AN5056
Application note

How to integrate the X-CUBE-SBSFU STM32Cube Expansion Package

Introduction

The X-CUBE-SBSFU secure boot and secure firmware update solution allows the update of the STM32 microcontroller built-in program with new firmware versions, adding new features and correcting potential issues. The update process is performed in a secure way to prevent unauthorized updates and access to confidential on-device data.

The secure boot (Root of Trust services) is an immutable code, always executed after a system reset. It checks STM32 static protections, activates STM32 runtime protections, and then verifies the authenticity and integrity of user application code before every execution to make sure that invalid or malicious code cannot be run.

The secure firmware update application receives the firmware image via a UART interface with the YMODEM protocol. It checks its authenticity, and the integrity of the code before installing it. The firmware update is done on the complete firmware image, or only on a portion of the firmware image. Examples can be configured to use asymmetric or symmetric cryptographic schemes with or without firmware encryption. They are provided:

- For single-slot configuration to maximize firmware image size
- For dual-slot configuration to ensure safe image installation and enable over-the-air firmware update capability commonly used in IoT devices.

For a complex system with multiple firmware such as protocol stack, middleware, and user application, the firmware image configuration can be extended up to three firmware images.

The secure key management services provide cryptographic services to the user application through the PKCS #11 APIs (KEY ID-based APIs) that are executed inside a protected and isolated environment. User application keys are stored in the protected and isolated environment for their secured update: authenticity check, data decryption, and data integrity check.

STSAFE-A110 is a tamper-resistant secure element (Hardware Common Criteria EAL5+ certified) used to host X509 certificates and keys and perform verifications used for firmware image authentication during secure boot and secure firmware update procedures.

The X-CUBE-SBSFU user manual (UM2262) explains how to get started with X-CUBE-SBSFU and details SBSFU functionalities. This application note describes how to adapt X-CUBE-SBSFU and integrate it with the user's application; It answers such questions as:

- How to port X-CUBE-SBSFU onto another board?
- How to tune the X-CUBE-SBSFU configuration to fit the user's needs?
- How to generate a new firmware encryption key?
- How to debug X-CUBE-SBSFU?
- How to adapt to SBSFU?
- How to adapt the user's application?

Note: Throughout this application note, the IAR Embedded Workbench® IDE is used as an example to provide guidelines for project configuration. secure boot and secure firmware update applications are referred to as SBSFU.

Note: The single-slot configuration is demonstrated in examples named 1_Image.
The dual-slot configuration is demonstrated in examples named 2_Images.

Contents

List of tables

List of figures

1 General information

Table 1 and *Table 2* present the definitions of acronyms and terms that are relevant for a better understanding of this document.

Table 1. List of acronyms

Acronym	Description
AES	Advanced encryption standard
DAP	Debug access port
ECDSA	Elliptic curve digital signature algorithm
GCM	AES Galois/counter mode
HAL	Hardware abstraction layer
IDE	Integrated development environment
FWALL	Firewall
MPU	Memory protection unit
OTFDEC	On-the-fly decryption
PEM	Privacy enhanced mail
PCROP	Proprietary code readout protection
RDP	Readout device protection
SB	Secure boot
SE	Secure Engine
SFU	Secure firmware update
SBSFU	Secure boot and secure firmware update
UART	Universal asynchronous receiver/transmitter
WRP	Write protection

Table 2. List of terms

Term	Description
Firmware image	An executable binary image run by the device as a user application.
Firmware header	Bundle of meta-data describing the firmware image to be installed. It contains firmware information and cryptographic information.
mbedTLS	mbed implementation of the TLS and SSL protocols and the respective cryptographic algorithms.
sfb file	Binary file packing the firmware header and the firmware image.

The X-CUBE-SBSFU secure boot and secure firmware update Expansion Package runs on STM32 32-bit microcontrollers based on the Arm® Cortex®-M processor.

arm

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.

2 Related documents

1. User manual *Getting started with STM32CubeH7 for STM32H7 series* MCUs (UM2204)[a]
2. User manual *Getting started with STM32CubeG4 for STM32G4 series* (UM2492)
3. User manual *Getting started with STM32CubeL0 for STM32L0 Series* (UM1754)
4. User manual *Getting started with STM32CubeL1 MCU Package for STM32L1 Series* (UM1802)
5. User manual *Getting started with STM32CubeWB for STM32WB Series* (UM2550)
6. User manual *Getting started with STM32CubeL4 MCU Package for STM32L4 series and STM32L4+ series* (UM1860)
7. User manual *Getting started with STM32CubeF4 for STM32F4 series* MCUs (UM1730)
8. User manual *Getting started with STM32CubeF7 MCU Package for STM32F7 Series* (UM1891)
9. User manual *Getting started with STM32CubeG0 for STM32G0 Series* (UM2303)
10. User manual *Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package* (UM2262)
11. User manual *Development guidelines for STM32Cube Expansion Packages* (UM2285)
12. User manual *Development checklist for STM32Cube Expansion Packages* (UM2312)
13. User manual *STM32CubeProgrammer software description* (UM2237)
14. *STM32 Cortex®-M4 MCUs and MPUs programming manual* (PM0214)
15. *STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual* (PM0253)
16. *STM32 Cortex®-M0+ MCUs programming manual* (PM0223)
17. Datasheet for STSAFE-A110 *Authentication, state-of-the-art security for peripherals and IoT devices* (DS12911)

a. STM32CubeH7 applies to the STM32H7 series but the STM32H7Rx/Sx MCUs. Similarly, the X-CUBE-SBSFU application note (AN5056) and user manual (UM2262) do not apply to the STM32H7Rx/Sx MCUs.

3 Porting X-CUBE-SBSFU onto another board

X-CUBE-SBSFU supplements the STM32Cube software technology, making portability across different STM32 microcontrollers easy. It comes with a set of examples implemented on given STM32 boards that are useful starting points to port the X-CUBE-SBSFU onto another STM32 board. The NUCLEO-L476RG and NUCLEO-L432KC boards are used as examples in this document.

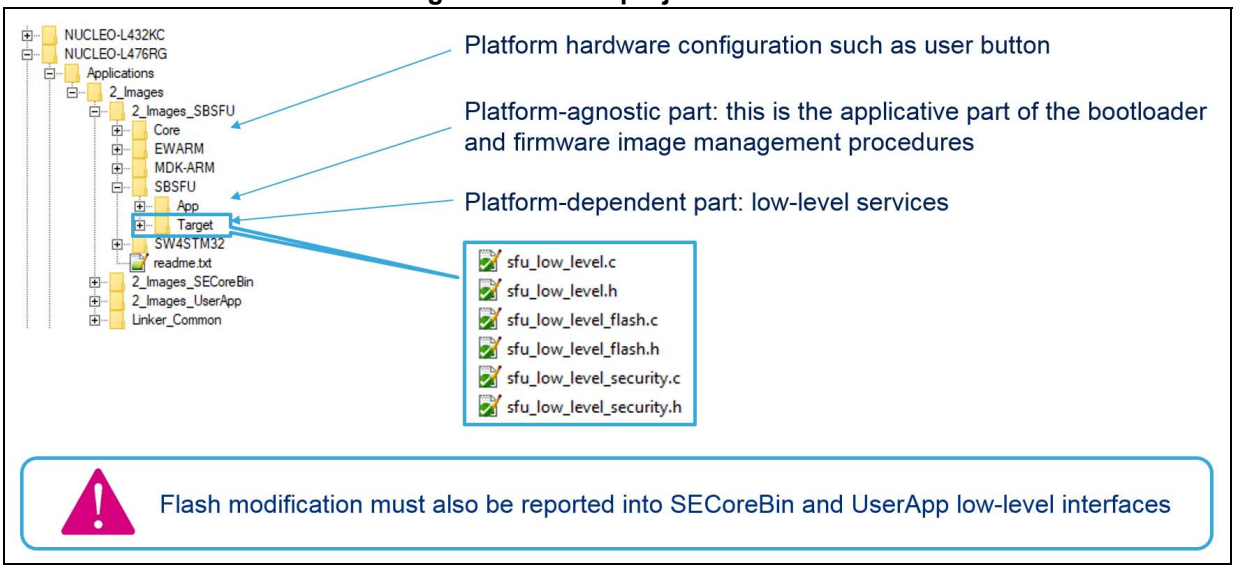
3.1 Hardware adaptation

A few changes are needed to adapt X-CUBE-SBSFU to another board:

1. GPIO configuration for UART communication with the host PC (In *sfu_low_level.h* file)
2. Flash memory configuration: NUCLEO-L432KC gives an example of a single-bank flash memory interface whereas NUCLEO-L476RG is dual-bank based (in *sfu_low_level.c* file)
3. Button configuration: NUCLEO-L476RG gives an example based on the push button whereas NUCLEO-L432KC simulates a virtual button with a GPIO (in *app_hw.h* file)
4. Tamper GPIO pin configuration (in *sfu_low_level_security.h* file)
5. DAP - Debug port configuration (in *sfu_low_level_security.h* file)
6. I^2C bus configuration for communication with STSAFE-A110 (in *stsafea_service_interface.c* file of *B-L4S5I-IOT01A\Applications\2_Images_STSAFE\2_Images_SECoreBin*).

Figure 1 presents the SBSFU project structure together with the location of the files where porting changes are expected.

Figure 1. SBSFU project structure



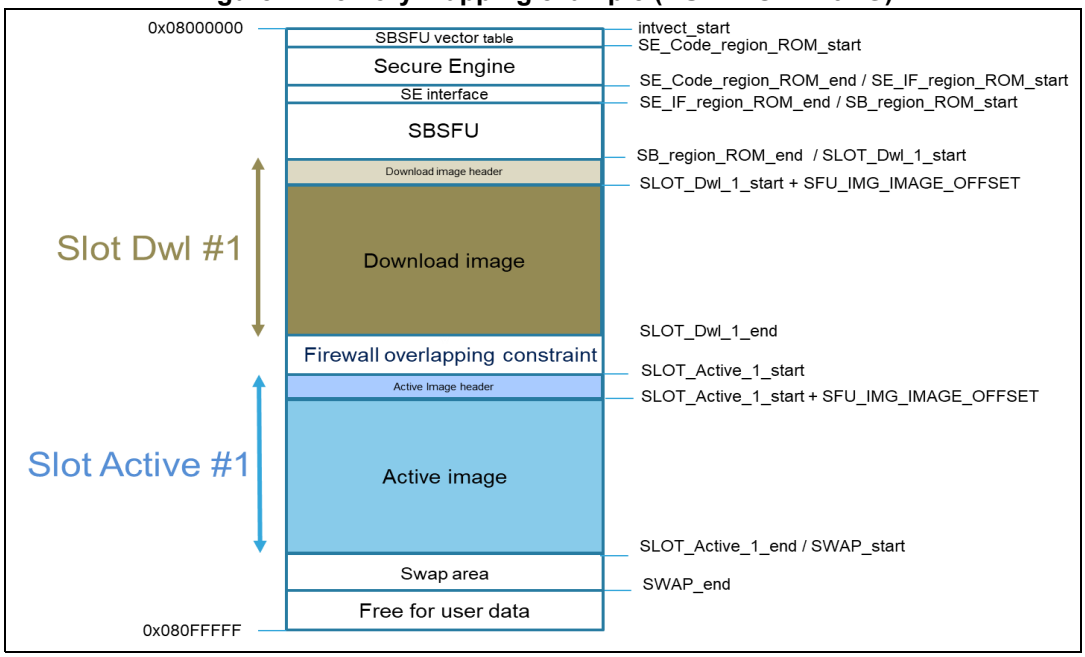
3.2 Memory mapping definition

As already highlighted in the X-CUBE-SBSFU user manual (refer to *[10]*), a key aspect is the placement of all elements inside the flash memory of the device:

- Secure Engine: protected environment to manage all critical data and operations.
- SBSFU: secure boot and secure firmware update
- Active slot: this slot contains active firmware (firmware header with firmware)
- Download slot: this slot stores downloaded firmware (firmware header with encrypted firmware) to be installed at the next reboot
- Swap area: Flash memory area used to swap the content of active and download slots during the installation process

Figure 2 presents the flash memory mapping illustrated by the NUCLEO-L476RG example.

Figure 2. Memory mapping example (NUCLEO-L476RG)

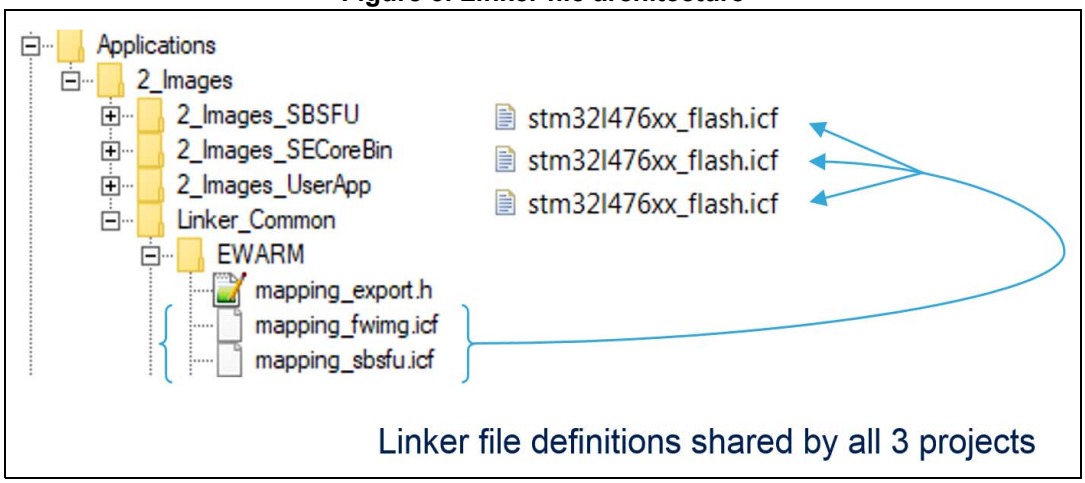


The linker file definitions shared between the three projects (SECoreBin, SBSFU, UserApp) are grouped in the *Linker_Common* folder as presented in *Figure 3*:

- *mapping_fwimg.icf*: contains firmware image definitions such as active slots, download slots, and swap area
- *mapping_sbsfu.icf*: contains SBSFU definitions such as `SE_Code_region`, `SE_Key_region`, and `SE_IF_region`
- *mapping_export.h*: export the symbols from *mapping_sbsfu.icf* and *mapping_fwimg.icf* to the SBSFU applications

Each region can be extended when adding more code is needed or shifted to another address as long as the resulting security settings satisfy security requirements.

Figure 3. Linker file architecture

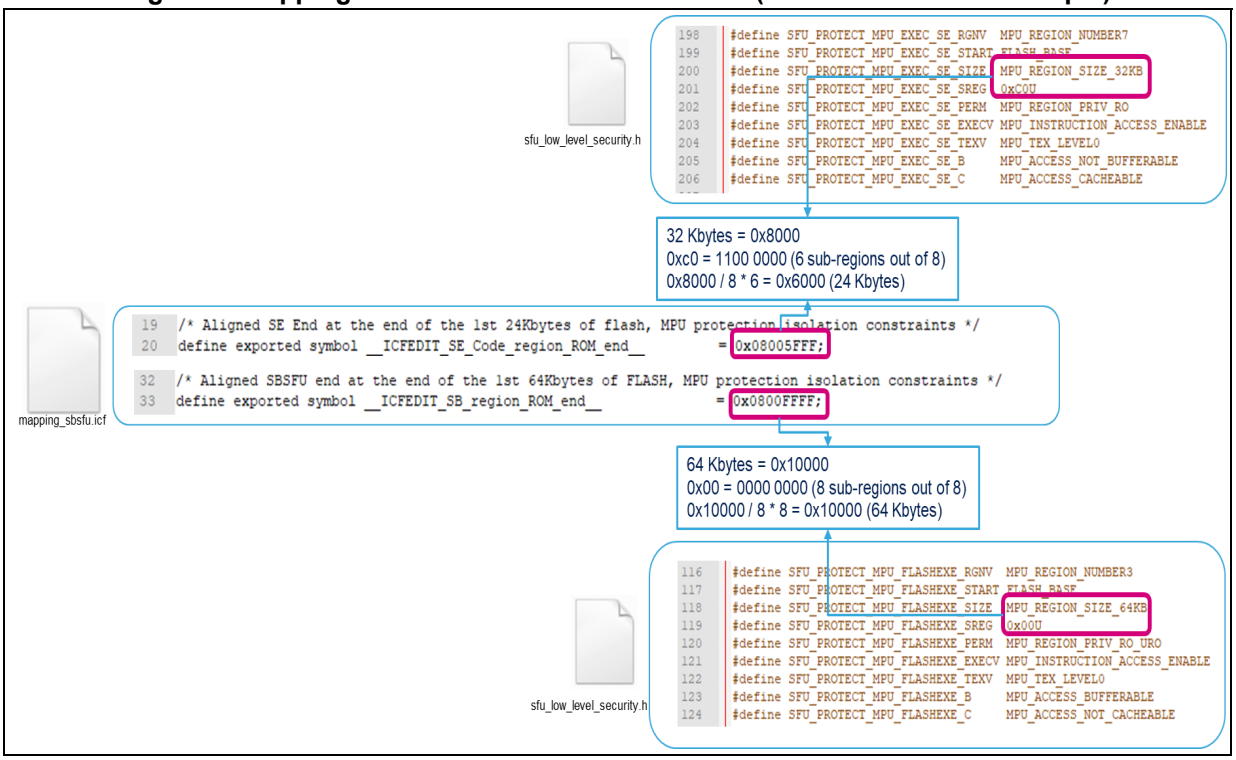


The security peripheral configuration (RDP, WRP, PCROP, FWALL, secure user memory if available for the series) is automatically computed based on the SBSFU linker symbols except for MPU configuration due to the following constraints:

- each MPU region base address must be a multiple of the MPU region size.
- each MPU region can be divided into 8 sub-regions to adjust the size.

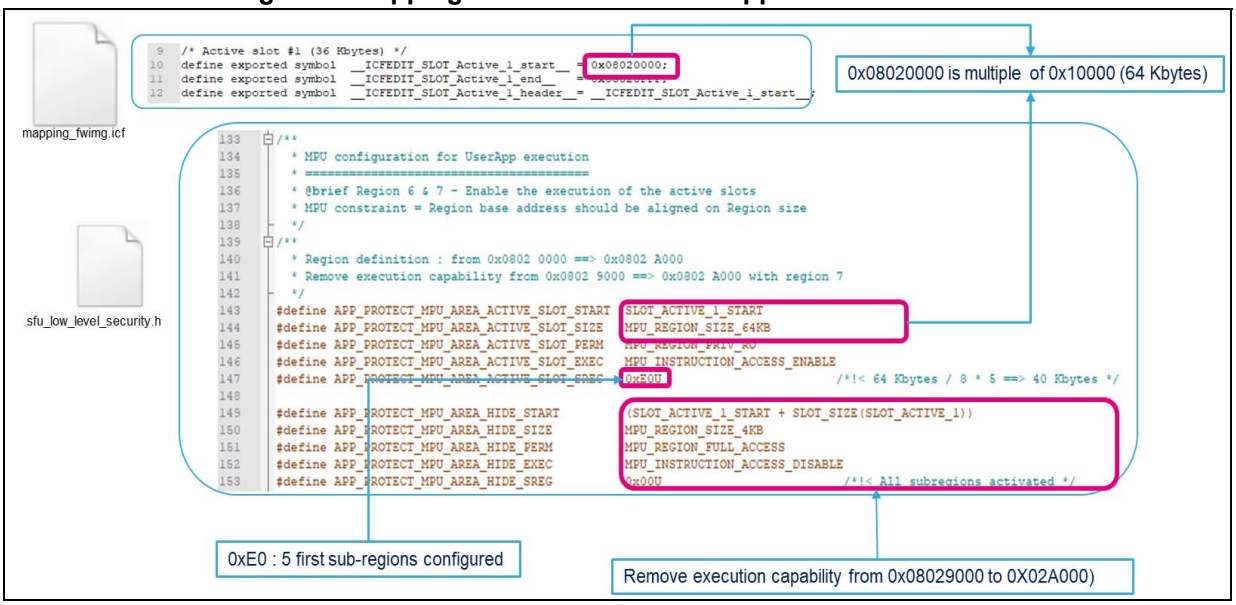
The mapping constraints with MPU isolation are illustrated in *Figure 4*.

Figure 4. Mapping constraints with MPU isolation (NUCLEO-G071RB example)



Another typical use case is the MPU configuration of the active-slot region to authorize user application execution. *Figure 5* shows how to respect the MPU constraints on NUCLEO-L073RZ.

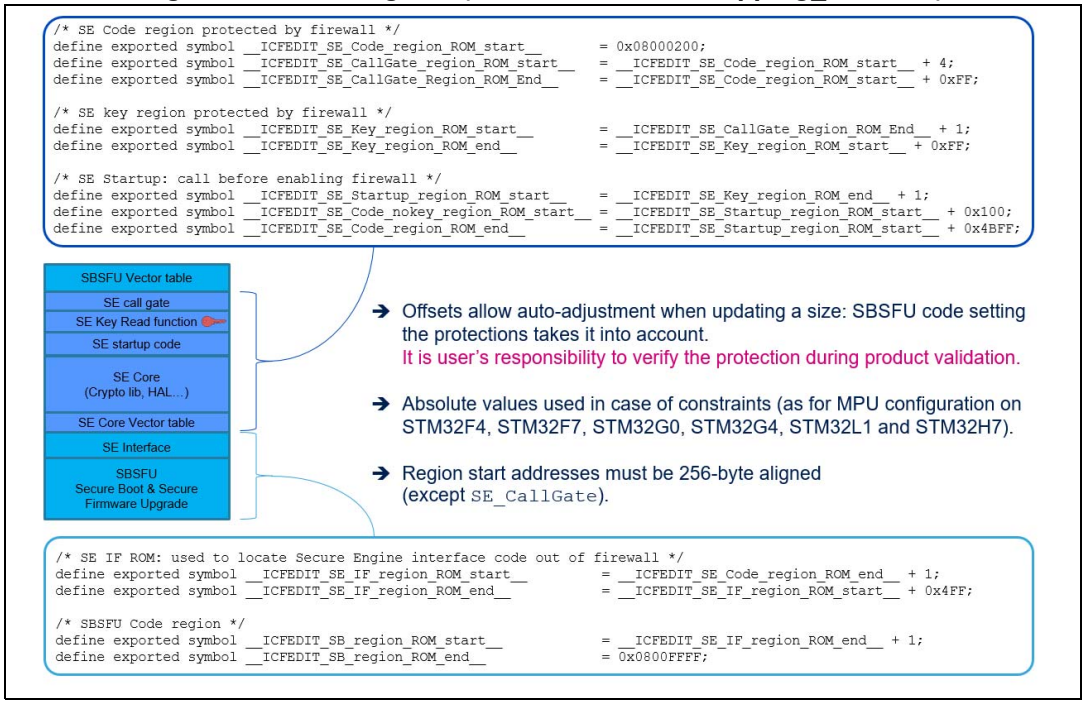
Figure 5. Mapping constraints for user application execution



3.2.1 SBSFU region definition parameters

Figure 6 presents the parameters in file *mapping_sbsfu.icf* that are used for the configuration of the SBSFU regions.

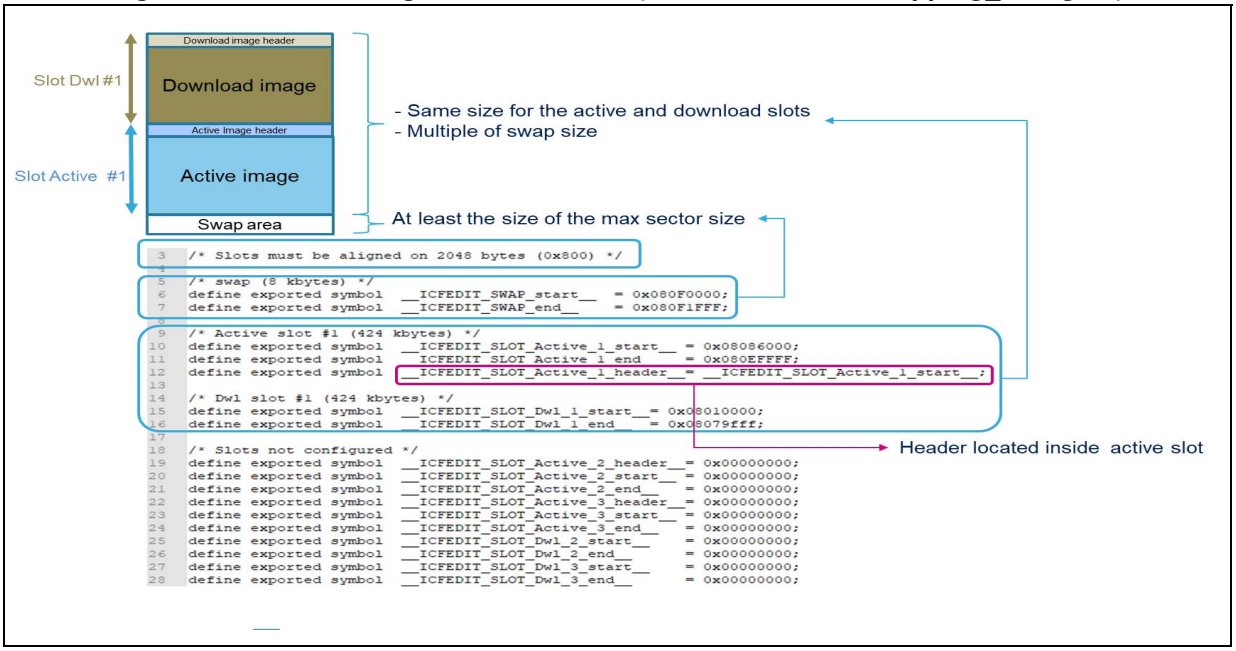
Figure 6. SBSFU regions (NUCLEO-L476RG *mapping_sbsfu.icf*)



3.2.2 Firmware image slot definition parameters

Figure 7 presents the parameters in file *mapping_fwimg.icf* that are used for the configuration of the image regions.

Figure 7. Firmware image slot definitions (NUCLEO-L476RG *mapping_fwimg.icf*)



Compliance with SBSFU constraints requires that the following conditions are met:

- Slots areas must be aligned on the flash memory sector size, which is 2048 bytes (0x800) for devices in the STM32L4 series.
- The minimum size of `SWAP` is 4 Kbytes and at least equal to the size of the largest sector.
- The size of active and download slots must be multiple of the `SWAP` size.
- The sizes of active and download slots must be equal, except when using the partial update feature.

In some configurations (external flash memory with OTFDEC, multiple image configuration) the header must be located outside the active slot in its own flash memory sector to remain protected inside the isolated environment.

For STM32L4 dual-bank flash memory devices, firewall specific constraints are:

- Firewall code segment must be in bank1, firewall non-volatile data (including the header of the active slot) segment must be in bank2.
- The non-volatile data segment must overlap the firewall code segment to ensure that secrets are always protected even if the banks are swapped.
  Figure 8: Firewall configuration constraint on dual bank products and *Figure 9: Firewall configuration after bank swap* illustrate the firewall configuration on the NUCLEO-L476RG and the consequences when banks are swapped.

Figure 8. Firewall configuration constraint on dual bank products

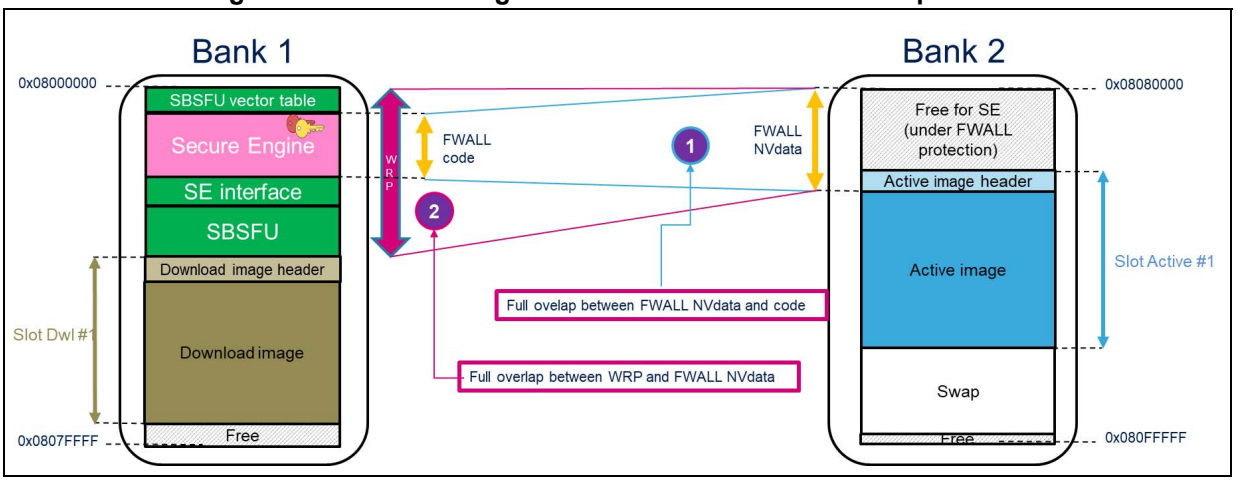
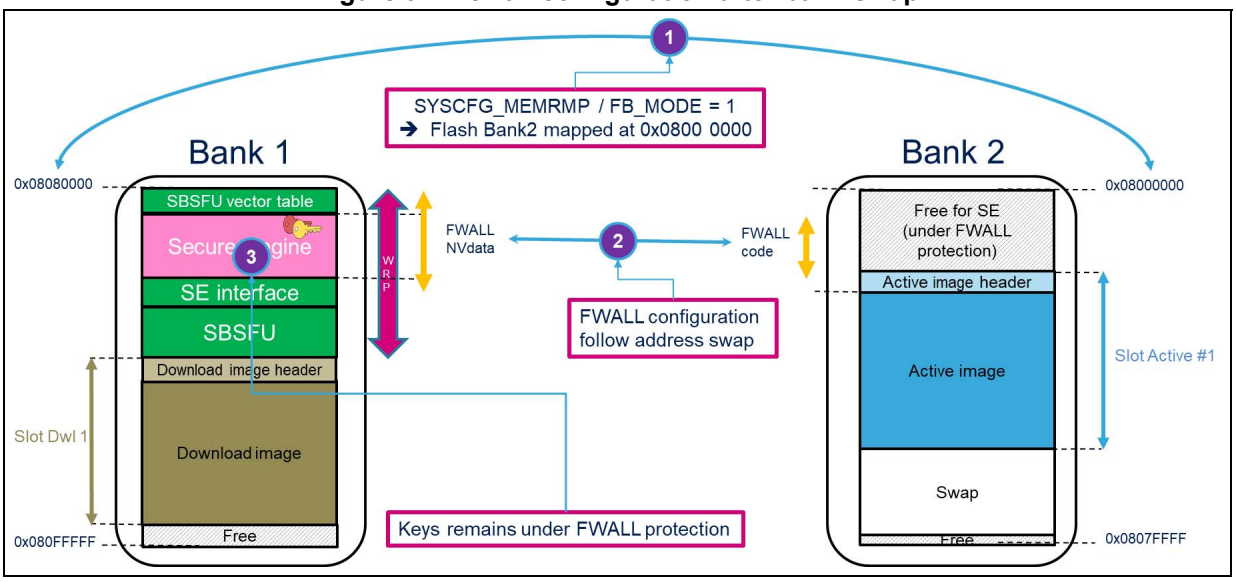


Figure 9. Firewall configuration after bank swap



For the STM32G0 series, STM32G4 series, and STM32H7 series, one constraint exists: the header of the active slot must be mapped just after the SBSFU code to be protected by the secured memory.

The SFU_IMAGE_OFFSET value depends on the STM32 microcontroller series:

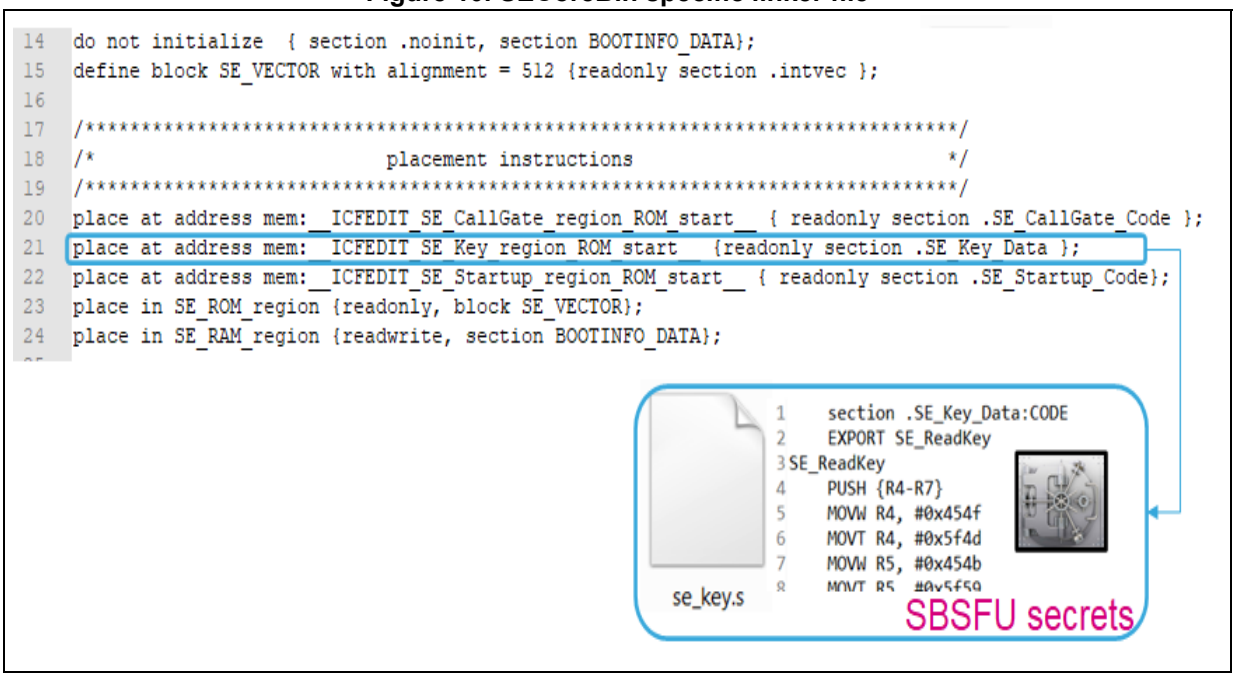
- For the STM32L4 series, STM32L0 series, STM32L1 series, STM32WB series, and STM32F4 series, the default value is used: 512 bytes.
- For the STM32F7 series and STM32H7 series: 1024 bytes.
  With the Cortex®-M7, the vector table must be aligned on 1024 bytes.
- For the STM32G0 series: 2048 bytes.
  The secure user memory end address is aligned on the flash memory sector size.
- For the STM32G4 series: 4096 bytes.
  The secure user memory end address is aligned on the flash memory sector size.
- For the STSAFE-A variant: 2048 bytes.
  The image header has a 2048-byte length to include X509 certificates.

Note: For series with MPU-based isolation or firewall-based isolation, the MPU constraint on the active-slot configuration must be verified as illustrated in Figure 5.

3.2.3 Project-specific linker files

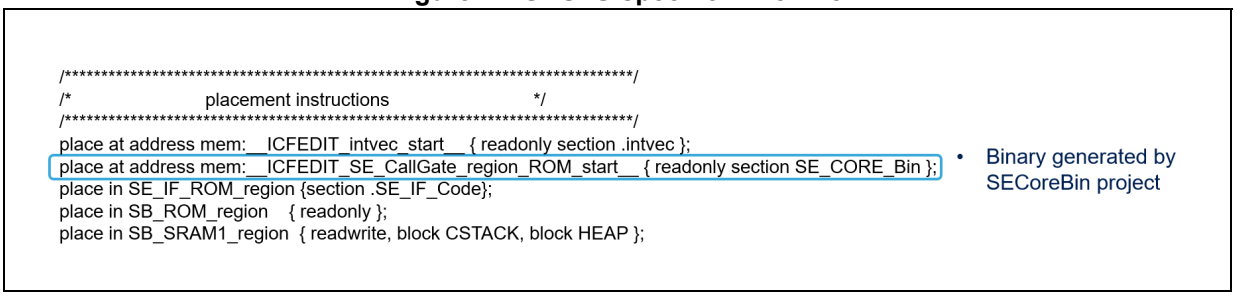
SECoreBin places critical code and data such as secrets, as illustrated in *Figure 10*.

Figure 10. SECoreBin specific linker file



The SBSFU linker file is in charge of SBSFU application placement that includes SECoreBin binary as shown in *Figure 11*.

Figure 11. SBSFU specific linker file



UserApp must be configured to run in the active slot (slot active start address with SFU_IMG_IMAGE_OFFSET) as illustrated in *Figure 12* where SFU_IMG_IMAGE_OFFSET is 512 bytes for the STM32L4 series.

Figure 12. UserApp specific linker file (NUCLEO-L476RG example)

```
/*-Specials-*/
define exported symbol __ICFEDIT_intvec_start__ = __ICFEDIT_SLOT_Active_1_start__ + 512;

/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__      = __ICFEDIT_intvec_start__;
define symbol __ICFEDIT_region_ROM_end__        = __ICFEDIT_SLOT_Active_1_end__ ;

define symbol __ICFEDIT_region_RAM_start__      = __ICFEDIT_SE_region_RAM_end__ + 1;
define symbol __ICFEDIT_region_RAM_end__        = 0x20017FFF;

/* to make sure the binary size is a multiple of the AES block size (16 bytes) and L4 flash
   writing unit (8 bytes) */
define root section aes_block_padding with alignment=16
{
udata8 "Force Alignment";
pad_to 16;
};

place in ROM_region   { readonly, last section aes_block_padding };
```

- UserApp must be configured to run from active slot start address + SFU_IMG_OFFSET (512 for STM32L4 Series)
- Protected RAM (FWALL, MPU(1)) used by SE cannot be re-used
- Firmware size should be multiple of AES block size and flash writing unit

1. Depends on the STM32 microcontroller series.

3.2.4 Multiple image configuration

Up to three active slots (*SFU_NB_MAX_ACTIVE_IMAGE*) and three download slots (*SFU_NB_MAX_DWL_AREA*) can be configured.

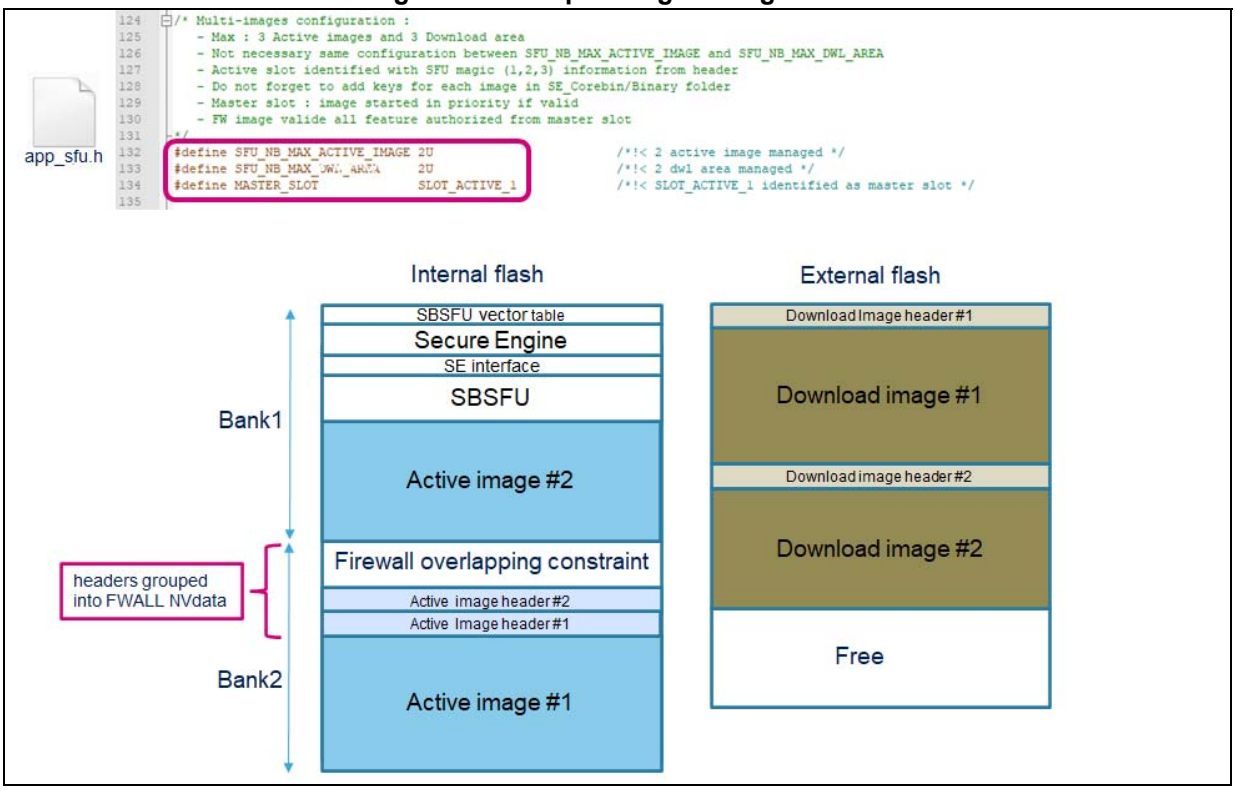
During the installation process, the active slot is identified with the SFU magic tag inside the firmware image header (SFU1, SFU2, or SFU3). Depending on firmware compatibility constraints, if the simultaneous firmware installation is not required, a single download slot can be configured for the three active slots to optimize the memory footprint.

At boot, after verification of the authenticity and integrity of all firmware images, SBSFU jumps into the active firmware image located inside the *MASTER_SLOT* in priority.

As a constraint, all the headers must be grouped in a single area to be protected inside the isolated environment. Each header must be located in its own flash memory memory sector.

Figure 13 shows the example of the multiple-image configuration provided in 2_Images_ExtFlash of the B-L475E-IOT01A board.

Figure 13. Multiple image configuration



3.3 Dual-core adaptation

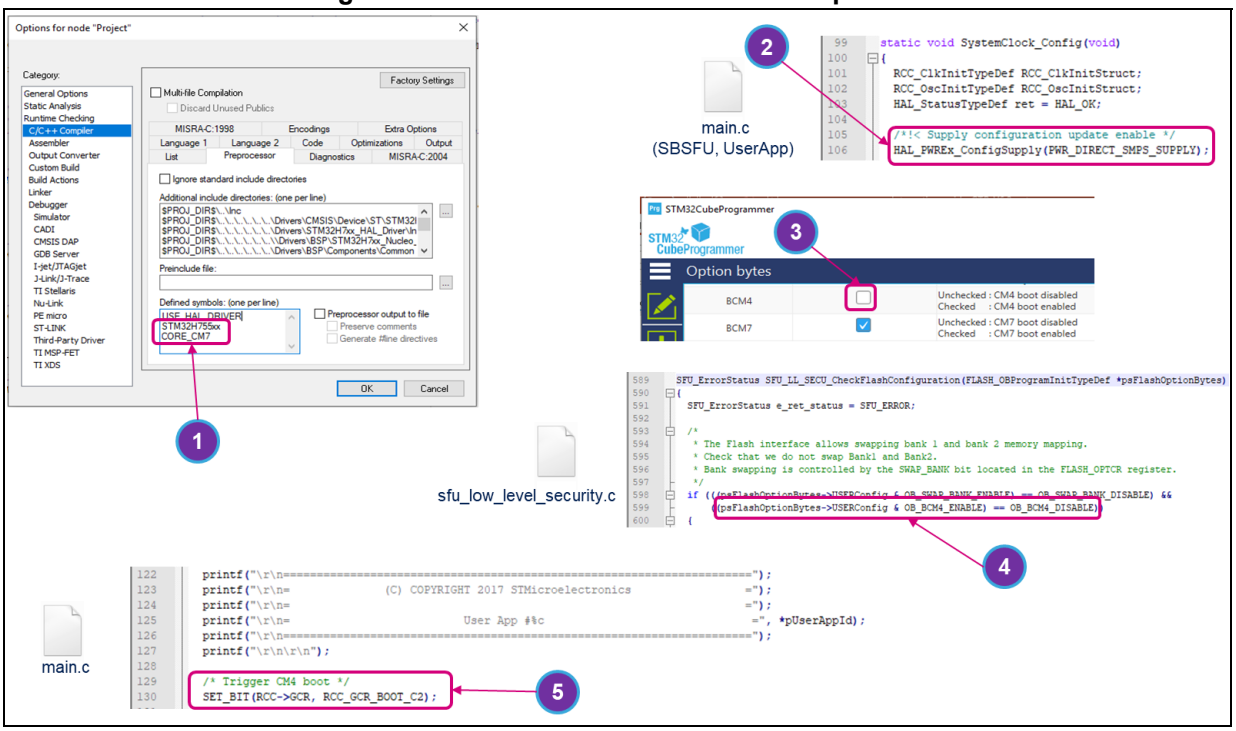
For the STM32H7 series dual-core products, it is mandatory to disable the CM4 boot while the SBSFU is running (on CM7).

Thus, once the authentication and the integrity of all firmware images are verified by the SBSFU, the user application starting on CM7 can trigger the boot of CM4.

As an example, to port applications provided for NUCLEO-H753ZI on NUCLEO-H755ZI-Q, the following modifications are needed as shown in *Figure 14*:

1. Modify the IDE configuration by adding STM32H755xx and CORE_CM7 defined symbols.
2. Change the supply configuration from LDO to SMPS in the `SystemClock_Config()` function.
3. Disable the Cortex M4 boot: BCM4 bit from option byte must be unchecked.
4. Add in the `SFU_LL_SECU_CheckFlashConfiguration()` function the control of the BCM4 bit state.
5. Add in the UserApp user application project, the trigger of CM4 boot.

Figure 14. STM32H7 series dual-core adaptation



Slots configuration may be adapted to manage two firmware images, one dedicated to CM7 and the other one dedicated to CM4. Refer to 3.2.4 Multiple image configuration for more details.

4 SBSFU configuration

4.1 Features to be configured

X-CUBE-SBSFU supports:

- 2 modes of operation: dual and single slot configurations
- 3 cryptographic schemes using symmetric and asymmetric cryptographic operations
- 2 cryptographic middleware:
  - STMicroelectronics middleware: X-CUBE-CRYPTOLIB library integrated into the *1_Image* and *2_Images* variants.
  - Third-party middleware: mbedTLS (open-source code) cryptographic services. Examples are provided for the 32L496GDISCOVERY, B-L475E-IOT01A, 32F413HDISCOVERY, 32F769IDISCOVERY, NUCLEO-WB55RG[a], and NUCLEO-H753ZI Nucleo boards in the *2_Images_OSC* variant.
- STSAFE-A110 secure element used to host X509 certificates and keys. An example is provided for the B-L4S5I-IOT01A board in the *2_Images_STSAFE* variant.
- KMS middleware. An example is provided for the B-L475E-IOT01A and B-L4S5I-IOT01A boards in the *2_Images_KMS* variant.
- External flash memory with on-the-fly decryption (OTFDEC). An example is provided for the STM32H7B3I-DK board in the *2_Images_ExtFlash* variant using a specific cryptographic scheme with AES-CTR firmware encryption.
- External flash memory without on-the-fly decryption (OTFDEC). An example is provided for the STM32H750B-DK board in the *2_Images_ExtFlash* variant. Active slot, as well as download slot, are mapped in an external flash memory, thus firmware confidentiality cannot be ensured.
- External flash memory without on-the-fly decryption (OTFDEC). An example is provided for the B-L475E-IOT01A board in the *2_Images_ExtFlash* variant. A specific installation process without swap is selected *SFU_NO_SWAP* to ensure confidentiality by keeping the download slot always encrypted.
- External flash memory without on-the-fly decryption (OTFDEC). An example is provided for the STM32WB5MM-DK board in the *2_Images_ExtFlash* variant. Download slot, as well as backup slot, is mapped in an external flash memory. A specific installation process without swap is selected *SFU_NO_SWAP* to ensure confidentiality by keeping both slots always encrypted. More details are provided in the *Appendix H* of the user manual *Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package [10]*.

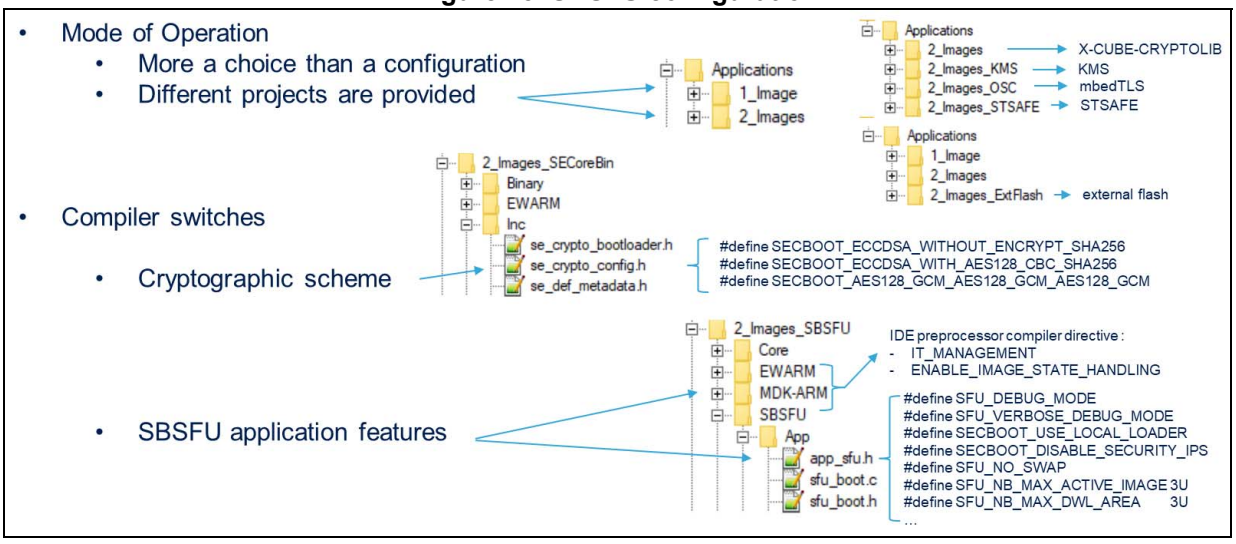
a. NUCLEO-WB55RG is referred to as P-NUCLEO-WB55.Nucleo in the X-CUBE-SBSFU package.

The configuration possibilities go beyond these options through compilation switches:

- Local loader can be removed to reduce the memory footprint (dual slots only).
- Verbose switch can be activated to make debugging easier.
- Debug mode can be disabled (no more printf on the terminal during SBSFU execution) to reduce the memory footprint.
- Security IPs can be turned off to make debugging easier.
- Installation process with firmware image validation. A rollback on the previous firmware image is triggered at the next reset if the firmware image has not been validated by the user application.
- Multiple image configuration for a complex system with multiple firmware such as protocol stack, middleware, and user application.
- Interruption management inside the firewall isolated environment for applications requiring low latency on interruption handling.

Figure 15 presents the SBSFU configuration solutions with the related files and compilation switches.

Figure 15. SBSFU configuration



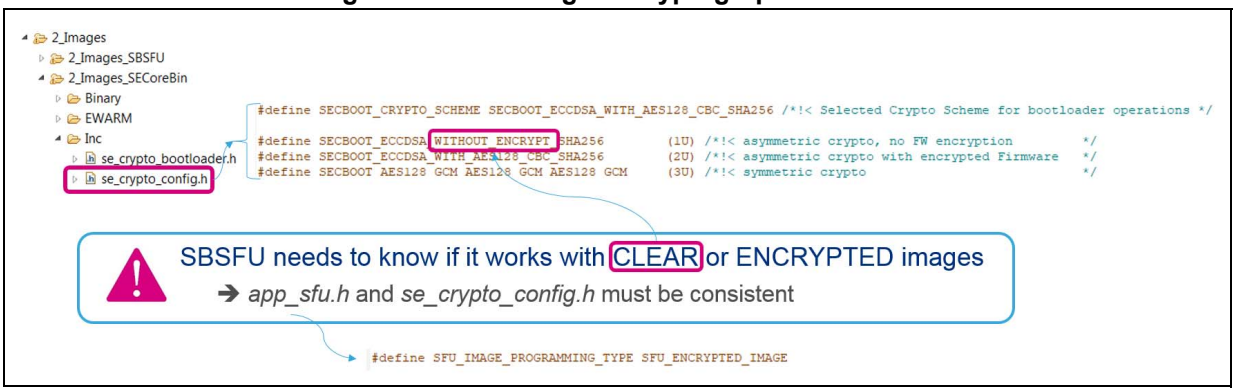
4.2 Cryptographic scheme selection

X-CUBE-SBSFU is delivered with three cryptographic schemes using both asymmetric and symmetric cryptography:

- ECDSA asymmetric cryptography for firmware verification and AES-CBC symmetric cryptography for firmware decryption
- ECDSA asymmetric cryptography for firmware verification without firmware encryption.
- AES-GCM symmetric cryptography for both firmware verification and decryption

The selection among these schemes is done using the `SECBOOT_CRYPTO_SCHEME` compilation switch as depicted in *Figure 16*.

Figure 16. Switching the cryptographic scheme



Note: For the B-L4S5I-IOT01A STSAFE and KMS variants, the SECBOOT_X509_ECDSA_WITHOUT_ENCRYPT_SHA256 cryptographic scheme is selected.

For the external flash memory variant with on-the-fly decryption (OTFDEC), the SECBOOT_ECCDSA_WITH_AES128_CTR_SHA256 cryptographic scheme is selected.

4.3 Security configuration

The SBSFU example is delivered with STM32 security protection configuration allowing protection secrets against both outer and inner attacks.

STM32 security peripherals can be deactivated independently as per the user's decision to achieve a different protection level (For example with STM32L4 series devices, firewall and PCROP allow the activation of protections against inner attacks). Any STM32 security configuration modification requires a security protection evaluation at the system product level to ensure that protections are well set according to product constraints and specifications.

During the development phase, the disabling of all IPs may be required for making debugging easier.

Figure 17 shows the various security configuration solutions available in file *app_sfu.h* for the STM32L4 series and STM32L0 series.

Figure 17. STM32L4 series and STM32L0 series security configuration (*app_sfu.h*)

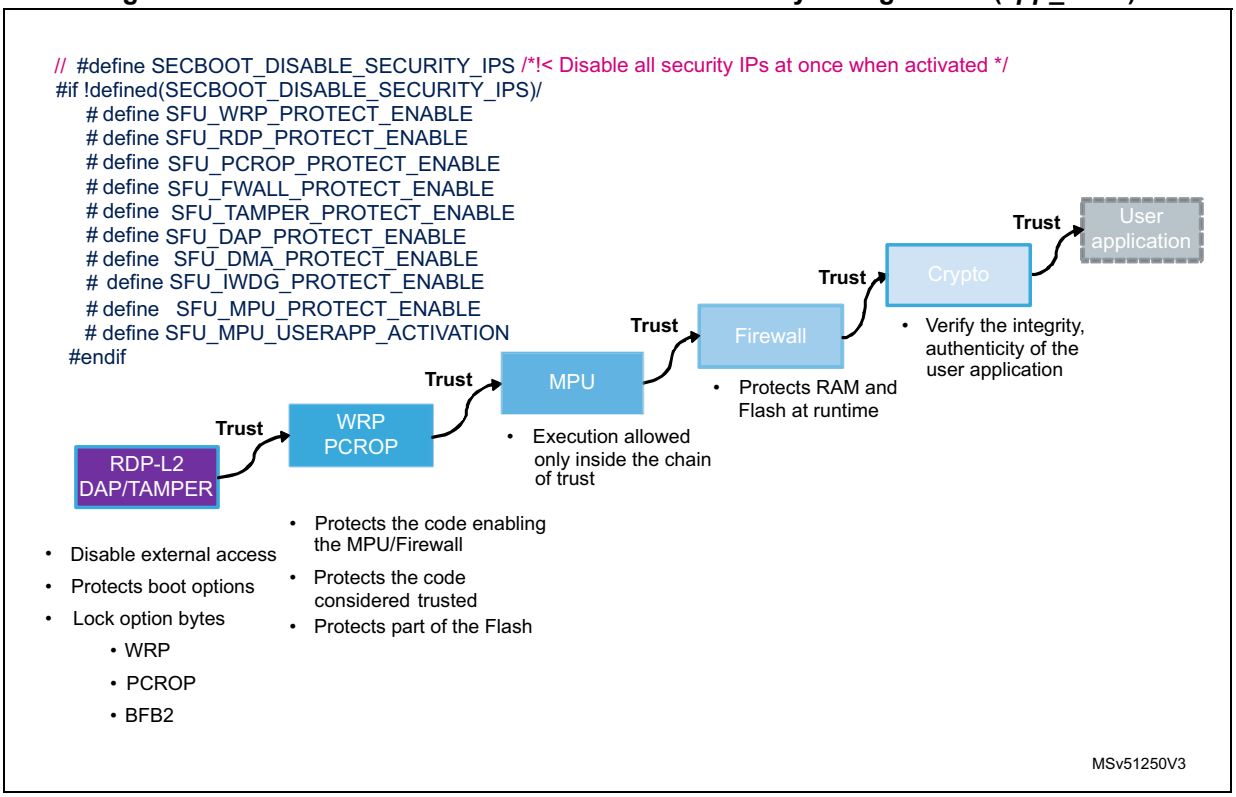


Figure 18 shows the various security configuration solutions available in file *app_sfu.h* for the STM32F4 series, STM32F7 series, and STM32L1 series.

Figure 18. STM32F4 series, STM32F7 series and STM32L1 series security configuration (*app_sfu.h*)

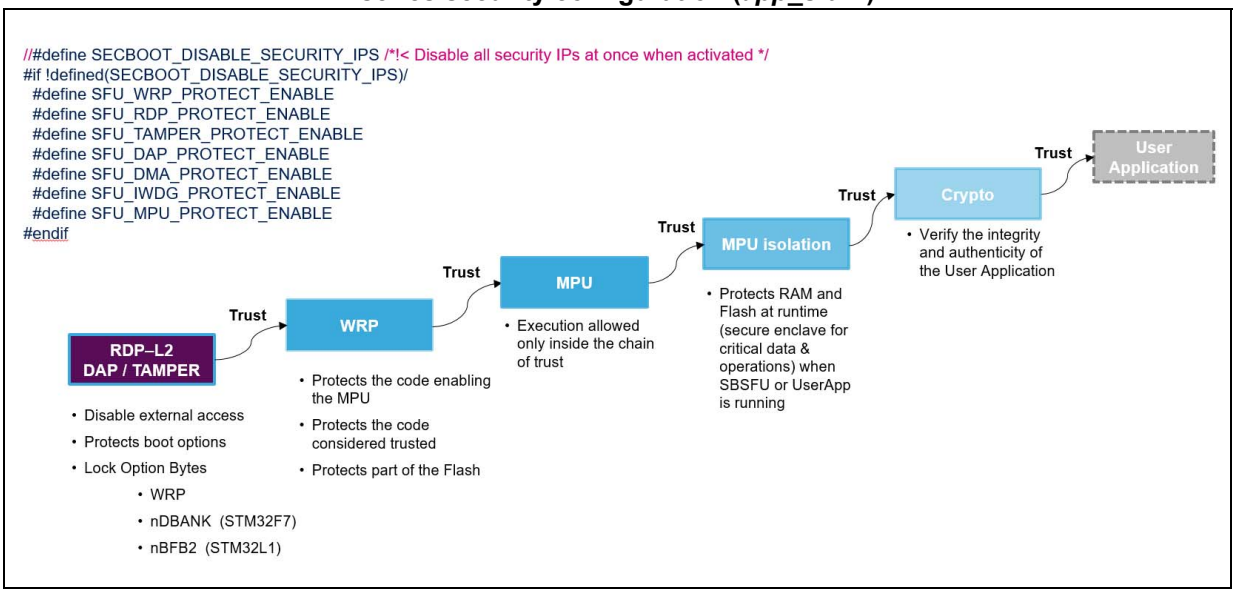


Figure 19 shows the various security configuration solutions available in file *app_sfu.h* for the STM32G0 series, STM32G4 series, and STM32H7 series.

Figure 19. STM32G0 series, STM32G4 series, and STM32H7 series security configuration (*app_sfu.h*)

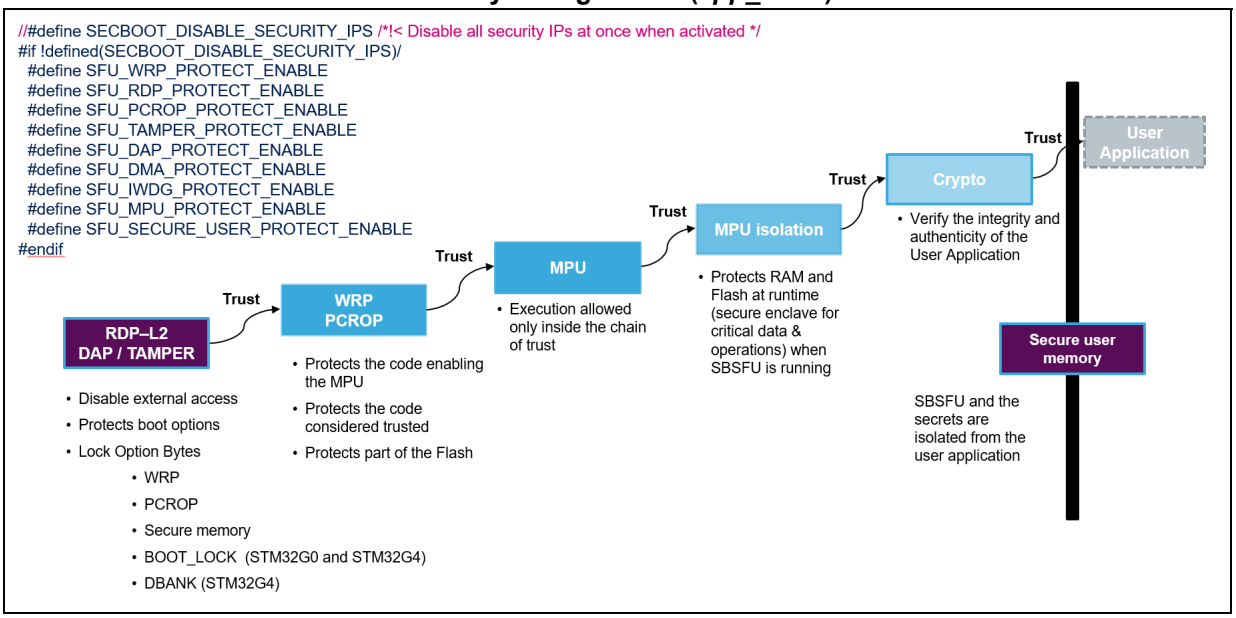
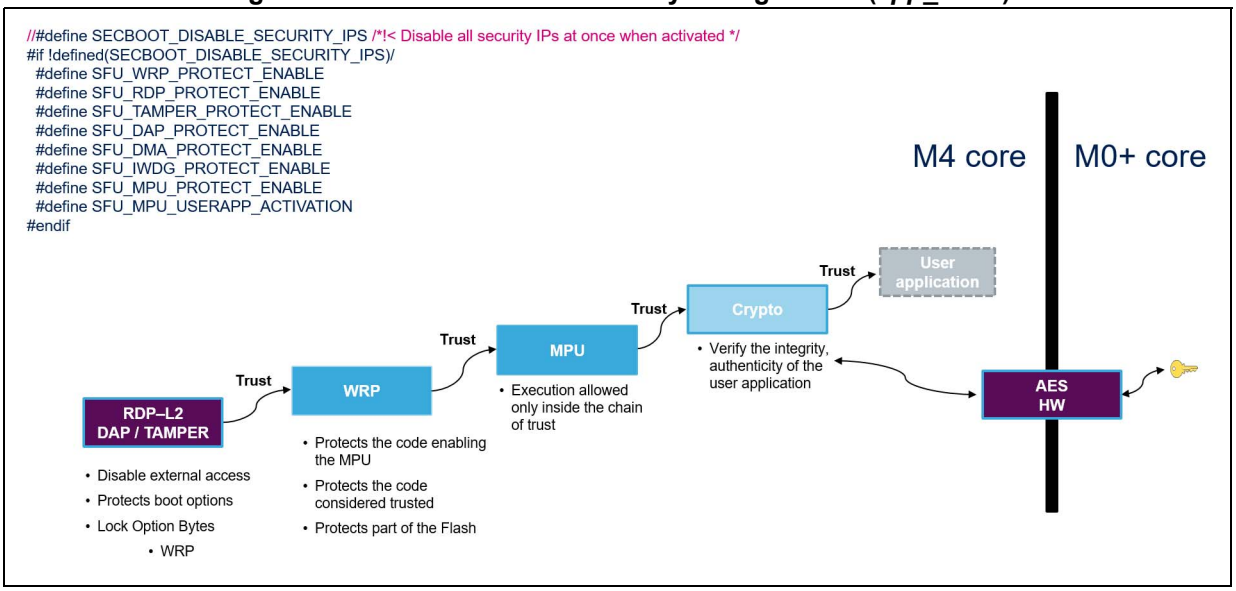


Figure 20 shows the various security configuration solutions available in file *app_sfu.h* for the STM32WB series.

Figure 20. STM32WB series security configuration (*app_sfu.h*)



4.4 Development or production mode configuration

The first step before any code modification is often to configure the SBSFU project in development mode to enable IDE debugging facilities and add SBSFU debug traces:

1. Deactivate all security protections: `SFU_xxx_PROTECT_ENABLE`
2. Deactivate `SFU_FINAL_SECURE_LOCK_ENABLE`
3. Activate `SFU_FWIMG_BLOCK_ON_ABNORMAL_ERRORS_MODE`
4. Activate `SECBOOT_OB_DEV_MODE`
5. Optionally, activate the verbose mode: `SFU_VERBOSE_DEBUG_MODE`. For details about the impact on mapping, refer to *Section 6.2: Memory mapping adaptation*.

At the end of the development phase, the SBSFU project must be configured in production mode for the final release:

1. Activate all required security protections: `SFU_xxx_PROTECT_ENABLE`
2. Deactivate verbose mode: `SFU_VERBOSE_DEBUG_MODE`
3. Deactivate `SFU_FWIMG_BLOCK_ON_ABNORMAL_ERRORS_MODE`
4. Deactivate `SECBOOT_OB_DEV_MODE`
5. Activate `SFU_FINAL_SECURE_LOCK_ENABLE` to configure the RDP level 2. On STM32H7 series, the secure user memory is also configured when `SFU_FINAL_SECURE_LOCK_ENABLE` is enabled.
6. Deactivate `SFU_DEBUG_MODE` to remove all prints of SBSFU that can be valuable information for an attacker.

Read Protection Level 2 is mandatory to achieve the highest level of protection and to implement a Root of Trust. It is the user's responsibility to activate it in the final SW to be programmed during the product manufacturing stage.

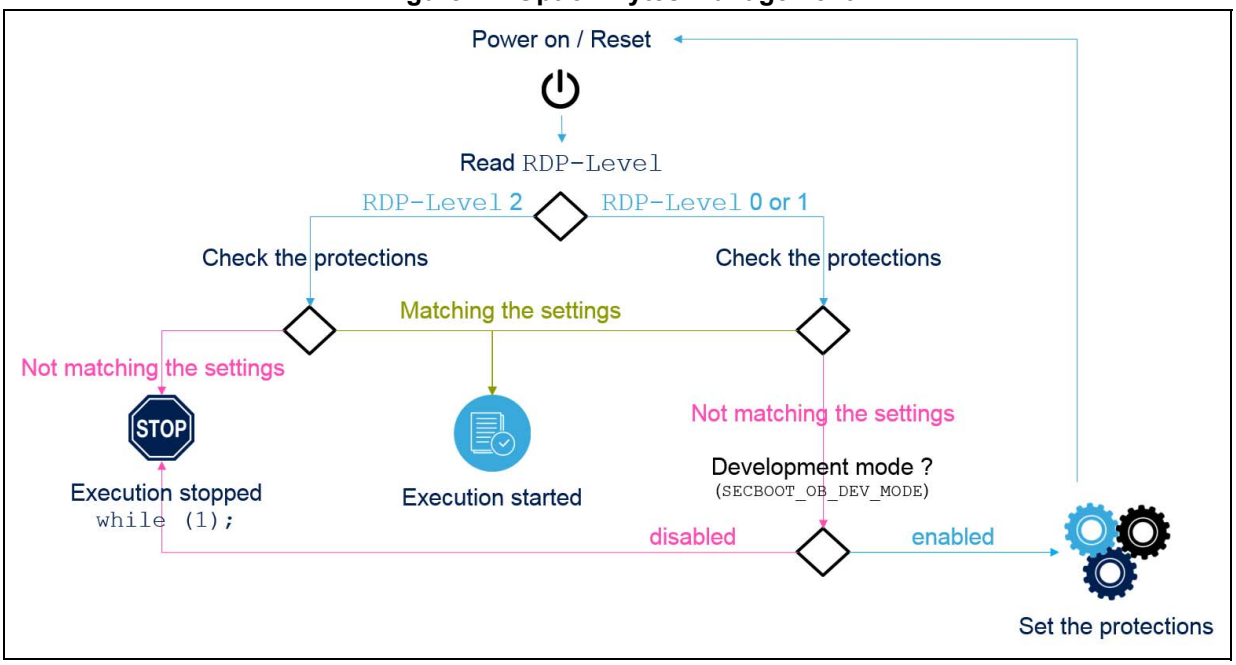
In production mode, the secure boot checks the Option Byte values (RDP, WRP, PCROP, Secure user memory) and blocks execution in case a wrong configuration is detected. Depending on the platform, a few other Option Bytes must be configured such as:

- BFB2 disabled for STM32L4 series and STM32L0 series devices with dual-bank flash memory
- nDBANK enabled for STM32F7 series
- nBFB2 enabled for STM32L1 series
- BOOT_LOCK enabled for STM32G0 series and STM32G4 series
- DBANK disabled on STM32G4 series and B-L4S5I-IOT01A board

Caution: Option Bytes must be configured to the production mode values using STM32CubeProgrammer (STM32CubeProg), just after programming the software during the production stage. If this is not done, the device remains unsecured. Refer to *[13]* for the way to use STM32CubeProgrammer.

Figure 21 shows how Option Bytes are managed at SBSFU startup:

Figure 21. Option Bytes management



5 Generating a cryptographic key

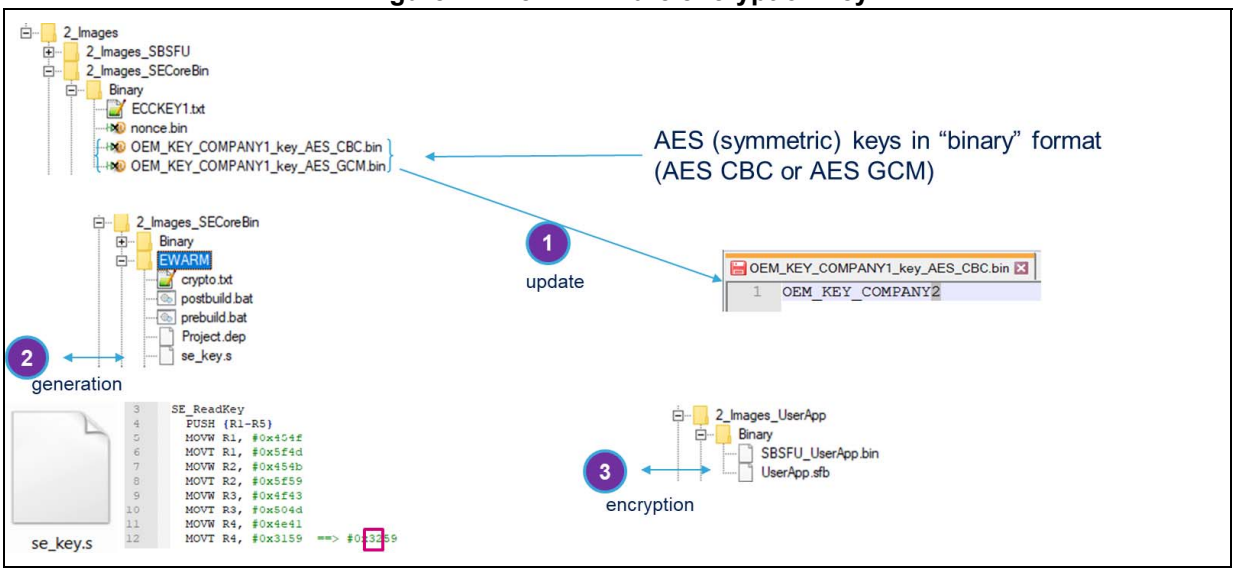
5.1 Generating a new firmware AES encryption key

Key generation and firmware encryption are performed automatically during the compilation process with the *prebuild.bat* and *postbuild.bat* scripts (refer to *[10]* for a detailed description of the build process).

Figure 22 shows the few steps to modify the firmware encryption key of active slot #1. The same applied to active slot #2 or #3:

1. Change the key value in file *OEM_KEY_COMPANY1_keys_AES_xxx.bin*
2. Compile SECoreBin: *prebuild.bat* is executed and *se_key.s* is generated
3. Compile UserApp: *postbuild.bat* is executed and UserApp is encrypted

Figure 22. New firmware encryption-key



5.2 Generating a new public/private ECDSA pair of keys for firmware verification

As for the AES encryption key, the public key (`SE_ReadKey_Pub()`) is automatically modified when the private key (*ECCKEY1.txt*) is changed.

Figure 23 shows the few steps to modify the private and public keys for ECDSA asymmetric cryptography firmware verification of the active slot #1. The same applied for active slot #2 or #3:

1. Change the key value in file *ECCKEY1.txt*
2. Compile SECoreBin: *prebuild.bat* is executed and *se_key.s* is generated
3. Compile UserApp: *postbuild.bat* is executed and UserApp is encrypted

Figure 23. New private/public keys

5.3 STM32WB series specificities

For STM32WB series, the AES encryption key is not processed through the *prebuild.bat* script but is provisioned into the M0+ core. The provisioning process is described in *SECoreBin/readme.txt*.

Another way to provision the AES key is to use the recent *STM32CubeProgrammer* release. Since V2.5.0, M0+ key provisioning is available as Firmware Upgrade Service (FUS).

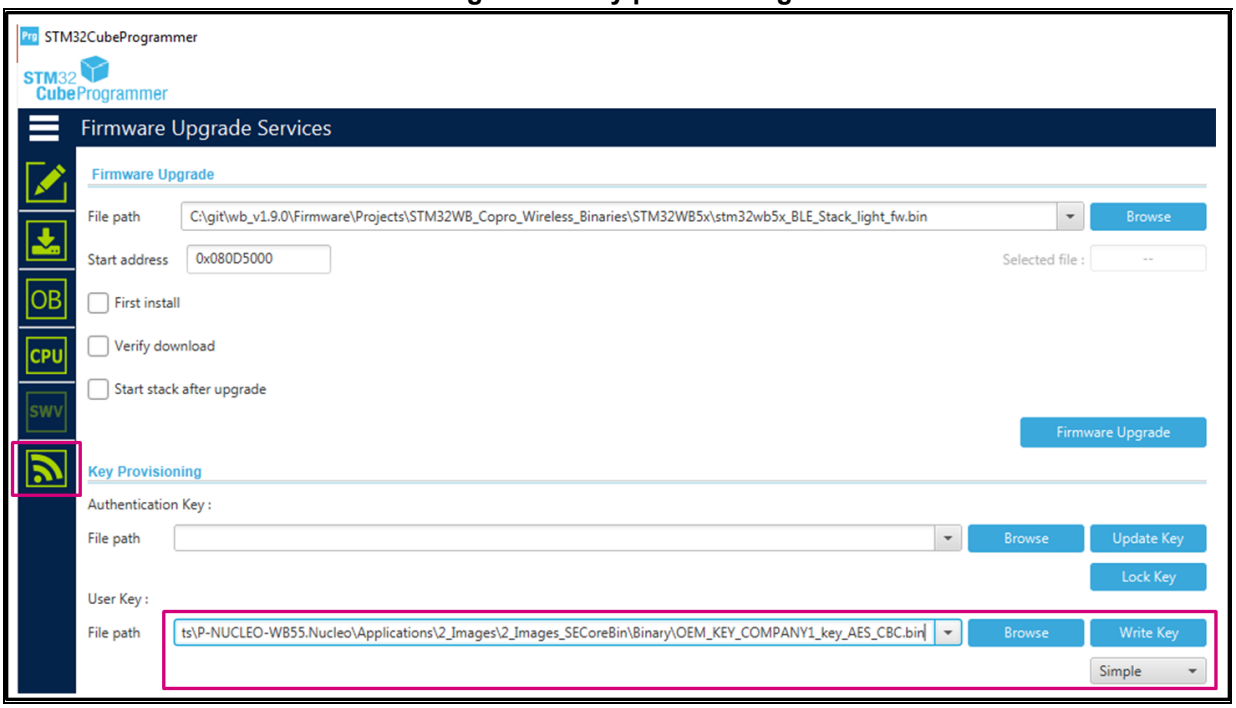
First, connect to the bootloader USB interface:

1. nBOOT1 and nSWBOOT0 are checked.
2. Correct boot mode is selected by setting Boot0 pin to VDD:
   a) With a NUCLEO-WB55RG Nucleo board[(a)]: The jumper is ON between CN7.5 (VDD) and CN7.7 (Boot0).
   b) With an STM32WB5MM-DK Discovery board: A jumper is ON on CN13(VDD-Boot0) after pin header soldering and another jumper selects ‘USB MCU’ on JP2.
3. A USB cable is connected to the USB_USER interface.
4. The power is ON (unplug/plug USB cable is connected to ST-LINK).

Then, the function *Key provisioning* of the *Firmware Upgrade Services* panel is allowed as shown in *Figure 24*.

a. NUCLEO-WB55RG is referred to as P-NUCLEO-WB55.Nucleo in the X-CUBE-SBSFU package.

Figure 24. Key provisioning



5.4 KMS specificities

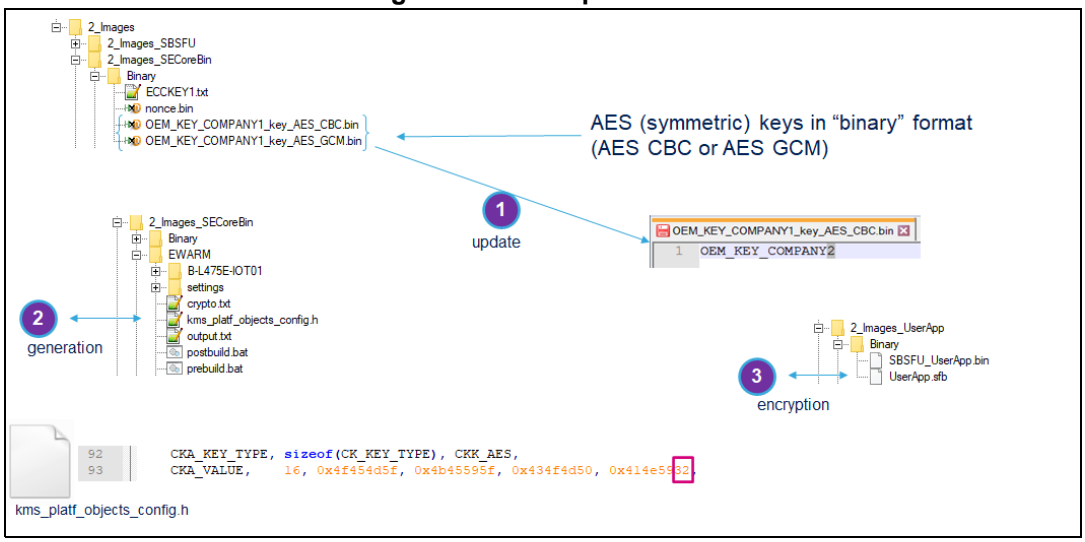
With KMS middleware integration, SBSFU keys are no longer stored in a section under PCROP protection. They are stored inside the KMS code as static embedded keys.

Figure 25 shows an example of the firmware encryption key modification of active slot #1. The same applied for active slot #2 or #3:

1. Change the key value in file *OEM_KEY_COMPANY1_keys_AES_xxx.bin*
2. Compile SECoreBin: *prebuild.bat* is executed and *kms_platf_objects_config.h* is generated
3. Compile UserApp: *postbuild.bat* is executed and UserApp is encrypted

The same process is applied for firmware ECDSA verification key, BLOB AES encryption key, and BLOB ECDSA verification key.

Figure 25. KMS specificities



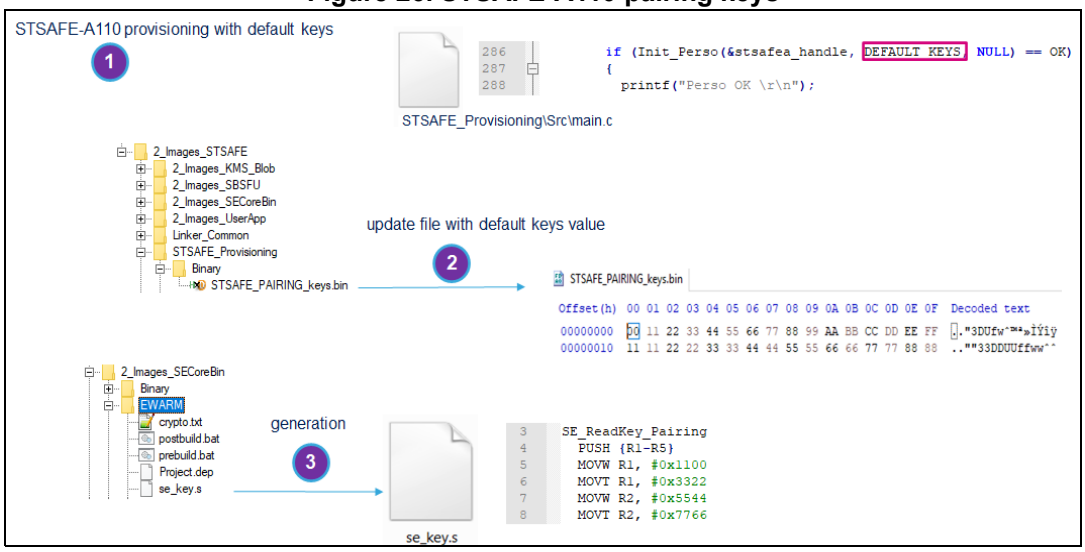
5.5 STSAFE-A110 specificities

As explained in Appendix G of the UM2262, STM32 and STSAFE-A110 must be provisioned with pairing keys and X509 certificates.
STSAFE-A110 provisioning process is described in *STSAFE_Provisioning/readme.txt*.

Figure 26 shows an example of pairing-key provisioning:

1. STSAFE-A110 provisioning with default pairing keys
2. Update *STSAFE_PAIRING_keys.bin* accordingly
3. Compile SECoreBin: *prebuild.bat* is executed and *se_key.s* is generated.

Figure 26. STSAFE-A110 pairing keys

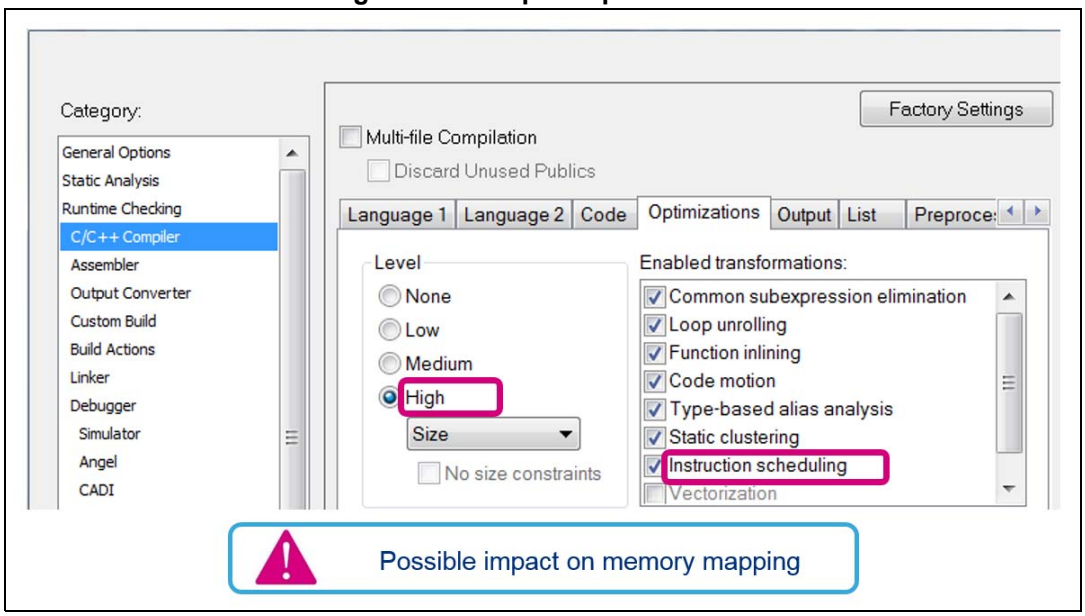


6 Tips for debugging

6.1 Compiler optimizations level

Projects are delivered with the highest level of compiler optimizations turned on for size aspects. Such optimizations can make the debug complex. Changing the compiler optimization level possibly impacts memory mapping.

Figure 27. Compiler optimizations



6.2 Memory mapping adaptation

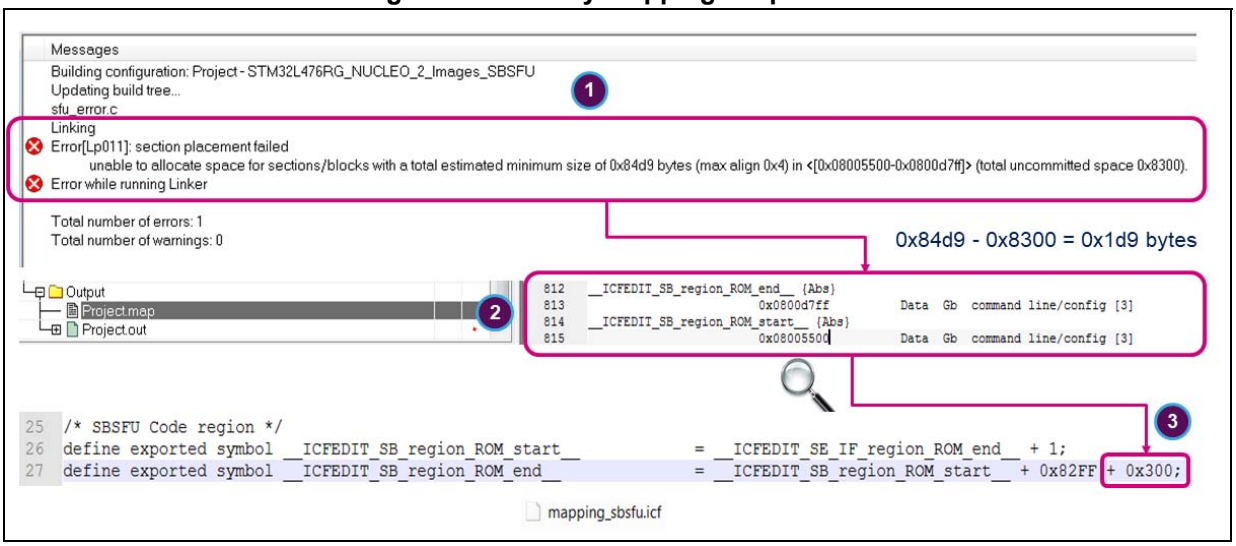
When changing the compiler optimizations level or activating the development mode with the verbose compilation switch, the user may have to adapt the SBSFU memory mapping, for instance reducing firmware image slots to avoid overlap.

Caution: The security peripheral configuration (RDP, WRP, PCROP, FWALL, secure user memory if available for the series) is automatically computed based on the SBSFU linker symbols except for the MPU configuration due to the constraints detailed in *Section 3.2: Memory mapping definition*. Disabling temporarily the MPU protection can be an efficient workaround for the debug.

Figure 28 depicts the 3 steps of the memory adaptation based on an example:

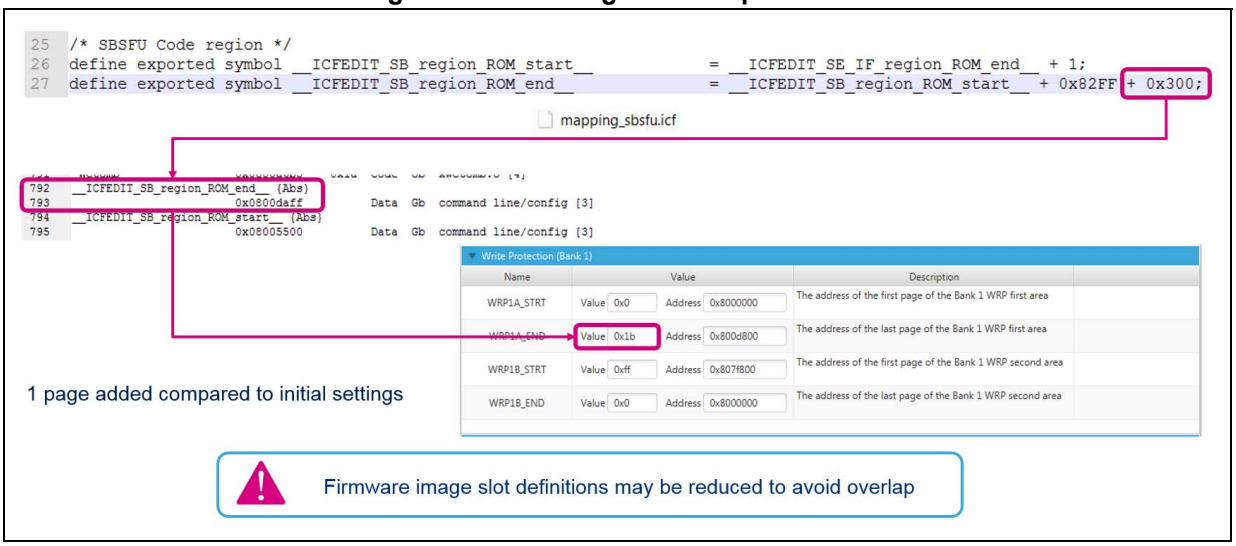
1. Identify the gap by analyzing the linker message: 0x1d9 bytes
2. Identify the concerned region by consulting the *project.map* file: `__ICFEDIT_SB_region_ROM_start__`
3. Apply the modification in file *mapping_sbsfu.icf:* 0x300 bytes

Figure 28. Memory mapping adaptations



The impact of memory mapping adaptation on security peripheral configurations must be checked even though it is automatically computed. For example, check the WRP configuration using STM32CubeProgrammer (STM32CubeProg) as shown in *Figure 29*.

Figure 29. Checking the WRP protection

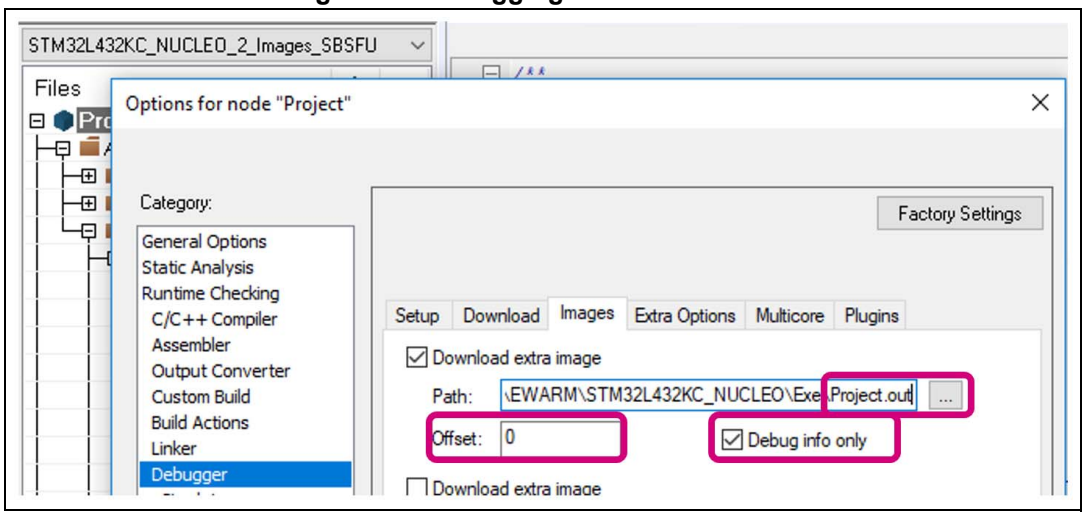


6.3 Debugging SECoreBin

To debug inside SECoreBin, the SBSFU project option must be changed to load SECoreBin symbols. This is performed in the debugger menu as presented in *Figure 30*:

- Browse to select file *Project.out*
- Set *Offset* to 0
- Check the *Debug info only* box

Figure 30. Debugging inside SECoreBin



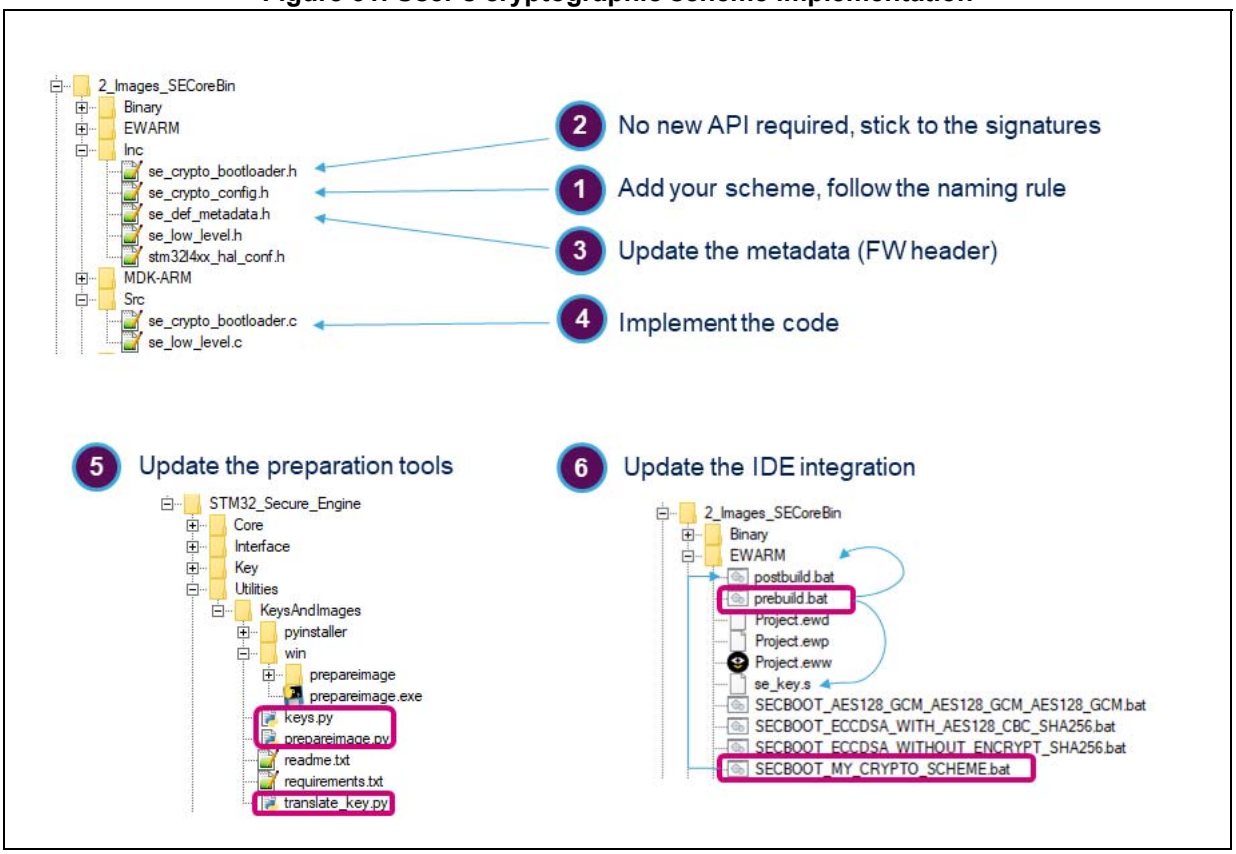
7 Adapting SBSFU

7.1 Implementing a new cryptographic scheme for SBSFU

X-CUBE-SBSFU comes with some predefined cryptographic schemes (refer to *Section 4.2: Cryptographic scheme selection on page 21*). It is also possible to extend the package with the user's cryptographic scheme.

To implement a new cryptographic scheme for SBSFU, follow the steps illustrated in *Figure 31* and described below.

Figure 31. User's cryptographic scheme implementation



Updating the code running on the device side:

1. **Step 1**: define a new value for `SECBOOT_CRYPTO_SCHEME`.
2. **Step 2**: look carefully at the signatures of the APIs that the bootloader requires. The cryptographic services must have the same signatures to avoid updating the SBSFU code.
3. **Step 3**: define a new `SE_FwRawHeaderTypeDef` structure and respect the constraints to remain compatible with the existing SBSFU code.
4. **Step 4**: implement the code of the cryptographic services in *se_crypto_bootloader.c*.

Updating the tools running on the host side to prepare the keys and the firmware image:

5. **Step 5**: update the preparation tools to support the new cryptographic scheme: *prepareimage.py*, *translate_key.py*, and *keys.py*.
6. **Step 6**: update the IDE integration to generate the appropriate keys and firmware image.
   – A new batch file is required to call the preparation tools with the appropriate commands; *prebuild.bat* copies this batch file to create *postbuild.bat*.
   – *prebuild.bat* must be updated to take into account the new cryptographic scheme and generate the proper keys and *postbuild.bat.*

7.2 Optimizing memory mapping

Several options exist to reduce SBSFU code size to maximize the size of the user application slot. Some of these options are summarized in *Table 3*.

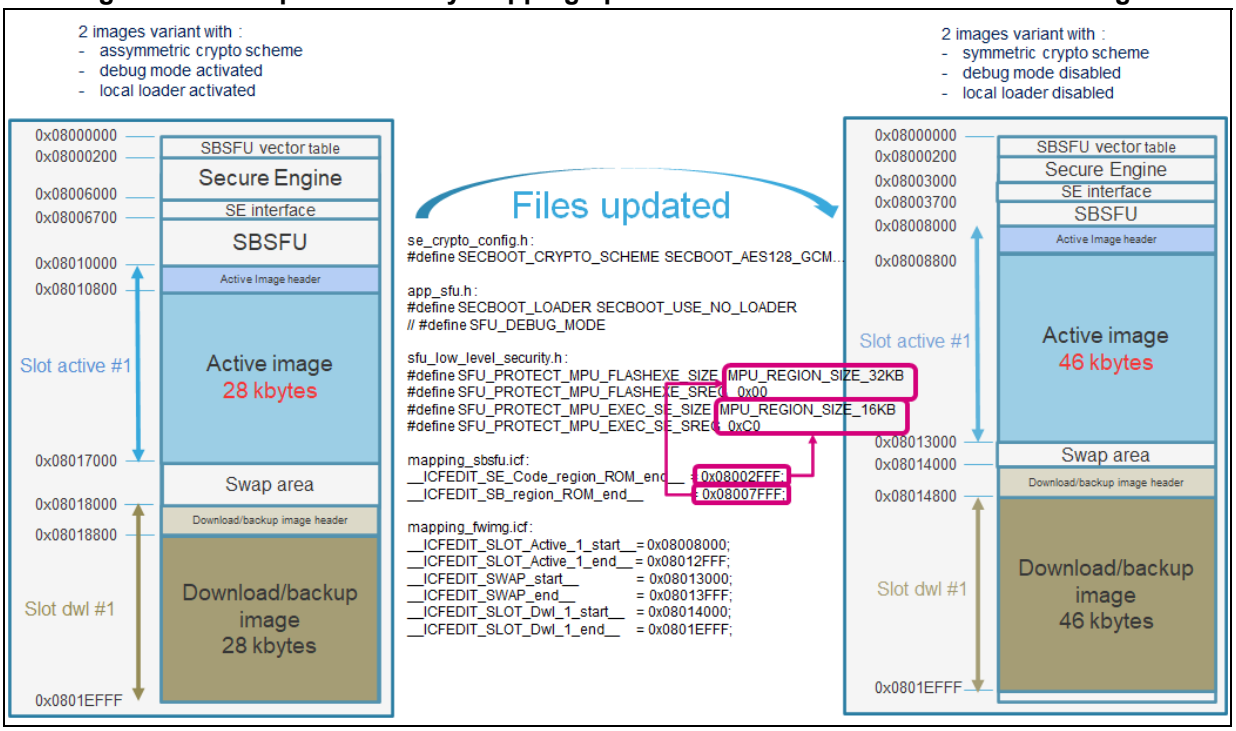
Table 3. SBSFU code-size reduction

Option	Description / Consequence	Gain
Select 1-image variant	Download a new firmware image from the user application is no more possible.	Slot size is doubled vs. 2-image projects
Select AES-GCM symmetric cryptographic scheme	Shared symmetric key secret stored in the device.	~ 9 Kbytes
Disable `SFU_DEBUG_MODE`	No more information displayed on the terminal during SBSFU execution	~ 9 Kbytes
Disable `SECBOOT_USE_LOCAL_LOADER`	No more local loader inside the SBSFU application. This is not compatible with 1-image variant.	~3 Kbytes
Implement a hardware decryption	Select STM32 devices integrating cryptographic hardware IP.	This depends on the user implementation
If all the code running on STM32 is fully trusted and robust then Secure Engine internal isolation based on MPU for STM32F4/F7/G0/G4/H7/L1 can be removed.	Removes alignment constraints with MPU regions.	Up to 12 Kbytes depending on products

The total gain depends on the mapping constraints described in *Section 3.2: Memory mapping definition on page 10*.

As an example, *Figure 32* highlights the mapping modifications to be done. Starting from 2 images with a symmetric cryptographic scheme, the `SFFU_DEBUG_MODE` and `SECBOOT_USE_LOCAL_LOADER` switches are disabled, resulting in a 16-Kbyte increase of the user application size.

Figure 32. Example of memory mapping optimization on NUCLEO-G071RB – 2 images



In the folder *NUCLEO-G031K8\Applications\1_Image,* another example of memory optimization is provided for the NUCLEO-G031K8, where 32 Kbytes are allocated to the user application among the 64 Kbytes available on this board.

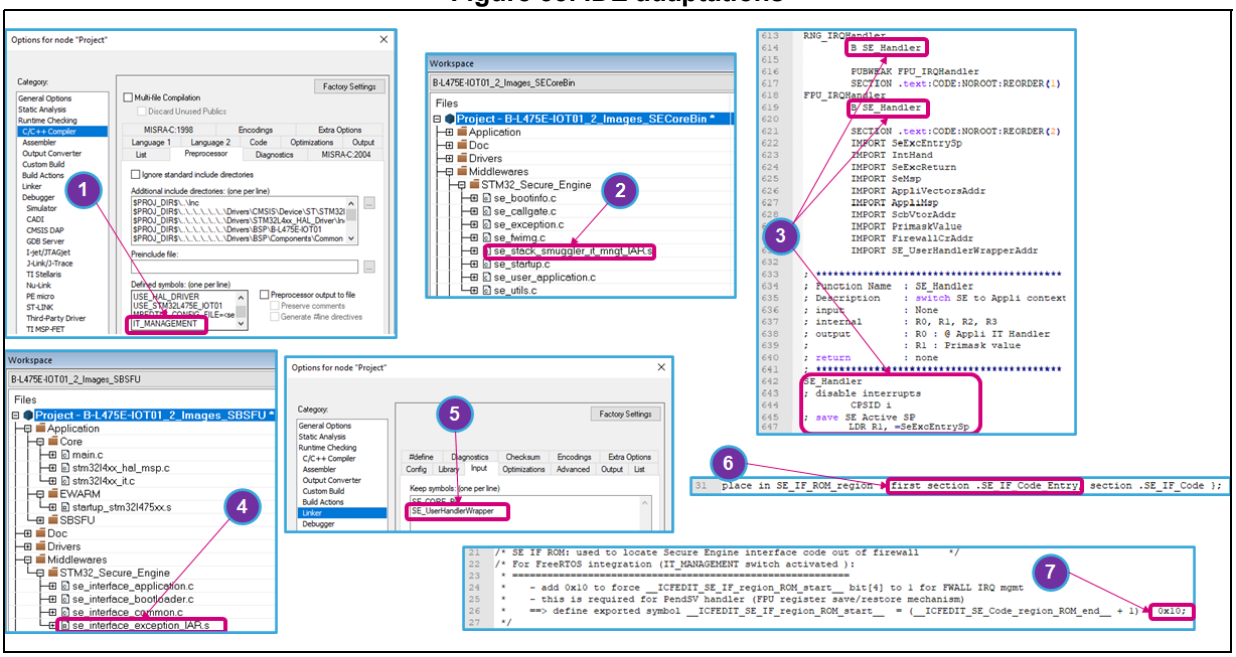
7.3 How to activate interruption management inside the firewall isolated environment

Interruption management inside the firewall isolated environment can be activated when low latency on interruption handling is required. Examples are provided in the *2_Images_OSC* variant for 32L496GDISCOVERY and B-L475E-IOT01A boards.

Figure 33 shows the different steps required to activate this option:

1. Add `IT_MANAGEMENT` as preprocessor directive in SECoreBin and SBSFU IDE configuration.
2. Select *se_stack_smuggler_it_mngt_IAR.c* instead of *se_stack_smuggler_IAR.c* in SECoreBin IDE configuration.
3. Modify *startup_xxx.s* file to branch required interrupt handler on *SE_Handler.*
4. Add *se_interface_exception_IAR.s* in SBSFU IDE configuration.
5. Modify the SBSFU linker option to keep `SE_UserHandlerWrapper` symbols.
6. Modify SBSFU *xxx_flash.icf* linker file to place `SE_IF_Code_Entry` symbol (*SE_UserHandlerWrapper*) at the beginning of *SE_IF_ROM_region*.
7. Specific FreeRTOS™: Modify *mapping_sbsfu.icf* by adding 0x10 to force *__ICFEDIT_SE_IF_region_ROM_start__ bit[4]* to 1. This is required for PendSV handler (FPU register save/restore mechanism).

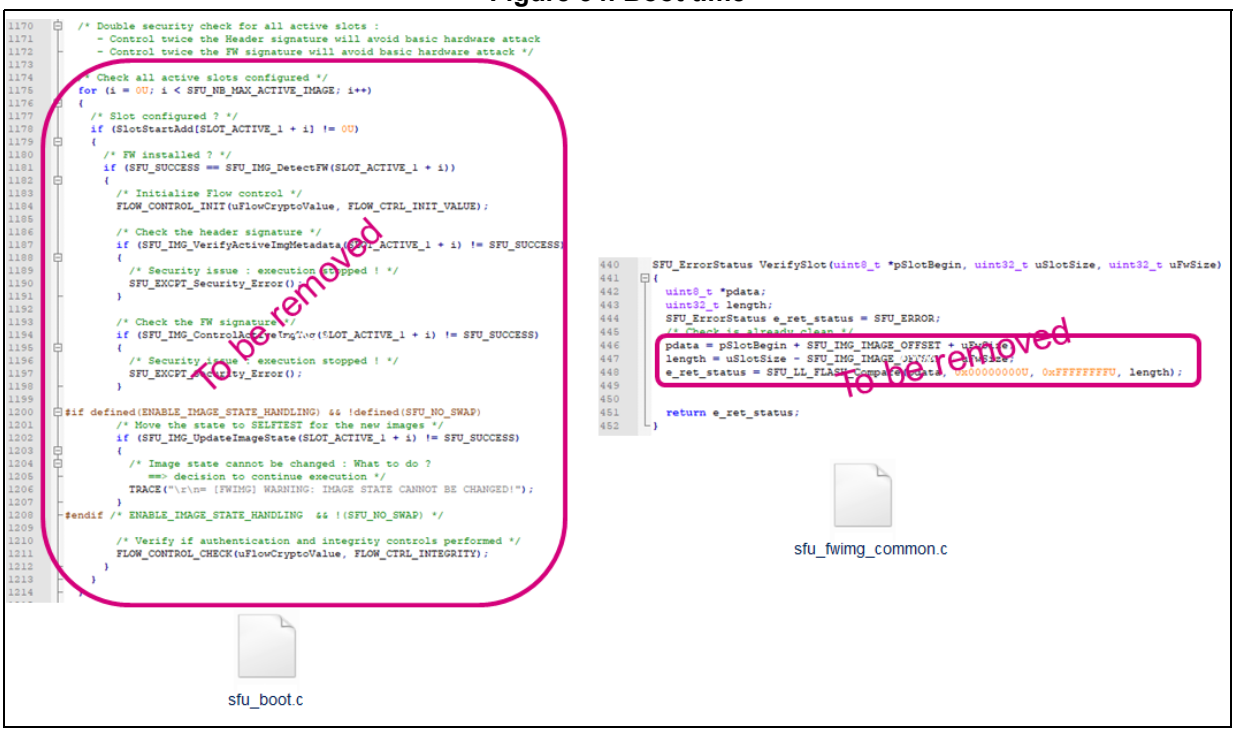
Figure 33. IDE adaptations



7.4 How to improve boot time

To resist a basic fault injection attack, some critical actions are duplicated thus are impacting the time to start the user application. If such protections are not needed, for example, if there is no physical access to the device, these counter-measures can be removed as shown in *Figure 34*.

Figure 34. Boot time



7.5 Handling ITCM/TCM regions in ELF files

When building an SBSFU project on STM32 microcontrollers that use the instruction tightly coupled memory (ITCM) region, users might encounter a situation in which the *prepareimage.py* script generates an incorrect SBSFU_UserApp image.

The *prepareimage.py* script uses the ELF section headers to determine the lowest memory address of the application. It calls the `find_lowest_section()` function, which iterates over all allocated, non-NOBITS sections to find the base address of the image.

The problem occurs because the ITCM region, typically at addresses such as 0x00000000 or below 0x08000000, is mapped at a very low address. When the *prepareimage.py* script reads the ELF section headers, it detects the .itcm_text section and incorrectly identifies it as the lowest address of the user application.

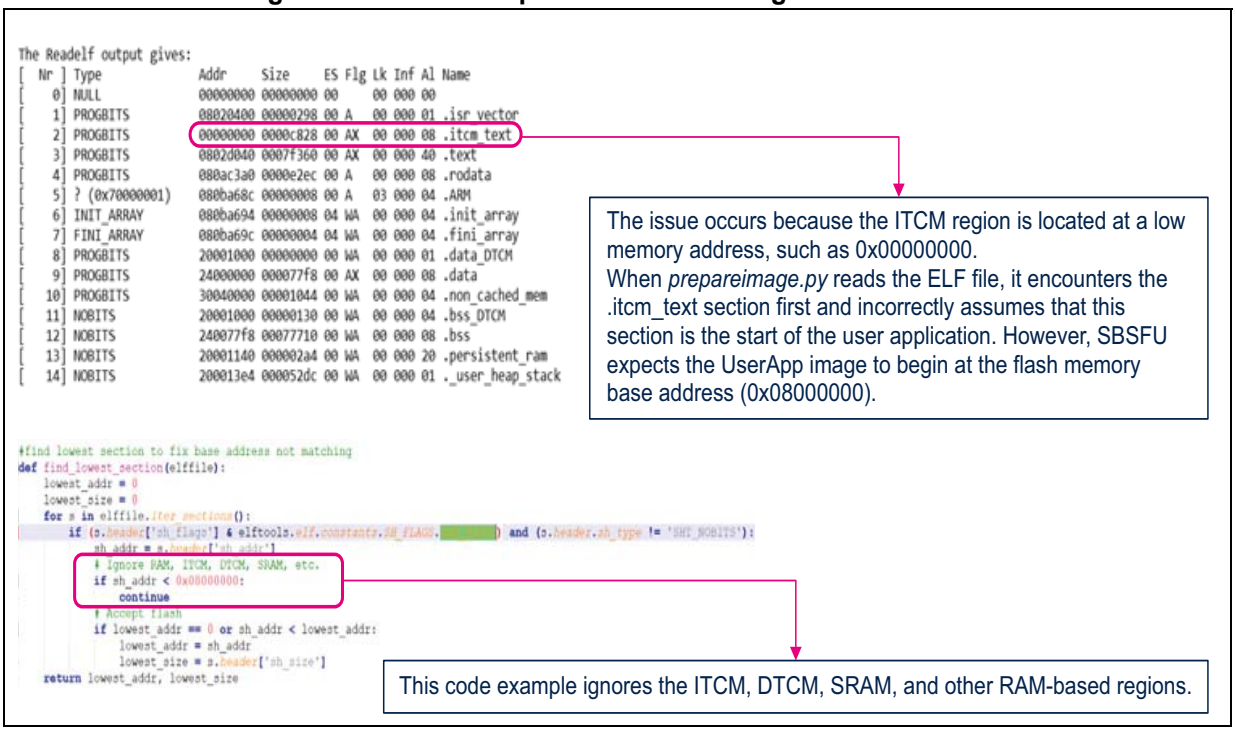
As a result, the image is generated with an incorrect base address, which leads to a corrupt or invalid SBSFU_UserApp image.

The *prepareimage.py* script does not recognize that the ITCM region (.itcm_text at 0x00000000) must not be included when calculating the base address. To resolve this issue, modify the `find_lowest_section()` function in *prepareimage.py* to skip any section with an address below 0x08000000. Ensure that the function correctly excludes

ITCM, DTCM, SRAM, and other RAM-mapped regions from the base address calculation as illustrated in *Figure 35*.

Note: This issue affects any STM32 project that uses ITCM for code execution and relies on the prepareimage.py script for SBSFU image generation. The original script does not recognize memory-mapped regions that are not part of flash memory, so the filtering must be performed explicitly.

Figure 35. Code example for ITCM/TCM regions in ELF files



8 Adapting the user application

8.1 How to make an application SBSFU compatible

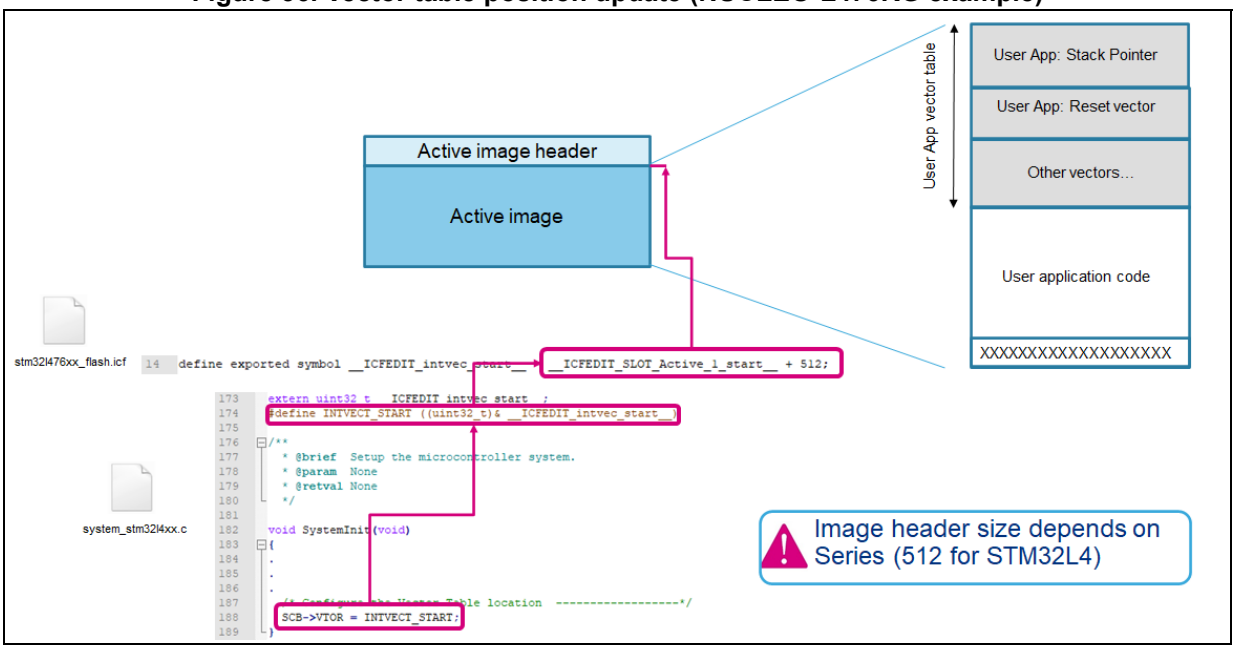
First of all, the mapping of the user application must be modified to allow the application to run in active slot #1. In a multiple image configuration the same applied for active slot #2 or #3:

- Code section starting by the vector table must be configured to run from active slot #1, just after the image header: `__ICFEDIT_SLOT_Active_1_start__` + 512 (`SFU_IMG_OFFSET` = 512 for the STM32L4 series)
- Data section must start after the Secure Engine protected area: (`__ICFEDIT_SE_region_SRAM1_end__` + 1)

Refer to *Section 3.2: Memory mapping definition on page 10* for more details on memory constraints.

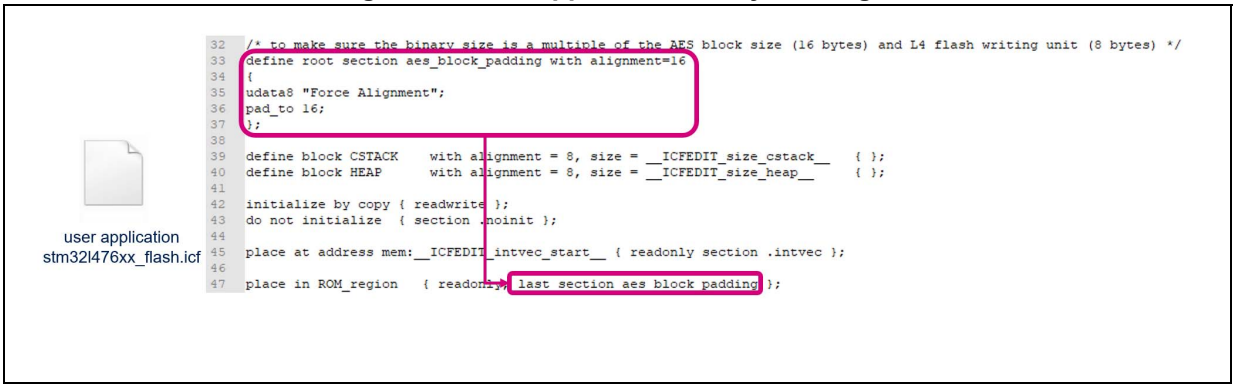
Then, during system initialization, VTOR must be set to the new location of the vector table as shown in *Figure 36*.

Figure 36. Vector table position update (NUCLEO-L476RG example)



For user application encryption, the user application binary file length must be a multiple of 16 bytes. *Figure 37* shows how to update the linker file to verify this constraint.

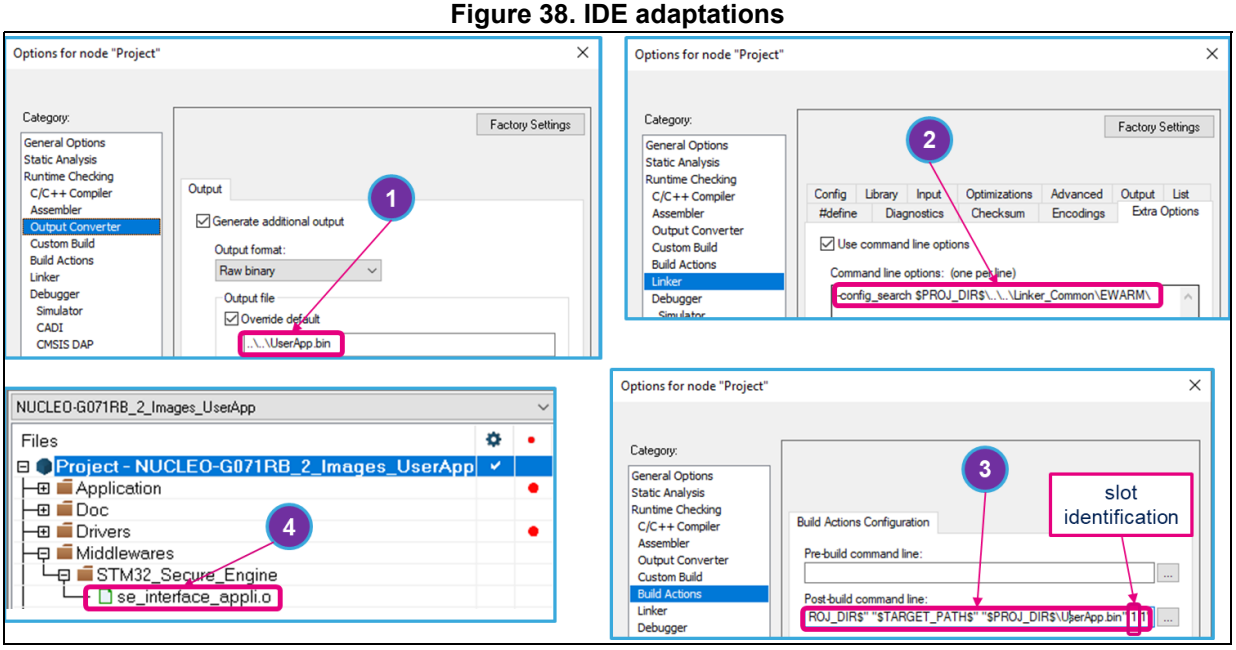
Figure 37. User application binary file length



Finally, as done in the UserApp example, the IDE configuration must be updated to:

1. Generate a *UserApp.bin* file
2. Include search path for linker common files
3. Call *postbuilb.bat* to generate *UserApp.sfb* and *SBFU_UserApp.bin* with the correct slot identification (1/2/3)
4. Integrate *se_interface_appli.o* to access Secure Engine runtime services if any

Figure 38. IDE adaptations



As explained in the X-CUBE-SBSFU user manual *[10]*, some additional constraints depend on the STM32 series:

- STM32F4 series, STM32F7 series, and STM32L1 series: MPU-based Secure Engine isolation relies fully on the fact that a privileged level of software execution is required to access the Secure Engine services. The user application must take this constraint into account and trust any piece of code running in privileged mode.
- STM32G0 series, STM32G4 series, and STM32H7 series: When secured, any access to securable memory area (fetch, read, programming, erase) is rejected, generating a bus error. As a consequence, there are no Secure Engine runtime services available for the user application.
- During the porting of the UserApp example generated with STM32CubeMX, the `HAL_RCC_DeInit()` function call is not automatically added for the STM32 family of boards. In the absence of the `HAL_RCC_DeInit()` function call, the UserApp might get stuck during boot. A specific call to `HAL_RCC_DeInit()` must be added before calling the `SystemClock_Configuration()` function.

Note: IWDG is started during SBSFU execution. It must be refreshed periodically.

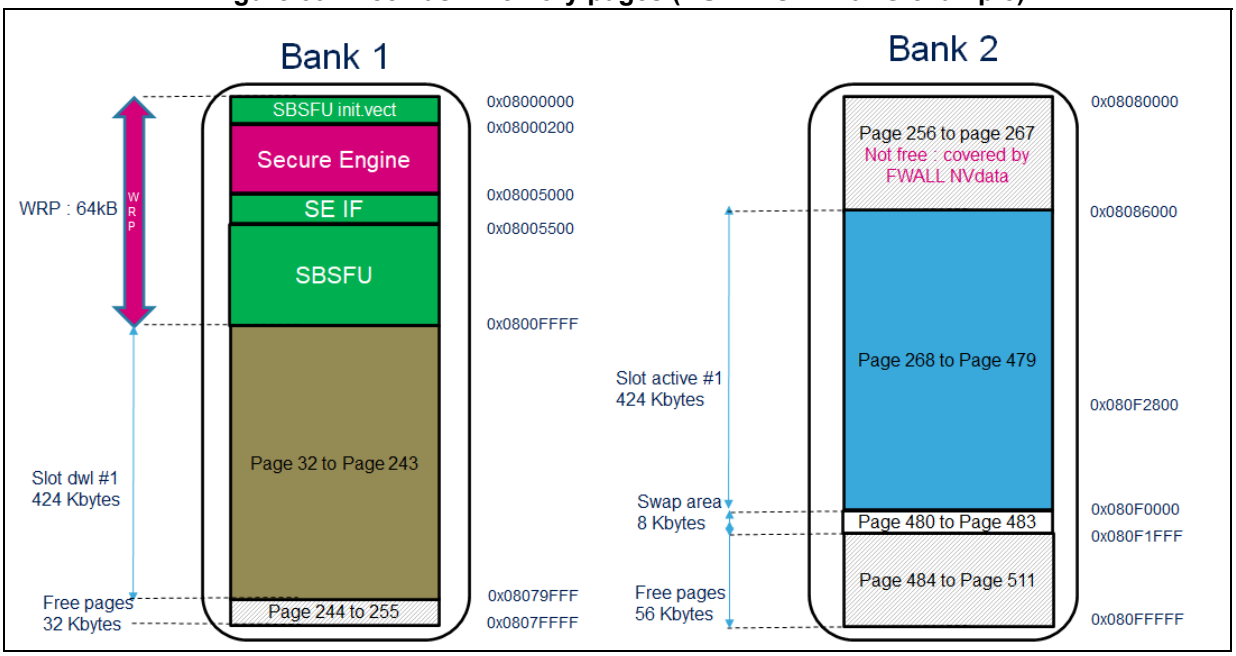
8.2 Use of flash memory to store user data

The storage of user data in flash memory pages or sectors is possible with some restrictions:

- Out of the SBSFU code area
- Not in the images slots
- Not in the swap area

Figure 39 provides a memory-mapping example based on the NUCLEO-L476RG where the flash memory is available from page 489 to page 511 for the user to store data, install a file system, or emulate an EEPROM.

Figure 39. Free flash memory pages (NUCLEO-L476RG example)

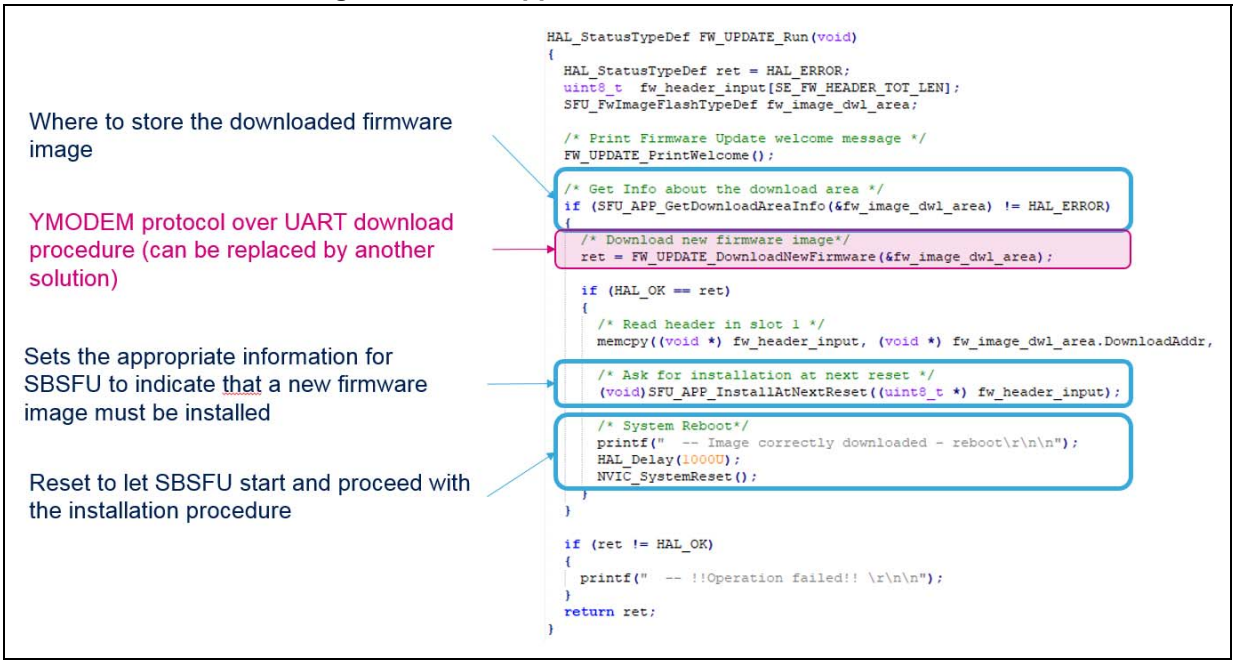


8.3 Changing the firmware download function in the user application

This possibility is available only in the dual-slot mode of operation.

A sample code based on the YMODEM protocol over UART is available in the X-CUBE-SBSFU UserApp project. The download procedure is located in file *fw_update_app.c* as illustrated in *Figure 40*.

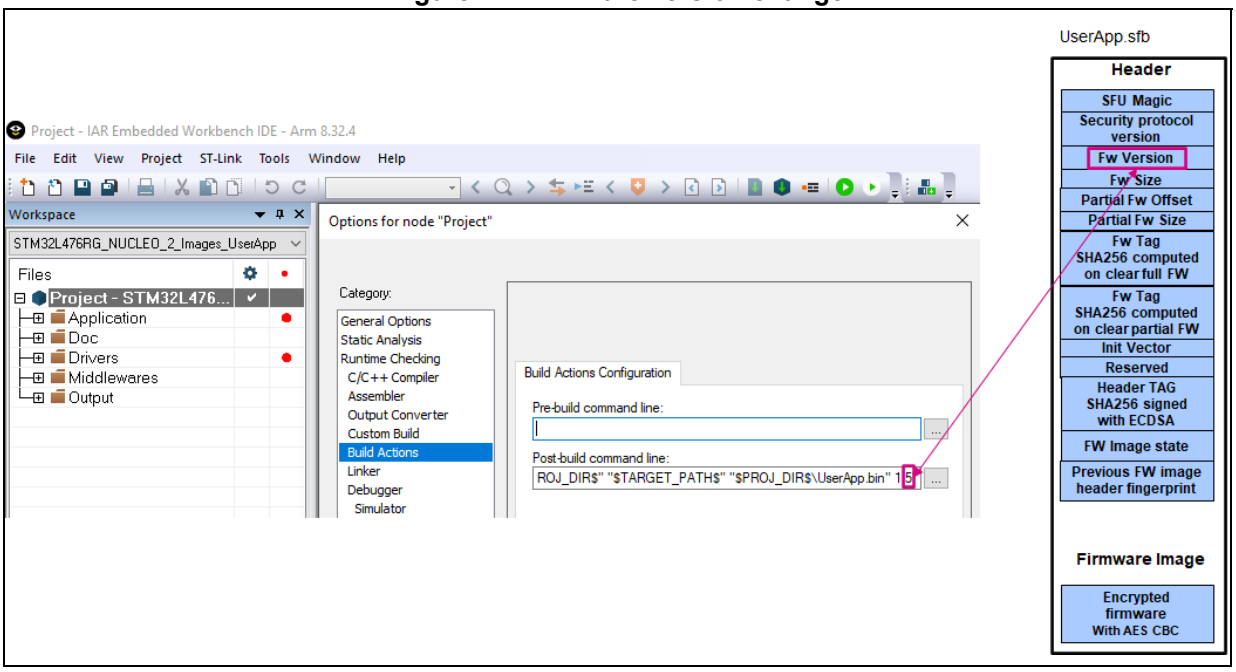
Figure 40. UserApp firmware download overview



8.4 How to change the firmware version

The firmware version is part of the firmware header generated with the *postbuild.bat* script. In the following example, the version is 5.

Figure 41. Firmware version change



Caution: The firmware with version SFU_FW_VERSION_INIT_NUM *app_sfu.h* is the only one allowed for installation when the header of the installed image is not valid. This is the case either because no firmware is installed (development phase) or due to an attack attempt. It is important to keep such firmware private as the only purpose of this version is to analyze and repair devices returned from the field.

8.5 How to validate a firmware image

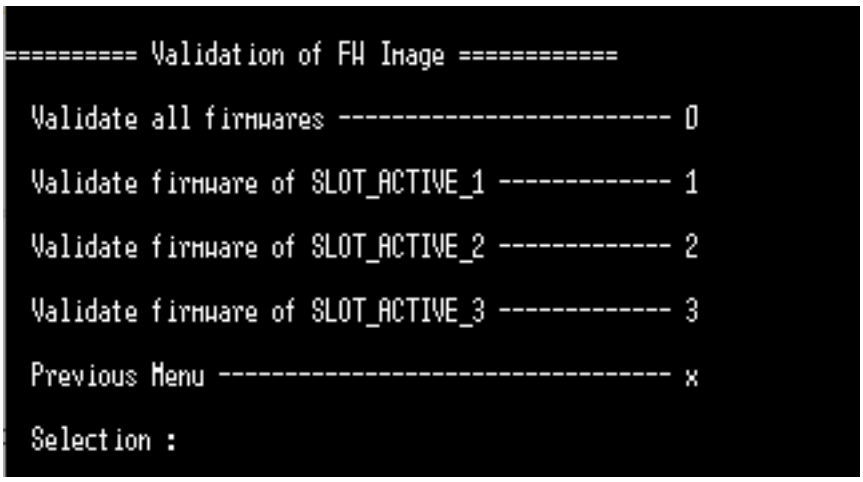
First of all, the `ENABLE_IMAGE_STATE_HANDLING` compilation switch must be defined in SECoreBin, SBSFU, and UserApp IDE configuration.

With STM32CubeIDE, when encountering undefined references to `SE_APP_ValidateFw` and `SE_APP_GetActiveFwState` during compilation, it is crucial to add the missing functions in *STM32CubeIDE/se_interface.txt file* (one per line).

At the first user application start-up, if the execution is correct (for example after self-tests execution) the user application must call a running service `SE_APP_Validate(slot_id)` if available or update dedicated flags in RAM otherwise to validate the firmware image. If not done, a rollback on the previous firmware image is performed by SBSFU at the next reset.

An example is provided in the user application through the menu `FW_VALIDATE_RunMenu()` as shown in *Figure 42*. In a multiple image configuration, the slot identification parameter can be either 1, 2, 3, or 255. The value 255 indicates that all new firmware images are validated through a single request. The objective is to ensure firmware compatibility between all new images in case of interruption during the validation phase.

Figure 42. Validation menu



Caution: This feature can be activated only on a dual-slot configuration example with the swap installation process selected.

9 Revision history

Table 4. Document revision history

Date	Revision	Changes
20-Dec-2017	1	Initial release.
31-Aug-2018	2	Document structure and content entirely updated: – Refocused on the integration topics presented in *Introduction* – Adapted to the asymmetric and symmetric cryptography schemes – Adapted to the single-image and dual-image modes
18-Dec-2018	3	Product scope extended to the STM32F4 Series, STM32F7 Series, and STM32G0 Series: – Updated *Chapter 1: General information*, *Chapter 2: Related documents*, *Section 3.2: Memory mapping definition*, *Section 4.3: Security configuration*, *Section: Figure 15 shows the various security configuration solutions available in file app_sfu.h for the STM32WB Series.*, and *Section 8.1: How to make an application SBSFU compatible* – Added *Chapter 7: Adapting SBSFU* Secure library offer extended to mbedTLS: – Updated *Section 4.1: Features to be configured*
06-Sep-2019	4	Updated *Introduction*. Product scope extended to the STM32H7 Series, STM32G4 Series, STM32L0 Series, STM32L1 Series and STM32WB Series. Updated *Chapter 2: Related documents*. Updated *Section 3.1: Hardware adaptation* Updated *Section 3.2: Memory mapping definition* Modified *Section 3.2.1: SBSFU region definition parameters* and *Section 3.2.2: Firmware image slot definition parameters* Updated *Section 4.1 on page 17* Updated *Chapter 4.3: Security configuration* (Updated figures and added *Figure 18: STM32WB Series security configuration (app_sfu.h)* Added note in *Section 4.2 on page 18*. Modified Option Byte configuration in *Section 4.4: Development or production mode configuration*. Added *Section 5.3: STM32WB Series specificities*, *Section 5.4: KMS specificities* and *Section 5.5: STSAFE-A100 specificities*. Updated *Table 3* in *Section 7.2: Optimizing memory mapping* Added *Section 8.4: How to replace the standalone loader with a BLE OTA loader* and *Section 8.5: How to change the firmware version*.

Table 4. Document revision history (continued)

Date	Revision	Changes
09-Jul-2020	5	Added OTFDEC information in *Section 4.1: Features to be configured* and *Section 4.2: Cryptographic scheme selection* (added one note) Updated *Section 3.2.2: Firmware image slot definition parameters*. Added *Figure 8: Firewall configuration constraint on dual bank products* and *Figure 9: Firewall configuration after bank swap*. Updated *Figure 11: SBSFU specific linker file*, *Figure 12: UserApp specific linker file (NUCLEO-L476RG example)* and *Figure 13: SBSFU configuration*. Updated *Section 4.4: Development or production mode configuration*, *Section 6.2: Memory mapping adaptation*, *Section 7.2: Optimizing memory mapping* Removed *Figure 28 Example of memory mapping optimization on the NUCLEO-G031K8 – 1 image.*
01-Sep-2020	6	Added: – *Section 3.2.4: Multiple image configuration* – *Section 3.3: Dual-core adaptation* – *Section 7.3: How to activate interruption management inside the firewall isolated environment* – *Section 7.4: How to improve boot time* – *Section 8.6: How to validate a firmware image* Updated: – Secure element STSAFE-A100 replaced by STSAFE-A110
22-Jul-2021	7	Added: – *Section 3.2.2: Firmware image slot definition parameters* SFU_IMAGE_OFFSET value for STM32H7 Series – *Section 4.1: Features to be configured* External Flash memory without on-the-fly decryption Updated: – *Section 3.2: Memory mapping definition* and *Section 5.1: Generating a new firmware AES encryption key* references to UM2262 – *Section 5.3: STM32WB Series specificities* with added *Figure 24: Key provisioning* Removed: – Former *Section 8.4 How to replace the standalone loader with a BLE OTA loader*
14-Dec-2021	8	Updated: – *Section 4.1: Features to be configured* for STM32WB5MM-DK Discovery board – *Section 5.3: STM32WB Series specificities* to select boot mode on P-NUCLEO-WB55 Nucleo and STM32WB5MM-DK Discovery boards – *Section 8.5: How to validate a firmware image*

Table 4. Document revision history (continued)

Date	Revision	Changes
05-Dec-2025	9	Updated: – Document title – Reference document titles in *Chapter 2: Related documents* – Introduction of *Figure 19* – Additional constraints in *Section 8.1: How to make an application SBSFU compatible* – P-NUCLEO-WB55, replaced by NUCLEO-WB55RG Added: – Compatible STM32H7 MCUs in *Chapter 2: Related documents* – Compilation guidance with STM32CubeIDE in *Section 8.5: How to validate a firmware image* Minor text edits across the whole document.
17-Apr-2026	10	Added *Section 7.5: Handling ITCM/TCM regions in ELF files*.