2023-03-31

Created: 2023-04-22 01:03:21

Questions

- CA & TA @ TEE.0328.md
  - `[08:36]` Can CA and TA be implemented in other languages?
  - `[08:46]` How does TA authenticate request from CA?
- Development @ TEE.0328.md
  - `[08:36]` Can CA and TA be implemented in other languages?
- Boot @ TEE.0328.md
  - `[08:46]` How does TA authenticate request from CA?
- Intel KGT & Intel SGX @ OP-TEE.Intel.0330.md
  - `[09:06]` Which isolation method is safe?
- Isolation @ OP-TEE.Intel.0330.md
  - `[09:06]` Which isolation method is safe?
- Development @ OP-TEE.0327.md
  - `[09:40]` Which languages do I use to develop apps in OP-TEE or to customize OP-TEE?
  - `[09:40]` Which languages do I use to develop apps in OP-TEE or to customize OP-TEE?
  - `[09:40]` Can I use other langauges?
  - `[09:40]` Can I use other langauges?
- Basic Concepts @ TrustZone.0329.md
  - `[16:11]` Is TrustZone a software or hardware?

Chats

- TrustZone.0329.md
  - `[16:22]` Can TrustZone replace HSM?
- OP-TEE.Automative.0331.md
  - `[16:59]` Joakim Bech | Wednesday, September 3, 2014 | 6 mins read … …

Q & A

CA & TA @ TEE.0328.md

Question: `[08:36]` Can CA and TA be implemented in other languages?

Answer:

Yes, both the Client Application (CA) and the Trusted Application (TA) can be implemented in other programming languages, as long as the language supports interfacing with the required TEE APIs. Many TEE implementations provide APIs and language bindings for multiple programming languages.

For instance, the GlobalPlatform TEE Client API (for the CA) is generally available as a C API, but you can create language bindings for other languages like Python, Java, or Rust. You might need to use the Foreign Function Interface (FFI) or a similar mechanism to interface with the C API from another language.

As for the Trusted Application (TA), the choice of programming languages may be more limited, depending on the TEE implementation and the underlying hardware. Some TAs are typically written in C or C++ for performance and low-level access to hardware features, but there are implementations that support other languages like Rust.

For example, the Rust programming language has gained traction in TEE development, as it provides strong safety guarantees and is suitable for systems programming. There are projects like Teaclave SGX SDK (https://github.com/apache/teaclave-sgx-sdk) for Intel SGX and Rust OP-TEE TrustZone SDK (https://github.com/OP-TEE/optee-rust) for ARM TrustZone, which enable developers to write TAs in Rust.

In summary, while the choice of programming languages for CA and TA might be determined by the specific TEE implementation and hardware, it is possible to use languages other than C or C++ for both normal world and secure world applications.

Question: `[08:46]` How does TA authenticate request from CA?

Answer:

Trusted Applications (TAs) can authenticate requests from Client Applications (CAs) using a variety of mechanisms. One common approach is to establish a secure session between the CA and the TA, which can involve authentication, key exchange, and secure communication. This ensures that only authorized CAs can access the services provided by the TA.

Here are some methods that can be used to authenticate requests from a CA to a TA:

1. Login Methods: Different TEE implementations support various login methods to authenticate the origin of the request. GlobalPlatform TEE, for instance, supports multiple login types such as `TEEC_LOGIN_PUBLIC`, `TEEC_LOGIN_USER`, `TEEC_LOGIN_GROUP`, and `TEEC_LOGIN_APPLICATION`. These login types define different levels of trust and can be used to authenticate the CA based on the user, group, or application identity.
2. Client Authentication: When a CA opens a session with a TA, it can provide authentication data (e.g., a certificate, a pre-shared key, or a signature) as part of the session's parameters. The TA can verify this authentication data to ensure the CA is authorized to access its services. This can be implemented using custom or standardized cryptographic protocols.
3. Secure Channels: The CA and TA can establish a secure communication channel, which provides message confidentiality, integrity, and optionally, replay protection. This can be done by performing a key exchange and using shared secrets to encrypt and authenticate the communication between the CA and the TA.
4. Attestation: In some cases, the CA and/or the TA may require additional trust guarantees from the TEE or the platform. This can be achieved through attestation mechanisms, where a trusted third party (e.g., the TEE vendor or the platform owner) vouches for the authenticity and integrity of the TEE, the TA, or the CA. Examples of attestation mechanisms include Intel SGX's remote attestation and ARM TrustZone's secure boot.

The specific method of authentication and the level of trust required between the CA and the TA depend on the use case, the TEE implementation, and the underlying hardware. To ensure the highest level of security, it is essential to choose appropriate authentication mechanisms and follow best practices in implementing them.

Development @ TEE.0328.md

Question: `[08:36]` Can CA and TA be implemented in other languages?

Answer:

Yes, both the Client Application (CA) and the Trusted Application (TA) can be implemented in other programming languages, as long as the language supports interfacing with the required TEE APIs. Many TEE implementations provide APIs and language bindings for multiple programming languages.

For instance, the GlobalPlatform TEE Client API (for the CA) is generally available as a C API, but you can create language bindings for other languages like Python, Java, or Rust. You might need to use the Foreign Function Interface (FFI) or a similar mechanism to interface with the C API from another language.

As for the Trusted Application (TA), the choice of programming languages may be more limited, depending on the TEE implementation and the underlying hardware. Some TAs are typically written in C or C++ for performance and low-level access to hardware features, but there are implementations that support other languages like Rust.

For example, the Rust programming language has gained traction in TEE development, as it provides strong safety guarantees and is suitable for systems programming. There are projects like Teaclave SGX SDK

(https://github.com/apache/teaclave-sgx-sdk) for Intel SGX and Rust OP-TEE TrustZone SDK (https://github.com/OP-TEE/optee-rust) for ARM TrustZone, which enable developers to write TAs in Rust.

In summary, while the choice of programming languages for CA and TA might be determined by the specific TEE implementation and hardware, it is possible to use languages other than C or C++ for both normal world and secure world applications.

Boot @ TEE.0328.md

Question: `[08:46]` How does TA authenticate request from CA?

Answer:

Trusted Applications (TAs) can authenticate requests from Client Applications (CAs) using a variety of mechanisms. One common approach is to establish a secure session between the CA and the TA, which can involve authentication, key exchange, and secure communication. This ensures that only authorized CAs can access the services provided by the TA.

Here are some methods that can be used to authenticate requests from a CA to a TA:

1. Login Methods: Different TEE implementations support various login methods to authenticate the origin of the request. GlobalPlatform TEE, for instance, supports multiple login types such as `TEEC_LOGIN_PUBLIC`, `TEEC_LOGIN_USER`, `TEEC_LOGIN_GROUP`, and `TEEC_LOGIN_APPLICATION`. These login types define different levels of trust and can be used to authenticate the CA based on the user, group, or application identity.
2. Client Authentication: When a CA opens a session with a TA, it can provide authentication data (e.g., a certificate, a pre-shared key, or a signature) as part of the session's parameters. The TA can verify this authentication data to ensure the CA is authorized to access its services. This can be implemented using custom or standardized cryptographic protocols.
3. Secure Channels: The CA and TA can establish a secure communication channel, which provides message confidentiality, integrity, and optionally, replay protection. This can be done by performing a key exchange and using shared secrets to encrypt and authenticate the communication between the CA and the TA.
4. Attestation: In some cases, the CA and/or the TA may require additional trust guarantees from the TEE or the platform. This can be achieved through attestation mechanisms, where a trusted third party (e.g., the TEE vendor or the platform owner) vouches for the authenticity and integrity of the TEE, the TA, or the CA. Examples of attestation mechanisms include Intel SGX's remote attestation and ARM TrustZone's secure boot.

The specific method of authentication and the level of trust required between the CA and the TA depend on the use case, the TEE implementation, and the underlying hardware. To ensure the highest level of security, it is essential to choose appropriate authentication mechanisms and follow best practices in implementing them.

Intel KGT & Intel SGX @ OP-TEE.Intel.0330.md

Question: `[09:06]` Which isolation method is safe?

Answer:

Both Intel Kernel Guard Technology (KGT) and Intel Software Guard Extensions (SGX) provide strong isolation and protection mechanisms, but their safety depends on the specific use case and the implementation.

KGT provides isolation through virtualization technology, using a hypervisor to create isolated virtual machines (VMs). This can provide a strong level of isolation, but it also introduces additional attack surfaces and potential vulnerabilities in the hypervisor layer. The safety of KGT depends on the security of the hypervisor and the underlying hardware, as well as the security of the operating system and applications running within the VMs.

SGX, on the other hand, provides isolation through secure enclaves, which are isolated regions of memory within a process that are protected from access by other processes, the operating system, and even the CPU itself. This provides a strong level of isolation for sensitive data and code, but it also requires careful attention to enclave design and implementation to ensure that potential attack vectors are minimized. The safety of SGX depends on the security of the enclave code, the cryptographic keys used for encryption, and the hardware features that protect against tampering and side-channel attacks.

Overall, both KGT and SGX can provide strong isolation and protection, but their safety depends on the specific use case and the implementation. It is important to carefully evaluate the security and performance implications of using either technology and implement appropriate security measures to mitigate any potential risks.

Isolation @ OP-TEE.Intel.0330.md

Question: `[09:06]` Which isolation method is safe?

Answer:

Both Intel Kernel Guard Technology (KGT) and Intel Software Guard Extensions (SGX) provide strong isolation and protection mechanisms, but their safety depends on the specific use case and the implementation.

KGT provides isolation through virtualization technology, using a hypervisor to create isolated virtual machines (VMs). This can provide a strong level of isolation, but it also introduces additional attack surfaces and potential vulnerabilities in the hypervisor layer. The safety of KGT depends on the security of the hypervisor and the underlying hardware, as well as the security of the operating system and applications running within the VMs.

SGX, on the other hand, provides isolation through secure enclaves, which are isolated regions of memory within a process that are protected from access by other processes, the operating system, and even the CPU itself. This provides a strong level of isolation for sensitive data and code, but it also requires careful attention to enclave design and implementation to ensure that potential attack vectors are minimized. The safety of SGX depends on the security of the enclave code, the cryptographic keys used for encryption, and the hardware features that protect against tampering and side-channel attacks.

Overall, both KGT and SGX can provide strong isolation and protection, but their safety depends on the specific use case and the implementation. It is important to carefully evaluate the security and performance implications of using either technology and implement appropriate security measures to mitigate any potential risks.

Development @ OP-TEE.0327.md

Question: `[09:40]` Which languages do I use to develop apps in OP-TEE or to customize OP-TEE?

Answer:

To develop trusted applications for OP-TEE or to customize the OP-TEE framework, you primarily work with the C programming language. Both the OP-TEE OS (the secure kernel) and trusted applications are typically written in C to ensure low-level control, performance, and compatibility with the Arm TrustZone architecture.

1. Trusted Applications: To develop a trusted application (TA) for OP-TEE, you write your application code in C, using the TEE Internal Core API as specified by GlobalPlatform. This API provides functions and types for managing memory, cryptography, secure storage, and other operations within the trusted application.
2. Customizing OP-TEE: If you want to modify or extend the OP-TEE OS itself, you will also work with C. The OP-TEE OS codebase is written in C and can be customized to meet specific requirements or to add new features. However, customizing the OP-TEE OS requires a deeper understanding of the secure kernel's internals and the Arm TrustZone technology.

For normal world applications that communicate with the trusted applications, you can use various programming languages, as long as they can interface with the TEE Client API, which is usually provided as a C library. This means that you can use C, C++, or other languages that support C-compatible FFI (Foreign Function Interface) to interact with the secure world.

To summarize, you primarily use the C programming language to develop trusted applications and customize the OP-TEE framework, while normal world applications can be developed using various programming languages, as long as they can interface with the TEE Client API.

Question: `[09:40]` Which languages do I use to develop apps in OP-TEE or to customize OP-TEE?

Answer:

To develop trusted applications for OP-TEE or to customize the OP-TEE framework, you primarily work with the C

programming language. Both the OP-TEE OS (the secure kernel) and trusted applications are typically written in C to ensure low-level control, performance, and compatibility with the Arm TrustZone architecture.

1. Trusted Applications: To develop a trusted application (TA) for OP-TEE, you write your application code in C, using the TEE Internal Core API as specified by GlobalPlatform. This API provides functions and types for managing memory, cryptography, secure storage, and other operations within the trusted application.
2. Customizing OP-TEE: If you want to modify or extend the OP-TEE OS itself, you will also work with C. The OP-TEE OS codebase is written in C and can be customized to meet specific requirements or to add new features. However, customizing the OP-TEE OS requires a deeper understanding of the secure kernel's internals and the Arm TrustZone technology.

For normal world applications that communicate with the trusted applications, you can use various programming languages, as long as they can interface with the TEE Client API, which is usually provided as a C library. This means that you can use C, C++, or other languages that support C-compatible FFI (Foreign Function Interface) to interact with the secure world.

To summarize, you primarily use the C programming language to develop trusted applications and customize the OP-TEE framework, while normal world applications can be developed using various programming languages, as long as they can interface with the TEE Client API.

Question: `[09:40]` Can I use other langauges?

Answer:

While the primary language for developing trusted applications (TAs) and customizing OP-TEE is C, it is possible to use other languages in some scenarios. However, this usually involves additional steps, and you may need to consider the limitations and compatibility of the chosen language with the TEE environment.

For normal world applications, you have more flexibility in choosing a programming language, as long as it can interface with the TEE Client API. Many popular programming languages have Foreign Function Interface (FFI) capabilities, which allow them to call C functions from native libraries. Some examples of such languages include:

- C++
- Python (using `ctypes` or `cffi`)
- Rust (using `bindgen` or `ctypes`)
- Go (using `cgo`)
- Node.js (using `ffi-napi` or `node-ffi`)

For trusted applications, using a language other than C is more challenging, as it must meet certain requirements, such as being compatible with the TEE environment, the TEE Internal Core API, and the constraints of the secure world. Some potential options include:

1. C++: You may be able to use a subset of C++ to develop trusted applications, as long as you handle low-level details and ensure compatibility with the TEE environment. However, you would need to be cautious about using certain C++ features, such as exceptions and the standard library, which may not be supported or might introduce security risks.
2. Rust: Rust is a systems programming language with strong safety guarantees, which could be suitable for developing trusted applications. There has been some work on using Rust for OP-TEE trusted applications, such as the `rust-optee-trustzone-sdk` project (https://github.com/apache/incubator-teaclave-trustzone-sdk). This project provides a Rust SDK for creating TAs and a set of bindings to the TEE Internal Core API.

Using other high-level languages like Python, Java, or JavaScript for trusted applications is generally not feasible due to their runtime environments, memory management, and performance characteristics, which may not be compatible with the constraints of the secure world.

In summary, while it is possible to use other languages for normal world applications and, to some extent, for trusted applications, the primary language for OP-TEE development remains C. When considering an alternative language, be sure to evaluate its compatibility with the TEE environment, the API, and the specific requirements of your application.

Question: `[09:40]` Can I use other langauges?

Answer:

While the primary language for developing trusted applications (TAs) and customizing OP-TEE is C, it is possible to use other languages in some scenarios. However, this usually involves additional steps, and you may need to consider the limitations and compatibility of the chosen language with the TEE environment.

For normal world applications, you have more flexibility in choosing a programming language, as long as it can interface with the TEE Client API. Many popular programming languages have Foreign Function Interface (FFI) capabilities, which allow them to call C functions from native libraries. Some examples of such languages include:

- C++
- Python (using `ctypes` or `cffi`)
- Rust (using `bindgen` or `ctypes`)
- Go (using `cgo`)
- Node.js (using `ffi-napi` or `node-ffi`)

For trusted applications, using a language other than C is more challenging, as it must meet certain requirements, such as being compatible with the TEE environment, the TEE Internal Core API, and the constraints of the secure world. Some potential options include:

1. C++: You may be able to use a subset of C++ to develop trusted applications, as long as you handle low-level details and ensure compatibility with the TEE environment. However, you would need to be cautious about using certain C++ features, such as exceptions and the standard library, which may not be supported or might introduce security risks.
2. Rust: Rust is a systems programming language with strong safety guarantees, which could be suitable for developing trusted applications. There has been some work on using Rust for OP-TEE trusted applications, such as the `rust-optee-trustzone-sdk` project (https://github.com/apache/incubator-teaclave-trustzone-sdk). This project provides a Rust SDK for creating TAs and a set of bindings to the TEE Internal Core API.

Using other high-level languages like Python, Java, or JavaScript for trusted applications is generally not feasible due to their runtime environments, memory management, and performance characteristics, which may not be compatible with the constraints of the secure world.

In summary, while it is possible to use other languages for normal world applications and, to some extent, for trusted applications, the primary language for OP-TEE development remains C. When considering an alternative language, be sure to evaluate its compatibility with the TEE environment, the API, and the specific requirements of your application.

Basic Concepts @ TrustZone.0329.md

Question: `[16:11]` Is TrustZone a software or hardware?

Answer:

TrustZone is a hardware-based security technology implemented in ARM processors. It provides a security extension that enables the creation of a secure environment alongside the normal environment, isolating the execution of sensitive code and data from potential threats in the normal environment. TrustZone itself is not a software; however, it requires software support to fully utilize its features, such as secure operating systems, trusted applications, and secure bootloaders.

TrustZone.0329.md

Question: `[16:22]` Can TrustZone replace HSM?

Answer:

TrustZone and Hardware Security Modules (HSMs) serve different purposes and have different characteristics. While TrustZone can provide a secure environment for sensitive code execution and data storage, it may not be a direct replacement for HSMs in all use cases.

HSMs are dedicated hardware devices designed to provide robust security for cryptographic operations and key management. They are typically certified to high security standards (such as FIPS 140-2 or Common Criteria) and are

resistant to various physical and logical attacks.

TrustZone, on the other hand, is a security extension integrated into ARM processors, which aims to provide a secure environment within the same processor. It is not a separate dedicated hardware device like an HSM. TrustZone can offer a reasonable level of security but may not provide the same level of robustness or certification as an HSM.

In some use cases, TrustZone can be used as a lightweight alternative to an HSM, especially when cost, power consumption, and space are critical factors. However, for applications that require the highest levels of security and assurance, an HSM may still be the preferred choice.

Ultimately, whether TrustZone can replace an HSM depends on the specific security requirements and constraints of the system being designed.

OP-TEE.Automative.0331.md

Question: `[16:59]` Joakim Bech | Wednesday, September 3, 2014 | 6 mins read … …

Answer:

This article, written by Joakim Bech and published on September 3, 2014, discusses the Trusted Execution Environment (TEE), a small OS-like environment that operates alongside a rich operating system (e.g., Android) to protect secret credentials and data manipulation from malware and hackers. The concept of TEE was formalized around 2007 by the OMTP standardization forum, and the GlobalPlatform organization later defined standard APIs to support TEE functionality.

In 2009, ST began working on TEE as part of the ST-Ericsson mobile joint-venture, promoting the technology with its customers, including Nokia. The TEE implementation relied on Arm TrustZone technology for isolation from the rich OS. Keeping the TEE small and simple, the designers relied on the rich OS to schedule the TEE, treating Trusted Applications (TAs) as extensions of rich OS threads.

The goal was to make the implementation available to the entire industry to avoid fragmentation in mobile platform security implementations. In 2013, ST-Ericsson obtained GlobalPlatform compliance qualification for its implementation. Linaro, an organization focused on building and maintaining the Linux baseport on Arm, collaborated with ST to open-source the TEE, now called OP-TEE.

Linaro and ST engineers worked together to revamp the code base, make it portable, and remove any legacy or ST-specific code. OP-TEE is now available on GitHub and adheres to GlobalPlatform APIs. It is under a BSD license, allowing SoC vendors and device manufacturers to modify it without disclosing the modifications. This encourages the creation of an industry community around OP-TEE.

OP-TEE targets Arm cores and includes secure monitor code for TrustZone. It is expected that OP-TEE could be used on architectures other than Arm TrustZone, such as Cortex-M and Cortex-R range of Arm cores, to further reduce fragmentation in embedded electronics security. The release of OP-TEE aims to reduce fragmentation in the way Arm TrustZone is used, benefiting everyone, and fostering private and public applied research in security worldwide.
