NXP Semiconductors

User's Guide

Document Number: MCUBOOTELFTOSBGUIUG

Rev 0, 10/2018

elftosb-gui User's Guide

Contents

Chapter 1
Overview

The elftosb-gui is GUI tool with a main focus to help the user prepare a secure application image, as well as other useful security operation specific to target MCU platform. Elftosb-gui tool provides intuitive graphical interface on top of elftosb and blhost command-line applications and it guides user in preparation of secure boot images required by ROM bootloader.

Supported operation system are: Windows® OS, Linux® OS, and Apple Mac® OS. The following chapters provide information for how to use elftosb-gui, as well as what you can do using elftosb-gui for supported devices.

Chapter 2
How to start

Unzip the elftosb-gui archive and start the application by executing one of the prepared scripts based on your operating system:

- elftosb-gui(darwin).sh
- elftosb-gui(linux).sh
- elftosb-gui(windows).cmd

After starting, the main application window appears:

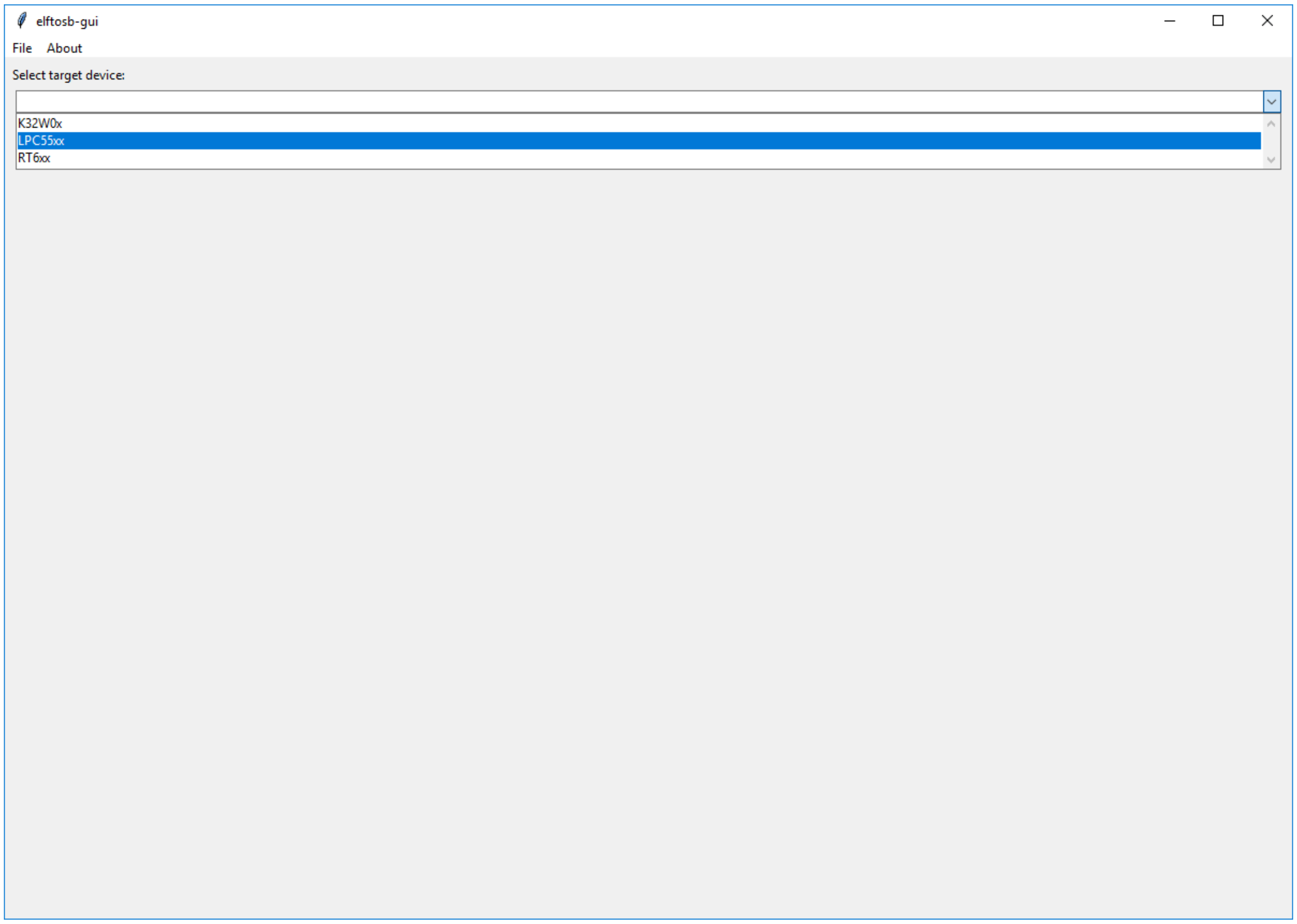


Figure 1. Device family selection

Continue with the selection of targeting the MCU family, which opens the family-specific user interface.

Chapter 3 K32W0x MCU platform

For the k32w0x MCU platform, the elftosb-gui offers a wizard for creating the master boot image.

3.1 Master boot image generation

The elftosb-gui allows user to create, modify or use an image configuration file. The tool can open an existing image configuration file, or, create a new image configuration file and save it for later use. The image configuration file is a json text file and it is an input required by elftosb command line tool for master boot image generation. The elftosb command line tool is available with the elftosb-gui, thus, user can directly generate the master boot image from the GUI. Alternatively, if the user has the image configuration file, the master boot image can be generated from command line (by calling elftosb) without involving the GUI.

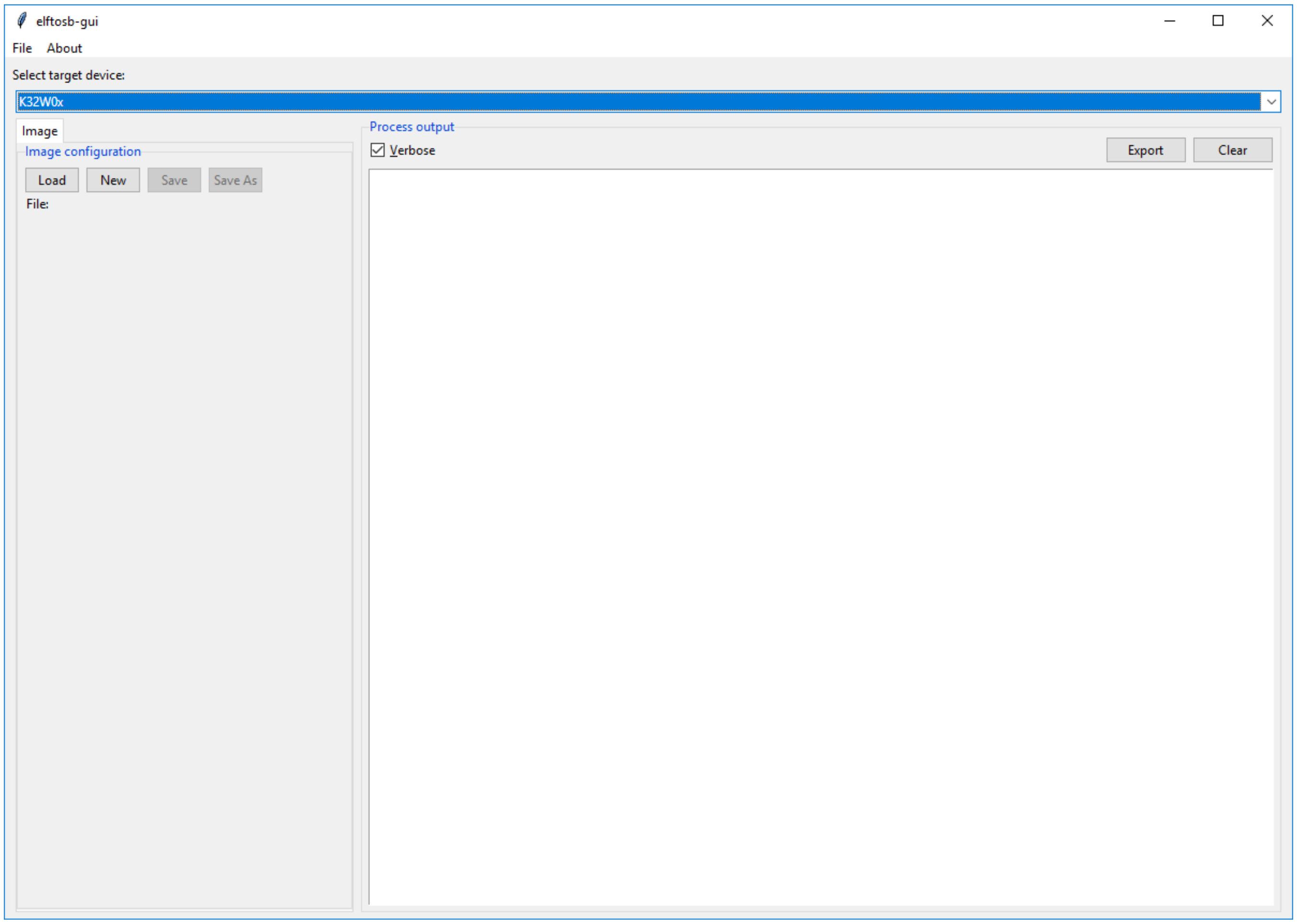


Figure 2. k32w0x family main layout

Click the "Load" button to open an existing configuration, the "New" button for creating a new configuration, and "Save" or "Save As" for saving the created or modified configuration.

During creation of the new image configuration, the user will be guided by elftosb-gui to successfully create the correct configuration.

If the configuration is finished, click the "Process" button to call the elftosb application to parse the configuration and create the output image file. In case of missing configuration inputs, elftosb-gui marks problematic fields in red.

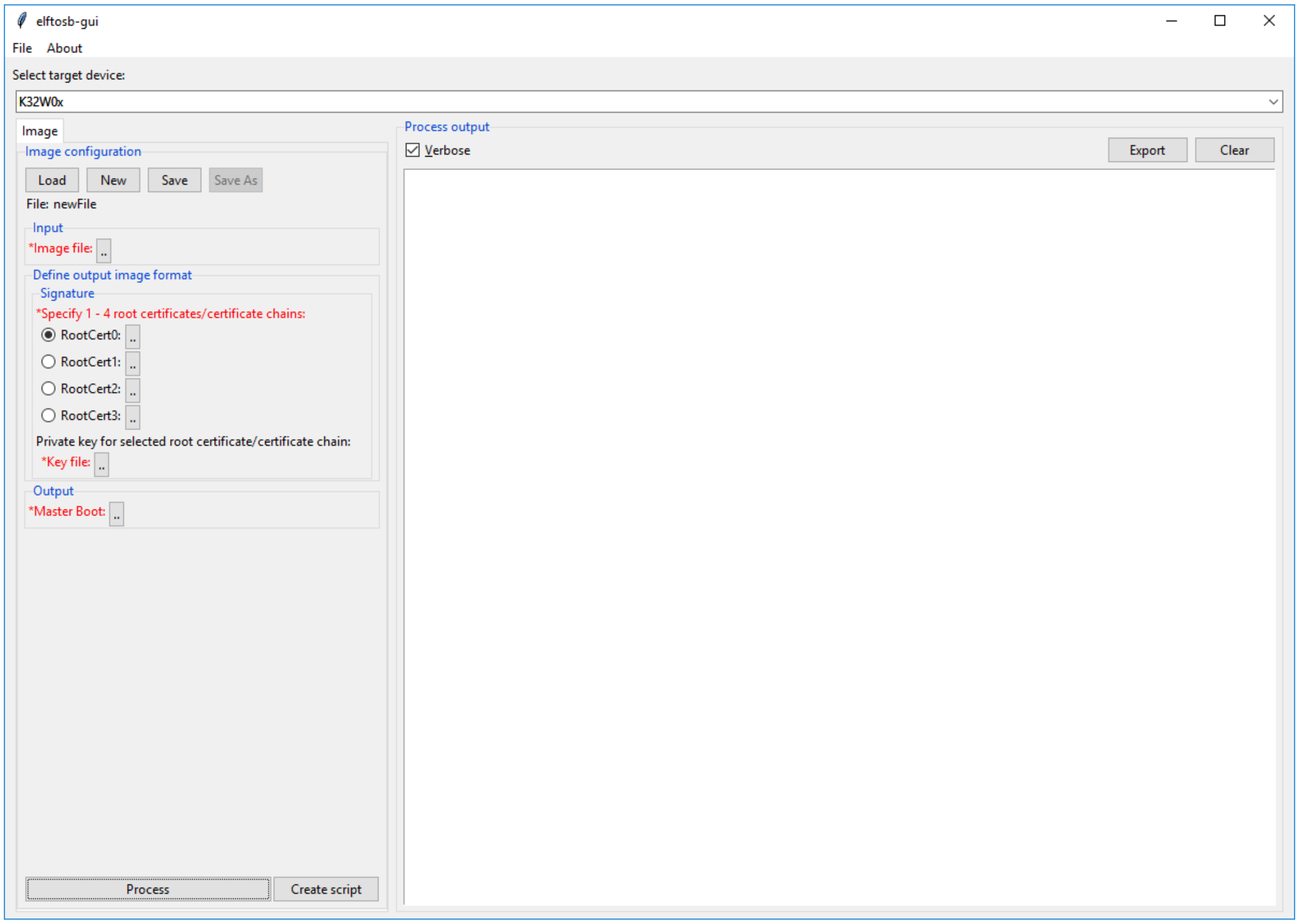
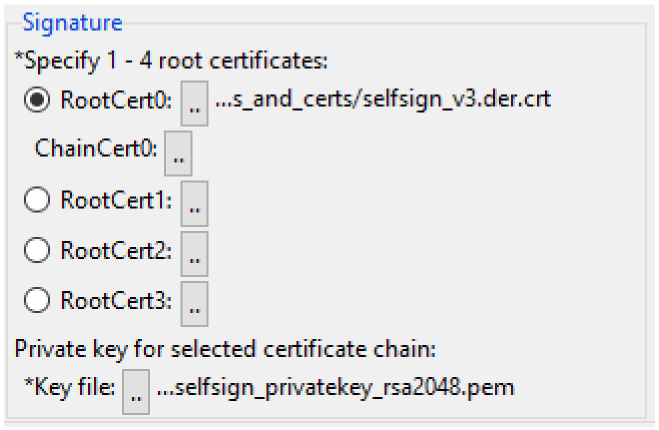


Figure 3. Missing configuration inputs for k32w0x

To sign an application image with the tool, user shall include 1 to 4 root certificates. The root certificate can be alone in certificate chain, but in this case the root certificate must be a self-signed nonCA certificate. The elftosb-gui supports certificate chains with two certificates (root certificate and image signing certificate). In the case of two certificates in a chain, the root certificate must be self-signed CA, and the second signed by a root certificate and is nonCA. For creating bigger certificate chains, it is necessary to manually update the json image configuration file. For details, see Appendix D in the *Kinetis Elftosb User's Guide* (document MBOOTELFTOSBUG).



Signature

*Specify 1 - 4 root certificates:

● RootCert0: .. ...s_and_certs/selfsign_v3.der.crt

ChainCert0: ..

○ RootCert1: ..

○ RootCert2: ..

○ RootCert3: ..

Private key for selected certificate chain:

*Key file: .. ...selfsign_privatekey_rsa2048.pem

Figure 4. Image signature configuration

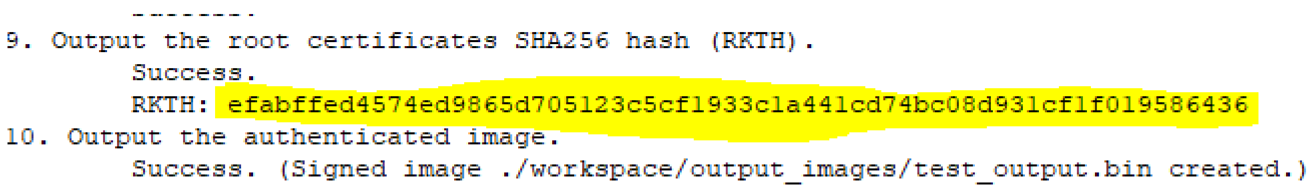
All certificates are expected to be X.509 v3 certificates in DER format.

One of the specified certificate chains must be selected by the "Radio" button next to the RootCert specification. The selected certificate chain is used for signature of an image, the other certificate chains are stored for later use.

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

For signed images, elftosb-gui shows (in the output window) the RKTH value generated during the signature process by the elftosb command line tool. RKTH is the hash value of hashes of provided root certificates. This value must be programmed to the target device. The programming can be done by the blhost command line application. For more details, see the *blhost User's Guide* (document MCUBLHOSTUG).

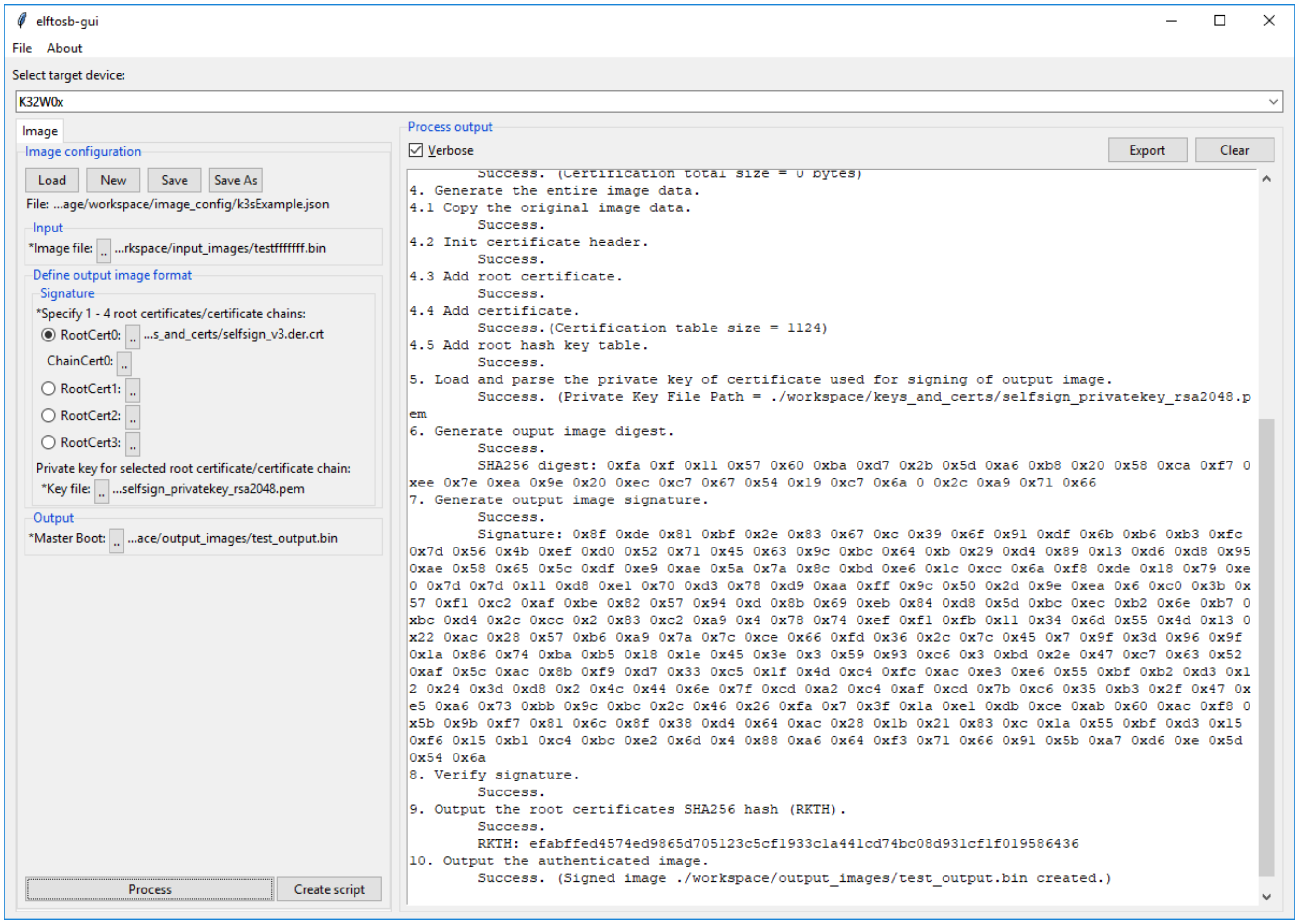
If the new image is produced with different root certificates, the new image will not be accepted by the device due to different RKTH values.



```
        --------.
9. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: efabffed4574ed9865d705123c5cf1933c1a441cd74bc08d931cf1f019586436
10. Output the authenticated image.
        Success. (Signed image ./workspace/output_images/test_output.bin created.)
```

Figure 5. RKTH value in the elftosb-gui output window

Output details can be limited by unchecking the 'Verbose' option in output area. The "Clear" button can also be used to remove old output between runs.



Select target device: K32W0x

Image configuration

File: ...age/workspace/image_config/k3sExample.json

Input

*Image file: ...rkspace/input_images/testfffffff.bin

Define output image format

Signature

*Specify 1 - 4 root certificates/certificate chains:

(●) RootCert0: ...s_and_certs/selfsign_v3.der.crt

ChainCert0:

( ) RootCert1:

( ) RootCert2:

( ) RootCert3:

Private key for selected root certificate/certificate chain:

*Key file: ...selfsign_privatekey_rsa2048.pem

Output

*Master Boot: ...ace/output_images/test_output.bin

Process output

[✓] Verbose

```
        Success. (Certification total size = 0 bytes)
4. Generate the entire image data.
4.1 Copy the original image data.
        Success.
4.2 Init certificate header.
        Success.
4.3 Add root certificate.
        Success.
4.4 Add certificate.
        Success.(Certification table size = 1124)
4.5 Add root hash key table.
        Success.
5. Load and parse the private key of certificate used for signing of output image.
        Success. (Private Key File Path = ./workspace/keys_and_certs/selfsign_privatekey_rsa2048.p
em
6. Generate ouput image digest.
        Success.
        SHA256 digest: 0xfa 0xf 0x11 0x57 0x60 0xba 0xd7 0x2b 0x5d 0xa6 0xb8 0x20 0x58 0xca 0xf7 0
xee 0x7e 0xea 0x9e 0x20 0xec 0xc7 0x67 0x54 0x19 0xc7 0x6a 0 0x2c 0xa9 0x71 0x66
7. Generate output image signature.
        Success.
        Signature: 0x8f 0xde 0x81 0xbf 0x2e 0x83 0x67 0xc 0x39 0x6f 0x91 0xdf 0x6b 0xb6 0xb3 0xfc
0x7d 0x56 0x4b 0xef 0xd0 0x52 0x71 0x45 0x63 0x9c 0xbc 0x64 0xb 0x29 0xd4 0x89 0x13 0xd6 0xd8 0x95
0xae 0x58 0x65 0x5c 0xdf 0xe9 0xae 0x5a 0x7a 0x8c 0xbd 0xe6 0x1c 0xcc 0x6a 0xf8 0xde 0x18 0x79 0xe
0 0x7d 0x7d 0x11 0xd8 0xe1 0x70 0xd3 0x78 0xd9 0xaa 0xff 0x9c 0x50 0x2d 0x9e 0xea 0x6 0xc0 0x3b 0x
57 0xf1 0xc2 0xaf 0xbe 0x82 0x57 0x94 0xd 0x8b 0x69 0xeb 0x84 0xd8 0x5d 0xbc 0xec 0xb2 0x6e 0xb7 0
xbc 0xd4 0x2c 0xcc 0x2 0x83 0xc2 0xa9 0x4 0x78 0x74 0xef 0xf1 0xfb 0x11 0x34 0x6d 0x55 0x4d 0x13 0
x22 0xac 0x28 0x57 0xb6 0xa9 0x7a 0x7c 0xce 0x66 0xfd 0x36 0x2c 0x7c 0x45 0x7 0x9f 0x3d 0x96 0x9f
0x1a 0x86 0x74 0xba 0xb5 0x18 0x1e 0x45 0x3e 0x3 0x59 0x93 0xc6 0x3 0xbd 0x2e 0x47 0xc7 0x63 0x52
0xaf 0x5c 0xac 0x8b 0xf9 0xd7 0x33 0xc5 0x1f 0x4d 0xc4 0xfc 0xac 0xe3 0xe6 0x55 0xbf 0xb2 0xd3 0x1
2 0x24 0x3d 0xd8 0x2 0x4c 0x44 0x6e 0x7f 0xcd 0xa2 0xc4 0xaf 0xcd 0x7b 0xc6 0x35 0xb3 0x2f 0x47 0x
e5 0xa6 0x73 0xbb 0x9c 0xbc 0x2c 0x46 0x26 0xfa 0x7 0x3f 0x1a 0xe1 0xdb 0xce 0xab 0x60 0xac 0xf8 0
x5b 0x9b 0xf7 0x81 0x6c 0x8f 0x38 0xd4 0x64 0xac 0x28 0x1b 0x21 0x83 0xc 0x1a 0x55 0xbf 0xd3 0x15
0xf6 0x15 0xb1 0xc4 0xbc 0xe2 0x6d 0x4 0x88 0xa6 0x64 0xf3 0x71 0x66 0x91 0x5b 0xa7 0xd6 0xe 0x5d
0x54 0x6a
8. Verify signature.
        Success.
9. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: efabffed4574ed9865d705123c5cf1933c1a441cd74bc08d931cf1f019586436
10. Output the authenticated image.
        Success. (Signed image ./workspace/output_images/test_output.bin created.)
```

Figure 6. Successfully created image

3.2 Create script

The elftosb-gui tool offers possibility to save output as a command line script for later use directly by command line elftosb command line tool.

Use the "Create Script" button, next to "Process" button. If some mandatory input is missing, the field is marked in red.

The user will be prompted to specify the output file (script). The script is generated for the actual operating system (Windows, Linux, MAC).

The script can be modified and used, for example, in process automation.

Chapter 4
LPC55xx MCU platform

For the lpc55xx MCU platform, the elftosb-gui offers a wizard for creating the master boot image, key store initialization, and security related configuration of the device.

4.1 Master boot image generation

The elftosb-gui allows user to create, modify or use an image configuration file. The tool can open an existing image configuration file, or, create a new image configuration file and save it for later use. The image configuration file is a json text file and it is an input required by elftosb command line tool for master boot image generation. The elftosb command line tool is available with the elftosb-gui, thus, user can directly generate the master boot image from the GUI. Alternatively, if the user has the image configuration file, the master boot image can be generated from command line (by calling elftosb) without involving the GUI.

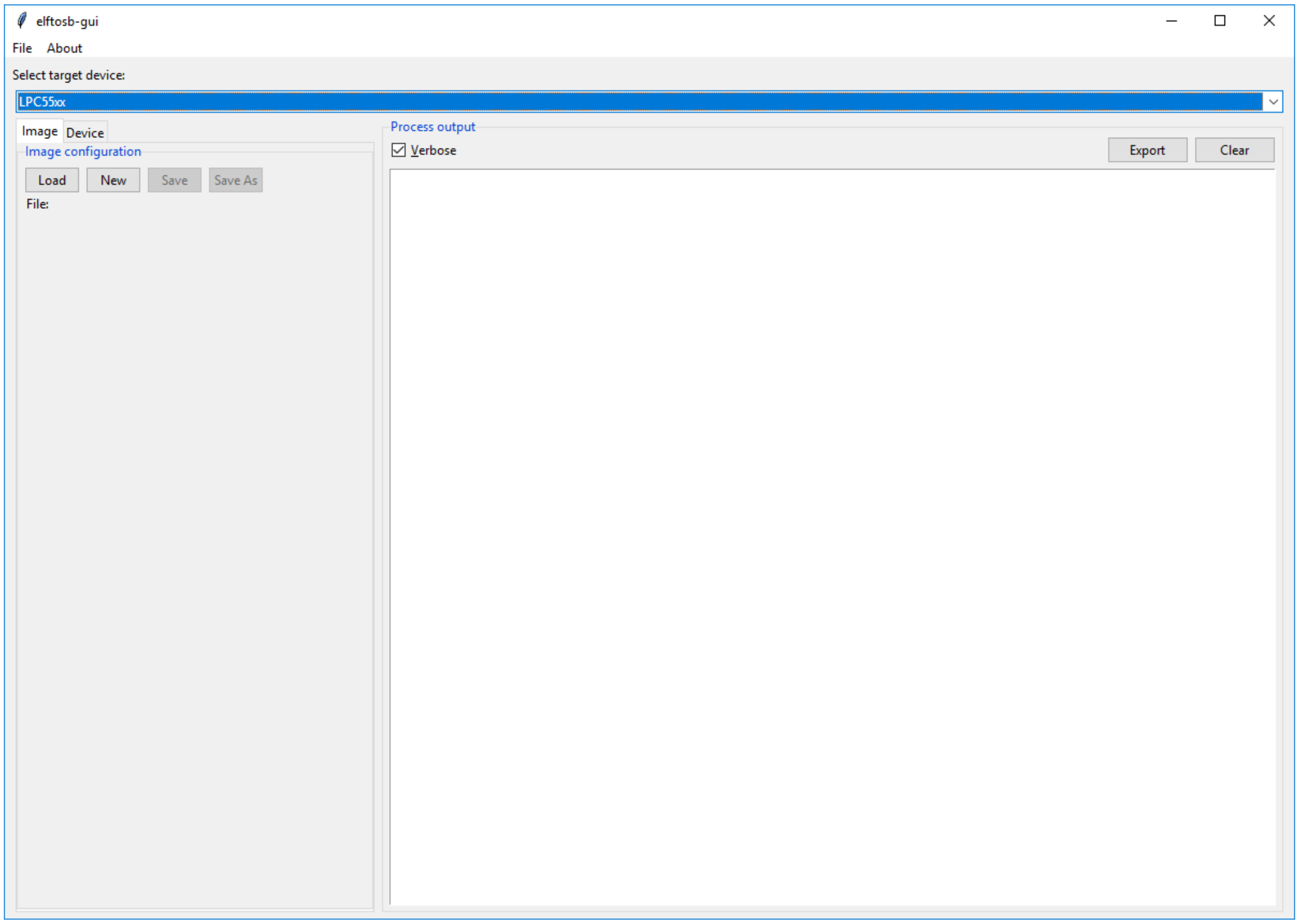


Figure 7. lpc55xx family main layout

Use the "Load" button to open an existing configuration, "New" for creating a new configuration, or "Save" or "Save As" for saving the created or modified configuration.

During creation of a new image configuration, the user is guided by elftosb-gui to successfully create the correct configuration.

If the configuration is finished, click the "Process" button to call the elftosb application to parse the configuration and create the output image file. In case of missing configuration inputs, elftosb-gui marks the problematic fields in red.

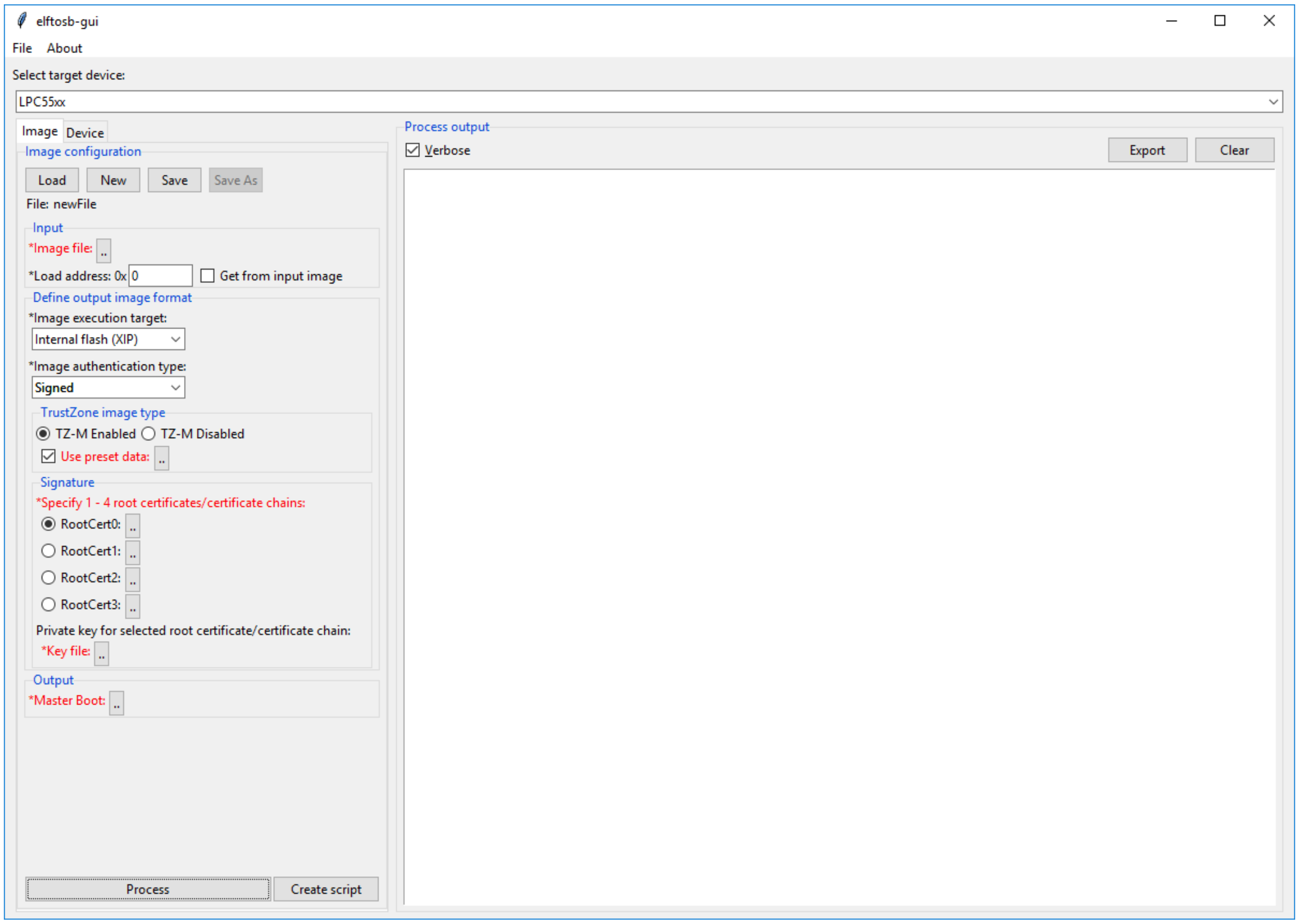


Figure 8. Missing configuration inputs for lpc55xx

The TrustZone-M preset configuration can be included as a binary file, which is directly copied into an output image. Alternatively, the TrustZone-M preset configuration can be specified as Trustzone-M preset configuration file, in this case, the configuration file is processed during the image creation process to produce the TZ-M preset binary data for the output image. For more information about the TrustZone-M preset configuration file, see Appendix E in the *Kinetis Elftosb User's Guide* (document MBOOTELFTOSBUG).

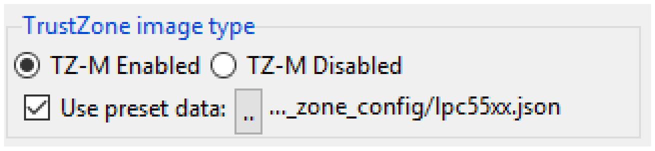
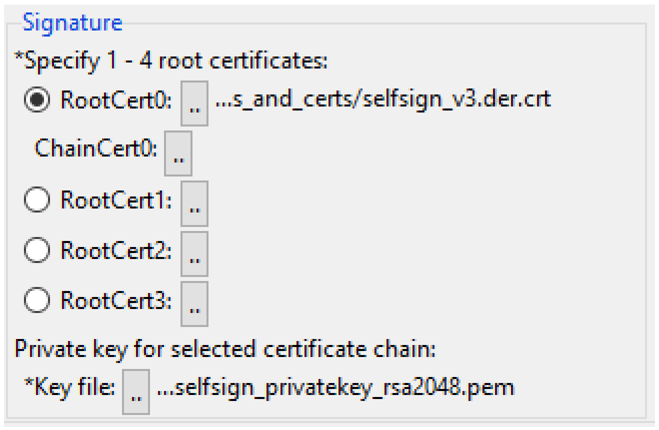

TrustZone image type
TZ-M Enabled TZ-M Disabled
Use preset data: ..._zone_config/lpc55xx.json

Figure 9. TrustZone-M configuration detail

A signed image must be included with a minimum of 1 and maximum of 4 root certificates. The root certificate can be alone in certificate chain. In this case, the root certificate must be a self-signed nonCA certificate. In elftosb-gui, is it possible to create a certificate chain maximally with two certificates. In the case of two certificates in a chain, the root certificate must be self-signed CA, and the second signed by a root certificate and is nonCA. For creating bigger certificate chains, it is necessary to manually update the json image configuration file. For details, see Appendix D in the *Kinetis Elftosb User's Guide* (document MBOOTELFTOSBUG).



Signature

*Specify 1 - 4 root certificates:

[✓ RootCert0:] .. ...s_and_certs/selfsign_v3.der.crt

ChainCert0: ..

RootCert1: ..

RootCert2: ..

RootCert3: ..

Private key for selected certificate chain:

*Key file: .. ...selfsign_privatekey_rsa2048.pem

Figure 10. Image signature configuration

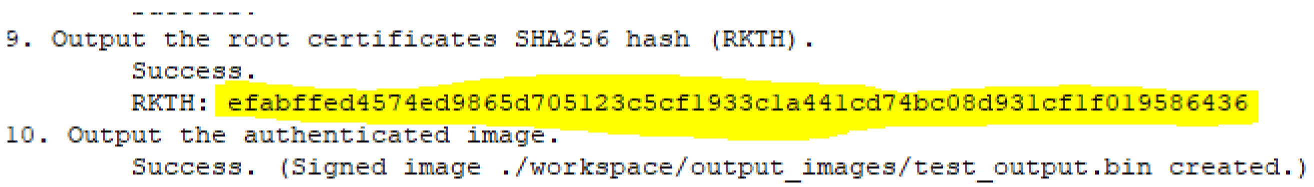
All certificates are expected to be X.509 v3 certificates in DER format.

One of the specified certificate chains must be selected by the "Radio" button next to the RootCert specification. The selected certificate chain is used for signature of an image, and the other certificate chains are stored for later use.

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

For signed images, elftosb-gui shows (in the output window) the RKTH value generated during the signature process by the elftosb command line tool. elftosb-gui shows (in the output window) the RKTH value generated during the signature process by the elftosb command line tool. RKTH is the hash value of hashes of provided root certificates. This value must be uploaded for the first time to the target device. Uploading is done by the blhost application. For more details, see the *blhost User's Guide* (document MCUBLHOSTUG).

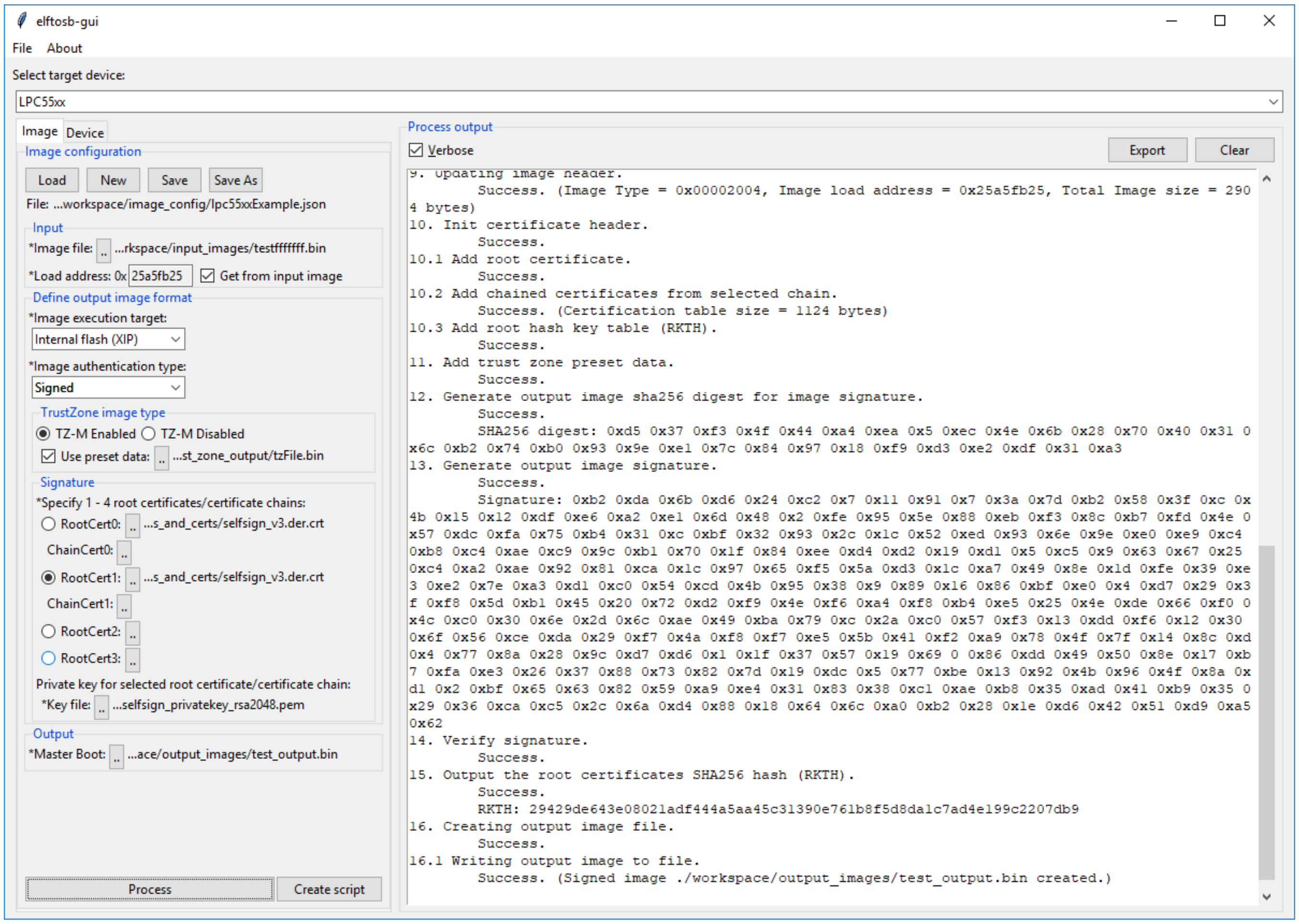
If the new image is produced with different root certificates, the new image will not be accepted by the device due to different RKTH values.



```
        --------
9. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: efabffed4574ed9865d705123c5cf1933c1a441cd74bc08d931cf1f019586436
10. Output the authenticated image.
        Success. (Signed image ./workspace/output_images/test_output.bin created.)
```

Figure 11. RKTH value in the elftosb-gui output window

Output details can be limited by unchecking the 'Verbose' option in output area. The "Clear" button can also be used to remove old output between runs.



Select target device: LPC55xx

Image configuration

File: ...workspace/image_config/lpc55xxExample.json

Input

*Image file: ...rkspace/input_images/testfffffff.bin

*Load address: 0x 25a5fb25 ☑ Get from input image

Define output image format

*Image execution target: Internal flash (XIP)

*Image authentication type: Signed

TrustZone image type

● TZ-M Enabled ○ TZ-M Disabled

☑ Use preset data: ...st_zone_output/tzFile.bin

Signature

*Specify 1 - 4 root certificates/certificate chains:

○ RootCert0: ...s_and_certs/selfsign_v3.der.crt

ChainCert0:

● RootCert1: ...s_and_certs/selfsign_v3.der.crt

ChainCert1:

○ RootCert2:

○ RootCert3:

Private key for selected root certificate/certificate chain:

*Key file: ...selfsign_privatekey_rsa2048.pem

Output

*Master Boot: ...ace/output_images/test_output.bin

Process output

☑ Verbose

```
9. Updating image header.
        Success. (Image Type = 0x00002004, Image load address = 0x25a5fb25, Total Image size = 290
4 bytes)
10. Init certificate header.
        Success.
10.1 Add root certificate.
        Success.
10.2 Add chained certificates from selected chain.
        Success. (Certification table size = 1124 bytes)
10.3 Add root hash key table (RKTH).
        Success.
11. Add trust zone preset data.
        Success.
12. Generate output image sha256 digest for image signature.
        Success.
        SHA256 digest: 0xd5 0x37 0xf3 0x4f 0x44 0xa4 0xea 0x5 0xec 0x4e 0x6b 0x28 0x70 0x40 0x31 0
x6c 0xb2 0x74 0xb0 0x93 0x9e 0xe1 0x7c 0x84 0x97 0x18 0xf9 0xd3 0xe2 0xdf 0x31 0xa3
13. Generate output image signature.
        Success.
        Signature: 0xb2 0xda 0x6b 0xd6 0x24 0xc2 0x7 0x11 0x91 0x7 0x3a 0x7d 0xb2 0x58 0x3f 0xc 0x
4b 0x15 0x12 0xdf 0xe6 0xa2 0xe1 0x6d 0x48 0x2 0xfe 0x95 0x5e 0x88 0xeb 0xf3 0x8c 0xb7 0xfd 0x4e 0
x57 0xdc 0xfa 0x75 0xb4 0x31 0xc 0xbf 0x32 0x93 0x2c 0x1c 0x52 0xed 0x93 0x6e 0x9e 0xe0 0xe9 0xc4
0xb8 0xc4 0xae 0xc9 0x9c 0xb1 0x70 0x1f 0x84 0xee 0xd4 0xd2 0x19 0xd1 0x5 0xc5 0x9 0x63 0x67 0x25
0xc4 0xa2 0xae 0x92 0x81 0xca 0x1c 0x97 0x65 0xf5 0x5a 0xd3 0x1c 0xa7 0x49 0x8e 0x1d 0xfe 0x39 0xe
3 0xe2 0x7e 0xa3 0xd1 0xc0 0x54 0xcd 0x4b 0x95 0x38 0x9 0x89 0x16 0x86 0xbf 0xe0 0x4 0xd7 0x29 0x3
f 0xf8 0x5d 0xb1 0x45 0x20 0x72 0xd2 0xf9 0x4e 0xf6 0xa4 0xf8 0xb4 0xe5 0x25 0x4e 0xde 0x66 0xf0 0
x4c 0xc0 0x30 0x6e 0x2d 0x6c 0xae 0x49 0xba 0x79 0xc 0x2a 0xc0 0x57 0xf3 0x13 0xdd 0xf6 0x12 0x30
0x6f 0x56 0xce 0xda 0x29 0xf7 0x4a 0xf8 0xf7 0xe5 0x5b 0x41 0xf2 0xa9 0x78 0x4f 0x7f 0x14 0x8c 0xd
0x4 0x77 0x8a 0x28 0x9c 0xd7 0xd6 0x1 0x1f 0x37 0x57 0x19 0x69 0 0x86 0xdd 0x49 0x50 0x8e 0x17 0xb
7 0xfa 0xe3 0x26 0x37 0x88 0x73 0x82 0x7d 0x19 0xdc 0x5 0x77 0xbe 0x13 0x92 0x4b 0x96 0x4f 0x8a 0x
d1 0x2 0xbf 0x65 0x63 0x82 0x59 0xa9 0xe4 0x31 0x83 0x38 0xc1 0xae 0xb8 0x35 0xad 0x41 0xb9 0x35 0
x29 0x36 0xca 0xc5 0x2c 0x6a 0xd4 0x88 0x18 0x64 0x6c 0xa0 0xb2 0x28 0x1e 0xd6 0x42 0x51 0xd9 0xa5
0x62
14. Verify signature.
        Success.
15. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: 29429de643e08021adf444a5aa45c31390e761b8f5d8da1c7ad4e199c2207db9
16. Creating output image file.
        Success.
16.1 Writing output image to file.
        Success. (Signed image ./workspace/output_images/test_output.bin created.)
```

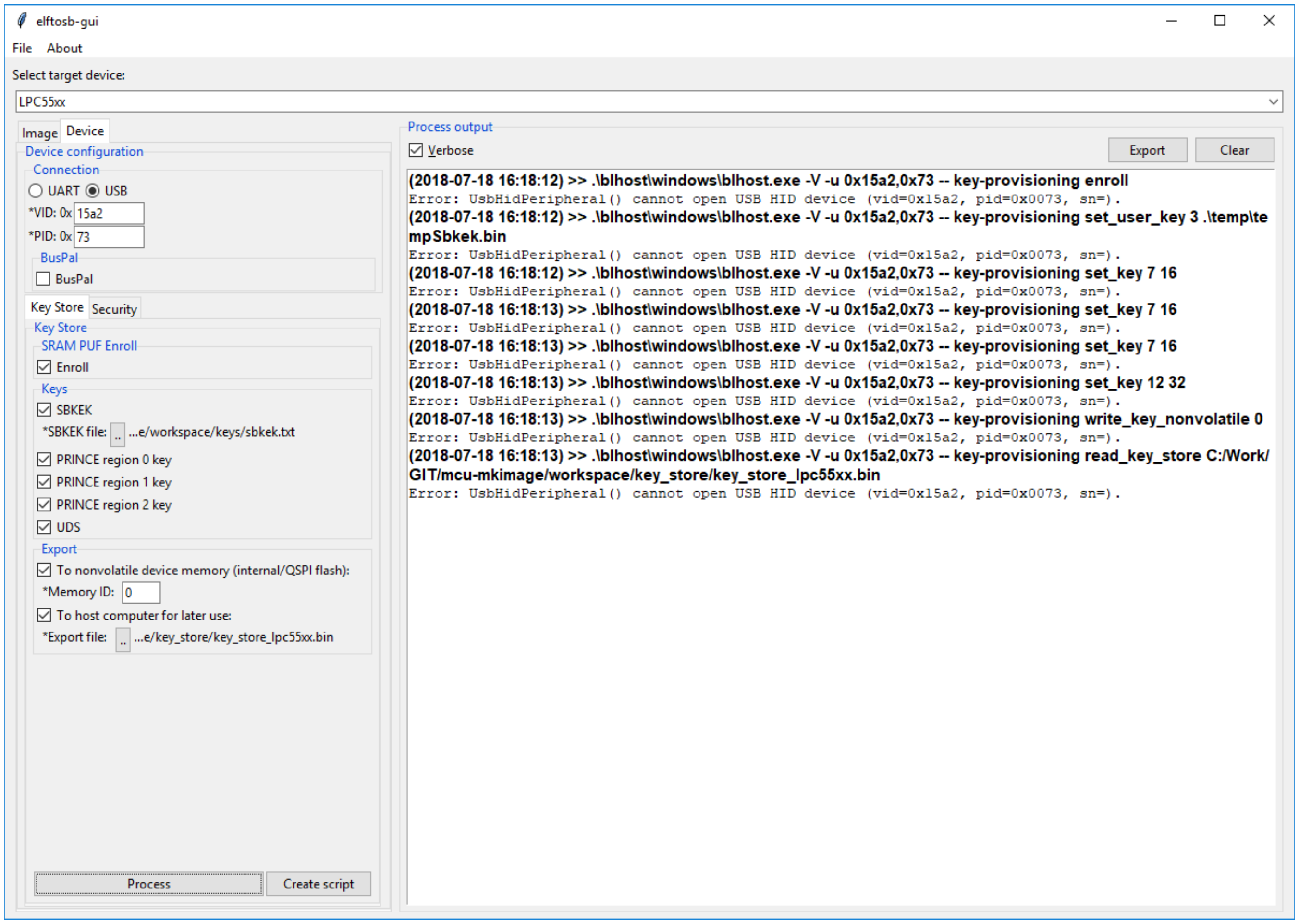
Figure 12. Successfully created image

4.2 Key store creation assist

Key store is a binary data structure that holds keys. This application can assist in creating key store with keys that are being used by the boot ROM. The initialization is done by the command line blhost tool. Elftosb-gui offers to do it more easily than command line.

The user needs to ensure that the targeting device is connected to the host PC and is in a state where it is able interact with blhost. For more details, see the *blhost User's Guide* (document MCUBLHOSTUG) and the device manual.

The user needs switch to the Device tab, define the connection to device (more details about connection in the *blhost User's Guide* (document MCUBLHOSTUG)), select the Key Store tab, specify which keys upload/generate, and specify how to save Key Store generated in the device RAM. Once the is configuration ready, click the "Process" button. In case of missing mandatory user input, the field is marked in red. If all inputs are provided, in output area displays the interaction with the blhost application.



Select target device: LPC55xx

UART [✓ USB]
*VID: 0x 15a2
*PID: 0x 73
☐ BusPal
☑ Enroll
☑ SBKEK
*SBKEK file: ...e/workspace/keys/sbkek.txt
☑ PRINCE region 0 key
☑ PRINCE region 1 key
☑ PRINCE region 2 key
☑ UDS
☑ To nonvolatile device memory (internal/QSPI flash):
*Memory ID: 0
☑ To host computer for later use:
*Export file: ...e/key_store/key_store_lpc55xx.bin
☑ Verbose

```
(2018-07-18 16:18:12) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning enroll
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:12) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning set_user_key 3 .\temp\te
mpSbkek.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:12) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning set_key 7 16
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:13) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning set_key 7 16
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:13) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning set_key 7 16
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:13) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning set_key 12 32
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:13) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning write_key_nonvolatile 0
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:18:13) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- key-provisioning read_key_store C:/Work/
GIT/mcu-mkimage/workspace/key_store/key_store_lpc55xx.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
```

Figure 13. Key Store intialization of lpc55xx

The provided sbkek file is expected in hex string format. PRINCE and UDS keys are generated by the device itself (using PUF).

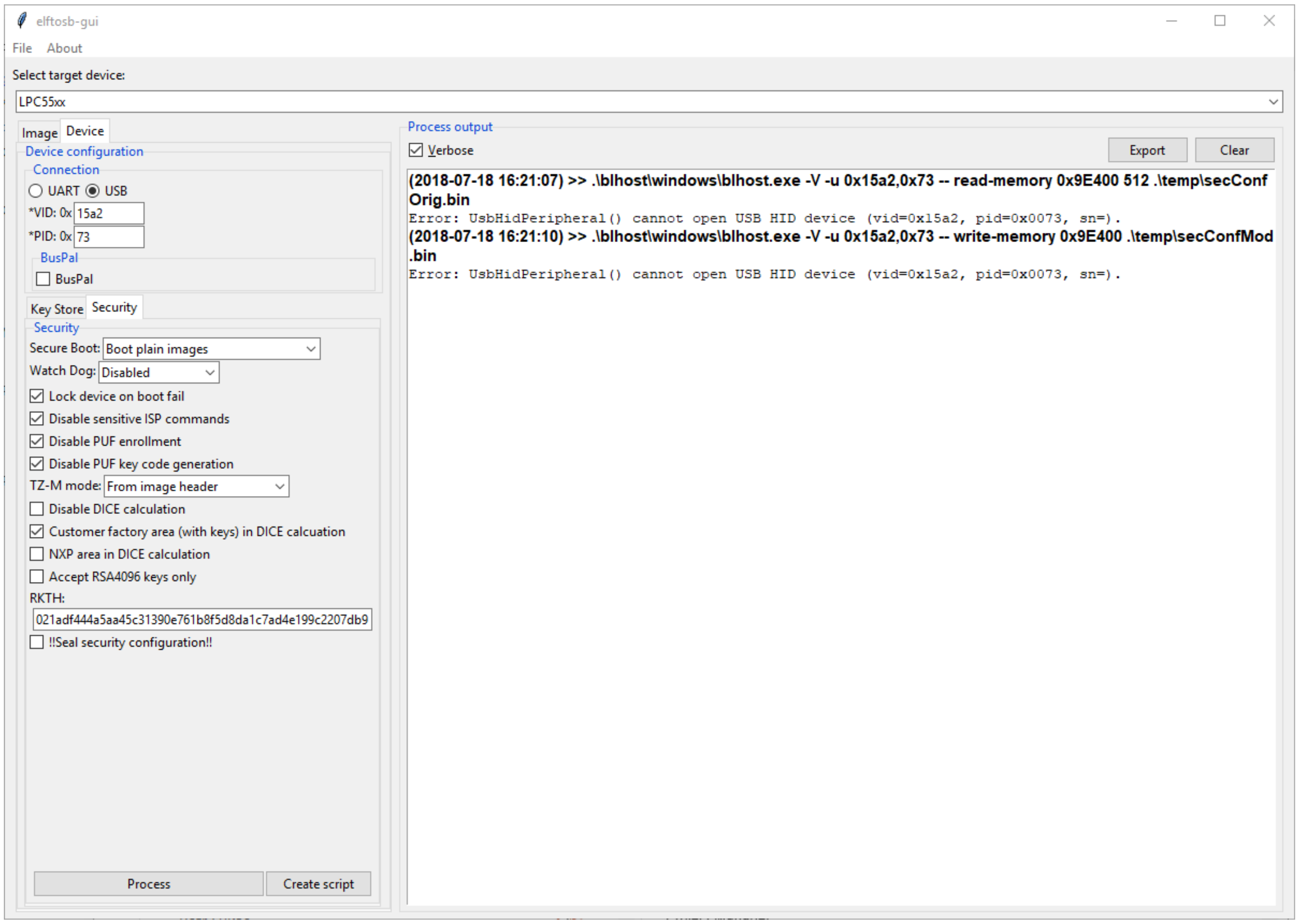
The generated Key Store can be exported from RAM to non-volatile memory in the device, or can be downloaded to the host PC and stored in binary format.

4.3 Security configuration

Elftosb-gui offers a security configuration for the lpc55xx MCU platform. The configuration is stored in the flash memory of the device and is accessible by blocks. For a security configuration, read-modify-write sequence is executed on the memory block with the security configuration. The elftosb-gui uses the blhost to get the memory block from the device and uploads it back with a new configuration. Only security registers in the memory block are modified, other device configuration registers on the page stay untouched.

The user needs to select the Device tab, define the connection to the device (more details about connection in the *blhost User's Guide* (document MCUBLHOSTUG), switch to the Security tab inside the Device tab, and specify the security configuration.

Once the security specification is done, click the "Process" button so the configuration is uploaded to the device. The user has to ensure that device is connected and ready to communicate with the blhost application. Follow the output window to check the progress and results of the upload.



Select target device: LPC55xx

Device configuration

Connection: UART [✓ USB]

*VID: 0x 15a2

*PID: 0x 73

BusPal: ☐ BusPal

Security

Secure Boot: Boot plain images

Watch Dog: Disabled

- ☑ Lock device on boot fail
- ☑ Disable sensitive ISP commands
- ☑ Disable PUF enrollment
- ☑ Disable PUF key code generation

TZ-M mode: From image header

- ☐ Disable DICE calculation
- ☑ Customer factory area (with keys) in DICE calcuation
- ☐ NXP area in DICE calculation
- ☐ Accept RSA4096 keys only

RKTH: 021adf444a5aa45c31390e761b8f5d8da1c7ad4e199c2207db9

- ☐ !!Seal security configuration!!

Process output

☑ Verbose

```
(2018-07-18 16:21:07) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- read-memory 0x9E400 512 .\temp\secConf
Orig.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-07-18 16:21:10) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -- write-memory 0x9E400 .\temp\secConfMod
.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
```

Figure 14. Security configuration of lpc55xx

If the RKTH entry is empty, the RKTH value in the security configuration memory block is not modified (it keeps the read value).

The !!Seal Security Configuration!! option adds SHA256 hash to the end of the security configuration memory page, and the content of the page locks for any other modification.

4.4 Create script

The elftosb-gui tool offers possibility to save output as a command line script for later use directly by command line elftosb command line tool.

Use the "Create Script" button, next to "Process" button, on each operation for the lpc55xx device family in mkimage. If some mandatory input is missing, the field is marked in red.

The user will be prompted to specify the output file (script). The script is generated for the actual operating system (Windows, Linux, MAC).

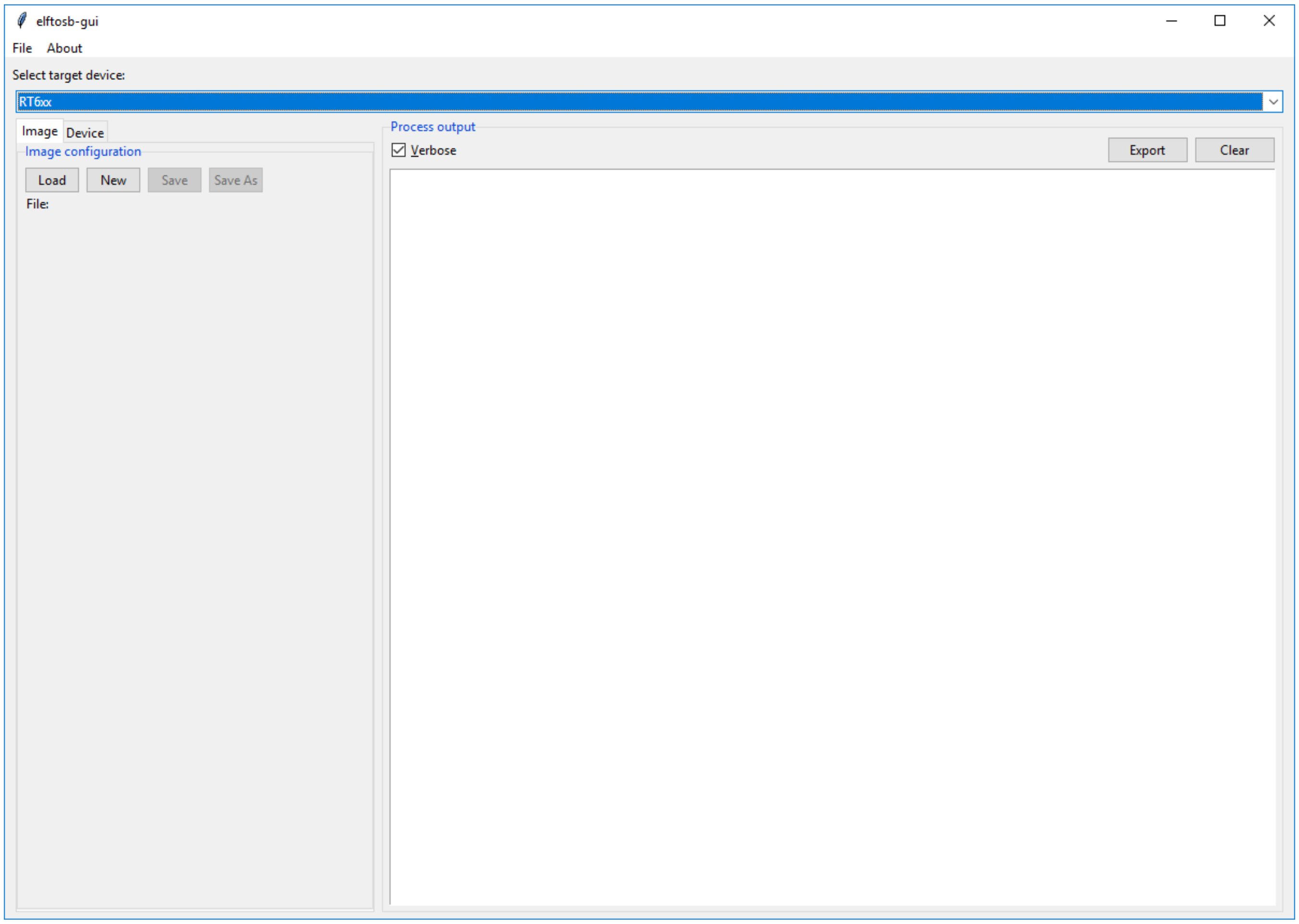
The script can be modified and used, for example, in process automation.

Chapter 5
RT6xx MCU platform

For the rt6xx MCU platform, the elftosb-gui offers a wizard for creating a master boot image and Key Store initialization of the device.

5.1 Master boot image generation

The elftosb-gui allows user to create, modify or use an image configuration file. The tool can open an existing image configuration file, or, create a new image configuration file and save it for later use. The image configuration file is a json text file and it is an input required by elftosb command line tool for master boot image generation. The elftosb command line tool is available with the elftosb-gui, thus, user can directly generate the master boot image from the GUI. Alternatively, if the user has the image configuration file, the master boot image can be generated from command line (by calling elftosb) without involving the GUI.



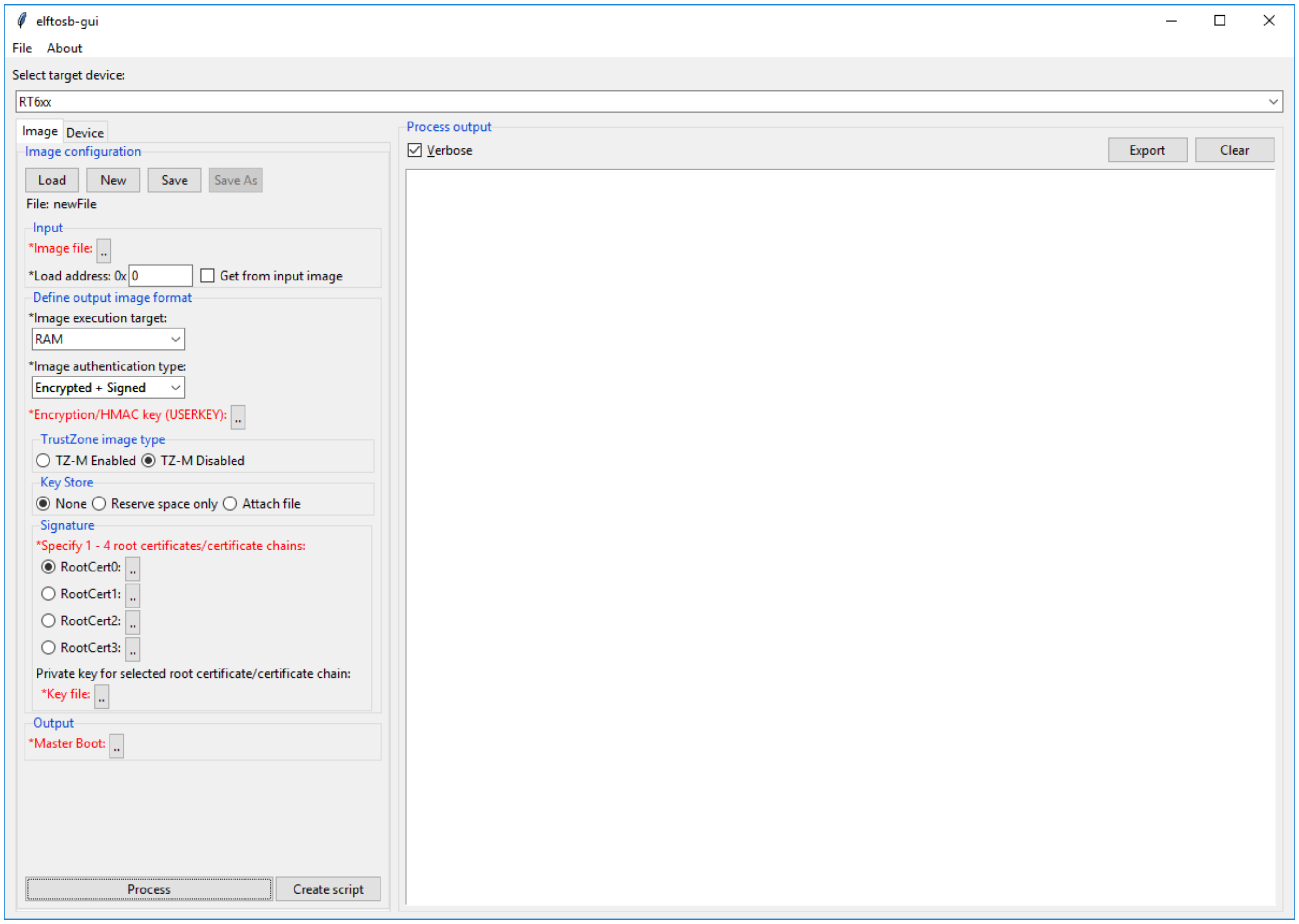
Select target device: RT6xx

Figure 15. rt6xx family main layout

Click the "Load" button to open an existing configuration, the "New" button for creating a new configuration, and "Save" or "Save As" for saving the created or modified configuration.

During creation of the new image configuration, the user will be guided by elftosb-gui to successfully create the correct configuration.

If the configuration is finished, click the "Process" button to call the elftosb application to parse the configuration and create the output image file. In case of missing configuration inputs, elftosb-gui marks problematic fields in red.



Select target device: RT6xx

File: newFile

*Load address: 0x 0 ☐ Get from input image

*Image execution target: RAM

*Image authentication type: Encrypted + Signed

*Encryption/HMAC key (USERKEY):

○ TZ-M Enabled [✓ ● TZ-M Disabled]

[✓ ● None] ○ Reserve space only ○ Attach file

[✓ ● RootCert0:] ○ RootCert1: ○ RootCert2: ○ RootCert3:

☑ Verbose

Figure 16. Missing configuration inputs for rt6xx

Images configured to be executed in RAM with signature require specifying the AES-256 used for HMAC key calculation and image encryption. The key is expected in hex string format. This key is also known as USERKEY, and must be uploaded to the device before loading the image. See the next chapter about Key Store initialization for more information.

Figure 17. Encryption/HMAC key (USERKEY)

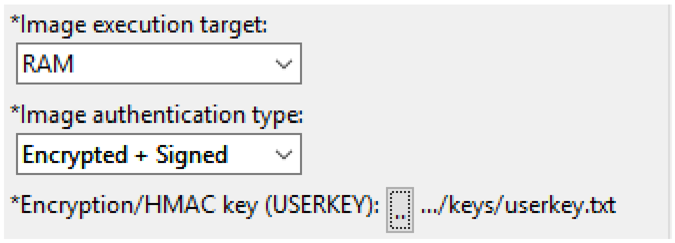


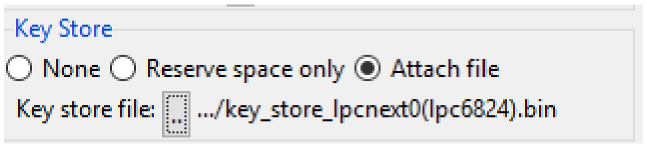
*Image execution target: RAM

*Image authentication type: Encrypted + Signed

*Encryption/HMAC key (USERKEY): .../keys/userkey.txt

The next option available is only for images with execution in RAM and signature possibility includes Key Store to the image. This feature can go unused, and only the Key Store space with zeros in the output image structure can be reserved for further replacement, or the Key Store file can be included. The file is expected in binary format. See the next chapter about Key Store initialization and how to get it from the device.

Figure 18. Including Key Store to the image



None [✓ Attach file] Reserve space only

Key store file: .../key_store_lpcnext0(lpc6824).bin

In case there is only Key Store reservation after processing, check the output area to find the position and size of the reserved Key Store in the output image.

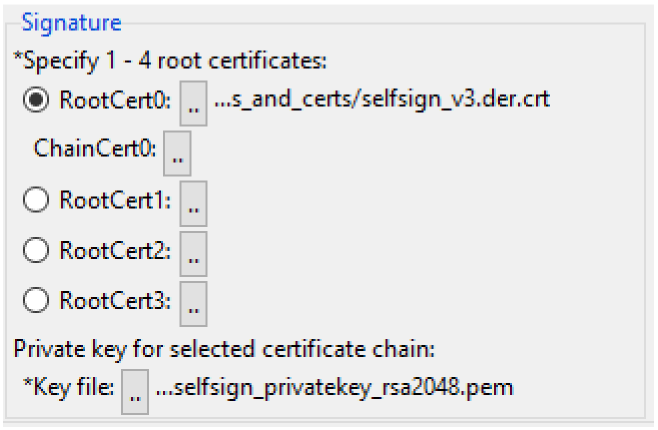
Figure 19. Key Store location in image

```
8.1 Reserve key store space only.
        Key store with size 1424 bytes will be reserved in image from address: 0x60.
        Success.
```

The TrustZone-M preset configuration can be included as a binary file, which is directly copied into the output image or, alternatively, as a json TrustZone-M preset configuration file, and is processed during image creation with binary data included in the output image. For more information about TrustZone-M preset configuration file, see Appendex E of the *Elftosb User*'s *Guide* (document MBOOTELFTOSBUG).

A signed image must be included with a minimum of 1 and maximum of 4 root certificates. The root certificate can be alone in certificate chain, in such case the root certificate must be a self-signed nonCA certificate. In elftosb-gui, is it possible to create a certificate chain maximally with two certificates. In the case of two certificates in a chain, the root certificate must be self-signed CA, and the second signed by a root certificate and is nonCA. For creating bigger certificate chains, it is necessary to manually update the json image configuration file. For details, see Appendix D in the *Kinetis Elftosb User*'s *Guide* (document MBOOTELFTOSBUG).

Figure 20. Image signature configuration



*Specify 1 - 4 root certificates:

[✓ RootCert0]: ...s_and_certs/selfsign_v3.der.crt

ChainCert0:

RootCert1:

RootCert2:

RootCert3:

Private key for selected certificate chain:

*Key file: ...selfsign_privatekey_rsa2048.pem

All certificates are expected as X.509 v3 certificates in DER format.

One of the specified certificate chains must be selected by using the "Radio" button next to the RootCert specification. The selected certificate chain is used for the signature of image, and other certificate chains are stored for later use.

The key file shall point to a private key in a PEM or DER format. This is the private key that corresponds to image signing certificate public key in the last certificate in the selected certificate chain.

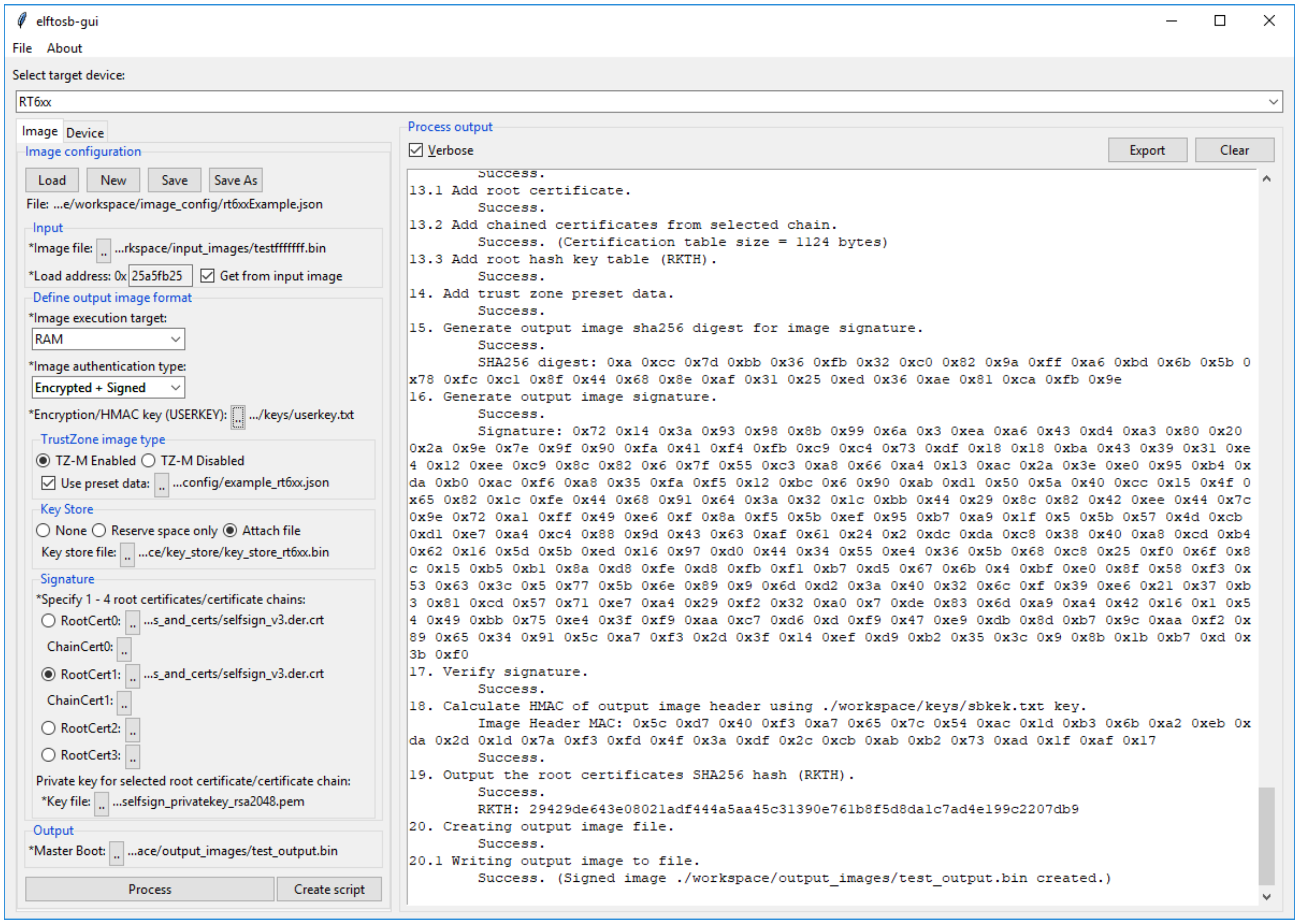
For signed images, elftosb shows RKTH value in the mkimage output window. RKTH is hash value of hashes for provided root certificates. This value must be uploaded the first time to the target device. The upload is done by the blhost application. For more details, see the *blhost User*'s *Guide* (document MCUBLHOSTUG).

If the new image is produced with different root certificates, the new image is not accepted by the device due to different RKTH values.

Figure 21. RKTH value in elftosb-gui output window

```
16. Output the certificate SHA256 hash.
        Success.
        RKTH SHA256 HASH: efabffed4574ed9865d705123c5cf1933c1a441cd74bc08d931cf1f019586436
        Success. (authenticated image file: C:/Work/Python/mkimage_ver101/workspace/output_images/te
```

The output details can be limited by unchecking the Verbose option in output area. The "Clear" button can be used to remove old output between runs.



Select target device: RT6xx

Image configuration

File: ...e/workspace/image_config/rt6xxExample.json

Input

*Image file: ...rkspace/input_images/testfffffff.bin

*Load address: 0x 25a5fb25 ☑ Get from input image

Define output image format

*Image execution target: RAM

*Image authentication type: Encrypted + Signed

*Encryption/HMAC key (USERKEY): .../keys/userkey.txt

TrustZone image type

[✓ TZ-M Enabled] TZ-M Disabled

☑ Use preset data: ...config/example_rt6xx.json

Key Store

None Reserve space only [✓ Attach file]

Key store file: ...ce/key_store/key_store_rt6xx.bin

Signature

*Specify 1 - 4 root certificates/certificate chains:

RootCert0: ...s_and_certs/selfsign_v3.der.crt

ChainCert0:

[✓ RootCert1:] ...s_and_certs/selfsign_v3.der.crt

ChainCert1:

RootCert2:

RootCert3:

Private key for selected root certificate/certificate chain:

*Key file: ...selfsign_privatekey_rsa2048.pem

Output

*Master Boot: ...ace/output_images/test_output.bin

Process output

☑ Verbose

```
        Success.
13.1 Add root certificate.
        Success.
13.2 Add chained certificates from selected chain.
        Success. (Certification table size = 1124 bytes)
13.3 Add root hash key table (RKTH).
        Success.
14. Add trust zone preset data.
        Success.
15. Generate output image sha256 digest for image signature.
        Success.
        SHA256 digest: 0xa 0xcc 0x7d 0xbb 0x36 0xfb 0x32 0xc0 0x82 0x9a 0xff 0xa6 0xbd 0x6b 0x5b 0
x78 0xfc 0xc1 0x8f 0x44 0x68 0x8e 0xaf 0x31 0x25 0xed 0x36 0xae 0x81 0xca 0xfb 0x9e
16. Generate output image signature.
        Success.
        Signature: 0x72 0x14 0x3a 0x93 0x98 0x8b 0x99 0x6a 0x3 0xea 0xa6 0x43 0xd4 0xa3 0x80 0x20
0x2a 0x9e 0x7e 0x9f 0x90 0xfa 0x41 0xf4 0xfb 0xc9 0xc4 0x73 0xdf 0x18 0x18 0xba 0x43 0x39 0x31 0xe
4 0x12 0xee 0xc9 0x8c 0x82 0x6 0x7f 0x55 0xc3 0xa8 0x66 0xa4 0x13 0xac 0x2a 0x3e 0xe0 0x95 0xb4 0x
da 0xb0 0xac 0xf6 0xa8 0x35 0xfa 0xf5 0x12 0xbc 0x6 0x90 0xab 0xd1 0x50 0x5a 0x40 0xcc 0x15 0x4f 0
x65 0x82 0x1c 0xfe 0x44 0x68 0x91 0x64 0x3a 0x32 0x1c 0xbb 0x44 0x29 0x8c 0x82 0x42 0xee 0x44 0x7c
0x9e 0x72 0xa1 0xff 0x49 0xe6 0xf 0x8a 0xf5 0x5b 0xef 0x95 0xb7 0xa9 0x1f 0x5 0x5b 0x57 0x4d 0xcb
0xd1 0xe7 0xa4 0xc4 0x88 0x9d 0x43 0x63 0xaf 0x61 0x24 0x2 0xdc 0xda 0xc8 0x38 0x40 0xa8 0xcd 0xb4
0x62 0x16 0x5d 0x5b 0xed 0x16 0x97 0xd0 0x44 0x34 0x55 0xe4 0x36 0x5b 0x68 0xc8 0x25 0xf0 0x6f 0x8
c 0x15 0xb5 0xb1 0x8a 0xd8 0xfe 0xd8 0xfb 0xf1 0xb7 0xd5 0x67 0x6b 0x4 0xbf 0xe0 0x8f 0x58 0xf3 0x
53 0x63 0x3c 0x5 0x77 0x5b 0x6e 0x89 0x9 0x6d 0xd2 0x3a 0x40 0x32 0x6c 0xf 0x39 0xe6 0x21 0x37 0xb
3 0x81 0xcd 0x57 0x71 0xe7 0xa4 0x29 0xf2 0x32 0xa0 0x7 0xde 0x83 0x6d 0xa9 0xa4 0x42 0x16 0x1 0x5
4 0x49 0xbb 0x75 0xe4 0x3f 0xf9 0xaa 0xc7 0xd6 0xd 0xf9 0x47 0xe9 0xdb 0x8d 0xb7 0x9c 0xaa 0xf2 0x
89 0x65 0x34 0x91 0x5c 0xa7 0xf3 0x2d 0x3f 0x14 0xef 0xd9 0xb2 0x35 0x3c 0x9 0x8b 0x1b 0xb7 0xd 0x
3b 0xf0
17. Verify signature.
        Success.
18. Calculate HMAC of output image header using ./workspace/keys/sbkek.txt key.
        Image Header MAC: 0x5c 0xd7 0x40 0xf3 0xa7 0x65 0x7c 0x54 0xac 0x1d 0xb3 0x6b 0xa2 0xeb 0x
da 0x2d 0x1d 0x7a 0xf3 0xfd 0x4f 0x3a 0xdf 0x2c 0xcb 0xab 0xb2 0x73 0xad 0x1f 0xaf 0x17
        Success.
19. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: 29429de643e08021adf444a5aa45c31390e761b8f5d8da1c7ad4e199c2207db9
20. Creating output image file.
        Success.
20.1 Writing output image to file.
        Success. (Signed image ./workspace/output_images/test_output.bin created.)
```

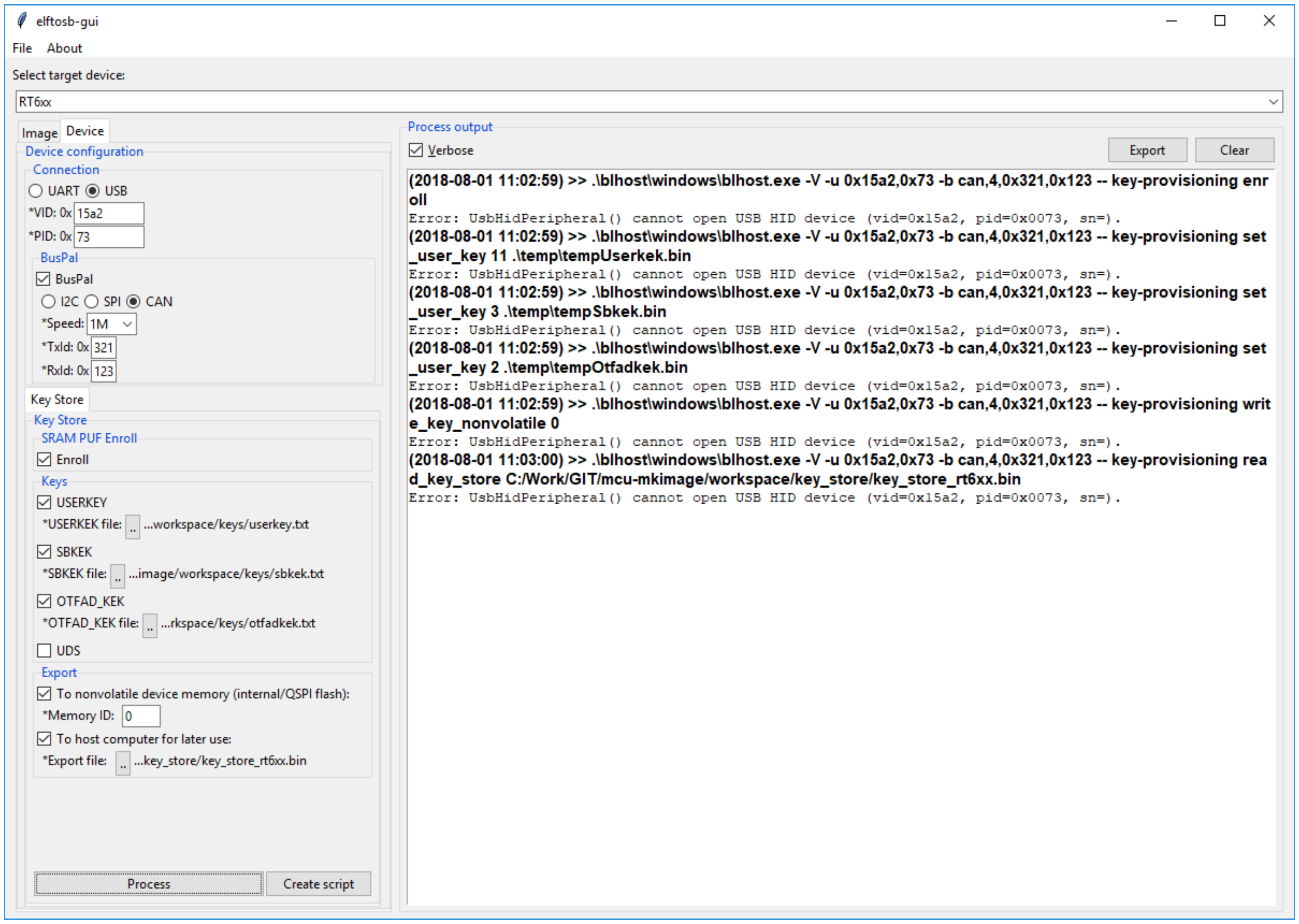
Figure 22. Successfully created image for rt6xx

5.2 Key Store assist

Key store is a binary data structure that holds keys. This application can assist in creating key store with keys that are being used by the boot ROM. The initialization is done by the command line blhost tool. Elftosb-gui offers to do it more easily than command line.

The user needs to ensure that the targeting device is connected to the host PC and is in a state where it is able interact with blhost. For more details, see the *blhost User's Guide* (document MCUBLHOSTUG) and the device manual.

The user needs switch to the Device tab, define the connection to device (more details about connection in the *blhost User's Guide* (document MCUBLHOSTUG)), select the Key Store tab, specify which keys upload/generate, and specify how to save Key Store generated in the device RAM. Once the is configuration ready, click the "Process" button. In case of missing mandatory user input, the field is marked in red. If all inputs are provided, in output area displays the interaction with the blhost application.



Select target device: RT6xx

UART [✓ USB]
*VID: 0x 15a2
*PID: 0x 73
[✓] BusPal
I2C SPI [✓ CAN]
*Speed: 1M
*TxId: 0x 321
*RxId: 0x 123

[✓] Enroll
[✓] USERKEY
*USERKEK file: ...workspace/keys/userkey.txt
[✓] SBKEK
*SBKEK file: ...image/workspace/keys/sbkek.txt
[✓] OTFAD_KEK
*OTFAD_KEK file: ...rkspace/keys/otfadkek.txt
[ ] UDS
[✓] To nonvolatile device memory (internal/QSPI flash):
*Memory ID: 0
[✓] To host computer for later use:
*Export file: ...key_store/key_store_rt6xx.bin

[✓] Verbose

```
(2018-08-01 11:02:59) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning enr
oll
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-08-01 11:02:59) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning set
_user_key 11 .\temp\tempUserkek.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-08-01 11:02:59) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning set
_user_key 3 .\temp\tempSbkek.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-08-01 11:02:59) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning set
_user_key 2 .\temp\tempOtfadkek.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-08-01 11:02:59) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning writ
e_key_nonvolatile 0
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
(2018-08-01 11:03:00) >> .\blhost\windows\blhost.exe -V -u 0x15a2,0x73 -b can,4,0x321,0x123 -- key-provisioning rea
d_key_store C:/Work/GIT/mcu-mkimage/workspace/key_store/key_store_rt6xx.bin
Error: UsbHidPeripheral() cannot open USB HID device (vid=0x15a2, pid=0x0073, sn=).
```

Figure 23. Key Store initialization of rt6xx

The provided key files are expected in hex string format. The UDS key is generated by the device itself (using PUF).

The generated Key Store can be exported from RAM to non-volatile memory in the device, or can be downloaded to host PC and stored in binary format.

5.3 Create script

The elftosb-gui tool offers possibility to save output as a command line script for later use directly by command line elftosb command line tool.

Use the "Create Script" button, next to "Process" button, on each operation in mkimage. If some mandatory input is missing, the field is marked in red.

The user will be prompted to specify the output file (script). The script is generated for the actual operating system (Windows, Linux, MAC).

The script can be modified and used, for example, in process automation.

Chapter 6 Revision history

This is the first revision of the document.

Chapter 7
Appendix A: Hex string key format

The hex string key is an uninterrupted string of hexadecimal characters.

- AES-128 bit key example in hex string format (32 hexadecimal characters):

  ```
  3F3CFBC001F399991035C3C6C7065924
  ```

- AES-256 bit key example in hex string format (64 hexadecimal characters):

  ```
  338D3E817A337A48BA3C46486571789B3CEEC831D1D93E635D186A75A33B7A7E
  ```