UM2238
User manual

STM32 Trusted Package Creator tool software description

Introduction

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

- Secure internal firmware install (SFI). SFI is a secure mechanism that allows secure installation of OEM internal firmware in untrusted production environments by encrypting the whole internal firmware with an AES-GCM key.
- Secure external firmware install (SFIx). SFIx is a secure mechanism that allows secure installation of external OEM firmware in untrusted production environments by encrypting the whole external firmware with an AES-GCM key.
- Secure secret provisioning (SSP). SSP allows OEM secrets to be programmed into the STM32 one time programming (OTP) area in a secure way (with confidentiality, authentication and integrity checks).

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

Contents

List of tables

List of figures

1 System requirements

Supported operating systems and architectures:

- Windows® 10 and 11, 32 bits (x86) or 64 bits (x64)[a]
- Linux®: Ubuntu® LTS 20.04 and LTS 22.04, and Fedora® 36[b] [c] [d]
- macOS® 12 (Monterey), macOS® 13 (Ventura), macOS® 14 (Sonoma)[e]

STM32CubeProgrammer and STM32 Trusted Package Creatorsupport STM32 32-bit devices based on Arm®[f] Cortex®-M processors.

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.

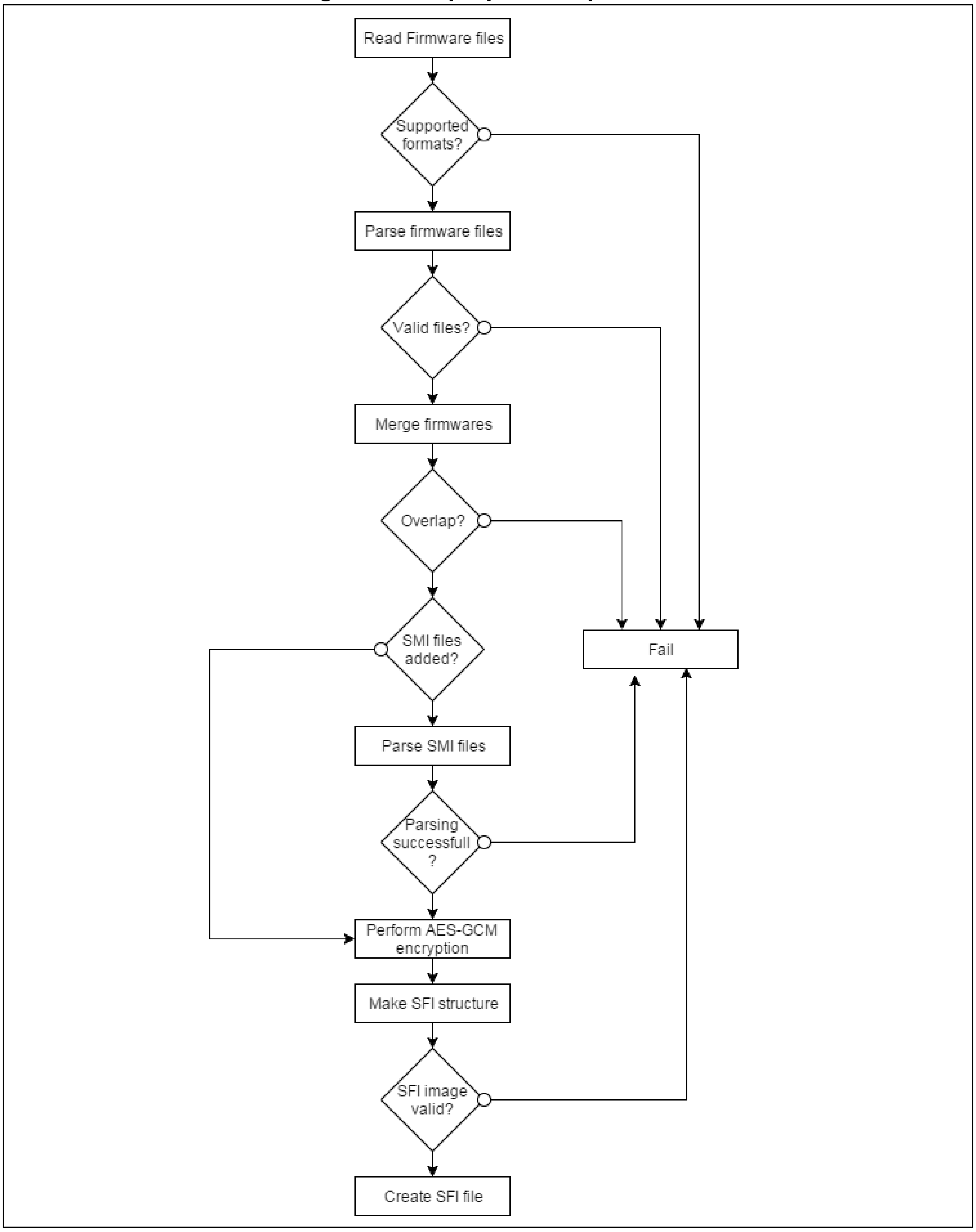
f. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.

2 Preparation processes

2.1 SFI preparation process

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

Figure 1. SFI preparation process



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

IV = nonce + area index

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

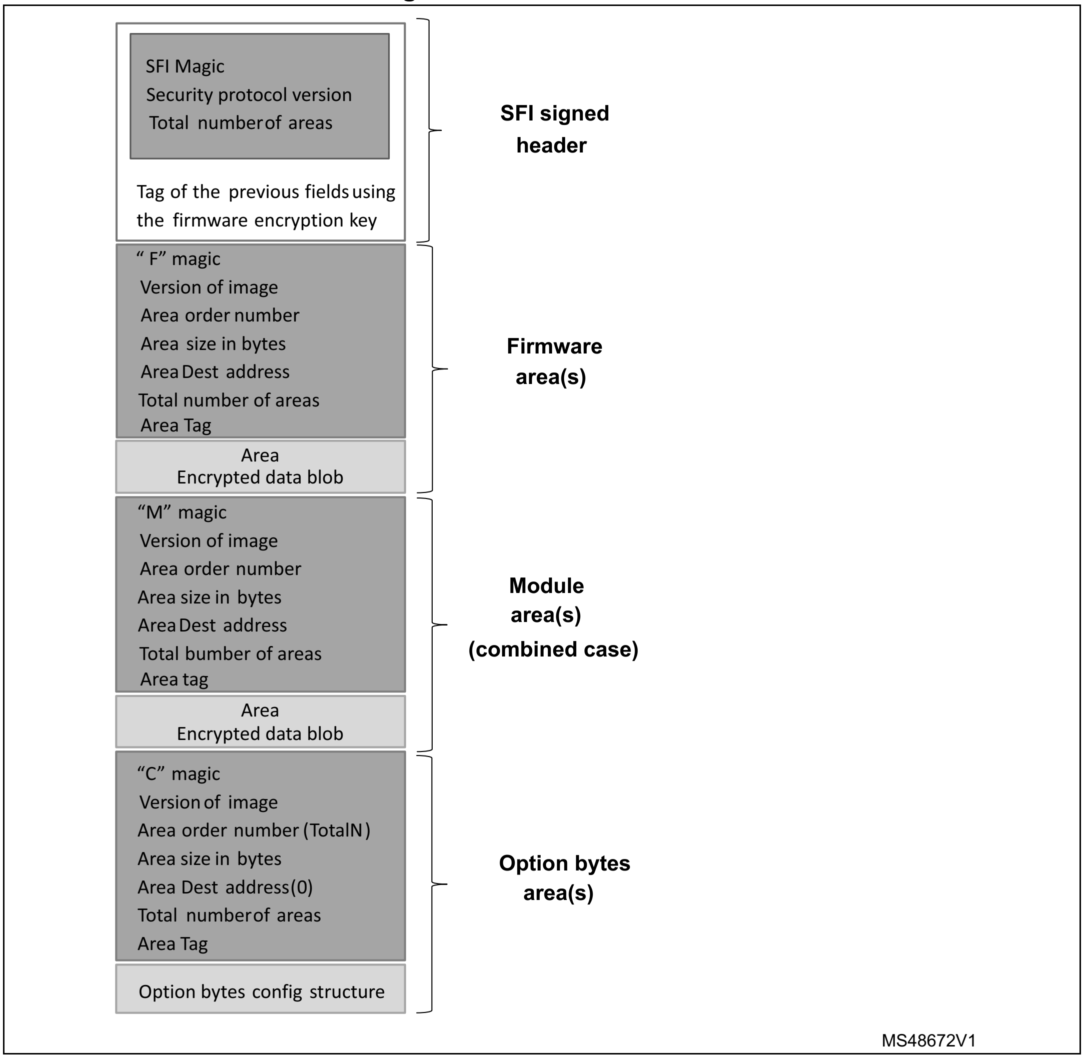
It then, it passes the area descriptor (starting from the magic to the total number of areas) as additional authenticated data (AAD).

- Each segment in the input firmware constitutes a firmware (F) area in the SFI file.
- The option bytes configuration constitutes the configuration (C) area.

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

The structure of an SFI file is shown in *Figure 2*.

Figure 2. SFI file structure



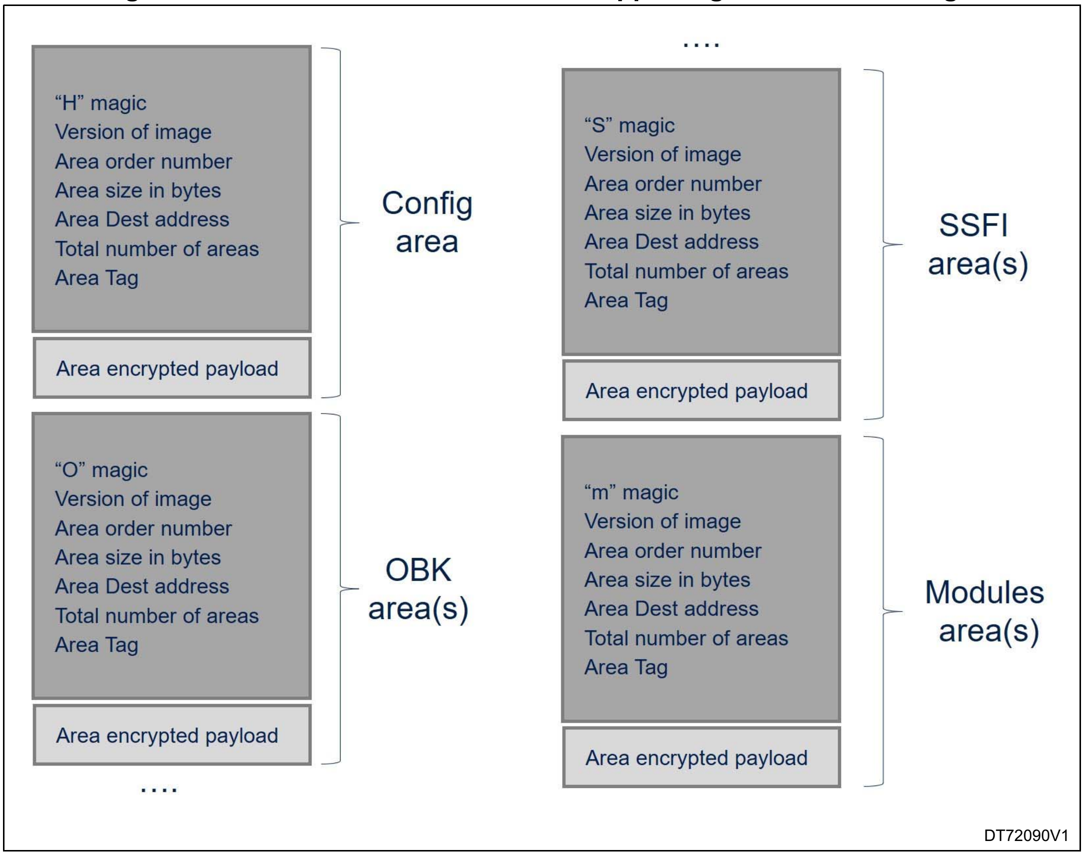
To prepare an SFI image from multiple firmware files, make sure that there is no overlap between their segments. Otherwise, an error message is shown: *"Overlap between segments, unable to merge firmware files"*.

Also, all SFI areas must be located in flash memory, otherwise the generation fails to give the error message: *"One or more SFI areas are not located in flash memory"*.

SFI file structure for devices supporting the secure manager

In addition to the previous area types, the four other types described below are specific to the secure manager usage.

Figure 3. SFI file structure for devices supporting the secure manager



- Area 'H' is an area created by the Trusted Package Creator tool to indicate if there is secure manager information in the SFI file.
- Area 'O' is for OBKey data introduced by .OBK files, each .OBK file being represented by a specific area of type 'O'.
  To perform debug authentication, it is mandatory to provide the .OBK file for DA.
- Area 'S' is for SSFI image (area mainly containing the secure manager). Each area of type 'S' corresponds to at least one area inside the SSFI image. Depending on the available RAM size, the tool tries to merge several SSFI areas into a single area 'S'.
- Area 'm' is for the installation of modules. Each module payload is represented as an area of type 'm' not exceeding 128 Kbytes.

2.2 SFIx preparation process

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

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

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

2.2.1 Area E

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

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

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

2.2.2 Area K

Area ‘K’ triggers random key generation by RSS. It contains N couples, each defining a key area as follows:

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

Example of an area ‘K’:

0x00000080, 0x08010000
0x00000020, 0x08010100

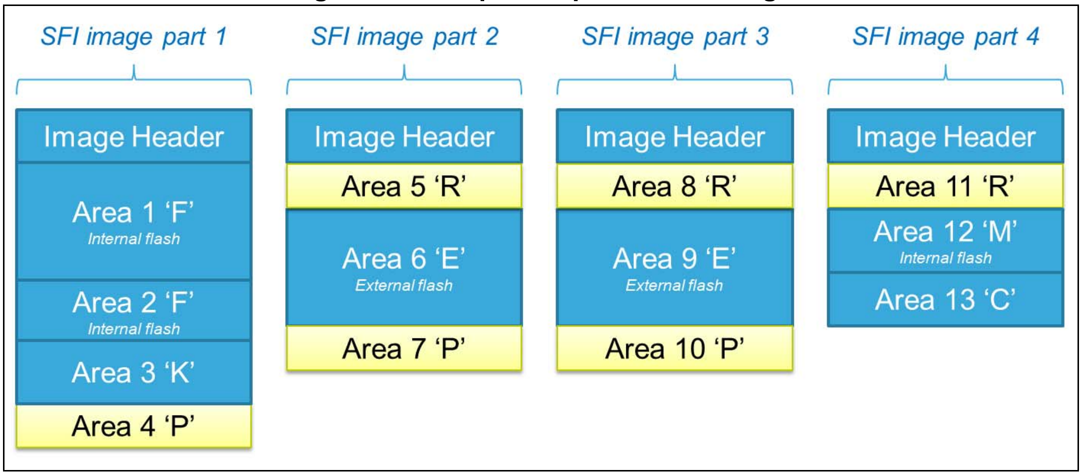
There are two key areas:

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

2.2.3 Constraints for SFIx images

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

Figure 4. Example of split of SFIx image

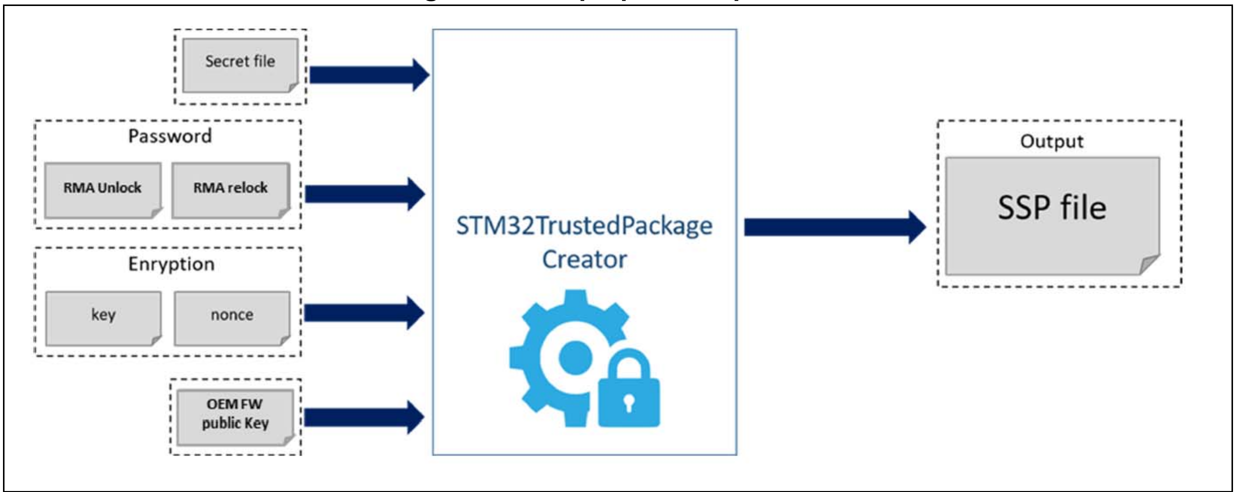


2.3 SSP preparation process

The SSP format is an encryption format for internal firmware created by STMicroelectronics that transforms secret files into encrypted and authenticated firmware in SSP format using an AES-GCM algorithm with a 128-bit key. The SSP preparation process used in the STM32TrustedPackageCreator tool is described in *Figure 5*.

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 side basing on the mentioned inputs.

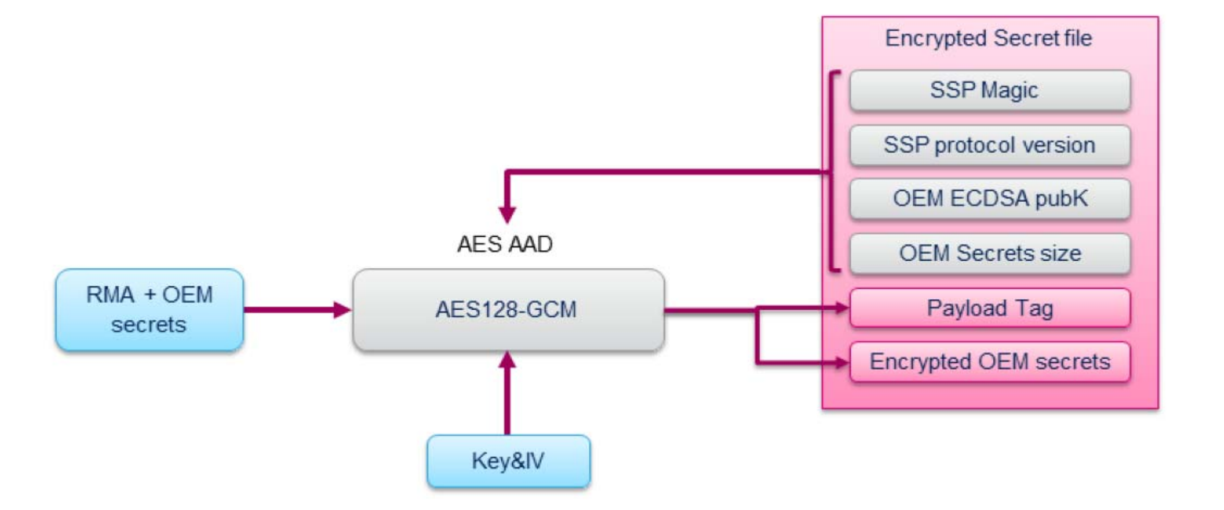
Figure 5. SSP preparation process



- **Secret file**: This secret file is of size 148 bytes, and must fit into the OTP area reserved for the customer. There are no tools or templates to create this file.
- **RMA password**: This is chosen by the OEM and constitutes the first 4-byte word the secret file. To ease RMA password creation and avoid issues, the STM32TrustedPackageCreator tool adds it 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 the decryption. This encrypted file is automatically generated by STM32TrustedPackageCreator tool.

Figure 6. Encryption file scheme



2.4 HSM preparation process

This section describes the steps required to configure a hardware secure module (HSM) to generate firmware licenses for STM32 secure programming using the STM32 trusted package creator.

2.4.1 Why we need an HSM

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

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

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

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

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

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

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

STM32 trusted package creator allows HSM configuration for secure firmware flashing as described in *Section 2.4.2*.

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

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

2.4.2 HSM programming

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

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

This OEM data is:

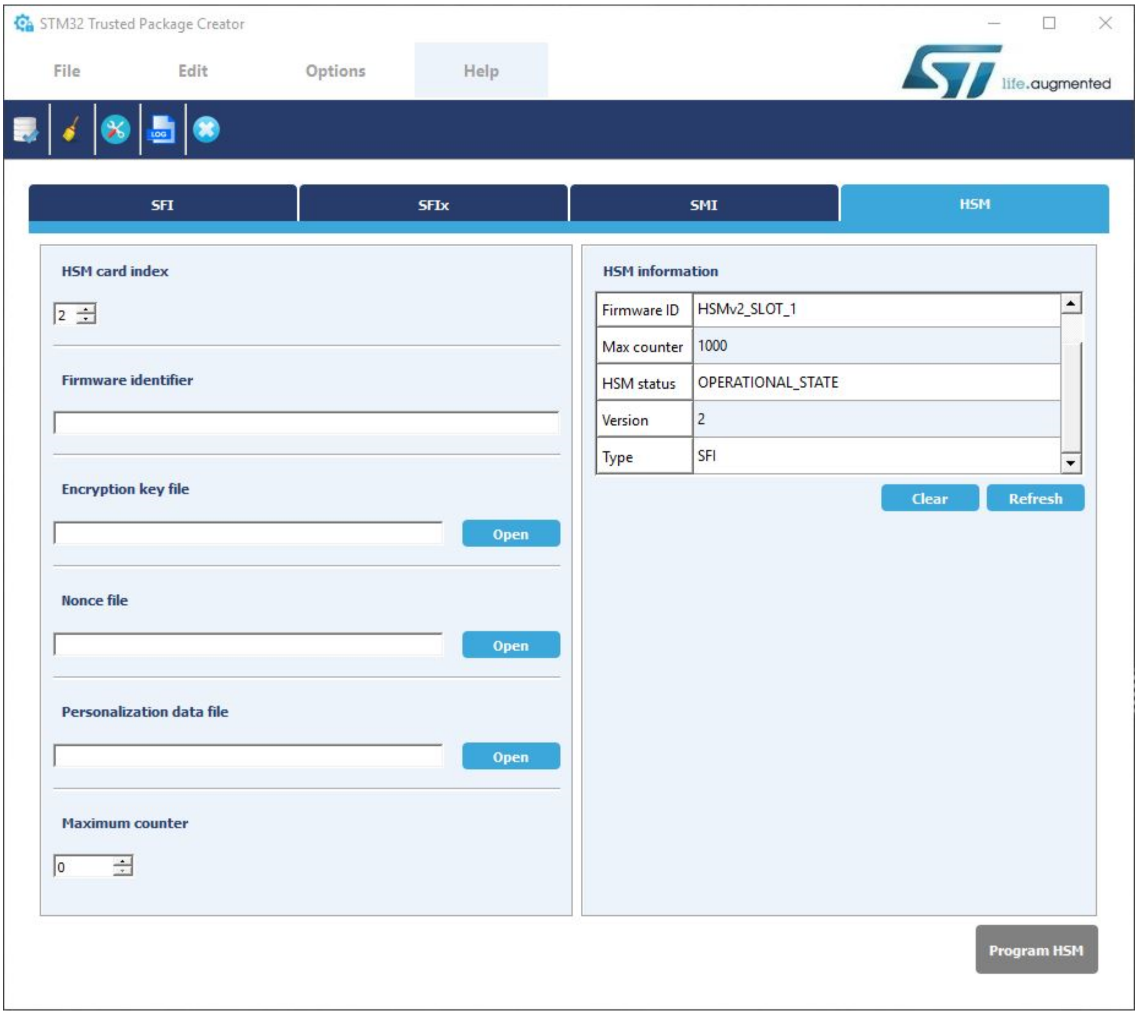
- **Counter**: The counter is set to a maximum value that corresponds to the maximum number of licenses that can be delivered by the HSM, it aims to prevent over programming.
  It is decremented with each license delivered by the HSM.
  No more licenses are delivered by the HSM once the counter is equal to zero.
  The maximum counter value must not exceed the predefined maximum value (1 million units).
- **Firmware key**: This key is 32 bytes and is composed of two fields, the initialization vector (IV) or nonce (first field), and the key (last field) that were used to AES128-GCM encrypt the firmware and generate the SFI file.
  Both fields are 16-byte long. The last 4 bytes of the nonce must be all 0s (only 96 bits of the nonce are used in the AES128-GCM algorithm).
  Both fields must remain secret; Therefore they are encrypted before being sent to the chip.
  The key and the nonce remain the same for all licenses for a given piece of firmware. However, they must be different for different firmware or different versions of the same firmware.
- **Firmware identifier**: Identifies the correct HSM for a given firmware.
- **Personalization data**: 108 bytes that are encrypted before being sent to the chip in cases where each STM32 series has its own configuration.

Once this data is programed into the HSM, the HSM is automatically locked. *Figure 7* shows the GUI provided by the tool for HSM programming by the firmware provider.

Note: The STMicroelectronics personalization data field is available only with HSMv2.

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

Figure 7. HSM programming tab



The tab parameters are as follows:

- **HSM card index:** Specifies the smartcard reader slot number.
- **Firmware identifier:** Identifies the correct HSM for a given piece of firmware.
- **Encryption key file:** A binary file containing the key used to encrypt the firmware and generate the SFI file. The key is 16 bytes long.
- **Nonce file:** A binary file containing the nonce used to encrypt the firmware and generate the SFI file. The nonce is 12 bytes long.
- **Personalization data file**: A binary file containing the personalization data used to configure the given device. The data are 108-byte long.

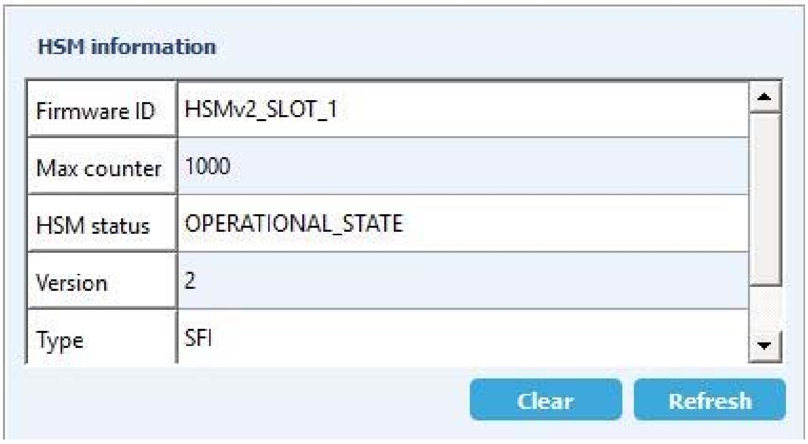
  This option is not required when using HSMv1. If it is used, the tool considers it as an empty input.
- **Maximum counter**: The maximum number of licenses that can be delivered by the HSM (it aims to prevent over programming). The counter value must not exceed a predefined maximum (1 million units).
  When all fields are properly filled in, the user can program the HSM file by clicking on the program HSM button (the button becomes active).

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

2.4.3 Reading HSM information

The programmed information is verifiable using the HSM information panel shown in *Figure 8*:

Figure 8. Reading HSM information



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

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

2.4.4 List of supported HSM cards

The STM32 trusted package creator tool was tested with the following list of smartcard readers:

- eWent USB2.0 smartcard reader (EW1052)
- HID OMNIKEY 3121
- SCM Microsystems SDI011 smartcard reader
- ROCKTEK X001CMYM1L

3 STM32 trusted package creator tool commands

3.1 Command-line interface (CLI)

The following sections describe how to use the STM32 Trusted Package Creator tool from the command-line interface. The available commands are shown in *Figure 9* and *Figure 10*.

Figure 9. STM32 Trusted Package Creator tool available commands (part 1)

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

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


Information options

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

SFI preparation options

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

SMI preparation options

 -smi,   --smi,             : Generate SMI image
                           You also need to provide the information listed below
    -elf,   --elfile        : Input ELF file
        <ELF_File>          : ELF file
    -s,      --sec          : Section to be encrypted
        <Section>           : Section name in the ELf file
    -k,      --key          : AES-GCM encryption key
        <Key_File>          : Bin file, its size must be 16 bytes
    -n,      --nonce        : AES-GCM nonce
        <Nonce_File>        : Bin file, its size must be 12 bytes
```

Figure 10. STM32 Trusted Package Creator tool available commands (part 2)

```
    -sv,     --sver          : Security version
        <SV_File>            : Its size must be 16 bytes
    -o,      --outfile       : Generated SMI file
        <Output_File>        : SMI file to be created
    -c,      --clear         : Clear ELF file
        <Clear_File>         : Clear ELF file to be generated

SFU preparation options

 -sfu,   --sfu               : Generate SFU image,
                            You also need to provide the information listed below
    -fir,    --firmware      : Add an input firmware file (must have only 1 segment)
        <Firm_File>          : Supported firmware files are ELF HEX SREC BIN
        [<Address>]          : Only in case of BIN input file (in any base)
     -k,       --key         : AES-GCM encryption key
         <Key_File>          : Bin file, its size must be 16 bytes
    -n,      --nonce         : AES-GCM nonce
         <Nonce_File>        : Bin file, its size must be 12 bytes
    -v,      --ver           : Image version
        <Image_Version>      : Its value must be in <0..255> (in any base)
    -oh,       --outheader   : Generated SFU header file
         <Output_File>       : SFU header file to be created
    -os,       --outsfu      : Generated SFU encrypted image file
       <Output_File>         : SFU encrypted image file to be created

SSP Payload preparation options

 -ssp,   --ssp            : Generate SSP Payload,
                             You also need to provide the information listed below
     -b,    --blob           : Binary to encrypt
         <Blob>              : Binary file
    -pk,    --pubk           : ECDSA pubK
        <PubK>               : binary file of size 178 bytes
    -k,      --key           : AES-GCM encryption key
        <Key_File>           : Bin file, its size must be 16 bytes
    -n,      --nonce         : AES-GCM nonce
        <Nonce_File>         : Bin file, its size must be 16 bytes
    -s,      --secretsize    : Secret Size
        <Secret_Size>        : number in any base
    -o,      --out           : Generated payload
        <Output_File>        : Payload file to be created

HSM provisioning

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

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

3.2 HSM provisioning command

-hsm, --hsm

Description: This command allows the HSM card programming.

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

-i, --index

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

Syntax: -i <number>

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: -k <Key_File>

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

-n, --nonce

Description: Set the AES-GCM nonce.

Syntax: -n <Nonce_File>

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

-id, --id

Description: Set the firmware identifier.

Syntax: -id: Input as a string with a maximum length of 15 characters. Do not use spaces.

-mc, --max counter

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

Syntax: -mc <Max_Counter>

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

-pd, --persoData

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

Syntax: -pd <PersoData_File>

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

-info, --info

Description: Get HSM information.

-hash, --hash

Description: Calculate the input firmware hash to check the firmware. Integrity (only for STM32H73). By default, the value of hash is 0.

Syntax: --hash <0/1>

Example of HSM version 1 provisioning:

STM32TrustedPackageCreator_CLI -hsm -i1 -k "C:\TrustedFiles\key.bin" –n "C:\TrustedFiles\nonce.bin" -id HSMv1_SLOT_1 -mc 2000

Example of HSM version 2 provisioning:

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

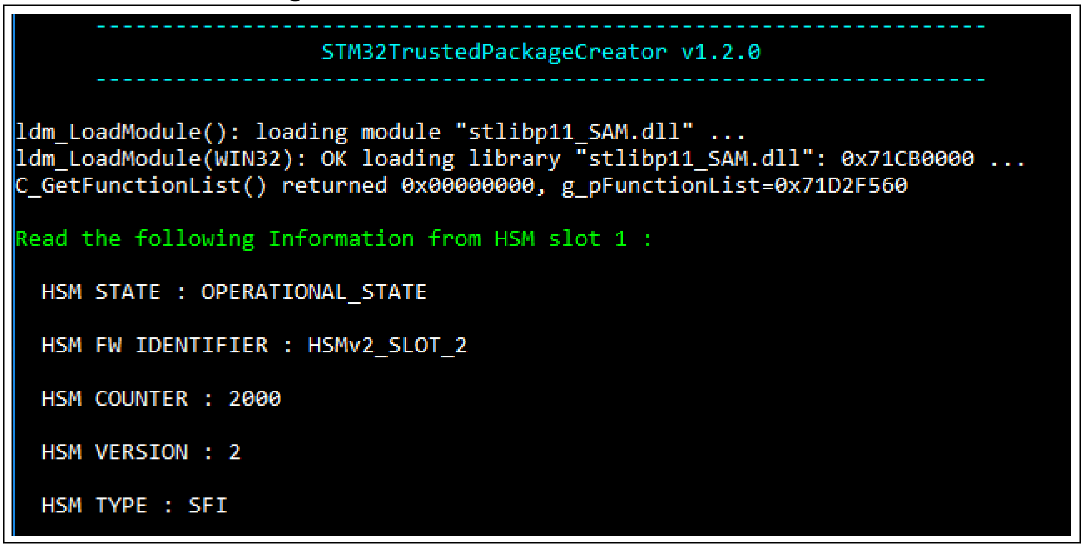
STM32TrustedPackageCreator_CLI -hsm -i1 -k "C:\TrustedFiles\key.bin" –n "C:\TrustedFiles\nonce.bin" -id HSMv2_SLOT_2 -mc 2000 -pd "C:\TrustedFiles\enc_ST_Perso_L5.bin"

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

Example of HSM get information

STM32TrustedPackageCreator_CLI -hsm –i 1 –info

Figure 11. Get HSM information in CLI mode



Example of HSMv2 personalization data

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

The STM32TrustedPackageCreator tool provides all personalization package files, ready to be used on SFI/SFIx and SSP flows. To obtain all the supported packages, go to the **PersoPackages** directory residing in the tool’s install path.

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

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

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

  ```
  STM32_Programmer_CLI -c port=swd -gc "certificate.bin"
  ```
- A file named “certificate.bin” is created in the same path of the STM32CubeProgrammer executable file.
- Open the certificate file with a text editor tool, then read the eight characters from the header that represent the product ID.
- For example:

  When using a STM32H7 device, you find: 45002001

  When using a STM32L4 device, you find: 46201002

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

ProdcutID_FlowType_LicenseVersion_SecurityVersion.enc.bin

For example: 47201003_SFI._01000000_00000000.enc.bin

Based on this name, the associated information is:

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

3.3 SFI/SFIx generation command

-sfi, --sfi

Description: This command generates an SFI image file.

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

-devid, --deviceid

Description: specify deviceID. If this option is not used, P and R areas are generated by default for all devices. This option is mandatory to avoid various issues with some devices.

Syntax: -devid <device_id>

`[<device_id>]: Device ID`

-fir, --firmware

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

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

`[<Firmware_file>]: Firmware file`

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

-firx, --firmwx

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

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

`<Firmware_file>`: Supported external firmware files are ELF, HEX, SREC, BIN.

[`<Address>`]: Only in the case of BIN input file (in any base).

[`<Region_Number>`]: Only in the case of BIN input file (in any base):

[0:3]: OTFD1 (STM32H7A / STM32L5)

[4:7]: OTFD2 (STM32H7A3/7B3 and STM32H7B0 case)

[`<Region_Mode>`]: Only in case of BIN input file (in any base), only two bits

[0:1] where
00: instruction only (AES-CTR)
01: data only (AES-CTR),
10: instruction + data (AES-CTR)
11: instruction only (EnhancedCipher)

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

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: `-k <Key_file>`

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

-kx, --keyx

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

Syntax: **`-kx`** `<Key_area_File>`

-n, --nonce

Description: Set the AES-GCM nonce.

Syntax: `-n <Nonce_file>`

<Nonce _file> : A 12-byte binary file.

-v, --ver

Description: Set the image version.

Syntax: `-v <Image_version>`

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

-ob, --obfile

Description: Provide an option bytes file.

Syntax: `-ob <CSV_file>`

`<CSV_file>`: A csv file with nine values.

--rs, --ram size

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

Syntax: -rs <size>

< size >: Size in bytes.

-ct, --token

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

Syntax: -ct <Address>

-o, --outfile

Description: Set the SFI file to be created.

Syntax: `-o <out_file>`

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

-hash, --hash

Description: Generate hash for integrity check.

Syntax: -hash <Flag>

-obk, --ob-keys

Description: Optional to insert multiple option bytes keys for the devices supporting this functionality.

Syntax: `-obk, --ob-keys <path1.obk> <path2.obk>...`

-ssfi, --sfi

Description: Optional to deploy an SSFI image in an SFI flow.

Syntax: `-ssfi, --sfi <ssfi_path.ssfi>`

-mcsv, --mcsv

Description: Optional to indicate a list of modules in an SFI image.

Syntax: `-mcsv, --mcsv <modules_path.mcsv>`

-ecsv, --ecsv

Description: Optional to indicate a list of external modules in an SFIx image.

Syntax: `-ecsv, --ecsv <exmodules_path.ecsv>`

Example for SFI:

With an ELF file:

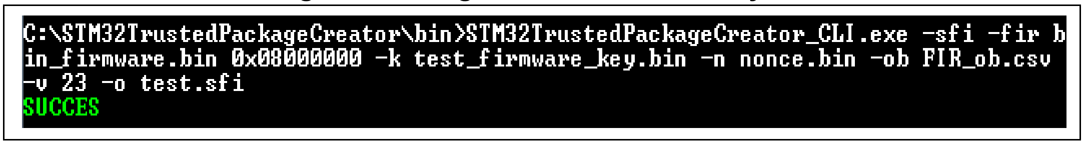
```
STM32TrustedPackageCreator_CLI -sfi   -fir   ELF_firmware.axf -k
test_firmware_key.bin   -n   nonce.bin   -ob   FIR_ob.csv   -v
23   -o   test.sfi
```

Figure 12. SFI generation with an ELF file

With a binary file:

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

Figure 13. SFI generation with a binary file



3.4 SSP generation command

The STM32 trusted package creator tool CLI exports an SSP command with various options to validate the SSP generation request.

-ssp, --ssp

Description: This command generates an SSP image file. In order to generate an SSP image, the user must provide the mandatory inputs by using the options listed below.

-b, --blob

Description: Binary to encrypt

Syntax: -b <Blob>

<Blob> : Secrets file of size 148 bytes

-pk, --pubk

Description: OEM key file

Syntax 1: -pk <PubK.pem> : pem file of size 178 bytes for STM32MP15xx
Syntax 2: -pk <rpkht.bin> : binary file of size 32 bytes for STM32MP13xx

-k, --key

Description: AES-GCM encryption key

Syntax: -k <Key_File>

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

-n, --nonce

Description: AES-GCM nonce

Syntax: -n <Nonce_File>

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

-o, --out

Description: Generate SSP file

Syntax: -out <Output_File.ssp>

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

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

Command example:

```
STM32TrustedPackageCreator_CLI -ssp
-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 finished successfully, otherwise an error occurs.

3.5 OBKey generation

-obk, --obk
Description: This command generates OBKey files used by the STM32CubeProgrammer to program the OBKey content.
Syntax: -obk <input_xml_file>
<input_xml_file> : An input XML file describing the OBKey region structure.

For more details, refer to *Appendix A*.
Example:

```
STM32TrustedPackageCreator_CLI.exe -obk C:\OBKey.xml
```

3.6 Image generation

-pb, --pb

Description: This command generates a signed/encrypted image using the imgtool.

Syntax: -pb <input_xml_file>

<input_xml_file>: An input XML file describing the commands/parameters to be passed to the imgtool.

For more details, refer to *Appendix B*.
Example:

```
STM32TrustedPackageCreator_CLI.exe -pb C: \ ImageGen.xml
```

Note: Only one command is executed at a time. It is not possible to use -obk and -pb at the same time.

3.7 Generation of encryption and authentication keys

-genkey, --generatekey

Description: This command generates encryption and authentication keys.

Syntax: -genkey -inkey <input_key> -keytype <key_type>

<input_key> : Key file to be generated

<key_type> : Possible Key types: ecdsa-p256, rsa-2048, rsa-3072

If <input_key> exists, the old key is renamed, and a new pair of keys is generated.

Example:

```
STM32TrustedPackageCreator_CLI.exe -genkey -inkey C:\KeysGen\enc-ec256-priv.pem -keytype ecdsa-p256
```

3.8 STM32WB firmware signing

-sign, --sign

Description: SignS a binary image using ECDSA algorithm and saveS the public key in a binary file in the same folder as the private key. provide the information listed below.

`-bin, --binary`: Input firmware image in a binary format to be signed

`<Img_File>`: Binary image file path

`-prvk, --privatekey` : EC private key in PEM format with Prime256v1 curve.

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

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

`<prvk_File>` : Private key file path

`-v, --version` : Signed image version

`<Img_Ver>` : Version number. Must be an integer.

`-o, --outfile` : Output image file path

`<Output_File>` : Signed image file path

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

For more information about these STM32CubeProgrammer commands, refer to UM2237 *[1]*.

Note: This section does not concern STM32WB0

3.9 SSP backup generation

The backup memory is data that is stored in the SSP-SFI area of type "B", this memory is programmed during the SSP sequence and serves as a backup memory for the device.

The STM32 trusted package creator tool CLI exports a Backup command to generate a backup binary which may be used in next step with SSP SFI image.

--backup-gen, -backupgen

Description: Generate SSP backup binary.

Syntax: --backup-gen or -backupgen -in <input_json> -out/-o <output_binary>

<input_json>: Input Json file path describing the backup memory files.

<output_binary>: Backup file to be created with (extension .bin)

Example:

```
--backup-gen -in "C:\STM32\SSP\Backup\backup-out.json" -o
"C:\STM32\SSP\Backup\backupOutput.bin"
```

If you do not already have a JSON file, you can generate or export one using the GUI interface through the Backup Gen tab (see *Section 4.6*).

The JSON file has the capability to contain multiple input files.

The tool verifies the integrity of the JSON file and checks its format validity to ensure that it is properly structured.

3.10 Key generation and FW signature (STM32WB0x/STM32WL3x only)

-keygen, --keygen

Description: This command generates a new key pair (public and private).

Syntax: -keygen -k <Output_File_priv_path> <Output_File_pub_path> -f <public_key.c_path> <public_key.py_path> <Information.txt>

<Output_File_priv_path> : Output file for private key

<Output_File_pub_path> : Output file for public key

<public_key.c_path> : Output file for public key

<public_key.py_path> : Output file for public key

<Information.txt> : Output file contains information about the public Key

Example:

```
>> STM32TrustedPackageCreator_CLI.exe -keygen -k C:\ private.pem C:\
public.pem -f C:\public_key.c C:\ public_key.py C:\ public_key.txt
```

-signbin, --signbin

Description: This command generates a signed image

Syntax: -signbin -k <Input_File_priv_path> -f <binary_to_sign_path> -o <output_signed_bin_path>

<Input_File_priv_path> : Input file for private key

<binary_to_sign_path> : Input binary to sign

<output_signed_bin_path> : Output signed file path

Example:

```
>> STM32TrustedPackageCreator_CLI.exe -signbin -k C: \test\priv_key.pem -f
C: \test\gpio.bin -o C:\C_Project\test\gpio_Signed.bin
```

3.11 SSP-SFI image generation

The STM32 trusted package creator tool CLI exports a SSP-SFI command with various options to validate the SSP-SFI generation request.

-ssp-sfi, --ssp-sfi

Description: This command generates an SSP image file in SFI format. In order to generate an SSP-SFI image, the user must provide the mandatory inputs by using the options listed below.

-b, --blob

Description: Secrets binary to encrypt

Syntax: -b <blob>

<blob>: Secrets file path

-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 12 bytes.

-backup, --backup

Description: The backup memory to deploy

Syntax: -backup <Backup_File> <Base_Address>

<Backup_File> : Binary file path

< Base_Address> : The RAM base address dedicated for the backup

-iv, --image-version

Description: The SSP-SFI image version

Syntax: -iv <Image_Version>

< Image_Version >: Its value must be in <0..255> (in any base).

-o, --out

Description: Generate SSP-SFI file

Syntax: -out <Output_File>

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

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

Command example:

```
--ssp-sfi -b secrets.bin -iv 1 -k key.bin -n nonce.bin -backup backup.bin
0x0C000000 -o out-sfi.ssp
```

Once the operation is done, a green message is displayed to indicate that the generation finished successfully displaying the list of the SFI areas, otherwise an error occurs.

3.12 Regeneration of encryption and authentication keys

-genkey, --generatekey
Description: This command generates encryption and authentication keys.
Syntax: -genkey -inkey <input_key> -keytype <key_type>

<input_key>: An input key file to be regenerated.

<key_type>: A key type.

Example:

```
>> STM32TrustedPackageCreator_CLI.exe -genkey -inkey C:\KeysGen\enc-ec256-
priv.pem -keytype ecdsa-p256
```

3.13 MCE image generation

Definition: Refer to the STM32 reference manual for more details about the MCE (Memory Cipher Engine) hardware feature.

The Trusted Package Creator role is to encrypt the firmware with a given symmetric 16-byte key to protect data located in external memory.

Syntax:

-mce -app <Application_Path> **-key** <Key_Path> **-directory** <Output_Folder>

-add <Start_Address> **-mode** <mode>

Application_Path: Path to the firmware's binary file to be encrypted.

Key_Path: Path to the key's binary file required for derivation.

Output_Folder: Location where the encrypted image will be generated.

Start_Address: Address to use when programming the image into external flash after encryption.

mode: Encryption mode (0 for **Block + Normal**, 1 for **Block + Fast**, 2 for **Noekeon**).

4 STM32 trusted package creator tool graphical user interface (GUI)

4.1 SFI generation

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

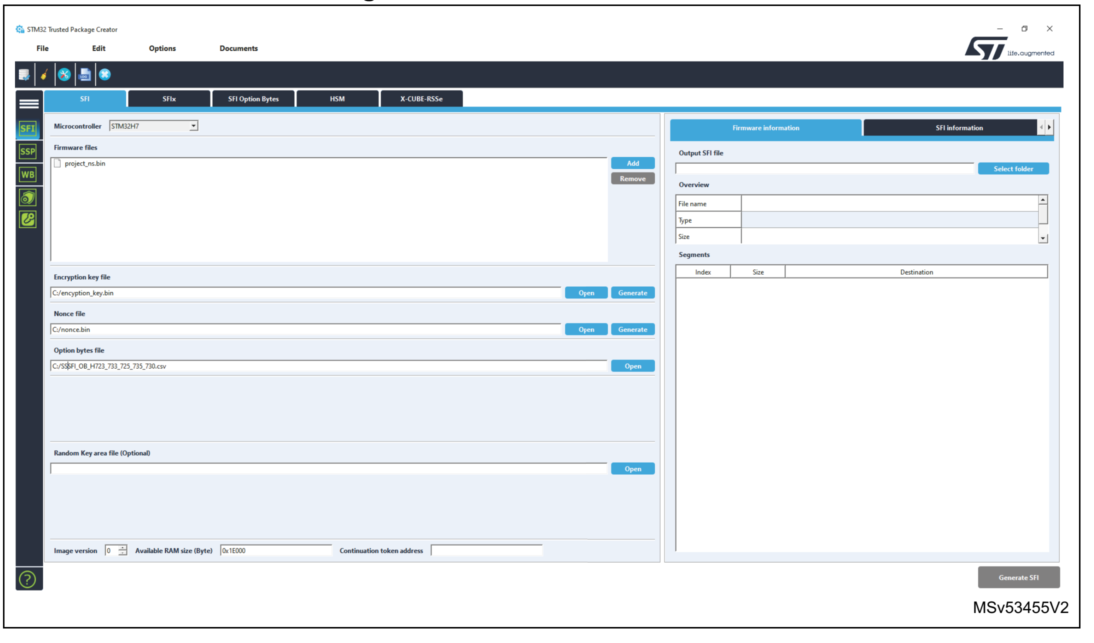
Firmware files

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

Note: If the file is valid, it is added to the firmware files list. Selecting it makes several pieces of related information appear in the firmware information section (Figure 14), otherwise an error message box is shown saying that the file could not be opened, or is not valid.

*If the file is in a binary format, a dialog box appears requesting that an address be provided. A file can be removed with the **Remove** button.*

Figure 14. Firmware file addition



Encryption key and nonce file

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

Option bytes file

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

Image version

Image version value in [0..255].

RAM size

Size of RAM memory available for SFI programming.

Continuation token address

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

Output file

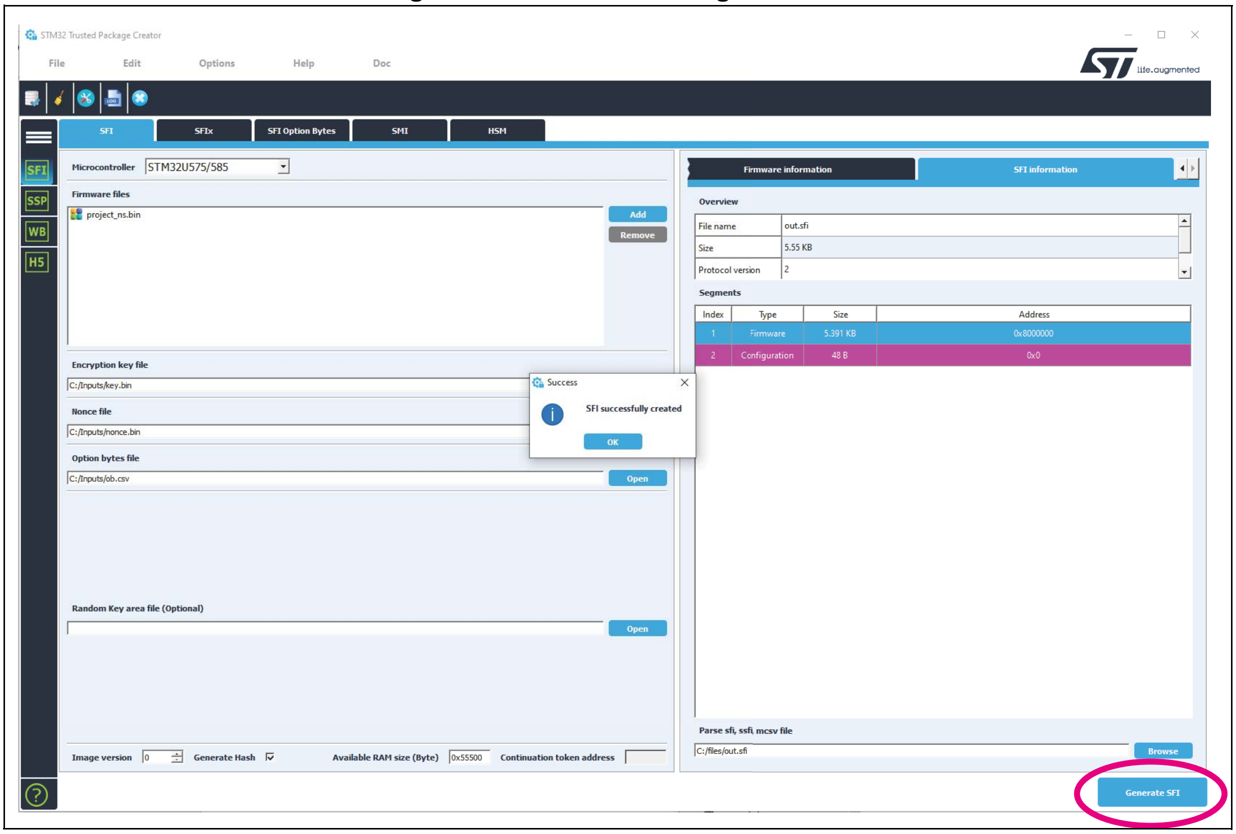
An output file can be selected by entering its path (absolute or relative), or with the select **folder** button, note that with the latter way, a name *out.sfi* is suggested, you can keep or change it.

Generate hash

When the *generate hash* option is checked, the STM32 trusted package creator calculates the input firmware hash and integrates it into the SFI firmware. The STM32H73 MCU is able to use this hash input to check the firmware integrity.

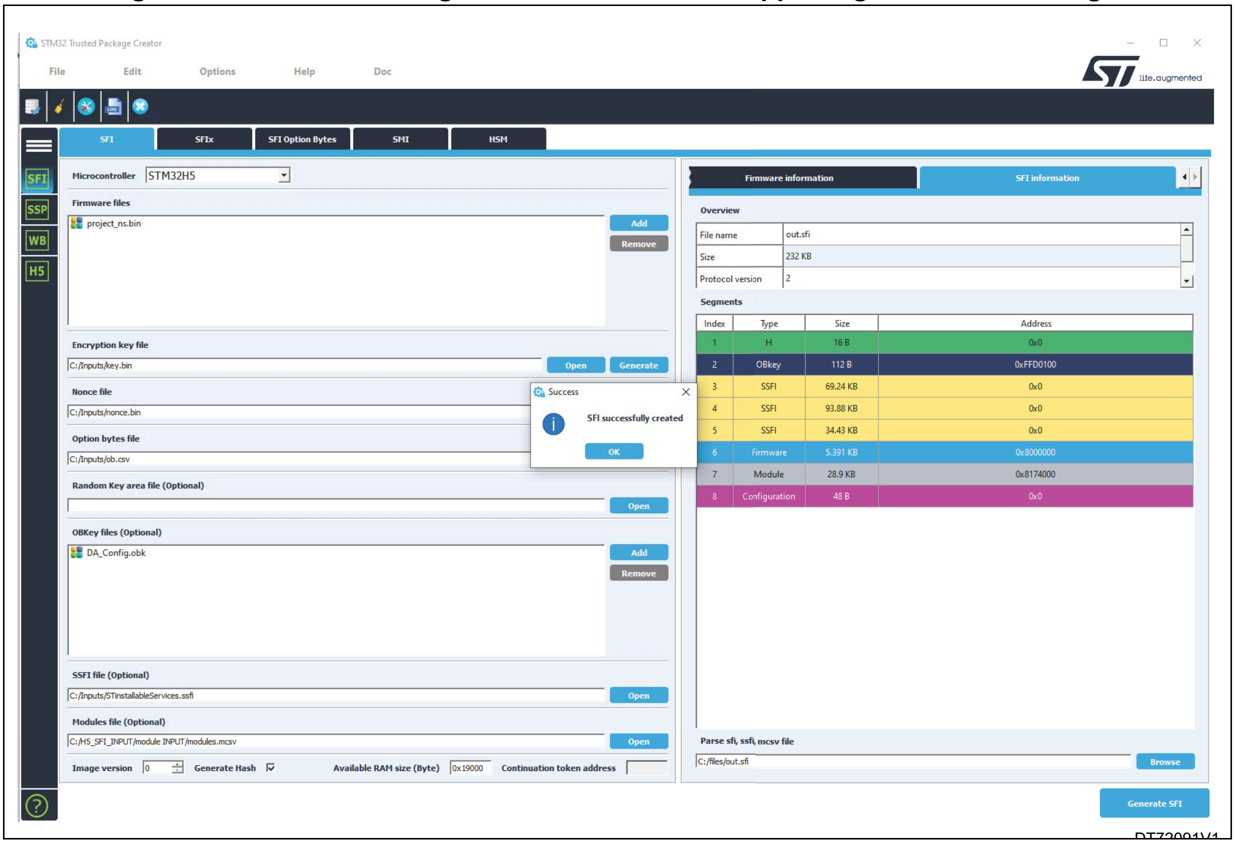
When all fields are properly filled in, the **Generate SFI** button becomes active. The user may generate the SFI file by clicking on it. A message box indicating successful generation appears (*Figure 15*) and information about the generated SFI file is displayed in the SFI information section.

Figure 15. Successful SFI generation



The devices supporting the secure manager include new security features, which can be configured at the SFI installation. The tool exposes new graphical components.

Figure 16. Successful SFI generation for devices supporting the secure manager



OBKey files

Possibility to add multiple OBKey files having the .OBK extension.

SSFI file

Select an SSFI image.

Modules file

Select an MCSV file that includes the list of modules to be installed.

A modules file is an input listing the modules with their associated information according to this syntax:

`ModulePath, LicensePath.bin` or `HSM FW ID, DestinationAddress`

Use # at the beginning of a line to ignore it:

`#ModulePath, LicensePath.bin` or `HSM FW ID, DestinationAddress`

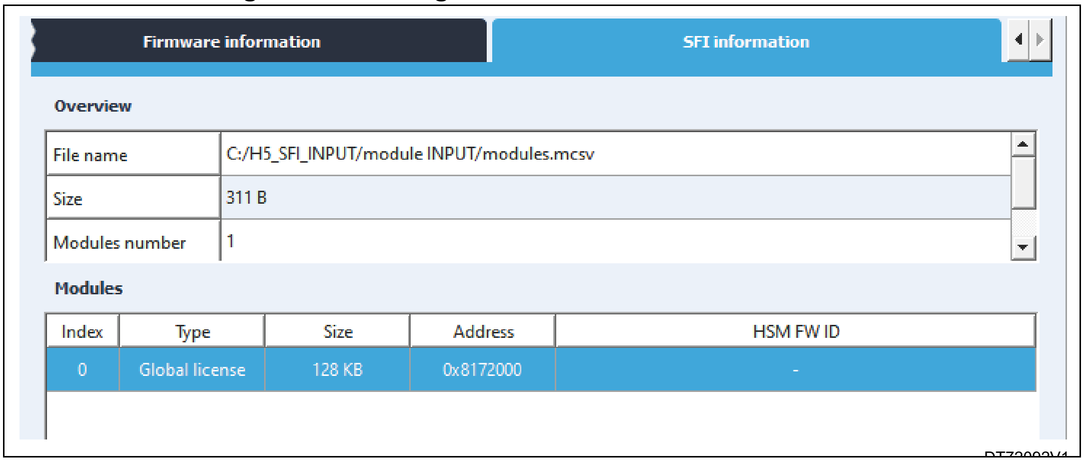
- Module path: file path of the payload module having the .smu or the .bin extension
- LicensePath.bin or HSM FW ID: use the license path for the global license install and the HSM FW ID for chip specific install.
- DestinationAddress: the target address to program the module

Each path can be a full path or a relative path. In the case of a relative path, the root directory is the folder where the MCSV file is located.

Example:

```
./../License/module0.smu, ./../License/LicenseV0.bin, 0x8172000
```

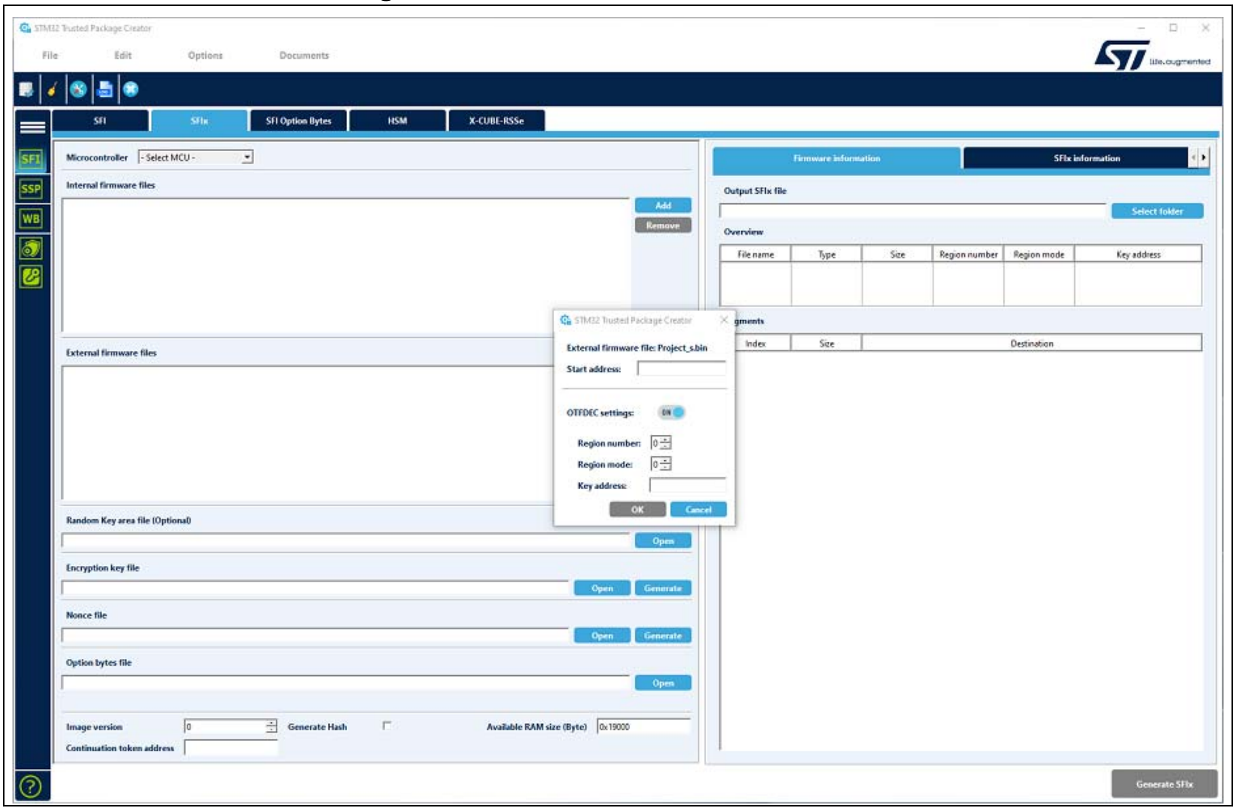
Figure 17. Parsing of the MCSV file for modules list



4.2 SFIx generation

- The user must enter the input for external firmware files. If the file is valid, several pieces of related information appear in the firmware information section. A popup also appears that contains fields to be filled in (Figure 18).
- Start address: Mandatory option, the start address of a binary.
- OTFDEC settings: Switch ON/OFF button.
  - In case OTFDEC settings ON: Set your customized value of region number, mode, and key address.
  - In case OTFDEC settings OFF: Region number, mode, and key address fields are disabled and set to the disabled value (0xFFFFFFFF).
- Region number: The OTFDEC configuration detailed in Section 2.2: SFIx preparation process.
- Region mode: The OTFDEC configuration detailed in Section 2.2: SFIx preparation process.
- Key address: The address of the key.

Figure 18. External firmware file selection



Output file

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

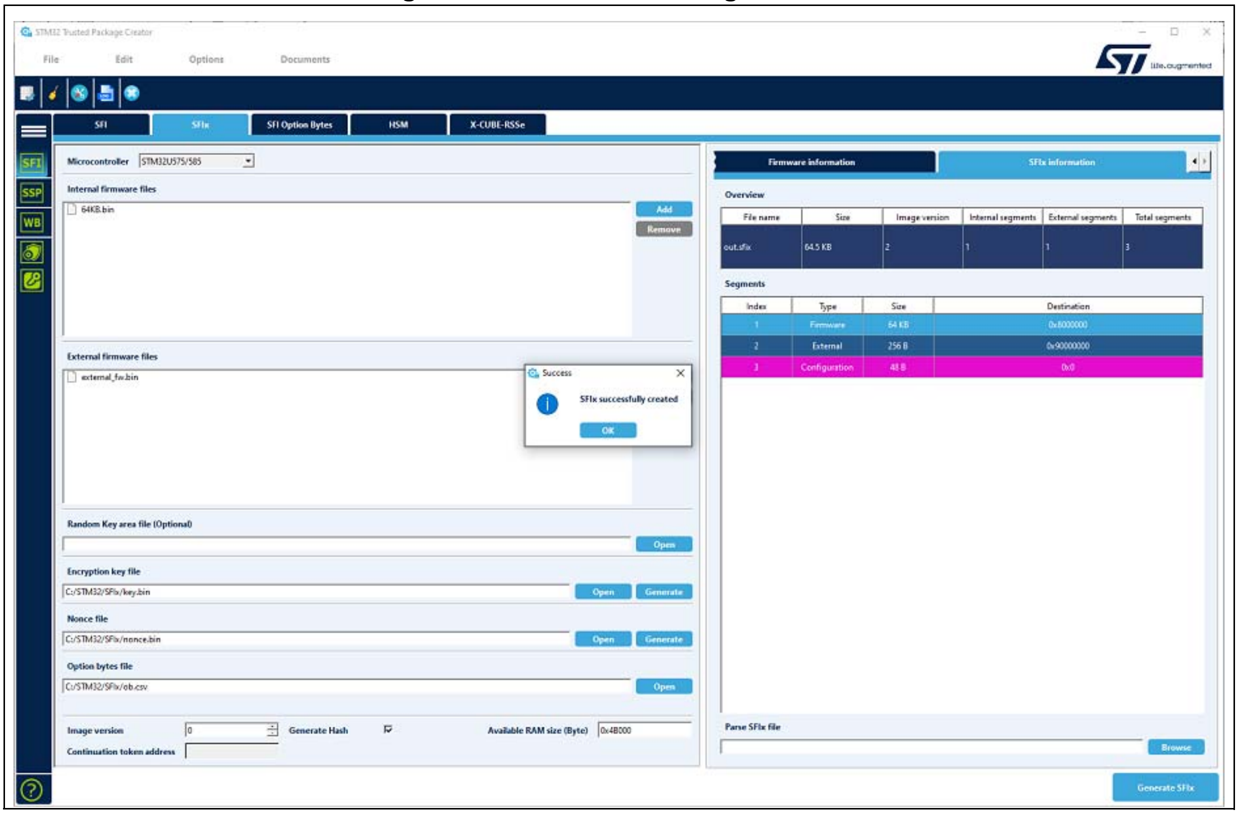
When all fields are properly filled in, the **Generate SFIx** button becomes active. The user may generate the SFIx file by clicking on it.

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

The SFIx information section (*Figure 19*), contains two sections:

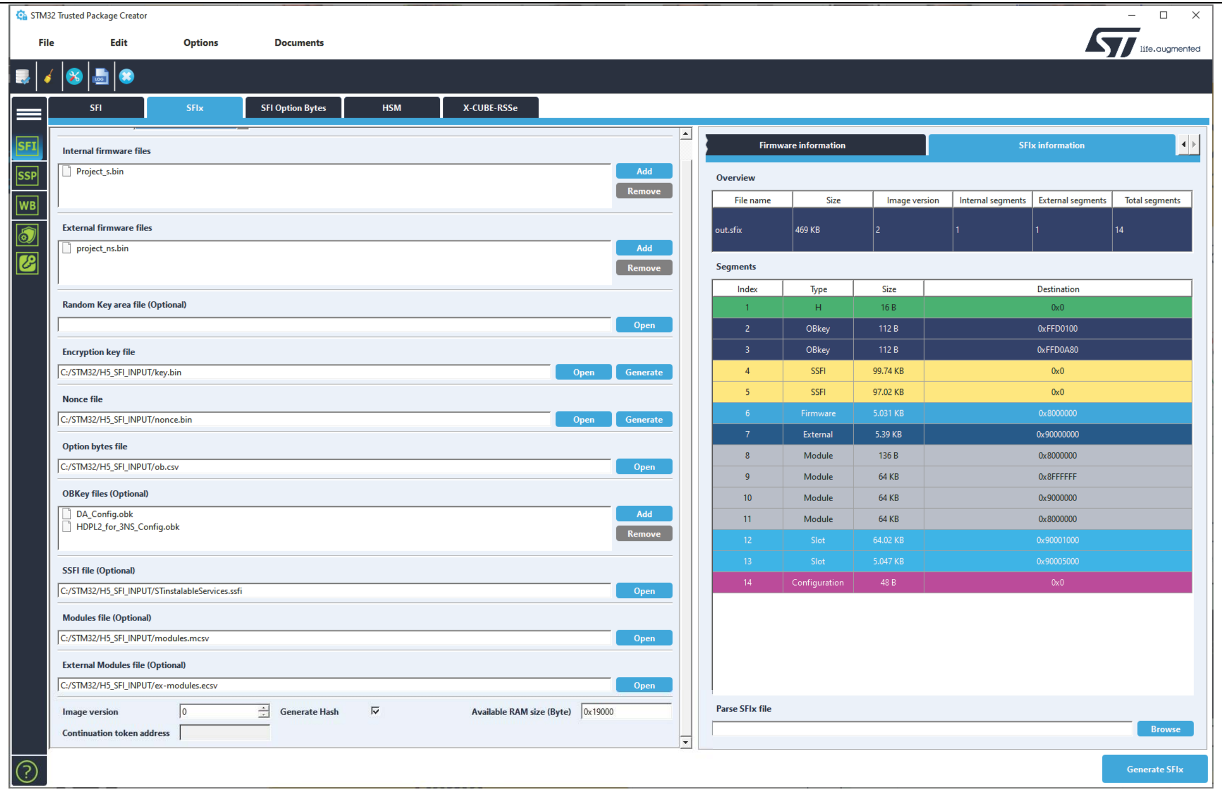
- An overview section mentioning the name of the output file and various information including the size of the file, its image version, internal/external segments and so on.
- A ‘segments’ section showing the different areas.
- A ‘parse SFIx file’ section, allowing the user to browse a *.sfix* output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

Figure 19. Successful SFIx generation



The SFIx window for devices supporting the secure manager displays three additional graphical components (OBK files, SSFI, and Modules), already covered in *Section 4.1*.

Figure 20. SFIx window for devices supporting the secure manager



External modules file

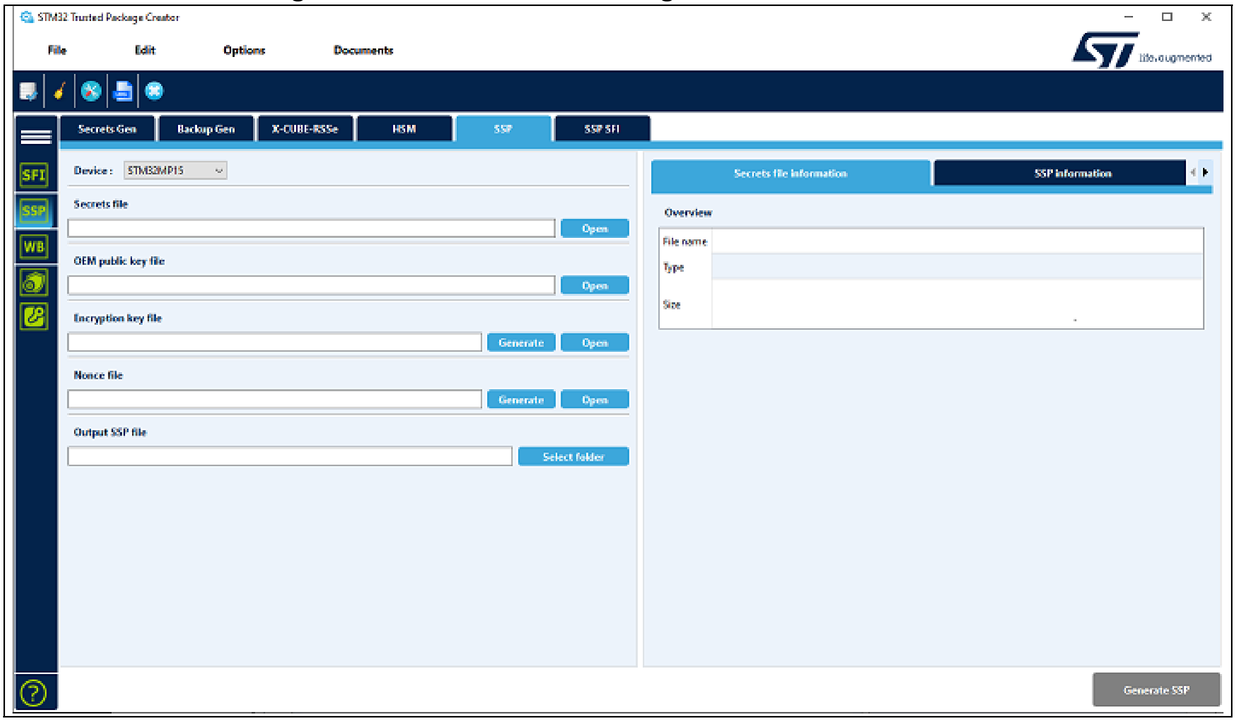
Select an ECSV file that includes the list of external modules to be installed. An external modules file is an input listing the modules with their associated information according to a specific syntax (same as MCSV already described for the modules file MCSV).

4.3 SSP generation

4.3.1 SSP image generation

The STM32 Trusted Package Creator tool GUI presents an SSP tab to generate an SSP image. The user must fill in the input fields with valid values.

Figure 21. STM32TrustedPackageCreator SSP GUI table



Device

The SSP image generation is different between **STM32MP15xx** and **STM32MP13xx** devices. The user can use the combo box to select the current device to be used.

Note: *RMA password is deprecated from SSP Generation. It is possible to manage it using the Secrets Gen panel.*

Secrets file

Binary file of size 148 bytes to encrypt. Can be selected by entering its path (absolute or relative), or by selection with the open button. Refer to *212 bytes for STM32MP13xx*.

OEM public key file / OEM RPKTH

This component has dynamic handling to present the input file of the OEM key.

OEM public key file for STM32MP15xx devices, PEM file of size 178 bytes.

OEM Root public Key Table hash for STM32MP13xx devices, bin file of size 32 bytes.

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 16 bytes for nonce).

Generate button

Is a flexible way to generate and save a random key/nonce by choosing the output folder (the same key/nonce must be provisioned on HSM) and you can ignore this button if you are already the necessary key/nonce.

Output SSP file

Select the output directory by mentioning the SSP file name to be created with an .ssp extension. It can be selected by entering paths (absolute or relative), or with the select folder button. Note that in the latter case, a name is suggested, which can be kept or changed. When all fields are properly filled in, the user can start the generation by clicking on the **generate SSP** button (the button becomes active).

Once the generation is done, we can get SSP information from the SSP overview section.

Figure 22. SSP output information for STM32MP15xx

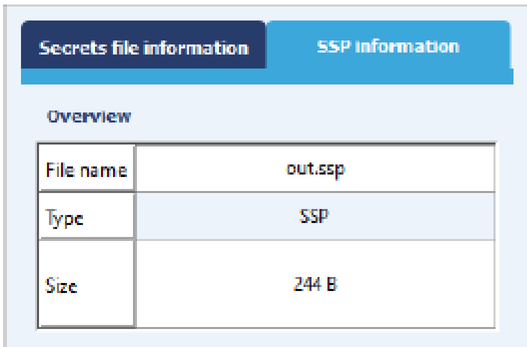
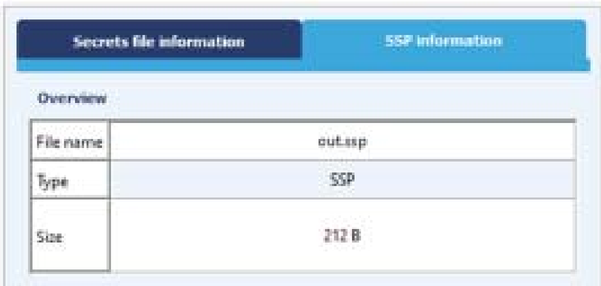


Figure 23. SSP output information for STM32MP13xx



File name: SSP output file name.

Type: SSP extension.

Size: indicates the generated file size including all data fields.

244 bytes for STM32MP15xx

212 bytes for STM32MP13xx

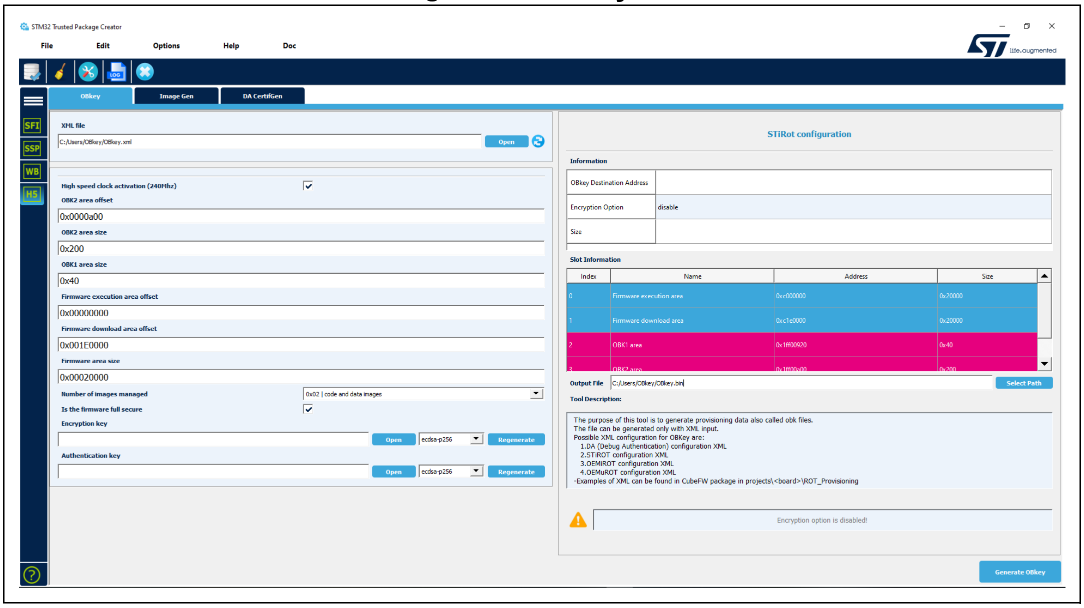
4.4 Security features

4.4.1 OBKey content generation

The STM32 Trusted Package Creator tool presents an OBKey tab. This tab enables generating the .OBK files used by the STM32CubeProgrammer to program the OBKey content.

The tool input is an XML file describing the OBKey region structure.

Figure 24. OBKey tab



XML file

It is the input XML file path to be loaded. For more details, refer to *Appendix A*.

Generate OBKey

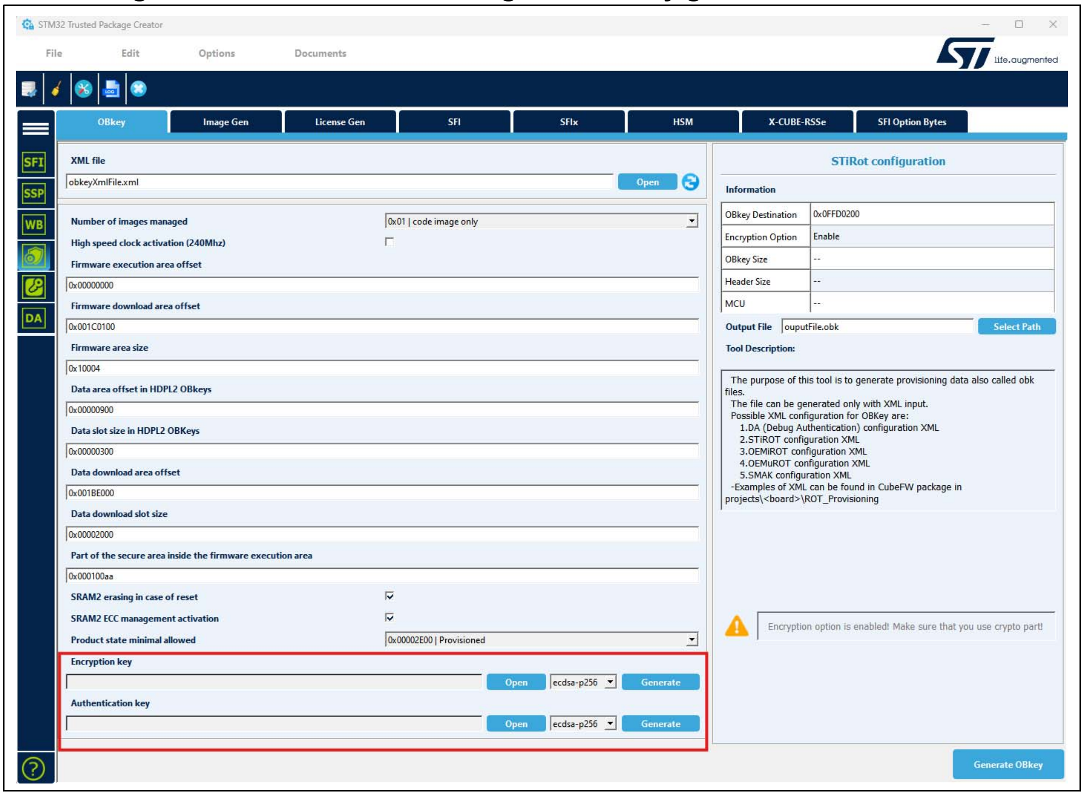
This is a flexible way to generate and save a .OBK file by choosing the output file path.

A key generation mechanism is offered to users to generate encryption and authentication keys by selecting a path, then clicking on Generate button.

Case 1: the key's path field is empty, a new private and public key is generated.

Case 2: the key's path field contains a key, the old key is renamed, and a new private and public key are generated.

Figure 25. STM32TrustedPackageCreator key generation mechanism

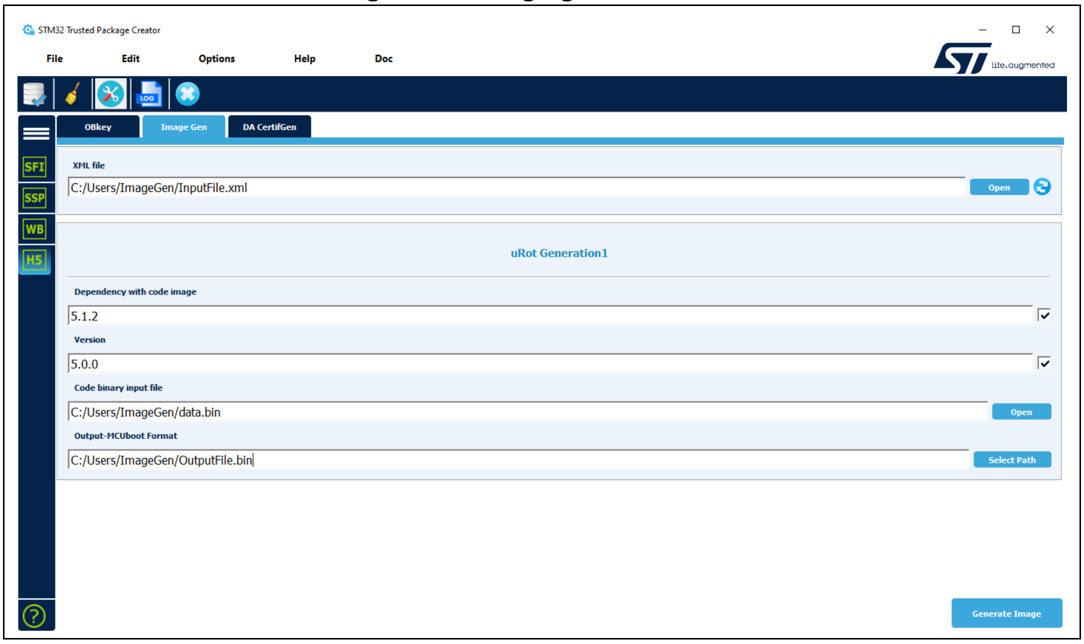


4.4.2 Image generation

The STM32 Trusted Package Creator tool presents an image generation tab to generate signed and encrypted image using the imgtool.

The tool input is an XML file describing the commands and parameters to be passed to the imgtool. The imgtool is embedded in the STM32CubeProgrammer package.

Figure 26. Image generation tab



XML file

It is the input XML file path to be loaded. For more details see *Appendix B*.

Generate image

It is a flexible process to generate an image by choosing the output folder in **output-mcuboot** format region.

4.4.3 Debug authentication

The STM32Trusted Package Creator tool presents a debug authentication panel (DA) with two tabs to generate the files for the debug authentication process.

The first tab is “OBkey Generation” (see *Figure 24*), responsible of the generation of the .OBK files used by the STM32CubeProgrammer to program the OBKey area containing user keys and permissions.

Select the target MCU, then select whether it is targeting a certificate or password based debug authentication. The user must provide the debug authentication root key (the tool gives the possibility to generate this key using “Generate” button), and the permissions.

The second tab is “Certificate Generation” (see *Figure 26*). It generates a signed certificate using an Arm tool (PSA ADAC) for debug authentication. The signed certificate is used as input to STM32CubeProgrammer for debug authentication and for certificate chaining.

For more details, refer to AN6008: *Getting started with debug authentication (DA) for STM32 MCUs*.

Figure 27. Debug authentication - OBkey generation

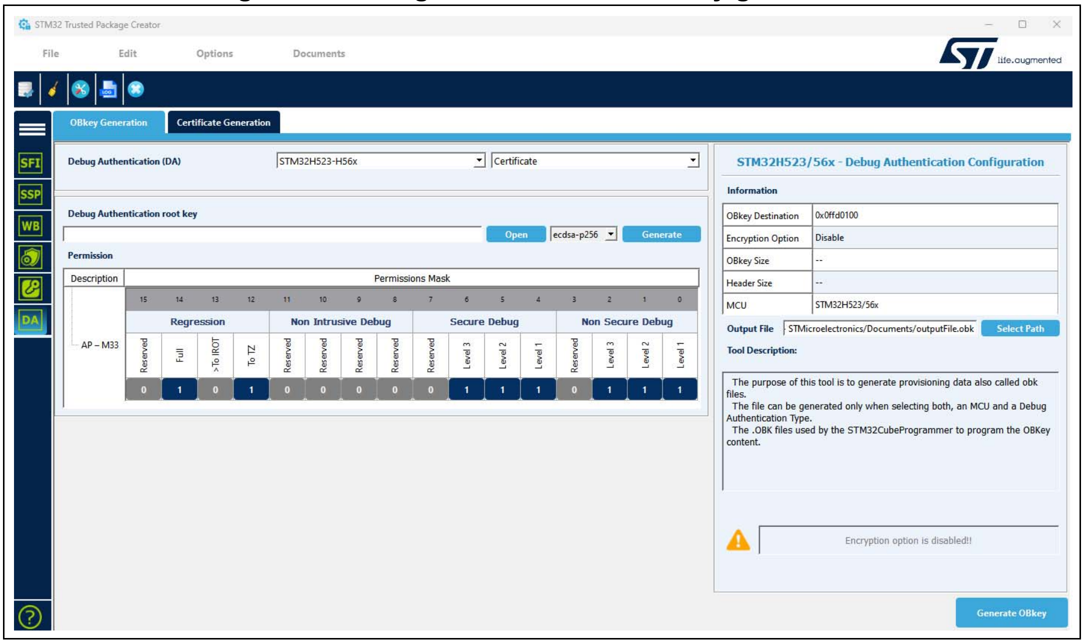
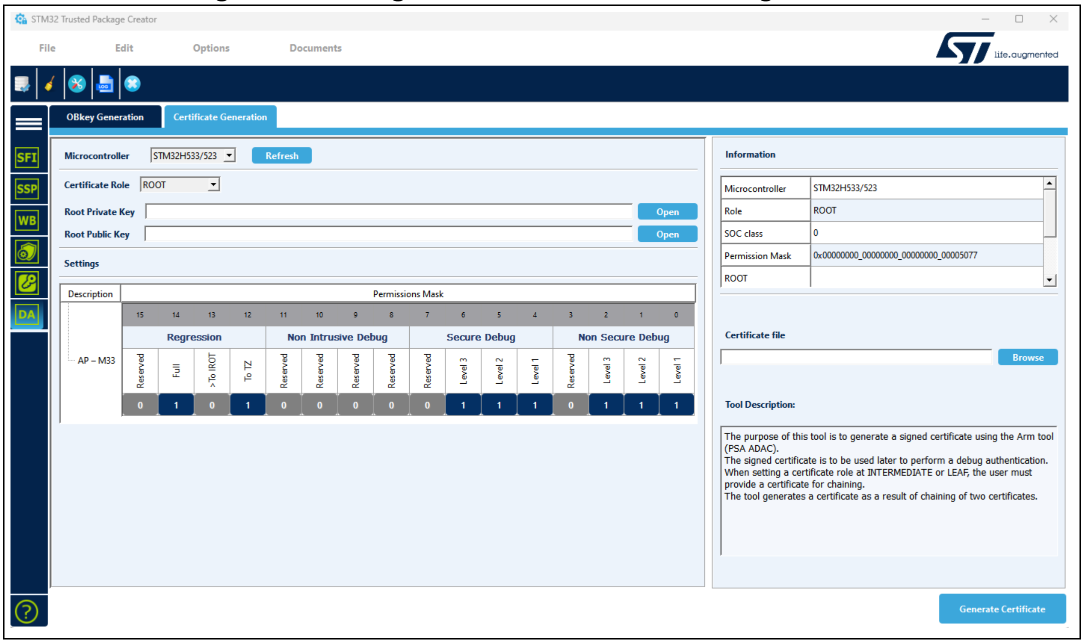


Figure 28. Debug authentication - Certificate generation



- Microcontroller: specifies the MCU used.
- Role: defines where this certificate may be used.
  - Possible values: 1 = ROLE ROOT, 2 = ROLE INTERMEDIATE, 3 = ROLE LEAF
- Private key: relative path to issue a key file. It is also the key that signs this certificate. For the root certificate, must be the same as the subject key.
- Public key: relative path to subject a key file. It is also the public key embedded within this certificate.

- Permissions mask: written on128-bit, it is the bit mask limiting the permissions that can be requested when performing the debug authentication. A set bit indicates that the permission corresponding to that bit position is allowed.

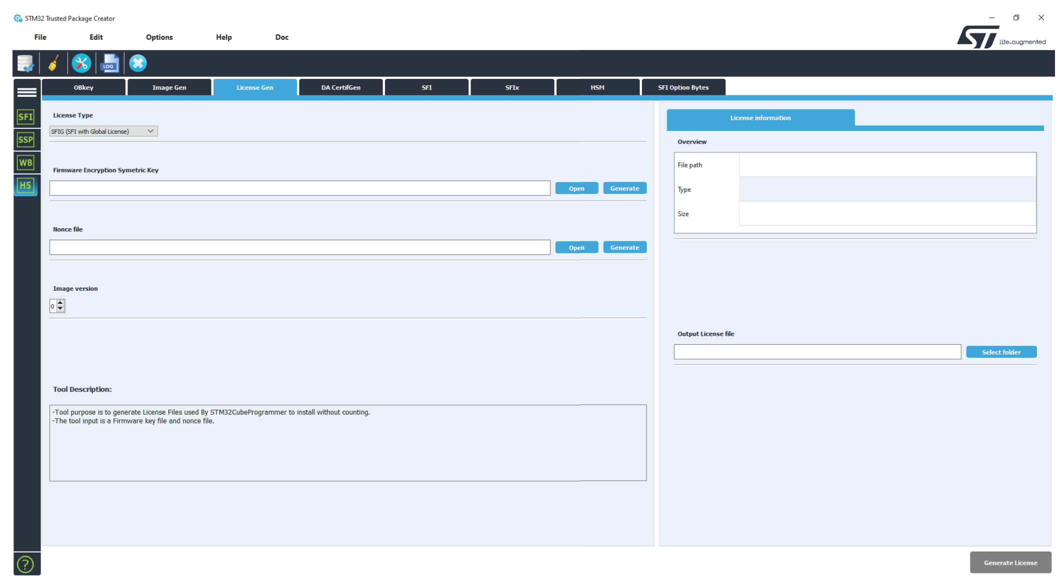
When setting a certificate role at INTERMEDIATE or LEAF, the user must provide a certificate for chaining. The tool generates a certificate as a result of chaining of two certificates.

4.4.4 Global license generation

The STM32 Trusted Package Creator tool presents a license generation tab to generate license files used by the STM32CubeProgrammer during the secure manager and module install.

The tool inputs are a firmware key file and a nonce file.

Figure 29. STM32TrustedPackageCreator license generation tab



DT72089V1

Firmware key and nonce files

This key is 32 bytes and is composed of two fields:

- The initialization vector (IV) or nonce, provided by the user in the nonce file field
- The key (firmware encryption symmetric key field)

This key is used for firmware or module encryption in the AES128-GCM format.

Both fields are 16-byte long. The last 4 bytes of the nonce must be all zeros (only 96 bits of the nonce are used in the AES128-GCM algorithm).

Both fields must remain secret; Therefore they are encrypted before being sent to the chip.

The key and the nonce remain the same for all licenses for a given piece of firmware. However, they must be different for different firmware or different versions of the same firmware.

License type

This option must be set either to SFIG for secure manager install or to SMUG for secure module install/update.

License version

This option is set to 0.

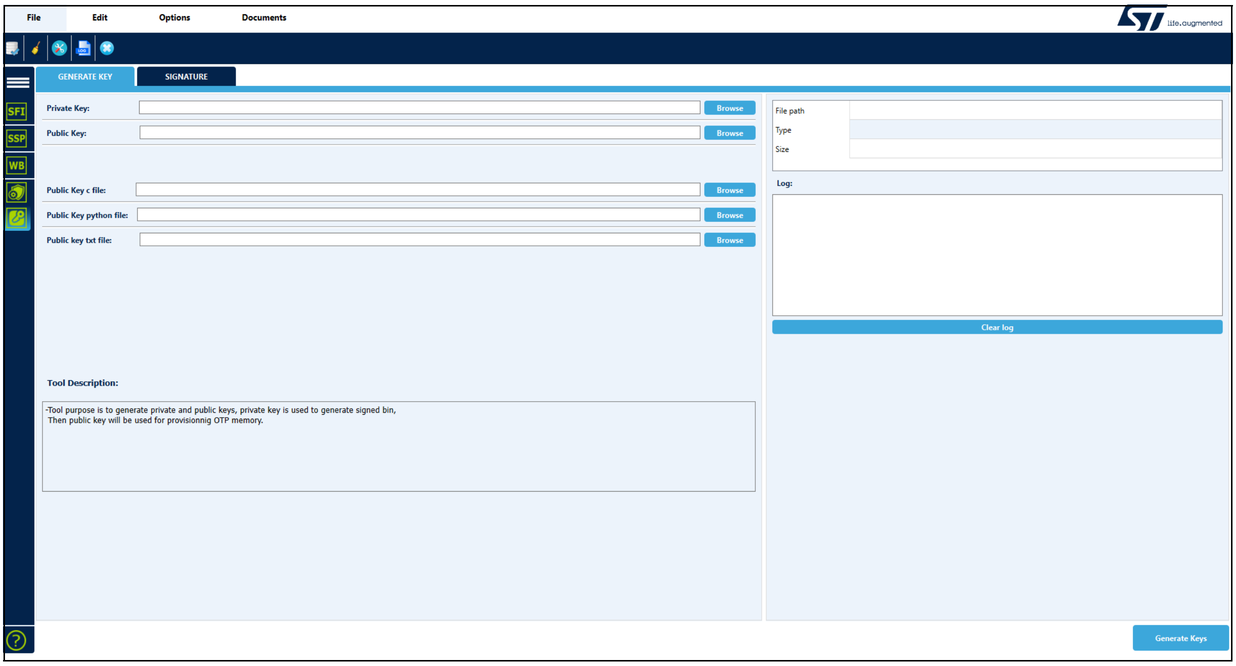
4.5 STM32WB0x/STM32WL3x secure boot

4.5.1 Generate key

The Key generation tab allows:

- Generating a new key pair (public and private)
- Selecting the destination folder for the generated key pair

Figure 30. STM32TrustedPackageCreator Generate key GUI

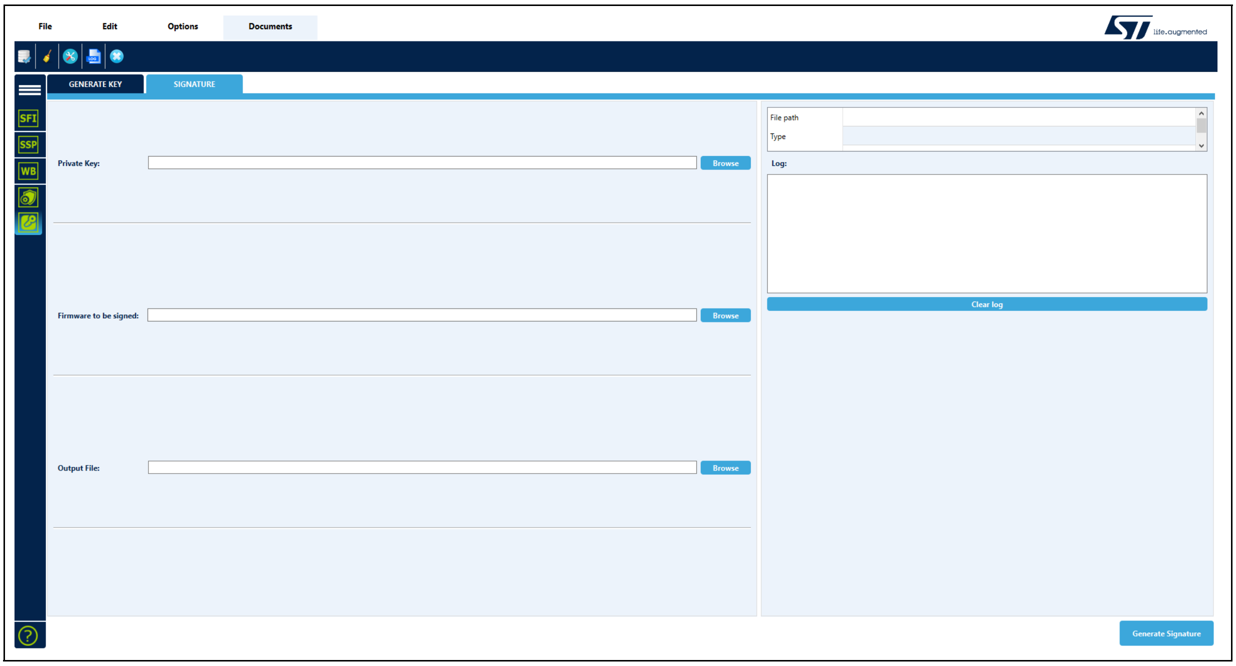


4.5.2 Firmware signature

The Firmware signature tab allows:

- Setting the key pair (public and private) to create the file signature
- Selecting the file to be signed
- Setting the name of the signed file
- Creating the signed image

Figure 31. STM32TrustedPackageCreator Firmware signature GUI

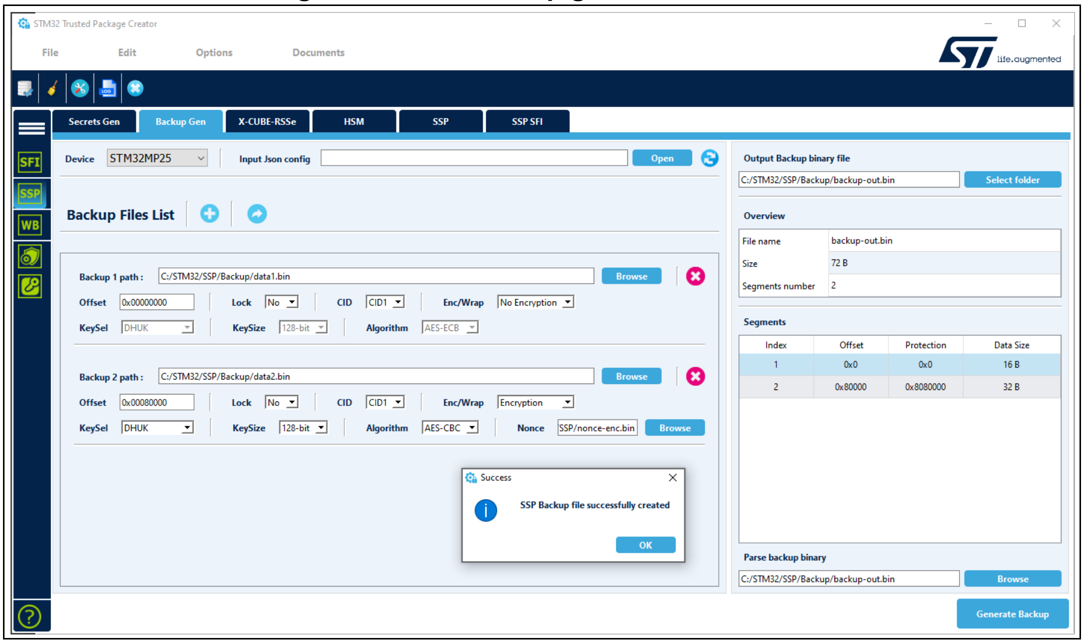


4.6 SSP backup generation

The **Backup Gen** tab, located within the SSP panel, is designed to achieve four main goals.

- Parse Json file.
- Customize the backup files list.
- Generate a backup binary file.
- Browse and decode a backup binary.

Figure 32. SSP Backup generation window



Parse Json file

Once you have selected the STM32 device, you can choose and open a JSON file by pressing the "Open" button located under the "Input JSON config" option. The tool will then launch a file chooser, allowing you to select the desired JSON file. Once the file has been selected, the tool will parse its contents and display them on the "Backup Files List" component.

The list of input files and its configuration are described on a JSON file:

Figure 33. SSP Backup input JSON file

```
{
    "Files": [
        {
            "Name": "Backup 1",
            "Offset": "0x00000000",
            "Path": "C:/STM32/SSP/Backup/data1.bin",
            "Protection": {
                "Lock": {
                    "BitOffset": "0x1F",
                    "BitValue": "0x01",
                    "BitWidth": "0x01",
                    "PossibleValues": {
                        "0": "No",
                        "1": "Yes"
                    }
                },
                "Wrapping": {
                    "BitOffset": "0x1D",
                    "BitValue": "0x00",
                    "BitWidth": "0x02",
                    "PossibleValues": {
                        "0": "No Encryption",
                        "1": "HUK derived for TDCID1",
                        "2": "HUK derived for TDCID2"
                    }
                }
            }
        }
    ],
    "Title": "STM32 Backup Config"
}
```

- **Name**: The identifier name of the file
- **Offset**: Offset within the memory area
- **Path**: The path of the considered binary file
- **Protection**: The security word is composed of the bitfields mentioned in the specification.
  - **Lock**
    BitOffset: 0x1F
    BitValue: 0x00, the default value of the field
    BitWidth: 0x01
    PossibleValues: the list of possible values
  - **EncWrap**
    BitOffset: 0x1D
    BitValue: 0x00, the default value of the field
    BitWidth: 0x02
    PossibleValues: the list of possible values

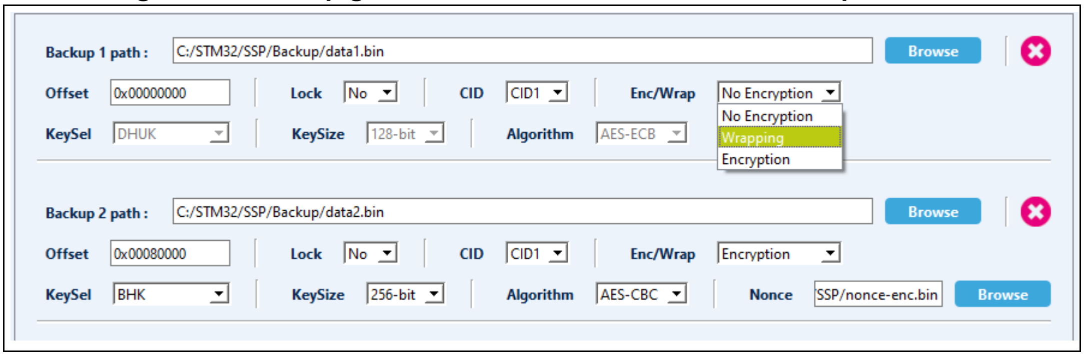
Same definition for: CID, KeySel, KeySize, Algorithm, nonce

The nonce field is available only with the algorithms AES-CB and AES-CTR.

Customize the backup files list:

– The possible actions that can be executed:
– Add new backup file: new backup section will be added to the view then you need to populate its fields (+ icon).
– Remove item: it is possible to delete a specific item (X icon).
– Mention the Offset value, hexadecimal value.
– Configure the item: Lock, EncWrap, KeySel, KeySize, Algorithm
– Browse the file path to edit the input binary location.
– Export the current view to an output Json file.

Figure 34. Backup generation, customize the list of backup sections



If the backup files list is empty, it is not possible to export a JSON file.

If you have been adding and removing items from the list view using the imported JSON file, you can press the "Refresh" icon to clear and reload the corresponding fields.

If you add a large number of backup items, a scroll bar will appear, allowing you to navigate through the items more easily.

You can create a backup view from scratch without needing to specify previously an input Json file, by adding directly backup items using the “+” icon.

Generate a backup binary file

If all necessary elements are present, you can proceed by pressing the "Generate Backup" button to initiate the preparation of the image. The resulting image will be saved into a binary file, which is specified in the "Output Backup binary file" field.

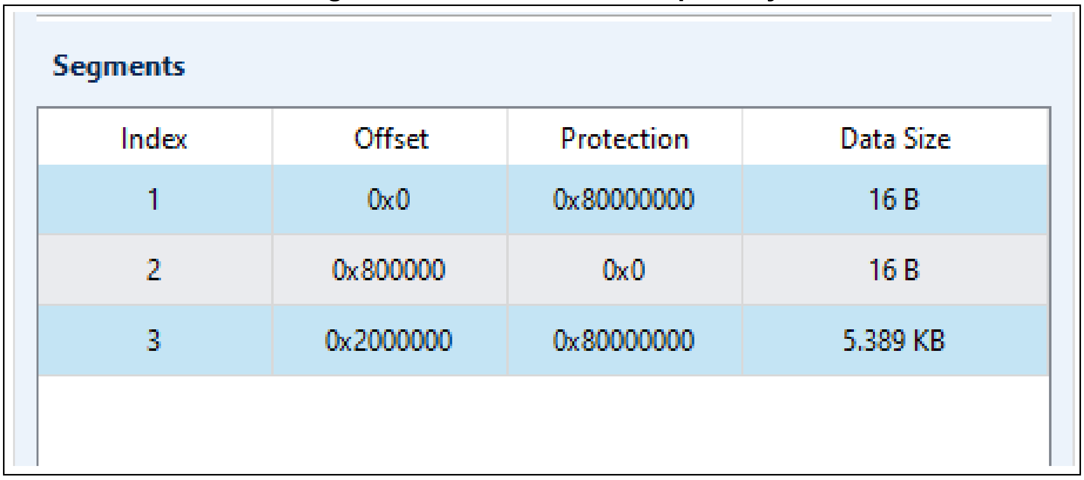
The result information is displayed in the Overview table:

– File name: the output binary name.
– Size: the total size of the output image
– Segments number: the total number of backup files used to prepare the image.

The list of sections is decoded and shown in the Segments table:

– Index: The index of the backup area
– Offset: The offset value.
– Protection: The protection value.
– Data size: The size of the corresponding backup area data.

Figure 35. Decode SSP backup binary



Segments

Index	Offset	Protection	Data Size
1	0x0	0x80000000	16 B
2	0x800000	0x0	16 B
3	0x2000000	0x80000000	5.389 KB

Browse backup binary

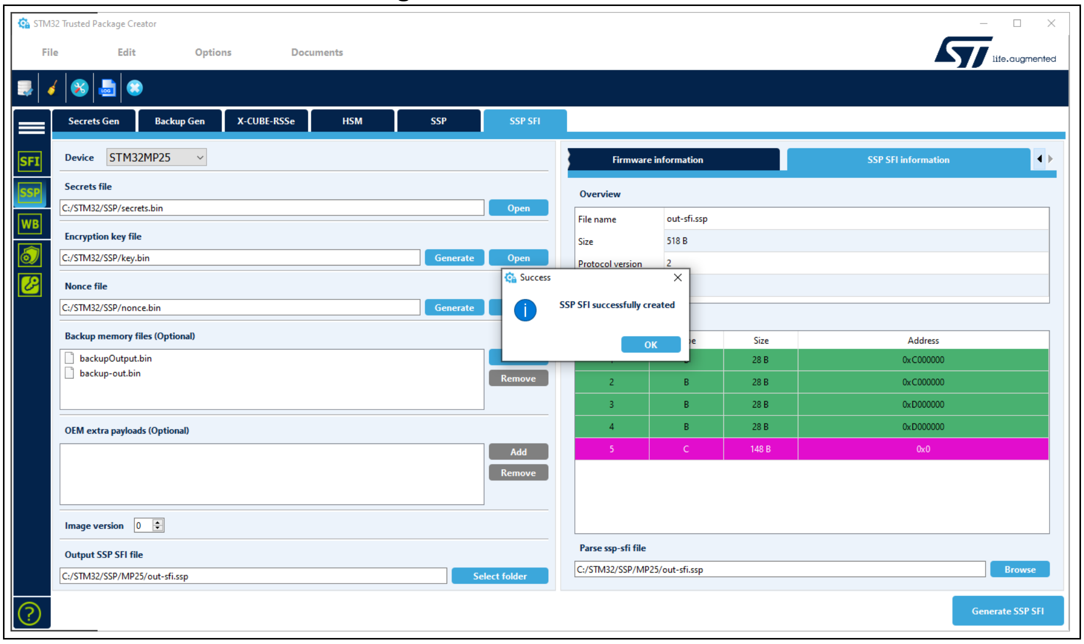
If you already have a backup binary and want to decode and check its contents, you can do so by clicking on the "Browse" button located under the "Parse backup binary" option.

This will launch a file explorer, allowing you to select the desired file. The tool will then parse the file, check its format validity, and display the sections in the "Segments" table.

4.7 SSP-SFI image generation

The STM32 Trusted Package Creator tool GUI presents an SSP-SFI tab to generate an SSP image in SFI format. The user must fill in the input fields with valid values.

Figure 36. SSP SFI window



Device

The user can use the combo box to select the current device to be used.

Secrets file

Binary file including the secrets to encrypt. Can be selected by entering its path (absolute or relative), or by selection with the open button.

Encryption key and nonce files

The encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selection with the open button.

Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).

Backup memory file

It is possible to integrate multiple backup files into an SSP-SFI image by adding the backup input file and indicating the RAM base address. Each backup segment is represented in an independent area of type 'B'.

OEM extra payload

It is reserved for future features.

Image version

Image version value in [0..255], It is a mark to generate several image with different version.

Output SSP SFI file

Select the output directory by mentioning the SSP-SFI file name to be created with an .ssp extension. It can be selected by entering paths (absolute or relative), or with the select folder button. Note that in the latter case, a name is suggested, which can be kept or changed.

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

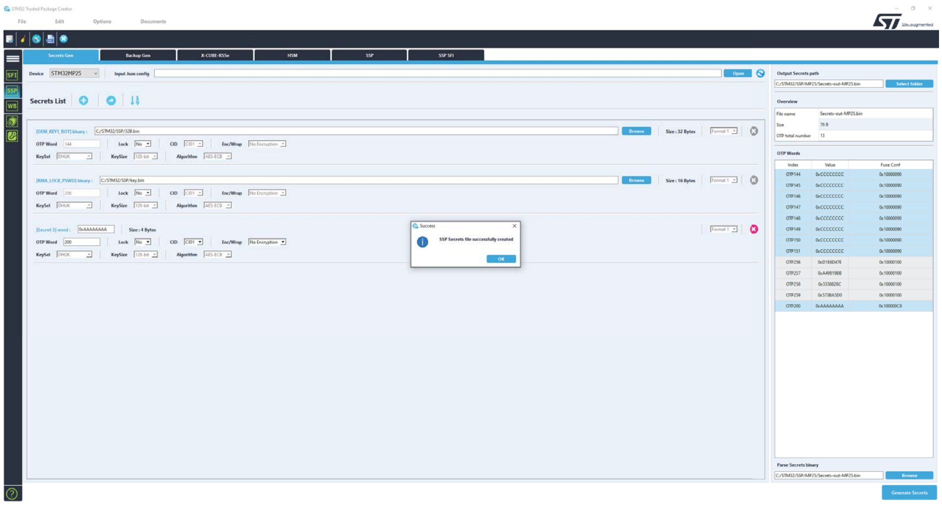
Once the generation is done, we can get SSP information from the SSP overview section.

4.8 SSP secrets generation

The Secrets Gen tab, located within the SSP panel, is designed to achieve 4 main goals.

- Customize the Secrets items list.
- Parse Secrets Json file.
- Generate a Secrets binary file.
- Browse and decode a Secrets binary.

Figure 37. SSP Secrets generation window



MSv75078

4.8.1 Customize the Secrets items list

The possibles actions that can be executed:

- Add new item: new input section will be added to the view then you need to populate its fields (+ icon). Two different possible suggestions: add binary or word.
- Remove item: it is possible to delete a specific item (X icon). Mandatory items are not removed to ensure the presence of the necessary items needed for secure boot.
- Mention the Word value, decimal value to indicate the index of the OTP (not editable for predefined items)
- Choose the adequate values for the Fuse Configuration (Lock, CID, Enc/Wrap, KeySel, KeySize, Algorithm and Nonce) for the device that supports this capability (as STM32MP25).
- Select the endianness format of the binary, Format 1, or Format 2. (Format 1 is fixed for word item)
- Browse the file path to edit the input binary location or mention the word item value.
- Export the current view to an output Json file.
- Sort the list of items based on the OTP word index to give an ordered view.

4.8.2 Parse Secrets JSON file

Once a customized view is populated, it is possible to export the current view into a JSON file to be used as reference or for future analysis.

you can choose and open a JSON file by pressing the "Open" button located under the "Input JSON config" option. The tool will then launch a file chooser, allowing you to select the desired JSON file. Once the file has been selected, the tool will decode its contents and display them on the "Secrets List" component.

4.8.3 Generate a Secrets binary file

If all necessary elements are present, you can proceed by pressing the "Generate Secrets" button to initiate the preparation of the image. The tool processes several checks on the input items (check of binary size, available index, memory overflow, regions overlap). The resulting image is saved into a binary file, which is specified in the "Output Backup binary file" field. The result information is displayed in the Overview table:

- File name: the output binary name.
- Size: the total size of the output image.
- OTP total number: the total number of used OTP words.

The list of areas is decoded and shown word by word in a table:

- Index: the index of the OTP word.
- Value: the OTP value.
- Fuse Conf: the protection value.

Each OTP region is highlighted with the same color to differentiate between areas (binary or word).

4.8.4 Browse Secrets binary

If you already have a Secrets binary and want to decode and check its contents, you can do so by clicking on the "Browse" button located under the "Parse Secrets binary" option. This launches a file explorer, allowing you to select the desired file. The tool then parses the file, checks its format validity, and displays the sections in the "OTP Words" table.

Note: Endianness given in the specification for format: Keys are represented as a string of bytes to be stored in consecutive OTP words. For example, a 64-bit key (0xAABBCCDDEEFF5566) is stored in two consecutive OTP words, KEY0 and KEY1. A key is stored in OTP words using one of the following formats:

- Format 1: KEY0 = 0xAABBCCDD, KEY1 = 0xEEFF5566
- Format 2: KEY0 = 0xDDCCBBAA, KEY1 = 0x6655FFEE

4.9 X-CUBE-RSSe and STM32MPUSSP-UTIL

4.9.1 X-CUBE-RSSe

The X-CUBE-RSSe comes in zip format and can be directly imported into the STM32TrustedPackageCreator. This package contains:

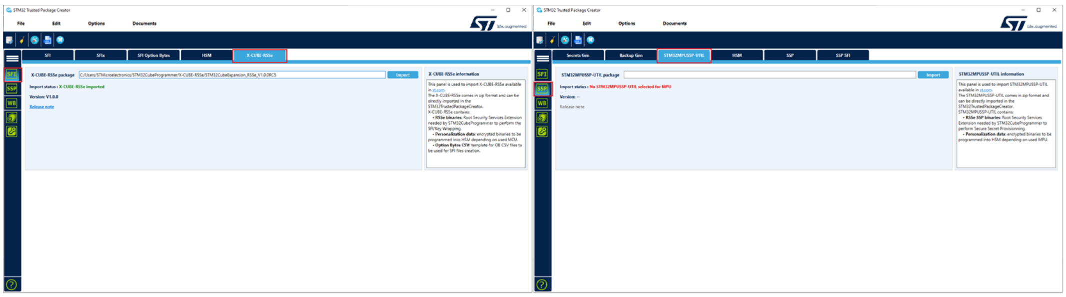
- RSSe binaries: Root Security Services Extension needed by STM32CubeProgrammer to perform the SFI/Key Wrapping.
- Personalization data: encrypted binaries to be programmed into HSM depending on the used MCU.
- Option bytes CSV: template for OB CSV files to be used for SFI files creation.

The user can import the package from the “X-CUBE-RSSe” window, which is available in both the Secure Firmware Install and Security windows.

This window contains:

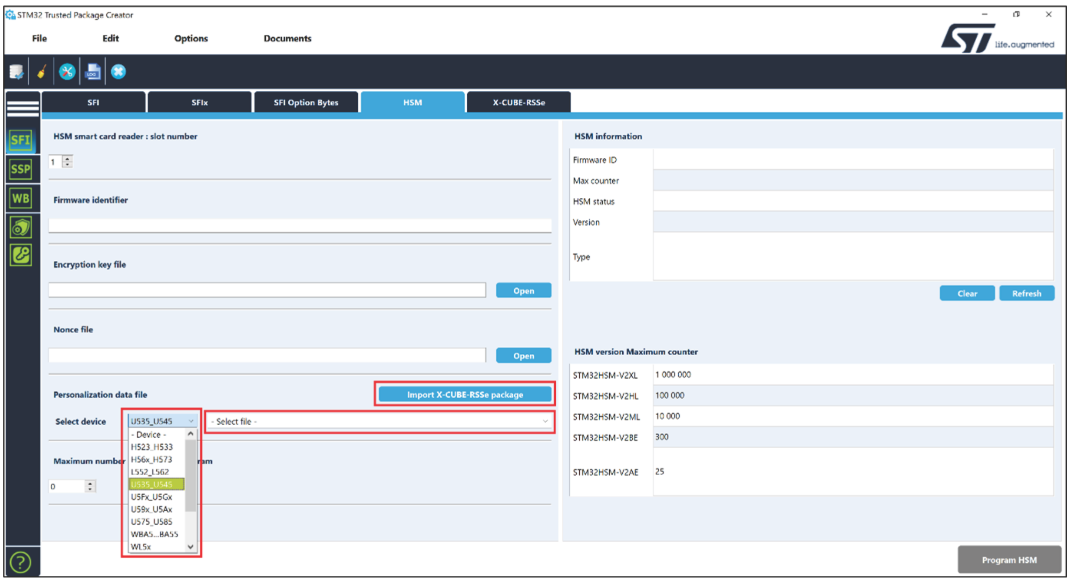
- Description of this package: provides detailed information about the package.
- Imported package path: automatically populated after importing the package.
- Import status: displays either "X-CUBE-RSSe Imported" or "No Selected X-CUBE-RSSe for MCU."
- Package version: automatically filled after importing the package.
- Release note link: the hyperlink will be enabled after importing the package.

Figure 38. The X-CUBE-RSSe interface in the Security and SFI windows



MSv75079

Figure 39. The X-CUBE-RSSe Package in HSM panel



MSv75080

4.9.2 STM32MPUSSP-UTIL

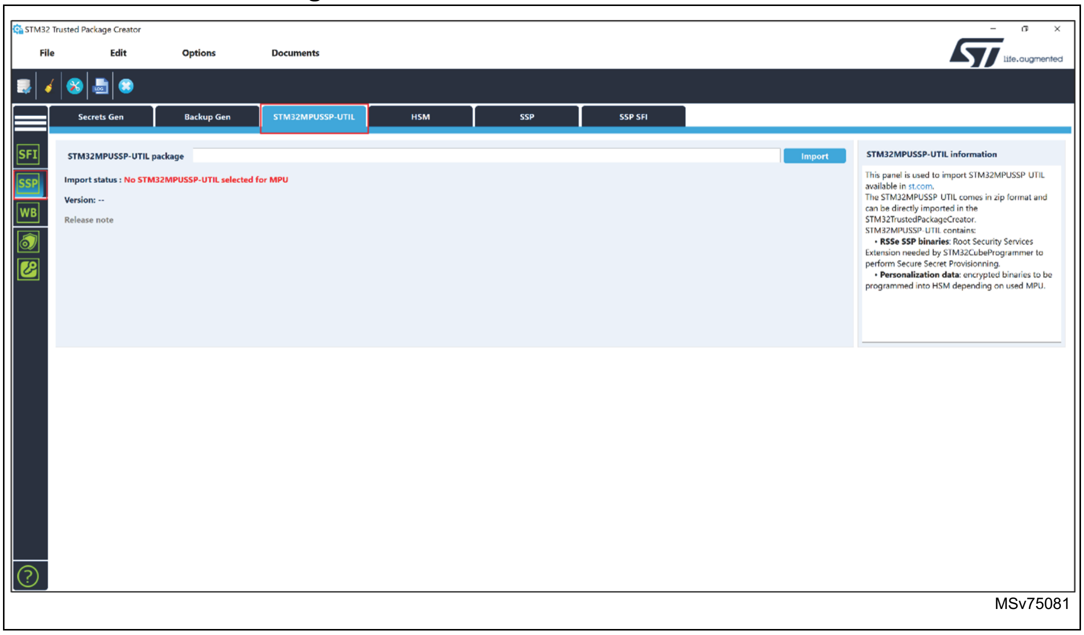
The STM32MPUSSP-UTIL comes in zip format and can be directly imported into the STM32TrustedPackageCreator. This package contains:

- RSSe SSP binaries: Root Security Services Extension needed by STM32CubeProgrammer to perform Secure Secret Provisioning.
- Personalization data: encrypted binaries to be programmed into HSM depending on the used MPU.

The user can import the package from the "STM32MPUSSP-UTIL" window, available within the "SSP" window. This window contains:

- Description of this package: provides detailed information about the package.
- Imported package path: automatically populated after importing the package.
- Import status: displays either "STM32MPUSSP-UTIL imported" or "No Selected STM32MPUSSP-UTIL for MPU."
- Package version: automatically filled after importing the package.
- Release note link: the hyperlink will be enabled after importing the package.

Figure 40. STM32MPUSSP-UTIL interface



5 Option bytes file

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

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

Figure 41. Example of an option bytes file

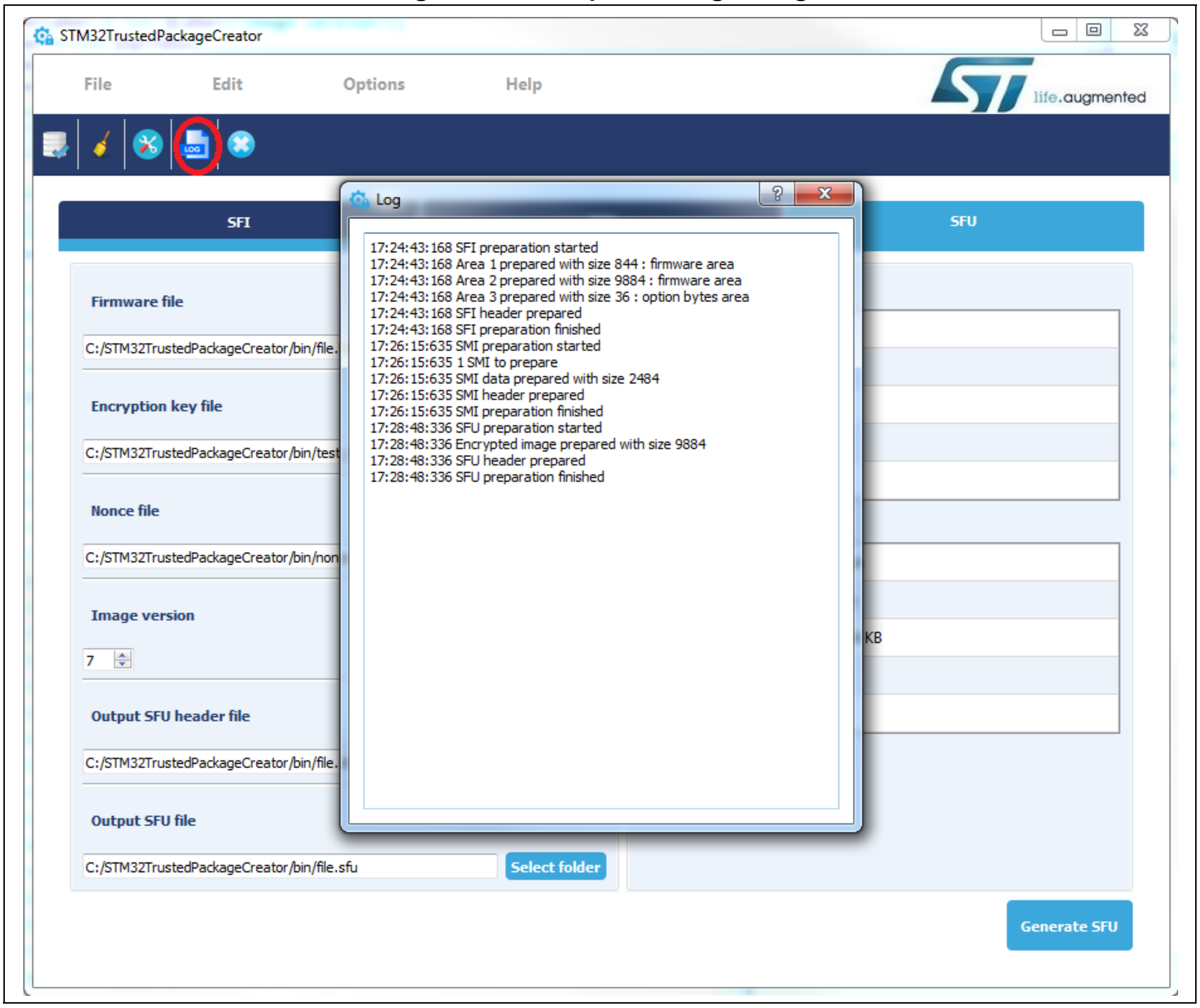
FIR_ob.csv

```
FOPTSR_PRG,0x13FE00F8
FPRAR_PRG_A,0x88060800
FPRAR_PRG_B,0x80000FFF
FSCAR_PRG_A,0x80000FFF
FSCAR_PRG_B,0x80000FFF
FWPSN_PRG_A,0x000000FF
FWPSN_PRG_B,0x000000FF
FBOOT7_PRG,0x1FF10800
RFU,0x10000810
```

6 Log dialog

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

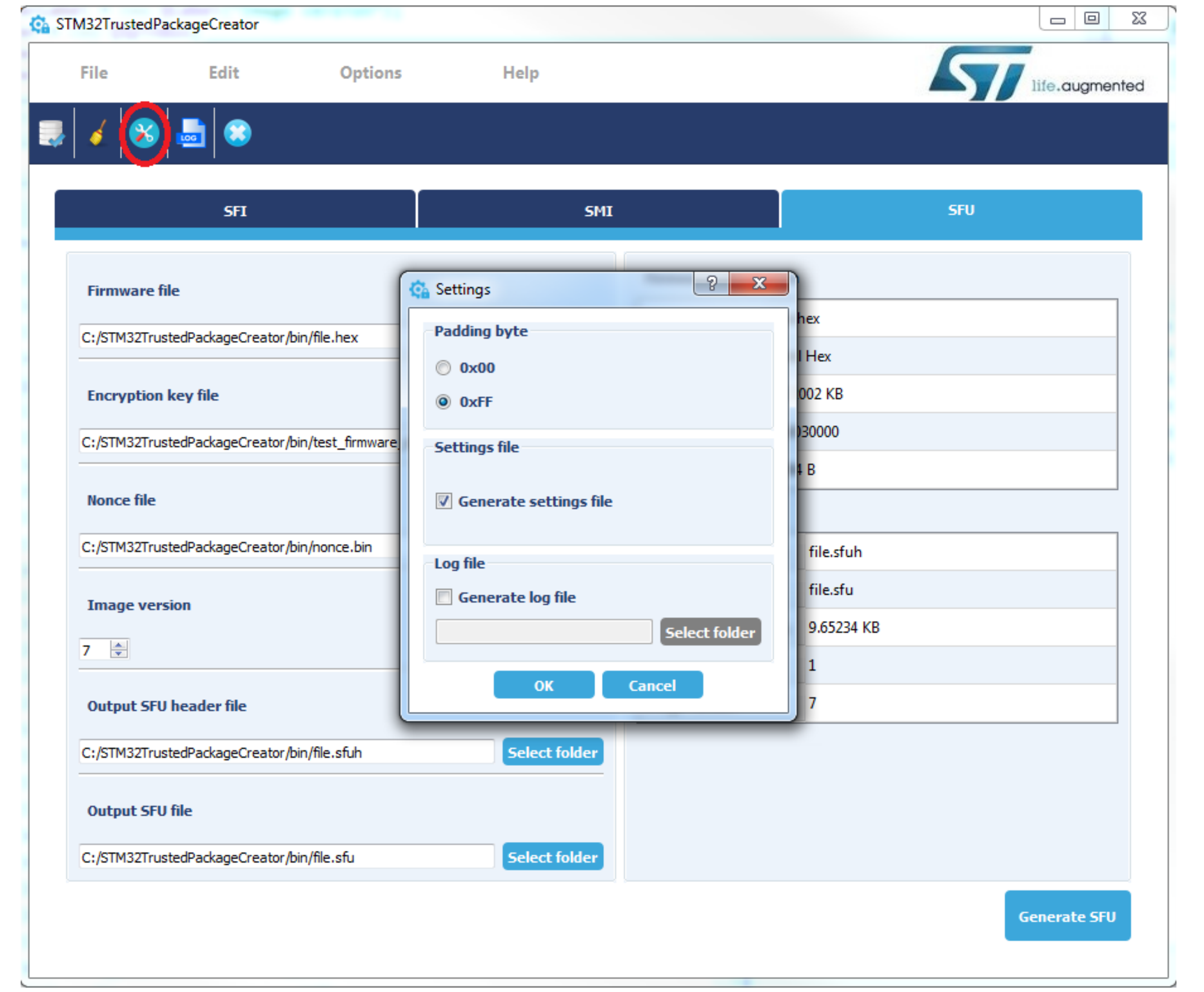
Figure 42. Example of a log dialog



7 Settings

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

Figure 43. Settings dialog



Padding byte

When parsing files, padding may be added to fill the gap between segments separated by 16 bytes or less, in order to merge them and reduce the number of segments. The user may have the choice between 0xFF (default value) or 0x00.

Settings file

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

Log file

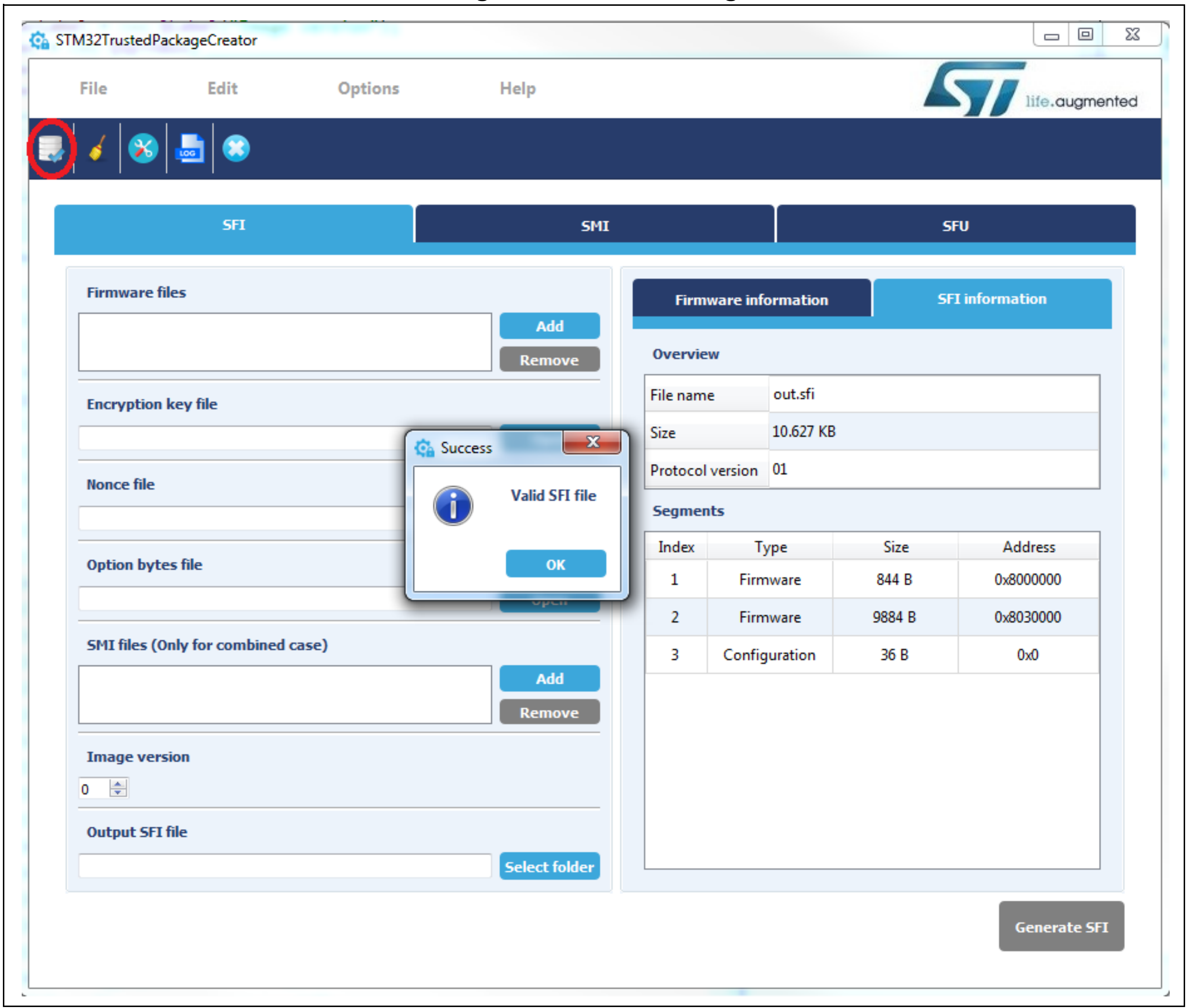
When checked, a log file is generated in the selected path.

8 SFI checking

SFI checking can be accessed by clicking the check SFI button in the tools bar or in the menu bar: File → Check SFI.

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

Figure 44. SFI checking

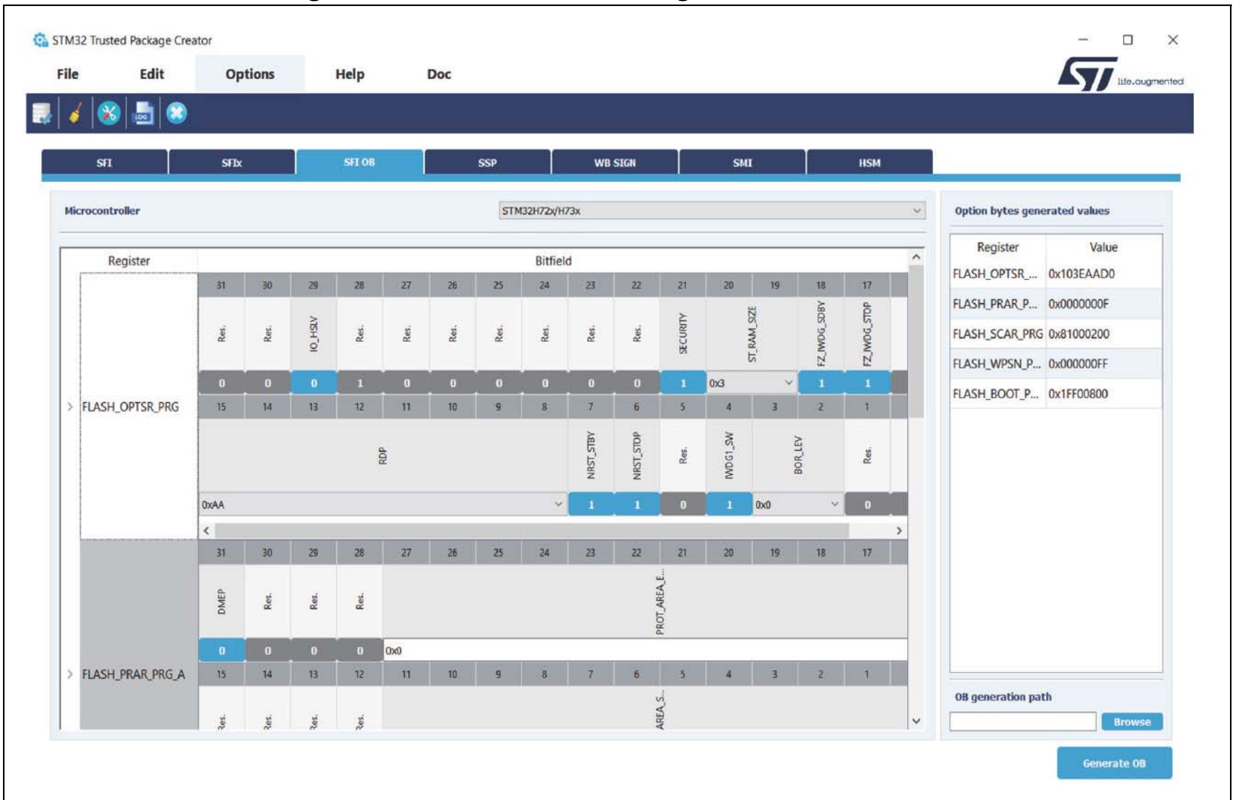


9 SFI OB generation

The STM32 trusted package creator tool GUI presents an SFI OB tab to generate an option bytes CSV file with custom option bytes value. To generate SFI CSV option bytes file:

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

Figure 45. STM32TrustedPackageCreator SFI OB GUI



Appendix A XML configuration for OBKey generation

A.1 Main tool purpose

The tool purpose is to generate the .OBK files used by the STM32CubeProgrammer to program the OBKey content. The tool input is an XML file describing the OBKey region structure.

A.2 OBKey file structure

The file with an OBK extension has two main areas:

- **Header**: this area is used by the STM32CubeProgrammer to identify the targeted OBKey address, payload size, and if an OBKey encryption has to be requested to RSS.
- **Payload**: this area is the content to be programmed in the OBKey editor. The payload contains the data described by the XML input. It is edited and provided by the user.

A.2.1 Header structure

The header is not programmed by the STM32CubeProgrammer into the OBKey. It helps exchange information between the STM32CubeProgrammer and the STM32TrustedPackageCreator.

The header is created by the STM32TrustedPackageCreator while generating the .OBK file. It is read and removed by the STM32CubeProgrammer while programming the .OBK file.

The header is coded in little endian and is composed of:

```
....uint32_t.destAddress;
....uint32_t.OBKeySize;
....uint32_t.doEncryption;
```

Table 1. Header area

Type	Size (bytes)	Usage
destAddress	4	OBKey physical destination address
OBKeySize	4	OBKey payload size, payload starts immediately after the header.
doEncryption	4	Ask the RSS to encrypt the OBKey payload or not. The possible values in the header are: – **0** that means no encryption requested to RSS, translated to ***0xCACA0AA0U*** for RSS. – **1** that means encryption requested to RSS, translated to ***0xF5F5A0AAU*** for RSS.

A.2.2 Payload

The payload starts immediately after the header and its size must be indicated in the header (**OBKeySize**). The payload is loaded into the memory without any modification.

A.3 Input XML file structure

The input XML can have one of the following tags:

A.3.1 Obdata tag

The <Obdata> tag is the global container of all the other tags. It specifies that an Obdata XML file is being parsed.

A.3.2 Info tag

The <Info> tag specifies the valid information for all the XML files.

The typical <Info> tag structure is:

```
<Info>
   <Title>iRoT_Configuration</Title>
   <ObDestAddress>0x1FF00000</ObDestAddress>
   <DoEncryption>1</DoEncryption>
   <GlobalAlign>16</GlobalAlign>
   <FlashStart>0x08000000</FlashStart>
   <FlashSize>0x200000</FlashSize>
   <FlashSectorSize>0x2000</FlashSectorSize>
   <OBKeyStart>0x1FF00000</OBKeyStart>
   <OBKeySize>0x20000</OBKeySize>
   <OBKeySectorSize>0x10</OBKeySectorSize>
   <AuthOBKeyStart>0x1Ff10000</AuthOBKeyStart>
   <AuthOBKeySize>0x10000</AuthOBKeySize>
</Info>
```

The <Info> tag is mandatory in the XML OBKey description.

- **<Title>** is an informative tag. The title is displayed inside the tool.
- **<ObDestAddress> [optional]** specifies the OBKey destination address.
- **<DoEncryption> [optional]** can be set at 1 or 0. It specifies if the user asks RSS to encrypt or not the OBKey during the installation process**.**
  - 1: Encryption is requested.
  - 0: No encryption is requested.
- **<GlobalAlign>** specifies the alignment of the output binary OBKey file. Based on this value, the tool aligns the file size. The value unit is expressed in bytes.
- **<FlashStart> [optional]** contains the MCU flash memory address. It is typically 0x08000000 on STMicroelectronics MCU.
- **<FlashSize> [optional]** contains the MCU flash memory size. The unit is expressed in bytes. FlashSize is used to check the download slot location.
- **<FlashSectorSize> [optional]** contains the MCU flash memory sector size. It is used to check that the download slots are aligned with the sectors.
- **<OBKeyStart> [optional]** contains the MCU OBKey address.
- **<OBKeySize> [optional]** contains the MCU OBKey size. The unit is expressed in bytes.
- **<OBKeySectorSize> [optional]** contains the MCU OBKey sector size. It is used to check the sector alignments of downloaded OBKey slots.
- **<AuthOBKeyStart> [optional]** contains the authorized OBKey start address. It must be located inside the OBKey memory area.
- **<AuthOBKeySize> [optional]** contains the authorized OBKey size.

*Note: The generated output format is .bin instead of .OBK when neither **ObDestAddress,** nor **DoEncryption** is specified.*

If either **ObDestAddress** or **DoEncryption** is present, the configuration is wrong.
When both **ObDestAddress** and **DoEncryption** are specified, outputs with an.OBK extension are generated by default.
In case of an output file generated with the .bin extension, there is no header (12 bytes) added to the .OBK file.
All tags following the <Info> tag result in data concatenated in the output binary.

The order of the tags determines the order in the output binary.

A.3.3 Hash tag

The <Hash> tag is used to indicate to the tool that a hash on the binary file .OBK must be calculated and embedded in the output binary file .OBK. The hash is calculated on all payload data. The final payload is the input data added to the hash. The hash algorithm used is ***SHA256***.

Typical hash tag structure:

```
<Hash></Hash>
```

The <Hash> tag is optional. When used, no specific data are requested. The only presence of the hash tag triggers the generation of the hash.

A.3.4 Data tag

The <Data> tag is a generic tag. Its main role is to host raw data with the size varying from 1 to 16 bytes.
A typical presentation is:

```
<Data>
   <Name>data1</Name>
   <Value>0xFFFFFFF1</Value>
   <Width>4</Width>
   <Hidden>1</Hidden>
   <Default>0xFFFFFFFF</Default>
   <Dependency>Image,0x02</Dependency>
   <Tooltip>Data1 is for storing slot 1 address</Tooltip>
</Data>
```

- **<Name> [mandatory]:** The name subtag hosts the <Data> tag name. This name is displayed by the tool and serves as a unique identifier. It is not permitted to have multiple tags with the same name. This includes Data, Bool, List, and File that are key tags.
- **<Value> [optional]:** The value tag hosts the data entered by the user. They can be decimal or hexadecimal values aligned with the width.
- **<Width> [mandatory]:** The width tag gives the maximum length of the data. It is expressed in bytes.
- **<Default> [mandatory]:** It hosts the default value of the data. The tool displays this value by default. If the value tag exists, the tool displays the data available in the value tag. The values can be hexadecimal or decimal and are aligned with the width.
- **<Hidden> [optional]:** If hidden = 0, the parameter is displayed in GUI. If hidden = 1, the parameter is hidden but is included in the command line.
- **<Reserved> [optional]:** It indicates that this data area is reserved for a future use. Reserved data are not displayed.
- **< dependency > [optional]:** It indicates that these data depend on another tag (List or Bool) marked also with the using tag with the first key word into this tag. The second word is a value that specifies the data field. It is displayed if this value equals the selected value dependency source (List or Bool). If it is not the case, the data are grayed.
- **<Tooltip> [optional]:** It is an informative tag. This information is displayed as a tool tip when this tag is hovered over with the mouse.

A.3.5 Data tag used as slot

It is possible to indicate to the tool that a data tag is a slot. This is useful to make some checks on slot definition.

Example of a data tag defined as a slot:

```
<Data>
   <Name>DownloadModuleSlotAddress</Name>
   <Value>0x08010000</Value>
   <Width>4</Width>
   <Default>0x08010000</Default>
```

```
    <Tooltip>download slot of application module install</Tooltip>
    <Slot>start#1<Slot>
</Data>
<Data>
    <Name>DownloadModuleSlotSize</Name>
    <Value>0x4000</Value>
    <Width>4</Width>
    <Default>0x4000</Default>
    <Tooltip>download slot of application module install size</Tooltip>
                        <Slot>size#1<Slot>
</Data>
```

The <Slot> tag is used to indicate the slot definition. The content of the tag must include the information on whether it is a tag size, or a tag start, and also the identification of the slot located after the #.

For memory constraints, a unique <Data> tag can hold the size of many slots. In this case, many slot sizes are concatenated. In this case, the tool knows that there is a size for slot 1, 2, and 3.

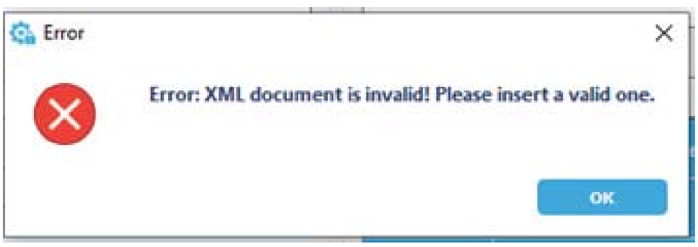
Example:

```
<Data>
    <Name>DownloadModuleSlotSize</Name>
    <Value>0x4000</Value>
    <Width>4</Width>
    <Default>0x4000</Default>
    <Tooltip>download slot of application module install size</Tooltip>
    <Slot>size#1<Slot>
    <Slot>size#2<Slot>
    <Slot>size#3<Slot>
</Data>
```

Checks done by the tool on the data and info tags

- Only hexadecimal or decimal values are allowed (for all data “default/value” values).
- Duplicated names are not authorized (for all data names).
- All required tags must be present in the right order.
- Any missed or wrong tag causes an error message. It leads to an invalid XML file.

Figure 46. Error message



DT71648

Below an XML order example.

```
<Info>
<Title>STiRot configuration</Title>
<FlashSize>0x200000</FlashSize>
<ObDestAddress>0x0FFD0200</ObDestAddress>
<DoEncryption>1</DoEncryption>
<GlobalAlign>16</GlobalAlign>
<FlashStart>0x0C000000</FlashStart>
<FlashSectorSize>0x2000</FlashSectorSize>
<OBKeyStart>0x1FF00000</OBKeyStart>
<OBKeySize>0x20000</OBKeySize>
<OBKeySectorSize>0x10</OBKeySectorSize>
<AuthOBKeyStart>0x1Ff00900</AuthOBKeyStart>
<AuthOBKeySize>0x300</AuthOBKeySize>
</Info>
<Hash></Hash>
<DA></DA>
<Text>
<Name>...</Name>
<Value>...</Value>
<MinWidth>..</MinWidth>
<Width>..</Width>
<Default>...</Default>
<Tooltip>...</Tooltip>
</Text>
<Bool>
<Name>...</Name>
<Value>...</Value>
<Width>..</Width>
<Default>...</Default>
<True>...</True>
<False>...</False>
<Tooltip>...</Tooltip>
</Bool>
<List>
<Name>number of images managed</Name>
<Value>0x01</Value>
<Width>1</Width>
<Default>0x02</Default>
<Val>0x01,code image only</Val>
<Val>0x02,code and data images</Val>
<Dependency>Image</Dependency>
<Tooltip>...</Tooltip>
</List>
<Data>
```

```
<Name>OBKey1 area offset</Name>
<Value>0x00000920</Value>
<Width>4</Width>
<Hidden>1</Hidden>
<Default>0x00000000</Default>
<OBKeySlot>start#1</OBKeySlot>
<Dependency>image,0x01</Dependency>
<Tooltip>...</Tooltip>
</Data>
<File>
<Name>authentication key</Name>
<Value>C:/Users/../../keys_OBKey/pubkey.pem</Value>
<Tooltip>...</Tooltip>
<KeyType>ecdsa-p256</KeyType>
<Align>3</Align>
<Type>public</Type>
<Default>...</Default>
</File>
<Permission>
<Name>...</Name>
<Value>...</Value>
<Width>..</Width>
<Default>...</Default>
<Tooltip>...</Tooltip>
</Permission>
<Output>
<Name>output file</Name>
<Value>...</Value>
<Default>...</Default>
<Tooltip>...</Tooltip>
</Output>
```

The <Info>, <Hash>, <DA>, <Text>, <Permission>, <Name>, <Value>, and <Output> tag order cannot be changed. But the other tag order can be modified.

Checks of the tool on slots

- If the tag <Value> is declared in the XML file, the offset and size values are taken from the “Value” setting. Otherwise, they take the <Default> tag value.
- The flash memory slots must be aligned with sector (start and size).
- The flash memory slot total size must not exceed the flash memory size.
- Having the <Slot> and <OBKeySlot> tags in the same <Data> tag is not authorized.

Checks of the tool on OBKslots

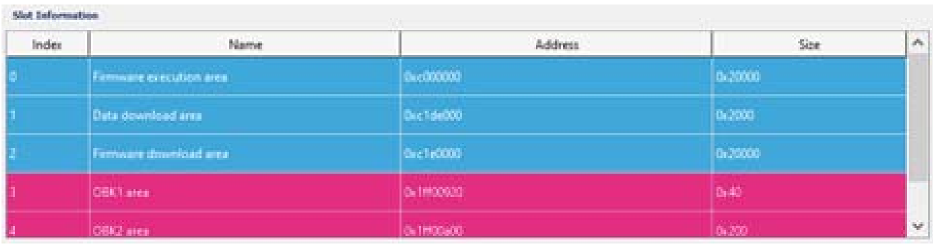
- If the tag <Value> is declared in the XML file, the offset and size values are taken from the “Value” setting, otherwise, they take the <Default> tag value.
- The slot OBK overlap check: overlap between OBKey slots is not authorized.
- The OBK slots must be aligned with sector (start and size).
- The OBK slot total size does not exceed the memory size.
- The OBK slots must be in the AuthOBK area. If the AuthOBK area was not defined, the check is applied on the OBK area
- Having the <Slot> and <OBKeySlot> tags in the same <Data> tag is not authorized.

Slot information table

- Flash memory and OBK slots are displayed in the slot information table if they exist. If there is no slot, the slot information table is not displayed.
- If there is <offset> in the declared slot name, the slot information table displays only the slot start addresses and names without <offset>.

The slot information table is updated when clicking on the “**Generate**” button. Moreover, the table is updated instantly and according to the dependency status. Only the visible slots must be displayed. The reserved slots must not be shown.

Figure 47. Slot information table



DT71649

Note: The slots can be unordered. The tool takes care of ordering them internally. However, for each slot, the size must be defined after its start, otherwise the size is considered as null value if it is not defined.

If one of the needed information is not defined in the XML file (such as flash memory size, ObkSector size), the related checks are not considered, nor applied.

*Only the dependency status for slot **Start** is taken into consideration, and it is not the case for its size.*

Table 2. Maximum tag length

Number max of characters	580 characters	100 characters	60 characters	12 characters	4 characters
Tags name[(1)]	Text Value Text Default All tool tips	Info Title All Names	All Dependency List Val File KeyType File Type	All contents that must contain addresses	Info GlobalAlign Info DoEncryption Widths Data Reserved Hidden File Align

1. Use a logical content for the other tags that are not mentioned in the table to avoid any errors. Any illogical content may lead to an invalid XML.

A.3.6 List tag

The <List> tag is an extension of the <Data> tag. It inherits all subtags from the <Data> tag.

The <List> tag adds the possibility to specify the possible values using the **<Val>** subtag.

Example:

```
<List>
   <Name>list1</Name>
   <Value>0x100</Value>
   <Width>4</Width>
   <Default>0x200</Default>
   <Val>0x100</Val>
   <Val>0x200</Val>
   <Val>0x300</Val>
   <Dependency>image</Dependency>
</List>
```

The possible values are displayed as a list in the tool. It is possible also to specify a displayed value when the value specified has a textual equivalent.

Example:

```
<List>
   <Name>list1</Name>
   <Value>0x100</Value>
   <Width>4</Width>
   <Default>0x200</Default>
   <Val>0x100,LOWEST_VALUE</Val>
   <Val>0x200, MEDIUM_VALUE</Val>
   <Val>0x300,HIGHEST_VALUE</Val>
</List>
```

In this example, the tool displays the textual data instead of the numerical ones.

The data inside the <Val> tag can be decimal or hexadecimal.

A.3.7 Boolean tag

The <Boolean> tag is an extension of the <Data> tag. It inherits all subtags from the <Data> tag. The <Boolean> tag is only used when two values can be inserted. The tool displays the <Boolean> tag as a tick box.

Example:

```
<Bool>
   <Name>bool1</Name>
   <Value>0x215</Value>
   <Width>4</Width>
   <Default>0x215</Default>
   <True>0x215</True>
   <False>0x1000</False>
   <Dependency>secure</Dependency>
</Bool>
```

The true value is selected when the tick box is selected. Otherwise, the false value is selected.

A.3.8 File tag

The <File> tag is used to input the user key. The result in the tool is a **Browse** button and a path area. The user can provide private or public keys as inputs. The output for a private key can be specified to become public. In this case, the tool derivates a public key from the private key. The supported key format is PEM (privacy enhanced mail) or RAW (binary).

Example:

```
<File>
   <Name>KeyFile2</Name>
   <Value>C:/Keys/pubkey.pem</Value>
   <Align>4</Align>
   <KeyType> ecdsa-p256, rsa-2048, rsa-3072 </KeyType>
   <Hash></Hash>
   <Type>public</Type>
   <Default>C:/Keys/pubkey.pem</Default>
</File>
<Type>public</Type>
```

The <Type> tag is optional and is useful to specify a public output from the private key.

```
<Type> Private_Raw </Type>
```

The private EC256 keys are imported with the DER information (70 bytes). Since the OBKey place in HDPL2 is not large, “Parivate_Raw” specifies that the key is provisioned in raw format (32 bytes), and not in 70 bytes. A key regeneration mechanism is offered to users and enabled through the KeyType tag.

Here is a typical tag with regeneration mechanism:

```
<KeyType>ecdsa256, rsa</KeyType>
```

If this subtag is specified in the key tag, a **Generate** button appears, and the possibility to select the key type is offered to users. The generated key has the same key name as the one available in the key selection area. The old key is backed up to keyname.pem_date_time.bak.

```
<Hash></Hash>
```

If this tag is added, the hash of the key is used instead of the whole key.

A.3.9 Permission tag

The <Permission> tag is used to input the permission for the debug configuration. It is optional.

Example:

```
<Permission>
   <Name>permission</Name>
   <Value>0x00005077</Value>
   <Width>4</Width>
   <Default>0x00000000</Default>
   <Tooltip>enter permission</Tooltip>
</Permission>
```

The <Permission> tag is mandatory for the creation of the .OBK file for debug authentication with certificate. For a debug authentication with password, the <Permission> tag is not used. Only a full regression is permitted.

Refer to *Table 3* for the allowed permissions.

Table 3. Permissions

Description	Permissions mask															
AP - M33	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Regression				Non intrusive debug				Secure debug				Nonsecure debug			
	-	Full	To IROT	To TZ	-	Reserved	Reserved	Reserved	-	Level 3	Level 2	Level 1	-	Level 3	Level 2	Level 1
	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

A.3.10 Text tag

The <Text> tag is used to input password for the configuration of the debug. It is optional.

Example:

```
<Text>
   <Name>password</Name>
   <Value>123456789123</Value>
   <MinWidth>4</MinWidth>
   <Width>16</Width>
   <Default>0000</Default>
   <Tooltip>enter password</Tooltip>
</Text>
```

The password length must be between 4 and 16 characters.

A.3.11 XML output

In addition to the binary output, the tool outputs an XML file with the same input name, where the data entered by the user are saved. To save user data, use the <Value> tag.

This XML file can be reused as an input. In this case, the tool updates the <Value> tag content and the XML file output is mandatory.

A.3.12 Debug authentication (DA)

Debug authentication with a certificate

For debug authentication with a certificate use case, a specific .OBK file must be generated using a specific XML file to configure the debug.

Below is a template of an XML file:

```
<Info>
   <Title>DA_DebugClose</Title>
   <ObDestAddress>0x0ffd0100</ObDestAddress>
   <DoEncryption>1</DoEncryption>
   <GlobalAlign>16</GlobalAlign>
   <FlashStart>0x00000008</FlashStart>
   <FlashSize>0x2000</FlashSize>
   <SectorSize>0x200</SectorSize>
</Info>
<Hash>
</Hash>
<DA>
</DA>
<Permission>
   <Name>permission</Name>
   <Value>0x00005077</Value>
   <Width>4</Width>
   <Default>0x00000000</Default>
   <Tooltip>enter permission</Tooltip>
</Permission>
<File>
   <Name>KeyFile1</Name>
   <Value>C:/Users/broukcha/Downloads/pubkey (4).pem</Value>
   <Align>4</Align>
   <KeyType>ecdsa-p256,rsa-2048,rsa-3072</KeyType>
   <Type>public</Type>
   <Default>C:/pubkey.pem</Default>
</File>

<Output>
```

```
    <Name>output file</Name>
    <Value>C:/DA_Close.OBKey</Value>
    <Default> C:/DA_Close.OBKey </Default>
    <Tooltip>enter output OBKey file with extension .OBK</Tooltip>
</Output>
```

Note: The debug authentication tag is optional.

A typical debug authentication OBKey consists of 3 * 32 bytes of data:

1. 32 bytes: hash of following 64 bytes using the hash tag.
2. 32 bytes: hash certificate using a certification tag. The hash mechanism is described below.
3. 32 bytes: permissions. Only the first U32 (32-bit unsigned integer type) is used to encode seven different permissions. Each permission can be selected separately. The final U32 value is an OR of the seven permissions.

Below an example of certificate hash computing:

1. Input the private key:


```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIMdGcLy36Fs4A++0KJQEkuc+P+nU97WorV5IDL28tVTCoAoGCCqGSM49AwEHoUQDQg
AE3PDQ9LzV4mpU7jbK1mDSg9EqvF9zB95YaJ53zWBFLnWMuttf6fiacQfloujqROwbCbfaKhqC
oCUqTBwm7h7Xzw==
-----END EC PRIVATE KEY-----
```


2. From the private key EcdsaP256Key-0.pem, generate a public key:

```
ssh-keygen -f EcdsaP256Key-0.pem -y > public.pub
```

The public key is:


```
ecdsa-sha2-
nistp256AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBNzw0PS81eJqVO
42ytZg0oPRKrxfcwfeWGied81gRS51jLrbX+n4mnEH5aLo6kTsGwm32ioagqAlKkwcJu4e188=
```


3. Convert b64 to hex.

   The input is:


```
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBNzw0PS81eJqVO42ytZg0o
PRKrxfcwfeWGied81gRS51jLrbX+n4mnEH5aLo6kTsGwm32ioagqAlKkwcJu4e188=
```


   The output is:


```
0000001365636473612D736861322D6E697374703235360000000086E697374703235360000
004104DCF0D0F4BCD5E26A54EE36CAD660D283D12ABC5F7307DE58689E77CD60452E758CBA
DB5FE9F89A7107E5A2E8EA44EC1B09B7DA2A1A82A0252A4C1C26EE1ED7CF
```


   The public key is:


```
DCF0D0F4BCD5E26A54EE36CAD660D283D12ABC5F7307DE58689E77CD60452E758CBADB5FE9
F89A7107E5A2E8EA44EC1B09B7DA2A1A82A0252A4C1C26EE1ED7CF
```


   Hash (SHA256) is equal to:

```
F4 0C 8F BF 12 DB 78 2A FD F4 75 96 6A 06 82 36 E0 32 AB 80 D1 B7 F1 BC 9F
E7 D8 7A 88 CB 26 D0
```

Debug authentication with a password

For a debug authentication with a password, a specific .OBK file must be generated using an XML file for the debug configuration. Below a template of an XML file:

```
<Info>
```

```
    <Title>DA Config</Title>
    <ObDestAddress>0x0ffd0100</ObDestAddress>
    <DoEncryption>1</DoEncryption>
    <GlobalAlign>16</GlobalAlign>
    <FlashStart>0x00000008</FlashStart>
    <FlashSize>0x2000</FlashSize>
    <FlashSectorSize>0x200</FlashSectorSize>
</Info>
<Hash>
</Hash>
<DA>
</DA>
<Text>
    <Name>password</Name>
    <Value>123456789123</Value>
    <MinWidth>4</MinWidth>
    <Width>16</Width>
    <Default>0000</Default>
    <Tooltip>enter password</Tooltip>
</Text>
<Output>
    <Name>output file</Name>
    <Value>C:/DA_Config.Obk</Value>
    <Default>DA_ Config.Obk</Default>
    <Tooltip>enter output OBKey file with extension .Obk</Tooltip>
</Output>
```

A password generation mechanism is offered to users in order to generate a password in TLV (tag-length-value) format. This password is used for a debug authentication.

Appendix B XML configuration for image generation

B.1 Main tool purpose

The tool purpose is to generate a signed and encrypted image using imgtool.

The tool input is an XML file describing the commands/parameters to be passed to imgtool. The tool imgtool is embedded in the STM32CubeProgrammer package.

B.2 Input XML file structure

The input XML has the following possible tags:

B.2.1 McubootFormat tag

The <McubootFormat> tag is the global container of all the rest of the tags. It specifies to the tool that a mcuboot data XML file is being parsed.

< McubootFormat > </ McubootFormat > must englobe all the other tags.

B.2.2 GlobalParam tag

The <GlobalParam> tag specifies the valid information for all the XML files.

A typical info tag structure is:

```
<GlobalParam>
   <Command>sign</Command>
   <Title>uRot generation</Title>
   <LinkedXML>C:/ STiRoT_Config.xml</LinkedXML>
</GlobalParam>
```

The info <GlobalParam> is mandatory in the mcuboot description XML.

- **<Command> [mandatory]:** It specifies the command to be passed to imgtool.
- **<Title> [mandatory]:** It describes the purpose of the XML.
- **<LinkedXML> [optional]:** The path of the XML to which the current XML can be linked to.

This occurrence can be 0 or more than 1 (multiple linked XML files).

Case of multiple LinkedXML (two or more)

The <LinkedXML> tag is described below. It indicates the LinkedXML file index between two # (example #02#) in the left of the LinkedXML file path.

```
<GlobalParam>
   <Command>sign</Command>
   <Title>uRot generation</Title>
   <LinkedXML>#01#STiRoT_Config1.xml</LinkedXML>
   <LinkedXML>#02#STiRoT_Config2.xml</LinkedXML>
</GlobalParam>
```

Case of a simple LinkedXML

Use the < LinkedXML ></ LinkedXML > tag format. There is no need to add any #. No index is needed.

All tags following the <GlobalParam> tag result in parameters passed to the command specified in the <Command> tag.

The order of tags specifies the order in the command line.

B.2.3 Param tag

The <Param> tag is used to indicate to the tool that a parameter must be added to the command line.

Below a typical <Param> tag structure example:

```
<Param>
   <Name>authentication key</Name>
   <Value>C:/ appli.bin</Value>
   <Link></Link>
   <Type>file</Type>
   <Command>-k</Command>
   <Enable>1</Enable>
   <Hidden>1</Hidden>
   <Tooltip>signature Key used</Tooltip>
   <Default>auto</Default>
</Param>
```

<Name> [mandatory]: The name is the text field that is displayed in the tool and describes this parameter.

<Value> [optional]: It is an optional tag that displays the value of the entered value.

<Link> [optional]: If this tag is present, the tool knows that it gets the parameter value from the linked XML specified in the <LinkedXML> tag (part of GlobalParam tag).
In the case of multiple LinkedXML files, the <Link> tag is described as below.

The LinkedXML file index refers to a specific LinkedXML file.

```
<Param>
   <Name>authentication key</Name>
   <Link>#01#</Link>
</Param>

<Param>
   <Name>firmware download area offset</Name>
   <Link>#02#+15</Link>
</Param>
```

If there is one LinkedXML, use the **<Link></Link>** tag format. There is no need to add any #. No index is needed.

Possible values of the Link tag are the following

- Empty (as described above)
- GetPublic in the case file (get public key of the mentioned key value defined into the LinkedXML file)

```
<Param>
   <Name>encryption key</Name>
   <Link>GetPublic</Link>
   <Type>file</Type>
   <Command>-E</Command>
   <Hidden>1</Hidden>
   <Default></Default>
</Param>
```

- A number to be added to the value preceded by a “+” character in the case of a data type.

```
<Param>
   <Name>firmware download area offset</Name>
   <Link>+15</Link>
   <Type>data</Type>
   <Command>-x</Command>
   <Hidden>1</Hidden>
   <Default></Default>
</Param>
```

Note: These possible values can also be used in the case of multiple LinkedXML files.

<Type> [mandatory]: The type specifies if the input is a file or data.

<Command> [optional]: It specifies the parameter to be passed to imgtool.

<Enable> [optional]: It is an optional tag. If enable = 1, a checked combo box is displayed in GUI activating this parameter and will be included in the command line. If enable = 0, an unchecked combo box is displayed (with the possibility to check it).

There is no checkbox displayed if no <Enable></ Enable >tag is defined.

<Hidden> [optional]: If hidden=0, the parameter is displayed in GUI. If hidden=1, the parameter is hidden but included in the command line.

<Tooltip> [optional]: The tool tip describes the purpose of using this parameter.

<Default> [mandatory]: It is the default value.

B.2.4 Output tag

The Output tag is a mandatory tag, which describes the output file.

The typical presentation is:

```
<Output>
   <Name>output-mcuboot format</Name>
   <Value>C /appli_enc_sign.bin</Value>
   <Tooltip>Signature Key used</Tooltip>
   <Default>C:/appli_enc_sign.bin</Default>
</Output>
```

<Name> [mandatory]: The name is the text field that is displayed in the tool describing this parameter.

<Value> [optional]: It is an optional tag that contains the entered value.

<Tooltip> [optional]: The tool tip describes the purpose of using this parameter.

<Default> [mandatory]: It is the default value.

When using the GUI:

By pressing the **Select Path** button, the output takes the same name of the input XML file name but with a ".bin" extension.
By pressing the **Cancel** button, the output path becomes the same as the input XML file.

B.2.5 Checks of the tool on Data and Info tags

All required tags must be present in the right order.

Any missing or wrong tag causes an error message. The XML file is invalid.

```
Error: XML document is invalid! Please insert a valid one.
```

Find below an XML order example:

```
<GlobalParam>
<Title>uRot Generation</Title>
<Command>sign</Command>
<LinkedXML>...</LinkedXML>
</GlobalParam>

<Param>
<Name>Authentication key</Name>
<Value>auto</Value>
<Link></Link>
<Type>File</Type>
<Command>-k</Command>
<Hidden>1</Hidden>
<Tooltip>Signature Key used</Tooltip>
<Default></Default>
</Param>

<Output>
<Name>output-mcuboot format</Name>
<Value>...</Value>
<Default>...</Default>
<Tooltip>...</Tooltip>
</Output>
```

The <Param> and <Output> tag order cannot be changed as well as the <Name> and <Value> tag order.

The other tag order can be changed.

Table 4. Maximum tag length

Tag name(1)	All tool tips	-GlobalParam -Title -All names	-Link -Type -All commands	-Enable -Hidden
Maximum number of characters	580	100	40	4

1. Use the right content for the other tags that are not mentioned in the table to avoid any errors. Any wrong content may lead to an invalid XML.

B.3 Image generation

The image generation can be done through CLI (command-line interface) or GUI (graphical user interface).

B.3.1 GUI execution

The imgtool GUI can be executed through the STM32TrustedPackageCreator.exe application or instantiated with the following command line. Then, the XML file is displayed in the GUI:

```
STM32TrustedPackageCreator.exe -pb File.xml
```

In this case, the **Open** button XML file in the imgtool tab is disabled in order to avoid the user to save these files in another directory than the one indicated by the calling application.

B.3.2 CLI execution

From the CLI, launch the following command and the output file is generated.

```
STM32TrustedPackageCreator_CLI.exe -pb File.xml
```

IDE can use the CLI command to generate the image.

During the image generation, the priority is given to the data in the <Value> tag, and not to the ones in the <Default tag>.

The save configuration process is applied on the same input XML file (Save modified parameter) specifically on: **<Value></Value>** and **<Enable></Enable>**

Press the **Refresh** button to refresh. It is used to update the loaded XML file in GUI to avoid reopening the same file.

To check the image generation, a log file named “imgtool-command.log” is created under the following path.

```
"C:\Users\username\STMicroelectronics\STM32CubeProgrammer".
```

This file contains the command used to generate the image.

B.4 TPC imgtool interaction with external tools

The user is able to edit the XML files (imgtool) via TPC to save the configuration instead of generating an image through –ng option.

```
STM32TrustedPackageCreator.exe -ng -pb imgtool.xml
```

The TPC is launched through a command line by external tools.

Without option -ng, the generate button is displayed. There is no **Save configuration** button.

```
STM32TrustedPackageCreator.exe -pb imgtool.xml
```

The **Open** button of the XML file in the imgtool tab must be disabled in order not to save these files in another directory than the one indicated by the external tool.
The TPC displays the content of the XML file already loaded.

In this case, only the imgtool tab must be displayed.

Case of saving configuration (with –ng):

```
STM32TrustedPackageCreator.exe -ng -pb MySTiRoT_S_Code_Image.xml -pb
MySTiRoT_NS_Code_Image.xml
```

Case of generating image (without –ng):

```
STM32TrustedPackageCreator.exe -pb MySTiRoT_S_Code_Image.xml -pb
MySTiRoT_NS_Code_Image.xml
```

Besides the imgtool tab, the user can display an OBKey tab using the command –OBKey.

Below an XML file example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SCHVerif.xsd">
<McubootFormat>
<GlobalParam>
   <Command>sign</Command>
   <Title>firmware nonsecure image generation</Title>
   <LinkedXML>../Config/OEMiRoT_Config.xml</LinkedXML>
</GlobalParam>
<Param>
   <Name>authentication nonsecure key</Name>
   <Link></Link>
   <Type>file</Type>
   <Command>-k</Command>
   <Hidden>1</Hidden>
   <Tooltip>signature Key used</Tooltip>
   <Default></Default>
</Param>
<Param>
   <Name>encryption key</Name>
   <Link>GetPublic</Link>
   <Type>file</Type>
   <Command>-E</Command>
   <Hidden>1</Hidden>
   <Default></Default>
</Param>
<Param>
   <Name>endianness</Name>
```

```
        <Type>data</Type>
        <Command>-e</Command>
        <Hidden>1</Hidden>
        <Default>little</Default>
    </Param>
    <Param>
        <Name>padding</Name>
        <Type>data</Type>
        <Command>--pad</Command>
        <Hidden>1</Hidden>
        <Default></Default>
    </Param>
    <Param>
        <Name>firmware area size</Name>
        <Value>0xA0000</Value>
        <Type>data</Type>
        <Command>-S</Command>
        <Default>0xa0000</Default>
    </Param>
    <Param>
        <Name>header size</Name>
        <Type>data</Type>
        <Command>-H</Command>
        <Hidden>1</Hidden>
        <Default>0x400</Default>
    </Param>
    <Param>
        <Name>padding header</Name>
        <Type>data</Type>
        <Command>--pad-header</Command>
        <Hidden>1</Hidden>
        <Default></Default>
    </Param>
    <Param>
        <Name>dependency with other image</Name>
        <Value>1,1.0.0</Value>
        <Type>dependency</Type>
        <Command>-d</Command>
        <Enable>0</Enable>
        <Tooltip>To be enabled when a compatibility with a specific firmware
        image version is required. Firmware image version format is
        x.y.z</Tooltip>
        <Default>1,1.0.0</Default>
    </Param>
    <Param>
```

```
    <Name>Write Option</Name>
    <Type>data</Type>
    <Command>--overwrite-only</Command>
    <Hidden>1</Hidden>
    <Default></Default>
</Param>
<Param>
    <Name>Version</Name>
    <Value>1.0.0</Value>
    <Type>data</Type>
    <Command>-v</Command>
    <Tooltip>Version of the firmware binary. Format is x.y.z</Tooltip>
    <Default>1.0.0</Default>
</Param>
<Param>
    <Name>security counter</Name>
    <Value>auto</Value>
    <Type>data</Type>
    <Command>-s</Command>
    <Hidden>1</Hidden>
    <Default>auto</Default>
</Param>
<Param>
    <Name>align</Name>
    <Type>data</Type>
    <Command>--align</Command>
    <Hidden>1</Hidden>
    <Default>16</Default>
</Param>
<Param>
    <Name>Hex format</Name>
    <Value>0x80C0000</Value>
    <Type>data</Type>
    <Command>-x</Command>
    <Hidden>1</Hidden>
    <Default>0x80c0000</Default>
</Param>
<Param>
    <Name>firmware binary input file</Name>


<Value>C:/Users/broukcha/Downloads/CubeFW/STM32Cube_FW_H5_V1.0.0RC2/Projec
ts/STM32H573I-DK/ROT_Provisioning//../Applications/ROT
/OEMiROT_Appli_TrustZone/EWARM/NonSecure/STM32H573I-DK_NS/Exe
/Project.bin</Value>
```

```
<Type>file</Type>
<Tooltip>select the firmware binary file to be processed for the image generation</Tooltip>
<Default>../../../Applications/ROT/OEMiROT_Appli_TrustZone/EWARM/NonSecure/STM32H573I-DK_NS/Exe/Project.bin</Default>
</Param>
<Output>
<Name>image output file</Name>
<Value>../../../Applications/ROT/OEMiROT_Appli_TrustZone/Binary/rot_tz_ns_app_enc_sign.hex</Value>
<Default>../../../Applications/ROT/OEMiROT_Appli_TrustZone/Binary/rot_tz_s_app_enc_sign.hex</Default>
</Output>
</McubootFormat>
</Root>
```

10 Reference documents

Table 5. Document references

ID	Revision	Title
[1]	Latest version	*STM32CubeProgrammer software description* user manual (UM2237), STMicroelectronics.

11 Revision history

Table 6. Document revision history

Date	Revision	Changes
20-Dec-2017	1	Initial release.
07-Mar-2019	2	Added *Section 3.6: STM32H5 OBKey generation*. Updated: – *Section 4.1: SFI generation* – *Figure 10: STM32 trusted package creator tool's available commands (part 1)* – *Figure 17: STM32 trusted package creator tool GUI SFI tab* – *Figure 20: Firmware file addition*
16-Apr-2019	3	Updated *Section 1: System requirements*. Added *Section 2.5: HSM preparation process*. Updated *Figure 17: STM32 trusted package creator tool GUI SFI tab*.
15-Oct-2019	4	Updated: – *Section 2.5.1: Why we need an HSM* – *Section 2.5.2: HSM programming* – *Figure 8: HSM programming tab* – *Section 2.5.3: Reading HSM information* – *Figure 9: Reading HSM information* – *Figure 10: STM32 trusted package creator tool's available commands (part 1)* and *Figure 11: STM32 trusted package creator tool's available commands (part 2)* – *Figure 17: STM32 trusted package creator tool GUI SFI tab* – *Section 3.2: HSM provisioning command* Added *Section 3.3: SFI/SFIx generation command*.
07-Jan-2020	5	Added: – *Section 2.2: SFIx preparation process* Updated: – *Figure 10: STM32 trusted package creator tool's available commands (part 1)* – *Figure 11: STM32 trusted package creator tool's available commands (part 2)* – *Section 3.3: SFI/SFIx generation command* – *Figure 17: STM32 trusted package creator tool GUI SFI tab* – *Figure 19: STM32 trusted package creator tool GUI SFIx tab* (changed image and title) Replaced *Section 4.3 SFU generation* with *Section 4.3: SFIx generation*.
25-Feb-2020	6	Updated: – *Introduction* – *Section 3.2: HSM provisioning command.* Removed sections: – *SFU preparation process* – *SFU generation command.*

Table 6. Document revision history (continued)

Date	Revision	Changes
27-Jul-2020	7	Updated: – *Introduction* – *Figure 6: SSP preparation process* Added: – *Section 2.4: SSP preparation process* – *Section 3.5: SSP generation command* – *Section 4.4: SSP generation*
05-Jul-2021	8	Updated: – In the whole document, replaced STM32H7A/B by STM32H7A3/7B3 and STM32H7B0. – *Section 2.5.2: HSM programming*: updated personalization data and personalization data file.
07-Mar-2022	9	Updated: – *Section 2.2.2: Area K* – *Section 3.2: HSM provisioning command* – *Section 4.1: SFI generation* Added *Section 9: SFI OB generation*.
24-Jun-2022	10	Updated: – *Section 3.2: HSM provisioning command* – *Section 3.3: SFI/SFIx generation command* – *Section 3.4: SMI generation command* – *Section 4.4: SSP generation* – *Section 2.2.2: Area K* Added: *Section 2.5.4: List of supported HSM cards*.
05-Dec-2022	11	Updated: – *Section 3.2: HSM provisioning command* – *Section 3.3: SFI/SFIx generation command* – *Section 3.4: SMI generation command* – *Section 4.4: SSP generation* – *Section 2.2.2: Area K*
01-Mar-2023	12	Added: – *Appendix A: XML configuration for OBKey generation* – *Appendix B: XML configuration for image generation* Updated *Chapter 5: Option bytes file*.

Table 6. Document revision history (continued)

Date	Revision	Changes
21-Jul-2023	13	Global document update for compatibility with the secure manager. Updated: – *Chapter 1: System requirements* – *Section 2.1: SFI preparation process* – *Section 3.3: SFI/SFIx generation command* – *Section 4.1: SFI generation* – *Section 4.2: SFIx generation* – *Figure 14: Firmware file addition* – *Figure 15: Successful SFI generation* Added: – *Section : The tool input is an MCU, a Debug Authentication Type and a path for the Output File to generate .OBK files.* – *Figure 3: SFI file structure for devices supporting the secure manager* – *Figure 16: Successful SFI generation for devices supporting the secure manager* – *Figure 17: Parsing of the MCSV file for modules list* – *Figure 20: SFIx window for devices supporting the secure manager* – *Figure 29: STM32TrustedPackageCreator license generation tab*
26-Mar-2024	14	Added: – *Section 3.12: Regeneration of encryption and authentication keys* – *Section 4.4.3: Debug authentication* Updated: – *Section 3.5: OBKey generation* – *Section 3.6: Image generation* – *Section 4.4: Security features* – *Section A.1: Main tool purpose* – *Section B.1: Main tool purpose* Removed: – section 4.5.3 Certificate generation for debug authentication
26-Jun-2024	15	Added: – *Section 3.9: Key generation and FW signature (STM32WB0x/STM32WL3x only)* – *Section 3.10: SSP-SFI image generation* – *Section 3.8: SSP Backup generation* – *Section 4.3.1: SSP image generation* – *Section : 212 bytes for STM32MP13xx* – *Section 4.5: STM32WB0x Secure Boot* – *Section 4.5.1: Generate key* – *Section 4.5.2: Firmware Signature* – *Section 4.6: SSP backup generation* – *Section 4.7: SSP-SFI image generation* Updated: – *Section 1: System requirements*

Table 6. Document revision history (continued)

Date	Revision	Changes
27-Nov-2024	16	Added: – *Section 3.13: MCE image generation*: Added new section for MCE Image generation. – *Section 3.3: SFI/SFIx generation command*: Added new syntax for external modules list. – *Section 4.8: SSP secrets generation*: Added new section for SSP Secrets Generation. – *Section 4.9: X-CUBE-RSSe and STM32MPUSSP-UTIL*: Added *Section 4.9.1: X-CUBE-RSSe* and *Section 4.9.2: STM32MPUSSP-UTIL*. Updated: – Introduction: Removed the list of secure programming solutions and SMI description. – *Section 2.2.3: Constraints for SFIx images*: Added new constraint for SFIx images. – *Section 2.4: HSM preparation process*: Reordered subsections under HSM preparation process. – *Section 3.10: Key generation and FW signature (STM32WB0x/STM32WL3x only)*: Added support for STM32WL3x. – *Section 4.2: SFIx generation*: Added OTFDEC settings switch. – *Section 8: SFI checking*: Changed section title. Deleted: – Introduction: Removed the SMI description.
07-Mar-2025	17	Added *Section 3.7: Generation of encryption and authentication keys*. Updated *Section 3.13: MCE image generation*, *Section 4.4.1: OBKey content generation*, and *Section 4.4.3: Debug authentication*. Minor text edits across the whole document.