AN14003

Programming the KW45 Flash for Application and Radio Firmware via Serial Wire Debug During Mass Production

Rev. 2.0 — 15 December 2025 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 the 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.

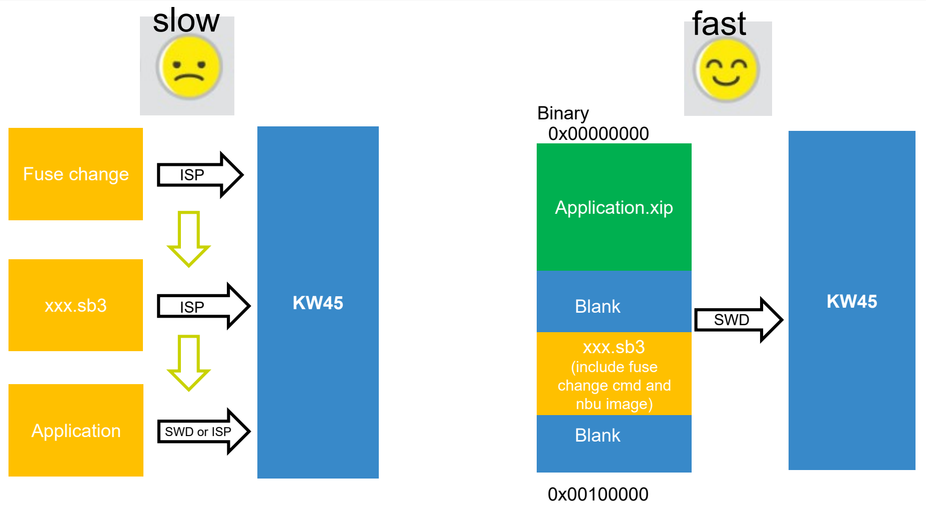


Figure 1. Burning image via SWD

Note: *The method of burning the fuse provided in this document cannot be reversed. The keys programmed to fuses on the 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* (document KW45RM).

In brief, the process implements fuse programming and updates NBU in the CM33 image. Then the CM33 image and NBU image are merged into a single image. After downloading the merged image to KW45 flash of CM33 via SWD, first program the fuse and then burn the image to NBU. The flow is shown in Figure 2.

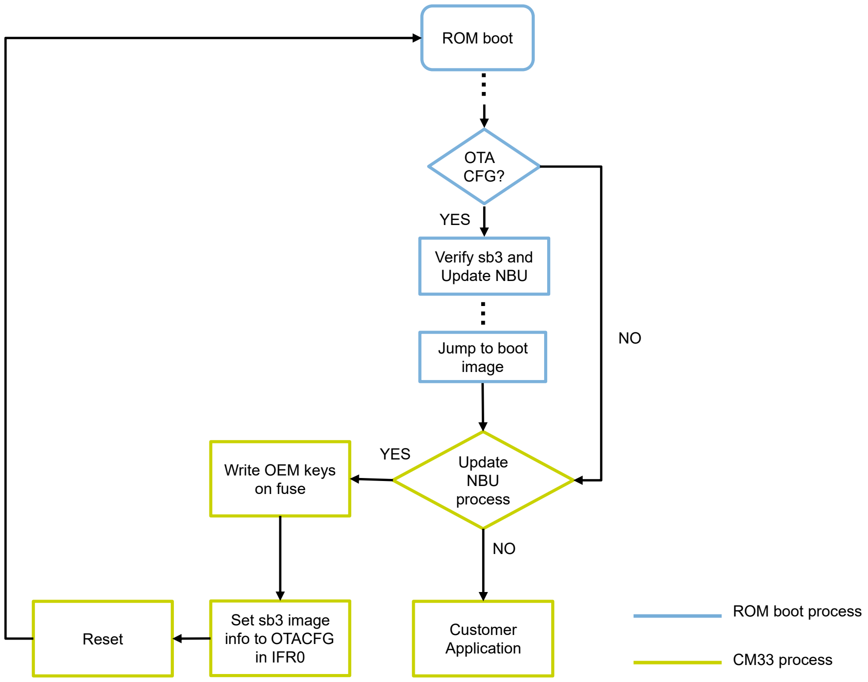


Figure 2. Process flow used

Two limitations for the method are as follows:

- 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 initiates without performing debug authentication.
- For a lifecycle that requires debug authentication, the debug session initiates 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 SEC tool environment

MCUXpresso Secure Provisioning Tool (SEC) is a graphical user interface (GUI) application designed to streamline the creation and provisioning of bootable executables for NXP microcontroller (MCU) platforms. The SEC tool is implemented based on SPSDK, which is an open source Python SDK library with its source code released on Github and PyPI.

The SEC tool is a public tool downloaded from the NXP website and installed on your PC. This document considers the SEC tool on the Window system as an example.

If the user does not already have a workspace, use the **File > New Workspace** menu to create a workspace quickly and generate all required keys.

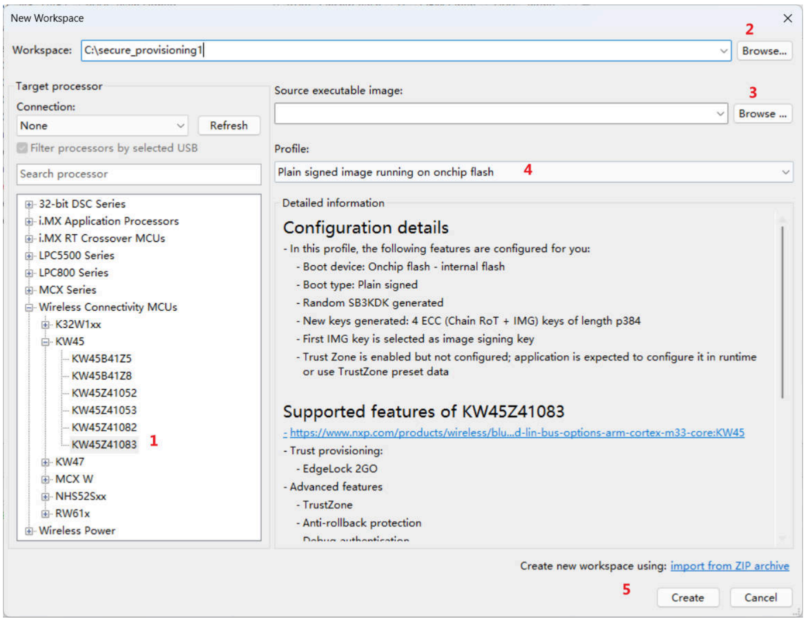


Figure 3. SEC tool create new workspace

To create a workspace, perform the steps below:

1. Select the part number that you are using.
2. To select a location to save the workspace files, use the **Browse** button.
3. Optionally, to select the path to the application image to load into the MCU, use the **Browse** button.
4. Optionally, select one of the signed image options for the profile type.
5. To generate the workspace, click **Create**.

4.2 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 fails.

A default value of RoTKTH and SB3KDK is provided for the KW45B41Z-EVK board. This document describes a method to write these default keys to a KW45B41Z chip in which these keys in fuse are null.

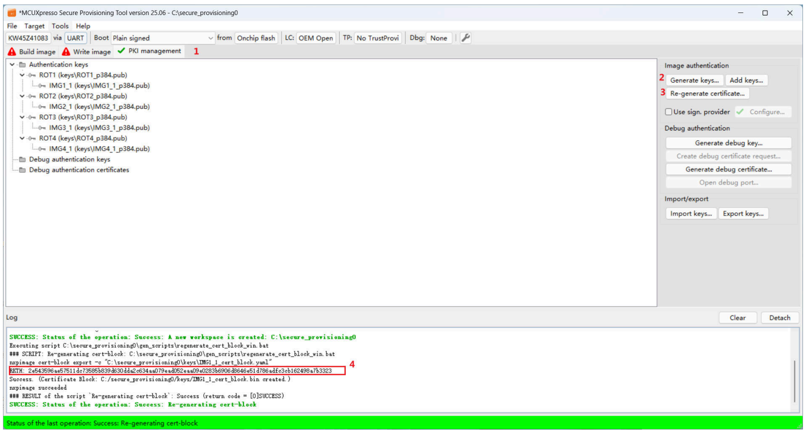


Figure 4. PKI management

To create keys, perform the steps below:

1. Select PKI management.
2. Create four keys with the certificate chain as Chain RoT + IMG and the key type is ECC P384.

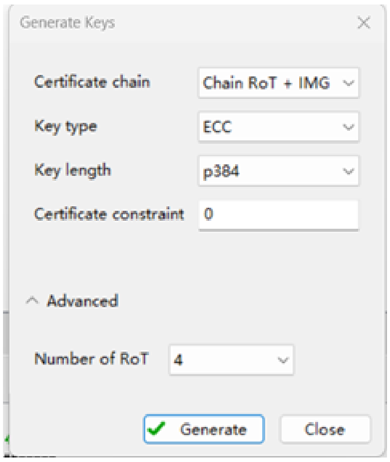


Figure 5. Generate keys

3. Optionally, if the user already has keys but no certificate, they can import the keys and then regenerate the certificates.
4. The ROTKTH can be observed in the log window of the SEC tool.
5. SB3KDK can be generated and observed in Figure 6.

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

SEC Tool generates the SB3 file.

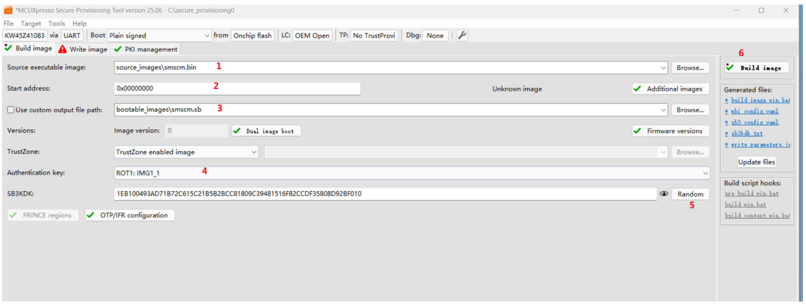


Figure 6. SB3 file generation configuration

To generate the SB3 file, perform the steps below:

1. To select the binary path to be signed or generated as a *.sb3 file, use the **Browse** button.
2. For application image, set the start address to the KW45 main core flash. For NBU image (*.xip), set the start address to 0x48800000 and also change the configure file (`sb3_config.yaml`).
3. To select a location for saving the *.sb3 file, use the **Browse** button.
4. Select a ROTK generate in PKI management, as an authentication key.
5. Generate SB3KDK.
6. To generate the *.sb3 file, press the **Build** button.

To generate the SB3 file by using a signed binary (*.xip), click **Tool** on the menu bar. Then select **SB Editor** and perform the steps below:

1. Select the **erase** command. The **load** command operates similarly to the **erase** command.
2. Specify the address and the size of the memory block to erase. For the **load** command, select the file path of signed NBU firmware (*.xip).
3. Select output path for **Configuration file** and **Secure binary file**.

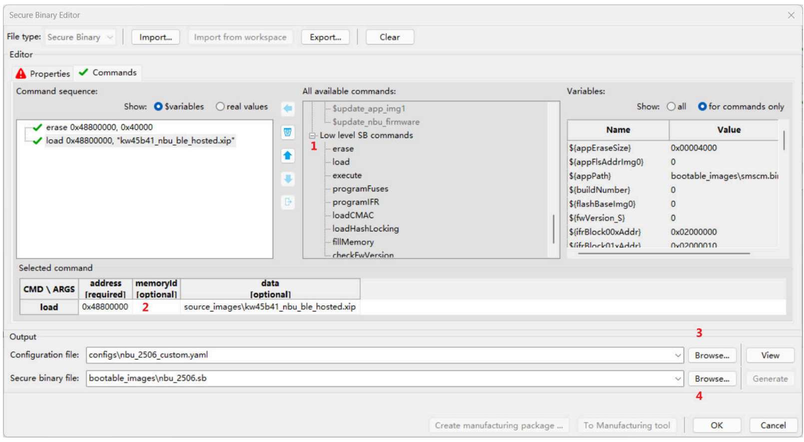


Figure 7. Add command on Secure Binary Editor

4. Select **Properties**.
5. Fill in the parameters as shown in Figure 8:
   - firmwareVersion: 0
   - certBlock: certification of ROTK generated in PKI management
   - family: kw45b41zb
   - signer: type=file;file_path= path of *.xip file
   - containerKeyBlobEncryptionKey: SB3KDK
   - description: KW45B41Z SB3
   - isNxpContainer: false
   - kdkAccessRights: 3
6. Generate the *.sb3 file.

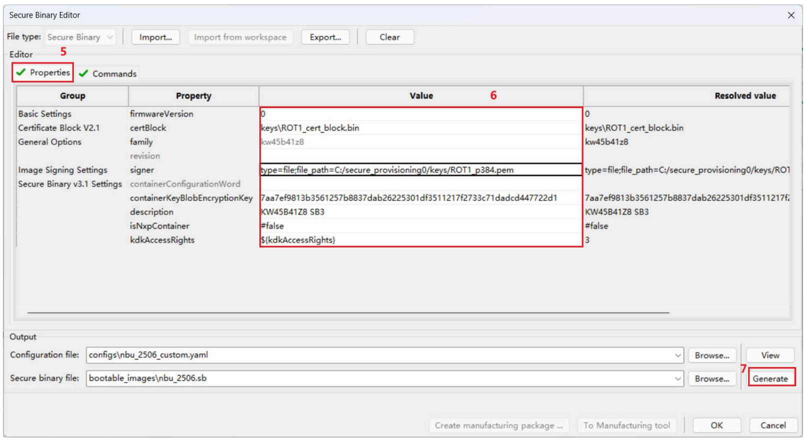
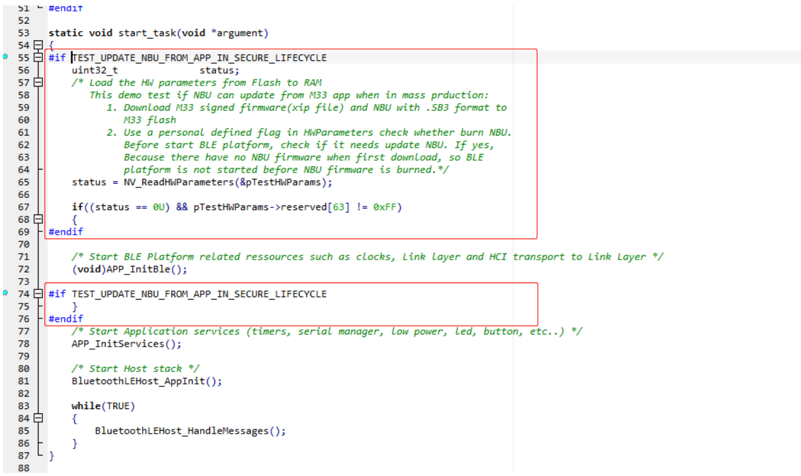


Figure 8. Add properties on Secure Binary Editor

5.2 CM33 image preparation

Consider SDK project `otac_att` as an example and set the `gEraseNVMLink_d` linker symbol to '0' while generating the binary file.

If the flash of NBU is empty, the Bluetooth LE example code is stacked when initializing NBU. Also an issue occurs when the Bluetooth LE host stack is initialized. So, use a flag stored in a non-volatile memory to indicate whether to perform the normal Bluetooth LE 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 9.

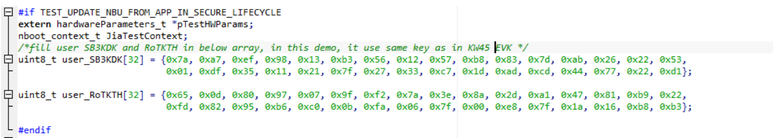


```
#endif

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 9. Burning NBU flag

5.2.1 Default OEM keys on KW45B41Z board

For the KW45B41Z-EVK board, NXP provides the default keys in fuse shown in Figure 10. However, for the KW45 chip received from the factory, by default, the SB3KDK and RoTKTH keys in the fuse are null. Therefore, writing these two keys to fuse is essential. Otherwise, the SB3 update fails.



```
#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 10. 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. To enable the nboot API, perform the steps below:

1. In the Bluetooth LE 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 can 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 10.
   The function code below shows how to program the SB3KDK and RoTKTH to the fuse:

   ```
   int Test_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 used for updating both CM33 and NBU firmware. For example, it indicates and provides metadata information for updating KW45 IFR0 OTACFG page. Refer to OTA update configuration in *KW45 Reference Manual* (document KW45RM). The target image is the NBU image prepared in Section 5.1 "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*/
            Test_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 previously generated *.sb3 file, it signifies that after the 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 performed, 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 is generated when generating *.sb3 file. User must check the output file path in* `mbi_config.yaml` *file in the SEC tool.*

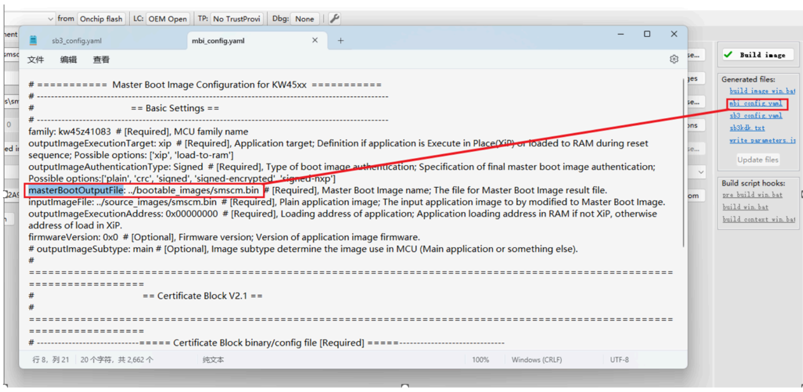


Figure 11. Configuration to generate signed image

5.4 Merge 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 12.

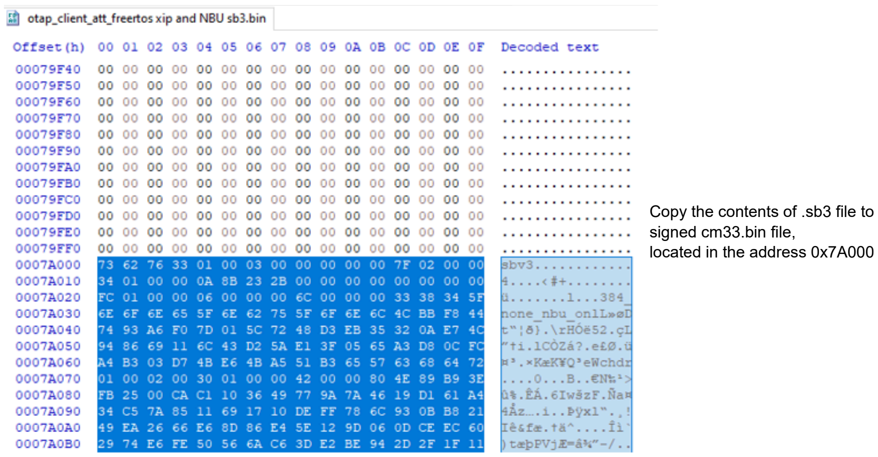


Figure 12. Merging the images

After merging the 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 the 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 in the SEC tool is as below:

1. To generate a debug key, select the **Generate debug key** button.
2. Select the ECC P384 key type and then generate the key.

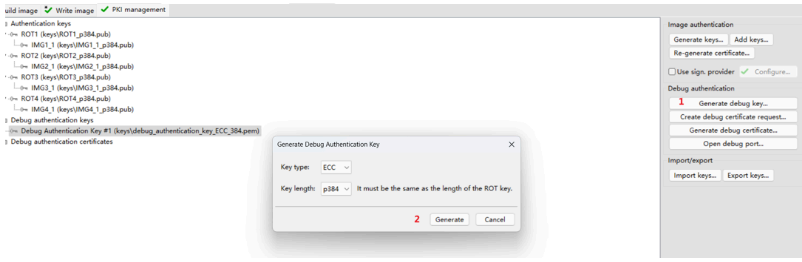


Figure 13. Key pair generation

6.2 Debug credentials

In the PKI management of SEC tool, to start debug credential generation, select the **Create debug certificate request…** button.

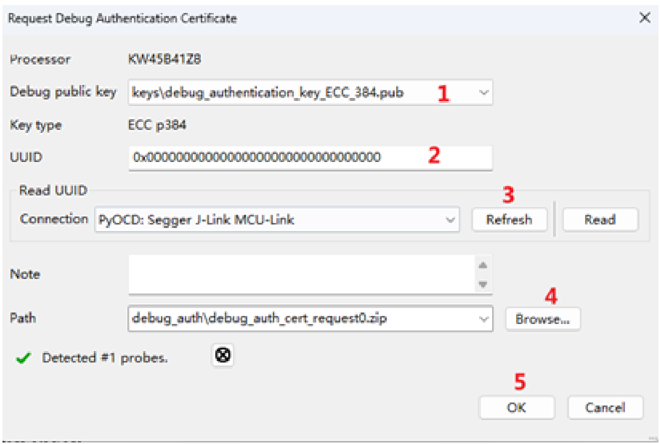


Figure 14. Create debug certificate request

1. Select the key generated before.
2. Set the UUID as '0'.
3. If the KW45 board is connected with the debugger, then use the **Refresh** button to scan the debugger.
4. To select a location to save the zip file, use the **Browse** button.
5. Generate a debug certificate request zip file.

6. To generate the debug certificate, click the **Generate debug certificate…** button.

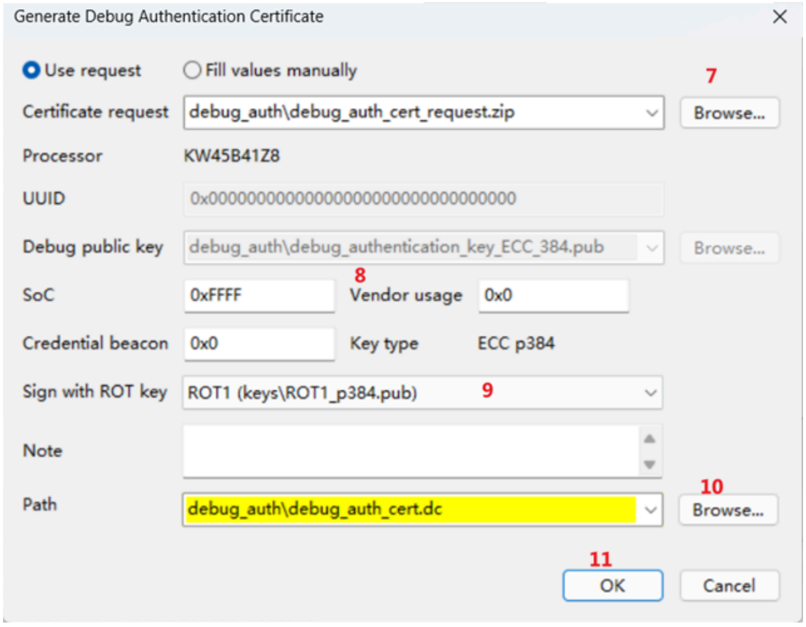


Figure 15. Debug certificate generation

7. To select a zip file generated in Step 5, use the **Browse** button.
8. Ensure that the configurations are same as shown in Figure 15.
9. To generate *.sb3 file, select an ROTK used.
10. To store the **Debug certificate** file, use the **Browse** button to select the location.
11. To generate the **Debug certificate file**, click the **OK** button.

6.3 Initiating debug authentication

In the SEC tool, on the PKI management page, select the **Open debug port…** button.

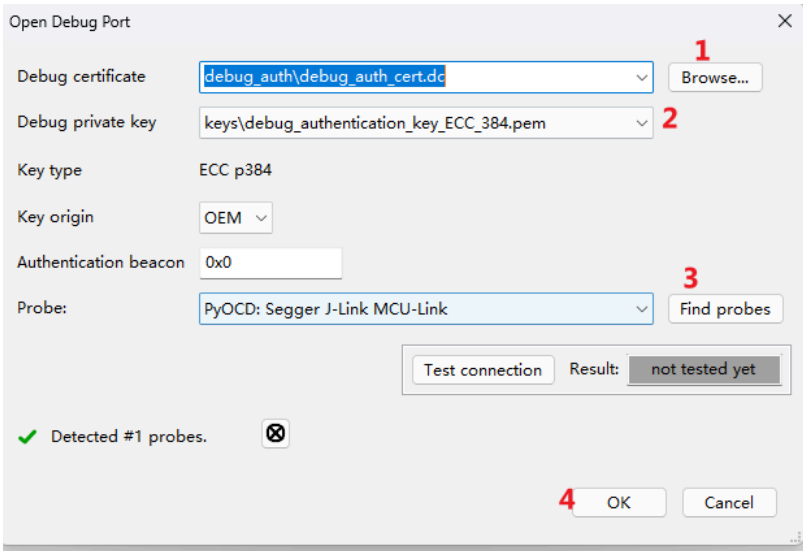


Figure 16. Initiating debug

1. Use the **Browse** button to select the Debug certificate generated in Section 6.2 "Debug credentials".
2. Select the debug private key generated in Section 6.1 "Preparing keys and certificates for debug".

3. If a KW45 board with a debugger is connected with a PC, then it can find the probe.
4. To start debug authentication, click the **OK** button.

Figure 17. Debug authentication

After debug authentication ends successfully, the device enables access to the debug domains permitted in the DC. Do not reset KW45, instead use J-Link commander to connect to the device directly. The log is shown in Figure 18.

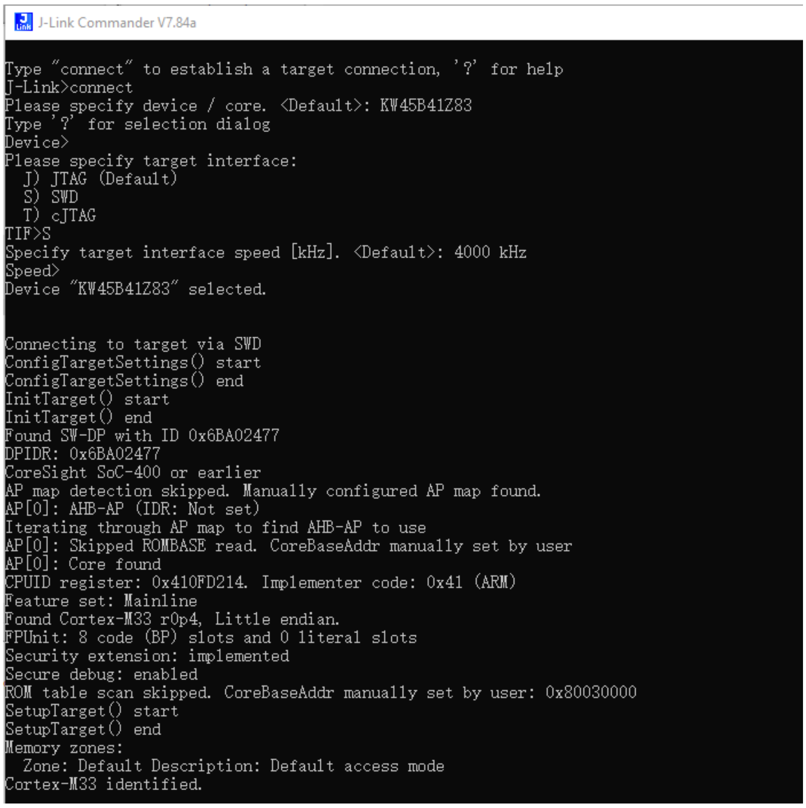


Figure 18. Secure Debug enabled

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 because the reset KW45 operation can be embedded in the IDE.

7 Acronyms

Table 1 lists the acronyms used in this document.

Table 1. Acronyms

Acronym	Description
AES	Advanced Encryption Standard
Bluetooth LE	Bluetooth Low Energy
DCK	Debug Credential Key
IDE	Integrated Design Environment
ISP	In-System Programming
OEM	Original Equipment Manufacturer
NBU	Narrow Band Unit
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

For more information, refer to the below documents:

- *Managing Lifecycles on KW45 and K32W148* (document AN13931)
- *Secure Boot for KW45 and K32W* (document AN13838)
- *KW45 Reference Manual* (document KW45RM)
- Bluetooth Low Energy Demo Applications User Guide

9 Note about the source code in the document

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

Copyright 2023-2025 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.

10 Revision history

Table 2 summarizes the revisions to this document.

Revision history

Document ID	Release date	Description
AN14003 v.2.0	15 December 2025	• Updated Section 4 "Preparing OEM keys and certificate using SPSDK" • Updated Section 5.1.1 "SB3 file generation"
AN14003 v.1.1	14 June 2024	Minor updates in Section 5.2.2
AN14003 v.1.0	16 August 2023	Initial public release

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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

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

Date of release: 15 December 2025
Document identifier: AN14003