STM32 signing tool software description

Introduction

The STM32 signing tool software (named STM32-SignTool in this document) is integrated in the STM32CubeProgrammer (STM32CubeProg).

STM32-SignTool is a key tool that guarantees a secure platform and ensures the signing of binary images using ECC keys generated by STM32-KeyGen software (refer to the user manual *STM32 key generator software description* (UM2542) for more details).

The signed binary images are used during the STM32 secure boot sequence that supports a trusted boot chain. This action ensures an authentication and integrity check of the loaded images.

STM32-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 STM32-KeyGen.

The private key file contains the encrypted ECC private key in PEM format, generated with STM32-KeyGen.

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

This document applies to the products listed in the table below.

Table 1. Applicable products

Product type	Part number or product series
Microcontroller	STM32N6 series
Microprocessor	STM32MP1 and STM32MP2 series

In the following sections, STM32 refers to the products listed in the above table, unless otherwise stated.

1 Install STM32-SignTool

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

This software supports STM32 products based on the Arm® Cortex® processor.

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

2 STM32-SignTool command line interface

The following sections describe how to use STM32-SignTool from the command line.

2.1 Commands

The available commands are listed below:

- `--binary-image(-bin), --input(-in)`
  - Description: binary image file path (.bin extension)
  - Syntax: 1 `-bin /home/User/binaryFile.bin`
  - Syntax: 2 `-in /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 and greater: use eight key paths for others
- `--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>`

- `--align (-align)`
  - Description: align the payload to the 0x400 offset by adding padding bytes at the beginning of the payload
  - Notice: the `--align` option is required for header version 2.3 on STM32N6 microcontrollers and does not apply to MPU devices
- `--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>`
- `--header-version (-hv)`
  - Description: signing header version, possible values: 1, 2, 2.1, 2.2, and 2.3
  - Example for STM32MP15xx: `-hv 2`
  - Example for STM32MP25xx: `-hv 2.2`
  - Example for STM32N6xxx: `-hv 2.3`
- `--no-keys (-nk)`
  - Description: adding empty header without key options
  - Notice: need to disable authentication option with option flags command

2.2 Examples for STM32-SignTool

The following examples show how to use STM32-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 flag 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 do 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 eight public keys for the authentication flow.

```
./STM32_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 eight public keys for authentication plus encryption flow.

```
./STM32_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.

```
./STM32_SigningTool_CLI.exe -dump /home/user/output.stm32
```

- **Example 6**
  Add a header without signing and without deploying keys.

```
STM32_SigningTool_CLI.exe -in input.bin -nk -of 0x0 -iv 1 -hv 2.2 -o output.stm32
```

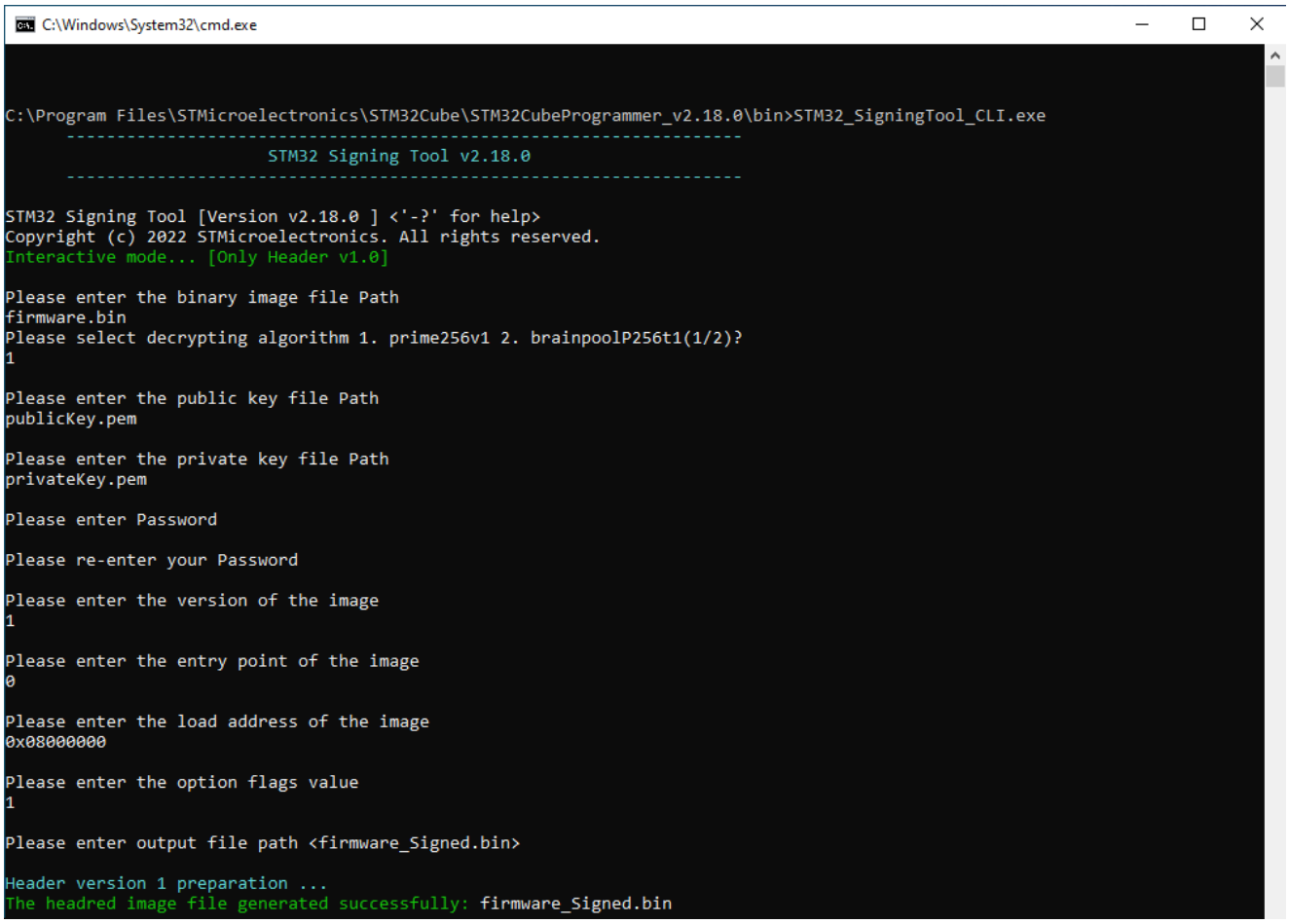
- **Example 7**
  Sign a binary file using header version 2.3 for STM32N6 products that include eight public keys for authentication.

```
./STM32_SigningTool_CLI.exe -bin /home/user/FSBL_N6.bin -pubk publicKey00.pem publicKey
01.pem publicKey02.pem publicKey03.pem publicKey04.pem publicKey05.pem publicKey06.pem
publicKey07.pem -prvk privateKey00.pem -hv 2.3 -iv 0x00000001 -pwd azerty -la 0x3418000
0 -ep 0x801073D -t fsbl -of 0x00000001 –align -o /home/user/FSBL_N6_Signed.stm32
```

2.3 Standalone mode

When executing STM32-SignTool in standalone mode, an absolute path must be entered first. A password is then requested twice for confirmation, as shown in the figure below.

Figure 1. STM32-SignTool in standalone mode



DT72216V1

The 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 flag 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 STM32 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 that all public and private keys be provided as input files. These are directly accessible by any person who is allowed to execute the signing service. Ultimately, this can be considered to be a security leak. There are several solutions to protect keys against any attempts to steal key data. In 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 smartcards. 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 smartcards 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 based on ECDSA public/private keys. These keys 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 one index for header v1 and eight 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>`
- `--slot--identifier (-sid)`
  - Description: specify the identifier of the slot to use (optional, in decimal or hexadecimal format)
  - Syntax:`-sid <value>`
    - If the option `--slot-identifier` is used simultaneously with `--slot-index`, the tool checks if this configuration matches the same slot. The identifier reflects the index that was mentioned; otherwise, an error occurs.
    - It is possible to use `--slot-identifier` without mentioning `--slot-index`. The tool searches the slot index systematically.
- `--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
```

An error on the command line, or an inability of the tool to identify the key objects that match, causes an error message to be displayed. This indicates the source of the problem.

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 free software to set up a suitable environment. The keys can then be generated, and information about objects obtained.

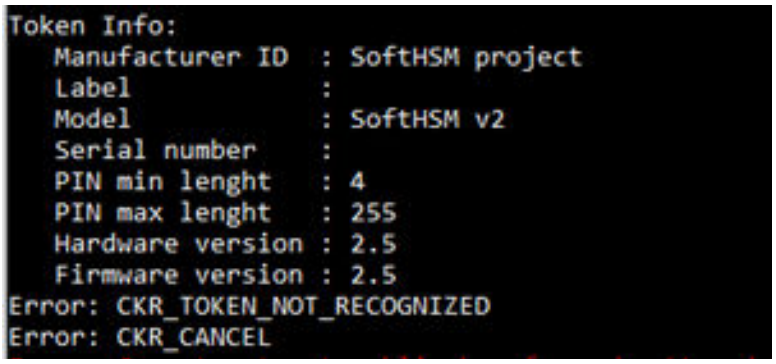
Slot identifier option:

-
```
-bin input.bin --type fsbl -hv 1 --key-index 0x40 -aki 0 --module softhsm2.dll --password prg-dev -ep 0x2ffe4000 -s -si 0 -sid 0x51a53ad8 -la 0x2ffc2500 -iv 0 -of 0x80000000 -o output.stm32
```

Error examples:

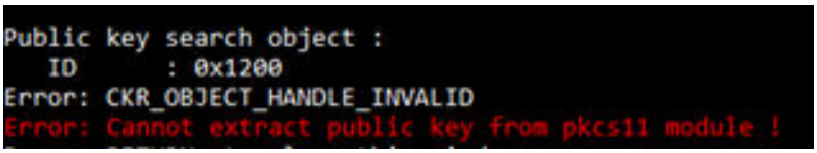
- Invalid slot index

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 the objects sequentially. If it cannot identify the matching key objects on the first try, the signing operation stops the process. An error message is then displayed to indicate the source of the problem.

Revision history

Table 2. **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 STM32-SignTool • Added Section 2.4: PKCS#11 solution
27-Jun-2022	3	Updated Section 2.1: Commands
26-Jun-2024	4	Replaced in the whole document: • STM32MP1 series by STM32MPx series • STM32MP1-SignTool by STM32MP-SignTool • STM32MP1-KeyGen by STM32MP-KeyGen Updated --public-key -pubk and added --header-version (-hv) and --no-keys (-nk) in Section 2.1: Commands. Added "Example 6" in Section 2.2: Examples for STM32-SignTool.
14-Nov-2024	5	Added: • STM32N6 series to applicable products Replaced in the whole document: • STM32MP by STM32 Updated: • Section 2.1: Commands
06-Mar-2025	6	Updated: • Section 2.4.1: Additional PKCS#11 commands • Section 2.4.3: Examples
04-Nov-2025	7	Updated: • Section 2.1: Commands • Section 2.4.3: Examples

Contents

IMPORTANT NOTICE – 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.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The 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.

The 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 the 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.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, 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.