Designing and Implementing Live Migration Support for Arm-based Confidential VMs

Fang-Jie Yang
National Taiwan University
r12922015@ntu.edu.tw

Jian-Lin Li
National Taiwan University
b07902103@ntu.edu.tw

Kaiwen Xue
Carnegie Mellon University
kaiwenx@andrew.cmu.edu

Shih-Wei Li
National Taiwan University
shihwei@csie.ntu.edu.tw

ABSTRACT

Confidential virtual machines (CVMs) are increasingly deployed to protect users' code and data in use against attackers with hypervisor privileges. Given Arm's growing adoption, various implementations have extended commodity hypervisors like KVM to support CVMs on Arm. However, none of these implementations supports live migration, a crucial virtualization feature. This work presents the first design of live migration support for Arm-based CVMs. The design utilizes functionality for migrating regular VMs from commodity hypervisors to simplify development efforts while preserving CVMs' safety. We extended two KVM-based CVM implementations for Arm to prototype the design to support existing live migration approaches from QEMU, including pre-copy, post-copy, and parallel migration.

CCS CONCEPTS

• **Security and privacy → Virtualization and security**.

KEYWORDS

Virtualization, Confidential Computing, Security

ACM Reference Format:
Fang-Jie Yang, Jian-Lin Li, Kaiwen Xue, and Shih-Wei Li. 2024. Designing and Implementing Live Migration Support for Arm-based Confidential VMs. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '24), September 4–5, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3678015.3680488

APSys '24, August 2024, Kyoto, Japan

ACM ISBN 979-8-4007-1105-3/24/09…$15.00
https://doi.org/10.1145/3678015.3680488

1 INTRODUCTION

Confidential virtual machines (CVMs) are increasingly deployed to protect users' code and data in use against attackers with hypervisor privileges. Given Arm's growing adoption, various implementations have been introduced to support CVMs on Arm-based cloud and mobile environments. Due to the lack of CVM support from Arm hardware, earlier work leverages existing features, such as Arm's virtualization extensions (VE) [21, 25, 26] to implement CVMs. Arm has recently introduced hardware extensions, the Confidential Compute Architecture (CCA) [11], to support CVMs. KVM [23] has been extended to support CCA [5].

Live migration of virtual machines refers to transferring running VMs from one physical server to another without disrupting their services. To migrate a VM from a source to a target platform, the hypervisor saves the VM's execution states in the VM's CPU (VCPU) registers, memory, and devices. Then, the source hypervisor sends the saved states to the hypervisor on the target server. The target hypervisor allocates resources based on the specifications of the migrated VM and imports the received states into the VM. Commodity hypervisors, like KVM, provide various migration approaches [15, 18, 29] to satisfy performance demands.

Live migration approaches for regular VMs are incompatible with CVMs. Hypervisors cannot access CVMs' unencrypted states. On x86, the trusted secure module from CVM implementations [9, 19] exposes commands [22, 33] to hypervisors such that the existing live migration mechanisms can be retrofitted to use the commands to migrate CVMs. Others [10, 14, 31] rely on an in-CVM software to support live migration and address the limitations of the secure module-based design in functionality and performance [24]. However, the CVM-based design requires a clean-slate implementation of migration mechanisms.

No existing Arm-based CVM implementations support live migration. This work introduces the first design of live migration support for Arm-based CVMs based on two observations. Arm-based CVM implementations include a trusted secure module to protect CVMs and provide virtualization

features that require access to protected CVM states. These modules are software-based and extensible [2, 3, 7] for additional functionality. Further, current Arm-based implementations extended commodity hypervisors, like KVM [21, 25–27] to reduce development efforts. Therefore, instead of adopting a clean-slate CVM-based approach, our design extends the CVM's secure module for Arm to expose live migration support to hypervisors. The secure module leverages cryptography and provides a migration API that consists of commands to export and import encrypted CVM states. The existing VM live migration mechanisms from hypervisors like KVM can be repurposed and modified to execute commands for migrating CVMs while maintaining safety. We ensure the design applies to Arm-based CVMs with different hardware requirements to facilitate real-world deployment.

We prototyped the proposed design in two KVM-based CVM implementations for Arm that utilize different hardware features. Specifically, we modified the secure module from SeKVM [2, 26] and Arm's CCA architecture [7] to implement the migration commands. Additionally, we extended KVM and QEMU [13] incorporated by SeKVM and CCA to invoke these commands. Our prototype supports migrating CVMs with various approaches from KVM, including pre-copy [15], post-copy [18], and parallel migration [29].

This work makes the following contributions.

- We introduced the first design for CVM live migration on Arm. Our secure module-based design simplifies implementation efforts and facilitates commodity mechanisms.
- We implemented live migration support for CVMs supporting the existing and future Arm hardware with CCA.

2 BACKGROUND

CVMs CVM frameworks rely on a trusted secure module, i.e., the TCB, to protect the confidentiality and integrity of CVMs' CPU and memory states against the untrusted hypervisor. The hypervisor, which could include a full host OS kernel in its codebase, is delegated with resource allocation but is deprived of accessing unencrypted data from resources assigned to CVMs. CVMs take an end-to-end approach to protect while reusing hypervisors' I/O virtualization functionality. CVMs may share their memory with the hypervisor for paravirtualized I/Os [28].

VE-based CVMs. Due to the lack of hardware support for CVMs on Arm, earlier works leverage Arm's virtualization extensions (VE) [21, 25, 26] and rely on a secure module to protect CVMs from the untrusted hypervisor. The secure module runs in Arm's EL2 mode to deprivilege the hypervisor in a less privileged EL1 mode. The module saves VCPU registers to a private memory that the hypervisor cannot access. It leverages Arm's nested page tables (NPT), stage 2 page tables (S2PTs), which translate Intermediate Physical Addresses (IPAs) to machine physical addresses, to restrict the hypervisor's memory access. It manages S2PTs for each CVM and the hypervisor. The secure module unmaps CVM's private memory from the hypervisor's S2PT. It grants the hypervisor access to CVMs' shared memory.

CCA-based CVMs. Arm recently introduced hardware extensions called Realm Management Extensions (RME) for the Confidential Compute Architecture (CCA) [11] to support CVMs, called Realms. The RME hardware checks each memory access against a Granule Protection Table (GPT). It ensures that the hypervisor cannot access a 4KB page (granule) delegated to Realms. Arm CCA incorporates a trusted secure module, Realm Management Monitor (RMM). The RMM interposes Realm enters/exits and saves the Realm's CPU execution context in a Realm Execution Context (REC) stored in a Realm's granule. The RMM exposes a Realm Management Interface (RMI) to the hypervisor to manage and create Realms, delegating a granule to a Realm, and mapping a granule to a Realm's S2PT called the Realm Translation Table (RTT). CCA divides Realm's IPAs into a protected and unprotected region, the later can be used for paravirtualized I/Os. CCA could optionally encrypt Realms's memory at runtime using a per-Realm secret key.

Live Migration of Virtual Machines. Commodity hypervisors like KVM incorporate a user space Virtual Machine Monitors (VMM), such as QEMU, to support VM live migration. The primary source of overhead of live migration is the transfer of VM memory. A naive migration approach that pauses the VM and transfers all its states could result in long VM downtime, i.e., the time when the VM is unresponsive.

Pre-copy [15] first migrates all CVM's memory and then dirty pages in subsequent iterations. Pre-copy pauses the source VM when the number of dirty pages in an iteration converges within a set threshold. The hypervisor then transfers the VM's dirty pages and remaining states to the target to resume the VM. Parallel migration [29], such as QEMU's MultiFD [4], leverages multi-threading to speed up dirty page transfers in pre-copy to optimize the throughput of pre-copy. To support MultiFD, QEMU spawns *sending threads* that run in parallel at the source to transfer dirty pages to respective *receiving threads* at the target. Post-copy [18] aims to reduce the migration downtime from pre-copy. Post-copy pauses the source VM when migration begins then transfers all VM states except memory to resume the VM at the target. It migrates memory states on demand from the source upon NPT faults of the target VM. To support post-copy, QEMU creates a monitor thread that requests the faulting page's contents from the source. The target hypervisor maps the VM's NPT to a newly allocated page that contains the imported contents to address the NPT fault.

3 THREAT MODEL

We consider attackers who aim to compromise the confidentiality and integrity of a CVM's states, encompassing its code and data stored in private memory or CPU registers. The attacker could exploit bugs from the hypervisor or VMMs, such as QEMU, on the host to obtain privileges to compromise CVMs. We assume the attacker does not have physical access to the machine hosting CVMs;, physical attacks [17] are excluded from the threat model. Attacks against availability and side-channel attacks are out of scope.

4 DESIGN

We discuss the requirements our design intends to achieve. Next, we introduce the concrete design and the API exposed to support live migration of Arm-based CVMs.

R1: Support live migration approaches from commodity hypervisors. Current Arm-based CVM implementations extend commodity hypervisors like KVM. To simplify implementation and maintenance efforts, our design should leverage the live migration functionalities provided by these hypervisors and ensure seamless integration and operation. For instance, the API should support write protection of CVMs' memory access required by pre-copy to track dirty pages.

R2: Preserve the safety of CVMs. The design should retain the confidentiality and integrity of CVMs while relying on hypervisors' migration functionalities. It should prevent the leakage of CVM secrets during state export and the tampering of migrated states. In addition, the design should ensure the integrity of the CVM's configuration metadata, including page sharing and mapping information.

R3: Achieve compatibility with Arm CVM implementations. The design should apply to CVMs with different hardware requirements. As of the date of writing, no existing Arm hardware implements CCA. The design must be compatible with VE-based CVMs that support the current Arm hardware. It should also support CCA and ensure backward compatibility when hardware becomes available.

4.1 Extending CVM Implementations

We propose to extend the secure module of Arm-based CVMs to expose live migration API commands listed in Table 1. We ensure the API supports hypervisors that host Arm-based CVMs, such that the hypervisors' live migration approaches for regular VMs can retrofitted with modest effort (**R1**) to invoke the commands to migrate CVMs while retaining their safety (**R2**) throughout the migration process. To achieve (**R3**), our design supports CCA and VE-based CVMs.

To incorporate the proposed migration design, the secure module should be extended with the support of Authenticated Encryption (AE). The secure module uses AE to encrypt the VM states exported to an untrusted hypervisor while protecting their integrity. For example, cryptographic schemes such as AES-GCM [16] could be adopted to export encrypted CVM states and a respective message authentication code (MAC). Both are sent to the target platform for imports and consumed for decryption. AES-GCM detects tampering if the MAC does not match the supplied ciphertext.

4.2 Live Migration API

Table 1: CVM Live Migration API Commands

API Commands	Description
MigStart & MigTerm	**MigStart** to create a migration session. **MigTerm** terminates a migration session.
MigTransKeyGen	The src and dest make **MigTransKeyGen** to establish a shared migration transport key, *TPK*, using Diffie-Hellman.
MigSessionKeyGen	The src uses **MigSessionKeyGen** to create a per migration session key, *MSK*. The command encrypts the *MSK* with the *TPK* and returns the ciphertext.
MigKeyImpt	The dest calls **MigKeyImpt** to decrypt the encrypted *MSK* sent from the src using the *TPK* and import the *MSK*.
ExptVCPU/ImptVCPU	**ExptVCPU** exports the encrypted VCPU states from the CVM. **ImptVCPU** imports the encrypted VCPU states to the CVM.
ExptMem/ImptMem	**ExptMem** exports the encrypted contents from a CVM's memory page. **ImptMem** imports the migrated contents to a CVM's memory page.
ExptMeta/ImptMeta	**ExptMeta** exports the encrypted Metadata of the CVM. **ImptMeta** imports the encrypted Metadata for the CVM.
WProtM/UnProtM	**WProtM** enables write protection of a CVM's memory page. **UnProtM** cancels the write protection of a CVM's memory page.
ZeroPage	**ZeroPage** returns true if a given CVM's memory page is a zero page; returns false if otherwise.

4.2.1 Migration Setup. Since our design leverages an AE scheme like AES-GCM to protect state transfers, both migration sites should use a shared secret key in a CVM's migration process. The source and target platforms could employ Diffie-Hellman exchange to establish a shared transport-protection symmetric key (*TPK*) and then use the key to protect a per migration session key (*MSK*). The *MSK* is used in the migration session to protect the CVM's states and metadata. An elliptic-curve public-private key pair could be generated statically for each CVM platform to support Diffie-Hellman. Each party must acquire the other party's public key before executing the protocol to derive a shared *TPK*. For CCA-based CVMs with encrypted runtime states, our design does not migrate the per-CVM key across sites to simplify key management.

Instead, the target platform must generate a new per-CVM key for the migrated VM. Our design provides the **MigStart** command for the source and destination hypervisors to start a migration session. Both sites then use **MigTransKeyGen** to create a shared *TPK*. The source then makes **MigSessionKeyGen**. The secure module returns a *MSK* encrypted by *TPK* and a MAC that the source hypervisor sends to the target. The target hypervisor makes **MigKeyImpt** to validate the integrity of the *MSK* and import the key to the platform. The *MSK* is used throughout the migration session.

4.2.2 Export/Import Commands. The migration API exposes commands to export and import encrypted CVM states in VCPU and memory, and metadata. Since CVMs assume end-to-end I/O protection, our design tasks the hypervisor to migrate CVM's I/O devices. The `export` commands (those start with **Expt** in Table 1) use the CVM's *MSK* to encrypt CVM states/metadata and produce the respective MAC. The `import` commands (those start with **Impt** in Table 1) take the exported ciphertext and MAC as input and verify the integrity of the migrated contents. If the verification fails, it implies that the ciphertext has been corrupted, in which case the secure module rejects the import; otherwise, it decrypts the ciphertext using the *MSK* and imports the contents to the migrated CVM.

For VE-based CVMs, **ExptVCPU** encrypts VCPU states stored in a shadow VCPU structure from the secure module's private memory and exports the ciphertext. For CCA, **ExptVCPU** outputs the encrypted states from a Realm's REC and REC auxiliary granules. The latter includes the VCPU's extended features, such as Scalable Vector Extension (SVE) and Scalable Matrix Extension (SME). **ImptVCPU** decrypts and outputs to the structure that the respective CVM's secure module uses to store VCPU states.

ExptMem emits a memory page's contents and attributes; both are encrypted. The latter is stored in a `memaux` field in the exported contents. **ImptMem** command decrypts and unfolds `memaux` and restores the attributes for the migrated CVM. For VE-based CVMs, **ImptMem** stores the imported states to a CVM's private page (a page that is unmapped from the hypervisor's S2PT). For CCA, **ImptMem** emits the resulting memory states to a granule that the hypervisor allocates and delegates to the Realm. If memory encryption is effective for a Realm, **ExptMem** first decrypts memory using the Realm's secret key and encrypts the plaintext using the *MSK*; **ImptMem** decrypts the ciphertext using the *MSK* then re-encrypts it with the target Realm's secret key.

The migration API protects associated attributes of the migrated source CVM pages. The attributes include the IPA that maps to the migrated page, the attribute bits of the IPA's respective S2PT/RTT entry, and the page's current sharing status. The secure module must ensure that the IPAs of the target CVM map to a page containing the same IPA contents at the source to prevent remap attacks. The S2PT/RTT entry for a respective IPA should use the attributes from `memaux` that specify the memory access attribute and permission. For CCA, the extended RMM should additionally restore the Realm IPA State (RIPAS) to the RTT entries. For VE-based CVMs, the secure module manages in-memory metadata to track the CVM's memory-sharing status. **ExptMem** populates the sharing information into `memaux` and **ImptMem** performs vice-versa. On Arm CCA, Realms allocate shared pages from the unprotected area of its IPA space. **ExptMem** and **ImptMem** identify a page's sharing status by its IPA.

A CVM's per memory page metadata can be migrated in `memaux`. Additional configuration metadata that the secure module manages for each CVM can be exported and imported via **ExptMeta** and **ImptMeta**. For CCA, the secure module migrates metadata attributes stored in the Realm Descriptor (RD), such as a Realm's RTT and SIMD configuration.

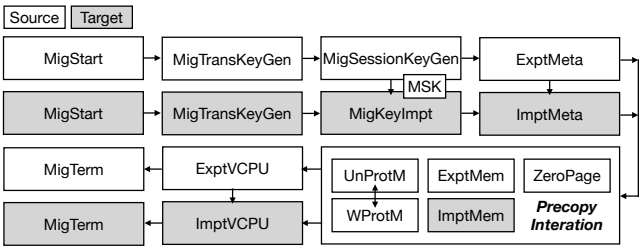

Figure 1: CVM live migration with Pre-copy

4.2.3 Support Live Migration Approaches. We discuss extending KVM and QEMU to use the proposed migration commands to support CVM live migration. Figure 1 demonstrates the command invocation for pre-copy migration. The source and target platforms use migration commands to create a shared *MSK* before transferring metadata. Here, we assume the metadata remains static after CVM creation. To support pre-copy, QEMU is extended to make **ExptMem** to export CVM's dirty pages and **ImptMem** to import the migrated memory. Since KVM cannot update CVM's S2PT or RTT to write-protect mappings, it invokes **WProtM** to track dirty pages. KVM uses **UnProtM** to disable write protection when the tracking completes. We provide the **ZeroPage** command to hypervisors to identify zero pages that require no migration. When pre-copy pauses the source VM, the source hypervisor makes **ExptVCPU** and sends the exported VCPU states for the target to import via **ImptVCPU**.

MultiFD reuses the changes for pre-copy to exchange keys and migrate CVMs' CPU states and metadata. QEMU's mechanisms to dispatch dirty pages to *sending threads* are also unmodified. Developers have to extend QEMU's sending thread to use **ExptMem** to export the encrypted contents of

the dirty page to a per-sender buffer and the *receiving thread* to import the encrypted contents with **ImptMem**. Post-copy can also reuse the changes for migrating non-memory states. QEMU invokes **ExptMem** at the request from the monitor thread to export source CVM's memory states. The monitor thread invokes **ImptMem** to import the received states.

5 IMPLEMENTATION

Based on the proposed design, we have extended the secure module in a VE and CCA-based CVM implementation. We extended SeKVM [26]'s trusted hypervisor core for the VE-based implementation and the RMM in Arm CCA to expose live migration commands. SeKVM and Arm CCA incorporate an untrusted KVM to host CVMs and provide essential virtualization features. Our implementation utilizes KVM and QEMU's comprehensive live migration support for Arm-based CVMs.

VE-based CVMs. We extended SeKVM based on KVM in Linux 4.18 and QEMU 3.0 from the opened source artifact [2]. We added 1,103 LOC to SeKVM to its trusted core and the host Linux kernel. In addition, we ported SeKVM's QEMU patch to QEMU 4.2.1 and added 321 LOC to support CVM live migration. We ported an AES implementation [1] to SeKVM. We extended SeKVM's trusted core to use AES to implement the `export` and `import` commands for CVM VCPU and memory, which use AES to encrypt and decrypt VM states, respectively. SeKVM requires no metadata export/import. SeKVM uses an access control-based approach. It does not encrypt CVM memory at runtime. Additionally, we extended SeKVM to make **WProtM** and **UnProtM**, and QEMU to invoke **ZeroPage**. SeKVM exposes hypercalls for each of the migration commands. We extended the host Linux to wrap hypercalls as ioctls. We modified QEMU to make the exposed ioctls. The current SeKVM prototype supports pre-copy, MultiFD, and post-copy migration.

CCA-based CVMs. We modified the RMM [7] (tf-rmm-v0.4.0) to expose the migration commands via RMIs. The RMM provides `export` and `import` commands for VCPU, memory, and per-Realm metadata. The RMM updates the CVM's RTT to enable and disable write protection for CVM memory within a given range to support the **WProtM** and **UnProtM** commands. The RMM performs TLB maintenance to ensure the RTT updates are effective. We added 406 LOC to the RMM. We modified CCA-KVM [5], the KVM implementation for Arm CCA, to wrap the RMIs for `export` and `import` commands into ioctls so that QEMU can issue them as a system call. In total, we added 311 LOC to CCA-KVM. CCA-KVM uses **WProtM** to track dirty pages and uses **UnProtM** to resolve write faults caused by dirty page tracking. The current KVM implementation for CCA does not support creating Realms via QEMU. We ported the patch for Arm CCA from QEMU-CCA [6] to QEMU v8.2. We extended QEMU v8.2 with 532 LOC to support Realms and live migration. The current CCA-based prototype supports live migration of Linux-based Realms with pre-copy and post-copy on the same host. Since no Arm hardware with CCA support is available, we tested the current implementation on Arm's FVP for CCA (FVP_Base_RevC-2xAEMvA) and ensured it delivers the intended functionality. We discuss the limitations of our prototype in Section 8.

While implementing the CCA prototype, we resolved incompatibility issues in the RMM and KVM. For RMM, due to dirty page logging, our prototype results in Realm exits caused by Realms writing to read-only pages. The current RMM implementation acquires the last level RTT table lock when examining memory abort type; it does not release the RTT lock before transferring control to KVM, causing a deadlock in KVM's subsequent call to `UnProtM`, which requires locking the same RTT, to update the RTT and resolve write-fault. To address the issue, we modified the RMM to release the RTT lock before switching back to KVM. In the post-copy scenario, a Realm might execute code from unmapped memory, causing an instruction abort that transfers to CCA-KVM. However, CCA-KVM currently does not handle instruction aborts for Realms. We extended CCA-KVM to handle the abort and support post-copy. Furthermore, we adapted CCA-KVM's existing fault-handling logic to utilize **ImptMem** for importing migrated memory states into the Realm.

6 PERFORMANCE EVALUATION

The FVP simulator that our CCA prototype supports does not report correct timing information. Therefore, we report the live migration performance of our SeKVM-based prototype and KVM on the HP Moonshot m400 Arm server [8] to measure the impact of the cryptography-based approach. Each VM and CVM has 4 VCPUs, 1GB RAM, a virtio network device using VHOST, and virtual disk storage with the `cache=none` setting. On KVM, the VM ran the mainline Linux 4.18; on SeKVM the CVM ran Linux 4.18 with an enlightened virtio drive, both with Ubuntu 18.04. The virtual disk was shared via NFS across sites. We evaluated a CVM running two benchmarks: **Idle VM** and **Apache**. We ran the Apache v2.4.41 server in the VM to host its default web page. Linux's stress command runs in the background to generate memory pressure. The client runs the ApacheBench [30] v2.3 on another m400 machine to request the server in 100 concurrent connections.

Table 2 shows the average performance of SeKVM and KVM using three settings: pre-copy, post-copy, and MultiFD with 8 threads. The results are an average of 10 runs. We report two key metrics reported from the source QEMU:

- *Total time (Total):* the total time taken to complete the migration process.
- *Downtime (Down):* the time spent when the VM is paused.

Table 2: Live migration performance - SeKVM and KVM

Benchmark	Time: *ms*	pre-copy		MultiFD-8		post-copy	
		Down	Total	Down	Total	Down	Total
Idle	SeKVM	28.8	32205.7	29.1	5808.0	27.6	31668.8
	KVM Arm	23.0	848.0	20.2	600.4	23.7	1097.3
Apache	SeKVM	22341.9	95027.4	3047.0	23744.6	32.2	55231.6
	KVM Arm	368.5	1618.2	215.4	1177.3	24.2	1763.2

Table 2 shows that live migration of CVMs on SeKVM performs significantly worse than VMs on KVM. During pre-copy, a given page may be exported and imported multiple times. Cryptography operations involved in the process incur significant performance overhead. Future implementations could leverage hardware cryptography to enhance performance. The overhead of live migration became much higher in Apache than in Idle VM due to a larger working set. In our prototype, MultiFD performs cryptographic operations in SeKVM concurrently on different processors. It resulted in significant improvements in total time and downtime in Apache over pre-copy. Compared to pre-copy and MultiFD, post-copy reduced the downtime significantly in Apache for CVMs by avoiding cryptographic operations in migrating dirty pages after the VM is paused. QEMU's post-copy implementation relies on a single thread to migrate VM memory, resulting in a longer total time than MultiFD.

7 RELATED WORK

CVM Live Migration. AMD SEV provides hardware extensions to encrypt CVM memory (SEV) and VCPU states (SEV-ES) and protect the integrity of CVM memory and NPT (SEV-SNP). SEV adopts a secure module-based design. SEV's secure firmware exposes commands to hypervisors to implement live migration of CVMs with SEV and SEV-ES protection, but not SEV-SNP. AMD has extended KVM [22] and QEMU to support pre-copy. Unlike our design, SEV does not ensure the integrity of the migrated CVM states. Further, SEV cannot support parallel migration and post-copy [24].

Alternative CVM-based design [10, 14, 31] has been introduced to address the limitations of SEV. These works rely on a separate CVM [10, 31] or an in-CVM Service Module [14] to provide migration service to the hypervisor or perform self-migration. The design aims to offload migration operations from the closed-source SEV firmware to provide enhanced performance and support migrating SEV-SNP-based CVMs. Compared to our design, the CVM-based design requires additional resources allocated to the special CVM and SVSM. Further, these approaches reinvent a new interface (e.g., upcalls) to use migration functionalities, complicating hypervisor and CVM communication; our design reuses the hypercall or RMI interface. Finally, self-migration requires new migration mechanisms, limiting portability.

Intel TDX adopts a hybrid CVM-based approach. It incorporates a special CVM, MigTD, to enforce migration policy and create migration sessions while delegating functionalities requiring access to CVM states, including exposing migration commands, to the secure TDX module. Like our design, TDX also leverages authenticated encryption to protect CVM state exports and imports. VirTee [32] proposed a new framework for the live migration of RISC-V-based CVMs. In contrast to our design, VirTee takes a clean-slate approach. It relies on a privileged TCB (Enclave Monitor) in the CVM to do all migration jobs, including transferring data. VirTee is incompatible with hypervisors like KVM and thus cannot leverage existing migration approaches.

8 CONCLUSION AND FUTURE WORK

This paper introduced a design for the live migration of Arm-based CVMs for CCA and non-CCA-based implementations. The design aims to reuse commodity live migration features to simplify development efforts. We demonstrated the design's applicability and effectiveness to Arm-based CVM implementations to support various KVM's existing live migration approaches and evaluated their performance.

Future Work. We aim to address the following limitations in the future. Unlike SeKVM, the RMM does not support AES; RMM's export and import commands thus operate on plaintext. Further, neither SeKVM nor CCA prototypes support authenticated encryption or the `Setup` migration commands. For SeKVM, we statically assign a `MSK` for each migration session. In addition, we plan to extend the CCA prototype to support parallel migration in the future. Service providers may abort an active live migration session when detecting errors. We could add a new abort command to our design to prevent the hypervisor from resuming the CVM on the target. Further, our design could be extended to migrate CVM's SMMU [12] (Arm's IOMMU) configurations. The pre-copy approach can cause a CVM crash if a malicious hypervisor imports outdated page versions. To mitigate this issue, we can track page versions like previous work [20] and ensure only the most recent page version are imported to CVMs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful feedback. This research was partly supported by the National Science and Technology Council of Taiwan under research grants 112-2628-E-002-027- and 113-2634-F-002-001-MBK.

REFERENCES

[1] 2018. *kokke/tiny-AES-c: Small portable AES128/192/256 in C.* https://github.com/kokke/tiny-AES-c

[2] 2021. *SOSP 21: Artifact Evaluation: Verifying a Multiprocessor Hypervisor on Arm Relaxed Memory Hardware.* https://github.com/VeriGu/usenix-ae-linux/tree/bf0e5d0a4ec4f2b6d2524a5c8cd86e12cc0f6e0e

[3] 2023. *Google pKVM Project.* https://android-kvm.googlesource.com/linux/+/refs/heads/pkvm

[4] 2023. *QEMU/Migration-Multiple-fds.* https://wiki.qemu.org/Features/Migration-Multiple-fds

[5] 2024. *Linux-CCA.* https://gitlab.arm.com/linux-arm/linux-cca/-/tree/cca-full/rmm-v1.0-eac5

[6] 2024. *QEMU-CCA.* https://jpbrucker.net/git/qemu/log/?h=cca/rmm-v1.0-eac5

[7] 2024. *TF-RMM.* https://git.trustedfirmware.org/TF-RMM/tf-rmm.git/+/refs/tags/tf-rmm-v0.4.0

[8] 2024. The CloudLab Manual: 12 Hardware. https://docs.cloudlab.us/hardware.html.

[9] Advanced Micro Devices. 2020. Secure Encrypted Virtualization API Version 0.24. https://www.amd.com/system/files/TechDocs/55766_SEV-KM_API_Specification.pdf.

[10] Advanced Micro Devices. 2022. TSEV Secure Nested Paging Firmware ABI Specification. https://www.amd.com/system/files/TechDocs/56860.pdf.

[11] ARM Ltd. 2022. Introducing Arm Confidential Compute Architecture Version 1. https://developer.arm.com/documentation/den0125/0100/What-is-Arm-CCA-.

[12] ARM Ltd. 2024. Arm System Memory Management Unit Architecture Specification SMMU: architecture version 3. IHI0070F_b System Memory Management Unit Architecture Specification.

[13] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the 2005 USENIX Annual Technical Conference (USENIX ATC 2005)*. Anaheim, CA.

[14] Carlos Bilbao. 2023. The Linux SVSM project. https://lwn.net/Articles/921266/.

[15] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *2nd Symposium on Networked Systems Design & Implementation (NSDI 05)*. Boston, MA.

[16] Morris Dworkin. 2007. NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf.

[17] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2008. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of the 17th USENIX Security Symposium (USENIX Security 2008)*. San Jose, CA, 45–60.

[18] Michael R. Hines and Kartik Gopalan. 2009. Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*. Washington, DC, USA, 51–60.

[19] Intel Corporation. 2023. Intel® Trust Domain Extensions (Intel® TDX). https://cdrdv2.intel.com/v1/dl/getContent/733582.

[20] Intel Corporation. 2023. Trust Domain Extension (TDX) Migration TD Design Guide. https://cdrdv2.intel.com/v1/dl/getContent/733580.

[21] Jake Edge. 2020. KVM for Android. https://lwn.net/Articles/836693/.

[22] Ashish Kalra. 2021. Add AMD SEV guest live migration support. https://lwn.net/Articles/851648/.

[23] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the 2007 Ottawa Linux Symposium (OLS 2007)*. Ottawa, ON, Canada.

[24] Jian-Lin Li and Shih-Wei Li. 2024. Performance Implications of SEV Virtual Machine Live Migration. In *Proceedings of the 19th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2024)*. Madrid, Spain.

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

[26] Shih-Wei Li, Xupeng Li, Ronghui Gu, Jason Nieh, and John Zhuang Hui. 2021. A Secure and Formally Verified Linux KVM Hypervisor. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1782–1799.

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

[28] Rusty Russell. 2008. virtio: Towards a De-Facto Standard for Virtual I/O Devices. *SIGOPS Operating Systems Review* 42, 5 (July 2008), 95–103.

[29] Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. 2013. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13)*. Houston, Texas, USA, 85–96.

[30] The Apache Software Foundation. 2015. ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.4/programs/ab.html.

[31] Tobin Feldman-Fitzthum, Dov Murik. 2021. Secure Live Migration of Encrypted VMs. https://research.ibm.com/publications/secure-live-migration-of-encrypted-vms.

[32] Jianqiang Wang, Pouya Mahmoody, Ferdinand Brasser, Patrick Jauernig, Ahmad-Reza Sadeghi, Donghui Yu, Dahan Pan, and Yuanyuan Zhang. 2022. VirTEE: A Full Backward-Compatible TEE with Native Live Migration and Secure I/O. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC '22)*. San Francisco, California, 241–246.

[33] Wei Wang. 2021. TDX Live Migration. https://kvmforum2021.sched.com/event/ke3F/tdx-live-migration-wei-wang-intel-corp.