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

Introduction

The outsourcing of product manufacturing enables 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, 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 new security concept, the secure secret provisioning (SSP), allowing the programming of OEM secrets into the STM32MP1 series 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.

In the remainder of the document:

- STM32MP1 refers to the STM32MP1 series.
- STM32MP15 refers to the STM32MP151, STM32MP153 and STM32MP157 lines of products.
- STM32MP13 refers to the STM32MP131, STM32MP133 and STM32MP135 lines of products.

1 Reference documents

Table 1. Reference documents

Reference number	Document title
[AN4992]	STM32 MCUs secure firmware install (SFI) overview.
[AN5827]	STM32MP13x lines return material analysis (RMA) state.
[AN5275]	USB DFU/USART protocols used in STM32MP1 series bootloaders.
[ROM_CODE]	https://wiki.st.com/stm32mpu/wiki/STM32_MPU_ROM_code_overview.
[SECURE_BOOT]	https://wiki.st.com/stm32mpu/wiki/How_to_secure_STM32_MPU.
[SSP]	https://wiki.st.com/stm32mpu/wiki/How_to_deploy_SSP:_a_step_by_step_approach.
[STM32Trust]	https://www.st.com/stm32trust.
[TrustedFirmware-A]	https://github.com/STMicroelectronics/arm-trusted-firmware(1).

1. *This URL belongs to a third party. It is active at document publication, however STMicroelectronics shall not be liable for any change, move or inactivation of the URL or the referenced material.*

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.
IV	Initialization vector
OEM	Original equipment manufacturer.
OTP	One time programming.
PKH	Public key hash.
PKHTH	Public key hash table hash.
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 PKH or PKHTH: the root of trust for the secure boot authentication.
- RMA passwords: used to change the device life state for hardware investigation purpose.
- Other OEM secrets that may be used by secure software.

The SSP is derived from the STM32 microcontrollers secure firmware install [AN4992], so it reuses the same kind of authentication and encryption mechanism.

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

The SSP embedded code is split in two parts:

- One part in the [ROM_CODE] accessing the device private information.
- Another part in the SSP secure firmware (signed by OEM key) 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.

The SSP process is limited to a serial boot process during the initial provisioning. It can only be run once (except if an issue occurs during the process) and automatically sets the device as a secure close device once provisioning is done. It ensures that the device enables the secure boot feature, protects the secrets and only let OEM signed firmware to be executed.

3.2 OEM data and key exchange

Thanks to STM32MP1 security features and cryptographic algorithm, STM32MP1 supports secure OEM secrets programming in one time programming area (OTP). It ensures OEM firmware protection (confidentiality, authenticity, and integrity) during the STM32MP1 [SECURE_BOOT] process. The ROM code securely decrypts secrets and authenticates the secure firmware which program OTP. The OTP area provides secrets obfuscation thanks to robust hardware mechanisms.

3.2.1 Secrets encryption mechanism

This mechanism consists in having the whole OEM secrets encrypted using AES GCM with an OEM secret 128-bit key (and 128-bit IV) thanks to STM32MP1 trusted package creator tool.

This binary file (244 bytes) is protected and can be sent to the manufacturer for provisioning. It is a required entry for the SSP process.

3.2.2 OEM encryption key protection

A hardware security module (HSM) is able to generate and provide the OEM encryption key (and IV for AES GCM algorithm) in a secure way. Thanks to asymmetric encryption, a dedicated per-device license is generated by the HSM to embed the OEM encryption key and IV.

The HSM manages device authentication and counts the number of generated licenses to avoid over production.

The HSM must be provisioned by the OEM using STM32MP1 trusted package creator tool. The OEM is able to:

- define the OEM AES secret key and nonce used to encrypt secrets.
- choose the personalization data file to identify the device to be provisioned
- choose the maximum number of devices to be provisioned (depends on the STM32HSM-V2 version used).

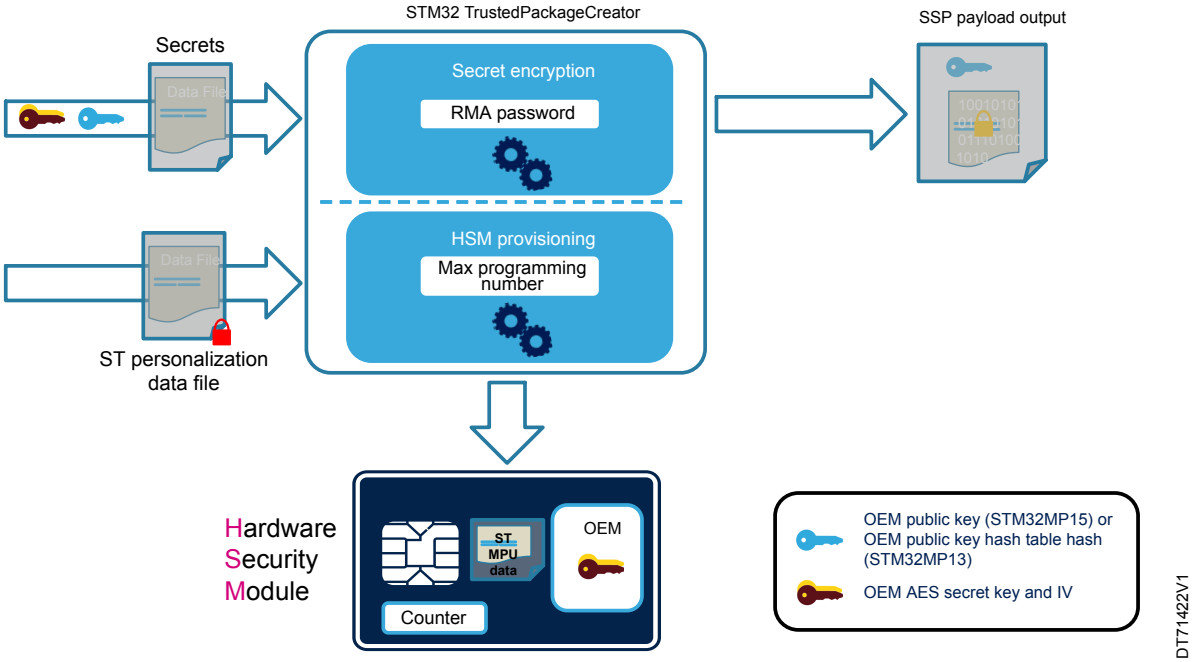
3.2.3 Device authentication

To control the provisioning, the device certificate is requested to generate the dedicated license. The STM32MP1 are default provisioned with an ST certificate used by the HSM to verify the device authenticity. It avoids using the HSM (and the secret files) on untrusted devices.

3.3 SSP process flow

3.3.1 OEM secrets encryption and HSM provisioning

Figure 1. OEM encryption and HSM provisioning



This part is made by the OEM. It prepares the materials to be sent to the manufacturer. After the ECC key pair generation and the secrets generation, it is sent to the STM32 TrustedPackageCreator tool to:

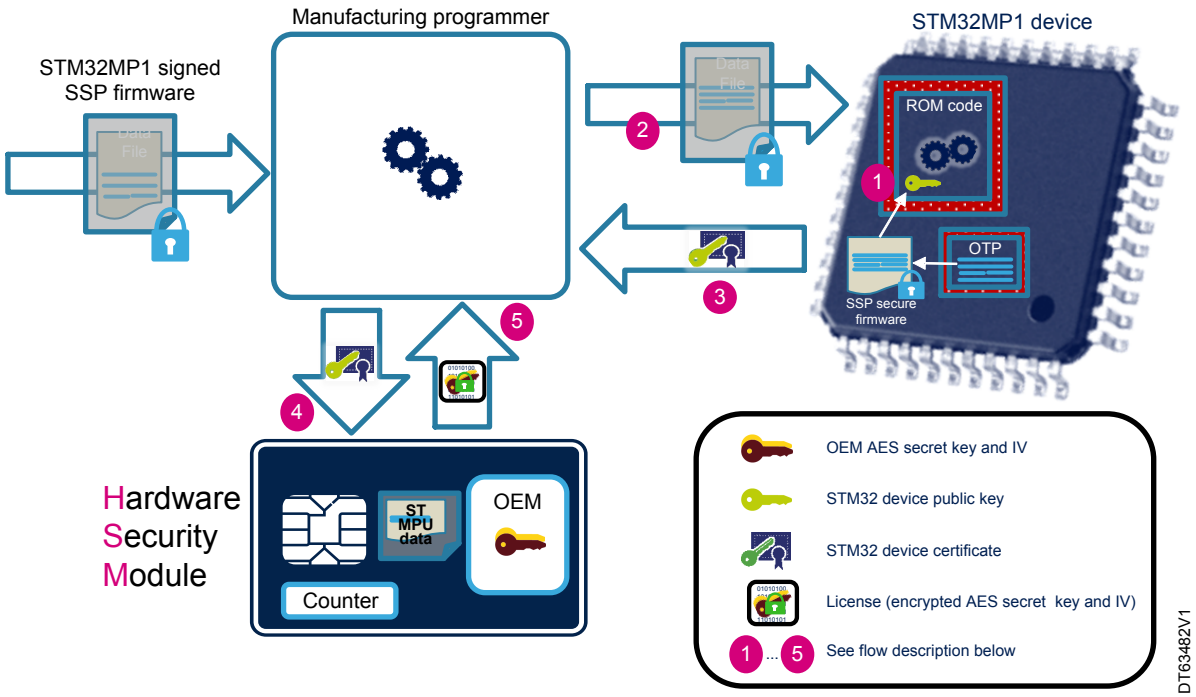
o Generate the final encrypted SSP secret file

o Provision the STM32HSMv2 with the associated device personalization data and the OEM encryption key (and IV).

The SSP firmware must have been signed with the ECC key linked to the OEM PKH (or PKHTH for STM32MP13). See secure boot page in [SECURE_BOOT].

3.3.2 SSP initialization and device authentication

Figure 2. Device authentication

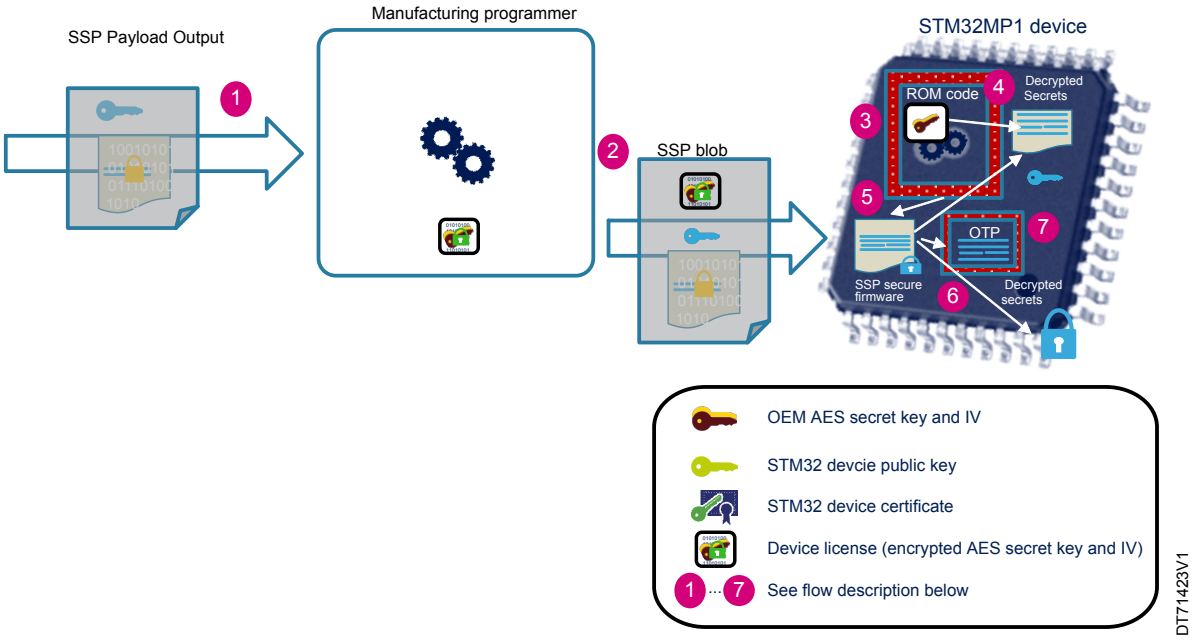


Flow description:

1. The device start ROM code in serial boot and wait for programmer connection
2. The programmer sends the SSP firmware
   This is done twice:
   a. once to initialize the SSP request
   b. once to reconnect to the programmer for certificate exchange
3. The SSP firmware sends the certificate to programmer
4. The programmer sends the certificate to HSM for device authentication
5. The HSM gives to the programmer the device dedicated license with the encrypted OEM key/IV

3.3.3 Secrets download and provisioning

Figure 3. Secrets download and provisioning



Flow description:

1. SSP Payload output is sent to the programmer.
2. Programmer concatenates the license with the payload and sends it to the SSP firmware and resets the device.
3. ROM code decrypts the license to retrieve the OEM AES key.
4. ROM code decrypts and checks the secrets integrity.
5. ROM code authenticates the SSP firmware thanks to the OEM key (or PKHTH in STM32MP13) and jumps in SSP firmware once verified.
6. SSP firmware writes the OEM public key hash or public key hash table hash and changes the device state to secure close.
7. SSP firmware provisions the OTP fuses with the decrypted secrets. Contexts are cleaned and device is reset.

3.4 SSP process steps

The SSP processing is split in three different phases:

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

The SSP firmware can only be run in serial mode.

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

The SYSRAM must be supplied during all the transitions to keep the operations integrity and the exchange context safe.

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

3.4.1 SSP initialization

This step is the entry point to start the SSP process. It is directly started using the SSP secure firmware. It consists of programming the SSP OTP request value to set the device in SSP mode and send the SSP start command to the ROM code.

It sets the device in a specific transition state so that the ROM code and the SSP firmware can access specific information.

The SSP secure firmware is used to start the SSP sequence. It is downloaded by the programmer and loaded by the ROM code (configured in serial boot mode). It automatically programs the SSP OTP request bit (if no SSP sequence previously started) and fills the ROM code context with the start command. It requires a software reset to switch to the next sequence. At this stage, the SSP firmware is not yet authenticated because it does not manage any critical information.

3.4.2 SSP device identification and encrypted secret file upload

This second step starts after a software reset. The device is still in serial boot mode.

The programmer must download one more time the SSP signed firmware. To allow the device identification, a certificate is required. The SSP firmware prepares the device certificate that is sent to the programmer for authentication.

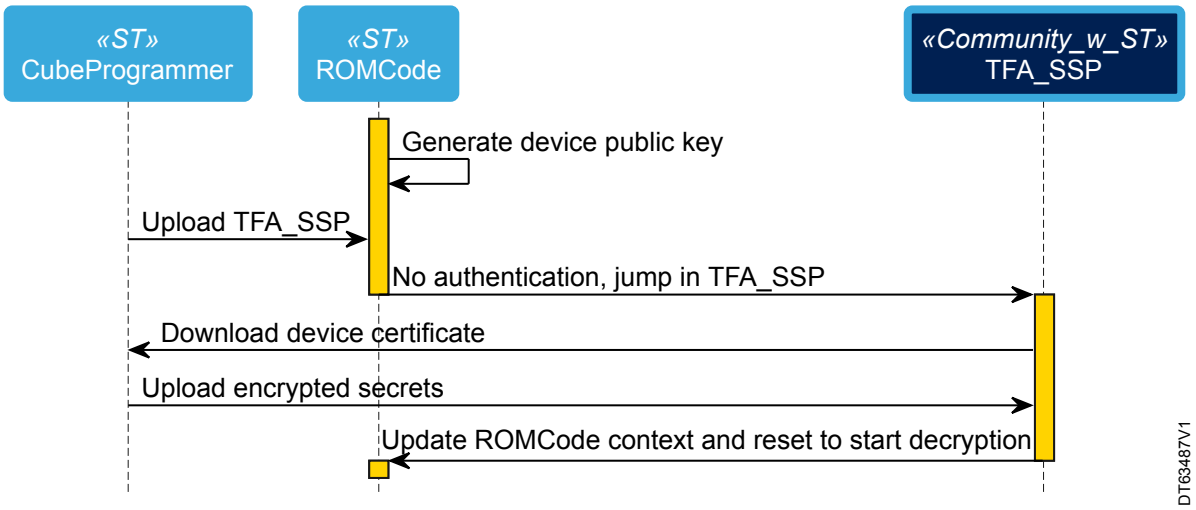
The programming tool downloads the SSP firmware again and it is executed without authentication.

The SSP secure firmware creates the 136-byte certificate using OTP values and generated ECC public key. The certificate is downloaded by the host tool using a dedicated command.

The device certificate is sent to the HSM connected to the programming tool (ex: STM32CubeProgrammer) to authenticate the STM32MP1 device. A specific device license is generated if the certificate has been properly identified. Each generated license decreases the programming counter number in the HSM.

When receiving the device license, the programming tool can upload it, appended with the OEM encrypted secret file previously generated by OEM. Both are uploaded to the device in a single command request, and stored in internal secure memory. When the package is completely received, the SSP secure firmware stops the serial connection and sets a new magic value in the ROM code context to complete the current state. A software reset is automatically initiated in the last step.

Figure 4. Device certificate and secret upload



3.4.3 Secret decryption and OTP programming

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

Once resetting, the ROM code:

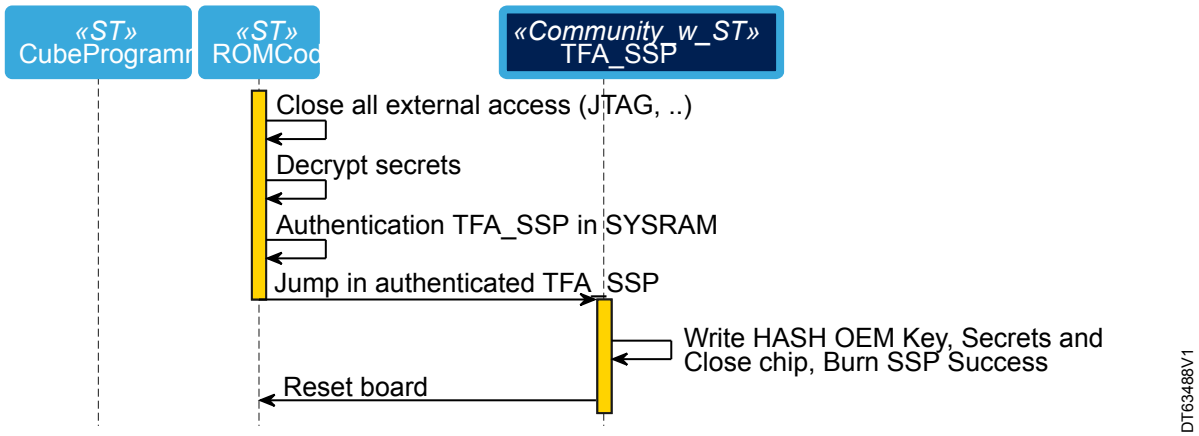
- Controls the license (stored in SYSRAM during the previous step) and decrypts the AES secret key and IV from HSM generated license.
- Decrypts the secrets (stored in SYSRAM during the previous step) using the AES secret key and checks the encryption tag thanks to the AES-GCM algorithm.

- 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 OTPs are free for programming,
- Programs the OEM public key hash,
- Programs the OTP secure close device 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



Upon completion of this sequence, the device is fully secure, secure boot authentication is mandatory based on the OEM public key used, the debugger is closed by default and secrets are available for the secure domain.

3.4.4 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 (SSP firmware authentication error, corrupted license, corrupted secrets file), the attempt counter is incremented till it reaches the maximum tries. The STM32MP1 allows four tries, once achieved, the SSP processing can not be fully ended.

4 Security concern

Once ST information (certificate) has been provisioned to each device, the OEM can rely on secure secrets provisioning and let the OEM-CM populating the devices with their secrets.

The secrets prepared by the OEM are of three different types:

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

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

4.1 Return material authorization (RMA)

Refer to [AN5827].

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

This mechanism reopens the device (RMA unlock) from a secure close state and hides the upper OTP (secrets). On STM32MP15 it is possible to lock it again thanks to RMA relock password. The RMA process is protected by a password (or 2 passwords for STM32MP15). Password(s) are set by the OEM, as part of his secrets during the provisioning step.

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

4.2 OEM authentication key(s)

OEM authentication key(s) is a major part of the secure boot sequence. Secure boot is ECDSA verification based, the authentication key is used to validate the signature of the loaded binary.

The key hash on STM32MP15 or key hash table hash on STM32MP13 (to manage multiple keys) is stored into the device to be used as a reference during the secure boot authentication process. On STM32MP15, the ROM code must check the key against the hash that has been safely programmed in OTP. On STM32MP13, the key is verified by checking the hash in the STM32 header and verifying the hash table from the header against the hash table hash stored 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 customers.

The OTP memory is organized as 32-bit words.

On the STM32MP1 microprocessor:

- One OTP word is reserved for RMA password(s): OTP 56.
- 37 free words are reserved for customer usage, the secret size can be up to 148 bytes: OTP 59 to 95 (OTP 57 and 58 are defined as reserved for MAC address and cannot be provisioned using SSP).
  A 148-byte binary OTP secret file must be used as the reference to construct the secret file.

Note: In the secret files, bytes set to 0 and aligned on 32-bit word are ignored during the OTP programming.

5.1 RMA password(s)

The RMA password(s) are chosen by OEM. There are part of the secret file and placed as the first 4-byte word.

On the STM32MP15xx, 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. STM32MP15 RMA 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)

On the STM32MP13xx, the RMA password is provided using 32 bits.

Table 4. STM32MP13 RMA password

Bit [31-0]
32-bit password

Examples:

RMA unlock: 0x87654321

Final OTP: 0x87654321 (8-bit coding style: 21 43 65 87)

To ease the RMA passwords creation and avoid any formatting issue, the STM32TrustedPackageCreator 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 5. 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 STM32TrustedPackageCreator 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     00 00 00 00 5e 03 43 f4 28 58 59 7c 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 red, 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.

The STM32TrustedPackage tool creates an .ssp file as described below:

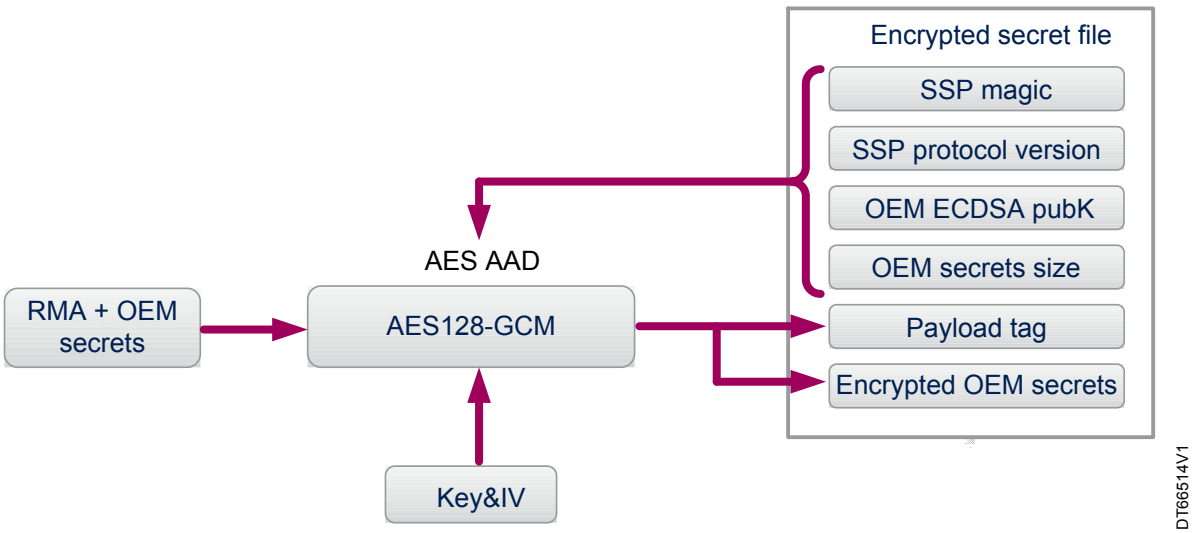
Table 6. Encrypted secret file layout

-	Size	Content
SSP magic field	4 bytes	'SSPP': magic identifier for SSP payload,
SSP protocol version	4 bytes	SSP version used,
OEM ECC public key or OEM public keys HASH table HASH	64 bytes	OEM ECC public key (STM32MP15) or OEM ECC public keys hash table hash,
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 (AES GCM tag),
Encrypted OEM secrets	152 bytes	Encrypted OEM secrets. Including RMA password, excluding 57 and 58 OTP defined as MAC ADDR.

The first layout part (SSP identification fields, ECC public key or public key hash table hash, and secret size) is used as additional authenticated data (AAD) to generate the payload tag. This is checked by the ROM code during the decryption. It ensures the integrity of the downloaded data. The generated encrypted secret file usage is limited to a specific HSM as the key and IV used for encryption are stored in the HSM.

This encrypted file is automatically generated by STM32TrustedPackageCreator tool.

Figure 6. Encryption file scheme



6 SSP secure firmware

The SSP secure firmware is derived from the standard secure firmware (see [TrustedFirmware-A]) used as the first stage bootloader from the STM32MP1 ecosystem release. It includes additional features for the SSP processing.

It reuses the serial boot protocol (USB and UART) from the generic code. The major change is the SSP library that manages the different SSP phases. Other updates are linked to size optimization in order to reduce the binary to only the required functions.

6.1 Build

The build instructions are described in the TF-A wiki page, see [SSP].

6.2 Signed binary

The SSP firmware must be STM32 signed with the OEM private key to ensure the OEM firmware authenticity. It is authenticated by the ROM at the final SSP stage based on the HASH provided with the OEM secrets.

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 perform using SSP tools:

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

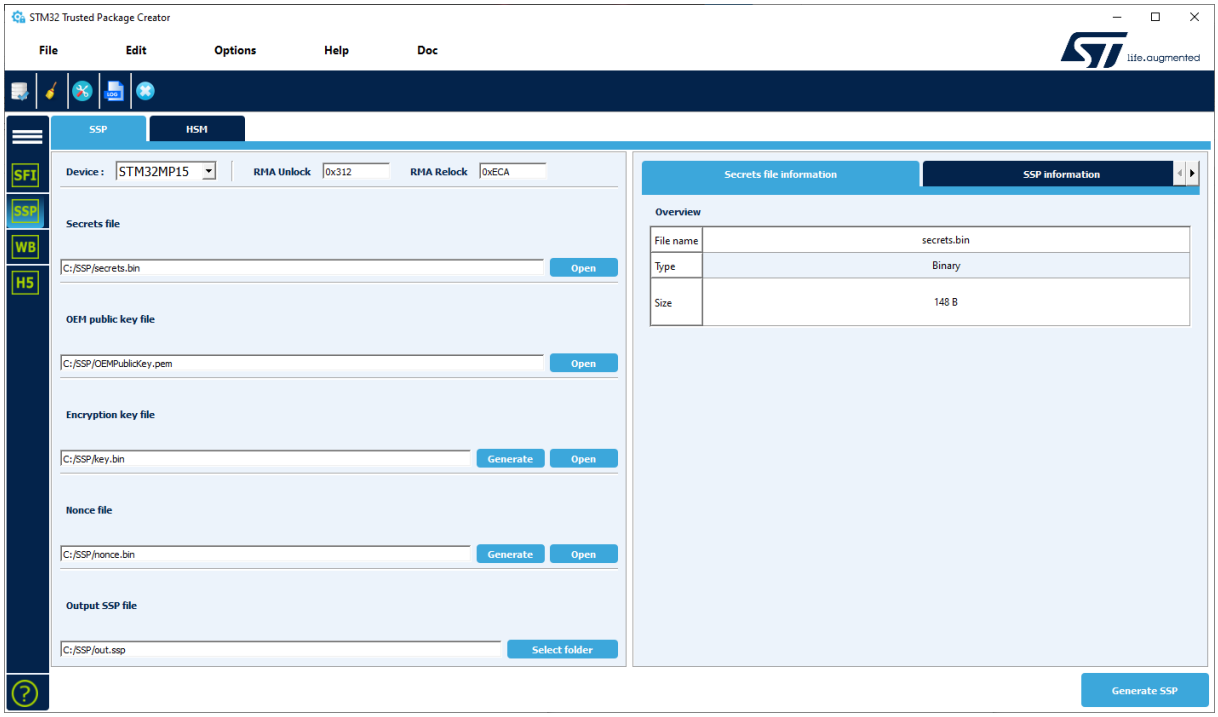
7.1 Encrypted secrets file generation

This section describes how to use the STM32TrustedPackageCreator 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 TrustedPackageCreator tool GUI presents a SSP tab. In order to generate an encrypted secret file, the user must fill in the input fields with valid values.

Figure 7. STM32TrustedPackageCreator SSP GUI tab



Device: choose STM32MP13 or STM32MP15

RMA unlock: unlock password, hexadecimal value

RMA relock: relock password, hexadecimal value from 0x0000 to 0x7FFF (only for STM32MP15)

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.

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

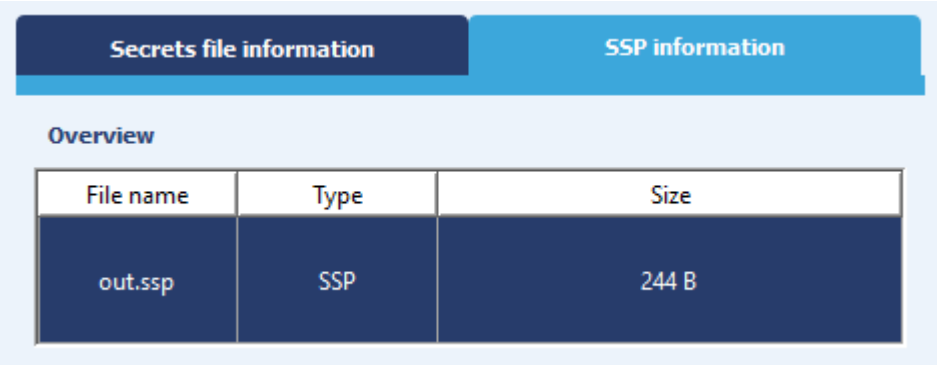
Encryption key and nonce files: the encryption key and nonce file can be selected by entering their paths (absolute or relative), or by selection with the **Open** button. The **Generate** button can be used to directly generate a random value.

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)

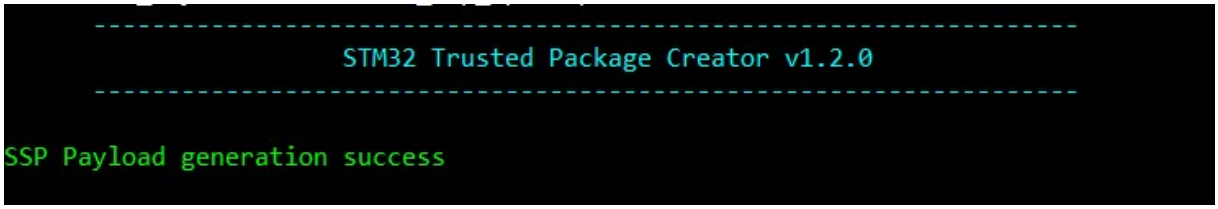
If all input fields are well validated, an SSP file is generated in the directory path already mentioned on the “-o” option.

Command example (for STM32MP15xx):

```
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 STM32MP1 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 STM32MP1 secure programming devices that are chosen via a personalization data at the OEM site.

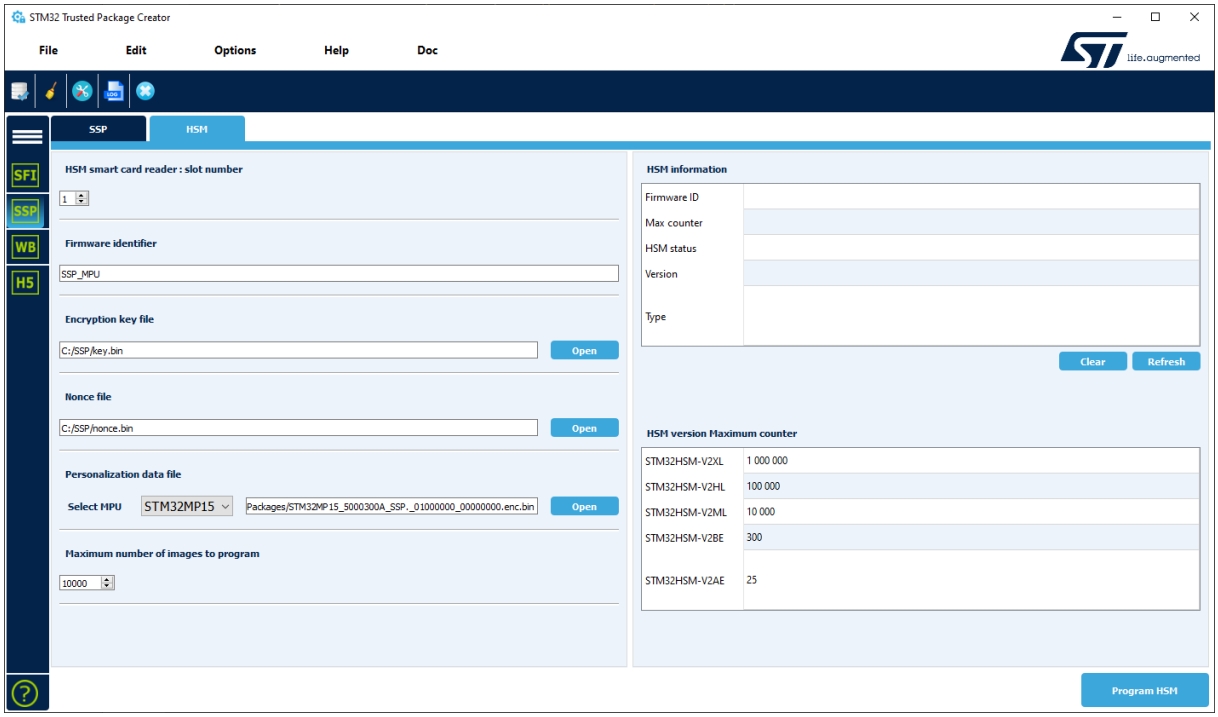
Caution: The SSP process using STM32MP1 series devices works only with HSMv2.

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.
- **Select MPU**: Select the personalization date used to identify the device. First, select the MPU device STM32MP15 or STM32MP13 for an easier filter.
- **Maximum counter**: the maximum number of licenses that can be delivered by the HSM (it aims to prevent over programming).

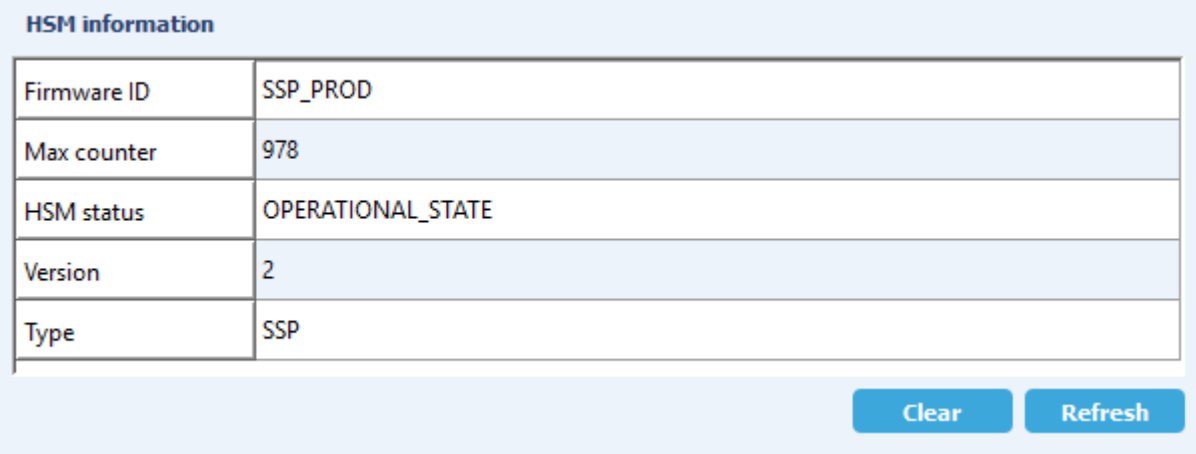
When all fields are properly filled in, the user can program the HSM file by clicking on the program HSM button (the button becomes active).

The STM32TrustedPackageCreator 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 installation 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, do not 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 STM32TrustedPackageCreator 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 installation path. Each file name starts with a number, which is the product ID of the device. The user must select the correct one.

Example:

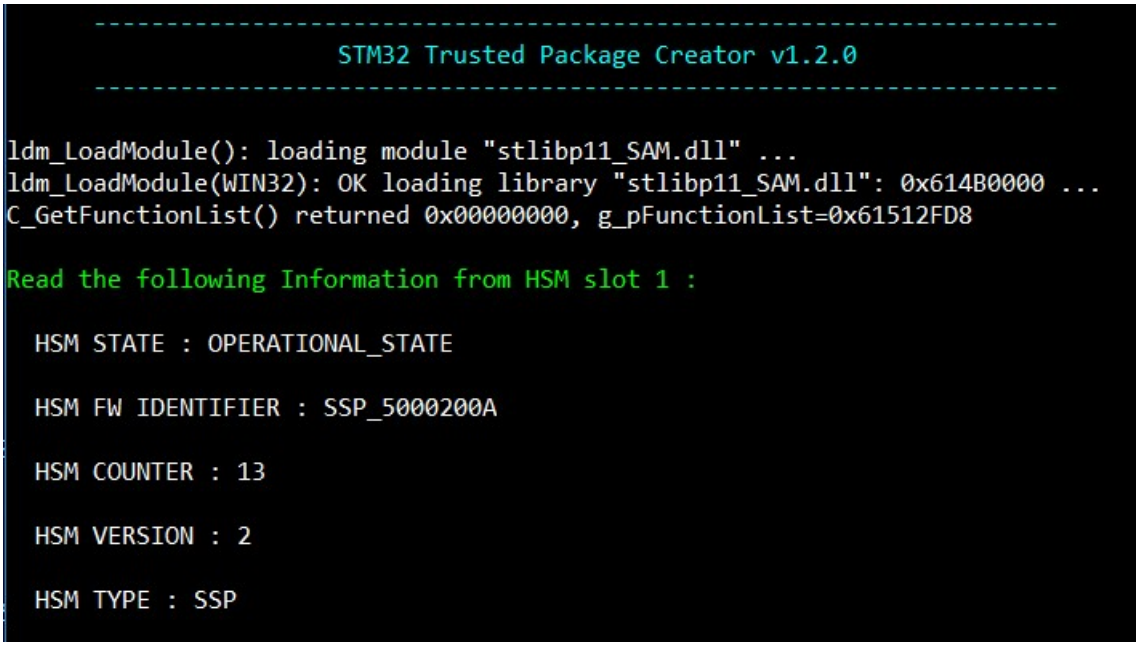
```
STM32TrustedPackageCreator_CLI -hsm -i 1 -k
          "C:\TrustedFiles\key.bin" -n "C:\TrustedFiles\nonce.bin" -id SSP_5000200A -mc 13 -p
d
          "C:\SSP\enc_ST_Perso_MPU.bin"
```

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 (example)

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 STM32TrustedPackageCreator tool. The STM32CubeProgrammer supports communication with ST HSMs (hardware secure modules based on smart card) to generate a license for the connected STM32MP1 MPU device during the SSP install.

The SSP flow can be performed using both 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)

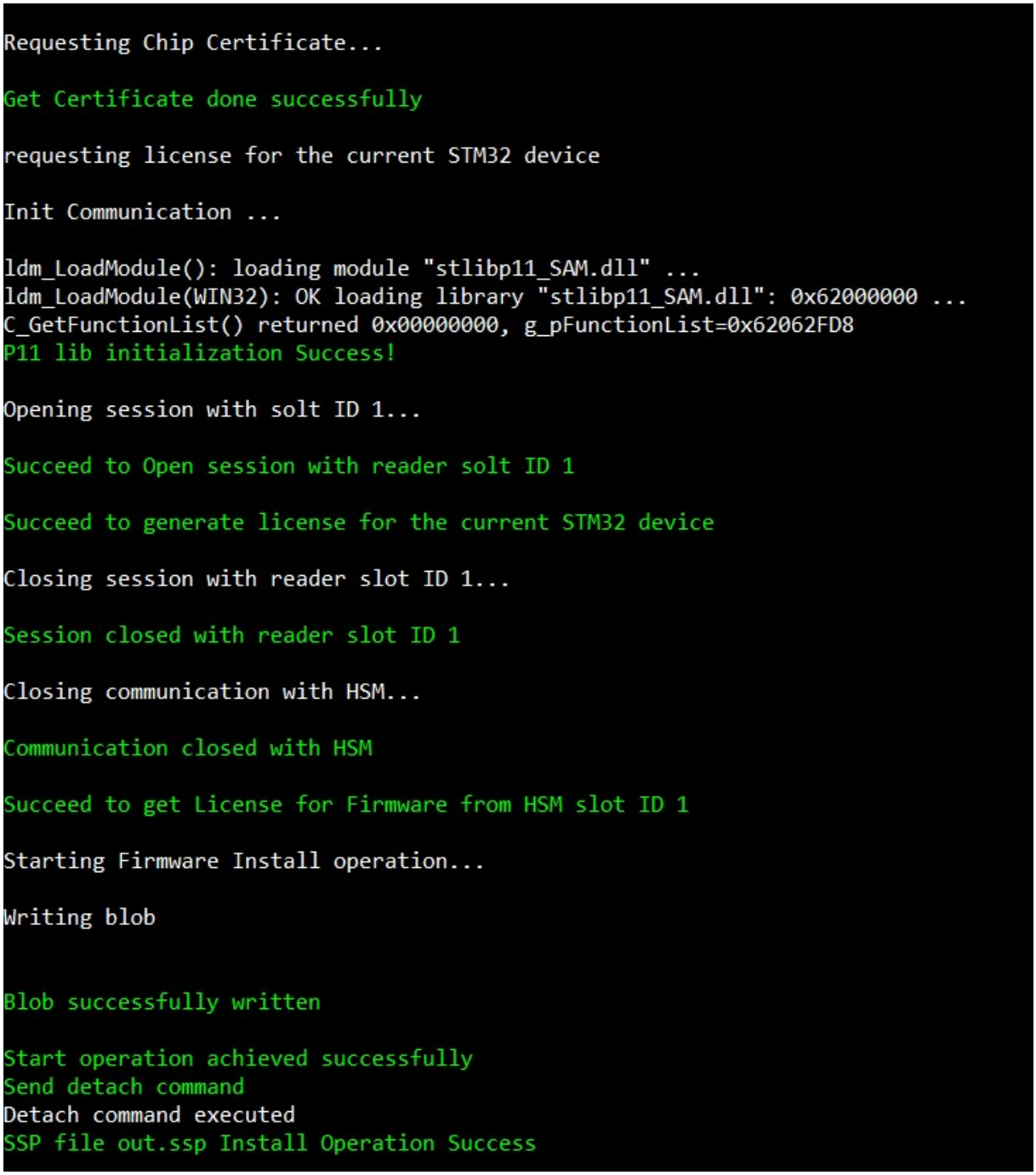
reader slot ID if HSM is used (if hsm=1)

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**



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.

STM32 CubeProgrammer is delivered as an example. See [STM32Trust] for partner tools.

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, see [AN5275].

The protocol add-ons is 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 device 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 so that the STM32CubeProgrammer can download the device certificate.

Table 7. 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 device using the default download command using the partition 0x0.

8.2.1 STM32MP13

TF-A SSP is doing a USB re-enumeration of the USB. It can refresh the partition table to add the specific SSP partition.

The **0xF3** partition is dedicated to the SSP communication to upload certificate and download data.

8.2.2 STM32MP15

Because TF-A and ROM code share the same USB DFU stack, it is not possible to add more partition ID.

The partition ID 0 is reused to download and upload the SSP data.

8.2.3 DFU download

Table 8. STM32MP13 DFU SetOffset command

Byte	Description
1	Phase ID: 0xF3 for SSP binary download
2-5	Offset int the partition

Table 9. STM32MP15 SetOffset command

Byte	Description
1	Phase ID: 0x00 for SSP binary download
2-5	Offset int the partition

Start command is still required to finalize the transfer and update to the next phase.

Table 10. DFU Start command

Byte	Description
1	0xFF
2-5	Address

8.2.4 DFU upload

DFU upload has been updated to allow retrieving the device certificate.

Table 11. STM32MP13 DFU Upload command

Byte	Description
1	Phase ID: Use 0xF3 for device certificate
2-5	Download address (not used)
6-9	Offset (not used)

Table 12. STM32MP15 DFU Upload command

Byte	Description
1	Phase ID: Use 0x00 for device certificate
2-5	Download address (not used)
6-9	Offset (not used)

Example for SSP TF-A execution:

1. Upload certificate
   Byte 1 **0xF3** (or Byte 1 **0x00** on STM32MP15)
   Byte 2-5 <Address>
   Byte 6-9 0x00000000
2. License and encrypted secrets file download
   Byte 1 **0xF3** (or Byte 1 **0x00** on STM32MP15)
   Byte 1 **0xF3** (or Byte 1 **0x00** on STM32MP15)
   Byte 2-5 <Address>
3. Send start command to end process
   Byte 1 0xFF
   Byte 2-5 0xFFFFFFFFF

4. Request DFU_DETACH after manifestation
   Byte 1 0x00
   Byte 2-5 0xFFFFFFFF
   Byte 6 0x01

Revision history

Table 13. **Document revision history**

Date	Version	Changes
03-Sep-2020	1	Initial release.
03-Mar-2023	2	Introduced STM32MP131, STM32MP133 and STM32MP135 lines of products. All sections have been updated. Addition of: Section 8.2.1 STM32MP13. Section 8.2.2 STM32MP15. Section 8.2.3 DFU download. Section 8.2.4 DFU upload.

Contents

List of tables

List of figures