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

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-A100 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 SBSFU?
- How to adapt the user’s application?

Note: Throughout this application note, the IAR™ EWARM IDE is used as an example to provide guidelines for project configuration. Secure Boot and Secure Firmware Update applications are referred to as SBSFU.
## Contents

1. General information ........................................... 6

2. Related documents ............................................. 7

3. Porting X-CUBE-SBSFU onto another board .................... 8
   3.1 Hardware adaptation ........................................ 8
   3.2 Memory mapping definition .................................. 9
      3.2.1 SBSFU region definition parameters ..................... 12
      3.2.2 Firmware image slot definition parameters .......... 13
      3.2.3 Project specific linker files ............................ 15

4. SBSFU configuration ............................................ 17
   4.1 Features to be configured ................................... 17
   4.2 Cryptographic scheme selection ............................. 18
   4.3 Security configuration ...................................... 18
   4.4 Development or production mode configuration ............. 21

5. Generating cryptographic key .................................. 23
   5.1 Generating a new firmware AES encryption key ............. 23
   5.2 Generating a new public/private ECDSA pair of keys for firmware verification ............ 23
   5.3 STM32WB Series specificities ............................... 24
   5.4 KMS specificities ........................................... 24
   5.5 STSAFE-A100 specificities .................................. 25

6. Tips for debugging ............................................. 27
   6.1 Compiler optimizations level ................................ 27
   6.2 Memory mapping adaptation .................................. 27
   6.3 Debugging SECoreBin ........................................ 28

7. Adapting SBSFU ................................................ 30
   7.1 Implementing a new cryptographic scheme for SBSFU .... 30
   7.2 Optimizing memory mapping ................................. 31
8 Adapting the user application ........................................... 33
  8.1 How to make an application SBSFU compatible .................. 33
  8.2 Use of the Flash memory to store user data ....................... 36
  8.3 Changing the firmware download function in the user application ...... 37
  8.4 How to replace the standalone loader with a BLE OTA loader ........ 37
  8.5 How to change the firmware version .................................. 39

9 Revision history ............................................................... 40
List of tables

Table 1. List of acronyms ......................................................... 6
Table 2. List of terms .............................................................. 6
Table 3. SBSFU code size reduction .......................................... 31
Table 4. Document revision history ......................................... 40
List of figures

Figure 1. SBSFU project structure ........................................... 8
Figure 2. Memory mapping example (NUCLEO-L476RG) ...................... 9
Figure 3. Linker file architecture ............................................ 10
Figure 4. Mapping constraints with MPU isolation (NUCLEO-G071RB example) .... 11
Figure 5. Mapping constraints for user application execution ............... 11
Figure 6. SBSFU regions (mapping_sbsfu.icf from NUCLEO-L476RG) ........ 12
Figure 7. Firmware image slot definitions (mapping_fwimg.icf from NUCLEO-L476RG) .... 13
Figure 8. Firewall configuration constraint on dual bank products .......... 14
Figure 9. Firewall configuration after bank swap ........................... 14
Figure 10. SCoreBin specific linker file ...................................... 15
Figure 11. SBSFU specific linker file ......................................... 16
Figure 12. UserApp specific linker file (NUCLEO-L476RG example) .......... 16
Figure 13. SBSFU configuration .............................................. 17
Figure 14. Switching the cryptographic scheme .............................. 18
Figure 15. STM32L4 Series and STM32L0 Series security configuration (app_sfu.h) ...... 19
Figure 16. STM32F4 Series, STM32F7 Series and STM32L1 Series security configuration (app_sfu.h) .... 19
Figure 17. STM32G0 Series, STM32G4 Series and STM32H7 Series security configuration (app_sfu.h) .... 20
Figure 18. STM32WB Series security configuration (app_sfu.h) .............. 20
Figure 19. Option Bytes management ........................................ 22
Figure 20. New firmware encryption key ..................................... 23
Figure 21. New private/public keys .......................................... 24
Figure 22. KMS specificities .................................................. 25
Figure 23. STSAFE-A100 pairing keys ....................................... 26
Figure 24. Compiler optimizations .......................................... 27
Figure 25. Memory mapping adaptations .................................... 28
Figure 26. Checking the WRP protection .................................... 28
Figure 27. Debugging inside SCoreBin ....................................... 29
Figure 28. User’s own cryptographic scheme implementation .............. 30
Figure 29. Example of memory mapping optimization on NUCLEO-G071RB – 2 images .......... 32
Figure 30. Vector table position update (NUCLEO-L476RG example) ....... 33
Figure 31. User application binary file length .............................. 34
Figure 32. IDE adaptations .................................................... 34
Figure 33. Free Flash pages (example of NUCLEO-L476RG) ................ 36
Figure 34. UserApp firmware download overview ......................... 37
Figure 35. BLE OTA loader replacement .................................... 38
Figure 36. Firmware version change ........................................ 39
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**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced encryption standard</td>
</tr>
<tr>
<td>DAP</td>
<td>Debug access port</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic curve digital signature algorithm</td>
</tr>
<tr>
<td>GCM</td>
<td>AES Galois/counter mode</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware abstraction layer</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
</tr>
<tr>
<td>FWALL</td>
<td>Firewall</td>
</tr>
<tr>
<td>MPU</td>
<td>Memory protection unit</td>
</tr>
<tr>
<td>OTFDEC</td>
<td>On-the-fly decryption</td>
</tr>
<tr>
<td>PEM</td>
<td>Privacy enhanced mail</td>
</tr>
<tr>
<td>PCROP</td>
<td>Proprietary code read out protection</td>
</tr>
<tr>
<td>RDP</td>
<td>Readout device protection</td>
</tr>
<tr>
<td>SB</td>
<td>Secure Boot</td>
</tr>
<tr>
<td>SE</td>
<td>Secure Engine</td>
</tr>
<tr>
<td>SFU</td>
<td>Secure Firmware Update</td>
</tr>
<tr>
<td>SBSFU</td>
<td>Secure Boot and Secure Firmware Update</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver/transmitter</td>
</tr>
<tr>
<td>WRP</td>
<td>Write protection</td>
</tr>
</tbody>
</table>

**Table 2. List of terms**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firmware image</td>
<td>A binary image (executable) run by the device as user application.</td>
</tr>
<tr>
<td>Firmware header</td>
<td>Bundle of meta-data describing the firmware image to be installed. It contains firmware information and cryptographic information.</td>
</tr>
<tr>
<td>mbedTLS</td>
<td>mbed implementation of the TLS and SSL protocols and the respective cryptographic algorithms.</td>
</tr>
<tr>
<td>sfb file</td>
<td>Binary file packing the firmware header and the firmware image.</td>
</tr>
</tbody>
</table>

The X-CUBE-SBSFU Secure Boot and Secure Firmware Update Expansion Package runs on STM32 32-bit microcontrollers based on the Arm\textsuperscript{\textregistered}(a) Cortex\textsuperscript{\textregistered}-M processor.
2 Related documents

1. Getting started with STM32CubeH7 for STM32H7 Series (UM2204)
2. Getting started with STM32CubeG4 for STM32G4 Series (UM2492)
3. Getting started with STM32CubeL0 for STM32L0 Series (UM1754)
4. Getting started with STM32CubeL1 MCU Package for STM32L1 Series (UM1802)
5. Getting started with STM32CubeWB for STM32WB Series (UM2550)
6. Getting started with STM32CubeL4 for STM32L4 Series and STM32L4+ Series user manual (UM1860)
7. Getting started with STM32CubeF4 MCU Package for STM32F4 Series user manual (UM1730)
8. Getting started with STM32CubeF7 MCU Package for STM32F7 Series user manual (UM1891)
9. Getting started with STM32CubeG0 for STM32G0 Series user manual (UM2303)
10. Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package user manual (UM2262)
11. Development guidelines for STM32Cube Expansion Packages user manual (UM2285)
12. Development checklist for STM32Cube Expansion Packages user manual (UM2312)
13. STM32CubeProgrammer software description user manual (UM2237)
14. STM32F3 Series, STM32F4 Series, STM32L4 Series and STM32L4+ Series Cortex®-M4 programming manual (PM0214)
15. STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual (PM0253)
16. STM32L0 Series and STM32G0 Series Cortex®-M0+ programming manual (PM0223)
17. Authentication, state-of-the-art security for peripherals and IoT devices datasheet for STSAFE-A100 (DS12911).

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.
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 in order to adapt X-CUBE-SBSFU to another board:

1. GPIO configuration for UART communication with the host PC (in file `sfu_low_level.h`)
2. Flash configuration: NUCLEO-L432KC gives an example of single-bank Flash interface whereas NUCLEO-L476RG is dual-bank based (in file `sfu_low_level.c`)
3. Button configuration: NUCLEO-L476RG gives an example based on the push button whereas NUCLEO-L432KC simulates a virtual button with a GPIO (in file `app_hw.h`)
4. Tamper GPIO pin configuration (in file `sfu_low_level_security.h`)
5. DAP - Debug port configuration (in file `sfu_low_level_security.h`)
6. I²C bus configuration for communication with STSAFE-A100 (in file `stsafea_service_interface.c` of B-L475E-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](image-url)
3.2 Memory mapping definition

As already highlighted in the X-CUBE-SBSFU user manual (refer to [5]), 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
- Slot #0: this slot contains active firmware (firmware header + firmware)
- Slot #1: this slot stores downloaded firmware (firmware header + encrypted firmware) to be installed at next reboot
- Swap area: Flash area used to swap the content of Slot #0 and Slot #1 during the installation process

Figure 2 presents the Flash memory mapping illustrated by the NUCLEO-L476RG example.
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 Slot #0, Slot #1, 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.

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 following constraints:

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

The mapping constraints with MPU isolation are illustrated in Figure 4.
Another typical use case is the MPU configuration of the Slot #0 region in order to authorize user application execution. Figure 5: Mapping constraints for user application execution shows how to respect the MPU constraints on NUCLEO-L073RZ.
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 (mapping_sbsfu.icf from NUCLEO-L476RG)*

```c
/* SE Code region protected by firewall */
define exported symbol __CFRESET_SE_Code_region_KOM_start__ = 0x00000000;
define exported symbol __CFRESET_SE_Code_region_KOM_end__  = __CFRESET_SE_Code_region_KOM_start__ + 4;

/* SE Key region protected by firewall */
define exported symbol __CFRESET_SE_Key_region_KOM_start__ = __CFRESET_SE_CallGateRegion_KOM_end__ + 1;
define exported symbol __CFRESET_SE_Key_region_KOM_end__  = __CFRESET_SE_CallGateRegion_KOM_start__ + 0xFFF;

/* SE startup: call before enabling firewall */
define exported symbol __CFRESET_SE_Startup_region_KOM_start__ = __CFRESET_SE_Key_region_KOM_end__ + 1;
define exported symbol __CFRESET_SE_Code_nKey_region_KOM_start__ = __CFRESET_SE_Startup_region_KOM_start__ + 0x100;
define exported symbol __CFRESET_SE_Code_nKey_region_KOM_end__  = __CFRESET_SE_Startup_region_KOM_start__ + 0x6000;
```

- Offsets allow auto-adjustment when updating a size: SBSFU code setting the protections takes it into account.
- Absolute values used in case of constraints (as for MPU configuration on STM32F4, STM32F7, STM32G0, STM32G4, STM32L1 and STM32H7).
- Region start addresses must be 256-byte aligned (except SE_CallGate).

/* SE IF ROM used to locate Secure Engine interface code out of firewall */
define exported symbol __CFRESET_SE_IF_region_KOM_start__ = __CFRESET_SE_Code_region_KOM_end__ + 1;
define exported symbol __CFRESET_SE_IF_region_KOM_end__  = __CFRESET_SE_Code_region_KOM_start__ + 0xFFF;

/* SBSFU Code region */
define exported symbol __CFRESET_SE_Code_region_KOM_start__ = __CFRESET_SE_IF_region_KOM_end__ + 1;
define exported symbol __CFRESET_SE_Code_region_KOM_end__  = 0x60000000;
```
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 (mapping_fwimg.icf from NUCLEO-L476RG)

```c
/* Slots Regions must be aligned on the Flash sector size */

/* swap region 8 Kbytes */
define exported symbol __ICEXIT_region_SWAP_start__ = 0x0080 F2800;  
define exported symbol __ICEXIT_region_SWAP_end__ = 0x0080 F47FF;

/* slot 0 region 0x70000 = 57 sections of 8 Kbytes (456 Kbytes) */
define exported symbol __ICEXIT_region_SLOT_0_start__ = 0x0080 0000;  
define exported symbol __ICEXIT_region_SLOT_0_end__ = 0x0080 F7FF;

/* slot 1 region 0x70000 = 57 sections of 8 Kbytes (456 Kbytes) */
define exported symbol __ICEXIT_region_SLOT_1_start__ = 0x0080 E000;  
define exported symbol __ICEXIT_region_SLOT_1_end__ = 0x0080 FFFF;
```

The compliance with SBSFU constraints requires that the following conditions are met:

- Slots areas must be aligned on the Flash 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 SLOT_0 must be a multiple of the SWAP size
- The sizes SLOT_0 and SLOT_1 must be equal, except when using partial update feature

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

- Firewall code segment must be in bank1, firewall non-volatile data (including header of Slot #0) segment must be in bank2.
- The non-volatile data segment must overlap the firewall code segment in order 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.
For the STM32G0 Series, STM32G4 Series and STM32H7 Series, one constraint exists: the Slot #0 header must be mapped just after the SBSFU code in order 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: 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 sector size.
- For the STM32G4 Series: 4096 bytes. The secure user memory end address is aligned on the Flash sector size.
- For the STSAFE-A variant: 2048 bytes.

The image header has a 2048-byte length in order to include X509 certificates.

**Note:** For series with MPU-based isolation or firewall-based isolation, the MPU constraint on Slot#0 configuration must be verified as illustrated on Figure 5.

### 3.2.3 Project specific linker files

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

**Figure 10. SECoreBin specific linker file**

don't initialize { section .noinit, section BOOTINFO_DATA};
define block SE_VECTOR with alignment = 512 {readonly section .intvec};

/***************************************************/
*/
placement instructions
/
***************************************************/
place at address mem:__CFEDIT_SE_CallGate_region_ROM_start__ {readonly section .SE_CallGate_Code};

place in SE_Entry Secure_ROM Region {readonly section .SE_KeyConst};
place in SE_Key_ROM_region {readonly section .SE_Key_Data};

place at address mem:__CFEDIT_SE_Startup_region_ROM_start__ {readonly section .SE_Startup_Code};
place in SE_ROM_region {readonly, block SE_VECTOR};
place in SE_SHRAM_region {rewrite, section BOOTINFO_DATA};
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**

```
/*-----------------------------------------------*/
/* placement instructions */
/*-----------------------------------------------*/
place at address:  ICEDIT_invec_start (readonly section_invec);
place at address:  ICEDIT_SE_CallGate_region_ROM_start (readonly section SE_CORE_Bin);
place in SE_IF_ROM_region (section SE_IF_Code);
place in SB_RAM1_region (readonly);
place in SB_RAM1_region (readwrite, block CSTACK, block HEAP);
```

UserApp must be configured to run in Slot #0 (SLOT_0 start address + 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 ICEDIT_invec_start = ICEDIR_region SLOT_0_start + 512;

/* Memory Regions */
define symbol ICEDIR_region_ROM_start = ICEDIR_region SLOT_0_start;
define symbol ICEDIR_region_ROM_end = ICEDIR_region SLOT_0_end;

define symbol ICEDIR_region_RAM_start = ICEDIR_region SRAM1_end + 1;

to make sure the binary size is a multiple of the AES block size (16 bytes) and L4 flash writing unit (8 bytes)
#define aes_block_padding with alignment=16

if defined "Force Alignment";
pad_to 16;

place in ROM_region (readonly, last section aes_block_padding);
```

1. Depends on the ST32 microcontroller Series.
4 SBSFU configuration

4.1 Features to be configured

X-CUBE-SBSFU supports:

- 2 modes of operation: dual and single image
- 3 cryptographic schemes using symmetric and asymmetric cryptographic operations
- 2 cryptographic middleware:
  - STMicroelectronics middleware: X-CUBE-CRYPTOLIB library integrated into the 1_Images and 2_Images variants.
  - Third-party middleware: mbedTLS (open-source code) cryptographic services. Examples are provided for the 32L496GDISCOVERY, B-L475E-IOT01A, 32F413HDISCOVERY and 32F769IDISCOVERY boards in the 2_Images_OSC variant.
- STSAFE-A100 secure element used to host X509 certificates and keys. An example is provided for the B-L475E-IOT01A board in the 2_Images_STSAFE variant.
- KMS middleware. An example is provided for the B-L475E-IOT01A board 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.

The configuration possibilities go beyond these options through compilation switches:

- Local loader can be removed to reduce the memory footprint (dual image only)
- Verbose switch can be activated to make the debug 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 the debug easier

Figure 13 presents the SBSFU configuration solutions with the related files and compilation switches.

![Figure 13. SBSFU configuration](image-url)
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 by means of the SECBOOT_CRYPTO_SCHEME compilation switch as depicted in Figure 14.

![Figure 14. Switching the cryptographic scheme](image)

Note: For the STSAFE variant, 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 to protect secrets against both outer and inner attacks.

STM32 security peripherals can be deactivated independently as per user’s decision in order to achieve a different protection level (for example, for 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 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 debug easier.

*Figure 15* shows the various security configuration solutions available in file `app_sfu.h` for the STM32L4 Series and STM32L0 Series.
Figure 15. STM32L4 Series and STM32L0 Series security configuration (app_sfu.h)

```c
// #define SECBOOT_DISABLE_SECURITY_IPS /*!< Disable all security IPs at once when activated */
#endif(SECBOOT_DISABLE_SECURITY_IPS)/*
#define SFU_WRP_PROTECT_ENABLE
#define SFU_RDP_PROTECT_ENABLE
#define SFU_PCRP_PROTECT_ENABLE
#define SFU_FWALL_PROTECT_ENABLE
#define SFU_TAMPER_PROTECT_ENABLE
#define SFU_DAP_PROTECT_ENABLE
#define SFU_DMA_PROTECT_ENABLE
#define SFU_WDG_PROTECT_ENABLE
#define SFU_MPU_PROTECT_ENABLE
#define SFU_MPU_USERAPP_ACTIVATION
#endif
```

Figure 16 shows the various security configuration solutions available in file app_sfu.h for the STM32F4 Series, STM32F7 Series and STM32L1 Series.

Figure 16. STM32F4 Series, STM32F7 Series and STM32L1 Series security configuration (app_sfu.h)

```c
// #define SECBOOT_DISABLE_SECURITY_IPS /*!< Disable all security IPs at once when activated */
#endif(SECBOOT_DISABLE_SECURITY_IPS)/*
#define SFU_WRP_PROTECT_ENABLE
#define SFU_RDP_PROTECT_ENABLE
#define SFU_PCRP_PROTECT_ENABLE
#define SFU_FWALL_PROTECT_ENABLE
#define SFU_TAMPER_PROTECT_ENABLE
#define SFU_DAP_PROTECT_ENABLE
#define SFU_DMA_PROTECT_ENABLE
#define SFU_WDG_PROTECT_ENABLE
#define SFU_MPU_PROTECT_ENABLE
#define SFU_MPU_PROTECT
#endif
```
Figure 17 shows the various security configuration solutions available in file *app_sfu.h* for the STM32WB Series.

**Figure 17. STM32G0 Series, STM32G4 Series and STM32H7 Series security configuration (app_sfu.h)**

Figure 18 shows the various security configuration solutions available in file *app_sfu.h* for the STM32WB Series.

**Figure 18. 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 in order to enable IDE debug 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 Chapter 5: Generating cryptographic key)

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.

Read Protection Level 2 is mandatory to achieve the highest level of protection and to implement a Root of Trust. It is 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, few other Option Bytes must be configured such as:

- BFB2 disabled for STM32L4 Series and STM32L0 Series devices with dual-bank Flash
- nDBANK enabled for STM32F7 Series
- nBFB2 enabled for STM32L1 Series
- BOOT_LOCK enabled for STM32G0Series and STM32G4 Series
- DBANK disabled on STM32G4 Series

Caution: Option Bytes must be configured to the production mode values by means of 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 19 shows how Option Bytes are managed at SBSFU startup:

**Figure 19. Option Bytes management**
5 Generating 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 [5] for a detailed description of the build process).

*Figure 20* shows the few steps to modify the firmware encryption key:

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

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 (`ECCKEY.txt`) is changed.

*Figure 21* shows the few steps to modify the private and public keys for ECDSA asymmetric cryptography firmware verification:

1. Change the key value in file `ECCKEY.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
5.3 **STM32WB Series specificities**

For STM32WB Series, the AES encryption key is not processed through the `prebuild.bat` script but provisioned into the M0+ core. Provisioning process is described into `SECoreBin/readme.txt`.

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 22* shows an example of the firmware encryption key modification:
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 22. KMS specificities**

---

**5.5 STSAFE-A100 specificities**

As explained in the Appendix G of the UM2262, STM32 and STSAFE-A100 must be provisioned with pairing keys.

STSAFE-A100 provisioning process is described in STSAFE_Provisioning/readme.txt.

Figure xx shows an example of pairing key provisioning:
1. STSAFE-A100 provisioning with default pairing keys
2. Update STSAFEPAIRING_keys.bin accordingly
3. Compile SECoreBin: prebuild.bat is executed and se_key.s is generated.
Figure 23. STSAFE-A100 pairing keys

STSAFE-A100 provisioning with default keys

update file with default keys value

STSAFE_PROVISION_keybin

DE_ReceivePairing

set_key.s
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 24. Compiler optimizations

6.2 Memory mapping adaptation

When changing the compiler optimizations level or activating the development mode with 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 25 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
The impact of memory mapping adaptation on security peripheral configurations must be checked despite the fact that it is automatically computed. For example, check the WRP configuration using STM32CubeProgrammer (STM32CubeProg) as shown in Figure 26.

6.3 Debugging SECoreBin

To debug inside SECoreBin, SBSFU projects option must be changed to load SECoreBin symbols. This is performed in the debugger menu as presented in Figure 27:

- Browse to select file *Project.out*
- Set Offset to 0
- Check the *Debug info only* box
Figure 27. 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 18). It is also possible to extend the package with the user’s own cryptographic scheme.

In order to implement a new cryptographic scheme for SBSFU, follow the steps illustrated in Figure 28 and described below.

Figure 28. User’s own cryptographic scheme implementation

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 code running on 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 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; 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 in order to maximize the size of the user application slot. Some of these options are summarized in *Table 3*.

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

The total gain depends on the mapping constraints described in *Section 3.2: Memory mapping definition on page 9*.

As an example, *Figure 29* highlights the mapping modifications to be done. Starting from 2 images with symmetric crypto scheme, the `SFU_DEBUG_MODE` and `SECBOOT_USE_LOCAL_LOADER` switches are disabled, resulting in a 16-Kbyte increase of the user application size.
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.
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 in order to allow the application to run in Slot #0:

- Code section starting by the vector table must be configured to run from Slot #0, just after the image header: __ICFEDIT_region_SLOT_0_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 9 for more details on memory constraints.

Then, during system initialization, VTOR must be set to the new location of vector table as shown in Figure 30.

For user application encryption, the user application binary file length must be a multiple of 16 bytes. Figure 31 shows how to update the linker file to verify this constraint.
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 postbuild.bat to generate UserApp.sfb and SBFU_UserApp.bin
4. Integrate se_interface_appli.o to access Secure Engine runtime services if any

As explained in UM2262, there are some additional constraints depending on the STM32 series:
- STM32F4 Series, STM32F7 Series and STM32L1 Series: MPU-based Secure Engine isolation relies fully on the fact that 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 is no Secure Engine runtime services available for the user application.

Note: IWDG is started during SBSFU execution. It must be refreshed within a 6-second period.
8.2 Use of the Flash memory to store user data

The storage of user data in Flash pages (or Flash sectors) is possible with some restrictions:
- Out of the SBSFU code area
- Not in the images slots (Slot #0, Slot #1)
- Not in the Swap area

*Figure 33* provides a memory-mapping example based on the NUCLEO-L476RG where the Flash is available from page 489 to page 511 for the user to store data, install a file system or emulate an EEPROM.

*Figure 33. Free Flash pages (example of NUCLEO-L476RG)*
8.3 Changing the firmware download function in the user application

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

A sample code based on the YMODEM protocol over UART is available in X-CUBE-SBSFU UserApp project. The download procedure is located in file `fw_update_app.c` as illustrated in Figure 34.

Figure 34. UserApp firmware download overview

8.4 How to replace the standalone loader with a BLE OTA loader

For STM32WB Series, an example of BLE OTA loader application is provided in STM32WB cube package.

Figure 35 shows a list of rules to be followed when replacing the standalone loader:
1. Integrate loader project inside SBSFU common mapping definition
2. Downloaded firmware storage must take into account partial image offset
3. When the new firmware is downloaded, trigger the installation at next reset by writing header in the swap area
4. SECBOOT_BYPASS_MODE_ENABLED switch can be activated if the loader is designed to update BLE stack through M0+ core.
Adapting the user application

Figure 35. BLE OTA loader replacement

1. Integrate loader project inside SBSFU mapping
2. Take into account partial firmware offset for firmware storage in Slot#1
3. Trigger installation at next reset
4. Bypass mode
8.5 How to change the firmware version

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

**Figure 36. Firmware version change**
9 Revision history

Table 4. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-Dec-2017</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Document structure and content entirely updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Refocused on the integration topics presented in Introduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Adapted to the asymmetric and symmetric cryptography schemes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Adapted to the single-image and dual-image modes</td>
</tr>
<tr>
<td>31-Aug-2018</td>
<td>2</td>
<td>Product scope extended to the STM32F4 Series, STM32F7 Series, and STM32G0 Series:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– 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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Added Chapter 7: Adapting SBSFU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure library offer extended to mbedTLS:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Updated Section 4.1: Features to be configured</td>
</tr>
<tr>
<td>18-Dec-2018</td>
<td>3</td>
<td>Updated Introduction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product scope extended to the STM32F4 Series, STM32F7 Series, and STM32G0 Series:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Updated Chapter 2: Related documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 3.1: Hardware adaptation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 3.2: Memory mapping definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Section 3.2.1: SBSFU region definition parameters and Section 3.2.2: Firmware image slot definition parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 4.1 on page 17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Chapter 4.3: Security configuration (updated figures and added Figure 18: STM32WB Series security configuration (app_sfu.h))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added note in Section 4.2 on page 18.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Option Byte configuration in Section 4.4: Development or production mode configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Section 5.3: STM32WB Series specificities, Section 5.4: KMS specificities and Section 5.5: STSAFE-A100 specificities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Table 3 in Section 7.2: Optimizing memory mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>06-Sep-2019</td>
<td>4</td>
<td>Updated Introduction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product scope extended to the STM32H7 Series, STM32G4 Series, STM32L0 Series, STM32L1 Series and STM32WB Series.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Chapter 2: Related documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 3.1: Hardware adaptation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 3.2: Memory mapping definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Section 3.2.1: SBSFU region definition parameters and Section 3.2.2: Firmware image slot definition parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 4.1 on page 17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Chapter 4.3: Security configuration (updated figures and added Figure 18: STM32WB Series security configuration (app_sfu.h))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added note in Section 4.2 on page 18.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified Option Byte configuration in Section 4.4: Development or production mode configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Section 5.3: STM32WB Series specificities, Section 5.4: KMS specificities and Section 5.5: STSAFE-A100 specificities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Table 3 in Section 7.2: Optimizing memory mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision</td>
<td>Changes</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>20-Jan-2020</td>
<td>5</td>
<td>Added OTFDEC information in Section 4.1: Features to be configured and Section 4.2: Cryptographic scheme selection (added one note)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 3.2.2: Firmware image slot definition parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Figure 8: Firewall configuration constraint on dual bank products and Figure 9: Firewall configuration after bank swap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Figure 11: SBSFU specific linker file, Figure 12: UserApp specific linker file (NUCLEO-L476RG example) and Figure 13: SBSFU configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Section 4.4: Development or production mode configuration, Section 6.2: Memory mapping adaptation, Section 7.2: Optimizing memory mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed Figure 28 Example of memory mapping optimization on NUCLEO-G031K8 – 1 image.</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

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

© 2020 STMicroelectronics – All rights reserved