AN5054
Application note

How to perform secure programming using STM32CubeProgrammer

Introduction

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

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

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

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

Refer also to:

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

Contents

List of tables

List of figures

1 General information

1.1 Licensing information

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

1.2 Acronyms and abbreviations

Table 1. List of abbreviations

Abbreviations	Definition
AES	Advanced encryption standard
CLI	Command-line interface
CM	Contract manufacturer
GCM	Galois counter mode (one of the modes of AES)
GUI	Graphical user interface
HSM	Hardware security module (such as STM32HSM)
HW	Hardware
MAC	Message authentication code
MCU	Microcontroller unit
OEM	Original equipment manufacturer
OTP	One-time programmable
PCROP	Proprietary code read-out protection
PI	Position independent
ROP	Read-out protection
RSS	Root security service (secure)
RSSe	Root security service extension
SFI	Secure (internal) firmware install
SFIx	Secure external firmware install
SSP	Secure secret provisioning
STPC	STM32 Trusted Package Creator
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 Encrypted firmware (SFI) preparation using the STPC tool

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

2.1 System requirements

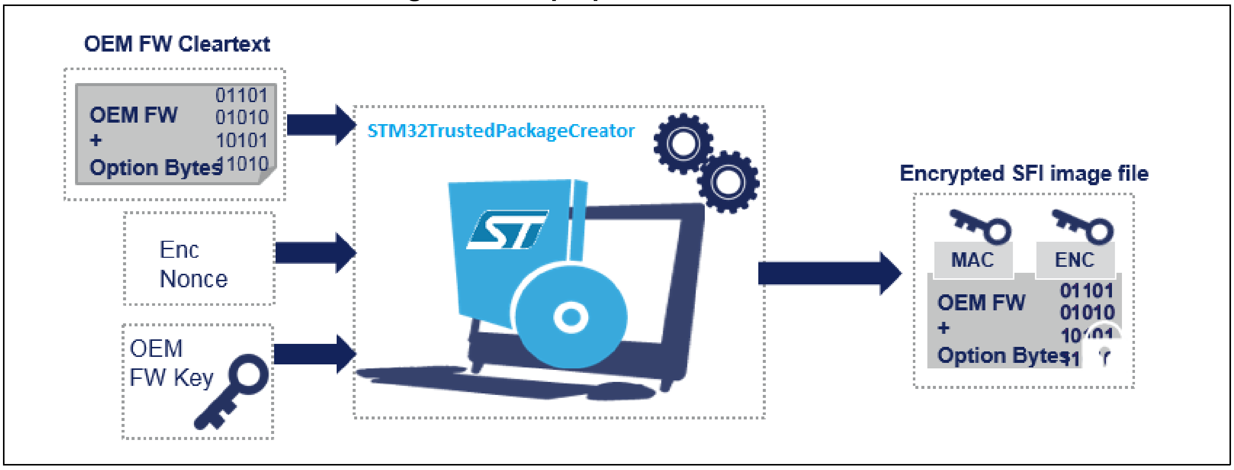
Using the STM32 Trusted Package Creator tool for SFI/SFIx and SSP image generation requires a PC running on either Windows®[a], Linux®[b] Ubuntu®[c] or Fedora®[d], or macOS®[e].

Note: *Refer to [4] or [5] for the supported operating systems and architectures.*

2.2 SFI generation process

The SFI format is an encryption format for internal firmware created by STMicroelectronics that transforms internal firmware (in ELF, Hex, Bin, or Srec formats) into encrypted and authenticated firmware in a SFI format using the AES-GCM algorithm with a 128-bit key. The SFI preparation process used in the STM32 Trusted Package Creator tool is described in *Figure 1*.

Figure 1. SFI preparation mechanism



a. Windows is a trademark of the Microsoft group of companies.

b. Linux® is a registered trademark of Linus Torvalds.

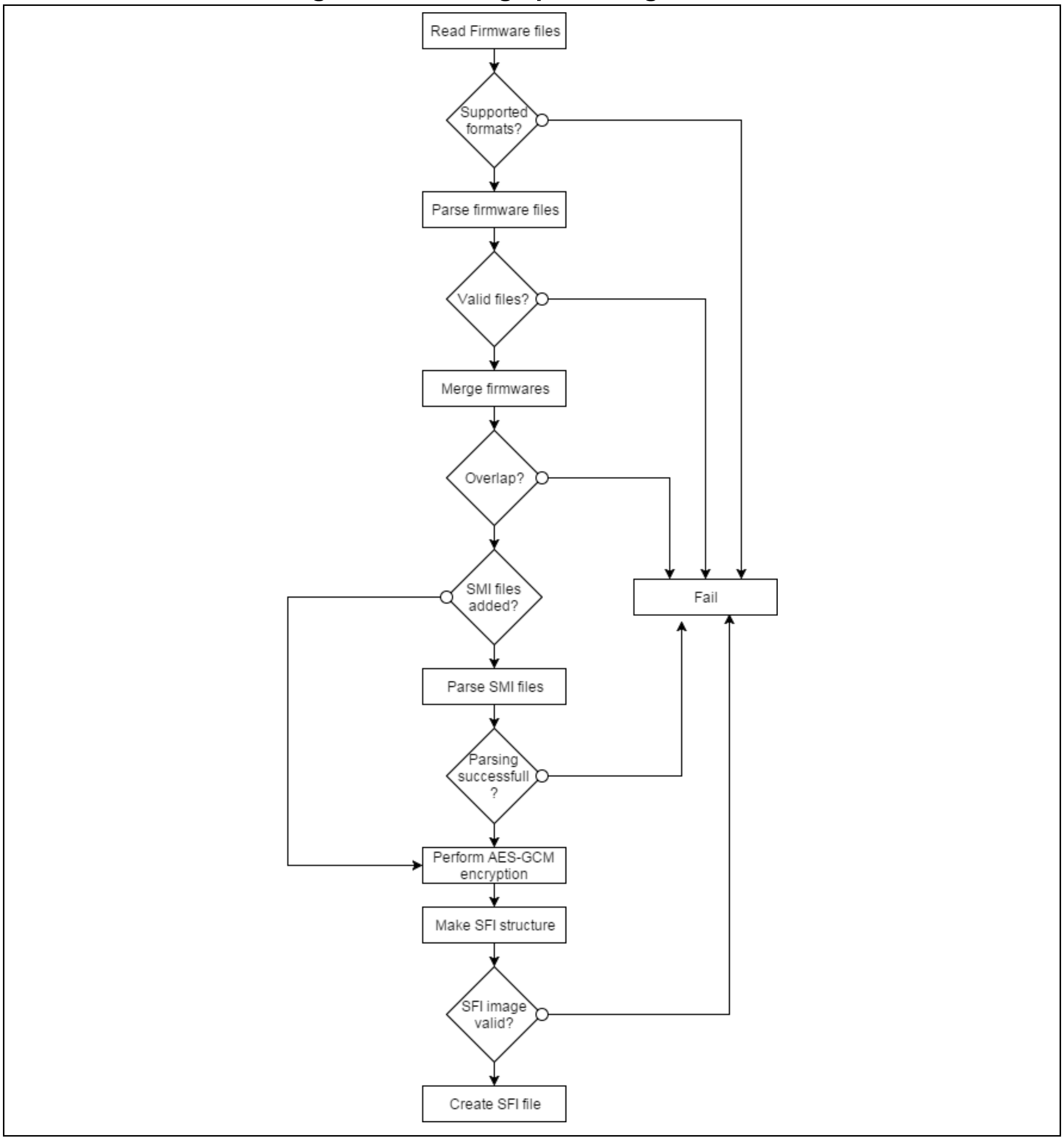
c. Ubuntu® is a registered trademark of Canonical Ltd.

d. Fedora® is a trademark of Red Hat, Inc.

e. macOS® is a trademark of Apple Inc., registered in the U.S. and other countries and regions.

The SFI generation steps as currently implemented in the tool are described in *Figure 2*.

Figure 2. SFI image process generation



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

IV = nonce + area index

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

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

These areas can be of different types:

- 'F' for a firmware area (a regular segment in the input firmware)
- 'C' for a configuration area (used for option-byte configuration)
- 'P' for a pause area
- 'R' for a resume area.

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

The STM32 Trusted Package Creator overview below (*Figure 3*) shows the 'RAM size' input as well as the 'Continuation token address' input, which is used to store states in flash memory during SFI programming.

Figure 3. RAM size and CT address inputs used for SFI

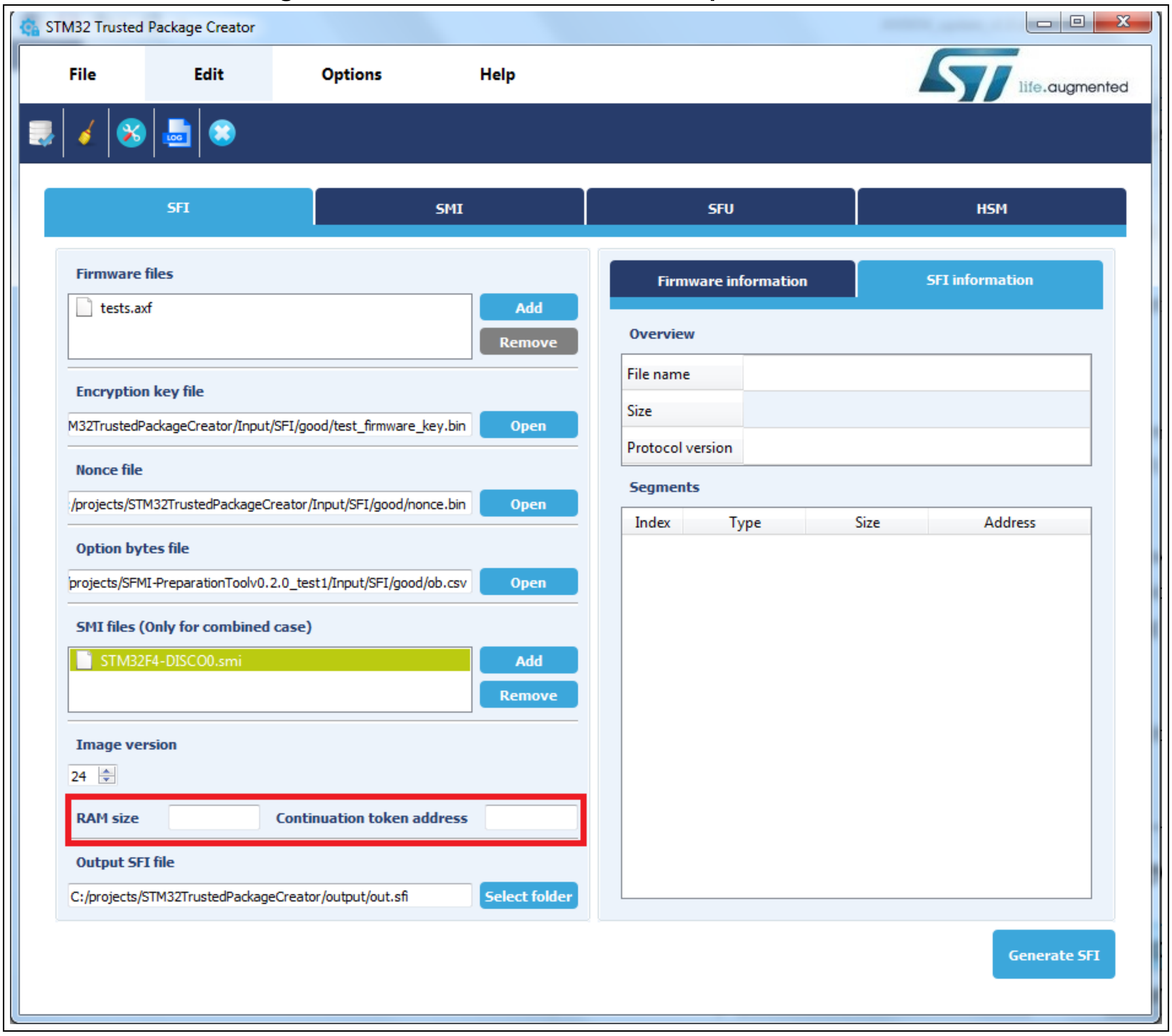
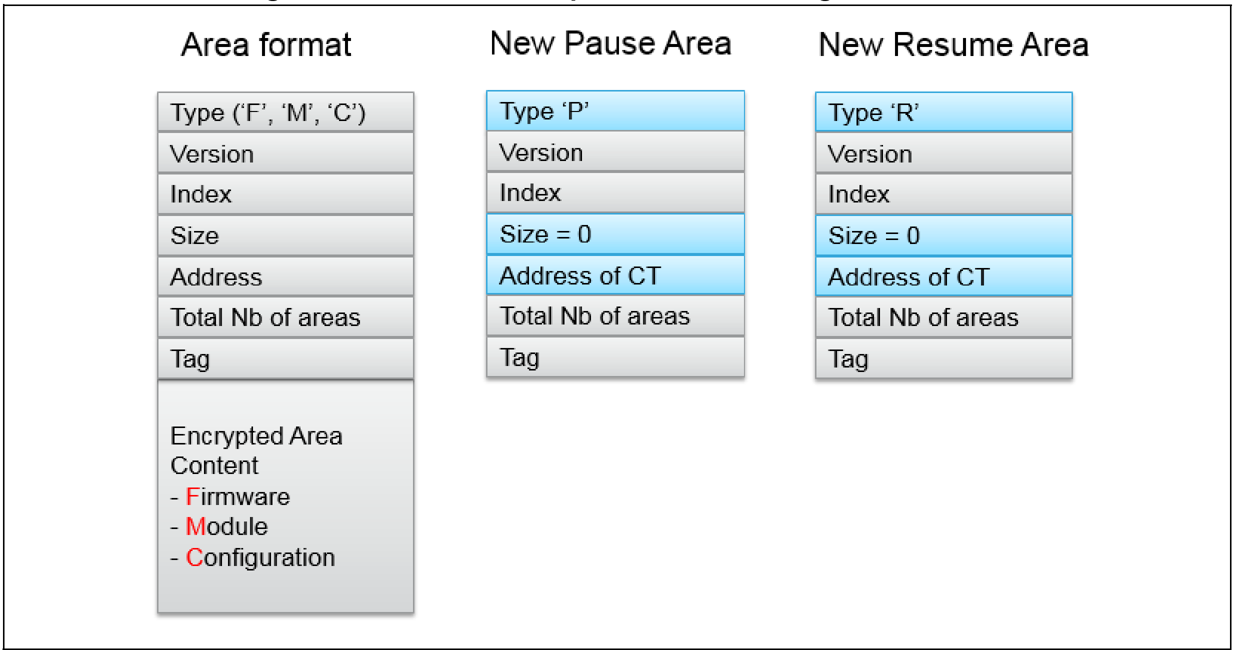


Figure 4 shows the specifics of these new areas compared to a regular SFI area.

Figure 4. 'P' and ‘R’ area specifics versus a regular SFI area

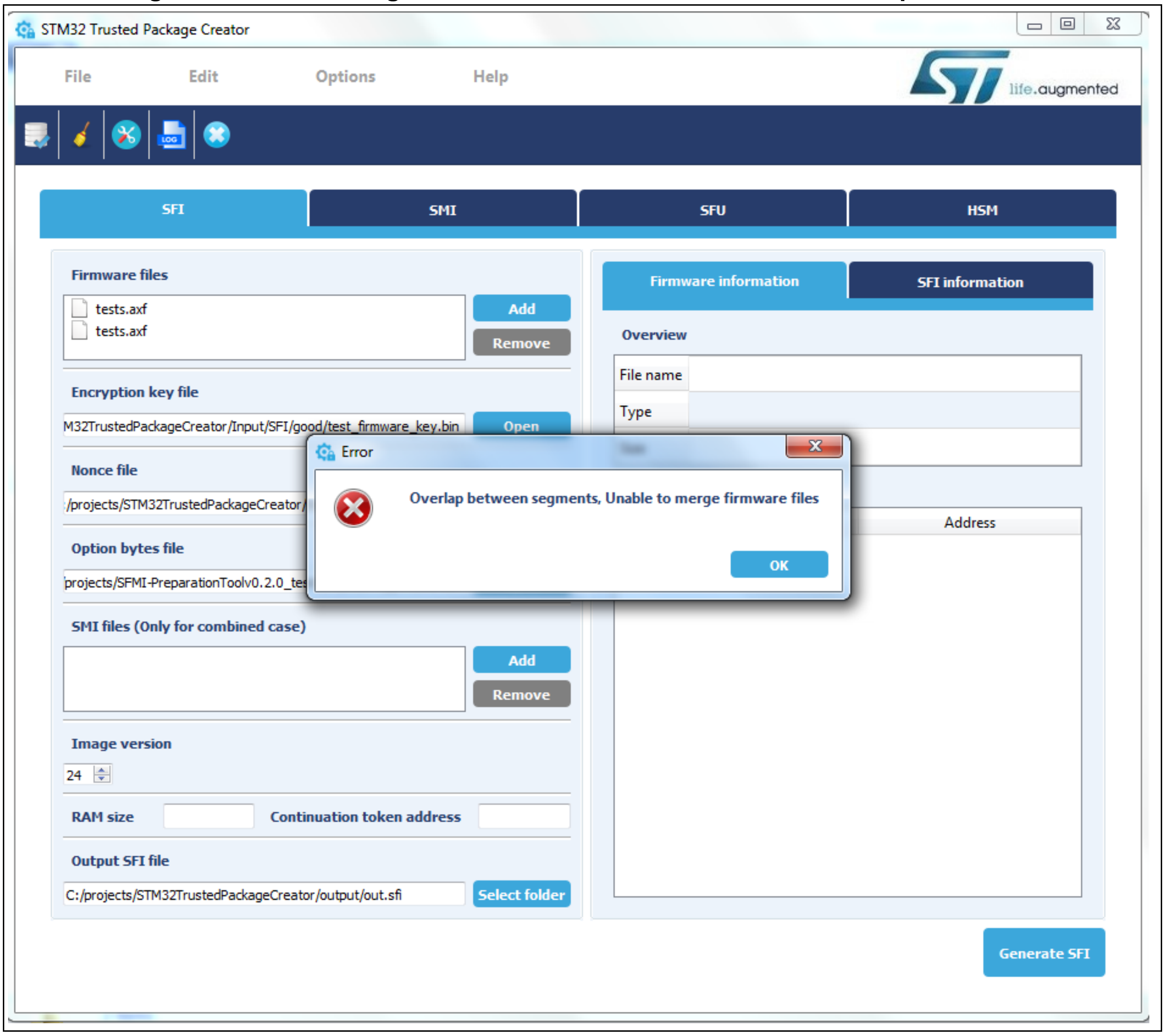


A top-level image header is generated and then authenticated.The tool performs AES-GCM with authentication only (without encryption), using the SFI image header as an AAD, and the nonce as IV.

An authentication tag is generated as output.

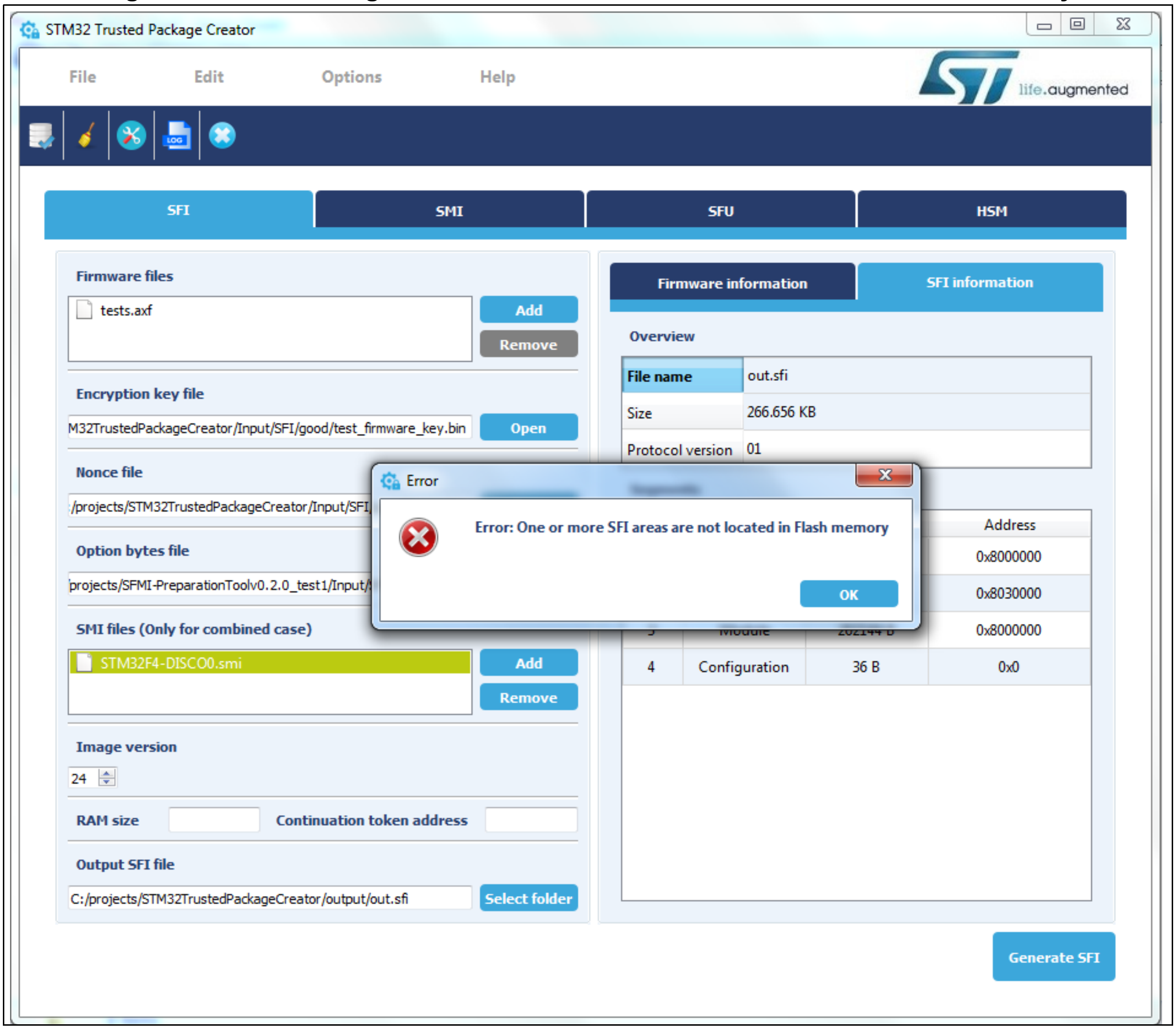
Note: To prepare an SFI image from multiple firmware files, make sure that there is no overlap between their segments, otherwise an error message appears (Figure 5).

Figure 5. Error message when firmware files with address overlaps are used



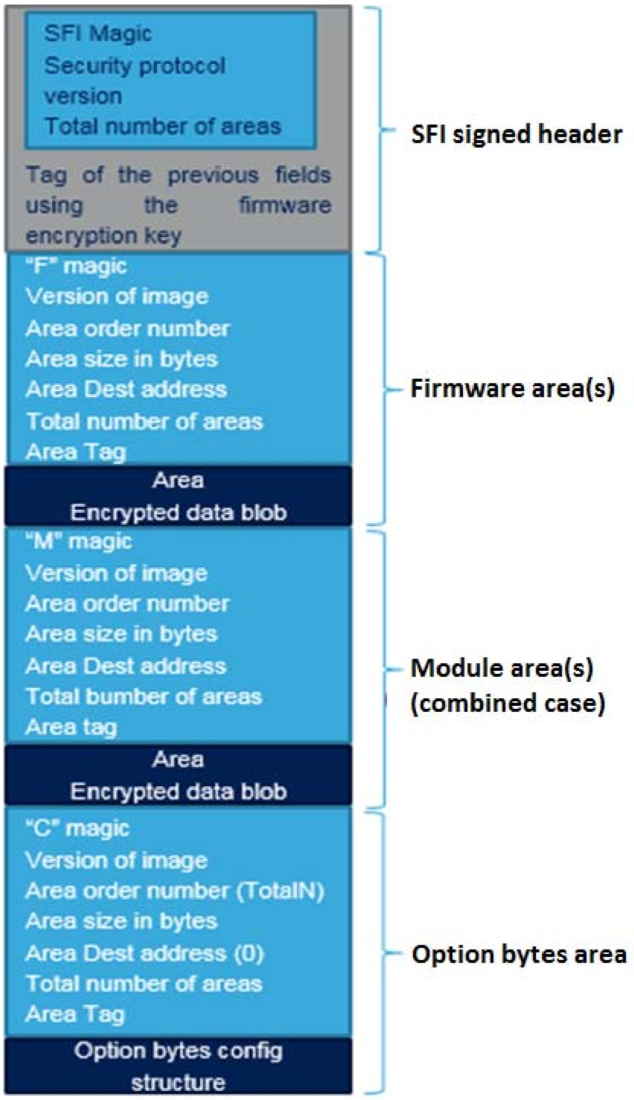
Also, all SFI areas must be located in flash memory, otherwise the generation fails, and the following error message appears (*Figure 6*).

Figure 6. Error message when a SFI area address is not located in flash memory



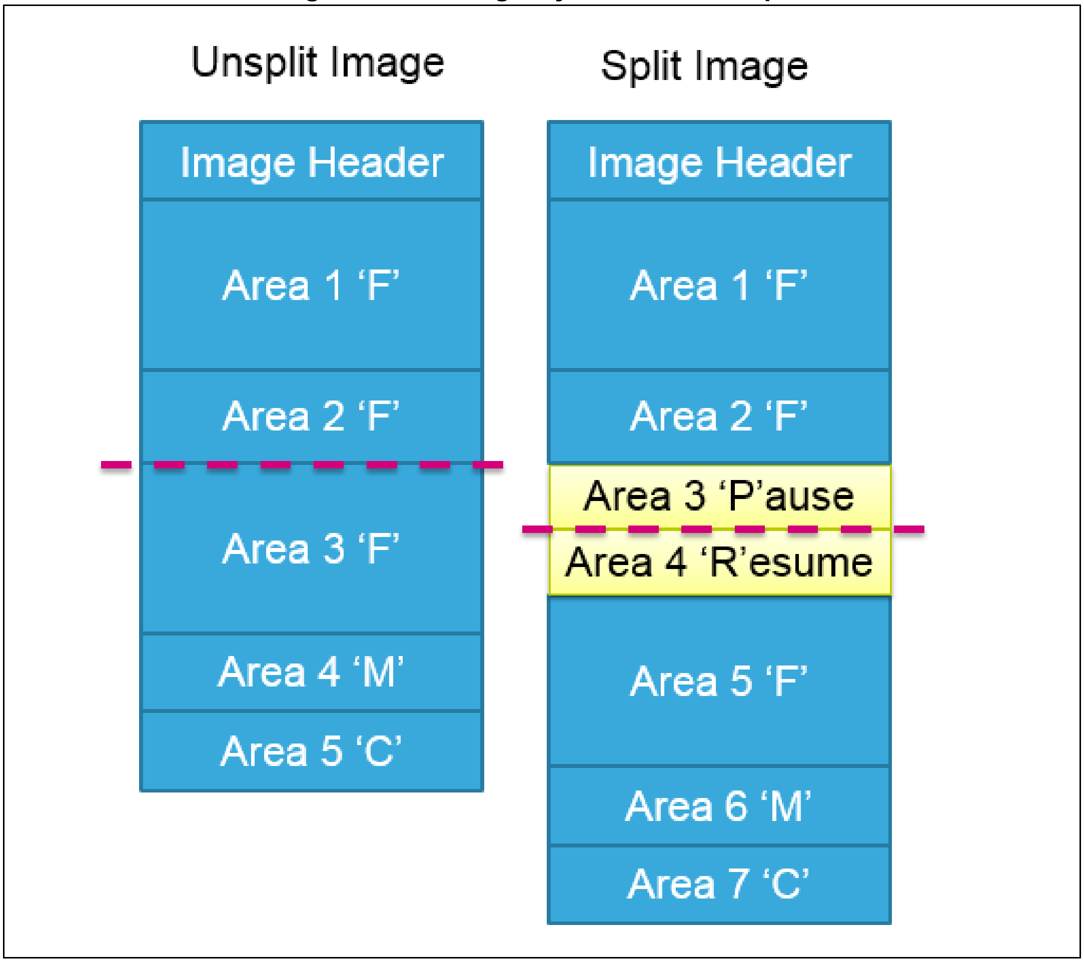
The final output from this generation process is a single file, which is the encrypted and authenticated firmware in “.sfi” format. The SFI format layout is described in *Figure 7*.

Figure 7. SFI format layout



When the SFI image is split during generation, areas 'P' and 'R' appear in the SFI image layout, as in the following example *Figure 8*.

Figure 8. SFI image layout in case of split



2.3 SFIx generation process

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

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

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

Area E

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

- OTFD region_number (uint32_t):
  - 0…3: OTFD1 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, STM32L5, and STM32U5)
  - 4…7: OTFD2 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333, STM32H725/335, and STM32U5)
- OTFD region_mode (uint32_t) bit [1:0]:
  - 00: instruction only AES-CTR)
  - 01: data only (AES-CTR)
  - 10: instruction + data (AES-CTR)
  - 11: instruction only (EnhancedCipher)
- OTFD key_address in internal flash memory (uint32_t).

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

Area K

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

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

Example of an area 'K':

```
0x00000010, 0x0C020000
0x00000010, 0x08000060
```

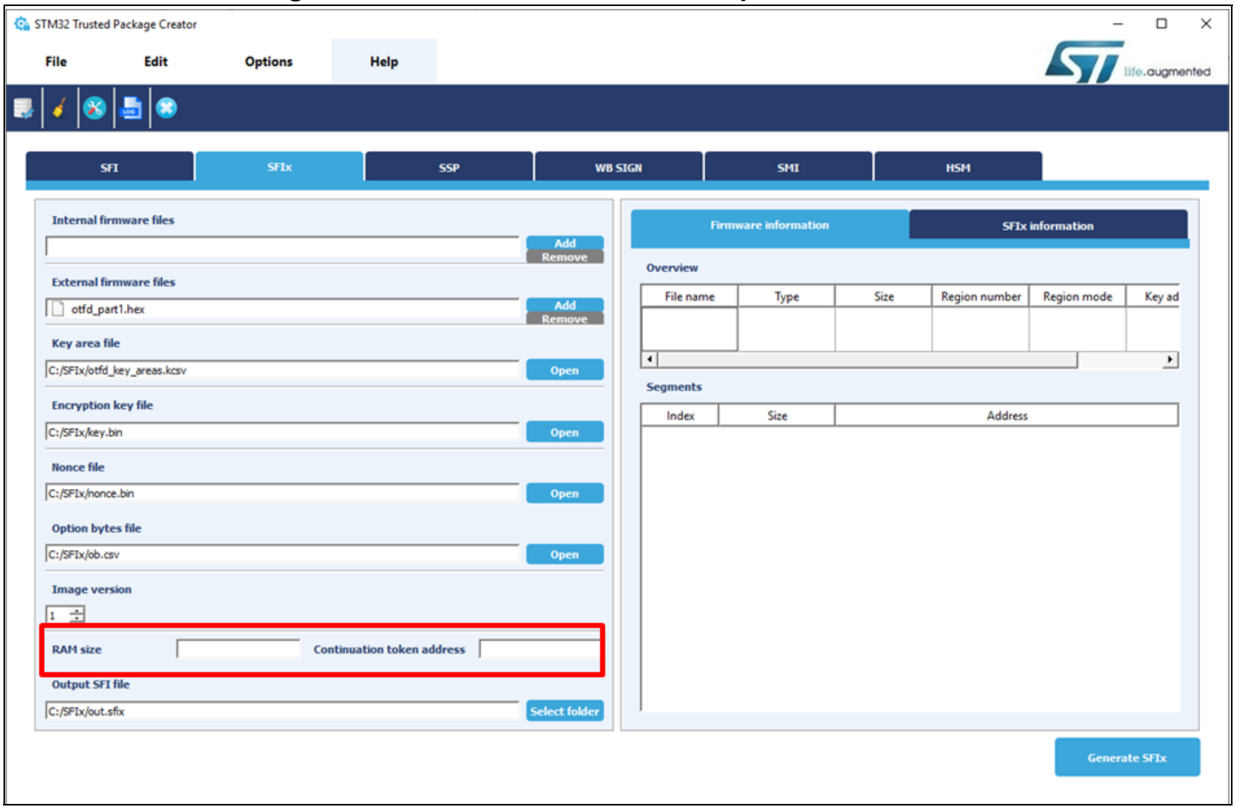
There are two key areas:

- 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).

The STM32 Trusted Package Creator overview (*Figure 9*) shows the RAM size input for SFIx image generation, and also the 'Continuation token address' input, which is used by SFIx to store states in external/internal flash memory during SFIx programming.

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

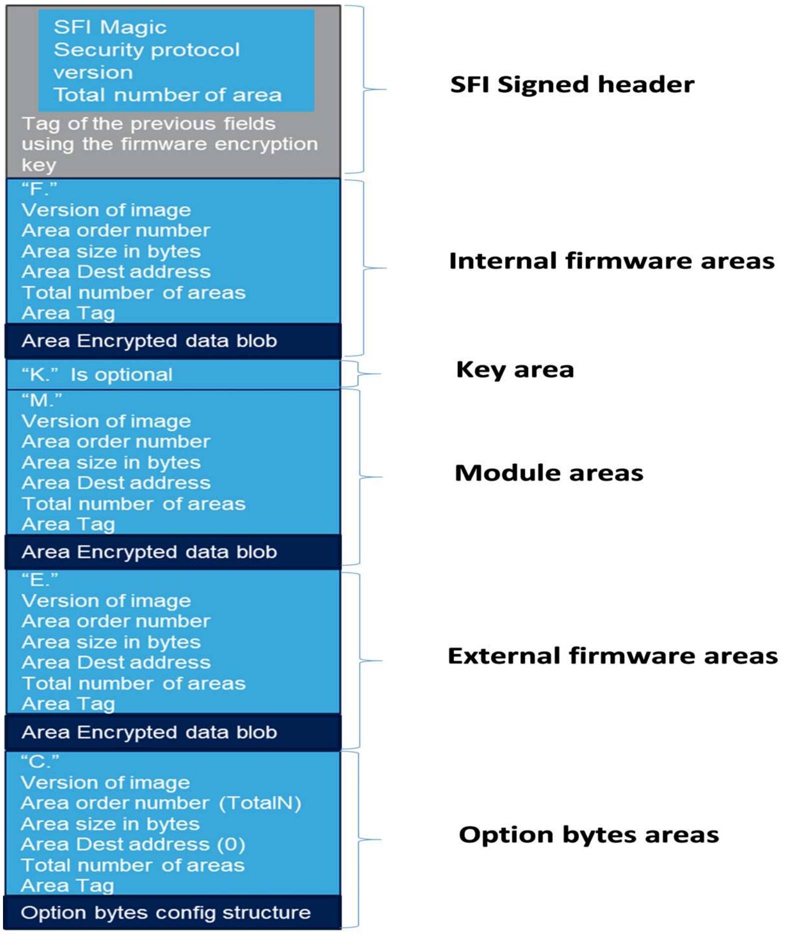
Figure 9. RAM size and CT address inputs used for SFIx



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

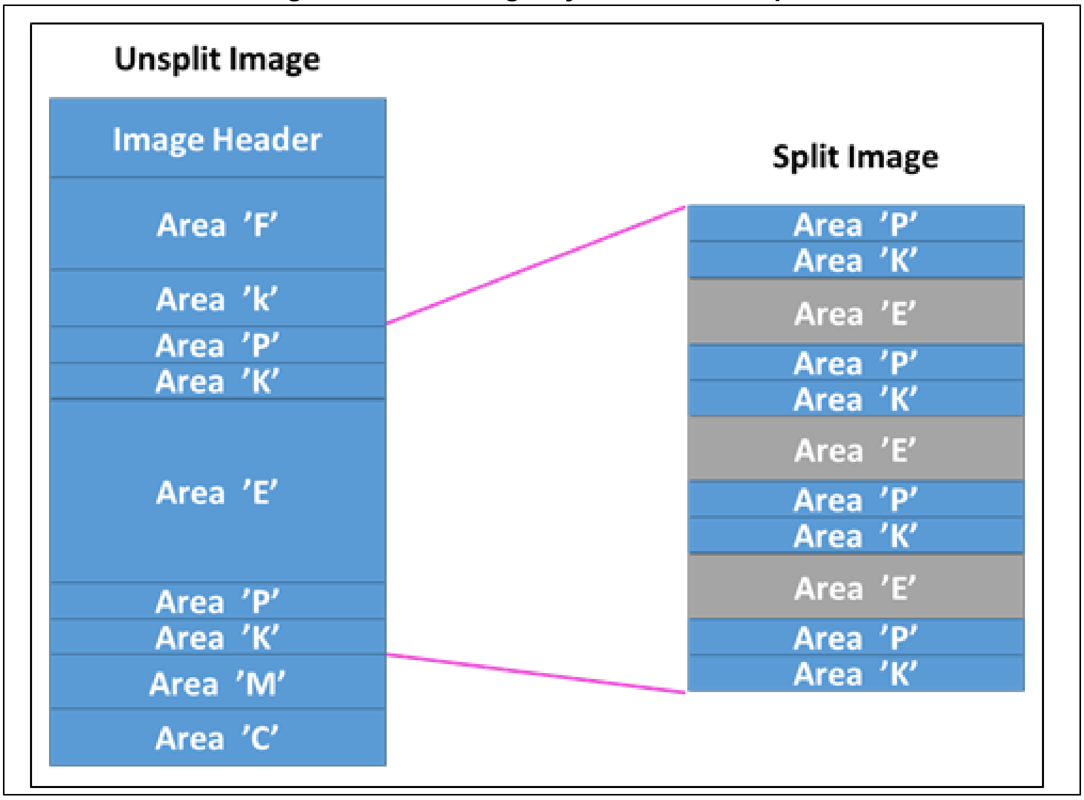
The final output from this generation process is a single file, which is the encrypted and authenticated internal/external firmware in ".sfix" format. The SFIx format layout is described in *Figure 10*.

Figure 10. SFIx format layout



When the SFIx image is split during generation, the areas ‘P’ and ‘R’ appear in the SFIx image layout, as in the example below *Figure 11*.

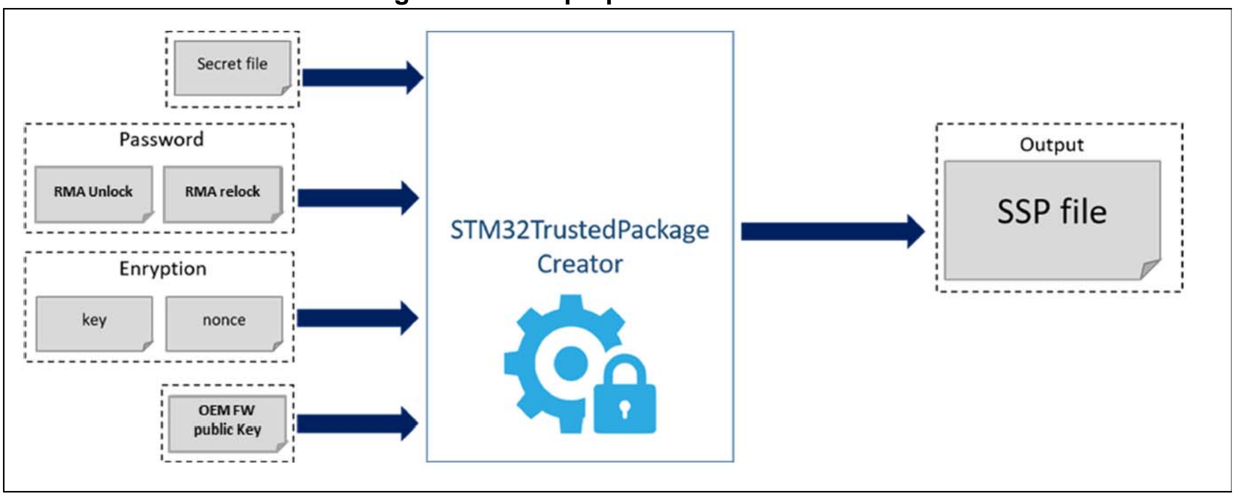
Figure 11. SFIx image layout in case of split.



2.4 SSP generation process

SSP is an encryption format that transforms customer secret files into encrypted and authenticated firmware using an AES-GCM algorithm with a 128-bit key. The SSP preparation process used in the STM32 Trusted Package Creator tool is shown in *Figure 12*.

Figure 12. SSP preparation mechanism



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

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

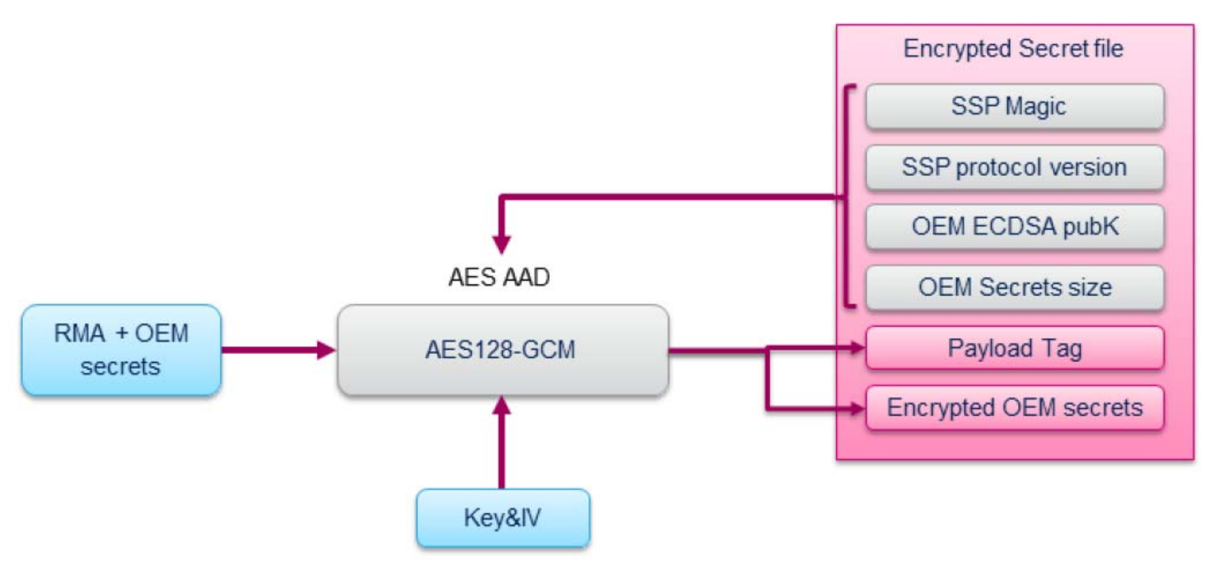
The first layout part (SSP magic, protocol version, ECDSA public key, secret size) is used as additional authenticated data (AAD) to generate the payload tag. This is checked by the ROM code during decryption.

Table 2. SSP preparation inputs

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

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

Figure 13. Encryption file scheme



2.5 STM32 Trusted Package Creator tool in the command-line interface

This section describes how to use the STM32 Trusted Package Creator tool from the command-line interface to generate SFI/SFIx image. The available commands are listed in *Figure 14*.

Figure 14. STM32 Trusted Package Creator tool - SFI preparation options

```
SFI preparation options

 -sfi,   --sfi          : Generate SFI image,
                          You also need to provide the information listed below
     -devid,   --deviceid  : Add an input device ID
        <device_id>        : Device ID value of the concerning product
     -fir,    --firmware   : Add an input firmware file
        <Firm_File>        : Supported firmware files are ELF HEX SREC BIN
     -firx,    --firmwx    : Add input for external firmware file
        <Firmx_File>       : Supported externalfirmware files are ELF HEX SREC BIN
        [<Address>]        : Only in case of BIN input file (in any base)
        [<Region_Number>]  : Value in any base, [0:3]: OTFD1 (STM32H7A / STM32L5), [4:7]: OTFD2 (STM32H7A/B case)
        [<Region_Mode>]    : Value 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>]    : Value 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
    -hash,    --hash       : Generate Hash for integrity check
      <Hash_Flag>          : Possible values         0 : Hash generation disabled
                           :                         1 : Hash generation enabled
                           : By default if this option is not present the Hash is disabled
    -obk,    --ob-keys     : Example: -obk input1.obk input2.obk input3.obk ...
        <obk_Files>          Supported SSFI file extension is .bin .ssfi
    -ssfi,    --ssfi       : Add SSFI module for STM32 devices which supports this security feature.
        <Ssfi_File>        : Supported SSFI file extension is .bin .ssfi
    -o,      --outfile     : Generated SFI/SFIx file
         <Output_File>     : SFI/SFIx file to be created with sfi/sfix extension
```

2.5.1 Steps for SFI generation (CLI)

To generate an SFI/SFIx image in CLI mode, the user must use the "-sfi, --sfi" command followed by the appropriate inputs. Inputs for the "sfi" command are:

-**devid**, **--deviceid**

Description: Specifies deviceID. If this option is not used, P and R areas are generated by default for all devices.

Syntax: -devid <device_id>

<device_id>: Device ID

-fir, --firmware

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

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

<Firmware_file>: Firmware file.

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

-firx, --firmwx

Description: Add an input for an external firmware file. Supported formats are Bin, Hex, Srec, and ELF. This option can be used more than once 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 (STM32H7A3/7B3, STM32H7B0, or STM32L5), [4:7]: OTFD2 (STM32H7A3/7B3 and STM32H7B0 case).

<Region_Mode>: Only in the 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: 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.

-v, --ver

Description: sets the image version.

Syntax: -v <Image_version>

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

-ob, --obfile

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

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

Example: for STM32H75x devices, nine option byte registers must be configured, and they correspond to a total of nine lines in the csv file (*Figure 15*).

Syntax: -ob <CSV_file>

<CSV_file >: A csv file with nine values.

Figure 15. Option bytes file example

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

-m, --module

-rs, --ramsize

Description: Defines the available ram size (in the case of SFI)

Syntax: -rs <Size>

< Size >: RAM available size in bytes

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

-ct, --token

Description: Continuation token address (in the case of SFI)

Syntax: -ct <Address>

< Address >: continuation token flash memory address

-o, --outfile

Description: Sets the output SFI file to be created.

Syntax: -o <out_file>

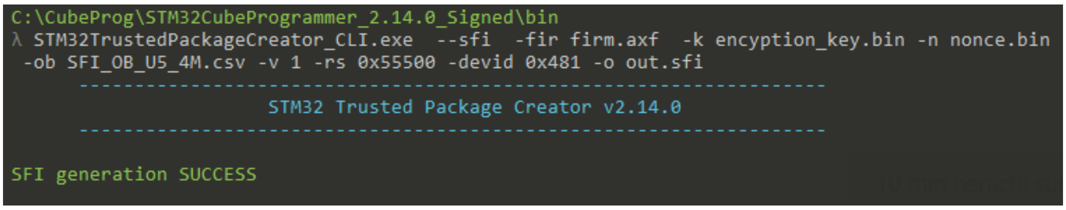
<out_file > : the SFI file to be generated (must have the ".sfi" extension).

Example of SFI generation command using an ELF file:

```
STM32TrustedPackageCreator_CLI.exe  --sfi  -fir firm.axf
-k encyption_key.bin -n nonce.bin  -ob SFI_OB_U5_4M.csv -v 1
-rs 0x55500 -devid 0x481 -o out.sfi
```

The result of the previous command is shown in *Figure 16*.

Figure 16. SFI generation example using an ELF file



DT48249V3

2.5.2 Steps for SSP generation (CLI)

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

Inputs for the “ssp” command are:

-ru, --rma_unlock

Description: RMA unlock password

Syntax: -ru <RMA_Unlock>

<RMA_Unlock> : Hexadecimal value 0x0000 to 0x7FFF

-rr, --rma_relock

Description: RMA relock password

Syntax: -rr <relock_value>

<relock_value> : Hexadecimal value 0x0000 to 0x7FFF

-b, --blob

Description: Binary to encrypt

Syntax: -b <Blob>

<Blob> : Secrets file of size 148 bytes

-pk, --pubk

Description: OEM public key file

Syntax: -pk <PubK.pem>

<PubK> : pem file of size 178 bytes

-k, --key

Description: AES-GCM encryption key

Syntax: -k <Key_File>

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

-n, --nonce

Description: AES-GCM nonce

Syntax: -n <Nonce_File>

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

-o, --out

Description: Generates an SSP file

Syntax: -out <Output_File.ssp>

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

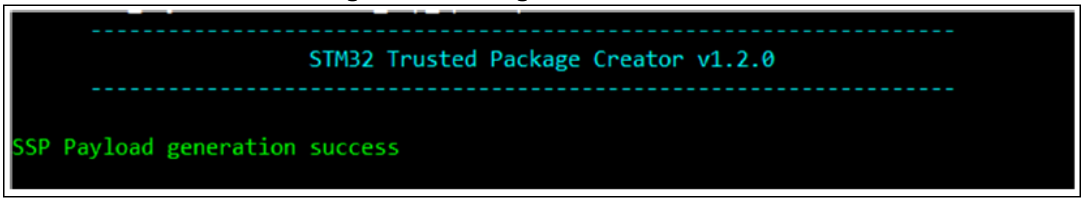
If all input fields are validated, an SSP file is generated in the directory path already mentioned in the “-o” option.

Example SSP generation command:

```
STM32TrustedPackageCreator_CLI -ssp -ru 0x312 -rr 0xECA
-b "C:\SSP\secrets\secrets.bin"
-pk "C:\SSP\OEMPublicKey.pem" -k "C:\SSP\key.bin"
-n "C:\SSP\nonce.bin" -o "C:\out.ssp"
```

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

Figure 17. SSP generation success



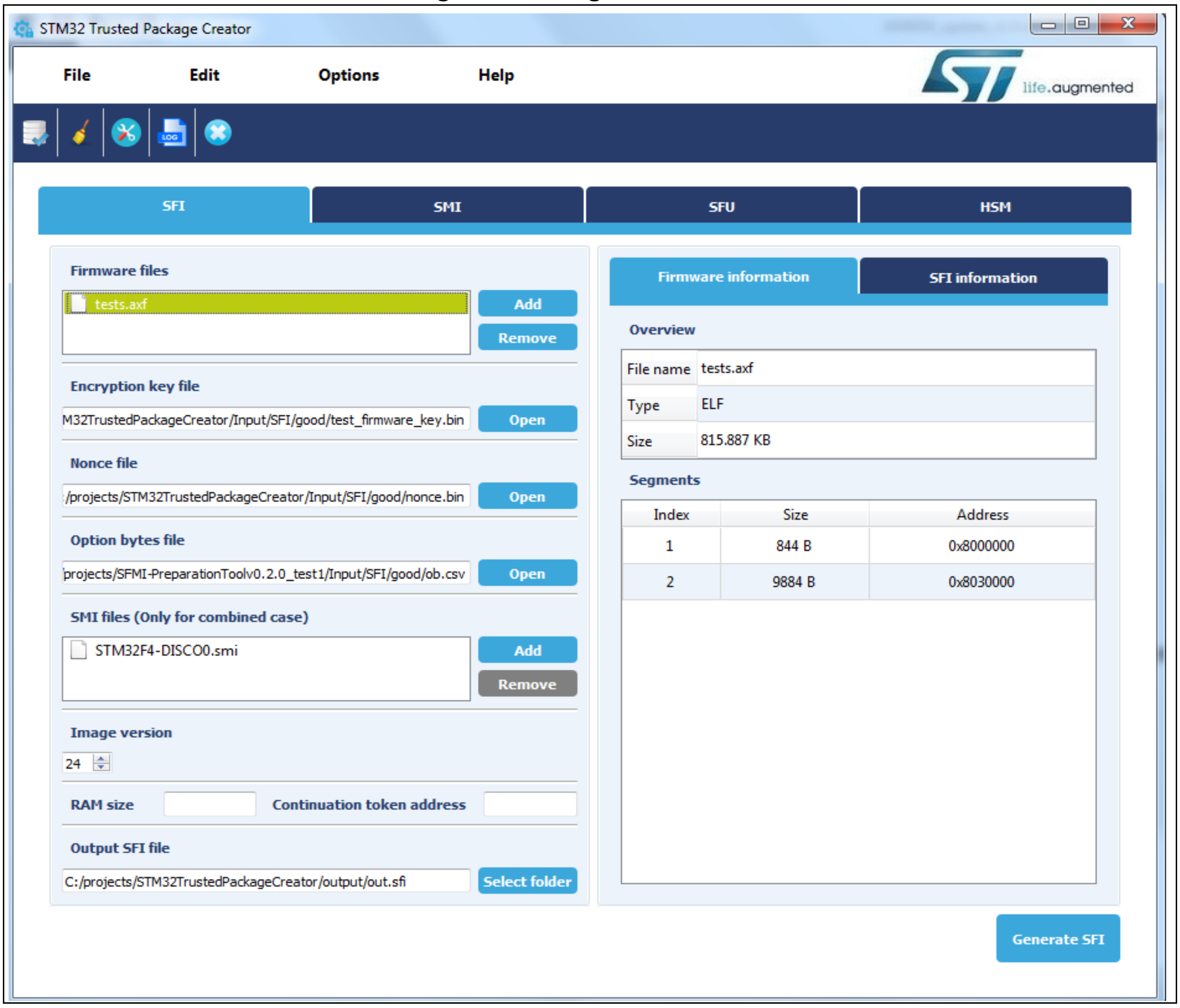
2.6 Using the STM32 Trusted Package Creator tool graphical user interface

The STPC is also available in graphical mode. This section describes its use.

2.6.1 SFI generation using STPC in GUI mode

Figure 18 shows the graphical user interface tab corresponding to SFI generation.

Figure 18. SFI generation Tab



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

SFI GUI tab fields

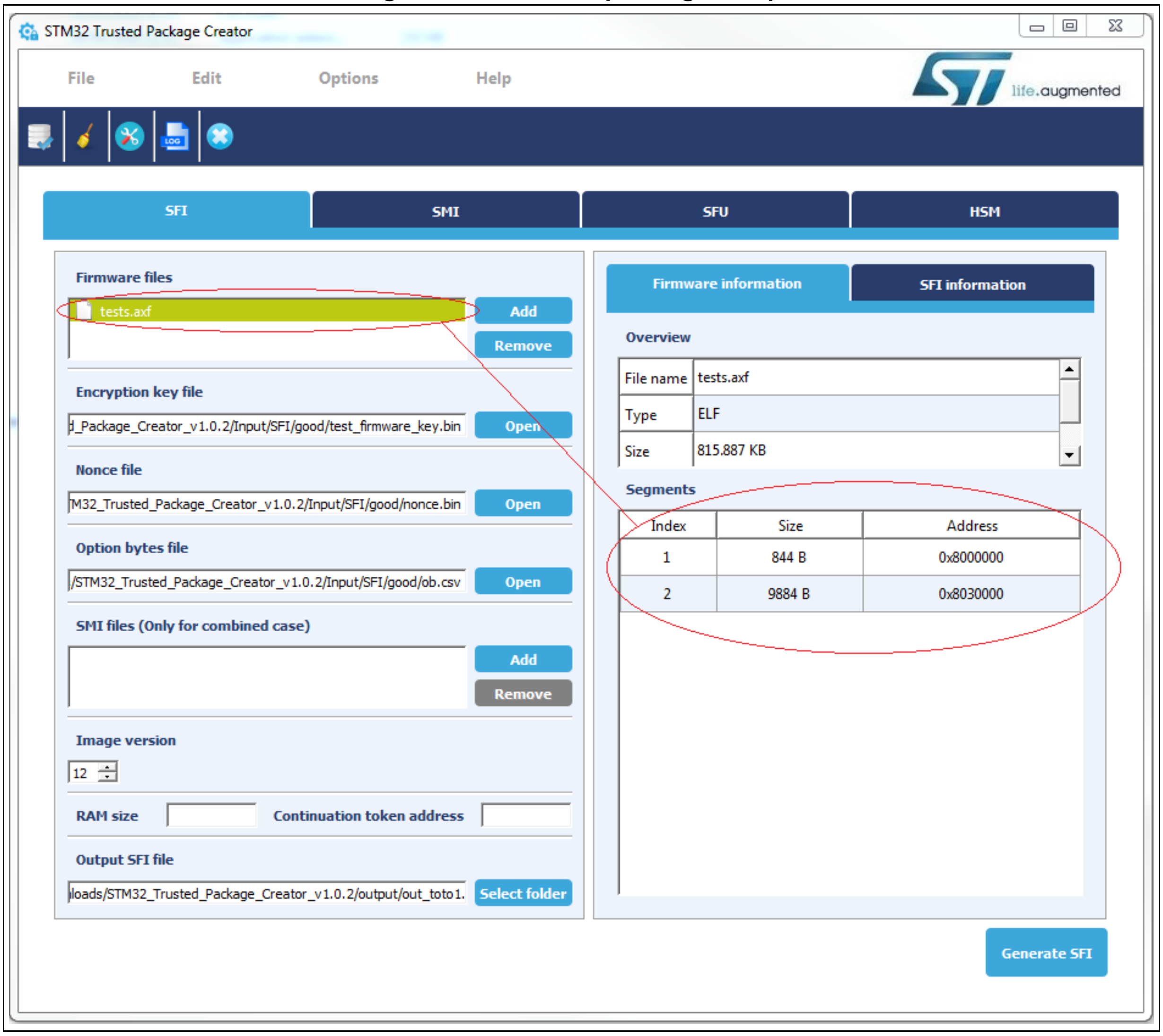
- Firmware files:

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

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

  Clicking on "input firmware file" causes related information to appear in the "Firmware information" section (*Figure 19*).

Figure 19. Firmware parsing example



- Encryption key and nonce file:

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

- Option bytes file:

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

Note: STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

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

- Image version:

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

- Output file:

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

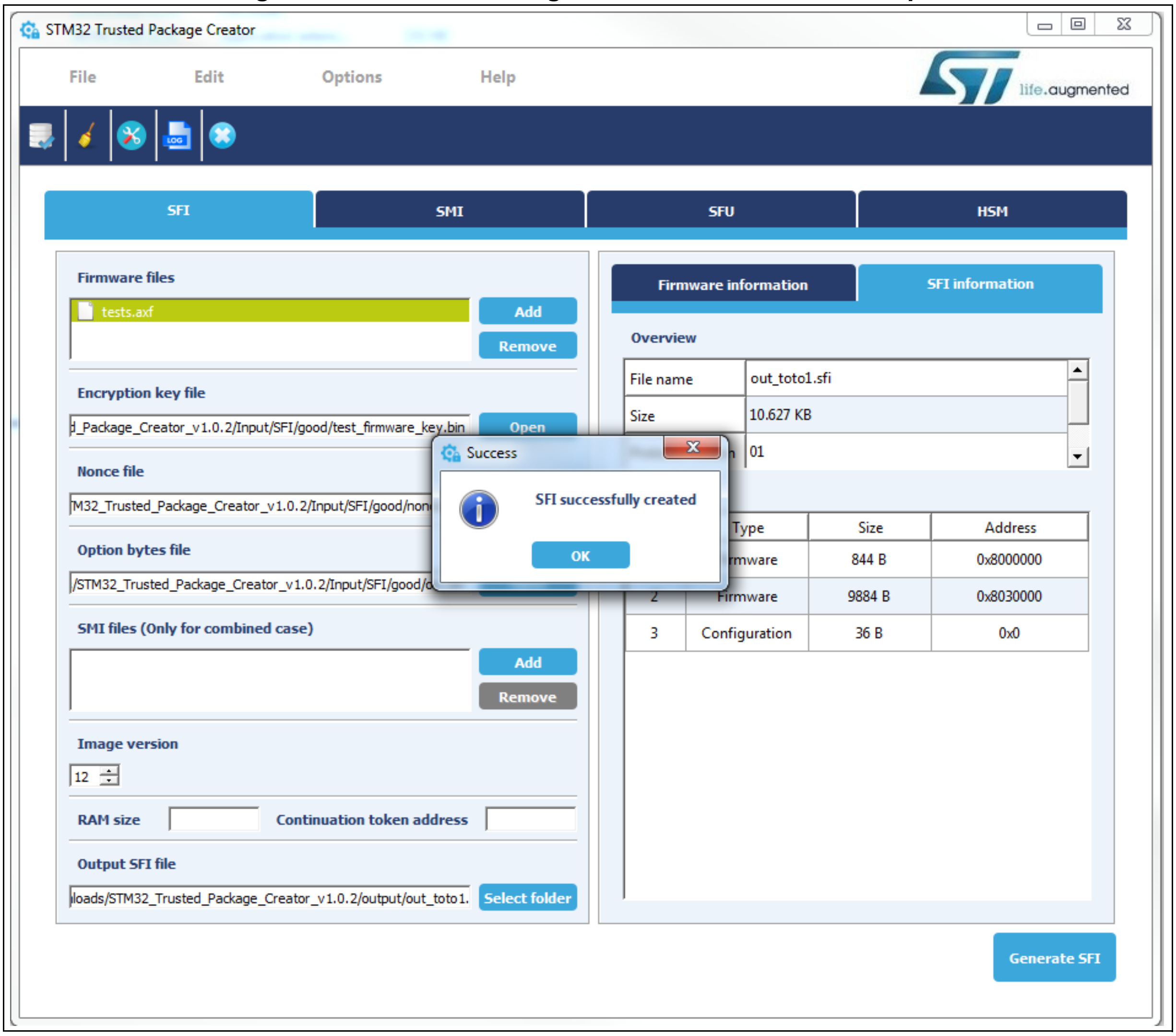
Note: By using the “Select folder” button, the name “out.sfi” is automatically suggested. This can be kept or changed.

- ‘Generate SFI’ button:

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

  If everything goes well, a message box indicating successful generation appears (*Figure 20*) and information about the generated SFI file is displayed in the SFI information section.

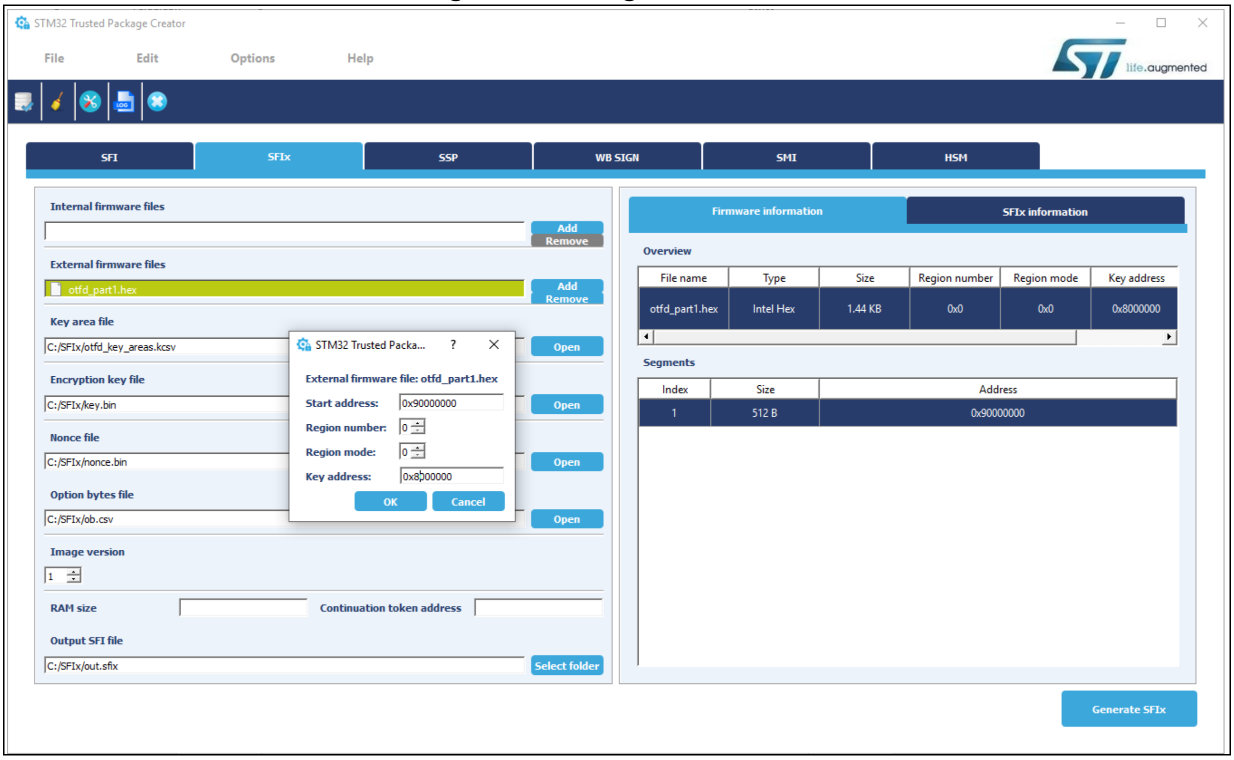
Figure 20. SFI successful generation in GUI mode example



2.6.2 SFIx generation using STPC in GUI mode

Figure 21 shows the graphical user interface tab corresponding to SFIx generation.

Figure 21. SFIx generation Tab

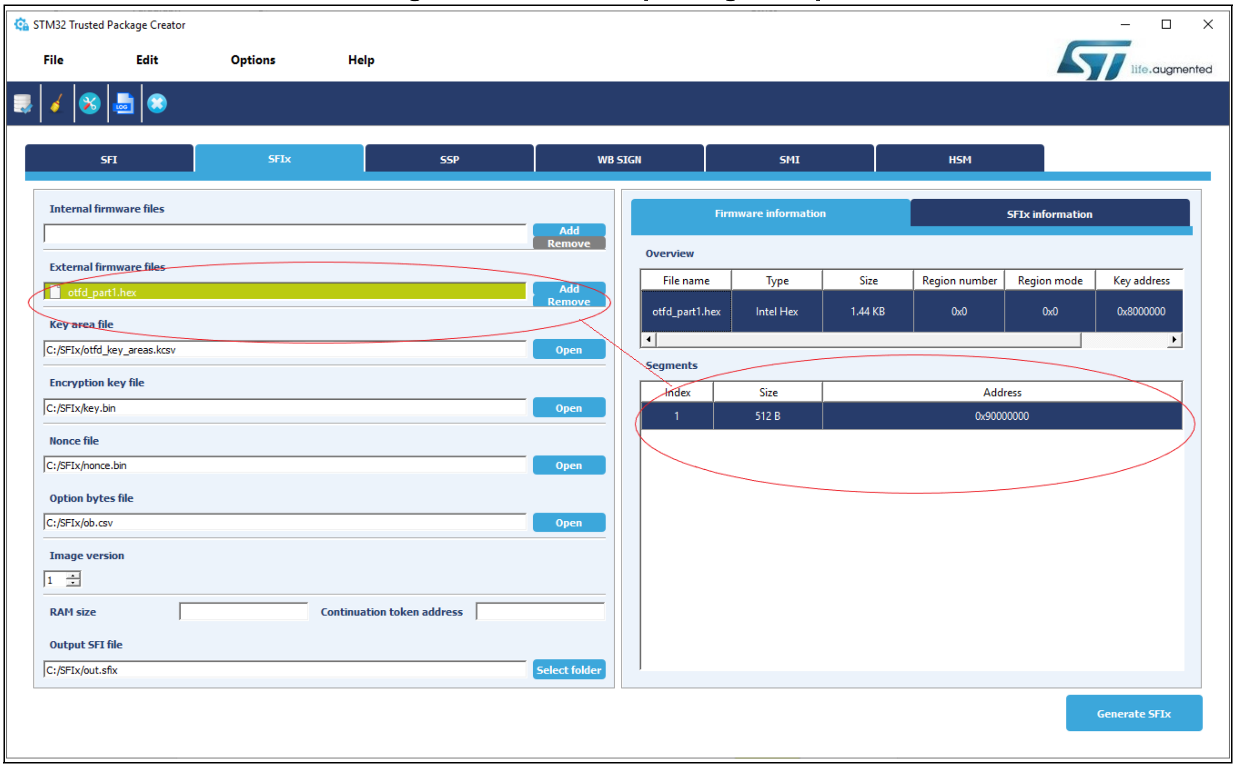


To generate an SFIx image successfully from the supported input firmware formats, the user must fill in the interface fields with valid values.

SFIx GUI tab fields

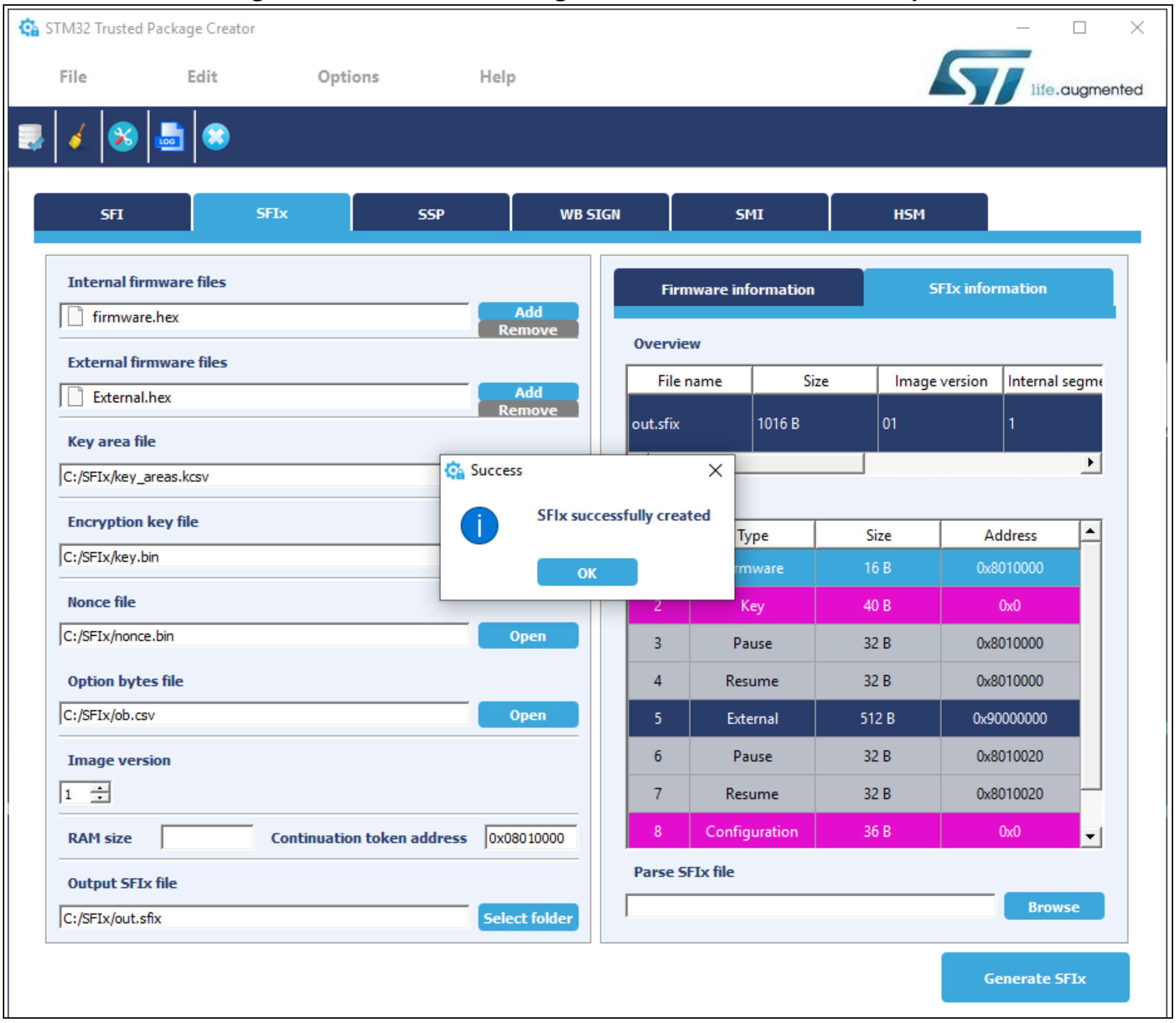
Firmware files: The user needs to add the input firmware files with the “Add” button. If the file is valid, it is appended to the “input firmware files “list, otherwise an error message box appears to notify the user that either the file could not be opened, or the file is not valid. Clicking on “input firmware file“ causes information related information to appear in the “Firmware information” section (*Figure 22*).

Figure 22. Firmware parsing example



As is the case for the SFI use case, once all fields are filled in properly, the “Generate SFIx” button becomes enabled. The user can generate the SFIx file by a single click on it. If everything goes well, a message box indicating successful generation appears (*Figure 23*) and information about the generated SFIx file is displayed in the SFIx information section.

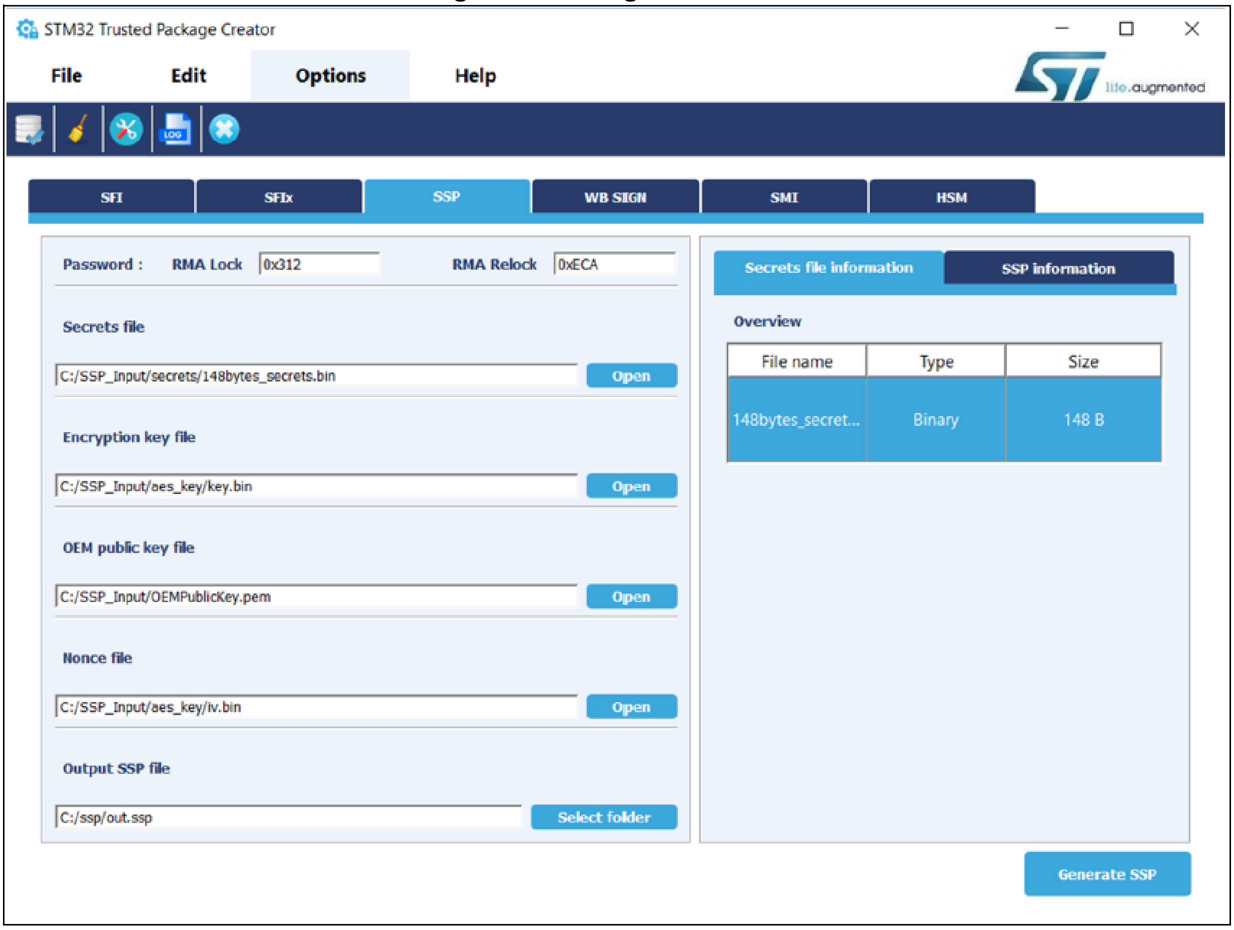
Figure 23. SFIx successful generation in GUI mode example



2.6.3 SSP generation using STPC in GUI mode

Figure 24 shows the SSP generation graphical user interface tab.

Figure 24. SSP generation tab



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

SSP GUI tab fields

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

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

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

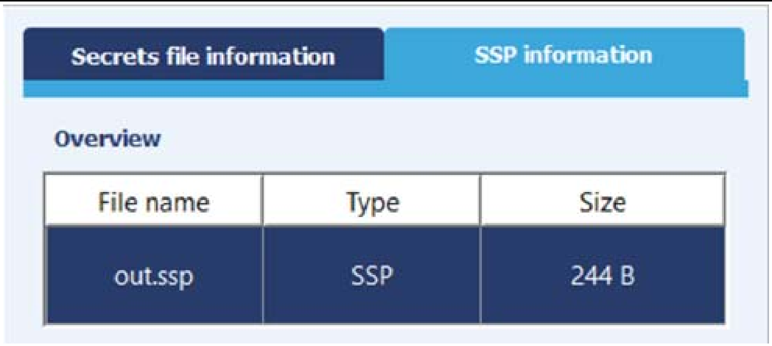
Encryption key and nonce files: The encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selection with the **Open** button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for the nonce).

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

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

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

Figure 25. SSP output information



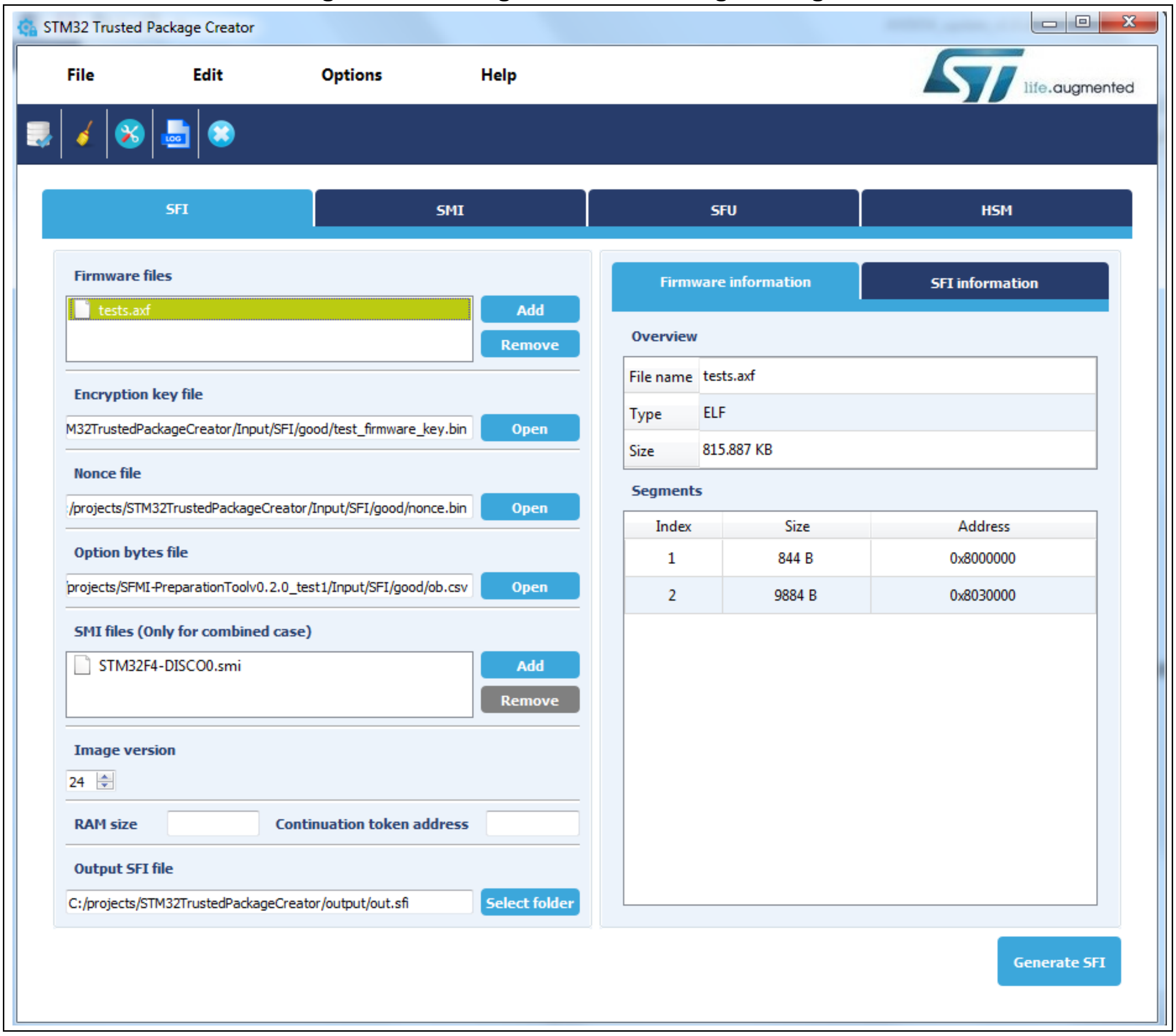
When the generation is complete, SSP information is available in the SSP overview section.

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

2.6.4 Settings

The STPC allows generation to be performed respecting some user-defined settings. The settings dialog is displayed by clicking the settings icon (see *Figure 26*) in the tool bar or in the menu bar by choosing: Options -> settings.

Figure 26. 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 to merge them and reduce the number of segments. The user might choose to perform padding either with 0xFF (the 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:

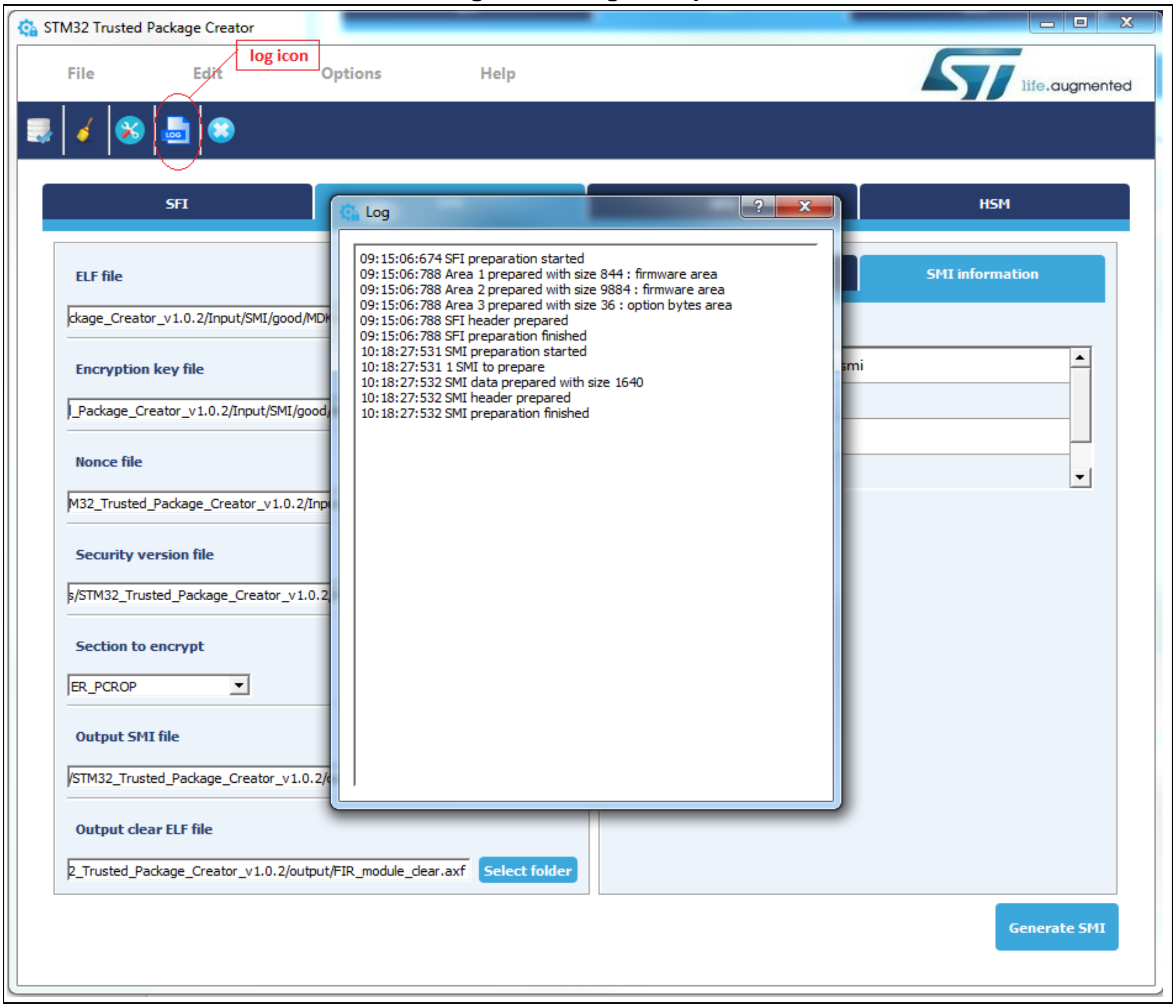
  When checked, a log file is generated in the selected path.

2.6.5 Log generation

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

Figure 27 shows a log example:

Figure 27. Log example



3 Encrypted firmware (SFI/SFIx) programming with STM32CubeProgrammer

STM32CubeProgrammer is a tool for programming STM32 devices through UART, USB, SPI, CAN, I^2C, JTAG, and SWD interfaces. So far, programming via JTAG/SWD is only supported with an ST-LINK probe.

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

3.1 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.

3.1.1 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 attackers who could substitute the public key with their own key.

3.1.2 License mechanism

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

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

License mechanism general scheme

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

When a customer wants to download the firmware to a chip, they send a chip identifier to the provider server, STM32HSM, 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.

ST solution is based on STM32HSM: a standalone chip in a smartcard form factor that could be programmed during the SFI preparation then used on the device production line. This solution is securing end to end transport of the firmware. Only the STM32 is capable to authenticate and decrypt the firmware. In addition, an ST solution based on STM32HSM is protecting device production against cloning.

Other solutions could be considered and STMicroelectronics, through its partnership program, is offering programming services. Find yours from the following link: Global Services from Partners - STMicroelectronics.

STM32HSM programming by OEM for license distribution

Before an OEM delivers an STM32HSM to a programming house for deployment as a license generation tool for programming of relevant STM32 devices, some customization of the STM32HSM must be done first.

The STM32HSM needs to be programmed with all the data needed for the license scheme deployment. In the production line, a dedicated API is available for STM32HSM personalization and provisioning.

This data is as follows:

- **The counter**: the counter is set to a maximum value that corresponds to the maximum number of licenses that can be delivered by the STM32HSM. It aims to prevent overprogramming.

  It is decremented with each license delivered by the STM32HSM.

  No more licenses are delivered by the STM32HSM once the counter is equal to zero.

  The maximum counter value must not exceed a maximum predefined value, which depends on the STM32HSM used.
- **The firmware key**: the key size is 32 bytes. It is composed of two fields: the initialization vector field (IV) and the key field, which are used for AES128-GCM firmware encryption.

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

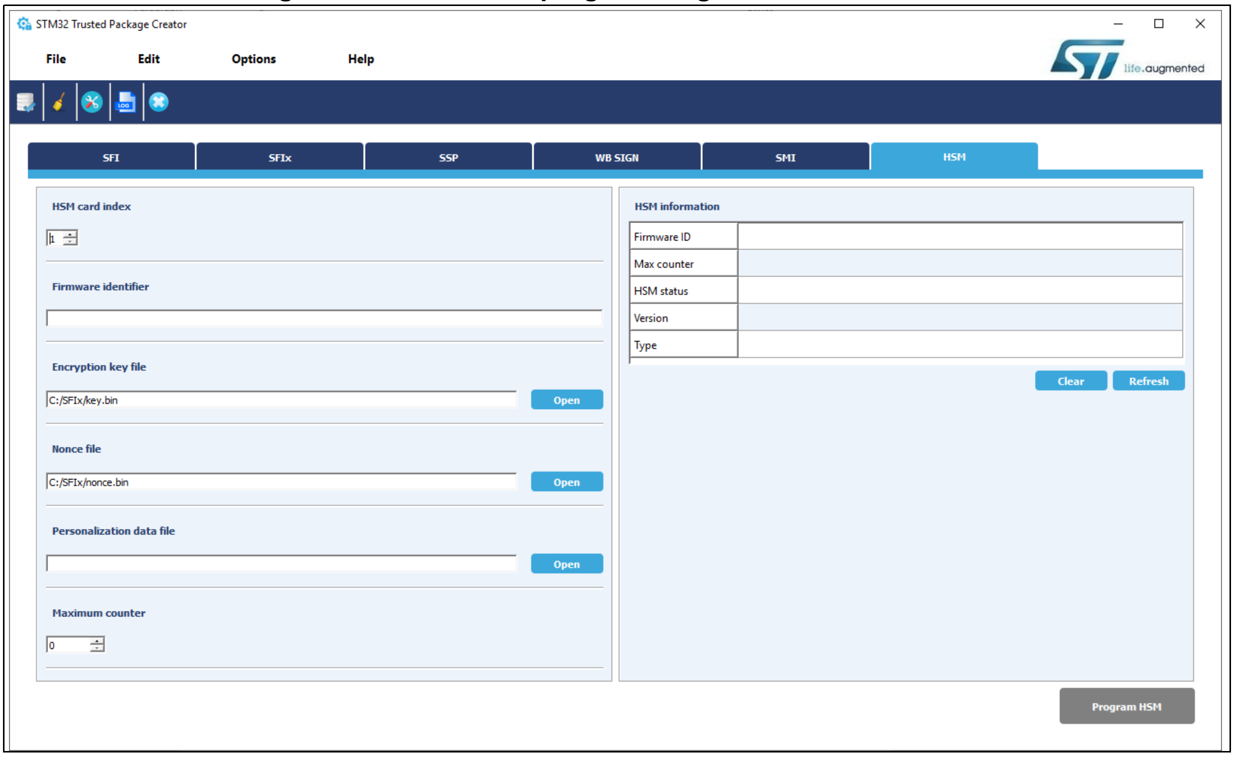
  Both fields must remain secret; that is why there are encrypted before being sent to the chip.

  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. As a consequence, the STM32HSM must be changed.
- **The firmware identifier**: allows the correct STM32HSM to be identified for a given firmware.
- **The personalization dat**a: this is specific to each MCU and delivered inside the TPC directory. More info about personalization data in *Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode)*.

The STM32HSM must be in “OPERATIONAL STATE” (locked) when shipped by the OEM to guarantee the OEM’s data confidentiality and privacy.

STMicroelectronics provides the tools needed to support SFI/SFIx via STM32HSM. In fact, STM32HSM programming is supported by the STM32 Trusted Package Creator tool. *Figure 28* shows the GUI for STM32HSM programming in the STPC tool.

Figure 28. STM32HSM programming GUI in the STPC tool



During SFI install, STM32CubeProgrammer communicates with the device to get the chip certificate, upload it into the STM32HSM to request the license. Once the license is generated by the STM32HSM, it gives it back to the STM32 device.

3.2 Secure programming using a bootloader interface

3.2.1 STM32CubeProgrammer for SFI using a bootloader interface

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

-sfi, --sfi

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

[<protocol=Ptype>] : Protocol type to be used: static/live
Only a static protocol is supported so far

Default value static

<file_path> : Path of sfi file to be programmed

[hsm=0|1] : Set a user option for STM32HSM use value in
{0 (do not use STM32HSM), 1 (use STM32HSM)}
Default value : hsm = 0

<lic_path|slot=slotID> : Path to the SFI license file (if hsm = 0)
or reader slot ID if STM32HSM is used (hsm = 1)

[During th SFI process, the generated license can be used multiple times with the same MCU, without the need of an STM32HSM card.

Example using the UART bootloader interface:

To use an STM32HSM, the command is:

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

To use a license file, the command is:

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

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

3.2.2 STM32CubeProgrammer for SSP via a bootloader interface

In this part, the STM32CubeProgrammer tool is used in CLI mode (the only mode available so far for secure programming) to program the SSP image already created with STM32 Trusted Package Creator. STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on smartcard) to generate a license for the connected STM32 MPU device during SSP install.

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

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

-ssp, --ssp

Description: Programs an SSP file

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

- <ssp_file_path>: SSP file path to be programmed, bin, or ssp extensions
- <ssp-fw-path>: SSP signed firmware path
- <hsm=0|1>: Set user option for STM32HSM use (do not use STM32HSM / use STM32HSM)
  Default value: hsm=0
- <license_path|slot=slotID>:
  Path to the license file (if hsm=0), Reader slot ID if STM32HSM is used (if hsm=1)

Example using USB DFU bootloader interface:

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

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

Figure 29. SSP installation success

```
Requesting Chip Certificate...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

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

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

Writing blob


Blob successfully written

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

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

3.2.3 STM32CubeProgrammer get certificate via a bootloader interface

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

-gc, --getcertificate

Syntax: –gc <file_path>

Example using the 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 30*.

Figure 30. 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
```

3.3 Secure programming using the JTAG/SWD interface

3.3.1 SFI/SFIx programming using JTAG/SWD flow

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

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

SFI via debug interface is currently supported for STM32H753XI, STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, and STM32L5 devices.

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

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

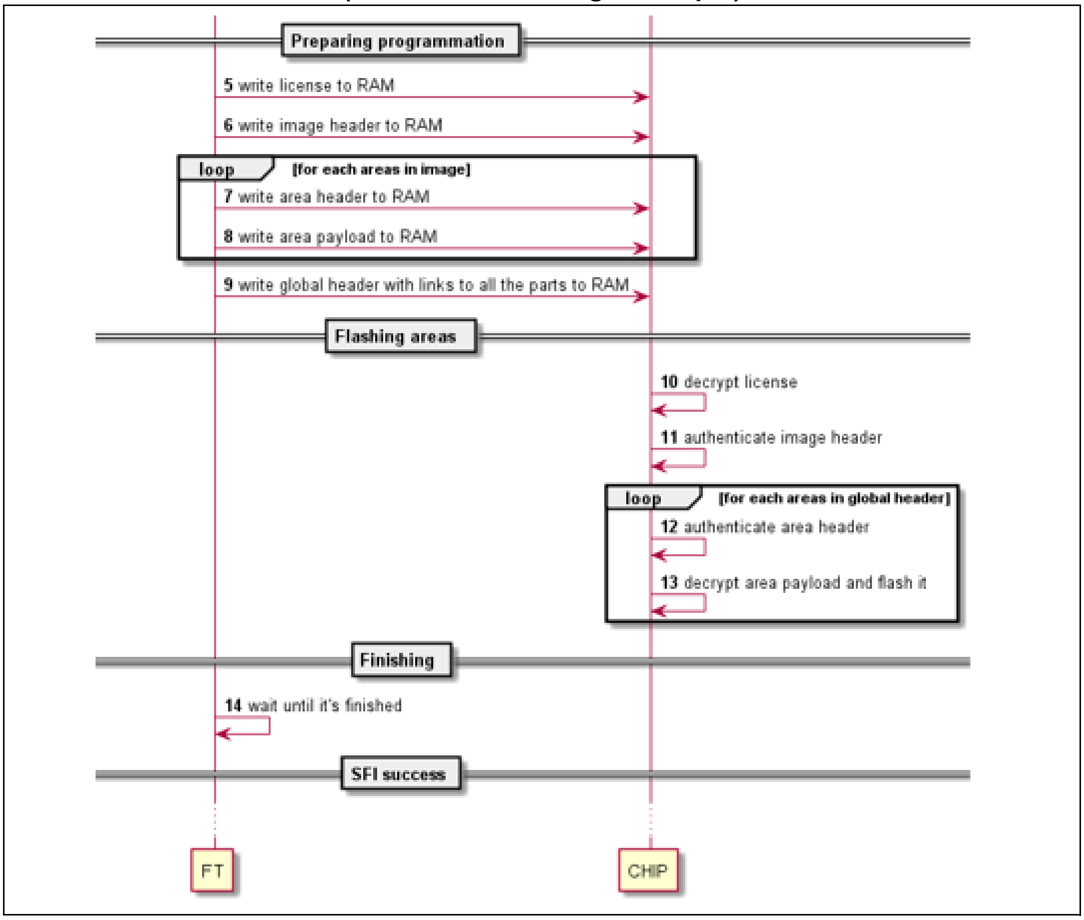
To remedy this, the SFI/SFIx image is split into several parts, so that each part fits into the allowed RAM size.

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

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

The SFI flow for programming through JTAG is described in *Figure 31*.

Figure 31. SFI programming by JTAG/SWD flow overview (monolithic SFI image example)



3.3.2 STM32CubeProgrammer for secure programming using JTAG/SWD

The only modification in the STM32CubeProgrammer secure command syntax is the connection type that 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.

The result of this example is shown in *Figure 32*.

Figure 32. 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
```

3.4 Secure programming using bootloader interface (UART/I^{2}C/SPI/USB)

It is also possible to program the SFI/SFIx image using the bootloader interface (UART/I^{2}C/SPI/USB). FDCAN is not supported by STLINK-V3.

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

SFI via the bootloader interface (UART/I^{2}C/SPI/USB) is currently supported for STM32L5 devices. It needs to load an external loader using the **-elbl** command in the SRAM.

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

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

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

SFI example

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

SFIx example

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

4 Example of SFI programming scenario

4.1 Scenario overview

The actual user application to be installed on the STM32H753XI (or STM32L5) device makes “`printf`” packets appear in serial terminals. The application was encrypted using the STPC.

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

4.2 Hardware and software environment

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

- STM32H743I-EVAL board
- STM32H753XI with bootloader and RSS programmed
- RS-232 cable for SFI programming via UART
- Micro-B USB for debug connection
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v0.2.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v0.4.0 (or greater) package available from *www.st.com*

Note: Refer to [4] or [5] for the supported operating systems and architectures.

4.3 Step-by-step execution

4.3.1 Build OEM application

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

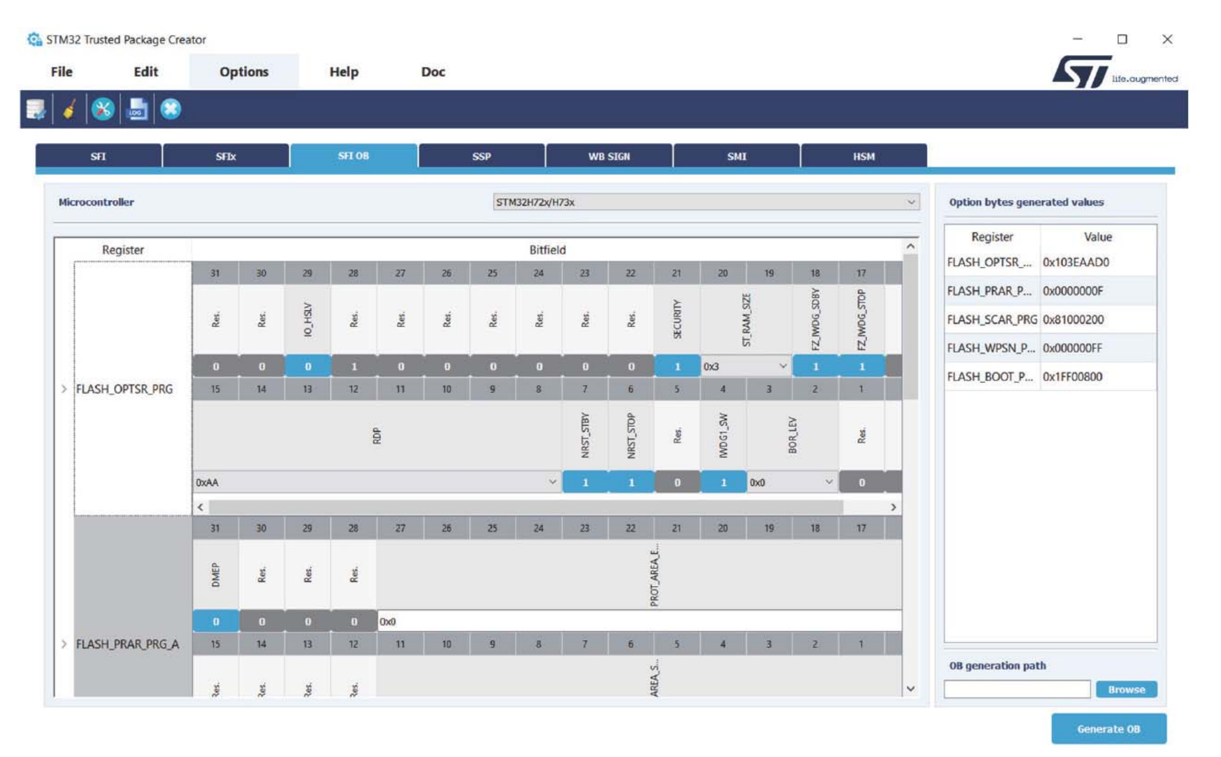
4.3.2 Performing the option byte file generation (GUI mode)

The STM32 Trusted Package Creator tool GUI presents an SFI OB tab to generate an option bytes CSV file with a custom option byte value.

To generate an SFI CSV option bytes file, the user must:

1. Select the concerned product.
2. Fill the option bytes fields with desired values.
3. Select the generation path.
4. Click on the *Generate OB* button.

Figure 33. STM32Trusted Package Creator SFI OB GUI



4.3.3 Perform the SFI generation (GUI mode)

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

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

Note: *STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product.*
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

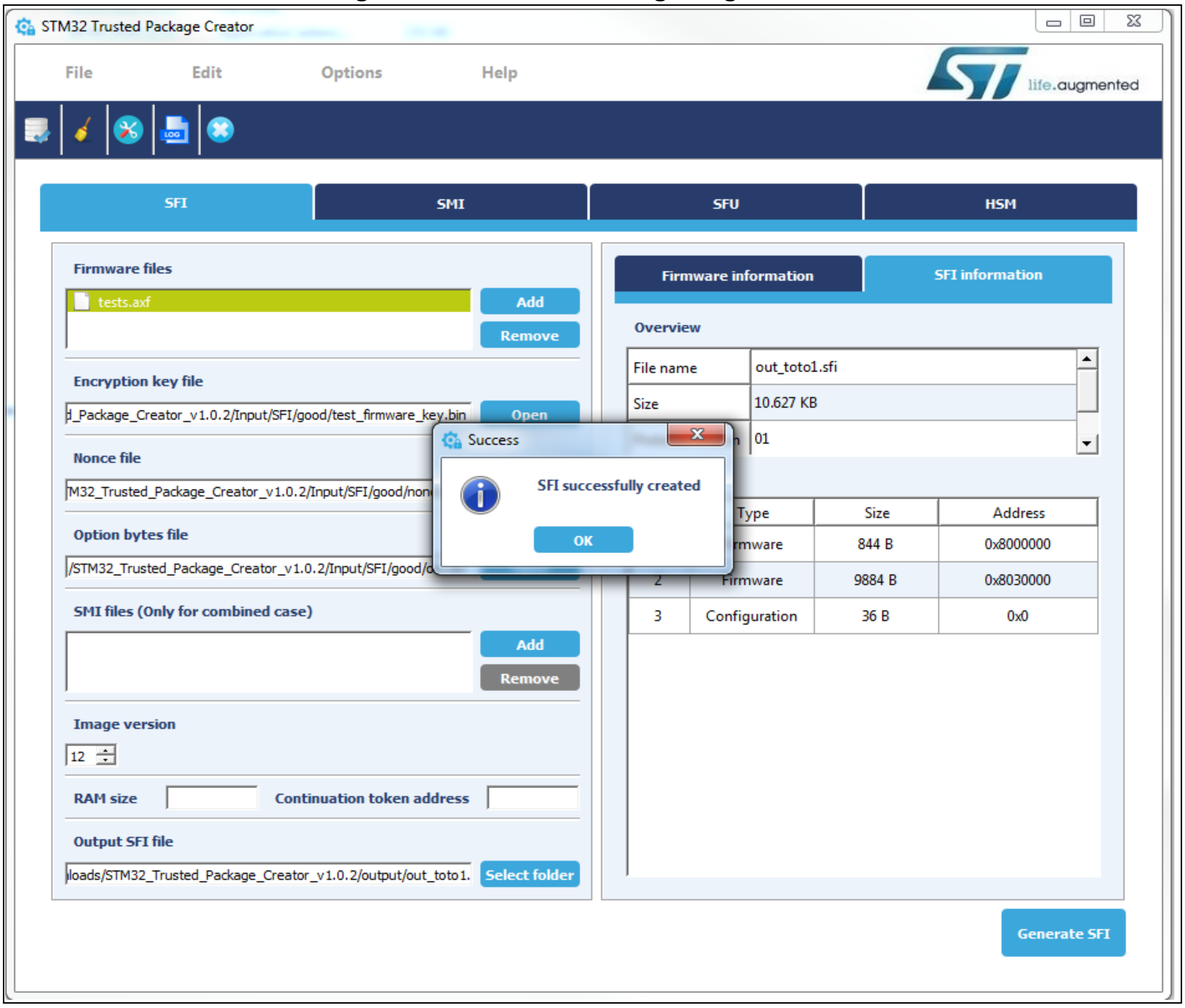
The option bytes are described in the product reference manual.

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

Note: *If you want to reopen the Device using the Debug Authentication mechanism, a DA ObKey file must be included in the SFI image, otherwise the device becomes inaccessible.*

Figure 34 shows the STPC GUI during the SFI generation.

Figure 34. STPC GUI during SFI generation



4.3.4 Performing STM32HSM 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, STM32HSMs are used as license generation tools in the field. See *Section 3.1.2: License mechanism* for STM32HSM use and programming.

Figure 35 shows an example for STM32HSM programming by OEM to be used for SFI install.

The maximum number of licenses delivered by the STM32HSM in this example is 1000.

This example uses STM32HSM-V2, and is also valid for STM32HSM-V1 when the 'version' field is set accordingly. The STM32HSM version can be identified before performing the programming operation by clicking the Refresh button to make the version number appear in the 'version' field.

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

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

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

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

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

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

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

Once you have the product ID, you can differentiate the personalization package to be used on the STM32HSM provisioning step respecting the following naming convention:

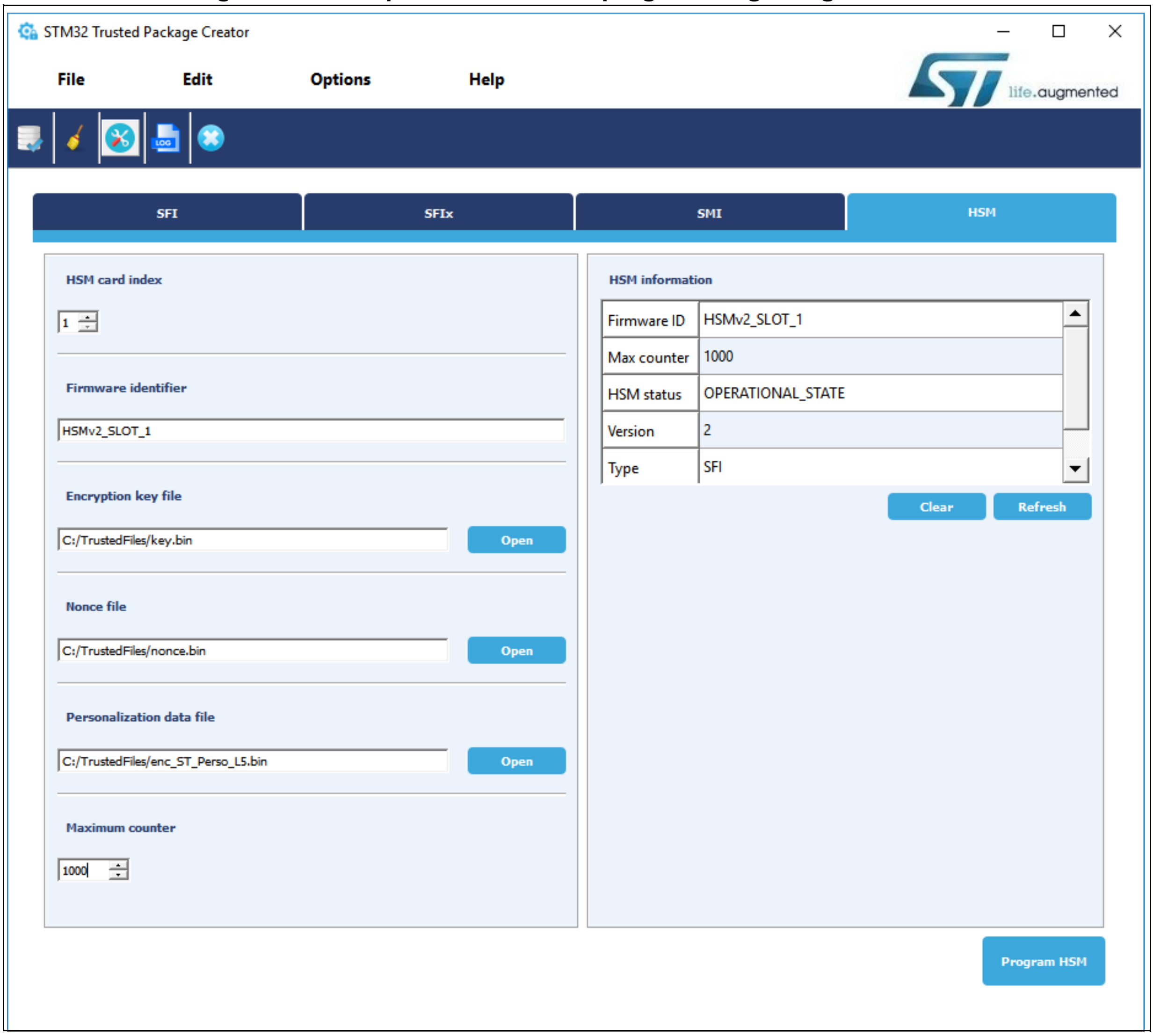
ProdcutID_FlowType_LicenseVersion_SecurityVersion.enc.bin

For example: 47201003_SFI._01000000_00000000.enc.bin

Based on this name we can retrieve the associated information:

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

Figure 35. Example of STM32HSM programming using STPC GUI



Note: *When using STM32HSM-V1, the "Personalization data file" field is ignored when programming starts. It is only used with STM32HSM-V2.*

When the card is successfully programmed, a popup window message "HSM successfully programmed" appears, and the STM32HSM is locked. Otherwise, an error message is displayed.

4.3.5 Performing STM32HSM programming for license generation using STPC (CLI mode)

STM32 Trusted Package Creator provides CLI commands to program STM32HSM cards. To configure the STM32HSM before programming, the user must provide the mandatory inputs by using the specific options.

Example of STM32HSM-V1 provisioning

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

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

STM32HSM-V2 allows users to personalize their own HSM to achieve, for example, compatibility with the desired STM32 device. This solution covers the limitations of STM32HSM-V1 (static behavior), so it is possible to support new devices that are not available on STM32HSM-V1.

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

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

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

Figure 36. Example product ID

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

The product ID of the STM32WL used is: 49701005

In the second step, the users provision their own STM32HSM-V2 by programming it using STPC. The personalization data file .bin can be found under *"..\bin\PersoPackages"*.

Example of STM32HSM-V2 provisioning

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

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

- -pd: Set the personalization data file path.

To obtain the appropriate personalization data file and for further information, refer to *Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode)*.

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

If the STM32HSM 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 STM32HSM is locked.

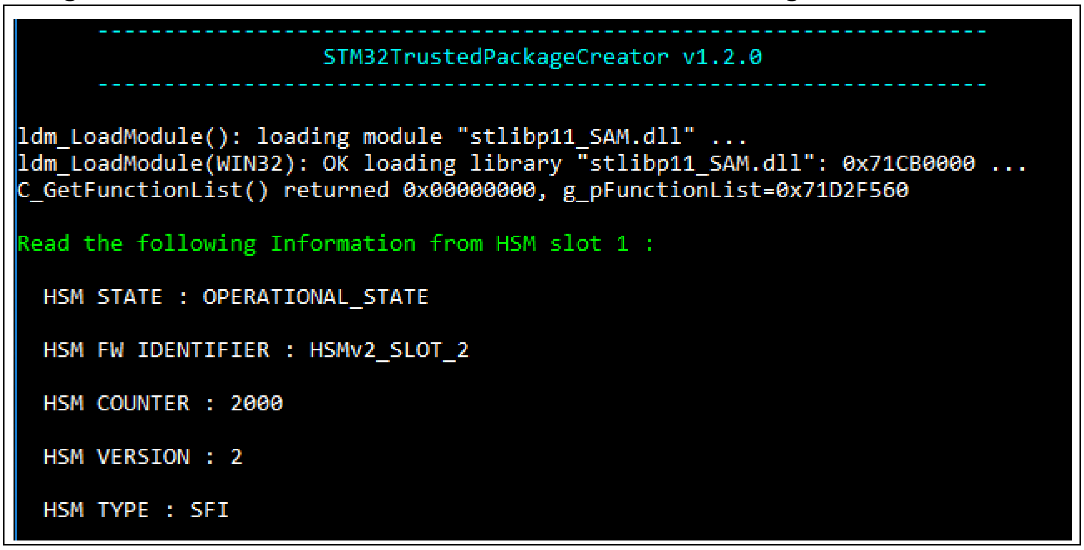
STM32HSM-V1 supports a list of a limited number of STM32 devices such as STM32L4, STM32H7, STM32L5, and STM32WL.

Example of STM32HSM get information

If the STM32HSM 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 STM32HSM by using the command below:

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

Figure 37. STM32HSM information in STM32 Trusted Package Creator CLI mode



4.3.6 Programming input conditions

Before performing an SFI install make sure that:

- Flash memory is erased.
- No PCROPed zone is active, otherwise destroy it.
- The chip must support security (a security bit must be present in the option bytes).
- When using a UART interface, the user security bit in option bytes must be enabled before launching the SFI command. For this, the following STM32CubeProgrammer command is launched:
  – Launch the following command (UART bootloader used => Boot0 pin set to VDD):
  `-c port=COM9 -ob SECURITY=1`
- When using a UART interface the Boot0 pin must be set to VSS:
  – After enabling security (boot0 pin set to VDD), a power off/power on is needed when switching the Boot0 pin from VDD to VSS: power off, switch pin then power on.
- When performing an SFI install using the UART bootloader then, no debug interface must be connected to any USB host. If a debug interface is still connected, disconnect it then perform a power off/power on before launching the SFI install to avoid any debug intrusion problem.
- Boot0 pin set to VDD When using a debug interface.
- A valid license generated for the currently used chip must be at your disposal, or a license generation tool to generate the license during SFI install (STM32HSM).
- For STM32L5 products, TZEN must be set at 0 (TZEN=0).

4.3.7 Performing 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 STMicroelectronics STM32HSMs (Hardware secure modules based on smartcard) to generate a license for the connected STM32 device during SFI install.

Using JTAG/SWD

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

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

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

The STM32CubeProgrammer command is as follows:

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

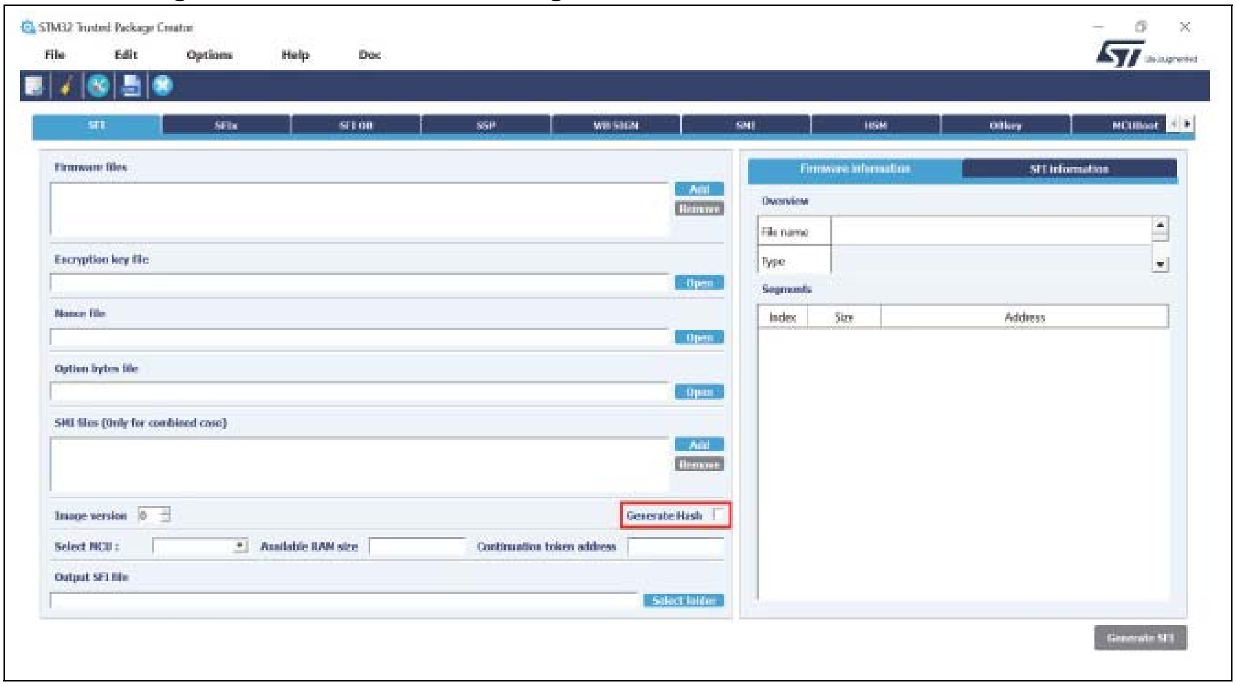
4.3.8 SFI with integrity check (for STM32H73xxx)

For the STM32H73xxx, an integrity check mechanism is implemented. STM32 Trusted Package Creator calculates the input firmware hash and integrates it into the SFI firmware. The STM32H73xxx MCU is able to use this hash input to check the firmware integrity.

Enabling this mechanism is mandatory for STM32H73xxx, and it can be done through GUI and CLI.

For the GUI part, hash is enabled by checking Generate hash.

Figure 38. STM32Trusted Package Creator SFI ‘hash Generator ‘check box



For the CLI part SFI command line must integrate the -hash option.

Usage example:

```
STM32TrustedPackageCreator_CLI.exe -sfi -fir OEM_Dev.bin
0x08000000 -k aeskey.bin -n nonce.bin -ob ob.csv -v 0 --
ramsize 0x1E000 --token 0x080FF000 -hash 1 -o outCLI.sfi
```

Figure 39 shows the SFI install via SWD execution and the STM32HSM as license generation tool in the field.

Figure 39. SFI installation success using SWD connection (1)

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

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

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

  SFI File Information          :

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

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

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

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

Reading the chip Certificate...


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

Requesting Licesne for firmware with ID : 111

requesting license for the current STM32 device

Init Communication ...

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

Succeed to generate license for the current STM32 device

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware with ID 111

Starting Firmware Install operation...


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

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

RSS command execution OK
```

Figure 40. SFI installation success using SWD connection (2)

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

Reconnected !

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

5 Example of SFI programming scenario for STM32WL

5.1 Scenario overview

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

- The encrypted firmware of STM32WL
- STM32HSM-V1 or provisioned STM32HSM-V2
- STM32CubeProgrammer

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

5.2 Hardware and software environment

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

- STM32WL5x board with bootloader and RSS programmed
- RS-232 cable for SFI programming via UART
- Micro-B USB for debug connection
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v1.7.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.16.0 (or greater) package available from *www.st.com*
- STM32HSM-V1 or STM32HSM-V2

Note: Refer to [4] or [5] for the supported operating systems and architectures.

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)

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

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

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

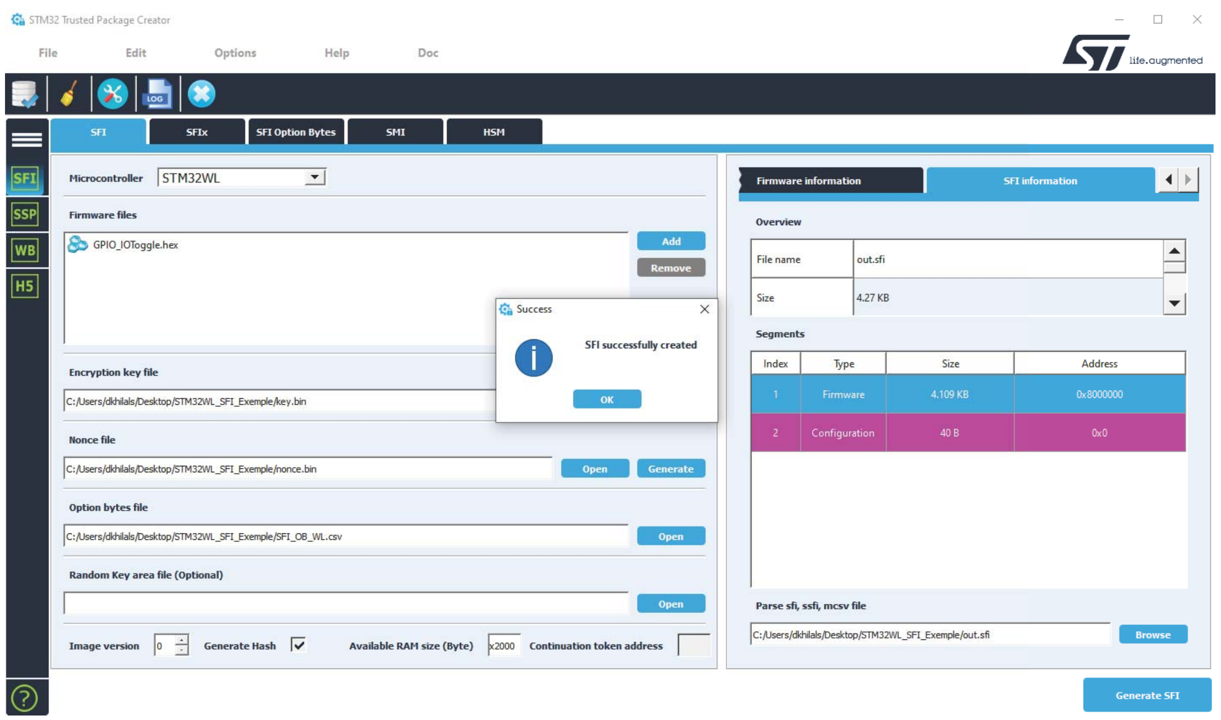
Note: STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

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

A programmed STM32HSM card must be inserted in the PC, and an *"out.sfi"* image is then generated.

Figure 41. STPC GUI during the SFI generation (STM32WL)



DT54151V2

Note: To perform STM32HSM programming for license generation using STPC (GUI mode and CLI mode) refer to the following sections:

Section 4.3.4: Performing STM32HSM programming for license generation using STPC (GUI mode)

Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode)

5.3.3 Programming input conditions

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

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

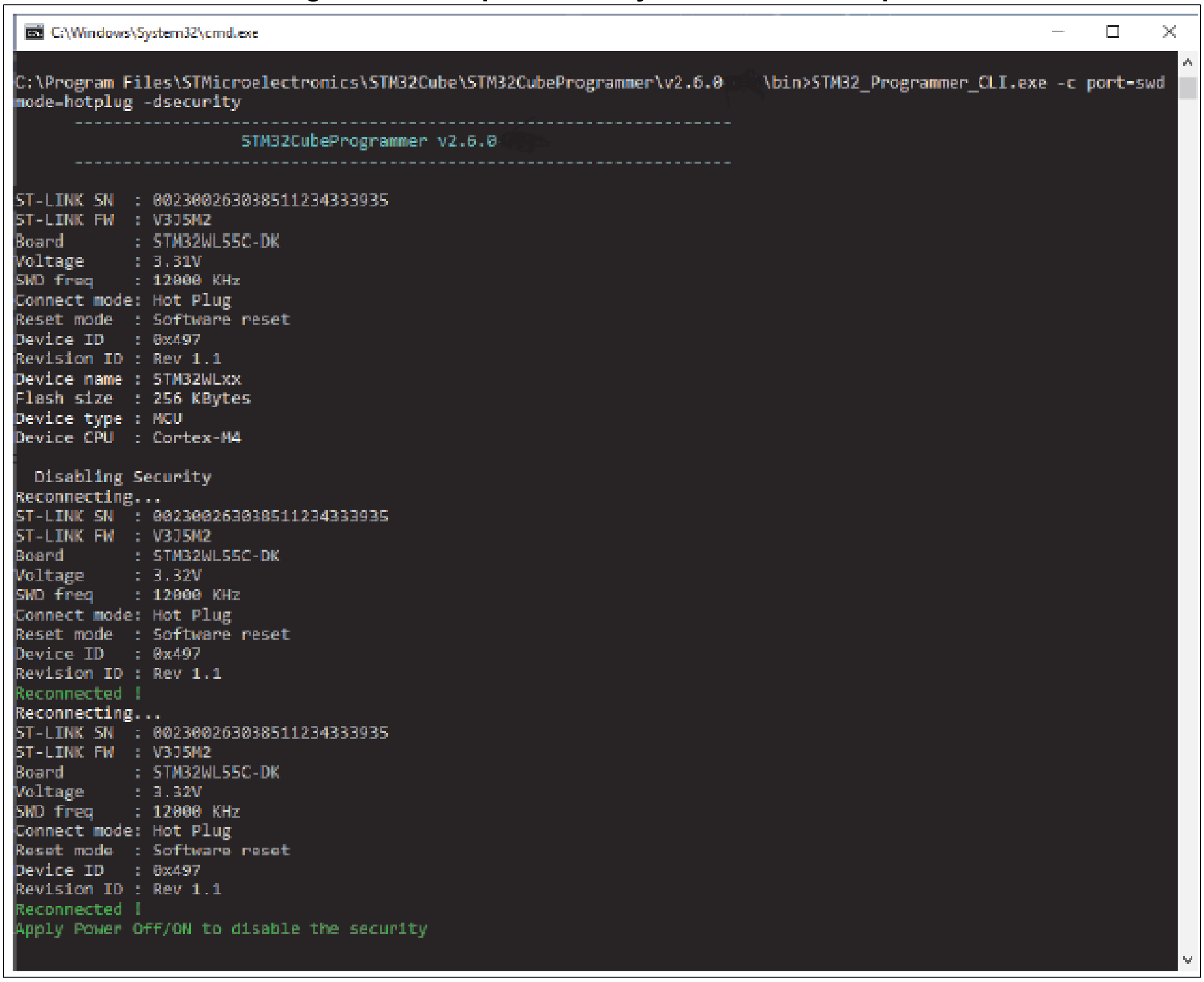
-desurity: This option allows the user to disable security. After executing this command, a power OFF / power ON must be done.

Example:

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

Figure 42 hows the resulting output on the command line.

Figure 42. Example -dsecurity command-line output



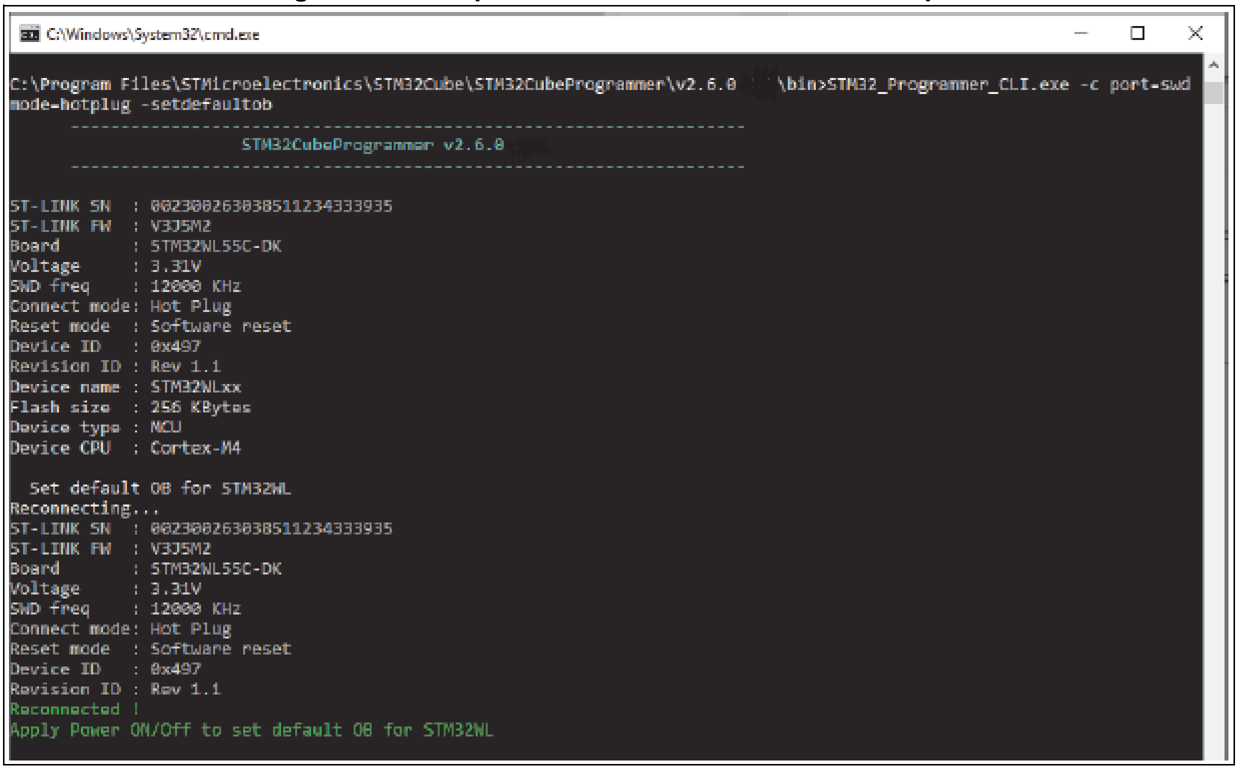
-setdefaultob: This command allows the user to configure option bytes to their default values. After executing this command, a power OFF/power ON must be done.

Example:

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

Figure 43 shows the resulting output on the command line.

Figure 43. Example -setdefaultob command-line output



5.3.4 Perform the SFI install using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (Hardware secure modules based on smartcard) to generate a license for the connected STM32 device during SFI install.

Using JTAG/SWD

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

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

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

Figure 44 shows the SFI install via SWD execution.

Figure 44. SFI installation via SWD execution command-line output

```
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammerv2.14.0-RO\bin>STM32_Programmer_CLI.exe -c port=swd mode=hotplug -sfi "C:\Users\dkhilals\Desktop\STM32WL_SFI_Exemple\out.sfi" hsm=0 "C:\Users\
dkhilals\Desktop\STM32WL_SFI_Exemple\license_wl.bin" -rsse "C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammerv2.14.0-RO\bin\RSSe\WL\enc_signed_RSSe_sfi.bin"
      -------------------------------------------------------------------
                        STM32CubeProgrammer v2.14.0
      -------------------------------------------------------------------

ST-LINK SN  : 002A000B5553500B20393256
ST-LINK FW  : V3J10M3
Board       : NUCLEO-WL55JC
Voltage     : 3.27V
SWD freq    : 30000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x497
Revision ID : Rev Y
Device name : STM32WLxx
Flash size  : 256 KBytes
Device type : MCU
Device CPU  : Cortex-M4
BL Version  : 0xC3

  Protocol Information         : static

  SFI File Information         :

     SFI file path             : C:\Users\dkhilals\Desktop\STM32WL_SFI_Exemple\out.sfi
     SFI license file path     : C:\Users\dkhilals\Desktop\STM32WL_SFI_Exemple\license_wl.bin
     SFI header information    :
         SFI protocol version        : 2
         SFI total number of areas   : 2
         SFI image version           : 0
     SFI Areas information     :

     Parsing Area 1/2    :
         Area type                   : F
         Area size                   : 4208
         Area destination address    : 0x8000000

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

Installing RSSe

Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi.bin
  File          : enc_signed_RSSe_sfi.bin
  Size          : 23.23 KB
  Address       : 0x20002020


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

File download complete
Time elapsed during download operation: 00:00:00.073
Boot on RSS...
Reconnecting...
ST-LINK SN  : 002A000B5553500B20393256
ST-LINK FW  : V3J10M3
Board       : NUCLEO-WL55JC
Voltage     : 3.27V
SWD freq    : 30000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x497
Revision ID : Rev Y
Reconnected !
RSSE SFI INITIALISED


RSS version = 2.3.0


RSSe version = 5.0.0

Starting SFI

Processing license...
RSSE SFI LICENSE OK
Processing Image Header
RSSE SFI IMAGE HEADER OK
Processing Area 1...
RSSE SFI AREA OK
Area Address = 0x8000000
Area Type    = F
Processing Area 2...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file C:\Users\dkhilals\Desktop\STM32WL_SFI_Exemple\out.sfi Install Operation Success

Time elapsed during SFI install operation: 00:00:07.728
```

DT54154V2

6 Example of SFI programming scenario for STM32U5

6.1 Scenario overview

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

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

6.2 Hardware and software environment

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

- STM32U5 board with bootloader and RSS programmed
- RS-232 cable for SFI programming via UART
- Micro-B USB for debug connection
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v1.2.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.8.0 (or greater) package available from *www.st.com*
- STM32HSM-V2

Note: Refer to [4] or [5] for the supported operating systems and architectures.

6.3 Step-by-step execution

6.3.1 Build OEM application

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

6.3.2 Perform the SFI generation (GUI mode)

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

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

Note: *STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product.*
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

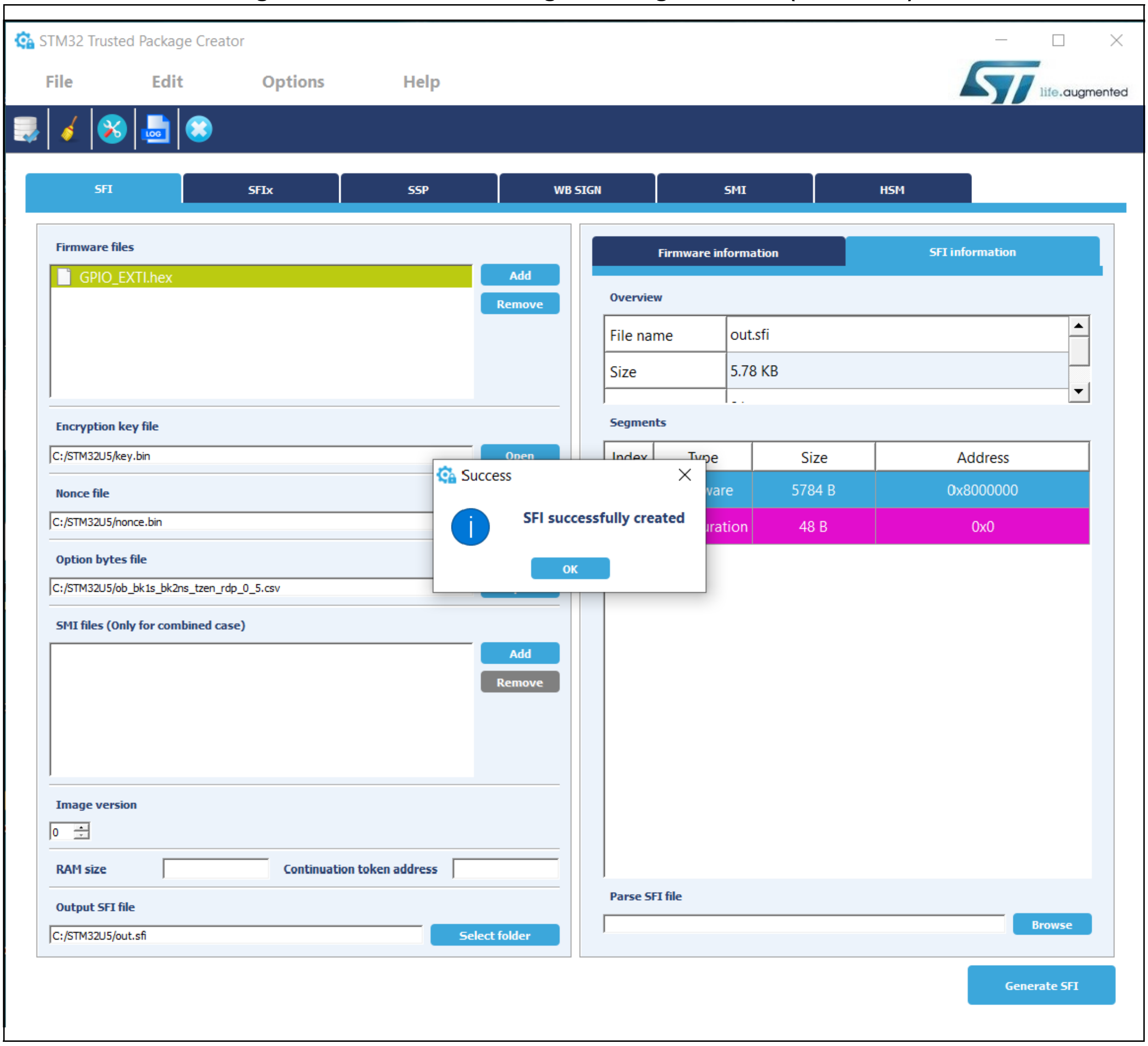
The option bytes are described in the product reference manual.

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

In addition, a programmed STM32HSM card must be inserted in the PC. An "*out.sfi*" image is then generated.

Figure 45 shows the STPC GUI during SFI generation.

Figure 45. STPC GUI during the SFI generation (STM32U5)



Note: *To perform STM32HSM programming for license generation using STPC (GUI and CLI modes), refer to Section 4.3.4: Performing STM32HSM programming for license generation using STPC (GUI mode) and Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode).*

6.3.3 Programming input conditions

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

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

6.3.4 Perform the SFI install using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on smartcards) to generate a license for the connected STM32 device during the SFI install process.

Using JTAG/SWD

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

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

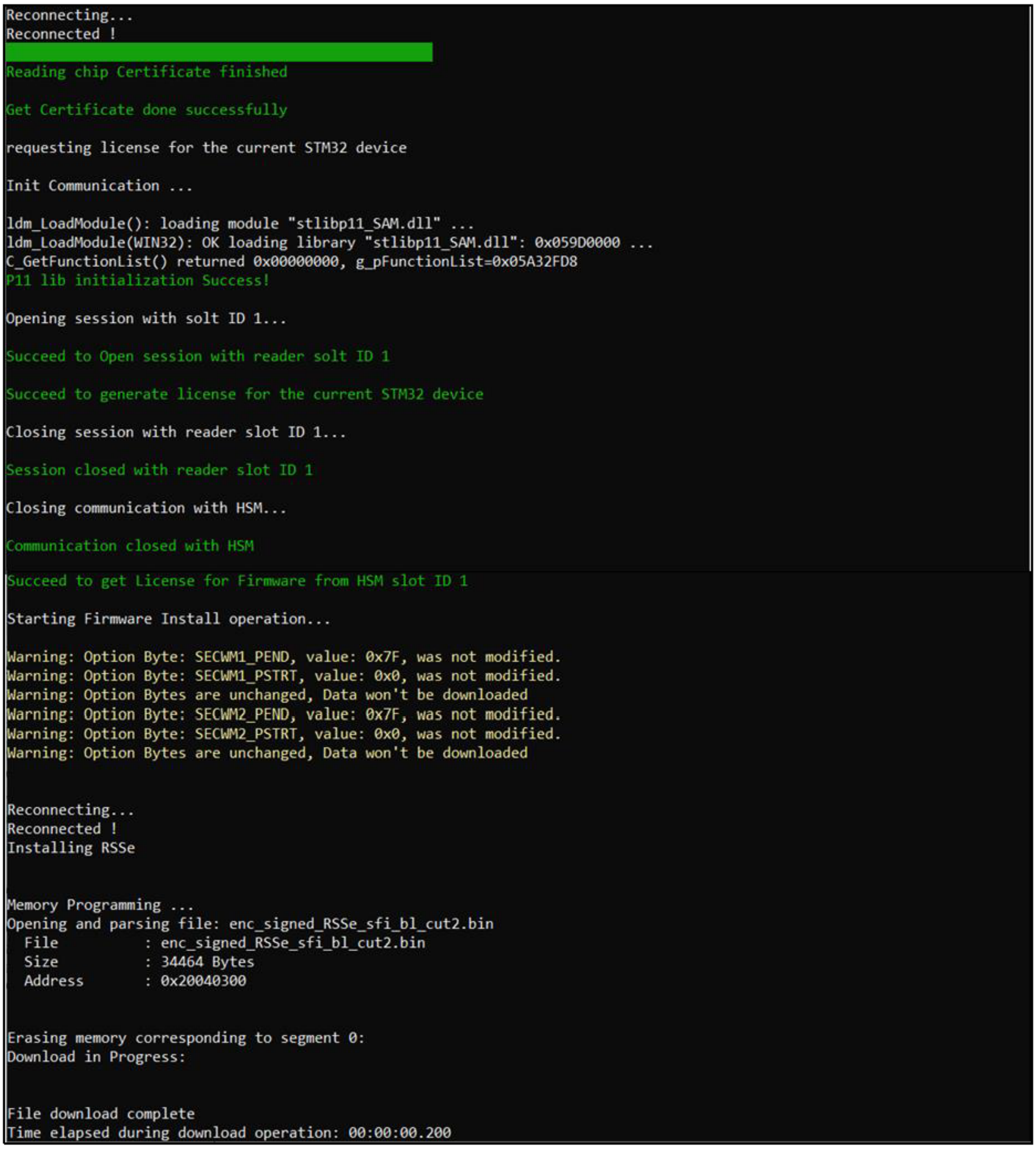
Note: The RSSe and the corresponding binary file are located in the STM32CubeProgrammer bin/RSSe/U5 *folder.*

Figure 46 and *Figure 47* show the STM32CubeProgrammer command used for the SFI install process via SWD execution.

Figure 46. SFI installation via SWD execution (1)

```
Reconnecting...
Reconnected !

Reading chip Certificate finished

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

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

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

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

Starting Firmware Install operation...

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


Reconnecting...
Reconnected !
Installing RSSe


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


Erasing memory corresponding to segment 0:
Download in Progress:


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

Figure 47. SFI installation via SWD execution (2)



7 Example of SFI programming scenario for STM32WBA5x and STM32WBA6x

7.1 Scenario overview

The actual user application to be installed on the STM32WBA5x/6x device. The application was encrypted using the STPC. The OEM provides tools to the CM to get the appropriate license for the concerned SFI application

7.2 Hardware and software environment

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

- STM32WBA5x/6x board with bootloader and RSS programmed
- RS-232 cable for SFI programming via UART
- Micro-B USB for debug connection
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v1.7.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer package available from *www.st.com*
  - For STM32WBA5x, v2.16.0 (or greater)
  - For STM32WBA6x, v2.19.0 (or greater)
- STM32HSM-V2

Note: Refer to [4] or [5] for the supported operating systems and architectures.

7.3 Step-by-step execution

7.3.1 Build OEM application

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

7.3.2 Perform the SFI generation (GUI mode)

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

- An OEM firmware
- A .csv file containing option bytes configuration
- A128-bit AES encryption key
- A 96-bit nonce

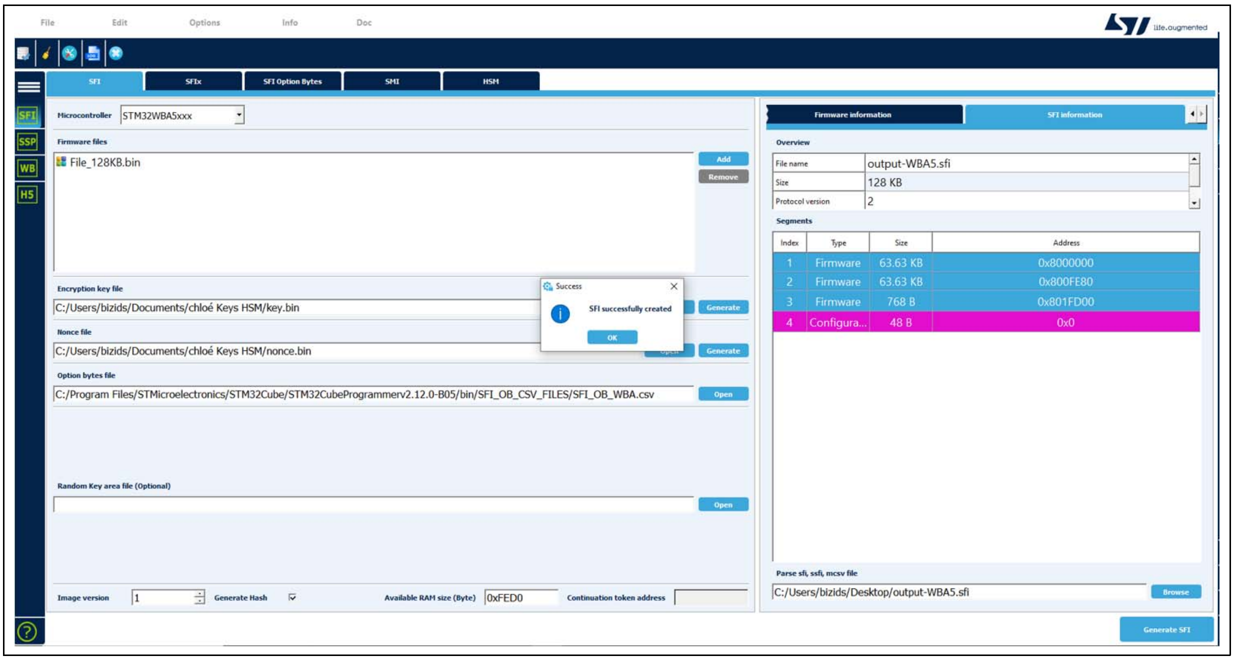
Note: STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product. It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual. In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

In addition, a programmed STM32HSM card must be inserted in the PC. An “output-WBA5.sfi” or “output-WBA6.sfi” image is then generated.

Figure 48 shows the STPC GUI during SFI generation.

Figure 48. STPC GUI during the SFI generation (STM32WBA5x)



Note: To perform STM32HSM programming for license generation using STPC (GUI and CLI modes), refer to Section 11.3.3: Performing STM32HSM programming for license generation using STPC (GUI mode) and Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode).

7.3.3 Programming input conditions

Before performing an SFI install on STM32WBA5x/6x devices, make sure that:

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

7.3.4 Perform the SFI install using STM32CubeProgrammer

In this section, the STM32CubeProgrammer tool is used in CLI mode to program the SFI image “output-WBA5.sfi” or “output-WBA6.sfi” already created in the previous section.

STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on smartcards) to generate a license for the connected STM32 device during the SFI install process.

Using the UART interface

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

STM32_Programmer_CLI.exe -c port=COM204 -sfi protocol=static "/output-WBAx.sfi" hsm=1 slot=1 -rsse "< RSSe_path >"

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

Figure 49 shows the STM32CubeProgrammer command used for the SFI install process via UART execution.

Figure 49. SFI installation via UART execution using CLI (1)

```
Serial Port COM204 is successfully opened.
Port configuration: parity = even, baudrate = 115200, data-bit = 8,
                     stop-bit = 1.0, flow-control = off
Activating device: OK
Board       : --
Chip ID: 0x492
BootLoader protocol version: 3.1
Device name : STM32WBA52/54/55
Flash size  : 1 MBytes (default)
Device type : MCU
Revision ID : --
Device CPU  : Cortex-M33
  Protocol Information         : static

  SFI File Information         :

     SFI file path             : C:\Users\bizids\Desktop\output-WBA5.sfi
     SFI HSM slot ID           : 1
     SFI header information    :
         SFI protocol version        : 2
         SFI total number of areas   : 4
         SFI image version           : 1
     SFI Areas information     :

     Parsing Area 1/4    :
         Area type                   : F
         Area size                   : 65152
         Area destination address    : 0x8000000

     Parsing Area 2/4    :
         Area type                   : F
         Area size                   : 65152
         Area destination address    : 0x800FE80

     Parsing Area 3/4    :
         Area type                   : F
         Area size                   : 768
         Area destination address    : 0x801FD00

     Parsing Area 4/4    :
         Area type                   : C
         Area size                   : 48
         Area destination address    : 0x0

Reading the chip Certificate...


Requesting Chip Certificate from device ...

Reading chip Certificate finished

Get Certificate done successfully

requesting license for the current STM32 device
```

MSv73794

Figure 50. SFI installation via UART execution using CLI (2)

```
Init Communication ...

ldm_LoadModule(): loading module "C:/Program Files/STMicroelectronics/STM32Cube/STM32C
ldm_LoadModule(WIN32): OK loading library "C:/Program Files/STMicroelectronics/STM32Cu
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x5CF643F0
P11 lib initialization Success!

Opening session with slot ID 1...

Succeed to Open session with reader slot ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...



Reconnection after Option Bytes Programming
Time elapsed during option Bytes configuration: 00:00:02.177
Warning: Option Byte: SECWM_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.006


Reconnection after Option Bytes Programming
Time elapsed during option Bytes configuration: 00:00:02.193
Warning: Option Byte: SRAM2_RST, value: 0x1, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.006


Reconnection after Option Bytes Programming
Verifying Read Out Protection...
Time elapsed during option Bytes configuration: 00:00:02.293
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi_WBA5_1M.bin
  File          : enc_signed_RSSe_sfi_WBA5_1M.bin
  Size          : 31.92 KB
  Address       : 0x20008100


Erasing memory corresponding to segment 0:
Not flash Memory : No erase done
Download in Progress:
```

MSv73795

Figure 51. SFI installation via UART execution using CLI (3)

```
File download complete
Time elapsed during download operation: 00:00:03.519
Get RSSe status...
Warning: Option Byte: TZEN, value: 0x1, was not modified.
Warning: Option Byte: nBoot0, value: 0x0, was not modified.
Warning: Option Byte: nSWBoot0, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.007


RSS version = 1.5.0


RSSe version = 1.1.0

Starting SFI

Processing license...
Get RSSe status...
Processing Image Header
Get RSSe status...
Processing Area 1...
Get RSSe status...
Area Address = 0x8000000
Area Type    = F
Processing Area 2...
Get RSSe status...
Area Address = 0x800FE80
Area Type    = F
Processing Area 3...
Get RSSe status...
Area Address = 0x801FD00
Area Type    = F
Processing Area 4...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file C:\Users\bizids\Desktop\output-WBA5.sfi Install Operation Success

Time elapsed during SFI install operation: 00:00:34.373
```

MSv73796

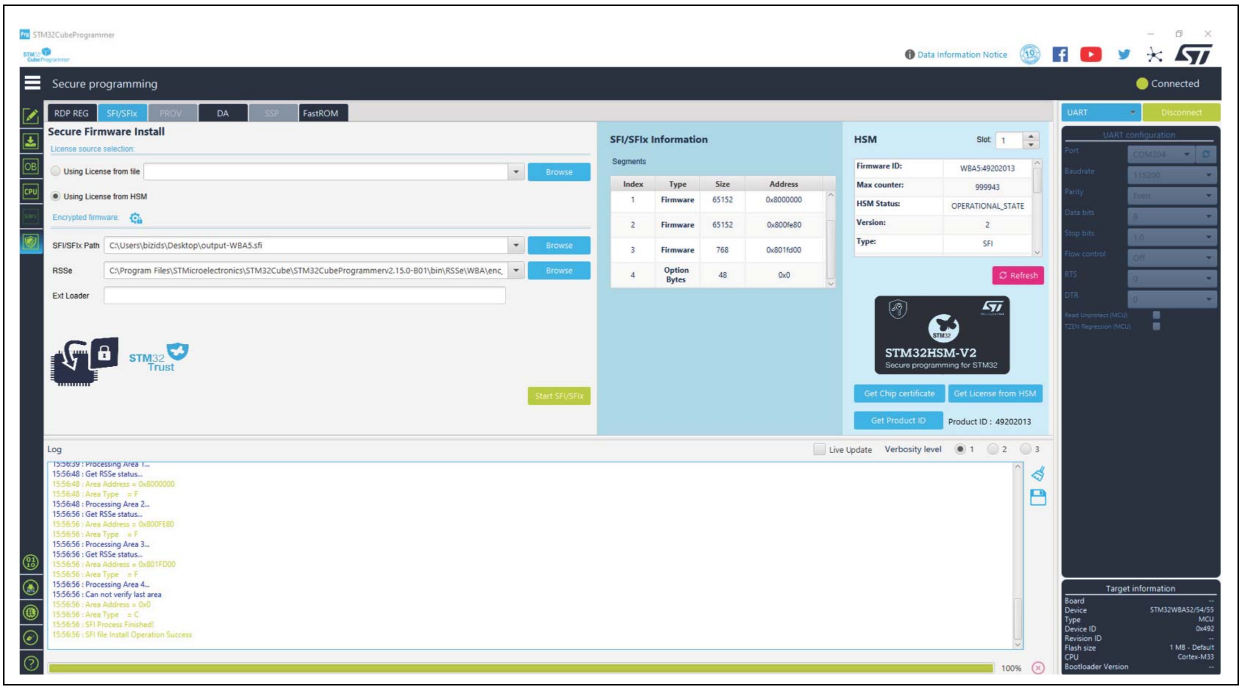
Graphical user interface mode (STM32WBA5x example)

Open the STM32CubeProgrammer and connect the board through the UART interface with the right COM port. Press on the "Security" panel and select the SFI/SFIx from the tab options with the following inputs:

- License source selection: "Using License from HSM"
- SFI/SFIx path: output-WBA5.sfi
- RSSe: /RSSe/WBA/enc_signed_RSSe_sfi_WBA5_1M.bin

Click on the "Start SFI/SFIx" button to launch the SFI installation.

Figure 52. STM32WBA5x SFI successful programming via UART interface using GUI



8 Example of SFIA programming scenario for STM32WBA5x

8.1 Scenario overview

SFIA is an SFI operation without a mass erase. It means that the user should perform an SFI install when all the flash memory is empty, or when the data written in the user flash memory is outside of the SFI firmware to install. (For more details refer to *[1]*).

In this example, the SFI is installed when the flash memory is already empty. The actual user application to be installed on the STM32WBA5x device. The application is encrypted using the STPC. The OEM provides the tools to the CM to get the appropriate license for the concerned SFIA application.

8.2 Hardware and software environment

For a successful SFIA programming, the following hardware and software prerequisites are needed:

- An STM32WBA5x board with boot loader and RSS programmed
- An RS-232 cable for SFIA programming via UART
- A Micro-B USB for debug connection
- A PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v2.17.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.17.0 (or greater) package available from *www.st.com*
- STM32HSM-V2 (To generate the SFIA license)

Note: Refer to [4] or [5] for the supported operating systems and architectures.

8.3 Step-by-step execution

8.3.1 Build an OEM application

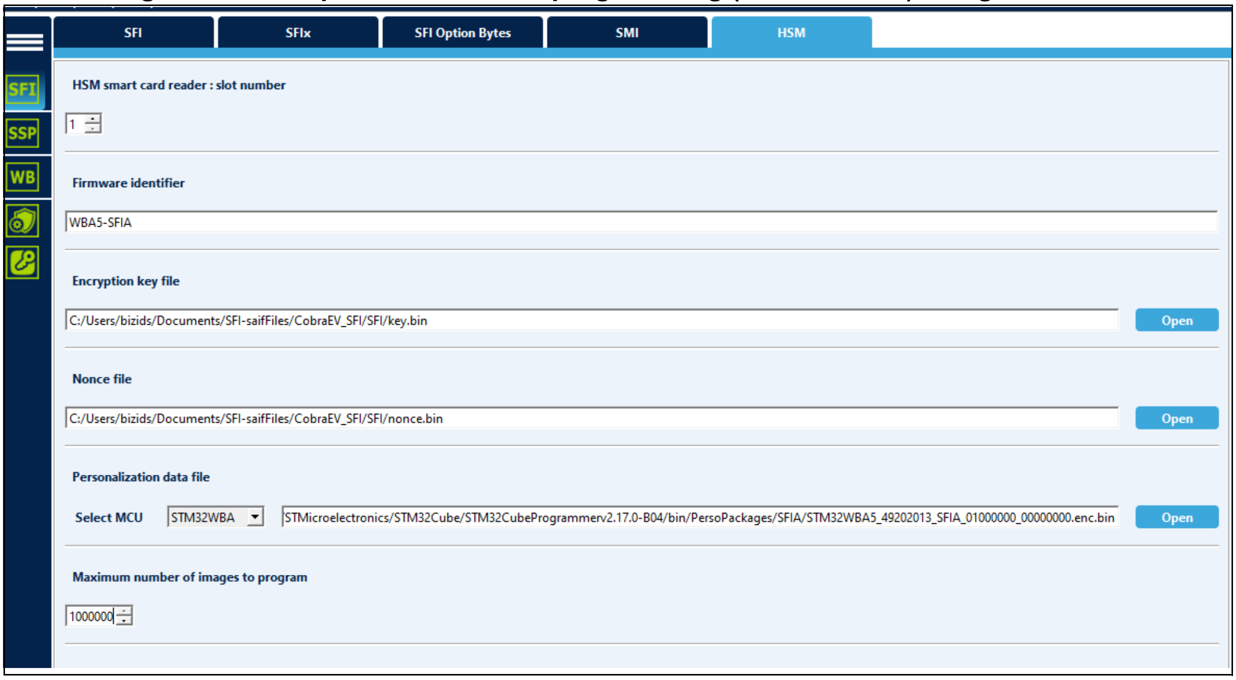
OEM application developers can use any IDE to build their own firmware.

8.3.2 Perform the STM32HSM programming for the SFIA license generation (GUI mode)

The STM32 Trusted Package Creator tool provides all the personalization package files ready to be used on the SFI/SFIA/SFIx and SSP flows. To get all the supported **SFIA** packages, go to the **PersoPackages/SFIA** directory in the install path of the tool. Each file name starts with a number, which is the product ID of the device. Select the correct one.

In this case, select: STM32WBA5_49202013_**SFIA**_01000000_00000000.enc.bin to program the STM32HSM card.

Figure 53. Example of STM32HSM programming (SFIA License) using STPC GU



8.3.3 Perform the SFI generation (GUI mode)

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

The steps described in *Section 7.3.2: Perform the SFI generation (GUI mode)* can be followed.

8.3.4 Programming input conditions

Before performing an SFI install on STM32WBA5x devices, the user must ensure that:

- The flash memory is erased.
- No WRP zone is active, otherwise it should be destroyed.
- The chip supports security (a security bit must be present in the option bytes).
- The security is disabled.

8.3.5 Perform the SFI installation using STM32CubeProgrammer

In this section, the STM32CubeProgrammer tool is used in CLI mode to program the SFI image "*output-WBA5.sfi*" that was created in the previous section.

The STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on smartcards) to generate a license for the connected STM32 device during the SFI install process.

The user must ensure that all the input conditions are respected, and then follow the steps described in *Section 7.3.4: Perform the SFI install using STM32CubeProgrammer*.

9 Example of SFI programming scenario for STM32H5

9.1 Scenario overview

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

- The encrypted STM32H5 firmware
- A global license binary
- STM32CubeProgrammer

The untrusted manufacturer is then required to securely program the encrypted firmware using these inputs.

9.2 Hardware and software environment

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

- STM32H5-based board with bootloader and RSS programmed
- SFI programming via UART (use RS-232 cable or ST-LINK VCOM)
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- aSTM32 Trusted Package Creator v2.14.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.14.0 (or greater) package available from *www.st.com*

Note: Refer to [4] or [5] for the supported operating systems and architectures.

9.3 Step-by-step execution

9.3.1 Build OEM application

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

9.3.2 Perform the SFI generation (GUI mode)

The first step to install the secure firmware on STM32H5 devices is the encryption of the user OEM firmware using the STM32 Trusted Package Creator tool.

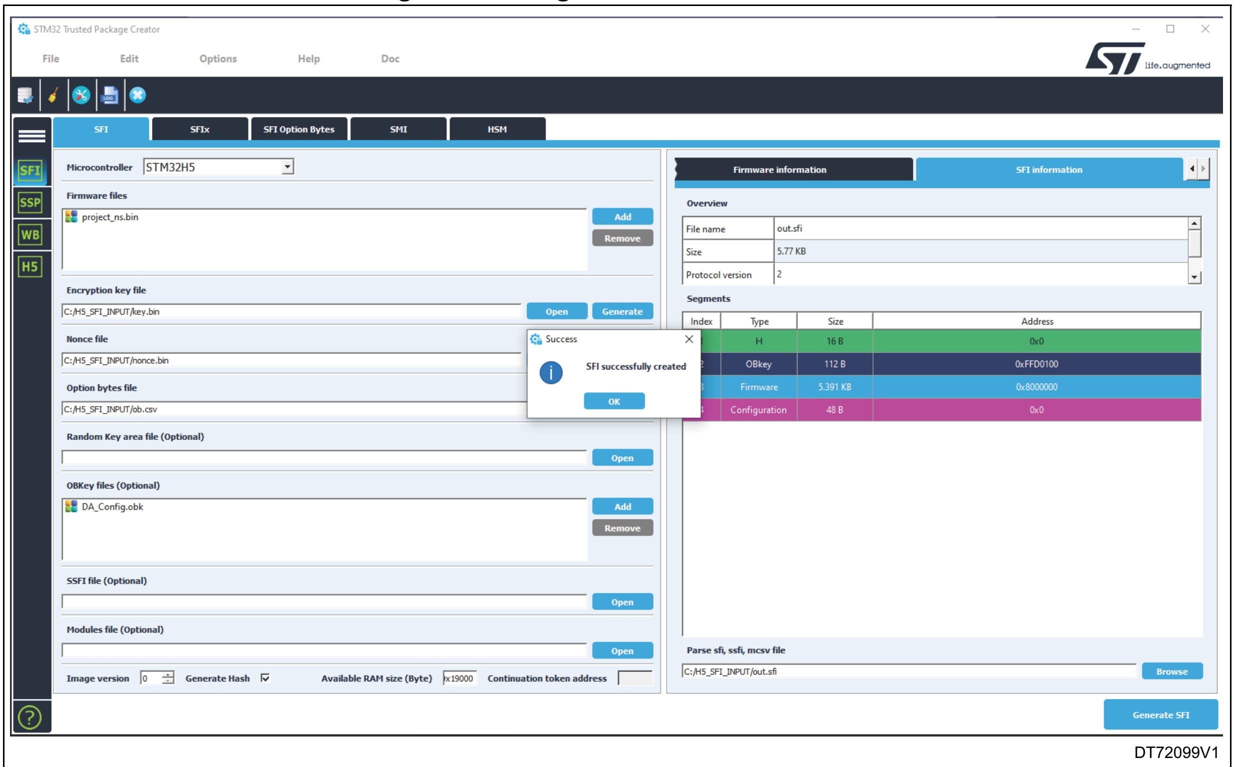
This step is done by including the following files in the STPC tool:

- An OEM firmware
- A .csv file containing option bytes configuration
- A 128-bit AES encryption key
- A 96-bit nonce
- OBKey files for device configuration (optional)
- An SSFI file to integrate the STMicroelectronics SFI image (optional, only for STM32H573xx)

Note: It is recommended to use the "SFI Option Bytes" feature from the "H5" panel of the STM32 Trusted Package Creator tool to obtain the option bytes file (.csv file).

Note: If you want to reopen the device using the Debug Authentication mechanism, a DA ObKey file must be included in the SFI image, otherwise the device becomes inaccessible.

Figure 54. SFI generation for STM32H5



9.3.3 Programming input requirements

Before performing an SFI install on STM32H5 devices, make sure that:

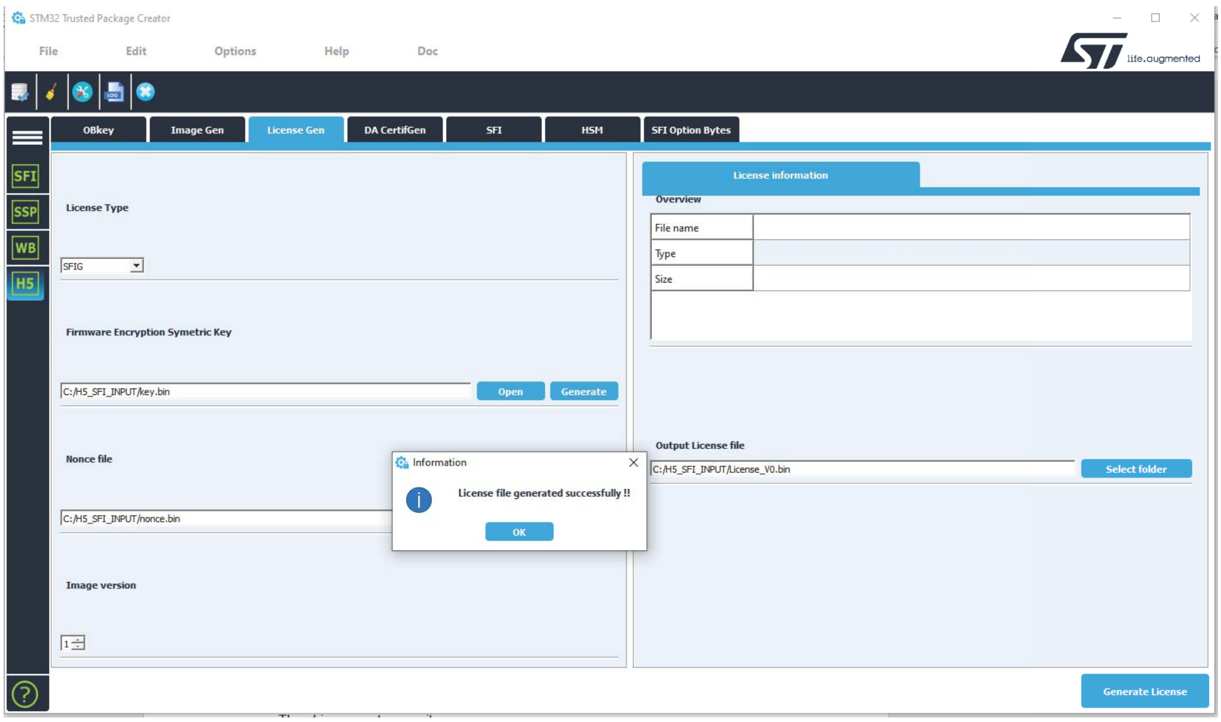
- Flash memory erased
- Chip supporting cryptography for a Secure Manager usage
- Product state open: 0xED
- Boot on bootloader: UART interface
- RSSe binary
- STMicroelectronics global license file (no need for an STM32HSM card in this use case)

Note: The RSSe binary file is in the STM32CubeProgrammer bin/RSSe/H5 folder.

Note: To embed an SSFI image into the SFI image, it is recommended to follow a specific secure sequence and choose an adequate start address of the nonsecure application that depends on the SSFI configuration. See the details in STM32CubeH5 MCU Package available from www.st.com.

To generate an STMicroelectronics global license binary, use the “H5” panel of the STM32 Trusted Package Creator GUI and select the "License Gen" option. Then, include the same key and nonce previously used to generate the SFI image (see the figure below).

Figure 55. STMicroelectronics global license generation for STM32H5



DT73500V1

9.3.4 Perform the SFI install using STM32CubeProgrammer

In this section, the STM32CubeProgrammer tool is used in CLI mode to program the SFI image “out.sfi” already created in the previous section.

STM32CubeProgrammer communicates with the device through the UART interface after it is confirmed that all the input conditions are respected.

Note that the same operation is possible using ST-LINK (SWD/JTAG) or any bootloader interface.

Command-line mode

Open a cmd terminal, go to /bin in the install path, and then launch the following command:

```
STM32_Programmer_CLI.exe -c port=COM8
-sfi "out.sfi" hsm=0 "ST_Global_License_V0.bin"
-rsse "\RSSe\H5\enc_signed_RSSe_SFI_STM32H5_v2.0.0.0"
```

Figure 56 shows the SFI execution traces.

Figure 56. STM32H5 SFI successful programming via CLI

```
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi_H5_2M.bin
  File          : enc_signed_RSSe_sfi_H5_2M.bin
  Size          : 45.89 KB
  Address       : 0x20050300


Erasing memory corresponding to segment 0:
Not flash Memory : No erase done
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:05.242
Get RSSe status...


RSS version = 1.12.0


RSSe version = 1.0.0

Starting SFI

Processing license...
Get RSSe status...
Processing Image Header
Get RSSe status...
Processing Area 1...
Get RSSe status...
Area Address = 0x0
Area Type    = H
Processing Area 2...
Get RSSe status...
Area Address = 0xFFD0100
Area Type    = O
Processing Area 3...
Get RSSe status...
Area Address = 0x8000000
Area Type    = F
Processing Area 4...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file out.sfi Install Operation Success

Time elapsed during SFI install operation: 00:00:22.172
```

DT73501V1

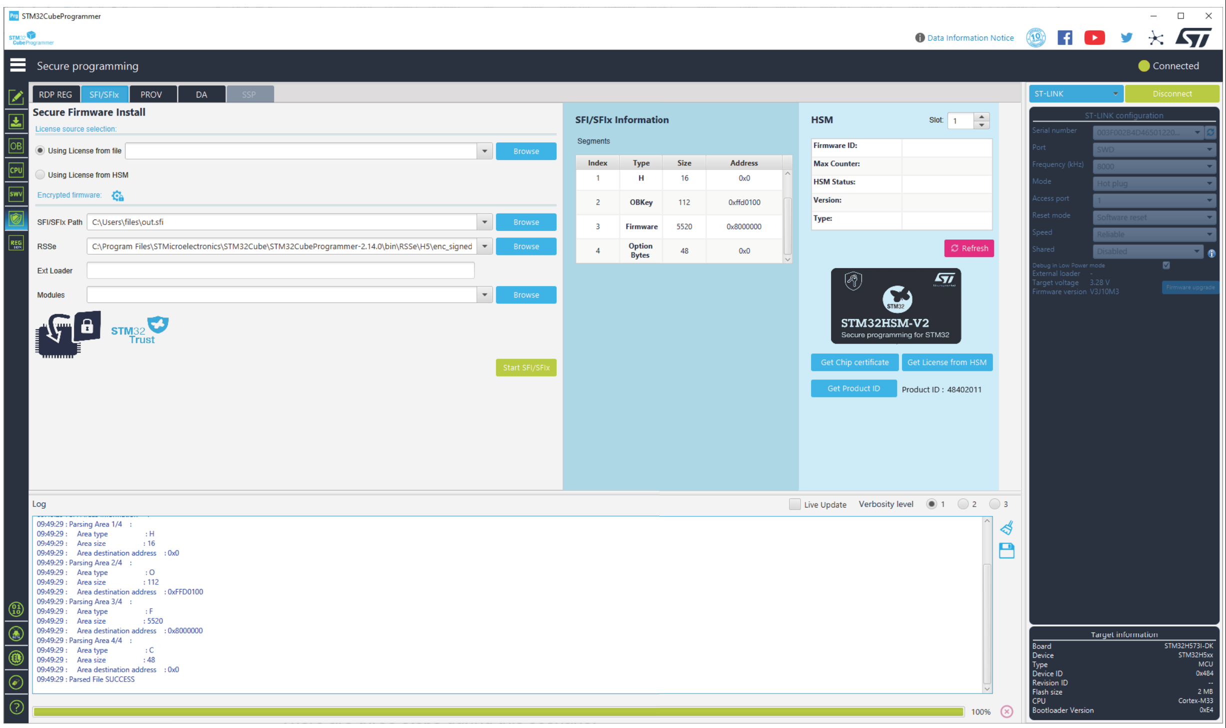
Graphical user interface mode

Open the STM32CubeProgrammer and connect the board through the UART interface with the right COM port. Press on the "Security" panel and select the SFI/SFIx from the tab options with the following inputs:

- License source selection: "Using License from file"
- SFI/SFIx path: out.sfi
- RSSe: \RSSe\H5\ enc_signed_RSSe_SFI_STM32H5_v2.0.0.0.bin

Click on the "Start SFI/SFIx" button to launch the SFI installation.

Figure 57. STM32H5 SFI successful programming via GUI



DT73502V1

10 Example of SFI programming scenario for STM32H7Rx/7Sx

10.1 Scenario overview

There are three steps during this scenario:

- Generate STM32H7Rx/7Sx encrypted firmware using the STPC
- STM32HSM card provisioning via STPC
- Use STM32CubeProgramer to perform the SFI process.

10.2 Hardware and software environment

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

- An STM32H7Rx/7Sx-based board and system flash security package (SFSP) v1.1.0 or greater
- USB Type-C® cable for SWD connection
- A PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- An STM32 Trusted Package Creator v2.16.0 (or later) package available from *www.st.com*
- An STM32CubeProgrammer v2.16.0 (or later) package available from *www.st.com*
- An STM32HSM-V2 smartcard

Note: Refer to [4] or [5] for the supported operating systems and architectures.

10.3 Step-by-step execution

10.3.1 Build an OEM application

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

10.3.2 Perform the SFI generation (GUI mode)

The first step to install the secure firmware on STM32H7Rx/7Sx devices is the encryption of the user OEM firmware using the STM32 Trusted Package Creator tool.

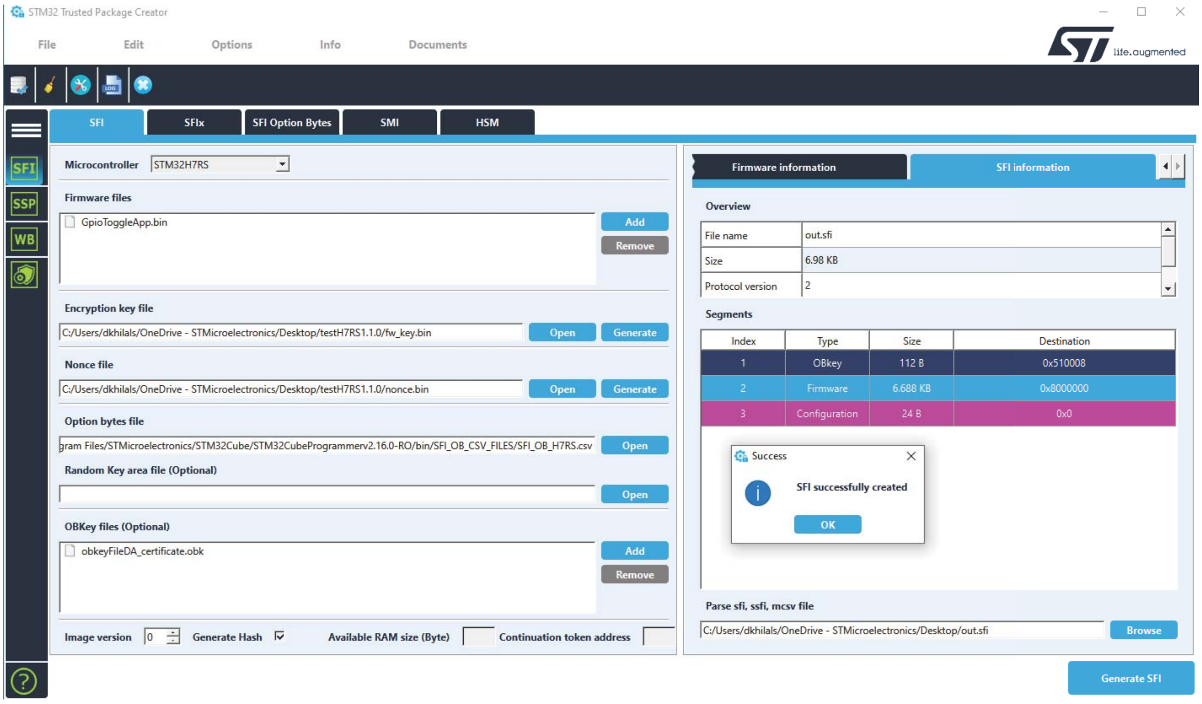
This step is done by including the following files in the STPC tool:

- An OEM firmware
- A .csv file containing option bytes configuration
- A 128-bit AES encryption key
- A 96-bit nonce
- Random key area file (optional)
- OBKey files for device configuration (optional)

Note: It is recommended to use the "SFI Option Bytes" feature of the STM32 Trusted Package Creator tool to obtain the option bytes file (.csv file).

Note: If you want to reopen the device using the Debug Authentication mechanism, a DA ObKey file must be included in the SFI image, otherwise the device becomes inaccessible.

Figure 58. SFI generation for STM32H7Rx/7Sx



10.3.3 Programming input requirements

Before performing an SFI install on STM32H7Rx/7Sx devices, make sure that:

- Product state is open: 0x39
- A ready generated SFI image using the STPC tool
- RSSe binary
- STMicroelectronics global license file (no need for an STM32HSM card in this use case)

Note: Using a non STM32H7Rx/7Sx sfi image might result in errors or issues during the installation process.

Note: The RSSe binary file is in the STM32CubeProgrammer bin/RSSe/H7RS folder.

10.3.4 Perform the SFI install using STM32CubeProgrammer

In this section, the STM32CubeProgrammer tool is used in CLI mode to program the SFI image *"out.sfi"* already created in the previous section.

STM32CubeProgrammer communicates with the device through the SWD interface after it is confirmed that all the input conditions are respected.

Note that the same operation is possible using ST-LINK (SWD/JTAG) or any bootloader interface.

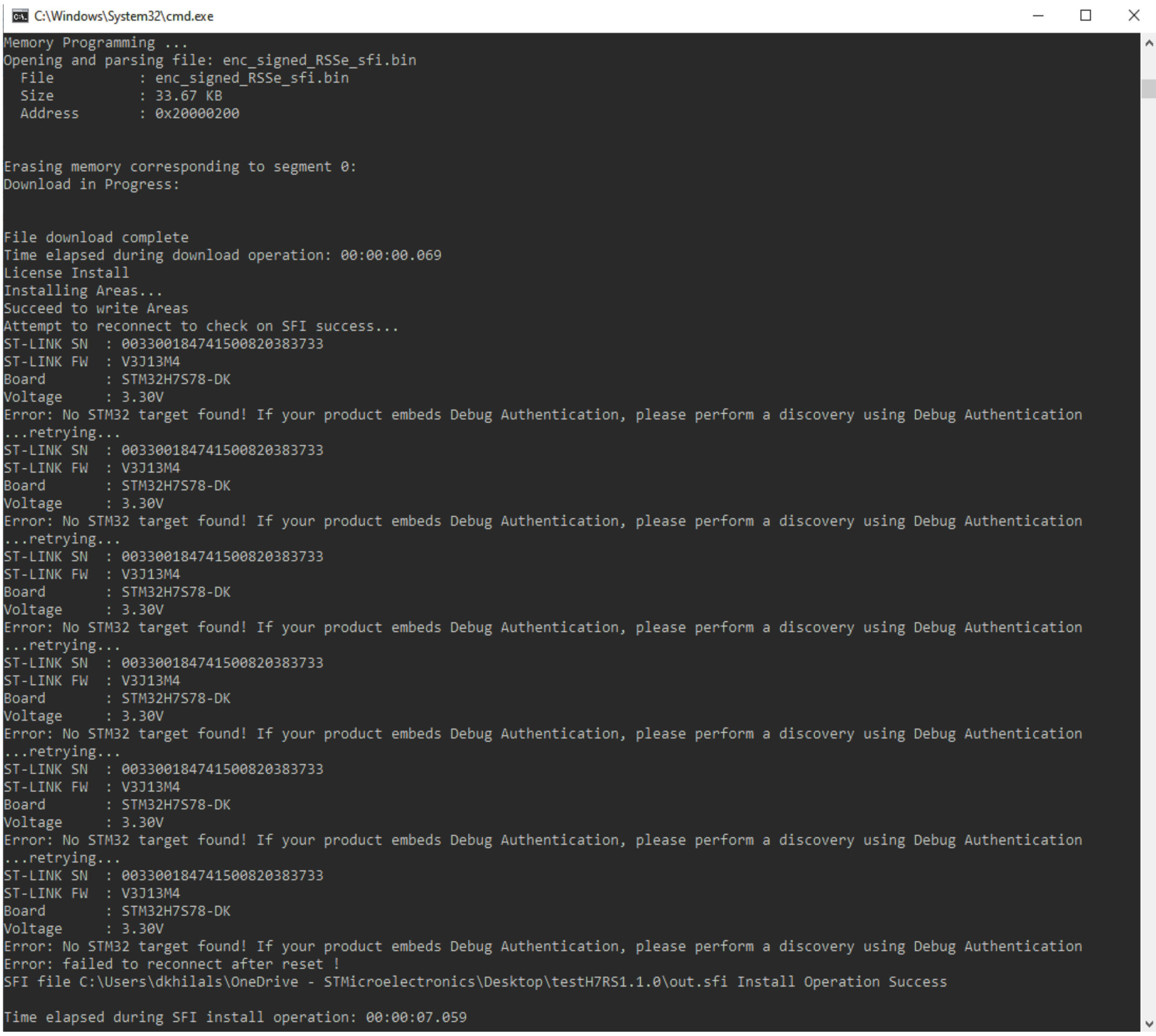
Command-line mode

Open a cmd terminal, go to /bin in the install path, and then launch the following command:

```
STM32_Programmer_CLI.exe -c port=swd mode=hotplug -sfi "out.sfi"
hsm=0 "ST_Global_License_V0.bin" -rsse "\RSSe\H7RS\
enc_signed_RSSe_sfi.bin"
```

Figure 5 shows the SFI execution traces.

Figure 59. STM32H7Rx/7Sx SFI successful programming via CLI



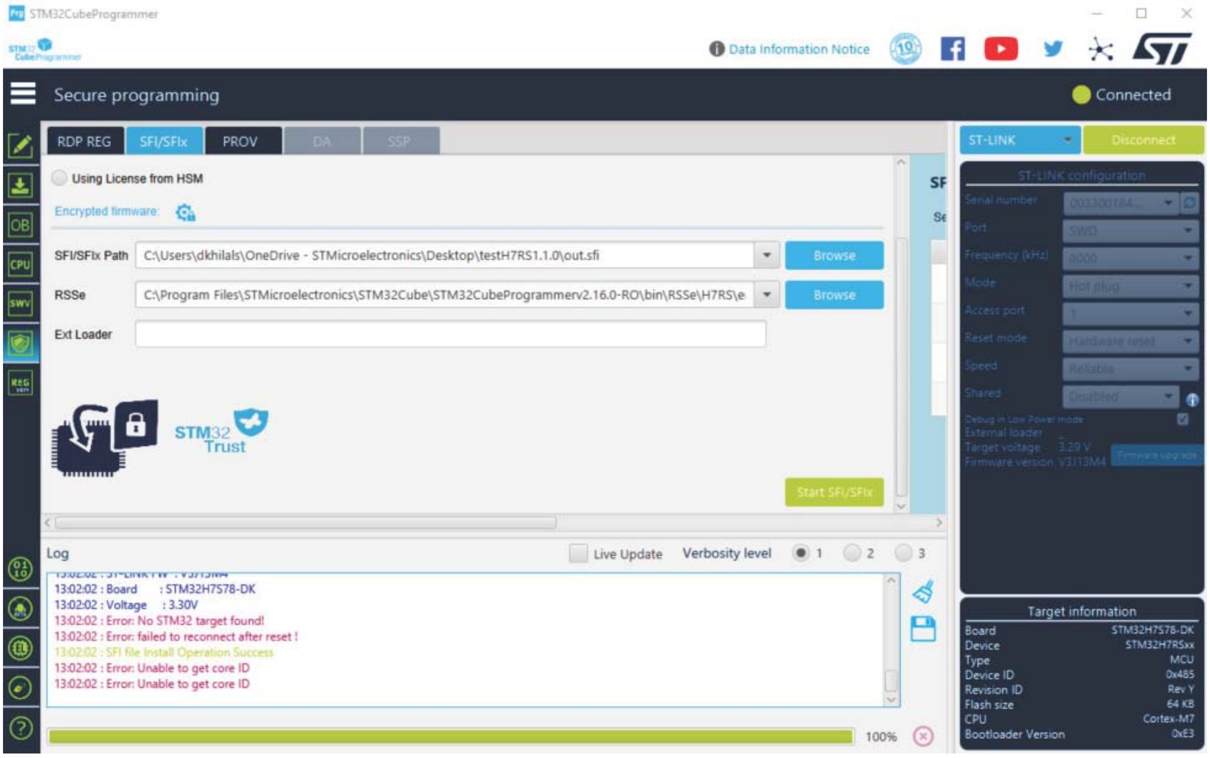
Graphical user interface mode

Open the STM32CubeProgrammer and connect the board through the SWD. Go to the security panel and select the SFI/SFIx from the tab options with the following inputs:

- License source selection: "Using License from file"
- SFI/SFIx path: out.sfi
- RSSe: \RSSe\H7RS\ enc_signed_RSSe_sfi.bin

Click on the "Start SFI/SFIx" button to launch the SFI installation.

Figure 60. STM32H7Rx/7Sx SFI successful programming via GUI



11 Example of SFIx programming scenario for STM32H7

11.1 Scenario overview

There are three steps during this scenario:

- Generate an SFIx image using the STPC.
- Provisioning STM32HSM card via STPC.
- Use the STM32CubeProgrammer to perform the SFIx process.

Once this scenario is successfully installed on the STM32H7B3I-EVAL, follow the steps below:

- Write internal firmware data in the internal flash memory starting at the address 0x08000000.
- Write external firmware data in the external flash memory starting at the address 0x90000000.
- Verify that the option bytes were correctly programmed (depends on area C).

11.2 Hardware and software environment

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

- STM32H7B3I-EVAL board containing external flash memory.
- Micro-B USB for debug connection.
- PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v2.3.0 (or greater) package available from www.st.com
- STM32HSM-V1.1 card

Note: Refer to [4] or [5] for the supported operating systems and architectures.

11.3 Step-by-step execution

11.3.1 Build OEM application

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

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

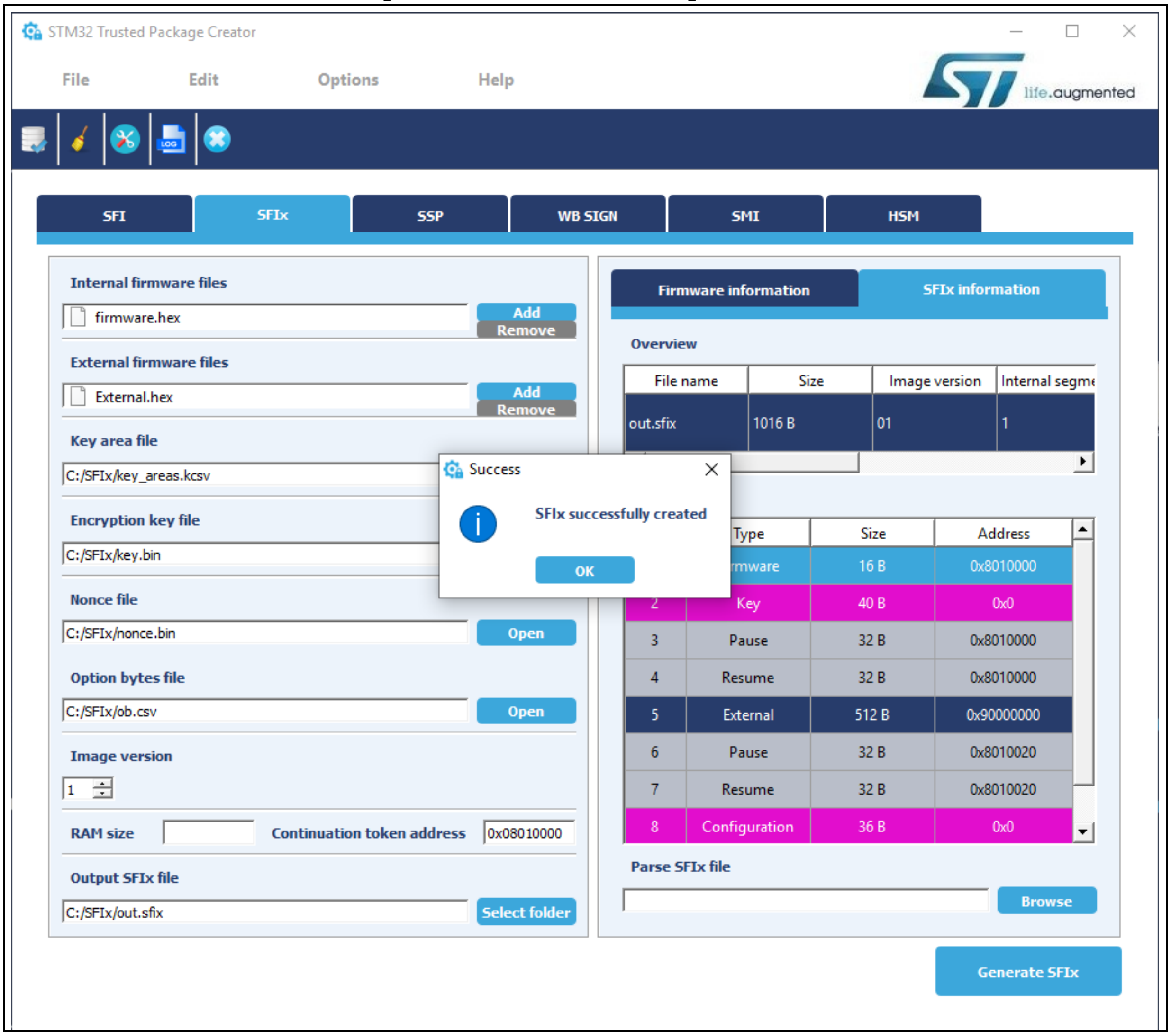
11.3.2 Perform the SFIx generation (GUI mode)

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

Note: *STM32CubeProgrammer v2.8.0 and later provide one option byte file example for each product.*
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES
The option bytes are described in the product reference manual.
In the case of customization of a provided example file, care must be taken not to change the number of rows, or their order.

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

Figure 61. Successful SFIx generation



11.3.3 Performing STM32HSM 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.

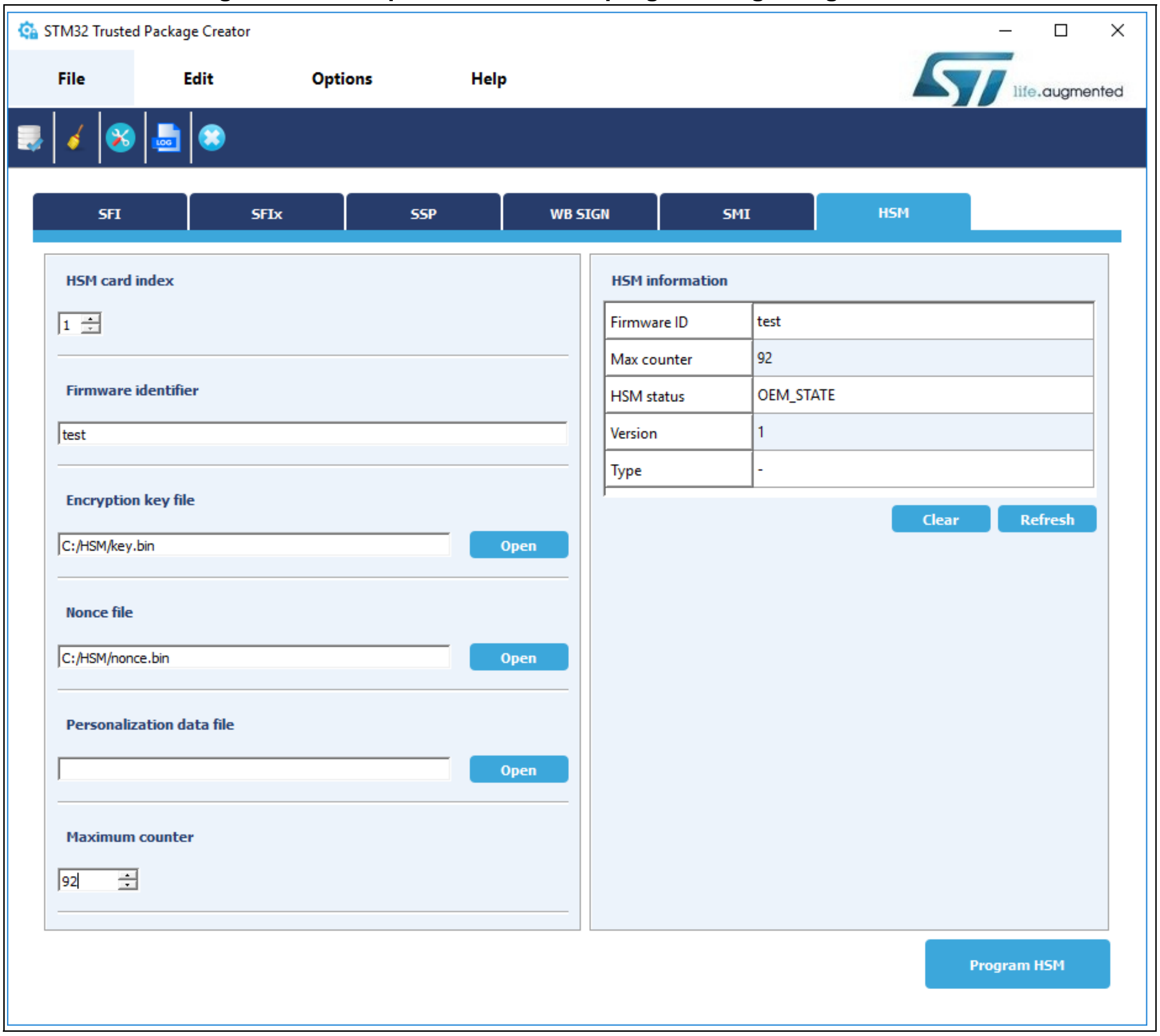
In this example, STM32HSMs are used as license generation tools in the field. See *Section 3.1.2: License mechanism* for STM32HSM use and programming.

Figure 62 shows an example for STM32HSM programming by OEM to be used for SFIx install.

The maximum number of licenses delivered by the STM32HSM in this example is 1000.

This example uses STM32HSM-V1. The STM32HSM 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 62. Example of STM32HSM programming using STPC GUI



Note: When using STM32HSM-V1, the “Personalization data file” field is ignored when programming starts. It is only used with STM32HSM-V2.

When the card is successfully programmed, a popup window message “HSM successfully programmed” appears, and the STM32HSM is locked. Otherwise, an error message is displayed.

11.3.4 Performing STM32HSM programming for license generation using STPC (CLI mode)

Refer to *Section 4.3.5: Performing STM32HSM programming for license generation using STPC (CLI mode)*.

11.3.5 Programming input conditions

Before performing an SFIx install, make sure that:

- Use the JTAG/SWD interface.
- No PCROPed zone is active, otherwise disable it.
- The chip must support security (a security bit must be present in the option bytes).
- The SFIx image must be encrypted by the same key/nonce used in the STM32HSM provisioning.

11.3.6 Perform the SFIx installation using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on smartcard) to generate a license for the connected STM32 device during SFIx install.

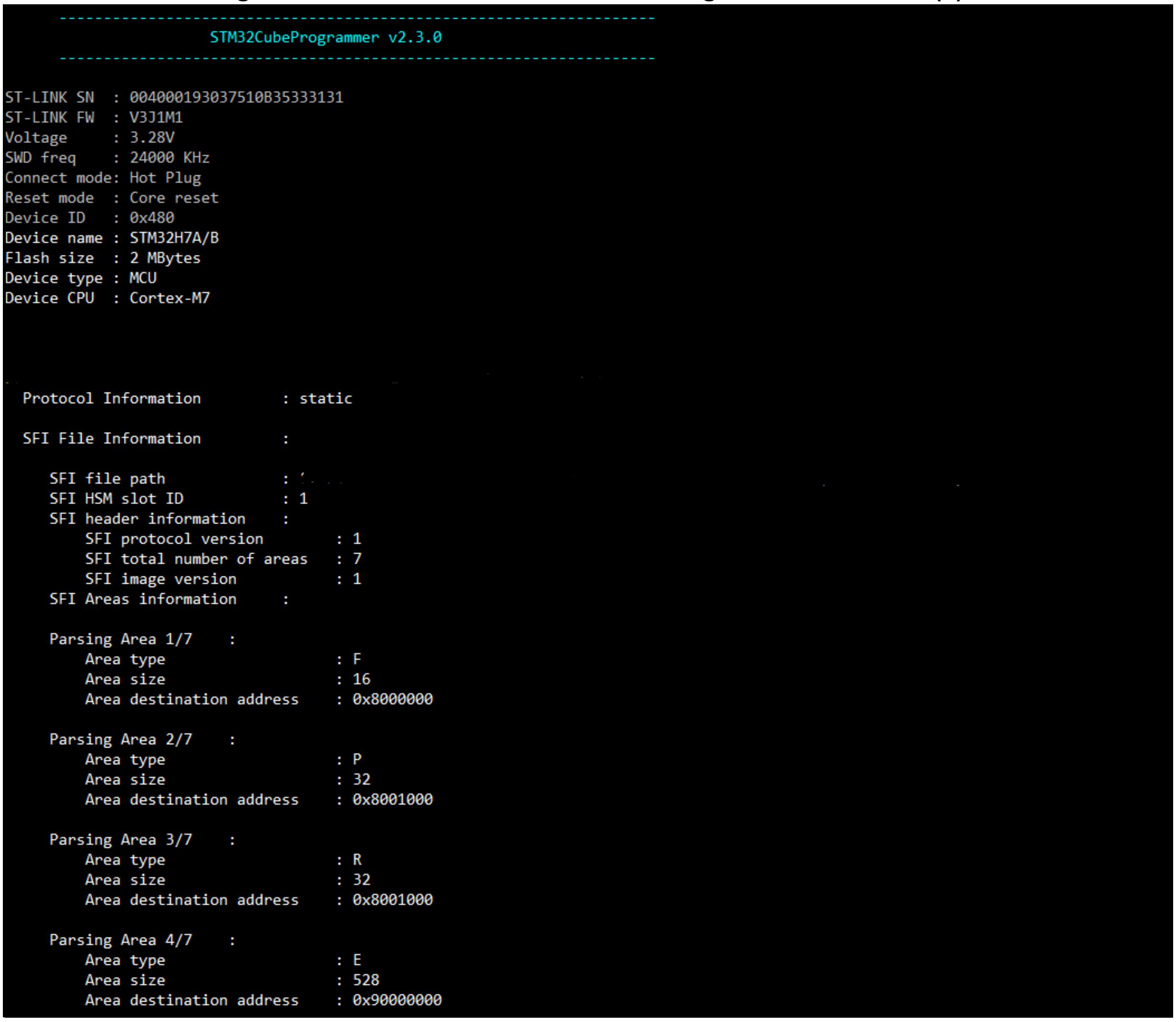
After making sure that all the input conditions are respected, open a cmd terminal and go to *<STM32CubeProgrammer_package_path>/bin*, then launch the following STM32CubeProgrammer command:

Using JTAG/SWD

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

Figure 63: SFIx installation success using SWD connection (1) *through* *Figure 66: SFIx installation success using SWD connection (4)* shows the SFIx install via SWD execution and the STM32HSM as license generation tool in the field.

Figure 63. SFIx installation success using SWD connection (1)



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

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



  Protocol Information         : static

  SFI File Information         :

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

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

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

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

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

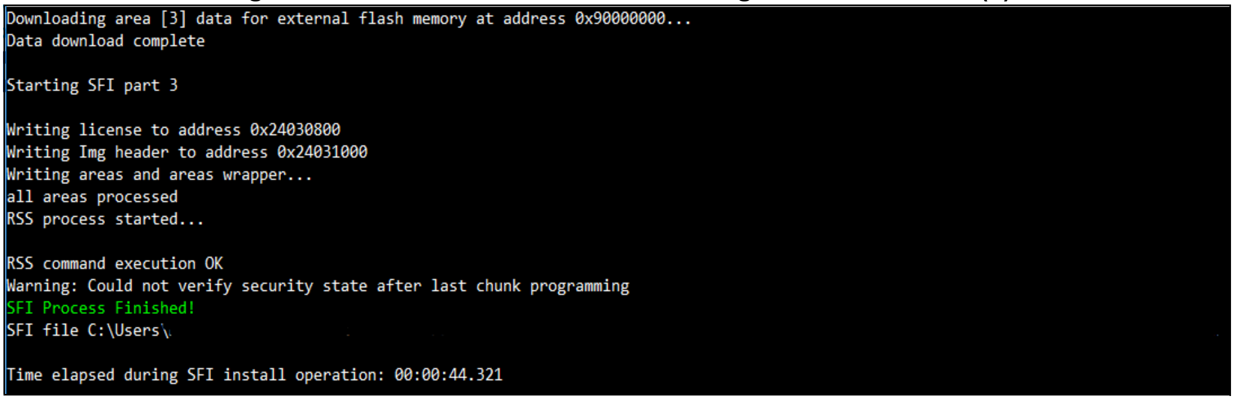
Figure 64. SFIx installation success using SWD connection (2)

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

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

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

Reading the chip Certificate...


Requesting Chip Certificate from device ...

Get Certificate done successfully

requesting license for the current STM32 device

Init Communication ...

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

Opening session with solt ID 1...

Succeed to Open session with reader solt ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get License for Firmware from HSM slot ID 1

Starting Firmware Install operation...

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

Figure 65. SFIx installation success using SWD connection (3)

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

Starting SFI part 1

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

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

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

Starting SFI part 2

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

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

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

Figure 66. SFIx installation success using SWD connection (4)



12 Example of SFIx programming scenario for STM32L5/STM32U5

12.1 Scenario overview

There are three steps during this scenario:

1. Generate an SFIx image using the STPC
2. STM32HSM card provisioning via STPC
3. Use STM32CubePrg to perform the SFIx process.

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

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

12.2 Hardware and software environment

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

- An STM32L5/STM32U5-based evaluation board containing external flash memory
- A Micro-B USB for debug connection
- A PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- An STM32 Trusted Package Creator v2.11.0 (or greater) package is available from *www.st.com*
- An STM32CubeProgrammer v2.11.0 (or greater) package is available from *www.st.com*
- An STM32HSM-V1.1 card

Note: Refer to [4] or [5] for the supported operating systems and architectures.

12.3 Step-by-step execution

12.3.1 Build an OEM application

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

12.3.2 Perform the SFIx generation (GUI mode)

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

Note: STM32CubeProgrammer v2.11.0 and later provide one option byte file example for each product.
It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI_OB_CSV_FILES

The option bytes are described in the product reference manual.

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

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

Use case 1: generation of SFIx without key area for STM32L5

Internal firmware files:

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

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

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

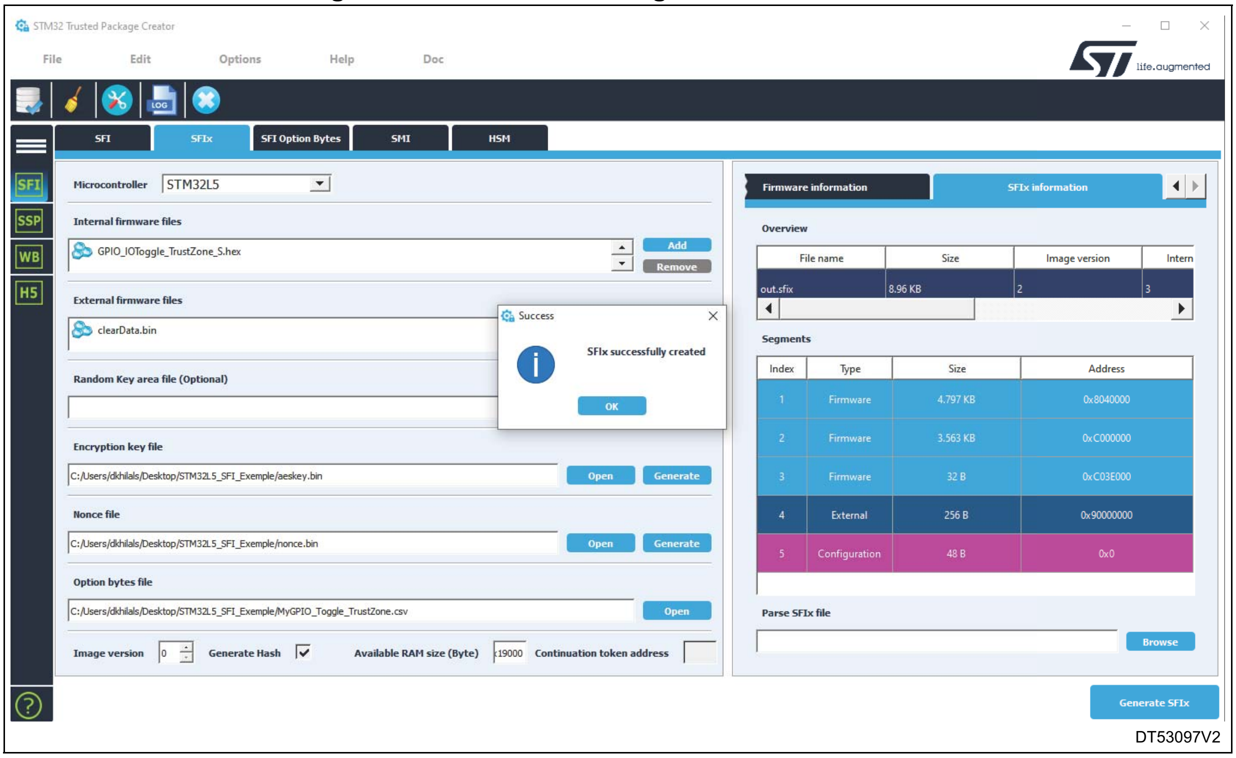
Encryption key: use the same key as STM32HSM.

Nonce file: use the same nonce as STM32HSM.

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

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

Figure 67. Successful SFIx generation use case 1

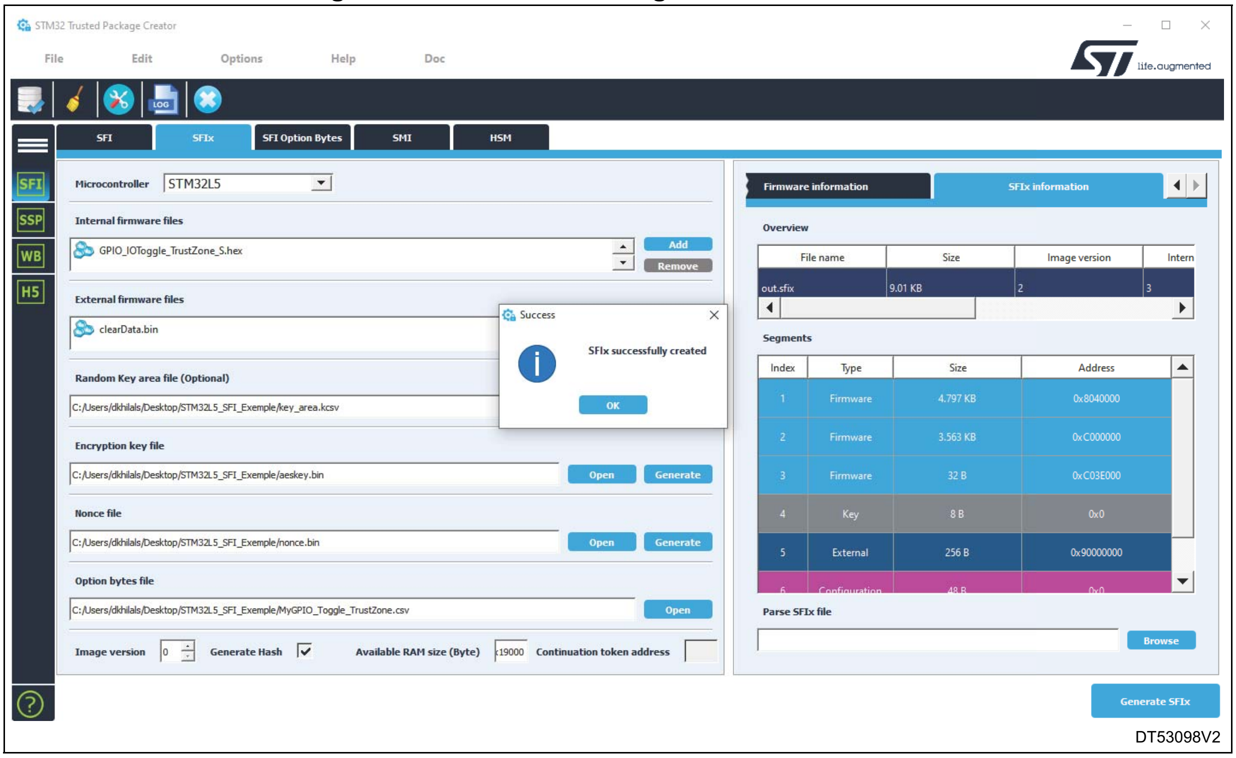


Use case 2: generation of SFIx with key area for STM32L5

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

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

Figure 68. Successful SFIx generation use case 2



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

Use case 3: generation of SFIx without key area for STM32U5

Find below an example for STM32U585xx.

Internal firmware files:

1. Add a nonsecure binary with a start address equal to 0x08100000.
2. Add an internal binary file at 0x0C000000 (application to be executed after downloading SFIx to verify full process success by blinking an LED).
3. Add an OTFDEC key binary at 0x0800A000 (to be used as the key in OTFD ENC-DEC).

External firmware files: add an external binary (at 0x70000000 for STM32U585xx) with these parameters:

- Region number = 4
- Region mode = 1
- Key address = 0x0800A000 (same as the OTFDEC key binary).

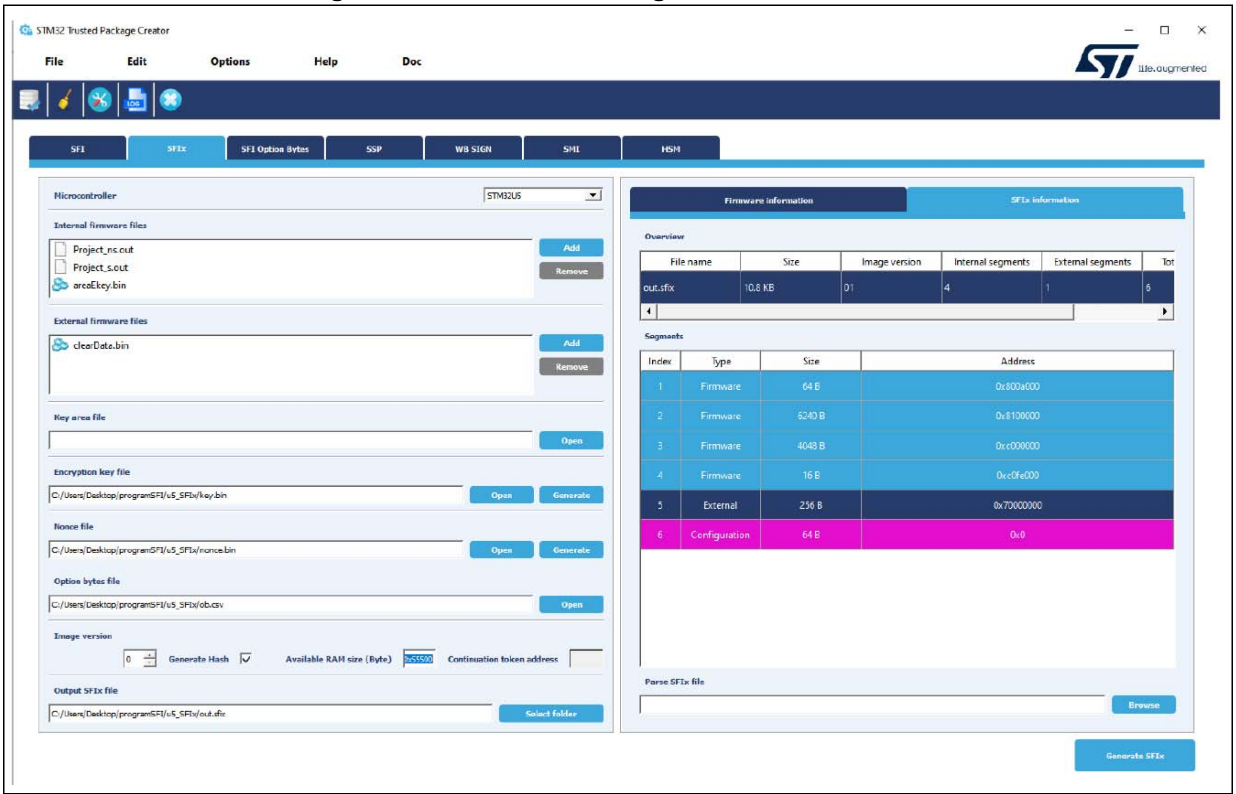
Encryption key: use the same key as STM32HSM.

Nonce file: use the same nonce as STM32HSM.

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

RAM size: 0x55500 to split the input areas avoiding memory overflow.

Figure 69. Successful SFIx generation use case 3



Find below an example for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx, and STM32U5Gxxx.

Internal firmware files:

1. Add a nonsecure binary with a start address equal to 0x08100000.
2. Add an internal binary file at 0x0C000000. It is an application to be executed after downloading SFIx to verify the full process success through a blinking LED.
3. Add an OTFDEC key binary at 0x0800A000. It is used as the key in OTFD ENCDEC.

External firmware files:

Add an external binary at 0x90000000 with these parameters:

- Region number = 3
- Region mode = 1
- Key address = 0x0800A000. It is the same as the OTFDEC key binary.

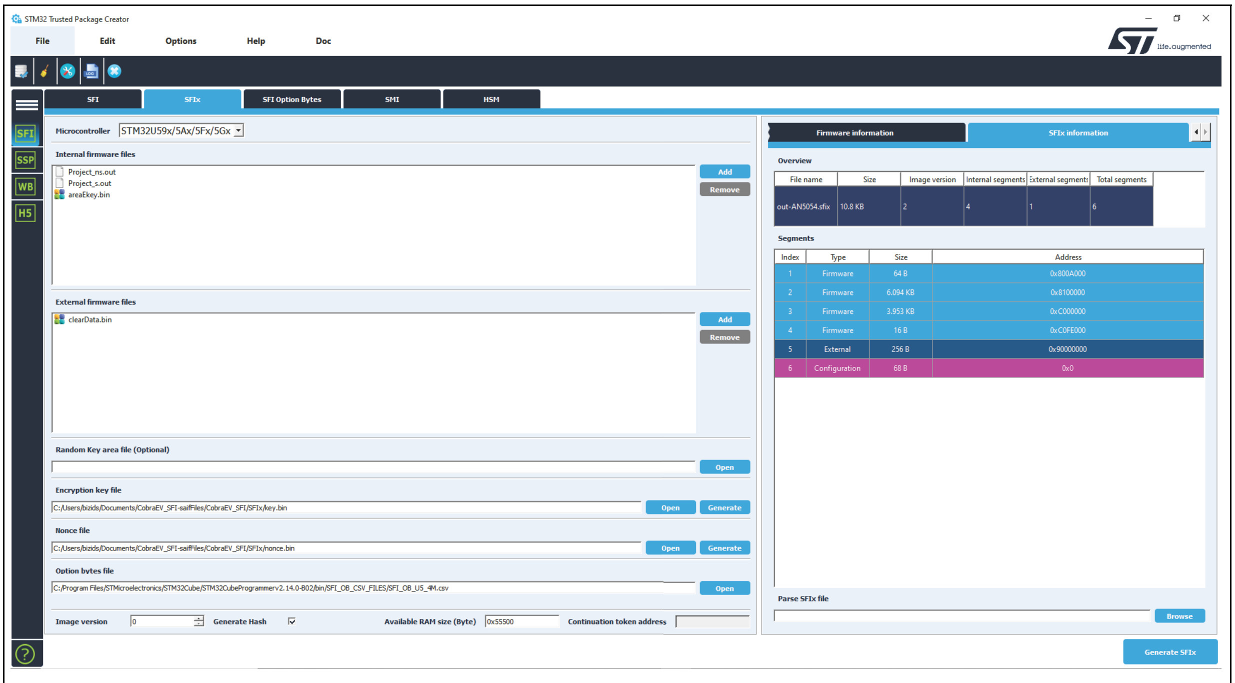
Encryption key: use the same key as STM32HSM.

Nonce file: use the same nonce as STM32HSM.

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

- **RAM size**: it is 0x55500 to split the input areas to avoid a memory overflow.

Figure 70. Successful SFIx generation use case 3 for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx, and STM32U5Gxxx



DT72095V1

Use case 4: generation of SFIx with key area for STM32U5

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

- Add a ".kcsv" file (to be used in OTFD ENC-DEC during SFIx downloading) in the key area field, instead of using an OTFDEC key binary file.
- The key address for external firmware files is the first address of the area 'K' key file, which is 0x0800A000.

Figure 71. Successful SFIx generation use case 4

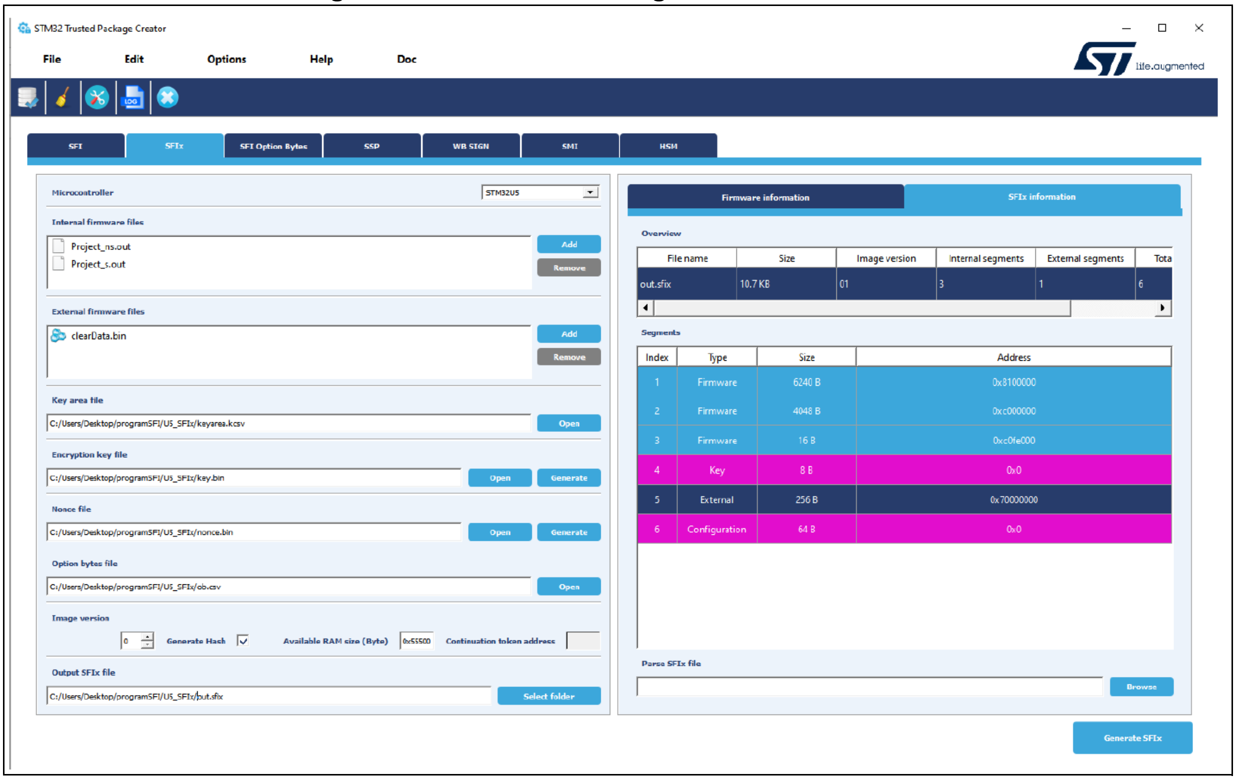
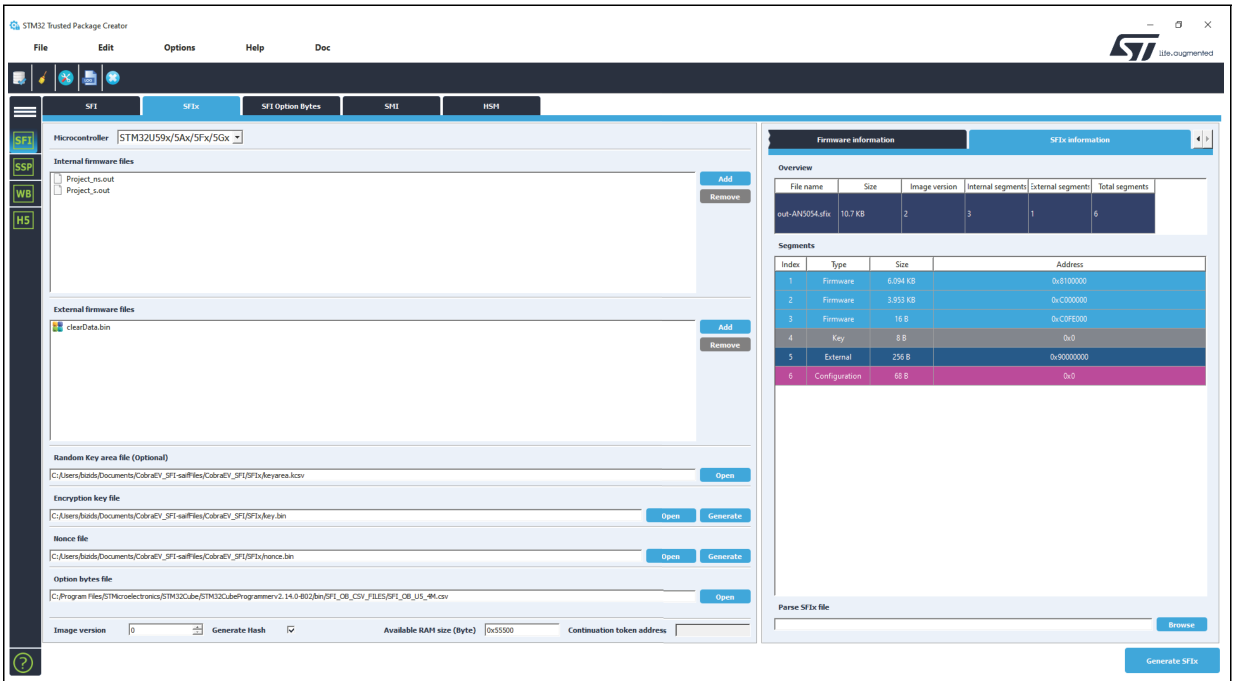


Figure 72. Successful SFIx generation use case 4 for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx, and STM32U5Gxxx



DT72096V1

12.3.3 Performing STM32HSM programming for license generation using STPC (GUI mode)

Refer to *Section 11.3.3: Performing STM32HSM programming for license generation using STPC (GUI mode)*.

12.3.4 Performing STM32HSM programming for license generation using STPC (CLI mode)

Refer to *Section 11.3.4: Performing STM32HSM programming for license generation using STPC (CLI mode)*.

12.3.5 Programming input conditions

Before performing an SFIx install, make sure that:

- A JTAG/SWD interface is used
- The chip supports security (a security bit must be present in the option bytes)
- The SFIx image is encrypted by the same key/nonce as is used in the STM32HSM provisioning.
- The option bytes are:
  - DBank=1
  - nSWBOOT0=1
  - nBOOT0=1
  - RDP=AA

12.3.6 Perform the SFIx installation using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (Hardware secure modules based on smartcard) to generate a license for the connected STM32 device during SFIx install.

Using JTAG/SWD

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

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

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

Figure 73: SFIx installation success using SWD connection (1) through *Figure 75: SFIx installation success using SWD connection (3)* show the SFIx install via SWD execution and the STM32HSM as license generation tool in the field.

Figure 73. SFIx installation success using SWD connection (1)

```
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammerv2.14.0-RO\bin>STM32_Programmer_CLI.exe -c port=swd freq=8000 mode=hotplug -sfi "C:\Users\dkhilals\Desktop\STM32L5_SFI_Exemple\out.sfix" hsm=0
 "C:\Users\dkhilals\Desktop\STM32L5_SFI_Exemple\STM32L552E_License.bin" -el "C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammerv2.14.0-B01\bin\ExternalLoader\MX25LM51245G_STM32L552E-EVAL-SFIX.
stldr"  -rsse "C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammerv2.14.0-RO\bin\RSSe\L5\enc_signed_RSSe_sfi.bin"
      -------------------------------------------------------------------
                       STM32CubeProgrammer v2.14.0
      -------------------------------------------------------------------

ST-LINK SN  : 0673FF505252717267114954
ST-LINK FW  : V2J37M26
Board       : STM32L552
Voltage     : 3.27V
SWD freq    : 4000 KHz
Connect mode: Hot Plug
Reset mode  : Software reset
Device ID   : 0x472
Revision ID : Rev Z
Device name : STM32L5xx
Flash size  : 512 KBytes
Device type : MCU
Device CPU  : Cortex-M33
BL Version  : 0x92

  Protocol Information         : static

  SFI File Information         :

     SFI file path             : C:\Users\dkhilals\Desktop\STM32L5_SFI_Exemple\out.sfix
     SFI license file path     : C:\Users\dkhilals\Desktop\STM32L5_SFI_Exemple\STM32L552E_License.bin
     SFI header information    :
         SFI protocol version        : 2
         SFI total number of areas   : 6
         SFI image version           : 0
     SFI Areas information     :

     Parsing Area 1/6    :
         Area type                   : F
         Area size                   : 4912
         Area destination address    : 0x8040000

     Parsing Area 2/6    :
         Area type                   : F
         Area size                   : 3648
         Area destination address    : 0xC000000

     Parsing Area 3/6    :
         Area type                   : F
         Area size                   : 32
         Area destination address    : 0xC03E000

     Parsing Area 4/6    :
         Area type                   : K
         Area size                   : 8
         Area destination address    : 0x0

     Parsing Area 5/6    :
         Area type                   : E
         Area size                   : 272
         Area destination address    : 0x90000000

     Parsing Area 6/6    :
         Area type                   : C
         Area size                   : 48
```

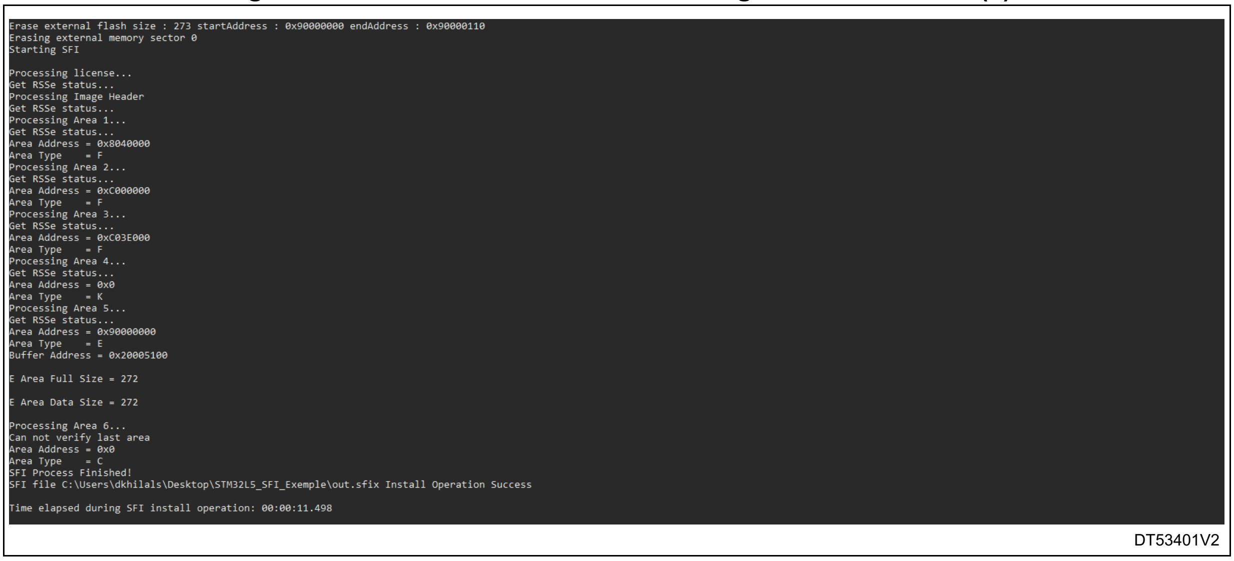
DT53099V2

Figure 74. SFIx installation success using SWD connection (2)

```
Reconnecting...
Reconnected !
Time elapsed during option Bytes configuration: 00:00:02.187
Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.030
Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified.
Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.027
Warning: Option Byte: SECBOOTADD0, value: 0x1FF000, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.015
Installing RSSe


Memory Programming ...
Opening and parsing file: enc_signed_RSSe_sfi.bin
  File          : enc_signed_RSSe_sfi.bin
  Size          : 35.56 KB
  Address       : 0x20005100


Erasing memory corresponding to segment 0:
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:00.273
Warning: Option Byte: TZEN, value: 0x1, was not modified.
Warning: Option Byte: nBoot0, value: 0x0, was not modified.
Warning: Option Byte: nSWBoot0, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.024


Reconnecting...
Reconnected !
Time elapsed during option Bytes configuration: 00:00:04.306
Get RSSe status...
Warning: Option Byte: TZEN, value: 0x1, was not modified.
Warning: Option Byte: nBoot0, value: 0x0, was not modified.
Warning: Option Byte: nSWBoot0, value: 0x0, was not modified.
Warning: Option Bytes are unchanged, Data won't be downloaded
Time elapsed during option Bytes configuration: 00:00:00.021


RSS version = 5.1.0


RSSe version = 5.0.0
```

DT53400V2

Figure 75. SFIx installation success using SWD connection (3)



13 Example of SFIx programming scenario for STM32H5

13.1 Scenario overview

There are three steps during this scenario:

1. Generate an SFIx image using the STPC
2. STM32HSM card provisioning via STPC
3. Use STM32CubePrg to perform the SFIx process.

13.2 Hardware and software environment

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

- An STM32H5-based board with an external flash memory and system flash security package (SFSP) v2.4.0 or greater
- SFI programming via SWD
- A PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- An STM32 Trusted Package Creator v2.14.0 (or greater) package is available from *www.st.com*
- An STM32CubeProgrammer v2.14.0 (or greater) package is available from *www.st.com*
- An STM32HSM-V2 smartcard

Note: Refer to [4] or [5] for the supported operating systems and architectures.

13.3 Step-by-step execution

13.3.1 Build an OEM application

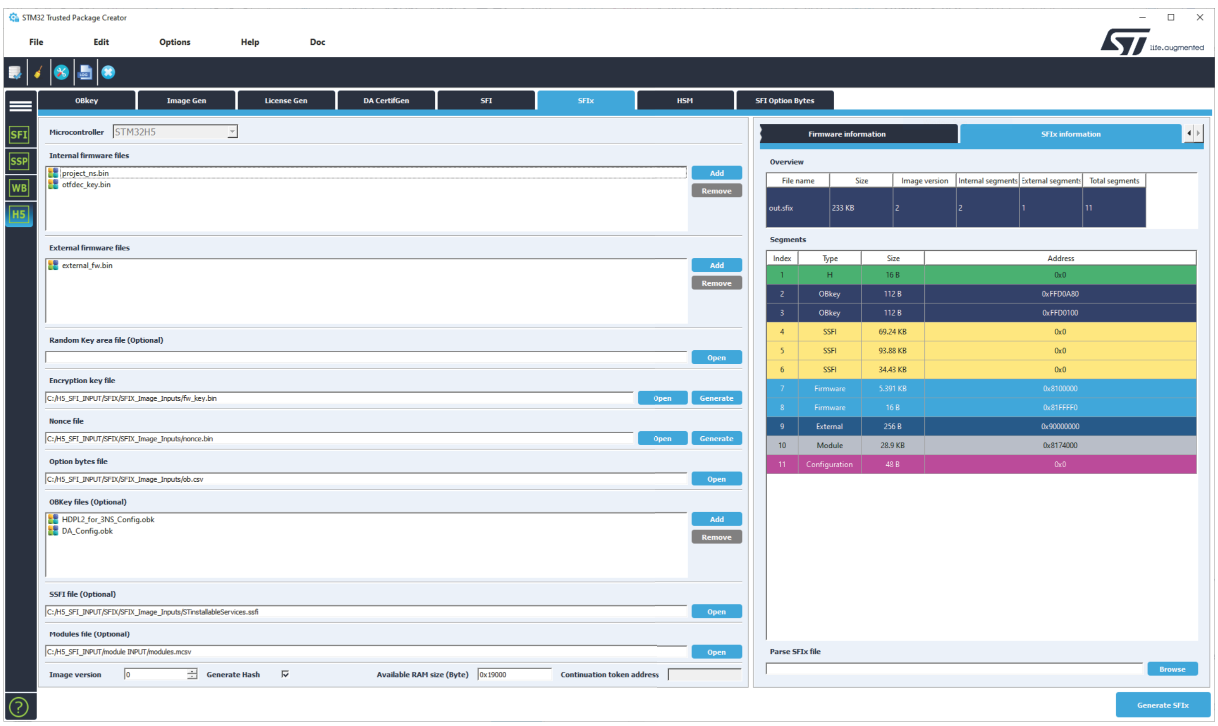
OEM application developers can use any IDE to build their own firmware.

13.3.2 Perform the SFIx generation (GUI mode)

The first step to install the secure firmware on STM32H5 devices is the encryption of the user OEM firmware using the STM32 Trusted Package Creator tool. This step is done by including the following files:

- An OEM firmware at 0x08100000
- A .csv file containing option bytes configuration
- A 128-bit AES encryption key
- A 96-bit nonce
- A binary file for an external firmware file
- OBKey files for device configuration
- An SSFI file to integrate the STMicroelectronics SFI image
- An OTFDEC key binary at 0x081FFFF0 (to be used as the key in OTFD ENC/DEC)
- External firmware files. Add an external binary at 0x90000000 with the following parameters:
  - Region number = 0
  - Region mode = 0x2
  - Key address = 0x081FFFF0 (same as the OTFDEC key binary)
- An MCSV file to insert the modules list:
  ./module.bin, ./LicenseV0.bin, 0x8172000

Figure 76. SFIx image generation for STM32H5



DT72097V1

13.3.3 Programming input conditions

Before performing an SFIx install on STM32H5 devices, make sure that:

- There is an accessible external memory loader file such as MX25LM51245G_STM32H573I-DK-RevB-SFIx.stldr
- The chip supports security and boots on system memory
- The product state is open: 0xED
- An RSSe binary is available
- The STM32HSM-V2 is provisioned for the STM32H5 product

Note: The RSSe binary file is in the STM32CubeProgrammer bin/RSSe/H5 folder.

Note: To embed an SSFI image into the SFI image, it is recommended to follow a specific secure sequence and choose an adequate start address of the nonsecure application that depends on the SSFI configuration. See the details in STM32CubeH5 MCU Package available from www.st.com.

13.3.4 Perform the SFIx installation using STM32CubeProgrammer CLI

In this section, the STM32CubeProgrammer tool is used in CLI mode to program the SFIx image *"out.sfix"* already created in the previous section.

STM32CubeProgrammer communicates with the device through the SWD interface after it is confirmed that all the input conditions are respected.

Open a cmd terminal, go to /bin in the install path, and then launch the following command:

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 ap=1
-sfi "out.sfix" hsm=1 slot=1 -rsse
"\RSSe\H5\enc_signed_RSSe_SFI_STM32H5_v2.0.0.0.bin"
-el "\ExternalLoader\MX25LM51245G_STM32H573I-DK-RevB-
SFIx.stldr"
-mcsv ".\modules.mcsv"
```

Figure 77. SFIx installation success for STM32H5

```
RSS version = 2.2.0


RSSe version = 2.0.0

Erase external flash size : 273 startAddress : 0x90000000 endAddress : 0x90000110
Erasing external memory sector 0
Starting SFI

Processing license...
Get RSSe status...
Processing Image Header
Get RSSe status...
Processing Area 1...
Get RSSe status...
Area Address = 0x0
Area Type    = H
Processing Area 2...
Get RSSe status...
Area Address = 0xFFD0A80
Area Type    = O
Processing Area 3...
Get RSSe status...
Area Address = 0xFFD0100
Area Type    = O
Processing Area 4...
Get RSSe status...
Area Address = 0x0
Area Type    = S
Processing Area 5...
Get RSSe status...
Area Address = 0x0
Area Type    = S
Processing Area 6...
Get RSSe status...
Area Address = 0x0
Area Type    = S
Processing Area 7...
Get RSSe status...
Area Address = 0x8100000
Area Type    = F
Processing Area 8...
Get RSSe status...
Area Address = 0x81FFFF0
Area Type    = F
Processing Area 9...
Get RSSe status...
Area Address = 0x90000000
Area Type    = E
Buffer Address = 0x20054100

E Area Full Size = 272

E Area Data Size = 272


MCSV file parsing...
Total modules number: 1
   +  Module number : [0]
          Name      : ./module_0_with_licence_globale.smu
          Type      : Global license
          Size      : 128.00 KB
          Address   : 0x08172000

Prepare module payload with license data...
Processing Area 10...
Get RSSe status...
Area Address = 0x8174000
Area Type    = m
Processing Area 11...
Can not verify last area
Area Address = 0x0
Area Type    = C
SFI Process Finished!
SFI file out.sfix Install Operation Success

Time elapsed during SFI install operation: 00:00:18.452
```

DT72098V1

14 Example of SSP programming scenario for STM32MP1

14.1 Scenario overview

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

Three main steps are done using SSP tools:

- Encrypted secret file generation with STM32 Trusted Package Creator
- STM32HSM provisioning with STM32 Trusted Package Creator
- SSP procedure with STM32CubeProgrammer.

14.2 Hardware and software environment

The following prerequisites are needed for successful SSP programming:

- an STM32MP157F-DK2 board
- a Micro-B USB for DFU connection
- a PC running on either Windows®, Linux® Ubuntu® or Fedora®, or macOS®
- STM32 Trusted Package Creator v1.2.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.5.0 (or greater) package available from *www.st.com*
- an STM32HSM-V2 card

Note: Refer to [4] or [5] for the supported operating systems and architectures.

14.3 Step-by-step execution

14.3.1 Building a secret file

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

On an STM32MP1 microprocessor:

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

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

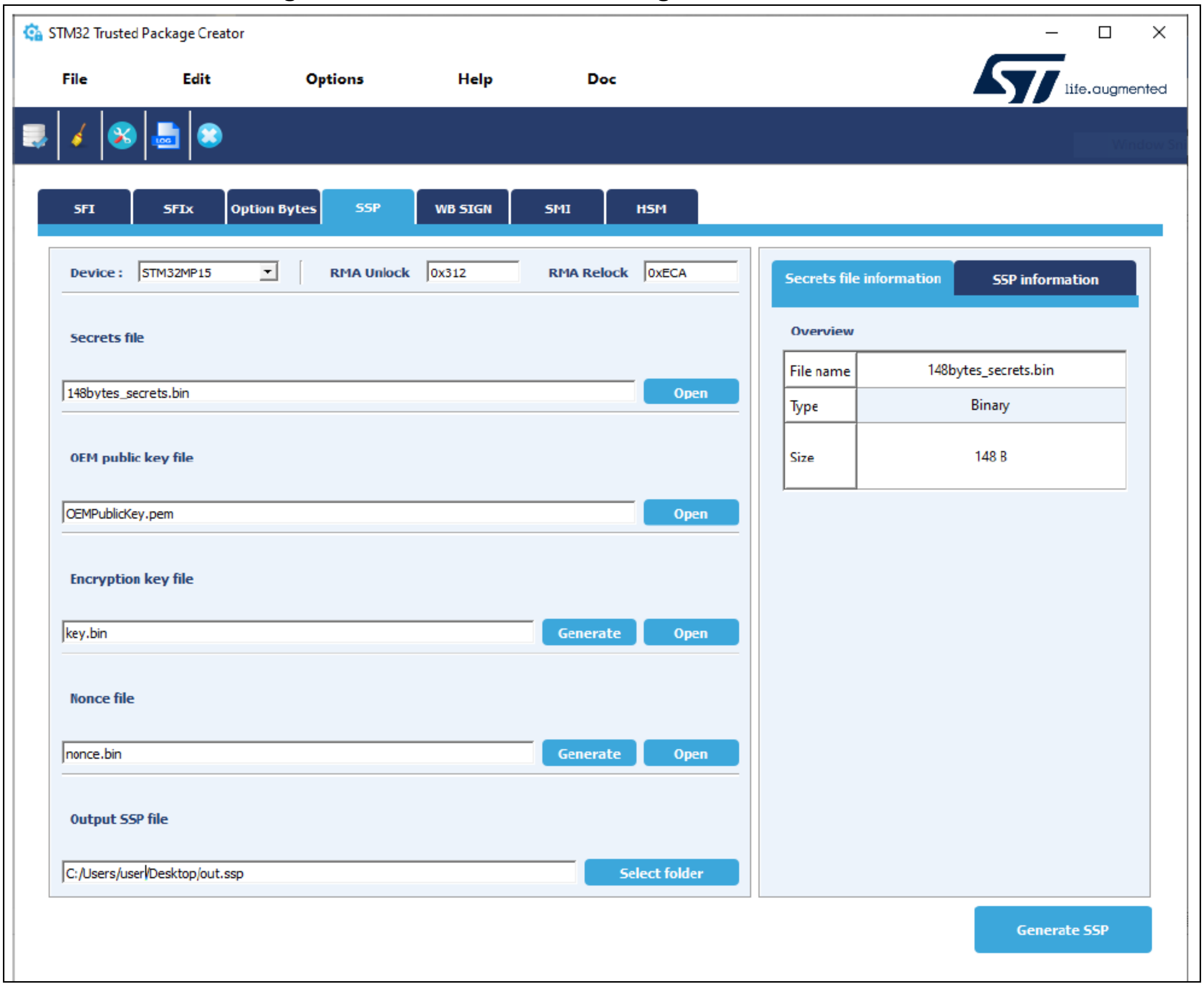
14.3.2 Performing the SSP generation (GUI mode)

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

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

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

Figure 78. STM32 Trusted Package Creator SSP GUI tab



14.3.3 Performing STM32HSM programming for license generation using STPC (GUI mode)

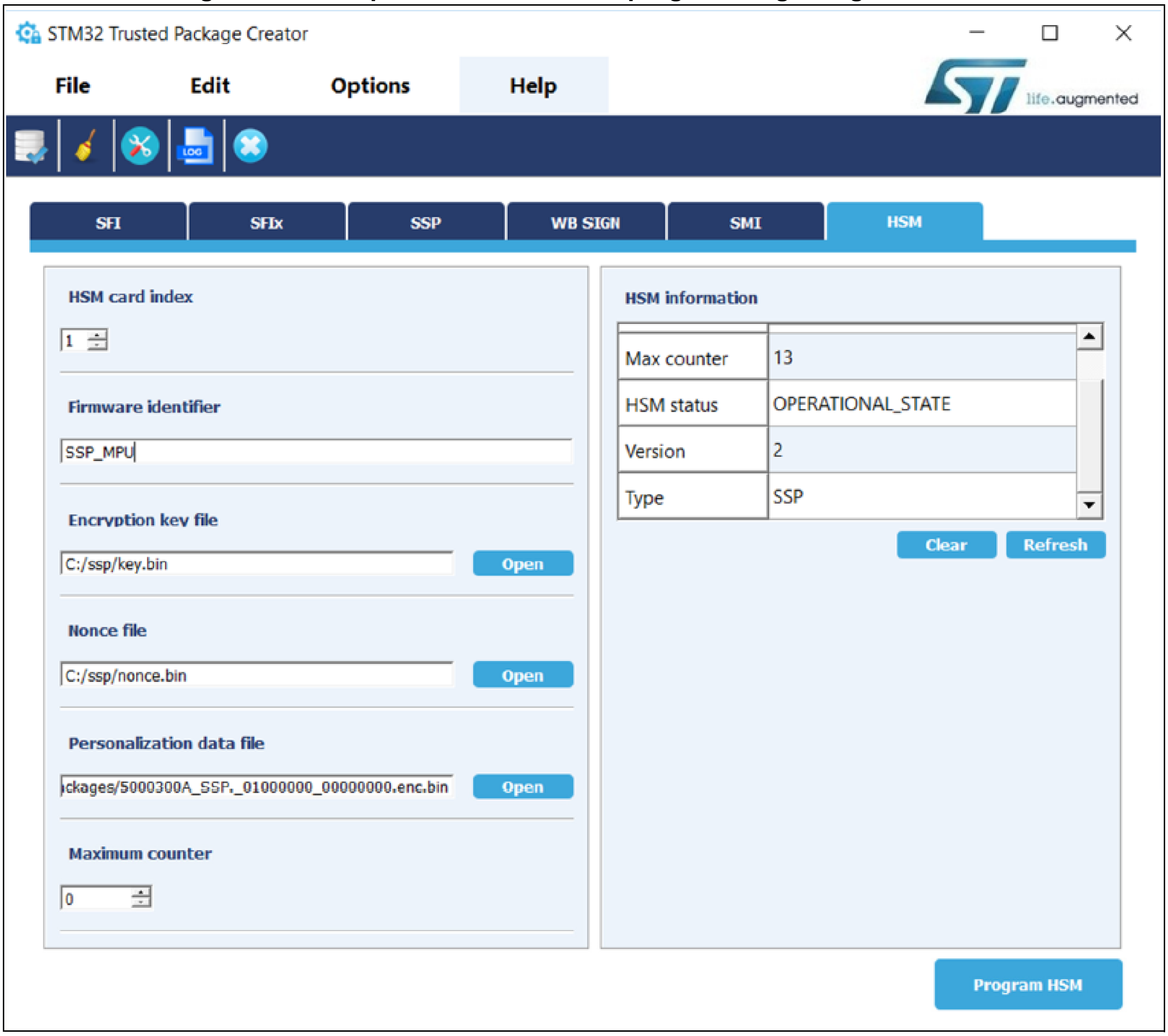
The OEM must provide a license generation tool to the programming house, to be used for license generation during the SSP install process. In this example, STM32HSMs are used as license generation tools in the field.

See *Section 3.1.2: License mechanism* for STM32HSM use and programming details.

This example uses STM32HSM-V2. The STM32HSM version can be identified before performing the programming operation by clicking the **Refresh** button to make the version number appear in the version field.

Note: STM32HSM-V2 must be used for STM32 MPU devices.

Figure 79. Example of STM32HSM-V2 programming using STPC GUI



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

14.3.4 SSP programming conditions

Before performing an SSP flow make sure that:

- only DFU or UART interfaces are used
- the chip supports security
- the SSP image is encrypted by the same key/nonce as used in the STM32HSM provisioning step.
- There is an adequate Trusted Firmware-A file, which is previously signed and ready for SSP use via USB or UART interface.

14.3.5 Perform the SSP installation using STM32CubeProgrammer

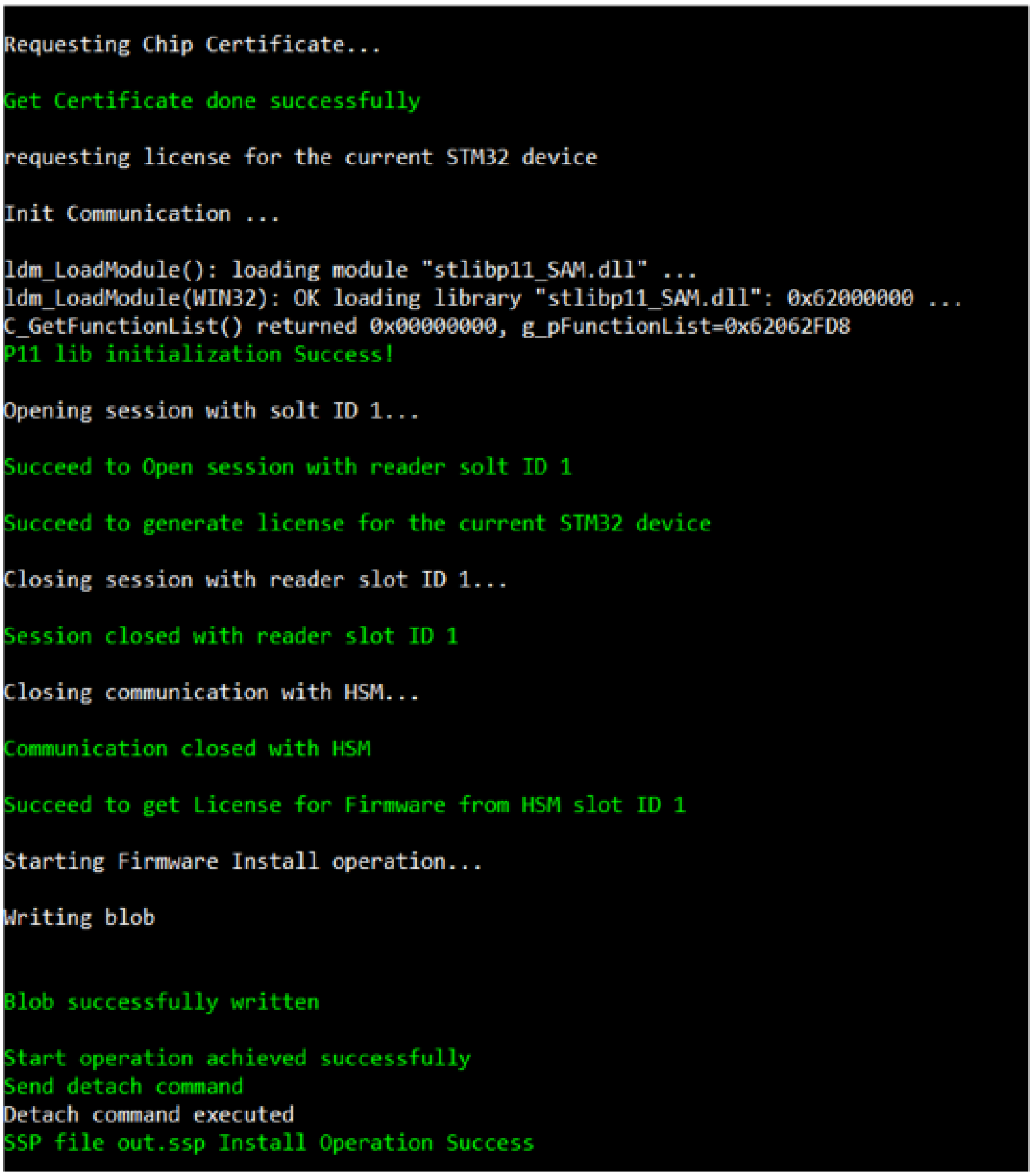
In this step, the STM32CubeProgrammer tool is used in CLI mode (the only mode available so far for secure programming) to program the SSP image already created with STM32 Trusted Package Creator. STM32CubeProgrammer supports communication with STMicroelectronics STM32HSMs (hardware secure modules based on a smartcard) to generate a license for the connected STM32 MPU device during SSP install.

Example using USB DFU bootloader interface:

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

All SSP traces are shown on the output console (*Figure 80*).

Figure 80. STM32MP1 SSP installation success



15 Example of SSP-SFI programming scenario for STM32MP2

15.1 Scenario overview

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

Five main steps are done using SSP tools:

- Secrets generation with STM32 Trusted Package Creator
- Backup memory generation with STM32 Trusted Package Creator (optional)
- SSP-SFI file generation with STM32 Trusted Package Creator
- STM32HSM provisioning with STM32 Trusted Package Creator
- SSP-SFI procedure with STM32CubeProgrammer.

15.2 Hardware and software environment

The following prerequisites are needed for a successful SSP-SFI programming:

- An STM32MP2 board
- A USB-C cable for DFU connection
- A PC running on either Windows®, Linux® or macOS®
- The STM32 Trusted Package Creator v2.17.0 (or greater) package available from *www.st.com*
- STM32CubeProgrammer v2.17.0 (or greater) package available from *www.st.com*
- An STM32HSM-V2 card

Note: Refer to [4] or [5] for the supported operating systems and architectures.

15.3 Step-by-step execution

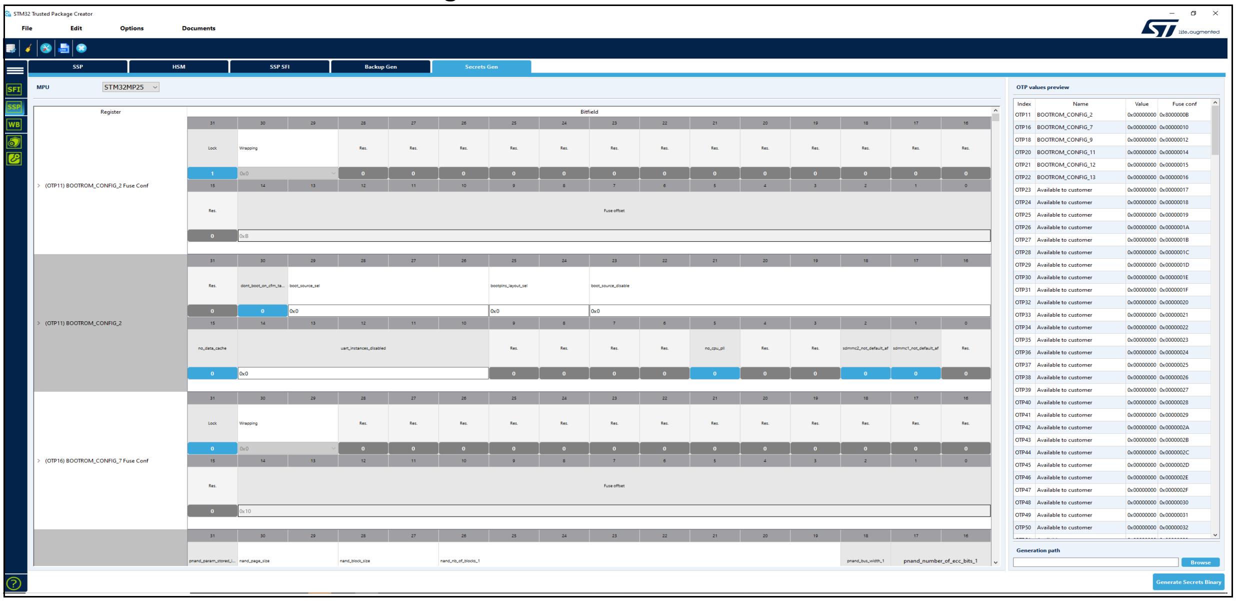
15.3.1 Building a secret file

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

The STM32Trusted Package Creator offers a graphical interface to edit and customize the secrets binary.

From the SSP panel, select the "*Secrets Gen*" tab and start the editing.

Figure 81. Secrets Gen Window



15.3.2 Building a backup memory file

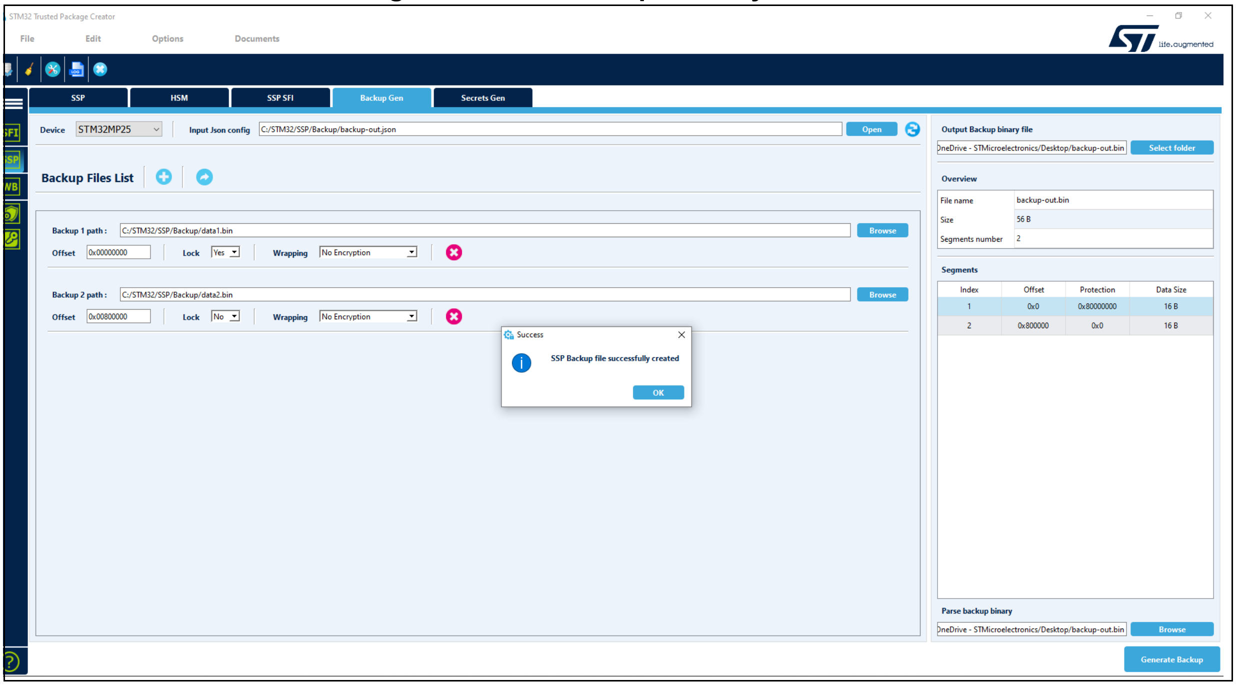
It is optional to integrate a backup file into an SSP-SFI image by specifying the backup input file.

The STM32Trusted Package Creator offers a graphical interface to edit and customize the secrets of the backup memory file.

From the SSP panel, select the "*Backup Gen"* tab and start the editing.

If all necessary elements are present,pressing the "*Generate Backup*" button initiates the preparation of the image. The resulting image is saved into a binary file, which is specified in the "*Output Backup binary file*" field.

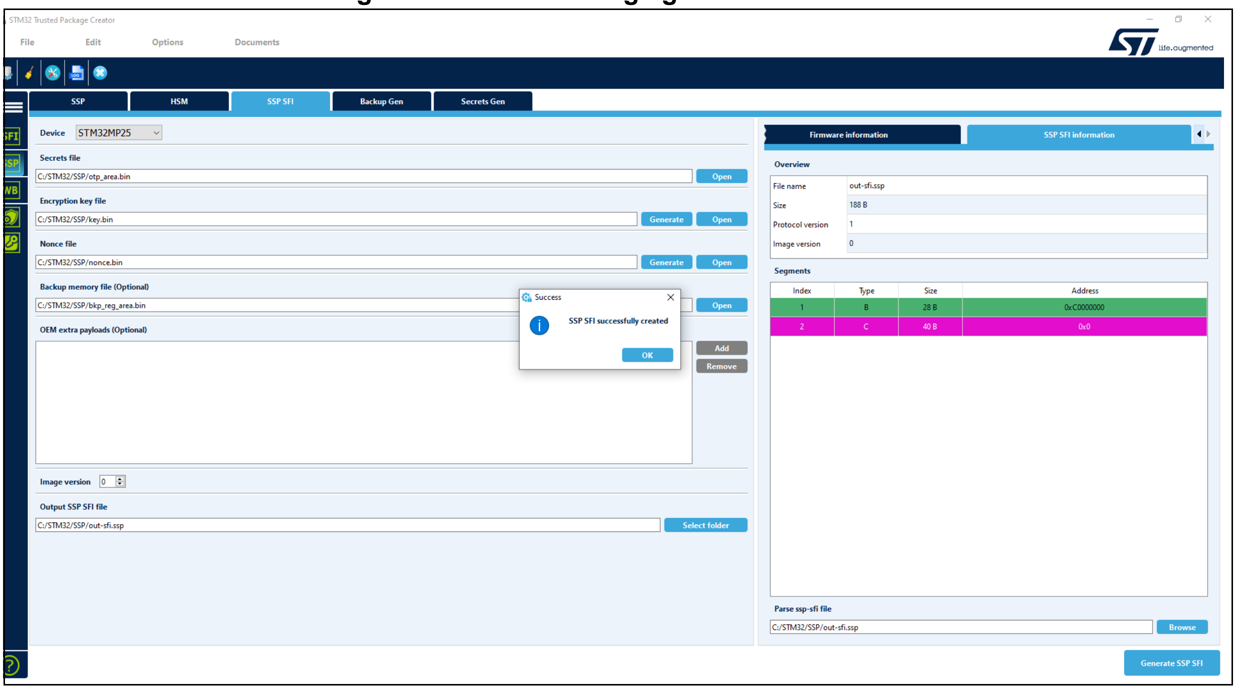
Figure 82. SSP Backup memory window



15.3.3 Performing the SSP-SFI generation (GUI mode)

The STM32Trusted Package Creator tool GUI presents an SSP-SFI tab located in the SSP panel to generate an SSP image in SFI format. The user must fill in the input fields with valid values.

Figure 83. SSP-SFI image generation window



15.3.4 Performing STM32HSM programming (GUI mode)

Refer to *Section 14.3.3: Performing STM32HSM programming for license generation using STPC (GUI mode)*.

15.3.5 SSP-SFI programming conditions

Before performing an SSP flow make sure that:

- Only DFU or UART interfaces are used.
- The chip supports security to deploy the SSP flow.
- The SSP image is encrypted by the same key/nonce that is used in the STM32HSM provisioning step.
- A trusted RSSe SSP binary provided by STMicroelectronics is used.

15.3.6 Perform the SSP installation using STM32CubeProgrammer

In this step, the STM32CubeProgrammer tool is used in CLI mode (in a similar way the GUI mode with the Security window can be used) to program the SSP-SFI image already created with STM32 Trusted Package Creator.

The STM32CubeProgrammer supports the communication with STMicroelectronics STM32HSMs (hardware secure modules based on a smartcard) to generate a license for the connected STM32MP2 device during the SSP installation.

Example using USB DFU bootloader interface:

```
STM32_Programmer_CLI.exe -c port=usb1 -ssp "image.ssp"
"EncBootExt_STM32_RSSE_SSP.bin" hsm=1 slot=1
```

The file **EncBootExt_STM32_RSSE_SSP.bin** is located in the STM32CubeProgrammer install path under the /bin/RSSe/MP25 folder.

All the SSP traces are shown on the output console.

Figure 84. SSSP-SFI installation

```
        Area size                    : 65536
        Area destination address     : 0xC0000000

    Parsing Area 2/2    :
        Area type                    : C
        Area size                    : 88
        Area destination address     : 0x0

Current phase ID  : 0x01
Reading the chip certificate...


Requesting Chip Certificate...
Product ID : 5050200E
Get certificate is done successfully
requesting license for the current STM32 device

Init Communication ...

ldm_LoadModule(): loading module "C:/Program Files/STMicroelectronics/STM32Cube/STM32CubeProgrammer2.17.0[redacted]/bin\stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "C:/Program Files/STMicroelectronics/STM32Cube/STM32CubeProgrammer2.17.0[redacted]/bin\stlibp11_SAM.dll": 0x28E30000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x28E9E490
P11 lib initialization Success!

Opening session with slot ID 1...

Succeed to Open session with reader slot ID 1

Succeed to generate license for the current STM32 device

Closing session with reader slot ID 1...

Session closed with reader slot ID 1

Closing communication with HSM...

Communication closed with HSM

Succeed to get license from HSM slot ID 1

Writing blob


Blob is successfully written


Memory Programming ...
Opening and parsing file: EncBootExt_STM32_RSSE_SSP_CubeProg_Validation_Tool_v0.1.0.bin
  File          : EncBootExt_STM32_RSSE_SSP_CubeProg_Validation_Tool_v0.1.0.bin
  Size          : 73.86 KB
  Partition ID  : 0xF3

Download in Progress:
                                                   100%

File download complete
Time elapsed during download operation: 00:00:00.861
Start operation achieved successfully
Send detach command
Detach command executed
[redacted]

Time elapsed during SSP install operation: 00:00:03.060
```

16 Reference documents

Table 3. Document references

Reference	Document title
[1]	Application note *Introduction to secure firmware install (SFI) for STM32 MCUs* (AN4992), STMicroelectronics.
[2]	User manual *Hardware secure module (HSM) for STM32CubeProgrammer secure firmware install (SFI)* (UM2428), STMicroelectronics.
[3]	Application note *Overview of the secure secret provisioning (SSP) on STM32MP1 series* (AN5510), STMicroelectronics.
[4]	Release note *STM32CubeProgrammer release vx.y.z* (RN0109), STMicroelectronics.
[5]	User manual *STM32 Trusted Package Creator tool software description* (UM2238), STMicroelectronics.

17 Revision history

Table 4. Document revision history

Date	Revision	Changes
03-Aug-2018	1	Initial release.
18-Apr-2019	2	Updated publication scope from ‘ST restricted’ to ‘Public’.
16-Oct-2019	3	Updated: – *Section 4.1.2: License mechanism* – *Section 5.3.4: Performing HSM programming for license generation using STPC (GUI mode)* – *Figure 44: HSM programming GUI in the STPC tool* (title caption) – *Figure 54: Example of HSM programming using STPC GUI*
03-Feb-2020	4	Replaced occurrences of STM32L451CE with STM32L462CE in *Section 4.2.1: Secure firmware installation using a bootloader interface flow*. Updated document to cover secure programming with SFIx.
26-Feb-2020	5	Updated: – *Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow* – *Section 5.3.4: Performing HSM programming for license generation using STPC (GUI mode)* – *Section 5.3.5: Performing HSM programming for license generation using STPC (CLI mode)* – *Figure 72: SFIx installation success using SWD connection (1)* – *Figure 75: SFIx installation success using SWD connection (4)*
27-Jul-2020	6	Updated: – *Introduction* – *Section 3.1: System requirements* Added: – *Section 3.5: SSP generation process* – *Section 3.6.3: Steps for SSP generation (CLI)* – *Section 3.7.4: SSP generation using STPC in GUI mode* – *Section 4.2.5: STM32CubeProgrammer for SSP via a bootloader interface* – *Section 12: Example of SSP programming scenario for STM32MP1*

Table 4. Document revision history (continued)

Date	Revision	Changes
19-Nov-2020	7	Updated: – *Introduction* on cover page – License mechanism general scheme – HSM programming by OEM for license distribution – *Section 5.3.5: Performing HSM programming for license generation using STPC (CLI mode)* Added: – *Section 4.4: Secure programming using bootloader interface (UART/I2C/SPI/USB)* – *Section 6: Example of SFI programming scenario for STM32WL*
29-Jun-2021	8	Updated: – In the whole document, replaced STM32H7A/B by STM32H7A3/7B3 and STM32H7B0, STM32H72/3 by STM32H723/333 and STM32H725/335, STM32H7B board by STM32H7B3I-EVAL – Replaced BL by bootloader. – *Section 3.2: SFI generation process: removed references to RSS* – *Section 4.1.2: License mechanism: removed Figure HSM programming toolchain* – *Section 4.2: Secure programming using a bootloader interface* – *Section 4.2.2: Secure module installation using a bootloader interface flow* – *Section 4.2.3: STM32CubeProgrammer for SFI using a bootloader interface* – *Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow and Section 4.3.2: SMI programming through JTAG/SWD flow* – *Section 4.4: Secure programming using bootloader interface (UART/I2C/SPI/USB)* – *Example of SFI programming scenario* – *Section 5.2: Hardware and software environment and Example of SFI programming scenario for STM32WL* – *Section 6.2: Hardware and software environment: removed bootloader and RSS versions* – *Section 5.3.5: Performing HSM programming for license generation using STPC (CLI mode)*: removed STM32L4 from the list of devices that support SFI via debug interface. Added: – Support for STM32U5 Series. – *Section 7: Example of SFI programming scenario for STM32U5*

Table 4. Document revision history (continued)

Date	Revision	Changes
02-Aug-2021	9	Added note about CSV file in *Section 3.6.1: Steps for SFI generation (CLI)* and *Figure 27: Option bytes file example*. Corrected binary file names in *Section 4.4: Secure programming using bootloader interface (UART/I^{2}C/SPI/USB)*. Added *Section 3.6.1: Steps for SFI generation (CLI)*. Added note about option byte file example in: – *Section 3.7.1: SFI generation using STPC in GUI mode* – *Section 5.3.3: Perform the SFI generation (GUI mode)* – *Section 6.3.2: Perform the SFI generation (GUI mode)* – *Section 7.3.2: Perform the SFI generation (GUI mode)* – *Section 9.3.2: Perform the SFIx generation (GUI mode)* – *Section 10.3.2: Perform the SFIx generation (GUI mode)* – *Section 11.3: Step-by-step execution* Updated: – Corrected board name in *Section 4.2: Secure programming using a bootloader interface* – Corrected board name in *Section 7.2: Hardware and software environment*
04-Mar-2022	10	Updated *Section 3.3: SFIx generation process*. Added: – *Section 5.3.2: Performing the option bytes file generation (GUI mode)* – *Section 5.3.8: SFI with Integrity check (for STM32H73)*
29-Jun-2022	11	Updated: – *Section 3.3: SFIx generation process* – *Section 4.2.3: STM32CubeProgrammer for SFI using a bootloader interface* – *Section 10.1: Scenario overview* – *Section 10.2: Hardware and software environment* – *Section 10.3.2: Perform the SFIx generation (GUI mode):* STM32CubeProgrammer version, use cases 1 and 2 scope STM32L5, and added subsections for use cases 3 and 4 for STM32U5, listed below. – *Figure 67: STPC GUI during SMI generation* – *Figure 88: STM32 Trusted Package Creator SSP GUI tab* – *Section 12.3.4: SSP programming conditions* Added: – Use case 3: generation of SFIx without key area for STM32U5 – Use case 4: generation of SFIx with key area for STM32U5
25-Nov-2022	12	Updated *Section 3.2: SFI generation process*. Removed “multi install” from document.

Table 4. Document revision history (continued)

Date	Revision	Changes
24-Feb-2023	13	Updated: – *Section 3.6: STM32 Trusted Package Creator tool in the command line interface* – *Section 3.6.1: Steps for SFI generation (CLI)*
04-Aug-2023	14	Global document update, and compatibility with the STM32H5 series and extended STM32U5 series. Updated: – *Figure 28: SFI generation example using an ELF file* and the related command line example – *Figure 60: STPC GUI showing the STPC GUI during the SFI generation* – *Figure 63: SFI installation via SWD execution command-line output* – *Figure 86: Successful SFIx generation use case 1* – *Figure 87: Successful SFIx generation use case 2* – *Figure 92: SFIx installation success using SWD connection (1)* – *Figure 93: SFIx installation success using SWD connection (2)* – *Figure 94: SFIx installation success using SWD connection (3)* Removed: – *Figure 83. SFIx installation success using SWD connection (4)* – *Figure 84. SFIx installation success using SWD connection (5)* Added: – *Chapter 9: Example of SFI programming scenario for STM32H5* – *Chapter 14: Example of SFIx programming scenario for STM32H5* – *Figure 89: Successful SFIx generation use case 3 for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx, and STM32U5Gxxx* – *Figure 91: Successful SFIx generation use case 4 for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx, and STM32U5Gxxx* Minor text edits across the document.
22-Mar-2024	15	Added: – *Example of SFI programming scenario for STM32WBA5* – *Example of SFI programming scenario for STM32H7RS* Updated: – Document title – *License mechanism* – *Perform the SFI generation (GUI mode)* – *Performing the SFI install using STM32CubeProgrammer*

Table 4. Document revision history (continued)

Date	Revision	Changes
24-Jun-2024	16	Added: – *Section 8: Example of SFIA programming scenario for STM32WBA5* – *Section 15: Example of SSP-SFI programming scenario for STM32MP2*
20-Nov-2024	17	Updated: – *Section 1: General information* – *Section 1.2: Acronyms and abbreviations* – *Section 2.1: System requirements* – *Section 2.5: STM32 Trusted Package Creator tool in the command-line interface* – *Section 3.2: Secure programming using a bootloader interface* – *Section 3.1.2: License mechanism* Removed different references to SMI.
10-Mar-2025	18	Updated: – *Chapter 7: Example of SFI programming scenario for STM32WBA5x and STM32WBA6x* extended to STM32WBA6x – The *Firmware key* item in *Section 3.1.2: License mechanism* – References of *Figure 50: SFI installation via UART execution using CLI (2)* and *Figure 51: SFI installation via UART execution using CLI (3)* – *Chapter 10: Example of SFI programming scenario for STM32H7Rx/7Sx* with STM32H7Rx/7Sx to refer to the microcontrollers in the STM32H7R3/7S3 and STM32H7R7/7S7 lines – The entire document with STM32HSM for the hardware security module – The entire document with ST-LINK as the generic term describing the probe interface – The AN4992 title in *Chapter 16: Reference documents*.