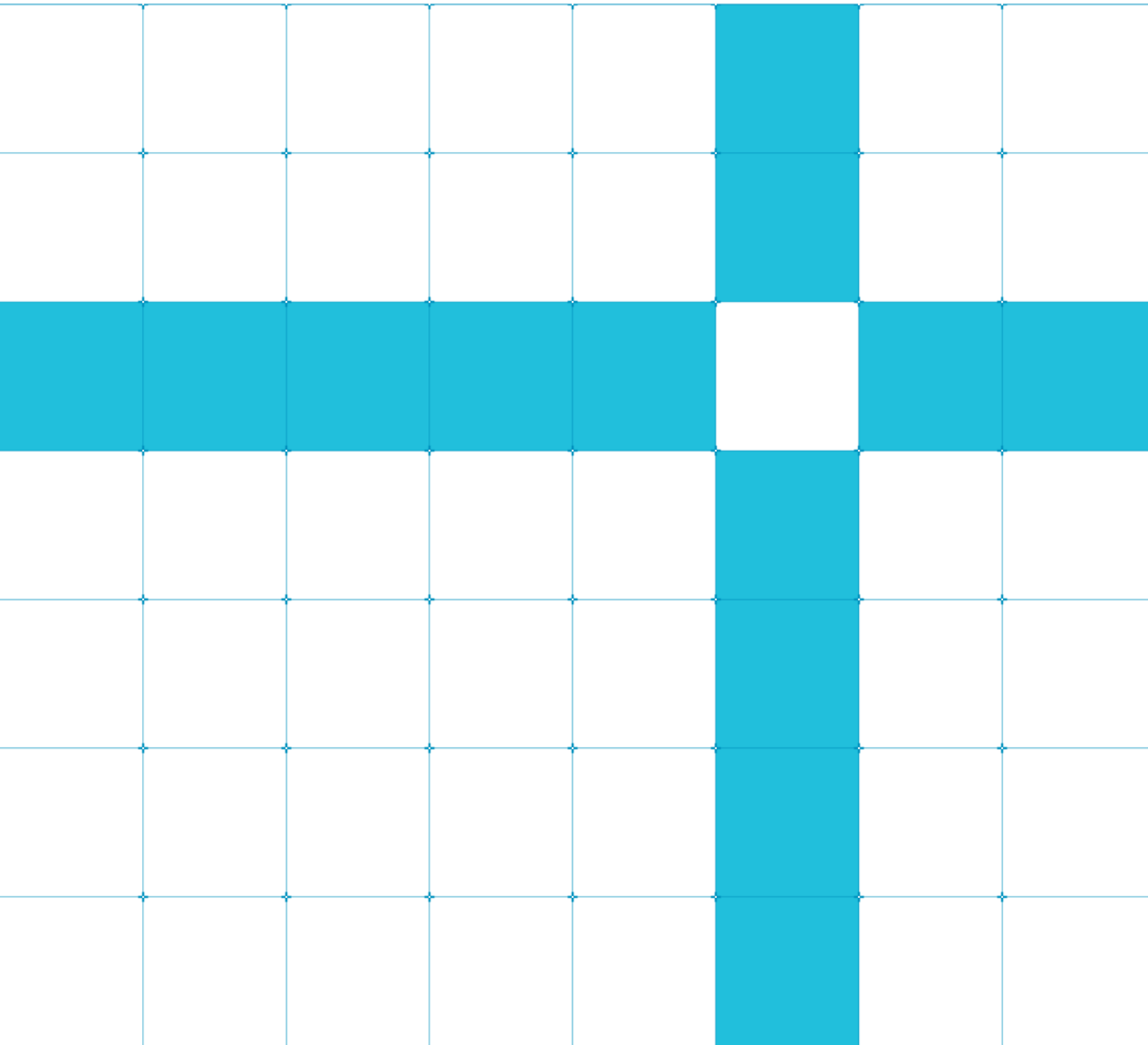
Isolation using virtualization in the Secure world

Secure world software architecture on Armv8.4

Version 1.0



Non-Confidential Proprietary Notice

LES-PRE-20349

Contents

1 Introduction

Armv8.4 architecture adds virtualization support in the Secure world. This whitepaper describes why this feature has been introduced, and the improvements in security that it provides. The paper introduces a software architecture to leverage these benefits. Secure EL2, and this software architecture provide the basis for an A-Profile processor version of the Platform Security Architecture (PSA) Firmware Framework.

2 Background

A brief history of Arm TrustZone™

Arm TrustZone was introduced to the Arm architecture A-profile in 2003. At the heart of the TrustZone approach is the concept of Secure and Normal worlds that are hardware separated. Secure hardware resources are only accessible by the software running in the Secure world. Software in the Normal world is blocked by the hardware from accessing these resources. This concept of Secure (trusted) and Normal (non-trusted) worlds extends beyond the processor to encompass memory, software, bus transactions, interrupts, and peripherals within an SoC.

The introduction of TrustZone paved the way for an ecosystem of trusted operating systems (OS) vendors. Initially, trusted OSs provided basic device security services, such as trusted boot, or handling of platform secrets. Today trusted OSs have evolved to support bespoke applications that might be used in a number of security use cases, such as secure payment or media protection.

Originally, trusted OSs occupied all processor modes associated with TrustZone, and included code devoted to switching between the Normal and Secure worlds, as well as code for providing secure services. In systems based on Armv7, and prior architecture revisions, the switching code runs in monitor mode, whereas services code runs in secure supervisor and secure user modes. In Armv8 monitor mode evolved into an exception level, EL3, with its own memory translation regime and interrupt vectors. This allowed for cleaner separation of code, making it easier to provide separate binary images for switching and for services.

The advent of EL3, provided new opportunities to standardize key platform management functions. Until then, these were implemented in Normal world software, leading to fragmentation, and solutions that had to be repeated for each OS and each individual platform. The addition of an exception level made it possible to provide standard firmware solutions for these management functions, removing the fragmentation and generalizing code in Normal world operating systems. The most prevalent example of this service is power management. These standardization efforts gave rise to some key specifications:

SMC Calling Convention (SMCCC) This document provides the standard methodology to be followed when requesting a service from the secure world. It describes how registers are to be used for arguments and return values, as well how the Secure Monitor Call (SMC) instruction is to be invoked.

Power State Coordination Interface (PSCI) A standard for system and processor power management, covering a number of use cases including boot, hotplug, idle, and system shutdown and reset.

Software Delegated Exceptions Interface (SDEI) Primarily used in enterprise systems, this specification provides a software equivalent to non-maskable interrupts.

This standardization momentum didn't stop at specifications, it also included the Trusted Firmware open source project, formerly known as the Arm Trusted Firmware project. Trusted Firmware implements the specifications above and provides a reference for trusted boot. Today, this project provides the Secure world firmware basis for most Arm vendors and supports a number of popular trusted OSs. In addition to the specifications above, Trusted Firmware can be used to provide other services, such as errata management, or silicon vendor specific services.

Figure 1 shows the kinds of services offered by the Secure world in today's systems, based on the Armv8-A architecture. The figure also shows the software agents that provide these services and the broad partitioning by exception level. A trusted OS provides security services. This typically spans Secure EL1 and Secure EL0. Services are often provided through trusted applications that run in Secure EL0. In order to support these services, the trusted OS will have drivers for trusted hardware resources, such as secure crypto accelerators, or secure storage devices. EL3, in addition to hosting

code to transition between the Normal and Secure worlds, provides platform services, such as core power management through PSCI. This code is provided by the silicon vendor platform firmware, and it is often based on code from the Trusted Firmware open source project.

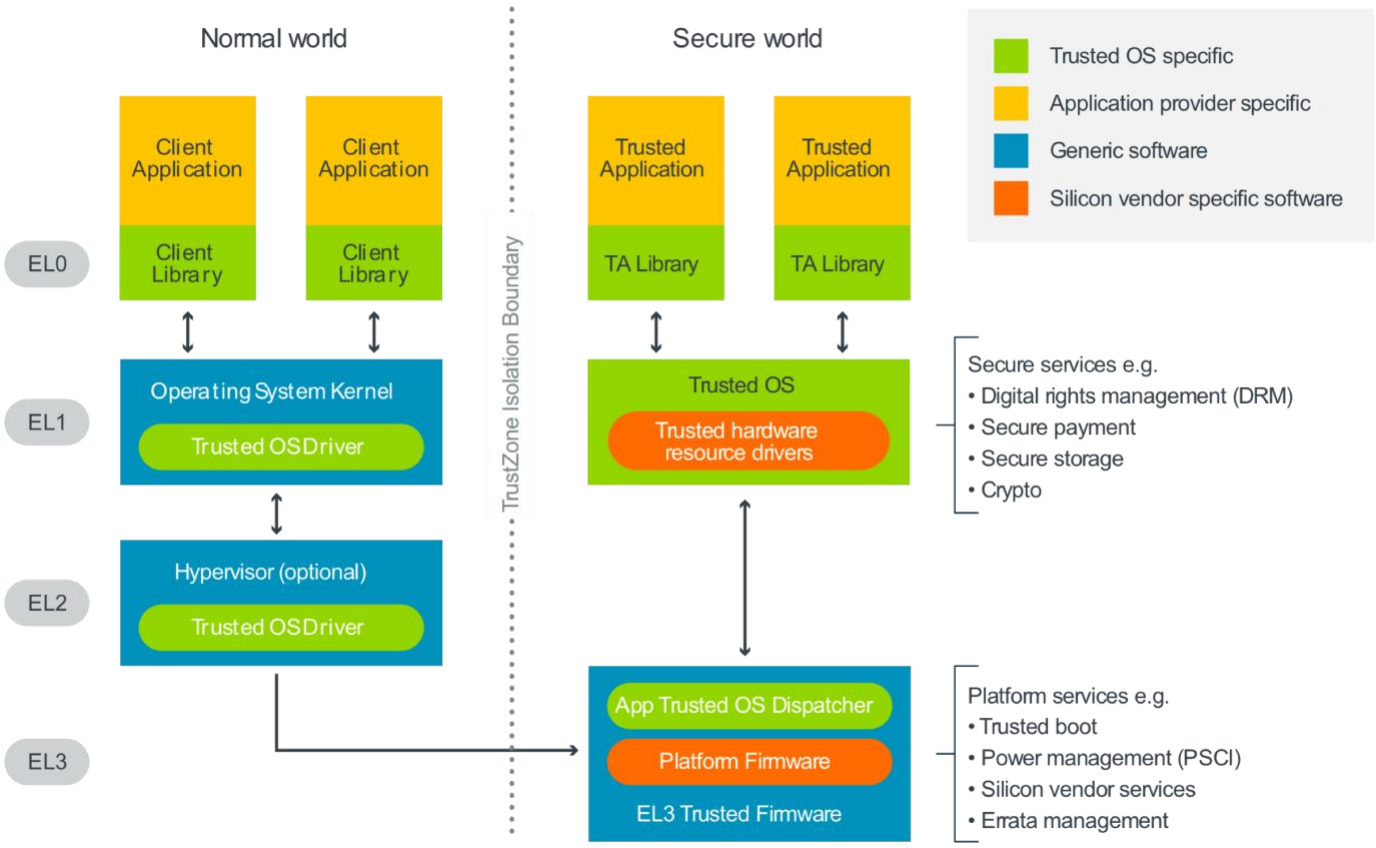


Figure 1 Typical services and partitioning of software agents in the Secure world

3 Today's challenges in the Secure world

The success of TrustZone has also created some architectural challenges, described in the following sections.

3.1. Trusted application ecosystem challenges

The success of TrustZone has spawned a number of trusted OSs from different vendors. Although these provide similar services, and there are some standards around trusted OS APIs, trusted applications are trusted OS specific. This is illustrated in Figure 1, which shows how both the Normal world client applications, and the trusted applications require linkage to trusted OS vendor specific libraries. This limits the scope of portability of applications from one trusted OS to another. The cost of porting means that applications tend to get tied to specific trusted OSs.

This presents a problem for OEMs that need to ship a rich set of applications, with different applications tied to different trusted OSs. While there are methods to host multiple trusted OS in Secure EL1, this requires trusted OS modification, and collaboration between different trusted OS vendors.

3.2. Integration of code from multiple vendors in the Secure world

A trusted OS needs to access trusted hardware resources to provide its services. In some cases, this is provided through silicon vendor drivers, which need to be integrated into the trusted OS. In other cases, the silicon vendor provides its own trusted OS, again leading the OEM to a situation where they might need to integrate more than one trusted OS. Both approaches require the OEM to integrate sources from separate vendors.

As described above, in addition to security services, platform services will be provided by platform firmware in EL3. The services require silicon vendor specific code to perform functions, such as power management. Running alongside these services is dispatcher code, which is involved in dispatching SMC calls to one or more trusted OSs, as well as silicon vendor firmware. The dispatching code includes trusted OS specific code, as well as generic code, which often originates from the Trusted Firmware project.

Figure 1 illustrates an example software stack that arises from the considerations above. As can be seen, there are a number of vendors involved. The key Secure world software components and vendors that make up this picture are:

- Trusted OSs to support security services to applications. Provided by one or more third party trusted OS vendors.
- A trusted OS, or driver model, to support trusted hardware resources on the platform, such as secure storage or a secure element. Provided by the silicon vendor.
- EL3 firmware to provide platform services, such as PSCI. Provided by the silicon vendor.
- EL3 firmware to provide switching between Normal world and appropriate Secure world payloads, which can have sources from various vendors.

3.3. The principle of least privilege

The *principle of least privilege* requires that a component must be able to access only the information and resources that are necessary for its correct operation. This contrasts with the current approaches to Secure world software, as depicted in Figure 1. EL3 and secure EL1 are privileged exception levels, both have the same level of access to the system address map and physical interrupts, on Armv8.3 or earlier. This means that it is not possible to isolate components running in EL3 firmware, from those running in secure EL1. In particular:

Firmware in EL3 cannot be isolated from a trusted OS. Firmware running in EL3 cannot constrain the visibility of the secure and non-secure system address maps from a trusted OS in Secure EL1.

EL3 firmware components cannot be isolated from each other. As depicted in **Figure 1,** generic firmware components, silicon vendor or third-party trusted OS vendor components, can merge in EL3. All arise from different vendors, and yet cannot be isolated from each other, as they have the same visibility of the physical address map, and of physical interrupts.

Normal world cannot be protected from trusted OSs in secure EL1. Secure EL1 has unconstrained access to the whole physical address map, both secure and non-secure. Unconstrained access to the system address map has been used in privilege escalation attacks on some trusted OS implementations.

Overall, these constraints make it impossible to enforce the principle of least privilege to each of the software components. This increases the complexity of auditing and certification, as all of the software entities need to implicitly trust each other, and therefore, cannot be audited separately. This approach also makes it hard to isolate trusted hardware resources to a particular software entity. A System MMU (SMMU) can alleviate this problem, however it does not eliminate it, as the software component programming the SMMU cannot be isolated from other Secure world software components.

3.4. Required solution

Solving the challenges listed above requires an architecture that can facilitate componentization and hardware isolation. The solution requires extending from the security separation provided by TrustZone, which provides just isolation between two worlds, to an architecture that can provide isolation between multiple, mutually-mistrusting software images. In particular, the solution requires:

- A method to provide hardware isolation between software components. This removes the need for mutual trust and allows components to be audited separately from each other.
    - Isolation requires restricting access to physical address space, registers, for processors and Direct Memory Access (DMA)-capable trusted peripherals.
- To enable the ecosystem of vendors to work together, the architecture will require standard APIs for common boundaries.
    - Standard APIs will enable the various software images to communicate and allow generalization and componentization of code. This enables removing trusted OS vendor specific code from EL3 and non-secure EL2.

4 Virtualization in Secure world

The Armv8.4 architecture introduces the Secure EL2 extension, adding support for virtualization in the Secure world. This brings the features that are available for virtualization in the non-secure state, to the secure state.

A key feature in virtualization support is the addition of a hypervisor-controlled second stage of translation. This allows a hypervisor to control which areas of physical memory are available to a virtual machine. Figure 2 illustrates the basic idea. A guest operating system controls the first stage of translation, which translates between the *virtual address space* (VA) and the *intermediate physical address space* (IPA). A second stage of translation, controlled by the hypervisor, translates between IPA and *physical address space* (PA). This enables a hypervisor to control which portions of the physical address space are visible to a given virtual machine.

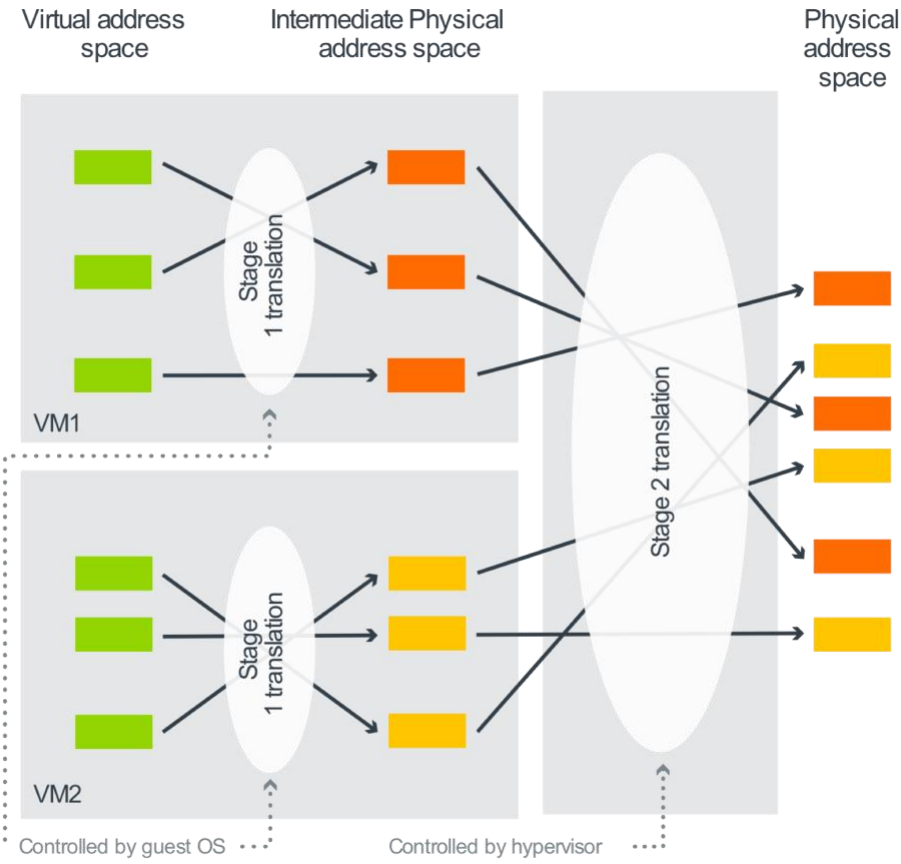


Figure 2 Stage 2 translation

Secure EL2 adds stage 2 translations to the secure state of processors. Further, the Arm System MMU architecture 3.2 and higher supports stage 2 translations in the secure state for other IO masters. Together, these features bring virtualization support to the Secure world and provide a hardware basis that meets a number of the architectural requirements described above. Secure virtualization enables:

1. The isolation of EL3 software from Secure EL1 software.
2. The isolation of Normal world software from Secure EL1 software.
3. Isolation of distinct Secure EL1 software components from each other.

The architecture extensions provided by Secure EL2 enable the isolation of Secure world software from different vendors. This architecture also enables isolating software stacks that have different purposes.

Traditionally, trusted OSs can access any location in the system address map. As the trusted OS is a privileged entity, a weakness in its implementation could be exploited to access any memory address in the system, secure or non-secure. Indeed, privilege escalation attacks of this nature have been reported on some trusted OS implementations. Secure EL2 prevents this escalation by restricting the memory regions accessible by software resident in Secure EL1 or Secure EL0. This is enabled by the use of stage 2 translations. Control of stage 2 translations in the secure state of a System MMU, extends this protection to include DMA-capable trusted hardware resources.

5 Secure partitions

Secure EL2 and SMMU provide the hardware isolation needed to meet the challenges described above. This section describes a software architecture that leverages these features to meet these challenges. This architecture also establishes standard APIs that enable collaboration between vendors of Secure world software.

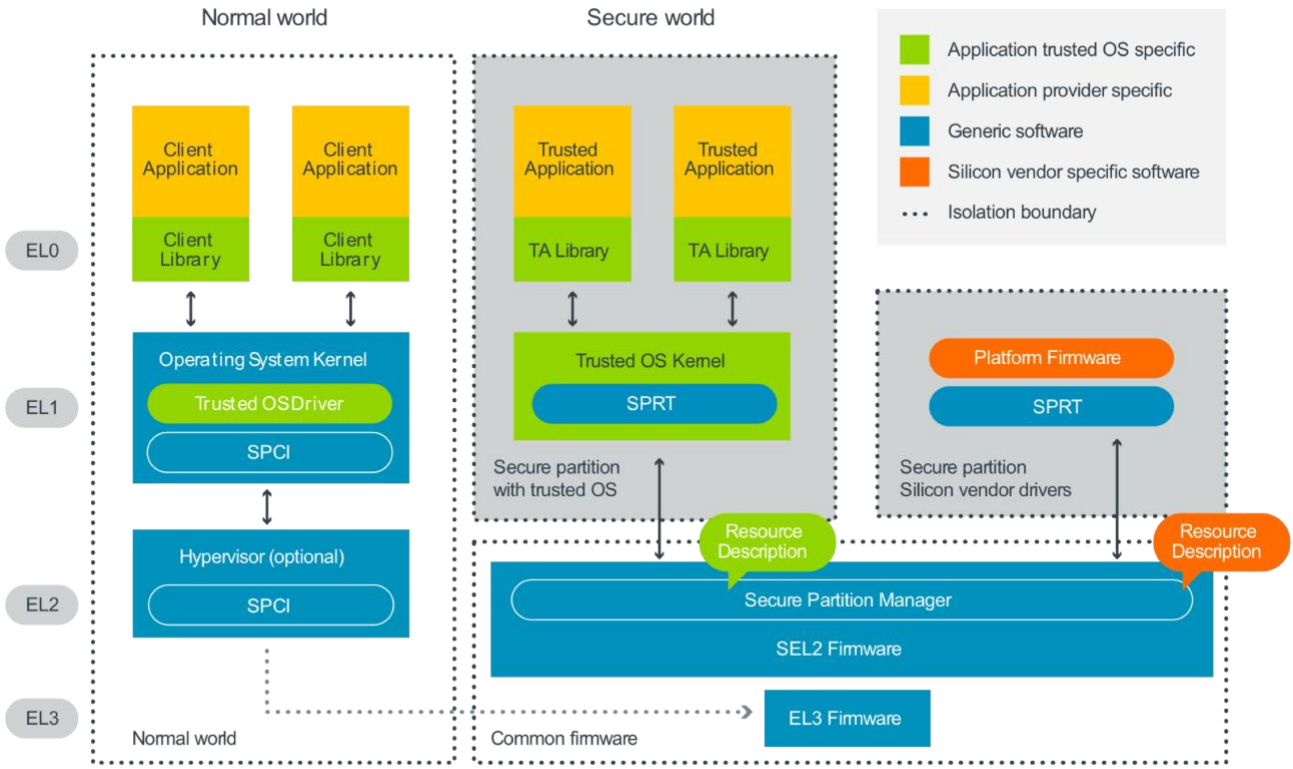


Figure 3 Software stack with secure partitions

Figure 3 describes the basic elements of a software architecture for the Secure world, in a system that includes Secure EL2. The basic elements are:

Secure partitions in Secure EL1. A secure partition is a sand-boxed environment, which has an isolated address space. This space cannot be directly accessed by other secure partitions. With Secure EL2, a secure partition is implemented through virtualization support and is essentially a Secure world virtual machine. The secure partition can be used to host a trusted OS, or may contain a driver stack for a trusted hardware resource. **Figure 3** depicts one secure partition hosting a trusted OS, and another hosting silicon-vendor-provided platform drivers.

Secure Partition Manager. Generic firmware in EL3 and Secure EL2 is responsible for managing these secure partitions. A key component here is the *Secure Partition Manager* (SPM), which is responsible for directing requests to the correct secure partition, and for policing accesses by secure partitions. The SPM can be thought of as a minimal partitioning hypervisor.

Figure 3 illustrates a software architecture in which the platform-specific firmware component, provided by the silicon vendor, is isolated from the firmware in Secure EL2 and EL3, as well as the trusted OS. This is achieved by placing the trusted OS and the platform firmware into separate secure partitions. To enable this architecture, the SPM needs to know which resources are required for each partition. This allows it to put in place the appropriate memory map restrictions, and to dispatch partitions when messages or interrupts are sent to them. This can be achieved through the use of a resource description manifest for each partition, which primarily describes:

1. The system resources it needs, memory, ownership of interrupts, and access to regions in the physical system address map.
2. A list of secure services that it implements through a combination of the requested resources and specified attributes.

At boot time, the SPM consumes the resource description to:

1. Validate and allocate resources requested by a secure partition.
2. Perform architectural and system setup, required by a secure partition e.g. maintenance of the vGIC and stage 2 translations.
3. Initialize the secure partition once its execution environment has been established.

In order to keep EL3 and Secure EL2 code generic, this software architecture will require standard APIs at the boundaries between the various software agents. APIs are needed between clients of secure services in the Normal world, and providers of services in secure partitions. The desired features for such an interface are:

1. Enable message passing between a client in the Normal world and a service in a secure partition.
2. Enable discovery of a secure service in a partition.
3. Enable memory sharing between a client and a secure partition.
4. A generic interface that requires no trusted OS specific drivers in the Normal world hypervisor and the SPM, and by consequence EL3 and Secure EL2.

Arm's proposal for this interface is described in the Secure Partition Client Interface (SPCI). Arm welcomes collaboration and feedback on this proposal.

In addition, standard APIs are required between secure partitions and the Secure Partition Manager. This provides the basis for secure partition runtime environment. The basic requirements for such an API:

1. Enable message passing between secure partitions.
2. Enable reception of messages and interrupts in secure partitions.
3. Enable initialization of secure partitions.

Together with resource management, message passing between secure partitions provides a means to create services for Secure world clients, and a way to securely share trusted hardware resources. Arm proposes the Secure Partition Run Time (SPRT), as the basis for secure partition runtime APIs. This is a specification that describes the run time model that a secure partition can rely on to implement secure services. It specifies the mandatory information that must be included in the resource description of a secure partition, as well as the run time model and interface to the SPM. The SPRT and SPCI can be thought of as the firmware framework of the Platform Security Architecture in A-profile processors. Both specifications cover approaches for systems with and without Secure EL2 (S-EL2). This enables software migration to these specifications, prior to the availability of hardware with S-EL2.

6 Migration of the secure ecosystem to Secure EL2

The software architecture described above, brings about changes and impacts to vendors for Secure world software. These are discussed below.

6.1. Impacts on trusted operating systems

Virtualizing trusted OSs changes the way they interact with physical resources on the platform. The SPM constrains access to the system address map and physical interrupts.

In systems without Secure EL2, secure physical interrupts are typically routed to Secure EL1. With Secure EL2, the interrupts are managed by secure firmware in a manner similar to how a hypervisor in the Normal world manages non-secure physical interrupts. These are first directed to the hypervisor, which then delegates their counterpart virtual interrupts to a guest OS VM. To enable this approach in the proposed Secure EL2 architecture, a system integrator needs to specify, for a given trusted OS in a secure partition, which secure interrupts it will handle. This information needs to be consumed by the Secure Partition Manager so that it can route the corresponding virtual interrupts to the appropriate secure partition. Trusted OS interrupt handling code should not be significantly affected by this. The trusted OS will run against a virtual GIC (vGIC), which is software transparent, and receive interrupts at the interrupt vector.

Much as in the interrupt case, a system integrator will have to specify which regions or memory can be seen by a secure partition. This information needs to be consumed by the SPM so that it can put in place the appropriate stage 2 page tables in the processor, and if needed, in System MMUs.

Placing a trusted OS in a secure partition, will require some changes to enable usage of the new standard APIs.

6.2. Impacts on Normal world software

Hypervisors can be used in mobile devices to provide additional security. When this is the case, a hypervisor can require trusted OS specific modifications, to allow a client operating system to send messages to a trusted OS.

For the architecture proposed in Figure 3, Arm is proposing a generalized message passing interface, the Secure Partition Client Interface (SPCI). This will allow removal of trusted OS specific drivers from EL3 and Secure EL2 firmware. It will lead to a generic implementation of the SPM, provided by the Trusted Firmware open source project. Migrating to the SPCI will require changing trusted OS drivers in client kernels to move to the SPCI, as well as providing a generic driver in Normal world hypervisors.

7 Conclusion

This whitepaper describes the rationale behind the introduction of the virtualization extension in the Secure world in Armv8.4 architecture. In addition, it proposes an approach to extend the existing Secure world software architecture to exploit the benefits of secure virtualization. These benefits include improved security isolation, and generalization of Normal and Secure world code. A more detailed description of this architecture can be found in the SPCI and SPRT specifications. These specifications, alongside the introduction of Secure EL2 in the architecture, provide the basis for a Platform Security Architecture (PSA) Firmware Framework for A-Profile processors.

8 Glossary

Term	Meaning
ABI	Application Binary Interface
GIC	Generic Interrupt Controller
IOMMU	Input/Output Memory Management Unit
IPA	Intermediate Physical Address
OS	Operating System
PA	Physical Address
PSCI	Power State Coordination Interface
SMMU	System Memory Management Unit
SPCI	Secure Partition Client Interface
SPM	Secure Partition Manager
SPRT	Secure Partition Run Time
TEE	Trusted Execution Environment
TOS	Trusted Operating System
VA	Virtual Address
VM	Virtual Machine