**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 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.

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

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

2. **Related documents** ............................................ 8

3. **Porting X-CUBE-SBSFU onto another board** ................. 9
   3.1 Hardware adaptation ........................................ 9
   3.2 Memory mapping definition ................................ 10
      3.2.1 SBSFU region definition parameters ............... 13
      3.2.2 Firmware image slot definition parameters ....... 14
      3.2.3 Project-specific linker files ....................... 16
      3.2.4 Multiple images configuration .................. 17
   3.3 Dual-core adaptation ..................................... 18

4. **SBSFU configuration** ........................................ 20
   4.1 Features to be configured ................................ 20
   4.2 Cryptographic scheme selection ....................... 21
   4.3 Security configuration ................................... 22
   4.4 Development or production mode configuration ....... 24

5. **Generating cryptographic key** .............................. 26
   5.1 Generating a new firmware AES encryption key .......... 26
   5.2 Generating a new public/private ECDSA pair of keys for firmware verification ................................. 26
   5.3 STM32WB Series specificities ............................. 27
   5.4 KMS specificities ......................................... 27
   5.5 STSAFE-A110 specificities ............................... 28

6. **Tips for debugging** .......................................... 29
   6.1 Compiler optimizations level ............................. 29
   6.2 Memory mapping adaptation .............................. 29
   6.3 Debugging SECoreBin .................................... 30

7. **Adapting SBSFU** ............................................ 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Implementing a new cryptographic scheme for SBSFU</td>
<td>32</td>
</tr>
<tr>
<td>7.2</td>
<td>Optimizing memory mapping</td>
<td>34</td>
</tr>
<tr>
<td>7.3</td>
<td>How to activate interruption management inside the firewall isolated environment</td>
<td>36</td>
</tr>
<tr>
<td>7.4</td>
<td>How to improve boot time</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>Adapting the user application</td>
<td>38</td>
</tr>
<tr>
<td>8.1</td>
<td>How to make an application SBSFU compatible</td>
<td>38</td>
</tr>
<tr>
<td>8.2</td>
<td>Use of the Flash memory to store user data</td>
<td>41</td>
</tr>
<tr>
<td>8.3</td>
<td>Changing the firmware download function in the user application</td>
<td>42</td>
</tr>
<tr>
<td>8.4</td>
<td>How to replace the standalone loader with a BLE OTA loader</td>
<td>43</td>
</tr>
<tr>
<td>8.5</td>
<td>How to change the firmware version</td>
<td>44</td>
</tr>
<tr>
<td>8.6</td>
<td>How to validate a firmware image</td>
<td>44</td>
</tr>
<tr>
<td>9</td>
<td>Revision history</td>
<td>46</td>
</tr>
</tbody>
</table>
## List of tables

Table 1. List of acronyms ............................................................... 6
Table 2. List of terms ................................................................. 6
Table 3. SBSFU code-size reduction ............................................... 34
Table 4. Document revision history ............................................... 46
List of figures

Figure 1. SBSFU project structure ......................................................... 9
Figure 2. Memory mapping example (NUCLEO-L476RG) .......................... 10
Figure 3. Linker file architecture ......................................................... 11
Figure 4. Mapping constraints with MPU isolation (NUCLEO-G071RB example) .................. 12
Figure 5. Mapping constraints for user application execution ......................... 12
Figure 6. SBSFU regions (mapping_sbsfu.icf from NUCLEO-L476RG) .............. 13
Figure 7. Firmware image slot definitions (mapping_fwimg.icf from NUCLEO-L476RG) .... 14
Figure 8. Firewall configuration constraint on dual bank products ..................... 15
Figure 9. Firewall configuration after bank swap ...................................... 15
Figure 10. SECoreBin specific linker file .............................................. 16
Figure 11. SBSFU specific linker file .................................................... 16
Figure 12. UserApp specific linker file (NUCLEO-L476RG example) ............... 17
Figure 13. Multiple images configuration ............................................. 18
Figure 14. STM32H7 Series dual-core adaptation ..................................... 19
Figure 15. SBSFU configuration .......................................................... 21
Figure 16. Switching the cryptographic scheme ...................................... 21
Figure 17. STM32L4 Series and STM32L0 Series security configuration (app_sfu.h) .... 22
Figure 18. STM32F4 Series, STM32F7 Series and STM32L1 Series security configuration (app_sfu.h) .............. 22
Figure 19. STM32G0 Series, STM32G4 Series, and STM32H7 Series security configuration (app_sfu.h) ......... 23
Figure 20. STM32WB Series security configuration (app_sfu.h) .................... 24
Figure 21. Option Bytes management .................................................... 25
Figure 22. New firmware encryption-key .............................................. 26
Figure 23. New private/public keys ..................................................... 27
Figure 24. KMS specificities ............................................................. 28
Figure 25. STSAFE-A110 pairing keys ............................................... 28
Figure 26. Compiler optimizations ..................................................... 29
Figure 27. Memory mapping adaptations ............................................. 30
Figure 28. Checking the WRP protection .............................................. 30
Figure 29. Debugging inside SECoreBin .............................................. 31
Figure 30. User’s own cryptographic scheme implementation ....................... 32
Figure 31. Example of memory mapping optimization on NUCLEO-G071RB – 2 images .... 35
Figure 32. IDE adaptations ............................................................... 36
Figure 33. Boot time ........................................................................... 37
Figure 34. Vector table position update (NUCLEO-L476RG example) ............. 38
Figure 35. User application binary file length ...................................... 39
Figure 36. IDE adaptations ............................................................... 39
Figure 37. Free Flash pages (example of NUCLEO-L476RG) ..................... 41
Figure 38. UserApp firmware download overview .................................. 42
Figure 39. BLE OTA loader replacement ............................................. 43
Figure 40. Firmware version change .................................................. 44
Figure 41. Validation menu ............................................................... 45
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 readout 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 a 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{(a)} Cortex\textsuperscript{(a)}-M processor.

\textsuperscript{a} Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.
2 Related documents

1. User manual Getting started with STM32CubeH7 for STM32H7 Series (UM2204)
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 for STM32L4 Series and STM32L4+ Series (UM1860)
7. User manual Getting started with STM32CubeF4 MCU Package for STM32F4 Series (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)
12. User manual Development checklist for STM32Cube Expansion Packages (UM2312)
14. STM32 Cortex®-M4 MCUs and MPUs programming manual (PM0214)
15. STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual (PM0253)
16. Cortex®-M0+ programming manual for STM32L0, STM32G0, STM32WL and STM32WB Series (PM0223)
17. Data sheet for STSAFE-A110 Authentication, state-of-the-art security for peripherals and IoT devices (DS12911)
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 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-A110 (in file `stsafea_service_interface.c` 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

Platform hardware configuration such as user button
Platform-agnostic part: this is the applicative part of the bootloader and firmware image management procedures
Platform-dependent part: low-level services

Flash modification must also be reported into SECoreBin and UserApp low-level interfaces
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
- Active slot: this slot contains active firmware (firmware header + firmware)
- Download slot: this slot stores downloaded firmware (firmware header + encrypted firmware) to be installed at 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 (SECORE Bin, 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](image)

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.
Another typical use case is the MPU configuration of the active-slot region 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)*

```
/* SE Code region protected by firewall */
define exported symbol _ICFEDIT_SE_Code_region_ROM_start_   = 0x00000020;
define exported symbol _ICFEDIT_SE_CallGate Region_ROM_start_ = _ICFEDIT_SE_Code_region_ROM_start_ + 4;
define exported symbol _ICFEDIT_SE_CallGate Region_ROM_end_   = _ICFEDIT_SE_Code_region_ROM_start_ + 0x0FF;
/* SE key region protected by firewall */
define exported symbol _ICFEDIT_SE_Key_region_ROM_start_     = _ICFEDIT_SE_CallGate Region_ROM_end_ + 1;
define exported symbol _ICFEDIT_SE_Key_region_ROM_end_       = _ICFEDIT_SE_Key_region_ROM_start_ + 0x0FF;
/* SE Startup: call before enabling firewall */
define exported symbol _ICFEDIT_SE_Startup_region_ROM_start_ = _ICFEDIT_SE_Key_region_ROM_end_ + 1;
define exported symbol _ICFEDIT_SE_Code_region_ROM_start_    = _ICFEDIT_SE_Startup_region_ROM_start_ + 0x1000;
define exported symbol _ICFEDIT_SE_Code_region_ROM_end_      = _ICFEDIT_SE_Startup_region_ROM_start_ + 0x0FFFF;
```

-Offsets allow auto-adjustment when updating a size: SBSFU code setting the protections takes it into account.
-It is user’s responsibility to verify the protection during product validation.
-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).
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)*

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 of active and download slots must be a multiple of the *SWAP* size.
- The sizes of active and download slots must be equal, except when using partial update feature.

In some configurations (external Flash with OTFDEC, multiple images 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.
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: 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 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 critical data such as the secrets as illustrated in Figure 10.

**Figure 10. SECoreBin specific linker file**

```c
14 do not initialize { section .noinit, section BOOTINFO_DATA};
15 define block SE_VECTORS with alignment = 512 {readonly section .intvec1;}
16 */
17 /******************SE_VECTORS with alignment = 512 {readonly section .intvec 1;}*/
18 /
19 */
20 place at address mem: _ICFEDIT_SE_CallGate_region_ROM_start {readonly section .SE_CallGate_Code};
21 place at address mem: _ICFEDIT_SE_Key_region_ROM_start {readonly section .SE_Key_Data};
22 place at address mem: _ICFEDIT_SE_BootRegion_ROM_start {readonly section .SE_BootRegion_Code};
23 place in SE_ROM region (readonly, block SEVECTOR);
24 place in SE_ROM region (readwrite, 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**

```c
1/*
2 /*******************************************/
3 /* placement instructions */
4 /*******************************************/
5 place at address mem: _ICFEDIT_intvec_start {readonly section intvec};
6 place at address mem: _ICFEDIT_SE_CallGate_region_ROM_start {readonly section SE_CORE_Bin};
7 place in SE_IF_ROM region (section SE_IF_Code);
8 place in SE_ROM region (readonly);
9 place in SE_ROM region (readwrite, block CSTACK, block HEAP);
```
UserApp must be configured to run in the active slot (slot active 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)

```
13 define exported symbol __ICEDIT_intvec_start__ = __ICEDIT_SLOT_Active_i_start__ + 32;
14
15 /*Memory Regs**/
16 define symbol __ICEDIT_region_ROM_start__ = __ICEDIT_intvec_start__;
17 define symbol __ICEDIT_region_ROM_end__ = __ICEDIT_SLOT_Active_i_end__;
18
19 define symbol __ICEDIT_region_RAM_start__ = __ICEDIT_region_ROM_end__ + 1;
20 define symbol __ICEDIT_region_RAM_end__ = 0x20017FF;

21 /* to make sure the binary size is a multiple of the AES block size (16 bytes) and let Flash writing unit (8 bytes) */
22 define root section aes_block_padding with alignment=16
23
24 //Dataset "Force Alignment":
25 pad_to 16;
26
27 1;
28
29 place in ROM_region { readonly, last section aes_block_padding ;
```

1. Depends on the STM32 microcontroller Series.

3.2.4 Multiple images 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 constraint, 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 sector.

Figure 13 shows the example of multiple images configuration provided in 2_Images_ExtFlash of the B-L475E-IOT01A board.
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 SystemClock_Config() function.
3. Disable the Cortex M4 boot: BCM4 bit from option byte must be unchecked.
4. Add in SFU_LL_SECU_CheckFlashConfiguration() function the control of the BCM4 bit state.
5. Add in the UserApplication project, the trigger of CM4 boot.
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 images 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_Images and 2_Images variants.
  - Third-party middleware: mbedTLS (open-source code) cryptographic services. Examples are provided for the 32L496GDISCOVERY, B-L475E-IOT01A, 32F413HDISCOVERY, 32F769IDISCOVERY, P-NUCLEO-WB55.Nucleo, and NUCLEO-H753ZI 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 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.

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 the 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 the 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 images 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.

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 16.
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 to protect secrets against both outer and inner attacks.

STM32 security peripherals can be deactivated independently as per user’s decision 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 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)

```c
// #define SECBOOT_DISABLE_SECURITY_IPS /*!
#if !defined(SECBOOT_DISABLE_SECURITY_IPS)/
#define SFU_WRP_PROTECT_ENABLE
#define SFU_RDP_PROTECT_ENABLE
#define SFU_PCROP_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
```

- **RDP-L2, DAP/TAMPER**
  - Disable external access
  - Protects boot options
  - Lock option bytes
    - WRP
    - PCROP
    - BFB2

- **WRP, PCROP**
  - Protects the code enabling the MPU/Firewall
  - Protects the code considered trusted
  - Protects part of the Flash

- ** MPU **
  - Execution allowed only inside the chain of trust

- ** FireWall **
  - Protects RAM and Flash at runtime

- ** Crypto **
  - Verify the integrity, authenticity of the user application

- ** Trust **

- ** User application **
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 STM32WB 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.
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 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 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 in order 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, 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 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 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 21* shows how Option Bytes are managed at SBSFU startup:

**Figure 21. 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 22* shows the few steps to modify the firmware encryption key of the active slot #1. The same applied for the 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
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. The provisioning process is described in `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 24* shows an example of the firmware encryption key modification of the 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.
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 \texttt{STSAFE_Provisioning/readme.txt}.

\textit{Figure 25} shows an example of pairing-key provisioning:

1. STSAFE-A110 provisioning with default pairing keys
2. Update \texttt{STSAFE\_PAIRING\_keys.bin} accordingly
3. Compile \texttt{SECoreBin}: \texttt{prebuild.bat} is executed and \texttt{se\_key.s} is generated.
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 26. 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 27 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 28.

### Figure 28. 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 29:

- Browse to select file Project.out
- Set Offset to 0
- Check the Debug info only box
Figure 29. 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 own cryptographic scheme.

In order to implement a new cryptographic scheme for SBSFU, follow the steps illustrated in Figure 30 and described below.

Figure 30. 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`.
5. **Update the preparation tools**
6. **Update the IDE integration**

**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; 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>
<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 SFU_DEBUG_MODE</td>
<td>No more information displayed on the terminal during SBSFU execution</td>
<td>~ 9 Kbytes</td>
</tr>
<tr>
<td>Disable SECBOOT_USE_LOCAL_LOADER</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 the user's implementation</td>
</tr>
<tr>
<td>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.</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 10.

As an example, Figure 31 highlights the mapping modifications to be done. Starting from 2 images with a symmetric cryptographic 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.
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 32* 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 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 32. IDE adaptations*
7.4 How to improve boot time

In order 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 33.

Figure 33. Boot time
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 the active slot #1. In a multiple images 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 34.

---

Figure 34. 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 35 shows how to update the linker file to verify this constraint.

**Figure 35. 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 postbuil.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 36. IDE adaptations**
As explained in the user manual 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 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.

*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
- Not in the swap area

Figure 37 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 37. 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-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 38.

Figure 38. UserApp firmware download overview
8.4 How to replace the standalone loader with a BLE OTA loader

For STM32WB Series, an example of the BLE OTA loader application is provided in STM32CubeWB Embedded Package.

Figure 39 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, except if this constraint is taken into account during userapp.sfb generation with a specific alignment option.
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 the BLE stack through the M0+ core.

Figure 39. BLE OTA loader replacement
8.5 How to change the firmware version

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

Figure 40. 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.

8.6 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.

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)` 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 41. 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 the firmware compatibility between all new images in case of interruption during the validation phase.
Caution: This feature can be activated only on a dual-slot configuration example with Secure Engine runtime services available and the swap installation process selected. This is not the case for series with secure memory.
## 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>31-Aug-2018</td>
<td>2</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>18-Dec-2018</td>
<td>3</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 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>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>
</tbody>
</table>
Table 4. Document revision history (continued)

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>09-Jul-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 the NUCLEO-G031K8 – 1 image.</td>
</tr>
<tr>
<td>1-Sep-2020</td>
<td>6</td>
<td>Added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 3.2.4: Multiple images configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 3.3: Dual-core adaptation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 7.3: How to activate interruption management inside the firewall isolated environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 7.4: How to improve boot time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 8.6: How to validate a firmware image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Secure element STSAFE-A100 replaced by STSAFE-A110</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 acknowledgment.

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

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

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

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, 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