How to use the secure bootloader on STM32WB0 MCUs

Introduction

The secure bootloader is part of the UART bootloader features on the STM32WB0 series that are preprogrammed at manufacturing.

The secure bootloader verifies the user application that needs to be authenticated, differentiating authorized firmware from nonauthorized firmware. Only authorized, trusted applications are allowed to run after this verification.

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

Table 1. Applicable products

Product type	Part number or product series
Microcontrollers	STM32WB0 series

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

1 General information

This document applies to Arm®-based devices.

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

arm

Reference documents

[1] STM32WB09xE datasheet (DS14210)

[2] STM32WB05xN datasheet (DS14620)

[3] STM32WB05xZ datasheet (DS14591)

[4] STM32WB07xC, STM32WB06xC datasheet (DS14676)

2 Secure bootloader

The secure bootloader uses asymmetric cryptographic algorithms with key pairs (public, private).

RSA-2048 is used with the 256 bytes public key size.

This authentication method allows an application firmware to be authenticated by generating and appending a digital sign to it.

The secure bootloader feature can be activated by writing a specific value on the OTP area.

2.1 How it works

The following steps describe the process to follow to authenticate an application firmware at user production time:

The user generates the key pairs (public and private), and programs its device using the signed firmware:

- The owner signs the application firmware using the private key. The generated sign is appended to the application firmware (digital sign size is 2048 bits).
- The device OTP sections are used to store the generated public key.

The secure bootloader uses the following components to verify/authenticate the application image:

- The public key stored on the device OTP.
- The application firmware image on the flash (with information related to the size).
- The digital signature appended at the end of the application firmware image.
- The OTP dedicated location for enabling the secure bootloader.

Only the application firmware image signed with the correct private key is executable and the private key is never shared or stored inside the devices.

The key point of this method is that the application firmware signing is done using the private key, which is not stored on the device, and authentication/verification is done using the associated public key. If the application firmware has not been signed with the correct private key (owned by the user), the devices cannot execute it.

2.2 How to store the public key in the OTP memory

The OTP memory is used to store the public key and other basic information needed to activate the secure bootloader feature.

To enable the secure bootloader feature, the OTP memory must contain the following information at specified addresses:

1. Activation Word@ OTP address 0x10001800
   a. The activation word is: 0xEC1CC10B
   b. This word is only able to activate the secure boot feature in the bootloader code
2. Start flash location of signed application firmware @ OTP address 0x10001804
   a. This is the interrupt vector table start address of the signed application firmware (for instance, 0x10040000)

*Note: To guarantee the correct secure verification, the application **must** start from the flash bottom at address 0x10040000. The same address **must** be stored at OTP address 0x10001804. In this way, only genuine signed firmware can be executed from the flash.*

3. R2 public key @ OTP address 0x10001808
   a. R2 public key with length 256 bytes
4. Modulus public key @ OTP address 0x10001908
   a. Modulus public key with length 256 bytes
5. Exponent public key @ OTP address 0x10001A08
   a. Exponents public key with length 4 bytes

The total size of the information to store inside the OTP area is 524 bytes. When all the data is stored inside the OTP area, it is mandatory to enable the OTP lock memory mechanism by writing a word different from 0xFFFFFFFF at OTP address 0x10001BFC.

2.3 How to sign an application firmware

The list below describes the steps to follow to sign an application firmware:

1. Generate the public/private keys.
2. Generate a digital sign from an input file.
3. Append the digital sign to the input file.
4. Store the public key on the selected device OTP.
5. Enable the secure bootloader on the selected device.

To enable the secure bootloader feature, the following secure bootloader utilities are available within the STM32Cube Programmer SW package:

Keys generation

STM32TrustedPackageCreator_CLI.exe -keygen: this utility generates the public and private keys for the RSA-2048 algorithm.

This utility requires the following parameters:

- `-k <Destination file with path for private key generated> <Destination file with path for public key generated>`
- `-f <Destination file with path to save the public_key.c,public_key.py files and public_key.txt files >`

The option “k” creates the public and private keys which are saved on the specified files.

The option “f” creates the public key saved in three formats on the specified destination files. The first format supports C projects, and the second format supports Python scripts.

A utility example is provided below:

```
STM32TrustedPackageCreator_CLI.exe -keygen -k <Output_File_priv_path>\private.pem
<Output_File_pub_path>\public.pem -f <Output_File_pub_path>\public_key.c
<Output_File_pub_path>\public_key.py <Output_File_pub_path>\public_key.txt
```

Firmware signing

STM32TrustedPackageCreator_CLI.exe -signbin: this utility creates a signed firmware starting from a binary file.

This utility takes a not signed raw binary file in input, and executes the following operations in this order:

- Addition of the firmware size inside the binary file at offset 0x18.
- Generation of the signature for the application firmware. To create the signature, the utility uses the private key already generated by the *STM32TrustedPackageCreator_CLI.exe -keygen* utility.
- Addition of the signature at the end of the binary file. A new signed file is created.

This utility requires the following parameters:

- `-k <File with path for private key used to create the signature. This is the same file with path specified with the option “k” of the STM32TrustedPackageCreator_CLI.exe -keygen utility>`
- `-f <Input raw binary image file with path not signed>`
- `-o <Output raw binary image file with path signed>`

The file generated with the option “o” is the signed application that can be stored in the main flash.

A utility example is provided below:

```
STM32TrustedPackageCreator_CLI.exe -signbin -k <Output_File_priv_path>\private.pem
-f <binary_to_sign_path>\<file_to_be_signed>.bin -o
<output_signed_bin_path>\<file_signed.bin>
```

Key storage on OTP

STM32_programmer_CLI.exe -storekeyotp: this utility stores all the key information inside the OTP area and locks it. This utility requires the device to be in bootloader mode.

The utility requires the following parameters:

- `-br= <baudrate>`: UART baud rate
  `-p= <parity>`: Parity bit, value in {NONE,ODD,EVEN}, default EVEN.
- `-c port= <PortName>`
- `<key_path>`: C file with path for the public key to store in OTP

- <start_address>: Start FW address, that is the interrupt vector table start address

An utility example is provided below:

```
STM32_Programmer_CLI.exe -c port=COMxx br=115200 p=none -storekeyotp
<Output_File_pub_path>\public_key.c 0x10040000
```

Revision history

Table 2. **Document revision history**

Date	Version	Changes
14-Jun-2024	1	Initial release.
22-Oct-2024	2	Updated Section 2.3: How to sign an application firmware.
28-Apr-2025	3	Updated Section Introduction and Section 2.2: How to store the public key in the OTP memory.

Contents

List of tables