GigaDevice Semiconductor Inc.

GD32VW553 安全启动和固件升级指南

应用笔记

AN260

1.3 版本

(2025 年 3 月)

目录

缩略语

缩略语	英文含义	中文含义
EFUSE	One Time Program memory	只能烧写一次的存储空间
IBL	Immutable Boot Loader	不可变引导
MBL	Main Boot Loader	主引导
MBL-PK	Main Boot Loader Public Key	主引导公钥
MP	Mass Production	量产，这里指模组产测
MSDK	Main SDK	主应用程序
OTA	Over the Air upgrade	空口升级
ROTPK	Root of Trust Public Key	根信任公钥
ROM	Read-Only Memory	只读存储

图索引

表索引

1. 安全启动

1.1. 简介

安全启动是为了确保从系统运行的第一条指令到主应用程序之间所有代码的合法性和完整性。这里有两个关键点：一是可信的第一条指令；二是发生程序段跳转时，确保下一个可执行程序段的合法性和完整性。

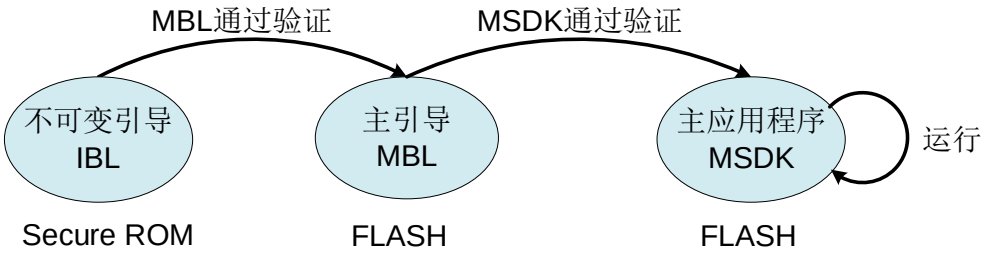
为了保证第一条指令始终是安全的，我们将第一个可执行程序段固化在 ROM 中，称之为不可变引导（IBL）。当启动方式被锁定为安全启动后，不论是上电还是重启，系统都会跳转到 IBL 来运行第一条指令，任何方式不能篡改。因此第一条指令就拥有了根信任。

第二个可执行程序段，放在 FLASH 开头，称之为主引导（MBL）。IBL 会负责验证 MBL 的合法性和完整性，验证通过后，才能跳转到 MBL。

后一级可执行程序段，我们称之为 MSDK，将由 MBL 来负责验证签名，验证成功后，才能跳转到 MSDK 运行。

引导过程示意图见图 1-1。

图 1-1. 引导过程



EFUSE 中会存储信任根公钥（ROTPK）的哈希值，用来验证第一个 FLASH 可执行程序段的携带的公钥是否合法，再用公钥来验证摘要签名或者证书签名。

1.2. FLASH 规划

FLASH 规划见图 1-2。GD32VW553 常规配置 FLASH 大小是 4M Bytes，我们就以 4M 为例。

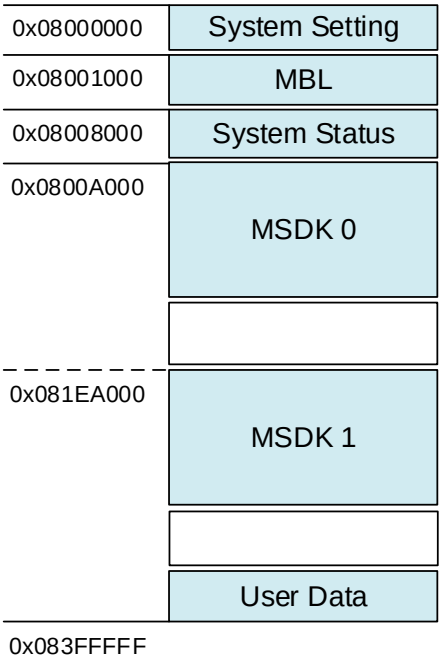
System Setting 主要存放 MBL 和 MSDK 的初始版本信息以及起始偏移量等。初始版本信息会在第一次启动时读取并存储到 System Status 中，然后立刻锁定。之后的更新，版本号只能前进，不能后退。

System Status 采用 PING/PONG 的方式维护，存放内容采用 TLV 格式，存储校验和，并采用 AES 加密。它主要用来存放版本信息以及 MSDK 0/1 的运行状态。版本信息用来防止回退攻击，MSDK 运行状态用来决定当前启动是准备校验 MSDK 0 还是 MSDK 1。它还存储了

一些其他与引导相关的内容，同时也允许用户自定义添加新的存储内容。

MSDK 0 的起始位置是固定的，不能修改，但是 MSDK 1 的起始位置，可以根据 FLASH 大小和 User Data 的大小做调整。MSDK 0 和 MSDK 1 将来是乒乓升级使用，因此建议两者留的空间差不多大小，以发布的 SDK 为例，MSDK 1 的起始位置目前我们设置的是 0x081EA000。

图 1-2. FLASH 规划



1.3. SRAM 规划

SRAM 规划见图 1-3。其中 ROM 使用到的全局变量占用了 512 字节。

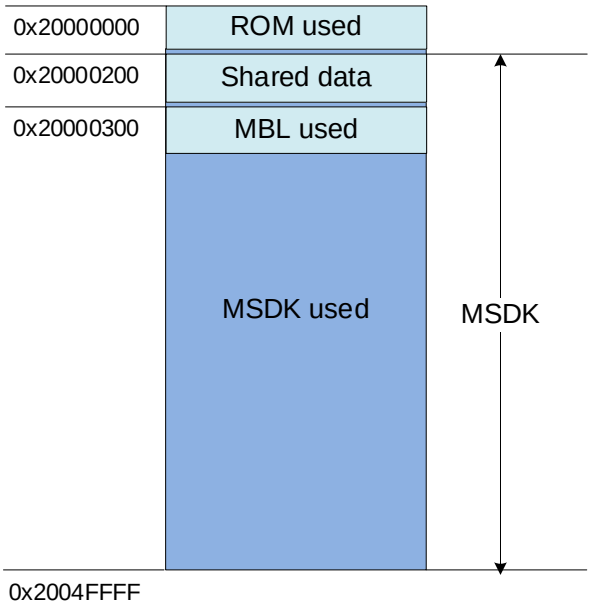
Shared data 是 ROM 传递给 MBL 的信息，包括 ROM 版本信息，ROTPK 哈希值以及 MBL 公钥等信息。

MBL used 是 MBL 运行时用到的堆和栈。

MSDK used 是 MSDK 可以使用的 SRAM 空间。

需要注意的是，图 1-3 中，MSDK 使用的 SRAM 跟 Shared data 区域以及 MBL 区域有重合，是因为在 MBL 运行之后，这两块空间都可以释放出来给 MSDK 使用。因此 MSDK 使用的 SRAM 起始位置也是 0x20000200。

图 1-3. SRAM 规划


0x20000000
ROM used
0x20000200
Shared data
0x20000300
MBL used
MSDK used
MSDK
0x2004FFFF

1.4. 规划配置文件

规划配置文件即%SDK%\config\config_gdm32.h，见图 1-4。本文件分为四段，第一段是基地址，第二段是 SRAM 规划，第三段是 FLASH 规划，第四段是固件版本。用户可以根据项目的实际需要来进行配置。标注为“Keep unchanged!”的行不能修改，这部分是跟芯片硬件绑定的。

在各个可执行程序进行编译、链接、封装的过程中，都会用到这个文件。也就是说，SRAM 和 FLASH 规划的改动只需要修改这一个文件。

需要注意 RE_MBL_OFFSET 这一行，当设为 0 的时候，MBL 固件是按照从 FLASH Boot 进行编译的，如果需要 Secure ROM Boot，需要将这个宏修改为 0x1000。

图 1-4. 规划配置文件

```
/* 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. 硬件设定

如果要启用安全启动功能，需要设置 EFUSE 特定栏位。

- 启动方式选择
  - 将安全启动的引导选项打开。此时 BOOT0 和 BOOT1 的 PIN 需要都拉低，代表从 FLASH 启动，BOOT0 为高代表从 ROM Bootloader 启动，BOOT0 和 BOOT1 都为高代表从 SRAM 启动。
  - 而 Secure ROM 启动是跟 FLASH 模式复用 PIN 脚设定的。
  - EFUSE_CTL0：BIT 0 为 1 代表从 Secure ROM 启动，为 0 代表从 FLASH 启动
  - 如果启用安全启动后，出厂阶段，可以将 BOOT 模式彻底固定下来，也就是说仅能从 Secure ROM 启动，可以设置以下栏位。
  - EFUSE_CTL0：写为 0x2B
- 设置验证选项
  - 验证选项决定 ROM 是否会验证 MBL 固件和证书的合法性，如果未设置，就不会验证也就没有真正使能安全启动。
  - EFUSE_CTL1：BIT6 代表验证固件签名，BIT6 + BIT7 代表会验证证书+固件签名
- 设置 ROTPKH
  - ROTPKH 用来验证 MBL 固件携带的 ROTPK 公钥是否合法，ROTPK 公钥用来验证 MBL 的摘要签名或者证书签名。ROTPK 密钥对以及 ROTPKH 的产生方法请参考 3.2.3。
  - EFUSE_ROTPKKEYx：共 32 字节
  - EFUSE_CTL1：BIT2 锁定 ROTPKH 不能再被写
- 设置 AESK
  - 可选项。如果选择对固件加密，还需要烧写固件加密密钥以及对一个的 Lock 位。密钥烧写进去后可以读出来校验是否写正确，但是一旦把 Lock 位置上，软件就再也无法读取。

    注意：烧写后就只能运行 AESK 加密的固件，无法再退回。
  - EFUSE_AESKEYx：共 16 字节
  - EFUSE_USERCTL：BIT5 AESEN 写 1 后，软件无法读写 AESKey

开发阶段可以使用 GD32AllInOneProgrammer.exe 来对 EFUSE 栏位进行烧录。量产阶段可以使用 MassProductionTool_CMD.exe，使用命令行方式进行烧录。

3. 固件包制作

3.1. 固件包格式

通常固件包有两种，一种是出厂包，一种是升级包。

3.1.1. 出厂包

出厂包是指在模组量产的时候，烧录到 FLASH 的固件包。出厂包包含的内容参见图 3-1。

System Setting 区域存放系统配置，这个在编译 MBL 时自动生成，MBL 区域存放 MBL 固件以及签名和证书，证书可选，System Status 区域使用 0xFF 补空，这三个会在编译 MBL 时由脚本 mbl_afterbuild.bat 自动生成，并合并为 mbl-sys.bin，存储在%SDK%\scripts\images\中。

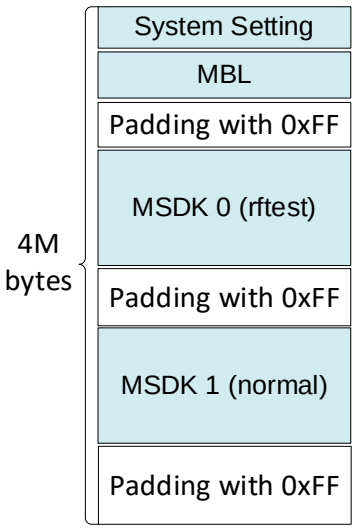
MSDK 0 存放产测固件及签名等内容，即 rftest.bin，它是将 rftest_cfg.h 中 CONFIG_RF_TEST_SUPPORT 打开后编译 MSDK 得到的，自动生成在%SDK%\scripts\images\中。如果使能了 Secure Boot （RE_MBL_OFFSET = 0x1000），编译时会自动加头部加尾部信息，详见 3.2 固件封装。

MSDK 1 存放用户固件(msdk.bin)，将 CONFIG_RF_TEST_SUPPORT 关闭后再次编译 MSDK，会生成 msdk.bin 存放在%SDK%\scripts\images\中。同样，如果使能了 Secure Boot （RE_MBL_OFFSET = 0x1000），编译时会自动加头部加尾部信息。

当依次编译 MBL，MSDK 0(RFTEST)，MSDK 1(Normal)后，会在%SDK%\scripts\images\中自动生成 image-all-mp.bin，它将三者合在一起，作为出厂量产固件。

工厂烧录之后，默认跳转到产测固件 MSDK 0。当产测完成后，再输入串口命令切换到用户固件 MSDK 1，简单测试后即可出厂。

图 3-1. 出厂固件包示意图

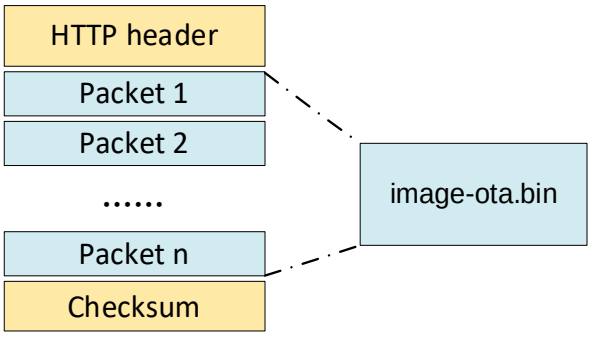


3.1.2. 升级包

升级包是指烧录到 MSDK 0 或者 MSDK 1 区域的 image-ota.bin，也即 msdk.bin，在 MSDK 编译后自动运行的 image_afterbuild.bat 中会进行更名，并放置到%SDK%\scripts\images\目录下。

通常使用 HTTPS 协议来从远端服务区下载固件升级包。以摘要签名模式为例，升级包示意见下图 3-2。

图 3-2. 升级固件包示意图



3.2. 固件封装

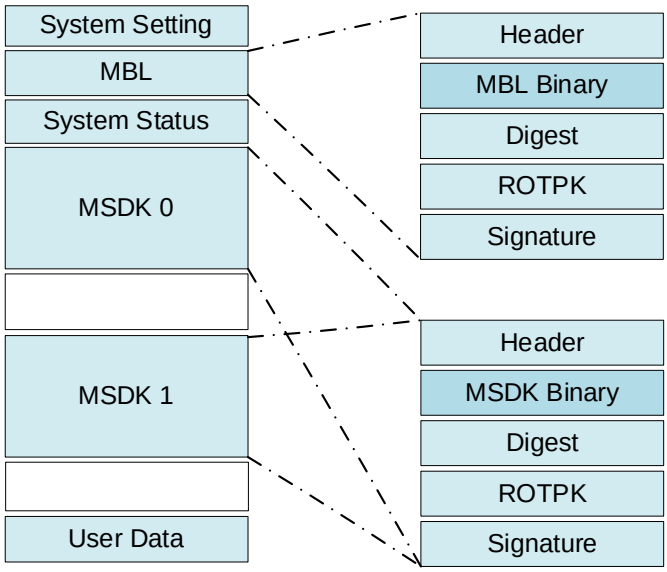
介绍完固件包，来看一下每个子固件是如何封装的。

3.2.1. 封装格式

固件封装格式，根据是否选择证书认证，分为两种格式：一种称之为无证书固件封装，如图 3-3 所示；一种称之为带证书固件封装，如图 3-4 所示。

无证书固件封装包括：固件头部、固件本身、固件摘要、验证签名的公钥以及固件摘要签名。其中固件摘要的数据源包含固件头部和固件本身。ROTPK 用来验证固件摘要的签名。这个模式下我们只需要产生一组 ROT 密钥对，ROT 私钥用来给 MBL 固件和 MSDK 固件签名，ROTPK 用来验证签名，如图 3-5 所示。

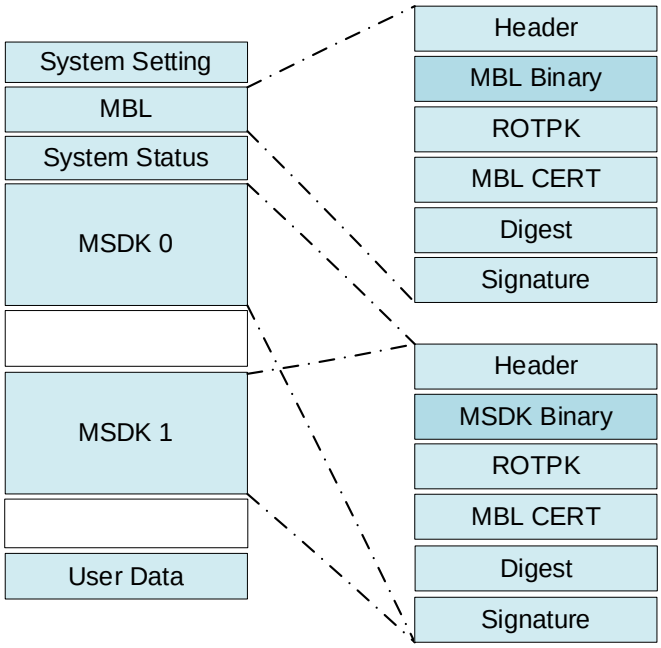
图 3-3. 无证书固件封装格式



带证书固件封装包括：固件头部、固件本身、固件摘要、验证证书的公钥、证书以及固件摘要签名。其中固件摘要的数据源包含固件头部和固件本身。ROTPK 用来验证证书的签名，而证书中包含的 MBL-PK 用来验证固件摘要的签名。这个模式至少需要产生两组密钥和证书，如图 3-6 所示。

MSDK 可以另外使用一套密钥和证书，也可以跟 MBL 共用一套。目前演示的是共用一套，因此在 MSDK 封装中放入的是 ROTPK 和 MBL CERT。否则 MSDK 封装会放入 MBL-PK 和 MSDK CERT，在 MSDK CERT 中包含 MSDK-PK，MSDK 私钥用来给 MSDK 摘要签名。

图 3-4. 带证书固件封装格式



固件封装是由编译后自动运行的脚本 mbl_afterbuild.bat 和 image_afterbuild.bat 完成的。

用户只要配好了%SDK%\config\config_gdm32.h，并生成好相应的密钥对和证书放在%SDK%\scripts\certs\下，最好配置好 IDE 调用 xxx_afterbuild.bat 脚本时传入的参数，就会自动生成相应的固件封装。

注意：目前支持 ECDSA 256 和 ED25519 两种算法的签名和认证。如果选择 ECDSA256 密钥对和证书，那 IDE 在调用 xxx_afterbuild.bat 时传入的参数需要设定为“ECDSA256”。例如：

cmd /C “..\..\..\image_afterbuild.bat riscv-nuclei-elf- **ECDSA256** CERT ${eclipse_home}/../Tools/OpenOCD/xpack-openocd-0.11.0-3/bin ${ProjDirPath}/../../../../ “” “

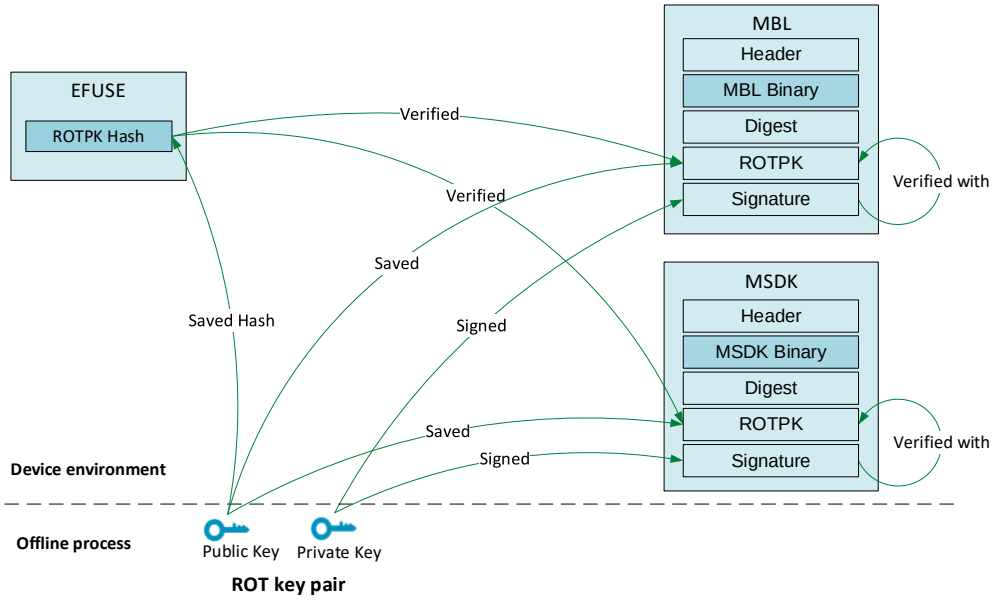
如果选择证书认证方式，参数就传“CERT”，否则就传“IMG”。

3.2.2. 启动信任链

ROT 密钥对的公钥称之为 ROTPK，MBL 密钥对的公钥称之为 MBL-PK。

如图 3-5 所示，安全启动不需要证书认证时，使用 ROT 一组密钥对来签名和验签名即可。我们使用 ROT 私钥用来给 MBL 和 MSDK 的摘要签名，使用 ROTPK 来验证签名。

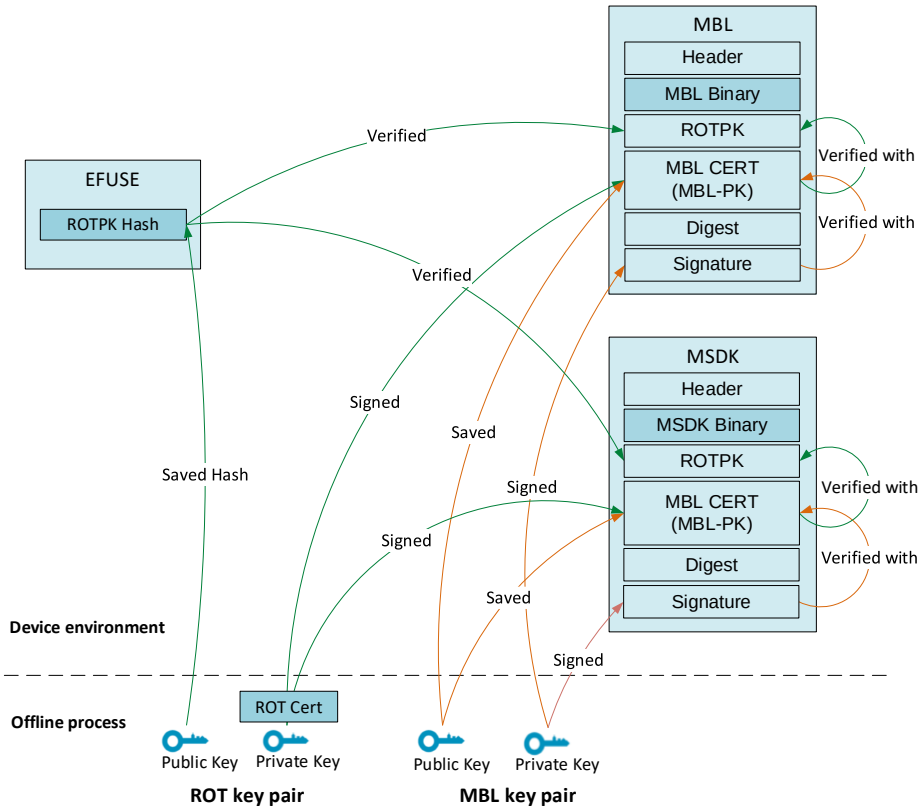
图 3-5 无证书认证信任链



安全启动如果支持证书认证，如图 3-6 所示。首先需要生成 ROT 和 MBL 两组密钥对以及两个证书。然后使用 ROT 证书给 MBL-PK 证书签名，使用 MBL 私钥给 MBL 和 MSDK 的摘要签名。生成密钥对的方法可以参考 3.2.3 节。

安全启动时，ROM 首先使用 EFUSE 中的 ROTPK Hash 来验证 MBL 携带的 ROTPK 的合法性，然后使用 ROTPK 验证 MBL-PK 证书的合法性，如果验证通过，就从里面取出 MBL-PK，再使用 MBL-PK 来验证 MBL 摘要签名的合法性。如果验证都通过，就会跳转给到 MBL 运行。跳转到 MBL 之后，MBL 会使用相同的方法来验证 MSDK 的合法性。

图 3-6 证书认证信任链



3.2.3. 密钥和证书生成

目前我们会使用到 ROT 密钥对和 MBL 密钥对，以及 ROT 证书和 MBL 证书。需要下载并安装好 OpenSSL Windows 工具。然后打开 Window CMD 窗口，依次输入下文的命令生成需要用到的密钥对和证书。

ROT 密钥对和证书

生成 ROT 密钥对：

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

图 3-7. 生成 ROT 密钥对

```
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 []:
```

过程中需要设置 PEM 密码，本文默认使用“12345678”。然后依次填写证书请求相关信息，成功后会得到 privkey.pem 和 rot-req.csr。其中 privkey.pem 就是 ROT 私钥，为了区分，我们把它更名为 rot-key.pem。由于公钥可以由私钥推算出来，所以只需要保存私钥。

> move privkey.pem rot-key.pem

接着生成 ROT 证书，由它给 MBL 证书签名：

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

图 3-8. 生成 ROT 证书

```
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 是证书请求，我们使用 rot-key.pem 自签名生成 rot-cert.pem。这步需要输入上面设置的 PEM 密码“12345678”，之后会生成 rot-cert.pem。

然后进入%SDK%\scripts\imgtool\路径下，打开 Windows CMD，运行如下命令：

> imgtool.py getpub -k ..\certs\ecdsa256\rot-key.pem –sha256 1

可以获取到需要写进 EFUSE 的 ROTPKH，见图 3-9 中的 test_rotpk_hash[].

图 3-9. 获取 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 密钥对和证书

生成 MBL 密钥对：

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

图 3-10. 生成 MBL 密钥对

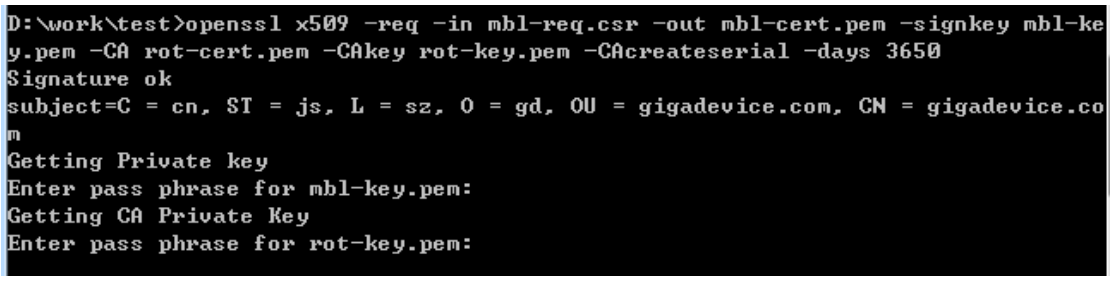
```
D:\work\test>openssl req -newkey ED25519 -new -out mbl-req.csr
Generating a ED25519 private key
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.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

运行成功会得到 privkey.pem 和 mbl-req.csr。为了区分，我们把 privkey.pem 更名为 mbl-key.pem。

> move privkey.pem mbl-key.pem

生成 MBL 证书：> 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

图 3-11 生成 MBL 证书



运行成功后生成 mbl-cert.pem。

4. 固件升级

4.1. Wifi

在发布的 SDK 里面提供了 ota_demo.c，用户在开发私有的 OTA 代码时，可以参考这个流程。

测试服务器，建议使用 Python3 HTTP Server，装好 Python3 之后就可以直接使用。命令如下所示，“HostIP”是运行这个命令的主机 IP。注意设备需要跟主机在同一个局域网内。

python -m http.server 80 –bind HostIP

示例代码假定用户已经搭建了一个 HTTP 服务器，并将新固件放在了服务器根目录下。用户可以通过设备串口输入命令来进行 OTA 测试。命令如下：

ota_demo test_ap password 192.168.3.100 image-ota.bin

其中，“test_ap”和“password”是要连接的局域网 AP 的 SSID 和密码，“192.168.3.100”是 HTTP Server 运行主机的 IP 地址，“image-ota.bin”是在 HTTP Server 根目录下放置的升级包。

主固件采用乒乓方式存储，当前运行在 MSDK 0 区域，升级固件就烧写到 MSDK 1 区域，如果当前运行在 MSDK 1，则升级固件就烧写到 MSDK 0。升级固件时，只检查云端下载到 FLASH 的固件的校验和或者 Hash，以确保固件下载的完整性和烧录到 FLASH 的正确性，并将 MSDK x 对应的 Image Status 标记为 NEW，存储到 System Status 区域。

升级完成后重启时，MBL 会先检查 MSDK 0 和 MSDK 1 的 Image Status，如果发现有 NEW 标签，会优先进行验证，即验证 MSDK 的固件签名或者证书。如果验证通过就将相应的 Image Status 标记为 VERIFY_OK，然后跳转到 NEW 固件运行。如果验证失败就把相应的 Image Status 标记为 VERIFY_FAIL 进入 While(1)，重启后会选择验证另一块 MSDK 固件，也即旧固件，验证通过就运行。

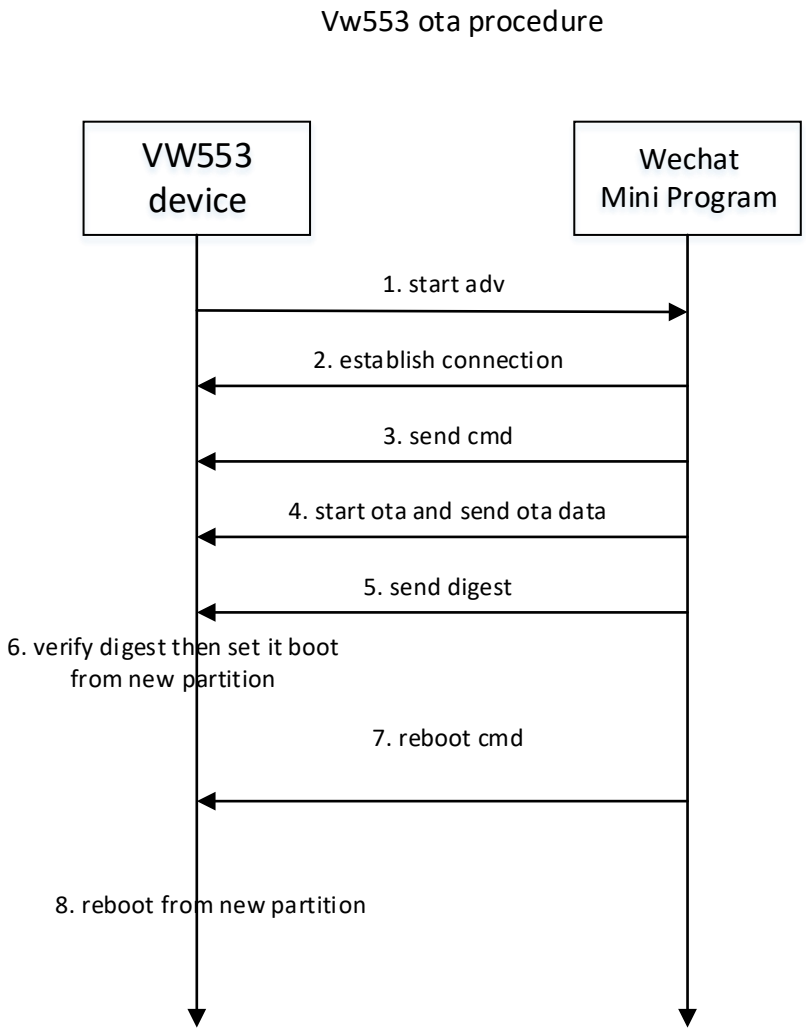
简单描述设备升级的步骤如下：

1) 设备首先确定当前运行的 MSDK 是 0 还是 1，如果当前是 0，则目标烧写位置为 1，反之为 0；
2) 设备跟 HTTP 服务器建立 TCP 连接；
3) 设备向服务器发送 HTTP Request；
4) 服务器响应 HTTP response 200 OK，将 MSDK Manifest 分片发送出来；
5) 设备收到正确响应后擦除目标烧写位置的数据，并依次将分片内容烧写进 FLASH；
6) 服务器在固件包数据发送结束后，发送固件包的校验和；
7) 设备收到校验和之后，读取目标烧写位置的 FLASH 内容计算校验和进行比较；
8) 校验通过则标注目标烧写位置的 Image Status 为 NEW，当前运行位置的 Image Status 为 Old；
9) 重启；

10) MBL 验证新固件的签名，如果验证通过，就跳转到新固件运行，否则重启后回到旧固件运行。

4.2. Ble

4.2.1. 升级流程



4.2.2. UUID 说明

Attribute	UUID	Property	Description
Service	0xFF00		
Control Characteristic	0xFF11	Write Without Response	发送控制指令
Data Characteristic	0xFF22	Write With Response	发送 image 数据

4.2.3. 信令控制

长度	Opcode	解释
2	0	dfu 模式（1byte）
5	1	Image size（4byte）
1	2	无
33	3	Sha256 value（32byte）
1	4	无
2	5	错误码（1byte）

DFU_OPCODE_MODE

- 发送指令: 0x00, 0x00
- 成功响应: 0x00, 0x00
- 出错响应: 0x00, error code(1 byte)
- 说明: 用于确认传输模式，后续可扩展

DFU_OPCODE_IMAGE_SIZE

- 发送指令: 0x01, image size (4 byte)
- 成功响应: 0x01, 0x00
- 出错响应: 0x01, error code(1 byte)
- 说明: 用于告诉被升级设备 image size

DFU_OPCODE_START_DFU

- 发送指令: 0x02
- 成功响应: 0x02, 0x00
- 出错响应: 0x02, error code(1 byte)
- 说明: 告诉被升级设备，开始升级

DFU_OPCODE_VERIFICATION

- 发送指令: 0x03, sha256 value (32 byte)
- 成功响应: 0x03, 0x00
- 出错响应: 0x03, error code(1 byte)
- 说明: 让被升级设备校验摘要，确认 image 完整性

DFU_OPCODE_REBOOT

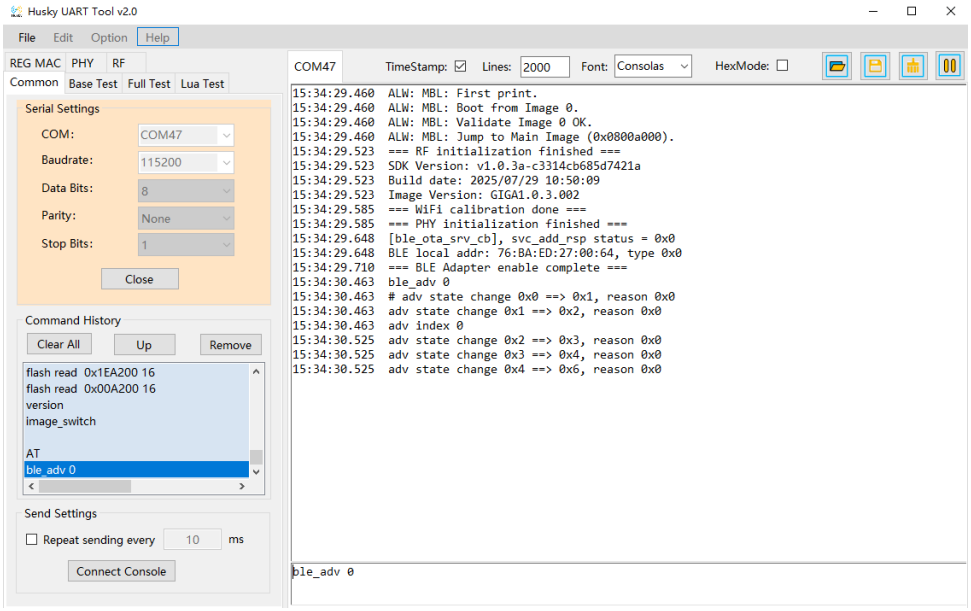
- 发送指令: 0x04,
- 成功响应: 0x04, 0x00
- 出错响应: 0x04, error code(1 byte)
- 说明: 让对端设备 reboot 并且从新的 image 启动

4.2.4. 操作步骤

Msdk 中提供了 app_dfu_srv.c 用于 ble ota。用户在开发私有的 OTA 代码时，可以参考这个流程。

步骤：

1. 首先打开宏 FEAT_SUPPORT_BLE_OTA 后进行编译。
2. 下载 image-all.bin 文件到 start board 中。
3. 上电后通过 `ble_adv 0` 打广播



4. 打开小程序，“gd 蓝牙配网”，点击搜索蓝牙，选中相应设备。

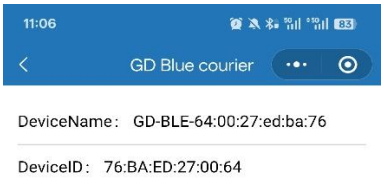

11:06
GD Blue courier


搜索蓝牙
蓝牙
版本


11:06
GD Blue courier
设备名称: GD-BLE-64:00:27:ed:ba:76
设备ID: 76:BA:ED:27:00:64
RSSI: -58

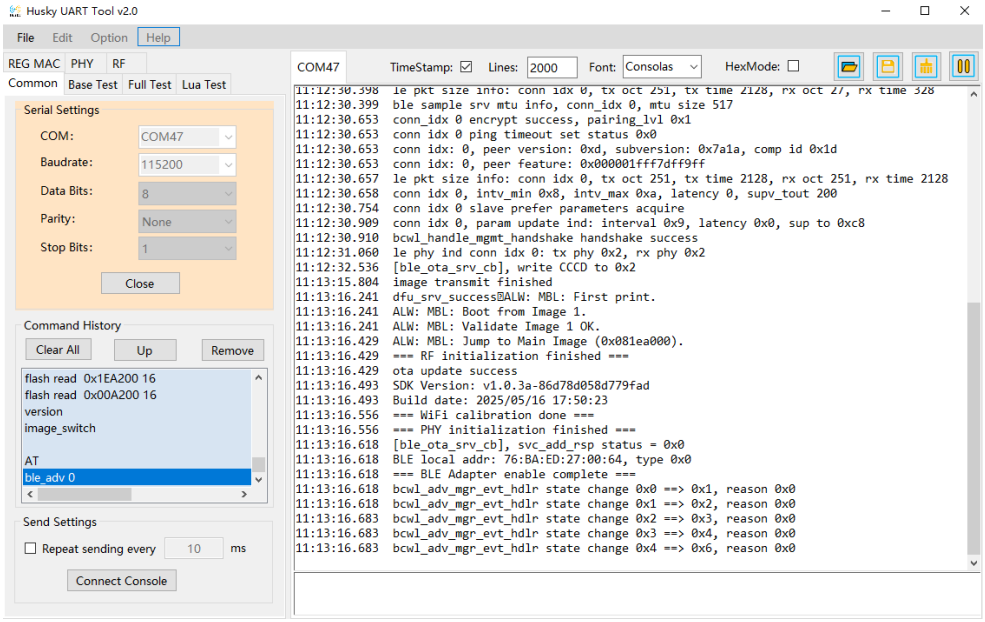

搜索中
蓝牙
版本

5. 点击进入 ota 界面



6. 加载 ota 文件后（可以加载微信聊天记录中的 bin 文件）点击开始升级

7. 升级完成设备重启并在新的 image 运行


Husky UART Tool v2.0
File Edit Option Help
REG MAC PHY RF
Common Base Test Full Test Lua Test
Serial Settings
COM: COM47
Baudrate: 115200
Data Bits: 8
Parity: None
Stop Bits: 1
Close
Command History
Clear All Up Remove
flash read 0x1EA200 16
flash read 0x00A200 16
version
image_switch
AT
ble_adv 0
Send Settings
Repeat sending every 10 ms
Connect Console
COM47 TimeStamp: Lines: 2000 Font: Consolas HexMode:
11:12:30.653 conn_idx 0 encrypt success, pairing_lvl 0x1
11:12:30.653 conn idx 0 ping timeout set status 0x0
11:12:30.653 conn idx: 0, peer version: 0xd, subversion: 0x7a1a, comp id 0x1d
11:12:30.653 conn idx: 0, peer feature: 0x000001fff7dff9ff
11:12:30.657 le pkt size info: conn idx 0, tx oct 251, tx time 2128, rx oct 251, rx time 2128
11:12:30.658 conn idx 0, intv_min 0x8, intv_max 0xa, latency 0, supv_tout 200
11:12:30.754 conn idx 0 slave prefer parameters acquire
11:12:30.909 conn idx 0, param update ind: interval 0x9, latency 0x0, sup to 0xc8
11:12:30.910 bcwl_handle_mgmt_handshake handshake success
11:12:31.060 le phy ind conn idx 0: tx phy 0x2, rx phy 0x2
11:12:32.536 [ble_ota_srv_cb], write CCCD to 0x2
11:13:15.804 image transmit finished
11:13:16.241 dfu_srv_success@ALW: MBL: First print.
11:13:16.241 ALW: MBL: Boot from Image 1.
11:13:16.241 ALW: MBL: Validate Image 1 OK.
11:13:16.429 ALW: MBL: Jump to Main Image (0x081ea000).
11:13:16.429 === RF initialization finished ===
11:13:16.429 ota update success
11:13:16.493 SDK Version: v1.0.3a-86d78d058d779fad
11:13:16.493 Build date: 2025/05/16 17:50:23
11:13:16.556 === WiFi calibration done ===
11:13:16.556 === PHY initialization finished ===
11:13:16.618 [ble_ota_srv_cb], svc_add_rsp status = 0x0
11:13:16.618 BLE local addr: 76:BA:ED:27:00:64, type 0x0
11:13:16.618 === BLE Adapter enable complete ===
11:13:16.618 bcwl_adv_mgr_evt_hdlr state change 0x0 ==> 0x1, reason 0x0
11:13:16.618 bcwl_adv_mgr_evt_hdlr state change 0x1 ==> 0x2, reason 0x0
11:13:16.683 bcwl_adv_mgr_evt_hdlr state change 0x2 ==> 0x3, reason 0x0
11:13:16.683 bcwl_adv_mgr_evt_hdlr state change 0x3 ==> 0x4, reason 0x0
11:13:16.683 bcwl_adv_mgr_evt_hdlr state change 0x4 ==> 0x6, reason 0x0

5. 版本历史

表 5-1. 版本历史

版本号	说明	日期
1.0	初稿发布	2024 年 7 月 10 日
1.1	修改了部分描述	2025 年 2 月 26 日
1.2	修改 ROTPK 相关部分	2025 年 3 月 6 日
1.3	添加对 OTA 升级的细节描述	2025 年 3 月 20 日

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.