Application note

Overview of the secure secret provisioning (SSP) on STM32MP1 Series

Introduction

The outsourcing of product manufacturing enables the original equipment manufacturers (OEMs) to reduce their direct costs and concentrate on high added-value activities such as research and development, sales and marketing.

However, the contract manufacturing puts the OEM's proprietary assets at risk, and since the contract manufacturer (CM) manipulates the OEM's intellectual property (IP), it might be disclosed to other customers, or appropriated.

To meet the new market security requests and protect customers against any leakage of their IPs, STMicroelectronics introduces a security feature, the secure secret provisioning (SSP), allowing the programming of OEM secrets into STM32MP1 one time programming (OTP) area in a secure way (with confidentiality, authentication and integrity checks).

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

This application note gives an overview of the STM32MP1 Series SSP solution with its associated tools ecosystem and explains how to use it to protect OEM secrets during the CM product manufacturing stage.

1 Reference documents

Table 1. Reference documents

Reference number	Document title
[1]	AN4992, Overview secure firmware install (SFI).
[2]	"ROM code overview", available: https://wiki.st.com/stm32mpu/wiki/STM32MP15_ROM_code_overview.
[3]	"STM32MP1 secure boot", available: https://wiki.st.com/stm32mpu/wiki/STM32MP15_secure_boot.
[4]	"Arm Trusted Firmware (STMicroelectronics GITHUB)", available on https://github.com/
[5]	"STM32MP15 TF-A", available: https://wiki.st.com/stm32mpu/wiki/STM32MP15_TF-A.
[6]	AN5275, USB DFU/USART protocols used in STM32MP1 Series bootloaders.

2 General information

This document applies to STM32MP1 Series Arm®-based devices.

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

The following table defines the acronyms needed for a better understanding of this document.

Table 2. List of acronyms

Acronym	Description
AES	Advanced encryption standard (symmetric cryptographic method)
CM	Contract manufacturer
ECC	Elliptic curve cryptography
ECDSA	Elliptic curve digital signature algorithm
ECIES	Elliptic curve integrated encryption scheme
HSM	Hardware security module
OEM	Original equipment manufacturer
OTP	One time programming
SHA-256	Secure hash algorithm on 256 bits. SHA-256 is one variant of SHA-2 family
SSP	Secure secret provisioning
TF-A	Trusted firmware-A, with A meaning Arm® Cortex®-A

3 STM32MP1 secure secret provisioning

3.1 SSP principle overview

The SSP is a secure mechanism implemented in STM32 microprocessors that allows a secure and counted installation of OEM secrets in untrusted production environment (such the OEM contract manufacturer).

The STM32MP1 Series device may contain several secure data in OTP fuses:

- OEM public key hash: the root of trust for the secure boot authentication.
- RMA passwords: used to change the chip life state for hardware investigation purpose.
- Other secrets that may be used by secure software.

The SSP is derived from the STM32 microcontrollers secure firmware install [1] , so it reuses the same kind of authentication and encryption mechanism.

- Hardware security module (HSM).
- Secure chip communication.

The SSP is split in two parts:

- One part in the ROM code [2] accessing the chip private information.
- Another part in the SSP secure firmware exchanging with the external tool and programming the OTP.

The SSP process prevents the OEM secrets from:

- Being accessed by the contract manufacturer.
- Being extracted or disclosed.

This mechanism consists in having the whole OEM secrets encrypted with an AES secret key, thanks to the STM32 Trusted Package Creator tool.

A hardware security module (HSM) is responsible for:

- Securely storing OEM AES secret key and nonce.
- Verifying the STM32MP1 Series device certificate that is used to authenticate the STM32MP1 Series device.
- Generating and providing the license to the chip to securely install the encrypted secrets on the STM32MP1 Series device.
- Counting number of produced STM32MP1 Series devices.

The OEM must use the STM32 Trusted Package Creator tool to program the HSM with its own AES secret key, its own nonce, and a maximum installation counter.

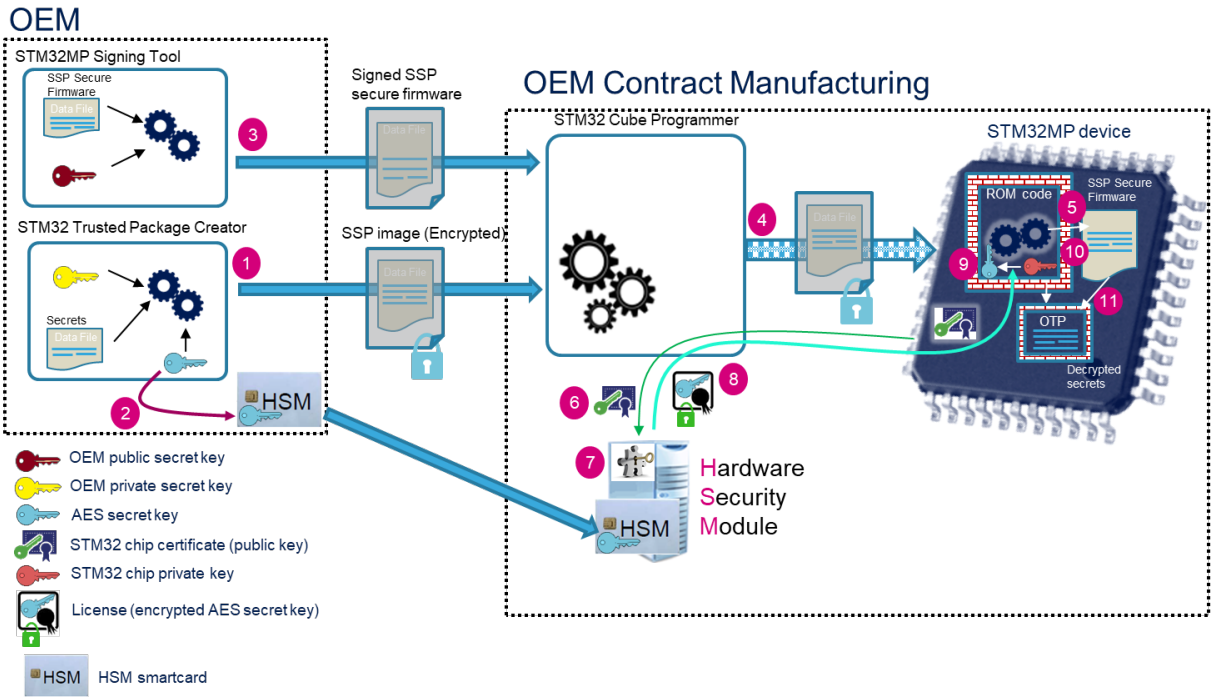
The OEM contract manufacturer must use STM32CubeProgrammer to initiate the SSP process and send an encrypted SSP image to the STM32MP1 Series device.

The STM32 microprocessors are provisioned by STMicroelectronics with the device dedicated ECC private key (unique key per device). This device private key can be only accessed through the ROM code. The AES secret key is decrypted from license using the device private key (ECIES).

Thanks to STM32MP1 security features and cryptographic algorithm, the STM32 microprocessors support secure OEM secrets programming in one time programming area (OTP) to ensure the OEM firmware protection (confidentiality, authenticity and integrity) during the STM32MP1 secure boot [2] process. The ROM code securely decrypts secrets and authenticates the secure firmware which programs the OTP. The OTP area provides secrets obfuscation thanks to robust hardware mechanisms.

3.2 SSP process flow

Figure 1. **SSP process flow**



1. SSP image (encrypted) available from the STM32 Trusted Package Creator tool.
2. Programs the HSM with AES secret key.
3. Signed SSP secure firmware using the STM32MP1 signing tool.
4. SSP process launch.
5. ROM code loads the SSP secure firmware.
6. Device certificate.
7. STM32MP1 Series device authentication.
8. Provides the license concatenate with the SSP image (encrypted).
9. Retrieves the AES decryption key and decrypt secrets.
10. Authenticates the secure firmware.
11. Programs the OTP from decrypted secrets.

3.3 SSP process steps

The SSP processing is split in three different phases:

- SSP initialization
- SSP chip authentication
- SSP secret decryption and OTP programming

These steps are managed between the secure firmware and ROM code, exchanging information using secure SYSRAM area and resetting the chip.

The SYSRAM must be supplied during all the transitions to keep the operations integrity.

A maximum number of SSP attempts is authorized on chip to avoid chip hack (STM32MP1 limited to 4 SSP attempts).

Prior to start the SSP processing, the secrets must have been generated (see Section 4 Security concern) and the SSP firmware must have been configured, built and signed (see Section 5 Secret preparation).

3.3.1 SSP initialization

This step is the entry point to start the SSP process. It can be started directly after the binaries programming or can be directly started using the SSP secure firmware.

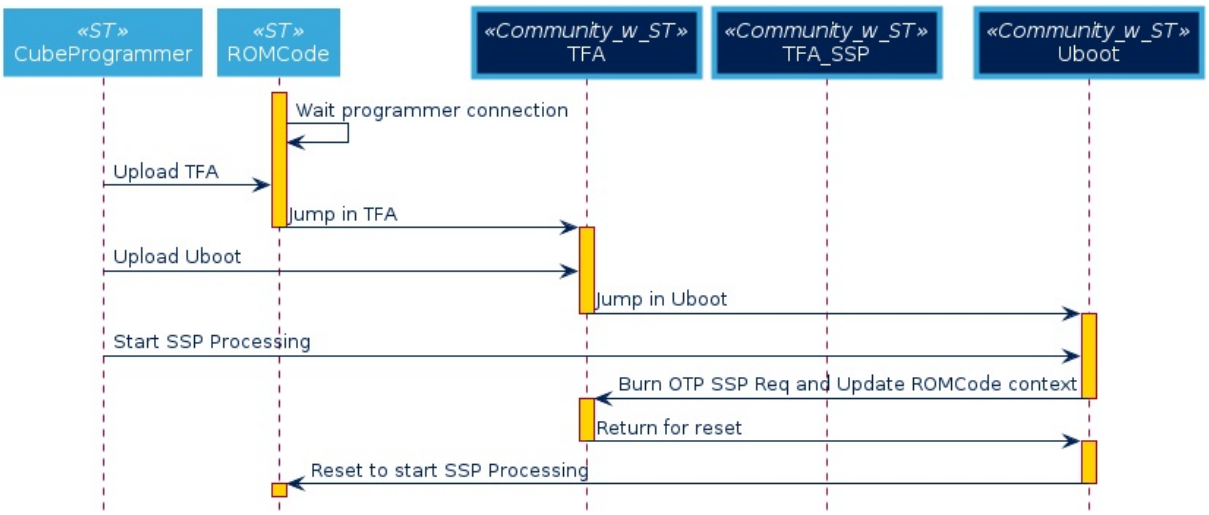
It consists in programming a specific SSP OTP request value and sending to the ROM code the start command.

3.3.1.1 After binaries programming

The current ecosystem release uses a programming flow based on the trusted firmware A and U-Boot. The SSP sequence can be directly launched at the end of the programming process by writing the SSP OTP request bit.

By writing this OTP bit, the secure monitor writes the OTP bit value and fills the requested SSP information in the ROM code context. A reset is required to allow the ROM code to start the next sequence.

Figure 2. SSP start request



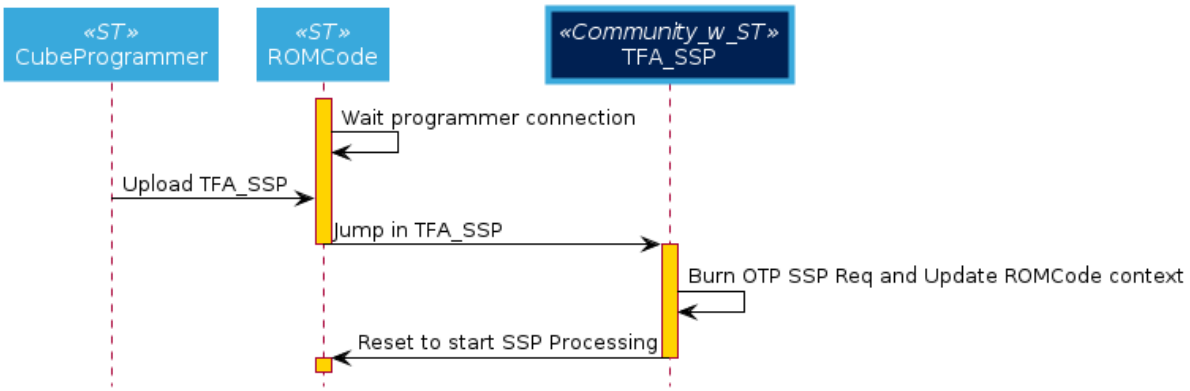
Light blue: ST developement Dark blue: community development with ST add-ons.

3.3.1.2 Standalone SSP start sequence

The SSP secure firmware can be used to start the sequence. It is loaded by the ROM code and, if no SSP sequence already running, it programs the SSP OTP request bit and fills the ROM code context. It also requires a reset to start the next sequence.

This use case allows the SSP secure firmware to be used only during all the SSP processing.

Figure 3. SSP start request from SSP secure firmware



Light blue: ST developement Dark blue: community development with ST add-ons.

3.3.2 SSP chip certificate and encrypted secret file upload

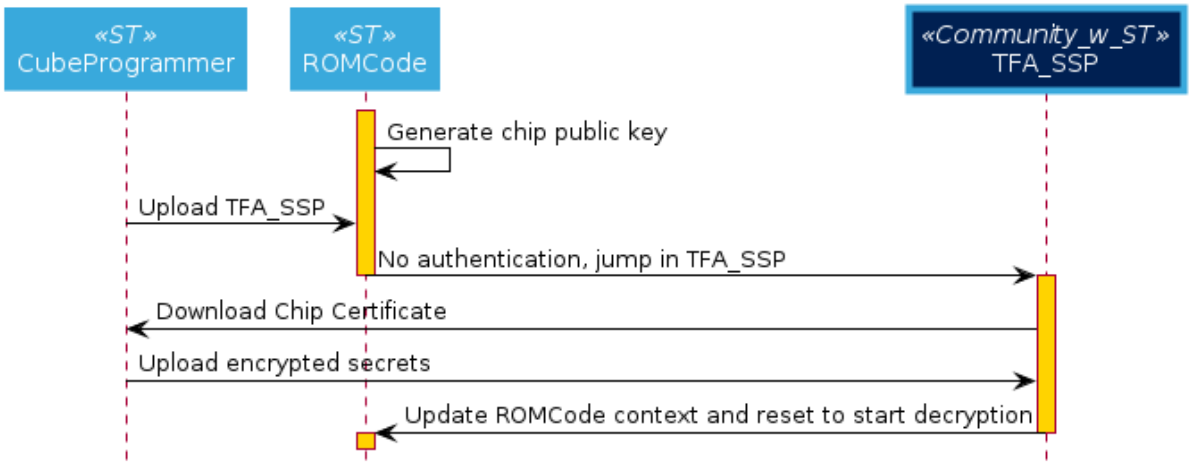
This new step is started after a reset. The ROM code generates a chip ECC public key corresponding to the chip private key in OTP.

The SSP secure firmware generates the chip certificate with the ECC public key generated, the chip certificate from OTP and shares it to the STM32CubeProgrammer tool.

If the chip certificate is validated by the HSM (connected to the STM32CubeProgrammer), it uploads the license file and the encrypted secret file to the SSP secure firmware. Both are stored in the secure SYSRAM for the last SSP step.

A reset is still required to start the final step.

Figure 4. Chip certificate and secret upload



Light blue: ST developement Dark blue: community development with ST add-ons.

3.3.3 Secret decryption and OTP programming

This is the final step of the SSP. This entire sequence is made in a fully secure environment. All external accesses are closed during this step including loaded interface and debugger.

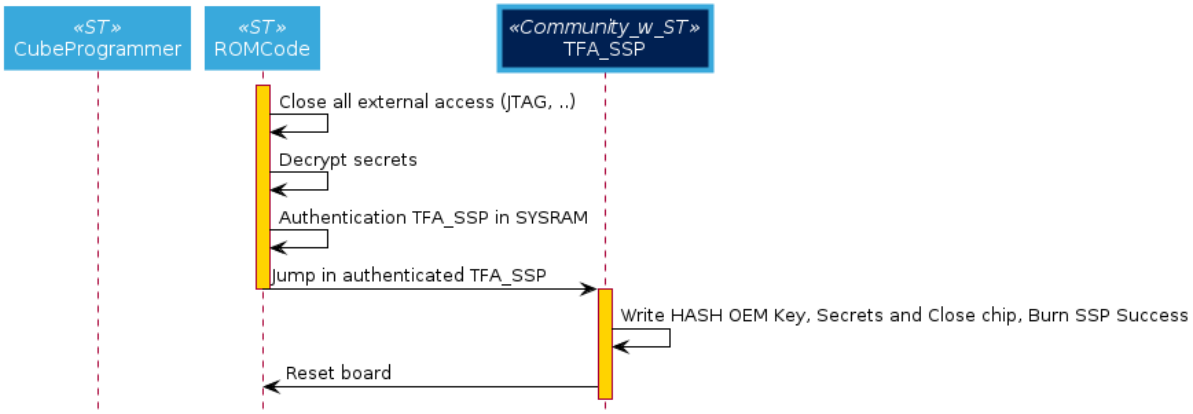
Once resetting, the ROM code:

- Controls the license (stored in SYSRAM during the previous step) and decrypts the AES secret key.
- Decrypts the secrets (stored in SYSRAM during the previous step) using the AES secret key and checks the encryption tag thanks to the AES-GCM algorithm.
- If the check is correct, the ROM code is able to authenticate the SSP secure firmware still resident in SYSRAM secure with the OEM public key from the encrypted secret file.
- If authentication is confirmed, the ROM code gives the decrypted secrets to the SSP secure firmware and jumps in it.

The SSP secure firmware:

- Checks the OTP are free for programming,
- Programs the OEM public key hash,
- Programs the OTP secure close chip to force firmware authentication,
- Programs the secrets in the OTP,
- Programs the SSP success bit to complete the OTP processing,
- Cleans all secrets in the secure SYSRAM memory,
- Resets the target.

Figure 5. Secrets decryption and programming



Light blue: ST developement Dark blue: community development with ST add-ons.

Finalizing this sequence, the chip is fully secure, the secure boot authentication is mandatory based on the OEM public key used, the debugger is default closed and secrets are available to the secure domain.

3.3.4 Error cases

In case of reset or error during the two first steps, the SSP processing can be relaunched without any attempts control.

In case of error during the last step, the SSP firmware authentication error, a corrupted license, a corrupted secrets file, the attempt counter is incremented till it reaches the maximum tries. The STM32MP1 allows four tries, once achieved, the SSP processing could not be fully ended.

4 Security concern

Once ST keys (ECC private key, certificate) are provisioned to each chip, the OEM can rely on secure secrets provisioning and let the OEM-CM populating the chips with its secrets.

The secrets prepared by the OEM are three different types:

- RMA (unlock/relock) passwords
- OEM public key
- OTP secrets

Before the SSP execution, the chip is in a secure open state (no secure boot). The SSP is in charge of switching automatically the chip to the secure close state that forces the secure boot authentication.

4.1 Return material authorization (RMA) passwords

In case of device failure, ST Microelectronics must provide a way to switch the chip into secure open mode to be able to run tests without seeing any secret.

This mechanism to reopen the chip (RMA unlock) and definitively lock it again (RMA relock) is managed by the ROM code. the RMA passwords (unlock/relock) are required to complete each RMA process. Those passwords are set by the OEM, as part of his secrets.

A RMA unlock, decided by OEM puts back the chip in an open state, blocking access to the secrets. It can be safely transmitted to ST Microelectronics for analysis.

The RMA relock returns the chip in a secure state, unblocking the access to the secrets.

The OTP RMA is read protected by the ROM code and cannot be accessed later.

4.2 OEM public key

The OEM public key 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 key hash is stored into the chip to be used as a reference key during the secure boot authentication process. The ROM code must check the key against the hash that have been safely programmed in OTP.

4.3 OTP secrets

Other secrets are chosen by OEM and can be used in secure environment.

It can be passwords, keys, all kind of sensitive information that must be only managed by the secure software and protected in a hardware secure area.

5 Secret preparation

A secret file must be created prior to SSP processing. This secret file must fit into the OTP area reserved for customer.

The OTP memory is organized as 32-bit words.

On the STM32MP1 microprocessor:

- One OTP word is reserved for RMA password (unlock/relock): OTP 56.
- 37 free words are reserved for customer usage, 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 secret file.

It follows these rules:

- 4-byte RMA passwords at the beginning of the file.
- 148 following bytes are containing the OTP secrets.
- 4 bytes set to 0 and aligned on 32-bit word are ignored during the OTP programming.

5.1 RMA passwords

The RMA passwords are chosen by OEM. there are part of the secret file and placed as the first 4-byte word.

On the STM32MP1, the RMA passwords are provided using 15 bits (maximum value is 0x7FFF).

The two passwords are contained in the same 32-bit word:

Table 3. RAM passwords

Bit [31-30]	Bit [29-15]	Bit [0-14]
Not used	RMA relock	RMA unlock

Examples:

RMA unlock: 0x3210

RMA relock: 0x7654

Final OTP: 0x3B2A3210 (8-bit coding style: 10 32 2A 3B)

To ease the RMA passwords creation and avoid a formatting issue, the STM32 Trusted Package Creator tool adds them automatically at the beginning of the secret file.

5.2 OTP secret file

The OTP secret file must be a 148-byte binary file format. It can be created from key files, certificates, all data required by OEM to run its secure software.

The file represents the OTP area referenced as “free for user”. The OTP fuses are accessible through 32-bit BSEC registers, so the padding (free words) is allowed by adding 0 in 32-bit word aligned.

Table 4. OTP area examples

OTP [59:66]	OTP 67	OTP 68	OTP [69:95]
256-bit key	4-bit padding (0)	16-bit password1 / 16-bit password2	….

5.3 Secret file example

Here is a file dump of a 148-byte secret file used as input of the STM32 Trusted Package Creator tool:

00000000 d7 10 de ce 7b 53 70 2d e8 c6 38 be ab fa fd cc |....{Sp-..8.....|
00000010 b6 89 a4 cd 0b c6 65 03 ee 0c 52 ec ab 38 4e 01 |......e...R..8N.|
00000020 5e 03 43 f4 28 58 59 7c **00 00 00 00** a0 99 80 02 |^.C.(XY|........|
00000030 c9 92 38 ed 80 ed 34 8e f6 80 ea a2 5b d0 40 5a |..8...4.....[.@Z|
00000040 f5 2e f0 65 ed 84 9f 12 eb 66 45 5b 33 e4 79 05 |...e.....fE[3.y.|
00000050 18 9b 78 54 c0 0f fc fd d4 53 25 46 57 ec 44 74 |..xT.....S%FW.Dt|
00000060 ad 83 9a a8 45 7d d1 88 8c ea c8 5e 40 b5 d0 5a |....E}.....^@..Z|
00000070 71 d7 ce 42 16 2e b4 b7 38 79 dc f1 47 87 a4 f7 |q..B....8y..G...|
00000080 ff d1 ac 7b 03 09 a5 39 46 7c 36 0c 39 a0 e3 dc |...{...9F|6.9...|
00000090 82 ba a6 57 |...W|

In bold, the padding is added and it keeps the eleventh OTP secret word empty.

5.4 Encrypted secret file

The encrypted secret file follows a specific layout that guarantees a secure transaction during transport and decryption sides.

Table 5. Encrypted secret file layout

-	Size	Content
SSP magic	4 bytes	'SSPP': magic identifier for SSP payload
SSP protocol version	4 bytes	Can be used to indicate how to parse the payload, if payload format changes in future. Current supported protocol version is 1
OEM ECDSA public key	64 bytes	OEM ECDSA public key
OEM secret size	4 bytes	Size of OEM secrets, in bytes
Payload tag	16 bytes	Cryptographic "signature" of all fields above, to ensure their integrity.
Encrypted OEM secrets	152 bytes	Encrypted OEM secrets.

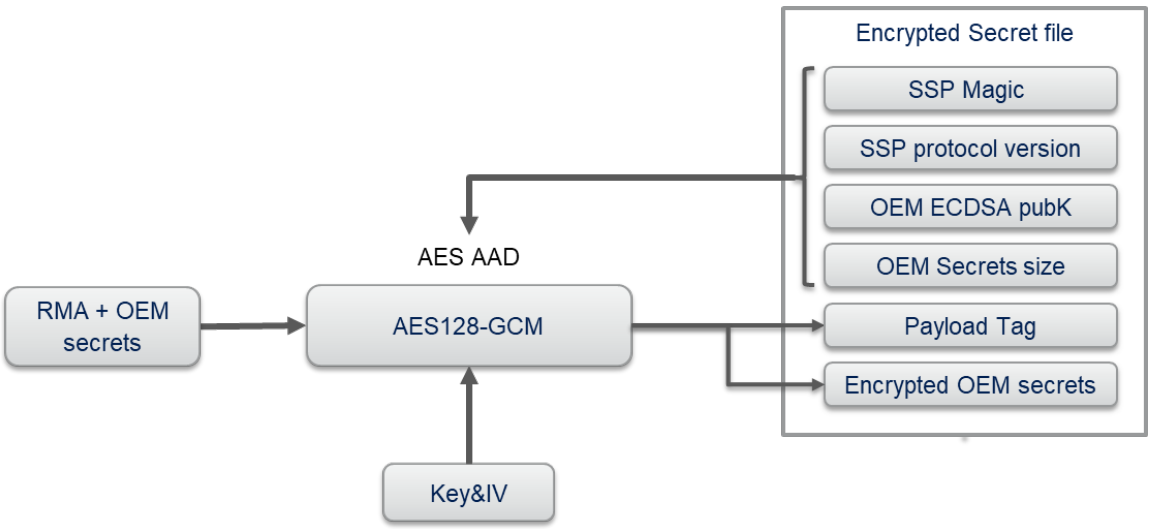
Only the OEM secrets part is AES-GCM encrypted using:

- AES encryption key (128 bits)
- AES nonce (128 bits)

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

This encrypted file is automatically generated by STM32 Trusted Package Creator tool as shown in the following figure.

Figure 6. Encryption file scheme



6 SSP secure firmware

The SSP secure firmware is derived from the standard secure firmware (trusted firmware A) used as the first stage bootloader from the STM32MP1 ecosystem release. It includes additional feature for the SSP processing.

The major change is the SSP library that manages the different SSP phases. Other part of the patches is linked to size optimization to reduce the binary only to the required functions.

A patch is available on top of each official release on a dedicated branch for the trusted firmware A component [3] .

6.1 Build

The build instructions are described in the TF-A wiki page [4] .

6.2 Signed binary

To keep the SSP processing secure, the SSP secure firmware must be signed with the OEM ECDSA key to be authenticated during the last processing phase.

Without this authentication, the SSP processing fails.

7 SSP tool management

On each SSP install step, the user must use the STM32 ecosystem tools to manage the secure programming and the SSP flow.

Overall, there are three main steps to do using SSP tools:

- The encrypted secrets file generation
- The HSM provisioning
- The SSP procedure with STM32CubeProgrammer

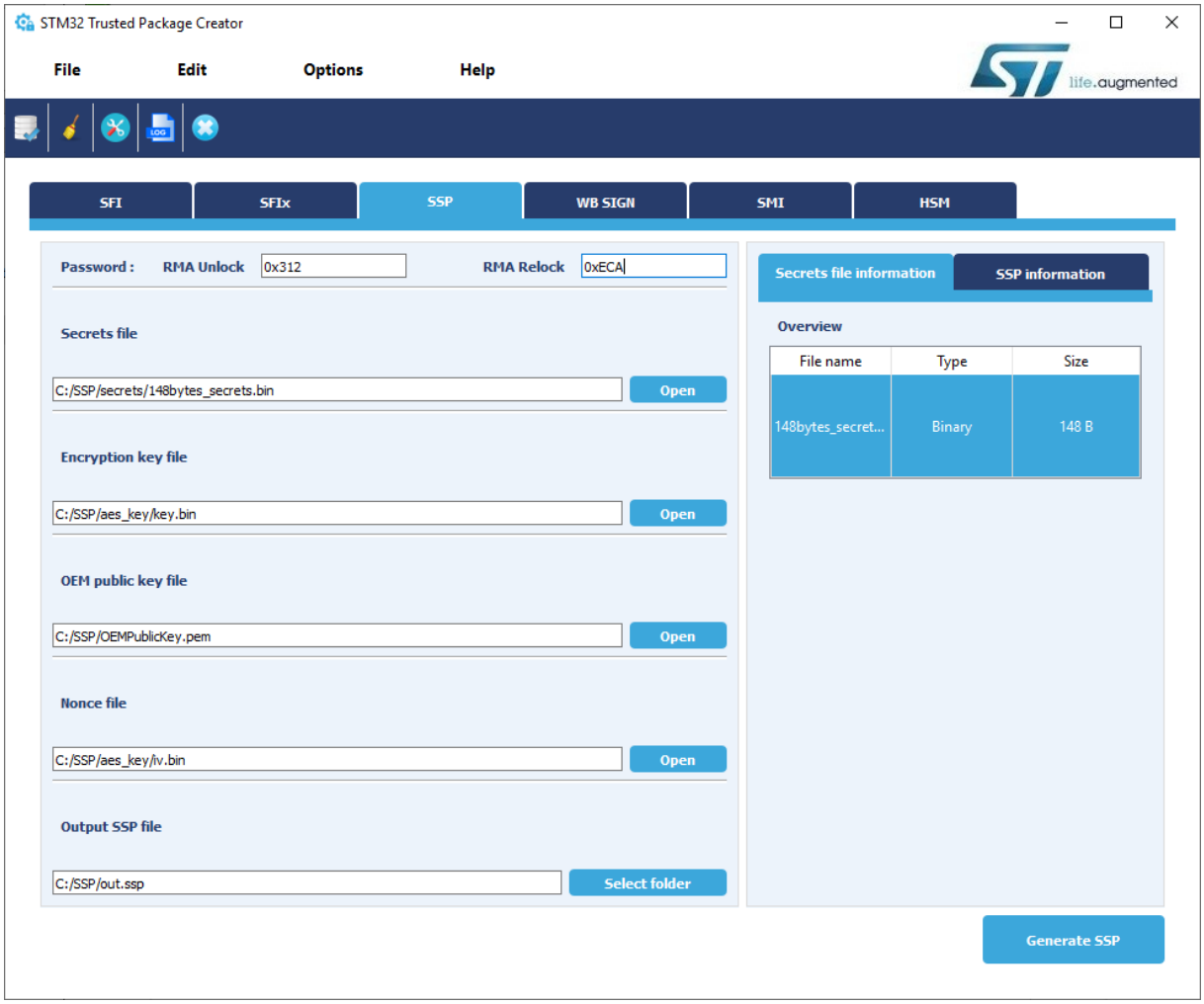
7.1 Encrypted secrets file generation

This section describes how to use the STM32 Trusted Package Creator tool with its graphical user interface and command line interface to generate a SSP file ready for use.

7.1.1 Encrypted secrets file generation with graphical user interface

The STM32 Trusted Package Creator tool GUI presents a SSP tab and in order to generate an encrypted secret file, the user must fill in the input fields with valid values.

Figure 7. STM32 Trusted Package Creator SSP GUI tab



RMA unlock: 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. Note that the sizes must be respected (16 bytes for the key and 12 bytes for nonce).

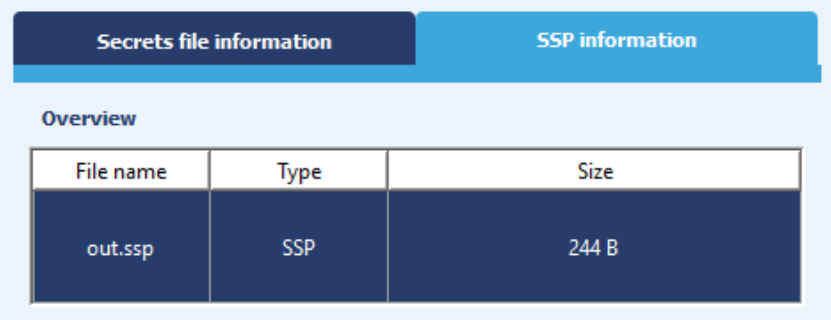
OEM public key file: pem file of size 178 bytes.

Output SSP file: selects the output directory by mentioning the SSP file name to be created with .ssp extension.

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

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

Figure 8. SSP output information



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

7.1.2 Encrypted secrets file generation with the command line interface

The STM32 Trusted Package Creator tool CLI exports an SSP command with various options to generate the encrypted secrets.

-ssp, --ssp

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

-ru, --rma_unlock

Description: RMA unlock password

Syntax: -ru<RMA_Unlock>

<RMA_Unlock>: hexadecimal value 0x0000 to 0x7FFF

-rr, --rma_relock

Description: RMA relock password

Syntax: -rr<relock_value>

<relock_value>: hexadecimal value 0x0000 to 0x7FFF

-b, --blob

Description: binary to encrypt

Syntax: -b<Blob>

<Blob>: secrets file of size 148 bytes

-pk, --pubk

Description: OEM public key file

Syntax: -pk<PubK.pem>

<PubK>: pem file of size 178 bytes

-k, --key

Description: AES-GCM encryption key

Syntax: -k<Key_File>

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

-n, --nonce

Description: AES-GCM nonce

Syntax: -n<Nonce_File>

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

-o, --out

Description: generates the SSP file

Syntax: -out <Output_File.ssp>

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

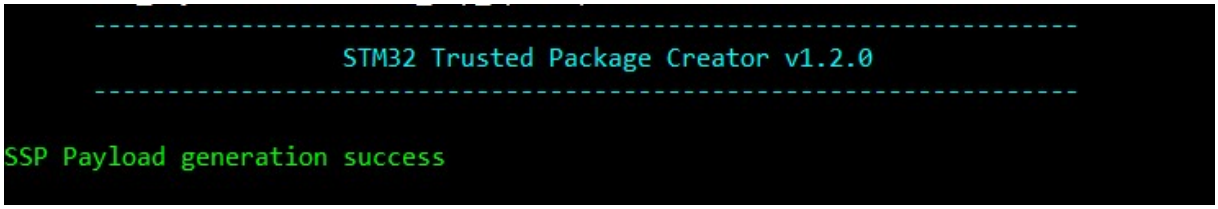
If all input fields are well validated, an SSP file is generated in the directory path already mentioned on the “-o” option.

Command example:

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

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

Figure 9. SSP generation success



7.2 HSM provisioning

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

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

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

Caution: The SSP process using STM32MP1 Series devices works only with HSMv2.

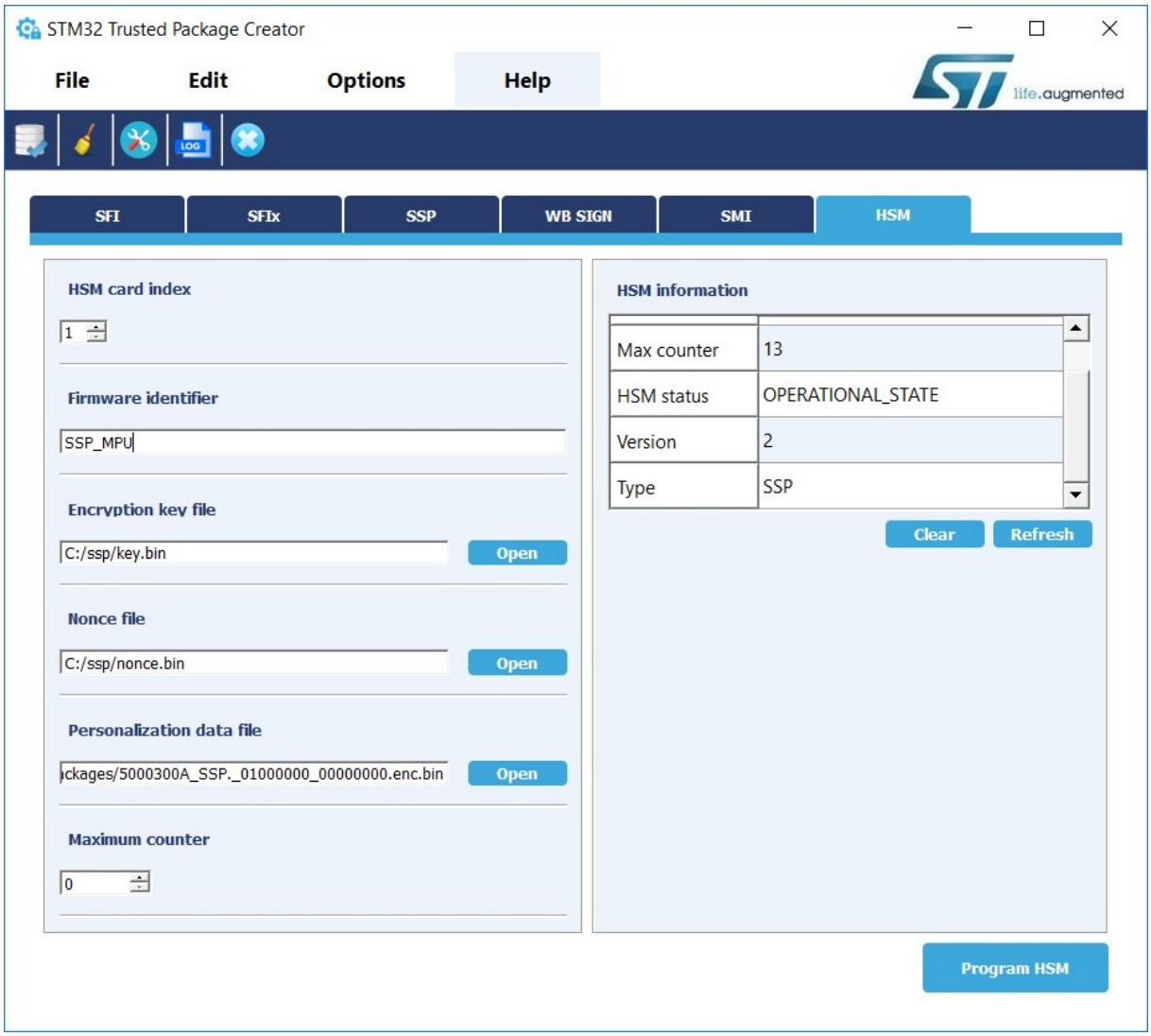
7.2.1 HSM provisioning with the graphical user interface

When an OEM needs to deliver an HSM to a programming house to be deployed as a license generation tool for a relevant STM32 device programming, the OEM must perform some customization on his HSM first. He must program the HSM with all the data needed for the license scheme deployment in the production line. This OEM data is:

- **Counter**: the counter is set to a maximum value that corresponds to the maximum number of licenses that can be delivered by the HSM, it aims to prevent over programming. It is decremented with each license delivered by the HSM. No more licenses are delivered by the HSM once the counter is equal to zero. The maximum counter value must not exceed the predefined maximum value (1 million units).

- **Firmware key**: this key is 32 bytes and is composed of two fields, the initialization vector (IV) or nonce (first field), and the key (last field) that are used to AES128-GCM encrypt the firmware and generate the SSP file. Both fields are 16-byte long, but the last 4 bytes of the nonce must be zero (only 96 bits of the nonce are used in the AES128-GCM algorithm). Both fields must remain secret, so they are encrypted before being sent to the chip. The key and the nonce remain the same for all licenses for a given piece of firmware. However, they must be different for different firmware or different versions of the same firmware.
- **Firmware identifier**: identifies the correct HSM for a given firmware.
- **Personalization data**: 108 bytes that are encrypted before being sent to the chip in cases where each device has its own configuration.

Figure 10. HSM programming tab



The tab parameters are as follows:

- **HSM card index**: specifies the smart card reader slot number.
- **Firmware identifier**: identifies the correct HSM for a given piece of firmware.
- **Encryption key file**: a binary file containing the key used to encrypt the firmware and generate the SSP file. The key is 16-byte long.
- **Nonce file**: a binary file containing the nonce used to encrypt the firmware and generate the SSP file. The nonce is 12-byte long.

- **Personalization data file**: a binary file containing the personalization data used to configure the given device. The data is 108-byte long.
- **Maximum counter**: the maximum number of licenses that can be delivered by the HSM (it aims to prevent over programming).

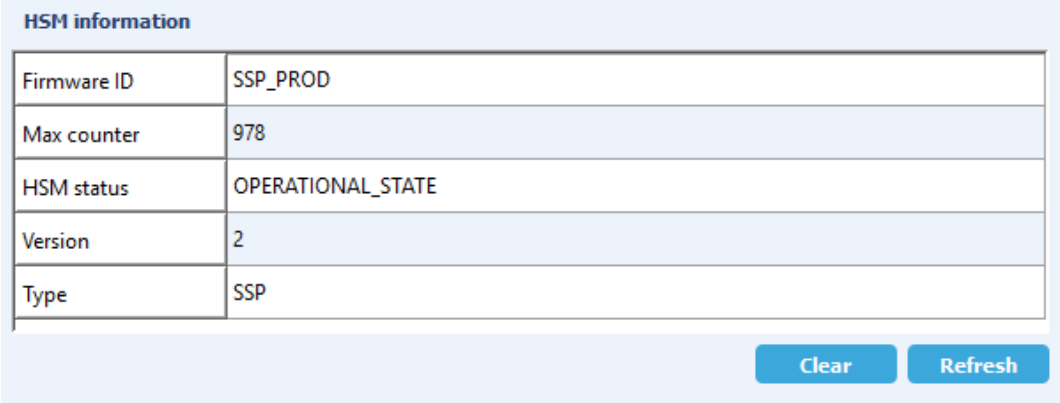
When all fields are properly filled in, the user can program the HSM file by clicking on the program HSM button (the button becomes active).

The STM32 Trusted Package Creator tool provides all personalization package files, ready to be used on the 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 user must select the correct one.

7.2.2 HSM GUI reading information

Once this data is programed into the HSM, the HSM is automatically locked then the user can extract the associated information.

Figure 11. Reading HSM information



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

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

7.2.3 HSM provisioning with command line interface

The following part describes how to use the STM32 Trusted Package Creator tool from the command line interface to program and read information on HSM.

-hsm, --hsm

Description: this command allows the HSM card programmation.

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

-i, --index

Description: selects the card index, number in decimal format.

Syntax: -i <number>

-k, --key

Description: sets the AES-GCM encryption key.

Syntax: -k <Key_File>

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

-n, --nonce

Description: sets the AES-GCM nonce.

Syntax: -n <Nonce_File>

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

-id, --id

Description: sets the firmware identifier.

Syntax: -id <Firmware_Id>

<Firmware_Id>: input as a string format, don't use spaces.

-mc, --maxcounter

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

Syntax: -mc <Max_Counter>

<Max_Counter>: number in decimal format.

-pd, --persoData

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

Syntax: -pd <PersoData_File>

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

-info, --info

Description: gets HSM information.

The STM32 Trusted Packag eCreator tool provides all personalization package files, ready to be used on the 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 user must select the correct one.

Example:

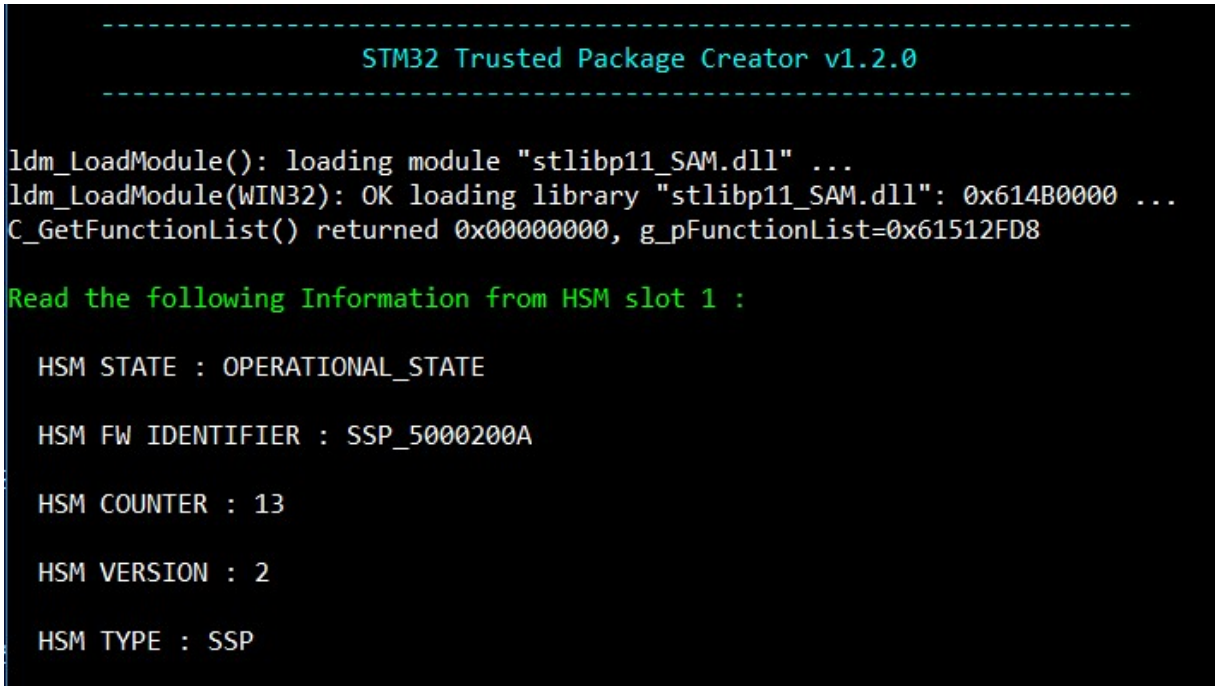
```
STM32TrustedPackageCreator_CLI -hsm -i 1 -k
          "C:\TrustedFiles\key.bin" -n "C:\TrustedFiles\nonce.bin" -id SSP_5000200A -mc 13 -
pd
          "C:\SSP\enc_ST_Perso_MPU.bin"
```

7.2.4 HSM CLI reading information

Get all the associate HSM information using the following command:

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

Figure 12. **Get HSM information in CLI mode**



7.3 SSP procedure with STM32CubeProgrammer

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

The SSP flow can be performed using USB or UART interfaces.

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

-ssp, --ssp

Description: programs an SSP file

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

<ssp_file_path>: SSP file path to be programmed, bin or ssp extensions

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

<hsm=0|1>: sets the user option for HSM use (do not use HSM or 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-ssp-stm32mp157f-dk2-trusted.stm32" hsm=1 slot=1
```

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

Figure 13. **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, then an error message is displayed to indicate the root cause of the issue.

8 Programming protocol extension

Because the SSP process is mainly based on the USB DFU or UART loading protocol, it requires to extend the already defined protocol for STM32MP1 Series [5] .

The protocol addons are part of the SSP secure firmware.

8.1 UART/USART command set

In UART/USART protocol, the 0xF3 partition ID is used for SSP exchange. 0xF3 read downloads the chip certificate and 0xF3 write uploads the encrypted secrets.

8.1.1 Read partition command (0x12)

The read command is not implemented in the default TF-A. It is required in case of SSP to allow STM32CubeProgrammer to download the chip certificate.

Table 6. Read partition command

Byte	Description
1	0x12 = Read partition
2	0xED = XOR of byte 1
-	Wait for ACK or NACK
3	Partition Id = 0xF3 for SSP
4-7	Offset address
8	Checksum byte: XOR (byte 3 to byte 7)
-	Wait for ACK or NACK
9	Number of bytes to be received -1 (N = [0,255])
10	Checksum byte: XOR (byte 9)
-	Wait for ACK or NACK

8.2 USB

Because the TF-A and ROM code share the same USB DFU stack, it is not possible to add more partition ID. The 0x0 partition (used for Flash layout) is reused for the SSP management.

- The certificate is retrieved using the DFU Upload command on the phase ID 0x0.
- The encrypted binary file is sent to the chip using the default download command using the partition 0x0.

Revision history

Table 7. **Document revision history**

Date	Version	Changes
03-Sep-2020	1	Initial release.

Contents

List of tables

List of figures