



STM32MP15 secure boot



Contents

1. STM32MP15 secure boot	74
2. BSEC internal peripheral	10
3. Boot chain overview	17
4. Category:ROM code	24
5. KeyGen tool	32
6. NVMEM overview	39
7. STM32CubeProgrammer	46
8. STM32CubeProgrammer release note	53
9. STM32MP15 ROM code overview	60
10. STM32MP15 microprocessor	67
11. Signing tool	81
12. TF-A overview	88
13. U-Boot overview	95



A quality version of this page, approved on 5 January 2021, was based off this revision.

Contents

1 Purpose	75
2 Authentication processing	76
2.1 Key generation	76
2.2 Key registration	76
2.2.1 Register hash public key	77
2.3 Image signing	77
2.3.1 STM32 Header	77
2.4 Image programming	79
2.5 PKH check	79
2.6 Authentication	80
2.6.1 Bootrom authentication	80
2.6.2 TF-A authentication	80
2.7 Closing the device	80



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

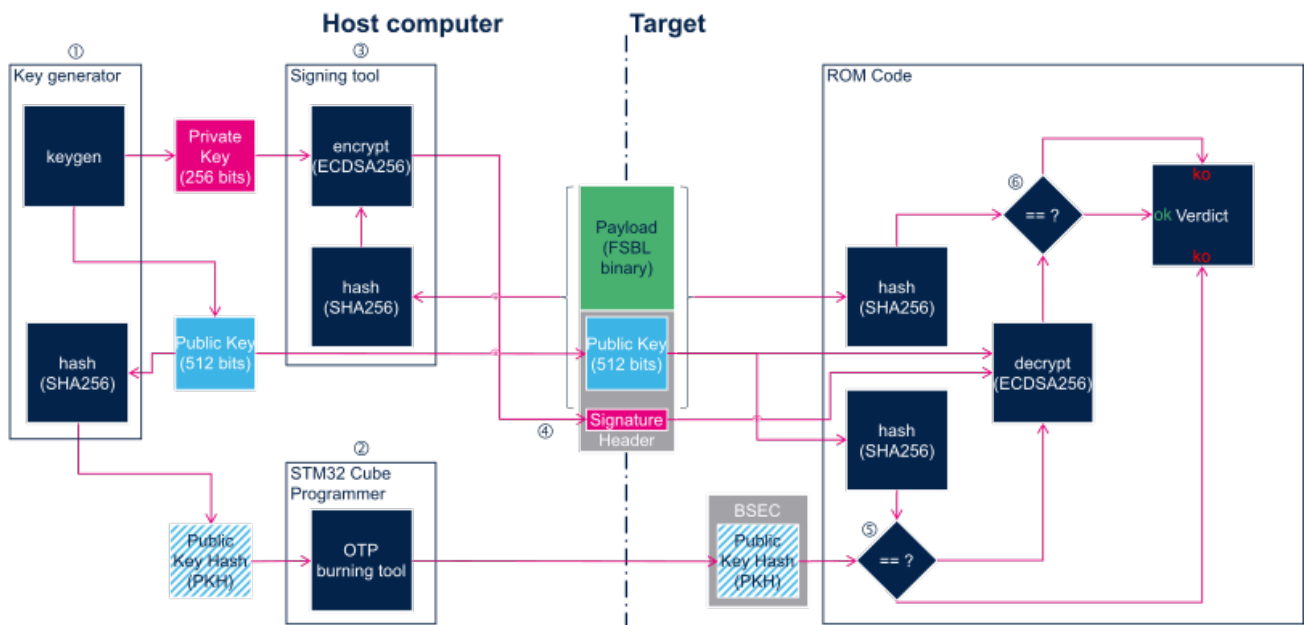
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the OTP WORD 24 to 31 in BSEC with the corresponding public key hash (PKH, output file from STM32 KeyGen). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use NVMEM framework.

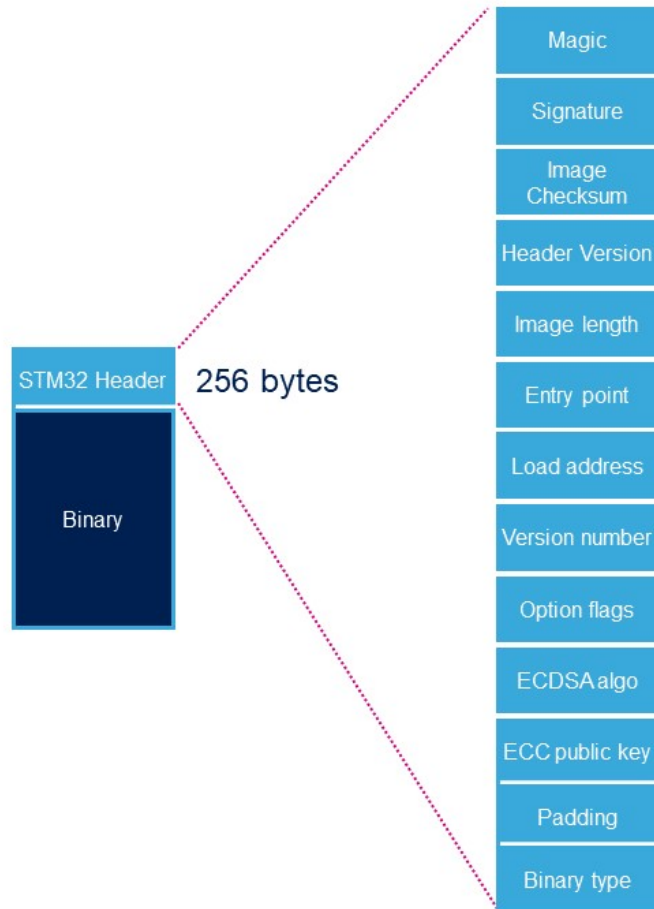
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. STM32 Signing tool allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit
Stable: 24.09.2019 - 14:06 / Revision: 24.09.2019 - 07:57

Contents

1 Purpose	11
2 Authentication processing	12
2.1 Key generation	12
2.2 Key registration	12
2.2.1 Register hash public key	13
2.3 Image signing	13
2.3.1 STM32 Header	13
2.4 Image programming	15
2.5 PKH check	15
2.6 Authentication	16
2.6.1 Bootrom authentication	16
2.6.2 TF-A authentication	16
2.7 Closing the device	16



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

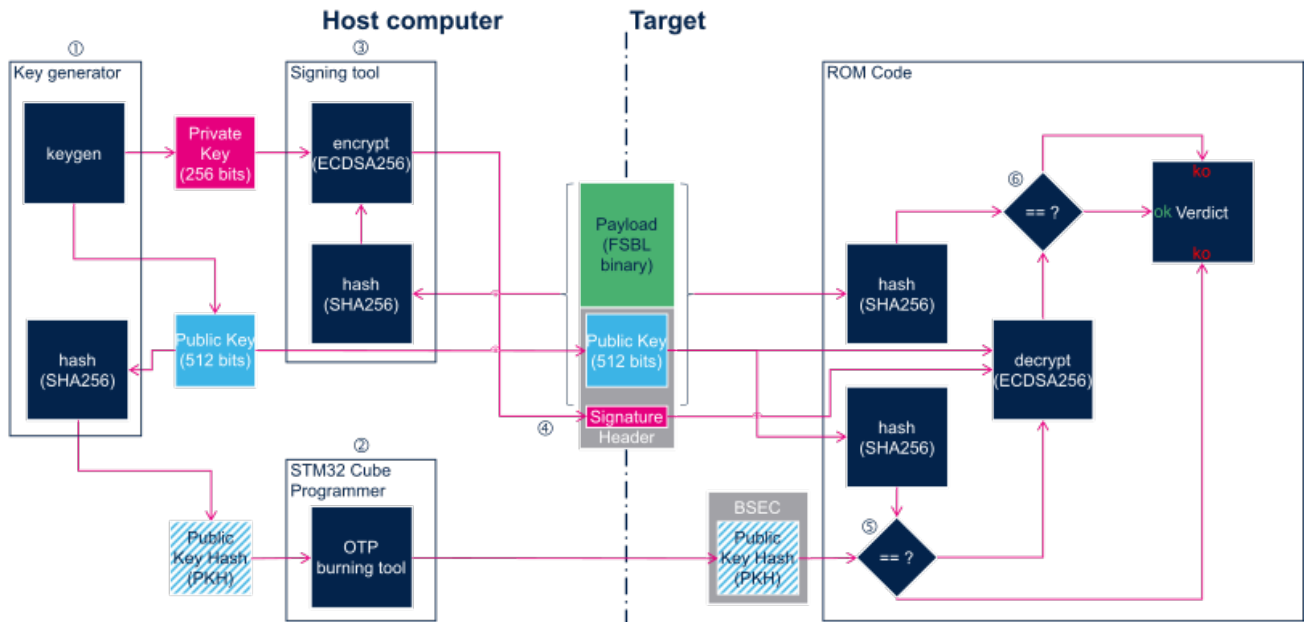
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

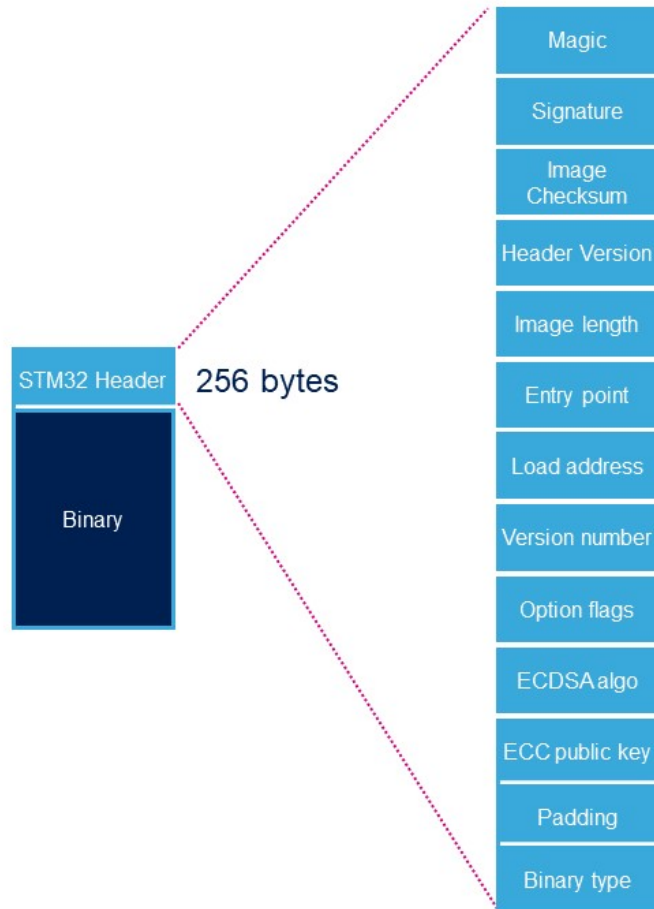
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 10.11.2020 - 07:59 / Revision: 10.11.2020 - 07:58

Contents

1 Purpose	18
2 Authentication processing	19
2.1 Key generation	19
2.2 Key registration	19
2.2.1 Register hash public key	20
2.3 Image signing	20
2.3.1 STM32 Header	20
2.4 Image programming	22
2.5 PKH check	22
2.6 Authentication	23
2.6.1 Bootrom authentication	23
2.6.2 TF-A authentication	23
2.7 Closing the device	23



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

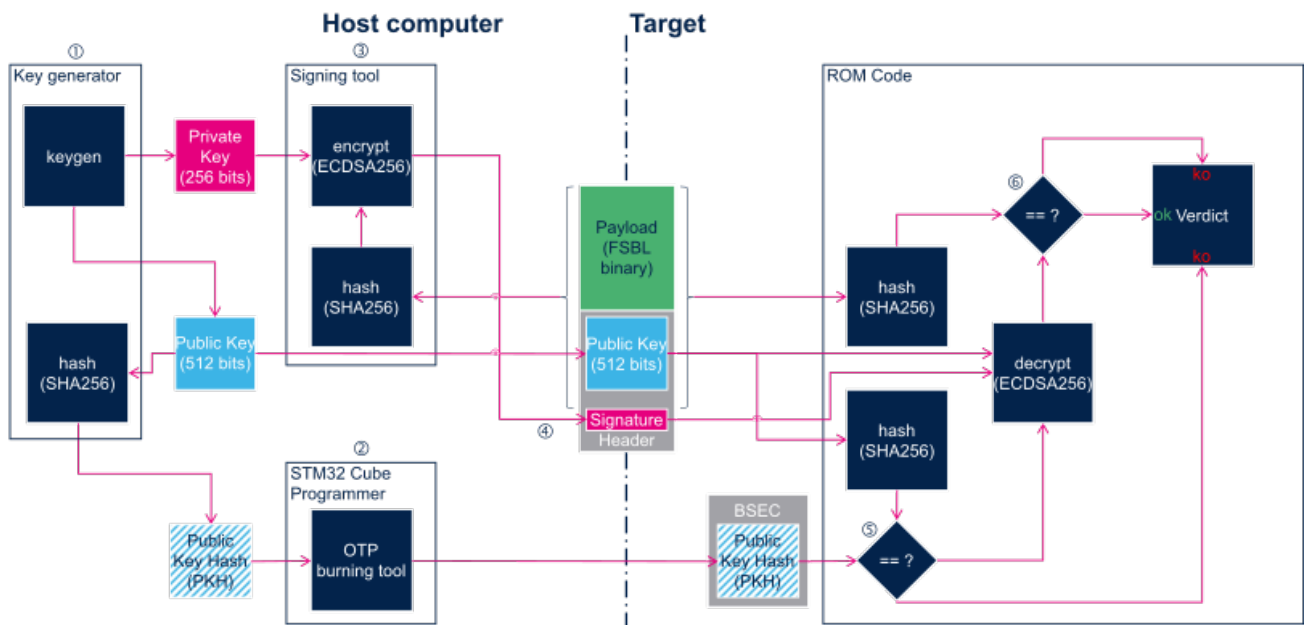
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the OTP WORD 24 to 31 in BSEC with the corresponding public key hash (PKH, output file from STM32 KeyGen). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use NVMEM framework.

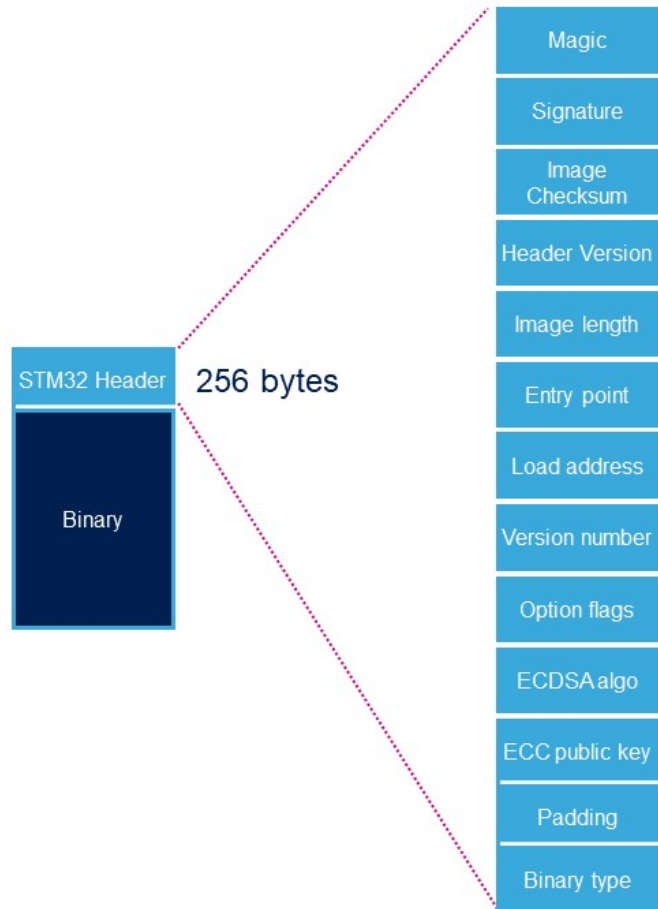
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. STM32 Signing tool allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the STM32CubeProgrammer or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 17.06.2020 - 18:26 / Revision: 16.01.2020 - 09:28

Contents

1 Purpose	25
2 Authentication processing	26
2.1 Key generation	26
2.2 Key registration	26
2.2.1 Register hash public key	27
2.3 Image signing	27
2.3.1 STM32 Header	27
2.4 Image programming	29
2.5 PKH check	29
2.6 Authentication	30
2.6.1 Bootrom authentication	30
2.6.2 TF-A authentication	30
2.7 Closing the device	30



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

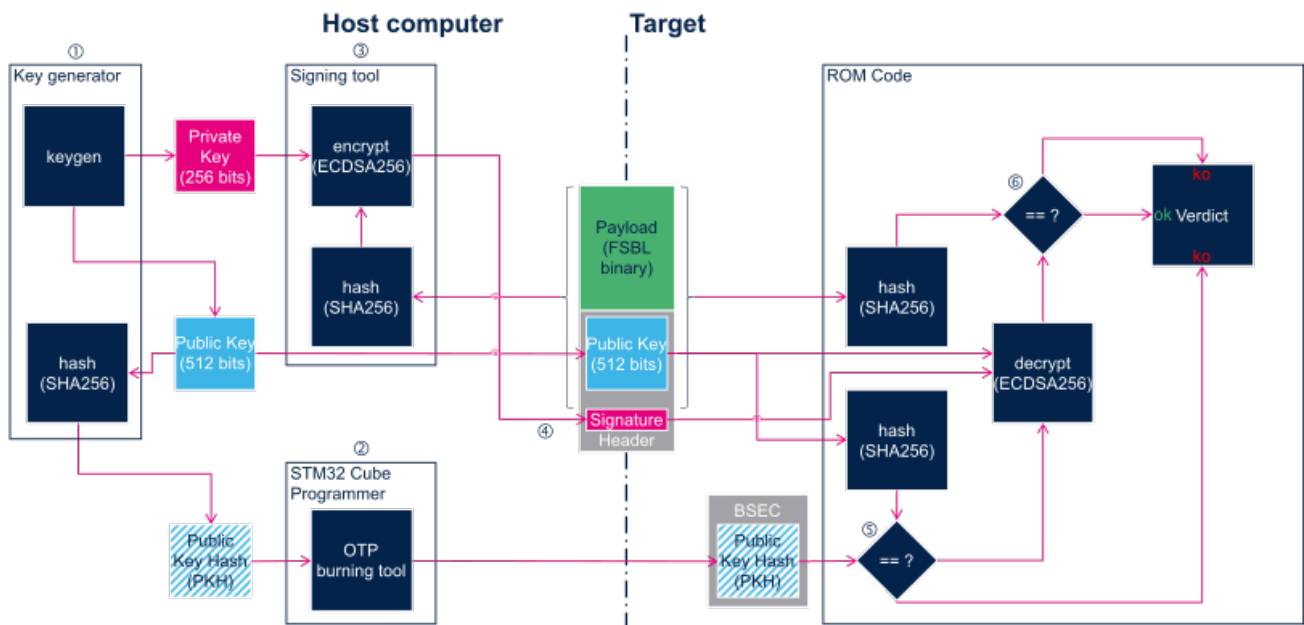
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

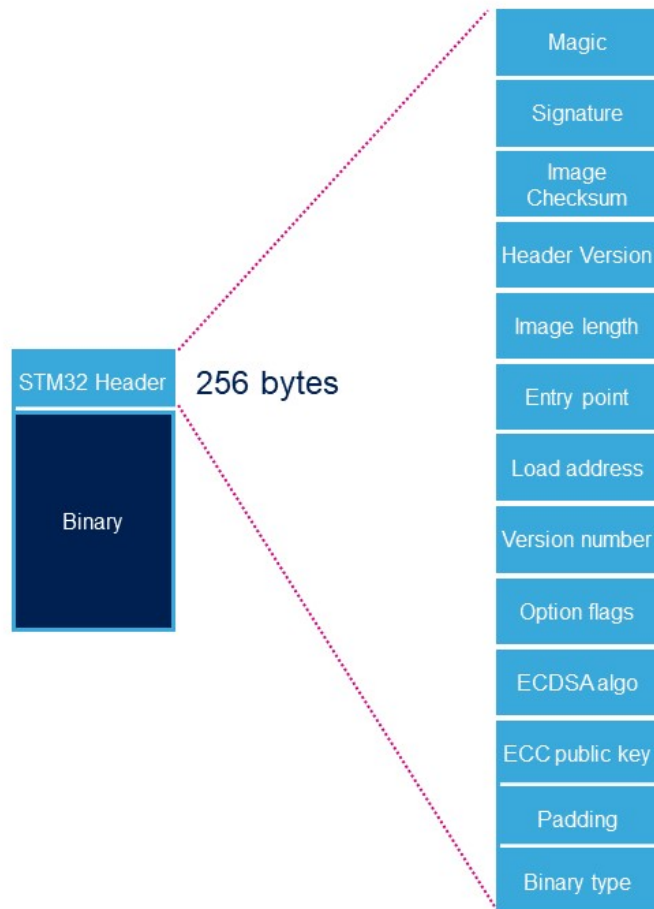
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the STM32CubeProgrammer or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability

One Time Programmed

Doubledata rate (memory domain)

First Stage Boot Loader

Second Stage Boot Loader

Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))

Trusted Firmware for Arm Cortex-A

Read Only Memory

Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)

Secure Hash Algorithm

Light-emitting diode

Open Portable Trusted Execution Environment

Central processing unit



Pages in category "ROM code"

The following 4 pages are in this category, out of 4 total.

- [STM32 header for binary files](#)
- [STM32MP15 ROM code overview](#)
- [STM32MP15 ROM trace analyzer](#)
- [STM32MP15 secure boot](#)

Stable: 07.01.2021 - 11:34 / Revision: 20.11.2020 - 17:08

Contents

1 Purpose	33
2 Authentication processing	34
2.1 Key generation	34
2.2 Key registration	34
2.2.1 Register hash public key	35
2.3 Image signing	35
2.3.1 STM32 Header	35
2.4 Image programming	37
2.5 PKH check	37
2.6 Authentication	38
2.6.1 Bootrom authentication	38
2.6.2 TF-A authentication	38
2.7 Closing the device	38



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

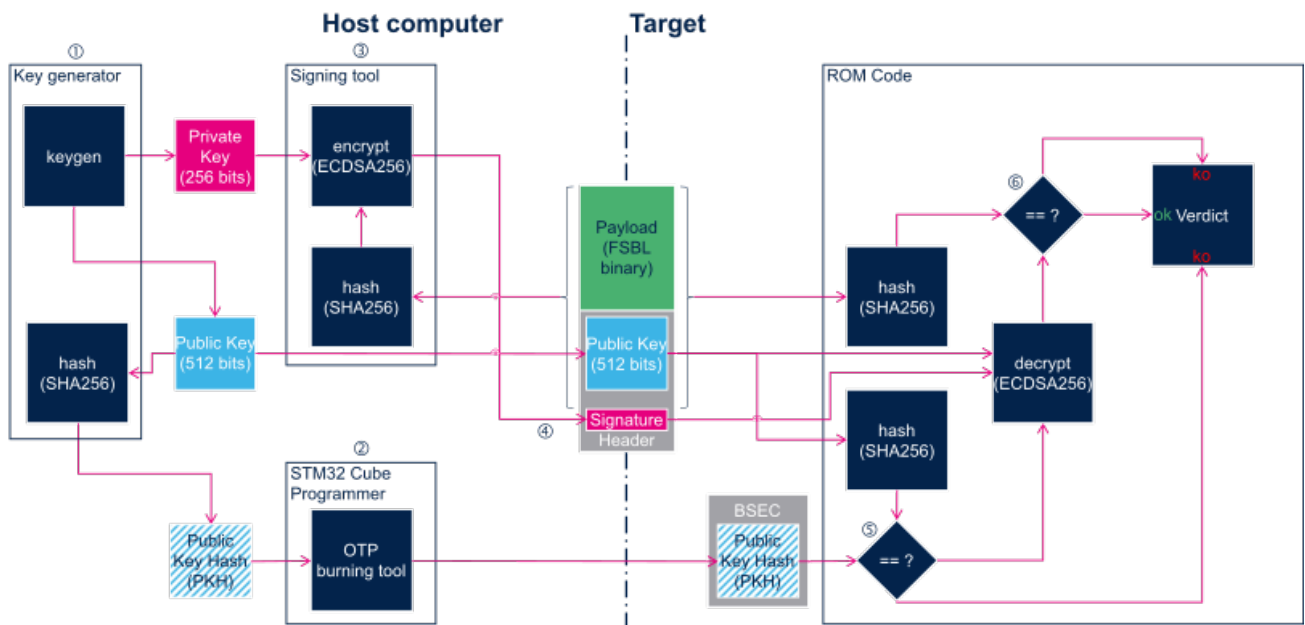
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

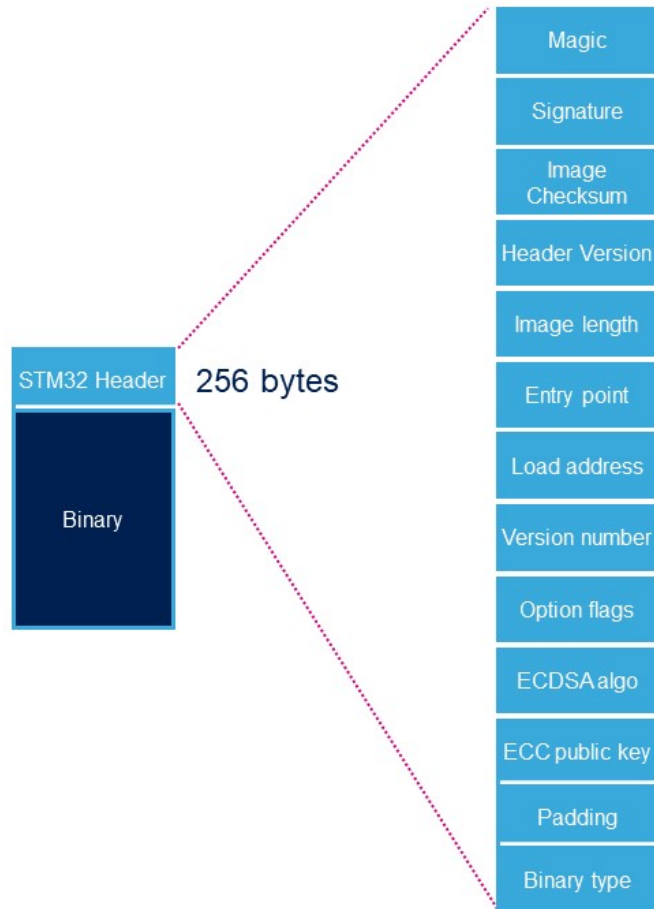
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 04.11.2020 - 14:14 / Revision: 04.11.2020 - 14:04

Contents

1 Purpose	40
2 Authentication processing	41
2.1 Key generation	41
2.2 Key registration	41
2.2.1 Register hash public key	42
2.3 Image signing	42
2.3.1 STM32 Header	42
2.4 Image programming	44
2.5 PKH check	44
2.6 Authentication	45
2.6.1 Bootrom authentication	45
2.6.2 TF-A authentication	45
2.7 Closing the device	45



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

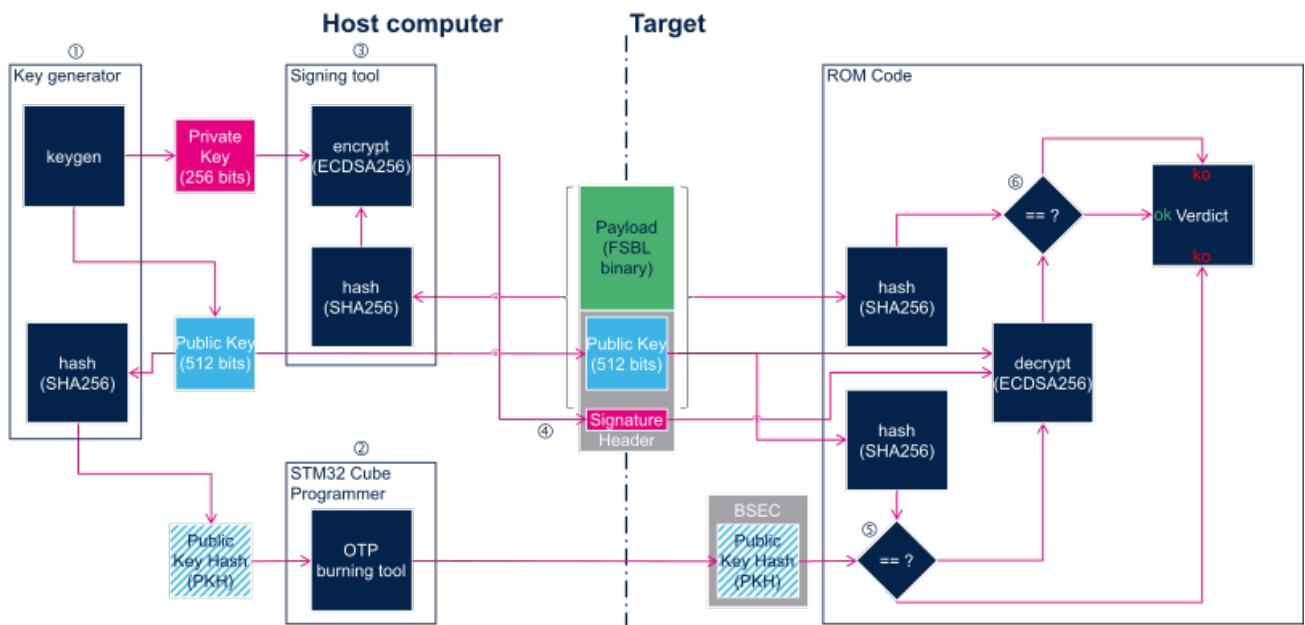
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the **OTP WORD 24 to 31** in BSEC with the corresponding public key hash (PKH, output file from **STM32 KeyGen**). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use **NVMEM** framework.

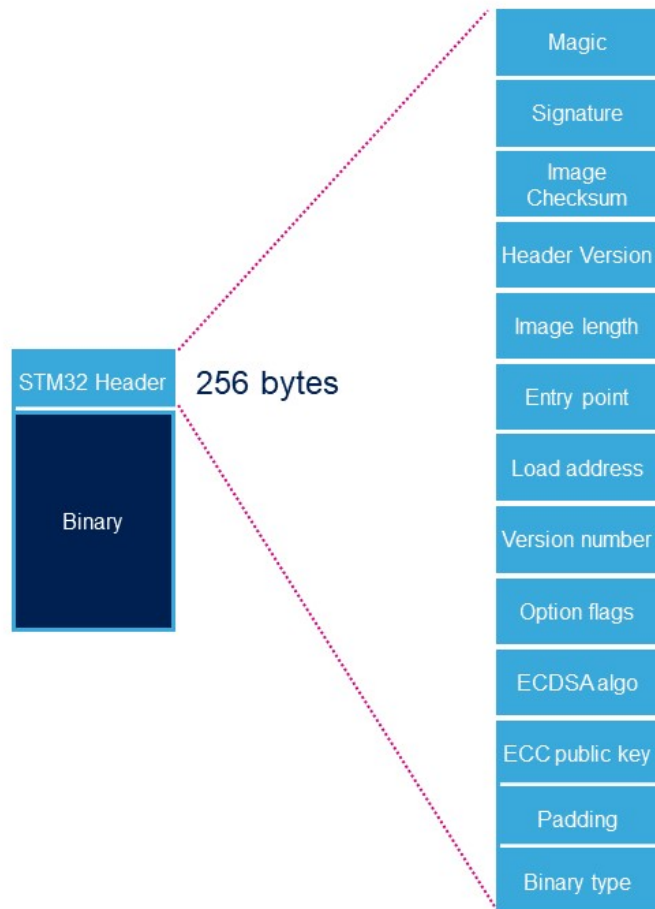
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. **STM32 Signing tool** allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 23.03.2021 - 10:30 / Revision: 23.03.2021 - 10:09

Contents

1 Purpose	47
2 Authentication processing	48
2.1 Key generation	48
2.2 Key registration	48
2.2.1 Register hash public key	49
2.3 Image signing	49
2.3.1 STM32 Header	49
2.4 Image programming	51
2.5 PKH check	51
2.6 Authentication	52
2.6.1 Bootrom authentication	52
2.6.2 TF-A authentication	52
2.7 Closing the device	52



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

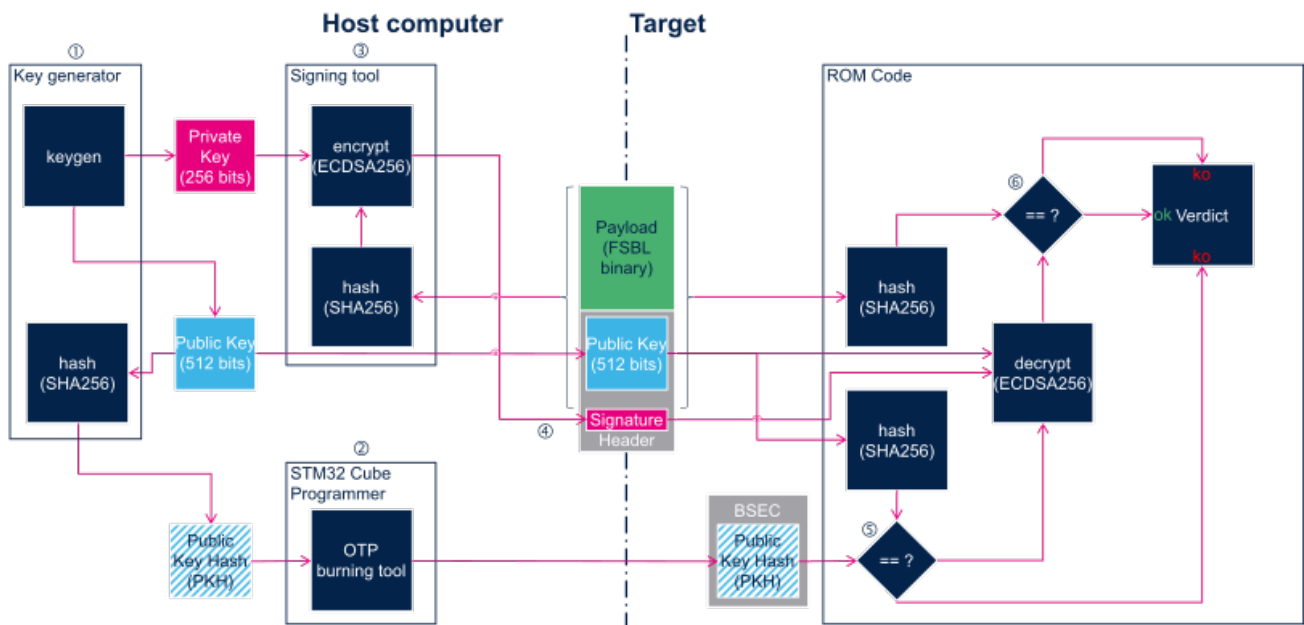
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration



Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

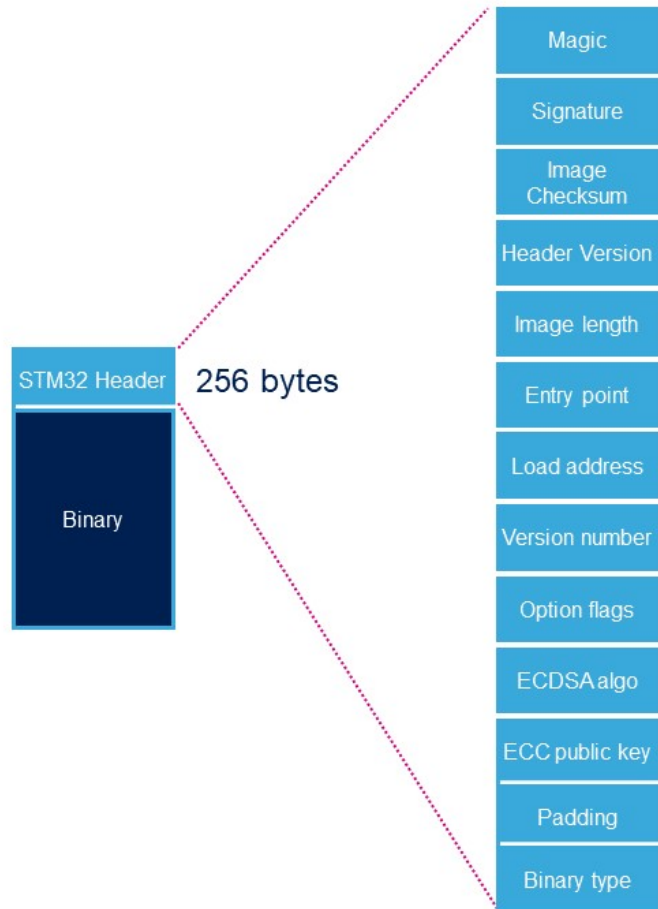
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 23.03.2021 - 10:30 / Revision: 23.03.2021 - 09:49

Contents

1 Purpose	54
2 Authentication processing	55
2.1 Key generation	55
2.2 Key registration	55
2.2.1 Register hash public key	56
2.3 Image signing	56
2.3.1 STM32 Header	56
2.4 Image programming	58
2.5 PKH check	58
2.6 Authentication	59
2.6.1 Bootrom authentication	59
2.6.2 TF-A authentication	59
2.7 Closing the device	59



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

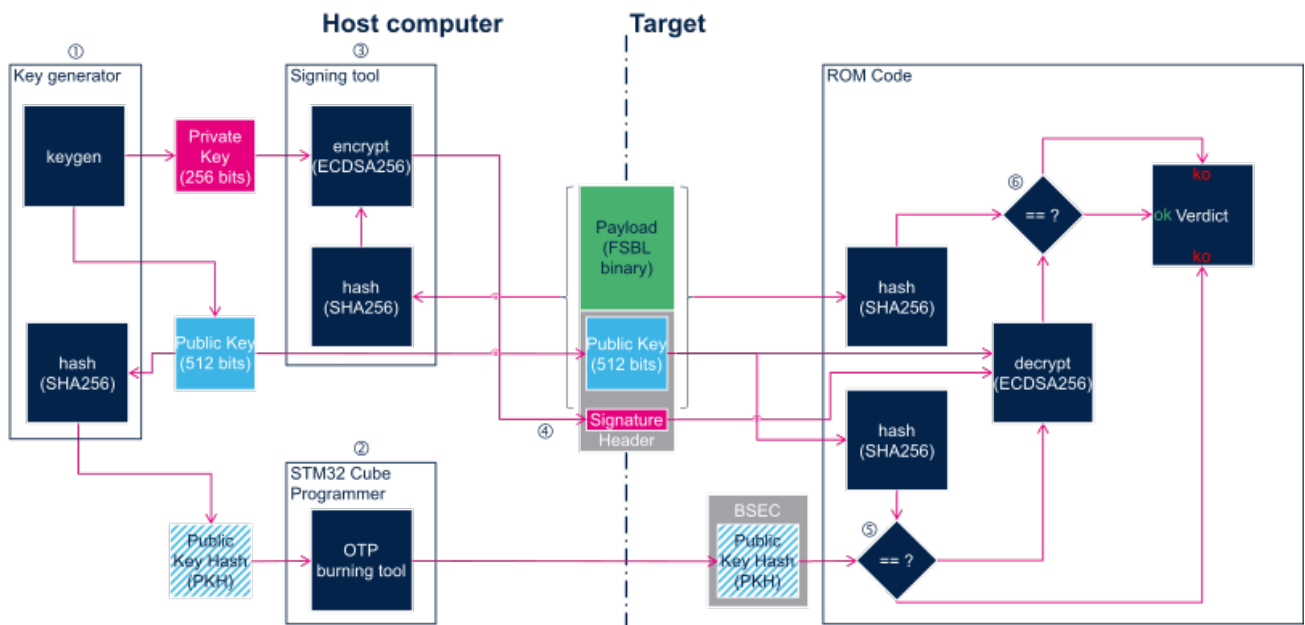
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

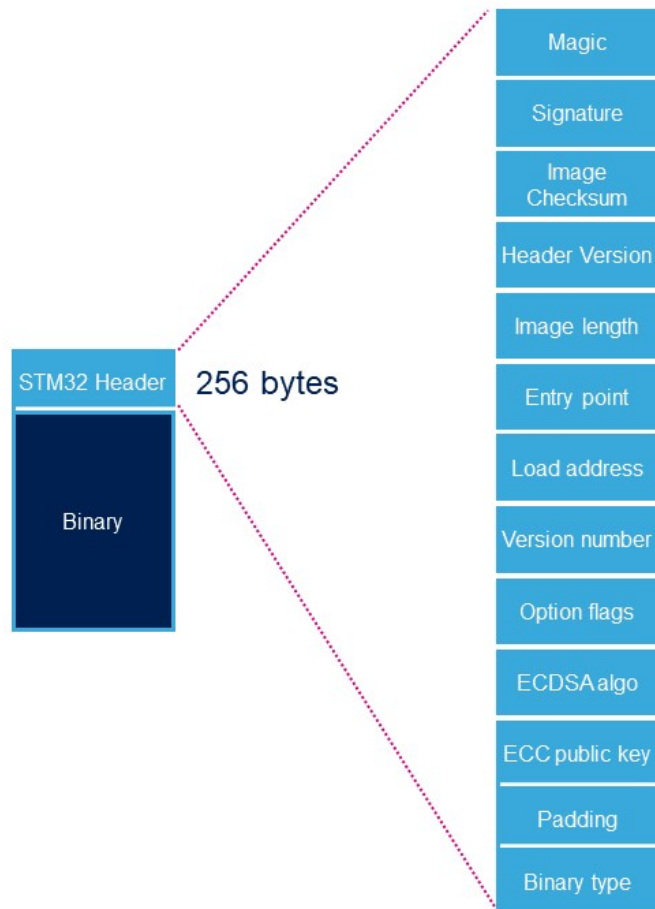
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisionned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 16.04.2021 - 12:27 / Revision: 23.02.2021 - 14:19

Contents

1 Purpose	61
2 Authentication processing	62
2.1 Key generation	62
2.2 Key registration	62
2.2.1 Register hash public key	63
2.3 Image signing	63
2.3.1 STM32 Header	63
2.4 Image programming	65
2.5 PKH check	65
2.6 Authentication	66
2.6.1 Bootrom authentication	66
2.6.2 TF-A authentication	66
2.7 Closing the device	66



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

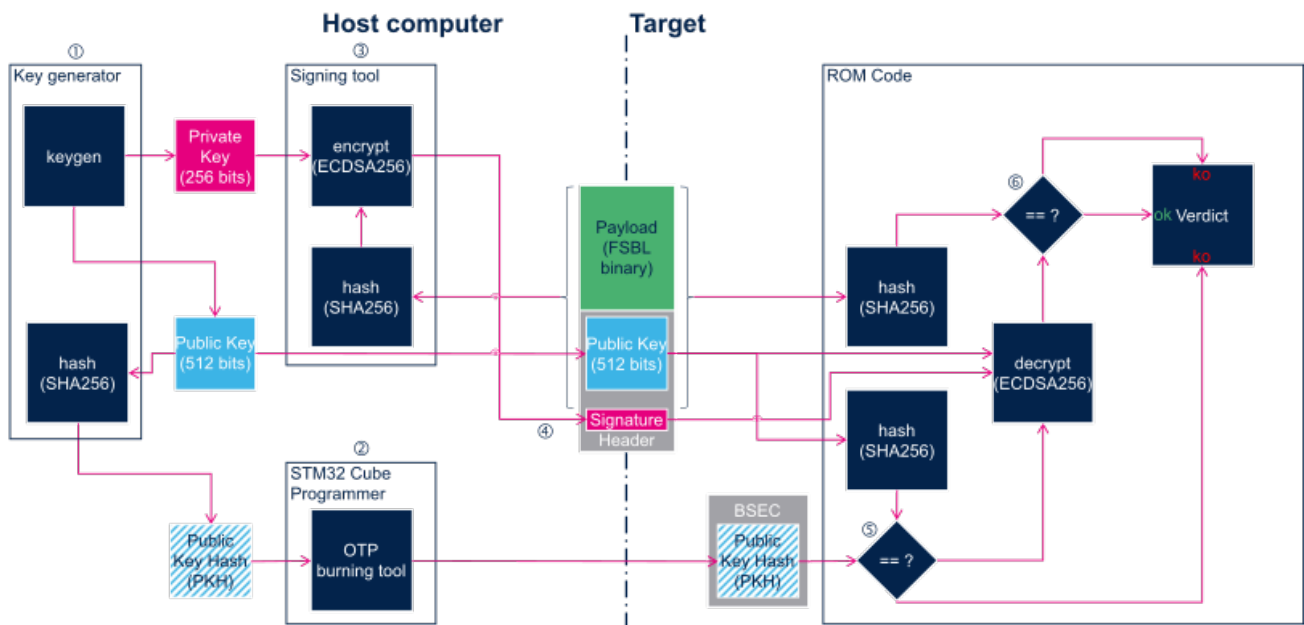
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the OTP WORD 24 to 31 in BSEC with the corresponding public key hash (PKH, output file from STM32 KeyGen). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use NVMEM framework.

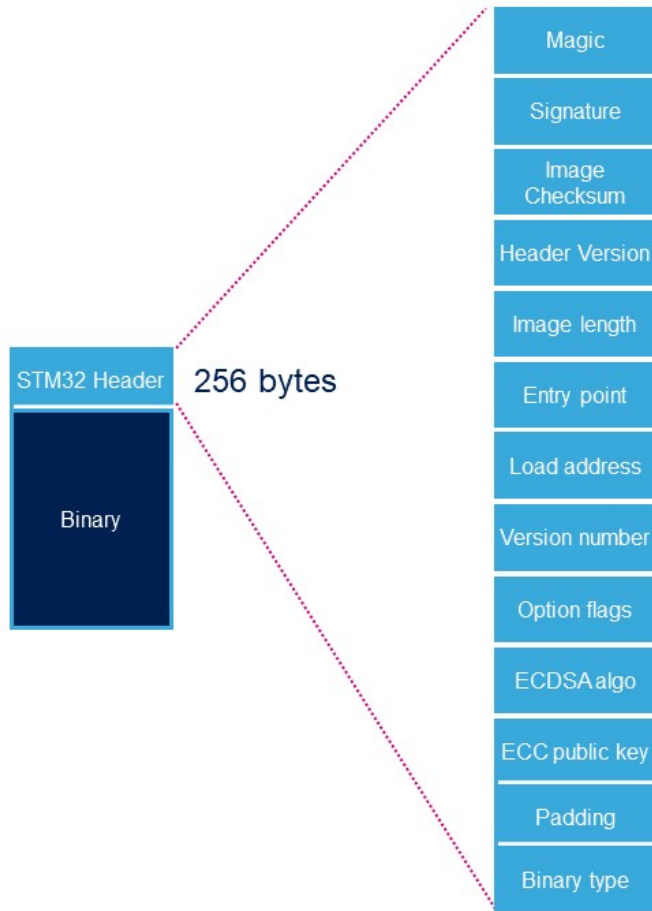
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. STM32 Signing tool allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the STM32CubeProgrammer or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 05.03.2021 - 09:42 / Revision: 05.03.2021 - 09:42

Contents

1 Purpose	68
2 Authentication processing	69
2.1 Key generation	69
2.2 Key registration	69
2.2.1 Register hash public key	70
2.3 Image signing	70
2.3.1 STM32 Header	70
2.4 Image programming	72
2.5 PKH check	72
2.6 Authentication	73
2.6.1 Bootrom authentication	73
2.6.2 TF-A authentication	73
2.7 Closing the device	73



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

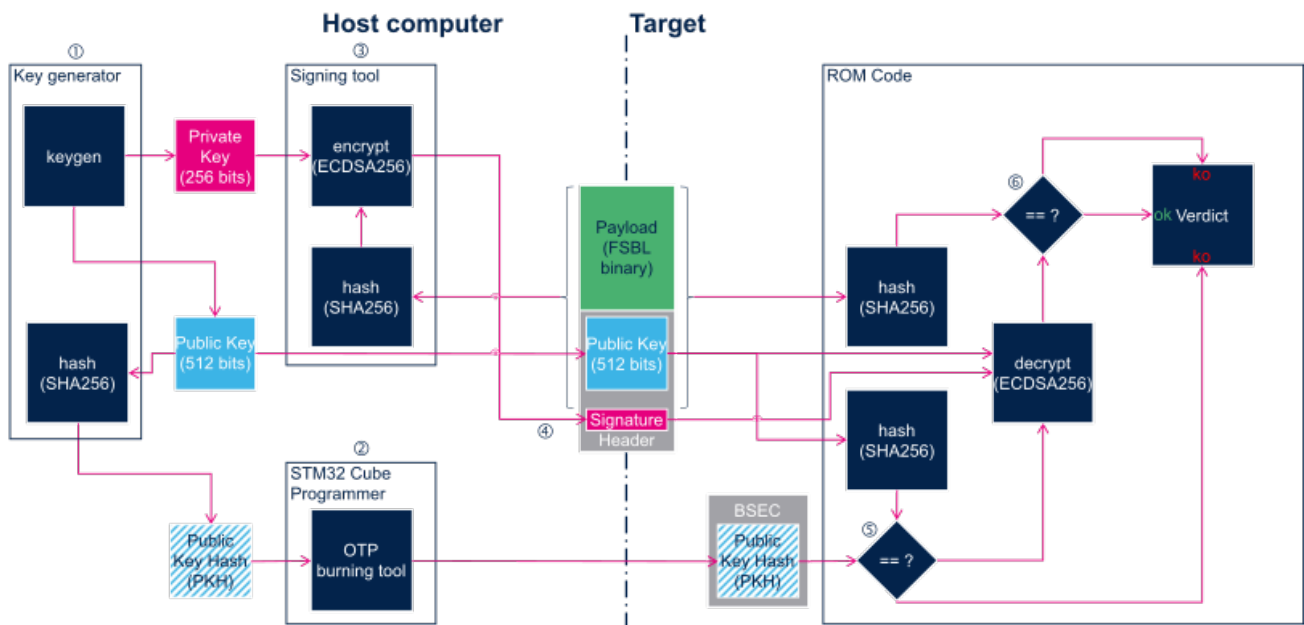
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the OTP WORD 24 to 31 in BSEC with the corresponding public key hash (PKH, output file from STM32 KeyGen). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use NVMEM framework.

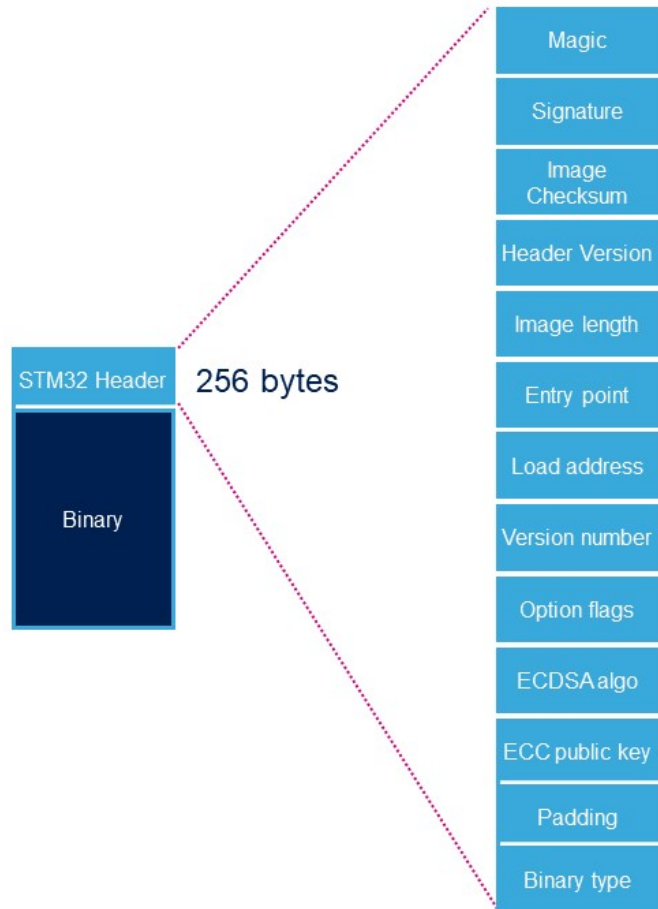
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. STM32 Signing tool allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit
Stable: 09.01.2021 - 17:10 / Revision: 20.11.2020 - 17:10

A quality version of this page, approved on *5 January 2021*, was based off this revision.

Contents

1 Purpose	75
2 Authentication processing	76
2.1 Key generation	76
2.2 Key registration	76
2.2.1 Register hash public key	77
2.3 Image signing	77
2.3.1 STM32 Header	77
2.4 Image programming	79
2.5 PKH check	79
2.6 Authentication	80
2.6.1 Bootrom authentication	80
2.6.2 TF-A authentication	80
2.7 Closing the device	80



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

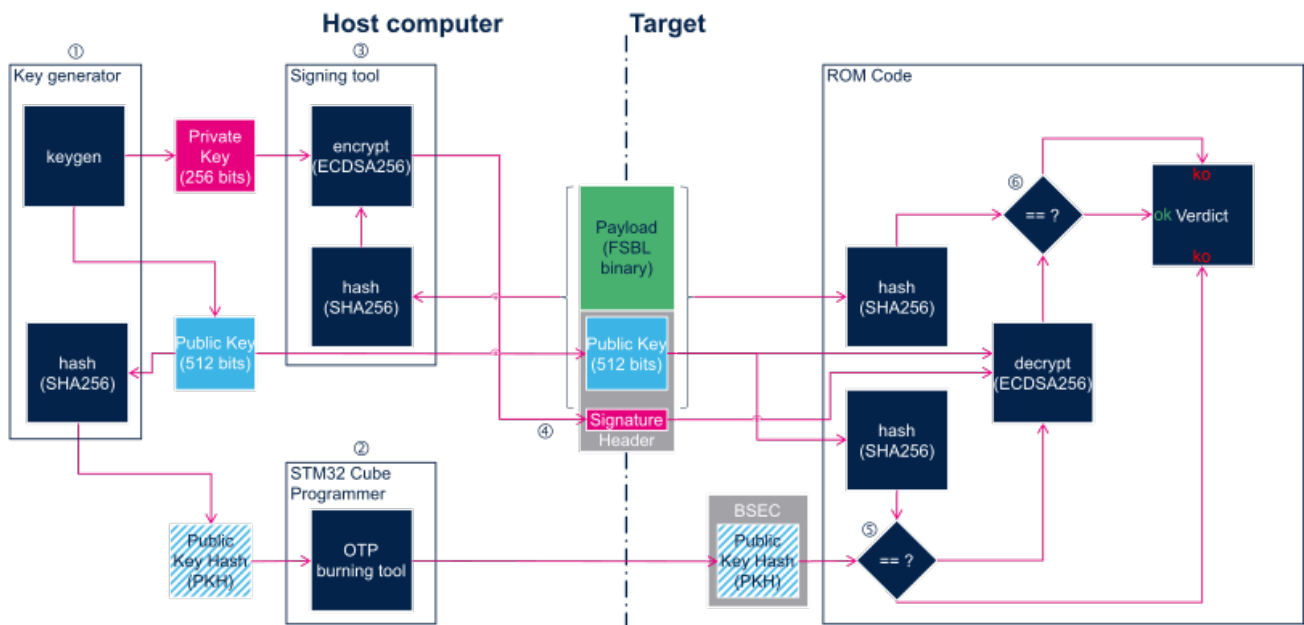
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the OTP WORD 24 to 31 in BSEC with the corresponding public key hash (PKH, output file from STM32 KeyGen). OpenSTLinux embeds a **stm32key** tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use NVMEM framework.

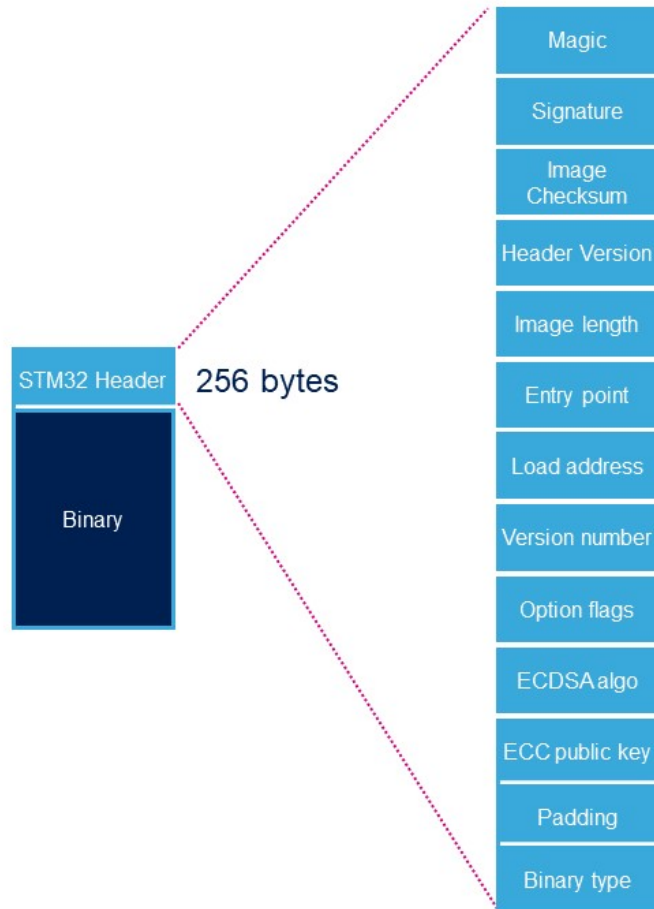
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. STM32 Signing tool allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the STM32CubeProgrammer or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit
Stable: 07.01.2021 - 14:13 / Revision: 20.11.2020 - 17:10

Contents

1 Purpose	82
2 Authentication processing	83
2.1 Key generation	83
2.2 Key registration	83
2.2.1 Register hash public key	84
2.3 Image signing	84
2.3.1 STM32 Header	84
2.4 Image programming	86
2.5 PKH check	86
2.6 Authentication	87
2.6.1 Bootrom authentication	87
2.6.2 TF-A authentication	87
2.7 Closing the device	87



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

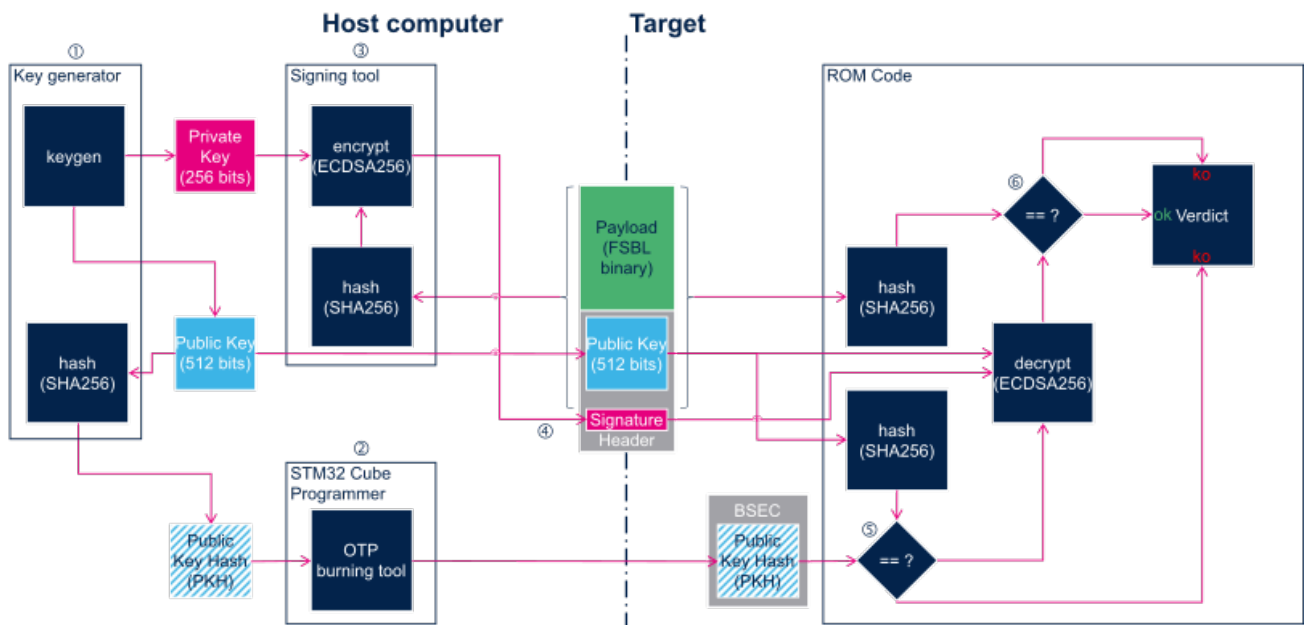
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

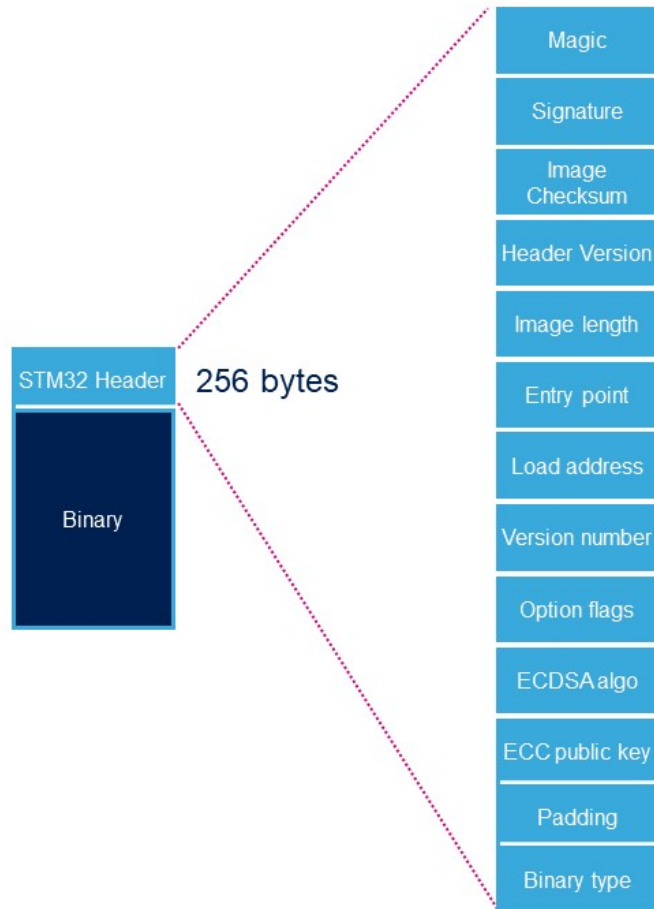
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the STM32CubeProgrammer or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 17.02.2021 - 19:40 / Revision: 16.02.2021 - 16:25

Contents

1 Purpose	89
2 Authentication processing	90
2.1 Key generation	90
2.2 Key registration	90
2.2.1 Register hash public key	91
2.3 Image signing	91
2.3.1 STM32 Header	91
2.4 Image programming	93
2.5 PKH check	93
2.6 Authentication	94
2.6.1 Bootrom authentication	94
2.6.2 TF-A authentication	94
2.7 Closing the device	94



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

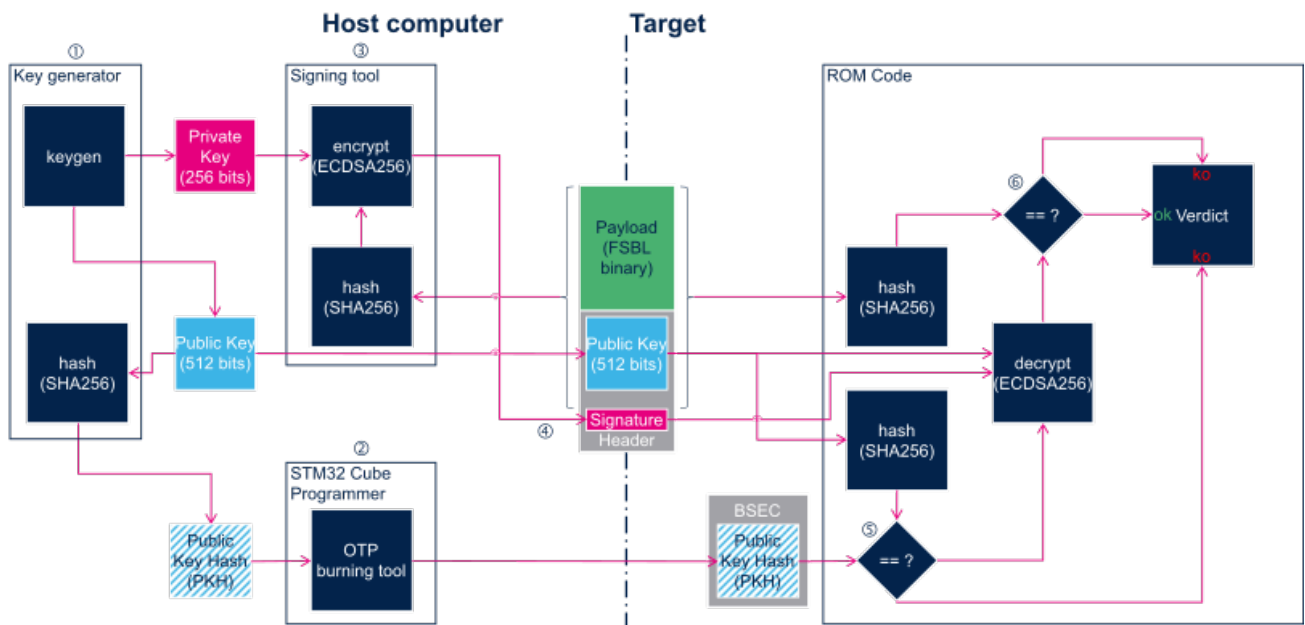
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

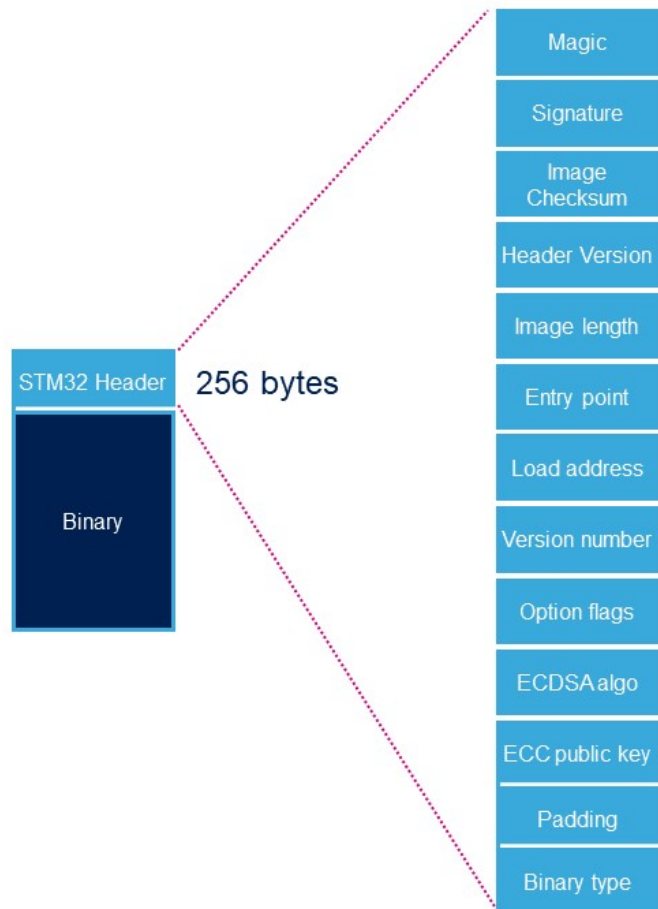
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability
 One Time Programmed
 Doubledata rate (memory domain)
 First Stage Boot Loader
 Second Stage Boot Loader
 Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))
 Trusted Firmware for Arm Cortex-A
 Read Only Memory
 Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)
 Secure Hash Algorithm
 Light-emitting diode
 Open Portable Trusted Execution Environment
 Central processing unit

Stable: 01.03.2021 - 10:54 / Revision: 01.03.2021 - 10:53

Contents

1 Purpose	96
2 Authentication processing	97
2.1 Key generation	97
2.2 Key registration	97
2.2.1 Register hash public key	98
2.3 Image signing	98
2.3.1 STM32 Header	98
2.4 Image programming	100
2.5 PKH check	100
2.6 Authentication	101
2.6.1 Bootrom authentication	101
2.6.2 TF-A authentication	101
2.7 Closing the device	101



1 Purpose

Secure boot is a key feature to guarantee a secure platform.

STM32MP1 boot sequence supports a trusted boot chain that ensures that the loaded images are authenticated and checked in integrity before being used.

Warning

The secure boot feature availability is indicated in the *security* field of the chip part number.



2 Authentication processing

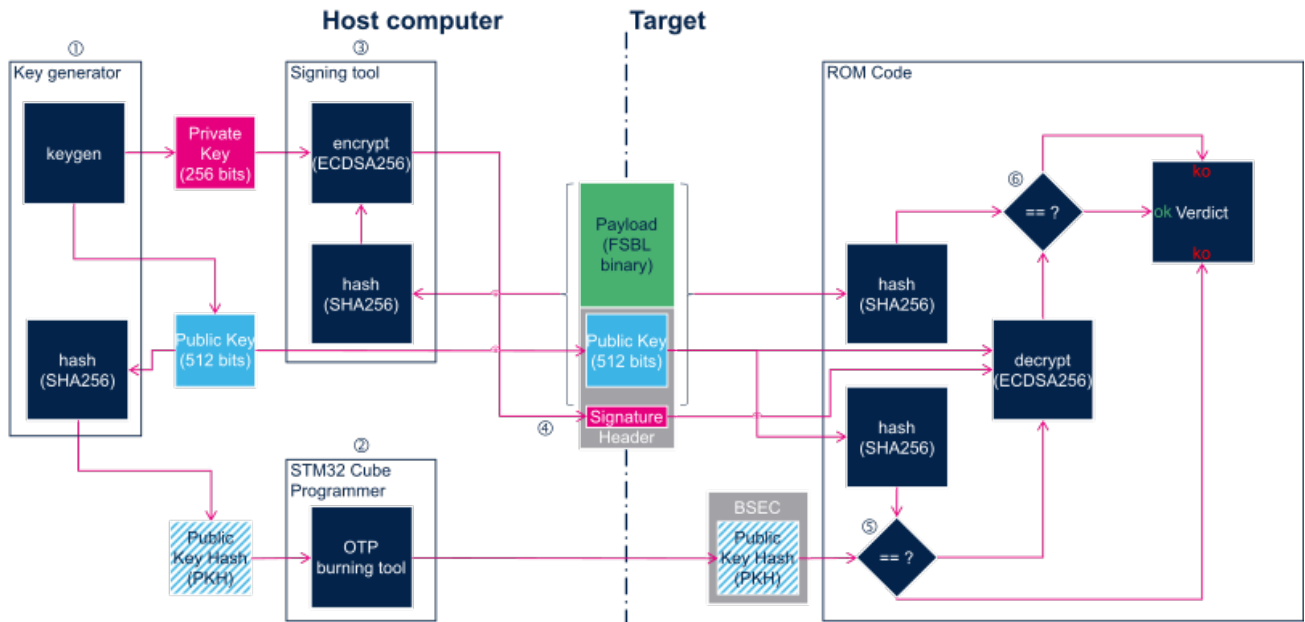
STM32 MPU provides authentication processing with ECDSA ^[1] verification algorithm, based on ECC ^[2]. ECDSA offers better result than RSA with a smaller key. STM32 MPU relies on a 256 bits ECDSA key.

Two algorithms are supported for ECDSA calculation:

- P-256 NIST
- Brainpool 256

The algorithm selection is done via the signed binary header, as shown in *STM32 header* (subchapter in this same article).

The ECDSA verification follows the process below:



2.1 Key generation

First step is to generate the ECC pair of keys with STM32 KeyGen tool. This is the key pair that will be used to sign the images. The tool also generates a third file containing the public key hash (PKH) that will be used to authenticate the public key on the target.

2.2 Key registration

⚠ Warning

Make sure that a device with Secure boot enabled is used: this is mentioned in the [part number](#), otherwise the device will become permanently unusable.



2.2.1 Register hash public key

First step to enable the authentication is to burn the `OTP WORD 24 to 31` in BSEC with the corresponding public key hash (PKH, output file from `STM32 KeyGen`). OpenSTLinux embeds a `stm32key` tool that can be called from U-Boot command line interface to program the PKH into the OTP.

Make sure that a trusted image was programmed on your board, because below operation will not be possible with optee boot.

PKH file (publicKeyhash.bin) must be available in a filesystem partition (like bootfs) on a storage device (like sdcard) before proceeding.

```
Board $> ext4load mmc 0:4 0xc0000000 publicKeyhash.bin
from mmc 0 partition 4 (ext4) in DDR
32 bytes read in 50 ms (0 Bytes/s)
```

Load hash file

```
Board $> stm32key read 0xc0000000
from DDR to confirm it is valid (without writing it in OTP)
OTP value 24: 12345678
OTP value 25: 12345678
OTP value 26: 12345678
OTP value 27: 12345678
OTP value 28: 12345678
OTP value 29: 12345678
OTP value 30: 12345678
OTP value 31: 12345678
```

Read loaded key

Warning

If hash key is ok, the key in OTP can be fused

```
Board $> stm32key fuse -y 0xc0000000
OTP
```

Write the key in

The device now contains the hash to authenticate images. To read back the OTP, you can use `NVMEM` framework.

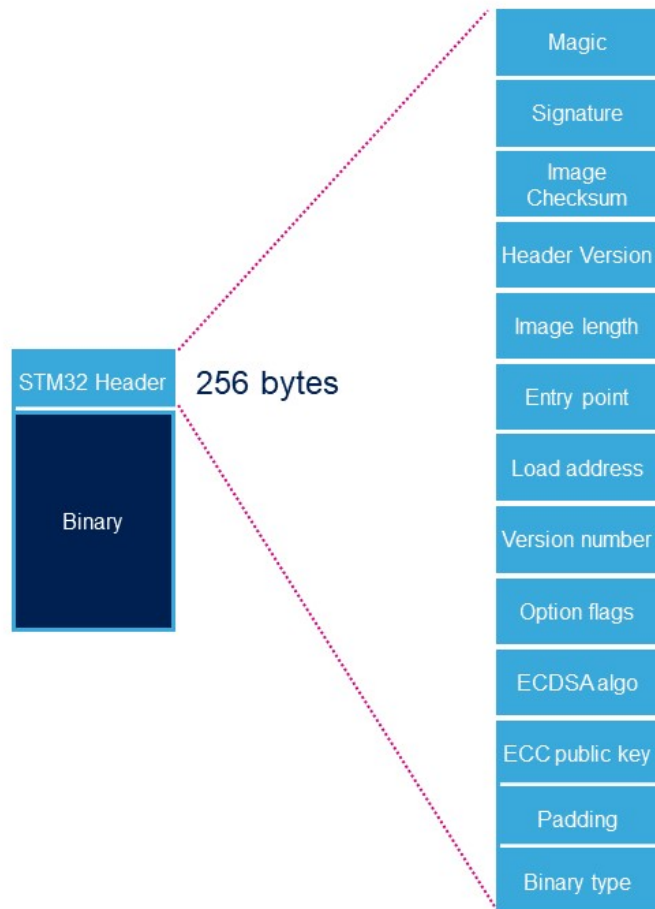
2.3 Image signing

In a second step, FSBL and SSBL binaries must be signed. `STM32 Signing tool` allows to fill the STM32 binary header that is parsed by the embedded software to authenticate each binary.

2.3.1 STM32 Header

Each binary image (signed or not) loaded by ROM code and by TF-A need to include a specific STM32 header added on top of the binary data.

The header includes the authentication information.



Name	Length	Byte Offset	Description
Magic number	32 bits	0	4 bytes in big endian: 'S', 'T', 'M', 0x32 = 0x53544D32
Image signature	512 bits	4	ECDSA signature for image authentication ^[Note 1]
Image checksum	32 bits	68	Checksum of the payload ^[Note 2]
Header version	32 bits	72	Header version v1.0 = 0x00010000 Byte0: reserved Byte1: major version = 0x01 Byte2: minor version = 0x00 Byte3: reserved
Image length	32 bits	76	Length of image in bytes ^[Note 3]
Image entry Point	32 bits	80	Entry point of image
Reserved1	32 bits	84	Reserved
Load address	32 bits	88	Load address of image ^[Note 4]
Reserved2	32 bits	92	Reserved



Name	Length	Byte Offset	Description
Version number	32 bits	96	Image Version (monotonic number) ^[Note 5]
Option flags	32 bits	100	b0=1: no signature verification ^[Note 6]
ECDSA algorithm	32 bits	104	1: P-256 NIST ; 2: brainpool 256
ECDSA public key	512 bits	108	ECDSA public key to be used to verify the signature. ^[Note 7]
Padding	83 Bytes	172	Reserved padding bytes ^[Note 8] . Must all be set to 0
Binary type	1 Byte	255	Used to check the binary type 0x00: U-Boot 0x10-0x1F: TF-A 0x20-0x2F: OPTEE 0x30: Copro

- Signature is calculated from first byte of header version field to last byte of image given by image length field.
- 32-bit sum of all payload bytes accessed as 8-bit unsigned numbers, discarding any overflow bits. Used to check the downloaded image integrity when signature is not used (if b0=1 in Option flags).
- Length is the length of the built image, it does not include the length of the STM32 header.
- This field is not used by ROM code.
- Image **version number** is an anti rollback monotonic counter. The ROM code checks that it is higher or equal to the monotonic counter stored in OTP.
- Enabling signature verification is mandatory on secure closed chips.
- This field is an extract of PEM public key file that only kept the ECC Point coordinates x and y in a raw binary format (RFC 5480). This field will be hashed with SHA-256 and compared to the **Hash of pubKey** that is stored in OTP.
- This padding forces STM32 header size to 256 bytes (0x100).

For STM32MP15x lines :

- the monotonic counter is stored in **OTP 4**
- the Public Key Hash is stored in **OTP WORD 24 to 31**

2.4 Image programming

Once the images are signed, they can be programmed into the flash on the target board with **STM32CubeProgrammer**.

2.5 PKH check

Before really starting the authentication process, the ROM code compares the hash of the public key carried in the STM32 header with the one that was provisioned in OTP.



2.6 Authentication

2.6.1 Bootrom authentication

Using a **signed** binary, the ROM code authenticates and starts the FSBL.

If the authentication fails, the ROM code enters into a serial boot loop indicated by the blinking Error LED (cf *Bootrom common debug and error cases*)

The ROM code provides secure services to the FSBL for image authentication with the same ECC pair of keys, so there is no need to support ECDSA algorithm in FSBL.

2.6.2 TF-A authentication

TF-A is the FSBL used by the Trusted boot chain. It is in charge of loading and verifying U-boot and (if used) OP-TEE image binaries.

Each time a **signed** binary is used, TF-A will print the following status:

```
INFO:    Check signature on Non-Full-Secured platform
```

If the image authentication fails the boot stage traps the CPU and no more trace is displayed.

2.7 Closing the device

Notice that this last step is not shown in the diagram above.

Without any other modification, the device is able to perform image authentication but non authenticated images can still be used and executed: the device is still opened, let's see this as a kind of test mode to check that the PKH is properly set.

As soon as the authentication process is confirmed, the device can be closed and the user forced to use signed images.

OTP WORD0 bit 6 is the OTP bit that closes the device. Burning this bit will lock authentication processing and force authentication from the Boot ROM. Non signed binaries will not be supported anymore on the target.

To program this bit, the *STM32CubeProgrammer* or U-Boot command line interface can be used.

Here is how to proceed with U-Boot:

```
Board $> fuse prog 0 0x0 0x40
```

Warning

Once this bit is written the platform is locked

- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Microprocessor Unit

Elliptic Curve Digital Signature Algorithm

Elliptic curve cryptography



Error Correction Capability

One Time Programmed

Doubledata rate (memory domain)

First Stage Boot Loader

Second Stage Boot Loader

Das U-Boot -- the Universal Boot Loader (see [U-Boot_overview](#))

Trusted Firmware for Arm Cortex-A

Read Only Memory

Privacy Enhanced Mail (File format for storing and sending cryptographic keys, certificates, and other data)

Secure Hash Algorithm

Light-emitting diode

Open Portable Trusted Execution Environment

Central processing unit