2023-04-09

Created: 2023-05-01 22:18:02

Chats

- FF-A.0409.md
  - `[09:00]` The below text are chapter 2 of “Arm Firmware Framework for Arm A-profile” … …`(FF-A.0409.md)`
  - `[09:09]` The below text are chapter 3 of “Arm Firmware Framework for Arm A-profile” … …`(FF-A.0409.md)`
  - `[09:17]` The below text are chapter 3.1 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`
  - `[09:21]` The below text are chapter 3.2 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`
  - `[09:45]` The below text are chapter 3.3 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`
  - `[09:52]` The below text are chapter 3.4 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`
  - `[10:05]` In ARM partition manager, Whan A is isolated from B. Could I say that B is also isolated from A?`(FF-A.0409.md)`
  - `[10:05]` Are you sure? `(FF-A.0409.md)`
  - `[10:14]` Can SPM see SP0 and SP1? `(FF-A.0409.md)`
  - `[10:15]` Can SP0 or SP1 see SPM? `(FF-A.0409.md)`
  - `[10:17]` Is TrustZone isolation boundary physical isolation or logical isolation? `(FF-A.0409.md)`
  - `[10:26]` The below text are chapter 4.1 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`
  - `[10:46]` What is SPMC? `(FF-A.0409.md)`
  - `[10:48]` What is SPMC in the document? `(FF-A.0409.md)`
  - `[10:48]` Is SPMC of TrustZone? `(FF-A.0409.md)`
  - `[11:15]` So SPM and Hypervisor are partition managers. Right? `(FF-A.0409.md)`
  - `[11:15]` What is VMSA? `(FF-A.0409.md)`
  - `[11:16]` Is VMSA a software or a hardware? `(FF-A.0409.md)`
- OP-TEE.0327.md
  - `[10:32]` Does TrustZone require hypervisor? `(OP-TEE.0327.md)`
  - `[20:37]` https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html `(OP-TEE.0327.md)`
  - `[20:40]` The following text are from https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html … … `(OP-TEE.0327.md)`
  - `[20:42]` The following continued text are from https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html … … `(OP-TEE.0327.md)`
  - `[20:44]` How does OP-TEE work with HUK? `(OP-TEE.0327.md)`
  - `[21:07]` What are common device that store HUK in ARM platform with OP-TEE?`(OP-TEE.0327.md)`
- TrustZone.0329.md
  - `[11:21]` What are components in ARM TCB? `(TrustZone.0329.md)`
  - `[18:24]` Does TrustZone require SPM? `(TrustZone.0329.md)`

Q & A

FF-A.0409.md

Question: `[09:00]` The below text are chapter 2 of “Arm Firmware Framework for Arm A-profile” … …`(FF-A.0409.md)`

Answer:

Chapter 2: Introduction

The Armv8.4 architecture introduces the Virtualization extension in the Secure state, and the Arm® SMMU v3.2 architecture adds support for stage 2 translations for Secure streams. These features enable the isolation of mutually mistrusting software components in the Secure state, thereby implementing the principle of least privilege. Virtualization in the Secure state allows for the isolation of firmware in EL3 from software in S-EL1, firmware components in EL3 from each other, and normal world software from software in S-EL1.

This specification outlines a software architecture called the Firmware Framework for Arm® A-profile processors (also referred to as FF-A), which aims to:

1. Use the Virtualization extension to isolate software images provided by different vendors.
2. Standardize interfaces for communication between various software images in the Secure and Normal worlds.
3. Generalize interactions between a software image and privileged firmware in the Secure state.

The Framework also provides guidance for systems without the Virtualization extension in the Secure state and for communication between VMs managed by a Hypervisor in the Normal world. This ensures a migration path for existing Secure world software images and allows Hypervisors to use the Firmware Framework for communication and isolation management between VMs.

More information about the introduction of the Virtualization extension in the Secure state and the goals of the Firmware Framework can be found in the white paper titled “Isolation using virtualization in the Secure world”.

Question: `[09:09]` The below text are chapter 3 of “Arm Firmware Framework for Arm A-profile” … … `(FF-A.0409.md)`

Answer:

Chapter 3: Building Blocks of the Firmware Framework

The Firmware Framework for Arm A-profile consists of five main building blocks, as shown in Figure 3.1:

1. Isolation Boundaries
2. Partition Interfaces
3. Partitions
4. Partition Manifest
5. Partition Manager

The following sub-sections provide more details about these building blocks:

1. Isolation Boundaries: These boundaries separate and protect software components from one another, ensuring that they cannot access each other’s resources without proper authorization. This helps to maintain the principle of least privilege and provides a secure environment for software execution.

2. Partition Interfaces: These interfaces standardize communication between software components within the Secure and Normal worlds. They enable the exchange of information and requests between partitions, ensuring that components can communicate effectively while maintaining isolation.

3. Partitions: Partitions are the individual software components that run within the framework. They can include firmware, trusted operating systems, or other secure services. Each partition is isolated from the others and operates within its own environment, with access to only the resources it needs to function properly.

4. Partition Manifest: The Partition Manifest is a metadata file that describes the properties and requirements of a partition. It includes information such as the partition’s entry points, memory requirements, and other resources it needs to function. The Partition Manager uses this manifest to configure and manage the partition within the system.

5. Partition Manager: The Partition Manager is a critical component of the Firmware Framework, responsible for managing partitions and their associated resources. It oversees the lifecycle of partitions, including their creation, execution, and termination. The Partition Manager also ensures that partitions are properly isolated and that they only have access to the resources defined in their Partition Manifest.

Question: `[09:17]` The below text are chapter 3.1 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`

Answer:

Chapter 3.1: Isolation Boundaries

The Firmware Framework defines two types of isolation boundaries: logical isolation and physical isolation.

1. Logical Isolation Boundary: This boundary can be used to isolate software modules (e.g. libraries or device drivers) within a software image in an exception level or to isolate software images from one another. These services can be accessed through an IMPLEMENTATION DEFINED API or an application binary interface (ABI).
2. Physical Isolation Boundary: This boundary is used to spatially isolate the physical address space of one software image from another. It relies on mechanisms such as the Arm® TrustZone Security extension and the Arm A-profile VMSA to protect the physical address space ranges assigned to software images.

The Framework assumes that a physically isolated software image is also logically isolated but does not assume that a logically isolated software image is physically isolated. The Framework defines ABIs that enable communication between software images across an exception level boundary, allowing for logical and potential physical isolation.

Question: `[09:21]` The below text are chapter 3.2 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`

Answer:

Chapter 3.2: Partitions

A partition in the Firmware Framework is a software module or image that implements one or more services within an isolation boundary. These services are accessible across the boundary only via well-defined interfaces, such as FF-A ABIs for software images and IMPLEMENTATION DEFINED APIs for software modules.

The Framework defines ABIs for partitions to discover other partitions, their properties, and services, as well as for synchronous and asynchronous message passing and memory management between partitions.

Partitions can be logical (logically isolated but not physically isolated) or physical (both physically and logically isolated). The term “endpoint” is used interchangeably with “partition.” There are two types of endpoints:

1. NS-Endpoints: VMs or OS kernels running in EL1 in the Non-secure security state. These can be either physical or logical partitions.
2. S-Endpoints: Secure Partitions (SPs) in the Secure security state, which can be logical partitions running in EL3, S-EL2, or S-EL1, or physical partitions running in S-EL1 or S-EL0.

A partition manifest describes the required physical address space ranges, system resources, partition services, and other attributes governing the partition’s run-time behavior.

Question: `[09:45]` The below text are chapter 3.3 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`

Answer:

Chapter 3.3: Partition Manager

The Partition Manager is responsible for creating and managing the physical isolation boundary of a partition. It uses a partition manifest to assign resources to a partition, initialize it, and enable the discovery of its services. The Partition Manager also implements FF-A ABIs for inter-partition communication. In the Secure world, this component is called the Secure Partition Manager (SPM), while in the Normal world, it is a Hypervisor.

A Partition Manager is physically isolated from physical partitions and logically isolated from logical partitions it manages. The term “partition manager” is used to refer to both the SPM and Hypervisor when their responsibilities are the same.

The Hypervisor uses the virtualization extension in the Arm A-profile VMSA to create physical isolation boundaries, while the SPM does so for SPs. The Firmware Framework defines trust boundaries concerning partition managers and partitions. The

SPM is part of the Trusted Computing Base (TCB) for resources assigned to the Secure state, and both the Hypervisor and SPM are part of the TCB for resources assigned to the Non-secure state. VMs trust the Hypervisor, and all endpoints trust the SPM to protect their resources. However, an SP does not trust the state of any Non-secure resource it has access to, and therefore does not trust the Hypervisor or a NS-Endpoint that could also access the same resource. The term “FF-A component” is used to refer to partitions and partition managers collectively.

Question: `[09:52]` The below text are chapter 3.4 of “Arm Firmware Framework for Arm A-profile”. Please summarize it. … … `(FF-A.0409.md)`

Answer:

Chapter 3.4: Example Configurations

The Arm Firmware Framework for Arm A-profile can be deployed in various configurations on an Arm A-profile system. The programming model is generalized to access a partition’s services within and between the Non-secure and Secure security states. Some example deployment scenarios are:

3.4.1 FF-A deployment without S-EL2: In this scenario, the virtualization extension is enabled in the Non-secure state, but unavailable or disabled in the Secure state. VM0 and VM1 use FF-A drivers to access services in S-Endpoints. The Hypervisor facilitates access to these services. In the Secure world, a firmware image in EL3 implements the SPM, and firmware images in S-EL1 (SP1) and a Trusted OS image in S-EL1 (SP0) are deployed as temporally isolated logical partitions.

3.4.2 FF-A deployment with S-EL2: In this scenario, the virtualization extension is enabled in both security states. The Normal world software stack remains unchanged from the previous example. In the Secure world, a firmware image in EL3, an SPM image in S-EL2, a firmware image in S-EL1 (SP1), and a Trusted OS image in S-EL1 (SP0) are deployed. SP0 and SP1 are physical partitions that can access each other’s services via the SPM, which is physically isolated from them.

3.4.3 FF-A deployment with S-EL2 and Armv8.1-VHE: In this scenario, the virtualization extension is enabled in both security states, and Armv8.1 VHE is enabled in the Secure world to manage S-EL0 SPs. The Normal world software stack remains unchanged from the first example. In the Secure world, a firmware image in EL3, an SPM image in S-EL2, and firmware images in S-EL0 (SP1 and SP0) are deployed. SP0 and SP1 are physical partitions that can access each other’s services, and the SPM is physically isolated from them.

Question: `[10:05]` In ARM partition manager, Whan A is isolated from B. Could I say that B is also isolated from A? `(FF-A.0409.md)`

Answer:

Yes, in the ARM partition manager, when partition A is isolated from partition B, it means that B is also isolated from A. Isolation boundaries ensure that both partitions cannot directly access each other’s resources, providing security and stability for each partition’s operation.

Question: `[10:05]` Are you sure? `(FF-A.0409.md)`

Answer:

Yes, I am sure. In the ARM Firmware Framework, isolation boundaries are designed to prevent unauthorized access between partitions. When partition A is isolated from partition B, it implies that both partitions are protected from each other, and neither partition can access the other’s resources without proper authorization. So, when A is isolated from B, it also means that B is isolated from A.

Question: `[10:14]` Can SPM see SP0 and SP1? `(FF-A.0409.md)`

Answer:

Yes, the Secure Partition Manager (SPM) can “see” and manage both SP0 and SP1. The SPM is responsible for creating and managing the physical isolation boundaries of Secure Partitions (SPs) like SP0 and SP1. It uses partition manifests to assign resources, initialize the partitions, and enable discovery of their services. Additionally, the SPM implements FF-A ABIs

(Application Binary Interfaces) to enable inter-partition communication for access to partition services. So, while the SPM isolates SP0 and SP1 from each other, it still has visibility and control over both partitions.

Question: `[10:15]` Can SP0 or SP1 see SPM? `(FF-A.0409.md)`

Answer:

In the ARM Firmware Framework, SP0 and SP1 can communicate with the Secure Partition Manager (SPM) through well-defined interfaces, such as FF-A ABIs (Application Binary Interfaces). While the SPM is physically isolated from SP0 and SP1 in scenarios where they are physical partitions, they can still interact with the SPM using these interfaces for communication, service discovery, and memory management.

However, it is important to note that SP0 and SP1 cannot access the internal workings or the memory of the SPM directly. They can only communicate with the SPM through the defined interfaces, ensuring that the isolation between them is maintained.

Question: `[10:17]` Is TrustZone isolation boundary physical isolation or logical isolation? `(FF-A.0409.md)`

Answer:

TrustZone isolation boundary is a form of physical isolation. TrustZone is a security extension in ARM architecture that separates the processor's resources, such as memory and peripherals, into two distinct security domains: the Secure world and the Non-secure world. By utilizing hardware-based isolation, TrustZone ensures that sensitive data and functions in the Secure world are protected from unauthorized access by software running in the Non-secure world.

Question: `[10:26]` The below text are chapter 4.1 of "Arm Firmware Framework for Arm A-profile". Please summarize it. … … `(FF-A.0409.md)`

Answer:

Chapter 4.1 of the "Arm Firmware Framework for Arm A-profile" discusses the Secure Partition Manager (SPM) architecture. The SPM is divided into two components: the SPM Dispatcher (SPMD) and the SPM Core (SPMC). Both components are part of the Trusted Computing Base and have access to the entire physical address space.

The SPMD resides in EL3 and is responsible for the SPM Core initialization at boot time, forwarding FF-A calls between the Normal world and the SPM Core. The SPMC can reside in EL3, S-EL1, or S-EL2 and is responsible for SP initialization and isolation at boot time, inter-partition isolation, and inter-partition communication between S-Endpoints and NS-Endpoints.

The document describes various SPM configurations based on the exception levels and execution states of the SPMD and SPMC. It also provides examples of these configurations for Secure EL2 SPM core component, S-EL1 SPM core component, and EL3 SPM core component.

These configurations are essential for enforcing the principle of least privilege, reducing the size of the Trusted Computing Base, making the TCB implementation more robust, and adopting the generalized programming model specified by the Framework.

Question: `[10:46]` What is SPMC? `(FF-A.0409.md)`

Answer:

SPMC stands for Secure Partition Manager Core. It is one of the two main components of the Secure Partition Manager (SPM) in the Arm Firmware Framework for Arm A-profile. The SPMC is responsible for Secure Partition (SP) initialization and isolation at boot time, ensuring inter-partition isolation during runtime, and managing inter-partition communication between Secure Endpoints (S-Endpoints) and Non-Secure Endpoints (NS-Endpoints). The SPMC can reside in EL3 (Exception Level 3), S-EL1 (Secure Exception Level 1), or S-EL2 (Secure Exception Level 2), depending on the configuration.

Question: `[10:48]` What is SPMC in the document? `(FF-A.0409.md)`

Answer:

In the document “Arm Firmware Framework for Arm A-profile,” the SPMC (Secure Partition Manager Core) is a component of the Secure Partition Manager (SPM) responsible for:

1. Initializing and isolating Secure Partitions (SPs) at boot time.
2. Ensuring inter-partition isolation during runtime.
3. Managing inter-partition communication between Secure Endpoints (S-Endpoints) and Non-Secure Endpoints (NS-Endpoints).

The SPMC can co-reside with the SPM Dispatcher (SPMD) in EL3 or reside in adjacent exception levels, such as S-EL1 or S-EL2. It is a part of the Trusted Computing Base and has access to the entire physical address space.

Question: `[10:48]` Is SPMC of TrustZone? `(FF-A.0409.md)`

Answer:

The SPMC (Secure Partition Manager Core) is part of the Arm Firmware Framework for Arm A-profile, which is built on top of the Arm TrustZone technology. TrustZone provides the underlying hardware support for isolation and security, while the SPMC is responsible for managing partitions and their communication within the Secure world enabled by TrustZone. So, while the SPMC is not TrustZone itself, it is a component that utilizes and operates within the TrustZone technology.

Question: `[11:15]` So SPM and Hypervisor are partition managers. Right? `(FF-A.0409.md)`

Answer:

Yes, that’s correct. In the context of the Arm Firmware Framework for Arm A-profile, the Secure Partition Manager (SPM) operates in the Secure world, while the Hypervisor operates in the Normal world. Both SPM and Hypervisor are partition managers responsible for creating and managing the physical isolation boundaries of partitions, assigning resources, and enabling inter-partition communication.

Question: `[11:15]` What is VMSA? `(FF-A.0409.md)`

Answer:

VMSA, or Virtual Memory System Architecture, is a part of the Arm architecture that defines the memory management and address translation mechanisms. It is responsible for managing the address spaces and memory protection features in Arm systems. VMSA allows for the creation of separate virtual address spaces for each process, providing isolation between processes and the operating system. In the context of the Arm A-profile, the Hypervisor uses the virtualization extension in VMSA to create physical isolation boundaries between different partitions or virtual machines.

Question: `[11:16]` Is VMSA a software or a hardware? `(FF-A.0409.md)`

Answer:

VMSA (Virtual Memory System Architecture) is not a specific piece of software or hardware, but rather a part of the Arm architecture specification that defines how memory management and address translation should be implemented in Arm-based systems. VMSA provides guidelines for designing hardware and software components that handle memory management, address translation, and access control, allowing for the creation of separate virtual address spaces and ensuring proper isolation between processes and the operating system.

OP-TEE.0327.md

Question: `[10:32]` Does TrustZone require hypervisor? `(OP-TEE.0327.md)`

Answer:

No, TrustZone does not require a hypervisor. TrustZone is a hardware-based security technology implemented in Arm processors that provides a secure execution environment by creating two separate worlds: the Secure World and the Normal World. TrustZone itself is not dependent on a hypervisor.

A hypervisor is a software layer that manages and coordinates multiple virtual machines (VMs) running on a single hardware platform. Hypervisors can be used in conjunction with TrustZone to provide additional isolation and management capabilities for virtualized systems.

While TrustZone and hypervisors can work together, they serve different purposes and are not dependent on one another. TrustZone focuses on providing a secure environment for trusted applications, while hypervisors manage VMs and their resources.

Question: `[20:37]` https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html `(OP-TEE.0327.md)`

Answer:

The link you provided directs to the OP-TEE porting guidelines, which offer information on how to port OP-TEE to a new platform. OP-TEE, or Open Portable Trusted Execution Environment, is an open-source implementation of TEE that provides a secure environment for trusted applications to run on Arm processors with TrustZone.

The porting guidelines cover the following topics:

1. Prerequisites: This section describes the requirements for porting OP-TEE, including the platform's support for Arm TrustZone, a GlobalPlatform TEE compliant API, a working toolchain, and the necessary cryptographic libraries.
2. Source tree structure: This section explains the directory structure of the OP-TEE project and where to add new platform-specific code.
3. Platform configuration: This section discusses the various configuration options available for a new platform, such as the platform flavor, the target CPU architecture, and the memory layout.
4. Adding a new platform: This section provides a step-by-step guide on how to add a new platform to the OP-TEE project, including creating a new directory for the platform, implementing platform-specific code, and updating the platform configuration files.
5. Platform-specific code: This section covers the platform-specific code that needs to be implemented when porting OP-TEE to a new platform, such as initialization routines, memory management, and interrupt handling.
6. Core APIs: This section discusses the core APIs that need to be implemented for a new platform, including cache maintenance, MMU configuration, and Secure Monitor Call (SMC) handling.
7. Testing and debugging: This section provides guidance on testing and debugging a new platform, including using the xtest suite for testing and recommendations for debugging tools.

By following these guidelines, developers can port OP-TEE to a new platform, allowing the platform to leverage the security features provided by OP-TEE and TrustZone.

Question: `[20:40]` The following text are from https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html … … `(OP-TEE.0327.md)`

Answer:

The text you shared from the OP-TEE porting guidelines provides a detailed overview of how to add a new platform to OP-TEE. This involves creating and configuring platform-specific files within the core/arch/arm directory, such as:

1. conf.mk: A device-specific makefile to define configurations unique to your platform, such as OP-TEE configuration variables (CFG_) and compiler flags.
2. main.c: A platform-specific file containing power management handlers and UART-related code.
3. platform_config.h: A mandatory header file for every platform, which contains memory configuration, base addresses, and other platform-specific definitions.

In the provided text, a fictitious platform named “gendev” is used as an example to illustrate the process of creating and configuring the necessary files for a new platform. The text covers the setup of the platform’s UART, memory addresses, and other configurations.

Keep in mind that this is only a part of the porting process; there are other aspects to consider, such as implementing platform-specific code, core APIs, and testing and debugging the new platform.

Question: `[20:42]` The following continued text are from https://optee.readthedocs.io/en/latest/architecture/porting_guidelines.html … … `(OP-TEE.0327.md)`

Answer:

and platform-specific security guidelines for more information on the security features and configurations specific to your device.

Conclusion

This document provides an overview of the process of porting OP-TEE to a new platform, including the steps involved and the various considerations to ensure security features are properly implemented. However, it is essential to consult platform-specific documentation and guidelines provided by the chip manufacturer to ensure the security of the end product.

Some of the key aspects to consider when porting OP-TEE to a new platform include:

1. Official board support in OP-TEE: Update platforms supported, .shippable.yml, maintainers, and build.git and manifest.git files.
2. Hardware Unique Key (HUK): Ensure a secure implementation for the device’s hardware unique key.
3. Secure Clock: Implement a trustable source of time if available on the hardware.
4. Root and Chain of Trust: Establish a trust anchor on the device to ensure untampered binaries are running.
5. Hardware Crypto IP: Implement the Crypto API for the device’s dedicated crypto block if available.
6. Random Number Generator: Make use of a hardware entropy source if available on the platform.
7. Power Management / PSCI: Implement power management functions specific to the device.
8. Memory firewalls / TZASC: Ensure proper configuration of memory firewalls on the device.
9. Trusted Application private/public keypair: Replace the default key and securely manage private keys used for signing Trusted Applications.

By carefully considering and implementing these aspects, you can ensure a secure port of OP-TEE to your target platform. Always refer to the chip manufacturer’s documentation and guidelines for platform-specific security configurations and best practices.

Question: `[20:44]` How does OP-TEE work with HUK? `(OP-TEE.0327.md)`

Answer:

OP-TEE (Open Portable Trusted Execution Environment) is designed to provide a secure environment for the execution of trusted applications on a device. One of the critical components for providing security within OP-TEE is the use of a Hardware Unique Key (HUK).

A Hardware Unique Key is a device-specific secret key that is usually derived from or securely stored in hardware. HUKs are essential for providing a secure root of trust, as they can be used to derive other cryptographic keys and establish secure communication channels between different components of a system.

Here’s how OP-TEE works with HUK:

1. In OP-TEE, the HUK is used to derive other keys for various purposes, such as secure storage or cryptographic operations. The derived keys are typically used in combination with other keys or cryptographic primitives to ensure the confidentiality, integrity, and authenticity of the data being processed.

2. The function tee_otp_get_hw_unique_key(…) in core/include/kernel/tee_common_otp.h is responsible for retrieving the HUK from the hardware. In a real secure product, this function must be replaced with an implementation that retrieves the HUK from the device’s hardware. This ensures that the HUK remains protected and cannot be accessed directly from

software, even on the secure side.

3. If the device does not have hardware support for a HUK, it is crucial to change the implementation of the tee_otp_get_hw_unique_key(…) function to use something other than zeroes. Storing a key in software is not a secure practice, especially for a key that serves as the root for all other keys. As such, it is not recommended, but it may be necessary if hardware support is lacking.
4. Once the HUK is securely retrieved, it can be used by OP-TEE to derive other keys or perform cryptographic operations necessary for securing trusted applications and their data. The HUK helps establish a root of trust that can be extended to other parts of the system.

In summary, OP-TEE works with HUKs by leveraging them to derive other cryptographic keys and establish a root of trust for secure operations. The HUK is typically retrieved from the hardware using a secure function, and it is essential for providing a strong foundation for the security of the entire system.

Question: `[21:07]` What are common device that store HUK in ARM platform with OP-TEE? `(OP-TEE.0327.md)`

Answer:

Many devices in the ARM platform, especially those utilizing TrustZone technology, store Hardware Unique Keys (HUKs) for providing a secure root of trust. Some common devices that store HUKs and are compatible with OP-TEE include:

1. NXP i.MX family: Devices in the i.MX family, like i.MX6, i.MX7, and i.MX8, use ARM TrustZone and secure boot capabilities. They have integrated security features, such as cryptographic accelerators and secure storage, where the HUK can be stored.
2. Xilinx Zynq UltraScale+ MPSoC: The Zynq UltraScale+ MPSoC has ARM Cortex-A53 and Cortex-R5 cores with TrustZone technology. It provides secure boot and supports storage of HUK in the on-chip eFUSE memory.
3. STMicroelectronics STM32MP1: The STM32MP1 series, with its ARM Cortex-A7 and Cortex-M4 cores, supports OP-TEE and TrustZone. It provides secure storage for HUKs using its OTP (One-Time Programmable) memory or other secure storage mechanisms.
4. HiSilicon HiKey and HiKey 960: These development boards, based on HiSilicon's Kirin SoCs, feature ARM Cortex-A cores with TrustZone technology. They can be used with OP-TEE and support secure storage for HUKs.
5. Raspberry Pi 3 (unofficially): While not officially supported by OP-TEE, Raspberry Pi 3, featuring a Broadcom SoC with ARM Cortex-A53 cores, has been known to run OP-TEE. However, the secure storage of HUKs on Raspberry Pi 3 may not be as robust as in other devices designed with hardware security features in mind.

These are just a few examples of devices that store HUKs and can work with OP-TEE on the ARM platform. It's important to note that the secure storage and management of HUKs may vary depending on the specific device, its hardware features, and the implementation of OP-TEE. Always consult the device's documentation and follow the manufacturer's guidelines to ensure proper security measures are in place.

TrustZone.0329.md

Question: `[11:21]` What are components in ARM TCB? `(TrustZone.0329.md)`

Answer:

The Trusted Computing Base (TCB) of an ARM system consists of the essential components responsible for maintaining security properties and enforcing security policies. In the context of ARM TrustZone and related technologies, the TCB can include the following components:

1. Secure Boot: This is the mechanism that ensures the integrity and authenticity of the boot process, making sure that only trusted software components are executed during the system startup.
2. ARM TrustZone: A hardware-based security feature in ARM processors, which provides a secure execution environment (Secure World) and isolates it from the less secure environment (Normal World).
3. Secure Monitor: This is the lowest-level software component running in the Secure World. It manages transitions

between the Secure and Normal Worlds and provides an interface for Secure World services.

4. Trusted Firmware-A (TF-A) or other trusted firmware: This firmware runs in the Secure World and provides various services, such as secure boot, power management, and platform-specific configurations.
5. OP-TEE or other Trusted Execution Environment (TEE) implementations: TEEs, like OP-TEE, provide a secure environment for running trusted applications (TAs) within the Secure World, ensuring that sensitive code and data are protected from unauthorized access or tampering.
6. Secure Partition Manager (SPM): This component manages Secure Partitions (SPs) within the TrustZone Secure World, including creating, configuring, and managing SPs, as well as inter-partition communication and resource management.
7. Trusted Applications (TAs): These are secure software components that run within the Secure World and provide security-sensitive services or perform security-critical operations.
8. Security-sensitive hardware components: These components, such as cryptographic accelerators, secure storage, and hardware random number generators, provide additional security features and services to the system.
9. Secure peripherals: These are peripherals that can be accessed only from the Secure World, providing additional isolation and protection for sensitive data and operations.

Please note that the specific components in an ARM TCB may vary depending on the system's architecture, SoC, and security requirements. The list above provides a general overview of the typical components in an ARM TrustZone-based TCB.

Question: `[18:24]` Does TrustZone require SPM? `(TrustZone.0329.md)`

Answer:

TrustZone itself does not inherently require a Secure Partition Manager (SPM). TrustZone is a hardware-based security feature in ARM processors that provides a secure execution environment (Secure World) and isolates it from the less secure environment (Normal World).

However, when using ARMv8.4-A architecture or later, which introduces the concept of Secure Partitions (SPs) to improve isolation and modularity of the Secure World components, an SPM becomes an important part of the system. The SPM is responsible for managing the lifecycle of Secure Partitions and handling inter-partition communication and resource management.

So, while TrustZone doesn't require an SPM in its basic form, utilizing SPM can enhance the security and modularity of the system when working with Secure Partitions in newer ARM architectures.
