How to use MCE for encryption/decryption on STM32 MCUs

Introduction

Memory cipher engine (MCE) is a cryptographic peripheral that ensures on-the-fly encryption and decryption on external nonvolatile or volatile memories. MCE can protect either code or data located in the external memories. This document covers three use cases where the MCE can provide protection.

- Execute in place (XiP): Code and data storage are located in a nonvolatile external memory while the code is executing from an external flash memory.
- Load and run in external RAM (L&R_Ext): Code and data storage are located in an external flash memory that is copied into an external RAM to be executed. The MCE allows the protection of both the external RAM and the flash memories.
- Load and run in internal RAM (L&R_Int): Code and data storage are located in an external flash memory that is copied into an internal RAM to be executed. In this case, the MCE only enables the protection of the external flash memory.

The three use-cases are based on the examples in the STM32CubeH7RS firmware as references. They are based on ST immutable and OEM updatable root of trust boot solutions, referred to as STiRoT and OEMuRoT respectively.

Users can define their own root of trust boot paths based on customized OEM immutable and updatable root of trust paths (OEMiRoT + OEMuRoT). This document covers only the STiRoT+OEMuRoT boot path, and does not cover the custom defined OEMiRoT + OEMuRoT boot paths.

This application note applies to the products in the table below, which are referred as STM32H7Sx in the rest of this document.

Table 1. Applicable products

Type	Root part number
Microcontrollers	STM32H7S3A8, STM32H7S3I8, STM32H7S3L8, STM32H7S3L8U, STM32H7S3N8, STM32H7S3R8, STM32H7S3V8, STM32H7S3Z8 STM32H7S78-DK, STM32H7S7A8, STM32H7S7I8, STM32H7S7L8, STM32H7S7N8, STM32H7S7Z8

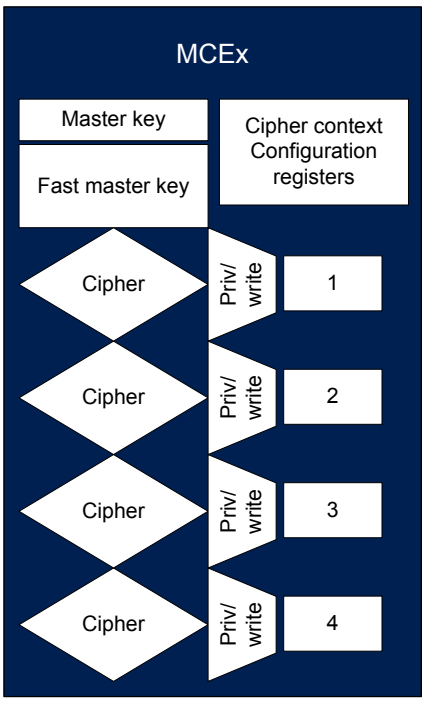
1 MCE (memory cipher engine) overview

The memory cipher engine (MCE) defines multiple regions with a specific security setup in a given address space. These security setups can refer to encryption, privilege, or write. All system bus traffic going through an encrypted region is managed on-the-fly by the MCE, automatically decrypting reads and encrypting writes if authorized.

Up to four regions can be defined on each MCE, with a granularity of 4 Kbytes. MCE supports the access-filtering feature that can be applied in different regions (privileged, write). Refer to the figure below.

The MCE supported features are listed in the table below. When a stream cipher is selected, it is important to activate the write protection as soon as the whole region has been encrypted (read-only region).

Figure 1. MCE block diagram



DT74414V1

The STM32H7Sx implementation defines three MCE instances connected to the external memory interfaces as depicted in Figure 2. When the MCE is used in conjunction with an XSPI, it is mandatory to set the flash memory controller in memory map mode. It is also required to use the DMA to perform writes to the flash memory using 16-bytes bursts. Table 2 details the main implementation differences between the three MCE instances.

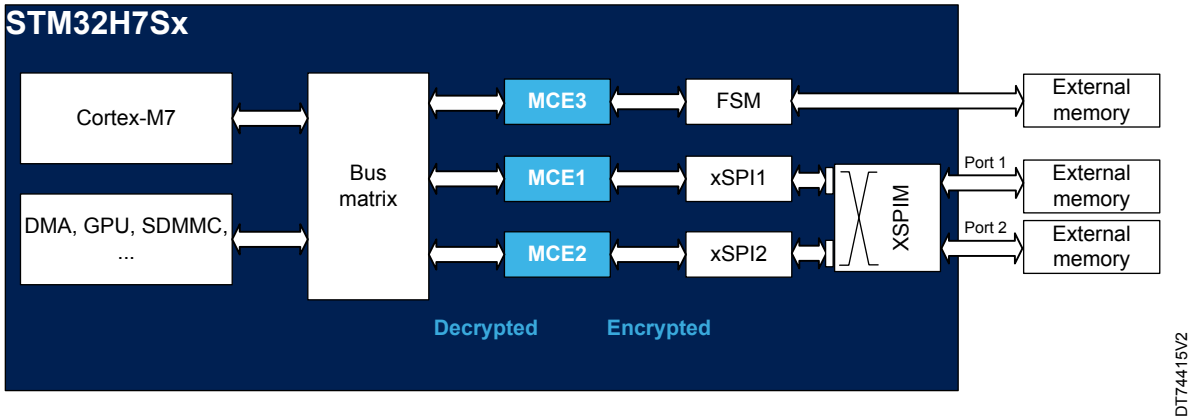
Table 2. MCE Main features

MCE features	MCE1	MCE2 and MCE3
Number of regions	4	
Cipher engines	AES	Noekeon
Derive key function	Normal, fast	
Master key	2	
Encryption mode	Block, stream	Block

2 MCE implementation in STM32H7Sx

The STM32H7Sx implementation define three MCE instances connected to the external memory interfaces as depicted in Figure 2. When MCE is used in conjunction with XSPI, it is mandatory to set the flash memory controller in memory map mode. It is also required to use the DMA to perform writes to the flash memory using 16-bytes bursts.

Figure 2. STM32H7S MCE instances



In the STM32H7Sx implementation (Figure 2) the MCE protects the confidentiality of code or data stored in the external memory (four regions/memory). The MCE peripheral instances are connected to the xSPI peripheral instances as follows:

- MCE1 is connected to the xSPI1 memory interface, which can be connected to a NOR flash or a PSRAM.
- MCE2 is connected to the xSPI2 memory interface, which can be connected to a NOR flash or a PSRAM.
- MCE3 is connected to the FMC memory interface, which can be connected to a PSRAM, an SDRAM, or a FRAM.

In Figure 2, the xSPI1 and xSPI2 are connected to an XSPIM (xSPI IO manager) which allows a multiplexed connection of external memories (NOR flash or PSRAM). Several use cases can be considered as follows (refer to the *XSPIM* section of RM0477 for more details):

- XSPI1 mapped to port 1, with XSPI2 mapped to port 2 (direct mode).
- XSPI1 mapped to port 2, with XSPI2 mapped to port 1 (swapped mode).
- Two XSPIs drive one external memory.
  - XSPI1 and XSPI2 both mapped to port 1, with arbitration (multiplexed mode to port 1).
  - XSPI1 and XSPI2 both mapped to port 2, with arbitration (multiplexed mode to port 2).
- One XSPI drives two external memories.

MCE encryption/decryption must take the corresponding IO manager use case/mode into account before any encrypted images installation. The STM32CubeH7RS firmware provides templates according to the two first use-cases of XSPIM.

2.1 MCE1 implementation

The confidentiality of MCE1 is based on legacy AES 128-bit cipher block (ECB) or stream (counter CTR) mode.

2.1.1 Block mode

The MCE1 block mode is based on the AES cipher that is compatible with the ECB mode specified in the NIST FIPS publication 197 advanced encryption standard (AES). The **new** AES 128-bit block cipher is used, the AES-ECB with the key derivation function based on the well-known Keccak-400 algorithm where each 256-bit word has its own AES key.

The block-cipher mode with key derivation, normal or fast, supports the partial encryption region updates where any part of the encrypted region can be updated anytime. As a block cipher it is resilient against bit-flip attacks.

The block-cipher mode key derivation can be either normal or fast (refer to RM0477 for more details).

Since the **normal** key derivation function is leakage resilient, the master key information in normal mode is protected against side channel attacks (SCA). When a tamper event is confirmed in TAMP, all MCE keys are erased.

In **fast** mode the SCA protection is not supported, *hence the master key used in normal mode is never used in fast mode.*

2.1.2 Stream mode

The MCE1 stream mode is based on the AES cipher that is compliant with the CTR mode specified in the *NIST SP800-38A Recommendation for block cipher modes of operation*. The **legacy** AES 128-bit counter cipher mode is used (stream). It has the lowest latency, but it has limitations in terms of security, since no side channel protection, or protection against bit-flip attacks is provided.

In stream cipher mode the partial update of encrypted region is not recommended. Each time the content of one encrypted region is changed the whole region must be reencrypted using a new key, a new 16-bit version, or a 64-bit nonce.

2.2 MCE2 and MCE3 implementation

MCE1 and MCE2 are based on the Noekeon algorithm (refer to https://gro.noekeon.org) that supports the block ciphering mode with normal or fast key derivation.

The block-cipher mode with key derivation (normal or fast) supports the partial encryption region updates where any part of the encrypted region can be updated anytime. As a block cipher it is resilient against bit-flip attacks.

The MCE1 and MCE2 **normal** key derivation mode is leakage resilient. The master key information in normal mode is protected against side channel attacks (SCA). When a tamper event is confirmed in TAMP, all MCE keys are erased. In **fast** mode the SCA protection is not supported, *hence the master key used in normal mode is never used in fast mode*, in MCE2 and MCE3.

2.3 Selecting an MCE configuration

The MCE configuration to be used depends on many considerations as follows:

- The implementation path: each MCE is connected to a fixed external memory interface (xSPI1, XSPI2, or FMC). Each MCE supports a set of memory technologies (volatile or nonvolatile). The MCE1 is connected to xSPI1. The MCE2 is connected to xSPI2. The MCE3 is connected to FMC.
- The memory type: volatile or nonvolatile. In addition, memories connected to xSPI1 or xSPI2 can be multiplexed via the XSPI IO manager (XSPIM) as depicted in Figure 2.
- Priority between security and performance:
  - *The Noekeon cipher has a security level of ~96 bits, lower than the 128 bits of the AES-128 cipher.*
  - Block or stream cipher modes: security and performance compromises must consider that the MCE stream cipher mode is faster than the block cipher mode, but the latter is more secure than the former. Indeed, block cipher provides protection against bit-flip attacks. And write protection must be used after encrypting an image with stream cipher (to prevent encrypting multiple times with same key and IV).
  - Normal or fast key derivation in block-cipher mode: key derivation can be normal giving higher protection with SCA resistance or fast, giving better performance without any SCA resistance. *Hence, the master key used in normal mode is never used in fast mode.*
- Partial updates: in block cipher mode any part of the encrypted region can be updated anytime whatever the key derivation: normal of fast. This feature is not supported in stream cipher mode.
- When the MCE is used in conjunction with xSPI1 or xSPI2, it is mandatory to read or write the flash memory using the memory map mode of the flash memory controller.

Table 3 summarizes the latency in cycle for 16-bytes data.

Table 3. MCEs performances comparison

Mode (1)	MCE1 AES-block	MCE2/3 Noekeon-block	MCE1 AES-stream
Normal	25 (2)	21	11 (masked)
Fast	15 (2)	11	N/A

1. *Latency in cycle for 16-bytes data.*
2. *Add 10 for writes.*

Table 4 summarizes the recommended MCE1 configurations to use regarding security protection and performance scales.

Table 4. MCE1 configuration selection

Selection criteria	Recommended MCE configuration	Selected configuration features
Security	• MCE1 block cipher mode, • Normal key derivation	• AES-128 bit ECB • Keccak-400 key derivation • SCA resistant • Partial encryption updates • Suitable for code+data encryption
Performance Non-SCA protection	• MCE1 block cipher mode • Fast key derivation	• AES-128 bit ECB • Fast key derivation • No SCA resistance • Partial encryption updates • Suitable for code + data protection
Performance	• MCE1 stream • No key derivation (n/a)	• AES-128 CTR • No SCA resistance • No partial encryption updates • Suitable for code protection

MCE2 and MCE3 are based on Noekeon with a security level of ~96 bits versus 128 bits for AES-128 bits in the case of MCE1. When a security level is key, the user can move to MCE1 configurations in Table 4.

Table 5 summarizes the possible configurations when MCE2 or MCE3 are used.

Table 5. MCE2 and MCE3 configuration selection

selection criteria	Recommended MCE configuration	Selected configuration features
Security	• MCE2 or MCE3 block cipher mode • Normal key derivation	• Noekeon block mode • Normal key derivation • SCA resistant • Partial encryption updates • Suitable for code + data encryption
Performance	• MCE2 or MCE3 block cipher mode • Fast key derivation	• Noekeon block mode • Fast key derivation • No SCA resistance • Partial encryption updates • Suitable for code + data protection

3 MCE configuration and external flash regions

The STM32H7Sx devices provide a small-sized internal user flash of 64 Kbytes. When the STiRoT is used, the internal user flash must hold the external memory manager/iLoader, for immutable loader, provided by ST. This middleware allows access to all types of external memories, and launches an application that is stored in an external memory. This file is used by the external memory manager/iLoader to copy or to load the application firmware (refer to Figure 3).

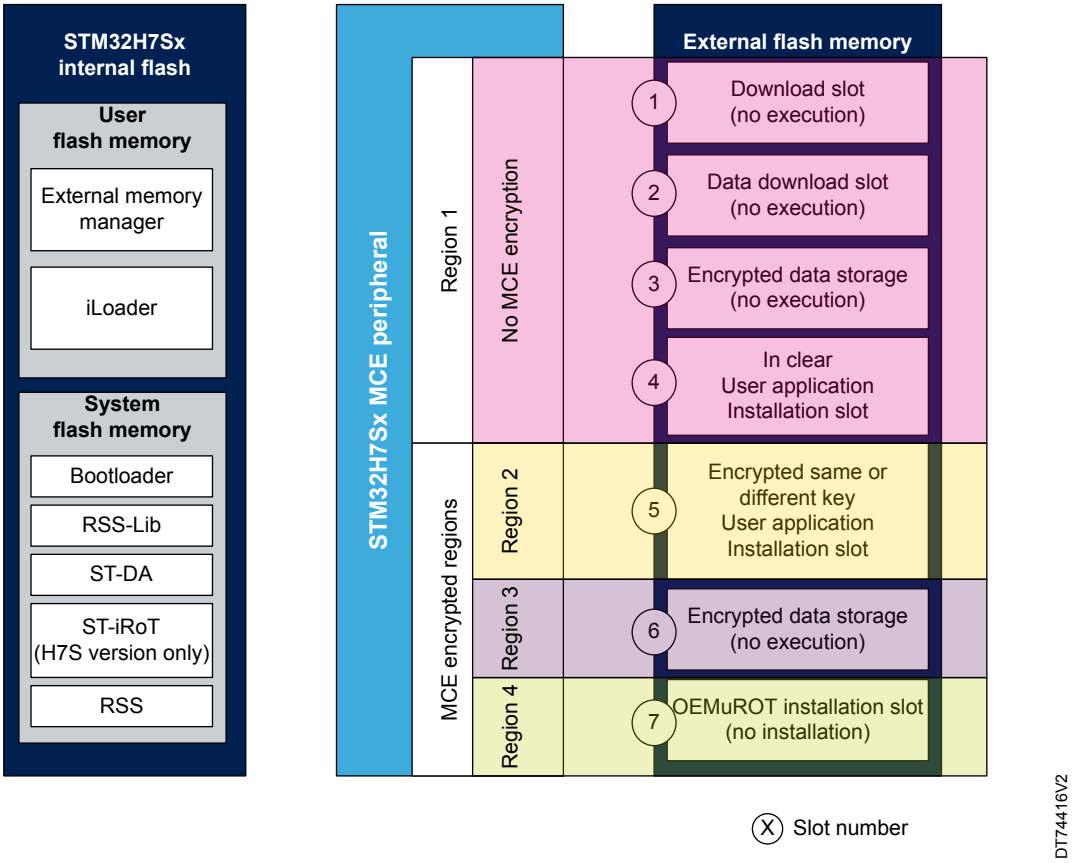
The STM32H7Sx devices also feature 128 Kbytes of internal flash as system flash memory. It is intended to integrate, in addition to the system configuration options and the bootloader, a set of security services. These security services are used to protect the final product assets and the application firmware as illustrated in Figure 3.

- ST immutable RoT (STiRoT): manages the secure boot and the secure update of the first stage of the application. It is present in the STM32H7Sx only as it is based on the hardware crypto accelerators.
- Debug control (ST-DA): debug reopening and regression controls.
- Secure services (RSS): secure firmware installation (SFI), ST immutable code allowing to install securely an extension (RSS-e).
- RSS-Lib: an RSS extension library that provides a set of APIs to manage the temporal isolation (HDPL levels), to control the product life cycle (using GetProductState() and SetProductState() functions), and to provision the option byte keys (OBKey).

The figure below provides a generic mapping of an external flash managed by the MCE encryption regions. This figure provides different slot possibilities to hold code and data storage. It can be customized according to user needs.

- In region 1: no MCE encryption. The download slots are used to host images that are already encrypted (with transport keys, not with MCE keys). When confidentiality is not required, there is no need to encrypt the user application. Data storage can be cleared or encrypted with DHUK keys using the SAES.
- In region 2: MCE encryption/decryption of the user application in the installation slot can be managed with the same key for all STM32H7Sx devices, or with a different key per device.
- In region3: MCE manages encrypted data storage in external memory. Region 3 can be protected with different MCE keys per STM32H7Sx device.
- In region 4: the MCE can manage the encryption in the installation slot with different keys per device of the first boot level.

Figure 3. **Typical internal and external flash mapping versus MCE regions**



The selection of the MCE configuration depends on the degree of performance and security required by the customers as described in Section 2: MCE implementation in STM32H7Sx. Hereafter, some recommendations of MCE configuration regarding the MCE encrypted regions 2, 3 and 4.

Regions 2 and 4

Regions 2 and 4 perform the encryption of code in the external flash slot #5 and slot #7, the user application installation slot and the OEMuRoT installation slot respectively.

- Encryption of user application slot #5 can be performed using either the same key for all STM32H7Sx devices or a different/unique MCE key per device.
- Encryption of updatable root of trust slot #7 with the same or different key depends on the OEMuRoT firmware implementation as detailed in Section 5: MCE protection in case of XiP. A different key is recommended in the case of the first boot level of this uRoT installation slot #7.
- **[MCE1 AES-block mode + normal]** for security: as detailed in Section 2: MCE implementation in STM32H7Sx, it is recommended to configure MCE1 in AES block ciphering with normal key derivation. This configuration is the best choice for security reasons, as it brings the SCA resistance of key derivation combined with AES-128.
- Selecting the **[MCE1 AES-block mode + fast]** configuration loses the SCA resistance of key derivation.
- **[MCE1 AES-stream]** for performance: when performance is a matter, the MCE1 configuration with stream gives the best performance. In that case, write protection has to be used, but it is not recommended to partially update the encryption of these regions. As a consequence, this stream mode is only applicable to code.
- **[MCE2 or MCE3 Noekeon-block]:** if MCE2 or MCE3 are used to protect an image, only Noekeon-block is available. Then normal or fast key derivation can be selected depending on security/performance targets. In normal mode the key derivation is SCA resistant, however the fast mode is not.

Region 3

Region 3 is dedicated to encryption of data storage in the external flash slot #6.

- Encrypted data storage in external memory can be protected with MCE/different keys.
- **[MCE1 AES-block mode + normal], [MCE1 AES-block mode + fast] and [MCE2 or MCE3 Noekeon-block]** configurations are the same as described in region 2 and 4 above. Encrypted regions can be partially updated.
- **[MCE1 AES-stream]** partial updates of an encrypted region can be an issue, and is not recommended.

4 Recommended keys for external memory protection

External memory protection can be encrypted by MCE, by transport, or by DHUK keys. This section describes the recommended encryption approach.

In volatile external SRAM memories, it is recommended to use dynamic values for keys as they usually store dynamic content. For example, keys that are different for each initialization can be used. This dynamic key can be a different key for each STM32H7Sx, which can be based on the application hardware key mechanism (AHK).

In the nonvolatile (NV) external flash memories, encryption is done only when needed. Different encryption key approaches can be considered as stated below:

- Images that are not encrypted by MCE keys but are encrypted at application level, using SAES for example.
- Images encrypted with MCE using the same key for all the STM32H7Sx devices.
- Images encrypted with MCE using a different key for each STM32H7Sx device.
- External memories are not erased by DA-regressions. When confidentiality protection is needed, the MCE encryption is recommended for all data storage and code that need to be erased from a full regression.

A different key per device is preferable for many reasons, for example the ones listed below:

- If all devices have the same key, the security of all devices is compromised in case of key leakage.
- With a different key per device, an external memory device is assigned to one product only. It is not possible to reuse an external memory device in a second product as the second STM32H7Sx is not provisioned with the same key.
- In the case of reverse engineering, it is not possible to reuse the external flash in other environments, making the reverse process very complicate.

In NV-external memories, the choice of “same” or “different key per device” depends on both initial provisionning, and on first boot timing constraints:

4.1 MCE usage with a different key per device

Consider the typical case where different keys are used for the firmware install slot #5 hosted in an external NV memory. As the keys are different from one device to another, at manufacturing the firmware is loaded in the download slot #7. It is then installed at the next boot of the product. Depending on the firmware size, this scheme could impact the first boot time when the image size is too large.

During manufacturing or during the initial programming of the images, the provisioning can be done with GANG programming or with an external flash loader.

When a different key per device is used, the binary to install can be flashed into the slot #1 download slot. The binary to install can be encrypted with the firmware update transport key. Then at the first time the OEMuRoT is executed, the installation of the image is performed via the OEMuRoT.

4.2 MCE usage with the same key for all devices

Same key for all devices: when the same key is used on all devices for the firmware install slot #5, then encrypted firmware can be installed directly in this install slot #5. Consequently, there are no first boot timing constraints.

During manufacturing, or during the initial programming of the images, provisioning can be done with GANG programming or with an external flash loader.

When the same key for all devices is used, the binary image can be flashed into slot #5 firmware installation slot. The binary is preencrypted with tools in a trusted environment or in an untrusted environment using SFI.

When using the same MCE key for all devices for a user application, the firmware can be installed directly preencrypted in its installation slot #5. The advantages are related to both security and initial boot time:

- (+) Security: the firmware is shared encrypted.
- (+) Boot time: the firmware is executed on the next boot. There is no time lost to install the firmware during the first boot then executed at a second boot.

On the other hand, when using the same MCE key for all devices for user application, all the devices are provisioned with the same key. This implies two major security drawbacks:

- (-) Security: if the key leaks, the confidentiality of the firmware or the data that it is supposed to protect is compromised.

- (-) Security: if all the devices use only the same key(s), it means that there is no protection for external memory exchanges. Besides, the encrypted image loaded into one external memory is the same for all other external memories.

5 MCE protection in case of XiP

When STiRoT is used as the execution entry point, the MCE usage in a XiP use-case can be described as follows. The MCE is configured according to four regions to control the security of the external flash memory.

The mapping of the external flash versus the MCE regions is depicted in Figure 3. The first level of protection is to define which regions have MCE encryption, and which ones do not have it. Then for each region, an MCE or non-MCE based key is defined, summarized in are summarized in Table 6.

- Region without MCE encryption: region 1.
  - Download slot: used to host already encrypted images with transport keys, no need to encrypt with MCE.
  - Data download slot, it is also encrypted with transport keys if needed, or kept in clear.
  - Encrypted data storage: can be protected by the DHUK key and encrypted using the SAES.
  - In a clear user application installation slot: no encryption of user application when no confidentiality is required.
- Regions with MCE encryption protection:
  - Region 2: user application installation slot. It can be encrypted with the same MCE key or with a different key per device.
  - Region 3: encrypted data storage in external memory can be protected with MCE, with the same key, or with a different key per device.
  - Region 4: installation slot of the first boot level where the OEMuRoT (that acts as the second boot stage) is installed. This slot is always encrypted using MCE keys that are different per STM32H7Sx device, as mentioned above.

All the configurations of STiRoT and OEMuRoT are set in HDPL1 and HDPL2 respectively. In this way, when the application is being executed in HDPL3, all the keys used to protect the external flash are automatically hidden for the application and protected.

With this typical external flash scheme, there are two sets of MCE keys that are used:

- The MCE encryption keys that are managed at HDPL1 for STiRoT:
  - STiRoT is responsible for the secure boot secure firmware update of the next level (OEMuRoT or application). It uses an “install slot” key to access the next level and “transport keys” to manage decryption of updates.
    - When OEMuRoT is acting as a second boot stage, the STiRoT configures the MCE in decryption mode with an “install slot” key. This action allows the external memory manager/iLoader to load the OEMuRoT image into the external SRAM, before launching it.
    - In case of an update of the OEMuRoT version, the STiRoT decrypts the new image using the MCU boot STiRoT keys (transport keys of the image). Then STiRoT checks the image authenticity and integrity. If validated, the STiRoT configures the MCE in encryption mode using the MCE “install slot” keys. Finally, the external memory manager/iLoader installs the encrypted image using HDPL1 MCE keys into the external flash at the OEMuRoT installation slot.
- The MCE encryption keys are provisioned at HDPL2 for OEMuRoT:
  - OEMuRoT is responsible for the secure boot secure firmware update of the next level (application). It uses an “install slot” key to access the next level and “transport keys” to manage decryption of updates.
    - The same flow as STiRoT is used to launch the application image or to install a new application version into the application installation slot.
    - In installing the new application version, OEMuRoT checks if a new user application is stored in the download slot. If there is any, OEMuRoT decrypts it with a “transport key” and controls its integrity, its authenticity, and its antirollback version. If successful, OEMuRoT configures the MCE with “install slot” encryption keys. The image is reencrypted by the MCE and copied in the user application installation slot.
    - When launching the user application, OEMuRoT controls the integrity and the authenticity of the user application from the installation slot. If successful, OEMuRoT executes it from external flash memory based on the MCE decryption using the MCE “install slot” keys configured by the OEMuRoT.

The following table summarizes the set of keys used to protect this typical scheme of the external flash memory:

Table 6. External flash slot protection with MCE and non-MCE keys

Slot index	External flash memory slot	MCE key protection	Non-MCE key protection
1	User or OEM download slot (no execution)	N/A (no need to encrypt with MCE for protection)	• AES encryption with MCUBoot STiRoT keys (update transport keys) when a new OEMuRoT image is available. • AES encryption with MCUBoot OEMuRoT keys (update transport keys) when a new user-application is available.
2	Data download slot (no execution)	N/A (no need to encrypt with MCE for protection)	AES encryption using one of SAES or AES source. Managed by the firmware accessing them.
3	Encrypted data storage (no execution)	N/A (no need to encrypt with MCE for protection)	Encrypted data storage in external memory can be protect with DHUKs keys using SAES.
4	In clear User application Installation slot	N/A (no need to encrypt with MCE for protection)	No encryption of user application when no confidentiality is required.
5	User application Installation slot	• Encryption with “installation slot” key with a different key (per product key) or with the same key. • Different keys: derived from HUK (AHK, application hardware key, secret to software). • Same key: stored in OBKeys. • AES “installation slot” key managed by OEMuRoT (HDPL2).	-
6	Encrypted data storage (no execution)	Encrypted data storage in external memory can be protected with different MCE keys (HDPL2).	-
7	OEMuRoT installation slot (no execution)	Encryption with “installation slot” key, always “different” with STiRoT. The key is controlled by STiRoT.	-

The provided use cases in the STM32CubeH7RS firmware are based on the STiRoT + OEMuRoT. The latter should be installed first in the OEMuRoT installation slot. In this use case, the STiRoT can only manage OEMuRoT images encrypted with different MCE keys.

When running from the internal SRAM the OEMuRoT manages to install a new user application or to launch the application available in the application download slot. By default, the OEMuRoT uses “images encrypted with different MCE keys “, but the user can customize the code to use the same key.

The choice to user different keys or the same key can be performed by firmware configuration in the STM32CubeH7RS firmware.

6 MCE protection in case of “load and run”

In a “load and run” scheme (L&R), there are two use cases for MCE usage in the STM32CubeH7RS: L&R from external RAM, and L&R from internal RAM (refer to Figure 4 and Figure 5).

For the use case of L&R from external RAM, the following rules apply:

- The execution from external RAM recommends the usage of keys that are different for each initialization. Dynamic external SRAM content always requires dynamic keys for protection. For this reason, different AHK keys per device are used in either:
  - Encrypted data storage slot (MCE-AHK1) or
  - Encrypted user application slot (MCE-AHK2) or
  - The same MCE-AHK for both data storage and user application.
- If the content is stored encrypted into the external flash slots, the user application in the L&R context must load, verify, and run it from the external RAM. The user application must have an extra code that gets the slot information, loads it into the external RAM and manages to verify and launch it. This is also applicable for encrypted content to be loaded into internal RAM in the context of L&R form internal RAM.

Figure 4. MCE usage in the context of L&R from external RAM

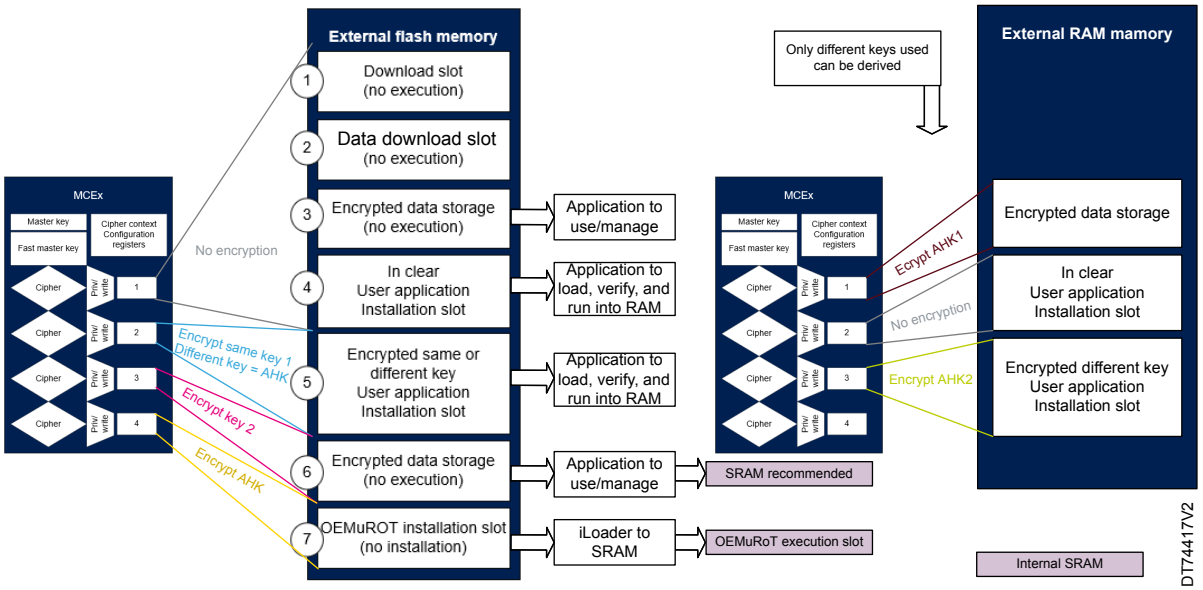
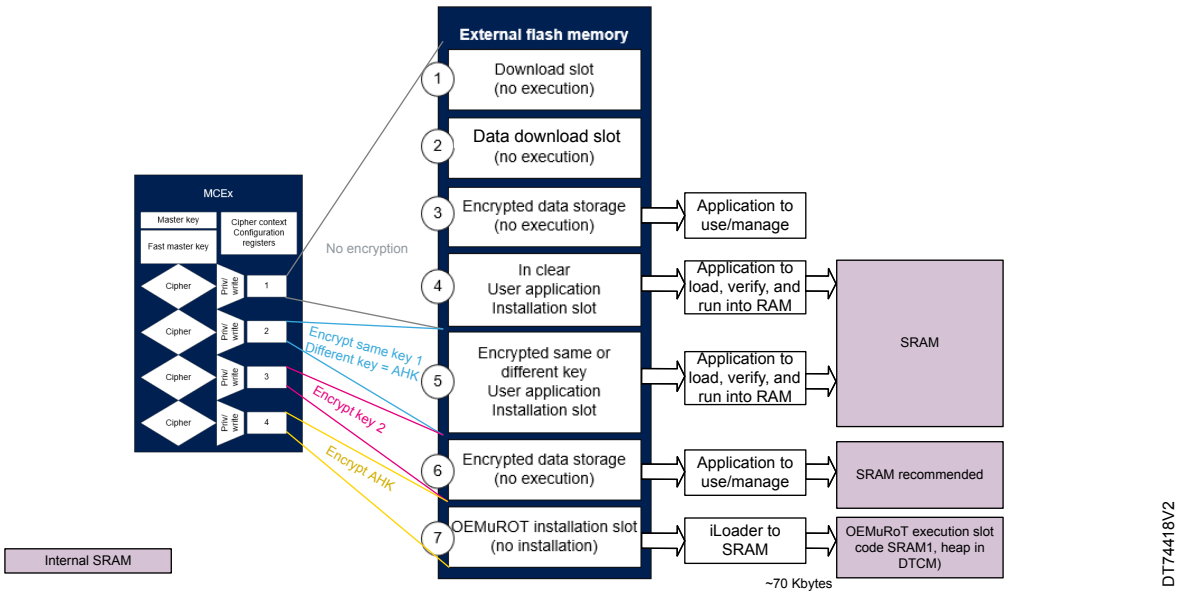


Figure 5. MCE usage in the context of L&R from internal RAM



In the context of L&R from external RAM (Figure 4) the slots of the external flash memory are loaded as follows into the internal or the external RAM:

- The OEMuRoT, used in case of boot or update, is always decrypted and loaded into the internal RAM. It is done using MCE STiRoT different keys per device. The same approach is followed in the contexts of L&R from internal RAM and of XiP.
- It is recommended to load the “encrypted data storage” slot 6, based on MCE key encryption, into the internal RAM. It is up to the application to manage the decryption while it is loading this slot from the external flash to the internal RAM. This rule is valid for both L&R from internal RAM, and L&R from external RAM.
- The “encrypted data storage” slot 3 with data security based on SAES with DHUK and no MCE encryption, is loaded into the external RAM. The access is managed by the user application through MCE.
- It is up to the application to load, verify, and run into the external RAM:
  - The “encrypted user application” slot 5 loaded into the MCE region encrypted using and “AHK different keys per device” or a derived key.
  - The “in clear user application” slot 4 loaded into the MCE region without encryption.

In the context of L&R from internal RAM (Figure 5), all installation slots of the external flash, including the OEMuRoT execution slot, are loaded into the internal RAM. This use case is simpler and faster than the L&R from external RAM. However, the user must consider that there is a maximum of 456 Kbytes of contiguous SRAM available per product, minus ~70 Kbytes already reserved for the OEMuRoT execution slot.

7 MCE key provisioning with different keys

The figure below shows how the MCE keys are provisioned for STiRoT in HDPL1, and for OEMuRoT in HDPL2. The keys are generated by calling the `RSSLib->DataProvisioning()` API. This API on the first call generates and stores keys: AHK for OBKeys, AHK for MCE1, and AHK for MCE2.

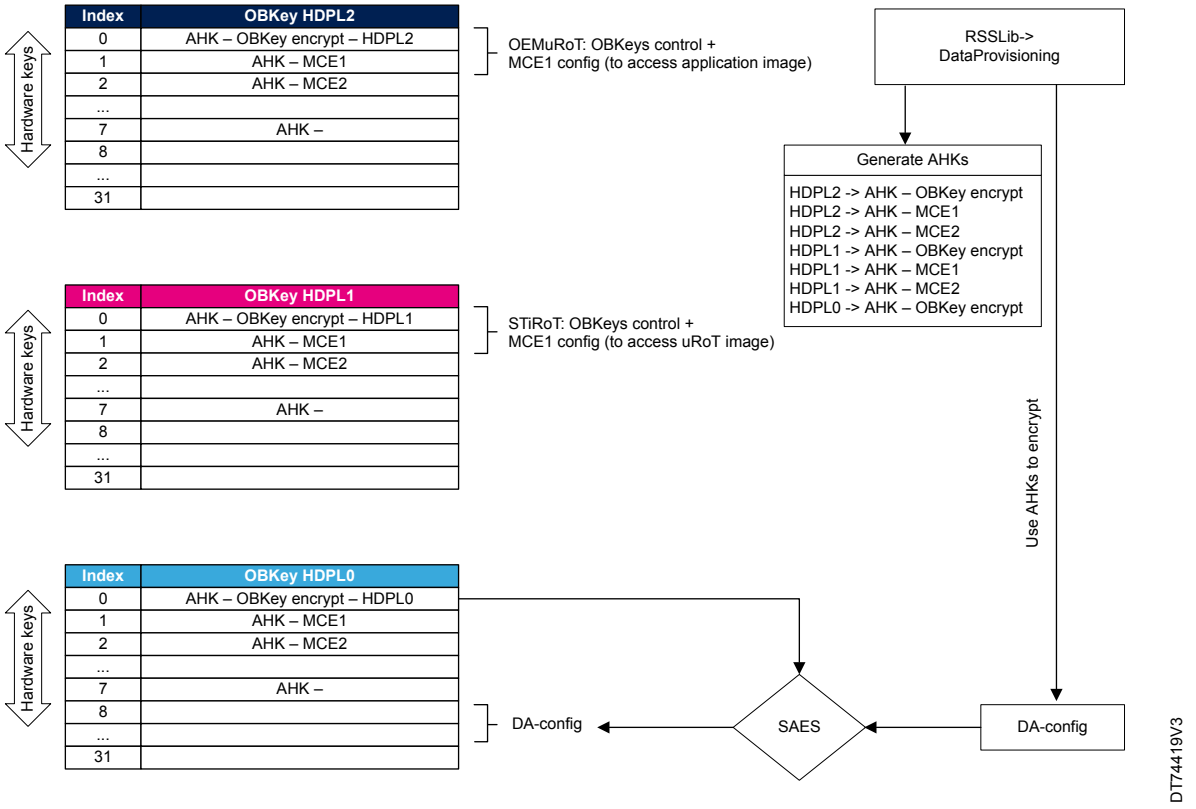
The MCE keys are then stored into OBKey HDPL1 for STiRoT usage and in HDPL2 for OEMuRoT usage. STiRoT is installed by default into the system flash for all STM32H7Sx devices. It is designed to use only these randomly generated keys, which explains why STiRoT MCE keys are always different per product or device.

OEMuRoT uses the randomly generated AHK-MCE1 in HDPL2, which explains why these OEMuRoT keys are by default different per product. However, this can be updated with a new version that is customized to provision the same MCE key for all products.

The HDPL temporal isolation enables the advantage of having keys provisioned for HDPL lower-level automatically protected in higher HDPL level. In the MCE usage, the configuration of the MCE to protect the OEMuRoT image is set in HDPL1 by STiRoT. When the OEMuRoT is running in HDPL2, it has all MCE keys automatically protected by the HDPL1.

In addition, the configuration of MCE to protect the user application image is set in HDPL2 by the OEMuRoT. All the applications running in HDPL3 from external flash memory have all their MCE keys automatically protected by HDPL2.

Figure 6. AHK-MCE provisioning



8 Debugging application in the MCE use cases

Debugging an application, when MCE must be used, depends on the life cycle state of the product. When the state is locked or closed, the product is at the final state with the root of trust. The debug is either not possible (locked), or can be opened using the debug authentication (DA) with certificates.

Debug on HDPL3 can be performed without compromising the MCE keys on a closed product thanks to the debug authentication.

Debugging a product during development, in open state, when MCE encryption is required, can rely on MCE setup using development keys. The setup of MCE can be done by simple initialization of the MCE key registers, regions, and context as done by the MCE example provided in the STM32CubeH7RS V1.0.0 or higher. A customized setup method is also possible, for example by calling a service loaded and executed from the SRAM.

Debugging a product in the field, when the state is closed and the MCE encryption is required, can rely on the MCE setup in HDPL1 (STiRoT AHK-MCE keys) or HDPL2 (OEM-uRoT AHK-MCE or OEMuRoT same key). The user must use the debug authentication to open debug for HDPL3, and set a breakpoint in the user application (HDPL3). After reset, the execution goes through the RSS, then through the STiRoT (HDPL1) to launch the OEMuRoT (HDPL2) which then launches the user application in HDPL3. Accordingly, it is possible to debug the user application code as soon as the execution starts in HDPL3. With this option, the MCE keys are protected as they are managed by the HDPL2.

9 MCE use cases in manufacturing

The STM32H7Sx provisioning and programming steps covered during manufacturing depend on whether the manufacturing environment is trusted or not. The secure firmware installation (SFI) is recommended in untrusted manufacturing environment and not required when the environment is trusted.

The AN6103, "*Third party programmers guide for the STM32H7R/S MCUs*", can help the third party companies specialized in programmers to understand the programming and loading steps covered during the manufacturing of STM32H7R/S products. (An NDA is required to have access to AN6103).

Revision history

Table 7. **Document revision history**

Date	Version	Changes
09-Apr-2024	1	Initial release.

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved