# OPENCCA: An Open Framework to Enable Arm CCA Research

Andrin Bertschi
*ETH Zurich*
*Zürich, Switzerland*
*andrin.bertschi@inf.ethz.ch*

Shweta Shinde
*ETH Zurich*
*Zürich, Switzerland*
*shweta.shinde@inf.ethz.ch*

***Abstract*—Confidential computing has gained traction across major architectures with Intel TDX, AMD SEV-SNP, and Arm CCA. Unlike TDX and SEV-SNP, a key challenge in researching Arm CCA is the absence of hardware support, forcing researchers to develop ad-hoc prototypes on CCA emulators and non-CCA Arm boards. This approach leads to high barriers to entry or duplicated efforts leading to unsound and inconsistent comparisons. To address this, we present OPENCCA, an open research platform that enables the execution of CCA-bound code on commodity Armv8.2 hardware. By systematically adapting the software stack (including bootloader, firmware, hypervisor, and kernel), OPENCCA emulates CCA operations for performance evaluation while preserving functional correctness. We demonstrate its effectiveness with typical life-cycle measurements and case-studies inspired by prior CCA-based papers on an easily available Arm v8.2 Rockchip board that costs $250.**

## 1. Introduction

Intel and AMD, leaders in x86 platforms, enable confidential computing with TDX and SEV-SNP respectively [1], [2]. Arm, in 2021, announced Realm Management Extension (RME) which is an optional feature on Armv9A to enable confidential computing architecture (CCA) [3]. Arm has rolled out support for building CCA-enabled platforms, including emulator and software changes [4]–[6]. As witnessed by several recent works in top-tier venues, Arm CCA has already received traction by providing a rich and fertile ecosystem for innovative designs to improve confidential computing [7]–[26].

One fundamental hurdle in using Arm CCA is the lack of hardware support, since no public Arm CPU supports CCA yet. Arm's system emulator, the Fixed Virtual Platform (FVP) provides functional correctness and instruction counts. But, it is not cycle accurate which hinders any performance measurements. To this end, researchers have resorted to building best-effort performance prototypes where they *transplant* their CCA-bound implementation to an Arm board that does not have CCA support (e.g., Armv8) and replace the CCA instructions and functionality with dummy operations. Our survey shows that of the 19 papers released in the last 4 years, 15 spend significant effort on such transplantation (see Tab. 1). This not only creates unnecessary work for the authors, it also detracts other researchers from experimenting on Arm CCA due to high barrier for entry. We observe that researchers choose different Arm boards for their performance prototypes.

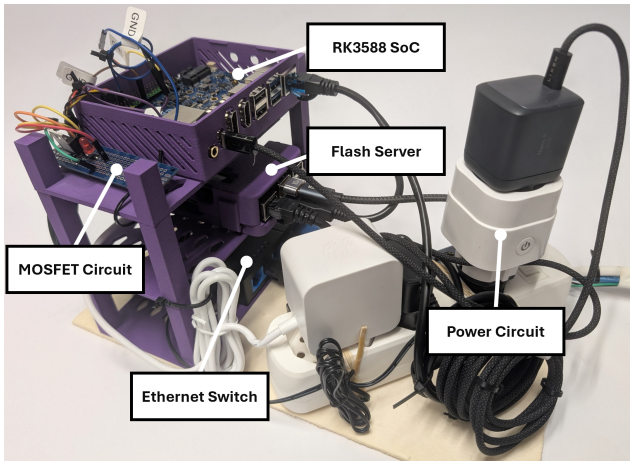


Figure 1. OPENCCA tooling. The RK3588 connects over ethernet to a flash server (Raspberry Pi). It controls a MOSFET and power circuit to flash new firmware and exposes UART access.

These efforts are focused on the scope of the paper and do not lend to full-fledged prototypes that are reusable by others. Thus researchers are repeating the same efforts across boards, which is not the best use of their time. The fragmentation in choice of boards further makes it challenging to compare the performance across different papers, especially since some boards are 10 years old, cost $10,000, or are no longer available (e.g., Juno R2). Worst of all, 4/19 papers omit hardware-based performance evaluation entirely due to lack of CCA support.

Our motivation to build OPENCCA stems from: (a) our research experience prototyping early CCA works [12], [27]; (b) our roadblocks in subsequent performance prototypes [20], [22]; (c) personal communications with researchers in the community who want to benchmark performance for CCA-based defenses; and (d) large time gaps between Arm announcements and hardware rollouts (e.g., SEL2 announced in 2017, remains unavailable on commodity boards). Taking inspiration from the *transplantation to performance prototype* approach from prior efforts on CCA [7]–[17], [23]–[25], we build OPENCCA (Fig. 1) that is capable of executing CCA-bound code on a non-CCA Arm board. Specifically, we aim to enable lift-and-shift from Arm FVP to Arm board. We first surveyed the most suitable boards for our goal (see Tab. 2) and chose the RK3588 Radxa Rock 5B due to its easy availability, support system, and affordability. We then systematically analyzed the entire stack from bootloader, firmware, hypervisor, drivers, host and guest

TABLE 1. SURVEY OF WORK ON CCA AS OF FEBRUARY 2025. COLUMN 2-3 SHOW IF THE WORK IS IMPLEMENTED ON SIMULATION SOFTWARE AND ARM BOARD RESPECTIVELY, WITH BOARD ARCHITECTURE VERSION IN COLUMN 4. COLUMN 5-6 INDICATE IF IT IS OPEN-SOURCE (✓), CLOSED (✗), OR NOT APPLICABLE (N/A).

| Related Work | Sim. | Board | Arch. | FVP | Board |
|---|---|---|---|---|---|
| Cage [8] | FVP | Juno R2 | 8.0 | ✓ | ✓ |
| Shelter [7] | FVP | Juno R2 | 8.0 | ✓ | ✗ |
| Scrutinizer [10] | FVP | Juno R2 | 8.0 | ✓ | ✗ |
| ACAI [12] | FVP | Zynq UltraScale+ | 8.0 | ✓ | ✗ |
| TZ & CCA [11] | FVP | Juno R2 | 8.0 | ✗ | ✗ |
| HitchHiker [9] | FVP | Juno R2 | 8.0 | ✗ | ✗ |
| FortifyPatch [23] | FVP | Raspberry Pi 3B | 8.0 | ✗ | ✗ |
| RContainer [13] | FVP | RK3399 Firefly | 8.0 | ✗ | ✗ |
| CubeVisor [24] | FVP | RK3399 Rock 4B | 8.0 | ✗ | ✗ |
| TwinVisor [25] | FVP | HiSilicon Kirin 990 | 8.2 | ✓ | ✗ |
| Portal [14] | FVP | OrangePi 5 Plus | 8.2 | ✗ | ✗ |
| Des. & Ver. [15] | FVP | Neoverse N1 | 8.2 | ✗ | ✗ |
| virtCCA [16] | – | Undisclosed | 8.4 | N/A | ✗ |
| Sharing [17] | – | AmpereOne | 8.6 | N/A | ✗ |
| CPC [19] | FVP | SEV SNP (x86) | – | ✗ | ✗ |
| GuaranTEE [18] | FVP | – | – | ✓ | N/A |
| Devlore [20] | FVP | – | – | ✗ | N/A |
| BarriCCAde [21] | QEMU | – | – | ✗ | N/A |
| Aster [22] | QEMU | – | – | ✗ | N/A |

kernel to identify what aspects need to be adapted to emulate CCA operations—for performance and compatibility. Finally, we added CCA awareness to all these components while carefully selecting non-CCA operations that can best estimate CCA overheads. OPENCCA presents a standard development and measurement framework to evaluate CCA-based solutions, akin to efforts on Intel SGX [28] and RISC-V [29]. Similar to OpenSGX [28], in its current form OPENCCA does not aim to enforce CCA equivalent security on non-CCA boards. Lessons from ongoing efforts on virtCCA [16] can be coupled with OPENCCA to address this limitation.

Our choice of a commodity board allows researchers to take their approach implemented on Arm FVP and lift-and-shift it to OPENCCA for performance estimates and compatibility. We demonstrate this by showcasing out-of-box use of OPENCCA for reporting typical life-cycle metrics. Next, we run standard benchmarks on FVP and OPENCCA to show that we preserve functionality. Lastly, we then build two representative case-studies of CCA-based designs from prior works, under five hours each. OPENCCA is available at [30].

## 2. Arm CCA in a Nutshell

Prior to CCA, computation on Arm processors could execute in either normal or secure worlds. Arm CCA extends Arm's ISA with Realm Management Extensions (RME) to enable 2 new worlds: realm, and root (Fig. 2(a)). To isolate these worlds, RME adds Granule Protection Checks (GPCs) to each processing element (e.g., cores) which look up Granule Protection Tables (GPTs). The GPTs map physical addresses to their corresponding worlds and are programmed by the trusted firmware (TF-A) that executes in the root world (Tab. 3). In addition to the worlds, the Arm ISA also allows computation to

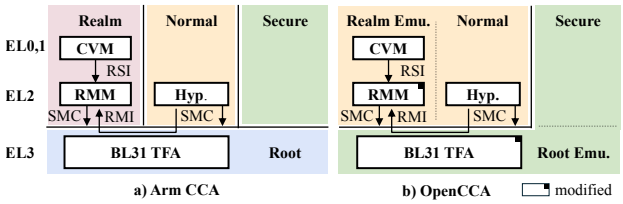


Figure 2. (a) Arm CCA. (b) OPENCCA implements the realm world in the normal world (EL2-EL0) and root world in EL3 (i.e., secure world).

execute in one of the 4 exception levels (EL0-EL3). With CCA, only root world computation can execute in EL3.

CCA enables the creation of confidential VMs (CVMs) in the realm world. To isolate mutually distrusting co-resident CVMs, CCA uses a trusted Realm Management Monitor (RMM) in realm EL2 which programs stage-2 translation tables for the CVMs. The RMM exposes a Realm Management Interface (RMI) to the hypervisor and a Realm Service Interface (RSI) to the CVMs. The hypervisor invokes RMIs to create and manage the CVMs. Finally, the RMM and the hypervisor use Secure Monitor Calls (SMCs) to communicate with TF-A.

## 3. Design

We design OPENCCA to meet the following goals:

- **G1** Enable Arm CCA on commodity Armv8 hardware with minimal software modifications and preserving functionality.
- **G2** Demonstrate adaptability and ease of integration as a research framework.

**Exploring Hardware Boards.** In Tab. 2 we list our survey on suitable boards for OPENCCA. We highlight the boards used by related works in gray color. The Orange Pi 5 Plus board uses the same RK3588 as Radxa Rock 5b (OPENCCA), however, its software stack is currently more outdated. RK3399, although cheaper than RK3588, has older Cortex cores and, as a result, fewer hardware features. Interestingly, the Radxa Orion O6 board has modern cores and interrupt controller, but currently lacks TF-A support and a publicly available technical reference manual (TRM). In general, we consider a board suitable if two key conditions are met: (1) EL3 must be flashable with custom firmware, i.e., not locked down by the vendor; and (2) the board must offer at least rudimentary support of TF-A with source code available. The RK3588 fulfills both. Furthermore, it is affordable, has good software support [31] and a wide set of peripherals, making it a suitable platform for OPENCCA at the time of writing.

**Main Insight.** We aim to lift-and-shift CCA-bound software tested on Arm FVP to RK3588. We enlighten the three core firmware components on RK3588: the TF-A (Sec. 3.1), RMM (Sec. 3.2), and U-Boot (Sec. 3.3) such that they can emulate RME. This way, all other components such as hypervisor kernel, guest kernel, and the VMM remain unchanged—a developer can test it on Arm FVP with RME support and then transplant it to OPENCCA. This design choice means that OPENCCA remains independent of any particular hypervisor or CVM configuration used (see Fig. 2(b.)).

TABLE 2. EXPLORING HARDWARE BOARDS FOR OPENCCA. TF-A CODE: TF-A PORTED TO BOARD AND SOURCE CODE PUBLICLY AVAILABLE. GRAY HIGHLIGHTED: BOARDS USED IN RELATED WORKS IN TAB. 1. GREEN AND BOLD: BOARD USED IN OPENCCA.

| Board | Released | SoC | GIC | Price (USD) | Cores | GPU | TF-A Code |
|---|---|---|---|---|---|---|---|
| Intel Stratix 10 SX DK | 2013 | Intel Stratix 10 | GICv2 | 9,000 | A53 | N/A | ✓ |
| AmloGIC Meson S905 (GXBB) | 2015 | S905 | GICv2 | unknown | A53 | Mali 450 | ✓ |
| HiKey | 2015 | Kirin 620 | GICv2 | 75-100 | A53 | Mali 450 | ✓ |
| Arm Juno r2 | ca. 2015 | Juno r2 SoC | GICv2 | 10,000 | A72, A53 | Mali T624 | ✓ |
| NXP i.MX7 WaRP7 | 2016 | i.MX 7 Solo | GICv2 | 100 | A7, M4 | N/A | ✓ |
| A64-OLinuXino | 2016 | Allwinner A64 | GICv2 | 100 | A53 | Mali 400 | ✓ |
| AmloGIC Meson S905x (GXL) | 2016 | S905x | GICv2 | unknown | A53 | Mali 450 MP3 | ✓ |
| NXP i.MX 8QM MEK | 2016/17 | i.MX 8QM | GICv3 | 1,200 | A72, A53, M4F | GC7000XSVX | ✓ |
| NXP i.MX 8MQ EVK | 2016/17 | i.MX 8MQ | GICv3 | 500 | A53, M4 | GC7000Lite | ✓ |
| NXP i.MX 8ULP EVK | 2016/17 | i.MX 8 ULP | GICv3 | 550-650 | A53, M33 | GC520 | ✓ |
| Xilinx Zynq ZCU102 EVK | ca. 2017 | 2FFVB1156E | GICv2 | 3,200 | A53, R5F | Mali 400 RP2 | ✓ |
| AmloGIC Meson A113D (AXG) | 2017 | S400 | GICv2 | unknown | A53 | 2D GFX Engine | ✓ |
| HiKey 960 | 2017 | Kirin 960 SoC | GICv2 | 250 | A73, A53 | Mali G71 MP8 | ✓ |
| AmloGIC Meson S905X2 (G12A) | 2018 | S905x2 | GICv2 | unknown | A53 | Mali-G31 MP2 | ✓ |
| HiKey 970 | 2018 | Kirin 970 | GICv2 | 300 | A73, A53 | Mali G72 MP12 | ✓ |
| Raspberry Pi 3 (B+) | 2018 | BCM2837B0 | custom | 25 | A53 | VideoCore IV | ✓ |
| Intel Agilex 7M HBM2e DK | 2019 | Intel Agilex 7 | GICv2 | 10,000 | A53 | N/A | ✓ |
| Marvell CEx7 CN9132 EVB | 2019 | CN9132 | GICv2 | 600-700 | A72 | N/A | ✓ |
| Ziver MTK8183 Dev. Board | 2019 | MT8183 | GICv3 | 150 | A73, A53 | Mali G72 MP3 | ✓ |
| Raspberry Pi 4 | 2019 | BCM2711 | GICv2 | 35 | A72 | VideoCore VI | ✓ |
| Huawei Mate 30 Pro | 2019 | Kirin 990 | unknown | 300 | A76, A55 | Mali-G76 | ✗ |
| Arm Neoverse N1 SDP | 2020 | Dawn Ares | GICv4.1 | 10,000 | N1 | HDLCD | ✓ |
| Aspeed AST2700 EVB | 2020 | AST2700 | GICv3 | unknown | A35, M4 | AST2700 2D VE | ✓ |
| RK3399 Rock4 | 2021 | RK3399 | GICv3 | <200 | A72, A53 | Mali-T864 | ✓ |
| NVIDIA Jetson TX2 NX DK | 2021 | Tegra X2 | GICv2 | 350 | Denver2, A57 | GP10B | ✓ |
| MediaTek 8186 | 2021 | Kompanio 520 | GICv3 | unknown | A76, A55 | Mali-G52 MP2 | ✓ |
| MediaTek 8192 | 2021 | Kompanio 820 | GICv3 | unknown | A55, A76 | Mali G57 MC5 | ✓ |
| MediaTek 8188 | 2022 | Kompanio 838 | GICv3 | unknown | A55, A78 | Mali G57 MC3 | ✓ |
| MediaTek 8195 | 2022 | Kompanio 1380 | GICv3 | unknown | A55, A78 | Mali G57 MC5 | ✓ |
| Genio 700 (MT8390) | 2023 | MT8390 | GICv3 | 700 | A78, A55 | Mali-G57 | ✓ |
| Orange Pi 5 Plus | 2023 | RK3588 | GICv3 | <200 | A76, A55 | Mali-G610 | ✓ |
| Supermicro MegaDC (Server) | 2023 | AmpereOne | unknown | unknown | custom built | N/A | ✗ |
| Raspberry Pi 5 | 2023 | BCM2712 | GICv2 | 120 | A76 | VideoCore VII | ✓ |
| **Radxa Rock 5b** (OPENCCA) | **2023** | **RK3588** | **GICv3** | **250** | **A76, A55** | **Mali-G610** | ✓ |
| NXP i.MX 93 QS EVK | ca. 2024 | i.MX 93 | GICv3/v4 | 300 | A55, M33 | N/A | ✓ |
| Arrow AXE5-Eagle DK | 2024 | Intel Agilex 5 | GICv3 | 900-1,000 | A55, A76 | N/A | ✓ |
| Radxa Orion O6 | 2024 | Cix CD8180 | GICv4 | 500-600 | A720, A520 | Imtls. G720 MC6 | ✗ |

### 3.1. Enabling OPENCCA in TF-A

Since the RK3588 lacks the realm world, OPENCCA first introduces a new world in software and addresses the absence of Granular Protection Table (GPT) instructions to enable OPENCCA in TF-A.

**Introducing a New World.** The Arm architecture does not automatically bank registers per world. Instead, the EL3 runtime firmware saves and restores the CPU context when switching between worlds. With RME, the security context of a core is defined by two bits in the Secure Configuration Register (`SCR_EL3`), where the combination $\{\texttt{NS}, \texttt{NSE}\} = 11$ denotes the realm world (Tab. 3). Without RME, the hardware lacks the NSE bit, making it impossible to distinguish the realm world at the architectural level. During a world transition (e.g., SMC to schedule a CVM), TF-A saves the CPU state in a memory region specific to the current core and world. To switch worlds, TF-A updates the context to match the target world. When EL3 exits, the CPU restores this context and continues execution in the new world.

To compensate for the missing NSE bit, OPENCCA introduces a software bit: NSE' and stores it in the world context of each CPU. Both NS (from `SCR_EL3`) and NSE' (from memory) are then referenced to determine the active world. This method allows for handling both synchronous (e.g., SMC) and asynchronous (e.g., interrupts) EL3 entries, but with added memory lookup. By maintaining NS=1, OPENCCA ensures that the realm world operates in the architectural normal world. Since we now multiplex both the realm and normal world within the architectural normal world, we must flush the EL2 TLB on each context switch between RMM and normal world hypervisor. This aligns with **G1** as we preserve functionality and keep the hypervisor agnostic of OPENCCA. We also maintain an alternative patchset (outside the scope of this evaluation) that eliminates the need for TLB flushes by requiring a small change in the hypervisor to reserve an address space identifier (ASID) range for the RMM [30].

TABLE 3. NS AND NSE (`SCR_EL3`) ON ARMV9 TO SELECT EL0/1/2 WORLD. ROOT WORLD IS NOT ENCODED, ALWAYS IN EL3.

| World | NS | NSE | World | NS | NSE |
|---|---|---|---|---|---|
| Normal | 1 | 0 | Secure | 0 | 0 |
| Realm | 1 | 1 | Root | - | - |

**Absence of GPT Instructions.** With the realm world in place, the challenge now shifts to memory isolation. RME relies on GPTs to program world isolation, but RK3588 lacks the instructions to configure them. Specifically, RME introduces the GPT base register (`GPTBR_EL3`), GPC configuration register (`GPCCR_EL3`), and TLB instructions (`TLBI PAALLOS`) which are unavailable. To compensate, OPENCCA replaces these with dummy system registers (`AFSRx`) and returns predefined values. For instance, instead of querying the platform for GPC configuration, OPENCCA returns a fixed configurable value. We approximate the TLB instruction with a flush of the entire TLB cache (all shareability domains, exception levels, worlds). This follows prior work [7]–[10], [12], [14].

**Building the GPTs.** With GPT instructions substituted, OPENCCA can now use them to initialize the protection tables. The RK3588 does not utilize TF-A stage 2 bootloader (BL2) for early boot initialization. Instead, it relies on U-Boot to set up the platform. Since TF-A implements

GPT initialization in BL2 and BL2 is not deployed on the RK3588, we integrate this functionality into BL31. This allows OPENCCA to account for GPT overhead during system boot without having GPC available on the board.

### 3.2. Enabling OPENCCA in RMM

With TF-A modified for OPENCCA, the system can now delegate memory between worlds and execute a bare-metal payload in realm EL2. The next step is to enable OPENCCA in the RMM to run CVMs on the RK3588.

**No Small Translation Tables.** The RMM uses identity mappings in low virtual memory (TTBR0) to map data and code that is shared across cores. It places core-private memory in high virtual memory (TTBR1). For TTBR1, the RMM only requires an address space size of 2 MB. As a result, it uses Small Translation Tables (TTST) for high-memory mappings. TTST is a feature introduced in Armv8.4 and decreases the lower limit on the size of translation tables [32]. As such, the page table walk is shorter because the MMU traverses fewer levels to reach the leaf node. The RK3588 lacks this feature. Hence, we change TTBR1 mappings to use an address space size of 64 MB. This forces a base level of 2, the smallest level supported on the RK3588. With the RMM's own memory mappings correctly configured, OPENCCA can now address CVM stage-2 mappings.

**Stage-2 Mappings without FWB.** Force Write Back (FWB) is an Armv8.3 feature that enables the hypervisor to enforce write-back behavior on non-cacheable translation mappings set by the VM [32]. This eliminates the need for explicit cache maintenance because guest writes become immediately visible to the hypervisor. In the RMM, stage-2 management requires FWB; however, RK3588 does not implement this feature. As a result, we encountered issues during the CVM early boot process, leading to stale data and inconsistent crashes. To mitigate this, OPENCCA changes the memory attributes in stage-2 and adds cache maintenance instructions.

**Timer Virtualization.** Armv8.6 introduces new system registers to control time for VMs with Enhanced Counter Virtualization (ECV) [32]. This functionality is missing on the RK3588. This includes an offset for time (`CNTPOFF`) between the guest and the hypervisor, which can be useful in scenarios like live migration, where counters may differ between source and destination hosts. The RMM specification mandates that these offsets must be fixed for the lifetime of a realm [33]. Since we do not have ECV available, OPENCCA sets these offsets to zero.

The RMM also controls the firing of timer interrupts; when a physical timer occurs, the RMM traps the CVM into EL2 and transitions control to the normal world hypervisor. Subsequently, the hypervisor delivers a virtual interrupt to the CVM. This design keeps the RMM's TCB minimal and delegates interrupt management to the hypervisor. The RMM uses EL2 system registers to mask the physical timer, preventing an immediate exit of the CVM upon re-entry (`CNTPMASK`). Due to missing RME and ECV, this mechanism is unavailable and leads to CVM stalls on the RK3588. OPENCCA addresses this by overriding timer masking with EL0 registers, ensuring the guest can continue making progress (`CNTP_CTL_EL0`).

TABLE 4. SYSTEM STACK IN OPENCCA. FIRMWARE BASED ON LATEST VERSIONS AVAILABLE AT THE OUTSET OF PROJECT. NO CHANGES IN SOFTWARE STACK.

| Firmware Stack | | | | Software Stack | |
|---|---|---|---|---|---|
| **Component** | **Version** | **Modification** | **RK3588 Specific** | **Component** | **Version** |
| TF-A [36] | v2.11 | 940 LoC | 59% | Linux Hyp. [31], [37], [38] | v6.12.0 |
| TF-RMM [39] | v0.5.0 | 1440 LoC | 16% | Linux CVM [37], [38] | v6.12.0 |
| U-Boot [40] | v2024.01 | 216 LoC | 0% | kvmtool [41] | v3/cca |

**FP Traps and Timer Interplay.** Until this point, the CVM successfully boots to EL0, but experiences stalls due to traps caused by lazy floating-point (FP) state restoration. As an optimization, the RMM only restores FP registers when the CVM actually uses them. Specifically, the RMM traps on the first FP used, restores the FP state and then disables further traps until the next transition to the hypervisor. This leads to a loop where the CVM traps for FP state restoration, the RMM restores the FP state, but before execution can proceed, a physical timer interrupt forces a transition to the hypervisor, repeating the cycle. To prevent this, OPENCCA keeps the timer masked if the previous CVM exit was triggered by FP state restoration.

### 3.3. Enabling OPENCCA in U-Boot

At this stage, OPENCCA successfully boots a CVM in realm EL1/EL0. Bundling the RMM into the firmware image, although presented last, is the first step during compilation and a prerequisite for the boot process.

**Bundling a new Firmware.** U-Boot uses Binman [34] to package multiple firmware components into a single image. We introduce the RMM as a new firmware component and ensure that the firmware chain includes both TF-A and the RMM.

## 4. Implementation

We prototype OPENCCA on Arm's reference implementation of CCA. Tab. 4 summarizes our firmware stack and lines of code (LoC) modified to support OPENCCA. OPENCCA introduces minimal; 940 (+0.3%), 1440 (+6%) and 216 (+0.01%) LoC for TF-A, RMM and U-Boot respectively (**G1**), compared to the total sizes of 309K, 25K and 1.5M LoC (C/C++/ASM). We keep hypervisor, CVM, and VMM (kvmtool) unchanged. We see that platform-specific code constitutes a significant portion of the overall code changes. This includes a new console driver, a new memory layout for the RK3588, and GPT code we moved from BL2 in TF-A to BL31. We refer to our extended version for more implementation details [35].

## 5. Evaluation

We demonstrate OPENCCA functionality and report runtime measurements.

**Experimental Setup.** We boot Linux in a CVM with 1 vCPU, and 256MB and 1GB of RAM. We pin kvmtool to core 2 and isolate the core from general-purpose scheduling (`isolcpus`). The normal world hypervisor uses the 4 Cortex A55 cores on the RK3588 (see Tab. 5) and the CPU governor userspace, that we set to a fixed frequency of 1.8GHz. For instructions and cycles on the RK3588,

TABLE 5. RADXA ROCK5B BOARD SPECIFICATIONS.

| Component | Specification | Component | Specification |
|---|---|---|---|
| **SoC** | Rockchip RK3588 | **GPU** | Mali-G610 |
| **Board** | Radxa Rock5b v1.46-2023-11.06 | **RAM** | 16 GB |
| | | **Storage** | 64 GB eMMC |
| **CPU** | 4× Cortex-A76 @ 2.4 GHz<br>4× Cortex-A55 @ 1.8 GHz | **IRQ** | GICv3 |
| | | **PCI** | PCIe 3.0, 2.0 |

we use Performance Monitor Unit (PMU) with events: Instructions Retired and Cycles. We build TF-A and RMM in release mode and disable all but ERROR output.

**PMU across Worlds.** TF-A and RMM manage performance counters by saving and restoring them for EL2 and EL1 upon context switch. However, for OPENCCA's evaluation, we are interested in measuring overhead across all exception levels. To achieve this, we introduce a patch set that bypasses the standard save-and-restore mechanism for PMU counters across worlds and exception levels.

**Verifying Results with FVP.** For completeness, we benchmark a CPU-intensive workload on both RK3588 and FVP using identical binaries (hypervisor, CVM, payload), and manually verify that the results are consistent.

**Benchmarks.** To benchmark OPENCCA, we review related works [7], [8], [12], [14] and identify key performance metrics they use to evaluate overheads on Arm CCA. These include RMI and context switch costs, VM boot overheads, and GPT costs across different setups (i.e., a baseline against new changes introduced by research). We report cycles and instructions with standard deviation and average CVM boots across 100 iterations and SMC/RMI benchmarks across 5 million invocations.

In Tab. 6, we show an overview of OPENCCA runtime measurements. As expected, CVM boot increases with larger RAM sizes due to the additional memory delegation required. We further compare OPENCCA against a *Two-GPT* case study (see Sec. 6) and observe an overhead of 1.19% in instructions and 1.15% in cycles when booting a CVM with 1GB of RAM. For context switch costs, an SMC round trip that saves and restores the world context without invoking a service in TF-A incurs 182 instructions and 421 cycles. Similarly, an RMI round trip that directly returns from the RMM requires 932 instructions and 3370 cycles, which is 213 cycles less than an RMI that queries the version. Microarchitectural noise affects short calls; grouping multiple calls into a batch may reduce variance.

## 6. Case Studies

We address **G2** and show OPENCCA's adaptability by reimplementing prior designs, reporting implementation time and modified LoC.

**Two-GPT.** In this case study, we implement a dual GPT mechanism building on designs proposed in [7]–[10], [12]–[14], [20], [22]–[24]. We introduce a second GPT while retaining the existing GPT. During system boot, we mark all memory in GPT2 as root world. Upon memory delegation (`RMI_GRANULE_DELEGATE`), GPT1 marks the memory as realm world, while GPT2 marks it as normal world. During memory undelegation (`RMI_UNGRANULE_DELEGATE`), the process is reversed, restoring normal world in GPT1 and root world in GPT2. We modify OPENCCA memory layout to reserve space for both GPT data structures. We use the existing GPT1 code as a template and duplicate it for GPT2. In total, we change 7 files in TF-A (2348 lines added, 13 deleted), and the implementation took 4 person hours. Delegating a single page with Two-GPT takes 3488 (+21.7%) instructions and 8654 (+8.3%) cycles compared to delegation with a single GPT.

**Shadow GPT.** We implement a GPT management design that Shelter [7] uses for Shelter-Apps. For GPT construction, Shelter creates a shadow GPT, a pre-configured template that is copied instead of being built from scratch. We modify RK3588 memory layout to reserve space for new GPTs. In total, we change 10 files in TF-A (1398 lines added, 24 deleted) and the implementation took 5 person hours. Creating a shadow GPT takes 50.86M instructions and 34.61M cycles.

TABLE 6. EVALUATION, RT: ROUND TRIP, DELEGATE: 4KB

| Benchmark | Mean | | Stdev | | Scale |
|---|---|---|---|---|---|
| | Instr | Cycles | Instr | Cycles | |
| OPENCCA | | | | | |
| CVM Boot 256 MB | 1900 | 2647 | 6 | 15 | 1M |
| CVM Boot 1 GB | 2015 | 2869 | 8 | 18 | 1M |
| RMI Delegate | 2865 | 7988 | 187 | 365 | 1 |
| RMI Version | 994 | 3583 | 120 | 222 | 1 |
| RMI RT | 932 | 3370 | 115 | 209 | 1 |
| SMC RT | 182 | 421 | 44 | 68 | 1 |
| *Two-GPT* Case Study | | | | | |
| CVM Boot 256 MB | 1928 | 2690 | 9 | 10 | 1M |
| CVM Boot 1 GB | 2039 | 2902 | 7 | 18 | 1M |
| RMI Delegate | 3488 | 8654 | 182 | 372 | 1 |

## 7. Using OPENCCA for Research

Prior work uses FVP's instruction counts as a proxy for performance (Tab. 1). It is not a meaningful metric; the FVP is not designed for timing analysis but only functional validation, it lacks realistic models for out-of-order execution, superscalar pipelines, and memory hierarchies [42]. In the absence of CCA-enabled hardware, OPENCCA provides a best-effort estimation. We argue that this approach introduces fewer errors than relying on the FVP since the impact of missing RME features (e.g., GPC, realm/root world, memory encryption) is smaller than the inaccuracies caused by the lack of a microarchitectural performance model. Therefore, we recommend using a hardware prototype for performance evaluation and limiting the FVP to functional validation.

**Leveraging Features on RK3588.** Researchers can modify OPENCCA TF-A and RMM to use existing hardware functionality (e.g., cryptography extensions, PCIe, and SMMU integrate into TF-A as RK3588 natively supports them). We recommend identifying the feature in the A-profile list [32], confirming it is in Armv8.2, and verifying its availability in the TRM for RK3588.

**Addressing Missing Features on RK3588.** If the RK3588 lacks a hardware feature (e.g., Memory Tagging, PAC), we propose two solutions: simulate functionality in software and approximate overheads like we do for RME, or upgrade to a newer board. For OPENCCA, a new hardware target with more hardware features requires only a subset of the existing porting work, as more CCA-

related functionality is already natively supported by the hardware. To effectively diagnose issues, we suggest a platform with hardware debugging support [35].

## 8. Conclusion

Researching on Arm CCA remains challenging due to the lack of hardware support, causing inefficiencies and inconsistent performance comparisons. To overcome this, we introduce OpenCCA, an open research framework that enables CCA-bound code execution on affordable Armv8.2 hardware. OpenCCA emulates CCA operations for performance evaluation while maintaining functionality and enabling lift-and-shift from FVP.

**Acknowledgment.** We thank the anonymous reviewers, Supraja Sridhara, and Mark Kuhne for their constructive feedback. Thanks to Dual Tachyon for support with RK3588 debugging.

## References

[1] Intel, "Intel Trust Domain Extensions (Intel TDX)," Accessed: Sep. 2, 2024.

[2] AMD, "AMD SEV-SNP Strengthening VM Isolation with Integrity protection and more," Accessed: Sep. 2, 2024.

[3] ARM, "Arm Confidential Compute Architecture (ARM-CCA)," Accessed: Jan. 1, 2025.

[4] ——, "Trusted Firmware Implementation of the Realm Management Monitor (RMM), Project Page," Accessed: Sep. 2, 2024 .

[5] ——, "Fast Models Fixed Virtual Platforms (FVP), Reference Guide, Version 11.21," Accessed: Jan. 1, 2025,.

[6] ——, "Arm Trusted Firmware-A, Project Page," Accessed: Sep. 1, 2024, .

[7] 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 *USENIX Security*, 2023.

[8] 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 *NDSS*, 2024.

[9] C. Zhang, J. Zeng, Y. Zhang, A. Ahmad, F. Zhang, H. Jin, and Z. Liang, "The HitchHiker's Guide to High-Assurance System Observability Protection with Efficient Permission Switches," in *ACM CCS*, 2024.

[10] Y. Zhang, F. Zhang, X. Luo, R. Hou, X. Ding, Z. Liang, S. Yan, T. Wei, and Z. He, "SCRUTINIZER: Towards Secure Forensics on Compromised TrustZone," in *NDSS*, 2025.

[11] H. Huang, F. Zhang, S. Yan, T. Wei, and Z. He, "SoK: A Comparison Study of Arm TrustZone and CCA," in *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*, 2024.

[12] S. Sridhara, A. Bertschi, B. Schlüter, M. Kuhne, A. Aliberti, and S. Shinde, "Acai: Protecting Accelerator Execution with Arm Confidential Computing Architecture," in *USENIX Security*, 2024.

[13] Q. Zhou, W. Cao, X. Jia, P. Liu, S. Zhang, J. Chen, S. Xu, and Z. Song, "RContainer: A Secure Container Architecture through Extending ARM CCA Hardware Primitives," in *NDSS*, 2025.

[14] F. Sang, J. Lee, X. Zhang, and T. Kim, "PORTAL: Fast and Secure Device Access with Arm CCA for Modern Arm Mobile System-on-Chips (SoCs)," in *IEEE S&P*, 2025.

[15] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and Verification of the Arm Confidential Compute Architecture," in *USENIX OSDI*, 2022.

[16] X. Xu, W. Wang, Y. Wu, C. Wang, H. Zhu, H. Ma, Z. Min, Z. Pang, R. Hou, and Y. Jin, "virtCCA: Virtualized Arm Confidential Compute Architecture with TrustZone," *arXiv preprint arXiv:2306.11011*, 2023.

[17] C. Castes and A. Baumann, "Sharing is leaking: blocking transient-execution attacks with core-gapped confidential VMs," in *ACM ASPLOS*, 2024.

[18] S. Siby, S. Abdollahi, M. Maheri, M. Kogias, and H. Haddadi, "GuaranTEE: Towards Attestable and Private ML with CCA," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, 2024.

[19] J. Chen, Z. Mi, Y. Xia, H. Guan, and H. Chen, "CPC: Flexible, secure, and efficient CVM maintenance with confidential procedure calls," in *USENIX ATC*, 2024.

[20] A. Bertschi, S. Sridhara, F. Groschupp, M. Kuhne, B. Schlüter, C. Thorens, N. Dutly, S. Capkun, and S. Shinde, "Devlore: Extending Arm CCA to Integrated Devices A Journey Beyond Memory to Interrupt Isolation," *arXiv preprint arXiv:2408.05835*, 2024.

[21] M. Schulze, C. Lindenmeier, and J. Rockl, "BarriCCAde: Isolating Closed-Source Drivers with ARM CCA," in *2024 IEEE EuroS&PW*, 2024.

[22] M. Kuhne, S. Sridhara, A. Bertschi, N. Dutly, S. Capkun, and S. Shinde, "Aster: Fixing the Android TEE Ecosystem with Arm CCA," *arXiv preprint arXiv:2407.16694*, 2024.

[23] Z. Ye, L. Zhou, F. Zhang, W. Jin, Z. Ning, Y. Hu, and Z. Qin, "FortifyPatch: Towards Tamper-Resistant Live Patching in Linux-Based Hypervisor," in *ISSTA*, 2024.

[24] J. Chen, Q. Zhou, X. Yan, N. Jiang, X. Jia, and W. Zhang, "CubeVisor: A Multi-realm Architecture Design for Running VM with ARM CCA," in *2024 Annual Computer Security Applications Conference (ACSAC)*, 2024.

[25] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "TwinVisor: Hardware-isolated Confidential Virtual Machines for ARM," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021.

[26] M. Moon, M. Kim, J. Jung, and D. Song, "ASGARD: Protecting On-Device Deep Neural Networks with Virtualization-Based Trusted Execution Environments," in *Proceedings 2025 Network and Distributed System Security Symposium*, 2025.

[27] A. Bertschi, "Protecting Accelerator Execution with Arm Confidential Computing Architecture," Master's Thesis, ETH Zurich, 2023.

[28] P. Jain, S. Desai, S. Kim, M.-W. Shih, J. Lee, C. Choi, Y. Shin, T. Kim, B. B. Kang, and D. Han, "OpenSGX: An Open Platform for SGX Research," in *NDSS*, 2016.

[29] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *EuroSys*, 2020.

[30] A. Bertschi and S. Shinde, "OpenCCA: An Open Framework to Enable Arm CCA Research, Project Page," 2025. [Online]. Available: https://opencca.github.io

[31] Collabora, "Upstream support for Rockchip's RK3588: Progress and future plans," Accessed: Feb. 1, 2025.

[32] ARM, "Feature names in A-profile architecture," Accessed Feb. 1, 2025.

[33] ——, "Realm Management Monitor (RMM) Specification (1.0-REL0)," Accessed: Feb. 2, 2025.

[34] U-Boot, "Binman, Project Page," Accessed: Feb. 2, 2025.

[35] A. Bertschi and S. Shinde, "OpenCCA: An Open Framework to Enable Arm CCA Research, Extended Version," 2025. [Online]. Available: https://opencca.github.io/extended-version

[36] Collabora, "TF-A: Upstream support for Rockchip's RK3588, Commit: 44418fce30," Accessed: Feb. 1, 2025.

[37] ARM, "CCA Host+Guest Patchset for Linux, Version: cca-full/v5+v7, Commit: fad35572db," Accessed: Feb. 1, 2025.

[38] Collabora, "Linux Kernel: Upstream support for Rockchip's RK3588, Commit: f7e1ed901e7," Accessed: Feb. 1, 2025.

[39] ARM, "RMM, Commit: 1313d31ad9," Accessed: Feb. 1, 2025.

[40] Collabora, "U-Boot: Upstream support for Rockchip's RK3588, Commit: 889c316b59e2," Accessed: Feb. 1, 2025.

[41] ARM, "Kvmtool for Arm CCA, Version: cca/v3, Commit: 54241e378," Accessed: Feb. 1, 2025.

[42] ——, "FVP Models, Cycle Accuracy," Accessed Feb. 1, 2025.