GigaDevice Semiconductor Inc.

GD32VW553 Secure Boot and Firmware Upgrade Guide

Application Note

AN260

Revision 1.3

(March 2025)

Table of Contents

Abbreviations

Term	Meaning
EFUSE	One Time Program memory
IBL	Immutable Boot Loader
MBL	Main Boot Loader
MBL-PK	Main Boot Loader Public Key
MP	Mass Production
MSDK	Main SDK
OTA	Over the Air upgrade
ROTPK	Root of Trust Public Key
ROM	Read-Only Memory

List of Figures

List of Tables

1. Secure Boot

1.1. Introduction

Secure Boot is designed to ensure the authenticity and integrity of all code from the first instruction executed by the system to the main application. There are two key points: one is the trusted first instruction; the other is to ensure the legality and integrity of the next executable program segment when a program segment jump occurs.

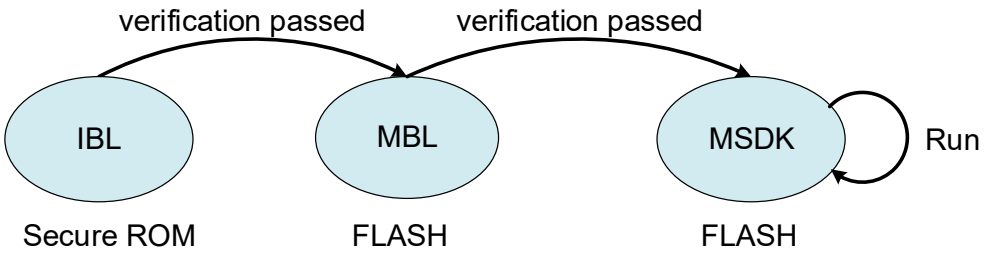
To ensure that the first instruction is always secure, the first executable program segment is solidified in ROM, referred to as Immutable Boot Loader (IBL). When the boot mode is locked to secure boot, the system will jump to IBL to execute the first instruction upon power-up or reset, and it cannot be tampered with in any way. Therefore, the first instruction possesses root trust.

The second executable program segment is placed at the beginning of FLASH and is referred to as the Main Boot Loader (MBL). The Immutable Boot Loader (IBL) is responsible for verifying the legality and integrity of the MBL, it can jump to the MBL only after the verification is passed.

The subsequent executable program segment, referred to as MSDK, will be verified by MBL, and can the jump to MSDK only after successful verification.

The boot process diagram is shown in Figure 1-1.

Figure 1-1 Boot Process



The EFUSE stores the hash value of the Root of Trust Public Key (ROTPK) which verify the legality of the public key carried by the first FLASH executable program segment, and then verify the digest signature or certificate signature with the verified public key.

1.2. FLASH

FLASH map is shown in Figure 1-2. The default configuration of GD32VW553 FLASH size is 4M Bytes, and we will take 4M as an example.

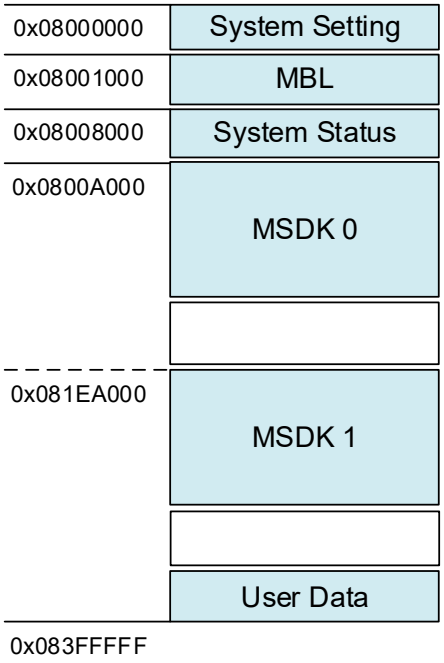
The System Setting primarily stores the initial version information of MBL and MSDK, as

well as starting offsets, etc. The initial version information will be read and stored in the System Status upon the first startup, and then immediately will be locked. Subsequent updates can only advance the version number, not regress.

The System Status is maintained using a PING/PONG method, with content stored in TLV format, including a checksum and encrypted with AES. It is primarily used to store version information and the operational status of MSDK 0/1. The version information is used to prevent rollback attacks, while the MSDK operational status determines whether the current boot is preparing to verify MSDK 0 or MSDK 1. It also stores other boot-related content and allows users to add custom storage content.

The starting position of MSDK 0 is fixed and cannot be modified, but the starting position of MSDK 1 can be adjusted based on the size of FLASH and User Data. MSDK 0 and MSDK 1 will be used for ping-pong upgrades in the future, so it is recommended that the spaces reserved for both are roughly the same size. Taking the released SDK as an example, the current starting position we have set for MSDK 1 is 0x081EA000.

Figure 1-2 Flash Map



1.3. SRAM

SRAM map is shown in Figure 1-3. The global variables used by ROM occupy 512 bytes.

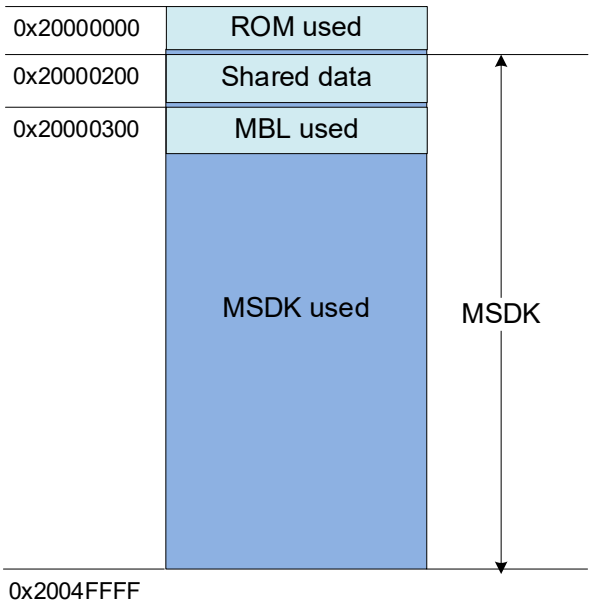
Shared data is the information passed from ROM to MBL, including ROM version information, ROTPK hash value, and MBL public key, etc.

MBL used refers to the heap and stack utilized by MBL during runtime.

MSDK used refers to the SRAM space available for MSDK utilization.

It should be noted that in Figure 1-3, the SRAM used by MSDK overlaps with the Shared data area and MBL area because after the execution of MBL, both spaces can be released for MSDK use. Therefore, the starting address of the SRAM used by MSDK is also 0x20000200.

Figure 1-3 SRAM Map



1.4. Configuration file planning

The planning configuration file is located at %SDK%\config\config_gdm32.h, as shown in Figure 1-4. This file is divided into four sections: the first section is the base address, the second section is the SRAM layout, the third section is the FLASH layout, and the fourth section is the firmware version. Users can modify configuration according to the actual needs of the project. Lines marked with "Keep unchanged!" should not be modified, as they are hardware-bound to the chip.

During the compilation, linking, and packaging process of various executable programs, this file is used. In other words, changes to the layout of SRAM and FLASH only require modifications to this single file.

Attention should be paid to the line RE_MBL_OFFSET. When set to 0, the MBL firmware is compiled to boot from FLASH Boot. If Secure ROM Boot is required, this macro needs to be modified to 0x1000.

Figure 1-4 Configuration File

```
/* REGION DEFINE */
#define RE_FLASH_BASE           0x08000000      /* !Keep unchanged! */
#define RE_SRAM_BASE            0x20000000      /* !Keep unchanged! */

/* SRAM LAYOUT */
#define RE_MBL_DATA_START       0x300           /* !Keep unchanged! */
#define RE_IMG_DATA_START       0x200           /* !Keep unchanged! */

/* FLASH LAYEROUT */
#define RE_VTOR_ALIGNMENT       0x200           /* !Keep unchanged! */
#define RE_SYS_SET_OFFSET       0x0             /* !Keep unchanged! */
#define RE_MBL_OFFSET           0x0             /* 0x0:    Boot from MBL */
                                                /* 0x1000: Boot from ROM */
#define RE_SYS_STATUS_OFFSET    0x8000          /* !Keep unchanged! */
#define RE_IMG_0_OFFSET         0xA000          /* !Keep unchanged! */
#define RE_IMG_1_OFFSET         0x1EA000
#define RE_IMG_1_END            0x3CA000        /* reserved 196KB for user data */
#define RE_NVDS_DATA_OFFSET     0x3FB000        /* reserved 20KB for nvds data */
#define RE_END_OFFSET           0x400000        /* equal to flash total size */

/* FW_VERSION */
#define RE_MBL_VERSION          0x01000002
#define RE_IMG_VERSION          0x01000002
```

2. Hardware configuration

To enable the secure boot feature, it is necessary to configure specific bits in the EFUSE.

- Boot mode selection
  - Enable the secure boot startup option. At this time, both BOOT0 and BOOT1 pins need to be pulled low, indicating booting from FLASH. A high BOOT0 indicates booting from the ROM Bootloader, and both BOOT0 and BOOT1 being high indicates booting from SRAM.
  - Secure ROM boot is shared with the FLASH mode through the pin configuration.
  - EFUSE_CTL0: BIT 0 set to 1 indicates booting from Secure ROM, while set to 0 indicates booting from FLASH.
  - If Secure Boot is enabled, during the factory stage, the BOOT mode can be permanently fixed, meaning that the system can only boot from the Secure ROM. The following bits can be set for this purpose.
  - EFUSE_CTL0: written as 0x2B
- Verification option
  - The verification option determines whether the ROM will verify the authenticity of the MBL firmware and certificates; if not set, verification will not occur, and secure boot is not truly enabled.
  - EFUSE_CTL1: BIT6 represents verification of firmware signature, BIT6 + BIT7 represents verification of certificate + firmware signature.
- ROTPKH
  - ROTPKH is used to verify the legality of the ROTPK public key carried by the MBL firmware, and the ROTPK public key is used to verify the signature of the MBL digest or the certificate signature. The generation method of the ROTPK key pair and ROTPKH is referred to in section 3.2.3.
  - EFUSE_ROTPKKEYx: a total of 32 bytes.
  - EFUSE_CTL1: BIT2 locks the ROTPKH from being written again.
- AES KEY
  - Optional. If firmware encryption is selected, it is also necessary to burn the firmware encryption key and set a Lock bit. After the key is burned, it can be read out to verify whether it is written correctly, but once the Lock bit is set, the software can no longer read it.

    Note: After flashing, only firmware encrypted with AESK can be executed, and it is not possible to revert.
  - EFUSE_AESKEYx: 16 bytes in total.
  - After setting BIT5 AESEN to 1 in EFUSE_USERCTL, the software is unable to read or write the AESKey.

During the development phase, GD32AllInOneProgrammer.exe can be used to program the EFUSE bits. In the mass production phase, MassProductionTool_CMD.exe can be used

to perform programming via command line.

3. Firmware package creation

3.1. Firmware package format

There are generally two types of firmware packages, one is the factory package, and the other is the upgrade package.

3.1.1. Factory package

The factory package refers to the firmware package burned into FLASH during the mass production of modules. The contents of the factory package are shown in Figure 3-1.

The System Setting area stores system configuration, which is automatically generated during the compilation of MBL. The MBL area contains the MBL firmware as well as signatures and certificates, with certificates being optional. The System Status area is padded with 0xFF to fill empty spaces. These three areas are automatically generated by the script mbl_afterbuild.bat during the compilation of MBL and are merged into mbl-sys.bin, stored in %SDK%\scripts\images\.

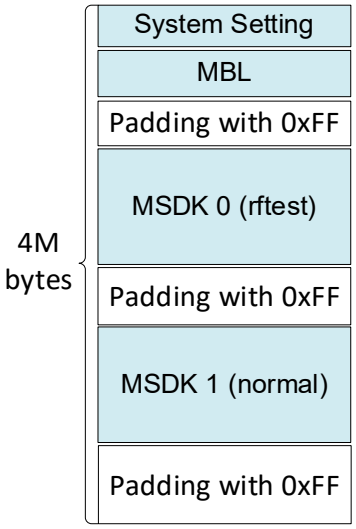
MSDK 0 stores production test firmware and signatures, etc., namely rftest.bin, which is compiled from MSDK when CONFIG_RF_TEST_SUPPORT in rftest_cfg.h is enabled, and is automatically generated in %SDK%\scripts\images\. If Secure Boot is enabled (RE_MBL_OFFSET = 0x1000), the compilation will automatically add header and footer information, see 3.2 Firmware encapsulation for details.

MSDK 1 stores the user firmware (msdk.bin). After disable CONFIG_RF_TEST_SUPPORT and recompiling MSDK, the msdk.bin will be generated and stored in %SDK%\scripts\images\. Similarly, if Secure Boot is enabled (RE_MBL_OFFSET = 0x1000), the compilation will automatically add header and footer information.

When MBL, MSDK 0 (RFTEST), and MSDK 1 (Normal) are compiled in sequence, an image-all-mp.bin file is automatically generated in the %SDK%\scripts\images\ directory, combining the three to serve as the factory mass production firmware.

After factory programming, it defaults to jump to the production testing firmware MSDK 0. Once the production testing is completed, a serial port command is entered to switch to the user firmware MSDK 1, and after a simple test, it is ready for shipment.

Figure 3-1 Schematic diagram of the factory firmware package

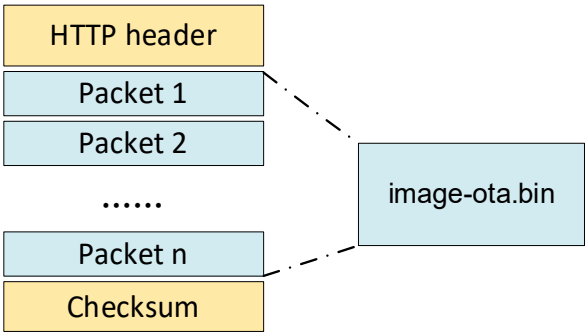


3.1.2. Firmware update package

The upgrade package refers to the image-ota.bin, also known as msdk.bin, that is burned into the MSDK 0 or MSDK 1 area. After the MSDK compilation, the image_afterbuild.bat is automatically executed to rename the file and place it in the %SDK% \scripts\images\ directory.

HTTPS protocol is commonly used to download firmware upgrade packages from remote service areas. An example of an upgrade package is shown in Figure 3-2 using the summary signature mode.

Figure 3-2 Firmware Upgrade Package



3.2. Firmware encapsulation

After introducing the firmware package, let's take a look at how each sub-firmware is encapsulated.

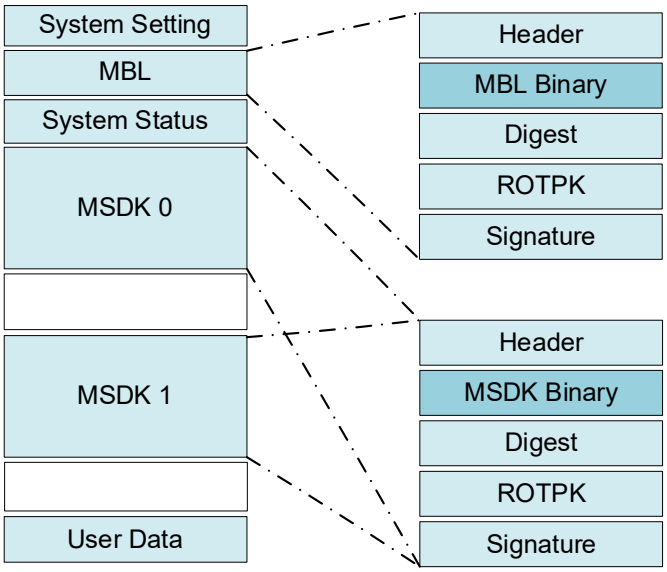
3.2.1. Packaging format

Firmware packaging formats are divided into two types based on whether certificate

authentication is selected: one is called non-certificate firmware packaging, as shown in Figure 3-3; the other is called certificate-enabled firmware packaging, as shown in Figure 3-4.

Firmware encapsulation without a certificate includes: firmware header, firmware itself, firmware digest, verification signature public key, and firmware digest signature. The data source for the firmware digest includes the firmware header and the firmware itself. The ROTPK is used to verify the signature of the firmware digest. In this mode, we only need to generate a pair of ROT keys, the ROT private key is used to sign the MBL firmware and MSDK firmware, and the ROTPK is used to verify the signature, as shown in Figure 3-5.

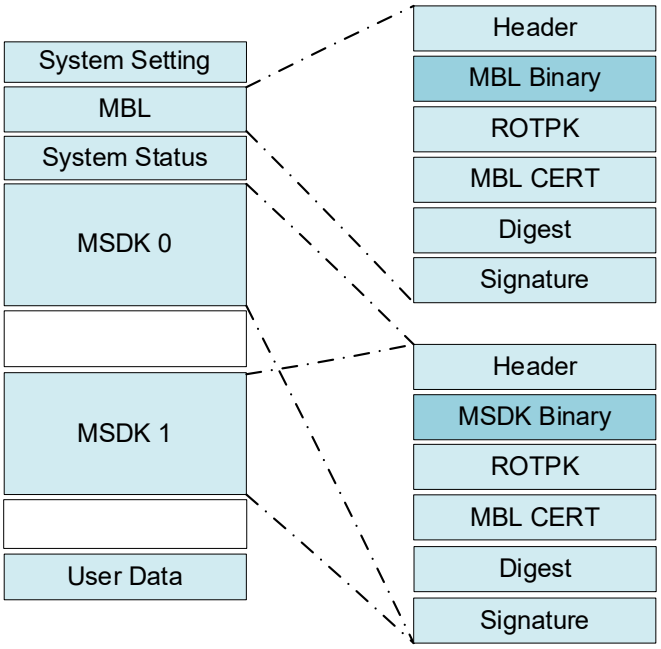
Figure 3-3 Firmware packaging format without certificate



The firmware encapsulation with a certificate includes: firmware header, firmware itself, firmware digest, verification certificate public key, certificate, and firmware digest signature. The data source for the firmware digest includes the firmware header and the firmware itself. The ROTPK is used to verify the signature of the certificate, while the MBL-PK included in the certificate is used to verify the signature of the firmware digest. This pattern requires at least two sets of keys and certificates to be generated, as shown in Figure 3-6.

MSDK can use a separate set of keys and certificates, or share a set with MBL. Currently, the demonstration uses a shared set, hence the MSDK encapsulation contains ROTPK and MBL CERT. Otherwise, the MSDK encapsulation would contain MBL-PK and MSDK CERT, with MSDK-PK included in the MSDK CERT, and the MSDK private key is used to sign the MSDK digest.

Figure 3-4 Firmware packaging format with certificate



Firmware encapsulation is completed by the automatically executed scripts mbl_afterbuild.bat and image_afterbuild.bat after compilation. Users only need to configure the %SDK%\config\config_gdm32.h, generate the corresponding key pairs and certificates, and place them in the %SDK%\scripts\certs\ directory. It is best to configure the parameters passed to the xxx_afterbuild.bat script when called by the IDE, and the corresponding firmware encapsulation will be automatically generated.

Note: Currently, both ECDSA 256 and ED25519 algorithms are supported for signature and authentication. If an ECDSA256 key pair and certificate are selected, the IDE needs to pass the argument "ECDSA256" when calling xxx_afterbuild.bat. For example:

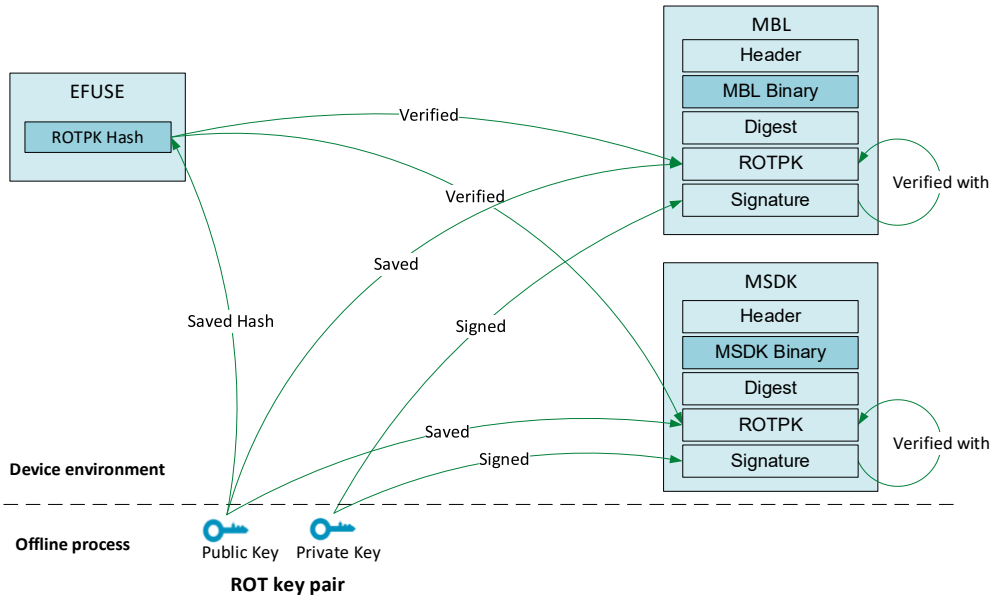
cmd /C "..\..\..\image_afterbuild.bat riscv-nuclei-elf- ECDSA256 CERT ${eclipse_home}/../Tools/OpenOCD/xpack-openocd-0.11.0-3/bin ${ProjDirPath}/../../../../ " " "
If the certificate authentication method is selected, the parameter should be passed as "CERT"; otherwise, it should be passed as "IMG".

3.2.2. Boot the chain of trust.

The public key of the ROT key pair is referred to as ROTPK, and the public key of the MBL key pair is referred to MBL-PK.

As shown in Figure 3-5, when secure boot does not require certificate authentication, a set of ROT keys can be used for signing and verifying signatures. The ROT private key is used to sign the digests of MBL and MSDK, and the ROTPK is used to verify the signatures.

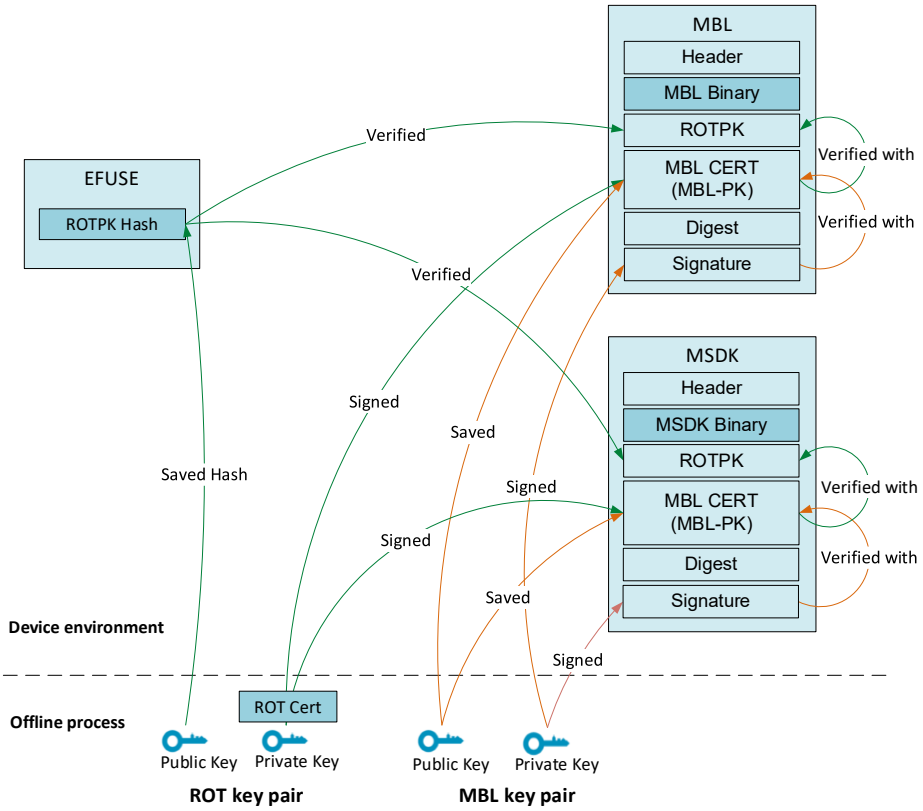
Figure 3-5 Certificate-less Authentication Trust Chain



If secure boot supports certificate authentication, as shown in Figure 3-6. First, a pair of ROT and MBL keys and two certificates need to be generated. Then, the ROT certificate is used to sign the MBL-PK certificate, and the MBL private key is used to sign the digests of MBL and MSDK. The method for generating key pairs can be referred to Section 3.2.3.

During secure boot, the ROM first uses the ROTPK Hash in EFUSE to verify the legitimacy of the ROTPK carried by the MBL, then uses the ROTPK to verify the legitimacy of the MBL-PK certificate. If the verification passes, the MBL-PK is extracted from it, and then the MBL-PK is used to verify the legitimacy of the MBL digest signature. If all verifications pass, the system jumps to the MBL for execution. After jumping to the MBL, the MBL uses the same method to verify the legitimacy of the MSDK.

Figure 3-6 Certificate Authentication Trust Chain



3.2.3. Key and certificate generation

Currently, we will use the ROT key pair and MBL key pair, as well as the ROT certificate and MBL certificate. It is necessary to download and install the OpenSSL Windows tool. Then, open the Windows CMD window and enter the following commands to generate the required key pairs and certificates.

ROT key pairs and certificates.

Generate ROT key pair.

>openssl req -key rot-key.pem -new -out rot-req.csr

Figure 3-7 Generate ROT Key Pair

```
D:\work\test>openssl req -newkey ED25519 -new -out rot-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 []:
```

During the process, a PEM password needs to be set, with "12345678" being the default in this text. Then, fill in the certificate request information in sequence. Upon success, you will obtain privkey.pem and rot-req.csr. The privkey.pem is the ROT private key, which is renamed to rot-key.pem for differentiation. Since the public key can be derived from the private key, only the private key needs to be saved.

Move privkey.pem to rot-key.pem.

Generate the ROT certificate, which then signs the MBL certificate.

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

Figure 3-8 Generate ROT Certificate

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

rot-req.csr is a certificate request; we use the self-signed rot-key.pem to generate rot-cert.pem. This step requires inputting the PEM password "12345678" that was set above, after which rot-cert.pem will be generated.

Navigate to the directory %SDK%\scripts\imgtool\, open the Windows CMD, and execute the following command:

imgtool.py getpub -k ..\certs\ecdsa256\rot-key.pem --hash-algo SHA256 1

The required ROTPKH to be written into EFUSE can be obtained, as shown in test_rotpk_hash[] in Figure 3-9.

Figure 3-9 Obtaining ROTPKH

```
D:\Work\GD32VW553\GDM32103_ALL\scripts\imgtool>imgtool.py getpub -k ..\certs\ecdsa256\rot-key.pem --sha256 1
/* Autogenerated by imgtool.py, do not edit. */
const unsigned char ecdsa_pub_key[] = {
    0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2a, 0x86,
    0x48, 0xce, 0x3d, 0x02, 0x01, 0x06, 0x08, 0x2a,
    0x86, 0x48, 0xce, 0x3d, 0x03, 0x01, 0x07, 0x03,
    0x42, 0x00, 0x04, 0x1f, 0x96, 0xcb, 0x28, 0xe4,
    0x23, 0x77, 0xb4, 0x96, 0xd3, 0xfd, 0x11, 0x13,
    0x1f, 0x7f, 0xef, 0x2f, 0x55, 0xb8, 0x68, 0x09,
    0x29, 0xf5, 0x65, 0xb6, 0x59, 0x68, 0x7a, 0xf7,
    0xad, 0xa8, 0x0b, 0x11, 0xd7, 0x2f, 0x3b, 0x7a,
    0xee, 0x4b, 0x1e, 0x1a, 0x2a, 0x70, 0x12, 0x3b,
    0xeb, 0x17, 0x15, 0x57, 0x2b, 0x17, 0x10, 0xd7,
    0xc2, 0x72, 0x58, 0xc4, 0xbc, 0x51, 0xc5, 0xca,
    0x15, 0x74, 0x11,
};
const unsigned int ecdsa_pub_key_len = 91;
static const uint8_t test_rotpk_hash[] = {
    0xbc, 0xc0, 0x9d, 0x37, 0xaf, 0x86, 0xdb, 0xc6,
    0x84, 0x9d, 0x2e, 0x10, 0x51, 0x33, 0x55, 0x87,
    0x13, 0xbc, 0xc4, 0xb1, 0x21, 0x83, 0x52, 0xb5,
    0xc4, 0xa3, 0x76, 0x8b, 0x42, 0x22, 0xf8, 0x28,
};
```

MBL key pairs and certificates

Generate MBL key pair.

>openssl req -newkey ED25519 -new -out mbl-req.csr

Figure 3-10 Generate MBL key pair

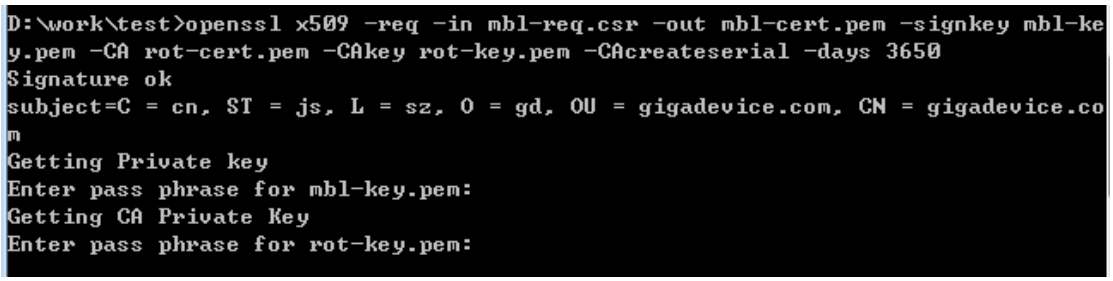
```
D:\work\test>openssl req -newkey ED25519 -new -out mbl-req.csr
Generating a ED25519 private key
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
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 []:
```

Upon successful execution, privkey.pem and mbl-req.csr will be obtained. To distinguish them, privkey.pem is renamed to mbl-key.pem.

Move privkey.pem to 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 3-11 Generate MBL certificate



Upon successful execution, a file named mbl-cert.pem is generated.

4. Firmware upgrade

The SDK released provides an ota_demo.c, which users can refer to when developing their private OTA code.

For testing the server, it is recommended to use the Python3 HTTP Server. Once Python3 is installed, it can be used directly. The command is as follows, where "HostIP" is the IP of the host running this command. Note that the device must be on the same local network as the host.

python -m http.server 80 --bind HostIP

The example code assumes that the user has set up an HTTP server and placed the new firmware in the server's root directory. Users can input commands through the device's serial port to perform OTA testing.

ota_demo test_ap password 192.168.3.100 image-ota.bin

"test_ap" and "password" are the SSID and password of the local area network AP to be connected, "192.168.3.100" is the IP address of the host running the HTTP Server, and "image-ota.bin" is the upgrade package placed in the root directory of the HTTP Server.

The primary firmware employs a ping-pong buffering strategy for storage. If currently operating in the MSDK 0 region, the upgrade firmware is burned to the MSDK 1 region. Conversely, if operating in the MSDK 1 region, the upgrade firmware is burned to the MSDK 0 region. During firmware upgrade, only the checksum or Hash of the firmware downloaded from the cloud to FLASH is verified to ensure the integrity of the firmware download and the correctness of the burn to FLASH. The Image Status corresponding to MSDKx is then marked as NEW and stored in the System Status area.

After the upgrade is completed and a restart is initiated, the MBL will first check the Image Status of MSDK 0 and MSDK 1. If a NEW label is detected, it will prioritize verification, specifically validating the firmware signature or certificate of the MSDK. If the verification is successful, the corresponding Image Status will be marked as VERIFY_OK, and the system will jump to run the NEW firmware. If the verification fails, the corresponding Image Status will be marked as VERIFY_FAIL, entering a While(1) loop. Upon restart, the system will select to verify the other MSDK firmware, which is the old firmware. If the verification is successful, it will run.

The steps for device upgrade are as follows:

1) The device first determines whether the currently running MSDK is 0 or 1. If it is 0, the target burn position is 1; otherwise, it 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 an HTTP response 200 OK, and sends out the MSDK Manifest

in chunks.

5) After receiving the correct response, the device erases the data at the target programming location and sequentially writes the sliced content into FLASH.
6) After the server finishes sending the firmware package data, it sends the checksum of the firmware package.
7) After receiving the checksum, the device reads the FLASH content at the target programming location to calculate the checksum for comparison.
8) If the verification is successful, the Image Status of the target burn location is marked as NEW, and the Image Status of the current running location is marked as OLD.
9) reset
10) MBL verifies the signature of the new firmware; if the verification passes, it jumps to run the new firmware, otherwise, it restarts and returns to running the old firmware.

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Jul.10.2024
1.1	Modify description	Feb.26.2025
1.2	Modify ROTPK	Mar.6.2025
1.3	Add OTA details	Mar.20.2025

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 according to the laws of the People’s Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. 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.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its 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 Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.