AN12283

LPC55Sxx Secure Boot

Rev. 2 — 11/2020 Application Note

Contents

1 Introduction

The LPC55xx/LPC55Sxx is an Arm Cortex® M33-based microcontroller for embedded applications.

These devices include:

- Up to 320 KB of on-chip SRAM
- Up to 640 KB on-chip flash
- High-speed and full-speed USB host
- Device interface with crystal-less operation for full-speed
- Five general-purpose timers
- One SCTimer/PWM
- One RTC/alarm timer
- One 24-bit Multi-Rate Timer (MRT)
- Windowed Watchdog Timer (WWDT)
- Eight flexible serial communication peripherals, each of it is a USART, SPI, I2C, or I2S interface
- One 16-bit up to 2.0 Msamples/sec ADC, temperature sensor

LPC55S/LPC55Sxx devices offer support for real-time encryption and decryption for on-chip flash using the PRINCE encryption engine. The following sections explain the use of LPC55Sxx PRINCE for data encryption of on-chip flash contents.

2 Secure boot options

This application note describes the secure boot feature for LPC55Sxx. The following sections provide an overview of secure boot and key provisioning details.

The LPC55Sxx allows booting of Public-Key signed images. The secure boot ROM supports three types of security protected modes. Secure boot with signed image, boot from encrypted PRINCE flash region, and secure boot with CDI calculation for the Device Identifier Composition Engine (DICE). Each of these options has attributes related to manufacturability, the firmware update scheme, and level of protection from attacks.

The ROM supports booting from encrypted Prince regions from internal flash. Encrypting images allows strong protection of intellectual property.

The ROM supports public keys and image revocation, that is, the method of not allowing new updates applies unless they are of a specific version. This is the basis for rollback protection.

The boot ROM supports pre-configuration of TrustZone-M settings, enablement of DICE and redundant image support (fall back image).

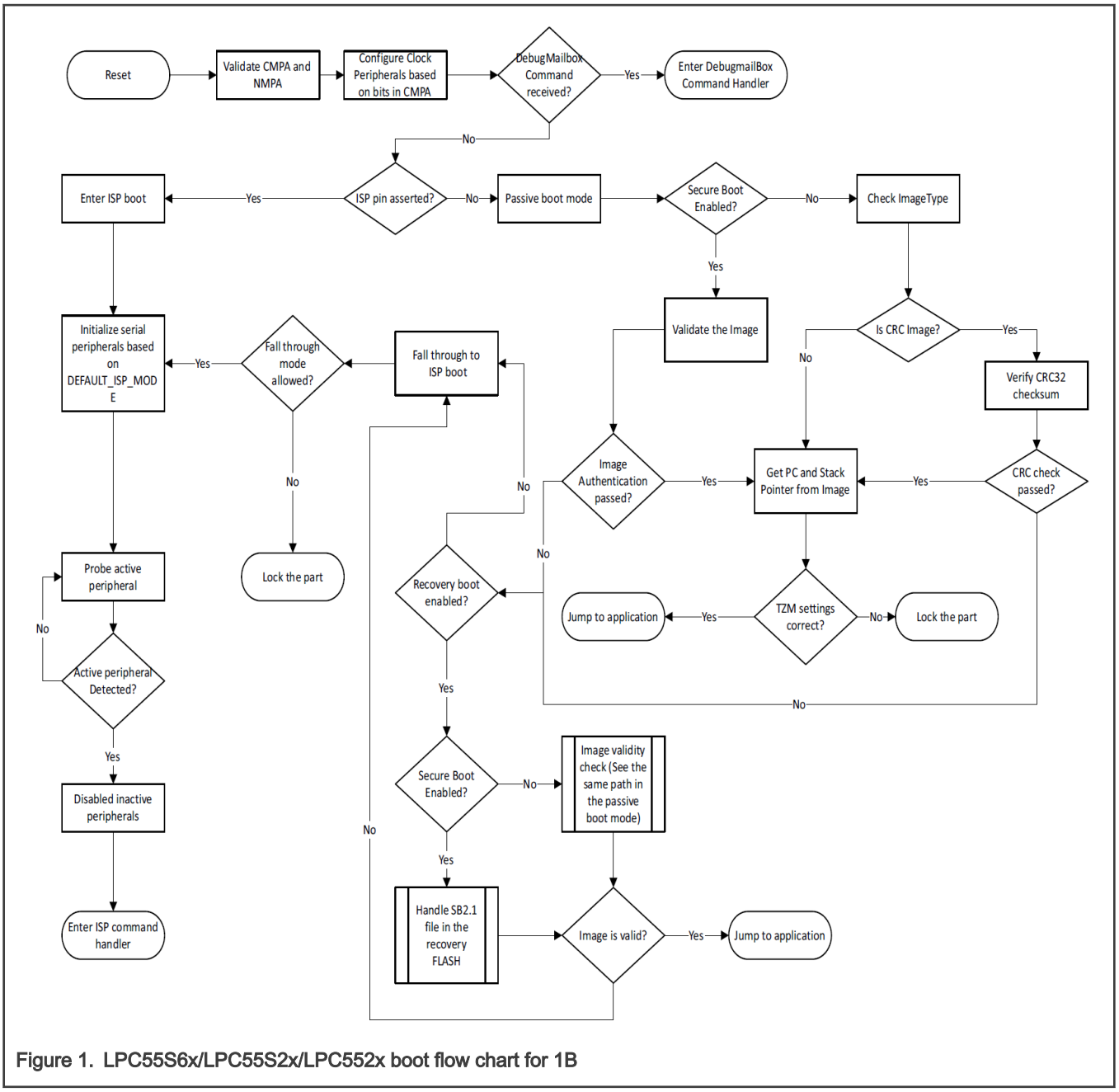

Figure 1. LPC55S6x/LPC55S2x/LPC552x boot flow chart for 1B

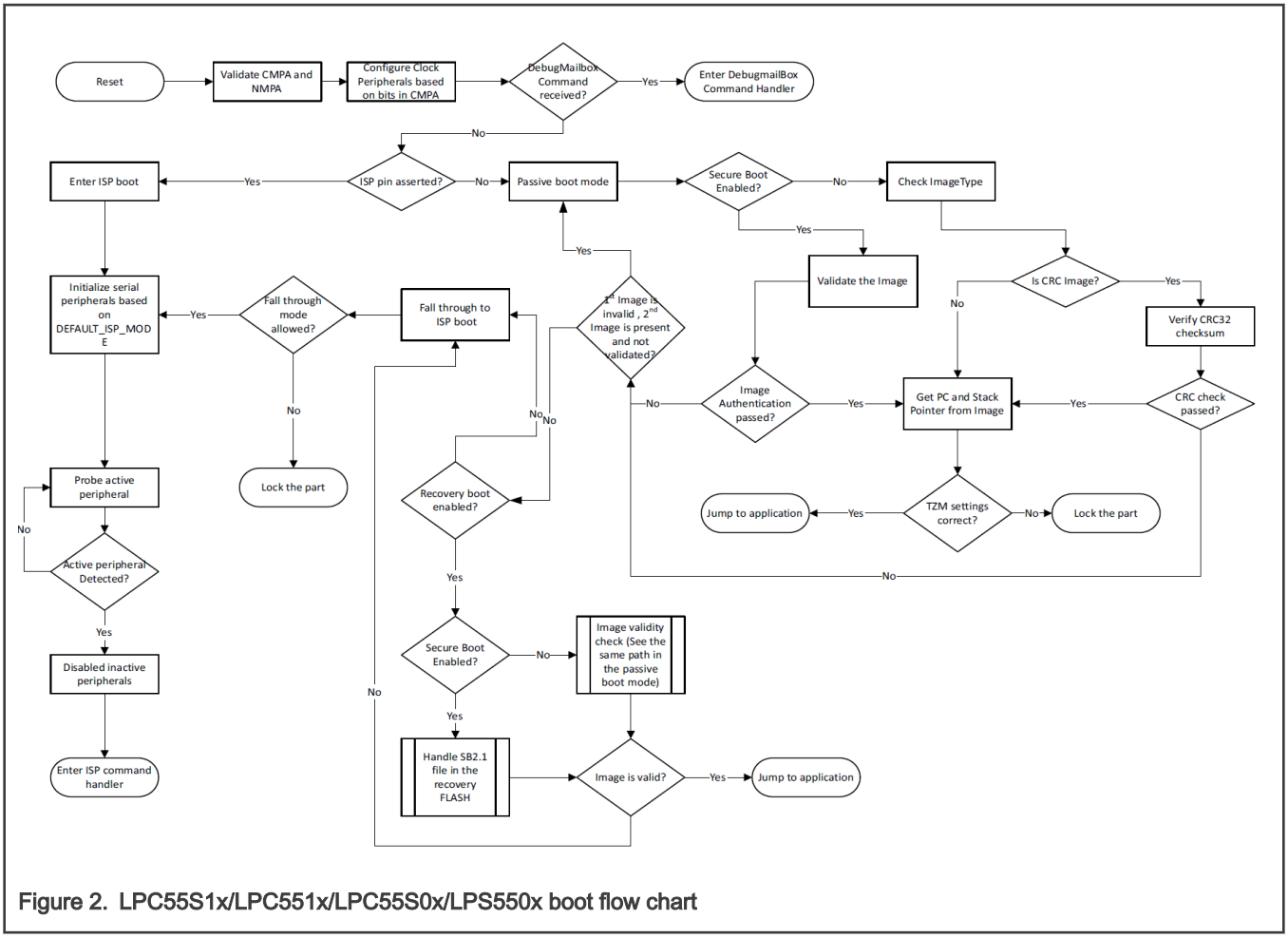

Figure 2. LPC55S1x/LPC551x/LPC55S0x/LPS550x boot flow chart

The application note describes the various types of non-secure and secure boot options.

- Non-secure boot images
  - Unsigned plain image
  - Unsigned plain CRC image
- Secure boot image
  - Public key signed image
    - Up to 4 Root of Trust (RoT) keys
    - Up to 16 image key certificates with image revocation feature
- Boot from encrypted PRINCE regions in internal flash
- Support pre-configuration of TrustZone-M® Settings

2.1 Unsigned plain image

Unsigned plain images are basic types of images. This is raw format generated by IDE for LPC55Sxx.

Figure 3. Unsigned plain image

2.2 Unsigned plain CRC image

The non-S versions of LPC55xxx support Unsigned plain CRC (UPC) images. The development life cycle state of secure devices (LPC55Sxx) supports the UPC images. These images contain a CRC32 field computed on the entire image (excluding the CRC32 field).

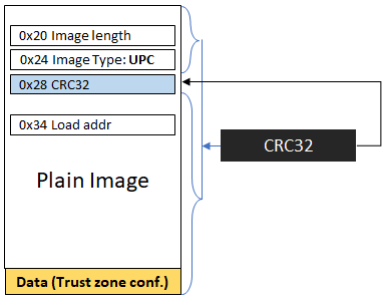

Figure 4. Unsigned plain CRC image

2.3 Signed image

LPC55Sxx devices support booting of RSA2048 signed images using RSASSA-PKCS1-v1_5 signature verification.

LPC55Sxx devices support 2048-bit or 4096-bit RSA keys and X.509 V3 certificates.

Image validation is a two-step process. The first step is the validation of the X.509 certificate inserted in the image. This contains the image public key used in the second step to validate the entire image (including the certificate) to allow customers to add additional PKI structure.

The signed image boot supports up to 4 Root of Trust (RoT) keys and up to 16 Image key certificates with image revocation feature.

Signed image format shows the signed image layout.

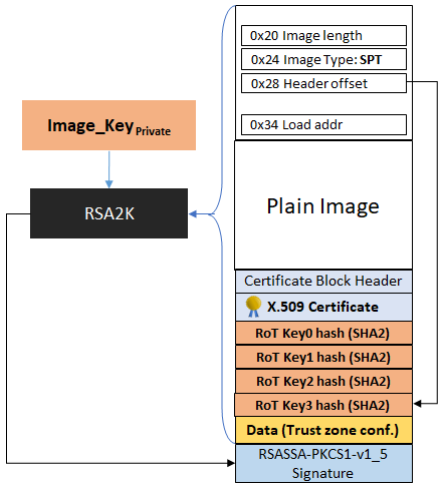

Figure 5. Signed image format

ROM does the following image authentication.

1. Validate X.509 digital certificate embedded in the image.
2. Uses Image_key (Public) to validate image signature.
    - ROM validates and extracts the Image public Key from x509 certificate embedded in the image.

ROM uses “SHA-256 with RSA signing” for image authentication.

- ‘RSA4K’ field in SECURE_BOOT_CFG control word present in the Protected Flash Region (PFR) determines the RSA key size.
- Default is 2048-bit (public key modulus size).
- If RSA4K bits are set to a non-zero value, then 4096-bit keys are enforced.

2.3.1 Image revocation

This section describes image revocation scheme in detail. Inside the certificate, there is a field with serial number of the certificate.

Inside the chip, there are bits in protected flash region (PFR). If serial number of digital certificate (which can be trusted because it is signed) does not align with the bits in PFR, the image validation process fails. The serial number can be a later version (higher by 1 version) but it cannot be an earlier version i.e. serial number value can jump forward but cannot roll back.

- A signed released image can be revoked by revoking the image key certificate.
- The serial number field (20 octets) in X.509 Image key certificate is used for revocation logic.
    - The 4 left-most octet shall have two mandatory bytes 0x3c, 0xc3, then 16-bit revocation identifier (little endian byte order).
    - For example, following is a command to generate X.509 certificate with a revocation identifier 0x0000.

      ```
      . openssl x509 -req -days 365 -in selfsign_v3.csr -signkey selfsign_privatekey_rsa2048.pem
      -sha256 -outform der -out selfsign_v3.der.crt -extfile v3_noca.ext -set_serial 0x3cc30000
      abababab
      ```

      Result in certificate: serial:3C:C3:00:00: AB : AB : AB : AB.
    - Only 17 revocation IDs are possible. (0x0, 0x1, 0x3, 0x7, 0xF, 0x1F, 0x3F, 0x7F, 0xFF … 0xFFFF).
- LPC55Sxx contains revocation identifier (IMAGE_KEY_REVOKE) in Customer Field Programmable Area (CFPA) of PFR.
- The FW update application is responsible to update the IMAGE_KEY_REVOKE.

- 1-bit should be set on every revocation starting from lower bit 0 to 16.
  - 0b0 - >0b1 -> 0b11->0b111 .....
- First FW should be written to flash and then the IMAGE_KEY_REVOKE should be updated.

- ROM validates an image public key using following steps:
  - Certificate is validated by verifying the RSA2K signature of the certificate.
  - Revocation ID in certificate should be same as IMAGE_KEY_REVOKE field.
  - To avoid bricking the device if power loss happens after FW update but before IMAGE_KEY_REVOKE is updated, LPC55Sxx boot ROM allows a roll-forward (only by 1), and cannot be rolled back.

2.4 Encrypted PRINCE flash region

LPC55Sxx supports on-the-fly encryption/decryption to/from internal flash through PRINCE.

Data stored in on-chip internal Flash could be encrypted in real time. LPC55Sxx supports 3 regions that allow multiple code images from independent encryption base to co-exist. Each PRINCE region has a secret-key supplied from on-chip SRAM PUF via secret-bus interface (not SW accessible). PRINCE encryption algorithm does not add latency.

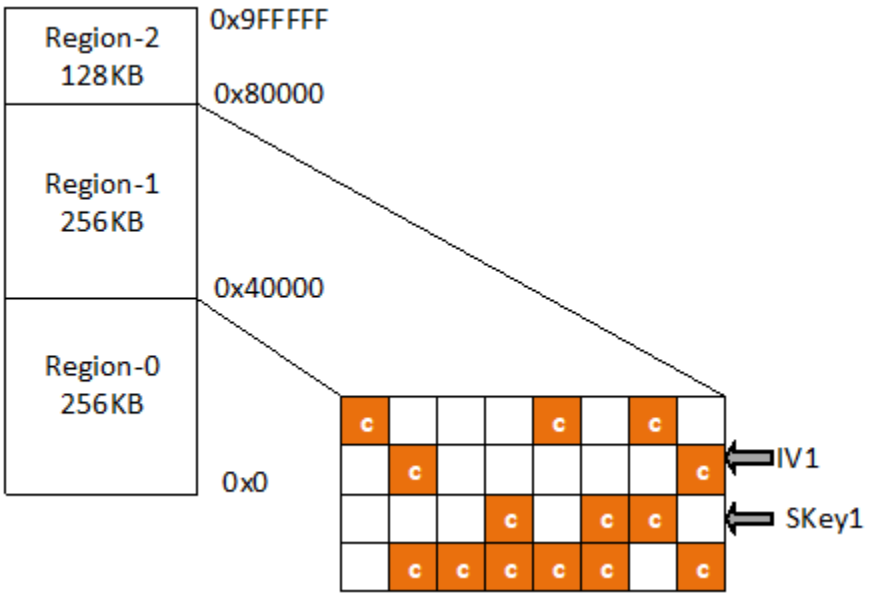

Figure 6. PRINCE region in LPC55S6x and LPC55S2x

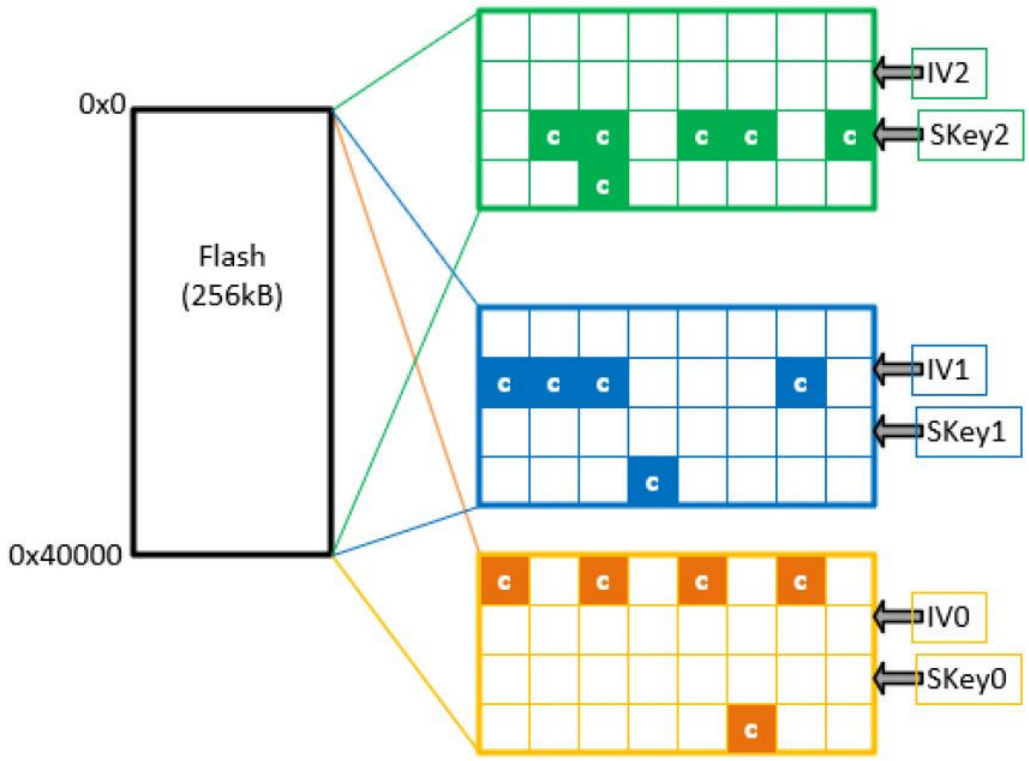

Figure 7. PRINCE regions in LPC55S1xand LPC55S0x

2.5 Protected flash region

LPC55Sxx contains configuration for the boot ROM in flash region which is protected. This protected region contains settings of boot configuration, security policy, PRINCE settings and so on.

Protected Flash Region with four regions:

1. Customer in-field Programming Area (CFPA)
   - Image revoke
   - RoT key revoke
2. Customer Manufacturing Programming Area (CMPA)
   - Boot configuration
   - RoT key table hash
   - Debug configuration
   - Prince configuration
3. Key Storage for PUF
4. NXP unique ID and manufacturing system

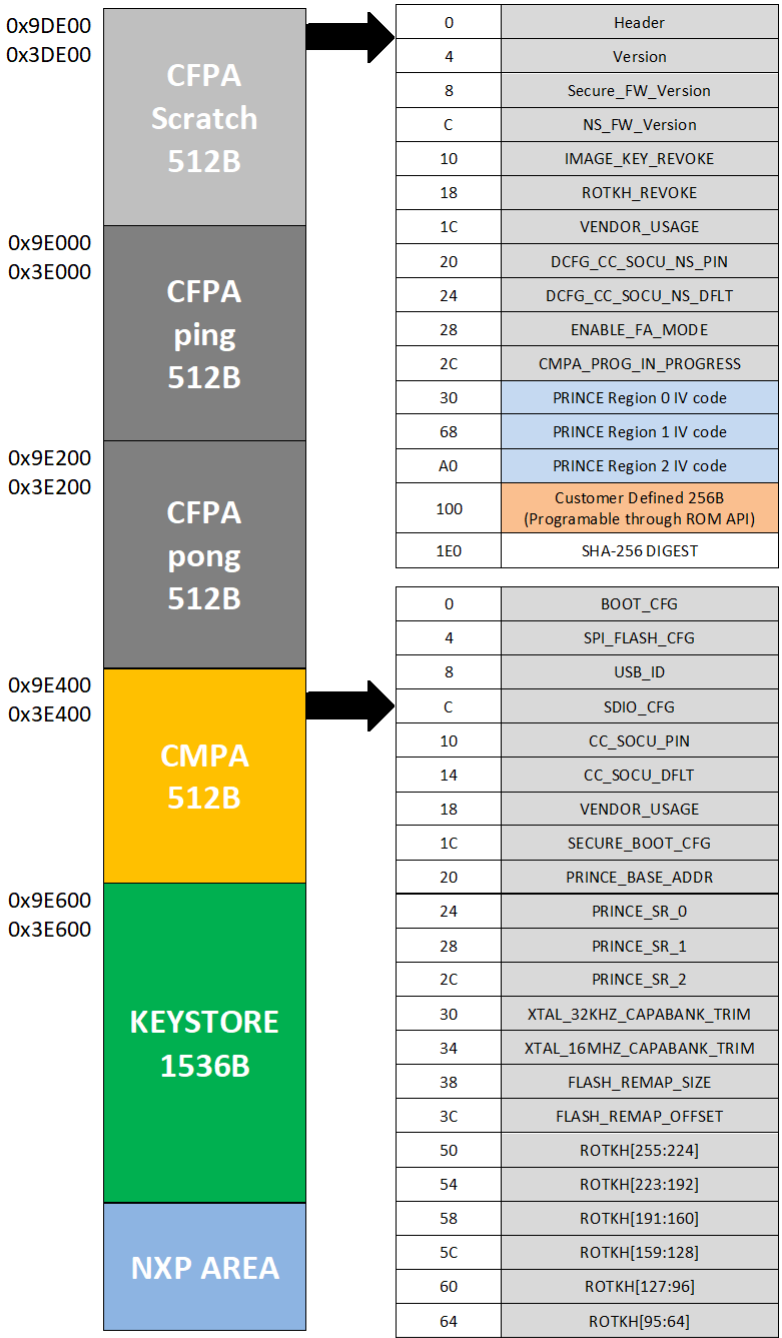

Figure 8. Protected flash region LPC55xx family

NOTE

The 0x9xxxx addresses are for the LPC55S6 and 55S2x devides. The 0x3xxxx addresses are for the LPC55S1x and LPC55S0x devices. FLASH_REMAP is valid only for LPC55S1x/551x/LPC55S0x/LPC550x devices.

2.5.1 CMPA page

The CMPA page contains settings for signed image in secure boot configuration and PRINCE configuration registers if encrypted flash is needed.

Table 1. SECURE_BOOT_CFG

Bit	Symbol
29:0	Reserved
31:30	SEC_BOOT_EN

Value	Description
0b00	Plain image
0b01	Signed image
0b10	Signed image
0b11	Signed image

This page could be locked after manufacturer programs the page. This is done by writing the SHA 256 digest of the CMPA page into the SHA memory space of CMPA area.

2.5.2 CFPA page

CFPA Ping and Pong pages are used for selecting actual CFPA page with higher version number.

CFPA page is updated through Scratch Page using following steps:

1. Application code uses FLASH API to update the scratch page which remains outside the protected region.
2. Core is reset to make the page effective.
3. On subsequent boot, ROM checks if the scratch page is valid and has higher version. ROM erases the oldest of the two protected pages (ping pong pages) and copies the scratch page contents to the erased area.
4. The newest version of CFPA is loaded.

CFPA page contains setup for Image revocation, RoT keys revocation and debug authentication process setup.

Table 2. ROTKH_REVOKE

Bit	Symbol
1:0	RoTK0_EN
3:2	RoTK1_EN
5:4	RoTK2_EN
7:6	RoTK3_EN
31:8	Reserved

Value	Description
0b00	Invalid

Table continues on the next page...

Table continued from the previous page...

0b01	RoTKx Enabled
0b10	RoTKx Key revoked
0b11	RoTKx Key revoked

Table 3. IMAGE_KEY_REVOKE

Bit	Symbol
15:0	IMG revocation counter
31:16	Reserved

Only 17 revocation IDs are possible. (0x0=0b0, 0x1=0b1, 0x3=0b11, 0x7=0b111, 0xF=0b1111 … 0xFFFF=0b1111 1111 1111 1111)

3 Keys and certificates

Public key infrastructure (PKI) is set up to load signed image and certificates distribution.

All certificates are expected to be X.509 v3 certificates in DER format.

The Key file is included as a private key in PEM or DER format, which contains private key of the last certificate in the selected certificate chain (the certificates is used for signing of the image).

Keys could be 2048 bit (default) or 4096 bit.

3.1 Private key generation

Here is example how to prepare keys and certificates with openssl v 1.1.0.

```
openssl genrsa -out private_key_2048.pem 2048
openssl genrsa -out private_key_4096.pem 4096
```

3.2 Certificate signing request

After creation of private keys, certificate signing request (csr) is created.

```
openssl.exe req -new -key private_key_2048.pem -out
certificate_2048.csr -extensions v3_ca
```

3.3 Self-signed certificate

Self-signed certificate is CA:FALSE signed.

The serial number field (20 octets) in X.509 Image key certificate is used for revocation logic. The 4 left-most octet shall have two mandatory bytes 0x3c, 0xc3, then 16-bit revocation identifier (little endian byte order).

Example command to generate:

```
openssl.exe x509 -req -days 365 -in certificate_2048.csr -signkey private_key_2048.pem -sha256 -
outform der -out certificate_2048.der.crt -extfile x509_v3.ext -set_serial 0x3cc30000abababab
```

- Result in certificate: serial:3C:C3:**00:00**: AB: AB: AB: AB.
- Only 17 revocation IDs are possible. (0x0, 0x1, 0x3, 0x7, 0xF, 0x1F, 0x3F, 0x7F, 0xFF … 0xFFFF).

x509_v3.ext file should contain:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
```

3.4 Certificate chain sign

Root certificate is self signed by root key. From security aspects is better to create chain certificates for signing the image. Chain certificate is signed with root key. Root certificate is CA:TRUE and last chain certificate is CA:FALSE.

Chain certificate sign the image. If the chain private key is stolen so image with stolen certificates should be revoked. New chain certificate is signed with root key and image with new certificate loaded.

4 Non-secure boot on LPC55Sxx

This section describes how to create and load an unsigned plain CRC image into LPC55Sxx device. ELFTOSB and ELFTOSB-GUI are tools provided by NXP to create various boot images supported by LPC55Sxx. This tool is in *SDK/middleware/mcu-boot* or at www.nxp.com/mcuBoot. The CRC calculation and insertion of the computed CRC into the plain image is done using the ELFTOSB tool.

4.1 Unsigned plain CRC image preparation

The following image shows the ELFTOSB-GUI setup for UPC image.

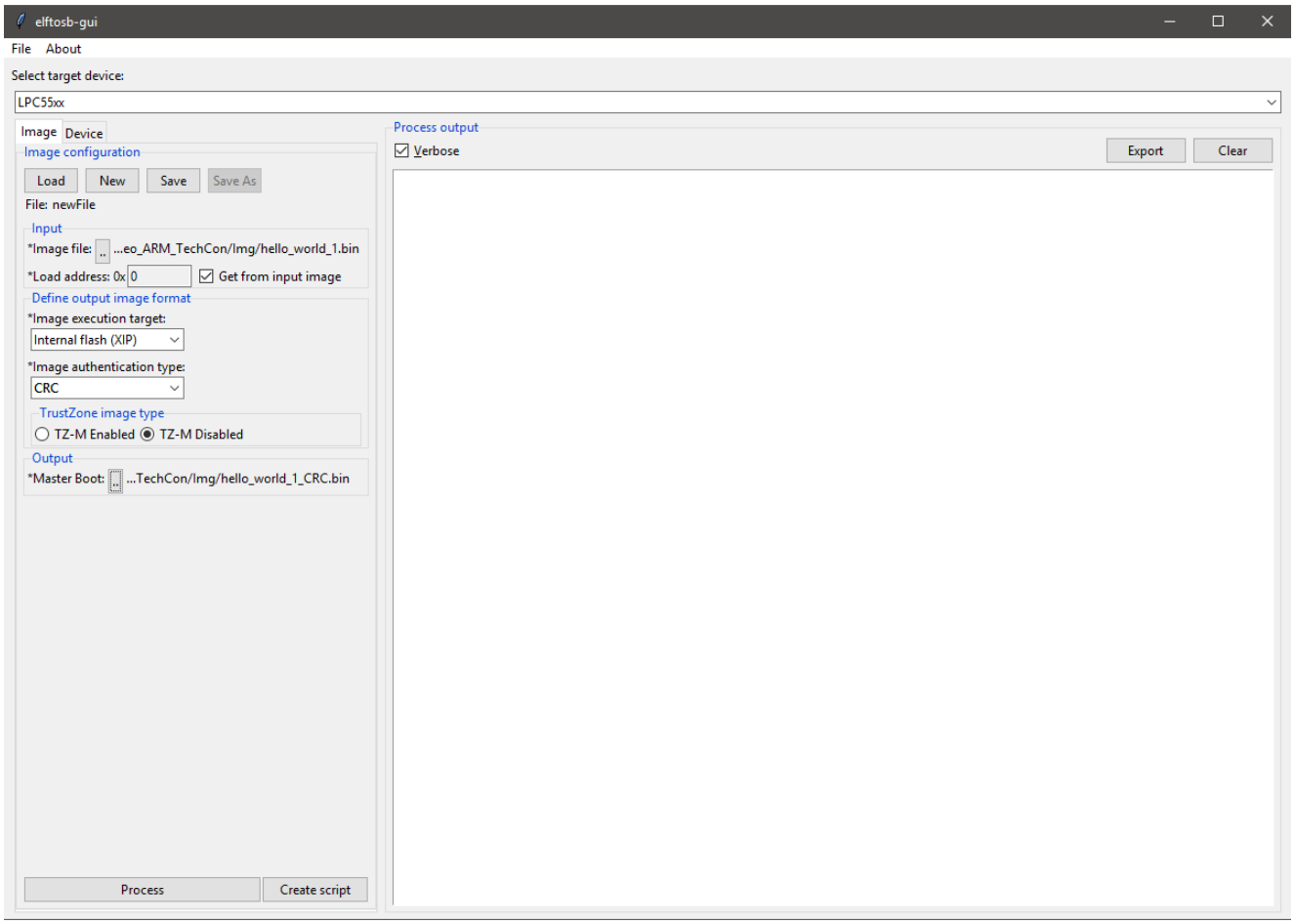

Figure 9. ELFTOSB-GUI setup for UPC image

1. Select LPC55Sxx device.
2. Load previous configuration or create a new configuration.
3. Select the plain binary image generated for LPC55Sxx device.
4. Select Internal flash and CRC check.

Configure TZ settings. In this example, TrustZone-M is disabled.

1. Select folder and name of output CRC binary file.
2. To create the unsigned plain CRC boot image, click the **Process** button.

After these steps, unsigned CRC binary file is saved in the selected output folder.

4.2 Loading unsigned plain CRC image

When unsigned CRC image is used as boot image, the CMPA or CFPA pages is configured.

The BLHOST utility is used for communication with the LPC55Sxx device through ISP mode.

The BLHOST utility is found in the SDK.

Here is basic ISP communication through UART with BLHOST tool. For more information, see the BLHOST manual.

1. Enter UART ISP mode

- Press and hold the ISP button (Switch S1) on the LPC55Sxx development board while pressing the RESET button (Switch S4) to enter ISP mode.

2. Test communication through BLHOST tool.

```
blhost -p COMxx get-property 1
```

If ping failure shows, it has wrong connection.

3. Write image into flash memory – address 0x0 is used for boot.

```
blhost -p COMxx write-memory 0 <path to the image(.bin)>
```

4. Reset device.

- Press reset pin or send command blhost -p COMxx reset.

App loaded into the Flash is running.

5 Secure boot on LPC55Sxx

WARNING

In ROM A0 after programming signed image there is no way to read or write flash memory through ISP. Configure the settings carefully. Only signed images with selected certificates are used.

The following section describes the steps for key provisioning, creating signed images and loading the signed images into the target. Tools used are blhost, elftosb command line, or elftosb-gui.

The figure below shows how to prepare, load, and execute signed images.

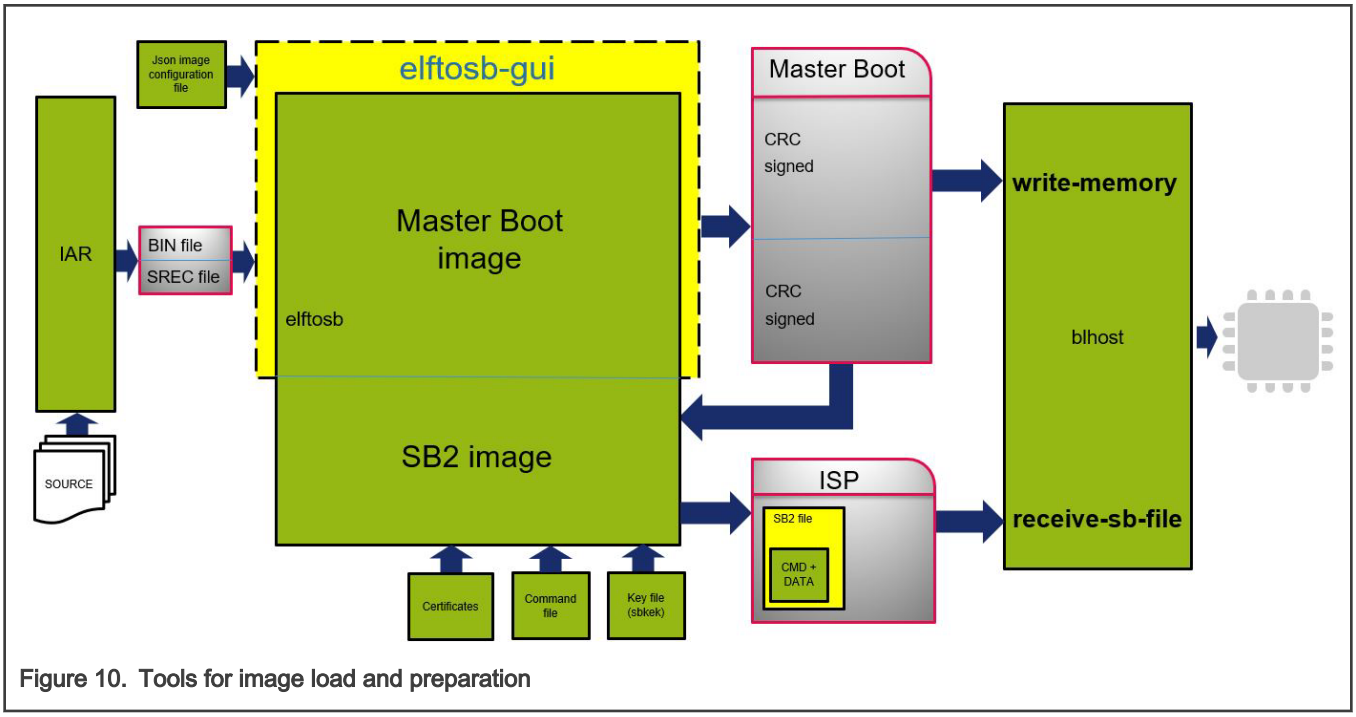

Figure 10. Tools for image load and preparation

5.1 Binary creation

Prepare binary file (.bin or .srec) from IDE.

Binary generated from IDE is plain image for unsecure boot.

5.2 Signed image preparation

The elftosb tool is used for creating the signed images from plain images.

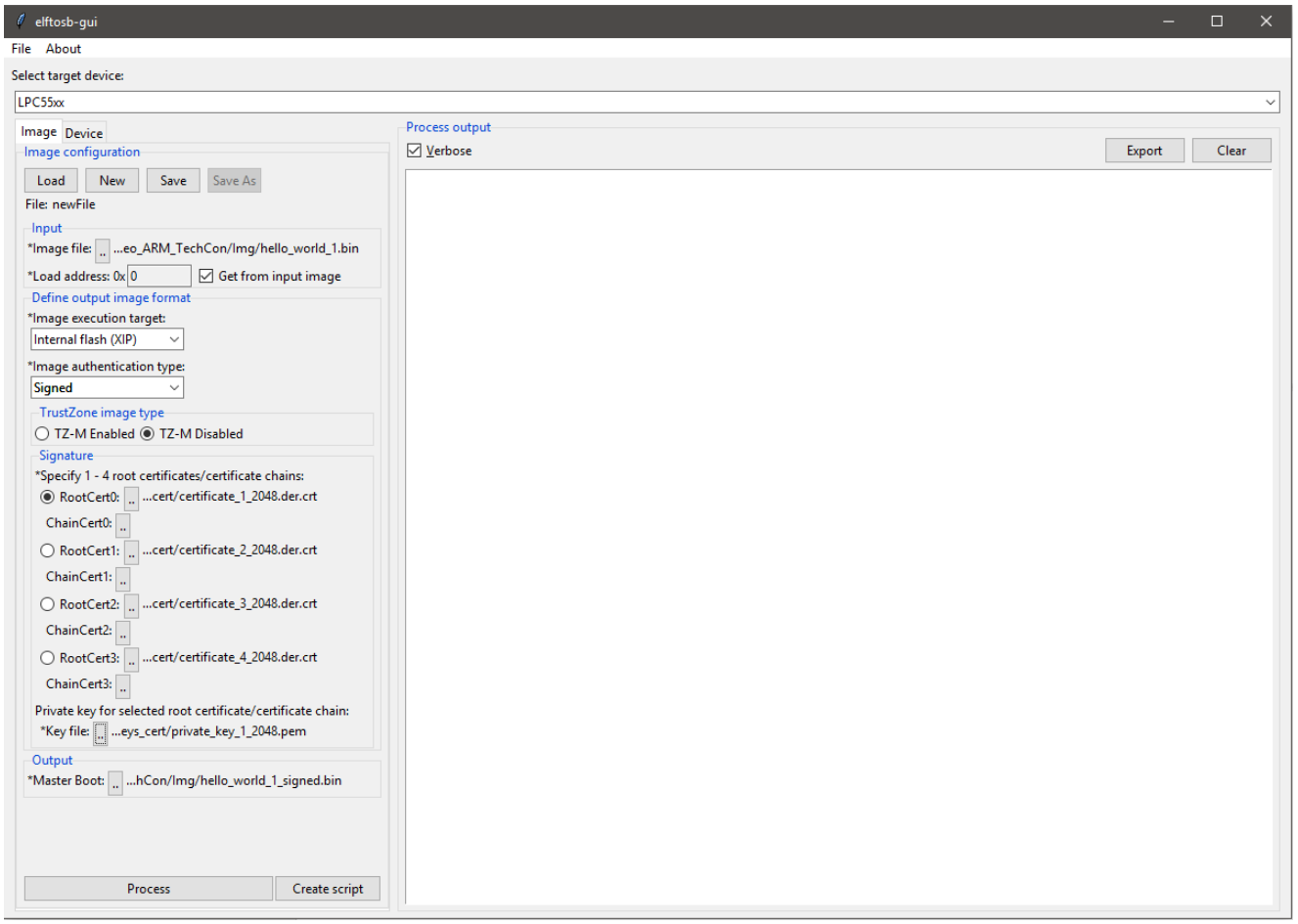

Figure 11. ELFTOSB-GUI setup for signed image

1. Select LPC55Sxx device.
2. Load previous configuration or create a new configuration.
3. Select the plain binary generated for LPC55Sxx device.
4. Select Internal flash and signed image.

Configure the TZ settings. In this example, TrustZone-M is disabled.

1. Select certificates or chain certificates.

Select private key for selected certificate (last certificate in chain).

1. Select folder and name of output binary.
2. To create signed binary image, click the **Process** button.

5.3 Loading signed image

The signed image is programmed into the device using BLHOST utility in ISP mode. Here is basic ISP communication through UART with BLHOST tool. For more information, see the BLHOST manual.

Enter UART ISP mode

- Press and hold the ISP button (Switch S1) on the LPC55Sxx development board while pressing the RESET button (Switch S4) to enter ISP mode.

1. Test communication through BLHOST tool

```
blhost -p COMxx get-property 1
```

   - If ping failure show it has got wrong connection

2. Write image into flash memory – address 0x0 is used for boot

```
blhost -p COMxx write-memory 0 <path to the signed image(.bin)>
```

After these steps, the signed image is loaded in the flash.

5.4 CFPA page preparation

By default, the CFPA page is cleared. There are registers related to secure boot whichis set up.

ROTKH_REVOKE field at CFPA page has to be set up to accept signed images with created certificates.

1. Enter ISP mode
   - Press and hold the ISP button (Switch S1) on the LPC55Sxx development board while pressing the RESET button (Switch S4) to enter ISP mode.
2. Test communication through BLHOST tool.

```
blhost -p COMxx get-property 1
```

   - If ping failure show it has got wrong connection.
3. Prepare CFPA page in .bin file (example with RoT key 0-3 enabled is attached). CFPA update checker has strict rules.

   Version > existing Version secureFwVersion >= existing secureFwVersion nsFwVersion >= existing nsFwVersion vendorUsage >=existing vendorUsage imageKeyRevoke >= existing imageKeyRevoke imagekeyRevoke are treated like OTP, the version increasing flow, (0, 1, 3, 7, etc.) rotkhRevoke >= existing rotkhRevoke rotkhRevoke are treated like OTP, the version increasing flow, (0, 1, 3, 7, etc.)

   If this is not used, the Scratch page is not accepted.
4. Write prepared CFPA page into flash memory – address 0x9DE00.

```
blhost -p COMxx write-memory 0x9DE00 <path to the CFPA(.bin)>
```

WARNING

In ROM A0 after programming signed image there is no way to read or write CFPA, CMPA pages through ISP. Carefully configure the settings. Only signed images with selected certificates are used.

5.5 CMPA page preparation

The CMPA page is configured to boot the signed image .

NOTE

In ROM A0 after programming signed image there is no way to read or write CFPA, CMPA pages through ISP. Carefully configure these settings. Only signed images with selected certificates are used.

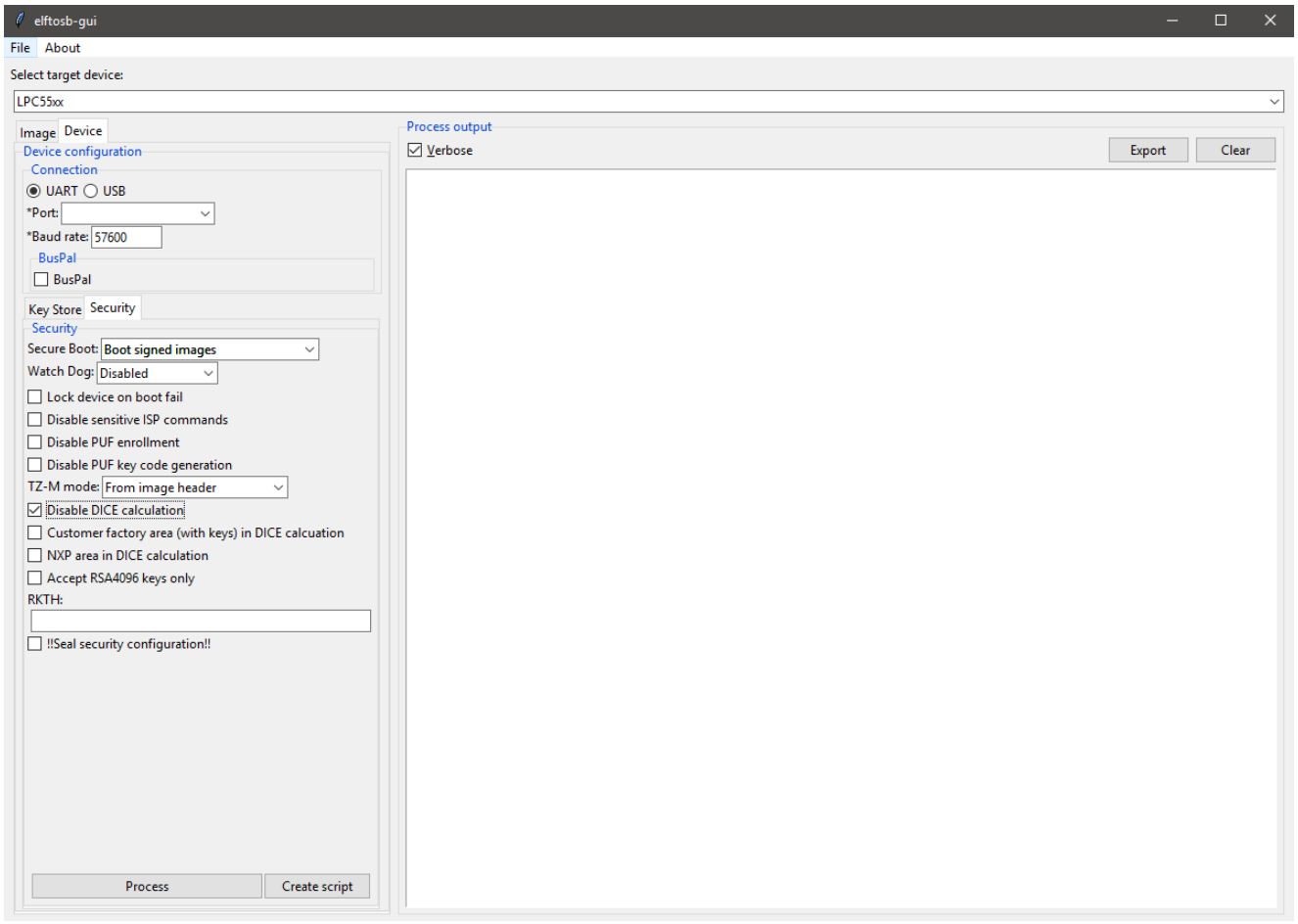

Figure 12. ELFTOSB-GUI setup for CMPA

1. Select LPC55xx device.
2. Select the **Device** tab.
3. Configure connection (ISP mode). In this example UART ISP is selected with COM port and baud rate for communication with the target.
4. In the **Security** section, select the Secure Boot as "Boot Signed Images"
5. Disable DICE calculation (optionally select only RSA4096 keys).
6. Insert RKTH (Generated from certificates during signing process in previous section).
7. Click „Process“ button and CMPA page will be setup (Device in ISP mode).
8. To reset device, press the reset pin or send command blhost -p COMxx reset.
9. The signed image loaded into the flash memory of target starts executing.
10. Until the CMPA is not sealed (HASH of CMPA written), you can change the configuration. The empty CMPA is attached. Use "blhost -p COMxx write-memory 0x9E400 CMPA_empty.bin". This is not applicable to the LPC55S69 0A silicon.

NOTE

After this setup only signed images are loaded into the LPC55Sxx device.

5.6 Signed image update capsule SB2

This section describes the steps to load an updated image into LPC55Sxx.

The new signed image is created as described in the section Signed Image Preparation.

After successful creation of signed image, follow the below steps for creating SB2 file.

SB2 file is symmetrically encrypted file which contains new image and address position. This SB2 file is loaded into device and boot ROM code updates the image.

Information about elftosb command-line tool and SB2 files which are used here can be found in the user manual for this tool.

5.6.1 Secure binary key creation and loading

SB2 file is symetrically encrypted. For decryption of the file the key has to be loaded into device. The key size for SB2 file is 256 bits. During boot, the SB key is used with AES to decrypt the SB2 file. The SB key input for ELFTOSB-GUI is .txt file with plain text key. This key is loaded into PUF key store of LPC55Sxx.

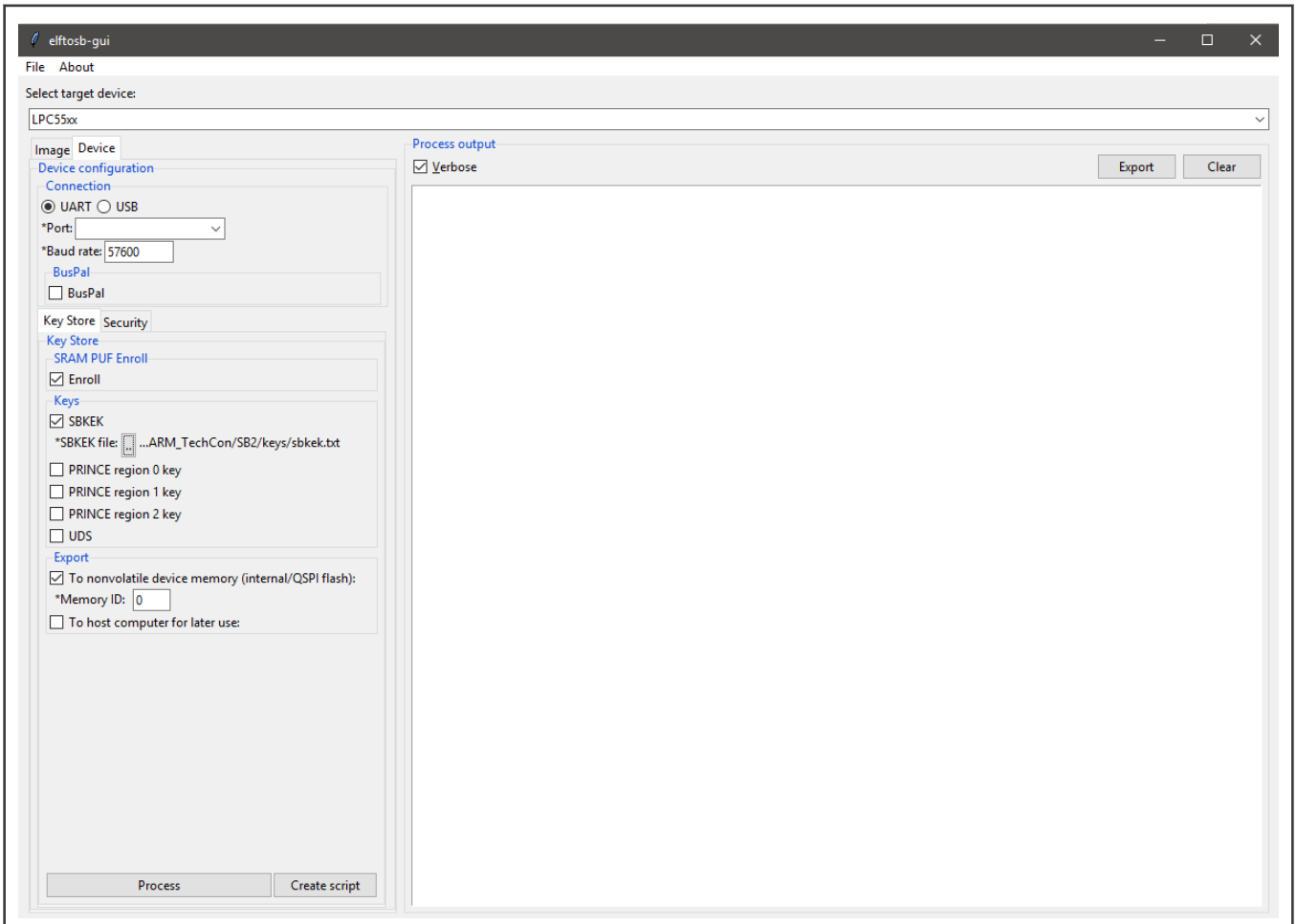

Figure 13. ELFTOSB-GUI setup for SBKEK load

SB key text file example: AC801E99BD3592E419B757EADC0985B3D3D0BC0FDB6B057AA88252204C2DA732

5.6.2 SB2 file creation

The SB2 container is described in the elftosb user's guide. The SB configuration file contains the configuration commands that are processed after the SB2 file is loaded in the device. The image location is stated in the "sources" section of the *.bd* file. The SB key in the text file is used for encryption with the elftosb command line tool.

The 0A version of the LPC55S6xx silicon supports version 2.0 of the SB image format.

The 1B version of the LPC55S6xx silicon supports version 2.1 of the SB image format.

The main difference between version 2.0 and version 2.1 is the usage of a digital signature.

SB 2.0 is **encrypted** and SB 2.1 is **encrypted + signed**.

Example of use (Encrypted SB 2.0):

```
elftosb -f lpc55xx -k "sbkek.txt" -c "commandFile.bd" -o "output.sb2" "input.bin"
```

where:

- -f = LPC55xx family.
- -k = path to the KEK file (SBKEK).
- -c = path to the command file to be processed:

```
options
{
    flags = 0x4; // 0x8 encrypted + signed, 0x4 encrypted
    buildNumber = 0x1;
    productVersion = "1.00.00";
    componentVersion = "1.00.00";
}
sources
{
    inputFile = extern(0);
}
section (0)
{
    erase 0x0..0x40000;
    load inputFile > 0x0;
}
```

- -o = path to the output file.
- files... = path to the files (usually image files) that replace the placeholders defined in the command file. The paths can be hardcoded in the command file and then not inserted as the input.

Example of use (Encrypted + Signed SB 2.1):

1 root key

```
elftosb.exe -f lpc55xx -k "sbkek.txt" -c "commandFile.bd" -o "output.sb2" -s
"selfsign_privatekey_rsa2048.pem" -S
"selfsign_v3.der.crt" -R "selfsign_v3.der.crt" -h "RKTH.bin" "input.bin"
```

4 root keys

```
elftosb.exe -f lpc55xx -k "sbkek.txt" -c "commandFile.bd" -o "output.sb2" -s private_key_1_2048.pem
-S certificate_1_2048.der.crt -R certificate_1_2048.der.crt -R
certificate_2_2048.der.crt -R certificate_3_2048.der.crt -R certificate_4_2048.der.crt -h "RHKT.bin"
"input.bin"
```

where:

- -f = LPC55xx family.
- -k = path to the KEK file (SBKEK).

- c = path to the command file to be processed:

```
options {
    flags = 0x8; // for sb2.1 use only 0x8 encrypted + signed
    buildNumber = 0x1;
    productVersion = "1.00.00";
    componentVersion = "1.00.00";
    secureBinaryVersion = "2.1";
}
sources
{
    inputFile = extern(0);
}
section (0)
{
    erase 0x0..0x40000;
    load inputFile > 0x0;
}
```

- -o = path to the output file.
- -s = path to the private key of the certificate used for signing.
- -S = path(s) to the certificates in the certificate chain. Each certificate in the chain must be specified with a new -S switch in the same order as the chain was created (root certificate first).
- -R = path(s) to the root certificate(s). From one to four root certificates can be specified. Each root certificate must be specified with a new -R switch and one root certificate must be the first certificate specified by the -S switch.
- -h = path and name of the output binary file generated by elftosb, which contains the hash of all root certificates' hashes (RKTH) that must be uploaded to the device register.
- files... = path to the files (usually image files), which replaces the placeholders defined in the command file. Paths can be hardcoded in the command file and then not inserted as the input.

5.6.3 SB2 file load

The SB2.0 file, created with the updated binary image, can be loaded into the device through the ISP command handler using the “receive-sb-file“ command.

```
blhost -p COMxx receive-sb-file <path to the secured binary(.sb2)>
```

The SB2.1 file, created with the updated binary image, can be loaded into the device through the ISP command handler using the “receive-sb-file“ command. However, before sending the SB2.1 file into the device, the RKTH must be already in the CMPA and the RoT keys must be enabled in the "ROTKH_REVOKE" field at the CFPA page address 0x9DE18 (see chapter CFPA page preparation).

```
blhost -p COMxx receive-sb-file <path to the secured binary(.sb2)>
```

After successfully loading the SB2 file, it is executed, as configured in the SB configuration file (**.bd* file). The above figure shows an example of the SB configuration file. When the file is executed, the internal flash address (from 0x0 to 0x40000) is erased. After the flash erase operation, the image mentioned in the sources parameter is loaded to address 0x0.

Reset the device after these operations. The updated image loaded into the internal flash starts to execute.

5.7 Flash encryption using PRINCE

NOTE

In ROM A0 after ISP command “configure-memory“, LPC55S69 device is sealed and settings cannot be changed in the PFR CMPA page. FLASH content is encrypted with keys from PUF. Carefully configure these settings.

PRINCE engine could be used for real-time encryption and decryption of flash content of program. This application note, describes using PRINCE for encrypting/decrypting application code. During flash programming, PRINCE is configured to program the image in encrypted format using a key and Initialization vector. On boot, PRINCE decrypts the image on-the-fly and executes the image from internal flash.

The following sections describe the tools required to perform key provisioning and programming application image using PRINCE.

5.7.1 PUF key store setup

The keys used for PRINCE encryption/decryption are generated in the device using on-chip SRAM PUF.

The keys are delivered through internal hardware bus which is not software accessible. This hardware bus is configured from PUF engine.

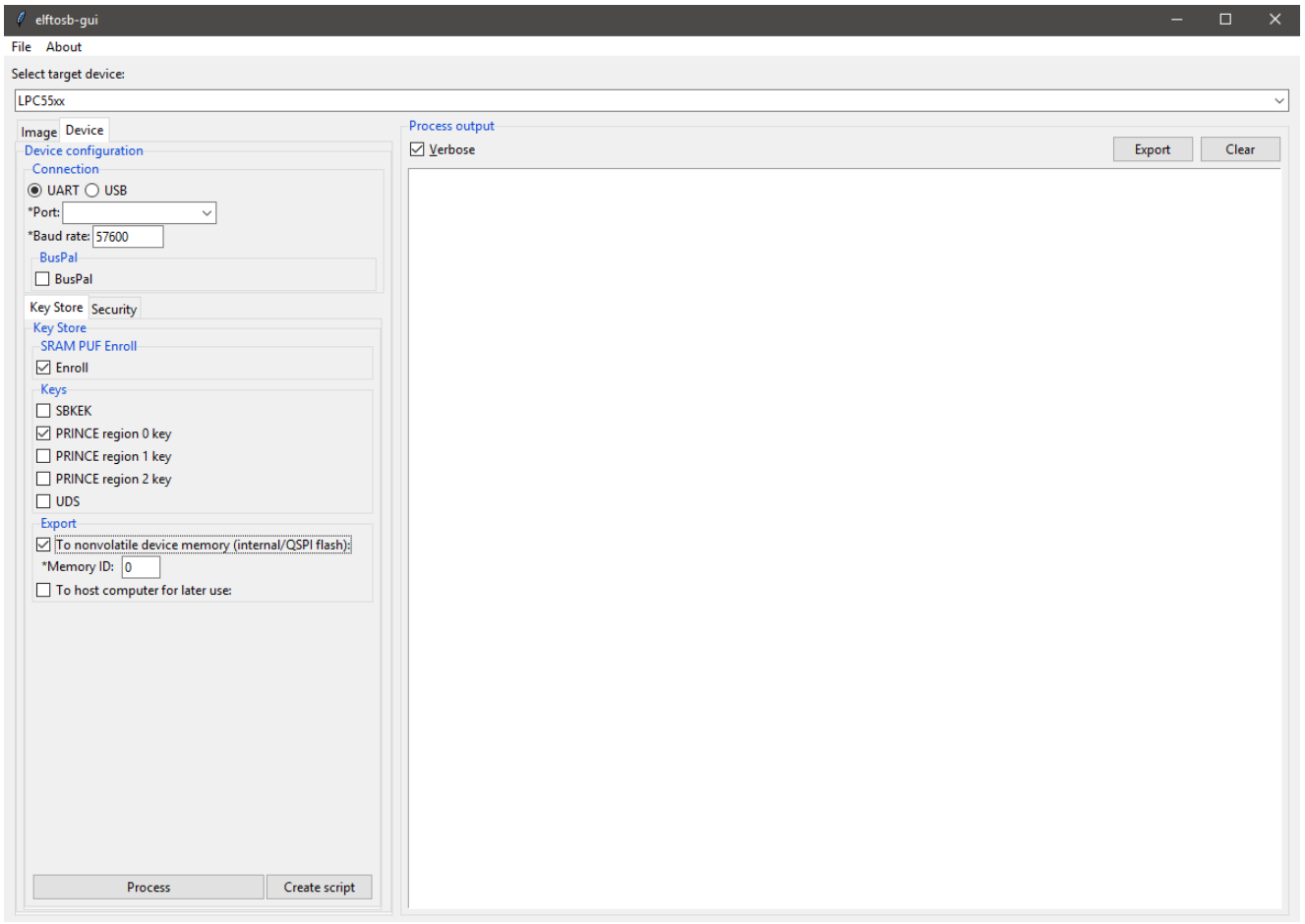

Figure 14. ELFTOSB-GUI setup for PUF Key Store

1. Select LPC55xx device.
2. Select Device tab.
3. Configure connection (ISP mode). In this example UART ISP is selected with the desired COM port and baudrate for communication.

4. Configure PUF Enroll (Activation code creation).
5. Select PRINCE PUF keys for setup. In the example shown, PRINCE region 0 is used for encryption. Hence PRINCE key 0 is selected.
6. Select that the key encryption keys (KEK) are saved into flash memory (PFR Key store).
7. Click “Process” button and Key store will be setup in the internal flash of LPC55Sxx (Device in ISP mode).
8. Reset the device to apply

NOTE

After this configuration key encryption keys are saved into PFR key store.

5.7.2 PRINCE configuration with blhost

For PRINCE encryption and decryption the regions and sub-regions for the crypto operation needs to be configured. This can be done with ISP command “configure-memory”. This command have to be called with following data structure.

Table 4. Structure for configure-memory command

Offset	Size	Description
0	4	PRINCE Configuration
4	8	PRINCE Region info

Table 5. PRINCE configuration register for configure-memory command

Bit	Symbol
1:0	0x00 – PRINCE Region 0 0x01 – PRINCE Region 1 0x10 – PRINCE Region 2
25:2	Reserved
31:8	0x50 ('P') – Configure PRINCE

Table 6. PRINCE Region info register for configure-memory command

Bit	Symbol
31:0	PRINCE Region X Start
63:32	PRINCE Region X size

Load structure into the appropriate not used RAM memory and call “configure-memory” command with this structure.

1. Region selection (Region 0 in this example).

```
blhost.exe -p COMxx fill-memory 0x20034000 4 0x50000000
```

2. Start address of encrypted area (Address 0x0 in this example)

```
blhost.exe -p COMxx fill-memory 0x20034004 4 0
```

3. Length of the encrypted area (0x40000 in this example)

```
blhost.exe -p COMxx fill-memory 0x20034008 4 0x40000
```

4. Call configure-memory with prepared structure in RAM

```
blhost.exe -p COMxx configure-memory 0 0x20034000
```

Following this command, PRINCE is configured for flash encryption.

WARNING

!! DO NOT reset the board and continue with commands for loading the image !!

5.7.3 Upload image

Due to several limitations of the "prince erase checker" and "prince flash write checker" implementations in the boot ROM, it is recommend to properly check the flash-erase-region and write-memory command parameters to strictly cover the whole encrypted area. Using the MCUXpresso SDK Prince driver is the preferred way for the Prince region configuration, erase, and write operations.

5.8 Conclusion

This application note describes the basic function of secure boot of LPC55Sxx device and how to run secure boot and how to configure secure boot.

This application note also describes basic configuration of PRINCE engine for on the fly encryption and decryption.

6 Revision history

The following table lists the changes made to this document since the initial release.

Table 7. Revision history

Revision number	Date	Substantive changes
0	02/2019	Initial release
1	03/2020	Updated sections SB2 file creation, SB2 file load, and Upload image.
2	11/2020	Added info about LPC55S0x support

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 11/2020

Document identifier: AN12283

arm