Software-defined SEV for Confidential VMs Based on the Delegation Architecture in RISC-V

Junpei Matsushita
Kyushu Institute of Technology
ma2shita@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@csn.kyutech.ac.jp

***Abstract*—Recently, the threat of cloud insiders is increasing as users handle sensitive information in virtual machines (VMs). To prevent cloud insiders from accessing sensitive information, clouds provide confidential VMs (CVMs), whose memory is protected. In RISC-V Confidential VM Extension (CoVE), a trusted software component guarantees correct address translation for CVMs. In contrast, AMD SEV-SNP delegates the management of the page tables to an untrusted hypervisor and verifies the correctness of address translation using an inverted page table. However, it is unclear whether this *delegation architecture* of SEV-SNP can be implemented in RISC-V without adding significant hardware components. This paper proposes *software-defined SEV (SD-SEV)*, which realizes the delegation architecture for CVMs using a small, trusted software module and a minimal hardware extension. In SD-SEV, the SEV module securely maintains the inverted page table to guarantee the correctness of address translation. Upon memory access, it checks whether the results of address translation match the table using a hardware extension called a *page success exception*. We have implemented SD-SEV on QEMU and confirmed the detection of attacks against memory integrity and authenticity as well as the performance overhead.**

I. Introduction

Recently, the threat of cloud insiders is increasing as users handle sensitive information in virtual machines (VMs) provided by cloud providers. To protect such sensitive information, several clouds, such as Amazon Web Services, Google Cloud, and Microsoft Azure, offer confidential VMs (CVMs). CVMs can prevent cloud insiders from eavesdropping on and tampering with sensitive data inside them. A CVM is a VM whose memory is protected using a trusted execution environment (TEE), which is provided by hardware. A TEE is isolated even from the hypervisor running CVMs. Examples of TEEs for CVMs are AMD SEV-SNP [1], Intel TDX [2], and Arm CCA [3].

RISC-V processors also provide Confidential VM Extension (CoVE) [4] to run CVMs. RISC-V is an open-source instruction set architecture that has recently gained attention in both academic and industrial domains. CoVE uses the TEE security manager (TSM) as a part of the software trusted computing base (TCB) and guarantees correct address translation for CVMs. In contrast, AMD SEV-SNP does not use such a trusted software component but delegates the management of the page tables to an untrusted hypervisor. Then, it checks the correctness of address translation on every memory access using the inverted page table. However, it is unclear whether such a delegation architecture can be implemented in RISC-V without adding large trusted hardware components.

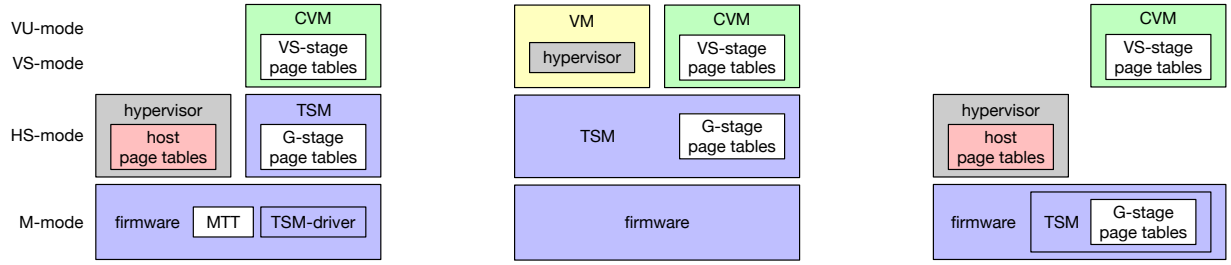
This paper proposes *software-defined SEV (SD-SEV)*, which realizes the delegation architecture of SEV-SNP using a small, trusted software module and a minimal hardware extension. In SD-SEV, the SEV module runs in the most privileged M-mode of RISC-V processors and securely maintains the inverted page table. To verify the correctness of address translation on memory access, SD-SEV uses a hardware extension called a *page success exception (#PS)* [5], which is raised when a processor succeeds in address translation after a TLB miss. In the exception handler, the SEV module checks whether the results of address translation match the inverted page table. In addition, the SEV module measures the boot image loaded by the hypervisor and guarantees the authenticity of software running in CVMs.

We have implemented SD-SEV using OpenSBI [6], Xvisor [7], and Linux on QEMU. According to our experiments, SD-SEV could detect various attacks against the memory, address translation, and the boot image of a CVM. It was shown that the boot time of a CVM became 5.5x longer than that of a non-confidential VM. The performance of a CVM was degraded by 5-90%. In addition, reducing #PS exceptions could improve performance in some workloads, but it was not effective in others.

The organization of this paper is as follows. Section II compares the system architecture of CoVE with that of SEV-SNP. Section III proposes software-defined SEV to enable the delegation architecture, and Section IV explains its implementation details. Section V analyzes the security of SD-SEV, and Section VI shows experimental results. Section VII describes related work, and Section VIII concludes this paper.

II. Confidential VMs in RISC-V

In RISC-V, Confidential VM Extension (CoVE) [4] has been proposed to support CVMs. CoVE achieves CVMs using the trusted TEE security manager (TSM) and, optionally, TEE hardware primitives. There are three deployment models in CoVE. The first model runs the TSM in parallel to an untrusted hypervisor in HS-mode (Fig. 1(a)). It divides physical memory into confidential memory assigned to CVMs and non-confidential memory using the memory tracking table (MTT) as additional hardware. Each CVM is isolated using the G-stage page tables, which are used by processors to translate



(a) Model 1 (b) Model 2 (c) Model 3

Fig. 1: The direct management architecture of CoVE.

a guest physical address (GPA) into a host physical address (HPA). Note that processors perform address translation from a guest virtual address (GVA) into a GPA using the VS-stage page tables.

The second model runs only the TSM in HS-mode and runs the hypervisor in VS-mode using nested virtualization (Fig. 1(b)). It divides physical memory into confidential and non-confidential using memory virtualization without the MTT. Like the first model, each CVM is isolated using the G-stage page tables. The third model runs the TSM in the most privileged M-mode and runs only the hypervisor in HS-mode (Fig. 1(c)). The TSM isolates physical memory using physical memory protection (PMP), which enables access control for physical memory regions. Like the other models, each CVM is isolated using the G-stage page tables. In any model, the hypervisor invokes the TSM or the TSM-driver running in M-mode when creating a CVM to convert non-confidential memory where the boot image is loaded into confidential memory. At the same time, the TSM measures the boot image to guarantee authenticity.

In CoVE, the correctness of two-stage address translation is guaranteed for CVMs because the two types of page tables for CVMs are protected. The VS-stage page tables are stored in the confidential memory assigned to each CVM. Therefore, they can be managed only by the guest OS in the CVM. They cannot be accessed by an untrusted hypervisor. Similarly, the G-stage page tables are stored in the confidential memory of the TSM. They can be managed only by the TSM and cannot be accessed by the hypervisor. Note that the host page tables are stored in the memory of an untrusted hypervisor. Although the hypervisor can configure the page tables so as to access confidential memory, such access is denied by the MTT, memory virtualization, or PMP. This *direct management architecture* is adopted for most of the TEEs for CVMs, such as Intel TDX and Arm CCA.

On the other hand, AMD SEV-SNP does not use such a trusted component like the TSM. Therefore, it could reduce the size of the software trusted computing base (TCB). As illustrated in Fig. 2, it can protect the guest page tables in each CVM, like the VS-stage page tables. However, it cannot protect the nested page tables (NPT), which correspond to the G-stage page tables. Since the NPT is stored in the memory of an untrusted hypervisor, it can be tampered with

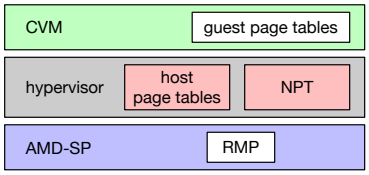


Fig. 2: The delegation architecture of SEV-SNP.

by the hypervisor. This means that the correctness of address translation is not guaranteed. For example, the hypervisor can mount memory remapping attacks [1], which change the assignment of physical memory to a CVM so that the CVM uses arbitrary code and data. It can also mount memory aliasing attacks [1], which map the same physical memory to multiple locations in a CVM so that the CVM stores sensitive data in unintentional locations.

To guarantee the correctness of two-stage address translation, SEV-SNP uses an inverted page table called the reverse map table (RMP). The RMP maintains the mapping from the HPA of each memory page to a pair of the ID and GPA of a CVM. It is used by processors on memory access to check that the CVM accessing memory and the GPA of the memory accessed by the CVM match the RMP entry corresponding to the HPA obtained by address translation. As a result, processors deny illegal memory access if the inverted translation fails. This RMP check also fails when the hypervisor accesses the confidential memory of CVMs. The RMP is securely maintained by the processors and the co-processor called the AMD Secure Processor (AMD-SP). Since the hypervisor can register arbitrary mappings to the RMP, CVMs securely validate the registered mappings. The AMD-SP is also used to measure the boot images loaded into the memory of CVMs.

However, it is unclear whether this *delegation architecture* of SEV-SNP is implementable in RISC-V. The aims of this paper are (1) to design the delegation architecture for RISC-V without large trusted hardware components and (2) to clarify the overhead of the runtime checks on address translation.

III. Software-defined SEV

This paper proposes *software-defined SEV (SD-SEV)*, in contrast to hardware-based SEV, which uses SEV-enabled

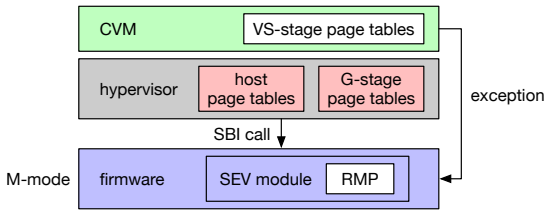


Fig. 3: The system architecture of SD-SEV.

processors and the AMD-SP. SD-SEV realizes the delegation architecture for CVMs using a small trusted software module and a minimal hardware extension. The SEV module runs inside the firmware, which runs in M-mode on RISC-V processors, as illustrated in Fig. 3. Therefore, the SEV module is isolated from an untrusted hypervisor running in HS-mode and is securely executed. It emulates the SEV-SNP features of AMD EPYC processors and the AMD-SP.

The threat model of this paper is as follows. We assume that software running in M-mode, including the SEV module, is trusted. We do not trust the hypervisor or the G-stage page tables it manages. We assume attacks against two-stage address translation by modifying the G-stage page tables, e.g., memory remapping and aliasing attacks. Also, we assume that the hypervisor directly attacks the memory of CVMs. In addition, we assume attacks against the boot images loaded into CVMs by the hypervisor. We exclude physical attacks because it is difficult for a software-based approach to prevent such attacks.

In SD-SEV, the hypervisor and CVMs invoke the SEV module to use SEV-SNP features. Instead of executing the instructions provided by SEV-enabled processors, the hypervisor issues SBI calls to invoke the functions of the SEV module. An SBI call is a mechanism for invoking the underlying software in RISC-V and is used for system calls and hypervisor calls. The hypervisor also issues SBI calls instead of using memory-mapped IO (MMIO) to execute the commands provided by the AMD-SP. To invoke the SEV module directly from CVMs without going through the hypervisor, CVMs execute undefined instructions and raise illegal instruction exceptions. Then, the firmware directly traps them and invokes the SEV module.

For memory integrity, SD-SEV checks memory assignment to CVMs on every memory access and detects illegal changes by the hypervisor. To enable this, the SEV module maintains the RMP like hardware-based SEV. Instead of executing the instruction for registering information on memory assignment, the hypervisor invokes the SEV module and sets a pair of the ID and GPA of a CVM to the RMP entry corresponding to an HPA. In addition, the CVM validates that memory assignment by invoking the SEV module and marking that RMP entry as validated.

Since it is difficult to efficiently perform an RMP check on memory access using standard hardware mechanisms, SD-SEV uses a hardware extension called a *page success exception (#PS)* [5]. A #PS exception is an exception raised when a processor succeeds in address translation by a page

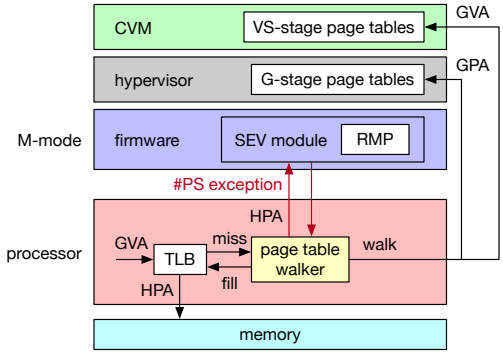


Fig. 4: RMP checks with #PS exceptions.

table walk after a TLB miss, as illustrated in Fig. 4. It is the opposite of a page fault exception, which is raised when a processor fails address translation. For CVMs, this exception is raised after the success of two-stage address translation. Upon a #PS exception, the SEV module obtains the ID of the CVM accessing memory and the GPA and HPA of the accessed memory from the processor. If a pair of the ID and the GPA does not match the RMP entry corresponding to the HPA, the SEV module detects an illegal change in memory assignment to the CVM. Note that an RMP check is not necessary on a TLB hit because the TLB entry is filled only after an RMP check and cannot be modified by the hypervisor.

For memory authenticity, SD-SEV checks the state of software loaded into the memory of a CVM at boot time and guarantees that unmodified software runs in the CVM. Instead of invoking the AMD-SP, the hypervisor invokes the SEV module and computes the hash value of the memory, called a launch digest. Then, it compares the computed launch digest with the one pre-computed by the user and determines whether the loaded software has been tampered with.

IV. Implementation

We have implemented SD-SEV to guarantee the memory integrity and authenticity.

A. RMP

The RMP has an entry for each 4-KB host physical page. Each RMP entry consists of the Assigned field, the address-space identifier (ASID) field, the GPA field, the Validated field, and so on. The Assigned field means whether the page is owned by a CVM. If this bit is 0, the page is owned by the hypervisor. The ASID field holds the ASID of the VM that should own the page. An ASID is uniquely assigned to a VM by incrementing the VMID of the VM because the VMID range starts at 0 but the ASID range starts at 1. The GPA field holds the GPA at which the page should be mapped. The Validated field means whether a CVM has validated the mapping between the HPA and the GPA. If this field is 0, the CVM cannot access the page.

1) SNP_RMP_CREATE Command: The SEV module provides an SBI call for this command to the hypervisor to initialize the RMP. In hardware-based SEV, this command is provided by the AMD-SP. The hypervisor issues this command with a memory region used for the RMP. To prevent the hypervisor from directly modifying the RMP, the SEV module protects the passed memory region using PMP in RISC-V. The SEV module configures the physical memory region used for the RMP to be read-only for the hypervisor. Then, it creates an entry for each 4-KB physical page in the RMP.

2) RMPUPDATE Instruction: The SEV module provides an SBI call for this instruction to the hypervisor to change an RMP entry. The hypervisor executes this instruction with the HPA for a page, the ASID of a VM, the GPA mapped to the HPA, and so on. The SEV module sets a pair of the ASID and the GPA to the RMP entry corresponding to the HPA. It always clears the Validated field in the RMP entry so that the CVM does not access an unintentional page without notice.

When a CVM shares a page with the hypervisor or other VMs, it issues an SBI call for executing this instruction via the hypervisor. Then, the hypervisor translates the specified GPA into an HPA and issues the SBI call for the instruction with the obtained HPA. For example, it is necessary to share the memory used by the virtio devices between a CVM and the hypervisor.

3) PVALIDATE Instruction: The SEV module provides this instruction as an undefined instruction in RISC-V to CVMs to validate the mapping of a page. The guest OS in a CVM executes this instruction with a GVA. The SEV module directly traps the instruction as an exception without being trapped by the hypervisor. Then, it first translates the specified GVA into a GPA using the VS-stage page tables in the CVM. The address of the page tables is identified using the control and status register (CSR) called vsatp. Next, it translates the obtained GPA into an HPA using the G-stage page tables in the hypervisor. The address of the page tables is identified using the hgatp register. Finally, the SEV module searches for the RMP entry corresponding to the obtained HPA and sets the Validated field if the RMP entry matches the ASID of the CVM and the obtained GPA.

The SEV module walks the page tables and performs two-level address translation by itself. Currently, it supports the Sv57 address translation mode among several address translation modes in 64-bit RISC-V processors. The Sv57 address translation mode uses 5-level page tables. Strictly speaking, the SV57x4 address translation mode is used for VS-stage address translation. It is a variant of the SV57 address translation mode and is extended so that GVAs are 2-bit wider.

In a CVM, the firmware validates the memory used by the guest OS. Since it is the first software executed in a CVM, its memory cannot be validated by other software in the CVM. Therefore, the SEV module validates the memory of the firmware when it measures the memory, as described in Section IV-B.

4) RMP Check: When a TLB miss occurs in the memory access of a VM, the processor first walks the VS-stage page tables in the VM to translate the GVA into a GPA, as shown in Fig. 4. Then, it saves the obtained GPA to the mpsegpa register for a #PS exception. Then, it walks the G-stage page tables in the hypervisor to translate the GPA into an HPA and saves the obtained HPA to the mpsepa register. If it succeeds in the page table walk, it raises a #PS exception and invokes the exception handler registered by the SEV module. The handler reads the target HPA from the mpsepa register and searches for the RMP entry corresponding to the HPA.

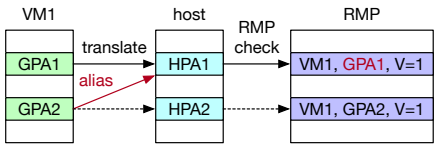
If the Assigned field in the RMP entry is 1, the SEV module first confirms that the memory access is not performed by the hypervisor. If the privilege level on the memory access is HS-mode, the SEV module detects illegal memory access by the hypervisor. Next, it obtains the VMID of the VM from the hgatp register and computes the ASID by adding one to the VMID. Also, it obtains the target GPA from the mpsegpa register. If a pair of the ASID and the GPA does not match the RMP entry, the SEV module detects an attack by the hypervisor. If the pair is correct, the SEV module finally checks the Validated field in the RMP entry. If that field is 0, the SEV module detects an attack by the hypervisor.

B. Measurement

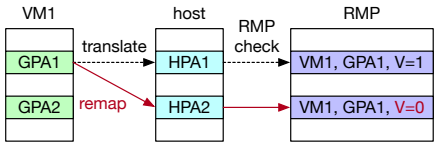
Like hardware-based SEV, SD-SEV measures the contents and metadata of the initial set of memory pages in a CVM into a launch digest. At the creation time of a CVM, the hypervisor stores the boot image, such as the firmware and the guest OS, in the memory of the CVM. Then, it executes several commands provided by the SEV module for each page and computes a launch digest.

The SEV module provides SBI calls for the SNP_LAUNCH command family to the hypervisor. First, the hypervisor issues the SBI call for the SNP_LAUNCH_START command to start the measurement of the CVM. Next, the hypervisor issues the SBI call for the SNP_LAUNCH_UPDATE command to measure the specified page. The SEV module creates the PAGE_INFO structure, which contains the hash value of the target page, the GPA of the page, and the current value of the launch digest. Then, it computes the hash value of this structure using the open-source SSL/TLS library called Mbed TLS. In addition, the SEV module sets the Validated field in the RMP entry corresponding to the HPA of the page. This enables the first software, like the firmware, to be executed in the CVM.

Finally, the hypervisor issues the SBI call for the SNP_LAUNCH_FINISH command to finalize the measurement. The owner of the CVM can pass an identity (ID) block to the SEV module via the hypervisor. An ID block contains the value of the launch digest pre-computed by the owner. If the hypervisor specifies an ID block, the SEV module compares the launch digest stored in the guest context with the pre-computed one and detects tampering of the boot image. The owner of the CVM can also pass ID authentication information, which contains the signature of the ID block and its verification key. Using the ID authentication information, the SEV module can check the integrity of the ID block.



(a) Memory aliasing attack



(b) Memory remapping attack

Fig. 5: The detection of memory integrity attacks.

V. Security Analysis

Like hardware-based SEV, SD-SEV can detect memory aliasing attacks [1]. As illustrated in Fig. 5(a), an untrusted hypervisor configures the G-stage page table for VM1 so that GPA1 is translated into HPA1 at the creation time of VM1. Then, VM1 validates the mapping at boot time. After that, the hypervisor can maliciously modify the G-stage page tables so that GPA2 is also translated into HPA1. At this time, it does not update the RMP by invoking the SEV module. When VM1 accesses GPA2 for the first time after a TLB flush, a #PS exception occurs due to a TLB miss. In this case, the SEV module fails in an RMP check because GPA2 does not match the GPA field (i.e., GPA1) of the RMP entry corresponding to HPA1. If the hypervisor does not perform a TLB flush, VM1 can properly access HPA2 without an RMP check when there is an old TLB entry.

SD-SEV can also detect memory remapping attacks [1]. Let us consider that the hypervisor configures the mapping from GPA1 to HPA1 and that VM1 validates it, as in Fig. 5(b). The hypervisor can maliciously modify the G-stage page tables so that GPA1 is translated into HPA2. If it updates the RMP, the SEV module clears the Validated field of the RMP entry corresponding to HPA2. When VM1 accesses GPA1, a #PS exception occurs due to a TLB miss. At this time, the SEV module can detect the attack because the page is not validated by VM1. If the hypervisor does not flush the TLB, VM1 can properly access HPA1 on a TLB hit. If the hypervisor does not update the RMP, the SEV module can detect the attack because GPA1 does not match the old GPA field (e.g., GPA2) of the RMP entry.

In addition, SD-SEV can detect illegal access to the memory of CVMs from the hypervisor. When the hypervisor accesses HPA1 in Fig. 5(a) for the first time, a #PS exception occurs due to a TLB miss. Since the page for HPA1 is assigned to VM1, the Assigned field of the RMP entry corresponding to HPA1 is 1. Therefore, the SEV module can detect the attack. It should be noted that the hypervisor can read the memory of CVMs in hardware-based SEV. Since the memory of CVMs is not encrypted in SD-SEV, the SEV module denies read access from the hypervisor as well. To precisely emulate the behavior of hardware-based SEV, we need an extra hardware extension to encrypt the confidential memory.

TABLE I: The specifications of execution environments.

	host	emulation	VM
CPU	Intel Core i7-14700	4	2
memory	32 GB	4 GB	1 GB
OS	Linux 6.14	-	Linux 6.15
hypervisor	-	Xvisor 0.3.2	-
firmware	-	OpenSBI 1.6	Basic Firmware
emulator	QEMU 10.0.50	-	-

TABLE II: The binary size of the TCB.

	original firmware	TCB
CoVE (model 2)	113 KB (OpenSBI 1.1)	2124 KB (18.8x)
ACE	264 KB (OpenSBI 1.4)	526 KB (1.99x)
SD-SEV	267 KB (OpenSBI 1.6)	277 KB (1.04x)

VI. Experiments

We conducted several experiments to show the effectiveness of SD-SEV. First, we confirmed the integrity and authenticity of the memory of a CVM using SD-SEV. Then, we examined the overhead of SD-SEV during and after the boot of a CVM and compared the performance with that of a non-confidential VM. We used the QEMU that emulated 64-bit RISC-V processors with the #PS feature because there existed no hardware that supported a #PS exception yet. We ran the Xvisor hypervisor and one or two CVMs on it. Table I shows the specifications of the host PC, the emulation environment, and VMs.

A. TCB Size

We compared the binary size of the TCB in CoVE (model 2) [8], ACE [9] based on CoVE (model 3), and SD-SEV. The implementation of CoVE (model 1) does not exist yet. Table II shows the sizes of the original firmware and the TCB for CVM support. The TCB of CoVE (model 2) is 19x larger than the original firmware because it includes the hypervisor-like TSM. The TCB of ACE is twice as large as the original firmware because the TSM in the firmware is written in Rust. In contrast, the TCB of SD-SEV is only 4% larger. It includes the SEV module with 1,061 lines of code (LOC) and the necessary part of Mbed TLS with 75,166 LOC.

B. Detection of Attacks against CVMs

We examined whether SD-SEV could actually detect illegal changes of memory assignment to CVMs and illegal memory access by the hypervisor. First, we mounted the memory aliasing attack by mapping two different GPAs onto the same HPA in a CVM using the command newly added to the hypervisor. As a result, the SEV module could detect this attack because the accessed GPA did not match the GPA field of the corresponding RMP entry. Second, we mounted the memory remapping attack by exchanging physical pages between two CVMs using a newly added hypervisor command. When we accessed the exchanged physical pages in the CVMs, the SEV

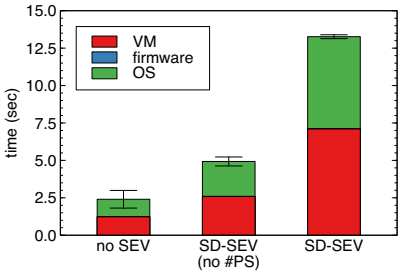
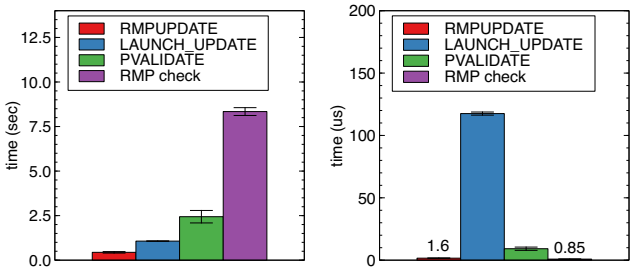


Fig. 6: The boot time of a VM with 1 GB of memory.



(a) Breakdown (b) Per-page execution time

Fig. 7: The breakdown of the boot time.

module could detect the attack because each VM accessed the page that was not owned by itself. Third, we accessed a page in a CVM from the hypervisor. At this time, the SEV module detected illegal access to the page owned by the CVM.

Next, we examined whether SD-SEV could detect tampering when a modified OS was loaded into a CVM. When the hypervisor executed the SNP_LAUNCH_FINISH command, the SEV module compared the computed launch digest with the one pre-computed by the owner of the CVM, which was contained in the ID block. As a result, the SEV module could detect the modification of the boot image.

C. Boot Time

We measured the boot time of a CVM, which was from the creation of a CVM to the boot completion of the guest OS in the CVM. The boot time consists of (1) the time for creating a CVM and loading the boot image, (2) the time for starting up the CVM and booting the firmware, and (3) the time for booting the guest OS. For comparison, we used a CVM with the #PS feature disabled to examine the impact of #PS exceptions. In addition, we measured the boot time of a non-confidential VM on an unmodified QEMU.

Fig. 6 shows the boot time of a CVM with 1 GB of memory. The boot time of the CVM with SD-SEV was 5.5x longer than that of the normal VM. The creation time of the CVM and the boot time of the guest OS were increased by 5.8x and 5.3x, respectively, while the boot time of the firmware was increased by only 1.9x. It was shown that the overhead of #PS exceptions was much larger. The rest of the overhead came from the registration of the pages to the RMP, the validation of the pages, and the measurement of the boot image.

Fig. 7 shows the breakdown of the increase in boot time and the per-page execution time of each task. The time for performing one RMP check with a #PS exception was only 0.85 μs, but the total time of RMP checks was the longest. This is because the number of RMP checks was much larger. In contrast, the number of SBI calls for SNP_LAUNCH_UPDATE was much smaller because the number of pages used for the boot image was small. Therefore, the execution time of one SBI call was much longer due to heavyweight hash computation, but the total time was not so long.

The number of SBI calls for RMPUPDATE was almost the same as that for PVALIDATE, which was almost the same as

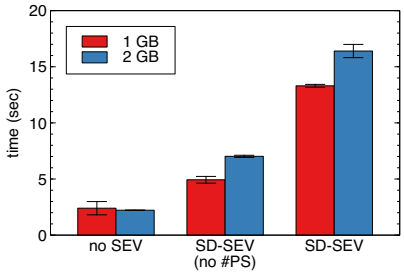


Fig. 8: The impact of the memory size on the boot time.

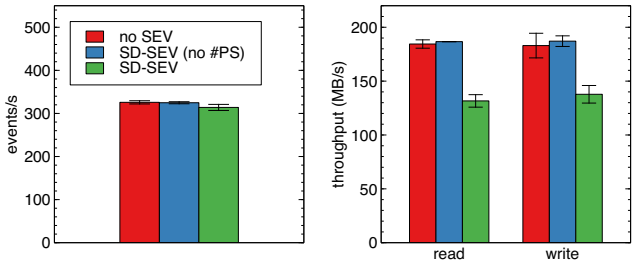
the number of pages assigned to the CVM. However, the time for the registration for the RMP was the shortest, while that for page validation was much longer. This difference came from the execution time of these two types of SBI calls: 1.6 μs for RMPUPDATE and 9.2 μs for PVALIDATE. When the CVM issued an SBI call for PVALIDATE, it took time to trap the execution of the illegal instruction and perform address translation twice per exception.

Fig. 8 shows the comparison of the boot time of a CVM between 1 GB and 2 GB of memory. When the memory size of a CVM was increased from 1 GB to 2 GB, the boot time became 3.1 seconds longer. Even if the #PS feature was disabled, the boot time was increased by 2.1 seconds. This overhead came from RMPUPDATE and PVALIDATE, whose execution counts were proportional to the number of pages.

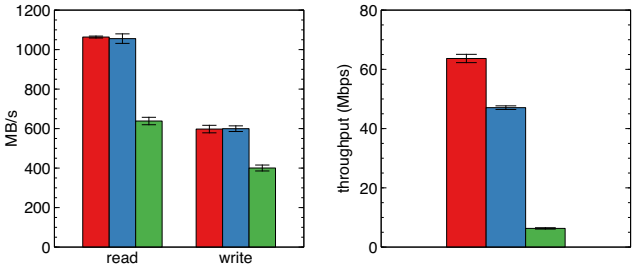
D. Performance

To examine the overhead of SD-SEV after the boot of a CVM, we measured the performance of CPUs, memory access, and file access using sysbench. In addition, we measured the network performance between two CVMs using iperf.

As shown in Fig. 9(a), the CPU performance was degraded by only 4.6% using SD-SEV. This overhead came from RMP checks because the performance was almost the same between the #PS-disabled CVM and the normal VM. The reason why the overhead was small is that the number of RMP checks was small, as shown in Fig. 10. This benchmark accessed only a small amount of memory. The hypervisor performed most of the memory access.



(a) CPU (b) Memory



(c) File (d) Network

Fig. 9: The performance of a CVM with SD-SEV.

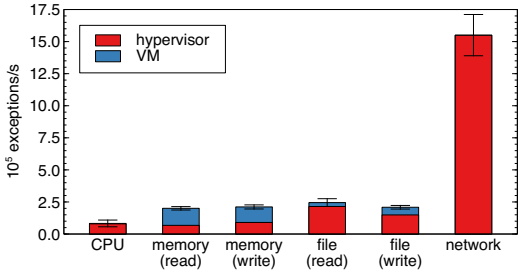
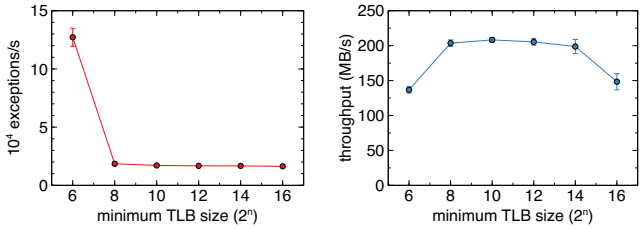


Fig. 10: The number of #PS exceptions.

Fig. 9(b) shows the memory access performance when we accessed 1 GB of memory. Unlike the CPU performance, the memory access performance was degraded by 29% in reads and 25% in writes. Since the performance of the #PS-disabled CVM was almost the same as that of the normal VM, RMP checks are the root cause of this relatively large overhead. This benchmark accessed a larger amount of memory and caused more TLB misses. This benchmark caused 50-75% of the RMP checks in the CVM.

Fig. 9(c) shows the file access performance when we accessed a 500 MB file. Since the hypervisor provides a virtual disk to a VM using a RAM disk in Xvisor, the trend was similar to the memory access performance. Note that the overhead was larger than that of memory access: 40% in reads and 33% in writes. This is because the hypervisor emulated the virtual disk device and raised a larger number of #PS exceptions, as shown in Fig. 10.

Fig. 9(d) shows the network performance when two CVMs performed TCP communication. Unlike the other benchmarks, the performance was degraded by 90%. The reason is that too many #PS exceptions occurred in the hypervisor to emulate virtual network devices, as shown in Fig. 10. Even in the #PS-disabled CVMs, the performance was 26% lower than that in the normal VMs. This is because the code added for the #PS feature was executed too many times.



(a) #PS exceptions (b) Performance

Fig. 11: The impact on the memory read performance.

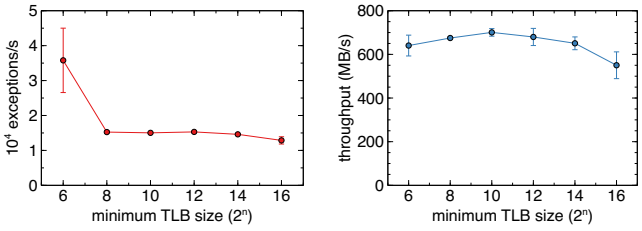
E. Impact of #PS Exceptions

As shown in Section VI-D, #PS exceptions affect the performance of a CVM significantly. Therefore, we examined whether the reduction of #PS exceptions could improve the performance. Since a #PS exception is raised by a TLB miss, we increased the minimum TLB size in QEMU to reduce TLB misses. QEMU dynamically adjusts the TLB size between the minimum and maximum values to improve performance. The minimum TLB size is 2^6 by default, while the maximum is 2^{20}. Therefore, we changed the minimum value between 2^6 and 2^{16}. As the minimum TLB size increases, it is expected to reduce the number of TLB misses and #PS exceptions.

Fig. 11(a) shows the number of #PS exceptions during memory reads for each minimum TLB size. When we increased the minimum TLB size from 2^6 to 2^8, the number of #PS exceptions was reduced by 86%. It was almost the same when we changed the minimum TLB size to more than 2^8. The performance was increased by 50%, as shown in Fig. 11(b). This means that it is effective to reduce the number of #PS exceptions in memory reads.

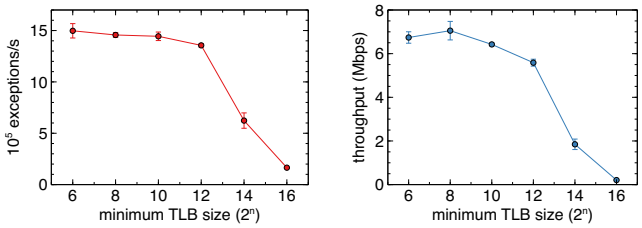
Fig. 12 shows the number of #PS exceptions during file reads and the file read performance. When we increased the minimum TLB size from 2^6 to 2^8, the number of #PS exceptions was reduced by 57%. Like memory reads, it was almost the same after the minimum TLB size became 2^8. However, the file read performance was increased only by 5.4%. It was slightly increased up to 2^{10} but was decreased after that. Therefore, the reduction of #PS exceptions was not so effective in file reads.

Fig. 13 shows the results for network communication. When we increased the minimum TLB size from 2^6 to 2^{12}, the number of #PS exceptions was only slightly reduced. It is possible that the working set size was much larger than the other benchmarks. When we increased the minimum TLB size more largely, the number of #PS exceptions was reduced by up to 89%. However, the network performance was decreased by 97%. This is probably due to the access overhead of the



(a) #PS exceptions (b) Performance

Fig. 12: The impact on the file read performance.



(a) #PS exceptions (b) Performance

Fig. 13: The impact on the network performance.

large TLB. In this benchmark, the reduction of #PS exceptions was not effective at all.

VII. Related Work

Intel TDX [2] securely runs a trusted software component called the TDX module in the secure arbitration mode (SEAM). TDX divides the memory assigned to a CVM called a trust domain (TD) into confidential and non-confidential. Then, the TDX module maintains the secure extended page tables (EPT) for the confidential memory. This secure EPT corresponds to the G-stage page tables in CoVE. Therefore, correct address translation is guaranteed for the confidential memory of CVMs.

Arm CCA [3] provides the Realm world for securely running CVMs called Realms. The physical memory of the Realm world is isolated from the normal world using the granule protection table (GPT), which is similar to the MTT in CoVE. CCA runs the Realm management monitor (RMM) in the Realm world and maintains the Realm translation tables (RTT) in the RMM. The RMM and the RTT are similar to the TSM and the G-stage page tables in CoVE, respectively. They guarantee the correctness of the address translation of CVMs.

ACE [10] is an implementation of the third model of CoVE. It runs the TSM in the most privileged M-mode, but the TSM is written in Rust, which is a memory-safe programming language. Using Rust's ownership, it is guaranteed that each physical page is assigned to only one CVM. In addition, the memory safety of a core part of the implementation has formally been verified using RefinedRust. We could apply a similar technique to our SEV module running in M-mode.

SofTEE [11] is a software framework to implement TEEs for applications without special hardware. It deprivileges the OS kernel by replacing privileged instructions and delegates the execution of privileged operations, such as memory management, to the security monitor. However, it is unclear whether SofTEE can be used to implement TEEs for CVMs.

AnyTEE [12] is a software-defined TEE (sdTEE) framework to implement custom TEEs across multiple instruction set architectures. It uses widely available hardware virtualization primitives to create sdTEEs. To support CVMs, it runs the hypervisor using nested virtualization. However, the paper presents only the implementation of sdTEE versions of SGX and TrustZone. The implementation of TEEs for CVMs such as CoVE is future work.

VIII. Conclusion

This paper proposes SD-SEV, which achieves the delegation architecture of SEV-SNP using a small, trusted software module and a minimal hardware extension. The SEV module maintains the RMP and checks the correctness of address translation on memory access using a #PS exception. We have implemented SD-SEV on QEMU and confirmed that the memory integrity and authenticity of a CVM were guaranteed. Also, we showed that the performance of a CVM was largely affected by SD-SEV.

One of our future work is to improve the performance of SD-SEV. In particular, it is necessary to reduce the overhead of #PS exceptions. We are planning to extend RISC-V processors so as to perform RMP checks inside processors in a usual case, while preserving the generality of a #PS exception.

Acknowledgment

This work was supported by JST K Program Grant Number JPMJKP24U4, Japan.

References

[1] Advanced Micro Devices, Inc., "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," 2020.
[2] Intel Corporation, "Intel Trust Domain Extension," 2023.
[3] 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 *Proc. USENIX Symp. Operating Systems Design and Implementation*, 2022, pp. 465–484.
[4] RISC-V International, "Confidential VM Extension (CoVE) for Confidential Computing Version 0.7," 2024.
[5] H. Yamamoto, T. Hata, and K. Kono, "Page success exception," https://github.com/sslab-keio/page-success-exception.
[6] RISC-V International, "RISC-V Open Source Supervisor Binary Interface (OpenSBI)," https://github.com/riscv-software-src/opensbi.
[7] A. Patel, M. Daftedar, M. Shalan, and M. El-Kharashi, "Embedded Hypervisor Xvisor: A Comparative Analysis," in *Proc. Euromicro Int. Conf. Parallel, Distributed, and Network-based Processing*, 2015, pp. 682–691.
[8] A. Patra, "CoVE (RISC-V Confidential VM Extension)," https://github.com/rivosinc/cove.
[9] W. Ozga et al., "Assured Confidential Execution (ACE) for RISC-V," https://github.com/IBM/ACE-RISCV.
[10] W. Ozga, G. Hunt, M. Le, L. Gaeher, A. Shinnar, E. Palmer, H. Jamjoom, and S. Dragone, "ACE: Confidential Computing for Embedded RISC-V Systems," https://arxiv.org/pdf/2505.12995, 2025.
[11] U. Lee and C. Park, "SofTEE: Software-Based Trusted Execution Environment for User Applications," *IEEE Access*, vol. 8, 2020.
[12] D. Cerdeira, J. Martins, N. Santos, and S. Pinto, "AnyTEE: An Open and Interoperable Software Defined TEE Framework," *IEEE Access*, vol. 13, 2025.