Trustzone学习

学习资料来源于：

> Learn the architecture - TrustZone for AArch64

1. Overview

该指南中介绍了TrustZone技术。TrustZone通过CPU内置的硬件强制隔离机制，提供了一种高效的系统级安全解决方案。

重点阐述内容：

- TrustZone为处理器架构新增的安全特性
- 支持TrustZone的内存系统设计
- 典型软件架构实现

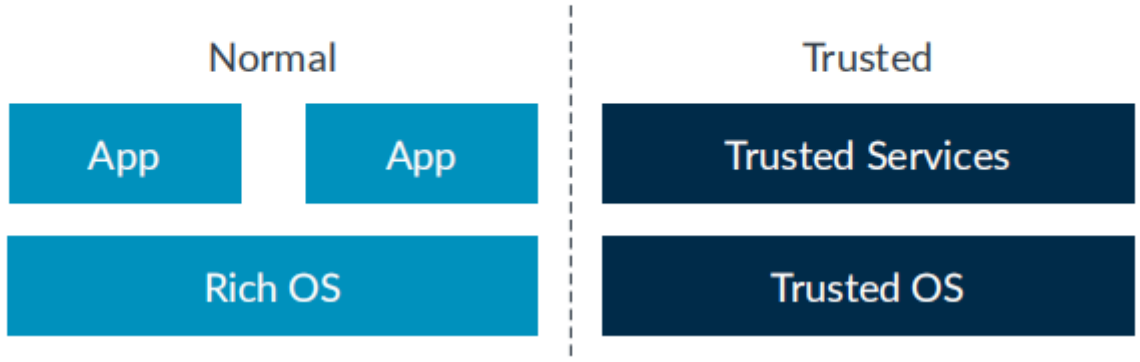
涵盖以下核心知识点：

- 应用场景分析
  - 能举例说明TrustZone如何满足实际安全需求
- 系统架构认知
  - 明确TrustZone系统的安全状态:secure,non-secure
  - 了解物理地址空间划分机制:secure,non-secure
- 关键组件理解
  - 安全监视器（Secure Monitor）的核心功能
  - 需保存/恢复的典型系统状态示例（如寄存器状态、中断配置等）
- 内存系统构建
  - 识别TrustZone内存系统核心组件（如TZASC、TZMA等）
  - 阐述各组件的安全防护作用
- Arm安全标准体系
  - Trusted Base System Architecture(TBSA)的设计目标
  - Trusted Board Boot Requirements(TBBR)规范
- 安全启动机制
  - 详解信任链（Chain of Trust）在设备启动过程中的安全保障机制

2. Waht is Trustzone

TrustZone是Arm A系列架构中集成的安全子系统技术方案。该技术最早于Armv6K中引入，并在后续的Armv7-A与Armv8-A架构中持续演进增强。TrustZone通过硬件机制在系统层面创建了两个隔离的执行环境，如示意图所示。

Figure 2-1: Normal and Trusted world



Normal World运行着丰富的软件生态体系。该环境通常包含：

- 大量应用程序
- 复杂的操作系统（如Linux）
- 虚拟化（Hypervisor）

这类软件栈具有以下特征：

- 规模庞大且复杂度高
- 尽管可采取安全加固措施,但较大的攻击面使其更易遭受安全威胁

相比之下，Trusted World运行着精简的软件栈，即可信执行环境（TEE，Trusted Execution Environment）。TEE的典型架构包括：

- 轻量级内核（Lightweight Kernel）
- 多个可信服务（Trusted Services）

TEE的功能特性:

- 提供密钥管理等核心安全功能
- 软件栈规模显著缩小,攻击面大幅减少,有效降低安全漏洞风险

这种架构设计实现了安全性与功能性的平衡,通过硬件隔离机制确保两者既相互独立又可安全交互。

- Normal World：满足丰富的应用需求
- Trusted World：提供高安全性的基础服务

2.1. Armv9-A Realm Management Extension

ARMv9A中的RME(Realm Management Extension)扩展了TrustZone的概念。

3. TrustZone in the processor

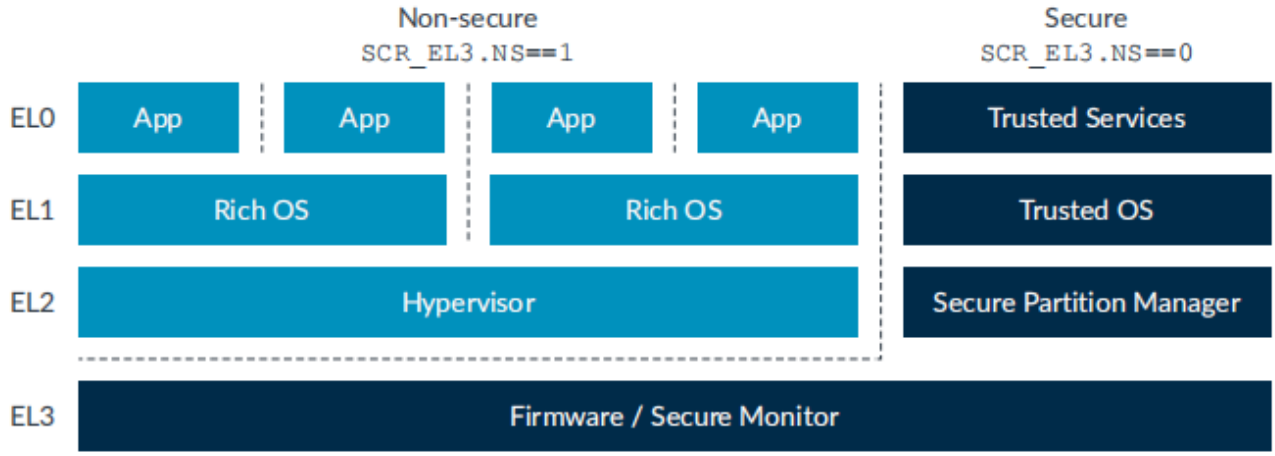
3.1. Security States

ARM架构中有两种架构状态:secure和non-secure.分别对应Trustzone中的trusted world和normal world.

> 在ARMv9-A的RME扩展中有四种架构状态：secure,non-secure,reaml,root.

在异常等级处于EL0,EL1,EL2时，处理器可以处于secure或者non-secure状态，被SCR_EL3的ns bit所控制。EL3只有secure状态。如下所示：

Figure 3-1: Non-secure and Secure state

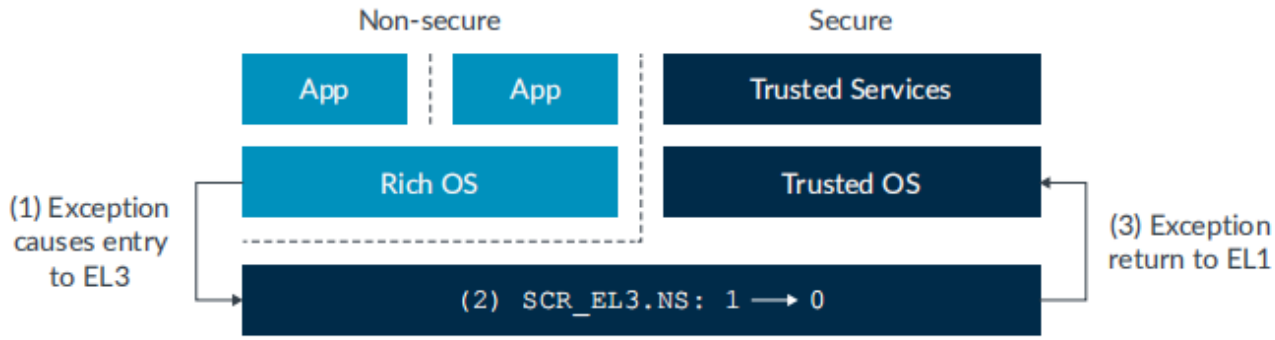


> secure EL2首次于ARMv8-A中提出

3.2. Switching between Security states

无论是从s到ns，还是ns到s,安全状态的更改都需要经过EL3,如下图：

Figure 3-2: Change security state



前图展示了安全状态切换的典型流程步骤。让我们逐步解析这一过程：

1. 进入更高异常等级。触发条件通常有FIQ中断或者SMC调用

2. EL3跳转到相应的异常向量表处理，切换SCR_EL3.NS位

3. 异常返回。处理器从EL3切换到S.EL1

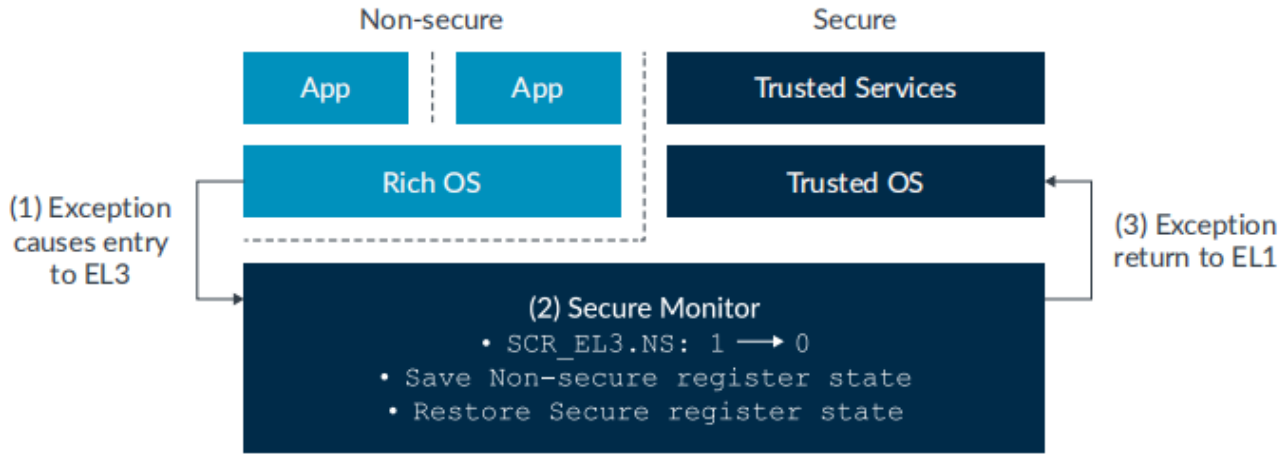
安全状态切换不仅涉及异常等级转换和SCR_EL3.NS位修改，还需考虑处理器状态管理：

- 共享资源
  - 向量寄存器
  - 通用寄存器
  - 大部分系统寄存器
- 状态保存机制

- 硬件不自动保存寄存器状态
- 软件负责保存和恢复寄存器状态，该软件模块通常称为安全监控器（Secure Monitor）

考虑上述因素后，完整的切换流程如下图所示：

Figure 3-3: Secure Monitor



少量寄存器会按安全状态进行分组存储（banked）。这意味着该寄存器有两个副本，并且core会自动使用属于当前安全状态的那个副本。 这些寄存器仅限于处理器在任何时候都需要了解其两种状态的那些寄存器。一个例子是`ICC_BPR1_EL1`，这是一个通用中断控制器（GIC）寄存器，用于控制中断抢占。

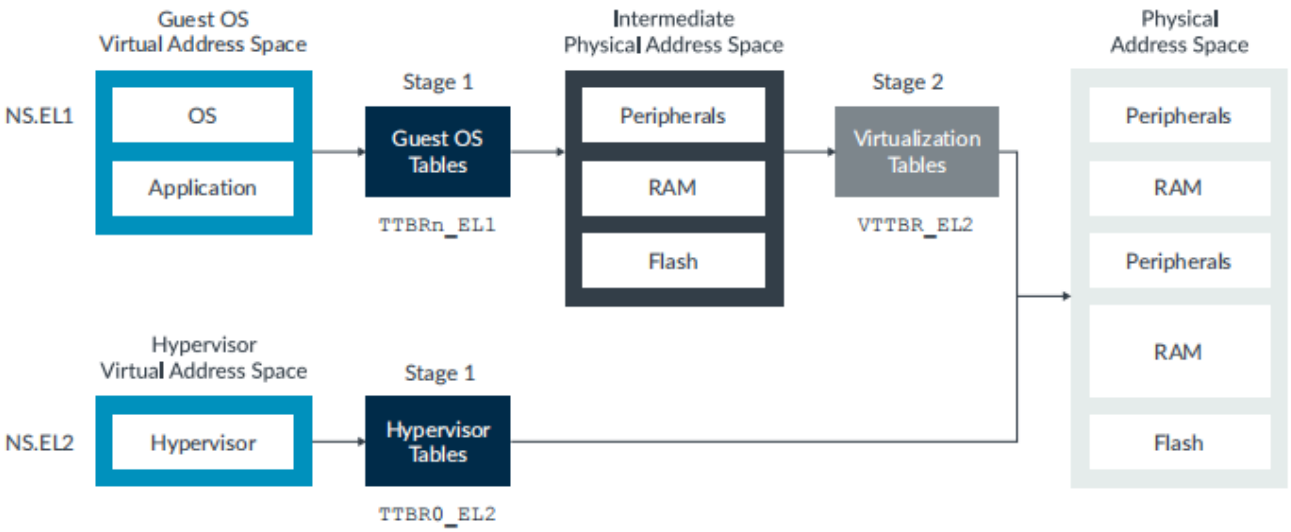
寄存器bank是一种特殊情况，而非普遍规则，并且会在你所使用处理器的架构参考手册中明确指出。当一个系统寄存器是banked时，我们使用（S）和（NS）来标识我们所指的是哪个副本。例如：

> ICC_BPR1_EL1（S）和 ICC_BPR1_EL1（NS）。

3.3. Virtual address spaces

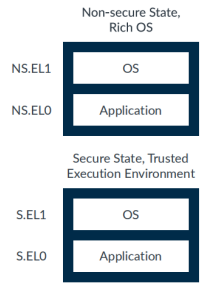
本系列中的内存管理指南介绍了多个虚拟地址空间（即转换机制）的概念。例如，存在一个用于EL0和EL1（EL0/1）的转换机制，以及一个用于EL2的独立转换机制，如下所示：

Figure 3-4: Virtual address spaces



对于secure和non-secure状态，也存在独立的转换机制。例如，存在一种secure EL0/1转换机制和一种non-secure的EL0/1转换机制，如下所示：

Figure 3-5: Secure EL0/1 translation regime and Non-secure EL0/1 translation regime



当使用地址时，一种惯例是在地址前面加上前缀来标识使用哪种translation regime.例如：

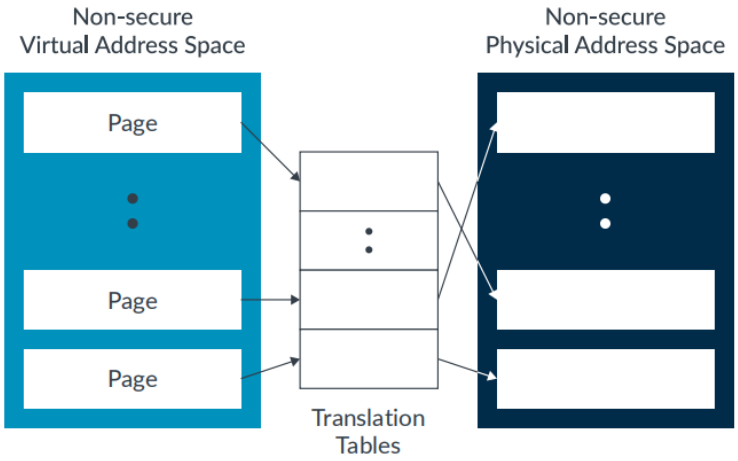
- NS.EL1:0x8000 : Virtual address 0x8000 in the Non-secure EL0/1 translation regime
- S.EL1:0x8000 : Virtual address 0x8000 in the Secure EL0/1 translation regime

3.4. Physical address spaces

除了两种安全状态之外，该架构还提供了两种物理地址空间：secure地址空间和non-secure地址空间。

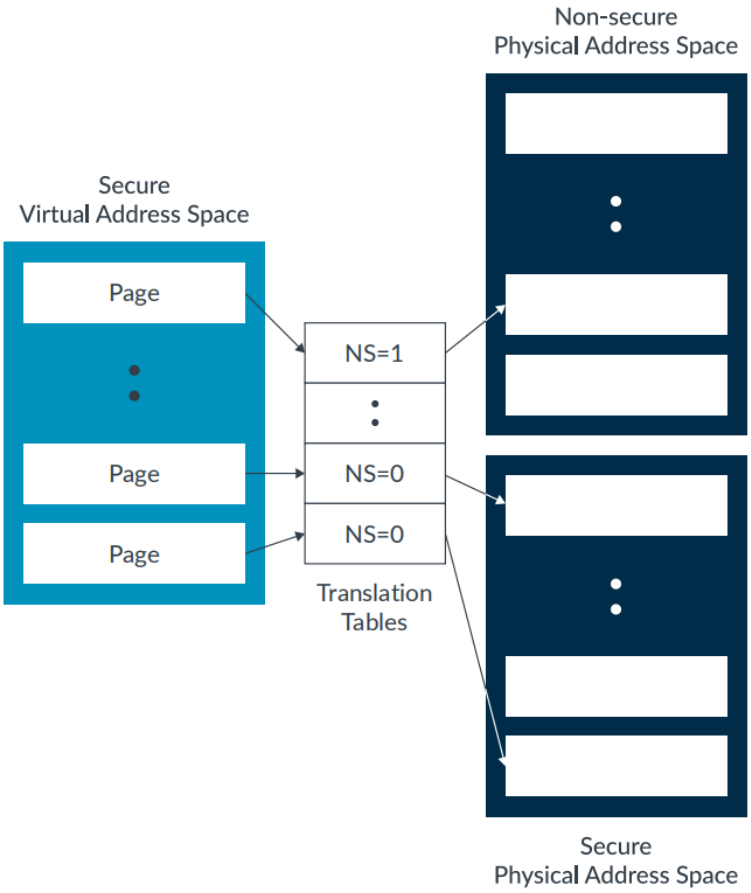
处于非安全状态时，虚拟地址总是转换为非安全物理地址。这意味着处于非安全状态的软件只能看到非安全资源，而永远无法看到安全资源。如下所示：

Figure 3-6: Physical address spaces



处于安全状态时，软件可以访问安全物理地址空间和非安全物理地址空间。页表中的NS位控制着虚拟内存的一个块或页面将转换到哪个物理地址空间，如下图所示：

Figure 3-7: NS bit



> 在secure状态下，如果stage1的MMU被禁用，则所有地址都会被处理为secure.

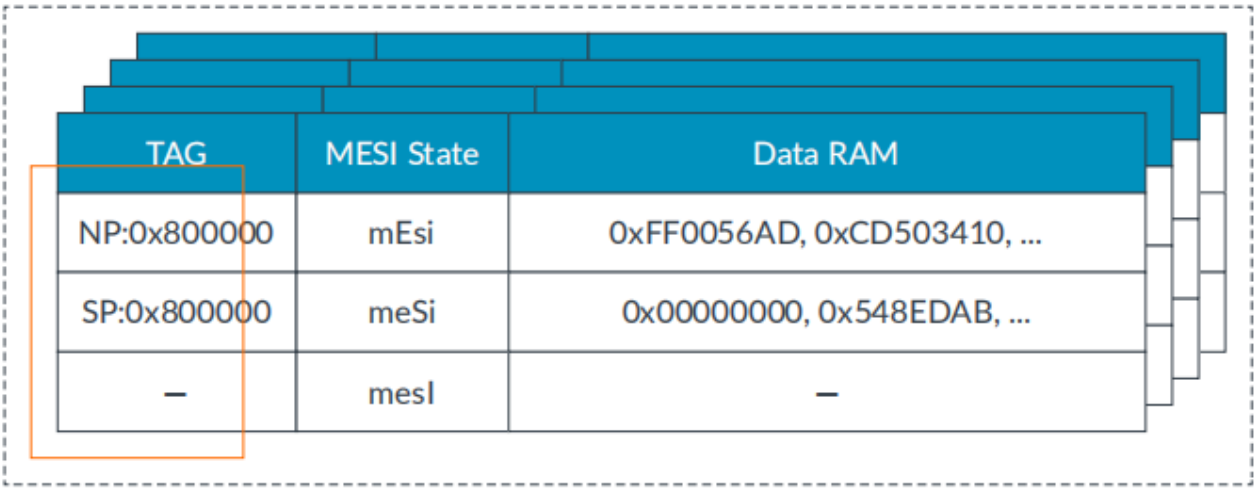
像使用虚拟地址一样，物理地址也需要加前缀：

- NP:0x8000 ： Address 0x8000 in the Non-secure physical address space
- SP:0x8000 ： Address 0x8000 in the Secure physical address space

3.5. Data, instruction, and unified caches

在arm架构的处理器中，data cache通常使用物理地址作为tag.如下所示：

Figure 3-8: Data-caches



Example 4-way data-cache

对非安全物理地址（NP）0x800000 进行缓存查找时，永远不会命中标记为安全物理地址（SP）0x800000 的缓存行。这是因为 NP:0x800000 和 SP:0x800000 是不同的地址。

这也会影响缓存维护操作。以前面图中的data cache为例。如果虚拟地址 va1 映射到物理地址 0x800000, 当软件在非安全状态下发出`DC IVAC, va1`指令时，在非安全状态下，所有虚拟地址都转换为非安全物理地址。因此，va1映射到 NP:0x800000。缓存仅对包含指定地址（在这种情况下是 NP:0x800000）的缓存行进行操作。包含 SP:0x800000 的缓存行不受影响。在非安全状态下，虚拟地址只能映射到非安全物理地址。根据定义，在非安全状态下通过虚拟地址（VA）执行的缓存操作只能针对非安全缓存行。

对于set/way的操作，例如`DC ISW, Xt`，在非安全状态下发出的此类操作只会影响包含非安全地址的缓存行。而在安全状态下，set/way操作会影响包含安全地址和非安全地址的缓存行。 这意味着软件只有在安全状态下才能使整个缓存完全失效。在非安全状态下，软件只能使非安全数据失效。

3.6. Translation Look aside Buffer

TLB会缓存最近使用过的地址转换条目。处理器拥有多种独立的地址转换机制。TLB会记录每个条目所对应的转换机制，包括安全状态。下图给出了一个TLB的示例：

Figure 3-9: Translation Lookaside Buffer (TLBs)

TAG	Translation Regime	NS	VMID	ASID	Descriptor
0x800000	EL1	1	0	5	NP:0x900000
0x500000	EL2	0	–	–	SP:0xA00000
0xC00000	EL1	0	5	3	NP:0x500000

Translation Look-aside Buffer (TLB)

当软件在EL1或EL2发出TLBI时，该软件操作的目标是当前处理器所处的安全状态。因此，在安全状态下执行`TLBI ALLE1`指令会使S.EL0/1地址转换机制下的所有缓存条目失效。

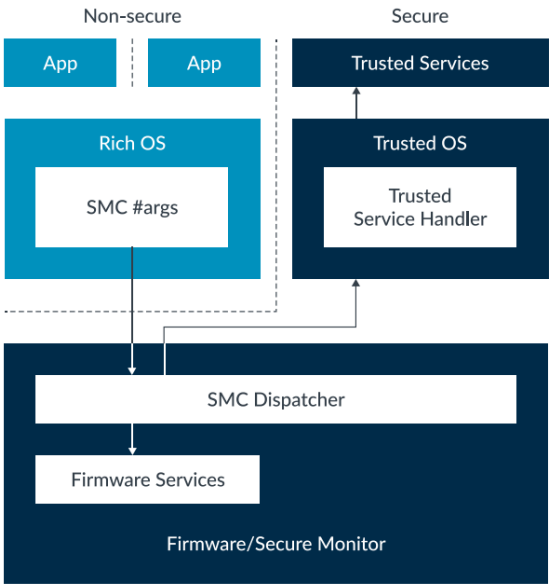
EL3是一个特殊情况。如前面在安全状态部分所述，当处于EL0/1/2 时，SCR_EL3中的NS位控制着处理器所处的安全状态。然而，无论SCR_EL3.NS位的值如何，EL3始终处于安全状态。因此当处于EL3时，SCR_EL3.NS位控制TLBI操作针对的安全状态。例如：

- SCR_EL3.NS==0: Affects Secure EL0/1 translation regime
- SCR_EL3.NS==1: Affects Non-secure EL0/1 translation regime

3.7. SMC exceptions

作为对两种安全状态支持的一部分，架构引入了SMC指令。执行SMC指令会引发一个SMC调用异常，该异常的目标异常级别为EL3。SMC 通常用于请求服务，这些服务可以来自驻留在 EL3的固件，也可以来自TEE的某个服务。SMC调用首先会被传递到 EL3，在那里，一个 SMC调度器会确定由哪个实体来处理该调用。如下图所示：

Figure 3-10: SMC dispatcher



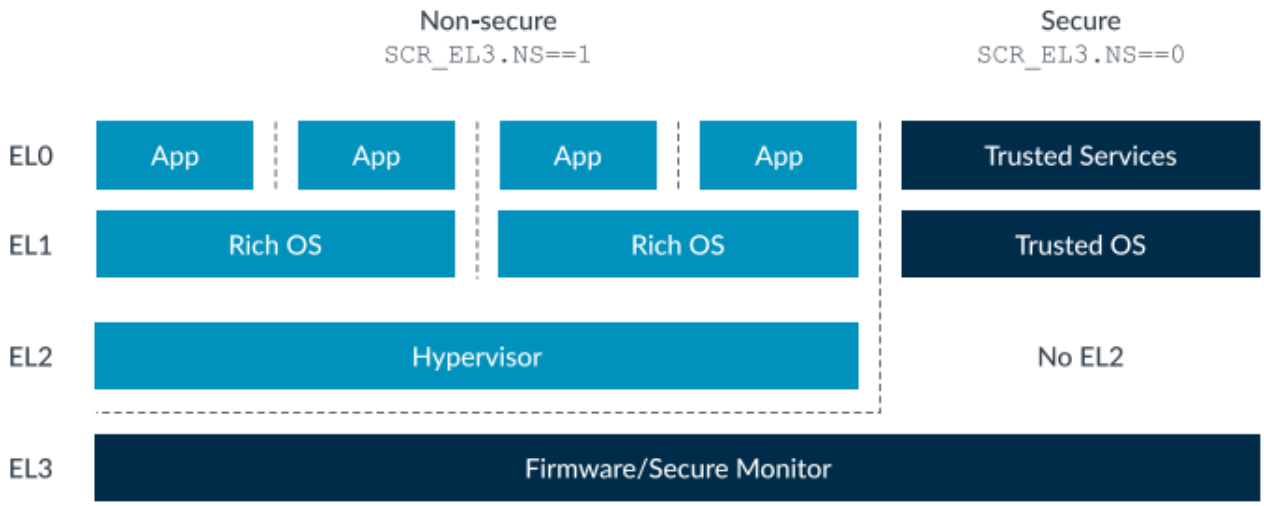
在EL1执行SMC指令时，可以将其trap到EL2处理。这对hypervisor很有用，因为虚拟机管理程序可能希望模拟虚拟机所看到的固件接口。

> SMC在EL0下不可用

3.8. Secure virtualization

当虚拟化技术首次在Armv7-A架构中引入时，它仅被应用于非安全状态。直到Armv8.3版本，Armv8架构也是如此，如下图所示：

Figure 3-11: Secure virtualization

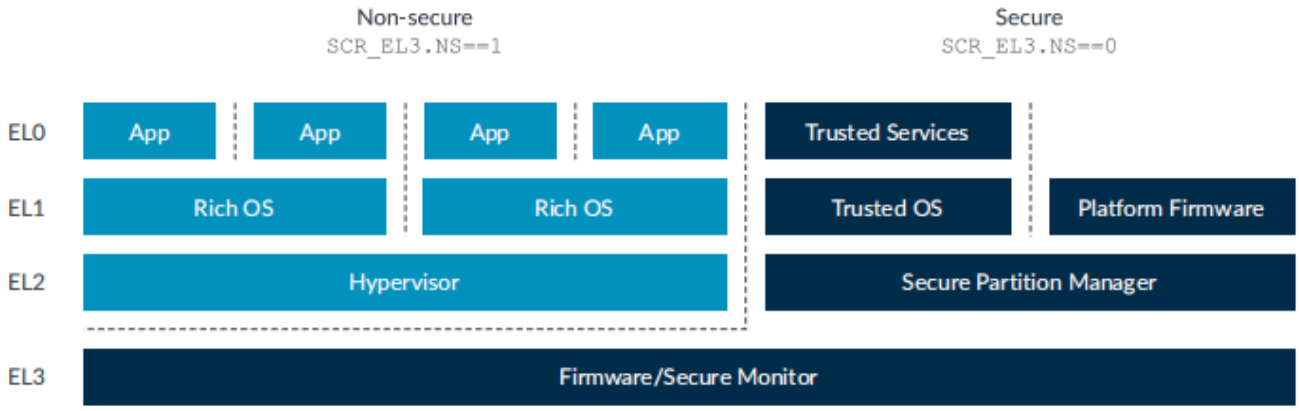


正如之前在“Switching between Security states”部分所描述的，EL3用于承载固件和安全监控器。Secure EL0/1承载可信执行环境（TEE），该环境由trusted service和kernel组成。 此前，人们认为在安全状态下没有必要支持多个虚拟机，这意味着无需为安全状态提供虚拟化支持。随着 TrustZone技术应用的增多，一些需求逐渐凸显出来：

- 某些可信服务与特定的可信内核相关联。为了让设备支持多种服务，可能需要运行多个可信内核。
- 遵循最小权限运行原则，需要将部分固件功能从 EL3中剥离出来。

解决方案是在 Armv8.4-A版本中引入对安全状态下EL2的支持，如本图所示：

Figure 3-12: Support for EL2 in Secure state

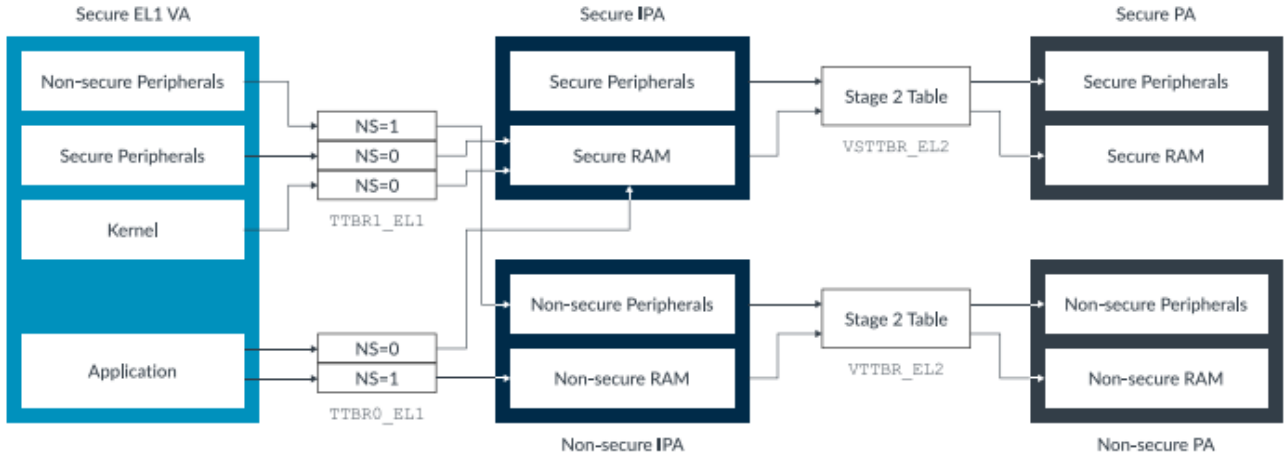


S.EL2通常承载的不是一个完整的虚拟机管理程序，而是一个安全分区管理器（SPM）。安全分区管理器允许创建隔离的分区，这些分区无法看到其他分区的资源。一个系统可以有多个包含可信内核及其可信服务的分区。还可以创建一个分区来存放平台固件，从而无需在EL3运行该代码。

- 打开secure EL2:
  - 当secure EL2被支持时，可以通过SCR.EL3.EEL2来打开或者关闭secure EL2.0是禁用，1是打开。
- 新增secure状态下的stage2页表转换机制

在安全状态下，虚拟机（VM）的stage1地址转换可以输出安全地址和非安全地址，并且由转换表描述符中的非安全（NS）位进行控制。这就产生了两个IPA空间，即安全空间和非安全空间，如你在下图中所见，每个空间都有其各自的stage2地址转换表集。

Figure 3-13: Stage 2 translation in Secure state



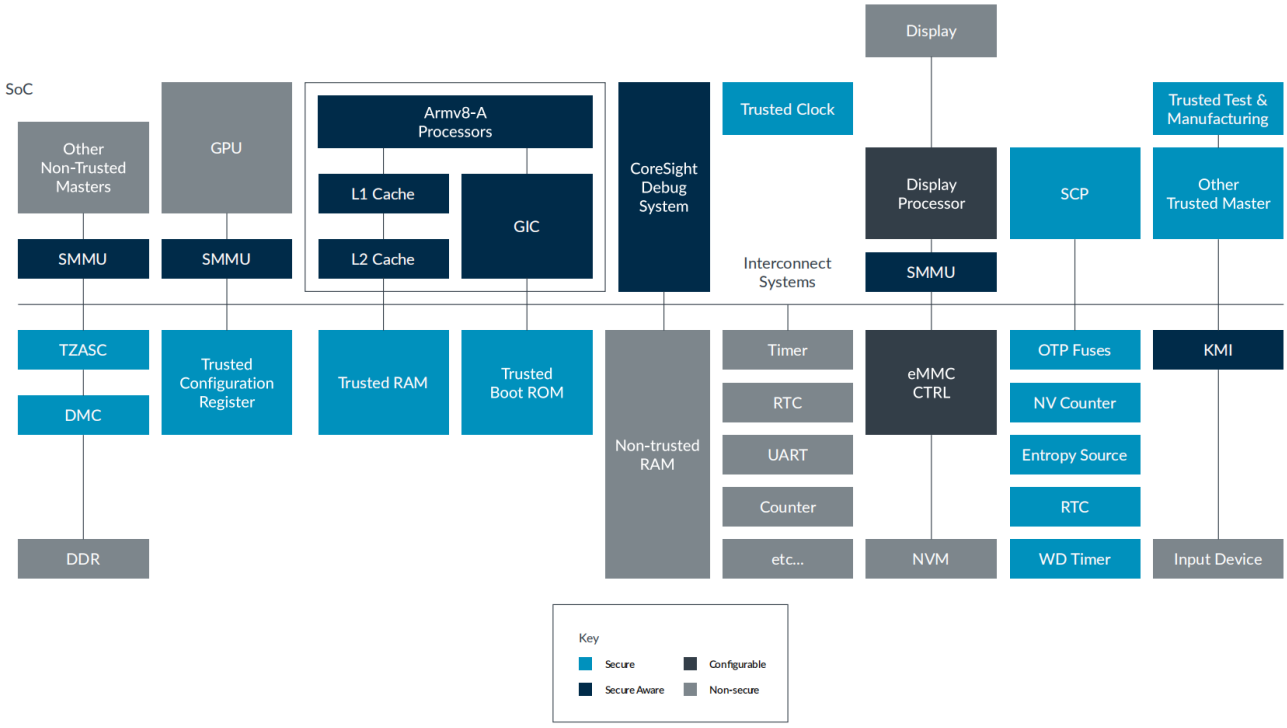
与stage1不同，stage2中没有NS位。对于给定的IPA,非安全的中间物理地址（IPA）会转换为非安全的物理地址（PA），而安全的中间物理地址（IPA）会转换为安全的物理地址（PA）。

4. System architecture

在本指南中，到目前为止我们一直专注于处理器，但TrustZone不只是处理器特性。为了充分利用TrustZone的特性，我们还需要系统其他部分的支持。 以下是一个启用了安全扩展区（TrustZone）的系统示

例：

Figure 4-1: System architecture



4.1. Completers: peripherals, and memories

在前面的"物理地址空间"部分，我们介绍了两种物理地址空间的概念，即安全物理地址空间和非安全物理地址空间。处理器将正在被访问的地址空间输出到内存系统。内存系统利用这一信息来实现隔离。

在本主题中，我们将提到bus Secure和bus Non-secure.bus Secure是指对安全物理地址空间的总线访问。bus Non-secure是指对非安全物理地址空间的总线访问。需要注意在安全状态下软件可以访问这两种物理地址空间，这意味着总线访问的安全性不一定与发起该访问的处理器的安全状态相同。

> 在AMBA AXI和ACE中，AxPROT[1]被用于指定访问哪个地址空间，0代表secure,1代表non-secure.

从理论上讲，一个系统可以拥有两个完全独立的内存系统，通过所访问的物理地址空间（AxPROT）在它们之间进行选择。但在实际中，这种情况不太可能出现，系统将物理地址空间当作一种属性来使用，以此控制对内存系统中不同设备的访问。

一般来说，我们讨论的存储器、外设以及总线等completer可以分为两种类型：

- Trustzone aware
    - 这类设备在构建时就考虑了TrustZone的相关特性，并且在内部会利用访问的安全性。
    - 一个例子是通用中断控制器（GIC）。无论是处于安全状态还是非安全状态的软件都可以访问通用中断控制器（GIC）。非安全访问只能看到非安全中断，而安全访问则可以看到所有中断。通用中断控制器（GIC）会利用总线事务的安全性来决定展示哪种视图。
- non-trustzone aware
    - 这代表了典型系统中的大多数completer。这类设备在内部不会利用总线访问的安全性。

- 一个例子是像timer这样的简单外设，或者on-chip memory。这样的设备要么是安全的，要么是非安全的，不会两者兼具。

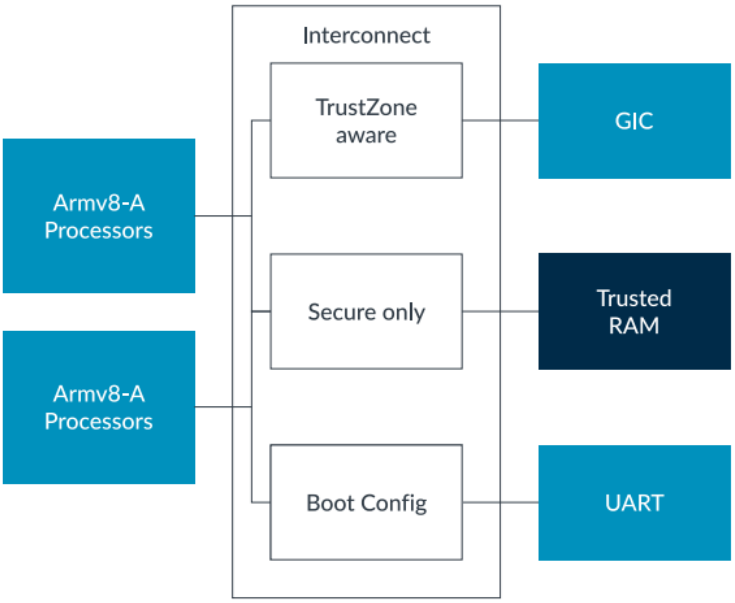
4.2. Enforcing isolation

TrustZone有时被称为completer-enforced protection system. 请求者会发出其访问的安全状态信号，然后由内存系统来决定是否允许该访问。那么基于内存系统的检查是如何进行的呢？

在大多数现代系统中，内存系统的检查是由互连结构来完成的。例如，Arm公司的NIC-400允许系统设计者为每个连接的completer指定：

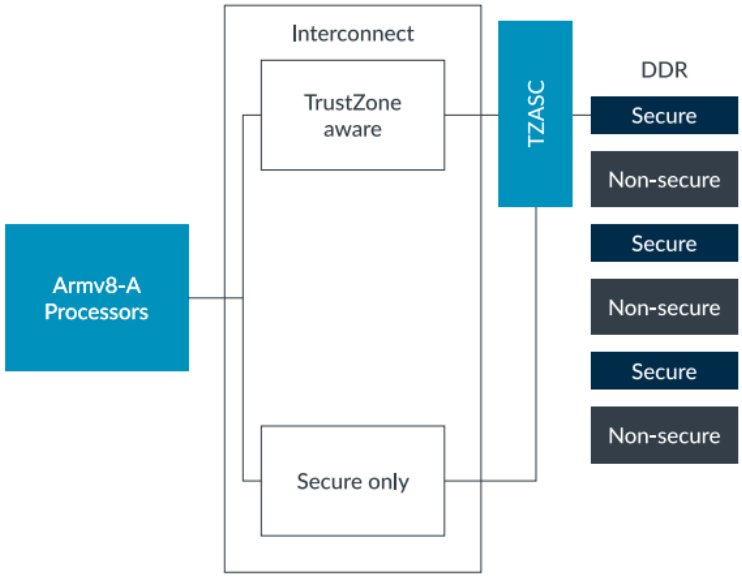
- secure
  - 只有Secure访问会被传递到device。对于所有Non-secure访问，互连结构会产生一个fault，并且不会将该访问提交给completer.
- non-secure
  - 只有Non-secure访问会被传递到device。对于所有Secure访问，互连结构会产生一个fault，并且不会将该访问提交给device.
- boot time configurable
  - 在启动时，系统初始化软件可以将device编程为Secure或Non-secure。默认设置为Secure
- trustzone aware
  - 互连结构允许所有访问通过。isolation由连接的device保证。

Figure 4-2: Implement isolation



Trustzone aware的方法适用于支持TrustZone的device，或者那些完全处于一个地址空间内的device。对于像片外DDR这样的大容量存储器，我们可能希望将其划分为Secure和Non-secure区域。A TrustZone Address Space Controller(TZASC)可以让我们做到这一点，如下图所示：

Figure 4-3: Partition memory



TZASC类似于Memory Protection Unit(MPU)，它允许将一个device的地址空间划分为多个区域，每个区域可指定为Secure或Non-secure。用于控制TZASC的寄存器仅支持Secure访问，这意味着只有Secure软件才能对内存进行分区。 TZASC的一个例子是Arm TZC-400，它最多支持九个区域。

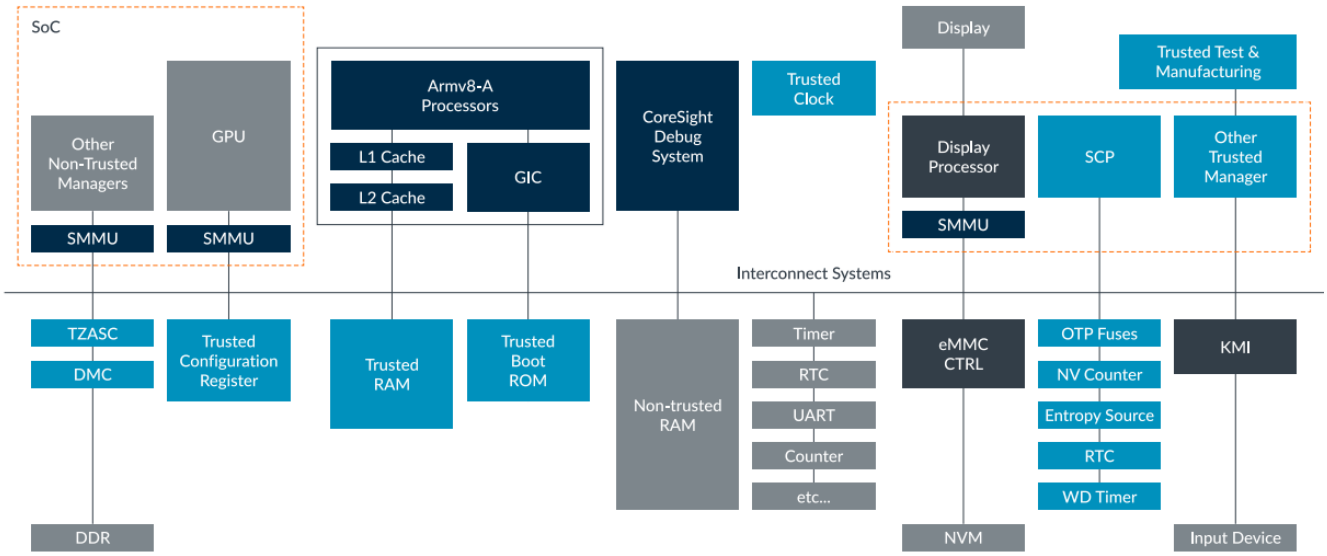
> 片外内存相较于片上内存更不安全，因为攻击者更容易读取或修改其内容。片上内存更安全，但成本更高且容量有限。我们必须在成本、可用性和安全性之间取得平衡。在决定哪些数据存放在片外内存以及哪些数据需要保存在片上时要谨慎。

当实现了Armv9-ARealm Management Extension(RME)时，内存可以通过Granule Protection Table在物理地址空间之间动态移动。

4.3. Bus requesters

接下来，我们将查看系统中的bus requester，如下图所示：

Figure 4-4: Bus requesters in the system



系统中的A-profile处理器支持TrustZone，并在每次总线访问时发送其安全状态。此外，大多数现代SoC中还包含非处理器的bus requester，例如，GPU和DMA控制器。

与completer device类似，我们可以大致将系统中的requester device分为以下几类：

- TrustZone aware
  - 一些requester是TrustZone aware的，并且和处理器一样，在每次总线访问时提供相应的安全信息。例如，按照Arm SMMUv3规范构建的SMMU.
- Non-TrustZone aware
  - 并非所有requester都具备TrustZone aware能力，特别是在复用旧有IP时。这类requester在进行总线访问时通常不提供安全信息，或者总是发送相同的值

Non-TrustZone aware的requester如何访问系统资源呢？我们可以选择以下几种方法之一：

- Design time tie-off
  - 当requester只需要访问单一物理地址空间时，系统设计人员可以通过固定相应ns信号来确定其可访问的地址空间。这种解决方案简单，但缺乏灵活性。
- Configurable logic
  - 提供指示secure信息的配置逻辑。如Arm NIC-400的一些互连结构提供了配置寄存器，Secure软件可以在启动时使用这些寄存器来设置所连接的requester访问的安全性。这会覆盖requester自身提供的值。这种方法仍然只允许requester访问单一物理地址空间，但比固定连接的方式更灵活。
- SMMU
  - 更灵活的选择是使用SMMU。对于受信任的requester，SMMU在Secure状态下的行为类似于内存管理单元（MMU）。这包括在转换表条目中包含NS位，以控制访问哪个物理地址空间。

4.4. M and R profile Arm processors

许多现代设计中会同时包含A系列、R系列和M系列处理器。例如，移动设备可能会使用A系列处理器来运行移动操作系统，使用R系列处理器用于蜂窝调制解调器，以及使用M系列处理器进行底层系统控制。

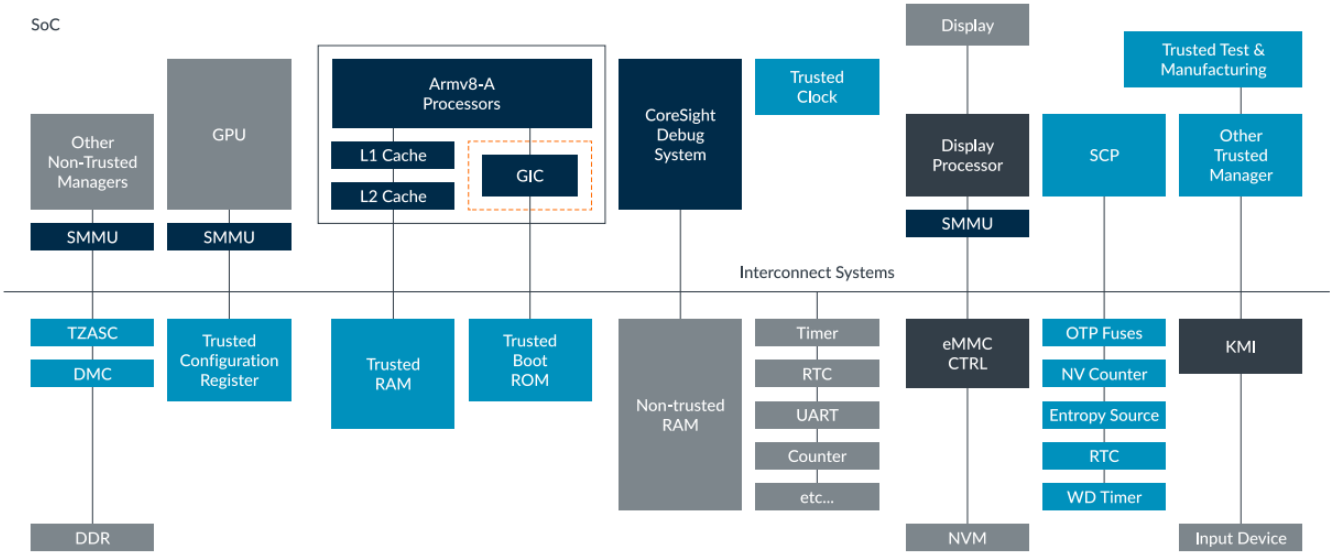
R系列处理器并不像A系列处理器那样支持两种安全状态。这意味着运行在这些处理器上的软件无法控制输出的物理地址空间。从这个角度来看，它们的行为与其他Non- TrustZone aware的总线请求者非常相似。对于未实现适用于Armv8 - M的TrustZone的M系列处理器，情况也是如此。

通常，这些处理器只需要访问单一物理地址空间。以我们的移动设备示例来说，这些处理器通常包含一个用于底层系统控制的M系列处理器。这有时被称为系统控制处理器（SCP）。在许多系统中，SCP是仅支持Secure的设备。这意味着它只具备发起总线Secure访问的能力。

4.5. Interrupts

接下来，我们将查看系统中的中断，如下图所示。

Figure 4-6: Interrupts in the system



Generic Interrupt Controller(GIC)支持TrustZone。每个中断源（在GIC规范中称为INTID）被分配到以下三个Group之一：

- Group 0：Secure中断，以FIQ形式发出信号
- Secure Group 1：Secure中断，以IRQ或FIQ形式发出信号
- Non-secure Group 1：Non-secure中断，以IRQ或FIQ形式发出信号

中断源分组是通过软件写GIC[D|R]_IGROUPR和GIC[D|R]_IGRPMODR寄存器来控制的，且只能在Secure状态下完成。这种分配不是静态的，软件可以在运行时更新。

对于配置为Secure的INTID，只有bus Secure访问才能修改其状态和配置。对于Non-secure的bus访问而言，与Secure中断对应的寄存器字段读取值为0。

对于配置为Non-secure的INTID， Secure和Non-secure的总线访问都可以修改其状态和配置。

为什么会有两个Secure Group(group0和secure group 1)？通常，Group0用于由EL3固件处理的中断。这些中断与底层系统管理功能相关。Secure Group1用于所有其他Secure中断源，通常由S.EL1或S.EL2软件处理。

4.6. Handling interrupts

处理器有两种中断：IRQ和FIQ。当中断变为pending状态时，GIC会根据中断所属的Group以及处理器当前的安全状态，使用不同的中断信号：

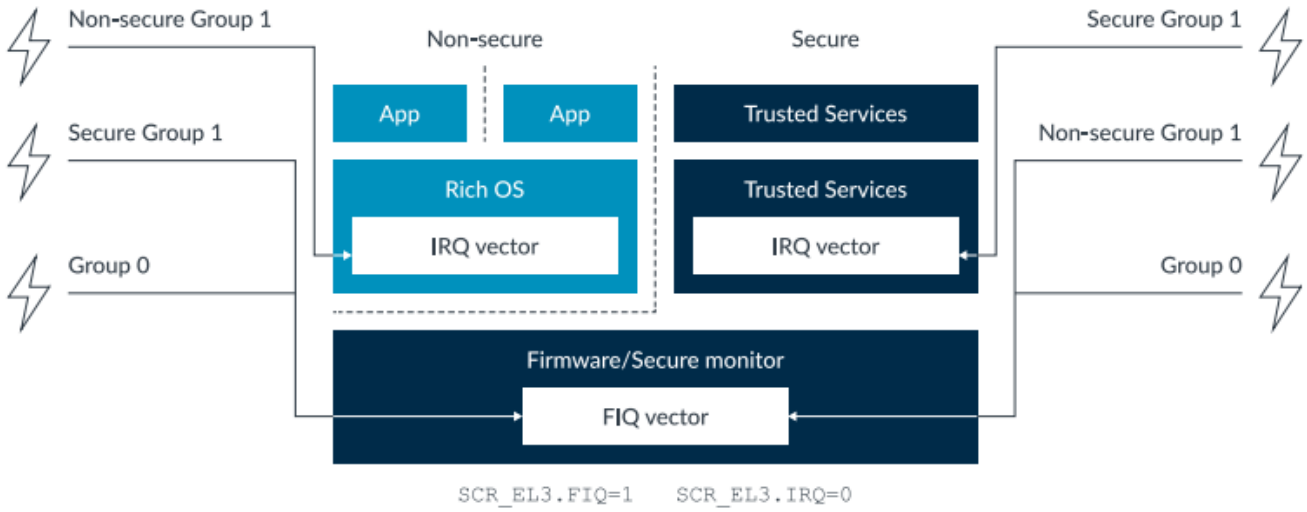
- Group 0中断
  - 始终以FIQ发出信号
- Secure Group 1
  - 处理器当前处于Secure状态 —— 以IRQ发出信号
  - 处理器当前处于Non-secure状态 —— 以FIQ发出信号
- Non-secure Group 1
  - 处理器当前处于Secure状态 —— 以FIQ发出信号
  - 处理器当前处于Non-secure状态 —— 以IRQ发出信号

Group 0中断通常用于EL3固件,这意味着：

- IRQ代表当前安全状态下的Group 1中断。
- FIQ意味着我们需要进入EL3，要么是为了切换安全状态，要么是让固件来处理该中断。

以下示例展示了如何配置异常路由控制：

Figure 4-7: Exception routing controls



前面的图展示了一种可能的配置。另一种常见的选择是，在处于Secure状态时，将FIQ路由到EL1。可信操作系统（Trusted OS）将FIQ视为一个请求，该请求要么是要让出控制权给固件，要么是要切换到Non - secure状态。这种中断路由方式使可信操作系统有机会以可控的方式退出。

4.7. Debug, trace and profiling

接下来，我们将查看系统中的debug和trace组件。

现代Arm系统具备丰富的功能来支持debug和trace。在引入TrustZone技术后，我们必须确保这些功能不会被用于破坏系统的安全性。

就debug功能而言，以一款SoC的开发为例。不同的开发人员被授权调试系统的不同部分。芯片公司的工程师有需求且被信任，能够调试系统的所有部分，包括Secure状态下的代码。因此，所有的debug功能都应被启用。

当芯片交付给原始设备制造商（OEM）时，他们仍需要调试Non-secure状态下的软件栈。然而，可能会限制OEM对Secure状态下代码进行debug。

在搭载该芯片的成品中，我们可能希望为应用开发者提供一些debug功能。但同时，我们也希望限制他们对芯片供应商和OEM代码进行debug的能力。

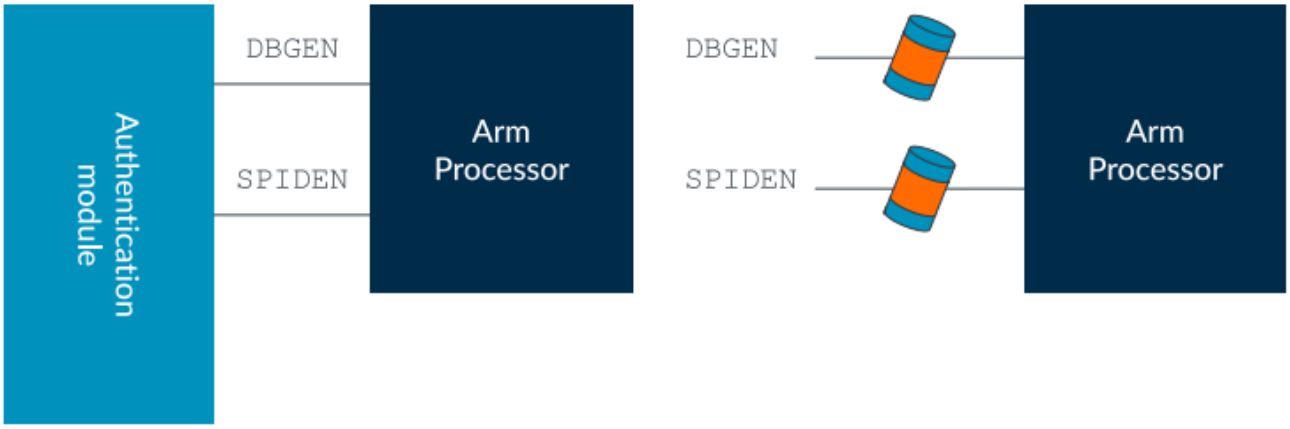
用于启用不同debug、trace和性能分析功能的信号有助于我们应对这种情况。这些信号分为控制Secure状态和Non - secure状态:

- DBGEN：Top-level invasive debug enable, controls external debug in both Security states
- SPIDEN： Secure Invasive Debug Enable, controls external ability to debug in Secure state

> 这两个信号只是示例。实际上还有其他的调试权限信号，请参考你所使用处理器的技术参考手册。

由于我们希望在开发的不同阶段使用不同的信号值，因此通常会使用电熔丝（e-fuses）或认证模块来连接这些信号。以下是一个示例：

Figure 4-9: Debug and trace components



在制造过程中熔断熔丝，就可以永久禁用外部调试功能。不过，使用熔丝确实会让现场调试变得更加困难。因为熔丝一旦熔断，就无法恢复。而认证模块则更具灵活性，Arm RSE（Runtime Security Engine）就是认证模块的一个例子。

4.8. Other devices

最后看一下系统中的其他设备。

我们这个支持TrustZone的示例系统包含了几个我们尚未提及，但构建一个实用系统又必不可少的设备。

- 一次性可编程内存（OTP）或熔丝
    - 这些是一旦写入数据就无法更改的内存。与每个芯片上都包含相同镜像的启动只读存储器（ROM）不同，OTP可以被编程写入设备独有的值，还可能写入OEM（原始设备制造商）独有的值。
    - OTP中存储的内容之一是设备独有的私钥。在每颗芯片制造时，会生成一个随机的唯一密钥并写入OTP。这个设备独有的私钥用于将数据与芯片绑定。
    - 设备独有的私钥的优势在于它能防止批量攻击。如果每颗芯片都使用相同的密钥，那么一旦有一台设备被攻破，所有类似的设备也会面临安全风险。
    - OTP还经常用于存储OEM公钥的哈希值。与其他类型的内存相比，OTP的成本相对较高。对于公钥而言，只存储其哈希值而非完整的密钥可以节省成本。
- Non-volatile counter
    - 非易失性（NV）计数器，它的实现方式可能类似于更多的熔丝。这是一种只能递增且永远无法重置的计数器。
    - NV计数器用于防范回滚攻击。假设某设备的固件版本3存在一个已知漏洞，而该设备当前运行的是已修复此漏洞的版本4。攻击者可能会尝试将固件降级回版本3，以利用这个已知漏洞。为了防范这种情况，每次固件更新时，计数器的值都会增加。在设备启动时，会将固件版本与NV计数器的值进行核对。如果两者不匹配，设备要么拒绝启动，要么进入恢复模式，要么以受限功能启动。
- Trusted RAM和Trusted ROM
    - 这些都是片上的仅支持安全访问的存储器。

- Trusted ROM是获取初始启动代码的地方。由于它位于片上，攻击者无法对其进行替换；又因为它是只读存储器，攻击者也无法对其重新编程。这意味着我们有一个已知且可信的执行起点，本指南的软件架构部分将对此进行讨论。
- Trusted RAM通常是一个容量为几百千字节的静态随机存取存储器（SRAM）。它是在安全状态下运行的软件的工作内存。同样，由于它位于片上，攻击者很难获取其中的内容。

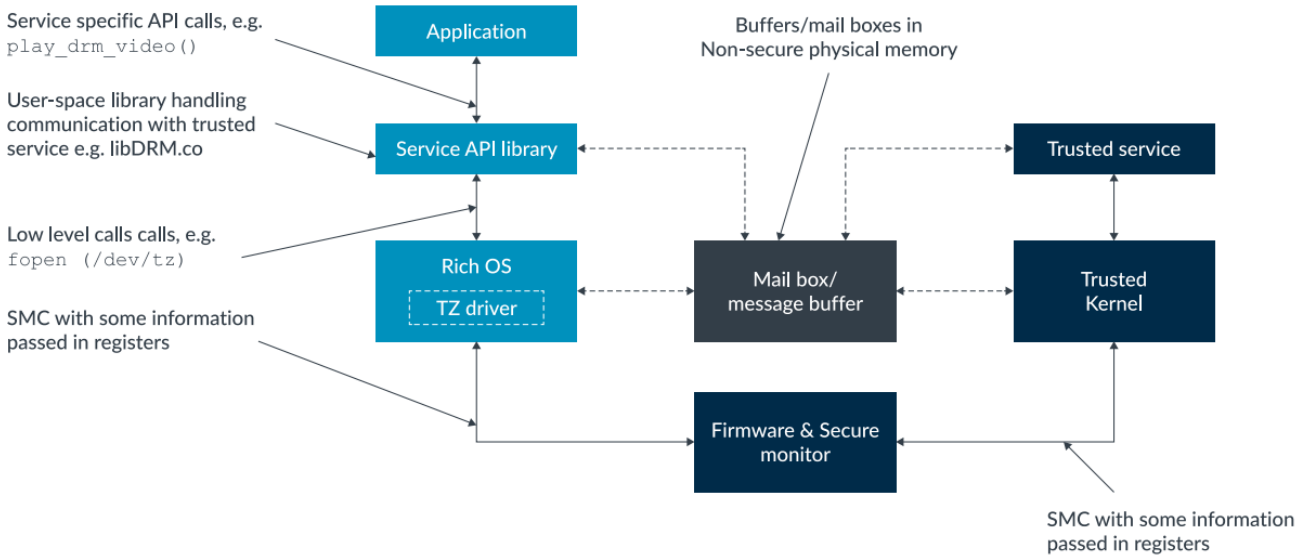
5. Software architecture

在处理器和系统架构层面的TrustZone相关内容中，我们探讨了硬件方面对TrustZone的支持，涵盖了Arm处理器以及更广泛的内存系统。接下来这个主题将聚焦于TrustZone系统中的软件架构。

5.1. Top-level software architecture

下面是一个典型的支持trustzone的软件栈：

Figure 5-1: Software stack for a TrustZone enabled system



> 出于简化，图中未包含hypervisor

处于安全状态（Secure state）的可信内核（Trusted kernel）承载着诸如密钥管理或数字版权管理（DRM）之类的服务。运行在非安全状态（Non-secure state）的软件需要以受控的方式访问这些服务。

用户空间应用程序不太能直接感知到TrustZone的存在。相反，它会使用用户空间库提供的高级应用程序编程接口（API）。该库负责处理与可信服务的通信。这类似于图形API为底层图形处理单元（GPU）提供抽象的方式。

服务库与可信服务之间的通信通常通过内存中的消息队列或邮箱来处理。“世界共享内存（World Shared Memory，WSM）”这一术语有时会用来描述用于这种通信的内存。这些队列必须位于双方软件都能访问到的内存中，也就是非安全内存。这是因为非安全状态下的软件只能访问非安全内存。

服务库会将一个或多个请求放入邮箱，然后调用内核空间中的驱动程序。该驱动程序负责与可信执行环境（TEE）进行底层交互，这可能包括为消息队列分配内存，并将这些队列注册到TEE中。要记住，安全和非安全这两个“世界”运行在不同的虚拟地址空间中，因此它们不能使用虚拟地址进行通信。

驱动程序通常会使用SMC来调用安全状态。控制权会通过EL3 secure monitor传递到TEE中的可信内核。内核会调用所请求的服务，该服务随后可以从队列中读取请求。

5.2. Trusting the message

在我们刚刚描述的流程中，请求会存放在位于非安全内存的队列里。那么如果出现以下情况该怎么办呢：

- 发起初始请求的应用程序是恶意的？
- 其他恶意软件替换了队列中的消息？

可信执行环境（TEE）必须假定从非安全状态提供的任何请求或数据都可能是恶意的。这意味着对请求或请求方进行身份验证的操作需要在安全状态下完成。具体的验证方式将取决于所提供的可信服务及其安全要求，并没有一个统一的答案。

5.3. Scheduling

在TrustZone系统中有两个软件栈，一个用于非安全状态，另一个用于安全状态。处理器核心一次只能处于一种状态。那么，由谁来决定每个“世界”（安全状态和非安全状态）何时可以运行呢？

像使用电源状态协调接口（PSCI）发起的电源管理请求这类对EL3固件的显式调用，通常是阻塞式的。这意味着只有在请求的操作完成后，控制权才会返回给非安全状态。不过，这类调用往往持续时间短且发生频率低。

可信执行环境（TEE）通常在非安全状态操作系统调度器的控制下运行。一种可行的设计是在操作系统下运行一个守护进程作为TEE的占位程序。当操作系统调度到这个守护进程时，守护进程会通过安全监控调用（SMC）将控制权交给TEE。然后TEE开始运行，处理未完成的请求，直到下一个调度周期或中断到来。之后，控制权会返回到非安全状态的操作系统。

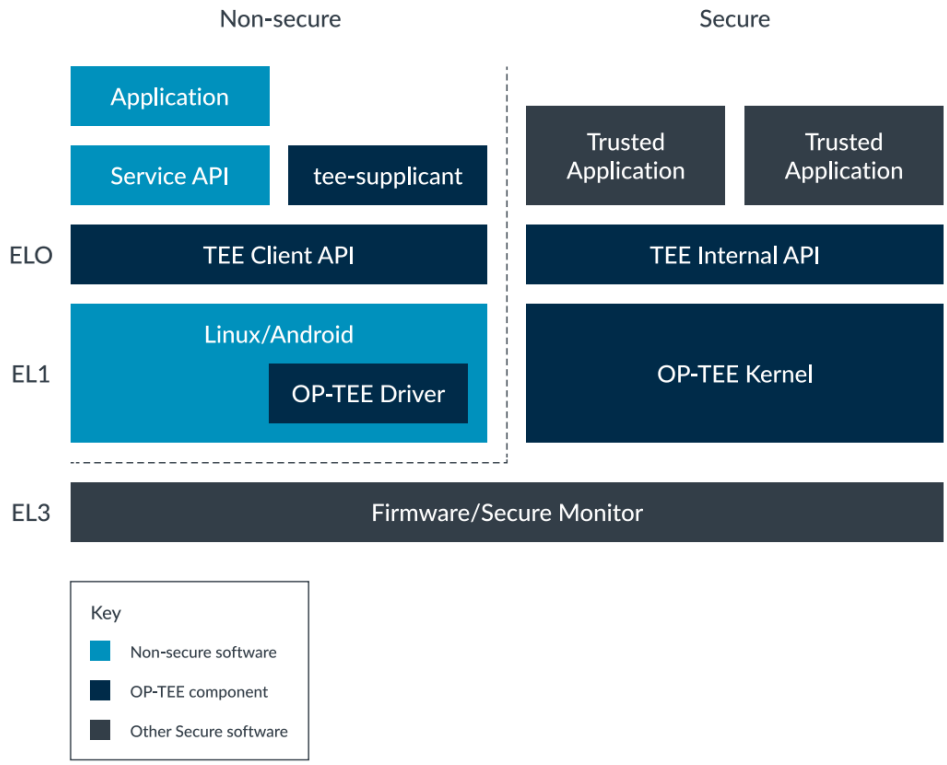
这看起来可能有些奇怪，因为这种方式让不可信的软件（非安全状态下的软件）来控制可信软件（安全状态下的TEE）何时执行，这可能会引发拒绝服务攻击。然而，由于TEE是为非安全状态提供服务的，阻止它运行只会让这些服务不可用。例如，攻击者可以阻止用户播放受数字版权管理（DRM）保护的视频。但这种攻击并不会导致任何信息泄露。这种设计能够确保数据的机密性，但无法保证服务的可用性。

我们可以对软件栈进行设计以同时保障可用性。通用中断控制器（GIC）允许将安全中断设置为比非安全中断更高的优先级，从而防止非安全状态的软件阻止安全中断的响应。

5.4. Example TZ SW: OP-TEE

市面上存在许多可信内核，既有商业版本，也有开源版本。其中一个例子是 OP-TEE，它最初由意法爱立信（ST - Ericsson）开发，如今已成为由利纳罗（Linaro）托管的开源项目。OP-TEE 提供了功能完备的可信执行环境，你可以在aOP-TEE项目网站上找到关于它的详细介绍。

Figure 5-2: OP-TEE structure



OP-TEE内核运行在S.EL1,在S.EL0托管可信应用程序。可信应用程序通过可信执行环境内部应用程序编程接口（TEE Internal API）与 OP- TEE 内核进行通信。

TEE内部API是由全球平台组织（GlobalPlatform）开发的标准API。全球平台组织致力于开发标准 API，这些 API 得到了许多不同的可信执行环境（TEE）的支持，而不仅仅是 OP-TEE。

在非安全状态下，内核空间中有一个底层的OP -TEE驱动程序。它负责处理与OP-TEE内核的底层通信。

在非安全用户空间EL0，有一个用户空间库实现了另一个全球平台组织（GlobalPlatform）定义的 API。TEE Client API是应用程序用来访问可信应用程序或服务的接口。在大多数情况下，我们并不期望应用程序直接使用 TEE Client API。相反，会有另一个特定服务的库来提供更高级别的接口。

OP-TEE 还包含一个名为 tee-supplicant 的组件。tee-supplicant 负责处理 OP-TEE 所支持且需要一定程度与丰富操作系统进行交互的服务。例如，代表可信执行环境存储安全数据。

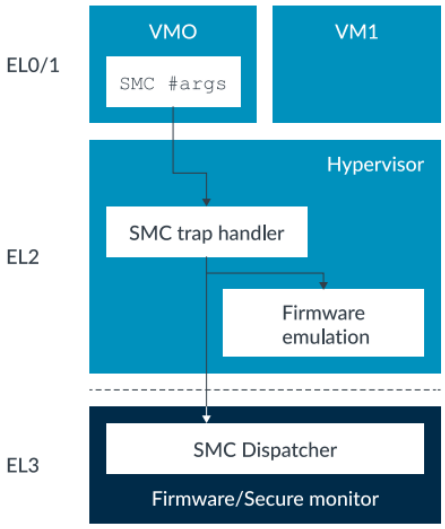
5.5. Interacting with Non-secure virtualization

在我们目前所涉及的示例中，我们忽略了非安全状态下可能存在的管理程序（hypervisor）。当存在管理程序时，虚拟机（VM）与安全状态之间的大部分通信都将通过管理程序进行。

例如，在虚拟化环境中，SMC用于访问固件功能和可信服务。固件功能包括诸如电源管理之类的操作，管理程序通常不希望允许虚拟机直接访问这些操作。

管理程序可以捕获来自EL1的SMC，这使得管理程序能够检查请求是针对固件服务还是可信服务。如果请求是针对固件服务，管理程序可以模拟相应接口，而不是直接传递调用。管理程序可以将可信服务请求转发到EL3。你可以在下面的图表中看到这一过程：

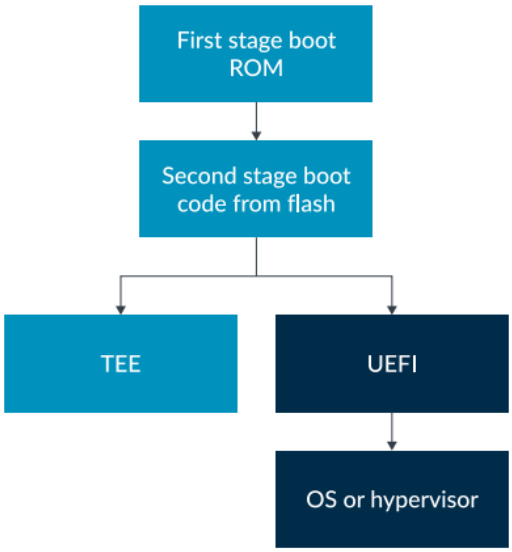
Figure 5-3: Interacting with Non-secure virtualization diagram



5.6. Boot and the chain of trust

启动过程是TrustZone系统的关键部分。只有当我们信任启动流程中在某个软件组件之前运行的所有软件组件时，才能信任该组件。这通常被称为信任链。下面的图表展示了一个简化的信任链：

Figure 5-4: Simplified chain of trust



在我们的示例中，最先运行的代码存于启动只读存储器（boot ROM）中。我们必须默认信任启动 ROM，因为在启动流程中没有更早的阶段来验证其内容。由于代码存储在 ROM 中，这就保护了初始启动代码不被重写。将初始启动代码置于芯片上可防止其被替换，所以我们可以默认信任它。启动 ROM 代码通常体量小且逻辑简单，其主要功能是从闪存中加载并验证第二阶段的启动代码，然后将其载入安全静态随机存取存储器（Secure SRAM）。

第二阶段的启动代码会对平台进行系统初始化，例如为片外动态随机存取存储器（DRAM）配置内存控制器。这段代码还负责加载并验证将在安全状态和非安全状态下运行的镜像。比如，在安全状态下加载可信执行环境（TEE），在非安全状态下加载诸如统一可扩展固件接口（UEFI）这类更高级别的固件。

早些时候我们介绍了系统控制处理器（SCP）。SCP是一种微控制器，在许多SoC中执行底层系统控制。如果存在SCP或类似的组件，它也会成为信任链的一部分。

5.7. Boot failures

在可信启动系统中，每个组件在加载下一个组件之前都会对其进行验证，从而形成一条信任链。现在让我们来看看当验证失败时会发生什么。

对于这种情况并没有一个统一的答案。这取决于系统的安全需求以及验证失败发生在启动过程的哪个阶段。以移动设备中的片上系统（SoC）为例。如果验证在以下阶段失败：

- 第二阶段启动镜像
  - 第二阶段启动镜像用于片上系统（SoC）和处理器的初始化。如果在此阶段验证失败，我们可能无法确定设备能否安全启动并正常运行。因此，如果此阶段的验证失败，通常是致命的，设备将无法启动。
- 可信执行环境（TEE）
  - TEE 提供诸如密钥管理之类的服务。即使没有 TEE，设备仍可能继续运行，尽管功能可能会受到一定限制。因此，我们可以选择不加载TEE，但仍允许非安全状态的软件加载。
- 非安全状态固件或全功能操作系统镜像
  - 非安全状态的软件本身的可信级别就较低。我们可以选择允许其启动，但阻止其访问通过TEE提供的高级功能。

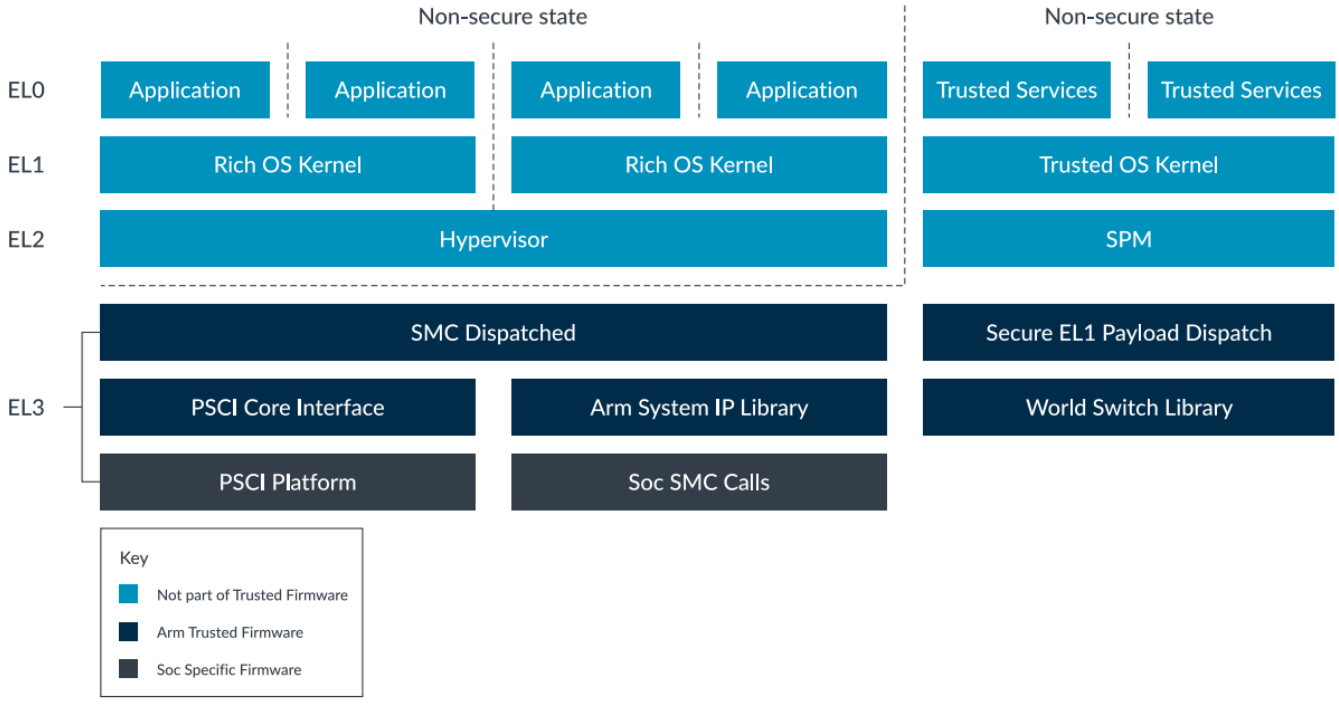
这些仅仅是例子，不同的系统中有不同的处理方式。

5.8. Trusted Board Boot Requirements

早些时候我们介绍了Trusted Base System Architecture（TBSA），它是为系统设计师提供的指导。而e Trusted Board Boot Requirements（TBBR）则是一套类似的、面向软件开发人员的准则。TBBR为在支持TrustZone的系统中构建可信启动流程提供了指引。

5.9. Trusted Firmware

可信固件（Trusted Firmware）是针对 Armv8-A 架构设备的安全世界软件的开源参考实现。它为片上系统（SoC）开发者和原始设备制造商（OEM）提供了一个符合相关 Arm 规范（包括可信主板启动要求（TBBR）和安全监控调用约定（SMCC））的参考可信代码库。 Trusted Firmware结构图如下：



SMC dispatcher负责处理传入的SMC请求。SMC dispatcher会识别哪些SMC请求应由可信固件在EL3进行处理，哪些SMC请求应转发给TEE。

可信固件提供了用于处理Arm系统IP（如互连组件）的代码。芯片供应商则需要提供处理自定义或第三方IP的代码，这其中就包括特定于片上系统（SoC）的电源管理代码。

6. Example use cases

在处理器与系统架构中的 TrustZone 部分，我们介绍了TrustZone在硬件方面的特性，之后又探讨了利用这些特性的典型软件栈。在这个主题中，让我们整合这些知识，来看看一些实际的应用案例。

6.1. Encrypted filesystem

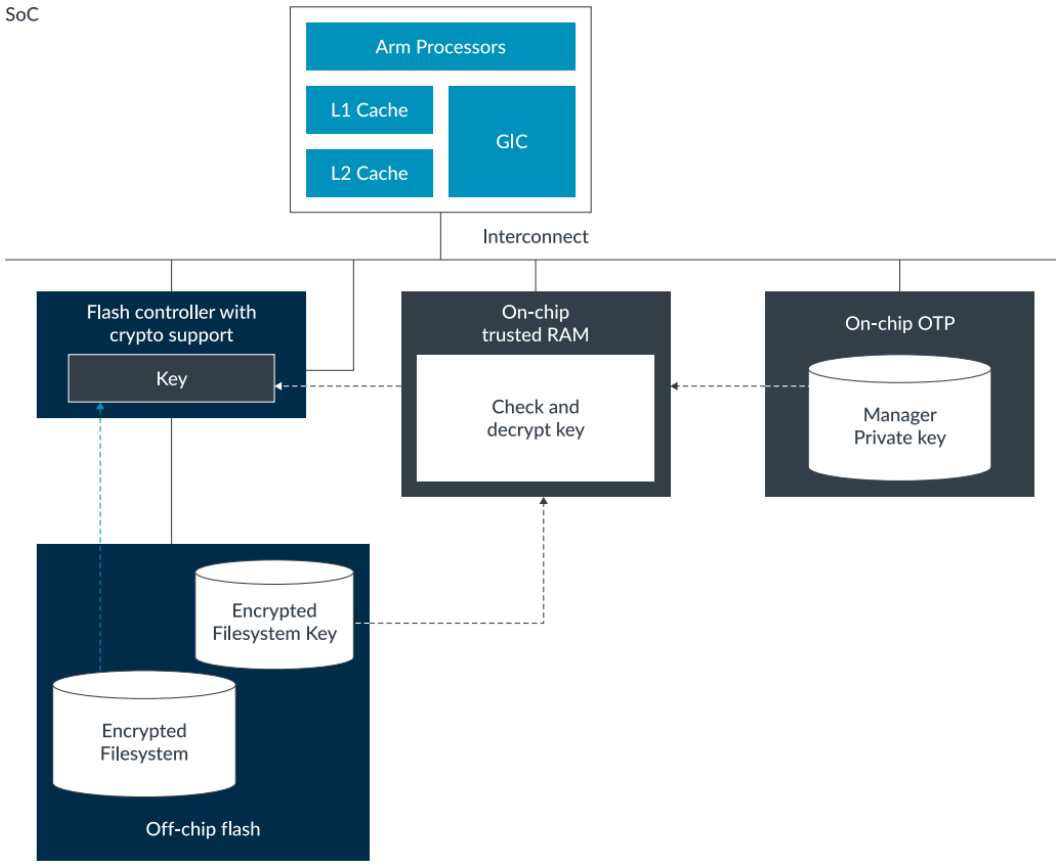
智能手机等移动设备存储着大量个人数据。用户非常关心在设备丢失或被盗时这些数据的保密性，这就是为什么大多数新款设备都支持文件系统加密。TrustZone 可以作为保护这些数据的解决方案的一部分。

存储在外部闪存中的数据是经过加密的。在设备启动时，会对用户进行身份验证，然后提供用于解密文件系统的密钥。解密操作可能由加速器处理，也可能集成在闪存控制器中完成。

文件系统的密钥本身也需要保护其保密性。如果密钥泄露，攻击者就可以对文件系统进行解密。

认证之后的流程在下图中展示：

Figure 6-1: Processes after authentication



在安全状态下：

- 身份验证完成后，已加密的文件系统密钥会被读取到片上安全内存中。然后，使用存储在片上的请求设备唯一密钥对该密钥进行解密并检查。

- 文件系统密钥会被配置到加密引擎或内存控制器中一个仅能安全访问的寄存器里。
- 后续对闪存中文件系统的总线访问操作，都将使用已配置的密钥进行加密或解密。

通过在安全状态下执行这些操作，TrustZone使我们能够确保文件系统密钥永远不会暴露给非安全状态的软件。这意味着非安全状态下的恶意代码无法提取这些密钥用于日后的攻击。 为什么文件系统密钥要存储在片外呢？

与片外闪存相比，片上内存的容量往往有限，而且成本较高。将文件系统密钥存储在片外可以降低成本。对其进行加密意味着我们能够确保密钥的保密性。存在一种风险，即恶意软件可能会损坏密钥，这将破坏数据的完整性，但不会导致数据泄露。

为什么在这个例子中我们使用一个单独的文件系统密钥，而不是请求设备唯一的私钥呢？

从理论上讲，我们可以使用设备唯一密钥。但这意味着我们永远无法更改该密钥，因为请求设备唯一的私钥存储在一次性可编程（OTP）存储器中。例如，如果我们出售手机，这可能会成为一个问题。相反，我们生成一个新的随机文件系统密钥。如果您想要格式化或重置设备，我们可以删除该文件系统密钥并生成一个新的密钥。任何使用旧密钥加密的数据现在都无法恢复。

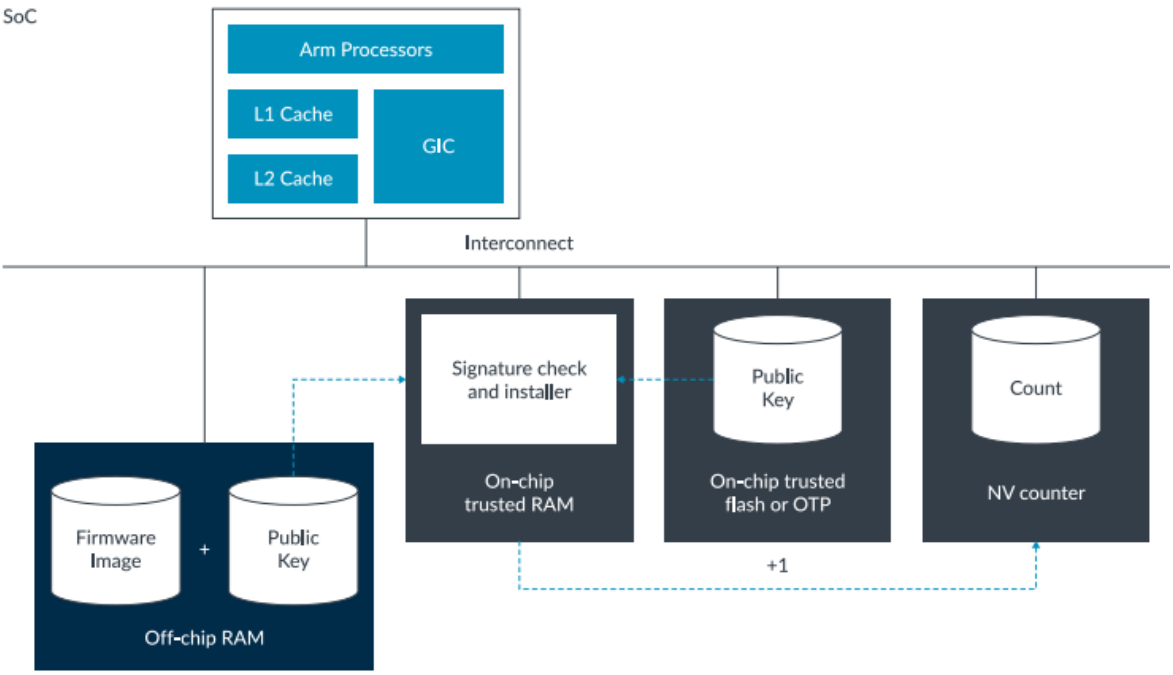
6.2. Over the air firmware update

第二个例子与启动固件的更新有关。我们的系统有如下要求：

- 新的固件镜像通过网络提供
- 只有经过验证的（真实的）镜像才能被安装
- 固件版本不能回滚

为了实现这些目标，原始设备制造商（OEM）会用其私钥对镜像进行签名。下载该镜像的设备会预先配置好公钥，设备可以用公钥来验证签名。当固件更新时，一个非易失性计数器会增加计数，这样就能检测到是否存在固件版本回滚的情况。该系统如下图所示：

Figure 6-2: Processes after authentication



镜像的下载是在非安全状态下进行的。镜像本身并非机密信息，所以我们无需保护其保密性。下载后的镜像会被放置在内存中，然后会向安全状态发出安装该镜像的请求。

安全状态下的软件负责进行身份验证。它会使用原始设备制造商（OEM）的公钥来完成这一操作，该公钥通常存储在片外闪存中。这个公钥并非机密信息，所以我们无需确保其保密性。但我们确实需要确保该公钥的真实性，并检测是否存在替换公钥的企图。我们通过在片上保存该公钥的哈希值来实现这一点，在需要时可以用这个哈希值来检查公钥。哈希值所需的位数较少，而且片上内存成本较高。

当公钥被加载并检查无误后，就可以检查新的固件镜像了。我们要确保它是真实有效的（即签名匹配），并且它是比已安装固件更新的版本。

假设这些检查都通过了，就会安装该镜像，并增加非易失性（NV）计数器的值。增加NV计数器的值意味着，如果攻击者试图安装较旧版本的固件，设备将能够检测到这种企图。

在这个例子中，TrustZone使我们能够确保用于验证固件镜像的密钥得到保护，并且固件镜像无法被回滚。