AN13814

Debug Authentication on RW61x

Rev. 4.0 — 14 November 2024 Application note

Document information

Information	Content
Keywords	Debug authentication protocol, debugger, secure provisioning SDK (SPSDK), system state, system information, credentials
Abstract	Describes the steps for debug authentication using the secure provisioning SDK tool.

1 Introduction

The RW61x family of devices includes many possibilities to configure the debug port and a possibility to debug the firmware. The fundamental principles of debugging, which require access to the system state and system information, conflict with the principles of security, which require the restriction of access to assets. Therefore, many products disable the debug access completely before deploying the product. It causes challenges for product design teams to do a proper Return Material Analysis (RMA). To address these challenges, the RW61x offers the Debug Authentication Protocol (DAP) as a mechanism to authenticate the debugger (an external entity) which has the credentials approved by the product manufacturer before granting the debug access to the device. The OEM is the owner of the root key pairs. The root key hash is programmed into the device during manufacturing. When the end customer faces an issue, the device is shipped to a repair center. The field technician can send a request to the OEM to provide the Debug Credential certificate (DC), which is signed by a private root key. The field technician uses this DC to provide debug access. This application note describes an example of configuration and usage on RW61x.

Note: *Examples in this application note were written using SPSDK version 2.0.0 [2]. The MCUXpresso Secure Provisioning (SEC) Tool is the latest tool [3]. The information in this application note describing the secure flow within the device still applies, but we recommend using the latest tools (MCUXpresso SEC or SPSDK) instead of following the steps described in the document. Contact your NXP representative if you have any question.*

2 Debug authentication overview

The device supports Arm serial wire debug (SWD) and JTAG interface. SWD is the default function for pins GPIO[13] (SWCLK) and GPIO[14] (SWDIO) after a reset.

The ROM controls debug access via remote host. Debug access is only enabled when permitted through the device configuration, and when the correct protocol is followed to initiate a debug session. If the device has been configured for debug authentication, a debug session must be initiated following the correct authentication sequence.

Table 1 shows the protocol to use for a specific life-cycle state.

Table 1. Debug protocol and device life-cycle state

Protocol	Device life-cycle state
Debug session	Development
Debug authentication	Deployed

Figure 1 shows the top-level debug ports and the connections in the device.

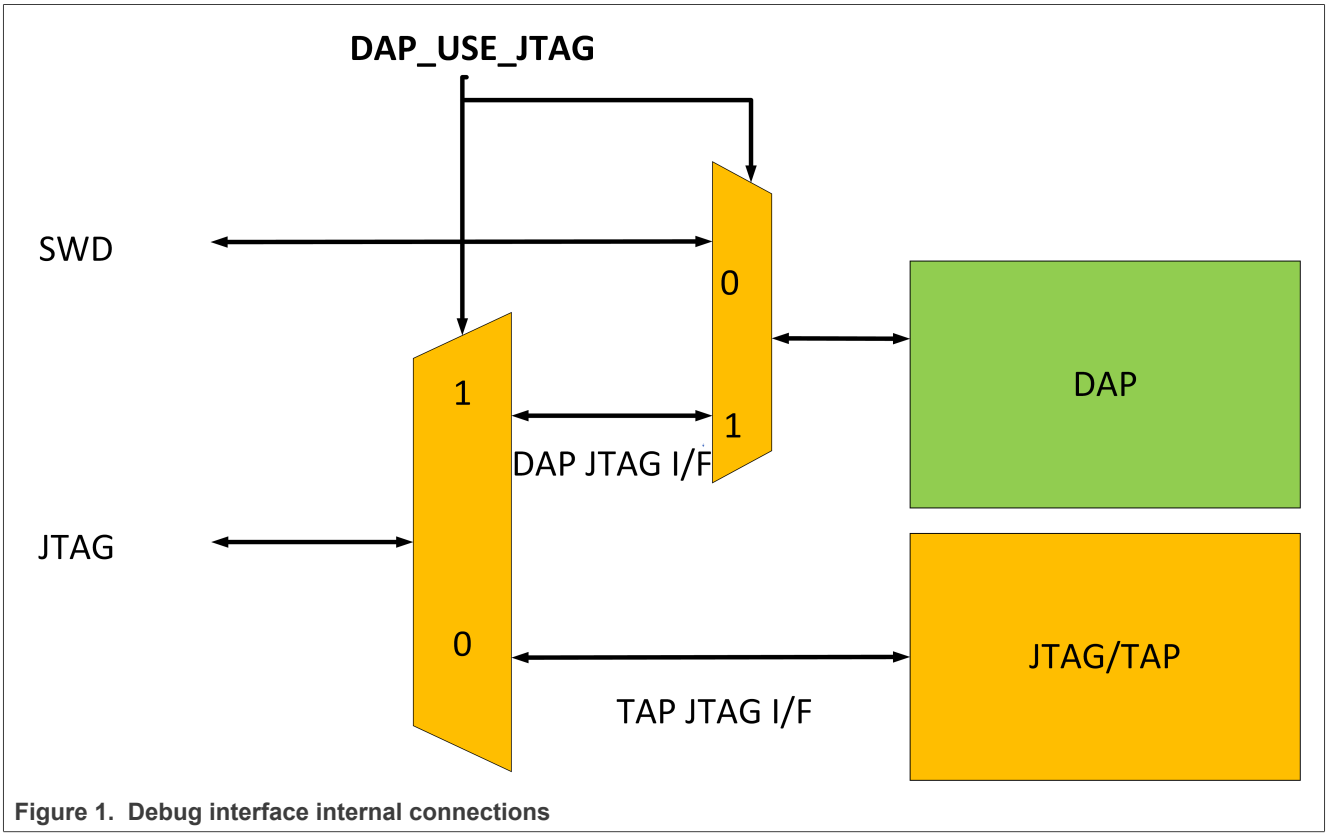

Figure 1. Debug interface internal connections

The five JTAG pins and the two SWD pins are separate. The GPIOs used for JTAG and SWD can be used for other modes than scan and debugging.

The two figures below show use case examples:

- Use case 1: Connect SWD to DAP and use JTAG for TAP.
- User case 2: Connect JTAG to DAP and TAP must be disabled as it cannot be controlled through JTAG.

Use case 1: TAP accessed by JTAG and DAP by SWD

– Select TAP on JTAG interface by setting DAP_USE_JTAG strap pin as 0
– JTAG only accesses TAP
– SWD can access DAP optionally, as the GPIO for SWD interface is independent.

Figure 2 illustrates the connectivity of use case 1

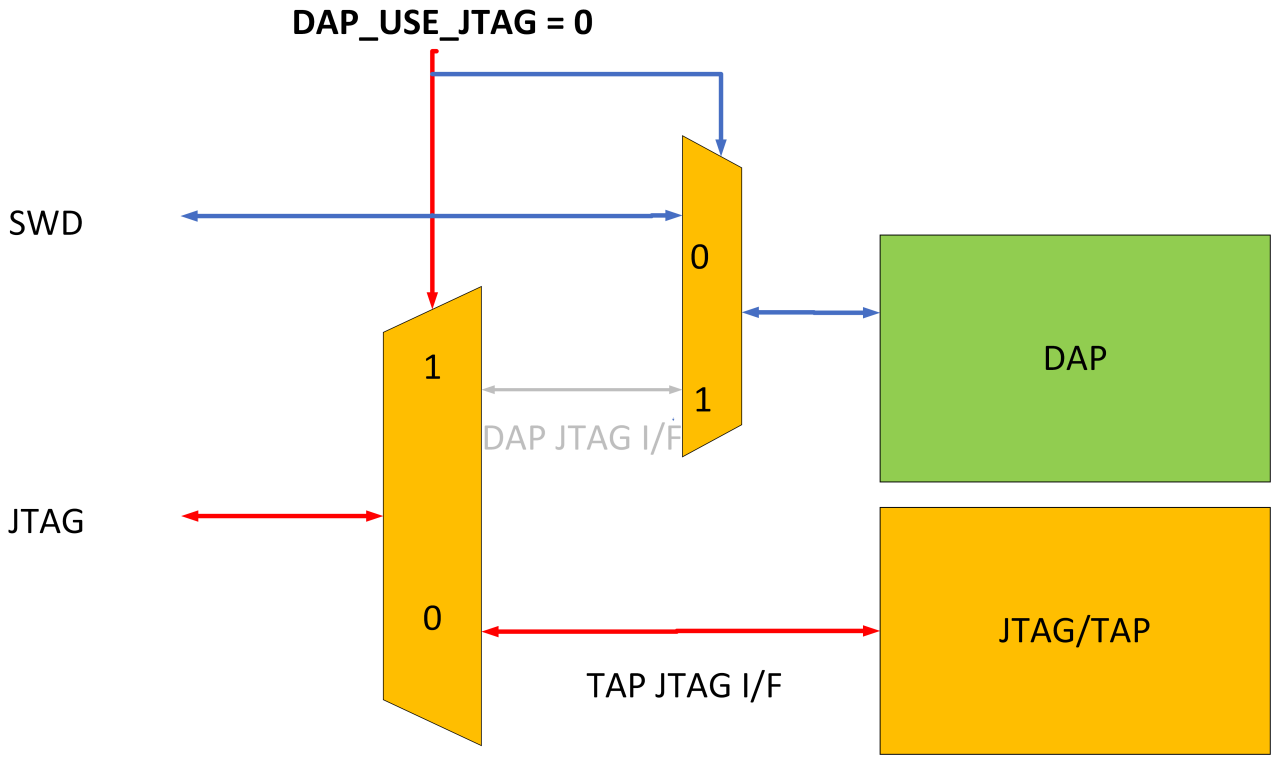

Figure 2. Use case-1: JTAG for TAP and SWD for DAP

Use case 2: DAP accessed by JTAG

- Select DAP on JTAG interface by setting DAP_USE_JTAG as 1 (default value of the strap pin).
- Only JTAG accesses DAP
- The required sequence on JTAG pins accesses DAP

Figure 3 illustrates the connectivity of use case 2.

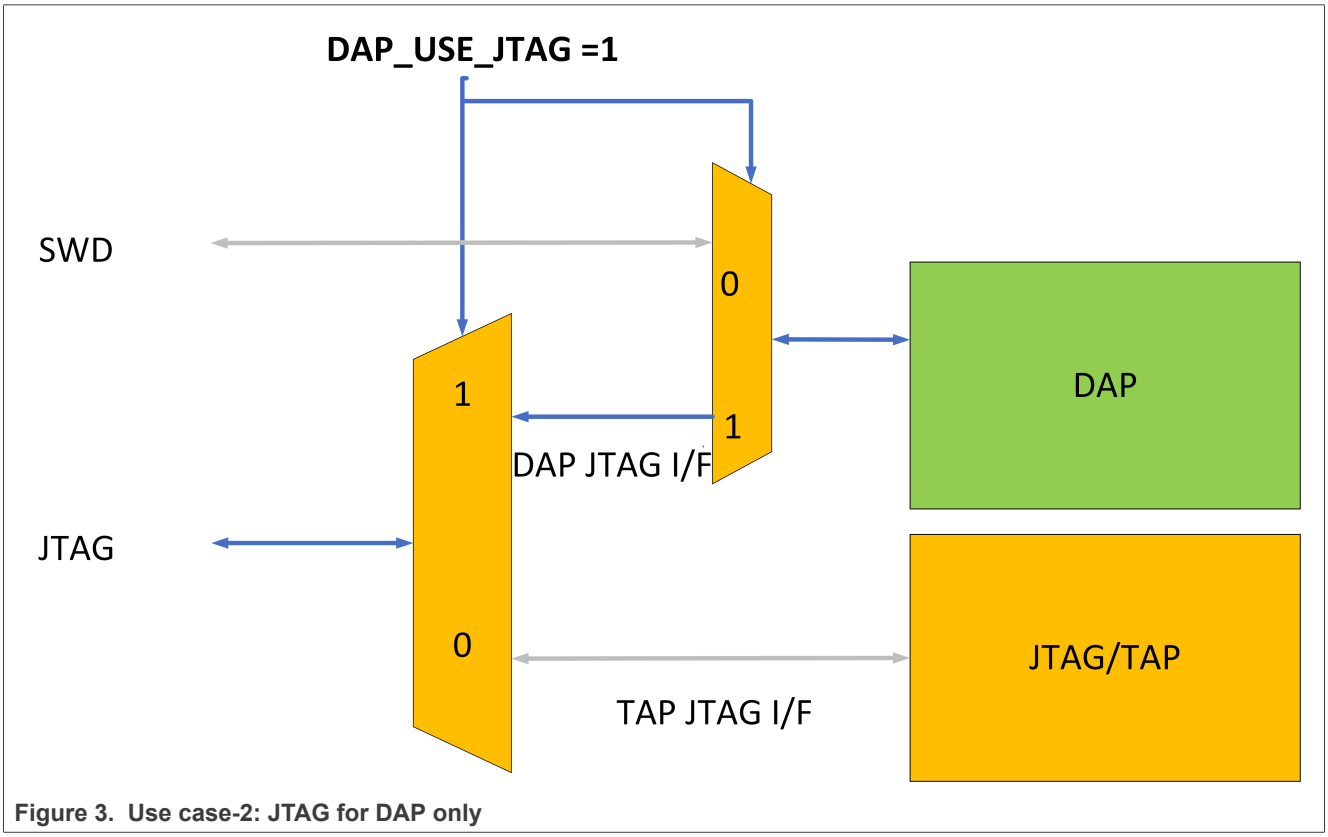

Figure 3. Use case-2: JTAG for DAP only

Note: *The debugger connected to JTAG pins cannot access DAP by SWD protocol.*

- DAP: Debug access port with Serial Wire port (SWJ-DP). Interprets incoming data and routes to appropriate Access Port (AP). The external I/O pins that interface with DAP are SWCLK and SWDIO. The DAP block is always enabled but the I/O pins that provide access to the SWD signals may be used for other functions controlled by software.
- DM-AP: Debug access port for debug mailbox. Debug Mailbox sends and receives messages from code executing from the ROM.
  - DM-AP is always enabled and the external world can send data to and receive data from the ROM.
  - DM-AP implements NXP debug authentication protocol versions 2.0 and 2.1.

2.1 Cortex M33 debug

The debug access port (DAP) for the Arm Cortex-M33 processor is disabled during power-on reset or during the assertion of the reset pin. The ROM enables the DAP when the correct debug initiation procedures are followed. If the DAP is not used, the debug enablement protocol can be used to initiate a debug session. If debug authentication is required, refer to Section 6 "Perform debug authentication". The debug authentication process allows control of the DBGEN, NIDEN, SPIDEN, and SPNIDEN signals generated by the Cortex-M33. The signals are described below.

- **DBGEN:** Invasive debugging of TrustZone for Armv8-M architecture defined non-secure domain
  - Breakpoints and watch points halt the processor on a specific activity.
  - A debug connection examines and modifies registers and memory, and it provides single-step execution.
- **NIDEN:** Noninvasive debugging of TrustZone for a defined non-secure domain
  - Collects information on instruction execution and data transfers.
  - Delivers trace to off-chip in real time to tools to merge data with source code on a development workstation for future analysis.
- **SPIDEN:** Invasive debugging of TrustZone for Armv8-M architecture defined secure domain
- **SPNIDEN:** Noninvasive debugging of TrustZone for Armv8-M architecture defined secure domain

2.2 Debug authentication protocol key size

The device supports two instantiations of debug authentication protocol versions. The versions are defined according to the different-sized ECDSA keys.

- Version 2.0: Uses ECDSA P-256 signature verification using ECC keys.
- Version 2.1: Uses ECDSA P-384 signature verification using ECC.

To enforce the usage of ECDSA P-384, set the ENF_CNSA field in the BOOT_CFG3 word (OTP fuse index 18) to 0x1, 0x2, or 0x3. Otherwise ECDSA P-256 algorithm is used for debug authentication.

Both the debug-authentication certificates and image-signing certificates use the same Root of Trust Keys (RoTK). When the ENF_CNSA field is set, the secure-boot image-signing key certificate chain should also use P-384 keys.

3 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 repository.

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 SB31 format using the *elftosb* tool
- Programming and configuring the devices using the *blhost* tool
- Enabling the debug port using the *nxpdebugmbox* tool
- Testing OTP configurations (before writing into the OTP fields) using *shadowregs* tool.

3.1 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.

4 Prepare the debug authentication configuration

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

4.1 Keys

CAUTION: *All visualized keys are used as examples only. Generate your own keys to secure your target devices.*

The RW61x device uses the elliptic curve digital signature algorithm (ECDSA) P-256 or P384 as the basic security authentication. Up to four root of trust private keys are held by the OEM/developer securely. Each device contains a hash of the hashes from public keys for a possibility to authenticate the signature during secure boot, firmware update, or debug authentication.

The ECC P-256 or P-384 keys are also used as the debug keys. For debug-authentication purposes, the NXP protocol uses the debug credential certificate (DC), which is signed by a RoT key and can be validated on the RW61x device.

The following sections contain information about keys and how to create keys using NXP tools. The NXP tools use standard key formats and you can generate/load your own keys.

4.1.1 Debug keys

The debug-authentication scheme has a key pair for the debug authentication protocol. The debug authentication key pair is independent from the key pairs of RoT keys.

The Debug Cedential Key (DCK) **(4)** is a user-owned key pair. The public part of the key is associated with a DC, the private part is held by the user and used to produce signatures during authentication. The DCK public key **(2)** is part of the Debug Credential (DC) certificate **(3)** and signed by one of 4 RoT private keys **(1)**. This DC certificate is then used for debug authentication **(5)**.

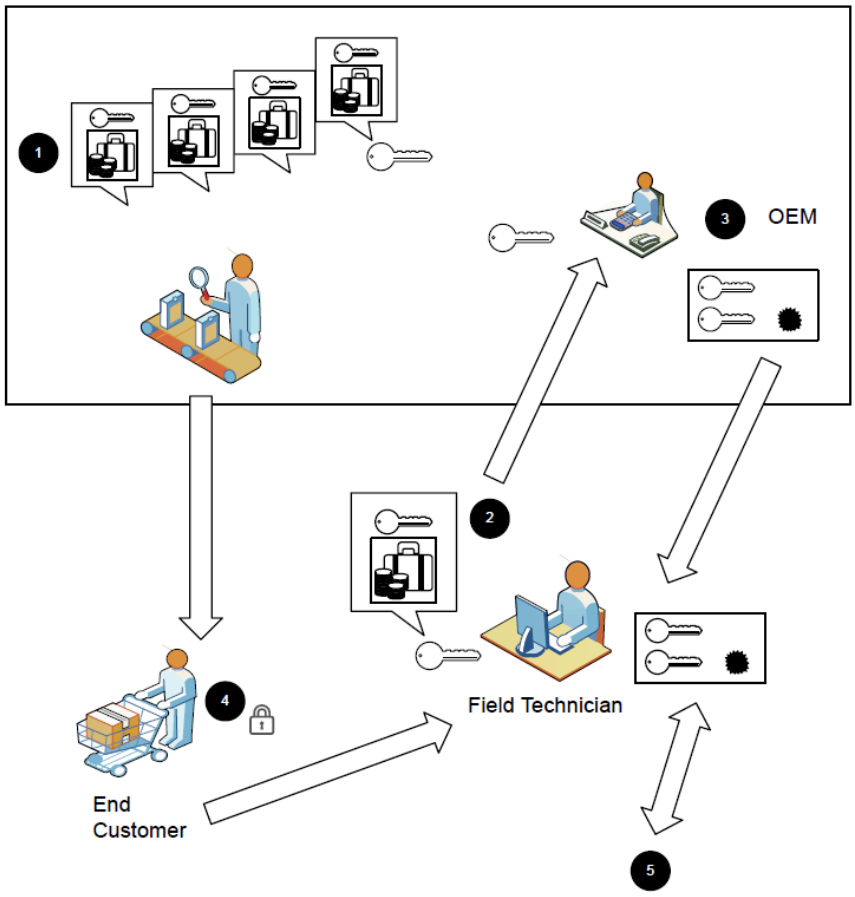

Figure 4. Key management for debug authentication

The debugging users create a key pair to authenticate with the DC. They send the DC public key to the owner of the RoT keys, who generates a DC with all the configuration needed (debug access restrictions invasive/non-invasive/secure/non-secure, vendor_usage, credential beacon). This certificate is signed by the RoT private key for authentication.

```
nxpcrypto key generate -k secp256r1 --force -o DCK_secp256r1.pem
```

4.1.2 ROTKH

ROTKH (Root Of Trust Key Hash) table is a table generated once by the OEM and permanently stored in the OTP.

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] is used. The remaining 16 bytes are unused and should be filled with zeros. The structure of this table is shown in Figure 5.

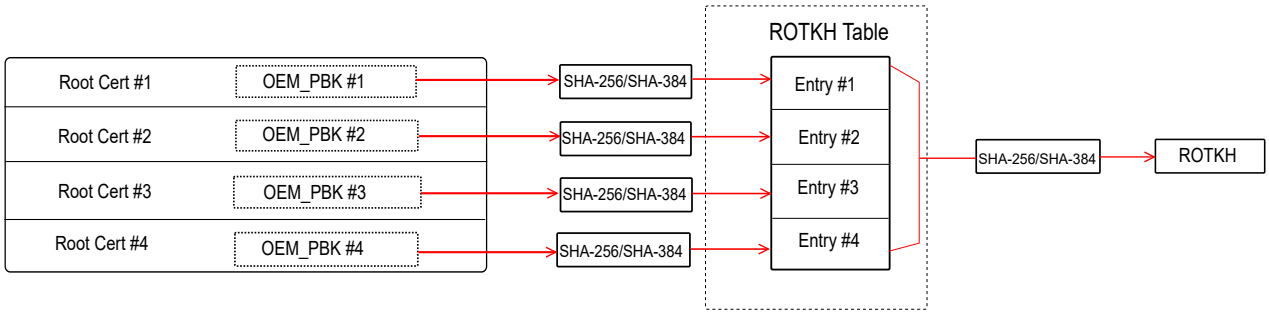

Figure 5. ROTKH generation process

The following commands generate key pairs with the expected key length using the “nxpkeygen” tool.

ECC P-256 key is used as an example in this application note.

```
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
```

Example of key creation of ECC P-384:

```
nxpcrypto key generate -k secp384r1 --force -o WORKSPACE\keys\ROT1_p384.pem
```

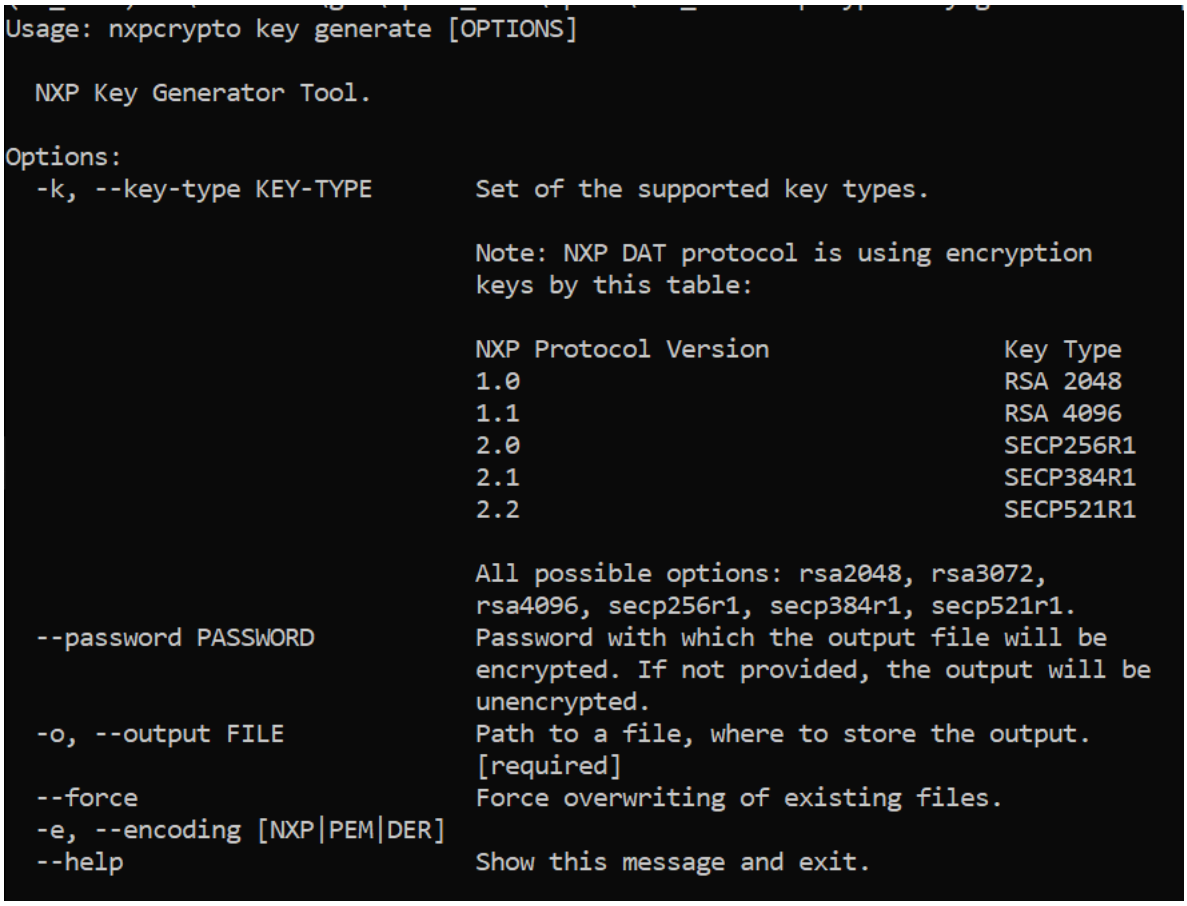

Figure 6. nxpkeygen genkey --help

The keys from the tool are in the standard format. The tool outputs *.pem* and *.pub* files. The *.pem* file contains a private key and the *.pub* file contains only a public key. The *ROT1_p256.pem* file content is as follows:

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgZcbt3v522cNlCUDU
ZRDHpnpt92VkFSYbXEPw02RohpehRANCAARO+Vt4e0TIEo/isz8DhK2mdvzihJ3w
she8BXS/Mz+5FLnpSbFmDlAbMFZCeX91D9V/6hUeWFmSgwx/1/dTStFi
-----END PRIVATE KEY-----
```

Content of *ROT1_p256.pub* file:

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAETvlbeHtEyBKP4rM/A4Stpnb84oSd
8LIXvAV0vzM/uRS56UmxZg5QGzBWQnl/dQ/Vf+oVHlhZkoMMf9f3U0rRYg==
-----END PUBLIC KEY-----
```

Example of parsing via OpenSSL:

```
openssl ec -in RoTkey0_secp256r1.pub -text -noout -pubin
```

```
read EC key
Public-Key: (256 bit)
pub:
    04:4e:f9:5b:78:7b:44:c8:12:8f:e2:b3:3f:03:84:
    ad:a6:76:fc:e2:84:9d:f0:b2:17:bc:05:74:bf:33:
    3f:b9:14:b9:e9:49:b1:66:0e:50:1b:30:56:42:79:
    7f:75:0f:d5:7f:ea:15:1e:58:59:92:83:0c:7f:d7:
    f7:53:4a:d1:62
ASN1 OID: prime256v1
NIST CURVE: P-256
```

Figure 7. RoTkey0_secp256r1.pub parsed OpenSSL

```
openssl ec -in RoTkey0_secp256r1.pem -text -noout
```

```
read EC key
Private-Key: (256 bit)
priv:
    65:c6:ed:de:fe:76:d9:c3:65:09:40:d4:65:10:c7:
    a6:7a:6d:f7:65:64:15:26:1b:5c:43:f0:d3:64:68:
    86:97
pub:
    04:4e:f9:5b:78:7b:44:c8:12:8f:e2:b3:3f:03:84:
    ad:a6:76:fc:e2:84:9d:f0:b2:17:bc:05:74:bf:33:
    3f:b9:14:b9:e9:49:b1:66:0e:50:1b:30:56:42:79:
    7f:75:0f:d5:7f:ea:15:1e:58:59:92:83:0c:7f:d7:
    f7:53:4a:d1:62
ASN1 OID: prime256v1
NIST CURVE: P-256
```

Figure 8. RoTkey0_secp256r1.pem parsed OpenSSL

These generated keys are used for device configuration and, later on in this document, for device configuration. Only public parts of keys 0-3 are used for device configuration. Hashes are calculated and saved in the device as ROTKH.

Table 2. ROTKH layout in OTP (for 256 bits keys)

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

Table 3. ROTKH layout in OTP (for 384 bits keys)

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]

Table 3. ROTKH layout in OTP (for 384 bits keys)*...continued*

Fuseword	Description	Fuseword	Description
108	ROTKH[255:224]	114	ROTKH [63:32]
109	ROTKH [223:192]	115	ROTKH [31:0]

4.2 Debug access control configuration

The boot code present in the device ROM handles the device side of the debug authentication process. The debug access control rights and security policies are configurable. The configuration fields are referred to as device configuration for credential constraints (DCFG_CC). The fields are present in the OTP fusemap.

- **DCFG_VER:** Controls the cryptographic primitives used during authentication.
- **DCFG_ROTID:** Defines the root of trust identifier (ROTID). The ROTID field is used to bind the devices to specific certificate authority (CA) keys issuing the debug credentials. These CA keys are referred as Root of Trust (RoTK) keys.
- **DCFG_UUID:** Controls whether to enforce UUID check during debug credential (DC) validation or not. If this field is set, only the DC with matching device UUID can unlock the debug access.
- **DCFG_CC_SOCU:** Specifies the access rights to various debug domains.
- **DCFG_VENDOR_USAGE:** Defines the vendor-specific debug policy use case such as DC revocations or department identifier. Use the field for revocation of already issued debug certificates.

These above fields should be programmed as part of the OEM provisioning process.

- **CC_BEACON:** Credential beacon is associated with DC and system product. With credential beacon, debug authentication can be restricted to specific parts having matching system product ID in OTP (fuse 9)

4.2.1 Protocol version (DCFG_VER)

The device supports two instantiations of debug authentication protocol versions. The versions are defined based on the different-sized ECDSA keys.

- Version 2.0: uses ECDSA P-256 signature verification using ECC keys.
- Version 2.1: uses ECDSA P-384 signature verification using ECC.

To enforce the usage of ECDSA P-384, set the ENF_CNSA field in the BOOT_CFG3 word (OTP fuse index 18) to 0x1, 0x2, or 0x3. Otherwise, the ECDSA P-256 algorithm is used for debug authentication.

Note: *The debug authentication certificates and the image signing certificates use the same Root of Trust keys (RoTK). Therefore, when the DCFG_VER field is set, the secure boot image signing key certificate chain also uses P-384 keys.*

4.2.2 Root of trust identifier (DCFG_ROTID)

The root of trust identifier used in the debug authentication protocol is composed of three elements.

1. A 256-bit cryptographic hash (SHA256) for version 2.0, or a 384-bit cryptographic hash (SHA384) for version 2.1 over the root of trust keys table. This element is the root key table hash (RKTH): a 32-byte/48-byte SHA-256/SHA-384 of SHA-256/SHA-384 hashes of up to four root public keys. The OTP fusewords 104 to 115 specify the RKTH.
2. For each key, a 3-bit field indicates the usage (OTP fuseword 18).
3. A single-bit field indicates if the root of the trust key is revoked (OTP fuseword 22).

4.2.3 Enforce UUID checking (DCFG_UUID)

The field controls whether to enforce UUID check during debug credential (DC) validation or not. If the field is set, only DC containing a UUID attribute that is an exact match to the device can unlock the debug access.

This field can be set by programming FORCE_UUID_MATCH (bit 31) in DCFG_CC_SOCU (OTP word 32) and FORCE_UUID_MATCH_NS (bit 31) in DCFG_CC_SOCU_NS (OTP word 31).

This device-specific constraint, if enabled, is in addition to all the other constraints defined and enforced by the authentication protocol.

4.2.4 Credential constraints (DCFG_CC_SOCU)

The DCFG_CC_SOCU configuration field specifies the debug access restrictions per debug domain. The access restrictions are referred as constraint attributes in this section. The debug subsystem is subdivided into multiple debug domains to allow finer access control. Table 5 shows the debug domains and their corresponding control bit position in DCFG_CC_SOCU. DCFG_CC_SOCU is composed of two components logically:

- **SOCU_PIN:** a bitmask that specifies the debug domains that the device configuration predetermines.
- **SOCU_DFLT:** the final access level for the bits of the SOCU_PIN field that the device configuration predetermines.

Table 4 shows the restriction levels.

Table 4. Access restriction levels

Restriction level	SOCU_PIN [n]	SOCU_DFLT [n]	Description
0	1	1	Access to the subdomain is always enabled. This setting is provided for the use case scenario where DCFG_CC_SOCU_NS is used to define further access restrictions before final deployment of the product.
1	0	0	Access to the subdomain is disabled at startup. But the access can be enabled through the debug authentication process by providing an appropriate Debug Credential (DC) certificate. ***Note:*** *Non-S parts (no security features) do not support the debug authentication process. This option is not available, but other options can be used to enable/disable debug access on those devices permanently.*
-	0	1	Illegal setting. If this setting is selected, the part can lock-up.
2	1	0	Access to the subdomain is disabled permanently and cannot be reversed. This setting offers the highest level of restriction.

The following OTP fields define the restriction levels.

- DCFG_CC_SOCU (OTP word 33) and DCFG_CC_SOCU_AP (OTP word 34).
- DCFG_CC_SOCU_NS (OTP word 31).
  - This OTP word can be used to define further restrictions in scenarios where OEM Tier1 and OEM Tier 2 product life-cycle states are used. These fields can be used to increase the restriction level specified in DCFG_CC_SOCU (OTP word 33) but cannot be used to reduce the restriction level.

The OTP fields are security critical. The filed are constructed with a three-fold protection mechanism to resist from several side-channel attacks:

- ECC mechanism built-in OTP controller protects the OTP words and catches any programming hacks. The OTP words can be programmed only once and all 32-bits together only.

- CRC8 checks offer a second level of protection from side-channel glitch attacks during the read transaction from OTP fuses.
- The redundant OTP word DCFG_CC_SOCU_AP (OTP word 34) provides the third layer of protection. DCFG_CC_SOCU_AP should be programmed with the exact antipole (inverse value) value of DCFG_CC_SOCU (OTP word 33). Any mismatch between DCFG_CC_SOCU and DCFG_CC_SOCU_AP is deemed as an attack and debug is disabled permanently.

The device supports the debug domains described in the tables below.

Table 5. DCFG_CC_SOCU (OTP fuseword 33)

Bit	Symbol	Description
0-7	CRC8	CRC-8/ITU of upper 3 bytes (bits 8 to 31). Since these fields are security critical, they are constructed with built-in integrity protection to protect from side-channel glitch attacks. The lower byte (0 bits to 7 bits) of these OTP words should be written with CRC-8/ITU of the upper 3 bytes (bits 8 to 31). This construction makes the probability of a successful glitch attack to flip the exact control bits extremely difficult. The CRC8 calculation should be based on x8 + x2 + x + 1 polynomial. • Polynomial=0x07, initial value= 0x00, XorOut=0x55.
8	DFLT_NIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
9	DFLT_DBGEN	Controls invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
10	DFLT_SPNIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0
11	DFLT_SPIDEN	Controls invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
12	DFLT_TAPEN	Controls TAP (Test access point) controller used for structural integrity testing of silicon by NXP as part of Return Material Analysis (RMA).
13	DFLT_CPU1NIDEN	Controls non-Invasive debugging of CPU1.
14	DFLT_CPU1 DBGEN	Controls invasive debugging of CPU1.
15	DFLT_CPU2NIDEN	Controls non-Invasive debugging of CPU2.
16	DFLT_CPU2 DBGEN	Controls invasive debugging of CPU2.
17	DFLT_ISPCMDEN	Controls whether ISP boot flow DM-AP command (command code: 0x05) can be issued after authentication.
18	DFLT_FACMDEN	Controls whether DM-AP Set FA Mode command (command code: 0x06) can be issued after authentication
19	PINNED_NIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
20	PINNED_DBGEN	Controls invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
21	PINNED_SPNIDEN	Controls non-Invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0
22	PINNED_SPIDEN	Controls invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
23	PINNED_TAPEN	Controls TAP (Test access point) controller used for structural integrity testing of silicon by NXP as part of Return Material Analysis (RMA).
24	PINNED_CPU1 NIDEN	Controls non-Invasive debugging of CPU1.
25	PINNED_CPU1 DBGEN	Controls invasive debugging of CPU1.

Table 5. DCFG_CC_SOCU (OTP fuseword 33)*...continued*

Bit	Symbol	Description
26	PINNED_CPU2 NIDEN	Controls non-Invasive debugging of CPU2.
27	PINNED_CPU2 DBGEN	Controls invasive debugging of CPU2.
28	PINNED_ ISPCMDEN	Controls whether ISP boot flow DM-AP command (command code: 0x05) can be issued after authentication.
29	PINNED_ FACMDEN	Controls whether DM-AP Set FA Mode command (command code: 0x06) can be issued after authentication
30	FORCE_UUID_ MATCH	Force UUID matching.
31	Reserved	Reserved.

Table 6. DCFG_CC_SOCU_NS (OTP fuseword 31)

Bit	Symbol	Description
0-7	CRC8	CRC-8/ITU of upper 3 bytes (bits 8 to 31). Since these fields are security critical, they are constructed with built-in integrity protection to protect from side-channel glitch attacks. The lower byte (0 bits to 7 bits) of these OTP words should be written with CRC-8/ITU of the upper 3 bytes (bits 8 to 31). This construction makes the probability of a successful glitch attack to flip the exact control bits difficult. The CRC8 calculation should be based on x8 + x2 + x + 1 polynomial. • Polynomial=0x07, initial value= 0x00, XorOut=0x55.
8	DFLT_NIDEN_NS	Controls non-Invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
9	DFLT_DBGEN_NS	Controls invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
10	DFLT_SPNIDEN_ NS	Controls non-Invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0
11	DFLT_SPIDEN_NS	Controls invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
12	DFLT_TAPEN_NS	Controls TAP (Test access point) controller used for structural integrity testing of silicon by NXP as part of Return Material Analysis (RMA).
13	DFLT_CPU1 NIDEN_NS	Controls non-Invasive debugging of CPU1.
14	DFLT_CPU1 DBGEN_NS	Controls invasive debugging of CPU1.
15	DFLT_CPU2 NIDEN_NS	Controls non-Invasive debugging of CPU2.
16	DFLT_CPU2 DBGEN_NS	Controls invasive debugging of CPU2.
17	DFLT_ISPCMDEN_ NS	Controls whether ISP boot flow DM-AP command (command code: 0x05) can be issued after authentication.
18	DFLT_FACMDEN_ NS	Controls whether DM-AP Set FA Mode command (command code: 0x06) can be issued after authentication
19	PINNED_NIDEN_ NS	Controls non-Invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.

Table 6. DCFG_CC_SOCU_NS (OTP fuseword 31)*...continued*

Bit	Symbol	Description
20	PINNED_DBGEN_NS	Controls invasive debugging of TrustZone for Arm8-M defined nonsecure domain of CPU0.
21	PINNED_SPNIDEN_NS	Controls non-Invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0
22	PINNED_SPIDEN_NS	Controls invasive debugging of TrustZone for Arm8-M defined secure domain of CPU0.
23	PINNED_TAPEN_NS	Controls TAP (Test access point) controller used for structural integrity testing of silicon by NXP as part of Return Material Analysis (RMA).
24	PINNED_CPU1 NIDEN_NS	Controls non-Invasive debugging CPU1.
25	PINNED_CPU1 DBGEN_NS	Controls invasive debugging of CPU1.
26	PINNED_CPU2 NIDEN_NS	Controls non-Invasive debugging of CPU2.
27	PINNED_CPU2 DBGEN_NS	Controls invasive debugging of CPU2.
28	PINNED_ISPCMDEN_NS	Controls whether ISP boot flow DM-AP command (command code: 0x05) can be issued after authentication.
29	PINNED_FACMDEN_NS	Controls whether DM-AP Set FA Mode command (command code: 0x06) can be issued after authentication
30	FORCE_UUID_MATCH_NS	Force UUID matching.
31	Reserved	Reserved.

4.2.5 DCFG_CC_SOCU authentication and life-cycle dependency

The tables below shows the behavior of the debug domains based on the current life-cycle state and debug authentication status. For a given life-cycle state and debug authentication status, the domain can be enabled or disabled. Or, DCFG_SOCU & DCFG_SOCU_NS configuration is used to determine the settings.

Table 7. Life-cycle status for NIDEN/DBGEN and SPNIDEN/SPIDEN CPU0

Life-Cycle	NIDEN/DBGEN		SPNIDEN/SPIDEN	
	not authenticated	authenticated	not authenticated	authenticated
Blank	disabled	disabled	disabled	disabled
Provisioned	enabled	enabled	enabled	enabled
Develop	enabled	enabled	enabled	enabled
Develop2	enabled	enabled	disabled	DCFG_SOCU, DCFG_ SOCU_NS, and CC_SOCU
In-field	disabled	DCFG_SOCU & DCFG_ SOCU_NS & CC_SOCU	disabled	DCFG_SOCU, DCFG_ SOCU_NS, and CC_SOCU
In-field Locked	disabled	disabled	disabled	disabled
Field Return OEM	enabled	enabled	enabled	enabled
Failure Analysis (FA)	enabled	enabled	enabled	enabled
Bricked	disabled	disabled	disabled	disabled

Table 8. Life-cycle status for NIDEN/DBGEN CPU1, CPU2

Life cycle	NIDEN/DBGEN	
	not authenticated	authenticated
Blank	disabled	disabled
Provisioned	enabled	enabled
NXP Develop Secure	disabled	disabled
Develop	disabled	disabled
Develop2	disabled	disabled
In-field	disabled	disabled
In-field Locked	disabled	disabled
Field Return OEM	disabled	disabled
Failure Analysis (FA)	disabled	disabled
Bricked	disabled	disabled

Table 9. Life cycle status for ISP_CMD_EN and FA_CMD_EN

Life cycle	ISP_CMD_EN		FA_CMD_EN	
	not authenticated	authenticated	not authenticated	authenticated
Blank	disabled	disabled	disabled	disabled
Provisioned	enabled	enabled	enabled	enabled
Develop	enabled	enabled	enabled	enabled
Develop2	disabled	DCFG_SOCU & DCFG_ SOCU_NS & CC_SOCU	disabled	DCFG_SOCU, DCFG_ SOCU_NS, and CC_SOCU
In-field	disabled	DCFG_SOCU & DCFG_ SOCU_NS & CC_SOCU	disabled	DCFG_SOCU, DCFG_ SOCU_NS, and CC_SOCU
In-field Locked	disabled	disabled	disabled	disabled
Field Return OEM	disabled	enabled	disabled	enabled
Failure Analysis (FA)	disabled	disabled	disabled	disabled
Bricked	disabled	disabled	disabled	disabled

4.2.6 DCFG_VENDOR_USAGE

The field defines the vendor-specific debug policy use case such as: debug credential (DC) certificate revocations, department identifier, or model identifier. Use the field for revocation of already issued debug certificates. During the debug authentication response (DAR) process, the device checks that the value specified in the vendor usage field of DC matches exactly the value programmed in DCFG_VENDOR_USAGE fields of the device configurations.

RW61x provides a 16-bit redundant OTP word called DAP_VENDOR_USAGE (OTP Word 23). A redundant OTP word can be programmed 1 bit at a time.

4.2.7 CC_BEACON_USAGE

The field defines the system-specific debug policy use case such as: restricting debug authentication to only certain devices having specific system product ID during manufacturing phase.

RW61x provides a 16-bit redundant OTP word called SYSCTL0_PRODUCT_ID (OTP Word 9).

4.3 Debug credential certificate (DC)

The debug keys created in the previous section must be inserted into the debug credential certificate.

The debug certificate contains specific configuration data and is signed by a RoT key. The configuration file is created using “nxpdebugmbox” tool.

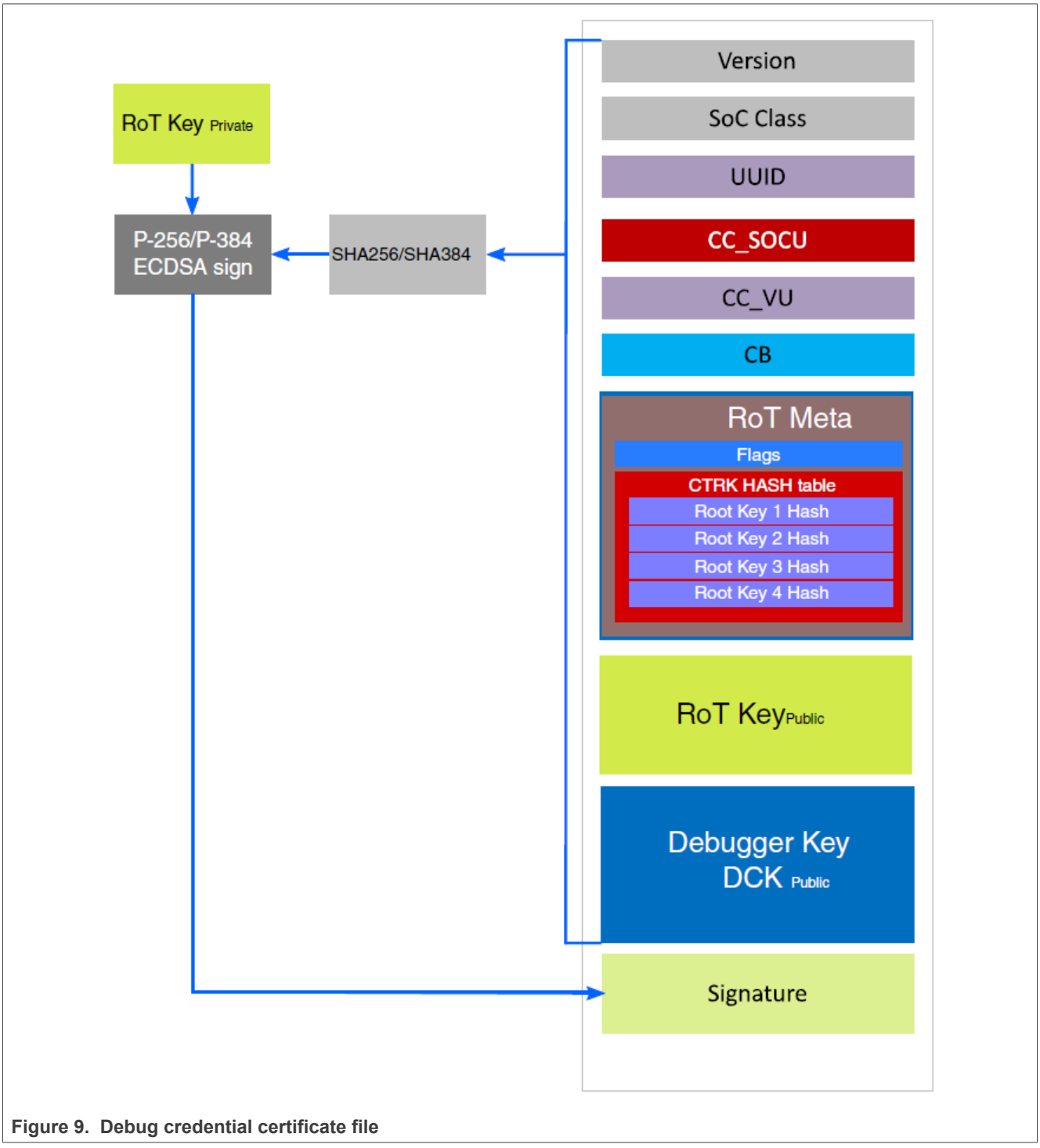

Figure 9. Debug credential certificate file

The YAML *DAT_config.yml* template for the debug credential certificate file is created using the tools:

```
nxpdebugmbox get-template -o DAT_config.yml
```

Specify your configuration in the YAML file. This configuration is done at the RoT key owner side. Using a specific configuration, you can grant access to different debug modes of RW61x target.

The configuration of the RW61x device with all debug possibilities enabled is shown below. The vendor usage is 0 and the beacon is 0. A DC file will be created for the *DCK_secp256r1.pub* public key. RoTkey0-3 public keys are used as the input for the hash calculation. This file is signed using *RoTkey0_secp256r1.pem*.

Example of RW61x YAML configuration:

```
# ============ SoC Class ============
# A unique identifier for a set of SoCs that require no SoC-specific differentiation in
# their debug authentication. The main usage is to allow a different set of debug
# domains and options to be negotiated between the device configuration and
# credentials. A class can contain just a single revision of a single SoC model, if the
# granularity of debug control warrants it.
# Examples list of possible settings:
# 0x000A:       RW61x
socc: 0x000A

# ============ Device UUID ============
# 128-bit IETF RFC4122 compliant non-sequential Universally Unique Identifier (UUID)
uuid: "00000000000000000000000000000000"

# ============ SoC Usage ============
# A CC (constraint) value that is a bit mask, and whose bits are used in an
# SoCC-specific manner. These bits are typically used for controlling which debug
# domains are accessed via the authentication protocol, but device-specific debug
# options can be managed in this way also.
cc_socu: 0xFF

# ============ Vendor Usage ============
# A CC (constraint) value that is opaque to the debug authentication protocol itself but
# which can be leveraged by vendors in product-specific ways.
cc_vu: 0

# ============ Credential Beacon & Authentication beacon ============
# A value that is passed through the authentication protocol, which is not interpreted
# by the protocol but is instead made visible to the application being debugged. A
# credential beacon is associated with a DC and is therefore vendor/RoT-signed. An
# authentication beacon is provided and signed by the debugger during the
# authentication process.
cc_beacon: 0

# ============ RoT meta-data ============
# The RoT meta-data required by the device to corroborate; the ROTID sent in the
# DAC, the field in this DC, and any additional RoT state that is not stored within the
# device. This allows different RoT identification, management and revocation
# solutions to be handled.
rot_meta:
- ROT1_p256.pub
- ROT2_p256.pub
- ROT3_p256.pub
- ROT4_p256.pub

# ============ RoT Identifier ============
# RoTID allows the debugger to infer which RoT public key(s) are acceptable to the
# device. If the debugger cannot or does not provide such a credential, the
# authentication process will fail.
rot_id: 0

# ============ Debug Credential Key ============
# A user-owned key pair. The public part of the key is associated with a DC, the
# private part is held by the user and used to produce signatures during
# authentication.
dck: DCK_secp256r1.pub

# ============ RoT signature private key ============
# Private key for for the RoT meta chosen by rot_id to sign the image.
rotk: ROT1_p256.pem
```

The DC file is generated using the “nxpdebugmbox” tool at the RoT key owner side:

```
nxpdebugmbox -p 2.0 gendc -c DAT_config_rw61x.yml -o DAT_certificate.dc
Also prints RKTH which can be used directly to test using shadowregs utility
```

5 Program the debug authentication configuration

The following sections describe ways to program the debug authentication configuration into the device before debug authentication:

- Load debug authentication configuration into the device temporarily via OTP shadows as means of testing the configuration values(Section 5.1 "Use OTP shadow with the configuration example").
- Load the debug authentication configuration into the device permanently using OTP fuses (Section 5.2 "Program RW61x configuration in OTP").

From the previous section, the following inputs are prepared:

- Debug Keys
  - DCK_secp256r1.pem
  - DCK_secp256r1.pub
- ROTKH
  - ROT1_p256.pem
  - ROT2_p256.pem
  - ROT3_p256.pem
  - ROT4_p256.pem
- DCFG_VER
- DCFG_ROTID
- DCFG_UUID
- DCFG_CC_SOCU
- DCFG_VENDOR_USAGE
- Debug Credential Certificate
  - DAT_certificate.dc

5.1 Use OTP shadow with the configuration example

As programming the fuses in OTP cannot be reversed, use the mechanism via the OTP shadow to test the device configuration before its programming to OTP. Set up the device for debug authentication or secure boot. When the device is in the development lifecycle, the shadow registers are loadable (through debug port or by an application). Shadows are cleared upon reset and reloaded again from OTP fuses.

The ROM provides a mechanism to override the reloading of OTP shadows from OTP fuse and instead load from SRAM. Therefore, it is possible to:

- Configure a secure boot by writing to SRAM.
- Load the configuration values to the shadow registers.
- Debug the secure boot flow.
- Program fuses (change the device life cycle) once the device configuration is stable.

Note: *Not all fuses have shadow registers. To get the list of OTP fuses with which can be overridden via shadows, follow the steps below.*

This section shows how to use the SPSDK shadowreg tool to test the OTP configurations before writing into the OTP fields. For details on `shadowregs` commands, refer to [1].

Note: *Prerequisite:*

- *The device LifeCycle is Develop (0x0303).*
- *The Life-Cycle state in the OTP shadow to be configured is either Develop2 (0x0707) or In-Field (0x0F0F).*

Step 1 – Generate 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
```

Use *saveconfig_rw61x.yml* as an input to test the required OTP fuses.

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

1. ROTKH
2. PRIMARY_BOOT_SOURCE bits in BOOT_CFG0
3. DCFG_CC_SOCU_NS
4. DCFG_CC_SOCU
5. LifeCycle

The rest of the fuse settings can be removed or kept default. Example of configuration (*updated_shadow_rw61x.yml*):

```
description:
  device: rw61x
  author: Documenration
registers:
  BOOT_CFG0:
    value: '0x0001'
  BOOT_CFG3:
    value: '0x0'
  SEC_BOOT_CFG2:
    value: '0x0'
  SEC_BOOT_CFG1: # DAP Vendor Usage configurations fuse word
    bitfields:  # The register bitfields
      DAP_VENDOR_USAGE: '0x0000'  # Offset: 0b, Width: 16b, Description: Lower 16-bits of Vendor
 Usage field in Debug Credentials defined in NXP's Debug Authentication Protocol specifications
 Version 1.0.
      Redundancy: '0x0000' # Offset: 16b, Width: 16b, Description: The word is set in redundant
 mode.
  DCFG_CC_SOCU_NS:
    value: '0x3FFFFF14'
  DCFG_CC_SOCU:
    value: '0x3FFFFF14'
  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 provided either as a 256-bit string (384-bit string) 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 the updated shadow RW61x configurations.

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

Step 5 – Reset RW61x.

The reset command implements the shadow changes.

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

After the reset command, the application boots from flash. The shadows have been loaded with new values and the device reboots with new LifeCycle as selected in the configuration file.

In LifeCycle, Develop2 and In-Field debug ports have been disabled. To enable debug ports, the debug authentication must be executed with the correct credentials.

Step 6 – Issue the debug authentication command.

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

If all settings are correct, the debug authentication ends successfully, and the debug ports is enabled. After successful debug authentication, verify the changed device configuration in the shadow either by saving or by printing 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:       0x00000001
Register raw value:   0x00000001

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:
 B9ED9C3CB1359A75A63BA0D6AB5E33CA160A597123F51564D718E093701F940A00000000000000000000000000000000
Register raw value:
 3C9CEDB9759A35B1D6A03BA6CA335EAB71590A166415F52393E018D70A941F700000000000000000000000000000000
```

If the debug authentication fails, review the configuration values and repeat the steps. When you are certain of the setup and OTP configuration values, use the shadowreg application to generate a batch script and program RW61x OTP fuses permanently.

Step 8 – Generate the OTP fuse word writing script.

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

5.2 Program RW61x configuration in OTP

CAUTION: *OTP fuses cannot be changed once they are programmed. Only load a valid configuration in the device.*

OTP fuses can be programmed using blhost tool with device booted in ISP mode.

From batch script generated at the end of previous chapter, *blhost_rw61x_script.bat* must be updated with the correct ISP boot host interface.

```
# blhost_rw61x_script.bat
# BLHOST fuses programming script
# Generated by SPSDK 2.0.0
# Chip: rw61x rev:a1

# Fuse BOOT_CFG0, index 15 and value: 0x00000001.
blhost -p com22 -t 60000 efuse-program-once 0xf 0x00000001
# Fuse BOOT_CFG3, index 18 and value: 0x00000000.
blhost -p com22 -t 60000 efuse-program-once 0x12 0x00000000
# Fuse LIFE_CYCLE_STATE, index 45 and value: 0x00000f0f.
blhost -p com22 -t 60000 efuse-program-once 0x2d 0x00000f0f
# Fuse RKTH0, index 104 and value: 0x6b0ca180.
blhost -p com22 -t 60000 efuse-program-once 0x68 0x6b0ca180
# Fuse RKTH1, index 105 and value: 0xe53df964.
blhost -p com22 -t 60000 efuse-program-once 0x69 0xe53df964
# Fuse RKTH2, index 106 and value: 0xf76f895f.
blhost -p com22 -t 60000 efuse-program-once 0x6a 0xf76f895f
# Fuse RKTH3, index 107 and value: 0x02c356f9.
blhost -p com22 -t 60000 efuse-program-once 0x6b 0x02c356f9
# Fuse RKTH4, index 108 and value: 0xf5da7cee.
blhost -p com22 -t 60000 efuse-program-once 0x6c 0xf5da7cee
# Fuse RKTH5, index 109 and value: 0xc7093a41.
blhost -p com22 -t 60000 efuse-program-once 0x6d 0xc7093a41
# Fuse RKTH6, index 110 and value: 0xb582f8c2.
blhost -p com22 -t 60000 efuse-program-once 0x6e 0xb582f8c2
# Fuse RKTH7, index 111 and value: 0x3bea333a.
blhost -p com22 -t 60000 efuse-program-once 0x6f 0x3bea333a
# Fuse RKTH8, index 112 and value: 0x00000000.
blhost -p com22 -t 60000 efuse-program-once 0x70 0x00000000
# Fuse RKTH9, index 113 and value: 0x00000000.
blhost -p com22 -t 60000 efuse-program-once 0x71 0x00000000
# Fuse RKTH10, index 114 and value: 0x00000000.
blhost -p com22 -t 60000 efuse-program-once 0x72 0x00000000
# Fuse RKTH11, index 115 and value: 0x00000000.
blhost -p com22 -t 60000 efuse-program-once 0x73 0x00000000
```

```
call blhost_rw61x_script.bat
```

Each OTP fuse can be programmed individually. The example below shows how to program OTP fuse index 104 with value 0x8007e6.

```
/*Copyright 2023 NXP. NXP Confidential. This software is owned or controlled by NXP
* and may only be used strictly in accordance with the applicable license terms found at
* https://www.nxp.com/LA_OPT_NXP_SW. The "production use license" in Section 2.3 in the
* NXP SOFTWARE LICENSE AGREEMENT is expressly granted for this software.
*/

blhost.exe -p <COMP_PORT>,<BAUD_RATE> -t <TIMEOUT> -- efuse-program-once <FUSE_INDEX>
 1234ABCD

blhost.exe -p COM3,9600 -t 60000 -- efuse-program-once 104  008007e6
Ping responded in 1 attempt(s)
Inject command 'efuse-program-once'
Successful generic response to command 'efuse-program-once'
Response status = 0 (0x0) Success.
```

Command to read back the OTP fuse:

```
blhost.exe -p COM3,9600 -t 60000 -- efuse-read-once 104
Ping responded in 1 attempt(s)
Inject command 'efuse-read-once'
Response status = 0 (0x0) Success.
Response word 1 = 4 (0x4)
Response word 2 = 8390630 (0x8007e6)
```

5.3 Program the firmware

You can load any image into the device. This test does not cover loading a signed image, because it is not enabled in the secure boot.

Figure 10 shows an example in MCUXpresso IDE via an on-board debugger, which is part of the EVK. The steps are similar with another IDE or debugger. The default "Led_blinky" SDK example is used.

- Load the image in RW61x
- Run the image
- Verify that the image is valid and running after a reset

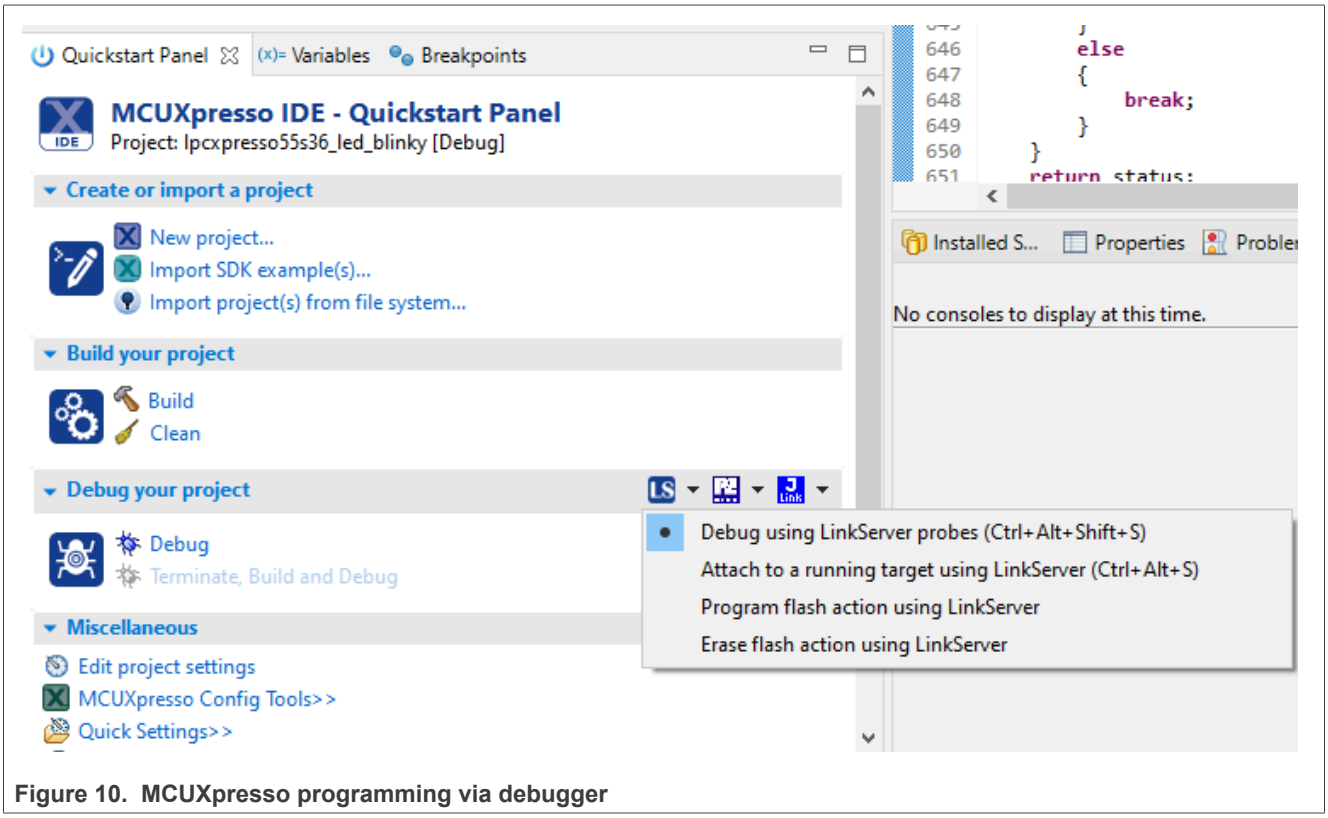

Figure 10. MCUXpresso programming via debugger

After switching the ISP pins to "AUTO" or "memory" boot, the device should reboot to the external flash, if the valid image is discovered after a reset.

6 Perform debug authentication

Before the debug authentication, check the value of the DEBUG_AUTH_BEACON register, which is part of the SYSCON peripheral. Next, disable the debug session in the IDE. It is not possible to use the debug from more than one instance (IDE/debug-authentication tool). The value must match the value in the configuration file *DAT_config_rw61x.yml*.

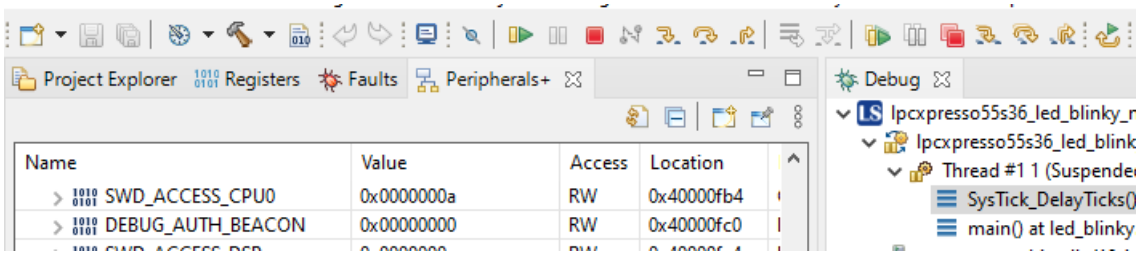

Figure 11. MCUXpresso IDE DEBUG_AUTH_BEACON default

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

After successful debug authentication, DEBUG_AUTH_BEACON (stored in WO_SCRATCH_REG2) contains updated information. The register WO_SCRATCH_REG2 contains data from:

- The authentication beacon [31:16] that is defined during the authentication session.
- The credential beacon [15:0] that is defined on the owner side of the RoT keys during the debug credential file generation and signing.

The information is communicated from the tool via a debugger. The debug mailbox is used to communicate with the target device.

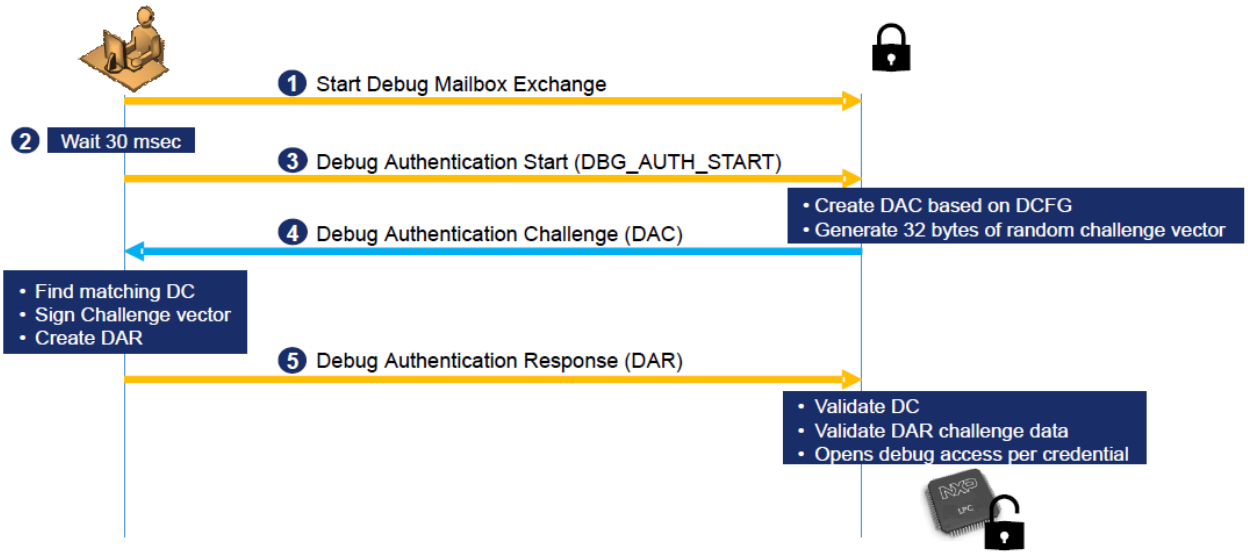

Figure 12. Debug authentication flow

After the debug-authentication start command is sent, the device sends a debug authentication challenge to the tool.

The tool must create a debug authentication response (DAR) and send it to the device. The device validates the DAR and sets up the configuration, as defined in the DAR.

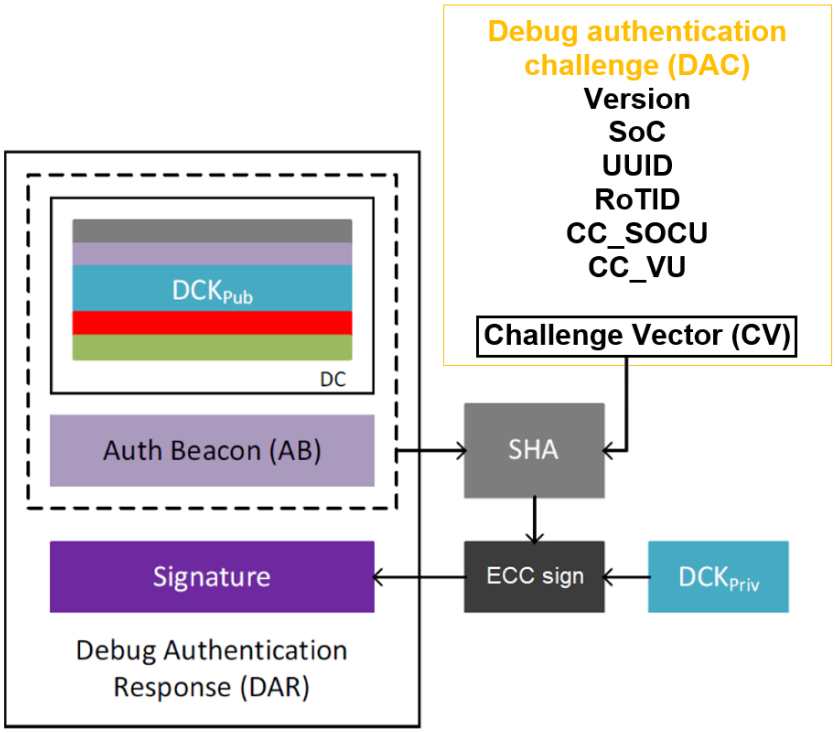

Figure 13. Debug authentication response and challenge

For example, send the authentication beacon 1. This authentication beacon is loaded into the DEBUG_AUTH_BEACON register, which is part of SYSCON. To validate the debug authentication, check DEBUG_AUTH_BEACON register. If using the MCUXpresso IDE, attach the running target and check the "Peripherals+" sheet.

After successful authentication, the debug ports are enabled as per the device configuration used. The debug authentication can be also used to enable ISP commands or Set FA mode commands (if allowed) by configuring the relevant bits in DCFG_SOCU parameters.

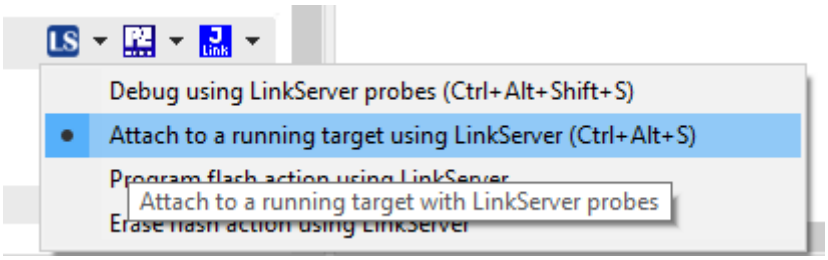

Figure 14. MCUXpresso IDE attaching to running target via debugger

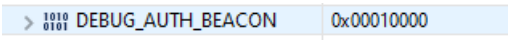

Figure 15. MCUXpresso DEBUG_AUTH_BEACON after successful authentication

7 Acronyms and abbreviations

Table 10. Abbreviations

Acronym	Description
CA	Certificate authority
DAP	Debug access port
DAR	Debug authentication response
DC	Debug credential
DCFG_CC	Device configuration for credential constraints
ECDSA	Elliptic curve digital signature algorithm
OTP	One time programmable
RKTH	Root keys table hash
RMA	Return material analysis
ROTID	Root of trust identifier
RoTK	Root of trust key
SPSDK	Secure provisioning SDK
SWD	Serial wire debug
UUID	Universally unique identifier

8 References

[1] User Guide – shadowregs (link)
[2] Webpage – Secure Provisioning SDK (SPSDK) (link)
[3] Webpage – MCUXpresso Secure Provisioning (SEC) Tool (link)

9 Revision history

Table 11. Revision history

Document ID	Release date	Description
AN13814 v.4.0	14 November 2024	• Changed the access to public. No changes in the content.
AN13813 v.3.0	12 December 2023	• Section 1 "Introduction": added a note. • Section 8 "References": updated. • Section 2 "Debug authentication overview": updated. • Section 4.1.1 "Debug keys": updated. • Section 4.1.2 "ROTKH": updated. • Section 4.2 "Debug access control configuration": updated. • Section 4.2.2 "Root of trust identifier (DCFG_ROTID)": updated. • Section 4.2.3 "Enforce UUID checking (DCFG_UUID)": updated. • Section 4.2.4 "Credential constraints (DCFG_CC_SOCU)": updated. • Section 4.2.5 "DCFG_CC_SOCU authentication and life-cycle dependency": updated. • Section 4.2.7 "CC_BEACON_USAGE": updated. • Section 4.3 "Debug credential certificate (DC)": updated. • Section 5 "Program the debug authentication configuration": updated. • Section 5.1 "Use OTP shadow with the configuration example": updated. • Section 5.2 "Program RW61x configuration in OTP": updated. • Section 6 "Perform debug authentication": updated.
AN13813 v.2.0	6 October 2023	• Section 1 "Introduction": updated. • Section 3.1 "Install SPSDK": updated. • Section 5.1 "Use OTP shadow with the configuration example": updated. • Section 10 "Note about the source code in the document": added.
AN13813 v.1.0	16 May 2023	• Initial version

10 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