

---

## Secure programming using STM32CubeProgrammer

---

### Introduction

This document specifies the steps and tools required to prepare SFI (secure firmware install) or SMI (secure module install) images (or a combination of both) and to program them into the STM32H7 on-chip Flash memory. It is based on the STM32CubeProgrammer tool set (STM32CubeProg).

These tools are compatible with all STM32 devices porting SFI.

The main objective of SFI and SMI processes, is the installation of security and cloning prevention in OEMs' and software-partner's firmware respectively.

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 the firmware development up to programming the device on-chip Flash memory.



# Contents

<b>1</b>	<b>General information</b>	<b>8</b>
1.1	Licensing information	8
1.2	Acronyms and abbreviations	8
<b>2</b>	<b>How to generate an execute-only/position independent library for SMI preparation</b>	<b>9</b>
2.1	Requirements	9
2.2	Toolchains allowing SMI generation	9
2.3	Execute-only/position independent library scenario example under EWARM	10
2.3.1	Relocatable library preparation steps	10
2.3.2	Relocatable SMI module preparation steps	14
2.3.3	Application execution Scenario	15
<b>3</b>	<b>Encrypted firmware (SFI) / module (SMI) preparation using STM32TrustedPackageCreator</b>	<b>17</b>
3.1	System requirements	17
3.2	SFI generation process	17
3.3	SMI generation process	26
3.4	STM32TrustedPackageCreator tool in the command line interface	29
3.4.1	Steps for SFI generation (CLI)	30
3.4.2	Steps for SMI generation(CLI)	32
3.5	Using the STM32TrustedPackageCreator tool graphical user interface	34
3.5.1	SFI generation using STPC in GUI mode	34
	SFI GUI tab fields	35
3.5.2	SMI generation using STPC in GUI mode	38
	SMI GUI tab fields	39
3.5.3	Settings	40
3.5.4	Log generation	42
3.5.5	SFI and SMI file checking function	43

<b>4</b>	<b>Encrypted firmware (SFI) / module (SMI) programming using STM32CubeProgrammer</b>	<b>44</b>
4.1	Chip certificate authenticity check and license mechanism	44
4.1.1	Device authentication	44
4.1.2	License mechanism	44
	Licenses mechanism general scheme	44
	License distribution	45
	HSM programming by OEM for License distribution	45
4.2	Secure programming using bootloader interface	47
4.2.1	Secure firmware installation using Bootloader interface flow	47
4.2.2	Secure Module installation using bootloader interface flow	49
4.2.3	STM32CubeProgrammer for SFI using bootloader interface	49
4.2.4	STM32CubeProgrammer for SMI via Bootloader interface	50
4.2.5	STM32CubeProgrammer for get certificate via Bootloader interface	50
4.3	Secure programming using JTAG/SWD interface	51
4.3.1	SFI programming using JTAG/SWD flow	51
4.3.2	SMI programming through JTAG/SWD flow	52
4.3.3	STM32CubeProgrammer for secure programming using JTAG/SWD	54
<b>5</b>	<b>Example SFI programming scenario</b>	<b>55</b>
5.1	Scenario overview	55
5.2	Hardware and software environment	55
5.3	Step-by-step execution	55
5.3.1	Build OEM application	55
5.3.2	Perform the SFI generation (GUI mode)	55
5.3.3	Performing HSM programming for license generation using STPC (GUI mode)	57
5.3.4	Performing HSM programming for license generation using STPC (CLI mode)	58
	Example of HSM version 1 provisioning:	58
	Example of HSM version 2 provisioning:	58
5.3.5	Programming input conditions	59
5.3.6	Perform the SFI install using STM32CubeProgrammer	60
	Using JTAG/SWD	60

<b>6</b>	<b>Example SMI programming scenario</b>	<b>63</b>
6.1	Scenario overview	63
6.2	Hardware and software environment	63
6.3	Step-by-step execution	63
6.3.1	Build 3rd party Library	63
6.3.2	Perform the SMI generation	64
6.3.3	Programming input conditions	65
6.3.4	Perform the SMI install	65
	Using JTAG/SWD	65
6.3.5	How to test for SMI install success	67
<b>7</b>	<b>Example combined SFI-SMI programming scenario</b>	<b>69</b>
7.1	Scenario overview	69
7.2	Hardware and software environment	69
7.3	Step-by-step execution	69
7.3.1	Using JTAG/SWD	71
7.3.2	How to test the combined SFI install success	73
<b>8</b>	<b>Reference documents</b>	<b>75</b>
<b>9</b>	<b>Revision history</b>	<b>76</b>

# List of tables

Table 1. List of abbreviations ..... 8

Table 2. Document references ..... 75

Table 3. Document revision history ..... 76

## List of figures

Figure 1.	IAR example project overview .....	10
Figure 2.	Update compiler extra options .....	11
Figure 3.	Linker extra options .....	12
Figure 4.	Setting post-build option .....	13
Figure 5.	Postbuild batch file .....	14
Figure 6.	How to exclude the "lib.o" file from build .....	15
Figure 7.	app.icf file .....	16
Figure 8.	SFI preparation mechanism .....	17
Figure 9.	SFI image process generation .....	18
Figure 10.	RAM size and CT address inputs used for SFI multi install .....	19
Figure 11.	'P' and 'R' area specifics versus a regular SFI area .....	20
Figure 12.	Error message when firmware files with address overlaps used .....	21
Figure 13.	Error message when SMI address overlaps with a firmware area address .....	22
Figure 14.	Error message when a SFI area address is not located in Flash memory .....	23
Figure 15.	SFI format layout .....	24
Figure 16.	SFI image layout in case of split .....	25
Figure 17.	SMI preparation mechanism .....	26
Figure 18.	SMI image generation process .....	27
Figure 19.	SMI format layout .....	28
Figure 20.	STM32TrustedPackageCreator tool - available commands .....	29
Figure 21.	Option bytes file example .....	31
Figure 22.	SFI generation example using an Elf file .....	31
Figure 23.	SMI generation example .....	33
Figure 24.	SFI generation Tab .....	34
Figure 25.	Firmware parsing example .....	35
Figure 26.	SFI successful generation in GUI mode example .....	37
Figure 27.	SMI generation Tab .....	38
Figure 28.	SMI successful generation in GUI mode example .....	40
Figure 29.	Settings icon and Settings dialog box .....	41
Figure 30.	Log example .....	42
Figure 31.	Check SFI file example .....	43
Figure 32.	HSM programming toolchain .....	46
Figure 33.	HSM programming GUI in the STPC tool .....	47
Figure 34.	Secure programming via STM32CubeProgrammer overview on STM32H7 devices ....	48
Figure 35.	Secure programming via STM32CubeProgrammer overview on STM32L4 devices ....	48
Figure 36.	Example of getcertificate command execution using UART interface .....	50
Figure 37.	SFI programming by JTAG/SWD flow overview (monolithic SFI image example) .....	52
Figure 38.	SMI programming by JTAG flow overview .....	53
Figure 39.	Example of getcertificate command using JTAG .....	54
Figure 40.	STPC GUI during SFI generation .....	56
Figure 41.	Example of HSM programming using STPC GUI .....	57
Figure 42.	HSM information in STM32TrustedPackageCreator CLI mode .....	59
Figure 43.	SFI install success using SWD connection (1) .....	61
Figure 44.	SFI install success using SWD connection (2) .....	62
Figure 45.	STPC GUI during SMI generation .....	64
Figure 46.	SMI install success via debug interface .....	66
Figure 47.	OB display command showing that a PCROP zone was activated after SMI. ....	68

Figure 48.	GUI of STPC during combined SFI-SMI generation . . . . .	70
Figure 49.	Combined SFI-SMI programming success using debug connection . . . . .	72
Figure 50.	Option bytes after combined SFI-SMI install success. . . . .	74

# 1 General information

## 1.1 Licensing information

STM32CubeProgrammer supports STM32 32-bit devices based on Arm<sup>®(a)</sup> Cortex<sup>®</sup>-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)
SFI	Secure firmware install
STPC	STM32TrustedPackageCreator
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, 3<sup>rd</sup> 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 3<sup>rd</sup> 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 should 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 EWARM toolchain.

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

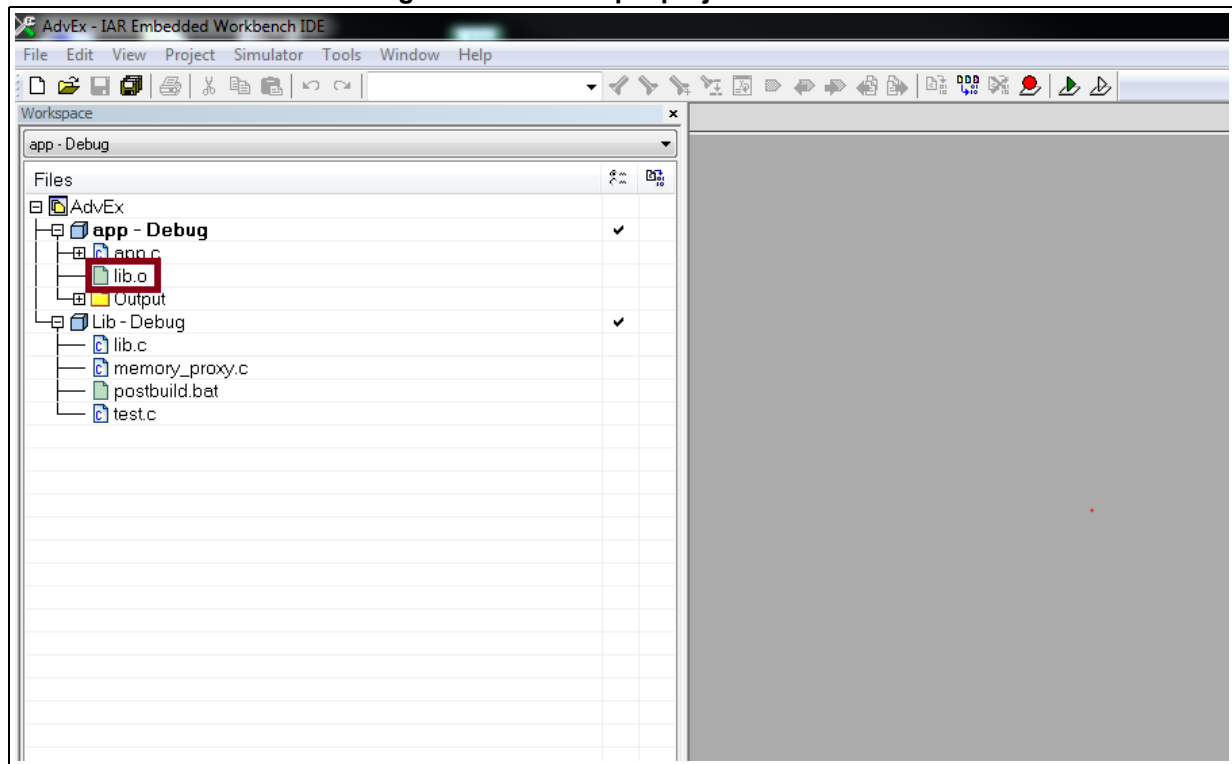
In order to generate an execute-only (XO) and position independent (PI) library a customized version of the IAR toolchain must be used: version 7.42.0.

### 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**

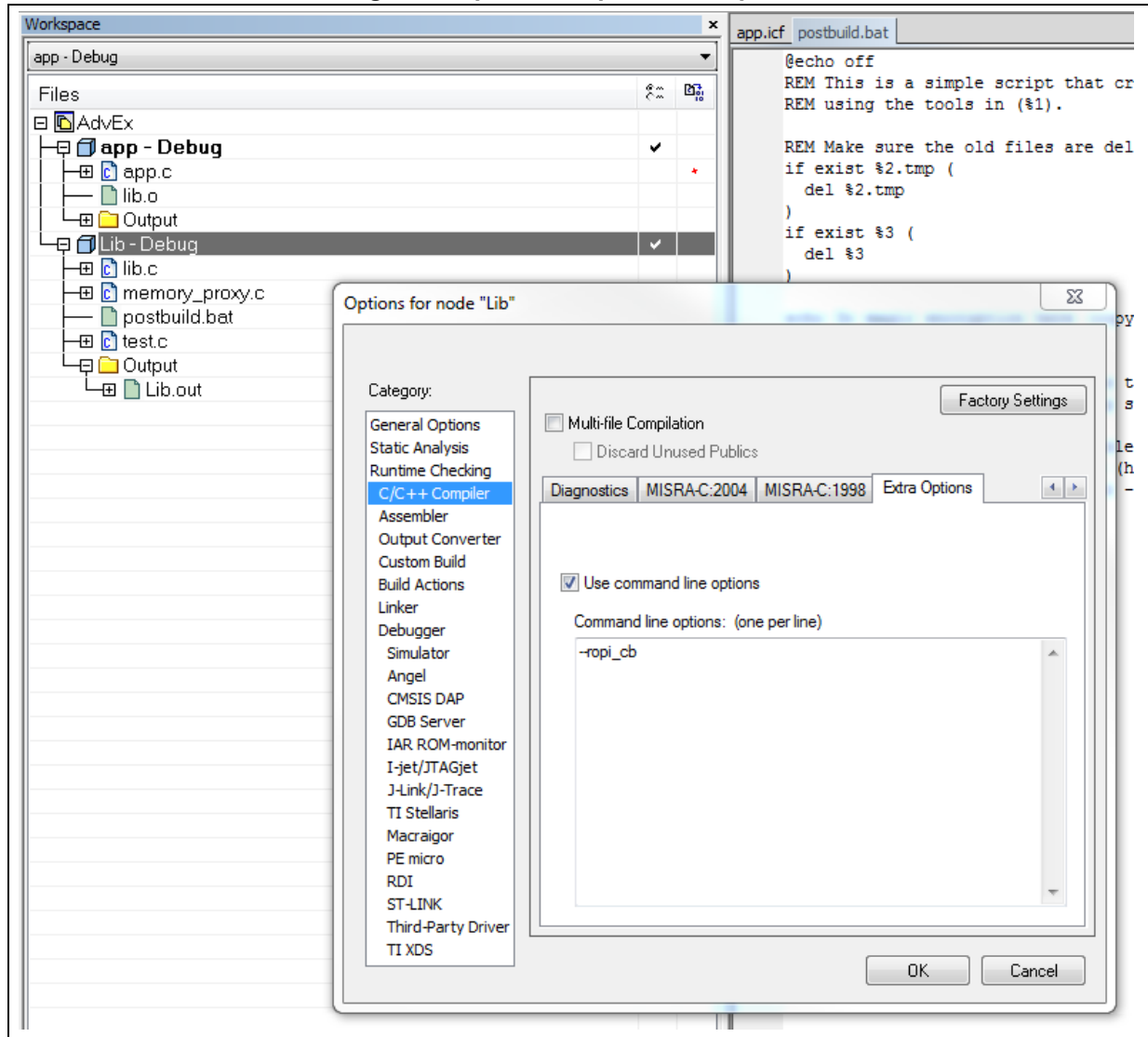


The following steps update the old “lib.o” linked to the example application by making it support both PI and XO features:

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**



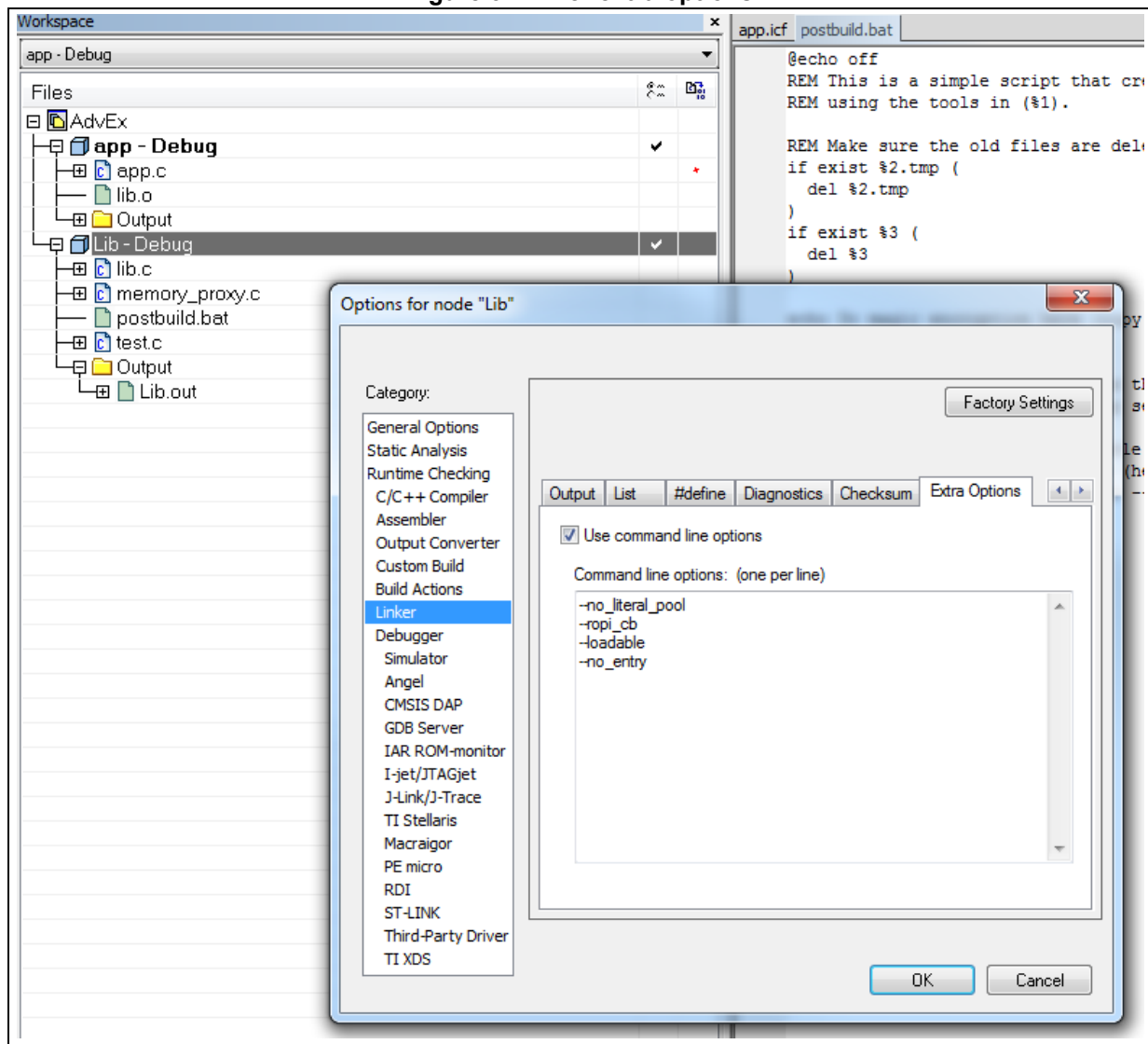
3. Within Lib-Debug options -> Linker: go to the “Extra Options” tab and add the following lines:

```
--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.

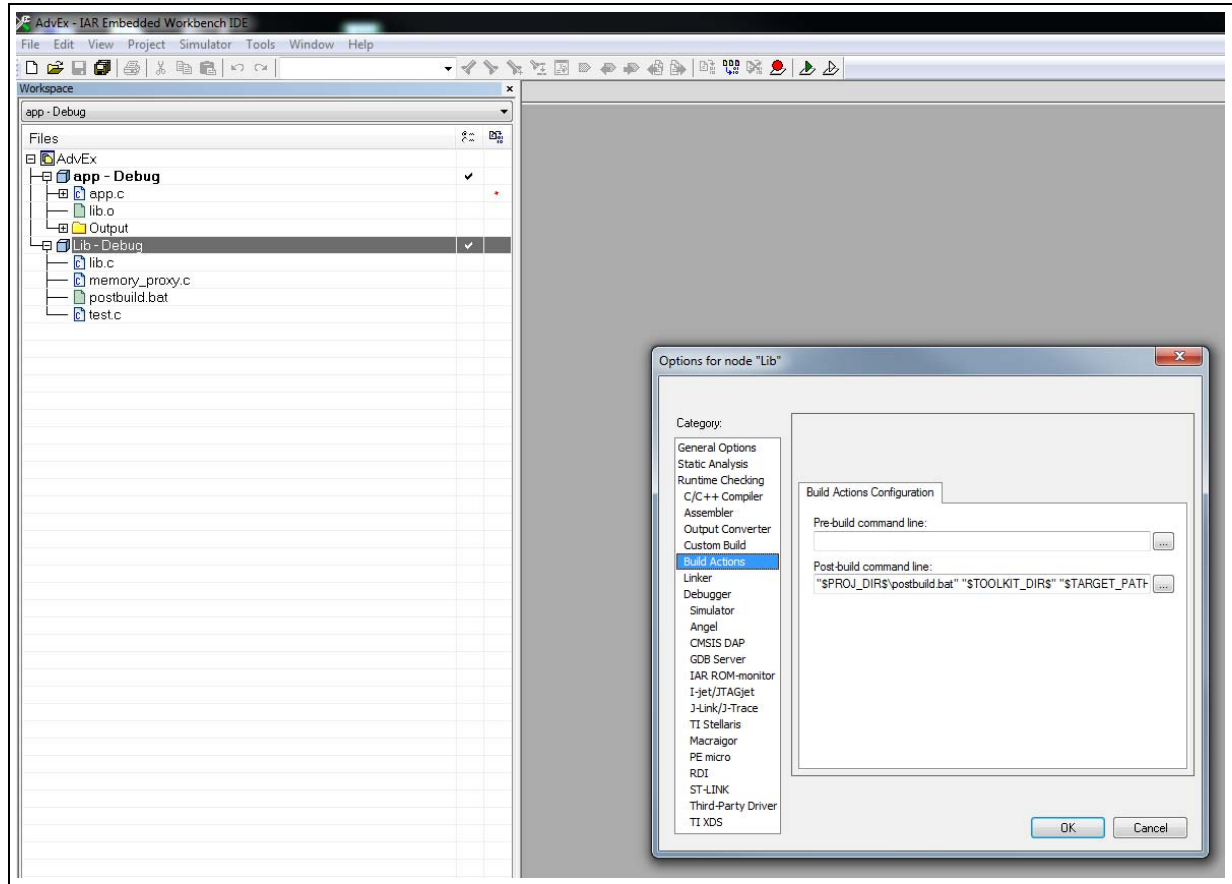
**Figure 3. Linker extra options**



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**

```
1 @echo off
2 REM This is a simple script that creates an object file (%3) from an image (%2)
3 REM using the tools in (%1).
4
5 REM Make sure the old files are deleted before we try to generate the new ones
6 if exist %2.tmp (
7     del %2.tmp
8 )
9 if exist %3 (
10     del %3
11 )
12
13 echo Do magic encryption here (copy is just a placeholder)
14 copy %2 %2.tmp
15
16 REM This is the list of functions that will have a wrapper generated
17 SET __WRAP=--wrap ToString --wrap setup_memory --wrap setup_memory2
18
19 REM convert the image to a linkable object file using _Lib as prefix
20 REM and keeping all mode symbols (helps a bit with debugging)
21 %1\bin\iexe2obj.exe --prefix Lib --keep mode symbols % __WRAP% %2.tmp %3
22
```

5. Rebuild the project "Lib"

### 2.3.2 Relocatable SMI module preparation steps

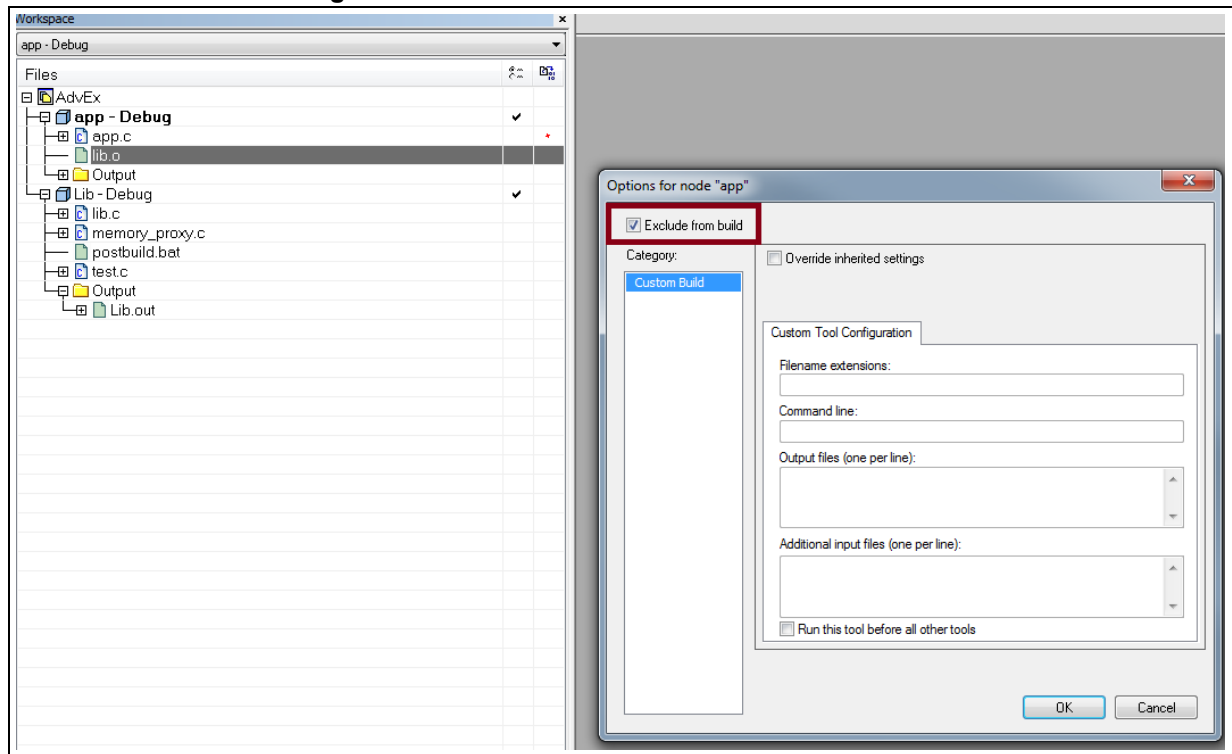
From the object file created, *"lib.o"*, generate the SMI relocatable module using the STM32TrustedPackageCreator tool *"libr.smi"* and its corresponding data clear part (*libr\_clear.o*: corresponding to the input *"lib.o"* without the protected section code).

To execute this step, follow the steps explained for SMI generation under section ["Section 3.4.2: 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 4 to perform this action).
2. Link the data clear part, "*libr\_clear.o*", generated from STM32TrustedPackageCreator 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 5](#).

Figure 7. app.icf file

```

app.icf
/** ICF Editor Section handled by ICF editor, don't touch! ****/
/*-Editor Annotation file-*/
/* IcfEditorFile="I:\TOOLKIT\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 no-load 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 should appear in the terminal output.



### 3 Encrypted firmware (SFI) / module (SMI) preparation using STM32TrustedPackageCreator

The STM32TrustedPackageCreator 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](http://www.st.com).

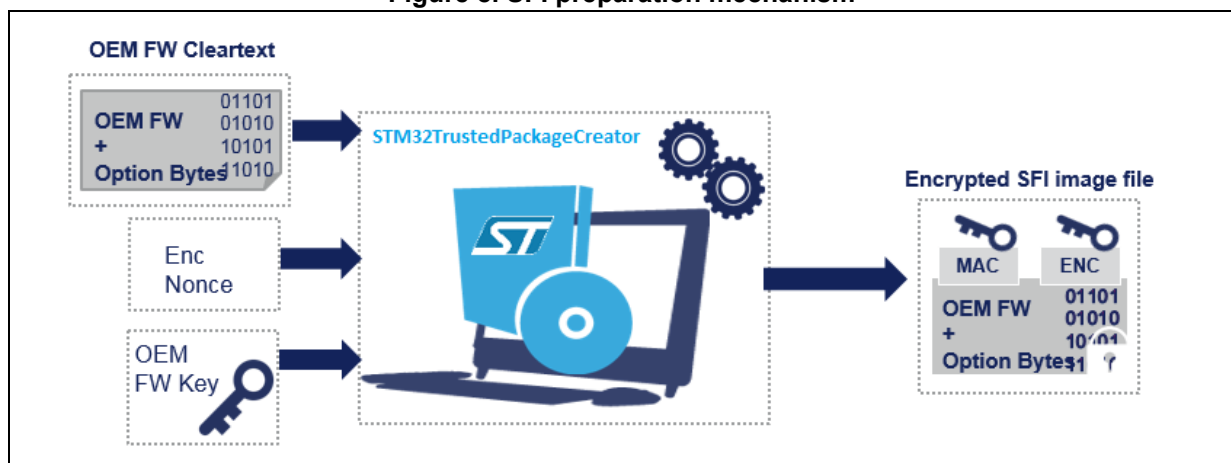
#### 3.1 System requirements

Using the STM32TrustedPackageCreator tool for SFI and SMI image generation requires a PC running on either Windows 7 or Ubuntu 14 in both 32-bit and 64-bit versions.

#### 3.2 SFI generation process

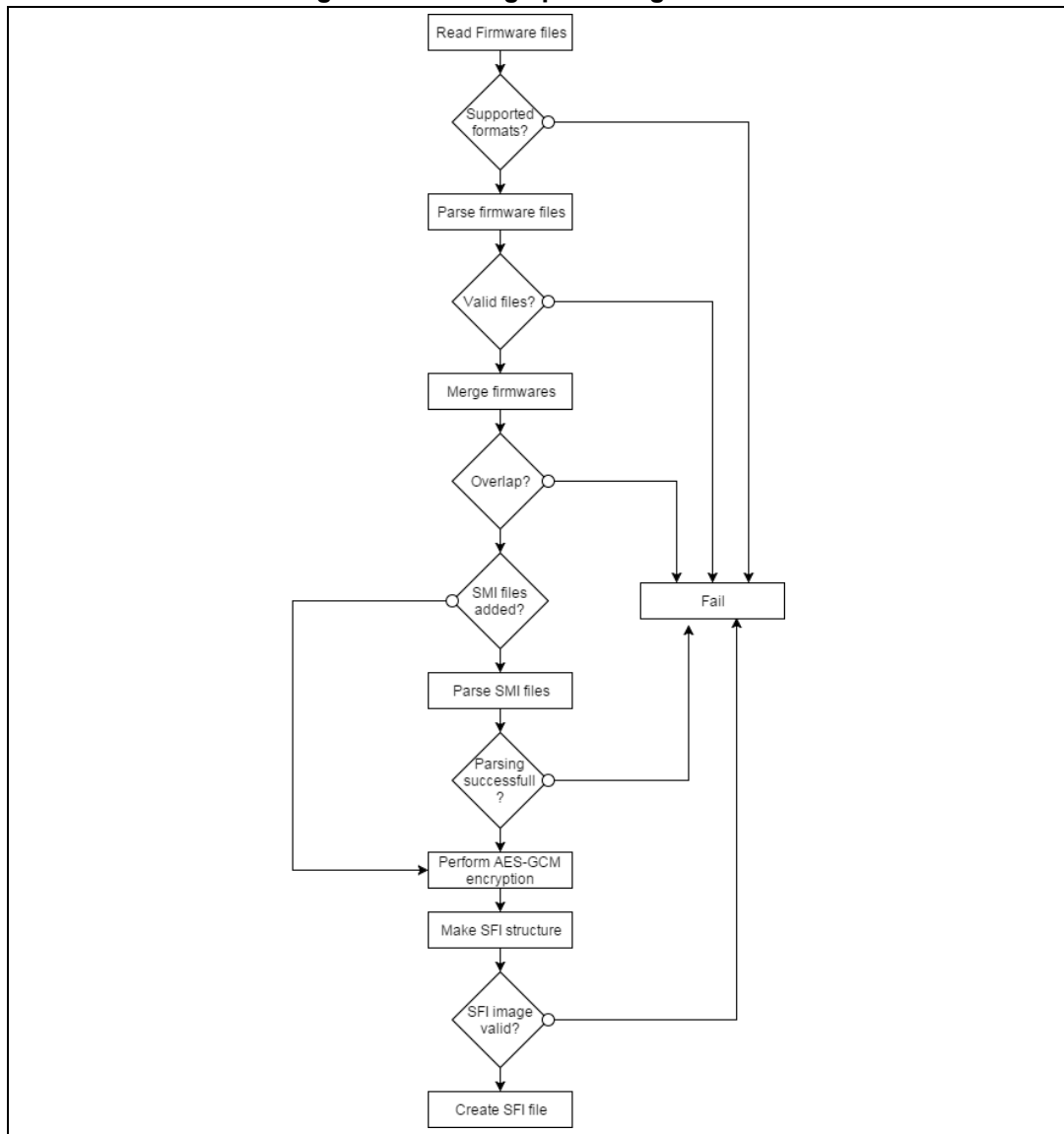
The SFI format is an encryption format for Firmware created by STMicroelectronics that transforms firmware (in Elf, Hex, Bin or Srec formats) into encrypted and authenticated firmware in SFI format using AES-GCM algorithm with a 128-bit key. The SFI preparation process used in the STM32TrustedPackageCreator 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:

$$IV = \text{nonce} + \text{Area Index}.$$

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 STM32TrustedPackageCreator overview below ([Figure 12](#)) 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**

STM32 Trusted Package Creator

File Edit Options Help

SFI SMI SFU HSM

**Firmware files**

tests.axf Add Remove

**Encryption key file**

M32TrustedPackageCreator/Input/SFI/good/test\_firmware\_key.bin Open

**Nonce file**

/projects/STM32TrustedPackageCreator/Input/SFI/good/nonce.bin Open

**Option bytes file**

projects/SFMI-PreparationToolv0.2.0\_test1/Input/SFI/good/job.csv Open

**SMI files (Only for combined case)**

STM32F4-DISCO.smi Add Remove

**Image version**

24

**RAM size**  **Continuation token address**

**Output SFI file**

C:/projects/STM32TrustedPackageCreator/output/out.sfi Select folder

**Firmware information** **SFI information**

**Overview**

File name

Size

Protocol version

**Segments**

Index	Type	Size	Address

Generate SFI

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

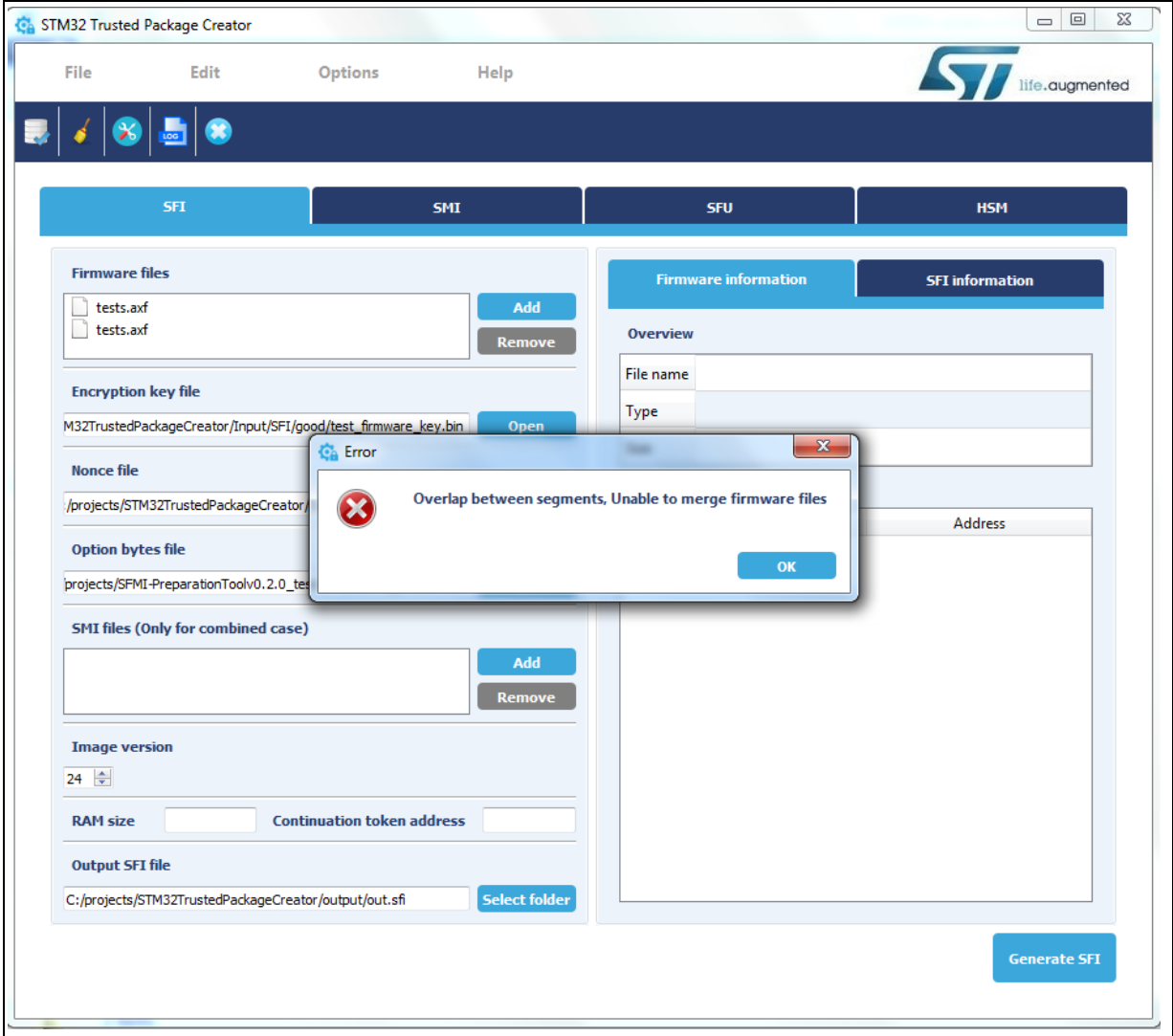
Area format	New Pause Area	New Resume Area
Type ('F', 'M', 'C')	Type 'P'	Type 'R'
Version	Version	Version
Index	Index	Index
Size	Size = 0	Size = 0
Address	Address of CT	Address of CT
Total Nb of areas	Total Nb of areas	Total Nb of areas
Tag	Tag	Tag
Encrypted Area Content <ul style="list-style-type: none"><li>- Firmware</li><li>- Module</li><li>- Configuration</li></ul>		

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.

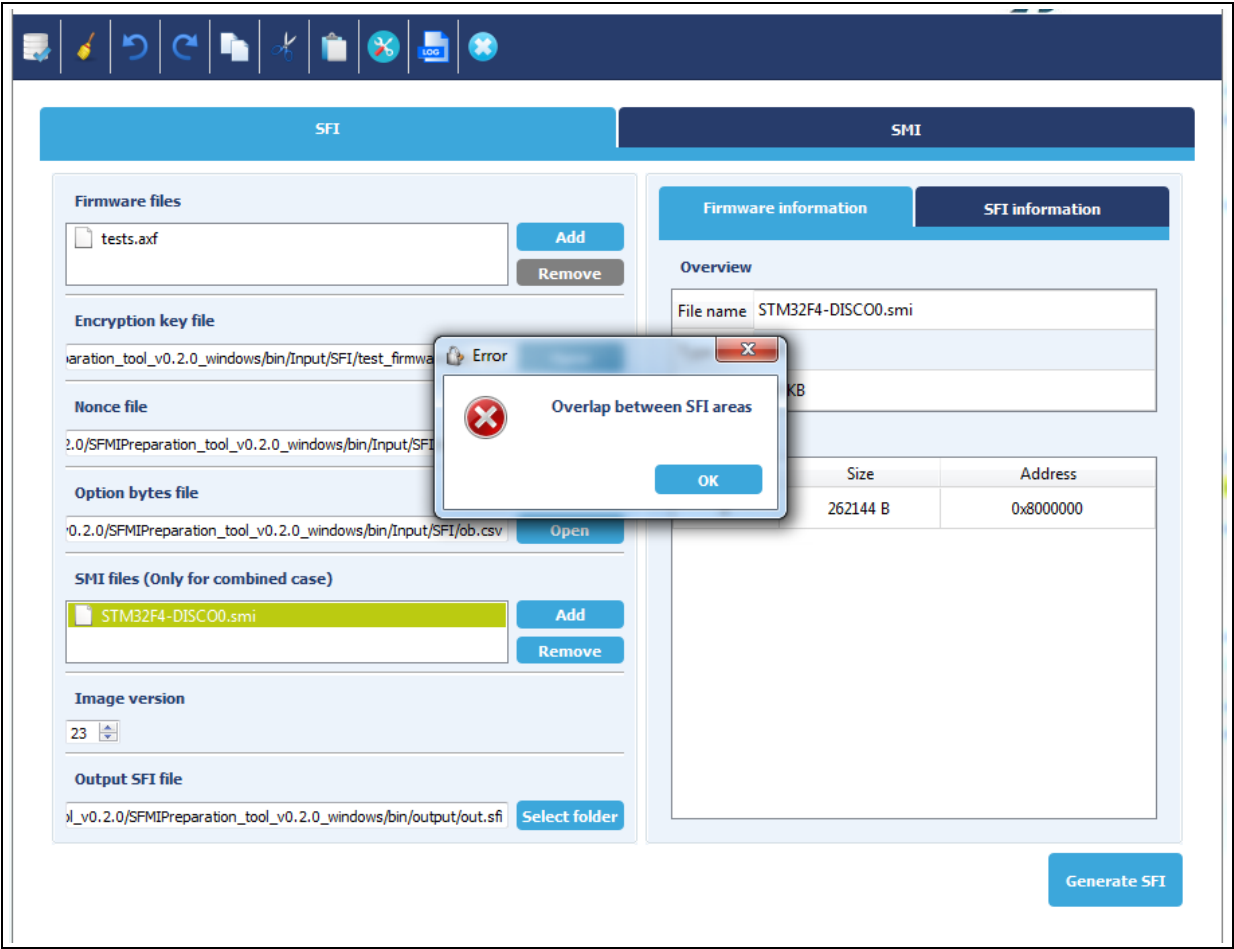
Note: To prepare an SFI image from multiple firmware files, make sure that there is no overlap between their segments, otherwise an error message appears (Figure 12).

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



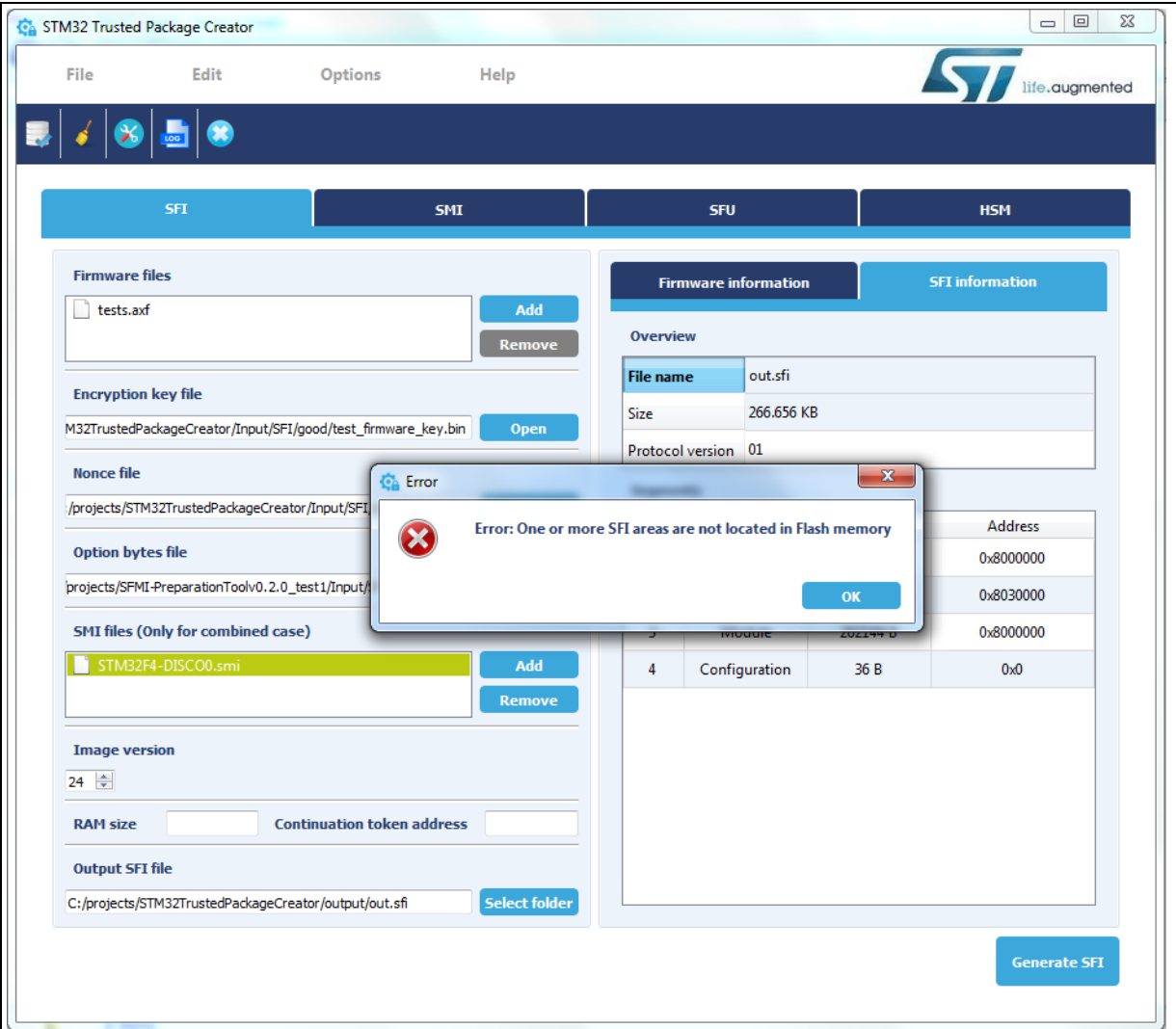
For combined SFI-SMI images, there is also an overlap check between firmware and module areas. If the check fails, an error message appears (Figure 13).

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



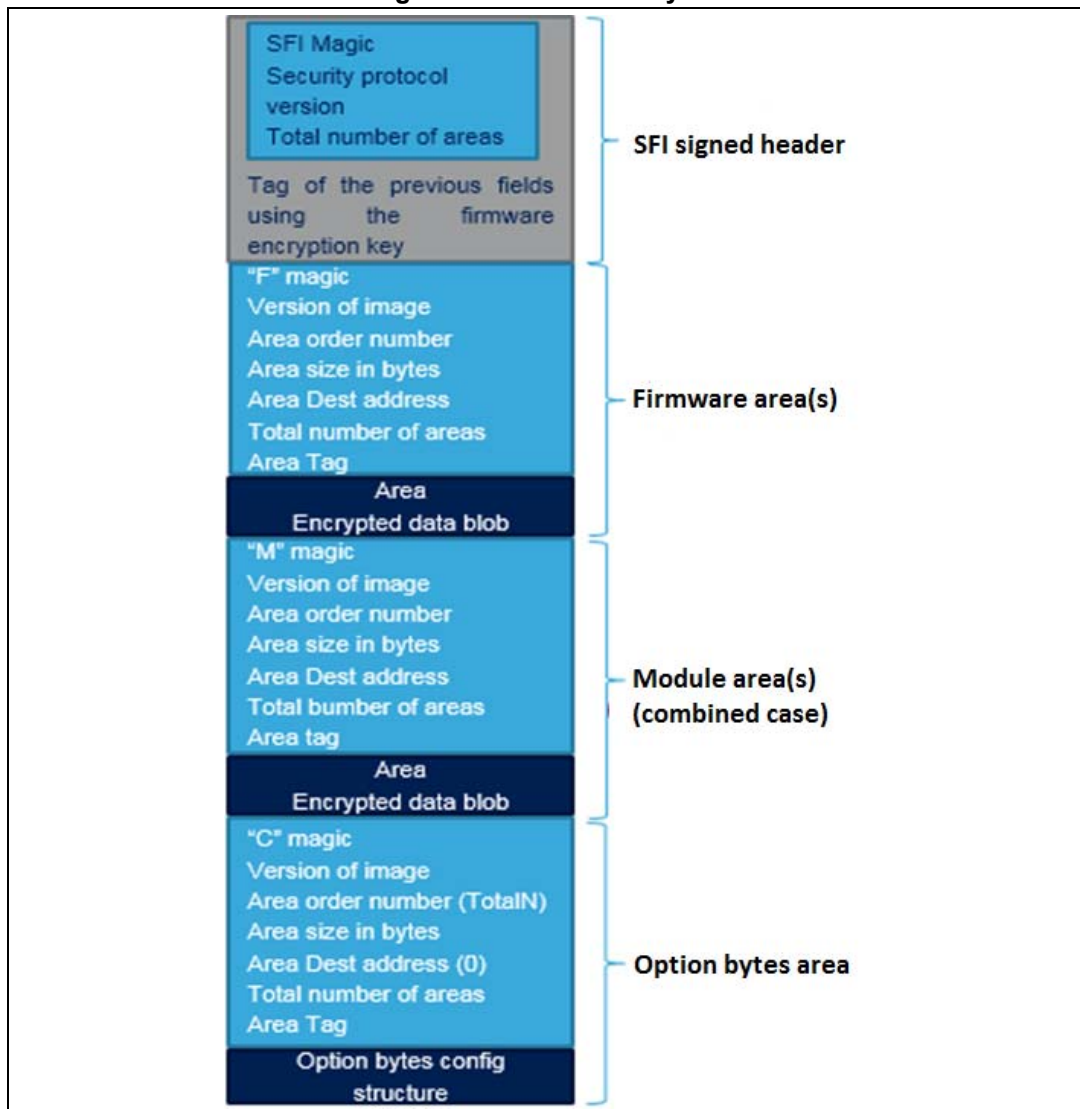
Also, all SFI areas must be located in Flash memory, otherwise the generation fails and the following error message appears (Figure 14).

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](#).

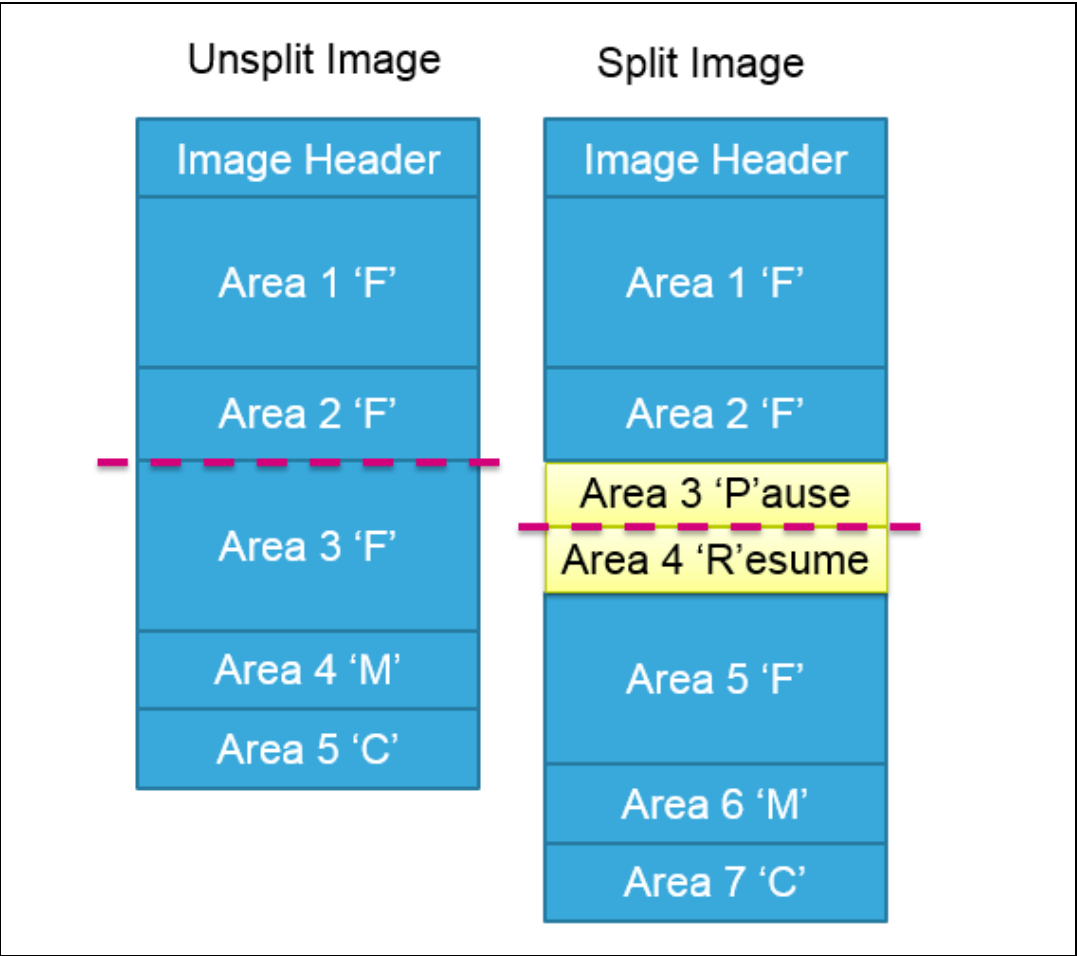
**Figure 15. SFI format layout**





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

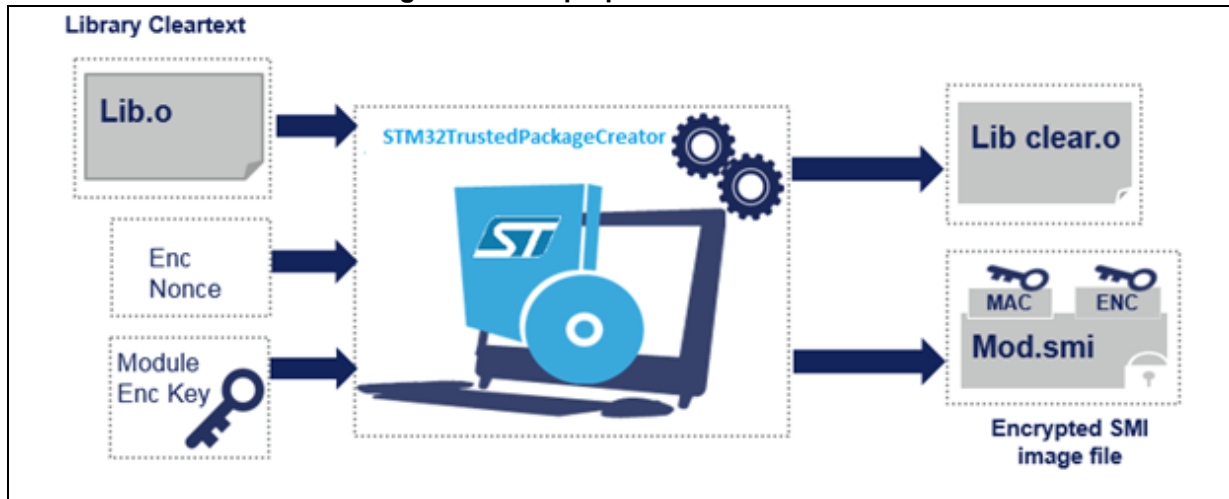


### 3.3 SMI generation process

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

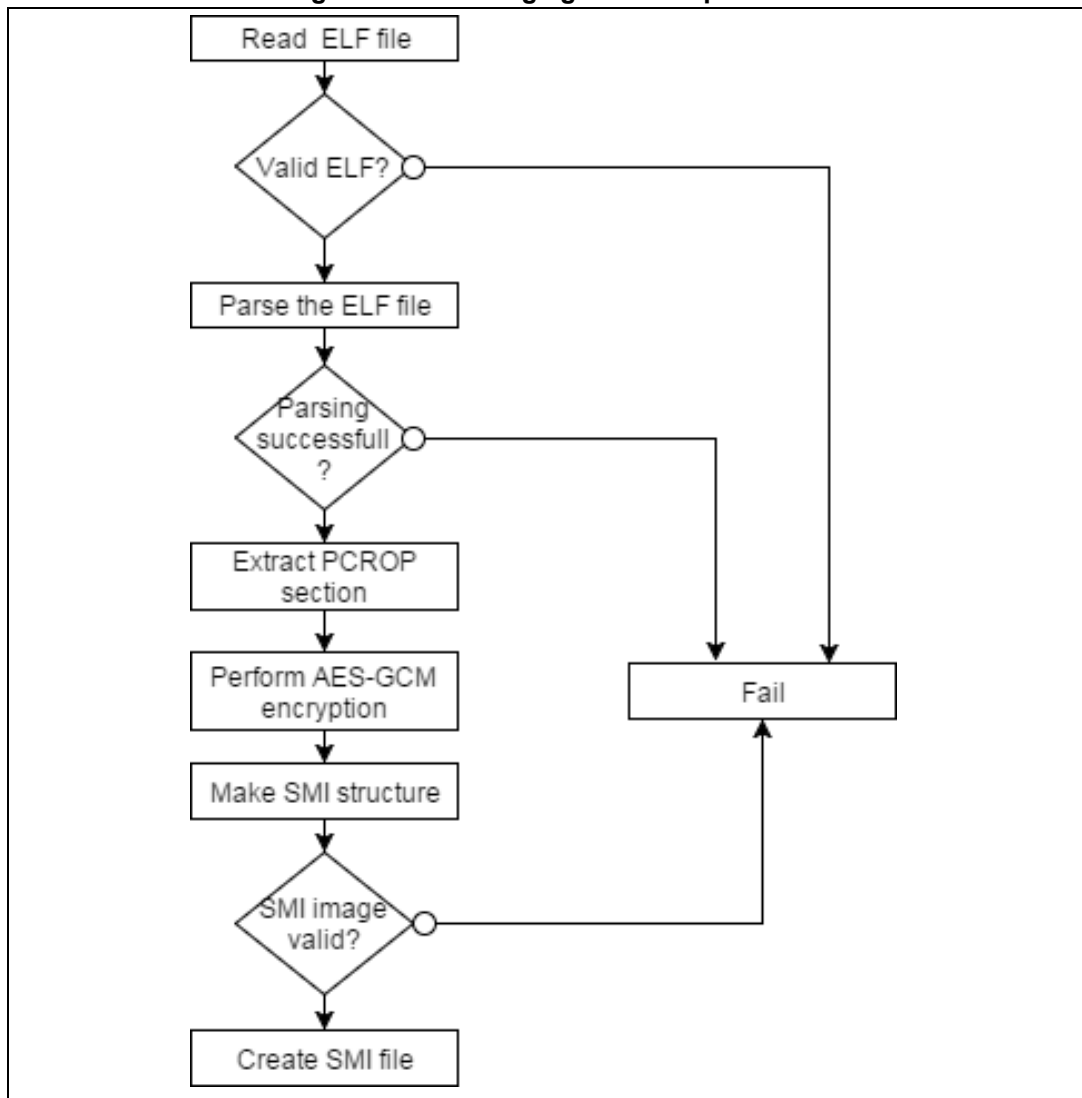
The SMI preparation process is described below ([Figure 17](#)).

**Figure 17. SMI preparation mechanism**



The SMI generation steps as currently implemented in the tool are described in the diagram below ([Figure 18](#)).

**Figure 18. 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).

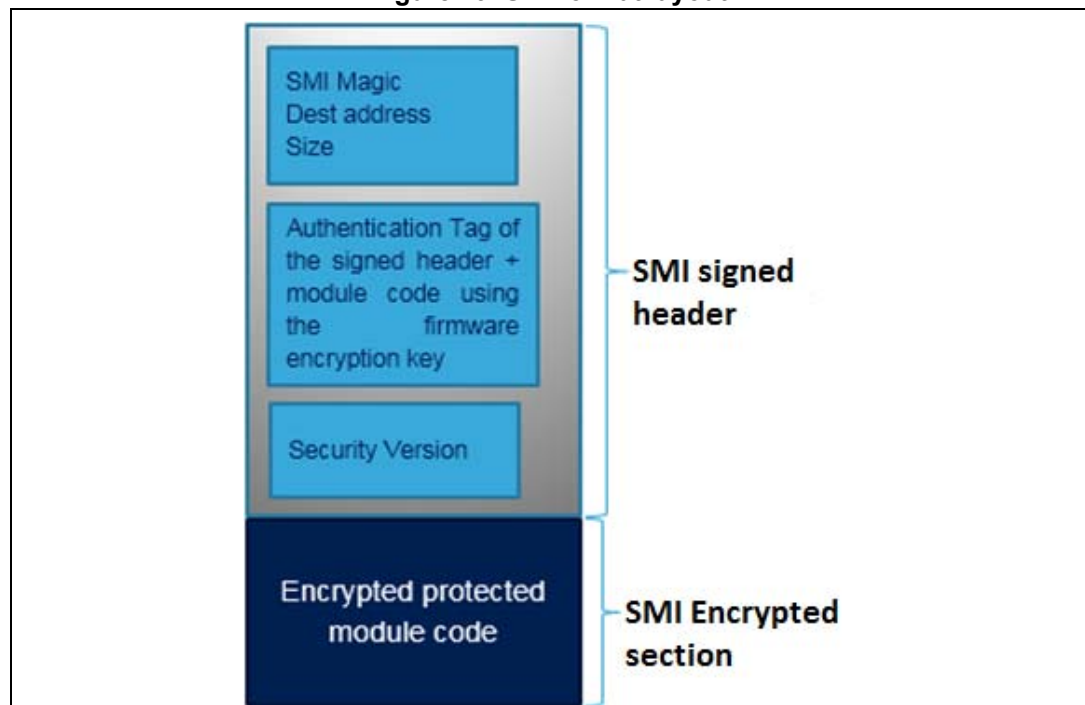
Before SMI image creation, PCROP checks are performed on the SMI image validity:

- 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, two outputs are created under the specified path:

- The SMI image ([Figure 19](#) represents the SMI format layout)
- The library data part.

**Figure 19. SMI format layout**



### 3.4 STM32TrustedPackageCreator tool in the command line interface

This section describes how to use the STM32TrustedPackageCreator tool from the command line interface in order to generate SFI and SMI images.

The available commands are listed in [Figure 20](#).

Figure 20. STM32TrustedPackageCreator 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
  [<Address>]         : Only in case of BIN input file (in any base)
-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
-v, --ver             : Image version
  <Image_Version>     : Its value must be in <0..255> (in any base)
-ob, --obfile         : Option bytes configuration file
  <CSU_File>          : CSU 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, --elffile       : 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

```

### 3.4.1 Steps for SFI generation (CLI)

In order to generate an SFI 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 firmwares.

#### **-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.

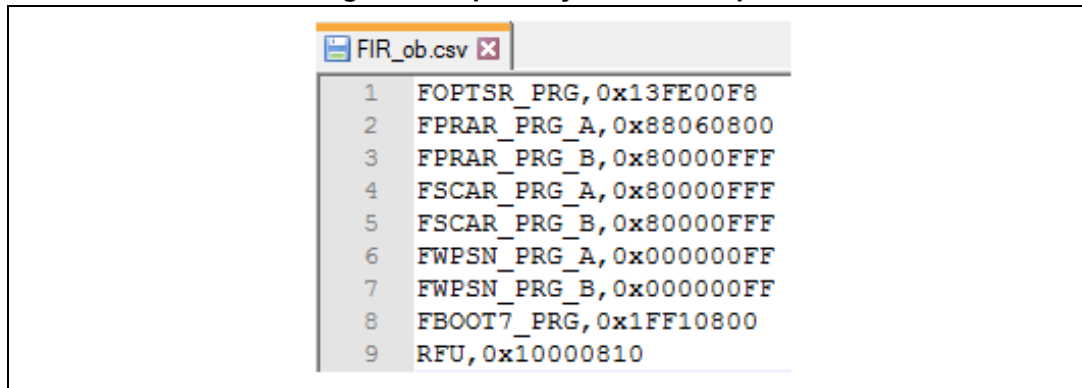
Only CSV (Comma Separated Value) file format is supported as input for this field, it is composed from two vectors: register name and register value respectively.

Example: for STM32H7xx devices, 9 option bytes registers must be configured, which corresponds to a total of 9 lines in the csv file ([Figure 21](#)).

**Syntax:** -ob <CSV\_file>

<CSV\_file>: A csv file with 9 values.

Figure 21. Option bytes file example



1	FOPTSR_PRG	0x13FE00F8
2	FPRAR_PRG_A	0x88060800
3	FPRAR_PRG_B	0x80000FFF
4	FSCAR_PRG_A	0x80000FFF
5	FSCAR_PRG_B	0x80000FFF
6	FWPSN_PRG_A	0x000000FF
7	FWPSN_PRG_B	0x000000FF
8	FBOOT7_PRG	0x1FF10800
9	RFU	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

**-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 22](#).

Figure 22. SFI generation example using an Elf file



```
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin>STM32Trust
edPackageCreator_CLI.exe -sfi -fir tests.axf -k test_firmware_key.bin -n nonce.b
in -ob ob.csv -v 23 -o out.sfi
SFI generation SUCCESS
```

### 3.4.2 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, --elffile**

**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>

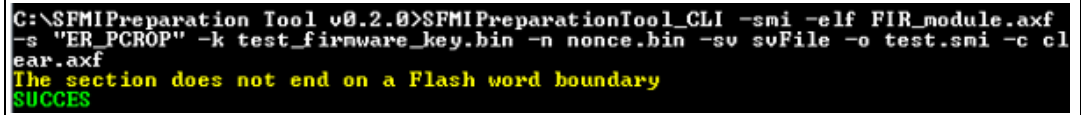


<ELF\_file> : Clear ELF file to be generated.

Example of 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 23. SMI generation example**



```
C:\SFMIPreparation Tool v0.2.0>SFMIPreparationTool_CLI -smi -elf FIR_module.axf -s "ER_PCROP" -k test_firmware_key.bin -n nonce.bin -sv svFile -o test.smi -c clear.smi
The section does not end on a Flash word boundary
SUCCESS
```

### 3.5 Using the STM32TrustedPackageCreator tool graphical user interface

The STPC is also available in graphical mode, this section describes its use. The STM32TrustedPackageCreator tool GUI presents two tabs, one for SFI generation and one for SMI generation.

#### 3.5.1 SFI generation using STPC in GUI mode

*Figure 23* shows the graphical user interface tab corresponding to SFI generation.

**Figure 24. SFI generation Tab**

The screenshot displays the STM32 Trusted Package Creator GUI. The 'SFI' tab is selected, showing the following fields and options:

- Firmware files:** A list containing 'tests.axf' with 'Add' and 'Remove' buttons.
- Encryption key file:** A text field with the path 'M32TrustedPackageCreator/Input/SFI/good/test\_firmware\_key.bin' and an 'Open' button.
- Nonce file:** A text field with the path '/projects/STM32TrustedPackageCreator/Input/SFI/good/nonce.bin' and an 'Open' button.
- Option bytes file:** A text field with the path 'projects/SFMI-PreparationToolv0.2.0\_test1/Input/SFI/good/ob.csv' and an 'Open' button.
- SMI files (Only for combined case):** A text field with 'STM32F4-DISCO0.smi' and 'Add'/'Remove' buttons.
- Image version:** A dropdown menu set to '24'.
- RAM size:** An empty text field.
- Continuation token address:** An empty text field.
- Output SFI file:** A text field with the path 'C:/projects/STM32TrustedPackageCreator/output/out.sfi' and a 'Select folder' button.

On the right side, the 'Firmware information' tab is active, showing an 'Overview' section with the following details:

- File name:** tests.axf
- Type:** ELF
- Size:** 815.887 KB

Below the overview is a 'Segments' table:

Index	Size	Address
1	844 B	0x8000000
2	9884 B	0x8030000

A 'Generate SFI' button is located at the bottom right of the interface.

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

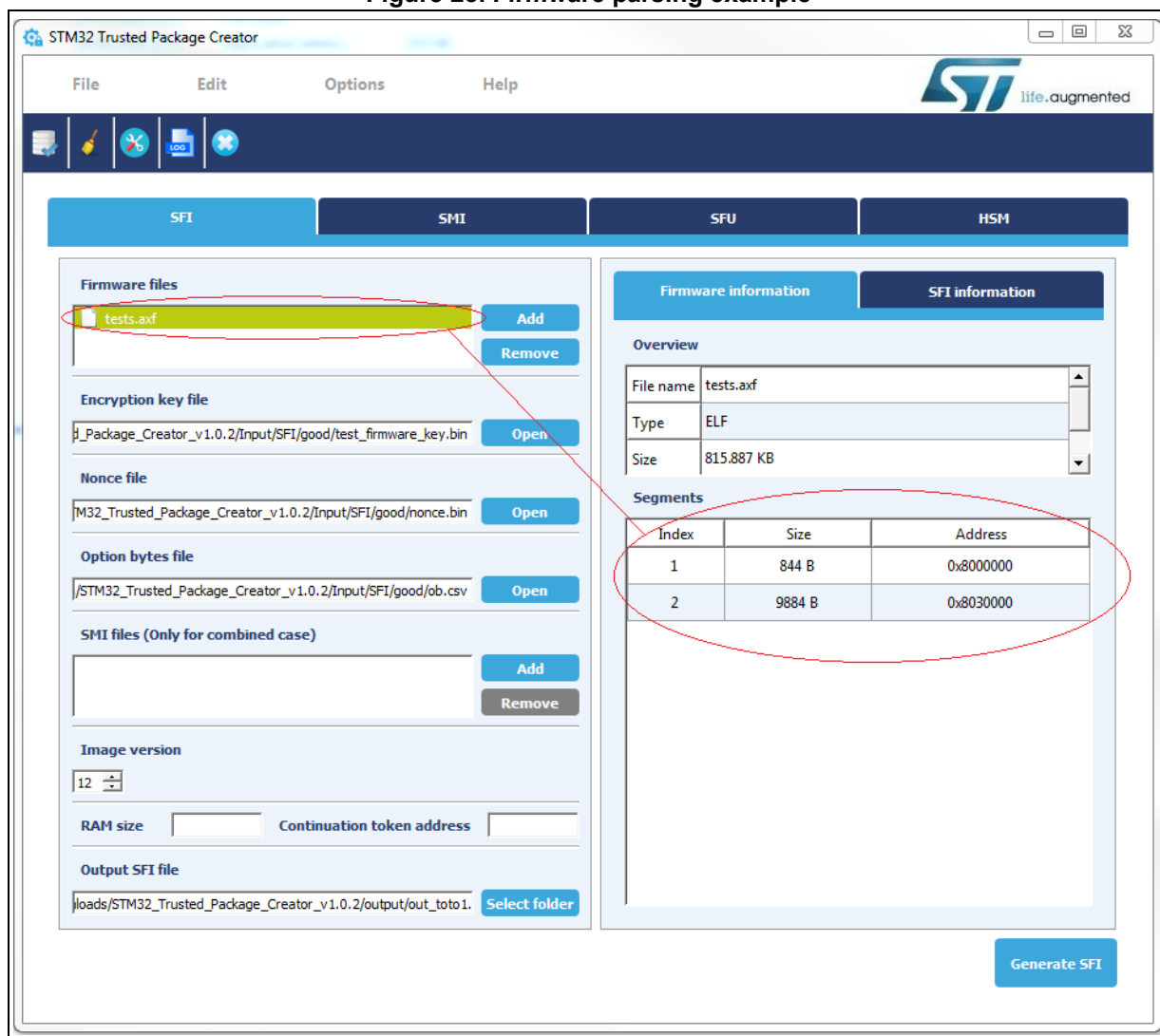
- Firmwares files:

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 25](#)).

**Figure 25. Firmware parsing example**

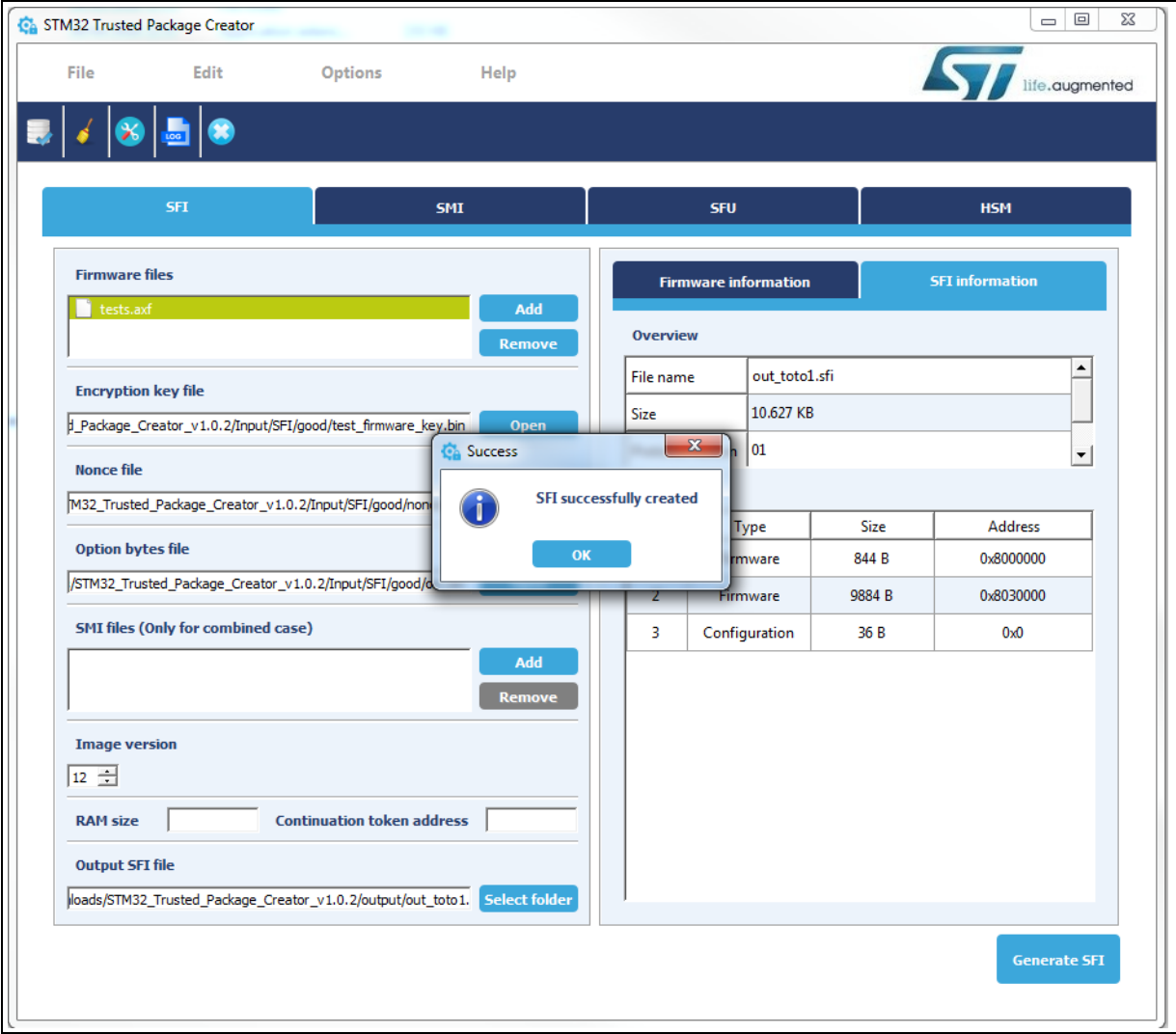


- Encryption key and nonce file:  
The encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selecting them with the “Open” button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).
- Option bytes file :  
The option bytes file can be selected the same way as the encryption key and nonce. Only csv files are supported.
- SMI files:  
SMI files can be 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 can be 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 26](#)) and information about the generated SFI file is displayed in the SFI information section.

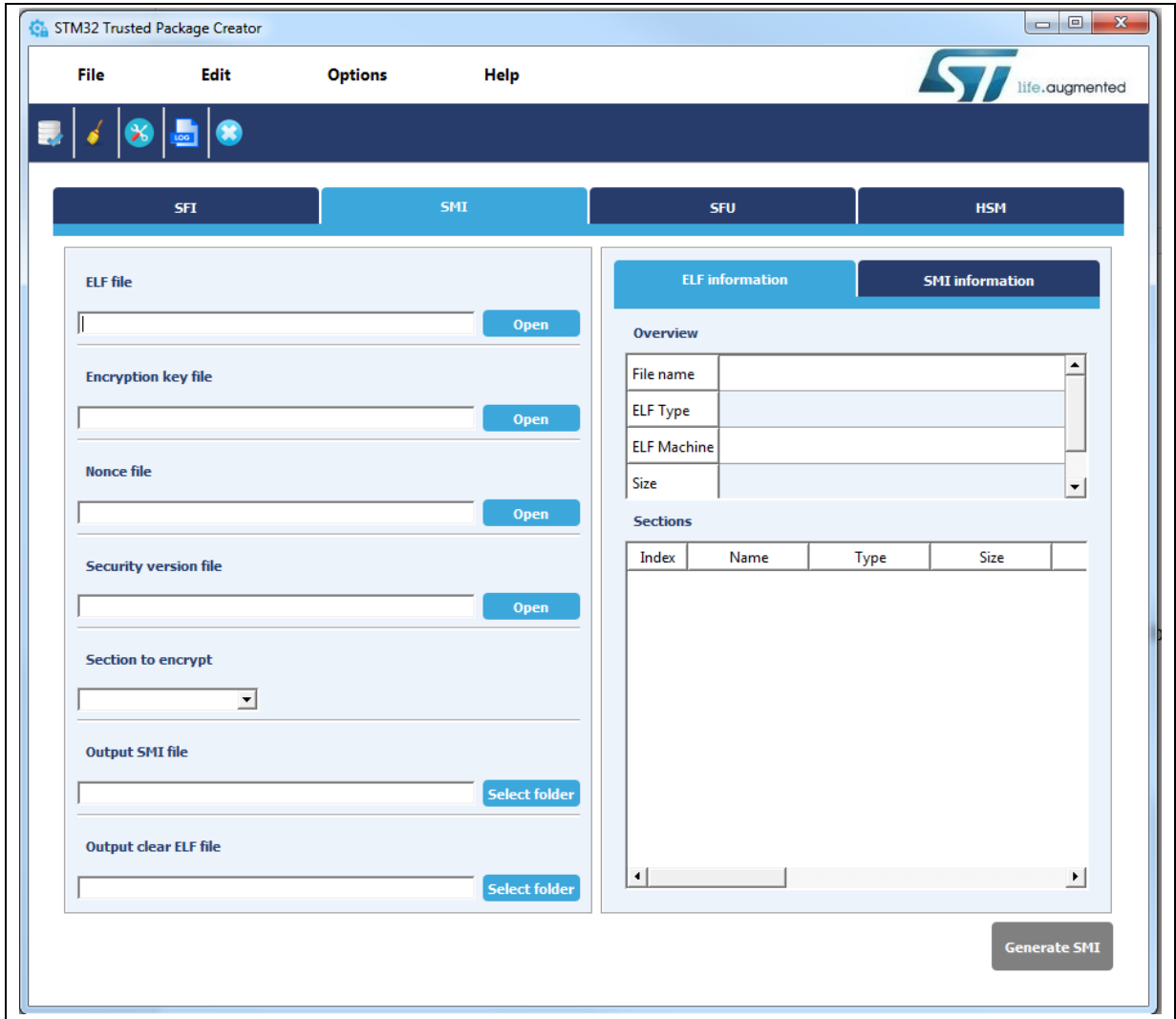
Figure 26. SFI successful generation in GUI mode example



3.5.2 SMI generation using STPC in GUI mode

Figure 27 shows the graphical user interface tab corresponding to SMI generation.

Figure 27. SMI generation Tab



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

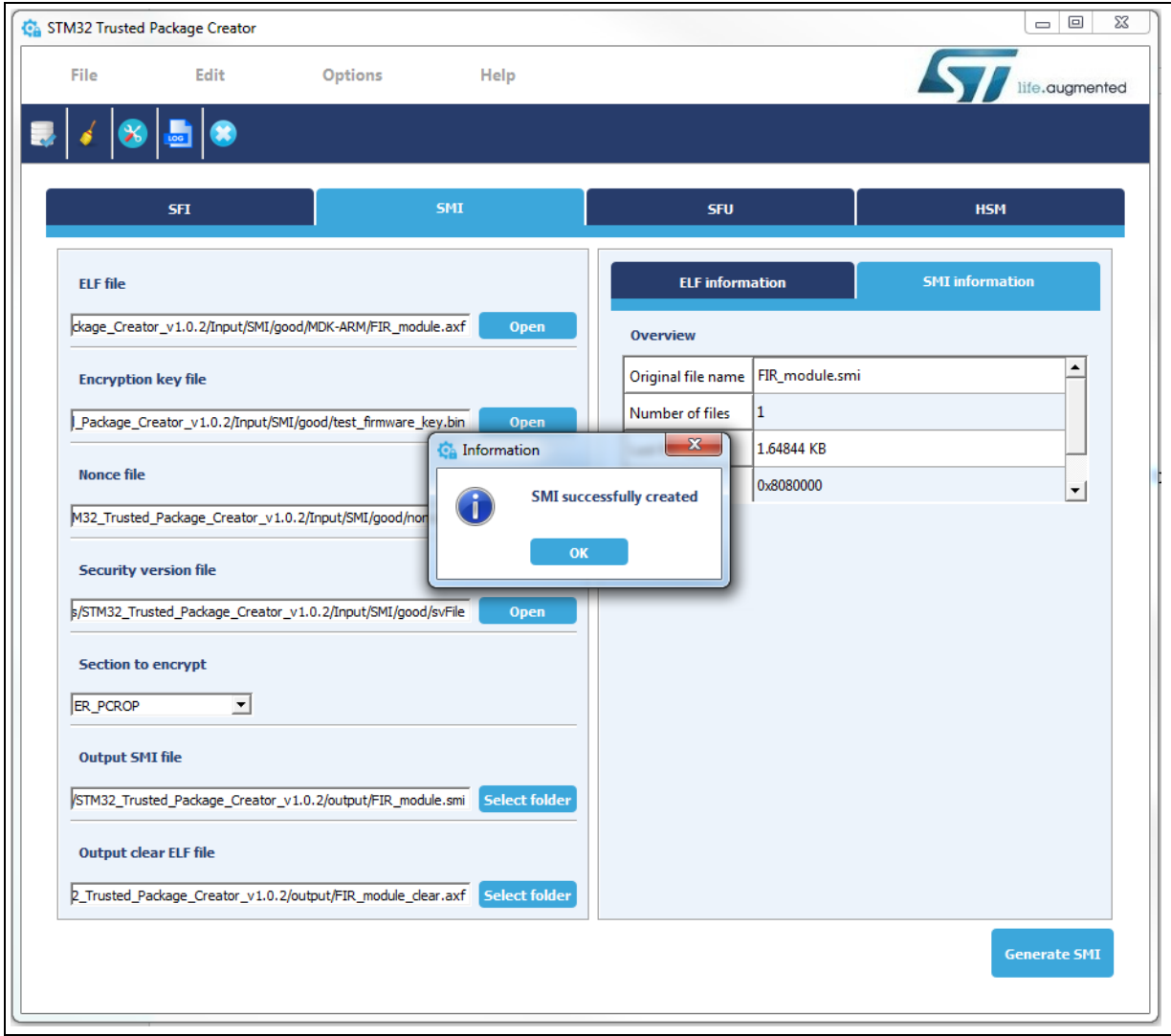
### SMI GUI tab fields

- Elf file:  
In this case the input file can be only 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 can be 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 can be 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 28](#)), with additional information about the generated SMI file displayed into the “SMI information” section. In the case of any invalid input data, an error message box appears instead.

Figure 28. SMI successful generation in GUI mode example

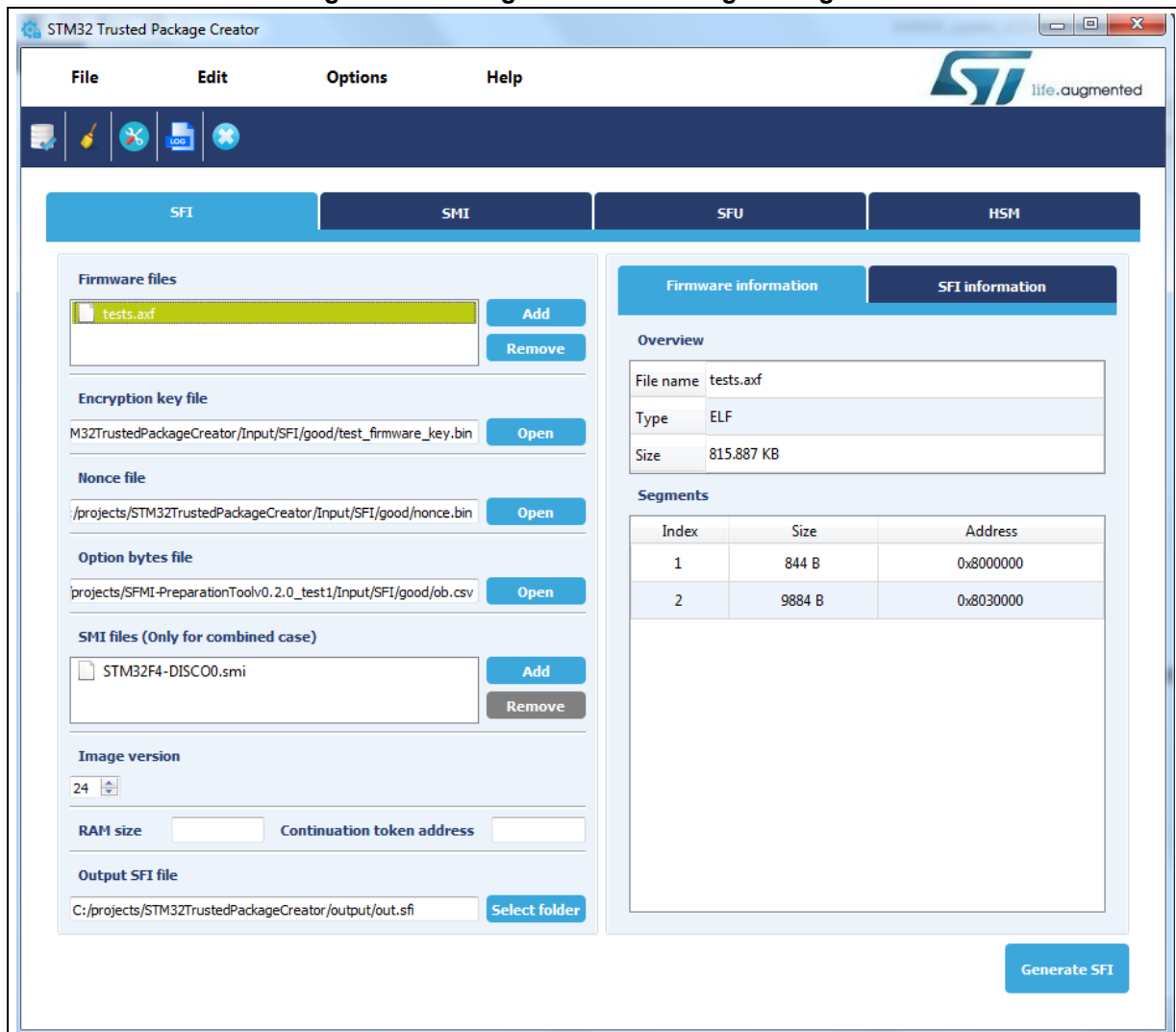


### 3.5.3 Settings

The STPC allows generation to be performed respecting some user-defined settings. The settings dialog can be displayed by clicking the settings icon (see [Figure 29](#)) in the tool bar or in the menu bar by choosing: Options -> settings.



Figure 29. Settings icon and Settings dialog box



Settings can be performed on:

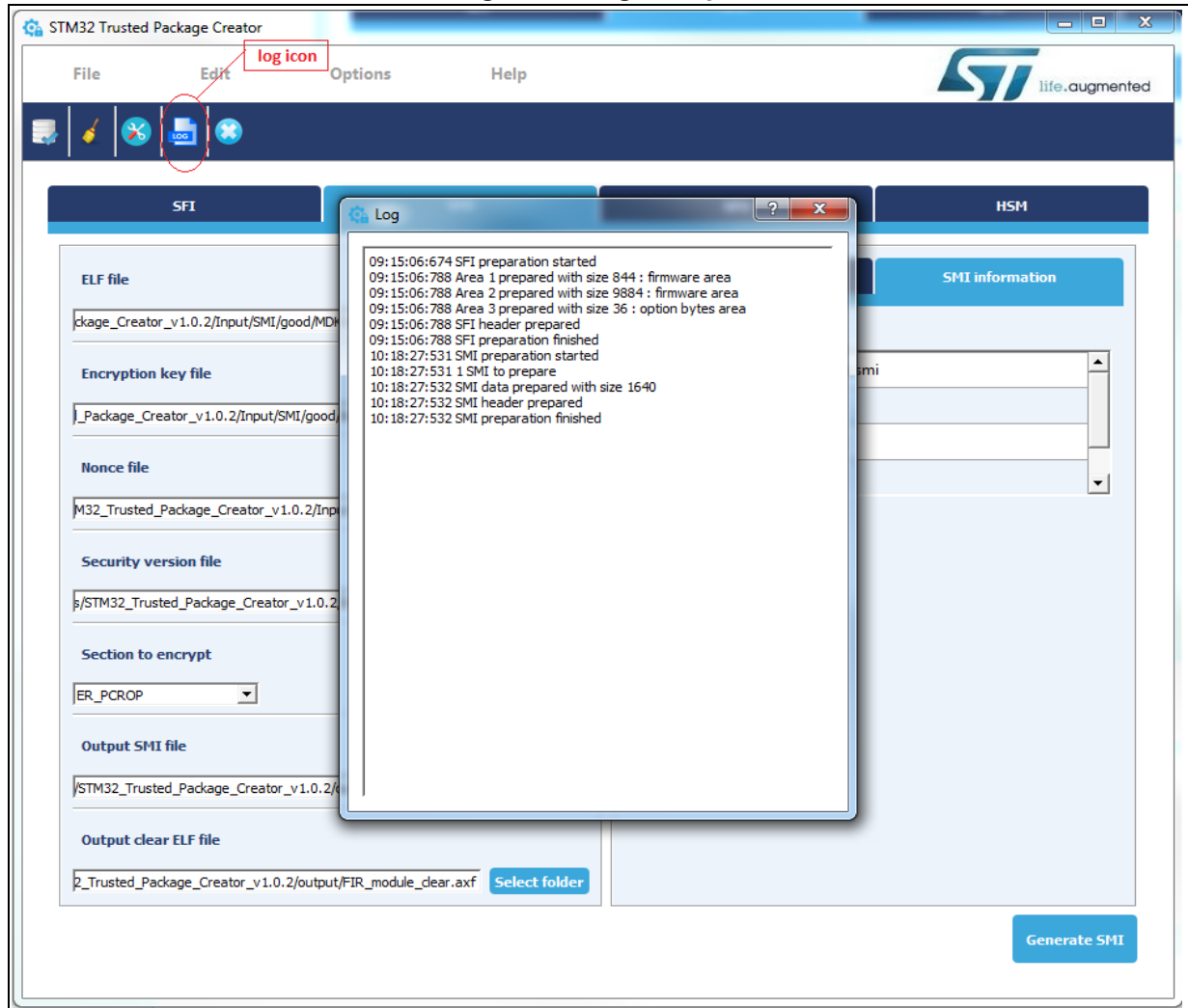
- **Padding byte:**  
When parsing Hex and Srec files, padding can be added to fill gaps between close segments in order to merge them and reduce the number of segments. The user might choose to perform padding either with 0xFF (default value) or 0x00.
- **Settings file:**  
When checked, a "settings.ini"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

### 3.5.4 Log generation

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

*Figure 30* shows a log example:

**Figure 30. Log example**



### 3.5.5 SFI and SMI file checking function

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

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

*Figure 31* shows a check SFI example:

**Figure 31. Check SFI file example**

The screenshot displays the STM32 Trusted Package Creator application window. The title bar reads "STM32 Trusted Package Creator". The menu bar includes "File", "Edit", "Options", and "Help". The toolbar contains icons for file operations and a "LOG" button. Below the menu bar, there are four tabs: "SFI", "SMI", "SFU", and "HSM". The "SFI" tab is currently selected. The main workspace is divided into two panels. The left panel, titled "Firmware identifier", contains the following fields and controls:

- Firmware identifier:** A text input field.
- Encryption key file:** A text input field with an "Open" button to its right.
- Nonce file:** A text input field with an "Open" button to its right.
- Maximum counter:** A numeric input field with a spinner, currently showing "0".
- ☐ **Set HSM to operational state (HSM will be locked)**

The right panel, titled "HSM information", contains a table with the following data:

HSM information	
Firmware ID	
Max counter	
HSM status	

Below the table is a "Refresh" button. At the bottom right of the window is a "Program HSM" button.

## 4 Encrypted firmware (SFI) / module (SMI) programming using 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 ST-LINK probe.

The STM32CubeProgrammer tool currently also supports secure programming of SFI and SMI images using UART, USB, SPI, JTAG/SWD interfaces.

The tool is currently available only in CLI mode, it is available free of charge from [www.st.com](http://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 count each install for a given piece of firmware, since each license is specific to a unique chip, identified by its public key.

##### Licenses 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.

For ST we offer an SFI solution based on SmartCards HSM as a license distribution tool for use in programming houses.

### HSM programming by OEM for License distribution

When an OEM needs to deliver 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 first be performed.

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 each piece of data to be programmed in the HSM.

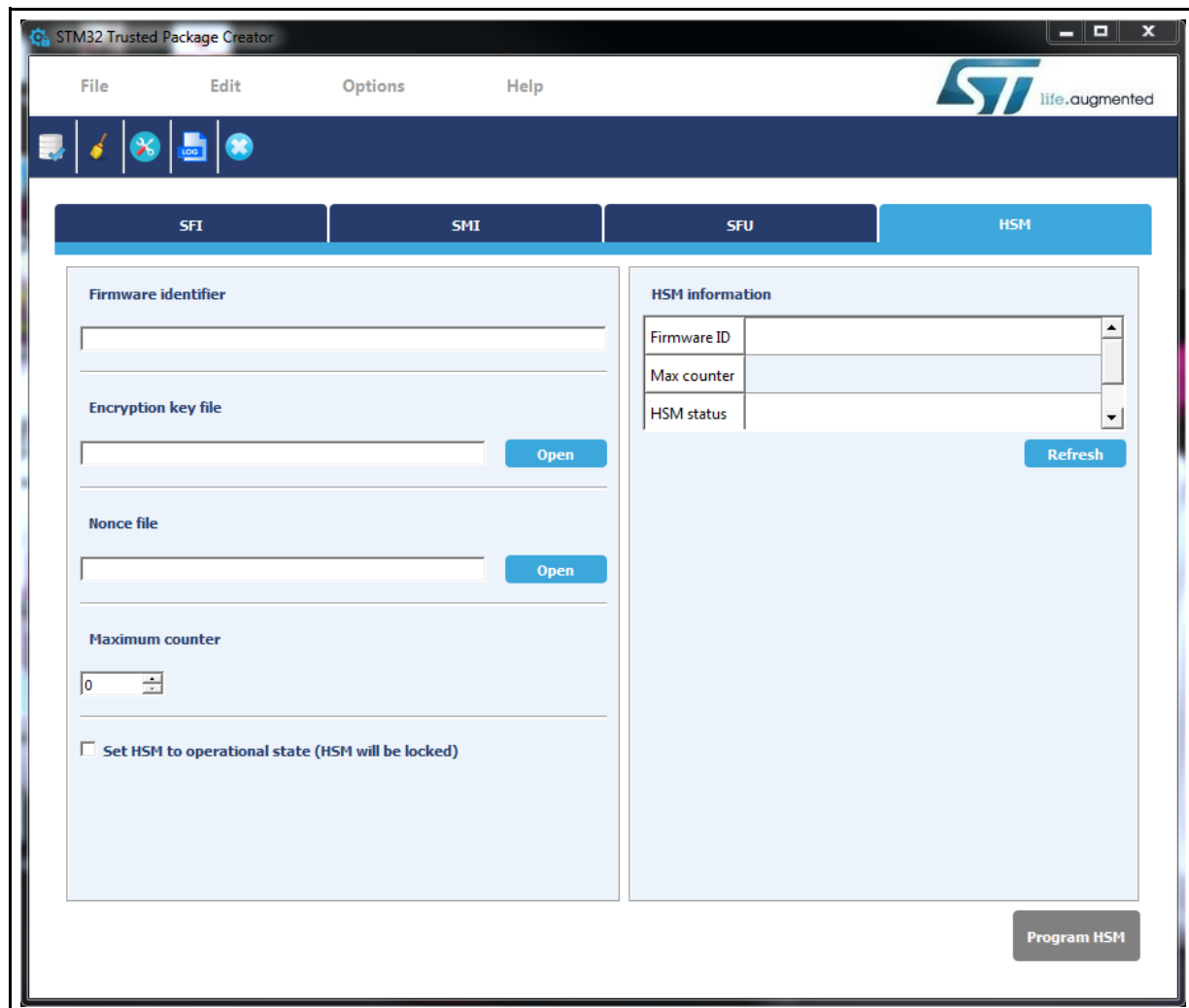
These data are:

- **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 is 16 Ku for HSM version 1.0.
- **The firmware key:** this key is 32 bytes and is composed of two fields, the initialization vector (IV) (first field) and the key (last field) that were used to AES128-GCM encrypt the firmware.  
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 data:** these 108 data bytes are encrypted before being sent to the chip, where each device has its own configuration. It is only used for HSM version 1.

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

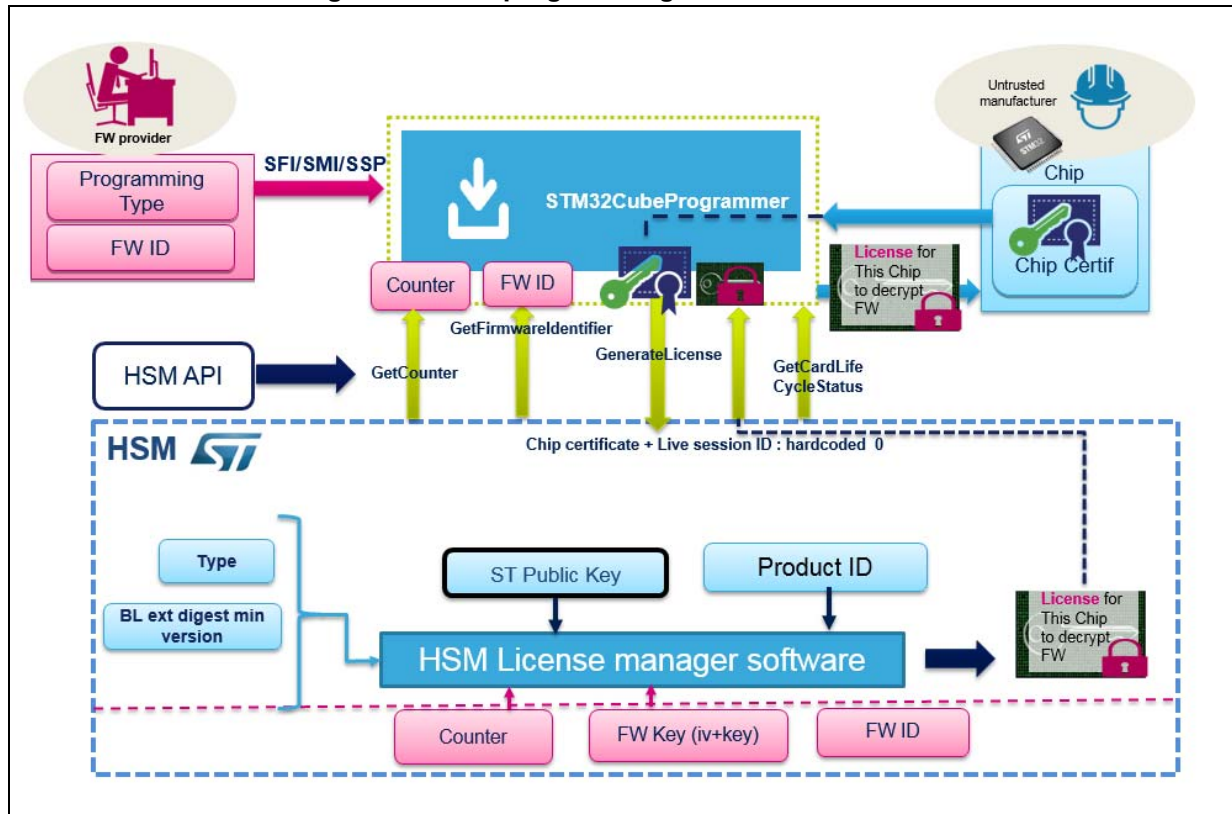
ST provides the tools needed to support SFI via HSM. In fact, HSM programming is supported by the STM32TrustedPackageCreator tool. [Figure 32](#) shows the GUI for HSM programming in STPC tool.

**Figure 32. HSM programming toolchain**



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. This process is illustrated in [Figure 33](#).

Figure 33. HSM programming GUI in the STPC tool



For further details, refer to the SFI-HSM specification document and the SFI-HSM user manual (UM2428) [2].

## 4.2 Secure programming using bootloader interface

### 4.2.1 Secure firmware installation using 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 STM32H753xl and STM32L451CE specific part number so far.

For STM32H753xl devices, one important outcome is that RSS fully manages the installation (no secure bootloader) and SFI is supported for USART, SPI and USB interfaces for those devices. Otherwise for STM32L451CE specific part number devices the installation is performed through a secure bootloader via USART or SPI interfaces only.

For more details on SFI over these STM32 devices refer to AN4992 [1]. This document is available on [www.st.com](http://www.st.com).

The general flow of the Secure Firmware Installation using bootloader interface on a chip for H7 and L4 secure devices is shown respectively in [Figure 33](#) and [Figure 34](#) below.

Figure 34. Secure programming via STM32CubeProgrammer overview on STM32H7 devices

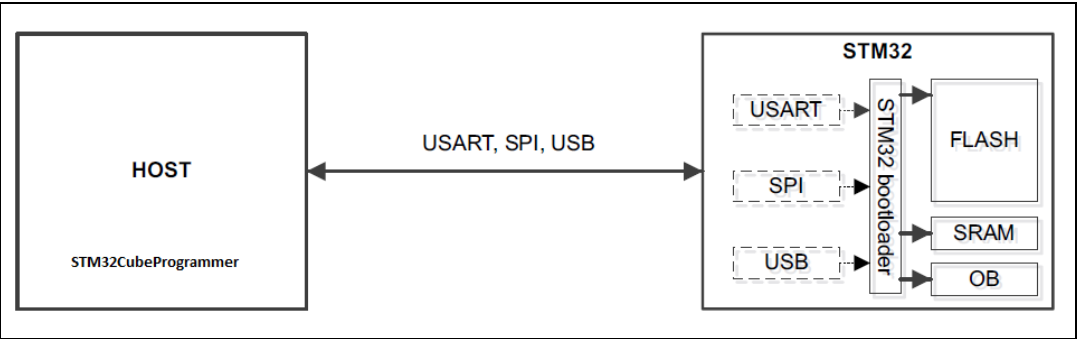
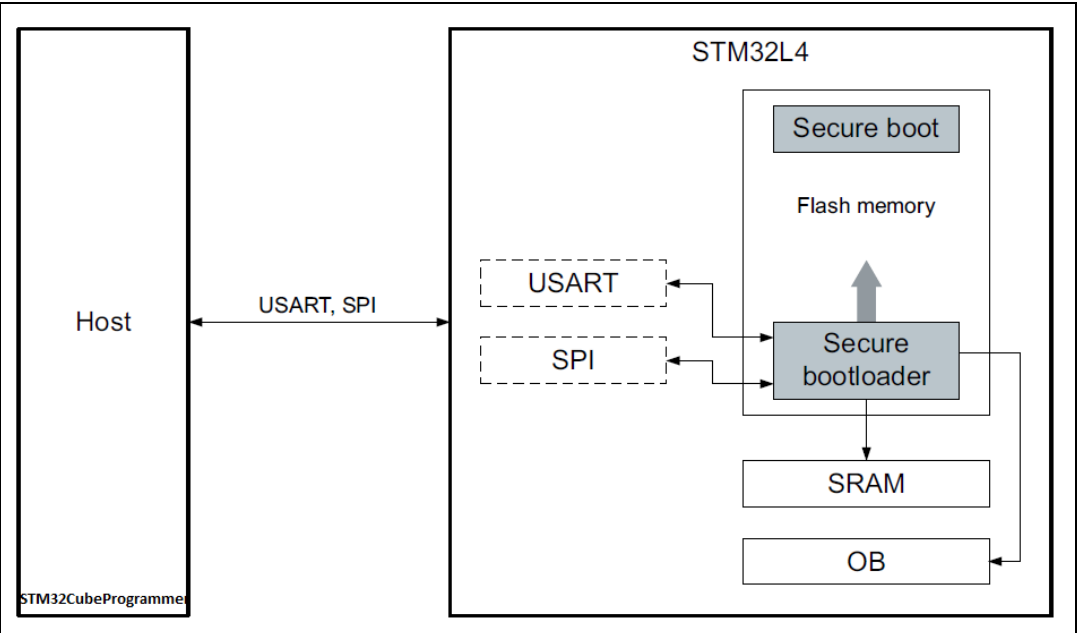


Figure 35. Secure programming via STM32CubeProgrammer overview on STM32L4 devices





## 4.2.2 Secure Module installation using bootloader interface flow

As explained in [Section 3.3: 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 can be 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 `RSS_SMI_resetAndInstallModules()` function is then invoked through the `start_smi()` secure bootloader command with the following parameters:

- Pointer to the license
- Pointer to the content of the SMI image file.

This causes a reset and the decryption, authentication and install of the protected module code into a properly setup Pc-ROP area

*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 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 currently supported).

`<file_path>` : Path of SFI file to be programmed.

`<licenseFile_path>`: Path to the license file of the smi to be programmed.

`[<licenseMod_path>]` : Path to the license files of the integrated SMI module(s), (used only when the SFI image is composed of one or many SMI modules areas).

Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 br=115200 -sfi protocol=static  
"C:\SFI\data.sfi" "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 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)

<file\_path>: Path of SMI file to be programmed.

[<address>]: Destination address of the SMI module (only needed when relocatable).

<licenseFile\_path>: Path to the license file of the SMI to be programmed.

Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 br=115200 -smi protocol=static
"C:\SMI\data.smi" 0x08080000 "C:\SMI\license.bin"
```

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

#### 4.2.5 STM32CubeProgrammer for get certificate via 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 36](#).

**Figure 36. Example of getcertificate command execution using UART interface**

```
Requesting Chip Certificate from connected device...
Serial Port COM1 is successfully opened.
Activating device: OK
Port configuration: parity = none, baudrate = 115200, data-bit = 8,
                  stop-bit = 1.000000, flow-control = off
Chip ID: 0x450
BootLoader version: 3.1
Certificate File      : C:\Demo_certificate.bin
File already exist. It will be overwritten.
Get Certificate done successfully
...Writing data to file C:\Demo_certificate.bin
writing chip certificate to file C:\Demo_certificate.bin finished successfully
```

## **4.3       Secure programming using JTAG/SWD interface**

### **4.3.1    SFI programming using JTAG/SWD flow**

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

The whole process happens in RDP level 0. The code in Flash memory is protected from the debugger by the PCROP mechanism. The whole user Flash memory is PCROPed during SFI.

One important outcome is that RSS fully manages the installation. This means that the whole SFI image and the license must be transferred to RAM before starting. The SFI image header and areas can be written to different locations.

SFI via debug interface is currently supported only for STM32H753I 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.

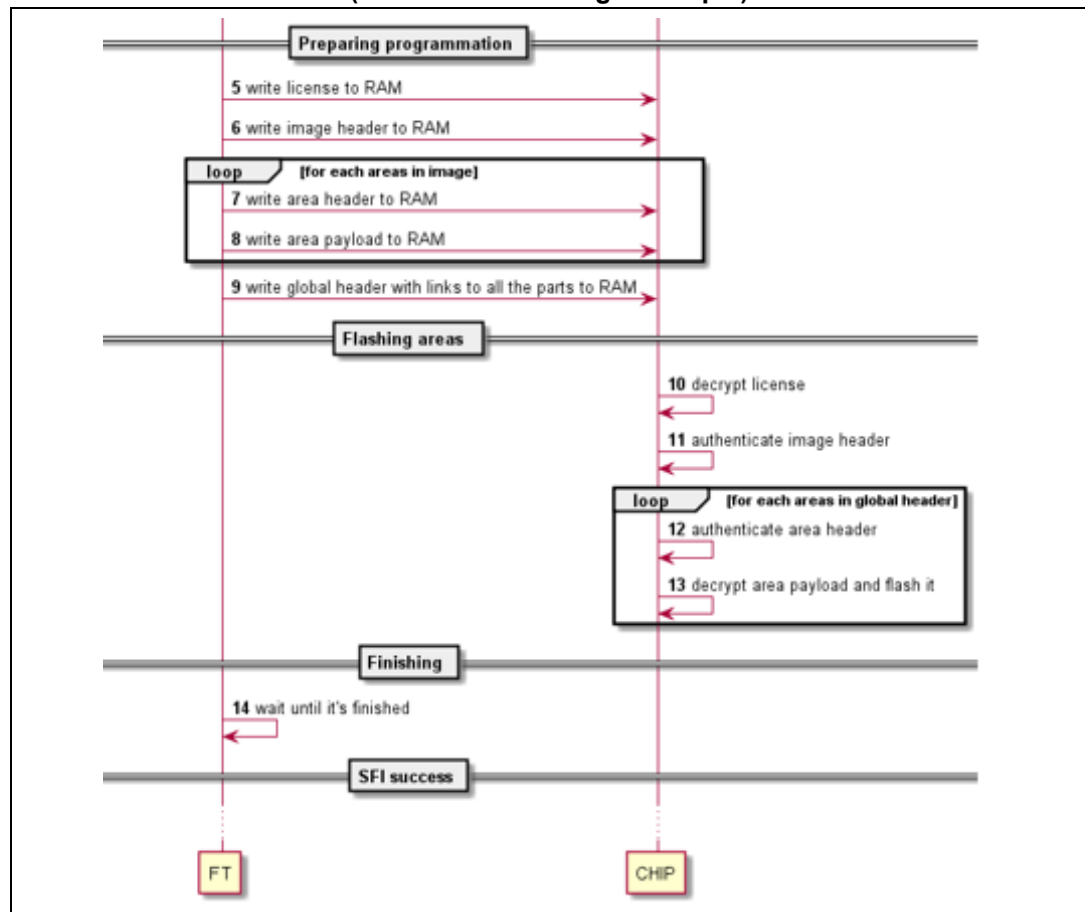
To remedy this, we resort to splitting the SFI image into several parts, so that each part fits into the allowed RAM size.

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

Other limitations are that security must be left activated in the configuration area if there is a PCROP area.

The SFI flow for programming through JTAG is described in [Figure 37](#).

**Figure 37. 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. In fact, RSS fully manages the installation in both cases.

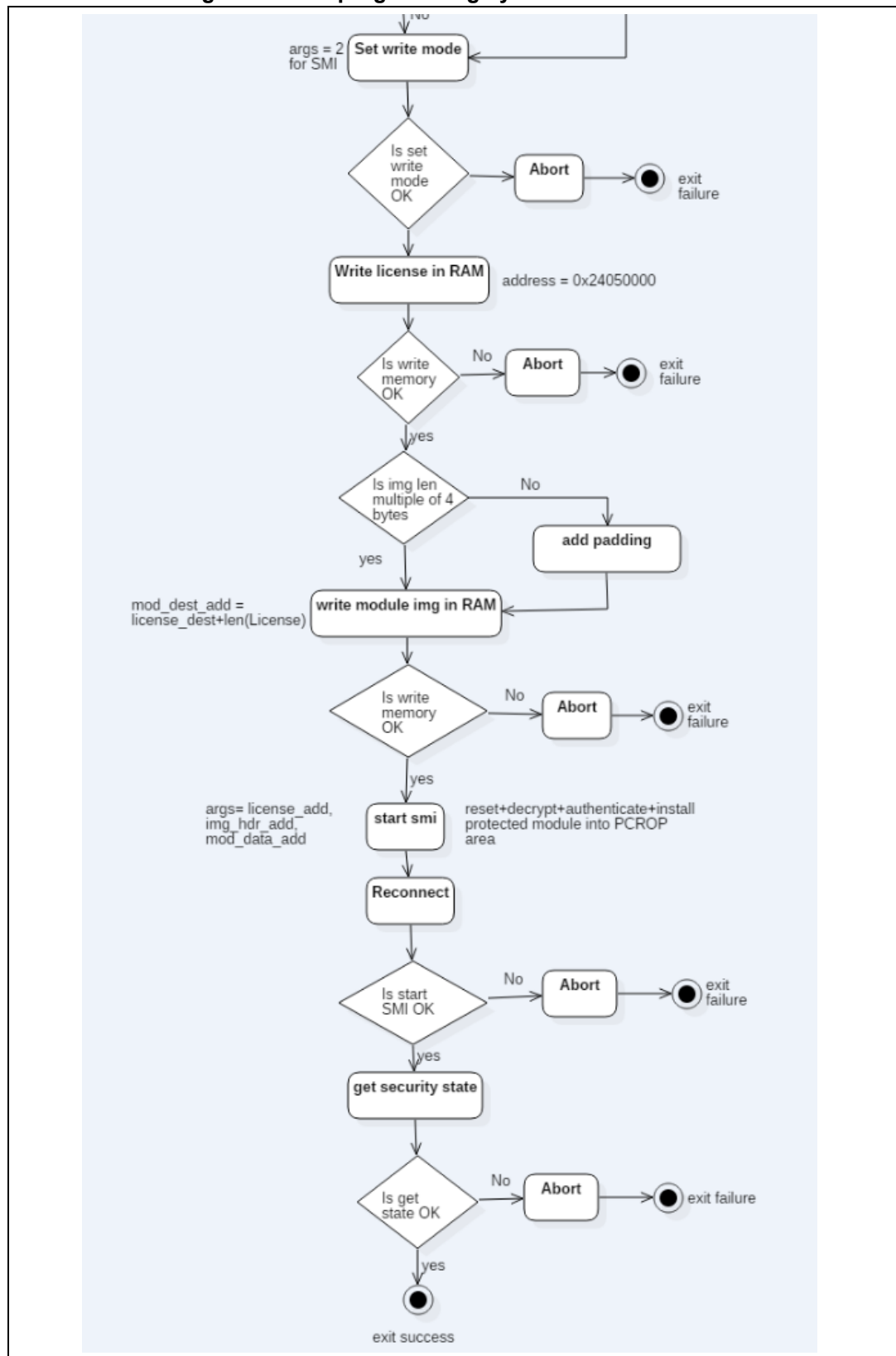
This means that the whole SMI image and its corresponding license must be transferred to RAM before starting. Then, to access RSS services through JTAG, there are two options:

- write a small program in RAM that calls the public API
- use the secure API directly.

Once the RSS function “RSS\_SMI\_resetAndInstallModules” execution has finished successfully, the SMI module is decrypted, Flashed and protected by the PCROP mechanism.

The essential steps of the SMI programming by JTAG flow are described in [Figure 38](#).

Figure 38. 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.

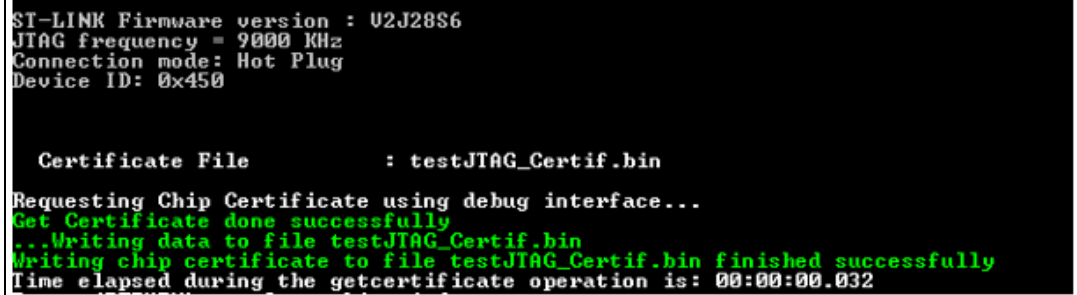
*Note:* Using a debug connection "HOTPLUG" mode must be used with the connect command.

**Example of "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 39](#).

**Figure 39. 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 of "smi" command using SWD**

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

## 5 Example SFI programming scenario

### 5.1 Scenario overview

The actual user application to be installed on the STM32H753xl 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 HW and SW prerequisites are needed:

- STM32H743I-EVAL board
- STM32H753xl with Bootloader v13.2-RC2 and RSS v0.9 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](http://www.st.com)
- STM32CubeProgrammer v0.4.0 (or greater) package available from [www.st.com](http://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.

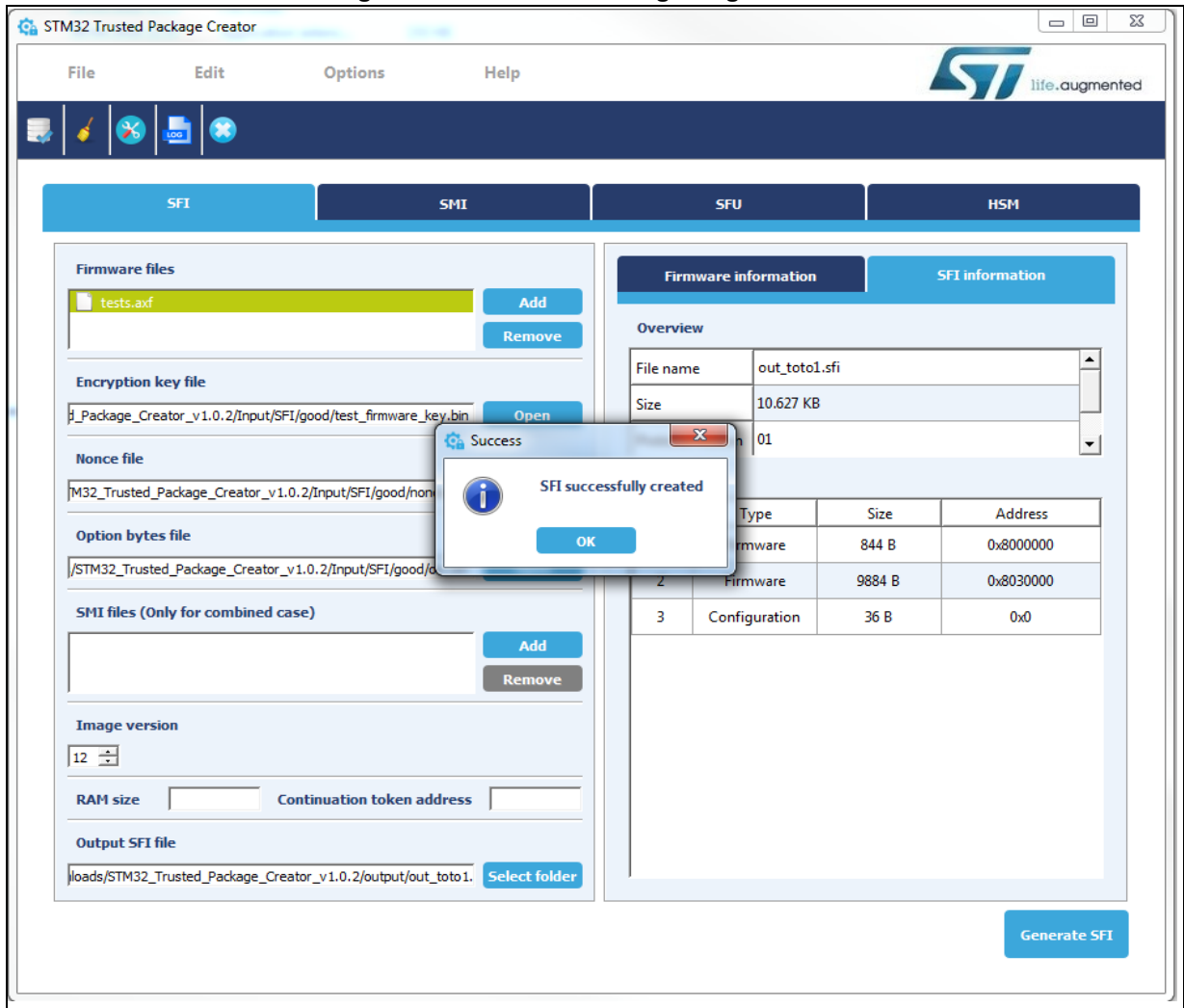
#### 5.3.2 Perform the SFI generation (GUI mode)

To be encrypted with the STM32TrustedPackageCreator 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*).

[Figure 40](#) shows the STPC GUI during the SFI generation.

Figure 40. 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 41](#) 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.

**Figure 41. Example of HSM programming using STPC GUI**

STM32 Trusted Package Creator

File Edit Options Help

SFI SFIx SMI HSM

HSM card index

1

Firmware identifier

HSMv2\_SLOT\_1

Encryption key file

C:/TrustedFiles/key.bin Open

Nonce file

C:/TrustedFiles/nonce.bin Open

Personalization data file

C:/TrustedFiles/enc\_ST\_Perso\_L5.bin Open

Maximum counter

1000

HSM information

Firmware ID	HSMv2_SLOT_1
Max counter	1000
HSM status	OPERATIONAL_STATE
Version	2
Type	SFI

Clear Refresh

Program HSM

**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.

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

STM32TrustedPackageCreator 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 -mc 2000
```

- -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 license.

#### Example of HSM version 2 provisioning:

A new option [-pd] should 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.

**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.

#### 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 42. HSM information in STM32TrustedPackageCreator CLI mode

```

-----
STM32TrustedPackageCreator v1.2.0
-----

ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x71CB0000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x71D2F560

Read the following Information from HSM slot 1 :

HSM STATE : OPERATIONAL_STATE

HSM FW IDENTIFIER : HSMv2_SLOT_2

HSM COUNTER : 2000

HSM VERSION : 2

HSM TYPE : SFI

```

### 5.3.5 Programming input conditions

Before performing an SFI install be 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 can be 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 BL 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).

### 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>
```

Figure 43 shows the SFI install via SWD execution and the HSM as license generation tool in the field.

Figure 43. SFI install success using SWD connection (1)

```

-----
STM32CubeProgrammer v1.0.7
-----
ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: U2J30M19
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 : 0x80000000
    Parsing Area 2/3      :
      Area type           : F
      Area size           : 10528
      Area destination address : 0x80300000
    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 0x24030000
Writing img header to address 0x24031000
Writing areas and areas wrapper...
all areas processed
RSS process started...

RSS command execution OK

```

Figure 44. SFI install success using SWD connection (2)

```
RSS command execution OK
Reconnecting...
ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: 02J30M19
Target voltage: 3.21V
Error: ST-LINK error <DEU_NO_DEVICE>
...retrying...
ST-LINK SN: 0672FF554949677067034831
ST-LINK Firmware version: 02J30M19
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 SMI programming scenario

### 6.1 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.

### 6.2 Hardware and software environment

The same environment as explained in [Section 4.1.1: Device authentication](#).

### 6.3 Step-by-step execution

#### 6.3.1 Build 3<sup>rd</sup> 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.

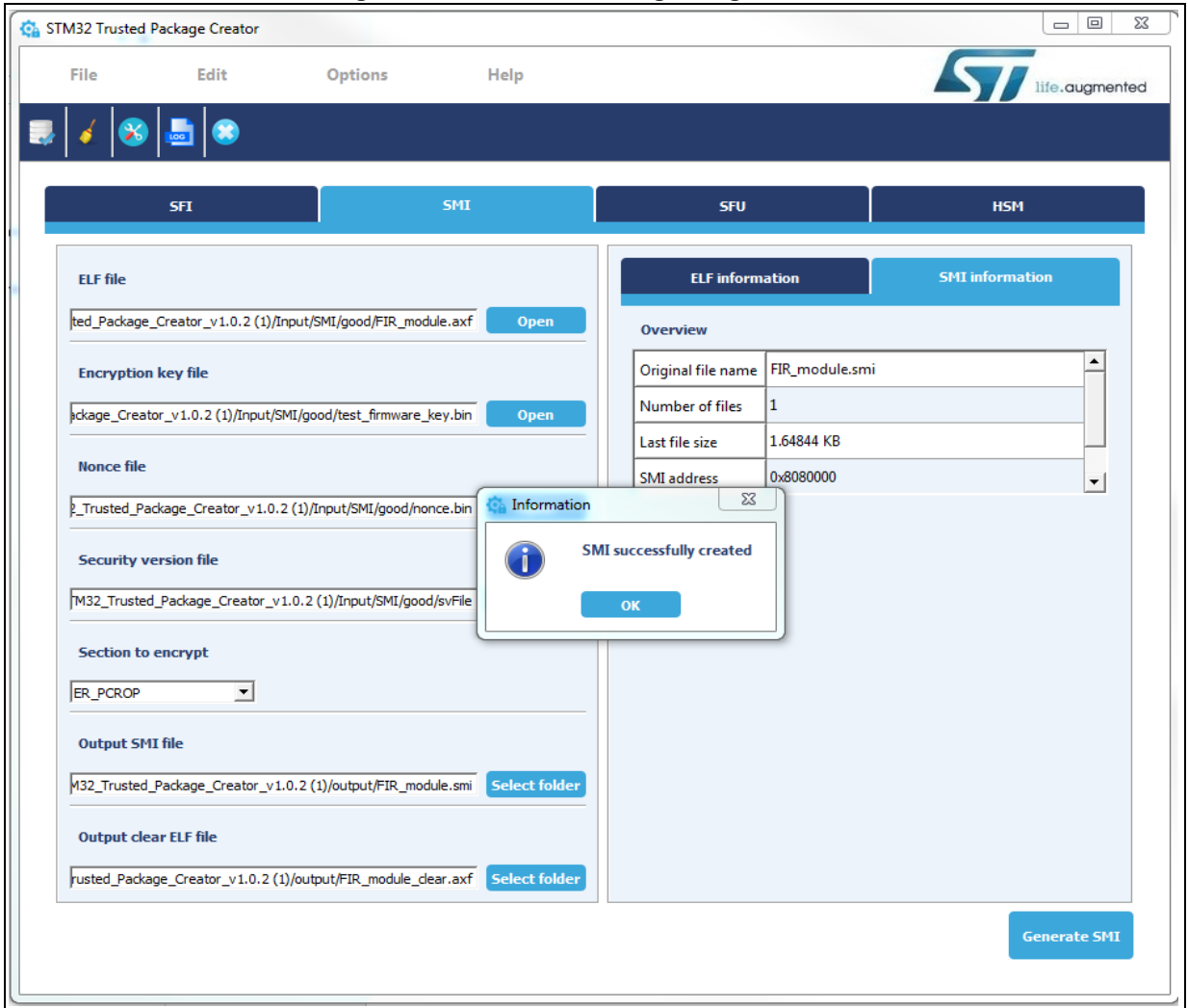
6.3.2 Perform the SMI generation

For encryption with the STM32TrustedPackageCreator 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 37 shows the STPC GUI during SMI generation.

Figure 45. STPC GUI during SMI generation





### 6.3.3 Programming input conditions

Before performing the SMI install be 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 BL, 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).

### 6.3.4 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 46](#) shows the SMI install via SWD execution:

Figure 46. 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\docs_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=HOTPUG -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_forSTM32H7Release\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.smi
SMI File Information :
SMI file path      : C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\FIR_module.smi
SMI license file path : C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7Release\AN\SMI_ImagePreparation\licenseSMI.bin
SMI code destination address : 0x80800000
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 address 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_forSTM32H7Release\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>

```

### 6.3.5 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 47](#)).

Figure 47. 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 U>

User Configuration:
  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 erase 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 should appear.

## 7 Example combined SFI-SMI programming scenario

### 7.1 Scenario overview

The actual user application to be installed on the STM32H753xl 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).

### 7.2 Hardware and software environment

The same environment as explained in [Section 5.2: Hardware and software environment.](#)

### 7.3 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 STMicroelectronics or 3<sup>rd</sup> parties for example.

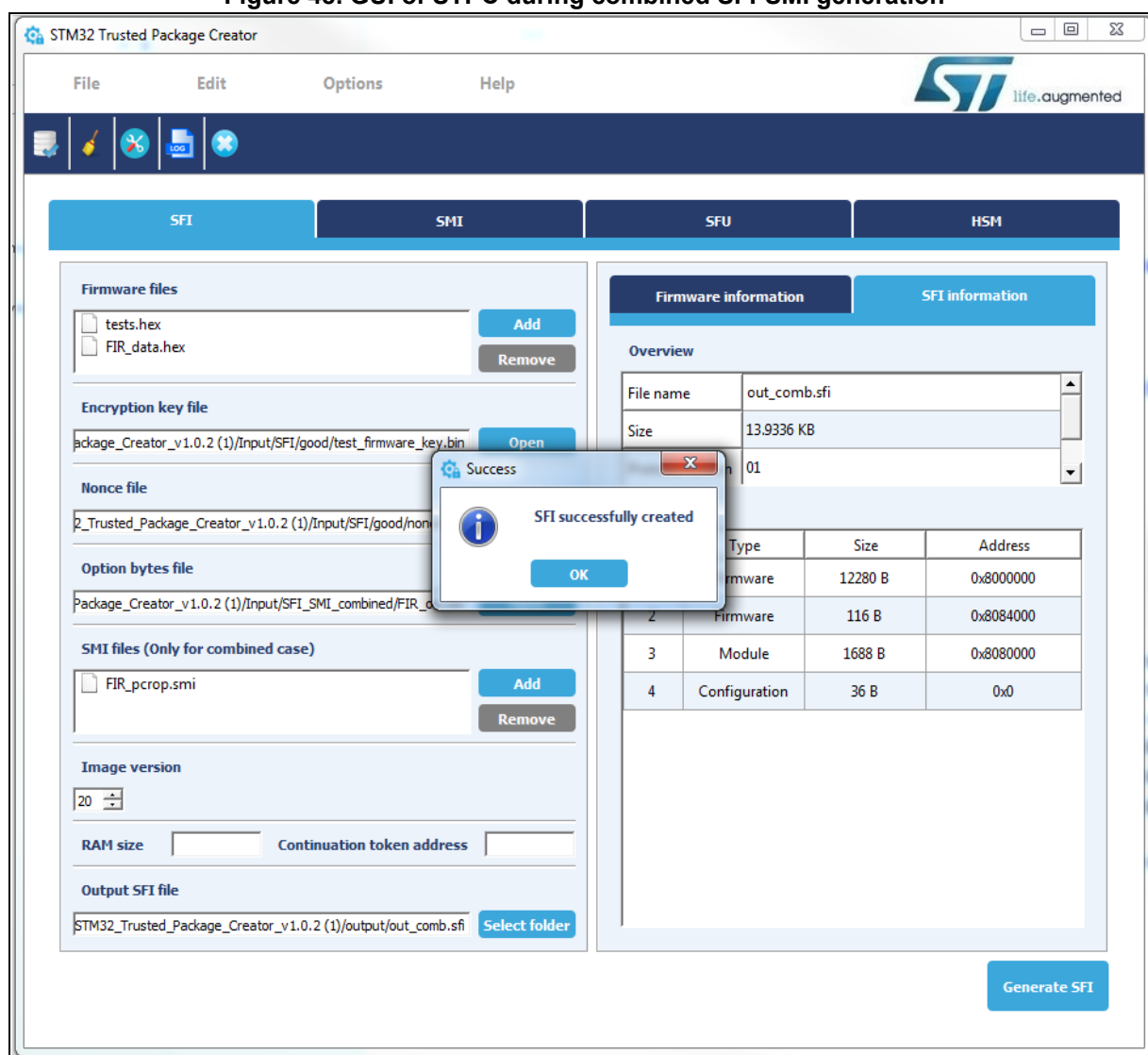
In this example we use firmware based on a single library (just one SMI module is integrated in the SFI image).

2. Perform the SFI generation.

For encryption with the STM32TrustedPackageCreator 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).

Figure 48 shows the STPC GUI during combined SFI generation.

Figure 48. GUI of STPC during combined SFI-SMI generation



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

### 7.3.1 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 49* shows the combined SFI-SMI install trace success.

Figure 49. Combined SFI-SMI programming success using debug connection

```

ST-LINK Firmware version : U2J26M15
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 : U2J26M15
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...

```



### 7.3.2 How to test the combined SFI install success

The option bytes configuration should be modified as shown in [Figure 50](#).

- 3<sup>rd</sup> party library module is programmed 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 should appear.

Figure 50. 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_LEU      : 0x2 <reset level is set to 2.7 V>

User Configuration:
  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 erase 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>

```

## 8 Reference documents

**Table 2. Document references**

Reference	Version	Document title
[1]	Latest version	AN4992, STM32H7 secure firmware/module install overview. STMicroelectronics.
[2]	Latest version	UM2428, Hardware secure modules (HSM) for secure firmware install (SFI). STMicroelectronics.

## 9 Revision history

**Table 3. 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: <ul style="list-style-type: none"><li>– <a href="#">Section 4.1.2: License mechanism</a></li><li>– <a href="#">Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)</a></li><li>– <a href="#">Figure 32: HSM programming toolchain</a> (title caption)</li><li>– <a href="#">Figure 41: Example of HSM programming using STPC GUI</a></li></ul>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

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

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

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

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

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

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

© 2019 STMicroelectronics – All rights reserved