M261 Trusted Boot

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	Introduce the M261 Secure Bootloader, Secure Boot verification mechanism, and how it works to perform a trusted boot; and demonstrate using the SecureBootDemo to create a trusted execution system launched from the Secure Bootloader.
Apply to	NuMicro® M261 series

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Overview

The M261 Secure Bootloader is a bootable code pre-written in the M261 Mask ROM. It is tamper-proof due to the Read-Only feature of Mask ROM. Besides, the memory attribute of Secure Bootloader is configured as Execute-Only to prevent a thief from reading the source code or tracing the instruction execution.

The trusted boot is a chain of trust process for verifying each software identity and integrity on the system. The M261 Secure Bootloader provides a root of trust solution for a system developer to create a trusted execution system in the M261 microcontroller (MCU).

In this document, Chapter 2 will detail the features of M261 Secure Bootloader (NuBL1). Chapter 3 introduces the features of NuBL2 and NuBL32. Chapter 4 describes the secure boot verification mechanism in NuBL1 and NuBL2.

Chapter 5 "SecureBootDemo" will demonstrate how to develop a trusted execution system launched from M261 Secure Bootloader (also named NuBL1), including a trusted boot code (also named NuBL2) and a system code (also named NuBL32). Figure 1-1 shows the trusted execution system in M261.

- System boots from M261 Mask ROM, then NuBL1 performs Secure Boot verification to NuBL2
- System runs in NuBL2 and performs the identity and integrity verification to NuBL32

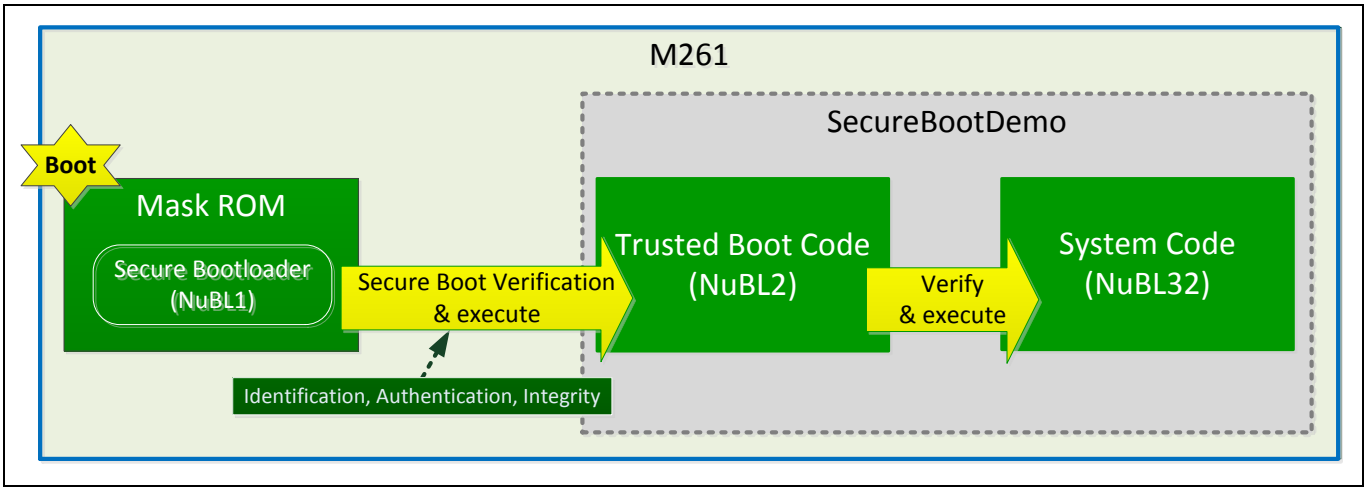

Figure 1-1 Trusted Execution System in M261

2 Secure Bootloader (NuBL1)

The Secure Bootloader (NuBL1) is a built-in bootloader in the M261 Mask ROM. It is a boot-up code to verify the next stage trusted boot code (NuBL2) in the M261 trusted execution system.

This chapter introduces the main features and related configuration of Secure Bootloader.

2.1 Features

The features of Secure Bootloader are as follows.

- Memory attribute is configured as Execute-Only Memory (XOM)
- Support Secure Boot verification for validating the identity and integrity of the next stage firmware
  - The verification is based on the ECDSA (Elliptic Curve Digital Signature Algorithm)
  - Store the verification fail status at SRAM 0x2000_0000
- Use internal HIRC-48M as system clock

2.2 Boot Source Selection

CONFIG0 in the User Configuration Block is an internal programmable configuration area for specifying boot options in the M261 series MCU. The boot source is selected by setting the CONFIG[5] MBS and CONFIG[7] CBS, as shown in Figure 2-1.

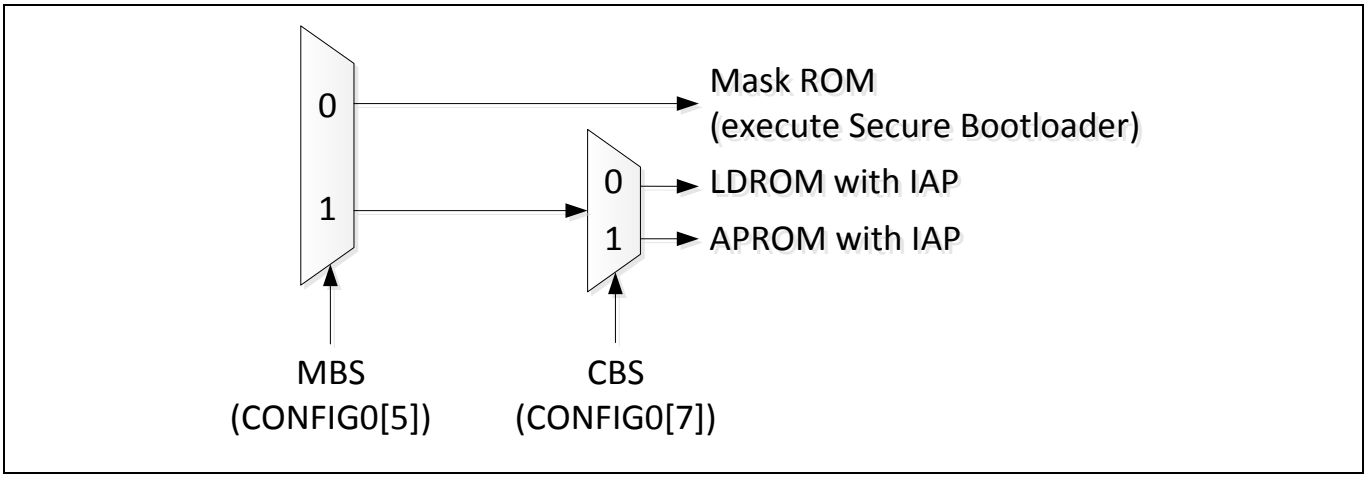

Figure 2-1 Boot Source Selection

According to the boot source selection, the CONFIG[5] MBS should be configured to 0 to execute the Secure Bootloader in the M261 Mask ROM. To ensure the trusted boot, the system developer should not erase CONFIG[5] MBS to 1, which will cause the system reboot and execute the unauthenticated code in APROM or LDROM.

Once a Flash mass erase command is executed, all Flash data including CONFIG0 will be erased to 1, thus the system will start from APROM on the next reboot. The system developer should configure MBS to 0 and program the next trusted boot code again to implement a trusted execution system launched from Secure Bootloader.

2.3 Execution Region

The Secure Bootloader code is built in the tamper-proof Mask ROM and placed between 0x0080_0000 to 0x0080_7FFF, with a total of 32KB.

The SRAM range used in the Secure Bootloader is from SRAM 0x2000_0000 to 0x2000_3FFF, with a total of 16KB.

Figure 2-2 shows the execution region of Secure Bootloader.

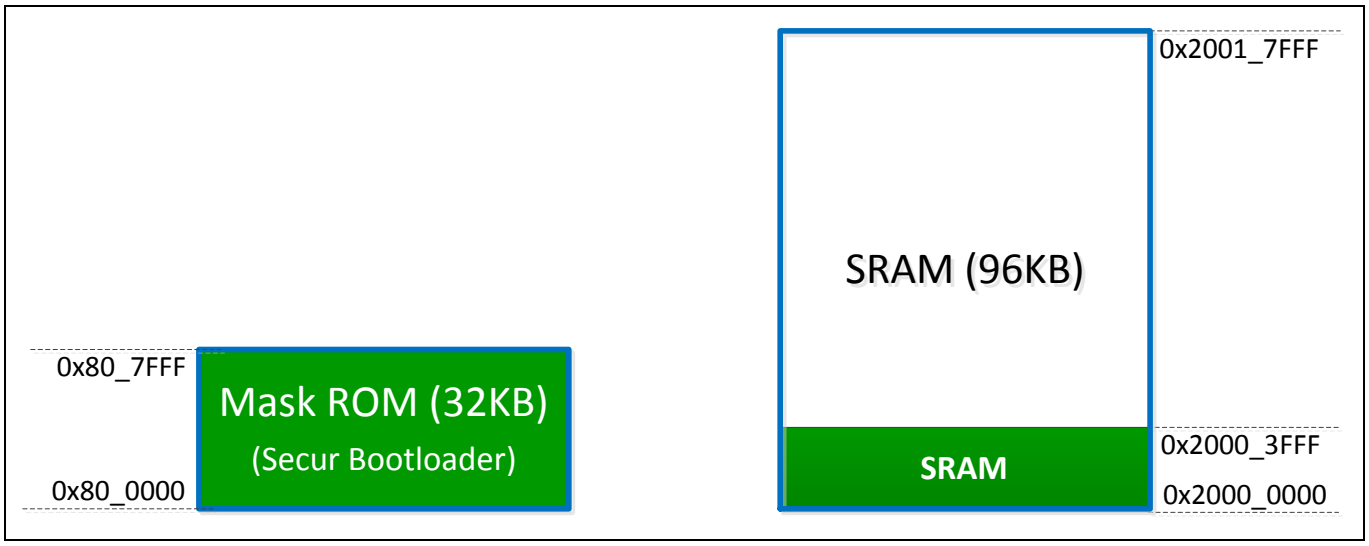

Figure 2-2 Execution Region of Secure Bootloader

3 NuBL2 and NuBL32

In addition to the Secure Bootloader (NuBL1) in the Mask ROM, the M261 trusted execution system also includes a trusted boot code (NuBL2) and a system code (NuBL32) to achieve the trusted boot. The function of these codes is briefly described below.

- **NuBL2:** A trusted boot code as a customer loader to load and execute the next stage system code (NuBL32).
- **NuBL32:** A system code to initialize system services and perform applications on the M261 MCU.

Figure 3-1 shows the relation among the NuBL2 and NuBL32 in the M261 trusted execution system.

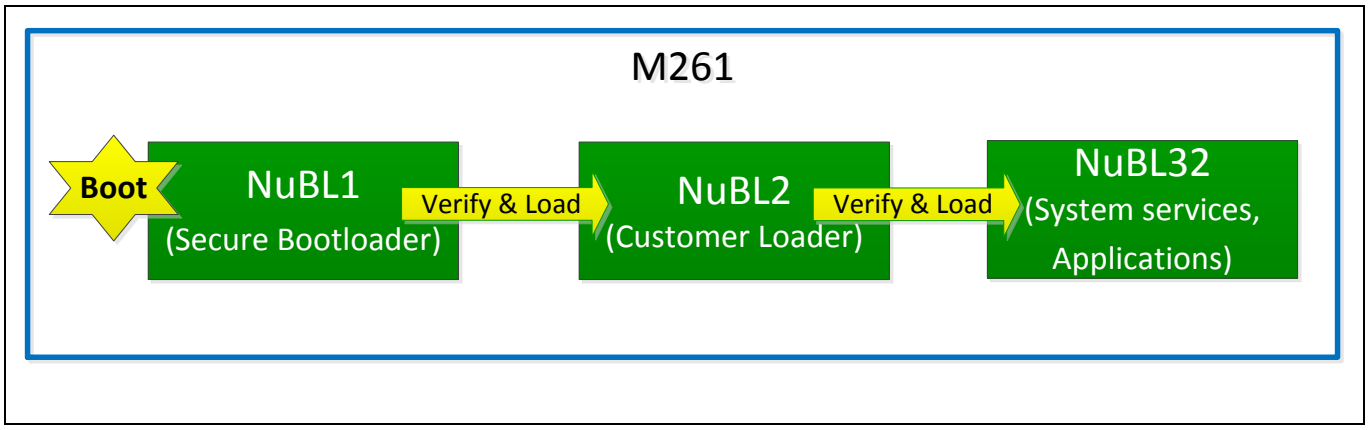

Figure 3-1 NuBL2/NuBL32 Function in M261 Trusted Execution System

Alternatively, a system developer can also integrate NuBL32 into NuBL2 to support NuBL2 only without customer loader function, as shown in Figure 3-2.

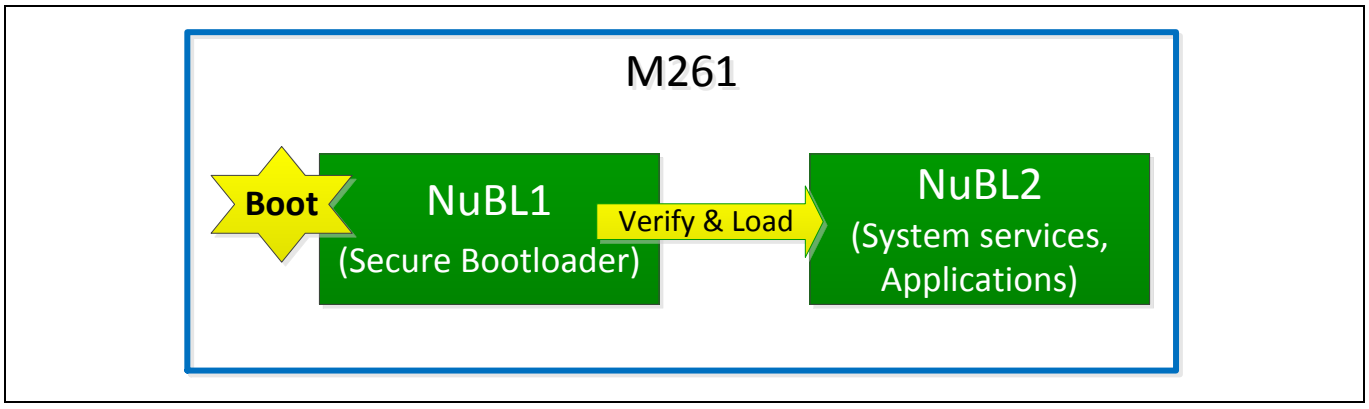

Figure 3-2 Only NuBL2 in Trusted Execution System

The following sections describe the main features of NuBL2 and NuBL32 in the M261 trusted

execution system.

3.1 NuBL2 Features

NuBL2 is the trusted boot code used to verify the NuBL32. The features of NuBL2 are as follows.

- The first stage code in the trusted execution system booted by NuBL1.
- Verifies NuBL32 identity and integrity. After successfully verifying NuBL32, the NuBL2 will generate a software reset to execute NuBL32.

3.2 NuBL32 Features

The NuBL32 is a system code. The features of NuBL32 is as follows.

- The system initialization and application code in the trusted execution system booted by NuBL2.

4 Firmware Verification Mechanism

The current firmware needs to verify its identity and integrity before executing the next stage firmware. In order to correctly verify the firmware, some components related to the next stage firmware are required. The necessary components will be introduced in section 4.1.
The verification procedure is supported in both NuBL1 and NuBL2. The NuBL1 or NuBL2 identifies the firmware and verifies its integrity. The detailed verification procedure will be introduced in section 4.2.

4.1 Required Components

4.1.1 Required Components to Verify the NuBL2

To achieve the Secure Boot verification in Secure Bootloader (NuBL1), the required components related to NuBL2 are listed as follows.

- NuBL1 FW (Secure Bootloader, built in the M261 Mask ROM)
- NuBL2 Marker (programmed in Flash)
- NuBL2 Public Key Hash (programmed in OTP0~3 region)
- NuBL2 FW INFO (programmed in Flash)
- NuBL2 FW (a trusted boot code, programmed in Flash)

Figure 4-1 shows the required components for verifying NuBL2 in the M261 trusted execution system.

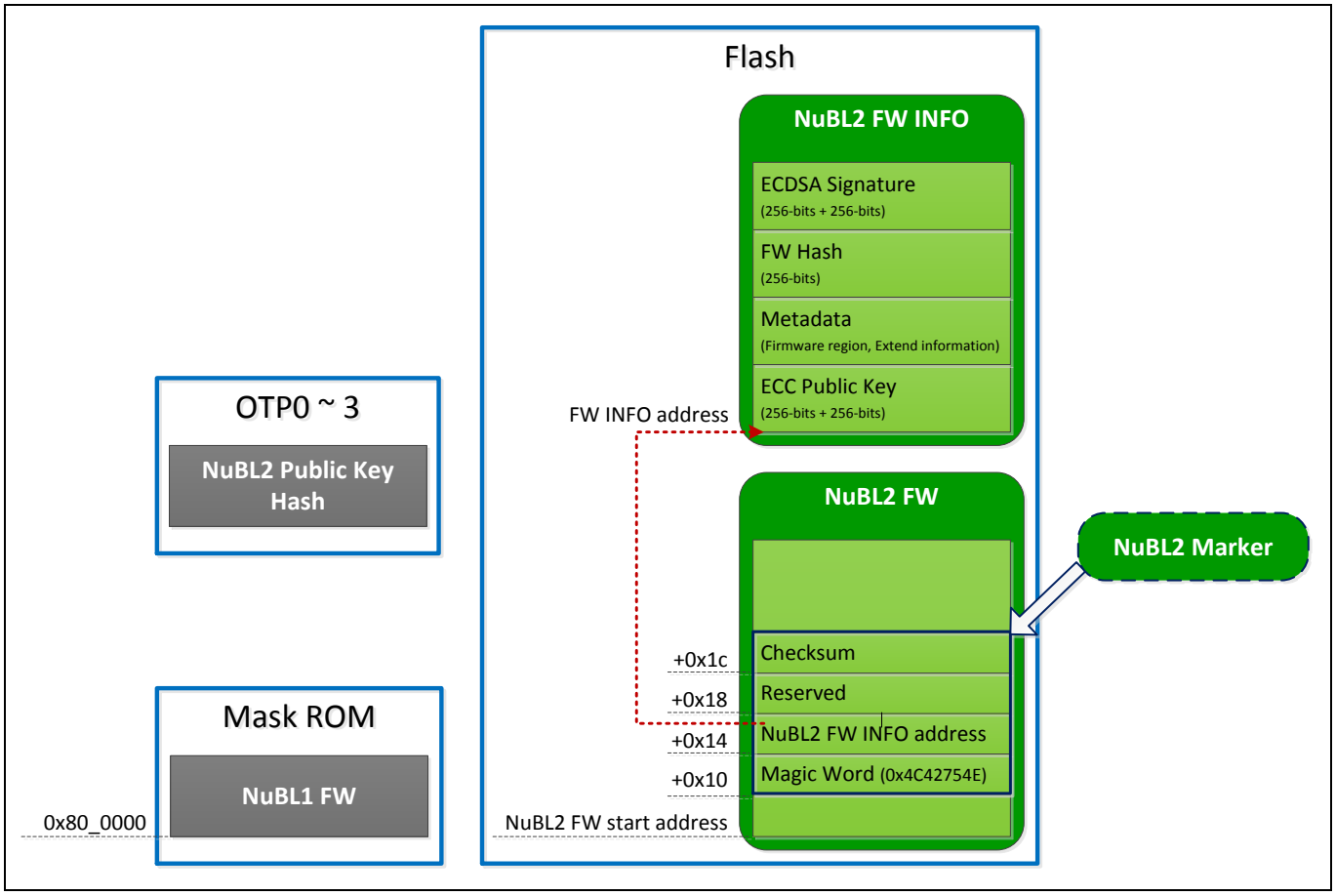

Figure 4-1 Required Component for Verifying the NuBL2

The function of each component is described as follows.

- NuBL1 FW is used to verify the identity and integrity of NuBL2
  - Use NuBL2 Public Key Hash to identify NuBL2Use NuBL2 FW INFO and its ECDSA signature for NuBL2 authentication
  - Use NuBL2 firmware hash for verifying NuBL2 integrity
- NuBL2 Marker is used to indicate the NuBL2 execution address and NuBL2 firmware information address.
- NuBL2 Public Key Hash located at OTP0~3 is a SHA-256 digest of NuBL2 ECC public key
  - If the OTP0~3 data are all “1”, it means that there is no public key hash in this system, and NuBL1 will skip to identify NuBL2.
- NuBL2 FW INFO (firmware information) is a data block consisting of NuBL2 “ECC Public Key”, “Metadata”, “FW Hash” and “ECDSA Signature”.
  - Metadata includes firmware region and extend NuBL2 information

- ECDSA Signature is generated with NuBL2 ECC private key and a hash message.
  - The hash message is a SHA-256 digest calculated by NuBL2 “ECC Public Key”, “Metadata” and “FW Hash”.

4.1.2 Required Components to Verify the NuBL32

NuBL2 needs the required components as listed below to verify the NuB32.

- NuBL32 Public Key Storage (programmed in Flash)
- NuBL32 FW INFO (programmed in Flash)
- NuBL32 FW (programmed in Flash)

Figure 4-2 shows the required components for verifying the NuBL32 in the M261 trusted execution system.

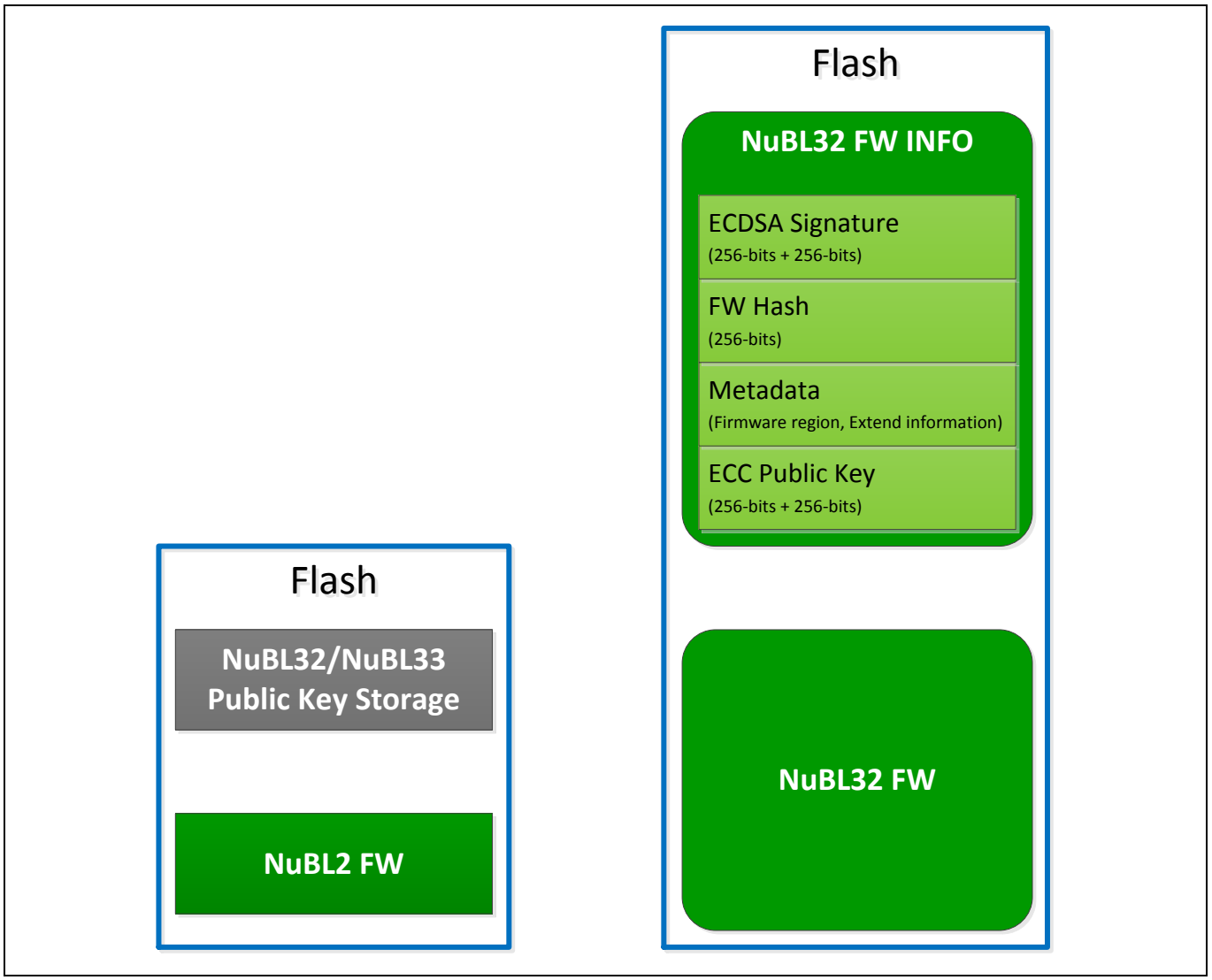

Figure 4-2 Required Component for Verifying the NuBL32

The function of each requirement is described as follows.

- NuBL32 Public Key Storage is used for NuBL2 to obtain the NuBL32 ECC public key.
  - Includes encrypted NuBL32 ECC public key and its SHA-256 digest
- NuBL32 FW INFO (firmware information) is a data block consisting of NuBL32 “ECC Public Key”, “Metadata”, “FW Hash” and “ECDSA Signature”
  - Metadata includes firmware region and extend NuBL32 information.
  - ECDSA Signature is generated with NuBL32 ECC private key and a hash message.
    - The hash message is a SHA-256 digest calculated by NuBL32 “ECC Public Key”, “Metadata” and “FW Hash”.

4.2 Verification Procedure

The following sections will detail the verification procedure in NuBL1 and NuBL2, including identification, authentication and firmware integrity.

4.2.1 NuBL1 Verification Procedure

4.2.1.1 Identification

The identification is to ensure that the NuBL2 ECC public key (as NuBL2 firmware identity) exists in the M261 MCU.

The NuBL1 obtains the NuBL2 ECC public key in NuBL2 FW INFO, calculates its hash and identifies it with the NuBL2 public key hash stored in M261 OTP0~3.

If NuBL1 does not recognize the NuBL2 identity, no firmware can be executed from NuBL1. It means that only the firmware with the NuBL2 ECC public key (NuBL2 identity) can be executed on the M261 MCU.

Figure 4-3 shows the process of identification in NuBL1.

1. Reset to execution in Secure Bootloader (NuBL1).
2. Get and calculate NuBL2 ECC public key hash in (1).
3. Read OTP0~3 (public key hash) in (2) and identify it using the calculated NuBL2 ECC public key hash.

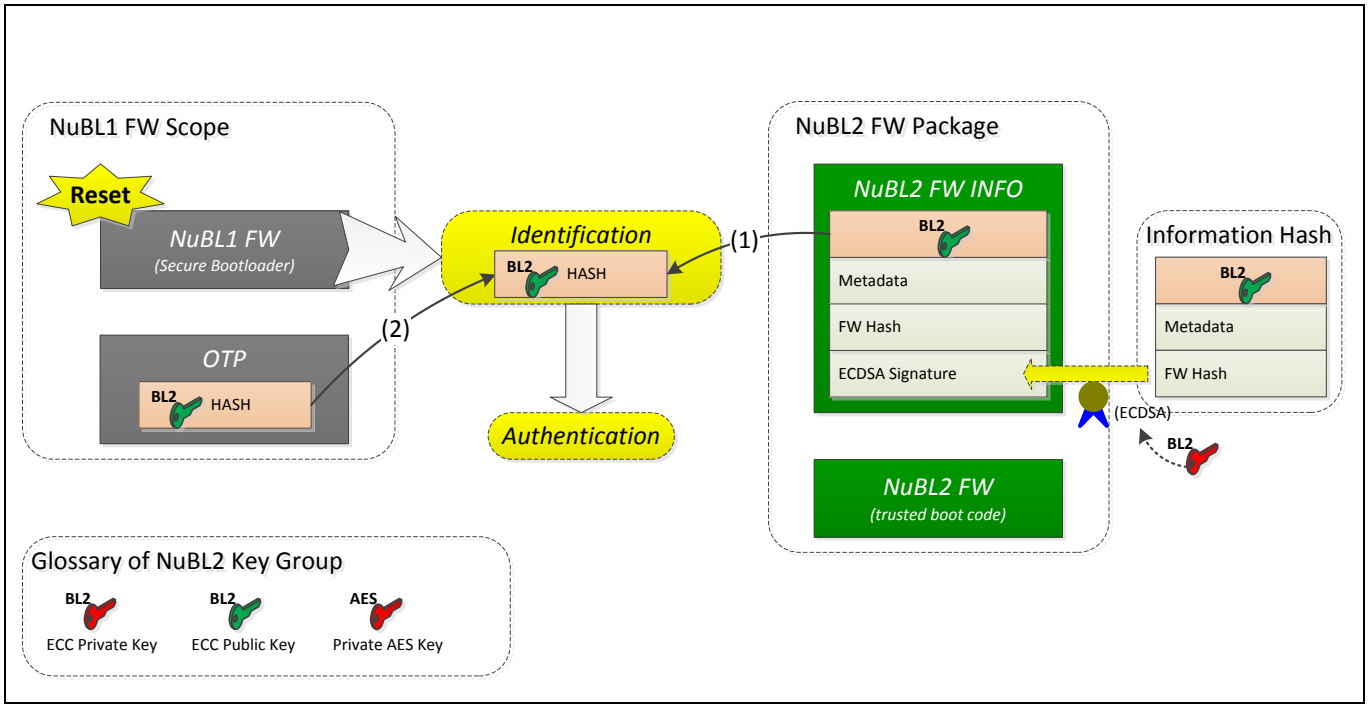

Figure 4-3 Identification in NuBL1

4.2.1.2 Authentication

The authentication intends to validate a NuBL2 FW INFO (firmware information) provided by the NuBL2 developer.

Since the “ECDSA Signature” in NuBL2 FW INFO is generated using the NuBL2 ECC private key and a hash message calculated by NuBL2 “ECC Public Key”, “Metadata” and “FW Hash”. NuBL1 can verify the “ECDSA Signature” using the NuBL2 ECC public key and the calculated hash message without knowing the exact NuBL2 ECC private key.

After authentication, NuBL1 can obtain the valid NuBL2 firmware region and hash for firmware integrity verification.

Figure 4-4 shows the process of authentication in NuBL1.

1. Calculate an information hash in (1), (2) and (3)
   - It’s a SHA-256 digest that calculated by NuBL2 “ECC Public Key”, “Metadata” and “FW Hash”
2. Perform ECDSA signature verification using the calculated information hash, the NuBL2 ECC public key in (4), and the ECDSA signature in (5)

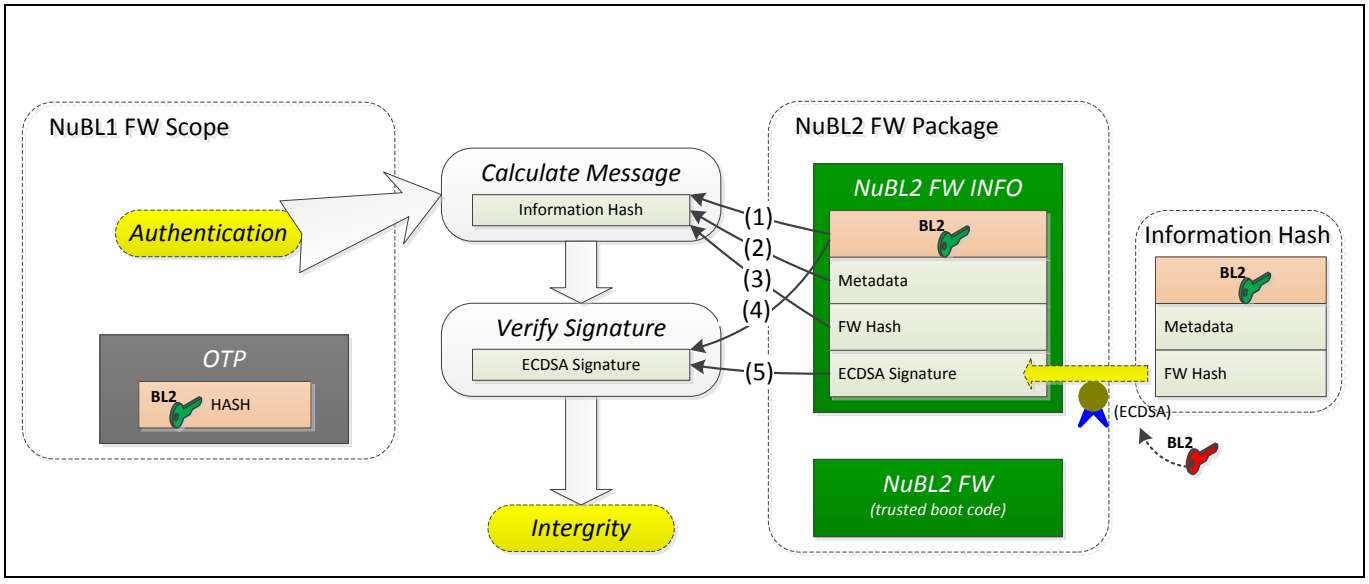

Figure 4-4 Authentication in NuBL1

4.2.1.3 Integrity

Performing the firmware (NuBL2 FW) integrity verification is the last verification mechanism in the trusted execution system.

NuBL1 obtains the firmware region in the Metadata from NuBL2 FW INFO to calculate the NuBL2 firmware hash.

After that, NuBL1 performs firmware integrity verification using the calculated NuBL2 firmware hash and FW Hash in NuBL2 FW INFO.

Figure 4-5 shows the process for verifying integrity in NuBL1.

1. Obtain FW Hash from NuBL2 FW INFO in (1).
2. Calculate the firmware hash in (2) and compare it with FW Hash.
   - The calculated NuBL2 firmware region is recorded in the Metadata.

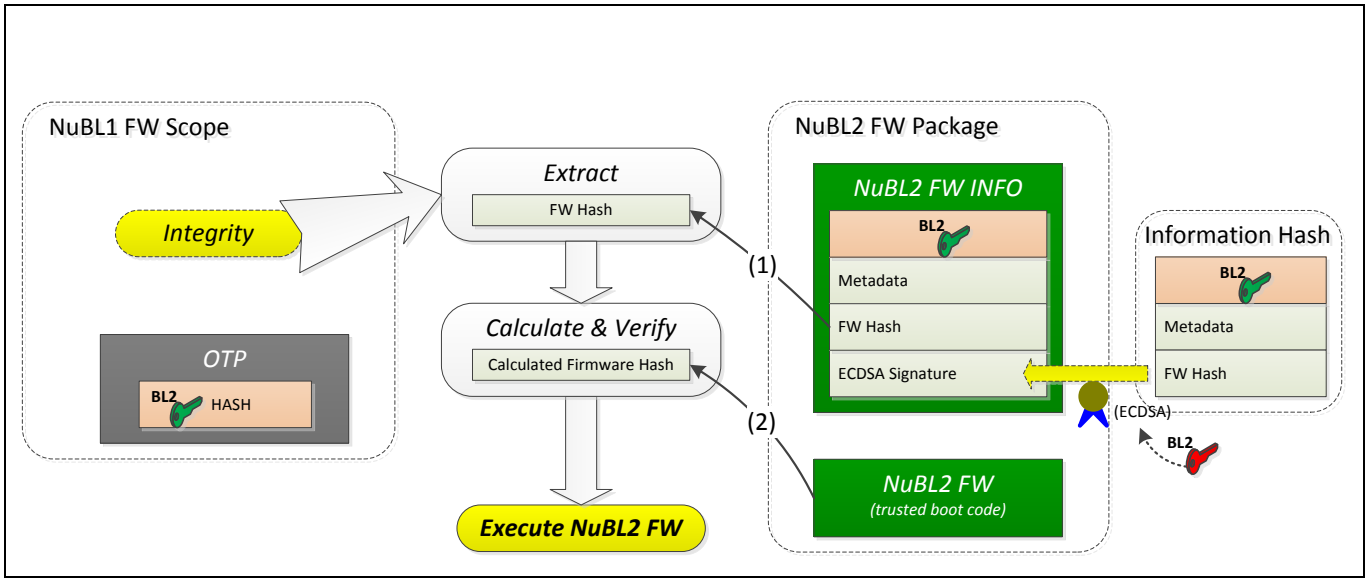

Figure 4-5 Integrity in NuBL1

4.2.2 NuBL2 Verification Procedure

The verification process in NuBL2 is similar to the NuBL1 verification process. But it uses the encrypted NuBL32 Public Key Storage instead of the public key hash in OTP0~3 for firmware identification.

The following sections detail the NuBL2 verification procedure.

4.2.2.1 Identification

The identification is to ensure that the NuBL32 ECC public key (as NuBL32 firmware identity) exists in the M261 MCU.

The Public Key Storage located in Flash consists of the encrypted NuBL32 ECC public key and its SHA-256 digest.

When system runs in NuBL2, NuBL2 verifies the integrity of Public Key Storage first, and then decrypts the Public Key Storage to obtain the NuBL32 ECC public key. After that, NuBL2 will perform an identification using this decrypted public key and the ECC public key in the NuBL32 FW INFO.

If NuBL2 does not recognize the NuBL32 identity, no firmware can be executed from NuBL2. It means that only firmware with a NuBL32 ECC public key (NuBL32 identity) can be executed on the M261 MCU.

Figure 4-6 shows the process to decrypt the Public Key Storage in NuBL2.

1. Use internal NuBL2 AES key to decrypt the Public Key Storage in (1).
2. Obtain the NuBL32 public key from the decrypted key storage in (2).

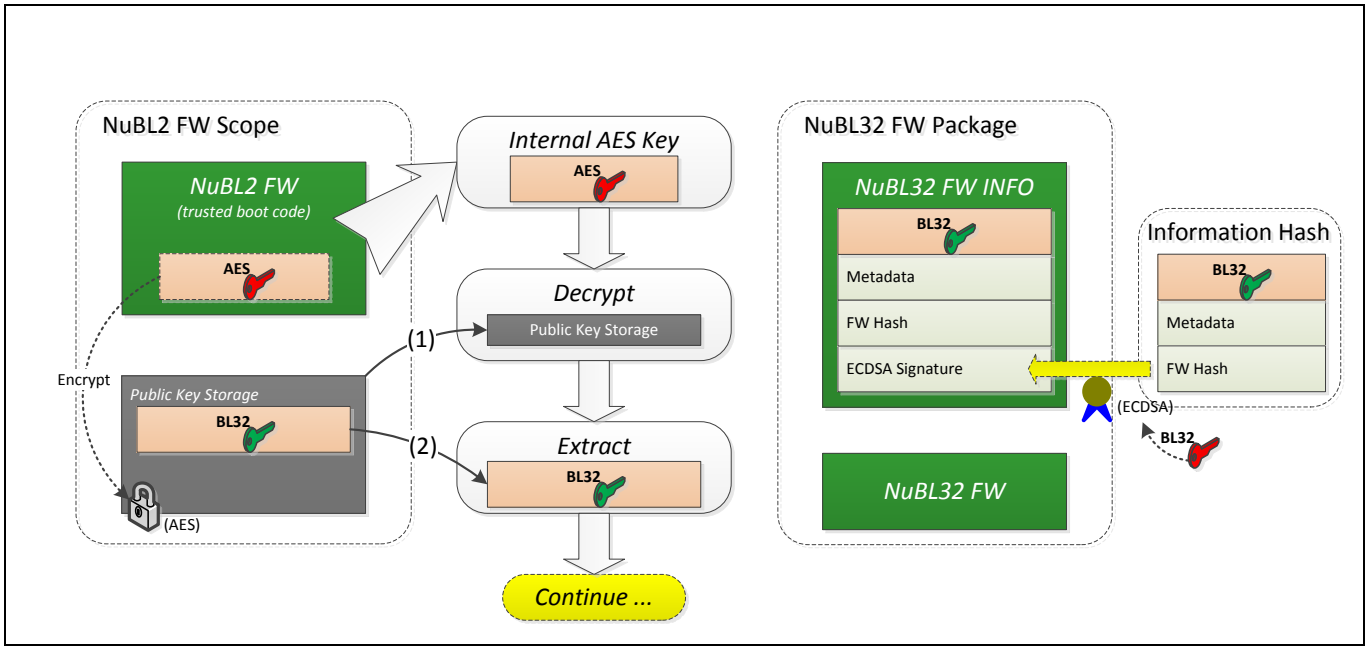

Figure 4-6 Decrypt Public Key Storage in NuBL2

Figure 4-7 shows the process for identification in NuBL2.

3. Get NuBL32 public key from the Public Key Storage in (3).
4. Obtain the NuBL32 public key in (4) and identify it using the public key decrypted from the Public Key Storage.

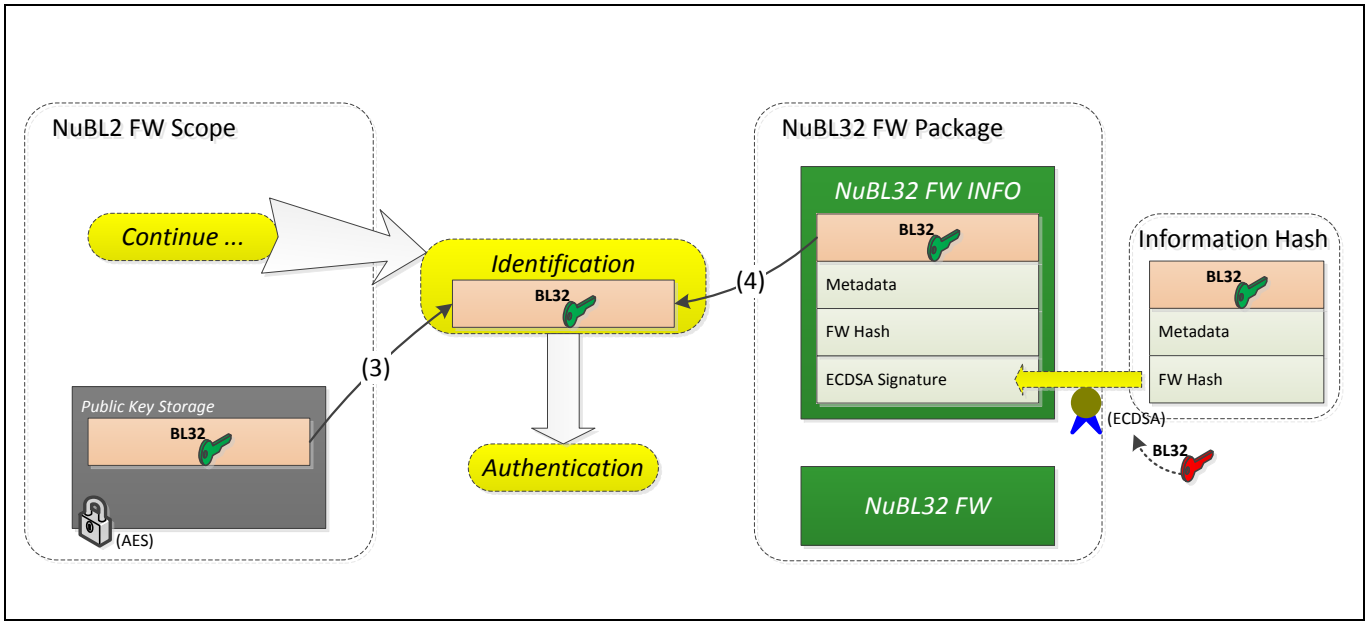

Figure 4-7 Identification in NuBL2

4.2.2.2 Authentication

The authentication intends to validate a NuBL32 FW INFO (firmware information) provided by the NuBL32 developer.

The NuBL32 FW INFO consists of “NuBL32 ECC Public Key”, “Metadata”, “FW Hash” and “ECDSA Signature”.

Since the “ECDSA Signature” in NuBL32 FW INFO is generated using the NuBL32 ECC private key and a hash message calculated by NuBL32 “ECC Public Key”, “Metadata” and “FW Hash”. NuBL2 can verify the “ECDSA Signature” using the NuBL32 ECC public key and the calculated hash message without knowing the exact NuBL32 ECC private key.

After authentication, NuBL2 can obtain the valid NuBL32 firmware region and hash for firmware integrity verification.

Figure 4-8 shows the process for authentication in NuBL2.

1. Calculate an information hash in (1), (2) and (3).
   - It’s a SHA-256 digest that calculated by NuBL32 “ECC Public Key”, “Metadata” and “FW Hash”.
2. Perform ECDSA signature verification using the calculated information hash, the NuBL32 ECC public key in (4), and the ECDSA signature in (5).

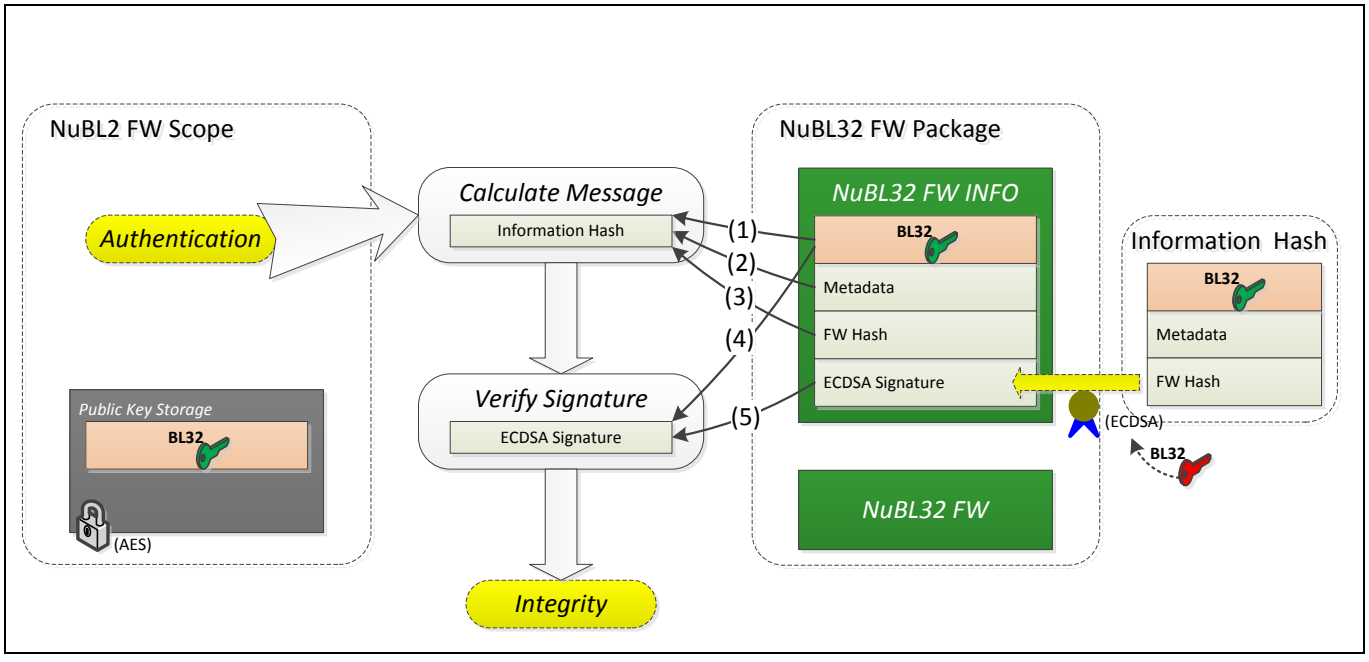

Figure 4-8 Authentication in NuBL2

4.2.2.3 Integrity

Performing the firmware (NuBL32 FW) integrity verification is the last verification mechanism in trusted execution system.

NuBL2 obtains the firmware region in the Metadata from NuBL32 FW INFO to calculate the NuBL32 firmware hash.

After that, NuBL2 performs firmware integrity verification using the calculated NuBL32 firmware hash and FW Hash in NuBL32 FW INFO.

Figure 4-9 shows the process for verifying integrity in NuBL2.

1. Obtain FW Hash from NuBL2 FW INFO in (1).
2. Calculate the firmware hash in (2) and compare it with FW Hash.
   - The calculated NuBL32 firmware region is recorded in the Metadata.

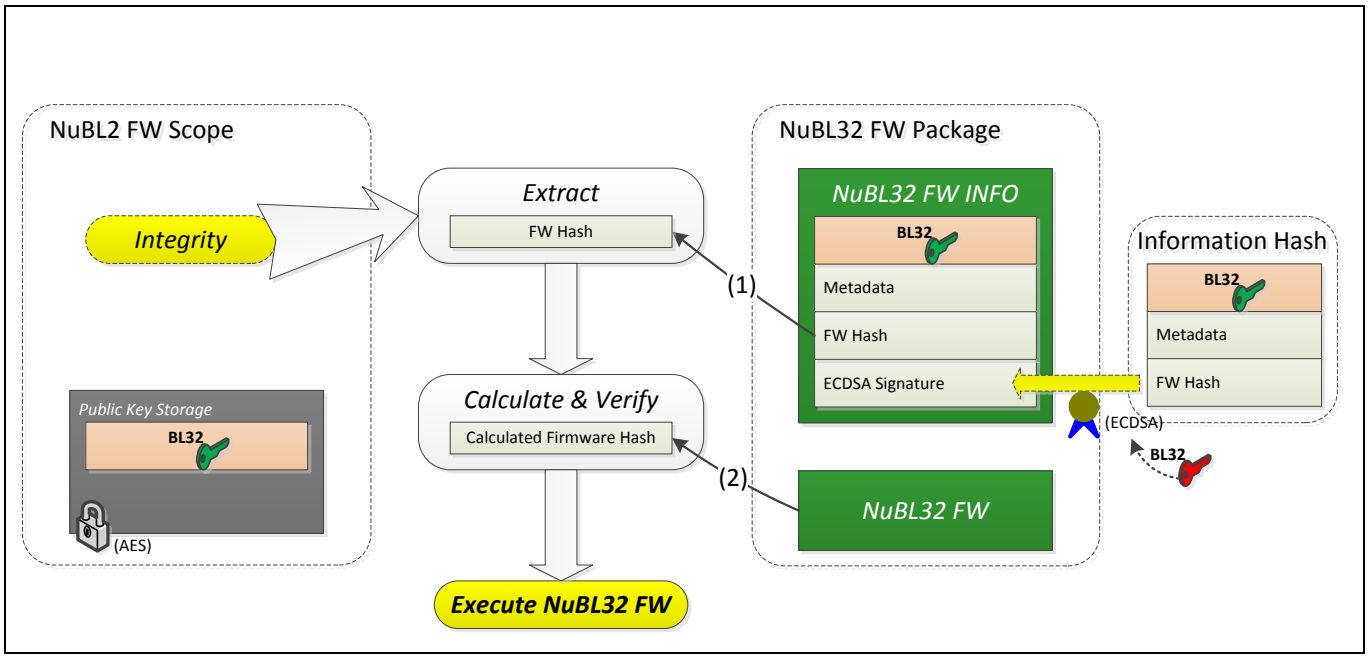

Figure 4-9 Integrity in NuBL2

5 SecureBootDemo

The SecureBootDemo sample code is created by using Keil® MDK to demonstrate how to create a trusted execution system, including the Secure Bootloader (NuBL1) verifies the trusted boot code (NuBL2) and the NuBL2 verifies the next application code (NuBL32).

All the sample code could be found at "\SampleCode\MKROM\SecureBootDemo" in the M261 BSP, including "NuBL2" folder containing the first stage code and "NuBL32" folder containing the second stage system code.

Figure 5-1 shows the execution regions of NuBL2/NuBL32 in SecureBootDemo.

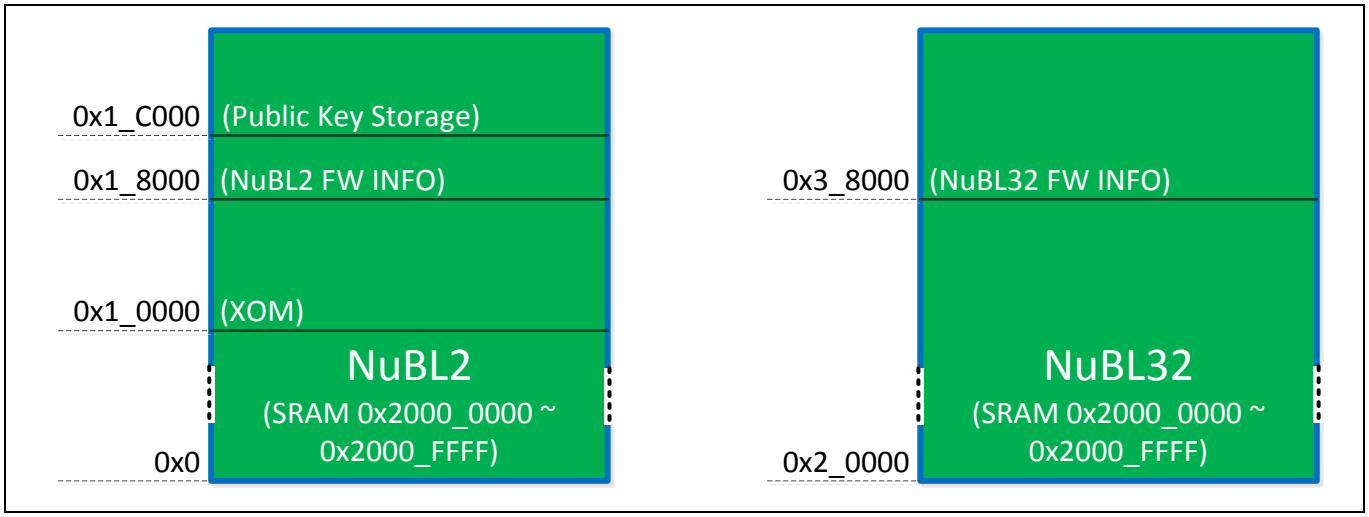

Figure 5-1 Execution Regions of NuBL2 and NuBL32

- Execution region of NuBL2:
  - Flash from 0x0 to 0x0001_FFFF, total 128KB
  - SRAM from 0x2000_0000 to 0x2000_FFFF, total 64KB
- Execution region of NuBL32:
  - Flash from 0x2_0000 to 0x0003_FFFF, total 128KB
  - SRAM from 0x2000_0000 to 0x2000_FFFF, total 64KB

Section 5.1 to Section 5.4 will introduce how to create FW INFO, NuBL2 Maker and Public Key Storage of NuBL32. Section 5.5 will introduce how to download NuBL2/NuBL32 firmware and the firmware information using an .ini file in the NuBL2 project.

5.1 NuBL2/NuBL32 FW INFO

The NuBL2/NuBL32 FW INFO (firmware information) is a required component for verifying firmware identity and integrity. It consists of “ECC Public Key”, “Metadata”, “FW Hash” and “ECDSA Signature”.

5.1.1 Information Template

The source file FwInfo.c includes an information template and is used for NuBL2/NuBL32 project to generate the target FW INFO and reserve a Flash region to store it.

Figure 5-2 shows the NuBL2 firmware information template. For the source file, refer to “M261BSP\SampleCode\MKROM\SecureBootDemo\NuBL2\FwInfo\FwInfo.c”.

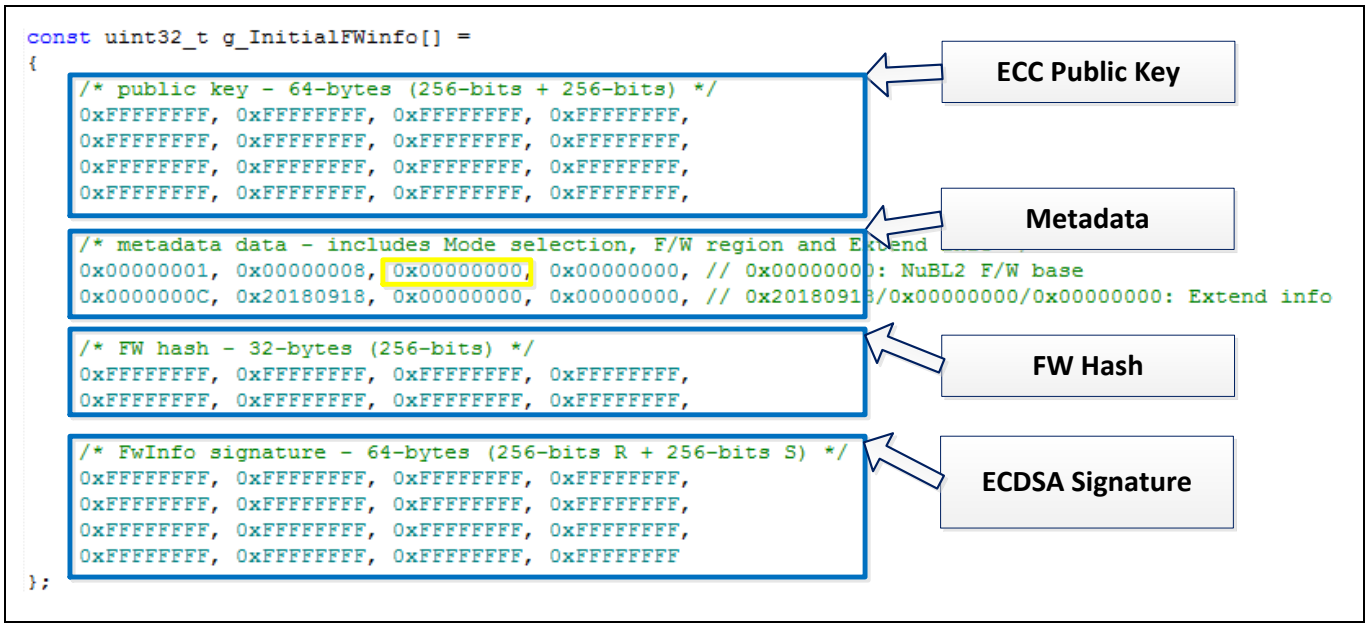

Figure 5-2 NuBL2 Firmware Information Template

Figure 5-3 shows the NuBL32 firmware information template. For the source file, refer to “M261BSP\SampleCode\MKROM\SecureBootDemo\NuBL32\FwInfo\FwInfo.c”.

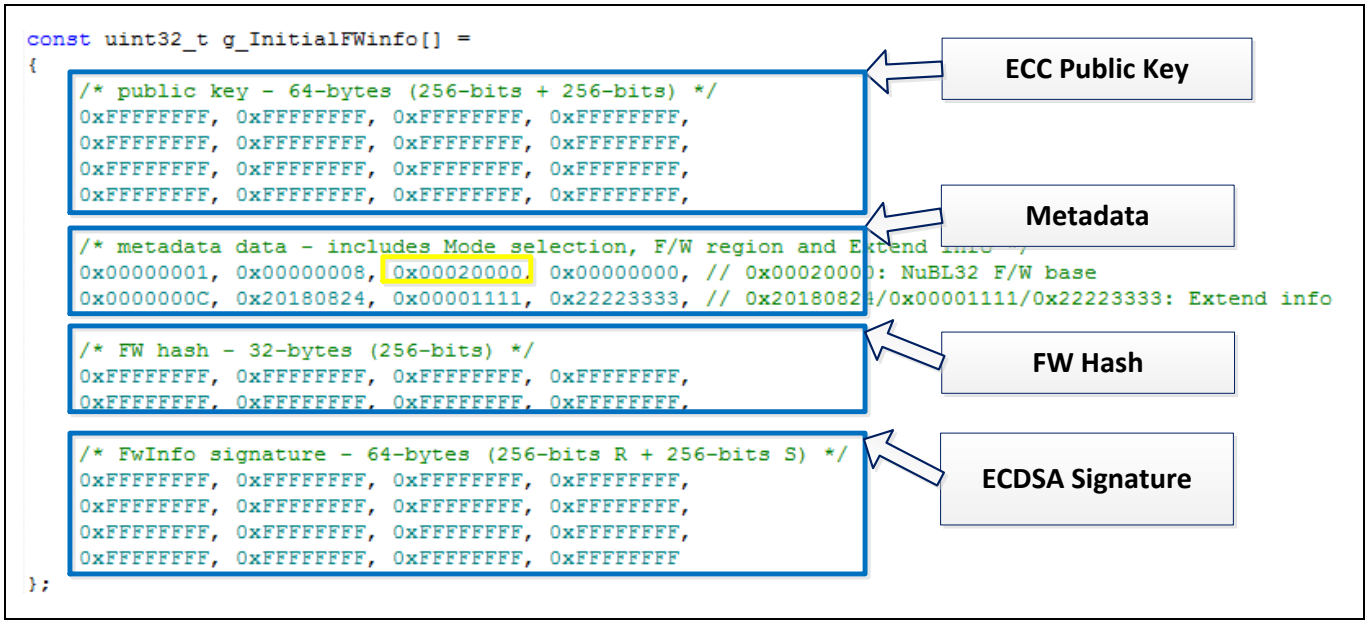

Figure 5-3 NuBL32 Firmware Information Template

After successfully building the NuBL2/NuBL32 project, the “ECC Public Key”, “Metadata”, “FW Hash” and “ECDSA Signature” will automatically update in the target firmware information according to its ECC private/public key pair and firmware region.

5.1.2 FW INFO Structure

5.1.2.1 ECC Public Key

The ECC Public Key consists of two sets of 256-bits key data, total 512-bits.

This public key pair intends to identify the firmware and authenticate the firmware information.

5.1.2.2 Metadata

The Metadata includes the Mode selection, Firmware region and Extend information field.

Figure 5-4 shows the Metadata in NuBL2 FW INFO.

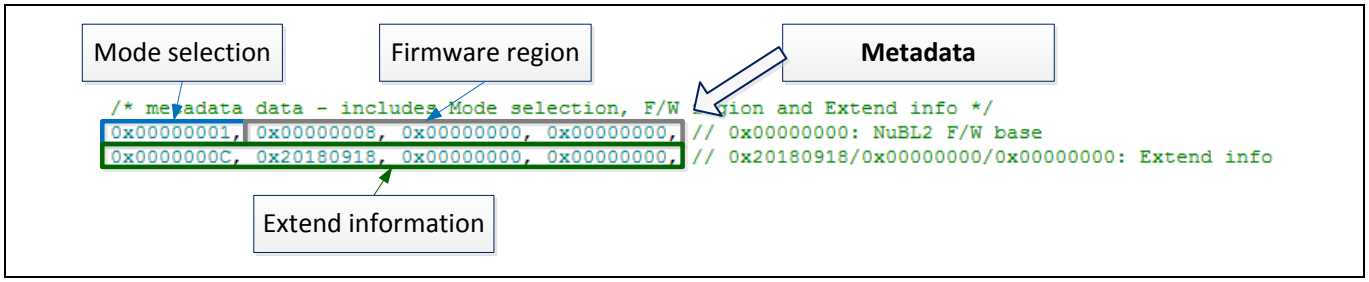

Figure 5-4 Metadata in NuBL2 FW INFO

The description of each field is as follows.

- Mode selection

The first word of Metadata is to configure the Mode selection as shown in Table 5-1.

Valid Bits	Description
BIT[1:0]	MUST be 1
BIT[2]	0: Not support PID in the information hash 1: Supports PID in the information hash
BIT[3]	0: Not support UID in the information hash 1: Supports UID[0]~[2] in the information hash
BIT[4]	0: Not support UCID in the information hash 1: Supports UCID[0]~[3] in the information hash
BIT[31:5]	Reserved, MUST be all 0

Table 5-1 Mode Selection

- Firmware region

  The first word of Firmware region indicates the size of firmware region, and the next two words are used to store the firmware start address and the firmware size as shown in Figure 5-5.

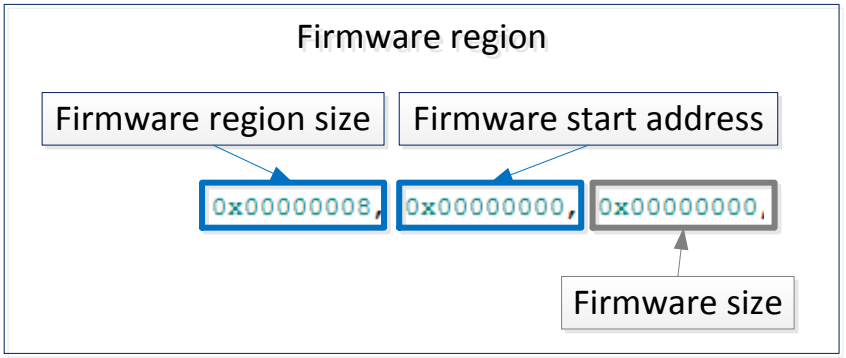

Figure 5-5 Example of Firmware Region

  - Firmware region size 0x8 indicates this field contains only one firmware region
    - 4-bytes is for Firmware start address, the other 4-bytes is for Firmware size
  - Firmware start address 0x0 configures the firmware start address for calculating firmware hash
  - Firmware size default to 0x0 in FwInfo.c. But the actual firmware size will be written in the target FwInfo region after built the NuBL2 project
- Extend information

  The first word of Extend information indicates the size of extend information, should be less than 256 bytes and a multiple of 4, and the follow words are used to set the extend information as shown in Figure 5-6.

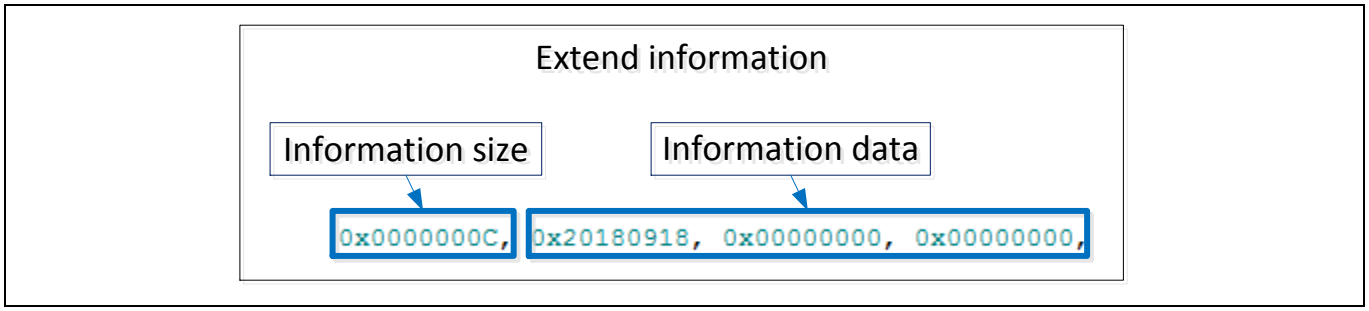

Figure 5-6 Example of Extend Information

5.1.2.3 FW Hash

The FW Hash (firmware hash) is a SHA-256 digest and intend for verifying the firmware integrity. It is calculated based on the firmware region specified in the Metadata.

5.1.2.4 ECDSA Signature

This ECDSA Signature is a total 512-bits ECDSA firmware information signature. An ECC private key and information hash are necessary for generating the target ECDSA Signature. The information hash is generated by the following data and order.

- ECC Public Key
- Metadata
- FW Hash
- ID Information
  - BIT[4:2] in the first word of Metadata is to configure whether the ID information is supported to be calculated in information hash
    - BIT[4:2] is 0x0, not support any ID
    - BIT[4:2] is not 0x0, supports specified ID

Moreover, using this ECDSA Signature can authenticate the firmware information with the ECC public key and information hash.

5.1.3 FW INFO Generation

This section describes how to generate the firmware information using the information template and FwSign tool in the NuBL2 project. The NuBL32 firmware information can be generated with the same flow in the NuBL32 project.

1. Add FwInfo.c to the NuBL2 project to allocate a firmware information region.

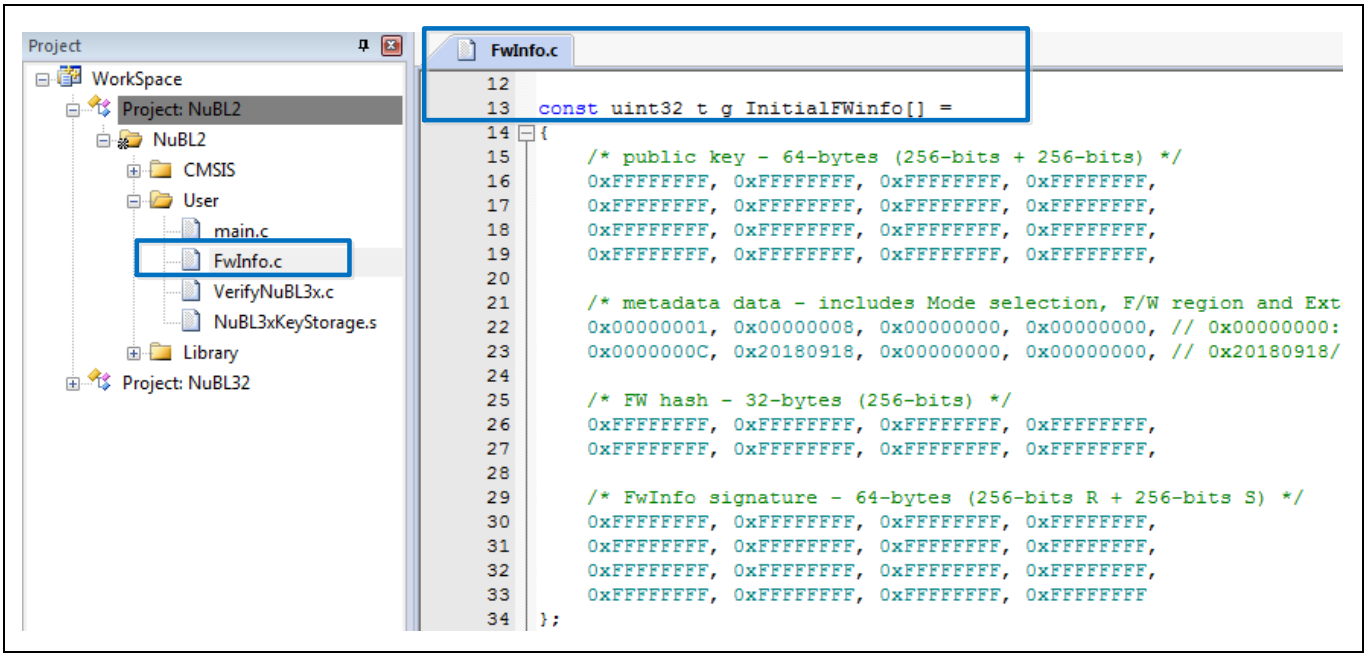

Figure 5-7 FwInfo.c in NuBL2 Project

2. Add a scatter file to configure the NuBL2 firmware information output file and its address as shown in Figure 5-8.

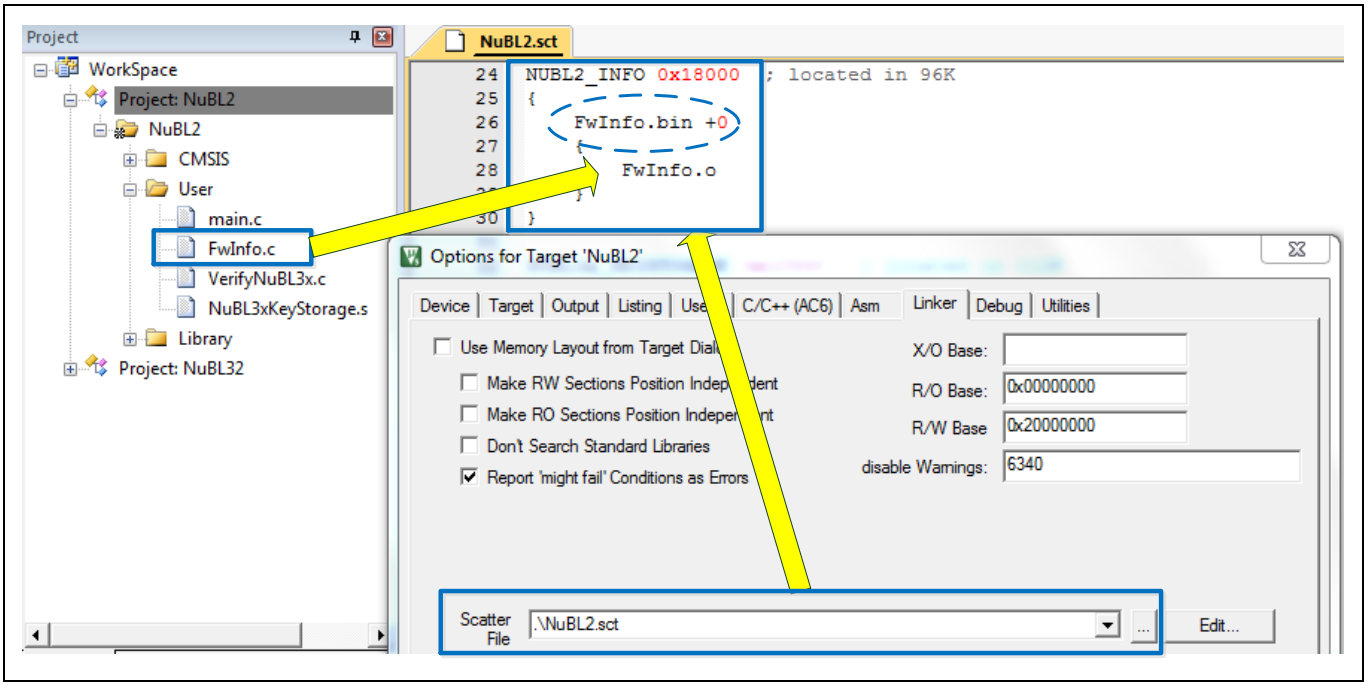

Figure 5-8 Configure NuBL2 Firmware Information

3. Open “After Build/Rebuild” in the “Options” and configure the parameters of FwSign for generating the NuBL2 firmware information as shown in Figure 5-9.

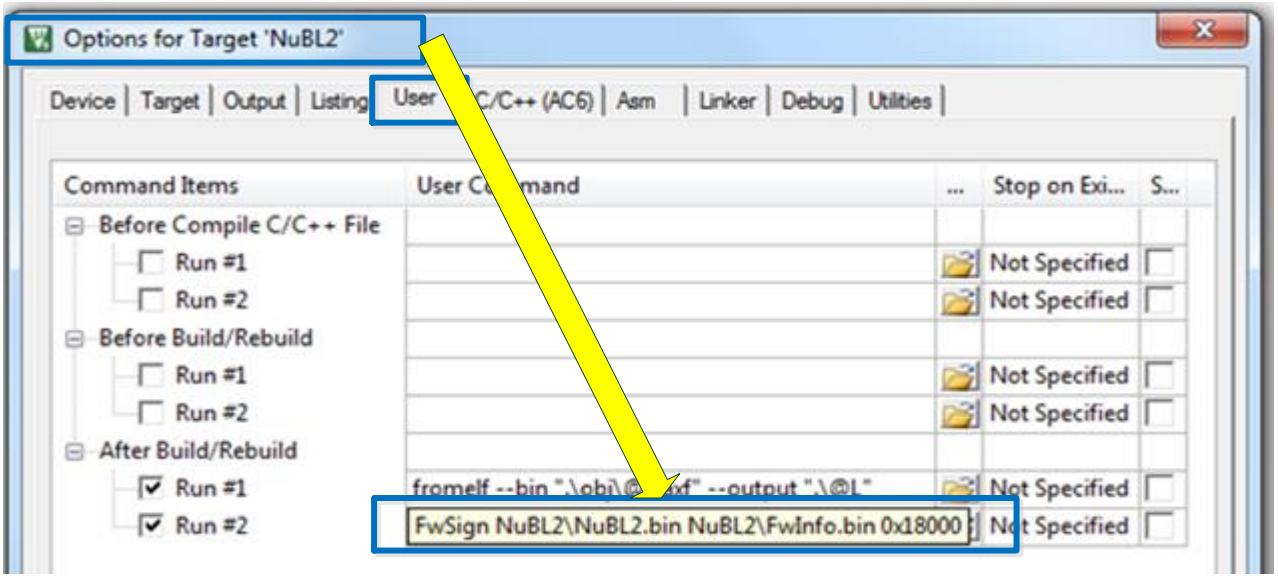

Figure 5-9 Configure FwSign.exe in NuBL2 Project

- The required files, FwSign.exe and FwSign.ini could be found at "\SecureBootDemo\NuBL2\Keil", are for generating the target NuBL2 firmware information.
- Figure 5-10 shows an example of the initial file FwSign.ini of FwSign.exe.

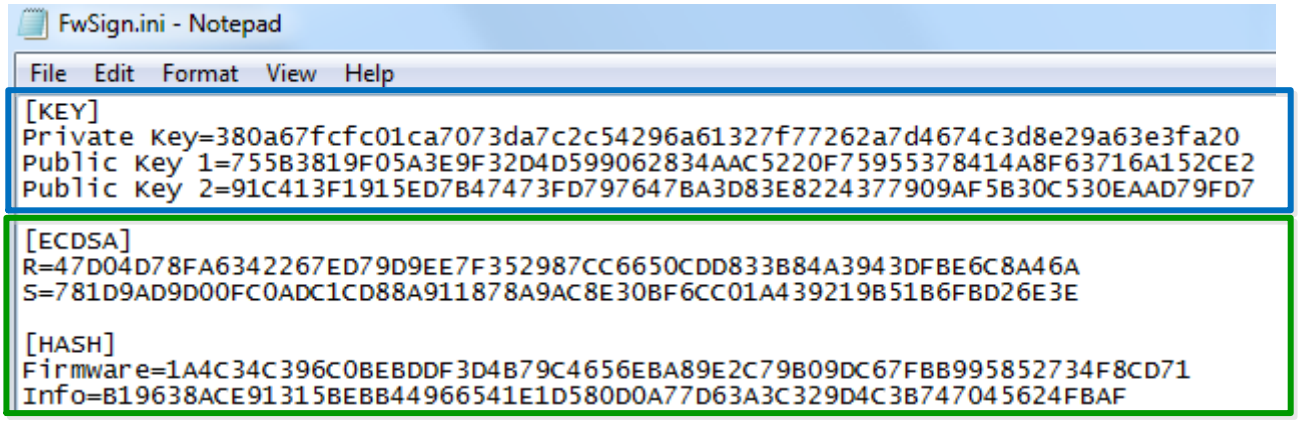

Figure 5-10 FwSign.ini in NuBL2 Project

- "KEY" field: stores the ECC private/public key pair for generating firmware information ECDSA Signature
- "ECDSA" field: stores the actual ECDSA Signature after successfully building the target project
- "HASH" field: stores the actual firmware and information hash after successfully building the NuBL2 project

- Output the target NuBL2 firmware information

After successfully building the NuBL2 project, FwInfo.bin and FwInfo.hex will be generated in the “\SecureBootDemo\NuBL2\Keil\NuBL2\” folder.

- Figure 5-11 shows the output FwInfo.bin

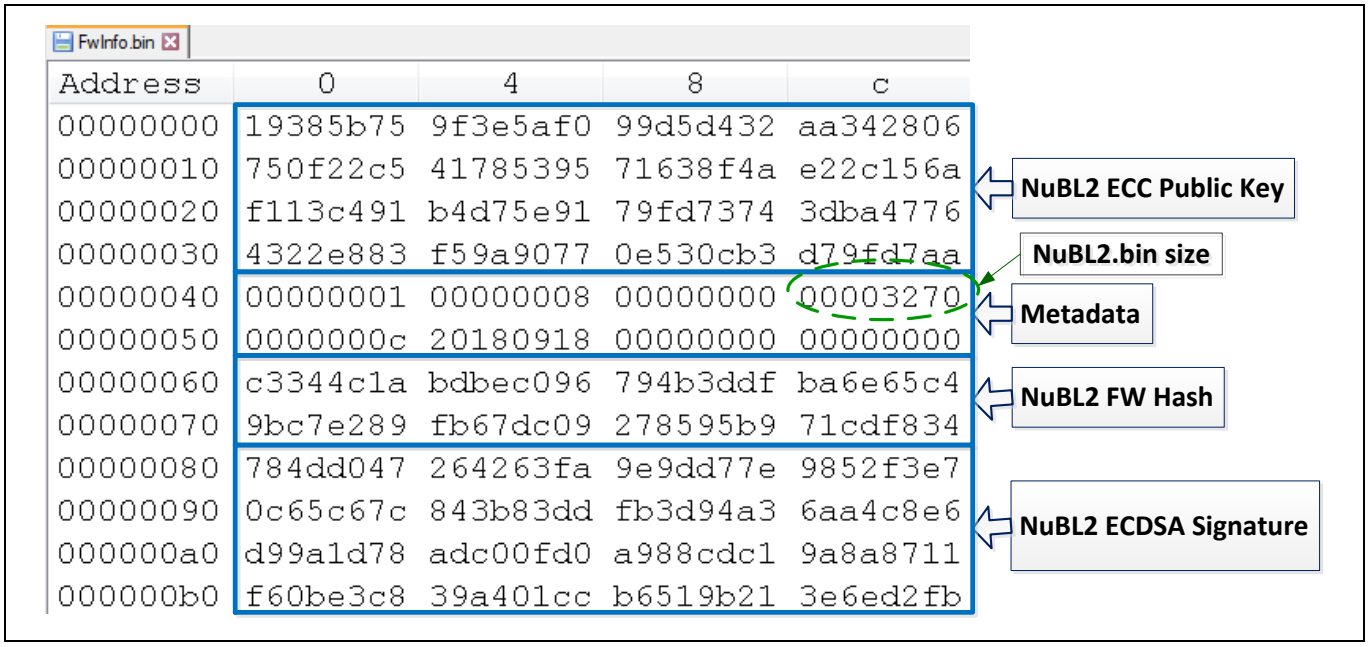

Figure 5-11 Target NuBL2 Firmware Information

- FwInfo.hex is also a NuBL2 firmware information, but output as Intel HEX format. Developer can use Load_NuBL2FwInfo_NuBL3x.ini to download the firmware information at Flash 0x18000.

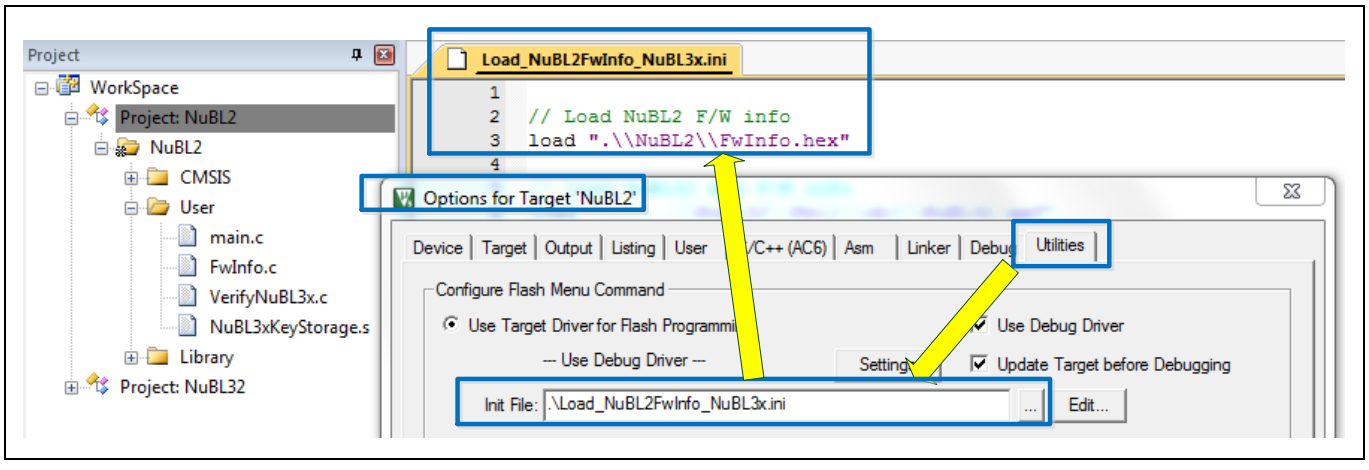

Figure 5-12 Load FwInfo.hxe in NuBL2 Project

The secure boot procedure also supports to identify CHIP PID, UID or UCID information that can be enclosed in the firmware information hash for ECDSA authentication. Figure 5-13 shows the example to add all ID information in FwSign.ini for secure boot verification. In addition, after the target project has successfully built, the first word of the Metadata will be updated to 0x1D in FwInfo.bin.

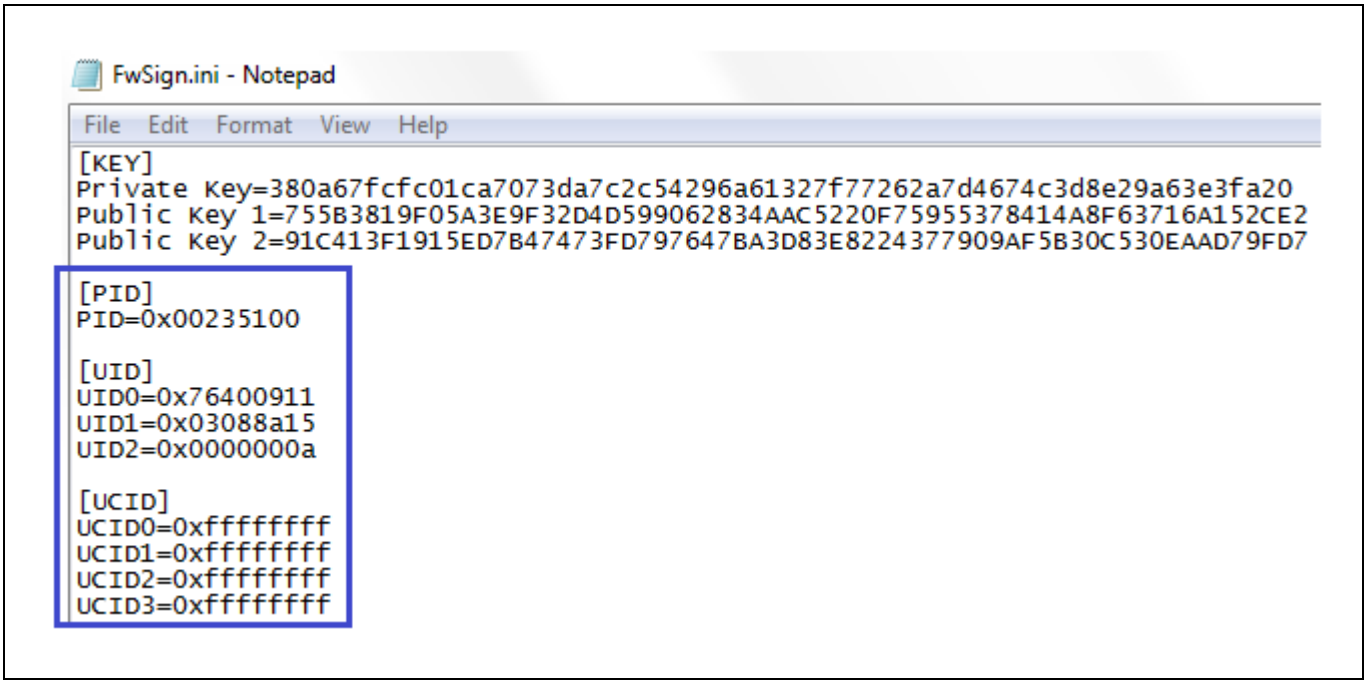

Figure 5-13 PID, UID and UCID Example in FwInfo.ini

5.2 NuBL2 Marker

The NuBL2 Marker is a consecutive 16-byte data placed at the NuBL2 execution address (must be page-size aligned) plus offset 0x10, it records a Flash address for storing NuBL2 firmware information.

The Secure Bootloader (NuBL1) will search the valid NuBL2 Marker first, and then obtain the NuBL2 firmware information to start the Secure Boot verification to NuBL2.

5.2.1 NuBL2 Marker Structure

This section describes the field and function of the NuBL2 Marker.

- Address and description

NuBL2 execution address (FW_ADDR)	Description
+ 0x10	Magic Word (0x4C42754E)
+ 0x14	NuBL2 FW INFO address
+ 0x18	Reserved (0x0)
+ 0x1C	Checksum

Table 5-2 NuBL2 Marker Structure

- Magic Word

- 0x4C42754E, it is the ASCII code of “NuBL”
- NuBL1 will search the Magic Word page by page from the bottom of LDROM, if not found, then search from the bottom of APROM

- NuBL2 FW INFO address
  - Record a Flash address for storing NuBL2 firmware information
- Checksum
  - The source data are “Magic Word”, “NuBL2 FW INFO address” and “Reserved”, and the calculation formula as shown follows,
    ((~M32(FW_ADDR+0x10)) + M32(FW_ADDR+0x14) + M32(FW_ADDR+0x18)) + 1
  - NuBL1 can obtain this checksum to check whether the valid NuBL2 Marker present

5.2.2 NuBL2 Marker Generation

The address and contents of the NuBL2 Marker must be allocated in the reserved area of NuBL2_startup.s.

Figure 5-14 shows the NuBL2 Marker in the NuBL2_startup.s, including “Magic Word”, “NuBL2 FW INFO address” (g_InitialFWinfo is declared in FwInfo.c) and “Checksum”.

NuBL2_startup.s

```
51                EXPORT  __Vectors
52                EXPORT  __Vectors_End
53                EXPORT  __Vectors_Size
54                IMPORT  SendChar_ToUART
55                IMPORT  g_InitialFWinfo
56
57  __Vectors     DCD     __initial_sp              ;     Top of Stack
58                DCD     Reset_Handler             ;     Reset Handler
59                DCD     NMI_Handler               ;     NMI Handler
60                DCD     HardFault_Handler         ;     Hard Fault Handler
61                ; The following four words data are NuBL2 Marker
62                DCD     0x4C42754E                      ; 0x10: signature for NuBL1 to find the valid FwInfo
63                DCD     g_InitialFWinfo                 ; 0x14: FwInfo base address
64                DCD     0                               ; Reserved
65                DCD     (((~0x4C42754E)+g_InitialFWinfo+0))+1 ; 0x1C: checksum = ((~M32(0x10))+M32(0x14)+M32(0x18))+1
66                DCD     0                         ;     Reserved
67                DCD     0                         ;     Reserved
68                DCD     0                         ;     Reserved
69                DCD     SVC_Handler               ;     SVCall Handler
70                DCD     0                         ;     Reserved
71                DCD     0                         ;     Reserved
72                DCD     PendSV_Handler            ;     PendSV Handler
73                DCD     SysTick_Handler           ;     SysTick Handler
```

Figure 5-14 NuBL2 Marker in NuBL2_startup.s

Figure 5-15 shows the target NuBL2 Marker in NuBL2 Firmware after successfully building the NuBL2 project.

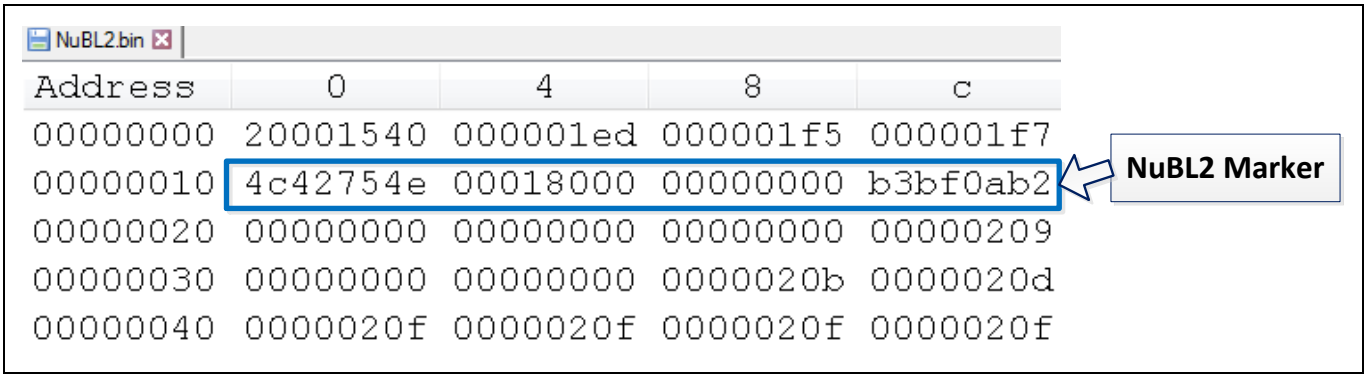

Figure 5-15 NuBL2 Marker in NuBL2 Firmware

5.3 NuBL2 Verification Functions

The VerifyNuBL3x.c is the source file that provides a set of verification functions in the NuBL2 project.

NuBL2 can call VerifyNuBL3x() API directly to perform NuBL32 identification, authentication and firmware integrity.

```
    ……
    /* Verify NuBL32 identity and F/W integrity */
    memcpy((void *)&g_NuBL3xFwInfo, (void *)NUBL32_FW_INFO_BASE, sizeof(FW_INFO_T));
    if(VerifyNuBL3x((uint32_t *)&g_NuBL3xFwInfo, NUBL32_FW_INFO_BASE) == -1)
    {
        printf("\n\nNuBL2 verifies NuBL32 FAIL.\n");
        while(1) {}
    }
    else
    {
        u32NuBL32Base = g_NuBL3xFwInfo.mData.au32FwRegion[0].u32Start;
        printf("\nNuBL2 identify NuBL32 public key and verify NuBL32 F/W integrity
PASS.\n");
    }
    ……
```

All the verification functions in VerifyNuBL3x.c could be configured as XOM code in the XOM region at address 0x10000 as shown in the following example code and Figure 5-16. This prevents the verification function in VerifyNuBL3x.c from being replaced or the secret/sensitive data being stolen. Even if ICE debug mode is entered, the source code and procedure in XOM region cannot be traced.

```
#define ENABLE_XOM0_REGION  (1) // Set 1 to configure VerifyNuBL3x.c code in XOM0 region,
and cannot trace VerifyNuBL3x.c flow in ICE debug mode
    ……
#if (ENABLE_XOM0_REGION == 1)
    /* Enable XOM0, and all the functions in VerifyNuBL3x.c cannot trace in ICE debug mode
    EnableXOM0();
#endif
    ……
```

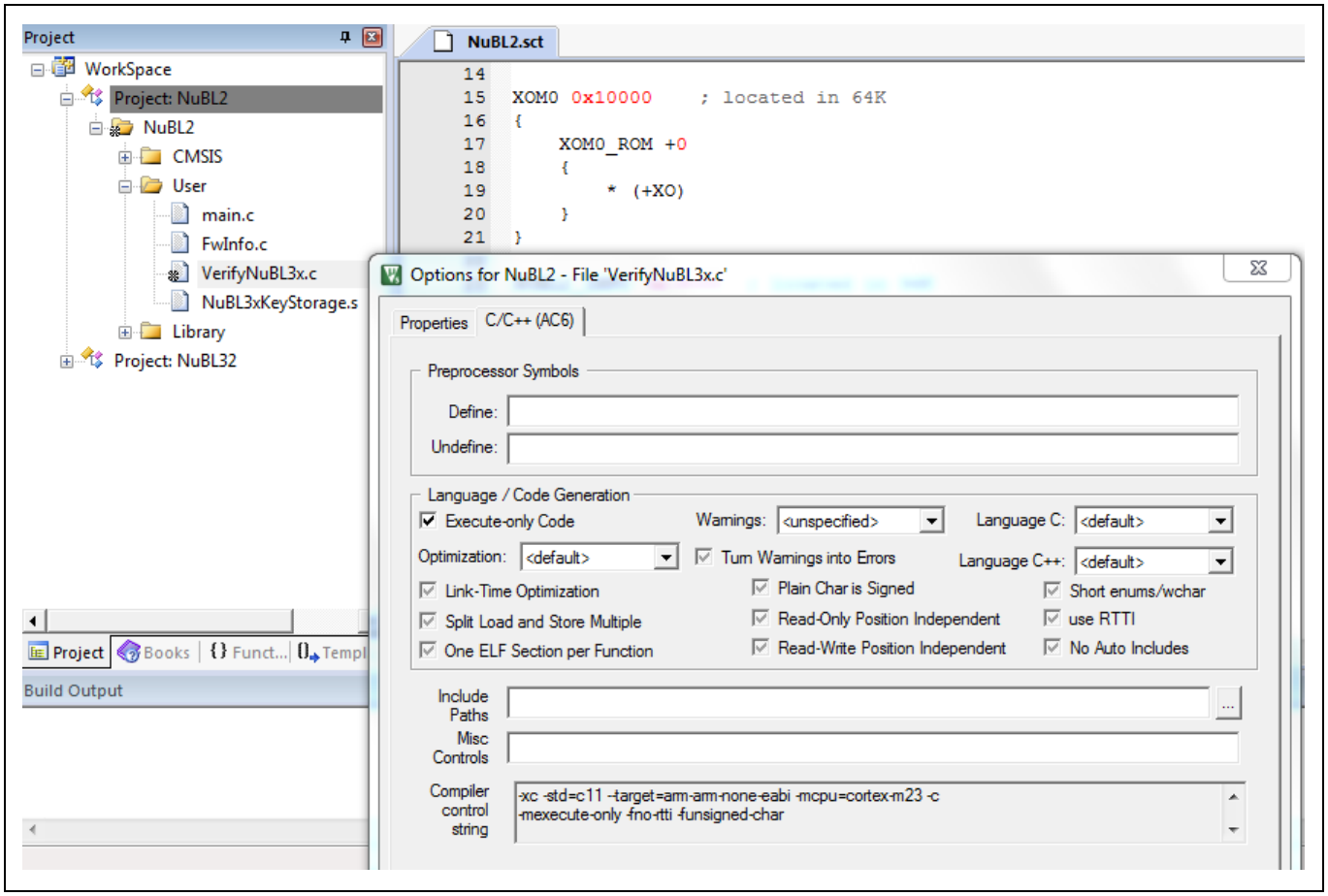

Figure 5-16 Configure VerifyNuBL3x.c in XOM

5.4 NuBL32 Public Key Storage

This section describes how to use CryptoTool to generate an encrypted NuBL32 Public Key Storage and its SHA-256 digest.

5.4.1 Encrypted Public Key

This section demonstrates how to generate an encrypted NuBL32 public key for storing in the Public Key Storage.

1. Prepare a NuBL32 public key raw data, NuBL32Pub_Raw.bin

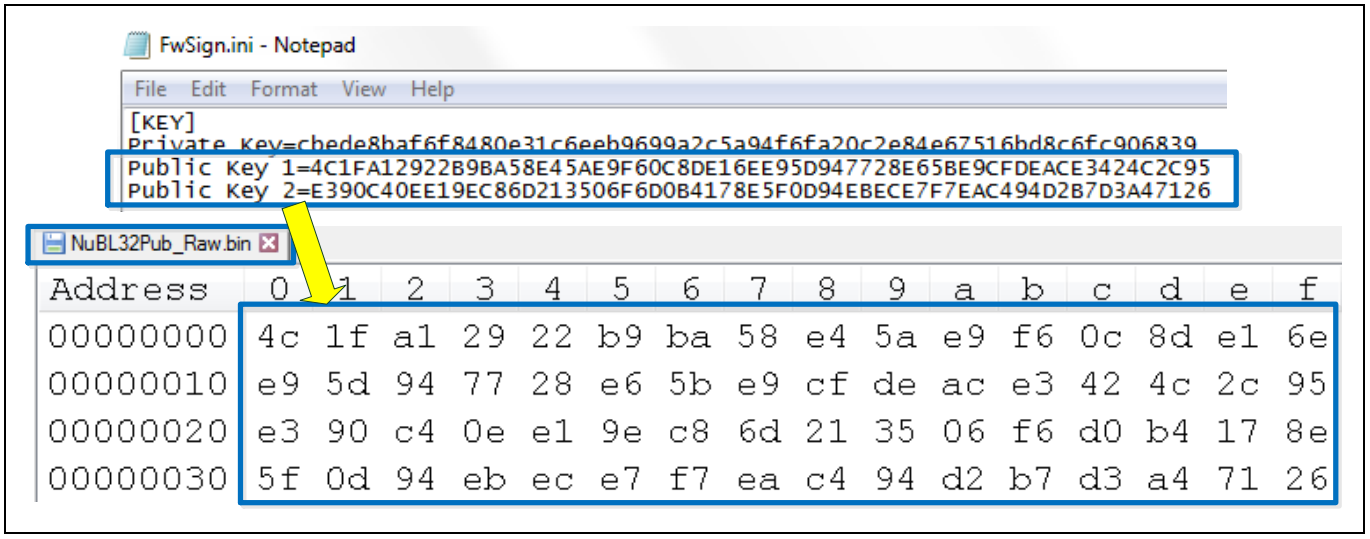

Figure 5-17 NuBL32 ECC Public Key Raw Data

2. Use CryptoTool to output an encrypted public key file

   Input AES Key, Initialization Vector and NuBL32Pub_Raw.bin to generate and output the encrypted key file, NuBL32PubKeyEncrypted.bin, as shown in Figure 5-18.

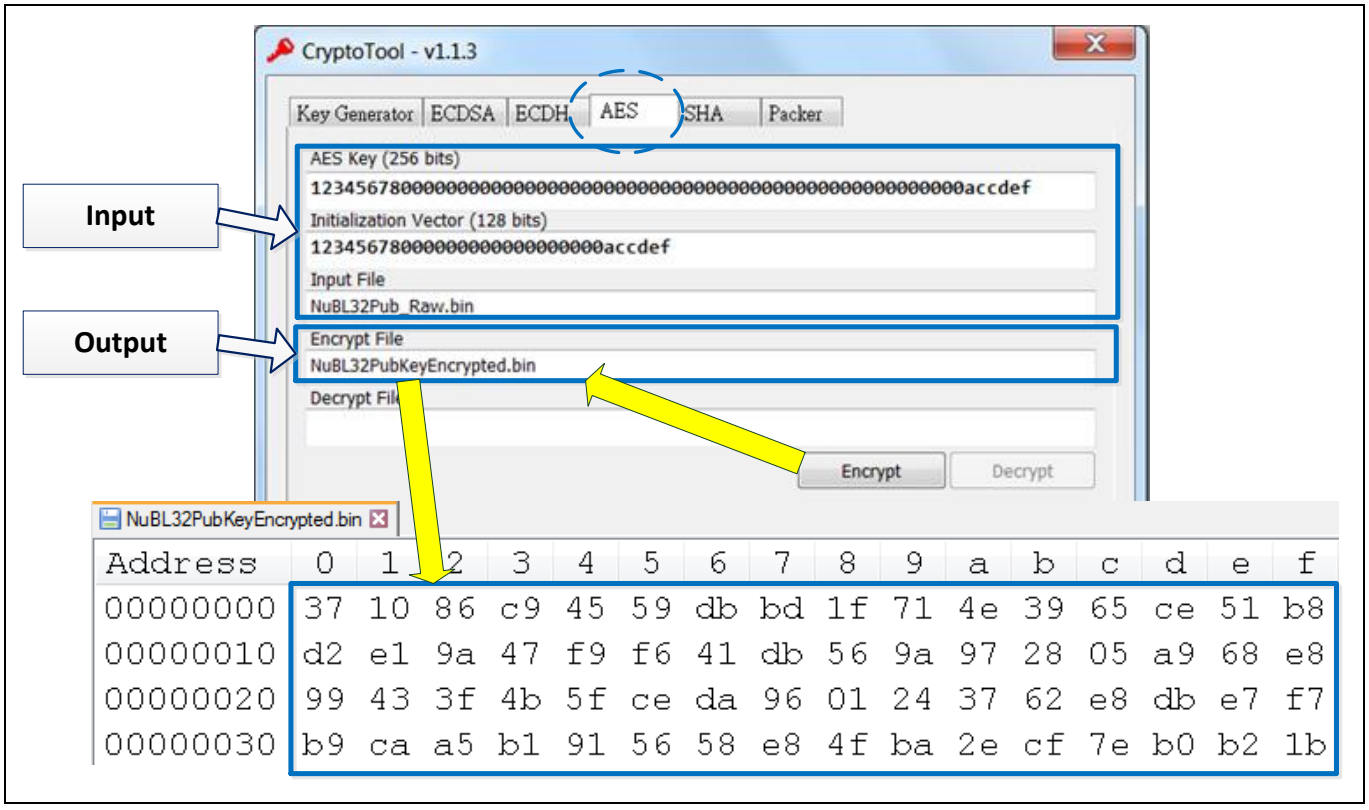

Figure 5-18 Encrypted NuBL32 ECC Public Key Raw Data

5.4.2 Encrypted Public Key Hash

The encrypted public key hash is a SHA-256 digest of NuBL32PubKeyEncrypted.bin (encrypted NuBL32 ECC public key), and used for NuBL2 can verify the Public Key Storage integrity before decrypting it.

Figure 5-19 shows using the CryptoTool to generate an encrypted public key hash.

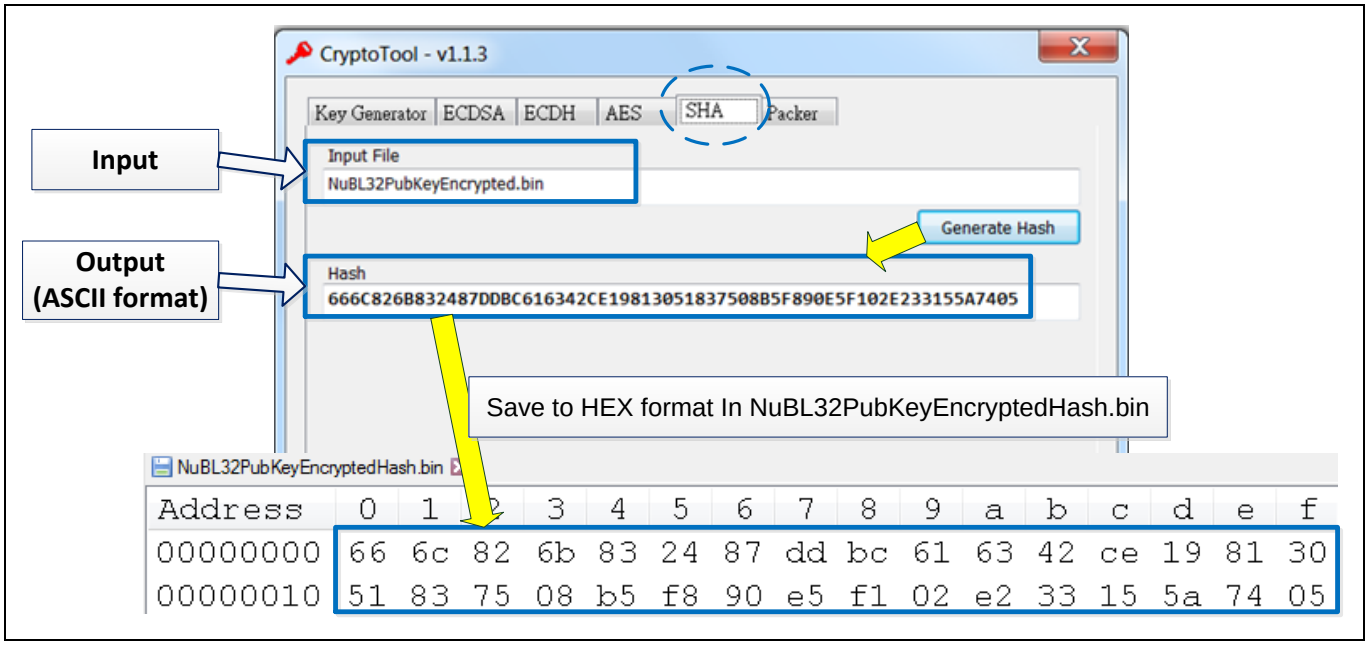

Figure 5-19 Encrypted NuBL32 ECC Public Key Hash

5.4.3 Allocate Key Storage

This section demonstrates how to allocate a Flash region to store the Public Key Storage, including encrypted NuBL32 ECC public key and its SHA-256 hash.

1. Add a NuBL3xKeyStorage.s shown in Figure 5-20 for including all NuBL32 key files in the NuBL2 project. All the related files could be found at "\SampleCode\MKROM\SecureBootDemo\NuBL2\KeyInfo\".

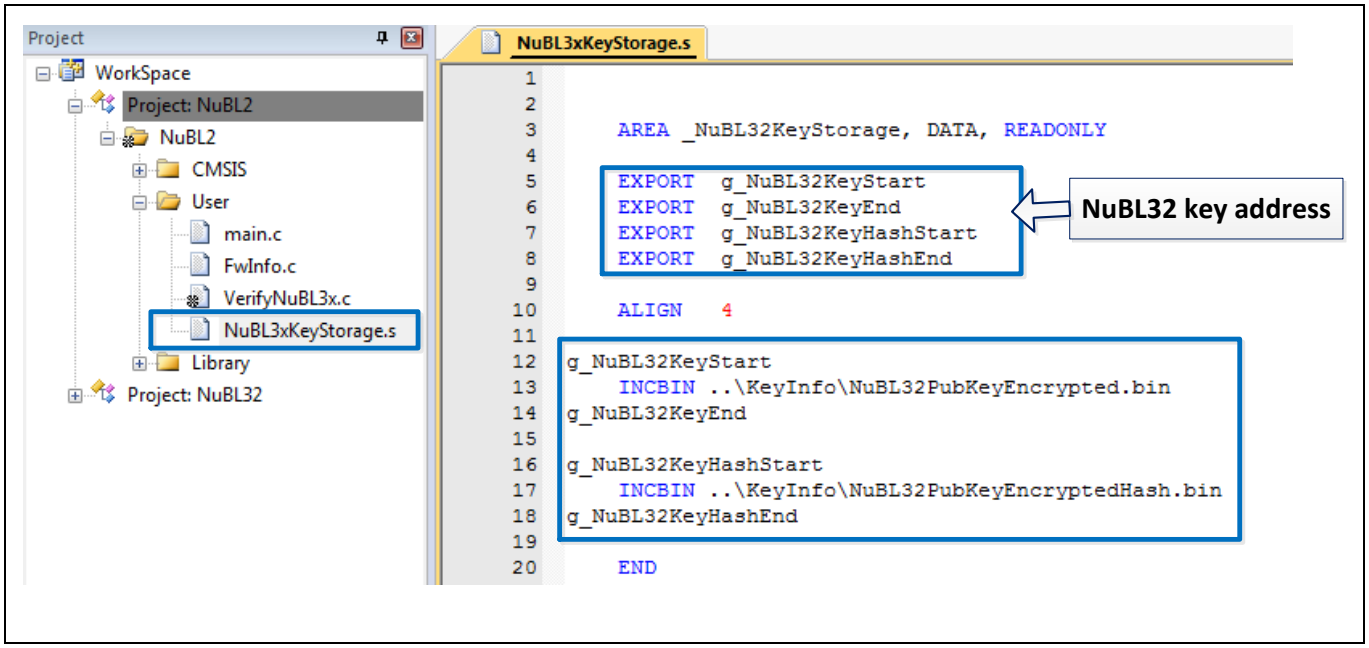

Figure 5-20 Include NuBL32 Key Files

2. Modify NuBL2.sct to allocate the key storage region at 0x1C000 as shown in Figure 5-21.

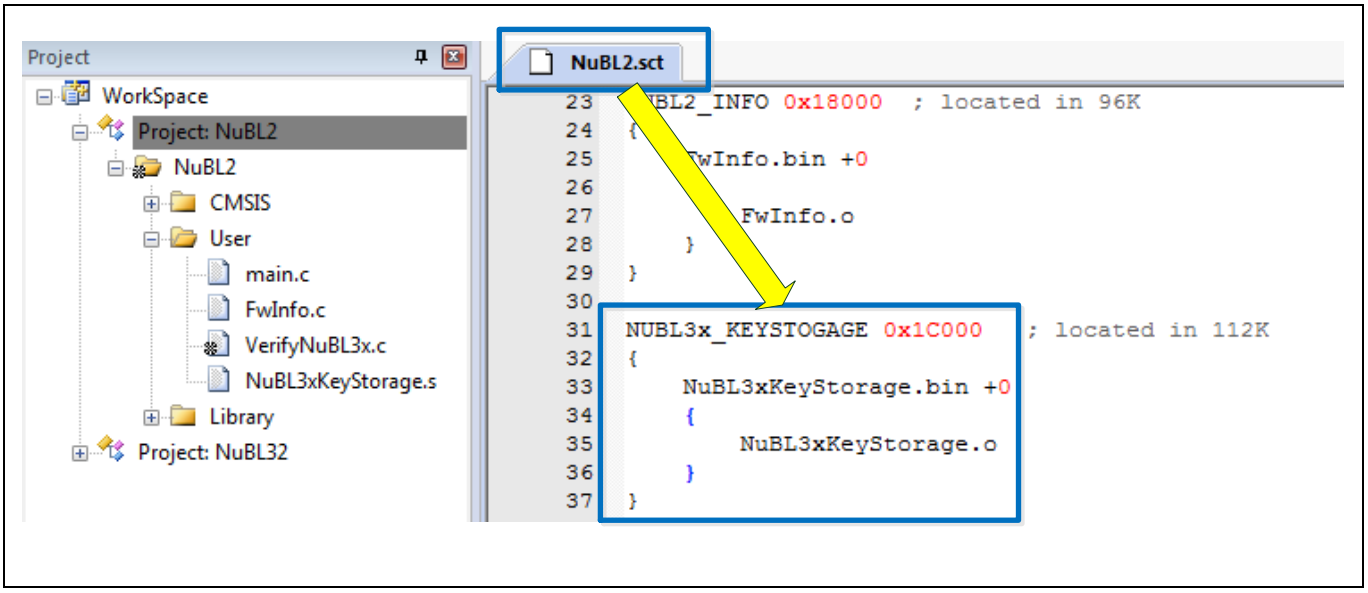

Figure 5-21 Allocate NuBL32 Key Storage Region

5.4.4 AES-256 Key

An AES-256 key and IV (Initialization Vector) are declared in the NuBL2 project for decrypting the Public Key Storage to obtain the NuBL32 ECC public key.

AES key:

12345678000000000000000000000000000000000000000000000000accdef

IV key:

1234567800000000000000000accdef

The functions for AES decryption below are also provided in the VerifyNuBL3x.c. NuBL2 can use the internal AES and IV through these functions for decrypting Public Key Storage to obtain the NuBL32 ECC public key at runtime.

```
static uint32_t *SetAES256Key(uint32_t *key)
{
// AES: 12345678000000000000000000000000000000000000000000000000accdef
    key[0] = 0x12345678;
    key[1] = 0x00000000;
    key[2] = 0x00000000;
    key[3] = 0x00000000;
    key[4] = 0x00000000;
    key[5] = 0x00000000;
    key[6] = 0x00000000;
    key[7] = 0x00accdef;
```

```
    return key;
}

static uint32_t *SetAESIV(uint32_t *iv)
{
// IV: 12345678000000000000000000accdef
    iv[0] = 0x12345678;
    iv[1] = 0x00000000;
    iv[2] = 0x00000000;
    iv[3] = 0x00accdef;

    return iv;
}
```

5.5 Download Firmware and Information

After successfully building the NuBL2 and NuBL32 project, developer can click the “Load” button in NuBL2 project to program the NuBL2 firmware into Flash. In addition, according to the Load_NuBL2FwInfo_NuBL3x.ini file in NuBL2 project, the NuBL2 firmware information, NuBL32 firmware and NuBL32 firmware information could be programmed to the specific Flash regions.

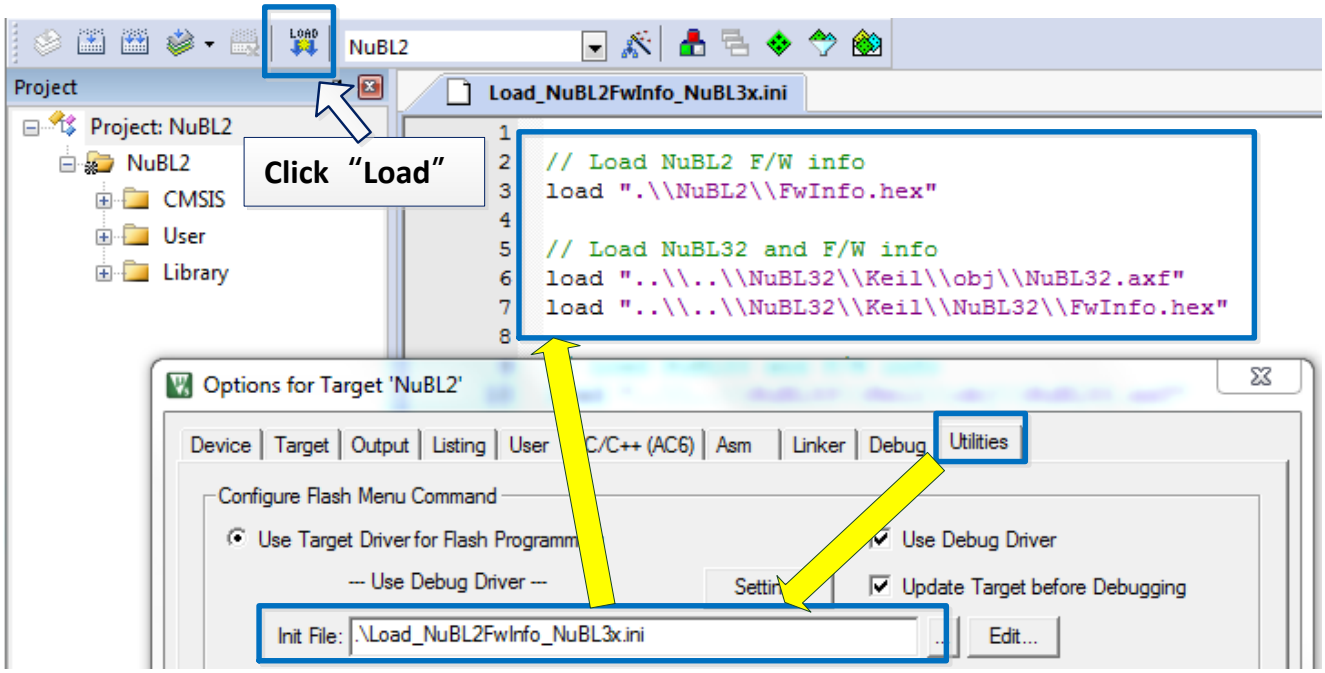

Figure 5-22 Load Firmware and Information

Note that MCU will execute NuBL2 (APROM code) directly not start booting from M261 Secure Bootloader after “Load” operation is completed.

Developer can configure booting form Secure Bootloader at KEIL IDE environment as shown in Figure 5-23, or configure it in NuBL2 execution by set SET_SECURE_BOOT 1 in NuBL2 code as shown below.

```
#define SET_SECURE_BOOT     (1) // Set 1 to support modify CFG0[5] MBS 0 for booting from Secure Bootloader
```

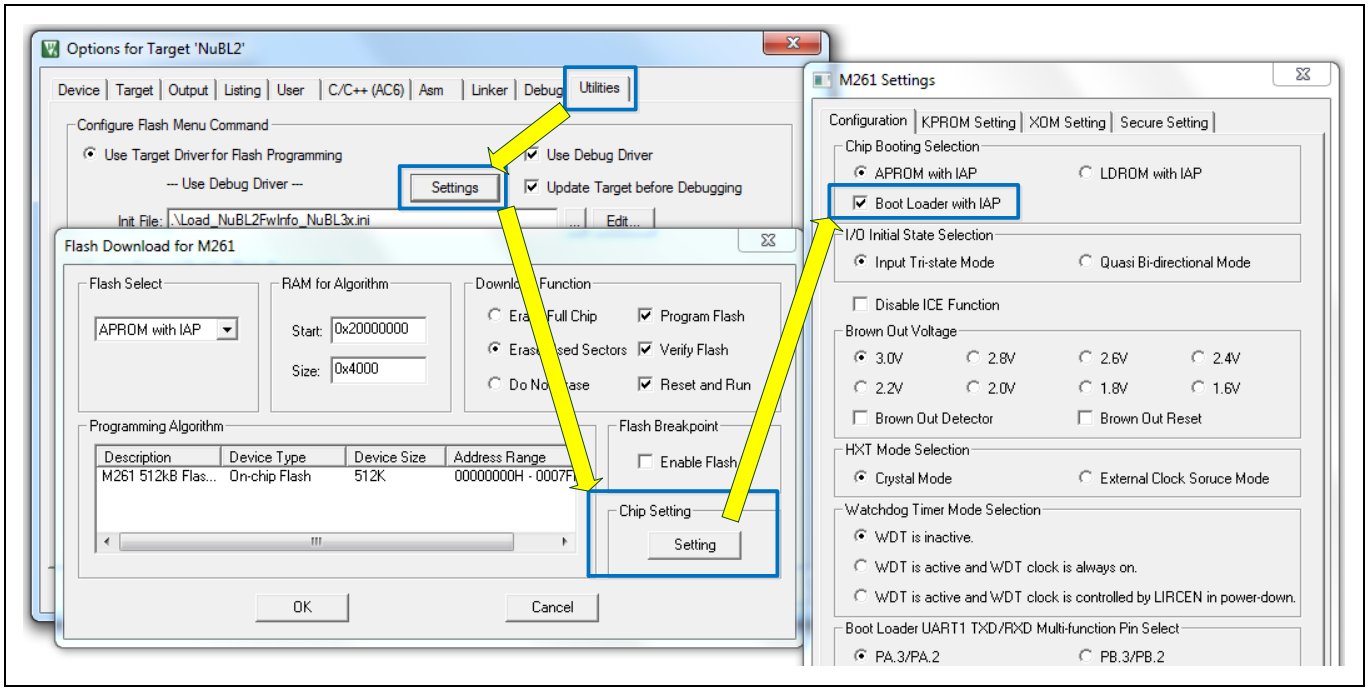

Figure 5-23 Configure Secure Boot in KEIL IDE Environment

6 Conclusion

Through the Secure Boot verification mechanism in M261 Secure Bootloader, a system developer can create a trusted execution system launched form Secure Bootloader. All software requires authentication and integrity checks before executes on the M261 series MCU. It prevents malicious code from stealing secret/sensitive data and replacing authenticated code.

The SecureBootDemo sample code provides templates for the system developer to create a trusted execution system, including a trusted boot code (NuBL2) and an application code (NuBL32).

Revision History

Date	Revision	Description
2019.04.08	1.00	1. Initially issued.
2019.08.13	1.01	1. Revised ID information description and an example usage of firmware information ECDSA in section 5.1.2 and 5.1.3. 2. Described the limitation of extend information size in firmware information in section 5.1.2. 3. Added the XOM usage and limitation of NuBL2 project in section 5.3. 4. Described how to enable the secure boot after the firmware and firmware information has been updated in section 5.5.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.