ZION leveraging RISC-V's mature hardware ecosystem, provides a practical alternative. It supports confidential VMs without requiring specialized hardware, making it a valuable solution even after CoVE's specifications and hardware become available.

On the Arm platform, works like Twinvisor [29] and VirtCCA [35] aim to support confidential VMs before CCA is available. These works share similar design goals with ZION. Compared to them, ZION expands on isolation mode design, memory isolation and management, and shared memory, achieving higher security, flexibility, and efficiency, in addition to differences in underlying architecture.

VII. Conclusion

ZION is a practical confidential VM architecture built on commodity RISC-V processors. In addition to ensuring hardware compatibility, ZION relies on innovative designs in isolation mode, vCPU protection and update, memory isolation and management, and secure memory sharing to ensure flexibility, security, and efficiency. Evaluations show that ZION's performance overhead is less than 5% in most real-world applications.

References

[1] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/Ispa*, vol. 1. IEEE, 2015, pp. 57–64.

[2] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: properties, applications, and challenges," *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

[3] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[4] A. Sev-Snp, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, vol. 53, pp. 1450–1465, 2020.

[5] I. Corporation, "Intel Trust Domain Extensions (intel TDX) whitepaper," 2024, accessed: 2024-11-20. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf

[6] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and verification of the arm confidential compute architecture," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 465–484.

[7] A. W. Services, "Sev-snp support on amazon ec2," 2024, accessed: 2024-11-20. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html

[8] Microsoft, "Confidential virtual machine options on azure," 2024, accessed: 2024-11-20. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-options

[9] G. Cloud, "Confidential vm overview," 2024, accessed: 2024-11-20. [Online]. Available: https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview

[10] R.-V. Foundation, "The RISC-V privileged architecture, version 1.10," 2017, accessed: 2024-11-20. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

[11] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874.

[12] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[13] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Scalable memory protection in the penglai enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 275–294.

[14] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "Cure: A security architecture with customizable and resilient enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1073–1090.

[15] J. Wang, P. Mahmoody, F. Brasser, P. Jauernig, A.-R. Sadeghi, D. Yu, D. Pan, and Y. Zhang, "Virtee: A full backward-compatible tee with native live migration and secure i/o," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 241–246.

[16] R. Sahita, V. Shanbhogue, A. Bresticker, A. Khare, A. Patra, S. Ortiz, D. Reid, and R. Kanwal, "Cove: Towards confidential computing on risc-v platforms," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023, pp. 315–321.

[17] RISC-V Tech AP TEE, "Risc-v technical advisory panel (ap) tee mailing list," 2024, accessed: 2024-11-20. [Online]. Available: https://lists.riscv.org/g/tech-ap-tee

[18] RISC-V, "Risc-v specification status," 2024, accessed: 2024-11-20. [Online]. Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154861/RISC-V+Specification+Status

[19] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *ACM SIGARCH Computer Architecture News*, vol. 8, no. 6, pp. 25–33, 1980.

[20] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[21] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[22] A. Cui and R. Housley, "Badfet: Defeating modern secure boot using second-order pulsed electromagnetic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[23] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svid voltage scaling interface," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 699–716.

[24] R. Guanciale, H. Nemati, C. Baumann, and M. Dam, "Cache storage channels: Alias-driven attacks and verified countermeasures," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 38–55.

[25] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[26] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[27] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 974–987.

[28] M. Orenbach, A. Baumann, and M. Silberstein, "Autarky: Closing controlled channels with self-paging enclaves," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[29] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "Twinvisor: Hardware-isolated confidential virtual machines for arm," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 638–654.

[30] J. Yang, A. Cui, S. Stolfo, and S. Sethumadhavan, "Concurrency attacks," in *4th USENIX Workshop on Hot Topics in Parallelism (HotPar 12)*, 2012.

[31] R.-V. Foundation, "Iopmp specification," 2024, accessed: 2024-11-20. [Online]. Available: https://github.com/riscv-non-isa/iopmp-spec

[32] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 640–656.

[33] 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 *Proceedings 2019-Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

[34] D. Schrammel, M. Waser, L. Lamster, M. Unterguggenberger, and S. Mangard, "Spear-v: Secure and practical enclave architecture for risc-v," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 457–468.

[35] X. Xu, W. Wang, Y. Wu, C. Wang, H. Zhu, H. Ma, Z. Min, Z. Pang, R. Hou, and Y. Jin, "virtcca: Virtualized arm confidential compute architecture with trustzone," *arXiv preprint arXiv:2306.11011*, 2023.