Questions

- Intel KGT & Intel SGX
  - Be data in memory enclaves must encrypted in Intel SGX?
  - How are data encrypted in memory enclave in Intel SGX?
  - How do I attack Intel SGX?
- Isolation
  - What are isolation methods of Intel KGT and Intel SGX?
  - Which isolation method is safe?
- Boot
  - How to swtich secure world to normal world when booting?
  - How does TA authenticate request from CA?
- Development
  - Can CA and TA be implemented in other languages?
  - Which languages do I use to develop apps in OP-TEE or to customize OP-TEE?
  - Can I use other langauges?

Q & A

Intel KGT & Intel SGX

Question: Be data in memory enclaves must encrypted in Intel SGX?

Answer:

Yes, data in memory enclaves must be encrypted in Intel Software Guard Extensions (SGX) to provide protection against unauthorized access or tampering.

SGX provides a secure enclave within a process that is isolated from the rest of the process and the system. The enclave is created by the process and is encrypted using keys that are generated by the CPU. Data within the enclave is also encrypted using these keys, which means that even if an attacker gains access to the memory of the process, they cannot access or modify the data within the enclave.

In addition to encryption, SGX also provides other security features such as sealing and attestation. Sealing allows data to be protected and securely transported between enclaves, while attestation verifies the identity and integrity of the enclave to ensure that it has not been tampered with.

Overall, encryption is a critical component of SGX's security features and is essential for protecting data in memory enclaves from unauthorized access or tampering.

Question: How are data encrypted in memory enclave in Intel SGX?

Answer:

Data in memory enclaves in Intel Software Guard Extensions (SGX) is encrypted using a combination of hardware and software encryption mechanisms.

When an enclave is created, the CPU generates a unique key pair for that enclave, consisting of a public key and a private key. The private key is stored securely within the CPU and cannot be accessed or modified by software outside of the enclave. The public key is used to establish a secure channel between the enclave and the software outside of the enclave.

Once the secure channel is established, software within the enclave can use standard cryptographic algorithms, such as AES or RSA, to encrypt and decrypt data as needed. The CPU provides hardware support for these cryptographic algorithms, which means that they can be executed within the enclave with minimal performance overhead.

To protect against potential side-channel attacks, SGX also implements various security features, such as randomizing memory locations, encrypting the entire memory space of the enclave, and using secure hashing algorithms.

Overall, the combination of hardware and software encryption mechanisms, along with additional security features, provides strong protection for data in memory enclaves in SGX. However, it is still important to follow best practices for secure coding and system hardening to ensure that the system is protected against potential attacks.

Question: How do I attack Intel SGX?

Answer:

Attacking Intel Software Guard Extensions (SGX) is a complex task that requires advanced knowledge of the technology and the underlying hardware. While it is not impossible to attack SGX, doing so requires a significant amount of effort and resources. Here are some possible attack vectors that have been identified by security researchers:

1. Side-channel attacks: Side-channel attacks involve measuring physical characteristics of a system, such as power consumption or electromagnetic radiation, to extract sensitive information. SGX is vulnerable to some types of side-channel attacks, such as cache timing attacks and power analysis attacks.
2. Memory attacks: Memory attacks involve exploiting vulnerabilities in the memory management of a system to gain access to protected data. SGX is designed to protect against memory attacks, but there have been some vulnerabilities identified in the past, such as the Foreshadow attack.
3. Malware attacks: Malware attacks involve infecting a system with malicious software that can bypass SGX protections or steal sensitive data from within the enclave. SGX is not designed to protect against malware attacks, so it is important to ensure that the system is secure and free from malware.
4. Physical attacks: Physical attacks involve physically tampering with the system to gain access to sensitive information. This type of attack is difficult to execute and requires physical access to the system.

It is worth noting that Intel is continually improving SGX and addressing potential vulnerabilities as they are discovered. Regular software updates and security patches should be installed to ensure that the system is protected against known vulnerabilities. Additionally, best practices for secure coding, system hardening, and access control should be followed to reduce the risk of successful attacks.

Isolation

Question: What are isolation methods of Intel KGT and Intel SGX?

Answer:

Both Intel Kernel Guard Technology (KGT) and Intel Software Guard Extensions (SGX) provide isolation mechanisms to protect against unauthorized access and tampering. However, the specific methods of isolation are different for each technology.

Intel KGT provides isolation through virtualization technology, using a hypervisor to create virtual machines (VMs) that are isolated from each other and the host system. The hypervisor uses Intel VT hardware to enforce isolation between the VMs, ensuring that each VM has its own isolated memory space and access to only the resources that have been explicitly granted. This virtualization-based isolation is similar to the isolation provided by other hypervisors such as VMware or Hyper-V.

On the other hand, Intel SGX 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. The memory within an enclave is encrypted and can only be accessed by code that has been trusted and signed by the enclave owner. The enclave also includes hardware protections against tampering, such as sealing and attestation, which further enhance its security.

Overall, both Intel KGT and Intel SGX provide effective isolation mechanisms, but they are used in different contexts and have different strengths and weaknesses. Intel KGT is well-suited for creating secure and isolated virtualized environments, while Intel SGX is designed to protect against a wide range of attacks and provide strong protection for sensitive data and code.

Question: 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.

Boot

Question: How to swtich secure world to normal world when booting?

Answer:

When booting a system with TrustZone technology, the processor starts in the Secure World by default. To switch from the Secure World to the Normal World during the boot process, you can follow these general steps:

1. **Initialize the Secure World**: In the Secure World, the secure bootloader initializes the necessary hardware components, sets up the secure environment, and configures the memory and peripherals access permissions.
2. **Validate and load the Normal World software**: The secure bootloader validates the Normal World software, such as the operating system or bootloader, using cryptographic techniques like digital signatures. Once validation is successful, it loads the software into the memory designated for the Normal World.
3. **Configure the processor**: Before switching to the Normal World, the secure bootloader sets up the processor's state, registers, and exception levels. It also configures the system registers, such as the Secure Configuration Register (SCR) and the Monitor Vector Base Address Register (MVBAR), to define the behavior of the system when switching between the Secure World and the Normal World.
4. **Switch to the Normal World**: The secure bootloader performs a Secure Monitor Call (SMC) or an Exception Return (ERET) to switch the processor from the Secure World to the Normal World. This transition is facilitated by the Secure Monitor, which is responsible for handling the state switch between the two worlds.

After these steps, the processor enters the Normal World, and the Normal World software, such as the operating system or bootloader, starts executing.

Please note that these steps are a general outline and may vary depending on the specific platform, hardware, and firmware implementations.

Question: 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

Question: 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: 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: 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.