Security in an ARMv8 System

Version 1.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
17 March 2017	0100	Non-Confidential	First release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means "ARM or any of its subsidiaries as appropriate".

Confidentiality Status

This document is Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents

1 Security in an ARMv8-A system

A secure or trusted system is one that protects assets, for example passwords or credit card details from a range of plausible attacks, to prevent them from being copied or damaged, or made unavailable.

Security is defined by the principles of:

Confidentiality Preventing unauthorized access to the asset. Confidentiality is a key security concern. There are several methods of preventing unauthorized access to the asset. For example, passwords and cryptographic keys.

Integrity Preventing unauthorized changes to the asset using methods such as public keys.

Availability. Ensuring that the asset comes from a trusted source and detecting unauthorized changes using firmware updates.

Defense against modification and proof of authenticity is vital for security software and on-chip secrets that are used for security. Examples of trusted systems might include password protected mobile payments, digital rights management, and e-ticketing. Security is harder to achieve in the world of open systems, you might download a wide range of software onto a platform, while inadvertently also downloading malicious or untrusted code, which can tamper with your system.

Mobile devices can be used to view videos, listen to music, play games, or for browsing the Web and accessing financial services. This requires both the user and the bank or service provider to trust the device. The device runs a complex OS with high levels of connectivity and might be vulnerable to attack by malicious software. A virus or malware that is accidentally downloaded onto the device must never be allowed access to financial services information held securely. You can achieve some measure of security through software system design, but you can obtain higher levels of protection through core and system level memory partitioning.

Software and hardware attacks can be classified into the following categories:

Software attacks Attacks by malicious software typically do not require physical access to the device and can exploit vulnerabilities in the operating system or an application.

Simple hardware attacks These are passive, mostly non-destructive attacks that require access to the device and exposure to the electronics, and use commonly available tools such as logic probes and JTAG run-control units.

Laboratory hardware attack This kind of attack requires sophisticated and expensive tools, such as Focused Ion Beam (FIB) techniques or power analysis techniques, and is more commonly used against smartcard devices.

TrustZone technology is designed to protect against software attacks. Good design practice with the TrustZone Extension can also give a good defense against simple hardware attacks.

2 The TrustZone hardware architecture

The TrustZone architecture provides a means for system designers to help secure systems, using the TrustZone Security Extensions, and Secure peripherals. Low-level programmers should understand the design requirements that are placed on the system by the TrustZone architecture, even if they do not use the security features.

The ARM Security Extensions model allows system developers to partition device hardware and software resources, so that they exist in either the Secure world for the security subsystem, or the Normal world for everything else. Correct system design can ensure that no Secure world assets can be accessed from the Normal world. A Secure design places all sensitive resources in the Secure world, and ideally has robust software running that can protect assets against a wide range of possible software attacks.

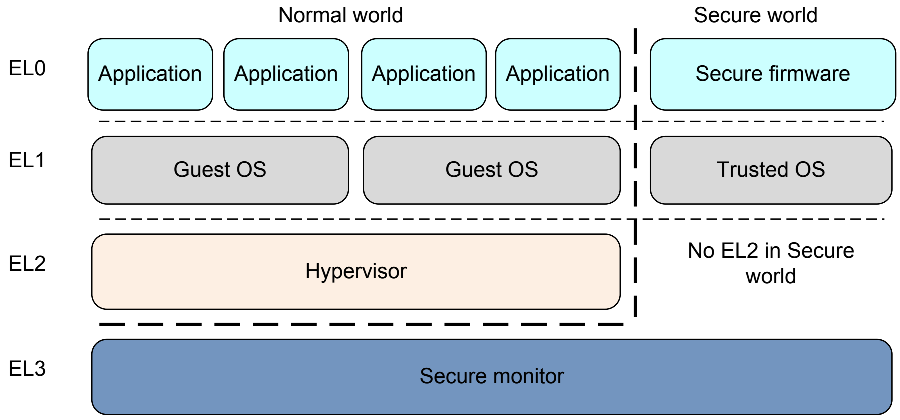

Figure 1 Security model for AArch64

The ARM Architecture Reference Manual uses the terms Secure and Non-secure to refer to system security states. A Non-secure state does not automatically mean security vulnerability, but rather normal operation and is therefore the same as the Normal world. Typically, there is a master and slave relationship between Non-secure and Secure worlds. Code in the Secure world is only executed when the OS permits Secure world execution through a mechanism that is initiated by the Secure Monitor Call (SMC) instruction.

Note

The use of the word world is used to describe not just the Execution state, but also all memory and peripherals that are accessible in that state. Non-secure memory and functions are also accessible to the Secure world.

The role of the Secure monitor is to provide a gatekeeper which manages the switches between the Secure and Non-secure worlds. In most designs its functionality is similar to a traditional operating system context switch, ensuring that state of the world that the core is leaving is safely saved, and the state of the world the processor is switching to is correctly restored.

The additions to the architecture mean that a single physical core can execute code from both the Normal world and the Secure world, each world can yield to or call the other, although this depends on the availability of interrupt-generating peripherals that can be configured to be accessible only by the Secure world. For example, a Secure timer interrupt could be used to guarantee some execution time for the Secure world. Such peripherals might be available, depending on the level of security and use cases that the platform designer intends to support.

The Secure monitor is a security critical component, as it provides the interface between the two worlds. For robustness reasons, that the monitor code should execute with interrupts disabled. Writing a reentrant monitor would add complexity and is unlikely to provide significant benefits over a simpler design.

Alternatively, an execution model closer to cooperative multitasking can be used. Although the Secure world is independent of the Normal world in terms of the resources each world can access, each world can yield to the other to enable code execution.

Like firmware or any other piece of system software, software in the Secure world must minimize its impact on other parts of the system. For example, consumption of significant execution time should usually be avoided unless performing some action requested by and expected by the Normal world. Non-secure interrupts should be passed to the Normal world as quickly as possible. This helps to ensure good performance and responsiveness of Normal world software without the need for extensive porting.

The memory system is divided by an extra bit that accompanies the physical address of peripherals and memory. This bit, called the NS-bit, indicates whether the access is Secure or Non-secure. This bit is added to all memory system transactions, including cache tags and access to system memory and peripherals. The NS bit can give a different physical address space for the Secure and the Normal worlds.

Software running in the Normal world can only make Non-secure accesses to memory, because the core always sets the NS bit to 1 in any memory transaction that is generated by the Normal world, irrespective of what is set in the Normal world translation tables. Software running in the Secure world usually makes only Secure memory accesses, but can also make Non-secure accesses for specific memory mappings using the NS and NSTable flags in its translation table entries.

Trying to perform a Non-secure access to cached data that is marked as Secure causes a cache miss. Trying to perform a Non-secure access to external memory marked as Secure will generally return an error response to the core. There is no indication to the Non-secure system that the error is caused by an attempted access to Secure memory.

EL3 has its own translation tables, which are governed by the *Translation Table Base Register* TTBR0_EL3 and the *Translation Control Register* TCR_EL3. Only stage one translations are allowed in the Secure world and there is no TTBR1_EL3. The EL1 translation table registers are not banked between security states and therefore the value of TTBR0_EL1, TTBR1_EL1, and TCR_EL1 must be saved and restored for each world as part of the context switching operation by the Secure Monitor.

This enables each world to have a local set of translation tables, with the Secure world mappings hidden and protected from the Normal world. Entries in the Secure world translation tables contain NS and NSTable attribute bits that determine whether particular accesses can access the Secure or Non-secure physical address space.

Secure and Non-secure entries can co-exist within the caches and TLBs. There is no need to invalidate cache data when switching between worlds. The Normal world can only generate Non-secure accesses, so can only hit on cache lines marked as Non-secure, whereas the Secure world can generate both Secure and Non-secure accesses. This may require some cache management if the security state is changed between accesses, this may require some cache management if the security state is changed between accesses.

Entries in the TLB record which world generates a particular entry, and although the Non-secure state can never operate on Secure data, the Secure world can allocate NS lines into the cache. Also, the caches are enabled and disabled separately for each of the Exception levels. Cache control is independent for the two worlds, but is not independent for all Exception levels, so EL0 can never enable or disable the caches directly, and EL2 can override behavior for Non-secure EL1.

3 Interaction of Normal and Secure worlds

If you are writing code in a system that contains some Secure services, it can be useful to understand how these are used. A typical system has a light-weight kernel or *Trusted Execution Environment* (TEE), hosting services, for example, such as encryption in the Secure world. This interacts with a rich OS in the Normal world that can access the Secure services using the SMC call. In this way, the Normal world has access to service functions, without exposing the keys to risk.

Generally, application developers do not directly interact with Security Extensions, TEEs, or Trusted Services. Instead, they use a high-level API such as, `authenticate()`, provided by a Normal world library. The library is provided by the same vendor as the Trusted Service, for example, a credit card company, and handles the low-level interactions. The following figure shows this interaction in the form of a flow from the user application calling the API that makes an appropriate OS call, which passes to the driver code, and then passes execution into the TEE through the Secure monitor.

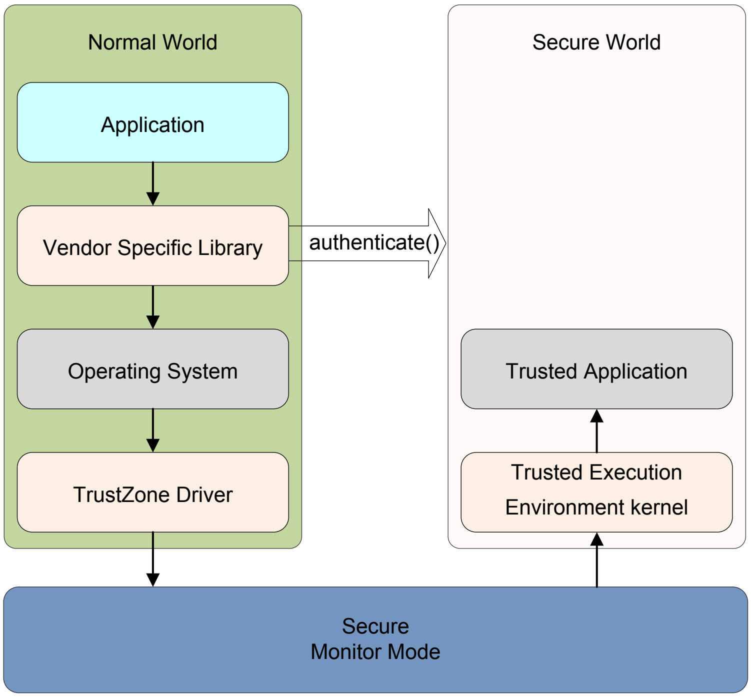

Figure 2 Interaction with Security Extension

It is often necessary for data to be passed between the Secure and Normal worlds.

For example, in the Secure world you might have a signature checker. The Normal world can request that the Secure world verifies the signature of a downloaded update, using the `SMC` call. The Secure world requires access to the memory used by the Normal world. The Secure world can use the NS-bit in its translation table descriptors to ensure that it uses Non-secure accesses to read the data.

This is important because data relating to the package might already be in the caches, because of the accesses that are executed by the Normal world with addresses marked as Non-secure. The security attribute can be thought of as an extra address bit. If the core uses Secure memory accesses to try to read the package, it will not hit on Non-secure data already in the cache.

If you are a Normal world programmer, in general, you can ignore what happens in the Secure world, as its operation is hidden from you. One side effect is that interrupt latency can increase slightly. The Secure world can be fully blocking so if an interrupt occurs to request Secure kernel execution this might block the Normal world interrupts, but this increase is small compared to the overall latency on a typical OS. Quality-of-service issues of this type depend on good design and implementation of the Secure world OS.

4 Switching between the Secure and Normal worlds

With the ARMv7 Security Extensions, Monitor mode is used by software to switch between the Secure and Non-secure state. This mode is a peer of the other privileged modes within the Secure state. In ARMv8-A processors, AArch32 is the equivalent of ARMv7-A.

For the ARMv8 architecture, when EL3 is using AArch32 the system behaves as ARMv7 to ensure full compatibility, with the result that all the privileged modes within the Secure state are treated as being at EL3.

The security model for AArch32 is shown in the figure below. In this scenario, EL3 is AArch32 to provide a Secure OS and monitor.

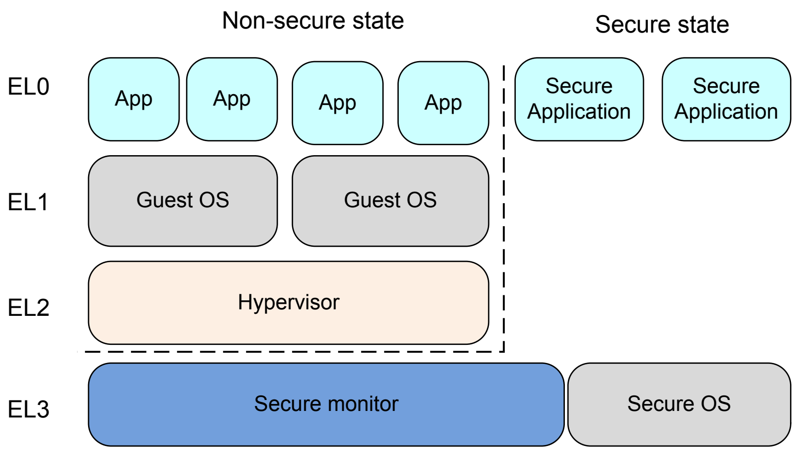

Figure 3 Security model when EL3 is AArch32

The following figure shows the security model when EL3 is executing AArch64 to provide a Secure monitor. EL1 is used for the secure OS. When EL3 is using AArch64, the EL3 level is used to execute the code responsible for switching between the Non-secure state and the Secure state.

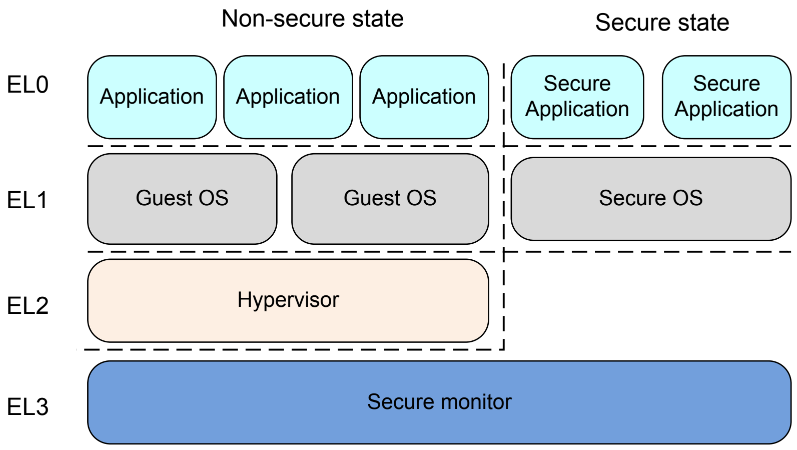

Figure 4 Security model when EL3 is AArch64

In keeping with AArch32, the Secure state EL1 and EL0 have a different virtual address space from the Non-secure state EL1 and EL0. This permits secure side code from AArch32 32-bit architecture to be used in a system with a 64-bit operating system or hypervisor running on the Non-secure side.

As Normal world execution ceases and Secure world execution starts, context switching between them occurs through execution of the Secure Monitor (SMC) instruction or by hardware exception mechanisms, such as interrupts or asynchronous aborts. ARM processors have two interrupt types, FIQ, and IRQ.

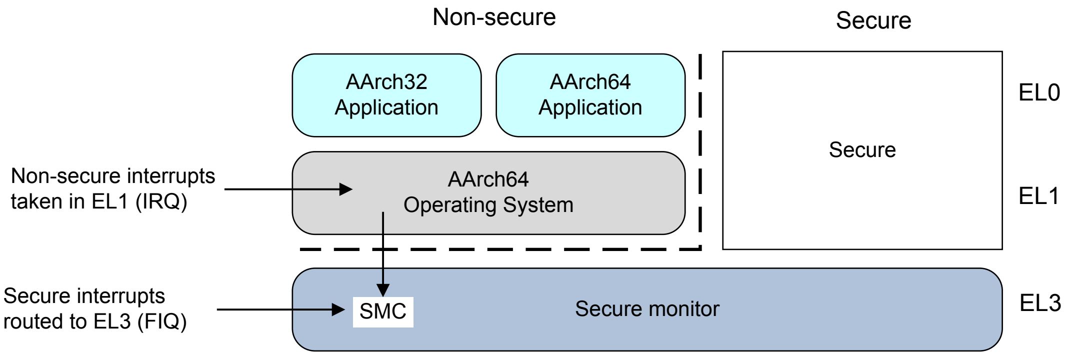

Figure 5 Non-secure interrupts

There is explicit support for Secure interrupts in the form of controls for redirecting exceptions and interrupts to EL3, independently of the current DAIF fields. However, these controls only distinguish between the main interrupt types: IRQ, FIQ, and asynchronous aborts. More detailed control requires interrupts to be filtered into Secure and Non-secure groups. Doing this efficiently requires support from the GIC, which has explicit facilities for this purpose.

One typical use case is for FIQs to be used as Secure interrupts, by mapping Secure interrupt sources as FIQ within the interrupt controller. The relevant peripheral and interrupt controller registers must be marked as Secure access only, to prevent the Normal world from reconfiguring these interrupts. These Secure FIQ interrupts must be routed to handlers in the Secure Execution state.

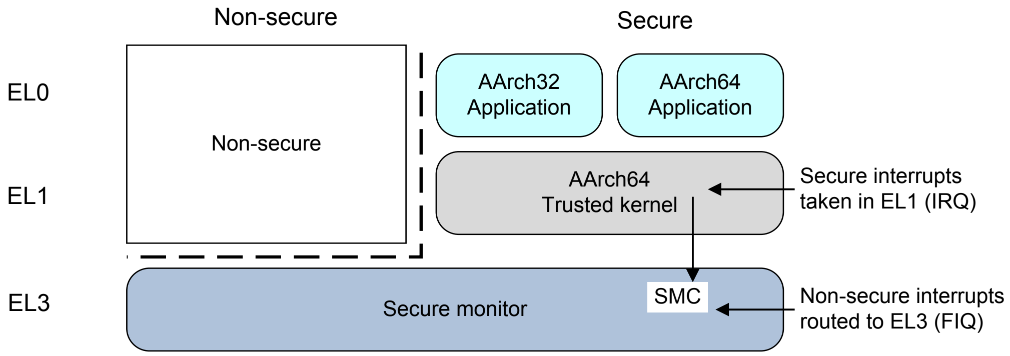

Figure 6 Secure interrupts

Implementations that use Security Extensions typically have a light-weight trusted kernel that hosts secure services, such as encryption, in the Secure world. A full operating system runs in the Normal world and is able to access the Secure services using the SMC instruction. In this way, the Normal world gets access to service functions without risking exposure of secure assets, such as key material or other protected data, to arbitrary code executing in the Normal world.

5 Security in clusters

Each core in a cluster system has the same security features. Any number of the cores in the cluster can be executing in the Secure world at any point in time, and cores are able to transition between the worlds independently of each other.

Registers control whether Normal world code can modify *Snoop Control Unit* (SCU) settings. Similarly, the GIC that distributes prioritized interrupts across the cluster must be configured to be aware of security concerns.

5.1 Secure debug

The security system also controls availability of debug provision. You can configure separate hardware over full JTAG debug and trace control for Normal and Secure software worlds, so that no information about the trusted system leaks. You can control hardware configuration options through a Secure peripheral or you can hardwire them and control them using the following signals:

- Secure Privileged Invasive Debug Enable (**SPIDEN**): JTAG debug.
- Secure Privileged Non-Invasive Debug Enable (**SPNIDEN**): Trace and Performance Monitor.