PORTAL: Fast and Secure Device Access with Arm CCA for Modern Arm Mobile System-on-Chips (SoCs)

Fan Sang[1], Jaehyuk Lee[1], Xiaokuan Zhang[2] and Taesoo Kim[1]
[1]*Georgia Institute of Technology,* [2]*George Mason University*

Abstract*—The increasing integration of diverse co-processors and peripherals within mobile Arm System-on-Chips (SoCs) presents significant challenges for secure and efficient device I/O. Existing approaches relying on memory encryption introduce substantial performance and power overheads, which are exacerbated by the need for real-time data processing and strict power efficiency requirements in mobile platforms. These issues hinder the wider adoption of Arm Confidential Compute Architecture (CCA), which aims to provide robust security guarantees. To address these challenges, we present PORTAL, a secure and efficient device I/O interface for Arm CCA on mobile Arm SoCs. PORTAL achieves secure I/O through strict memory isolation without the need for memory encryption. By leveraging the memory isolation mechanism in Arm CCA, PORTAL enforces hardware-level access control, ensuring that only designated Realm virtual machines and peripherals can access the PORTAL-protected plaintext memory regions. This design eliminates the overhead associated with encryption, supports dynamic peripheral integration, and maintains robust security guarantees. The evaluation results demonstrate that PORTAL incurs a minimal one-time overhead of 9.8%, while enhancing scalability and power efficiency, making it a pivotal solution for fostering the adoption of the upcoming Arm CCA in mobile and resource-constrained environments.

1. Introduction

Arm's CPU architecture dominates mobile processors with a 99% market share, 40.8% in automotive, and an overall 48% share in related markets [27]. The historical trend in the development of mobile Arm processors, has been characterized by an increasing integration of diverse co-processors, peripherals, and devices alongside Central Processing Unit (CPU). This trend began with the integration of basic elements such as memory management units and has evolved to include a wide range of specialized components such as Graphic Processing Units (GPUs), Neural Processing Units (NPUs), reality processors (e.g., Apple R1 chip [1]) and various communication modules [60], [74], [8], [65], [50], [51], [57]. As the evolution of mobile Arm processors continues, these advancements are supporting emerging computing platforms that demand low latency, high power efficiency, substantial computational power, and maximum space efficiency within a single chip, such as in mobile devices, virtual reality, autonomous vehicles, and edge computing applications.

Meanwhile, the security measurement on Arm processors has also leaped forward significantly. Modern advancements in Confidential Computing [16], [30], [44], [5] have introduced confidential Virtual Machines (VMs) [4], [35], [34] which protect both the application and the entire VM software stack, including the guest Operating System (OS). Arm recently announced its adoption of confidential VMs, called Arm Confidential Compute Architecture (CCA) [10]. Due to the strong security guarantees of confidential VMs, devices are untrusted and cannot directly access the protected memory allocated for confidential VMs. Workarounds for Arm CCA facilitate device communication through an untrusted *shared memory* region, such as `virtio` [39] or bounce buffer [48]. To meet the security requirements of confidential VMs, data in transit is encrypted when in untrusted shared memory.

Arm CCA aims to provide robust security mechanisms, but struggles to keep up with the trend of mobile Arm System on Chips (SoCs) [20], hindering its wider adoption upon the upcoming release.

The scalability and performance of Arm CCA are increasingly challenged as the number of integrated devices within mobile Arm System on a Chip (SoC) grows. For instance, customized Arm processors in modern autonomous vehicles [74], [14], [49], [45], [59], featuring CPUs, GPUs, sensor fusion processors, and AI accelerators (e.g., NPUs), exemplify the complexity an Arm SoC must handle. When these devices concurrently access encrypted memory shared by all devices and the CPU, significant performance degradation occurs. In addition, memory encryption naturally conflicts with existing optimizations such as deduplication [62]. Encryption makes identical data blocks appear different, leading to inefficient memory usage, especially in systems requiring rapid, frequent access to shared data. These drawbacks impact real-time data processing in resource-constrained platforms such as autonomous vehicles, where minor lags can compromise safety and efficiency. Furthermore, point-to-point memory encryption in current secure device Input/Output (I/O) approaches for Arm CCA [25], [82], [69], [76] ties the device to the entire lifespan of the confidential VM, preventing dynamic device assignment during runtime, a common scenario when integrated devices on mobile SoCs are shared by multiple tenants. Lastly, unlike desktop platforms, mobile Arm platforms operate under stringent power efficiency requirements (e.g., battery). Repetitive memory encryption and decryption significantly increase energy consumption [61], [46], [3], [55], [72], [32], [47]. The growing number of

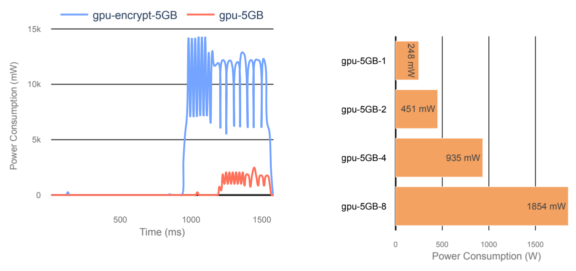

Figure 1: Overhead incurred by memory encryption with software-based 128-bit AES-GCM. Left: power usage and execution time (indicated by spikes) when copying 5GB of memory with and without encryption. Right: additional power consumption for multiple GPUs due to p2p memory encryption. Experiment conducted on Apple M1 SoC with 16GB unified memory.

integrated devices that require secure interactions exacerbates this problem when memory is shared in the mobile Arm SoCs, as shown in our Apple M1 SoC experiment [7] in Figure 1.

We propose a bold approach: *achieving Arm CCA secure device I/O by sole isolation without memory encryption*. Historically, memory encryption is deployed in Confidential Computing because if any of the isolation techniques have been compromised, the data being accessed is still protected by cryptography. One particular case is to prevent physical attacks such as memory probing [71], [29], [68], [38] and cold boot attacks [33], [31], [80], [40], as isolation does not extend to the memory bus. In other words, memory encryption can be redundant if memory isolation is *strictly* and *always* enforced. Thanks to the *integrated memory* design adopted in common mobile Arm SoCs, we argue that encrypting the shared memory between the confidential VMs and the peripheral devices is unnecessary. On the one hand, external attackers cannot launch physical attacks by probing the memory bus, as the integrated memory design connects the memory directly to the CPU and devices within the Arm SoC [66], [81], [24], which is commonly equipped with tamper detection against unauthorized physical accesses. On the other hand, unlike traditional process-based Confidential Computing (CC), Arm CCA enforces strict hardware-based memory access control even for privileged software (i.e., OS and Virtual Machine Monitor (VMM)) so that privileged attackers cannot access protected memory regions [16], [10]. As a result, memory encryption is unnecessary if no other entities except designated confidential VMs and devices can read or write to the shared memory region. In return, it is possible to achieve plaintext secure device I/O, which enhances Arm CCA for mobile platforms by 1) boosting the performance for secure data transmission, 2) retaining scalability with the increasing trend of integrated devices, and 3) reducing burden on energy efficiency.

PORTAL. In this paper, we present PORTAL, a secure and efficient device I/O interface for Arm CCA on mobile Arm SoCs. PORTAL achieves secure I/O by strict memory isolation without memory encryption. In essence, PORTAL provides a plaintext memory region whose access control is strictly enforced by Arm CCA so that no parties except the designated Realm virtual machines (Realm VMs) and peripherals can access it. PORTAL leverages the capabilities of Arm CCA's Granule Protection Check (GPC) and System Memory Management Unit (SMMU) to enforce hardware-level memory isolation. GPC enforces strict access control on the host physical address space and is mandatory for any component that generates memory transactions under the CCA model. SMMU provides integrated devices with virtual to physical address translation and access controls when accessing the host memory. The two trusted access control components collaborate to enforce two-way exclusive memory access between designated Realm VMs and devices, achieving secure I/O by isolation without encryption.

To ensure the integrity of the isolation, PORTAL employs a specialized Realm VM to protect the configuration and management of the sensitive data structures (e.g., translation tables) that establish the isolation. As PORTAL removes the memory encryption and adopts a plaintext-based approach, we also conduct an in-depth security analysis of PORTAL and make sure that PORTAL does not open new attack surfaces.

We prototype PORTAL on two official platforms, the Arm Fixed Virtual Platforms (FVPs) emulator [15] for software-simulated Arm CCA features and the Orange Pi 5 Plus [53] with Arm Mali-G610 GPU. Evaluation results show that PORTAL-based secure I/O incurs a minimal one-time overhead of 9.8% with runtime device management and can achieve a significant performance gain (1.07×-9.07× on selected benchmarks) on data-intensive applications compared to memory encryption solutions.

PORTAL tries to foster a broader adoption of Arm CCA on the most widely deployed mobile processor architecture upon its official release in the near future. PORTAL serves as the first plaintext secure I/O interface for Arm CCA to achieve performance, scalability, and power efficiency on demanding mobile Arm platforms. PORTAL will be publicly available as an open-source project, allowing communities to test and contribute towards a more practical secure I/O for peripheral access on mobile Arm CC platforms.

Contributions. This paper makes following contributions:

- **New approach.** We present PORTAL, the first plaintext-based secure I/O interface for mobile Arm CCA that ensures efficiency, scalability, and power efficiency.
- **Fostering broader adoption of Arm CCA.** We point out the current challenges of Arm CCA in the trend of increasing integration of diverse devices on mobile Arm devices, the most widely adopted computing platform. PORTAL boosts the practicality of Arm CCA within the architectural trend of mobile Arm processors, fostering a wider adoption of Arm CCA upon its upcoming official release.
- **Comprehensive evaluation.** Our evaluation shows that PORTAL is feasible and achieves its performance, scalability, and power efficiency goal. As PORTAL adopts plaintext instead of encrypted shared memory for data transmission, we also conduct comprehensive security analysis of PORTAL to demonstrate its security.

Security State	NS PAS	Secure PAS	Realm PAS	Root PAS
NS	✓	×	×	×
Secure	✓	✓	×	×
Realm	✓	×	✓	×
Root	✓	✓	✓	✓

TABLE 1: Accessibility of physical memory pages having different Physical Address Space (PAS) from different processor security states. NS stands for Non-Secure.

2. Background

2.1. Arm Integrated Memory

Integrated memory is a configuration where the memory subsystem is embedded onto the processor's silicon. This design allows shared access between the CPU and peripherals like GPUs, NPUs, Digital Signal Processors (DSPs), and co-processors, optimizing space, power efficiency, and data access speeds.

Integrated memory is prevalent in Arm processors, especially in mobile and embedded systems where compact design and energy efficiency are crucial [60], [74], [8], [65], [50], [51], [57]. This architecture is standard in smartphones, tablets, and portable devices [65], [57], which dominate the mobile processor market and benefit from the space efficiency and power savings of integrated memory. In addition, integrated memory is increasingly used in automotive systems [74], [14], [49], [45], [59], extended reality headsets [60], [1], industrial machines [50], and edge computing devices [51], [8]. This trend shows the demand for efficient, responsive, and robust computing across industries, making integrated memory essential in modern Arm processors.

Feasibility of physical memory attacks. Considering physical attacks [71], [29], [68], [38], [33], [31], [80], [40] on Arm systems with integrated memory, several factors render these attacks impractical. The compact and integrated nature of Arm SoCs makes memory components physically inaccessible without specialized tools and risk of significant damage [66], [81], [24]. Advancements in packaging technology [26], [77] protect these components by making direct memory access exceedingly difficult without sophisticated equipment. Furthermore, security features such as tamper detection [18], [67], [64], [43], [79] safeguard against unauthorized physical access by triggering automatic data deletion or device locking upon tampering attempts. These protections greatly reduce the chances of successful physical attacks on integrated memory.

2.2. Arm CCA

Arm CCA [16], [10] enables the creation of VM-based trusted execution environments via a hardware extension to the Armv9 Instruction Set Architecture (ISA) [9], referred to as Realm Management Extensions (RME) [12]. RME introduces two new worlds, *Realm* and *Root*, in addition to the existing *Normal* and *Secure* worlds. CCA manages these worlds by implementing a Granule Protection Table (GPT), a crucial data structure that assigns physical addresses (or

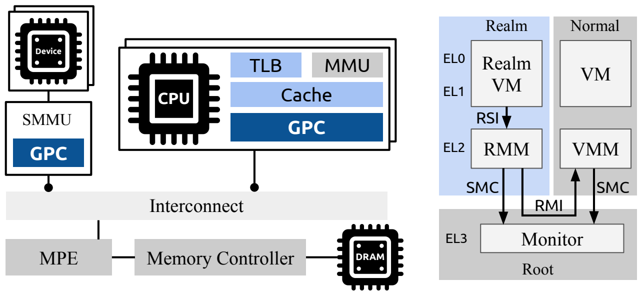

Figure 2: GPC in Arm CCA. Blue components are subject to GPC enforcement. The right figure shows interfaces between the Realm VM, Realm Management Monitor (RMM), VMM, and Monitor. The SMC instruction allows the RMM and VMM (EL2) to return control to the Monitor (EL3). Realm service interface (RSI) is the channel for requesting services from the RMM, and realm management interface (RMI) is the communication channel from the host to the RMM.

Physical Address Spaces (PAS) in CCA terminology) to one of the four worlds, where a *granule* is the smallest unit of memory that can be managed. This setup enhances isolation by equipping Processing Elements (PE) (i.e., Arm cores, SMMU, caches, and Translation Lookaside Buffers (TLBs)) with GPC, which check if the source and destination PAS conform to the access control policies (see Table 1). For example, when a CPU core executes in Realm mode, it can generate memory transactions for the Realm PAS. If an invalid transaction is detected, CCA triggers a Granule Protection Fault (GPF) and handles it in EL3.

The GPTs, stored in the main memory under the *Root* world, are accessible only by the trusted Monitor, which updates them as necessary. Changes to the GPTs trigger synchronization of the GPC, which requires flushing outdated states to maintain consistency. These checks are pivotal in preventing unauthorized access to the *Realm* memory from the untrusted *Normal* or *Secure* worlds by controlling all memory accesses and ensuring that memory transactions initiated from the core are securely managed.

The trusted Monitor at EL3 in the Root world adjusts the world bit during context switches, enhancing security. Concurrently, the trusted RMM at EL2 in the *Realm* world isolates confidential VMs by managing stage-2 translations from guest to host physical addresses. Lastly, memory encryption and integrity protection prevent physical tampering with main memory data.

2.3. Arm Peripherals and SMMU

Unlike discrete peripherals in Intel-based systems, Arm-based devices use a unified memory architecture (§2.1) shared with the CPU and other peripherals (e.g., GPU and NPU). Arm handles data transfers and communication between the host and device through comprehensive software, including kernel drivers and user runtimes. This software manages the device computation environment and facilitates hardware interactions.

To set up the execution environment, the device software allocates physical memory and creates buffers for specific tasks. It then loads essential task elements into device memory. Additionally, the software stack creates the page table and configures registers for Direct Memory Access (DMA) to access critical components. Interaction with hardware is managed via task scheduling and submission through Memory-Mapped Input/Output (MMIO). After computations, the software retrieves results and restores the system environment.

Given the shared memory architecture, Arm has integrated the SMMU to oversee DMA-capable peripherals. Most Arm GPUs [11], [52], [56] and other peripherals are connected to an SMMU. Similar to the CPU's Memory Management Unit (MMU), the SMMU [13] performs stage-1 and stage-2 address translations to regulate peripheral access to the physical address space. Privileged software configures the SMMU registers, including page table and translation configuration registers, through MMIO. Besides address translation, the SMMU supports GPC under Arm CCA. To secure these features, CCA adds SMMU MMIO registers accessible only in the *Root* world, offering configurations for SMMU GPC, such as GPT base, GPC controls, fault handling, and TLB invalidation.

Device identity management. Each device connected to the SMMU is assigned a unique *Stream ID (SID)* included in its memory access requests. The *Stream Table (ST)* maps SIDs to their corresponding *Context Descriptors (CDs)*, which contain the virtual-to-physical address translation table. To retrieve the physical address, the SMMU uses the device's SID to traverse the ST and CD for the translation table, which is then used to find the physical address. Besides address translation, the ST and CD in the SMMU manage memory access, access control, and security for connected devices.

3. Overview

3.1. Motivation

As more devices are integrated into Arm SoCs, managing secure environments becomes increasingly complex. Including many devices in a single Arm SoC presents challenges for Arm CCA's practicality upon its upcoming release to mobile Arm SoCs.

Limitations of existing approaches. Specifically, existing solutions for secure device I/O in Arm CCA do not align with the ongoing architectural evolution of Arm SoCs for several reasons:

1) Performance and TCB overhead: Existing shared memory solutions [76], [39], [48] rely on software-based memory encryption which causes significant performance overhead (more than 40%), defeating the original goals of using devices such as accelerators, while recent study [69] requires non-trivial modification on hypervisor hardware and significantly bloats the Trusted Computing Base (TCB). Worse, memory encryption naturally hinders existing mature performance optimizations such as memory deduplication [62], opposing mobile platforms that demand real-time processing.

2) Inscalability: The performance overhead increases when multiple peripheral devices access unified memory simultaneously [60], [74], [8], [65], [50], [51], [57]. Each device must encrypt every memory access and undergoes complex key management overhead for multiple sessions. Current solutions do not scale with the increasing integration of devices in Arm SoCs, hindering the adoption of Arm CCA in the most prevalent mobile processor market.

3) Inflexibility: In order to enforce exclusive access, existing solutions employ one-to-one binding [25], [82], [69], [76] between the VM identity and the target device, which is persist throughout the VM's lifespan. Such a direct binding hinders the support of multiple peripherals or dynamic attachment of peripherals.

4) Power inefficiency: Mobile Arm platforms operate under stringent power efficiency requirements. The computational overhead associated with frequent memory encryption for various peripheral devices increases energy consumption substantially [61], [46], [3], [55], [72], [32], [47], making Arm CCA even less appealing for mobile Arm platforms.

5) Lack of device generality: Constrained devices that lack the capability to support cryptographic operations, including key negotiation and secure encryption, cannot be integrated into existing solutions that demand point-to-point encryption.

Reevaluating memory encryption in Arm SoCs. Historically, Memory encryption arose to secure data in transit and at rest within complex processor and memory architectures. This was crucial in shared and cloud computing environments to protect data from unauthorized access at the hardware level. Given the strong physical security and low risk of physical attacks on Arm SoCs, the need for memory encryption in Arm CCA for mobile processors should be reconsidered. In environments where processors are not physically threatened, strict memory access controls (e.g., hardware-based GPC in Arm CCA) may provide sufficient protection without the drawbacks of memory encryption. We believe that carefully redesigning Arm CCA memory isolation can ensure data security without memory encryption, maintaining performance, scalability, and power efficiency for mobile Arm SoCs.

3.2. Targeted Platforms

We consider SoC-based Arm processors where the memory is embedded on chip (i.e., integrated memory) and is shared between the CPU and peripheral devices (i.e., unified memory model), a common configuration for mobile Arm processors. We assume that communication within the Arm processor package is secure and physical attacks are infeasible, owing to its built-in resilience against tampering and interference with interconnections (§2.1). As a result, the data in transit among the memory, the processors, and peripherals on chip, remains secure against physical memory attacks.

3.3. Threat Model

We assume that the next-generation Arm SoC will incorporate security primitives including the RME, CCA, and

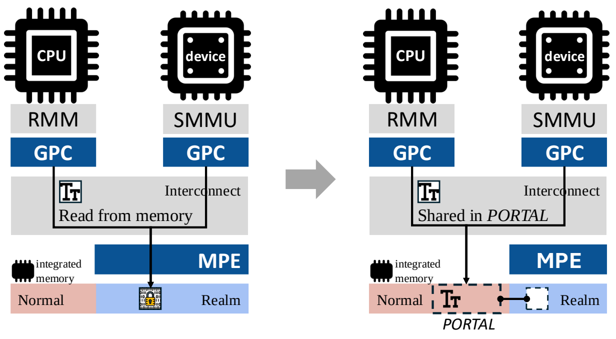

Figure 3: Access modes of integrated devices. Left: encrypted memory in native Arm CCA. Right: plaintext memory access through PORTAL.

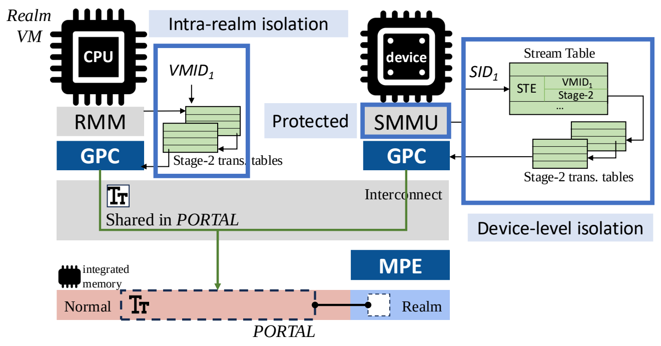

Figure 4: Protected PORTAL regions.

a hardware root of trust to support secure boot and remote attestation. We trust the Realm and the Root world that hosts the RMM and the Monitor, conforming to the TCB of Arm CCA. We assume a strong attacker who controls both the *Normal* world and the *Secure* world, including the host OS and the VMM. The attacker aims to leak or manipulate the sensitive data transmitted between the confidential VM and the peripheral device. The attacker can access the unified memory holding the sensitive data or employ DMA-capable peripherals to execute similar attacks. Additionally, the attacker might disrupt the isolated execution environment by compromising memory management and modifying the states of device registers. The attacker could also introduce malicious device tasks to access or tamper with sensitive data within other realms. We assume that all platform devices integrated into the SoC packages are not forgeable. We do not consider denial-of-service attacks from the malicious hypervisor or host OS. We also do not consider speculation attacks and side-channel leakage due to microarchitectural implementation.

3.4. PORTAL Overview

The core of PORTAL is a strictly isolated plaintext shared memory region, called PORTAL region, for data transmission between Arm CCA confidential VMs (i.e., Realm VMs) and integrated devices on Arm SoCs. PORTAL ensures that only dedicated Realm VMs and peripherals can access the PORTAL region, while unauthorized entities, including privileged software (i.e., VMM and host OS), are prohibited from access. PORTAL removes the requirement for memory encryption against physical attacks by relying on the robustness of Arm SoC packages.

Specifically, when authorized confidential VMs and peripherals initiate memory transactions to the PORTAL region, PORTAL configures their GPTs so that the PORTAL region is treated as the Normal world PAS, allowing them to issue memory transactions without encryption. The CPU and SMMU stage-2 translation tables are utilized to ensure Realm VM-level and device-level isolation. In contrast, the PORTAL region is configured as the Root world PAS for unauthorized entities, blocking their accesses.

Challenges. The design of PORTAL faces three key challenges. **C1:** The current SMMU design lacks Realm VM context (i.e., whether a device belongs to a Realm VM) and cannot issue memory transactions targeting the Realm PAS. Meanwhile, the SMMU hosts multiple page tables in system memory (i.e., *Normal world* and *Secure world*), which is considered untrusted under Arm CCA. PORTAL thus establishes an isolated memory region in the Normal world PAS for device communication, while securing it by restricting access to authorized entities. **C2:** It is critical to ensure PORTAL does not bloat the TCB of trusted components of Arm CCA. PORTAL uses a dedicated Realm VM, the *System Realm*, for critical management tasks (e.g., configuring the SMMU), preserving the original TCB of EL2 and EL3 without expanding the RMM in EL2 or the Monitor in EL3. **C3:** As PORTAL removes memory encryption and adopts a plaintext shared memory region for device communication, the strict isolation enforced by PORTAL must span all layers of the Arm CCA stack to maintain data integrity and confidentiality. We conduct an in-depth security analysis on PORTAL, and demonstrate that PORTAL does not open new attack surfaces and the design decisions are well justified (§6.2).

4. Design

4.1. Protected Memory Regions

PORTAL region. PORTAL strictly isolates a shared memory region, called PORTAL *region*, for a particular pair of authorized Realm VM and device to achieve secure I/O without encryption. However, the existing SMMU under CCA is not aware of the Realm world context and all devices are treated in the *Normal* world. Therefore, a PORTAL region must reside in the Normal world to be accessible to both the Realm VM and the device. As shown in Figure 4, PORTAL utilize the stage-2 translation tables and SMMU translation tables to enforce memory isolation. The stage-2 translation table for Realm VMs is managed by the trusted RMM and is used to isolate mutually distrusting Realm VMs. PORTAL uses the RMM's stage-2 translations table to ensure that when a PORTAL region is mapped to a particular Realm VM, it will not be accessible by other Realm VMs that do not possess the

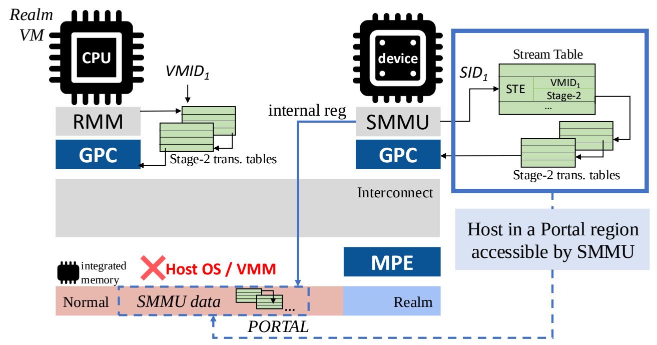

Figure 5: Protecting the SMMU data structures.

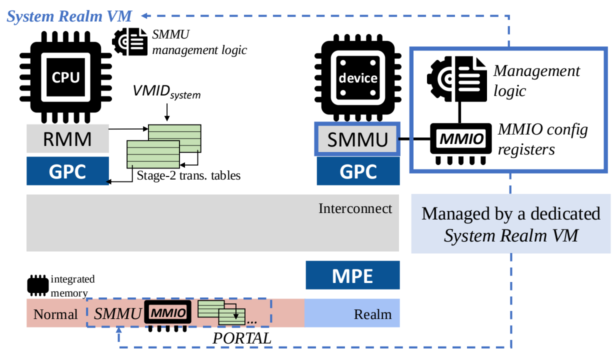

Figure 6: Protecting SMMU management.

mapping. As a result, even though GPC allow any Realm VM to access a PORTAL region, which is within the Normal world PAS, RMM's stage-2 translation table ensures the isolation of the PORTAL region from unauthorized Realm VMs. Similarly, PORTAL configures the SMMU translation table to ensure that only the authorized device has the mapping to the specific PORTAL region in its stage-2 translation. However, as the SMMU is managed by the untrusted host OS and the VMM, it can be manipulated by the attackers to allow attacker-controlled devices to access unauthorized PORTAL regions, demanding effective protection measurements. We discuss how PORTAL protects the SMMU in Section §4.2.

4.2. Protection of the SMMU

Protecting the SMMU data structures. The SMMU is not a part of the Arm CCA's TCB and is originally managed by the untrusted host OS and the VMM. Hosting the SMMU data structures (i.e., Stream Tables and stage-2 translation tables per device) in highly privileged software layers such as the trusted Monitor or the RMM can prevent access from the attackers. However, the SMMU, without the corresponding permission, cannot perform accesses to the SMMU data structures located in either the RMM or the Monitor. Therefore, PORTAL hosts the SMMU data structures in a PORTAL region (EL1) (Figure 5), which resides in the Normal world PAS and can be freely accessed by the SMMU, while isolated from the host OS and the VMM to prevent malicious manipulations.

Protecting SMMU management. In addition to protecting the SMMU data structures, the code logic that manages the SMMU should be secured as well. Specifically, originally located in the untrusted host OS and VMM, the SMMU management code should be relocated to memory regions inaccessible by the attackers. Existing approaches [76], [69] choose to move the SMMU management code to the trusted Monitor or the RMM to achieve intrinsic security offered by Arm CCA. However, as the Monitor and the RMM are the most security-critical layers of Arm CCA, introducing the SMMU management code will enlarge the TCB and might cause total compromise of Arm CCA (i.e., both the Root and the Realm world) if security flaws exist in the added logic. In addition, as the RMM and the Monitor are located in the high privileged exception levels, it incurs additional overhead of context switching and TLB/cache flushing whenever the Realm VM, the host OS, or the VMM request SMMU updates.

PORTAL employs a dedicated Realm VM called *System Realm* (EL1) to manage the SMMU on behalf of the host OS and the Realm VM that is authorized to communicate with the device, shown in Figure 6. The System Realm owns the PORTAL region (EL1) that hosts the SMMU data structures. The System Realm also has exclusive access to the MMIO registers that configure the SMMU and its data structures, as these registers can be used by the attackers to compromise the security of PORTAL (e.g., turn off stage-2 translations or the SMMU entirely). This is achieved by mapping the MMIO configuration registers only to the System Realm's translation table, forcing every SMMU-related operation to be routed to and handled by the System Realm. Note that the Stream Table I/O page tables are set up using the memory pages that belong to the System Realm, and are inherently only accessible by the System Realm itself, the RMM, and the Monitor.

We assume that the System Realm is implemented and distributed by trusted vendors such as Arm or SoC vendors. Since the System Realm is also an instance of Realm VM, it benefits from the existing Arm CCA security guarantees. First, based on the attestation report provided by Arm CCA, the Realm VM wishing to access the PORTAL region can verify whether the System Realm is provisioned by the trusted vendors. Meanwhile, the measurement provided in the attestation report validates whether the expected code logic (e.g., no intentional information leak) is running in the System Realm. In addition, potential vulnerabilities in the System Realm (EL1) will not compromise the RMM (EL2) and the Monitor (EL3), unlike the existing approaches. Lastly, to reduce the overhead caused by frequent world switching and exception level changes, PORTAL provides an exclusive interface to interact with the System Realm. Through this interface, Realm VMs are able to send commands to the System Realm to configure the SMMU on demand. We provide more details about the PORTAL System Realm interface in Section §4.4.

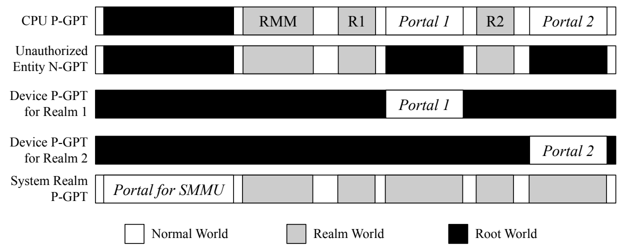

Figure 7: Memory accessibility for authorized and unauthorized entities in different security states (i.e., Realm or Normal) based on GPT configurations.

4.3. Memory Isolation

All memory requests generated by peripheral devices are checked by the SMMU for access control and address translations provided in the I/O page table. As PORTAL regions belong to the Normal world PAS, memory transactions from devices to a PORTAL region are marked as Normal world transactions. However, since multiple devices can be attached to the same SMMU, a device can access any existing PORTAL region using Normal world memory transactions, even when the particular PORTAL region is established for other devices. PORTAL thus requires device-level isolation within the Normal world to achieve two-way isolation between a dedicated pair of Realm VM and device, which can be achieved through the existing access control mechanism provided by the SMMU.

Intra-realm and device-level isolation. Arm CCA GPC enforce access control at world granularity, which is not sufficient to isolate mutually distrusting peripherals. PORTAL employs stage-2 translations to enforce fine-grained isolation. Within PORTAL's model, stage-2 translations have two sides: CPU side and device side, as shown in Figure 4. On the CPU side, the RMM isolates mutually distrusting Realm VMs using stage-2 translations, which are stored in the protected Realm memory (EL2) inaccessible to the host OS and the VMM. On the device side, similarly, to enable the secure device I/O at device granularity, PORTAL uses the SMMU stage-2 translations to isolate mutually distrusting devices from accessing each other's PORTAL regions. Since the isolation guarantee solely depends on correctly identifying the world to which the VM and the device belong, the identity of VMs and devices must be unforgeable. PORTAL uses the VMID of each VM and the SID of each device to identify VMs and devices, respectively. Both identifications belong to the SMMU data structures and are protected by the dedicated PORTAL region (EL1) owned by the System Realm, allowing its exclusive management on behalf of other Realm VMs as described in §4.2. Moreover, the configuration of the SMMU is managed by the PORTAL System Realm instead of the untrusted host OS and the VMM.

Isolating authorized and unauthorized entities. PORTAL designs two types of custom GPTs, PORTAL-GPT (P-GPT) and Normal-GPT (N-GPT), to configure the memory views for authorized and unauthorized entities, respectively, as shown in Figure 7. The P-GPT of an authorized entity (i.e., a Realm VM or a device assigned to a Realm VM) sets its PORTAL region as the Normal world PAS. For a Realm VM on a CPU core, its P-GPT sets the PORTAL region as Normal world PAS and maintains the rest of the PAS layout of memory regions as in its original GPT, preventing untrusted software from accessing the Realm and Root regions (Figure 8). Isolation among distrusting Realm VMs is enforced by the RMM's stage-2 translation. For a device attached to the Realm VM, the P-GPT in the SMMU configures non-PORTAL regions as Root world PAS, restricting its access to only the PORTAL region for DMA. Isolation among distrusting devices accessing any PORTAL region is enforced by SMMU stage-2 translation, ensuring each device accesses only its own PORTAL region. In contrast, the active GPT of an unauthorized entity (i.e., a Normal world VM or a device not assigned to a Realm VM) is N-GPT, which configures the PORTAL regions as Root world PAS to block unauthorized access while preserving the original access control of other memory regions (Figure 8). Consequently, memory transactions from unauthorized entities to the PORTAL regions are bound by lower exception levels and rejected by GPC before reaching the memory controller. Without such a dual-GPT setup of P-GPT and N-GPT, the untrusted VMM managing Normal world VMs can maliciously insert mappings from Normal world VMs to PORTAL regions in its stage-2 translations to enable unauthorized access.

Switching between P-GPT and N-GPT in CPU. Each CPU core's GPT is set as a P-GPT for the authorized Realm VM running on it. When not executing a Realm VM, the P-GPT transitions to an N-GPT to block unauthorized access to the PORTAL region. The Monitor in the Root world intercepts Realm Management Interface (RMI) calls (e.g., `RMI_REC_ENTER`) that start a Realm VM, transforming the N-GPT to a P-GPT before switching from the Normal to the Realm world, and vice versa (Figure 7).

Synchronizing P-GPT and N-GPT. Regardless of the access control on PORTAL regions, N-GPT and P-GPT must provide the same access control between different PAS as per CCA policies. Except for PORTAL regions, P-GPT and N-GPT should be identical for other memory sectors. In PORTAL, updates on the GPT requested via RMI and Realm Service Interface (RSI) interfaces into the Monitor thus are applied to both P-GPT and N-GPT for any Realm page not in a PORTAL region. Such a synchronisation does not incur significant performance overhead. Originally without PORTAL, one GPT (i.e., N-GPT in PORTAL terms) is updated by the Monitor. P-GPT, introduced by PORTAL, is updated based on the same mechanism except for the PORTAL region. As invoking the Monitor already incurs an expensive context-switching overhead and as N-GPT and P-GPT only differ in the PORTAL region, these additional updates add minimal overhead.

Isolating MMIO in Realm VMs. Given that MMIO controls integrated devices (e.g., GPU and SMMU) in SoC, granting exclusive MMIO access to authorized Realm VMs effectively protects each device. Since MMIO uses physical memory addresses to map peripheral registers, PORTAL employs GPC

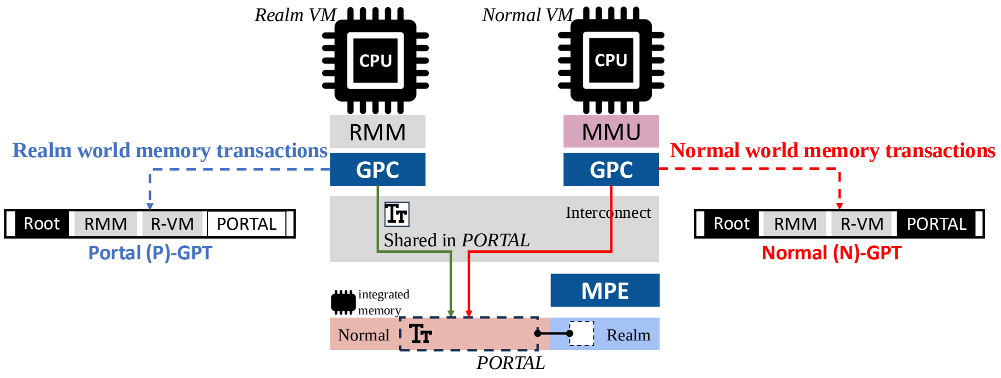

Figure 8: Isolating authorized and unauthorized entities via P-GPT and N-GPT.

API	Description
`RMI_REC_ENTER*`	initiate the execution of a Realm VM. PORTAL updates P-GPT and N-GPT. PORTAL check the device attachment / detachment.
`RSI_ATTACH_DEV`	allocate a device from a Realm VM.
`RSI_DEV_MNG`	attach and detach a device from Realm world.
`RSI_SET_PORTAL`	request a PORTAL region for DMA.
`RMI_ATTACH_DEV`	allocate device memory from Normal world to Realm world.
`RMI_CREATE_Q`	create a command queue for a newly instantiated Realm VM.
`SMC_SET_PORTAL`	delegate Realm memory to a PORTAL region. add stage-2 translation for the SMMU.

TABLE 2: Updated and new interfaces introduced by PORTAL. `RMI_REC_ENTER` is an existing interface and is updated for PORTAL-specific operations.

protection for access control on MMIO pages and host them in the Realm VM's PORTAL region. However, Arm CCA does not validate the mapping between the guest Physical Address (PA) and the host PA by default when adding new memory pages to the Realm VM. Each platform has unique and fixed MMIO physical addresses for devices, listed in the device tree and immutable at runtime. PORTAL can verify that the MMIO page mappings match these fixed addresses. Without verification, attackers can map the guest PA to a malicious host PA, enabling man-in-the-middle attacks.

4.4. System Realm Internals

We adhere to the Arm CCA specification for RMM communication with the untrusted host OS and VMM [16] and propose new interfaces (Table 2). The interfaces `RSI_ATTACH_DEV` and `RMI_ATTACH_DEV` allocate devices for Realm VMs, while `SMC_SET_PORTAL` configures the device under PORTAL protection.

System Realm initialization. We illustrate the initialization of the System Realm for exclusive SMMU management in Figure 9. Assigning a device (i.e., SMMU and peripheral) to a Realm VM is achieved by providing it exclusive access to the MMIO region of the device. Since the host VMM manages all memory resources, including SMMU MMIO addresses, the System Realm invokes `RSI_ATTACH_DEV` on the SMMU address to transfer the SMMU device from the host VMM to the System Realm (❶). As PORTAL allows only

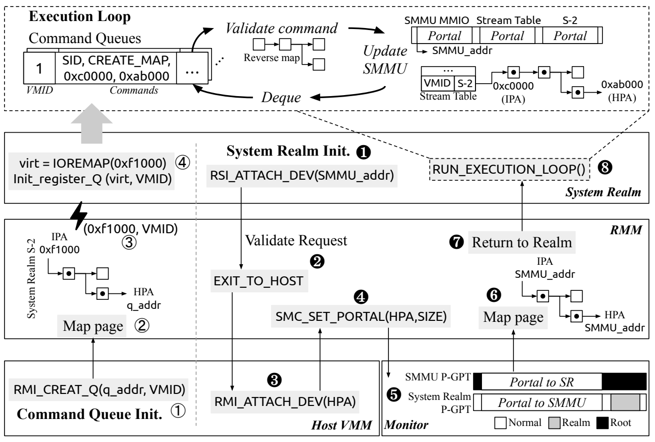

Figure 9: System Realm lifecycle.

the System Realm to acquire the SMMU, the RMM validates the `RSI_ATTACH_DEV` call by checking the requesting Realm VM's measurement against the System Realm (❷). If the System Realm initiates the SMMU device attachment request, the host delegates the PA pages mapped to the SMMU to the RMM. That is, the host detaches the SMMU driver and delegates the memory pages mapped to the SMMU MMIO region to the System Realm via `RMI_ATTACH_DEV` (❸). Since the host is not trusted, the delegated host PA pages are checked against the requested addresses. If they match, the RMM invokes `SMC_SET_PORTAL` (❹) to configure the pages within a PORTAL region for the System Realm (❺) through the Monitor, which updates the P-GPT and N-GPT accordingly. After the RMM creates a mapping in the stage-2 page table of the System Realm (❻), it returns to the System Realm (❼), which then initiates an execution loop (❽) to exclusively process the commands submitted by Realm VMs to manage the SMMU.

Command queue for efficient command submission. To reduce inter-realm context switching overhead for SMMU management, PORTAL uses ring buffers between the System Realm and other Realm VMs as command queues for efficient command submission, as shown in Figure 9. When a Realm VM is instantiated, users can set a flag to indicate PORTAL support. Based on this flag, the host VMM invokes `RMI_CREATE_Q` to create a command queue for the Realm VM. The host then delegates the command queue memory pages to the RMM (①). The RMM maps the command queue in the stage-2 page table, ensuring exclusive access for the RMM and System Realm (②). An exception is raised to the System Realm (③) to notify it of the new command queue. A pre-registered exception handler establishes the virtual-to-physical address mapping for the command queue, allowing access by the System Realm during execution. After registering the command queue, the System Realm returns from the exception and continues the execution loop (④).

Execution loop for command handling. After initializing the System Realm and command queues, the System Realm dequeues pending requests to process SMMU configuration requests for the requesting Realm VMs (Figure 9). Under benign scenarios, to establish a PORTAL region between a Realm VM and a device, the Realm VM requests the System Realm to map its PORTAL region into the SMMU page table. To prevent unauthorized mappings, the System Realm verifies 1) the host PA of the PORTAL region and 2) the owner of the PORTAL region (e.g., VMID of the Realm VM). Each command queue is a dedicated channel between the System Realm and a Realm VM, making it easy to identify the source of requests (i.e., the VMID of the requesting Realm VM). Requests are submitted by a Realm VM through an RMI call into the RMM, and the RMM holds the VMID information of the requesting Realm VM. By comparing the VMID of the requesting Realm VM with the VMID that owns the target PORTAL region, PORTAL verifies if the requesting Realm VM has the appropriate permissions to configure a specific PORTAL region for the peripheral device in the SMMU (⑤). To aid in verification, the System Realm maintains a *reverse mapping* to verify that the requested physical addresses have not been already mapped in the other I/O page table to establish the PORTAL region. After verification passes, the System Realm updates the SMMU metadata, such as the Stream Table, to generate mapping to allow authorized devices to access the PORTAL region (⑥).

On-demand command handling. Keeping the System Realm running constantly wastes CPU resources when idle. Instead, it is activated on demand for SMMU-related tasks. However, frequent context switching between Realm VMs and the System Realm can still occur. For example, devices like GPUs need memory pages for metadata, such as command and interrupt queues, which are only used after device initialization. Thus, invoking the System Realm for each allocation request of these pages is inefficient. In addition, since the host VMM [19] manages inter-realm context switching, additional overhead between the Realm and Normal world is inevitable. To reduce overhead, PORTAL *lazy-loads* the System Realm when a device encounters a page fault from invalid DMA mappings in the SMMU. PORTAL defers SMMU management until the device accesses the PORTAL region for DMA, avoiding excessive context switching during initialization.

4.5. Direct Memory Access (DMA)

Devices and the host use DMA to transfer large amounts of data. After a device is attached to a Realm VM, memory should be allocated for DMA and protected by PORTAL for secure transmission. Unlike MMIO, which has a fixed address in the host PA and is vendor-configured, the DMA region is dynamically generated and configured at runtime. Thus, a PORTAL region for DMA cannot be pre-allocated in the Realm VM like MMIO but must be requested and configured during runtime.

To allocate a PORTAL region for DMA, the requesting Realm VM submits a request to the System Realm by calling `RSI_SET_PORTAL` to change the granule of the desired physical pages from Realm PAS to Normal world PAS. The Realm VM sends a list of guest PA with size along with the device id to the RMM. The RMM fulfills this request by translating the guest PA to host PA using its stage-2 translation tables for the Realm VM. Note that the Realm memory transitioned to the Normal world PAS as the PORTAL region already belongs exclusively to the requesting Realm VM, ensuring intra-realm isolation. After the granule changes, only the requesting Realm VM can access the PORTAL region as plaintext memory, while other Realm VMs and untrusted peripherals are prohibited by the N-GPT. Besides CPU configuration, the SMMU page table must be updated for peripherals to access the PORTAL region via DMA. To transition device memory to the initialized PORTAL region, the Realm VM requests the System Realm to update the SMMU 's stage-2 tables (§4.3). With P-GPT configured, the device can access the PORTAL region via DMA after proper SMMU stage-2 mappings. This approach only requires kernel modification for DMA setup, without changes to the application, device driver, runtimes, or device logic.

4.6. Device Management

Unlike the traditional VM model in which a device is dedicated to a Realm VM throughout its lifetime, under the SoC model, multiple Realm VMs need to share the device. However, current CCA design, utilized by existing studies [69], [76], incorporates the device in the initial attestation of the Realm VM, which forces the device to be bond to the Realm VM for its entire lifespan. To support runtime device management, PORTAL introduces per-device states managed by the RMM.

Supporting runtime device management securely is non-trivial. Before deeming the ownership transfer of a device from one Realm VM to another is completed and the device is ready for use, not only the stage-2 page table mapping on the CPU side, but also the I/O page table of the SMMU should be completely transferred to the destination Realm VM. For example, assume that Realm_B wants to attach a device occupied by Realm_A. If the device is deemed ready for use after updating the SMMU I/O page table, but the stage-2 page table mapping on the CPU side remains unchanged, then Realm_A can

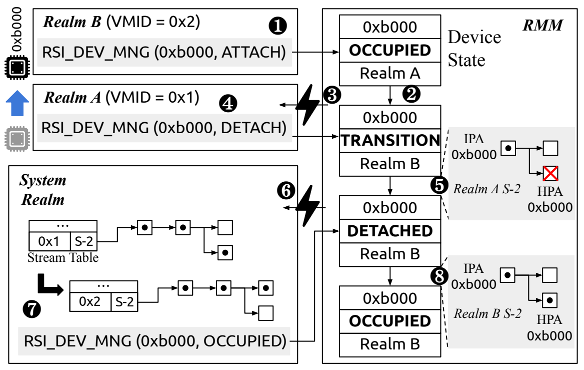

Figure 10: Device management of PORTAL.

still interact with the device, which will now have access to memory pages of Realm$_B$. This breaks the isolation between Realm VMs and is detrimental when Realm$_A$ is malicious.

We illustrate how PORTAL achieves runtime device management securely in Figure 10. The device attachment request is initiated from Realm$_B$ through `RSI_DEV_MNG` (❶). It requires the base address of the device's MMIO region and a command to notify RMM that it wants to attach a new device. Since the device has been owned by Realm$_A$, RMM first updates the state of the device from `OCCUPIED` to `TRANSITION`. The RMM also updates the owner to indicate that the device will be transferred to Realm$_B$ (❷). The RMM then enters Realm$_A$ by injecting an interrupt so that Realm$_A$ can handle device detachment operation as it is the current owner of the device (❸). Realm$_A$ handles device detachment and invokes `RSI_DEV_MNG` to notify the RMM that it is safe to detach the device from Realm$_A$ (❹). The RMM destroys the mapping to the device MMIO pages in the stage-2 page table of Realm$_A$ to prevent its further access to the device. Meanwhile, the RMM updates the device state to `DETACHED` (❺). In order to request the System Realm to update device ownership in the Stream Table, the RMM injects an interrupt to the System Realm (❻). The System Realm switches the I/O page table pointer stored in the Stream Table and updates VMID to initiate device support for Realm$_B$. Also, it invokes `RSI_DEV_MNG` to notify the RMM of the completion of the SMMU update (❼). Finally, the RMM updates the device state to `OCCUPIED` and updates the stage-2 page table of Realm$_B$ so that the Realm$_B$ accesses the device exclusively (❽).

Device Attestation. If the device remains attached until the Realm VM is destroyed, it can be attested during Realm VM initialization. However, the initial measurement value will not account for runtime device attachment and detachment events. Therefore, PORTAL makes use of the Realm Extensible Measurement (REM) register in Arm CCA to track device management histories maintained by the RMM. When the status of the device managed by PORTAL changes, PORTAL logs the changes and updates the REM register to measure the logs. The device status can only be updated through the RMI/RSI calls, so the attackers cannot manipulate the log directly. Furthermore, even though attackers can invoke RMI/RSI calls to detach devices from the Realm VM or prevent their assignment, the entire device management log is securely maintained with its measured hash. This allows the victim to analyze and triage the attacks afterward. The only feasible attack against device management is a denial-of-service attack, which is considered out of scope.

5. Implementation

We implement PORTAL in two types of prototypes: 1) a functionality prototype on Arm FVPs that verifies the design and security of PORTAL, and 2) a performance prototype that uses Armv8 instructions to emulate the latency of Arm CCA features in the coming Armv9.

5.1. Functionality Prototype

PORTAL is prototyped on Arm FVP_Base_RevC-2XAEMvA [15] with RME support. The FVPs simulates the Arm system, including processors, memory, and peripherals. Nevertheless, the FVPs does not support genuine unified-memory devices found on commercial SoC Arm chips. Instead, the FVPs includes a connected test engine that handles peripheral memory accesses as if it were a DMA-capable peripheral, along with an SMMU that supports RME. We use the simulated devices to verify the isolation guarantee of PORTAL. We use Linux v5.1 kernel as the host OS and the TF-A v2.10 as the Monitor. To verify the isolation guarantees of PORTAL, we initialize GPTs in the FVPs prototype and test with the emulated CPU and the test engine peripheral. As the existing TF-A assigns the same GPT for both components, we additionally reserve a 0.5MB memory for device GPTs. To use the GPTs to control the data transmission between the CPU and the peripheral, we configure the system registers of MMU and SMMU. We also provide a reference implementation of the System Realm, which in reality should be mediated and distributed by trusted parties such as Arm and processor vendors.

5.2. Performance Prototype

Given that the FVPs does not provide cycle-accurate emulation [15], [82], we developed a physical prototype based on Armv8 with Armv9 CCA features to evaluate PORTAL's performance. We migrated our FVPs prototype to the Orange Pi 5 Plus [53], equipped with an RK3588 SoC [63] featuring an 8-core 64-bit Arm processor (4-core A76 and 4-core A55). The SoC includes an Arm Mali-G610 GPU and 8GB of DRAM shared between the processor and the GPU. We run a customized TF-A to emulate Arm CCA. We run Ubuntu 24.04 with Linux v6.1 kernel as the Normal world host. User Realm VMs and the System Realm run a customized Linux v6.2 kernel. We disabled the A55 cores and used the A76 cores to measure overhead reliably. Note that, although Armv9 boards are recently available [42], [58], Arm CCA hardware extensions are still pending. Following state-of-the-art projects [76], [69], we emulate CCA instructions using Armv8 instructions for cycle-accurate performance evaluation. Since no non-CCA Armv9 instructions were needed, emulating on Armv9 or Armv8 makes no difference.

Emulation of Arm CCA. We replace all CCA instructions and registers with Armv8 features to simulate CCA. First, TF-A (i.e., the trusted Monitor) retrieves the GPT hardware configuration from `GPCCR_EL3`. We replace it with instructions returning a fixed value to ensure GPT initialization without exceptions. In addition, the ARMv8 processor differentiates only between Secure and Normal worlds, lacking Realm world support. We created a Realm context in the Normal world and patched the TF-A to validate the security state origin (e.g., Normal or Realm) of SMC calls for proper RSI handling from the Realm VM. Lastly, we moved the GPT initialization code (i.e., TF-A `bl2`) to `bl3` since the boot sequence of the evaluation board (i.e., the Orange Pi 5 Plus) is designed to run `u-boot` secondary program loader as its second-stage bootloader instead of TF-A `bl2`.

6. Evaluation

In this section, we evaluate our PORTAL prototypes according to the following research questions:

- **RQ1:** How large is the TCB of PORTAL?
- **RQ2:** Can PORTAL defend against privileged adversaries?
- **RQ3:** How much performance and power benefit does PORTAL provide?
- **RQ4:** How much performance overhead does PORTAL incur?
- **RQ5:** How much memory overhead does PORTAL incur?

We measure the performance benefit of our prototype using the Rodinia benchmark suites [22], covering various Arm GPU use cases. We also evaluate the performance overhead of PORTAL due to its lifecycle, the System Realm, and device management. Each experiment is conducted 10 times, and we calculate the average values as the final results.

6.1. RQ1: TCB Size

Layer	Lines of Code (LoC)
TF-A	96
RMM	784
System Realm	550
Realm VM (Guest kernel)	230
All	1,660

TABLE 3: Introduced TCB size of PORTAL measured in LoC.

We measure the TCB size of PORTAL using cloc [2] in terms of standard lines of source code. PORTAL introduces 1,660 LoC additions in total.

6.2. RQ2: Security Analysis

The removal of memory encryption by PORTAL necessitates a meticulous security analysis, compared to existing approaches. We analyze the security of PORTAL on a wide range of attacks based on our threat model §3.3. We also combine and include attack scenarios from state-of-the-art solutions [76], [69] to ensure the security compliance of PORTAL aligns with them.

Unauthorized memory access and modification. To compromise data security, an attacker may directly leak or manipulate the data transmitted between the Realm VM and the peripheral. PORTAL utilizes Arm CCA's GPC to protect the shared PORTAL region from unauthorized access by entities such as the host OS, VMM, and untrusted peripherals. In particular, the PORTAL region is visible as Normal world PAS only to authorized Realm VMs and peripherals, whereas it is seen as Root world PAS by all other entities. This configuration is set in GPTs and is enforced by GPC. An attacker might also directly request a Realm VM under her control to directly access the sensitive data in other Realm VMs. However, such a request fails to bypass the memory isolation on the CPU side due to stage-2 translation in the RMM. Same attacks using malicious applications on the device side fails for the same reason due to the SMMU GPC.

Illegal device management. Given that device memory is managed by the untrusted hypervisor, a malicious hypervisor could attempt to expose sensitive information by dishonestly handling memory requests (e.g., Iago attacks [23]). To defend against these attacks, the System Realm ensures that the device memory does not overlap with the memory allocated to other realms and devices. Specifically, the System Realm verifies whether the PORTAL region used for the device communication overlaps with the Normal world PAS of other entities in the GPTs. The attacker might also attempt to deliberately map the device memory to an unauthorized area or create duplicate mappings. In such scenarios, the System Realm checks the mappings before making changes to the actual device page table (i.e., the SMMU stage-2 page table).

Fake device and SMMU. An attacker (e.g., VMM) could emulate a fake peripheral device and attempt to transfer sensitive data into this simulated device. In addition, the attacker could also simulate the SMMU to compromise the GPT isolation guarantees. PORTAL guarantees that the CCA Monitor communicates with actual hardware devices rather than simulated ones. PORTAL takes advantage of the fact that the physical addresses of Arm peripheral devices and SMMU MMIO registers are fixed and unmodifiable in most Arm-based devices [6], [17], [28], [70], [76]. Consequently, malicious or emulated devices must be assigned to different fixed addresses, enabling PORTAL to confirm that the data transfer is occurring with the genuine device. Furthermore, an attacker could try using peripheral multiplexing to assign the same memory address to another fake peripheral. However, the initial configuration of device multiplexing is performed by the firmware of SoC during the initial boot of the system, which is within Arm CCA 's TCB and thus prevents such attacks.

Malicious co-tenants. An attacker can deploy a malicious Realm VM that is co-located with the victim Realm VM on the same platform. The malicious Realm VM may attempt to allocate DMA regions that overlap with the victim's devices to gain access to sensitive data. In PORTAL, the System Realm ensures that the malicious Realm VM cannot request DMA mappings on PA that it does not own by verifying the stage-2 translation table, thereby preventing overlapped DMA regions.

CPU GPC bypass. An attacker might attempt to circumvent the CPU GPC to gain access to restricted memory. One possible method is to disable the GPC or replace the GPT with a malicious version. However, since the GPC registers are located in the Root world, the attacker lacks the necessary privileges to access them. The attacker might also try to directly alter the GPT to compromise memory isolation, but this is also thwarted as the GPTs resides in the Root world. Finally, the attacker might exploit GPC TLB entries to access the protected region. That is, because the TLB is cached following the GPC, when the GPT is updated, the TLB will continue to correspond to the GPC based on the old GPT version. PORTAL mitigates such attacks by ensuring that TLB entries are flushed whenever the CPU GPT is modified.

Device GPC bypass. Just as with CPU GPC, attackers may attempt to circumvent the GPC on the device side to gain unauthorized access to the device memory, other realms, and the TCB of PORTAL. Nevertheless, the attacker is unable to directly access the registers that control GPC because she does not possess Root world privileges. Furthermore, PORTAL protects the device GPTs within the Root world, preventing the attacker from directly altering them to undermine access control. Similarly, the attacker could exploit the TLB to circumvent GPC. Thus, PORTAL ensures that TLB entries in the SMMU are flushed whenever there are changes to the device GPT.

Malicious DMA. An attacker could exploit other devices to gain unauthorized access to the PORTAL region for DMA operations between the Realm VM and the connected device. Although a PORTAL region is a plaintext shared memory area, PORTAL depends on the GPT's stage-2 translation table to block untrusted devices from accessing the PORTAL region designated for DMA. Specifically, the PORTAL region currently in use is configured as Root world PAS in the GPTs for untrusted entities. As the attacker might instead launch a malicious application on the victim's device to achieve the same goal, PORTAL ensures that when a device is assigned to other realms, a PORTAL region that is not overlapped with other PORTAL regions is assigned for DMA.

Physical Attacker. A physical attacker can take advantage of a compromised device (e.g., with debugging features enabled or loaded with vulnerable firmware). When such a device is connected to a Realm VM, it can compromise the platform's security. PORTAL depends on the remote verifier to confirm that the device connected to the Realm VM is correctly configured during Arm CCA remote attestation. Moreover, a physical attacker might attempt to directly probe the memory related to PORTAL regions to extract sensitive information. However, this type of attack is not viable under PORTAL's threat model, which relies on the robustness of SoC-based Arm processor packages. Additionally, we assume that probing or maliciously redirecting the memory transactions of PCIe peripherals is prevented by the security features of Integrity and Data encryption (IDE) included in PCIe-5 [73].

Application	Problem Size	Data Buffers	Memory
`Gaussian`	1024 × 1024 nodes	3	8.39 MB
`LUD`	2048 × 2048 nodes	1	16.00 MB
`Pathfinder`	100000 × 100 points	4	40.46 MB
`NW`	2048 × 10 nodes	2	16.79 MB
`Hotspot3D`	512 × 512 × 8 nodes	3	25.16 MB
`NN`	42764 nodes	2	0.51MB

TABLE 4: Configuration of the selected Rodinia benchmark

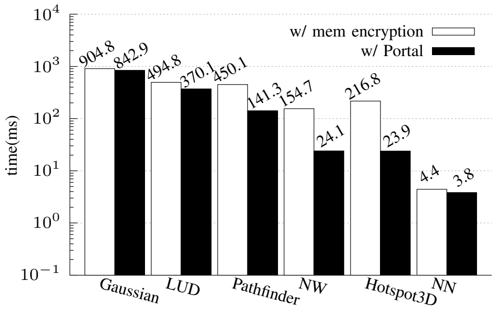

Figure 11: Performance of GPU tasks when protected by memory encryption and by PORTAL.

6.3. RQ3-1: Performance Benefit of PORTAL

We follow best practices from state-of-the-art solutions [76], [69] and evaluated the performance of well selected six GPU tasks from the Rodinia benchmark [22] both under the traditional confidential model with memory encryption and PORTAL with plaintext isolated memory for DMA. The selected benchmarks cover a wide range of domains, data workloads, and sizes of applications, including a heavy-weight `Gaussian`, two medium-weight applications (`LUD` and `Pathfinder`), and three light-weight applications (`NW`, `Hotspot3D` and `NN`). We adopt AES-GCM encryption on both CPU and GPU to establish a baseline where Realm VM and the device communicate via encrypted shared memory for DMA. We report the problem size and memory consumption in Table 4.

As shown in Figure 11, based on these configurations, the performance overhead due to memory encryption is heavier when the processing logic is simpler but the data size is larger, which conforms to the characteristic of memory encryption. It also shows that PORTAL benefit the most when protecting data intensive applications where larger size of data is transmitted frequently. Specifically, `Hotspot3D` deals with a large 3D grid of temperatures, making it data intensive, and achieves the highest performance gain of 9.07× among the six selected benchmarks. `NW` (Needleman-Wunsch) and `Pathfinder` involve frequent data movement due to their dynamic programming nature (i.e., requiring frequent updates to tables or grids), and therefore also achieve great performance increase of 6.42× and 3.19×, respectively. `LUD` (1.34×), `NN` (1.16×), and `Gaussian` (1.07×), which showed relatively lower performance increases, involve more complex control flow and are more compute-bound than others, meaning

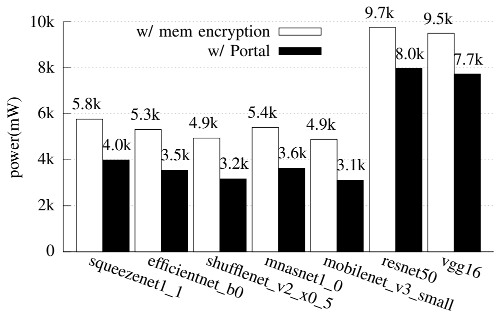

Figure 12: Power consumption of machine learning workloads when protected by memory encryption and by PORTAL.

the bottleneck is in the CPU/GPU processing rather than memory bandwidth or data movement. In compute-bound benchmarks, optimizations to memory access or data transmission become less impactful compared to those focusing on vectorization or parallelism. Overall, PORTAL achieves an average performance improvement of 3.71× (1.07×-9.07×) over the six selected Rodinia benchmark GPU tasks thanks to its plaintext isolated memory for device communication.

Note that, the hardware-based encryption adopted in the upcoming RME should have a much better performance compared to AES-GCM. However, eliminating the memory encryption by RME should still grant a noticeable performance benefit for PORTAL. Furthermore, comparing the relative performance with other works is challenging. Existing solutions [76], [69], [25] are implemented on different platforms (e.g., Hikey960, Juno R2, custom FPGA, etc.), as well as different trusted firmware, RMM, and kernel images, incurring costly re-implementation efforts. Nonetheless, PORTAL is the first to propose the removal of memory encryption in device access for Arm CCA on mobile SoCs, which brings considerable performance and efficiency benefits while maintaining security, while existing works incur different levels of performance overhead instead.

6.4. RQ3-2: Power Benefit of PORTAL

We also evaluated the energy consumption of seven real-world lightweight machine learning models when protected with memory encryption and with PORTAL. Specifically, we ran five smaller models (i.e., `squeezenet1_1`, `efficientnet_b0`, `shufflenet_v2_x0_5`, `mnasnet1_0`, and `mobilenet_v3_small`) and two larger models (i.e., `resnet50`, `vgg16`) to simulate the device's power consumption under various scenarios. We gathered power consumption data at 50 ms intervals over a five-minute period and calculated the average power consumption. As shown in Figure 12, PORTAL provides an average power consumption reduction of 1.43×.

6.5. RQ4-1: Performance of PORTAL Lifecycle

Initializing GPTs. PORTAL uses two GPTs to protect regions (i.e., P-GPT and N-GPT), initialized at boot time. The existing system-wide GPT is used as N-GPT, and P-GPT is additionally populated. Initializing the additional GPTs incurs a one-time overhead.

Initializing Realm VM. PORTAL incurs minimal overhead during Realm VM creation if the PORTAL flag is not set. When PORTAL is activated, additional overhead occurs due to invoking `RMI_CREATE_Q` to establish the command queue page. Note that `RMI_CREATE_Q` can be invoked at any point during Realm VM creation, as injecting exceptions and handling the command queue can occur concurrently. Realm VM creation time varies with its size, but no noticeable overhead is detected if `RMI_CREATE_Q` is invoked early. Invoking `RMI_CREATE_Q` at the end of creation introduces a 2% overhead.

Entering Realm VM. PORTAL adds a conditional check in `RMI_REC_ENTER` to address device management before entering the Realm VM. The runtime overhead, regardless of PORTAL usage, is a single-bit flag check in the Realm Descriptor, equivalent to 5 instructions. If there are pending device management operations for Realm VM, the RMM injects a fault to notify Realm VM. The injection process involves reading system registers, determining the handler address, and updating the system register to resume execution from the exception handler. This injection overhead is 6.3% compared to executing `RMI_REC_ENTER` with the device management flag unset.

6.6. RQ4-2: Performance Overhead of System Realm

Initializing System Realm. The System Realm is instantiated during host OS boot and persists until system shutdown. PORTAL incurs a one-time 1.5% overhead for the System Realm during boot, which involves Realm VM creation, SMMU device attachment, and data structure setup (e.g., Stream Tables). Most of the overhead arises from Realm VM creation, not from the additional RMI/RSI calls for SMMU attachment.

Processing SMMU management commands. PORTAL incurs two types of overhead in processing commands. The first is the cost of engaging the system realm when an SMMU interrupt occurs due to an invalid mapping in the PORTAL region. This involves handling the SMMU interrupt in the VMM and forwarding the virtual interrupt to the System Realm. Since CCA supports interrupt injection through existing `RMI_REC_ENTER`, PORTAL does not require additional operations to engage the System Realm. It takes 2 µs, with most of the overhead from world switching incurred by entering the System Realm. This can be minimized by allocating a dedicated physical core to handle commands in a polling style. The other overhead involves processing commands, which includes traversing queues, checking reverse mappings, and creating SMMU I/O page table mappings. In our evaluation, two Realm VM instances using PORTAL generated 10 commands per queue, requesting to insert SMMU entries. Processing 20 commands took 1.9 µs to 2.4 µs. We dedicated one core for the execution loop in the

System Realm. This overhead can be optimized by assigning multiple cores as the number of command queues increases.

6.7. RQ4-3: Performance Overhead of Device Management

Device initial attachment. The overhead of device attachment varies based on the device's status requested by the Realm VM. If the device is not assigned to a Realm VM, its MMIO pages must be first delegated to the RMM and configured as a PORTAL region via `SMC_SET_PORTAL`. Our evaluation shows that attaching the MMIO pages of the Mali-G610 GPU (512 pages) takes $46.29\,\mu s$.

Device reassignment. If the device is occupied by another Realm VM, delegating MMIO pages is unnecessary. Instead, there are extra communication costs between the current Realm VM and the System Realm, as detailed in Figure 10, incurring $175\,\mu s$ overhead. Note that most of the overhead comes from multiple world switches due to inter-Realm VMs communication. While device attachment is costly, it is not frequent. In addition, in contrast to the current RMM design [19], which involves the VMM for each MMIO page access, PORTAL assigns MMIO pages directly to the Realm VM's PORTAL region, eliminating additional host involvement.

6.8. RQ5: Memory Overhead

Maintaining P-GPT. In our evaluation, the protected memory region is configured as 4GB. The GPT uses two levels of page tables: 1GB for the first-level and 4KB for the last-level. Adding one GPT as P-GPT requires a 4KB page in SRAM (first-level GPT) and 0.5MB of DRAM (second-level GPT). Compared to existing approaches [76], PORTAL does not need additional GPT per device because the SMMU provides extra protection using the P-GPT configured for SMMU, enabling PORTAL to be used in constrained devices with limited memory.

Device management & Command queue. The RMM must maintain per-device data structures for secure device management. Considering the device status and the owner, the next transition state and the authorized Realm VM to initiate the transition can be determined. Since PORTAL does not trust the VMM for device management, per-device status is stored within the RMM. PORTAL allocates 8 bytes for each device: 40 bits for Realm Descriptor address (page-aligned), 16 bits for VMID, and 2 bits for device state. PORTAL also allocates a 256KB command queue per Realm VM, including VMID, locks, and the queue. With a 16-byte entry size, up to 16,384 requests can be queued per Realm VM. The command queue size is based on the experiment in §6.6.

7. Related Work

Secure Device Access in Arm CCA. Strongbox [25] enhances TrustZone-based isolation for integrated GPUs on Arm platforms, excluding the driver from the TCB. However, StrongBox trusts the Secure World, which PORTAL excludes from its TCB to align with Arm CCA's threat model. StrongBox is also incompatible with hypervisors, which, if compromised, can bypass the stage-2 translation, PORTAL's primary protection mechanism. Compared to StrongBox, PORTAL is more suitable for next-generation Arm devices. Recently, Arm CCA has introduced a proposal to support peripheral devices. However, this support, known as RME Device Assignment (RME-DA) [16], remains a theoretical concept without finalized hardware implementation. To address the issue of data security in external peripherals (e.g., PCIe), a recent study called ACAI [69] suggests a similar design to RME-DA by extending CCA security invariants to device-side access. Compared to these designs, PORTAL has a more compact TCB. Both RME-DA and ACAI add additional management logic to the privileged RMM (EL2) and Monitor (EL3). PORTAL utilizes a dedicated System Realm for device management tasks, preventing the TCB of CCA core components from becoming bloated. In addition, RME-DA introduces hardware changes on the SMMU and still employs memory encryption, which is undesirable for mobile Arm SoCs, while ACAI replaces the SMMU configuration code with special interfaces. Instead, PORTAL does not introduce any modification to the SMMU by relying on the Normal World PAS for isolated memory, and the System Realm for exclusive SMMU management. CAGE [76] seeks to enhance unified-memory GPU support on Arm CCA. CAGE introduces a novel shadow task mechanism to flexibly manage confidential GPU applications and utilizes multiple GPCs to achieve two-way isolation. There are several distinct differences between CAGE and PORTAL. CAGE aims to efficiently execute confidential GPU tasks on Arm CCA, while PORTAL is designed to provide fast and secure I/O for a wide range of Arm peripheral devices. CAGE retains memory encryption, which presents disadvantages in mobile SoCs as discussed. PORTAL, leveraging the integrated memory model of mobile SoCs, safely removes memory encryption, benefiting from the SoC's physical robustness. For SMMU management, CAGE relies on the untrusted host, whereas PORTAL introduces a System Realm for exclusive SMMU management, enhancing security. In addition, CAGE requires a GPT for each Realm VM due to its absence of the RMM, which PORTAL overcomes by utilizing the trusted RMM, needing only one GPT for all Realm VMs, with intra-Realm isolation managed by the RMM. While CAGE assumes one SMMU per device, common Arm mobile SoCs have an integrated SMMU [76], so PORTAL only requires one GPT for the SMMU, with intra-device isolation managed by the stage-2 table in the Stream Table.

The most distinct difference of PORTAL is its usage of plaintext memory for secure device access in Arm CCA. Such a design respects the modern trend of the increasing number of integrated devices on mobile Arm SoCs while managing the unique challenges posed by those platforms. Besides efficient secure device I/O, PORTAL enables dynamic device management for Arm CCA thanks to the non-binding plaintext shared memory, unlike existing solutions that require device assignment to a Realm VM for its entire lifetime. We believe that PORTAL can effectively foster a wider adoption of Arm CCA to mobile platforms upon its coming release.

GPU TEEs. GPU TEEs have been proposed recently to secure the data security of GPUs. In order to establish a secure I/O between the user and the GPU, a typical GPU TEE relies on the CPU-side TEE to transfer the sensitive data to the GPU and manage the access control to the GPU MMIO. For example, HIX [36] and HoneyComb [41] use Intel Software Guard Extensions (SGX) and AMD Secure Encrypted Virtualization (SEV) to secure the GPU MMIO, respectively. Cure [21] secures the GPU TEE by adding an access control filer in the system bus. HETEE [83] protects the GPU resources within isolation environments by introducing a security controller on the FPGA hardware. Due to the difference in architecture and chip design, these approaches might not be suitable to adopt these approaches to protect Arm GPUs. Recent works have also proposed Arm-based TEEs for GPUs [25], [37], [54], [78]. These approaches utilize traditional Arm security primitives, such as the Arm TrustZone, while they provide insufficient security guarantees under the security model of Arm CCA. Lastly, several efforts have been made to directly house TEEs inside GPUs without the CPU-side TEEs, including Graviton [75] and Nvidia H100 GPUs [48]. The standalone GPU TEEs monitor the commands and data submitted by the host and ensures confidentiality during task execution. However, these approaches rely on the fact that the GPU memory and the host are naturally isolated as external GPUs possess private memory, while such assumptions do not hold for integrated GPUs on Arm processors with unified memory model.

8. Discussion

Limitations. PORTAL relies on the assumption that the SoC package is physically secure, making physical memory attacks infeasible. While this is valid for mobile SoCs, one of the largest Arm markets, extending PORTAL to other platforms where this assumption does not hold presents a challenge. Potential solutions could involve integrating additional physical security measures or incorporating selective memory encryption where necessary. PORTAL also requires vendors to implement and distribute a System Realm tailored to their specific SoC configurations. This increases the burden on vendors, who must ensure the System Realm's correctness and security through meticulous validation and testing. Collaborating with Arm and industry stakeholders to develop standardized implementations of the System Realm could help mitigate this burden.

Discrete devices. PORTAL's utility extends beyond SoC environments to include scenarios where data must be transmitted from SoC integrated memory to external peripherals connected through PCIe, such as discrete GPUs. In these cases, PORTAL can leverage the link-layer encryption provided by PCIe-5 [73], which ensures secure data transmission between the SoC and the external devices. This approach requires only a one-time encryption provided by the PCIe link layer, eliminating the need for additional software-based or hardware-based encryption on the SoC. This not only maintains the performance and power efficiency benefits of PORTAL but also enhances its applicability to a broader range of devices and configurations. By relying on PCIe link-layer encryption, PORTAL can effectively secure data in transit without the overhead associated with traditional encryption methods, making it a versatile solution for both integrated and discrete peripheral environments.

Real world applications. PORTAL offers significant benefits across a range of real-life applications, enhancing both performance and energy efficiency. In practical use cases like mobile gaming, video streaming, and AR/VR, it accelerates secure data exchanges between GPUs, CPUs, and memory, reducing latency and power consumption for smoother experiences and longer battery life. Security-critical AI-driven tasks, such as facial recognition or on-device voice assistants, benefit from faster inference and lower energy overhead, while autonomous vehicles can process sensor data in real-time without the encryption bottlenecks, improving safety and operational duration. Innovative applications like wearable health devices, smart cities, and IoT systems see improved data transmission efficiency while maintaining privacy, enabling longer device lifespans and more accurate monitoring. Additionally, in healthcare, PORTAL can help ensure the integrity and confidentiality of crucial real-time data, such as operations during robot-assisted surgeries, and in industrial automation, it can improve the reliability of control systems while improving responsiveness and scalability.

Regulatory and compliance considerations. The adoption of PORTAL in real-world applications must navigate various regulatory and compliance considerations, particularly in industries with stringent data protection requirements. While PORTAL enhances performance and power efficiency by eliminating memory encryption, some applications may still require encryption to comply with regulations such as GDPR, HIPAA, and PCI-DSS. To address this, PORTAL could integrate optional encryption mechanisms to meet these specific regulatory standards without compromising its core benefits. Additionally, the implementation of the System Realm by vendors necessitates thorough validation to ensure compliance with security certifications and standards. Engaging with regulatory bodies and industry consortia to establish PORTAL as a compliant solution across different sectors will be crucial.

9. Conclusion

PORTAL provides a novel solution for secure and efficient device I/O in Arm CCA on mobile Arm SoCs. By implementing strict memory isolation without memory encryption, PORTAL tackles performance, scalability, and power efficiency challenges hindering Arm CCA adoption in the near future. PORTAL leverages CCA's GPC and SMMU for robust hardware-level access control, enabling secure and efficient data interactions between Realm VMs and devices. It also introduces dynamic device management via a dedicated

System Realm, facilitating flexible peripheral integration at runtime. The evaluation results show that PORTAL improves data transmission and reduces energy consumption.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd for their helpful feedback. We also would like to thank Bokdeuk Jeon from Samsung for constructive discussions. This research was funded by a research grant from Samsung.

References

[1] "Apple vision pro - technical specifications," 2024, https://support.apple.com/en-us/117810.

[2] AlDanial, "cloc," 2024, https://github.com/AlDanial/cloc.

[3] A. S. D. Alluhaidan and P. Prabu, "End-to-end encryption in resource-constrained iot device," *IEEE Access*, vol. 11, pp. 70 040–70 051, 2023.

[4] AMD, "AMD Secure Encrypted Virtualization (SEV)," 2024, https://developer.amd.com/sev/.

[5] AMD, "Confidential computing solution brief," 2024, https://www.amd.com/en/processors/epyc-confidential-computing-cloud.

[6] Amlogic, Inc., "S905 Datasheet," 2016, https://dn.odroid.com/S905/DataSheet/S905_Public_Datasheet_V1.1.4.pdf.

[7] Apple, "Apple unleashes M1," 2020, https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/.

[8] ——, "Apple unveils M3, M3 Pro, and M3 Max, the most advanced chips for a personal computer," 2023, https://www.apple.com/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/.

[9] Arm, "Arm Architecture Reference Manual for A-profile architecture," 2024, https://developer.arm.com/documentation/ddi0487/latest/.

[10] ARM, "Arm Confidential Compute Architecture," 2024, https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture.

[11] Arm, "Arm Mali Graphics Processing Units (GPUs)," 2024, https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus.

[12] ——, "Arm Realm Management Extension (RME) System Architecture," 2024, https://developer.arm.com/documentation/den0129/latest/.

[13] ——, "Arm System Memory Management Unit Architecture Specification," 2024, https://developer.arm.com/documentation/ihi0070/latest/.

[14] ——, "Autonomous Vehicles," 2024, https://www.arm.com/markets/automotive/autonomous-vehicles.

[15] ——, "Fixed Virtual Platforms," 2024, https://www.arm.com/products/development-tools/simulation/fixed-virtual-platforms.

[16] ARM, "Introducing arm confidential compute architecture," 2024, https://developer.arm.com/documentation/den0125/0300/.

[17] Arm, "Juno r2 ARM Development Platform SoC," 2024, https://developer.arm.com/documentation/den0125/0300/Arm-CCA-Hardware-Architecture.

[18] ——, "Layered Security for Your Next SoC," 2024, https://www.arm.com/products/silicon-ip-security.

[19] ——, "Realm Management Monitor specification," 2024, https://developer.arm.com/documentation/den0137/latest/.

[20] ——, "SoC Development," 2024, https://www.arm.com/glossary/soc-development.

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

[22] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2009, pp. 44–54.

[23] S. Checkoway and H. Shacham, "Iago attacks: Why the system call api is a bad untrusted rpc interface," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 253–264, 2013.

[24] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. de Lara, H. Raj, S. Saroiu, and A. Wolman, "Protecting Data on Smartphones and Tablets from Memory Attacks," in *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Istanbul, Turkey, Mar. 2015.

[25] Y. Deng, C. Wang, S. Yu, S. Liu, Z. Ning, K. Leach, J. Li, S. Yan, Z. He, J. Cao, and F. Zhang, "StrongBox: A GPU TEE on Arm Endpoints," in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, Nov. 2022.

[26] C. Feng, H. Sun, J. Wang, L. Zhang, and S. Xu, "A soc architecture compatible with cortex-m4," in *2023 IEEE 7th Information Technology and Mechatronics Engineering Conference (ITOEC)*, vol. 7, 2023, pp. 514–518.

[27] Forbes, "Arm Stock: AI Chip Favorite Is Overpriced," 2024, https://www.forbes.com/sites/bethkindig/2024/03/21/arm-stock-ai-chip-favorite-is-overpriced/?sh=759485ec69d6.

[28] FuZhou Rockchip Electronics Co., Ltd., "Rockchip RK3288 Technical Reference Manual Part1," 2017, https://opensource.rock-chips.com/images/8/8f/Rockchip_RK3288_TRM_V1.2_Part1-20170321.pdf.

[29] S. Ghosh, M. N. I. Khan, A. De, and J.-W. Jang, "Security and privacy threats to on-chip non-volatile memories and countermeasures," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–6.

[30] Google, "Confidential computing," 2024, https://cloud.google.com/security/products/confidential-computing.

[31] M. Gruhn and T. Müller, "On the practicability of cold boot attacks," in *2013 International Conference on Availability, Reliability and Security*, 2013, pp. 390–397.

[32] B. Halak, T. Gibson, M. Henley, C.-B. Botea, B. Heath, and S. Khan, "Evaluation of performance, energy, and computation costs of quantum-attack resilient encryption algorithms for embedded devices," *IEEE Access*, vol. 12, pp. 8791–8805, 2024.

[33] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," in *Proceedings of the 17th USENIX Security Symposium (Security)*, San Jose, CA, Jul.–Aug. 2008.

[34] G. D. H. Hunt, R. Pai, M. V. Le, H. Jamjoom, S. Bhattiprolu, R. Boivie, L. Dufour, B. Frey, M. Kapur, K. A. Goldman, R. Grimm, J. Janakirman, J. M. Ludden, P. Mackerras, C. May, E. R. Palmer, B. B. Rao, L. Roy, W. A. Starke, J. Stuecheli, E. Valdez, and W. Voigt, "Confidential computing for OpenPOWER," in *Proceedings of the 16th European Conference on Computer Systems (EuroSys)*, Virtual, Apr. 2021.

[35] Intel, "Intel Trust Domain Extensions (TDX)," 2024, https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.

[36] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.

[37] J. Jiang, J. Qi, T. Shen, X. Chen, S. Zhao, S. Wang, L. Chen, G. Zhang, X. Luo, and H. Cui, "Cronus: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environment," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 124–143.

[38] D. Lee, D. Jung, I. T. Fang, C. che Tsai, and R. A. Popa, "An Off-Chip attack on hardware enclaves via the memory bus," in *Proceedings of the 29th USENIX Security Symposium (Security)*, Virtual, Aug. 2020.

[39] Linux, "Support for Arm CCA VMs on Linux," 2023, https://lwn.net/Articles/921482/.

[40] J. Mahmod and M. Hicks, "UnTrustZone: Systematic Accelerated Aging to Expose On-chip Secrets," in *Proceedings of the 45th IEEE Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2024.

[41] H. Mai, J. Zhao, H. Zheng, Y. Zhao, Z. Liu, M. Gao, C. Wang, H. Cui, X. Feng, and C. Kozyrakis, "Honeycomb: Secure and efficient {GPU} executions via static validation," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 155–172.

[42] MediaTek, "MediaTek Dimensity 9000," 2024, https://www.mediatek.com/products/smartphones-2/mediatek-dimensity-9000.

[43] D. Mehmedagić, M. R. Fadiheh, J. Müller, A. L. D. Antón, D. Stoffel, and W. Kunz, "Design of access control mechanisms in Systems-on-Chip with formal integrity guarantees," in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.

[44] Microsoft, "Azure confidential computing," 2024, https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[45] Mobileye, "EyeQ: The System-on-Chip for Automotive Applications," 2024, https://www.mobileye.com/technology/eyeq-chip/.

[46] P. S. Munoz, N. Tran, B. Craig, B. Dezfouli, and Y. Liu, "Analyzing the resource utilization of aes encryption on iot devices," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018, pp. 1200–1207.

[47] M. Mössinger, B. Petschkuhn, J. Bauer, R. C. Staudemeyer, M. Wójcik, and H. C. Pöhls, "Towards quantifying the cost of a secure iot: Overhead and energy consumption of ecc signatures on an arm-based device," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2016, pp. 1–6.

[48] Nvidia, "Confidential Compute on NVIDIA Hopper H100," 2024, https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf.

[49] ——, "In-Vehicle Computing for AI-Defined Cars," 2024, https://www.nvidia.com/en-us/self-driving-cars/in-vehicle-computing/.

[50] ——, "Nvidia Grace CPU," 2024, https://www.nvidia.com/en-us/data-center/grace-cpu/.

[51] ——, "Nvidia Jetson Xavier," 2024, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[52] ——, "Tegra X1," 2024, https://developer.nvidia.com/content/tegra-x1/.

[53] Orange Pi, "Orange Pi 5 Plus (4GB/8GB/16GB)," 2024, http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5-plus.html.

[54] H. Park and F. X. Lin, "Safe and practical gpu computation in trustzone," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 505–520.

[55] P. Porambage, A. Braeken, A. Gurtov, M. Ylianttila, and S. Spinsante, "Secure end-to-end communication for constrained devices in iot-enabled ambient assisted living systems," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 711–714.

[56] Qualcomm, "Adreno Graphics Processing Units," 2024, https://developer.qualcomm.com/software/adreno-gpu-sdk/gpu/.

[57] ——, "Snapdragon 8 Gen 3 Mobile Platform," 2024, https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3-mobile-platform.

[58] ——, "Snapdragon 8 Gen 3 Mobile Platform," 2024, https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-3-mobile-platform.

[59] ——, "Snapdragon Ride Platform SoCs," 2024, https://www.qualcomm.com/products/automotive/automated-driving/.

[60] ——, "Snapdragon XR2 5G Platform," 2024, https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-xr2-5g-platform.

[61] J. Raigoza and K. Jituri, "Evaluating performance of symmetric encryption algorithms," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 1378–1379.

[62] Y. Ren, J. Li, Z. Yang, P. P. C. Lee, and X. Zhang, "Accelerating Encrypted Deduplication via SGX," in *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, Renton, WA, Jul. 2019.

[63] Rockchip, "RK3588 Brief Datasheet," 2022, https://www.rock-chips.com/uploads/pdf/2022.8.26/192/RK3588%20Brief%20Datasheet.pdf.

[64] M. S. U. I. Sami, T. Zhang, A. M. Shuvo, M. S. U. Haque, P. E. Calzada, K. Z. Azar, H. M. Kamali, F. Rahman, F. Farahmandi, and M. Tehranipoor, "Advancing trustworthiness in system-in-package: A novel root-of-trust hardware security module for heterogeneous integration," *IEEE Access*, vol. 12, pp. 48 081–48 107, 2024.

[65] Samsung, "Mobile performance redefined," 2024, https://semiconductor.samsung.com/us/processor/mobile-processor/.

[66] A. Selinger, K. Rupp, and S. Selberherr, "Evaluation of mobile arm-based socs for high performance computing," in *Proceedings of the 24th High Performance Computing Symposium*, ser. HPC '16. San Diego, CA, USA: Society for Computer Simulation International, 2016.

[67] A. T. Sheikh, A. Shoker, and P. Esteves-Verissimo, "Resilient and secure system on chip with rejuvenation in the wake of persistent attacks," in *Proceedings of the 16th European Workshop on System Security*, ser. EUROSEC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 37–43.

[68] S. P. Skorobogatov, "Physical attacks and tamper resistance," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:107662043

[69] S. Sridhara, A. Bertschi, B. Schlüter, M. Kuhne, F. Aliberti, and S. Shinde, "ACAI: Protecting Accelerator Execution with Arm Confidential Computing Architecture," in *Proceedings of the 33rd USENIX Security Symposium (Security)*, Philadelphia, PA, Aug. 2024.

[70] STMicroelectronics, "GPU device tree configuration," 2023, https://wiki.st.com/stm32mpu/wiki/GPU_device_tree_configuration.

[71] Y. Su and D. C. Ranasinghe, "Leaving your things unattended is no joke! memory bus snooping and open debug interface exploits," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2022, pp. 643–648.

[72] I. Sultan and M. T. Banday, "An energy efficient encryption technique for the internet of things sensor nodes," *International Journal of Information Technology*, 02 2024.

[73] Synopsys, "Synopsys IDE Security IP Module for PCI Express 5.0," 2024, https://www.synopsys.com/dw/ipdir.php?ds=security-pcie5-ide.

[74] Tesla, "FSD Chip," 2024, https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip.

[75] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on {GPUs}," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.

[76] C. Wang, F. Zhang, Y. Deng, K. Leach, J. Cao, Z. Ning, S. Yan, and Z. He, "CAGE: Complementing Arm CCA with GPU Extensions," in *Proceedings of the 2024 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2024.

[77] H. Wang, J. Ma, Y. Yang, M. Gong, and Q. Wang, "A review of system-in-package technologies: application and reliability of advanced packaging," *Micromachines*, vol. 14, no. 6, p. 1149, 2023.

[78] J. Wang, Y. Wang, and N. Zhang, "Secure and timely gpu execution in cyber-physical systems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2591–2605.

[79] Xilinx, "Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices," 2024, https://docs.amd.com/v/u/en-US/xapp1323-zynq-usp-tamper-resistant-designs.

[80] S. F. Yitbarek, M. T. Aga, R. Das, and T. Austin, "Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors," in *Proceedings of the 23rd IEEE Symposium on High Performance Computer Architecture (HPCA)*, Austin, TX, Feb. 2017.

[81] N. Zhang, K. Sun, W. Lou, and Y. T. Hou, "CaSE: Cache-Assisted Secure Execution on ARM Processors," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2016.

[82] Y. Zhang, Y. Hu, Z. Ning, F. Zhang, X. Luo, H. Huang, S. Yan, and Z. He, "SHELTER: Extending arm CCA with isolation in user space," in *Proceedings of the 32nd USENIX Security Symposium (Security)*, Anaheim, CA, Aug. 2023.

[83] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang *et al.*, "Enabling rack-scale confidential computing using heterogeneous trusted execution environment," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1450–1465.

Appendix A. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper proposes PORTAL, a secure and efficient device I/O interface designed for Arm CCA on mobile Arm SoCs. The key innovation of PORTAL lies in achieving secure device I/O through strict memory isolation without relying on traditional memory encryption techniques. This approach addresses the significant performance and power overheads associated with memory encryption, making PORTAL a promising solution for dynamic peripheral integration in resource-constrained environments.

A.2. Scientific Contributions

- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) This paper addresses a long-known Issue. Enabling secure I/O on mobile devices is important. Existing methods for secure device I/O on Arm CCA rely on memory encryption, which introduces considerable performance and power overhead. PORTAL directly tackles this problem by leveraging Arm CCA GPC and SMMU to achieve isolation, making it relevant for performance-sensitive mobile devices.
2) Provides a Valuable Step Forward in an Established Field: While secure device I/O for Arm CCA is a well-explored area, PORTAL offers a novel approach by leveraging access control, memory isolation instead of encryption, achieving significant performance gains while maintaining similar security guarantees.