AN13037

LPC55Sxx Debug Authentication

Rev. 3 — 12 December 2023 Application note

Document information

Information	Content
Keywords	LPC55Sxx, Debug Authentication Protocol (DAP), field technician, Debug Credential certificate (DC)
Abstract	This application note describes using Debug Authentication Protocol (DAP) as a mechanism to authenticate the debugger.

1 Introduction

The LPC55Sxx 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 LPC55Sxx 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. Figure 1 shows an example usage of the debug authentication. 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.

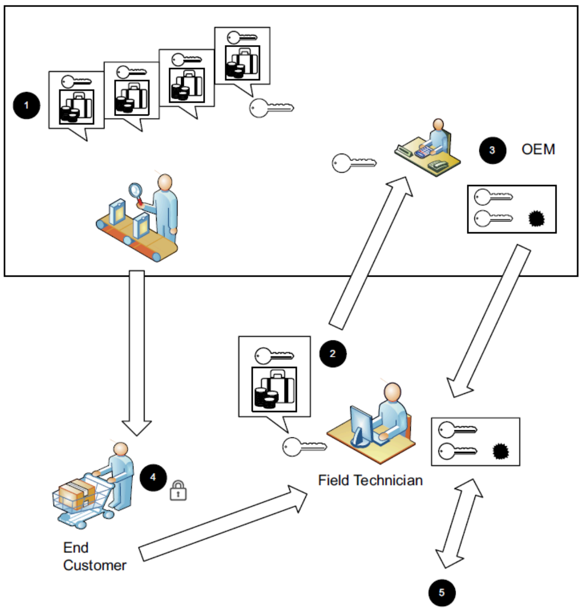


Figure 1. Debug Authentication protocol usage example

Note: *This application note is based on legacy tools and flow, but the description of the secure flow within the chip still applies. We recommend using the latest tools (the MCUXpresso Secure Provisioning (SEC) Tool and SPSDK) instead of following the steps presented here. For any questions, contact your local support.*

2 Overview

The debug port uses the Single-Wire Debug (SWD) connection. The ARMv8-M extension specifies a possibility to secure the debug port with an option to open the debug port on a device that has been locked. NXP provides a debug authentication protocol based on different-size cryptography keys.

Version 1.0 uses the RSASSA-PKCS1-v1_5 signature verification using RSA keys with a 2048-bit modulus and a 32-bit exponent.

Version 1.1 uses the RSASSA-PKCS1-v1_5 signature verification using RSA keys with a 4096-bit modulus and a 32-bit exponent.

The debug authentication scheme on LPC55Sxx assures that the debugger, in possession of the required debug credentials, can successfully authenticate over the debug interface and access-restricted parts of the device.

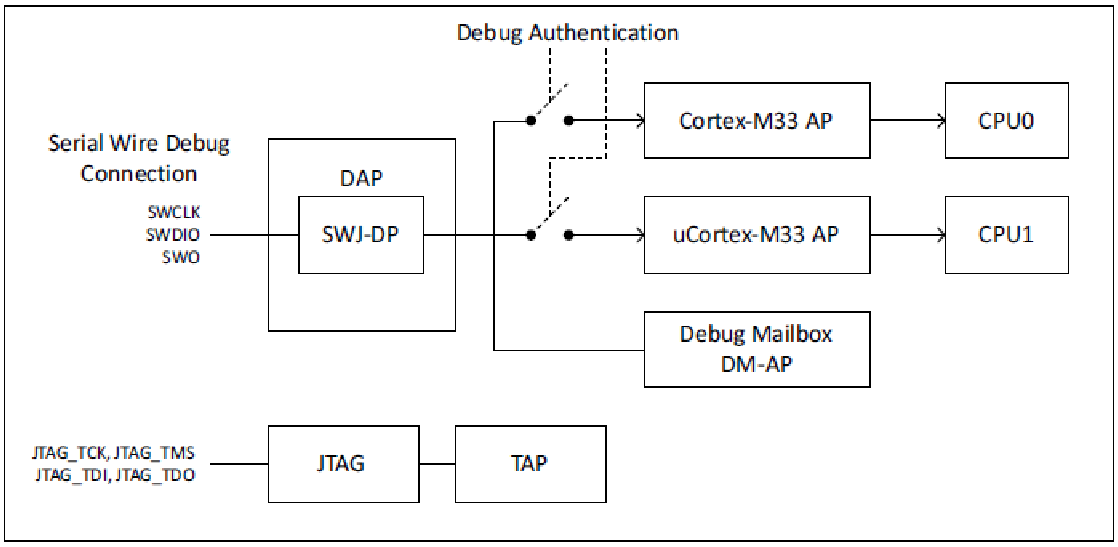


Figure 2. Internal debug system elements and connections

- DAP: The Debug Access Port has a Serial Wire port (SWJ-DP) that interprets the data coming in and routes it to an appropriate Access Port (AP).
- CPU0 AP: Debug access port for the Cortex-M33 core instantiated as CPU0.
- CPU1 AP: Debug access port for the Cortex-M33 core instantiated as CPU1. This instance of CM33 does not have a security extension (TrustZone for Armv8-M).
- DM-AP: Debug access port for the Debug Mailbox (DM). The DM is used to communicate with the code executing from the ROM by sending/receiving messages.
  - This port is always enabled and the external world can send and receive data to/from the ROM.
  - This port is used to implement the NXP debug authentication protocol.

The SWJ-DP and DM-AP blocks are always accessible through the SWD interface. The remaining blocks (CPU0 and CPU1 AP) are enabled/disabled under the hardware state machine and software control.

The DAP for CPU0 is disabled during a power-on reset or an assertion of the reset pin and it is enabled by the ROM if/when the correct debug initiation procedures are followed. If the DAP is not being used, the debug enablement protocol can be used to initiate a debug session. The debug authentication process allows the control of the DBGEN, NIDEN, SPIDEN, and SPNIDEN signals generated by the Cortex-M33, as described below.

DBGEN: Invasive debugging of TrustZone for the Arm8-M defined non-secure domain.

- Breakpoints and watch points to halt the processor on a specific activity.
- A debug connection to examine and modify registers and memory and provide single-step execution.

NIDEN: Non-invasive debugging of the TrustZone for the Arm8-M defined non-secure domain.

- A collection of information on instruction execution and data transfers.
- Deliver the trace to the off-chip in real time to tools to merge data with source code on a development workstation for future analysis.

SPIDEN: Invasive debugging of the TrustZone for the Arm8-M defined secure domain.

SPNIDEN: Non-invasive debugging of the TrustZone for the Arm8-M defined secure domain.

CPU1 is in the reset mode by default and the reset must be released by CPU0 to make the CPU1 AP accessible. The debug access port for CPU1 (Cortex-M33 core) is disabled on reset and enabled by the hardware state machine.

This port has additional control to enable/disable different features as described below, controlled through the DEBUG_FEATURES register (offset 0xFA4 in SYSCON) and the DEBUG_FEATURES_DP register (offset 0xFA8 in SYSCON).

The Debugger Mailbox (DM) AP is a register-based mailbox that is accessible by CPU0 and the device Debug Port (DP) of the MCU. This port is always enabled and an external host communicating via the SWD interface can exchange messages and data with the boot code executing from ROM on CPU0. This port is used to implement the NXP debug authentication protocol.

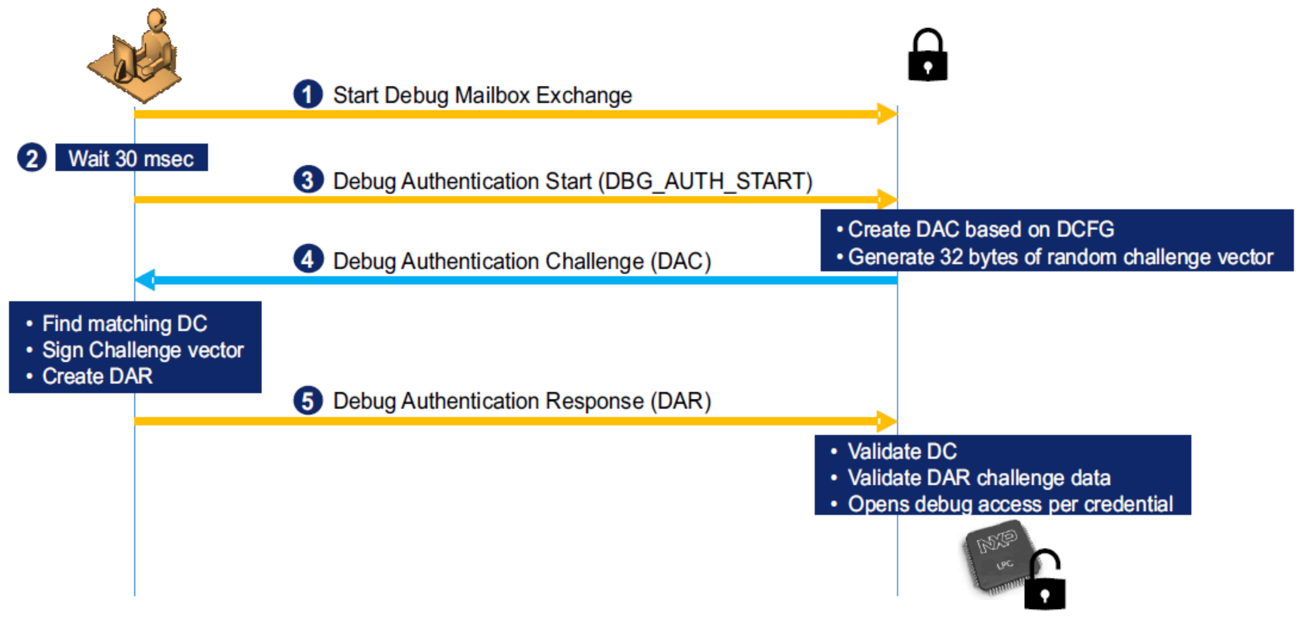

1 Start Debug Mailbox Exchange
2 Wait 30 msec
3 Debug Authentication Start (DBG_AUTH_START)
• Create DAC based on DCFG
• Generate 32 bytes of random challenge vector
4 Debug Authentication Challenge (DAC)
• Find matching DC
• Sign Challenge vector
• Create DAR
5 Debug Authentication Response (DAR)
• Validate DC
• Validate DAR challenge data
• Opens debug access per credential

Figure 3. Debug authentication flow

3 Sample example application

The section describes a sample example application.

3.1 Environment

This section gives information on the hardware and software environment.

3.1.1 Hardware environment

- Board:
  - LPC55S69EVK- Rev. A2 with 1B silicon.
- Debugger:
  - Integrated CMSIS-DAP debugger on the board.
- Miscellaneous:
  - 1 micro-USB cable.

  - PC.
- Board setup:
  - Connect the micro-USB cable to the PC and the P6 link on the board for loading and running the demo.

3.1.2 Software environment

- Toolchain:
  - MCUXpresso IDE 11.1.1 or later
  - Python 3.6 or newer
  - SPSDK 1.4.0 - https://pypi.org/project/spsdk
- Software package:
  - SDK_2.8.2_LPCXpresso55S69

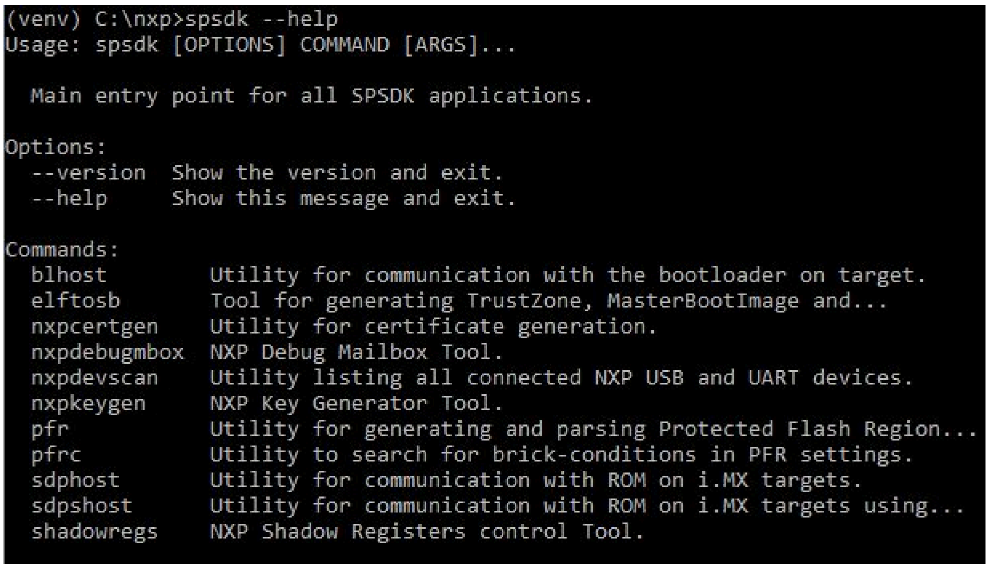
3.2 Steps

The process below describes how to install the SPSDK and use the example configuration for the LPC55S69 device to enable the debug authentication as a feature. In this example, all debug ports are enabled for the debug authentication, except for the In-System Programming (ISP) mode. ISP mode is enabled all the time. ISP mode is used for testing purposes if a wrong configuration is loaded. There is a possibility to reload the configuration. This is not expected in the end-customer final configuration. The example can be found in the associated software package of this application note.

Be careful with the device configuration. When the SHA-256 digest is written into the CMPA or a wrong configuration is loaded, there is no way back to unlock the device.

3.2.1 Installing SPSDK

1. Install Python 3.6 or newer into your laptop.
2. It is recommended to create a Python virtual environment in your workspace:
   ```
   python -m venv nxp\venv
   ```
   (The example uses the *C:\nxp* path as the root folder).
   Enable your Python virtual environment:
   ```
   venv\Scripts\activate
   ```
   After a valid activation of the virtual environment, you will see `(venv) C:\nxp>)` in your command line.
3. Install the SPSDK into your VENV:
   ```
   pip install -U spsdk
   ```
4. Check if the SPSDK is correctly installed into your VENV:
   ```
   spsdk --help
   ```



```
(venv) C:\nxp>spsdk --help
Usage: spsdk [OPTIONS] COMMAND [ARGS]...

  Main entry point for all SPSDK applications.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  blhost        Utility for communication with the bootloader on target.
  elftosb       Tool for generating TrustZone, MasterBootImage and...
  nxpcertgen    Utility for certificate generation.
  nxpdebugmbox  NXP Debug Mailbox Tool.
  nxpdevscan    Utility listing all connected NXP USB and UART devices.
  nxpkeygen     NXP Key Generator Tool.
  pfr           Utility for generating and parsing Protected Flash Region...
  pfrc          Utility to search for brick-conditions in PFR settings.
  sdphost       Utility for communication with ROM on i.MX targets.
  sdpshost      Utility for communication with ROM on i.MX targets using...
  shadowregs    NXP Shadow Registers control Tool.
```

Figure 4. SPSDK - help output

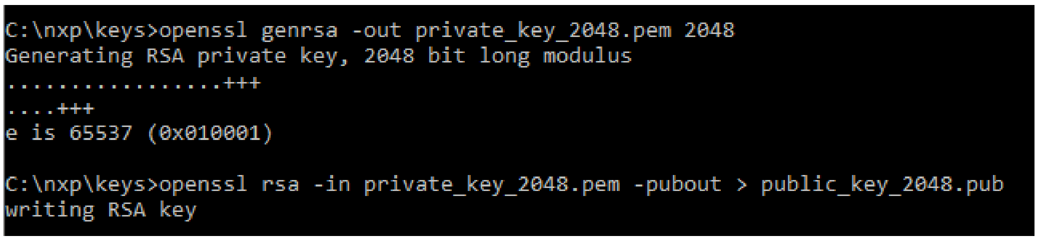
3.2.2 Load/generate keys

The RSA keypairs are used for authentication. A private key is used to sign the debug credential certificate and a hash of the Root of Trust public Keys (RoTK) is stored in the non-volatile memory of the MCU (PFR).

Locate the created and used cryptographic keys or generate keypairs for securing the device or generate new keypairs for the device configuration and security:

```
openssl genrsa -out rotk0_rsa_2048.pem 2048
```

```
openssl rsa -in rotk0_rsa_2048.pem -pubout > rotk0_rsa_2048.pub
```



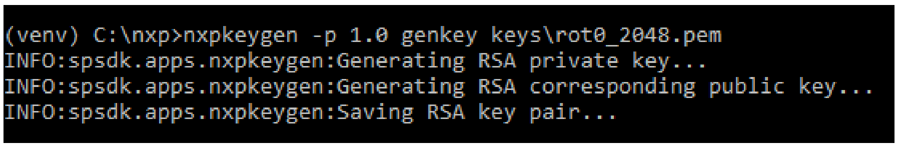
```
C:\nxp\keys>openssl genrsa -out private_key_2048.pem 2048
Generating RSA private key, 2048 bit long modulus
.................+++
....+++
e is 65537 (0x010001)

C:\nxp\keys>openssl rsa -in private_key_2048.pem -pubout > public_key_2048.pub
writing RSA key
```

Figure 5. Generate key pairs by OpenSSL

Otherwise, use the NXPKEYGEN tool to generate a key pair in your VENV:

```
nxpkeygen -p 1.0 genkey keys\rot0_2048.pem
```



```
(venv) C:\nxp>nxpkeygen -p 1.0 genkey keys\rot0_2048.pem
INFO:spsdk.apps.nxpkeygen:Generating RSA private key...
INFO:spsdk.apps.nxpkeygen:Generating RSA corresponding public key...
INFO:spsdk.apps.nxpkeygen:Saving RSA key pair...
```

Figure 6. Generate key pairs by NXPKEYGEN

You must have four Roots of Trust (RoT) key pairs and one Debug Credential Key (DCK) pair.

3.2.3 Creating configuration for device

The LPC55-family devices have a special memory called Protected Flash Region (PFR). This region is used for device configuration during onboarding and also during future updates. The PFR includes the Customer Manufacturing Programmable Area (CMPA) and Customer Field Programmable Area (CFPA).

Initially, there is a possibility to generate an example configuration file:

```
pfr get-cfg-template -d lpc55s6x -t cmpa -o cmpa_config.yml
```

```
pfr get-cfg-template -d lpc55s6x -t cfpa -o cfpa_config.yml
```

For debug authentication, the following are the most important registers:

CMPA:

CC_SOCU_PIN: configuration that specifies debug access restrictions per a debug domain. In the PIN, you can set up 0 - access is controlled by debug authentication or 1 - access restrictions are always fixed.

CC_SOCU_DFLT: configuration that specifies debug access restrictions per a debug domain. If the fixed access restrictions in the PIN are selected, then 1- enabled and 0 - disabled. If the debug authentication is selected in the PIN, the only correct setting for the DFLT is 0.

PIN/DFLT bit field settings:

- 1/1 - debug access always enabled (pinned)
- 1/0 - debug access always disabled (pinned)
- 0/0 - debug access enabled using debug authentication

The following bit fields in the CC_SOCU_PIN and CC_SOCU_DFLT registers configure the different security settings for the MCU:

NIDEN: Controls non-invasive debugging of the TrustZone non-secure domain of CPU0

DBGEN: Controls invasive debugging of the TrustZone non-secure domain of CPU0

SPNIDEN: Controls non-invasive debugging of the TrustZone secure domain of CPU0

SPIDEN: Controls invasive debugging of the TrustZone secure domain of CPU0

TAPEN: Controls the TAP (Test Access Point) controller

CPU1_DBGEN: Controls invasive debugging of CPU1

ISP_CMD_EN: Controls whether the ISP boot flow can be issued

FA_CMD_EN: Controls whether the Set FA Mode command can be issued

ME_CMD_EN: Controls whether the Bulk Erase command can be issued

CPU1_NIDEN: Controls non-invasive debugging of CPU1

UUID_CHECK: Enables checking during the DAR-value-specified UUID in DC

VENDOR_USAGE: During the Debug Authentication Response (DAR) processing, the device checks that the value specified in the “Vendor Usage” field of the debug credential certificate matches exactly the value programmed in the “VENDOR_USAGE” fields of device configurations. Upper 2 bytes from CMPA.VENDOR_USAGE, lower 2 bytes from CFPA.VENDOR_USAGE.

RKTH: The MCU supports up to four RoT keys for the secure boot and debug authentication. Later, these individual RoT keys can be revoked. The Root Key Table Hash (RKTH) is the value programmed in the CMPA, and it is an SHA-256 hash of the RoT public keys.

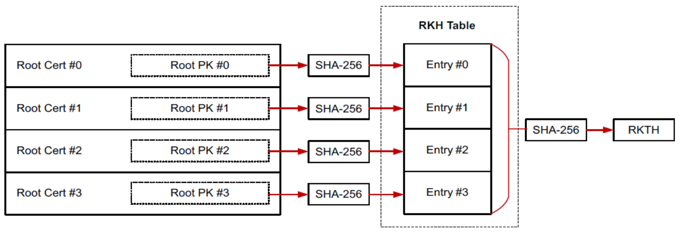


Figure 7. RKTH generation process

CFPA:

VERSION: The CFPA update mechanism checks the value of the version at each update. The new version must be higher than the previous one.

ROTKH_REVOKE: The RoT keys programmed in the MCU can be revoked later. This field specifies which keys were revoked and which can still be used for the debug authentication and secure boot.

VENDOR_USAGE: During the Debug Authentication Response (DAR) processing, the device checks that the value specified in the “Vendor Usage” field of the debug credential certificate matches exactly the value programmed in the “VENDOR_USAGE” fields of device configurations. Upper 2 bytes from CMPA.VENDOR_USAGE, lower 2 bytes from CFPA.VENDOR_USAGE.

DCFG_CC_SOCU_PIN/DCFG_CC_SOCU_DFLT: They can further restrict the debug access configured in the CMPA. Because the CMPA is programmed at manufacturing, the CFPA settings can be updated later to tighten the restrictions. The CFPA settings can be programmed to be the same as the CMPA to keep the same restrictions. Here are the possible combinations where the CFPA is different from the CMPA, tightening the debug access restrictions:

CMPA.DCFG_CC_SOCU -> CFPA.DCFG_CC_SOCU

- Pinned enabled (1/1) -> Enabled through debug authentication (0/0)
- Pinned enabled (1/1) -> Pinned disabled (1/0)
- Enabled through Debug Authentication (0/0) -> Pinned disabled (1/0)

After the modification of the configuration YAML files, it is possible to generate binary files. The inputs to the "pfr generate-binary" command are the CMPA/CFPA YAML configuration input files **-c**, calculate inverse registers where needed **-i**, root of trust keys **-f** (it is important to have a correct order of RoT keys) or **-e** if the ELFTOSB configuration file is used, and the output name of the BIN file **-o.** As an example, the *cmpa_example_config.yml* and *cfpa_example_config.yml* files are attached (keep in mind that there are update rules for the CFPA-like version that it must be higher than the previous CFPA content, and so on) for the LPC55S69 device. For *cmpa.bin*, only one of these options is used.

```
pfr generate-binary -c cmpa_config.yml -i -e config_elftosb.json -o cmpa.bin
pfr generate-binary -c cmpa_config.yml -i -f keys\rotk0_rsa_2048.pub -f keys
\rotk1_rsa_2048.pub -f keys\rotk2_rsa_2048.pub -f keys\rotk3_rsa_2048.pub -o
 cmpa.bin
pfr generate-binary -c cfpa_config.json -i -o cfpa.bin
```

3.2.4 Creating debug authentication credential certificate

NXP provides a debug authentication protocol, which is a challenge-response mechanism. When the debugger sends the debug authentication start command to the device through the debug mailbox, the device generates the Debug Authentication Challenge (DAC). The DAC message includes the following elements, further documented in the "Debug Authentication Challenge" section of the MCU user manual.

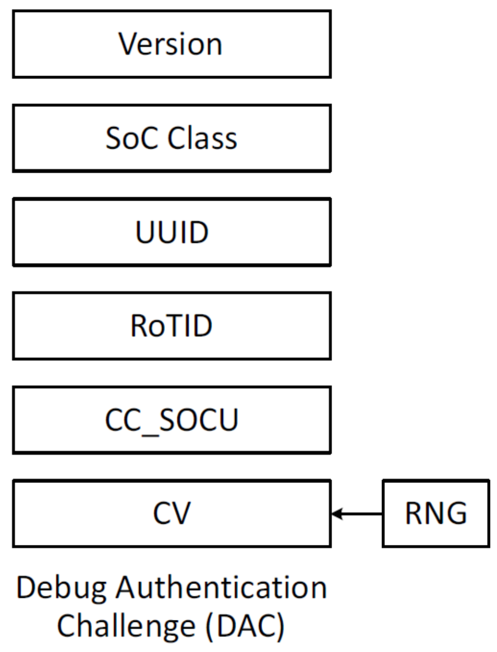


Figure 8. Debug Authentication Challenge (DAC) fields

When the debug authentication tool receives this DAC, it calculates the Debug Authentication Response (DAR), which also contains the Debug Credential Certificate (DC).

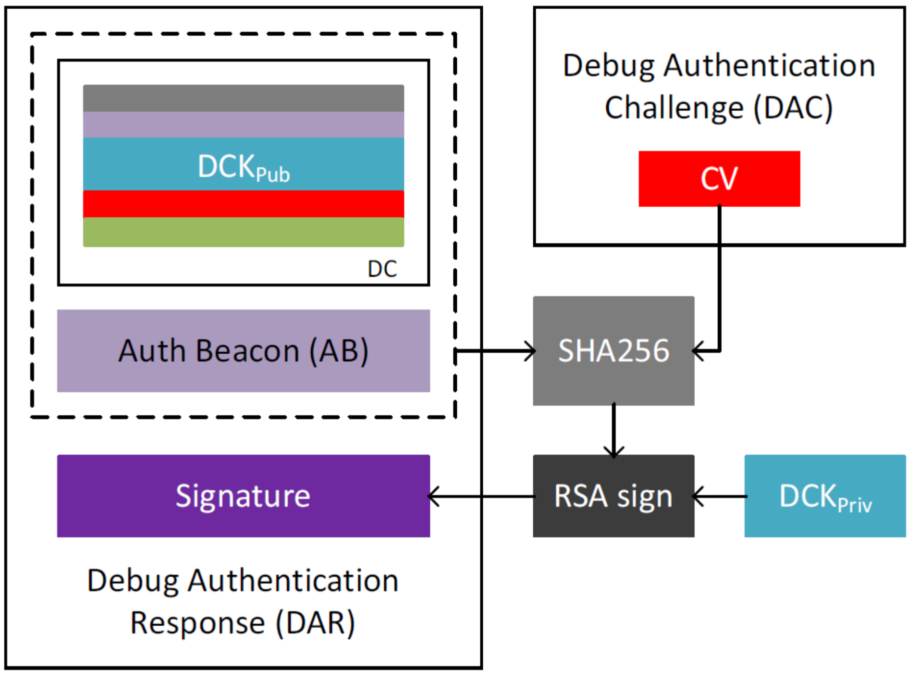

DCKPub
DC
Auth Beacon (AB)
Signature
Debug Authentication Response (DAR)
Debug Authentication Challenge (DAC)
CV
SHA256
RSA sign
DCKPriv

Figure 9. Debug Authentication Response (DAR) fields

The debugging user creates 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 certificate 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. Figure 10 shows the structure of this DC. This example command generates the key pair for the DC.

```
nxpkeygen -p 1.0 genkey keys\dck_rsa_2048.pem
```

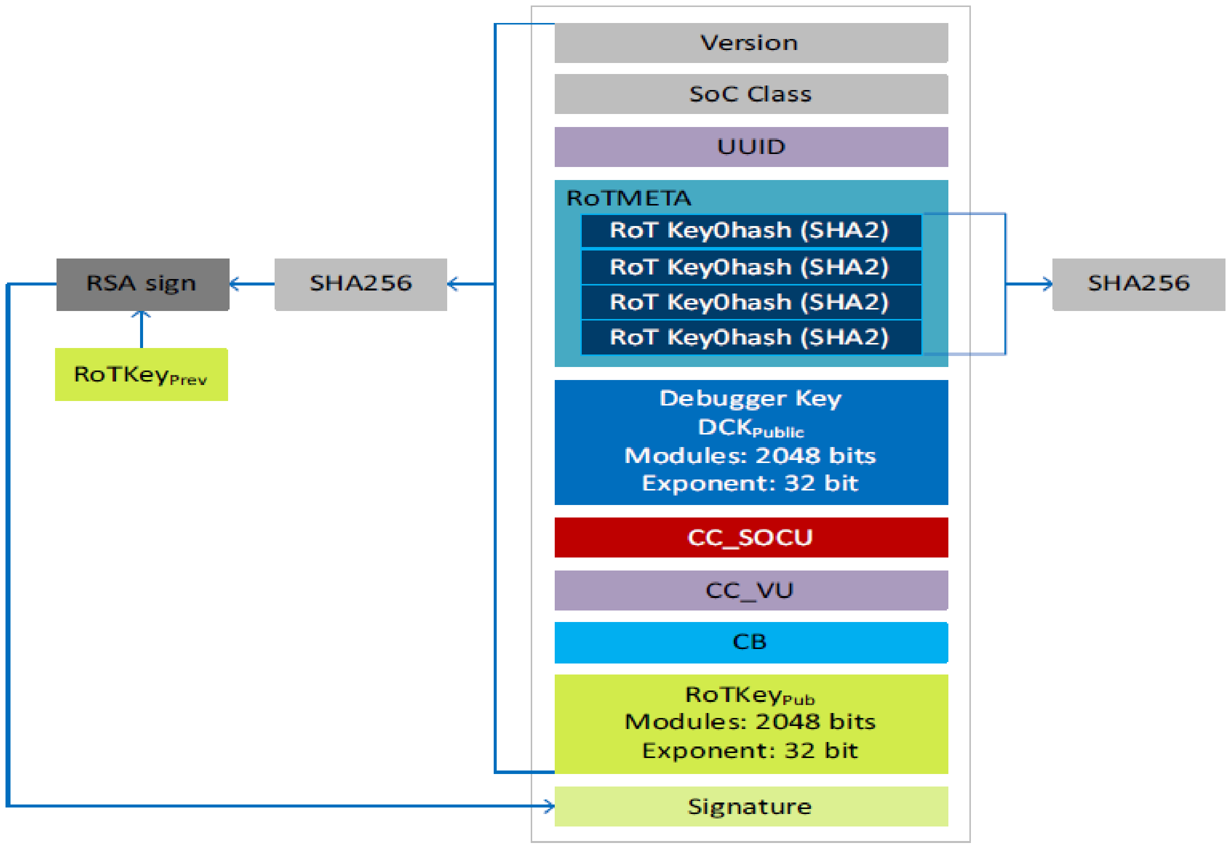
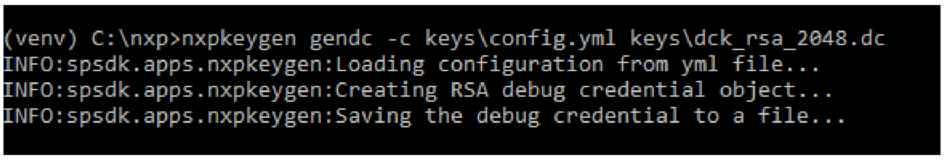


Figure 10. Debug Credential certificate (DC) fields

The RoT key owner must configure the debug access restrictions and so on in the *config.yaml* file. The ELFTOSB configuration file with the RoT path can be used with parameter **-e**. In that case, “rot_meta” and “rotk” are not needed in the configuration file for NXPKEYGEN. The following are the examples of the GENDC command to generate the DC. Only one of the following options is used:

```
nxpkeygen gendc -c keys\config.yml keys\dck_rsa_2048.dc
```

```
nxpkeygen gendc -c keys\config.yml -e keys\config_elftosb.json keys
\dck_rsa_2048.dc
```



```
(venv) C:\nxp>nxpkeygen gendc -c keys\config.yml keys\dck_rsa_2048.dc
INFO:spsdk.apps.nxpkeygen:Loading configuration from yml file...
INFO:spsdk.apps.nxpkeygen:Creating RSA debug credential object...
INFO:spsdk.apps.nxpkeygen:Saving the debug credential to a file...
```

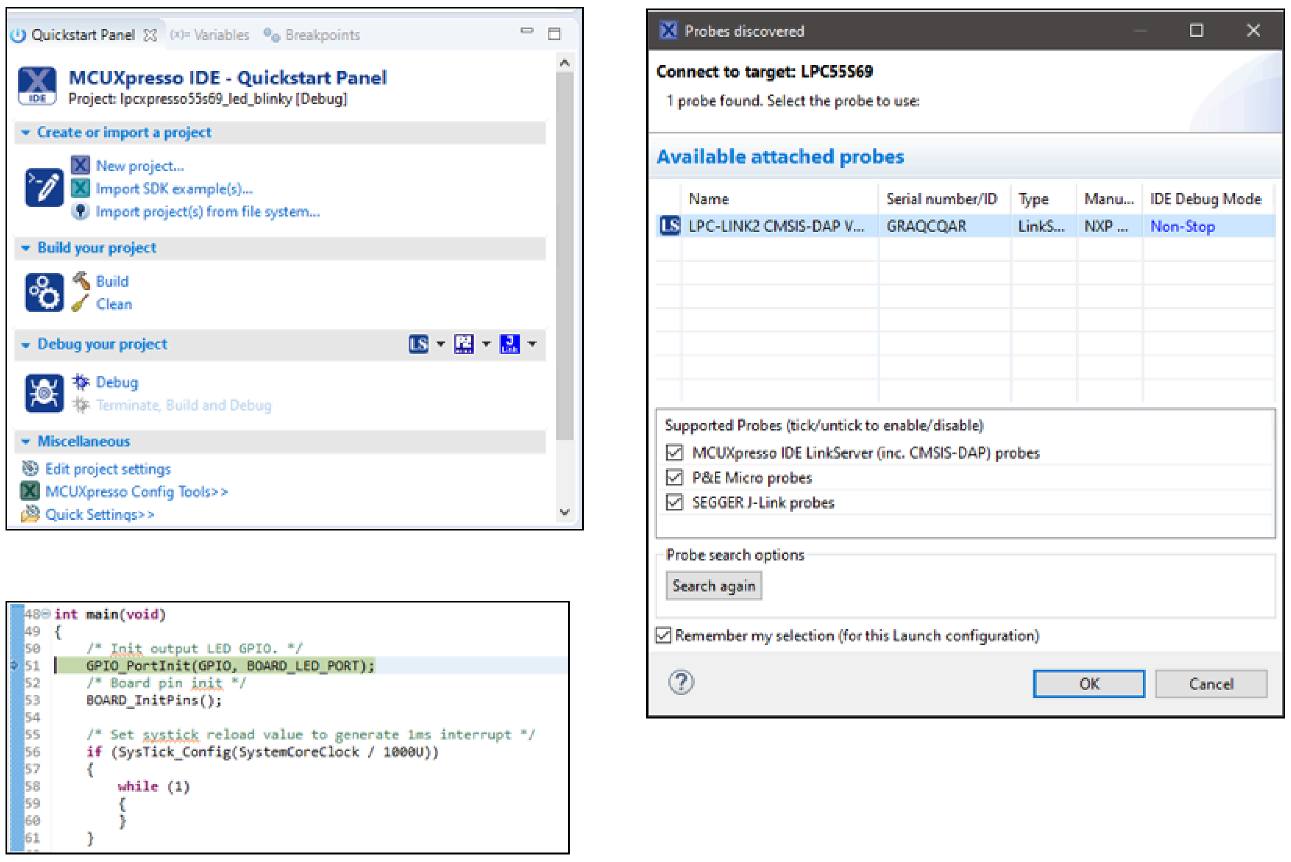
Figure 11. Generate DC by NXPKEYGEN

After this, the DC file is generated by the NXPKEYGEN tool. It is possible to use this debug credential certificate for authenticating through the debugger in the future.

3.2.5 Loading configuration into device

You must have a correct configuration generated in the previous chapter and this binary configuration must be loaded into the device.

By default, all devices have the debug probes enabled, so there is a possibility to use the IDE to load and debug the project. For this example, the MCUXpresso IDE is used. You can debug and connect to a device. Below, the “led_blinky” SDK example is used.



```
int main(void)
{
    /* Init output LED GPIO. */
    GPIO_PortInit(GPIO, BOARD_LED_PORT);
    /* Board pin init */
    BOARD_InitPins();

    /* Set systick reload value to generate 1ms interrupt */
    if (SysTick_Config(SystemCoreClock / 1000U))
    {
        while (1)
        {
        }
    }
```

Connect to target: LPC55S69

1 probe found. Select the probe to use:

Available attached probes

Name	Serial number/ID	Type	Manu...	IDE Debug Mode
LPC-LINK2 CMSIS-DAP V...	GRAQCQAR	LinkS...	NXP ...	Non-Stop

Supported Probes (tick/untick to enable/disable)

- ✓ MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
- ✓ P&E Micro probes
- ✓ SEGGER J-Link probes

Probe search options

✓ Remember my selection (for this Launch configuration)

Figure 12. Debugger connection from MCUXpresso

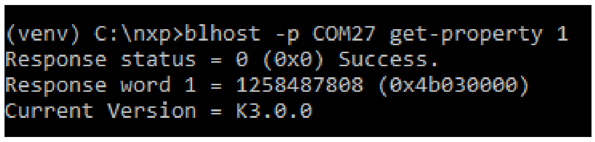
The “led_blinky” example project is loaded in the board. After the reset, you can see the LED flashing.

To load the device configuration, it is recommended to use the ISP mode and configuration through UART.

It is possible to use BLHOST, which is a part of SPSDK. For the ISP communication, the device must enter the ISP mode (hold the ISP button and reset the device using the reset pin) and test the ISP communication.

```
blhost -p COMx get-property 1
```

This command confirms that BLHOST is communicating with the boot ROM and it should receive the following response from the device.



```
(venv) C:\nxp>blhost -p COM27 get-property 1
Response status = 0 (0x0) Success.
Response word 1 = 1258487808 (0x4b030000)
Current Version = K3.0.0
```

Figure 13. Device response

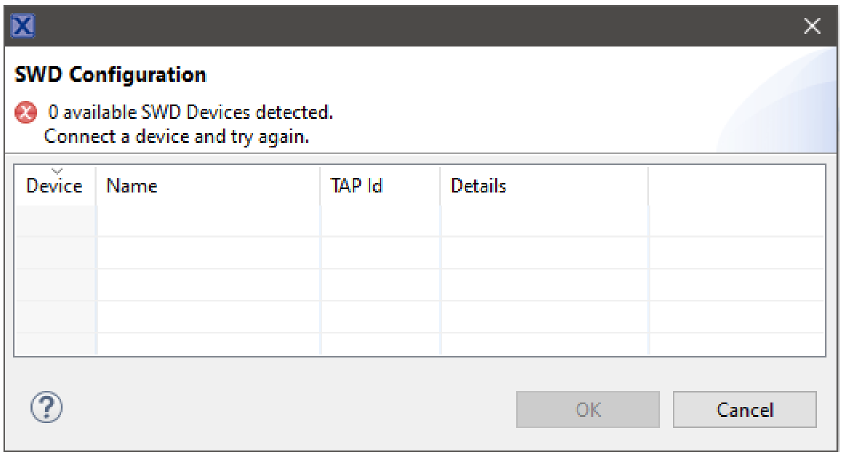
When the ISP communication is tested, it is possible to load the configuration into the device. It is expected that the *cmpa.bin* and *cfpa.bin* files are already created in the previous steps with the correct configuration.

The example of loading the CMPA/CFPA configuration into LPC55S6x/LPC55S2x is shown below. This device has the PFR start address at 0x9de00. LPC55S1x/LPC55S0x have the PFR at address 0x3de00.

```
blhost -p COMx write-memory 0x9e400 cmpa.bin
```

```
blhost -p COMx write-memory 0x9de00 cfpa.bin
```

When you reset the device by the power-on reset or by the pin reset, the configuration specified in the *cmpa.bin* and *cfpa.bin* files is loaded. When enabling the DAP for debug probes, you will see that the IDE cannot find an SWD device available for a debug connection after the reset.


SWD Configuration

0 available SWD Devices detected.
Connect a device and try again.

Device	Name	TAP Id	Details

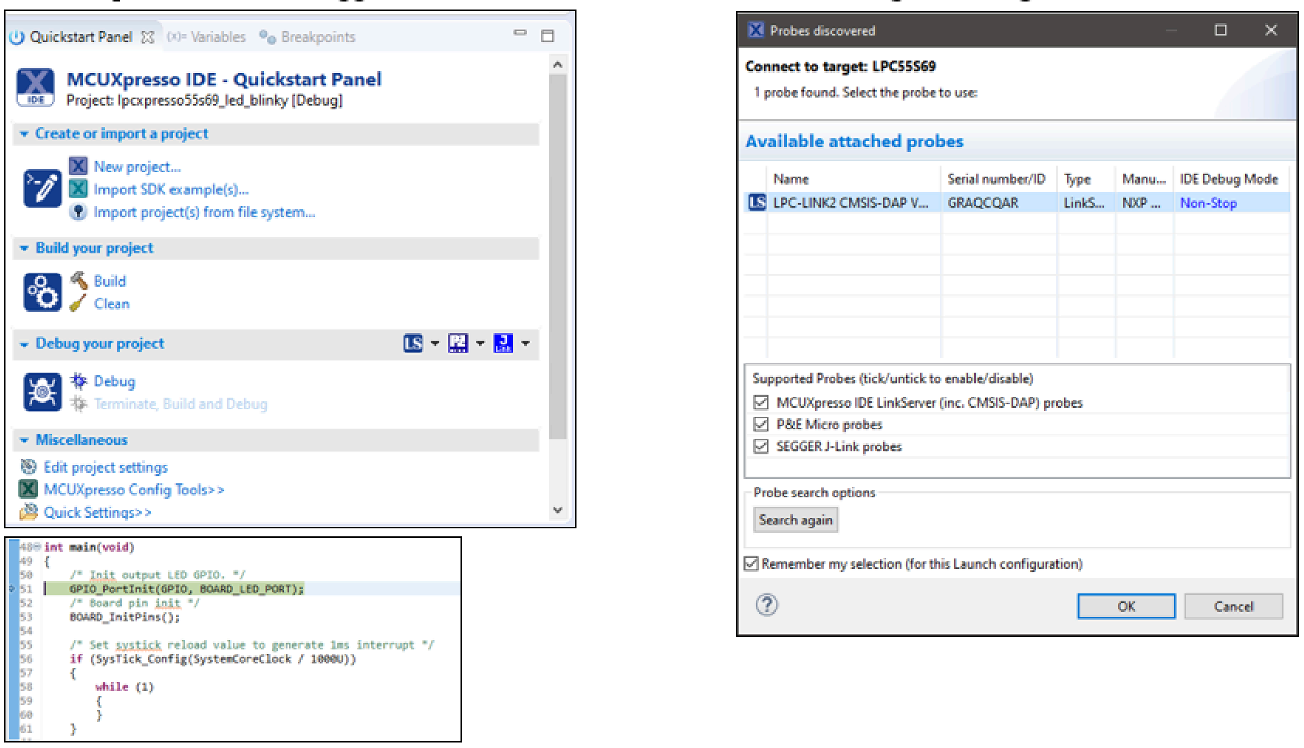
Figure 14. Unavailable debug port

3.2.6 Using debug authentication tool to open debug port

In this step, it is possible to test the debug authentication tool, which is a part of SPSDK. This tool communicates with the debug mailbox through a debugger probe and attempts to authenticate to enable debug access. Interface selection ***-i*** is right now supporting PYOCD for CMSIS-DAP and JLINK for the J-Link debugger. Some processing info printed from the tool is in the console. When the debug authentication is successful, then it is again possible to connect the debugger to the core. For example, the “led_blinky” project with the MCUXpresso IDE. The **-b** parameter is an authentication beacon, which is handled through the ROM together with “cc_beacon” from the DC file into the “SYSCON->DEBUG_AUTH_BEACON” register, and it can be read by the application running in the device. The **-c** parameter is the DC file, which is signed by the owner of the RoT key and it is authenticated to open debug ports, as configured in the DC file. The **-k** parameter is the private key generated initially before the DC file is signed. It is the approach to achieve the OEM (RoT key owner) and the field-technician debug credential security.

```
nxpdebugmbox -p 1.0 auth -b 0 -c keys\dck_rsa_2048.dc -k keys\dck_rsa_2048.pem
```

Debug ports are unlocked on the device after the debug authentication until the power-on reset or pin reset are pressed. The debugger can make a software reset without losing the debug access.



```
int main(void)
{
    /* Init output LED GPIO. */
    GPIO_PortInit(GPIO, BOARD_LED_PORT);
    /* Board pin init */
    BOARD_InitPins();

    /* Set systick reload value to generate 1ms interrupt */
    if (SysTick_Config(SystemCoreClock / 1000U))
    {
        while (1)
        {
        }
    }
```

Connect to target: LPC55S69

1 probe found. Select the probe to use:

Available attached probes

Name	Serial number/ID	Type	Manu...	IDE Debug Mode
LPC-LINK2 CMSIS-DAP V...	GRAQCQAR	LinkS...	NXP ...	Non-Stop

Supported Probes (tick/untick to enable/disable)

- [✓] MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
- [✓] P&E Micro probes
- [✓] SEGGER J-Link probes

Probe search options

[✓] Remember my selection (for this Launch configuration)

Figure 15. Debug port available after debug authentication

Note: *Debug authentication can be affected by the ROM errata: ROM. 10 in LPC55S6x devices and ROM.3 in LPC55S1x and LPC55S0x devices. See the corresponding device errata sheet for details.*

4 Conclusion

NXP debug authentication is one of the key features that enable customer security during the complete lifecycle of a product. In this application note, an example configuration and usage for the LPC55S69 devices is described. The principles and tools are the same for other LPC55xx devices.

5 Reference

1. *LPC55S6x/LPC55S2x/LPC552x User Manual* (document UM11126)
2. *LPC55S6x Data Sheet* (document LPC55S6x)
3. *MCUXpresso IDE User Guide* (document MCUXPRESSO-UG)

6 Note about the source code in the document

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

7 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN13037 v.3.0	12 December 2023	The document is updated to correspond to the latest guidelines, Section 1 and Section 3.2.6 are updated.
AN13037 v.2.0	19 September 2023	The document is updated to correspond to the latest guidelines, Section 1 is updated.
AN13037 v.1.0	06 October 2021	SPSDK version updates
AN13037 v.0.0	11/2020	Initial version

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

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

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

Trademarks

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

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

Contents

Date of release: 12 December 2023
Document identifier: AN13037