STM32N6xx security guidance for SESIP level 3 certification

Introduction

This document describes the preparative procedures and operational user guidance of the STM32N6xx microprocessor to make a secure system solution according to the SESIP level 3 certification scheme.

The security guidance described in this document applies to any board based on the devices listed in the table below.

Table 1. Applicable products

Reference	Products
Platform identification	STM32N6xx

1 General information

This document applies to STM32N6xx Arm®-based MCUs.

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

arm

Table 2. Specific acronyms

Acronym	Description
CID	Compartment identifier
HUK	Hardware unique key
HW	Hardware
IoT	Internet of Things
MCU	Microcontroller unit
PKA	Public key accelerator engine
RIF	Resource isolation framework
RMA	Return material for analysis
SAES	Secure AES engine
SCA	Side-channel attack
SESIP	Security evaluation standard for IoT platforms
SFR	Security-functional requirement
TD	Trusted domain
TDCID	Trusted domain compartment identifier
TOE	Target of evaluation

2 Reference documents

Table 3. List of reference documents

Reference	Document title and revision
[RM]	Reference manual *STM32N647/657xx Arm®-based 32-bit MCUs* (RM0486), release 1
[ES]	Errata sheet *STM32N6xxxx device errata* (ES0620), release 1
[ST]	Technical note *STM32N6x7 Security Target for SESIP level 3 certification* (TN1562), revision 1
[UM2237]	User manual *STM32CubeProgrammer software description* (UM2237), version 27
[UM2238]	User manual *STM32 Trusted Package Creator tool software description* (UM2238), revision 15
[UM2542](1)	User manual *STM32 key generator software description* (UM2542), revision 5
[UM2543](1)	User manual *STM32MPx series signing tool software description* (UM2543), revision 5
[Readme_PRGFW]	STM32PRGFW-UTIL, revision 1.0.4 *https://github.com/STMicroelectronics/STM32PRGFW-UTIL/blob/main/README.md*
[IEEE1149]	IEEE 1149.1–2013
[WIK]	stm32mcu

1. *UM2542 and UM2543 are written for STM32MPx chips, but they are also applicable to STM32N6.*

3 TOE preparative procedures

This section provides useful information for ensuring that the target of evaluation (TOE) has been received and installed in a secure manner as intended by the developer.

- 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 prepare the final product.
- Secure installation: procedure to program and configure the product to be prepared.

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 MCU device, with corresponding firmware packages that can be obtained from *www.st.com*. Refer to the cover page for the applicable devices.

How to accept an STM32N6xx MCU device

The device is delivered in the Closed_Unlocked default state. TOE genuineness can be verified by reading the boot ROM version registers at `0x18001000` and `0x18001004` (or `0x08001000` and `0x08001004` for nonsecure reads).

- A read of `0x18001000` returns a value of `0x0000 8604`, meaning the device identifier (ChipVersion) is `0x486`.
- A read of `0x18001004` returns a value of `0x0000 0200`, meaning the cut identifier (CutVersion) is 2.0.
- A read of `0x18001008` returns a value of `0x0000 0001`, meaning the mask identifier (MaskVersion) is 1.
- A read of `0x1800100C` returns a value of `0x0000 0501`, meaning the boot ROM identifier (BootromVersion) is 5.1.

An STM32N6xx device with these settings in the ROM is said to have a revision ID "Rev B". If STM32CubeProgrammer attaches to an STM32N6xx and finds different settings in these registers, it does not report it as a 'Rev B' Revision.

To connect using STM32CubeProgrammer on a Closed_Unlocked part (selecting dev boot)

- Set BOOT1 switch SW1 to the ON position to select dev_boot (BOOT0 switch SW2 setting is irrelevant).
- Set the jumper JP2 on 1-2: 5V_STLK
- Connect the board to the user computer running STM32CubeProgrammer via the STLINK_V3EC (CN6) port.
- Power up the board and press the reset button.

The STM32CubeProgrammer can now be used to verify the TOE genuineness via the command-line interface:

```
./STM32_Programmer_CLI -c port=swd
```

The expected answer from the device is:

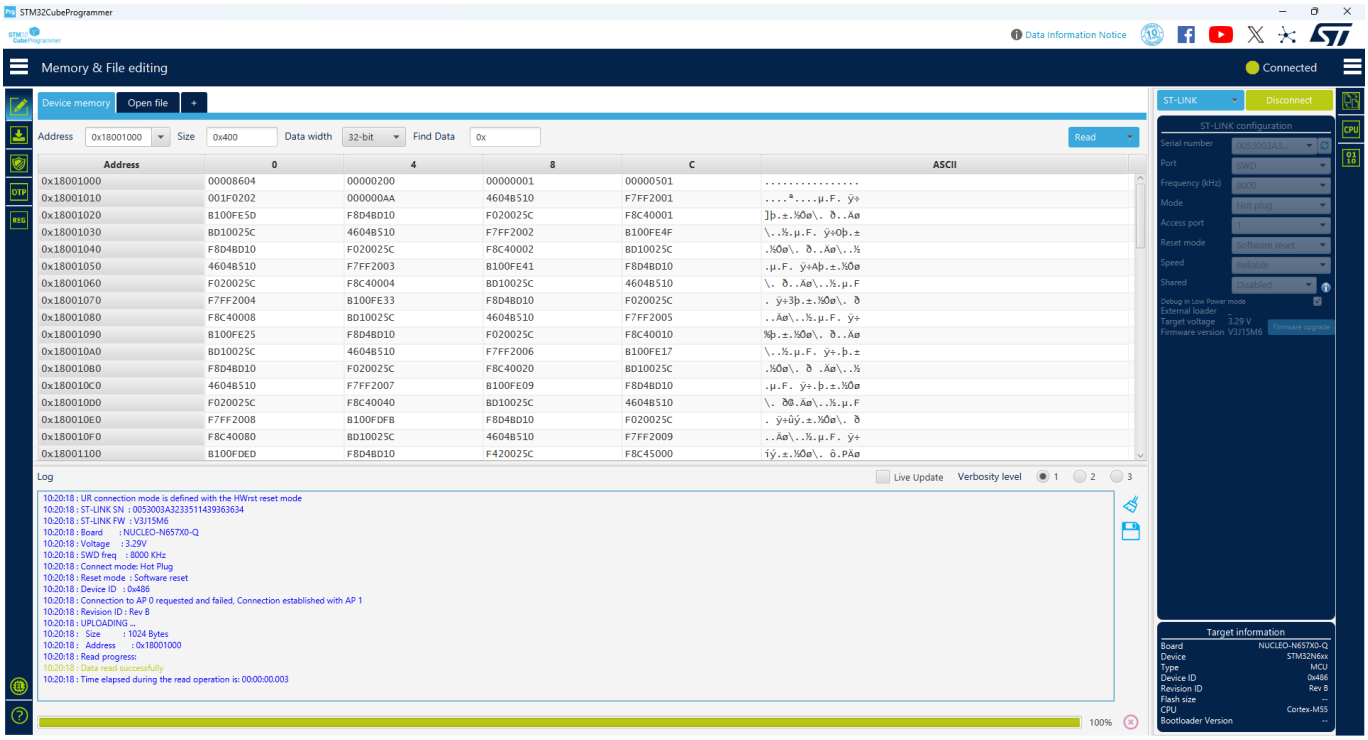
```
        -------------------------------------------------------------------
                              STM32CubeProgrammer v2.18.0
       -------------------------------------------------------------------

ST-LINK SN  : 0050003E3233511439363634
ST-LINK FW  : V3J15M6
Board       : NUCLEO-N657X0-Q
Voltage     : 3.29V
SWD freq    : 8000 KHz
Connect mode: Normal
Reset mode  : Software reset
Device ID   : 0x486
Revision ID : Rev B
Device name : STM32N6xx
Device type : MCU
Device CPU  : Cortex-M55
BL Version  : --
```

Alternatively, it can be done via the STM32CubeProgrammer graphical user interface (GUI) as follows.

- Open STM32CubeProgrammer GUI and choose the **ST-Link** (set by default) in the connection picklist and click on the refresh button. The serial number is displayed if **ST-Link** is detected.
- Click on the green button **Connect**.
- The log panel displays the product ID as shown in Figure 1.

Figure 1. STM32N6xx acceptance using STM32CubeProgrammer GUI



How to select software packages?

Even though software packages are not part of the TOE, they can be useful to the procedure defined in Section 3.2: Secure installation and secure preparation (AGD_PRE.1.2C).

3.2 Secure installation and secure preparation (AGD_PRE.1.2C)

This preparative procedure section describes all the steps necessary for the operational environment in accordance with the security objectives for the operational environment as described in [ST].

3.2.1 Setup procedures

Hardware setup procedure

To set up the hardware environment, the development board must be connected to the user's PC via SWD in 'DEV BOOT' mode. This connection with the PC running STM32CubeProgrammer allows the user to:

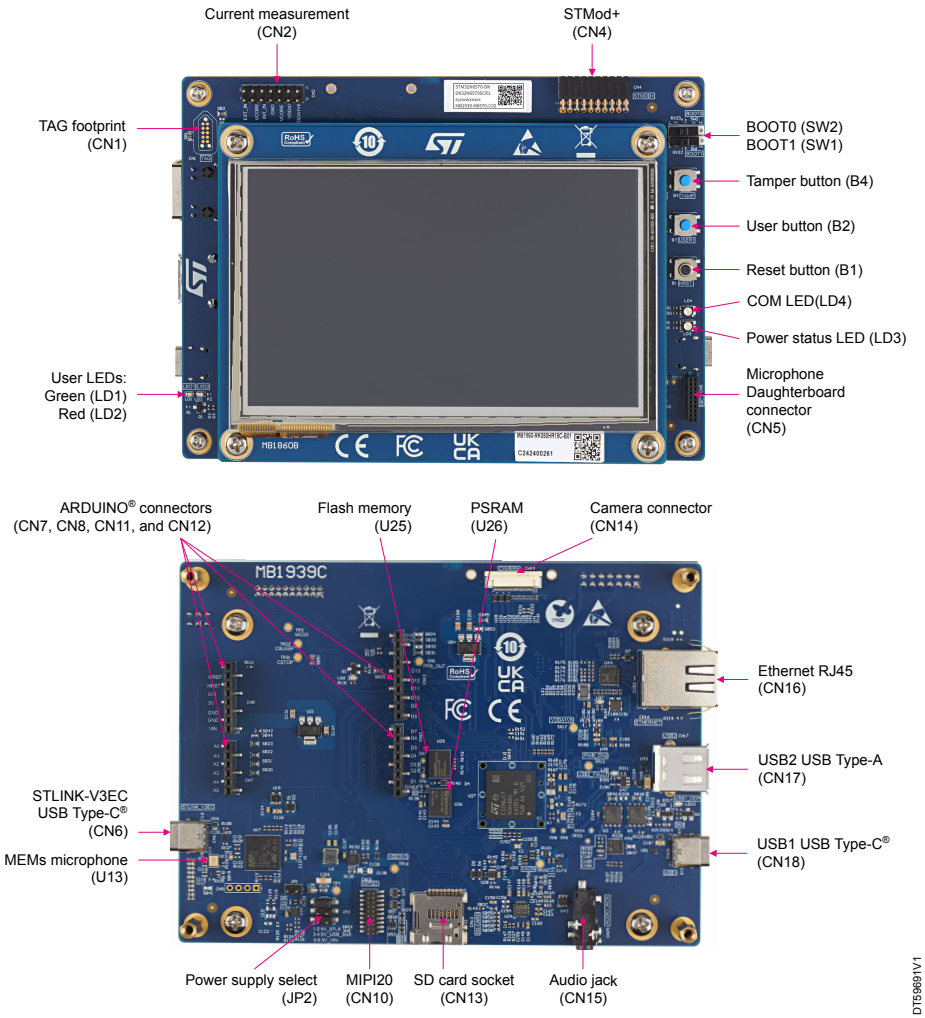
- Configure the nonvolatile memory (OTP and flash) of the platform
- Debug the platform when the protections are disabled

If the STM32N6570-DK board shown in Figure 2 is used, the hardware setup procedure is:

- Set BOOT1 switch SW1 to the ON position (BOOT0 switch SW2 is irrelevant, but it can be left OFF).
  - This configuration is called DEV BOOT.
- Set the jumper JP2 on 1-2: 5V_STLK
- Connect the board to the user's computer via the STLINK-V3EC CN6 port of the board.
- Power up the board and press the reset button.

If another board is used, refer to the guidance in Section 4.2.2.

Figure 2. STM32N6570-DK board



Software setup procedure

All the steps to install and use starter packages are described in the ST wiki pages ([WIK]).

More specifically, the following tools are required for the preparation and installation of the TOE:

- STM32CubeProgrammer v2.18.0, described in [UM2237]
  - Includes STM32 signing tool and STM32 key generator

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

3.2.2 Secure installation

The STM32N6x product preparation is done in six steps, to get a complete installation with fully activated security configuration. All the steps must be performed successfully to have the product in its certified configuration. It is assumed that the user performs these steps in a trusted secure environment.

Step 1: 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 (the “active key”). The revocation mechanism requires that all eight keys are generated at the first step.

With the same tool, the user generates the hash of a table composed of the hashes of those eight public keys. This hash must then be stored in OTP words 160 to 167.

Note: Activating a root key in the TOE revokes all the previous ones. For example, if key#3 is activated and used for boot, then keys#1 and #2 are permanently revoked by an automatic process managed by the ROM code.

To generate the keys and the hash of the table composed of the eight hashes of the eight public keys, the STM32 keygen tool can be used with the following command:

```
$ STM32_KeyGen_CLI.exe -abs . -pwd abc0 abc1 abc2 abc3 abc4 abc5 abc6 abc7 -n 8
```

The private keys, public keys, and hash table are generated in the working directory.

The private keys are encrypted with the default algorithm aes256 and `abc0-7` are the passwords for each private key.

If the user wants to encrypt the firmware, a 128-bit encryption key must be created.

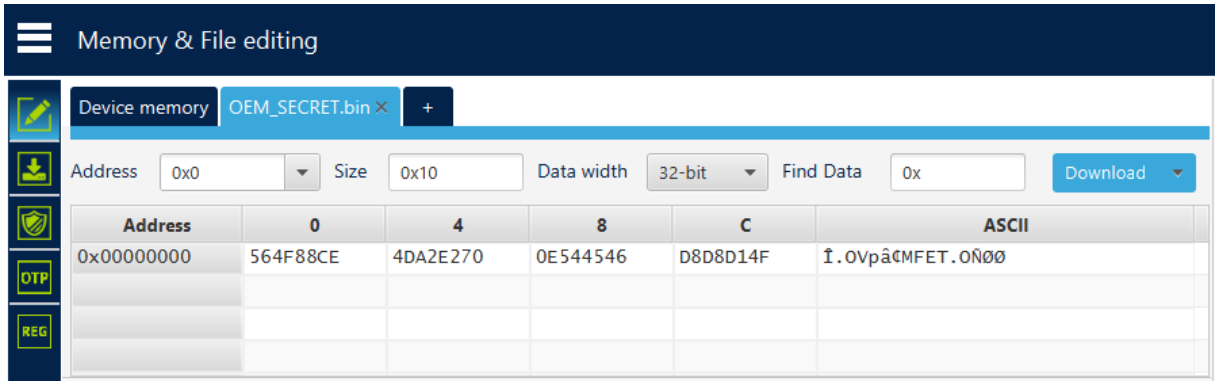
To generate a random key, the keygen tool can be used with the following command:

```
$ STM32_KeyGen_CLI.exe -abs . -rand 16 OEM_SECRET.bin
```

This key is stored in OTP words 364 to 367 of the TOE in step 2.

OEM_secret.bin can be viewed with STM32CubeProgrammer as follows:

Figure 3. Viewing OEM secret with STM32CubeProgrammer



A 32-bit derivation constant must also be created to be stored in the encrypted FSBL extension header (using the `-encdc` signing tool option). With this information, the STM32 signing 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.

Step 2: Provisioning of secrets

The user must program the OTP 160 to 167 with the *publicKeysHashHashes.bin* table previously generated. If these OTPs are not programmed, the signed firmware is not executed in a closed-locked state because the signature is checked with these values. In the Closed_Unlocked state, the signature is checked but it is ignored if it is wrong.

STM32CubeProgrammer tool can be used to update the OTP:

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -el OTP_FUSES_STM32N6xx .stldr
-otp fwrite lock publicKeysHashHashes.bin word=160
```

To check if OTPs are programmed correctly, this command can be used:

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -el OTP_FUSES_STM32N6xx .stldr
-otp displ
```

If the user wants to use signed and encrypted firmware, OTP 364 to 367 must also be programmed with the *OEM_secret.bin* file previously generated for decryption.

These OTP locations cannot be read back once programmed. They can only be read by ROM code.

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -el OTP_FUSES_STM32N6xx .stldr
-otp fwrite lock OEM_SECRET.bin word=364
```

Step 3: Provisioning and locking the RMA password

The user can generate a file called *RMA_password.bin* that contains a 128-bit secret.

To generate a random key, the keygen tool can be used with the following command:

```
$ STM32_KeyGen_CLI.exe -abs . -rand 16 RMA_password.bin
```

STM32CubeProgrammer tool can be used to program OTP 256 to 259 with this file.

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -el OTP_FUSES_STM32N6xx .stldr
-otp fwrite lock RMA_password.bin word=256
```

Step 4: FSBL image signing/encrypting

The FSBL image loaded by the ROM must be signed to be used on the certified TOE.

STM32 signing tool [UM2543] can be used for that purpose. Suppose that the image is in a file called *binary_file.bin*. The user can either:

- Sign the *binary_file.bin* file with the following command (to get *trusted_file.bin*):

```
$ STM32_SigningTool_CLI.exe -bin binary_file.bin -pubk publicKey00.pem publicKey01.pem
publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem publicKey06.pem
publicKey07.pem -prvk privateKey00.pem -iv 0x00000000 -pwd abc0 -of 0x80000001
-la 0x34180000 -t fsbl -o trusted_file.bin -hv 2.3
```

- Sign and encrypt the *binary_file.bin* file with the following command (to get *trusted_encrypted_file.bin*):

```
$ STM32_SigningTool_CLI.exe -bin binary_file.bin -pubk publicKey00.pem publicKey01.pem
publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem publicKey06.pem
publicKey07.pem -prvk privateKey00.pem -iv 0x00000000 -pwd abc0 -of 0x00000003 -la
0x34180000 -t fsbl -encdc 0x25205f0e -enck OEM_SECRET.bin -o trusted_encrypted_file.bin
 -hv 2.3
```

The `-pwd abc0` option here means that the image is signed with the index0 key. If an image is signed with a higher index key, then when the ROM authenticates it, it blows fuses to ensure that it never again accepts images signed with a lower index key, revoking those earlier keys.

Step 5: Image programming

Once the image is signed, it can be programmed into the external flash memory on the target board with the STM32CubeProgrammer tool. Supported flash memory and its associated flash memory mapping are described in Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C).

Enter this for a signed firmware:

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -w trusted_file.bin
0x70000000 -el MX66UW1G45G_STM32N6570-DK.stldr
```

Enter this for signed and encrypted firmware:

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -w trusted_encrypted_file.bin
0x70000000 -el MX66UW1G45G_STM32N6570-DK.stldr
```

Note: At this step, the device is in the Closed_Unlocked state. At boot, the boot ROM checks the signature but it is ignored if wrong: unauthenticated images can still be used and executed. If the installed firmware is encrypted, the firmware is executed at boot only if the decryption succeeds.
Once in the Closed_Locked_Provd state, if the check of the signature is wrong, the firmware is not executed.

Step 6: Lock the device

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.

Burn the following OTP to change the product state from default Closed_Unlocked to Closed_Locked_Provd:

- Write `0x00001FFF` in the word 18.
- Write `0x00100000` in the word 124 to the disabled scan state.
- Perform a power cycle.

Command to be used with STM32CubeProgrammer tool:

```
$ STM32_Programmer_CLI.exe -c port=SWD ap=1 speed=fast -el OTP_FUSES_STM32N6xx
.stldr -otp write word=124 value=0x100000 word=18 value=0x1FFF
```

Note: Words 18 and 124 must be programmed with these values at the same time (that is, in the same reset cycle).

Note: Bits [12:9] of word 18 are the enable_fingerprint bits. This is a mechanism to protect against glitch attacks.

Bits [25:22] of word 18 can be blown to forbid debug opening (if the boot ROM sees these bits blown, it prevents the FSBL from opening any sort of debug). So, one can choose to write `0x03C01FFF` instead of `0x00001FFF` in the word 18 when arranging to change the state (as above), but if not, one can disable debug later on by blowing `0x03C00000` in the word 18.

Once the device is in the Closed_Locked_Provd state, unsigned first-stage bootloaders are no longer supported on the target device.

3.2.3 Certified configuration

After installation, the certified device configuration is:

- Life cycle: Closed_Locked_Provd
- Boot image verification: activated
- Boot image encryption: optional
- Internal and external tamper sources in the TAMP peripheral: deactivated (default), or not.

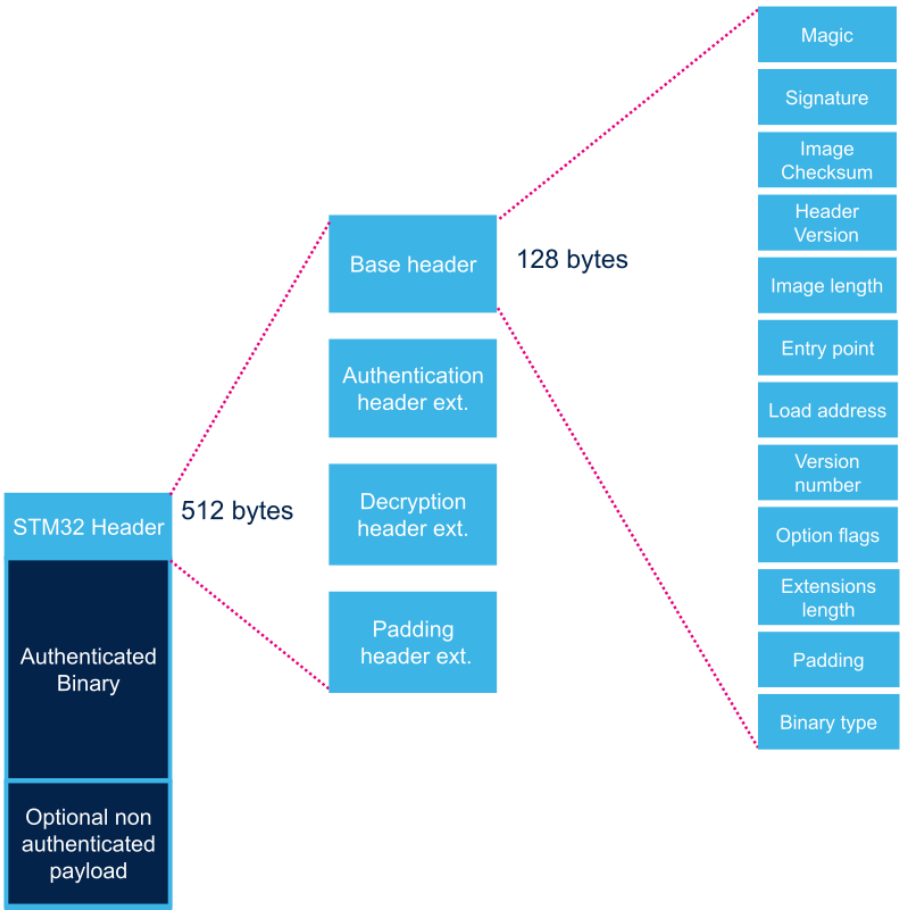
After installation, the certified device fuse configuration is:

- BSEC status: *BSEC-closed* (refer to [RM] Section 4.3.7)
- Root key hash (OTP_ROT_HASH): provisioned in OTP words 160 to 167.
  - The platform uses it to implement authentication and integrity protection.
- Two algorithms are supported for ECDSA signature verification: NIST P-256 or Brainpool 256T1.
- OEM secret key (OEM_SECRET_FOR_CRYPTED_BOOT): provisioned in OTP words 364 to 367.
  - OEM secret used to derive FSBL decryption key
- RMA password (OTP_RMA_LOCK_PSWD): provisioned and locked in OTP words 256 to 259.
- Bit 4 "fsbl_decrypt_prio": set in OTP word 18.
  - To protect the FSBL decryption key against side-channel attacks.
- Bit 0 "Boot_traces_disabled": set in OTP word 16.

Refer to Section 5 *OTP mapping* in [RM] for details on the fuse configuration in the devices (including endianness).

Each binary image (signed or not) loaded by the platform needs to include a specific STM32 header (version 2.3), 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 roles

The user role integrator is the only role involved in the preparative TOE procedure. The integrator is responsible to:

- Receive the TOE.
- Perform the preparative procedures as described in Section 3: TOE preparative procedures.
- And integrate the TOE into a final product.

Once the TOE is in certified configuration, the user-operational guidance is described in Section 4.2: Operational guidance for the integrator role.

The integrator has full access to the TOE security features, as the STM32N6xx devices are delivered in the Closed_Unlocked state with the secure boot feature deactivated. 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 final product. To this end, the system integrator has access to interfaces that are unavailable to other users, as described in Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C). The integrator cannot change any parts inside the TOE but must configure the TOE to make it functional in the final secure state. The integrator can change parts outside the TOE without compromising the security of the TOE.

Follow the procedures described in Secure acceptance to check if the TOE is acceptable for the secure configuration. The secure configuration of the TOE might be impacted when changing some parts of the TOE, but may also be impacted when changing some parts located outside the TOE scope. This section describes the changes that the integrator can make and highlights what is covered in the evaluation scope and what might impact the secure configuration of the TOE.

Boot image authentication scheme

The secure boot on the TOE is certified in the ECDSA P-256 asymmetric crypto scheme configuration, as described in Section 3.2: Secure installation and secure preparation (AGD_PRE.1.2C). Such configuration, stored in a dedicated authenticated part in the STM32 header, is summarized in Table 4.

Table 4. STM32 header information for authentication

Name		Length (in bits)	Byte offset	Description
Version number		32	96	Image version (monotonic number)
Extension flags		32	104	Bit[0]=1: Authentication is enabled(1) Bit[1]=1: Decryption is enabled(2) Bit[31]=1: Padding extension is enabled(3)
Authentication extension header	Extension type	32	0	4 bytes in big-endian: 'S', 'T', 0x00, 0x02 = 0x53540002
	Extension length	32	4	Number of bytes of header extension = 340
	Public key index	32	8	Index of the public key to be used
	Public key number	32	12	Number of public keys in table = 8
	ECDSA algorithm	32	16	1: P-256 NIST; 2: brainpool 256
	ECDSA public key	512	20	ECDSA public key to be used to verify the signature
	Algo and public key1 hash	256	84	Hash of (ECDSA algorithm and public key1)
	...	...	...	Hash of (ECDSA algorithm and public keyX), (X=2 to 7)
	Algo and public key8 hash	256	308	Hash of (ECDSA algorithm and public key8)

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

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 BSEC interface in Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C).

Version number and boot image revocation

The platform always verifies the image version and key revocation, as defined below:

- If the platform finds that the version listed in the STM32 header (Table 4) is greater than the one stored in the OTP word 20, it updates the version in OTP by blowing additional fuse bits, forbidding future usage of an older version. It is possible to update up to 32 times.
- The platform manages a revocation mask for signature authentication. If it detects a change between the key used in the header and the revocation mask in OTP word 17, then the ECDSA key counter is updated.

Figure 5 illustrates how keys with an ID inferior to the active one are revoked. The mechanism for firmware versions is similar.

Figure 5. Usage of revocation mask stored in OTP

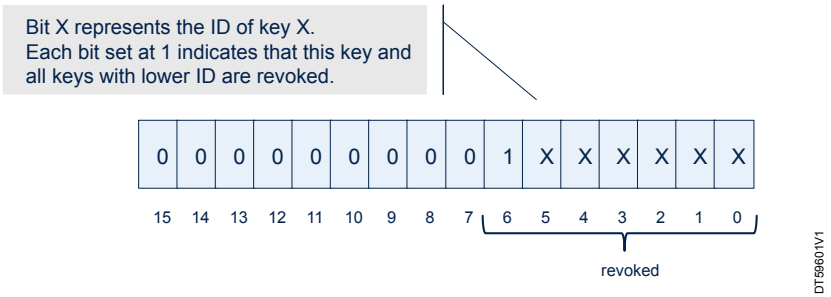


Image encryption

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. To ensure protection against side-channel attacks, bit 4 of OTP word 18 must be blown (use SAES).

Table 5. STM32 header information for encryption

Name		Length (in bits)	Byte offset	Description
Extension flags		32	104	Bit[0]=1: Authentication is enabled(1) Bit[1]=1: Decryption is enabled(2) Bit[31]=1: Padding extension is enabled(3)
Decryption header extension	Extension type	32	0	4 bytes in big-endian: 'S', 'T', 0x00, 0x01 = 0x53540001
	Extension length	32	4	Number of bytes of header extension = 32
	Key size	32	8	Size of extension key (128 bits)
	Derivation constant	32	12	Constant used to derive the decryption key from the controller key stored in OTP
	Plain hash	128	16	128 MSB bits of plain payload SHA256

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

Jump to the boot image

At boot, the default protections are set by the hardware to comply with the mandatory requirements to allow the platform to securely authenticate, and if necessary, decrypt the firmware. The secure ROM code starts as the TDCID and is allowed to control the entire system (RIF configuration).

At boot, the ROM code authenticates the firmware located in SYSRAM at 0x34180000:

Related boot information is given in the STM32 header in the table below:

Table 6. STM32 boot information

Name	Length (in bits)	Byte offset	Description
Image length	32	76	Length of image in bytes
Image entry point	32	80	Entry point of the image
Binary type	32	108	Used to check binary type. It must be set to 0x30.

Fault injection attack countermeasures

To meet the target of platform resistance against physical attackers, the integrator must implement the following software countermeasures within its application when performing sensitive operations (including cryptographic ones):

- Redundancy:
  - For example, perform sensitive operation twice and verify that the results are equal, or verify decryption operations with encryption and encryption operations with decryption.
- Random timing jitter:
  - For example, apply a random loop (using an RNG peripheral) before a sensitive operation.
- Execution control flow:
  - For example:
    - Use a finite state machine in which transitions are verified to be legit.
    - Use a scattered known computation with a verified result at the end.

In case of errors, the integrator should implement a security response, for example, a platform reset.

True random number generator

For the device to generate random numbers as specified in NIST SP800-90B, the integrator must use the TRNG peripheral with the A configuration. Refer to Subsection 47.6.2 "Validation conditions" of the RNG section in the [RM].

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 [RM] (Section 48 for the SAES and Section 52 for the PKA).

- SAES
  - AES-128 and AES-256 encryption/decryption, authenticated encryption or decryption, cipher-based message authentication code computation
- PKA
  - Modular exponentiation for RSA decryption (MODE=`0x3`), scalar multiplication (MODE=`0x20`), and signature (MODE=`0x24`) for ECC
- RNG for cryptographic key generation (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 (*extra random bits* or *testing candidates*). The integrator must select one of those two methods when using the RNG peripheral to compute the ECC private keys.

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

When implementing an RSA or ECC function with the PKA peripheral the integrator must check that, when writing sensitive data such as nonces or private keys into the PKA RAM, the written data is correct and has not been altered before starting the PKA operation.

When implementing an AES function with the SAES peripheral, the integrator must follow the below guidelines to meet SESIP security assurance level 3 (or equivalent).

- 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.
- Implement redundancy when verifying results. The integrator must implement a random timing jitter between each result comparison.
- Implement a control flow that verifies that each step mentioned above is completed.

Additionally, the integrator should activate the internal tamper 9 dedicated to cryptographic peripherals fault.

Finally, when an error related to the aforementioned countermeasures is detected, the integrator must take appropriate action according to its security policy (as an example, the application might reset the system).

Cryptographic key storage

Using the SAES peripheral the integrator can protect the integrity and confidentiality of AES 128 or 256-bit keys in its KeyStore, using encryption or authenticated encryption with 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.

If the integrator needs to check that an unwrapped key is not corrupted, he can run a known AES calculation with it and compare the result with an expected value.

Refer to [RM] Section 48.4.14 *SAES operation with wrapped keys* for details.

Note: Keys stored in SAES (in registers or as derived hardware- unique key) are not usable in case of a tamper event, or when the HKLOCK sticky bit is set in the BSEC_LOCKR register.

As with any software dealing with sensitive data, the software driving the SAES peripheral should follow the guidelines described in Fault injection attack countermeasures, for example, timing randomization and control flow.

Nonvolatile storage of secrets in OTP

BSEC implements hardware countermeasures against physical attacks that TOE uses to go to a locked state before the attacker compromises any of the other functional requirements.

The platform uses the BSEC reading, writing, or programming mode lock capability of BSEC to isolate itself from the application. The integrator can do the same for its assets.

After any reading or programming request in BSEC, the peripheral reports a specific status structure when the BUSY bit is cleared. When this structure is different than 0, the read or program operation might have failed.

To benefit from the automatic protection of OTP secrets, when the device is decommissioned for return material for analysis (field return), secure applications must store their secret keys in the OTP words 256 to 367.

For more information on BSEC, refer to [RM] Section 4.

Secure execution

Through a mix of special software and hardware features, the devices ensure the correct operation of their functions against abnormal situations caused by programmer errors, software attacks through network access, or a local attempt to tamper code execution. Some of those features are listed below:

- The memory protection unit (MPU) of Cortex®-M55 can restrict the read and write accesses to memory regions (including regions mapped to peripherals), based on operating mode (privileged, unprivileged) or based on data/instruction fetch criteria.
- The independent watchdog can be used for secure software monitoring.

Resource isolation

Like the platform ROM code, the integrator can use resource isolation hardware features such as the Cortex® privilege mode and Arm® TrustZone® security extension, extended to allocated I/Os, DMA channels, memories, and peripherals.

For other controllers than the Cortex®-M55, the integrator can use the STM32 resource isolation framework (RIF) to allocate peripherals, fixed-sized embedded memory blocks, and variable-sized address space regions, under the control of the security firmware. Refer to [RM] Section 3.5, and Sections 6 to 8 for details.

Return material for analysis (field return)

The final product manufacturer must invoke this functionality before returning the TOE device for failure analysis to STMicroelectronics. It is defined in [RM] Section 4.3.7 *Opening BSEC*. The RMA lifecycle state is terminal.

The TOE is certified with a provisioned and locked 128-bit RMA password. The integrator has the privilege and responsibility to secure it following the recommendations described in Section 3.2: Secure installation and secure preparation (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 should configure anti-tamper methods available in the device to add a layer of security when managing the following functions:

- Nonvolatile fuse storage (including derived hardware unique key)
- Volatile secret key storage in backup domain registers
- Embedded AHBSRAM2 and BKPRAM storage
- Crypto functions in hardware engines (SAES, CRYP, HASH, PKA)

Those methods are described in [RM] Section 62 tamper and backup registers (TAMP).

When a tamper flag is raised, the software must implement a security response, for example, a platform reset.

Secure debug

Secure Debugging is not included as an SFR in the certification. In the certified Closed_Locked_Provd configuration, the platform ensures all debug features are disabled at cold boot. If *debug_lock* fuses in OTP word 18 are blown, the platform ROM code sets BSEC_DBGCR to 0, ensuring that the software cannot enable debug.

On parts where these fuses are not blown, software can choose to enable either full secure debug, or non-secure debug. It is not possible to enable debugging of the ROM code. Refer to the JTAG interface in Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) for details.

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:

- Physical chip interface
- BOOT pins
- BSEC interface
- Boot image interface
- True random number generation interface
- Platform identification
- Cryptographic functions interface
- JTAG interface
- RIF interface

This section highlights all security parameters under the control of the integrator, indicating the secure values as appropriate.

Physical chip interface

After providing the power supply and clocks and deasserting the reset signal, the platform starts on secure ROM code.

Optionally, the integrator can activate the tamper-detection and response hardware present on the physical chip interface.

Method of use:

- If cold boot, activate the power supplies and clocks of the platform.
- Reset the device (power-on-reset, global system reset).
- Secured Cortex®-M55 executes platform ROM code, triggering image download via the boot interface.

Parameters:

- Refer to [RM] Section 14 about reset pins.
- Refer to [RM] Section 62 about TAMP registers

Actions:

- Define the reset behavior of the platform using BOOT pins and OTP words (see next two subsections)
- Activate tamper detection method defined in [RM] Section 62.3.9 (not required for SESIP certification)
- Activate backup registers and other device secrets erase (refer to [RM] Section 62.3.10) (not required for SESIP certification)

Errors:

- The platform goes to a locked state if the authenticity or integrity of the platform cannot be ensured. A power-on-reset or global system reset is necessary to exit this locked state.

BOOT pins

After providing the power supply and clocks but before deasserting the reset signals, the user can instruct the platform how to manage the boot process via dedicated boot pins BOOT1 and BOOT0.

Method of use:

- If serial boot is needed, connect the USB Type-A host port of the TOE to the user's computer.
- Modify the electrical values on the boot pins.
- Activate the power supplies and clocks of the platform or wake up the platform.
- Secured Cortex®-M55 executes platform ROM code that consults the state of the boot pins before starting the secure initialization process.

Parameters:

- On a Closed_Unlocked device, if BOOT1 is set, then debug via SWD/JTAG is available, and the boot ROM sits in an endless loop.
- If BOOT1 is 0, or the part is Closed_Locked_Provd, then BOOT0 determines the boot source: if BOOT0 is 0, the ROM attempts to boot from the flash memory, otherwise if BOOT0 is 1, it attempts a serial boot.

Actions:

- Set electrical values to BOOT pins. Those are connected to user switches SW1 on the STM32N6570-DK evaluation kit.

Errors:

- Not applicable.

BSEC interface

After providing the power supply and clocks but before deasserting the reset signal, the user can instruct the platform how to manage the boot process using dedicated OTP words.

Other usage of the BSEC peripheral is up to the user of the platform.

Method of use:

- Boot:
  - If cold boot, activate the power supplies and clocks of the platform.
  - Reset the device. The platform ROM code starts.
  - The platform branches to authenticated code in SRAM after locking some of its assets in OTP.
- Trusted domain CPU with TDCID performs read and write access to BSEC registers.
- Via BSEC registers, the trusted domain CPU can modify the secure boot behavior of the platform by blowing fuses.

Parameters:

- Refer to [RM] Section 4.5 BSEC registers.
- Flash memory boot configuration in OTP word 11 (BOOTROM_CONFIG_2)

OTP field	Offset	Size (in bits)	Default	Description
flash_boot_source	[8:5]	4	0	[0-15]: Select flash boot source: 0x1: SD card SDMMC1 0x2: eMMC SDMMC1, 0x3: NOR, 0x5: HyperFlash™ 0x7: SD card SDMMC2 0x8: eMMC SDMMC2
boot_source_disable	[16:9]	8	0	Each bit disables a boot source. bit[0]: disable USB, bit[1]: disable UART, bit[3]: disable SPI, bit[4]: disable I^2C. Default to UART if all are disabled.
spi_instance_disable	[19:17]	3	0	bit[0]: disable SPI1 instance, bit[1]: disable SPI2 instance, bit[2]: disable SPI3 instance.
uart_instance_disable	[22:20]	3	0	bit[0]: disable USART1 instance, bit[1]: disable USART2 instance, bit[2]: disable USART3 instance.
i2c instance disable	[28:26]	3	0	bit[0]: disable I2C1 instance, bit[1]: disable I2C2 instance, bit[2]: disable I2C3 instance.

Actions:

- If the serial or flash boot configuration must be changed, write to OTP word 11 (BOOTROM_CONFIG_2).
- For other changes to on-chip nonvolatile memory, the CPU secure domain can read and write to BSEC registers (refer to [RM] Section 4).

Errors:

- Refer to [RM] Section 4.3.16 BSEC error management

Boot image interface

The platform uses the boot image stored in SYSRAM, after updating the RIF firewall configuration.

Method of use:

- Prepare the boot image in external flash (refer to Step 5: Image programming in Section 3.2.2), or ready the STM32 boot tool (refer to Software setup procedure in Section 3.2.1).
- Follow the method in the *Physical chip interface*, cold boot mode of operation.

Parameters:

- Boot image, defined in *Jump to cold boot image* of Section 4.2.1.

Actions:

- No direct action to boot image area in available SYSRAM.

Errors:

- If the boot image selected by the BOOT pins and OTP parameters fails, the platform goes to serial boot mode.

True random number generation interface

When the platform does not run, the application can use the RNG peripheral to generate true random numbers.

Method of use:

- The firmware interacts with the hardware TRNG through a bank of memory-mapped registers.

Parameters:

- Refer to [RM] Section 47.7 *RNG registers*.

Actions:

- Refer to [RM] Section 47.3.4 *RNG initialization*.
- Refer to [RM] Section 47.3.5 *RNG operation*.
- Refer to [RM] Section 47.3.8 *RNG low-power use*.
- Refer to [RM] Section 47.4 *RNG interrupts*.

Errors:

- Refer to [RM] Section 47.3.7 *Error management*.

Platform identification

The platform provides a unique identification of the platform, including all its parts and their versions.

Method of use:

- Read the boot ROM ChipVersion, CutVersion, MaskVersion, and ROM identifier registers (`0x18001000`, `0x18001004`, `0x18001008`, and `0x1800100C` respectively for secure domain, `0x08001000`, `0x08001004`, `0x08001008`, and `0x0800100C` respectively for nonsecure domain).

Parameters:

- The boot ROM ChipVersion, CutVersion, MaskVersion, and ROM identifier registers (`0x18001000`, `0x18001004`, `0x18001008`, and `0x1800100C` respectively for secure domain, `0x08001000`, `0x08001004`, `0x08001008`, and `0x0800100C` respectively for nonsecure domain).

Actions:

- Verify that the value read back for the ChipVersion is `0x0000 8604`, the value read back for the CutVersion is `0x0000 0200`, the value read back for the MaskVersion is `0x0000 0001`, and the value read back for the ROM identifier is `0x0000 0501`.

Errors:

- Not applicable

Cryptographic functions interface

When the platform is not executing, the platform provides the application with a set of hardware IP registers to access cryptographic operations and cryptographic key store resources. Some of those cryptographic operations are protected against side-channel attacks (AES, modular exponentiation, signature, ECC scalar multiplication).

Method of use:

- The firmware interacts with the hardware SAES and PKA through a bank of memory-mapped registers, and the embedded RAM in the peripheral (for the PKA).

Parameters:

- Refer to [RM] Section 48.8 *SAES registers*.
- Refer to [RM] Section 52.7 *PKA registers*, and Section 52.4 (for PKA RAM parameters).

Actions:

- Refer to [RM] SAES sections 48.4, 48.5 and 48.6.
- Refer to [RM] PKA sections 52.3 and 52.6.

Errors:

- Refer to [RM] Section 48.4.19 *SAES error management*.
- Refer to [RM] Section 52.3.7 *PKA error management*.

JTAG interface

The standard JTAG with SWD interface allows debugging of the application. It is used according to [IEEE1149]. When the device is in the Closed_Locked_Provd state, all debug features are disabled after a cold reset. The JTAG/SWD interface remains enabled on reset only to inject the RMA password to request the transition to “RMA” state.

The integrator can use the BSEC peripheral to enable debugging of its code or prevent such debugging until the next system reset.

The platform ROM code can never be debugged or traced using the JTAG interface.

Method of use:

- JTAG physical link to transport RMA password value. The method is defined in [RM] Section 4.3.7 Opening BSEC.
- Following platform secure boot, use BSEC_DBGCR and BSEC_AP_UNLOCK registers to allow debugging of the integrator application. JTAG/SWD can then be used with a debugger.

Parameters:

- OTP_RMA_LOCK_PSWD, stored in OTP words 256 to 259 (refer to [RM] Section 7).
- BSEC_DBGCR and BSEC_AP_UNLOCK register (refer to [RM] Section 4.3.11 for details).
- *debug_lock* fuses in OTP fuse word 18. Burn any bits 22 to 25 in OTP18 (BOOTROM_CONFIG_9) to ensure that writes to the BSEC_DBGCR register are always ignored.

Actions:

- Inject the RMA password through JTAG/SWD under reset. If the received value is identical to the programmed OTP_RMA_LOCK_PSWD value, the hardware starts a transition to the RMA state.
- Connect a debugger for a debug session.

Errors:

- When injecting the RMA passwords:
  - The device state does not change following a reset when the provided RMA password is wrong.
  - After more than four incorrect RMA attempts, the device state cannot be changed.
- When connecting a debugger:
  - The debug session fails if the BSEC_DBGCR or BSEC_AP_UNLOCK settings are not correct.

RIF interface

The RIF interface can be used to partition the resources of the system between the secure and nonsecure domains, using a system of firewalling.

The RIFSC block allows the integrator to configure firewalls protecting peripheral access, and determine the security and privilege levels of accesses made by peripheral bus controllers.

RISAF blocks are placed in front of memory targets and allow the integrator to define a partitioning of the associated memory target.

The IAC collects error signals from the firewalls and allows the software to determine which firewall is reporting an illegal access.

The authenticated firmware starts in secure privileged mode, and by default can access all resources without any RIF configuration required. Secure privileged software can configure the firewalls. It is up to the integrator to define the firewalling regime.

The certified configuration assumes that RIF is in its default reset state.

Method of use:

- The integrator can configure firewalling using the RIFSC, RISAF, and IAC registers.

Parameters:

- Refer to [RM] Section 6.4 *RIFSC registers*.
- Refer to [RM] Section 7.5 *RISAF registers.*
- Refer to [RM] Section 8.6 *IAC registers.*
-

Actions:

- Refer to [RM] Section 6.3 *RIFSC functional description*.
- Refer to [RM] Section 7.4 *RISAF functional description*
- Refer to [RM] Section 8.4 *IAC functional description*

Errors:

- Refer to [RM] Section 8.5 *IAC interrupts*.

4.2.3 Security-relevant events (AGD_OPE.1.4C)

Once configured according to Section 3.2: Secure installation and secure preparation (AGD_PRE.1.2C), the platform triggers the following security-relevant events when the integrator uses the functions and privileges described in Section 4.2.1.

- Images authenticity or integrity violation:
  - An incorrect signature verification ends with a boot error (red LED). It can be due to an incorrect key pair usage or a corrupted memory. The second image (backup) is loaded to check if it can boot, otherwise the boot process is stalled.
- Images decryption error:
  - An incorrect decryption is detected when the hash of the plain text does not match the header one. It ends in a boot error (red LED). The second image (backup) is loaded to check if it can boot, otherwise the boot process is stalled.
- Unexpected boot image:
  - Wrong header, nonconsistent header (for example, image size not consistent with header), illegal parameter in header.
  - Image stored at the wrong address.
  - All these unrecoverable issues end in a backup image boot or a ROM boot failure.
- Revoked boot image:
  - The signed key in the boot image is revoked in fuses.
  - The image version number is lower than the one stored in fuses.
  - All these unrecoverable issues end in a backup image boot or a ROM boot failure.
- Erroneous values found in the OTP of the security configuration at boot time:
  - Abort of the boot sequence.
- Closed JTAG/SWD access violation:
  - The connection request is not transmitted to the access port and debug port. The request is ignored.
- Erroneous RMA password injection:
  - The platform ignores the RMA request and restarts in its current life cycle state at the next boot.
- Detection of attacks to RNG, SAES, and PKA by an attacker with physical access
  - Peripheral goes to locked mode + generation of internal tamper (tamp_itamp9 input), disabled by default.
- Error when reading a fuse word
  - Refer to [RM] Section 4.3.6 BSEC read and programming status reporting.
- Illegal access to a resource protected by RIF
  - Refer to [RM] Section 7.4.7 RIF illegal access definition.
- Attempts to compromise RNG entropy properties by disturbing physical RNG interfaces:
  - The RNG hardware raises alarms on clock and noise source defects with a possible interrupt by a user implementation-defined error handler.

4.2.4 Security measures (AGD_OPE.1.6C)

This section describes the security measures for the integrator user role to be followed to fulfill the security objectives for the operational environment as described in the [ST] (Section 2.1).

- The operating system or application code is expected to verify the correct version of all platform components that it depends on.
- The operating system or application code must use the secure boot feature.

The integrator user role must take the following measures:

- Verify the genuineness of the TOE as described in Section 3.1: Secure acceptance.
- Follow all guidelines described and referenced in Section 3.2: Secure installation and secure preparation (AGD_PRE.1.2C).
- Follow all guidelines described in Section 4.2.1: User-accessible functions and privileges (AGD_OPE.1.1C) and Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) to integrate the TOE into a full IoT solution.
- In the manufacturing phase, the integrator must securely provision the TOE immutable data specific to the integrator or specific to the product as stated in Step 2: Provisioning of secrets
- Once the integrator finishes the production of a final user application, he must set the STM32N6xx device hardware static protections as stated in Section 3.2.2: Secure installation and Section 3.2.3: Certified configuration.

4.2.5 Modes of operation (AGD_OPE.1.5C)

This section identifies all possible modes of operation of the platform (including operation following failure or operational error), their consequences, and implications for maintaining secure operation.

[Boot]

The integrator selects the boot device from where the ROM code loads the image. After resetting, the platform requests the nonsecure ROM to load the boot image in embedded SRAM, then branches the boot CPU to execute this image if it is authentic.

In case of an error detected during the image authentication/decryption or validation (header parameters), the platform aborts the boot process and stays in a never-ending loop.

[RMA password injection]

The integrator owning the RMA password value injects it over the JTAG/SWD interface while maintaining the platform under reset as explained in subsection “Opening BSEC” of [RM] Section 4.3.7. If the RMA password value is wrong, the platform ignores the RMA request and waits for a reboot.

5 Annexes

5.1 Nonsecure ROM bootloader interfaces

When allowed by secure boot ROM the nonsecure ROM bootloader of the device supports the following flash memory interfaces:

- Serial NOR flash via Quad-SPI peripheral
- SPI HyperFlash™
- SD card via SDMMC peripheral
- eMMC via SDMMC peripheral

Depending on the case, the storage of the boot ROM image must follow the rules defined in the [WIK] product wiki page.

Revision history

Table 7. **Document revision history**

Date	Revision	Changes
28-Feb-2025	1	Initial release.

Contents

List of tables

List of figures