CPC: Flexible, Secure, and Efficient CVM Maintenance with Confidential Procedure Calls

Jiahao Chen, *Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University; Engineering Research Center for Domain-specific Operating Systems, Ministry of Education; Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University;* Zeyu Mi and Yubin Xia, *Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University; Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China;* Haibing Guan, *Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University;* Haibo Chen, *Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University; Engineering Research Center for Domain-specific Operating Systems, Ministry of Education; Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University*

https://www.usenix.org/conference/atc24/presentation/chen-jiahao

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

CPC: Flexible, Secure, and Efficient CVM Maintenance with Confidential Procedure Calls

Jiahao Chen[1,2,3], Zeyu Mi[1,2]✉, Yubin Xia[1,2], Haibing Guan[3], and Haibo Chen[1,2,3]

[1]*Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University*
[2]*Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China*
[3]*Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University*

Abstract

Confidential virtual machines (CVMs), while providing strong data privacy for cloud tenants, pose significant challenges to VM maintenance like live migration and snapshotting. Traditional host-based maintenance, while applicable to conventional VMs, is infeasible for CVMs due to the lack of trust in the host and the prevention of mandated intrusive access from the host. State-of-the-art approaches depend on non-trivial modifications to hardware and firmware and thus lead to notable compromises in security and/or performance. Furthermore, such approaches lack flexibility for upgrades and cross-platform compatibility, hindering the popularity of CVMs on the cloud.

In this paper, we introduce **C**onfidential **P**rocedure **C**alls (CPCs), a flexible approach to the efficient and secure execution of CVM maintenance modules from within the guest. We have implemented prototypes on two leading CVM platforms. Our prototype on AMD SEV showcases the high performance of CPCs, with 3× (resource reclamation) or even 138× (live migration) faster than existing approaches. Our prototype on ARM CCA further confirms CPCs' outstanding security and flexibility.

1 Introduction

Confidential virtual machines (CVMs), which effectively safeguard the privacy of cloud tenants against curious or malicious host software stacks, are increasingly popular in major cloud vendors like AWS and Google Cloud. As such, mainstream hardware vendors have released their respective CVM support such as AMD SEV [4], ARM CCA [7], and Intel TDX [36]. These approaches typically make the guest internal data inaccessible to the host software stack by encrypting the CVM's private memory and isolating its internal states.

However, such approaches present grand challenges for VM maintenance such as live migration and VM snapshotting, which are crucial for cloud operations, as evidenced by the widespread deployment shown in Table 1. For example,

✉Corresponding author: Zeyu Mi (yzmizeyu@sjtu.edu.cn).

Table 1: Common maintenance examples on the cloud.

	Description	Use Cases
Host-driven	The host directly accesses the internal data of the guests to realize these services.	(Live) Migration [19, 26] Snapshot [11, 25, 52] Disaster Recovery [18]
Guest-driven	The tenant installs host agents in VMs that bridge the host-guest semantic gap and work in conjunction with the host software stack.	Logging [9, 16, 27] Security Scanning [1, 12, 46] Monitoring [13, 28, 29] Backup [15] Resource Reclamation [41]

cloud vendors rely on live VM migration to upgrade infrastructure and optimize deployment strategies without having to stop VMs. Meanwhile, cloud tenants depend on maintenance services like monitoring and security scanning to guarantee the secure and stable operations of their workloads.

Traditional maintenance approaches typically require either intrusive accesses from the host to the guest (host-driven) or the deployment of agent software within the guest to access its internal data (guest-driven). In the CVM scenario, such conventional approaches become impractical, because CVMs essentially block intrusive and direct host accesses to the guest, and the guest denies the untrusted agent software provided by the host.

State-of-the-art approaches to addressing the above challenges mandate non-trivial modifications to the firmware and/or hardware by CVM hardware vendors [38, 44, 45, 50, 51]. In particular, they transfer maintenance functionalities, which the host software stack cannot perform, to more privileged components such as the firmware with higher privileges to access the private guest data. However, such an approach has several limitations. First, it lacks sufficient flexibility to support the wide variety of customization services listed in Table 1, primarily due to the difficulties in modifying firmware and hardware. The lack of cross-platform compatibility exacerbates this inflexibility, as these methods are confined to specific hardware platforms. Second, this approach enlarges the firmware's code base and complicates its interactions with the host and guest, making it more vulnerable to security breaches [59, 68, 76–78, 89, 90, 99]. Given that the firmware acts as the universal trusted computing base

(TCB) for the entire system, any reduction in its security poses risks to both the guest and host. Lastly, current methods encounter significant performance issues in critical scenarios. For instance, on AMD SEV platforms, the firmware runs on the AMD Secure Processor (AMD-SP), whose limited computing power becomes a bottleneck for maintenance operations. According to our tests, it imposes an overhead of 1,986× in live migration scenarios (§7.4).

A fundamental issue with current approaches lies in their inappropriate choice of vantage point for maintenance operations. We argue that a more attractive option to explore is to position the tenant-trusted maintenance modules within the guest, especially since mechanisms like upcalls exist for hosts to activate specific guest functions. Nonetheless, directly applying existing mechanisms to support in-CVM maintenance modules faces two challenges. First, these in-CVM maintenance activities can lead to performance degradation due to resource contention with the guest workload, potentially leaving the maintenance function resource-starved when initiated by the host. According to our tests, this can result in about a 3× slowdown in resource reclamation scenarios (§7.3). Second, certain maintenance tasks, such as disaster recovery [11, 18, 25, 52], have strong fault tolerance requirements to work correctly even when the guest OS fails. Current methods fail to provide them with isolated execution environments.

To reduce resource contention with the guest OS, a new design should allow the maintenance module to access separate computing resources, thereby ensuring its efficiency. Additionally, for specific maintenance tasks requiring isolation from the guest OS, the maintenance modules should run within protected execution environments. Therefore, the key to addressing the issues lies in exploring a new mechanism capable of providing the host with the semantics of ***host invocation of targeted maintenance operations with separate and protected resources***.

We observe that while current CVMs limit the host's intrusive access to the guest's data plane, the hypervisor still exerts influence over the control plane. The data plane refers to the guest's code logic and data processing, whereas the control plane involves the host's management of physical resource allocation and reclamation for the guest, such as scheduling vCPUs. Leveraging such residual control planes, we come up with an idea that ***extends the semantics of vCPU scheduling into the semantics of host invocations of maintenance procedures***.

Based on this idea, this paper introduces **C**onfidential **P**rocedure **C**all (CPC), a flexible, secure, and efficient mechanism for CVM maintenance. CPCs enable the host to invoke maintenance functions agreed and trusted by the guest via scheduling vCPUs for the host (hvCPUs). Their bound CPCs thus operate on these dedicated vCPU threads without competition with other guest workloads. Cloud tenants can develop and customize maintenance functions within CPCs, leading to enhanced flexibility and efficiency free from factors such as firmware and guest workloads. The internal code and data of the CPCs, forming their data plane, are safeguarded by the CVM. The host only triggers and suspends these CPCs through the legitimate control plane outside the CVM, akin to traditional vCPUs, without security degradation on the CVM.

To further protect the critical maintenance modules from potential attacks by a misbehaving guest OS in scenarios such as disaster recovery and snapshot [11, 18, 25, 52], we introduce Confidential Page Table Isolation (CPTI), which is a new intra-CVM isolation method that is compatible with all current leading CVM platforms [4,7,36]. This technique aids CVMs in creating internal isolated execution environments specifically for CPCs so that they can achieve complete isolation of their code and data from the guest OS. This enables the CPCs to execute securely in accordance with the cloud tenant's initial setup plans, even under attack attempts from a compromised guest OS.

We have implemented CPC prototypes on AMD SEV-SNP and ARM CCA platforms. The AMD-based prototype demonstrates the excellent performance of CPCs on real machines. Freed from the constraints of AMD-SPs, CPCs can encrypt and extract private data at more than 340× the speed of current approaches. In a resource reclamation scenario, CPCs obtain sufficient computing power for the maintenance modules under busy guests and thus reclaim free memory 3× faster than the current virtio frontends. In the live migration test, CPCs are 138× faster than the current AMD-SP-based solution. Although no hardware is available for ARM CCA, we implemented CPCs and CPTI on the official simulator to verify the security and excellent cross-platform compatibility of our design.

In summary, the contributions of the paper are:

- We systematically analyze the dilemma of current mainstream CVM platforms on maintenance operations.
- We propose CPCs, a new mechanism bridging the semantic gap between the host and in-guest maintenance procedures, to support flexible, secure, and efficient CVM maintenance.
- We show CPC's security improvements on the overall system through the prototype implemented on the ARM CCA platform.
- We show the performance advantages of CPCs over existing approaches by conducting performance experiments on the AMD SEV-SNP platform.

2 Background and Motivation

2.1 Mainstream CVMs

CVM platforms protect cloud tenants' VM instances in trusted execution environments (TEEs) isolated from the host. The current mainstream CVM platforms are AMD

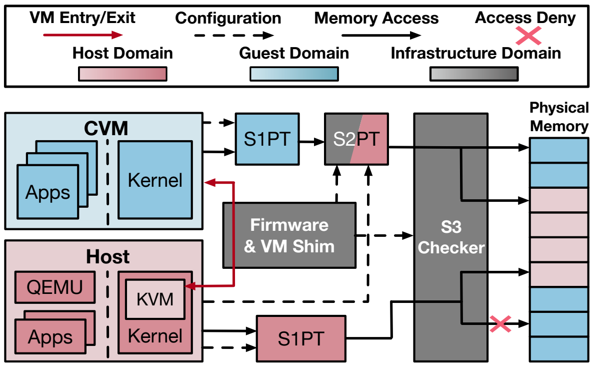

Figure 1: The architecture overview of mainstream CVM platforms. The hypervisor in the figure is exemplified by QEMU/KVM. "S1PT" refers to the stage-1 page table, which translates CVM's guest virtual addresses (GVAs) to guest physical addresses (GPAs), or host's virtual addresses (VAs) to physical addresses (PAs). "S2PT" stands for the stage-2 page table, which translates CVM's GPAs to host physical addresses (HPAs). The S2PTs for the private memory on ARM CCA and Intel TDX platforms are checked and configured by the trusted infrastructure domain, while those of AMD SEV are configured directly by the host.

SEV [4], ARM CCA [7], and Intel TDX [36]. Their architectures can all be summarized in Figure 1, where the system is divided into three domains: the host domain controlled by the cloud vendor, the guest domain running the tenant's VM instances, and the infrastructure domain consisting of the trusted firmware and hardware. Both cloud vendors and tenants place trust in the infrastructure domain and the privileged software in their domains.

CVM platforms primarily rely on preventing intrusive access by the host to the guest's private memory and registers as their core protection mechanism at runtime. This protection is typically achieved through encryption and isolation.

As shown in Figure 1, for encryption, the guest memory is divided into private memory (in blue), which is isolated from the host, and shared memory (in red), which can be accessed from the host. Each time a guest writes to its private memory, the data is encrypted by the hardware automatically, and when the guest reads data from its private memory, it is automatically decrypted inside the CPU. Both of the operations are transparent to the guest.

For memory isolation, CVM platforms also provide additional stage-3 memory isolation mechanisms (S3 checker) [32, 35, 47], such as RMP for AMD platforms and GPT for ARM platforms. These mechanisms record the ownership of the physical memory and block all host access to guests' private memory. The configuration of S3 checkers is typically done by the trusted firmware beyond the hosts' control.

Furthermore, the isolation of vCPU states is accomplished through trusted firmware or hardware filtering so that the host can only access the necessary portion of the vCPU states when serving guests.

It is also worth noting that side-channel attacks and denial-of-service (DoS) attacks are not considered in the threat model of CVMs. Thus, the host OS still controls various resources, such as vCPU scheduling and physical memory management.

2.2 Dilemma of CVM Maintenance

Both cloud vendors and tenants perform maintenance operations on VMs to ensure the security, efficiency, and cost-effectiveness of cloud computing [1, 9–16, 18, 25, 27–29, 46]. However, CVMs make the traditional methods of maintenance no longer feasible. These failed maintenance operations can be grouped into two categories: host-driven and guest-driven as shown in Table 1.

Host-driven maintenance refers to a set of transparent maintenance operations performed by cloud vendors on the guest [11, 18, 25]. These operations include tasks like migrating VM instances, disaster recovery, and taking snapshots. Traditionally, these operations rely on the hypervisor's intrusive access to the guest's internal states, but CVMs prevent such access, making traditional maintenance methods ineffective.

On the other hand, guest-driven maintenance services are embraced by cloud tenants to ensure the long-term security and efficiency of their VMs and workloads [1, 9, 10, 12–16, 27–29, 46]. For example, tenants install security agents provided by the cloud vendor for additional security services. Such maintenance services are built on the additional information collected by the in-guest agents. However, according to CVM's threat model, tenants refuse to install the untrusted agent software provided by cloud vendors and distrust all of the host software stack. Therefore, guest-driven maintenance services are also disabled by CVMs.

The absence of maintenance poses challenges for both cloud vendors and tenants, hindering the widespread adoption of CVMs in cloud computing. Cloud vendors encounter higher costs, deployment difficulties, and infrastructure upgrade challenges. Similarly, cloud tenants face obstacles in ensuring the security and stability of their workloads, along with additional self-maintenance expenses.

2.3 Limitations of Existing Approaches

Current primary approaches involve the CVM hardware vendors integrating the maintenance modules into the infrastructure domain, typically the firmware, and exporting additional interfaces for the host [38, 44, 45, 50, 51]. However, such approaches have several limitations.

First, the infrastructure-based approach lacks flexibility. Unlike the guest and host domains, which can be flexibly defined, upgrading the infrastructure domain is more difficult. This limits current approaches to relatively fixed and generic maintenance scenarios [11, 18, 19, 25, 26, 52], and makes it difficult to adapt to the scenarios that are customized by cloud vendors and are constantly evolving [1, 9, 12, 13, 16, 27–29, 46], such as various monitoring services and frequently

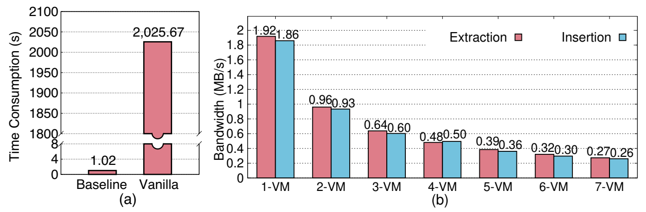

Figure 2: Time consumption for CVM live migration and performance of AMD-SPs on AMD platforms. Figure (a) depicts the time consumption of the live migration for a traditional VM and a CVM with identical configurations on the same machine. The source and destination instances are also located on the same machine to minimize the impact of unstable networks on the results. "Baseline" represents the time consumption for traditional VMs, while "Vanilla" represents the time consumption for the SEV CVMs. Figure (b) illustrates the bandwidth of CVM memory extraction and insertion based on the AMD-SP for different numbers of CVMs. The data represents the average bandwidth across all the CVMs.

updated security scanning policies. The inflexibility is further evident in the absence of cross-platform compatibility and alignment. Specifically, infrastructure-based approaches from various hardware vendors are exclusive to their own platforms, and their development progress also varies greatly. For instance, ARM CCA currently lacks an officially determined migration solution [48], and Intel TDX lacks an available snapshot implementation. These issues result in inconsistent CVM maintenance practices and increase the development efforts required by cloud vendors and tenants to adapt to each platform.

Second, they lead to a degradation of security. Moving maintenance functionalities from the untrusted hypervisor to the infrastructure domain increases the TCB for all parties in the system. The complexity and extended interfaces make the infrastructure domain more vulnerable [34, 59, 68, 76–78, 89, 90, 99]. Take ARM CCA as an example, its infrastructure domain is very compact, and according to previous work [83], the tiny Realm Management Monitor (RMM) with only 3.2K LOCs is formally verified to ensure security. Introducing additional functionalities into it exposes the RMM to larger attack surfaces and makes verification more challenging.

Finally, existing approaches suffer from performance issues. Take the AMD SEV platform as an example, which is currently the most widely deployed CVM platform. It relies on the trusted firmware working on the ARM-based AMD Secure Processor (AMD-SP) in the SoC. However, the AMD-SP has very limited computing power. This results in performance bottlenecks for certain maintenance scenarios. According to our test, the firmware-based live migration is nearly 1,985.95× slower than normal VMs on the same machine, as demonstrated in Figure 2a. Upon analysis, we found that the primary overhead comes from two steps: the *extraction* step involves encrypting and extracting memory pages from the source CVM, and the *insertion* step involves inserting and decrypting memory pages into the destination CVM. So we further tested these two steps on the machine, an AMD platform with 128 cores and 502GB of memory supporting the SEV-SNP feature. As shown in Figure 2b, the AMD-SP only provides a throughput of 1.92MB per second for CVM memory extraction, and 1.86MB per second for insertion. As the number of CVMs increases, this computing power is further shared out, resulting in the inverse trend as shown in the figure. Such throughput would completely collapse many important maintenance services, such as the disaster recovery service provided by Azure, which needs to be built on a constant snapshot data throughput of more than 100MB per second [18]. The low throughput also clarifies the data in Figure 2a, where simultaneous memory extraction and insertion of the source and destination instances respectively on the same machine would result in a throughput of only 0.93 to 0.96MB per second on both sides. Consequently, the migration of the CVM with 2GB of memory takes over 2 thousand seconds to complete.

In addition, certain upcoming maintenance solutions are based on nested virtualization [51, 65]. However, they still cannot get rid of the three limitations mentioned above. Specifically, they need to rely on specific hardware features to support the L1 hypervisor with the guest OS running in the L2 VM. Such approaches not only depend on specific hardware platforms but also significantly increase the guest TCB by the L1 hypervisor, while performance is degraded by the lengthy VM exit handling of nested virtualization.

3 System Design Overview

The limitations of current approaches motivate us to question the validity of the current technological route that relies on modifying the infrastructure domain, and to explore an alternative design that meets the following objectives:

- **Flexibility:** Our design should allow cloud vendors and tenants to flexibly customize and update maintenance modules without having to suspend or migrate the VMs, or even reboot the physical machine. Additionally, the design should ensure compatibility with all major CVM solutions without hardware modifications.
- **Security:** The design should uphold the security of CVMs. The clear security boundary between the guest and host must be maintained, without new attack surfaces introduced for both sides.
- **Efficiency:** The design should mitigate performance limitations on maintenance modules caused by factors such as AMD-SPs or guest workloads.

Observation and Key Idea. The root cause of existing approaches' limitations lies in the inappropriate choice of vantage point for maintenance modules. We choose to place maintenance modules in the guest. While mechanisms like upcalls already support host calls to functional modules in the guest, seemingly sufficient for moving maintenance modules into the guest OS, this approach has two challenges.

First, these in-CVM maintenance activities can lead to performance degradation due to resource competition with the guest workload, potentially leaving the maintenance function resource-starved when initiated by the host. Second, certain critical maintenance scenarios require isolating the maintenance modules from the potentially compromised guest OS, such as security logging and disaster recovery [18]. However, current mechanisms cannot provide such isolated execution environments for the maintenance modules. We conclude that these challenges stem from the semantic gap in the current mechanism. This prompts us to explore a new design that supports the host invoking the target maintenance operations with additional resources that are separate and protected.

Fortunately, we observe that while the CVM restricts the host's influence on the guest in the data plane, it retains control in the control plane. Building upon this observation, we come up with the basic idea for our design, extending the semantics of vCPU scheduling into the semantics of host invocations of maintenance procedures. Based on this fundamental idea, we further design a comprehensive solution for CVM maintenance, which we call Confidential Procedure Calls (CPCs).

As shown in Figure 3, CPCs are new entities trusted by cloud tenants in the guest domain, and each of them contains a maintenance procedure to be invoked by the host. They operate on a distinct set of vCPUs called hvCPUs (vCPUs for hosts), while the guest OS uses gvCPUs (vCPUs for guests).

gvCPUs are involved in scheduling within the host kernel and provide ongoing support to the guest OS with the same behavior as traditional vCPUs. While hvCPUs do not participate in the standard host kernel scheduling. Instead, once associated with a specific CPC by the guest OS, the hvCPU thread relinquishes CPU resources to the host and enters a sleep state. When a maintenance task arises, the host can initiate the execution of the corresponding CPC by scheduling the relevant hvCPU.

During the startup of a CVM, hvCPUs and CPCs are created and initialized. First, the CVM partitions the vCPUs into gvCPUs and hvCPUs at startup. The guest OS recognizes only gvCPUs as the CPU abstraction and runs on them. Next, the CPC loader within the guest OS packages the necessary maintenance modules into CPCs and loads them onto the hvCPUs based on the tenant's definitions. It also establishes their execution contexts. Since the CPC loader and maintenance modules are compiled and packaged with other guest components, both of them can be validated by existing CVM attestation. The CPC loader then registers the mapping between CPCs with different maintenance functions and hvCPUs to the CPC monitor in the host via hypercalls. This allows the CPC monitor to distinguish between gvCPUs and hvCPUs and obtain the corresponding maintenance semantic for each hvCPU. Finally, the CPC monitor places hvCPU threads into a sleep state and invokes the corresponding hvCPUs according to the incoming maintenance scenarios.

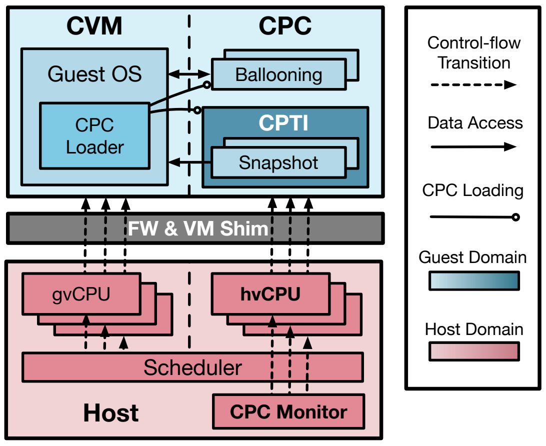

Figure 3: The architecture overview of CPCs.

In the runtime of the CVM, if the cloud vendor initiates a maintenance operation, such as migration or snapshot, the hypervisor actively activates the relevant hvCPUs through the CPC monitor, allowing the corresponding maintenance functions to start working. When the guest requires specific maintenance services, such as security logging and kernel scanning, the host triggers corresponding CPCs based on the notifications from the guest or a pre-planned schedule provided by the tenant.

Assumptions and Threat Model. Based on the maintenance scenarios, the operation modes of CPCs are categorized into normal mode and secure mode. In normal mode, we adhere to the existing threat model of CVMs, maintaining a clear security boundary between the guest and host. The untrusted host software stacks all run outside the CVM and do not have access to any in-CVM state, while the trusted CPCs established by the guest operate entirely within the CVM. Both the guest and host software assume that the hardware is correctly implemented and trust the firmware and software provided by hardware vendors.

In secure mode, tenants activate Confidential Page Table Isolation (CPTI) for critical CPCs, to establish isolation between the maintenance modules and the guest OS. This scenario is reasonable and practical, especially given the extensive code base and the large number of vulnerabilities of the guest kernel [22,62,72,91,92]. Upon the initial launch of the CVM, the cloud tenant places trust in both the guest domain and the infrastructure domain, and configures CPTI, which remains unmodifiable thereafter. Once the guest workload starts, the tenant no longer trusts any components within the guest domain, except for the CPCs protected by CPTI. The critical functions in these CPCs, such as snapshots and disaster recovery, can provide correct services to the tenant based on the initial settings, even in the presence of an errored guest OS.

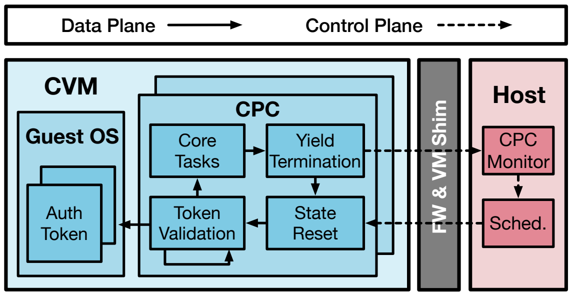

Figure 4: CPC state machine with the in-host control plane and in-CVM data plane.

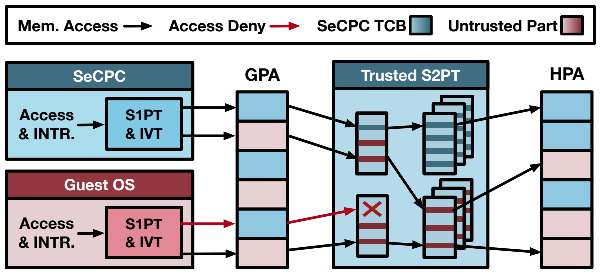

Figure 5: Triple table isolation of CPTI. "Trusted S2PT", "S1PT" and "IVT" in blue respectively correspond to the three layers of isolation introduced in §4.2.

4 CPC Design

In this section, we commence with a detailed introduction of the state machine of CPCs in §4.1 to explain the execution procedures of CPC-based maintenance tasks. We then introduce the security mode that should be activated in specific critical scenarios in §4.2. In §4.3, we discuss how to optimize the CPC code to reduce guest's development efforts. Finally, to address the practical challenges arising from the mutual distrust between the host and guest in CVM-based maintenance, we propose a set of abort protocols for cloud computing in §4.4.

4.1 CPC State Machine

The core maintenance logic of a CPC is designed as a state machine driven by both the in-host control plane and the in-guest data plane, as shown in Figure 4.

On the in-host control plane side, the hypervisor modules invoke the CPC monitor's encapsulated interfaces to initiate maintenance operations. The CPC monitor then awakens and reschedules the corresponding hvCPU based on the requested semantics. This hvCPU thread then enters the CVM for processing the maintenance task on the data plane, which remains confidential to the host. Once the maintenance task has been completed, it proactively makes a specific hypercall to generate a VM exit to the host. Finally, the CPC monitor suspends this hvCPU thread again to free up the occupied computing resources.

On the in-guest data plane side, the maintenance procedure can be split into four primary steps:

- **Token Validation:** Before executing the core functionality of a CPC, it must validate the authorization token set by the guest to confirm whether the guest OS permits this execution. This step ensures that the unintended CPC invocations are avoided.
- **Core Tasks:** After authorization confirmation, the CPC executes the core maintenance functions bound to it.
- **Yield Termination:** Upon completing the core functionalities, the CPC clears the authorization token and makes a hypercall to notify the CPC monitor, which results in the suspension of its hvCPU thread.
- **State Reset:** When the host reactivates the CPC, it returns to the first step to reassess whether execution is currently permissible. This design enables the CPC to be repeatedly invoked as needed.

By splitting the control plane and data plane, the CPC state machine maintains a clear security boundary between the host and guest, ensuring no security degradation on the CVM. Meanwhile, the host can precisely invoke the desired maintenance tasks through the semantics bridged by CPCs.

4.2 Confidential Page Table Isolation

There is a gap between the current CPC approach and traditional approaches as well as infrastructure-based approaches. The guest cannot modify host or infrastructure components, but it can access the code and data within CPCs. This gap makes CPCs inadequate for certain scenarios [11, 18, 25]. For example, tenants require access to the security logging services to retrieve diagnostic information even in the event of a guest OS crash or compromise. They may also need a snapshot service for data recovery. In such cases, the corresponding CPC should be able to execute the tenant's predefined functions reliably even in the presence of a misbehaving guest OS. Therefore, we need to explore a design for making CPCs reliable and secure in these scenarios, which we call secure CPCs (SeCPCs).

Hecate [65] addressed this challenge by using the AMD VMPL (Virtual Machine Privilege Level) feature, which runs the maintenance modules in the L1 hypervisor in VMPL0, isolated from the guest kernel in VMPL1. However, as mentioned in the paper, *"Other new confidential VM technologies such as Intel TDX and ARM Realm lack a VMPL-like isolation inside their confidential VMs."*, this solution is only applicable to AMD platforms and limits the cross-platform compatibility of SeCPCs. Fortunately, we observe that stage-2 page tables for guest private memory on the other two major CVM platforms are trusted for the protection of infrastructure domains. This allows us to design the Confidential Page Table Isolation (CPTI) technique, depicted in Figure 5, making SeCPCs compatible with all CVM platforms. CPTI consists of the following three layers of isolation.

Stage-2 Page Table Isolation: Traditionally, all vCPUs in a VM share the same view of the guest physical address (GPA) space controlled by the shared stage-2 page table.

However, with CPTI, a separate stage-2 page table is established by the trusted firmware for the hvCPU where the SeCPC resides. This enables the hvCPU to access GPA regions beyond the private memory views of other vCPUs. By storing critical data and code of SeCPCs in these isolated GPA regions, a compromised guest OS running on other vCPUs cannot modify SeCPCs after the secure mode is activated. Moreover, when the hvCPUs of SeCPCs share the same physical cores with other vCPUs, the trusted firmware flushes the TLBs on demand to enhance the isolation. This design achieves CVM internal isolation for ARM CCA and Intel TDX platforms. It should be noted that this approach is not applicable on AMD platforms where the untrusted host directly controls the S2PT without check by the infrastructure domain, so a VMPL-based scheme should be adopted instead.

Furthermore, to minimize the memory footprint of the additional stage-2 page tables, trusted firmware reuses most of the existing page table entries (PTEs) in the root of the hvCPU's stage-2 page table, and only rebuilds a few new ones for the isolated GPA regions.

Stage-1 Page Table Isolation: Simply isolating the stage-2 address translation is insufficient. A malicious guest kernel may modify the stage-1 page table of an SeCPC to remap its GVA to the unprotected GPA regions. To address this issue, the SeCPC builds its own independent stage-1 page table within the isolated memory, effectively mitigating such remapping attacks.

Interrupt Vector Table Isolation: In order to prevent a malicious guest kernel from disrupting the execution flow of SeCPCs by sending Inter-Processor Interrupts (IPIs) or triggering other unexpected interrupts, separate interrupt vector tables (IVTs) with secure handlers are established in the isolated memory as shown in Figure 5. This ensures the stable execution of SeCPC's code.

SeCPCs are set by the tenant once at the beginning of the CVM startup and cannot be reset or changed thereafter. Unlike normal CPCs, they do not check guest authorization tokens, ensuring that they consistently execute the core functions specified by the tenant upon being scheduled. With the three layers of isolation provided by CPTI, SeCPCs can operate reliably and as expected throughout the lifecycle of the CVM.

4.3 Optimizations for CPC Development

Improper implementations of CPCs by cloud tenants lead to extensive development efforts and compromised security. To further enhance the practicality of CPCs, we propose four optimizations for CPC development.

Optimization 1: Following the philosophy of separating the control plane from the data plane. For example, in the case of snapshot operations, the CPC only needs to be equipped with tiny operators for the encrypted extraction of private memory and vCPU states. Other steps involved in the snapshot operations, such as image splicing and file building, can be offloaded to the host. Since the extracted data is encrypted, the tenant's privacy cannot be compromised by these offloaded steps.

Table 2: Description of generic maintenance operators.

Name	Description
Memory Encryption Extraction (**MEE**)	Encrypt and extract the private data from the target GPA to the host domain.
State Encryption Extraction (**SEE**)	Encrypt and extract the private states from the target vCPU to the host domain.
Memory Decryption Insertion (**MDI**)	Insert and decrypt the private data to the target GPA in CVM.
State Decryption Insertion (**SDI**)	Insert and decrypt the private states to the target vCPU in CVM.

Optimization 2: Reusing generic operators in multiple scenarios. The data-extracting operators mentioned above can be reused for other maintenance operations, such as migration and security logging. By adopting these generic operators outlined in Table 2, tenants' development efforts on CPCs can be effectively reduced.

Optimization 3: Open sourcing for public validation. Cloud vendors can contribute to open-sourcing certain maintenance modules, such as security scanning and monitoring. This allows for open testing and verification, enabling tenants to reuse these components directly or with additional customizing privacy filters at low development costs.

Optimization 4: Distinction between CPC and SeCPC scenarios. It is worth noting that implementing SeCPCs requires more efforts compared to normal CPCs. Since SeCPCs require the three layers of isolation and cannot share guest OS modules at runtime, they need to additionally implement the functionalities on their own. Consequently, implementing maintenance operations without isolation requirements in normal CPCs can significantly decrease development efforts.

With the optimizations recommended above, cloud tenants can minimize development efforts while guaranteeing CVM security.

4.4 Confidential Abort Protocol

In most cases, additional resources for maintenance are charged [2, 30, 31]. However, in certain scenarios, the cloud vendor offers maintenance for free (or at a discount), such as during service trials or resource reclamation for overcommitment [2, 17, 24]. This allows dishonest tenants to temporarily obtain hvCPUs beyond the host billing for other workloads. This can be addressed through a straightforward approach named Confidential Abort Protocol (CAP), where the basic idea is that dishonest tenants only hurt themselves. Take resource reclamation and migration scenarios as examples.

For a *CPC-Reclamation*, the host only needs to set a throughput threshold based on the economic value of the reclaimed resources. When the CPC cannot provide a sufficient amount of reclaimed resources, the host assumes that the free resources in the guest are depleted and stops *CPC-Reclamation*. A dishonest guest cannot excessively divert resources from the hvCPU to avoid reclaiming below the threshold. On the other hand, if it deceptively commits unrecoverable resources to the host to boost throughput, it will error out due to those resources being taken without any damage to the host.

In the case of *CPC-Migration*, the host can set a migration time limit. Specifically, since the migration time is proportional to the size of the guest memory, the host can accurately estimate the reasonable CPU time that *CPC-Migration* should occupy. When the time limit expires, the host just deschedules the *CPC-Migration*. A dishonest guest that overappropriates hvCPU resources will cause the migration to not complete, resulting in errors in its destination instance.

In summary, cloud vendors can utilize CAP as a practical way to prevent resource encroachment. For other potential scenarios, CAP can also be extended by aligning to the basic idea and examples mentioned above.

5 Implementation

We have implemented prototypes of CPCs and their corresponding use cases on both the AMD SEV and ARM CCA platforms. The AMD use cases are implemented for the performance evaluation on real-world machines. Meanwhile, implementing the SeCPC use cases on the ARM CCA platform, which lacks a VMPL-like feature, demonstrates the security and cross-platform compatibility of (Se)CPCs.

5.1 CPCs on AMD Platforms

We implemented CPCs using QEMU/KVM on an AMD platform that supports the SEV-SNP technology. The host kernel version is Linux 6.1.0-rc4 and the guest kernel version is Linux 6.2.0.

First, since common maintenance operations like migration and resource reclamation are usually initiated through the QEMU command line and implemented within QEMU, we integrated the CPC Monitor into QEMU to collaborate more closely with these modules. Modules in QEMU call the CPC monitor's interface to awaken the corresponding hvCPU thread and invoke the desired CPC. Once the CPC completes its task, it triggers a hypercall *CPC_FINISH* and KVM forwards the handling to QEMU so that the CPC monitor can suspend the hvCPU thread. The modifications to QEMU amounted to approximately 870 lines of code (LOCs).

Second, we introduced a set of CPC-related hypercalls like *CPC_FINISH* and *CPC_HVCPU_REGISTER* to the KVM module in the host kernel. The host kernel simply forwards these hypercalls to the CPC Monitor in QEMU for further processing. Adjustments to the host kernel amounted to approximately 280 LOCs.

Finally, we developed CPC use cases within the guest kernel and AMD SVSM [5, 20]. We rectified the failure of the *isolcpus* kernel command-line parameter under AMD SEV-SNP and utilized it to designate certain vCPUs as hvCPUs. The following four use cases were developed:

CPC-Ballooning: We transplanted the frontend of virtio-balloon to a CPC. When the backend in QEMU attempts memory reclamation, it activates *CPC-Ballooning* through the CPC monitor to collect the guest's free memory.

CPC-Snapshot: We implemented an SeCPC with an MEE operator (as listed in Table 2) in the VMPL0 SVSM, which is isolated from the guest kernel in VMPL1. When the QEMU intends to retrieve encrypted data from the CVM to assemble the image, the CPC monitor activates this SeCPC and submits the target GPAs.

CPC-SecureLog: To ensure the security of critical logs from being tampered with by the guest kernel, we implemented an SeCPC in the SVSM to preserve the logs in the isolated VMPL0 memory. Meanwhile, the CPC monitor can utilize *CPC-Snapshot* to obtain the encrypted security logs and forward them to the cloud tenant.

CPC-LiveMigration: Since the current official live migration scheme is not yet adapted to SEV-SNP [38], for comparison, we implement CPCs in a non-SNP SEV environment. According to the optimizations suggested in §4.3, we only implement the MEE and SEE operators within the CPC of the source CVM and reuse the modules of QEMU on the host side to transfer the encrypted data. Similarly, only MDI and SDI operators are implemented within the CPC at the destination.

5.2 CPCs on ARM Platforms

We implemented the ARM CCA prototype on the FVP_Base_RevC_2xAEMvA platform, which is the only available platform supporting the Realm Management Extension (RME). Both the guest and host utilize the Linux 6.2.0-rc1 kernel, while the VMM is KVMTOOL 3.18.0.

First, we implemented the CPTI prototype in the Realm Management Monitor (RMM) by providing separate S2PTs for SeCPCs. The RMM configures the *vttbr_el2* register to the address of the corresponding S2PT root for each vCPU upon entering the Realm, instead of sharing the same copy for all vCPUs. Additionally, RMM exports an RSI (Realm Services Interface [8]) to the guest to register the range of protected memory accessible only by the SeCPC. Once the initial configuration is complete for the SeCPC, any subsequent attempts to modify the configuration are rejected to maintain isolation even if the guest kernel is compromised.

Next, in the host kernel, we modified the RMI (Realm Management Interface [8]) that creates Realms. This modification requires the host to allocate additional memory pages to the RMM for building the necessary additional S2PT structures required for CPTI. Additionally, the host forwards all CPC-related hypercalls to the CPC monitor in KVMTOOL for further processing.

Finally, within the guest, we utilize the CPTI support of RMM to implement two SeCPC use cases: *CPC-Snapshot* and *CPC-SecureLog*. The *CPC-Snapshot* contains an MEE operator that encrypts and exports private memory data within the protected SeCPC. On the other hand, the *CPC-SecureLog* persistently records security logs in CPTI memory and also allows *CPC-Snapshot* to export its logs in the encrypted format defined by the tenants.

6 Security Analysis

In this section, we analyze the overall system security of CPCs from the perspective of the infrastructure domain, host domain, guest domain, and SeCPC, respectively.

6.1 Compacting Infrastructure Domain

The analysis regarding CPCs' modifications in the infrastructure domain is based on the ARM prototypes, given that the firmware of Intel TDX and AMD SEV is not open-source.

Normal CPCs without CPTI do not introduce any modifications to the infrastructure domain. For maintenance operations that require the SeCPC support, our prototype introduces only 242 LOCs to the RMM. These modifications enable the RMM to configure the hvCPUs corresponding to SeCPCs with isolated stage-2 page tables. Compared to the RMM with 15K LOCs we used, and the version with only 3.2K LOCs which was formally verified in previous work [83], the modification introduced by CPTI is relatively small. In contrast, with the implementation of two simple maintenance operations, *CPC-Snapshot* and *CPC-SecureLog*, the guest code increased by 1,749 LOCs, which is 7.23× more than the modification of CPTI in RMM. Obviously, if more and more maintenance operations are implemented in the infrastructure domain, it will seriously inflate its TCB, while the SeCPC support is a one-time modification.

6.2 Isolation between Host and Guest

The security of the CPC design is ensured not only by maintaining a compact infrastructure domain but also by strictly maintaining a clear security boundary between the host and guest. This design philosophy aligns with original CVM solutions, ensuring that the security of the host and guest is not degraded. In the following sections, we will analyze the security of each party separately, considering potential risks from both the host and guest perspectives.

Guest Security. We acknowledge that neither the CPC design nor the infrastructure-based approaches can eliminate the impacts of vulnerabilities in maintenance modules on the guest. However, our design offers the following four security advantages. First, compared to the size of the guest OS, most maintenance modules are relatively small, especially after being optimized as discussed in §4.3. Second, flexible CPCs can be updated and fixed more quickly than inflexible infrastructure domains. Third, CPTI can be used not only to protect SeCPCs but also to limit CPC's access to the guest OS. This isolation helps minimize the impact of CPCs on the guest OS in the event of errors. Finally, this design provides guests with the flexibility to choose the necessary maintenance functions without bloating their TCBs for a large number of maintenance modules uniformly provided by the infrastructure domain. This makes them immune to non-essential maintenance module vulnerabilities.

On the other hand, CPCs do not introduce new immature guest-host interaction mechanisms. Instead, they utilize the mature hypercall mechanism to notify the host upon completion of maintenance tasks. This process only exports the semantic of task completion to the host, without exposing any additional internal CVM states.

Host Security. Based on the ARM platform prototype, most of the newly introduced code base, including the CPC monitor, resides in the user-level VMM. The host kernel only needs to forward CPC-related hypercalls to the user-level VMM and provide additional memory to the RMM to support the CPTI technology, which modifies less than 150 LOCs. Even after accounting for the code changes of the infrastructure domain in §6.1, the impact of CPCs on the host's TCB is negligible.

From another perspective, the host kernel does not record the hvCPU and maintenance semantic information registered by the guest. In its view, all hvCPUs and gvCPUs are consistent with the behavior of traditional vCPUs. This consistency also guarantees the same level of security as the original CVMs.

6.3 Resilience of SeCPCs

We developed two maintenance operations, *CPC-Snapshot* and *CPC-SecureLog*, using SeCPCs on both AMD SEV and ARM CCA platforms to evaluate the resilience of SeCPCs in the event of guest OS errors. After configuring these SeCPCs during the CVM startup, we intentionally tampered with the internal data and code within them from the guest kernel. As a result, the attacks were mitigated and the SeCPCs worked correctly as expected. *CPC-SecureLog* correctly recorded and protected the security logs, while *CPC-Snapshot* successfully encrypted and exported the internal critical data. This experiment highlights the resilience of SeCPCs and their potential for more CVM security tools. Additionally, by implementing SeCPCs on the two significantly different CVM platforms, the compatibility and practicality of our design are further validated.

7 Performance Evaluation

In this section, we will compare the CPC-based use cases with existing solutions on AMD platforms to answer the fol-

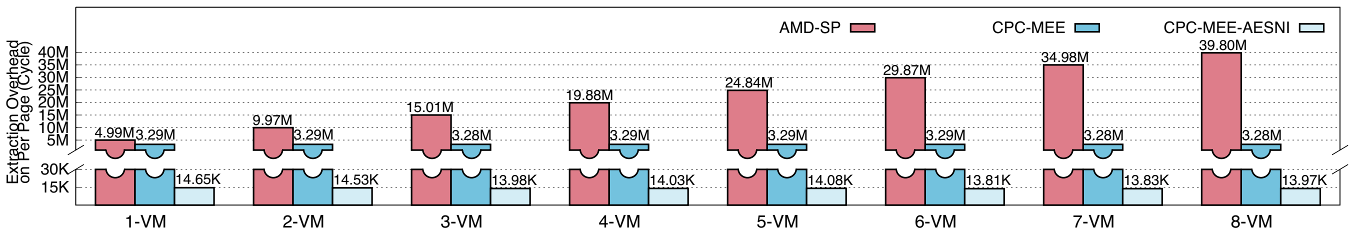

Figure 6: Encrypted Memory Extraction. The number of cycles required to cryptographically extract each memory page. Lower is better. "AMD-SP" refers to all CVMs simultaneously encrypting and extracting guest memory via the AMD-SP. "CPC-MEE" involves using the MEE operator in *CPC-Snapshot* to encrypt and extract memory, the encryption algorithm is AES-CFB. "CPC-MEE-AESNI" stands for using the AES-NI extension on x86 to further accelerate the AES used by *CPC-Snapshot*.

lowing questions:

- From the micro-operation perspective, what is the performance advantage of CPCs compared to existing solutions limited by the AMD-SP?
- In resource reclamation scenarios, how much performance improvement can CPC-based use cases achieve compared to the current scheme by reducing resource contention with the guest workloads?
- In live migration scenarios, what is the performance improvement of a CPC-based solution compared to the existing AMD-SP-based solution?

7.1 Experimental Setup

We conducted experiments on an AMD platform with the SEV-SNP feature, which is equipped with two AMD EPYC 7T83 64-core CPUs, 502 GB of DRAM, and a 1024 GB SSD. To demonstrate the performance and scalability benefits of CPCs compared to existing schemes, we present three case studies.

First, we compare *CPC-Snapshot* with the current firmware-based memory transferring scheme to demonstrate the significant performance limitations of the original approach and highlight the scalability advantages of the CPC design. (§7.2)

Next, we evaluate *CPC-Ballooning* against the virtio-balloon in existing SEV CVMs to illustrate the efficiency of CPCs in resource reclamation scenarios alongside busy guests. (§7.3)

Finally, we examine *CPC-LiveMigration* in comparison to the current firmware-based solution, emphasizing its advantages in live migration scenarios. (§7.4)

7.2 Case Study: Confidential Data Extraction with CPC-Snapshot

Guest memory extraction is the fundamental primitive on which most VM maintenance operations rely, such as migration, snapshot, and security scanning. *CPC-Snapshot* achieves this primitive via the Memory Encryption Extraction (MEE) operator as listed in Table 2, whereas current SEV CVMs require AMD-SP for encryption extraction of internal protected memory. We conducted a performance comparison on CVMs with 1 vCPU and 2 GB of RAM, as depicted in the first data set of Figure 6. The results indicate that *CPC-Snapshot*, utilizing the AES-CFB algorithm for memory encryption, achieves a 34.07% faster performance compared to the current scheme in a single CVM scenario. The improvement is mainly due to the performance advantage of a host processor compared to the AMD-SP as a coprocessor. It could be better because our implementation of the AES-CFB algorithm is simple for minimizing the code base for this SeCPC. We then utilize the AES-NI extension on x86 to accelerate it, and the performance advantage of *CPC-Snapshot* reaches up to 340.61× that of the baseline.

To further explore the impact of the multi-tenant environment on the cloud, we increased the number of CVMs. As illustrated in the 2nd to 8th data sets of Figure 6, the current scheme experiences significant performance degradation due to contention for the use of the AMD-SP as the number of CVMs increases. The overhead required to extract each page on average is proportional to the number of CVMs. In contrast, both the software and hardware-accelerated versions of *CPC-Snapshot* demonstrate excellent scalability.

7.3 Case Study: Resource Reclamation with CPC-Ballooning

We enabled the virtio-balloon in SEV CVMs as the baseline and further transplanted the frontend into *CPC-Ballooning*. The CVMs of the baseline and *CPC-Ballooning* both had 1 vCPU and 8GB of memory, and we compared the speed of collecting 1GB of guest internal free memory. First, we executed and exited the "memhog" process in the guest to ensure that at least 4GB of free memory was available for host reclamation [55]. Then, we performed CPU-intensive decompression calculations to make the guest busy and recorded their time consumption of reclaiming the 1GB of memory. The results, presented in the first set of data in Figure 7, reveal that although the baseline performs similarly to *CPC-Ballooning* when the guest is idle, it is 3.05× slower than *CPC-Ballooning* when the guest is busy. This is due to the fact that the baseline has severe CPU resource competition with the guest workload, whereas the CPC enables the host to provide separate CPU resources, which accelerates the maintenance modules. We will further discuss the situation when host resources are insufficient in §8.

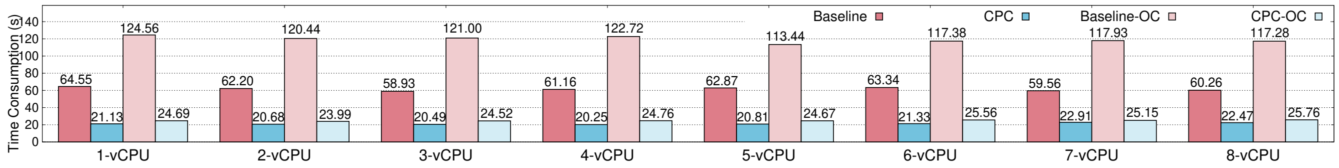

Figure 7: Time consumption for memory reclamation. "Baseline" represents the current CVM, "CPC" represents the CVM with the *CPC-Ballooning*, and "OC" represents two busy CVMs in a 2× overcommitment environment.

In Figure 8, we recorded the performance of two guest workloads, *Untar* and *Sysbench-Thrd* [49], under the ballooning operation. Even though *CPC-Ballooning* improves the speed of free memory reclamation, there is still a significant performance degradation of the guest workloads due to the stress the ballooning operation puts on the guest's memory management and interrupt subsystem. We can see that maintenance modules tightly coupled to the guest OS may lead to substantial performance impacts on the guest even after obtaining independent physical resources. However, the total completion of guest workloads is still improved over the same time period for the shorter reclaim duration.

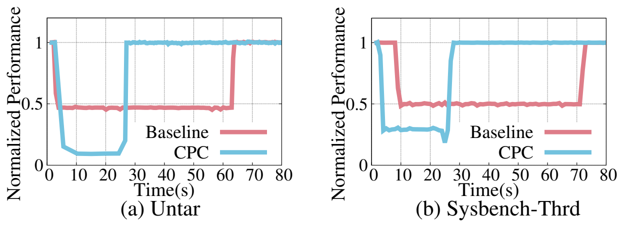

Figure 8: CPC's impact on guest workloads. Figure (a) and Figure (b) show the performance of a guest running *Untar* and *Sysbench-Thrd*, respectively, under the ballooning operation.

We then tested in busy guests with different numbers of vCPUs and observed that the baseline performance did not improve significantly, and thus CPC maintains its advantage by a factor of about 3. We further explored the performance comparison between the two solutions in the scenario where the cloud vendor overcommitted vCPUs. As depicted by the two light-colored bars in Figure 7, the reclamation speed of the baseline decreases by a factor of two in the scenario of 2× vCPU overcommitment. This is because the frontend of the baseline halves in performance as the amount of guest vCPU resources halves. In contrast, the *CPC-Ballooning* maintains stable performance, thereby increasing the performance advantage to approximately 5×.

7.4 Case Study: CVM Live Migration with CPC-LiveMigration

We utilized *CPC-LiveMigration* to free the SEV live migration from AMD-SPs' restrictions. We tested this on SEV CVMs with 1 vCPU and 2 GB of memory and recorded the total time for the live migration. To minimize instability caused by the network, the source instance and destination instance ran on the same machine. All vCPUs of the instances were bound to different physical cores on the same NUMA node. Our baseline is traditional VMs, and the opponent is the current solution based on AMD-SPs (vanilla). The results, as depicted in Figure 9, show that free from the limitations of the AMD-SP, *CPC-LiveMigration* achieves a 55.90× performance improvement compared to the existing approach. This improvement comes mainly from our efficient AES-GCM software implementation ported from *mbedtls* [39] making our operators (MEE and MDI in Table 2) nearly 60 times faster than the AMD-SP.

Although *CPC-LiveMigration* has greatly improved the migration efficiency, there is still a 35.53× overhead from the baseline. We hypothesized that this gap came from the overhead of encryption and decryption. To address this, we accelerated the MEE and MDI operators with the AES-NI extension again, but the improvement was only 19.54%, with the 28.59× overhead from the baseline. The further breakdown of the AES-GCM algorithm used by *CPC-LiveMigration* revealed that the overhead lay primarily within the GCM encoding, not the AES. Therefore acceleration for AES does not provide further significant performance improvement. To investigate the upper bound on migration performance for the current CVM architecture, we replaced the AES-GCM algorithm with a simple *memcpy* operation, thus eliminating the additional overhead due to cryptography and encoding. This resulted in a further speedup of 43.72% and reduced the gap with normal VMs to 16.09×.

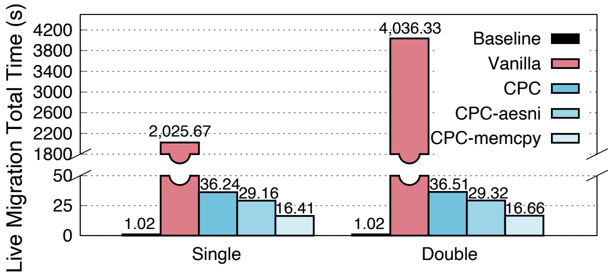

Figure 9: Time consumption for live migration. "Baseline" represents the time consumed for traditional VMs, "Vanilla" represents the time consumed for current SEV CVMs, "CPC" represents the time consumed for *CPC-LiveMigration*, "CPC-aesni" represents accelerating *CPC-LiveMigration* with the AES-NI extension. "CPC-memcpy" represents accelerating *CPC-LiveMigration* without data transformations. "Single" represents the time consumed when one instance is migrated. "Double" represents the time consumed when two identical instances are migrated simultaneously.

Based on these findings, we can summarize that *CPC-LiveMigration* significantly improves the speed of SEV live migration. However, there is still overhead compared to traditional virtualization. Cryptographic and encoding algorithms are the primary source of the overhead. Even assuming that the hardware can accelerate these two steps to close to the overhead of a simple memory copy, there still remains a non-negligible gap compared to traditional VMs that require no extra copies. Future research should explore more methods, such as multi-threading and pipeline, to further accelerate it. In addition, current SEV firmware interfaces do not support *post-copy* live migration [69], which could be supported by *CPC-LiveMigration* for further acceleration.

We further conducted speed tests for 2-instance migration. The results demonstrate that the vanilla experiences a significant drop in performance, nearly doubling the time consumption. However, *CPC-LiveMigration* maintains its efficiency, achieving a remarkable speedup of 137.66× with AES-NI acceleration. The substantial decrease in vanilla performance aligns with the inverse trend illustrated in Figure 2b of §2.3. In real-world scenarios, where numerous CVM instances are present on such 128-core platforms, this performance degradation becomes more serious. This further highlights the substantial performance advantage of CPCs.

We also tested the throughput of guest workloads during the live migration to explore the impact on guest workload. As shown in Figure 10, no significant performance degradation ($< 9\%$) was observed on both the vanilla and CPC versions compared to the baseline. This is attributed to the high degree of decoupling between the CPC code logic and the guest OS, resulting in minimal interference with each other's performance.

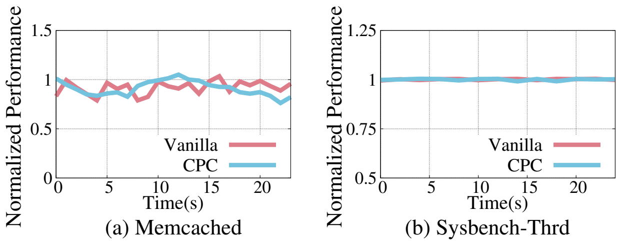

Figure 10: Normalized throughput of guest workloads under live migration. Figure (a) and Figure (b) show the performance of a guest running *Memcached* [40] and *Sysbench-Thrd*, respectively, under the live migration operation.

8 Discussion and Limitations

Compatibility for more secure VMs. In addition to the previously mentioned platforms, our design can be adapted to more CVM platforms [21, 36, 43, 74, 75, 80, 81]. This adaptation relies solely on the ***host's scheduling of vCPUs*** and ***trusted S2PTs*** (or AMD VMPL). For example, Intel TDX [36] offers shared S2PTs and secure S2PTs, with the TDX module protecting only the secure S2PTs. When implementing CPTI, the TDX module must ganrantee the security of SeCPCs by the secure S2PTs. This ensures that in the event of a collaboration between the host and a compromised guest to attack SeCPC data through shared S2PTs, their attempt will fail due to the distinct memory keys for secure S2PTs and shared S2PTs. Furthermore, we observed that AP-TEE (CoVE) [21], an emerging CVM approach on RISC-V, also employs host-controlled vCPU scheduling and trusted S2PTs. Consequently, our design can be tailored to this platform with a few modifications to the TEE Security Manager (TSM). It is essential to note that CPC presents unique advantages to each CVM platform. For instance, our prototype achieved remarkable performance improvements on AMD platforms, whereas primarily focused on enhancing firmware security and the intra-CVM isolation for ARM CCA platforms.

Moreover, our design relocates more maintenance modules from the hypervisor to the guest, benefiting secure virtualization schemes that are influenced by the size of the in-kernel hypervisor [60, 87, 88, 95, 97, 100, 102].

Further technological route comparisons. We investigated alternative solutions, all of which present notable limitations for CVMs. Delegating services in other CVMs [50, 71, 82, 86, 105] would rapidly deplete the allowable instances on AMD and Intel platforms with hardware restrictions on CVM quantity [3, 45]. Additionally, eBPF-based services are closely coupled to the guest OS, making isolation challenging. Placing guest code snippets to the host for execution, as with hyperupcalls [55], results in potential guest code logic exposure. Even if host access to such snippets is regulated through hardware modifications, it could compromise the well-defined security boundaries of CVMs, introducing more complexity and potential development errors. For example, the host could gather additional information from the guest code snippets, compromising the guests' KASLR through the pointers to critical guest kernel functions and structures. Furthermore, hardware modifications are impractical given the slow hardware development of the CVM platforms.

Isolation between SeCPCs. While in accordance with the threat models presented in this paper, all SeCPCs are TCBs of cloud tenants and the isolation between them is deemed non-essential, our design can still be extended to scenarios where SeCPCs inherently mistrust each other. Specifically, each critical maintenance module in an SeCPC can be isolated from other SeCPCs by using CPTI based on its dedicated S2PTs. On AMD platforms, an SeCPC monitor can be placed in CPL0 of VMPL0 and all the SeCPCs can be isolated in VMPL1 or in CPL3 of VMPL0.

Overhead of SeCPCs. When the AMD VMPL feature is not present, the CPTI technology must rely on trusted S2PTs. This design introduces additional S2PT switching compared to traditional VMs, potentially leading to overhead. Given the unavailability of ARM CCA hardware, we approximate this based on two prior studies, TxIntro [85] and Black-Box [70], which employ multiple S2PTs for isolated software within a single VM. As these studies demonstrate minimal overhead, we anticipate that the overhead associated with SeCPCs will also be minimal.

Reactive services from guests. CPC-based services can be invoked not only by the host but also by the guest. There are two methods to implement such services. First, the guest actively notifies the host via hypercalls to schedule specific hvCPUs to trigger the corresponding services. Second, the trusted firmware provides more interfaces for the guest to configure the handling of specific events. For instance, the guest can utilize the trusted firmware to revoke the write permission of particular protected data from the S2PT, causing a page fault when an internal misbehaving component attempts to change the data and is intercepted by the trusted firmware. Subsequently, the trusted firmware can redirect the execution flow to a designated guest handler, which will then invoke the appropriate CPC to handle the event based on the tenant's pre-setting.

Resource isolation. CPCs offer isolated CPU resources for maintenance modules, and SeCPCs can additionally provide memory isolation. However, in certain scenarios, maintenance modules may need to leverage the internal data structure and semantics of the guest OS. Consequently, they cannot be completely isolated from the guest workloads, as illustrated in Figure 8.

Flexible resource allocation. CPCs can be applied to both CVMs and traditional VMs to achieve flexibility in adjusting guests' resources. This can be extended to scenarios such as serverless computing [6, 37, 42, 61, 73, 84], online resource scheduling [64,79,93,101,104], and harvest VMs [54,63,98, 103]. Its potential for higher execution density and resource utilization is worth exploring in future work.

Extra CPUs. CPCs require additional vCPU threads, which could pose challenges when host resources are scarce. However, this is an infrequent situation as the additional CPU usage by CPCs is typically brief. The CPUs occupied by CPCs are relinquished once the maintenance tasks are finished. Meanwhile, in such situations, the hvCPU can still acquire computing power by obtaining CPU time slices from the gvCPUs of the same CVM. This merely downgrades the performance to original schemes with resource contention.

Support for security tools. The CPTI technology has the ability to defend against compromised guest kernels in emergencies. Therefore, it has the potential to be further developed into more useful security tools for CVMs like debugging tools. It is reasonable to expect the tenants who choose CVMs to have higher security requirements, and such use cases will be explored in future work.

9 Related Work

Transformed vCPU abstraction. Several works novelly transform the vCPU abstraction to address various system issues [23,33,57,58,66,85,96]. gVisor [33], Enarx [23], Enclavisor [66], and Song et al. [96] transform vCPUs into user-level application threads to further meet the requirements for secure containers, efficient scheduling, and secure runtimes. Dune [57] uses virtualization hardware to provide a process, rather than a machine abstraction. It improves the performance of certain applications while ensuring isolation. Based on Dune, IX [58] further places the network application logic as the data plane in the guest, and exports the resource management as the control plane left to the host. CPCs transform the vCPU abstraction into the procedure abstraction, with the control plane for the host and the data plane hiding in the guest.

In-TEE security. To mitigate the potential vulnerabilities posed by the huge code base in TEE, several works focus on reducing the TCB or implementing internal isolation. SCONE [56] achieves TCB reduction by eliminating the LibOS and protects particular syscalls using a shim C library. Meanwhile, Occlum [94] shrinks the TCB by isolating applications within the same enclave through SFI. Lightenclave [67] proposes a hardware extension that prevents the host from tampering with the page table, and it employs the Intel MPK to establish efficient and secure internal isolation within SGX based on this extension. For CVMs, the huge code base of the guest kernel also could not be ignored for practicality. SeCPCs, enabled by CPTI, effectively safeguard specific maintenance components during emergencies.

In-VM maintenance. Some previous work explored the technological route of in-VM maintenance. Apart from the solutions mentioned earlier based on nested virtualization [51, 65], VEIL [53] also utilizes the VMPL on AMD SEV-SNP platforms to protect critical system services. They all rely on specific CVM platforms. Additionally, TxIntro [85], similar to CPCs, utilizes in-VM core planting to introduce an implanted core running VMI code into the guest VM's space. However, its internal isolation relies on trusted hosts, which is not allowed in CVM. Moreover, the lack of semantics makes implanted cores to be continuously scheduled like normal vCPUs, leading to reduced resource utilization.

10 Conclusion

This paper introduces Confidential Procedure Calls (CPCs), a novel approach to enhancing the flexibility, security, and efficiency of CVM maintenance by bridging the semantic gap between the host and in-CVM maintenance modules. Our prototypes implemented on the leading CVM platforms demonstrate that the design achieves all the three objectives. Perhaps this brick of maintenance patched by CPCs can improve the practicality of CVMs and facilitate their large-scale popularization on the cloud.

11 Acknowledgments

We sincerely thank the anonymous reviewers for their insightful suggestions. This work was partially supported by National Key R&D Program of China (2023YFB4503702) and NSFC (No. 62372287 and 61925206).

References

[1] Alibaba Security White Paper International Edition. https://resource.alibabacloud.com/whitepaper/alibaba-cloud-security-whitepaper---international-edition-v20-2020_1717. Referenced December 2023.

[2] Amazon CloudWatch Pricing. https://aws.amazon.com/cn/cloudwatch/pricing/. Referenced December 2023.

[3] AMD SEV Secure Nested Paging Firmware ABI Specification. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf. Referenced May 2024.

[4] AMD SEV-SNP: Strengthening VM Isolation with Itegrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf. Referenced May 2023.

[5] AMD SVSM. https://github.com/AMDESE/linux-svsm. Referenced May 2023.

[6] Apache OpenWhisk: Open Source Serverless Cloud Platform. https://openwhisk.apache.org/. Referenced May 2024.

[7] ARM CCA Hardware Architecture. https://developer.arm.com/documentation/ddi0615/latest/. Referenced May 2023.

[8] ARM CCA: Realm Management Monitor specification. https://developer.arm.com/documentation/den0125/0300/Arm-CCA-Software-Architecture. Referenced January 2024.

[9] AWS CloudWatch agent. https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html. Referenced December 2023.

[10] AWS CodeDeploy agent. https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent.html. Referenced December 2023.

[11] AWS Hibernation. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Hibernate.html. Referenced December 2023.

[12] AWS Inspector agent. https://docs.aws.amazon.com/inspector/v1/userguide/inspector_agents.html. Referenced December 2023.

[13] AWS Monitor agent. https://learn.microsoft.com/en-us/azure/azure-monitor/agents/agents-overview. Referenced December 2023.

[14] AWS Pipelines agent. https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=yaml%2Cbrowser. Referenced December 2023.

[15] Azure Backup MARS agent. https://learn.microsoft.com/en-us/azure/backup/install-mars-agent. Referenced December 2023.

[16] Azure Log analytics agent. https://learn.microsoft.com/en-us/azure/azure-monitor/agents/log-analytics-agent. Referenced December 2023.

[17] Azure Migrate pricing. https://azure.microsoft.com/en-us/pricing/details/azure-migrate/. Referenced December 2023.

[18] Azure Site Recovery. https://learn.microsoft.com/en-us/azure/site-recovery/site-recovery-overview. Referenced December 2023.

[19] Azure: What is cloud migration? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-migration. Referenced December 2023.

[20] COCONUT SVSM. https://github.com/coconut-svsm/svsm. Referenced May 2023.

[21] Confidential VM Extension (CoVE) on RISC-V. https://github.com/riscv-non-isa/riscv-ap-tee. Referenced March 2024.

[22] CVE of Linux. https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux. Referenced January 2024.

[23] Enarx. https://enarx.dev/docs/technical/syscall-proxy. Referenced December 2023.

[24] Free Cloud Migration Services on AWS. https://aws.amazon.com/free/migration/. Referenced December 2023.

[25] Google Cloud: Create and manage disk snapshots. https://cloud.google.com/compute/docs/disks/create-snapshots. Referenced December 2023.

[26] Google Cloud: Live migration process during maintenance events. https://cloud.google.com/compute/docs/instances/live-migration-process. Referenced December 2023.

[27] Google Cloud Logging Agent. https://cloud.google.com/logging/docs/agent/logging. Referenced December 2023.

[28] Google Cloud Monitoring Agent. https://cloud.google.com/monitoring/agent/monitoring. Referenced December 2023.

[29] Google Cloud Ops Agent. https://cloud.google.com/monitoring/agent/ops-agent. Referenced December 2023.

[30] Google Cloud: Price of Cloud Logging. https://cloud.google.com/stackdriver/pricing#logging-costs. Referenced December 2023.

[31] Google Cloud: Price of Cloud Monitoring. https://cloud.google.com/stackdriver/pricing#monitoring-costs. Referenced December 2023.

[32] Granule Protection Tables in TF-A. https://www.trustedfirmware.org/docs/tfa_tech_forum_2021_10_21_gpt.pdf. Referenced December 2023.

[33] gvisor. https://gvisor.dev/. Referenced December 2023.

[34] INTEL-SA-01036. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01036.html. Referenced May 2024.

[35] Intel Trust Domain Extensions (Intel TDX) Module Base Architecture Specification. https://cdrdv2-public.intel.com/795471/intel-tdx-module-1.5-base-spec-348549003.pdf. Referenced December 2023.

[36] Intel® Trust Domain Extensions (Intel® TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Referenced May 2023.

[37] Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications. https://knative.dev/docs/. Referenced May 2024.

[38] Live Migration on AMD SEV. https://github.com/AMDESE/qemu/tree/sev_live_migration_v4. Referenced December 2023.

[39] Mbed TLS. https://github.com/Mbed-TLS/mbedtls. Referenced January 2024.

[40] Memcached. https://memcached.org/. Referenced January 2024.

[41] Memory ballooning. https://en.wikipedia.org/wiki/Memory_ballooning. Referenced December 2023.

[42] OpenFaas: Serverless Functions, Made Simple. https://www.openfaas.com/. Referenced May 2024.

[43] pKVM of AVF architecture. https://source.android.com/docs/core/virtualization/architecture. Referenced December 2023.

[44] Realm Management Monitor specification. https://documentation-service.arm.com/static/6361431cc5a70d2cdb15fe1b?token=. Referenced December 2023.

[45] Secure Encrypted Virtualization API Version 0.24. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/programmer-references/55766_SEV-KM_API_Specification.pdf. Referenced December 2023.

[46] Security Center agent of Alibaba Cloud. https://www.alibabacloud.com/help/en/security-center/user-guide/install-the-security-center-agent. Referenced December 2023.

[47] SEV Secure Nested Paging Firmware ABI Specification. https://www.amd.com/system/files/TechDocs/56860.pdf. Referenced December 2023.

[48] Support for Arm CCA VMs on Linux. https://lwn.net/Articles/921482/. Referenced January 2024.

[49] Sysbench. https://github.com/akopytov/sysbench. Referenced January 2024.

[50] TD Migration Architecture Specification. https://cdrdv2-public.intel.com/733578/intel-tdx-module-1.5-td-migration-spec-348550002.pdf. Referenced December 2023.

[51] TDX Module TD Partitioning Architecture Specification. https://www.intel.com/content/www/us/en/content-details/773039/intel-tdx-module-v1-5-td-partitioning-architecture-specification.html. Referenced December 2023.

[52] VMWare: Take a Snapshot of a Virtual Machine. https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.vm_admin.doc/GUID-9720B104-9875-4C2C-A878-F1C351A4F3D8.html. Referenced December 2023.

[53] Adil Ahmad, Botong Ou, Congyu Liu, Xiaokuan Zhang, and Pedro Fonseca. Veil: A protected services framework for confidential virtual machines. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ASPLOS '23, page 378–393, New York, NY, USA, 2024. Association for Computing Machinery.

[54] Pradeep Ambati, Iñigo Goiri, Felipe Vieira Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing slos for resource-harvesting vms in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 735–751. USENIX Association, 2020.

[55] Nadav Amit and Michael Wei. The design and implementation of hyperupcalls. In Haryadi S. Gunawi and Benjamin C. Reed, editors, *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pages 97–112. USENIX Association, 2018.

[56] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. SCONE: secure linux containers with intel SGX. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 689–703. USENIX Association, 2016.

[57] Adam Belay, Andrea Bittau, Ali José Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis. Dune: Safe user-level access to privileged CPU features. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 335–348. USENIX Association, 2012.

[58] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In Jason Flinn and Hank Levy, editors, *11th*

USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014, pages 49–65. USENIX Association, 2014.

[59] Robert Buhren, Shay Gueron, Jan Nordholz, Jean-Pierre Seifert, and Julian Vetter. Fault attacks on encrypted general purpose compute platforms. In Gail-Joon Ahn, Alexander Pretschner, and Gabriel Ghinita, editors, *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017*, pages 197–204. ACM, 2017.

[60] Jiahao Chen, Dingji Li, Zeyu Mi, Yuxuan Liu, Binyu Zang, Haibing Guan, and Haibo Chen. Security and performance in the delegated user-level virtualization. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 209–226, Boston, MA, July 2023. USENIX Association.

[61] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 467–481, New York, NY, USA, 2020. Association for Computing Machinery.

[62] Pedro Fonseca, Rodrigo Rodrigues, and Björn B. Brandenburg. SKI: Exposing kernel concurrency bugs through systematic schedule exploration. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 415–431, Broomfield, CO, October 2014. USENIX Association.

[63] Alexander Fuerst, Stanko Novakovic, Iñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-harvesting vms in cloud platforms. In Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch, editors, *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, pages 583–594. ACM, 2022.

[64] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: Practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, pages 135–151, New York, NY, USA, 2021. Association for Computing Machinery.

[65] Xinyang Ge, Hsuan-Chi Kuo, and Weidong Cui. Hecate: Lifting and shifting on-premises workloads to an untrusted cloud. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1231–1242. ACM, 2022.

[66] Jinyu Gu, Xinyue Wu, Bojun Zhu, Yubin Xia, Binyu Zang, Haibing Guan, and Haibo Chen. Enclavisor: A hardware-software co-design for enclaves on untrusted cloud. *IEEE Trans. Computers*, 70(10):1598–1611, 2021.

[67] Jinyu Gu, Bojun Zhu, Mingyu Li, Wentai Li, Yubin Xia, and Haibo Chen. A hardware-software co-design for efficient intra-enclave isolation. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3129–3145. USENIX Association, 2022.

[68] Felicitas Hetzelt and Robert Buhren. Security analysis of encrypted virtual machines. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2017, Xi'an, China, April 8-9, 2017*, pages 129–142. ACM, 2017.

[69] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, jul 2009.

[70] Alexander Van't Hof and Jason Nieh. BlackBox: A Container Security Monitor for Protecting Containers on Untrusted Operating Systems. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 683–700, Carlsbad, CA, July 2022. USENIX Association.

[71] Zhichao Hua, Jinyu Gu, Yubin Xia, Haibo Chen, Binyu Zang, and Haibing Guan. vTZ: Virtualizing ARM TrustZone. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 541–556, Vancouver, BC, August 2017. USENIX Association.

[72] Yongzhe Huang, Vikram Narayanan, David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. KSplit: Automating device driver isolation. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 613–631, Carlsbad, CA, July 2022. USENIX Association.

[73] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A berkeley view on serverless computing, 2019.

[74] Dingji Li, Zeyu Mi, Chenhui Ji, Yifan Tan, Binyu Zang, Haibing Guan, and Haibo Chen. Bifrost: Analysis and Optimization of Network I/O Tax in Confidential Virtual Machines. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 1–15, Boston, MA, July 2023. USENIX Association.

[75] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. TwinVisor: Hardware-Isolated Confidential Virtual Machines for ARM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 638–654, New York, NY, USA, 2021. Association for Computing Machinery.

[76] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected I/O operations in amd's secure encrypted virtualization. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1257–1272. USENIX Association, 2019.

[77] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 717–732. USENIX Association, 2021.

[78] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. Tlb poisoning attacks on amd secure encrypted virtualization. In *Proceedings of the 37th Annual Computer Security Applications Conference*, ACSAC '21, page 609–619, New York, NY, USA, 2021. Association for Computing Machinery.

[79] Qian Li, Bin Li, Pietro Mercati, Ramesh Illikkal, Charlie Tai, Michael Kishinevsky, and Christos Kozyrakis. Rambo: Resource allocation for microservices using bayesian optimization. *IEEE Computer Architecture Letters*, 20(1):46–49, 2021.

[80] Shih-Wei Li, John S. Koh, and Jason Nieh. Protecting Cloud Virtual Machines from Hypervisor and Host Operating System Exploits. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*, pages 1357–1374, Santa Clara, CA, August 2019. USENIX Association.

[81] Shih-Wei Li, Xupeng Li, Ronghui Gu, Jason Nieh, and John Zhuang Hui. Formally Verified Memory Protection for a Commodity Multiprocessor Hypervisor. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, pages 3953–3970. USENIX Association, August 2021.

[82] Wenhao Li, Yubin Xia, Long Lu, Haibo Chen, and Binyu Zang. Teev: virtualizing trusted execution environments on mobile platforms. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2019, page 2–16, New York, NY, USA, 2019. Association for Computing Machinery.

[83] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and Verification of the ARM Confidential Compute Architecture. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 465–484, Carlsbad, CA, July 2022. USENIX Association.

[84] Qingyuan Liu, Yanning Yang, Dong Du, Yubin Xia, Ping Zhang, Jia Feng, James R. Larus, and Haibo Chen. Harmonizing efficiency and practicability: Optimizing resource utilization in serverless computing with Jiagu. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, Santa Clara, CA, July 2024. USENIX Association.

[85] Yutao Liu, Yubin Xia, Haibing Guan, Binyu Zang, and Haibo Chen. Concurrent and consistent virtual machine introspection with hardware transactional memory. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*, pages 416–427. IEEE Computer Society, 2014.

[86] Yutao Liu, Tianyu Zhou, Kexin Chen, Haibo Chen, and Yubin Xia. Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1607–1619, New York, NY, USA, 2015. Association for Computing Machinery.

[87] José Martins, Adriano Tavares, Marco Solieri, Marko Bertogna, and Sandro Pinto. Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems. In Marko Bertogna and Federico Terraneo, editors, *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, volume 77 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[88] Zeyu Mi, Dingji Li, Haibo Chen, Binyu Zang, and Haibing Guan. (Mostly) Exitless VM Protection from Untrusted Hypervisor through Disaggregated Nested Virtualization. In Srdjan Capkun and Franziska Roesner, editors, *Proceedings of the 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1695–1712. USENIX Association, 2020.

[89] Mathias Morbitzer, Manuel Huber, and Julian Horsch. Extracting secrets from encrypted virtual machines. In Gail-Joon Ahn, Bhavani Thuraisingham, Murat Kantarcioglu, and Ram Krishnan, editors, *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019*, pages 221–230. ACM, 2019.

[90] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. Severed: Subverting amd's virtual machine encryption. In Angelos Stavrou and Konrad Rieck, editors, *Proceedings of the 11th European Workshop on Systems Security, EuroSec@EuroSys 2018, Porto, Portugal, April 23, 2018*, pages 1:1–1:6. ACM, 2018.

[91] Vikram Narayanan, Abhiram Balasubramanian, Charlie Jacobsen, Sarah Spall, Scott Bauer, Michael Quigley, Aftab Hussain, Abdullah Younis, Junjie Shen, Moinak Bhattacharyya, and Anton Burtsev. LXDs: Towards isolation of kernel subsystems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 269–284, Renton, WA, July 2019. USENIX Association.

[92] Vikram Narayanan, Yongzhe Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. Lightweight kernel isolation with virtualization and vm functions. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '20, page 157–171, New York, NY, USA, 2020. Association for Computing Machinery.

[93] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. Autopilot: Workload autoscaling at google. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[94] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel

SGX. In James R. Larus, Luis Ceze, and Karin Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 955–970. ACM, 2020.

[95] Le Shi, Yuming Wu, Yubin Xia, Nathan Dautenhahn, Haibo Chen, Binyu Zang, and Jinming Li. Deconstructing Xen. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[96] Xiang Song, Jicheng Shi, Haibo Chen, and Binyu Zang. Schedule processes, not vcpus. In *Proceedings of the 4th Asia-Pacific Workshop on Systems*, APSys '13, New York, NY, USA, 2013. Association for Computing Machinery.

[97] Udo Steinberg and Bernhard Kauer. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, page 209–222, New York, NY, USA, 2010. Association for Computing Machinery.

[98] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J. Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. Smartharvest: harvesting idle cpus safely and efficiently in the cloud. In Antonio Barbalace, Pramod Bhatotia, Lorenzo Alvisi, and Cristian Cadar, editors, *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*, pages 1–16. ACM, 2021.

[99] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. Sevurity: No security without integrity : Breaking integrity-free memory encryption with minimal assumptions. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1483–1496. IEEE, 2020.

[100] Chiachih Wu, Zhi Wang, and Xuxian Jiang. Taming Hosted Hypervisors with (Mostly) Deprivileged Execution. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[101] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. *SIGARCH Comput. Archit. News*, 41(3):607–618, jun 2013.

[102] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP '11, page 203–216, New York, NY, USA, 2011. Association for Computing Machinery.

[103] Yanqi Zhang, Iñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and cheaper serverless computing on harvested resources. In Robbert van Renesse and Nickolai Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 724–739. ACM, 2021.

[104] Yunqi Zhang, Michael A. Laurenzano, Jason Mars, and Lingjia Tang. Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 406–418, USA, 2014. IEEE Computer Society.

[105] Shixuan Zhao, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. vsgx: Virtualizing sgx enclaves on amd sev. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 321–336, 2022.