

Secure programming using STM32CubeProgrammer

## Introduction

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

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

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

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

Refer also to:

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





August 2021

AN5054 Rev 9

1/130 www.st.com

# Contents

1	Gen	eral info	ormation	10
	1.1	Licens	ing information	10
	1.2	Acrony	/ms and abbreviations	10
2	How inde	to gene penden	erate an execute-only/position t library for SMI preparation	11
	2.1	Requir	ements	.11
	2.2	Toolch	ains allowing SMI generation	.11
	2.3	Execu under	te-only/position independent library scenario example EWARM	12
		2.3.1	Relocatable library preparation steps	. 12
		2.3.2	Relocatable SMI module preparation steps	. 16
		2.3.3	Application execution scenario	. 17
3	Encr prep	rypted f aration	irmware (SFI) and module (SMI) using the STPC tool	19
	3.1	Syster	n requirements	19
	3.2	SFI ge	neration process	19
	3.3	SFIx g	eneration process	28
			Area E	28
			Area K	28
	3.4	SMI ge	eneration process	32
	3.5	SSP g	eneration process	35
	3.6	STM3	2 Trusted Package Creator tool in the command line interface	37
		3.6.1	Steps for SFI generation (CLI)	. 38
		3.6.2	Steps for SMI generation (CLI)	. 40
		3.6.3	Steps for SSP generation (CLI)	. 42
	3.7	Using	the STM32 Trusted Package Creator tool graphical user interface	44
		3.7.1	SFI generation using STPC in GUI mode	. 44 45
		3.7.2	SFIx generation using STPC in GUI mode	. 48
			SFIx GUI tab fields	49
		3.7.3	SMI generation using STPC in GUI mode	. 51 52



		3.7.4	SSP generation using STPC in GUI mode	54
		3.7.5	Settings	56
		3.7.6	Log generation	57
		3.7.7	SFI and SMI file checking function	58
4	Encr prog	ypted f rammir	irmware (SFI/SFIx)/ module (SMI) ng with STM32CubeProgrammer	. 59
	4.1	Chip c	ertificate authenticity check and license mechanism	. 59
		4.1.1	Device authentication	59
		4.1.2	License mechanism	59
			License mechanism general scheme	59
			License distribution	60
			HSM programming by OEM for license distribution	60
	4.2	Secure	e programming using a bootloader interface	. 61
		4.2.1	Secure firmware installation using a bootloader interface flow	61
		4.2.2	Secure Module installation using a bootloader interface flow	63
		4.2.3	STM32CubeProgrammer for SFI using a bootloader interface	63
		4.2.4	STM32CubeProgrammer for SMI via a bootloader interface	64
		4.2.5	STM32CubeProgrammer for SSP via a bootloader interface	65
		4.2.6	STM32CubeProgrammer get certificate via a bootloader interface	67
	4.3	Secure	e programming using JTAG/SWD interface	. 67
		4.3.1	SFI/SFIx programming using JTAG/SWD flow	67
		4.3.2	SMI programming through JTAG/SWD flow	69
		4.3.3	STM32CubeProgrammer for secure programming using JTAG/SWD	71
			Example "getcertificate" command using JTAG	71
			Example "smi" command using SWD	71
	4.4	Secure	e programming using Bootloader interface (UART/I2C/SPI/USB)	. 72
			SFI example	72
			SFIx example	72
5	Exar	nple of	SFI programming scenario	. 73
	5.1	Scena	rio overview	. 73
	5.2	Hardw	are and software environment	. 73
	5.3	Step-b	y-step execution	. 73
		5.3.1	Build OEM application	73
		5.3.2	Perform the SFI generation (GUI mode)	73



<ul> <li>5.3.4 Performing HSM programming for license generation using STPC (CLI mode)</li></ul>	77 78 78 79 80 80 80 80 83 83 83 83 83 83 83 83 83
Example of HSM version 1 provisioning.         Example of HSM version 2 provisioning.         Example of HSM get information         5.3.5       Programming input conditions         5.3.6       Perform the SFI install using STM32CubeProgrammer         Using JTAG/SWD       Using JTAG/SWD         6.1       Scenario overview         6.2       Hardware and software environment         6.3       Step-by-step execution         6.3.1       Build OEM application         6.3.2       Perform the SFI generation (GUI mode)         6.3.3       Programming input conditions         6.3.4       Perform the SFI used using STM32CubeProgrammer	77 78 78 79 80 80 80 83 83 83 83 83 83 83 83 83 83
Example of HSM version 2 provisioning.         Example of HSM get information         5.3.5       Programming input conditions         5.3.6       Perform the SFI install using STM32CubeProgrammer         Starple of SFI programming scenario for STM32WL         6       Example of SFI programming scenario for STM32WL         6.1       Scenario overview         6.2       Hardware and software environment         6.3       Step-by-step execution         6.3.1       Build OEM application         6.3.2       Perform the SFI generation (GUI mode)         6.3.3       Programming input conditions         6.3.4       Perform the SFI using STM32CubeProgrammer	78 79 80 80 80 83 83 83 83 83 83 83 83 83 83
Example of HSM get information         5.3.5       Programming input conditions         5.3.6       Perform the SFI install using STM32CubeProgrammer         Using JTAG/SWD       Using JTAG/SWD         6       Example of SFI programming scenario for STM32WL         6.1       Scenario overview         6.2       Hardware and software environment         6.3       Step-by-step execution         6.3.1       Build OEM application         6.3.2       Perform the SFI generation (GUI mode)         6.3.3       Programming input conditions         6.3.4       Perform the SFI install using STM32CubeProgrammer	78 79 80 80 83 83 83 83 83 83 83 85 85
<ul> <li>5.3.5 Programming input conditions</li> <li>5.3.6 Perform the SFI install using STM32CubeProgrammer</li> <li>Using JTAG/SWD</li> <li>6 Example of SFI programming scenario for STM32WL</li> <li>6.1 Scenario overview</li> <li>6.2 Hardware and software environment</li> <li>6.3 Step-by-step execution</li> <li>6.3.1 Build OEM application</li> <li>6.3.2 Perform the SFI generation (GUI mode)</li> <li>6.3.3 Programming input conditions</li> <li>6.3.4 Perform the SFI install using STM32CubeProgrammer</li> </ul>	79 80 80 83 83 83 83 83 83 83 83 83
<ul> <li>5.3.6 Perform the SFI install using STM32CubeProgrammer</li></ul>	80 80 . 83 . 83 83 83 83 83 83 85 85
6       Example of SFI programming scenario for STM32WL         6.1       Scenario overview         6.2       Hardware and software environment         6.3       Step-by-step execution         6.3.1       Build OEM application         6.3.2       Perform the SFI generation (GUI mode)         6.3.3       Programming input conditions         6.3.4       Perform the SFI using STM32CubeProgrammer	80 . 83 . 83 . 83 83 83 83 83 85 85
<ul> <li>Example of SFI programming scenario for STM32WL</li> <li>6.1 Scenario overview</li> <li>6.2 Hardware and software environment</li> <li>6.3 Step-by-step execution</li> <li>6.3.1 Build OEM application</li> <li>6.3.2 Perform the SFI generation (GUI mode)</li> <li>6.3.3 Programming input conditions</li> <li>6.3.4 Perform the SFI install using STM32CubeProgrammer</li> </ul>	. 83 . 83 . 83 . 83 . 83 . 83 . 83 . 85 . 86
<ul> <li>6.1 Scenario overview</li> <li>6.2 Hardware and software environment</li> <li>6.3 Step-by-step execution</li> <li>6.3.1 Build OEM application</li> <li>6.3.2 Perform the SFI generation (GUI mode)</li> <li>6.3.3 Programming input conditions</li> <li>6.3.4 Perform the SEI install using STM32CubeProgrammer</li> </ul>	. 83 . 83 . 83 . 83 . 83 . 83 . 85 . 85
<ul> <li>6.2 Hardware and software environment</li> <li>6.3 Step-by-step execution</li> <li>6.3.1 Build OEM application</li> <li>6.3.2 Perform the SFI generation (GUI mode)</li> <li>6.3.3 Programming input conditions</li> <li>6.3.4 Perform the SEI install using STM32CubeProgrammer</li> </ul>	. 83 . 83 . 83 . 83 . 83 . 83
<ul> <li>6.3 Step-by-step execution</li></ul>	. 83 83 83 85 86
<ul> <li>6.3.1 Build OEM application</li></ul>	83 83 85
<ul> <li>6.3.2 Perform the SFI generation (GUI mode)</li> <li>6.3.3 Programming input conditions</li> <li>6.3.4 Perform the SFI install using STM32CubeProgrammer</li> </ul>	83 85 85
6.3.4 Perform the SEI install using STM32CubeProgrammer	85 86
6.3.4 Perform the SEI install using STM32CubeProgrammer	86
7 Example of SFI programming scenario for STM32U5	. 88
7.1 Scenario overview	. 88
7.2 Hardware and software environment	. 88
7.3 Step-by-step execution	. 88
7.3.1 Build OEM application	88
7.3.2 Perform the SFI generation (GUI mode)	88
7.3.3 Programming input conditions	89
7.3.4 Perform the SFI install using STM32CubeProgrammer	90
Using JTAG/SWD.	90
8 Example of SMI programming scenario	. 93
8.1 Scenario overview	~~
	. 93
8.2 Hardware and software environment	. 93 . 93
<ul><li>8.2 Hardware and software environment</li><li>8.3 Step-by-step execution</li></ul>	. 93 . 93 . 93
<ul> <li>8.2 Hardware and software environment</li> <li>8.3 Step-by-step execution</li> <li>8.3.1 Build 3rd party Library</li> </ul>	. 93 . 93 . 93 . 93 . 93
<ul> <li>8.2 Hardware and software environment</li> <li>8.3 Step-by-step execution</li> <li>8.3.1 Build 3rd party Library</li> <li>8.3.2 Perform the SMI generation</li> </ul>	. 93 . 93 . 93 . 93 . 93 . 93



#### Contents

		8.3.4	Perform the SMI install	)5 95		
		8.3.5	How to test for SMI install success9	)7		
9	Exam	ple of S	Flx programming scenario for STM32H79	9		
	9.1	Scenario	o overview	9		
	9.2	Hardwa	re and software environment	9		
	9.3	Step-by-	-step execution	9		
		9.3.1	Build OEM application	99		
		9.3.2	Perform the SFIx generation (GUI mode)9	9		
		9.3.3	Performing HSM programming for license generation using STPC (GUI mode)	)1		
		9.3.4	Performing HSM programming for license generation using STPC (CLI mode)	)2		
		9.3.5	Programming input conditions	)2		
		9.3.6	Perform the SFIx install using STM32CubeProgrammer	)2		
			Using JTAG/SWD10	)2		
10	Exam	ple of S	Flx programming scenario for STM32L5	7		
	10.1	Scenario	o overview	17		
	10.2	Hardwa	re and software environment	17		
	10.3	Step-by-	-step execution	)7		
		10.3.1	Build OEM application	)7		
		10.3.2	Perform the SFIx generation (GUI mode)	)8		
			Use case 1 generation of SFIx without key area:	)8		
			Use case 2 generation of SFIx with key area:	10		
		10.3.3	Performing HSM programming for license generation using STPC (GUI mode)	11		
		10.3.4	Performing HSM programming for license generation using STPC (CLI mode) 11	11		
		10.3.5	Programming input conditions 11	1		
		10.3.6	Perform the SFIx install using STM32CubeProgrammer 11	1		
11	Exam	ple of c	ombined SFI-SMI programming scenario	5		
	11.1	Scenario overview				
	11.2	Hardware and software environment1				
	11.3	Step-by-	-step execution	5		
		11.3.1	Using JTAG/SWD 11	7		



		11.3.2	How to test the combined SFI install success
12	Exan	nple of \$	SSP programming scenario for STM32MP1 121
	12.1	Scenar	io overview
	12.2	Hardwa	are and software environment 121
	12.3	Step-by	<i>y</i> -step execution
		12.3.1	Building a secret file
		12.3.2	Performing the SSP generation (GUI mode)122
		12.3.3	Performing HSM programming for license generation using STPC (GUI mode)
		12.3.4	SSP programming conditions
		12.3.5	Perform the SSP install using STM32CubeProgrammer
13	Refe	rence d	ocuments
14	Revis	sion his	tory



# List of tables

Table 1.	List of abbreviations	. 10
Table 2.	SSP preparation inputs	. 36
Table 3.	Document references	126
Table 4.	Document revision history	127



# List of figures

Figure 1.	IAR example project overview	12
Figure 2.	Update compiler extra options	13
Figure 3.	Linker extra options	14
Figure 4.	Setting post-build option	15
Figure 5.	Postbuild batch file	16
Figure 6.	How to exclude the "lib.o" file from build	17
Figure 7.	app.icf file	18
Figure 8.	SFI preparation mechanism	19
Figure 9.	SFI image process generation	20
Figure 10.	RAM size and CT address inputs used for SFI multi install	21
Figure 11.	'P' and 'R' area specifics versus a regular SFI area	22
Figure 12.	Error message when firmware files with address overlaps used	23
Figure 13.	Error message when SMI address overlaps with a firmware area address	24
Figure 14.	Error message when a SFI area address is not located in Flash memory	25
Figure 15.	SFI format layout	26
Figure 16.	SFI image layout in case of split	27
Figure 17.	RAM size and CT address inputs used for SFIx multi install	29
Figure 18.	SFIx format layout.	30
Figure 19.	SFIx image layout in case of split	31
Figure 20.	SMI preparation mechanism	32
Figure 21.	SMI image generation process	33
Figure 22.	SMI format layout	34
Figure 23.	SSP preparation mechanism	35
Figure 24.	Encryption file scheme	36
Figure 25.	STM32 Trusted Package Creator tool - available commands	37
Figure 26.	Option bytes file example	39
Figure 27.	SFI generation example using an Elf file	40
Figure 28.	SMI generation example	41
Figure 29.	SSP generation success.	43
Figure 30.	SFI generation Tab	44
Figure 31.	Firmware parsing example	45
Figure 32.	SFI successful generation in GUI mode example	47
Figure 33.	SFIx generation Tab	48
Figure 34.	Firmware parsing example	49
Figure 35.	SFIx successful generation in GUI mode example	50
Figure 36.	SMI generation Tab	51
Figure 37.	SMI successful generation in GUI mode example	53
Figure 38.	SSP generation tab	54
Figure 39.	SSP output information	55
Figure 40.	Settings icon and Settings dialog box	56
Figure 41.	Log example	57
Figure 42.	Check SFI file example	58
Figure 43.	HSM programming GUI in the STPC tool	61
Figure 44.	Secure programming via STM32CubeProgrammer overview on STM32H7 devices .	62
Figure 45.	Secure programming via STM32CubeProgrammer overview on STM32L4 devices	62
Figure 46.	SSP install success	66
Figure 47.	Example of getcertificate command execution using UART interface	67



Eigung 10	SEL programming by ITAC/SMD flow everying (monolithic SEL image everyle)	60
Figure 48.	SFI programming by JTAG/SWD now overview (monoilunic SFI image example)	
Figure 49.		
Figure 50.		/1
Figure 51.		
Figure 52.		70
Figure 53.	Example product ID	//
Figure 54.	HSM Information in STM32 Trusted Package Creator CLI mode	78
Figure 55.		81
Figure 56.	SFI install success using SWD connection (2)	
Figure 57.	STPC GUI showing the STPC GUI during the SFI generation	84
Figure 58.	Example -dsecurity command-line output.	85
Figure 59.	Example -setdefaultob command-line output	86
Figure 60.	SFI install via SWD execution command-line output	87
Figure 61.	STPC GUI during the SFI generation	89
Figure 62.	SFI install via SWD execution (1)	91
Figure 63.	SFI install via SWD execution - (2)	92
Figure 64.	STPC GUI during SMI generation	94
Figure 65.	SMI install success via debug interface	96
Figure 66.	OB display command showing that a PCROP zone was activated after SMI	97
Figure 67.	Successful SFIx generation	. 100
Figure 68.	Example of HSM programming using STPC GUI	. 101
Figure 69.	SFIx install success using SWD connection (1)	. 103
Figure 70.	SFIx install success using SWD connection (2)	. 104
Figure 71.	SFIx install success using SWD connection (3)	. 105
Figure 72.	SFIx install success using SWD connection (4)	. 106
Figure 73.	Successful SFIx generation use case 1	. 109
Figure 74.	Successful SFIx generation use case 2	. 110
Figure 75.	SFIx install success using SWD connection (1)	. 112
Figure 76.	SFIx install success using SWD connection (2)	. 112
Figure 77.	SFIx install success using SWD connection (3)	. 113
Figure 78.	SFIx install success using SWD connection (4)	. 114
Figure 79.	SFIx install success using SWD connection (5)	. 114
Figure 80.	GUI of STPC during combined SFI-SMI generation	. 116
Figure 81.	Combined SFI-SMI programming success using debug connection	. 118
Figure 82.	Option bytes after combined SFI-SMI install success.	. 120
Figure 83.	STM32 Trusted Package Creator SSP GUI tab	. 122
Figure 84.	Example of HSMv2 programming using STPC GUI	. 123
Figure 85.	STM32MP1 SSP install success	. 125



# 1 General information

## 1.1 Licensing information

STM32CubeProgrammer supports STM32 32-bit devices based on  $\text{Arm}^{\textcircled{B}(a)}$  CortexB-M processors.

# arm

# 1.2 Acronyms and abbreviations

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

#### Table 1. List of abbreviations

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 executeonly (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, 3<sup>rd</sup> 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 3<sup>rd</sup> 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 must 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 the 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*.

🔆 AdvEx - IAR Embedded Workbench IDE			
File Edit View Project Simulator Tools Window Help			
	1	5 5	a 🗽 🐼 🐡 📣 🏟 📴 🛤 🥨 🕺 🕭
Workspace		×	
app - Debug		•	
Files	82	e:	
📙 🕂 🗊 app - Debug	~		
└─₽ 🗇 Lib - Debug	•		
memory_proxy.c			
test c			

#### 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:

- 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
	_	••••••••	•••••••••	•/	000000

app-Debug       €cho off         Files       €cho off         AdvEx       Files         AdvEx       Files         Dapp       Debug         Ibo       Ibo	Workspace	× app.icf postbuild.bat
Files       #EM This is a simple script that or REM using the tools in (%1).         Bapp - Debug          Bapp	app - Debug	Recho off
Image: Comparison of the comparison	Files	REM This is a simple script that cr REM using the tools in (%1). REM Make sure the old files are del
Induction       Induction         Image: State Composition       State Composition         Image: State Composition       Image: State Composition         Image: State Composition       Image: State Composition         Image: Composition       Image: Compo		<pre></pre>
└────────────────────────────────────	Image: Provide the second	ру
	Category: General Options Static Analysis Runtime Checking C/C++ Compiler Assembler Output Converter Custom Build Build Actions Linker Debugger Simulator Angel CMSIS DAP GDB Server IAR ROM-monitor I-jet/JTAGjet J-Link/J-Trace TI Stellaris Macraigor PE micro RDI ST-LINK Third-Party Driver TI XDS	Factory Settings         Discard Unused Publics         Diagnostics       MISRA-C:2004         MISRA-C:1998       Extra Options         Use command line options         Command line options: (one per line)         -ropi_cb         OK



#### How to generate an execute-only/position independent library for SMI preparation

- 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\$\postbuild.bat" "\$TOOLKIT\_DIR\$" "\$TARGET\_PATH\$" "\$PROJ\_DIR\$\lib.o"

This action is illustrated in *Figure 4*.



AdvEx - IAR Embedded Workbench IDE			
File Edit View Project Simulator Tools Window Help			
D 🗳 🖬 🕼 👗 🛍 💼 👘 여 여	- 1	8	* 12 🖻 🖻 🗢 🖷 🏰 🛤 👷 🕭 🕭
Workspace		×	
app - Debug		•	
Files	82	21 <sup>2</sup>	
⊨⊖ 🗇 app - Debug	*		
H H I app.c		*	
	~		
		_	
memory_proxy.c			
postbuild.bat			
C test.c			
			Options for node "Lib"
		_	Category:
			General Options
			Static Analysis
			C/C++ Compiler Build Actions Configuration
			Assembler Be huld an energy line
			Output Converter Output Converter
			Build Actions
			Linker "SPBOL DIRS'postbuild bat" "STOOLKIT DIRS" "STARGET PATE
			Debugger
			Simulator
			CMSIS DAP
			GDB Server
			IAR ROM-monitor
		_	I-jet/JIAGjet I-link/1-Trace
			TI Stellaris
			Macraigor
			PE micro
			ST-LINK
			Third-Party Driver
			TI XDS OK Cancel



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. F	Postbuild	batch	file
i igaio	••••	ootsana	Naton	

1	lecho off
2	REM This is a simple script that creates and object file (\$3) from an image (\$2)
3	REM using the tools in (%1).
4	
5	REM Make sure the old files are deleted before we try to generate the new ones
6	if exist 82.tmp (
7	del %2.tmp
8	
9	if exist 83 (
10	del %3
11	
12	
13	echo Do magic encryption here (copy is just a placeholder)
14	copy 82 82.tmp
15	
16	REM This is the list of functions that will have a wrapper generated
17	SETWRAP=wrap ToStringwrap setup_memorywrap setup_memory2
18	
19	REM convert the image to a linkable object file using _Lib as prefix
20	REM and keeping all mode symbols (helps a bit with debugging)
21	81\bin\iexe2obj.exeprefix Libkeep mode symbols % WRAP% %2.tmp %3
22	

5. Rebuild the project "Lib"

#### 2.3.2 Relocatable SMI module preparation steps

From the object file created, *"lib.o"*, generate the SMI relocatble module using the STM32 Trusted Package Creator 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 3.6.2: 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 Figure 63.: SFI install via SWD execution (2)* to perform this action).
- 2. Link the data clear part, *"libr\_clear.o"*, generated from STM32 Trusted Package Creator tool to the final IAR example application instead of the old previously used *"lib.o"*.
- 3. Exclude *"lib.o"* from the build (*Figure 6*).

Norkspace	×	
app - Debug		
app - Debug         Files         app - Debug		Options for node "app"         Image: Category:         Custom Build         Custom Build         Custom Build         Custom Tool Configuration         Filename extensions:         Command line:         Output files (one per line):         Additional input files (one per line):
		Run this tool before all other tools      OK Cancel

#### Figure 6. How to exclude the "lib.o" file from build

- 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 7.

pp.icf	f0 •	×
<pre>/*###ICF### Section handled by ICF editor, don't touch! ****/ /*-Editor annotation file-*/</pre>		Á
<pre>/* IcfEditorFile="\$TOOLKIT_DIR\$\config\ide\IcfEditor\cortex_v1_0.xml" */ /* Createle */</pre>		
/*-Specials-*/ define symbol ICFEDIT intyec start = 0x24000000:		
/*-Memory Regions-*/		
<pre>define symbolICFEDIT_region_ROM_start_ = 0x24000000;</pre>		
define symbolICFEDIT_region_ROM_end = 0x24002FFF;		
define symbol ICFEDIT region RAM end = 0x2407FFFF;		
<pre>iefine symbolICFEDIT_region_PCROP_start_ = 0x08080000; iefine symbolICFEDIT_region_PCROP_end_ = 0x0809FFFF;</pre>		
/*_S1769_*/		
define symbol ICFEDIT size cstack = 0x2000;		
define symbolICFEDIT_size_heap = 0x2000;		
/**** End of ICF editor section. ###ICF###*/		
define symbolregion RAM1 start = 0x10000000;		
<pre>define symbolregion_RAM1_end = 0x1000FFFF;</pre>		
define memory mem with size = 46:		
define region ROM_region = mem:[from _ICFEDIT_region_ROM_starttoICFED	II_region_ROM_end_];	
define region RAM_region = mem: [from _ICFEDIT_region_RAM_start_ to _ICFED	/IT_region_RAM_end];	
define region RAMI region = mem: [from region RAMI start to region RAMI define region PCROP region = mem: [from ICFEDIT region PCROP start to ICF	ena ;; TDII region ECROP end 1:	
<pre>define block CSTACK with alignment = 8, size = _ICFEDIT_size_cstack { }; define block HEAP with alignment = 8, size = _ICFEDIT_size_heap { };</pre>		=
/*define block PCROP_block with alignment = 256 {ro code sectioncodeLib};*	1	
initialize by copy { readwrite };		
do not initialize { section .noinit };		
<pre>place at address mem:ICFEDIT_intvec_start { readonly section .intvec };</pre>		
place in ROM region { readonly };		
place in RAM_region { readwrite,		
block CSTACK, block HEAP };		
prace in Nami_region ( Section .stam );		
"SMI": place noload in PCROP_region { ro code sectioncodeLib};		
/*place in PCROP region { block PCROP block };*/		

#### Figure 7. app.icf file

- 6. To check that example application executed successfully on the STM32H7 device:
  - a) Check that address 0x08080000 was protected with PCROP.
  - b) The expected "printf" packets appears in the terminal output.



# 3 Encrypted firmware (SFI) and module (SMI) preparation using the STPC tool

The STM32 Trusted Package Creator (STPC) 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 STM32 Trusted Package Creator tool for SFI/SFIx, SMI and SSP image generation requires a PC running on either Windows<sup>®</sup> 7/10, or Ubuntu<sup>®</sup> 14 in 64-bit versions and macOS<sup>®</sup>.

# 3.2 SFI generation process

The SFI format is an encryption format for internal firmware created by STMicroelectronics that transforms internal firmware (in ELF, Hex, Bin or Srec formats) into encrypted and authenticated firmware in SFI format using AES-GCM algorithm with a 128-bit key. The SFI preparation process used in the STM32 Trusted Package Creator tool is described in *Figure 8*.









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



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

#### IV = nonce + Area Index

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

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



These areas can be of different types:

- 'F' for a firmware area (a regular segment in the input firmware)
- '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 STM32 Trusted Package Creator overview below (*Figure 10*) 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.

M32 Trusted P	ackage Creator				
File	Edit	Options	Help		life.augmen
6 🛞	<b>.</b>				
	SFI		SMI	SFU	нѕм
Firmware fi	les			Firmware information	SFI information
tests.axf			Add Remove	Overview	
Encryption	key file		i	File name	
M32TrustedPac	ckageCreator/Input/Si	FI/good/test_firmwar	e_key.bin Open	Size	
Nonce file				Segments	
/projects/STM	32TrustedPackageCre	ator/Input/SFI/good/	nonce.bin Open	Index Type Size	Address
Option byte	s file				
projects/SFMI-	PreparationToolv0.2.0	0_test1/Input/SFI/go	od/ob.csv Open		
SMI files (0	nly for combined c	ase)			
511013214	I-DISCOU.SMI		Remove		
Terrore	ian				
24 🚔	ion.				
RAM size	C	ontinuation toker	address		
Output SFI	file				
C:/projects/ST	M32TrustedPackageC	reator/output/out.sf	Select folder		
					Generate SEI

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.

Area format	New Pause Area	New Resume Area
Type ('F', 'M', 'C')	Type 'P'	Type 'R'
Version	Version	Version
Index	Index	Index
Size	Size = 0	Size = 0
Address	Address of CT	Address of CT
Total Nb of areas	Total Nb of areas	Total Nb of areas
Tag	Tag	Tag
Encrypted Area Content - Firmware - Module - Configuration		

Figure 11. 'P' and 'R' area specifics versus a regular SFI area

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, make sure that there is no overlap between their segments, otherwise an error message appears (Figure 12: Error message when firmware files with address overlaps used).



STM32 Trusted	Package Creator		-		- Ο Σ
File	Edit	Options	Help		
8 🗸 🖗	🔤   😻				
	SFI		SMI	SFU	н5м
Firmware	files			Firmware information	SFI information
tests.ax tests.ax	xf xf		Add Remove	Overview	
Encryption	n key file			File name	
M32TrustedP	PackageCreator/Input/S	FI/good/test_firmwa	re_key.bin Open	Туре	0
Nonce file		tror			
/projects/STI	M32TrustedPackageCre	ator/	Overlap between segme	nts, Unable to merge firmware files	Address
Option by	tes file				Address
projects/SFM	1I-PreparationToolv0.2.	0_tes		ОК	
SMI files (	(Only for combined o	ase)			
			Add Remove		
Image ve	rsion				
24 🚔					
RAM size		Continuation toke	n address		
Output SF	I file				
C:/projects/s	STM32TrustedPackage(	Creator/output/out.s	fi Select folder		
					Generate SFI

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*).

SFI	SMI	
Firmware files  tests.axf Add	Firmware information	SFI information
Encryption key file aration_tool_v0.2.0_windows/bin/Input/SFI/test_firmwa	File name STM32F4-DISCO0.smi	
Nonce file Overlap bet Overlap bet Overlap bet Overlap bet	OK Size	Address
0.2.0/SFMIPreparation_tool_v0.2.0_windows/bin/Input/SFI/ob.csv Open SMI files (Only for combined case)	262144 B	0x8000000
STM32F4-DISCO0.smi Add Remove		
23 ★ Output SFI file		
_v0.2.0/SFMIPreparation_tool_v0.2.0_windows/bin/output/out.sfi Select folder		

#### 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 the following error message appears (*Figure 14*).

TM32 Trusted Package Creator	-				
File Edit	Options	Help			
🤞 😵 📇 😒					
SFI		SMI	SF	υ	нѕм
Firmware files			Firmware	information	SFI information
tests.axf		Add Remove	Overview		
Encountion koy file			File name	out.sfi	
M32TrustedPackageCreator/Top	ut/SEI/good/test_firmware_ke	v.bin Open	Size	266.656 KB	
nozin asteur ackageoreator/imp		open-	Protocol versio	n 01	
Nonce file	Ca Erro	r	Sugarantin .		×
/projects/STM32TrustedPackag	eCreator/Input/SFI	Error: One or mo	re SFI areas are not	located in Flash memo	ry Address
Option bytes file		y			0x8000000
projects/SFMI-PreparationToolv	0.2.0_test1/Input/			ОК	0x8030000
SMI files (Only for combine	ed case)			viouuie 20214	+ - Ox8000000
STM32F4-DISCO0.smi		Add	4 Con	figuration 36 I	3 0x0
Image version					
RAM size	Continuation token add	iress			
Output SFI file					
C:/projects/STM32TrustedPacka	ageCreator/output/out.sfi	Select folder			
					Generate SEI

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 SFIx generation process

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

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

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

#### Area E

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

- OTFD region\_number (uint32\_t):
  - 0...3: OTFD1 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, and STM32L5)
  - 4...7: OTFD2 (STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335)
- OTFD region\_mode (uint32\_t) bit [1:0]:
  - 00: instruction only AES-CTR)
  - 01: data only (AES-CTR)
  - 10: instruction + data (AES-CTR)
  - 11: instruction only (EnhancedCipher)
  - OTFD key\_address in internal Flash memory (uint32\_t).

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

#### Area K

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

- the size of the key area (uint32\_t)
- the start address of the key area (uint32\_t): address in internal Flash memory.

Example of an area 'K':

0x0000002 0x0000080 0x08010000 0x00000020 0x08010100

There are two key areas:

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

The STM32 Trusted Package Creator overview below (*Figure 17: RAM size and CT address inputs used for SFIx multi install*) shows the RAM size input for SFIx image generation, and also the 'Continuation token address' input, which is used by SFIx multi install to store states in external/internal Flash memory during SFIx programming.

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



TM32 Truste	d Package Creator	<u> </u>					-				-	
File	Edit	Options	Help									ife.augmer
6 😵	a 🔒 🙁											
											_	
5	ศ	SFIx		SSP	WB	SIG		SMI		HSM		
Internal fi	irmware files						Firr	nware information		SFIx	information	
					Add Remove		Overview					
External f	irmware files						File name	Type	Size	Region number	Region mode	Key ad
otfd_p	art1.hex				Add Remove							
Key area	file											•
C:/SFIx/otfo	l_key_areas.kcsv				Open		Segments					÷
Encryption	n key file						Index	Size		Address		
C:/SFIx/key	.bin				Open							
Nonce file												
C:/SFIx/non	ce.bin				Open							
Option by	tes file											
C:/SFIx/ob.	CSV				Open							
Image ve	rsion											
1 ±												
RAM size		Cor	ntinuation token a	idress								
Output SF	I file											
C:/SFIx/out	sfix				Select folder		1					
											Genera	te SFIx

Figure 17. RAM size and CT address inputs used for SFIx multi install

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



The final output from this generation process is a single file, which is the encrypted and authenticated internal/external firmwares in ".sfix" format. The SFIx format layout is described in *Figure 18*.







When the SFIx image is split during generation, areas 'P' and 'R' appear in the SFIx image layout, as in the example below *Figure 19*.



Figure 19. SFIx image layout in case of split



## 3.4 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 20).







The SMI generation steps as currently implemented in the tool are described in the diagram below (*Figure 21*).



Figure 21. 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, tow outputs are created under the specified path:

- The SMI image (Figure 22 represents the SMI format layout).
- The library data part.







## 3.5 SSP generation process

SSP is an encryption format that transforms customer secret files into encrypted and authenticated firmware using an AES-GCM algorithm with a 128-bit key. The SSP preparation process used in the STM32 Trusted Package Creator tool is shown in *Figure 23*.





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 based on the following inputs:

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



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

#### Table 2. SSP preparation inputs

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

Figure 24. Encryption file scheme




# 3.6 STM32 Trusted Package Creator tool in the command line interface

This section describes how to use the STM32 Trusted Package Creator tool from the command line interface in order to generate SFI/SFIx and SMI images. The available commands are listed in *Figure 25*.

Figure	25	STM32	Trustad	Packano	Creator	tool -	availahlo	commands
Iguie	<b>Z</b> J.	JINJZ	nusieu	гаскауе	Cleator	1001 -	available	commanus

SFI prep	aration options	
-sfi,	sfi :	Generate SFI image,
		You also need to provide the information listed below
-fi	r,firmware	: Add an input firmware file
	<firm_file></firm_file>	: Supported firmware files are ELF HEX SREC BIN
-fi	rx,firmwx	: Add input for external firmware file
	<firmx_file></firmx_file>	: Supported externalfirmware files are ELF HEX SREC BIN
	[ <address>]</address>	: Only in case of BIN input file (in any base)
	[ <region_number>]</region_number>	: Only in case of BIN input file (in any base): [0:3]: OTFD1 (STM32H7A / STM32L5), [4:7]: OTFD2 (STM32H7A/B case)
(AFC. 070	[ <region_mode>]</region_mode>	: Only in case of BIN input file (in any base),only two bit [0:1] where 00 : instruction only (AES-CTR), 01 : data only
(AES-CIR	), 10: Instruction	+ data (RES-CTR), II: INSTRUCTION ONLY (ENNANCEOLIPHEY)
- 12	[ <key_address>]</key_address>	: Only in case of Bin input file (in any base), random key values in internal flash memory
-к,	(Key Files	· Rin file its size must be 16 butes
-kx	kevx	· key area for external firmure
κ.,	<kev area="" file=""></kev>	: CSV file contains a set of couple (size.start address)
-n.	nonce	: AES-GCM nonce
	<nonce_file></nonce_file>	: Bin file, its size must be 12 bytes
-V,	ver	: Image version
	<image_version></image_version>	: Its value must be in <0255> (in any base)
-ob,	obfile	: Option bytes configuration file
	<csv_file></csv_file>	: CSV file with 9 values
-m,	module	: Add an SMI file (optional for combined case)
	<smi_file></smi_file>	: SMI tile
	[ <address>]</address>	: Only in case of a relocatable SMI (with Address = 0)
-rs,	ramsize	; derine available ram size (for multi-image)
	<size></size>	: Size in bytes
-ct,	token	. Contrination token address (for multi-image)
-0	outfile	. Address
-0,	<output files<="" td=""><td>: SET file to be created</td></output>	: SET file to be created
SMI prep	aration options	
-smi.	smi,	: Generate SMI image
		You also need to provide the information listed below
-elf	,elfile	: Input ELF file
	<elf_file></elf_file>	: ELF file
-s,	sec	: Section to be encrypted
	<section></section>	Section name in the ELf file
-k,	key	: AES-GLM ENCRYPTION KEY
	<key_file></key_file>	: Bin Tile, its size must be to bytes
-n,	nonce	. AL2-BCM HONCE
<b>C1</b>	chonce_rile>	· Servita version
-50,	(SV File)	: Its size must be 16 bytes
-0	outfile	Generated SMI file
•,	(Output File>	: SMI file to be created
-c.	clear	: Clear ELF file
	<pre>KClear_File&gt;</pre>	: Clear ELF file to be generated



## 3.6.1 Steps for SFI generation (CLI)

In order to generate an SFI/SFIx 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>	:Firmware file.
[ <address>]</address>	:Used 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< th=""><th>_file&gt;</th><th>[<address>]</address></th><th>[<region_< th=""><th>_Number&gt;]</th></region_<></th></firmware<>	_file>	[ <address>]</address>	[ <region_< th=""><th>_Number&gt;]</th></region_<>	_Number>]
---------	---	--------	------------------------	--	-----------

	[ <region_mode>] [<key_address>]</key_address></region_mode>
<firmware_file></firmware_file>	: Supported external firmware files are ELF HEX SREC BIN.
[ <address>]</address>	:Only in case of BIN input file (in any base).
<region_number></region_number>	: Only in case of BIN input file (in any base): [0:3]: OTFD1 (STM32H7A3/7B3, STM32H7B0 or
STM32L5), [4:7]:	OTFD2 (STM32H7A3/7B3 and STM32H7B0 case).
<region_mode></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></key_address>	: Only in case of BIN input file (in any base), random key values in internal Flash memory.

#### -k, --key

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

Syntax: -k <Key\_file>

< Key \_file> : A 16 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.

Note:

The number of rows in the CSV file is product dependent (refer to the example available for each product). For instance there are 9 rows for all STM32H7 products, with the last row "reserved", except for dual-core devices. It is important to neither change the order of, nor delete, rows.

Example: for STM32H7xx devices, 9 option bytes registers must be configured, which corresponds to a total of 9 lines in the csv file (*Figure 26*).

Syntax: -ob <CSV\_file>

<CSV\_file >: A csv file with 9 values.

#### Figure 26. Option bytes file example

1	FOPTSR PRG, 0x1026AAD0
2	FPRAR PRG A,0x81000200
3	FPRAR PRG B,0x81000200
4	FSCAR_PRG_A,0x81000200
5	FSCAR_PRG_B,0x81000200
6	FWPSN_PRG_A, 0xFFFFFFFF
7	FWPSN_PRG_B, 0xFFFFFFFF
8	FBOOT7_PRG,0x24000800
9	RESERVED, 0x10000810

#### -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

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



AN5054 Rev 9

-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>

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 27.

Figure 27. SFI generation example using an Elf file

C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin>STM32Trust edPackageCreator\_CLI.exe -sfi -fir tests.axf -k test\_firmware\_key.bin -n nonce.b in -ob ob.csv -v 23 -o out.sfi SFI generation SUCCES

## 3.6.2 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 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 28. SMI generation example

C:\SFMIPreparation Tool v0.2.0>SFMIPreparationTool\_CLI -smi -elf FIR\_module.axf -s "ER\_PCROP" -k test\_firmware\_key.bin -n nonce.bin -sv svFile -o test.smi -c cl ear.axf The section does not end on a Flash word boundary



## 3.6.3 Steps for SSP generation (CLI)

In order to generate an SSP image in CLI mode, the user must use the "-ssp, --ssp" command followed by the appropriate inputs.

Inputs for the "ssp" command are:

-ru, --rma\_unlock

Description: RMA unlock password

Syntax: -ru <RMA\_Unlock>

<RMA\_Unlock> : Hexadecimal value 0x0000 to 0x7FFF

-rr, --rma\_relock

Description: RMA relock password

Syntax: -rr <relock\_value>

<relock\_value> : Hexadecimal value 0x0000 to 0x7FFF

-b, --blob

Description: Binary to encrypt

Syntax: -b <Blob>

<Blob> : Secrets file of size 148 bytes

-pk, --pubk

Description: OEM public key file

Syntax: -pk <PubK.pem>

<PubK> : pem file of size 178 bytes

-k, --key

Description: AES-GCM encryption key

```
Syntax: -k <Key_File>
```

<Key\_File> : Bin file, its size must be 16 bytes

-n, --nonce

Description: AES-GCM nonce

Syntax: -n <Nonce\_File>

<Nonce\_File> : Bin file, its size must be 16 bytes

-o, --out

Description: Generate SSP file

Syntax: -out <Output\_File.ssp>

<Output\_File> : SSP file to be created with (extension .ssp)

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



Example SSP generation command:

STM32TrustedPackageCreator\_CLI -ssp -ru 0x312 -rr 0xECA
-b "C:\SSP\secrets\secrets.bin"
-pk "C:\SSP\OEMPublicKey.pem" -k "C:\SSP\key.bin"
-n "C:\SSP\nonce.bin" -o "C:\out.ssp"

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

Figure 29. SSP generation success





# 3.7 Using the STM32 Trusted Package Creator tool graphical user interface

The STPC is also available in graphical mode, this section describes its use. The STM32 Trusted Package Creator tool GUI presents two tabs, one for SFI generation, one for SFIx generation and one for SMI generation.

# 3.7.1 SFI generation using STPC in GUI mode

*Figure 30* shows the graphical user interface tab corresponding to SFI generation.

132 Trusted F	Package Creator						
File	Edit	Options	Help				
6 😣	<b>.</b>						
	SFI		SMI		SFU		ням
Firmware f	iles			Firm	ware ir	nformation	SFI information
tests.axi			Remove	Overview	,		
Encryption	key file	/SET/good/test_firmware	kay bin Open	File name Type	tests. ELF	axf	
Nonce file	coage creator / Input	, 31 1/good/test_niniware	vey.bit	Size	815.8	87 KB	
/projects/STM	132TrustedPackageC	Creator/Input/SFI/good/n	once.bin Open	Segment	s	Size	Address
Option byte	es file			1		844 B	0x8000000
projects/SFMI	-PreparationToolv0.	2.0_test1/Input/SFI/goo	d/ob.csv Open	2		9884 B	0x8030000
SMI files (0	only for combined	case)					
STM32F	4-DISCO0.smi		Add Remove				
Image vers	sion						
RAM size		Continuation token a	oddress				
Output SFI	file						
C:/projects/S	TM32TrustedPackag	eCreator/output/out.sfi	Select folder				
							Gene <u>rate SFI</u>

Figure 30. SFI generation Tab

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

• Firmware files:

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

If the file is valid, it is appended to the "input firmware files" list, otherwise an error message box appears 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 31*).

M32 Trusted	Package Creator						
File	Edit	Options	Help				life.augment
6 😵							
	SFI	s	МІ			SFU	HSM
Firmware	files				Firmw	are information	SFI information
tests.a	xf		Add Remove		Overview		
Encryption	n kev file				File name	tests.axf	<b>_</b>
I Package C	Creator v1.0.2/Input/	SFI/good/test firmware kev	bin Open		Туре	ELF	
		, , , , , , , , , , , , , , , , , , ,			Size	815.887 KB	<b>_</b>
Nonce file				X	Segments		
M32_Trusted	d_Package_Creator_v	1.0.2/Input/SFI/good/nonce	bin Open		Index	Size	Address
Option by	tes file			1	1	844 B	0x8000000
/STM32_Trus	sted_Package_Creator	r_v1.0.2/Input/SFI/good/ob.	csv Open		2	9884 B	0x8030000
SMI files (	Only for combined	case)					
			Add				
			Remove				
Image ve	rsion						
12 ÷							
RAM size		Continuation token addr	ess				
Output SF	I file						
loads/STM32	2_Trusted_Package_Cr	reator_v1.0.2/output/out_to	to1. Select folder				
							Course Cla
							Generate SFI

Figure 31. Firmware parsing example



- Encryption key and nonce file:
   The encryption key and nonce file are selected by entering their paths (absolute or relative), or by selecting them with the "Open" button. Notice that sizes must be respected (16 bytes for the key and 12 bytes for nonce).
- Option bytes file: The option bytes file are selected the same way as the encryption key and nonce. Only csv files are supported.
- *Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES

The option bytes are described in the product reference manual.

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

SMI files:

SMI files are 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 is done by entering the folder path (absolute or relative) or by using the "Select folder" button.

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

• 'Generate SFI' button:

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

If everything goes well, a message box indicating successful generation appears (*Figure 32: SFI successful generation in GUI mode example*) and information about the generated SFI file is displayed in the SFI information section.

M32 Trusted	Package Creator								
File	Edit	Options	Help						igmei
<b>é</b> 😵	) 👼 😵								
	SFI		SMI			SFU		нѕм	
Firmware	files				Firmwa	are information		SFI information	
tests.a	xf		Add Remove		Overview				_
Encryptio	n key file				ile name	out_toto1.	sfi		-
J_Package_(	Creator_v1.0.2/Input/S	FI/good/test_firmwa	re_key.bin Open		Size	10.627 KB			
Nonce file	•		Ca Success			<b>b</b>   01			-
M32_Truste	d_Package_Creator_v1	1.0.2/Input/SFI/good	/non SFI su	ccessfull	y created	- 1		1	_
Option by	rtes file			ж		Type	844 B	Address	-
/STM32_Tru	sted_Package_Creator	_v1.0.2/Input/SFI/go	od/o		2	Firmware	9884 B	0x8030000	-
SMI files (	(Only for combined o	ase)			3 (	Configuration	36 B	0x0	_
			Add						
			Remove						
Image ve	ersion								
12 🔹									
RAM size		Continuation toke	address						
Output SF	I file								
loads/STM3	2_Trusted_Package_Cr	eator_v1.0.2/output,	out_toto1. Select folder						
								Generat	e SEI
								Generat	

# Figure 32. SFI successful generation in GUI mode example



# 3.7.2 SFIx generation using STPC in GUI mode

*Figure 33 shows* the graphical user interface tab corresponding to SFIx generation.

File	Edit	Options	Help									S	life.augme
۵	5												
SF	I	SFIx		SSP	w	B SIGN			SMI		HSM		
Internal firm	ware files							Firmware i	informatior			SFIx information	
					Remove		Overview						
External firm	ware files				Add		File name	-	Туре	Size	Region number	Region mode	Key address
Kow area file					Remove		otfd_part1.h	x Int	tel Hex	1.44 KB	0x0	0x0	0x8000000
C:/SFIx/otfd_k	ey_areas.kcsv	<u>c</u>	STM32 Trusted Packa	? ×	Open		4						Þ
Encryption k	ey file	E	xternal firmware file: o	tfd_part1.hex		Ш.	Segments						
C:/SFIx/key.bir	1	5	tart address: 0x90	000000	Open		Index 1	Si 51	ze 2 B		Add 0x9000	ress	
Nonce file		R	tegion number: 0 🔅										
C:/SFIx/nonce.	bin	ĸ	legion mode: 0 🖂	00000	Open								
Option bytes	file		ок	Cancel									
C:/SFIx/ob.csv					Open								
Image versi	n												
1 I			Continuation teleon	ddrass									
Output SET 6	h		continuation token a	uurc33									
C:/SFIx/out.sfi	ĸ				Select folder								

#### Figure 33. SFIx generation Tab

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



#### SFIx GUI tab fields

Firmware files: The user needs to add the input firmware files with the "Add" button. If the file is valid, it is appended to the "input firmware files "list, otherwise an error message box appears 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 34*).

File     Edit     Options     Help       Image: State of the state of	
STI     STIX     SSP     WD SIGH     SHI     HSM       Internal firmware files     Add Remove     Firmware information     SFD       Certain firmware files     Add Remove     SED     Overview       External firmware files     Add Remove     Size     Region number       Regrarea file     Add Remove     Size     Region number	(information
SFT     SFT     SSP     VB STGN     SHI     HSM       Internal firmware files     Add Remove     Firmware information     SFD       External firmware files     Add Remove     Overview     Size     Region number     Region number       Key area file     Intel Hex     1.44 KB     0x0     Region     Region	s information
Internal firmware files       Add     Remove     Firmware information     SFD       External firmware files     Add     Overview     Overview       i ortid part/hex     Add     Remove     Remove       Reyrarea file     Intel Hex     1.44 KB     0x0	k information
Add Remove     Overview       External firmware files     Add Remove       ottid part1.hex     Add Remove       Key area file     Intel Hex	uine model - Knowthere
External firmware files     Add remove       i dtd.part1.hex     Type       Key area file	
Companyation         Remove         off_part1.hex         Intel Hex         1.44 KB         Dx0	Igion mode Key address
key area_iiic	0x0 0x8000000
C:/SFIx/otfd_key_areas.kcsv Open	<u>&gt;</u>
Encryption key file	
Cr/SFIX/key.bin Open 1 512.8 Overnoord	
Nonce file	
C:/SFIx/honce.bin Open	
Option bytes file	
Cr/SFix/do.csv Open	
Image version	
RAM size Continuation token address	
Output SFI file	
C;SF1x/out.stix Select folder	
	Generate SFIx



As is the case for the SFI use case, once all fields are filled in properly, the "Generate SFIx" button becomes enabled. The user can generate the SFIx file by a single click on it. If everything goes well, a message box indicating successful generation appears (*Figure 35: SFIx successful generation in GUI mode example*) and information about the generated SFIx file is displayed in the SFIx information section.



		guie do. di ix	successiul ye	enei	ation			zvamb			
TM32 Trusted	Package Creato	r							-	- 🗆	
File	Edit	Options	Help							life quan	mer
	<b></b>									metadgi	
SFI		SFIX	ssp w	B SIGN		SMI		HSM			
Internal fir	mware files				Firm	ware informa	tion		SFIx informat	tion	
firmwar	e.hex		Add Remove								
External fir	mware files				Overview	w (			. [i		_
External.hex					File n	ame Size		Image version		Internal segm	
Key area fi	e				out.sfix	1016	В	01	1		
C:/SFIx/key_	areas.kcsv		Ca Success			×				ŀ	
Encryption	key file		SFlx:	succes	sfully creat	ed			1		-
C:/SFIx/key.b	pin		- <b>·</b>		_	Туре		Size	Addre	255	-
Nonce file				ОК		rmware		10 B	0x8010	000	
C:/SFIx/nonc	e.bin		Open		2	Device		40 B	0,0010	000	
Option but	as filo				5	Pause		22 0	0,0010	000	
C:/SEIx/ob.cs	es nie		Open		5	External		512 D	0x8010	000	
			open		6	Pause		22.0	0x90000	020	
Image vers	sion				7	Recurso		32 D	0x0010	020	
1					0	Configuratio		26 P	0x0010	020	
RAM size		Continuation token a	ddress 0x08010000	-	• •	eoninguratio		30 0	Ux0	•	
Output SFI:	x file				Parse SF	·1x file				Dunna	
C:/SFIx/out.s	fix		Select folder		1					browse	
									Cono	rate SFTx	
									Gene		

# Figure 35. SFIx successful generation in GUI mode example



# 3.7.3 SMI generation using STPC in GUI mode

*Figure 36* shows the graphical user interface tab corresponding to SMI generation.

•   •								
	SFI	SMI		9	SFU		HSM	
ELF file				ELFi	nformation		SMI information	
		Open		Overview				
Encryptio	n key file			File name				
		Open		ELF Type				
Nonce file				ELF Machine	2			_
		Open		Size				-
		open	Г	Sections	Name	Tune	Size	
Security v	/ersion file		-	Index	Name	Туре	5126	
		Open						
Section to	encrypt							
	•							
Output SI	1I file							
		Select folder						
Output cle	ear ELF file							
		Calact faller		•				▶

Fiaure	36.	SMI	generation	Tab
	•••	•••••	generation	

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 only be 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 are 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 is 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. This can be kept or changed.

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 37: SMI successful generation in GUI mode example*), with additional information about the generated SMI file displayed in the 'SMI information' section. In the case of invalid input data, an error message box appears instead.



STM32 Trusted Pa	ckage Creator					0 13
File	Edit	Options	Help			
					Life.au	igmented
«   •   •	<u>•</u>					
	SFI		SMI	SFU	нѕм	
ELF file				ELF information	SMI information	
ckage_Creator_	v1.0.2/Input/SMI/goo	od/MDK-ARM/FIR_mo	dule.axf Open	Overview		
Encryption k	ey file			Original file name FIR_mod	dule.smi	
_Package_Crea	tor_v1.0.2/Input/SMI	[/good/test_firmware	_key.bin Open	Number of files 1		
Nonce file			Canal Information	1.64844	KB	
M32_Trusted_P	ackage_Creator_v1.0	.2/Input/SMI/good/n	or i SMI suc	ccessfully created	<i></i>	
Security ver	sion file					
\$/STM32_Truste	d_Package_Creator_	v1.0.2/Input/SMI/goo	od/svFile Open			
Section to er	ncrypt					
ER_PCROP	•					
Output SMI f	ìle					
/STM32_Trusted	l_Package_Creator_v	1.0.2/output/FIR_mo	odule.smi Select folder			
Output clear	ELF file					
2_Trusted_Pack	age_Creator_v1.0.2/	output/FIR_module_o	dear.axf Select folder			
					Generat	e SMI

Figure 37. SMI successful generation in GUI mode example



## 3.7.4 SSP generation using STPC in GUI mode

*Figure 38* shows the SSP generation graphical user interface tab.

ile	Edit	Options	Help				life.au
6 😣	) 📑 🔇	)					
SFI		SFIx	SSP	WB SIGN	SMI	HSM	
Password	: RMA Loo	k 0x312	RMA Reloc	k 0xECA	Secrets file infor	mation	SSP information
Secrets fi	le				Overview		
C:/SSP_Inp	ut/secrets/148b	ytes_secrets.bin		Open	File name	Туре	Size
Encryptio	n key file				148bytes_secret		148 B
C:/SSP_Inp	ut/aes_key/key	bin		Open	,		
OEM publi	c key file						
C:/SSP_Inp	ut/OEMPublicKe	y.pem		Open			
Nonce file							
C:/SSP_Inp	ut/aes_key/iv.b	in		Open			
Output SS	P file						
C:/ssp/out.	ssp			Select folder			

#### Figure 38. SSP generation tab

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

#### SSP GUI tab fields

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

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

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

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

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

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



Г

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



Figure 39. SSP output information

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

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



## 3.7.5 Settings

The STPC allows generation to be performed respecting some user-defined settings. The settings dialog are displayed by clicking the settings icon (see *Figure 40*) in the tool bar or in the menu bar by choosing: Options -> settings.

File	Edit	Options	Help			
6 😵						
	SFI		SMI	s	FU	HSM
Firmware	files			Firmware	e information	SFI information
tests.a>	f		Add Remove	Overview		
Encryption M32TrustedP	1 key file ackageCreator/Input/	SFI/good/test_firmware_l	ey.bin Open	File name tes Type EL	its.axf F	
Nonce file				Size 81	5.887 KB	
/projects/STI	M32TrustedPackageCr	eator/Input/SFI/good/no	oce.bin Open	Index	Size	Address
Option by	tes file			1	844 B	0x8000000
projects/SFM	II-PreparationToolv0.2	.0_test1/Input/SFI/good,	ob.csv Open	2	9884 B	0x8030000
SMI files (	Only for combined	case)				
STM32	F4-DISCO0.smi		Add Remove			
Image ver	rsion					
RAM size		Continuation token ad	ldress			
Output SF	I file					
C:/projects/S	STM32TrustedPackage	Creator/output/out.sfi	Select folder			
						Generate SFI

#### Figure 40. 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.



# 3.7.6 Log generation

A log can be visualized by clicking the "log" icon in the tool bar or menu bar: Options-> log. *Figure 41* shows a log example:

😘 STM32 Trusted Package Creator		
File Edit Op	tions Help	
3		
SFI	Log	ням
ELF file	09:15:06:674 SFI preparation started 09:15:06:788 Area 1 prepared with size 844 : firmware area 09:15:06:788 Area 2 prepared with size 9884 : firmware area	SMI information
ckage_Creator_v1.0.2/Input/SMI/good/MDK	09: 15:06:788 Area 3 prepared with size 36 : option bytes area 09: 15:06:788 SFI header prepared	
Encryption key file	US: LISUO: 745 S-1 preparation hinshed 10: 18: 27: 531 SMI preparation started 10: 18: 27: 531 SMI data prepared with size 1640 10: 18: 27: 532 SMI data prepared with size 1640	
]_Package_Creator_v1.0.2/Input/SMI/good,	10:18:27:532 SMI reparation finished	
Nonce file		<b></b>
M32_Trusted_Package_Creator_v1.0.2/Inp		
Security version file		
\$/STM32_Trusted_Package_Creator_v1.0.2		
Section to encrypt		
ER_PCROP		
Output SMI file		
STM32_Trusted_Package_Creator_v1.0.2/		
Output clear ELF file		
2_Trusted_Package_Creator_v1.0.2/output/F	R_module_dear.axf Select folder	
		Generate SMI

Figure	41.	Loa	examp	le
i igui c	<b></b> • •	LUG	CAUIIP	



# 3.7.7 SFI and SMI file checking function

This function checks the validity and information parsing of an SFI or SMI file.

It is accessed by clicking the Check SFI/SMI button in the tool bar or the menu bar: File -> Check SFI/SMI.

Figure 42 shows a check SFI example:

Figure	42.	Check	SFI	file	example
i igaio		011001	••••		onampio

TM32 Trusted	Package Creator				
File	Edit	Options	Help		life.augmented
🤞 😵					
	SFI	2	IMI	SFU	ням
Firmware	identifier			HSM information	
				Firmware ID	<u> </u>
Encryption	n key file			HSM status	
			Open		Refresh
Nonce file				-	
			Open		
				-	
	Counter				
				-	
Set HSM	I to operational state	(HSM will be locked)			
					Program HSM



# 4 Encrypted firmware (SFI/SFIx)/ module (SMI) programming with 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 an ST-LINK probe.

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

# 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 control the number of devices to be programmed, since each license is specific to a unique chip, identified by its public key.

#### License mechanism general scheme

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

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

ST offers an SFI solution based on smartcard HSMs as a license distribution tool, which can be used in programming houses.

## HSM programming by OEM for license distribution

Before an OEM delivers 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 be done first.

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 HSM personalization and provisioning.

This data is as follows:

• **The counter**: the counter is set to a maximum value that corresponds to the maximum number of licenses that 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 depends on the HSM used.

• **The firmware key**: the key size is 32 bytes. It is composed of two fields: the initialization vector field (IV) and the key field, which are used for AES128-GCM firmware encryption.

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

Both fields must remain secret; that'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 personalization dat**a: this is specific to each MCU and delivered inside TPC directory. More info about personalization data in *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.



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/SFIx via HSM. In fact, HSM programming is supported by the STM32 Trusted Package Creator tool. *Figure 43* shows the GUI for HSM programming in STPC tool.

STM32 Trusted	Package Creator								_	
File	Edit	Options	Help						57	life.augme
6 😵	ی 😂									
					1					
9	FI	SFIx		SSP	WB	SIGN	SMI	HSM		
HSM card i	ndex					HSM informa	tion			
þ ÷						Firmware ID				
						Max counter				
Firmware i	dentifier					HSM status				
						Version				
Encryption	key file					Туре			_	
C:/SFIx/key.t	pin				Open				Clear	Refresh
Nonce file										
C:/SFIx/nonc	e.bin				Open					
Personaliza	ation data file									
					Open					
Maximum	counter									
0 ÷										
									Progr	am HSM

## Figure 43. HSM programming GUI in the 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.

# 4.2 Secure programming using a bootloader interface

## 4.2.1 Secure firmware installation using a 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 such devices as well as all STM32H7, STM32L4, STM32L5, STM32WL, STM32U5 and STM32MP devices available so far.

For more details on SFI over these STM32 devices refer to AN4992 [1]. This document is available on *www.st.com*.



AN5054 Rev 9

The general flow of the secure firmware installation using a bootloader interface on a chip for H7 and L4 secure devices is shown respectively in *Figure 44* and *Figure 45* below.

Figure 44. Secure programming via STM32CubeProgrammer overview on STM32H7 devices









## 4.2.2 Secure Module installation using a bootloader interface flow

As explained in *Section 3.4: 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 is 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 SMI has to be invoked via the secure bootloader.

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 a bootloader interface

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

#### -sfi, --sfi

Syntax: -sfi protocol=<Ptype> <file\_path> <licenseFile\_path>

[ <protocol=ptype>]</protocol=ptype>	: Protocol type to be used : static/live Only static protocol is supported so far
	Default value static
<file_path></file_path>	: Path of sfi file to be programmed
[hsm=0 1]	: Set user option for HSM use value in {0 (do not use HSM), 1 (use HSM)} Default value : hsm = 0
<lic_path slot=slotid></lic_path slot=slotid>	: Path to the SFI license file (if hsm = 0) or reader slot ID if HSM is used (hsm = 1)
[ <licmod_path> slot=slot</licmod_path>	ID] : List of the integrated SMI license files paths

If hsm = 1, the user must provide the Slot ID parameter.

If hsm = 0, the user must provide the license path file that can be generated separately using the following command line, provided an HSM card is available:

-hsmgetlicense

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



Example using UART bootloader interface:

```
STM32_Programmer.exe -c port=COM1 br=115200 -sfi "C:\SFI\data.sfi"
hsm=1 "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 a 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>] licenseFile\_path>

<protocol=ptype></protocol=ptype>	: Protocol type to be used : static/live Only static protocol is supported so far Default value static
<file_path></file_path>	: Path of smi file to be programmed
[hsm=0 1]	: Set user option for HSM use value in {0 (do not use HSM), 1 (use HSM)} Default value: hsm=0
[ <address>]</address>	: Destination address of the smi module needed only for relocatable SMI
<lic_path slot=slotid></lic_path slot=slotid>	: Path to the SMI license file (if hsm=0) or reader slot ID if HSM is used (hsm=1)

Example using UART bootloader interface:

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

This command allows programming of the SMI specified file *"data.smi"* into a dedicated PCROPed area.



## 4.2.5 STM32CubeProgrammer for SSP via a bootloader interface

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

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

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

-ssp, --ssp

Description: Program an SSP file

**Syntax**: -ssp <ssp\_file\_path> <ssp-fw-path> <hsm=0|1> <license\_path|slot=slotID>

<ssp\_file\_path> : SSP file path to be programmed, bin or ssp extensions

<ssp-fw-path> : SSP signed firmware path

<hsm=0|1> : Set user option for HSM use (do not use HSM / use HSM)

Default value : hsm=0

license\_path|slot=slotID> :Path to the license file (if hsm=0)

Reader slot ID if HSM is used (if hsm=1)

Example using USB DFU bootloader interface:

STM32\_Programmer\_CLI.exe -c port=usb1 -ssp "out.ssp" "tf-a-sspstm32mp157f-dk2-trusted.stm32" hsm=1 slot=1

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



Figure 46	. SSP	install	success
-----------	-------	---------	---------

```
Requesting Chip Certificate...
Get Certificate done successfully
requesting license for the current STM32 device
Init Communication ...
ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
P11 lib initialization Success!
Opening session with solt ID 1...
Succeed to Open session with reader solt ID 1
Succeed to generate license for the current STM32 device
Closing session with reader slot ID 1...
Session closed with reader slot ID 1
Closing communication with HSM...
Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1
Starting Firmware Install operation...
Writing blob
Blob successfully written
Start operation achieved successfully
Send detach command
Detach command executed
SSP file out.ssp Install Operation Success
```

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



## 4.2.6 STM32CubeProgrammer get certificate via a bootloader interface

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

#### -gc, --getcertificate

**Syntax**: –gc <file\_path>

Example using 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 47.



# 4.3 Secure programming using JTAG/SWD interface

#### 4.3.1 SFI/SFIx programming using JTAG/SWD flow

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

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

SFI via debug interface is currently supported for STM32H753xI, STM32H7A3/7B3 and STM32H7B0, STM32H723/333 and STM32H725/335, and STM32L5 devices.

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

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

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

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



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

The SFI flow for programming through JTAG is described in *Figure 48*.

Figure 48. SFI programming	by JTAG/SWD flow overview
(monolithic SFI	image example)

Preparing programmation
5 write license to RAM
6 write image header to RAM
100p [for each areas in image] 7 write area header to RAM
8 write area payload to RAM
9 write global header with links to all the parts to RAM
Flashing areas
10 decrypt license
11 authenticate image header
[for each areas in global header]  12 authenticate area header
13 decrypt area payload and flash it
Finishing
14 wait until it's finished
SEI succase



## 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.

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

- write a small program in RAM that calls the public API (API details are available under non-disclosure agreement)
- use the secure API directly.

The essential steps of the SMI programming by JTAG flow are described in *Figure 49: SMI programming by JTAG flow overview*.





Figure 49. 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 "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 50.

#### Figure 50. Example of getcertificate command using JTAG

ST-LINK Firmware version : U2J28S6 JTAG frequency = 9000 KHz Connection mode: Hot Plug Device ID: 0x450 Certificate File : testJTAG\_Certif.bin Requesting Chip Certificate using debug interface... Get Certificate done successfully ...Writing data to file testJTAG\_Certif.bin Writing chip certificate to file testJTAG\_Certif.bin finished successfully Imme elapsed during the getcertificate operation is: 00:00:00.032

## Example "smi" command using SWD

-c port=swd mode=HOTPLUG -smi protocol=static "RefSMI MDK/FIR module.smi" "RefSMI MDK/licenseSMI.bin" -vb 3 -log



# 4.4 Secure programming using Bootloader interface (UART/I2C/SPI/USB)

It is also possible to program the SFI/SFIx image using the Bootloader interface (UART/I2C/SPI/USB). FDCAN is not supported by STM32CubeProgrammer since it not managed by ST-Link v3.

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

SFI via the Bootloader interface (UART/I2C/SPI/USB) is currently supported for STM32L5 devices. It needs to load an external loader using the **-elbi** command in the SRAM.

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

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

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

## SFI example

STM32\_Programmer\_CLI.exe -c port=usb1 -sfi out.sfix hsm=0 license.bin -rsse RSSe\L5\enc\_signed\_RSSe\_sfi\_bl.bin

#### SFIx example

STM32\_Programmer\_CLI.exe -c port=usb1 -elbl MX25LM51245G\_STM32L552E-EVAL-SFIX-BL.stldr -sfi out.sfix hsm=0 license.bin -rsse RSSe\L5\enc signed RSSe sfi bl.bin


# 5 Example of SFI programming scenario

## 5.1 Scenario overview

The actual user application to be installed on the STM32H753xl (or STM32L5) device makes "printf" packets appear in serial terminals. The application was encrypted using the STPC.

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

# 5.2 Hardware and software environment

For successful SFI programming, some and SW prerequisites apply:

- STM32H743I-EVAL board
- STM32H753xI with bootloader and RSS 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 STM32 Trusted Package Creator tool, OEM firmware is provided in AXF format in addition to a CSV file to set the option bytes configuration. A 128-bit AES encryption key and a 96-bit nonce are also provided to the tool. They are available in the *"SFI\_ImagePreparation"* directory.

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

*Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES

The option bytes are described in the product reference manual.

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

*Figure 51: STPC GUI during SFI generation* shows the STPC GUI during the SFI generation.



M32 Trusted	Package Creator									
File	Edit	Options	Help						ugment	
6 😵	5									
	SFI		SMI			SFU		нѕм		
Firmware	files				Firm	ware information		SFI information		
tests.ax	f		Add Remove		Overview	v				
Encryption	ı kev file				File name	out_toto	1.sfi	a		
J_Package_C	reator_v1.0.2/Input/Si	FI/good/test_firmware	key.bin Open		Size	10.627 K	3			
Nonce file M32_Trusted	l_Package_Creator_v1	.0.2/Input/SFI/good/n	Success	essfu	Illy created	3   01 1			<b>-</b>	
Ontion byt	tes file					Туре	Size	Address	_	
/STM32 Trus	ted Package Creator	v1.0.2/Input/SFI/good		<		rmware	844 B	0x8000000	_	
				-	2	Firmware	9884 B	0x8030000	_	
SMI files (	Unly for combined c	ase)			3	Configuration	36 B	0x0		
			Remove							
Image ver	rsion									
RAM size		Continuation token a	address							
Output SFI	, I file		1							
loads/STM32	_Trusted_Package_Cre	eator_v1.0.2/output/ou	ut_toto1. Select folder							
								General	te SFI	

#### Figure 51. 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 52 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.

This example uses HSM version 2, and is also valid for version 1 when the 'Version' field is set accordingly. The HSM version can be identified before performing the programming operation by clicking the Refresh button to make the version number appear in the 'Version' field.

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

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

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

- Use the STM32CubeProgrammer tool to launch a Get Certificate command to generate a certificate file containing some chip security information, bearing in mind that this command is only recognized only for devices that support the security feature: STM32\_Programmer\_CLI -c port=swd -gc "certificate.bin" A file named "certificate.bin" is created in the same path of the STM32CubeProgrammer executable file.
- Open the certificate file with a text editor tool, then read the 8 characters from the header which represents 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 differentiate the personalization package to be used on the HSM provisioning step respecting the following naming convention:

ProdcutID\_FlowType\_LicenseVersion\_SecurityVersion.enc.bin

For example: 47201003\_SFI.\_01000000\_0000000.enc.bin

Based on this name we can retrieve the associated information:

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



File	Edit	Options	Help					ile.augmen
6 😵	<b></b>							
	SFI		SFIx			SMI	нѕм	
HSM card ind	ex			HS	M informati	on		
1 -				Fir	mware ID	HSMv2_SLOT_1		
				Ma	ax counter	1000		
Firmware ide	entifier			HS	M status	OPERATIONAL_STATE		
HSMv2_SLOT_	1			Vei	rsion	2		
Encryption ke	ey file			Ту	pe	SFI	Clear	Refresh
C:/TrustedFiles	/key.bin		Open					
Nonce file								
C:/TrustedFiles	/nonce.bin		Open					
Personalizati	on data file							
C:/TrustedFiles	/enc_ST_Perso_L5.bin		Open					
Maximum cou	inter							
1000								
							Pr	ogram HSM

## Figure 52. Example of HSM programming using STPC GUI

Note: When using HSM version1, the "Personalization data file" field is ignored when programming starts. It is only used with HSM version 2.

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



# 5.3.4 Performing HSM programming for license generation using STPC (CLI mode)

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

#### 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-mc2000

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

HSMv2 allows users to personalize their own HSM to achieve, for example, compatibility with the desired STM32 device. This solution covers the limitation of HSMv1 (static behavior), so it is possible to support new devices that are not available on HSMv1.

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

STM32\_Programmer.exe -c port=COM1 -gc "C:\SFI\Certificate.bin"

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

00000000	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	34	39	37	30	31	30	30	35	07	d7	60	65	98	2a	fe	36	4 <mark>9701005.</mark> ×`e~*þ6
00000010	29	ca	59	£3	d5	29	9b	99	f7	a3	4e	c0	bb	15	5f	dl	)ÊYóÕ) >™÷£NÀ»Ñ
00000020	ld	82	f4	8a	9a	13	2d	d3	c9	2a	9a	02	c0	9b	db	10	.,ôŠš.−ÓÉ*š.À>Û.
00000030	fc	2d	28	d9	c9	77	bc	4c	ba	38	5b	15	e5	b0	8d	bd	ü−(ÙÉw¼L°8[.å° ½
00000040	d0	4d	c3	4a	e9	dl	24	6b	a8	fc	3f	51	af	42	41	dd	ÐMÃJÉÑ\$k¨ü?Q¯BAÝ
00000050	be	b3	e4	bb	77	48	14	fa	4b	d6	3b	bb	67	44	e5	al	¾³ä≫wH.úKÖ;≫gDå;
00000060	63	ca	76	6b	db	a3	80	cf	e0	61	f3	01	07	05	dd	6c	cÊvkÛ£€ÏàaóÝl
00000070	74	f6	29	23	17	8f	bd	e7	c5	cb	3a	5c	0e	5b	58	a3	tö)#. ½çÅË:\.[X£
00000080	8c	dc	8d	13	97	le	ab	52									ŒÜ .—.«R
00000090		• •	• •	• •	• •	• •	• •		• •	• •	• •			• •	• •		

Figure 53. Example product ID

The product ID of the STM32WL used is: 49701005

In the second step the user provisions their own HSMv2 by programming it using STPC. The personalization data file .bin can be found under "..\bin\PersoPackages".



#### Example of HSM version 2 provisioning

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

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

-pd: Set the personalization data file path.

To obtain the appropriate personalization data file and for further information, refer to *Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)*.

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

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

HSM v1 supports a list of limited number of STM32 devices such as STM32L4, STM32H7, STM32L5, and STM32WL.

#### Example of HSM get information

If the HSM is already programmed or is virgin yet and whatever the version, a get information command can be used to show state details of the current HSM by using the command below:

STM32TrustedPackageCreator\_CLI -hsm -i 1 -info

```
Figure 54. HSM information in STM32 Trusted Package Creator CLI mode
```

STM32TrustedPackageCreator v1.2.0 ldm\_LoadModule(): loading module "stlibp11\_SAM.dll" ... ldm\_LoadModule(WIN32): OK loading library "stlibp11\_SAM.dll": 0x71CB0000 .... C\_GetFunctionList() returned 0x00000000, g\_pFunctionList=0x71D2F560 Read the following Information from HSM slot 1 : HSM STATE : OPERATIONAL\_STATE HSM FW IDENTIFIER : HSMv2\_SLOT\_2 HSM COUNTER : 2000 HSM VERSION : 2 HSM TYPE : SFI



## 5.3.5 Programming input conditions

Before performing an SFI install make sure that:

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



## 5.3.6 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>

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

#### The STM32CubeProgrammer command is as follows:

STM32\_Programmer\_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local\_path>/out.sfi" hsm=1 slot=<slot\_id> -rsse <RSSe\_path>



*Figure 55* shows the SFI install via SWD execution and the HSM as license generation tool in the field.

STM320	ubeProgrammer v1.0.7
ST-LINK SN: 0672FF55494967706 ST-LINK Firmware version: U2J Target voltage: 3.21V SWD frequency: 4000 KHz Connection mode: Hot Plug Device ID: 0x450	7034831 30M19
Device name: STM32H7xx Device type: MCU Device CPU : Cortex-M7/M4 Protocol Information	: static
SFI File Information	:
SFI file path SFI ID SFI header information SFI protocol version SFI total number of SFI image version SFI Areas information	: out_EH.sfi : 111 : areas : 3 : 23 :
Parsing Area 1/3 : Area type Area size Area destination add	: F : 844 ress : 0x800000
Parsing Area 2/3 : Area type Area size Area destination add	: F : 10528 ress : 0x8030000
Parsing Area 3/3 : Area type Area size Area destination add	: C : 36 ress : 0x0
Reading the chip Certificate.	
Requesting Chip Certificate u Get Certificate done successf	sing debug interface ully
Requesting Licesne for firmwa	re with ID : 111
requesting license for the cu	rrent STM32 device
Init Communication	
ldm_LoadModule(): loading mod ldm_LoadModule(WIN32): OK loa C_GetFunctionList() returned Init Communication with slot	ule "stlibp11_SAM.dll" ding library "stlibp11_SAM.dll": 0x5FC00000 0x00000000, g_pFunctionList=0x5FCC8A78 2 Success!
Succeed to generate license f	or the current STM32 device
Closing communication with HS	М
Communication closed with HSM	
Succeed to get License for Fi	rmware with ID 111
Starting Firmware Install ope	ration
Activating security Warning: Option Byte: SECURIT Warning: Option Bytes are unc Activating security Success Setting write mode to SFI Warning: Option Byte: SECURIT Warning: Option Byte: ST_RAM_ Succeed to set write mode for Starting SFI part 1	Y, value: Øx1, was not modified. hanged, Data won't be downloaded Y, value: Øx1, was not modified. SIZE, value: Øx3, was not modified. SFI
Writing license to address Øx Writing Img header to address Writing areas and areas wrapp all areas processed RSS process started	24030800 Øx24031000 er
RSS command execution OK	

Figure 55. SFI install success using SWD connection (1)





RSS command execution OK Reconnecting... ST-LINK SN: 0672FF554949677067034831 ST-LINK Firmware version: U2J30M19 Target voltage: 3.21U Error: ST-LINK error (DEU\_NO\_DEVICE) ...retrying... ST-LINK SN: 0672FF554949677067034831 ST-LINK SN: 0672FF54949677067034831 ST-LINK SN: 0672FF5494678 ST-LINK SN: 0672FF55494678 ST-



# 6 Example of SFI programming scenario for STM32WL

# 6.1 Scenario overview

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

- the encrypted firmware of STM32WL
- HSMv1 or previsioned HSMv2
- STM32CubeProgrammer.

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

# 6.2 Hardware and software environment

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

- STM32WL5x board with Bootloader and RSS programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running on either Windows or Ubuntu 14 (in both 32-bit and 64-bit versions) or macOS
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v2.6.0 (or greater) package available from www.st.com
- HSMv1 or HSMv2.

# 6.3 Step-by-step execution

## 6.3.1 Build OEM application

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

## 6.3.2 Perform the SFI generation (GUI mode)

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

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

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



*Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES The option bytes are described in the product reference manual.

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

A programmed HSM card should be insert in the PC, and an *"out.sfi"* image is then generated.

#### Figure 57. STPC GUI showing the STPC GUI during the SFI generation

File	Edit	Options	Help						
18	5								
SFI	SFD	¢ 55	IP WB SI	GN	5M1		н5М		
Firmware	files			Firm	ware infor	mation	5	F1 Information	
LPUAR	T_WakeUpFromStop.h	e:	Add Remove	Overvie	w				
Encryption	i key file			File nam	e	out.sfi		*	
C:/testBee/k	ev.bin	QL Success		Size	Size		10.4 KB		
Nonce file		0	SFI successfully created	Protocol	Protocol version 01			•	
C:/testBee/n	ionce.bin		ок	Segmen	ts	1		1	
Option by	tes file	07		Index	Туре		Size	Address	
C:/STM32WL	./ob_wl_default.csv		Open	1 Contraction			10544.6	0.0	
SMI files (	Only for combined ca	ise)		e .	Contigur	auon	40 B	1010	
		*****	Add Remove						
Image ver	rsion								
RAM size	0x5000 Cd	ontinuation token ad	Iress 0x8000000						
Output SF	l file			Parse SI	I file				
C:/STM32WL	,/out.sfi		Select folder	-				Browse	

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

# Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)

Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)



## 6.3.3 Programming input conditions

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

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

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

#### Example:

STM32\_Programmer\_CLI.exe -c port=swd mode=hotplug -dsecurity

Figure 58 hows the resulting output on the command line.

#### Figure 58. Example -dsecurity command-line output

C:\Windows\System32\cmd.exe	-		×
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0 \bin>STM32_Programmer_CLI.ex mode-hotplug -dsecurity	e-c	port-s	wd
STM32CubeProgrammer v2.6.0			
<pre>mode-hotplug -dsecurity</pre>			
Reset mode : Software reset Device ID : 8x497 Revision ID : Rev 1.1			
Reconnected [ Apply Power Off/ON to disable the security			v



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

#### Example:

STM32\_Programmer\_CLI.exe -c port=swd mode=hotplug -setdefaultob

Figure 59 shows the resulting output on the command line.



C:\Windows\System32\cmd.exe	-		$\times$
C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\v2.6.0 \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	-с	port-s	nd
STM32CubeProgrammer v2.6.0			
ST-LINK SN : 002300263038511234333935 ST-LINK FW : V335M2 Board : STM32NL55C-DK Voltage : 3.31V SMD freq : 12000 KHz Connect mode: Hot Plug Reset mode : Software reset Device ID : 0x497 Revision ID : Rev 1.1 Device name : STM32NLxx Flash size : 256 KBytes Device type : NCU Device CPU : Cortex-M4			
Set default 08 for STM32WL Reconnecting ST-LINK SN : 002300263038511234333935 ST-LINK FW : V3J5M2 Board : STM32WL55C-DK Voltage : 3.31V SWD freq : 12000 KH2 Connect mode : Hot Plug Reset mode : Hot Plug Reset mode : Software reset Device ID : 0x497 Revision ID : Rev 1.1 Reconnected ! Apply Power 0W/Off to set default 08 for STM32WL			

## 6.3.4 Perform the SFI install using STM32CubeProgrammer

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

STM32CubeProgrammer supports communication with 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
"<local path>/out.sfi" hsm=1 slot=<slot id> -rsse "< RSSe path >"

Note: The RSSe and its binary file is located in the STM32CubeProgrammer bin/RSSe/WL folder. Figure 60 shows the SFI install via SWD execution.



Select C//Window/System32/cmd.exe	- 🗆 X	6
C:\Program Files\STHicroolectronics\STH32Cube\STH32CubeProgrammer\v2.6.0-A05\bin>STH32_Programmer_CLT.er	xe ∘c port-swd mo	~
de-hotplug -sfi "C:\testPlan \tesssst\Sfi%.sfi" ham=0 "C:\testPlan \tesssst\License .bin" -rsse #_sfi_V4.6.0.out.bin"	e "C:\RSSe\WL\RSS	
5TR32CubeProgrammer v2.6.0		
5T-LINK SN : 50FF0E007205575458302007 5T-LINK FW : V233657		
Board : Voltage : 3.24V SWD freq : 4000 KNL Connect mode: Hot Plug Reset mode : Software reset Device ID : 8xW 1.1 Device nome : STNSZHUKK Flash size : 256 KBytes Device type : MCU		
Device CPU : Cortex-M4		
SIT File Information		
SFI file path       i C:\testPlan         SFI license file path       : C:\testPlan       \tessssst\License         SFI header information       :         SFI protocol version       : 1         SFI total number of areas       : 2         SFI Areas information       : 0		
Parsing Area 1/2 : Area type : F Area size : 10544 Area destination address : 0x8000000		
Parsing Area 2/2 ; Area type : C Area 5120 : 40 Area destination address : 800		
Installing HSSe		
Memory Programming Opening and parsing file: RSSe_sfi_V4.6.0.out.bin file : RSSe_sfi_V4.6.0.out.bin Size : 24032 Bytes Address : 0x20002020 Erasing memory corresponding to segment 0:		2
Download in Progress:		1
<pre>file download complete Time elapsed during download operation: 00:00:00:00 Boot on HSS Recommenting ST-LINK FW : V233057 Bobind : Voltage : 3.24V SAD freq : 4000 KHz Connect mode: Hut Plug Reset mode : Software reset Oevice ID : 0x407 Revision ID : Rev 3.1 Recommental ! MSSE SPI INITIALISED Net HSSE Version HSSE version : 0x40000 Starting SFI</pre>		
Processing license RSSE SF1 LICENSE OK Processing Tange Header RSSE SF1 DEADE windown om Processing Area 1 RSSE SF1 AREA ON Area Address = 0x0000000 Area Type - F Processing Area 2 Can not verify last area Area Address = 0x0 Area Type - C SF1 Process Finished! SF1 Frocess Finished! SF1 File CitestPlanBeo(tassesst)SF13.ef1 Install Operation Success		
Time elapsed during SFI install operation: 00:00:15.415		

Figure 60. SFI install via SWD execution command-line output



AN5054 Rev 9

# 7 Example of SFI programming scenario for STM32U5

# 7.1 Scenario overview

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

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

# 7.2 Hardware and software environment

For successful SFI programming, some HW and SW prerequisites apply:

- STM32U5 board with bootloader and RSS programmed
- RS232 cable for SFI programming via UART
- Micro-USB for debug connection
- PC running either on Windows, Ubuntu 14 (64-bit version) or macOS.
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer v2.8.0 (or greater) package available from www.st.com
- HSMv2.

# 7.3 Step-by-step execution

## 7.3.1 Build OEM application

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

## 7.3.2 Perform the SFI generation (GUI mode)

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

- an OEM firmware
- a.csv file containing option byte configuration
- a128-bit AES encryption key
- a96-bit nonce
- *Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES The option bytes are described in the product reference manual.

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

In addition, a programmed HSM card should be insert in the PC. An "*out.sfi*" image is then generated.



Figure 61 shows STPC GUI during SFI generation.

STM32 Trus	ed Package Crea	tor									-
File	Edit	Options	Help								life.augm
🤞	3 📑 🕄										
SFI		SFIx	SSP	WBS	SIGN		SMI		нѕм		
Firmware f	iles					Firmware	informatio	n		SFI information	
GPIO	EXTI.hex			Add							
				Remove	0ven	iew					
					File	ame	out.sfi				
					Size		5.78 K	В			
Encryption	key file				Segm	ents	1				
C:/STM32U5/	key.bin			Open	Inde		e	Size		Address	
Nonce file			🔅 Suce	cess		×	are	5784 B		0x8000000	
C:/STM32U5/	nonce.bin			SFI succ	essfully c	reated	ation	48 B		0x0	
Option byt	es file										_
C:/STM32U5/	ob_bk1s_bk2ns_tzen_r	dp_0_5.csv									
SMI files (	only for combined ca	ase)									
				Add							
				Remove							
Image ver	sion										
0 📩											
RAM size		Continuation t	oken address								
Output SFI	file				Parse	SFI file					
C:/STM32U5/	out.sfi		Selec	t folder	1						Browse
										Cono	rate SFT
										Gene	and SIT

Figure 61. STPC GUI during the SFI generation
---

Note: To perform HSM programming for license generation using STPC (GUI and CLI modes), refer to Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode) and Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode).

## 7.3.3 Programming input conditions

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

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



## 7.3.4 Perform the SFI install using STM32CubeProgrammer

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

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

#### Using JTAG/SWD

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

STM32\_Programmer\_CLI.exe -c port=swd mode=HOTPLUG -sfi
"<local path>/out.sfi" hsm=1 slot=<slot id> -rsse "< RSSe path >"

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

*Figure 62* and *Figure 63* show the STM32CubeProgrammer command used for the SFI install process via SWD execution.



<u> </u>
Reconnecting Reconnected !
Reading chip Certificate finished
Get Certificate done successfully
requesting license for the current STM32 device
Init Communication
<pre>ldm_LoadModule(): loading module "stlibp11_SAM.dll" ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x059D0000 C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x05A32FD8 P11 lib initialization Success!</pre>
Opening session with solt ID 1
Succeed to Open session with reader solt ID 1
Succeed to generate license for the current STM32 device
Closing session with reader slot ID 1
Session closed with reader slot ID 1
Closing communication with HSM
Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1
Starting Firmware Install operation
Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded
Reconnecting Reconnected ! Installing RSSe
Memory Programming Opening and parsing file: enc_signed_RSSe_sfi_bl_cut2.bin File : enc_signed_RSSe_sfi_bl_cut2.bin Size : 34464 Bytes Address : 0x20040300
Erasing memory corresponding to segment 0: Download in Progress:
File download complete Time elapsed during download operation: 00:00:00.200

#### Figure 62. SFI install via SWD execution (1)



Reconnecting Reconnected !	
Reading chip Certificate finished	
Get Certificate done successfully	
requesting license for the current STM32 device	
Init Communication	
<pre>ldm_LoadModule(): loading module "stlibp11_SAM.dll" ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x059D0000 C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x05A32FD8 P11 lib initialization Success!</pre>	
Opening session with solt ID 1	
Succeed to Open session with reader solt ID 1	
Succeed to generate license for the current STM32 device	
Closing session with reader slot ID 1	
Session closed with reader slot ID 1	
Closing communication with HSM	
Communication closed with HSM	
Succeed to get License for Firmware from HSM slot ID 1	
Starting Firmware Install operation	
Warning: Option Byte: SECWM1_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECWM1_PSTRT, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded Warning: Option Byte: SECWM2_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECWM2_PSTRT, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded	
Reconnecting Reconnected ! Installing RSSe	
Memory Programming Opening and parsing file: enc_signed_RSSe_sfi_bl_cut2.bin File : enc_signed_RSSe_sfi_bl_cut2.bin Size : 34464 Bytes Address : 0x20040300	
Erasing memory corresponding to segment 0: Download in Progress:	
File download complete Time elapsed during download operation: 00:00:00.200	

#### Figure 63. SFI install via SWD execution - (2)



# 8 Example of SMI programming scenario

## 8.1 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.

# 8.2 Hardware and software environment

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

# 8.3 Step-by-step execution

# 8.3.1 Build 3<sup>rd</sup> 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.



# 8.3.2 Perform the SMI generation

For encryption with the STM32 Trusted Package Creator 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 64* shows the STPC GUI during SMI generation.

STM32 Trusted	Package Creator							
File	Edit	Options	Help					gmente
k 🤞 😵	) 👼 😣							
	SFI		SMI		SFU		нѕм	
ELF file					ELF inform	ation	SMI information	
ted_Package	e_Creator_v1.0.2 (1)/Inp	out/SMI/good/FIR_mod	lule.axf Open		Overview			
Encryptio	n key file				Original file name	FIR_module.smi		-
ckage_Crea	ator_v1.0.2 (1)/Input/SM	I/good/test_firmware_	key.bin Open		Number of files	1		
Nonce file	1				Last file size SMI address	1.64844 KB		_
2_Trusted_P	ackage_Creator_v1.0.2	(1)/Input/SMI/good/no	nce.bin 🔄 Informati	ion	X			
Security	version file			SM	I successfully created			
M32_Truste	ed_Package_Creator_v1.	0.2 (1)/Input/SMI/goo	d/svFile		ок			
Section to	o encrypt							
ER_PCROP	•			_				
Output SM	MI file							
M32_Trusted	d_Package_Creator_v1.0	).2 (1)/output/FIR_mod	dule.smi Select folde					
Output clo	ear ELF file							
rusted_Pack	(age_Creator_v1.0.2 (1)	/output/FIR_module_d	ear.axf Select folde	r				
							Generate	sMI

Figure 64. STPC GUI during SMI generation



## 8.3.3 **Programming input conditions**

Before performing the SMI install make 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 bootloader, no debug interface is connected to any USB host. If a debug interface is still connected, disconnect it then perform a power off/power on before launching the SMI install to avoid any debug intrusion problem.
- The proper license generated for the currently-used chip must be at your disposal (or an HSM or secure server to generate it during SMI programming).

### 8.3.4 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:

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 65: SMI install success via debug interface shows the SMI install via SWD execution.



#### Figure 65. SMI install success via debug interface

23 - 0 Administrator: C:\Windows\system32\cmd.exe Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved. . C:\Users\hannachi>cd C:\Users\hannachi\Documents\Projects\SIM32H7\_project\docs\d ocs\_forSTM32H7Release\AN\Sw\_packages\stm32\_programmer\_package\_v0.4.0 C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\docs\_forSTM32H7Release \AN\Sw\_packages\stm32\_programmer\_package\_v0.4.0>cd bin C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\docs\_forSTM32H7Release \AN\Sw\_packages\stm32\_programmer\_package\_v0.4.0\bin>STM32\_Programmer\_CLI.exe -c port=swd mode=HOTPLUG -smi protocol=static "C:\Users\hannachi\Documents\Projects \STM32H7\_project\docs\docs\_forSTM32H7Release\AN\SMI\_ImagePreparation\FIR\_module. smi" "C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\docs\_forSTM32H7R elease\AN\SMI\_ImagePreparation\licenseSMI.bin" STM32CubeProgrammer v0.4.0 ST-LINK Firmware version : V2J27M15 SWD frequency = 4000 KHz Connection mode: Hot Plug Device ID: 0x450 Protocol : static SMI File : C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\ docs\_forSTM32H7Release\AN\SMI\_ImagePreparation\FIR\_module.smi Starting SMI install operation for file : C:\Users\hannachi\Documents\Projects\ SIM32H7\_project\docs\docs\_forSIM32H7Release\AN\SMI\_ImagePreparation\FIR\_module.s SMI File Information SMI file path: C:\Users\hannachi\Documents\Projects\STM32H?\_project\docs\docs\_forSTM32H7Release\AN\SMI\_ImagePreparation\FIR\_module.smiSMI license file path: C:\Users\hannachi\Documents\Projects\STM32H?\_project\docs\_forSTM32H7Release\AN\SMI\_ImagePreparation\licenseSMI.binSMI code destination address: 0x8080000SMI code size: 1688 Setting write mode to SMI Succeed to set write mode for SMI Writing license C address 0x24050000... License file successfully written at adress 0x24050000 Writing SMI module image to address 0x24050088... SMI image successfully written at address 0x24050088 Starting SMI process with license @ 0x24050000 and image @ 0x24050088... RSS process started... OR execution Reconnecting... ST-LINK Firmware version : V2J27M15 SWD frequency = 4000 KHz Connection mode: Hot Plug Device ID: 0x450 Reconnected ! Requesting security state... MI\_SUCCESS: MI\_file\_C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\docs\_forSTM32 |7Release\AN\SMI\_ImagePreparation\FIR\_module.smi\_Install\_Operation\_Success Time elapsed during the SMI install operation is: 00:00:03.294 C:\Users\hannachi\Documents\Projects\STM32H7\_project\docs\docs\_forSTM32H7Release \AN\Sw\_packages\stm32\_programmer\_package\_v0.4.0\bin>



## 8.3.5 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 66).

#### Figure 66. OB display command showing that a PCROP zone was activated after SMI

Read Out Protection: RDP : 0xAA (Level 0, no p <u>rotection</u> )
RDP : ØxAA (Level Ø, no protection)
RSS :
RSS1 : ØxØ (No SFI process on going)
BOR Level:
BOR_LEU : 0x0 (reset level is set to 2.1 V)
User Configuartion:
IWDG1 : Øx1 (Independent watchdog is controlled by hardware)
NRST_STOP_D1 : 0x1 (STOP mode on Domain 1 is entering without reset)
NRST_STBY_D1 : 0x1 (STANDBY mode on Domain 1 is entering without reset)
FZ_IWDG_STOP : Øx1 (Independent watchdog is running in STOP mode)
$FZ_IWDG_SDBY$ : $0 \times 1$ (Independent watchdog is running in STANDBY mode)
SECURITY : 0x1 (Security feature enabled)
BCM7 : 0x1 (CM-7 boot enabled)
NRST_STOP_D2 : $0 \times 1$ (STOP mode on Domain 2 is entering without reset)
NRST_STBY_D2 : 0x1 (STANDBY mode on Domain 2 is entering without reset)
SWAP_BANK : 0x0 (after boot loading, no swap for user sectors)
DMEPA : 0x1 (delete PcROP protection and earse protected area)
DMESA : Øx1 (delete Secure protection and erase protected area)
Boot address Option Bytes:
BOOT_CM7_ADD0: 0x800 <0x8000000>
BOOT_CM7_ADD1: 0x1FF0 <0x1FF00000>
PCROP Protection:
PCROPA_str : 0x800 (0x8010000)
PCROPA_end : 0x806
Secure Protection:
SECA_str : 0xFF (0x8001FE0)
SECA_end : 0x0 <0x80000FF>
DICH RAM Protection:
ST_RAM_SIZE : 0x2 (8 KB)
Write Protection:
nWRPO : Øx1 (Write protection not active on this sector)
nWRP1 : Øx1 (Write protection not active on this sector)
nWRP2 : Øx1 (Write protection not active on this sector)
nWRP3 : Ux1 (Write protection not active on this sector)
nWRP4 : Øx1 (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)



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 the dedicated "printf" packets appears.



# 9 Example of SFIx programming scenario for STM32H7

# 9.1 Scenario overview

There are three steps during this scenario:

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

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

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

# 9.2 Hardware and software environment

For successful SFIx programming, some HW and SW prerequisites apply:

- STM32H7B3I-EVAL board containing external Flash memory.
- Micro-USB for debug connection.
- PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra.
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer V2.3.0 (or greater) package available from www.st.com
- HSMv1.1 card.

# 9.3 Step-by-step execution

## 9.3.1 Build OEM application

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

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

## 9.3.2 Perform the SFIx generation (GUI mode)

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



*Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES The option bytes are described in the product reference manual.

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

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

Figure 67. Successful SFIx generation

File	d Package Creator Edit	Options	Help					F		
6 8								<b>_</b> )	life.	augmer
SFI	SFD	< 55	5P WB 5	IGN	SM	п	ŀ	ISM		
Internal f	irmware files			Firr	nware info	ormation		SFIx info	rmation	
firmwa	ire.hex		Remove	Overvie	2w					
External f	irmware files			File	name	Size	2	mage version	Internal	segme
Externa	al.hex		Add Remove	out sfix		1016 B	01	-	1	
Key area	file					1010 0			<u> </u>	•
C:/SFIx/key	_areas.kcsv		Ch Success		×					<u> </u>
Encryptio	n key file		SFIx suc	cessfully crea	ited	ne	Size		ddress	
C:/SFIx/key	.bin				rm	ware	16 B	0x	8010000	
Nonce file				2	Ke	₂y	40 B		0x0	
C:/SFIx/non	ce.bin		Open	3	Pau	ise	32 B	0x	8010000	
Option by	tes file			4	Resu	ime	32 B	0x	8010000	
C:/SFIx/ob.	CSV		Open	5	Exte	rnal	512 B	0x9	0000000	
Image ve	rsion			6	Pau	ise	32 B	0x	8010020	
1 ÷				7	Resu	ime	32 B	0x	8010020	
RAM size	C	ontinuation token ad	dress 0x08010000	8	Configu	uration	36 B		0x0	-
Output SF	Tx file			Parse S	FIx file					
C:/SFIx/out	sfix		Select folder						Brow	/se
								G	enerate s	FIX



# 9.3.3 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 68: Example of HSM programming using STPC GUI* shows an example for HSM programming by OEM to be used for SFIx install.

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

This example uses HSM version 1. The HSM version can be identified before performing the programming operation by clicking the "Refresh" button to make the version number appear in the Version field.

M32 Trusted	Package Creator							
File	Edit	Options	Help				5	life.augm
6 🛞	<b></b>							
SFI	SFI	Íx :	SSP V	B 510	in	SMI	HSM	
HSM card in	ndex				HSM informatio	on		
1 📩					Firmware ID	test		
	1			-	Max counter	92		
Firmware id	dentifier			_	HSM status	OEM_STATE		
test					Version	1		
Encryption	key file				Туре	-		
C:/HSM/key.b	in		Open				Clear	Refresh
				-				
Nonce file			_					
C:/HSM/nonce	e.bin		Open					
Personaliza	ation data file							
			Op <u>en</u>					
				-				
Maximum o	counter							
92 🛨								
				-				
								Program HSM

Figure 68. Example of HSM programming using STPC GUI



Note: When using HSM version 1, the "Personalization data file" field is ignored when programming starts. It is only used with HSM version 2.

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

# 9.3.4 Performing HSM programming for license generation using STPC (CLI mode)

Refer to Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode).

### 9.3.5 **Programming input conditions**

Before performing an SFIx install make sure that:

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

### 9.3.6 Perform the SFIx 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 SFIx image *"out.sfix"* 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 SFIx install.

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

#### Using JTAG/SWD

STM32\_Programmer\_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static "<local\_path>/out.sfix" hsm=1 slot=<slot\_id> -el <ExternalLoader\_Path>

*Figure 69: SFIx install success using SWD connection (1) through Figure 72: SFIx install success using SWD connection (4)* shows the SFIx install via SWD execution and the HSM as license generation tool in the field.



liguie	
STM32CubeProg	grammer v2.3.0
ST-LINK SN : 004000193037510B353333 ST-LINK FW : V3J1M1 Voltage : 3.28V SWD freq : 24000 KHz Connect mode: Hot Plug Reset mode : Core reset Device ID : 0x480 Device name : STM32H7A/B Flash size : 2 MBytes Device type : MCU Device CPU : Cortex-M7	.31
Protocol Information : sta	atic
SFI file path : '. SFI HSM slot ID : 1 SFI header information : SFI protocol version SFI total number of areas SFI image version SFI Areas information :	: 1 : 7 : 1
Parsing Area 1/7 : Area type Area size Area destination address Parsing Area 2/7 : Area type Area size Area destination address	: F : 16 : 0x8000000 : P : 32 : 0x8001000
Parsing Area 3/7 : Area type Area size Area destination address Parsing Area 4/7 : Area type Area size	: R : 32 : 0x8001000 : E : 528

Figure 69. SFIx install success using SWD connection (1)



Parsing Area 5/7 : Area type : P Area size : 32 Area destination address : 0x8001020								
Parsing Area 6/7:Area type: RArea size: 32Area destination address: 0x8001020								
Parsing Area 7/7 : Area type Area size Area destination address	: C : 36 : 0x0							
Reading the chip Certificate								
Requesting Chip Certificate from device								
Get Certificate done successfully								
requesting license for the current STM32 device								
Init Communication								
ldm_LoadModule(): loading module "stlibp11_SAM.dll" ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62070000 C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x620EF560 P11 lib initialization Success!								
Opening session with solt ID 1								
Succeed to Open session with reader solt ID 1								
Succeed to generate license for the current STM32 device								
Closing session with reader slot ID 1								
Session closed with reader slot ID 1								
Closing communication with HSM								
Communication closed with HSM								
Succeed to get License for Firmware from HSM slot ID 1								
Starting Firmware Install operation								
Erase external flash size : 513 startAddress : 0x90000000 endAddress : 0x90000200 Erasing external memory sector 0								

## Figure 70. SFIx install success using SWD connection (2)



Figure 71. SFIx install success using SWD connection (3)

Warning: Option Byte: SECURITY, value: 0x1, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded Activating security Success Activating security... Setting write mode to SFI Warning: Option Byte: SECURITY, value: 0x1, was not modified. Warning: Option Byte: ST\_RAM\_SIZE, value: 0x3, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded Succeed to set write mode for SFI Starting SFI part 1 Writing license to address 0x24030800 Writing Img header to address 0x24031000 Writing areas and areas wrapper... RSS process started... RSS command execution OK RSS complete Value = 0x0Reconnecting... ST-LINK SN : 004000193037510B35333131 ST-LINK FW : V3J1M1 Voltage : 3.28V SWD freq : 24000 KHz Connect mode: Hot Plug Reset mode : Core reset Device ID : 0x480 Reconnected ! Requesting security state... Warning: Could not verify security state after last chunk programming Starting SFI part 2 Writing license to address 0x24030800 Writing Img header to address 0x24031000 Writing areas and areas wrapper... RSS process started... RSS command execution OK RSS complete Value = 0x0Reconnecting... ST-LINK SN : 004000193037510B35333131 ST-LINK FW : V3J1M1 Voltage : 3.28V SWD freq : 24000 KHz SWD freq Connect mode: Hot Plug Reset mode : Core reset Device ID : 0x480 Reconnected ! Requesting security state... Warning: Could not verify security state after last chunk programming



#### Figure 72. SFIx install success using SWD connection (4)





# **10** Example of SFIx programming scenario for STM32L5

# **10.1** Scenario overview

There are three steps during this scenario:

- Generate SFIx image using the STPC
- HSM card provisioning via STPC
- Use STM32CubePrg to perform the SFIx process.

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

- The internal Flash memory is readable from base addresses 0x08000000 and 0x08040000. It contains the internal firmware.
- The external Flash is programmed so as to be readable with external Flash loader. You can then read the external Flash encrypted by the OTFDEC keys. The pattern of values must be present in the binary files of external firmware.
- If the application works correctly, LED4 blinks.

# **10.2** Hardware and software environment

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

- an STM32L5-based evaluation board containing external Flash memory
- a Micro-USB for debug connection
- a PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra
- an STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- an STM32CubeProgrammer V2.3.0 (or greater) package available from *www.st.com*
- an HSMv1.1 card.

# 10.3 Step-by-step execution

## **10.3.1** Build OEM application

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



## **10.3.2 Perform the SFIx generation (GUI mode)**

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

Note: STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES

The option bytes are described in the product reference manual.

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

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

#### Use case 1 generation of SFIx without key area:

Internal firmware files:

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

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

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

Encryption key: use the same key as HSM.

Nonce file: use the same nonce as HSM.

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

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




Figure 73. Successful SFIx generation use case 1



## Use case 2 generation of SFIx with key area:

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

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

STM32 Trusted	Package Creator						-	
File	Edit	Options	Help				ST life.	augmente
l 🖌 👧	<b></b>							
SFI		SFIx	SSP WE	SIGN	SMI	нѕм		
Internal firr	nware files			Firm	ware information		FTx information	
Project_r	ns.bin		Add					
External fire	mware files		Kelilove	Overvie	N			
fw_ext_fl	ash_1MB.bin		Add	Filen	ame Si	ize Image	version Internal	segme
Key area fil	e			out.sfix	1.09 MB	01	12	
C:/SFIx/key_a	reas.kcsv		Open	•				•
Encryption	key file			Segmen	Time	Size	Address	
C:/SFIx/key.b	in		Open	33	Resume	32 B	0x8060140	
Nonce file				34		2176 B		
C:/SFIx/nonce	.bin		Open	35	Key	40 B	0x0	
Option byte	s file			36	Pause	32 B	0x8060160	
C:/SFIx/ob.cs	v		Open	37	Resume	32 B	0x8060160	
Image vers	ion			38	External	6320 B	0x9000000	
1 🔅				39	Pause	32 B	0x8060180	
RAM size	0x1900	Continuation token	address 0x08060000	40	Resume	32 B	0x8060180	-
Output SFIx	file			Parse St	Tx file			
C:/SFIx/out.st	fix		Select folder				Brow	se
							Generate S	FIX
							Generates	

#### Figure 74. Successful SFIx generation use case 2

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



# 10.3.3 Performing HSM programming for license generation using STPC (GUI mode)

Refer to Section 9.3.3: Performing HSM programming for license generation using STPC (GUI mode).

# 10.3.4 Performing HSM programming for license generation using STPC (CLI mode)

Refer to Section 9.3.4: Performing HSM programming for license generation using STPC (CLI mode).

## **10.3.5 Programming input conditions**

Before performing an SFIx install make sure that:

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

## 10.3.6 Perform the SFIx 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 SFIx image *"out.sfix"* 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 SFIx install.

### Using JTAG/SWD

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

STM32\_Programmer\_CLI.exe -c port=swd mode=HOTPLUG -sfi protocol=static
"<local\_path>/out.sfix" hsm=1 slot=<slot\_id> -rsse <RSSe\_Path> -el
<ExternalLoader\_Path>

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

*Figure 75: SFIx install success using SWD connection (1)* through *Figure 79: SFIx install success using SWD connection (5)* show the SFIx install via SWD execution and the HSM as license generation tool in the field.



C:\Program Files\STMicroelectronics\STM3Cube\STM3Cube\Programmer\bin\STM32_Programmer_CLI.eve -c port-sud mode+HOTPLUG reset-Crst -sfi protocol=static C:\data_store\app_LED_V.sfix hsm-1 slot-1 -rsse "C:\Program Files\STMicroelec STM3Zube\STM3Zube\Programmer\bin\BSSe\LS\enc_signed_RSSe_sfi_jtag.bin" -el ExternalLoader\WZ5LM512456_STM32L552E-EVAL-SFIX.stldr	tronics
STR02CubeProgrammer v2.3.0	
ST-LINK SN : 0678FF535351717867133324 ST-LINK FN : V2J34425 ST-LINK FN : V2J34425 SNOTreq : 4400 KHz Koncet mode : hof Plug Reset mode : Core reset Device ID : 0 k472 Device ID : 0 k472 Device ID : 0 k472 Device ID : 2 k4745 Device ID : 2 k4745 Devic	
Device type : MCU	
Device CPU : Contex-M33	
Protocol Information : static	
SFI File Information :	
SF1       F1Le path       : C:\data_store\upp_LED_V.sfix         SF1       Model of ID       :         SF1       Model of Information       :         SF1       total number of areas       : 39         SF1       SF1       total number of areas       : 0         SF1       Areas information       :       .	
Parsing Area 1/39 : Area type : F Area size : 6536 Area destination address : 6x200000	
Parsing Area 2/39 : Area type : F Area size : 16 Area destination address : 0xc020000	
Persing Area 3/39 : Area type : F Area size : 222 Area detination address : 8xcAd4000	

Figure 75. SFIx install success using SWD connection (1)



Parsing Area 4/39 :	
Area type	• p
Aneo cipe	
Alea Size	
Area destination address	: 0X0
Parsing Area 5/39 :	
Area type	: R
Area size	: 32
Area destination address	- AvA
Area acsemberon address	
D	
Parsing Area 6/59	
Area type	
Area size	: 102336
Area destination address	: 0x9000000
Parsing Area 7/39 :	
Area type	• P
Anea cire	
Area size	· 32
Area destination address	: 0x20
Parsing Area 8/39 :	
Area type	: R
Area size	: 32
Area destination address	: 0x20
Pancing Anap 9/39	
Farsting Area 9/39	
Area type	
Area size	: 102336
Area destination address	: 0x90018fb0
Parsing Area 10/39 :	
Area type	: P
Area size	- 32
Anno doctination address	- 0×10
Alea descinación address	. 0,40
D 1 1 11 (20	
Parsing Area 11/39 :	
Area type	: R
Area size	: 32
Area destination address	: 0x40
Parsing Area 12/39 :	
Area type	: E
Area size	102336
Area size	. 102550 . 0x00021f50
Area descination address	. 071000
D 1 4 42/20	
Parsing Area 13/39 :	
Area type	: P
Area size	: 32
Area destination address	: 0x60
Parsing Area 14/39	
	• P
Area cype	
Area size	: 32
Area destination address	: 0260



Succeed to get License for Firmware from HSM slot ID 1
Starting Firmware Install operation
Set RDP to 0xAA
Reconnecting Reconnected ! Installing RSSe
Memory Programming Opening and parsing file: enc_signed_RSSe_sfi_jtag.bin File : enc_signed_RSSe_sfi_jtag.bin Size : 33200 Bytes Address : 0x20005100
Erasing memory corresponding to segment 0: Download in Progress:
File download complete Time elapsed during download operation: 00:00:00.455 Warning: Option Byte: TZEN, value: 0x1, was not modified. Warning: Option Byte: nBoot0, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded Warning: Option Byte: SECBOOTADDO, value: 0x1FF000, was not modified. Warning: Option Byte: SECBOOTADDO, value: 0x7F, was not modified. Warning: Option Byte: SECMM1_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECMM1_PEND, value: 0x7F, was not modified. Warning: Option Byte: SECMM2_PEND, value: 0x0, was not modified. Warning: Option Byte: SECMM2_PEND, value: 0x0, was not modified. Warning: Option Byte: SECMM2_PEND, value: 0x0, was not modified. Warning: Option Bytes are unchanged, Data won't be downloaded
Reconnecting Reconnected ! Get RSSe status Erase external flash size : 1048577 startAddress : 0x90000000 endAddress : 0x901000000 Erasing external memory sectors [0 16] Starting SFI
Processing license Processing Image Header Processing Area 1 Area Address = 0xc000000 Area Type = F Processing Area 2 Area Address = 0xc020000 Area Type = F Processing Area 3 Area Address = 0xc040000

Figure 77. SFIx install success using SWD connection (3)



	rigure 70. Si ix install success using SWD connection (4)	
Area Type = F		
Processing Area 4		
Area Address = 0x0		
Area Type = P		
Processing Area 5		
Area Address = 0x0		
Area Type = R		
Processing Area 6		
Area Address = 0x90000000		
Area Type = E		
Buffer Address = 0x20005100	0	
E Area Full Size = 102336		
E Area Data Size = 102320		
Processing Area 7		
Area Address = 0x20		
Area Type = P		
Processing Area 8		
Area Address = 0x20		
Area Type = R		
Processing Area 9		
Area Address = 0x90018fb0		
Area Type 🛛 = E		
Buffer Address = 0x20005100		
E Area Full Size = 102336		
E Area Data Size = 102320		
Processing Area 10		
Area Address = 0x40		
Area Type = P		
Processing Area 11		
Area Address = 0x40		
Area Type = R		
Processing Area 12		
Area Address = 0x90031f60		
Area Type = E		
Buffer Address = 0x20005100	8	
E Area Full Size = 102336		
E Area Data Size = 102320		

Figure 78. SFIx install success using SWD connection (4)



Area Address = 0x140
Area Type = R
Processing Area 36
Area Address = 0x900f9ce0
Area Type = E
Suffer Address = 0x20005100
Area Full Size = 25392
Area Data Size = 25376
Processing Area 37
Area Address = 0x160
Area Type = P
Processing Area 38
Area Address = 0x160
Area Type = R
Processing Area 39
Can not verify last area
Area Address = 0x0
Area Type = C
SFI Process Finished!
FI file C:\data_store\app_LED_V.sfix Install Operation Success
Time elapsed during SFI install operation: 00:01:25.116



## 11 Example of combined SFI-SMI programming scenario

## 11.1 Scenario overview

The user application to be installed on the STM32H753xl 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).

## **11.2** Hardware and software environment

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

## 11.3 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 STMicoelectronics or 3<sup>rd</sup> 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 STM32 Trusted Package Creator 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).

*Note:* STM32CubeProgrammer V2.8.0 and later provide one option byte file example for each product.

It is located in the directory: STM32CubeProgrammer\vx.x.x\bin\SFI\_OB\_CSV\_FILES

The option bytes are described in the product reference manual.

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



*Figure 80* shows the STPC GUI during combined SFI generation.

TM32 Trusted	Package Creator								
File	Edit	Options	Help						ugment
🤞 💰									
	SFI		SMI			SFU		нѕм	
Firmware	files		_		Firm	ware informatio	n	SFI information	
FIR_dat	a.hex		Add Remove		Overvie	w			
Encryption	1 key file				File nam	e out_cor	nb.sfi		-
ackage_Crea	tor_v1.0.2 (1)/Input/5	FI/good/test_firmware_	key.bin Open		Size	13.9336	KB		
Nonce file	ackage Creator v1.0	2 (1)/Input/SEI/good/po	SFI suc	cessfu	ully create	201 ed			•
	indige_oreator_ritor					Туре	Size	Address	
Option byt	es file		•	К		rmware	12280 B	0x8000000	
Package_Cre	ator_v1.0.2 (1)/Input/	/SFI_SMI_combined/FIR_		-	2	Firmware	116 B	0x8084000	
SMI files (	Only for combined o	case)			3	Module	1688 B	0x8080000	
FIR_pcr	op.smi		Add		4	Configuration	36 B	0x0	
			Reinore						
Image ver	sion								
RAM size		Continuation token a	ddress						
Output SF	I file								
STM32_Trust	ed_Package_Creator_	v1.0.2 (1)/output/out_c	omb.sfi Select folder						
								General	te SFI

### Figure 80. GUI of STPC during combined SFI-SMI generation

- 3. Programming input conditions are the same as for the SFI programming scenario (Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)).
- 4. Perform the SFI install using the SWD/JTAG or a bootloader interface (here the SWD interface is used).



## 11.3.1 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 81: Combined SFI-SMI programming success using debug connection* shows the combined SFI-SMI install trace success.



Figure 81. Combined SFI-SMI programming success using debug connection

```
ST-LINK Firmware version : V2J26M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450
                                     : static
: RefSFI_MDK/SFI_Combined/out_comb.sfi
    Protocol
SFI File
Starting SFI install operation for file : RefSFI_MDK/SFI_Combined/out_comb.sfi
   SFI File Information
          SFI file path
SFI license file path
                                                              RefSFI_MDK/SFI_Combined/out_comb.sfi
RefSFI_MDK/SFI_Combined/licenseSFIcomb_h753bEH.
                                                           -
bin
         SFI header information :
SFI protocol version
SFI total number of areas
SFI image version
SFI Areas information :
                                                                         1
4
23
                                                                      Parsing Area 1/4
                 Area type
Area size
Area destination address
                                                                      н
                                                                         F
                                                                         r
12280
0<u>x</u>8000000
         Parsing Area 2/4 :
Area type
Area size
Area destination address
                                                                         F
                                                                      r
116
0x8084000
         Parsing Area 3/4 :
Area type
Area size
Area destination address
                                                                      : M
: 1688
: 0×8080000
         Parsing Area 4/4 :
Area type
Area size
Area destination address
                                                                         С
36
ØxØ
                                                                      Succeed to set write mode for SFI
Writing license to address 0x24007800
Writing Img header to address 0x24008000
Writing areas and areas wrapper...
RSS process started...
Setting write mode to SFI
  228
       command execution OK
Reconnecting...
ST-LINK Firmware version : V2J26M15
SWD frequency = 4000 KHz
Connection mode: Hot Plug
Device ID: 0x450
 Reconnected !
Requesting security state...
SECURITY_State_Success
        SUCCESS*
        file RefSFI_MDK/SFI_Combined/out_comb.sfi Install Operation Success
  SF I
Time elapsed during the SFI install operation is: 00:00:04.056
Press <RETURN> to close this window...
```



## 11.3.2 How to test the combined SFI install success

The option bytes configuration must be modified as shown in *Figure 82: Option bytes after combined SFI-SMI install success*.

- 3<sup>rd</sup> party library module is programed 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 the dedicated "printf" packets appears.



OPTION BYTES BANK: Ø
Read Out Protection:
RDP : 0x0 (Level 1, read protection of memories)
RSS:
RSS1 : ØxØ (No SFI process on going)
BOR Level:
BOR_LEV : 0x2 (reset level is set to 2.7 V)
User Configuartion:
IWDG1 : 0x1 (Independent watchdog is controlled by hardware)
IWDG2 : Øx1 (Window watchdog is controlled by hardware)
NRST_STOP_D1 : 0x1 (STOP mode on Domain 1 is entering without reset)
NRST_STBY_D1 : 0x1 (STANDBY mode on Domain 1 is entering without reset)
FZ_IWDG_STOP : 0x1 (Independent watchdog is running in STOP mode)
FZ_IWDG_SDBY : 0x1 (Independent watchdog is running in STANDBY mode)
SECURITY : Øx1 (Security feature enabled)
BCM7 : Øx1 (CM-7 boot enabled)
NRST_STOP_D2 : 0x1 (STOP mode on Domain 2 is entering without reset)
NRST_STBY_D2 : 0x1 (STANDBY mode on Domain 2 is entering without reset)
SWAP_BANK : 0x0 (after boot loading, no swap for user sectors)
DMEPA : Øx1 (delete PcROP protection and earse protected area)
DMESA : Øx1 (delete Secure protection and erase protected area)
Boot address Option Bytes:
BOOT_CM7_ADD0: 0x800 <0x8000000)
BOOT_CM7_ADD1: 0x1FF1 (0x1FF10000)
PCROP Protection:
PCROPA_str : 0x800 (0x8010000)
PCROPA_end : 0x806 (0x8080600)
Secure Protection:
SECA_str : 0xFFF (0x801FFE0)
SECA_end : 0x0 (0x80000FF)
DTCM RAM Protection:
ST_RAM_SIZE : 0x3 (16 KB)
Write Protection:
nWRPO : 0x1 (Write protection not active on this sector)
nWRP1 : Øx1 (Write protection not active on this sector)
nWRP2 : Øx1 (Write protection not active on this sector)
nWRP3 : Øx1 (Write protection not active on this sector)
nWRP4 : Øx1 (Write protection not active on this sector)
nWRP5 : Øx1 (Write protection not active on this sector)
nWRP6 : Øx1 (Write protection not active on this sector)
nWRP7 : Øx1 (Write protection not active on this sector)

Figure 82.	Option b	ytes after	combined	SFI-SMI	install success
------------	----------	------------	----------	---------	-----------------



## 12 Example of SSP programming scenario for STM32MP1

## 12.1 Scenario overview

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

Three main steps are done using SSP tools:

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

## **12.2** Hardware and software environment

The following prerequisites are needed for successful SSP programming:

- an STM32MP1-DK2 board
- a Micro-USB for DFU connection
- a PC running on either Windows 7/10 or Ubuntu 14 64-bit or macOS High Sierra
- STM32TrustPackageCreator v1.2.0 (or greater) package available from www.st.com
- STM32CubeProgrammer V2.5.0 (or greater) package available from www.st.com
- an HSMv2 card.

## 12.3 Step-by-step execution

## 12.3.1 Building a secret file

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

On an STM32MP1 microprocessor:

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

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



## 12.3.2 Performing the SSP generation (GUI mode)

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

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

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

STM32 Trus	ted Package Cre	ator					- 0
File	Edit	Options	Help				life.ou
1 🖌 🎼	۵ 🚐 🗠						10
SFI		SFIx	SSP	WB SIGN	SMI	нѕм	
Password	d : RMA Lock	0x312	RMA Relock	0xECA	Secrets file in	formation	SSP information
Secrets f	ile				Overview	_	
C:/SSP_In	put/secrets/148byt	es_secrets.bin		Open	File name	Туре	Size
Encryptic	on key file				148bytes_secre		148 B
C:/SSP_In	put/aes_key/key.bi	n		Open			
OEM publ	lic key file						
C:/SSP_In	put/OEMPublicKey.	pem		Open			
Nonce fik	e						
C:/SSP_In	put/aes_key/iv.bin			Open			
Output S	SP file						
C:/ssp/out	ssp			Select folder			
							Generate SS

#### Figure 83. STM32 Trusted Package Creator SSP GUI tab



# 12.3.3 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 SSP 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 details.

This example uses HSM version 2. The HSM version can be identified before performing the programming operation by clicking the **Refresh** button to make the version number appear in the Version field.

Note: HSM version 2 must be used for STM32 MPU devices.

Figure 84. Example	of HSMv2 p	rogramming u	ising STPC GUI	
🚱 STM32 Trusted Package Creator			-	-
File Edit Options	Help		5	life.augmented
🔜 🖌 🛞 📠 😂				
SFI SFIx SSP	WB SI	GN SMI	НЅМ	
HSM card index		HSM information		
1 ÷		Max counter	13	<b>_</b>
Firmware identifier		HSM status	OPERATIONAL_STATE	
SSP_MPU		Version	2	
Encryption key file		Туре	SSP	•
C:/ssp/key.bin	Open		Clear	Refresh
Nonce file				
C:/ssp/nonce.bin	Open			
Personalization data file				
ckages/5000300A_SSP01000000_0000000.enc.bin	Open			
Maximum counter				
0				
				Program HSM



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

## 12.3.4 SSP programming conditions

Before performing an SSP flow make sure that:

- a DFU or UART interface is used
- the chip supports security
- the SSP image is encrypted by the same key/nonce as used in the HSM provisioning step.

## 12.3.5 Perform the SSP install using STM32CubeProgrammer

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

Example using USB DFU bootloader interface:

STM32\_Programmer\_CLI.exe -c port=usb1 -ssp "out.ssp" "tf-a-sspstm32mp157f-dk2-trusted.stm32" hsm=1 slot=1



All SSP traces are shown on the output console (Figure 85).

```
Figure 85. STM32MP1 SSP install success
```

```
Requesting Chip Certificate...
 Get Certificate done successfully
requesting license for the current STM32 device
Init Communication ...
ldm_LoadModule(): loading module "stlibp11_SAM.dll" ...
ldm_LoadModule(WIN32): OK loading library "stlibp11_SAM.dll": 0x62000000 ...
C_GetFunctionList() returned 0x00000000, g_pFunctionList=0x62062FD8
 P11 lib initialization Success!
Opening session with solt ID 1...
 Succeed to Open session with reader solt ID 1
 Succeed to generate license for the current STM32 device
Closing session with reader slot ID 1...
 Session closed with reader slot ID 1
Closing communication with HSM...
 Communication closed with HSM
Succeed to get License for Firmware from HSM slot ID 1
Starting Firmware Install operation...
Writing blob
 Blob successfully written
 Start operation achieved successfully
 Send detach command
 Detach command executed
 SP file out.ssp Install Operation Success
```



## **13** Reference documents

Т

Table 3. Document references	
Document title	

Reference	Document title
[1]	AN4992, STM32H7 secure firmware/module install overview. STMicroelectronics.
[2]	UM2428, Hardware secure modules (HSM) for secure firmware install (SFI). STMicroelectronics.
[3]	AN5510, Overview of secure secret provisioning (SSP)



## 14 Revision history

Date	Revision	Changes
03-Aug-2018	1	Initial release.
18-Apr-2019	2	Updated publication scope from 'ST restricted' to 'Public'.
16-Oct-2019	3	<ul> <li>Updated:</li> <li>Section 4.1.2: License mechanism</li> <li>Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)</li> <li>Figure 43: HSM programming GUI in the STPC tool (title caption)</li> <li>Figure 52: Example of HSM programming using STPC GUI</li> </ul>
03-Feb-2020	4	Replaced occurrences of STM32L451CE with STM32L462CE in Section 4.2.1: Secure firmware installation using a bootloader interface flow. Updated document to cover secure programming with SFIx.
26-Feb-2020	5	<ul> <li>Updated:</li> <li>Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow</li> <li>Section 5.3.3: Performing HSM programming for license generation using STPC (GUI mode)</li> <li>Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode)</li> <li>Figure 69: SFIx install success using SWD connection (1)</li> <li>Figure 72: SFIx install success using SWD connection (4).</li> </ul>
27-Jul-2020	6	Updated: - Introduction - Section 3.1: System requirements Added: - Section 3.5: SSP generation process - Section 3.6.3: Steps for SSP generation (CLI) - Section 3.7.4: SSP generation using STPC in GUI mode - Section 4.2.5: STM32CubeProgrammer for SSP via a bootloader interface - Section 12: Example of SSP programming scenario for STM32MP1.

 Table 4. Document revision history



## **Revision history**

Date	Revision	Changes
19-Nov-2020	7	<ul> <li>Updated:</li> <li>Introduction on cover page</li> <li>License mechanism general scheme</li> <li>HSM programming by OEM for license distribution</li> <li>Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode).</li> <li>Added:</li> <li>Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB)</li> <li>Section 6: Example of SFI programming scenario for STM32WL.</li> </ul>
29-Jun-2021	8	<ul> <li>Updated:</li> <li>In the whole document, replaced STM32H7A/B by STM32H7A3/7B3 and STM32H7B0, STM32H7Z/3 by STM32H723/333 and STM32H7E0, STM32H7B board by STM32H7B3I-EVAL</li> <li>Replaced BL by bootloader.</li> <li>Section 3.2: SFI generation process: removed references to RSS.</li> <li>Section 4.1.2: License mechanism: removed Figure HSM programming toolchain.</li> <li>Section 4.2: Secure programming using a bootloader interface, Section 4.2.2: Secure Module installation using a bootloader interface flow, Section 4.2.3: STM32CubeProgrammer for SFI using a bootloader interface</li> <li>Section 4.3.1: SFI/SFIx programming using JTAG/SWD flow and Section 4.3.2: SMI programming through JTAG/SWD flow.</li> <li>Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB)</li> <li>Example of SFI programming scenario/Section 5.2: Hardware and software environment and Example of SFI programming scenario for STM32WLISection 6.2: Hardware and software environment: removed bootloader and RSS versions</li> <li>Section 5.3.4: Performing HSM programming for license generation using STPC (CLI mode): removed STM32L4 from the list of devices that support SFI via debug interface.</li> <li>Added:</li> <li>Support for STM32U5 Series.</li> <li>Section 7: Example of SFI programming scenario for STM32U5.</li> </ul>

Table 4. Document revision history



Date	Revision	Changes
	9	Added note about CSV file in <i>Section 3.6.1: Steps for SFI generation (CLI)</i> and <i>Figure 26: Option bytes file example.</i>
		Corrected binary file names in Section 4.4: Secure programming using Bootloader interface (UART/I2C/SPI/USB).
		Section 3.6.1: Steps for SFI generation (CLI)
		Added note about option byte file example in:
		<ul> <li>Section 3.7.1: SFI generation using STPC in GUI mode</li> </ul>
		– Section 5.3.2: Perform the SFI generation (GUI mode)
02-Aug-2021		– Section 6.3.2: Perform the SFI generation (GUI mode)
		– Section 7.3.2: Perform the SFI generation (GUI mode)
		<ul> <li>Section 9.3.2: Perform the SFIx generation (GUI mode)</li> </ul>
		<ul> <li>Section 10.3.2: Perform the SFIx generation (GUI mode)</li> </ul>
		- Section 11.3: Step-by-step execution.
		Updated Corrected board name in <i>Section 4.2:</i> <i>Secure programming using a bootloader interface.</i>
		Corrected board name in <i>Section 7.2: Hardware and software environment</i> .

Table 4. Document revision history



#### 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 acknowledgment.

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.

© 2021 STMicroelectronics – All rights reserved

AN5054 Rev 9

