M467 Secure ISP with Key Store

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code is a security application for the M460_ISP_HID. After downloading encrypted firmware via the ISP Programming Tool, the M467 uses the AES crypto engine and Key Store to decrypt the firmware content and perform updates to the APROM, thereby completing a secure download process.
BSP Version	M460 Series BSP CMSIS V3.00.002
Hardware	NuMaker-M467HJ V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

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

www.nuvoton.com

1. Overview

This example code allows users to perform firmware updates via the USB_HID interface. Users can first encrypt the firmware using Nuvoton's CryptoTool, then use the Nuvoton NuMicro ISP Programming Tool to write the encrypted firmware to the NuMaker-M467HJ. Upon successful reception, the M467 will invoke the decryption API within the XOM, leveraging hardware encryption accelerators and a 256-bit AES key stored in the Key Store Flash to decrypt the encrypted firmware. Subsequently, the firmware will be written to APROM to complete the entire update process.

1.1 AES

The firmware encryption employs the Advanced Encryption Standard (AES), which is a symmetric encryption algorithm. It uses the same key for both encryption and decryption, with key lengths of 128, 192, or 256 bits. The data to be encrypted is divided into fixed-size blocks for encryption. Its advantages include high security and fast processing speed. AES is a U.S. National Institute of Standards and Technology (NIST) publication and is considered a national standard.

NIST offers various modes for the AES algorithm, and the Nuvoton CryptoTool utilizes AES-CFB-256 mode. Before performing file encryption or decryption, a 256-bit key and a 128-bit Initialization Vector (IV) need to be configured. Additionally, the location of the file to be encrypted, as well as the path and filename for the encrypted file need to be specified. Here's an example for reference:

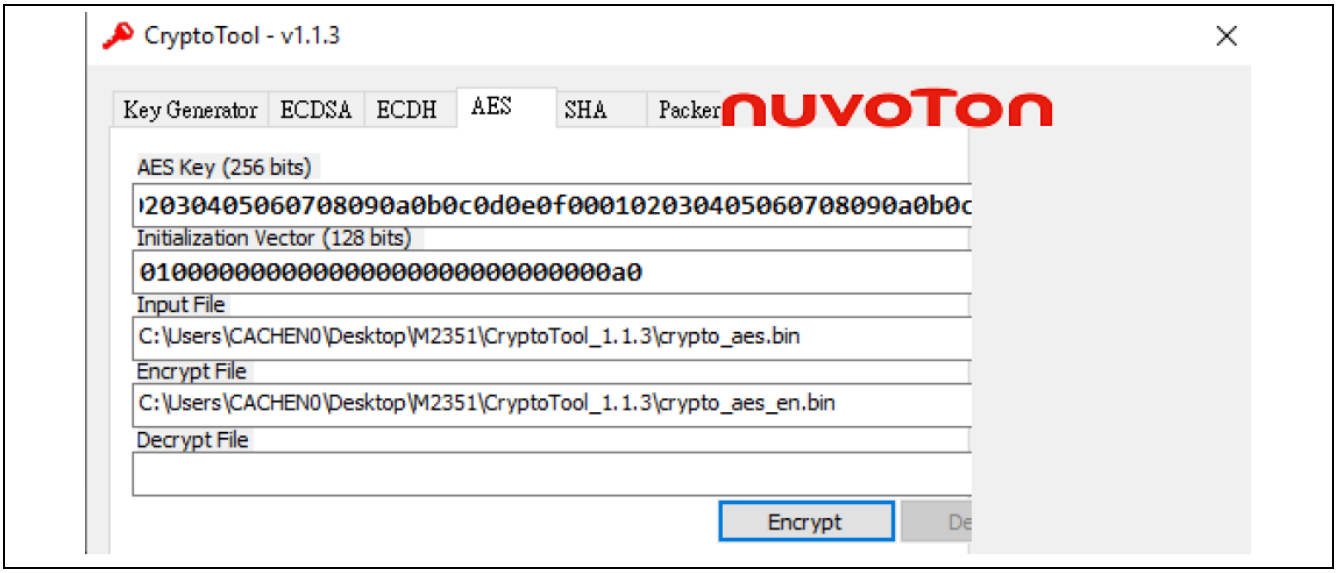


Figure 1-1 Nuvoton CryptoTool -v1.1.3

1.2 Execution Only Memory (XOM)

The M467 can configure its internal APROM as independent XOM regions. The instructions (code) stored in these regions cannot be read but only executed, and up to four independent XOM regions can be configured. To prevent accidental data access and leakage of sensitive software information, developers can position confidential code within these regions to protect it. For detailed information, please refer to section 6.5.4.6 of the M463/M467 Series Technical Reference Manual. In the example code, the API used for decrypting firmware and the IV used for AES encryption/decryption will be placed within this area.

1.3 Key Store

The M467 Key Store serves as a secure key management device, providing a dedicated memory space for secure storage. Keys or critical data can be stored in SRAM, Flash, or OTP. During the key written process, it's possible to specify that a key can only be used with specific encryption algorithms. For example, if Key 1 stored in Flash is designated for use with AES, then the CPU and other hardware encryption modules such as ECC or RSA won't be able to access that key. This enhances the security of key protection. For detailed information on security features and usage instructions, please refer to section 6.42 of the M463/M467 Series Technical Reference Manual.

1.4 Software Architecture

1.4.1 Key Store Test

The software flow is divided into two parts for explanation. The first part is the Key Store writing test, as shown in Figure 1-2. Using the NuMaker-M467HJ board connected to Nu-Link2-me on the testing computer, when the reset button is pressed, the program starts running. Initially, the Key Store is initialized. Then, you'll be prompted whether to clear the Key Store Flash. If it's the first time using the evaluation board, there's no need to clear it, and you'll see "KS FLASH remained size = 4060" displayed on the terminal. The next step on the terminal will ask if you want to write the key to Flash. This step is necessary for the first-time use. After completing this step, the keys written to the Key Store Flash will undergo AES CFB mode encryption/decryption testing, and the results will be displayed on the terminal.

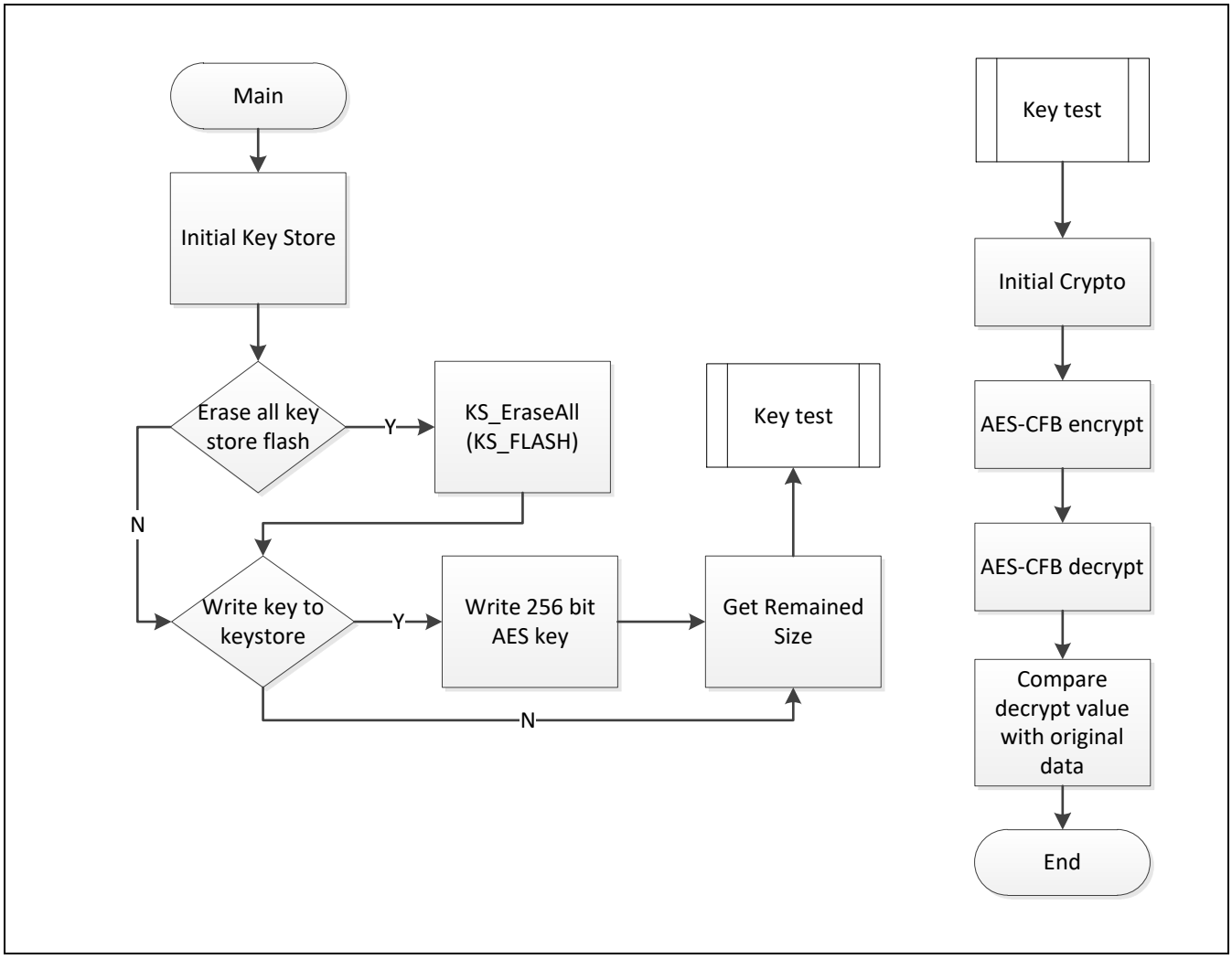


Figure 1-2 Key Test Flow Chart

1.4.2 Secure ISP Update Flow

The second part is the main update process, which is divided into two sections. In the first section, USB is initialized, and the ISP function is enabled. Then, the program detects the input level of PC.13 to determine whether to enter the ISP update process. If PC.13 is low, the update program will begin downloading. If it's high, the program will jump to execute APROM.

Once the download update is complete, the main program calls the API inside the XOM to execute the decryption and writing into APROM. Firstly, the encryption engine is initialized. Then, AES keys, initialization vectors, and DMA addresses are configured. Subsequently, the encryption engine begins decrypting the code, and finally writes it into APROM.

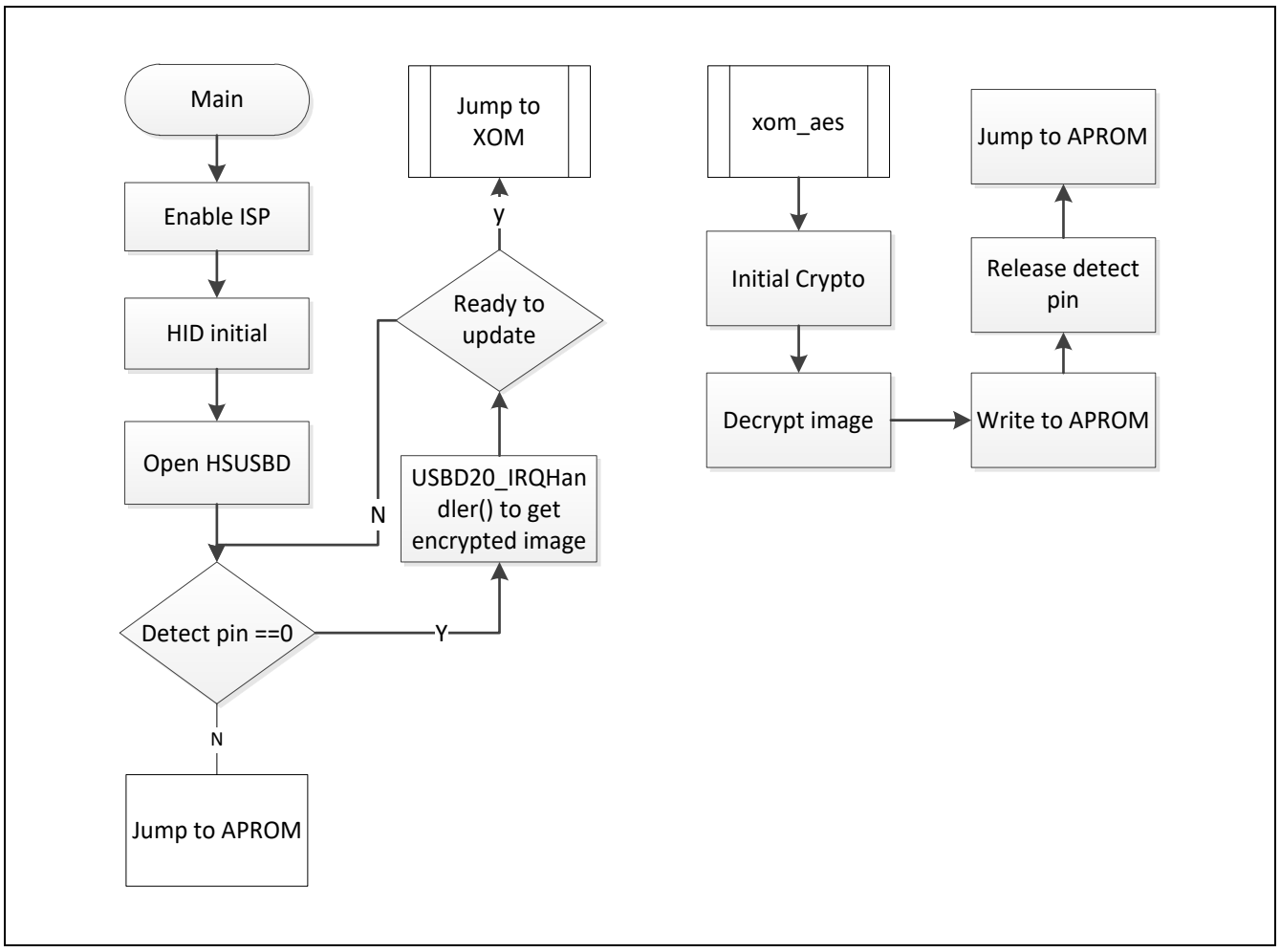


Figure 1-3 Secure ISP Flow Chart

1.5 Function Description

1. Connect the NuMaker-M467HJ to the PC. Open the Keil M467_KS_KeyTest project. After compiling, download it to the evaluation board. Execute the AES key writing test.

```
KS FLASH remaind size = 4060
Erase all keystore flash?(y/n)
KS FLASH remaind size = 4060
write key to KeyStore?(y/n)
KS FLASH remind size: 4024
The key in key store:
key number is: 1
0x0000  03 02 01 00 07 06 05 04 0b 0a 09 08 0f 0e 0d 0c   ................
0x0010  03 02 01 00 07 06 05 04 0b 0a 09 08 0e 0e 0d 0c   ................

The input data:
0x0000  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff   .."3DUfw........
0x0010  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff   .."3DUfw........

AES encrypt done. The output data:
0x0000  ed 62 a0 fe f1 da e1 41 cc a7 ba 2b 91 9d 19 49   .b.....A...+...I
0x0010  a1 77 bd 71 7b 7f e1 53 65 bf f7 64 11 7e d1 1e   .w.q{..Se..d.~..

AES decrypt done. Decode Data:
0x0000  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff   .."3DUfw........
0x0010  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff   .."3DUfw........

Done!
```

Figure 1-4 AES Key Test Flow

2. Use the ISP Programming Tool to download the encrypted image. The code inside XOM will decrypt the image and write it into APROM, then jump to execute the code in APROM.

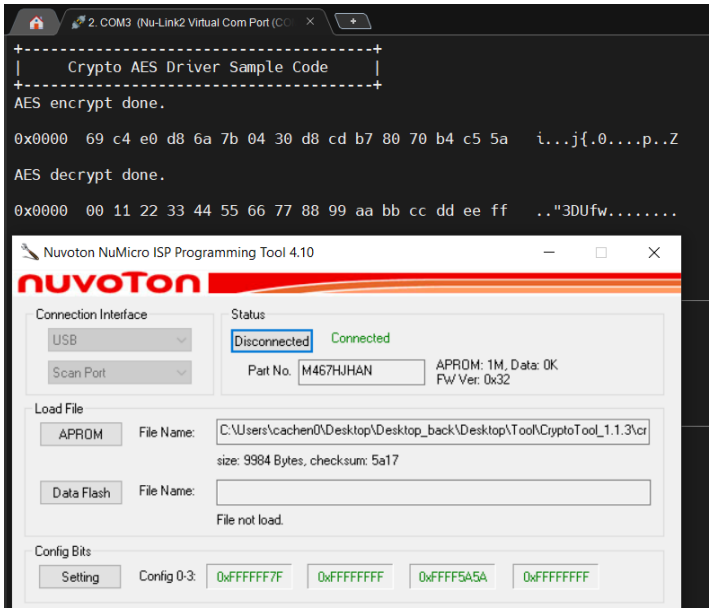


Figure 1-5 ISP Programming Tool

2. Code Description

2.1 Key Store Test

The code below meets the architecture in section 1.4.1. The main task is to implement the functional testing of the key. The crucial AES key is set through the au32Key array. After becoming familiar with the operation, users can replace it with the actual key used in the product and store it securely. It is recommended to set the AES key attribute as CPU-unreadable to ensure that the key can only be accessed by the AES engine, thus achieving complete protection. After successfully writing the AES key, a key number (i32KeyIdx) is returned. When calling the AES engine with the Key Store for encryption and decryption, users need to input the correct key number to successfully restore the original data.

```
int main(void)
{
    int8_t i8Test = 0;
    int32_t i32KeyIdx = 1;
    /* au32Key value is written into Key Store Flash which is used to decrypt image.
    au32Key value is determined by the user and should be the same with CryptoTool. */
uint32_t au32Key[8] = {
        0x00010203, 0x04050607, 0x08090a0b, 0x0c0d0e0f,
        0x00010203, 0x04050607, 0x08090a0b, 0x0c0d0e0f
    };
    char     chr;

    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();
    /* Init UART for printf */
    UART_Init();
    /* Init Key Store  */
    KS_Open();

    printf("KS FLASH remaind size = %d\n", KS_GetRemainSize(KS_FLASH));

    printf("Erase all keystore flash?(y/n)\n");
    chr = getchar();
    if((chr == 'y') || (chr == 'Y')) {
        KS_EraseAll(KS_FLASH);
        printf("KS FLASH remaind size = %d\n", KS_GetRemainSize(KS_FLASH));
    }
    printf("write key to KeyStore?(y/n)\n");
    chr = getchar();
    /*---------------------------------------
    *  Key attribute: flash key
    *  Key meta data: AES-256 bits
    *  Reture Value :
    *  i32KeyIdx, the index value sould be set
    *  when using cryto encrypt/decrypt with
    *  key store flash key
    *---------------------------------------*/
    if((chr == 'y') || (chr == 'Y')) {
        i32KeyIdx = KS_Write(KS_FLASH, KS_META_256| KS_META_AES, au32Key);
    }
    if(i32KeyIdx < 0) {
        printf("Fail to write key to Key Store!\n");
        printf("KS FLASH remaind size = %d\n", KS_GetRemainSize(KS_FLASH));
        return -1;
    }
```

```
    printf("KS FLASH remind size: %d\n", KS_GetRemainSize(KS_FLASH));

    printf("The key in key store:\n");

    printf("key number is: %d\n", i32KeyIdx);
    DumpBuff((uint8_t *)&au32Key[0], 32);
    /* use AES_Test to check the encryption and decryption with Key Store */
    i8Test = AES_Test(CRPT, KS_FLASH, i32KeyIdx);
    if(i8Test == 0) {
        printf("Done!\n");
    } else {
        printf("fail!\n");
    }
    while(1) {}
}
```

2.2 ISP_HID_20_SecureUpdate

The code below meets the architecture in section 1.4.2. The first part involves writing the encrypted firmware into a pre-defined SRAM area (refer to *fw_image.c* and *scatter.sct)* through USB with ISP protocols. The second part, using the *void* UpdateRoutine(void), implements the details of secure updating and puts the whole functions into the XOM area to prevent hackers from accessing detailed process instructions. The Secure ISP program provides users with a basic example code of secure updating. Users can customize the UpdateRoutine according to their specific requirements, such as modifying the used KEYNUM, adjusting AES encryption/decryption modes and IV values, to enhance the updating mechanism tailored to their product.

```
int32_t main(void)
{
    /* Unlock write-protected registers */
    SYS_UnlockReg();

    /* Init System, peripheral clock and multi-function I/O */
    if(SYS_Init() < 0) {
        goto _APROM;
    }

    /* Enable ISP */
    CLK->AHBCLK0 |= CLK_AHBCLK0_ISPCKEN_Msk;

    FMC->ISPCTL |= FMC_ISPCTL_ISPEN_Msk | FMC_ISPCTL_APUEN_Msk | FMC_ISPCTL_ISPFF_Msk;

    /* Get APROM and Data Flash size */
    g_u32ApromSize = GetApromSize();
    GetDataFlashInfo(&g_u32DataFlashAddr, &g_u32DataFlashSize);
    if(DetectPin != 0) {
        goto _APROM;
    }

    /* Open HSUSBD controller */
    HSUSBD_Open(NULL, NULL, NULL);

    /* Endpoint configuration */
    HID_Init();

    /* Start transaction */
    HSUSBD->OPER = HSUSBD_OPER_HISPDEN_Msk;    /* high-speed */
    HSUSBD_CLR_SE0();
```

```
    g_u8ISPDone = FALSE;
    /* check PC.13 input status */
    while(DetectPin == 0) {
        /* Polling HSUSBD interrupt flag */
        USBD20_IRQHandler();

        if(g_u8UsbDataReady == TRUE) {
            ParseCmd((uint8_t *)g_u8UsbRcvBuff, 64);
            EPA_Handler();
            g_u8UsbDataReady = FALSE;
        }
        if(ISPDone == TRUE) {
            UpdateRoutine();
            g_u8ISPDone = FALSE;
        }
    }

_APROM:
    /* Reset system and boot from APROM */
    /*set vector to APROM "with IAP", only use witg IAP need to set vector */
    FMC->ISPCMD = FMC_ISPCMD_VECMAP;              /* ISP command */
    FMC->ISPADDR = 0x00000000;                    /* Vector remap address */
    FMC->ISPTRG = FMC_ISPTRG_ISPGO_Msk;           /* Trigger ISP command */
    while(FMC->ISPTRG & FMC_ISPTRG_ISPGO_Msk) ;   /* Wait for ISP command done. */
    /*mcu reset and start from APROM*/
    SYS->IPRST0 |= SYS_IPRST0_CPURST_Msk;
    /* Trap the CPU */
    while(1);
}

void UpdateRoutine(void)
{

    uint32_t StartAddress = 0x0000,  TotalLen;
    TotalLen = g_au32Encryptedimage[0];

    /*---------------------------------------
    *  step 1, decrypt image
    *  AES-256 CFB mode decrypt fw image
    *---------------------------------------*/
    SYS_UnlockReg();
    KS_Open();
    AES_Open(CRPT, 0, 0, AES_MODE_CFB, AES_KEY_SIZE_256, AES_IN_OUT_SWAP);
    /* Use key in key store */
    AES_SetKey_KS(CRPT, KS_FLASH, KEYNUM);
    AES_SetInitVect(CRPT, 0, (uint32_t*)au32MyAESIV);
    AES_SetDMATransfer(CRPT, 0, (uint32_t)&g_au32Encryptedimage[1],
(uint32_t)au8OutputData, TotalLen);
    AES_Start(CRPT, 0, CRYPTO_DMA_ONE_SHOT);
    while(((CRPT)->AES_STS & CRPT_AES_STS_BUSY_Msk));

    /* step 2, update APROM */
    FMC_ENABLE_AP_UPDATE();
    WriteData(StartAddress, StartAddress + TotalLen, (uint32_t *)au8OutputData);

    return;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M460 Series BSP CMSIS V3.00.002
- IDE version
  - Keil uVersion 5.39

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M467HJ V1.0
- Pin Connect
  - Dupont line *1& Connecting Pin (PC.13)

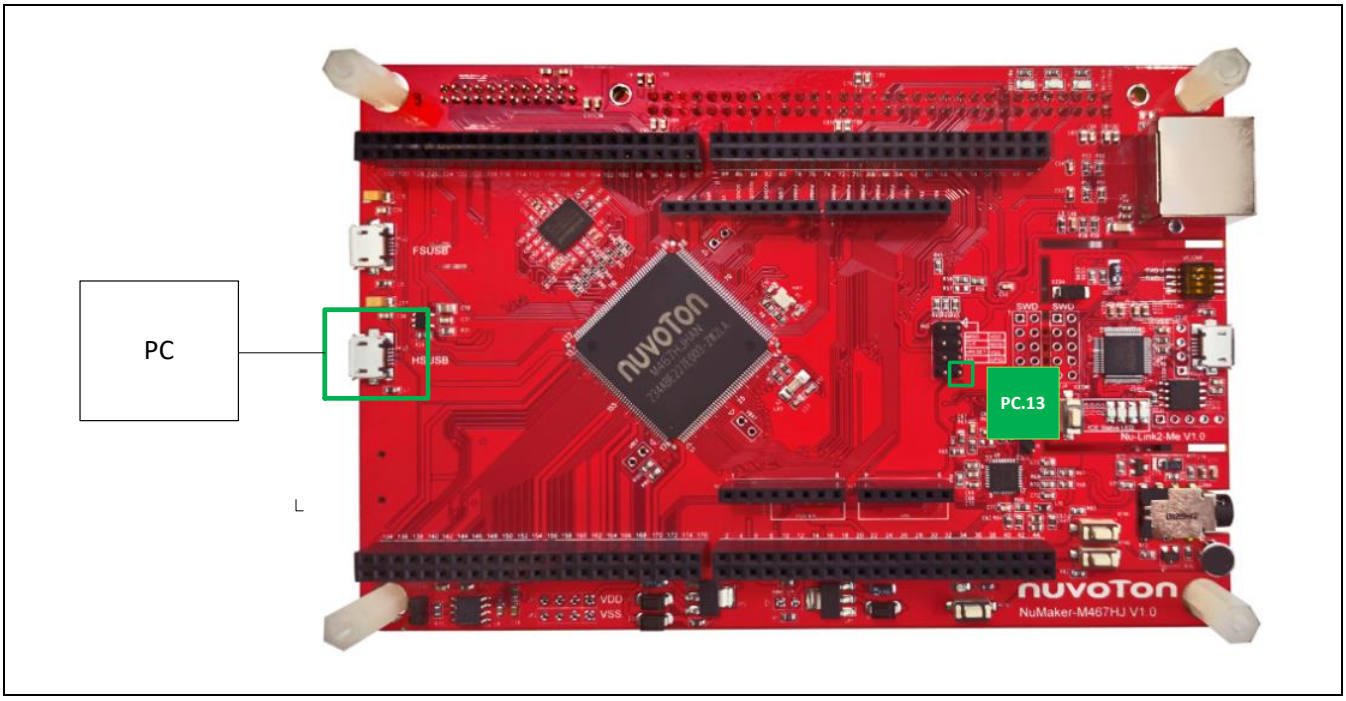


Figure 3-1 Connection Diagram

4. Directory Information

The directory structure is shown below.

EC_M467_Secure_ISP_with_Key_Store_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by ARM® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1Directory Structure

5. Example Code Execution

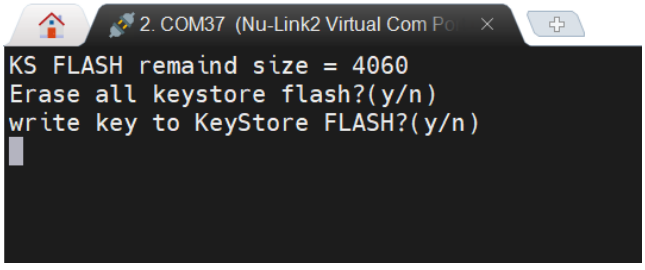
5.1 Test Flow

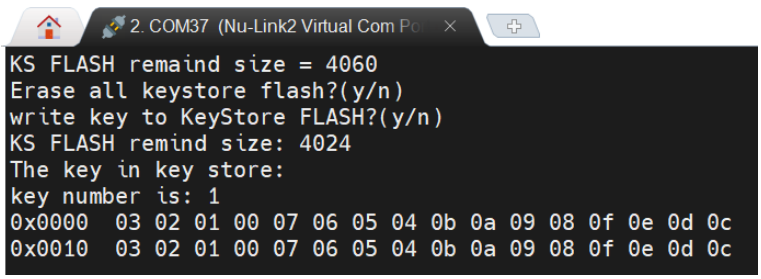
5.1.1 Key Store Test

This example code will be used to write the AES key used for decrypting the update code into the Flash area of the Key Store and configure it for exclusive use by the AES hardware encryption module to protect it.

1. Open the KEIL folder for M460_KS_KeyTest and double-click on *KS_AESKey.uvprojx.*
2. Compile the example code and program it into the NuMaker-M467HJ. Then, open the terminal device to display print messages.
3. Initially, it will display the remaining space in the Key Store Flash. If it's the first time using it, it will show 4060 bytes, and there's no need to perform the "Erase all key store flash" operation.

4. The next step will prompt whether to proceed with writing the key. Input "y" or "Y", and the program will write the data in the variable au32Key into the Key Store Flash. Upon successful writing, it will return the index of the storage location and print the data that has been written.





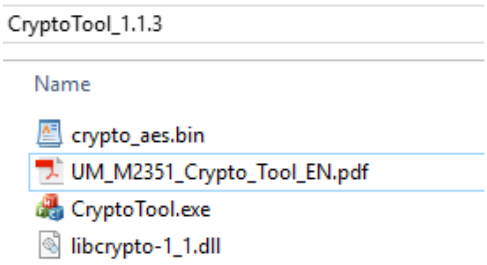
5. After completing the Key writing, the program initiates AES-CFB testing with the crypto engine. It performs encryption and decryption operations once and compares the resulting data with the original data to ensure consistency.

```
The input data:
0x0000  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0x0010  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

AES encrypt done. The output data:
0x0000  09 c4 84 cf 9f da 7b e7 6b 00 71 2a c8 e3 78 bd
0x0010  0c 38 3d e8 5b 48 8b e9 18 43 62 e6 00 14 63 1c

AES decrypt done. Decode Data:
0x0000  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0x0010  00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Done!
```

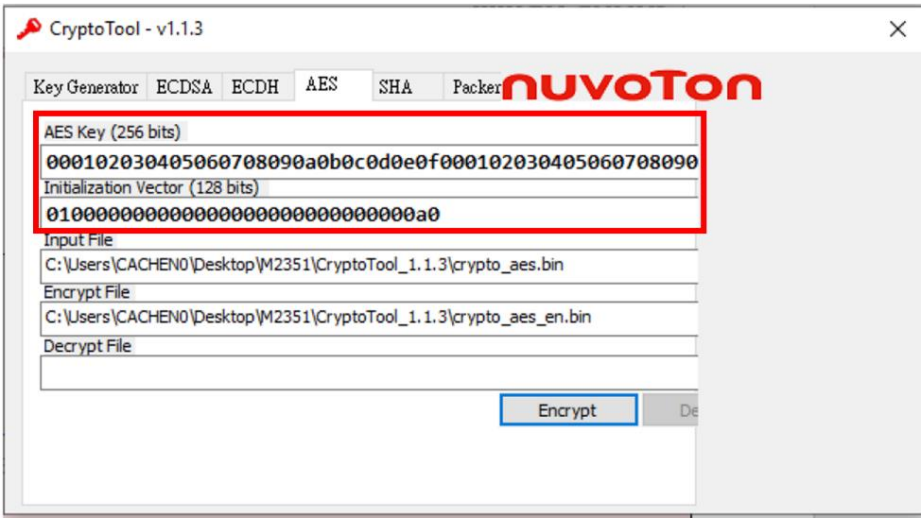
5.1.2 Image Encrypted by CryptoTool

Download link for Crypto tool: [Nuvoton NuTool Crypto Tool]

1. Run CrptoTool.exe.

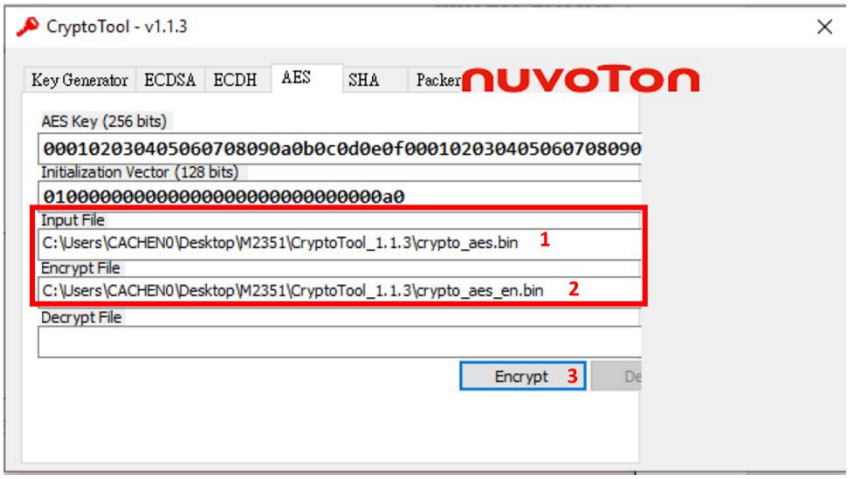


2. Select the AES field and configure AES Key as well as the Initialization Vector.



3. Build any example code in the M460 BSP, and then copy the compiled bin file to the CryptoTool folder.

4. Set the paths for the Input File and Encrypt File, then press the Encrypt button. After a few seconds, you will see the encrypted bin file generated at that path.



5.1.3 Download ISP Code and XOM Configuration

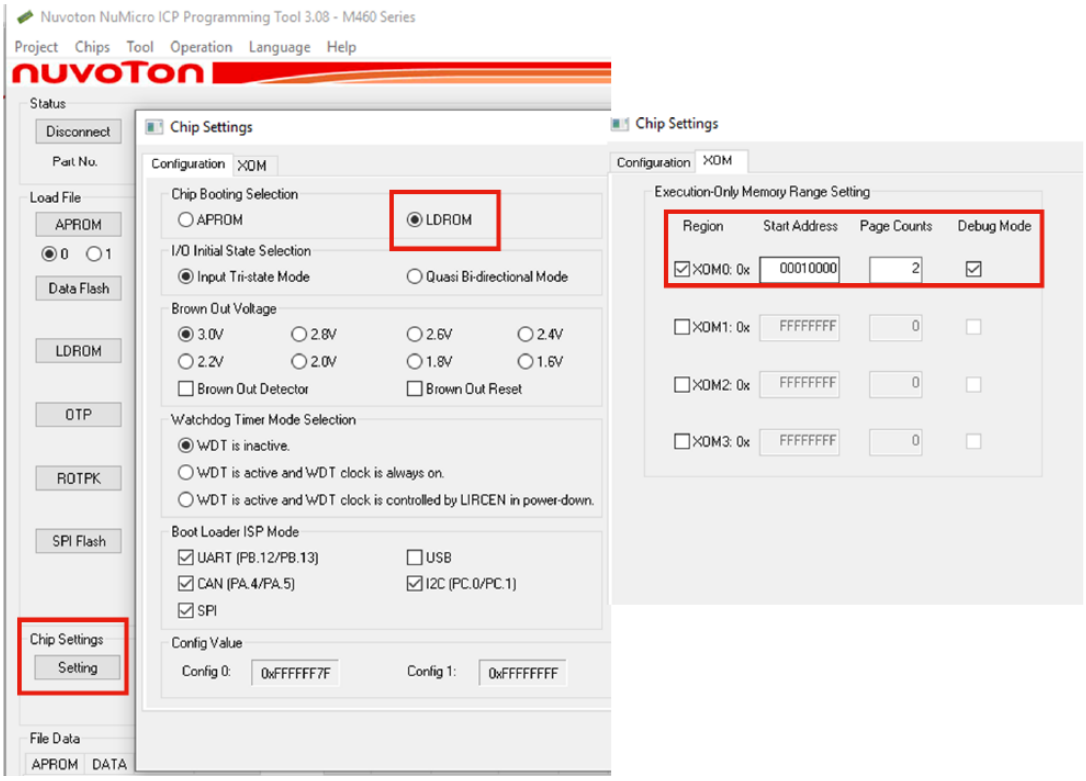
1. Open the KEIL folder for M467_ISP_HID_20_SecureUpdate and click *M467_ISP_HID_20_SecureUpdate.uvprojx*.

2. Build the program and press Download to flash it into the evaluation board.

3. Open the Nuvoton NuMicro_ICP_Programming_Tool and select the M460 series to connect to the evaluation board.

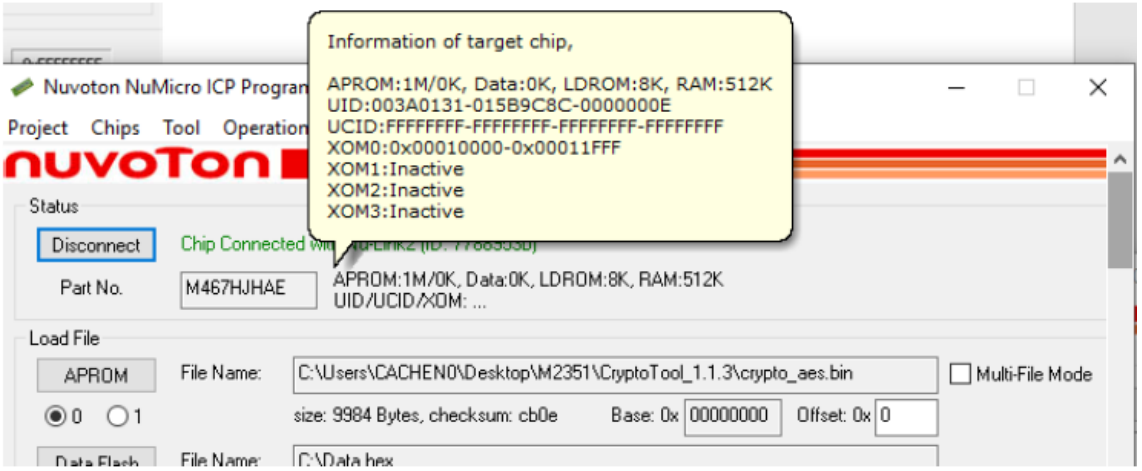
4. Select Chip Settings -> choose LDROM on the Chip Booting Selections.

5. Select Chip Settings -> XOM. Follow the settings highlighted by the red box in the below figure, if Debug Mode is not selected, you will be unable to perform single-step debugging and view instruction content in debug mode.



6. Confirm that only "Chip Setting" is checked within the Programming area, then press Start to begin flashing.

7. Reconnect the tool and ensure that XOM0 is activated.



5.1.4 ISP Update Step

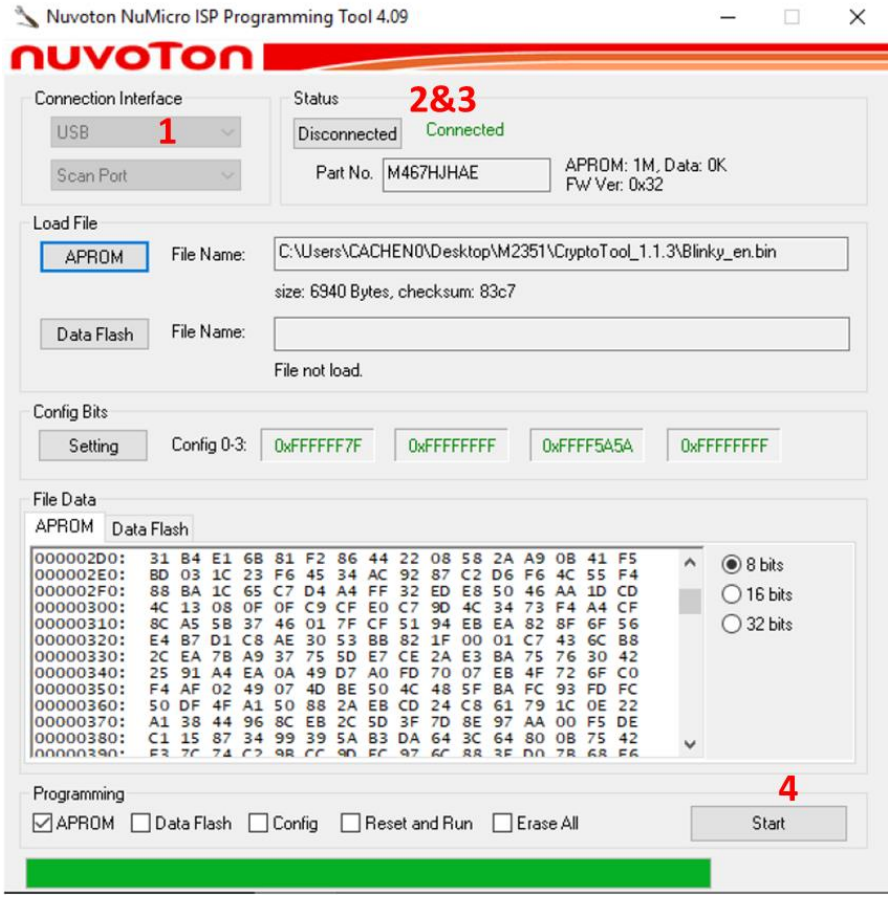
After completing the above steps, this section will begin to explain how to update the code through the Nuvoton ISP Programming Tool.

1. Open NuMicro ISP Programming Tool, and choose USB on the connection interface.

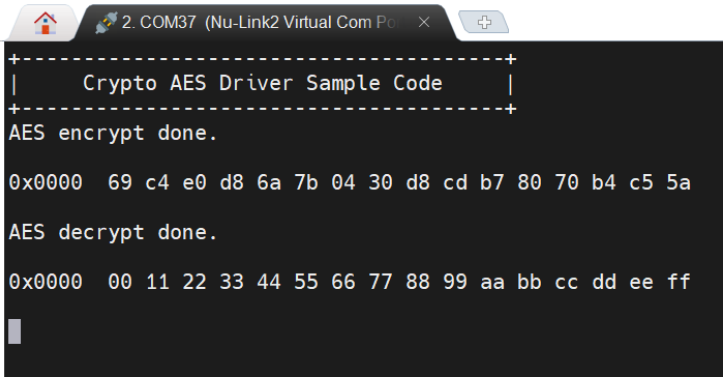
2. Connect the HSUSB port of the NuMaker-M467HJ to the PC and short PC.13 to ground.

3. Click "Connect" and then press the Reset button on the evaluation board to ensure connection with the tool.

4. Click on APROM and load the encrypted update file. Choose the APROM setting in the Programming area and then click Start.



5. Connect PC.13 to V_{CC}. When the program detects that the input signal changes from a low to a high level, it will automatically jump to execute in APROM, allowing you to view the program's execution on the terminal.



6. Revision History

Date	Revision	Description
2024.3.29	1.00	Initial version.

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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.