User manual

STM32MP13xx security guidance for SESIP level 3 certification

Introduction

This document describes how to prepare an STM32MP13xx microprocessor to make a secure system solution compliant with SESIP level 3.

The security guidance described in this document applies to any board based on the devices listed in the table below, for die revision Y.

Table 1. Applicable products

Reference	Products
STM32MP13xx	STM32MP131C, STM32MP131F, STM32MP133C, STM32MP133F, STM32MP135C, STM32MP135F

1 General information

This document applies to STM32MP13xx Arm®-based MPUs.

The SESIP3 certification includes the following functionality:

- Secure boot
- Residual information purging
- Cryptographic operations, including AES, RSA, ECDSA, ECDH, ECIES, SHA-2, and SHA-3
- Cryptographic key storage

Any references to non-certified functionality within this guidance should be considered as recommendations.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

Table 2. **Specific acronyms**

Acronym	Description
BHK	Boot hardware key
GPT	Global partition table
HUK	Hardware-unique key
MPU	Microprocessor
OEM	Original equipment manufacturer
PCI	Payment card industry
RMA	Return material for analysis
SCA	Side-channel attack
SESIP	Security evaluation standard for IoT platforms
SFR	Security-functional requirement
TOE	Target of evaluation

2 Reference documents

- Reference manual *STM32MP13xx Reference Manual* (RM0475) - revision 4
- Application note *Overview of the secure secret provisioning (SSP) on STM32MP1 Series* (AN5510) - revision 3
- User manual *STM32CubeProgrammer software description* (UM2237) - revision 28
- User manual *STM32 Trusted Package Creator tool software description* (UM2238) – revision 18
- User manual *STM32MP1 Series Key Generator software description* (UM2542) – revision 5
- User manual *STM32MP1 Series Signing Tool software description* (UM2543) -- revision 6
- Errata sheet *STM32MP131x/3x/5x device errata* (ES0539) - revision 6
- Wiki page Populate the target and boot the image -- revision 3.0.0 (June 27th, 2025)
- Wiki page Getting started -- revision 3.1.0 (July 3rd, 2025)

3 TOE preparative procedures

The target of evaluation (TOE) consists of the following components:

- The STM32MP13xx MPU device hardware, defined in Table 1
- The boot ROM, embedded in the device

The TOE serves as a Root of Trust (RoT) for the IoT solution running on top of it.

This chapter describes the procedures to prepare the environment and the TOE before starting to use the device or before testing the IoT product:

- Secure acceptance: procedures to check the device to be tested
- Secure preparation of the operational environment: procedures to set up the environment needed to manage and test the IoT product.
- Secure installation: procedure to program and configure the IoT product to be tested

3.1 Secure acceptance

Secure acceptance is the process in which the user securely receives the TOE and verifies its genuineness.

The TOE is distributed as an STM32 MPU device, with a software package that can be obtained from www.st.com. Refer to the cover page for the applicable devices.

How to accept an STM32MP13xx MPU device

When the device is in the OTP-SECURE Open default state, TOE genuineness can be verified using a debugger, reading data `0x10036501` at address `0x50020380`. More specifically:

- Bits [11:0] returns the die ID (`0x501` for STM32MP13xx MPU)
- Bits [31:16] returns the revision ID (`0x1003` is silicon revision 1.2)

Additionally, the device part number must be read at adress `0x5C005204`. Expected RPN values are:

- `0x6C8` (STM32MPI31C) or `0xEC8` (STM32MP131F)
- `0x0C0` (STM32MPI33C) or `0x8C0` (STM32MP133F)
- `0x000` (STM32MPI35C) or `0x800` (STM32MP135F)

Alternatively, the following methods can be used with any board mounted with the TOE:

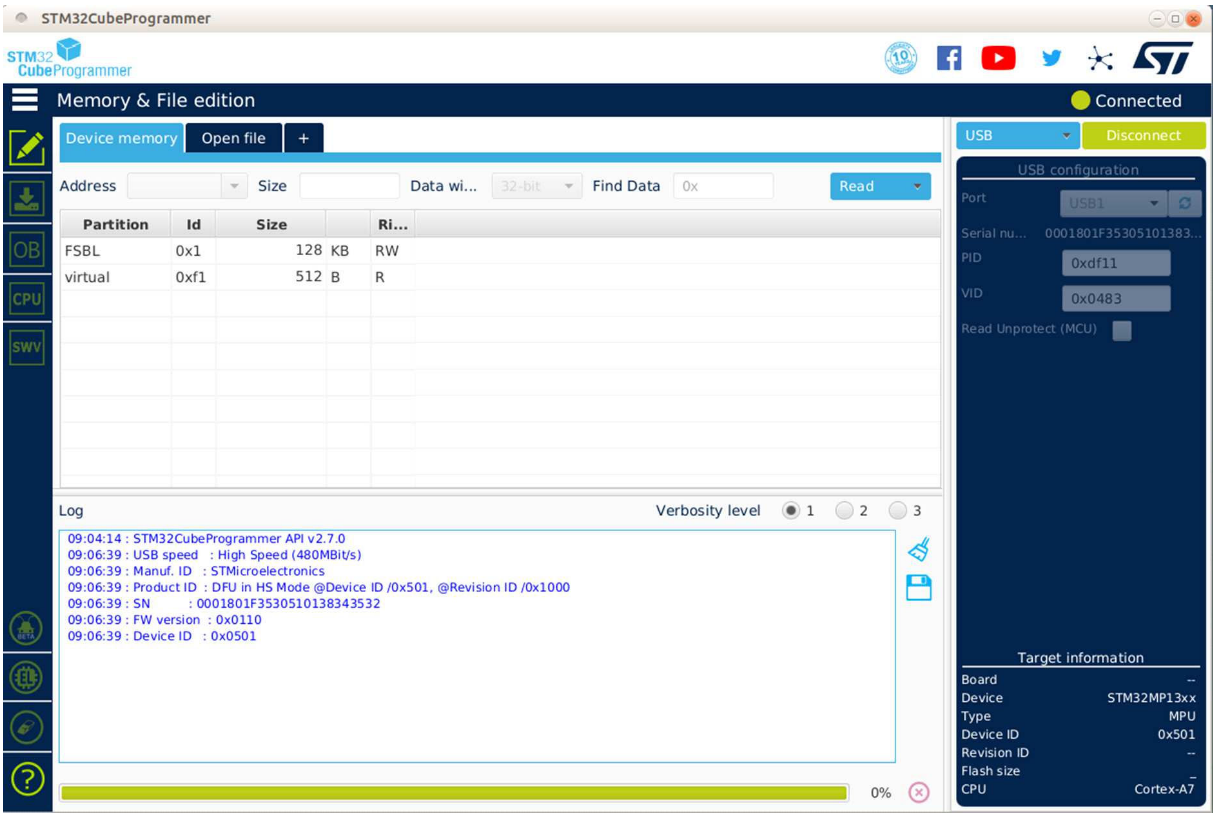
- Set the boot switches to the off position (bootpin[2:0]=000).
- Connect the USB Type-A host port of the board to the user's computer.
- Power up the board and press the reset button.
- Use the STM32CubeProgrammer command-line interface (CLI):
  - Enter: `./STM32_Programmer_CLI -c port=usb1 -p.`
  - Expect the below answer from the device, prepared by ROM code.

```
----------------------------------------------------------------
STM32CubeProgrammer v2.7.0
----------------------------------------------------------------
USB speed : High Speed (480MBit/s)
Manuf. ID : STMicroelectronics
Product ID : DFU in HS Mode @Device ID /0x501, @Revision ID /0x1003
SN : 0001801F3530510138343532
FW version : 0x0110
Device ID : 0x0501
Device name : STM32MP13xx
Device type : MPU
Device CPU : Cortex-A7
PhaseID: 0x01
```

- We find in the product ID the die ID (`0x501` for STM32MP13xx MPU) and the revision ID (`0x1003` for silicon revision 1.2)

- Alternatively, use the STM32CubeProgrammer graphical user interface (GUI) as follows:
  – On the right, select **USB** (not STLINK, set by default) in the connection picklist and click on the **refresh** button. The serial number is displayed if a USB connection is detected.
  – Click on the **Connect** green button.
  – The information is shown in Figure 1, with the product ID that includes the die ID (0x501 for STM32MP13xx MPU) and the revision ID (0x1000 is silicon revision 1.0, a silicon revision 1.2 returns `0x1003`).

Figure 1. STM32MP13xx acceptance using STM32CubeProgrammer



Partition	Id	Size	Ri...
FSBL	0x1	128 KB	RW
virtual	0xf1	512 B	R

Log

```
09:04:14 : STM32CubeProgrammer API v2.7.0
09:06:39 : USB speed   : High Speed (480MBit/s)
09:06:39 : Manuf. ID   : STMicroelectronics
09:06:39 : Product ID  : DFU in HS Mode @Device ID /0x501, @Revision ID /0x1000
09:06:39 : SN          : 0001801F3530510138343532
09:06:39 : FW version  : 0x0110
09:06:39 : Device ID   : 0x0501
```

USB configuration

Port	USB1
Serial nu...	0001801F35305101383...
PID	0xdf11
VID	0x0483
Read Unprotect (MCU)	

Target information

Board	--
Device	STM32MP13xx
Type	MPU
Device ID	0x501
Revision ID	--
Flash size	–
CPU	Cortex-A7

How to select software packages?

Even though software packages are not part of the TOE, they can be useful to the procedure defined in Secure installation and preparation of the operational environment (AGD_PRE.1.2C). Refer to the section for details.

3.2 Secure installation and preparation of the operational environment (AGD_PRE.1.2C)

3.2.1 Hardware and software setup procedures

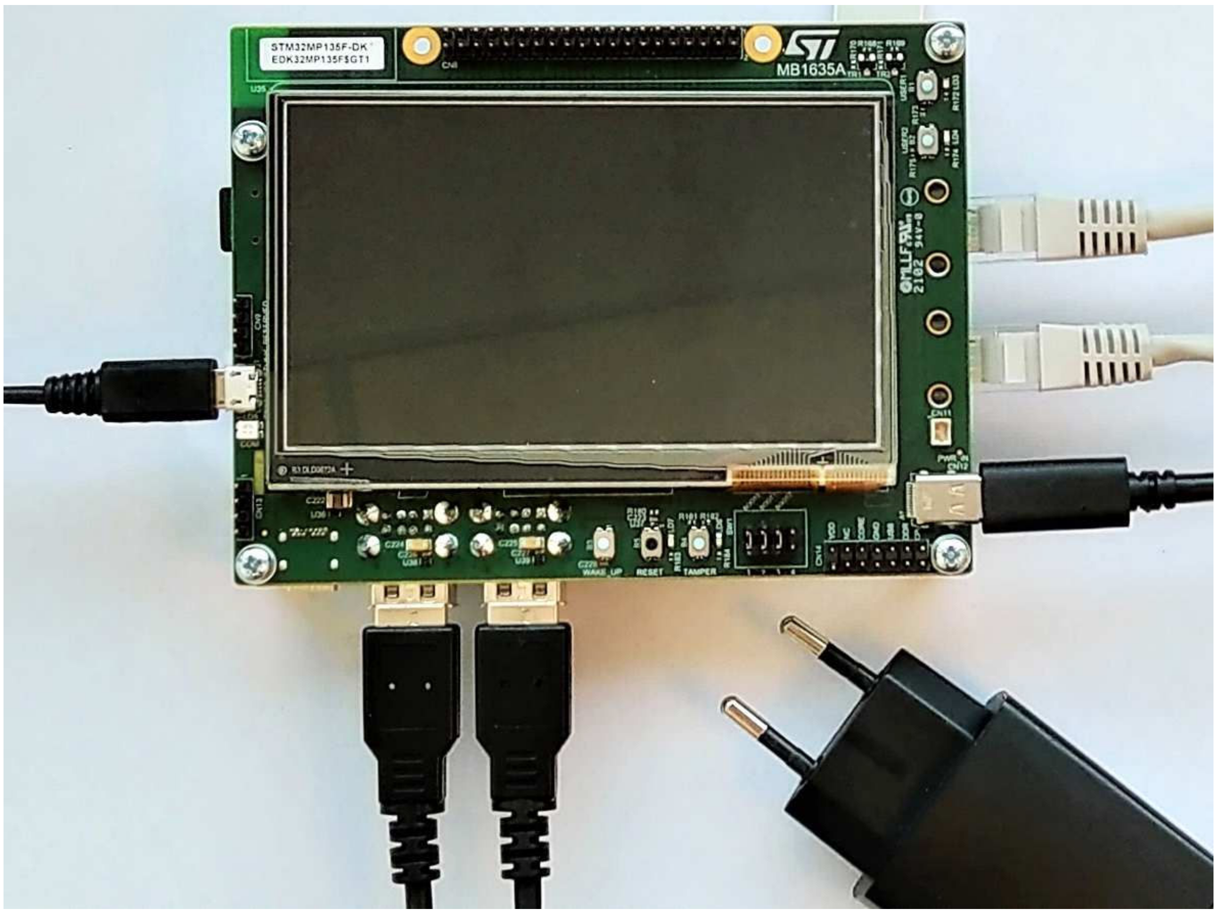
To exercise the TOE, it must be mounted on a board with USB, external flash memory, and DDR-SDRAM memory. Alternatively the application board STM32MP135F-DK Discovery kit can be used.

Hardware setup

STM32MP135F-DK Discovery kit board is used as described below.

1. Connect the USB Micro-B to Type-A cable between your laptop and the ST-LINK/V2-1 port of the board.
2. Connect the power supply using the USB Type-C® connector (power 5V-3A).
3. Optionally use the USB Type-A host to connect, for example, a mouse or a keyboard.
4. Optionally connect the Ethernet cable between your Ethernet network and the Ethernet port of the board.
5. Use the reset button when needed.

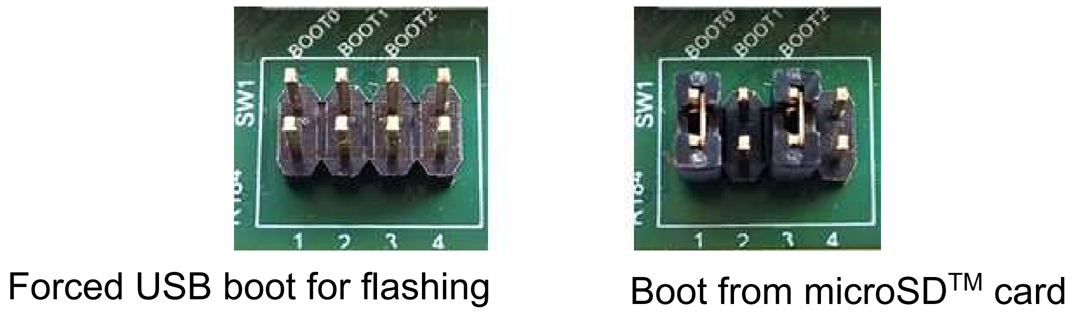
Figure 2. **STM32MP135F-DK Discovery kit connections**



Use one of the below SW1 switch configurations to select the boot mode.

- Forced USB boot for flashing: BOOT0= 0; BOOT1= 0; BOOT2= 0
- Boot on microSD™ card: BOOT0= 1; BOOT1= 0; BOOT2= 1

Figure 3. **STM32MP135F-DK boot relay switch configurations**



This connection with the PC allows the user to:

- Flash the board
- Interact with the board via a UART console
- Debug when the protections are disabled

Software setup

STMicroelectronics provides OpenSTLinux binary packages ("starter" packages) that can run directly on ST boards mounted with the TOE. Each starter package contains a set of complete, configured images to boot a non-secure platform. OpenSTLinux is based on the Trusted Firmware-A (TF-A) reference implementation, which can be found at https://trustedfirmware-a.readthedocs.io.

All the steps to install and use starter packages are described on the Getting started wiki page. Those packages, with the associated tools (STM32 key generator, fiptool, and STM32 signing tool) can be used on both Linux and Windows.

More specifically, the following tools are required for the preparation and installation of the TOE, as described in Secure installation and preparation of the operational environment (AGD_PRE.1.2C):

- STM32CubeProgrammer version 2.10.0
  - Includes STM32 signing tool and STM32 key generator
- Arm® TF-A fiptool version 2.4

For more details on the signing tool and key generator, refer to UM2542 and UM2543 respectively.

Note: *Key generation is not functional with STM32MP_KeyGen_CLI on Linux under Ubuntu 16.04 (it is functional with Ubuntu 18.04 or higher versions).*

3.2.2 Secure installation

STMicroelectronics delivers STM32MP13xx MPU devices without secrets and with provisioned Root of Trust key. Hence, those devices are booting on unsigned images, with usable debug and trace features (OTP-SECURE Open device).

Installation of the TOE corresponds to the provisioning of OTP information used during secure boot. After this step, device security is fully activated. The necessary steps are described in this section.

Step A: Generating secrets

In a trusted location, the user generates multiple ECC key pairs (up to 8) using the STM32 key generator tool (refer to UM2542 for details). Only one private key is used to sign the boot images ("active key").

With the same tool, the user generates the hash of the table composed of the hash of those eight public keys. This hash must then be stored in the OTP words 24 to 31 of the TOE. Refer to Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) for details on the TOE boot image authentication scheme.

Note: *Activating a root key in the TOE revokes all the previous ones. This means that if key#3 is activated and used for boot, then keys#1 and #2 are automatically permanently revoked.*

The user also needs to create a 128-bit secret to store the OTP words 92 to 95 of the TOE. He also creates a 32-bit derivation constant to be stored in the encrypted FSBL extension header. With this information, the STM32 key generator tool can compute a 128-bit encryption key that is used to encrypt the FSBL image using AES CBC chaining mode. The plain hash stored in the encrypted FSBL extension header is used as an IV input. Refer to Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) for details on image encryption.

Step B: Provisioning of secrets

If the provisioning environment is non-trusted, STM32 tools can be used, as detailed in UM2238. This non-trusted provisioning environment scenario is out of the scope of the evaluation.

In a trusted environment, the user can use the stm32key tool that is included in the OpenSTLinux image provided with the TOE. For example, the provision of the public key hash table in OTP words 24 to 31 is done as follows:

- Copy the PKH file (publicKeyhash.bin) in a file system partition like bootfs on a storage device like an SD card
- Load in BSEC hash file, for example, from mmc0 partition 4 (ext4) in DDR

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
32 bytes read in 50 ms (0 Bytes/s)
```

- Read-loaded key from DDR to confirm it is valid (without writing it in OTP)

```
Board $> stm32key read 0xc0000000
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

- If the hash key is right, the key in OTP can be written in OTP. Warning! fusing is a permanent operation and thus cannot be undone.

```
Board $> stm32key fuse -y 0xc0000000
```

The device now contains the hash of the table of hashes, to authenticate images.

A similar method can be used to write in OTP the secret key in OTP words 92 to 95.

Step C: Provisioning and locking the RMA password

1) Write in the OTP the password value, for example, `0x78563412`:

```
Board $> fuse prog -y 0 56 0x78563412
```

Note: This prog command cannot be executed twice because the values of this upper OTP word are ECC-protected. To avoid any issues, this OTP must be locked when programmed.

2) Lock OTP:

```
Board $> fuse prog 0 0x10000038 1

Board $> fuse read 0 0x10000038 1
Reading bank 0:
Word 0x10000038: 00000001
```

Step D: FSBL image signing

The FSBL image, loaded by the ROM, must be signed to be used on the certified TOE. STM32 signing tool can be used for that purpose, with the following command:

```
$ STM32MP_SigningTool_CLI.exe -bin tf-a-dk-sdcard.stm32 -pubk publicKey00.pem
publicKey01.pem publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem
publicKey06.pem publicKey07.pem - prvk privateKey00.pem -pwd passwd -of 0x00000001
-o tf-a-dk-sdcardsigned.stm32
```

Note: This command is later updated to manage the key revocation mechanism.

Step E: Image programming

Once the image is signed, it can be programmed into the flash memory on the target board with the STM32CubeProgrammer tool. Supported flash memory and its associated flash memory mapping are described in Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C).

Step F: Closing the device

Without any other modification, the device can perform image authentication but unauthenticated images can still be used and executed: the device is still opened, as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed, and the user is forced to use signed images. The recommended process follows.

1. Burn bits in OTP word 0 `0x17` to change the default OTP-SECURE Open device:
   – Write `0x28` in word 0 to change to OTP-SECURE Closed state, locking the authentication process by the ROM code, with default debug disabled.
   – Write `0x168` in word 0 to change to the OTP-SECURE Closed with boundary-scan disabled state, as the OTP-SECURE Closed state, with boundary-scan disabled.
   – Write `0x3E8` in word 0 to change to the OTP-SECURE Closed with JTAG disabled state, the OTP-SECURE Closed state, with boundary-scan disabled, and JTAG or SWD usage limited to the RMA sequence.

2. Nonsigned binaries are no more supported on the target device. Read back OTP word 0 to verify the correct programming:
   - For an OTP-SECURE Closed device word 0[9:0] `0x3F`
   - For an OTP-SECURE Closed with boundary scan disabled device word 0[9:0] `0x17F`
   - For an OTP-SECURE Closed with JTAG disabled device word 0[9:0] `0x3FF`

For example, to program the device in OTP-SECURE Closed state, the U-Boot command-line interface, available in the OpenSTLinux image, can be used as below.

```
Board $> fuse prog 0 0x0 0x28
Board $> fuse read 0 0 1
Reading bank 0:
Word 0x00000000: 0x0000 003F
```

Certified configuration

After installation the certified device configuration is the following:

- Boot image verification activated
- Boot image encryption optional
- RMA password provisioned
- Internal and external tamper sources are deactivated in the TAMP peripheral by default

The fuse configuration for the certified configuration is the following:

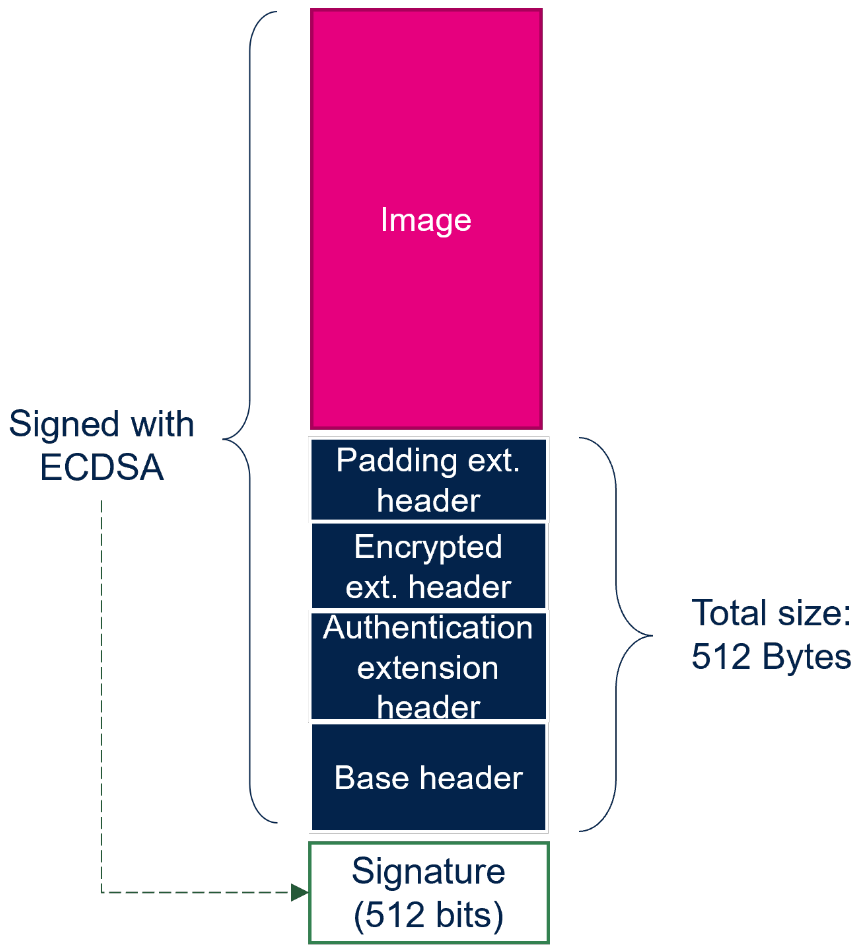
- BSEC status is OTP-SECURE Closed.
- Root key hash is provisioned in 24 to 31 fused words. It is used by ROM code to implement authentication and integrity protection. Two algorithms are supported for ECDSA signature verification: P-256 NIST or Brainpool 256.
- OEM secret key is provisioned in 92 to 95 fused words. It is used by ROM code to implement optional AES-128 GCM FSBL encryption protection.
- RMA password, stored in OTP word 56.
- Bit 7 "fsbl_decrypt_prio" is set in OTP word 9, to protect the FSBL decryption key against side-channel attacks.
- Bit 0 "Boot_traces_disabled" is set in OTP word 9.

Note: *Refer to the OTP-mapping section in RM0475 for details (including endianness).*

Boot device selection with fuses is not part of the certified configuration, as it does not impact the security.

Each binary image (signed or not) loaded by ROM code need to include a specific STM32 header added on top of the binary data. This header includes two extension headers: one for FSBL authentication, and one for FSBL decryption. It is shown in Figure 4.

Figure 4. **Authenticated STM32 header (with extensions) with binary files**



4 Operational user guidance

4.1 User role

The user role integrator, also called original equipment manufacturer (OEM), is the most relevant for this TOE. Indeed, the integrator is the one to:

- Receive the TOE,
- Perform the preparative procedures as described in TOE preparative procedures,
- And integrate the TOE into a full IoT solution.

The user-operational guidance is described in the Operational guidance for the integrator role.

The integrator has full access to the TOE security features, as the STM32 MPU devices are delivered in OTP-SECURE Open state with the secure boot feature de-activated. The integrator also has full access to the tools needed to program the TOE.

4.2 Operational guidance for the integrator role

4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C)

The main task of the integrator is to integrate the TOE into a full IoT solution. To this end, the integrator has access to interfaces that are unavailable to other users, as described in Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C). The integrator can also change some parts outside or inside the TOE, without impacting the certified configuration of the TOE.

The evaluated TOE scope covers the STM32 device hardware and ROM code, with a focus on secure boot, secure update, and hardware attacks. Hence, most of the user-accessible functions in the certified TOE are either hardware or immutable (ROM code, OTP bits).

Follow the procedures described in Secure acceptance to check if the TOE in the certified configuration is used. This section describes the changes that the integrator can do inside the TOE. It also clarifies what is covered in the scope of the evaluation, and what may impact the certified configuration of the TOE.

The integrator must follow the guidelines described in that section, as a failure to do so means that the TOE is not used in the certified configuration.

Boot image authentication scheme

The secure boot on the TOE is certified in the ECDSA P-256 asymmetric crypto scheme configuration, as described in Secure installation and preparation of the operational environment (AGD_PRE.1.2C). Such configuration, stored authenticated in the STM32 header, is summarized in Table 3.

If the ROM code finds that the active public key has changed, meaning that the public-key ID in the STM32 header impacts the OTP word 22, then the ECDSA key counter is updated according to the rule pictured in Figure 5.

During certified TOE boot image authentication, if boot image authenticity and integrity are not confirmed, the device either goes to the reboot state (watchdog reset) or switches to a different boot interface state, as defined in Flash memory interfaces in Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C).

Table 3. STM32 header information for authentication

	Name	Length (in bits)	Byte offset	Description
Version number		32	0x60	Image version (monotonic number)
Extension flags		32	0x64	Bit[0]=1: Authentication is enabled.[1] Bit[31]=1: Padding extension is enabled.[2]
Authentication extension header	Extension header type	32	0x80	Fixed value 0x53740002
	Extension header length		0x84	Number of bytes in extension header: 84+N*32. Value is 340 Bytes, as N=8 (fixed).
	Public-key id		0x88	Index in the hash table (at offset 0xD4) of the "active" public-key ROM code must use

	Name	Length (in bits)	Byte offset	Description
Authentication extension header	Number of public keys in the table N	32	0x8C	Number of public keys in the table (N=8)
	ECDSA algorithm	32	0x90	1: P-256 NIST 2: Brainpool 256
	ECDSA public key	512	0x94	Public key hash table, to check the hash table starting at offset `0xD4`
	Algo+PublicKey1 hash	256	0xD4	SHA-256 of {ECDSA algorithm + public key 1}
	...			
	Algo+PublicKey8 hash	256	0x1B4	SHA-256 of {ECDSA algorithm + public key 8}

1. *Clear the bit to disable it. Authentication can only be disabled on OTP-SECURE Open devices.*
2. *Used to have a fixed header size of 512 bytes.*

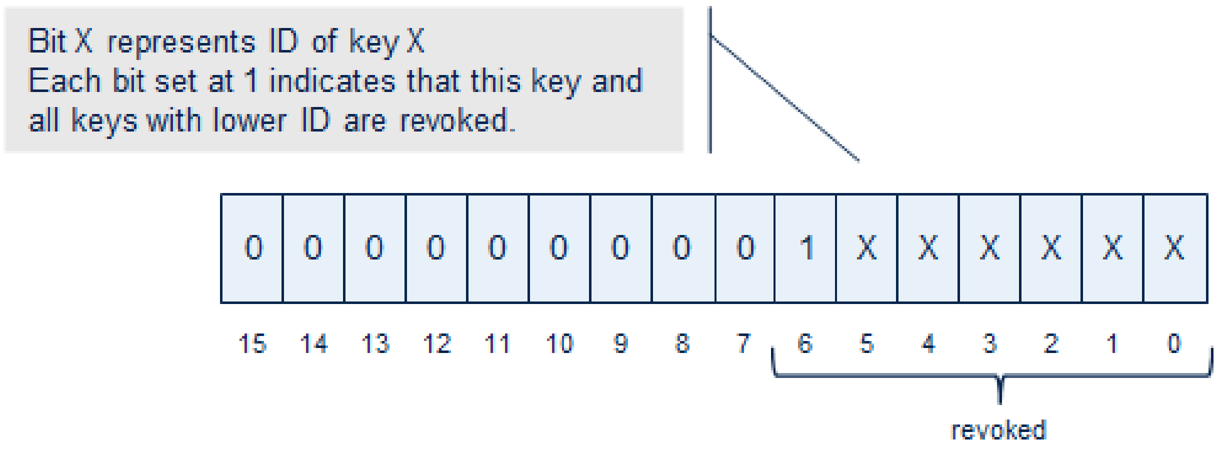

Bit X represents ID of key X
Each bit set at 1 indicates that this key and all keys with lower ID are revoked.
0 0 0 0 0 0 0 0 0 1 X X X X X X
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
revoked

Figure 5. **Active-key monotonic counter in OTP word 22**

Version number and boot image revocation

When the ROM code finds that the version number in the STM32 header is greater than the monotonic counter stored in OTP word 4, it considers it as a new image.

When it occurs, and the image authentication and decryption are successful, the ROM code increments automatically OTP word 4.

Note: The ROM code never executes an image with a version number lower than its monotonic counter in OTP (image revocation).

Table 4. **STM32 header information for the version number**

Name	Length (in bits)	Byte offset	Description
Version number	32	`0x60`	Image version (monotonic counter)

Image encryption

The TOE is certified with image encryption capability enabled or not. Such configuration, stored authenticated in the STM32 header, is summarized in Table 5. Enabling FSBL encryption ensures the confidentiality of the boot image.

Table 5. **STM32 header information for FSBL encryption**

	Name	Length (in bits)	Byte offset	Description
Extension flags		32	`0x64`	Bit[0]=1: Authentication is enabled.(1) Bit[1]=0 or 1: FSBL encryption enabled if set. Bit[31]=1: Padding extension is enabled.(2)
Encrypted FSBL extension header	Extension header type	32	`0x1D4`	Fixed value 0x53740001
	Extension header length		`0x1D8`	Fixed value 0x20
	Key size		`0x1DC`	Size of the encryption key in bits (128-bit)
	Derivation constant		`0x1E0`	Constant used to derive the key from the secret stored in OTP (92 to 95 fused words)
	Plain hash	128	`0x1E4`	128 MSB bits of plain payload hash

1. *Clear the bit to disable it. Authentication can only be disabled on SECURE-OPEN devices.*
2. *Used to have a fixed header size of 512 bytes.*

Jump to authenticated image

When image authentication is successful, the ROM code stores the address of the boot context in the Cortex-A7 r0 register, and jumps to the FSBL entry point defined in the STM32 image header. Image length is also recorded in the header, as shown below.

Table 6. **STM32 header information for FSBL entry point**

Name	Length (in bits)	Byte offset	Description
Image length	32	0x4C	Length of the image in bytes(1)
Image entry point	32	0x50	FSBL entry point in SYSRAM

1. *Length of the built image. It does not include the length of the STM32 header.*

Additionally, the ROM code stores in the first 512 Bytes of SYSRAM a boot context that contains information on the boot process, such as the selected boot device, and pointers to the ROM code services, used for secure boot authentication.

Refer to the **boot_api_context_t** structure in https://github.com/STMicroelectronics/arm-trusted-firmware/blob/v2.4-stm32mp/plat/st/stm32mp1/include/boot_api.h for details.

Number of images in external flash memory

The integrator can configure the TOE to use one or two copies of the FSBL in the external flash memory. In the case of using two copies, FSBL1 and FSBL2, the ROM code tries to load and launch the first copy and in case of failure, it then tries to load the second copy. Both configurations are within the scope of the certified configuration.

Note: *NAND flash memory includes more than two copies of the FSBL.*

For more details on supported flash memories and associated image mapping, refer to Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C). Note that any of those interfaces can be disabled with a fuse. Refer to Flash memory interfaces for details.

STM32CubeProgrammer can be used to prepare correct flash memory images.

Serial boot

The TOE is certified with boot support on UART and USB serial interfaces. In this configuration, if there is no valid image installed in the FSBL1 or FSBL2 slot, or in case the user presses the reset button with a specific bootpin or fuse configuration (refer to Serial boot interfaces), then the ROM code in the TOE scans in parallel all bootable UART instances and the USB OTG. When activity is detected on an interface, the boot process goes on with this interface and the others are ignored. Any of those interfaces can be disabled with a fuse. Refer to Serial boot interfaces.

The integrator can change the TOE serial boot management or remove the serial boot functionality. Both options are in the scope of the certified configuration. Refer to Secure installation and preparation of the operational environment (AGD_PRE.1.2C) for details.

Fault injection attacks countermeasures

The platform claims resistance against physical attackers. To fully leverage it, the Integrator must enable the below protections.

The integrator must implement the following software countermeasures within its application when performing sensitive operations (including cryptographic ones):

- Redundancy.
  For example:
  1. Perform the sensitive operation twice and verify that the results are equal.
  2. Verify ciphering operation with enciphering or enciphering operation with ciphering.
  3. In case of verification error, implement a security response, for example, platform reset.
- Random timing jitter:
  For example, apply a random loop (using the RNG peripheral) before sensitive operation.
- Execution control flow:
  For example:
  1. Use a finite state machine in which transitions are verified to be legit.
  2. Use a scattered known computation with a verified result at the end.
  3. In case of control flow error, implement a security response. For example, a platform reset.

In addition to the above protection, the integrator should implement anti-tampering protection:

- Enable system and cryptographic internal tamper detection (TAMP peripheral):
  When a tamper flag raises (tampering detected), the integrator must implement a security response, for example a platform reset.

TOE hardware-accelerated cryptography

The TOE is certified with hardware-accelerated cryptography that is protected against SCA, fault injection, and timing analysis attacks.

To achieve the required resistance against hardware attacks, the integrator must use the following hardware-accelerated cryptography function, detailed in the corresponding sections of the RM0475 reference manual.

- AES-128 and AES-256 encryption/decryption, authenticated encryption or decryption, cipher-based message authentication code computation, done in the SAES peripheral.
- Modular exponentiation for RSA decryption (MODE=`0x3`), scalar multiplication (MODE=`0x20`), and signature (MODE=`0x24`) for ECC, done with the PKA peripheral.

Note: *SAES and PKA cannot operate if the RNG peripheral is not properly configured, with its AHB clock running.*

As with any software dealing with sensitive data, the software driving hardware cryptography accelerators must follow the guidelines described in Fault injection attacks countermeasures, such as timing randomization and control flow.

With the TOE, the integrator can do cryptographic key generation for ECDH, ECIES, and ECDSA algorithms. In FIPS PUB 186-4 specification section B.4 NIST proposes two methods for the generation of the ECC private key d ("extra random bits" or "testing candidates"). The integrator must select one of those two methods when computing ECC private keys.

Countermeasures for SAES implementation targeting SESIPL3 (or equivalent), the integrator:

- Must implement systematically the inverse of the cryptographic operation (encrypt then decrypt or decrypt then encrypt) and compare the result with the initial cryptographic input. The integrator must implement a random timing jitter between the cryptographic operation and its inverse.
- Must implement redundancy when verifying results. The integrator must implement a random timing jitter between each result comparison.
- Must implement a control flow that verifies each step mentioned above is completed.
- Should activate the crypto (iTamper9) internal tamper and a security response.
- Must take appropriate action when an error linked to countermeasures is detected according to its security policy (as an example, the application might reset the system).

Cryptographic hash operations, available in the HASH peripheral, must not be used to manipulate sensitive information.

When executing sensitive code from the DDR-SDRAM, the application can also activate the on-the-fly AES-128 cipher engine, which uses a Keccak-based key derivation technique to derive a unique key per 128 bits of data. This key derivation primitive is resistant to side-channel attacks.

Note: On-the-fly DDR encryption/ decryption usage is described in the memory cipher engine (MCE) section of the RM0475 reference manual.

Random number generation

For the device to generate random numbers as specified in NIST SP800-90B, the integrator must use the RNG peripheral with the configuration A. Refer to the True random number generator (RNG) section of the RM0475 reference manual for details.

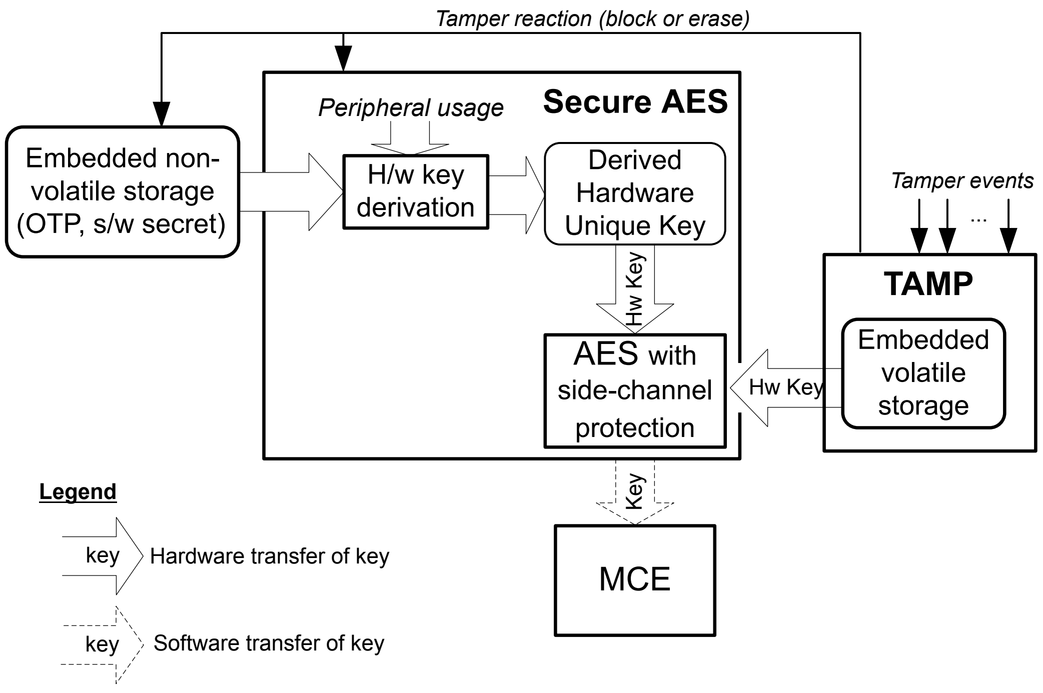
Cryptographic key storage

Thanks to the TOE hardware secret-key functions, the application can store secret keys that it can use without access to its value in the clear, and only when running on this instance of the TOE. Such key storage is also protected against hardware attacks like SCA, fault injection, or timing attacks, thanks to the usage of the SAES peripheral.

Figure 6 summarizes how secure and non-secure key storage applications can use the side-channel protected SAES engine available in the TOE. Nonvolatile keys are stored in external flash memory or OTP, while embedded volatile storage consists of the battery-powered, tamper-protected SRAM3 or the registers in the TrustZone®-aware, tamper-protected TAMP peripheral.

Note: Software can manipulate keys (like MCE keys), or keys can only transit via hardware key buses, as shown.
Keys stored in SAES (in registers or as derived hardware-unique key) are not usable in case of a tamper event.

Figure 6. Key management principle


Tamper reaction (block or erase)
Embedded non-volatile storage (OTP, s/w secret)
Peripheral usage
Secure AES
H/w key derivation
Derived Hardware Unique Key
Hw Key
AES with side-channel protection
Tamper events
...
TAMP
Embedded volatile storage
Hw Key
Key
MCE
Legend
key
Hardware transfer of key
key
Software transfer of key

As shown in Figure 6 there are three different sources of keys that are never visible in the clear by any application (called hardware secret keys):

- **Derived HUK (DHUK)** – 128 or 256-bit derived keys based on nonvolatile, device-unique, 256-bit software secret stored in fuses. DHUK generation triggers each time it is needed, and handles the use or not of TrustZone® and the context of the key usage in SAES.
- **Boot Hardware Key (BHK)** – 256-bit application key stored in tamper-resistant volatile storage in TAMP peripheral. This key is written at boot time, then read- or write-locked until the next backup domain reset. Refer to the TAMP section of the RM0475 for details.
- Result of an **XOR between BHK and DHUK (XORK)**, computed each time it is needed.

Note: DHUK, BHK, and XORK are only usable in a physical attack-protected SAES peripheral.

Hardware secret keys can be used in three different modes:

- As a normal key, loaded in write-only key registers of the SCA-resistant SAES, in the software key mode
- As an encryption/decryption key for a 128 or 256-bit key, used only in the SCA-resistant SAES, in the wrapped key mode
- As an encryption/ decryption key for a 128 or 256-bit key, used either in SCA-resistant SAES or in the faster, nonprotected CRYP engine, in the shared key mode

Note: Proper usage of the last two modes is described respectively in SAES operation with wrapped keys, in SAES operation with shared keys, and in the SAES section of the RM0475.

Thanks to this hardware secret key feature, a security application in the TOE can protect the integrity and confidentiality of AES 128 or 256-bit keys in its KeyStore, using encryption or authenticated encryption with the DHUK. The resulting decrypted keys are automatically stored in SCA-protected, write-only SAES key registers, without disclosing any clear-text key data to the application. Additionally, if the application tries to overwrite part of the key (integrity attack) the whole key is automatically erased.

When the application-defined BHK is xored with DHUK when encrypting KeyStore items, each of those items cannot be decrypted until the secure boot code installs the correct BHK.

Note: Any keys encrypted by DHUK or BHK are not usable when a tamper event occurs.

Crypto peripherals critical to secure ROM code are made secure only. Refer to TrustZone® and MMU isolation usage in this section for details.

Nonvolatile storage of secrets in OTP

In the certified TOE, the secure application can use the BSEC peripheral to store nonvolatile keys or data using two kinds of 32-bit words:

- ECC-protected OTP words (32 bits encoded into 38 bits), one-time programmable.
- One to two redundancy-protected OTP words, bit per bit programmable from 0 to 1.

Note: As ECC-protected words can only be programmed once, the application can use the hardware function of OTP word permanent program locking to enforce it.

BSEC is only writable by secure code.

To benefit from the automatic protection of OTP secrets, when the device is decommissioned for field return to STMicroelectronics (RMA sequence), secure applications must store their secret keys in 59 to 91 fused words. The application can trigger this protection until the next reset, setting bit 0 in the BSEC_OTP_LOCK register.

To enforce secret integrity, the application must write lock shadowed-OTP words 59 to 91, setting corresponding sticky bits in the BSEC_OTP_SPLOCKx registers.

Each time the application reads one OTP word, it is recommended to check the BSEC status registers for "disturbed" and "error" status (respectively BSEC_OTP_DISTURBEDx and BSEC_OTP_ERRORx). When disturbed status occurs, a new read operation is required. When error status occurs the fault might be permanent (for example, bad ECC).

For more information on BSEC, refer to the RM0475 reference manual.

TOE-specific information personalization

Once the integrator finishes its IoT product development and wants to start production, the integrator must securely provision the TOE assets, as summarized below:

- The integrator has the privilege and responsibility to configure the cryptographic keys used by the TOE to authenticate, optionally decrypt the FSBL, and also put the TOE in field return mode. During this process, the RMA password and optional FSBL encryption key must be kept confidential.

- The integrator must implement appropriate security measures in the operating environment, to protect the private key involved in the signature of the authenticated FSBL image. However, if such a private key is compromised, the integrator can select the next ECDSA public key in the STM32 header, as described in the Boot image authentication scheme of this section.
- The integrator must protect the integrity of the public key hash table of the TOE until it is programmed and protected in integrity inside each TOE.

Any failure in the above responsibilities can result in the creation of malicious firmware, which violates the assumptions made in the security target.

Note: TOE is secured as described in Secure installation and preparation of the operational environment (AGD_PRE.1.2C) the integrity and confidentiality of the above assets are ensured.

When the integrator cannot rely on a trusted environment, such as a trusted manufacturing facility, to provision the TOE assets and secure the device, the secure secret provisioning service (SSP) embedded inside the ROM code may be used. Refer to AN5510 for details.

TrustZone® and MMU isolation usage

In the certified TOE, the boot ROM uses multiple hardware mechanisms, to protect the assets used during secure boot, low-power mode wake-up, lifecycle management (debug control, RMA), or integrator secrets secure provisioning. Corresponding hardware functions are:

- Usage of TrustZone® to isolate from the less trusted code executing in non-secure mode critical peripherals and a portion of SYSRAM. Such configuration can be locked until the next system reset.
- Leverage of Arm® Cortex®-A7 memory management unit (MMU), to assign secure and non-secure tasks to its own virtual address space.

It is recommended that the secure boot executing in SYSRAM, and the following runtime security applications use the same techniques. The flexibility for an integrator to use or not TrustZone® or the Cortex®-A7 MMU for security falls within the scope of the TOE evaluation.

For example, it is recommended when cryptographic drivers execute in the secure mode of the Cortex-A7, that the integrator verifies that the peripherals and the memory used by those drivers are read/write secure only. More specifically, the following hardware features must be used:

- ETZPC_DECPROT2 register controls secure access to RNG, HASH, CRYP, SAES, and PKA. This configuration can be locked until the next reset using the corresponding LOCK bits in the ETZPC_DECPROT_LOCK1 register.
- ETZPC_DECPROT2 register also defines secure or non-secure access to BKPSRAM memory (read and write modes). This configuration can be locked until the next reset using the LOCK13 bit in the ETZPC_DECPROT_LOCK1 register.
- ETZPC_TZMA1_SIZE register defines the size of the secure area in SYSRAM, with the possibility to lock the configuration until the next reset.
- TZC registers can define at least one secure region in DDR-SDRAM. This region can be encrypted using the memory cipher engine (DDRMCE).

Note: ETZPC and TZC registers are write-secure.

BSEC peripheral does not need to be allocated to the secure world, as upper OTP secrets are secure only by hardware (TrustZone-aware peripheral).

More details of the above are found in the RM0475 reference manual.

Resource isolation

For an optimal security device, the boot stages of the application must be isolated from each other, doing the following:

- The code in each stage must execute its tasks, complete them, and then hand over the CPU execution to the next stage code. In particular, the secure boot application must update its exception vectors and IRQ handlers at the beginning of its execution.
- Private data for one stage, in particular any secrets, must not be accessible from later stages.
- The code in each stage must exit its stage carrying forward well-defined results and nonprivate data to the next stage.

To help enforce such isolation between boot stages, the TOE ROM code and the secure boot application can use the following temporal isolation hardware capabilities:

- In BSEC peripheral:
  - After its execution, the ROM partially hides itself using the ROMLOCK bit in the BSEC_OTP_LOCK register.
  - Any 32-bit OTP word can be made unreadable and/or unwritable until the next system reset. It can also be made permanently unprogrammable. The ROM code uses this protection to hide secrets used for boot image decryption and to provision integrator secrets. Refer to the swprlock1 and swprlock2 columns in the OTP mapping (OTP) section of the RM0475 reference manual.
- In TAMP peripheral:
  - Until the next system reset, 256-bit backup registers can be made usable only by the SAES engine after secure initialization of the platform. This feature is not used by the ROM code.

For more information on BSEC security features, refer to the boot and security and OTP control (BSEC) section of the RM0475.

Return material for analysis (field return)

The user must invoke this functionality before returning the TOE device for failure analysis to STMicroelectronics. As the user data, stored in 59 to 91 fused words, is permanently hidden once the RMA sequence is completed, the user can be assured that not even the vendor can access the personal information stored there.

See the **swprlock3** column in the OTP mapping (OTP) section of the RM0475 reference manual.

The TOE is certified with a 32-bit RMA password stored in fuse word 56. The integrator has the privilege and responsibility to provide its RMA password as described in the Secure installation and preparation of the operational environment (AGD_PRE.1.2C).

Anti-tamper

The TOE is certified with internal and external tamper sources deactivated by default in the TAMP peripheral. The integrator has the privilege and responsibility of configuring anti-tamper methods available in the device to add a layer of security.

Those methods are described in the tamper and backup registers (TAMP) section of the RM0475 reference manual and summarized in the following table.

Note: When activated only a reset of the backup domain can deactivate the tamper protections.

Table 7. **TOE internal tamper sources**

Tamper input number	Description	Tamper input number	Description
1	Backup domain voltage monitoring	7	12-bit ADC analog watchdog 1[(1)]
2	Temperature monitoring	8 and 10	Monotonic counter overflow (x2)
3	LSE monitoring	9	SAES, or PKA fault detection
4	HSE monitoring	11	Secure IWDG1 timeout and potential tamper[(2)]
5	RTC calendar overflow	12	12-bit ADC analog watchdog[(1)] 2 and 3
6	Any JTAG or SWD access[(3)]	13	

1. *ADC analog watchdog function, with low and high thresholds.*
2. *IWDG1 reset when at least one enabled tamper flag is set.*
3. *Triggered when the device-level JTAG port switches from reset to idle state. This transition is guaranteed when boundary-scan, RMA, or debug is triggered via JTAG or serial wire.*

When anti-tamper protections are activated, the boot ROM can always load and execute the secure boot image, even if the tamper event is confirmed, meaning NOERASE is set to 0. Note that in this case, ROM code is always doing the following:

1. For each x matching, the condition TAMPxF = 1 in TAMP_SR, and TAMPxNOER = 0 in TAMP_CR2, the ROM sets TAMPxNOER to 1.
2. For each x matching, the condition ITAMPxF = 1 in TAMP_SR, and ITAMPxNOER = 0 in TAMP_CR3, the ROM sets ITAMPxNOER to 1.

3. After reboot, the secure boot in SYSRAM must verify which TAMPxNOER (resp. ITAMPxNOER) bits in the TAMP_CR2 (resp. TAMP_CR3) register are transformed by the ROM code, using the application reference stored in SYSRAM. It can then react accordingly.

4.2.2 Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C)

To exercise the functions and privileges described in User-accessible functions and privileges (AGD_OPE.1.1C), the integrator interacts with the TOE interfaces described in this section:

- Cryptographic functions interface (registers) needed to access cryptographic operations and cryptographic key store resources.
- Nonvolatile storage as OTP fuse words, accessible via BSEC peripheral
- Physical chip interface involved in detecting transient perturbation attacks on some chip interfaces (HSE, LSE, selected power supplies)
- Flash memory interfaces, to access boot images: SD card, eMMC, FMC, Quad-SPI
- Serial interface from which boot image can be downloaded out of reset: UART, USB
- JTAG or SWD debug interface, closed out of reset
- BOOT pins PI4, PI5, PI6, corresponding to the user switches SW1 on the STM32MP135F-DK discovery kit

In the context of the certification, the TOE implements several mechanisms to secure the initialization of the platform, and increase robustness against physical attacks.

Cryptographic functions interface

In each cryptographic peripheral protected against physical attacks, the integrator can use a set of registers to access cryptographic operations and optionally cryptographic key store resources. Register access can be restricted to secure code only when TrustZone® protection is activated for the peripheral.

Method of use:

- Power on the product as defined in RM0475
- Reset the device as defined in RM0475
- Device executes the ROM code.
- ROM executes the authenticated code of the integrator when the TOE is in its certified configuration. This code uses SAES and PKA peripherals freely after the RNG peripheral is properly configured and clocked (in RCC).

Parameters:

- Integrator can access the registers and memory of cryptographic peripherals protected against physical attacks (SAES, PKA), as defined in RM0475.
- Non-secure access to SAES, PKA, and RNG can be prevented by configuring the ETZPC_DECPROT2 register.

Actions:

- Integrator defines if only the secure application can access SAES, PKA, and RNG registers and memory, using the ETZPC_DECPROT2 register.
- If the security level is enough, the integrator code can freely use SAES, and PKA registers and memory, as defined in RM0475.

Errors:

- The SAES and PKA peripherals have the precise error events described in RM0475.
- When SAES or PKA detects a fault injection, an input tamper event is raised in the TAMP peripheral. Depending on the integrator code, such an event can block the TOE application until the product is reset. In the certified configuration, when the ROM code detects a hardware fault in SAES or PKA, the TOE is blocked until reset by hardware, or the IWDG goes to time out.

BSEC interface

After each device power-on or system reset the TOE executes the ROM code that manages the secure initialization of the platform, leveraging nonvolatile information accessible via the BSEC peripheral. BSEC includes some hardware protections against physical attacks.

Like the ROM, the integrator code can use the BSEC peripheral, after some fuse information is locked in reading, writing, or programming mode during boot. The integrator can do the same, at any time.

Fuses are used in IoT products to store among other things chip state, security level, and secure boot configuration. The application can also get information about platform identity there.

Method of use:

- Power on the product as defined in RM0475.
- Reset the device as defined in RM0475. BSEC registers BSEC_OTP_DATA0 to BSEC_OTP_DATA95 are updated with OTP data. BSEC_OTP_STATUS, BSEC_OTP_DISTURBEDx, and BSEC_OTP_ERRORx registers are also updated.
- BSEC defines the device security state (OTP-SECURE Closed, OTP_INVALID...), then asserts a signal that releases the CPU reset.
- Device executes the ROM code. Then ROM executes the authenticated code of the integrator when the TOE is in its certified configuration. Some OTP information is not accessible to the integrator, after ROM code execution.
- BSEC updates the disturbed and error status registers after each OTP read operation.
- To program a fuse, follow the guidance in RM0475.
- BSEC OTP word usage, and related protection activated by TOE after reset, are described in section 4 OTP mapping (OTP) in RM0475.

Parameters:

- Sticky read-lock: Integrator can use the BSEC_SRLOCKx register to prevent reloading of selected shadow registers until the next system reset.
- Sticky write-lock: Integrator can use the BSEC_SWLOCKx register to lock the write to the selected shadow register until the next system reset.
- Sticky programming lock: Integrator can use the BSEC_SPLOCKx register to prevent programming of selected OTP words until the next system reset.
- OTP permanent write-lock: To prevent further programming of any OTP word, the integrator can perform the same sequence as a normal programming operation, setting the LOCK bit in the BSEC_OTP_CONTROL register.
- BSEC locks: integrator can use the BSEC_OTP_LOCK register to lock read upper OTP fuses or to disable further BSEC programming. Those locks are active until the next system reset.

Actions:

- The integrator's secure or non-secure code reads shadow registers BSEC_OTP_DATA0 to BSEC_OTP_DATA31 (lower OTP bits) to access general purpose and nonsecret data. Only the integrator secure code can write to those OTP words, if possible (words not locked or not already written if it is ECC protected).
- The integrator secure code reads shadow registers BSEC_OTP_DATA32 to BSEC_OTP_DATA95 (upper OTP bits) to access usually security-sensitive information (secrets keys, certificates, and RMA key). Only integrator secure code can write to those OTP words, if possible (words not locked or not already written if it is ECC protected).
- Each time the application reads one OTP word, it is recommended to check the BSEC status registers for "disturbed" and "error" status (respectively BSEC_OTP_DISTURBEDx and BSEC_OTP_ERRORx). When disturbed status occurs, a new read operation is required. When error status occurs the fault might be permanent (for example, bad ECC).
- The integrator must trigger a system reset when the OTP-INVALID state is found in BSEC. If this state persists the device cannot be trusted anymore, and a decommissioning of the part is recommended.

Errors:

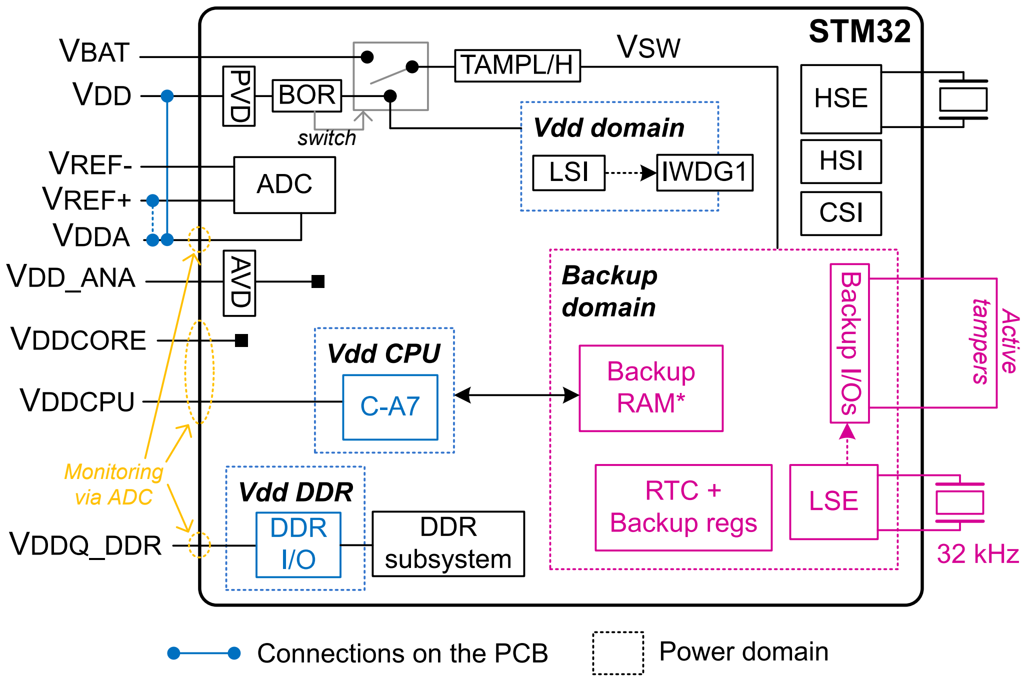
- When security errors are found out of reset BSEC goes into OTP-INVALID mode, with the following effects:
  - Registers BSEC_OTP_DATA32 to BSEC_OTP_DATA95 are set to 0, and cannot be updated until the next system reset.
  - More generally, secure applications can no longer load or program OTP words, as BSEC control registers become read-only. Non-secure applications cannot access BSEC anymore.
- When a disturbed error occurs, BSEC_OTP_DISTURBEDx is updated accordingly. Any bit set means that the last reading of the corresponding word is disturbed, due to abnormal reading conditions found in the decoding circuitry and related voltages.
- When a fuse error occurs, BSEC_OTP_ERRORx is updated accordingly. Any bit set means that the last reading of the corresponding word revealed a redundancy or ECC check error.
- In the certified configuration, when the ROM code detects a hardware fault in BSEC, the TOE is blocked until it is reset by hardware, or the IWDG goes to time out.

Physical chip interface

Following every secure initialization of the platform, the application can initialize active tamper logic with its related I/Os, and can use various pieces of hardware involved in detecting transient perturbation attacks on some chip interfaces. Protected interfaces are defined below and in Figure 7.

- LSE and HSE clocks
- Backup domain supply VSW
- VDD power supply
- VDDCORE and VDDCPU power supplies
- VDDA (or VREF+) power supplies, used as ADC reference
- VDDQ_DDR and VDD_ANA power supplies

Figure 7. Tamper-protected physical chip interfaces



Method of use:

- Power on the product as defined in RM0475
- Reset the device as defined in RM0475
- Device executes the ROM code.
- ROM executes the authenticated code of the integrator when the TOE is in its certified configuration. This code configures then enables tamper protection on the required device physical interfaces (power, clocks). Those protections are disabled by default.

Parameters:

- Tamper detection configuration and initialization are defined in RM0475. Internal tamper inputs in the TOE are summarized in the Anti-tamper section of User-accessible functions and privileges (AGD_OPE.1.1C).
- ITAMPxNOER bits in TAMP_CR3
  - If the bit is set: when a related internal tamper occurs the device secrets are not automatically erased. A software loop must execute first to confirm or not the tamper event. In this case, access to device secrets is blocked until TOE is reset, or after the application clears the event in TAMP, unblocking the accesses.
  - If the bit is cleared: when a related internal tamper occurs, the device secrets are automatically erased. The TOE must reboot to be functional again.
- IWDG peripheral timeout parameter. Refer to the following actions.

Actions:

- When the integrator activates a tamper input event linked to a physical chip interface, it must decide if it is the source of a potential tamper (ITAMPxNOER=1 in TAMP_CR3) or the source of a confirmed tamper (ITAMPxNOER=0 in TAMP_CR3).
- When a potential tamper occurs, and the integrator code has analyzed the tamper situation, there are two options:
  - The application launches erase of secrets using a software command (confirmed tamper).
  - The application does not confirm the tamper (false tamper), hence clearing the event in the TAMP peripheral, unblocking access to selected TOE secrets.
- If the integrator code fails to process the above potential tamper event in time (timeout), the event automatically becomes a confirmed tamper, erasing secrets. A timeout event is triggered when an IWDG reset occurs after a potential tamper is detected, meaning that the tamper flag with NOERASE = 1 is already set when the IWDG reset occurs.
- Integrator can trigger any tamper event by writing to the TAMP register.

Errors:

- As described before, any error occurring on the protected physical chip interface, like power or clock, can generate either an interrupt to execute the tamper code, or automatically trigger the deletion of selected TOE secrets.

BOOT pins

After each product reset, TOE is checking the state of BOOTPIN[2:0] pins in GPIOs PI4, PI5, and PI6. Depending on the pin state, the TOE downloads the boot image from flash memory or a serial interface. Refer to Flash memory interfaces or Serial boot interfaces in this section for details.

Flash memory interfaces

Method of use:

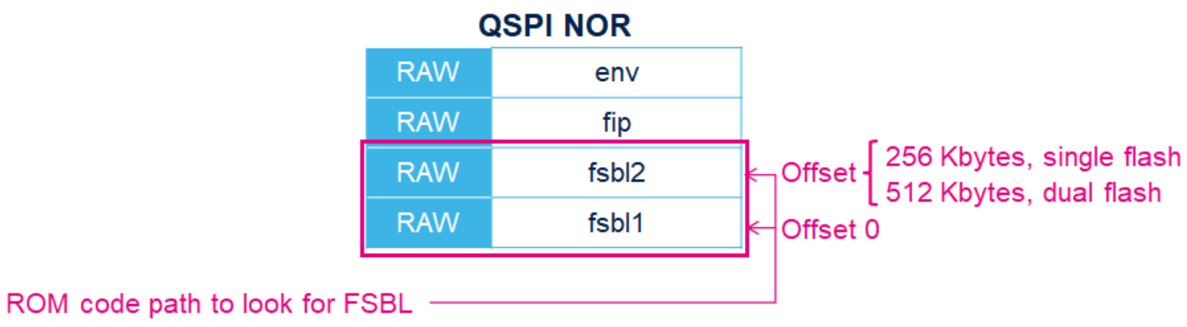
The corresponding boot images must first be stored in flash memory, for example, using the recommended STM32CubeProgrammer tool. Expected flash memory layouts are described hereafter.

Once the flash memory is properly initialized, the flash memory image interface usage goes as follows:

- When the STM32MP135F-DK Discovery kit board is used, do not set the boot switches to the OFF position (bootpin[2:0]=000).
- Power on the product as defined in RM0475
- Reset the device as defined in RM0475
- Device executes the ROM code that supports the following flash memory interfaces:
  - Serial NOR flash memory via Quad-SPI peripheral
  - Serial NAND flash memory via Quad-SPI peripheral
  - Parallel NAND flash memory via FMC peripheral
  - SD card via SDMMC peripheral
  - eMMC via SDMMC peripheral
- Once the TOE ROM code jumps on authenticated FSBL code, the integrator can design an application that accesses the device hardware, in secure or non-secure modes. He can also unlock the debug interface, as described in the JTAG interface.

Quad-SPI NOR layout contains two FSBL partitions. FSBL1 partition is located from offset LBA0 to offset LBA511. FSBL2 partition is located from offset LBA512 to offset LBA1023.

Figure 8. **Quad-SPI NOR flash memory layout without GPT**

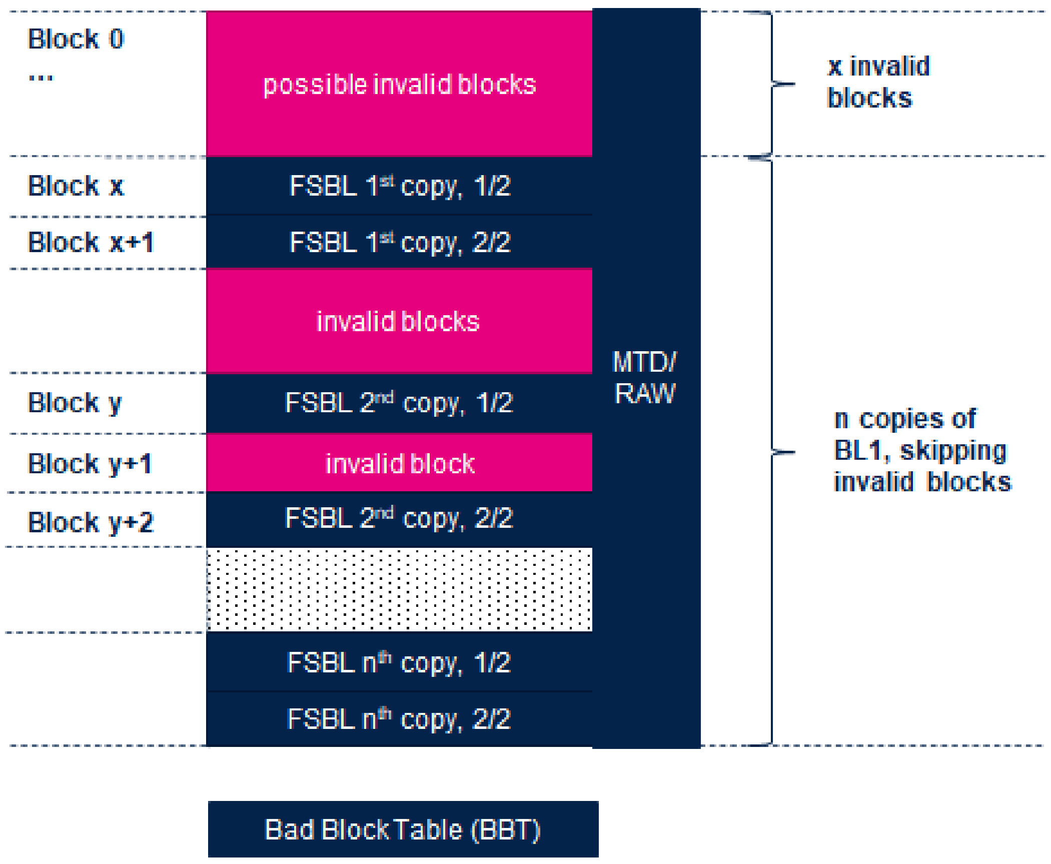


Note: It is possible to use NOR flash memory either in single or dual mode. In dual-mode, two NOR flash memories are connected to the two ports of the NOR interface and the two memories are used in interlaced mode.

Parallel NAND (via FMC) and serial NAND (via Quad-SPI) layouts contain n copies of FSBL in the first valid blocks. If FSBL uses N blocks in the memory (N is usually 2 for an FSBL maximum size of 247 Kbytes with a memory having a block size of 128 Kbytes), the ROM code scans blocks from the first and looks for FSBL in the first N consecutive valid blocks found.

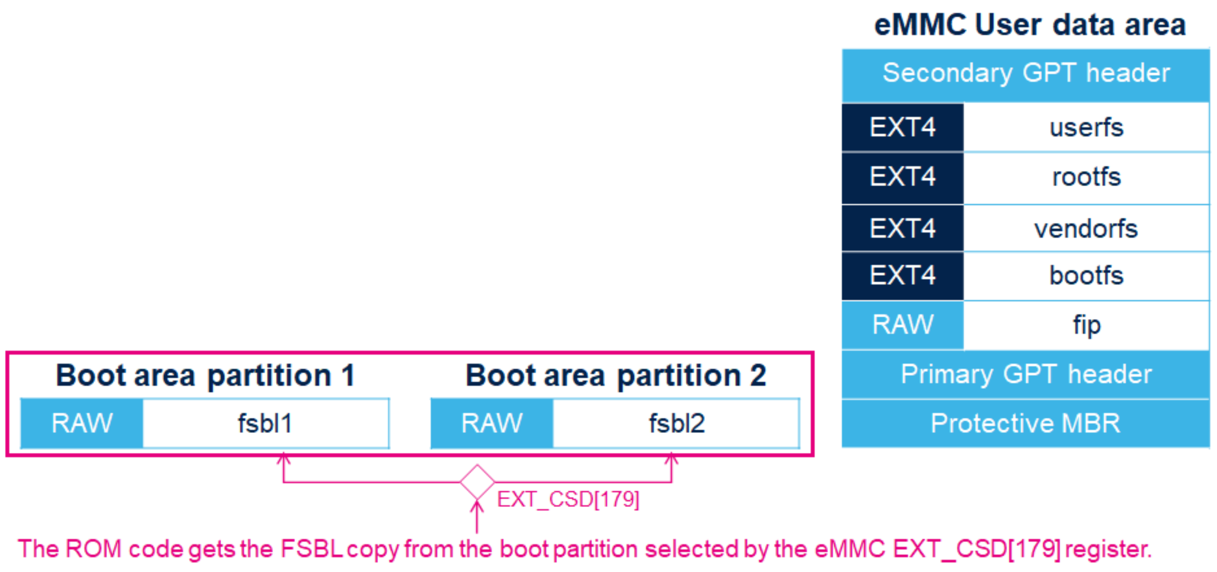
For more information, refer to Boot from parallel and serial NANDs.

Figure 9. NAND flash memory layout for ROM code



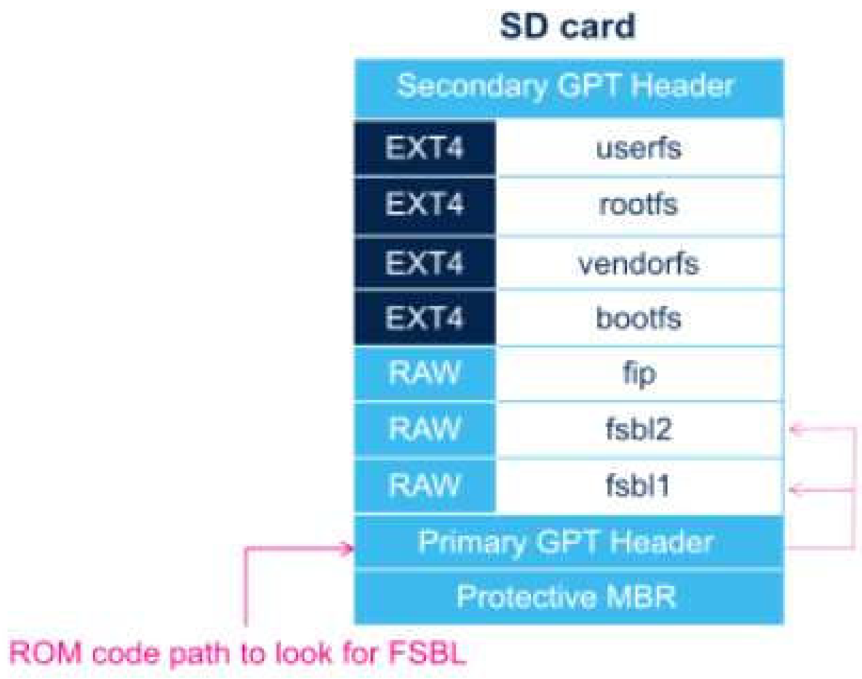
On the SD-MMC interface, the eMMC layout contains two copies of FSBL in its two boot regions, but only one boot region is active at a time. The ROM code tries to load and launch the copy of FSBL contained in the active boot region.

Figure 10. eMMC flash memory layout



On the SD/MMC interface, the SD card layout contains versions of FSBL. The ROM code first looks for a GPT. If it finds it, it locates two FSBLs by looking for the two first GPT entries of which names begin with 'fsbl'. If it cannot find a GPT, the ROM code looks for FSBL1 at offset LBA34 and FSBL2 at offset LBA546.

Figure 11. **SD card flash memory layout**



Parameters:

- Physical boot pin usage is summarized below. OTP word 3 is detailed in the next bullet.

Note: If boot pins select a source disabled by OTP, then the serial boot is selected.

Table 8. **Boot device selection via the boot pins and OTP (Flash memory)**

BOOT pins	TAMP_REG[20] (force serial)	OTP word 3 (primary boot source)	OTP word 3 (secondary boot source)	Boot source #1	Boot source #2 if #1 fails	Boot source if #2 fails
0b001	!= 0xFF	0 (virgin)	0 (virgin)	QSPI NOR	Serial	-
0b010	!= 0xFF	0 (virgin)	0 (virgin)	eMMC	Serial	-
0b011	!= 0xFF	0 (virgin)	0 (virgin)	FMC NAND	Serial	-
0b101	!= 0xFF	0 (virgin)	0 (virgin)	SD card	Serial	-
0b111	!= 0xFF	0 (virgin)	0 (virgin)	QSPI NAND	Serial	-
!= 0b100	!= 0xFF	Primary(1)	0 (virgin)	Primary(1)	Serial	-
!= 0b100	!= 0xFF	0 (virgin)	Secondary(1)	Secondary(1)	Serial	-
!= 0b100	!= 0xFF	Primary(1)	Secondary(1)	Primary(1)	Secondary(1)	Serial

1. *Primary and secondary are fields of OTP word 3.*

- OTP word 3 fuse configuration

Table 9. **Flash memory boot configuration in fuse word 3**

OTP field	Offset	Size (in bits)	Default	Description
primary_boot_source	[29:27]	3	0	If different from zero, it identifies the source used for the boot. One choice possible among 1: FMC NAND, 2: QSPI NOR, 3: eMMC, 4: SD, 5: QSPI NAND
secondary_boot_source	[26:24]	3	0	If different from zero, it identifies the source used for the boot. One choice possible among 1: FMC NAND, 2: QSPI NOR, 3: eMMC, 4: SD, 5: QSPI NAND
boot_source_disable	[23:16]	8	0	If different from zero, each bit disables a boot source. bit[0]: FMC NAND, bit[1]: QSPI NOR, bit[2]: eMMC, bit[3]: SD, bit[4]: UART, bit[5]: USB, bit[6]: QSPI NAND. Default to UART if all are disabled.
sd_if_id	[4:3]	2	0	If different from zero, it identifies the default instance to be used for the SD boot (SDMMC1, SDMMC2)
emmc_if_id	[2:1]	2	0	If different from zero, it identifies the default instance to be used for the eMMC boot (SDMMC1, SDMMC2)

- Quad-SPI NOR configuration: SPI legacy mode MOSI/MISO only, meaning uses only two pins IO0 and IO1 to transfer data, with QSPI_CLK set to HSI/2, meaning 32 MHz. The ROM code automatically detects single and dual modes. The default AFmux configurations, which can be overwritten by OTP values defined by OTP words 5 to 7, are described below.

Note: In dual-mode, BK1_NCS must be connected to BK2_NCS on board.

Table 10. **Serial NOR AFmux default configurations**

Single NOR mode		Dual NOR mode	
IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)
QUADSPI_CLK	PF10 (AF9)	QUADSPI_CLK	PF10 (AF9)
QUADSPI_BK1_NCS	PB2 (AF9)	QUADSPI_BK1_NCS	PB2 (AF9)
QUADSPI_BK1_IO0	PF8 (AF10)	QUADSPI_BK1_IO0	PF8 (AF10)
QUADSPI_BK1_IO1	PF9 (AF10)	QUADSPI_BK1_IO1	PF9 (AF10)
-	-	QUADSPI_BK2_IO0	PH2 (AF9)
-	-	QUADSPI_BK2_IO1	PH3 (AF9)

- NAND configurations.
  – For serial NANDs, the AFmux default setting is the same as for serial NOR. Refer to Table 10.
  – For parallel NANDs, the AFmux default configurations, which are overwritten by OTP values defined by OTP words 5 to 7, are described in Table 11.

Table 11. **Parallel NAND AFmux default configurations**

8-bit parallel mode				16 bits complement mode	
IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)
FMC_NOE	PD4 (AF12)	FMC_D2	PD0 (AF12)	FMC_D8	PE11 (AF12)
FMC_NWAIT	PA9 (AF10)	FMC_D3	PD1 (AF12)	FMC_D9	PE12 (AF12)
FMC_NWE	PD5 (AF12)	FMC_D4	PE7 (AF12)	FMC_D10	PE13 (AF12)
FMC_NCE	PG9 (AF12)	FMC_D5	PE8 (AF12)	FMC_D11	PE14 (AF12)
FMC_ALE	PD12 (AF12)	FMC_D6	PE9 (AF12)	FMC_D12	PE15 (AF12)
FMC_CLE	PD11 (AF12)	FMC_D7	PE10 (AF12)	FMC_D13	PB8 (AF12)
FMC_D0	PD14 (AF12)	-	-	FMC_D14	PD9 (AF12)

8-bit parallel mode				16 bits complement mode	
IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)
FMC_D1	PD15 (AF12)	-	-	FMC_D15	PD10 (AF12)

- SD/MMC for eMMC configuration: The ROM code uses only one data bit. By default, the ROM code uses the SDMMC2 instance. If the OTP bit emmc_if_id in OTP word 3 is not equal to 0, the ROM code uses the value of this field (1 or 2) to determine which SDMMC interface to use. SDMMC clock is set to HSI / 4 = 16 MHz.
- The AFmux default settings, which are overwritten by OTP values defined by OTP words 5 to 7 when emmc_if_id is set to 0, are described below.

Table 12. **SD/MMC eMMC AFmux default configuration**

IO name	Pin ID (AFmux)
SDMMC2_CK	PE3 (AF09)
SDMMC2_CMD	PG6 (AF10)
SDMMC2_D0	PB14 (AF09)

- SD/MMC for SD card configuration: The ROM code uses only one data bit. By default, the ROM code uses the SDMMC1 instance. If the OTP bit sd_if_id in OTP word 3 is not equal to 0, the ROM code uses the value of this field (1 or 2) to determine which SDMMC interface to use. SDMMC clock is set to HSI / 4 = 16 MHz.
- The AFmux default configurations, which can be overwritten by OTP values defined by OTP words 5 to 7 when sd_if_id is set to 0, are described below.

Table 13. **SD/MMC SD card AFmux default configurations**

IO name	Pin ID (AFmux)	IO name	Pin ID (AFmux)
SDMMC1_CK	PC12 (AF12)	SDMMC1_CKDIR	PB9 (AF11)
SDMMC1_CMD	PD2 (AF12)	SDMMC1_D0DIR	AFMUX OTP needed(1)
SDMMC1_D0	PC8 (AF12)	-	-

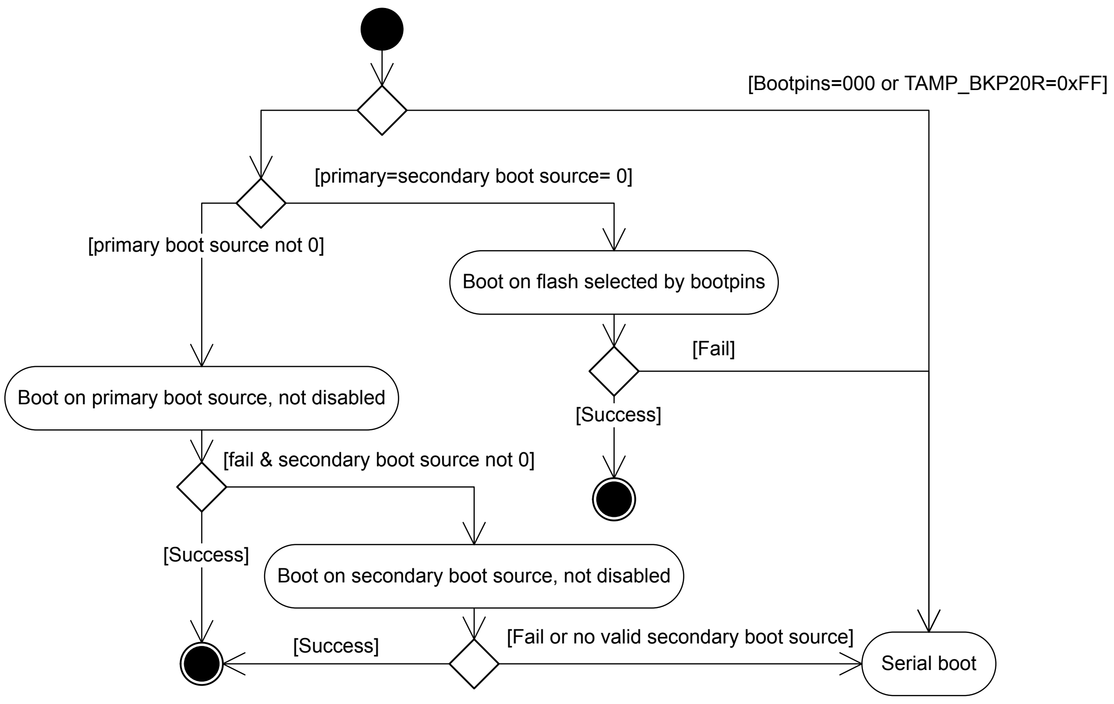
1. *No default value for this AFmux. If needed, the correct value must be defined via OTP words 5 to 7.*

Actions:

Once the integrator configures the TOE and then selects the flash memory boot, the ROM code follows the secure initialization flow defined in Figure 12. No action by the integrator can modify the ROM behavior.

As part of the TOE configuration, the integrator can permanently disable flash memory interfaces by burning the relevant fuses of OTP word 3, as described in the parameters above. Disabling all the flash memory interfaces is not part of the certified TOE configuration.

Figure 12. ROM boot source selection


[Bootpins=000 or TAMP_BKP20R=0xFF]
[primary=secondary boot source= 0]
[primary boot source not 0]
Boot on flash selected by bootpins
[Fail]
[Success]
Boot on primary boot source, not disabled
[fail & secondary boot source not 0]
[Success]
Boot on secondary boot source, not disabled
[Success]
[Fail or no valid secondary boot source]
Serial boot

Errors:

- OTP error: When device fuses are not correctly configured to ensure TOE security, the TOE secure boot procedure detects the problem and blocks the TOE secure boot procedure execution. A reset is generated.
- Bad flash memory interface: In case of failure, the ROM code tries another boot interface, as shown in Figure 12.

Serial boot interfaces

Method of use:

- When the STM32MP135F-DK Discovery kit board is used set the boot switches to the off position (bootpin[2:0]=000).
- Connect the USB Type-A host port of the TOE to the user's computer.
- Power up the board with the TOE and press the reset button.
- Device executes the ROM code that scans bootable UARTs and USB interfaces. More specifically, the ROM code first muxes only the Rx pins of bootable UART instances and scans in parallel all these Rx lines. When an activity is detected on a UART interface, the ROM code muxes the related Tx, all other Rx are unmuxed, and the boot process goes on with this interface. In the case of USB boot detection, all UART Rxs are unmuxed.

Parameters:

- Physical boot pin usage is summarized in Table 14. OTP word 3 is detailed in the next bullet.

Table 14. **Boot device selection via the boot pins and OTP (serial)**

BOOT pins	TAMP_REG[20] (force serial)	OTP word 3 (primary boot source)(1)	OTP word 3 (secondary boot source)(1)	Boot source #1	Boot source #2 if #1 fails	Boot source if #2 fails
0b000	x (don't care)	x (don't care)	x (don't care)	Serial	-	-
0b100	x (don't care)	x (don't care)	x (don't care)	NoBoot	-	-
0b110	!= 0xFF	0 (virgin)	0 (virgin)	Serial	-	-
!= 0b100	!= 0xFF	x (don't care)	x (don't care)	Serial	-	-

1. *Primary and secondary are fields of OTP WORD3.*

- OTP fuse word 3 configuration for serial boot

Note: If OTP disables both UART and USB by OTP, then UART is forced enabled.
If all UART instances are disabled, then all UART instances are forced enabled.

Table 15. **Serial boot configuration in fuse word 3**

OTP field	Offset	Size (in bits)	Default	Description
primary_boot_source	[29:27]	3	0	Keep both to zero (virgin) to boot on the serial interface, if the bootpin is correct (refer to Table 14).
secondary_boot_source	[26:24]	3	0	
boot_source_disable	[23:16]	8	0	Set bit[4] to disable UART serial boot. Set bit[5] to disable USB serial boot.
uart_instance_disable	[14:7]	8	0	If different from zero, each bit disables a UART instance for serial boot. bit[1]: USART2, bit[2]: USART3, bit[3]: UART4, bit[4]: UART5, bit[5]: UART6, bit[6]: UART7, bit[7]: USART8.

Actions:

Once the integrator configures the TOE and the serial boot host, then selects the serial boot method, the ROM code follows the secure initialization flow defined in Figure 12. ROM boot source selection. No action by the integrator can modify the ROM behavior.

As part of the TOE configuration, the integrator can permanently disable USB or some UART instances by burning the relevant fuses of OTP word 3, as described in the parameters above. Disabling all the serial interfaces is not part of the certified TOE configuration.

Note that, in OTP word 3, the user can assist the ROM code to select which HSE it must use.

- If the hse_value is 0b00 (default), the ROM code autodetects HSE by using the HSI clock (from RCC). Recognized values are either 8, 10, 12, 14, 16, 20, 24, 32, 36, 28, 40, or 48 MHz. If none of these values can be recognized, the ROM code considers that the HSE frequency is 24 MHz.
- If the hse_value is 0b01, then the ROM code considers that the HSE frequency is 24 MHz.
- If the hse_value is 0b10, then the ROM code considers that the HSE frequency is 25 MHz.
- If the hse_value is 0b11, then the ROM code considers that the HSE frequency is 26 MHz.

Errors:

- In the TOE configuration two serial boot error scenarios can occur:
  – The boot LED is blinking: authentication error, ROM code is waiting on another active serial interface.
  – The boot LED is not blinking: secure boot failed. Secrets are hidden and the device is frozen, waiting for a board reset.

JTAG interface

Standard JTAG with an SWD interface allows debugging of the TOE and integrator application. It is used according to IEE1149, as described in RM0475. Per Secure installation and preparation of the operational environment (AGD_PRE.1.2C), the TOE is certified with all debug features disabled. The JTAG or SWD interface remains enabled under reset only to inject the RMA password to switch to the RMA_LOCK state.

Method of use:

- The method to provision and lock the RMA password, when the device is OTP-SECURE Open, is described in Section 3.2.2: Secure installation. Alternatively, the method described in AN5510 can be used.
- When the device is in OTP-SECURE Closed state, the user can request to put the device in the RMA_LOCK state, by pushing via JTAG or SWD the 32-bit RMA unlock password in the BSEC JTAG_IN register, that the ROM code finally compares to the password stored in OTP word 56.
- Although not part of the TOE-certified configuration, it is possible to select any JTAG or SWD connection as a source of internal tamper (TAMP input tamper 6).

Parameters:

- RMA password, stored in OTP fuse word 56 (32 bits). Example value: <RMA password>: `0xFACEB00C`

Actions:

- When the user injects the correct RMA password into the certified TOE through JTAG or SWD, the device state is changed to a permanent RMA_LOCK state.
- After the user attempts to inject **three wrong passwords**, the RMA sequence always fails.

Errors:

- Device remains in OTP-SECURE Closed when three wrong RMA passwords are provided.

4.2.3 Security-relevant events (AGD_OPE.1.4C)

Once configured according to Secure installation and preparation of the operational environment (AGD_PRE.1.2C), TOE detects any unauthorized access and any unexpected configuration:

- Images authenticity or integrity violation: When ECDSA signature verification fails to follow any device reset, the TOE does not start to execute the corrupted image.
- Images decryption error: When a corrupted decrypted image is found following any device reset, the TOE does not start to execute the corrupted image, and instead automatically switches to serial boot on authorized interfaces.
- OTP values violation: In case OTP values are not correctly configured to ensure the TOE security, or some OTP values are read from OTP memory with disturb/error status, the ROM code launched after any device reset detects the problem and blocks the TOE secure boot procedure execution. A device reset is mandatory to try to recover the situation. It only works if the OTP disturb event is transient.
- JTAG access violation: Once TOE security is fully configured, the JTAG or SWD connection can only be used to inject the RMA password, which means any other usage like debug does not generate any access violation because it has no access.
  - Optionally, the user can enable an intrusion event as soon as a connection is detected on the JTAG or SWD interface, blocking access to all protected memories (flash memory, protected SRAMs, and backup registers). This option is not enabled in the certified TOE configuration.
- RMA attempts: No more than three attempts are possible. Any further attempt is ignored.
- Tampering attempt: The default hardware protections in BSEC and SCA-protected crypto peripherals are available in the certified TOE configuration. See secure storage below.
  - To add an extra layer of security, the user can enable multiple tamper events in the TAMP peripheral. This option is not enabled in the certified TOE configuration.
- Secure storage: Fault injection in the OTP memory is detected, thanks to the "disturb" readout status in BSEC. SCA-protected peripherals (SAES, PKA) automatically go into error mode when a hardware fault is detected. Last but not least, any software attempt to overwrite partially a key stored in SAES registers automatically erases the key and triggers a key error.
  - Additionally, when the feature is activated, any keys encrypted with DHUK or BHK are not usable when a tamper event occurs.

4.2.4 Security measures (AGD_OPE.1.6C)

The trusted integrator personalizes the TOE and uses the TOE security functionalities so that the IoT product meets its final certification target. The integrator is trusted and does not attempt to thwart the TOE security functionalities, nor attempt to bypass them.

To achieve the above TRUSTED_INTEGRATOR and TOE_PREPARATION security objectives, the following measures must be taken.

- The integrator must verify the genuineness of the TOE as described in Secure acceptance.
- The integrator must follow all the guidelines described in User-accessible functions and privileges (AGD_OPE.1.1C) and Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) to integrate the TOE into a full IoT solution.
- Once the integrator finishes its IoT product development and wants to start to validate the complete product with the security fully activated, he must follow the installation guidelines described in Secure installation and preparation of the operational environment (AGD_PRE.1.2C) to validate that the IoT product is in its final security configuration.
- Once the integrator finishes its IoT product development and wants to start production, the integrator must securely provision the TOE assets as stated in User-accessible functions and privileges (AGD_OPE.1.1C).

Additionally, TOE secrets defined as the Root of Trust-related sensitive material must be protected. To achieve this TOE_SECRETS security objective, the following measures must be taken.

- The integrator must protect the integrity and confidentiality of the private cryptographic key used to build the FSBL images.
- The integrator must protect the integrity of the public key hash table of the TOE until it is programmed and protected in integrity inside each TOE.
- The persons responsible for the application of the procedures described in TOE preparative procedures, and the persons involved in the delivery of the TOE must have the required skills and must be aware of the security issues.

4.2.5 Modes of operation (AGD_OPE.1.5C)

The TOE operates after a reset of the device by executing the TOE ROM code that ensures the immutable first stage of the STM32MP13xx trusted boot chain:

- FSBL installation mode: TOE loads FSBL image from the selected boot interface. When flash memory is selected at least two copies of this image are available
- Authentication/decryption mode: TOE authenticates and optionally decrypts the FSBL image
- Operative mode: if the image is authenticated, TOE jumps to the FSBL image entry point in SYSRAM

Note: TOE uses the hardware isolation techniques described in TrustZone® and MMU isolation usage of User-accessible functions and privileges (AGD_OPE.1.1C).

Supported boot interfaces are flash memories (serial NOR, serial or parallel NAND, SD card, e.MMC) and serial boot interfaces (UART, USB). As part of the TOE configuration, it is possible to deactivate some of those boot interfaces.

When the FSBL image needs to be updated, the IoT application must properly program it in the two FSBL slots of the target flash memory, then following a device reset the TOE ROM code verifies it and if a new public key has been used for authentication the public keys with lower indexes are permanently revoked.

In case the TOE OTP values are not correctly configured to ensure the TOE security, following a reset the TOE ROM code detects the problem and blocks the TOE secure boot procedure execution. As OTP programming is permanent this situation can repeat each reset, unless the RMA sequence is used to decommission the device, via JTAG or SWD.

In its certified configuration, in case the TOE detects any violation, as described in Security-relevant events (AGD_OPE.1.4C), the TOE hides its secrets and waits for a device reset unless described otherwise in the above section.

The TOE does not support the update or patching of its immutable ROM code. However, it offers the ability for customers to update their part of the trusted boot chain (the FSBL image).

5 Technical annexes

5.1 Boot from parallel and serial NANDs

Supported parallel NANDs

The ROM code supports parallel NAND with the following parameters.

Table 16. **Parallel NAND support by ROM code**

Block size (Kbytes)	Page size (Kbytes)	Data width	ECC (bits and code)
128	2	8, 16	4 (bch), 8 (bch), 1 (hamming)
256	4		
512			
512	8		

1. The ROM code supports both parallel NAND with or without on-die ECC. The ECC given here is for NAND without on-die ECC.

Supported serial NANDs

The ROM code supports serial NAND with the following parameters.

Table 17. **Serial NAND support by ROM code**

Block size (Kbytes)	Page size (Kbytes)
128	2
256	4
512	
512	8

Some serial NANDs are multiplanes. The ROM code supports such multiplane features when the spinand_need_plane_select OTP bit is set to 1 in OTP word 9.

Configuration

To read a NAND flash memory, the ROM code needs to know the page size, the block size, and the number of blocks. For parallel NAND, it also needs to know the width and the number of ECC bits.

The ROM code detects NAND parameters storage location by checking OTP bit nand_param_stored_in_otp in OTP word 9.

If OTP bit nand_param_stored_in_otp in OTP word 9 is equal to 0, the default value, then the NAND must provide an ONFI-compatible parameter table in which the ROM code looks for NAND parameters. "ONFI compatible" means that a NAND provides a get parameter feature that returns a table where at least the parameters needed by ROM code are located at standard ONFI offsets.

The following table shows which parameters are needed by the ROM code.

Table 18. **ROM code parameters for parallel or serial NAND**

Parameter table offset	Description	Needed for parallel NAND	Needed for serial NAND
6	Supported features (used to determine data width)	X	-
[83:80]	Number of data bytes per page	X	X

Parameter table offset	Description	Needed for parallel NAND	Needed for serial NAND
[85:84]	Number of spare bytes per page	X	-
[95:92]	Number of pages per block	X	X
[99:96]	Number of blocks per unit	X	X
112	Number of ECC bits correctability	X	-

Note: *Serial NAND memories are not ONFI compliant but most of them are ONFI compatible.*

The number of ECC bits is a particular case, as it can be set by OTP even if OTP bit nand_param_stored_in_otp is equal to 0 in OTP word 9. This is to allow the user to override the recommended number of ECC bits given by the parameter table of an ONFI NAND.

5.2 How to update OTP with U-Boot

This section explains how to update manually the OTP with the U-Boot fuse command.

The fuse command

Caution: *Programming fuses is an irreversible operation! This may brick your system. Use this command only if you are sure of what you are doing!*

The fuse command allows you to update the OTP words in U-Boot:

- **sense/program** to access directly the OTP value (for a permanent update)
- **read/override** to access only the shadow cache value (for a temporary update).

```
Board $> help fuse
fuse - Fuse sub-system

Usage:
fuse read <bank> <word> [<cnt>] - read 1 or 'cnt' fuse words, starting at 'word'
fuse sense <bank> <word> [<cnt>] - sense 1 or 'cnt' fuse words, starting at 'word'
fuse prog [-y] <bank> <word> <hexval> [<hexval>...] - program 1 or several fuse
words, starting at 'word' (PERMANENT)
fuse override <bank> <word> <hexval> [<hexval>...] - override 1 or several fuse
words, starting at 'word'
```

See https://github.com/STMicroelectronics/u-boot/blob/v2020.10-stm32mp/doc/ README.fuse for details.

Note: *`<bank> = 0` provides access to the 96 OTP words of the TOE device.*

Simple OTP examples

1) Read OTP values for OTP 57 and 58 (two OTP words)

```
Board $> fuse sense 0 57 2
Sensing bank 0:
Word 0x00000039: 42e18000 0000e448
```

2) Check the lock status of OTP 57 - 60 (four words at index 57 = `0x39`)

```
Board $> fuse sense 0 0x10000039 4
Sensing bank 0:
Word 0x10000039: 00000001 00000001 00000001 00000000
```

3) Display shadow values for all OTPs

When only the 32 lower OTPs are accessible (OTP-SECURE close device):

```
Board $> fuse read 0 0 32
Reading bank 0:
Word 0x00000000: 00000017 00008001 00000000 00000000
Word 0x00000004: 00000000 00000000 00000000 00000000
Word 0x00000008: 00000000 82004000 00000000 00000000
Word 0x0000000c: 7d04f0db 00470022 33385115 34383330
Word 0x00000010: 22986562 27010551 7a470140 06cc1608
Word 0x00000014: 5e560054 00000000 00000000 401a300c
Word 0x00000018: ffffffff ffffffff ffffffff ffffffff
Word 0x0000001c: ffffffff ffffffff ffffffff ffffffff
```

When all the 96 OTPs are available (OTP-SECURE open device):

```
Board $> fuse read 0 0 96
Reading bank 0:
Word 0x00000000: 00000017 00008000 00000000 00000000
Word 0x00000004: 00000000 00000000 00000000 00000000
Word 0x00000008: 00000000 00000000 00000000 00000000
Word 0x0000000c: 7cf5f0f9 00410032 33385116 34383330
Word 0x00000010: 129675aa 2931215e 7a550000 069013ec
(...)
Word 0x00000050: 00000000 00000000 00000000 00000000
Word 0x00000054: 00000000 00000000 00000000 00000000
Word 0x00000058: 00000000 00000000 00000000 00000000
Word 0x0000005c: 00000000 00000000 00000000 00000000
```

4) Override value for one OTP

```
Board $> fuse override 0 0x0000005c 1
Overriding bank 0 word 0x0000005c with 0x00000001...
Board $> fuse read 0 0x0000005c
Reading bank 0:
Word 0x0000005c: 00000001
Board $> fuse sense 0 0x0000005c
Sensing bank 0:
Word 0x0000005c: 00000000
```

Revision history

Table 19. Document revision history

Date	Revision	Changes
29-Nov-2022	1	Initial release.
25-Jan-2023	2	Second release.
9-Sep-2025	3	Updated: • Section 1: General information • Section 2: Reference documents • Section 3.1: Secure acceptance • Section 3.2.2: Secure installation • Section 4.2.1: User-accessible functions and privileges (AGD_OPE.1.1C) • Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) • Section 5.2: How to update OTP with U-Boot

Contents

List of figures

List of tables