GigaDevice Semiconductor Inc.

GD32W51x Secure Boot User Guide

Application Note
AN082

Table of Contents

Table of Contents 2
List of Figures 3
List of Tables 4
1. Secure boot 5
1.1. Introduction 5
1.2. FLASH layout 5
1.3. SRAM layout 6
2. Hardware configuration 7
2.1. SIP Flash 7
2.2. QSPI Flash 8
3. Firmware package generation 9
3.1. Environment preparation 9
3.1.1. Keil uVision 9
3.1.2. Python 9
3.1.3. OpenSSL 9
3.2. Firmware package format 9
3.2.1. Factory package 9
3.2.2. Upgrade package 10
3.3. Firmware encapsulation 11
3.3.1. Encapsulation format 11
3.3.2. Key and certificate generation 11
3.3.3. Configuration file layout 15
3.3.4. Executable program generation 16
3.3.5. Executable program encapsulation 16
3.4. Firmware packaging 18
3.4.1. Packaging tool 18
3.4.2. Packaging process 18
4. Firmware upgrade 19
5. Abbreviations 20
6. Revision history 21

List of Figures

Figure 1-1. Bootloader process 5
Figure 1-2. FLASH layout 6
Figure 1-3. SRAM layout 6
Figure 3-1. Factory firmware package diagram 10
Figure 3-2. Upgrade firmware package diagram 10
Figure 3-3. Firmware encapsulation format 11
Figure 3-4. Generate ROT key pairs 12
Figure 3-5. Generate ROT certificate 12
Figure 3-6. ROT key contents 13
Figure 3-7. Generate the MBL key pairs 13
Figure 3-8. Generate the MBL certificate 14
Figure 3-9. Generate NSPE key pairs 14
Figure 3-10. Generate NSPE certificate 15
Figure 3-11. Configuration file layout 15

List of Tables

Table 5-1. Abbreviations 20
Table 6-1. Revision history 21

1. Secure boot

1.1. Introduction

Secure boot ensures the legitimacy and integrity of all code from the time the system runs the first instruction until it jumps to the main application. There are two key points here: The first is that the trusted first instruction of the code, the second is that the next executable segment is integrated and legitimate before the program jumps.

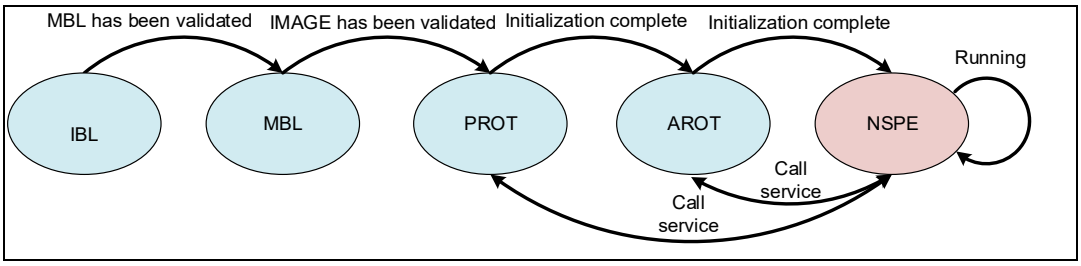
To ensure that the first instruction is always safe, the first executable segment is solidified in ROM, called immutable bootloader (IBL). When the boot mode is locked as secure boot, the system will jump to the IBL to run the first instruction no matter whether it is powered on or restarted. the boot mode cannot be tampered with. So the first instruction gets the root trust.

The second executable segment, placed at the beginning of FLASH, is called mutable bootloader (MBL). IBL is responsible for verifying the legitimacy and integrity of MBL, and the program can jump to MBL only after the verification is passed.

Later executable segments can be placed in FLASH or SRAM depending on the user's choice. If there is no special requirement, it is recommended to run the code in FLASH. The executable segments loaded later, depending on user selection, include PSA root of trust (PROT), application root of trust (AROT), and non-secure processing environment (NSPE).

The bootloader process is illustrated in ***Figure 1-1. Bootloader process***, IMAGE is the combination of PROT, AROT and NSPE. AROT can be customized according to customer requirements and is empty in the released SDK.

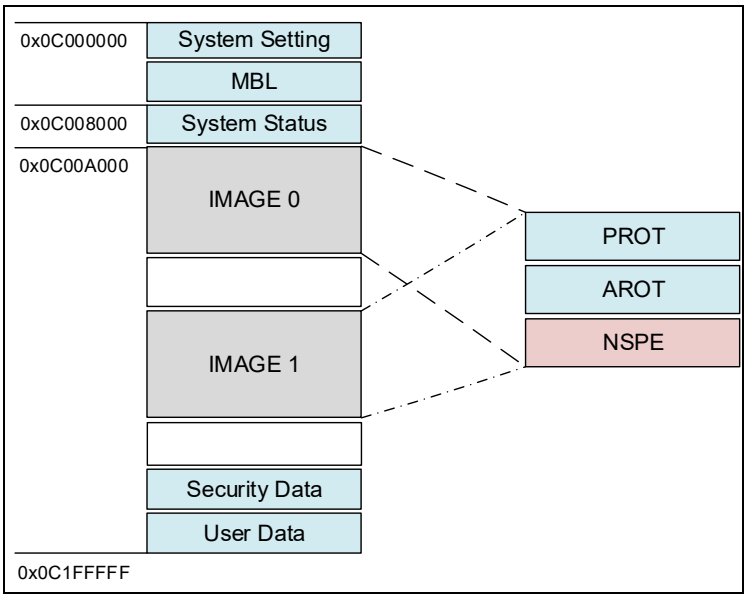
Figure 1-1. Bootloader process



1.2. FLASH layout

FLASH layout is shown in ***Figure 1-2. FLASH layout***. FLASH logical base address is classified into secure base address (0x0c000000) and non-secure base address (0x08000000). If the memory space is set to secure attribute, the secure address is used for access; otherwise, the non-secure address is used for access. In order to understand the overall layout, the secure base address is taken as an example in ***Figure 1-2. FLASH layout*** to illustrate the distribution of each part.

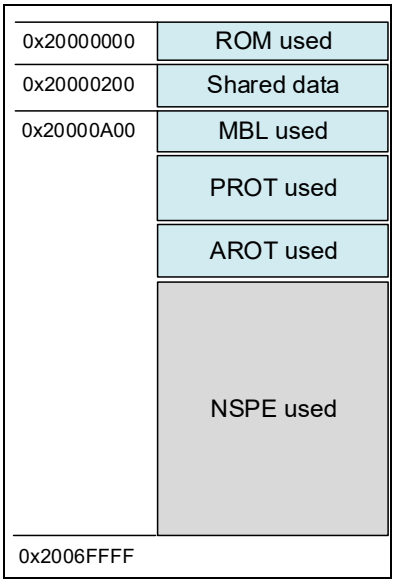
Figure 1-2. FLASH layout



1.3. SRAM layout

SRAM layout is shown in ***Figure 1-3. SRAM layout***. SRAM logical base address is classified into secure base address (0x30000000) and non-secure base address (0x20000000). If the memory space is set to secure attribute, the secure address is used for access; otherwise, the non-secure address is used for access. In the figure, the non-secure base address is used as an example to illustrate the distribution of the various parts.

Figure 1-3. SRAM layout



2. Hardware configuration

The EFUSE and Flash option bytes need to be set if the secure boot function is to be enabled. If it is SIP Flash, EFUSE and Flash option bytes need to be set. If the QSPI Flash is used, only EFUSE needs to be set.

Burn image-mp.bin, open the serial port tool, and enter commands according to the following sections to burn necessary parameters.

2.1. SIP Flash

- Boot mode (EFUSE)
  - Turn on the security bootloader option
  - *# efuse set ctl 1*
- Set validation options (EFUSE)
  - The validation option determines whether or not the ROM validates the MBL firmware and certificates. If not set, it does not validate and therefore does not truly enable a secure boot.
  - *# efuse set tzctl 0xc0*
- Set ROTPK（EFUSE）
  - ROTPK is used to verify the signature and certificate of the MBL firmware, see ***Key and certificate generation*** for how it is generated, ROTPK in the following commands can verify the certificates and firmware generated by the released SDK. After setting BIT2 of tzctl, ROTPK is locked and cannot be rewritten. The ROTPK data in the following commands is only an example.
  - *# efuse set rotpk f0cc01c8a384bdc8694b254f4dd3f8b86a25ae6ca2082c838e780e42c2b157a2*
  - *# efuse set tzctl 4*
- Enable Trust Zone（FLASH OP）
  - Enable the Trust Zone and set the SPC level to 0. 0xAA indicates that the read protection level is 0, 0x55 indicates that the read protection level is 0.5, and 0x11 indicates that the read protection level is 1.
  - *# fmcob set obr 0x80AA*
- Set SECMARK（FLASH OP）
  - Register SECMARK[0:3] can be used to configure FLASH security zone range. The command format is as follows:
  - fmcob set secmark <index> <start page> <end page>
  - The value of index ranges from 0 to 3, the value of [start page, end page] is a secure space, and the page size is 4KB. According to the user's own FLASH layout, the Page Index at the end of the PROT firmware should be filled to the end page, and the space for future PROT upgrade expansion should be reserved. Please refer to GD32W51x User Manual for detailed definition of SECMARK.
  - *# fmcob set secmark 0 0 0x39*

- Set AESK（EFUSE）
  - Optional. When the firmware needs to be encrypted, the firmware encryption key should be burned. Note that once written, only AESK-encrypted firmware will run and cannot be returned. The following AESK must be the same as the key used for generating the firmware package. After setting BIT5 of usctl, the AESK is locked and cannot be rewritten. The AESK data in the following command is an example only.
  - *# efuse set aesk 112233445566778899aabbccddeeff00*
  - *# efuse set usctl 0x20*

2.2. QSPI Flash

- Boot mode (EFUSE)
  - Turn on the security *bootloader* option
  - *# efuse set ctl 1*
- Set validation options (EFUSE)
  - The validation option determines *whether* or not the ROM validates the MBL firmware and certificates. If not set, it does not validate and therefore does not truly enable a secure boot.
  - *# efuse set tzctl 0xc0*
- Set ROTPK (EFUSE)
  - ROTPK is used to verify the signature and certificate of the MBL firmware, see ***Key and certificate generation*** for how it is generated, ROTPK in the following commands can verify the certificates and firmware generated by the released SDK. After setting BIT2 of tzctl, ROTPK is locked and cannot be rewritten. The ROTPK data in the following commands is only an example.
  - *# efuse set rotpk f0cc01c8a384bdc8694b254f4dd3f8b86a25ae6ca2082c838e780e42c2b157a2*
  - *# efuse set tzctl 4*
- Enable Trust Zone (EFUSE)
  - *# efuse set tzctl 1*
- Set AESK（EFUSE）
  - Optional. When the *firmware* needs to be encrypted, the firmware encryption key should be burned. Note that once written, only AESK-encrypted firmware will run and cannot be returned. The following AESK must be the same as the key used for generating the firmware package. After setting BIT5 of usctl, the AESK is locked and cannot be rewritten. The AESK data in the following command is an example only.
  - *# efuse set aesk 112233445566778899aabbccddeeff00*
  - *# efuse set usctl 0x20*

3. Firmware package generation

3.1. Environment preparation

3.1.1. Keil uVision

To support Arm® Cortex®-M33, please install uVision 5.25 or later.

3.1.2. Python

It is recommended to install Python3. After installing Python3, run the %SDK% \ setup.bat to automatically install the library used by the firmware encapsulation script. After the installation, add the Python path to the system environment variable PATH.

3.1.3. OpenSSL

If the operating system is Windows, OpenSSL1.1.1 is recommended. After the installation is complete, add the OpenSSL path to the system environment variable PATH.

3.2. Firmware package format

Usually there are two kinds of firmware packages, one is the factory package and the other is the upgrade package.

3.2.1. Factory package

Factory package is the firmware package burned into FLASH when the module is mass-produced. The contents of the factory package are shown in ***Figure 1-2. FLASH layout***, the System Status and User Data fields are filled with 0xFF. IMAGE1 stores production test firmware (image-mp.bin), IMAGE0 stores user firmware (image-all.bin), and Security Data stores identity data, for example, the device certificate issued by the cloud server.

After burning, jump to production test firmware by default. After the production test is complete, enter the serial port command to switch to the user firmware. After a simple test, wait for delivery.

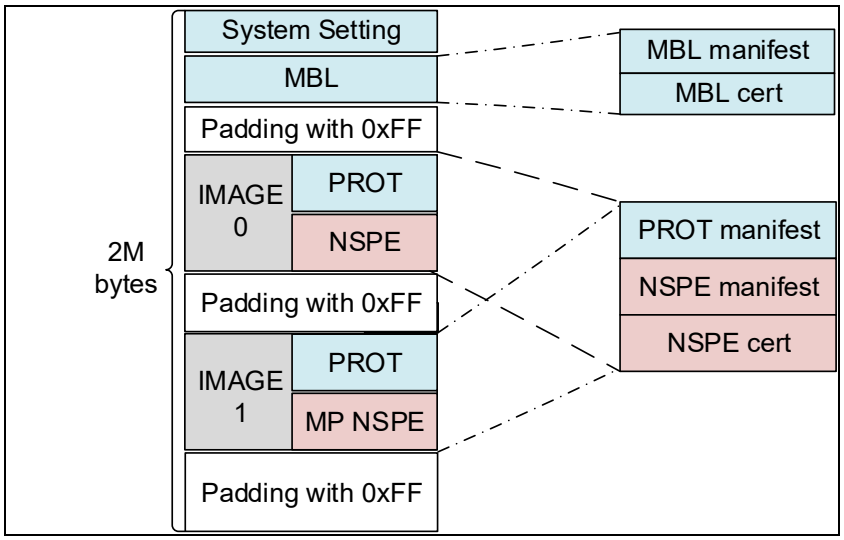
The sample factory package for this application note is based on the released SDK and has the following features:

- No AROT.
- No Security Data.
- There are two certificates, MBL certificate and NSPE certificate.
- Each executable segment has its own firmware manifest, MBL firmware manifest, PROT

firmware manifest, and NSPE firmware manifest.

The factory package format for this application note example is shown in ***Figure 3-1. Factory firmware package diagram***. See Encapsulation format for the format of the firmware manifest.

Figure 3-1. Factory firmware package diagram

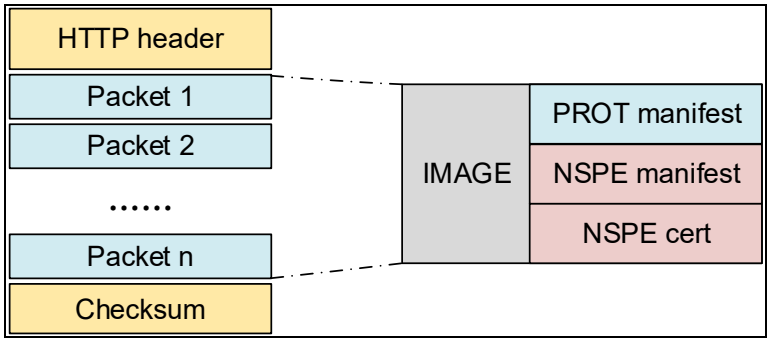


3.2.2. Upgrade package

The upgrade package means IMAGE0 or IMAGE1, each firmware update is in IMAGE, and PROT, AROT and NSPE are updated as a whole. Since the NSPE needs to call the services provided by PROT and AROT, it needs to link to the NSC libraries provided by both, which means that if the two are changed, the NSPE must update simultaneously. Considering that PROT and AROT should be as small as possible and the upgrade process should be as simple as possible, the three should be combined for unified update. AROT is not included in the upgrade package for the examples in this application note.

The HTTPS protocol is usually used to download the firmware upgrade package from the remote service area. ***Figure 3-2. Upgrade firmware package diagram*** shows the upgrade package.

Figure 3-2. Upgrade firmware package diagram



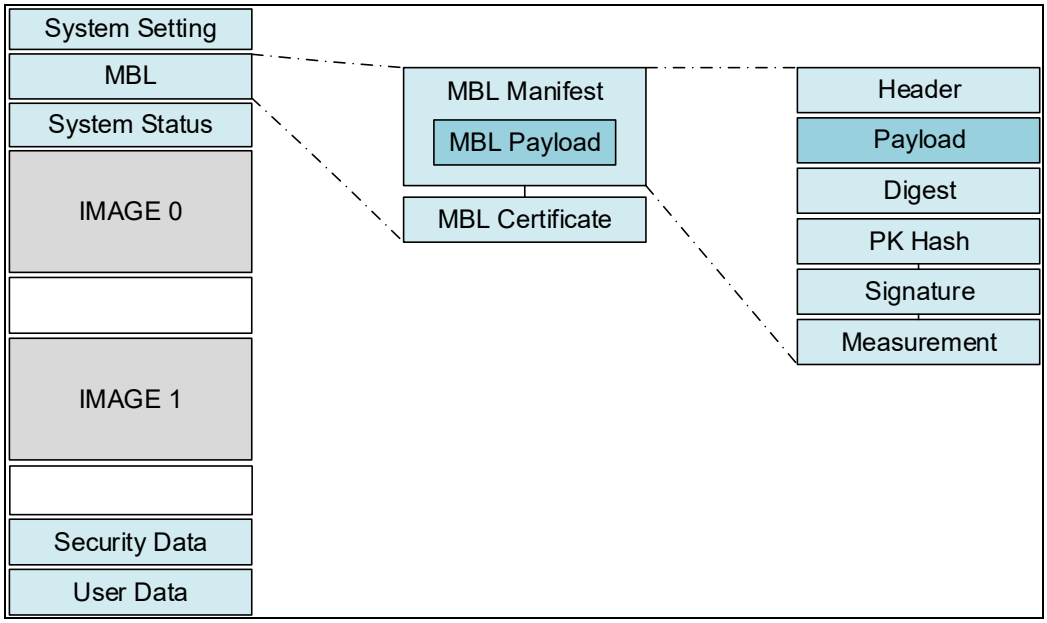
3.3. Firmware encapsulation

This section describes how each firmware is encapsulated and how the related files are configured and generated.

3.3.1. Encapsulation format

Take MBL for example, see ***Figure 3-3. Firmware encapsulation format***, the Payload of the MBL executable is wrapped in the firmware manifest, followed by the certificate.

Figure 3-3. Firmware encapsulation format



3.3.2. Key and certificate generation

Before describing the specific encapsulation, it is important to understand how to generate key pairs and certificates and which key pairs and certificates are required.

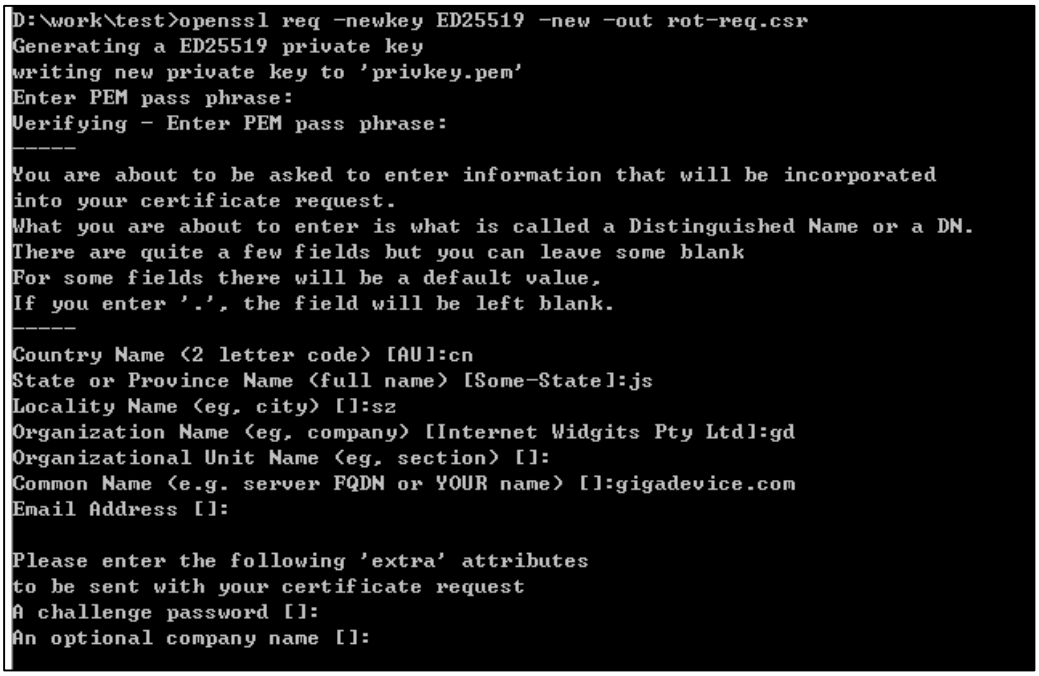
A total of three key pairs need to be generated, which are ROT, MBL and NSPE key pairs. Three certificates need to be generated, which are ROT certificate, MBL certificate, and NSPE certificate.

Open the Windows CMD window, switch to the directory “%SDK% \scripts \ images \” where the key and certificate are saved, and enter the following commands to generate the key pairs and certificate.

ROT key pairs and certificate

Generate ROT key pairs: penssl req -key rot-key.pem -new -out rot-req.csr

Figure 3-4. Generate ROT key pairs



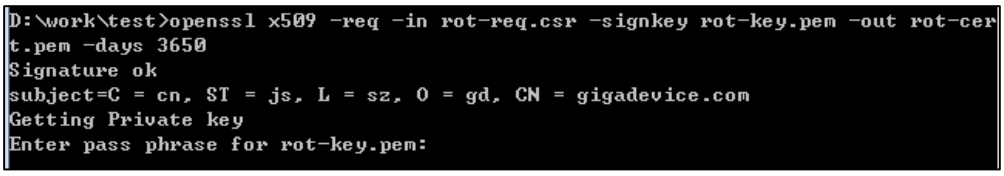
The PEM password needs to be set, the default for this application note is “12345678”. Then fill in the certificate request information in turn, and after success, privkey.pem and rot-req.csr are obtained. privkey.pem is the ROT private key, which is renamed rot-key.pem for the purpose of differentiation. Since the public key can be inferred from the private key, only the private key needs to be saved.

> move privkey.pem rot-key.pem

Generate a ROT certificate that signs the MBL certificate

> openssl x509 -req -in rot-req.csr -signkey rot-key.pem -out rot-cert.pem -days 3650

Figure 3-5. Generate ROT certificate



rot-req.csr is a certificate request that is self-signed using rot-key.pem to generate rot-cert.pem.

Enter the PEM password "12345678" set above, and rot-cert.pem will be generated.

Get ROTPK: > openssl pkey -in rot-key.pem -pubout -out rot-key.txt -text

Use the command above to get rot-key.txt, the contents are shown in ***Figure 3-6. ROT key contents***, saving the “pub” part is the ROTPK that needs to be burned to EFUSE in the future.

Figure 3-6. ROT key contents

```
-----BEGIN PUBLIC KEY-----
MCowBQYDK2VwAyEAXxFfWdjpDFtvVrk9+vPgsQ9U9jk+FcPQ6bIiCsUZMPQ=
-----END PUBLIC KEY-----
ED25519 Private-Key:
priv:
    35:ba:2c:ee:49:5a:c4:8c:67:30:b3:1d:ca:ae:e4:
    0f:fb:56:12:62:aa:2c:3f:9e:9c:86:84:a2:08:50:
    26:29
pub:
    5f:11:5f:59:d8:e9:0c:5b:6f:56:b9:3d:fa:f3:e0:
    b1:0f:54:f6:39:3e:15:c3:d0:e9:b2:22:0a:c5:19:
    30:f4
```

MBL key pairs and certificate

Generate MBL key pairs:> openssl req -newkey ED25519 -new -out mbl-req.csr

Figure 3-7. Generate the MBL key pairs

```
D:\work\test>openssl req -newkey ED25519 -new -out mbl-req.csr
Generating a ED25519 private key
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:js
Locality Name (eg, city) []:sz
Organization Name (eg, company) [Internet Widgits Pty Ltd]:gd
Organizational Unit Name (eg, section) []:gigadevice.com
Common Name (e.g. server FQDN or YOUR name) []:gigadevice.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

The privkey.pem and mpl-req.csr will be obtained after running successfully. to differentiate, here rename privkey.pem to mbl-key.pem.

> move privkey.pem mbl-key.pem

Generate MBL certificate:> openssl x509 -req -in mbl-req.csr -out mbl-cert.pem -signkey mbl-key.pem -CA rot-cert.pem -CAkey rot-key.pem -CAcreateserial -days 3650

Figure 3-8. Generate the MBL certificate

```
D:\work\test>openssl x509 -req -in mbl-req.csr -out mbl-cert.pem -signkey mbl-ke
y.pem -CA rot-cert.pem -CAkey rot-key.pem -CAcreateserial -days 3650
Signature ok
subject=C = cn, ST = js, L = sz, O = gd, OU = gigadevice.com, CN = gigadevice.co
m
Getting Private key
Enter pass phrase for mbl-key.pem:
Getting CA Private Key
Enter pass phrase for rot-key.pem:
```

mbl-cert.pem is generated after the command is successfully run.

NSPE key pairs and certificate

Generate NSPE key pairs: > openssl req -newkey ED25519 -new -out nspe-req.csr

Figure 3-9. Generate NSPE key pairs

```
D:\work\test>openssl req -newkey ED25519 -new -out nspe-req.csr
Generating a ED25519 private key
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:js
Locality Name (eg, city) []:sz
Organization Name (eg, company) [Internet Widgits Pty Ltd]:gd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:gigadevice.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

The privkey.pem and nspe-req.csr will be obtained after running successfully. to differentiate, here rename privkey.pem to nspe-key.pem.

> move privkey.pem nspel-key.pem

Generate NSPE certificate: > openssl x509 -req -in nspe-req.csr -out nspe-cert.pem -signkey nspe-key.pem -CA mbl -cert.pem -CAkey mbl-key.pem -CAcreateserial -days 3650

Figure 3-10. Generate NSPE certificate

```
D:\work\test>openssl x509 -req -in nspe-req.csr -out nspe-cert.pem -signkey nspe
-key.pem -CA mbl-cert.pem -CAkey mbl-key.pem -CAcreateserial -days 3650
Signature ok
subject=C = cn, ST = js, L = sz, O = gd, CN = gigadevice.com
Getting Private key
Enter pass phrase for nspe-key.pem:
Getting CA Private Key
Enter pass phrase for mbl-key.pem:
```

nspe-cert.pem is generated after the command is successfully run.

3.3.3. Configuration file layout

The configuration file layout is in %SDK% \ config \ config_gdm32.h, see ***Figure 3-11. Configuration file layout***. This file is divided into four sections, the first is the base address, the second is the SRAM layout, the third is the FLASH layout, and the fourth is the firmware version. Users can configure according to the actual requirements of the project. The line tagged "Keep unchanged!" cannot be modified because this part is bound to the device hardware.

This file is used in the compilation, linking, and encapsulation of various executable programs. That is, changes to the SRAM and FLASH layout only require changes to this file.

Figure 3-11. Configuration file layout

```
1   /* REGION DEFINE */
2   #define RE_FLASH_BASE_S       0x0C000000    /* !Keep unchanged! */
3   #define RE_FLASH_BASE_NS      0x08000000    /* !Keep unchanged! */
4   #define RE_SRAM_BASE_S        0x30000000    /* !Keep unchanged! */
5   #define RE_SRAM_BASE_NS       0x20000000    /* !Keep unchanged! */
6
7   /* SRAM LAYOUT */
8   #define RE_SHARED_DATA_START  0x0200        /* !Keep unchanged! */
9   #define RE_MBL_DATA_START     0x0A00        /* !Keep unchanged! */
10  #define RE_PROT_DATA_START    0x0A20
11  #define RE_AROT_DATA_START    0xE000
12  #define RE_NSPE_DATA_START    0xE000  // 0x4B00
13
14  /* FLASH LAYEROUT */
15  #define RE_VTOR_ALIGNMENT     0x200         /* !Keep unchanged! */
16  #define RE_SYS_SET_OFFSET     0x0           /* !Keep unchanged! */
17  #define RE_MBL_OFFSET         0x1000        /* !Keep unchanged! */
18  #define RE_SYS_STATUS_OFFSET  0x8000
19  #define RE_IMG_0_PROT_OFFSET  0xA000
20  #define RE_IMG_0_AROT_OFFSET  0x3A000
21  #define RE_IMG_0_NSPE_OFFSET  0x3A000  // 0xA000
22  #define RE_IMG_1_PROT_OFFSET  0xF0000
23  #define RE_IMG_1_AROT_OFFSET  0x120000
24  #define RE_IMG_1_NSPE_OFFSET  0x120000  // 0x100000
25  #define RE_IMG_1_END_OFFSET   0x1D6000
26  #define RE_SST_OFFSET         0x1F5000
27  #define RE_AUDIT_OFFSET       0x1FF000
28
29  /* FW_VERSION */
30  #define RE_MBL_VERSION        0x01000000
31  #define RE_PROT_VERSION       0x01000000
32  #define RE_AROT_VERSION       0x00000001
33  #define RE_NSPE_VERSION       0x01000000
34
```

3.3.4. Executable program generation

Open %SDK% \ MultiProject.uvmpw, there are three projects, MBL, PROT and NSPE in order. After the project has successfully compiled in sequence, the corresponding AXF files, namely mbl.axf, prot.axf, and nspe.axf, are available in their respective output directories. These three files are used to convert the corresponding binary file in the following section.

3.3.5. Executable program encapsulation

The goal of each executable package is to generate a binary file or HEX file from the AXF file that can be downloaded into the FLASH to run. This final file contains the firmware manifest, the firmware certificate, and, if necessary, the firmware encryption.

The encapsulation process has been implemented in afterbuild scripts, which run automatically after the link is compiled to generate the encapsulated firmware. So the details of the encapsulation process is not necessary to be understood in depth, and the contents of ***Encapsulation process*** can be skipped.

Encapsulation script

The %SDK% \ scripts \ imgtool \ directory contains a number of tools written in Python.

- imgtool.py: Adds firmware manifest, firmware certificate to executable binary files
- hextool.py: Convert binary files to Intel HEX for direct download using JLINK.
- aestool.py: AES-CTR encryption for binary files.
- sysset.py: Generates the System Setting binary file sysset.bin.

Encapsulation process

Taking NSPE as an example, the encapsulation process is as follows:

- First, convert the binary BIN file from the AXF file
  - > C:\Keil_v5\ARM\ARMCC\bin\fromelf.exe --bin --8x1 --bincombined --output=.\freertos \nspe.bin .\freertos\output\nspe.axf
- Add the firmware manifest and certificate for nspe.bin
  - > python %IMGTOOL% sign --config %CONFIG_FILE% ^
    -k %CERT_PATH%\nspe-key.pem ^
    -P %KEY_PASSPHRASE% ^
    -t "NSPE" ^
    --cert %CERT_PATH%\nspe-cert.pem ^
    .\freertos\nspe.bin ^
    %OUTPUT_IMAGE_PATH%\nspe-sign.bin
  - In the command, %CONFIG_FILE% is %SDK%\config\config_gdm32.h. The firmware version number needs to be added when assembling the firmware header.
  - nspe-key.pem is the NSPE private key generated in ***NSPE key pairs and certificate*** used to sign the NSPE firmware.

- “12345678” is the MBL PEM password defined earlier.
- “NSPE” is the type of current firmware, along with “MBL”, “PROT”, and “AROT”.
- nspe-cert.pem is the NSPE certificate generated in NSPE key pairs and certificate.

■ Firmware encryption (Optional)

- > python %AESTOOL% --c %CONFIG_FILE% ^
  -t "IMG_%INDEX%_NSPE" ^
  -i %OUTPUT_IMAGE_PATH%\nspe-sign.bin ^
  -o %OUTPUT_IMAGE_PATH%\nspe-sign%AES_SUFFIX%.bin ^
  -k %AESK%
- Since the encryption uses the address as the counter, the corresponding executable start offset address is found from %CONFIG_FILE% according to type "IMG_%INDEX%_NSPE" as the counter.
- %AESK% is the encryption key and must be the same as the AESK burned to EFUSE.

■ Convert to HEX file (optional, if need to download separately via JLINK).

- > python %HEXTOOL% --c %CONFIG_FILE% ^
  -t "IMG_%INDEX%_NSPE" ^
  -e %SREC_CAT% ^
  %OUTPUT_IMAGE_PATH%\nspe-sign%AES_SUFFIX%.bin ^
  %OUTPUT_IMAGE_PATH%\nspe-sign%AES_SUFFIX%.hex
- Use type "IMG_%INDEX%_NSPE" to find the corresponding executable program start offset address and base address from %CONFIG_FILE% to get the absolute address where the firmware is stored.
- Pass the absolute address and nspe-sign-aes.bin to %SREC_CAT% (srec_cat.exe) to generate the Intel HEX file nspe-sign-aes.hex.

Compared to the NSPE encapsulation process, the MBL encapsulation requires an additional step. This is to generate sysset.bin first and then combine it with mbl.bin to generate mbl-sys.bin and then do the above.

AfterBuild script

The procedures in the previous section are written out in xxxx_afterbuild.bat in each Project directory, such as NSPE, and can be implemented through the corresponding nspe_afterbuild.bat.

When these Projects are compiled using KEIL, the file xxxx_afterbuild.bat will be automatically executed after compilation to automatically complete the firmware encapsulation and obtain the corresponding binary file.

3.4. Firmware packaging

3.4.1. Packaging tool

gentool.py: Package the encapsulated firmware together to generate a factory package or upgrade package.

3.4.2. Packaging process

- Factory package
  - *python gentool.py --config ..\..\config\config_gdm32.h --sys_set ..\images\mbl-sys.bin --nspe_0 ..\images\mp-sign.bin --prot_1 ..\images\prot-sign.bin --nspe_1 ..\images\nspe-sign.bin -o ..\images\image-all-release.bin*
  - The factory package is not automatically generated. Run the preceding command in the Windows cmd window to generate the package. Before running the command, compile all executable program segments of the SDK, copy the mp-sign.bin obtained after signing the production test firmware released by the original factory to the %SDK% \ scripts \ images \ directory, then switch the directory in the cmd window to %SDK% \ scripts \ imgtool \ and run the preceding command.
- Upgrade package
  - *> python %GENTOOL% --config %CONFIG_FILE% ^*
    --prot_0 %OUTPUT_IMAGE_PATH%\prot-sign.bin ^
    --nspe_0 %OUTPUT_IMAGE_PATH%\nspe-sign.bin ^
    -o %OUTPUT_IMAGE_PATH%\image-%VERSION%.bin

The upgrade package is automatically generated using the nspe_afterbuild.bat script.

4. Firmware upgrade

The released SDK provides ota_demo.c, which users can refer to when developing their own OTA code. This example code assumes that an HTTP server has been set up and the new firmware has been placed on the server.

Users can change the link address of the directory where the new firmware resides by modifying the macro DOWNLOAD_URL at the beginning of the file. The new firmware file name can be passed by calling ota_demo (bin_name).

Perform the following operations to upgrade the device:

1) First, check whether the IMAGE running on the current device is 0 or 1. If the current IMAGE is 0, the target burn position is 1; otherwise, the target burn position is 0.
2) The device establishes a TCP connection with the HTTP server.
3) The device sends an HTTP Request to the server.
4) The server responds with HTTP response 200 OK and sends the IMAGE firmware in fragments.
5) After receiving the correct response, the device erases the data of the target burn location and successively burns the fragment contents into the FLASH.
6) After sending firmware package data, the server sends the checksum of the firmware package.
7) After receiving the checksum, the device reads the FLASH content of the target burn position to calculate the checksum and compares it with the received checksum.
8) If the verification passes, the IMAGE's status of the target burn location is New, and the IMAGE's status of the current running location is Old.
9) Reboot.
10) MBL verifies the certificate and signature of the new firmware, and if the verification passes, jumps to the new firmware to run.

5. Abbreviations

Table 5-1. Abbreviations

Abbreviation	Meaning
AROT	Application Root of Trust
EFUSE	One Time Program memory
IBL	Immutable Bootloader
MBL	Mutable Bootloader
MP	Mass Production
NSPE	Non-Secure Processing Environment
OTA	Over the Air upgrade
PROT	PSA Root of Trust
PSA	Platform Security Architecture
ROTPK	Root of Trust Public Key
ROM	Read-Only Memory
SPC	Security Protection

6. Revision history

Table 6-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.3, 2023

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.