STM32MP1 Series Signing Tool software description

Introduction

The STM32MP1 Series Signing Tool software (named STM32MP1-SignTool in this document) is integrated in the STM32CubeProgrammer (STM32CubeProg).

STM32MP1-SignTool is a key tool that guarantees a secure platform and ensures the signing of binary images using ECC keys generated by STM32MP1-KeyGen software (refer to the user manual *STM32MP1 Series Key Generator software description* (UM2542) for more details).

The signed binary images are used during the STM32MP1 Series MPU secure boot sequence that supports a trusted boot chain. This action ensures an authentication and integrity check of the loaded images.

STM32MP1-SignTool generates a binary image file, a public key file and a private key file.

The binary image file contains the binary data to be programmed for the device.

The public key file contains the ECC public key in PEM format, generated with STM32MP1-KeyGen.

The private key file contains the encrypted ECC private key in PEM format, generated with STM32MP1-KeyGen.

A signed binary file can also be generated from an already signed file with the batch file mode. In that case, the image entry point, the image load address and the image version parameters are not mandatory.

1 Install STM32MP1-SignTool

This tool is installed with the STM32CubeProgrammer package (STM32CubeProg). For more information about the set-up procedure, refer to the section 1.2 of the user manual *STM32CubeProgrammer software description* (UM2237).

This software applies to the STM32MP1 Series Arm®-based MPUs.

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

2 STM32MP1-SignTool command line interface

The following sections describe how to use STM32MP1-SignTool from command line.

2.1 Commands

The available commands are listed below:

- `--binary-image (-bin)`
  - Description: binary image file path (.bin extension)
  - Syntax: `-bin /home/User/binaryFile.bin`
- `--image-version (-iv)`
  - Description: enters the image version of the signed image file.
  - Syntax: `-iv <version_number>`
- `--private-key (-prvk)`
  - Description: private key file path (.pem extension)
  - Syntax: `-prvk <private_key_file_path>`
  - Example: `-prvk ../privateKey.pem`
- `--public-key -pubk`
  - Description: public key file paths
  - Syntax: `-pubk <File_Path{1..8}>`
    - for header v1: use just one key path for STM32MP15xx products
    - for header v2: use 8 key paths for STM32MP13xx products
- `--password (-pwd)`
  - Description: password of the private key (this password must contain at least four characters)
  - Example: `-pwd azerty`
- `--load-address (-la)`
  - Description: image load address
  - Example: `-la <load_address>`
- `--entry-point (-ep)`
  - Description: image entry point
  - Example: `-ep <entry_point>`
- `--option-flags (-of)`
  - Description: image option flags (default value = 0)
  - Example: `-of <option_flags>`
- `--algorithm (-a)`
  - Description: specifies one of the prime256v1 (value 1, default) or brainpoolP256t1 (value 2).
  - Example: `-a <2>`
- `--output (-o)`
  - Description: output file path. This parameter is optional. If not specified, the output file is generated at the same source file path (for example, the binary image file is *C:\BinaryFile.bin*). The signed binary file is *C:\BinaryFile_Signed.bin*.
  - Syntax: `-o <Output_File_Path>`
- `--type (-t)`
  - Description: binary type. Possible values are `ssbl`, `fsbl`, `teeh`, `teed`, `teex` and `copro`.
  - Syntax: `-t <type>`
- `--silent (-s)`
  - Description: no message displayed for replacing an existing output file

- `--help (-h and -?)`
  - Description: shows help.
- `--version (-v)`
  - Description: displays the tool version.
- `--enc-dc (-encdc)`
  - Description: encryption derivation constant for FSBL encryption [header v2]
  - Syntax: `-encdc <Deriv_hexVal>`
- `--enc-key (-enck)`
  - Description: OEM secret file for FSBL encryption [header v2]
  - Syntax: `-enck <Key_Path>`
- `--dump-header (--dump)`
  - Description: parse and dump image header
  - Syntax: `-dump <File_Path>`

2.2 Examples for STM32MP1-SignTool

The following examples show how to use STM32MP1-SignTool:

- **Example 1**

```
-bin /home/User/BinaryFile.bin -pubk /home/user/publicKey.pem -prvk /home/user/
privateKey.pem -iv 5 -pwd azerty -la 0x20000000 -ep 0x08000000
```

The default algorithm (prime256v1) is selected and the option flags value is 0 (default value). The signed output binary file (BinaryFile_Signed.bin) is created in the */home/user/* folder

- **Example 2**

```
-bin /home/User/Folder1/BinaryFile.bin -pubk /home/user/publicKey.pem -prvk /home/user/
privateKey.pem -iv 5 -pwd azerty -s -la 0x20000000 -ep 0x08000000 -a 2 -o /home/user/
Folder2/Folder3/signedFile.bin
```

The BrainpoolP256t1 algorithm is selected in this case. Even if Folder2 and Folder3 does not exist, they are created. With the `-s` command, even if a file exists with the same specified name, it is automatically replaced without any message.

- **Example 3**

Sign a binary file using header version 2 that includes 8 public keys for authentication flow.

```
./STM32MP_SigningTool_CLI.exe -bin /home/user/input.bin -pubk publicKey00.pem
publicKey01.pem publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem
publicKey06.pem publicKey07.pem -prvk privateKey00.pem -pwd azerty -t fsbl -iv
0x00000000 -la 0x20000000 -ep 0x08000000 -of 0x80000001 -o /home/user/output.stm32
```

- **Example 4**

Sign a binary file using header version 2 that includes 8 public keys for authentication plus encryption flow.

```
./STM32MP_SigningTool_CLI.exe -bin /home/user/input.bin -pubk publicKey00.pem
publicKey01.pem publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem
publicKey06.pem publicKey07.pem -prvk privateKey00.pem -iv 0x00000000 -pwd azerty -la
0x20000000 -ep 0x08000000 -t fsbl -of 0x00000003 -encdc 0x25205f0e -enck /home/user/
OEM_SECRET.bin -o /home/user/output.stm32
```

- **Example 5**

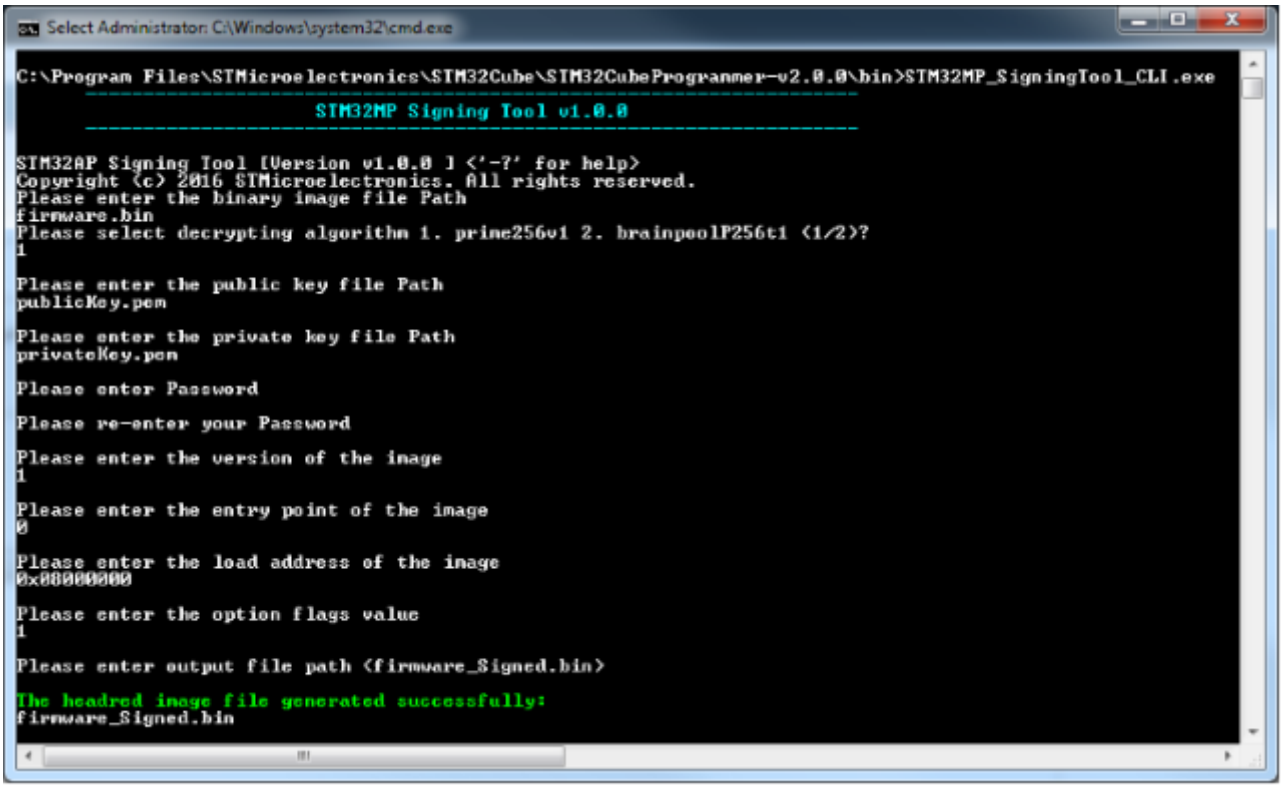
Verify the resulted image by parsing the output file and check each header field.

```
./STM32MP_SigningTool_CLI.exe -dump /home/user/output.stm32
```

2.3 Standalone mode

When executing STM32MP1-SignTool in Standalone mode, an absolute path must be entered first, then a password is requested twice for confirmation, as shown in the figure below.

Figure 1. STM32MP1-SignTool in Standalone mode



Next steps are the following:

- Select one of the two algorithms.
- Enter the image version, the image entry point and the image load address.
- Enter the option flags value.

Another output file path can be specified if needed or press enter to continue with the existing one.

2.4 PKCS#11 solution

The signed binary images are used during the STM32MP secure boot sequence that supports a trusted boot chain, this action ensures an authentication and integrity check of the loaded images.

The classic signing command requests to put all public and private keys as input files which are directly accessible by any person who is allowed to execute the signing service, ultimately it can be considered as a security leak. There are several solutions to protect keys against any attempts to steal keys data and based on this context, the PKCS#11 solution has been adopted.

The PKCS#11 API can be used to handle and store cryptographic keys. This interface specifies how to communicate with cryptographic devices such as HSMs (hardware security modules) and smart cards. The purpose of these devices is to generate cryptographic keys and sign information without revealing private-key material to the outside world.

Software applications can call the API to use these objects for:

- Generate symmetric/asymmetric keys
- Encryption and decryption
- Computing and verifying the digital signature

PKCS #11 presents to applications a common, logical view of the device that is called a cryptographic token and it assigns a slot ID to each token. An application identifies the token that it wants to access by specifying the appropriate slot ID.

The STM32SigningTool is used to manage the key objects stored on smart cards and similar PKCS#11 security tokens where sensitive private keys never leave the device.

The STM32SigningTool uses the PKCS#11 interface to manipulate and sign input binaries basing on ECDSA public/private keys which are stored in security tokens (hardware or software).

2.4.1 Additional PKCS#11 commands

- `--module (-m)`
  - Description: specify a PKCS#11 module/Library path to load (dll, so)
  - Syntax:`-m <Module_Path>`
- `--key-index (-ki)`
  - `--key-index (-ki)`
  - Description: list of used keys indexes in hex format.
    - Use 1 index for header v1 and 8 indexes for header v2 (separated by space)
  - Syntax: `-ki <values>`
- `--slot-index (-si)`
  - Description: specify the index of the slot to use (default 0x0)
  - Syntax:`-si <hexValue>`
- `--active-keyIndex (-aki)`
  - Description: specify the actual active key index (default 0)
  - Syntax: `-aki < hexValue >`

2.4.2 PKH/PKTH file generation

After the processing of the signing operation, the tool systematically generates the PKH files to use after for OTP Fuse.

- PKH file named **`pkcsHashPublicKey0x`**`{active_key_index}.bin for header v1`
- PKTH file named **`pkcsPublicKeysHashHashes`**`.bin for header v2`

2.4.3 Examples

The tool can sign input files for both header v1 and header v2 with a minimal difference in the command line.

- Header v1

```
-bin input.bin -iv <value> -pwd <value> -la <value> -ep <value> -t <type> -of <value>
--key-index <value> -aki 0 --module <module_path> --slot-index <index> -o output.stm32
```

- Header v2

```
-bin input.bin -iv <value> -pwd <value> -la <value> -ep <value> -t <type> -of <value> --
key-index <value0> <value1> <value2> <value3> <value4> <value5> <value6> <value7> -aki
<active_index> --module <module_path> --slot-index <index> -o output.stm32
```

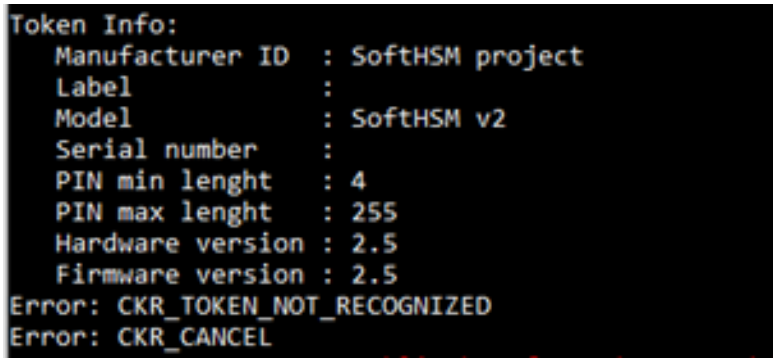
If there is an error on the command line or if the tool is not able to identify the key objects that match, an error message is displayed and indicates the problem's source.

The SigningTool is able only to use preconfigured HSMs and it is not designed to manage or create new security objects. therefore it is necessary to install and resort to some free software, to setup a suitable environment then generates keys and get information about objects.

Error examples:

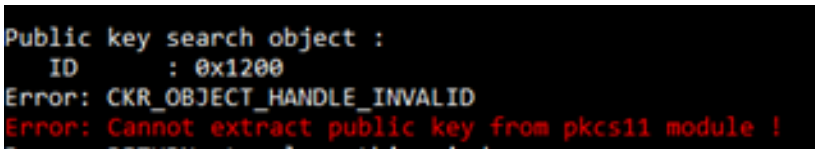
- Invalid slot index

Figure 2. HSM TOKEN_NOT_RECOGNIZED



- Unknown key object that is mentioned in `--key-index command`

Figure 3. HSM OBJECT_HANDLE_INVALID



The tool treats sequentially the objects and if it is not able to identify the key objects that match at first time, the signing operation stops the process, and an error message is displayed to indicate the problem's source.

Revision history

Table 1. **Document revision history**

Date	Version	Changes
14-Feb-2019	1	Initial release.
26-Nov-2021	2	Updated: • Section 2.1 Commands • Section 2.2 Examples for STM32MP1-SignTool • Added Section 2.4 PKCS#11 solution

Contents