

凌思微电子（厦门）有限公司

LINKEDSEMI



LE5010 OTA 应用说明

Revision 1.1
2021/7/28

修订记录

版本	修订日期	修订说明	作者
V1.0	2021-3-30	初始版本	jxia
V1.1	2021-7-28	增加签名认证的加密 OTA 介绍	wyu

目 录

第 1 章	OTA 镜像处理流程	4
1.1	Flash 的分区使用情况:	4
1.2	Bootloader 启动流程 (OTA 相关)	5
1.3	OTA 和 Flash 操作相关 API:	8
1.3.1	OTA 结束后将镜像信息写入 OTA SETTINGS 区域	8
1.3.2	Flash 接口	8
1.3.3	软件复位.....	9
第 2 章	OTA 服务的添加和 APP 升级流程测试	10
2.1	添加 OTA 服务	10
2.1.1	添加 OTA 相关头文件	10
2.1.2	添加 OTA profile.....	10
2.1.3	添加 ECC 校验函数和 OTA 服务回调函数	10
2.1.4	修改 DK_MAX_PROFILE_NUM 宏.....	11
2.2	OTA APP 升级测试.....	12
2.2.1	烧录好添加了 OTA 服务的 dis 固件;	12
2.2.2	生成 uart server bin 文件	12
2.2.3	将生成的 ble_uart_server .bin 文件传到手机;	13
2.2.4	使用 OTA APP 连接对应的设备	13
2.2.5	开始更新.....	13
2.2.6	OTA 完毕	14
2.3	安卓 OTA APP 源码.....	14
第 3 章	OTA 之固件签名升级流程	15
3.1	固件签名简介	15
3.2	操作步骤 (前台 OTA)	15
3.2.1	密钥生成.....	15
3.2.2	带签名验证功能的固件生成.....	16
3.2.3	签名文件生成.....	19
3.3	操作步骤 (后台 OTA)	20
3.3.1	添加相应代码.....	20
3.3.2	密钥生成.....	22
3.3.3	带签名验证功能的固件生成.....	22
3.3.4	签名文件生成.....	23
3.4	OTA APP 升级测试.....	24
3.4.1	准备升级文件.....	24
3.4.2	升级操作步骤.....	24

第1章 OTA 镜像处理流程

1.1 Flash 的分区使用情况：

	NO.	Section Name	Start Offset	End Offset	Size	Description
	8	OTA SETTINGS	0x7f000	0x80000	4KB	存放OTA 相关的一些标志位和镜像信息；
	7	BLE PROTOCOL STACK	__stack_lma__ (0x4bc00)	0x7f000	About 205KB	BLE 协议栈区域
在不使用 mesh 功能时，用户应用程序可用 Flash 空间大概 283KB	6	BLE MESH STACK (Optional)	__mesh_stack_lma__	__stack_lma__	About 120KB	BLE mesh 协议栈区域（可选）；
	5	SINGLE BANK FOTA UTILITY (optional)	0x3d000		About 40KB	前台单分区 OTA 时，FOTA 镜像存放的区域；（可选）
	4	IMAGE and OTA IMAGE	0x5000			用户 APP 固件区域
	3	PERSISTENT DATA	0x2000	0x5000	SECTION_NUM * SECTION_SIZE (Default 12KB)	tinyfs 文件系统数据区，存放用户掉电不丢失数据；
	2	SECOND BOOTLOADER	0x300	0x2000	7.25 KB	bootloader 区域
	1	INFORMATION	0x0	0x300	768 Bytes	存放MAC地址等一些启动相关的信息；
注：BLE协议栈的起始地址 __stack_lma__ 是根据协议栈的大小确定的；						

参考在线文档：https://ls-ble-sdk.readthedocs.io/zh/latest/getting_started/memory.html

和 OTA 相关的有两个区域：OTA SETTINGS 和 INFORMATION。下面具体分析：

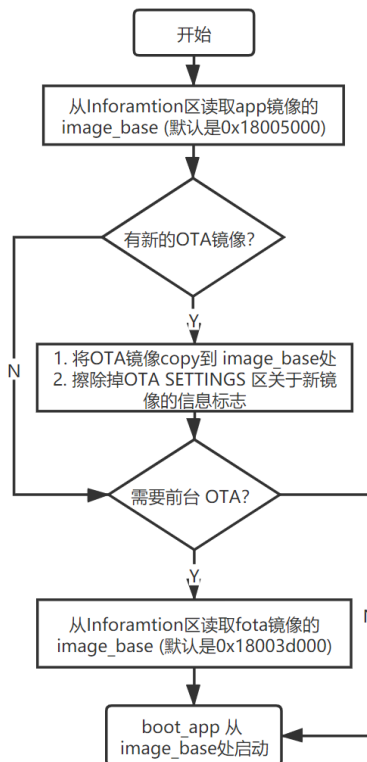
1.2 Bootloader 启动流程 (OTA 相关)

代码路径: ble_sdk_app\dev\bootloader\boot_ram\boot_ram_le501x.c

```

void boot_ram_start(uint32_t exec_addr)
{
    disable_global_irq();
    switch_to_rc32k();
    clk_switch();
    uint8_t wkup_stat = REG_FIELD_RD(SYS_CFG->PMU_WKUP, SYS_CFG_WKUP_STAT);
    set_wakeup_source(wkup_stat);
    REG_FIELD_WR(SYS_CFG->PMU_WKUP, SYS_CFG_LP_WKUP_CLR, 1);
    DELAY_US(200);
    SYS_CFG->PMU_PWR = 0;
    systick_start();
    spi_flash_drv_var_init(false, false);
    spi_flash_init();
    spi_flash_qe_status_read_and_set();
    spi_flash_xip_start();
    lscache_cache_enable(0);
    io_init();
    swd_pull_down();
    trim_val_load();
    uint32_t image_base;
    image_base = get_app_image_base(); ← 1
    struct fota_image_info image;
    if(ota_copy_info_get(&image)) ← 2
    {
        fw_copy(&image, image_base);
        ota_settings_erase();
    }
    if(need_foreground_ota()) ← 3
    {
        image_base = get_fota_image_base();
    }
    boot_app(image_base); ← 4
}
    
```

Bootloader 的一项功能就是引导应用程序的启动，这时就需要知道 image 的起始地址 image_base；这个 image_base 地址一般是在 Flash 的 INFORMATION 区域。这里还需要考虑前后台单双分区 OTA 的情况。

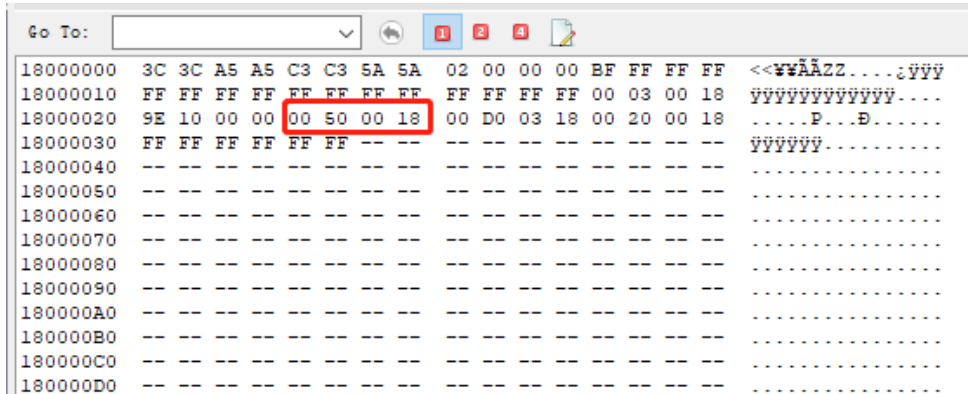


图表 1: OTA 镜像处理流程

关于前后台单双分区 OTA 的概念可以参考线上文档：

https://ls-ble-sdk.readthedocs.io/zh/latest/getting_started/fota.html

步骤一：从 INFORMATION 区域读取出 image base 确定镜像起始地址；默认都是 0x18005000 (这个值是在 ls_ble_sdk\tools\le501x\after_build.bat 中写入的)：



```

Go To: [ ]
18000000  3C 3C A5 A5 C3 C3 5A 5A 02 00 00 00 BF FF FF FF <<vwAAZZ....:yyy
18000010  FF FF FF FF FF FF FF FF FF FF FF FF 00 03 00 18 yyyyyyyyyyyy....
18000020  9E 10 00 00 00 50 00 18 00 D0 03 18 00 20 00 18 .....P...D.....
18000030  FF FF FF FF FF FF -- -- -- -- -- -- -- -- yyyyyyy.....
18000040  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18000050  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18000060  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18000070  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18000080  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18000090  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
180000A0  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
180000B0  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
180000C0  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
180000D0  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    
```

步骤二：从 OTA SETTINGS 区读取新 image 的信息，包括起始地址和镜像大小；如果读回的数据不是全 F，表示有 OTA 的镜像已经更新，需要

- 1) 将新的镜像搬移到步骤一读取到的 image base 地址（一般是 0x18005000）；
- 2) 清除前台 OTA 升级标志位（使用的是系统的寄存器 SYSCFG->BKD[7]）；

步骤三：判断是否进行前台 OTA；

前台 OTA 需要判断，前台 OTA 升级标志和 OTA 的类型是前台单区 OTA；满足条件就从 Flash 0x28 的偏移地址读取 image base（默认是 0x0x1803d000，这个值是在 ls_ble_sdk\tools\le501x\after_build.bat 中写入的）

```

1  %2\tools\srec_cat.exe -o .\UVBuild\info_sbl.hex -I
   %2\dev\soc\arm_cm\le501x\bin\bram.bin -Bin -of
   0x18000300 -crc32-l-e -max-a %2
   \dev\soc\arm_cm\le501x\bin\bram.bin -Bin -of
   0x18000300 %2\tools\le501x\info.bin -Bin -of
   0x18000000 -gen 0x1800001c 0x18000020 -const-l-e
   0x18000300 4 -gen 0x18000020 0x18000024 -const-l-e
   -l %2\dev\soc\arm_cm\le501x\bin\bram.bin -Bin 4
   -gen 0x18000024 0x18000028 -const-l-e 0x18005000 4
   -gen 0x18000028 0x1800002c -const-l-e 0x1803d000 4
   -gen 0x1800002c 0x18000030 -const-l-e 0x18002000 4
   -gen 0x18000030 0x18000036 -rep-d 0xff 0xff 0xff
   0xff 0xff 0xff
2  fromelf --i32 --output=.\UVBuild\%1.hex .\UVBuild\
   %1.axf
3  %2\tools\srec_cat.exe -Output .\UVBuild\%1
   _production.hex -Intel .\UVBuild\info_sbl.hex
   -Intel .\UVBuild\%1.hex -Intel %2\%3 -Intel
4  fromelf -c -a -d -e -v -o .\UVBuild\%1.asm
   .\UVBuild\%1.axf
    
```

步骤四: boot_app 从 image_base 处启动;

- 设置主栈指针;
- 进入 rerest_handler;

1.3 OTA 和 Flash 操作相关 API:

1.3.1 OTA 结束后将镜像信息写入 OTA SETTINGS 区域

OTA 结束以后需要将 OTA 镜像的地址和镜像的大小写到 Flash 的 OTA SETTINGS 区域；调用的接口：

```
1. void ota_copy_info_set(struct fota_image_info *ptr)
```

参数说明：

```
1. struct fota_image_info
2. {
3.     uint32_t base;    // OTA 镜像起始地址
4.     uint32_t size;    // OTA 镜像大小
5. };
```

1.3.2 Flash 接口

1.3.2.1 Read

```
1. void spi_flash_quad_io_read(uint32_t offset, uint8_t * data, uint
   16_t length)
```

参数说明：

```
1. uint32_t offset : 偏移地址；
2. uint8_t * data : 读取数据的 buffer 指针；
3. uint16_t length : 读取的数据长度；
```

1.3.2.2 Program

```
1. void spi_flash_quad_page_program(uint32_t offset, uint8_t *data, un
   t16_t length)
```

参数说明：

```
1. uint32_t offset : 偏移地址；
2. uint8_t * data : 要写入数据的 buffer 指针；
3. uint16_t length : 写入数据长度；
```


1.3.2.3 Sector Erase (4KB 擦除)

```
1. void spi_flash_sector_erase(uint32_t offset)
```

参数说明:

```
1. uint32_t offset : 偏移地址;
```

1.3.2.4 Page Erase (256 Bytes 擦除)

```
1. void spi_flash_page_erase(uint32_t offset)
```

参数说明:

```
1. uint32_t offset : 偏移地址;
```

1.3.3 软件复位

```
1. void platform_reset(uint32_t error)
```

第2章 OTA 服务的添加和 APP 升级流程测试

这里以在 SDK 中的 ble_dis 例程中添加 OTA 服务为例，演示如何在项目中添加 OTA 服务，以及如何利用安卓 OTA APP 进行固件升级；

2.1 添加 OTA 服务

2.1.1 添加 OTA 相关头文件

```
1. #include "prf_fotas.h"
2. #include "tinycrypt/ecc_dsa.h"
```

2.1.2 添加 OTA profile

```
static void prf_added_handler(struct profile_added_evt *evt)
{
    LOG_I("profile:%d, start handle:0x%x\n", evt->id, evt->start_hdl);
    switch(evt->id)
    {
        case PRF_DIS_SERVER:
            prf_dis_server_callback_init(prf_dis_server_callback);
            dev_manager_prf_fota_server_add(NO_SEC);
            //create_adv_obj();
            break;
        case PRF_FOTA_SERVER:
            prf_fota_server_callback_init(prf_fota_server_callback);
            create_adv_obj();
            break;
        default:
            break;
    }
}
```

2.1.3 添加 ECC 校验函数和 OTA 服务回调函数

具体代码参考 SDK 中的 fota 示例代码：

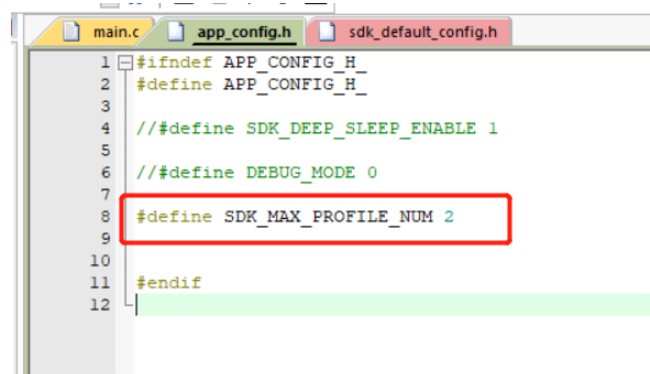
```
#if FW_ECC_VERIFY
extern const uint8_t fotas_pub_key[64];
bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
{
    return uECC_verify(fotas_pub_key, digest->data, sizeof(digest->data), signature->data, uECC_secp256r1());
}
#else
bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
{
    return true;
}
#endif
```

```

static void prf_fota_server_callback(enum fotas_evt_type type, union fotas_evt_u *evt, uint8_t con_idx)
{
    switch(type)
    {
        case FOTAS_START_REQ_EVT:
        {
            // ota_settings_write(SINGLE_FOREGROUND);
            ota_settings_write(DOUBLE_BACKGROUND); |
            enum fota_start_cfm_status status;
            if(fw_signature_check(evt->fotas_start_req.digest, evt->fotas_start_req.signature))
            {
                status = FOTA_REQ_ACCEPTED;
            }else
            {
                status = FOTA_REQ_REJECTED;
            }
            LOG_I("OTA Start \n");
            prf_fotas_start_confirm(con_idx, status);
        }break;
        case FOTAS_FINISH_EVT:
            if(evt->fotas_finish.integrity_checking_result)
            {
                if(evt->fotas_finish.new_image->base != get_app_image_base())
                {
                    ota_copy_info_set(evt->fotas_finish.new_image);
                }
                else
                {
                    ota_settings_erase();
                }
                LOG_I("OTA SUCCEED!");
                platform_reset(RESET_OTA_SUCCEED);
            }else
            {
                platform_reset(RESET_OTA_FAILED);
            }
        }break;
        default:
            LS_ASSERT(0);
        }break;
    }
}
    
```

2.1.4 修改 DK_MAX_PROFILE_NUM 宏

这里仅以 dis server 例程为例，项目中以具体使用的 profile 数量为准；



```

main.c  app_config.h  sdk_default_config.h
1  #ifndef APP_CONFIG_H
2  #define APP_CONFIG_H
3
4  // #define SDK_DEEP_SLEEP_ENABLE 1
5
6  // #define DEBUG_MODE 0
7
8  #define SDK_MAX_PROFILE_NUM 2
9
10
11 #endif
12
    
```

2.2 OTA APP 升级测试

测试方法:

烧录 dis server 的例程，通过安卓 OTA APP 将 uart server 固件更新进去；

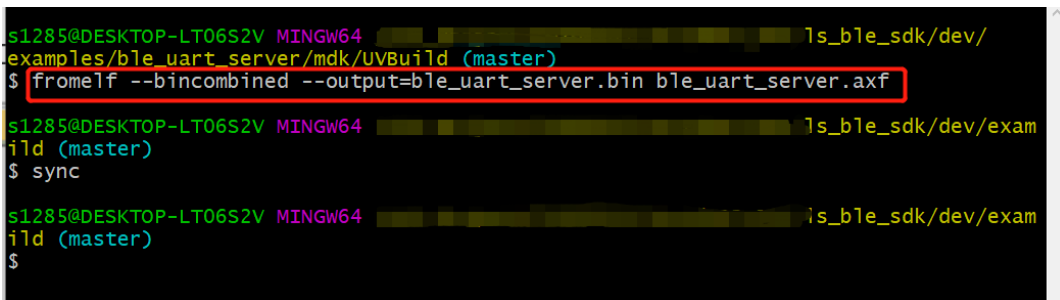
2.2.1 烧录好添加了 OTA 服务的 dis 固件；

烧录好添加完添加了 OTA 服务的 dis server 固件。

2.2.2 生成 uart server bin 文件

以 KEIL 环境为例，编译好 uart server 例程；用下面的命令将 `ls_ble_sdk\dev\examples\ble_uart_server\mdk\UVBuild` 目录下的 `ble_uart_server.axf` 转成 `ble_uart_server.bin` 文件：

```
1. fromelf --bincombined --
   output=ble_uart_server.bin ble_uart_server.axf
```

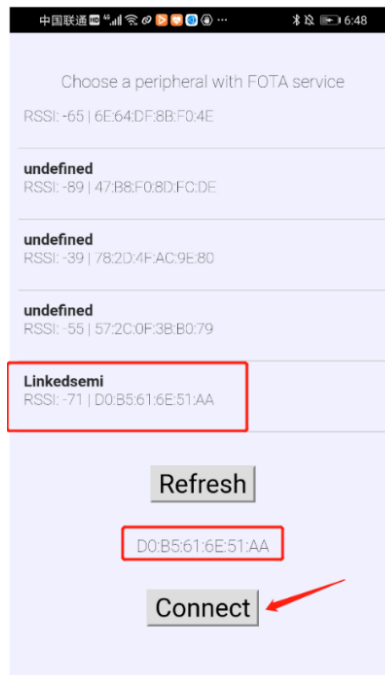


```
s1285@DESKTOP-LT06S2V MINGW64 ls_ble_sdk/dev/
examples/ble_uart_server/mdk/UVBuild (master)
$ fromelf --bincombined --output=ble_uart_server.bin ble_uart_server.axf
s1285@DESKTOP-LT06S2V MINGW64 ls_ble_sdk/dev/exam
ild (master)
$ sync
s1285@DESKTOP-LT06S2V MINGW64 ls_ble_sdk/dev/exam
ild (master)
$
```

名称	修改日期	类型	大小
ble_uart_server.asm	2021/2/21 18:38	ASM Source File	1,045 KB
ble_uart_server.axf	2021/2/21 18:38	AXF 文件	2,979 KB
ble_uart_server.bin	2021/2/21 18:40	BIN 文件	32 KB
ble_uart_server.build_log.htm	2021/2/21 18:38	Chrome HTML D...	6 KB
ble_uart_server.htm	2021/2/21 18:38	Chrome HTML D...	250 KB
adpcm.crf	2021/2/21 18:38	CRF 文件	9 KB
calc_acc.crf	2021/2/21 18:38	CRF 文件	12 KB
calc_div.crf	2021/2/21 18:38	CRF 文件	6 KB
common.crf	2021/2/21 18:38	CRF 文件	11 KB

2.2.3 将生成的 ble_uart_server .bin 文件传到手机;

2.2.4 使用 OTA APP 连接对应的设备



2.2.5 开始更新

选择要更新固件的 bin 文件，点击 Start FOTA 进行升级：（默认双分区升级镜像放在 0x18020000 地址，可以根据项目情况修改）



2.2.6 OTA 完毕

OTA 结束后,APP 底部会出现 ‘OTA complete,status:0’ 的提示;忽略结束时 APP 提示的错误;



2.3 安卓 OTA APP 源码

源码仓库 : https://github.com/linkedsemi/BLE_FOTA_APP

第3章 OTA 之固件签名升级流程

3.1 固件签名简介

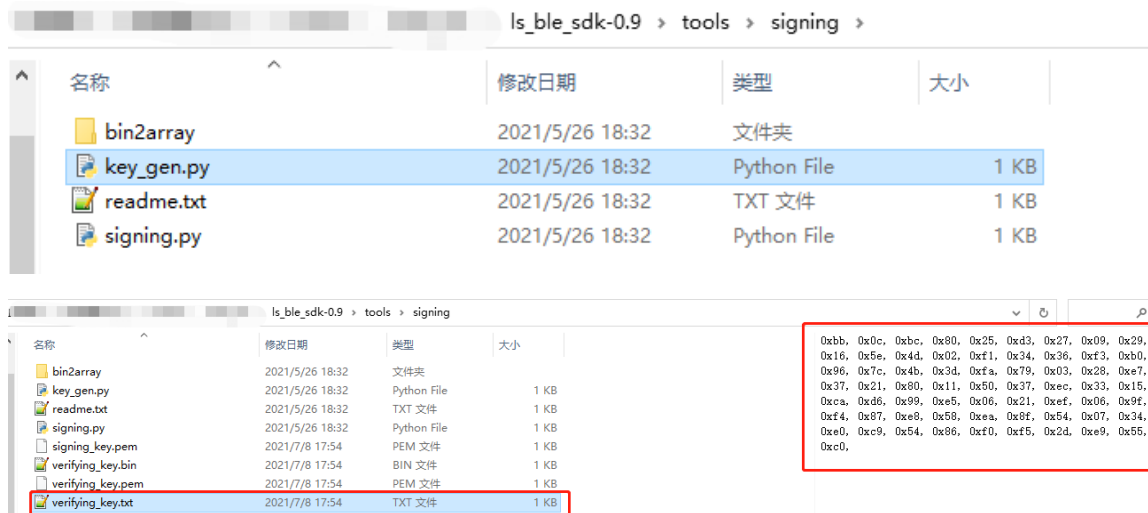
对于于升级的固件有安全性要求的场景下，可以使用固件签名。启用固件签名功能后，SBL 会检查签名是否合法，如果不合法，则拒绝升级，以此保障新固件的来源可靠，固件签名采用 ECDSA 算法。

1. 利用“tools/signing/key_gen.py”生成一对密钥，包含公钥、私钥。
2. FOTA (dev\examples\ble\fota) 中，定义宏“FW_ECC_VERIFY”为 1，并将公钥拷贝到“sbl/pub_key.c”文件中，编译生成带有验证签名功能的 FOTA。
3. 新固件生成后，利用 tools/signing/signing.py 生成固件签名文件。手机 OTA 升级时，除了选择固件文件之外，还要选择此签名文件。

3.2 操作步骤（前台 OTA）

3.2.1 密钥生成

打开“tools/signing/key_gen.py”中脚本,运行后会生成一对密钥:



3.2.2 带签名验证功能的固件生成

打开“dev\examples\ble\fota”文件夹,并将宏“FW_ECC_VERIFY”设置为1,并将公钥“verifying_key”拷贝到“pub_key.c”文件中,再编译出相应的“fota.hex”固件。

ls_ble_sdk-0.9 > dev > examples > ble > fota

名称	修改日期	类型	大小
compiler	2021/5/26 18:32	文件夹	
mdk	2021/5/26 18:32	文件夹	
app_config.h	2021/5/26 18:32	H 文件	1 KB
main.c	2021/5/26 18:32	C 文件	5 KB
pub_key.c	2021/5/26 18:32	C 文件	1 KB
SConscript	2021/5/26 18:32	文件	1 KB

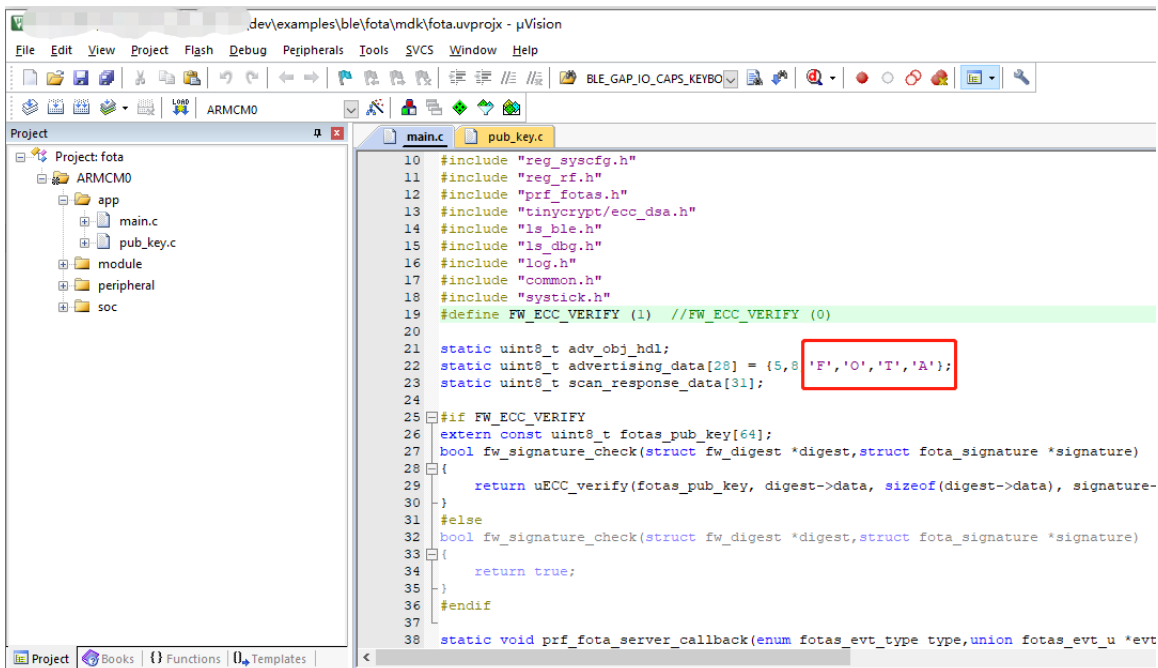
```

13 #include "tinycrypt/ecc_dsa.h"
14 #include "ls_ble.h"
15 #include "ls_dbg.h"
16 #include "log.h"
17 #include "common.h"
18 #include "systick.h"
19 #define FW_ECC_VERIFY (1) //FW_ECC_VERIFY (0)
20
21 static uint8_t adv_obj_hdl;
22 static uint8_t advertising_data[28] = {5,8,'F','O','T','A'};
23 static uint8_t scan_response_data[31];
24
25 #if FW_ECC_VERIFY
26 extern const uint8_t fotas_pub_key[64];
27 bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
28 {
29     return uECC_verify(fotas_pub_key, digest->data, sizeof(digest->data), signature->data, uECC_secp256r1());
30 }
    
```

```

1 #include <stdint.h>
2
3 //const uint8_t fotas_pub_key[64] = {0};
4 const uint8_t fotas_pub_key[64] = {
5     0xbb, 0x0c, 0xbc, 0x80,
6     0x25, 0xd3, 0x27, 0x09,
7     0x29, 0x16, 0x5e, 0x4d,
8     0x02, 0xf1, 0x34, 0x36,
9     0xf3, 0xb0, 0x96, 0x7c,
10    0x4b, 0x3d, 0xfa, 0x79,
11    0x03, 0x28, 0xe7, 0x37,
12    0x21, 0x80, 0x11, 0x50,
13    0x37, 0xec, 0x33, 0x15,
14    0xca, 0xd6, 0x99, 0xe5,
15    0x06, 0x21, 0xef, 0x06,
16    0x9f, 0xf4, 0x87, 0xe8,
17    0x58, 0xea, 0x8f, 0x54,
18    0x07, 0x34, 0xe0, 0xc9,
19    0x54, 0x86, 0xf0, 0xf5,
20    0x2d, 0xe9, 0x55, 0xc0, };
    
```

前台 OTA 的模式需要注意,我们进入 OTA 模式是需要调用这个函数:“request_ota_reboot()”,这时候我们会进行复位进入 OTA 的应用里,而且广播名称也会变成 fota 项目工程中定义的名称,如下图部分代码所示:



```

10 #include "reg_syscfg.h"
11 #include "reg_rf.h"
12 #include "prf_fotas.h"
13 #include "tinycrypt/ecc_dsa.h"
14 #include "ls_ble.h"
15 #include "ls_dbg.h"
16 #include "log.h"
17 #include "common.h"
18 #include "systick.h"
19 #define FW_ECC_VERIFY (1) //FW_ECC_VERIFY (0)
20
21 static uint8_t adv_obj_hdl;
22 static uint8_t advertising_data[28] = {5, 8, 'F', 'O', 'T', 'A'};
23 static uint8_t scan_response_data[31];
24
25 #if FW_ECC_VERIFY
26 extern const uint8_t fotas_pub_key[64];
27 bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
28 {
29     return uECC_verify(fotas_pub_key, digest->data, sizeof(digest->data), signature->
30 }
31 #else
32 bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
33 {
34     return true;
35 }
36 #endif
37 static void prf_fota_server_callback(enum fotas_evt_type type, union fotas_evt_u *evt
    
```

将上面的步骤全部走完就可以编译 **fota** 这个项目工程，如下图所示生成了一个“**fota.hex**”文件，这个文件将在下面会使用到，也就是在调用“**request_ota_reboot()**”这个函数后就会跑 **fota** 这个应用中，在这个应用中便可以使用 APP 进行升级。

ls_ble_sdk-0.9 > dev > examples > ble > fota > mdk > UVBuild

名称	修改日期	类型	大小
fota_ARMCM0.dep	2021/7/8 18:24	DEP 文件	127 KB
fota.hex	2021/7/8 18:24	HEX 文件	106 KB

当然上面的步骤还只是调用“**request_ota_reboot()**”这个函数后进行的操作，现在将介绍在哪调用这个函数，这个步骤和和前台 OTA 操作不一样，因为前台 OTA 是直接将 OTA 这个功能移植到了我们的应用中去了，而后台 OTA 的操作就需要调用一个函数后才能进这个 OTA 的应用中。

详细步骤就是：在调用“**request_ota_reboot()**”这个函数后，我们会在做一个标记，做完这个标记后设备会进行复位重启，这时候设备在重启的时候检测这个标记是否被标记上，如果有被标记就会跑 OTA 的应用，否则就会走原来的应用，在 OTA 操作成功后会将这个标记清除，此时我们的应用部分也被升级掉了，这时去跑原来的应用也就是跑升级后的固件了。

下图就是在我们的应用部分调用“`request_ota_reboot()`”这个函数然后再去下载的文件，这个例程是在 AT 指令中添加了一个 OTA 的指令，在发送“`AT+OTA`”这条指令后便会触发这个函数：

```
case AT_CMD_IDX_OTA:
{
    fota_flag = false;
    msg_len = sprintf((char *)msg_rsp, "\r\n+OTA\r\nOK\r\n");
    uart_write(msg_rsp, msg_len);
    request_ota_reboot();
}
```

需要注意一共需要下载 4 个文件：

- ①协议栈“`fw.hex`”；
- ②second bootloader “`info_sbl.hex`”；
- ③应用部分“`xxxx.hex`”；
- ④OTA 部分“`fota.hex`”

或者下载 2 个文件：

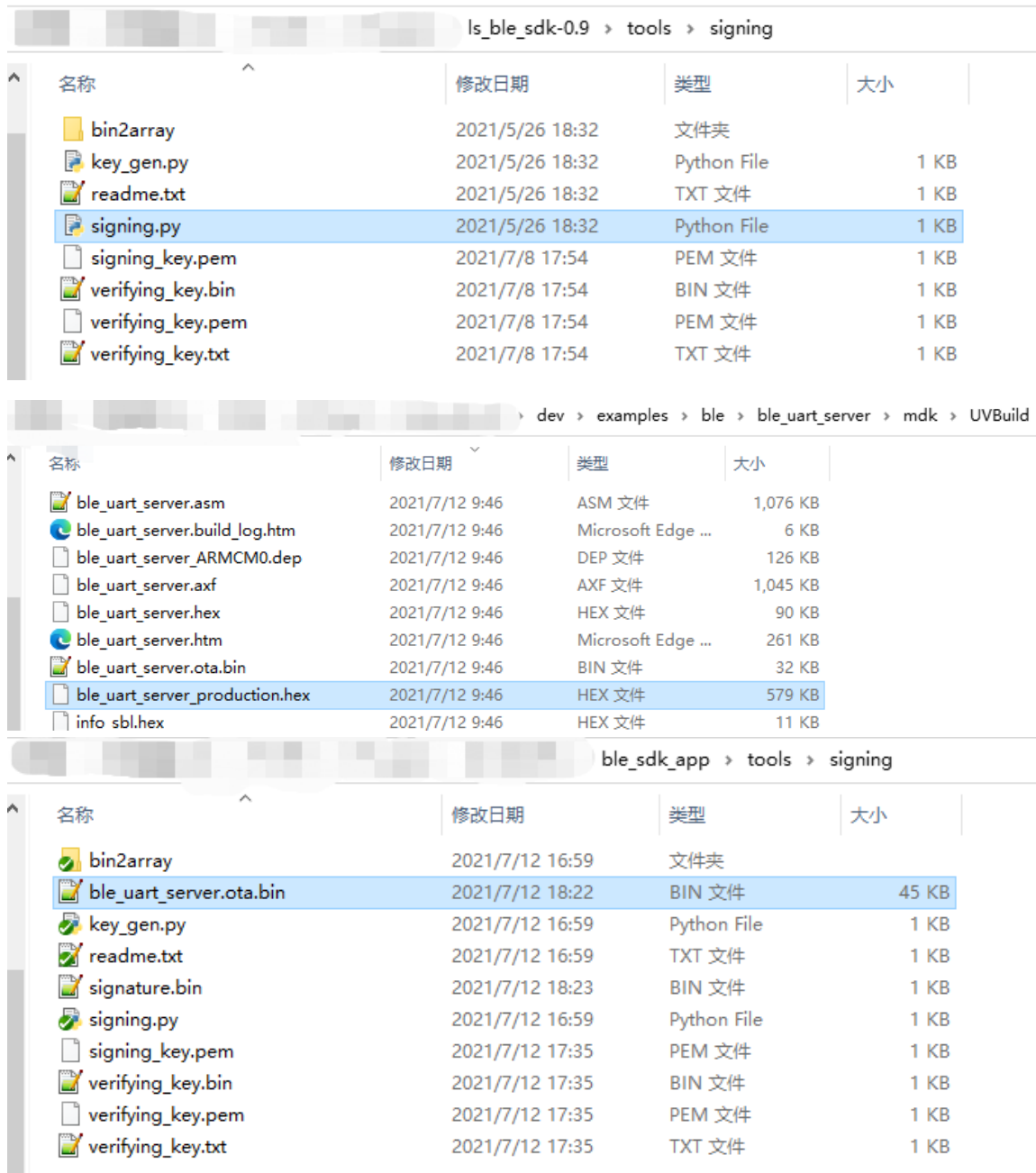
- ①协议栈、second bootloader、应用部分的合并文件：“`XXX_production.hex`”
- ②OTA 部分“`fota.hex`”

3.2.3 签名文件生成

新固件生成后，利用“tools/signing/signing.py”生成固件签名文件。手机 OTA 升级时，除了选择固件文件之外，还要选择此签名文件，注意需要使用以下命令进行生成“signing_key.pem”

```
“python signing.py [ota_firmware].bin signing_key.pem”
```

注：此时我使用的是 ble_uart_server 作为测试例程，所以在生成签名文件的时候 “[ota_firmware].bin”使用的就是 “ble_uart_server.ota.bin”，（注意：此时的 ble_uart_server 这个工程是模拟修改了一些 bug 后的新固件，所以后面是将这个新固件去升级原始的老固件。）



ls_ble_sdk-0.9 > tools > signing

名称	修改日期	类型	大小
bin2array	2021/5/26 18:32	文件夹	
key_gen.py	2021/5/26 18:32	Python File	1 KB
readme.txt	2021/5/26 18:32	TXT 文件	1 KB
signing.py	2021/5/26 18:32	Python File	1 KB
signing_key.pem	2021/7/8 17:54	PEM 文件	1 KB
verifying_key.bin	2021/7/8 17:54	BIN 文件	1 KB
verifying_key.pem	2021/7/8 17:54	PEM 文件	1 KB
verifying_key.txt	2021/7/8 17:54	TXT 文件	1 KB

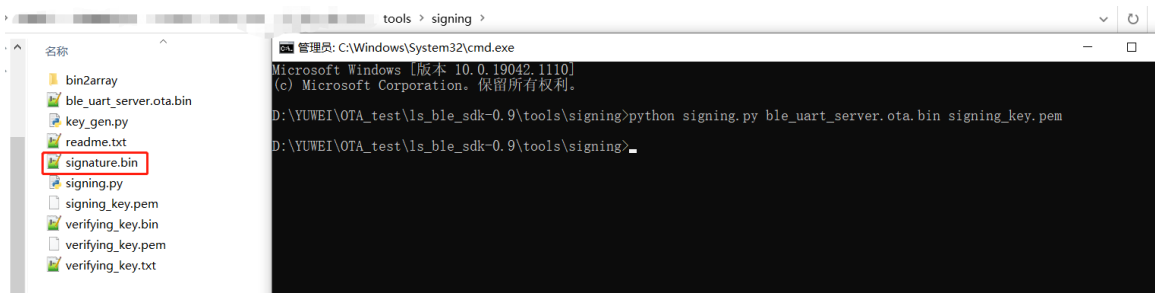
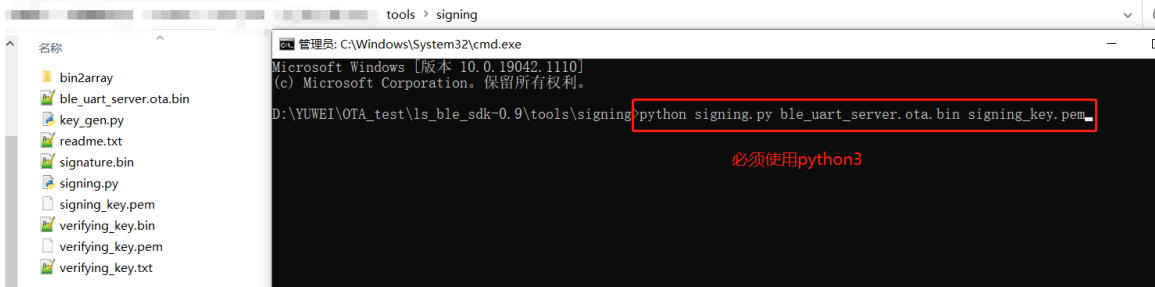
dev > examples > ble > ble_uart_server > mdk > UVBuild

名称	修改日期	类型	大小
ble_uart_server.asm	2021/7/12 9:46	ASM 文件	1,076 KB
ble_uart_server.build_log.htm	2021/7/12 9:46	Microsoft Edge ...	6 KB
ble_uart_server_ARMCM0.dep	2021/7/12 9:46	DEP 文件	126 KB
ble_uart_server.axf	2021/7/12 9:46	AXF 文件	1,045 KB
ble_uart_server.hex	2021/7/12 9:46	HEX 文件	90 KB
ble_uart_server.htm	2021/7/12 9:46	Microsoft Edge ...	261 KB
ble_uart_server.ota.bin	2021/7/12 9:46	BIN 文件	32 KB
ble_uart_server_production.hex	2021/7/12 9:46	HEX 文件	579 KB
info_sbl.hex	2021/7/12 9:46	HEX 文件	11 KB

ble_sdk_app > tools > signing

名称	修改日期	类型	大小
bin2array	2021/7/12 16:59	文件夹	
ble_uart_server.ota.bin	2021/7/12 18:22	BIN 文件	45 KB
key_gen.py	2021/7/12 16:59	Python File	1 KB
readme.txt	2021/7/12 16:59	TXT 文件	1 KB
signature.bin	2021/7/12 18:23	BIN 文件	1 KB
signing.py	2021/7/12 16:59	Python File	1 KB
signing_key.pem	2021/7/12 17:35	PEM 文件	1 KB
verifying_key.bin	2021/7/12 17:35	BIN 文件	1 KB
verifying_key.pem	2021/7/12 17:35	PEM 文件	1 KB
verifying_key.txt	2021/7/12 17:35	TXT 文件	1 KB

名称	日期/时间	文件类型	大小
bin2array	2021/5/26 18:32	文件夹	
ble_uart_server.ota.bin	2021/7/12 9:46	BIN 文件	32 KB
key_gen.py	2021/5/26 18:32	Python File	1 KB
readme.txt	2021/5/26 18:32	TXT 文件	1 KB
signature.bin	2021/7/12 9:49	BIN 文件	1 KB
signing.py	2021/5/26 18:32	Python File	1 KB
signing_key.pem	2021/7/8 17:54	PEM 文件	1 KB
verifying_key.bin	2021/7/8 17:54	BIN 文件	1 KB
verifying_key.pem	2021/7/8 17:54	PEM 文件	1 KB
verifying_key.txt	2021/7/8 17:54	TXT 文件	1 KB



注意:这条指令必须使用 **python3** 打开,否则会出问题,如果没有生成这个“signature.bin”文件,有可能是因为安装了 Python2 导致不能识别这条指令。并且注意这个签名文件只是针对这个“ble_uart_server.ota.bin”进行了签名,在 OTA 时需要将这两个文件添加进去才能进行升级,并且这个签名文件错误也会升级不成功。

3.3 操作步骤（后台 OTA）

3.3.1 添加相应代码

这部分在需要添加后台 OTA 的应用代码中进行加入。

```
#define FW_ECC_VERIFY (1)
#if FW_ECC_VERIFY
extern const uint8_t fotas_pub_key[64];
bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
{
    return uECC_verify(fotas_pub_key, digest->data, sizeof(digest->data), signature->data, uECC_secp256r1());
}
```

```

#else
bool fw_signature_check(struct fw_digest *digest, struct fota_signature *signature)
{
    return true;
}
#endif

static void prf_fota_server_callback(enum fotas_evt_type type, union fotas_evt_u *evt, uint8_t con_idx)
{
    switch(type)
    {
    case FOTAS_START_REQ_EVT:
    {
        // ota_settings_write(SINGLE_FOREGROUND);
        ota_settings_write(DOUBLE_FOREGROUND);
        enum fota_start_cfm_status status;
        if(fw_signature_check(evt->fotas_start_req.digest, evt->fotas_start_req.signature))
        {
            status = FOTA_REQ_ACCEPTED;
        }else
        {
            status = FOTA_REQ_REJECTED;
        }
        prf_fotas_start_confirm(con_idx, status);
    }break;
    case FOTAS_FINISH_EVT:
        if(evt->fotas_finish.integrity_checking_result)
        {
            if(evt->fotas_finish.new_image->base != get_app_image_base())
            {
                ota_copy_info_set(evt->fotas_finish.new_image);
            }
            else
            {
                ota_settings_erase();
            }
            platform_reset(RESET_OTA_SUCCEEDED);
        }else
        {
            platform_reset(RESET_OTA_FAILED);
        }
        break;
    default:
        LS_ASSERT(0);
        break;
    }
}

```

```

static void prf_added_handler(struct profile_added_evt *evt)
{
    LOG_I("profile:%d, start handle:0x%x\n", evt->id, evt->start_hdl);
    switch(evt->id)
    {
    case PRF_FOTA_SERVER:
        prf_fota_server_callback_init(prf_fota_server_callback);
        create_adv_obj();
        break;
    default:
        break;
    }
}

```

```

case SERVICE_ADDED:
    dev_manager_prf_fota_server_add(NO_SEC);
    break;

```

```

case PROFILE_ADDED:
    prf_added_handler(&evt->profile_added);
    break;

```

3.3.2 密钥生成

前后台 OTA 的固件签名是一样操作的，详见：[3.2.1 密钥生成](#) 章节

3.3.3 带签名验证功能的固件生成

前后台 OTA 带签名验证功能固件却不一样，

打开我们需要进行添加后台 OTA 的应用工程，我这边测试使用的是 ble_uart_serve ，首先打开“dev\examples\ble\ble_uart_serve”文件夹,并按照 [3.3.1 添加相应代码](#) 章节所述进行代码的添加。

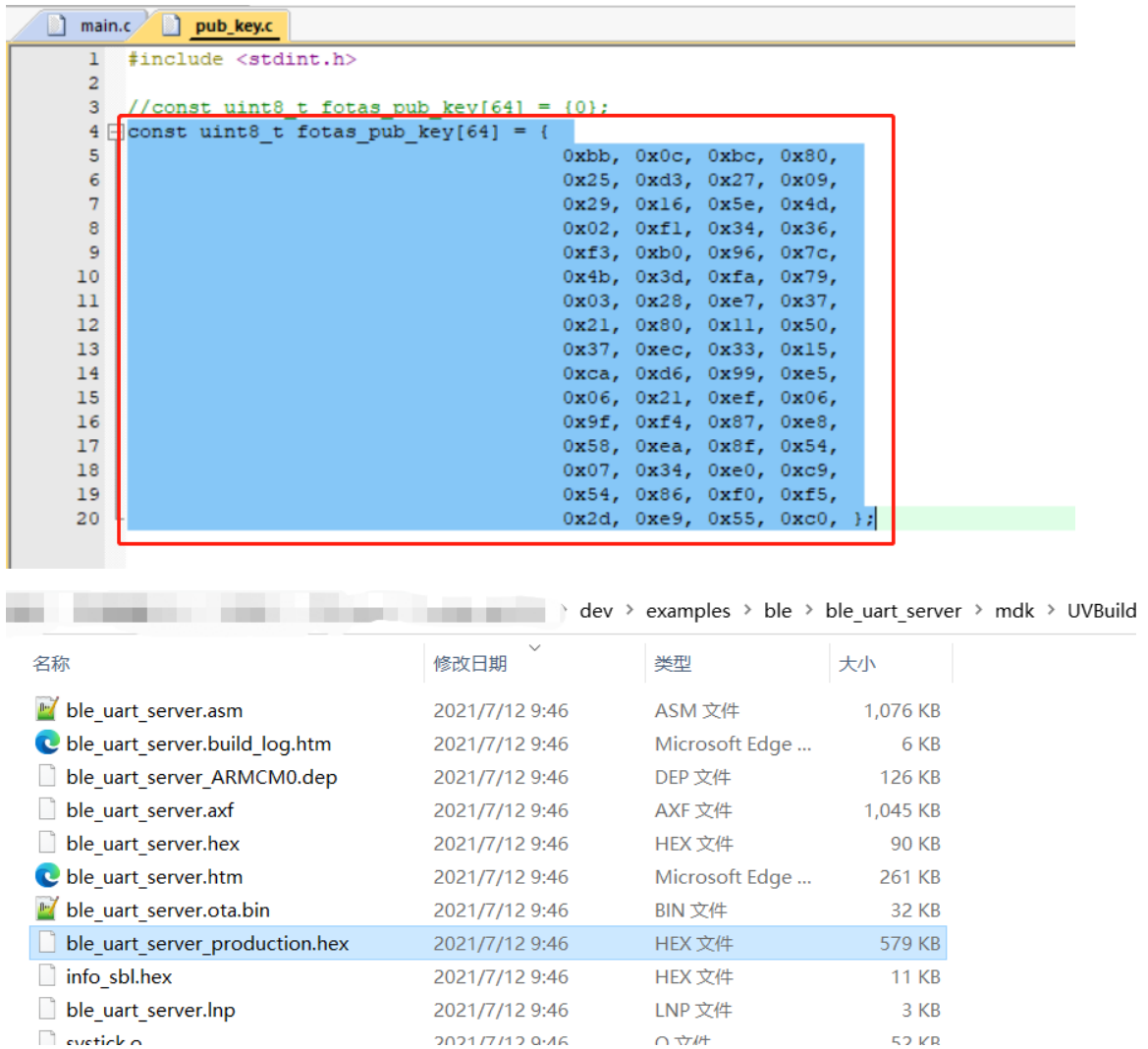
dev > examples > ble > ble_uart_server > mdk >

名称	修改日期	类型	大小
UVBuild	2021/7/12 9:46	文件夹	
ble_uart_server.map	2021/7/12 9:46	Linker Address ...	615 KB
ble_uart_server.uvguix.YUWEI	2021/7/12 16:51	YUWEI 文件	89 KB
ble_uart_server.uvoptx	2021/7/8 21:21	UVOPTX 文件	36 KB
ble_uart_server.uvprojx	2021/7/8 21:27	碘ision5 Project	30 KB
delay_asm.lst	2021/7/12 9:46	MASM Listing	3 KB
JLinkSettings.jlinkscript	2021/5/26 18:32	JLINKSCRIPT 文件	1 KB
ls_dbg_asm.lst	2021/7/12 9:46	MASM Listing	4 KB
sleep_asm.lst	2021/7/12 9:46	MASM Listing	5 KB
startup.lst	2021/7/12 9:46	MASM Listing	23 KB
svcall_asm.lst	2021/7/12 9:46	MASM Listing	4 KB

并将公钥“`verifying_key`”拷贝到“`pub_key.c`”文件中,（如下图所示）再编译相应的应用工程，我这边测试使用的是 ble_uart_serve ，所以会生成一个“`ble_uart_server_production.hex`”固件。

ls_ble_sdk-0.9 > dev > examples > ble > fota

名称	修改日期	类型	大小
compiler	2021/5/26 18:32	文件夹	
mdk	2021/5/26 18:32	文件夹	
app_config.h	2021/5/26 18:32	H 文件	1 KB
main.c	2021/5/26 18:32	C 文件	5 KB
pub_key.c	2021/5/26 18:32	C 文件	1 KB
SConscript	2021/5/26 18:32	文件	1 KB



```

1 #include <stdint.h>
2
3 //const uint8_t fotas_pub_key[64] = (0);
4 const uint8_t fotas_pub_key[64] = {
5     0xbb, 0x0c, 0xbc, 0x80,
6     0x25, 0xd3, 0x27, 0x09,
7     0x29, 0x16, 0x5e, 0x4d,
8     0x02, 0xf1, 0x34, 0x36,
9     0xf3, 0xb0, 0x96, 0x7c,
10    0x4b, 0x3d, 0xfa, 0x79,
11    0x03, 0x28, 0xe7, 0x37,
12    0x21, 0x80, 0x11, 0x50,
13    0x37, 0xec, 0x33, 0x15,
14    0xca, 0xd6, 0x99, 0xe5,
15    0x06, 0x21, 0xef, 0x06,
16    0x9f, 0xf4, 0x87, 0xe8,
17    0x58, 0xea, 0x8f, 0x54,
18    0x07, 0x34, 0xe0, 0xc9,
19    0x54, 0x86, 0xf0, 0xf5,
20    0x2d, 0xe9, 0x55, 0xc0, };
    
```

dev > examples > ble > ble_uart_server > mdk > UVBuild

名称	修改日期	类型	大小
ble_uart_server.asm	2021/7/12 9:46	ASM 文件	1,076 KB
ble_uart_server.build_log.htm	2021/7/12 9:46	Microsoft Edge ...	6 KB
ble_uart_server_ARMCM0.dep	2021/7/12 9:46	DEP 文件	126 KB
ble_uart_server.axf	2021/7/12 9:46	AXF 文件	1,045 KB
ble_uart_server.hex	2021/7/12 9:46	HEX 文件	90 KB
ble_uart_server.htm	2021/7/12 9:46	Microsoft Edge ...	261 KB
ble_uart_server.ota.bin	2021/7/12 9:46	BIN 文件	32 KB
ble_uart_server_production.hex	2021/7/12 9:46	HEX 文件	579 KB
info_sbl.hex	2021/7/12 9:46	HEX 文件	11 KB
ble_uart_server.lnp	2021/7/12 9:46	LNP 文件	3 KB
evstick	2021/7/12 9:46	文件	52 KB

3.3.4 签名文件生成

前后台 OTA 的签名文件是一样操作的，详见：[3.2.3 签名文件生成](#) 章节

3.4 OTA APP 升级测试

3.4.1 准备升级文件

在使用 APP 对模块进行升级的时候需要准备三个文件：

- 一、需要准备好带签名验证功能的固件，可以查看“带签名验证功能的固件生成”的篇章；
- 二、需要准备好签名文件以及做了签名操作的新固件，可以查看“签名文件生成”的篇章；
- 三、需要准备好 OTA 升级的手机版应用程序（.APK 文件）。

3.4.2 升级操作步骤

步骤如下：

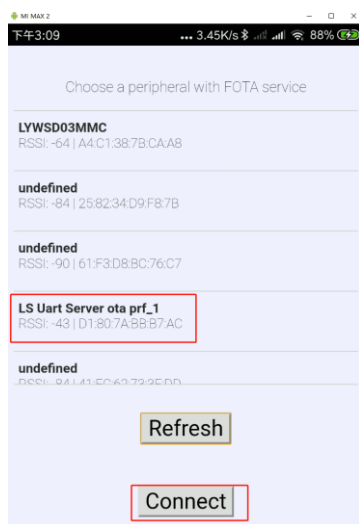
首先通过 j-flash 将“带签名验证功能的固件”下载到开发板中，此项操作我们暂且称为“老固件”；



然后再将手机中安装好 APP；



将签名文件以及做了签名操作的新固件下发给手机，打开 APP 后，选择这两个文件进入升级，搜索到设备后点击连接；



选择文件将做了签名操作的新固件和签名文件导入进去;



开始 OTA，注意地址位置需要和你们的代码的位置，以及前后台的模式进行调整。下图便是启动了验证以及升级成功的界面。

