AN5054
Application note

Secure programming using STM32CubeProgrammer

Introduction

This document specifies the steps and tools required to prepare SFI (secure internal firmware install), SFIx (secure external firmware install), SMI (secure module install) or SSP (secure secret provisioning) images. It then describes how to program these into STM32 MCU devices that support SFI/SFIx on-chip internal memory, external Flash memory or, for the SSP install procedure, STM32 MPU devices. It is based on the STM32CubeProgrammer tool set (STM32CubeProg). These tools are compatible with all STM32 devices.

The main objective of the SFI/SFIx and SMI processes is the secure installation of OEM and software-partner's firmware, which prevents firmware cloning.

The STM32MP1 Series supports protection mechanisms allowing protection of critical operations (such as cryptography algorithms) and critical data (such as secret keys) against unexpected access.

This application note also gives an overview of the STM32 SSP solution with its associated tool ecosystem, and explains how to use it to protect OEM secrets during the CM product manufacturing stage.

Refer also to:

- AN4992 *[1]*, which provides an overview of the secure firmware install (SFI) solution, and how this provides a practical level of protection of the IP chain - from firmware development up to programming the device on-chip Flash memory.
- AN5510 *[3]*, which provides an overview of secure secret provisioning (SSP).

Contents

List of tables

List of figures

1 General information

1.1 Licensing information

STM32CubeProgrammer supports STM32 32-bit devices based on Arm®[a] Cortex®-M processors.

1.2 Acronyms and abbreviations

Table 1. List of abbreviations

Abbreviations	Definition
AES	Advanced encryption standard
CLI	Command line interface
CM	Contract manufacturer
GCM	Galois counter mode (one of the modes of AES)
GUI	Graphical user interface
HSM	Hardware security model
HW	Hardware
MAC	Message authentication code
MCU	Microcontroller unit
OEM	Original equipment manufacturer
PCROP	Proprietary code read-out protection
PI	Position independent
ROP	Read-out protection
RSS	Root security service (secure)
RSSe	Root security service extension
SFI	Secure firmware install
STPC	STM32 Trusted Package Creator
SMI	Secure modules install
STM32	ST family of 32-bit ARM based microcontrollers
SW	Software
XO	Execute only

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

2 How to generate an execute-only/position independent library for SMI preparation

This section describes the requirements and procedures for the preparation of an execute-only (XO) and position independent (PI) library using a partner toolchain

These kinds of libraries serve in encrypted SMI-module generation.

2.1 Requirements

SMI modules run in execute-only (XO) areas, also called PCROP areas, and must be relocatable to be linkable with final OEM application. Nevertheless, today, 3rd party toolchains for STM32 devices (such as MDK-ARM™ for ARM, EWARM for IAR™ and GCC based IDEs) do not allow both features to be activated at the same time. So, starting from particular versions of 3rd party toolchains, the two features below are possible for SMI support:

- Position independent support (code + rw data + ro data)
- No literal pool generation - needed for the PCROP feature.

2.2 Toolchains allowing SMI generation

Three toolchains allow SMI generation:

- EWARM

  Version 7.42.0 allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: “--ropi_cb” + “rwpi” + “--no_literal_pool”.
  - “--ropi_cb” + “rwpi” are needed for position independent support
  - option “no_literal_pool” is needed for the PCROP feature.
- MDK-ARM

  The customized version allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: “-fropi-cb”, “-frwpi”, “-mexecute-only”.
  - “fropi-cb” is needed for ro data independent position
  - “frwpi” is needed for rw data independent position
  - option “-mexecute-only” is needed for PCROP feature.

  All library symbols being used in the final application must be added to the final project in a “.txt” file format.
- GCC

  The customized version of GCC based toolchains allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: “-masset”.

  Option “-masset” has the same role as “--ropi --ropi_cb --rwpi --no_literal_pool” options used for the EWARM toolchain.

2.3 Execute-only/position independent library scenario example under EWARM

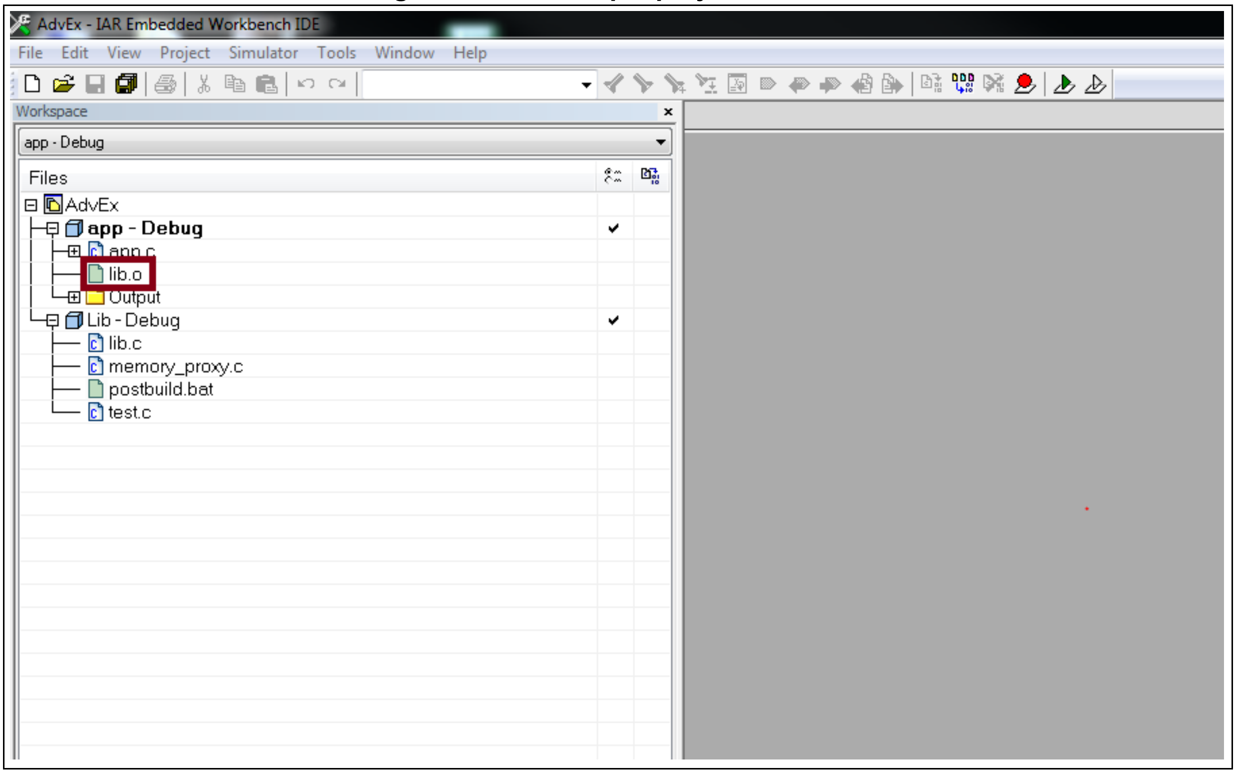
In order to generate an execute-only (XO) and position independent (PI) library a customized version of the IAR toolchain must be used: version 7.42.0.

2.3.1 Relocatable library preparation steps

1. Open the project available in the “Example” folder: double click on *“Example/AdvEx.eww”*.

   The project architecture is illustrated in *Figure 1*.

Figure 1. IAR example project overview

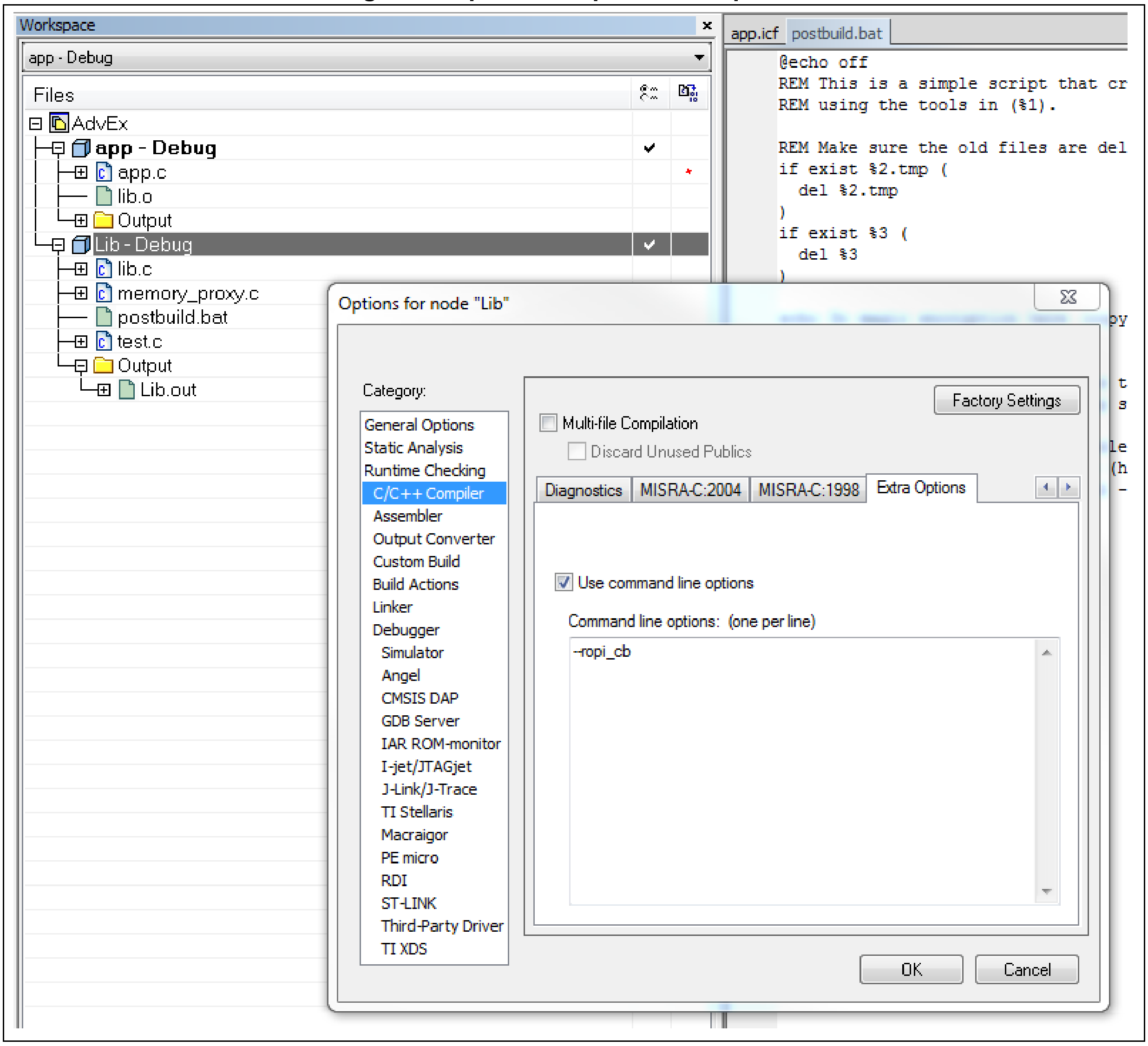


The following steps update the old *"lib.o"* linked to the example application by making it support both PI and XO features:

2. Within Lib-Debug options -> C/C++ Compiler. Go to tab "Extra Options" and add the following line:
   "--ropi_cb"

   This action is illustrated in *Figure 2*.

Figure 2. Update compiler extra options



3. Within Lib-Debug options -> Linker. Go to the “Extra Options” tab and add the following lines:

   --no_literal_pool

   --ropi_cb

   --loadable

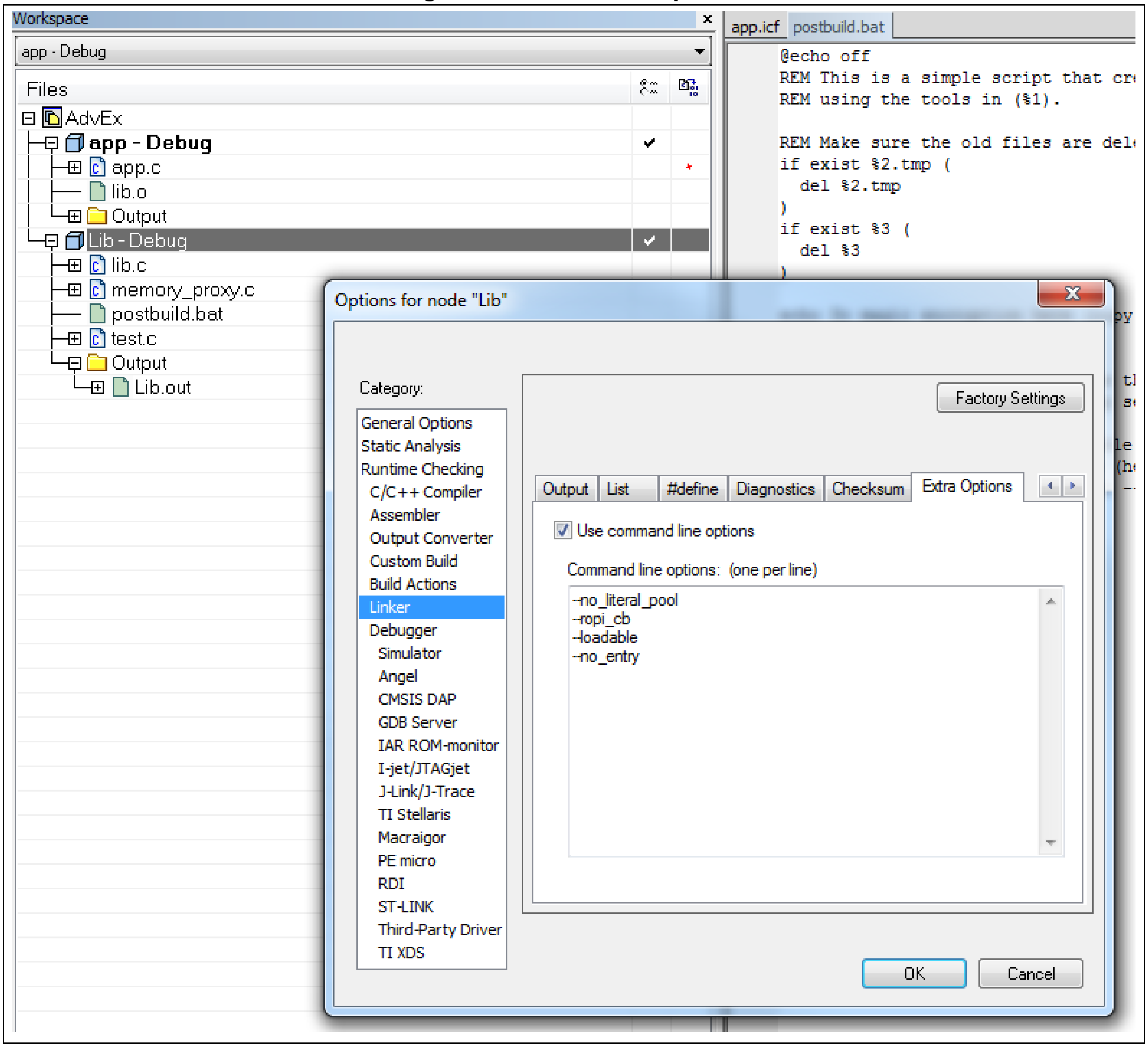
   --no_entry

   This action is illustrated in *Figure 3*.

   – “ropi_cb” is needed for Position Independent support

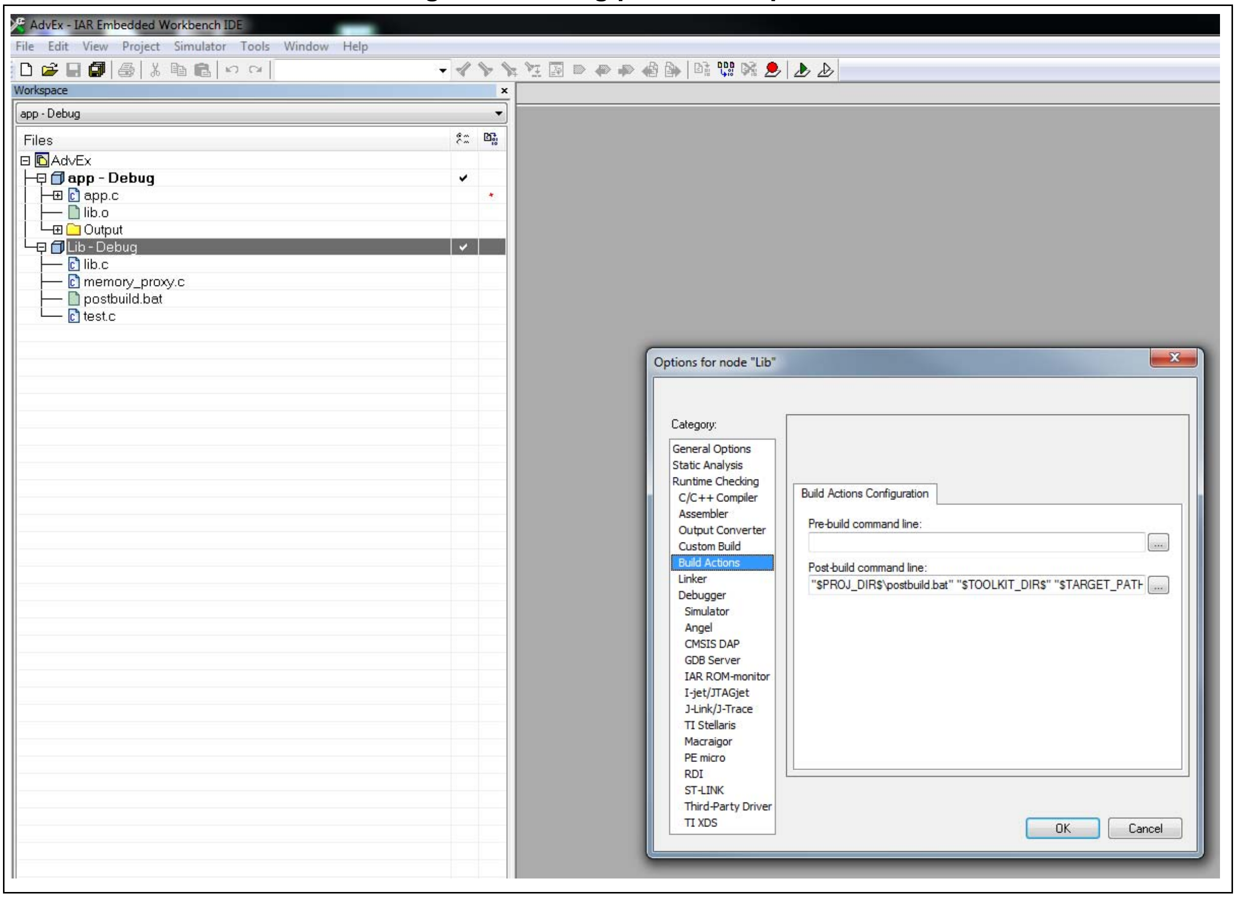
   – the “no_entry” is a linker option that sets the entry point field to zero.

Figure 3. Linker extra options



4. Within Lib-Debug options -> Build actions. In post build command line execute the batch file *“postbuild.bat”* by inserting, if it is not already configured, the following command line:
`"$PROJ_DIR$\postbuild.bat" "$TOOLKIT_DIR$" "$TARGET_PATH$" "$PROJ_DIR$\lib.o"`
This action is illustrated in *Figure 4*.

Figure 4. Setting post-build option



The *“postbuild.bat”* file is used to perform some key actions:

- **--wrap**: adds veneers to library functions to initialize registers used for ropi code
- *“iexe2obj.exe”*: transforms the elf into a linkable object file.

See *Figure 5*.

Figure 5. Postbuild batch file

```
@echo off
REM This is a simple script that creates and object file (%3) from an image (%2)
REM using the tools in (%1).

REM Make sure the old files are deleted before we try to generate the new ones
if exist %2.tmp (
  del %2.tmp
)
if exist %3 (
  del %3
)

echo Do magic encryption here (copy is just a placeholder)
copy %2 %2.tmp

REM This is the list of functions that will have a wrapper generated
SET __WRAP=--wrap ToString --wrap setup_memory --wrap setup_memory2

REM convert the image to a linkable object file using _Lib as prefix
REM and keeping all mode symbols (helps a bit with debugging)
%1\bin\iexe2obj.exe --prefix _Lib --keep_mode_symbols %__WRAP% %2.tmp %3
```

5. Rebuild the project “Lib”

2.3.2 Relocatable SMI module preparation steps

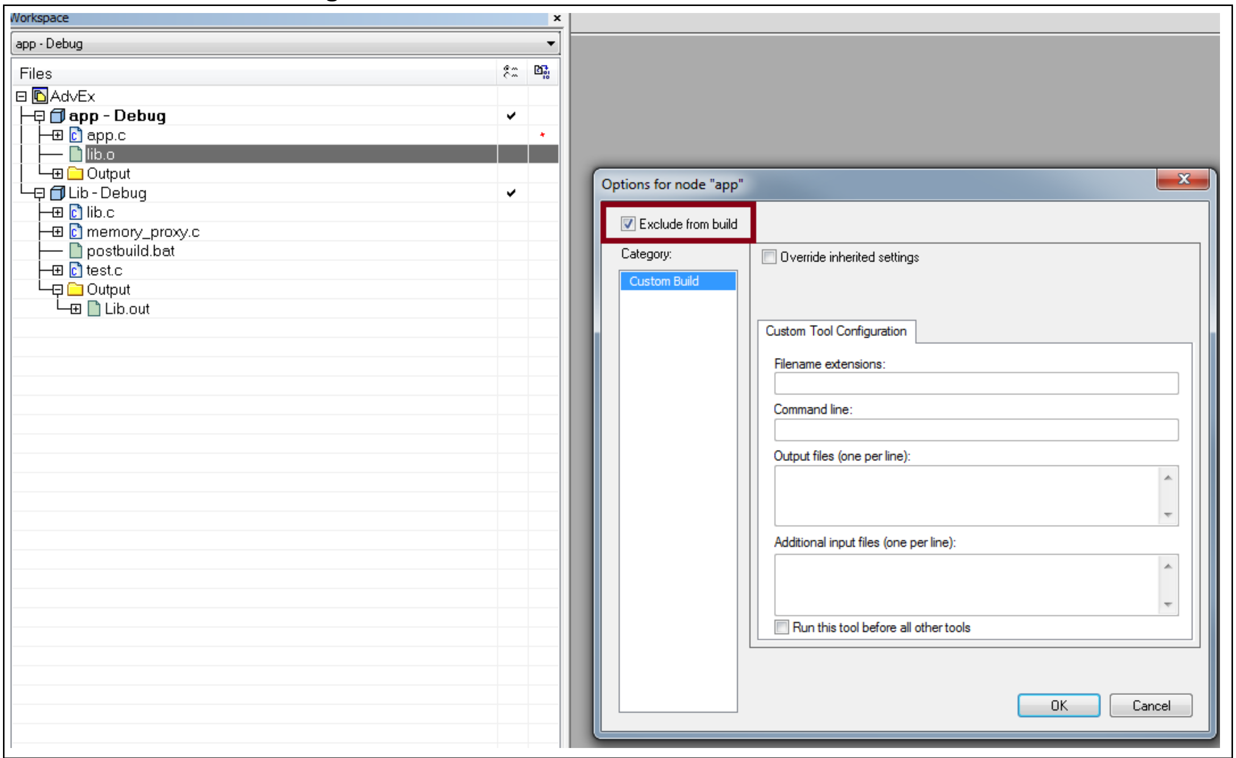
From the object file created, *“lib.o”*, generate the SMI relocatble module using the STM32 Trusted Package Creator tool *“libr.smi”* and its corresponding data clear part (*libr_clear.o*: corresponding to the input *“lib.o”* without the protected section code).

To execute this step, follow the steps explained for SMI generation under section “*Section 3.6.2: Steps for SMI generation (CLI)*”.

2.3.3 Application execution scenario

1. Flash the already generated SMI relocatable module to address 0x08080000 using STM32Cube Programmer v0.4.0 or newer (see *Section Figure 63.: SFI install via SWD execution - (2)* to perform this action).
2. Link the data clear part, *"libr_clear.o"*, generated from STM32 Trusted Package Creator tool to the final IAR example application instead of the old previously used *"lib.o"*.
3. Exclude *"lib.o"* from the build (*Figure 6*).

Figure 6. How to exclude the "lib.o" file from build



4. Rebuild the application.
5. Do these modifications in an example application ICF file:
   a) Define region for PCROP block:

      define symbol __ICFEDIT_region_PCROP_start__ = 0x08080000;

      define symbol __ICFEDIT_region_PCROP_end__ = 0x0809FFFF;

      define region PCROP_region = mem:[from __ICFEDIT_region_PCROP_start__ to __ICFEDIT_region_PCROP_end__];

   b) Define PCROP region as 'noload' (since it is already installed using STM32CubeProgrammer so no need to load it again)

      'SMI': place noload in PCROP_region { ro code section __code__Lib};

These modifications are illustrated within the *"app.icf"* file, which is shown in *Figure 7*.

Figure 7. app.icf file

app.icf

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x24000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x24000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x24002FFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x24003000;
define symbol __ICFEDIT_region_RAM_end__   = 0x2407FFFF;

define symbol __ICFEDIT_region_PCROP_start__ = 0x08080000;
define symbol __ICFEDIT_region_PCROP_end__   = 0x0809FFFF;

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x2000;
define symbol __ICFEDIT_size_heap__   = 0x2000;
/**** End of ICF editor section. ###ICF###*/

define symbol __region_RAM1_start__ = 0x10000000;
define symbol __region_RAM1_end__   = 0x1000FFFF;

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];
define region RAM1_region  = mem:[from __region_RAM1_start__    to __region_RAM1_end__ ];
define region PCROP_region  = mem:[from __ICFEDIT_region_PCROP_start__  to __ICFEDIT_region_PCROP_end__];


define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

/*define block PCROP_block  with alignment = 256 {ro code section __code__Lib};*/

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
place in RAM1_region  { section .sram };

"SMI": place noload in PCROP_region { ro code section __code__Lib};

/*place in PCROP_region { block PCROP_block };*/
```

6. To check that example application executed successfully on the STM32H7 device:
   a) Check that address 0x08080000 was protected with PCROP.
   b) The expected "`printf`" packets appears in the terminal output.

3 Encrypted firmware (SFI) and module (SMI) preparation using the STPC tool

The STM32 Trusted Package Creator (STPC) tool allows the generation of SFI and SMI images for STM32H7 devices. It is available in both CLI and GUI modes free of charge from *www.st.com*.

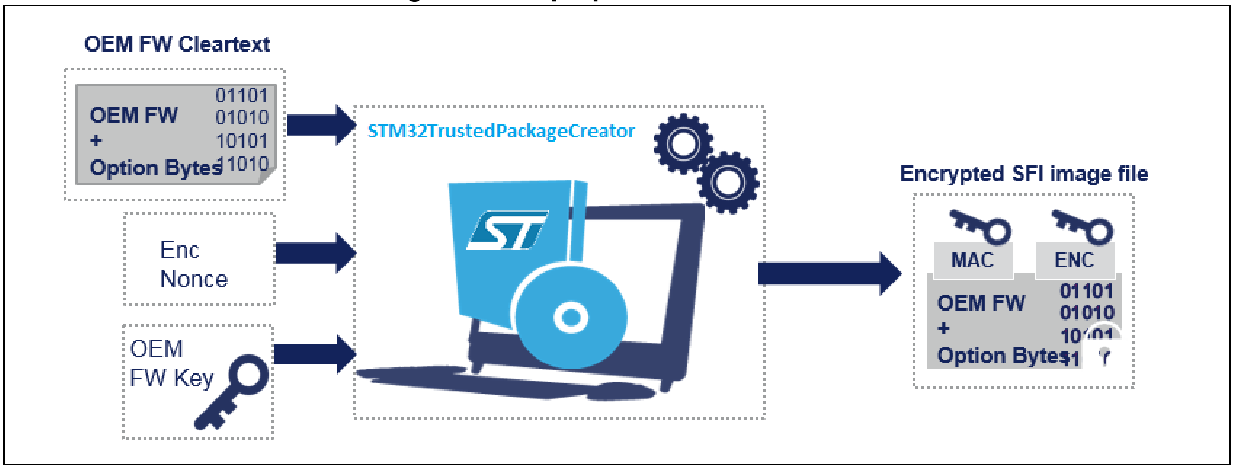
3.1 System requirements

Using the STM32 Trusted Package Creator tool for SFI/SFIx, SMI and SSP image generation requires a PC running on either Windows® 7/10, or Ubuntu® 14 in 64-bit versions and macOS®.

3.2 SFI generation process

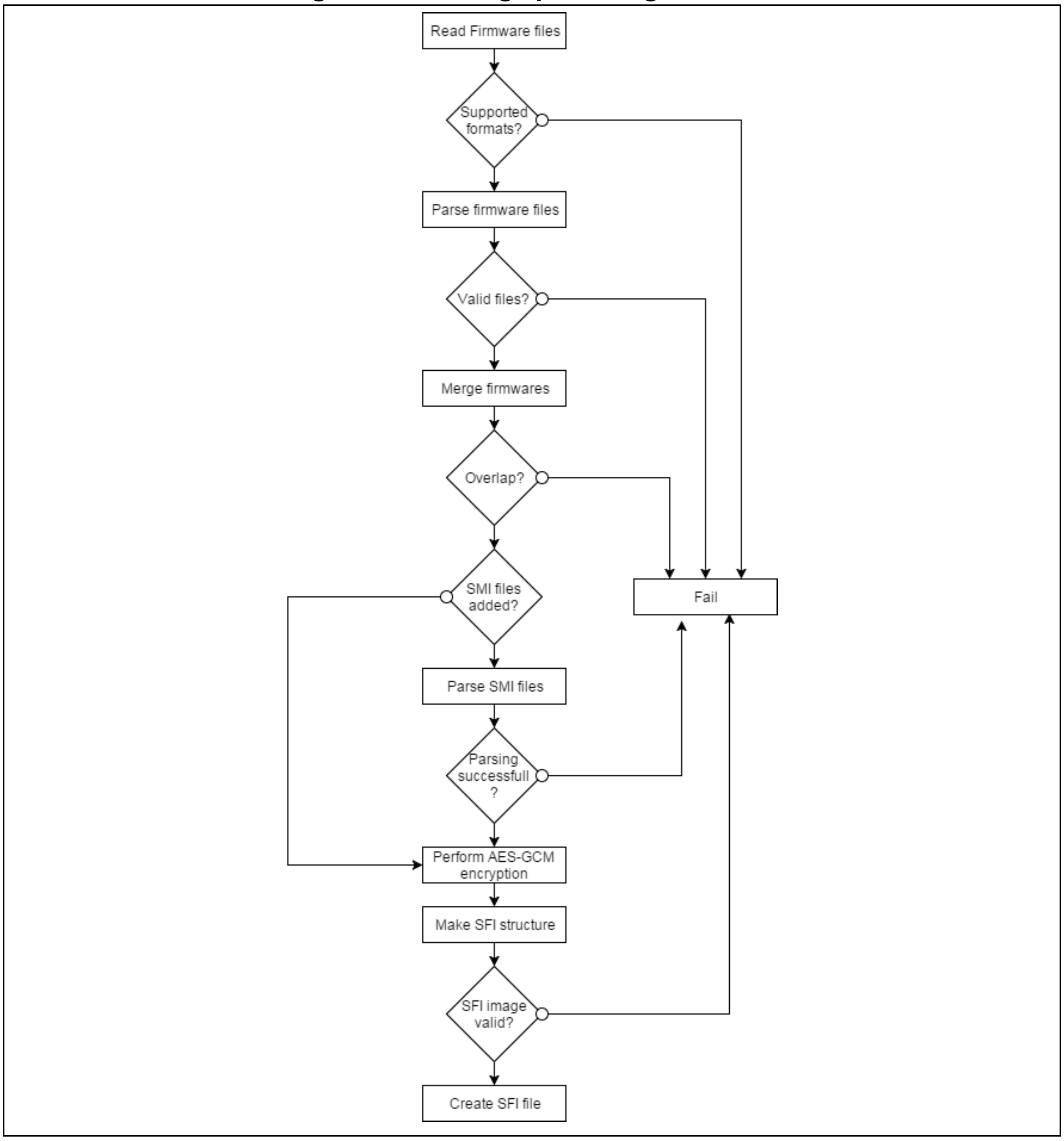
The SFI format is an encryption format for internal firmware created by STMicroelectronics that transforms internal firmware (in ELF, Hex, Bin or Srec formats) into encrypted and authenticated firmware in SFI format using AES-GCM algorithm with a 128-bit key. The SFI preparation process used in the STM32 Trusted Package Creator tool is described in *Figure 8*.

Figure 8. SFI preparation mechanism



The SFI generation steps as currently implemented in the tool are described in *Figure 9*.

Figure 9. SFI image process generation



Before performing AES-GCM to encrypt an area, we calculate the Initialization Vector (IV) as:

IV = nonce + Area Index

The tool partitions the firmware image into several encrypted parts corresponding to different memory areas.

These encrypted parts appended to their corresponding descriptors (the unencrypted descriptive header generated by the tool) are called areas.

These areas can be of different types:

- ‘F’ for a firmware area (a regular segment in the input firmware)
- ‘M’ for a module area (used in SFI-SMI combined-image generation, and corresponds to input from an SMI module)
- ‘C’ for a configuration area (used for option-byte configuration)
- ‘P’ for a “pause” area
- ‘R’ for a “resume area.

Areas ‘P’ and ‘R’ do not represent a real firmware area, but are created when an SFI image is split into several parts, which is the case when the global size of the SFI image exceeds the allowed RAM size predefined by the user during the SFI image creation.

The STM32 Trusted Package Creator overview below (*Figure 10*) shows the RAM size input for SFI image generation, and also the ‘Continuation token address’ input, which is used by SFI multi install to store states in Flash memory during SFI programming.

Figure 10. RAM size and CT address inputs used for SFI multi install

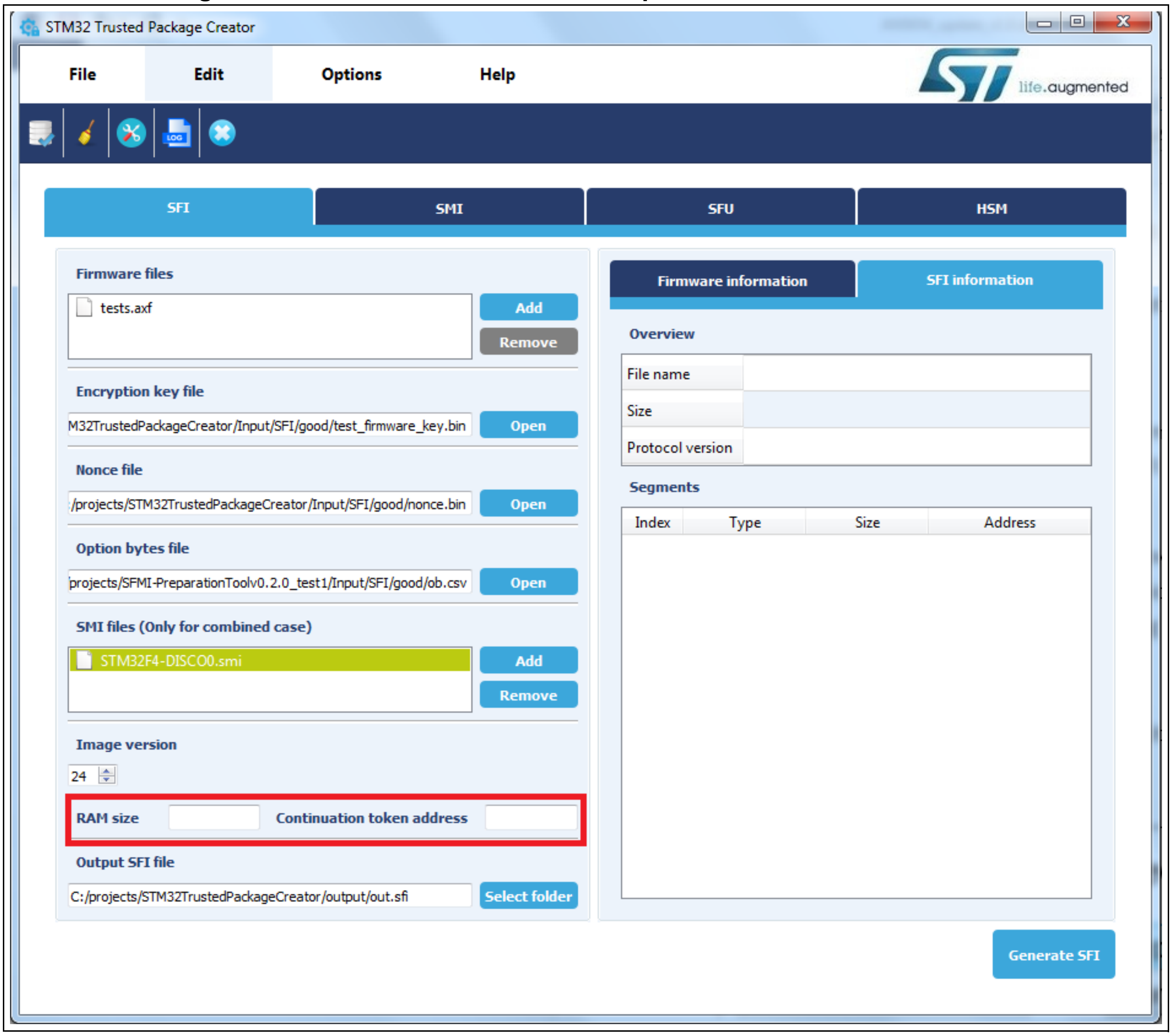
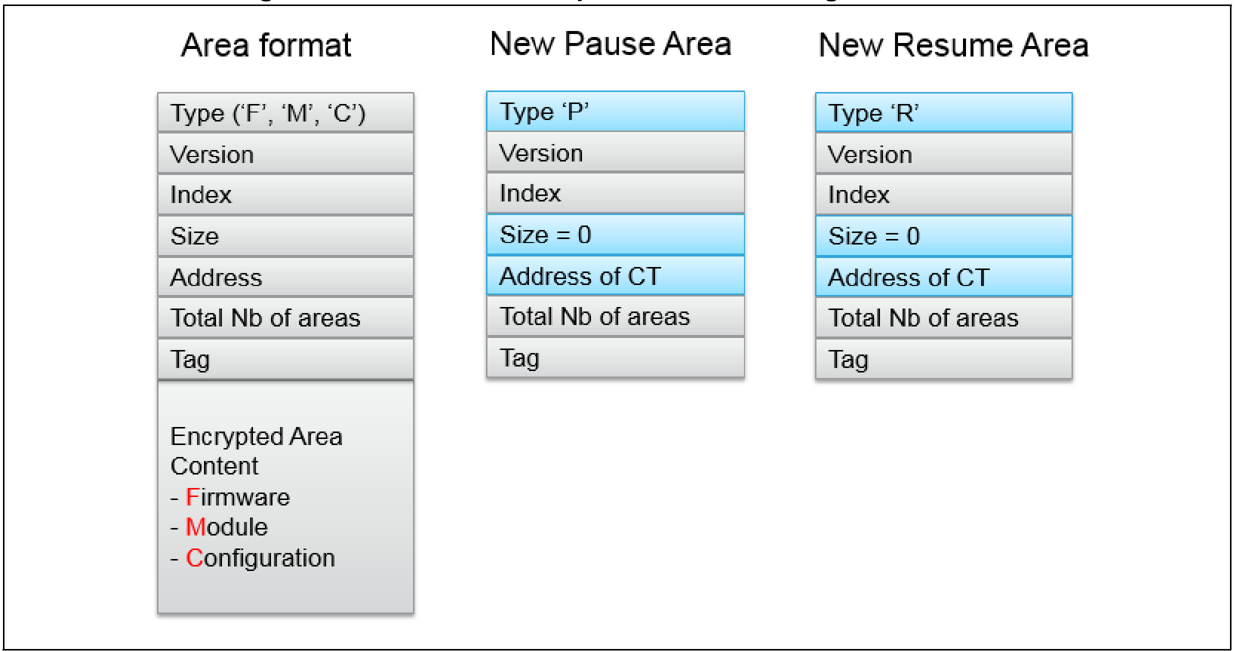


Figure 11 (below) shows the specifics of these new areas compared to a regular SFI area.

Figure 11. 'P' and 'R' area specifics versus a regular SFI area

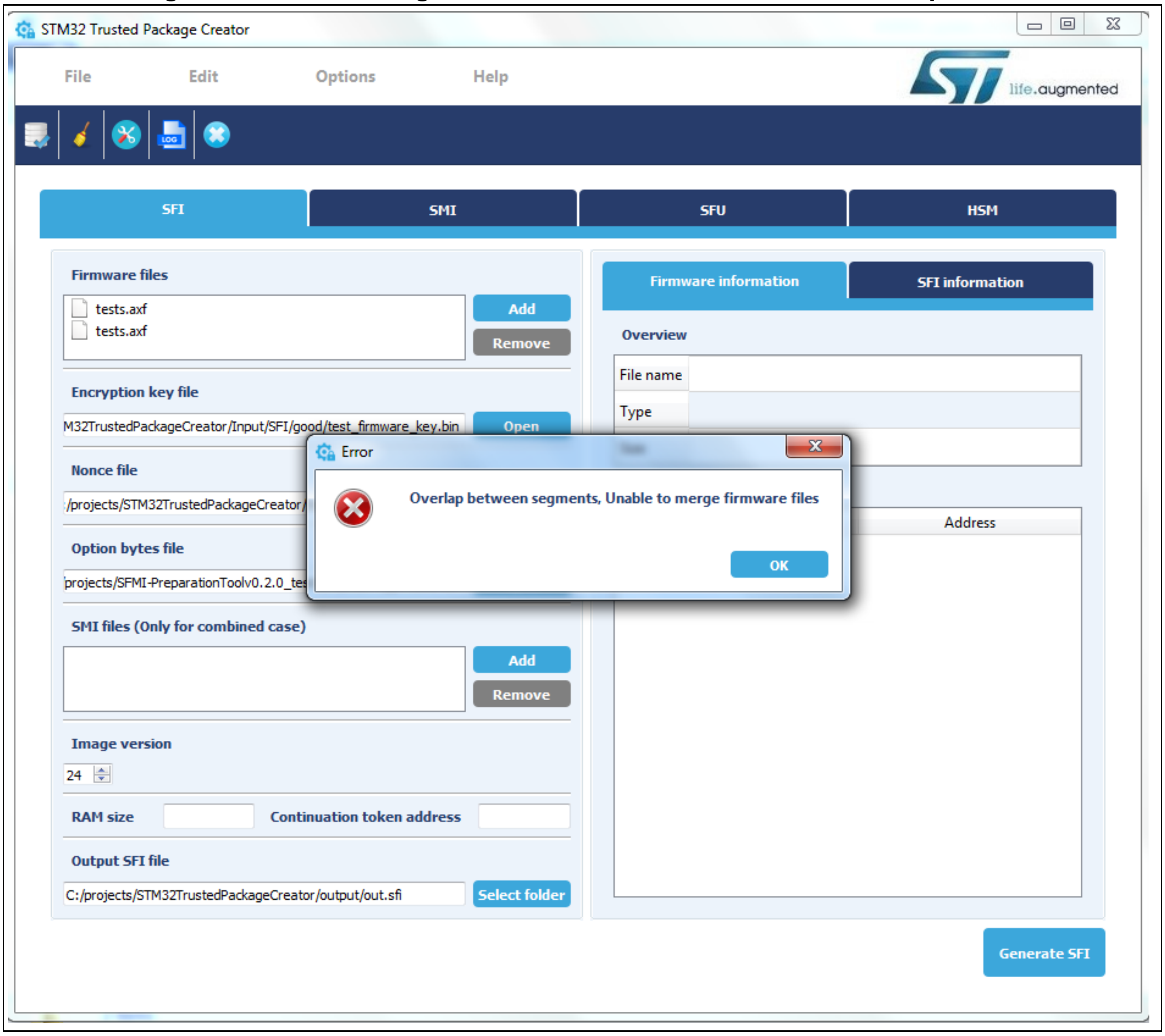


A top-level image header is generated then authenticated, for this the tool performs AES-GCM with authentication only (without encryption), using the SFI image header as an AAD, and the nonce as IV.

An authentication tag is generated as output.

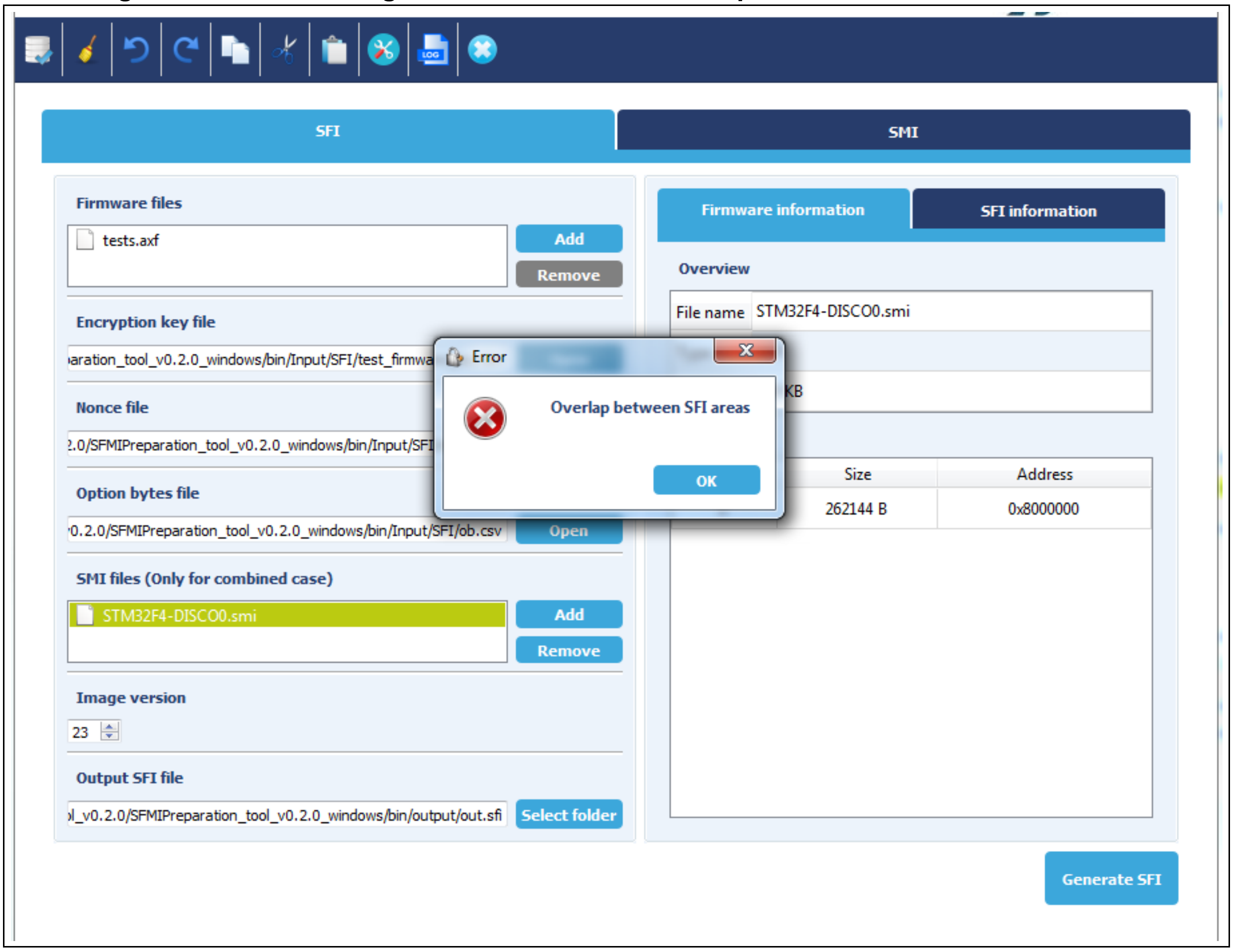
Note: To prepare an SFI image from multiple firmware files, make sure that there is no overlap between their segments, otherwise an error message appears (Figure 12: Error message when firmware files with address overlaps used).

Figure 12. Error message when firmware files with address overlaps used



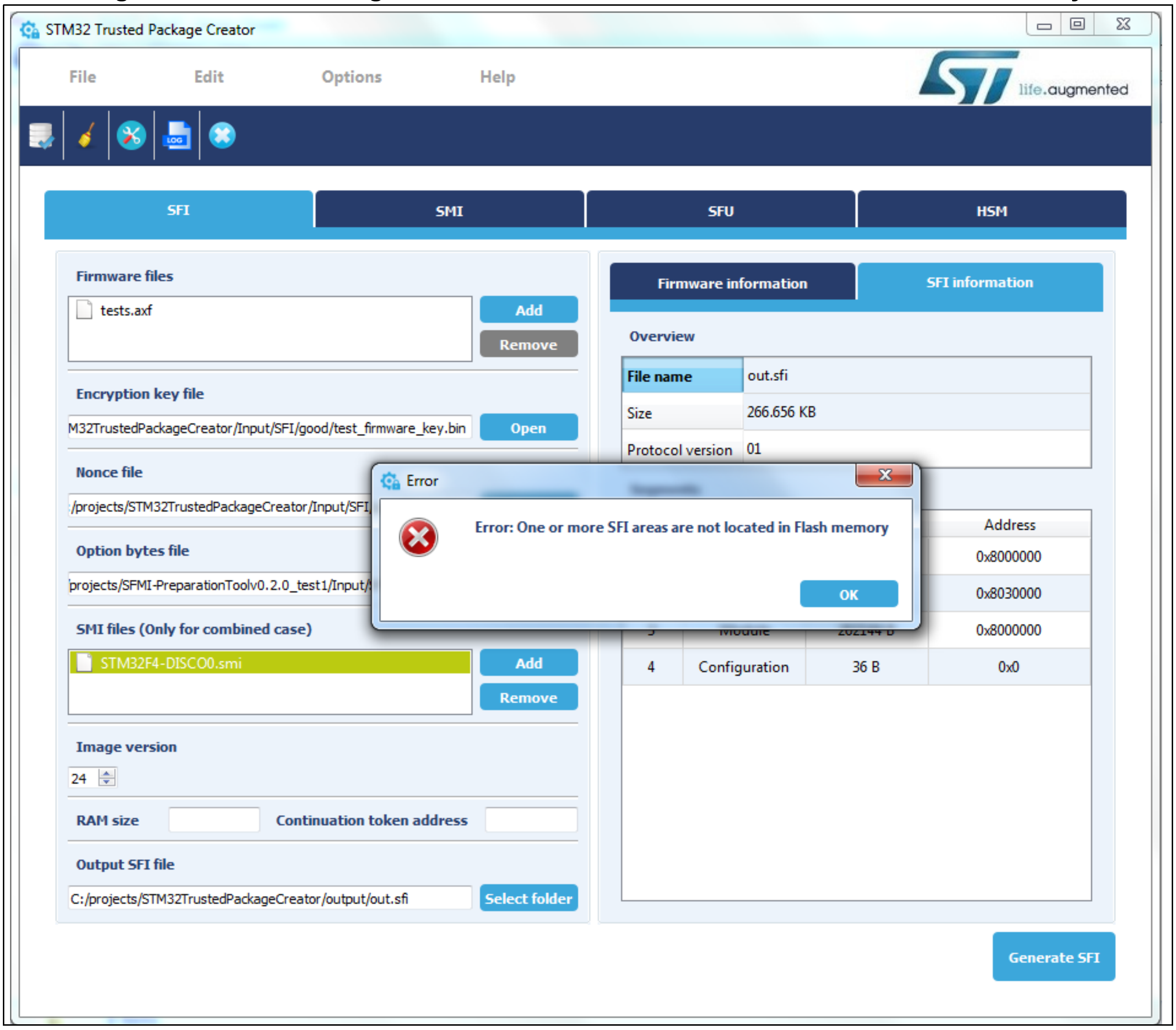
For combined SFI-SMI images, there is also an overlap check between firmware and module areas. If the check fails, an error message appears (*Figure 13*).

Figure 13. Error message when SMI address overlaps with a firmware area address



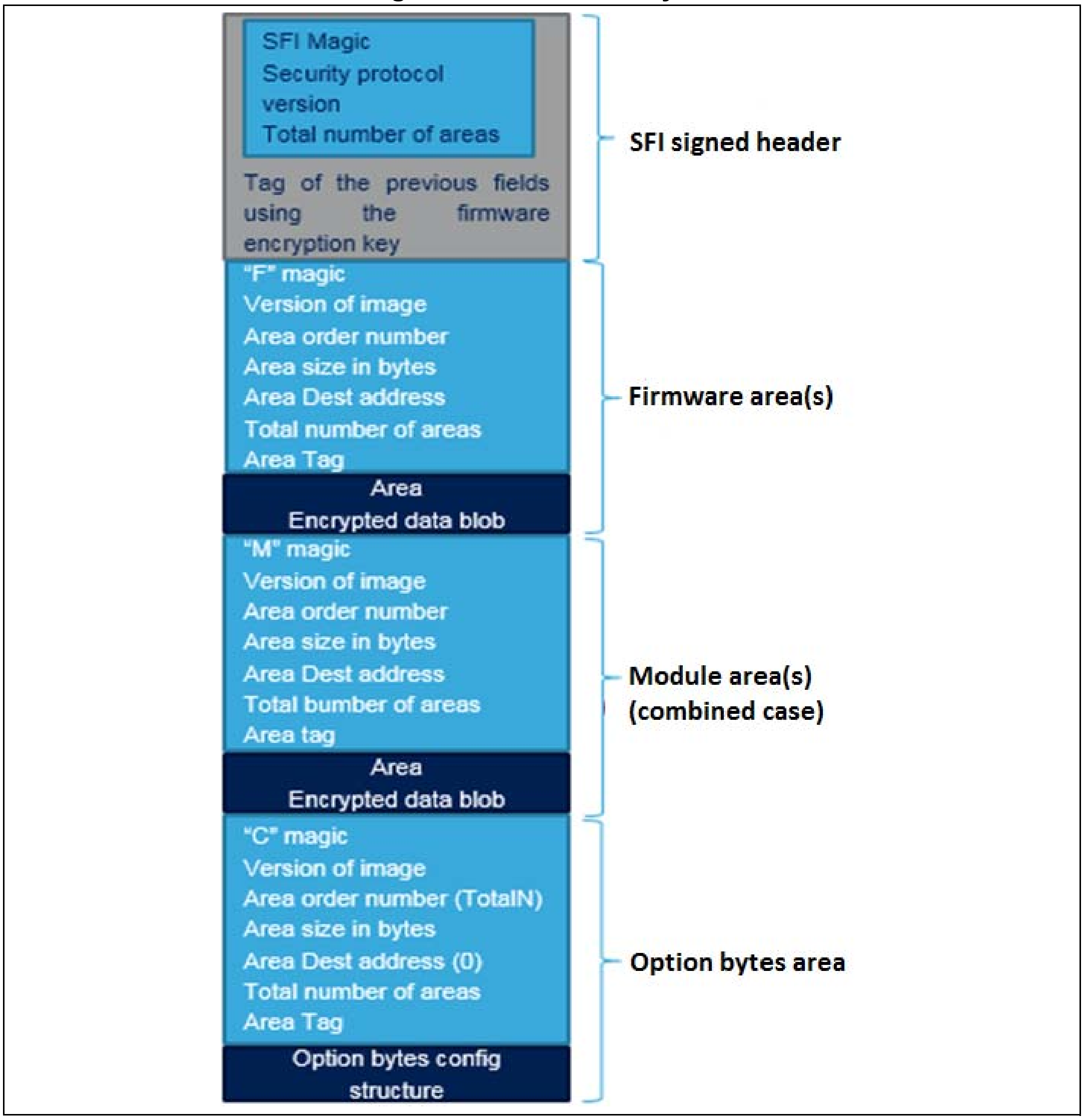
Also, all SFI areas must be located in Flash memory, otherwise the generation fails and the following error message appears (*Figure 14*).

Figure 14. Error message when a SFI area address is not located in Flash memory



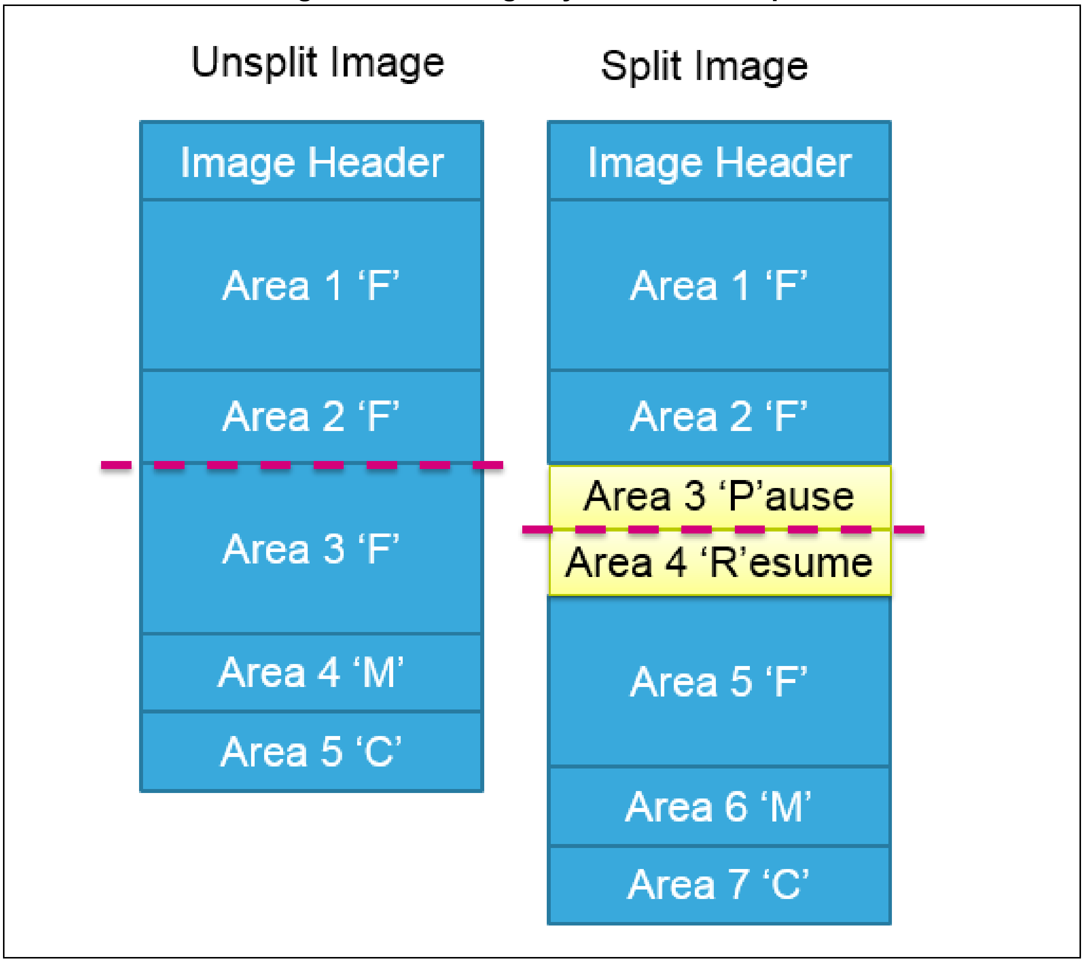
The final output from this generation process is a single file, which is the encrypted and authenticated firmware in ".sfi" format. The SFI format layout is described in *Figure 15*.

Figure 15. SFI format layout



When the SFI image is split during generation, areas ‘P’ and ‘R’ appear in the SFI image layout, as in the example below *Figure 16*.

Figure 16. SFI image layout in case of split



3.3 SFIx generation process

In addition to the SFI preparation process mentioned in the previous section, two extra areas are added in the SFI image for the SFIx preparation process:

- 'E' for an external firmware area
- 'K' for a key area (used for random keys generation)

The key 'K' area is optional and it can be stored in the area 'F'.

Area E

The area 'E' is for external Flash memory. It includes the following information at the beginning of an encrypted payload:

- OTFD region_number (uint32_t):
  - 0…3: OTFD1 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, and STM32L5)
  - 4…7: OTFD2 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335)
- OTFD region_mode (uint32_t) bit [1:0]:
  - 00: instruction only AES-CTR)
  - 01: data only (AES-CTR)
  - 10: instruction + data (AES-CTR)
  - 11: instruction only (EnhancedCipher)
- OTFD key_address in internal Flash memory (uint32_t).

After this first part, area 'E' includes the firmware payload (as for area 'F'). The destination address of area 'E' is in external Flash memory (0x9… / 0x7…).

Area K

The area 'K' triggers generation of random keys. It contains N couples; each one defines a key area as follows:

- the size of the key area (uint32_t)
- the start address of the key area (uint32_t): address in internal Flash memory.

Example of an area 'K':

```
0x00000002 0x00000080 0x08010000 0x00000020 0x08010100
```

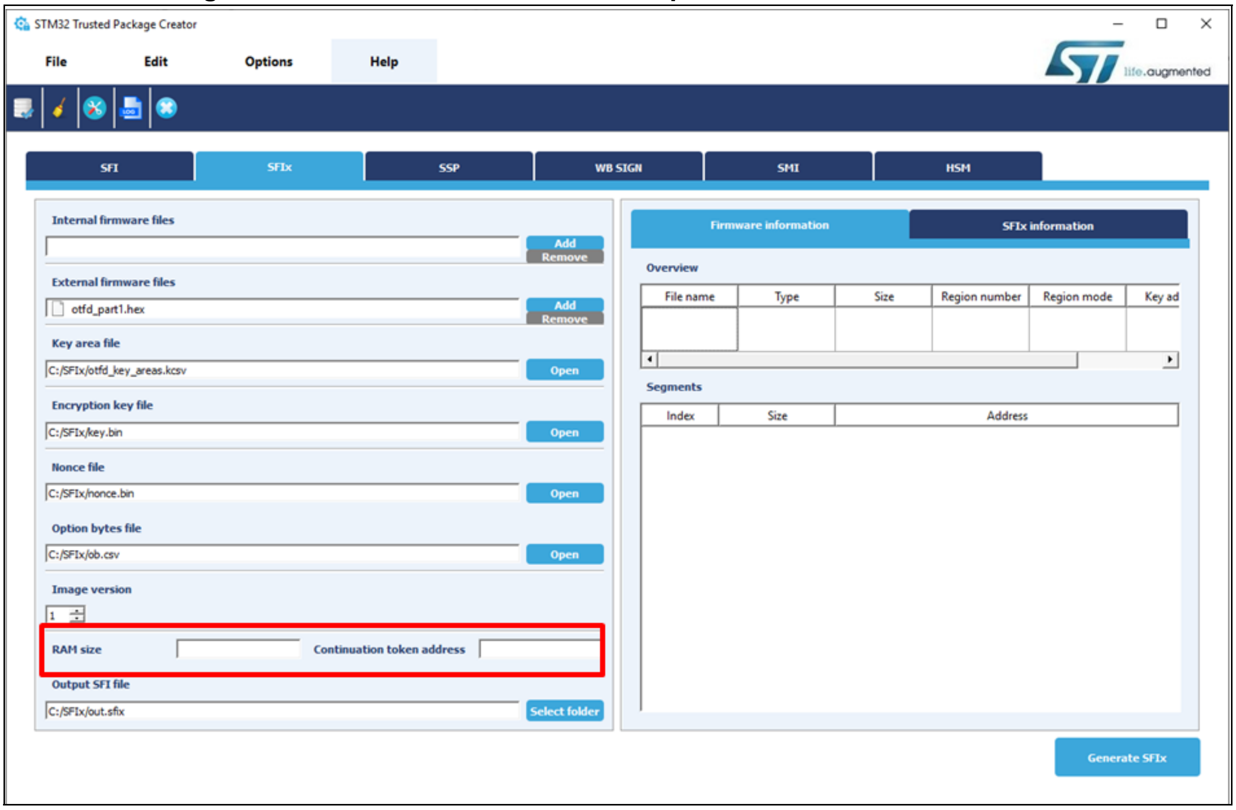
There are two key areas:

- the first key area starts at 0x08010000 with size = 0x80 (8 x 128-bits keys)
- the second key area starts at 0x08010100 with size 0x20 (256-bits key).

The STM32 Trusted Package Creator overview below (*Figure 17: RAM size and CT address inputs used for SFIx multi install*) shows the RAM size input for SFIx image generation, and also the 'Continuation token address' input, which is used by SFIx multi install to store states in external/internal Flash memory during SFIx programming.

The 'Continuation token address' is mandatory due to the image generation which adds areas P and R whatever be the configuration.

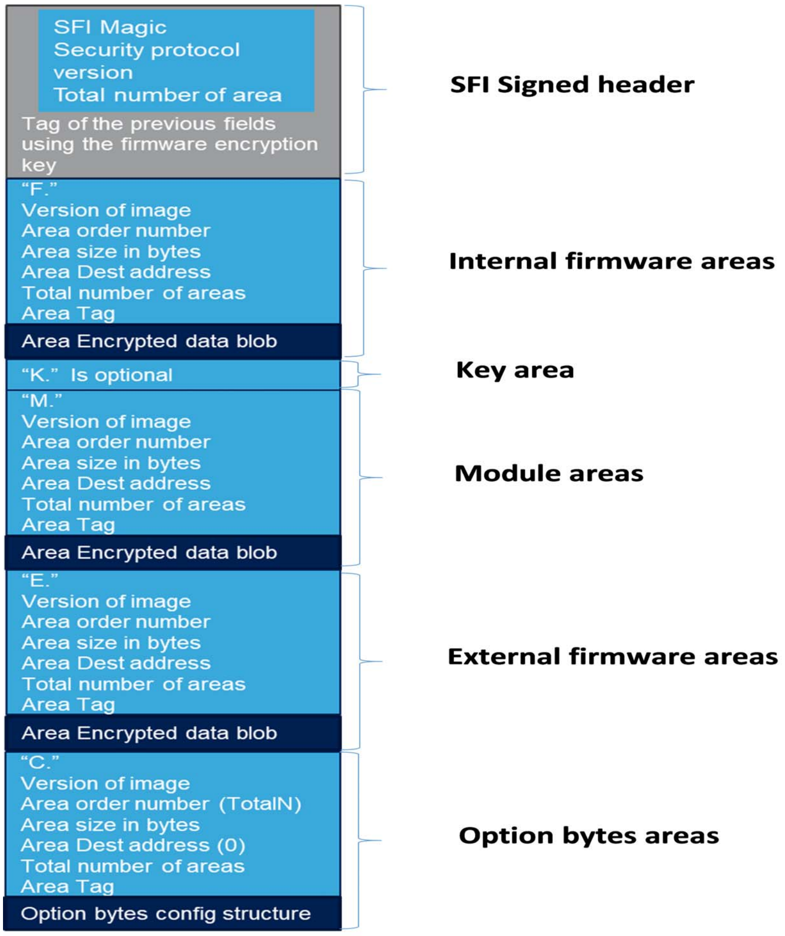
Figure 17. RAM size and CT address inputs used for SFIx multi install



Note: To prepare an SFIx image from multiple firmware files, make sure that there is no overlap between their segments (Intern and extern), otherwise an error message appears as same as in the SFI use case.

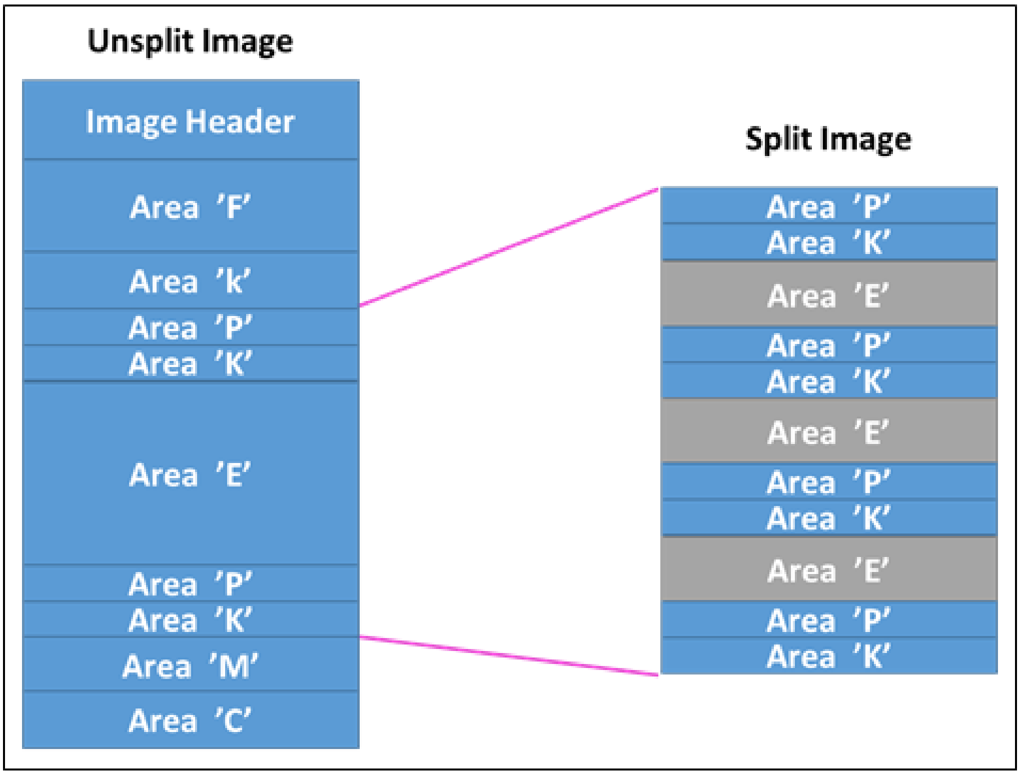
The final output from this generation process is a single file, which is the encrypted and authenticated internal/external firmwares in ".sfix" format. The SFIx format layout is described in *Figure 18*.

Figure 18. SFIx format layout



When the SFIx image is split during generation, areas ‘P’ and ‘R’ appear in the SFIx image layout, as in the example below *Figure 19*.

Figure 19. SFIx image layout in case of split

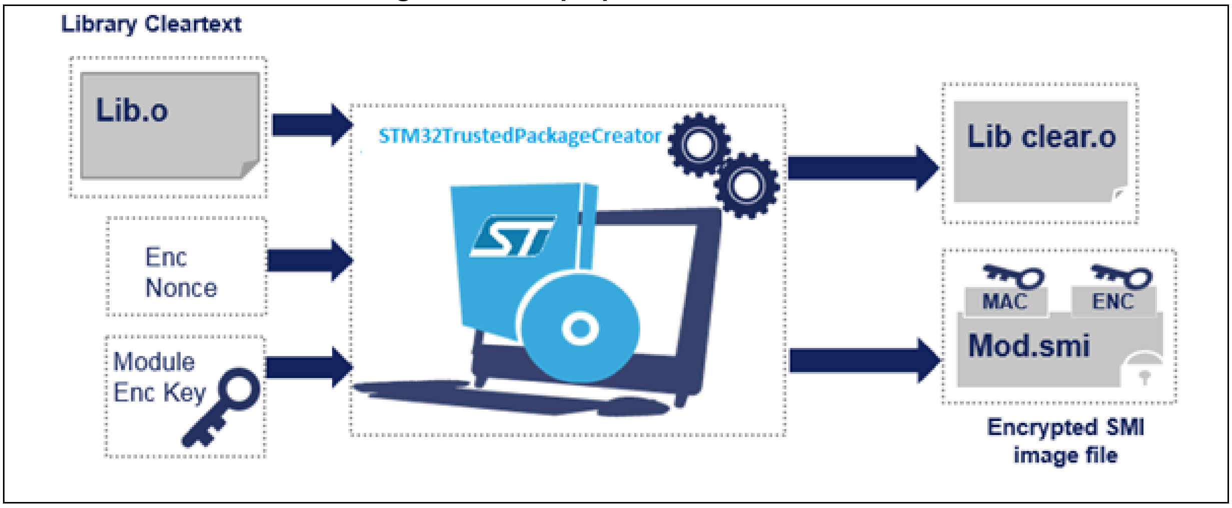


3.4 SMI generation process

SMI is a format created by STMicroelectronics that aims to protect partners' software (SW: software modules and libraries).

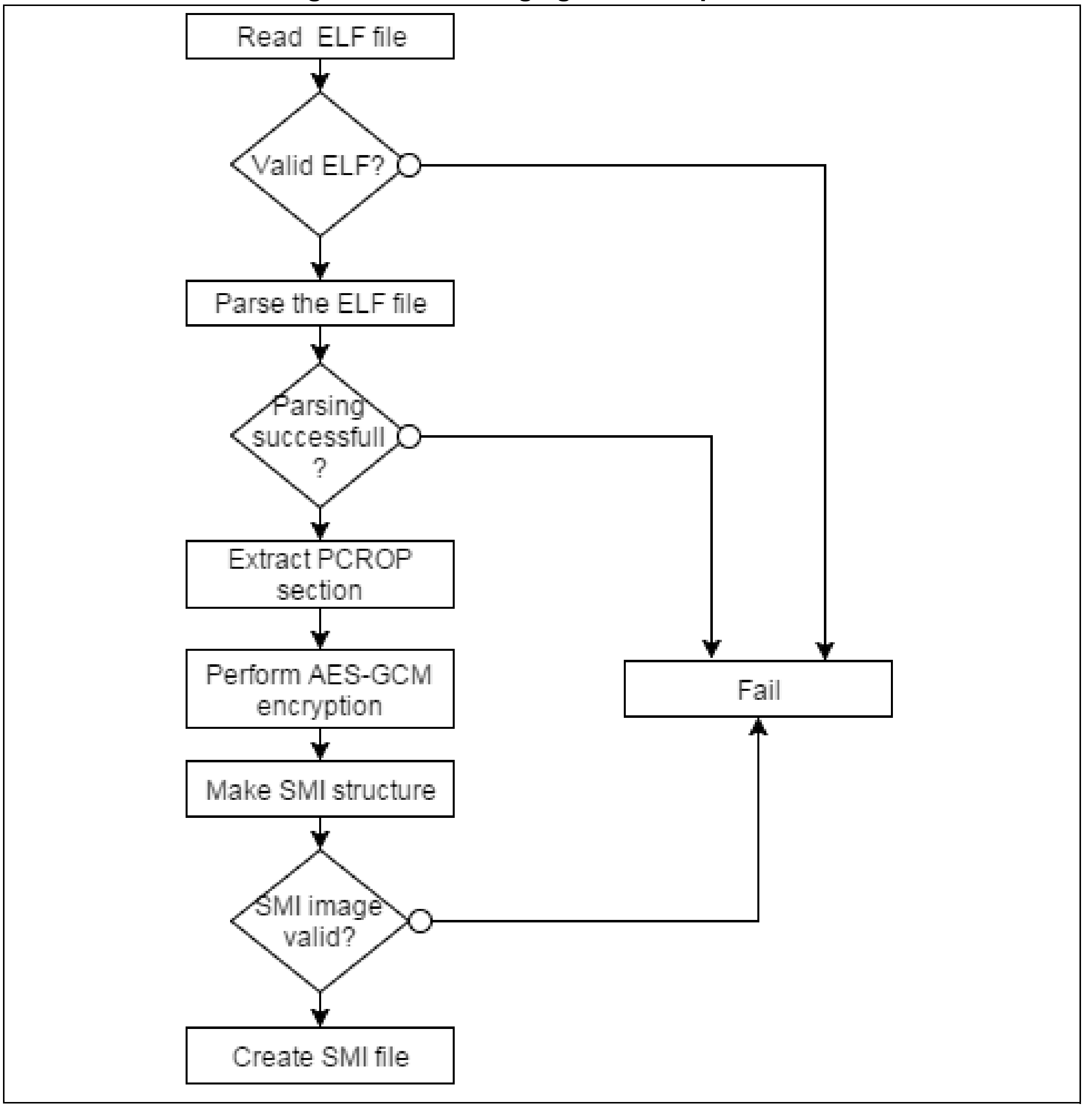
The SMI preparation process is described below (*Figure 20*).

Figure 20. SMI preparation mechanism



The SMI generation steps as currently implemented in the tool are described in the diagram below (*Figure 21*).

Figure 21. SMI image generation process



The AES-GCM encryption is performed using the following inputs:

- 128-bit AES encryption key
- The input nonce as Initialization Vector (IV)
- The security version as Additional Authenticated Data (AAD).

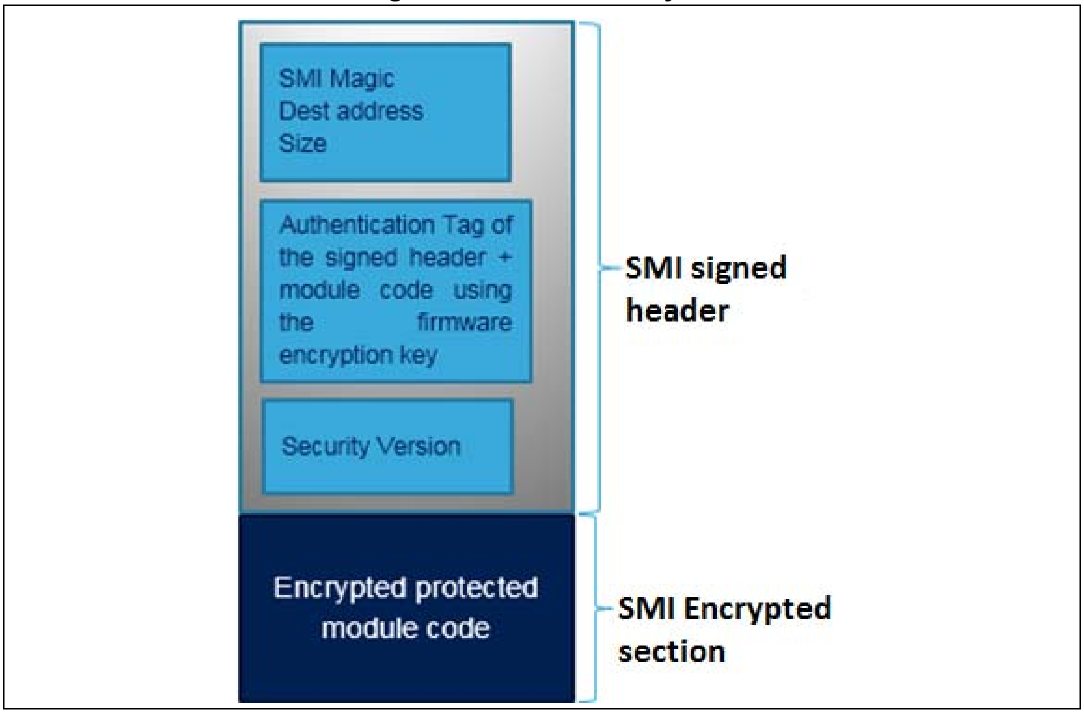
Before SMI image creation, PCROP checks are performed on the SMI image validity:

- A PCROP section must be aligned on a Flash word (256 bits), otherwise a warning is shown.
- The section's size must be at least 2 Flash words (512 bits), otherwise a warning is shown.
- The section must end on a Flash word boundary (a 256-bit word), otherwise a warning is shown.
- If the start address of the section immediately following the PCROP section overlaps the last Flash word of the PCROP section (after performing the PCROP alignment constraint), the generation fails and an error message appears.

If everything is OK, tow outputs are created under the specified path:

- The SMI image (*Figure 22* represents the SMI format layout).
- The library data part.

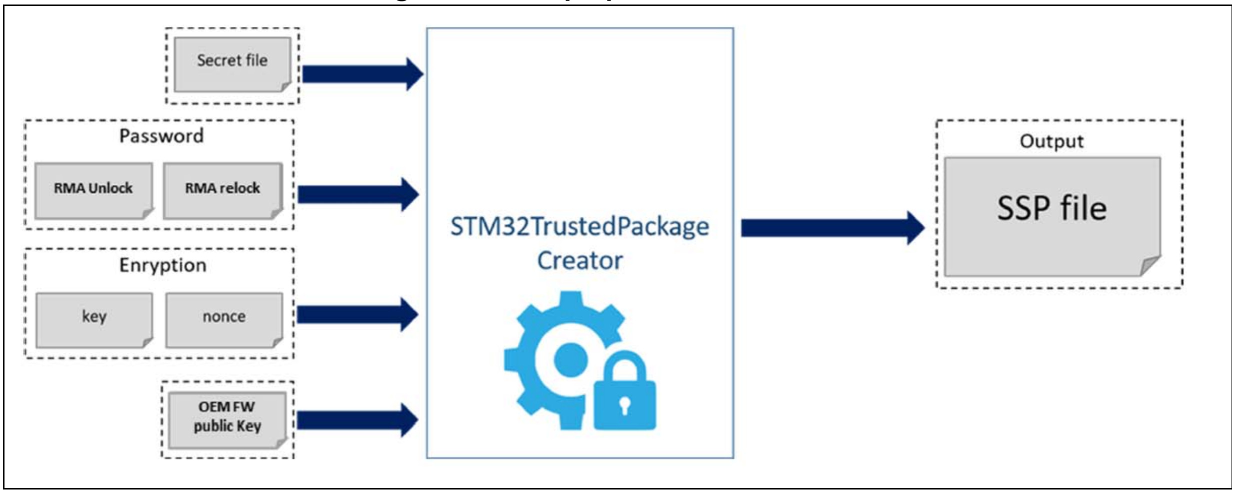
Figure 22. SMI format layout



3.5 SSP generation process

SSP is an encryption format that transforms customer secret files into encrypted and authenticated firmware using an AES-GCM algorithm with a 128-bit key. The SSP preparation process used in the STM32 Trusted Package Creator tool is shown in *Figure 23*.

Figure 23. SSP preparation mechanism



An SSP image must be created prior to SSP processing. The encrypted output file follows a specific layout that guarantees a secure transaction during transport and decryption based on the following inputs:

- **Secret file**: This 148-byte secret file must fit into the OTP area reserved for the customer. There is no tool or template to create this file.
- **RMA password**: This password is chosen by the OEM. It is part of the secret file and is placed as the first 4-byte word. To make RMA password creation easier and avoid issues, the STM32 Trusted Package Creator tool add sit directly at the beginning of the 148-byte secret file.
- **Encryption key**: AES encryption key (128 bits).
- **Encryption nonce**: AES nonce (128 bits).
- **OEM FW key**: This is the major part of the secure boot sequence. Based on ECDSA verification, the key is used to validate the signature of the loaded binary.

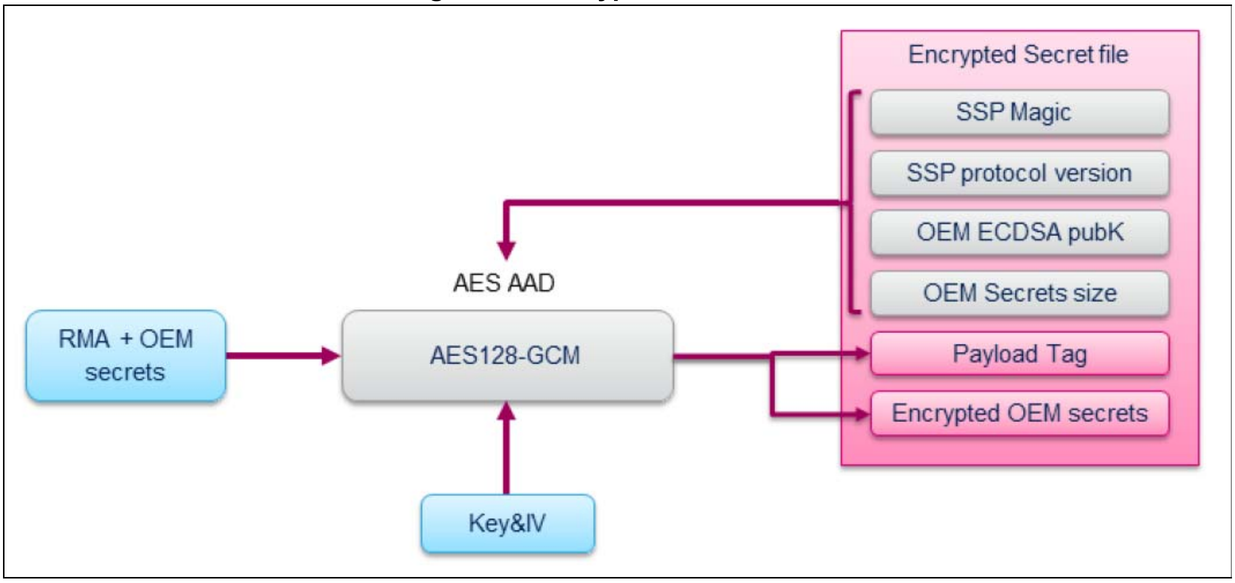
The first layout part (SSP magic, Protocol version, ECDSA public key, secret size) is used as additional authenticated data (AAD) to generate the payload tag. This is checked by the ROM code during decryption.

Table 2. SSP preparation inputs

Input	Size (bytes)	Content
SSP magic	4	‘SSPP’: magic identifier for SSP Payload
SSP Protocol Version	4	Can be used to indicate how to parse the payload, if payload format changes in future
OEM ECDSA public key	64	OEM ECDSA public key
OEM secret size	4	Size of OEM secrets, in bytes
Payload tag	16	Cryptographic signature of all fields above, to ensure their integrity.
Encrypted OEM secrets	152	Encrypted OEM secrets. 152 is given by previous field.

This encrypted file is automatically generated by the STM32 Trusted Package Creator tool.

Figure 24. Encryption file scheme



3.6 STM32 Trusted Package Creator tool in the command line interface

This section describes how to use the STM32 Trusted Package Creator tool from the command line interface in order to generate SFI/SFIx and SMI images. The available commands are listed in *Figure 25*.

Figure 25. STM32 Trusted Package Creator tool - available commands

```
SFI preparation options

 -sfi,   --sfi          : Generate SFI image,
                          You also need to provide the information listed below
     -fir,   --firmware    : Add an input firmware file
        <Firm_File>        : Supported firmware files are ELF HEX SREC BIN
     -firx,    --firmwx    : Add input for external firmware file
        <Firmx_File>       : Supported externalfirmware files are ELF HEX SREC BIN
        [<Address>]        : Only in case of BIN input file (in any base)
        [<Region_Number>]  : Only in case of BIN input file (in any base): [0:3]: OTFD1 (STM32H7A / STM32L5), [4:7]: OTFD2 (STM32H7A/B case)
        [<Region_Mode>]    : Only in case of BIN input file (in any base),only two bit [0:1] where 00 : instruction only (AES-CTR), 01 : data only
(AES-CTR), 10: instruction + data (AES-CTR), 11: instruction only (EnhancedCipher)
        [<key_address>]    : Only in case of BIN input file (in any base), random key values in internal flash memory
    -k,      --key         : AES-GCM encryption key
         <Key_File>        : Bin file, its size must be 16 bytes
    -kx,      --keyx       :  key area for external firmware
         <Key_area_File>   : CSV file contains a set of couple (size,start address)
    -n,      --nonce       : AES-GCM nonce
         <Nonce_File>      : Bin file, its size must be 12 bytes
    -v,      --ver         : Image version
         <Image_Version>   : Its value must be in <0..255> (in any base)
    -ob,     --obfile      : Option bytes configuration file
         <CSV_File>        : CSV file with 9 values
    -m,      --module      : Add an SMI file (optional for combined case)
        <SMI_File>         : SMI file
         [<Address>]       : Only in case of a relocatable SMI (with Address = 0)
    -rs,      --ramsize    : define available ram size (for multi-image)
         <Size>            : Size in bytes
    -ct,      --token      : Continuation token address (for multi-image)
      <Address>            : Address
    -o,      --outfile     : Generated SFI file
         <Output_File>     : SFI file to be created

SMI preparation options

 -smi,   --smi,            : Generate SMI image
                           You also need to provide the information listed below
    -elf,   --elfile       : Input ELF file
        <ELF_File>         : ELF file
    -s,      --sec         : Section to be encrypted
        <Section>          : Section name in the ELf file
    -k,      --key         : AES-GCM encryption key
        <Key_File>         : Bin file, its size must be 16 bytes
    -n,      --nonce       : AES-GCM nonce
        <Nonce_File>       : Bin file, its size must be 12 bytes
    -sv,     --sver        : Security version
        <SV_File>          : Its size must be 16 bytes
    -o,      --outfile     : Generated SMI file
        <Output_File>      : SMI file to be created
    -c,      --clear       : Clear ELF file
        <Clear_File>       : Clear ELF file to be generated
```

3.6.1 Steps for SFI generation (CLI)

In order to generate an SFI/SFIx image in CLI mode, the user must use the “-sfi, --sfi” command followed by the appropriate inputs.Inputs for “sfi” command are:

-fir, --firmware

Description: adds an input firmware file (supported formats are Bin, Hex, Srec and ELF). This option can be used more than once in order to add multiple firmware files.

Syntax: -fir <Firmware_file> [<Address>]

<Firmware_file>	:Firmware file.
[<Address>]	:Used only for binary firmware.

-firx, --firmwx

Description: Add an input for external firmware file. Supported formats are Bin, Hex, Srec and ELF. This option can be used more than once in order to add multiple firmware files.

Syntax: -firx <Firmware_file> [<Address>] [<Region_Number>] [<Region_Mode>] [<key_address>]

<Firmware_file>	: Supported external firmware files are ELF HEX SREC BIN.
[<Address>]	:Only in case of BIN input file (in any base).
<Region_Number>	: Only in case of BIN input file (in any base): [0:3]: OTFD1 (STM32H7A3/7B3, STM32H7B0 or STM32L5), [4:7]: OTFD2 (STM32H7A3/7B3 and STM32H7B0 case).
<Region_Mode>	: Only in case of BIN input file (in any base), only two bits [0:1] where 00: instruction only (AES-CTR) 01: data only (AES-CTR) 10: instruction + data (AES-CTR) 11: instruction only (EnhancedCipher)
<key_address>	: Only in case of BIN input file (in any base), random key values in internal Flash memory.

-k, --key

Description: sets the AES-GCM encryption key.

Syntax: -k <Key_file>

< Key _file>	: A 16 bytes binary file.

-n, --nonce

Description: sets the AES-GCM nonce.

Syntax: -n <Nonce_file>

<Nonce _file>	A 12-byte binary file.

-v, --ver

Description: sets the image version.

Syntax: -v <Image_version>

<Image_version> : A value between 0 and 255 in any base.

-ob, --obfile

Description: provides an option bytes configuration file.
The option bytes file field is only mandatory for SFI applications (first install) to allow option bytes programming, otherwise it is optional.
Only csv (comma separated value) file format is supported as input for this field, it is composed from two vectors: register name and register value respectively.

Note: The number of rows in the CSV file is product dependent (refer to the example available for each product). For instance there are 9 rows for all STM32H7 products, with the last row "reserved", except for dual-core devices. It is important to neither change the order of, nor delete, rows.

Example: for STM32H7xx devices, 9 option bytes registers must be configured, which corresponds to a total of 9 lines in the csv file (*Figure 26*).

Syntax: -ob <CSV_file>

<CSV_file >: A csv file with 9 values.

Figure 26. Option bytes file example

```
FOPTSR_PRG,0x1026AAD0
FPRAR_PRG_A,0x81000200
FPRAR_PRG_B,0x81000200
FSCAR_PRG_A,0x81000200
FSCAR_PRG_B,0x81000200
FWPSN_PRG_A,0xFFFFFFFF
FWPSN_PRG_B,0xFFFFFFFF
FBOOT7_PRG,0x24000800
RESERVED,0x10000810
```

-m, --module

Description: adds an input SMI file.
This option can be used more than once in order to add multiple SMI files.
This is optional (used only for combined SFI-SMI).

Syntax: -m <SMI_file>

<SMI_file >: SMI file.[<Address>]: Address is provided only for relocatable SMI.

-rs, --ramsize

Description: define the available ram size (in case of SFI multi-install)

Syntax: -rs <Size>

< Size >: RAM available size in bytes

Note: The maximum RAM size of each device is mentioned in the descriptor. For example the maximum RAM size of the STM32WL is 20 Kbytes.

-ct, --token

Description: continuation token address (in case of SFI multi-install)

Syntax: -ct <Address>

< Address >: continuation token Flash address

-o, --outfile

Description: sets the output SFI file to be created.

Syntax: -o <out_file>

<out_file > : the SFI file to be generated (must have the ".sfi" extension).

Example of SFI generation command using an ELF file:

```
STM32TrustPackageCreator_CLI.exe -sfi -fir tests.axf -k
test_firmware_key.bin -n nonce.bin -ob ob.csv -v 23 -o out.sfi
```

The result of previous command is shown in *Figure 27*.

Figure 27. SFI generation example using an Elf file

3.6.2 Steps for SMI generation (CLI)

In order to generate an SMI image in CLI mode, the user must use the "-smi, --smi" command followed by the appropriate inputs.

Inputs for the "smi" command are:

-elf, --elfile

Description: sets the input ELF file (only ELF format is supported).

Syntax: - elf <ELF_file>

<ELF_file> : ELF file. An ELF file can have any of the extensions: ".elf", ".axf", ".o", ".so",".out".

-s, --sec

Description: sets the name of the section to be encrypted.

Syntax: -s <section_name>

<section_name> : Section name.

-k, --key

Description: sets the AES-GCM encryption key.

Syntax: -k <Key_file>

< Key _file> : A 16-byte binary file.

-n, --nonce

Description: sets the AES-GCM nonce.

Syntax: -n <Nonce_file>

<Nonce_file> : A 12-byte binary file.

-sv, --sver

Description: sets the security version file

The security version file is used to make the SMI image under preparation compatible with a given RSS version, since it contains a corresponding identifying code (almost the HASH of the RSS).

Syntax: -sv <**SV_file**>

<SV_file> : A 16-byte file.

-o, --outfile

Description: Sets the SMI file to be created as output

Syntax: -o <out_file>

<out_file > : SMI file to be generated, must have the .smi extension.

-c, --clear

Description: Sets the clear ELF file to be created as output corresponding to the data part of the input file

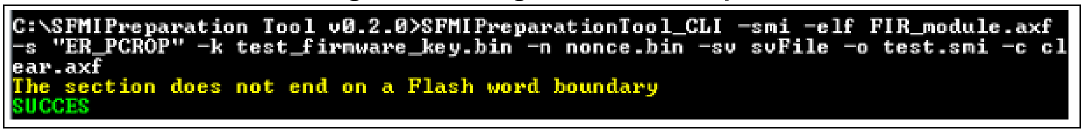
Syntax: -c <ELF_file>

<ELF_file > : Clear ELF file to be generated.

Example SMI generation command:

```
STM32TrustPackageCreator_CLI.exe -smi -elf FIR_module.axf -s "ER_PCROP" -k test_firmware_key.bin -n nonce.bin -sv svFile -o test.smi -c clear.smi
```

Figure 28. SMI generation example



3.6.3 Steps for SSP generation (CLI)

In order to generate an SSP image in CLI mode, the user must use the “-ssp, --ssp” command followed by the appropriate inputs.

Inputs for the “ssp” command are:

-ru, --rma_unlock

Description: RMA unlock password

Syntax: -ru <RMA_Unlock>

<RMA_Unlock> : Hexadecimal value 0x0000 to 0x7FFF

-rr, --rma_relock

Description: RMA relock password

Syntax: -rr <relock_value>

<relock_value> : Hexadecimal value 0x0000 to 0x7FFF

-b, --blob

Description: Binary to encrypt

Syntax: -b <Blob>

<Blob> : Secrets file of size 148 bytes

-pk, --pubk

Description: OEM public key file

Syntax: -pk <PubK.pem>

<PubK> : pem file of size 178 bytes

-k, --key

Description: AES-GCM encryption key

Syntax: -k <Key_File>

<Key_File> : Bin file, its size must be 16 bytes

-n, --nonce

Description: AES-GCM nonce

Syntax: -n <Nonce_File>

<Nonce_File> : Bin file, its size must be 16 bytes

-o, --out

Description: Generate SSP file

Syntax: -out <Output_File.ssp>

<Output_File> : SSP file to be created with (extension .ssp)

If all input fields are validated, an SSP file is generated in the directory path already mentioned in the “-o” option.

Example SSP generation command:

STM32TrustedPackageCreator_CLI -ssp -ru 0x312 -rr 0xECA

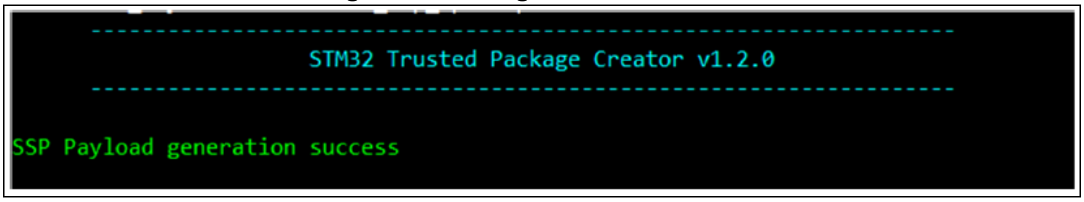
-b “C:\SSP\secrets\secrets.bin”

-pk “C:\SSP\OEMPublicKey.pem” -k “C:\SSP\key.bin”

-n “C:\SSP\nonce.bin” -o “C:\out.ssp”

Once the operation is done, a green message is displayed to indicate that the generation was finished successfully. Otherwise, an error occurred.

Figure 29. SSP generation success



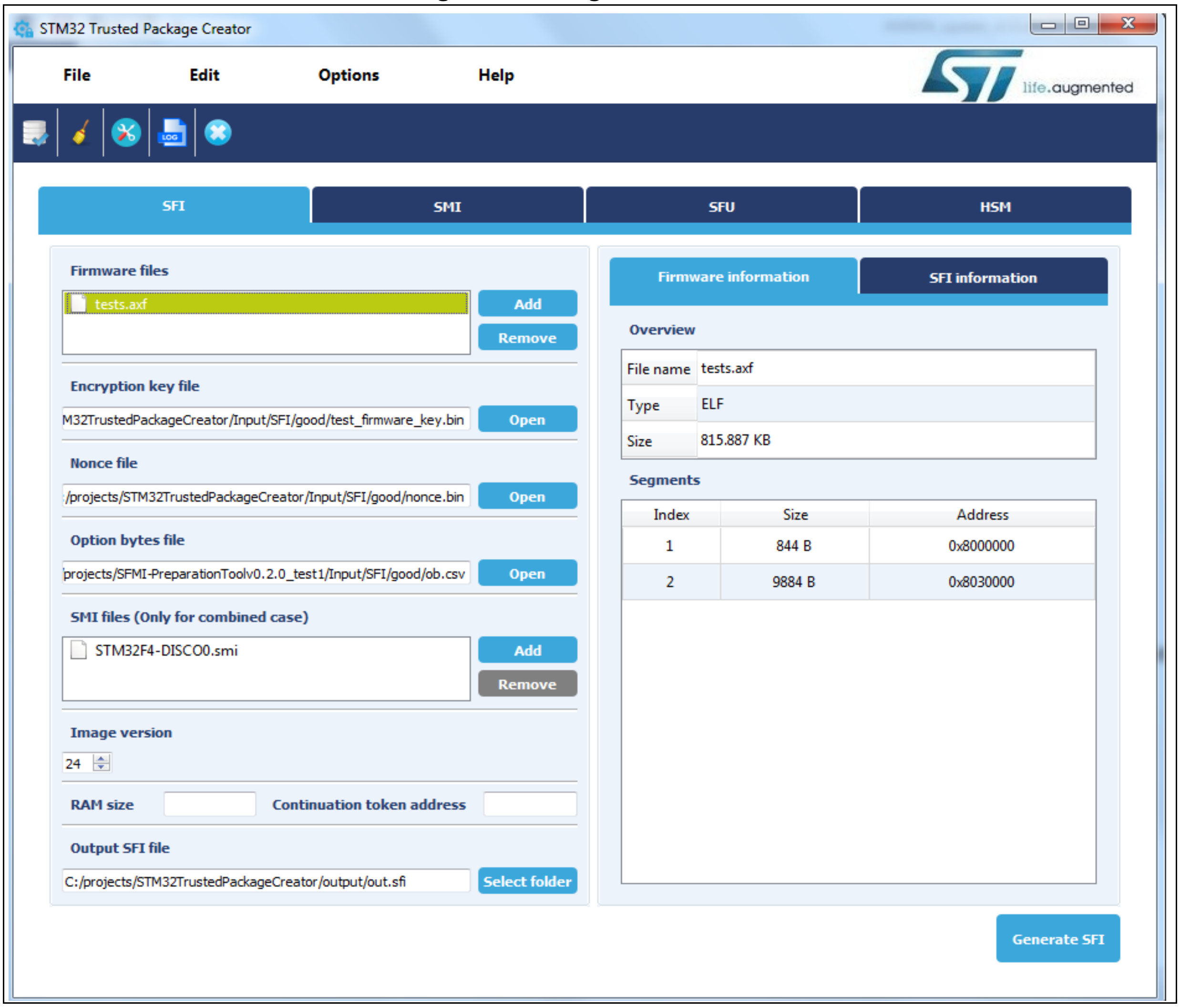
3.7 Using the STM32 Trusted Package Creator tool graphical user interface

The STPC is also available in graphical mode, this section describes its use. The STM32 Trusted Package Creator tool GUI presents two tabs, one for SFI generation, one for SFIx generation and one for SMI generation.

3.7.1 SFI generation using STPC in GUI mode

Figure 30 shows the graphical user interface tab corresponding to SFI generation.

Figure 30. SFI generation Tab



To generate an SFI image successfully from the supported input firmwares formats, the user must fill in the interface fields with valid values.

SFI GUI tab fields

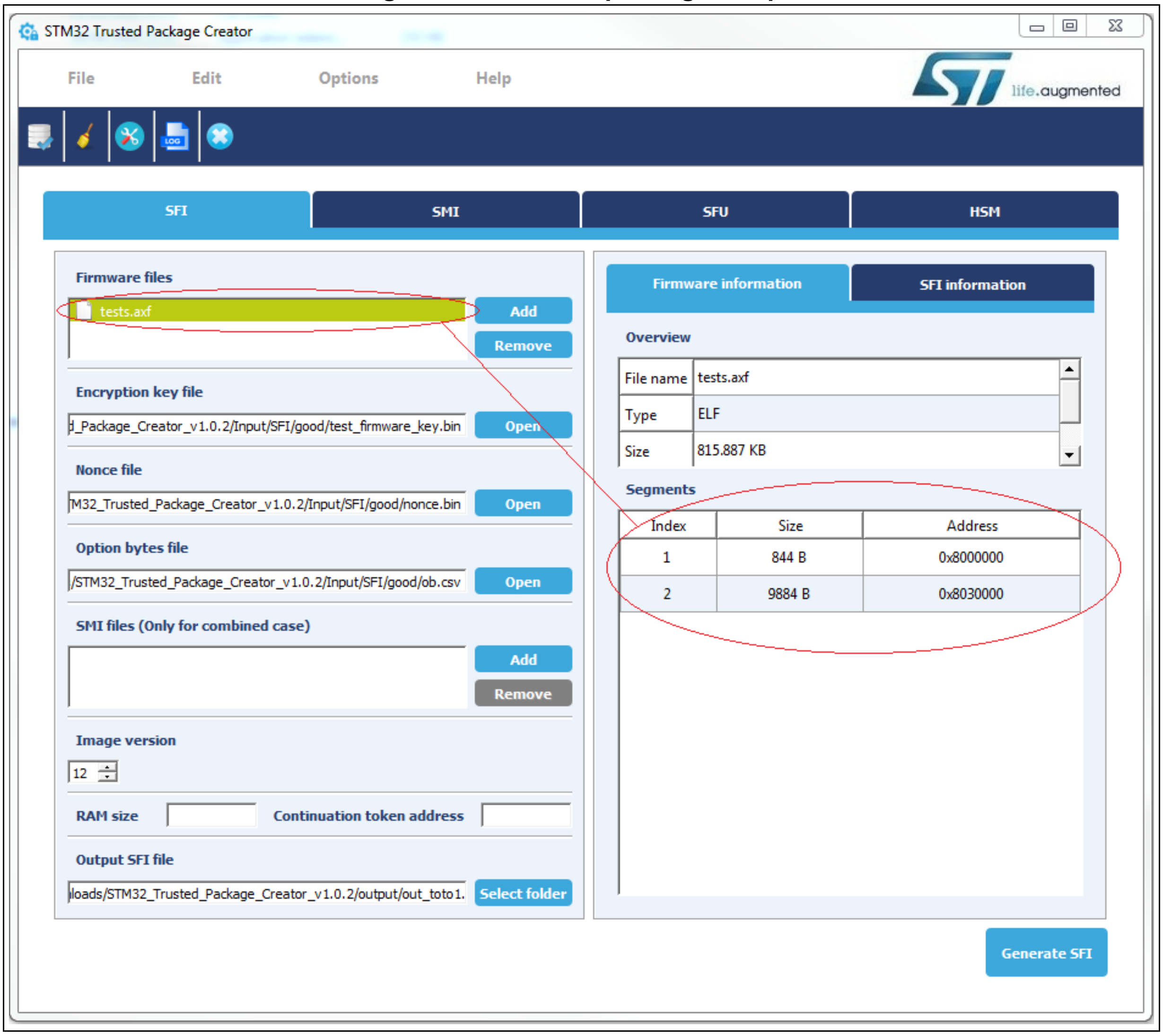
- Firmware files:

  The user needs to add the input firmware files with the "Add" button.

  If the file is valid, it is appended to the "input firmware files" list, otherwise an error message box appears notifying the user that either the file could not be opened, or the file is not valid.

  Clicking on "input firmware file" causes information related information to appear in the "Firmware information" section (*Figure 31*).

Figure 31. Firmware parsing example



- Encryption key and nonce file:

  The encryption key and nonce file are selected by entering their paths (absolute or relative), or by selecting them with the “Open” button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).
- Option bytes file:

  The option bytes file are selected the same way as the encryption key and nonce. Only csv files are supported.

Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

- SMI files:

  SMI files are added the same way as the firmware files. Selecting a file causes related information to appear in the “Firmware information” section.
- Image version:

  Choose the image version value of the SFI under generation within this interval: [0..255].
- Output file:

  Sets the folder path in which the SFI image is to be created. This is done by entering the folder path (absolute or relative) or by using the “Select folder” button.

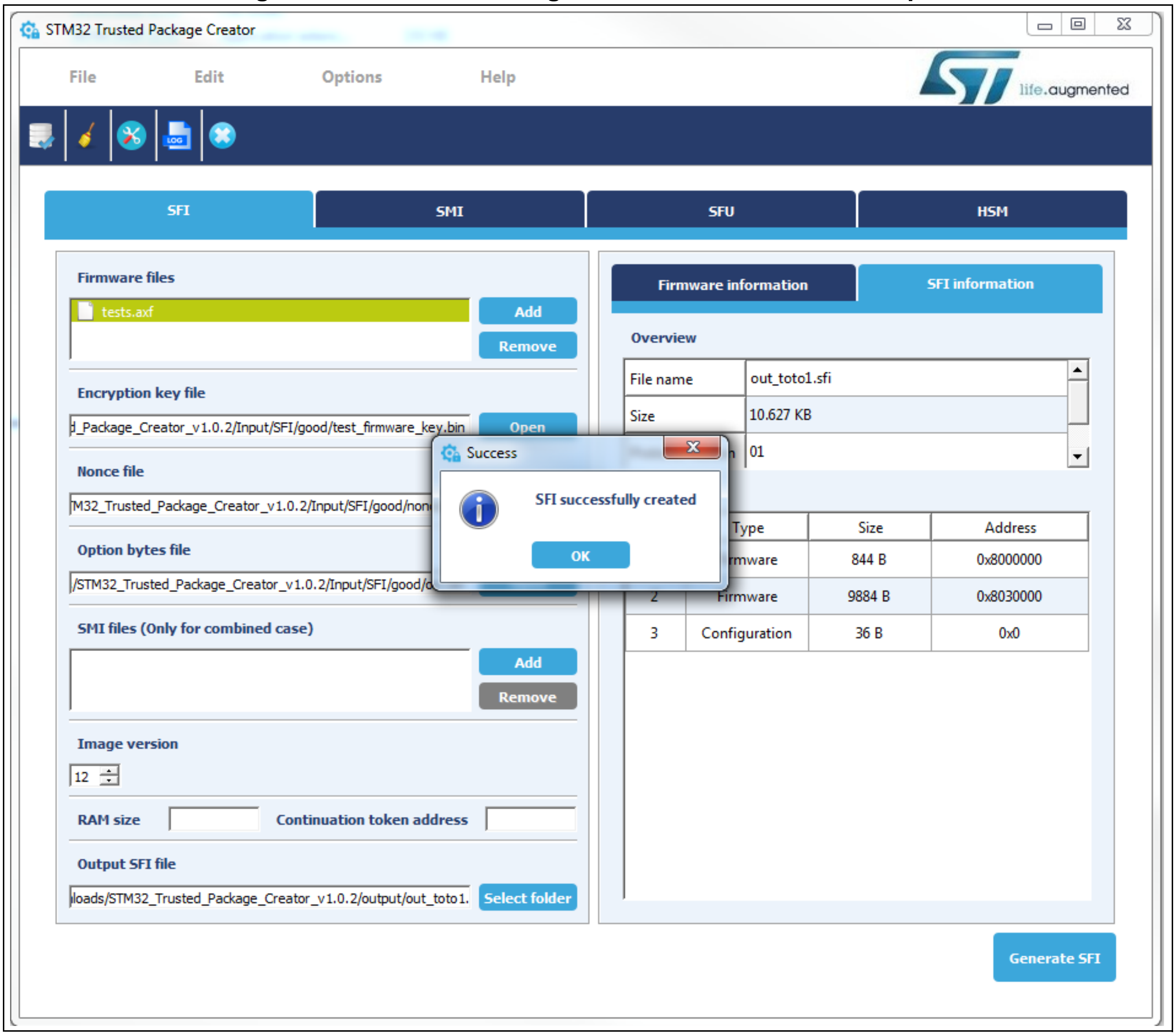
Note: By using the “Select folder” button, the name “out.sfi” is automatically suggested. This can be kept or changed.

- ‘Generate SFI’ button:

  Once all fields are filled in properly, the “Generate SFI” button becomes enabled. The user can generate the SFI file by a single click on it.

  If everything goes well, a message box indicating successful generation appears (*Figure 32: SFI successful generation in GUI mode example*) and information about the generated SFI file is displayed in the SFI information section.

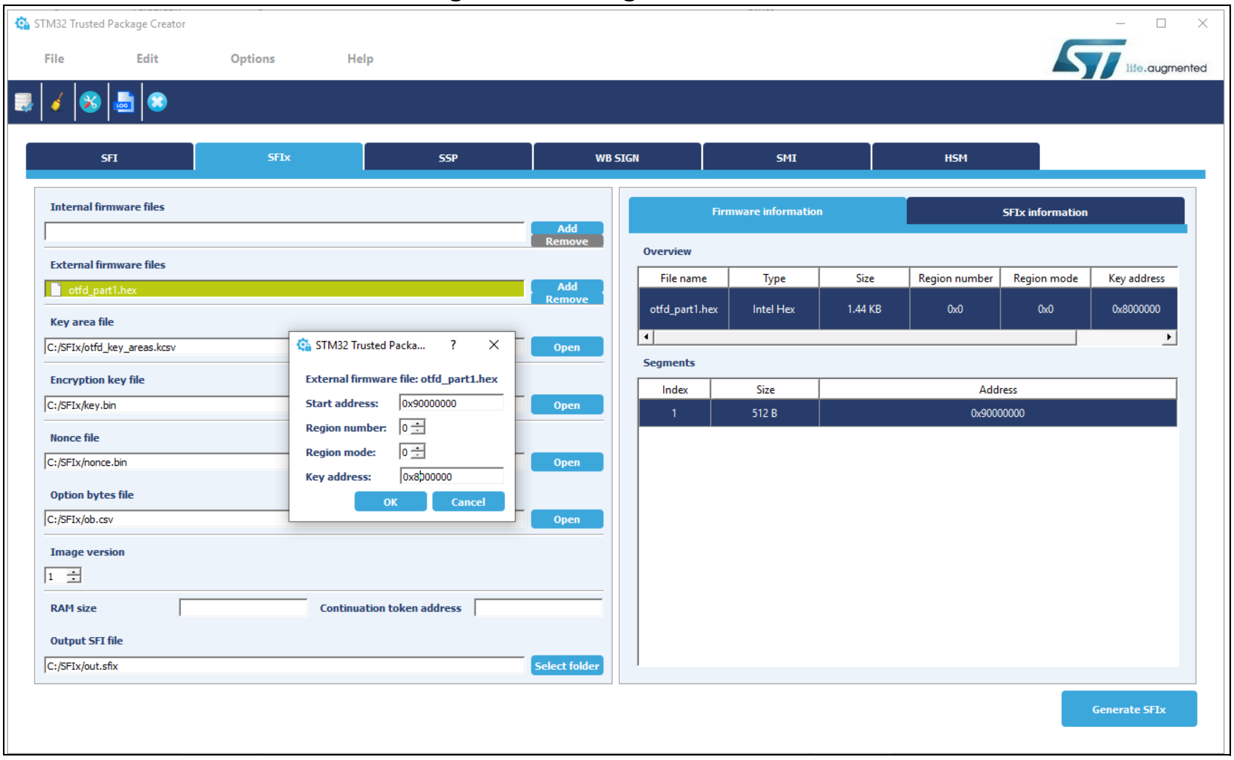
Figure 32. SFI successful generation in GUI mode example



3.7.2 SFIx generation using STPC in GUI mode

Figure 33 shows the graphical user interface tab corresponding to SFIx generation.

Figure 33. SFIx generation Tab

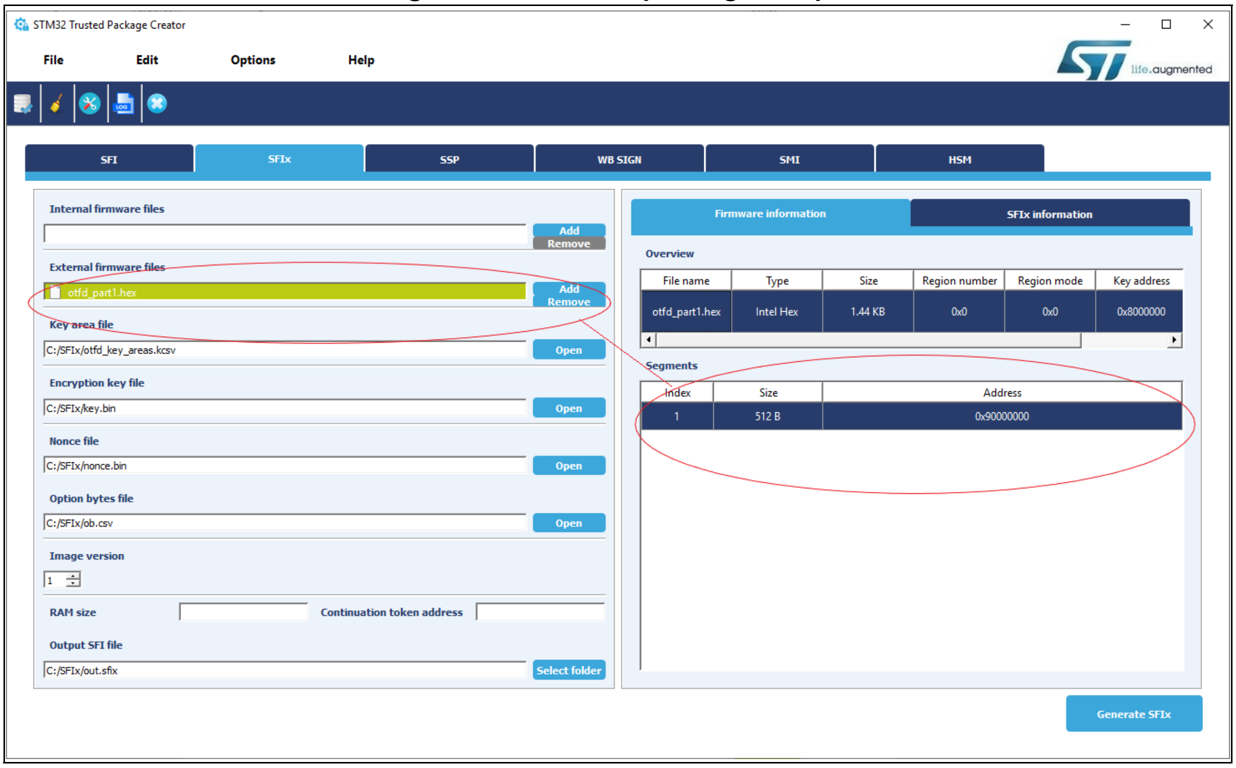


To generate an SFIx image successfully from the supported input firmware formats, the user must fill in the interface fields with valid values.

SFIx GUI tab fields

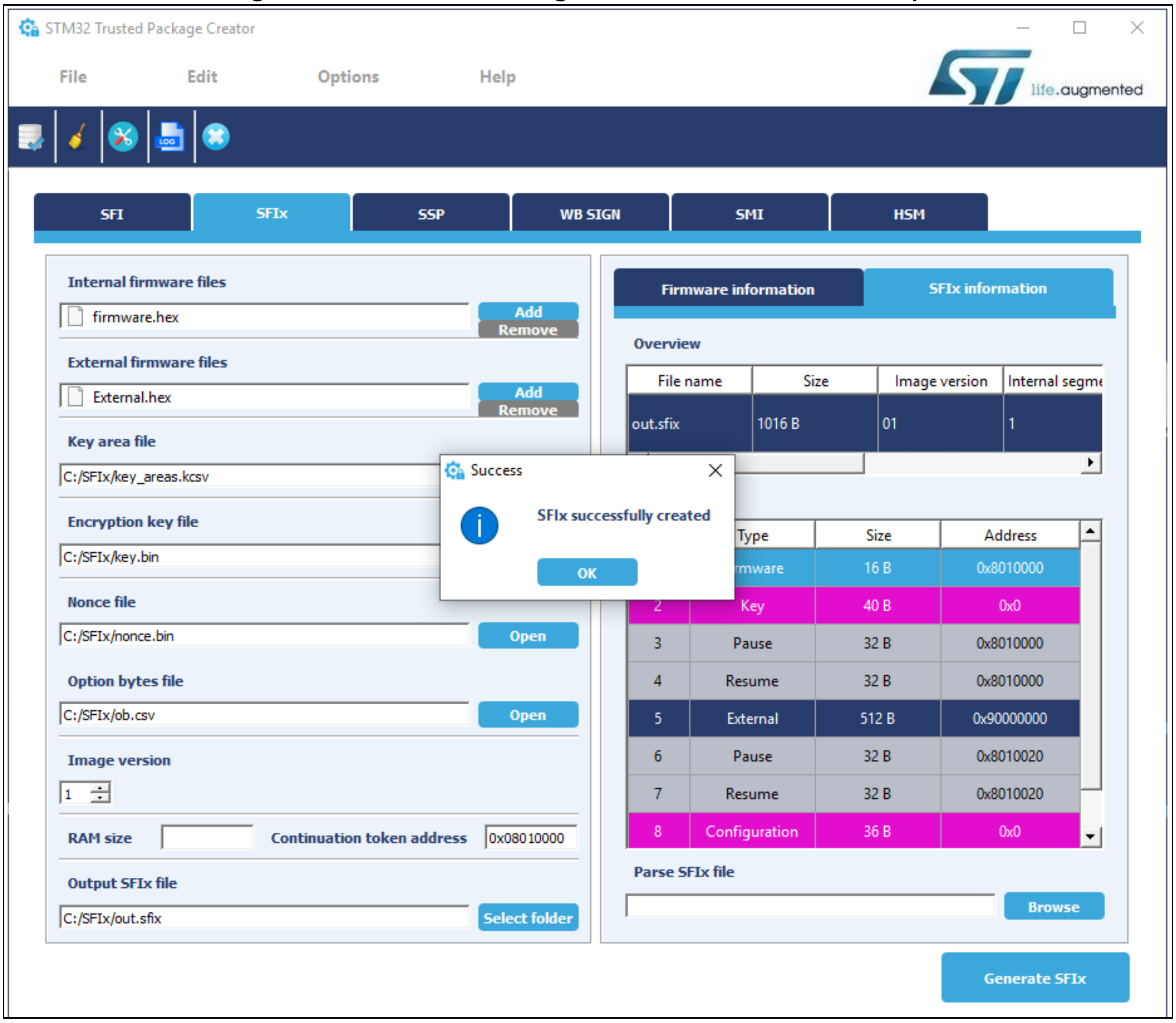
Firmware files: The user needs to add the input firmware files with the “Add” button. If the file is valid, it is appended to the “input firmware files “list, otherwise an error message box appears notifying the user that either the file could not be opened, or the file is not valid. Clicking on “input firmware file“ causes information related information to appear in the “Firmware information” section (*Figure 34*).

Figure 34. Firmware parsing example



As is the case for the SFI use case, once all fields are filled in properly, the “Generate SFIx” button becomes enabled. The user can generate the SFIx file by a single click on it. If everything goes well, a message box indicating successful generation appears (*Figure 35: SFIx successful generation in GUI mode example*) and information about the generated SFIx file is displayed in the SFIx information section.

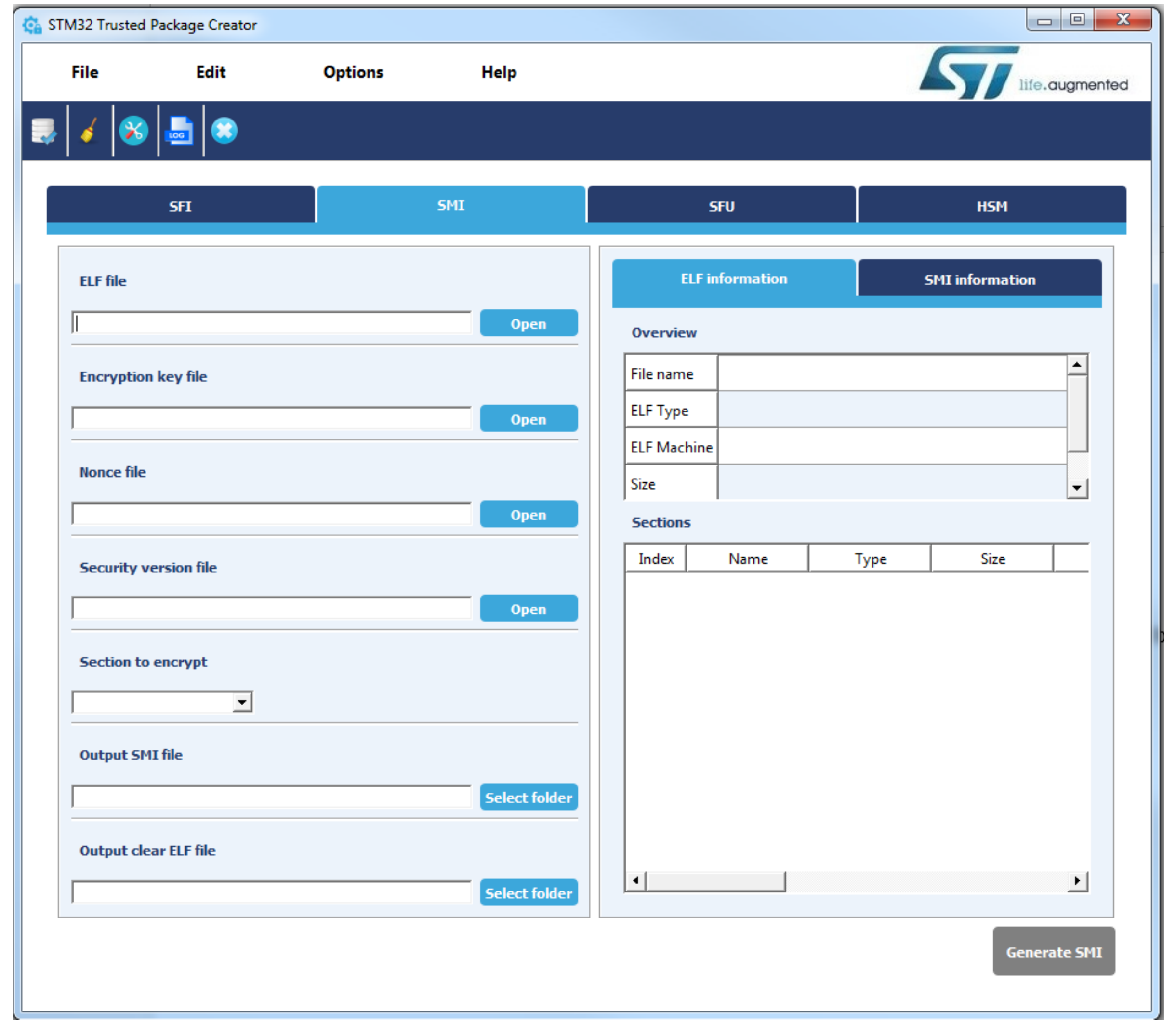
Figure 35. SFIx successful generation in GUI mode example



3.7.3 SMI generation using STPC in GUI mode

Figure 36 shows the graphical user interface tab corresponding to SMI generation.

Figure 36. SMI generation Tab



To generate an SMI image successfully from an Elf file, the user must fill in the interface fields with valid values.

SMI GUI tab fields

- Elf file:

  In this case the input file can only be an elf file.

  If the file is valid, information is displayed in the “ELF information” tab, otherwise an error message box appears notifying the user that either the file could not be opened or the file is not valid.
- Encryption key and nonce file:

  As for SFI, the encryption key and nonce file are selected in the same way as the Elf file. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce file).
- Security version file:

  The security version file is used for the same purpose as explained in the CLI section.

  The security version file size must be 16 bytes.
- Section:

  This is a section list that can be used to select the name of the section to be encrypted.
- Output files:

  Sets the folder path into which the SMI image and its clear part are to be created. This is done by entering the folder path (absolute or relative) or by using the “Select folder” button.

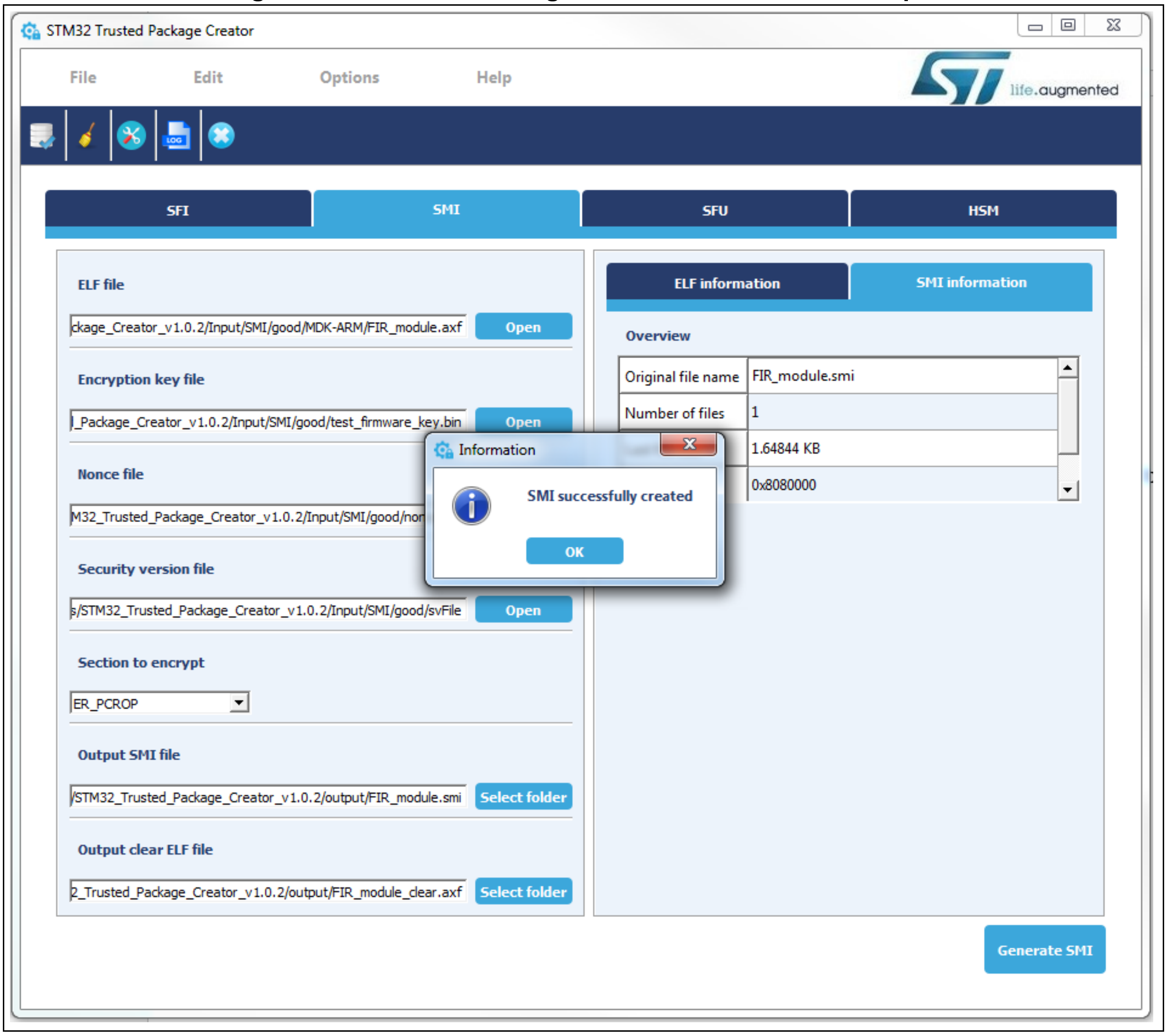
Note: For both output fields, when using the “Select folder” button, a name is suggested automatically. This can be kept or changed.

- ‘Generate SMI’ button:

  When all fields are filled in properly the ‘Generate SMI’ button is enabled, and the user can generate the SMI file and its corresponding clear data part by a single click on it.

  A message box informing the user that generation was successful must appear (*Figure 37: SMI successful generation in GUI mode example*), with additional information about the generated SMI file displayed in the ‘SMI information’ section. In the case of invalid input data, an error message box appears instead.

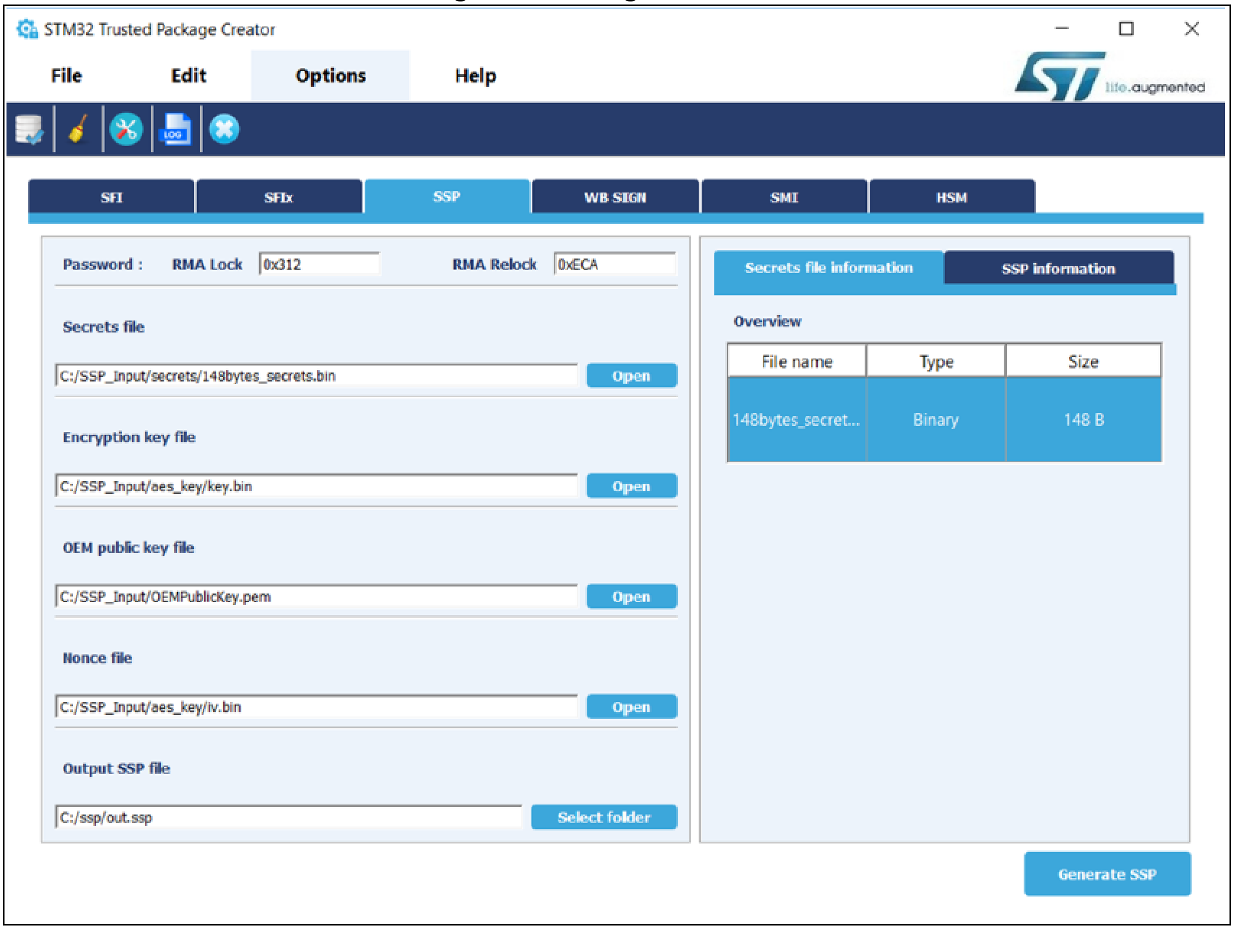
Figure 37. SMI successful generation in GUI mode example



3.7.4 SSP generation using STPC in GUI mode

Figure 38 shows the SSP generation graphical user interface tab.

Figure 38. SSP generation tab



To generate an SSP image successfully from the supported firmware input formats, the user must fill in the interface fields with valid values.

SSP GUI tab fields

RMA Lock: Unlock password, hexadecimal value from 0x0000 to 0x7FFF

RMA Relock: Relock password, hexadecimal value from 0x0000 to 0x7FFF

Secrets file: Binary file of size 148 bytes to be encrypted. Can be selected by entering file path (absolute or relative), or by selection with the **Open** button.

Encryption key and nonce files: The encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selection with the **Open** button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).

OEM public key file: 178-byte .pem file.

Output SSP file: Select the output directory by entering the SSP file name to be created with a .ssp extension.

When all fields are properly filled in, the user can start the generation by clicking on the **Generate SSP** button (the button becomes active).

Figure 39. SSP output information

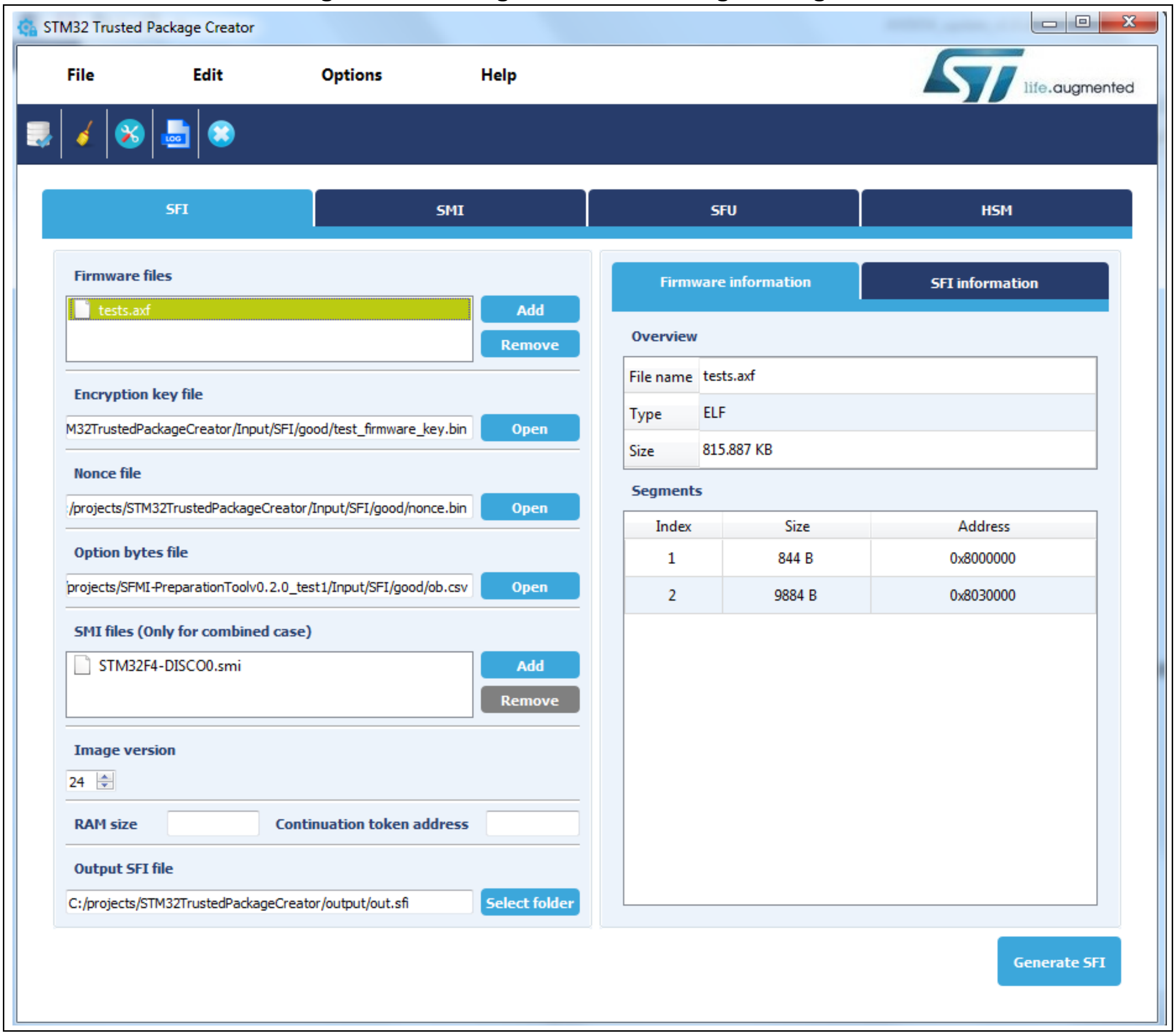
When the generation is complete, SSP information is available in the SSP overview section.

- **File name**: SSP output file name.
- **Type**: SSP format.
- **Size**: indicates the generated file size including all data fields.

3.7.5 Settings

The STPC allows generation to be performed respecting some user-defined settings. The settings dialog are displayed by clicking the settings icon (see *Figure 40*) in the tool bar or in the menu bar by choosing: Options -> settings.

Figure 40. Settings icon and Settings dialog box



Settings can be performed on:

- Padding byte:

  When parsing Hex and Srec files, padding can be added to fill gaps between close segments in order to merge them and reduce the number of segments. The user might choose to perform padding either with 0xFF (default value) or 0x00.
- Settings file:

  When checked, a *"settings.ini"* file is generated in the executable folder. It saves the application state: window size and fields contents.
- Log file:

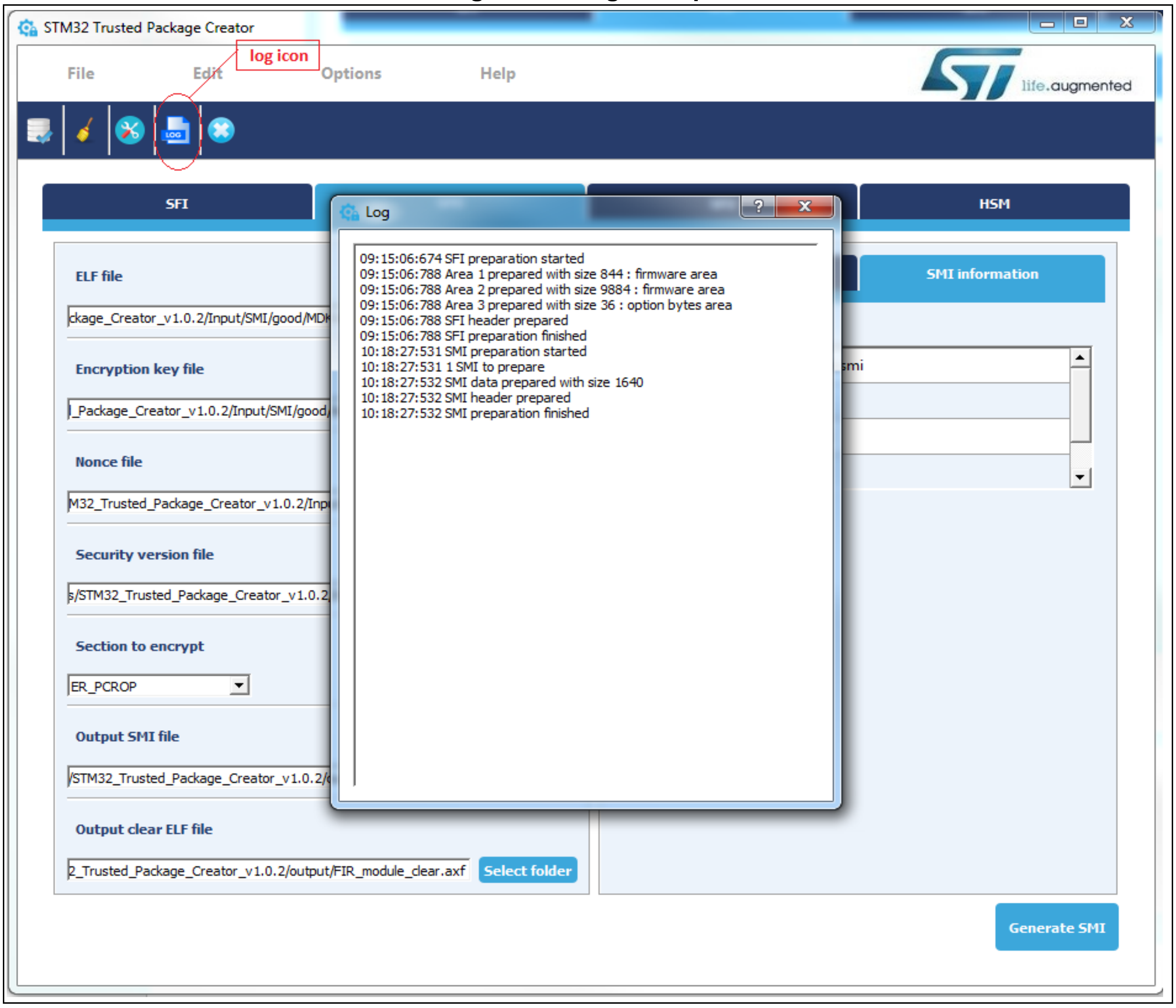
  When checked, a log file is generated in the selected path.

3.7.6 Log generation

A log can be visualized by clicking the “log” icon in the tool bar or menu bar: Options-> log.

Figure 41 shows a log example:

Figure 41. Log example



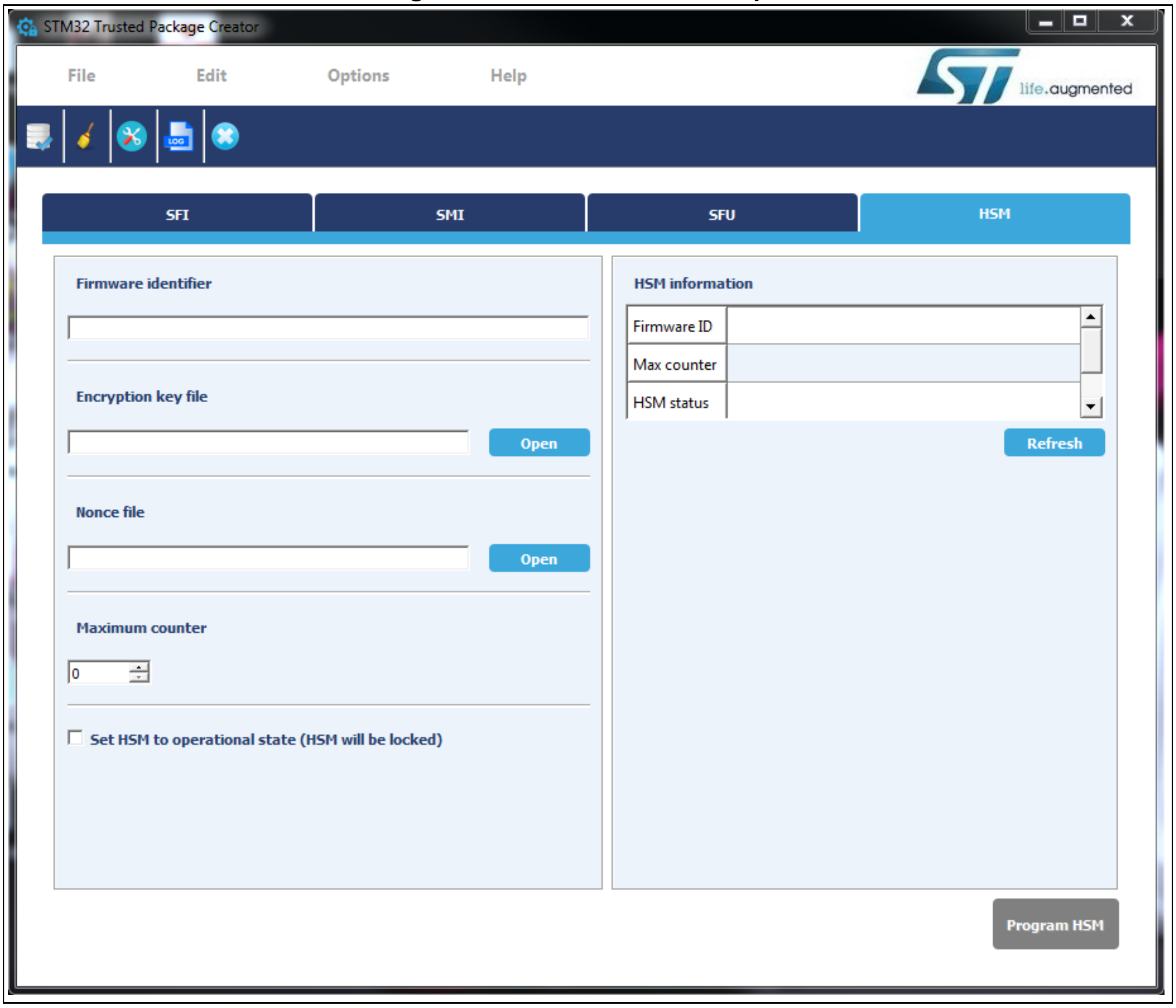
3.7.7 SFI and SMI file checking function

This function checks the validity and information parsing of an SFI or SMI file.

It is accessed by clicking the Check SFI/SMI button in the tool bar or the menu bar: File -> Check SFI/SMI.

Figure 42 shows a check SFI example:

Figure 42. Check SFI file example



4 Encrypted firmware (SFI/SFIx)/ module (SMI) programming with STM32CubeProgrammer

STM32CubeProgrammer is a tool for programming STM32 devices through UART, USB, SPI, CAN, I2C, JTAG and SWD interfaces. So far, programming via JTAG/SWD is only supported with an ST-LINK probe.

The STM32CubeProgrammer tool currently also supports secure programming of SFI and SMI images using UART, USB, SPI, JTAG/SWD interfaces, and SFIx using only JTAG/SWD interfaces.The tool is currently available only in CLI mode, it is available free of charge from *www.st.com*.

4.1 Chip certificate authenticity check and license mechanism

The SFI solution was implemented to provide a practical level of IP protection chain from the firmware development up to Flashing the device, and to attain this objective, security assets are used, specifically device authentication and license mechanisms.

4.1.1 Device authentication

The device authentication is guaranteed by the device’s own key.

In fact, a certificate is related to the device’s public key and is used to authenticate this public key in an asymmetric transfer: the certificate is the public key signed by a Certificate Authority (CA) private key. (This CA is considered as fully trusted).

This asset is used to counteract usurpation by any attacker who could substitute the public key with their own key.

4.1.2 License mechanism

One important secure Flashing feature is the ability of the firmware provider to control the number of chips that can be programmed. This is where the concept of licenses comes in to play. The license is an encrypted version of the firmware key, unique to each device and session. It is computed by a derivation function from the device’s own key and a random number chosen from each session (the nonce).

Using this license mechanism, the OEM is able to control the number of devices to be programmed, since each license is specific to a unique chip, identified by its public key.

License mechanism general scheme

When a firmware provider wants to distribute new firmware, they generate a firmware key and use it to encrypt the firmware.

When a customer wants to download the firmware to a chip, they send a chip identifier to the provider server, HSM or any provider license generator tool, which returns a license for the identified chip. The license contains the encrypted firmware key, and only this chip can decrypt it.

License distribution

There are many possible ways for the firmware provider to generate and distribute licenses:

- Server based: an Internet server can be set up, and when a customer needs to Flash the firmware on to a chip, they connect to the server which generates a license for this chip.
- HSM based: Hardware security modules can be built, one of which is installed on the programming house production line.
- Licenses can be generated in advance (but the firmware provider must know which chips to generate licenses for).

There is no STMicroelectronics secret involved in license generation, so each firmware provider is free to choose their preferred method.

ST offers an SFI solution based on smartcard HSMs as a license distribution tool, which can be used in programming houses.

HSM programming by OEM for license distribution

Before an OEM delivers an HSM to a programming house for deployment as a license generation tool for programming of relevant STM32 devices, some customization of the HSM must be done first.

The HSM needs to be programmed with all the data needed for the license scheme deployment. In the production line, a dedicated API is available for HSM personalization and provisioning.

This data is as follows:

- **The counter**: the counter is set to a maximum value that corresponds to the maximum number of licenses that could be delivered by the HSM. It aims to prevent over-programming.

  It is decremented with each license delivered by the HSM.

  No more licenses are delivered by the HSM once the counter is equal to zero.

  The maximum counter value must not exceed a maximum predefined value, which depends on the HSM used.
- **The firmware key**: the key size is 32 bytes. It is composed of two fields: the initialization vector field (IV) and the key field, which are used for AES128-GCM firmware encryption.

  Both fields are 16 bytes long, but the last 4 bytes of the IV must be zero (only 96 bits of IV are used in the AES128-GCM algorithm).

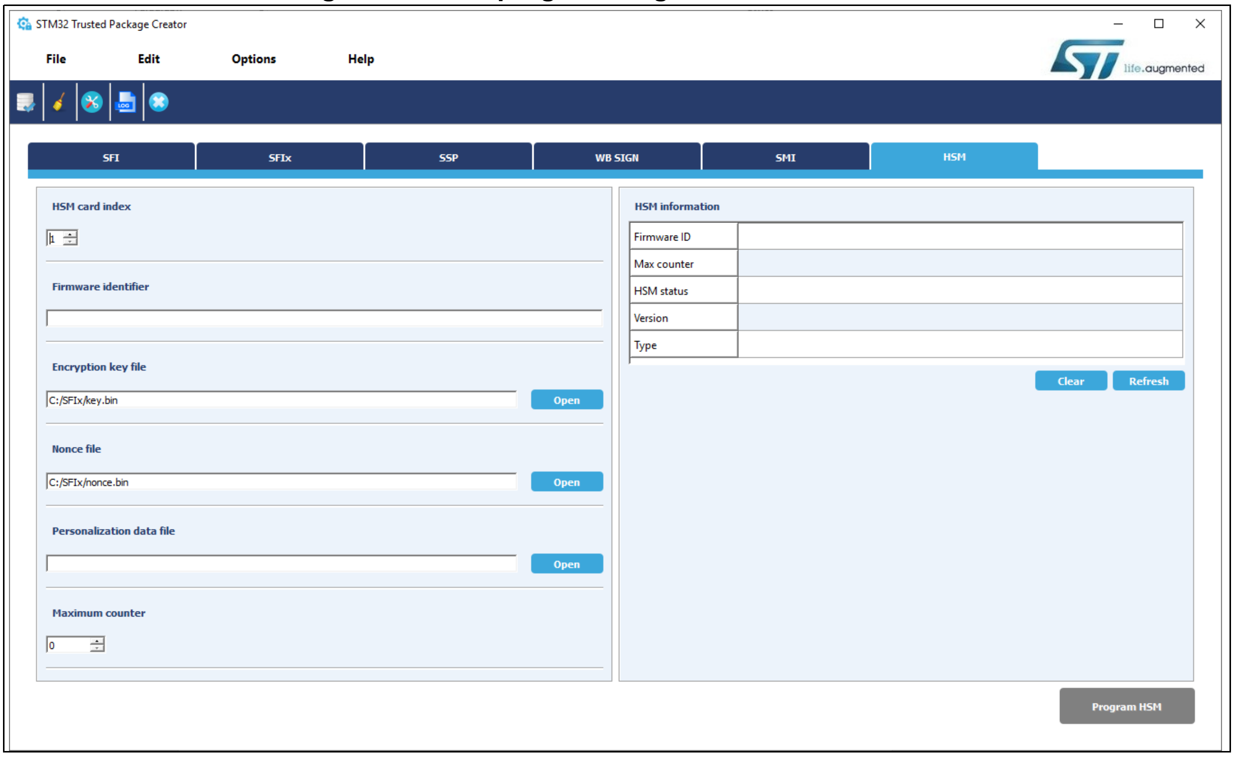
  Both fields must remain secret; that's why there are encrypted before being sent to the chip.

  The key and IV remains the same for all licenses for a given piece of firmware. However, they must be different for different firmware or different versions of the same firmware.
- **The firmware identifier**: allows the correct HSM to be identified for a given firmware.
- **The personalization dat**a: this is specific to each MCU and delivered inside TPC directory. More info about personalization data in *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.

The HSM must be in “OPERATIONAL STATE” (locked) when shipped by the OEM to guarantee his data confidentiality and privacy.

ST provides the tools needed to support SFI/SFIx via HSM. In fact, HSM programming is supported by the STM32 Trusted Package Creator tool. *Figure 43* shows the GUI for HSM programming in STPC tool.

Figure 43. HSM programming GUI in the STPC tool



During SFI install, STM32CubeProgrammer communicates with the device to get the chip certificate, upload it into the HSM to request the license. Once the license is generated by the HSM, it gives it back to the STM32 device.

4.2 Secure programming using a bootloader interface

4.2.1 Secure firmware installation using a bootloader interface flow

The production equipment on the OEM-CM production line needs to be equipped with a Flashing tool (FT) supporting the programming of SFI images. The Flashing tool to be used on OEM-CM production line is STM32CubeProgrammer, which is given the data blob prepared by the STPC, containing the image header and the encrypted image data blob.

Note: The SFI install is performed successfully only if a valid license is given to the Flashing tool.

STM32CubeProgrammer supports secure firmware install for such devices as well as all STM32H7, STM32L4, STM32L5, STM32WL, STM32U5 and STM32MP devices available so far.

For more details on SFI over these STM32 devices refer to AN4992 *[1]*. This document is available on *www.st.com*.

The general flow of the secure firmware installation using a bootloader interface on a chip for H7 and L4 secure devices is shown respectively in *Figure 44* and *Figure 45* below.

Figure 44. Secure programming via STM32CubeProgrammer overview on STM32H7 devices

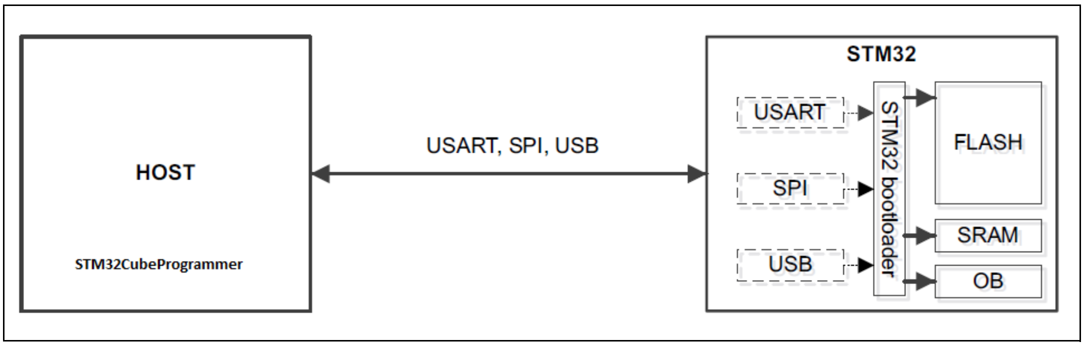
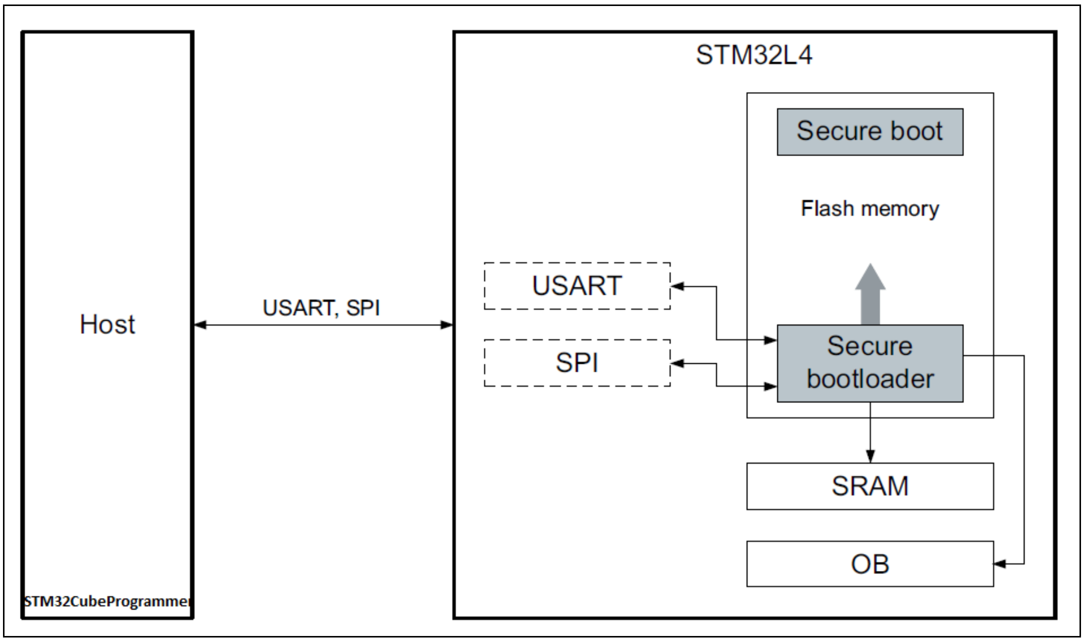


Figure 45. Secure programming via STM32CubeProgrammer overview on STM32L4 devices



4.2.2 Secure Module installation using a bootloader interface flow

As explained in *Section 3.4: SMI generation process*, outputs are generated for this particular use case:

- The first part, not encrypted: this is a regular ELF/AXF file, containing all the sections except the code section extracted by the STPC to prepare the SMI module.
- The encrypted SMI module, which contains the protected code.

The first part is programmed into the chip using any means (JTAG Flasher, UART bootloader and so on, as for any regular ELF/AXF file.

The full content of the SMI image file and its corresponding license are given to STM32CubeProgrammer which places them in RAM.

The SMI has to be invoked via the secure bootloader.

Note: The SMI install is performed successfully only if the adequate license is given to the Flashing tool.

4.2.3 STM32CubeProgrammer for SFI using a bootloader interface

For SFI programming, the STM32CubeProgrammer is used in CLI mode (the only mode so-far available) by launching the following command:

-sfi, --sfi

Syntax: -sfi protocol=<Ptype> <file_path> <licenseFile_path>

[<protocol=Ptype>]	: Protocol type to be used : static/live Only static protocol is supported so far Default value static
<file_path>	: Path of sfi file to be programmed
[hsm=0|1]	: Set user option for HSM use value in {0 (do not use HSM), 1 (use HSM)} Default value : hsm = 0
<lic_path|slot=slotID>	: Path to the SFI license file (if hsm = 0) or reader slot ID if HSM is used (hsm = 1)
[<licMod_path>|slot=slotID]	: List of the integrated SMI license files paths

If hsm = 1, the user must provide the Slot ID parameter.

If hsm = 0, the user must provide the license path file that can be generated separately using the following command line, provided an HSM card is available:

`-hsmgetlicense`

During th SFI process, the generated license can be used multiple times with the same MCU, without the need of an HSM card.

Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 br=115200 -sfi "C:\SFI\data.sfi"
hsm=1 "C:\SFI\license.bin"
```

This command allows secure installation of firmware *“data.sfi”* into a dedicated Flash memory address.

4.2.4 STM32CubeProgrammer for SMI via a bootloader interface

For SMI programming, STM32CubeProgrammer is used in CLI mode by launching the following command:

-smi, --smi

Syntax: -smi protocol=<Ptype> <file_path> [<address>] <licenseFile_path>

<protocol=Ptype>	: Protocol type to be used : static/live Only static protocol is supported so far Default value static
<file_path>	: Path of smi file to be programmed
[hsm=0|1]	: Set user option for HSM use value in {0 (do not use HSM), 1 (use HSM)} Default value: hsm=0
[<address>]	: Destination address of the smi module needed only for relocatable SMI
<lic_path|slot=slotID>	: Path to the SMI license file (if hsm=0) or reader slot ID if HSM is used (hsm=1)

Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 br=115200 -sfi "C:\SFI\data.sfi"
hsm=0 "C:\SFI\license.bin"
```

This command allows programming of the SMI specified file *“data.smi”* into a dedicated PCROPed area.

4.2.5 STM32CubeProgrammer for SSP via a bootloader interface

In this part the STM32CubeProgrammer tool is used in CLI mode (the only mode available so far for secure programming) to program the SSP image already created with STM32 Trusted Package Creator. STM32CubeProgrammer supports communication with ST HSMs (hardware secure modules based on Smart Card) to generate a license for the connected STM32 MPU device during SSP install.

The SSP flow can be performed using both USB or UART interfaces (not the STLINK interface).

STM32CubeProgrammer exports a simple SSP command with some options to perform the SSP programming flow.

-ssp, --ssp

Description: Program an SSP file

Syntax: -ssp <ssp_file_path> <ssp-fw-path> <hsm=0|1> <license_path|slot=slotID>

<ssp_file_path> : SSP file path to be programmed, bin or ssp extensions

<ssp-fw-path> : SSP signed firmware path

<hsm=0|1> : Set user option for HSM use (do not use HSM / use HSM)

Default value : hsm=0

<license_path|slot=slotID> :Path to the license file (if hsm=0)

Reader slot ID if HSM is used (if hsm=1)

Example using USB DFU bootloader interface:

```
STM32_Programmer_CLI.exe -c port=usb1 -ssp "out.ssp" "tf-a-ssp-stm32mp157f-dk2-trusted.stm32" hsm=1 slot=1
```

Note: All SSP traces are shown on the output console.

Figure 46. SSP install success

```
Requesting Chip Certificate...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Writing blob


Blob successfully written

Start operation achieved successfully
Send detach command
Detach command executed
SSP file out.ssp Install Operation Success
```

If there is any faulty input, the SSP process is aborted and an error message is displayed to indicate the root cause of the issue.

4.2.6 STM32CubeProgrammer get certificate via a bootloader interface

To get the chip certificate, STM32CubeProgrammer is used in CLI mode by launching the following command:

-gc, --getcertificate

Syntax: –gc <file_path>

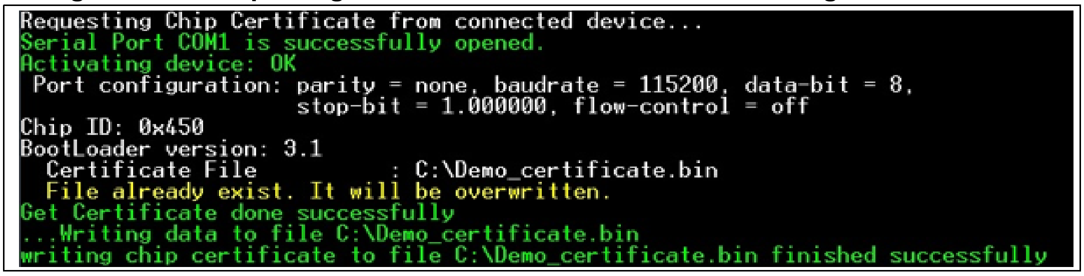
Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 -gc "C:\Demo_certificate.bin"
```

This command allows the chip Certificate to be read and uploaded into the specified file: *"C:\Demo_certificate.bin"*

The execution results are shown in *Figure 47*.

Figure 47. Example of getcertificate command execution using UART interface



4.3 Secure programming using JTAG/SWD interface

4.3.1 SFI/SFIx programming using JTAG/SWD flow

It is also possible to program the SFI/SFIx image using the JTAG interface. Here the read out protection mechanism (RDP level 1) cannot be used during SFI/SFIx as user Flash memory is not accessible after firmware chunks are written to RAM through the JTAG interface.

The whole process happens in RDP level 0. In the case of SFIx programming the code is protected by the OTFDEC encryption.

SFI via debug interface is currently supported for STM32H753xI, STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, and STM32L5 devices.

SFIx via debug interface is currently supported for STM32H7A3/7B3 and STM32H7B0, STM32H723/733, STM32L5 and STM32U5 devices.

For these devices, there is around 1 Mbyte of RAM available, with 512 Kbytes in main SRAM. This means that the maximum image size supported is 1 Mbyte, and the maximum area size is 512 Kbytes.

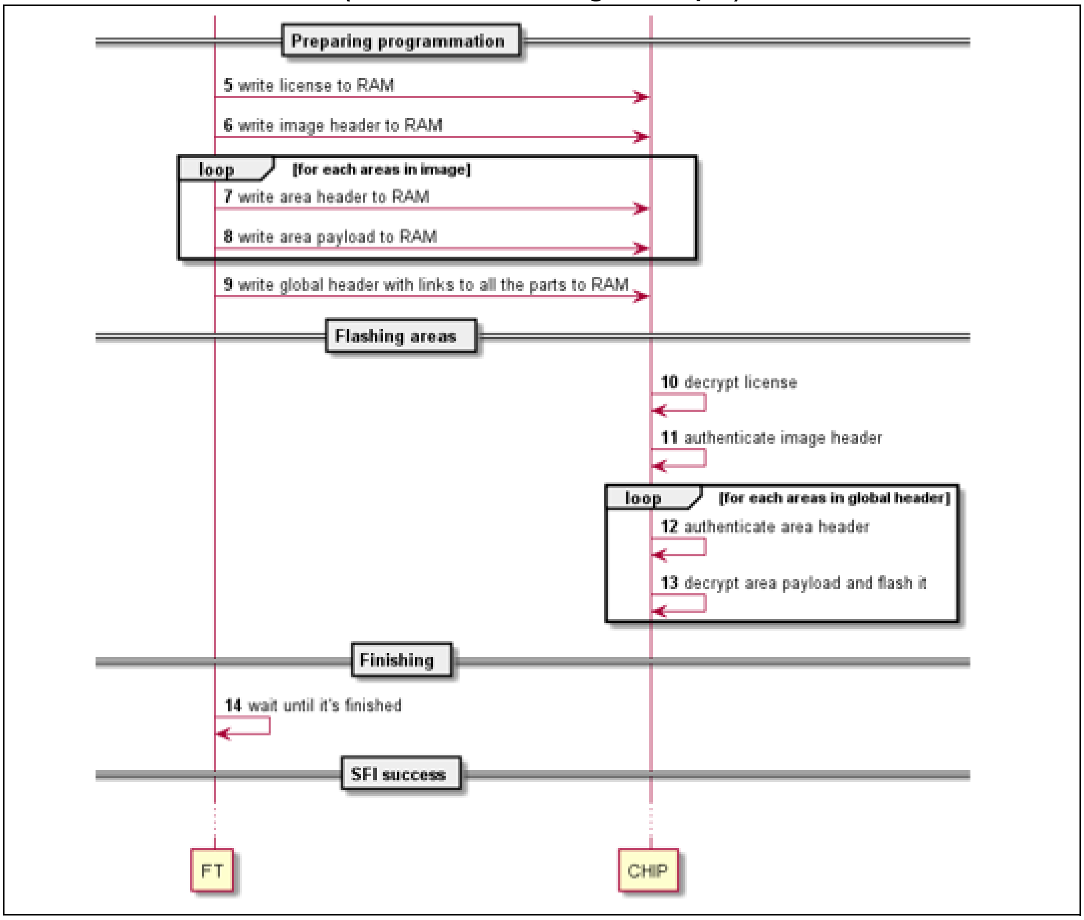
To remedy this, the SFI/SFIx image is split into several parts, so that each part fits into the allowed RAM size.

An SFI/SFIx multi install is then performed. Once all its SFI/SFIx parts are successfully installed, the global SFI/SFIx image install is successful.

Other limitations are that security must be left activated in the configuration area if there is a PCROP area. In the case of STM32L5 and STM32U5 devices, STM32CubeProgrammer sets the RDP Level on 0.5.

The SFI flow for programming through JTAG is described in *Figure 48*.

Figure 48. SFI programming by JTAG/SWD flow overview (monolithic SFI image example)



4.3.2 SMI programming through JTAG/SWD flow

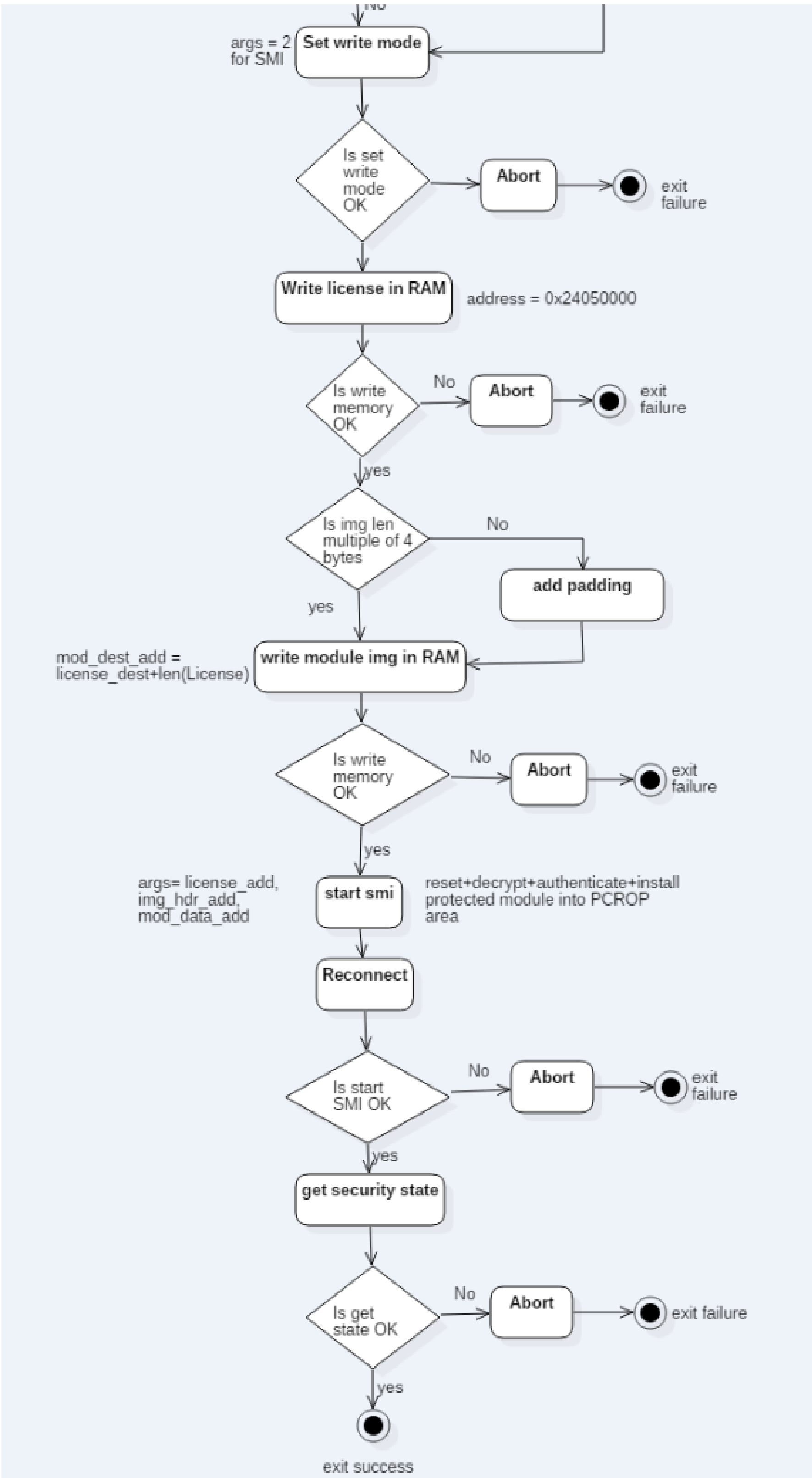
For SMI programming through JTAG/SWD the process flow is similar to that using the UART bootloader.

This means that the whole SMI image and its corresponding license must be transferred to RAM before starting. Then, there are two options to access SMI services through JTAG:

- write a small program in RAM that calls the public API (API details are available under non-disclosure agreement)
- use the secure API directly.

The essential steps of the SMI programming by JTAG flow are described in *Figure 49: SMI programming by JTAG flow overview*.

Figure 49. SMI programming by JTAG flow overview



4.3.3 STM32CubeProgrammer for secure programming using JTAG/SWD

The only modification in the STM32CubeProgrammer secure command syntax is the connection type which must be set to “jtag” or “swd”, otherwise all secure programming syntax for supported commands is identical.

Note: Using a debug connection “HOTPLUG” mode must be used with the connect command.

Example “getcertificate” command using JTAG

`STM32_Programmer.exe -c` **`port=jtag mode=HOTPLUG`** `-gc testJTAG_Certif.bin`

The result of this example is shown in *Figure 50*.

Figure 50. Example of getcertificate command using JTAG

```
ST-LINK Firmware version : V2J28S6
JTAG frequency = 9000 KHz
Connection mode: Hot Plug
Device ID: 0x450


  Certificate File          : testJTAG_Certif.bin

Requesting Chip Certificate using debug interface...
Get Certificate done successfully
...Writing data to file testJTAG_Certif.bin
Writing chip certificate to file testJTAG_Certif.bin finished successfully
Time elapsed during the getcertificate operation is: 00:00:00.032
```

Example “smi” command using SWD

```
-c port=swd mode=HOTPLUG -smi protocol=static
"RefSMI_MDK/FIR_module.smi" "RefSMI_MDK/licenseSMI.bin" -vb 3 -log
```

4.4 Secure programming using Bootloader interface (UART/I2C/SPI/USB)

It is also possible to program the SFI/SFIx image using the Bootloader interface (UART/I2C/SPI/USB). FDCAN is not supported by STM32CubeProgrammer since it not managed by ST-Link v3.

The whole process happens in RDP level 0.5. In the case of SFIx programming the code is protected by the OTFDEC encryption.

SFI via the Bootloader interface (UART/I2C/SPI/USB) is currently supported for STM32L5 devices. It needs to load an external loader using the **-elbl** command in the SRAM.

For STM32L5 devices, 1 Mbyte of SRAM is available, with 512 Kbytes in main SRAM. This means that the maximum image size supported is 1 Mbyte, and the maximum area size is 512 Kbytes.

To remedy this, the SFI/SFIx image is split into several parts, so that each part fits into the allowed SRAM size.

An SFI/SFIx multi install is then performed. Once all its SFI/SFIx parts are successfully installed, the global SFI/SFIx image install is successful.

SFI example

```
STM32_Programmer_CLI.exe -c port=usb1 -sfi out.sfix hsm=0
license.bin -rsse RSSe\L5\enc_signed_RSSe_sfi_bl.bin
```

SFIx example

```
STM32_Programmer_CLI.exe -c port=usb1 -elbl
MX25LM51245G_STM32L552E-EVAL-SFIX-BL.stldr -sfi out.sfix hsm=0
license.bin -rsse RSSe\L5\enc_signed_RSSe_sfi_bl.bin
```

5 Example of SFI programming scenario

5.1 Scenario overview

The actual user application to be installed on the STM32H753xI (or STM32L5) device makes "`printf`" packets appear in serial terminals. The application was encrypted using the STPC.

The OEM provides tools to the CM to get the appropriate license for the concerned SFI application.

5.2 Hardware and software environment

For successful SFI programming, some and SW prerequisites apply:

- STM32H743I-EVAL board
- STM32H753xI with bootloader and RSS programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running on either Windows 7 or Ubuntu 14 in both 32-bit and 64-bit versions
- STM32TrustPackageCreator v0.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v0.4.0 (or greater) package available from www.st.com.

5.3 Step-by-step execution

5.3.1 Build OEM application

OEM application developers can use any IDE to build their own firmware.

5.3.2 Perform the SFI generation (GUI mode)

To be encrypted with the STM32 Trusted Package Creator tool, OEM firmware is provided in AXF format in addition to a CSV file to set the option bytes configuration. A 128-bit AES encryption key and a 96-bit nonce are also provided to the tool. They are available in the *"SFI_ImagePreparation"* directory.

An *".sfi"* image is then generated (*out.sfi*).

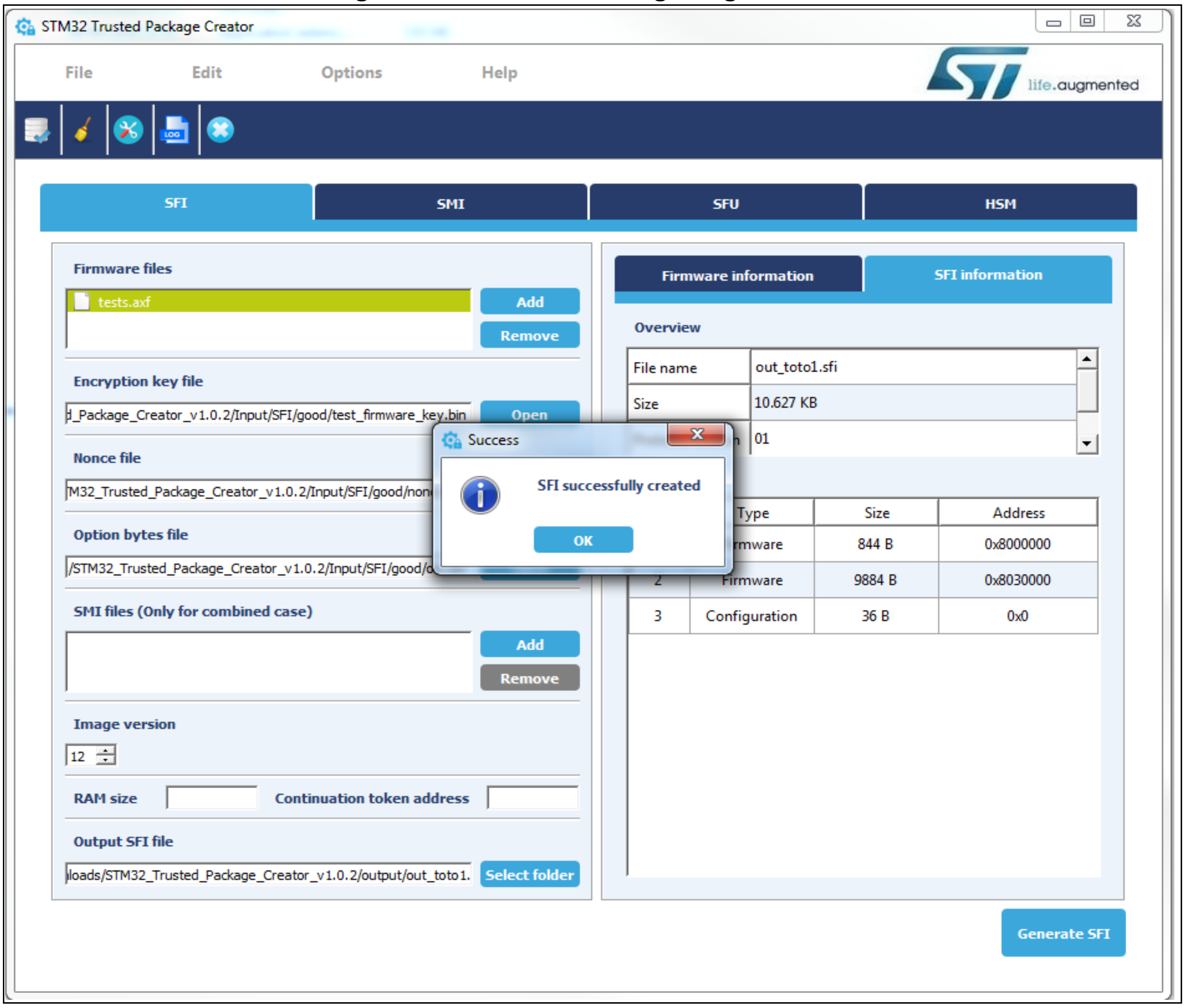
Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

Figure 51: STPC GUI during SFI generation shows the STPC GUI during the SFI generation.

Figure 51. STPC GUI during SFI generation



5.3.3 Performing HSM programming for license generation using STPC (GUI mode)

The OEM must provide a license generation tool to the programming house to be used for license generation during the SFI install process.

In this example, HSMs are used as license generation tools in the field. See *Section 4.1.2: License mechanism* for HSM use and programming.

Figure 52 shows an example for HSM programming by OEM to be used for SFI install.

The maximum number of licenses delivered by the HSM in this example is 1000.

This example uses HSM version 2, and is also valid for version 1 when the 'Version' field is set accordingly. The HSM version can be identified before performing the programming operation by clicking the Refresh button to make the version number appear in the 'Version' field.

The STM32 Trusted Package Creator tool provides all personalization package files ready to be used on SFI/SFIx and SSP flows. To get all the supported packages, go to the **PersoPackages** directory residing in the tool's install path.

Each file name starts with a number, which is the product ID of the device. You must select the correct one.

To obtain the appropriate personalization data, you first need to obtain the product ID:

- Use the STM32CubeProgrammer tool to launch a Get Certificate command to generate a certificate file containing some chip security information, bearing in mind that this command is only recognized only for devices that support the security feature:

  ```
  STM32_Programmer_CLI -c port=swd -gc "certificate.bin"
  ```

  A file named "certificate.bin" is created in the same path of the STM32CubeProgrammer executable file.
- Open the certificate file with a text editor tool, then read the 8 characters from the header which represents the product ID.

  For example:
  - When using STM32H7 device, you would find: 45002001.
  - When using STM32L4 device, you would find: 46201002.

Once you have the product ID, you can differentiate the personalization package to be used on the HSM provisioning step respecting the following naming convention:

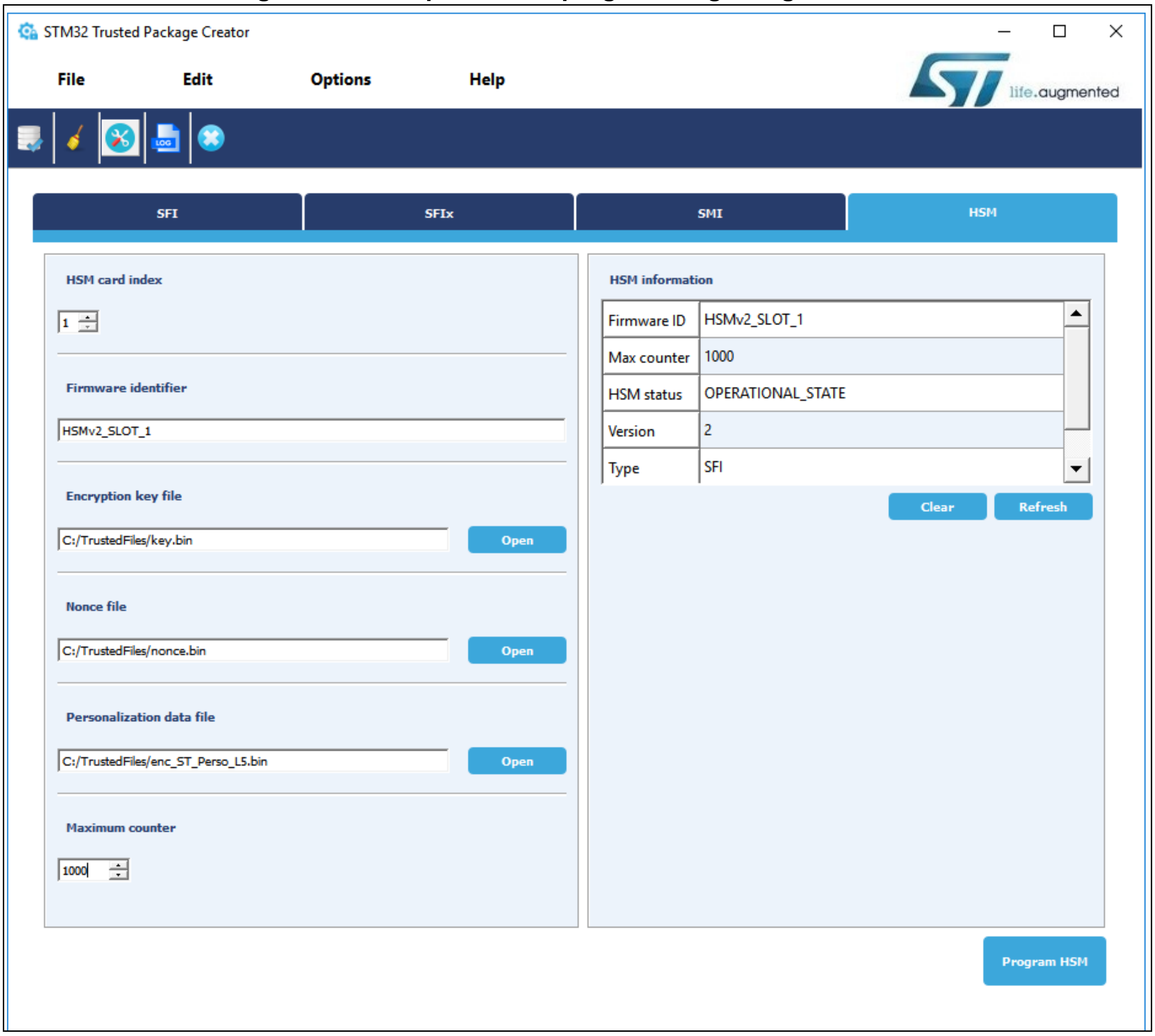
ProdcutID_FlowType_LicenseVersion_SecurityVersion.enc.bin

For example: 47201003_SFI._01000000_00000000.enc.bin

Based on this name we can retrieve the associated information:

- Product ID = 47201003 for STM32L5 devices (0x472 as device ID).
- Type = SFI
- License version = 01 (Big Endian)
- Security version = 0

Figure 52. Example of HSM programming using STPC GUI



Note: *When using HSM version1, the "Personalization data file" field is ignored when programming starts. It is only used with HSM version 2.*

When the card is successfully programmed, a popup window message "HSM successfully programmed" appears, and the HSM is locked. Otherwise an error message is displayed.

5.3.4 Performing HSM programming for license generation using STPC (CLI mode)

STM32 Trusted Package Creator provides CLI commands to program HSM cards. In order to configure the HSM before programming, the user must provide the mandatory inputs by using the specific options.

Example of HSM version 1 provisioning

```
STM32TrustedPackageCreator_CLI -hsm -i 1 -k "C:\TrustedFiles\key.bin" -n
"C:\TrustedFiles\nonce.bin" -id HSMv1_SLOT_1-mc2000
```

- -i: select the slot ID
- -k: set the encryption key file path
- -n: set the nonce file path
- -id: set the firmware identifier
- -mc: set the maximum number of licenses.

HSMv2 allows users to personalize their own HSM to achieve, for example, compatibility with the desired STM32 device. This solution covers the limitation of HSMv1 (static behavior), so it is possible to support new devices that are not available on HSMv1.

To perform this operation the user first needs to know the product ID of the device. This information is provided in the STM32 device certificate, which can be obtained with the following command:

```
STM32_Programmer.exe -c port=COM1 -gc "C:\SFI\Certificate.bin"
```

After getting the binary file of the device certificate, is necessary to open this file using a HEX editor application. Once these steps are done the user can read the product ID.

Figure 53. Example product ID

```
00000000  00 01 02 03  04 05 06 07  08 09 0a 0b  0c 0d 0e 0f
00000000  34 39 37 30  31 30 30 35  07 d7 60 65  98 2a fe 36   49701005.×`e˜*þ6
00000010  29 ca 59 f3  d5 29 9b 99  f7 a3 4e c0  bb 15 5f d1   )ÊYóÕ)›™÷£NÀ»._Ñ
00000020  1d 82 f4 8a  9a 13 2d d3  c9 2a 9a 02  c0 9b db 10   .‚ôŠš.-ÓÉ*š.À›Û.
00000030  fc 2d 28 d9  c9 77 bc 4c  ba 38 5b 15  e5 b0 8d bd   ü-(ÙÉw¼Lº8[.å° ½
00000040  d0 4d c3 4a  e9 d1 24 6b  a8 fc 3f 51  af 42 41 dd   ÐMÃJéÑ$k¨ü?Q¯BAÝ
00000050  be b3 e4 bb  77 48 14 fa  4b d6 3b bb  67 44 e5 a1   ¾³ä»wH.úKÖ;»gDå¡
00000060  63 ca 76 6b  db a3 80 cf  e0 61 f3 01  07 05 dd 6c   cÊvkÛ£€Ïàaó...Ýl
00000070  74 f6 29 23  17 8f bd e7  c5 cb 3a 5c  0e 5b 58 a3   tö)#. ½çÅË:\.[X£
00000080  8c dc 8d 13  97 1e ab 52  .. .. .. ..  .. .. .. ..   ŒÜ .—.«R........
00000090  .. .. .. ..  .. .. .. ..  .. .. .. ..  .. .. .. ..   ................
```

The product ID of the STM32WL used is: 49701005

In the second step the user provisions their own HSMv2 by programming it using STPC. The personalization data file .bin can be found under *"..\bin\PersoPackages"*.

Example of HSM version 2 provisioning

A new option [-pd] must be inserted to include the personalization data:

```
STM32TrustedPackageCreator_CLI -hsm -i 1 -k "C:\TrustedFiles\key.bin" -n
"C:\TrustedFiles\nonce.bin" -id HSMv2_SLOT_2 -mc 2000 -pd
"C:\TrustedFiles\enc_ST_Perso_L5.bin"
```

- -pd: Set the personalization data file path.

To obtain the appropriate personalization data file and for further information, refer to *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.

Note: *A green message display indicates that the programming operation succeeded, otherwise a red error message is displayed.*

If the HSM is already programmed and there is a new attempt to reprogram it, an error message being displayed to indicate that the operation failed and the HSM is locked.

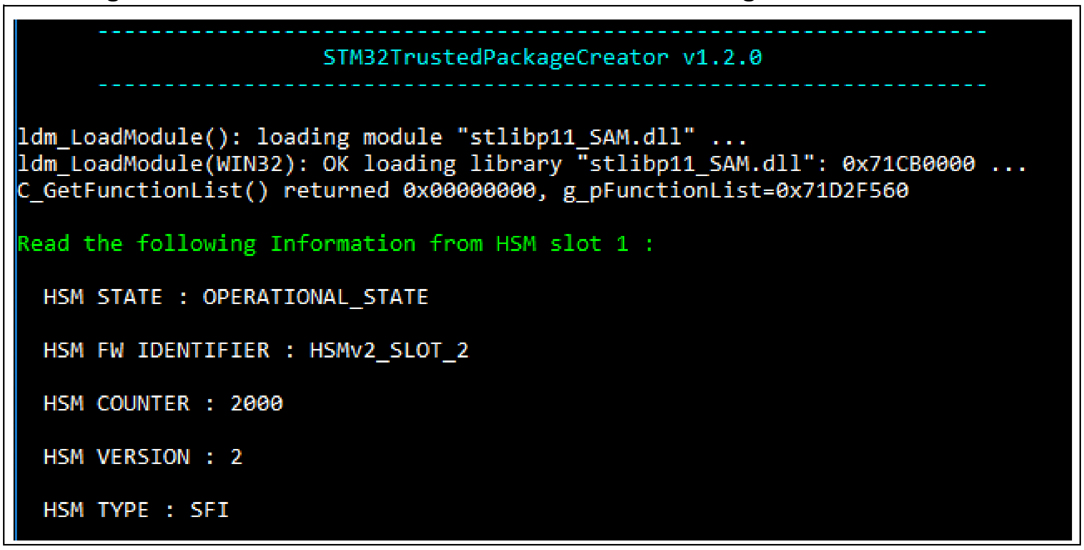
HSM v1 supports a list of limited number of STM32 devices such as STM32L4, STM32H7, STM32L5, and STM32WL.

Example of HSM get information

If the HSM is already programmed or is virgin yet and whatever the version, a get information command can be used to show state details of the current HSM by using the command below:

```
STM32TrustedPackageCreator_CLI -hsm -i 1 -info
```

Figure 54. HSM information in STM32 Trusted Package Creator CLI mode



5.3.5 Programming input conditions

Before performing an SFI install make sure that:

- Flash memory is erased.
- No PCROPed zone is active, otherwise destroy it.
- The chip must support security (a security bit must be present in the option bytes).
- When using a UART interface the User security bit in option bytes must be enabled before launching the SFI command. For this, the following STM32CubeProgrammer command is launched:
  - Launch the following command (UART bootloader used => Boot0 pin set to VDD):
    `-c port=COM9 -ob SECURITY=1`
- When using a UART interface the Boot0 pin must be set to VSS:
  - After enabling security (boot0 pin set to VDD), a power off/power on is needed when switching the Boot0 pin from VDD to VSS: power off, switch pin then power on.
- When performing an SFI install using UART bootloader then, no debug interface must be connected to any USB host. If a debug interface is still connected, disconnect it then perform a power off/power on before launching the SFI install to avoid any debug intrusion problem.
- Boot0 pin set to VDD When using a debug interface.
- A valid license generated for the currently-used chip must be at your disposal, or a license generation tool to generate the license during SFI install (HSM).
- For STM32L5 products, TZEN must be set at 0 (TZEN=0).

5.3.6 Perform the SFI install using STM32CubeProgrammer

In this section the STM32CubeProgrammer tool is used in CLI mode (the only mode so-far available for secure programming) to program the SFI image *"out.sfi"* already created in the previous section.

STM32CubeProgrammer supports communication with ST HSMs (Hardware Secure Modules based on smart card) to generate a license for the connected STM32 device during SFI install.

Using JTAG/SWD

After making sure that all the input conditions are respected, open a cmd terminal and go to *<STM32CubeProgrammer_package_path>/bin*, then launch the following STM32CubeProgrammer command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local_path>/out.sfi" hsm=1 slot=<slot_id>
```

Note: In the case of an STM32L5 device the SFI install uses the RSSe and its binary file is located in the STM32CubeProgrammer bin/RSSe folder.

The STM32CubeProgrammer command is as follows:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local_path>/out.sfi" hsm=1 slot=<slot_id> -rsse <RSSe_path>
```

Figure 55 shows the SFI install via SWD execution and the HSM as license generation tool in the field.

Figure 55. SFI install success using SWD connection (1)

```
      -------------------------------------------------------------------
                        STM32CubeProgrammer v1.0.7
      -------------------------------------------------------------------

ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: V2J30M19
Target voltage: 3.21V
SWD frequency: 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450

Device name: STM32H7xx
Device type: MCU
Device CPU : Cortex-M7/M4
  Protocol Information          : static

  SFI File Information          :

     SFI file path              : out_EH.sfi
     SFI ID                     : 111
     SFI header information     :
         SFI protocol version         : 1
         SFI total number of areas    : 3
         SFI image version            : 23
     SFI Areas information      :

     Parsing Area 1/3     :
         Area type                    : F
         Area size                    : 844
         Area destination address     : 0x8000000

     Parsing Area 2/3     :
         Area type                    : F
         Area size                    : 10528
         Area destination address     : 0x8030000

     Parsing Area 3/3     :
         Area type                    : C
         Area size                    : 36
         Area destination address     : 0x0

Reading the chip Certificate...


Requesting Chip Certificate using debug interface...
Get Certificate done successfully

Requesting Licesne for firmware with ID : 111

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x5FC00000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x5FCC8A78
Init Communication with slot 2 Success!

Succeed to generate license for the current STM32 device

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware with ID 111

Starting Firmware Install operation...


Activating security...
Warning: Option Byte: SECURITY, value: 0x1, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Activating security Success
Setting write mode to SFI
Warning: Option Byte: SECURITY, value: 0x1, was not modified.
Warning: Option Byte: ST_RAM_SIZE, value: 0x3, was not modified.
Succeed to set write mode for SFI
Starting SFI part 1

Writing license to address 0x24030800
Writing Img header to address 0x24031000
Writing areas and areas wrapper...
all areas processed
RSS process started...

RSS command execution OK
```

Figure 56. SFI install success using SWD connection (2)

```
RSS command execution OK
Reconnecting...
ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: V2J30M19
Target voltage: 3.21V
Error: ST-LINK error (DEV_NO_DEVICE)
...retrying...
ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: V2J30M19
Target voltage: 3.21V
SWD frequency: 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450

Reconnected !

Requesting security state...
SECURITY State Success
SFI SUCCESS!
SFI file out_EH.sfi Install Operation Success
```

6 Example of SFI programming scenario for STM32WL

6.1 Scenario overview

The user application is developed by the OEM and encrypted by STPC. The OEM provides the following elements to the programming house:

- the encrypted firmware of STM32WL
- HSMv1 or previsioned HSMv2
- STM32CubeProgrammer.

With these inputs the untrusted manufacturer is able to securely program the encrypted firmware.

6.2 Hardware and software environment

For successful SFI programming, the following hardware and software prerequisites apply:

- STM32WL5x board with Bootloader and RSS programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running on either Windows or Ubuntu 14 (in both 32-bit and 64-bit versions) or macOS
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v2.6.0 (or greater) package available from www.st.com
- HSMv1 or HSMv2.

6.3 Step-by-step execution

6.3.1 Build OEM application

OEM application developers can use any IDE to build their own firmware.

6.3.2 Perform the SFI generation (GUI mode)

The first step to install the secure firmware on STM32 devices is the encryption of the user OEM firmware (already provided in AXF format) using the STM32 Trusted Package Creator tool.

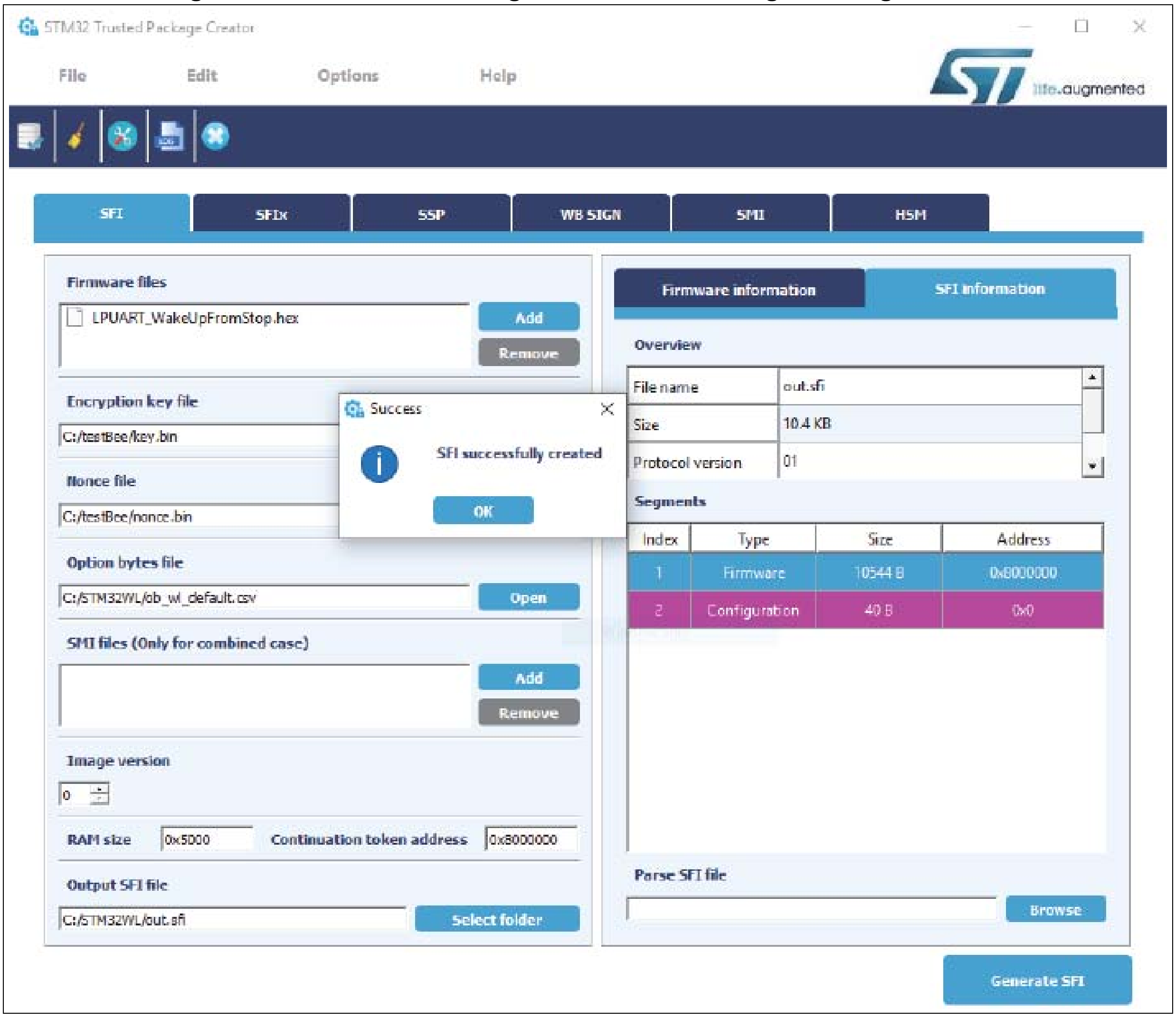
This is done by adding the following files in the STPC tool:

- OEM firmware
- a .csv file containing option bytes configuration
- a 128-bit AES encryption key
- a 96-bit nonce

Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES
The option bytes are described in the product reference manual.
In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

A programmed HSM card should be insert in the PC, and an *"out.sfi"* image is then generated.

Figure 57. STPC GUI showing the STPC GUI during the SFI generation



Note: To perform HSM programming for license generation using STPC (GUI mode and CLI mode) refer to the following sections:

Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)

Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)

6.3.3 Programming input conditions

Before performing an SFI install on STM32WL devices make sure that:

- Flash memory is erased
- No PCROPed zone is active, otherwise remove it
- The chip supports security (a security bit must be present in the option bytes)
- The security should be disabled, if activated
- The option bytes of the device are set to default values. This step is done by the two commands given below.

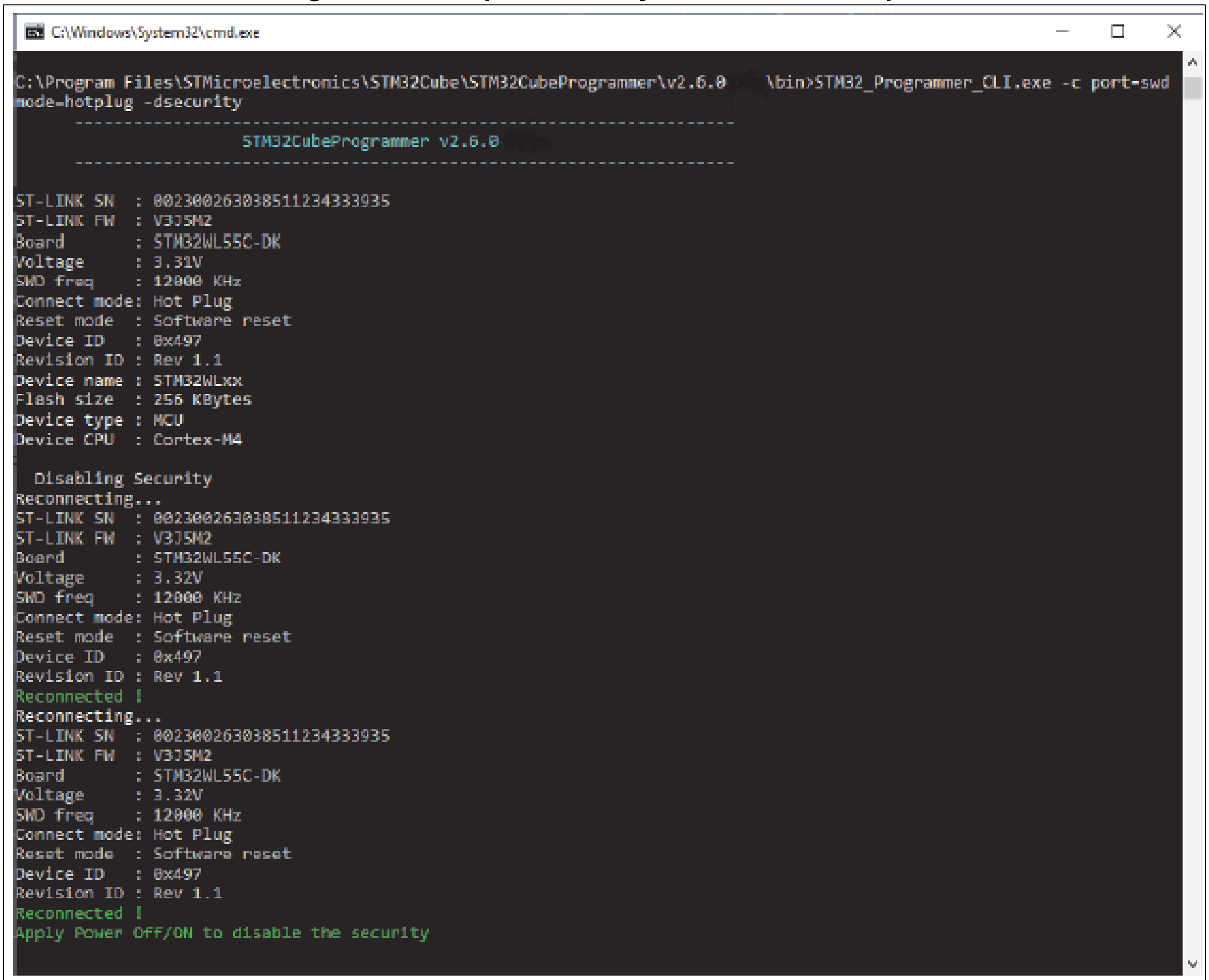
-desurity: this option allows the user to disable security. After executing this command, a power OFF / power ON should be done.

Example:

```
STM32_Programmer_CLI.exe -c port=swd mode=hotplug -dsecurity
```

Figure 58 hows the resulting output on the command line.

Figure 58. Example -dsecurity command-line output



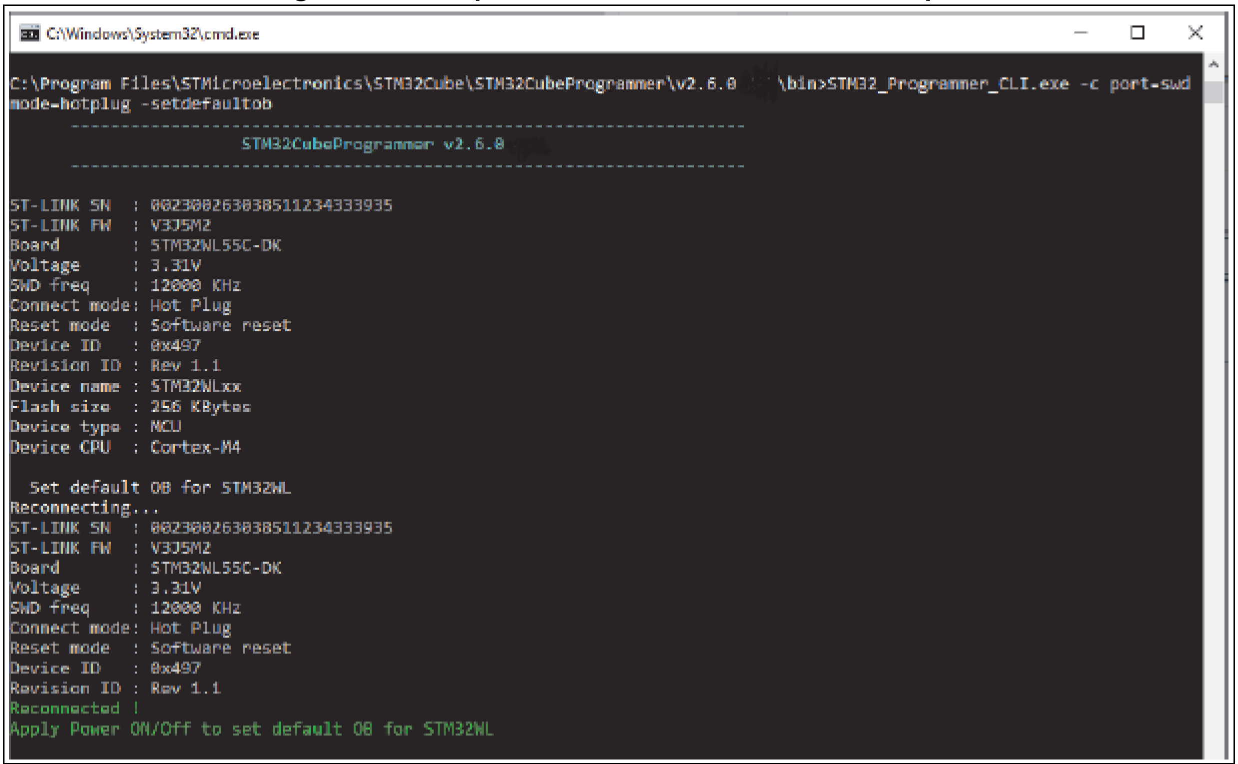
-setdefaultob: this command allows user to configure option bytes to their default values. After executing this command, a power OFF/power ON should be done.

Example:

```
STM32_Programmer_CLI.exe -c port=swd mode=hotplug -setdefaultob
```

Figure 59 shows the resulting output on the command line.

Figure 59. Example -setdefaultob command-line output



6.3.4 Perform the SFI install using STM32CubeProgrammer

In this section the STM32CubeProgrammer tool is used in CLI mode (the only mode so-far available for secure programming) to program the SFI image "out.sfi" already created in the previous section.

STM32CubeProgrammer supports communication with ST HSMs (Hardware Secure Modules based on smart card) to generate a license for the connected STM32 device during SFI install.

Using JTAG/SWD

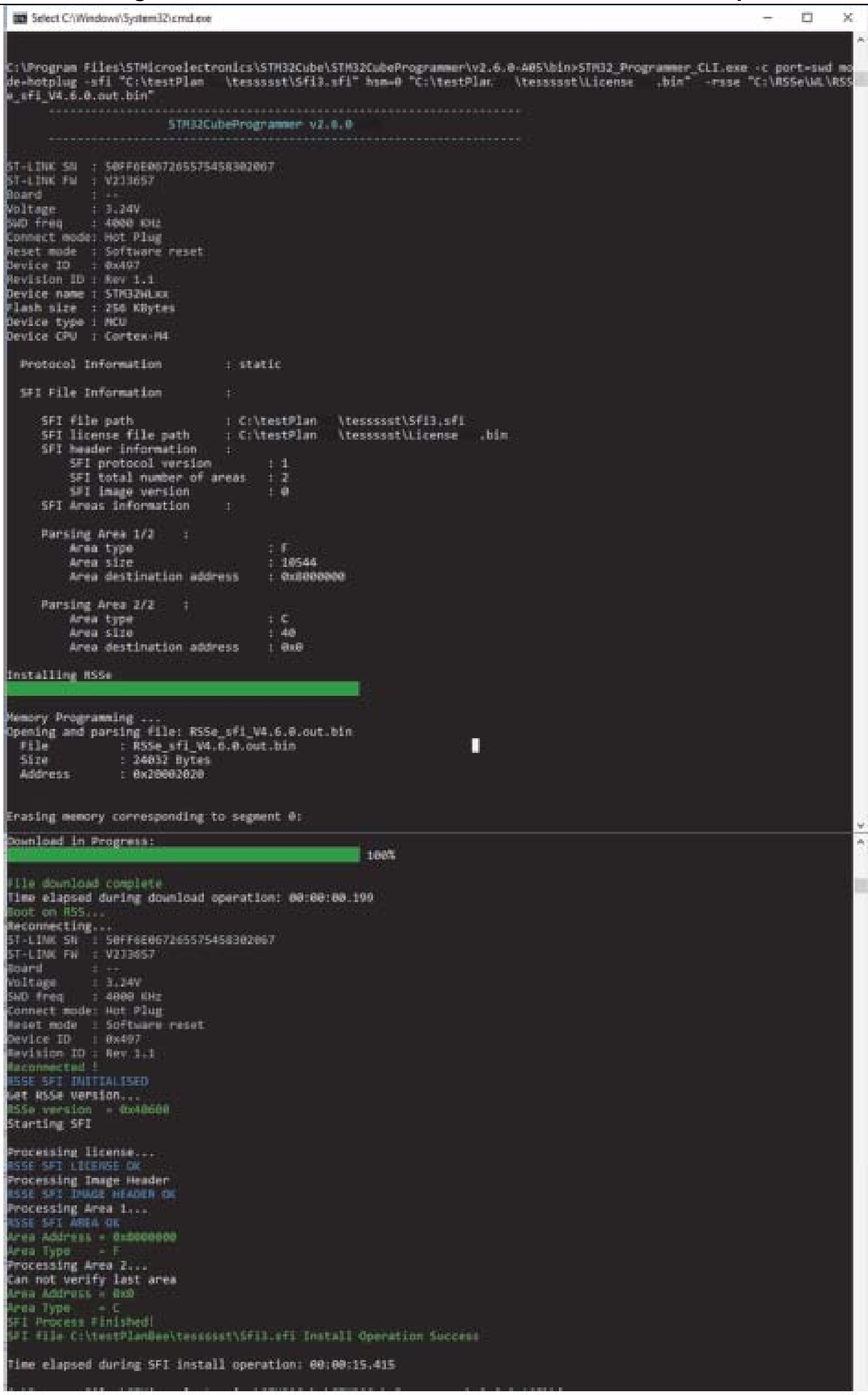
After making sure that all the input conditions are respected, open a cmd terminal and go to *<STM32CubeProgrammer_package_path>/bin*, then launch the following STM32CubeProgrammer command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi
"<local_path>/out.sfi" hsm=1 slot=<slot_id> -rsse "< RSSe_path >"
```

Note: The RSSe and its binary file is located in the STM32CubeProgrammer bin/RSSe/WL folder.

Figure 60 shows the SFI install via SWD execution.

Figure 60. SFI install via SWD execution command-line output



```
Select C:\Windows\System32\cmd.exe

C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0-A05\bin>STM32_Programmer_CLI.exe -c port=swd mo
de=hotplug -sfi "C:\testPlan   \tessssst\Sfi3.sfi" hsm=0 "C:\testPlan   \tessssst\License    .bin" -rsse "C:\RSSe\WL\RSS
e_sfi_V4.6.0.out.bin"
      -------------------------------------------------------------------
                      STM32CubeProgrammer v2.6.0
      -------------------------------------------------------------------

ST-LINK SN  : 50FF6E067265575458302067
ST-LINK FW  : V2J36S7
Board       : --
Voltage     : 3.24V
SWD freq    : 4000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x497
Revision ID : Rev 1.1
Device name : STM32WLxx
Flash size  : 256 KBytes
Device type : MCU
Device CPU  : Cortex-M4

  Protocol Information         : static

  SFI File Information         :

     SFI file path             : C:\testPlan   \tessssst\Sfi3.sfi
     SFI license file path     : C:\testPlan   \tessssst\License    .bin
     SFI header information    :
         SFI protocol version        : 1
         SFI total number of areas   : 2
         SFI image version           : 0
     SFI Areas information     :

     Parsing Area 1/2    :
         Area type                   : F
         Area size                   : 10544
         Area destination address    : 0x8000000

     Parsing Area 2/2    :
         Area type                   : C
         Area size                   : 40
         Area destination address    : 0x0

Installing RSSe


Memory Programming ...
Opening and parsing file: RSSe_sfi_V4.6.0.out.bin
  File          : RSSe_sfi_V4.6.0.out.bin
  Size          : 24032 Bytes
  Address       : 0x20002020


Erasing memory corresponding to segment 0:
Download in Progress:
                                                   100%

File download complete
Time elapsed during download operation: 00:00:00.199
Boot on RSS...
Reconnecting...
ST-LINK SN  : 50FF6E067265575458302067
ST-LINK FW  : V2J36S7
Board       : --
Voltage     : 3.24V
SWD freq    : 4000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x497
Revision ID : Rev 1.1
Reconnected !
RSSE SFI INITIALISED
Get RSSe version...
RSSe version  = 0x40600
Starting SFI

Processing license...
RSSE SFI LICENSE OK
Processing Image Header
RSSE SFI IMAGE HEADER OK
Processing Area 1...
RSSE SFI AREA OK
Area Address = 0x8000000
Area Type    = F
Processing Area 2...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file C:\testPlanBee\tessssst\SFI3.sfi Install Operation Success

Time elapsed during SFI install operation: 00:00:15.415
```

7 Example of SFI programming scenario for STM32U5

7.1 Scenario overview

The actual user application to be installed on the STM32U5 device makes "`printf`" packets appear in serial terminals. The application was encrypted using the STPC.

The OEM provides tools to the CM to get the appropriate license for the concerned SFI application.

7.2 Hardware and software environment

For successful SFI programming, some HW and SW prerequisites apply:

- STM32U5 board with bootloader and RSS programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running either on Windows, Ubuntu 14 (64-bit version) or macOS.
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v2.8.0 (or greater) package available from *www.st.com*
- HSMv2.

7.3 Step-by-step execution

7.3.1 Build OEM application

OEM application developers can use any IDE to build their own firmware.

7.3.2 Perform the SFI generation (GUI mode)

The first step to install the secure firmware on STM32 devices is the encryption of the user OEM firmware (already provided in AXF format) using the STM32 Trusted Package Creator tool. This step is done by adding the following files in the STPC tool:

- an OEM firmware
- a.csv file containing option byte configuration
- a128-bit AES encryption key
- a96-bit nonce

Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

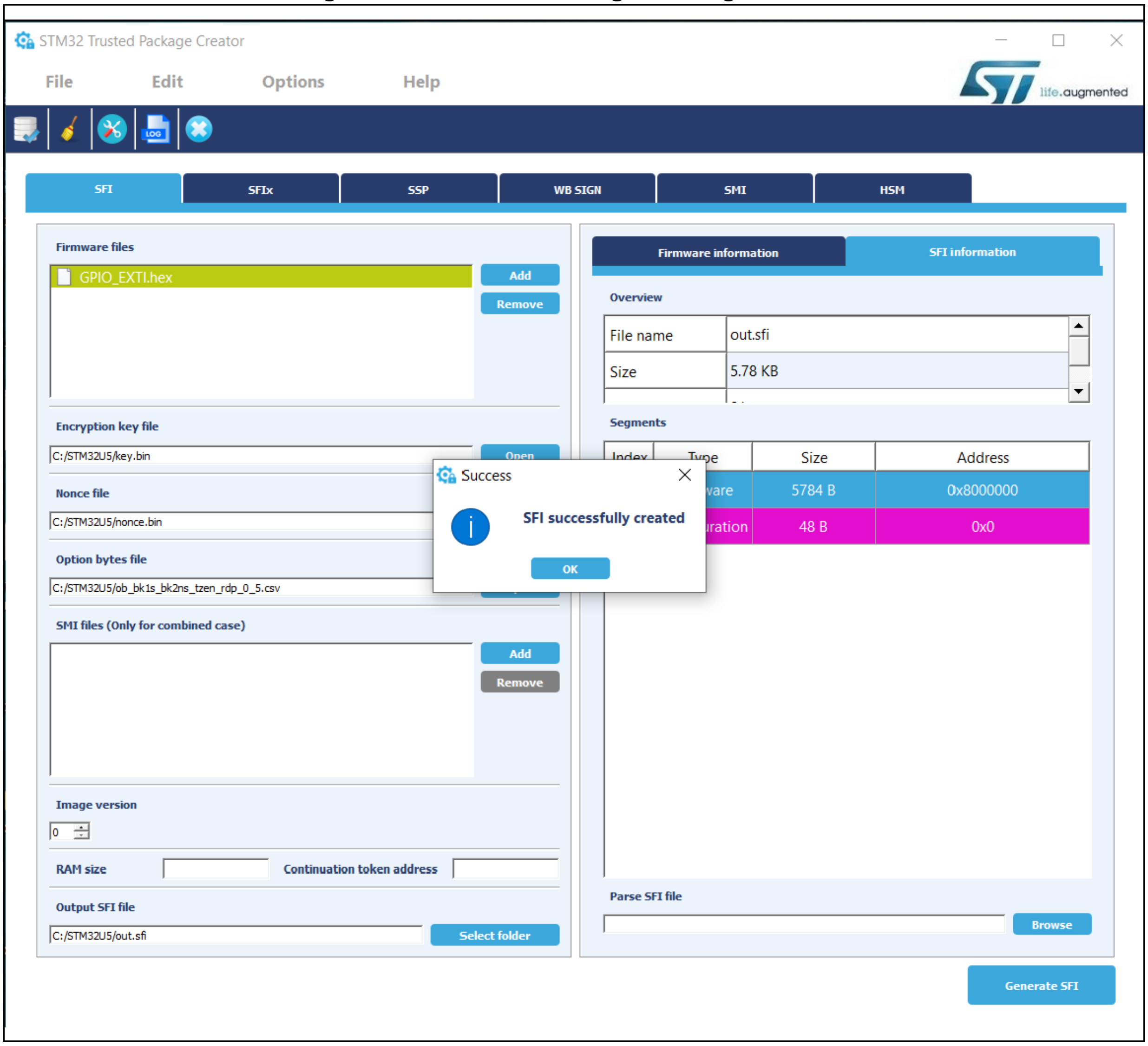
The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

In addition, a programmed HSM card should be insert in the PC. An "*out.sfi*" image is then generated.

Figure 61 shows STPC GUI during SFI generation.

Figure 61. STPC GUI during the SFI generation



Note: *To perform HSM programming for license generation using STPC (GUI and CLI modes), refer to Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode) and Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode).*

7.3.3 Programming input conditions

Before performing an SFI install on STM32U5 devices, make sure that:

- The Flash memory is erased.
- No WRP zone is active, otherwise destroy it.
- The chip supports security (a security bit must be present in the option bytes).
- If the security is activated, disable it.

7.3.4 Perform the SFI install using STM32CubeProgrammer

In this section the STM32CubeProgrammer tool is used in CLI mode (the only mode so far available for secure programming) to program the SFI image "*out.sfi*" already created in previous section.

STM32CubeProgrammer supports communication with ST HSMs (hardware secure modules based on smartcards) to generate a license for the connected STM32 device during the SFI install process.

Using JTAG/SWD

First make sure that all the input conditions are respected, then open a cmd terminal, go to *<STM32CubeProgrammer_package_path>/bin* and launch the following STM32CubeProgrammer command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi
"<local_path>/out.sfi" hsm=1 slot=<slot_id> -rsse "< RSSe_path >"
```

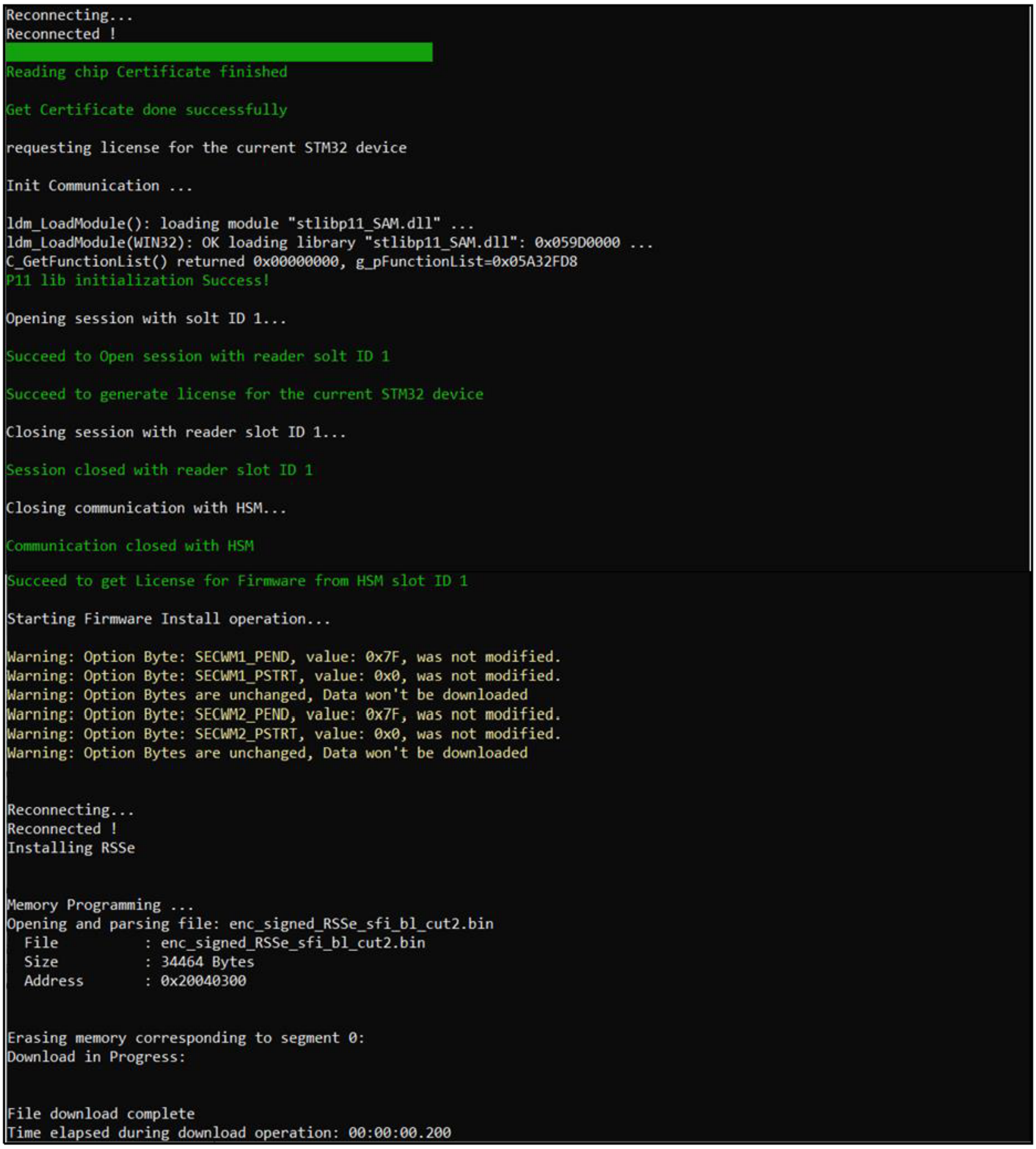
Note: The RSSe and the corresponding binary file are located in the STM32CubeProgrammer bin/RSSe/U5 *folder.*

Figure 62 and *Figure 63* show the STM32CubeProgrammer command used for the SFI install process via SWD execution.

Figure 62. SFI install via SWD execution (1)

```
Reconnecting...
Reconnected !

Reading chip Certificate finished

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x059D0000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x05A32FD8
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded


Reconnecting...
Reconnected !
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi_bl_cut2.bin
  File          : enc_signed_RSSe_sfi_bl_cut2.bin
  Size          : 34464 Bytes
  Address       : 0x20040300


Erasing memory corresponding to segment 0:
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:00.200
```

Figure 63. SFI install via SWD execution - (2)



```
Reconnecting...
Reconnected !

Reading chip Certificate finished

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x059D0000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x05A32FD8
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded


Reconnecting...
Reconnected !
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi_bl_cut2.bin
  File          : enc_signed_RSSe_sfi_bl_cut2.bin
  Size          : 34464 Bytes
  Address       : 0x20040300


Erasing memory corresponding to segment 0:
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:00.200
```

8 Example of SMI programming scenario

8.1 Scenario overview

In this scenario, the 3rd party's library to be installed on the STM32H753xI device makes "`printf`" packets appear in the serial terminal if the library code execution called by the application does not crash.

The library code was encrypted using the STPC.

The OEM provides tools to the CM to get the appropriate license for the concerned SMI module.

8.2 Hardware and software environment

The same environment as explained in *Section 4.1.1: Device authentication*.

8.3 Step-by-step execution

8.3.1 Build 3rd party Library

ST or 3rd party developers can use any IDE to build the library to be encrypted and installed into the STM32H7 device.

In this scenario the SMI module based on the built library is not relocatable. The destination address is hardcoded in SMI module to the following value: 0x08080000.

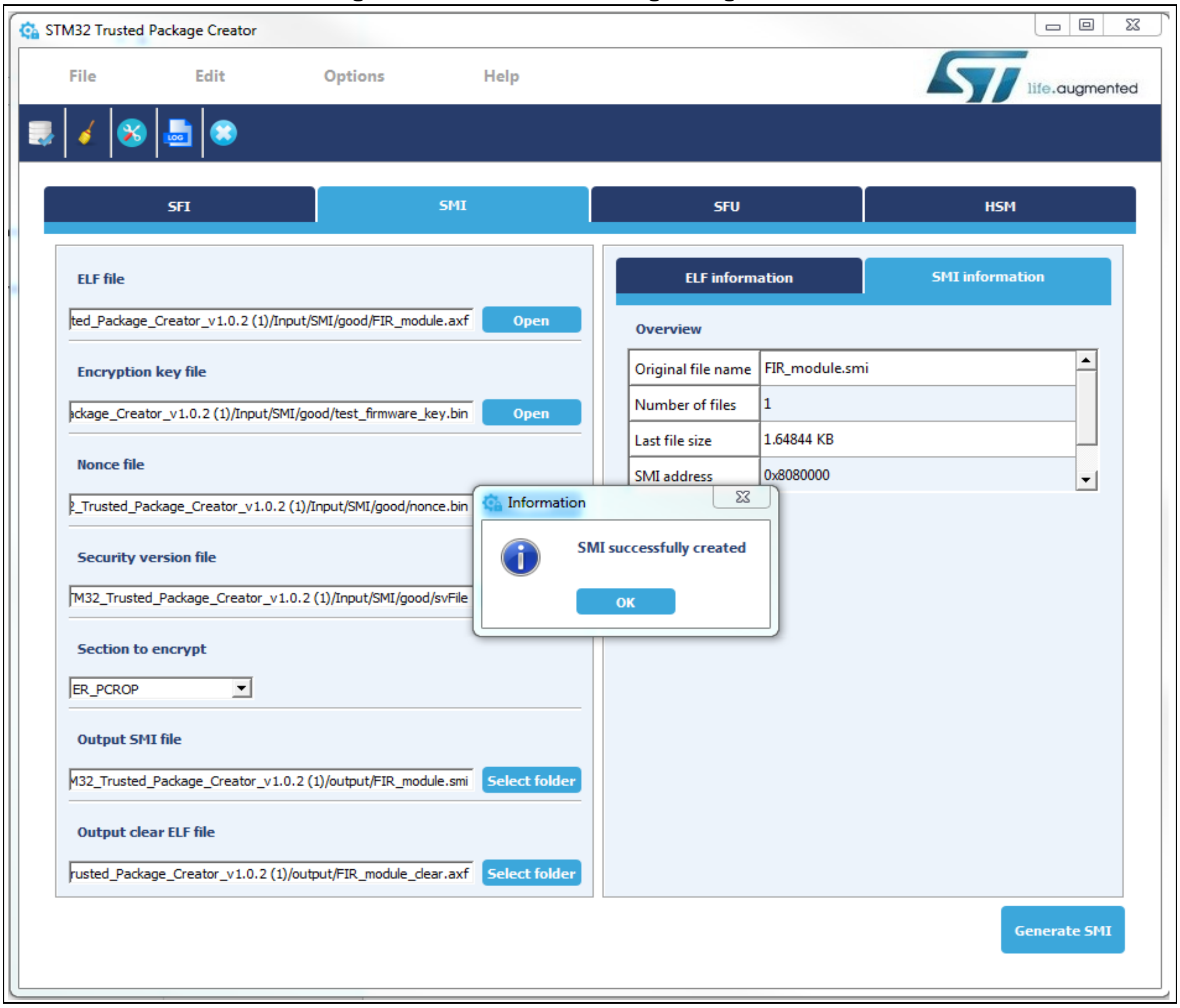
8.3.2 Perform the SMI generation

For encryption with the STM32 Trusted Package Creator tool, the 3rd party module is provided in ELF format. A 128-bit AES encryption key, a 96-bit nonce and a security version file are also provided to the tool. They are available in the "*SMI_ImagePreparation"* directory. After choosing the name of the section to be encrypted, a *".smi"* image is then generated (*FIR_module.smi)*.

The clear data part of the library without the encrypted section is also created in ELF format (*FIR_module_clear.axf*).

Figure 64 shows the STPC GUI during SMI generation.

Figure 64. STPC GUI during SMI generation



8.3.3 Programming input conditions

Before performing the SMI install make sure that:

- The SMI module destination address is not already PCROPed, otherwise destroy this PCROPed area.
- The Boot0 pin set to VDD.
- The chip supports security (existing security bit in option bytes).
- When performing SMI install using UART bootloader, no debug interface is connected to any USB host. If a debug interface is still connected, disconnect it then perform a power off/power on before launching the SMI install to avoid any debug intrusion problem.
- The proper license generated for the currently-used chip must be at your disposal (or an HSM or secure server to generate it during SMI programming).

8.3.4 Perform the SMI install

Using JTAG/SWD

After making sure that all the input conditions are respected, open a cmd terminal and go to <STM32CubeProgrammer_package_path>/bin, then launch the following STM32CubeProgrammer command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -smi
protocol=static "<local_path>/FIR_module.smi"
"<local_path>/<licenseSMI.bin>"
```

This command allows the SMI specified file "*FIR_module.smi*" to be programmed into a dedicated PCROPed area at address (0x08080000).

Figure 65: SMI install success via debug interface shows the SMI install via SWD execution.

Figure 65. SMI install success via debug interface

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\hannachi>cd C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\d
ocs_forSTM32H7Release\AN\Sw_packages\stm32_programmer_package_v0.4.0

C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7Release
\AN\Sw_packages\stm32_programmer_package_v0.4.0>cd bin

C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7Release
\AN\Sw_packages\stm32_programmer_package_v0.4.0\bin>STM32_Programmer_CLI.exe -c
port=swd mode=HOTPLUG -smi protocol=static "C:\Users\hannachi\Documents\Projects
\STM32H7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\FIR_module.
smi" "C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7R
elease\AN\SMI_ImagePreparation\licenseSMI.bin"
      -------------------------------------------------------------------
                        STM32CubeProgrammer v0.4.0
      -------------------------------------------------------------------

ST-LINK Firmware version : V2J27M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450



  Protocol          : static
  SMI File          : C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\
docs_forSTM32H7Release\AN\SMI_ImagePreparation\FIR_module.smi

Starting SMI install operation for file :  C:\Users\hannachi\Documents\Projects\
STM32H7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\FIR_module.s
mi  ...
  SMI File Information          :

     SMI file path                : C:\Users\hannachi\Documents\Projects\STM32H
7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\FIR_module.smi
     SMI license file path        : C:\Users\hannachi\Documents\Projects\STM32H
7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\licenseSMI.bin
     SMI code destination address  : 0x8080000
     SMI code size                : 1688

Setting write mode to SMI
Succeed to set write mode for SMI
Writing license @ address 0x24050000...


License file successfully written at adress 0x24050000
Writing SMI module image to address 0x24050088...


SMI image successfully written at address 0x24050088
Starting SMI process with license @ 0x24050000 and image @ 0x24050088...

RSS process started...
RSS command execution OK
Reconnecting...
ST-LINK Firmware version : V2J27M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450

Reconnected !

Requesting security state...
SECURITY State Success
SMI SUCCESS!
SMI file C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32
H7Release\AN\SMI_ImagePreparation\FIR_module.smi Install Operation Success

Time elapsed during the SMI install operation is: 00:00:03.294

C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7Release
\AN\Sw_packages\stm32_programmer_package_v0.4.0\bin>
```

8.3.5 How to test for SMI install success

1. Flash the clear data part *"FIR_module_clear.hex"* (available under the "*Tests"* directory) into address 0x08084000 using STM32Cubeprogrammer or any other Flashing tool.
2. Flash the test application *"tests.hex"* (which is based on the SMI module), available under the "*Tests"* directory at start user Flash address "0x08000000" using STM32Cubeprogrammer or any other Flashing tool.

   The option bytes configuration becomes as below (*Figure 66*).

Figure 66. OB display command showing that a PCROP zone was activated after SMI

```
OPTION BYTES BANK: 0
   Read Out Protection:
     RDP          : 0xAA (Level 0, no protection)
   RSS:
     RSS1         : 0x0 (No SFI process on going)
   BOR Level:
     BOR_LEV      : 0x0 (reset level is set to 2.1 V)
   User Configuartion:
     IWDG1        : 0x1 (Independent watchdog is controlled by hardware)
     NRST_STOP_D1 : 0x1 (STOP mode on Domain 1 is entering without reset)
     NRST_STBY_D1 : 0x1 (STANDBY mode on Domain 1 is entering without reset)
     FZ_IWDG_STOP : 0x1 (Independent watchdog is running in STOP mode)
     FZ_IWDG_SDBY : 0x1 (Independent watchdog is running in STANDBY mode)
     SECURITY     : 0x1 (Security feature enabled)
     BCM7         : 0x1 (CM-7 boot enabled)
     NRST_STOP_D2 : 0x1 (STOP mode on Domain 2 is entering without reset)
     NRST_STBY_D2 : 0x1 (STANDBY mode on Domain 2 is entering without reset)
     SWAP_BANK    : 0x0 (after boot loading, no swap for user sectors)
     DMEPA        : 0x1 (delete PcROP protection and earse protected area)
     DMESA        : 0x1 (delete Secure protection and erase protected area)
   Boot address Option Bytes:
     BOOT_CM7_ADD0: 0x800  (0x8000000)
     BOOT_CM7_ADD1: 0x1FF0  (0x1FF00000)
   PCROP Protection:
     PCROPA_str   : 0x800  (0x8010000)
     PCROPA_end   : 0x806  (0x8080600)
   Secure Protection:
     SECA_str     : 0xFF  (0x8001FE0)
     SECA_end     : 0x0  (0x80000FF)
   DTCM RAM Protection:
     ST_RAM_SIZE  : 0x2 (8 KB)
   Write Protection:
     nWRP0        : 0x1 (Write protection not active on this sector)
     nWRP1        : 0x1 (Write protection not active on this sector)
     nWRP2        : 0x1 (Write protection not active on this sector)
     nWRP3        : 0x1 (Write protection not active on this sector)
     nWRP4        : 0x1 (Write protection not active on this sector)
     nWRP5        : 0x1 (Write protection not active on this sector)
     nWRP6        : 0x1 (Write protection not active on this sector)
     nWRP7        : 0x1 (Write protection not active on this sector)
```

3. If a UART connection is available on the board used, open the “*Hercule.exe”* serial terminal available under the *“Tests*” directory, open the connection. On reset the dedicated `“printf”` packets appears.

9 Example of SFIx programming scenario for STM32H7

9.1 Scenario overview

There are three steps during this scenario:
- Generate SFIx image using the STPC.
- Provisioning HSM card via STPC.
- Use STM32CubeProgrammer to perform the SFIx process.

Once this scenario is successfully installed on the STM32H7B3I-EVAL, follow the steps below:
- Write internal firmware data in the internal Flash memory starting at the address 0x08000000.
- Write external firmware data in the external Flash memory starting at the address 0x90000000.
- Verify that the option bytes were correctly programmed (depends on area C).

9.2 Hardware and software environment

For successful SFIx programming, some HW and SW prerequisites apply:
- STM32H7B3I-EVAL board containing external Flash memory.
- Micro-USB for debug connection.
- PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra.
- STM32TrustPackageCreator v1.2.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer V2.3.0 (or greater) package available from *www.st.com*
- HSMv1.1 card.

9.3 Step-by-step execution

9.3.1 Build OEM application

OEM application developers can use any IDE to build their own firmware.

Note: In this use case there are different user codes. Each one is specific to a Flash memory type (internal/external).

9.3.2 Perform the SFIx generation (GUI mode)

To be encrypted with the STM32 Trusted Package Creator tool, OEM firmware is provided in Bin/Hex/AXF format in addition to a CSV file to set the option bytes configuration. A 128-bit AES encryption key and a 96-bit nonce are also provided to the tool.

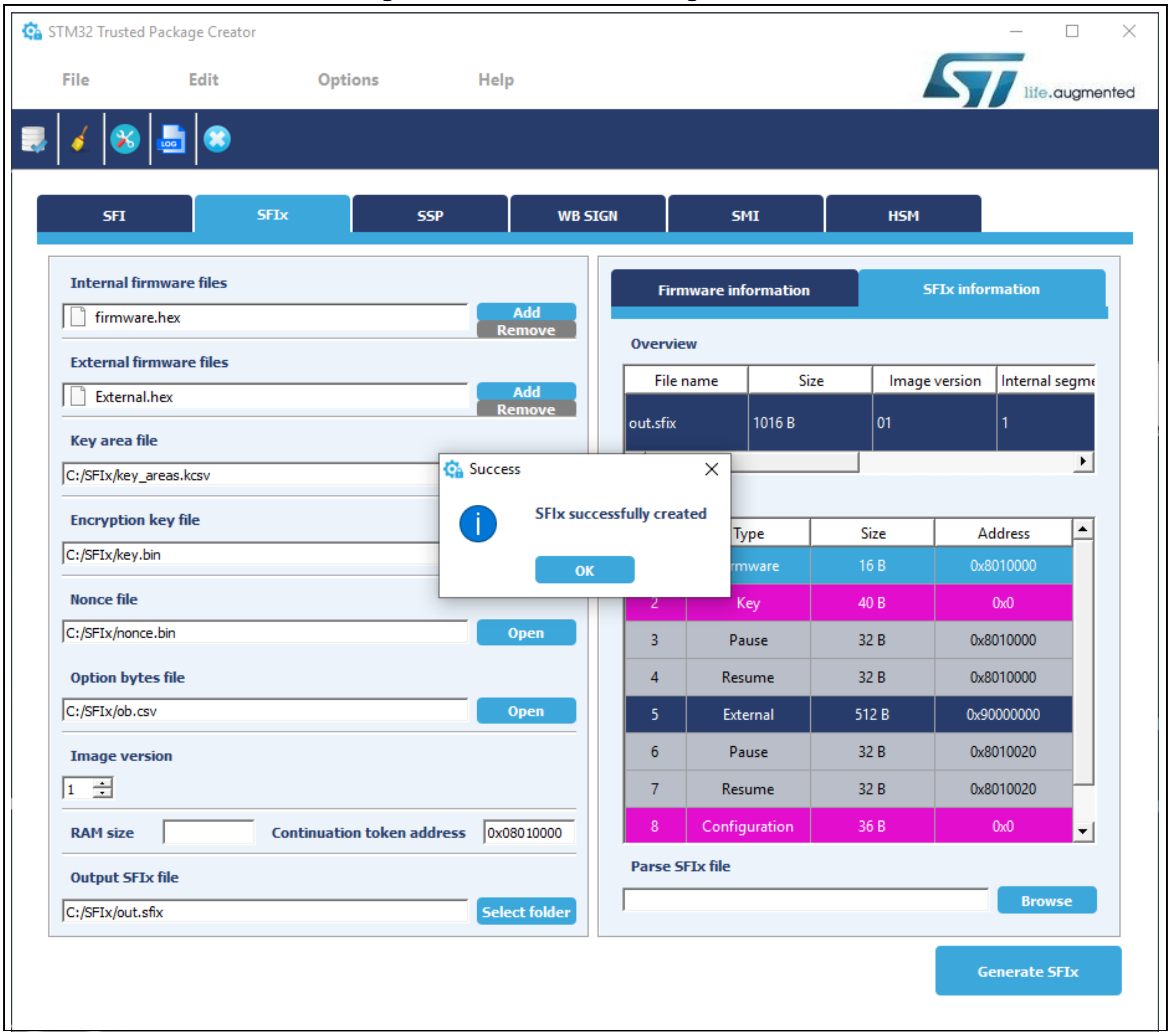
Note: *STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.*
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

An *".sfix"* image is then generated (*out.sfix)*.

Figure 67. Successful SFIx generation



9.3.3 Performing HSM programming for license generation using STPC (GUI mode)

The OEM must provide a license generation tool to the programming house to be used for license generation during the SFI install process.

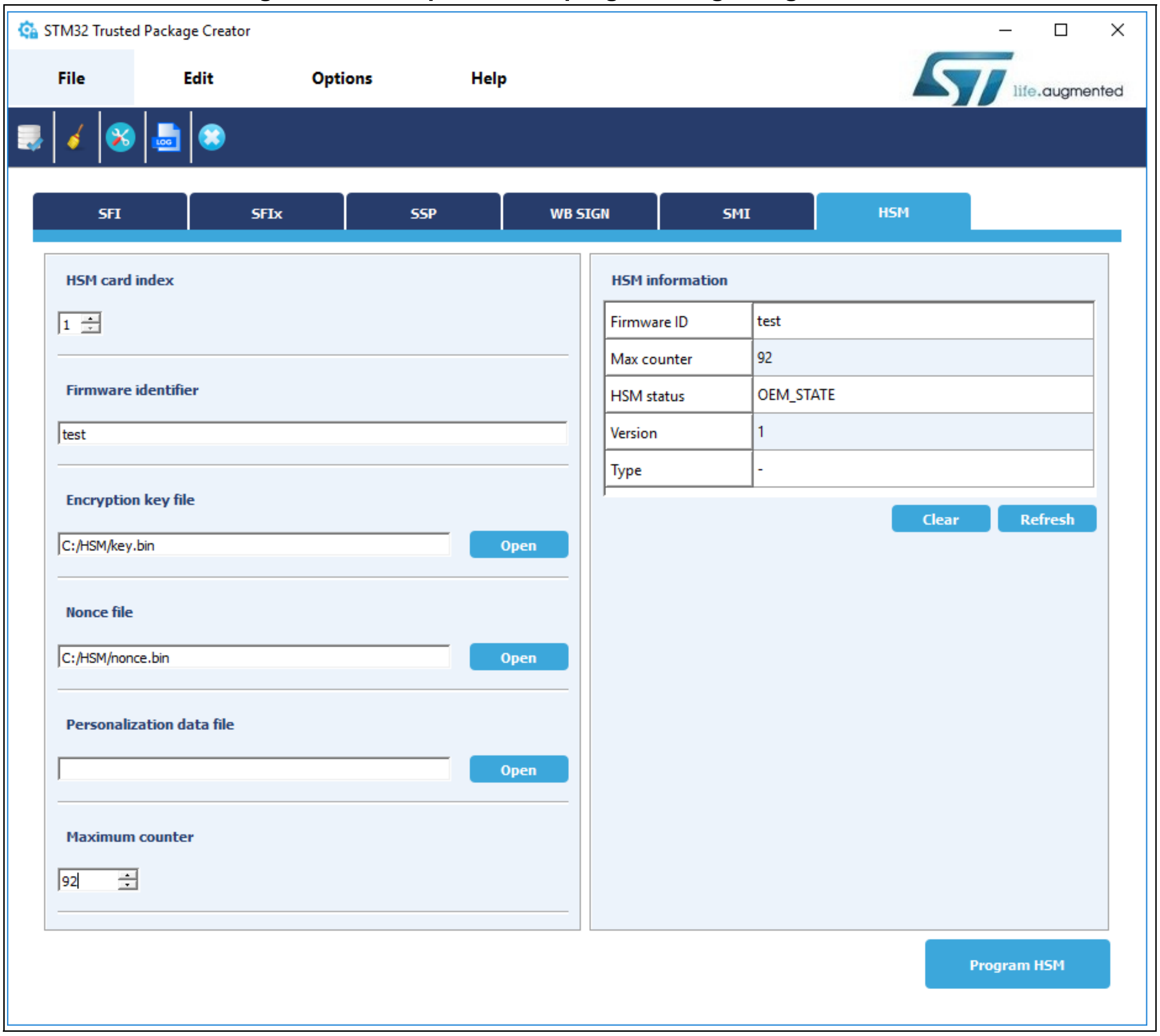
In this example, HSMs are used as license generation tools in the field. See *Section 4.1.2: License mechanism* for HSM use and programming.

Figure 68: Example of HSM programming using STPC GUI shows an example for HSM programming by OEM to be used for SFIx install.

The maximum number of licenses delivered by the HSM in this example is 1000.

This example uses HSM version 1. The HSM version can be identified before performing the programming operation by clicking the “Refresh” button to make the version number appear in the Version field.

Figure 68. Example of HSM programming using STPC GUI



Note: When using HSM version 1, the "Personalization data file" field is ignored when programming starts. It is only used with HSM version 2.

When the card is successfully programmed, a popup window message "HSM successfully programmed" appears, and the HSM is locked. Otherwise an error message is displayed.

9.3.4 Performing HSM programming for license generation using STPC (CLI mode)

Refer to *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.

9.3.5 Programming input conditions

Before performing an SFIx install make sure that:

- Use JTAG/SWD interface.
- No PCROPed zone is active, otherwise disable it.
- The chip must support security (a security bit must be present in the option bytes).
- The SFIx image must be encrypted by the same key/nonce used in the HSM provisioning.

9.3.6 Perform the SFIx install using STM32CubeProgrammer

In this section the STM32CubeProgrammer tool is used in CLI mode (the only mode so-far available for secure programming) to program the SFIx image *"out.sfix"* already created in the previous section.

STM32CubeProgrammer supports communication with ST HSMs (hardware secure modules based on smart card) to generate a license for the connected STM32 device during SFIx install.

After making sure that all the input conditions are respected, open a cmd terminal and go to *<STM32CubeProgrammer_package_path>/bin*, then launch the following STM32CubeProgrammer command:

Using JTAG/SWD

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local_path>/out.sfix" hsm=1 slot=<slot_id> -el <ExternalLoader_Path>
```

Figure 69: SFIx install success using SWD connection (1) through *Figure 72: SFIx install success using SWD connection (4)* shows the SFIx install via SWD execution and the HSM as license generation tool in the field.

Figure 69. SFIx install success using SWD connection (1)

```
      -------------------------------------------------------------------
                       STM32CubeProgrammer v2.3.0
      -------------------------------------------------------------------

ST-LINK SN  : 004000193037510B35333131
ST-LINK FW  : V3J1M1
Voltage     : 3.28V
SWD freq    : 24000 KHz
Connect mode: Hot Plug
Reset mode  : Core reset
Device ID   : 0x480
Device name : STM32H7A/B
Flash size  : 2 MBytes
Device type : MCU
Device CPU  : Cortex-M7



  Protocol Information         : static

  SFI File Information         :

     SFI file path             :
     SFI HSM slot ID           : 1
     SFI header information    :
         SFI protocol version        : 1
         SFI total number of areas   : 7
         SFI image version           : 1
     SFI Areas information     :

     Parsing Area 1/7     :
         Area type                   : F
         Area size                   : 16
         Area destination address    : 0x8000000

     Parsing Area 2/7     :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x8001000

     Parsing Area 3/7     :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x8001000

     Parsing Area 4/7     :
         Area type                   : E
         Area size                   : 528
         Area destination address    : 0x90000000
```

Figure 70. SFIx install success using SWD connection (2)

```
     Parsing Area 5/7    :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x8001020

     Parsing Area 6/7    :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x8001020

     Parsing Area 7/7    :
         Area type                   : C
         Area size                   : 36
         Area destination address    : 0x0

Reading the chip Certificate...


Requesting Chip Certificate from device ...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62070000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x620EF560
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Erase external flash size : 513 startAddress : 0x90000000 endAddress : 0x90000200
Erasing external memory sector 0
```

Figure 71. SFIx install success using SWD connection (3)

```
Activating security...
Warning: Option Byte: SECURITY, value: 0x1, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Activating security Success
Setting write mode to SFI
Warning: Option Byte: SECURITY, value: 0x1, was not modified.
Warning: Option Byte: ST_RAM_SIZE, value: 0x3, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Succeed to set write mode for SFI

Starting SFI part 1

Writing license to address 0x24030800
Writing Img header to address 0x24031000
Writing areas and areas wrapper...
RSS process started...

RSS command execution OK
RSS complete Value = 0x0
Reconnecting...
ST-LINK SN   : 004000193037510B35333131
ST-LINK FW   : V3J1M1
Voltage      : 3.28V
SWD freq     : 24000 KHz
Connect mode: Hot Plug
Reset mode   : Core reset
Device ID    : 0x480
Reconnected !

Requesting security state...
Warning: Could not verify security state after last chunk programming

Starting SFI part 2

Writing license to address 0x24030800
Writing Img header to address 0x24031000
Writing areas and areas wrapper...
RSS process started...

RSS command execution OK
RSS complete Value = 0x0
Reconnecting...
ST-LINK SN   : 004000193037510B35333131
ST-LINK FW   : V3J1M1
Voltage      : 3.28V
SWD freq     : 24000 KHz
Connect mode: Hot Plug
Reset mode   : Core reset
Device ID    : 0x480
Reconnected !

Requesting security state...
Warning: Could not verify security state after last chunk programming
```

Figure 72. SFIx install success using SWD connection (4)

```
Downloading area [3] data for external flash memory at address 0x90000000...
Data download complete

Starting SFI part 3

Writing license to address 0x24030800
Writing Img header to address 0x24031000
Writing areas and areas wrapper...
all areas processed
RSS process started...

RSS command execution OK
Warning: Could not verify security state after last chunk programming
SFI Process Finished!
SFI file C:\Users\

Time elapsed during SFI install operation: 00:00:44.321
```

10 Example of SFIx programming scenario for STM32L5

10.1 Scenario overview

There are three steps during this scenario:

- Generate SFIx image using the STPC
- HSM card provisioning via STPC
- Use STM32CubePrg to perform the SFIx process.

Successful installation of this scenario on the STM32L5 provides the following results:

- The internal Flash memory is readable from base addresses 0x08000000 and 0x08040000. It contains the internal firmware.
- The external Flash is programmed so as to be readable with external Flash loader. You can then read the external Flash encrypted by the OTFDEC keys. The pattern of values must be present in the binary files of external firmware.
- If the application works correctly, LED4 blinks.

10.2 Hardware and software environment

For successful SFIx programming, some hardware and software prerequisites apply:

- an STM32L5-based evaluation board containing external Flash memory
- a Micro-USB for debug connection
- a PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra
- an STM32TrustPackageCreator v1.2.0 (or greater) package available from *www.st.com*
- an STM32CubeProgrammer V2.3.0 (or greater) package available from *www.st.com*
- an HSMv1.1 card.

10.3 Step-by-step execution

10.3.1 Build OEM application

OEM application developers can use any IDE to build their own firmware. Note that in this use case there are different user codes, each being specific for a Flash memory type (internal/external).

10.3.2 Perform the SFIx generation (GUI mode)

To be encrypted with the STM32 Trusted Package Creator tool, OEM firmware is provided in Bin/Hex/AXF format in addition to a CSV file to set the option bytes configuration. A 128-bit AES encryption key and a 96-bit nonce are also provided to the tool.

Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

An *".sfix"* image is then generated (*out.sfix)*.

Use case 1 generation of SFIx without key area:

Internal firmware files:

1. Add a non-secure binary with start address equal to 0x08040000.
2. Add an internal binary file at 0x0C000000 (application to be executed after downloading SFIx to verify full process success by blinking an LED).
3. Add an OTFDEC key binary at 0x0C020000 (to be used as the key in OTFD ENC-DEC).

External FW files: add an external binary at 0x90000000 with these parameters:

- Region number = 0
- Region mode = 0x2
- Key address = 0x0C020000 (same as the OTFDEC key binary).

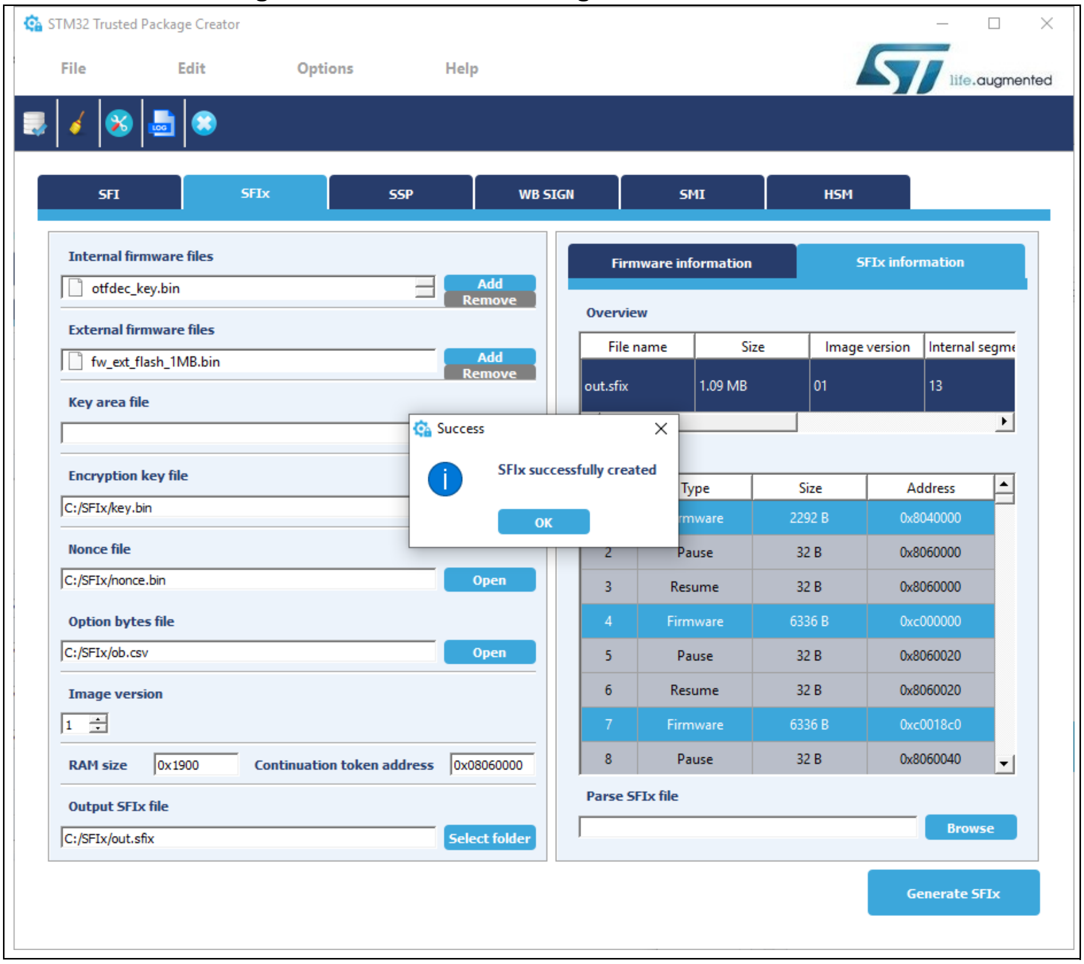
Encryption key: use the same key as HSM.

Nonce file: use the same nonce as HSM.

Option bytes file: use .csv contains option-byte configuration.

RAM size: 0x19000 to split the input areas avoiding memory overflow.

Figure 73. Successful SFIx generation use case 1

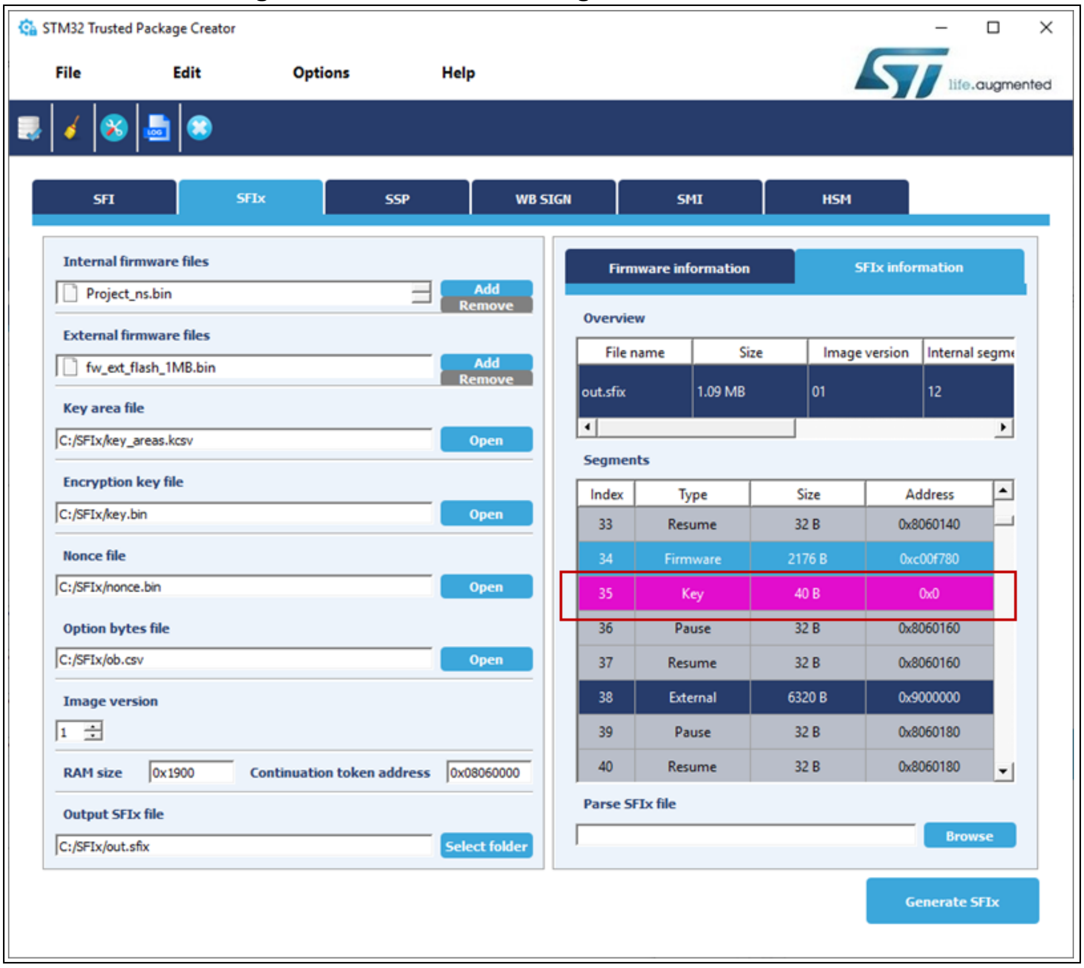


Use case 2 generation of SFIx with key area:

This is essentially the same process as test case1. The main difference is:

- Add a ".kcsv" file (to be used in OTFD ENC-DEC during SFIx downloading) in the key area field, instead of using an OTFDEC key binary file.
- The key address for external FW files is the first address of the Area 'K' key file, which is 0x0C020000.

Figure 74. Successful SFIx generation use case 2



After the generation of the SFIx image in this use case the output file should contain 12 internal segments (F area), and 166 external segments (E area).

10.3.3 Performing HSM programming for license generation using STPC (GUI mode)

Refer to *Section 9.3.3: Performing HSM programming for license generation using STPC (GUI mode)*.

10.3.4 Performing HSM programming for license generation using STPC (CLI mode)

Refer to *Section 9.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.

10.3.5 Programming input conditions

Before performing an SFIx install make sure that:

- A JTAG/SWD interface is used
- The chip supports security (a security bit must be present in the option bytes)
- The SFIx image is encrypted by the same key/nonce as is used in the HSM provisioning.
- The option bytes are:
  - DBank=1
  - nSWBOOT0=1
  - nBOOT0=1
  - RDP=AA

10.3.6 Perform the SFIx install using STM32CubeProgrammer

In this section the STM32CubeProgrammer tool is used in CLI mode (the only mode so-far available for secure programming) to program the SFIx image *"out.sfix"* already created in the previous section.

STM32CubeProgrammer supports communication with ST HSMs (Hardware Secure Modules based on smart card) to generate a license for the connected STM32 device during SFIx install.

Using JTAG/SWD

After making sure that all the input conditions are respected, open a cmd terminal and go to *<STM32CubeProgrammer_package_path>/bin*, then launch the following STM32CubeProgrammer command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local_path>/out.sfix" hsm=1 slot=<slot_id> -rsse <RSSe_Path>  -el
<ExternalLoader_Path>
```

Note: The RSSe binary file is located in STM32CubeProgrammer install path in the bin/RSSe folder.

Figure 75: SFIx install success using SWD connection (1) through *Figure 79: SFIx install success using SWD connection (5)* show the SFIx install via SWD execution and the HSM as license generation tool in the field.

Figure 75. SFIx install success using SWD connection (1)

```
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin>STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG reset=Crst -sfi protocol=static C:\data_store\app_LED_V.sfix hsm=1 slot=1 -rsse "C:\Program Files\STMicroelectronics\
STM32Cube\STM32CubeProgrammer\bin\RSSe\L5\enc_signed_RSSe_sfi_jtag.bin" -el ExternalLoader\MX25LM51245G_STM32L552E-EVAL-SFIX.stldr
      -------------------------------------------------------------------
                        STM32CubeProgrammer v2.3.0
      -------------------------------------------------------------------

ST-LINK SN  : 0670FF535351717867133324
ST-LINK FW  : V2J34M25
Voltage     : 3.26V
SWD freq    : 4000 KHz
Connect mode: Hot Plug
Reset mode  : Core reset
Device ID   : 0x472
Device name : STM32L5xx
Flash size  : 512 KBytes
Device type : MCU
Device CPU  : Cortex-M33

  Protocol Information         : static

  SFI File Information         :

     SFI file path             : C:\data_store\app_LED_V.sfix
     SFI HSM slot ID           : 1
     SFI header information    :
         SFI protocol version        : 1
         SFI total number of areas   : 39
         SFI image version           : 0
     SFI Areas information     :

     Parsing Area 1/39    :
         Area type                   : F
         Area size                   : 65536
         Area destination address    : 0xc000000

     Parsing Area 2/39    :
         Area type                   : F
         Area size                   : 16
         Area destination address    : 0xc020000

     Parsing Area 3/39    :
         Area type                   : F
         Area size                   : 2292
         Area destination address    : 0xc040000
```

Figure 76. SFIx install success using SWD connection (2)

```
     Parsing Area 4/39    :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x0

     Parsing Area 5/39    :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x0

     Parsing Area 6/39    :
         Area type                   : E
         Area size                   : 102336
         Area destination address    : 0x90000000

     Parsing Area 7/39    :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x20

     Parsing Area 8/39    :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x20

     Parsing Area 9/39    :
         Area type                   : E
         Area size                   : 102336
         Area destination address    : 0x90018fb0

     Parsing Area 10/39    :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x40

     Parsing Area 11/39    :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x40

     Parsing Area 12/39    :
         Area type                   : E
         Area size                   : 102336
         Area destination address    : 0x90031f60

     Parsing Area 13/39    :
         Area type                   : P
         Area size                   : 32
         Area destination address    : 0x60

     Parsing Area 14/39    :
         Area type                   : R
         Area size                   : 32
         Area destination address    : 0x60
```

Figure 77. SFIx install success using SWD connection (3)

```
Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Set RDP to 0xAA


Reconnecting...
Reconnected !
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi_jtag.bin
  File          : enc_signed_RSSe_sfi_jtag.bin
  Size          : 33200 Bytes
  Address       : 0x20005100


Erasing memory corresponding to segment 0:
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:00.455
Warning: Option Byte: TZEN, value: 0x1, was not modified.
Warning: Option Byte: nBoot0, value: 0x0, was not modified.
Warning: Option Byte: nSWBoot0, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Warning: Option Byte: SECBOOTADD0, value: 0x1FF000, was not modified.
Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified.
Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded


Reconnecting...
Reconnected !
Get RSSe status...
Erase external flash size : 1048577 startAddress : 0x90000000 endAddress : 0x90100000
Erasing external memory sectors [0 16]
Starting SFI

Processing license...
Processing Image Header
Processing Area 1...
Area Address = 0xc000000
Area Type    = F
Processing Area 2...
Area Address = 0xc020000
Area Type    = F
Processing Area 3...
Area Address = 0xc040000
```

Figure 78. SFIx install success using SWD connection (4)

```
Area Type    = F
Processing Area 4...
Area Address = 0x0
Area Type    = P
Processing Area 5...
Area Address = 0x0
Area Type    = R
Processing Area 6...
Area Address = 0x90000000
Area Type    = E
Buffer Address = 0x20005100

E Area Full Size = 102336

E Area Data Size = 102320

Processing Area 7...
Area Address = 0x20
Area Type    = P
Processing Area 8...
Area Address = 0x20
Area Type    = R
Processing Area 9...
Area Address = 0x90018fb0
Area Type    = E
Buffer Address = 0x20005100

E Area Full Size = 102336

E Area Data Size = 102320

Processing Area 10...
Area Address = 0x40
Area Type    = P
Processing Area 11...
Area Address = 0x40
Area Type    = R
Processing Area 12...
Area Address = 0x90031f60
Area Type    = E
Buffer Address = 0x20005100

E Area Full Size = 102336

E Area Data Size = 102320
```

Figure 79. SFIx install success using SWD connection (5)

```
Area Address = 0x140
Area Type    = R
Processing Area 36...
Area Address = 0x900f9ce0
Area Type    = E
Buffer Address = 0x20005100

E Area Full Size = 25392

E Area Data Size = 25376

Processing Area 37...
Area Address = 0x160
Area Type    = P
Processing Area 38...
Area Address = 0x160
Area Type    = R
Processing Area 39...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file C:\data_store\app_LED_V.sfix Install Operation Success

Time elapsed during SFI install operation: 00:01:25.116
```

11 Example of combined SFI-SMI programming scenario

11.1 Scenario overview

The user application to be installed on the STM32H753xI device makes `"printf"` packets appear in the serial terminal.

In this case the OEM application is built based on a third party's library as explained in IAR example (*Section 2.3: Execute-only/position independent library scenario example under EWARM*).

The application is encrypted using the STPC, the SMI module corresponding to 3rd party's library code is uploaded as input during combined SFI generation and represented as an area of type 'M' within firmware application areas.

The SFI OEM application firmware could then be uploaded (on an OEM server for example) with all the inputs needed for license generation by the CM.

The OEM provides tools to the CM to get the appropriate licenses for the SFI application concerned and the integrated SMI module(s).

11.2 Hardware and software environment

The same environment as explained in *Section 5.2: Hardware and software environment*.

11.3 Step-by-step execution

1. Build the OEM application.

   OEM application developers may use any IDE to build their firmware as well as using SMI modules provided by STMicoelectronics or 3rd parties for example.

   In this example we use firmware based on a single library (just one SMI module is integrated in the SFI image).
2. Perform the SFI generation.

   For encryption with the STM32 Trusted Package Creator tool, OEM firmware and the clear data part are both provided in Hex format (corresponding to the SMI module to be integrated within the SFI image). A CSV file to set the option bytes configuration is also necessary. The SMI module used is also provided as an input to the tool, in addition to a 128-bit AES encryption key and a 96-bit nonce. All inputs needed are available in the *"SFI_ImagePreparation/Combined"* directory. A *".sfi"* image is then generated (*out_comb.sfi*).

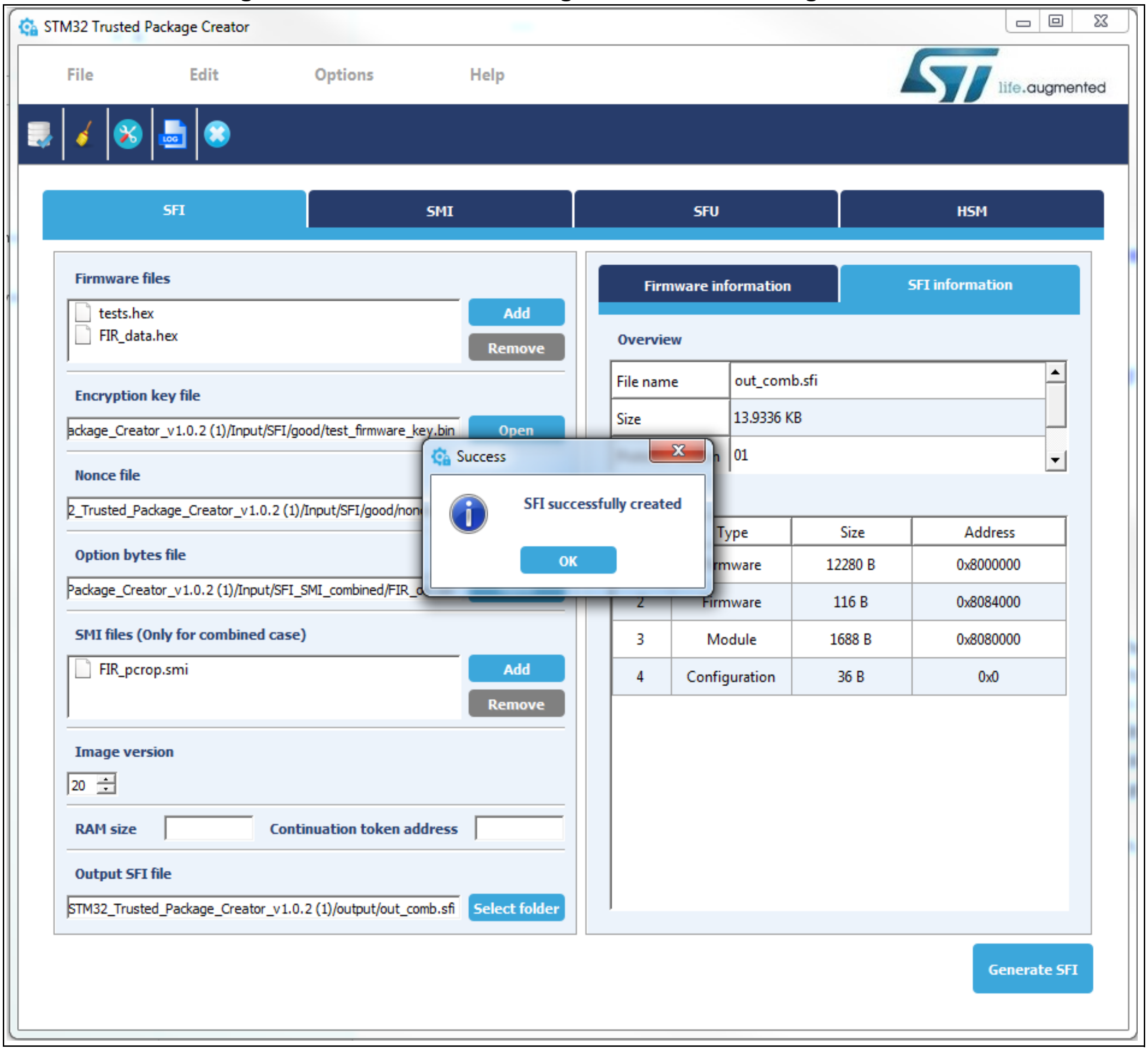
Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

Figure 80 shows the STPC GUI during combined SFI generation.

Figure 80. GUI of STPC during combined SFI-SMI generation



3. Programming input conditions are the same as for the SFI programming scenario (*Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*).
4. Perform the SFI install using the SWD/JTAG or a bootloader interface (here the SWD interface is used).

11.3.1 Using JTAG/SWD

Once all input conditions are respected, go to the “*stm32_programmer_package_v0.4.1/bin”* directory and launch the following command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi
protocol=static "<local_path>/out_comb.sfi" "<local_path>/
<licenseSFI.bin>"
```

Once all input conditions are respected, go to the “*<STM32CubeProgrammer_package_path>/bin”* directory and launch the following command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi
protocol=static "<local_path>/out_comb.sfi"
"<local_path>/<licenseSFI.bin>"
```

Figure 81: Combined SFI-SMI programming success using debug connection shows the combined SFI-SMI install trace success.

Figure 81. Combined SFI-SMI programming success using debug connection

```
ST-LINK Firmware version : V2J26M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450


  Protocol           : static
  SFI File           : RefSFI_MDK/SFI_Combined/out_comb.sfi

Starting SFI install operation for file :  RefSFI_MDK/SFI_Combined/out_comb.sfi
 ...
  SFI File Information          :

     SFI file path              : RefSFI_MDK/SFI_Combined/out_comb.sfi
     SFI license file path      : RefSFI_MDK/SFI_Combined/licenseSFIcomb_h753bEH.
bin
     SFI header information     :
         SFI protocol version         : 1
         SFI total number of areas    : 4
         SFI image version            : 23
     SFI Areas information      :

     Parsing Area 1/4     :
         Area type                    : F
         Area size                    : 12280
         Area destination address     : 0x8000000

     Parsing Area 2/4     :
         Area type                    : F
         Area size                    : 116
         Area destination address     : 0x8084000

     Parsing Area 3/4     :
         Area type                    : M
         Area size                    : 1688
         Area destination address     : 0x8080000

     Parsing Area 4/4     :
         Area type                    : C
         Area size                    : 36
         Area destination address     : 0x0

Setting write mode to SFI
Succeed to set write mode for SFI
Writing license to address 0x24007800
Writing Img header to address 0x24008000
Writing areas and areas wrapper...
RSS process started...

RSS command execution OK
Reconnecting...
ST-LINK Firmware version : V2J26M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450

Reconnected !

Requesting security state...
SECURITY State Success
SFI SUCCESS!
SFI file RefSFI_MDK/SFI_Combined/out_comb.sfi Install Operation Success

Time elapsed during the SFI install operation is: 00:00:04.056
Press <RETURN> to close this window...
```

11.3.2 How to test the combined SFI install success

The option bytes configuration must be modified as shown in *Figure 82: Option bytes after combined SFI-SMI install success*.

- 3rd party library module is programed into a PCROP area
- The SFI image is protected using RDP level1.

If a UART connection is available on the board used, open the "*Hercule.exe*" serial terminal available under the *"Tests*" directory, open the connection and on reset the dedicated `"printf"` packets appears.

Figure 82. Option bytes after combined SFI-SMI install success

```
OPTION BYTES BANK: 0
   Read Out Protection:
     RDP           : 0x0 (Level 1, read protection of memories)
   RSS:
     RSS1          : 0x0 (No SFI process on going)
   BOR Level:
     BOR_LEV       : 0x2 (reset level is set to 2.7 V)
   User Configuartion:
     IWDG1         : 0x1 (Independent watchdog is controlled by hardware)
     IWDG2         : 0x1 (Window watchdog is controlled by hardware)
     NRST_STOP_D1  : 0x1 (STOP mode on Domain 1 is entering without reset)
     NRST_STBY_D1  : 0x1 (STANDBY mode on Domain 1 is entering without reset)
     FZ_IWDG_STOP  : 0x1 (Independent watchdog is running in STOP mode)
     FZ_IWDG_SDBY  : 0x1 (Independent watchdog is running in STANDBY mode)
     SECURITY      : 0x1 (Security feature enabled)

     BCM7          : 0x1 (CM-7 boot enabled)
     NRST_STOP_D2  : 0x1 (STOP mode on Domain 2 is entering without reset)
     NRST_STBY_D2  : 0x1 (STANDBY mode on Domain 2 is entering without reset)
     SWAP_BANK     : 0x0 (after boot loading, no swap for user sectors)
     DMEPA         : 0x1 (delete PcROP protection and earse protected area)
     DMESA         : 0x1 (delete Secure protection and erase protected area)
   Boot address Option Bytes:
     BOOT_CM7_ADD0: 0x800  (0x8000000)
     BOOT_CM7_ADD1: 0x1FF1  (0x1FF10000)


   PCROP Protection:
     PCROPA_str    : 0x800  (0x8010000)
     PCROPA_end    : 0x806  (0x8080600)
   Secure Protection:
     SECA_str      : 0xFFF  (0x801FFE0)
     SECA_end      : 0x0  (0x80000FF)
   DTCM RAM Protection:
     ST_RAM_SIZE   : 0x3 (16 KB)
   Write Protection:
     nWRP0         : 0x1 (Write protection not active on this sector)
     nWRP1         : 0x1 (Write protection not active on this sector)
     nWRP2         : 0x1 (Write protection not active on this sector)
     nWRP3         : 0x1 (Write protection not active on this sector)
     nWRP4         : 0x1 (Write protection not active on this sector)
     nWRP5         : 0x1 (Write protection not active on this sector)
     nWRP6         : 0x1 (Write protection not active on this sector)
     nWRP7         : 0x1 (Write protection not active on this sector)
```

12 Example of SSP programming scenario for STM32MP1

12.1 Scenario overview

On each SSP install step, STM32 ecosystem tools are used to manage the secure programming and SSP flow.

Three main steps are done using SSP tools:

- Encrypted secret file generation with STM32 Trusted Package Creator
- HSM provisioning with STM32 Trusted Package Creator
- SSP procedure with STM32CubeProgrammer.

12.2 Hardware and software environment

The following prerequisites are needed for successful SSP programming:

- an STM32MP1-DK2 board
- a Micro-USB for DFU connection
- a PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer V2.5.0 (or greater) package available from www.st.com
- an HSMv2 card.

12.3 Step-by-step execution

12.3.1 Building a secret file

A secret file must be created prior to SSP processing. This secret file must fit into the OTP area reserved for the customer. OTP memory is organized as 32-bit words.

On an STM32MP1 microprocessor:

- One OTP word is reserved for RMA password (unlock/relock): OTP 56.
- 37 free words are reserved for customer use. The secret size can be up to 148 bytes: OTP 59 to 95.

There is no tool or template to create this file. A 148-byte binary file must be used as the reference to construct the secret file.

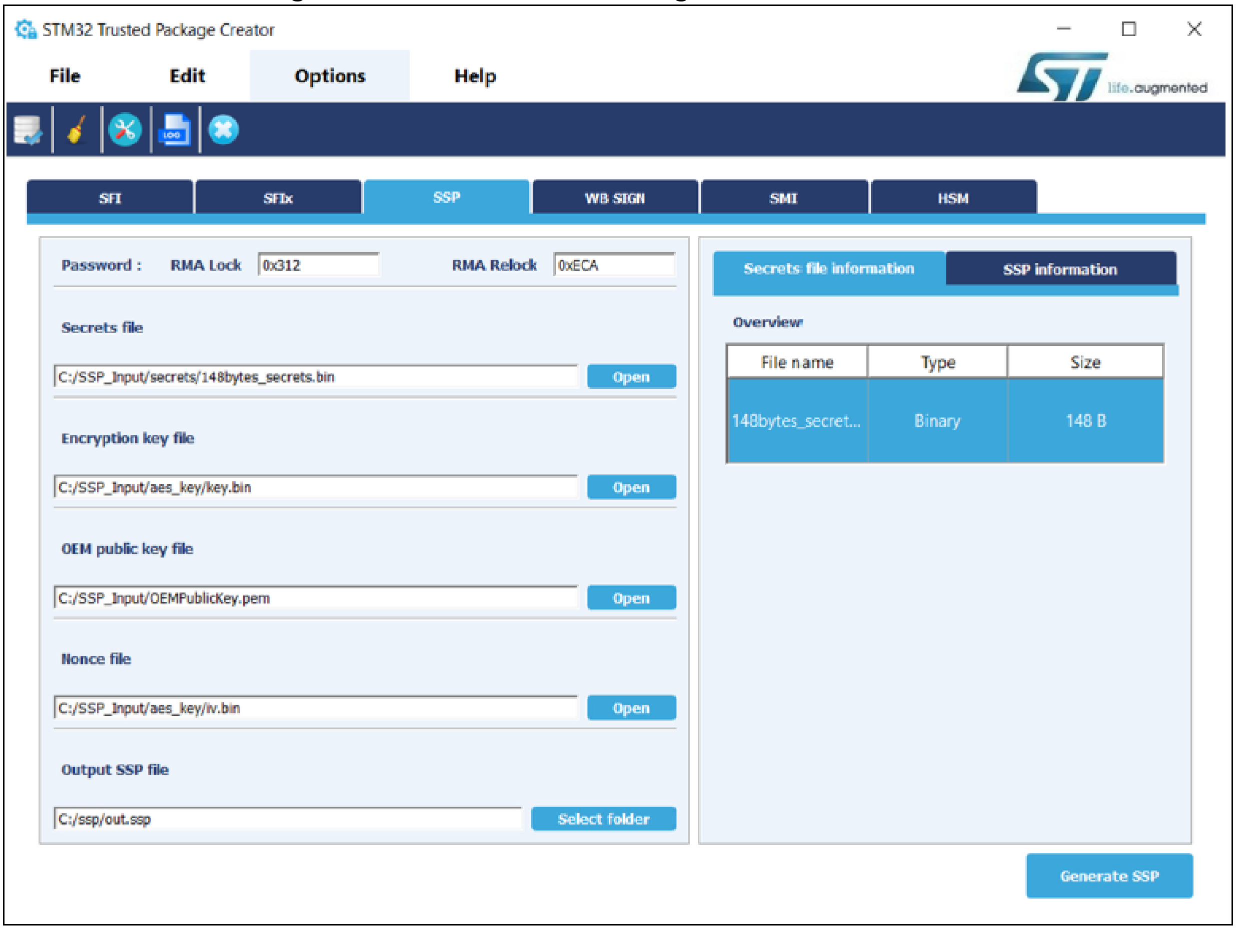
12.3.2 Performing the SSP generation (GUI mode)

For encryption with the STM32 Trusted Package Creator tool, the secret file is provided in BIN format in addition to the RMA password values.

An OEM public key, a 128-bit AES encryption key and a 96-bit nonce are also provided to the tool.

An *".ssp"* image is then generated (*out.ssp*).

Figure 83. STM32 Trusted Package Creator SSP GUI tab



12.3.3 Performing HSM programming for license generation using STPC (GUI mode)

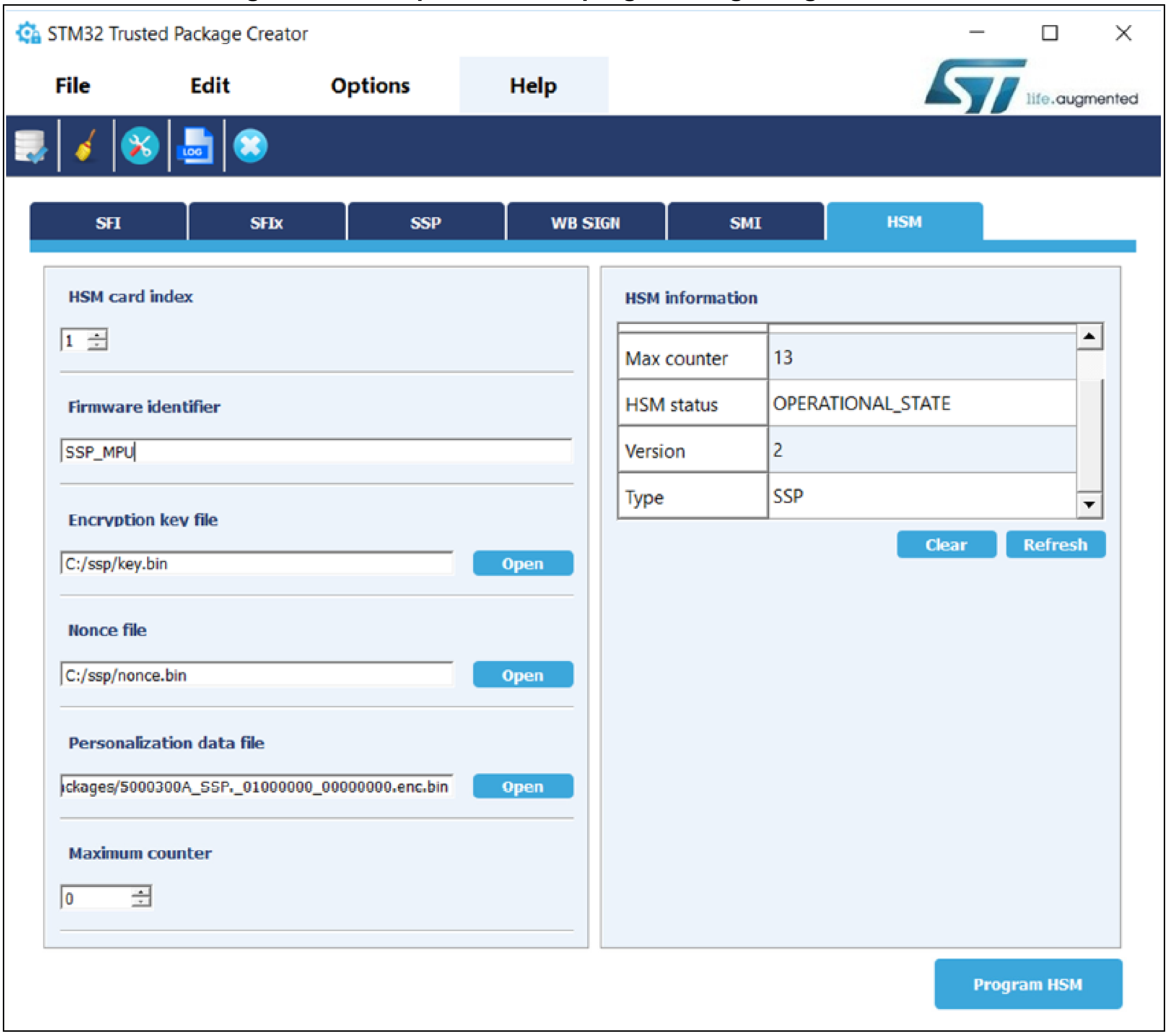
The OEM must provide a license generation tool to the programming house, to be used for license generation during the SSP install process. In this example, HSMs are used as license generation tools in the field.

See *Section 4.1.2: License mechanism* for HSM use and programming details.

This example uses HSM version 2. The HSM version can be identified before performing the programming operation by clicking the **Refresh** button to make the version number appear in the Version field.

Note: HSM version 2 must be used for STM32 MPU devices.

Figure 84. Example of HSMv2 programming using STPC GUI



The STM32 Trusted Package Creator tool provides all personalization package files, ready to be used on SSP flow. To obtain all the supported packages, go to the “PersoPackages” directory residing in the tool’s install path. Each file name starts with a number, which is the product ID of the device. The correct one must be selected.

12.3.4 SSP programming conditions

Before performing an SSP flow make sure that:

- a DFU or UART interface is used
- the chip supports security
- the SSP image is encrypted by the same key/nonce as used in the HSM provisioning step.

12.3.5 Perform the SSP install using STM32CubeProgrammer

In this step the STM32CubeProgrammer tool is used in CLI mode (the only mode available so far for secure programming) to program the SSP image already created with STM32 Trusted Package Creator. STM32CubeProgrammer supports communication with ST HSMs (hardware secure modules based on a Smart Card) to generate a license for the connected STM32 MPU device during SSP install.

Example using USB DFU bootloader interface:

```
STM32_Programmer_CLI.exe -c port=usb1 –ssp “out.ssp” “tf-a-ssp-
stm32mp157f-dk2-trusted.stm32” hsm=1 slot=1
```

All SSP traces are shown on the output console (*Figure 85*).

Figure 85. STM32MP1 SSP install success

```
Requesting Chip Certificate...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
P11 lib initialization Success!

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Writing blob


Blob successfully written

Start operation achieved successfully
Send detach command
Detach command executed
SSP file out.ssp Install Operation Success
```

13 Reference documents

Table 3. Document references

Reference	Document title
[1]	AN4992, STM32H7 secure firmware/module install overview. STMicroelectronics.
[2]	UM2428, Hardware secure modules (HSM) for secure firmware install (SFI). STMicroelectronics.
[3]	AN5510, Overview of secure secret provisioning (SSP)

14 Revision history

Table 4. Document revision history

Date	Revision	Changes
03-Aug-2018	1	Initial release.
18-Apr-2019	2	Updated publication scope from 'ST restricted' to 'Public'.
16-Oct-2019	3	Updated: – *Section 4.1.2: License mechanism* – *Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)* – *Figure 43: HSM programming GUI in the STPC tool* (title caption) – *Figure 52: Example of HSM programming using STPC GUI*
03-Feb-2020	4	Replaced occurrences of STM32L451CE with STM32L462CE in *Section 4.2.1: Secure firmware installation using a bootloader interface flow*. Updated document to cover secure programming with SFIx.
26-Feb-2020	5	Updated: – *Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow* – *Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)* – *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)* – *Figure 69: SFIx install success using SWD connection (1)* – *Figure 72: SFIx install success using SWD connection (4)*.
27-Jul-2020	6	Updated: – *Introduction* – *Section 3.1: System requirements* Added: – *Section 3.5: SSP generation process* – *Section 3.6.3: Steps for SSP generation (CLI)* – *Section 3.7.4: SSP generation using STPC in GUI mode* – *Section 4.2.5: STM32CubeProgrammer for SSP via a bootloader interface* – *Section 12: Example of SSP programming scenario for STM32MP1*.

Table 4. Document revision history

Date	Revision	Changes
19-Nov-2020	7	Updated: – *Introduction* on cover page – *License mechanism general scheme* – *HSM programming by OEM for license distribution* – *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*. Added: – *Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB)* – *Section 6: Example of SFI programming scenario for STM32WL*.
29-Jun-2021	8	Updated: – In the whole document, replaced STM32H7A/B by STM32H7A3/7B3 and STM32H7B0, STM32H72/3 by STM32H723/333 and STM32H725/335, STM32H7B board by STM32H7B3I-EVAL – Replaced BL by bootloader. – *Section 3.2: SFI generation process*: removed references to RSS. – *Section 4.1.2: License mechanism*: removed Figure *HSM programming toolchain*. – *Section 4.2: Secure programming using a bootloader interface*, *Section 4.2.2: Secure Module installation using a bootloader interface flow*, *Section 4.2.3: STM32CubeProgrammer for SFI using a bootloader interface* – *Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow* and *Section 4.3.2: SMI programming through JTAG/SWD flow*. – *Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB)* – *Example of SFI programming scenario*/*Section 5.2: Hardware and software environment* and *Example of SFI programming scenario for STM32WL*/*Section 6.2: Hardware and software environment*: removed bootloader and RSS versions – *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*: removed STM32L4 from the list of devices that support SFI via debug interface. Added: – Support for STM32U5 Series. – *Section 7: Example of SFI programming scenario for STM32U5*.

Table 4. Document revision history

Date	Revision	Changes
02-Aug-2021	9	Added note about CSV file in *Section 3.6.1: Steps for SFI generation (CLI)* and *Figure 26: Option bytes file example*. Corrected binary file names in *Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB)*. *Section 3.6.1: Steps for SFI generation (CLI)* Added note about option byte file example in: – *Section 3.7.1: SFI generation using STPC in GUI mode* – *Section 5.3.2: Perform the SFI generation (GUI mode)* – *Section 6.3.2: Perform the SFI generation (GUI mode)* – *Section 7.3.2: Perform the SFI generation (GUI mode)* – *Section 9.3.2: Perform the SFIx generation (GUI mode)* – *Section 10.3.2: Perform the SFIx generation (GUI mode)* – *Section 11.3: Step-by-step execution*. Updated Corrected board name in *Section 4.2: Secure programming using a bootloader interface*. Corrected board name in *Section 7.2: Hardware and software environment*.