Arm Confidential
Compute Architecture
An Introduction

David Hsu

© 2022 Arm
The Future is Built on Arm

Semiconductor IP Business

The global leader in the development of licensable compute technology
- R&D outsourcing for semiconductor companies

Focused on freedom and flexibility to innovate
- Technology reused across multiple applications

With a partnership based culture & business model
- Licensees take advantage of learnings from a uniquely collaborative ecosystem

2000+
Active licenses, growing by 100+ every year

600+
licensees
Industry leaders and high-growth start-ups; chip companies and OEMs

230bn+
Arm-based chips shipped to-date

29bn
Arm-based chips shipped in FY2021
70% of the world’s population use Arm processor technology
Strong Industry Trend Towards Confidential Computing

- **Edge Devices:** increasing use of sensitive private data for ML Models (Biometrics, Health, Shopping, Fintech, Behavioural, NLP...)

- **Regulation:** concerns about personal data privacy are growing rapidly.
  - GDPR (EU)
  - HIPAA (USA)
  - SHIELD (New York)
  - CCPA (California)

Bank of America, Daimler, and Apple partnering with IBM for confidential computing services
Proliferation of Confidential Compute Solutions

Confidential Compute Consortium
- Linux foundation

Members
- Premier
  - arm
  - facebook
  - Google
  - Huawei
  - Intel
  - Microsoft
  - Accenture
  - Red Hat
- General
  - AMD
  - Ant Group
  - Baidu
  - Binance
  - ByteDance
  - Cybereason
  - Cysec
  - Decentrix
  - Dell
  - Fortanix
  - Google Cloud
  - iExec
  - iXIL
  - KINDE
  - KPMG
  - LINE
  - NVIDIA
  - Oracle
  - Philips
  - Prisma Cloud
  - R3
  - Samsung
  - Siemens
  - Swisscom
  - Xilinx
  - Western Digital
- Associate
  - Linaro
  - LaTeX
  - Lumen
  - Microsoft EnConf
  - NCRM
  - NAMI
  - OASIS TSC
  - OASIS Labs
  - OEC
  - Samsung SDS
  - SAS
  - Scyata
  - ServiceNow
  - Silkworm
  - STMicro
  - Thycotic
  - Veracruz
  - Veraison

- Microsoft
  - OpenEnclave
- Redhat
  - Enarx
- Google
  - Asylo
  - Google OAK
- AWS Nitro Enclaves
- Baidu MesaTEE
- Apache Teaclace
- arm
  - Confidential VMs
  - Google confidential VMs
  - Veracruz
  - Veraison
Hardware-backed Isolation for All Workloads

**Mobile / PC / DTV**
- Social Media
- Email
- Browsing
- Content Protection
- OS Services
- Enterprise Apps
- Health / Fitness Apps

**IoT**
- Machine Learning
- Ambient Compute
- Tenant 1

**Cloud**
- Tenant n

**Edge Compute**
- Multi-Vendor Environment

**Automotive**
- OEM Apps
- Personal Data
- Autonomous Driving
- Content Protection

**High end Wearables**
- Health
- SmartKey
- Fitness
- Medical

© 2022 Arm
Arm Confidential Compute Architecture

- Introduced as supplement and optional in Armv9.2-A
- Driven by the expanding need to ensure privacy and security while harnessing data in ever more powerful ways
- Confidential Compute Architecture (Arm CCA) was announced in March 2021 and first specs publicly released in June 2021.
- Arm CCA protects all data and code wherever computing happens
- Protects data in-use by preventing privileged **access** to the resources, whilst retaining the right to **manage** them
Confidential Computing is the protection of data in use, by performing computation in a hardware-based secure environment, to shield portions of code and data from access or modification, even from privileged software.

Arm architecture isolation technologies that enable confidential compute

- **TrustZone®**
  - Protected from Host OS/Hypervisor
  - Platform security use cases for SiP + OEM
  - Specific tool chains/Trusted OS
  - Limited resource

- **Virtualization**
  - Can be used to protect from host primary OS kernel
  - Standard OS development
  - Limited only by available memory

- **S-EL2 Virtualization**
  - Complementary to first two
  - Improves modularity and isolation in TrustZone

- **NEW - Arm CCA**
  - Protects from host OS/Hyp/TrustZone
  - Standard OS development
  - Limited only by available memory

Confidential compute on Arm happening across the ecosystem
Arm Confidential Compute Architecture – Overview

Requirements

1. Secure execution environment that isolates content from more privileged SW: Host OS / Hypervisor, or TrustZone

2. Scale: There should be no specific limits on resources for these environments

3. Standard OS agnostic programming model – no platform specific drivers / toolchains

4. Attestable

Architecture

1. Armv9-A Realm Management Extension (FEAT_RME) introduces Realms and new isolation boundaries so that Realm content cannot be accessed by other Realms, by Host OS / Hypervisor, or by TrustZone

2. Dynamic memory: available memory dynamically moved between Realms or other use, e.g regular processes / VMs

3. Standard ABI to manage Realms

4. System HW architecture and standard ABI specifications to support attestation for Realms
Realms at the Virtual Machine Level

- Realms are supported at the virtual machine (VM) level
  - Very similar to AMD SEV-SNP / Intel TDX
- Hypervisor manages the resources of a Realm VM (memory, scheduling), but cannot access those resources
- Memory is protected in two ways:
  - Isolation: invalid accesses result in faults
    - e.g. Hypervisor access of Realm memory causes a fault to that hypervisor
  - Encryption: mainly for reboot attacks
- Protection covers device DMA and processors
Realm Threat Model

Confidentiality

Hypervisor/Kernel/Secure world read private Realm memory or register state

Device DMA reads private Realm memory or register state

Mitigated by Arch

Integrity

Hypervisor/Kernel/Secure world modifies private Realm memory or register state

Examples:
• Modify saved context
• Writing to Realm pages
• Memory remapping or aliasing

Device DMA modifies private Realm memory or register state

Mitigated by Arch

Availability

Denial of service to a Realm – it is scheduled by OS/Hyp

Realm mounts a DoS attack on the hypervisor

Mitigated by Arch

Indirect SW attacks

Known SW error injection – E.g.: Rowhammer, CLKSCREW

Known side channels E.g.: Spectre / Meltdown

Mitigated by Arch and Realm

Direct HW attacks

Physical DRAM probe and replay

Mitigation requires additional HW

Arch Mitigated by Arm CCA (processor/SMMU/system and FW)

HW

Realm SW in the Realm has the tools to protect itself

Mitigated by Realm

Not mitigated
Arm CCA
Hardware Architecture
Realm Management Extension (RME)
Like most architectures, we provide different levels of privilege:

- EL0: User mode
- EL1: Kernel mode
- EL2: Hypervisor mode

There are two stages of address translation:

- Stage 1 to translate VA → IPA (Intermediate Physical Address, a.k.a. Guest Physical Address)
- Stage 2 to translate IPA → PA

Note:

- Armv8.1 introduced the Virtual Host Extensions that supports running the kernel in EL2
- This is how Linux/KVM is run in many deployments today
Arm CCA Hardware Architecture – TrustZone

- Two physical address spaces: Secure and Non-Secure
- At any point in time a processor can be in one of two security states: Secure or Non-Secure
  - Orthogonal to Exception level
e.g. a processor can be in Non-Secure-EL1 or Secure EL1
- Isolation boundaries based on security state
- EL3: monitor mode - used mainly to switch between security states
- No architectural mechanism to dynamically move memory/devices between Secure and Non-Secure PA
  - Memory for Secure PA is typically statically carved out
- TrustZone typically used by device vendors to provide platform security use case – includes processor and device support
Arm CCA Hardware Architecture – What Does RME Add?

RME adds another two security states and physical address spaces

- **Realm**: new space for confidential compute
  - Realm VMs run in Realm state, but are managed from Non-secure state
  - Realm VMs can access their own private Realm physical address space, and share data via Non-secure physical address space
  - Secure and Realm are mutually distrusting

- **Root**: EL3 FW gets its own private address space

<table>
<thead>
<tr>
<th>Security State/PA space</th>
<th>Non-Secure</th>
<th>Secure</th>
<th>Realm</th>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-secure</td>
<td>Allow</td>
<td>Block</td>
<td>Block</td>
<td>Block</td>
</tr>
<tr>
<td>Secure</td>
<td>Allow</td>
<td>Allow</td>
<td>Block</td>
<td>Block</td>
</tr>
<tr>
<td>Realm</td>
<td>Allow</td>
<td>Block</td>
<td>Allow</td>
<td>Block</td>
</tr>
<tr>
<td>Root</td>
<td>Allow</td>
<td>Allow</td>
<td>Allow</td>
<td>Allow</td>
</tr>
</tbody>
</table>
Arm CCA Hardware Architecture – What Does RME Add?

- Memory can move between physical address spaces **dynamically**
- **Granule Protection Table (GPT):** new data structure that determines the current physical address space of a page
- GPT is controlled by the monitor in EL3
- **Granule Protection Check (GPC):** HW checks GPT on an access and faults invalid accesses
  - Faults in illegal accesses routable to EL2
  - Faults due to broken configuration go to EL3
- HW isolation between components in a security state uses page tables as before
  - Realm-to-Realm separation is based on page tables
- Root, Realm and Secure use encrypted memory
  - Boot ephemeral keys per address space and address tweak

Security State/PA space | Non-Secure PA | Secure PA | Realm PA | Root PA
---|---|---|---|---
Non-secure | Allow | Block | Block | Block
Secure | Allow | Allow | Block | Block
Realm | Allow | Block | Allow | Block
Root | Allow | Allow | Allow | Allow
MMU provides stage 2 isolation
Between Realms
A Granule Protection Check checks Access attributes against the Physical Granule’s attributes Stored in the Granule Protection Table
Arm CCA System Concepts

System Bus

Per Physical Address Space + address tweak encryption
Arm CCA System Concepts

- Page table and GPT isolation plaintext Data
- Encrypted per Physical Address Space

CPU 0
- MMU
- GPC
- L1I$
- L1D$
- L2$

CPU 1
- MMU
- GPC
- L1I$
- L1D$

System Bus

Memory ctrl

MPE

DRAM

DMA

SMMU

GPC

Per Physical Address Space + address tweak encryption
Arm Confidential Compute Architecture

- Arm CCA adds a new environment for 3rd party confidential compute: **Realm world**
- By isolating Realms into their own private Realm World, system wide security analysis is greatly simplified
- With Arm CCA, memory can move dynamically between worlds
- Hypervisor owns management of resources
Arm CCA = RME Hardware + CCA Firmware

Realm Management Monitor (RMM)
- Manages Realm execution environment and inter-Realm isolation
- Allows Non-secure host to create, schedule and manage Realms

Monitor manages
- Context switching CPU execution between security states
- Memory access permissions

Hardware provides isolation primitives
- Additional processor security states
- Memory access control
Published Arm CCA Resources
Enabling system level software developers

• Publicly released specs
  • [https://developer.arm.com/armcca](https://developer.arm.com/armcca)

• Joint Arm / Linaro Tech Event on 23rd June 2021
  • Videos at [https://connect.linaro.org/resources/arm-cca/](https://connect.linaro.org/resources/arm-cca/)

• Confidential specs
  • Device Assignment (DA) - beta
  • Realm Management Monitor (RMM) - alpha
  • Memory Encryption Contexts (MEC) - alpha
  • [https://arm.causewaynow.com/](https://arm.causewaynow.com/)

• FOSDEM talk 5th Feb 2022
  • [https://fosdem.org/2022/schedule/event/tee_arm_cca/](https://fosdem.org/2022/schedule/event/tee_arm_cca/)
Arm CCA ISA Impacts

New system instructions are introduced to be used by the monitor when it updates the GPT:

- **Data Cache Clean/Invalidate to the PoPA**
  - The architecture introduces a point of physical aliasing above which cache lines are tagged with the Physical Address Space they reside in
  - The instruction guarantees that no copies of a PA tagged with a Physical Address Space are above the PoPA

- **TLB invalidation for GPT caching:**
  - GPT association of a page with a physical address space is cacheable in TLBs – A TLB invalidation instruction is added to invalidate the association

Execution and data prediction restriction system instructions extended with new security states - CFP RCTX, CPP RCTX, DVP RCTX
Architectural State Impacts

To keep HW simple most state is reused and replicated across Security States

• Configuration changes:
  • Added two Root registers for Granule Protection Table base address and configuration

• Granule Protection Check faults - treated similar existing MMU faults
  • Additional syndrome information (for loads/stores but also trace/sample profiling)
  • Routing of Granule protection faults

• Fields that represent Security state or Physical address space extended with new encodings
  • Mainly affects self hosted and external debug, trace, performance monitoring and sample profiling
MMU Impacts

• The Physical Address Space of an access is derived from the output of MMU Stage 1/2 and verified by a Granule Protection Check:
  – Non-secure state: PAS is hard-wired to Non-secure
  – Secure state: existing NS bit in Stage 1 descriptor selects between Secure and Non-secure PAS
  – Realm state: new NS bit in Stage 2 descriptor selects between Realm and Non-secure PAS
  – EL3 stage 1: two bits (one new) allow specifying 4 PAS-ès

• The following Translation Regimes are supported in the Realm security state:
  • Realm EL1&0
    – e.g. Stage 1 and Stage2 of an EL1 Realm VM
  • Realm EL2
    – for Real management firmware
  • Realm EL2&0
    – allows RMM to manage the memory for EL0
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה