Introduction

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

These tools are compatible with all STM32 devices supporting SFI.

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

Please refer to AN4992 [1], which provides an overview of the secure firmware install (SFI) solution and how this provides a practical level of protection of the IP chain from the firmware development up to programming the device on-chip Flash memory.
## Contents

1  **General information** ......................................................... 8  
   1.1  Licensing information ................................................. 8  
   1.2  Acronyms and abbreviations ......................................... 8  

2  **How to generate an execute-only/position independent library for SMI preparation** ......................................................... 9  
   2.1  Requirements ............................................................. 9  
   2.2  Toolchains allowing SMI generation .................................. 9  
   2.3  Execute-only/position independent library scenario example under EWARM ................................................................. 10  
       2.3.1  Relocatable library preparation steps .......................... 10  
       2.3.2  Relocatable SMI module preparation steps ...................... 14  
       2.3.3  Application execution Scenario .................................. 15  

3  **Encrypted firmware(SFI)/module(SMI) preparation using STM32TrustedPackageCreator** ......................................................... 17  
   3.1  System requirements ..................................................... 17  
   3.2  SFI generation process .................................................. 17  
   3.3  SMI generation process .................................................. 26  
   3.4  STM32TrustedPackageCreator tool in the command line interface ................................................................. 29  
       3.4.1  Steps for SFI generation (CLI) .................................... 30  
       3.4.2  Steps for SMI generation (CLI) .................................... 32  
   3.5  Using the STM32TrustedPackageCreator tool graphical user interface ................................................................. 34  
       3.5.1  SFI generation using STPC in GUI mode .......................... 34  
       3.5.2  SMI generation using STPC in GUI mode .......................... 38  
       3.5.3  Settings .............................................................. 40  
       3.5.4  Log generation ....................................................... 42  
       3.5.5  SFI and SMI file checking function ............................... 43
4 Encrypted firmware (SFI)/module(SMI) programming using STM32CubeProgrammer ........................................... 44
  4.1 Chip certificate authenticity check and license mechanism ............... 44
    4.1.1 Device authentication ........................................ 44
    4.1.2 License mechanism ............................................. 44
      Licenses mechanism general scheme .................................. 44
      License distribution ................................................. 45
      HSM programming by OEM for License distribution .................. 45
  4.2 Secure programming using bootloader interface .......................... 47
    4.2.1 Secure firmware installation using Bootloader interface flow ........ 47
    4.2.2 Secure Module installation using bootloader interface flow ........... 49
    4.2.3 STM32CubeProgrammer for SFI using bootloader interface .......... 49
    4.2.4 STM32CubeProgrammer for SMI via Bootloader interface ............. 50
    4.2.5 STM32CubeProgrammer for get certificate via Bootloader interface .. 50
  4.3 Secure programming using JTAG/SWD interface .......................... 51
    4.3.1 SFI programming using JTAG/SWD flow .................................. 51
    4.3.2 SMI programming through JTAG/SWD flow ............................ 52
    4.3.3 STM32CubeProgrammer for secure programming using JTAG/SWD ....... 54

5 Example SFI programming scenario ........................................ 55
  5.1 Scenario overview .................................................. 55
  5.2 Hardware and software environment .................................... 55
  5.3 Step-by-step execution .............................................. 55
    5.3.1 Build OEM application ............................................ 55
    5.3.2 Perform the SFI generation (GUI mode) .............................. 55
    5.3.3 Performing HSM programming for license generation using STPC (GUI mode) ...................... 57
    5.3.4 Programming input conditions .................................... 58
    5.3.5 Perform the SFI install using STM32CubeProgrammer ........................... 58
      Using JTAG/SWD .................................................... 58
## Contents

6 Example SMI programming scenario .......................... 61
  6.1 Scenario overview ........................................... 61
  6.2 Hardware and software environment ....................... 61
  6.3 Step-by-step execution ..................................... 61
    6.3.1 Build 3rd party Library ............................... 61
    6.3.2 Perform the SMI generation ............................ 62
    6.3.3 Programming input conditions ........................... 63
    6.3.4 Perform the SMI install ................................ 63
      Using JTAG/SWD ............................................. 63
    6.3.5 How to test for SMI install success ................... 65

7 Example combined SFI-SMI programming scenario ............ 67
  7.1 Scenario overview .......................................... 67
  7.2 Hardware and software environment ....................... 67
  7.3 Step-by-step execution ..................................... 67
    7.3.1 Using JTAG/SWD ....................................... 69
    7.3.2 How to test the combined SFI install success ........ 71

8 Reference documents .......................................... 73

9 Revision history .................................................. 74
List of tables

Table 1. List of abbreviations ................................................................. 8
Table 2. Document references .............................................................. 73
Table 3. Document revision history ......................................................... 74
List of figures

Figure 38. SMI programming by JTAG flow overview (monolithic SFI image example) .......... 52
Figure 37. SFI programming by JTAG/SWD flow overview (monolithic SFI image example) .... 51
Figure 36. Example of getcertificate command execution using UART interface ........... 50
Figure 35. Secure programming via STM32CubeProgrammer overview on STM32L4 devices ... 49
Figure 34. Secure programming via STM32CubeProgrammer overview on STM32H7 devices ... 48
Figure 33. SMI generation example .................................................. 31
Figure 32. HSM programming GUI in the STPC tool .................................. 46
Figure 31. Option bytes file example ................................................. 31
Figure 30. Log example ................................................................. 42
Figure 29. Settings icon and Settings dialog box ................................... 41
Figure 28. SMI successful generation in GUI mode example ......................... 38
Figure 27. SMI generation Tab .......................................................... 34
Figure 26. SFI successful generation in GUI mode example ........................ 29
Figure 25. Firmware parsing example ............................................... 35
Figure 24. SFI generation Tab ........................................................... 33
Figure 23. SMI generation example .................................................... 31
Figure 22. SFI generation example using an Elf file ................................ 31
Figure 21. STM32TrustedPackageCreator tool - available commands ............... 29
Figure 20. STM32TrustedPackageCreator tool - available commands ............... 29
Figure 19. SMI format layout ............................................................ 24
Figure 18. SMI image generation process ............................................. 27
Figure 17. SMI preparation mechanism ................................................ 26
Figure 16. SFI image layout in case of split ........................................... 25
Figure 15. SFI format layout ............................................................. 24
Figure 14. Error message when a SFI area address is not located in Flash memory .... 23
Figure 13. Error message when SMI address overlaps with a firmware area address .... 21
Figure 12. Error message when firmware files with address overlaps used ............ 21
Figure 11. 'P' and 'R' area specifics versus a regular SFI area ........................ 20
Figure 10. RAM size and CT address inputs used for SFI multi install ................. 19
Figure 9. SFI image process generation ................................................. 18
Figure 8. SFI preparation mechanism .................................................. 17
Figure 7. app.icf file ........................................................................ 16
Figure 6. How to exclude the “lib.o” file from build .................................. 15
Figure 5. Postbuild batch file ............................................................. 14
Figure 4. Setting post-build option ...................................................... 13
Figure 3. Linker extra options .............................................................. 12
Figure 2. Update compiler extra options .............................................. 11
Figure 1. IAR example project overview ............................................... 10

List of figures AN5054
AN5054

List of figures

Figure 47. GUI of STPC during combined SFI-SMI generation ................. 68
Figure 48. Combined SFI-SMI programming success using debug connection .......... 70
Figure 49. Option bytes after combined SFI-SMI install success ......................... 72
1 General information

1.1 Licensing information

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

1.2 Acronyms and abbreviations

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced encryption standard</td>
</tr>
<tr>
<td>CLI</td>
<td>Command line interface</td>
</tr>
<tr>
<td>CM</td>
<td>Contract manufacturer</td>
</tr>
<tr>
<td>GCM</td>
<td>Galois counter mode (one of the modes of AES)</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware security model</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>MAC</td>
<td>Message authentication code</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller unit</td>
</tr>
<tr>
<td>OEM</td>
<td>Original equipment manufacturer</td>
</tr>
<tr>
<td>PCROP</td>
<td>Proprietary code read-out protection</td>
</tr>
<tr>
<td>PI</td>
<td>Position independent</td>
</tr>
<tr>
<td>ROP</td>
<td>Read-out protection</td>
</tr>
<tr>
<td>RSS</td>
<td>Root security service (secure)</td>
</tr>
<tr>
<td>SFI</td>
<td>Secure firmware install</td>
</tr>
<tr>
<td>STPC</td>
<td>STM32TrustedPackageCreator</td>
</tr>
<tr>
<td>SMI</td>
<td>Secure modules install</td>
</tr>
<tr>
<td>STM32</td>
<td>ST family of 32-bit ARM based microcontrollers</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>XO</td>
<td>Execute only</td>
</tr>
</tbody>
</table>

(a) Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
2 How to generate an execute-only/position independent library for SMI preparation

This section describes the requirements and procedures for the preparation of an execute-only (XO) and position independent (PI) library using a partner toolchain. These kinds of libraries serve in encrypted SMI-module generation.

2.1 Requirements

SMI modules run in Execute Only (XO) areas, also called PCROP areas, and must be relocatable to be linkable with final OEM application. Nevertheless, today, 3rd party toolchains for STM32 devices (such as MDK-ARM for ARM, EWARM for IAR and GCC based IDEs) do not allow both features to be activated at the same time. So, starting from particular versions of 3rd party toolchains, the two features below are possible for SMI support:

- Position independent support (code + rw data + ro data)
- No literal pool generation; needed for the PCROP feature.

2.2 Toolchains allowing SMI generation

Three toolchains allow SMI generation:

- EWARM
  Version 7.42.0 allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: "--ropi_cb" + "rwpi" + "--no_literal_pool".
  - "--ropi_cb" + "rwpi" are needed for position independent support
  - option "no_literal_pool" is needed for the PCROP feature.

- MDK-ARM
  The customized version allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: "-fropi-cb", "-frwpi", "-mexecute-only".
  - "fropi-cb" is needed for ro data independent position
  - "frwpi" is needed for rw data independent position
  - option "mexecute-only" is needed for PCROP feature.

All library symbols being used in the final application should be added to the final project in a .txt file format.

- GCC
  The customized version of GCC based toolchains allows execute-only (XO) and position independent (PI) library generation for SMI support through the following options: "-masset".
  Option "-masset" has the same role as "--ropi --ropi_cb --rwpi --no_literal_pool" options used for EWARM toolchain.
2.3 **Execute-only/position independent library scenario example under EWARM**

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

2.3.1 **Relocatable library preparation steps**

1. Open the project available in the “Example” folder: double click on “Example/AdvEx.eww”.
   
   The project architecture is illustrated in *Figure 1*.

---

*Figure 1. IAR example project overview*
The following steps update the old “lib.o” linked to the example application by making it support both PI and XO features:

2. Within Lib-Debug options -> C/C++ Compiler: go to tab “Extra Options” and add the following line:
   “--ropi_cb”
   
   This action is illustrated in Figure 2.

Figure 2. Update compiler extra options
3. Within Lib-Debug options -> Linker: go to the “Extra Options” tab and add the following lines:
   --no_literal_pool
   --ropi_cb
   --loadable
   --no_entry
   This action is illustrated in Figure 3.
   – “ropi_cb” is needed for Position Independent support
   – the “no_entry” is a linker option that sets the entry point field to zero.

Figure 3. Linker extra options
4. Within Lib-Debug options -> Build actions: in post build command line execute the batch file "postbuild.bat" by inserting, if it is not already configured, the following command line:

```
$PROJ_DIR$\postbuild.bat  "$TOOLKIT_DIR$"  "$TARGET_PATH$"  "$PROJ_DIR$\lib.o"
```

This action is illustrated in Figure 4.

Figure 4. Setting post-build option
The postbuild.bat file is used to perform some key actions:

- **--wrap**: adds veneers to library functions to initialize registers used for ropi code
- **“iexe2obj.exe”**: transforms the elf into a linkable object file.

See Figure 5.

**Figure 5. Postbuild batch file**

```
@echo off
REM This is a simple script that creates and object file (%3) from an image (%2)
REM using the tools in (%1).
REM Make sure the old files are deleted before we try to generate the new ones
if exist %9.tmp {
    del %9.tmp
}
if exist %3 {
    del %3
}
echo Do magic encryption here (copy is just a placeholder)
copy %2 %2.tmp
REM This is the list of functions that will have a wrapper generated
STM_WRAP--wrap Tostring --wrap setup_memory --wrap setup_memory2
REM convert the image to a linkable object file using Lib as prefix
REM and keeping all node symbols (holos a bit with debugging)
\bin\iexe2obj.exe --prefix Lib --keep node symbols & \WRAPS %2.tmp %3
```

5. Rebuild the project “Lib”

### 2.3.2 Relocatable SMI module preparation steps

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

To execute this step, follow the steps explained for SMI generation under section “Section 3.4.2: Steps for SMI generation(CLI)”. 
2.3.3 Application execution Scenario

1. Flash the already generated SMI relocatable module to address 0x08080000 using STM32Cube Programmer v0.4.0 or newer (see section 4 to perform this action).
2. Link the data clear part, "libr_clear.o", generated from STM32TrustedPackageCreator tool to the final IAR example application instead of the old previously used "lib.o".
3. Exclude "lib.o" from the build (Figure 6).

4. Rebuild the application.
5. Do these modifications in an example application ICF file:
   a) Define region for PCROP block:
      
      define symbol __ICFEDIT_region_PCROP_start__ = 0x08080000;
      define symbol __ICFEDIT_region_PCROP_end__   = 0x0809FFFF;
      define region PCROP_region = mem:[from __ICFEDIT_region_PCROP_start__ to __ICFEDIT_region_PCROP_end__];
   
   b) Define PCROP region as 'noload' (since it is already installed using STM32CubeProgrammer so no need to load it again) :
      "SMI": place noload in PCROP_region { ro code section __code__Lib};
These modifications are illustrated within the “app.icf” file, which is shown in Figure 5.

6. To check that example application executed successfully on the STM32H7 device:
   a) check that address 0x08080000 was protected with PCROP.
   b) You should see the expected “printf” packets in the terminal output.
3 Encrypted firmware(SFI)/module(SMI) preparation using STM32TrustedPackageCreator

The STM32TrustedPackageCreator tool allows the generation of SFI and SMI images for STM32H7 devices. It is available in both CLI and GUI modes free of charge from www.st.com.

3.1 System requirements

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

3.2 SFI generation process

The SFI format is an encryption format for Firmware created by STMicroelectronics that transforms firmware (in Elf, Hex, Bin or Srec formats) into encrypted and authenticated firmware in SFI format using AES-GCM algorithm with a 128-bit key. The SFI preparation process used in the STM32TrustedPackageCreator tool is described in Figure 8.

**Figure 8. SFI preparation mechanism**
The SFI generation steps as currently implemented in the tool are described in Figure 9.

**Figure 9. SFI image process generation**

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

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

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

These encrypted parts appended to their corresponding descriptors (the unencrypted descriptive header generated by the tool) are called areas.
These areas can be of different types:

- ‘F’ for a firmware area (a regular segment in the input firmware)
- ‘M’ for a module area (used in SFI-SMI combined-image generation, and corresponds to input from an SMI module)
- ‘C’ for a configuration area (used for option-byte configuration)
- ‘P’ for a “pause” area
- ‘R’ for a “resume” area.

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

The STM32TrustedPackageCreator overview below (Figure 12) shows the RAM size input for SFI image generation, and also the ‘Continuation token address’ input, which is used by SFI multi install to store states in Flash memory during SFI programming.

**Figure 10. RAM size and CT address inputs used for SFI multi install**
Figure 11 (below) shows the specifics of these new areas compared to a regular SFI area.

**Figure 11. ‘P’ and ‘R’ area specifics versus a regular SFI area**

<table>
<thead>
<tr>
<th>Area format</th>
<th>New Pause Area</th>
<th>New Resume Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type (‘F’, ‘M’, ‘C’)</td>
<td>Type ‘P’</td>
<td>Type ‘R’</td>
</tr>
<tr>
<td>Version</td>
<td>Version</td>
<td>Version</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>Size</td>
<td>Size = 0</td>
<td>Size = 0</td>
</tr>
<tr>
<td>Address</td>
<td>Address of CT</td>
<td>Address of CT</td>
</tr>
<tr>
<td>Total Nb of areas</td>
<td>Total Nb of areas</td>
<td>Total Nb of areas</td>
</tr>
<tr>
<td>Tag</td>
<td>Tag</td>
<td>Tag</td>
</tr>
</tbody>
</table>

Encrypted Area Content
- Firmware
- Module
- Configuration

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

An authentication tag is generated as output.

**Note:** To prepare an SFI image from multiple firmware files you have to make sure that there is no overlap between their segments, otherwise an error message appears (Figure 12).
Figure 12. Error message when firmware files with address overlaps used
For combined SFI-SMI images, there is also an overlap check between firmware and module areas. If the check fails, an error message appears (Figure 13).

**Figure 13. Error message when SMI address overlaps with a firmware area address**
Also, all SFI areas must be located in Flash memory, otherwise the generation fails and you see the following error message (Figure 14).

Figure 14. Error message when a SFI area address is not located in Flash memory
The final output from this generation process is a single file, which is the encrypted and authenticated firmware in "sfi" format. The SFI format layout is described in Figure 15.

Figure 15. SFI format layout
When the SFI image is split during generation, areas 'P' and 'R' appear in the SFI image layout, as in the example below *Figure 16*.

*Figure 16. SFI image layout in case of split*
3.3 SMI generation process

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

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

Figure 17. SMI preparation mechanism
The SMI generation steps as currently implemented in the tool are described in the diagram below (Figure 18).

**Figure 18. SMI image generation process**

The AES-GCM encryption is performed using the following inputs:
- 128-bit AES encryption key
- The input nonce as Initialization Vector (IV)
- The security version as Additional Authenticated Data (AAD).
Before SMI image creation, PCROP checks are performed on the SMI image validity:

- A PCROP section must be aligned on a Flash word (256 bits), otherwise a warning is shown.
- The section’s size must be at least 2 Flash words (512 bits), otherwise a warning is shown.
- The section must end on a Flash word boundary (a 256-bit word), otherwise a warning is shown.
- If the start address of the section immediately following the PCROP section overlaps the last Flash word of the PCROP section (after performing the PCROP alignment constraint), the generation fails and an error message appears.

If everything is OK, two outputs are created under the specified path:

- the SMI image (*Figure 19* represents the SMI format layout)
- the library data part.

*Figure 19. SMI format layout*
3.4 STM32TrustedPackageCreator tool in the command line interface

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

The available commands are listed in Figure 20.

**Figure 20. STM32TrustedPackageCreator tool - available commands**

<table>
<thead>
<tr>
<th>SPI preparation options</th>
<th>SMI preparation options</th>
</tr>
</thead>
<tbody>
<tr>
<td>-sfi, --sfi</td>
<td>-smi, --smi</td>
</tr>
<tr>
<td></td>
<td>You also need to provide the information listed below</td>
</tr>
<tr>
<td>-fir, --firmware</td>
<td>-elf, --elfFile</td>
</tr>
<tr>
<td>(&lt;Firm_File&gt;)</td>
<td>Input ELF file</td>
</tr>
<tr>
<td>(&lt;Address&gt;)</td>
<td></td>
</tr>
<tr>
<td>-k, --key</td>
<td>-s, --sec</td>
</tr>
<tr>
<td>(&lt;Key_File&gt;)</td>
<td>Section to be encrypted</td>
</tr>
<tr>
<td>-n, --nonce</td>
<td>-k, --key</td>
</tr>
<tr>
<td>(&lt;Nonce_File&gt;)</td>
<td>AES-GCM encryption key</td>
</tr>
<tr>
<td>-v, --ver</td>
<td>-n, --nonce</td>
</tr>
<tr>
<td>(&lt;Image_Version&gt;)</td>
<td>AES-GCM nonce</td>
</tr>
<tr>
<td>-oh, -- outputFile</td>
<td>-v, --ver</td>
</tr>
<tr>
<td>(&lt;SFM_File&gt;)</td>
<td>Image version</td>
</tr>
<tr>
<td>-n, --nmodule</td>
<td>-oh, -- outputFile</td>
</tr>
<tr>
<td>(&lt;SFI_File&gt;)</td>
<td>Option hosts configuration file</td>
</tr>
<tr>
<td>(&lt;Address&gt;)</td>
<td>-n, --nmodule</td>
</tr>
<tr>
<td>-rs, --rsize</td>
<td>(&lt;SFI_File&gt;)</td>
</tr>
<tr>
<td>(&lt;Size&gt;)</td>
<td>Only in case of a relocatable SMI (with Address - 0)</td>
</tr>
<tr>
<td>-ct, --token</td>
<td>-rs, --rsize</td>
</tr>
<tr>
<td>(&lt;Address&gt;)</td>
<td>define available run size (for multi-image)</td>
</tr>
<tr>
<td>-o, --outfile</td>
<td>-ct, --token</td>
</tr>
<tr>
<td>(&lt;Output_File&gt;)</td>
<td>Address</td>
</tr>
<tr>
<td></td>
<td>-o, --outfile</td>
</tr>
<tr>
<td></td>
<td>Generated SPI file</td>
</tr>
<tr>
<td></td>
<td>-o, --outfile</td>
</tr>
<tr>
<td></td>
<td>SPI File to be created</td>
</tr>
<tr>
<td></td>
<td>-clear</td>
</tr>
<tr>
<td></td>
<td>Clear ELF file</td>
</tr>
<tr>
<td></td>
<td>-clear</td>
</tr>
<tr>
<td></td>
<td>Clear ELF file to be generated</td>
</tr>
</tbody>
</table>
3.4.1 Steps for SFI generation (CLI)

In order to generate an SFI image in CLI mode, the user must use the “-sfi, --sfi” command followed by the appropriate inputs.

Inputs for “sfi” command are:

- fir, --firmware
  Description: adds an input firmware file (supported formats are Bin, Hex, Srec and ELF). This option can be used more than once in order to add multiple firmware files.
  Syntax:     -fir <Firmware_file> [<Address>]
              <Firmware_file>   :Firmware file.
              [<Address>]      :Used only for binary firmwares.

- k, --key
  Description: sets the AES-GCM encryption key.
  Syntax:     -k <Key_file>
              <Key_file>     : A 16 bytes binary file.

- n, --nonce
  Description: sets the AES-GCM nonce.
  Syntax:     -n <Nonce_file>
              <Nonce_file>   : A 12-byte binary file.

- v, --ver
  Description: sets the image version.
  Syntax:     -v <Image_version>
              <Image_version> : A value between 0 and 255 in any base.

- ob, --obfile
  Description: provides an option bytes configuration file.
The option bytes file field is only mandatory for SFI applications (first install) to allow option bytes programming, otherwise it is optional.
  Only CSV (Comma Separated Value) file format is supported as input for this field, it is composed from two vectors: register name and register value respectively.
  Example: for STM32H7xx devices, 9 option bytes registers must be configured, which corresponds to a total of 9 lines in the csv file (Figure 21).
  Syntax:     -ob <CSV_file>
              <CSV_file>     : A csv file with 9 values.
-m, --module
Description: adds an input SMI file.
This option can be used more than once in order to add multiple SMI files.
This is optional (used only for combined SFI-SMI).
Syntax: -m <SMI_file>
<SMI_file > : SMI file.[<Address>] : Address is provided only for relocatable SMI.

-rs, --ramsize
Description: define the available ram size (in case of SFI multi-install)
Syntax: -rs <Size>
< Size > : RAM available size in bytes

-ct, --token
Description: continuation token address (in case of SFI multi-install)
Syntax: -ct <Address>
< Address > : continuation token Flash address

-o, --outfile
Description: sets the output SFI file to be created.
Syntax:     -o     <out_file>
<out_file > : the SFI file to be generated (must have the “.sfi” extension).

Example of SFI generation command using an ELF file:
```
STM32TrustPackageCreator_CLI.exe -sfi -fir tests.axf -k
test_firmware_key.bin -n nonce.bin -ob ob.csv -v 23 -o out.sfi
```
The result of previous command is shown in Figure 22.

Figure 22. SFI generation example using an Elf file
3.4.2 Steps for SMI generation (CLI)

In order to generate an SMI image in CLI mode, the user must use the “-smi, --smi” command followed by the appropriate inputs.

Inputs for the “smi” command are:

-elf, --elfile

**Description**: sets the input ELF file (only elf format is supported).

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

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

-s, --sec

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

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

*<section_name>*: Section name.

-k, --key

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

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

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

-n, --nonce

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

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

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

-sv, --sver

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

The security version file is used to make the SMI image under preparation compatible with a given RSS version, since it contains a corresponding identifying code (almost the HASH of the RSS).

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

*<SV_file>*: A 16-byte file.

-o, --outfile

**Description**: Sets the SMI file to be created as output.

**Syntax**: `-o <out_file>`

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

-c, --clear

**Description**: Sets the clear ELF file to be created as output corresponding to the data part of the input file.

**Syntax**: `-c <ELF_file>`
<ELF_file> : Clear ELF file to be generated.

Example of SMI generation command:
```
STM32TrustPackageCreator_CLI.exe  -smi -elf FIR_module.axf  -s "ER_PCROP"  -k test_firmware_key.bin  -n nonce.bin  -sv svFile  -o test.smi  -c clear.smi
```

Figure 23. SMI generation example
3.5 Using the STM32TrustedPackageCreator tool graphical user interface

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

3.5.1 SFI generation using STPC in GUI mode

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

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

- Firmwares files:
  The user needs to add the input firmware files with the “Add” button.
  If the file is valid, it is appended to the “input firmware files” list, otherwise an error message box appears notifying the user that either the file could not be opened, or the file is not valid.
  Clicking on “input firmware file” causes information related information to appear in the “Firmware information” section (Figure 25).

Figure 25. Firmware parsing example
Encrypted firmware(SFI)/module(SMI) preparation using STM32TrustedPackageCreator  AN5054

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

- **Option bytes file:**
  The option bytes file can be selected the same way as the encryption key and nonce. Only csv files are supported.

- **SMI files:**
  SMI files can be added the same way as the firmware files. Selecting a file causes related information to appear in the “Firmware information” section.

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

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

*Note:* By using the “Select folder” button, the name “out.sfi” is automatically suggested, you can keep this or change it.

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

  If everything goes well, a message box indicating successful generation appears (*Figure 26*) and information about the generated SFI file is displayed in the SFI information section.
Figure 26. SFI successful generation in GUI mode example
3.5.2 SMI generation using STPC in GUI mode

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

*Figure 27. SMI generation Tab*

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

- Elf file:
  In this case the input file can be only an elf file.
  If the file is valid, information is displayed in the “ELF information” tab, otherwise an error message box appears notifying the user that either the file could not be opened or the file is not valid.

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

- Security version file:
  The security version file is used for the same purpose as explained in the CLI section. The security version file size must be 16 bytes.

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

- output files:
  Sets the folder path into which the SMI image and its clear part are to be created. This can be done by entering the folder path (absolute or relative) or by using the “Select folder” button.

Note: For both output fields, when using the “Select folder” button, a name is suggested automatically, you can keep this or change it.

- ‘Generate SMI’ button:
  When all fields are filled in properly the ‘Generate SMI’ button is enabled, and the user can generate the SMI file and its corresponding clear data part by a single click on it.
  A message box informing the user that generation was successful must appear (Figure 28), with additional information about the generated SMI file displayed into the “SMI information” section. In the case of any invalid input data, an error message box appears instead.
3.5.3 Settings

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

Settings can be performed on:

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

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

- **Log file:**
  When checked, a log file is generated in the selected path.
3.5.4 Log generation

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

*Figure 30* shows a log example:

*Figure 30. Log example*
3.5.5 **SFI and SMI file checking function**

This function checks the validity and information parsing of an SFI or SMI file. It can be accessed by clicking the Check SFI/SMI button in the tool bar or the menu bar: File -> Check SFI/SMI.

*Figure 31* shows a check SFI example:

*Figure 31. Check SFI file example*
4 Encrypted firmware(SFI)/module(SMI) programming using STM32CubeProgrammer

STM32CubeProgrammer is a tool for programming STM32 devices through UART, USB, SPI, CAN, I2C, JTAG and SWD interfaces. So far, programming via JTAG/SWD is only supported with ST-LINK probe.

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

The tool is currently available only in CLI mode, it is available free of charge from www.st.com.

4.1 Chip certificate authenticity check and license mechanism

The SFI solution was implemented to provide a practical level of IP protection chain from the firmware development up to Flashing the device, and to attain this objective, security assets are used, specifically device authentication and license mechanisms.

4.1.1 Device authentication

The device authentication is guaranteed by the device’s own key.

In fact, a certificate is related to the device’s public key and is used to authenticate this public key in an asymmetric transfer: the certificate is the public key signed by a Certificate Authority (CA) private key. (This CA is considered as fully trusted).

This asset is used to counteract usurpation by any attacker who could substitute the public key with their own key.

4.1.2 License mechanism

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

Using this license mechanism the OEM is able to count each install for a given piece of firmware, since each license is specific to a unique chip, identified by its public key.

Licenses mechanism general scheme

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

When a customer wants to download the firmware to a chip, they send a chip identifier to the provider server, HSM or any provider license generator tool, which returns a license for the identified chip. The license contains the encrypted firmware key, and only this chip can decrypt it.
License distribution

There are many possible ways for the firmware provider to generate and distribute licenses:

- **Server based**: an internet server can be set up, and when a customer needs to Flash the firmware on to a chip, they connect to the server which generates a license for this chip.
- **HSM based**: Hardware Security Modules can be built, one of which is installed on the programming house production line.
- **Licenses can be generated in advance** (but the firmware provider must know which chips to generate licenses for).

There is no STMicroelectronics secret involved in license generation, so each firmware provider is free to choose their preferred method.

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

HSM programming by OEM for License distribution

When an OEM needs to deliver an HSM to a programming house for deployment as a license generation tool for programming of relevant STM32 devices, some customization of the HSM must first be performed.

The HSM needs to be programmed with all the data needed for the license scheme deployment. In the production line, a dedicated API is available for each piece of data to be programmed in the HSM.

These data are:

- **The counter**: the counter is set to a maximum value that corresponds to the maximum number of licenses that could be delivered by the HSM. It aims to prevent over-programming.
  
  It is decremented with each license delivered by the HSM.
  
  No more licenses are delivered by the HSM once the counter is equal to zero.
  
  The maximum counter value must not exceed a maximum predefined value, which is 16 Ku for HSM version 1.0.

- **The Firmware key**: this key is 32 bytes and is composed of two fields, the initialization vector (IV) (first field) and the key (last field) that were used to AES128-GCM encrypt the firmware.
  
  Both fields are 16 bytes long, but the last 4 bytes of the IV must be zero (only 96 bits of IV are used in the AES128-GCM algorithm).
  
  Both fields must remain secret; that’s why there are encrypted before being sent to the chip.
  
  The key and IV remains the same for all licenses for a given piece of firmware. However, they must be different for different firmware or different versions of the same firmware.

- **The Firmware identifier**: allows the correct HSM to be identified for a given firmware.

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

ST provides the tools needed to support SFI via HSM. In fact, HSM programming is supported by the STM32TrustedPackageCreator tool. Figure 32 shows the GUI for HSM programming in STPC tool.
During SFI install, STM32CubeProgrammer communicates with the device to get the chip certificate, upload it into the HSM to request the license. Once the license is generated by the HSM, it gives it back to the STM32 device. This process is illustrated in Figure 33.
4.2 Secure programming using bootloader interface

4.2.1 Secure firmware installation using Bootloader interface flow

The production equipment on the OEM-CM production line needs to be equipped with a Flashing Tool (FT) supporting the programming of SFI images. The Flashing tool to be used on OEM-CM production line is STM32CubeProgrammer, which is given the data blob prepared by the STPC, containing the image header and the encrypted image data blob.

Note: The SFI install is performed successfully only if a valid license is given to the Flashing tool.

STM32CubeProgrammer supports secure firmware install for STM32H753xI and STM32L451CE specific part number so far.

For STM32H753xI devices, one important outcome is that RSS fully manages the installation (no secure bootloader) and SFI is supported for USART, SPI and USB interfaces for those devices. Otherwise for STM32L451CE specific part number devices the installation is performed through a secure bootloader via USART or SPI interfaces only.

For more details on SFI over these STM32 devices refer to AN4992 [1]. This document is available on www.st.com.
The general flow of the Secure Firmware Installation using bootloader interface on a chip for H7 and L4 secure devices is shown respectively in **Figure 33** and **Figure 34** below.

**Figure 34. Secure programming via STM32CubeProgrammer overview on STM32H7 devices**

**Figure 35. Secure programming via STM32CubeProgrammer overview on STM32L4 devices**
4.2.2 Secure Module installation using bootloader interface flow

As explained in Section 3.3: SMI generation process, outputs are generated for this particular use case:

- The first part, not encrypted: this is a regular ELF/AXF file, containing all the sections except the code section extracted by the STPC to prepare the SMI module
- The encrypted SMI module, which contains the protected code.

The first part can be programmed into the chip using any means (JTAG Flasher, UART Bootloader and so on, as for any regular ELF/AXF file.

The full content of the SMI image file and its corresponding license are given to STM32CubeProgrammer which places them in RAM.

The RSS_SMI_resetAndInstallModules() function is then invoked through the start_smi() secure bootloader command with the following parameters:

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

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

Note: The SMI install is performed successfully only if the adequate license is given to the Flashing tool.

4.2.3 STM32CubeProgrammer for SFI using bootloader interface

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

-sfi, --sfi

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

protocol=<Ptype >: Protocol type to be used: static/live (only static protocol is currently supported).

$file_path$: Path of SFI file to be programmed.

$licenseFile_path$: Path to the license file of the smi to be programmed.

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

Example using UART bootloader interface:

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

This command allows secure installation of firmware “data.sfi” into a dedicated Flash memory address.
4.2.4 **STM32CubeProgrammer for SMI via Bootloader interface**

For SMI programming, STM32CubeProgrammer is used in CLI mode by launching the following command:

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

**Syntax**

```
-smi protocol=\<Ptype\> \<file_path> [\<address\>] \<license_file_path>
```

- `protocol=\<Ptype\>`: Protocol type to be used: static/live (only static protocol is supported so far)
- `\<file_path\>`: Path of SMI file to be programmed.
- `[\<address\>]`: Destination address of the SMI module (only needed when relocatable).
- `\<license_file_path\>`: Path to the license file of the SMI to be programmed.

Example using UART bootloader interface:

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

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

4.2.5 **STM32CubeProgrammer for get certificate via Bootloader interface**

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

```
-gc, --getcertificate
```

**Syntax**

```
-gc \<file_path>
```

Example using UART bootloader interface:

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

This command allows the chip Certificate to be read and uploaded into the specified file: “C:\Demo_certificate.bin”

The execution results are shown in **Figure 36**.

---

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

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

4.3.1 SFI programming using JTAG/SWD flow

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

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

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

SFI via debug interface is currently supported only for STM32H753l devices.

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

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

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

Other limitations are that security must be left activated in the configuration area if there is a PCROP area.
The SFI flow for programming through JTAG is described in Figure 37.

Figure 37. SFI programming by JTAG/SWD flow overview (monolithic SFI image example)

4.3.2 SMI programming through JTAG/SWD flow

For SMI programming through JTAG/SWD the process flow is similar to that using the UART bootloader. In fact, RSS fully manages the installation in both cases.

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

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

Once the RSS function "RSS_SMI_resetAndInstallModules" execution has finished successfully, the SMI module is decrypted, Flashed and protected by the PCROP mechanism.

The essential steps of the SMI programming by JTAG flow are described in Figure 38.
Figure 38. SMI programming by JTAG flow overview
4.3.3 STM32CubeProgrammer for secure programming using JTAG/SWD

The only modification in the STM32CubeProgrammer secure command syntax is the connection type which must be set to “jtag” or “swd”, otherwise all secure programming syntax for supported commands is identical.

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

Example of “getcertificate” command using JTAG:

STM32_Programmer.exe –c port=jtag mode=HOTPLUG -gc testJTAG_Certif.bin

The result of this example is shown in Figure 39.

Figure 39. Example of getcertificate command using JTAG

```
Certificate File : testJTAG_Certif.bin

Requesting Chip Certificate using debug interface...
Get Certificate done successfully
...Writing data to file testJTAG_Certif.bin
Writing chip certificate to file testJTAG_Certif.bin finished successfully
Time elapsed during the getcertificate operation is: 00:00:00.032
```

Example of “smi” command using SWD

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

5.1 Scenario overview

The actual user application to be installed on the STM32H753xI device makes “printf” packets appear in serial terminals.

The application was encrypted using the STPC.

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

5.2 Hardware and software environment

For successful SFI programming, some HW and SW prerequisites are needed:

- STM32H743I-EVAL board
- STM32H753xI with Bootloader v13.2-RC2 and RSS v0.9 programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running on either Windows 7 or Ubuntu 14 in both 32-bit and 64-bit versions
- STM32TrustPackageCreator v0.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v0.4.0 (or greater) package available from www.st.com.

5.3 Step-by-step execution

5.3.1 Build OEM application

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

5.3.2 Perform the SFI generation (GUI mode)

To be encrypted with the STM32TrustPackageCreator Tool, OEM firmware is provided in Axf format in addition to a csv file to set the option bytes configuration. A 128-bit AES encryption key and a 96-bit nonce are also provided to the tool. They are available in the SFI_ImagePreparation directory.

A “.sfi” image is then generated (out.sfi).

Figure 40 shows the STPC GUI during SFI generation.
Figure 40. STPC GUI during SFI generation
5.3.3 Performing HSM programming for license generation using STPC (GUI mode)

The OEM must provide a license generation tool to the programming house to be used for license generation during the SFI install process.

In this example, HSMs are used as license generation tools in the field. See Section 4.1.2: License mechanism for HSM use and programming.

Figure 41 shows an example for HSM programming by OEM to be used for SFI install. The maximum number of licenses delivered by the HSM in this example is 1000.

Figure 41. Example of HSM programming using STPC GUI

Note: When programming the HSM for real in-the-field use (here it is just an example scenario), the field “Set HSM to operational state” must be checked to lock the HSM before it is shipped to the programming house (untrusted environment).
5.3.4 Programming input conditions

Before performing an SFI install be sure that:

- Flash memory is erased.
- No PCROPed zone is active, otherwise destroy it.
- The chip must support security (a security bit must be present in the option bytes)
- When using a UART interface the User security bit in option bytes must be enabled before launching the SFI command. For this, the following STM32CubeProgrammer command can be launched:
  - Launch the following command (Uart Bootloader used => Boot0 pin set to VDD):
    \[-c port=COM9 -ob SECURITY=1\]
- When using a UART interface the Boot0 pin must be set to VSS:
  - After enabling security (boot0 pin set to VDD), a power off/power on is needed when switching the Boot0 pin from VDD to VSS: power off, switch pin then power on.
- When performing an SFI install using UART BL then, no debug interface must be connected to any USB host, if there is a debug interface that was used/connected, then, disconnect it then perform a power off/power on before launching the SFI install to avoid any debug intrusion problem.
- Boot0 pin set to VDD When using a debug interface.
- You have a valid license at your disposal, generated for the currently used chip, or a license generation tool to generate the license during SFI install (HSM).

5.3.5 Perform the SFI install using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with ST HSMs (Hardware Secure Modules based on smart card) to generate a license for the connected STM32 device during SFI install.

Using JTAG/SWD

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

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static "<local_path>/out.sfi" hsm=1 slot=<slot_id>
```
Figure 42 shows the SFI install via SWD execution and the HSM as license generation tool in the field.

Figure 42. SFI install success using SWD connection (1)
Figure 43. SFI install success using SWD connection (2)

```
RSS command execution OK
Reconnecting...
ST-LINK SW: 0672FFE54949677067834031
ST-LINK Firmware version: V2J33M19
Target voltage: 3.3V
Error: ST-LINK error <DEU_NO_DEVICE>
---retrying---
ST-LINK SW: 0672FFE54949677067834031
ST-LINK Firmware version: V2J33M19
Target voltage: 3.3V
SFD Frequency: 4000 kHz
Connection mode: Hot Plug
Device ID: 0x450
Reconnected!

Requesting security state...
SECURITY State Success
SFI SUCCESS!
SFI file out_EN.sfi Install Operation Success
```
6 Example SMI programming scenario

6.1 Scenario overview

In this scenario, the 3rd party's library to be installed on the STM32H753xl device makes "printf" packets appear in the serial terminal if the library code execution called by the application does not crash.

The library code was encrypted using the STPC.

The OEM provides tools to the CM to get the appropriate license for the concerned SMI module.

6.2 Hardware and software environment

The same environment as explained in Section 4.1.1: Device authentication.

6.3 Step-by-step execution

6.3.1 Build 3rd party Library

ST or 3rd party developers can use any IDE to build the library to be encrypted and installed into the STM32H7 device.

In this scenario the SMI module based on the built library is not relocatable. The destination address is hardcoded in SMI module to the following value: 0x08080000.
6.3.2 Perform the SMI generation

For encryption with the STM32TrustedPackageCreator Tool, the 3rd party module is provided in Elf format. A 128-bit AES encryption key, a 96-bit nonce and a security version file are also provided to the tool. They are available in the SMI_ImagePreparation directory. After choosing the name of the section to be encrypted, a ".smi" image is then generated (FIR_module.smi).

The clear data part of the library without the encrypted section is also created in Elf format (FIR_module_clear.axf).

Figure 37 shows the STPC GUI during SMI generation.

Figure 44. STPC GUI during SMI generation
6.3.3 Programming input conditions

Before performing the SMI install be sure that:

- The SMI module destination address is not already PCROPed, otherwise destroy this PCROPed area.
- The Boot0 pin set to VDD.
- The chip supports security (existing security bit in option bytes).
- When performing SMI install using UART BL, no debug interface is connected to any USB host. If there is a debug interface that was used/connected, disconnect it then perform a power off/power on before launching the SMI install to avoid any debug intrusion problem.
- You have the proper license generated for the currently used chip, or an HSM or secure server to generate it during SMI programming.

6.3.4 Perform the SMI install

Using JTAG/SWD

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

```cmd
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -smi protocol=static "<local_path>/FIR_module.smi" "<local_path>/licenseSMI.bin"
```

This command allows the SMI specified file "FIR_module.smi" to be programmed into a dedicated PCROPed area at address (0x08080000).

*Figure 45* shows the SMI install via SWD execution:
Figure 45. SMI install success via debug interface

```
ST-LINK Firmware version : V2.32.7M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x458

Protocol : static
SMI File : C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7\Release\AN\Nu_package\stm32_programmer_package_v0.4.0\bin

Starting SMI install operation for file : C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7\Release\AN\Nu_package\STM32H7_install\stm32cubeprogrammer\bin\stm32cubeprogrammer_v8.4.0.exe

SMI File Information :

SMI file path : C:\Users\hannachi\Documents\Projects\STM32H7\project\docs\docs_forSTM32H7\Release\AN\Nu_package\STM32H7_install
SMI license file path : C:\Users\hannachi\Documents\Projects\STM32H7\project\docs\docs_forSTM32H7\Release\AN\Nu_package\STM32H7_install
SMI code destination address : 0x24050000
SMI code size : 1688

Setting write mode to SMI
Success to set write mode for SMI
Writing license & address 0x24050000...
Writing SMI module image to address 0x24050008...

SMI image successfully written at address 0x24050000
Starting SMI process with license @ 0x24050000 and Image @ 0x24050008...

HS process started...
Waiting for execution...
Reconnecting...
ST-LINK Firmware version : V2.32.7M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x458
Reconnected

Requesting security state...
SECURITY State Success

SMI file C:\Users\hannachi\Documents\Projects\STM32H7_project\docs\docs_forSTM32H7\Release\AN\Nu_package\stm32_programmer_package_v0.4.0\bin

Time elapsed during the SMI install operation is: 08:00:03.204
```
6.3.5 How to test for SMI install success

1. Flash the clear data part “FIR_module_clear.hex” (available under the Tests directory) into address 0x08084000 using STM32Cubeprogrammer or any other Flashing tool.

2. Flash the test application “tests.hex” (which is based on the SMI module), available under the Tests directory at start user Flash address “0x08000000” using STM32Cubeprogrammer or any other Flashing tool.

The option bytes configuration becomes as below (Figure 46).
Figure 46. OB display command showing that a PCROP zone was activated after SMI

```
OPTION_BYTES BANK: @

Read Out Protection:
RDPE: 0x0A (Level 0, no protection)

NS1:
NSS: 0x0 (No SPI process on going)

BOR Level:
BOR_LEV: 0x0 (reset level is set to 2.1 V)

Hardware:

User Configuration:

IWMCI: 0x1 (Independent watchdog is controlled by hardware)

NWRST_STOP_D1: 0x1 (STOP node on Domain 1 is entering without reset)

NWRST_STBY_D1: 0x1 (STANDBY node on Domain 1 is entering without reset)

PE banks:
PE_STOP: 0x1 (Independent watchdog is running in STOP mode)

PE_STBY: 0x1 (Independent watchdog is running in STANDBY mode)

SECURITY: 0x1 (Security feature enabled)

HCM7: 0x1 (CM-7 boot enabled)

BWRST_STOP_D2: 0x1 (STOP node on Domain 2 is entering without reset)

BWRST_STBY_D2: 0x1 (STANDBY node on Domain 2 is entering without reset)

SWAP_BANK: 0x0 (after boot loading, no swap for user sectors)

DMAP: 0x1 (delete PCHP protection and erase protected area)

DMERA: 0x1 (delete Secure protection and erase protected area)

Boot address Option Bytes:

BOOT_CM0 ADD: 0x000 <0x00000000>

BOOT_CM1 ADD: 0x1FF0 <0x1FF00000000>

PCROP Protection:

PCROP_start: 0x000 <0x00000000>

PCROP_end: 0x000 <0x00000000>

Secure Protection:

SECA_start: 0x0FF <0x001000F0>

SECA_end: 0x080 <0x00000000>

NTCM RAM Protection:

ST_RAM_SIZE: 0x02 <0 KB>

Write Protection:

nWRP0: 0x1 (Write protection not active on this sector)

nWRP1: 0x1 (Write protection not active on this sector)

nWRP2: 0x1 (Write protection not active on this sector)

nWRP3: 0x1 (Write protection not active on this sector)

nWRP4: 0x1 (Write protection not active on this sector)

nWRP5: 0x1 (Write protection not active on this sector)

nWRP6: 0x1 (Write protection not active on this sector)

nWRP7: 0x1 (Write protection not active on this sector)
```

3. If a UART connection is available on the board used, open the *Hercule.exe* serial terminal available under the *Tests* directory, open the connection. On reset you should see the dedicated “printf” packets.
7 Example combined SFI-SMI programming scenario

7.1 Scenario overview

The actual user application to be installed on the STM32H753XI device makes "printf" packets appear in the serial terminal.

In this case the OEM application is built based on a third party's library as explained in IAR example (Section 2.3: Execute-only/position independent library scenario example under EWARM.)

The application is encrypted using the STPC, the SMI module corresponding to 3rd party's library code is uploaded as input during combined SFI generation and represented as an area of type 'M' within firmware application areas.

The SFI OEM application firmware could then be uploaded (on an OEM server for example) with all the inputs needed for license generation by the CM.

The OEM provides tools to the CM to get the appropriate licenses for the SFI application concerned and the integrated SMI module(s).

7.2 Hardware and software environment

The same environment as explained in Section 5.2: Hardware and software environment.

7.3 Step-by-step execution

1. Build the OEM application

   OEM application developers may use any IDE to build their firmware as well as using SMI modules provided by STMicroelectronics or 3rd parties for example.

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

2. Perform the SFI generation.

   For encryption with the STM32TrustedPackageCreator Tool, OEM firmware and the clear data part are both provided in hex format (corresponding to the SMI module to be integrated within the SFI image). A csv file to set the option bytes configuration is also necessary. The SMI module used is also provided as an input to the tool, in addition to a 128-bit AES encryption key and a 96-bit nonce. All inputs needed are available in the “SFI_ImagePreparation/Combined” directory. A “.sfi” image is then generated (out_comb.sfi).
3. Programming input conditions are the same as for SFI programming scenario (Section 5.3.4: Programming input conditions).

4. Perform the SFI install using SWD/JTAG or bootloader interface (here SWD interface will be used).
7.3.1 Using JTAG/SWD

Once all input conditions are respected, go to the `stm32_programmer_package_v0.4.1/bin` directory and launch the following command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static "<local_path>/out_comb.sfi" "<local_path>/licenseSFI.bin"
```

Once all input conditions are respected, go to the `<STM32CubeProgrammer_package_path>/bin` directory and launch the following command:

```
STM32_Programmer_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static "<local_path>/out_comb.sfi" "<local_path>/licenseSFI.bin"
```

*Figure 48* shows the combined SFI-SMI install trace success.
Figure 48. Combined SFI-SMI programming success using debug connection

```
ST-LINK Firmware version : V2J26M15
SWD frequency = 40000 KHz
Connection mode: Hot Plug
Device ID: 0x458

Protocol : static
SPI File : RefSPI_MDM/SFI_Combined/out_comb.sfi

Starting SPI install operation for file : RefSPI_MDM/SFI_Combined/out_comb.sfi

SPI File Information :

  SPI file path : RefSPI_MDM/SFI_Combined/out_comb.sfi
  SPI license file path : RefSPI_MDM/SFI_Combined/licenseSFIcomb_h253b9p.bin

SPI header information :
  SPI protocol version : 1
  SPI total number of areas : 4
  SPI image version : 23

Parsing Area 1/4 :
  Area type : F
  Area size : 12288
  Area destination address : 0x00000000

Parsing Area 2/4 :
  Area type : F
  Area size : 1152
  Area destination address : 0x00040000

Parsing Area 3/4 :
  Area type : M
  Area size : 1688
  Area destination address : 0x00800000

Parsing Area 4/4 :
  Area type : C
  Area size : 36
  Area destination address : 0x0

Setting write mode to SPI
Successfully set write mode for SPI
Writing license to address 0x24007800
Writing line header to address 0x24000000
Writing areas and areas wrapper...
RSS process started...

RSS command execution OK
Reconnecting...
ST-LINK Firmware version : V2J26M15
SWD frequency = 40000 KHz
Connection mode: Hot Plug
Device ID: 0x458

Reconnected!

Requesting security state...
SECURITY State Success
SFI SUCCESS!
SFI file RefSPI_MDM/SFI_Combined/out_comb.sfi Install Operation Success

Time elapsed during the SPI install operation is: 00:00:01.056
Press <RETURN> to close this window...
```
7.3.2 How to test the combined SFI install success

The option bytes configuration should be modified as shown in Figure 49.

- 3rd party library module is programmed into a PCROP area
- The SFI image is protected using RDP level1.

If a UART connection is available on the board used, open the Hercule.exe serial terminal available under the Tests directory, open the connection and on reset you should see the dedicated "printf" packets.
Figure 49. Option bytes after combined SFI-SMI install success

```plaintext
OPTION BYTES BANK: 0

Read Out Protection:
  ROP      : 0x0 (Level 1, read protection of memories)
  RSP:
    RSP1    : 0x0 (No SFI process on going)

BOR Level:
  BOR_LVL  : 0x2 (reset level is set to 2.7 V)

User Configuration:
  IVAC1     : 0x1 (Independent watchdog is controlled by hardware)
  IVAC2     : 0x1 (Window watchdog is controlled by hardware)
  NLST_STOP_D1 : 0x1 (STOP node on Domain 1 is entering without reset)
  NLST_STOP_D2 : 0x1 (STOP node on Domain 2 is entering without reset)
  NLST_STDBY_D1: 0x1 (STANDBY node on Domain 1 is entering without reset)
  NLST_STDBY_D2: 0x1 (STANDBY node on Domain 2 is entering without reset)
  FZ_IWDS_STOP: 0x1 (Independent watchdog is running in STOP node)
  FZ_IWDS_STDBY: 0x1 (Independent watchdog is running in STANDBY node)
  SECURITY   : 0x1 (Security feature enabled)

BCMP       : 0x1 (<CM-7 boot enabled>)

Boot address Option Bytes:
  BOOT_CM7_ADDR0 : 0x0000 (0x00000000)
  BOOT_CM7_ADDR1 : 0x1FF1 (0x1FF10000)

PCROP Protection:
  PCROP_A_str   : 0x000 (0x00000000)
  PCROP_A_end   : 0x806 (0x00000000)

Secure Protection:
  SEC_A_str     : 0x0FF (0x00000000)
  SEC_A_end     : 0x0 (0x00000000)

DISC RAM Protection:
  ST_RAM_SIZE   : 0x3 (16 KB)

Write Protection:
  WRP0        : 0x1 (Write protection not active on this sector)
  WRP1        : 0x1 (Write protection not active on this sector)
  WRP2        : 0x1 (Write protection not active on this sector)
  WRP3        : 0x1 (Write protection not active on this sector)
  WRP4        : 0x1 (Write protection not active on this sector)
  WRP5        : 0x1 (Write protection not active on this sector)
  WRP6        : 0x1 (Write protection not active on this sector)
  WRP7        : 0x1 (Write protection not active on this sector)
```
# Reference documents

## Table 2. Document references

<table>
<thead>
<tr>
<th>Reference</th>
<th>Version</th>
<th>Document title</th>
</tr>
</thead>
</table>
9 Revision history

Table 3. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>03-Aug-2018</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>18-Apr-2019</td>
<td>2</td>
<td>Updated publication scope from 'ST restricted' to 'Public'.</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

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

© 2019 STMicroelectronics – All rights reserved