230418.SGX

Created: 2023-04-29 22:44:44

Questions

- Documents
  - Sealing and Attestation in Intel SGX `[title]` `(SGX.0418.md)`
  - What is relationship between secure enclave and secure memory? `(SGX.0418.md)`
  - Intel SGX Product Brief `(SGX.0418.md)`
- Paper - Innovative Technology for CPU Based Attestation and Sealing
  - Abstract and INTRODUCTION `[title]` `(Papers.SGX.0419.md)`
  - 2 MEASUREMENT `[title]` `(Papers.SGX.0419.md)`
  - 3 ATTESTATION `[title]` `(Papers.SGX.0419.md)`
  - 4 SEALING `[title]` `(Papers.SGX.0419.md)`
  - 5 RELATED WORK, 6 CONCLUSIONS, 7 ACKNOWLEDGMENTS `[title]` `(Papers.SGX.0419.md)`
  - 8 REFERENCES `[title]` `(Papers.SGX.0419.md)`
- Keys
  - What are types of Intel SGX Keys? `(SGX.0418.md)`
- Sealing Key
  - What is Sealing Key? `(Papers.SGX.0419.md)`
- Attestation Key
  - What is attestation key? `(SGX.0418.md)`
  - Is there only one attestation key in a device? `(SGX.0418.md)`
  - Where is attestation key sotred in Intel SGX arch? `[error]` `(SGX.0418.md)`
- EGETKEY
  - What is enclave software? `(SGX.0418.md)`
  - Is the EGETKEY instruction only called by enclave software? `(SGX.0418.md)`
  - Can I get hardware-protected key by calling EGETKEY instruction in enclave software? `(SGX.0418.md)`
  - So after I get the hardware-protected key, I can print it. Is it safe? `(SGX.0418.md)`
- Attestation
  - What is attestation? `(SGX.0418.md)`
  - Please build a cross table. The x axis is hardwared-based and software-based, The y axis is Remote and Local. Please contain Intel SGX, TPM, TLS, SSH, digital signatures, secure boot `(SGX.0418.md)`
  - Is secure boot a hardware or software attestation? `(SGX.0418.md)`
  - Do you mean that UEFI Secure Boot requires TPM? `(SGX.0418.md)`
- Enclave
  - What is enclave in Intel SGX? `(SGX.0418.md)`
  - Which key is encrypt memory encrypted by? `(SGX.0418.md)`
  - How does CPU run code in the encrypted memory in enclave? `(SGX.0418.md)`
  - Can untrusted software create an enclave? `[title]` `(Papers.SGX.0419.md)`
  - Can a program in an enclave see outside? `(SGX.0418.md)`
  - What are Intel provided enclaves? `(SGX.0418.md)`
- My Enclave
  - Can I create myself enclave with SIGSTRUCT signed by my RSA private key and run the enclave in Intel SGX environment? `(SGX.0418.md)`
  - How do I allow the platform owner to trust my enclave? `(SGX.0418.md)`
  - Where the trusted signer list reside in Intel SGX environment? `(SGX.0418.md)`
- MRENCLAVE
  - What is MRENCLAVE? `(Papers.SGX.0419.md)`
  - Who generates MRENCLAVE? `(Papers.SGX.0419.md)`

    - I have enclave A and enclave B. How does enclave A know the enclave B before A wants to get MRENCLAVE value of B? `(Papers.SGX.0419.md)`
- MRSIGNER
    - What is MRSIGNER? `(SGX.0418.md)`
    - Who does generate MRSIGNER? `(SGX.0418.md)`
    - So the MRSIGNER value is generated in offline not at runtime. Right? `(SGX.0418.md)`
    - Where is the private key stored? `(SGX.0418.md)`
    - What is enclave identity described in the paper? `[error]` `(Papers.SGX.0419.md)`
    - Is MRSIGNER unique to every enclave? `[title]` `(SGX.0418.md)`
    - Who can write MRENCLAVE and MRSIGNER? `(Papers.SGX.0419.md)`
    - Only hardware can write the both registers. Right? `(Papers.SGX.0419.md)`
    - Does MRSIGNER reside in SIGSTRUCT? `[error]` `(SGX.0418.md)`
    - What is Intel's hard-coded MRSIGNER? `(SGX.0418.md)`
    - What is the value of the hard-coded MRSIGNER provided by Intel? `(SGX.0418.md)`
- Sealing Authority
    - What is Sealing Authority? `(Papers.SGX.0419.md)`
    - Is Sealing Authority a enclave builder? `(Papers.SGX.0419.md)`
- SIGSTRUCT
    - What does SIGSTRUCT contain? please list them. `(Papers.SGX.0419.md)`
    - Who does prepare SIGSTRUCT? `(Papers.SGX.0419.md)`
    - Is a SIGSTRUCT a certificate? `(SGX.0418.md)`
    - Can I say that SIGSTRUCT contains enclave identity? `(SGX.0418.md)`
    - Please explain the sentence in the paper: "The enclave builder presents the hardware with an RSA signed enclave certificate (SIGSTRUCT)" and list the steps. `(Papers.SGX.0419.md)`
- Sealing Identity
    - What is Sealing Identity? `(Papers.SGX.0419.md)`
    - Is "Sealing Identity" a MRSIGNER? `(Papers.SGX.0419.md)`
    - Coud I say that SIGSTRUCT is Sealing Identity? `(Papers.SGX.0419.md)`
    - What is the value of Sealing Identity in the paper? `(Papers.SGX.0419.md)`
    - What is the value of Sealing Identity in the policy "Sealing to the Enclave Identity" in the paper? `(Papers.SGX.0419.md)`
    - What is difference between Sealing Identity and Identity of the Sealing Authority? `(Papers.SGX.0419.md)`
    - How does "Sealing to the Sealing Identity" work? Please list steps. `(Papers.SGX.0419.md)`
    - Please compare "Sealing to the Enclave Identity" and "Sealing to the Sealing Identity" in a table. `(Papers.SGX.0419.md)`
- Attestation Assertion
    - Who produces attestation assertion? `(Papers.SGX.0419.md)`
- EREPORT
    - What does REPORT contain? Please list them? `(Papers.SGX.0419.md)`
- EPID
    - What is EPID? `(SGX.0418.md)`
    - How does EPID work? `(SGX.0418.md)`
    - How is EPID keys enrolled into a device? `(SGX.0418.md)`
    - Which problems does EPID solve? `(SGX.0418.md)`
    - How do I make sure that the EPID issuer doesn't retain private keys? `(SGX.0418.md)`
    - How does EPID revoke a compromised device? `(SGX.0418.md)`
    - How to get the EPID of the compromised device? `(SGX.0418.md)`
    - But if the compromised device reports a fake EPID, what should I do? `(SGX.0418.md)`
    - Does Intel EPID require TPM? `(SGX.0418.md)`

Q & A

Documents

Question: Sealing and Attestation in Intel SGX [title] `(SGX.0418.md)`

Answer: … …

Question: What is relationship between secure enclave and secure memory? `(SGX.0418.md)`

Answer: … …

Question: Intel SGX Product Brief `(SGX.0418.md)`

Answer: … …

Paper - Innovative Technology for CPU Based Attestation and Sealing

Question: Abstract and INTRODUCTION [title] `(Papers.SGX.0419.md)`

Answer: … …

Question: 2 MEASUREMENT [title] `(Papers.SGX.0419.md)`

Answer: … …

Question: 3 ATTESTATION [title] `(Papers.SGX.0419.md)`

Answer: … …

Question: 4 SEALING [title] `(Papers.SGX.0419.md)`

Answer: … …

Question: 5 RELATED WORK, 6 CONCLUSIONS, 7 ACKNOWLEDGMENTS [title] `(Papers.SGX.0419.md)`

Answer: … …

Question: 8 REFERENCES [title] `(Papers.SGX.0419.md)`

Answer: … …

Keys

Question: What are types of Intel SGX Keys? `(SGX.0418.md)`

Answer:

Intel SGX provides several types of keys to facilitate secure operations within and around enclaves. The main types of Intel SGX keys are:

1. Seal Key: A key used to encrypt and authenticate data that will be stored outside the enclave, ensuring the confidentiality and integrity of the data. Seal keys are derived from a hardware root key unique to the processor and are not accessible

outside the enclave.

2. Provisioning Key: A key that is used during the provisioning process to establish a secure channel between the enclave and a provisioning service. This key is derived from a hardware root key and can only be generated by an enclave with the provisioning attribute set.
3. Provisioning Seal Key: A seal key used specifically for the provisioning process. It is derived from the same hardware root key as the provisioning key but is used to protect data used during provisioning.
4. Report Key: A key used to generate and verify enclave reports, which are data structures containing information about an enclave's identity and state. The report key is derived from a hardware root key and is accessible only within the enclave.
5. Attestation Key: A key used in the remote attestation process to sign enclave quotes, which are assertions about the enclave's identity and state. The attestation key is stored inside the enclave and is linked to an Intel Enhanced Privacy ID (EPID) or an ECDSA Attestation key.

These keys play different roles in ensuring the security and privacy of enclaves and their associated data. They are generated and managed by the SGX architecture in a way that provides strong isolation and protection against unauthorized access or tampering.

Sealing Key

Question: What is Sealing Key? `(Papers.SGX.0419.md)`

Answer:

A Sealing Key is a cryptographic key used in Intel SGX (Software Guard Extensions) to protect sensitive data in the context of an enclave. The purpose of the sealing key is to encrypt and decrypt (or “seal” and “unseal”) the data, ensuring that it remains confidential and can only be accessed by the enclave that generated the sealing key or other authorized enclaves.

Sealing keys are derived from a combination of the enclave's identity (MRENCLAVE or MRSIGNER) and the platform-specific key material. This ensures that the sealing keys are unique to a particular enclave and the platform it is running on.

In the Intel SGX architecture, sealing keys are used for the following purposes:

1. Confidentiality: Ensuring that the sensitive data remains confidential and is not accessible to unauthorized entities, including other enclaves, applications, or the operating system.
2. Integrity: Protecting the sensitive data from unauthorized modifications and ensuring that it remains consistent and unaltered.
3. Binding: Tying the sensitive data to a specific enclave or group of enclaves, ensuring that only authorized enclaves can access the data.

The sealing and unsealing processes are critical to the overall security of Intel SGX, as they ensure that sensitive data remains protected even when it is stored outside the secure enclave memory or transmitted between enclaves.

Attestation Key

Question: What is attestation key? `(SGX.0418.md)`

Answer:

An attestation key is a cryptographic key used in the process of attestation to prove the authenticity and integrity of a system, software, or hardware component. The attestation key is typically an asymmetric key pair, consisting of a private key and a public key.

In the context of secure enclaves or trusted execution environments like Intel SGX, the attestation key is used to sign data, such as measurements or claims about the secure environment, to provide a proof that the data is genuine and has not been tampered with. The private attestation key is kept securely within the enclave, while the public key is used by a verifier or remote party to validate the signature and ensure the integrity and authenticity of the data.

Attestation keys play a critical role in establishing trust in systems, as they provide a strong cryptographic guarantee that the information being attested to is genuine and originates from a trusted source.

Question: Is there only one attestation key in a device? `(SGX.0418.md)`

Answer:

In a device that supports hardware-based attestation, there can be multiple attestation keys depending on the specific technology being used and the device's intended purposes. For example, with Intel SGX, a device can have multiple enclaves, each with its own set of keys for secure operations.

In the case of TPM (Trusted Platform Module), a single device can have multiple attestation keys, each with a different purpose. For instance, there may be separate keys for platform configuration, software integrity, and user identity. These keys can be generated on-demand by the TPM and securely stored within the module.

In summary, a device can have more than one attestation key depending on the specific attestation technology being used and the device's security requirements.

Question: Where is attestation key sotred in Intel SGX arch? `[error]` `(SGX.0418.md)`

Answer:

In the Intel SGX architecture, the attestation key is not stored directly on the platform. Instead, the hardware-protected enclave generates an ephemeral attestation key pair during the attestation process.

The private attestation key is created and kept within the enclave, and it is not accessible by any other software or hardware outside the enclave. The public part of the attestation key pair is included in the attestation report, which is then sent to a remote party for verification.

The attestation process also involves the use of Intel's Enhanced Privacy ID (EPID), which is a group signature scheme that provides anonymity and revocability. The EPID private key is embedded in the processor during manufacturing and is securely stored inside the processor. The corresponding EPID group public key is maintained by Intel, allowing verification of EPID signatures.

To summarize, the attestation keys in Intel SGX are stored as follows:

1. Ephemeral attestation key pair: Generated within the enclave during the attestation process. The private key is kept inside the enclave, while the public key is included in the attestation report.
2. EPID private key: Embedded in the processor during manufacturing and securely stored inside the processor.

EGETKEY

Question: What is enclave software? `(SGX.0418.md)`

Answer:

Enclave software refers to a part of an application that runs inside a protected area of memory called an enclave. Enclaves are created using Intel Software Guard Extensions (Intel SGX) technology to provide a higher level of security for sensitive code and data. The enclave isolates the protected code and data from the rest of the application, the operating system, and other processes running on the same system.

Enclave software is designed to perform specific tasks or process sensitive information while ensuring confidentiality, integrity, and protection against unauthorized access or modification. This can be particularly useful for applications handling sensitive data, such as cryptographic keys, personal information, or proprietary algorithms. By running the sensitive part of the application within an enclave, developers can reduce the attack surface and provide an additional layer of security.

Question: Is the EGETKEY instruction only called by enclave software? `(SGX.0418.md)`

Answer:

Yes, the EGETKEY instruction is specific to Intel Software Guard Extensions (Intel SGX) and is used only by enclave software. EGETKEY is an instruction designed to derive a cryptographic key from a hardware-protected key hierarchy within an enclave. The key derived using EGETKEY is unique to each enclave and is based on the enclave's measurement and identity, providing an additional layer of security.

Enclave software uses the EGETKEY instruction to obtain a secret key for various cryptographic operations, such as encrypting or decrypting sensitive data or for generating a Message Authentication Code (MAC). By using EGETKEY, the key material is kept secure, and the risk of unauthorized access is minimized, since the key is not exposed outside of the enclave.

Only code running inside the enclave is authorized to execute the EGETKEY instruction, ensuring that the derived key remains protected and accessible only to the enclave software.

Question: Can I get hardware-protected key by calling EGETKEY instruction in enclave software?`(SGX.0418.md)`

Answer:

Yes, by calling the EGETKEY instruction in enclave software, you can obtain a hardware-protected key. This key is derived from a hardware-protected key hierarchy within an enclave, which is part of Intel Software Guard Extensions (Intel SGX).

The EGETKEY instruction generates a secret key unique to each enclave based on the enclave's measurement and identity. This provides an added layer of security, ensuring that the key material remains secure and minimizing the risk of unauthorized access. The derived key can then be used for various cryptographic operations, such as encrypting/decrypting sensitive data or generating a Message Authentication Code (MAC).

Keep in mind that only code running inside the enclave is authorized to execute the EGETKEY instruction. This restriction ensures that the derived key remains protected and is accessible only to the enclave software.

Question: So after I get the hardware-protected key, I can print it. Is it safe?`(SGX.0418.md)`

Answer:

Printing or logging the hardware-protected key would undermine its security. One of the primary purposes of using a hardware-protected key is to ensure that sensitive data remains secure even in the presence of compromised software or hardware. Exposing the key by printing or logging it outside of the enclave would make it vulnerable to theft and unauthorized access.

If you need to use the hardware-protected key outside of the enclave for some purpose, it is crucial to maintain the key's confidentiality. You should transmit the key securely, ideally using encryption or secure channels, and only share it with trusted parties. Printing or logging the key should be avoided to minimize the risk of exposure.

Attestation

Question: What is attestation? `(SGX.0418.md)`

Answer:

Attestation is a security process that allows one entity to verify the authenticity, integrity, and trustworthiness of another entity, typically in the context of software or hardware systems. In the realm of computer security, attestation involves checking that a system or application is running as intended, without unauthorized modifications or tampering.

Attestation can occur between different entities, such as:

1. Local attestation: This takes place between two entities on the same platform, such as two enclaves within the same system. They verify each other's trustworthiness before proceeding with a secure communication or data exchange.
2. Remote attestation: This occurs when an entity on one platform attests its trustworthiness to another entity on a different platform, usually over a network. In this case, the verifier, which could be a service provider or another system, checks the trustworthiness of the prover, which is the platform running the enclave or secure application.

Attestation often relies on cryptographic techniques, such as digital signatures, to ensure that the evidence provided during the attestation process is genuine and trustworthy. In the context of Intel SGX (Software Guard Extensions), attestation involves verifying the authenticity and integrity of an enclave to ensure that the software running inside it is trusted and has not been tampered with.

Question: Please build a cross table. The x axis is hardwared-based and software-based, The y axis is Remote and Local. Please contain Intel SGX, TPM, TLS, SSH, digital signatures, secure boot `(SGX.0418.md)`

Answer:

Here’s the cross table with the x-axis representing hardware-based and software-based attestation, and the y-axis representing remote and local attestation. Intel SGX, TPM, TLS, SSH, digital signatures, and secure boot are included:

```
+-------------------+----------------+-----------------+
|                   | Hardware-based | Software-based  |
+-------------------+----------------+-----------------+
| Local Attestation | Intel SGX      | Secure Boot     |
|                   | TPM            | Digital Signatures |
+-------------------+----------------+-----------------+
| Remote Attestation| Intel SGX      | TLS             |
|                   | TPM            | SSH             |
+-------------------+----------------+-----------------+
```

In this table, Intel SGX and TPM are listed under the “Hardware-based” column for both “Local Attestation” and “Remote Attestation.” Secure Boot and Digital Signatures are listed under the “Software-based” column for “Local Attestation.” TLS and SSH are listed under the “Software-based” column for “Remote Attestation.” This table provides an overview of the different types of attestation used by various technologies.

Question: Is secure boot a hardware or software attestation? `(SGX.0418.md)`

Answer:

Secure Boot can be considered a form of hardware-based attestation, as it relies on the hardware components of a system to ensure the integrity of the boot process. It leverages the capabilities of a Trusted Platform Module (TPM) or similar security hardware to verify the digital signatures of firmware, boot loaders, and operating system components. This ensures that only trusted software is executed during the boot process.

However, Secure Boot also involves software components, such as firmware, boot loaders, and digital signature verification routines. So, while it is primarily hardware-based attestation, it also relies on software to perform certain tasks and checks.

Question: Do you mean that UEFI Secure Boot requires TPM? `(SGX.0418.md)`

Answer:

I apologize for any confusion. UEFI Secure Boot does not strictly require a Trusted Platform Module (TPM). However, a TPM can be used in conjunction with UEFI Secure Boot to enhance security and provide additional functionality.

UEFI Secure Boot primarily focuses on verifying the digital signatures of firmware, boot loaders, and operating system components to ensure that only trusted and authorized software is executed during the boot process. This verification process can be carried out without the need for a TPM.

That said, a TPM can be used to securely store and manage cryptographic keys and attest to the integrity of the boot process, providing a hardware root of trust. When used in combination with UEFI Secure Boot, a TPM can enhance security by offering additional features, such as measuring the boot process, providing a secure way to store cryptographic keys, and supporting remote attestation.

In summary, while UEFI Secure Boot does not require a TPM, the two can be used together to provide a more secure boot environment.

Enclave

Question: What is enclave in Intel SGX? `(SGX.0418.md)`

Answer:

An enclave in the context of Intel Software Guard Extensions (SGX) is a protected region of memory within the application's address space. This region is isolated and secured from other processes running on the same system, including privileged processes and the operating system itself. The primary purpose of an enclave is to provide a secure environment for executing sensitive code and processing sensitive data.

When an application creates an enclave, the SGX hardware creates a secure container that encrypts the enclave's memory. This encrypted memory can only be accessed by the enclave's code, and even the operating system or hypervisor cannot directly access it. Enclaves also have their own isolated execution environment, which helps protect against side-channel attacks and other threats.

Developers can use Intel SGX to create enclaves for various purposes, such as secure key management, encrypted data storage, or privacy-preserving computation. Enclaves are useful for protecting sensitive code and data from potential threats, ensuring the confidentiality and integrity of the data being processed.

Question: Which key is encrypt memory encrypted by? `(SGX.0418.md)`

Answer:

In Intel SGX, the memory of an enclave is encrypted using a key called the Memory Encryption Engine Key (MEE-K). This key is hardware-generated and unique to each processor. The MEE-K is not accessible to software, ensuring the confidentiality and integrity of the enclave's memory.

Question: How does CPU run code in the encrypted memory in enclave? `(SGX.0418.md)`

Answer:

When the CPU executes code inside an enclave, it decrypts the memory contents on-the-fly using the Memory Encryption Engine (MEE) integrated into the CPU. This process is transparent to the software running inside the enclave.

When data is read from the enclave memory, the MEE decrypts it before passing it to the CPU for processing. When data is written back to the enclave memory, the MEE encrypts it before storing it. This encryption and decryption process is done in real-time and ensures that the code and data inside the enclave are protected from unauthorized access or tampering, even if an attacker has physical access to the system memory.

Question: Can untrusted software create an enclave? `[title]` `(Papers.SGX.0419.md)`

Answer:

Yes, even untrusted software can create an enclave. The primary purpose of Intel SGX is to allow secure execution of sensitive code and data within an enclave, even if the environment outside the enclave is not trusted. The untrusted software can create an enclave, but it cannot access or tamper with the code and data inside the enclave. Once the enclave is created and initialized, only the trusted code inside the enclave can execute and manipulate the data securely.

Question: Can a program in an enclave see outside? `(SGX.0418.md)`

Answer:

A program running inside an Intel SGX enclave cannot directly access memory outside the enclave. The enclave's memory is isolated from the rest of the system, including other processes and even the operating system. This isolation helps protect the confidentiality and integrity of the data and code within the enclave.

However, an enclave program can communicate with the outside world through well-defined interfaces. The enclave can expose secure entry points, called e-calls, that allow the application outside the enclave to call functions within the enclave. Similarly, the enclave can make o-calls to interact with the outside world, for example, to request external services or perform I/O operations.

These interactions must be carefully designed to avoid leaking sensitive information or exposing vulnerabilities. Data passed between the enclave and the external environment should be appropriately encrypted and authenticated to protect it from unauthorized access or tampering. Additionally, the enclave's exposed interfaces should be minimal and thoroughly reviewed to ensure their security.

Question: What are Intel provided enclaves? `(SGX.0418.md)`

Answer:

Intel provides some enclaves, known as Intel-provided enclaves, that offer specific functionality to support the Intel SGX ecosystem. These enclaves are designed to help developers implement security features and other services in their SGX applications more efficiently. Some of the Intel-provided enclaves include:

1. Quoting Enclave (QE): The Quoting Enclave is responsible for generating quotes as part of the SGX remote attestation process. It takes a report from another enclave, verifies it, and creates a quote that can be sent to a remote party for validation.

2. Provisioning Enclave (PvE): The Provisioning Enclave is responsible for managing the EPID (Enhanced Privacy ID) provisioning process. It helps in obtaining the EPID key and certificate for the platform from the Intel Attestation Service (IAS).

3. Platform Services Enclave (PSE): The Platform Services Enclave provides additional services and functionalities that may be required by other enclaves, such as monotonic counters, trusted time, and sealed storage.

4. Architectural Enclave Service Manager (AESM): While not an enclave itself, the AESM is a system service that manages the interaction between Intel-provided enclaves and the platform, helping to facilitate the attestation and provisioning processes.

These Intel-provided enclaves serve as a foundation for the Intel SGX ecosystem, assisting developers in creating secure applications and services on SGX-enabled platforms.

My Enclave

Question: Can I create myself enclave with SIGSTRUCT signed by my RSA private key and run the enclave in Intel SGX environment? `(SGX.0418.md)`

Answer:

Yes, you can create an enclave with a SIGSTRUCT signed by your own RSA private key and run the enclave in the Intel SGX environment. When you create an enclave, you'll need to generate a key pair (private and public keys) and use your private key to sign the enclave's SIGSTRUCT. The MRSIGNER value will then represent the hash of your public key.

However, running the enclave on a platform will require the platform owner or administrator to trust your enclave. Trust is established by either including your public key in the platform's trusted signer list or by having a trusted third party (such as Intel) vouch for your enclave through a certificate chain.

In summary, while it's technically possible to create an enclave signed by your own RSA private key, the enclave's trustworthiness and acceptance in the Intel SGX environment will depend on the platform owner or administrator's trust configuration.

Question: How do I allow the platform owner to trust my enclave? `(SGX.0418.md)`

Answer:

To allow the platform owner to trust your enclave, you can follow these steps:

1. Share your public key: After signing your enclave with your private key, share the corresponding public key with the platform owner.

2. Establish trustworthiness: The platform owner needs to verify that your enclave is trustworthy. This can be done through a code review, an audit from a trusted third party, or other means of establishing trust in your enclave's code and

functionality.

3. Add your public key to the trusted signer list: The platform owner can then add your public key to the trusted signer list on their platform. This list is used by the Intel SGX environment to determine which enclaves are trusted to run on the platform. The platform owner may maintain this list locally on the platform or use a remote attestation service to manage trusted keys.
4. Configure the platform: The platform owner may need to configure the Intel SGX environment to accept enclaves signed by non-Intel entities or update the trusted signer list to include your public key.
5. Remote attestation (optional): In some cases, you might want to use remote attestation to prove to a relying party that your enclave is running securely on the platform. To do this, you'll need to provide the remote attestation service with your enclave's measurement, along with any other required information, such as your enclave's MRSIGNER value. The attestation service can then verify the enclave's identity and provide an attestation report to the relying party.

By following these steps, you can establish trust with the platform owner and enable them to trust and run your enclave in their Intel SGX environment.

Question: Where the trusted signer list reside in Intel SGX environment? `(SGX.0418.md)`

Answer:

In the Intel SGX environment, the trusted signer list is maintained by the platform owner, and it is not explicitly part of the SGX hardware or SDK. The list can be managed in various ways, depending on the implementation chosen by the platform owner.

Some possible ways to manage the trusted signer list are:

1. Locally on the platform: The platform owner can maintain a list of trusted public keys within the local configuration of the Intel SGX environment. This list can be updated as needed, and the platform owner can decide which enclaves are allowed to run based on the MRSIGNER values of those enclaves.
2. Remote attestation service: The platform owner can use a remote attestation service, such as Intel's Attestation Service, to manage the list of trusted public keys. In this case, the attestation service verifies the identity of the enclave, and the platform owner can decide whether to trust the enclave based on the attestation report provided by the service.
3. Certificate-based trust: The platform owner may choose to trust enclaves signed by specific Certificate Authorities (CAs). In this case, the platform owner maintains a list of trusted CAs, and enclaves signed by those CAs are considered trustworthy.

Ultimately, the management of the trusted signer list is determined by the platform owner and their specific implementation of the Intel SGX environment.

MRENCLAVE

Question: What is MRENCLAVE? `(Papers.SGX.0419.md)`

Answer:

MRENCLAVE is a hash value that represents the identity and code of an enclave in Intel SGX (Software Guard Extensions). It is generated during the enclave's creation process and is based on the enclave's content, initial state, and memory layout.

The MRENCLAVE value is calculated as a hash over the enclave's code and data, including its initial register values, memory layout, and other attributes. This hash uniquely identifies the enclave, as even a small change in the code or data would result in a different MRENCLAVE value.

MRENCLAVE is used in various security mechanisms, such as attestation, sealing, and key management, to ensure that the enclave's integrity and authenticity are maintained. It is included in the REPORT data structure, which is used for local and remote attestation processes to prove the identity, integrity, and authenticity of an enclave running on an SGX-enabled platform.

Question: Who generates MRENCLAVE? `(Papers.SGX.0419.md)`

Answer:

MRENCLAVE is generated by the Intel SGX hardware during the enclave creation process. As the enclave is loaded into memory and initialized, the hardware computes the MRENCLAVE value by hashing the enclave's content, initial state, and memory layout. This ensures that the MRENCLAVE value uniquely represents the identity and integrity of the specific enclave.

Question: I have enclave A and enclave B. How does enclave A know the enclave B before A wants to get MRENCLAVE value of B? `(Papers.SGX.0419.md)`

Answer:

Before enclave A can get the MRENCLAVE value of enclave B, it needs some knowledge about enclave B. Typically, this knowledge is shared through some out-of-band mechanism or by the application that hosts both enclaves. Here's a high-level overview of the process:

1. The application creates enclave A and enclave B.
2. The application shares some information about enclave B with enclave A. This could be done by passing the targetinfo structure of enclave B to enclave A, which contains the MRENCLAVE value of enclave B.
3. Enclave A uses the shared information about enclave B to call the EREPORT instruction and generate a report targeting enclave B.
4. The report generated by enclave A is passed to enclave B, either by the application or through some inter-enclave communication mechanism.
5. Enclave B verifies the report's authenticity using the EGETKEY instruction to ensure it comes from enclave A. If the report is valid, enclave B can extract the MRENCLAVE value of enclave A from the report.

Keep in mind that the specific implementation details and communication methods may vary depending on the application's design and security requirements. The above steps are a high-level description of the process.

MRSIGNER

Question: What is MRSIGNER? `(SGX.0418.md)`

Answer:

MRSIGNER is a term used in the context of Intel SGX (Software Guard Extensions) to refer to the measurement of the signer of an enclave. It is a 256-bit hash value that represents the public key of the signer who created and signed the enclave. This value is stored within the enclave's metadata and is part of the enclave's identity.

MRSIGNER is used during the attestation process to ensure that the enclave was created by a trusted party. During attestation, the MRSIGNER value is checked against a list of trusted signer public keys to determine whether the enclave's creator is trusted. If the MRSIGNER matches one of the trusted public keys, the attestation can proceed, indicating that the enclave is genuine and created by a trusted party.

Question: Who does generate MRSIGNER? `(SGX.0418.md)`

Answer:

MRSIGNER is generated by the enclave's developer or creator. The developer creates the enclave code and then signs it using their private key. The MRSIGNER value is a hash of the public key that corresponds to the private key used for signing the enclave.

During the build process, the enclave's code, data, and metadata are combined, and a measurement of the enclave's contents (MRENCLAVE) is calculated. The developer then signs the measurement and other metadata with their private key, creating a signature. The corresponding public key is hashed to generate the MRSIGNER value, which is stored as part of the enclave's metadata.

When the enclave is loaded and initialized, the MRSIGNER value is checked during the attestation process to verify the enclave's origin and ensure that it was created by a trusted party.

Question: So the MRSIGNER value is generated in offline not at runtime. Right? `(SGX.0418.md)`

Answer:

Yes, that's correct. The MRSIGNER value is generated offline during the enclave build process, not at runtime. The enclave developer signs the enclave with their private key, and the corresponding public key is hashed to generate the MRSIGNER value. This value is then stored as part of the enclave's metadata.

During runtime, the attestation process uses the MRSIGNER value to verify the enclave's origin and ensure that it was created by a trusted party. The value is not generated or altered during runtime.

Question: Where is the private key stored? `(SGX.0418.md)`

Answer:

The private key used to sign an enclave is typically stored securely by the enclave developer or their organization. It is crucial to keep the private key confidential to prevent unauthorized access or modification of the enclave software.

In practice, private keys may be stored in various secure storage solutions, such as hardware security modules (HSMs), encrypted storage, or secure key management systems. The choice of storage depends on the specific security requirements and policies of the organization responsible for the enclave.

Question: What is enclave identity described in the paper? `[error]` `(Papers.SGX.0419.md)`

Answer:

In the paper, the term "enclave identity" refers to the unique identifier associated with an enclave. The enclave identity is a combination of the MRENCLAVE value, which represents the enclave's code and data, and the MRSIGNER value, which represents the enclave's signer or the entity that signed the enclave.

The enclave identity is used in various security policies, such as attestation and sealing. For example, when an enclave wants to attest its identity to a remote party, it provides a REPORT containing the MRENCLAVE and MRSIGNER values as part of the attestation process. Similarly, when sealing data, the sealing policy may be based on the enclave identity to ensure that the sealed data can only be accessed by specific enclaves with matching MRENCLAVE and/or MRSIGNER values.

In summary, the enclave identity in the paper is a combination of MRENCLAVE and MRSIGNER values that uniquely identify an enclave, and it is used in various security policies and operations related to attestation and sealing.

Question: Is MRSIGNER unique to every enclave? `[title]` `(SGX.0418.md)`

Answer:

MRSIGNER is not unique to every enclave; it is unique to the entity that signed the enclave. Multiple enclaves signed by the same entity (using the same public-private key pair) will have the same MRSIGNER value.

In other words, MRSIGNER is a way to identify the signer of an enclave rather than the enclave itself. When you verify an enclave's identity during the remote attestation process, you may use the MRSIGNER to check if the enclave was signed by a trusted entity.

For example, if a company produces multiple enclaves, they may sign all their enclaves with the same key pair. All these enclaves will have the same MRSIGNER, which represents the company's public key. In this case, MRSIGNER serves as an identifier for the company as the enclave signer.

Question: Who can write MRENCLAVE and MRSIGNER? `(Papers.SGX.0419.md)`

Answer:

MRENCLAVE and MRSIGNER are values that are generated and populated by the Intel SGX hardware during the enclave creation process.

1. MRENCLAVE: This value is a measurement (hash) of the enclave's code and data. It is generated by the hardware when the enclave is being built and loaded into memory. The MRENCLAVE value is a result of hashing the enclave's content during the build process, and it represents the identity of the specific enclave. No one writes MRENCLAVE directly; it is produced by the hardware as a part of the enclave creation process.
2. MRSIGNER: This value is a measurement (hash) of the public key of the Sealing Authority, which is responsible for signing the enclave's certificate. MRSIGNER is populated by the hardware when the enclave is loaded into memory and is derived from the enclave's signature. The Sealing Authority generates the key pair used for signing the enclave, and during the enclave loading process, the hardware calculates the MRSIGNER value from the public key in the enclave's signature. This value represents the identity of the Sealing Authority.

In summary, MRENCLAVE and MRSIGNER are not written by any individual or entity directly; instead, they are generated and populated by the Intel SGX hardware during the enclave creation and loading process.

Question: Only hardware can write the both registers. Right? `(Papers.SGX.0419.md)`

Answer:

Yes, that's correct. Only the Intel SGX hardware can write both MRENCLAVE and MRSIGNER registers. These values are generated and populated by the hardware during the enclave creation and loading process, ensuring that they are secure and trustworthy.

Question: Does MRSIGNER reside in SIGSTRUCT? `[error]` `(SGX.0418.md)`

Answer:

Yes, MRSIGNER resides in the SIGSTRUCT. MRSIGNER is the hash of the public key used to sign the enclave. The SIGSTRUCT is a data structure that contains the enclave's identity and integrity information, including the MRSIGNER value, MRENCLAVE (hash of the enclave's initial state), and the enclave's version, among other things. The SIGSTRUCT is used during the enclave's loading process to verify its authenticity and integrity.

Question: What is Intel's hard-coded MRSIGNER? `(SGX.0418.md)`

Answer:

Intel does not have a universal hard-coded MRSIGNER value applicable to all enclaves. However, Intel provides MRSIGNER values for their specific Intel-provided enclaves or other enclaves they develop. These MRSIGNER values are associated with Intel's signing key and are used to verify the authenticity of the enclaves. In the context of Intel SGX, MRSIGNER values are specific to the enclaves they represent and are unique to the entity or organization that signed the enclave.

If you are working with an Intel-provided enclave, you can find the associated MRSIGNER value in the enclave's documentation or other relevant materials. Keep in mind that MRSIGNER values are specific to individual enclaves, and the value you need to use will depend on the particular enclave you are working with.

Question: What is the value of the hard-coded MRSIGNER provided by Intel? `(SGX.0418.md)`

Answer:

As an AI language model, I do not have real-time access to the specific MRSIGNER values provided by Intel for their enclaves. However, I can provide you with some guidance on how to find the MRSIGNER value associated with a particular Intel-provided enclave.

If you are working with an Intel-provided enclave, you can find the associated MRSIGNER value in the enclave's documentation, SDK materials, or other relevant resources provided by Intel. The MRSIGNER value will be unique to the specific enclave you are working with, so it is important to consult the appropriate materials related to that enclave.

Additionally, you can find the MRSIGNER value programmatically by extracting it from the enclave's SIGSTRUCT structure when the enclave is loaded. Keep in mind that doing so requires a proper understanding of the enclave's layout and structure.

Sealing Authority

Question: What is Sealing Authority? `(Papers.SGX.0419.md)`

Answer:

The Sealing Authority is an entity responsible for signing and endorsing enclaves in the context of Intel SGX. It is typically the enclave developer or an organization that signs and vouches for the integrity and authenticity of the enclave software. The Sealing Authority signs enclaves using its private key, and the corresponding public key is embedded within the enclave to verify its authenticity.

When sealing data to a Sealing Authority Identity (Sealing Identity), the Sealing Key is based on the MRSIGNER value (the measurement of the enclave's signing authority) and the enclave's version. This policy allows newer enclaves signed by the same Sealing Authority to access data stored by previous versions, enabling data migration between enclave versions and ensuring that only trusted and authenticated enclaves can access sensitive data.

Question: Is Sealing Authority a enclave builder? `(Papers.SGX.0419.md)`

Answer:

Yes, the Sealing Authority can be considered as the enclave builder or the entity responsible for creating and signing enclaves. The Sealing Authority has the necessary cryptographic keys to sign the enclave's certificate, which in turn establishes trust in the enclave's identity and authenticity. The MRSIGNER value reflects the identity of the Sealing Authority, and it is used in sealing policies to determine which enclaves can access the same sealing keys, enabling them to transparently access data that was sealed by other enclaves under the authority's control.

SIGSTRUCT

Question: What does SIGSTRUCT contain? please list them. `(Papers.SGX.0419.md)`

Answer:

SIGSTRUCT is a data structure in Intel SGX that contains the enclave's signature, which is generated by the enclave author or builder during the enclave signing process. The SIGSTRUCT contains the following fields:

1. Header: A constant value identifying the SIGSTRUCT.
2. Vendor: Indicates whether the enclave is a debug or production enclave.
3. Date: The date when the enclave was created.
4. Header2: A constant value representing the second part of the header.
5. SWDefined: Software-defined field, reserved for future use.
6. Reserved1: Reserved field for padding.
7. Modulus: The RSA public key modulus (n) of the enclave signer.
8. Exponent: The RSA public key exponent (e) of the enclave signer.
9. Signature: The RSA signature of the enclave, generated using the enclave signer's private key.
10. MiscSelect: A set of flags indicating which additional information should be included in the enclave measurement.
11. MiscMask: A mask for the MiscSelect field.
12. Reserved2: Reserved field for padding.
13. Attributes: Flags describing various enclave properties.
14. AttributeMask: A mask for the Attributes field.
15. EnclaveHash: The MRENCLAVE value, which is the hash of the enclave's initial state.
16. Reserved3: Reserved field for padding.

17. ISVProdID: The Product ID of the enclave.
18. ISVSVN: The Security Version Number (SVN) of the enclave.
19. Reserved4: Reserved field for padding.
20. Q1, Q2: Two values used in the RSA signature generation process.

These fields are used by the Intel SGX hardware to verify the integrity and authenticity of the enclave during the enclave loading process.

Question: Who does prepare SIGSTRUCT? `(Papers.SGX.0419.md)`

Answer:

SIGSTRUCT is prepared by the enclave developer or the entity responsible for creating the enclave. This entity, sometimes called the Sealing Authority, generates the SIGSTRUCT as part of the enclave building process. They sign the enclave with their private key, which is then verified by the Intel SGX hardware using the public key (MRSIGNER) to ensure the enclave's authenticity and integrity.

Question: Is a SIGSTRUCT a certificate? `(SGX.0418.md)`

Answer:

A SIGSTRUCT can be thought of as a certificate-like structure in the Intel SGX ecosystem, but it is not a traditional X.509 certificate. The SIGSTRUCT is a data structure that contains information about an enclave, such as the MRENCLAVE (measurement of the enclave), MRSIGNER (measurement of the enclave signer), and other attributes. It also contains the enclave's signature, generated using the enclave author's private key.

The purpose of the SIGSTRUCT is to provide a signed description of an enclave, which can be used during the enclave's loading process. The Intel SGX-enabled platform uses the SIGSTRUCT to verify the integrity and authenticity of the enclave before allowing it to be executed. In this sense, it serves a similar purpose to a certificate, but its format and usage are specific to the Intel SGX environment.

Question: Can I say that SIGSTRUCT contains enclave identity? `(SGX.0418.md)`

Answer:

Yes, you can say that the SIGSTRUCT contains enclave identity, as it includes important attributes such as MRENCLAVE and MRSIGNER, which together contribute to the unique identity of an enclave. These attributes, along with other information in the SIGSTRUCT, are used to verify the integrity and authenticity of an enclave during the loading process.

Question: Please explain the sentence in the paper: "The enclave builder presents the hardware with an RSA signed enclave certificate (SIGSTRUCT)" and list the steps. `(Papers.SGX.0419.md)`

Answer:

The sentence "The enclave builder presents the hardware with an RSA signed enclave certificate (SIGSTRUCT)" describes a step in the process of creating, loading, and initializing an enclave in the Intel SGX architecture. Let's break down the steps involved:

1. Enclave Creation: The enclave developer writes the enclave code and prepares it for execution in a secure environment provided by the Intel SGX.

2. Enclave Signing: The enclave developer or a trusted authority signs the enclave using a private key, generating a SIGSTRUCT, which is an RSA-signed enclave certificate. The SIGSTRUCT contains important information about the enclave, such as the MRENCLAVE (measurement of the enclave code), MRSIGNER (measurement of the enclave signer), and various attributes.

3. Enclave Loading: The enclave builder prepares the enclave for loading into the SGX-enabled hardware. This includes setting up the memory layout, copying the enclave code into the secure enclave memory, and preparing any necessary data structures.

4. Presenting SIGSTRUCT to Hardware: In the step described by the sentence, the enclave builder presents the hardware with the SIGSTRUCT. This step is necessary for the hardware to verify the authenticity and integrity of the enclave.
5. Verification by Hardware: The Intel SGX hardware verifies the SIGSTRUCT by checking the RSA signature, the MRENCLAVE, the MRSIGNER, and other attributes. If the verification is successful, the hardware determines that the enclave is authentic and has not been tampered with.
6. Enclave Initialization: If the verification is successful, the hardware initializes the enclave, setting up the necessary runtime environment for the enclave code to execute securely.
7. Secure Execution: Once the enclave is initialized, it can begin secure execution within the protected environment provided by the Intel SGX architecture.

Sealing Identity

Question: What is Sealing Identity? `(Papers.SGX.0419.md)`

Answer:

The Sealing Identity is a set of criteria in Intel SGX that determines which enclaves can access a specific Sealing Key. These criteria help control the accessibility of sensitive data to future versions of an enclave.

When an enclave needs to store sensitive data securely outside the enclave, it uses the Sealing Key to encrypt and integrity-protect the data. The Sealing Identity specifies the conditions under which a later instantiation of an enclave can access this Sealing Key and decrypt the sealed data.

Intel SGX supports two policies for Sealing Identity:

1. Sealing to the Enclave Identity: This policy bases the Sealing Key on the MRENCLAVE value (the measurement of the enclave’s initial state). The key is available to any instance of the exact enclave, and future software versions cannot access the secrets of the original enclave.
2. Sealing to the Sealing Authority Identity (Sealing Identity): This policy bases the Sealing Key on the MRSIGNER value (the measurement of the enclave’s signing authority) and the enclave’s version. It allows newer enclaves signed by the same Sealing Authority to access data stored by previous versions, enabling data migration between enclave versions.

In summary, the Sealing Identity determines which enclaves can access the Sealing Key based on the selected policy, either by restricting access to the same enclave version or allowing access to newer enclave versions signed by the same Sealing Authority.

Question: Is “Sealing Identity” a MRSIGNER? `(Papers.SGX.0419.md)`

Answer:

Sealing Identity is not the same as MRSIGNER. However, MRSIGNER is a part of the Sealing Identity. Sealing Identity is a policy used when invoking EGETKEY in Intel® SGX to determine which enclaves can access a sealing key. The Sealing Identity policy is based on the enclave’s MRSIGNER value (identity of the Sealing Authority) and the enclave’s version. This policy allows for data migration between different versions of enclaves signed by the same Sealing Authority.

Question: Coud I say that SIGSTRUCT is Sealing Identity? `(Papers.SGX.0419.md)`

Answer:

No, SIGSTRUCT and Sealing Identity are not the same thing. SIGSTRUCT is a data structure that contains the enclave’s measurement (MRENCLAVE), the signer’s measurement (MRSIGNER), and other metadata related to the enclave, including the signature of the enclave created by the Sealing Authority.

On the other hand, Sealing Identity is a concept that refers to the criteria or policy used to determine which enclaves may access the sealing key generated by the EGETKEY instruction. It is either based on the enclave’s identity (MRENCLAVE) or the identity of the Sealing Authority (MRSIGNER).

Question: What is the value of Sealing Identity in the paper? `(Papers.SGX.0419.md)`

Answer:

In the paper, the Sealing Identity refers to the criteria or policy used to determine which enclaves may access the sealing key generated by the EGETKEY instruction. Intel® SGX supports two policies for Seal Keys:

1. Sealing to the Enclave Identity: This policy is based on the enclave’s MRENCLAVE. Any change that impacts the enclave’s measurement will yield a different key. This results in a different key for each enclave, providing full isolation between enclaves.
2. Sealing to the Sealing Identity: This policy is based on the enclave’s MRSIGNER and the enclave’s version. MRSIGNER reflects the key/identity of the Sealing Authority that signed the enclave’s certificate. This allows offline migration of sealed data between enclave versions.

The actual value of the Sealing Identity will depend on which policy is chosen and the specific enclave measurements involved.

Question: What is the value of Sealing Identity in the policy “Sealing to the Enclave Identity” in the paper? `(Papers.SGX.0419.md)`

Answer:

In the policy “Sealing to the Enclave Identity,” the value of the Sealing Identity is based on the enclave’s MRENCLAVE. MRENCLAVE is a unique value that represents the measurement of the enclave’s code and data. Any change that impacts the enclave’s measurement will yield a different MRENCLAVE value. Thus, the Sealing Identity in this case would be the MRENCLAVE value of the enclave.

Question: What is difference between Sealing Identity and Identity of the Sealing Authority? `(Papers.SGX.0419.md)`

Answer:

The Sealing Identity and the Identity of the Sealing Authority are related but not the same.

Sealing Identity refers to the policy used when invoking the EGETKEY instruction in Intel® SGX to determine which enclaves can access a sealing key. The Sealing Identity is based on the enclave’s MRSIGNER value and the enclave’s version. This policy allows for data migration between different versions of enclaves signed by the same Sealing Authority.

On the other hand, the Identity of the Sealing Authority is represented by the MRSIGNER value, which reflects the key or identity of the authority that signed the enclave’s certificate. The Sealing Authority is responsible for signing enclaves and determining which enclaves can access the same seal key, enabling them to transparently access data that was sealed by other enclaves under their authority.

In summary, the Sealing Identity is a policy based on the MRSIGNER and enclave version, while the Identity of the Sealing Authority is represented by the MRSIGNER value itself.

Question: How does “Sealing to the Sealing Identity” work? Please list steps. `(Papers.SGX.0419.md)`

Answer:

“Sealing to the Sealing Identity” refers to the process of encrypting data with a sealing key that is derived from the enclave’s identity. This ensures that the sealed data can only be accessed by the same enclave or other authorized enclaves that share the same identity. Here are the steps involved in this process:

1. Determine the Sealing Identity: The sealing identity is typically a combination of the MRENCLAVE (measurement of the enclave code and data) or MRSIGNER (measurement of the enclave signer) values. It is used as the basis for generating the sealing key.
2. Generate the Sealing Key: The enclave requests a sealing key from the Intel SGX hardware using the EGETKEY instruction. The key is derived from the sealing identity, the platform-specific key material, and the enclave’s attributes.

This ensures that the sealing key is unique to the enclave and the platform it runs on.

3. Encrypt the Data: The enclave uses the sealing key to encrypt the sensitive data. This process is known as “sealing.” It ensures that the data remains confidential and can only be accessed by the enclave that generated the sealing key or other authorized enclaves.
4. Store the Sealed Data: The sealed data can be stored outside the enclave, either in the application’s untrusted memory or in a persistent storage location such as a disk file. Since the data is encrypted, it remains protected even when stored outside the secure enclave memory.
5. Unseal the Data: When the enclave or another authorized enclave with the same identity needs to access the sealed data, it follows a similar process to generate the sealing key using the EGETKEY instruction. With the sealing key, the enclave can decrypt and access the sealed data.
6. Verify Integrity: Optionally, the sealing process can also include a message authentication code (MAC) or digital signature to ensure the integrity of the sealed data. During the unsealing process, the enclave verifies the MAC or digital signature to ensure that the data has not been tampered with.

By following these steps, “Sealing to the Sealing Identity” ensures that sensitive data remains confidential and can only be accessed by authorized enclaves that share the same identity. This provides a secure way to store and share data between enclaves and protect it from unauthorized access or tampering.

Question: Please compare “Sealing to the Enclave Identity” and “Sealing to the Sealing Identity” in a table. `(Papers.SGX.0419.md)`

Answer:

Here is a comparison table of “Sealing to the Enclave Identity” and “Sealing to the Sealing Identity”:

Aspect	Sealing to the Enclave Identity	Sealing to the Sealing Identity
Identity Basis	MRENCLAVE	MRENCLAVE or MRSIGNER
Enclave Measurement	Unique to each enclave build	Shared by same signer or enclave
Key Derivation	Based on MRENCLAVE	Based on MRENCLAVE or MRSIGNER
Use Case	Specific enclave instance	Enclaves with same identity
Data Accessibility	Limited to the specific enclave	Accessible by authorized enclaves
Flexibility	Lower, tied to a single enclave	Higher, shared among same identity
Software Update Considerations	Need to migrate data to new MRENCLAVE	Easier with same MRSIGNER

In summary, “Sealing to the Enclave Identity” provides more restrictive access to sealed data, ensuring that it can only be accessed by a specific enclave instance. This method is suitable for scenarios where data must be strictly protected and accessible only to that specific enclave.

On the other hand, “Sealing to the Sealing Identity” allows for more flexibility in data access among enclaves with the same identity, which can be based on MRENCLAVE or MRSIGNER. This method is suitable for scenarios where multiple enclaves need to access shared data or when enclaves have a common signer, such as when dealing with software updates or versions.

Attestation Assertion

Question: Who produces attestation assertion? `(Papers.SGX.0419.md)`

Answer:

In the context of Intel SGX, the attestation assertion is produced by the enclave software running on the hardware. The enclave generates a data structure called a quote, which contains the attestation information, including the measurement of the enclave (MRENCLAVE) and the enclave’s identity. The quote is then typically sent to a remote party (verifier) who evaluates the attestation assertion to determine if they trust the enclave and its execution environment. In some cases, an additional step is involved where the quote is sent to the Intel Attestation Service, which returns a signed attestation report that the verifier can then use to verify the enclave’s authenticity.

EREPORT

Question: What does REPORT contain? Please list them? `(Papers.SGX.0419.md)`

Answer:

A REPORT in Intel SGX is a data structure that contains information about an enclave, attesting its identity and integrity. The main components of a REPORT include:

1. MRENCLAVE: A hash representing the enclave’s identity and code. It is generated during enclave creation and depends on the enclave’s content and initial state.
2. MRSIGNER: A hash of the public key of the enclave’s signer, which represents the identity of the Sealing Authority or enclave developer.
3. ISVPRODID: The product ID assigned by the enclave developer.
4. ISVSVN: The security version number assigned by the enclave developer, representing the security-related version or patch level of the enclave.
5. REPORTDATA: A user-defined data field, which can be used to store additional information, such as a challenge-response for the remote attestation process.
6. ATTRIBUTES: Flags indicating properties of the enclave, such as its access permissions, execution modes, or other relevant characteristics.
7. CPU_SVN: A value representing the security version number of the CPU and microcode.
8. MiscSelect: A value specifying the set of miscellaneous properties to include in the measurement.
9. Timestamp: The time at which the REPORT was generated by the CPU.

The REPORT is used as a part of the local and remote attestation processes to prove the identity, integrity, and authenticity of an enclave running on an SGX-enabled platform.

EPID

Question: What is EPID? `(SGX.0418.md)`

Answer:

EPID (Enhanced Privacy ID) is a privacy-enhancing technology developed by Intel for anonymous attestation of trusted computing platforms. It is primarily used in Intel SGX (Software Guard Extensions) to verify the authenticity and integrity of an enclave without revealing the identity of the platform.

EPID enables a group signature scheme that allows a platform to prove its membership in a group without revealing its individual identity. This allows for anonymous attestation, where a service provider can verify that a platform is genuine and running trusted enclave software without needing to know the specific identity of the platform. This helps protect user privacy and reduces the risk of tracking or linking activities back to individual users.

EPID is particularly useful in scenarios where privacy is essential, such as in cloud computing or IoT environments, where platforms may need to attest their trustworthiness to remote service providers without revealing their unique identities.

Question: How does EPID work? `(SGX.0418.md)`

Answer:

EPID (Enhanced Privacy ID) is a group signature scheme developed by Intel, which allows anonymous attestation for Intel SGX-enabled platforms. It provides privacy-preserving remote attestation, meaning the verifier cannot link the attestation to a specific device or user. The key principles of EPID are:

1. Anonymous credentials: Each platform receives a unique private key that is part of a group, and the corresponding public key represents the group. When an attestation is performed, it's not possible to determine which platform or user was involved, only that the platform belongs to the group.
2. Issuer: A trusted authority (Intel, in the case of SGX) issues the credentials to the platforms. The issuer creates the group public key and a private key for each platform in the group. The issuer signs the credentials and ensures the platforms receive their private keys securely.
3. Revocation: The issuer can revoke credentials for individual platforms if required (e.g., when a security vulnerability is discovered). Revocation lists are distributed to verifiers, who can then reject attestations from revoked platforms.
4. Signature generation: When a platform wants to attest to its identity, it creates a signature using its EPID private key and the message (typically a quote from an enclave) to be signed. The signature proves that the platform belongs to the group without revealing its specific identity.
5. Signature verification: The verifier checks the signature using the group public key and the revocation list. If the signature is valid and the platform's credentials haven't been revoked, the verifier accepts the attestation as genuine.

In summary, EPID enables privacy-preserving remote attestation for Intel SGX-enabled platforms by using anonymous group signatures. The verifier can confirm that the platform is part of a trusted group without being able to link the attestation to a specific device or user.

Question: How is EPID keys enrolled into a device? `(SGX.0418.md)`

Answer:

The EPID keys are enrolled into a device during its manufacturing process. Here's an outline of the typical EPID key enrollment process:

1. Device manufacturer generates an EPID private key and public key pair for each device during the manufacturing process.
2. The device manufacturer contacts the EPID Issuer (which is responsible for managing EPID keys). The manufacturer provides the device's public key and a unique device identifier.
3. The EPID Issuer validates the authenticity of the device manufacturer and the device public key. After validation, the EPID Issuer issues a certificate (also known as an EPID signature) that binds the device's public key to its unique identifier. The certificate serves as proof that the device's public key is part of the EPID group.
4. The device manufacturer then securely stores the EPID private key and the corresponding certificate within the device, typically in a protected storage area.

The enrollment process ensures that each device has a unique EPID key pair and a valid certificate. The private key is securely stored within the device and is used for signing attestation requests, while the public key can be used by remote parties to verify the authenticity of the device's attestation signatures.

Question: Which problems does EPID solve? `(SGX.0418.md)`

Answer:

EPID (Enhanced Privacy ID) solves several problems related to remote attestation and device identity management, particularly for Intel SGX-enabled platforms. Some of the key problems EPID addresses are:

1. Privacy: Traditional attestation schemes can potentially reveal the identity of the device or user, which may lead to privacy concerns. EPID utilizes anonymous group signatures to ensure that the verifier can only determine that the attesting platform belongs to a trusted group, without revealing the specific identity of the device or user.
2. Scalability: In scenarios where a large number of devices require attestation, managing individual public keys for each device can be cumbersome. EPID simplifies this by using a single group public key for all devices within a group. This allows verifiers to validate attestations using a single public key, making the process more scalable.
3. Revocation: EPID provides an efficient mechanism to revoke the credentials of compromised or vulnerable devices. The issuer maintains a revocation list that allows verifiers to reject attestations from revoked platforms. This helps maintain

the security and trust of the attestation process.

4. Device onboarding: EPID allows for a smoother and more secure onboarding process for new devices. The trusted authority (e.g., Intel) issues credentials for each device, ensuring that they receive their private keys securely. This helps streamline device provisioning and management.

5. Interoperability: EPID can be integrated into various attestation protocols and systems, promoting interoperability between different devices, platforms, and services. It can be used in conjunction with other security mechanisms, such as Transport Layer Security (TLS) or secure boot, to create secure communication channels and establish trust.

In summary, EPID addresses key challenges in remote attestation, device identity management, and privacy. It helps create a more secure and scalable system for verifying the trustworthiness of devices and protecting user privacy.

Question: How do I make sure that the EPID issuer doesn't retain private keys? `(SGX.0418.md)`

Answer:

To ensure that the EPID issuer doesn't retain private keys, you can implement a few strategies:

1. Use a Trusted Third Party (TTP): Instead of relying on the EPID issuer to generate private keys, you can employ a TTP to issue the private keys to devices. The TTP can be an independent entity that all parties trust, ensuring that the EPID issuer doesn't have access to the private keys.

2. Generate keys on the device: One way to prevent the EPID issuer from retaining private keys is by generating the keys directly on the device itself. This can be done using a hardware-based secure element or a Trusted Platform Module (TPM) to generate and securely store the private key. By doing this, the private key never leaves the device, ensuring that the EPID issuer doesn't have access to it.

3. Zero-knowledge proofs: Zero-knowledge proofs can be used to prove the validity of a private key without revealing the key itself. This can help ensure that the EPID issuer doesn't have access to the private keys while still being able to verify their authenticity.

4. Audits and transparency: Regular audits and transparency reports can help build trust between the EPID issuer and the devices. By allowing third-party audits of their key management practices and infrastructure, the EPID issuer can demonstrate that they do not retain private keys.

5. Legal and contractual agreements: Establishing legal and contractual agreements between the EPID issuer and the device manufacturers or users can help define the responsibilities and limitations of the issuer. These agreements can include clauses that prohibit the issuer from retaining private keys and specify penalties for non-compliance.

In summary, ensuring that the EPID issuer doesn't retain private keys requires a combination of technical and procedural measures, along with trust-building mechanisms like audits and legal agreements.

Question: How does EPID revoke a compromised device? `(SGX.0418.md)`

Answer:

EPID (Enhanced Privacy ID) revocation works through the use of revocation lists, specifically the Group Revocation List (GRL) and the Private Key-Based Revocation List (PRL). When a device is compromised or its private key is deemed insecure, its identifier (member ID or private key-based identifier) is added to one of these lists. This process ensures that the compromised device will no longer be considered valid during the attestation process.

Here's an overview of how EPID revocation works:

1. Detection of compromise: The first step is identifying that a device or its private key has been compromised. This can be done through monitoring, reporting, or other means.

2. Adding the identifier to the revocation list: Once a device is identified as compromised, its identifier (member ID or private key-based identifier) is added to the appropriate revocation list (GRL or PRL).

3. Distribution of revocation lists: The revocation lists are periodically updated and distributed to relying parties, such as service providers and verifiers, who need to validate the authenticity of devices during attestation.

4. Attestation process: During the attestation process, the device generates a signature using its private key and EPID parameters. The verifier checks the signature against the public parameters and the revocation lists.

5. Verification and revocation: If the device's identifier is found in a revocation list, the verifier will consider the attestation invalid, effectively revoking the device's access to services or data. If the device's identifier is not in any revocation list, the verifier will proceed with the attestation process and grant access accordingly.

In summary, EPID revokes a compromised device by adding its identifier to a revocation list, which is then distributed to relying parties. During the attestation process, the device's signature is checked against the revocation lists, and if it's found on a list, the attestation is considered invalid, effectively revoking the device's access to services or data.

Question: How to get the EPID of the compromised device? `(SGX.0418.md)`

Answer:

To get the EPID (Enhanced Privacy ID) of a compromised device, you'll typically need to follow an incident response process to gather information about the device, its security status, and potential threats. Here's a general outline of the steps you should take:

1. Incident detection: The first step is to detect a potential security incident or compromise involving the device. This can be done through monitoring systems, alerts, or reports from users or administrators.

2. Incident investigation: Once you've detected a potential compromise, you need to investigate the incident to determine the cause, scope, and impact. This may involve analyzing logs, network traffic, and other relevant data.

3. Device identification: During the investigation, you'll need to identify the specific device(s) involved in the incident. This may involve correlating information from logs, network traffic, or other sources to pinpoint the affected device(s).

4. Collecting device information: Once you've identified the compromised device, collect relevant information about it, such as the serial number, hardware and software configurations, and any unique identifiers, like EPID parameters or other security-related data.

5. Identifying the EPID: With the collected device information, identify the device's EPID by analyzing its EPID parameters (e.g., member ID or private key-based identifier). This may require accessing the device directly or working with the device manufacturer or platform provider.

6. Revoking the EPID: Once you have the EPID of the compromised device, you can add it to the appropriate revocation list (Group Revocation List or Private Key-Based Revocation List) to revoke its access to services or data during the attestation process.

Remember that the specific steps and processes may vary depending on the device type, platform, and organization's incident response procedures. It's crucial to have a well-defined incident response plan in place to ensure an efficient and effective response to security incidents.

Question: But if the compromised device reports a fake EPID, what should I do? `(SGX.0418.md)`

Answer:

If a compromised device is suspected of reporting a fake EPID, there are several steps you can take to mitigate the risk and verify the authenticity of the device:

1. Verify attestation reports: When using remote attestation, check the attestation reports provided by the device to ensure they are genuine and have been signed by a trusted attestation service. This can help detect fake EPIDs by verifying the signature, certificate chain, and the data contained within the report.

2. Monitor for anomalous behavior: Continuously monitor the behavior of devices and their network communications. Look for unusual patterns or activities that could indicate a compromised device or the use of a fake EPID.

3. Implement intrusion detection and prevention systems (IDS/IPS): Use IDS/IPS to monitor network traffic and detect potential threats, such as attempts to use fake EPIDs or unauthorized access to sensitive data.

4. Implement strong authentication and access controls: Ensure that devices are required to authenticate themselves using strong methods, such as multi-factor authentication, before they can access sensitive data or services. This can help

prevent unauthorized access by compromised devices using fake EPIDs.

5. Regularly update and patch software: Keep the software on devices up-to-date and apply security patches promptly to minimize the risk of vulnerabilities being exploited.
6. Conduct regular security assessments and audits: Perform regular security assessments and audits to identify potential risks and vulnerabilities in your environment. This can help uncover compromised devices and attempts to use fake EPIDs.
7. Establish and enforce security policies: Develop and implement security policies that define the acceptable use of devices, outline the responsibilities of users and administrators, and specify the requirements for reporting and responding to security incidents.
8. Train users and administrators: Provide regular training and awareness programs to educate users and administrators about the risks of compromised devices and the use of fake EPIDs. This can help ensure that they are better prepared to detect and respond to potential threats.
9. Collaborate with vendors and partners: Work closely with device manufacturers, platform providers, and other partners to share information about potential threats, vulnerabilities, and best practices for detecting and mitigating the risk of compromised devices and fake EPIDs.

In case a fake EPID is detected, follow your organization’s incident response plan to contain the threat, investigate the issue, and recover from the incident.

Question: Does Intel EPID require TPM? `(SGX.0418.md)`

Answer:

Intel EPID (Enhanced Privacy ID) does not require a Trusted Platform Module (TPM) to function. EPID is a separate privacy-preserving technology designed for remote attestation and device authentication in a scalable and anonymous manner.

Although both Intel EPID and TPM provide security features, they serve different purposes and can be used independently or in conjunction with each other, depending on the specific use case and security requirements of a system.
