

WIP: BARESLICE: Extending Arm CCA to Support Bare-Metal Confidential Virtual Machines

Yuxin Hu
Vanderbilt University
yuxin.hu@vanderbilt.edu

Fengwei Zhang
Southern University of Science and
Technology
zhangfw@sustech.edu.cn

Kevin Leach
Vanderbilt University
kevin.leach@vanderbilt.edu

Abstract

The Arm architecture is becoming popular in cloud computing due to its efficiency, low power consumption, and rich ecosystem, making it highly suitable for data centers. For example, Nvidia introduced the Grace CPU, a 144-core Arm architecture processor, to serve as the foundation for next-generation data centers. However, as cloud infrastructures grow, the use of complex hypervisors to manage tenant environments introduces security concerns. In 2021, Arm released the Confidential Computing Architecture (CCA), which provides secure execution environments for confidential VMs and removes the hypervisor from the Trusted Computing Base. Despite CCA’s improvements, confidential VMs still face security and performance challenges due to the shared nature of physical resources. While bare-metal cloud services offer a promising solution by allocating directly-accessible physical resources to each tenant, the substantial costs of dedicating hardware to each tenant has limited its widespread adoption.

In this paper, we present BARESLICE, a novel bare-metal cloud system that supports multiple untrusted tenants on shared bare-metal hardware. BARESLICE assigns dedicated physical resources to each tenant, which we call a *slice*, by leveraging the hardware features of CCA to ensure robust isolation and create a secure computing environment free from virtualization overhead. We analyze the security of BARESLICE to show its effectiveness against common attacks. Additionally, our performance assessment illustrates that BARESLICE achieves bare-metal performance levels across several real-world applications like Apache and Memcached. BARESLICE achieves a combination of enhanced security and high efficiency, providing a compelling solution for modern confidential cloud computing challenges.

1 Introduction

Virtualization serves as a cornerstone of today’s public cloud services [40, 47]. Virtualization allows a single physical server to be partitioned into multiple isolated Virtual Machines, offering flexible and scalable environments for cloud customers. However, existing cloud services face many security challenges [44, 50]. As additional functionality is incorporated, the hypervisor software for managing VMs continues to increase in size and complexity, resulting in a substantial increase to the exposed attack surface for potential threats [13, 24].

To address these concerns, confidential VM architectures have emerged. These architectures, including AMD Secure Encrypted Virtualization [3], Intel Trust Domain eXtensions [28], and Arm Confidential Computing Architecture (CCA) [15], provide secure execution environments for guests that are opaque to untrusted privileged software through new architectural extensions. Arm released

the CCA specification in 2021 [17], which is slated for widespread deployment in the latest Arm v9 devices. Arm CCA extends Arm TrustZone [16] by introducing two new Worlds (Realm and Root) alongside two existing Worlds (Normal and Secure). The Realm World provides secure computing environments, called Realm VMs, for use as confidential VMs. These Realm VMs are isolated from external privileged software, including OS’s and hypervisors. This design ensures that, even if a specific Realm VM or the host OS becomes compromised, the other aspects of the system remain intact. The Root World executes the most privileged firmware, the EL3 Monitor (EL3M), which monitors the switching between and isolation of different Worlds. Additionally, Arm CCA integrates support for remote attestation and memory encryption for Realm VMs, further enhancing the security of the system.

However, security and performance concerns persist with confidential VMs, including those built upon CCA. The sharing of a single physical server across different users or customers may present security risks, as malicious tenants could potentially obtain private data by analyzing resource usage patterns to break isolation guarantees provided by confidential VM techniques [27, 34, 35, 46, 53]. Furthermore, virtualization (including Realm VMs) introduces a non-negligible performance overhead [26, 37, 38]. To reduce this overhead, cloud providers like Amazon [47] and Azure [40] typically use fixed allocations for CPU cores and memory in their VMs [14, 39], focusing on simpler and more predictable resource management. They also delegate I/O processing to dedicated hardware for minimizing resource contention and oversubscription. This strategy leads to many VMs having static resource allocations, with the hypervisor mainly serving to partition resources for distribution [57].

Due to these limitations and the overhead associated with virtualization, there is a growing interest in bare-metal cloud services. These services allow users to rent dedicated physical servers, eliminating concerns associated with resource sharing and virtualization overhead. However, current bare-metal services lack scalability and efficiency, and face a high cost for access [22].

In this paper, we present BARESLICE, an approach that provides multiple tenants with isolated confidential environments while achieving bare-metal performance. BARESLICE extends Arm CCA to enable isolating hardware resources—which we call a *slice*—on a shared physical host among multiple tenants without incurring virtualization overhead. BARESLICE uses the Realm Management Extension (RME), a hardware primitive available in CCA, to isolate each guest’s slice from other slices as well as untrusted privileged software. Slices are granted exclusive access to resources, including physical memory regions, CPU cores, and I/O devices. User code within the slice is granted higher privilege, which allows execution of a native hypervisor or guest OS inside a slice. Following CCA’s

design, a Granule Protection Table (GPT) partitions pages of memory into different Physical Address Spaces (PAS). When attempting to access memory, the RME employs a Granule Protection Check (GPC) to validate the security state of the CPU core against the value stored in the GPT to determine whether the CPU core has the appropriate privilege level to access the requested page. Illegal memory accesses (e.g., Normal World software accessing a page marked in the GPT as Realm memory) are blocked and trigger exceptions. BARESLICE leverages this mechanism to separate resources for each slice, enabling strong hardware-enforced isolation between slices to achieve a bare-metal multi-tenant (cloud) environment.

Implementing BARESLICE presented us with several challenges: (1) slice memory isolation, (2) slice core isolation, and (3) slice execution control. We modify the EL3M firmware to address these challenges, as discussed below.

First, while the GPT has been used to isolate memory belonging to different *Worlds*, we had to augment this mechanism to support isolating *slices*, all of which are part of Realm World. To achieve this goal, we adapt the multi-GPT mechanism from prior work [56]. In this regime, we configure each slice with a unique GPT. Doing so allows us to achieve three key desirable isolation properties: (a) each slice *cannot* access memory belonging to any other slice, (b) each slice *cannot* access memory belonging to any other World (e.g., a slice cannot access the host OS), and (c) no other World or Realm can access memory belonging to each slice (e.g., the host OS cannot access a slice’s memory).

Second, each slice is assigned physical CPU cores, which means we must prevent malicious Inter-Processor Interrupts (IPIs) from reaching invalid cores (e.g., if the host OS sends IPIs to a slice’s core). To prevent unauthorized IPIs, we disable the forwarding of IPIs in slice cores by setting appropriate control registers. For intercepting illegal attempts to enable forwarding of IPIs, we configure control registers as inaccessible in GPTs so that all illegal accesses to these registers are blocked and reported to the EL3M. To support intra-slice communication for slice cores, we implement an SMC request for slice cores to send IPIs. In this SMC handler, the EL3M temporarily enables IPI forwarding in the target cores to transfer the IPI during a brief window.

These mechanisms provide a highly-isolated bare-metal environment to support multiple tenants on a shared physical host (e.g., cloud infrastructure). Slices only need to trust the underlying firmware, removing the hypervisor from the TCB while defending against a variety of sophisticated attacks from untrusted software.

We implemented a prototype on Arm Fixed Virtual Platform (FVP) [4], which incorporates CCA support, to demonstrate the security and performance attributes of our system. Our evaluation encompasses both a security analysis and a performance assessment. In the security analysis, we scrutinize common attack techniques and describe the defense mechanisms we implemented in BARESLICE. Our analysis illustrates that BARESLICE is resilient against potential attacks originating from highly-privileged software, such as the trusted OS or hypervisor. We evaluate the system’s runtime performance against bare-metal Linux and KVM in real-world applications like Apache and Memcached. The results show that BARESLICE achieves comparable performance to bare-metal Linux and substantially outperforms KVM.

We claim the following contributions in this paper:

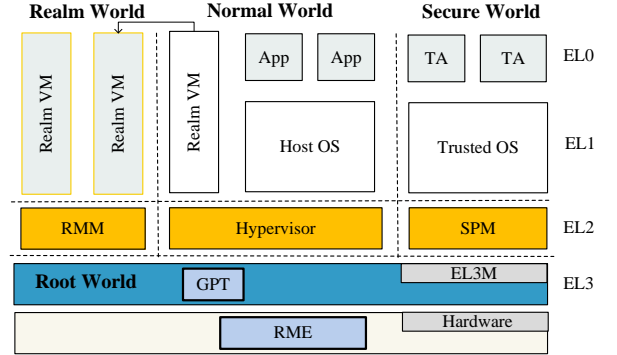


Figure 1: Arm CCA components. The Realm VM is initialized by the host OS and transitioned to Realm World. Once in Realm World, a Realm VM cannot be accessed by other Realm VMs, Normal World, or Secure World.

- We present the design of BARESLICE, the first system that enables bare-metal confidential virtual machines called *slices*, allowing high isolation among tenants and without any hardware modifications on Arm CCA-enabled platforms.
- We develop a functional prototype based on Arm FVP to demonstrate the security features for slices created with BARESLICE against a variety of sophisticated threats.
- We evaluate the performance of BARESLICE on real hardware. Our evaluation demonstrates bare-metal performance for several indicative real-world applications.

2 Background

In this section, we introduce several basic concepts relevant for implementing BARESLICE. Specifically, we discuss (1) Arm TrustZone and (2) the Arm Confidential Computing Architecture.

2.1 Arm TrustZone

Arm TrustZone establishes a dual-World system to implement a Trusted Execution Environment: the *Secure World* for sensitive operations, and the *Normal World* for routine computing activities. This robust isolation guarantees that all activities and data in the Secure World are fully protected from any unauthorized access or interference originating from the Normal World—even if the entire OS stack in Normal World becomes compromised, the Secure World activities can continue executing faithfully.

However, TrustZone does not address more recent challenges associated with confidential VMs, especially as Arm grows in prevalence as a target architecture in cloud computing scenarios. We discuss the CCA specification below, which augments TrustZone with additional capabilities that we leverage in this paper.

2.2 Arm Confidential Computing Architecture

In 2021, Arm released the CCA specification, designed to support the creation of confidential VMs, termed *Realm VMs*. For our purposes, the main contribution of CCA is the introduction of *Realms*, which are secure execution environments that exist in parallel with

Table 1: Arm’s GPC provides hardware-enforced page-level access control across Security state denoted by Physical Address Spaces (PAS)—A ✓ indicates that Security state can access memory belonging to that PAS.

State	Normal PAS	Secure PAS	Realm PAS	Root PAS
Normal	✓	×	×	×
Secure	✓	✓	×	×
Realm	✓	×	✓	×
Root	✓	✓	✓	✓

the conventional Normal World OS. These Realms offer a dedicated environment for storing and executing sensitive data and tasks while protecting against threats from privileged software such as a trusted OS or hypervisor. In this subsection, we discuss key elements of CCA, including Realm and Root Worlds, the Realm Management Monitor, memory isolation using the EL3M firmware, and device isolation using the System Memory Management Unit.

2.2.1 CCA Foundations. Arm CCA expands upon the TrustZone architecture by adding two new security states: the *Realm World* for hosting Realm VMs and the *Root World* to manage security states and isolation through the EL3M firmware. The Realm World isolates Realm VMs from Normal and Secure Worlds, with these VMs being initialized by the host OS and hypervisor. Each Realm VM transitions into the Realm World, achieving a distinct separation from the untrusted OS and hypervisor. Despite this, CCA continues to rely on traditional virtualization techniques for VM isolation and management, incurring notable performance overhead. In response, our design implements slices, or dedicated physical resources, on the Arm architecture to eliminate virtualization overhead, allowing each slice to operate independently in isolation, thus enhancing both performance and security. We augment the hardware support for Realm VMs to implement slices on the Arm architecture.

2.2.2 Realm Management Monitor. A key component within Realm World is the Realm Management Monitor (RMM), which functions like a lightweight hypervisor. The RMM oversees the security-critical operations of the Realm VMs, including the management of Stage-2 page tables and the interaction with Normal World software. For tasks that extend beyond its security scope, such as device emulation, the RMM delegates these tasks back to the untrusted hypervisor. Given the central role of the RMM in managing Realm VMs, a compromised RMM poses a significant risk, potentially affecting all slices. Our design counters this by excluding the RMM from the TCB and implementing specific security measures to mitigate such risks, which are detailed in our Security Analysis in Section 6.

2.2.3 EL3M firmware and memory isolation. The most privileged firmware, EL3M, operates within the Root World. The EL3M is tasked with managing transitions between different security states and Worlds. Unlike TrustZone, which is partitioned only into Normal and Secure Worlds, the EL3M executes within Root memory, which is inaccessible from software in the Secure, Normal, or Realm Worlds. To conduct a regular memory access across different Worlds, the EL3M uses the GPC to validate a CPU core’s access to a page of

memory against the GPT. The GPT defines access and permission by security state for each page of memory. For example, pages marked as Root PAS is exclusively accessible from the Root World security state, while pages marked as Normal PAS are accessible from any security state. Table 1 illustrates the security states that can access each type of memory page during a GPC. Notably, Realm PAS are only accessible from the Realm or Root security states, which means that Realm VMs must trust the EL3M. Meanwhile, Secure PAS are accessible both to the Secure and Root security states, allowing the EL3M to manage Secure memory interactions.

The GPC plays a critical role by ensuring that each memory access complies with the permissions outlined in the GPT and aligns with the current security state of the CPU core making the access. Unauthorized access attempts, such as Normal World software reading Realm PAS, are intercepted by the GPC, triggering Granule Protection Fault (GPF) exceptions. The GPF traps to EL3M, which allows us to introspect and block such accesses.

2.2.4 Device isolation. The SMMU is integral to Arm’s architecture for managing the interactions between DMA-capable peripherals and the main system memory. It allows privileged software to configure SMMU registers through Memory-Mapped I/O (MMIO), which include settings for managing page tables and translation configurations, thereby enforcing strict access controls.

CCA secures DMA transactions by integrating a GPC for relevant addresses within the SMMU [6]. To safeguard this functionality, CCA uses specific MMIO registers in the SMMU that are accessible only from the Root World. These registers are crucial for configuring the SMMU’s GPC functions, including establishing the base for the SMMU GPT and managing the SMMU GPC settings.

3 Overview

In this section, we outline the objectives and architecture of BARESLICE, followed by a threat model.

3.1 Goals and Security Properties

BARESLICE aims to extend Arm CCA to support bare-metal confidential virtual machines, enabling the partitioning of a single physical machine into multiple independent execution environments, which we call *slices*. Through BARESLICE, each slice is assigned direct and isolated access to hardware resources, including CPU cores, memory, and I/O devices. Unlike traditional virtualization, BARESLICE provides bare-metal performance levels and provides slices with elevated privileges (up to EL2). This setup allows tenants to run either a full OS stack or a native hypervisor within their own slice, while relying solely on the trustworthiness of the underlying EL3M firmware and hardware, and safeguarding against potential vulnerabilities in privileged software like hypervisors or a trusted OS. To realize these objectives, BARESLICE enforces several key security properties, inspired by previous work [57]:

- **Property 1:** Ensuring that once a slice is established, its allocated resources remain fixed and exclusive so that they are inaccessible by other slices.
- **Property 2:** Guaranteeing that the resources designated for a slice are inaccessible by external software, and that a slice cannot access resources beyond its allocated boundaries.

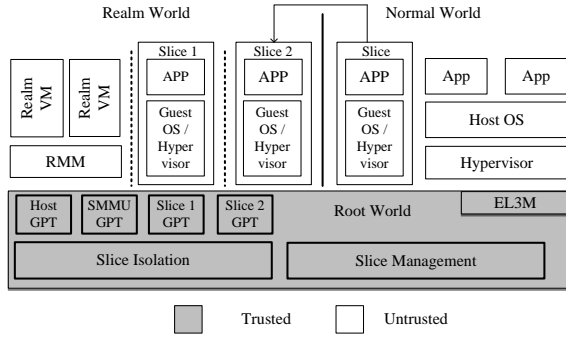


Figure 2: Architectural overview of BARESLICE, illustrating the initialization of a slice in the Normal World and its subsequent transfer to the Realm World. Note that the Secure World is omitted for simplicity. Our custom EL3M firmware facilitates slice management and is the TCB for BARESLICE.

- **Property 3:** Restricting each slice from dispatching IPIs to cores outside that slice’s control, while allowing intra-slice IPI communication.
- **Property 4:** Facilitating the dynamic creation and termination of slices with minimal disruption while the underlying system runs.

This paper presents BARESLICE to support these security properties, which allows us to create and manage secure, efficient, and flexible computing environments, including bare-metal cloud operations.

3.2 Architecture Overview

Figure 2 illustrates BARESLICE’s architecture. Our approach allows creating and managing *slices*, which are fixed and exclusive hardware resources consisting of CPU cores, regions of memory, and I/O devices for each tenant in a cloud computing environment. Initially, slice resources are allocated and initialized by the host OS, similar to Realm VMs. Once operational, these slices are transitioned into the Realm World, effectively segregating the slice from any access from untrusted software, including the host OS, hypervisor, trusted OS, Secure Partition Manager (SPM), and Realm Management Monitor (RMM). We implement custom changes to the EL3M to provide facilities for managing and isolating each slice and its associated resources. Specifically, we use the EL3M to create a unique GPT for each slice to delineate the pages of memory that belong to a slice and ensure that slice cannot access other Realms and slices (and vice versa). In our CCA-based design, these GPTs are securely stored in the Root World, which can only be accessed by the EL3M. This capability allows the EL3M to safeguard memory regions of each slice from untrusted software and to manage unauthorized DMA accesses via the SMMU GPT.

Placing slices within the Realm World (rather than Normal or Secure Worlds) provides several security properties. First, the Realm World cannot be accessed by the Normal World, forming a robust barrier against common attacks from the host OS or hypervisor. Moreover, BARESLICE leverages per-slice GPTs to maintain a strict separation between slices and other Realm World entities, including

Realm VMs and the RMM. Second, Arm supports encrypting Realm memory with a unique encryption key using the Memory Protection Engine, providing a high degree of slice confidentiality without additional engineering effort. Third, all transitions between Worlds are controlled by the EL3M in Arm, which we can use as a trusted intermediary that captures and introspects attempts to jump from Normal or Secure Worlds into any slice.

Our design ensures that each slice is isolated with its own dedicated environment while being conveniently managed by the EL3M. This mitigates risks associated with shared resources and external software interactions while providing a high degree of resource utilization in cloud computing workloads.

3.3 Threat Model

We consider a scenario in which a malicious attacker intends to compromise confidentiality or integrity by extracting or modifying sensitive data. We assume the attacker might have gained control over software or data contained within the Normal, Secure, or Realm Worlds, including the host OS, hypervisor, trusted OS, SPM, Realm VMs, or RMM. An attacker might try to infiltrate the system through compromised software, a malicious slice, or by gaining control over devices to create malicious DMA requests.

The integrity and security of slices in our system depend on both the EL3M and the underlying hardware. We assume that the EL3M code is securely validated and can be loaded through secure boot technology, and it assumes that the hardware can be trusted. We assume that the guest code contained within a slice does not deliberately attempt to expose its own sensitive data. Following similar research [52, 56], we assume that Denial-of-Service (DoS) attacks are not in scope for this work. Note that side-channels attacks are not fully addressed in this work. However, BARESLICE can incorporate existing methodologies [25, 31, 42, 48] that can mitigate side-channel attacks to bolster its defenses against such vulnerabilities. Although not the focus of this paper, physical threats like Rowhammer [41], fault injection [19, 23], cold boot [55], or bus snooping [32] could potentially reveal slice data. However, these risks can be curtailed through the use of the Memory Protection Engine, such as providing each slice with a unique encryption key to secure its memory. This layered security approach ensures that BARESLICE maintains a robust defense against a variety of attack vectors, safeguarding the system’s integrity and the confidentiality of its data.

4 Design

BARESLICE introduces the notion of *slices* using Arm CCA—isolated execution environments consisting of cores, memory regions, and I/O devices. These resources are exclusively allocated to each slice during its lifetime, ensuring they remain inaccessible to other slices or software applications. Moreover, we implement strict access control to prevent any slice from interacting with resources beyond its designated allocation. User code within the slice is allowed to execute in a high privilege (up to EL2), supporting the execution of a native hypervisor or guest OS. In the section, we discuss the design of slice isolation and management in BARESLICE.

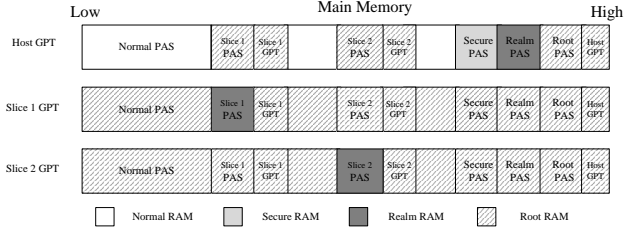


Figure 3: Illustration of Multi-GPT memory isolation in BARESLICE. Each slice receives its own GPT in which all main memory is marked as Root PAS except for the region of memory allocated for that slice.

4.1 Slice Isolation

We leverage CCA hardware features to grant each slice direct, isolated, bare-metal access to resources, thereby eliminating the need for virtualization. We discuss our approach to isolating slices in this subsection.

To achieve memory isolation among slices, we develop custom EL3M firmware. In CCA, the hardware conducts a GPC during an attempted memory access. The GPC consults the GPT to determine whether the requested memory address is accessible by the CPU core’s current security state. The GPT indicates which security state is allowed to access each PAS contained therein. In BARESLICE, unauthorized memory accesses lead to a GPF exception that traps to the EL3M, which can block the memory access from completion. The GPC is controlled by the hardware’s design, and precludes software from disabling the GPC or modifying the GPT.

Just like Realm VMs, BARESLICE positions its slices within the Realm World, making them accessible exclusively through either the Realm World security state or the Root World security state. This arrangement guarantees that the slices are inherently isolated from both the Normal and Secure Worlds. To further enhance this separation, we employ a variety of slice isolation mechanisms, designed to ensure each slice remains distinct and protected from any other software within the Realm World as well as other slices. In the following sections, we describe these slice isolation mechanisms in detail.

4.1.1 Memory. In the Arm architecture, whenever a memory access occurs, the GPC uses the GPT to determine whether that access is allowed. Therefore, the GPT plays a crucial role in separating Worlds. However, in BARESLICE, we must isolate slices in addition to Worlds. To address this challenge, we adapt the multi-GPT mechanism [56], which allows reconfiguring CPU cores with different GPTs by assigning specific values to each core’s GPTBR_EL3 register. We take this a step further by assigning a unique GPT to each slice. This allows us to achieve high isolation among slices without undermining any existing properties of the GPC mechanism.

In our system, each slice is allocated dedicated physical cores and a unique GPT. To maintain memory access controls, the GPT associated with each slice sets up allocated memory regions as Realm PAS, while other areas are labeled as Root PAS. As a result, the memory dedicated to each slice is shielded from access by other slices, even though they all operate within Realm World. This

strategy ensures each slice’s memory regions are isolated against unauthorized access.

Root PAS is accessible solely by the EL3M, which operates in the highest privilege Root World security state. The EL3M is responsible for the creation and management of these GPTs, ensuring they are securely stored within the Root PAS. Additionally, the EL3M controls the GPTBR_EL3 register, which contains the base address of the specific GPT allocated to each core. When creating a new slice, the EL3M allocates memory to contain a new GPT in Root PAS. Then, when the slice executes on the assigned core, the GPTBR_EL3 register is updated to contain that slice’s GPT. This ensures that only the cores designated for a particular slice can access its respective slice memory regions.

Note that the multi-GPT mechanism maintains compatibility with the CCA design. We retain the host GPT, responsible for checking memory access for software entities beyond slices, including the host OS, hypervisor, trusted OS, SPM, Realm VMs, and RMM. As shown in Figure 3, within the host GPT, all memory regions pertaining to slices are configured as Root PAS, accessible solely by the EL3M. Although slices operate within the Realm World, slice cores cannot access the Realm memory regions outside the slice, which are configured as Root PAS in the slice GPTs. Conversely, Realm VMs and the RMM cannot access the slice memory regions, as these memory regions are configured as Root PAS in the host GPT. The EL3M ensures the correct configuration of the target GPT for each slice, switching to the corresponding slice GPT during slice execution. BARESLICE enforces strict memory isolation among slices, preserving each slice’s data integrity and preventing interference, while maintaining the security guarantees of the underlying system architecture.

4.1.2 TLB/Cache. According to the Arm CCA design, GPT entries can be cached in the Translation Lookaside Buffer (TLB). Since BARESLICE uses GPTs to control memory accessibility for each core, outdated GPT information cached in the TLB could potentially compromise the memory isolation between slices—for example, malicious software might exploit stale TLB entries to access sensitive data from a victim slice. The EL3M must update GPTs to maintain memory isolation when creating and destroying slices. To ensure these GPT modifications are effectively applied to target cores, the EL3M invalidates all cached GPT information in TLBs across all cores after GPT modifications using TLB maintenance instructions (e.g., TLBI PAALLOS). This forces CPUs to read the most up-to-date GPT data, ensuring that any modifications to the GPT cannot be bypassed with stale content from the TLB.

Additionally, information within the GPT can be shared across cores through the data cache, posing a potential risk. To address this, BARESLICE disables GPT fetch sharing through the custom EL3M firmware. Specifically, the EL3M sets the SH bit to be 0b00 in the GPC control register GPCCR_EL3, rendering GPT non-sharable, which mitigates this potential side-channel attack. Note that the GPCCR_EL3 register cannot be modified outside the EL3M, preventing slices or software from enabling GPT fetch sharing.

4.1.3 Interrupts. To ensure the isolation of cores, BARESLICE prohibits cores within a slice sending IPIs to cores outside the slice. This is achieved by the EL3M writing to the GICR_ICENABLER registers, thereby disabling the forwarding of all Software Generated

Interrupts (used for inter-processor communication in Arm architecture) within slices. Additionally we prevent untrusted software from enabling these IPIs by updating the memory-mapped regions for the GIC redistributor registers as Root PAS in their GPTs. If a slice were to attempt enabling or sending an IPI by accessing these memory-mapped registers, it would raise a GPF and trap to our EL3M firmware. By preventing IPIs, BARESLICE mitigates security threats that exploit inter-processor communication, which could lead to resource exhaustion, instability, or data exposure [45].

To enable IPIs for intra-slice communication, BARESLICE introduces a custom SMC request with a minor modification to the slice OS. The EL3M checks that both the sending and receiving cores are within the same slice before proceeding. Once confirmed, the EL3M briefly activates the IPI on the target cores by updating the GICR_ISENABLER register and transferring the IPI. This temporary activation, however, creates a potential transient vulnerability, as it could be exploited by attackers to send unauthorized IPIs that disrupt the execution of the target cores.

To counter this risk, the EL3M first clears all pending IPIs before enabling them, and then only activates the specific type of IPI required by the request on the target cores. This approach dramatically reduces the likelihood of unauthorized IPI attacks during slice operations. Even if an attacker manages to send an IPI of the same type just after its activation and while the IPI is being transmitted, the potential damage is minimal—memory isolation between slices remains intact, and the slice OS could identify and ignore any unauthorized IPIs.

Nonetheless, when multiple cores within a slice request sending IPIs via an SMC request, there is a temporary activation window that could be exploited. However, single-core slices are continually protected against malicious IPIs.

4.1.4 I/O Devices. In BARESLICE, devices are allocated to slices based on core assignment, while shared devices require additional mechanisms for access control. Shared devices typically interact with the system through MMIO and Shared Peripheral Interrupts (SPIs), such as UART. To ensure each slice accesses only its designated MMIO regions and SPIs, device details are provided to the EL3M during slice creation. The EL3M records this information to prevent any conflicts in device access permissions across slices.

EL3M configures slice GPTs to control which MMIO memory regions are accessible to each slice, ensuring that any unauthorized MMIO access triggers a GPF. SPI routing is managed by setting the affinity in the GIC distributor (GICD), with the GICD memory marked as Root PAS in all GPTs so that only the EL3M can modify SPI affinities. Any attempt by the host or a slice to access GICD memory is trapped to the EL3M, which verifies the access request, including the address, type of operation (read or write), and data involved. For critical changes, such as updating SPI affinity or active status, the EL3M validates the request and performs the modification only if it pertains to an SPI assigned to the requesting slice; otherwise, the request is denied, and an error message is returned. This approach preserves the integrity of SPI attributes, preventing unauthorized changes by other slices or the host.

In BARESLICE, each device is typically assigned to a single slice or host, similar to the allocation of cores or memory. However, some devices may be limited in availability and therefore require

sharing between slices through virtualization. To virtualize a device for multiple slices, its MMIO memory page can be configured as Root PAS, ensuring that all accesses are trapped to the EL3M. Within the GPF handler, the EL3M emulates MMIO memory access, enabling shared device usage across slices. The design can be extended to support more advanced devices, such as those using DMA. For DMA-capable devices, our design can incorporate Realm Management Extensions Device Assignment (RME-DA), which assigns devices to Realm VMs with controlled memory access via DMA and includes device attestation for enhanced security. Since RME-DA is currently unavailable, methods from prior work [49, 52], designed for Realm VMs, could be integrated to enable slices to securely access DMA-capable devices. These methods use SMMU GPT and page tables to provide the necessary isolation and controlled access.

4.2 Slice Management

This subsection discusses the lifecycle of slices maintained by BARESLICE: allocating and managing resources for each slice, starting a new slice, and terminating an existing slice.

4.2.1 Resource management. Resource management in our system mirrors the principles applied in Realm VMs. The host OS is tasked with allocating resources to each slice, subsequently transferring the slice into the Realm World for isolation from both Normal and Secure World software. Each slice is allocated fixed resources, ensuring no sharing between slices or with other software, and is further isolated using the multi-GPT mechanism.

The host OS allocates unused resources to a slice, and loads a guest OS binary and a Device Tree (DT) file into the slice’s memory. This DT file outlines the resources available to the slice, such as core addresses, memory regions, and device details. Additionally, the host OS sends SMC requests to the EL3M, providing information about resources, the OS entry point, and the address of the DT for the slice.

The EL3M verifies the legitimacy of the requested memory regions by checking the host GPT, ensuring they are classified as Normal PAS, indicating they can be assigned to the slice and are not shared with or currently occupied by other slices. Upon passing this check, the EL3M updates the allocated memory regions both on the host GPT and the slice GPT. In the host GPT, the allocated memory regions are updated to Root PAS; in the slice GPT, the allocated memory regions are updated to Realm PAS. This ensures that the slice cannot access external resources, nor can external privileged software access the slice’s memory. If slice memory regions are accessed by other slices or software, the GPC fails and triggers a GPF exception, halting the unauthorized access and reporting it to the EL3M.

4.2.2 Starting a slice. The process of starting a slice begins with a user submitting a request to the host OS through `ioctl` interfaces. Figure 4 illustrates the boot flow for a slice with a single core. Upon receiving this request, the host OS allocates currently-unused resources to the slice and proceeds to load the guest OS binary and the DT file into the designated memory regions for the slice.

Following the allocation and setup, the host OS initiates the core assigned to the slice. The slice cores then transition to the EL3M, which is responsible for validating the slice creation request. The

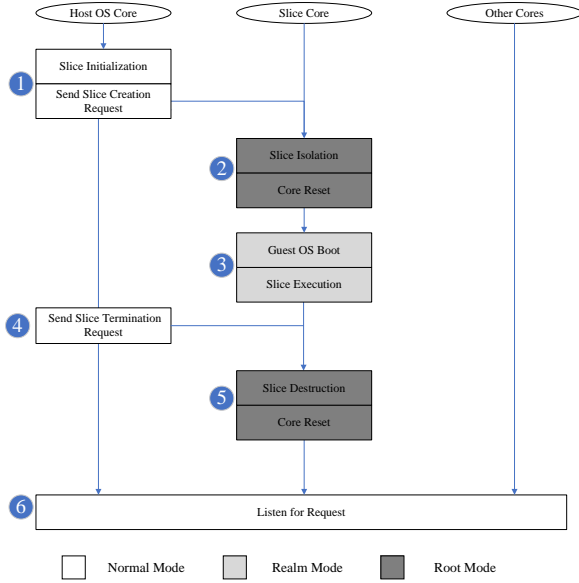


Figure 4: Lifecycle of a slice. In Step 1, the Host OS allocates unused CPU cores to be used by the new slice. Then, in Step 2, our EL3M firmware initializes the new slice (e.g., by creating a new GPT and loading an OS image). In Step 3, the slice’s assigned cores boot the guest OS. The slice executes in isolation until a slice termination request is received in Step 4. Then, in Step 5, the EL3M firmware destroys the slice. In parallel, Step 6 shows that the system can continue to service requests for slice creation and destruction.

EL3M ensures that the resources allocated do not overlap with those of other slices and belong to the Normal PAS in the host GPT. Upon validation, the EL3M initializes metadata specific to the slice, encompassing core IDs, memory regions, available devices, the guest OS entry point, the DT address, and the new GPT base address.

A critical step in the initialization phase involves updating the GPTBR_EL3 register for each of the slice cores. This directs the cores to target the newly-created slice’s GPT to establish a secure and isolated execution environment. This measure guarantees that only the designated slice and the EL3M can access the slice’s resources.

Next, the slice’s cores must be reset by the EL3M. This process is informed by the core IDs supplied as part of the slice creation parameters. Throughout the execution phase, the slice benefits from bare-metal access to its allocated resources, enabling the guest code within the slice to operate at a high privilege level (up to EL2).

4.2.3 Terminating a slice. When the tasks within a slice are completed, or if the host OS needs to reclaim the resources, the slice may need to be terminated. To manage this, upon receiving such a request, the EL3M updates the slice’s GPT, changing the memory regions of the slice to Root PAS. Since the slice GPT is stored in the Root World—where it can be modified by the EL3M but not by the slices—this modification is secure and unalterable by the slice. Following the GPT modification, the EL3M invalidates the cached

GPT information in the TLBs, inducing a GPF in the targeted slice and halting its execution.

After the slice has been deactivated, the EL3M proceeds with the destruction phase, erasing all data related to the slice. This comprehensive cleanup process involves clearing memory regions, metadata, context, the slice GPT, and all associated cache and TLB entries, ensuring that the host OS cannot access the content left by the destroyed slice. Once this phase is complete, the EL3M updates the status of the slice’s memory regions in the host GPT to Normal PAS, effectively transferring control back to the host OS and rendering the resources ready for reallocation.

5 Implementation

We developed the prototype using the Arm FVP Base RevC-2xAEMvA, an official software simulator for CCA testing. Using the FVP, we validated the security measures of BARESLICE as outlined in Section 4, including memory isolation, IPI isolation, and device isolation. We modified the official Arm firmware (Trusted Firmware-A v2.8 [10]) to implement our changes to the EL3M.

Slice initialization mirrors that of Realm VMs, involving resource allocation and loading the guest OS binary by the host OS. For both the host and guest OS, we opted for Linux kernel v5.3. We developed a kernel module to act as an interface during slice initialization, designating physically contiguous memory regions to slices.

Each slice is assigned a unique UART, with the GPTs updated to ensure that only the corresponding slice can access it. Each slice also uses a ramdisk as its disk for storage. Other devices in limited quantity on FVP (e.g., one Ethernet device) could be exclusively accessible to a slice in our prototype. However, for experimental expediency, we allow these devices to be configured by both the host OS and slices. While RME-DA support is currently unavailable in hardware, our prototype implementation can support I/O with complex DMA devices by allocating each slice a Normal World region for DMA.

After resource allocation, the host OS loads the Linux binary and DT file for each slice. The host OS then activates slice-assigned cores, which request slice creation and isolation via the EL3M. The EL3M ensures resource ownership verification, ensuring memory is configured as Normal PAS and cores are not shared. Memory isolation uses multi-GPT, with GPTs in Root World regions inaccessible to untrusted software. We modified GPT initialization to ensure efficient page-level isolation, enabling the GPF bit in SCR_EL3 to trap illegal memory accesses.

As discussed in Section 4.1, the EL3M handles IPIs and SPIs, configuring registers and designating memory-mapped regions as Root PAS. We partitioned GIC redistributor regions for different slices, ensuring isolation. The GIC distributor memory is designated as Root World Memory to ensure proper isolation and prevent unauthorized access. We modified the slice Linux kernel to implement a custom SMC request for IPIs, enabling multi-core functionality within slices. The slice Linux issues an SMC to the EL3M to activate the IPI on the target core, which is deactivated once the core receives it.

6 Security Evaluation

This section evaluates the security architecture of BARESLICE, with a focus on the TCB and the system’s resilience against potential attacks. We explore the various defense mechanisms embedded within BARESLICE designed to protect against these threats.

6.1 Trusted Computing Base

The TCB of BARESLICE comprises the EL3M, which is located within the Root World. Using the cloc [5] tool, we analyzed the codebase size pertinent to BARESLICE. We build upon the Trusted Firmware-A v2.8, which consists of approximately 383 KLoC. Our additional customizations comprise 1.4 KLoC to support the prototype of BARESLICE.

6.2 Security Analysis

BARESLICE is designed to secure slice operations from a range of attack types identified in our threat model, which assumes potential compromises by attackers via software, devices, malicious slices, or Realm VMs. Below, we highlight BARESLICE’s defensive capabilities against each attack scenario.

6.2.1 Attack with Normal or Secure World Software. Attackers may exploit software in the Normal or Secure Worlds to access sensitive slice data. BARESLICE counters this by isolating slices in the Realm World after initialization (§4.1), shielding them from compromised software, such as hypervisors or trusted OS. The GPC monitors physical memory accesses, blocking illegal memory mappings and MMU disable attempts, reporting violations to the EL3M via GPF exceptions.

Malicious IPIs sent by compromised privileged software to disrupt slice execution are mitigated by the EL3M, which disables IPI forwarding during slice operation by configuring GIC redistributor registers (§4.1.3). Unauthorized access attempts to these registers are intercepted by the EL3M.

6.2.2 Attack with Realm VMs or RMM. Compromised Realm VMs or RMM may target slices in the Realm World. To counter this, BARESLICE enforces two-way isolation using multiple GPTs (§4.1.1). Slice GPTs mark Realm VM and RMM memory as Root PAS, while host GPTs reciprocally mark slice memory as Root PAS, ensuring mutual isolation. Unauthorized access attempts are intercepted by the EL3M and reported via GPF exceptions.

6.2.3 Attack with Malicious Slices. BARESLICE mitigates malicious slice threats by assigning each slice a unique GPT (§4.1.1), designating its memory as Realm PAS and unauthorized regions as Root PAS. This ensures exclusive memory access for each slice, with unauthorized attempts blocked by the GPC and reported to the EL3M.

6.2.4 Attack with Devices. Attackers may use devices to launch malicious DMA requests. BARESLICE configures the SMMU GPT to restrict device-accessible memory regions. Unauthorized DMA requests targeting slice memory are blocked by the SMMU GPC and reported to the EL3M. Future integration of device attestation mechanisms, such as PCIe’s TEE Device Interface Security Protocol [43], can further enhance security by ensuring only uncompromised devices are assigned to slices.

Table 2: Applications for performance evaluation of BARESLICE.

Name	Description
AES [2]	AES data encryption.
OTP [1]	HMAC-based One-Time Password generation.
Apache [7]	Apache v2.4.57 with ApacheBench v2.3 to measure requests per second (100 concurrency).
Memcached [8]	Memcached v1.6.18 with mcperf v0.1.1 to measure transactions per second (10 connections).
Nginx [9]	Nginx v1.22.1 with ApacheBench v2.3 to measure requests per second (100 concurrency).
FileIO	SysBench v1.0.20 FileIO test (4 threads, 1GB, random read/write).

6.2.5 Attack with Side-Channels. Attackers could exploit side-channel attacks via stale GPT data cached in the TLB or shared cache. BARESLICE mitigates this by invalidating cached GPT data in TLBs after any GPT modification or slice lifecycle change, ensuring strict isolation. Additionally, dedicated physical cores and static memory allocations for slices mitigate many cache side-channel threats.

7 Performance Evaluation

We evaluate BARESLICE’s performance by analyzing real-world application overheads and GPT overhead. These metrics offer a comprehensive view of the system’s efficiency.

7.1 Experimental Setup

Although the FVP Base RevC-2xAEMvA platform supports Arm CCA features, it lacks the ability to provide cycle-accurate performance measurements. To obtain runtime performance results, we conduct our experiments on the Neoverse N1 System Development Platform (N1SDP), which includes two dual-core 2.6 GHz Neoverse N1 CPUs and 6 GB of RAM. Since the N1SDP does not support CCA-specific features such as the GPT, we follow best practices from previous studies [49, 56]. We replaced access to GPT-related registers with access to idle EL3 registers and kept all GPTs in memory. Note that we use the N1SDP to simulate performance overhead rather than implementing slice execution. Given that the N1SDP lacks support for Realm World and CCA hardware features, we simulate Slice Linux execution within the Normal World by incorporating artificial overhead for testing purposes, such as IPIs and GIC register access. Additionally, since BARESLICE relies on memory access traps triggered by the GPC, which is not available on the N1SDP, we introduce a custom SMC to notify the EL3M of GIC register access. Each experiment is conducted 10 times and we report the average. Due to the lack of hardware features specific to Arm CCA on the N1SDP, the performance metrics presented here may differ from those on actual CCA-enabled hardware once such hardware becomes available in the future. The Host Linux and the Slice Linux are both set up with 2 CPU cores and direct access to a network device, with 6 GB of RAM allocated to the Host Linux and 1 GB to the Slice Linux. Apart from the differences in RAM allocation and the artificial overhead introduced for experimental purposes, the Slice Linux shares the same configuration as the Host Linux. The Vanilla VM is configured using QEMU with 2 vCPUs, 1 GB of

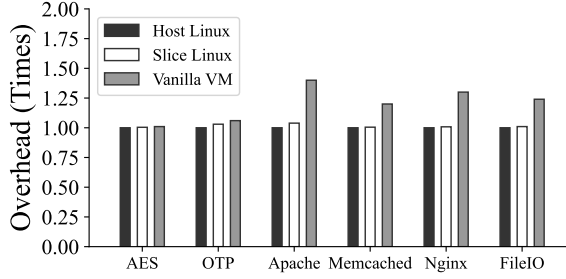


Figure 5: Performance overhead in real-world applications. BARESLICE achieves bare-metal performance while maintaining slice isolation.

RAM, and uses virtio-net for network forwarding and virtio-blk for block device emulation.

7.2 Real-world Application Overheads

We tested BARESLICE on six use cases detailed in Table 2, covering encryption, memory, and I/O-intensive tasks.

Figure 5 shows performance results on Host Linux, Slice Linux, and Vanilla VM, using Host Linux as the baseline. Generally, use cases in Slice Linux exhibit identical performance to Host Linux, attributed to bare-metal access with minimal overhead from IPI transfers and GIC register handling.

In contrast, Vanilla VM evaluations show varying levels of overhead. For tasks like OTP and AES, which are computation-intensive and mostly in user space, the overhead was moderate. However, Apache, Memcached, Nginx, and FileIO on Vanilla VM showed significant degradation, with overheads up to 1.4 times that of Host Linux. Conversely, network-bound tasks in Slice Linux show minimal overhead, as slices provide bare-metal resource access, allowing applications to perform similarly to non-virtualized Linux.

Note that Vanilla VM overhead with real CCA hardware may be lower due to support for direct virtual interrupt injection and device passthrough, reducing exits and associated overhead. Despite optimizations, workloads still suffer from virtualization overhead. A study [51] reported that up to 30% of CPU time was consumed by virtualization overhead in memory-intensive workloads.

Since no current hardware platform fully supports CCA, we did not directly measure Realm VM performance. However, based on its design, we expect higher overhead compared to Vanilla VM due to RMM involvement. A study [37] shows Realm VM adds up to 18% overhead for I/O-intensive workloads.

8 Discussion

Memory Encryption. In our design, we assume that slices derive security benefits from memory encryption, leveraging the same approach as Realm VMs—each slice could be assigned a distinct key for memory encryption. However, at present, there is no publicly accessible hardware supporting CCA features (including memory encryption) and the FVP simulator does not support the Memory Protection Engine component required for encrypting memory in the Realm World. Thus, our prototype does not support memory encryption for slices. We anticipate that enabling the Memory Protection Engine would enhance the security of slices without posing

conflicts with the design of BARESLICE. The introduction of memory encryption could potentially impact the performance overhead of slice execution (depending on how the Memory Protection Engine is implemented), requiring a reassessment of the performance overhead.

DMA Memory Protection. Since RME-DA is currently unavailable, to support DMA-capable devices, we could configure the DMA memory of each slice as Normal PAS in the SMMU GPT to enable DMA access. However, this introduces a potential vulnerability, as a malicious device could modify the contents of the DMA memory belonging to other slices. To mitigate this risk, we could use encryption to secure DMA memory. Devices that support encryption would write encrypted data to memory, and the slice could verify the data to ensure it originated from the correct device. Additionally, for each slice’s DMA memory, the Host and other slices’ GPT can set the DMA memory as Root PAS to protect it from CPU access, preventing CPU-based attacks from directly tampering with the DMA memory.

9 Related Work

Arm CCA Based Systems. Shelter [56] creates userspace enclaves at EL0 using the multi-GPT mechanism but is vulnerable to Iago attacks [20] due to reliance on an untrusted OS for system calls. In contrast, BARESLICE runs a Linux environment in independent slices, mitigating such risks. Shelter also incurs a performance overhead of up to 15%, whereas BARESLICE eliminates shared resources between slices, significantly enhancing performance to near bare-metal levels.

As CCA hardware is unavailable, efforts like Samsung Islet [12], virtCCA [54], and Huawei QEMU CCA [11] emulate its features. Notable advancements include GPU TEEs such as CAGE [52] and ACAI [49], which use GPC for memory protection, and Devlore [18], which allows Realm VMs to access integrated devices. These contribute to our secure computing framework.

Secure Hypervisor. Twinvisor [33] uses secure EL2 features to run secure hypervisors and VMs in the Secure World. DuVisor [21] and DeHype [30] aim to reduce the hypervisor’s attack surface by moving most functionality to user mode. HypSec [36] achieves this by splitting the hypervisor into a corevisor and untrusted hypervisor. NoHype [29] proposes eliminating the hypervisor entirely. In contrast, BARESLICE excludes the hypervisor from the TCB, ensuring slices only trust firmware and hardware, reducing attack surfaces.

10 Conclusion

In this paper, we present BARESLICE, a novel system that extends the capabilities of Arm CCA by integrating bare-metal support for confidential VMs. Our design leverages the RME hardware to provide robust isolation for slices with a minimal TCB. Through prototype implementation, we have developed and assessed BARESLICE, demonstrating the security of slices against diverse attacks while achieving bare-metal performance on real-world applications. Our work can support hypervisor execution and improves the state-of-the-art in delivering bare-metal performance in isolated computing environments.

References

- [1] DigisparkHOTP. <https://github.com/Akasurde/DigisparkHOTP>, 2016.
- [2] AES algorithm implementation. <https://github.com/dhuertas/AES>, 2020.
- [3] AMD Secure Encrypted Virtualization (SEV). <https://developer.amd.com/sev/>, 2021.
- [4] Arm fixed virtual platforms. <https://developer.arm.com/tools-and-software/simulation-models/fixed-virtual-platforms>, 2021.
- [5] cloc: Count lines of code. <https://github.com/AlDanial/cloc>, 2021.
- [6] The Realm Management Extension (RME), for SMMUv3. <https://developer.arm.com/documentation/dhi0094/latest/>, 2021.
- [7] Apache http server. <https://www.apache.org/>, 2022.
- [8] Memcached. <https://github.com/memcached/memcached>, 2022.
- [9] Nginx. <https://github.com/nginx/nginx>, 2022.
- [10] Trusted-Firmware-A. <https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/>, 2022.
- [11] Huawei_CCA_QEMU. https://github.com/Huawei/Huawei_CCA_QEMU, 2023.
- [12] Samsung Islet. <https://github.com/islet-project/islet.git>, 2024.
- [13] Zunaid Aalam, Vinod Kumar, and Surendra Gour. A review paper on hypervisor and virtual machine security. In *Journal of Physics: Conference Series*, volume 1950, page 012027. IOP Publishing, 2021.
- [14] Amazon Web Services. The Security Design of the AWS Nitro System. <https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.html>, 2024.
- [15] ARM. Arm Confidential Compute Architecture. <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>, 2021.
- [16] ARM. Arm TrustZone Technology. <https://developer.arm.com/ip-products/security-ip/trustzone>, 2021.
- [17] ARM. Deep dive into the Arm Confidential Compute Architecture. <https://static.linaro.org/connect/armcca/presentations/CCATechEvent-210623-CGT-2.pdf>, 2021.
- [18] Andrin Bertschi, Supraja Sridhara, Friederike Groschupp, Mark Kuhne, Benedict Schlüter, Clément Thorens, Nicolas Dutly, Srdjan Capkun, and Shweta Shinde. Devlore: Extending arm cca to integrated devices a journey beyond memory to interrupt isolation. *arXiv preprint arXiv:2408.05835*, 2024.
- [19] Robert Bühren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. One glitch to rule them all: Fault injection attacks against amd's secure encrypted virtualization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2875–2889, 2021.
- [20] Stephen Checkoway and Hovav Shacham. Iago attacks: Why the system call api is a bad untrusted rpc interface. 2013.
- [21] Jiahao Chen, Dingli 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, 2023.
- [22] Yiquan Chen, Jiexiong Xu, Chengkun Wei, Yijing Wang, Xin Yuan, Yangming Zhang, Xulin Yu, Yi Chen, Zeke Wang, Shuibing He, et al. Bm-store: A transparent and high-performance local storage architecture for bare-metal clouds enabling large-scale deployment. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1031–1044. IEEE, 2023.
- [23] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia. Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svd voltage scaling interface. In *30th USENIX Security Symposium (USENIX Security)*, 2021.
- [24] Christoffer Dall, Shih-Wei Li, and Jason Nieh. Optimizing the design and implementation of the linux {ARM} hypervisor. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 221–233, 2017.
- [25] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. 2018.
- [26] Davood Ghatreh Samani, Chavit Denninnart, Josef Bacik, and Mohsen Amini Salehi. The art of cpu-pinning: Evaluating and improving the performance of virtualization and containerization platforms. In *Proceedings of the 49th International conference on parallel processing*, pages 1–11, 2020.
- [27] Felicitas Hetzelt and Robert Bühren. Security analysis of encrypted virtual machines. *ACM SIGPLAN Notices*, 52(7):129–142, 2017.
- [28] Intel Corporation. Intel trust domain extensions, 2014.
- [29] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th annual international symposium on Computer architecture*, 2010.
- [30] Taehoon Kim, Kwangwon Koh, Changdae Kim, Eunji Pak, Yeonjeong Jeong, and Sang-Hoon Kim. Dehype: Retrofitting hypervisors for a resource-disaggregated environment. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 37–48. IEEE Computer Society, 2023.
- [31] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. Dawg: A defense against cache timing attacks in speculative execution processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.
- [32] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In *29th USENIX Security Symposium (USENIX Security)*, 2020.
- [33] Dingli 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)*, 2021.
- [34] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A systematic look at ciphertext side channels on amd sev-snp. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 337–351. IEEE, 2022.
- [35] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected {I/O} operations in {AMD's} secure encrypted virtualization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1257–1272, 2019.
- [36] Shih-Wei Li, John S Koh, and Jason Nieh. Protecting cloud virtual machines from hypervisor and host operating system exploits. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1357–1374, 2019.
- [37] 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 *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.
- [38] Zheng Li, Maria Kihl, Qinghua Lu, and Jens A Andersson. Performance overhead comparison between hypervisor and container based virtualization. In *2017 IEEE 31st International Conference on advanced information networking and applications (AINA)*, pages 955–962. IEEE, 2017.
- [39] Microsoft. Managing Hyper-V hypervisor scheduler types: The core scheduler. <https://docs.microsoft.com/windows-server/virtualization/hyper-v/manage/manage-hyper-v-scheduler-types#the-core-scheduler>, 2023.
- [40] Microsoft. Microsoft Azure. <https://docs.microsoft.com/en-us/azure/virtual-machines/acu>, 2024.
- [41] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019.
- [42] Meni Orenbach, Andrew Baumann, and Mark Silberstein. Autarky: Closing controlled channels with self-paging enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, 2020.
- [43] PCI-SIG. TEE Device Interface Security Protocol. <https://pcisig.com/tee-device-interface-security-protocol-tdisp>, 2022.
- [44] Michael Pearce, Sherali Zeadally, and Ray Hunt. Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys (CSUR)*, 45(2):1–39, 2013.
- [45] Hany Ragab, Andrea Mambretti, Anil Kurmus, and Cristiano Giuffrida. Ghosttrace: Exploiting and mitigating speculative race conditions. In *USENIX Security*, 2024.
- [46] Benedict Schlüter, Supraja Sridhara, Mark Kuhne, Andrin Bertschi, and Shweta Shinde. Heckler: Breaking confidential vms with malicious interrupts. In *USENIX Security*, 2024.
- [47] Amazon Web Services. Amazon EC2. <https://aws.amazon.com/ec2/>, 2024.
- [48] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS*, 2017.
- [49] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. Acai: Protecting accelerator execution with arm confidential computing architecture. In *USENIX Security*, 2024.
- [50] Darshan Tank, Akshai Aggarwal, and Nirbhay Chaubey. Virtualization vulnerabilities, security issues, and solutions: a critical study and comparison. *International Journal of Information Technology*, pages 1–16, 2019.
- [51] Boris Teabe, Peterson Yuhala, Alain Tchana, Fabien Hermenier, Daniel Hagimont, and Gilles Muller. (no) compromis: Paging virtualization is not a fatality. In *Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 43–56, 2021.
- [52] Chenxu Wang, Fengwei Zhang, Yunjie Deng, Kevin Leach, Jiannong Cao, Zhenyu Ning, Shoumeng Yan, and Zhengyu He. Cage: Complementing arm cca with gpu extensions. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium*, 2024.
- [53] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. Pwrleak: Exploiting power reporting interface for side-channel attacks on amd sev. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 46–66. Springer, 2023.
- [54] Xiangyi Xu, Wenhao Wang, Yongzheng Wu, Zhennan Min, Zixuan Pang, and Yier Jin. virtcca: Virtualized arm confidential compute architecture with trustzone. *arXiv preprint arXiv:2306.11011*, 2023.
- [55] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd Austin. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.
- [56] Yiming Zhang, Yuxin Hu, Zhenyu Ning, Fengwei Zhang, Xiapu Luo, Haoyang Huang, Shoumeng Yan, and Zhengyu He. Shelter: Extending arm cca with isolation in user space. In *32nd USENIX Security Symposium (USENIX Security'23)*, 2023.

- [57] Ziqiao Zhou, Yizhou Shan, Weidong Cui, Xinyang Ge, Marcus Peinado, and Andrew Baumann. Core slicing: closing the gap between leaky confidential vms and baremetal cloud. In *Proceedings of the 17th Symposium on Operating System Design and Implementation (OSDI)*, 2023.