AN14003

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

Rev. 1.1 — 14 June 2024 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.

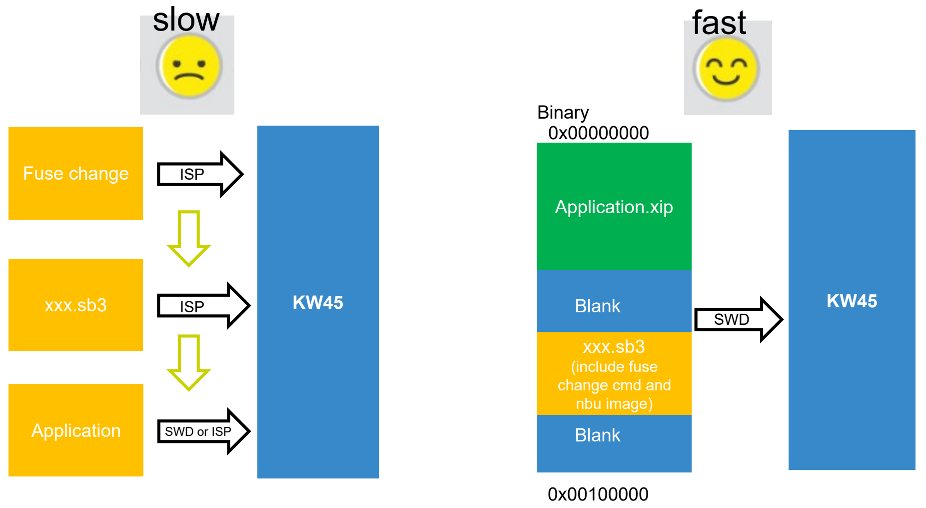

Figure 1. Burning image via SWD

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 the *KW45 Reference Manual*, see Section 8 "References".

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 the 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 "Prerequisites".

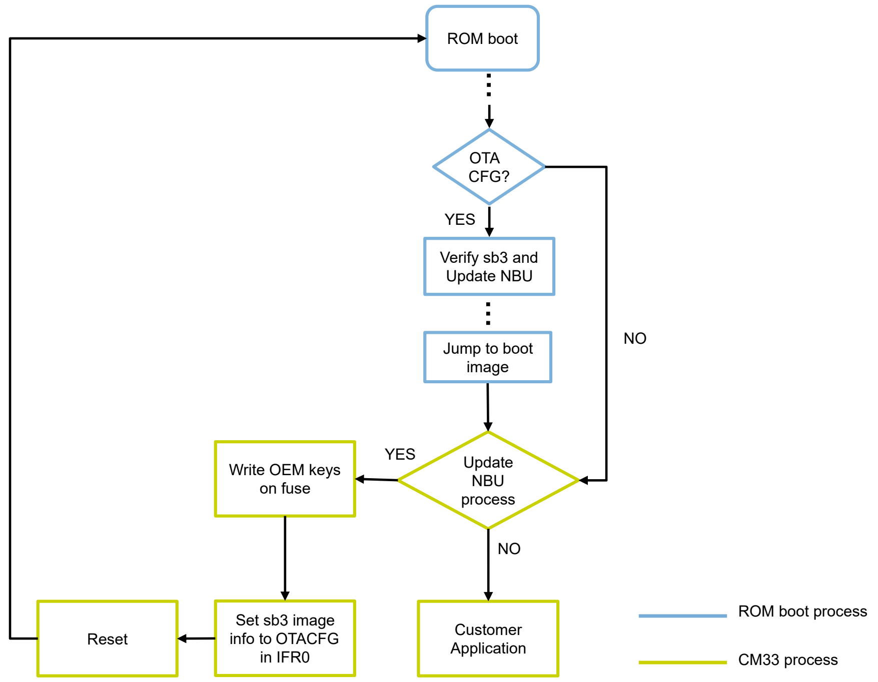

Figure 2. Process flow used

There are two limitations for the method:

- The flash size of the CM33 core must be large enough to store the application and the `.sb3` file.
- The lifecycle of the KW45 device must be in the 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 Preparing OEM keys and certificate using SPSDK

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

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 requiring more advanced secure provisioning flow. This document uses a Windows system as an 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
```

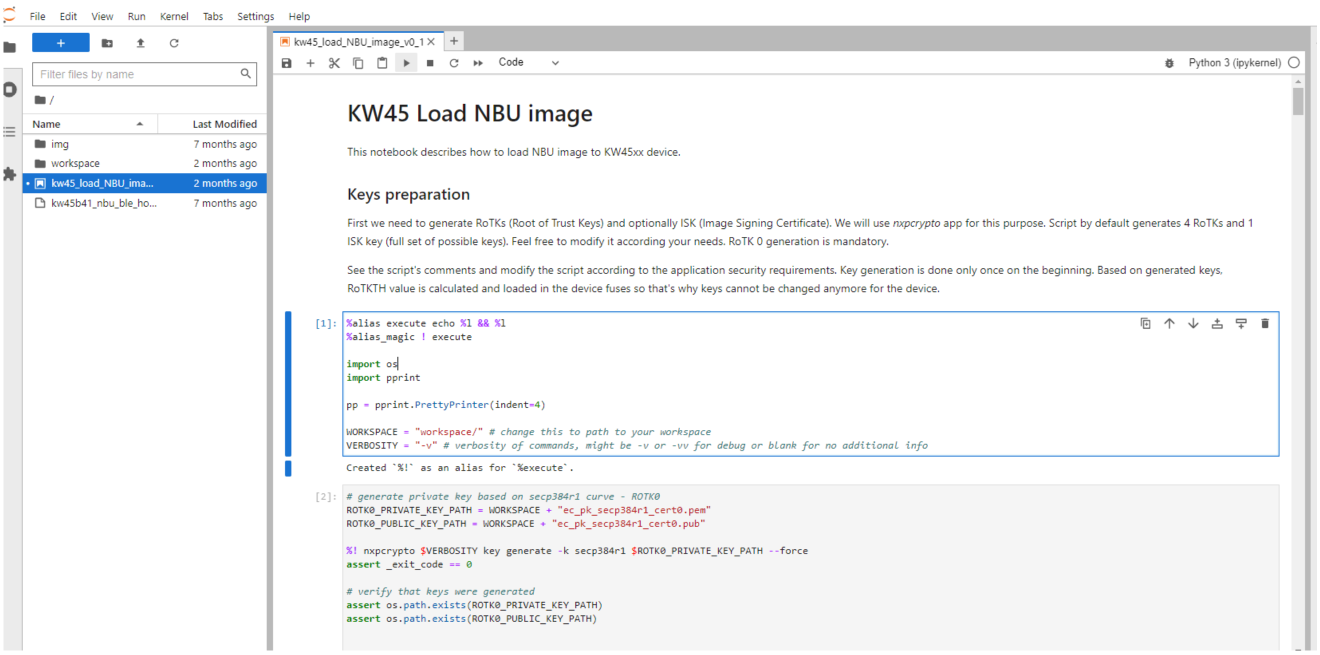

Figure 3. Jupyter Lab

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 the 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 the 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 keys can be generated and how they are used for secure boot, refer to the Application Note '*Secure Boot for KW45 and K32W* (*AN13838*). See Section 8 "References".

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 file storage path, set `VERBOSITY` as '`-v`' to observe the debug and additional information. Refer to Figure 4.

Keys preparation

First we need to generate RoTKs (Root of Trust Keys) and optionally ISK (Image Signing Certificate). We will use *nxpcrypto* app for this purpose. Script by default generates 4 RoTKs and 1 ISK key (full set of possible keys). Feel free to modify it according your needs. RoTK 0 generation is mandatory.

See the script's comments and modify the script according to the application security requirements. Key generation is done only once on the beginning. Based on generated keys, RoTKTH value is calculated and loaded in the device fuses so that's why keys cannot be changed anymore for the device.

```
[1]: %alias execute echo %l && %l
%alias_magic ! execute

import os
import pprint

pp = pprint.PrettyPrinter(indent=4)

WORKSPACE = "workspace/" # change this to path to your workspace
VERBOSITY = "-v" # verbosity of commands, might be -v or -vv for debug or blank for no additional info
```

```
Created `%!` as an alias for `%execute`.
```

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 key and certificate preparation steps, and generate the SB3KDK code. Then, change the keys and certificate path as shown in Section 5.1 "NBU image preparation".

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

SB3.1 generation

We have created certificates and keys required for the creation of SB3.1 file. Let's create a SB3.1.

```
[8]: %! nxpimage $VERBOSITY sb31 export $SB31_TEMPLATE_PATH
assert _exit_code == 0
assert os.path.exists(WORKSPACE+SB31_FILE_PATH)
```

```
nxpimage -v sb31 export workspace/sb31_config.yml
Success. (Secure binary 3.1: C:/jia/kw45 jupyter notebook/workspace/sb3.sb3 created.)
INFO:spsdk.sbfile.sb31.images:SB3KDK: 3cb97b3013ac513cf183e5f9fdce699d489caf7fad203b613d0fef8e55eff775
INFO:spsdk.utils.crypto.cert_blocks:RoTKTH: f4c503ee4c765debe293410b551c7e08bca5c226b3a61afcee6ae92b70db895c2d922cfbadfe15b654d55645b751780c
```

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 the fuse using EVK default keys, the `.sb3` file located in the above path can be used. Otherwise, the user must first generate the `.sb3` file for the customer using the `.xip` file.*

5.1.1 SB3 file generation

An `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 fields:

- 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.

To use the existing keys and/or certificate, you must define the path of existing keys and/or certificate in the above parameters as shown in Figure 6.

```
[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 a `programFuse` command in the 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_al_2_12_2.xip"}}
    {"programFuses": {"address": "0x0A","values": "0x0F"}},
```

Figure 7. `Program Fuse` command

Note:

- If 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 the CM33 image must be signed by the keys generated. Refer to Section 4.2 "Preparing the keys and certificate" 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 a case, the flash of CM33 cannot be accessed via SWD, unless debug authentication is enabled by using the steps mentioned in Section 6 "Debug authentication".

5.2 CM33 image preparation

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

If the flash of NBU is empty, the Bluetooth Low Energy example code is stacked when initializing NBU. Also an issue would occur when the Bluetooth Low Energy host stack is initialized. So, use a flag stored in a non-volatile memory to indicate whether to perform the 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 shown in Figure 8.

```
#endif

static void start_task(void *argument)
{
#if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
    uint32_t                status;
    /* Load the HW parameters from Flash to RAM
       This demo test if NBU can update from M33 app when in mass prduction:
          1. Download M33 signed firmware(xip file) and NBU with .SB3 format to
             M33 flash
          2. Use a personal defined flag in HWParameters check whether burn NBU.
             Before start BLE platform, check if it needs update NBU. If yes,
             Because there have no NBU firmware when first download, so BLE
             platform is not started before NBU firmware is burned.*/
    status = NV_ReadHWParameters(&pTestHWParams);

    if((status == 0U) && pTestHWParams->reserved[63] != 0xFF)
    {
#endif

    /* Start BLE Platform related ressources such as clocks, Link layer and HCI transport to Link Layer */
    (void)APP_InitBle();

#if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
    }
#endif
    /* Start Application services (timers, serial manager, low power, led, button, etc..) */
    APP_InitServices();

    /* Start Host stack */
    BluetoothLEHost_AppInit();

    while(TRUE)
    {
        BluetoothLEHost_HandleMessages();
    }
}

```

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 essential. Otherwise, the sb3 update would fail.

```
#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. Default keys on KW45B41Z-EVK board

5.2.2 Writing OEM keys in application code by nboot API

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

1. In the `BLE` example project, add the driver files `fsl_romapi.c` and `fsl_nboot.h` to the project.
2. Open the file `fsl_romapi.c` and remove the API related to the flash operation. This step is required to prevent duplicate definitions error that might appear while compiling the project.
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 below shows how to program the SBKDK and RoTKTH to the fuse:

```
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, TestSta4;

    PRINTF("\r\n------------- Test read and wirte 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);
    while(SPC_GetBusyStatusFlag(SPC0)); //wait here for a while, to let HW 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);
    OSA_TimeDelay(10); //just delay to let volitage reach the target level
    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(&TestContextForWriteLC);
        TestSta2 = NBOOT_FuseProgram(&TestContextForWriteLC,
 NBOOT_FUSEID_CUST_PROD_OEMFW_AUTH_PUK, (uint32_t *)pRoTKTH, 32);
        TestSta3 = NBOOT_FuseRead(&TestContextForWriteLC,
 NBOOT_FUSEID_CUST_PROD_OEMFW_AUTH_PUK, (uint32_t *)TestTempBuff, 32);
        TestSta4 = NBOOT_FuseProgram(&TestContextForWriteLC,
 NBOOT_FUSEID_CUST_PROD_OEMFW_ENC_SK, (uint32_t *)pSB3KDK, 32);

        NBOOT_ContextFree(&TestContextForWriteLC);

        /* 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);
        OSA_TimeDelay(10); //just delay to let volitage reach the target level
```

```
        PRINTF("\r\n Set SYS LDO VDD Regulator to Normal Voltage(1.8V)\r\n");
        PRINTF("\r\nTestSta1: %X, TestSta2: %X, TestSta3: %X, TestSta4: %X \r\n",
 TestSta1, TestSta2, TestSta3, TestSta4);
        for(uint8_t i=0; i < 32; i++)
        {
            PRINTF("%X", TestTempBuff[i]);
        }
        SPC_EnableActiveModeSystemHighVoltageDetect(SPC0,true);
        while(SPC_GetBusyStatusFlag(SPC0)); //wait here for a while, to let HW
 complete operation
        PRINTF("\r\n------------- End test ---------------\r\n");
    }
    else
    {
        PRINTF("\r\n Failed sta is %d \r\n", TestSta);
    }
    return 0;
}
#endif
```

The fuse programming steps are listed below:

- The SYS LDO VDD Regulator level must be regulated to Over Drive Voltage level (2.5 V) while 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. Therefore, you must disable the interrupts before making any nboot API calls.
- The fuse is a program-once region. Therefore, if the key region in fuse already has some values, then fuse programming fails except for the same keys.
- After the fuse is programmed, enable the interrupts.
- The SYS LDO VDD can only operate at the overdrive 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 "Updating OTA update configurations".

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 the code below.

```
void BluetoothLEHost_AppInit(void)
{
    /*Install callback for button*/
#if (defined(gAppButtonCnt_c) && (gAppButtonCnt_c > 0))
    (void)BUTTON_InstallCallback((button_handle_t)g_buttonHandle[0],
 (button_callback_t)BleApp_HandleKeys0, NULL);
#endif
#if TEST_UPDATE_NBU_FROM_APP_IN_SECURE_LIFECYCLE
    if(pTestHWParams->reserved[62] != 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 sucessful.\r\n");
            pTestHWParams->reserved[62] = 'S';
            NV_WriteHWParameters();
            PRINTF("\r\nSet HWParameter->reserved[62] 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 8Kb
                                                    //The OTA SelectedFlash-
>base_offset is 0x7a000, I also use it.
            loader_info.image_sz   = 194240;        //fill sb3 file size
            loader_info.pBitMap    = NULL;
            //use internal flash
            loader_info.partition_desc = &Test_ota_partition;
//            loader_info.sb_arch_in_ext_flash = false;
//            loader_info.spi_baudrate         = 0;

            st = PLATFORM_OtaNotifyNewImageReady(&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
}
```

5.3 CM33 signed image generation (optional)

If the device lifecycle is changed to OEM secure world closed via a command in the `.sb3` file generated earlier, it signifies that after ROM boot processes the `.sb3` file, the NBU image is burnt and the lifecycle is also changed. Therefore, the application image must also be signed because the 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.

Note:

The signed image feature is not provided in SPSDK for KW45 devices currently. 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 10. The signed image can be found in the path below:

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

- Use the Keys and Certificate files generated using the steps mentioned in Section 4.2 "Preparing the keys and certificate"" for signed image generation.

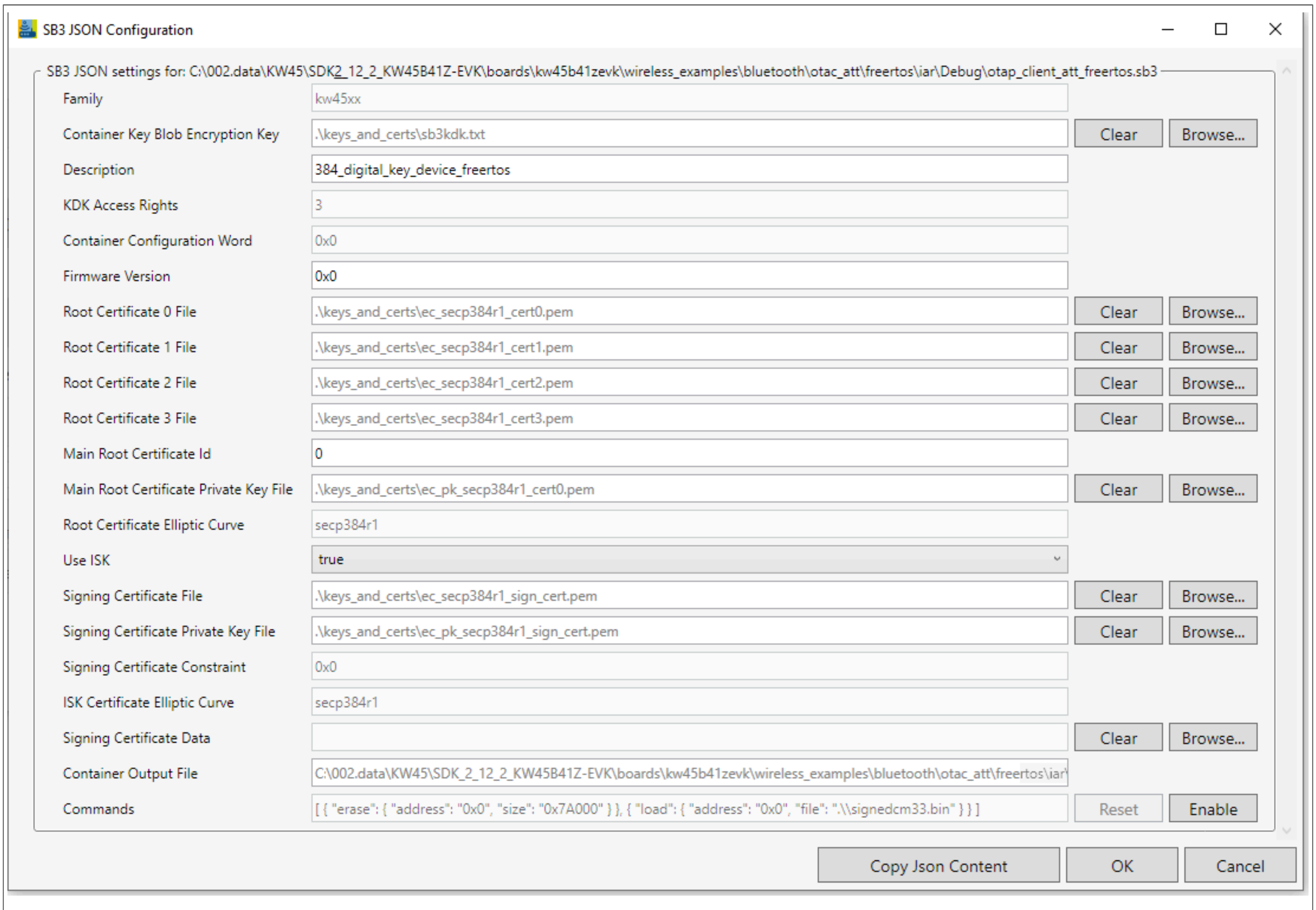

Figure 10. SB3 JSON configuration

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 11.

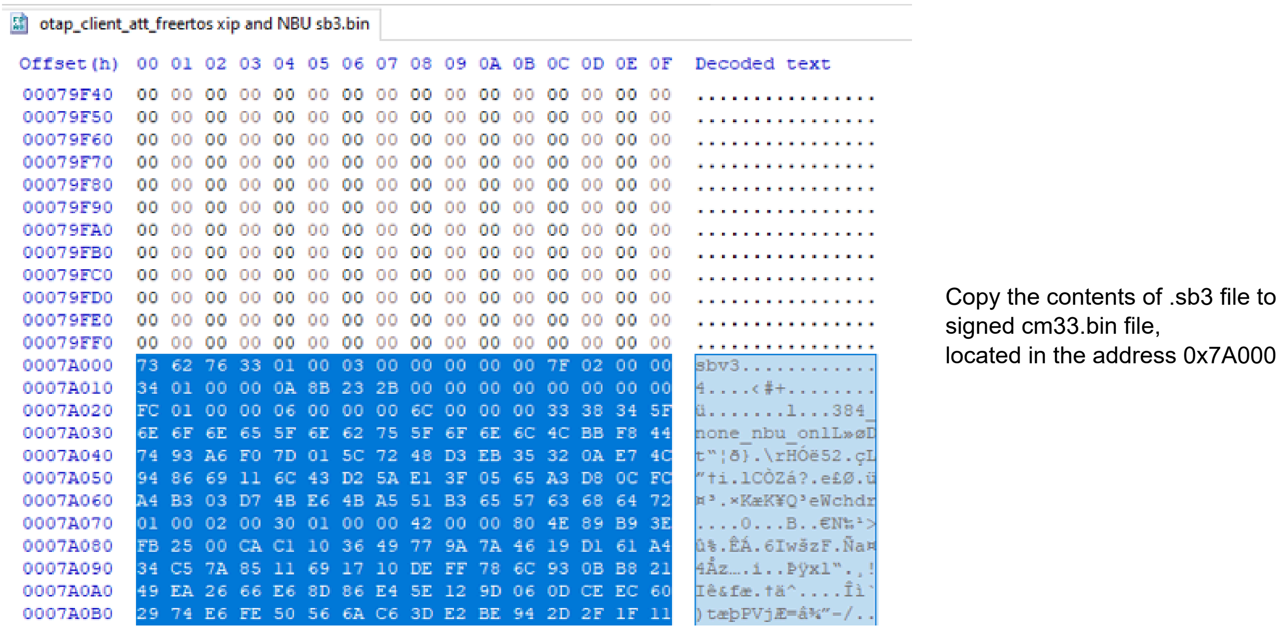

Figure 11. Merging the images

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 the debugger known as Debug Credential Key (DCK).

The method of generating DCK is the same as RoTKs generation in SPSDK Jupyter mentioned in Section 4.2 "Preparing the keys and certificate", shown in Figure 12. Users can use the same code and change the filename of the 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 12. Key pair generation

6.2 Debugging credentials

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

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

After successful creation, the message shown below is displayed:

```
The configuration template file has been created.
```

```
(venv) C:\Users\nxpdebugmbox get-template -f C:\kw45_debug_auth\test\test1.yml
The configuration template file has been created.
```

Figure 13. 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
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 the Configuration template:

```
nxpdebugmbox -p 2.1 gendc -c C: \XXXX\...\XXXX\test1.yml C: \YYYY\...\YYYY
\test.cert1
```

where

- <XXXX> denotes the user directory path and
- <YYYY> denotes the path of test.cert1 file

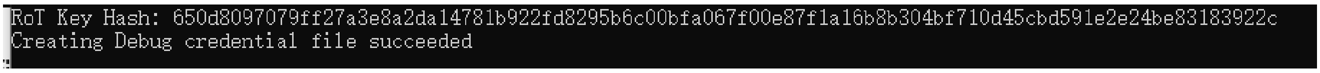

Figure 14. Debug credential generation

6.3 Initiating debug authentication

1. In the virtual environment, start the NXP debug box tool by using the command:

   ```
   nxpdebugmbox start
   ```

   Refer Figure 15.
2. Then, select J-Link as the debug probe.

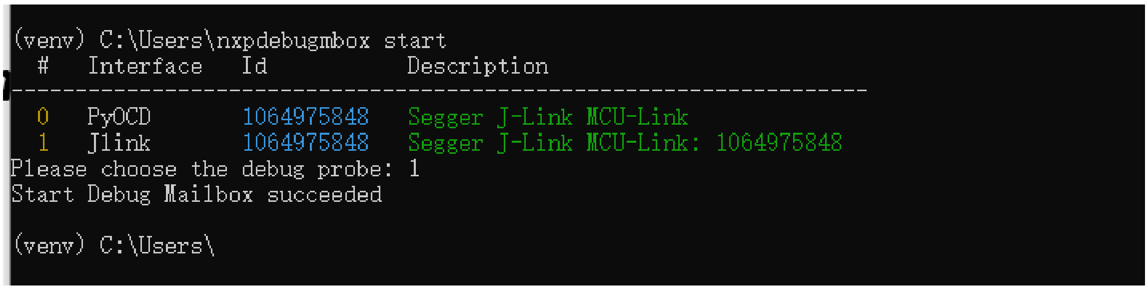

Figure 15. Initiating debug using the '`nxpdebugmbox start`' command

3. Use the command below to initiate debug authentication (refer Figure 16):

   ```
   nxpdebugmbox -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 "Preparing keys and certificates for debug".

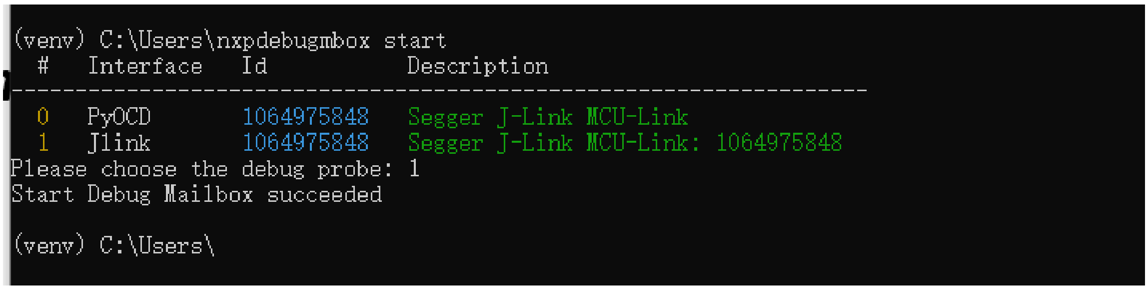

Figure 16. Debug authentication

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 17.

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 17. Secure Debug enabled

Note:

Access is disabled when the KW45 device is reset. Therefore, if KW45 is reset, debug authentication must be performed again. In such a case, do not use the IDE to perform debug authentication. This is because the reset KW45 operation might be embedded in the IDE.

7 Acronyms and abbreviations

Table 1 lists the acronyms used in this document.

Table 1. Acronyms and abbreviations

Acronym	Description
AES	Advanced Encryption Standard
Bluetooth LE	Bluetooth Low Energy
DCK	Debug Credential Key
IDE	Integrated Design Environment
ISP	In-System Programming
NBU	Narrow Band Unit
OEM	Original Equipment Manufacturer
RoTKTH	Root of Trust Key Table Hash
SB3KDK	SB3 Key Derivation Key
SDK	Software Development Kit
SPSDK	Secure Provisioning SDK
SWD	Serial Wire Debug
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*. Contact your local NXP field applications engineer (FAE) or sales representative for obtaining this document.

9 Revision history

Table 2 summarizes the revisions to this document.

Document revision history

Revision history		
Document ID	**Release date**	**Description**
AN14003 v.1.1	14 June 2024	Minor updates in Section 5.2.2 "Writing OEM keys in application code by nboot API"
AN14003 v.1.0	16 August 2023	Initial public release

10 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Legal information

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.

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 https://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.

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**	**3**
3	**Debug session initiation**	**4**
4	**Preparing OEM keys and certificate using SPSDK**	**5**
4.1	Setting up the SPSDK environment	5
4.2	Preparing the keys and certificate	6
5	**Mass processing and image preparation**	**8**
5.1	NBU image preparation	8
5.1.1	SB3 file generation	8
5.1.2	Changing the lifecycle (optional)	9
5.2	CM33 image preparation	10
5.2.1	Default OEM keys on KW45B41Z board	10
5.2.2	Writing OEM keys in application code by nboot API	11
5.2.3	Updating OTA update configurations	12
5.3	CM33 signed image generation (optional)	14
5.4	Merging CM33 image and NBU image	15
6	**Debug authentication**	**16**
6.1	Preparing keys and certificates for debug	16
6.2	Debugging credentials	16
6.3	Initiating debug authentication	17
7	**Acronyms and abbreviations**	**20**
8	**References**	**21**
9	**Revision history**	**22**
10	**Note about the source code in the document**	**23**
	Legal information	**24**