2023-04-02

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

Questions

- Secure Storage `(230327.OP-TEE.yaml)`
  - `[08:10]` How does OP-TEE generate SSK from HUK in tee_fs_init_key_manager()? `(OP-TEE.0327.md)`
  - `[08:12]` In the function, can I read the content of the HUK? `(OP-TEE.0327.md)`
  - `[08:18]` How do I protect HUK from reading in OP-TEE? `(OP-TEE.0327.md)`
  - `[08:20]` Is SSK defined in GlobalPlatform? `(OP-TEE.0327.md)`
- Basic Concept `(230402.DRM.yaml)`
  - `[14:25]` What is DRM? `(DRM.0402.md)`
  - `[14:26]` How does DRM protect digital contents? `(DRM.0402.md)`
  - `[14:28]` Please describe more detail about encryption method. `(DRM.0402.md)`
  - `[14:31]` Please explain Time Limitations used by DRM. `(DRM.0402.md)`
  - `[16:12]` What is ClearKey? `(DRM.0402.md)`
  - `[16:56]` What is PlayReady? `(DRM.0402.md)`
  - `[17:00]` What is kind of DRM? `(DRM.0402.md)`
- DRM `(230327.OP-TEE.yaml)`
  - `[14:35]` How do I implement DRM of content decryption with OP-TEE? `(OP-TEE.0327.md)`
  - `[14:39]` So the content is decrypted in TA in secure world. Right? `(OP-TEE.0327.md)`
  - `[14:41]` But does it slow down the playback speed because of switching between normal world and secure world? `(OP-TEE.0327.md)`
  - `[14:44]` Can I use multi cores to solve the problem? For example, one core decrypts content and another core play content at the same time. `(OP-TEE.0327.md)`
  - `[16:10]` How do I implement a DRM solution using ClearKey in OP-TEE? `(OP-TEE.0327.md)`
- Memory `(230327.OP-TEE.yaml)`
  - `[17:33]` What is SMAF? `(OP-TEE.0327.md)`
  - `[17:45]` What is CMA? `(OP-TEE.0327.md)`
- Qualcomm `(230327.OP-TEE.yaml)`
  - `[20:26]` Does Qualcomm support OP-TEE? `(OP-TEE.0327.md)`
  - `[20:28]` Is QTEE open source? `(OP-TEE.0327.md)`
- Static TA and Dynamic TA `(230327.OP-TEE.yaml)`
  - `[20:31]` What are Static TA and Dynamic TA? Please also summarize them in a table. `(OP-TEE.0327.md)`

Q & A

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

Question: `[08:10]` How does OP-TEE generate SSK from HUK in tee_fs_init_key_manager()? `(OP-TEE.0327.md)`

Answer:

In OP-TEE, the `tee_fs_init_key_manager()` function is responsible for generating the Secure Storage Key (SSK) from the Hardware Unique Key (HUK). This process takes place during the initialization of the secure file system. The function can be found in the `core/tee/tee_fs_key_manager.c` file in the OP-TEE source code.

Here's an overview of the steps involved in generating the SSK from the HUK in `tee_fs_init_key_manager()`:

1. Retrieve the HUK: The HUK is obtained by calling `tee_otp_get_hw_unique_key()`. This function retrieves the device-

specific hardware unique key and stores it in a `tee_hw_unique_key` structure.

```
struct tee_hw_unique_key huk = {0};
if (tee_otp_get_hw_unique_key(&huk) != TEE_SUCCESS)
    return TEE_ERROR_SECURITY;
```

2. Create a message containing a static string and the device's chip ID: The message is created by concatenating a static string (e.g., "ONLY_FOR_tee_fs_ssk") and the device's chip ID, which can be obtained using `tee_otp_get_die_id()`.

```
uint8_t message[sizeof(ssk_gen_message) + TEE_FS_KM_CHIP_ID_LENGTH] = {0};
memcpy(message, ssk_gen_message, sizeof(ssk_gen_message));
if (tee_otp_get_die_id(message + sizeof(ssk_gen_message), TEE_FS_KM_CHIP_ID_LENGTH) !=
    TEE_SUCCESS)
    return TEE_ERROR_SECURITY;
```

3. Derive the SSK: The SSK is derived from the HUK using the HMAC-SHA-256 function. The HMAC function takes the HUK as the key and the message created in the previous step as input data. The result is a 256-bit output, which is used as the SSK.

```
uint8_t digest[TEE_FS_KM_HMAC_SHA256_DIGEST_SIZE] = {0};
TEE_Result res = tee_do_hmac_sha256(huk.data, sizeof(huk.data), message,
                                   sizeof(message), digest);
if (res != TEE_SUCCESS)
    return res;

memcpy(tee_fs_ssk.key, digest, sizeof(tee_fs_ssk.key));
```

After these steps, the SSK is generated and stored in the `tee_fs_ssk` global variable, which is used by the secure file system for encryption and decryption.

In summary, the `tee_fs_init_key_manager()` function in OP-TEE generates the SSK from the HUK by first retrieving the HUK and then using the HMAC-SHA-256 function with a message consisting of a static string and the device's chip ID. The derived SSK is then used for encryption and decryption in the secure file system.

Question: `[08:12]` In the function, can I read the content of the HUK?`(OP–TEE.0327.md)`

Answer:

In OP-TEE's `tee_fs_init_key_manager()` function, the Hardware Unique Key (HUK) is retrieved and stored in a local variable. While it is technically possible to read the content of the HUK within this function, doing so would pose a security risk. The HUK is intended to be a secret key, unique to each device, and should not be exposed to unauthorized parties.

If you want to inspect the HUK for debugging or development purposes, you can use debug logging functions (such as `DMSG()` or `EMSG()`) to print the content of the HUK within the `tee_fs_init_key_manager()` function. However, you should remove or disable such logging statements before deploying your OP-TEE build in a production environment, to ensure the security of the HUK.

Keep in mind that exposing the HUK can lead to potential security vulnerabilities, as it serves as the basis for deriving various other keys (such as the Secure Storage Key) used in OP-TEE's secure services. It is crucial to handle the HUK and other sensitive data carefully to maintain the overall security of the system.

Question: `[08:18]` How do I protect HUK from reading in OP-TEE?`(OP–TEE.0327.md)`

Answer:

Protecting the Hardware Unique Key (HUK) is essential for ensuring the security of an OP-TEE-based system. Since the HUK is used as the basis for deriving other keys and securing various sensitive operations, it is crucial to keep it confidential. Here are some best practices for protecting the HUK in OP-TEE:

1. Limit access to the HUK: Ensure that the HUK is only accessed by the necessary parts of the OP-TEE core and limit its

exposure to other components. This can be achieved by using the HUK only within trusted functions and avoiding passing it as a parameter to other functions or storing it in global variables.

2. Avoid logging the HUK: When developing or debugging OP-TEE, avoid logging the HUK or any derived keys. If you need to debug HUK-related issues, make sure to remove or disable any logging statements before deploying your build in a production environment.
3. Protect HUK storage in hardware: The HUK is usually stored in a secure, tamper-resistant location within the device hardware, such as a one-time programmable (OTP) memory or a secure element. Ensure that the hardware implementation adequately protects the HUK from unauthorized access or tampering.
4. Protect the HUK during retrieval: When the HUK is retrieved from hardware, use platform-specific secure functions to access it, and make sure that the retrieval process is implemented securely to prevent unauthorized access to the HUK.
5. Secure software implementation: Follow secure coding practices to minimize the risk of software vulnerabilities that could lead to unauthorized access to the HUK or other sensitive data. Regularly review and update the OP-TEE implementation to incorporate the latest security patches and enhancements.
6. Secure boot process: Implement a secure boot process to ensure that only authorized and trusted code can execute on the device. This will help prevent potential attacks that could compromise the HUK or other sensitive information.

By following these best practices, you can help protect the HUK from unauthorized access and maintain the security of your OP-TEE-based system. Remember that safeguarding the HUK is crucial, as it serves as the foundation for various other keys and secure operations within the system.

Question: `[08:20]` Is SSK defined in GlobalPlatform? `(OP–TEE.0327.md)`

Answer:

The Secure Storage Key (SSK) concept, as implemented in OP-TEE, is not directly defined by the GlobalPlatform TEE specifications. The GlobalPlatform TEE specifications define a set of APIs and security requirements for Trusted Applications and the TEE itself but do not specifically address the implementation details of secure storage.

In OP-TEE, the SSK is used for encrypting and decrypting data stored in the secure storage. It is derived from the Hardware Unique Key (HUK) using a Key Derivation Function (KDF). While this implementation choice is specific to OP-TEE, it aligns with the broader security goals outlined by the GlobalPlatform TEE specifications, such as ensuring the confidentiality and integrity of data in secure storage.

The GlobalPlatform TEE specifications provide guidance on general security requirements, but individual TEE implementations, such as OP-TEE, can choose how to implement specific features like secure storage and key management, as long as they meet the overall security objectives of the specifications.

Basic Concept `(230402.DRM.yaml)`

Question: `[14:25]` What is DRM? `(DRM.0402.md)`

Answer:

DRM stands for Digital Rights Management. It is a set of technologies and techniques used by copyright holders, publishers, and content providers to control the access, distribution, and usage of digital media, such as software, ebooks, music, and movies. The primary goal of DRM is to protect intellectual property rights and prevent unauthorized copying, sharing, or modification of digital content.

DRM systems typically involve encryption, digital watermarks, or other mechanisms that restrict the use of content to authorized users or devices. Common DRM methods include requiring a user to enter a license key, limiting the number of devices on which content can be accessed, or implementing online authentication checks.

While DRM is intended to combat piracy and protect creators' rights, it has also been criticized for potentially infringing on consumer rights, hindering fair use, and causing inconvenience for legitimate users.

Question: `[14:26]` How does DRM protect digital contents? `(DRM.0402.md)`

Answer:

DRM protects digital content by implementing various access control measures and restrictions that prevent unauthorized copying, distribution, and usage. Here are some common methods used by DRM systems to protect digital content:

1. Encryption: DRM systems can encrypt content, which means the data is scrambled and can only be decrypted and accessed by authorized users or devices with the correct decryption key. This prevents unauthorized users from accessing the content.

2. Authentication: DRM systems may require users to authenticate their identity or ownership of the content, often through online servers. This could involve entering a unique license key, providing login credentials, or validating a purchase receipt.

3. Device Binding: DRM can restrict the number of devices on which content can be accessed or installed. This prevents users from sharing content with an unlimited number of devices, which could lead to piracy.

4. Time Limitations: Some DRM systems limit the duration for which content can be accessed, requiring users to renew their licenses periodically. This can be used for subscription-based services or time-limited trials.

5. Digital Watermarking: Digital watermarks are embedded into the content and can be used to trace unauthorized copies back to the original user. This discourages illegal sharing as it can lead to the identification of the source.

6. Copy Protection: DRM can restrict the ability to copy or duplicate the content. This can be achieved through various techniques, such as copy-protected CDs or the inability to use screen capture tools on protected video content.

7. Usage Restrictions: DRM can impose limitations on how content is used, such as preventing printing, editing, or sharing of ebooks or documents.

These methods work together to ensure that only authorized users can access and use the digital content while minimizing the potential for piracy or unauthorized sharing. However, DRM has also faced criticism for sometimes being overly restrictive, causing inconvenience to legitimate users, and limiting the fair use of content.

Question: `[14:28]` Please describe more detail about encryption method. `(DRM.0402.md)`

Answer:

Encryption is a key method used in DRM systems to protect digital content by converting the original data into a scrambled, unreadable format that can only be accessed by authorized users or devices with the correct decryption key. Here’s a more detailed look at the encryption process in DRM:

1. Symmetric Key Encryption: In symmetric key encryption, a single secret key is used for both encrypting and decrypting the content. The content provider encrypts the data using this secret key, and the authorized user or device must use the same key to decrypt the content. The main challenge with symmetric key encryption is securely distributing the secret key to authorized users without it falling into the hands of unauthorized users.

2. Asymmetric Key Encryption: Asymmetric key encryption, also known as public-key cryptography, uses two separate keys: a public key and a private key. The public key is used to encrypt the data, while the private key is used to decrypt it. The content provider encrypts the data using the public key of the authorized user, and only the corresponding private key can decrypt the content. The private key is never shared and must be securely stored by the user or device. This method resolves the key distribution issue of symmetric key encryption but can be more computationally intensive.

3. Hybrid Encryption: Hybrid encryption combines the advantages of both symmetric and asymmetric key encryption. In this approach, a symmetric key is generated for encrypting the content, and then the symmetric key itself is encrypted using the public key of the authorized user. This combines the computational efficiency of symmetric key encryption with the secure key distribution of asymmetric key encryption.

In a DRM system, the encrypted content is typically packaged with metadata that includes information about the content, usage rights, and restrictions. When an authorized user or device attempts to access the content, the DRM system verifies their credentials or license and provides the necessary decryption key or mechanism to unlock the content. The content remains encrypted during transmission and storage, which makes it difficult for unauthorized users to access or tamper with the data.

It’s important to note that while encryption is a powerful tool for protecting digital content, it is not foolproof. Skilled hackers or attackers may still find ways to circumvent encryption, and there is an ongoing arms race between DRM technologies and methods used to bypass them.

Question: `[14:31]` Please explain Time Limitations used by DRM. `(DRM.0402.md)`

Answer:

Time limitations are a method used by some DRM systems to control access to digital content for a specific period. By imposing a time constraint, content providers can offer temporary access to the content, such as rentals, trials, or subscription-based services. Here’s how time limitations work in DRM:

1. Time-based Licenses: When a user purchases or acquires access to digital content, the DRM system issues a license that specifies the duration of access. The license contains information about the start and end dates, as well as any usage rights and restrictions associated with the content. The DRM system ensures that the content is only accessible during the specified period, after which the license expires, and the user can no longer access the content without renewing or obtaining a new license.

2. Subscription-based Services: Time limitations are often used in subscription-based services, such as streaming platforms or software as a service (SaaS). Users pay a recurring fee to access the content or services for a specific period (e.g., monthly or annually). The DRM system continually checks the user’s subscription status and grants access to the content as long as the subscription is active. Once the subscription expires, access to the content is revoked until the user renews their subscription.

3. Time-limited Trials: Time limitations can also be used for offering trial versions of digital content, such as software, games, or premium features. The user is granted access to the content for a limited time (e.g., 7 days, 14 days, or 30 days), allowing them to evaluate the product before deciding whether to purchase or subscribe. After the trial period ends, the DRM system blocks access to the content, and the user must obtain a full license or subscription to continue using it.

4. Rentals: In the case of digital media rentals, such as movies or e-books, time limitations enable users to access the content for a short period (e.g., 24 hours, 48 hours, or a week) at a lower cost compared to purchasing the content outright. Once the rental period expires, the DRM system prevents further access to the content.

Time limitations used by DRM systems ensure that access to digital content is granted only for the agreed-upon duration, allowing content providers to monetize their products and services through various pricing models and usage scenarios. However, time limitations can also be a point of criticism, as they may limit the user’s ability to use the content freely and may require continuous internet connectivity for the DRM system to verify the validity of the license or subscription.

Question: `[16:12]` What is ClearKey? `(DRM.0402.md)`

Answer:

ClearKey is a simple, non-proprietary DRM system that is part of the Encrypted Media Extensions (EME) specification for HTML5. EME is a framework that allows web browsers to interact with DRM systems, enabling the playback of encrypted media content within web pages without the need for plugins like Adobe Flash or Microsoft Silverlight.

ClearKey is designed to be a lightweight, easy-to-implement DRM solution that uses clear-text key exchange for content decryption. It provides a basic level of content protection without the complexity and cost associated with more advanced DRM systems like Google’s Widevine, Apple’s FairPlay, or Microsoft’s PlayReady.

In ClearKey, the content is encrypted using a key, and the encrypted content is then transmitted to the user’s device. The decryption key is sent separately, typically in clear text, meaning it is not further encrypted or obfuscated. The user’s device, usually a web browser supporting EME, uses the ClearKey decryption key to unlock and play the content.

While ClearKey provides a basic level of protection, its simplicity and the use of clear-text key exchange make it less secure than other DRM solutions. As a result, ClearKey is not recommended for high-value content or scenarios that require robust protection against piracy or unauthorized access. It is more suited for low-risk applications or as a fallback option when other DRM systems are not available or supported by the user’s device.

Question: `[16:56]` What is PlayReady? `(DRM.0402.md)`

Answer:

PlayReady is a comprehensive DRM (Digital Rights Management) technology developed by Microsoft. It is designed to protect copyrighted digital content such as audio, video, and multimedia applications across various platforms and devices, including smartphones, tablets, PCs, gaming consoles, and smart TVs. PlayReady aims to provide a high level of content protection while offering flexibility for content providers, distributors, and device manufacturers.

Some of the key features of PlayReady include:

1. Wide platform support: PlayReady is compatible with a broad range of devices and operating systems, including Windows, Xbox, Android, iOS, and smart TVs, making it a popular choice for many content providers.
2. Content encryption and licensing: PlayReady uses advanced encryption techniques to protect digital content. It generates licenses containing usage rights and restrictions, which are distributed to authorized users or devices, enabling them to decrypt and play the content according to the specified rules.
3. Domain-based content sharing: PlayReady supports domain-based content sharing, which allows users to share purchased content with a limited number of devices within their “domain.” This feature provides flexibility for households with multiple devices while preventing unauthorized sharing of content.
4. Output protection: PlayReady includes output protection mechanisms that prevent unauthorized copying of content through digital or analog outputs, such as HDMI or component video, ensuring that the content is only played on authorized devices.
5. Secure key exchange: PlayReady uses secure key exchange protocols to protect the transmission of decryption keys between content servers and client devices, ensuring that the keys remain confidential and cannot be intercepted by unauthorized users.
6. Interoperability with other DRM systems: PlayReady is designed to work alongside other DRM technologies, such as Google’s Widevine and Apple’s FairPlay, allowing content providers to support multiple DRM systems for broader device compatibility.

PlayReady is widely used by content providers, streaming platforms, and device manufacturers to secure their digital content and enforce usage policies. However, like other DRM systems, PlayReady has faced criticism for potentially limiting consumer rights, hindering fair use, and causing inconvenience to legitimate users.

Question: `[17:00]` What is kind of DRM? `(DRM.0402.md)`

Answer:

There are several types of DRM (Digital Rights Management) systems, each with its specific features, technologies, and target platforms. Some of the most widely used DRM systems include:

1. Microsoft PlayReady: A comprehensive DRM technology developed by Microsoft that protects digital content across various platforms and devices, such as Windows, Xbox, Android, iOS, and smart TVs. PlayReady supports a wide range of content protection features, including content encryption, licensing, domain-based sharing, output protection, and secure key exchange.
2. Apple FairPlay: Apple’s proprietary DRM system used to protect digital content on Apple devices and platforms, including iOS, macOS, and tvOS. FairPlay is mainly used for securing content distributed through iTunes, Apple Music, and Apple TV+. It offers content encryption, secure key delivery, and access control features specific to the Apple ecosystem.
3. Google Widevine: A DRM solution developed by Google, primarily used for protecting video content on Android devices and the Chrome browser. Widevine offers a multi-tiered content protection approach, with three security levels (L1, L2, and L3) to accommodate various device capabilities and content protection requirements.
4. Adobe Primetime DRM (formerly Adobe Access): A DRM solution developed by Adobe, primarily used for securing digital content delivered through Adobe’s Primetime platform. Adobe Primetime DRM supports content protection features such as encryption, licensing, domain-based sharing, and output protection.
5. Marlin DRM: An open-standard DRM technology developed by the Marlin Developer Community, which includes companies like Sony, Panasonic, Samsung, and Philips. Marlin DRM is designed to be platform-agnostic and is used for

protecting digital content on various devices, including TVs, set-top boxes, and mobile devices.

6. ClearKey: A simple, non-proprietary DRM system that is part of the Encrypted Media Extensions (EME) specification for HTML5. ClearKey is a lightweight, easy-to-implement DRM solution that uses clear-text key exchange for content decryption, making it less secure than other DRM systems but suitable for low-risk applications or as a fallback option.

These DRM systems protect digital content by implementing various access control measures and restrictions, such as encryption, licensing, and usage limitations, to prevent unauthorized copying, sharing, and modification. Each DRM system targets different platforms, devices, and use cases, offering varying levels of content protection and compatibility.

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

Question: `[14:35]` How do I implement DRM of content decryption with OP-TEE?`(OP-TEE.0327.md)`

Answer:

Implementing DRM content decryption with OP-TEE requires the development of a Trusted Application (TA) that handles the decryption process securely within the TEE environment. Here is a step-by-step guide to help you implement DRM content decryption using OP-TEE:

1. Define decryption requirements: Identify the specific encryption schemes and formats used for the protected content, such as AES-CTR, AES-CBC, or others.

2. Develop a Trusted Application: Create a TA that handles the decryption process. The TA should receive encrypted content and any necessary decryption keys or metadata from the normal world and perform the decryption securely within the TEE environment. It should also return the decrypted content to the normal world application for playback.

```
TEE_Result TA_InvokeCommandEntryPoint(void *session_context, uint32_t command_id,
                                      uint32_t param_types, TEE_Param params[TEE_NUM_PARAMS]) {
    switch (command_id) {
        case TA_DRM_DECRYPT_CMD:
            return ta_drm_decrypt(param_types, params);
        ...
    }
}
```

3. Implement decryption functionality: Within the TA, implement the necessary decryption algorithms and functionality. You can use the cryptographic operations provided by the TEE Internal API to perform the decryption.

```
static TEE_Result ta_drm_decrypt(uint32_t param_types, TEE_Param params[TEE_NUM_PARAMS]) {
    TEE_Result res;
    TEE_OperationHandle op = TEE_HANDLE_NULL;
    TEE_ObjectHandle key_handle = TEE_HANDLE_NULL;
    ...
    // Create a transient key object and set the decryption key
    res = TEE_AllocateTransientObject(TEE_TYPE_AES, key_size, &key_handle);
    res = TEE_PopulateTransientObject(key_handle, key_attr, 1);

    // Allocate a new operation and set the key for decryption
    res = TEE_AllocateOperation(&op, TEE_ALG_AES_CBC_NOPAD, TEE_MODE_DECRYPT, key_size);
    res = TEE_SetOperationKey(op, key_handle);

    // Perform the decryption
    res = TEE_CipherDoFinal(op, encrypted_data, encrypted_data_size, decrypted_data, &decrypted_data_size
    ...
}
```

4. Integrate with the normal world application: Develop a normal world application that communicates with the TA using OP-TEE's GlobalPlatform TEE Client API. The normal world application should be responsible for obtaining encrypted content, the decryption key, and any necessary metadata from the DRM system, and then passing this information to the TA for decryption.

```
TEEC_Result res;
TEEC_Session session;
TEEC_Operation op;
...
// Prepare the operation parameters
op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT, TEEC_MEMREF_TEMP_INPUT, TEEC_MEMREF_TEMP_OUTPUT
op.params[0].tmpref.buffer = encrypted_data;
op.params[0].tmpref.size = encrypted_data_size;
op.params[1].tmpref.buffer = decryption_key;
op.params[1].tmpref.size = decryption_key_size;
op.params[2].tmpref.buffer = decrypted_data;
op.params[2].tmpref.size = decrypted_data_size;

// Invoke the TA command for decryption
res = TEEC_InvokeCommand(&session, TA_DRM_DECRYPT_CMD, &op, NULL);
```

5. Test and validate: Test your TA and normal world application to ensure proper decryption and playback of the protected content. Verify that the content is decrypted securely within the TEE environment and is protected from unauthorized access or tampering.

By following these steps, you can implement DRM content decryption with OP-TEE, ensuring that the decryption process is secure and protected within

---

Question: `[14:39]` So the content is decrypted in TA in secure world. Right?`(OP-TEE.0327.md)`

Answer:

Yes, that's correct. In the described DRM implementation, the content is decrypted within the Trusted Application (TA) running in the secure world of OP-TEE. The TA receives encrypted content and decryption keys from the normal world application and performs the decryption securely within the TEE environment. The decrypted content is then returned to the normal world application for playback. This approach ensures that sensitive information, such as decryption keys and decrypted content, is protected within the secure world, preventing unauthorized access or tampering.

Question: `[14:41]` But does it slow down the playback speed because of switching between normal world and secure world? `(OP-TEE.0327.md)`

Answer:

Switching between the normal world and the secure world does introduce some overhead, as the system needs to perform context switching and maintain the isolation between the two environments. However, this overhead is generally small compared to the overall processing time required for content decryption and playback.

The exact performance impact will depend on various factors, such as the hardware platform, the complexity of the decryption algorithm, and the size of the content being processed. In most cases, the benefits of secure content decryption within the TEE outweigh the slight performance overhead introduced by the world-switching.

To minimize the performance impact, you can optimize the design of your Trusted Application and normal world application. For example, you can:

1. Minimize the number of world switches by processing larger chunks of encrypted content in a single invocation of the Trusted Application.
2. Optimize the cryptographic operations within the Trusted Application by leveraging hardware-accelerated cryptographic functions, if available on your platform.
3. Optimize data handling and buffering between the normal world application and the Trusted Application to reduce data copying and memory allocation overhead.

By carefully designing and optimizing your implementation, you can ensure that the performance impact of secure content decryption in the TEE is minimized while still providing robust security for your DRM system.

Question: `[14:44]` Can I use multi cores to solve the problem? For example, one core decrypts content and another core play content at the same time. `(OP–TEE.0327.md)`

Answer:

Yes, using multi-core processing can help improve the performance of your DRM implementation by parallelizing the decryption and playback processes. If your platform supports multiple cores, you can assign different tasks to different cores to minimize the overhead introduced by secure world and normal world switching. Here's a general approach to achieve this:

1. Assign content decryption to one core: Designate one core to handle the decryption process within the Trusted Application in the secure world. This core will be responsible for processing encrypted content and performing the decryption using the provided keys.

2. Assign content playback to another core: Designate a different core to handle the playback process in the normal world. This core will be responsible for receiving the decrypted content from the Trusted Application and managing the playback process, including rendering, audio synchronization, and user interaction.

3. Implement inter-core communication: Develop a mechanism for inter-core communication to allow the decryption core and the playback core to exchange data efficiently. This can be achieved using shared memory buffers, message queues, or other communication techniques suitable for your platform.

4. Manage synchronization and buffering: Implement synchronization and buffering mechanisms to ensure smooth playback while maintaining the parallelism between decryption and playback processes. This may involve using semaphores, mutexes, or other synchronization primitives to coordinate data exchange between the cores and prevent buffer overflows or underflows.

By using multi-core processing, you can offload the decryption process to a separate core, reducing the impact of secure world and normal world switching on playback performance. However, keep in mind that implementing a multi-core DRM solution can be more complex, as it requires careful design and synchronization to ensure smooth operation and proper security. Be sure to thoroughly test and validate your implementation to ensure it meets the performance and security requirements of your DRM system.

Question: `[16:10]` How do I implement a DRM solution using ClearKey in OP-TEE?`(OP–TEE.0327.md)`

Answer:

To implement a DRM solution using ClearKey in OP-TEE, you'll need to develop a Trusted Application (TA) that handles the decryption process within the secure world of OP-TEE, and a normal world application (e.g., a browser or media player) that communicates with the TA for decrypting media content. Here are the general steps:

1. Develop a Trusted Application (TA): Create a TA that securely handles decryption keys and content decryption within the TEE environment. The TA will receive encrypted content and decryption keys from the normal world application and perform the decryption securely within the TEE.

2. Implement the ClearKey decryption algorithm: Since ClearKey is a simple and open-source key system, you can readily find implementations or libraries that can be integrated into your TA. You will need to implement the decryption algorithm (usually AES-128-CBC or AES-128-CTR) within the TA to decrypt the content.

3. Develop a normal world application: Create a browser or media player application that uses the Encrypted Media Extensions (EME) API with the ClearKey key system. This application will be responsible for handling media playback, parsing the encrypted content, and extracting the necessary metadata (e.g., content encryption keys, initialization vectors) to pass to the TA for decryption.

4. Communication between normal world application and TA: Establish a secure communication channel between your normal world application and the TA. Your normal world application will send encrypted media content, keys, and other required metadata to the TA, which will then decrypt the content and return it to the normal world application for playback.

5. Test and validate your implementation: Ensure that your DRM solution works correctly by testing it with various encrypted media content and key configurations. Make sure the implementation provides the necessary security features to protect the content and keys from unauthorized access or tampering.

Keep in mind that ClearKey is a basic key system primarily intended for testing and development purposes. In a production environment, you may want to use a more robust and secure DRM system, such as Widevine or PlayReady, depending on your requirements and platform support. Additionally, be aware that implementing a DRM solution in OP-TEE can be complex, and you'll need to consider the hardware and platform limitations, performance, and security requirements for your specific use case.

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

Question: `[17:33]` What is SMAF? `(OP-TEE.0327.md)`

Answer:

SMAF stands for Secure Memory Allocation Framework, which is a part of the OP-TEE project. SMAF is designed to provide secure memory allocation and management in a Trusted Execution Environment (TEE). It allows OP-TEE to allocate, deallocate, and manage memory resources efficiently and securely within the secure world.

SMAF provides various features, including:

1. Secure memory allocation: SMAF enables the allocation of secure memory for Trusted Applications (TAs) and the OP-TEE OS within the secure world.
2. Memory isolation: SMAF ensures that memory allocated to a TA is isolated from other TAs and the OP-TEE OS, protecting sensitive data from unauthorized access or tampering.
3. Fine-grained access control: SMAF provides fine-grained access control over memory regions, allowing TAs and the OP-TEE OS to specify access permissions for different memory areas.
4. Efficient memory management: SMAF includes mechanisms for efficient memory management, such as memory pooling and caching, to optimize resource utilization and reduce memory fragmentation.

Overall, SMAF plays a critical role in maintaining the security and integrity of data within the OP-TEE environment. By providing secure memory allocation and management features, SMAF helps ensure that sensitive information remains protected from potential threats in the normal world and other trusted applications running in the secure world.

Question: `[17:45]` What is CMA? `(OP-TEE.0327.md)`

Answer:

CMA stands for Contiguous Memory Allocator, a feature in the Linux kernel that helps to allocate large contiguous blocks of memory. In systems where memory fragmentation is an issue, CMA can be particularly useful for devices and drivers that require large contiguous memory regions for their operation, such as multimedia hardware or devices using Direct Memory Access (DMA).

CMA works by reserving a portion of memory during the system boot process. This reserved memory region, known as the CMA pool, is kept separate from the general-purpose system memory. When a device or driver needs a large contiguous block of memory, it can request the allocation from the CMA pool. This approach reduces the likelihood of memory fragmentation and ensures that large contiguous memory regions are available when required.

CMA is not specific to OP-TEE but can be used in conjunction with OP-TEE in some cases. For example, OP-TEE may use CMA to allocate a large block of memory for a Trusted Application (TA) that requires a significant amount of contiguous memory. Additionally, CMA can be used to manage memory shared between the normal world and the secure world, ensuring that the shared memory regions are allocated contiguously and can be efficiently accessed by both worlds.

It is important to note that CMA should be used with caution, as reserving a large portion of memory for the CMA pool can impact overall system performance and memory utilization. Careful consideration of the trade-offs and system requirements is necessary when using CMA in a specific implementation.

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

Question: `[20:26]` Does Qualcomm support OP-TEE? `(OP-TEE.0327.md)`

Answer:

As of my knowledge cutoff in September 2021, Qualcomm does not officially support OP-TEE for their platforms. Instead, they provide their own Trusted Execution Environment (TEE) solution, known as Qualcomm Trusted Execution Environment (QTEE). QTEE is based on ARM TrustZone technology and is designed to provide a secure environment for running trusted applications on Qualcomm Snapdragon processors.

QTEE offers similar functionality to OP-TEE, providing a secure environment for running sensitive code and protecting sensitive data from unauthorized access or tampering. It includes features such as secure boot, secure storage, cryptographic services, and key management.

While Qualcomm officially supports QTEE on their platforms, it is possible that some developers or third-party companies have attempted to port OP-TEE to Qualcomm devices. However, such ports may not be officially supported by Qualcomm and may not offer the same level of integration, performance, or security as the QTEE solution.

Question: `[20:28]` Is QTEE open source? `(OP–TEE.0327.md)`

Answer:

Qualcomm Trusted Execution Environment (QTEE) is not fully open-source. While some components and tools related to QTEE are available as open-source projects, the core implementation of QTEE is proprietary and closed-source.

Qualcomm provides certain tools, libraries, and APIs to facilitate the development of trusted applications (TAs) for QTEE, and some of these resources may be available as open-source projects. However, the QTEE OS and other critical components are closed-source and maintained by Qualcomm.

This is in contrast to OP-TEE, which is an open-source TEE solution maintained by Linaro and supported by various industry partners. With OP-TEE, developers have access to the full source code, enabling them to examine, modify, and contribute to the project as needed.

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

Question: `[20:31]` What are Static TA and Dynamic TA? Please also summarize them in a table.`(OP–TEE.0327.md)`

Answer:

Static TA (Trusted Application) and Dynamic TA are two types of Trusted Applications used within the OP-TEE framework. Here’s a summary of their differences in a table:

Feature	Static TA	Dynamic TA
Loading mechanism	Built into the OP-TEE core	Loaded at runtime as separate binaries
Memory management	Pre-allocated memory within the TEE core	Allocated dynamically when the TA is loaded
Lifespan	Lives for the entire duration of the TEE core	Loaded and unloaded on-demand, based on the client’s request
Flexibility	Limited, as changes require recompiling the TEE core	Highly flexible, can be updated or replaced independently of the TEE core
Initialization process	Initializes during TEE core startup	Initializes when first invoked by a client
Performance	Faster, as no additional loading is needed	Slower, due to the need to load and initialize the TA at runtime
Storage	No separate storage required, as it’s part of the TEE core	Requires storage for the separate TA binary

Static TAs are built directly into the OP-TEE core binary, which means they are always available when the TEE is running.

They have pre-allocated memory within the TEE core and are initialized during the TEE core startup. This makes them faster in terms of performance but less flexible since any changes to a Static TA require recompiling the entire OP-TEE core.

Dynamic TAs, on the other hand, are loaded at runtime as separate binaries. They are more flexible and can be updated or replaced independently of the OP-TEE core. However, they may have a slightly slower performance due to the need to load and initialize the TA at runtime. Dynamic TAs are loaded and unloaded on-demand based on the client’s request and have memory allocated dynamically when the TA is loaded.