AN13390

Step by step process to prepare Secure Boot for LPC54S0xx

Rev. 1 — 16-Dec-2021 Application Note

Contents

1 Introduction

The LPC54S0xx is a family of ARM© Cortex-M4 based microcontrollers for embedded applications that features a rich peripheral set with very low power consumption and enhanced security features. This family of microcontrollers includes:

- One Time Programmable (OTP) storage, which stores various flags (as fuses), Root of Trust Hash, and AES keys.
- Physically Unclonable Function (PUF) controller based on SRAM. This controller enables the secure generation of a unique device fingerprint and device-unique cryptographic keys.

The SRAM PUF mechanism is tightly integrated into the LPC54S0xx family. This feature enables keys from the PUF to be directly used by the device's internal AES-256 encryption engine. The unique and unclonable keys provide significant security benefits over other means of key injection or storage. The PUF keys reduce the threat of break-once repeat-everywhere attacks due to the foundation of the security on device-unique unclonable keys.

This application note describes a step by step process to create an Enhanced Boot Image (Encrypted then signed) using PUF key store.

Following Secure Boot features and Security peripherals are available for this family of devices.

- **Secure Boot features on LPC54S0xx devices:**
  - Supports boot image authentication using RSASSA-PKCS1-v1_5 signature verification with 2048-bit public keys (2048-bit modulus, 32-bit exponent).
  - Supports Root of Trust (RoT) establishment by comparing the SHA-256 hash digest of the RoT public key with OTP memory contents.
  - Supports secure anti-rollback of images through revocation of image key certificate. Supports up to 8 revocations through OTP fuses.
  - Supports boot of AES-GCM encrypted images with:
    - 128-bit symmetric key stored in OTP memory or
    - 256-bit symmetric key stored using on-chip SRAM PUF.
  - Supports Secure Authentication Boot by enforcing RSA-2048 signed image only boot.
  - Supports Encrypted Image Boot by enforcing AES-GCM encrypted images boot only.
  - Supports Enhanced Image Boot by enforcing encrypted, then signed images boot only.
- LPC54S0xx supports AES-128/AES-256 encryption/decryption engine with keys stored in poly-fuse OTP or PUF key store.
- Random number generator can be used to create keys.

- Secure Hash Algorithm (SHA1/SHA2) module supports secure boot with dedicated DMA controller.
- Physical Unclonable Function (PUF) root key using dedicated SRAM for silicon fingerprint. PUF can generate, store, and reconstruct key sizes ranging from 64 to 4096 bits.

2 Secure Boot

LPC54S0xx family of devices contain 64 KB ROM memory with firmware (known as boot code) in them. The boot code must always run when the device is powered-on or is hardware-reset. The LPC54S0xx has no internal flash for code and data storage. Therefore, images must be stored elsewhere to download on reset or it can be executed from an external memory (XIP). The boot ROM supports loading of images into on-chip RAM from external non-volatile memory devices connected to SPI, SPIFI, and EMC interfaces. The boot ROM forms the Root of Trust (RoT), which can secure the boot process by preventing the loading of non-authentic application code. If the code is authentic, control is transferred to it. This establishes a chain of trusted code from ROM to the user code. Boot ROM uses the following cryptography algorithms:

- SHA256 is used for hash function.
- NXP-defined 'Image key certificate' is used for public key certificates to verify that the public key in the certificate belongs to the OEM.
- The public key of the certificate authority or the Root of Trust Key is validated using SHA256 hash check against the OTP contents.
- RSASSA-PKCS1-v1_5-SIGN with SHA-256 hash digest is used for signature verification function. Using RSA keys with 2048-bit public key modulus and 32-bit public key exponent.
- AES algorithm is GCM mode used for encrypted boot.
  - A 128-bit AES key is used when OTP is used for key store.
  - A 256-bit AES key is used when PUF is used for key store.

LPC54S0xx family supports different secure boot modes based on the secure boot policy. The following sections describe important components and concepts required to prepare the device for secure boot.

2.1 Secure boot policies and types of secure boot images

The LPC54S0xx family supports the following secure boot policies enforced through OTP bit settings:

- Enforced booting of RSA-2048 signed images only (`SECUREBOOTTYPE = b'01` in OTP bank3 word 1). This is termed as **Secure Authentication Only Boot** in *LPC540xx/LPC54S0xx User Manual*.
- Enforced booting of AES-GCM encrypted images only (`SECUREBOOTTYPE = b'10` in OTP bank3 word 1). This is termed as **Encrypted Image Boot** in *LPC540xx/LPC54S0xx User Manual*.
- Enforced booting of encrypted, then signed images only (`SECUREBOOTTYPE = b'11` in OTP bank3 word 1). This is termed as **Enhanced Image Boot** in *LPC540xx/LPC54S0xx User Manual*.

The Table 1 summarizes the above policies.

Table 1. LPC540xx OTP memory bank 3, word 1 bits

Bits	Symbol	Value	Description (Secure Boot Type)
4:3	SECUREBOOTTYPE	b'00	Invalid if `SECUREBOOTEN` bit (Bit 2 of this register) is set in OTP memory bit or in header.
		b'01	Enforces booting of RSA-2048 signed images only.
		b'10	Enforces booting of AES-GCM encrypted images only.
		b'11	Enforces booting of encrypted then signed images only.

Based on the above policies to authenticate secure boot code before running, the plain user image is signed and/or encrypted. The secure boot supports the following types of secure boot images:

- Signed image: RSA-2048 signed images.
- Encrypted image: AES-GCM encrypted and authenticated images.
- Signed encrypted image: Plain image is encrypted first then signed. This policy is used in this application note.
- Encrypted signed image: Plain image is signed first, and then whole image including signature is encrypted.

All secure boot images have an image header that provides various parameters to the secure boot to initialize the boot interface, load the address, and authenticate or decrypt the image.

2.2 PUF key and OTP AES key store

An AES key is used to encrypt image when Enhanced Image Boot policy is used for secure boot. LPC54S0xx Family supports both OTP and PUF key store to store AES key. OTP only supports 128-bit AES key storage whereas PUF supports 256-bit AES key and the key is stored in the form of key code. In this product family, there is a difference in the sequence how keys are used by AES engine during encryption and decryption. This difference can be shown in below example.

Suppose, the 256-bit AES key generated by `elftosb` tool is:

```
73727170 63626160 53525150 43424140 33323130 23222120 13121110 03020100
```

User can use the above key sequence to generate PUF key store without changing sequence. However, when the above sequence is used for PUF key storage, the sequence to encrypt image using this key should be in below order:

```
03020100 13121110 23222120 33323130 43424140 53525150 63626160 73727170
```

As can be seen the last word goes first, then second last, and all other words follow in this order.

This example is demonstrated in this Application Note.

NOTE

When OTP is used to store key, AES-128 bit key is programmed in the same order because this key is used for image encryption. Therefore, when OTP is used for Key storage, key sequence change is not required.

2.3 Using blhost PC App to create Secure Boot

Programming OTP is required, for example, in SHA2 digest of RoT public Key, Secure boot type bits, and so on. To program OTP, the `blhost` utility tool is recommended, which can be downloaded from the NXP website. The blhost application is used on a host computer to issue commands to an NXP device running an implementation of the MCU bootloader.

The `blhost` application with the MCU bootloader, allows a user to program a firmware application onto the MCU device without a programming tool. LPC54S0xx does not have an inbuilt MCU bootloader that can communicate with `blhost`. Therefore, here we will build this utility from NXP SDK example (flashloader example) and use DFU utility to push flash loader in the device RAM. To create secure boot image, tools such as `elftosb-gui`, `elftosb`, `HxD`, and `image creator tool` are also used.

Below is the summary of tools used in a Table 2. These tools should be installed to prepare and program the secure boot image.

Table 2. Summary of tools used

Tool	Description
`DFU utility`	The DFU utility is the host application used to load the Flashloader binary into the internal RAM memory of LPC540xx device connected to the host in USB DFU mode. The `dfu-util.exe` is an open source command-line application. To download the tool, see dfu-util.

Table continues on the next page...

Table 2. Summary of tools used (continued)

Tool	Description
blhost	PC Command-Line Interface (CLI) tools to implement MCUBOOT protocol, it is part of MCUBOOT software package. The blhost.exe utility is an example host program used to interface with LPC54S0xx running the Flashloader program. This tool can be downloaded from MCUBOOT.
elftosb	The elftosb tool creates a binary output file that contains the user application image along with a series of bootloader commands. The output file is known as a Secure Binary or SB file for short. These files have the *.sb extension. The tool uses an input command file to control the sequence of bootloader commands present in the output file. This command file is called a boot descriptor file or BD file for short. This tool can be downloaded from MCUBOOT.
elftosb-gui	The GUI tool, elftosb-gui helps the user prepare a secure application image, as well as other useful security operations specific to target MCU platform. The Elftosb-gui tool provides intuitive graphical interface on top of elftosb and blhost command-line applications. It also guides the user in preparation of secure boot images required by ROM bootloader. This tool can be downloaded from MCUBOOT.
Image Creator Tool	The LPC54S0xx secure image creator tool is a command-line tool to create LPC54S0xx secure images. The "lpc54xxx_imgcr" tool has multiple sub-commands to perform the different operations related to LPC54S0xx secure image creation. This includes generating keys, certificates, signing images, and encrypting the binary images bootable by LPC54S0xx boot ROM. The example described in this application note uses lpc54xxx_imgcr_v0.1.013 (image creator Tool version v0.1.013).
Flashloader	The Flashloader is the secondary bootloader program loaded into the on-chip RAM of LPC54S0xx to support blhost. The project is located in SDK.
HxD	HxD is a binary file editor. It is easy to use and HxD is free of charge for private and commercial use.

3 Implementation

This section provides information how to implement secure boot in LPC54S0xx family of devices. This application note uses LPCXpresso54S018 development board (LPC54S018-EVK), MCUXpresso IDE version 11.4.0, and SDK version 2.10.0.

For other documents related to the board, refer to LPCXpresso54S018 development board Design Resources.

3.1 Implementation overview

Following steps are required to generate and program the encrypted then signed image for LPC54S0xx devices.

- Import **SDK Flashloader** Project in **MCUXPresso IDE** and build the binary file.
- Import the project from SDK or create your own application project. Build the binary file to be downloaded in the device.
- Download DFU Utility to program flashloader binary in the device. Use DFU mode to load flashloader binary in the device RAM.
- Generate AES key, RoT key, Image key, and the image certificate.
- Enroll PUF and create key store for AES key.
- First encrypt and then, sign the image using AES key.
- Program the above mentioned image in the SPIFI flash.
- Program RoT Hash, Secure boot option with PUF.
- Reset the board and run secure boot code.

3.2 Building flashloader binary

After importing `lpcxpresso54s018_flashloader` example in MCUXpresso from SDK, an executable file is built. This application interacts with the device in order to communicate with `blhost` utility. MCUXpresso, by default, builds an `axf` file. `Blhost` utility uses the binary format to load the application. Therefore, it is also necessary to convert the `lpcxpresso54s018_flashloader.axf` file to `lpcxpresso54s018_flashloader.bin` as shown in Figure 1.

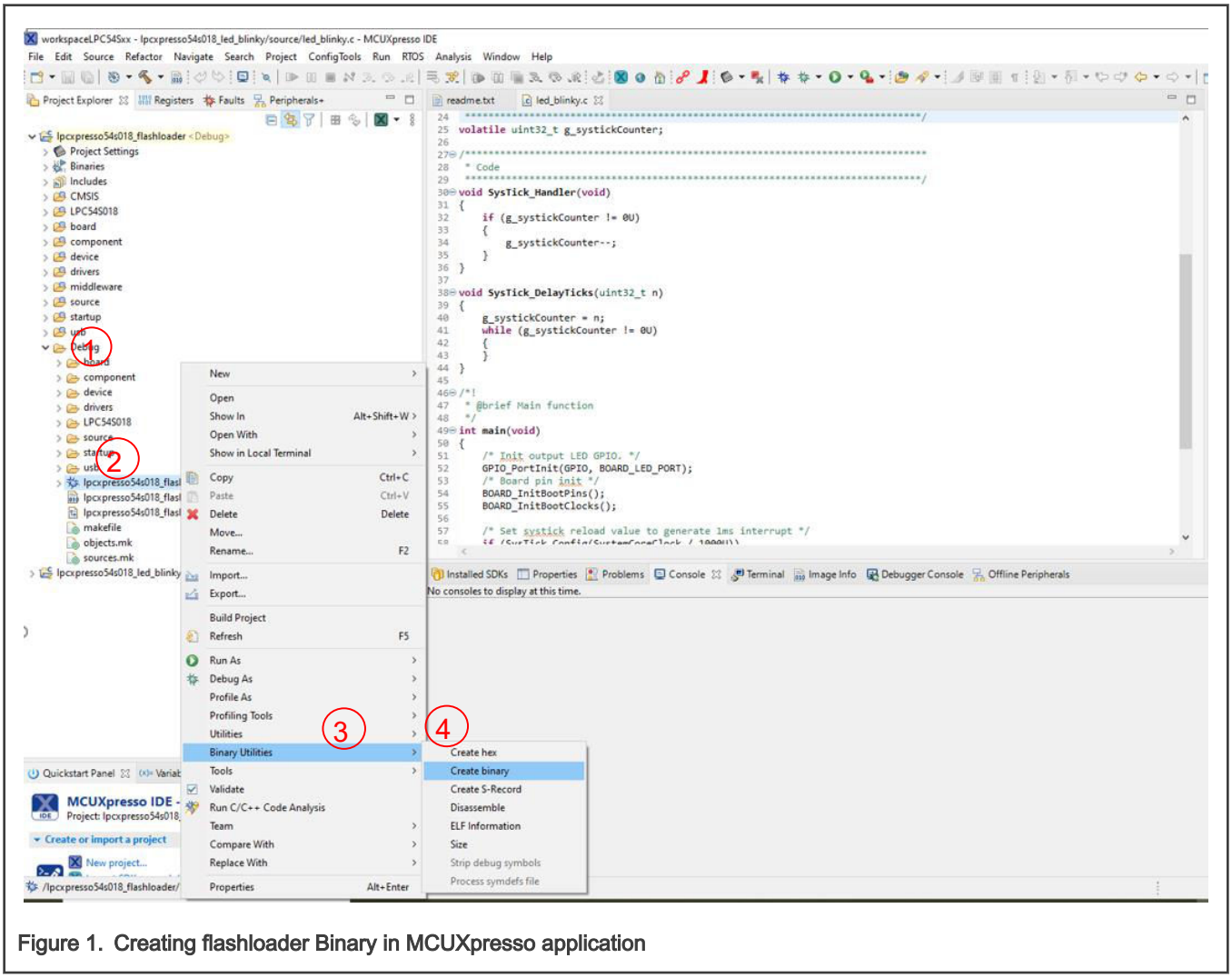

Figure 1. Creating flashloader Binary in MCUXpresso application

1. Click on `Debug` folder after building `lpcxpresso54s018_flashloader` project in MCUXpresso.
2. Select `lpcxpresso54s018_flashloader.axf` file and right click this file.
3. Select **Binary Utility**.
4. Select **Create binary** and it creates `lpcxpresso54s018_flashloader.bin` binary file in the `debug` folder.

The upcoming sections describe the process by which `lpcxpresso54s018_flashloader.bin` binary file is downloaded on the LPC54S018-EVK board.

3.3 Building the application binary

This application note describes procedure for encrypting and then signing the executable generated by `lpcxpresso54s018_led_blinky` project. The LED3 blinks after execution of the authenticated and decrypted file. Hence, this visually helps to identify that the file was authenticated and decrypted properly. The `blhost` utility uses the binary

format to load the application. Therefore, it is also necessary to convert the `lpcxpresso54s018_led_blinky.axf` file to `lpcxpresso54s018_led_blinky.bin` as shown in Figure 2.

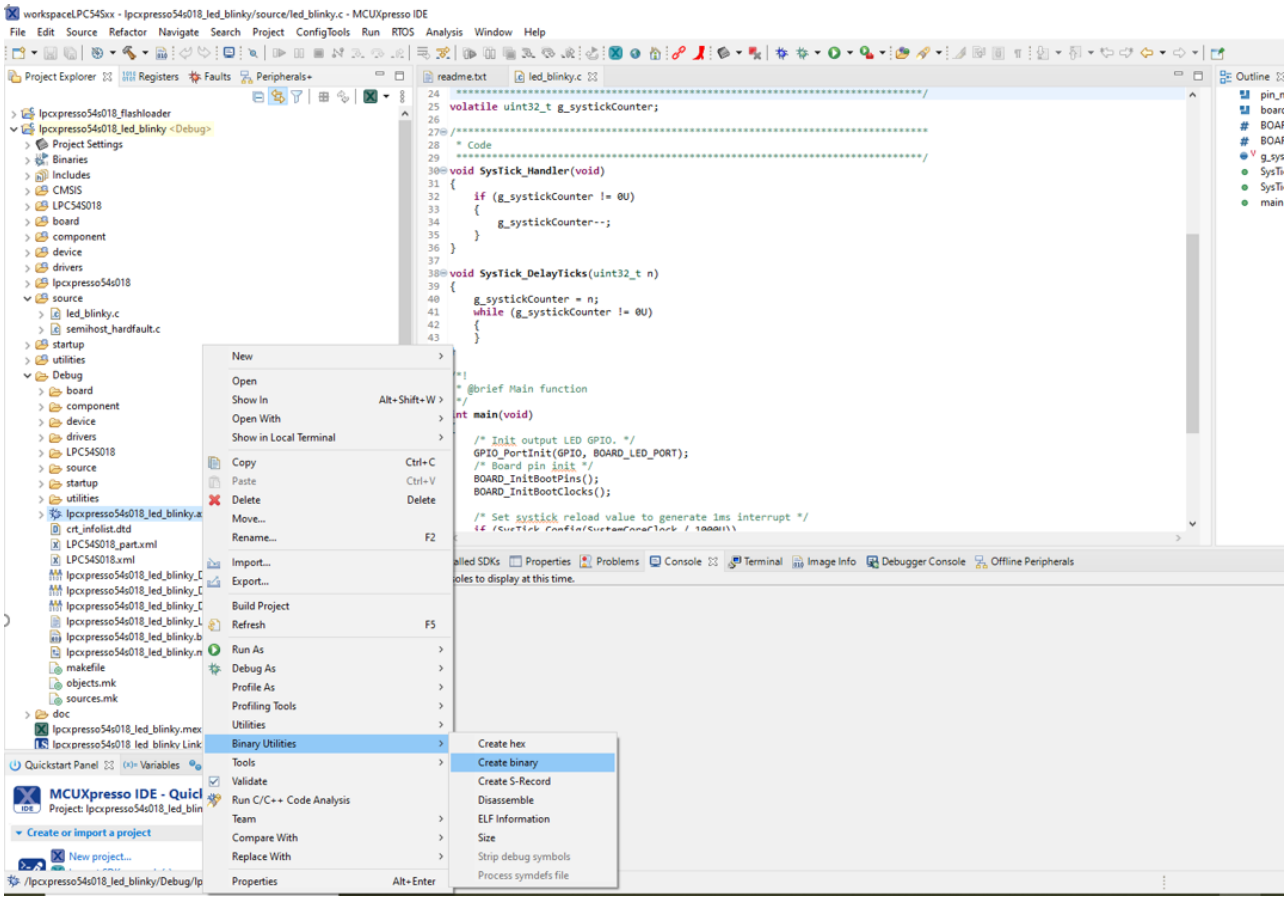

Figure 2. Creating `led blinky` Binary in MCUXpresso application

3.4 DFU utility and Flashloader download

Download DFU Utility from DFU Utility website. In this project, DFU Utility version 0.10 is used (`..\dfu-util-0.10-binaries\win64\dfu-util.exe`). This device supports USB DFU Utility to download flashloader binary into device RAM. To load flashloader binary, USB1 (High speed USB) should be configured in DFU mode. To download the flashloader binary, ISP pins must be configured as shown in below Table 3.

Table 3. Boot source selection based on ISP pins

Boot Mode	ISP2 (PIO0_6)	ISP1 (PIO0_5)	ISP0 (PIO0_4)	Description
USB1 DFU boot	LOW	LOW	HIGH	USB DFU class is used to download image over the USB1 High-Speed port into SRAM.

In the LPC54S018-EVK, use the ISP push buttons to select USB1 DFU mode. Then, connect J2 (USB1) to PC using a USB Micro cable. Run the following command in command prompt to load flashloader into RAM.

```
dfu-util.exe -D lpcxpresso54s018_flashloader.bin
```

The output is shown below.

```
C:\Work\DevTools\NXP\dfu-util-0.10-binaries\win64>dfu-util.exe -D lpcxpresso54s018_flashloader.bin
dfu-util 0.10

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2020 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Warning: Invalid DFU suffix signature
A valid DFU suffix will be required in a future dfu-util release!!!
Match vendor ID from file: 0000
Match product ID from file: 0000
Opening DFU capable USB device...
ID 1fc9:001f
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Device returned transfer size 512
Copying data from PC to DFU device
Download        [=========================] 100%        45120 bytes
Download done.
state(8) = dfuMANIFEST-WAIT-RESET, status(0) = No error condition is present
Resetting USB to switch back to runtime mode
Done!
```

Figure 3. Programming Flashloader using DFU Utility

After loading the flashloader, device LPC54S018-EVK is shown as USB Composite device with a `VID 0x1fc9` and `PID 0x01a2`. As mentioned before, the `blhost` utility is used to connect the device from host PC. To check if the device has connected successfully, use the following blhost command.

```
blhost -V -u 0x1fc9,0x01a2 -- get-property 1
```

If everything is fine, the blhost command displays success status. The command and output is shown below.

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -- get-property 1
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 1258424064 (0x4b020700)
Current Version = K2.7.0
```

Figure 4. Get Property command

3.5 Generating AES key and SHA256 of RoT key

To encrypt the image we need an AES key. Since PUF key store uses a 256-bit AES key, we must generate the key accordingly. Use the elftosb utility tool to generate the AES-256 bit key using the following command.

```
elftosb.exe --keygen 256 aes256_key.key
```

```
C:\Work\DevTools\NXP\elftosb_5.1.19\bin\win>elftosb.exe --keygen 256 aes256_key.key
wrote key file aes256_key.key

C:\Work\DevTools\NXP\elftosb_5.1.19\bin\win>
```

Figure 5. Generating AES key

The above key is a plain text key and can be view in any text editor. In Windows Command Prompt, the `Type` command shows the content as below.

```
Type aes256_key.key
B6B8F68E2CC99866EE9AA29B39988361D170658945D8BB8C56832840E0DFFC25
```

```
C:\Work\DevTools\NXP\elftosb_gui_1.0.12\bin\elftosb\win>Type aes256_key.key
B6B8F68E2CC99866EE9AA29B39988361D170658945D8BB8C56832840E0DFFC25
```

Figure 6. Show AES key

The above key is used to create the key store. However, to encrypt the image, the above key should be reversed.
We have reversed this key according to scheme described in Section PUF key and OTP AES key store, and stored in `aes256_keyReversed.key` file. The `Type` command can be used to see the contents again as shown below.

```
Type aes256_keyReversed.key
E0DFFC255683284045D8BB8CD170658939988361EE9AA29B2CC99866B6B8F68E
```

```
C:\Work\DevTools\NXP\elftosb_gui_1.0.12\bin\elftosb\win>Type aes256_keyReversed.key
E0DFFC255683284045D8BB8CD170658939988361EE9AA29B2CC99866B6B8F68E
```

Figure 7. AES reversed key

LPC54S0xx supports RSA key. To generate the RoT key, the image key, and the image key certificate, the image creator tool is used as shown below.

To generate the 2048-bit root key, use the command below:

```
lpc54xxx_imgcr.exe genrsakey rotk.pem
```

```
C:\Users\nxf67174>cd C:\Work\DevTools\NXP\ImageCreaterTool

C:\Work\DevTools\NXP\ImageCreaterTool>lpc54xxx_imgcr.exe genrsakey rotk.pem
[INFO] Generating 2048-bit RSA private key with exponent 3 ...
[INFO] done.
```

Figure 8. Generating the 2048-bit root key

Generate 2048-bit image key using the command below:

```
lpc54xxx_imgcr.exe genrsakey image_key.pem
```

```
C:\Work\DevTools\NXP\ImageCreaterTool>lpc54xxx_imgcr.exe genrsakey image_key.pem
[INFO] Generating 2048-bit RSA private key with exponent 3 ...
[INFO] done.
```

Figure 9. Generating 2048-bit image key

Generate image certificate with revocation ID 0 and signed by RoT key. Revocation ID is used during image key certificate validation:

```
lpc54xxx_imgcr.exe gencert -r rotk.pem -k image_key.pem --rid 0 image_key_cert.bin
```

```
C:\Work\DevTools\NXP\ImageCreaterTool>lpc54xxx_imgcr.exe gencert -r rotk.pem -k image_key.pem --rid 0 image_key_cert.bin
[INFO] Wrote signed certificate to image_key_cert.bin

C:\Work\DevTools\NXP\ImageCreaterTool>
```

Figure 10. Generating image certificate

3.6 Enrolling PUF and creating the Key Store

Use the steps below and the corresponding numbering as shown in the Figure 11. Use **elftosb-gui** to enroll PUF and generate key store.

1. Select the **LPC54S0xx** device.
2. Click the **Device** tab and select **USB**. Add `VID =0x1fc9` and `PID=0x01a2`.
3. Check **Enroll box** in SRAM PUF enroll.
4. Check **Image Key Code** box and add the AES keycode file.
5. Check the **Export** box and Export keystore in host PC.
6. Click the **Process** button to create keystore. After this step, the keystore binary is stored in the host PC at the location given in Step 5.

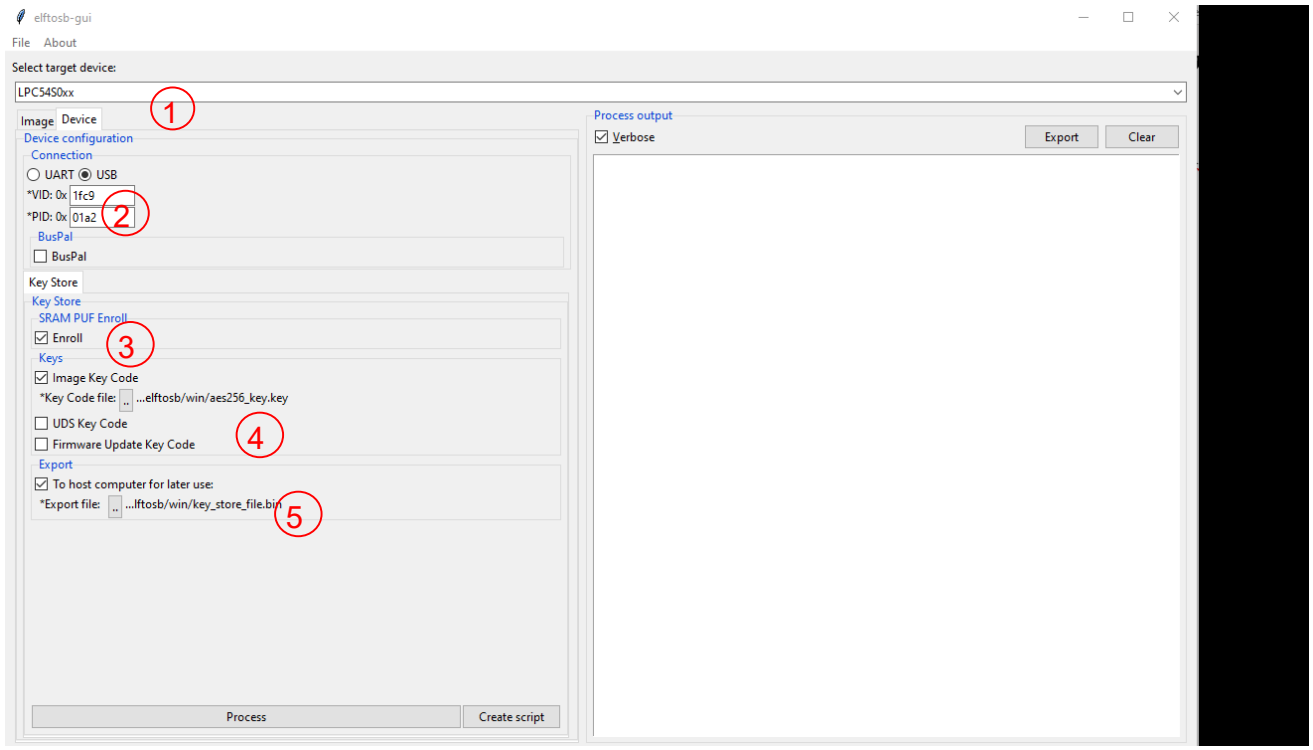

Figure 11. PUF key store generation

3.7 Generating encrypted then signed application binary

In this example, we are generating an encrypted and then signed blinky application image, which is then flashed in the device using `blhost` utility. Use the steps below and corresponding numbering shown in the Figure 12 to create encrypted signed binary.

1. Select the **LPC54S0xx** device.
2. Create a new configuration.
3. Select the plain blinky binary image generated in MCUXpresso IDE.
4. Get the load address from the input image.
5. Select the image execution target as RAM.
6. Select the image authentication type as **Signed +Encrypted**. This signifies that image is encrypted first and then it is signed.
7. Select the device key source as **Key Store** and select the reversed AES key **aes256_keyReversed.key** to encrypt the image.
8. Selected attached in Key Store and Key store file as **`key_store_file.bin`**.
9. Select `image_key_cert.bin` as Image Key Certificate. This Certificate is generated as described in Generating AES key and SHA256 of RoT key.
10. Select `image_key.pem` as private key for image required to sign the image. This Certificate is generated as described in Generating AES key and SHA256 of RoT key.
11. Select the path and name of the output encrypted, then signed image.
12. Click the **Process** button to create the secure signed bootable blinky image.

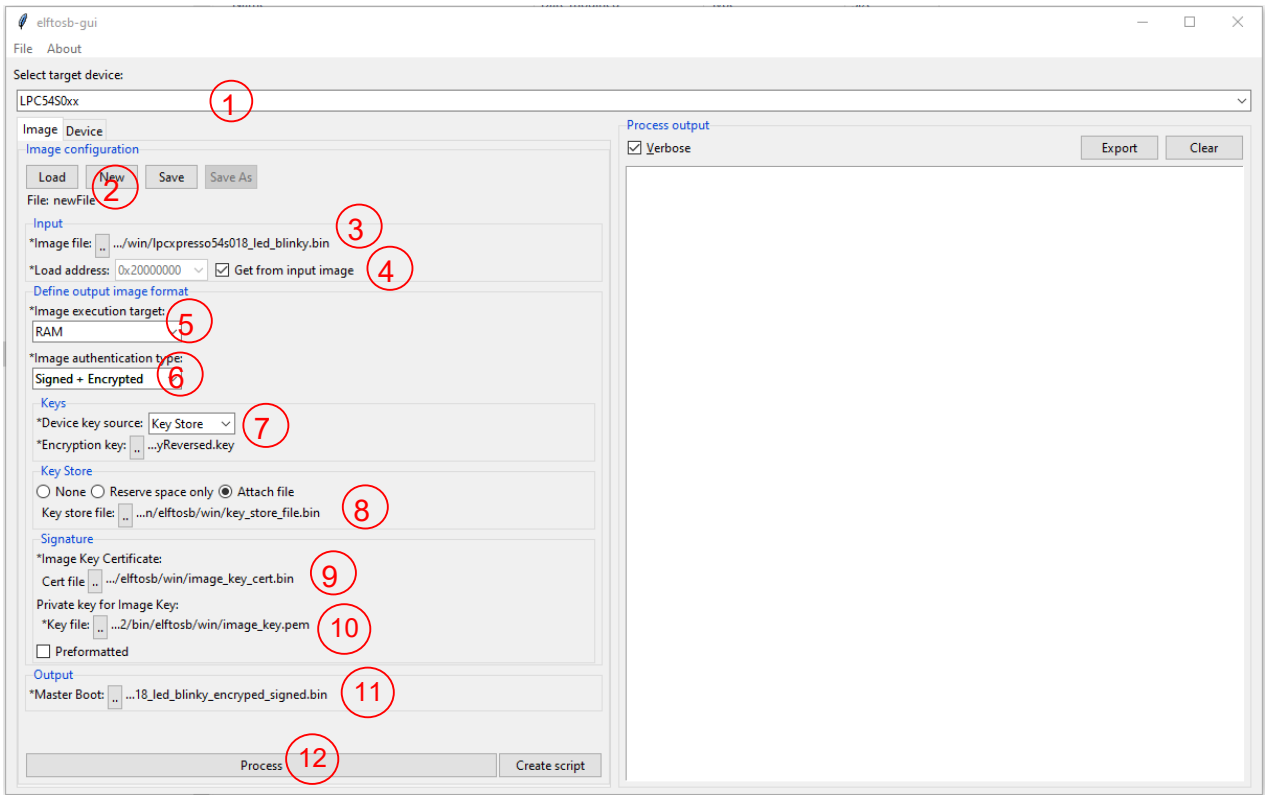

Figure 12. Example of data entry field

3.8 Programming the signed encrypted image

The encrypted and then signed blinky image can be programmed in the SPIFI NOR Flash using `blhost` utility as shown in below instructions.

1. First, check the reserved memory region by flashloader using the below command.

```
blhost -u 0x1fc9,0x01a2 -- get-property 12
```

The command above gives the following result.

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -- get-property 12
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 0 (0x0)
Response word 2 = 0 (0x0)
Response word 3 = 536870912 (0x20000000)
Response word 4 = 536928735 (0x2000e1df)
Reserved Regions =
        Region0: 0x20000000-0x2000E1DF (56.469 KB)
```

Figure 13. blhost property 12

2. Now, configure SPIFI flash in QUAD SDR read mode in unreserved region, as shown in the below instruction.

```
blhost -u 0x1fc9,0x01a2 -- fill-memory 0x2000f000 4 0xc0000004
```

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -- fill-memory 0x2000f000 4 0xc0000004
Inject command 'fill-memory'
Successful generic response to command 'fill-memory'
Response status = 0 (0x0) Success.
```

Figure 14. Configuring memory using blhost

3. Apply configuration from above address to SPIFI flash memory using the command below:

```
blhost -u 0x1fc9,0x01a2 -- configure-memory 0xa 0x2000f000
```

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -- configure-memory 0xa 0x2000f000
Inject command 'configure-memory'
Successful generic response to command 'configure-memory'
Response status = 0 (0x0) Success.
```

Figure 15. Configuring memory using blhost

4. The command below provides the starting address and other properties of the SPIFI flash.

```
blhost -u 0x1fc9,0x01a2 -- get-property 25 0xa
```

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -- get-property 25 0xa
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 15 (0xf)
Response word 2 = 268435456 (0x10000000)
Response word 3 = 16384 (0x4000)
Response word 4 = 256 (0x100)
Response word 5 = 4096 (0x1000)
Response word 6 = 0 (0x0)
UNKNOWN Attributes: Start Address = 0x10000000  Total Size = 16 MB  Page Size = 256 bytes  Sector Size = 4 KB
```

Figure 16. Checking Device configuration using blhost

5. As shown in the log from step 4, SPIFI flash address starts from 0x10000000. The command below erases SPIFI flash.

```
blhost -u 0x1fc9,0x01a2 -t 100000 -- flash-erase-region 0x10000000 0x100000
```

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -t 100000 -- flash-erase-region 0x10000000 0x100000
Inject command 'flash-erase-region'
Successful generic response to command 'flash-erase-region'
Response status = 0 (0x0) Success.
```

Figure 17. Erasing binary using blhost

6. Program SPIFI flash with the encrypted then signed blinky binary:

```
blhost -u 0x1fc9,0x01a2 -t 100000 -- write-memory
0x10000000 lpcxpresso54s018_led_blinky_signed.bin
```

```
C:\Work\DevTools\NXP\blhost_2.6.6\bin\win>blhost -u 0x1fc9,0x01a2 -t 100000 -- write-memory 0x10000000 lpcxpresso54s018_led_blinky_signed.bin
Inject command 'write-memory'
Preparing to send 12068 (0x2f24) bytes to the target.
Successful generic response to command 'write-memory'
(1/1)100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.
Wrote 12068 of 12068 bytes.
```

Figure 18. Programming the binary image using `blhost` utility

3.9 Programming RoTK HASH

Root of Trust Hash keys are programmed in the OTP bank 1 and bank 2. Image creator tool can be used to show the Hash value that should be programmed in the OTP.

```
lpc54xxx_imgcr.exe showotp -k rotk.pem
```

The output is shown below:

```
[INFO] Computing SHA256 hash of RoT key...
[INFO] SHA-256 of RoT key is: 0c8f92032ca5bc981e638a98e2585c50555d38ba0d19b739be4f9461bb60bd38
[INFO] RoTK SHA256 digest in
 OTP2_words[0,1,2,3]: ('555d38ba', '0d19b739', 'be4f9461', 'bb60bd38')
 OTP1_words[0,1,2,3]: ('0c8f9203', '2ca5bc98', '1e638a98', 'e2585c50')
```

```
C:\Work\DevTools\NXP\ImageCreaterTool>lpc54xxx_imgcr.exe showotp -k rotk.pem
[INFO] Computing SHA256 hash of RoT key...
[INFO] SHA-256 of RoT key is: 0c8f92032ca5bc981e638a98e2585c50555d38ba0d19b739be4f9461bb60bd38
[INFO] RoTK SHA256 digest in
 OTP2_words[0,1,2,3]: ('555d38ba', '0d19b739', 'be4f9461', 'bb60bd38')
 OTP1_words[0,1,2,3]: ('0c8f9203', '2ca5bc98', '1e638a98', 'e2585c50')
```

Figure 19. SHA-256 of RoT Key

The blhost utility is used to program the above OTP words as shown below.

```
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 4 0c8f9203
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 5 2ca5bc98
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 6 1e638a98
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 7 e2585c50
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 8 555d38ba
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 9 0d19b739
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 10 be4f9461
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 11 bb60bd38
```

3.10 Programming Secure Boot configuration bits

LPC54S0xx OTP memory bank 3, word 0 should be programmed to enable secure boot. Secure boot type in this case is the authentication and encryption enforced image, and PUF is enabled. The below table shows the description of the value to be programmed in OTP memory bank 3 word 0.

Table 4. LPC540xx OTP memory bank 3, word 0 bits

Bits	Symbol	Value	Description
1:0	PARITY	b'00	No parity of bits [16:2].

Table continues on the next page...

Table 4. LPC540xx OTP memory bank 3, word 0 bits (continued)

Bits	Symbol	Value	Description
2	SECUREBOOTEN	b’1	Secure is boot enabled.
4:3	SECUREBOOTTYPE	b’11	Enforces booting of encrypted then signed images only Secure Boot Type.
5	SIGNATURE_PREFORMAT	b’0	Signature is not pre-formatted.
6	SWDEN0	b’0	In secure mode, SWD is disabled irrespective of this bit.
7	ISP_PINS_DISABLED	b’0	Ability to enter ISP mode enabled,
8	ISP_IAP_DISABLED	b’0	Ability to enter ISP mode through IAP enabled.
10:9	BOOT_SRC	b’00	ISP pins are used as boot source.
12:11	ISP_MODE	b’00	Fall through ISP mode is USB1 DFU.
13	SWDEN1	b’0	In secure mode, SWD is disabled irrespective of this bit.
14	USE_PUF	b’1	PUF is enabled.
15	BLOCK_PUF_ENROLL_KEY_CODE	b’0	Allow PUF enrollment and key code generation.
16	-	b’0	Reserved.
19:17	QSPI_DELAY	b’000	ROM has to wait 0 microseconds after POR reset before access QSPI flash device.
23:20	USER APPLICATION	b’0000	Default value.
31:24	REVOKE_ID	b’00000000	No certificate revoked.

```
blhost -u 0x1fc9,0x01a2 -- efuse-program-once 12 0000401C
```

Reset the device. After resetting, the boot ROM verifies the secure image. If the image is authentic, control is passed to the user code. Then, led blinky application is executed and as a result, the LED3 on the LPC54S018-EVK blinks.

4 Revision history

The table below lists the changes to this document.

Table 5. Revision history

Revision number	Date	Substantive changes
1	16-Dec-2021	Minor updates
0	25-Oct-2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 16-Dec-2021

Document identifier: AN13390

arm