
STM32 Trusted Package Creator tool software description

Introduction

STM32 Trusted Package Creator is part of the STM32CubeProgrammer tool set (STM32CubeProg), and allows the generation of secure firmware and modules to be used for STM32 secure programming solutions which are:

- Secure internal firmware install (SFI): SFI is a secure mechanism that allows secure installation of OEM internal firmware in untrusted production environments by encrypting the whole internal firmware with an AES-GCM key.
- Secure external firmware install (SFIx): SFIx is a secure mechanism that allows secure installation of external OEM firmware in untrusted production environments by encrypting the whole external firmware with an AES-GCM key.
- Secure module install (SMI): SMI is intended to protect a part of the firmware (a section of an ELF file) by also encrypting this section using an AES-GCM key.

A combined SFI-SMI image is an SFI image that contains one or more module areas.

This user manual details the software environment prerequisites, as well as the available features of the STM32 Trusted Package Creator tool software.



Contents

1	System requirements	6
2	Preparation processes	7
2.1	SFI preparation process	7
2.2	SFlx preparation process	9
2.2.1	Area E	9
2.2.2	Area K	9
2.2.3	Constraints for SFlx images	10
2.3	SMI preparation process	11
2.4	HSM preparation process	13
2.4.1	Why we need an HSM	13
2.4.2	HSM programming	14
2.4.3	Reading HSM information	16
3	STM32 Trusted Package Creator tool commands	17
3.1	Command line interface (CLI)	17
3.2	HSM provisioning command	19
3.3	SFI/SFlx generation command	22
3.4	SMI generation command	25
3.5	STM32WB firmware signing	27
4	STM32 Trusted Package Creator tool graphical user interface (GUI)	28
4.1	SFI generation	30
4.2	SMI generation	33
4.3	SFlx generation	36
5	Option bytes file	39
6	Log dialog	40
7	Settings	41
8	SFI/SMI checking	42

9	Reference documents	43
10	Revision history	44

List of tables

Table 1. Document references 43

Table 2. Document revision history 44



List of figures

Figure 1.	SFI preparation process	7
Figure 2.	SFI file structure	8
Figure 3.	Example of split of SFIx image	10
Figure 4.	SMI preparation process	11
Figure 5.	SMI file structure	12
Figure 6.	HSM programming tab	15
Figure 7.	Reading HSM information	16
Figure 8.	STM32 Trusted Package Creator tool's available commands (part 1)	17
Figure 9.	STM32 Trusted Package Creator tool's available commands (part 2)	18
Figure 10.	Get HSM information in CLI mode	20
Figure 11.	SFI generation with an ELF file	24
Figure 12.	SFI generation with a binary file	24
Figure 13.	Combined SFI-SMI generation	25
Figure 14.	SMI generation	26
Figure 15.	STM32 Trusted Package Creator tool GUI SFI tab	28
Figure 16.	STM32 Trusted Package Creator tool GUI SMI tab	29
Figure 17.	STM32 Trusted Package Creator tool GUI SFIx tab	30
Figure 18.	Firmware file addition	31
Figure 19.	Successful SFI generation	32
Figure 20.	ELF file selection	33
Figure 21.	Successful SMI generation	35
Figure 22.	External firmware file selection	36
Figure 23.	Successful SFIx generation	38
Figure 24.	Example of an option bytes file	39
Figure 25.	Example of a log dialog	40
Figure 26.	Settings dialog	41
Figure 27.	SFI checking	42

1 System requirements

Supported operating systems and architectures:

- Linux® 64-bit
- Windows® 7/8/10 32-bit and 64-bit
- macOS® (minimum version OS X® Yosemite)

STM32CubeProgrammer and STM32 Trusted Package Creator support STM32 32-bit devices based on Arm®(a) Cortex®-M processors.

arm

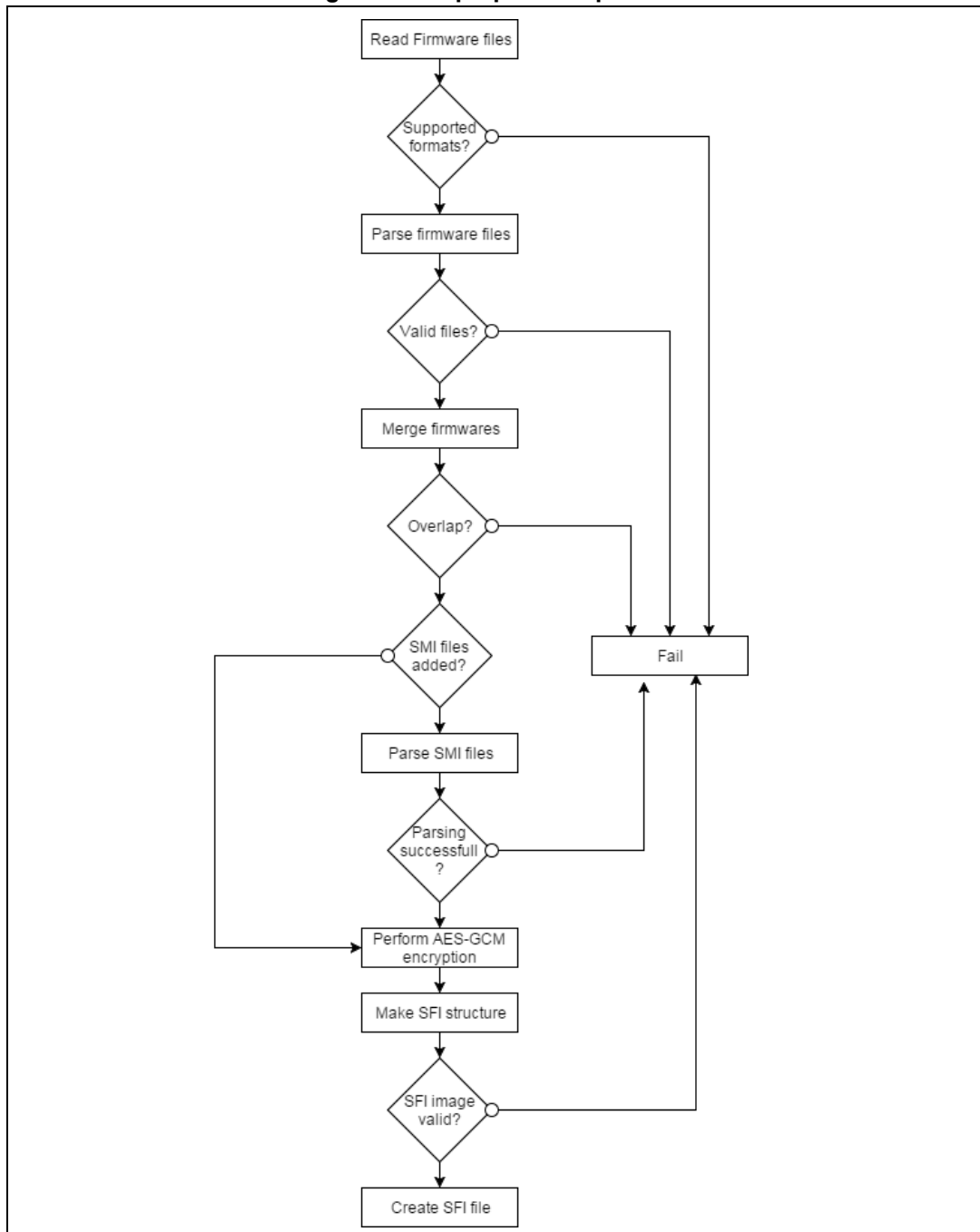
a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.

2 Preparation processes

2.1 SFI preparation process

An SFI (Secure firmware install) image is a format created by STMicroelectronics that contains an encrypted and authenticated piece of firmware using an AES-GCM algorithm. The SFI preparation process is described in [Figure 1](#).

Figure 1. SFI preparation process



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

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

Where nonce is a number used only once as a start of an iterated process in the AES-GCM algorithm to give different cipher texts to same blocks of data.

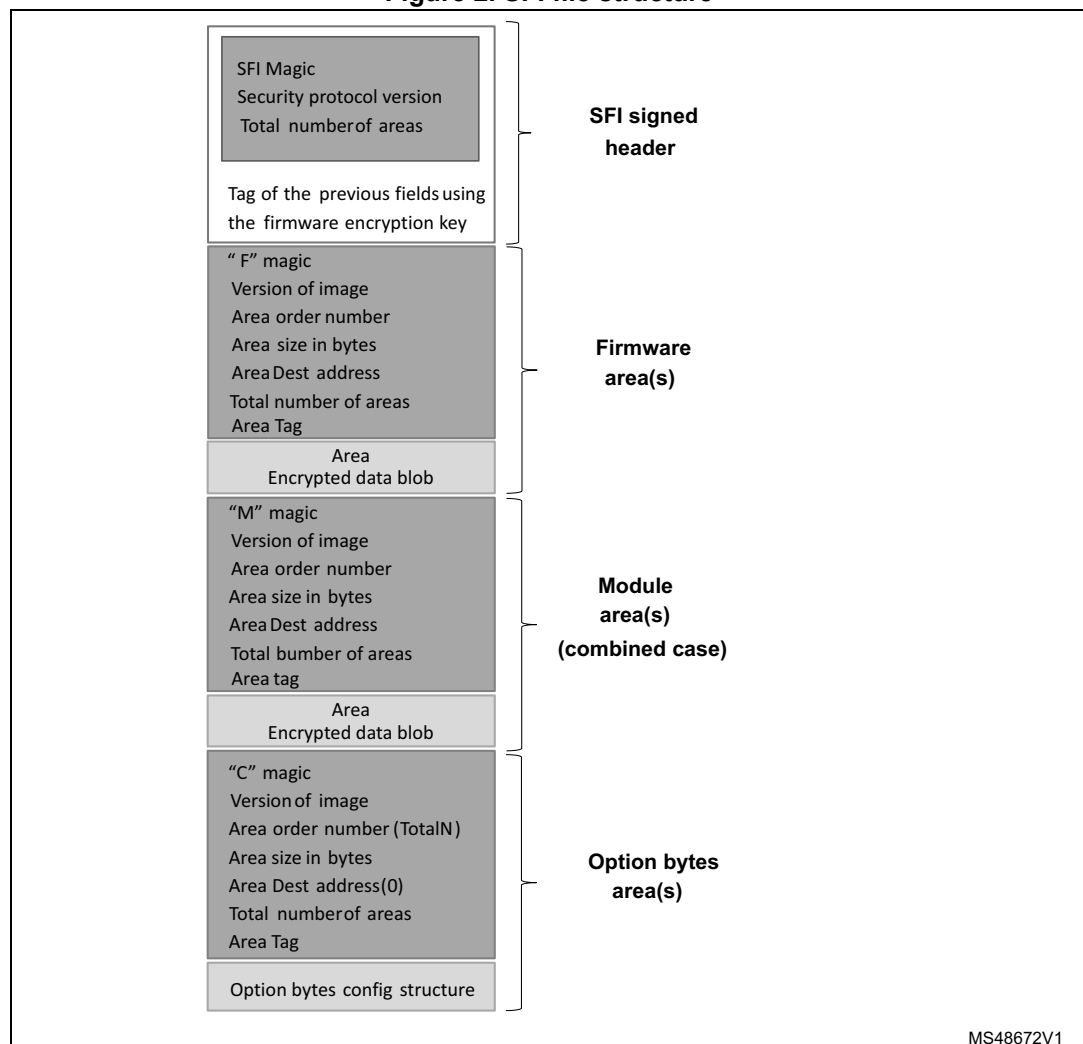
It then, it passes the area descriptor (starting from the magic to the total number of areas) as additional authenticated data (AAD).

- Each segment in the input firmwares constitutes a firmware (F) area in the SFI file.
- Each SMI file (combined case) constitutes a module (M) area.
- The option bytes configuration constitute the configuration (C) area.

To generate a header tag, the tool performs an authenticated only AES-GCM encryption (without a plain text nor a cipher text) using the SFI header as an AAD and the nonce as the IV.

The structure of an SFI file is shown in [Figure 2](#).

Figure 2. SFI file structure



To prepare an SFI image from multiple firmware files you have to make sure that there is no overlap between their segments, otherwise an error message is shown: *“Overlap between segments, unable to merge firmware files”*.

Furthermore, in the case of a combined SFI-SMI image, there is also an overlap check between the areas (in case there is an overlap between firmware and module areas). If the check fails, an error message is shown: *“Overlap between SFI areas”*.

Also, all SFI areas must be located in Flash memory, otherwise the generation fails giving the error message: *“One or more SFI areas are not located in Flash memory”*.

2.2 SFlx preparation process

In addition to the SFI preparation process, mentioned in the previous paragraph, two new areas are added in SFI image for the SFlx preparation process:

- Area ‘E’ = firmware for external flash memory.
- Area ‘K’ = area to trigger random keys generation by RSS.

The key ‘K’ area is optional and the key can be stored in the area ‘F’.

2.2.1 Area E

Area ‘E’ is for external Flash memory. It includes the following information at the beginning of the encrypted payload:

- OTFD region_number (uint32_t):
 - 0...3: OTFD1 (STM32H7A/B and STM32L5)
 - 4...7: OTFD2 (STM32H7A/B)
- 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 (Enhanced cipher)
- 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...).

2.2.2 Area K

Area ‘K’ triggers random key generation by RSS. It contains N couples, each defining 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
```

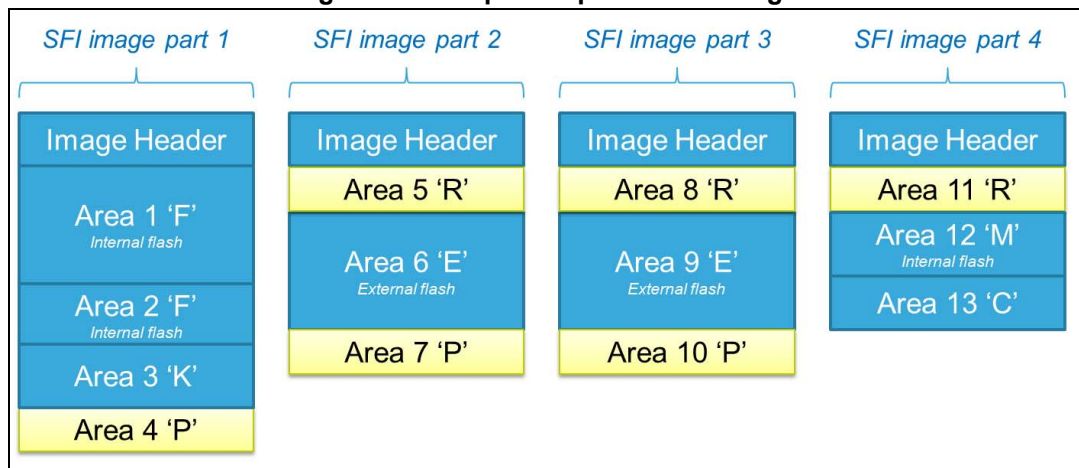
There are two key areas:

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

2.2.3 Constraints for SFlx images

- Area 'K' should be before any area 'E'.
- Several area 'E's can be in the same image part (but must be stored at different RAM addresses).
- One area 'E' or several successive area 'E's must be at the end of an SFI image part (hence followed by an area 'P').
- Areas 'M' and 'C' must be in the last part of the SFI image.

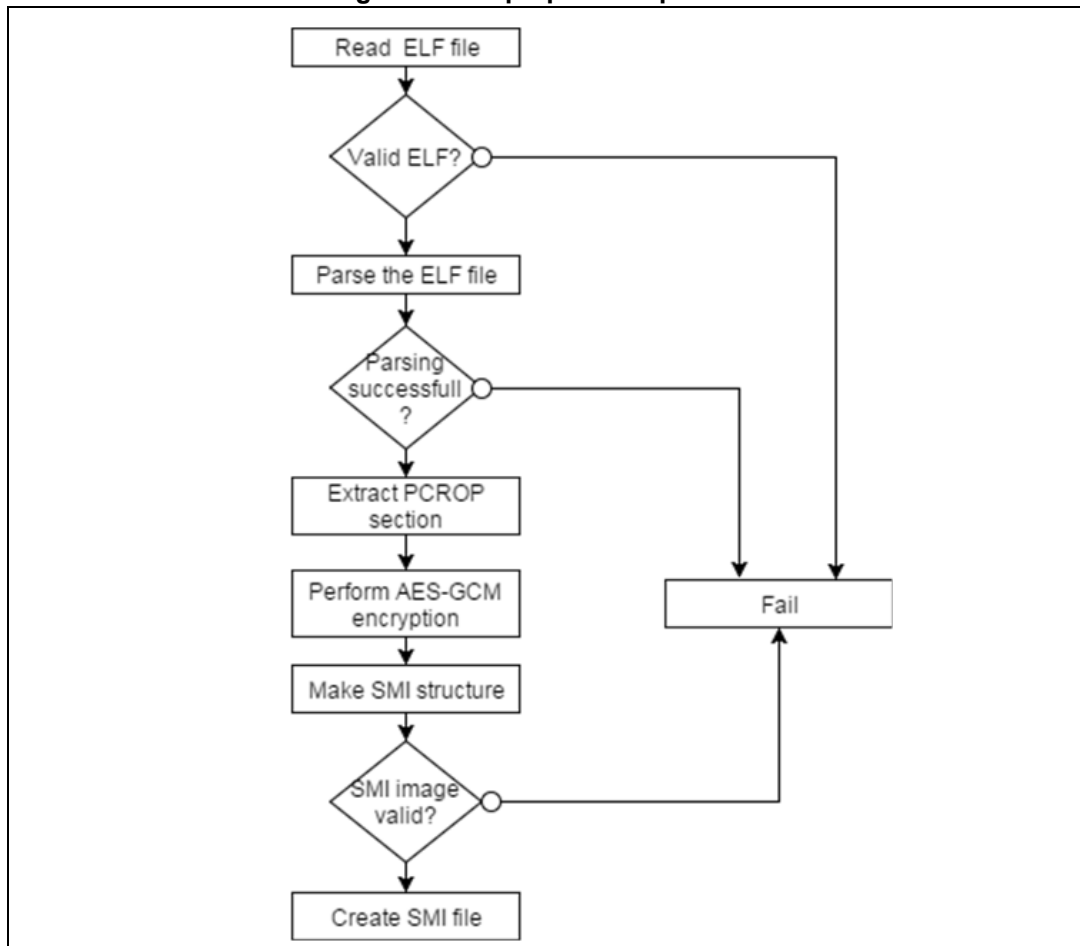
Figure 3. Example of split of SFlx image



2.3 SMI preparation process

An SMI image (Secure Module Install) protects only a module of the firmware. The SMI preparation process is shown in [Figure 4](#).

Figure 4. SMI preparation process



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

- Nonce as the initialization vector (IV)
- The security version as additional authenticated Data (AAD)

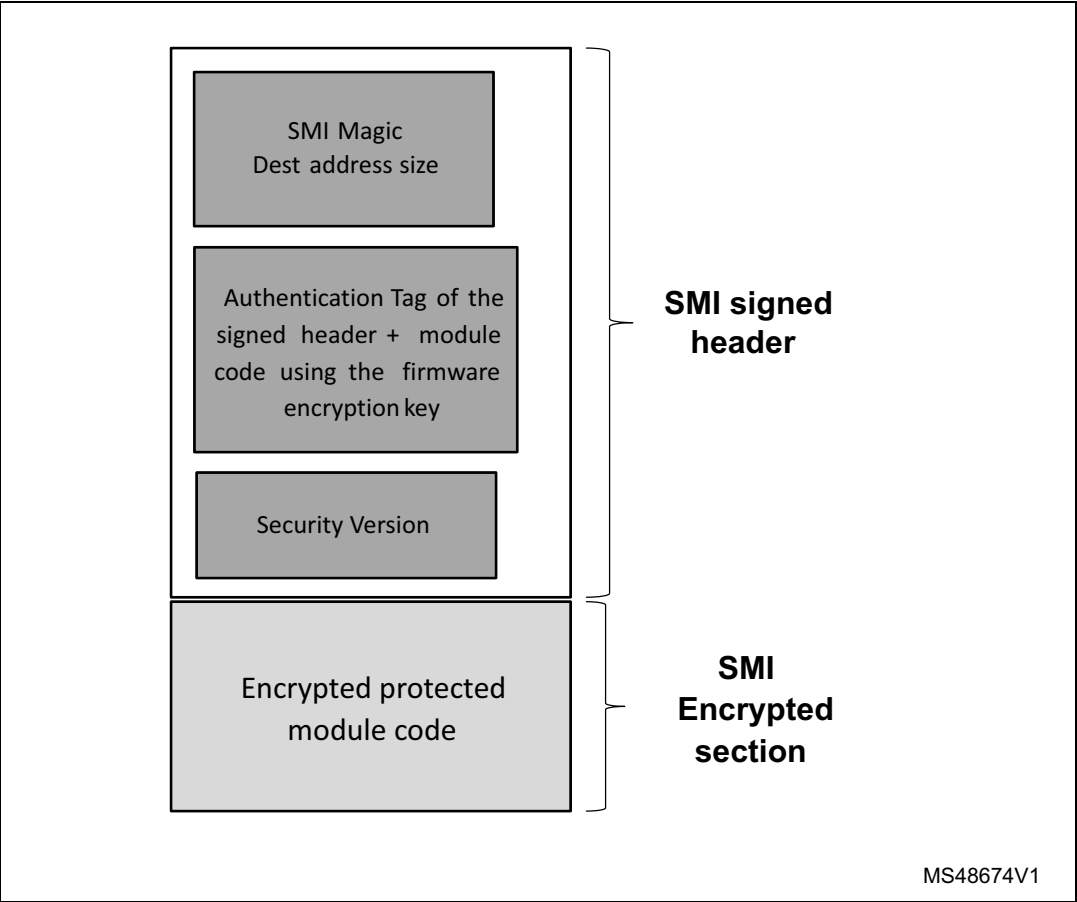
Before SMI preparation, the following checks are performed:

- A proprietary code read out protection (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) or a warning is shown
- If the section immediately following the PCROP area starts in the last Flash word of the PCROP section, the generation fails and an error message is shown.

After the SMI preparation, a clear (that is, not encrypted) ELF file is also generated containing program data and only clear sections of the code.

The structure of an SMI file is shown in [Figure 5](#).

Figure 5. SMI file structure



2.4 HSM preparation process

This section describes the steps required to configure a hardware secure module (HSM) to generate firmware licenses for STM32 secure programming using STM32 Trusted Package Creator.

2.4.1 Why we need an HSM

Some STM32 devices offer secure firmware flashing. Firmware can be a complete application, a library, a secure library (for STM32 with TrustZone), or other secret information.

An important secure flashing feature is controlled by the firmware provider of the number of devices that can be programmed. This is where the concept of licenses is important.

The firmware maker provides licenses to its customers, authorizing them to flash the encrypted firmware (SFI and/or SMI) on specific devices.

A license is only valid for a single device, but the firmware provider can distribute as many licenses as required.

STMicroelectronics offers the secure firmware flashing service based on HSM (hardware secure modules) as a license generation tool to be deployed in the programming house. The general scheme is as follows:

1. Generate a firmware key and use it to encrypt the firmware (SFI/SMI) when the firmware provider wishes to distribute a new firmware.
2. While downloading the firmware to a device, a device identifier is sent to the HSM, which returns a license for the identified device. The license contains the encrypted firmware key, and only this device can decrypt it.
3. The number of programmed devices is under HSM control. It is decremented once a license is generated.

STM32 Trusted Package Creator allows HSM configuration for secure firmware flashing as described in [Section 2.4.2: HSM programming](#).

STMicroelectronics provides two versions of HSM for secure programming, each having a specific use:

- **HSMv1:** static HSM. This allows generation of firmware licenses for STM32 secure programming devices that are chosen in advance. Each product ID needs a single HSM, to be configured by STMicroelectronics then shipped to the OEM.
- **HSMv2:** dynamic HSM. This version allows generation of firmware licenses targeting STM32 secure programming devices that are chosen via a personalization data at the OEM site.

2.4.2 HSM programming

When an OEM needs to deliver an HSM to a programming house to be deployed as a license generation tool for relevant STM32 device programming, they must perform some customization on his HSM first.

They must program the HSM with all the data needed for the license scheme deployment in the production line.

This OEM data is:

- **Counter:** The counter is set to a maximum value that corresponds to the maximum number of licenses that can 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 the predefined maximum value (1 million units).
- **Firmware key:** This key is 32 bytes and is composed of two fields, the initialization vector (IV) or nonce (first field), and the key (last field) that were used to AES128-GCM encrypt the firmware and generate the SFI file.
Both fields are 16 bytes long, but the last 4 bytes of the nonce must be zero (only 96 bits of the nonce are used in the AES128-GCM algorithm).
Both fields must remain secret, so they are encrypted before being sent to the chip.
The key and the nonce remain 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.
- **Firmware identifier:** Identifies the correct HSM for a given firmware.
- **Personalization data:** 108 bytes that are encrypted before being sent to the chip in cases where each device has its own configuration.

Once this data is programed into the HSM, the HSM is automatically locked. [Figure 6: HSM programming tab](#) shows the GUI provided by the tool for HSM programming by the firmware provider.

Note: The ST Personalization data field is available only with HSMv2

Note: When the ST personalization package is installed successfully, it is not possible to further modify the personalization data and the HSM goes in the OPERATIONAL_STATE.

Figure 6. HSM programming tab

STM32 Trusted Package Creator

File Edit Options Help

SFI SFIx SMI **HSM**

HSM card index

2

Firmware identifier

Encryption key file

Open

Nonce file

Open

Personalization data file

Open

Maximum counter

0

HSM information

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

Clear Refresh

Program HSM

The tab parameters are as follows:

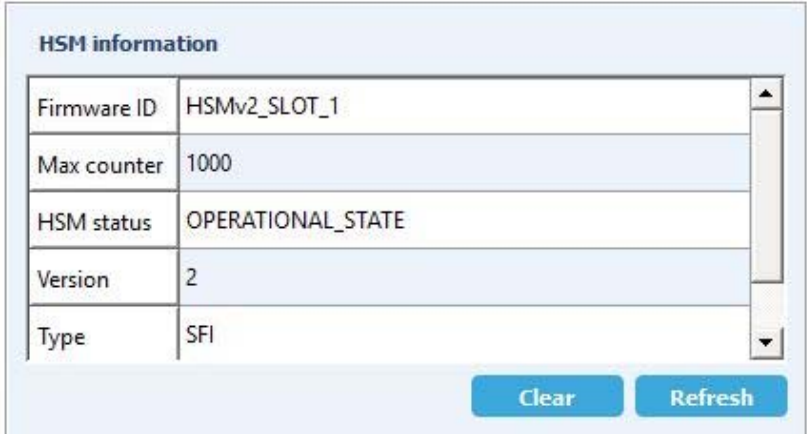
- **HSM card index:** Specifies the smart card reader slot number.
- **Firmware identifier:** Identifies the correct HSM for a given piece of firmware.
- **Encryption key file:** A binary file containing the key used to encrypt the firmware and generate the SFI file. The key is 16 bytes long.
- **Nonce file:** A binary file containing the nonce used to encrypt the firmware and generate the SFI file. The nonce is 12 bytes long.
- **Personalization data file:** A binary file containing the personalization data used to configure the given device. The data is 108 bytes long.
- **Maximum counter:** The maximum number of licenses that can be delivered by the HSM (it aims to prevent over programming). The counter value must not exceed a predefined maximum (1 million units).
When all fields are properly filled in, the user can program the HSM file by clicking on the Program HSM button (the button becomes active).

Caution: Once this data is programed into the HSM, the HSM is automatically locked.

2.4.3 Reading HSM information

The programmed information is verifiable using the HSM information panel shown in [Figure 7](#):

Figure 7. Reading HSM information



The screenshot shows a web-based interface titled "HSM information". It contains a table with five rows of data. To the right of the table is a vertical scrollbar. Below the table are two buttons: "Clear" and "Refresh".

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

Clear Refresh

This panel displays the Firmware identifier, the maximum counter and the HSM status.

- **HSM status** can be:
 - **OEM_STATE**: HSM not programmed.
 - **OPERATIONAL_STATE**: HSM programmed and locked. It can no longer be programmed.
- **Version**: indicates the hardware version of the used HSM, it can be
 - **1**: HSM version 1
 - **2**: HSM version 2.
- **Type**: indicates which security process being used:
 - **SFI**: Static license for a complete application.
 - **SMI**: Static license for a library
 - **SSP**: Static license for secrets.
 - -: type not available.

3 STM32 Trusted Package Creator tool commands

3.1 Command line interface (CLI)

The following sections describe how to use the STM32 Trusted Package Creator tool from the command line interface. The available commands are shown in [Figure 8](#) and [Figure 9](#).

Figure 8. STM32 Trusted Package Creator tool's available commands (part 1)

```

-----
STM32 Trusted Package Creator v1.2.0
-----

Usage :
SFMIPreparationTool_CLI.exe [option1] [value1] [option2] [value2]...

Information options
--?, -h, --help      : Display this help
-l, --log            : Generate a log file
                     [<File_Name>] : Log file's path and name, default file is ./trace.log

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
-firmx, --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, --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

```

Figure 9. STM32 Trusted Package Creator tool's available commands (part 2)

```

-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

SFU preparation options

-sfu, --sfu          : Generate SFU image,
    You also need to provide the information listed below
-fir, --firmware     : Add an input firmware file (must have only 1 segment)
    <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)
-oh, --outhead       : Generated SFU header file
    <Output_File>     : SFU header file to be created
-os, --outsfu        : Generated SFU encrypted image file
    <Output_File>     : SFU encrypted image file to be created

SSP Payload preparation options

-ssp, --ssp          : Generate SSP Payload,
    You also need to provide the information listed below
-b, --blob           : Binary to encrypt
    <Blob>            : Binary file
-pk, --pubk          : ECDSA pubK
    <PubK>            : binary file of size 178 bytes
-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 16 bytes
-s, --secretsSize    : Secret Size
    <Secret_Size>     : number in any base
-o, --out            : Generated payload
    <Output_File>     : Payload file to be created

HSM provisioning

-hsm, --hsm          : Program a HSM card,
    You also need to provide the information listed below
-i, --index          : HSM card index
    <Index>           : Index if the HSM (default 2)
-k, --key            : AES-GCM encryption key
    <Key_File>        : Bin file, its size must be 16 bytes,
    the same used in SFI/SMI creation
-n, --nonce          : AES-GCM nonce
    <Nonce_File>      : Bin file, its size must be 12 bytes,
    the same used in SFI/SMI creation
-id, --id            : Firmware identifier
    <Firmware_Id>     : Identifier as a string
-mc, --maxcounter    : Maximum number of licenses to request,
    must not exceed 1000000
    <Max Counter>     : the maximum number
-pd, --persodata     : Encrypted ST personalization data file for HSMv2
    <Persodata_File>  : Bin file, its size must be 108 bytes,
-info, --info        : Get HSM informations

ECC Sign binary commands

-sign, --sign        : Sign a binary image using ECDSA algorithm
    and save the public key in a binary file
    in the same folder as the private key.
    You need to provide the information listed below
-bin, --binary       : Input image to be signed
    <Img_File>        : Binary image file path
-prvk, --privatekey, : EC private key
    <prvk_File>       : Private key file Path
-v, --version        : Signed image version
    <Img_Ver>        : 32 bits value (ver,sub ver, branch , build)
-o, --outfile        : Output image file path
    <Output_File>    : Signed image file path

```

3.2 HSM provisioning command

-hsm, --hsm

Description: This command allows to program HSM card.

In order to configure the HSM before programming, the user must provide the mandatory inputs by using the options listed below.

-i, --index

Description: Select the card index, number in decimal format.

Syntax: -i <number>

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: -k <Key_File>

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

-n, --nonce

Description: Set the AES-GCM nonce.

Syntax: -n <Nonce_File>

<Nonce_File>: Bin file path, its size must be 12 bytes.

-id, --id

Description: Set the firmware identifier.

Syntax: -id <Firmware_Id>

<Firmware_Id>: Input as a string format, don't use spaces.

-mc, --maxcounter

Description: Set the maximum number of licenses to be requested.

Syntax: -mc <Max_Counter>

<Max_Counter>: Number in decimal format, must not exceed 1000000.

-pd, --persoData

Description: Set the personalization data configuring the HSM version 2.

Syntax: -pd <PersoData_File>

<PersoData_File>: Bin file path. Its size must be 108 bytes.

-info, --info

Description: Get HSM information.

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
```

Example of HSM version 2 provisioning:

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

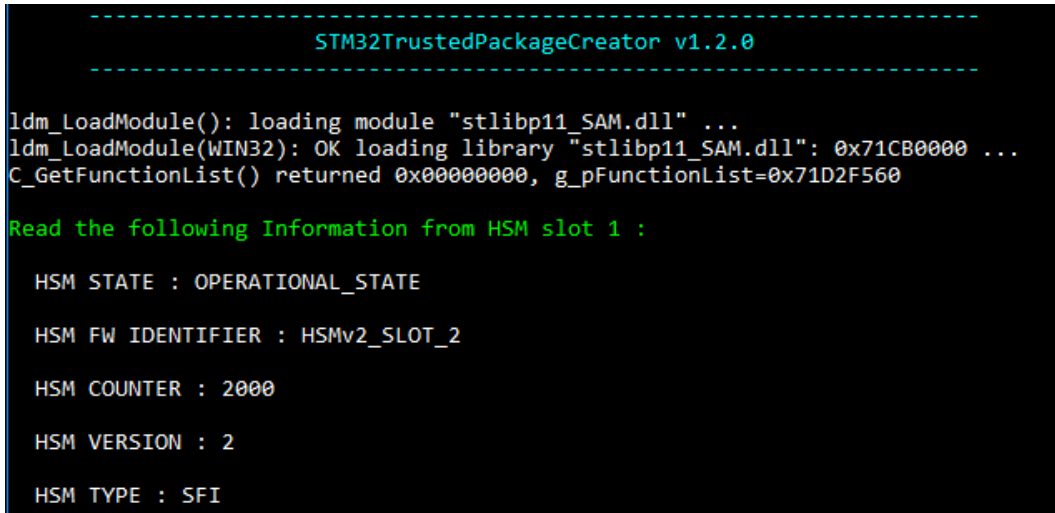
```
STM32TrustedPackageCreator_CLI -hsm -i 1 -k
"C:\TrustedFiles\key.bin" -n "C:\TrustedFiles\nonce.bin" -id
HSMv2_SLOT_2 -mc 2000 -pd "C:\TrustedFiles\enc_ST_Perso_L5.bin"
```

Note: *Note: If the HSM was 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:

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

Figure 10. Get HSM information in 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
```

Example of HSMv2 personalization data:

In order to configure the HSM before programming, the user must indicate a mandatory input, “-pd”, to set specific personalization data for the device by including the appropriate file containing the configuration.

The STM32TrustedPackageCreator tool provides all personalization package files, ready to be used on SFI/SFIx and SSP flows. 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. You must select the correct one.

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

- Use the STM32CubeProgrammer tool to launch the Get Certificate command to generate a certificate file containing some chip security information, knowing that this command is 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 represent 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 now 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, the associated information is:

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

3.3 SFI/SFlx generation command

-sfi, --sfi

Description: This command generates an SFI image file.

In order to generate an SFI image, the user must provide the mandatory inputs by using the options listed below.

-fir, --firmware

Description: Add 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>] : Address 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 the case of BIN input file (in any base).

[<Region_Number>] : Only in the 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 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 the case of BIN input file (in any base), random key values in internal flash memory

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: -k <Key_file>

<Key_file> : A 16-byte binary file.

-kx, --keyx

Description: Set the AES-GCM encryption key for external flash loader.

Syntax: -kx <Key_area_File>

-n, --nonce

Description: Set the AES-GCM nonce.

Syntax: -n <Nonce_file>

<Nonce_file> : A 12-byte binary file.

-v, --ver

Description: Set the image version.

Syntax: -v <Image_version>

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

-ob, --obfile

Description: Provide an option bytes file.

Syntax: -ob <CSV_file>

<CSV_file>: A csv file with 9 values.

-m, --module

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

Syntax: -m <SMI_file>

<SMI_file> :SMI file.

[<Address>] : Address only for relocatable SMI.

-rs, --ramsize

Description: Define available ram size for SFI programming (for multi-image).

Syntax: -rs <size>

< size >: Size in bytes.

-ct, --token

Description: Address at which to store the Continuation token. The address must not collide with other areas. (for multi-image).

Syntax: -ct <Address>

-o, --outfile

Description: Set the SFI file to be created

Syntax: -o <out_file>

<out_file> :SSFI file to be generated, must have the .sfi/sfix extension.

Example for SFI:

With an ELF file:

```
STM32TrustedPackageCreator_CLI -sfi -fir ELF_firmware.axf
-k
test_firmware_key.bin -n nonce.bin -ob FIR_ob.csv -v
23 -o test.sfi
```

Figure 11. SFI generation with an ELF file

```
C:\STM32TrustedPackageCreator\bin>STM32TrustedPackageCreator_CLI.exe -sfi -fir E
LF_firmware.axf -k test_firmware_key.bin -n nonce.bin -ob FIR_ob.csv -v 23 -o te
st.sfi
SUCCES
```

With a binary file:

```
STM32TrustedPackageCreator_CLI -sfi -fir bin_firmware.bin
0x80000000 -k test_firmware_key.bin -n nonce.bin -ob
FIR_ob.csv -v 23 -o test.sfi
```

Figure 12. SFI generation with a binary file

```
C:\STM32TrustedPackageCreator\bin>STM32TrustedPackageCreator_CLI.exe -sfi -fir b
in_firmware.bin 0x00000000 -k test_firmware_key.bin -n nonce.bin -ob FIR_ob.csv
-v 23 -o test.sfi
SUCCES
```


Combined SFI-SMI:

```
STM32TrustedPackageCreator_CLI -sfi -fir ELF_firmwrae.axf
-fir bin_firmware.bin 0x8000000 -m FIR_pcrop.smi -k
test_firmware_key.bin -n nonce.bin -ob FIR_ob.csv -v
23 -o test.sfi
```

Figure 13. Combined SFI-SMI generation



```
C:\STM32TrustedPackageCreator\bin>STM32TrustedPackageCreator_CLI.exe -sfi -fir b
in_firmware.bin 0x00000000 -m FIR_pcrop.smi -k test_firmware_key.bin -n nonce.bi
n -ob FIR_ob.csv -v 23 -o test.sfi
SUCCESS
```

3.4 SMI generation command

-smi, --smi

Description: This command generates an SMI image file.

In order to generate an SMI image, the user must provide the mandatory inputs by using the options listed below.

-elf, --elffile

Description: Set the input ELF file.

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: Set the name of the section to be encrypted.

Syntax: -s <section_name>

<section_name>: Section name.

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: -k <Key_file>

<Key_file> : A 16-byte binary file.

-n, --nonce

Description: Set the AES-GCM nonce.

Syntax: -n <Nonce_file>

<Nonce_file> : A 12-byte binary file.

-sv, --sver

Description: Set the security version file.

Syntax: -sv <SV_file>

<SV_file> : A 16 byte file.

-o, --outfile**Description:** Set the SMI file to be created**Syntax:** -o <out_file>

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

-c, --clear**Description:** Set the clear ELF file to be created.**Syntax:** -c <ELF_file>

<ELF_file> :Clear ELF file to be generated.

Example

```
STM32TrustedPackageCreator_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
```

Figure 14. SMI generation

```
C:\STM32TrustedPackageCreator\bin>STM32TrustedPackageCreator_CLI.exe -smi -elf F  
IR_module.axf -s "ER_PCROP" -k test_firmware_key.bin -n nonce.bin -sv svFile -o  
test.smi -c clear.axf  
Warning: The section does not end on a Flash word boundary  
SUCCESS
```

3.5 STM32WB firmware signing

-sign, --sign

Description: Sign a binary image using ECDSA algorithm and save the public key in a binary file in the same folder as the private key. You need to provide the information listed below.

-bin, --binary : Input firmware image in a binary format to be signed
<Img_File> : Binary image file path
-prvk, --privatekey : EC private key in PEM format with Prime256v1 curve.

It can be generated using the following OpenSSL-tool command:

openssl.exe ecparam -name prime256v1 -genkey -noout -out prvkey.pem

<prvk_File> : Private key file Path
-v, --version : Signed image version
<Img_Ver> : Version number. Should be integer.
-o, --outfile : Output image file path
<Output_File> : Signed image file path

Before loading the signed image to STM32WB devices using STM32CubeProgrammer via -fwupgrade command, you need to download the binary public key using the - authkeyupdate command.

For more information about these STM32CubeProgrammer commands please refer to UM2237.

4 STM32 Trusted Package Creator tool graphical user interface (GUI)

This section describes how to use the STM32 Trusted Package Creator tool with its graphical user interface.

The STM32 Trusted Package Creator tool GUI presents three tabs: one for SFI generation (Figure 15), one for SMI generation (Figure 16) and one for SFU generation (Figure 17).

Figure 15. STM32 Trusted Package Creator tool GUI SFI tab

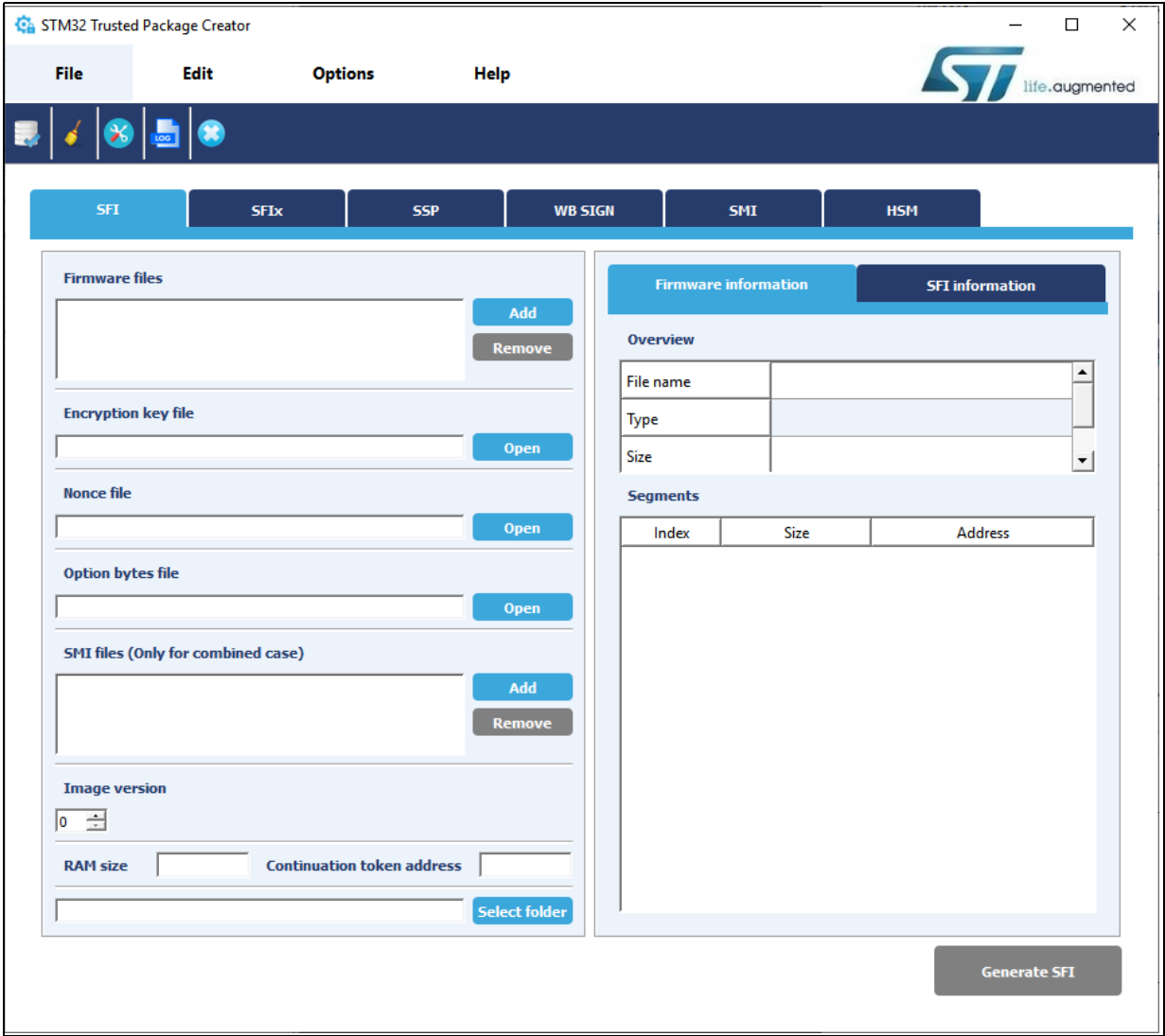


Figure 16. STM32 Trusted Package Creator tool GUI SMI tab

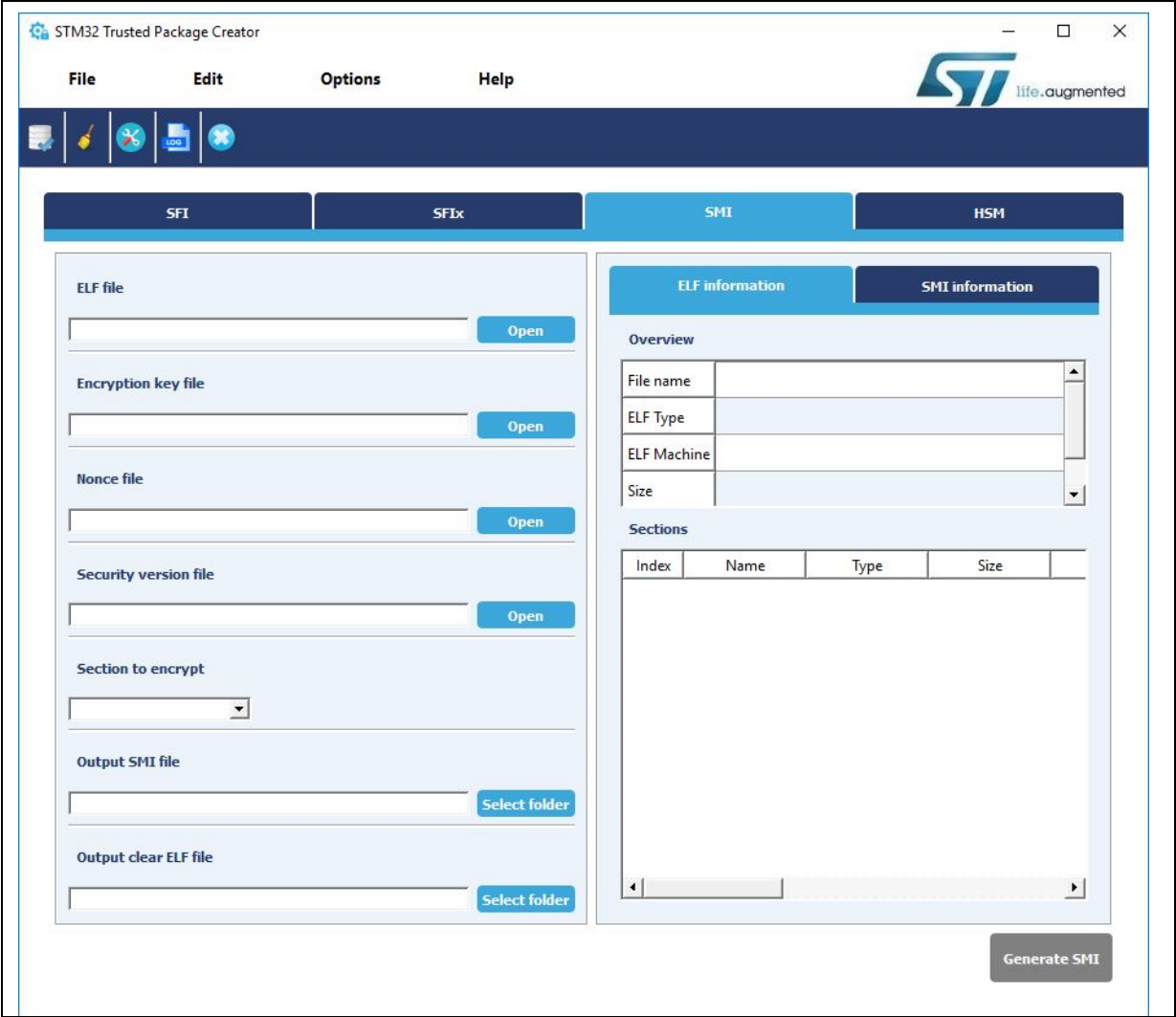
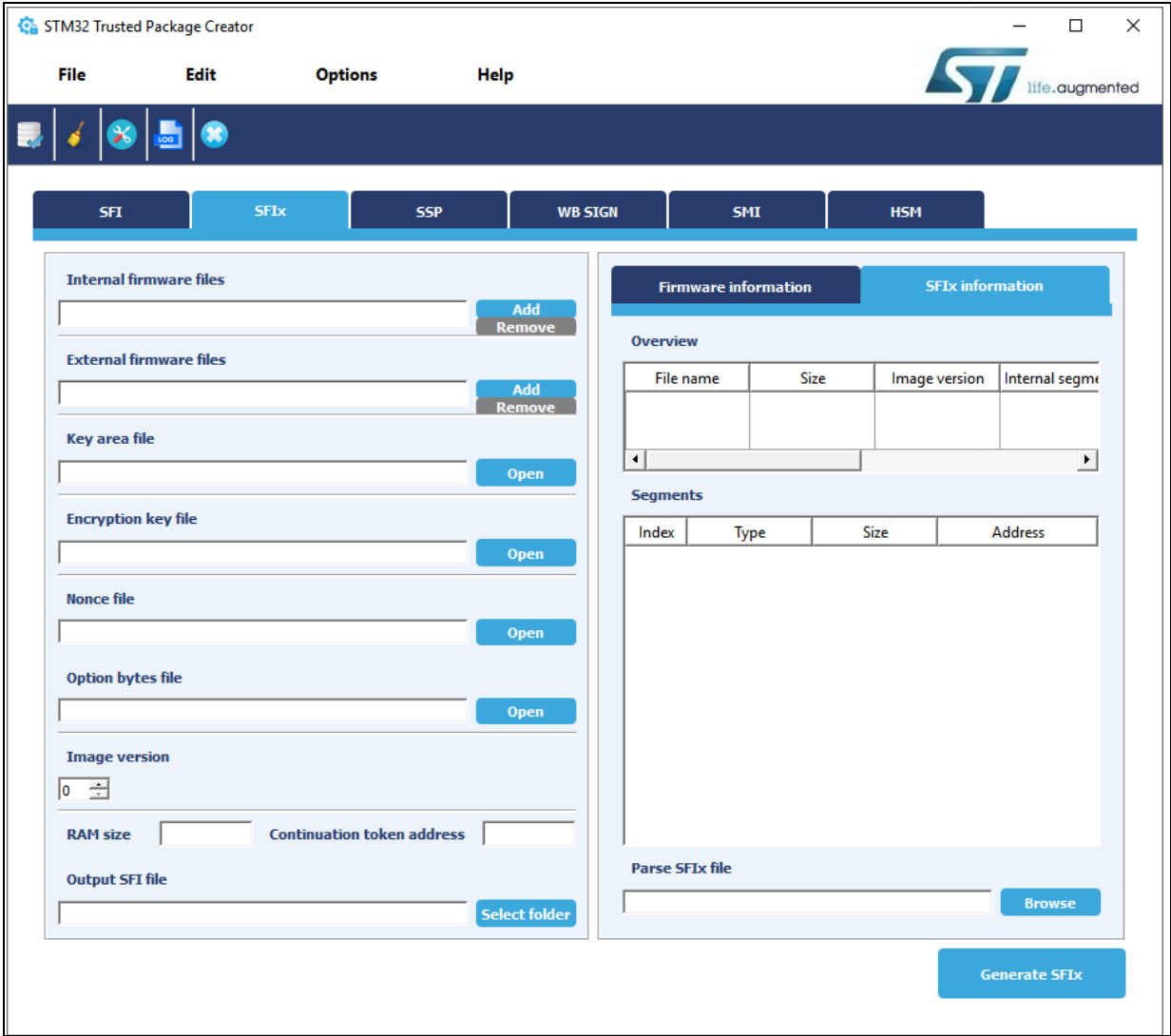


Figure 17. STM32 Trusted Package Creator tool GUI SFIx tab



4.1 SFI generation

To validate the SFI generation request, the user has to fill in the input fields with valid values:

Firmware files:

The user must add the input firmware files with the **add** button.

Note: If the file is valid, it is added in the firmware files list. Selecting it makes several pieces of related information appear in the Firmware information section (Figure 18), otherwise an error message box is shown saying either the file could not be opened or the file is not valid. If the file is in a binary format, a dialog box appears requesting that an address be provided. A file can be removed with the **remove** button.

Figure 18. Firmware file addition

The screenshot shows the STM32 Trusted Package Creator GUI. The 'SFI' tab is selected. On the left, there are several sections for file selection: 'Firmware files' (with 'sys_bl.bin' and 'Add/Remove' buttons), 'Encryption key file' (with a path and 'Open' button), 'Nonce file' (with a path and 'Open' button), 'Option bytes file' (with a path and 'Open' button), and 'SMI files (Only for combined case)' (with 'Add/Remove' buttons). Below these are 'Image version' (set to 1), 'RAM size' (set to 800), 'Continuation token address' (set to 0x08070000), and 'Output SFI file' (with a path and 'Select folder' button). On the right, the 'Firmware information' section shows an 'Overview' of 'out.sfi' (6 KB, Protocol version 01) and a 'Segments' table.

Index	Type	Size	Address
1	Firmware	742 B	0x8000000
2	Pause	32 B	0x8070000
3	Resume	32 B	0x8070000
4	Firmware	742 B	0x80002e6
5	Pause	32 B	0x8070020
6	Resume	32 B	0x8070020
7	Firmware	742 B	0x80005cc
8	Pause	32 B	0x8070040
9	Resume	32 B	0x8070040
10	Firmware	742 B	0x80008b2

A 'Generate SFI' button is located at the bottom right of the 'Firmware information' section.

Encryption key and nonce file:

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).

Option bytes file:

The option bytes file can be selected the same way. Only csv files are supported.

SMI files:

SMI files can be added the same way as the firmware files. Selecting a file makes several pieces of related information appear in the Firmware information section.

Image version:

Image version value in [0..255].

RAM size:

Size of RAM memory available for SFI programming.

Continuation token address:

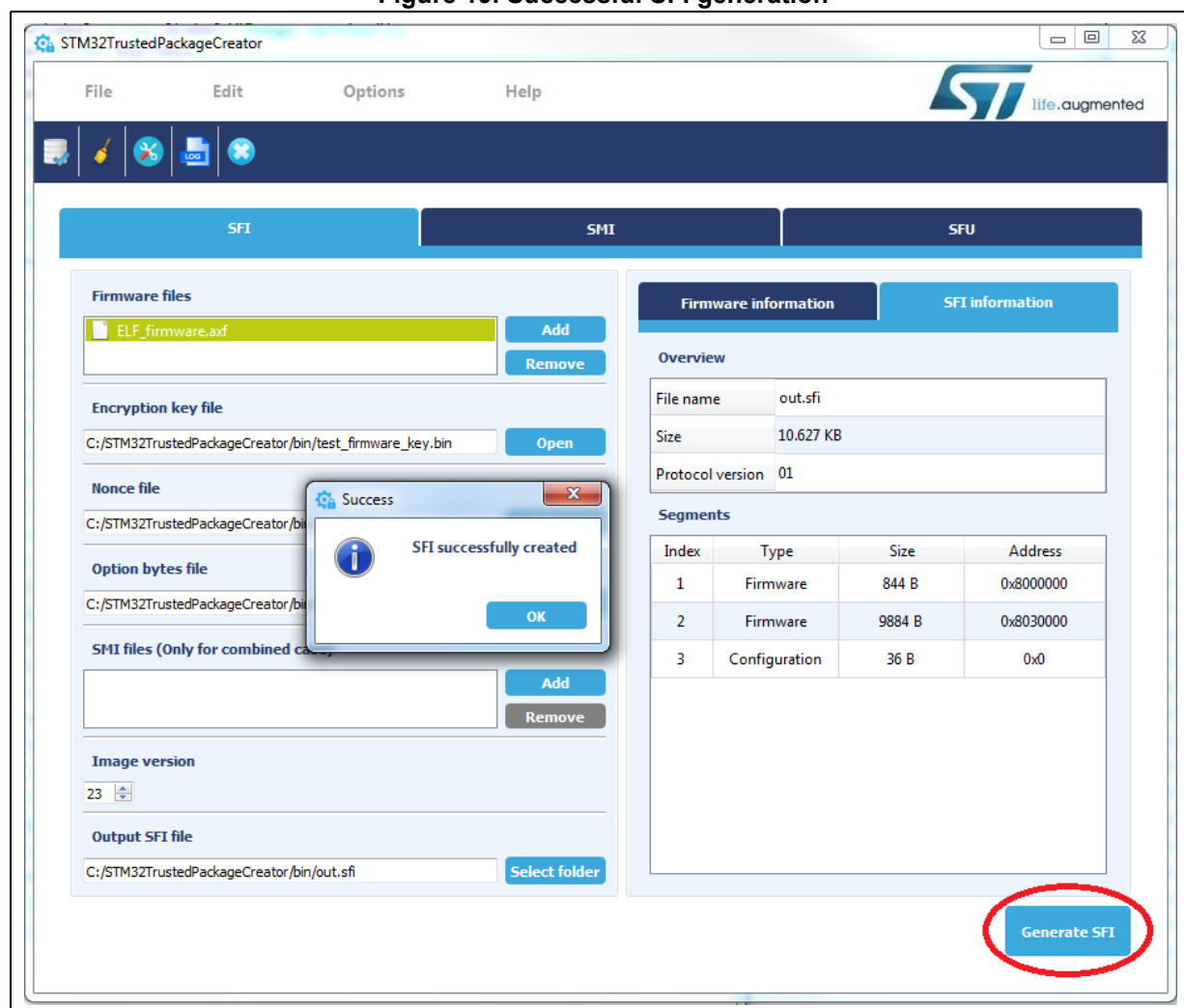
Address at which to store the Continuation token. The address must not collide with other areas.

Output file:

An output file can be selected by entering its path (absolute or relative), or with the **select folder** button, note that with the latter way, a name *out.sfi* is suggested, you can keep or change it

When all fields are properly filled in, the **Generate SFI** button becomes active. The user may generate the SFI file by clicking on it.

If everything goes well, a message box indicating successful generation appears ([Figure 19](#)) and information about the generated SFI file is displayed in the SFI information section.

Figure 19. Successful SFI generation

4.2 SMI generation

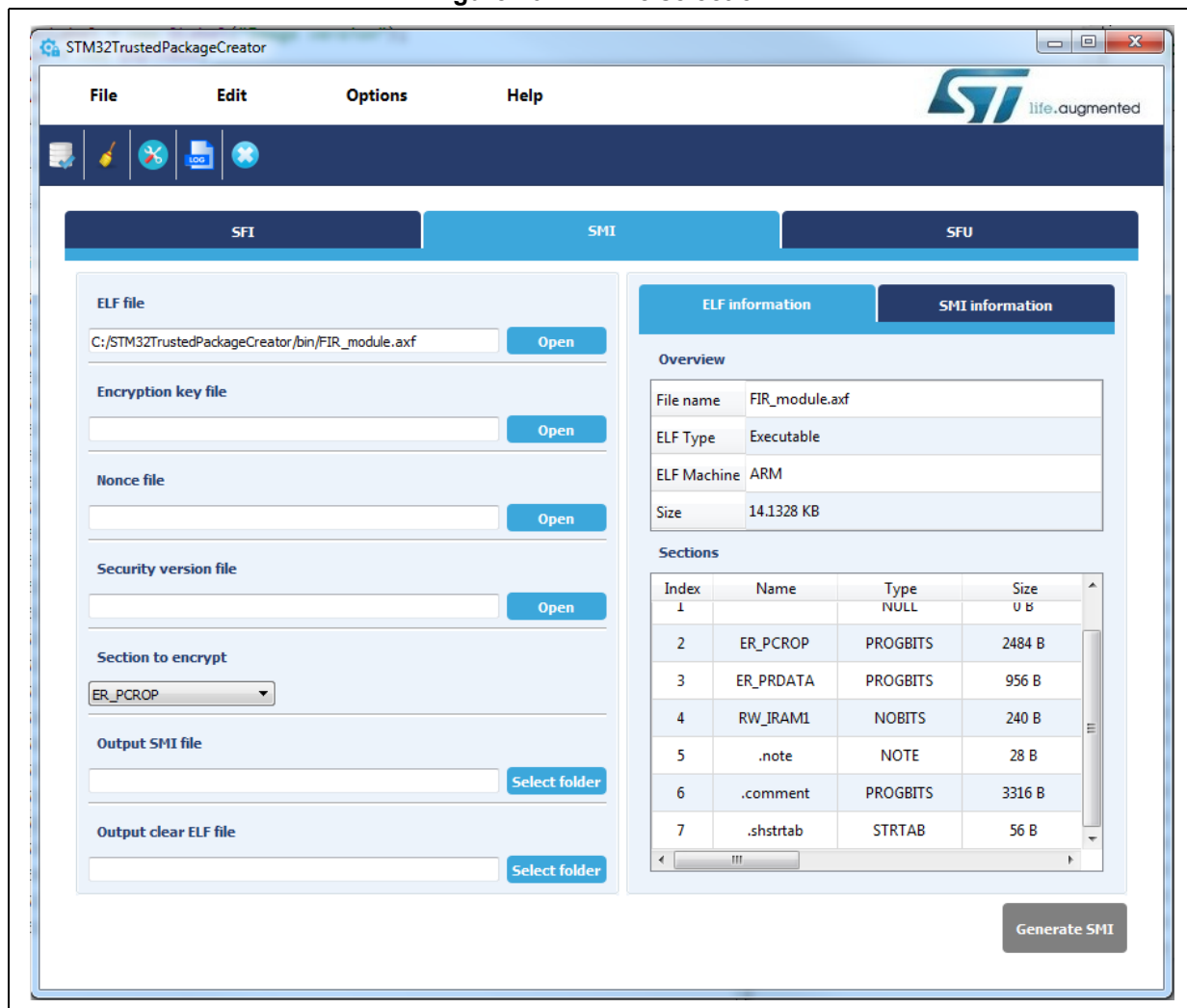
As for SFI generation, the user must provide the input information.

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 ([Figure 20](#)), otherwise an error message box is shown saying either the file could not be opened or that the file is not valid.

Figure 20. ELF file selection



Encryption key and nonce file:

As for SFI, the Encryption key and nonce file can be selected in the same way as the firmware file. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).

Security version file:

The security version file size must be 16 bytes.

A security version file is provided under the *Security_Version* folder.

Section:

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

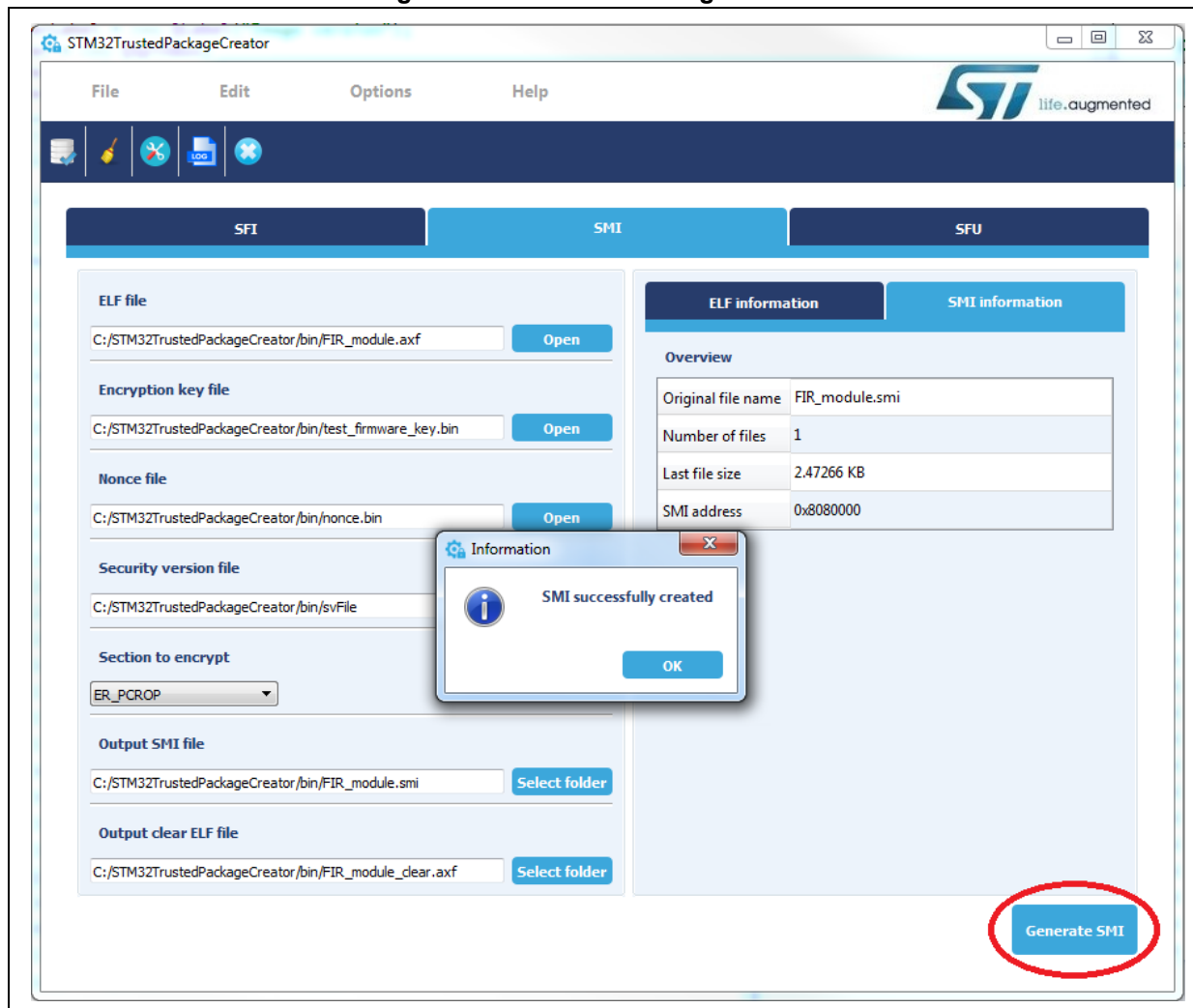
Output files:

Output files can be selected by entering their paths (absolute or relative), or with the **Select folder** button, note that with the latter way a name is suggested, which can be kept or changed.

When all fields are properly filled in, the user may generate the SFI file by a click on the **Generate SFI** button (the button becomes active).

A message box informing the user that generation was successful appears ([Figure 21](#)), in addition of information about the generated SMI file, otherwise an error is displayed.

Figure 21. Successful SMI generation



4.3 SFlx generation

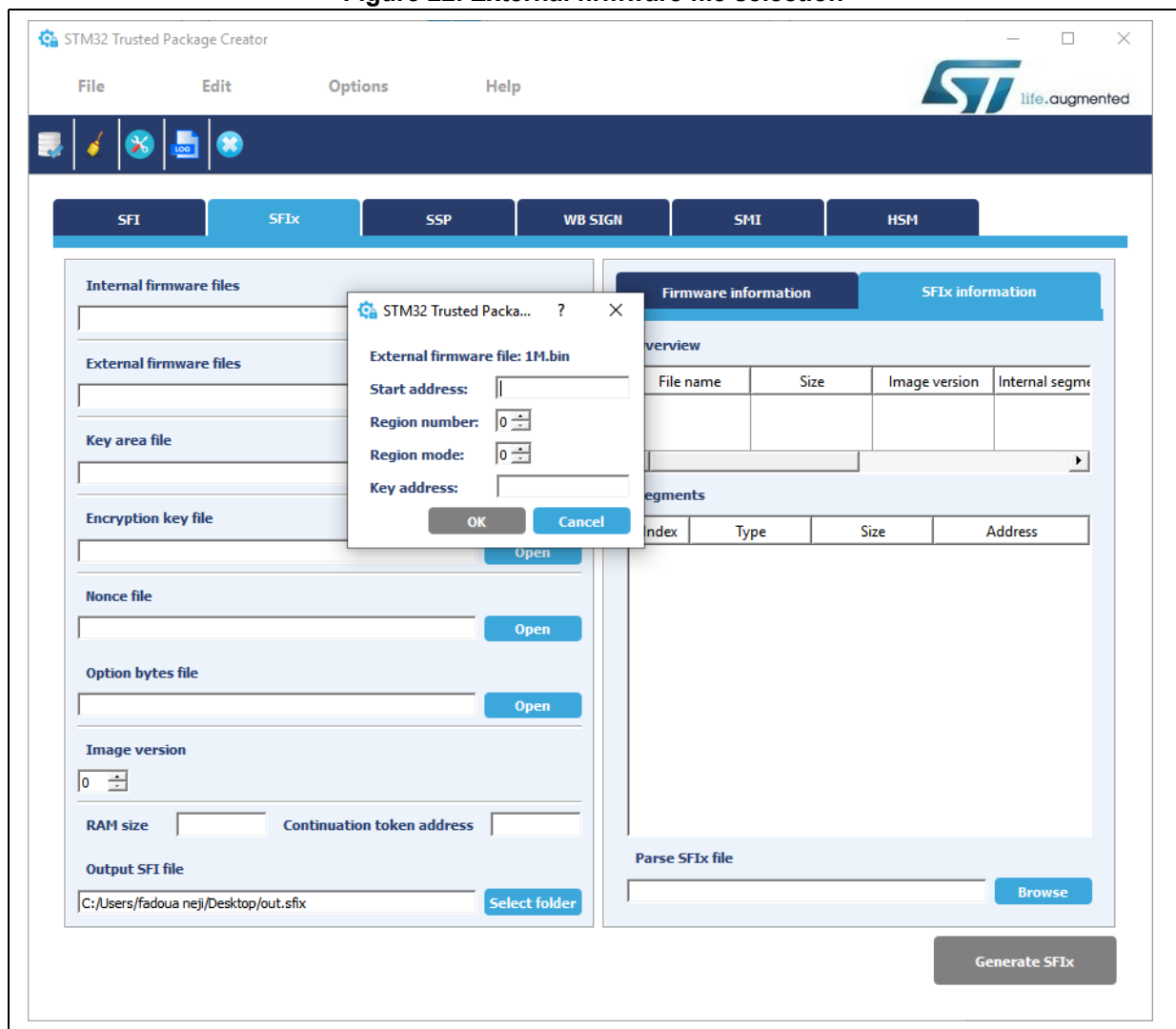
In addition to the SFI generation use case, the user needs to provide the input for external firmware files.

External firmware files:

The user needs to enter the input for external firmware files. If the file is valid, several pieces of related information appear in the Firmware information section. A popup also appears that contains fields to be filled-in ([Figure 22](#)).

- Start address: the start address of a binary
- Region number: the OTFDEC configuration detailed in [Section 2.2: SFlx preparation process](#).
- Region mode: the OTFDEC configuration detailed in [Section 2.2: SFlx preparation process](#).
- Key address: the address of the key.

Figure 22. External firmware file selection



Output file:

Output files can be selected by entering their paths (absolute or relative), or with the **select folder** button. Note that in the latter case, a name is suggested, which can be kept or changed.

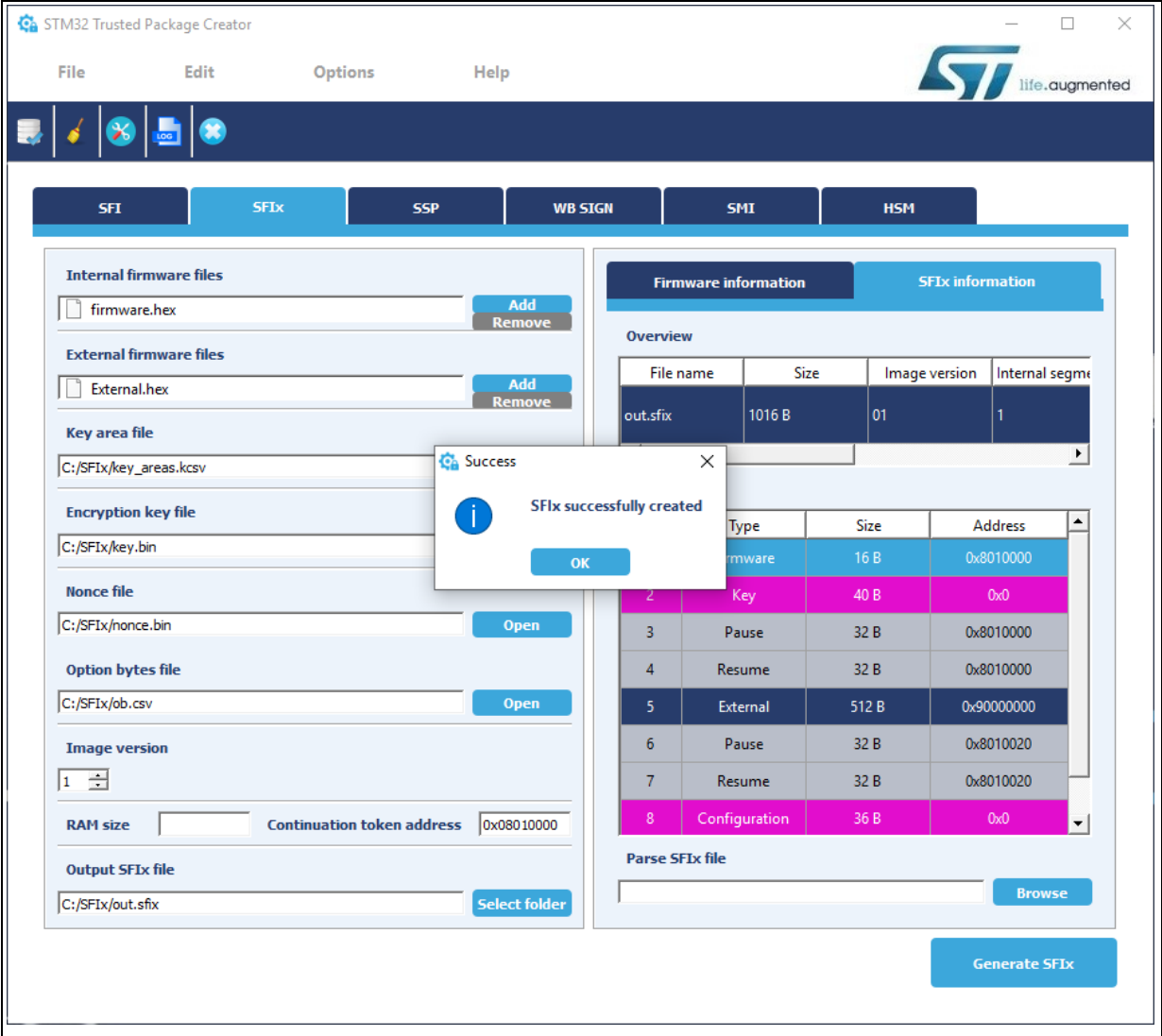
When all fields are properly filled in, the **Generate SFlx** button becomes active. The user may generate the SFlx file by clicking on it.

If the generation is successful, a confirmatory message box appears, and information about the generated SFlx file is displayed in the SFlx information section.

The SFlx information section (*Figure 25*), contains two sections:

- An overview section mentioning the name of the output file and various information including the size of the file, it's image version, internal/external segments and so on.
- A 'segments' section showing the different areas.
- A 'parse SFlx file' section, allowing the user to browse a *.sfix* output and parse it to display SFlx information

Figure 23. Successful SFIx generation



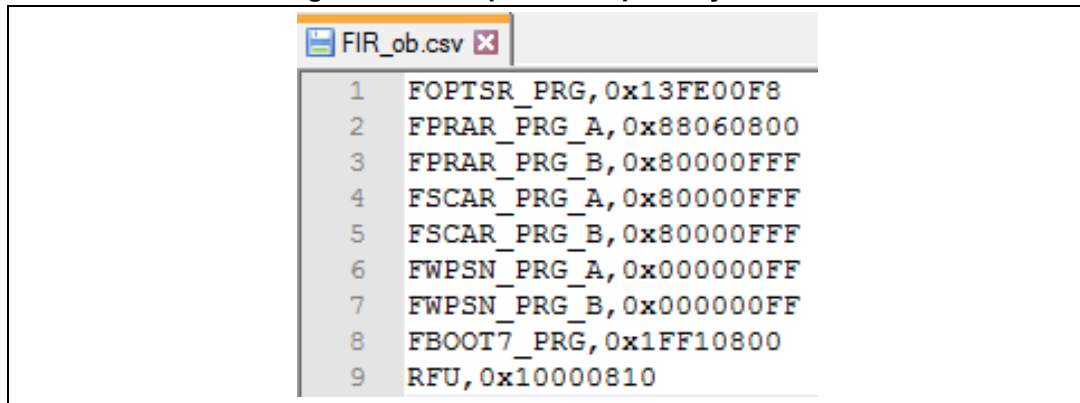
5 Option bytes file

The option bytes file field is mandatory for SFI applications only, it allows option bytes to be programmed during secure firmware install.

Only CSV (Comma Separated Value) format is supported for such files, it is composed of two vectors: a register name and its value.

All of the 9 option-byte registers must be configured (a total of 9 lines in a csv file).

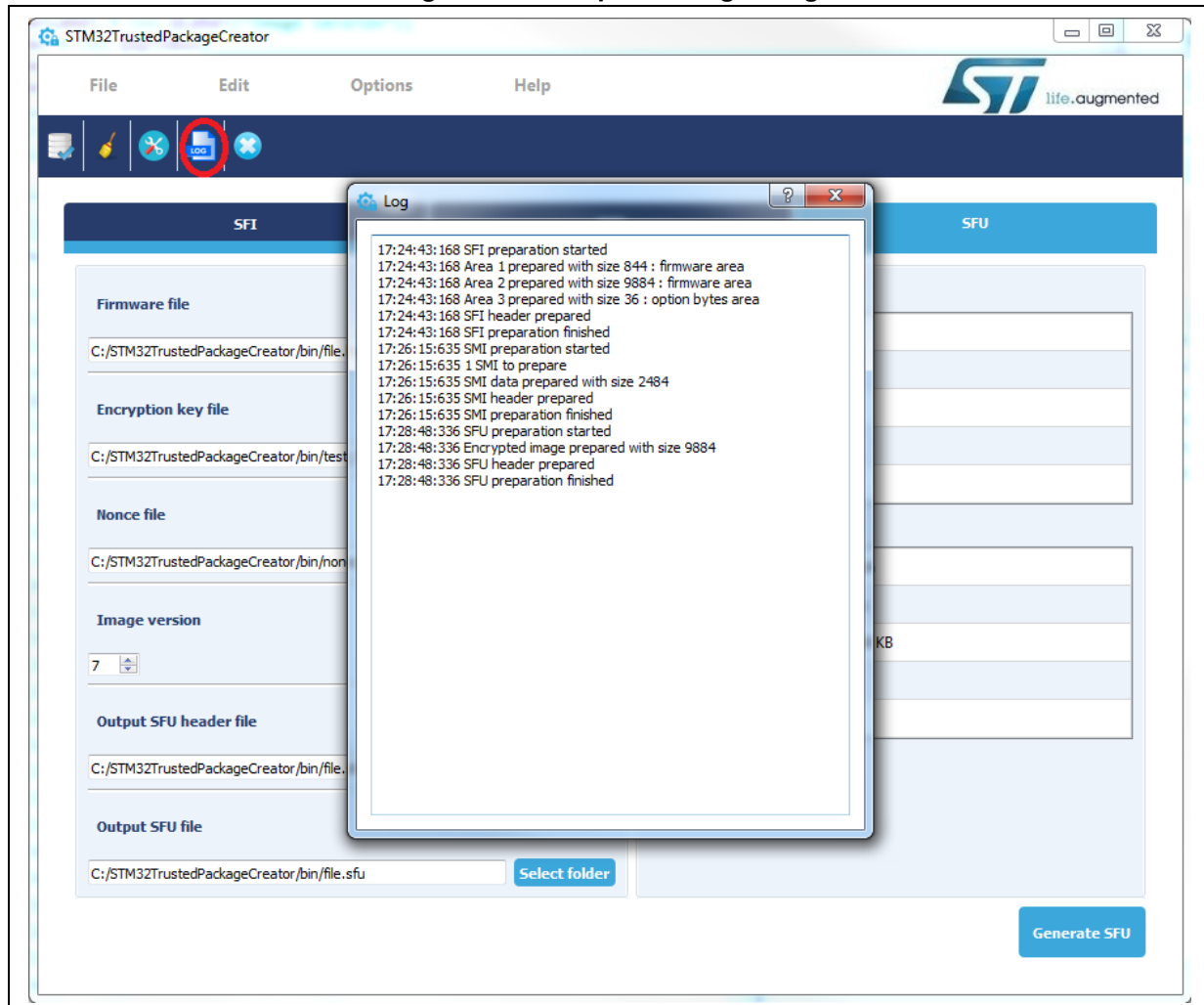
Figure 24. Example of an option bytes file



6 Log dialog

A log can be visualized by clicking the **log** button in the tools bar or in the menu bar:
Options-> log.

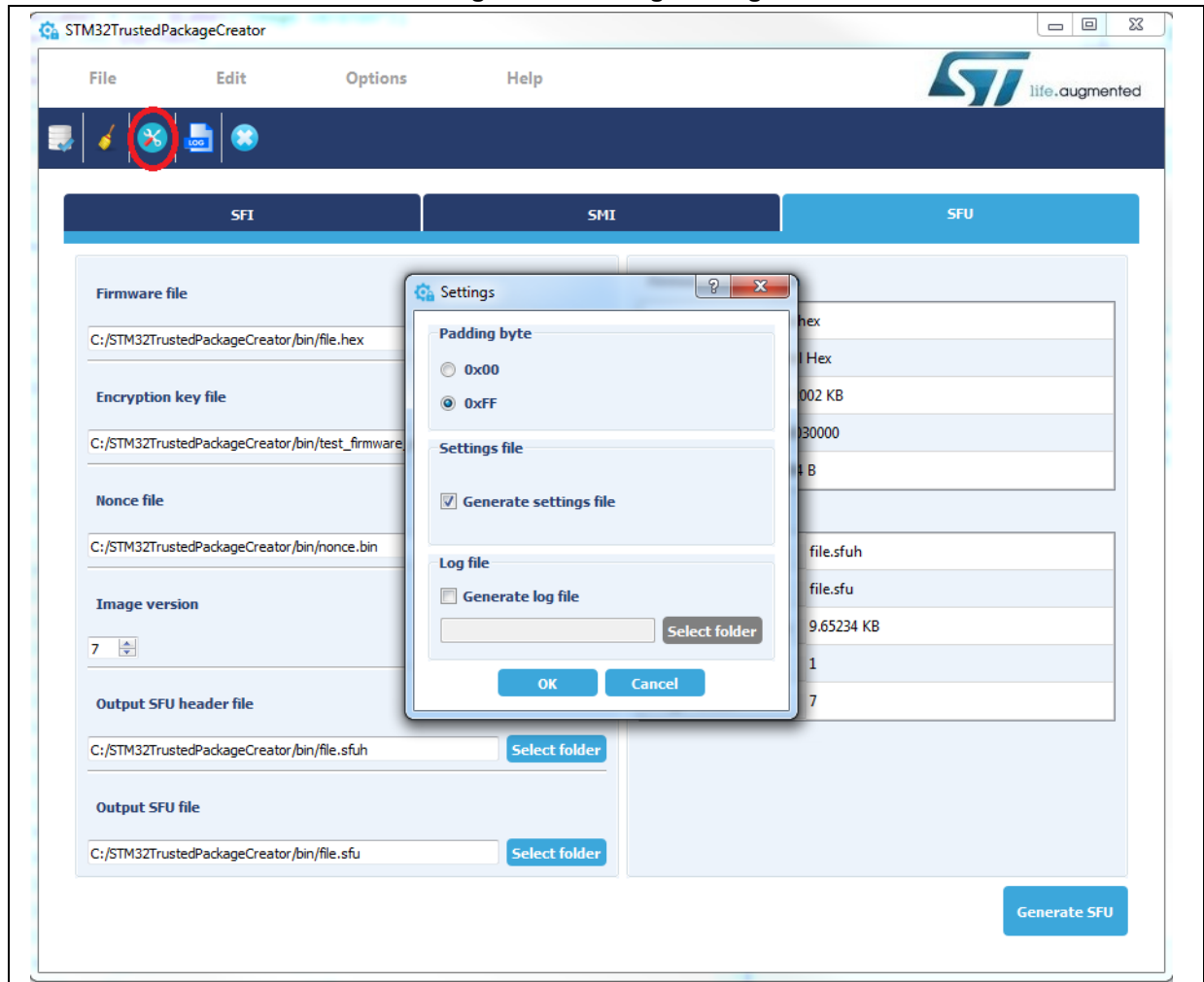
Figure 25. Example of a log dialog



7 Settings

The settings dialog can be accessed by clicking the **settings** button in the tools bar or in the menu bar: Options -> settings.

Figure 26. Settings dialog



Padding byte:

When parsing files, padding may be added to fill the gap between segments separated by 16 bytes or less, in order to merge them and reduce the number of segments. The user may have the choice between 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 field contents.

Log file:

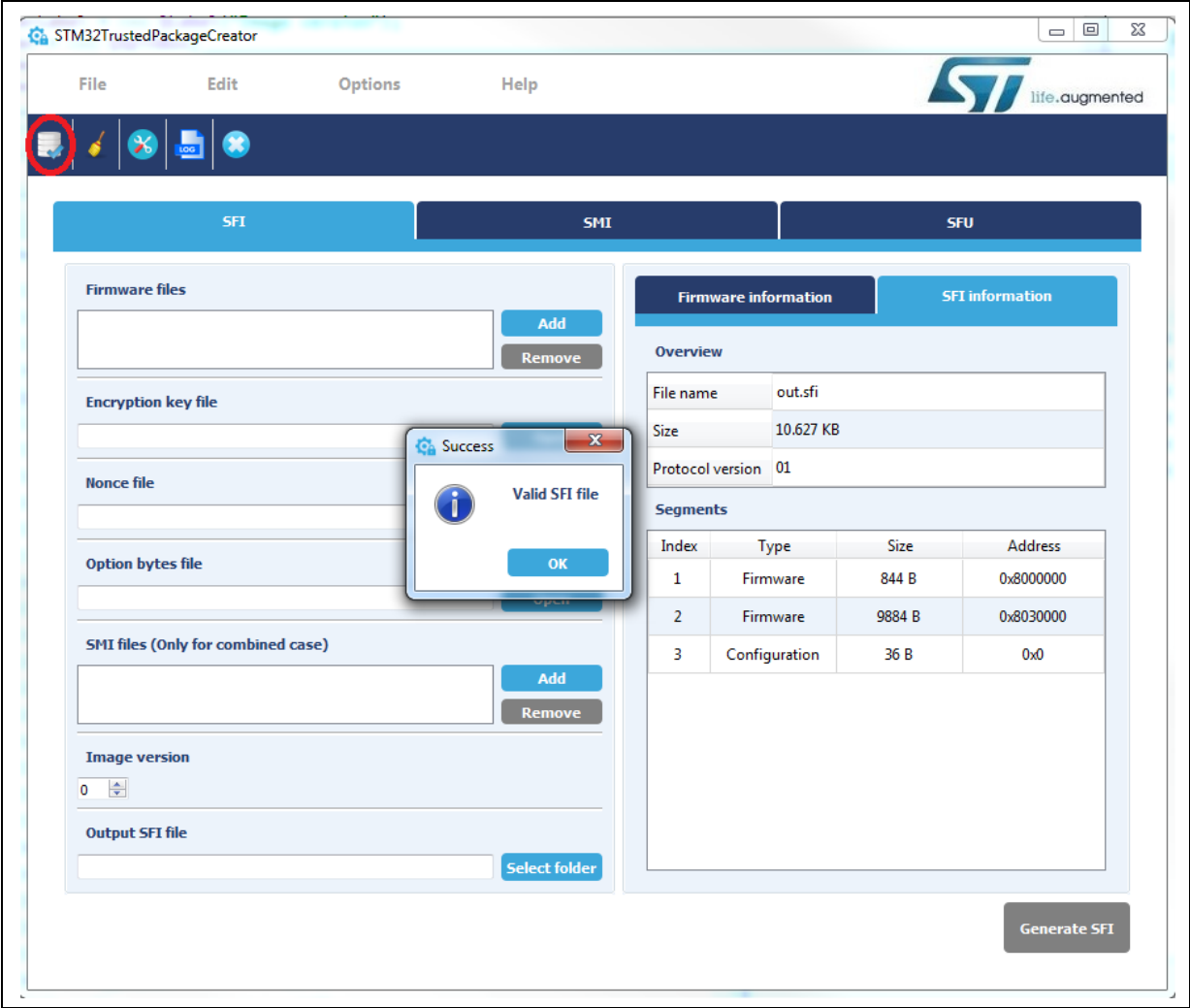
When checked, a log file is generated in the selected path.

8 SFI/SMI checking

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

This allows SFI or SMI file validity to be checked, in addition to displaying information about it.

Figure 27. SFI checking



9 Reference documents

Table 1. Document references

ID	Revision	Title
[1]	Latest version	STM32CubeProgrammer software description. User manual STMicroelectronics, UM2237

10 Revision history

Table 2. Document revision history

Date	Revision	Changes
20-Dec-2017	1	Initial release.
07-Mar-2019	2	Added Section 3.5: STM32WB firmware signing . Updated: – Section 4.1: SFI generation . – Figure 8: STM32 Trusted Package Creator tool's available commands (part 1) – Figure 15: STM32 Trusted Package Creator tool GUI SFI tab – Figure 18: Firmware file addition
16-Apr-2019	3	Updated Section 1: System requirements Added Section 2.4: HSM preparation process . Updated Figure 15: STM32 Trusted Package Creator tool GUI SFI tab .
15-Oct-2019	4	Updated: – Section 2.4.1: Why we need an HSM – Section 2.4.2: HSM programming – Figure 6: HSM programming tab – Section 2.4.3: Reading HSM information – Figure 7: Reading HSM information – Figure 8: STM32 Trusted Package Creator tool's available commands (part 1) and Figure 9: STM32 Trusted Package Creator tool's available commands (part 2) – Figure 15: STM32 Trusted Package Creator tool GUI SFI tab – Section 3.2: HSM provisioning command Added Section 3.3: SFI/SFIx generation command .

Table 2. Document revision history

Date	Revision	Changes
07-Jan-2020	5	<p>Added:</p> <ul style="list-style-type: none">– Section 2.2: SFlx preparation process <p>Updated:</p> <ul style="list-style-type: none">– Figure 8: STM32 Trusted Package Creator tool's available commands (part 1)– Figure 9: STM32 Trusted Package Creator tool's available commands (part 2)– Section 3.3: SFI/SFlx generation command– Figure 15: STM32 Trusted Package Creator tool GUI SFI tab– Figure 17: STM32 Trusted Package Creator tool GUI SFlx tab (changed image and title) <p>Replaced Section 4.3 SFU generation with Section 4.3: SFlx generation.</p>
25-Feb-2020	6	<p>Updated:</p> <ul style="list-style-type: none">– Introduction– Section 3.2: HSM provisioning command. <p>Removed sections:</p> <ul style="list-style-type: none">– SFU preparation process– SFU generation command.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

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

© 2020 STMicroelectronics – All rights reserved