

AN14003

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

Rev. 1.0 — 16 August 2023

Application note

Document information

Information	Content
Keywords	AN14003, KW45 processor, KW45B41Z board, fuse programming, burning CM33 and NBU firmware, mass production, keys preparation, debug authentication, J-Link, Secure Provisioning Software Development Kit (SPSDK)
Abstract	This application note describes an efficient method to merge programming the KW45 fuse, burning CM33 and NBU firmware operations into one binary file during mass production. It also describes a method for debug authentication.



1 Introduction

KW45 is a three-core platform that integrates a Cortex-M33 application core (CM33), a dedicated Cortex-M3 radio core, and an isolated EdgeLock Secure Enclave. The radio core, also called as Narrow Band Unit (NBU) features a Bluetooth Low Energy (LE) unit with a dedicated flash. The memories integrated in the NBU consist of Bluetooth LE controller stack and radio drivers.

On KW45, only boot ROM has access to the NBU flash. The ROM bootloader provides an in-system programming (ISP) utility that operates over a serial connection on the microcontroller units (MCUs).

The speed of programming the image using ISP is relatively slower than SWD. During the mass production of KW45, it is necessary to program the fuse first, download the NBU firmware and finally download the CM33 firmware. This document describes a method, which merges the fuse programming and burning CM33 and NBU firmware operations to produce a single binary file. The method increases the production efficiency as it requires downloading the merged binary file only once through Serial Wire Debug (SWD). The document also describes a method to write the RoTKTH and SB3KDK fuse keys to a KW45B41Z board in which these keys are null.

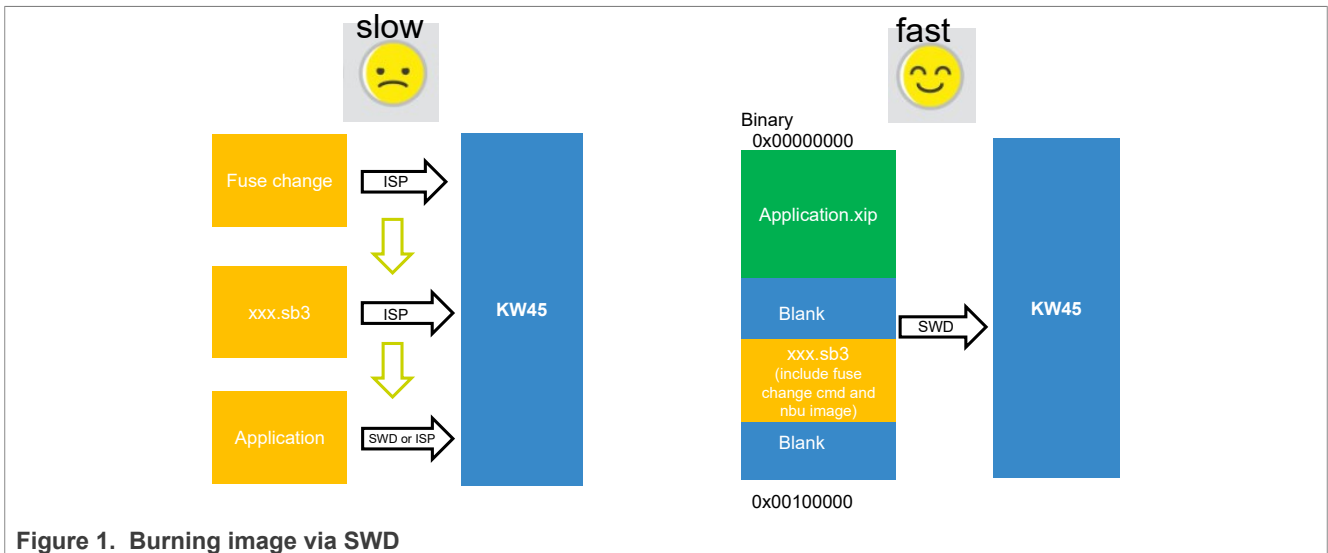


Figure 1. Burning image via SWD

Note: The method of burning the fuse provided in this document cannot be reversed. The keys programmed to fuses on KW45 cannot be changed anymore. Therefore, it is recommended to modify the fuse with caution.

2 Prerequisites

A basic understanding of the boot process of ROM boot and security is required for implementing the steps described in this document. For more details, refer to KW45 Reference Manual, see [Section 8](#).

In brief, the process implements fuse programming and updates NBU in the CM33 image. Then the CM33 image and NBU image are merged to a single image. After downloading merged image to KW45 flash of CM33 via SWD, fuse programming is done first and then the image is burnt to NBU. The flow is shown in [Section 2](#).

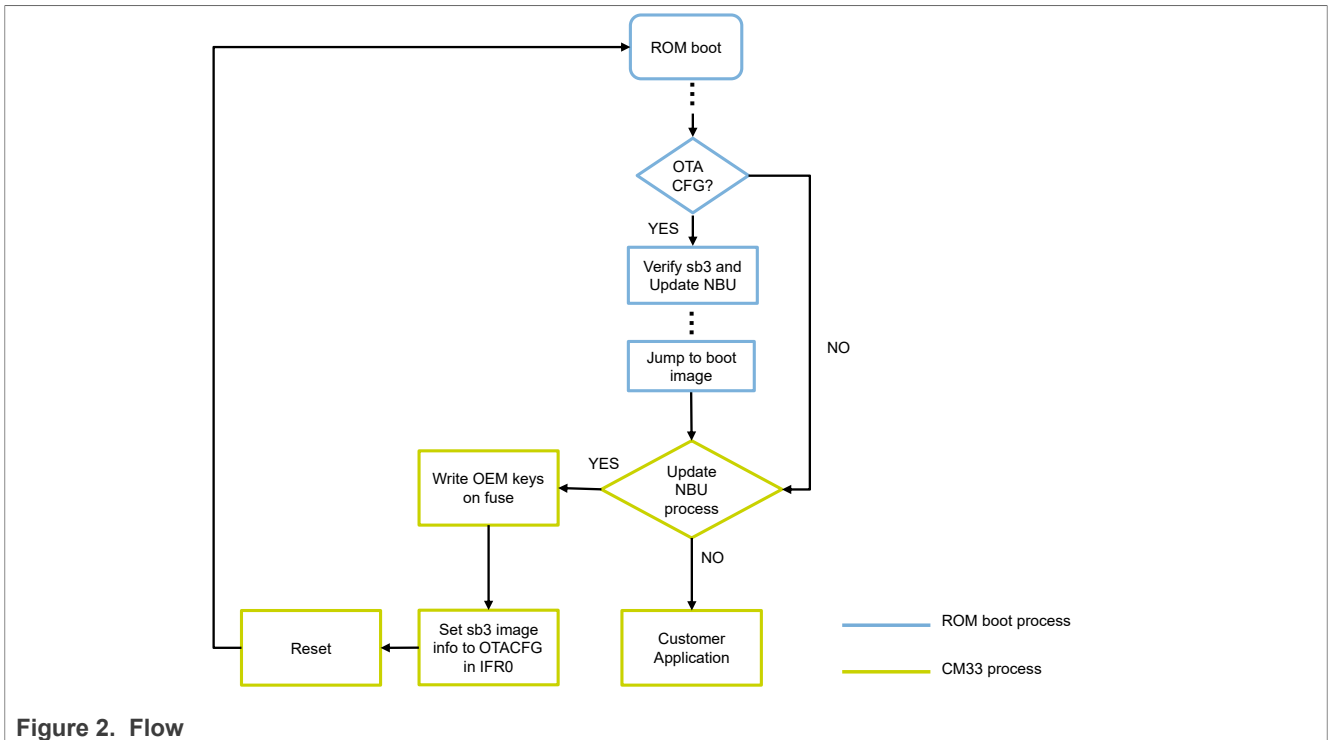


Figure 2. Flow

There are two limitations for the method:

- The flash size of CM33 core should be enough for application + .sb3 file.
- Lifecycle of KW45 should be in OEM-OPEN state.

3 Debug session initiation

The method to initiate a debug session varies depending on the device state and intended debug scenario.

- For a lifecycle that does not require debug authentication, the debug session can be initiated without performing debug authentication.
- For a lifecycle that requires debug authentication, debug session can be initiated only after debug authentication.

4 OEM keys and certificate preparation by SPSDK

This section describes the steps for preparing OEM keys and certificate preparation using SPSDK.

4.1 Setting up the SPSDK environment

Secure Provisioning SDK (SPSDK) is an open source python SDK library with its source code released on [Github](#) and [PyPI](#). It provides a set of API modules for custom production tool development, which requires more advanced secure provisioning flow. This document uses Windows system as example.

For more details, refer to SPSDK user guide on: <https://spsdk.readthedocs.io/en/latest/index.html>.

1. Install Python 3.7+ on a personal laptop.
2. Open the command prompt (cmd.exe). Then, run the following commands to install SPSDK:

```
python -m venv venv

venv\Scripts\activate
python -m pip install --upgrade pip
pip install spsdk
```

3. The above commands create a virtual environment on the user's laptop. Then install SPSDK also includes dependence file on the virtual environment.
4. Jupyter Lab is a web-based interactive development environment. NXP provides examples based on Jupyter Notebook as an easy interactive tool for user, on above virtual environment install Jupyter Lab by using the command:

```
pip install jupyterlab
```

Jupyter Notebook examples are also provided as interactive documentation. For users with no experience with the Jupyter environment, reading the tutorial available at the below URL is recommended:

<https://docs.jupyter.org/en/latest/start/index.html>

5. KW45 Jupyter notebook example file is located in path where SPSDK is released on [Github](#).
 - This path is: `spsdk/examples/jupyter_examples/kw45xx_k32w1xx/`
 - The file extension is `.ipynb`, download it to personal laptop.
6. In the virtual environment, launch the Jupyter Lab using the command below:

```
cd C:\path of store Jupyter notebook example.ipynb
jupyter-lab
```

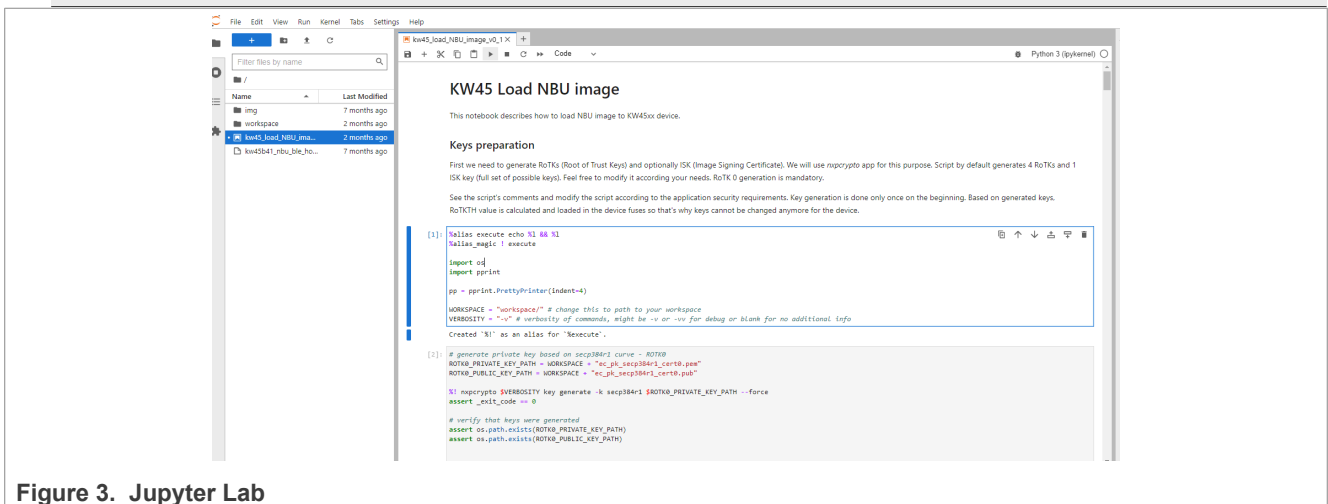


Figure 3. Jupyter Lab

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

Each section of Jupyter Notebook example contains an executable field and descriptions. To execute a field, select it and then click the **Execute** button in top menu icon of the window. Refer [Figure 3](#).

4.2 Preparing the keys and certificate

Two types of keys must be written to KW45 fuse during mass production. In a factory chip, the keys in the fuse are null. These keys are listed below:

- **RoTKTH** (CUST_PROD_OEMFW_AUTH_PUK): Four Roots of Trust Key pairs (RoTK) generate this key.
- **SB3KDK** (CUST_PROD_OEMFW_ENC_SK): It is an Advanced Encryption Standard (AES) key.

CAUTION:

The fuse is a program-once region. If the RoTKTH and SB3KDK keys are not written to fuse or the incorrect keys are written to the firmware, then the method described in this document would fail.

By default, RoTKTH and SB3KDK are provided for KW45B41Z-EVK board. This document describes a method to write these default keys to a KW45B41Z board in which these keys in fuse are null. For more details on how RoTKTH and SB3KDK can be generated and how these keys are used for secure boot, refer to 'Secure Boot for KW45 and K32W (AN13838). See [Section 8](#).

To execute the code to generate new keys, follow the description in the Jupyter Notebook example. Users can change the WORKSPACE as per the actual keys and certificate files storage path, set VERBOSITY as '-v' to observe the debug and additional information. Refer to [Figure 4](#).

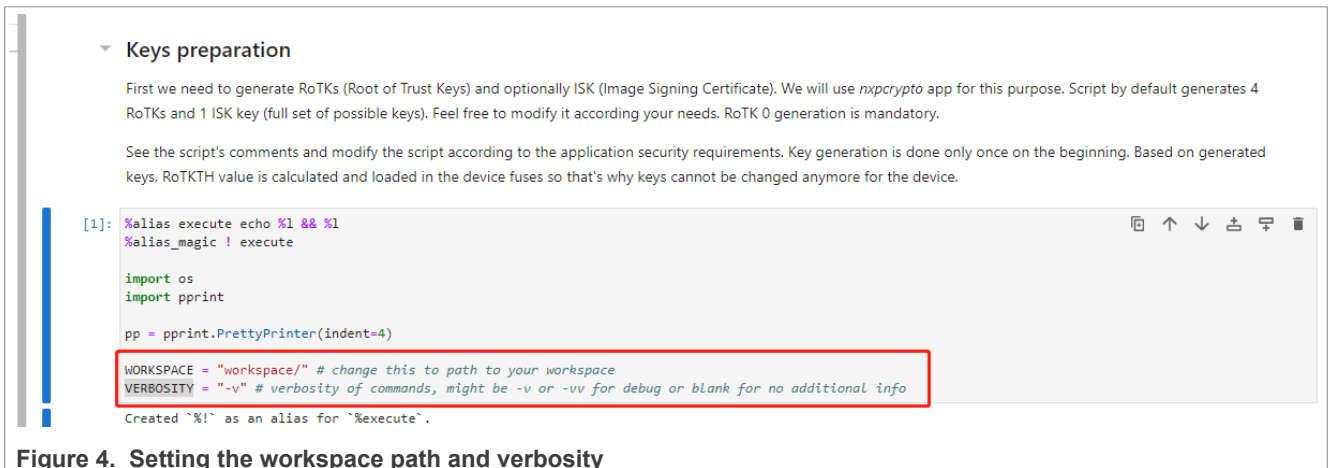


Figure 4. Setting the workspace path and verbosity

If you want to use existing keys and/or certificate, then execute the first executable field, skip executing the keys preparation, certificate preparation, and generate the SB3KDK code. Then, change the keys and certificate path as shown in [Section 5.1](#).

The RoTKTH and SB3KDK keys can be observed during Sb3.1 generation when VERBOSITY is set to “-v” as shown in [Figure 5](#).

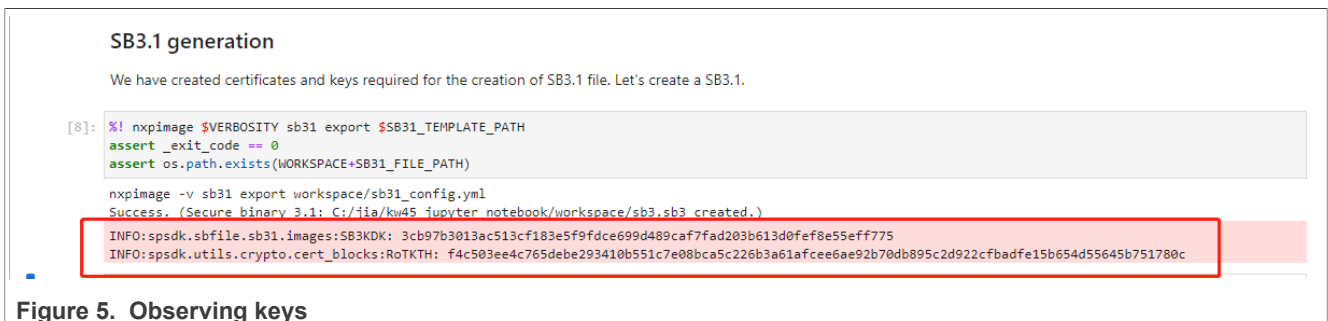


Figure 5. Observing keys

5 Mass processing and image preparation

5.1 NBU image preparation

NBU firmware file (.xip) is provided in the SDK path:

```
SDK store path\middleware\wireless\ble_controller\bin\
```

Note: If the user burns fuse using EVK default keys, the .sb3 file located in above path could be used. Otherwise, the .xip file should be used for generating customer .sb3 file first.

5.1.1 SB3 file generation

Sb3 file can be generated by executing the “Prepare SB3.1 configuration file” code in the Jupyter Notebook example. For this purpose, provide the following in the field as shown in [Figure 6](#):

- The path of the keys (SB3KDK_KEY_PATH, ROTK0_PRIVATE_KEY_PATH)
- The certificate path (ROOT_0_CERT_PATH, ROOT_1_CERT_PATH, ROOT_2_CERT_PATH, ROOT_3_CERT_PATH) The path is defined and filled to the above parameters if the user executes all executable fields on the Jupyter notebook example.

In order to use the existing keys and/or certificate, you should define the path of existing keys and/or certificate in the above parameters.

```
[7]: SB31_FILE_PATH = "sb3.sb3"
import yaml
assert os.path.exists(SB31_TEMPLATE_PATH)

# Create configuration for sb31
with open(SB31_TEMPLATE_PATH) as sb31_config:
    # Load yaml configuration to dictionary
    sb31 = yaml.safe_load(sb31_config)
    # change paths
    sb31['containerOutputFile'] = SB31_FILE_PATH
    sb31['containerKeyBlobEncryptionKey'] = SB3KDK_KEY_PATH
    sb31['description'] = "384_none_nbu_only"
    sb31['kdkAccessRights'] = 3
    sb31['containerConfigurationWord'] = 0
    sb31['rootCertificate0File'] = ROOT_0_CERT_PATH
    sb31['rootCertificate1File'] = ROOT_1_CERT_PATH
    sb31['rootCertificate2File'] = ROOT_2_CERT_PATH
    sb31['rootCertificate3File'] = ROOT_3_CERT_PATH
    sb31['mainRootCertId'] = 0
    sb31['mainRootCertPrivateKeyFile'] = ROTK0_PRIVATE_KEY_PATH
    sb31['rootCertificateEllipticCurve'] = "secp384r1"
    sb31['useIsk'] = False
    del sb31['signingCertificatePrivateKeyFile']
    del sb31['signingCertificateFile']
    del sb31['signCertData']
    del sb31['commands']
    del sb31['iskCertificateEllipticCurve']
    sb31['commands'] = [
        {
            "erase": {
                "address": "0x48800000 ",
                "size": "0x30000"
            }
        },
        {
            "load": {
                "address": "0x48800000 ",
                "file": "kw45b41_nbu_ble_hosted_04.xip"
            }
        }
    ]

with open(SB31_TEMPLATE_PATH, "w+") as sb31_config:
```

Figure 6. sb3 file generation configuration

5.1.2 Changing the lifecycle (optional)

The KW45 lifecycle can be changed by adding `programFuse` command in SB3 file generation code (refer to [Figure 7](#)). However, if you do not want to change the lifecycle, then do not add this command.

```
"commands": [
    {"erase": {"address": "0x48800000", "size": "0x30000"}},
    {"load": {"address": "0x48800000", "file": "..\\workspace\\signed images\\kw45b41_nbu_ble_hosted_a1_2_l2_2.xip"}}
]
{"programFuses": {"address": "0x0A", "values": "0x0F"}},
```

Figure 7. Program Fuse command

Note:

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

- In case the board lifecycle is changed to an after OEM-OPEN state (for example, ... 0x0F) by adding the `programFuse` command the secure boot is enabled after update of NBU. This implies that CM33 image should be signed by the keys generated. Refer to [Section 4.2](#) for generating keys.
- The `programFuse` command can be added to change the board lifecycle to any after OEM-OPEN state (for example, ... 0x0F). In such as case, the flash of CM33 cannot be accessed via SWD, unless enable debug authentication is done as mentioned in [Section 6](#).

5.2 CM33 image preparation

Take SDK project otac_att as example, set the gEraseNVMLink_d linker symbol to 0 while generating the binary file.

If flash of NBU is empty, the Bluetooth Low Energy example code is stacked when initializing NBU. Also there would be an issue when Bluetooth Low Energy host stack is initialized. So, use a flag stored in a non-volatile memory to indicate whether to perform normal Bluetooth Low Energy process, or perform burn NBU image process. For the process described in this document, this flag stores a reserve variable in HWParameters as show in Figure 8.

```

51 ~ #endif
52
53 static void start_task(void *argument)
54 {
55 #if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
56 uint32_t status;
57 /* Load the HW parameters from Flash to RAM
58 This demo test if NBU can update from M33 app when in mass production:
59 1. Download M33 signed firmware(xip file) and NBU with .SB3 format to
60 M33 flash
61 2. Use a personal defined flag in HWParameters check whether burn NBU.
62 Before start BLE platform, check if it needs update NBU. If yes,
63 Because there have no NBU firmware when first download, so BLE
64 platform is not started before NBU firmware is burned.*/
65 status = NV_ReadHWParameters(&TestHWParams);
66
67 if((status == 0U) && pTestHWParams->reserved[63] != 0xFF)
68 {
69 #endif
70
71 /* Start BLE Platform related resources such as clocks, Link Layer and HCI transport to Link Layer */
72 (void)APP_InitBLE();
73
74 #if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
75 }
76 #endif
77 /* Start Application services (timers, serial manager, Low power, Led, button, etc..) */
78 APP_InitServices();
79
80 /* Start Host stack */
81 BluetoothLEHost_AppInit();
82
83 while(TRUE)
84 {
85 BluetoothLEHost_HandleMessages();
86 }
87
88

```

Figure 8. Burning NBU flag

5.2.1 Default OEM keys on KW45B41Z board

For the KW45B41Z-EVK board, NXP provides the default keys in fuse and these are shown in Figure 9. However, for the KW45 chip received from the factory, by default, the SBKDK and RoTKTH keys in the fuse are null. Therefore, writing these two keys to fuse is necessary as the sb3 update would fail without these two keys in fuse.

```

# if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
extern hardwareParameters_t *pTestHWParams;
nboot_context_t JiaTestContext;
/*fill user SB3KDK and RoTKTH in below array, in this demo, it use same key as in KW45 EVK */
uint8_t user_SB3KDK[32] = {0x7a, 0xa7, 0xef, 0x98, 0x13, 0xb3, 0x56, 0x12, 0x57, 0xb8, 0x83, 0x7d, 0xab, 0x26, 0x22, 0x53,
0x01, 0xdf, 0x35, 0x11, 0x21, 0x7f, 0x27, 0x33, 0xc7, 0x1d, 0xad, 0xcd, 0x44, 0x77, 0x22, 0xd1};
uint8_t user_RoTKTH[32] = {0x65, 0x0d, 0x80, 0x97, 0x07, 0x9f, 0xf2, 0x7a, 0x3e, 0x8a, 0x2d, 0xa1, 0x47, 0x81, 0xb9, 0x22,
0xfd, 0x82, 0x95, 0xb6, 0xc0, 0x0b, 0xfa, 0x06, 0x7f, 0x00, 0xe8, 0x7f, 0x1a, 0x16, 0xb8, 0xb3};
# endif

```

Figure 9. KW45 EVK default keys

5.2.2 Writing OEM keys in application code by nboot API

KW45 ROM bootloader provides nboot API that can program the fuse. Follow the below steps to enable nboot API:

1. In the BLE example project, add the driver files fsl_romapi.c and fsl_nboot.h to the project.
2. Open fsl_romapi.c, remove the API related to flash operation, otherwise duplicate definitions error would appear while compiling the project.

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

3. In `otap_client_att.c`, add:

```
#include "fsl_nboot.h"
```

Then, define the SB3KDK and RoTKTH array as shown in [Figure 9](#).

The function code shown in [Figure 10](#) shows how to program the SBKDK and RoTKTH to the fuse:

```
#if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
int JiaTest_Set_LifeCycleAndKeys_Secure(uint8_t * pSB3KDK, uint8_t * pRoTKTH)
{
    static spc_active_mode_sys_ldo_option_t SysLdoOption;
    status_t TestSta;
    uint32_t RegPrimask;
    uint8_t TestTempBuff[32] = {0};
    nboot_status_t TestSta1, TestSta2, TestSta3;

    PRINTF("\r\n----- Test read and write fuse ----- \r\n");
    /* When select System LDO voltage level to Over Drive voltage, The HVD of System LDO must be disabled. */
    SPC_EnableActiveModeSystemHighVoltageDetect(SPC0, false);
    /* Use PRINTF to wait for a while for SPC_EnableActiveModeSystemHighVoltageDetect complete operation */
    PRINTF("\r\nSet SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V)\r\n");
    /* Set SYS LDO VDD Regulator regulate to Over Drive Voltage(2.5V) */
    SysLdoOption.SysLDOVoltage = kSPC_SysLDO_OverDriveVoltage;
    SysLdoOption.SysLDODriveStrength = kSPC_SysLDO_NormalDriveStrength;
    /* SPC use default configuration, so we just set sys Ldo option
    TestSta = SPC_SetActiveModeSystemLDORegulatorConfig(SPC0, &SysLdoOption);

    if(kStatus_Success == TestSta)
    {
        /* Disabling the interrupts before making any ROM API call is suggested,
        since API code does not deal with interrupts */
        RegPrimask = DisableGlobalIRQ();

        /* Set/read keys in fuse */
        TestSta1 = NBOOT_ContextInit(&JiaTestContext);
        TestSta2 = NBOOT_FuseProgram(&JiaTestContext, NBOOT_FUSEID_CUST_PROD_OEMFW_AUTH_PUK, (uint32_t *)pRoTKTH, 32);
        TestSta2 = NBOOT_FuseRead(&JiaTestContext, NBOOT_FUSEID_CUST_PROD_OEMFW_AUTH_PUK, (uint32_t *)TestTempBuff, 32);

        // TestSta3 = NBOOT_FuseRead(&JiaTestContext, NBOOT_FUSEID_CUST_PROD_OEMFW_ENC_SK, (uint32_t *)TestTempBuff, 32);
        TestSta3 = NBOOT_FuseProgram(&JiaTestContext, NBOOT_FUSEID_CUST_PROD_OEMFW_ENC_SK, (uint32_t *)pSB3KDK, 32);

        NBOOT_ContextFree(&JiaTestContext);

        /* Enable the interrupts after rom api calls */
        EnableGlobalIRQ(RegPrimask);

        /* Set SYS LDO VDD Regulator regulate to Normal Voltage(1.8V) */
        SysLdoOption.SysLDOVoltage = kSPC_SysLDO_NormalVoltage;
        SysLdoOption.SysLDODriveStrength = kSPC_SysLDO_NormalDriveStrength;
        TestSta = SPC_SetActiveModeSystemLDORegulatorConfig(SPC0, &SysLdoOption);

        PRINTF("\r\n Set SYS LDO VDD Regulator to Normal Voltage(1.8V)\r\n");
        PRINTF("\r\nTestSta1 is %X, TestSta2 is %X, TestSta3 is %X \r\n", TestSta1, TestSta2, TestSta3);
        for(uint8_t i=0; i < 32; i++)
        {
            PRINTF("%X", TestTempBuff[i]);
        }
        SPC_EnableActiveModeSystemHighVoltageDetect(SPC0, true);

        /* Use PRINTF to wait for a while for SPC complete operation */
        PRINTF("\r\n----- End test ----- \r\n");
    }
    else
    {
        PRINTF("\r\n Failed sta is %d \r\n", TestSta);
    }
    return 0;
}
#endif
```

Figure 10. Programming the fuse

The fuse programming steps are listed below:

- The SYS LDO VDD Regulator level needs to be regulated to Over Drive Voltage level (2.5 V) when trying to program the fuse. The default SYS LDO VDD Regulator level is regulated to normal voltage 1.8 V.
- The nboot API does not deal with any interrupts, so you should disable the interrupts before making any nboot API calls.

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

- The fuse is a program-once region. Hence, if the key region in fuse already has some values, then fuse programming will fail except for the same keys.
- After fuse is programmed, enable the interrupts.
- The SYS LDO VDD can only operate at the over-drive voltage for a limited amount of time for the life of the chip. Therefore, after programming the fuse, set the SYS LDO VDD to normal voltage.

The program keys to fuse operation must be done prior to modifying the OTA update configurations mentioned in [Section 5.2.3](#).

5.2.3 Updating OTA update configurations

KW45 Boot ROM has a firmware update feature that can be used for updating both CM33 and NBU firmware. For example, it indicates and provides metadata information for update to be performed in KW45 IFR0 OTACFG page. (Refer to OTA update configuration in *KW45 RM*). The target image is the NBU image prepared as mentioned in [NBU image preparation](#). The NBU image is placed in the address 0x7A000 as shown in [Figure 11](#).

```

void BluetoothLEHost_AppInit(void)
{
    /*Install callback for button*/
    #if (defined(gAppButtonCnt_c) && (gAppButtonCnt_c > 0))

    #if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
        if(pTestHWParams->reserved[63] != 0xFF)
        {
            #endif
            /* Initialize Bluetooth Host Stack */
            BluetoothLEHost_SetGenericCallback(BluetoothLEHost_GenericCallback);
            BluetoothLEHost_Init(BluetoothLEHost_Initialized);

        #if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
        }
        else
        {
            int res = -1;
            int st;
            OTALoaderInfo_t loader_info;
            res = PLATFORM_OtaClearBootFlags();
            if(res == 0)
            {
                /*OTA successful, OTA update configuration will be cleared
                PRINTF("\r\nFirmware update successful.\r\n");
                pTestHWParams->reserved[63] = 'S';
                NW_WriteHWParameters();
                PRINTF("\r\nset HWParameter->reserved[63] as 'S' as a flag, recover the flag by long press button SW2\r\n");
                PRINTF("\r\nReset MCU again for running normal application\r\n");
                HAL_ResetMCU();
            }
            else if (res == 1)
            {
                PRINTF("\r\nTest for update nbu firmware, NBU firmware is null, start nbu firmware update.\r\n");
                PRINTF("\r\nset new image flag to IFR0 -> OTACFG\r\n");

                /* Program Keys to fuse, this must before update NBU*/
                JiaTest_Set_LifeCycleAndKeys_Secure(user_SB3KDK, user_ROTKTH);

                loader_info.image_addr = 0x7a000; //this is sb3 file address that store in internal flash, align with 8k
                loader_info.image_sz = 197524; //The OTA SelectedFlash->base_offset is 0x7a000, I also use it.
                loader_info.pb1map = NULL; //fill sb3 file size
                //use internal flash
                loader_info.sb_arch_in_ext_flash = false;
                loader_info.spi_baudrate = 0;

                st = PLATFORM_OtaupdateBootFlags(&loader_info);

                PRINTF("\r\nUpdate OTACFG status is %d \r\n", st);
                PRINTF("\r\nReset MCU \r\n");
                HAL_ResetMCU();
            }
            else if (res == 2)
            {
                /*OTA failed, OTA update configuration will be cleared
                PRINTF("\r\nFirmware update failed.\r\n");
            }
            else
            {
                /*unknow OTA firmware update status
                PRINTF("\r\nUnknow firmware update status\r\n");
            }
        }
    #endif
}
    
```

Figure 11. Updating OTA update configuration

5.3 CM33 signed image generation (optional)

If lifecycle is changed to OEM secure world closed via command in .sb3 file generated earlier, that means after ROM boot processes .sb3 file, NBU image is burnt and lifecycle is also changed. Therefore, the application image should also be signed because secure boot feature is enabled in that lifecycle. If this step is not done, a command must be added to change the board lifecycle to an after OEM-OPEN state in the .sb3 file. In such a case, the signed image is not required.

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

Note:

The signed image feature is not provided in SPSDK for KW45 now. However, the signed image can be generated by using the [Over The Air Programming tool](#). Follow the steps described in *Bluetooth Low Energy Demo Applications User's Guide.pdf* to generate the .sb3 file. Refer to the step show in [Figure 12](#). The signed image can be found in the path below:

tool install path\Over The Air Programming 1.0.6.4\Secured\signedcm33.bin

- Keys and Certificate files generated in [Section 4.2](#)” should be used for signed image generation.

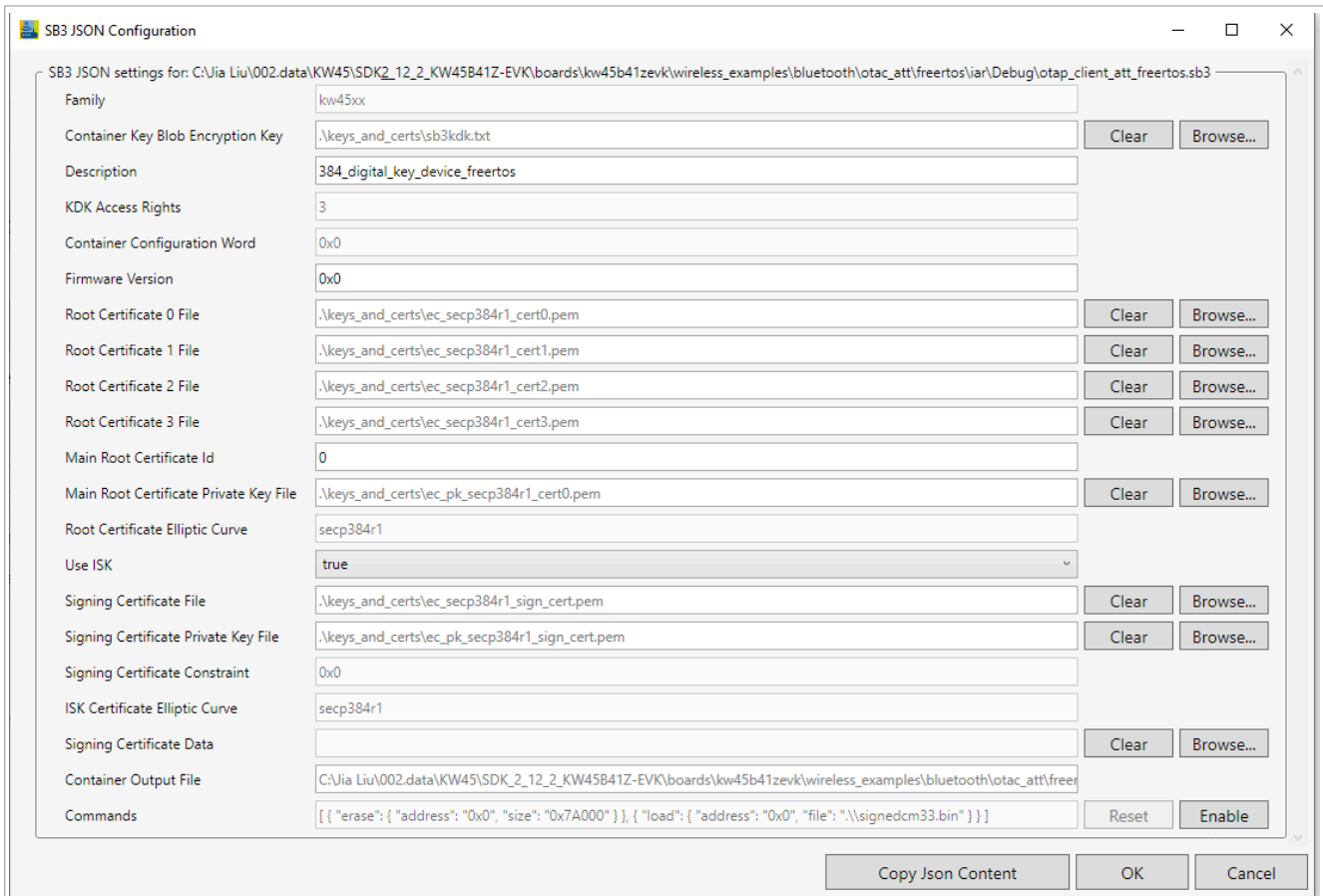


Figure 12. SB3 JSON configuration

5.4 Merging CM33 image and NBU image

Open the signed image file and .sb3 file using a Hex file editor tool. Copy the contents of .sb3 file to signedcm33.bin file, located in the address 0x7A000. Add a padding of 0x00 between the valid signedcm33.bin to 0x7A000 as shown in [Figure 13](#).

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

otap_client_att_freertos.xip and NBU.sb3.bin

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00079F40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079F50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079F60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079F70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079F80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079F90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FA0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00079FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0007A000	73	62	76	33	01	00	03	00	00	00	00	00	7F	02	00	00	sbv3.....
0007A010	34	01	00	00	0A	8B	23	2B	00	00	00	00	00	00	00	00	4....<#+.....
0007A020	FC	01	00	00	06	00	00	00	6C	00	00	00	33	38	34	5F	ä.....1...384
0007A030	6E	6F	6E	65	5F	6E	62	75	5F	6F	6E	6C	4C	BB	F8	44	none_nbu_onlL»eD
0007A040	74	93	A6	F0	7D	01	5C	72	48	D3	EB	35	32	0A	E7	4C	t";8).\rH0e52.çL
0007A050	94	86	69	11	6C	43	D2	5A	E1	3F	05	65	A3	D8	0C	FC	"ti.lC0Zá?.e20.d
0007A060	A4	B3	03	D7	4B	E6	4B	A5	51	B3	65	57	63	68	64	72	m³.*KekWQ³eWchdr
0007A070	01	00	02	00	30	01	00	00	42	00	00	80	4E	89	B9	3E	...0...B..eNt²>
0007A080	FB	25	00	CA	C1	10	36	49	77	9A	7A	46	19	D1	61	A4	0%.ÉÁ.6IwszF.Nam
0007A090	34	C5	7A	85	11	69	17	10	DE	FF	78	6C	93	0B	B8	21	4Äz...i..Byxl".,!
0007A0A0	49	EA	26	66	E6	8D	86	E4	5E	12	9D	06	0D	CE	EC	60	I&fa.+â^....îi\
0007A0B0	29	74	E6	FE	50	56	6A	C6	3D	E2	BE	94	2D	2F	1F	11)tapPVjE=â%"/..

Copy the contents of .sb3 file to signed cm33.bin file, located in the address 0x7A000

Figure 13. Merging the images

After merging CM33 image and NBU image, the merged image can be burnt to KW45 via SWD for mass production.

6 Debug authentication

Debug authentication scheme is a challenge-response scheme and assures that only a *debugger*, which has possession of the required debug credentials can successfully authenticate over debug interface. Such a debugger can also access restricted parts of the device. The below sections describe steps for debug authentication.

6.1 Preparing keys and certificates for debug

Prior to generating debug credentials, it is necessary to generate an EC keypair (secp256r1 or secp384r1) for debugger known as Debug Credential Key (DCK).

The method of generating DCK is same as RoTKs generation in SPSDK Jupyter mentioned in [Section 4.2](#), shown in [Figure 14](#). Users can use the same code and change the file name of keypair. They can use the same syntax for generating the keypair.

```
[2]: # generate private key based on secp384r1 curve - ROTK0
ROTK0_PRIVATE_KEY_PATH = WORKSPACE + "ec_pk_secp384r1_cert0.pem"
ROTK0_PUBLIC_KEY_PATH = WORKSPACE + "ec_pk_secp384r1_cert0.pub"

%! nxpcrypto $VERBOSITY key generate -k secp384r1 $ROTK0_PRIVATE_KEY_PATH --force
assert _exit_code == 0

# verify that keys were generated
assert os.path.exists(ROTK0_PRIVATE_KEY_PATH)
assert os.path.exists(ROTK0_PUBLIC_KEY_PATH)
```

keypair path and name

syntax for generating key pair

Figure 14. Key pair generation

6.2 Debugging credentials

In SPSDK virtual environment, use the syntax below for generating Configuration template:

```
nxpdebugmbox get-template -f C:\XXXX\...\XXXX\test.yml
```




Figure 15. Configuration template generation

Open the Configuration template file and modify the content accordingly. For details, refer to the comment mentioned in the Configuration template file.

```
socc: 0x0005
uuid: "00000000000000000000000000000000"
cc_socu: 0xFFFF
cc_vu: 0
cc_beacon: 0
rot_meta: path of RoTKs public part, should same as keys used by .sb3
generation(e.g.. ./secp384r1_private_key0.pub)
rot_id: 0
```

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

```
dck: path of public part of the DCK(e.g.. ./secp384r1_private_dck.pub)
rotk: path of RoTK private part for the rot_meta chosen by rot_id, should be the
      same as keys
used by .sb3 generation (e.g.. ./secp384r1_private_key0.pem)
socc: 0x0005
```

Use the syntax shown below for Debug Credential generation by using Configuration template:

```
nxpdebugmbx -p 2.1 gencd -c C: \XXXX\...\XXXX\test1.yml C: \YYYY\...\YYYY
\test.cert1
```

```
(venv) C:\Users\nxf41499>nxpdebugmbx -p 2.1 gencd -c C:\jia\kw45_debug_auth\test\test1.yml C:\jia\kw45_debug_auth\test\
test.cert1
RoT Key Hash: 650d8097079ff27a3e8a2da14781b922fd8295b6c00bfa067f00e87f1a16b8b304bf710d45cbd591e2e24be83183922c
Creating Debug credential file succeeded
..
```

Figure 16. Debug credential generation

6.3 Initiating debug authentication

1. In the virtual environment, start NXP debug box tool by using the command: ***nxpdebugmbx start***. Refer [Figure 17](#). Then, select J-Link as the debug probe.

```
(venv) C:\Users\nxf41499>nxpdebugmbx start
#  Interface  Id          Description
-----
0  PyOCD      1064975848  Segger J-Link MCU-Link
1  Jlink      1064975848  Segger J-Link MCU-Link: 1064975848
Please choose the debug probe: 1
Start Debug Mailbox succeeded

(venv) C:\Users\nxf41499>_
```

Figure 17. Initiating debug using the 'npxdebugmbx start' command

2. Use the command below to initiate debug authentication (refer [Figure 18](#)):

```
nxpdebugmbx -v -p 2.1 -i jlink auth -b 0x0 -c C: \YYYY\...\YYYY\test.cert -k
C:\ZZZZ\...\ZZZZ\secp384r1_private_dck.pem
```

C:\ZZZZ\...\ZZZZ\secp384r1_private_dck.pem is the private key generated using the steps described in [Section 6.1](#).

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

```
(venv) C:\Users\nxf41499>nxpdebugmbox -v -p 2.1 -i jlink auth -b 0x0 -c C:\jia\kw45_debug_auth\test\test.cert -k
C:\jia\kw45_debug_auth\test\secp384r1_private_dck.pem
INFO:spsdk. apps.nxpdebugmbox:Starting Debug Authentication
# Interface Id Description
-----
0 Jlink 1064975848 Segger J-Link MCU-Link: 1064975848
INFO:spsdk. apps.nxpdebugmbox:DAC:
Version : 2.1
SOCC : 5, KW45xx/K32W1xx
UUID : DA39A0723338B23DBFB5ED282A20DEC9
CC_VU : 0
ROTID_rkh_revocation : 00000000
ROTID_rkth_hash : 650d8097079ff27a3e8a2da14781b922fd8295b6c00bfa067f00e87f1a16b8b3
CC_soc_pinned : 00000000
CC_soc_default : 00000000
Challenge : 9a8306a884104efefbf7c4516aeabe7c6ebd60b5cf68455cfbac64be708cd8cb

INFO:spsdk. apps.nxpdebugmbox:DAR:
DAC:
Version : 2.1
SOCC : 5, KW45xx/K32W1xx
UUID : DA39A0723338B23DBFB5ED282A20DEC9
CC_VU : 0
ROTID_rkh_revocation : 00000000
ROTID_rkth_hash : 650d8097079ff27a3e8a2da14781b922fd8295b6c00bfa067f00e87f1a16b8b3
CC_soc_pinned : 00000000
CC_soc_default : 00000000
Challenge : 9a8306a884104efefbf7c4516aeabe7c6ebd60b5cf68455cfbac64be708cd8cb

DC:
Version : 2.1
SOCC : 5, KW45xx/K32W1xx
UUID : 00000000000000000000000000000000
CC_SOCC : 0x7f
CC_VU : 0x0
BEACON : 0
CRTK table has 4 entries
CRTK Hash: 650d8097079ff27a3e8a2da14781b922fd8295b6c00bfa067f00e87f1a16b8b304bf710d45cbd591e2e24be83183922c
Authentication Beacon: 0

INFO:spsdk. apps.nxpdebugmbox:Debug Authentication ends successfully.
```

Figure 18. Debug authentication

Starting debug

After debug authentication ends successfully, the device enables access to the debug domains permitted in the DC. Do not reset KW45, use J-Link Commander to connect to the device directly. The log is shown in [Figure 19](#).

Programming the KW45 flash for Application and Radio firmware via Serial Wire Debug during mass production

```
J-Link Commander V7.84a
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: KW45B41Z83
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "KW45B41Z83" selected.

Connecting to target via SWD
ConfigTargetSettings() start
ConfigTargetSettings() end
InitTarget() start
InitTarget() end
Found SW-DP with ID 0x6BA02477
DPIDR: 0x6BA02477
CoreSight SoC-400 or earlier
AP map detection skipped. Manually configured AP map found.
AP[0]: AHB-AP (IDR: Not set)
Iterating through AP map to find AHB-AP to use
AP[0]: Skipped ROMBASE read. CoreBaseAddr manually set by user
AP[0]: Core found
CPUID register: 0x410FD214. Implementer code: 0x41 (ARM)
Feature set: Mainline
Found Cortex-M33 r0p4, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
Security extension: implemented
Secure debug: enabled
ROM table scan skipped. CoreBaseAddr manually set by user: 0x80030000
SetupTarget() start
SetupTarget() end
Memory zones:
  Zone: Default Description: Default access mode
Cortex-M33 identified.
```

Figure 19. Secure Debug enabled

Note:

Access is disabled when KW45 is reset. Therefore, if KW45 is reset, debug authentication should be done again. This implies that the user should not use IDE to debug since reset KW45 operation may be embedded in IDE.

7 Acronyms and abbreviations

The table below lists the acronyms used in this document.

Table 1. Acronyms and abbreviations

S.No	Acronym	Description
1	AES	Advanced Encryption Standard
2	Bluetooth LE	Bluetooth Low Energy
3	DCK	Debug Credential Key
4	IDE	Integrated Design Environment
5	ISP	In-System Programming
6	NBU	Narrow Band Unit
7	OEM	Original Equipment Manufacturer
8	RoTKTH	Root of Trust Key Table Hash
9	SB3KDK	SB3 Key Derivation Key
10	SDK	Software Development Kit
11	SPSDK	Secure Provisioning SDK
12	SWD	Serial Wire Debug
13	XIP	Execute-In-Place

8 References

Refer to the below documents for more information:

- *Managing Lifecycles on KW45 and K32W148* ([AN13931](#))
- *Secure Boot for KW45 and K32W* ([AN13838](#))
- *KW45 Reference Manual* ([KW45RM](#))
- *Bluetooth Low Energy Demo Applications User's Guide.pdf*. This document can be downloaded from the KW45 SDK available on the [NXP website](#). Alternatively, contact the NXP field applications engineer (FAE) for obtaining this document.

9 Revision history

[Table 2](#) summarizes the revisions to this document.

Document revision history

Revision history		
Revision number	Release date	Description
1.0	16 August 2023	Initial public release

10 Legal information

10.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

10.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <http://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

10.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

EdgeLock — is a trademark of NXP B.V.

J-Link — is a trademark of SEGGER Microcontroller GmbH.

Contents

1	Introduction	2
2	Prerequisites	2
3	Debug session initiation	3
4	OEM keys and certificate preparation by SPSDK	4
4.1	Setting up the SPSDK environment	4
4.2	Preparing the keys and certificate	5
5	Mass processing and image preparation	6
5.1	NBU image preparation	6
5.1.1	SB3 file generation	6
5.1.2	Changing the lifecycle (optional)	7
5.2	CM33 image preparation	9
5.2.1	Default OEM keys on KW45B41Z board	9
5.2.2	Writing OEM keys in application code by nboot API	9
5.2.3	Updating OTA update configurations	11
5.3	CM33 signed image generation (optional)	11
5.4	Merging CM33 image and NBU image	12
6	Debug authentication	14
6.1	Preparing keys and certificates for debug	14
6.2	Debugging credentials	14
6.3	Initiating debug authentication	15
7	Acronyms and abbreviations	18
8	References	19
9	Revision history	20
10	Legal information	21

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.