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 module install (SMI). SMI is used to protect a part of the firmware (a section of an ELF file) by also encrypting this section using an AES-GCM key. A combined SFI-SMI image is an SFI image that contains one or more module areas.
- 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

1. System requirements ................................................. 6

2. Preparation processes .................................................. 7
   2.1 SFI preparation process ........................................... 7
   2.2 SFIx preparation process ......................................... 9
      2.2.1 Area E ......................................................... 9
      2.2.2 Area K ......................................................... 9
      2.2.3 Constraints for SFIx images ................................. 10
   2.3 SMI preparation process .......................................... 11
   2.4 SSP preparation process ......................................... 13
   2.5 HSM preparation process ......................................... 14
      2.5.1 Why we need an HSM ....................................... 14
      2.5.2 HSM programming ........................................... 15
      2.5.3 Reading HSM information ................................... 17

3. STM32 Trusted Package Creator tool commands ..................... 18
   3.1 Command line interface (CLI) .................................... 18
   3.2 HSM provisioning command ....................................... 20
   3.3 SFI/SFIx generation command .................................... 23
   3.4 SMI generation command .......................................... 26
   3.5 SSP generation command .......................................... 27
   3.6 STM32WB firmware signing ....................................... 29

4. STM32 Trusted Package Creator tool graphical user interface (GUI) 30
   4.1 SFI generation .................................................... 32
   4.2 SMI generation .................................................... 35
   4.3 SFIx generation .................................................... 38
   4.4 SSP generation .................................................... 41

5. Option bytes file ..................................................... 43

6. Log dialog ............................................................. 44
<table>
<thead>
<tr>
<th>#</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Settings</td>
<td>45</td>
</tr>
<tr>
<td>8</td>
<td>SFI/SMI checking</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>Reference documents</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>Revision history</td>
<td>48</td>
</tr>
</tbody>
</table>
List of tables

Table 1. Document references ................................................................. 47
Table 2. Document revision history .......................................................... 48
# List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>SFI preparation process</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>SFI file structure</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example of split of SFIx image</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4</td>
<td>SMI preparation process</td>
<td>11</td>
</tr>
<tr>
<td>Figure 5</td>
<td>SMI file structure</td>
<td>12</td>
</tr>
<tr>
<td>Figure 6</td>
<td>SSP preparation process</td>
<td>13</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Encryption file scheme</td>
<td>14</td>
</tr>
<tr>
<td>Figure 8</td>
<td>HSM programming tab</td>
<td>16</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Reading HSM information</td>
<td>17</td>
</tr>
<tr>
<td>Figure 10</td>
<td>STM32 Trusted Package Creator tool's available commands (part 1)</td>
<td>18</td>
</tr>
<tr>
<td>Figure 11</td>
<td>STM32 Trusted Package Creator tool's available commands (part 2)</td>
<td>19</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Get HSM information in CLI mode</td>
<td>21</td>
</tr>
<tr>
<td>Figure 13</td>
<td>SFI generation with an ELF file</td>
<td>25</td>
</tr>
<tr>
<td>Figure 14</td>
<td>SFI generation with a binary file</td>
<td>25</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Combined SFI-SMI generation</td>
<td>26</td>
</tr>
<tr>
<td>Figure 16</td>
<td>SMI generation</td>
<td>27</td>
</tr>
<tr>
<td>Figure 17</td>
<td>STM32 Trusted Package Creator tool GUI SFI tab</td>
<td>30</td>
</tr>
<tr>
<td>Figure 18</td>
<td>STM32 Trusted Package Creator tool GUI SMI tab</td>
<td>31</td>
</tr>
<tr>
<td>Figure 19</td>
<td>STM32 Trusted Package Creator tool GUI SFIx tab</td>
<td>32</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Firmware file addition</td>
<td>33</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Successful SFI generation</td>
<td>34</td>
</tr>
<tr>
<td>Figure 22</td>
<td>ELF file selection</td>
<td>35</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Successful SMI generation</td>
<td>37</td>
</tr>
<tr>
<td>Figure 24</td>
<td>External firmware file selection</td>
<td>38</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Successful SFIx generation</td>
<td>40</td>
</tr>
<tr>
<td>Figure 26</td>
<td>STM32TrustedPackageCreator SSP GUI table</td>
<td>41</td>
</tr>
<tr>
<td>Figure 27</td>
<td>SSP output information</td>
<td>42</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Example of an option bytes file</td>
<td>43</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Example of a log dialog</td>
<td>44</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Settings dialog</td>
<td>45</td>
</tr>
<tr>
<td>Figure 31</td>
<td>SFI checking</td>
<td>46</td>
</tr>
</tbody>
</table>
1 System requirements

Supported operating systems and architectures:
• Linux® 64-bit
• Windows® 7/8/10 32-bit and 64-bit
• macOS® (minimum version OS X® Yosemite)

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

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

2.1 SFI preparation process

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

Figure 1. SFI preparation process
Before performing AES-GCM to encrypt an area, the tool calculates the Initialization Vector (IV) as:

\[ IV = \text{nonce} + \text{Area Index} \]

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

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

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

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

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

**Figure 2. SFI file structure**
To prepare an SFI image from multiple firmware files you have to make sure that there is no overlap between their segments, otherwise an error message is shown: “Overlap between segments, unable to merge firmware files”.

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

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

2.2 SFIx preparation process

In addition to the SFI preparation process, mentioned in the previous paragraph, two new areas are added in 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 (STM32H7A/B an STM32L5)
  - 4…7: OTFD2 (STM32H7A/B)
- OTFD region_mode (uint32_t) bit [1:0]:
  - 00: instruction only (AES-CTR)
  - 01: data only (AES-CTR)
  - 10: instruction + data (AES-CTR)
  - 11: instruction only (Enhanced cipher)
- OTFD key_address in internal Flash memory (uint32_t)

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

2.2.2 Area K

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

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

Example of an area ‘K’:

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

There are two key areas:

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

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

**Figure 3. Example of split of SFIx image**
2.3 SMI preparation process

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

Figure 4. SMI preparation process

The AES-GCM encryption is performed using the following inputs:
- Nonce as the initialization vector (IV)
- The security version as additional authenticated Data (AAD)

Before SMI preparation, the following checks are performed:
- A proprietary code read out protection (PCROP) section must be aligned on a Flash word (256 bits) otherwise a warning is shown
- The section’s size must be at least 2 Flash words (512 bits) otherwise a warning is shown
- The section must end on a Flash word boundary (a 256 bit word) or a warning is shown
- If the section immediately following the PCROP area starts in the last Flash word of the PCROP section, the generation fails and an error message is shown.

After the SMI preparation, a clear (that is, not encrypted) ELF file is also generated containing program data and only clear sections of the code.
The structure of an SMI file is shown in Figure 5.

**Figure 5. SMI file structure**

- SMI Magic
- Dest address size
- Authentication Tag of the signed header + module code using the firmware encryption key
- Security Version
- Encrypted protected module code

**SMI signed header**

**SMI Encrypted section**

MS48674V1
2.4 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 1.

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

**Figure 6. 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 is 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 (128bits).
- **Encryption nonce**: AES nonce (128 bits).
- **OEM FW key**: This is the major part of the secure boot sequence. Based on ECDSA verification, the key is used to validate the signature of the loaded binary.

The first layout part (SSP Magic, Protocol version, ECDSA Public key, secret size) is used as additional authenticated data (AAD) to generate the payload tag. This is checked by the ROM code during the decryption. This encrypted file is automatically generated by STM32TrustedPackageCreator tool.
2.5 **HSM preparation process**

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

2.5.1 **Why we need an HSM**

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

An important secure flashing feature - 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 the encrypted firmware (SFI and/or SMI) on specific devices.

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

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

The general scheme is as follows:

1. Generate a firmware key and use it to encrypt the firmware (SFI/SMI) when the firmware provider wishes to distribute a new firmware.

2. While downloading the firmware to a device, a device identifier is sent to the HSM, which returns a license for the identified device. The license contains the encrypted firmware key, and only this device can decrypt it.

3. The number of programmed devices is under HSM control. It is decremented once a license is generated.
STM32 Trusted Package Creator allows HSM configuration for secure firmware flashing as described in Section 2.5.2: HSM programming.

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

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

- **HSMv2**: dynamic HSM. This version allows generation of firmware licenses targeting STM32 secure programming devices that are chosen via a personalization data at the OEM site.

### 2.5.2 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 bytes long, but the last 4 bytes of the nonce must be zero (only 96 bits of the nonce are used in the AES128-GCM algorithm).
  
  Both fields must remain secret, so they are encrypted before being sent to the chip.
  
  The key and the nonce remain the same for all licenses for a given piece of firmware.
  
  However, they must be different for different firmware or different versions of the same firmware.

- **Firmware identifier**: Identifies the correct HSM for a given firmware.

- **Personalization data**: 108 bytes that are encrypted before being sent to the chip in cases where each device has its own configuration.

Once this data is programed into the HSM, the HSM is automatically locked. Figure 8: HSM programming tab shows the GUI provided by the tool for HSM programming by the firmware provider.

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

When the ST personalization package is installed successfully, it is not possible to further modify the personalization data and the HSM goes in the OPERATIONAL_STATE.
The tab parameters are as follows:

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

When all fields are properly filled in, the user can program the HSM file by clicking on the Program HSM button (the button becomes active).

**Caution**: Once this data is programmed into the HSM, the HSM is automatically locked.
2.5.3 Reading HSM information

The programmed information is verifiable using the HSM information panel shown in Figure 9:

**Figure 9. Reading HSM information**

![HSM Information Panel](image)

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

- **HSM status** can be:
  - **OEM_STATE**: HSM not programmed.
  - **OPERATIONAL_STATE**: HSM programmed and locked. It can no longer be programmed.

- **Version**: indicates the hardware version of the used HSM, it can be
  - 1: HSM version 1
  - 2: HSM version 2.

- **Type**: indicates which security process being used:
  - **SFI**: Static license for a complete application.
  - **SMI**: Static license for a library
  - **SSP**: Static license for secrets.
  - type not available.
3 STM32 Trusted Package Creator tool commands

3.1 Command line interface (CLI)

The following sections describe how to use the STM32 Trusted Package Creator tool from the command line interface. The available commands are shown in Figure 10 and Figure 11.

Figure 10. STM32 Trusted Package Creator tool's available commands (part 1)
Figure 11. STM32 Trusted Package Creator tool's available commands (part 2)
3.2 HSM provisioning command

-hsm, --hsm

Description: This command allows to program HSM card.
In order to configure the HSM before programming, the user must provide the mandatory inputs by using the options listed below.

-i, --index
Description: Select the card index, number in decimal format.
Syntax: -i <number>

-k, --key
Description: Set the AES-GCM encryption key.
Syntax: -k <Key_File>
<Key_File>: Bin file path, its size must be 16 bytes.

-n, --nonce
Description: Set the AES-GCM nonce.
Syntax: -n <Nonce_File>
<Nonce_File>: Bin file path, its size must be 12 bytes.

-id, --id
Description: Set the firmware identifier.
Syntax: -id <Firmware_Id>
<Firmware_Id>: Input as a string format, don’t use spaces.

-mc, --maxcounter
Description: Set the maximum number of licenses to be requested.
Syntax: -mc <Max.Counter>
<Max.Counter>: Number in decimal format, must not exceed 1000000.

-pd, --persoData
Description: Set the personalization data configuring the HSM version 2.
Syntax: -pd <PersoData_File>
<PersoData_File>: Bin file path. Its size must be 108 bytes.

-info, --info
Description: Get HSM information.
Example of HSM version 1 provisioning:

STM32TrustedPackageCreator_CLI -hsm -i 1 -k "C:\TrustedFiles\key.bin" -n "C:\TrustedFiles\nonce.bin" -id HSMv1_SLOT_1 -mc 2000

Example of HSM version 2 provisioning:

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

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

Note: 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 12. 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. You must select the correct one.
To obtain the appropriate personalization data, you first need to get the product ID:

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

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

- A file named “certificate.bin” is created in the same path of the STM32CubeProgrammer executable file.

- Open the certificate file with a text editor tool, then read the 8 characters from the header which represent the product ID.

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

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

```
ProductID_FlowType_LicenseVersion_SecurityVersion.enc.bin
```

For example: 47201003_SFI_01000000_00000000.enc.bin

Based on this name, the associated information is:

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

-fir, --firmware

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

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

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

-firx, --firmwx

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

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

-Firmware_file: Supported external firmware files are: ELF, HEX, SREC, BIN
[<Address>]: Only in the case of BIN input file (in any base).
[<Region_Number>]: Only in the case of BIN input file (in any base):
[0:3]: OTFD1 (STM32H7A / STM32L5)
[4:7]: OTFD2 (STM32H7A/B case)
[<Region_Mode>]: Only in case of BIN input file (in any base), only two bits
[0:1] where
00: instruction only (AES-CTR)
01: data only (AES-CTR),
10: instruction + data (AES-CTR)
11: instruction only (EnhancedCipher)
[<key_address>]: Only in the case of BIN input file (in any base), random key values in internal Flash memory

-k, --key

Description: Set the AES-GCM encryption key.

Syntax: -k <Key_file>

-Key_file: A 16-byte binary file.

-kx, --keyx

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

Syntax: -kx <Key_area_File>

-n, --nonce
Description: Set the AES-GCM nonce.

Syntax:  -n  <Nonce_file>

<Nonce_file> : A 12-byte binary file.

-v, --ver

Description: Set the image version.

Syntax:  -v  <Image_version>

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

-ob, --obfile

Description: Provide an option bytes file.

Syntax:  -ob  <CSV_file>

<CSV_file>: A csv file with 9 values.

-m, --module

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

Syntax:  -m  <SMI_file>

<SMI_file> : SMI file.

[<Address>] : Address only for relocatable SMI.
UM2238 STM32 Trusted Package Creator tool commands

-**rs, --ramsize**
  **Description:** Define available ram size for SFI programming (for multi-image).
  **Syntax:** `-rs <size>`
  `<size>`: Size in bytes.

-**ct, --token**
  **Description:** Address at which to store the Continuation token. The address must not collide with other areas. (for multi-image).
  **Syntax:** `-ct <Address>`

-**o, --outfile**
  **Description:** Set the SFI file to be created
  **Syntax:** `-o <out_file>`
  `<out_file>`: SSFI file to be generated, must have the .sfi/sfix extension.

Example for SFI:

With an ELF file:

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

**Figure 13. 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 14. SFI generation with a binary file**
Combined SFI-SMI:

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

Figure 15. Combined SFI-SMI generation

3.4 SMI generation command

```
-smi, --smi
```

**Description:** This command generates an SMI image file.

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

```
-elf, --elfile
```

**Description:** Set the input ELF file.

**Syntax:** `-elf <ELF_file>`

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

```
-s, --sec
```

**Description:** Set the name of the section to be encrypted.

**Syntax:** `-s <section_name>`

```
<section_name> : Section name.
```

```
-k, --key
```

**Description:** Set the AES-GCM encryption key.

**Syntax:** `-k <Key_file>`

```
<Key_file> : A 16-byte binary file.
```

```
-n, --nonce
```

**Description:** Set the AES-GCM nonce.

**Syntax:** `-n <Nonce_file>`

```
<Nonce_file> : A 12-byte binary file.
```

```
-sv, --sver
```

**Description:** Set the security version file.

**Syntax:** `-sv <SV_file>`

```
```
-o, --outfile

Description: Set the SMI file to be created

Syntax:  -o  <out_file>

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

-c, --clear

Description: Set the clear ELF file to be created.

Syntax:  -c  <ELF_file>

<ELF_file>: Clear ELF file to be generated.

Example

STM32TrustedPackageCreator_CLI -smi -elf FIR_module.axf -s "ER_PCROP" -k test_firmware_key.bin -n nonce.bin -sv svFile -o test.smi -c clear.smi

Figure 16. SMI generation

3.5 SSP generation command

The STM32 Trusted Package Creator tool CLI exports a 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.

-ru, --rma_unlock

Description: RMA unlock password

Syntax:  -ru <RMA_Unlock>

<RMA_Unlock>: Hexadecimal value 0 to 0x7FFF

-rr, --rma_relock

Description: RMA relock password

Syntax:  -rr <relock_value>

<relock_value>: Hexadecimal value 0 to 0x7FFF

-b, --blob

Description: Binary to encrypt

Syntax:  -b <Blob>

<Blob>: Secrets file of size 148 bytes

-pk, --pubk
Description: OEM public key file
Syntax:  -pk <PubK.pem>
        <PubK> : pem file of size 178 bytes
-k,  --key
Description: AES-GCM encryption key
Syntax:  -k <Key_File>
        <Key_File> : Bin file, its size must be 16 bytes
-n,  --nonce
Description: AES-GCM nonce
Syntax:  -n <Nonce_File>
        <Nonce_File> : Bin file, its size must be 16 bytes
-o,  --out
Description: Generate SSP file
Syntax:  -out <Output_File.ssp>
        <Output_File> : SSP file to be created with (extension .ssp)

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

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

Once the operation is done, a green message is displayed to indicate that the generation finished successfully, otherwise an error occurs.
3.6 STM32WB firmware signing

- `sign`, `--sign`

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

  - `-bin`, `--binary` : Input firmware image in a binary format to be signed
    
    `<Img_File>` : Binary image file path
  
  - `-prvk`, `--privatekey` : EC private key in PEM format with Prime256v1 curve.
    
    It can be generated using the following OpenSSL-tool command:
    
    ```
    openssl.exe ecparam -name prime256v1 -genkey -noout -out prvkey.pem
    
    `<prvk_File>` : Private key file Path
    
    `-v`, `--version` : Signed image version
    
    `<Img_Ver>` : Version number. Should be integer.
  
  - `-o`, `--outfile` : Output image file path
    
    `<Output_File>` : Signed image file path

  Before loading the signed image to STM32WB devices using STM32CubeProgrammer via 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].
This section describes how to use the STM32 Trusted Package Creator tool with its graphical user interface.

The STM32 Trusted Package Creator tool GUI presents three tabs: one for SFI generation (Figure 17), one for SMI generation (Figure 18) and one for SFU generation (Figure 19).

Figure 17. STM32 Trusted Package Creator tool GUI SFI tab
Figure 18. STM32 Trusted Package Creator tool GUI SMI tab
4.1 SFI generation

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

**Firmware files:**

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

**Note:** If the file is valid, it is added in the firmware files list. Selecting it makes several pieces of related information appear in the Firmware information section (Figure 20), otherwise an error message box is shown saying either the file could not be opened or the file is not valid. If the file is in a binary format, a dialog box appears requesting that an address be provided. A file can be removed with the **remove** button.
Encryption key and nonce file:
The encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selection with the **open** button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).

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

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

**Image version:**
Image version value in [0..255].

**RAM size:**
Size of RAM memory available for SFI programming.
Continuation token address:
Address at which to store the Continuation token. The address must not collide with other areas.

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

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

If everything goes well, a message box indicating successful generation appears (Figure 21) and information about the generated SFI file is displayed in the SFI information section.

Figure 21. Successful SFI generation
4.2 SMI generation

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

Elf file:

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

If the file is valid, information is displayed in the “ELF information” tab (Figure 22), otherwise an error message box is shown saying either the file could not be opened or that the file is not valid.

Figure 22. ELF file selection
Encryption key and nonce file:
As for SFI, the Encryption key and nonce file can be selected in the same way as the firmware file. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).

Security version file:
The security version file size must be 16 bytes.
A security version file is provided under the Security_Version folder.

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

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

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

A message box informing the user that generation was successful appears (Figure 23), in addition of information about the generated SMI file, otherwise an error is displayed.
Figure 23. Successful SMI generation
4.3 SFIx generation

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

External firmware files:

The user needs to enter the input for external firmware files. If the file is valid, several pieces of related information appear in the Firmware information section. A popup also appears that contains fields to be filled-in (Figure 24).

- Start address: the start address of a binary
- 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 24. 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 25*), contains two sections:

- An overview section mentioning the name of the output file and various information including the size of the file, it’s image version, internal/external segments and so on.
- A ‘segments’ section showing the different areas.
- A ‘parse SFIx file’ section, allowing the user to browse a .sfix output and parse it to display SFIx information.
Figure 25. Successful SFIx generation
4.4 SSP generation

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

**Figure 26. STM32TrustedPackageCreator SSP GUI table**

- **RMA Lock:**
  Unlock password, hexadecimal value from 0x0 to 0x7FFF

- **RMA Relock:**
  Relock password, hexadecimal value from 0x0 to 0x7FFF

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

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

PEM file of size 178 bytes.

**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 was done, we can get SSP information from the SSP overview section.

![Figure 27. SSP output information](image)

**File name:**

SSP output file name.

**Type:**

SSP extension.

**Size:**

indicates the generated file size including all data fields.
5 Option bytes file

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

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

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

Figure 28. Example of an option bytes file
6 Log dialog

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

Figure 29. 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 30. Settings dialog**

Padding byte:

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

Settings file:

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

Log file:

When checked, a log file is generated in the selected path.
8 SFI/SMI checking

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

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

*Figure 31. SFI checking*
9 Reference documents

Table 1. Document references

<table>
<thead>
<tr>
<th>ID</th>
<th>Revision</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>STMicroelectronics, UM2237</td>
</tr>
</tbody>
</table>
## Revision history

**Table 2. Document revision history**

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-Dec-2017</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>07-Mar-2019</td>
<td>2</td>
<td>Added Section 3.6: STM32WB firmware signing. 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</td>
</tr>
<tr>
<td>16-Apr-2019</td>
<td>3</td>
<td>Updated Section 1: System requirements Added Section 2.5: HSM preparation process. Updated Figure 17: STM32 Trusted Package Creator tool GUI SFI tab.</td>
</tr>
<tr>
<td>15-Oct-2019</td>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>07-Jan-2020</td>
<td>5</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 2. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>25-Feb-2020</td>
<td>6</td>
<td>Updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Introduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 3.2: HSM provisioning command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– SFU preparation process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– SFU generation command</td>
</tr>
<tr>
<td>27-Jul-2020</td>
<td>7</td>
<td>Updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Introduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Figure 6: SSP preparation process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 2.4: SSP preparation process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 3.5: SSP generation command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Section 4.4: SSP generation.</td>
</tr>
</tbody>
</table>