2023-04-01

Created: 2023-05-01 22:17:45

Questions

- OP-TEE `(230410.Arm.yaml)`
  - `[10:50]` Does OP-TEE support ARMv7 and ARMv8? `(OP-TEE.0327.md)`
- Boot `(230329.TrustZone.yaml)`
  - `[15:46]` What is ATF? `(TrustZone.0329.md)`
- Basic Concepts `(230328.TEE.yaml)`
  - `[15:48]` Is TEE a software? `(TEE.0328.md)`
  - `[23:45]` Does GlobalPlatform contains Chinese Cryptographic Standard? `(TEE.0328.md)`
- Development `(230327.OP-TEE.yaml)`
  - `[19:29]` Does OP-TEE have a shell? `(OP-TEE.0327.md)`
- Secure Storage `(230327.OP-TEE.yaml)`
  - `[19:30]` Does OP-TEE support file system? `(OP-TEE.0327.md)`
- Static TA and Dynamic TA `(230327.OP-TEE.yaml)`
  - `[21:16]` Can I create a TA at run-time? `(OP-TEE.0327.md)`
  - `[21:23]` But what is TEECI_LoadSecureModule() `(OP-TEE.0327.md)`
  - `[23:20]` Does TEE define Static TA and Dynamic TA? `(OP-TEE.0327.md)`
- Memory `(230329.TrustZone.yaml)`
  - `[21:58]` What is MMU? `(TrustZone.0329.md)`
  - `[22:01]` What is TLB? `(TrustZone.0329.md)`
- CA & TA `(230328.TEE.yaml)`
  - `[23:16]` What are kinds of TA? `(TEE.0328.md)`
- CPU `(230329.TrustZone.yaml)`
  - `[23:31]` What is SMP? `(TrustZone.0329.md)`
  - `[23:41]` What are difference between primary and seconday CPU cores? `(TrustZone.0329.md)`
  - `[23:42]` Please summarize the two types of CPU cores as a table. `(TrustZone.0329.md)`
- Boot `(230327.OP-TEE.yaml)`
  - `[23:36]` Why does OP-TEE's reset function have two branches reset_primary and reset_secondary? `(OP-TEE.0327.md)`
- Signature `(230327.OP-TEE.yaml)`
  - `[23:57]` Does OP-TEE support to verify a signature of a TA in X.509 certificate? `(OP-TEE.0327.md)`

Q & A

OP-TEE `(230410.Arm.yaml)`

Question: `[10:50]` Does OP-TEE support ARMv7 and ARMv8? `(OP-TEE.0327.md)`

Answer:

Yes, OP-TEE supports both ARMv7 and ARMv8 architectures. OP-TEE is designed to work with Arm TrustZone technology, which is available on both ARMv7 and ARMv8-based processors. Arm TrustZone provides hardware-based isolation between the normal world and the secure world, allowing OP-TEE to implement a Trusted Execution Environment (TEE) that can run trusted applications securely.

While ARMv7 and ARMv8 have some differences in terms of instruction sets and features, OP-TEE is compatible with both architectures and can be used to build secure systems on devices using either ARMv7 or ARMv8 processors.

Boot `(230329.TrustZone.yaml)`

Question: `[15:46]` What is ATF? `(TrustZone.0329.md)`

Answer:

ATF stands for ARM Trusted Firmware, which is an open-source project that provides a reference implementation of secure software for ARMv8-A-based platforms. The ARM Trusted Firmware provides a set of software components that are required to implement security features on ARMv8-A processors, such as those that support ARM TrustZone technology.

The main components of ARM Trusted Firmware include:

1. BL1 (Boot Loader 1): The initial boot loader responsible for setting up a secure execution environment and loading the next stage of the boot process, usually BL2 or BL31.
2. BL2 (Boot Loader 2): Responsible for loading and authenticating the remaining firmware images, such as BL31, BL32, and BL33. It typically performs additional platform-specific initialization and sets up memory partitioning.
3. BL31 (EL3 Runtime Software): The runtime firmware that runs at Exception Level 3 (EL3) and provides services for managing the system power state, system resets, and the handling of SMC (Secure Monitor Call) between the secure and non-secure worlds.
4. BL32 (Secure-EL1 Payload): Optional firmware that provides services to the secure world, such as a Secure Partition Manager (SPM) or a Trusted Execution Environment (TEE) like OP-TEE.
5. BL33 (Non-secure world bootloader): The bootloader for the non-secure world, typically a bootloader like U-Boot or an operating system like Linux.

The ARM Trusted Firmware provides a foundation for developing secure software on ARM-based platforms and serves as a reference implementation for platform vendors and developers to build upon or customize for their specific needs.

Basic Concepts `(230328.TEE.yaml)`

Question: `[15:48]` Is TEE a software? `(TEE.0328.md)`

Answer:

A Trusted Execution Environment (TEE) is not just a software; it is a combination of hardware, software, and firmware components that work together to provide a secure and isolated environment for executing sensitive operations. The TEE aims to protect sensitive data and code from unauthorized access, tampering, or leakage, even if the main operating system is compromised.

The main components of a TEE include:

1. Hardware: The hardware provides the necessary features to create an isolated environment that is separate from the main processor or the “Normal World.” This can include dedicated CPU modes, memory protection mechanisms, and secure storage. Examples of hardware-based TEE implementations are ARM TrustZone and Intel Software Guard Extensions (SGX).
2. Software: The TEE software includes a secure operating system (often called a “Trusted OS” or “Secure OS”) and a set of APIs that enable developers to create and interact with Trusted Applications (TAs). The software may also include additional security components, such as secure boot, cryptographic libraries, and attestation mechanisms. Examples of TEE software include OP-TEE for ARM TrustZone and Intel SGX SDK.
3. Firmware: Firmware components can also play a role in the TEE by providing low-level support for the hardware and software components. Firmware can be involved in the secure boot process, managing the lifecycle of the TEE, or providing cryptographic services.

In summary, a TEE is a combination of hardware, software, and firmware components designed to provide a secure environment for executing sensitive operations. It is not merely a software solution but an integrated system that leverages various layers of the computing stack to ensure security and isolation.

Question: `[23:45]` Does GlobalPlatform contains Chinese Cryptographic Standard? `(TEE.0328.md)`

Answer:

GlobalPlatform does not specifically include Chinese Cryptographic Standards as part of its core specifications. GlobalPlatform provides a standardized framework for implementing secure elements, TEEs (Trusted Execution Environments), and SE (Secure Element) management. It focuses on defining the APIs, protocols, and security requirements for these environments, but it does not mandate specific cryptographic algorithms.

However, GlobalPlatform's specifications are designed to be flexible and extensible, allowing individual TEE or secure element implementations to support various cryptographic algorithms, including Chinese Cryptographic Standards, as required by the specific use case or market requirements.

For example, a TEE implementation targeting the Chinese market may choose to integrate Chinese Cryptographic Standards such as:

1. SM2: An elliptic curve cryptography (ECC) standard for digital signatures, key exchange, and public key encryption.
2. SM3: A cryptographic hash function with a 256-bit digest.
3. SM4: A symmetric encryption algorithm with a 128-bit block size and a 128-bit key size.

These algorithms can be integrated into the TEE or secure element implementation, either as built-in functions or through third-party cryptographic libraries. As long as the implementation adheres to GlobalPlatform's specifications regarding the APIs, protocols, and security requirements, it can support a wide range of cryptographic algorithms, including Chinese Cryptographic Standards.

Development `(230327.OP-TEE.yaml)`

Question: `[19:29]` Does OP-TEE have a shell? `(OP-TEE.0327.md)`

Answer:

OP-TEE does not have a traditional shell like those found in Unix-like operating systems. However, OP-TEE provides a feature called the "Secure Shell" (S-Shell), which allows developers and users to interact with the secure world using a command-line interface. The S-Shell is primarily used for debugging purposes and is not intended for production use.

The S-Shell can be accessed using a UART (Universal Asynchronous Receiver/Transmitter) connection to the device running OP-TEE. This connection allows you to view log messages, interact with the secure world, and execute basic commands to manage trusted applications, memory, and other secure resources.

It is important to note that the S-Shell feature should be used carefully and only in controlled environments, as it could potentially expose sensitive information or allow unauthorized access to secure resources if misused.

In summary, while OP-TEE does not have a traditional shell, it does provide an S-Shell feature for debugging purposes. This feature should be used with caution, as it can potentially expose sensitive information or allow unauthorized access to secure resources if not properly managed.

Secure Storage `(230327.OP-TEE.yaml)`

Question: `[19:30]` Does OP-TEE support file system? `(OP-TEE.0327.md)`

Answer:

OP-TEE itself does not provide a full-fledged file system like those found in traditional operating systems. However, it does offer secure storage functionality that allows trusted applications to store and manage persistent data securely. The secure storage feature in OP-TEE is designed to protect sensitive data from unauthorized access and tampering.

The secure storage is implemented using a key-value store model, where each object (file) is associated with a unique identifier. Trusted applications can use the TEE Internal Core API to read, write, and delete objects in the secure storage. This API provides functions like `TEE_CreatePersistentObject`, `TEE_OpenPersistentObject`, `TEE_ReadObjectData`, `TEE_WriteObjectData`, and `TEE_CloseAndDeletePersistentObject`.

Under the hood, the secure storage in OP-TEE uses encryption and authentication techniques to protect the stored data. The actual storage is typically located in the normal world file system, but the data is encrypted and protected in such a way that it can only be accessed and manipulated by the intended trusted applications in the secure world.

In summary, OP-TEE does not have a traditional file system, but it does provide secure storage functionality for trusted applications to store and manage sensitive data securely.

Static TA and Dynamic TA `(230327.OP-TEE.yaml)`

Question: `[21:16]` Can I create a TA at run-time? `(OP-TEE.0327.md)`

Answer:

Creating a Trusted Application (TA) at runtime is not a typical use case for OP-TEE. TAs are usually developed, compiled, and signed offline before being deployed to the target device. In the normal process, TAs are loaded by the secure kernel (in the secure world) when requested by a client application (in the normal world).

However, if you have a specific use case that requires creating a TA at runtime, you could theoretically implement a custom mechanism for loading and running code in the secure world dynamically. This would involve significant modifications to the OP-TEE framework and would need to be done with extreme caution to ensure security is maintained.

Such a dynamic TA loading mechanism would likely require the following steps:

1. Develop and compile the TA code in a suitable format that can be loaded into the secure world at runtime.
2. Transfer the compiled TA code to the target device securely.
3. Modify the secure kernel to support loading and executing the TA code dynamically, including verifying its authenticity and integrity.
4. Implement proper access control and isolation mechanisms to ensure the newly loaded TA does not compromise the security of the existing TAs and the secure world.

It is crucial to emphasize that implementing a runtime TA creation mechanism is a complex and potentially risky endeavor, and it may introduce security vulnerabilities if not done correctly. You should carefully consider the security implications and whether the benefits of such a feature outweigh the risks before attempting to implement it.

Question: `[21:23]` But what is TEECI_LoadSecureModule() `(OP-TEE.0327.md)`

Answer:

My apologies for any confusion earlier. `TEECI_LoadSecureModule()` is not a part of the standard OP-TEE or GlobalPlatform TEE APIs. It seems to be a custom function name that might have been used in a specific implementation or context.

It's important to reiterate that the standard OP-TEE and GlobalPlatform TEE APIs do not provide a built-in function for loading or creating Trusted Applications (TAs) at runtime. TAs are generally pre-built, signed, and deployed to the device as binaries. The secure kernel then loads the TA when requested by a client application.

If you have encountered `TEECI_LoadSecureModule()` in some source code or documentation, it might be a custom function that has been implemented for a specific use case or project. To understand its purpose and how it works, you would need to refer to the documentation or source code of that particular implementation.

Question: `[23:20]` Does TEE define Static TA and Dynamic TA? `(OP-TEE.0327.md)`

Answer:

The terms "Static TA" and "Dynamic TA" are specific to the OP-TEE implementation and are not part of the GlobalPlatform TEE (Trusted Execution Environment) specifications. The GlobalPlatform TEE specifications define the APIs and security requirements for trusted applications, but they do not dictate how TAs should be loaded, stored, or managed within a TEE implementation.

The concepts of static and dynamic TAs are unique to the OP-TEE project and serve as a way to distinguish between different loading and linking mechanisms for trusted applications within the OP-TEE framework. Other TEE implementations

might use different terminology or approaches to manage trusted applications.

In summary, the terms “Static TA” and “Dynamic TA” are specific to OP-TEE and are not part of the GlobalPlatform TEE specifications.

Memory `(230329.TrustZone.yaml)`

Question: `[21:58]` What is MMU? `(TrustZone.0329.md)`

Answer:

MMU stands for Memory Management Unit. It is a hardware component found in many computer systems, including those based on the ARM architecture. The primary function of the MMU is to manage the system’s memory by translating virtual memory addresses to physical memory addresses, enabling memory protection, and controlling access permissions for different regions of memory.

Key functions of an MMU include:

1. Virtual-to-physical address translation: MMUs use a system of page tables to translate virtual memory addresses, used by software, to physical memory addresses, which correspond to the actual location of data in the memory hardware. This allows software to access memory in a uniform and consistent manner, even when the actual memory layout is fragmented or changes dynamically.
2. Memory protection: MMUs enable separation and isolation of different memory regions based on the access permissions set for each region. This helps prevent unauthorized access to memory and protects the system from bugs, crashes, or security vulnerabilities that could result from unintended memory access.
3. Caching and cache management: MMUs often work closely with the CPU’s cache to ensure efficient and consistent access to memory. By managing cache attributes and maintaining a consistent view of memory, MMUs help improve overall system performance.
4. Supporting virtual memory: MMUs play a crucial role in implementing virtual memory, which allows multiple applications and processes to run simultaneously while maintaining the illusion that each has access to a large, contiguous block of memory. Virtual memory allows the system to use disk space as an extension of RAM, effectively increasing the available memory and enabling better multitasking and memory management.

Overall, the MMU is an essential component in modern computer systems that helps manage memory efficiently, enhances system performance, and provides a foundation for implementing advanced memory-related features such as virtual memory and memory protection.

Question: `[22:01]` What is TLB? `(TrustZone.0329.md)`

Answer:

TLB stands for Translation Lookaside Buffer. It is a specialized cache used by a computer’s memory management unit (MMU) to speed up the translation of virtual memory addresses to physical memory addresses. The TLB stores a small number of recently-used virtual-to-physical address mappings, allowing the MMU to quickly look up the physical address for a given virtual address without having to consult the full page table stored in main memory.

Since accessing the main memory is generally slower than accessing the cache, the TLB helps improve the overall performance of the system. When the MMU receives a virtual address to translate, it first checks the TLB to see if the required mapping is already present. If the mapping is found in the TLB (a TLB hit), the MMU can quickly obtain the corresponding physical address. If the mapping is not in the TLB (a TLB miss), the MMU must access the page table in main memory, which takes more time.

To maintain the efficiency of the TLB, the system uses various algorithms to decide which mappings should be stored in the TLB and which ones should be evicted when the TLB becomes full. These algorithms aim to optimize the TLB hit rate by keeping the most frequently or recently used mappings in the buffer.

In summary, the TLB is a critical hardware component that works with the MMU to accelerate virtual-to-physical address translation, thus improving overall system performance and reducing memory access latency.

CA & TA `(230328.TEE.yaml)`

Question: `[23:16]` What are kinds of TA? `(TEE.0328.md)`

Answer:

Trusted Applications (TAs) can be broadly classified into two main categories based on their purpose and level of trust:

1. Rich Trusted Applications (RTAs): Rich Trusted Applications are TAs that provide more complex services and often interact with external components, such as peripherals, network services, or other TAs. RTAs are typically used for high-level security services like data encryption, digital rights management (DRM), secure payment processing, or secure authentication. These applications might run in a more feature-rich environment with support for a wider range of APIs, libraries, and services. RTAs generally have higher privileges and access to more resources than LTAs.
2. Lightweight Trusted Applications (LTAs): Lightweight Trusted Applications are smaller, more focused TAs that provide specific security services. These applications usually have limited functionality and are designed for low-latency, low-power, or resource-constrained environments. LTAs typically have a smaller Trusted Computing Base (TCB), which reduces the attack surface and increases the security of the TA. Examples of LTAs include secure key storage, cryptographic operations, and basic device attestation.

Additionally, TAs can be classified based on their development model and level of trust:

1. Vendor Trusted Applications: These are TAs developed by the TEE vendor or the device manufacturer. They are often pre-installed on the device and have the highest level of trust. Vendor TAs are typically responsible for managing the TEE, providing basic security services, or implementing device-specific features.
2. Third-Party Trusted Applications: These are TAs developed by third-party developers, such as independent software vendors (ISVs) or enterprise developers. Third-party TAs usually provide additional security services or features that are not covered by the vendor TAs. The trust level of third-party TAs depends on the authentication and attestation mechanisms used by the TEE and the device.

In summary, TAs can be classified based on their purpose, complexity, level of trust, and development model. The specific types of TAs supported by a TEE may vary depending on the TEE implementation, the underlying hardware, and the security requirements of the device.

CPU `(230329.TrustZone.yaml)`

Question: `[23:31]` What is SMP? `(TrustZone.0329.md)`

Answer:

SMP stands for Symmetric Multiprocessing. It is an architecture used in multi-core and multi-processor computer systems where two or more identical processors or processor cores are connected to a single shared memory and are controlled by a single operating system. In an SMP system, each processor or core can execute different tasks independently, but they share the same memory resources and I/O devices.

SMP allows for more efficient processing and better utilization of system resources, as multiple processors or cores can work simultaneously on different tasks, improving overall system performance. This architecture also provides better fault tolerance and load balancing, as tasks can be distributed among the available processors, reducing the workload on any single processor and minimizing the impact of a processor failure on the overall system.

In an SMP system, the operating system and applications must be designed to take advantage of multiple processors, using techniques such as multithreading and parallelism to distribute tasks among the available processors. Modern operating systems, such as Linux, Windows, and macOS, are capable of managing SMP systems and can automatically distribute workloads across multiple processors or cores.

Question: `[23:41]` What are difference between primary and seconday CPU cores? `(TrustZone.0329.md)`

Answer:

In the context of multi-core or multi-processor systems, primary and secondary CPU cores generally refer to the roles these cores play in the system, with the primary core being responsible for the initialization process and secondary cores assisting in processing tasks once the system is up and running.

Here are the main differences between primary and secondary CPU cores:

1. Initialization and Boot-up: The primary CPU core (sometimes called the boot core or master core) is responsible for initializing the system during boot-up. It executes the initial instructions, loads the firmware or BIOS, and configures the hardware components. Once the system is initialized, the primary core starts the operating system and enables other secondary cores.
2. Secondary Core Activation: Secondary cores (also called slave cores or worker cores) are enabled by the primary core after the system initialization process is complete. They start to execute tasks assigned by the primary core or the operating system.
3. Task Management: The primary core typically manages tasks and distributes them among secondary cores. However, modern operating systems can manage tasks independently and distribute them across all available cores (including the primary core) for efficient processing.
4. Power Management: In some systems, the primary core may have more advanced power management capabilities compared to secondary cores, enabling it to control the power states of secondary cores, such as turning them off during periods of low workload to save power.

It is essential to note that these differences apply primarily to systems where a distinction is made between primary and secondary cores. In modern symmetric multiprocessing (SMP) systems, all cores are often treated as equals once the system is up and running, and the operating system manages tasks and power states across all cores without any significant difference in their roles.

Question: `[23:42]` Please summarize the two types of CPU cores as a table. `(TrustZone.0329.md)`

Answer:

Aspect	Primary Core (Boot/Master Core)	Secondary Core (Slave/Worker Core)
Initialization	Responsible for system initialization during boot-up	Activated by the primary core after system initialization
Task Management	Manages and distributes tasks among secondary cores	Executes tasks assigned by the primary core or OS
Power Management	May have advanced power management capabilities	Power states controlled by the primary core or OS
Core Activation	Always active during boot-up and system operation	Activated and deactivated as needed for workload demands

Please note that in modern symmetric multiprocessing (SMP) systems, these distinctions become less relevant, as all cores are treated as equals once the system is up and running, and the operating system manages tasks and power states across all cores without any significant difference in their roles.

Boot `(230327.OP-TEE.yaml)`

Question: `[23:36]` Why does OP-TEE's reset function have two branches reset_primary and reset_secondary? `(OP-TEE.0327.md)`

Answer:

In OP-TEE, the reset function has two branches, `reset_primary` and `reset_secondary`, to handle the initialization process for multi-core systems. These branches are responsible for initializing the primary and secondary CPU cores, respectively, within the OP-TEE secure world.

1. `reset_primary`: This branch handles the initialization of the primary CPU core (core 0). The primary core is responsible

for setting up the secure environment, initializing the secure kernel, and starting the other secondary cores. The primary core performs tasks such as configuring the MMU (Memory Management Unit), initializing the secure memory, setting up the interrupt controller, and more. Once the primary core has completed its initialization tasks, it starts executing the secure kernel.

2. `reset_secondary`: This branch handles the initialization of secondary CPU cores (core 1, core 2, etc.). The secondary cores also need to set up their own environment, such as configuring their local MMU and enabling their local interrupts, before they can start executing the secure kernel. The primary core signals the secondary cores to begin their initialization process after it has completed its own initialization.

The distinction between `reset_primary` and `reset_secondary` branches ensures that the primary core properly initializes the secure environment before the secondary cores start executing the secure kernel. This separation helps maintain a correct and secure boot process in multi-core systems, where multiple CPU cores need to work together within the secure world.

In summary, OP-TEE's reset function has two branches, `reset_primary` and `reset_secondary`, to handle the initialization process for multi-core systems, ensuring that the primary core sets up the secure environment correctly before the secondary cores start executing the secure kernel.

Signature `(230327.OP-TEE.yaml)`

Question: `[23:57]` Does OP-TEE support to verify a signature of a TA in X.509 certificate?`(OP-TEE.0327.md)`

Answer:

OP-TEE supports verifying the signature of a Trusted Application (TA) using X.509 certificates. This verification process is essential to ensure the authenticity and integrity of the TA before loading it into the secure world.

By default, OP-TEE uses a custom file format for TA binaries, which includes a signature generated with a private key. The signature is verified using the corresponding public key during the TA loading process. This public key is typically embedded into the OP-TEE core during the build process.

If you want to use X.509 certificates for verifying TA signatures, you can follow these steps:

1. Build the OP-TEE project with the `CFG_TA_ASN1_DER_FORMAT` configuration option enabled. This option tells OP-TEE to expect TAs to be signed with X.509 certificates instead of the default custom format.

```
make CFG_TA_ASN1_DER_FORMAT=y
```

2. Sign your TA using an X.509 certificate. You can use tools like OpenSSL to create a certificate and sign the TA binary.
3. Embed the root certificate (or a certificate chain) used to sign the TA into the OP-TEE core during the build process. This certificate is used to verify the TA signature at runtime.
4. When loading a TA, OP-TEE will use the embedded certificate(s) to verify the TA's signature in the X.509 certificate format.

Please note that you'll need to modify your build process and potentially some OP-TEE components to support this feature. Make sure to consult the OP-TEE documentation and source code for more information on how to enable and use X.509 certificates for TA signature verification.
