# AN12419

## Secure JTAG for i.MX RT10xx

Rev. 3 — 18 December 2023 Application note

**Document information**

| Information | Content |
|---|---|
| Keywords | Secure JTAG, i.MX RT10xx, MCU |
| Abstract | This document describes how the Secure JTAG on the i.MX RT10xx MCU family can be used. |

## 1 Introduction

This document describes how the Secure JTAG on the i.MX RT10xx MCU family can be used.

The i.MX RT series System JTAG Controller (SJC) provides a possibility to regulate the JTAG access. The three JTAG security modes are available in the i.MX RT series:

- No Debug mode—Maximum security is provided in this mode. All security-sensitive JTAG features are permanently blocked, preventing any debug.
- Secure JTAG mode—High security is provided in this mode. A secret key-based challenge/response authentication mechanism is used for JTAG access
- JTAG Enabled mode—Low security is provided in this mode. It is the default mode of operation for the SJC.

Moreover, you can also fully disable the SJC functionality. For configuration of these JTAG modes, One Time Programmable (OTP) eFuses are used and burned after packaging. The fuse burning process is irreversible. It is impossible to revert the fuse back to the unburned state. To explain, Secure JTAG mode is used in this document. The aim is to allow return/field testing. Authorized reactivation of the JTAG port is allowed in this mode. On the HW side, JTAG signals must be routed out and accessible in the application.

***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 Tool and SPSDK) instead of following the steps presented here. For any questions, contact your local support.*

## 2 i.MX RT10xx Secure JTAG support

JTAG access is limited in the Secure JTAG mode by using a challenge/response-based authentication. Any access to the JTAG port is internally checked. Only the devices authorized for debugging (with the right response) can access the JTAG port. Otherwise, JTAG access is denied. The external debugger tools (such as SEGGER J-Link, Lauterbach Trace32, Arm RVDS/DS5) supporting the challenge/response-based authentication mechanism can be used. The secure JTAG mode is typically enabled in the factory manufacturing and not used during the development.

### 2.1 How to put the chip in Secure JTAG mode

The Secure JTAG feature is only available in the **SJC** mode, and can be selected by the JTAG_MOD input pin (GPIO_AD_B0_08). To enable the SJC mode, the pin must be connected to log.1, which means that the Secure JTAG is unavailable in the CM7 DAP mode.

**Table 1. JTAG_MOD pin settings**

| Signal | Description | Pad | Mode | Direction |
|---|---|---|---|---|
| JTAG_MOD | SJC mode selection. This pin is sampled at TRST reset to determine two possible modes for the TAP connection configuration. | GPIO_AD_B0_08 | ALT0 | IO |

***Note:*** *Due to known HW conflicts on i.MX RT10xx EVKBs, small PCB modifications may be required for JTAG_TDO/TDI signals. Otherwise, the communication in JTAG mode is not possible. Refer to the EVKB schematic for changes needed.*

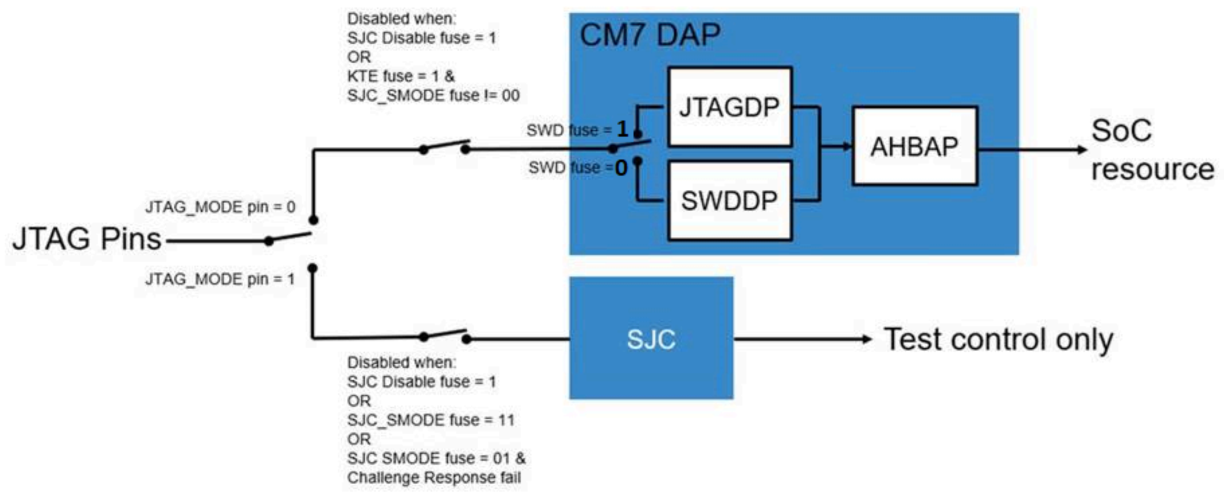


**Figure 1. JTAG_MODE pin configuration**

## 2.2 i.MX RT SJC Security Modes

The i.MX RT10xx System JTAG Controller (SJC) supports three different security modes. JTAG enabled is the default mode of operation for the SJC. The user can select the Secure JTAG mode by programing a value 0x1 to the eFuse labeled JTAG_SMODE, described in Table 2. The eFuse has the default value 0x0, which means that the JTAG controller is unsecured by default. Further details on eFuses are available in the Fusemap and On-Chip OTP Controller (OCOTP_CTRL) chapters in the appropriate SRM_RT10xx Security Reference Manual for the i.MX RT1050 available at www.nxp.com upon request.

To lock and prevent further modification to the JTAG_SMODE eFuse, program the BOOT_CFG_LOCK eFuses in addition to programming the JTAG_SMODE eFuse.

***Note:*** *Programming these fuses disables access to functions and JTAG Security Mode fuse bits. Ensure that it is programmed last, once the final fuse configuration has been decided. At a minimum, setting the fuse to Override Protect (OP) mode is recommended.*

**Table 2. eFuses associated with the Secure JTAG feature**

| Addr[bits] | Fuse Name | Fuse Function | Settings | Locked By |
|---|---|---|---|---|
| 0x460[27] | JTAG_HEO | JTAG HAB Enable Override. Disallows HAB JTAG enabling. The HAB may normally enable JTAG debugging with the HAB_JDE-bit in the OCOTP SCS register. The JTAG_HEO-bit can override this behavior | 0 - HAB may enable JTAG debug access<br>1 - HAB JTAG enable is overridden (HAB may not enable JTAG debug access) | BOOT_CFG_LOCK |
| 0x460[26] | KTE | Kill Trace Enable. Enables tracing | 0 - Bus tracing is allowed | BOOT_CFG_LOCK |

**Table 2. eFuses associated with the Secure JTAG feature***...continued*

| Addr[bits] | Fuse Name | Fuse Function | Settings | Locked By |
|---|---|---|---|---|
| | | capability on ETM, and other modules | 1 - Bus tracing is allowed in case the security state as defined by Secure JTAG allows it (for example, JTAG_ ENABLE or NO_DEBUG) | |
| 0x460[23:22] | JTAG_SMODE[1:0] | JTAG Security Mode. Controls the security mode of the JTAG debug interface | 00 - JTAG enable mode (Default)<br>01 - Secure JTAG mode<br>11 - No debug mode | BOOT_CFG_LOCK |
| 0x460[20] | SJC_DISABLE | Additional JTAG mode with the highest level of JTAG protection, thereby overriding the JTAG_ SMODE eFuses. In this mode, all JTAG features are disabled, including Secure JTAG and Boundary Scan | 0 - JTAG is enabled<br>1 - JTAG is disabled | BOOT_CFG_LOCK |
| 0x460[19] | DAP_SJC_SWD_SEL | Control DAP works in JTAG or SWD mode | 0 - DAP works in SWD mode<br>1 - DAP works in JTAG mode | BOOT_CFG_LOCK |
| 0x400[3:2] | BOOT_CFG_LOCK[1:0] | Perform lock protection on BOOT-related fuses. This fuse locks numerous functions including JTAG_SMODE | 00 - Unlock<br>1x - Override Protect (OP)<br>x1 - Write Protect (WP)<br>11 - Both OP and WP | N/A |
| 0x400[6] | SJC_RESP_LOCK | SJC response lock | 0 - Unlock<br>1 - Lock (WP,OP,RP, sense) | |
| 0x600 | SJC_RESP[55:0] | Response reference value for the secure JTAG controller | | SJC_RESP_LOCK (locks also for read and explicit sense) |

***Note:***

*The level of security cannot be reduced but only increased. Since debug modes are controlled by OTP (Hardware fuses), bits can only be blown once.*

*For example, the following mode changes are possible:*

*– “JTAG Enabled” to “Secure JTAG”*

*– “Secure JTAG” to “No debug”*

## 2.3 Secure JTAG eFuses

The challenge/response mechanism used to authenticate the JTAG access uses a challenge value and the associated secret response key. The keys are stored in eFuses inside the IC. The i.MX RT1050 series eFuses used to store the challenge value and the secret response key are listed below:

- The challenge value is the “Device Unique ID” that is programmed into the eFuses. This Device ID is unique for each IC and can be read from the OCOTP registers HW_ OCOTP_CFG0 and HW_ OCOTP_CFG1. The eFuses are programmed during manufacturing.
- The user program the secret response key (56 bits) into the eFuses marked SJC_RESP.
- The KTE fuse must be programmed to have JTAG secure mode work. Each POR only allows one-time response code input. If the response code is incorrect, the chip must have a POR reset before trying another response code. POR clears sensitive data except the SNVS domain.

After programming the secret response key, the user must disable the ability of software running on the Arm core to read or overwrite the response key. This is done by programming a 0x1 to the associated lock eFuse HW_OCOTP_LOCK_SJC_RESP.

The definition of the response value is left to the user. The Arm core cannot read the value once the response fuse field is provisioned and locked.

## 2.4 SW Enabled JTAG

The Secure JTAG authentication may be bypassed in SW by writing '1' to the HAB_JDE (HAB JTAG DEBUG ENABLE) bit in the e-fuse controller module. By this JTAG is opened, regardless of its security mode. The S/W JTAG enable allows JTAG enabling without activating the challenge-response mechanism.

The platform initialization software must set the LOCK bit for the JDE bit before transferring control to the application code to ensure that only the trusted SW can set the JDE bit.

The JTAG SW enable does not allow debug in case of boot or memory fault as it requires a reset before entering debug.

The JTAG_JDE bit SW enable backdoor access can be permanently disabled by burning the JTAG_HEO fuse.

***Note:*** *The S/W enabled JTAG feature reduces the overall security level of the system as it relies on S/W protections. If this feature is not required, it is recommended to burn the JTAG_HEO e-fuse that disables this feature.*

### 2.4.1 JDE bit control in HAB (High Assurance Boot)

The HAB_JDE can be set to ‘1’ by ROM boot SW after unlocking by the Authenticate CSF command.

Before generating of the signed program image, the user must edit the UNLOCK section in the .sb file and provide the device-specific UID in the proper format as a sequence of 8 bytes, see the below example for UID = 0x63e1841b440b81d2:

```
section (SEC_UNLOCK;
Unlock_Engine = "OCOTP",
Unlock_features = "JTAG, SCS, SRK REVOKE",
Unlock_UID = "0xe1, 0x63, 0x1b, 0x84, 0x0b, 0x44, 0xd2, 0x81"
)
```

For more information about the HAB_JDE SW control by platform initialization SW in HAB (High Assurance Boot) refer to section **5.2.13 Unlock (HAB only)** in HAB Code-Signing Tool User’s Guide.

## 2.5 Secure JTAG debug authentication protocol

When the SJC is in Secure Debug mode, the authentication process is as follows:

1. JTAG shifts the challenge key through the Test Data Output (TDO) chain.
2. On the host side, the debug tool takes the challenge key as an input and generates the expected response key.
3. The associated response key is shifted back through the Test Data Input (TDI) chain.
4. The SJC compares the expected internal fused response key with the one shifted in and enables the JTAG access only if it matches.

***Note:***

*Any device reset after JTAG access authorization shifts the JTAG controller back to its locked state.*

Figure 2 shows how the challenge/response mechanism works with the JTAG tools.

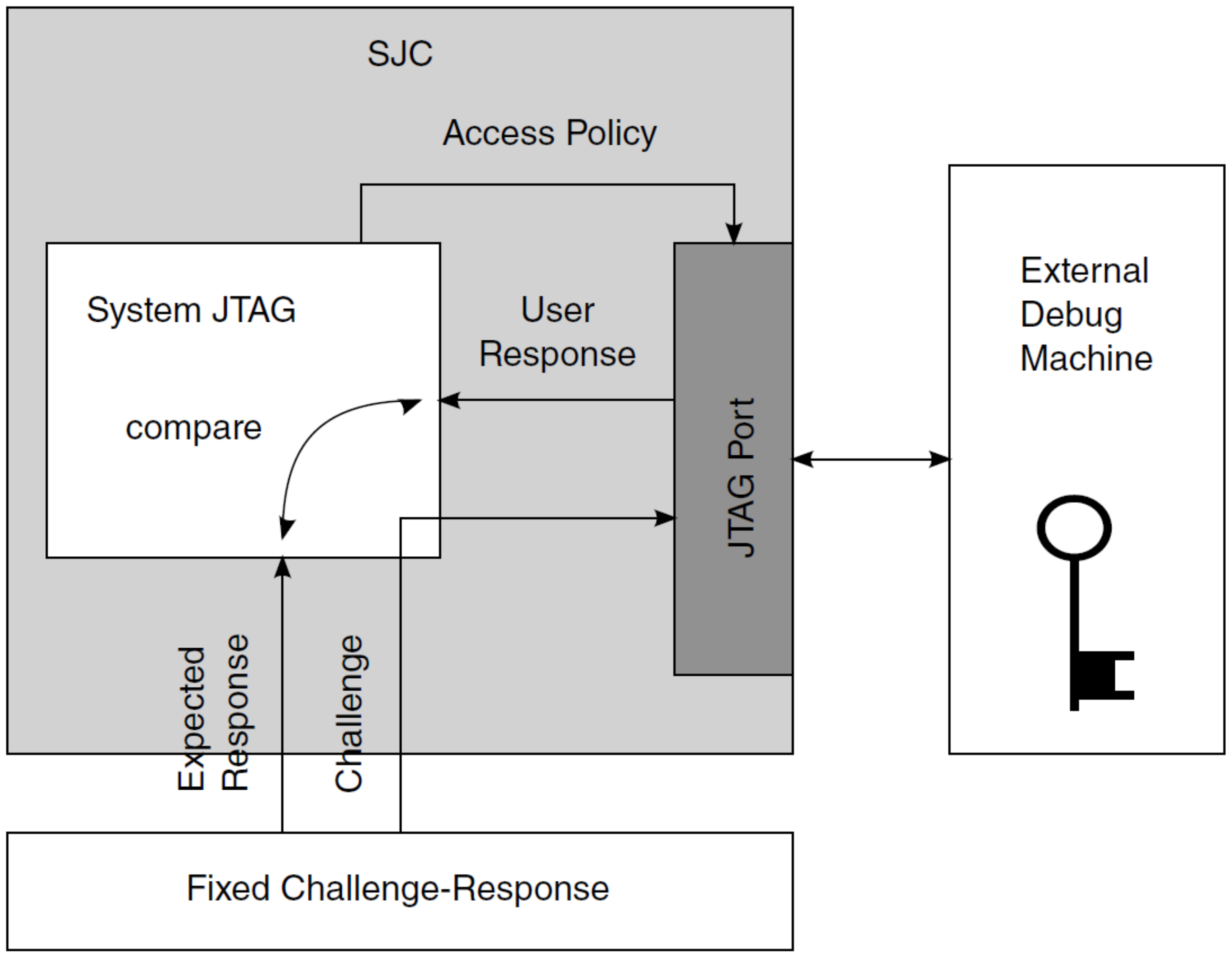


**Figure 2. Secure JTAG operation**

The JTAG debug tool passes the retrieved challenge key to the user's application and gets the associated response key in return. The management of the challenge/response pairs is user-dependent and not handled by NXP or the debug tool vendors. Key management is discussed further in Section 3.

### 2.6 SJC disable fuse

In addition to the various JTAG security modes implemented internally in the SJC, there is an option to disable the SJC functionality with the SJC_DISABLE eFuse. This eFuse creates an additional JTAG mode, JTAG Disabled with the highest level of JTAG protection, overriding the JTAG_SMODE eFuses. In this mode all JTAG features are disabled, including Secure JTAG and Boundary Scan; users must ensure that this fuse is not blown if they wish to use the Secure JTAG functionality.

## 3 Secret response key approaches

For every challenge value ("Device Unique ID" in i.MX RT10xx) that is retrieved with a JTAG instruction, there is an associated secret response key known only by the user. The JTAG tool vendor only handles the JTAG mechanism used by this authentication process, and does not know the secret response key value programmed into the eFuses. It is left to the user to determine the level of protection that is put in place.

The following are policies for secret response key management by the user application.

1. Identical Response Keys —The same response key is used for each chip. The user can choose a response key that is fused in all chips. This is the simplest, but least sophisticated usage from a security point of view. If an unauthorized user gains access to the fused response key, all the products fused with this response key can be accessed through the JTAG port.
2. Database of Unique Response Keys—The user maintains a database of all generated response keys. The user application can look up the table based on the challenge value. It is possible to implement a secure server holding the challenge/response pairs authenticating the user but this requires an independent implementation effort. The challenge values for all ICs must be read and a database of matching challenge response pairs must be built. Storing and managing numerous response keys is not trivial, but advantageous from a security standpoint, as it does not rely on any breakable algorithms.
3. Algorithmically Generated Response Keys—Response keys are generated based on an algorithm. With this method, there is no large database to manage. For instance, the challenge value can be used by the algorithm to generate a response key. This response key is programmed into SJC_RESP eFuses. Then, every time the challenge value is retrieved through JTAG, it can be processed by the user application and used to generate the expected response key for the JTAG debug tools. Once the algorithm is exposed or reverse-engineered, this method is no longer secure.

***Note:***

*NXP does not provide secure response key management or key generation services; these topics are not within the scope of this document.*

### 3.1 Programming Secure JTAG eFuses using the NXP tool

To program the relevant eFuses needed for Secure JTAG on the chip, follow the steps below. Information on the On-Chip OTP Controller (OCOTP_CTRL) and the Fusemap can be found in the appropriate i.MX RT10xx series reference manual available at www.nxp.com. The NXP MCU Boot Utility is used in the following steps to program eFuses.

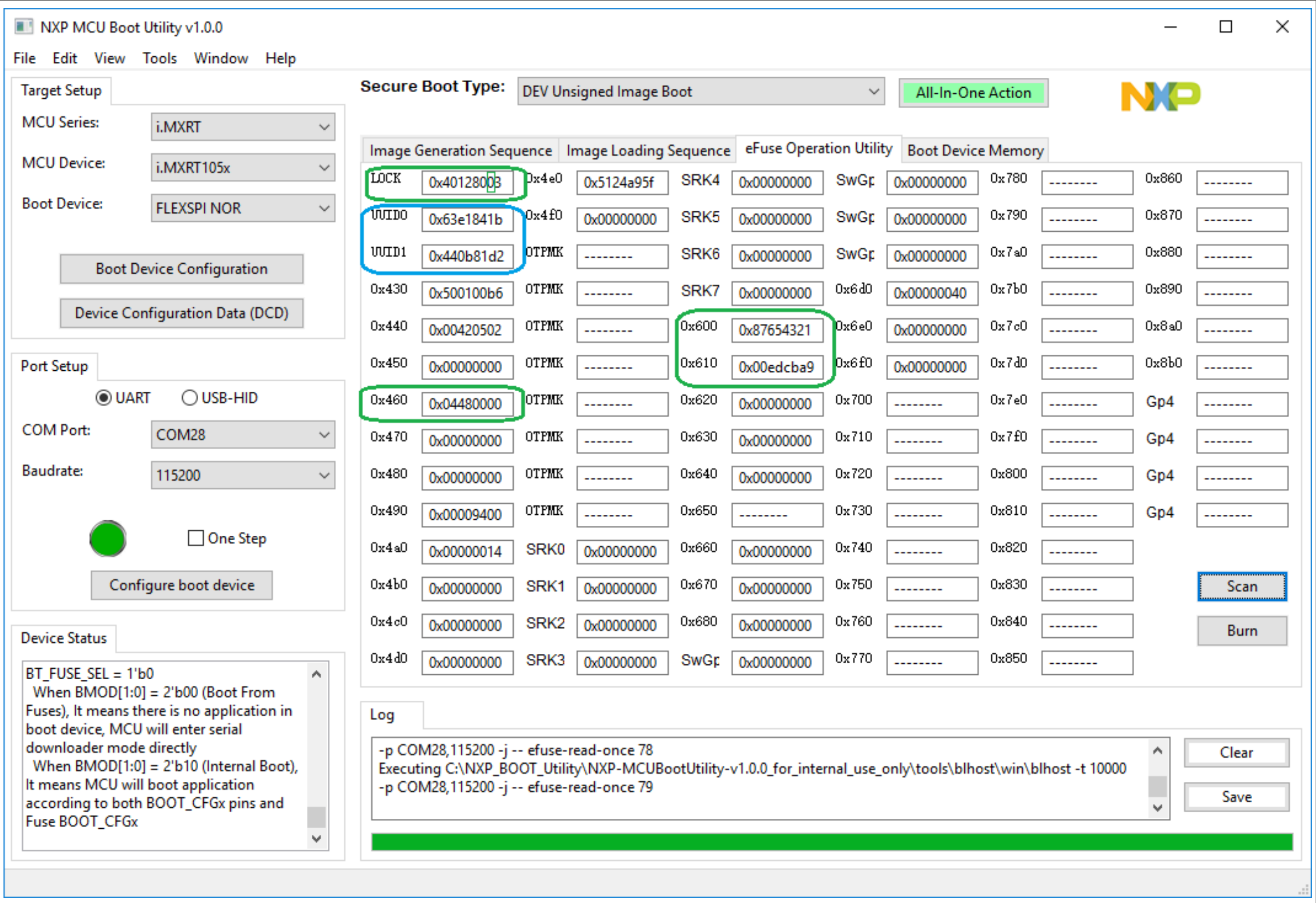
1. Download the latest NXP MCU Boot Utility from:http://www.nxp.com
2. Program the values below to the eFuses needed for secure JTAG:
   - Read and back-up the 64-bit "Challenge" value stored in the eFuse UUID[1,0], location (0x420, 0x410). See Figure 3.
   - Program a 56-bit (7 Bytes) secret response key in the eFuse SJC_RESP, location (0x610, 0x600). In the example below, value "*0xedcba987654321*" is programmed.
     ***Note:*** *In Figure 3 , MSB is stored on the higher address, while the most-left byte is 0x00 and it is ignored. Define the own response key and keep the key backup for further usage.*

- Program 0x1 in the eFuse DAP_SJC_SWD_SEL to switch the DAP to the JTAG mode.
- Program 0x1 in the eFuse JTAG_SMODE to switch the SJC to Secure JTAG mode.
- Program 0x1 in the eFuse KTE_FUSE
- Finally, the user must program 0x1 in the eFuse SJC_RESP _LOCK to disable read/write access of the secret response key. After this operation, the secret response field “SJC_RESP” becomes “invisible” in the fuse map. See Figure 3 and Figure 4 .

The following figures demonstrate how to use the NXP MCU Boot Utility to program the above eFuses. The eFuse operation utility is a part of the tool, which can read and write the eFuse map registers. Be careful when doing writing (Burn) operations, as it is irreversible and may lock some features or the device completely.

To have the Secure JTAG enabled, follow the steps mentioned above in Section 3.1 and refer to Table 2, Figure 3 and Figure 4 for more details about the appropriate eFuse bits.

The example does not program the BOOT_CFG_LOCK[1:0] and eFuses to prevent further modifications of the JTAG_SMODE eFuse. As programming these lock eFuses, disables the access to functions in addition to the JTAG mode bits. Therefore, it must be performed once the final configuration has been decided.



Target Setup

| | |
|---|---|
| MCU Series: | i.MXRT |
| MCU Device: | i.MXRT105x |
| Boot Device: | FLEXSPI NOR |

Port Setup

(● UART) ○ USB-HID

| | |
|---|---|
| COM Port: | COM28 |
| Baudrate: | 115200 |

☐ One Step

Device Status

```
BT_FUSE_SEL = 1'b0
  When BMOD[1:0] = 2'b00 (Boot From
Fuses), It means there is no application in
boot device, MCU will enter serial
downloader mode directly
  When BMOD[1:0] = 2'b10 (Internal Boot),
It means MCU will boot application
according to both BOOT_CFGx pins and
Fuse BOOT_CFGx
```

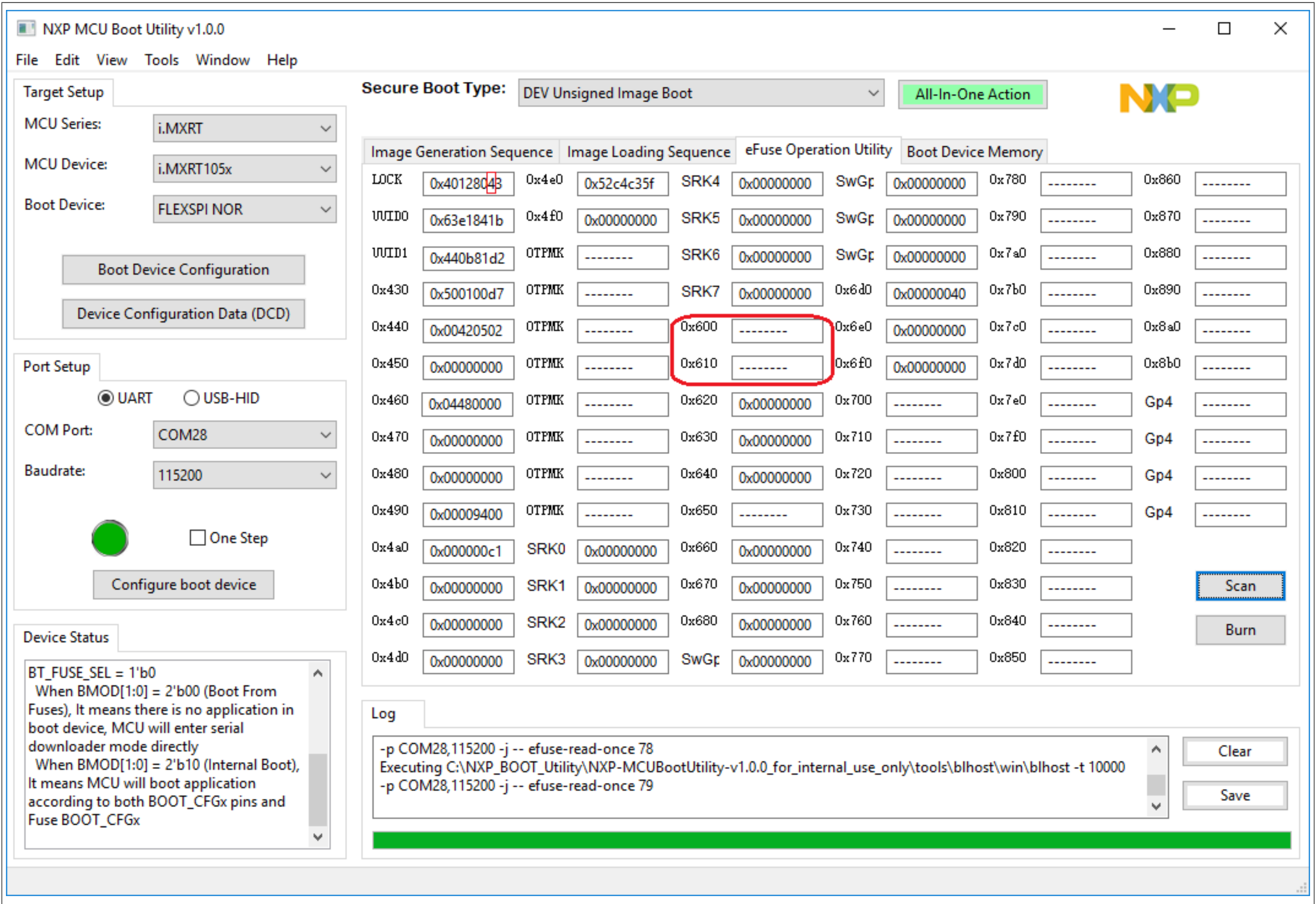
Secure Boot Type: DEV Unsigned Image Boot

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOCK | 0x40128003 | 0x4e0 | 0x5124a95f | SRK4 | 0x00000000 | SwGp | 0x00000000 | 0x780 | -------- | 0x860 | -------- |
| UUID0 | 0x63e1841b | 0x4f0 | 0x00000000 | SRK5 | 0x00000000 | SwGp | 0x00000000 | 0x790 | -------- | 0x870 | -------- |
| UUID1 | 0x440b81d2 | OTPMK | -------- | SRK6 | 0x00000000 | SwGp | 0x00000000 | 0x7a0 | -------- | 0x880 | -------- |
| 0x430 | 0x500100b6 | OTPMK | -------- | SRK7 | 0x00000000 | 0x6d0 | 0x00000040 | 0x7b0 | -------- | 0x890 | -------- |
| 0x440 | 0x00420502 | OTPMK | -------- | 0x600 | 0x87654321 | 0x6e0 | 0x00000000 | 0x7c0 | -------- | 0x8a0 | -------- |
| 0x450 | 0x00000000 | OTPMK | -------- | 0x610 | 0x00edcba9 | 0x6f0 | 0x00000000 | 0x7d0 | -------- | 0x8b0 | -------- |
| 0x460 | 0x04480000 | OTPMK | -------- | 0x620 | 0x00000000 | 0x700 | -------- | 0x7e0 | -------- | Gp4 | -------- |
| 0x470 | 0x00000000 | OTPMK | -------- | 0x630 | 0x00000000 | 0x710 | -------- | 0x7f0 | -------- | Gp4 | -------- |
| 0x480 | 0x00000000 | OTPMK | -------- | 0x640 | 0x00000000 | 0x720 | -------- | 0x800 | -------- | Gp4 | -------- |
| 0x490 | 0x00009400 | OTPMK | -------- | 0x650 | -------- | 0x730 | -------- | 0x810 | -------- | Gp4 | -------- |
| 0x4a0 | 0x00000014 | SRK0 | 0x00000000 | 0x660 | 0x00000000 | 0x740 | -------- | 0x820 | -------- | | |
| 0x4b0 | 0x00000000 | SRK1 | 0x00000000 | 0x670 | 0x00000000 | 0x750 | -------- | 0x830 | -------- | | |
| 0x4c0 | 0x00000000 | SRK2 | 0x00000000 | 0x680 | 0x00000000 | 0x760 | -------- | 0x840 | -------- | | |
| 0x4d0 | 0x00000000 | SRK3 | 0x00000000 | SwGp | 0x00000000 | 0x770 | -------- | 0x850 | -------- | | |

Log

```
-p COM28,115200 -j -- efuse-read-once 78
Executing C:\NXP_BOOT_Utility\NXP-MCUBootUtility-v1.0.0_for_internal_use_only\tools\blhost\win\blhost -t 10000
-p COM28,115200 -j -- efuse-read-once 79
```

**Figure 3. eFuse Operation Utility - Secure Response key configuration**



MCU Series: i.MXRT
MCU Device: i.MXRT105x
Boot Device: FLEXSPI NOR

(● UART) ○ USB-HID
COM Port: COM28
Baudrate: 115200
☐ One Step

```
BT_FUSE_SEL = 1'b0
  When BMOD[1:0] = 2'b00 (Boot From
Fuses), It means there is no application in
boot device, MCU will enter serial
downloader mode directly
  When BMOD[1:0] = 2'b10 (Internal Boot),
It means MCU will boot application
according to both BOOT_CFGx pins and
Fuse BOOT_CFGx
```

Secure Boot Type: DEV Unsigned Image Boot

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LOCK | 0x40128043 | 0x4e0 | 0x52c4c35f | SRK4 | 0x00000000 | SwGp | 0x00000000 | 0x780 | -------- | 0x860 | -------- |
| UUID0 | 0x63e1841b | 0x4f0 | 0x00000000 | SRK5 | 0x00000000 | SwGp | 0x00000000 | 0x790 | -------- | 0x870 | -------- |
| UUID1 | 0x440b81d2 | OTPMK | -------- | SRK6 | 0x00000000 | SwGp | 0x00000000 | 0x7a0 | -------- | 0x880 | -------- |
| 0x430 | 0x500100d7 | OTPMK | -------- | SRK7 | 0x00000000 | 0x6d0 | 0x00000040 | 0x7b0 | -------- | 0x890 | -------- |
| 0x440 | 0x00420502 | OTPMK | -------- | 0x600 | -------- | 0x6e0 | 0x00000000 | 0x7c0 | -------- | 0x8a0 | -------- |
| 0x450 | 0x00000000 | OTPMK | -------- | 0x610 | -------- | 0x6f0 | 0x00000000 | 0x7d0 | -------- | 0x8b0 | -------- |
| 0x460 | 0x04480000 | OTPMK | -------- | 0x620 | 0x00000000 | 0x700 | -------- | 0x7e0 | -------- | Gp4 | -------- |
| 0x470 | 0x00000000 | OTPMK | -------- | 0x630 | 0x00000000 | 0x710 | -------- | 0x7f0 | -------- | Gp4 | -------- |
| 0x480 | 0x00000000 | OTPMK | -------- | 0x640 | 0x00000000 | 0x720 | -------- | 0x800 | -------- | Gp4 | -------- |
| 0x490 | 0x00009400 | OTPMK | -------- | 0x650 | -------- | 0x730 | -------- | 0x810 | -------- | Gp4 | -------- |
| 0x4a0 | 0x000000c1 | SRK0 | 0x00000000 | 0x660 | 0x00000000 | 0x740 | -------- | 0x820 | -------- | | |
| 0x4b0 | 0x00000000 | SRK1 | 0x00000000 | 0x670 | 0x00000000 | 0x750 | -------- | 0x830 | -------- | | |
| 0x4c0 | 0x00000000 | SRK2 | 0x00000000 | 0x680 | 0x00000000 | 0x760 | -------- | 0x840 | -------- | | |
| 0x4d0 | 0x00000000 | SRK3 | 0x00000000 | SwGp | 0x00000000 | 0x770 | -------- | 0x850 | -------- | | |

```
-p COM28,115200 -j -- efuse-read-once 78
Executing C:\NXP_BOOT_Utility\NXP-MCUBootUtility-v1.0.0_for_internal_use_only\tools\blhost\win\blhost -t 10000
-p COM28,115200 -j -- efuse-read-once 79
```

**Figure 4. eFuse Operation Utility – Secure Response key lock**

## 4 Debugging with the Secure JTAG enabled

To use the Secure JTAG feature, the JTAG debugger must support it. The example provided in this section uses the SEGGER J-Link debug tool.

Although the procedures outlined in the example below use an i.MX RT1052 device on the IMXRT1050-EVKB board. They can be applied to other RT10xx devices too. The following steps assume that users have experience working with the debug tools and the NXP MCU Boot Utility.

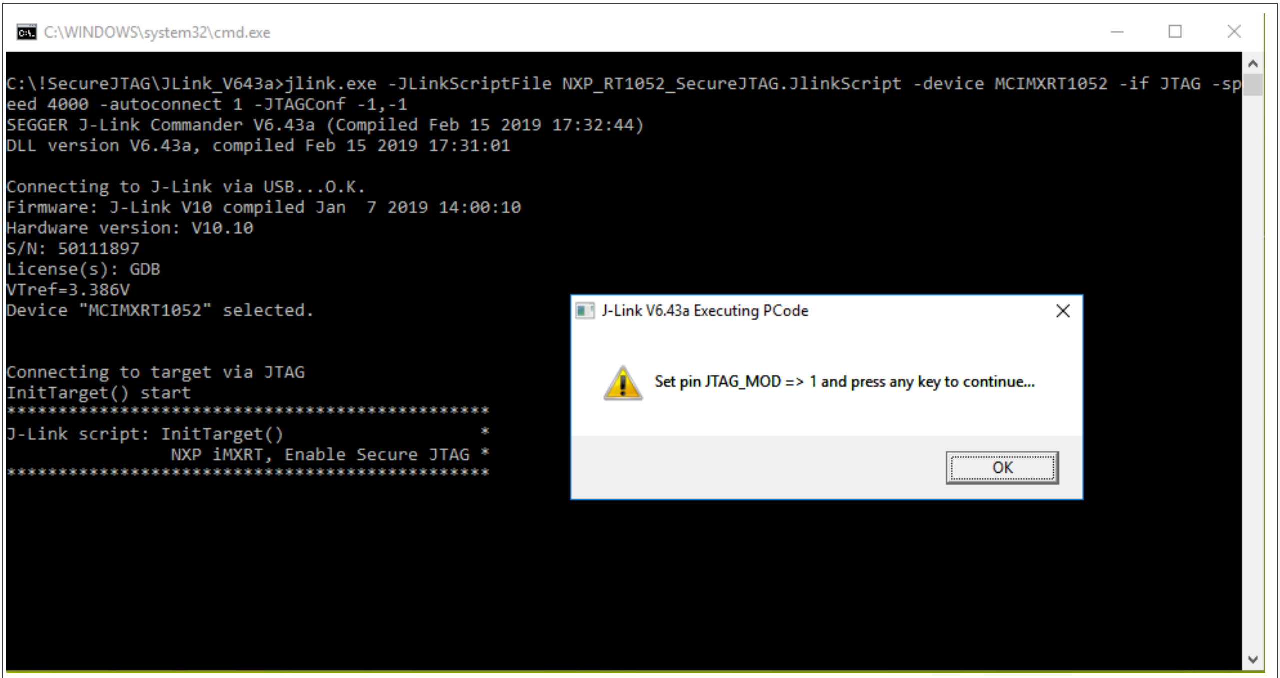
### 4.1 Steps to connect J-Link debugger via Secure JTAG

The following steps connect the SEGGER J-Link debug tool to the i.MX RT10xx when using Secure JTAG:

1. Download the SEGGER J-Link Software and documentation pack:
   https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack
   If you wish to navigate to these scripts from the SEGGER main page for reference, they are located under *"Downloads" - "J-Link / J-Trace" - "J-Link Software and Documentation Pack".*
2. Download and edit the file J-Link script file named "NXP_RT1052_SecureJTAG.JlinkScript". The script file can be received from NXP upon request. In this file, add the secret response key that was programmed into the SJC_RESP eFuse. In the following example, the secret response key is "*0xedcba987654321*", and matches the response key programmed in the eFuses in Section 3.1 .
   *// Secure response stored @ 0x600, 0x610 in eFUSE region (OTP memory)*

*Key0 = 0x87654321;*
*Key1 = 0xedcba9;*

3. Power-up or reset the board with the JTAG_MODE pin (GPIO_AD_B0_08) in log.1. The user must do it manually, since we do not have the signal connected to the debug connector on EVB.
4. Locate the SEGGER SW J-Link installation directory.
5. Run the “jlink.exe” with the mentioned script file as a parameter.
   For instance:
   *jlink.exe -JLinkScriptFile NXP_RT1052_SecureJTAG.JlinkScript -device MCIMXRT1052 -if JTAG -speed 4000 -autoconnect 1 -JTAGConf -1, -1*
   ***Note:*** *The external IDE tool can call “JLinkGDBServer.exe” application with the same script file to unsecure the target.*
   The tool script must read the Challenge value from the eFUSE UUID[1,0] location. And it provides the appropriate Response from the SJC for the authentication match.
   The JTAG_MODE pin must be switched to log.0, see Figure 1 and Figure 6 for details. Do it manually by changing the pin polarity on the board. If the pin is routed to the JTAG connector and the tool supports the control of this signal, the tool can do it automatically,

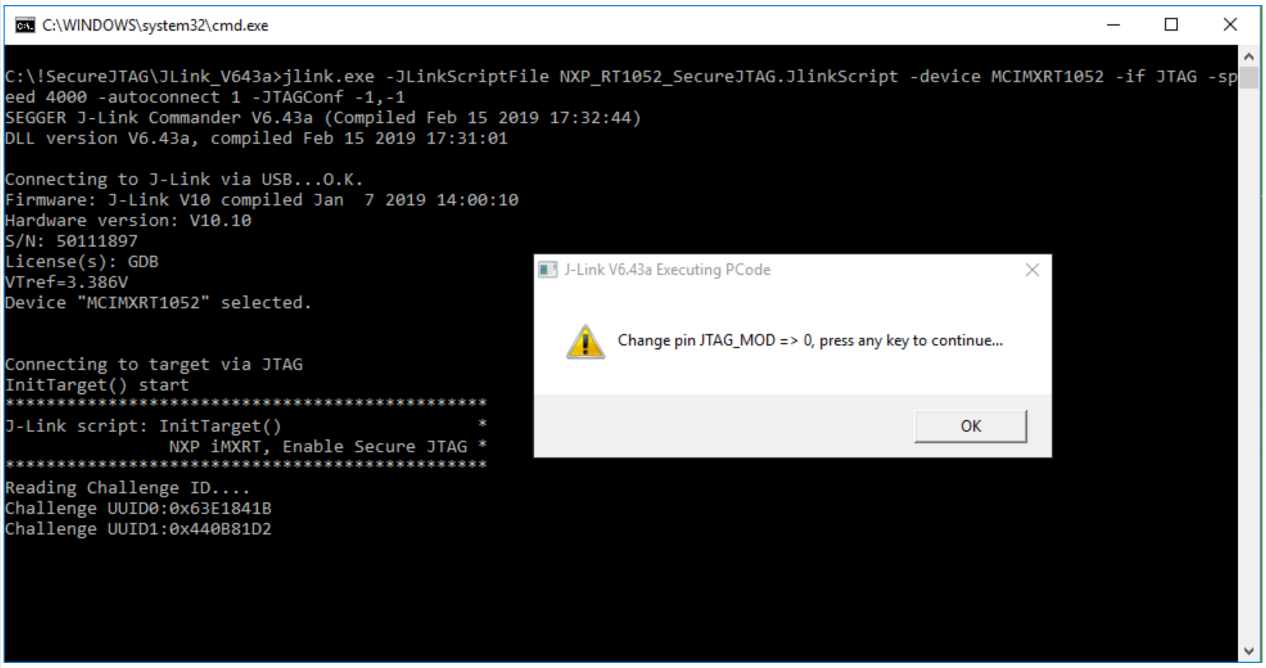


```
C:\!SecureJTAG\JLink_V643a>jlink.exe -JLinkScriptFile NXP_RT1052_SecureJTAG.JlinkScript -device MCIMXRT1052 -if JTAG -sp
eed 4000 -autoconnect 1 -JTAGConf -1,-1
SEGGER J-Link Commander V6.43a (Compiled Feb 15 2019 17:32:44)
DLL version V6.43a, compiled Feb 15 2019 17:31:01

Connecting to J-Link via USB...O.K.
Firmware: J-Link V10 compiled Jan  7 2019 14:00:10
Hardware version: V10.10
S/N: 50111897
License(s): GDB
VTref=3.386V
Device "MCIMXRT1052" selected.


Connecting to target via JTAG
InitTarget() start
***********************************************
J-Link script: InitTarget()                   *
                NXP iMXRT, Enable Secure JTAG *
***********************************************
```

Set pin JTAG_MOD => 1 and press any key to continue...

**Figure 5. Running SEGGER J-Link script on the secured i.MX RT device**

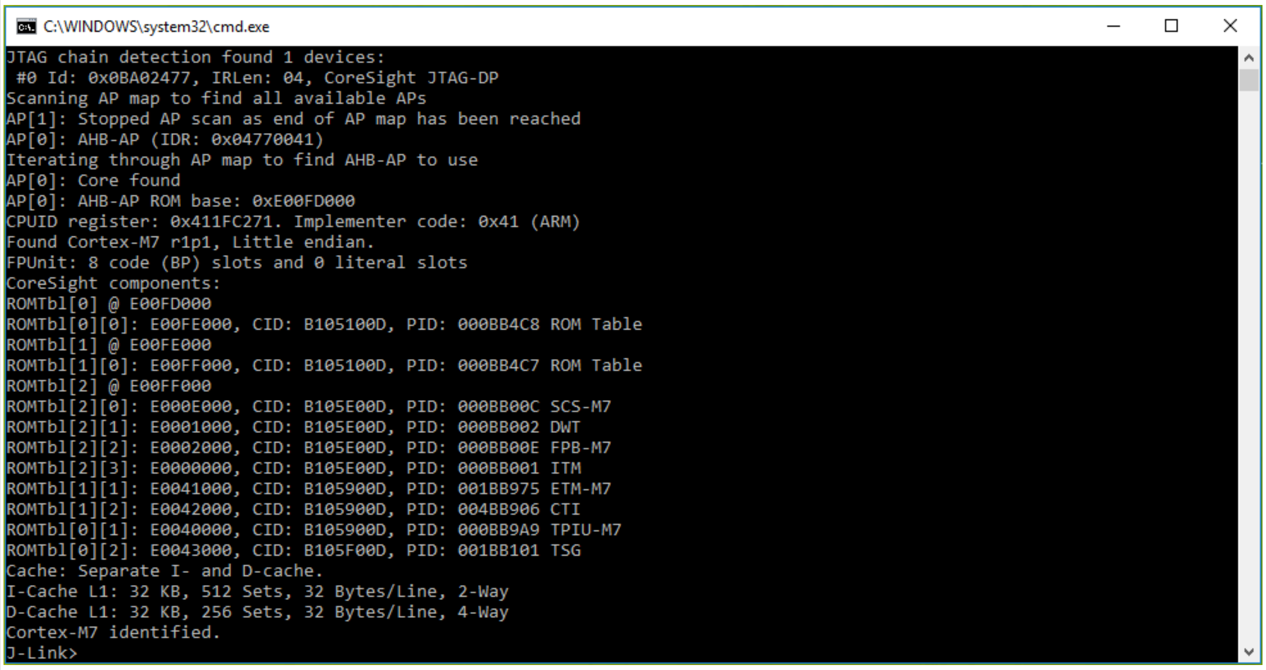
The debug tool should successfully attach to the i.MX RT10xx target over JTAG. The screen capture in Figure 7 shows a successful attach over Secure JTAG:



```
C:\!SecureJTAG\JLink_V643a>jlink.exe -JLinkScriptFile NXP_RT1052_SecureJTAG.JlinkScript -device MCIMXRT1052 -if JTAG -sp
eed 4000 -autoconnect 1 -JTAGConf -1,-1
SEGGER J-Link Commander V6.43a (Compiled Feb 15 2019 17:32:44)
DLL version V6.43a, compiled Feb 15 2019 17:31:01

Connecting to J-Link via USB...O.K.
Firmware: J-Link V10 compiled Jan  7 2019 14:00:10
Hardware version: V10.10
S/N: 50111897
License(s): GDB
VTref=3.386V
Device "MCIMXRT1052" selected.


Connecting to target via JTAG
InitTarget() start
***********************************************
J-Link script: InitTarget()                   *
                NXP iMXRT, Enable Secure JTAG *
***********************************************
Reading Challenge ID....
Challenge UUID0:0x63E1841B
Challenge UUID1:0x440B81D2
```

```
Change pin JTAG_MOD => 0, press any key to continue...
```

**Figure 6. SEGGER J-Link script reads Challenge ID**



```
JTAG chain detection found 1 devices:
 #0 Id: 0x0BA02477, IRLen: 04, CoreSight JTAG-DP
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x04770041)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FD000
CPUID register: 0x411FC271. Implementer code: 0x41 (ARM)
Found Cortex-M7 r1p1, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl[0] @ E00FD000
ROMTbl[0][0]: E00FE000, CID: B105100D, PID: 000BB4C8 ROM Table
ROMTbl[1] @ E00FE000
ROMTbl[1][0]: E00FF000, CID: B105100D, PID: 000BB4C7 ROM Table
ROMTbl[2] @ E00FF000
ROMTbl[2][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[2][1]: E0001000, CID: B105E00D, PID: 000BB002 DWT
ROMTbl[2][2]: E0002000, CID: B105E00D, PID: 000BB00E FPB-M7
ROMTbl[2][3]: E0000000, CID: B105E00D, PID: 000BB001 ITM
ROMTbl[1][1]: E0041000, CID: B105900D, PID: 001BB975 ETM-M7
ROMTbl[1][2]: E0042000, CID: B105900D, PID: 004BB906 CTI
ROMTbl[0][1]: E0040000, CID: B105900D, PID: 000BB9A9 TPIU-M7
ROMTbl[0][2]: E0043000, CID: B105F00D, PID: 001BB101 TSG
Cache: Separate I- and D-cache.
I-Cache L1: 32 KB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KB, 256 Sets, 32 Bytes/Line, 4-Way
Cortex-M7 identified.
J-Link>
```

**Figure 7. SEGGER J-Link successfully connected to Secured JTA**

Users can now perform normal JTAG debugger operations, as the device has been authenticated using the Challenge-Response mechanism.

***Note:***

*Any reset after JTAG access authorization shifts the JTAG controller back to its lock state, requiring that this authentication process is repeated.*

6. To ensure that the i.MX RT series SJC is operating in secure mode, edit the "NXP_RT1052_SecureJTAG.JlinkScript" file, provide an incorrect response key, and rerun the script. The debug tool should fail to attach to the i.MX RT10xx series target over JTAG.

### 4.2 Example of SEGGER J-link Secure JTAG unlock script

```
int InitTarget(void) {
  int v;
  int Key0;
  int Key1;
  JLINK_SYS_MessageBox("Set pin JTAG_MOD => 1 and press any key to
 continue...");
  // Secure response stored @ 0x600, 0x610 in eFUSE region (OTP memory)
  Key0 = 0x87654321;
  Key1 = 0xedcba9;
  JLINK_CORESIGHT_Configure("IRPre=0;DRPre=0;IRPost=0;DRPost=0;IRLenDevice=5");
  CPU = CORTEX_M7;
  JLINK_SYS_Sleep(100);
  JLINK_JTAG_WriteIR(0xC); // Output Challenge instruction
  // Readback Challenge, Shift 64 dummy bits on TDI
  JLINK_JTAG_StartDR();
  JLINK_SYS_Report("Reading Challenge ID....");
  // 32-bit dummy write on TDI / read 32 bits on TDO
  JLINK_JTAG_WriteDRCont(0xffffffff, 32);
  v = JLINK_JTAG_GetU32(0);
  JLINK_SYS_Report1("Challenge UUID0:", v);
  JLINK_JTAG_WriteDREnd(0xffffffff, 32);
  v = JLINK_JTAG_GetU32(0);
  JLINK_SYS_Report1("Challenge UUID1:", v);
  JLINK_JTAG_WriteIR(0xD); // Output Response instruction
  JLINK_JTAG_StartDR();
  JLINK_JTAG_WriteDRCont(Key0, 32);
  JLINK_JTAG_WriteDREnd(Key1, 24);
  JLINK_SYS_MessageBox("Change pin JTAG_MOD => 0, press any key to
 continue...");
  return 0;
}
```

## 5 Conclusion

This application note describes the eFuse configuration for Secure JTAG and the authentication process, which is validated and demonstrated using the SEGGER J-Link script.

## 6 Reference

1. *Configuring Secure JTAG for the i.MX 6 Series Family of Application Processors* (document AN4686)
2. *Security reference Manual for the i.MX RT1050 Processor* (document IMXRT1050SRM)
3. *J-Link / J-Trace User Guide* (document UM08001)
4. *Training JTAG Interface, Lauterbach TRACE32* (document Training JTAG Interface)
5. *HAB Code-Signing Tool User's Guide* (Rev. 3.2.0, 04/2019) (document IMX_CST3.2.0_TOOL)

## 7 Note about the source code in the document

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

## 8 Revision history

**Table 3. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN12419 v.3.0 | 18 December 2023 | Section 1 and Section 5 are updated. |
| AN12419 v.2.0 | 19 September 2023 | The document is updated to correspond to the latest guidelines, Section 1 is updated. |
| AN12419 v.1.0 | 11/2019 | HAB_JDE bit information is added. |
| AN12419 v.0.0 | 04/2019 | Initial version |

## Legal information

## Contents

**Date of release: 18 December 2023**
**Document identifier: AN12419**