AN13813

Secure boot on RW61x

Rev. 6.0 — 19 November 2024 Application note

Document information

Information	Content
Keywords	Key provisioning, secure boot image, plain image, signed images, signing certificate keys, SPSDK tool, ISP mode, boot ROM flash driver, OTP fuseword, OTP fields, configuration
Abstract	Describes how to generate and run the secure boot (signed image) on RW61x.

1 Introduction

This application note describes how to generate and run the secure boot (signed image) on RW61x using the secure provisioning SDK (SPSDK) tool.

Note: *The examples described in this document refer to SPSDK version 2.1.0.*

2 SPSDK tool

The SPSDK tool is a package of Python-based scripts used for secure key provisioning. The tool is available for download from NXP website [4]. Read more about SPSDK in [5].

SPSDK supports the following:

- Generating the keys using the *nxpkeygen* tool.
- Generating the signed images using the *elftosb* tool.
- Generating the secure firmware-update files in the SB3.1 format using the *elftosb* tool.
- Using the bootloader to perform a firmware update with the *blhost.exe* tool.
- Enabling the debug port using the *nxpdebugmbox* tool.

2.1 Install SPSDK

You must have a supported Python version installed for further steps.

To install the SPSDK, create a virtual Python environment using the console command window:

1. Create a virtual environment:

```
python -m venv <name> (e.g. venv)
```

2. Activate the virtual environment:

```
<name>\Scripts\activate (for windows)
<name>/bin/activate (for linux, mac)
```

   Make sure that your prompt starts with `(<name>)`. For example `(venv) c:\_projects\`.

3. Install SPSDK from Github into your Python virtual environment:

```
pip install -U spsdk
```

Do not close the command window when the virtual environment is active.

3 Prepare the secure boot image

The device can be configured to boot plain images during development. In this case, the ROM does not check the image to be booted, or the ROM only performs CRC32 checking, depending on the configuration. The binary file generated from the IDE is a plain image for non-secure boot. When secure boot is enabled, the plain image must be signed using the steps described in the following sections.

3.1 Plain image structure

Unsigned plain CRC images are supported as long as secure boot is not enforced. Preferably, this is limited to the development Life-Cycle state (Develop).

The structure of unsigned CRC images is shown in Figure 1.

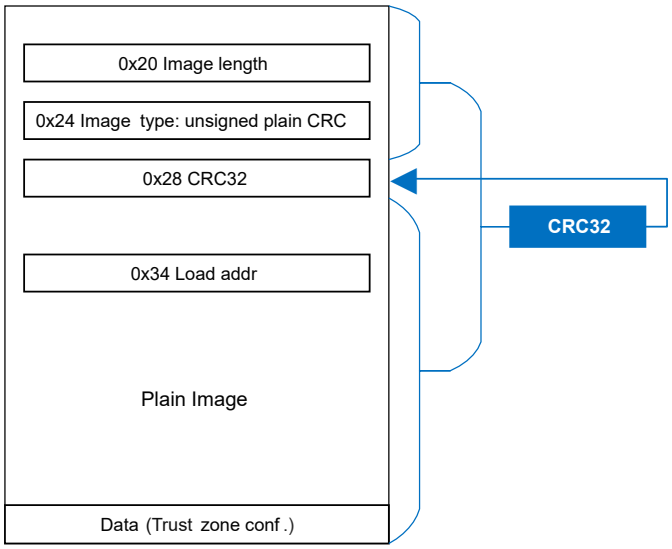

Figure 1. Structure of unsigned CRC images

Note: *When the image type is 0x0, CRC32 checking is bypassed. Such an image can be used as a generic image during development.*

3.2 Signed image structure

Images are signed using the ECDSA P-256 or P-384 algorithm. The digest is computed using SHA-256 for ECDSA P-256 signed images and SHA-384 for ECDSA P-384 signed images.

Figure 2 shows the structure of signed images.

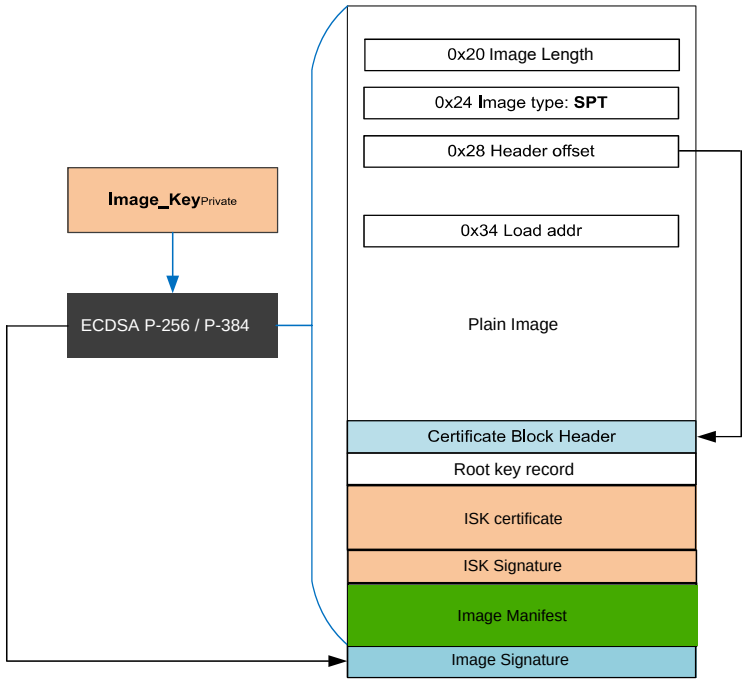

Figure 2. Structure of signed images

Image length - total length of the image in bytes including signature Image type - SPT (Signed Plain Text)= 0x4 or 0x5

Table 1. Image type (word at offset 0x24)

31:14	Reserved	Set to 0
13	TZ-M preset	0: No TZ-M peripherals preset. 1: TZ-M peripherals preset. The bootloader configures TZ-M related peripherals based on data stored in extended header
12:8	Reserved	Set to 0
7:0	Image type	0x0: plain image 0x2: plain image with CRC 0x4: Xip plain signed 0x5: Xip plain with CRC 0x6: SB3 manifest 0x7: NXP reserved for provisioning 0x8: NXP reserved for provisioning Other values are reserved.

Header offset – A 32-bit offset stored in a variable called offsetToCertificateBlockInBytes. The offset is calculated from the beginning of the signed image until the start of the certificate block header. The variable offsetToCertificateBlockInBytes must reside at offset 0x28 from the start of the signed image. An executable code image in the flash or RAM must start with an NVIC vector table. The word at offset 0x28 in NVIC vector table is a reserved slot for offsetToCertificateBlockInBytes.

For example, if an image resides in the flash at a non-zero address of 0x00008000, and the image certificate block header is at address 0x00024000, the NVIC vector table is located at 0x00008000 and the value at 0x00008028 contains offsetToCertificateBlockInBytes = 0x1C000.

Table 2 shows a standard Cortex-M33 NVIC vector table with the offset to the certificate block header highlighted.

Table 2. Cortex-M33 NVIC vector table

Offset (hex)	Usage
0	Initial SP
4	Reset
8	**NMI**
C	HardFault
10	MemManage
14	BusFault
18	UsageFault
1C	*Reserved*
20	Image Length
24	Image Type
28	**offsetToCertificateBlockInBytes**
30	SVC
34	DebugMon
38	*Reserved*
3C	SysTick

3.3 SB3.1 image structure

The Secure Binary (SB) container brings a secure and easy way to upload or update the firmware in the embedded device. The SB container in version 3.1 (SB3.1) uses the latest cryptographical algorithms to guarantee the authenticity and confidentiality of the carried firmware. The security level of SB3.1 is configurable. SB3.1 has the added possibility to reach Commercial National Security Algorithm Suite (CNSA) level of security based on project performance/boot time vs security requirements. The digital signature based on Elliptic Curve Cryptography (ECC) ensures the authenticity of SB3.1 container. And the Advance Encryption System (AES) in Cipher Block Chain (CBC) mode ensures the confidentiality of SB3.1 container.

SB3.1 is characterized as a chain of blocks (Figure 3), and each block i contains hash digest of block i+1.

Besides the hash digest of block 1, the block 0 also contains the digital signature. The digital signature guarantees the authenticity of the hash digest of block 1 and therefore the whole chain. The verification of block 0 digital signature is followed by the gradual verification of the next hashes and blocks in the chain. This way, the authenticity of the whole SB3.1 chain is verified. The last block in the chain (block N) contains only zeroes instead of hash digest value. For more details, refer to *Secure boot ROM* section in [2].

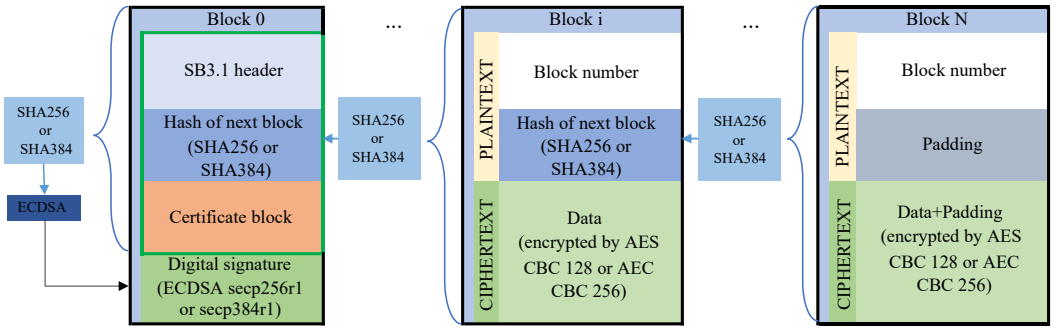

Figure 3. SB3.1 structure

3.4 Generate the signing certificate keys

The device supports up to four root of trust keys (RoTK) for different authentication purposes. Besides the main boot image authentication, the individual RoTK keys can be used for debug authentication or for firmware update authentication.

3.4.1 Certificate block

This section describes the generation of key pairs. *elftosb* tool is used to generate the final certificate which is attached to the image.

The certificate block is a concatenation of:

- The certificate block header
- The root key record
- The optional image signing key (ISK) certificate
- The ISK signature

The certificate block can reside at the end of the signed data, but it must be fully contained within the signed data, such that the certificate block itself is signed.

About ISK certificate:

- If ISK is not used, the ISK certificate section is removed from the certificate block. One section from the provided root certificates is used for the signature of the whole Block 0 or boot image.
- If an ISK certificate is provided, one section from the provided root certificates is used to sign ISK public key, ISK private key, and the whole Block 0 or boot image.
- The ISK public key is part of the ISK certificate.

The ISK private key is used to sign the image for authentication. It needs to be provided as external input for ECDSA signature.

nxpkeygen tool is used to generate the key pair. The pair is based on ECDSA P-256 or P-384.

The image signature is:

- Attached to the end of the signed image.
- Always checked during the image boot. This is the basic authentication check.

The ISK signature is:

- An optional security feature built on top of the image signature.
- Based on the additionally generated key pair based on ECDSA principles.

When the ISK is enabled, the boot image authentication is a two-stage process, which means that the boot process verifies both the ISK signature and image signature.

3.4.2 Root of trust keys

The OEM generates the root of trust key hash (ROTKH) table once. The table is stored permanently in the OTP.

Table 3. ROTKH layout in OTP

Fuseword	Description	Fuseword	Description
104	ROTKH[383:352]	110	ROTKH [191:160]
105	ROTKH[351:320]	111	ROTKH [159:128]
106	ROTKH[319:288]	112	ROTKH [127:96]
107	ROTKH [287:256]	113	ROTKH [95:64]
108	ROTKH[255:224]	114	ROTKH [63:32]
109	ROTKH [223:192]	115	ROTKH [31:0]

ROTKH: For ECC P-256 keys, ROTKH is a 32-byte SHA-256 digest of four SHA-256 digests computed over four OEM public keys. OEM has four private-public key pairs in case one of the private keys becomes compromised. If ECC P-384 keys are used, ROTKH is a 48-byte SHA-384 digest. The size of the used hash determines the actual size of the ROTKH in OTP. For P-256 keys, the ROTKH size is only 32 bytes. In this case, ROTKH should be written starting from 104, but only ROTKH [383:352] up to ROTKH [159:128] are used. The remaining 16 bytes are unused and must be filled with zeros. Figure 4 illustrates the generation process.

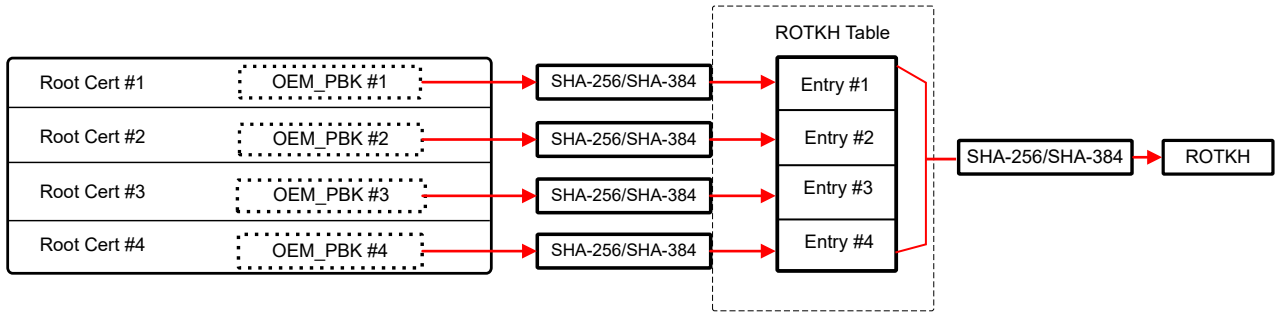

Figure 4. ROTKH generation process

If we denote OEM public keys as OEM_PBK1, OEM_PBK2, OEM_PBK3, OEM_PBK4, then ROTKH is computed as:

```
ROTKH = SHA-256(SHA-256(OEM_PBK1), SHA-256(OEM_PBK2), SHA-256(OEM_PBK3),
 SHA-256(OEM_PBK4))
```

Or

```
ROTKH = SHA-384(SHA-384(OEM_PBK1), SHA-384(OEM_PBK2), SHA-384(OEM_PBK3),
 SHA-384(OEM_PBK4))
```

The number of hashes of keys in the RKH table must range from one to four maximum. All the unused table entries must be set to zero. When searching the RKH table for a key hash, the loader stops at the first entry that is all zeroes.

The extra root public keys and root certificates must be created in advance and held in reserve for when a public key has to be revoked. The customer is responsible for implementing the mechanism to determine if a key must be revoked, and then set the appropriate ROTKH_ REVOKE bits. An authenticated connection with a server during a firmware update is often used for that purpose.

Note: *Only one of the root certificates which keys are listed in the RKH table are included in the certificate table at a time.*

3.4.3 Generate the keys

To generate the keys, use the *nxpkeygen* tool embedded in SPSDK. See Section 2.1.

Example of commands to generate the key pairs:

```
nxpcrypto key generate -k secp256r1 --force -o WORKSPACE\keys\ROT1_p256.pem"
nxpcrypto key generate -k secp256r1 --force -o WORKSPACE\keys\ROT2_p256.pem"
nxpcrypto key generate -k secp256r1 --force -o WORKSPACE\keys\ROT3_p256.pem"
nxpcrypto key generate -k secp256r1 --force -o WORKSPACE\keys\ROT4_p256.pem"
nxpcrypto key generate -k secp256r1 --force -o WORKSPACE\keys\IMG1_1_p256.pem"
```

The first four generated key pairs are the RoTK keys, which can be used for different purposes. One of the key pairs can be selected as the boot image signature key (see Section 3.4.1).

The last fifth key-pair generated can be used as the ISK keys.

3.5 SB3 processing keys

The bootloader uses the CUST_MK_SK key to decrypt a SB3.1 file. To process SB3.1 files, RFC3394 key blob key must be present in the OTP at fuseword 92. The key is generated and loaded during device provisioning. Device provisioning consists of two independent steps:

- Step 1: Generation of the secure provisioning image (*secure_provisisioning.sb*).
- Step 2: Execution of the secure provisioning image using devhsm loader application (*devhsm_loader.sb*).

3.5.1 Generate the secure provisioning image

This section shows how to generate the secure provisioning image (*secure_provisisioning.sb*) to provision CUST_MK_SK into OTP fuses securely. Details of trust provisioning are available in the subsection *Secure trust provisioning* of the section *Secure boot ROM* in [2].

Table 4. CUST_MK_SK layout in OTP

Fuseword	Description	Fuseword	Description
92	CUST_MK_SK[31:0]	98	CUST_MK_SK[223:192]
93	CUST_MK_SK[63:32]	99	CUST_MK_SK[255:224]
94	CUST_MK_SK[95:64]	100	CUST_MK_SK[287:256]
95	CUST_MK_SK[127:96]	101	CUST_MK_SK[319:288]
96	CUST_MK_SK[159:128]	102	CUST_MK_SK[351:320]
97	CUST_MK_SK[191:160]	103	CUST_MK_SK[383:352]

The SPSDK *nxpdevhsm* utility is used to generate the secure provisioning image, and provision CUST_MK_SK into OTP fuses securely.

The following items are necessary as inputs to generate the secure provisioning file:

- Customer main key – Customer Main Key Symmetric Key secret file (32-bytes long binary file). CUST_MK_SK (provisioned by OEM, known by OEM). This is a 256-bit pre-shared AES key provisioned by OEM. CUST_MK_SK is used to derive FW image encryption keys.
- OEM input share – OEM share input file to use as a seed to randomize the provisioning process (16-bytes long binary file).
- Config FILE – YAML/JSON configuration file containing the settings.

Step 1 – Generate the template of the configuration file.

```
nxpdevhsm get-template  -f rw61x --force -o WORKSPACE\secure_provisisioning.yaml
```

The output of the command is a template file for RW61x.

Step 2 – Update *secure_provisisioning.yaml* configuration file. One such example is shown below:

```
# ======================  DEVHSM procedure Secure Binary v3.1 Configuration template for rw61x.
 ======================

#
 ================================================================================================================================
#                                                 == Basic Settings ==

#
 ================================================================================================================================
# ----------------------------------------=====  Firmware version. [Optional]
 =====-------------------------------------
# Description: Value compared with Secure_FW_Version monotonic counter value stored in PFR/IFR. If value is lower
 than
# value in PFR/IFR, then is image rejected (rollback protection)..
firmwareVersion: 0
# ------------------------------------------------===== MCU family [Required]
 =====-----------------------------------------------
# Description: MCU family name.
# Possible options: <lpc55s3x, mc56f81xxx, mcxn9xx, mwct20d2x, rw61x>
family: rw61x
#
 ================================================================================================================================
#                                           == Secure Binary v3.1 Settings ==

#
 ================================================================================================================================
# ------------------------------------------=====  Description [Optional]
 =====-------------------------------------------
# Description: Description up to 16 characters, longer will be truncated. Stored in SB3.1 manifest.
description: This is description of generated SB file.
#
 ================================================================================================================================
#                                       == Secure Binary v3.1 Commands Settings ==

#
 ================================================================================================================================
# ------------------------------------------=====  SB3.1 Commands [Required]
 =====-------------------------------------------
# Description: Secure Binary v3.1 commands block, list of all possible options - Modify it according to your
 application
commands:
    # ---------------------------------------=====  Program Fuses [Required]
 =====-----------------------------------------
    - programFuses:  # [Required], Program Fuses; Address is OTP index of fuses to be programmed (Check the
 reference manual for more information). Values is a comma separated list of 32bit values.
        address: '45'  # [Required], Address; OTP Index of fuses to be programmed. Depends on the chip ROM.
        values: 0x0F0F # [Required], Binary values; 32bit binary values delimited by comma to be programmed.
    - programFuses:  # [Required], Program Fuses; Address is OTP index of fuses to be programmed (Check the
 reference manual for more information). Values is a comma separated list of 32bit values.
        address: '0x68'  # [Required], Address; OTP Index of fuses to be programmed. Depends on the chip ROM.
        values: 0x1, 0xCFCBDFD8, 0xC12344F3, 0x9758001F, 0xCE988C4F, 0xC556A79A, 0x786B4167, 0x45165DB4 #
 [Required], Binary values; 32bit binary values delimited by comma to be programmed.
    - programFuses:  # [Required], Program Fuses; Address is OTP index of fuses to be programmed (Check the
 reference manual for more information). Values is a comma separated list of 32bit values.
        address: '15'  # [Required], Address; OTP Index of fuses to be programmed. Depends on the chip ROM.
        values: 0x00980000 # [Required], Binary values; 32bit binary values delimited by comma to be programmed.
    - programFuses:  # [Required], Program Fuses; Address is OTP index of fuses to be programmed (Check the
 reference manual for more information). Values is a comma separated list of 32bit values.
        address: '400'  # [Required], Address; OTP Index of fuses to be programmed. Depends on the chip ROM.
        values: 0x131 # [Required], Binary values; 32bit binary values delimited by comma to be programmed.
```

In the above example, CUST_MK_SK and other OTP fuses related to secure boot are configured to be programmed.

NXP recommends the following order of provisioning:

1. Program lifecycle OTP fuse.
2. Program trust anchor ROTKH OTP fuses.
3. Program secure boot OTP fuse.
4. Program other non-secure OTP fuses if any.
5. Program the image encryption key into CUST_MK_SK OTP fuses.
6. Program the application to flash if necessary.

Step 3 – Set the device in ISP boot mode.

To generate the secure provisioning image, RW61x must be configured in ISP boot mode (boot from peripherals or handling of commands from a peripheral interface). For details about ISP boot refer to Section 4.

In this document ISP boot mode via UART is taken as an example.

Step 4 – Generate the secure provisioning image.

```
nxpdevhsm generate -p %comport% -k "cust_mk_sk.bin" -i "oem_share.bin" -w "nxpdevhsm" -f rw61x -o
 secure_provisisioning.sb -c secure_provisisioning.yaml
where:
cust_mk_sk.bin -> Customer Main Key Symmetric Key secret file (32-bytes long binary file).
 CUST_MK_SK (provisioned by OEM, known by OEM). This isa 256-bit pre-shared AES key provisioned by
 OEM. CUST_MK_SK is used to derive FW image encryption keys.
oem_share.bin  -> OEM share input file to use as a seed to randomize the provisioning process (16-
bytes long binary file).
dev_hsm_provi_sb3.yaml -> Config file from Step 2

Utility provided in SPSDK tool can be used to generate binary file an input file:
for example:
cust_mk_sk_text = 000102030405060708090a0b0c0d0e0f00112233405060708090a0b0c0d0e0f0
oem_share_text  = 12345678912345678912345678912345

nxpimage utils convert hex2bin -i cust_mk_sk_text.txt -o cust_mk_sk.bin

nxpimage utils convert hex2bin -i oem_share_text.txt -o oem_share.bin
```

Example of output log:

```
 1: Initial target reset is disabled
 2: Generating OEM main share.
 3: Generating 48 bytes FW signing keys.
 4: Generating 48 bytes FW encryption keys.
 5: Wrapping CUST_MK_SK key.
 6: Creating template un-encrypted SB3 header and data blobs.
 6.1: Creating template SB3 header.
 6.2: Creating un-encrypted SB3 data.
 7: Encrypting SB3 data on device
 7.1: Enriching encrypted SB3 data by mandatory hashes.
 7.2: Creating dummy certificate block.
 7.3: Updating SB3 header by valid values.
 7.4: Preparing SB3 manifest to sign.
 8: Creating SB3 manifest signature on device.
 9: Composing final SB3 file.
10: Resetting the target device
Final SB file has been written: secure_provisisioning.sb

The command reslults:
      secure_provisisioning.sb -> Secure provisioning image
      nxpdevhsm -> Output directory containing  temporary files that could be used for review
```

Note: *Support for the generation of the secure provisioning image for RW61x is added in SPSDK version 2.1.0. Make sure to use version 2.1.0 and above.*

3.5.2 Execute the secure provisioning image

This section describes how to execute the secure provisioning image (*secure_provisisioning.sb*). Use the NXP utility to load an application image and execute the secure provisioning image. NXP DevHSM loader image is part of the restricted data package for SEC Tool (v8.1 and above) and can be downloaded from [3].

Example of use case:

1. The OEM creates a secure provisioning image in their secure environment.
2. The OEM ships the devices and secure provisioning image to contract manufacturers.
3. The contract manufacturers download NXP DevHSM loader application image from the link. Or the OEMs deliver NXP DevHSM loader application image along with the provisioning image).
4. The contract manufacturers perform device provisioning using the secure provisioning image from step 1 in their production environment.

Step 1 - Set the device in ISP boot mode.

Or continue from step 4 of Section 3.5.1.

For details about ISP boot, refer to Section 4. In this document ISP boot mode via UART is taken as an example.

Step 2 - Execute the NXP DevHSM loader application.

NXP DevHSM loader application is used for the secure provisioning image in the production environment of the contract manufacturer. The image was originally generated in a secure OEM environment.

```
blhost.exe -p COM22,115200 -t 60000 -- receive-sb-file RW61x_DevHSM_Loader_FW.sb3
Sending SB file  [##################################]  100%
Response status = 0 (0x0) Success.
```

Step 3 - Check the version of the DevHSM loader application (optional).

```
blhost.exe -p COM22,115200 -t 60000 -- get-property 0x1
Response status = 0 (0x0) Success.
Response word 1 = 1157694208 (0x45010300)
Current Version = E1.x.x
```

Step 4 - Execute the secure provisioning image.

With the DevHSM loader application running, execute the secure provisioning image received from the OEM.

```
blhost.exe -p COM22,115200 -t 60000 -- receive-sb-file secure_provisisioning.sb
```

CAUTION:

1. *Executing the secure provisioning image permanently programs CUST_MK_SK fuses (other OTP fuses as configured). The fuses cannot be reprogrammed. Save all input and temporary files for review.*
2. *For production, it is recommended to advance the Life-Cycle state to In-Field to avoid exposing sb3 processing keys.*
3. *Before programming secure boot, and ROTKH fuses, it is advised to test them using the shadowregs utility in SPSDK Section 6.2.*

3.6 Prepare the main boot signed image

To prepare a main boot signed image, generate:

- The plain binary (without boot header)
- The signed image
- The flash configuration block (FCB) image
- The main boot image

3.6.1 Use the SPSDK tool

The SPSDK tool includes the templates for the configurations in YAML format.

This section provides the commands based on *nxpimage* to use the templates. For details on *nxpimage*, refer to [6].

SPSDK *nxpimage mbi* tool uses nxpimage mbi configuration file as input to generate the signed image.

The configuration file specifies the generated image type as:

- Plain image (CRC)
- Signed image
- Encrypted image

The signed image contains the paths to up to four private keys. One of the private keys is set to sign the image.

If `useIsk` is set to `true` in nxpimage mbi configuration file, the ISK key pair is defined as a secondary security option.

Note: *`useIsk` is set to `true` in the context of this application note, and the respective certificates are defined in the .yaml file.*

The nxpimage mbi configuration file is used to generate the RKTH from the four public keys. See Section 3.4.

Generate the application image.

To use the SDK example from IAR, select **BOOT_HEADER_ENABLE = 0**. To use the SDK example with MCUXpresso, select the **"check plain load image"** option to generate the plain image without the boot header.

Commands for the signed image

- Generate a signed image:

```
nxpimage mbi export -c WORKSPACE\configs\mbi_config.yaml
```

Commands for FCB image

- Generate an FCB image:

```
nxpimage bootable-image fcb export -c WORKSPACE\configs\mbi_config.yaml
\bootimg_rw61x_flexspi_nor.yaml WORKSPACE\output\fcb.bin
```

Commands for the main boot image

- Generate the bootable image (join fcb and signed image):

```
nxpimage bootable-image merge -c bootimg_rw61x_flexspi_nor.yaml -o masterboot_signed.bin
```

3.6.2 Generate the signed image

This section shows how to use the SPSDK nxpimage mbi utility to generate a signed image.

Step 1 - Generate the certification block template

```
nxpimage cert-block get-template -f rw61x -o rw61x WORKSPACE\configs
\cert_block_template.yaml --force
```

Step 2 - Update the certification block template and save as *cert_block.yaml*. One example is shown below:

```
# ===============  YAML configuration for Certification Block V21  ================
# ================================================================================
#                                 == ISK (Image signing key) Certificate Settings ==
# ================================================================================
useIsk: False  # [Required], Use ISK for signature certification.
# ================================================================================
#                                              == Root Keys Settings ==
# ================================================================================
# ROT0:
rootCertificate0File: ../keys/ROT1_p256.pub  # [Required], Root Certificate File 0.
# ROT1:
rootCertificate1File: ../keys/ROT2_p256.pub  # [Required], Root Certificate File 1.
# ROT2:
rootCertificate2File: ../keys/ROT3_p256.pub  # [Required], Root Certificate File 2.
# ROT3:
rootCertificate3File: ../keys/ROT4_p256.pub  # [Required], Root Certificate File 3.
mainRootCertId: 0  # [Conditionally required] Index of root key that is used as a main.
# --------------------------------------------------------------------------------
```

Step 3 - Generate the image configuration templates.

Note: *It is not necessary to provide the template name.*

```
nxpimage mbi get-templates -f rw61x WORKSPACE\configs --force
```

The output of the command is the main boot image (mbi) configuration template for RW61x in the *WORKSPACE\configs* directory (Figure 5).

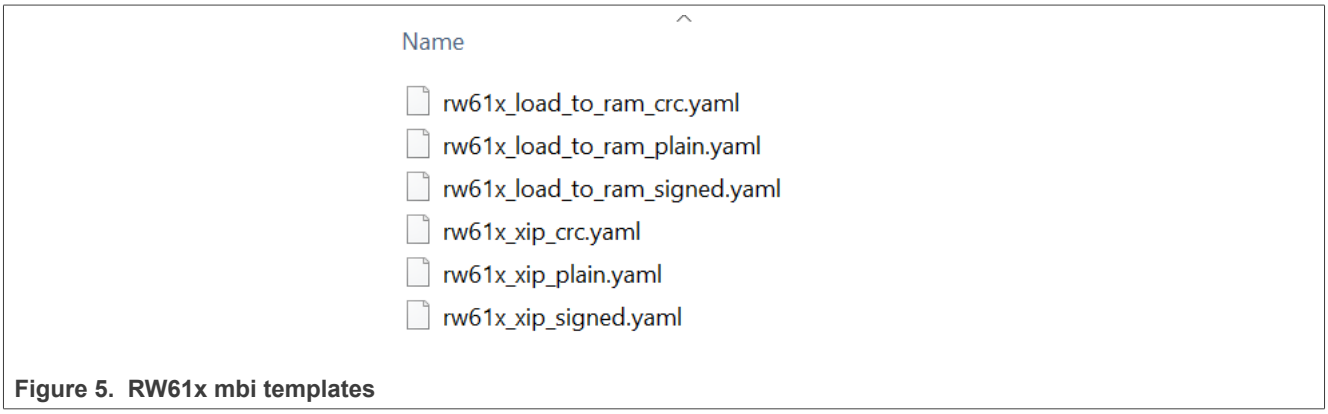

Figure 5. RW61x mbi templates

Step 4 - Open the *rw61x_xip_signed.yaml* file to view the settings for RW61x. Apply changes and save the file *mbi_config.yaml*.

One such example is shown below:

```
# ===========  Main Boot Image Configuration for RW61x  ===========
# ------------------------------------------------------------------------------------------------------
#                                        == Basic Settings ==
# ------------------------------------------------------------------------------------------------------
family: rw61x  # [Required], MCU family name
outputImageExecutionTarget: External flash (XIP)  # [Required], Application target; Definition if application is
 Execute in Place(XiP) or loaded to RAM during reset sequence; Possible options:['Internal flash (XIP)', 'External
 flash (XIP)', 'Internal Flash (XIP)', 'External Flash (XIP)', 'RAM', 'ram', 'xip']
outputImageAuthenticationType: Signed  # [Required], Type of boot image authentication; Specification of final
 main boot image authentication; Possible options:['Plain', 'CRC', 'Signed', 'Encrypted + Signed', 'NXP Signed',
 'encrypted', 'signed', 'crc']
masterBootOutputFile: WORKSPACE/app_signed.bin  # [Required], Main Boot Image name; The file for Main Boot Image
 result file.
inputImageFile: WORKSPACE/source_images/app.bin  # [Required], Plain application image; The input application image
 to by modified to Main Boot Image.
firmwareVersion: 0x1  # [Optional], Firmware version; Version of application image firmware.
outputImageExecutionAddress: 0x08001000 # [Required], Loading address of application; Application loading address
 in RAM if not XiP, otherwise address of load in XiP.#
# ======================================================================================================
#                                          == Certificate Block V2.1 ==
# ======================================================================================================
# ------------------------------===== Certificate Block binary/config file [Required] =====-------------------------
# Description: Path to certificate block binary or config file.
certBlock: cert_block.yaml
# ======================================================================================================
#                                          == Image Signing Settings ==
# ======================================================================================================
# ----------------------------------- Signature Provider [Conditionally required] ------------------------------
# Description: Signature provider configuration in format 'type=<sp_type>;<key1>=<value1>;<key2>=<value2>".
signProvider: type=file;file_path=../keys/ROT1_p256.pem

# ------------------------------------------------------------------------------------------------------
#                                      == Trust Zone Settings ==
# ------------------------------------------------------------------------------------------------------
enableTrustZone: true  # [Optional], TrustZone enable option; If not specified, the Trust zone is disabled.
```

Step 5 Generate the signed image.

```
nxpimage mbi export -c WORKSPACE\configs\mbi_config.yaml
- results in signed image app_signed.bin
```

- Use `-v` option to view the ROTKH generated when using the public keys used with any *.yaml* file.

Note: *Instead of using SPSDK to append an FCB image, user can also follow the steps in Section 7.2.*

3.6.3 Generate FCB image

This section shows how to use the SPSDK *nxpimage binary-image* utility to generate an image for the flash configuration block (FCB).

Step 1 - Generate the template.

Note: *It is not required to provide the template name.*

```
nxpimage bootable-image fcb get-templates -f rw61x -o WORKSPACE\fcb
```

The output of the command is *fcb_rw61x_flexspi_nor.yaml* file.

Step 2 - Edit the *.yaml* configuration file based on the flash model being used.

Step 3 - Generate the binary image for FCB.

```
nxpimage bootable-image fcb export -c WORKSPACE\fcb\fcb_rw61x_flexspi_nor.yaml -o
 WORKSPACE\fcb\rw61x_fcb.bin
```

3.6.4 Generate the main binary image

This section shows how to use the SPSDK *nxpimage* utility to generate the main binary image.

Step 1 – Generate the template.

Note: *It is not required to provide the template name.*

```
nxpimage bootable-image get-templates -f rw61x -o WORKSPACE\bootable-image
```

The output of the command is a template file for the RW61x main binary image.

Step 2 – Update *bootimg_rw61x_flexspi_nor.yaml* configuration file.

Example:

```
# ================================  Bootable Image Configuration template for rw61x.  ==========================

# ================================================================================================================
#                                                == General Options ==

# ================================================================================================================
# ------------------------------------------------====== MCU family [Required] =====------------------------------
# Description: MCU family name.
# Possible options: <lpc55s3x, rt101x, rt102x, rt104x, rt105x, rt106x, rt116x, rt117x, rt118x, rt5xx, rt6xx, rw61x>
family: rw61x
# ----------------------------------------====== Chip silicon revision [Optional] ============================
# Description: If needed this could be used to specify silicon revision of device.
# Possible options: <latest>
revision: latest
# ------------------------------------------------====== Memory type [Required] ===================================
# Description: Specify type of memory used by bootable image description.
# Possible options: <internal, flexspi_nor>
memory_type: flexspi_nor
# ================================================================================================================
#                                           == Bootable Segments definition ==

# ================================================================================================================
# ----------------------------------------=====  FCB block path [Optional] ==================================
# Description: Flash Configuration block Image path. It could be used as pre-prepared binary form of FCB and also
 YAML
# configuration file for FCB. In case that YAML configuration file is used, the Bootable image tool build the FCB
# itself.
fcb: fcb.bin
# ------------------------------------------------====== Image version [Optional] ================================
# Description: Image version
image_version: 0
# ================================================================================================================
#                                          == Executable Segment definition ==

# ================================================================================================================
# ----------------------------------=====  Main Boot Image [Conditionally required] =====--------------------
# Description: Main Boot Image path. It could be used as pre-prepared binary form of MBI and also YAML
 configuration
# file for MBI. In case that YAML configuration file is used, the Bootable image tool build the MBI itself.
mbi: app_signed.bin
-
```

Step 3 – Generate the main boot image.

```
nxpimage bootable-image merge -c bootimg_rw61x_flexspi_nor.yaml -o masterboot_signed.bin
```

Note: *You can use a single command to generate a main boot signed image with the .yaml files (bootimg_rw61x_flexspi_nor.yaml) instead of using fcb.bin and app_signed.bin.*

To enable the secure boot of *masterboot_signed.bin* image on RW61x, the SECURE_BOOT_EN and ROTKH fuses must be programmed (use shadowregs feature for testing).

Refer to Section 7 for guidance on programming an image.

3.7 Prepare SB3.1 image

This section shows how to use the SPSDK *nxpimage* utility to generate the Secure Binary v3.1 image.

Step 1 - Generate the template.

```
nxpimage sb31 get-template -f rw61x -o sb3_app_template.yaml
```

The output of the command is a template file for RW61x Secure Binary v3.1 image.

Step 2 - Update *sb3_app_template.yaml* configuration file. One example is shown below:

```
# ==============================  Secure Binary v3.1 Configuration template for rw61x.  =======================

# ============================================================================================================
#                                              == Basic Settings ==

# ============================================================================================================
# ----------------------------------------===== Firmware version. [Optional] =====-------------------------------
# Description: Value compared with Secure_FW_Version monotonic counter value stored in PFR/IFR. If value is lower
 than
# value in PFR/IFR, then is image rejected (rollback protection)..
firmwareVersion: 0
# ------------------------------------------------===== MCU family [Required] =====-------------------------------------
# Description: MCU family name.
# Possible options: <k32w1xx, kw45xx, lpc55s3x, mcxn9xx, rw61x>
family: rw61x
# ------------------------------------------------ SB3 filename [Required] ----------------------------------------
# Description: Generated SB3 container filename.
containerOutputFile: app_encrypted.sb
# ============================================================================================================
#                                          == Image Signing Settings ==

# ============================================================================================================
# ------------------------------ Main Certificate private key [Conditionally required] -------------------------
# Description: Main Certificate private key used to sign certificate. It can be replaced by signProvider key.
signPrivateKey: ../keys/ec_pk_secp256r1_cert0.pem
# ----------------------------------===== Signature Provider [Conditionally required] ----------------------------
# Description: Signature provider configuration in format 'type=<sp_type>;<key1>=<value1>;<key2>=<value2>".
#signProvider: type=file;file_path=../keys/ec_pk_secp256r1_cert0.pem
# ============================================================================================================
#                                          == Certificate Block V2.1 ==

# ============================================================================================================
# ------------------------------=====  Certificate Block binary/config file [Required] =====-----------------------
# Description: Path to certificate block binary or config file.
certBlock: cert_block.yaml
# ============================================================================================================
#                                       == Secure Binary v3.1 Settings ==

#
 ============================================================================================================
# ----------------------------------------===== Part Common Key [Optional]
 =====-------------------------------------
# Description: Path to PCK/NPK 256 or 128 bit key in plain hex string format or path to binary file or hex string.
containerKeyBlobEncryptionKey: cust_mk_sk.txt
# ----------------------------------=====  Enable NXP Container format [Optional] =====---------------------------
# Description: Internal usage only, used for generating SB files with NXP content e.g. provisioning firmware,
 etc...
isNxpContainer: false
# ----------------------------------------===== KDK access rights [Optional] =====---------------------------------
# Description: Accepted values are 0, 1, 2 and 3. Value used as key properties for key derivation process, more
 details
# can be found in CSSv2 manual.
# Possible options: <0, 1, 2, 3>
kdkAccessRights: 0
# ----------------------------------===== Container configuration word [Optional] =====---------------------------
# Description: Flag value in SB3.1 manifest, not used by silicons with LPC55S3x ROM. Value can be kept 0, or it can
 be
# removed from the configuration file.
containerConfigurationWord: 0
# ------------------------------------------------===== Description [Optional] ========================================
# Description: Description up to 16 characters, longer will be truncated. Stored in SB3.1 manifest.
description: This is description of generated SB file.
# ============================================================================================================
#                                     == Secure Binary v3.1 Commands Settings ==

# ============================================================================================================
# ----------------------------------------===== SB3.1 Commands [Required] =====-------------------------------------
# Description: Secure Binary v3.1 commands block, list of all possible options - Modify it according to your
 application
```

```
commands:
    - load:  # [Required], Load; If set, then the data to write immediately follows the range header. The length
 field contains the actual data length
        address: '0x20001000'  # [Required], Address of memory block to be loaded.
        values: '0xC0000008' # [Optional], Binary values delimited by comma to be loaded.
    - configureMemory:  # [Required], Configure memory.
        configAddress: '0x20001000'  # [Required], Address; Configuration address.
        memoryId: '0x09' # [Optional], Memory ID; ID of memory block to be configured.
    - erase:  # [Required], Erase; Performs a flash erase of the given address range. The erase will be rounded up
 to the sector size.
        address: '0x08000000'  # [Required], Address of memory block to be erased.
        size: '0x10000' # [Required], Size of memory block to be erased.
      memoryId: '0x09' # [Optional], Memory ID; ID of memory block to be erased.
    - load:  # [Required], Load; If set, then the data to write immediately follows the range header. The length
 field contains the actual data length
        address: '0x20001000'  # [Required], Address of memory block to be loaded.
        values: 0xB0000000,0x08000400 # [Optional], Binary values delimited by comma to be loaded.
    - configureMemory:  # [Required], Configure memory.
        configAddress: '0x20001000'  # [Required], Address; Configuration address.
        memoryId: '0x09' # [Optional], Memory ID; ID of memory block to be configured.
    - load:  # [Required], Load; If set, then the data to write immediately follows the range header. The length
 field contains the actual data length
        address: '0x08000600'  # [Required], Address of memory block to be loaded.
        values: '0xFFFE0001' # [Optional], Binary values delimited by comma to be loaded.
    - load:  # [Required], Load; If set, then the data to write immediately follows the range header. The length
 field contains the actual data length
        address: '0x08001000'  # [Required], Address of memory block to be loaded.
        file: app_signed.bin

            -
```

Step 3 - Generate the Secure Binary v3.1 image.

```
nxpimage sb31 export -c sb3_app.yaml
    - which generates image user_app.sb
```

Step 4 - Execute/load the Secure Binary v3.1 image.

The eecution of the Secure Binary v3.1 image can be done via ISP boot mode. For details about ISP boot refer to Section 4.

In this document ISP boot mode via UART is taken as an example.

```
blhost.exe -p COM22,115200 -t 60000 -- receive-sb-file user_app.sb
```

4 Program ISP mode

4.1 Set the device to ISP mode

To program the OTP, use the SPSDK blhost.exe tool. The blhost.exe tool can communicate with the device booted into ISP mode. A new device boots to ISP mode automatically. If not, force ISP mode during a reset by setting two ISP pins (Table 5).

Table 5. Boot mode and ISP download modes based on ISP pins

Boot mode	ISP3	ISP2	ISP1	ISP0	Description
ISP boot	HIGH	HIGH	HIGH	LOW	One of the serial interfaces (UART, I^2C, SPI, USB-HID) is used to configure the device, program the OTP or external. The first valid probe message on UART, I^2C, SPI, or USB locks in that interface.

Note: *For the available boot modes, refer to [2].*

The ISP mode can be specified using the DEFAULT_ISP_MODE bits (Table 6). For more details, see See [2].

Table 6. ISP download mode based on DEFAULT_ISP_MODE bits (6:4, fuseword 15 in OTP)

ISP Boot mode	ISP_MODE_2	ISP_MODE_1	ISP_MODE_0	Description
Auto ISP	0	0	0	RW61x probes the active peripheral from one of the serial interfaces and downloads the image from the probed peripherals: UART, I^2C, SPI, USB-HID
USB HID ISP	0	0	1	USB HID class used to download the image of the USB0 port.
UART ISP	0	1	0	The UART is used to download the image.
SPI target ISP	0	1	1	The SPI target is used to download the image.
I^2C target ISP	1	0	0	The I^2C target is used to download the image.
Disable ISP	1	1	1	Disable ISP mode.

Note: *The following sections describe the ISP mode using the USB0 HID mode to communicate with the device. Using the other modes is similar to the USB mode.*

When the device boots into the USB HID ISP mode, a new USB composite device is listed with its VID and PID values. Figure 6 shows an example of device enumeration in a Windows Control Panel.

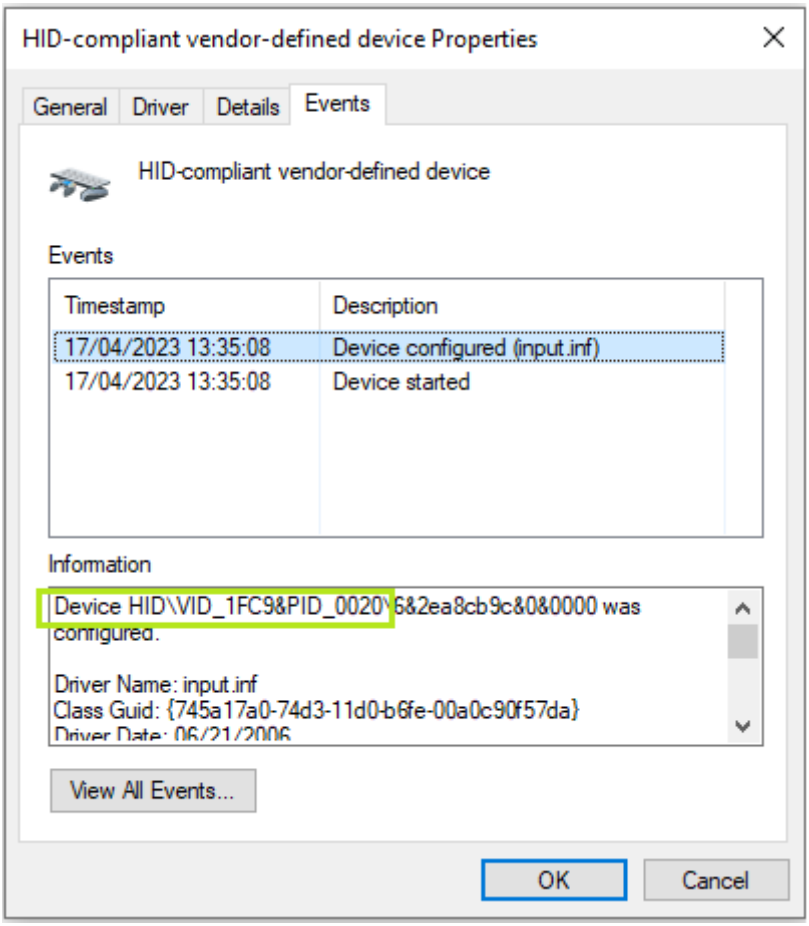

Figure 6. RW61x USB enumeration in ISP mode

4.1.1 Get the connected NXP device in ISP mode

To get the connected device set to ISP mode, use `nxpdevscan` SPSDK command (Figure 7). The command finds the devices connected through the USB or UART interfaces. The command returns the `VID` and `PID` values for the USB device, and the `path`. For the UART interface, the command returns the corresponding COM port number.

Use the path to address the target device when more devices with identical VID and PID are connected to the same PC.

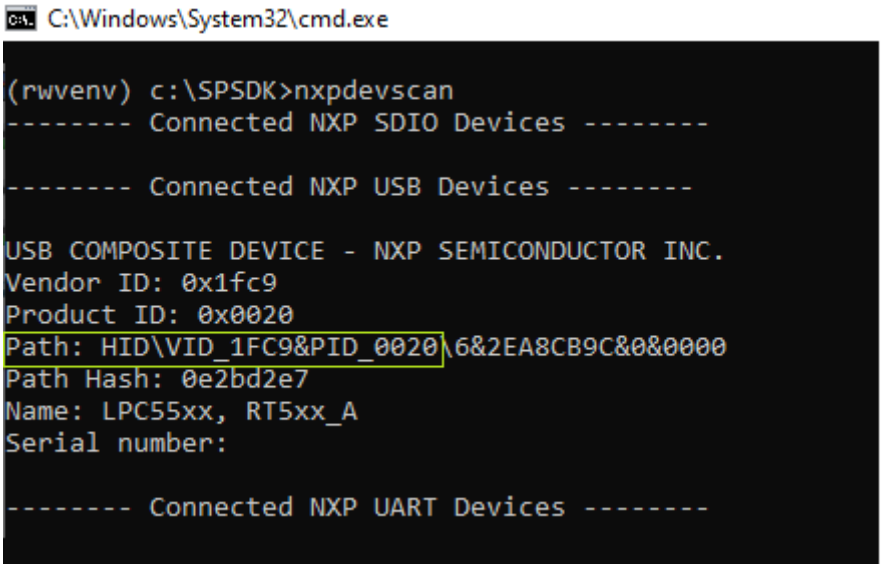

Figure 7. "nxpdevscan" command output

4.2 Use blhost.exe in ISP mode to communicate over USB

This section provide the SPSDK blhost.exe tool commands to communicate over USB when the device is in ISP mode.

Command to test the connection:

```
blhost.exe -u "0x1fc9,0x0020" -t 5000 get-property 1
```

Command to determine the security state of the device:

```
blhost.exe -u "0x1fc9,0x0020" -t 5000 get-property 0x11
```

The device returns the following:

- For the Unsecure state: 0x5aa55aa5
- For the Secure state: 0xc33cc33c

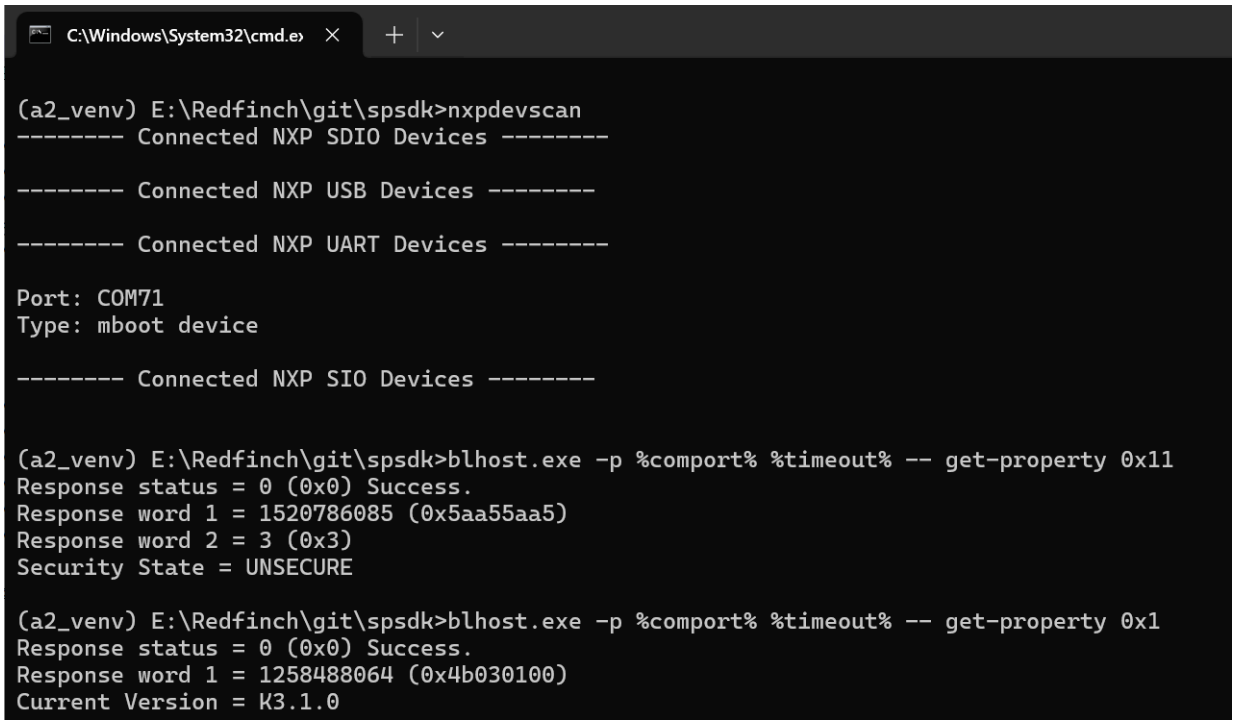

Figure 8. Testing "blhost.exe" connection in ISP USB HID mode

5 External memory support

This section shows the external memory devices that the boot ROM ISP command supports.

To use an external memory device, the device must be enabled with the corresponding configuration profile. If the external memory device is not enabled in the configuration profile, the ROM ISP command cannot access it. The boot ROM enables specific external memory devices using a pre-assigned memory identifier.

Table 7. Memory ID for external memory devices

Memory identifier	External memory device
0x09	'Serial NOR over FlexSPI module'

5.1 Boot ROM Flash driver configuration

The boot ROM is configured for JEDEC216 compliant flash devices by evaluating an SFDP read from the external flash memory. The boot ROM evaluates the SFDP data and internally generates an FCB from the retrieved information. Options to configure this process are described below.

Table 8. Option0 definition

Field Name	Offset	Width (bit)	Description
tag	28	4	The tag of the config option. Must be 0xC to identify a value as Option0.
option_size	24	4	Indicates whether a second option value is present. If the value of the field is 0, only Option0 is expected, otherwise Option1 is also expected.
device_type	20	4	Device detection type
query_pad	16	4	Number of I/O pads to use for command for reading Flash device information (read SFDP or read manufacturer id).
cmd_pad	12	4	Number of I/O pads to use for command for reading Flash. Applies to the command byte sent to Flash, not address or data bytes. For Flashes that use 1-1-4 or 1-4-4 mode, this field shall indicate to use one I/O pad.
quad_mode_setting	8	4	Certain Flash models require that their quad mode is explicitly enabled by writing a Quad-Enable (QE) bit to a status register. Information such as which bit is in which status register is part of the JEDEC Basic Flash Parameter Table in double word 15 of JESD216A. However, for Flashes that support only earlier versions (JESD216), this field provides an alternative way to provide that information. Note, the field only applies to devices that support JESD216.
misc_mode	4	4	Miscellaneous mode
max_freq	0	4	Max Flash Operation speed

Table 9. Option1 definition

Field	Offset	Width (bit)	Description
reserved	8	24	Reserved for future use
dummy_cycles	0	8	Dummy cycles for read command

Table 10. Possible values for Option0.device_type

Value	Description
b'0000	Use SDR instructions for reading device info. Generate SDR instructions in LUT.
b'0001	Use DDR instructions for reading device info. Generate DDR instructions in LUT.
any other value	Reserved

Table 11. Possible values for Option0.query_type and Option0.cmd_type

Value	Description
b'0000	Use one data line.
b'0010	Use four data lines (quad).
any other value	Reserved

Table 12. Possible values for Option0.quad_mode_setting

Value	Description
b'0000	Do not set a QE bit.
b'0001	Set bit 6 in status register 1.
b'0010	Set bit 1 in status register 2.
b'0011	Set bit 7 in status register 2.
b'0100	Set bit 1 in status register 2 using command 0x31.
any other value	Reserved

Table 13. Possible values for Option0.misc_mode

Value	Description
b'x000	Do not use a special mode.
b'x001	Use 0-4-4 mode for flash reads if supported by Flash.
b'0101	Force usage of dummy read strobe generated by FlexSPI controller and looped back internally for sampling read data.
b'0110	Force usage of SPI mode (even if SFDP indicates that Flash supports quad mode).
b'1xxx	Use external read strobe supplied via dqs pad for sampling read data.

Table 14. Possible values for Option0.max_freq

Value	Description
b'0000	Do not change the clock frequency.
b'0001	Use a clock frequency of 30 MHz.
b'0010	Use a clock frequency of 50 MHz.
b'0011	Use a clock frequency of 60 MHz.
b'0100	Use a clock frequency of 80 MHz.
b'0101	Use a clock frequency of 100 MHz.
b'0110	Use a clock frequency of 120 MHz.
b'0111	Use a clock frequency of 133 MHz.
b'1000	Use a clock frequency of 166 MHz.
b'1001	Use a clock frequency of 200 MHz.
any other value	Reserved

Table 15. Possible values for Option1.dummy_cycles

Value	Description
b'0000	Use the number of dummy cycles as detected.
any other value X	Use X dummy cycles.

JESD216A/B only defines the dummy cycles for Quad SDR read. To get the dummy cycles for DDR/DTR read mode, boot ROM also supports auto probing by writing test patterns to offset 0x200 on the external Flash devices.

To get optimal timing, readSampleClkSrc in the generated FCB is set to 1 for Flash devices that do not support externally provided DQS pad input. The readSampleClkSrct is set to 3 for Flash devices that support external provided DQS pad input. FlexSPI_DQS pad is not used for other purposes.

Table 16. Typical Flash config Option values

Value	Description
0xc0000002	SRD Flash, using 50 MHz clock frequency.
0xc0100002	DDR Flash, using 50 MHz clock frequency.

5.2 Flash driver example

The Flash model used for the example is a MX25U51245G. MX25U51245G supports JEDEC216 and operation at 133 MHz. MX25U51245G also supports DDR mode. Therefore, it is possible to have the boot ROM generate an FCB using an Option0 value of 0xC0100007. The listing below shows the invocation and output of the `blhost.exe` command to write the Option0 value to RAM, and the consecutive configuration of the boot ROM Flash driver from the Option0 value.

```
$ blhost.exe -p COM22,115200 -t 60000 fill-memory 0x2000F000 4 0xC0100007
Response status = 0 (0x0) Success.
$ blhost.exe -p COM22,115200 -t 60000 configure-memory 0x09 0x2000F000
Response status = 0 (0x0) Success.
```

After the Flash driver is configured, you can print information about the configuration. The following listing shows the configured Flash properties.

```
$ blhost.exe -p COM22,115200 -t 60000 -- get-property 0x19 0x9
Response status = 0 (0x0) Success.
Response word 1 = 31 (0x1f)
Response word 2 = 134217728 (0x8000000)
Response word 3 = 65536 (0x10000)
Response word 4 = 256 (0x100)
Response word 5 = 4096 (0x1000)
Response word 6 = 65536 (0x10000)
External Memory Attributes = Start Address: 0x08000000, Total Size:    64.0 MiB, Page Size:     256 B, Sector Size:
   4.0 kiB, Block Size:    64.0 kiB
```

6 Configure OTP fuses

6.1 OTP fields during manufacturing

The OTP fusemap contains the settings for a signed image in a secure boot configuration, and the 48 bytes of ROTKH. The signed image settings are contained in fusewords from 15 to 19 with ECC checksum. The image settings can only be written once, as a single fuse word update. This section only describes the fields related to secure boot in the tables below.

Table 17. BOOT_CFG0 fuse word security bit field definitions - Fuseword: 15

Bits	Name	Description
13:12	TZM_IMAGE_TYPE	TrustZone-M mode 2'b00: Ignored 2'b01: Enforce preset TZM data in the image manifest. 2'b10: Enforce preset TZM data in the image manifest. 2'b11: Enforce preset TZM data in the image manifest.
21:20	SECURE_BOOT_EN	Secure boot enable 2'b00: Plain image (with or without cyclic redundancy check (CRC)) 2'b01: RFU. 2'b1x: Complete ECDSA check of the signed images (ECDSA signed).
22:22	DICE_INC_OTP	Include OTP fuses area in dice computation 1'b0: not included 1'b1: included
23:23	DICE_SKIP	Skip dice computation 1'b0: Enable dice 1'b1: Disable dice

Table 18. BOOT_CFG3 fuse word security bit field definitions - Fuseword: 18

Bits	Name	Description
2:0	RoTK0_Usage	RoT key 0 usage properties. 3`b000 - Usable as debug cryptic acceleration (CA), image CA, FW CA, image, and FW key. 3`b001 - Usable as debug CA only. 3`b010 - Usable as image (boot and FW) CA only. 3`b011 - Usable as debug, boot, and FW image CA. 3`b100 - Usable as image key and FW update key only. 3`b101 - Usable as boot image key only. 3`b110 - Usable as FW update image key only. 3`b111 - Key slot is not used.

Table 18. BOOT_CFG3 fuse word security bit field definitions - Fuseword: 18*...continued*

Bits	Name	Description
5:3	RoTK1_Usage	RoT key 1 usage properties. 3`b000 - Usable as debug CA, image CA, FW CA, image, and FW key. 3`b001 - Usable as debug CA only. 3`b010 - Usable as image (boot and FW) CA only. 3`b011 - Usable as debug, boot, and FW image CA. 3`b100 - Usable as image key and FW update key only. 3`b101 - Usable as boot image key only. 3`b110 - Usable as FW update image key only. 3`b111 - Key slot is not used.
8:6	RoTK2_Usage	RoT key 2 usage properties. 3`b000 - Usable as debug CA, image CA, FW CA, image, and FW key. 3`b001 - Usable as debug CA only. 3`b010 - Usable as image (boot and FW) CA only. 3`b011 - Usable as debug, boot, and FW image CA. 3`b100 - Usable as image key and FW update key only. 3`b101 - Usable as boot image key only. 3`b110 - Usable as FW update image key only. 3`b111 - Key slot is not used.
11:9	RoTK3_Usage	RoT key 3 usage properties. 3`b000 - Usable as debug CA, image CA, FW CA, image, and FW key. 3`b001 - Usable as debug CA only. 3`b010 - Usable as image (boot and FW) CA only. 3`b011 - Usable as debug, boot, and FW image CA. 3`b100 - Usable as image key and FW update key only. 3`b101 - Usable as boot image key only. 3`b110 - Usable as FW update image key only. 3`b111 - Key slot is not used.
13:12	ENF_CNSA	Enforce the CNSA suite algorithm. 2'b00: ECC P-256 keys 2'b01: ECC P-384 keys, SHA384 & AES256 2'b10: ECC P-384 keys, SHA384 & AES256 2'b11: ECC P-384 keys, SHA384 & AES256
15:14	ENALBE_CRC_CHECK	Enable CRC checks over OTP area
22	FIPS_KDF_STEN	Enable self-test for CKDF block on power up. Required for FIPS certification. If this field is nonzero, run a self-test and log the result in BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot
23	FIPS_CMAC_STEN	Enable self-test for CMAC block on power up. Required for FIPS certification. If this field is nonzero, run a self-test and log the result in BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot

Table 18. BOOT_CFG3 fuse word security bit field definitions - Fuseword: 18*...continued*

Bits	Name	Description
24	FIPS_DRBG_STEN	Enable self-test for DRBG block on power up. Required for FIPS certification. If this field is nonzero, run the self-test and log in the result. BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot
25	FIPS_ECDSA_STEN	Enable self-test for ECDSA block on power up. Required for FIPS certification. If this field is nonzero, run the self-test and log in the result. BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot
26	FIPS_AES_STEN	Enable self-test for AES block on power up. Required for FIPS certification. If this field is nonzero, run the self-test and log in the result. BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot
27	FIPS_SHA_STEN	Enable self-test for SHA2 block on power up. Required for FIPS certification. If this field is nonzero, run the self-test and log in the result. BOOT_STATE register. 2'b00: not included 2'b01: On failure, continue to boot
29:28	SKIP_PM_SIGN_VERIFCATION	On boot-up from PM3/PM4, do not run ECDSA signature verification of the image

The OTP fusemap also contains ROTKH revocation and firmware image version settings as monotonic counters. The fuses can be updated incrementally as they are implemented as redundant 16-bit fusewords.

Table 19. Field programmable OTP fusewords

Address	Bytes	Name	Description
368 - 383	32	TZ_NSW_Version	Nonsecure firmware version (Monotonic counter)
384 - 387	8	TZ_SW_Version	Secure firmware version (Monotonic counter)
from 24 to 25	4	IMAGE_KEY _REVOKE	Image key revocation ID
22	2	SEC_BOOT_CFG0	Root key revocation
23	2	SEC_BOOT_CFG1	DAP Vendor Usage

TZ_NSW_Version: Optionally used during SB3.1 file loading. The value written in the configuration word must always be lower or equal to the nonsecure FW version specified in the *elftosb.bd* file used to create the SB3.1 file. Otherwise, if this version check command is included in the SB3.1 file, the file load is rejected.

TZ_SW_Version: Optionally used during SB3.1 file loading. The value written in the configuration word must always be lower or equal to the secure FW version specified in the elftosb.bd file used to create the SB3.1 file. Otherwise, if this version check command is included in the SB3.1 file, the file load is rejected.

IMAGE_KEY _REVOKE: This value is checked during the image authentication process.
IMAGE_KEY_REVOKE field in OTP is checked against the “constraints” field of the Image Signing Key (ISK)

certificate inside the image certificate block. To boot the image, the constraints field of the image certificate must be always higher than the OTP field.

SEC_BOOT_CFG0: Each of four RoT Keys can be revoked. If SEC_BOOT_EN is set to boot signed images only, images that are signed using a revoked RoT key are rejected during the authentication process and fail.

SEC_BOOT_CFG1: Contains information used by debug authentication. For more details on debug authentication, refer to the *Debug subsystem* section in [2].

Table 20. ROTKH table bit field definition - Fuseword: 22

Bits	Name	Description
0	REVOKE_ROOTKEY0	RoT Key 0 enable 1'b0: RoT Key 0 is enabled 1b1: RoT Key 0 is revoked
1	REVOKE_ROOTKEY1	RoT Key 1 enable 1'b0: RoT Key 0 is enabled 1b1: RoT Key 0 is revoked
2	REVOKE_ROOTKEY2	RoT Key 2 enable 1'b0: RoT Key 0 is enabled 1b1: RoT Key 0 is revoked
3	REVOKE_ROOTKEY3	RoT Key 3 enable 1'b0: RoT Key 0 is enabled 1b1: RoT Key 0 is revoked
4	RESERVED	Reserved
5	NXP_PROV_FW_EXEC_DIS	Flag to disable execution of NXP signed provisioning Firmwares. Execution of NXP provisioning fw is not disabled `1b0` Execution of NXP provisioning fw is disabled `1b1`
15:6	RESERVED	Must be filled with zeros

6.2 Test RW61x OTP configuration

This section shows how to use the *SPSDK shadowreg* tool to test the OTP configurations before writing these configurations into the OTP fields. For details on `shadowregs` commands, refer to [7].

The following example describes how to test a secure boot configuration (Using ECC P-256 keys) before programming the fuses.

Prerequisites:

- The Device Life cycle is Develop (0x0303).
- ROTKH preparation is as described in Section 3.4.2.
- The preparation of the main boot signed image is as described in Section 3.6.4.
- The new Life-Cycle state in the OTP shadow to be configured is either Develop2 (0x0707) or In-Field (0x0F0F).
- The application does not write to the RAM region 0x20126000 - 0x20126200.

Note:

- *The boot ROM uses the RAM memory region 0x20126000 - 0x20126200 (0x200 bytes) as a buffer for OTP shadows. The users must not overwrite this region when using the shadowregs utility.*
- *For the description of Develop2 and In-Field Life-Cycle states, refer to the section Life-cycle states in [2].*

Step 1 – Generate a RW61x template (optional).

```
shadowregs -i jlink -f rw61x get-template -o shadowreg_template_rw61x.yml
```

Step 2 – Save current RW61x device configurations.

```
shadowregs -i jlink -f rw61x saveconfig -o saveconfig_rw61x.yml
```

You can use *saveconfig_rw61x.yml* as an input to test required OTP fuses.

Step 3 – Update *saveconfig_rw61x.yml* configuration for the following fuses:

1. ROTKH
2. SECURE_BOOT_EN bit in BOOT_CFG0
3. PRIMARY_BOOT_SOURCE bits in BOOT_CFG0
4. LifeCycle

The rest of the fuse settings can either be removed or kept as default.

Example of minimalistic configuration (*updated_shadow_rw61x.yml*):

```
description:
  device: rw61x
  author: Documenration
registers:
  BOOT_CFG0:
    value: '0x180001'
  BOOT_CFG3:
    value: '0x0'
  LIFE_CYCLE_STATE:
    value: '0xF0F'
  RKTH0:
    value: '0x3C9CEDB9'
  RKTH1:
    value: '0x759A35B1'
  RKTH2:
    value: '0xD6A03BA6'
  RKTH3:
    value: '0xCA335EAB'
  RKTH4:
    value: '0x71590A16'
  RKTH5:
    value: '0x6415F523'
  RKTH6:
    value: '0x93E018D7'
  RKTH7:
    value: '0xA941F70'
  RKTH8:
    value: '0x0'
  RKTH9:
    value: '0x0'
  RKTH10:
    value: '0x0'
  RKTH11:
    value: '0x0'
```

Note: *RKTH can be either provided as 256-bit string (384-bit string) format or as individual fuse values in little-endian format. For example:*

```
RKTH: # ROTKH field is compounded by 12 32-bit fields and contains Root key table hash.
 For ECC P-256 keys RKTH is a 32-bit SHA-256 digest of four SHA-256 digests computed
 over four OEM public keys (OEM has four private-public key pairs in case one of its
 private keys becomes compromised) or in case that ECC P-384 keys are used, RKTH is 48-
bit SHA-384 digest.
  value: 'b9ed9c3cb1359a75a63ba0d6ab5e33ca160a597123f51564d718e093701f940a'  # The value
 width: 384b
```

Step 4 - Load updated shadow RW61x configurations.

```
shadowregs -i jlink -f rw61x loadconfig -c updated_shadow_rw61x.yml
```

Step 5 - Reset RW61x.

To apply the shadow changes, issue the reset command:

```
shadowregs -i jlink -f rw61x reset
```

After the reset command and if the ROTKH values are correct and match the values of the main boot-signed image, the application boots from flash. If not, carefully review the values. Repeat the above steps as many times as necessary to guarantee that all the configurations are correct.

Use a similar technique to verify the debug authentication configuration settings. After step 5, the device boots as it does in LifeCycle In-Field state where debug ports are disabled.

Step 6 - Issue the debug authentication command (refer to [1]).

```
nxpdebugmbox -i jlink -v -p 2.0 auth -b 0 -c DAT_certificate.dc --key DCK_secp256r1.pem
```

If the settings are correct, debug ports are enabled. After successful debug authentication, verify the changed device configuration in shadow either by saving or by printing the shadow registers.

Step 7 - Print the new shadow registers.

```
shadowregs -i jlink -f rw61x printregs
  #   Interface   Id          Description
-----------------------------------------------------------------------
  0   Jlink       851006710   Segger J-Link Compact PLUS: 851006710
Register Name:        BOOT_CFG0
Register value:       0x00180001
Register raw value:   0x00180001

Register Name:        BOOT_CFG1
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        BOOT_CFG2
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        BOOT_CFG3
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        BOOT_CFG5
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        BOOT_CFG6
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        SEC_BOOT_CFG0
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        SEC_BOOT_CFG1
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        SEC_BOOT_CFG2
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        SEC_BOOT_CFG3
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        DCFG_CC_SOCU_NS
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        DCFG_CC_SOCU
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        DCFG_CC_SOCU_AP
Register value:       0x00000000
Register raw value:   0x00000000

Register Name:        LIFE_CYCLE_STATE
Register value:       0x00000F0F
Register raw value:   0x00000F0F

Register Name:        RKTH
Register value:
 B9ED9C3CB1359A75A63BA0D6AB5E33CA160A597123F51564D718E093701F940A0000000000000000000000000000000000
Register raw value:
 3C9CEDB9759A35B1D6A03BA6CA335EAB71590A166415F52393E018D70A941F7000000000000000000000000000000000000
```

When you are certain of the setup and OTP configuration values, use the shdowreg application to generate a batch script and program RW61x OTP fuses.

Step 8 - Generate the OTP fuse word writing script.

```
shadowregs -i jlink -f rw61x fuses-script -c updated_shadow_rw61x.yml -o
 blhost_rw61x_script.bat
```

6.3 Program RW61x for secure boot

The batch script *blhost_rw61x_script.bat* generated in **step 8** of Section 6.2, must be updated with the correct ISP boot host interface. For details about ISP boot refer to Section 4.

The following example demonstrates ISP boot communication via UART interface:

```
# blhost_rw61x_script.bat
# BLHOST fuses programming script
# Generated by SPSDK 2.1.0
# Chip: rw61x rev:A2

# Fuse BOOT_CFG0, index 15 and value: 0x00180001.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0xf 0x00180001
# Fuse LIFE_CYCLE_STATE, index 45 and value: 0x00000f0f.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x2d 0x00000f0f
# Fuse RKTH0, index 104 and value: 0x6b0ca180.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x68 0x6b0ca180
# Fuse RKTH1, index 105 and value: 0xe53df964.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x69 0xe53df964
# Fuse RKTH2, index 106 and value: 0xf76f895f.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6a 0xf76f895f
# Fuse RKTH3, index 107 and value: 0x02c356f9.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6b 0x02c356f9
# Fuse RKTH4, index 108 and value: 0xf5da7cee.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6c 0xf5da7cee
# Fuse RKTH5, index 109 and value: 0xc7093a41.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6d 0xc7093a41
# Fuse RKTH6, index 110 and value: 0xb582f8c2.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6e 0xb582f8c2
# Fuse RKTH7, index 111 and value: 0x3bea333a.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x6f 0x3bea333a
# Fuse RKTH8, index 112 and value: 0x00000000.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x70 0x00000000
# Fuse RKTH9, index 113 and value: 0x00000000.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x71 0x00000000
# Fuse RKTH10, index 114 and value: 0x00000000.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x72 0x00000000
# Fuse RKTH11, index 115 and value: 0x00000000.
blhost.exe -p COM22,115200 -t 60000 efuse-program-once 0x73 0x00000000
```

Command to execute the batch script:

```
call blhost_rw61x_script.bat
```

CAUTION: *OTP fuses cannot be changed once they have been programmed. Make sure to load a valid configuration in the device.*

6.4 User-defined OTP fusewords

The RW61x OTP fusemap includes the fusewords that the users can program for their custom use cases. The fusewords are both ECC and redundant OTP fuses (Table 21).

Table 21. Field programmable OTP fusewords for custom use cases.

Address	Type	Description
315 – 358	ECC	User-defined space as one-time fuses
400 – 403	Redundant	User-defined space as monotonic counters
408 – 419	Redundant	User-defined space as monotonic counters

7 Program the flash

7.1 Erase the external flash

Step 1 - Use `blhost.exe` command to configure the flash memory.

```
blhost.exe -p COM22,115200 -t 60000 fill-memory 0x2000F000 4 0xC0000004
Response status = 0 (0x0) Success.
blhost.exe -p COM22,115200 -t 60000 configure-memory 0x09 0x2000F000
Response status = 0 (0x0) Success.
```

Step 2 - Use `blhost.exe flash-erase-all` command to erase the external flash before the image is programmed.

```
blhost.exe -p COM22,115200 -t 60000 flash-erase-all 0x09
```

7.2 Program the FCB

To keep the current driver configuration for the boot ROM used for the next boot, use the special configuration value of `0xB0000000` and the address `0x08000400`. This writes the FCB at the address `0x08000400`. The code sample below shows the use of `blhost.exe` command to generate and read out the FCB.

```
$ blhost.exe -p COM22,115200 -t 60000 -- write-memory 0x20001000 "{{ 000000b000040008 }}"
Response status = 0 (0x0) Success.
Response word 1 = 8 (0x8)
$ blhost.exe -p COM22,115200 -t 60000 -- configure-memory 0x09  0x20001000
Response status = 0 (0x0) Success.
$ blhost.exe -p COM22,115200 -t 60000 read-memory 0x08000400 0x200
Reading memory  [##################################]  100%
46 43 46 42 00 04 01 56 00 00 00 00 01 03 03 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 01 04 01 00 00 00 00 00 00 00 00 00
00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ec 04 20 0a 00 1e 04 32 04 26 00 00 00 00 00 00
05 04 04 24 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21 04 20 08 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
dc 04 20 08 00 00 00 00 00 00 00 00 00 00 00 00
3e 04 20 0a 04 22 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 01 00 00 00 10 00 00 01 00 00 00 00 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Response status = 0 (0x0) Success.
Response word 1 = 512 (0x200)
Read 512 of 512 bytes.
```

7.3 Program the signed binary

This section shows how to program the signed image in two cases:

- Case 1: FCB is appended in the image
- Case 2: FCB is not appended in the image

If FCB is appended in the image you prepared using Section 3.6.3 and Section 3.6.4, write the binary image at `0x08000400` address.

If you manually appended the FCB, write the image binary at `0x08001000` address.

Case 1 - FCB is appended in the image

Step 1 - Refer to Section 3.6.3 and Section 3.6.4 to generate the image with FCB appended

Step 2 - Program the signed image. Write the binary image at `0x08000400` address.

```
blhost.exe -p COM22,115200 -t 60000 -- write-memory 0x08000400
 mbi_fcb_signed_hello_world.bin
```

Figure 9 shows the signed image executed from the external flash.

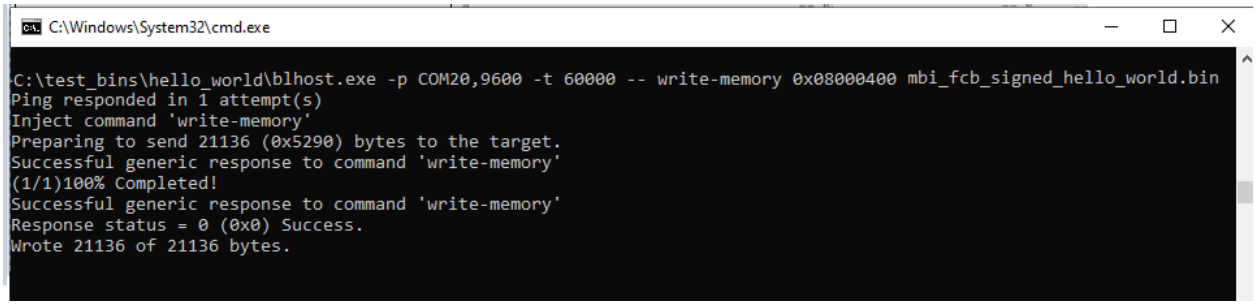

Figure 9. Example of `blhost.exe` command to program the flash memory - FCB is appended

Case 2 - FCB is not appended in the image

Step 1 - Program the signed image - Write the image binary at `0x08001000` address

Figure 10 shows the signed image executed from the external flash.

```
blhost.exe -p COM22,115200 -t 60000 -- write-memory 0x08001000 hello_world.bin
```

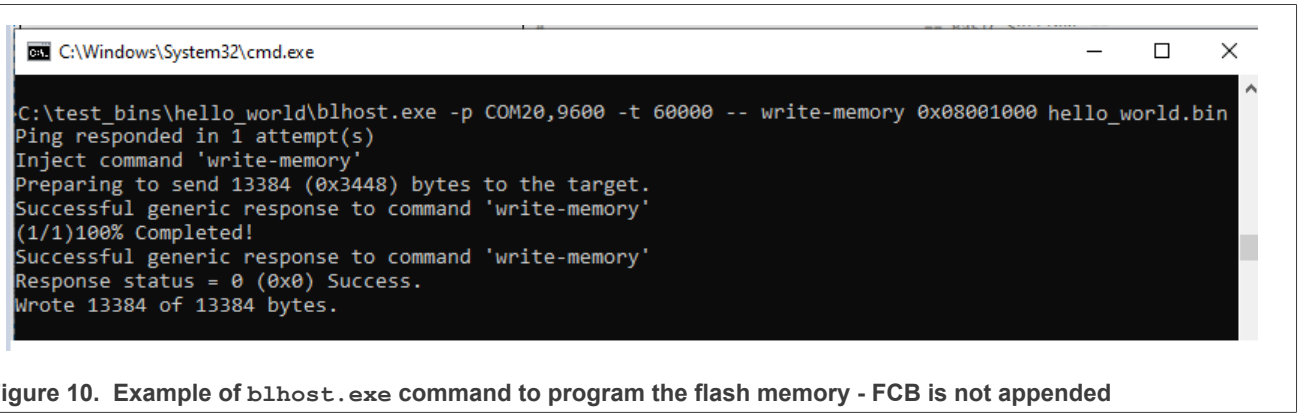

Figure 10. Example of `blhost.exe` command to program the flash memory - FCB is not appended

Next step - Switch ISP pins to `AUTO boot` or `FLEXSPI boot` (Section 4.1).

RW61x should reboot from the external flash if the valid image is found after the reset.

8 Abbreviations

Table 22. Abbreviations

Acronym	Description
FCB	Flash configuration block
ISK	Image signing key
MBI	Main boot image
NVIC	Nested vector interrupt controller
RoTK	Root of trust key
ROTKH	Root of trust key hash
SPSDK	Secure provisioning SDK
SPT	Signed plain text

9 References

[1] Application note – AN13814: RW61x Debug Authentication (link)
[2] User manual – UM11865: RW61x User Manual (link)
[3] Web page – MCUXpresso Secure Provisioning Tool (link)
[4] Software – Secure Provisioning SDK (SPSDK) (link)
[5] Web page – SPSDK documentation (link)
[6] Web page – User Guide - nxpimage (link)
[7] Web page – User Guide - shadowregs (link)

10 Revision history

Table 23. Revision history

Document ID	Release date	Description
AN13813 v.6.0	19 November 2024	• Changed the document access to public. No changes in the content.
AN13813 v.5.0	6 September 2024	• Section 2 "SPSDK tool": updated. • Section 3 "Prepare the secure boot image": updated. • Section 3.2 "Signed image structure": updated. • Section 3.5.1 "Generate the secure provisioning image": updated. • Section 3.5.2 "Execute the secure provisioning image": updated. • Section 3.7 "Prepare SB3.1 image": updated. • Section 4.1.1 "Get the connected NXP device in ISP mode": updated. • Section 5.1 "Boot ROM Flash driver configuration": updated. • Section 5.2 "Flash driver example": updated. • Section 6.1 "OTP fields during manufacturing": updated. • Section 6.3 "Program RW61x for secure boot": updated. • Section 7.2 "Program the FCB": updated.
AN13813 v.4.0	10 May 2024	• Section 1 "Introduction": updated. • Section 3.5 "SB3 processing keys": added. • Section 3.5.1 "Generate the secure provisioning image": updated. • Section 3.5.2: added. • Section 3.6.2 "Generate the signed image": updated. • Section 3.6.3 "Generate FCB image": updated. • Section 3.6.4 "Generate the main binary image": updated. • Section 4.2 "Use blhost.exe in ISP mode to communicate over USB": updated. • Section 6.2 "Test RW61x OTP configuration": updated. • Section 6.3 "Program RW61x for secure boot": updated.
AN13813 v.3.0	12 December 2023	• Section 1 "Introduction": updated. • Section 9 "References": updated. • Section 2 "SPSDK tool": updated. • Section 3.4.3 "Generate the keys": updated. • Section 3.3 "SB3.1 image structure": added. • Section 3.5.1 "Generate the secure provisioning image": added. • Section 3.6 "Prepare the main boot signed image": updated. • Section 3.6.1 "Use the SPSDK tool": updated. • Section 3.6.2 "Generate the signed image": updated. • Section 3.6.3 "Generate FCB image": updated. • Section 3.6.4 "Generate the main binary image": updated. • Section 3.7 "Prepare SB3.1 image": added. • Section 4.2 "Use blhost.exe in ISP mode to communicate over USB": updated. • Section 6.1 "OTP fields during manufacturing": updated. • Section 6.2 "Test RW61x OTP configuration": added. • Section 6.3 "Program RW61x for secure boot": updated.

Table 23. Revision history...*continued*

Document ID	Release date	Description
AN13813 v.2.0	10 October 2023	• Section 3.1 "Plain image structure": updated. • Section 3.6.4 "Generate the main binary image": updated. • Section 6.2 "Test RW61x OTP configuration": updated. • Section 11 "Note about the source code in the document": added
AN13813 v.1.0	12 May 2023	• Initial version

11 Note about the source code in the document

The 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 must be 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.

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.

Tables

Figures

Contents