

涂鸦蓝牙门锁公版 开发指南



文件修订历史

版本	日期	作者	变更描述
1.0	2019-11-04	苏钉	创建文档
1.1	2019-11-07	苏钉	1. 增加软件架构讲解 2. 增加 ota 流程 3. 增加本地存储管理
1.2	2019-11-13	苏钉	1. 增加串口指令格式 2. 增加串口模拟硬件的示例
1.3	2019-11-14	苏钉	增加固件烧录小节
1.4	2019-11-15	苏钉	修改示例固件路径
1.5	2019-11-27	苏钉	修改 license 申请
1.6	2019-12-18	苏钉	1. 修改本地存储管理图片 2. 修改 license 申请和 ota 流程图片 3. 增加出厂设置和动态密码串口指令
1.7	2019-12-25	苏钉	更新 license 申请小节
1.8	2020-1-15	苏钉	添加 bk3431q 相关的内容

目录

1 概述.....	4
2 软件架构	5
2.1 app	6
2.1.1 app_lock.....	6
2.1.2 app_common	6
2.2 app_port	6
2.3 tuyu_ble_sdk.....	6
2.4 芯片原厂 sdk.....	6
2.4.1 nrfs（nRF52832）	6
2.4.2 bk（bk3431q）	7
2.5 component.....	7
2.5.1 cpt_math.....	7
2.5.2 cpt_hash.....	7
2.5.3 cpt_mbedtls	7
2.5.4 cpt_fpe	7
2.5.5 cpt_easyflash.....	7
2.5.6 cpt_easylogger	8
3 重点解析	9
3.1 本地存储管理	9
3.1.1 硬件存储	9
3.1.2 事件存储	9
3.1.3 设置存储	9
3.2 蓝牙数据通道	10
3.3 dp 点解析	10
3.4 ota	10
3.5 产测	10
4 demo 讲解	12
4.1 demo 基本信息	12
4.2 demo 工程架构	12
4.3 固件烧录	12
4.4 license 申请	13
4.5 ota 流程	13
5 串口模拟硬件	14
5.1 串口指令格式（Hex）	14
5.2 示例（nRF52832）	15
5.3 示例（bk3431q）	16
6 其他平台移植	错误!未定义书签。
6.1 移植 easyLogger	错误!未定义书签。

1 概述

涂鸦蓝牙门锁公版指的是涂鸦蓝牙门锁的通用版本。

无论你的需求是什么，只要你开发的是**锁类业务应用**，想要通过**低功耗蓝牙**的通信方式接入**涂鸦智能 APP**和**涂鸦云平台**，你都可以基于涂鸦蓝牙门锁公版的 **sdk** 做开发。

我们的目标是：一份文档解决嵌入式软件开发过程中的所有问题。**所以遇到问题请首先查阅该文档。**

新手入门请首先跳转至 **5.2** 节，根据示例流程操作，通过串口模拟硬件，快速体验涂鸦锁类产品，该示例在后续开发过程中也是很好的调试工具。

2 软件架构

涂鸦低功耗蓝牙产品基于蓝牙通用配网协议，该协议主要实现了蓝牙通用配网流程和基本的数据通信协议，是涂鸦低功耗蓝牙产品的基础协议，协议详述参考《TuYa_ble_通用_sdk_协议_vxx.xx_xxxxxx》，源码位于“tuya_ble_sdk”文件夹内。如果用户的平台有对应的 Demo 例程，则无需关心该协议，涂鸦已经做好了所有的移植工作；如果用户使用新平台，则需要移植该协议，移植流程参考《TuYa_ble_通用_sdk_开发指南_vx.x.x_xxxxxx》。

在通用配网协议的基础上，涂鸦定义了一整套锁类产品的 dp 点（功能点）协议，详细描述了锁类应用业务和手机 APP 之间的通信逻辑。为了方便用户开发，涂鸦已经封装好了这些 dp 点协议，源码详见 app 层的 lock_dp_parser 和 lock_dp_report 文件，协议内容参考《TuYa_ble_锁类_dp 点规范_vxx.xx_xxxxxx》。

以上为涂鸦蓝牙配网、通信和锁类业务的实现，属于**主框架**。在主框架之外，还有一些重要的**支撑组件**。组件分为两部分：一部分是第三方组件，负责存储、调试等相关的通用功能；一部分是涂鸦的组件，负责授权、产测等相关的专用功能。

第三方组件主要包括 easyFlash、easyLogger 和 mbedtls。如果用户的平台有对应的 Demo 例程，则无需关心这些组件，涂鸦已经做好了所有的移植工作；如果用户使用新平台，则只需要对 easyFlash 和 easyLogger 组件做简单的移植即可，mbedtls 组件（主要为加密接口）无需改动。

涂鸦的组件是一些通用的源码包，适用于涂鸦的大部分低功耗蓝牙单点产品，此处仅以锁类应用为例介绍，其他应用可参考。主要包括授权产测、ota、本地存储管理和整机产测：授权产测用于烧录和校验通用配网协议所需的 mac、authkey 和 device id 等参数，详情参考《TuYa_ble_通用_授权产测协议_vxx.xx_xxxxxx》；ota 实现涂鸦自己的蓝牙 ota 协议（非 Demo 平台需移植），详情参考《TuYa_ble_通用_sdk_协议_vxx.xx_xxxxxx》；本地存储管理实现锁类应用在本地的存储逻辑，其他应用可参考；整机产测基于无线的方式实现对产品的整体功能测试，详情参考《TuYa_ble_通用_整机产测协议_vxx.xx_xxxxxx》。

以上对主框架和支撑组件都做了介绍。除此之外，还有一个概念需要强调一下：为了提高 app 层（包括涂鸦组件）的兼容性，将 app 层和其它层之间做了简单的隔离，隔离层叫做“app_port”。也就是说，在 app 层只能够见到隔离层封装好的底层 api（app_port 开头）和 app 层自己的 api 接口。

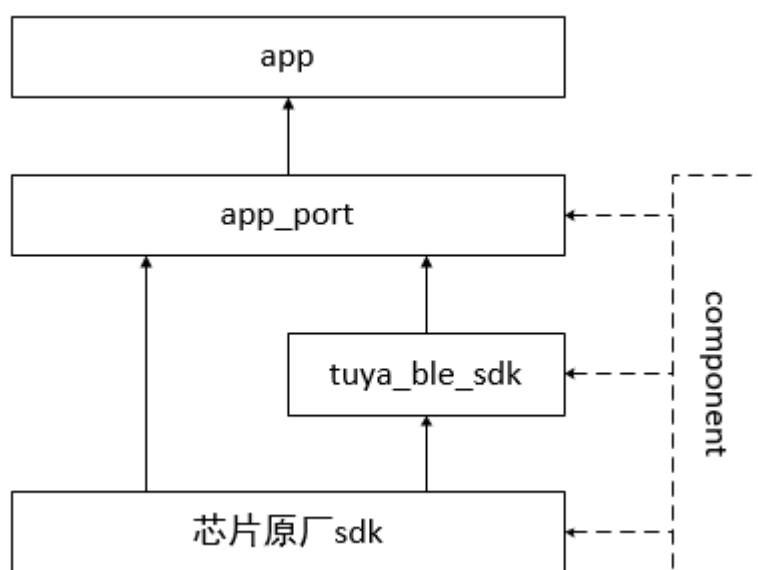


图 1 软件架构

2.1 app

2.1.1 app_lock

该文件夹主要实现应用层的锁类业务功能，仅适用于锁类产品。

lock_common: 未分类的

lock_dp_parser: dp 点解析，数据方向：手机→设备，《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》

lock_dp_report: dp 点上报，数据方向：设备→手机，《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》

lock_hard: 客户硬件接口和串口模拟硬件（入门或调试用）

lock_test: 整机产测，《Tuya_ble_通用_整机产测协议_vxx.xx_xxxxxx》

2.1.2 app_common

该文件夹主要实现应用层的通用功能，适用于各类蓝牙产品。

app_common: 未分类的

app_flash: 本地存储管理

app_ota: ota 功能，《Tuya_ble_通用_sdk_协议_vxx.xx_xxxxxx》

app_test: 授权产测，《Tuya_ble_通用_授权产测协议_vxx.xx_xxxxxx》

app_port: 如下

2.2 app_port

底层接口封装，为 app 层可能用到的所有接口提供统一的命名和管理，使应用层成为一个独立的模块，方便移植。

2.3 tuyable_sdk

涂鸦低功耗蓝牙通用配网 sdk，实现低功耗蓝牙设备和手机 APP 之间的配网流程和基础通信协议。

实现协议：《Tuya_ble_通用_sdk_协议_vxx.xx_xxxxxx》

开发指南：《Tuya_ble_通用_sdk_开发指南_vx.x.x_xxxxxx》

2.4 芯片原厂 sdk

nordic（nRF52832）原厂 sdk 的版本为：nRF5_SDK_15.3.0_59ac345

beken（bk3431q）原厂 sdk 的版本为：ble_3435_sdk_ext_39_0F0E

2.4.1 nrfs (nRF52832)

对 nordic 原厂 sdk 的接口封装：

nrfs_common: 未分类

nrfs_scan_adv: 扫描和广播

nrfs_ble: ble 参数

nrfs_svc: 服务和特征值

nrfs_uart: 串口
nrfs_timer: 定时器
nrfs_flash: flash
nrfs_gpio: gpio
nrfs_test: 研发测试用

2.4.2 bk (bk3431q)

对 bk 原厂 sdk 的接口封装:

bk_common: 未分类
bk_scan_adv: 扫描和广播
bk_ble: ble 参数
bk_svc: 服务和特征值
bk_uart: 串口
bk_timer: 定时器
bk_flash: flash
bk_gpio: gpio
bk_test: 研发测试用

2.5 component

此处指第三方组件。

2.5.1 cpt_math

实现一些常见的数学算法，例如校验和、crc16、crc32、字节反转等。
纯 c 语言库，无论任何平台，客户均无需对该组件进行移植和改动。

2.5.2 cpt_hash

哈希算法，纯 c 语言库，无论任何平台，客户均无需对该组件进行移植和改动。

2.5.3 cpt_mbedtls

实现一些常见的加密算法，例如 aes128_ecb、aes128_cbc、md5 计算等。
主要用于 tuya_ble_sdk 中的数据加密。
纯 c 语言库，无论任何平台，客户均无需对该组件进行移植和改动。

2.5.4 cpt_fpe

fpe 算法，纯 c 语言库，无论任何平台，客户均无需对该组件进行移植和改动。

2.5.5 cpt_easyflash

实现基于 Flash 的键值存储机制，主要用到如下 4 个 api（详情见注释）：
easyflash_init();

```
ef_set_env_blob(key, buf, size) ;  
ef_get_env_blob(key, buf, size) ;  
ef_del_env(key) ;
```

如果使用 Demo 平台，则已经实现了该组件的移植，客户可以直接使用；如果使用新平台，则需要客户移植该组件，至少需实现如上 4 个 api 的功能。

源码和教程: <https://github.com/armink/EasyFlash>

2.5.6 cpt_easylogger

实现整个 sdk 的 log 打印机制，主要用到如下 2 个 api（详情见注释）：

```
log_d();  
elog_hexdump(name, width, buf, size);
```

如果使用 Demo 平台，则已经实现了该组件的移植，客户可以直接使用；如果使用新平台，则需要移植该组件，至少需实现如上 2 个 api 的功能。

源码和教程: <https://github.com/armink/EasyLogger>

3 重点解析

3.1 本地存储管理

tuya_ble_sdk 占用 16k bytes Flash 空间用于存储授权和 tuya_ble_sdk 相关的系统信息，需单独分配，用户无需关心，也不可占用，起始地址为“TUYA_NV_START_ADDR”。

app 层默认占用 16k bytes Flash 空间用于存储离线记录数据和其他应用数据（key-value 存储结构，基于 easyFlash），用户可使用剩余空间，只要注意根据使用情况适当调整占用 Flash 空间的大小即可，起始地址为“EF_START_ADDR”，占用空间大小为“ENV_AREA_SIZE”，相关 api 如下：

```
uint32_t app_port_kv_set(const char *key, const void *buf, size_t size);
uint32_t app_port_kv_get(const char *key, void *buf, size_t size);
uint32_t app_port_kv_del(const char *key);
```

以上提到的 Flash 空间仅指存储区域，不包含代码区域。

本地存储主要指硬件存储、事件存储和设置存储，存储逻辑主要在“app_flash.c”文件中实现。

3.1.1 硬件存储

app 层已经实现了硬件存储的管理逻辑，支持最大 256 个硬件存储，一个开门方式为一个硬件，目前支持 5 种开门方式的存储，可根据实际情况调整每种开门方式的最大存储数量，默认每种开门方式为 10 个，如下图所示：

```
*/
//define different hard's max num, default value is 10, user can change it when needed
#define HARDID_MAX_PASSWORD      10
#define HARDID_MAX_DOORCARD      10
#define HARDID_MAX_FINGER        10
#define HARDID_MAX_FACE          10
#define HARDID_MAX_TEMP_PW       10
#define HARDID_MAX_TOTAL         (HARDID_MAX_PASSWORD+HARDID_MAX_DOORCARD+HARDID_MAX_FINGER+
```

3.1.2 事件存储

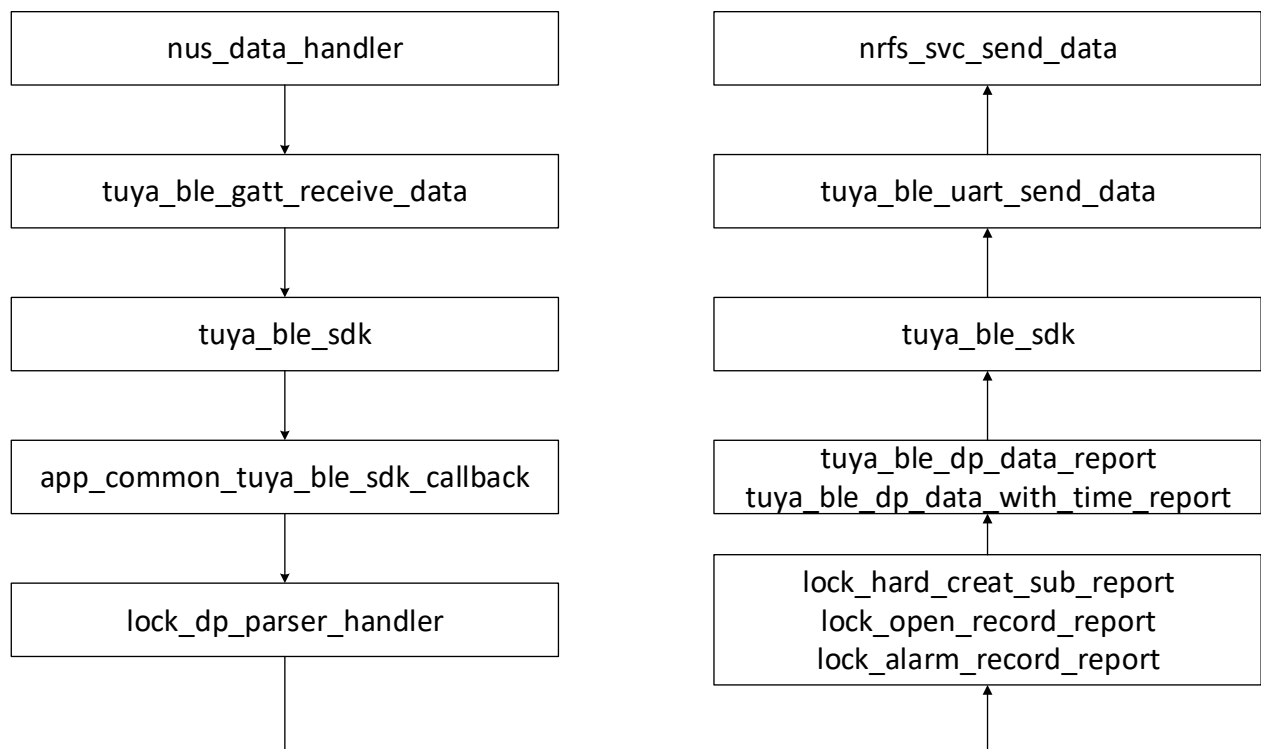
app 层已经实现了事件存储的管理逻辑，支持“EVTID_MAX”个事件存储，用户可根据实际情况调整事件的最大存储数量（若超出已分配 Flash 空间，需重新分配），默认存储最多 64 个事件，如下图所示：

```
//if user need more event storage, can change this value
#define EVTID_MAX                64
```

3.1.3 设置存储

app 层已经实现了设置存储的管理逻辑，用来存储跟门锁相关的设置项目，详见结构体“lock_settings_t”。想要修改门锁的默认设置，请找到 api “lock_settings_default”，修改相应参数即可。

3.2 蓝牙数据通道



上图给出了蓝牙数据在整个 sdk 中的内部传输通道，供用户参考。

3.3 dp 点解析

dp 点解析主要位于 `lock_dp_parser` 和 `lock_dp_report` 两个文件中，完全按照《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》协议实现，可以认为是该协议文档的 c 语言描述。

“`lock_dp_parser.c`”文件用于解析蓝牙接收到的锁类 dp 点数据，如果解析到的数据跟硬件相关，数据会被传到“`lock_hard.c`”文件中，该文件是 dp 点数据和本地逻辑的接口，也是用户最应该关心的文件；

“`lock_dp_report.c`”文件用于上报 dp 点数据给蓝牙，本地逻辑中如果有需要上报的数据，都可以通过调用该文件内封装好的 api 进行数据的上报。

3.4 ota

ota 主要位于 `app_ota` 文件中，demo 例程中是针对相应平台的实现过程，如果用户使用新平台，可以参考该例程，按照涂鸦提供的通信协议实现新平台上的 ota 过程。简单说：涂鸦提供 ota 文件存储平台（iot 平台）和数据通信协议，用户需要实现 ota 在本地的 Flash 存储逻辑（Demo 工程中已帮用户实现好）。

3.5 产测

涂鸦蓝牙锁类产品的产测主要分为两个部分：

1) 授权产测——主要包括检测产品信息，烧录 Mac 地址，device id（即 uuid）和 authkey 等；

2) 整机产测——主要包括射频 RSSI 测试（强制要求）和其他功能测试（可选）；

授权产测的主要功能是授权，主要包括检测产品信息，烧录 Mac 地址，device id（即 uuid）和 authkey 等通用配网协议中用到的重要参数，是产品安全性的基础；整机产测的主要功能是测试产品的众多外设，保证产品在产线生产过程中的良品率，涂鸦整机产测的优势在于提供了基于蓝牙无线通信方式的完整产测协议、产测 dongle 固件和产测上位机。关于产测的详细描述请参考相应协议文档。

4 demo 讲解

4.1 demo 基本信息

芯片: nRF52832

sdk: nRF5_SDK_15.3.0_59ac345

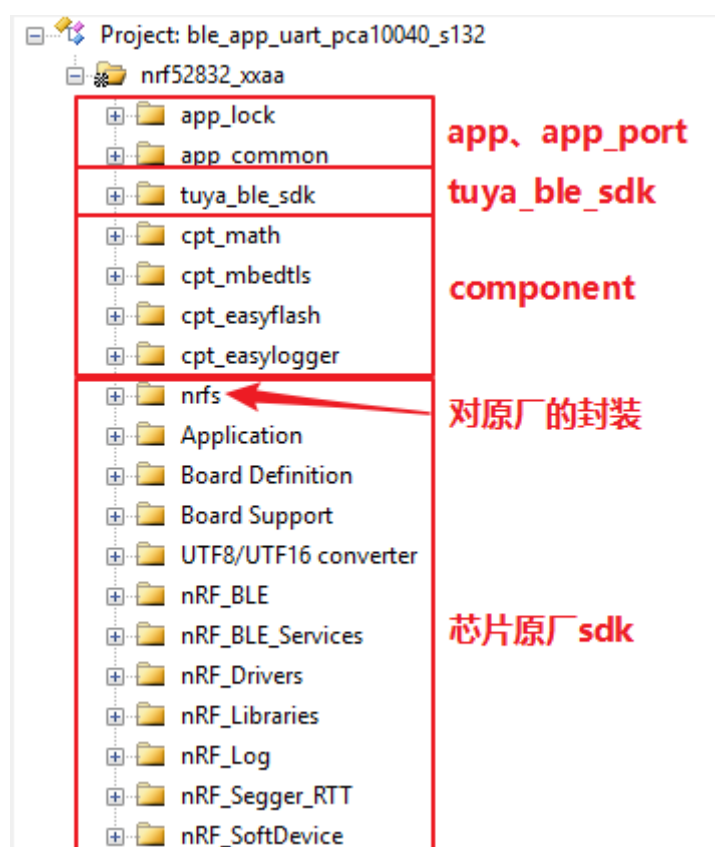
ide: μ Vision V5.28.0.0

project: ble_app_uart

串口: rx-8, tx-6 (授权测试)

调试信息: RTT_viewer

4.2 demo 工程架构



4.3 固件烧录

双击运行

“\nRF5_SDK_15.3.0_59ac345\examples\ble_peripheral\ble_app_uart\pca10040\s132\arm5_no_packs\hex\load_softdevice_bootloader_app.bat”

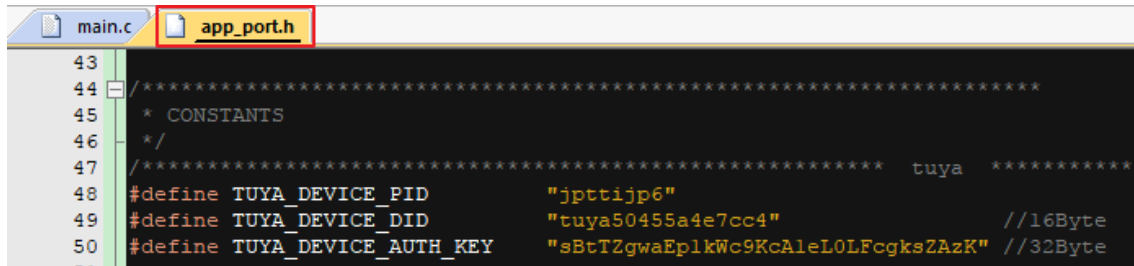
上述方式用于烧录 softdevice + bootloader + app, 仅烧录 app 可直接使用 keil 的 download。

4.4 license 申请

请联系项目经理参考《蓝牙门锁 SDKlicense 申请流程说明》进行申请。

正式项目请使用项目经理申请的 license 进行调试（完全擦除芯片后重新下载固件生效）！

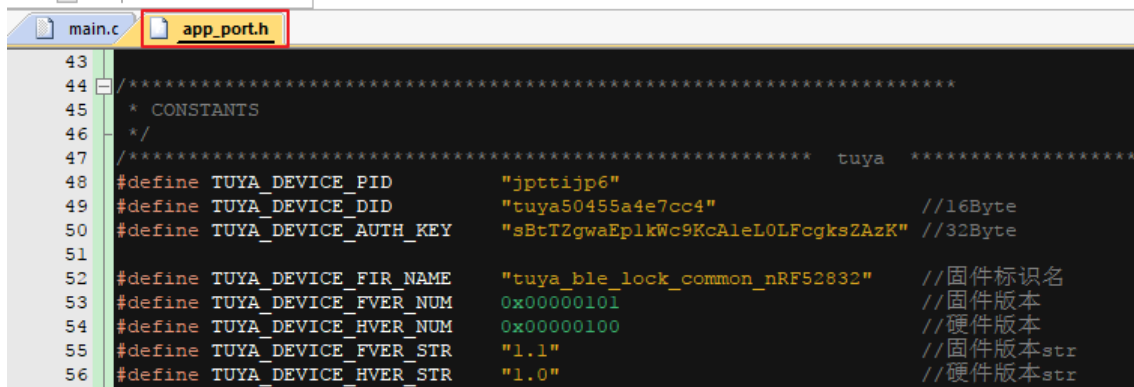
product id、authkey 和 device id 修改的位置如下图所示：



```
43
44
45 * CONSTANTS
46 */
47
48 #define TUYA_DEVICE_PID          "jpttjip6"
49 #define TUYA_DEVICE_DID          "tuya50455a4e7cc4" //16Byte
50 #define TUYA_DEVICE_AUTH_KEY     "sBtTZgwaEplkWc9KcAleL0LFcgksZAzK" //32Byte
```

4.5 ota 流程

1. 修改固件版本，固件版本规则参考《Tuya_ble_通用_sdk_协议_vxx.xx_xxxxxx》，如下图所示：




```
43
44
45 * CONSTANTS
46 */
47
48 #define TUYA_DEVICE_PID          "jpttjip6"
49 #define TUYA_DEVICE_DID          "tuya50455a4e7cc4" //16Byte
50 #define TUYA_DEVICE_AUTH_KEY     "sBtTZgwaEplkWc9KcAleL0LFcgksZAzK" //32Byte
51
52 #define TUYA_DEVICE_FIR_NAME     "tuya_ble_lock_common_nRF52832" //固件标识名
53 #define TUYA_DEVICE_FVER_NUM     0x00000101 //固件版本
54 #define TUYA_DEVICE_HVER_NUM     0x00000100 //硬件版本
55 #define TUYA_DEVICE_FVER_STR     "1.1" //固件版本str
56 #define TUYA_DEVICE_HVER_STR     "1.0" //硬件版本str
```

注意：有些平台在 ota 升级前还有其他一些平台相关的关键参数需要修改，需留意。

2. 重新编译工程，找到路径

“\nRF5_SDK_15.3.0_59ac345\examples\ble_peripheral\ble_app_uart\pca10040\s132\arm5_no_packs\ota” 内的 “tuya_ble_lock_common_nRF52832_xx.xx.bin” 文件；

3. 登录[涂鸦 iot 平台](#) → 找到需要操作的产品 → 进入产品页面 → 拓展功能 → 固件升级 → 创建新固件 → 填写新固件信息，并确定（若无权限联系涂鸦产品经理）即可；
4. 打开涂鸦智能 app，连接蓝牙设备 → 点击右上角的标志  → 检查固件升级，查看当前固件版本 → 设备信息 → 复制“虚拟 ID”并发送到电脑，填入步骤 3 中的测试设备中；
5. 再次点击步骤 4 中的“检查固件升级” → 更新 → 开始更新 → 等待更新完成即可，若失败请重试或检查固件。

5 串口模拟硬件

为方便客户调试和快速验证方案的可行性，使用简单的串口指令模拟多种硬件开门方式和其他指令。

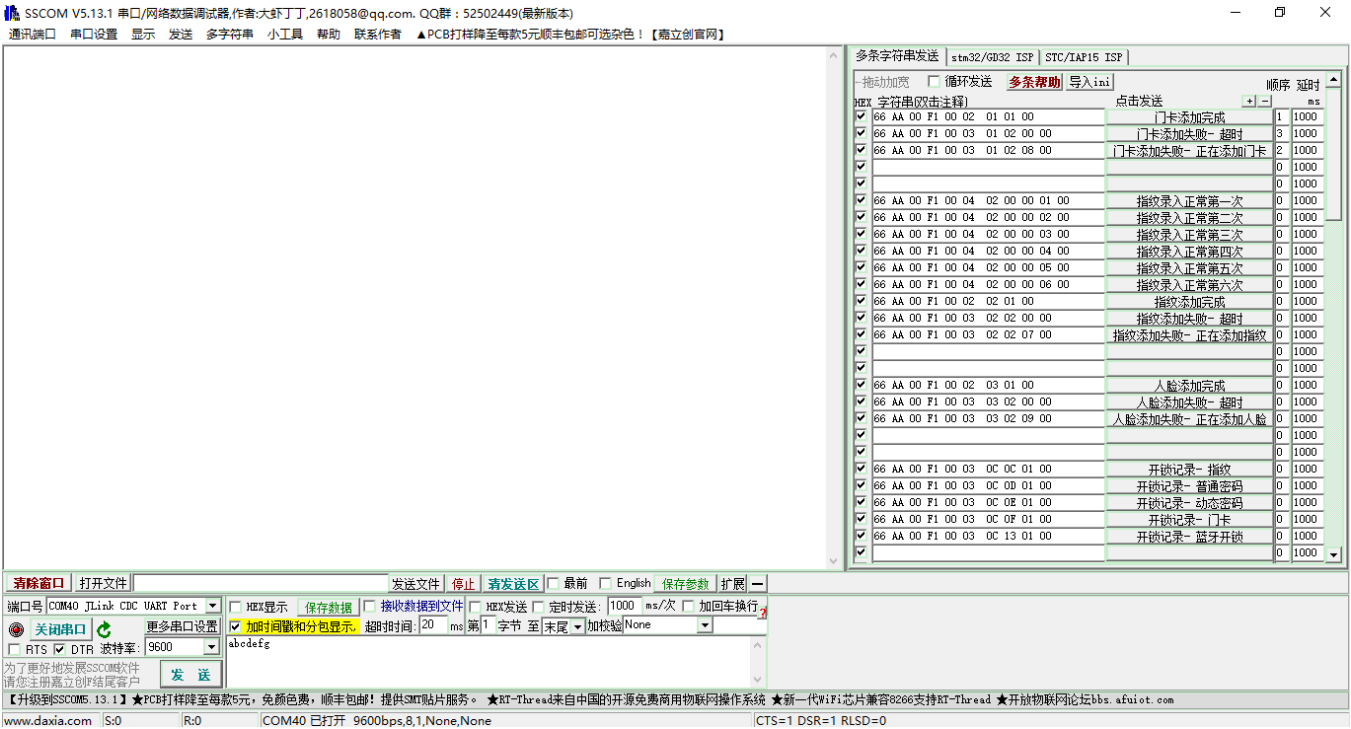
5.1 串口指令格式 (Hex)

固定前缀	长度	指令码（1 字节）	功能（1 字节）	数据（n 字节）		固定后缀
66 AA 00 F1 00	1 字节	01： 门卡	01： 添加完成	无		00
			02： 添加失败	添加失败原因（1 字节）		
		02： 指纹	00： 指纹录入中	0x00： 录入正常	录入次数（1 字节）	
			01： 添加完成	无		
			02： 添加失败	添加失败原因（1 字节）		
		03： 人脸	01： 添加完成	无		
			02： 添加失败	添加失败原因（1 字节）		
		0C： 开锁记录	dp_id1（1 字节）	硬件 id（1 字节）		
			dp_id=0x39（57）	组合枚举 + 多个硬件 id		
		0D： 报警记录	报警原因（1 字节）	无		
		0E： 状态显示	dp_id2（1 字节）	状态显示的数据内容（1 字节）		
		0F： 动态密码	无	动态密码（8 字节）		
		20： 出厂设置	无	无		

1. 添加失败原因
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.1.1 的“录入失败原因”。
2. dp_id1
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.2 的“dp_id”。
3. 硬件 id
默认 0x01，可根据不同 dp_id 值给予不同数值，
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.2 的“数据内容和取值范围”。
4. 报警原因
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.2 的“报警原因”。
5. dp_id2
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.3 的“dp_id”。
6. 状态显示的数据内容
详见《Tuya_ble_锁类_dp 点规范_vxx.xx_xxxxxx》文档的 3.3 的“数据内容和取值范围”。

5.2 示例 (nRF52832)

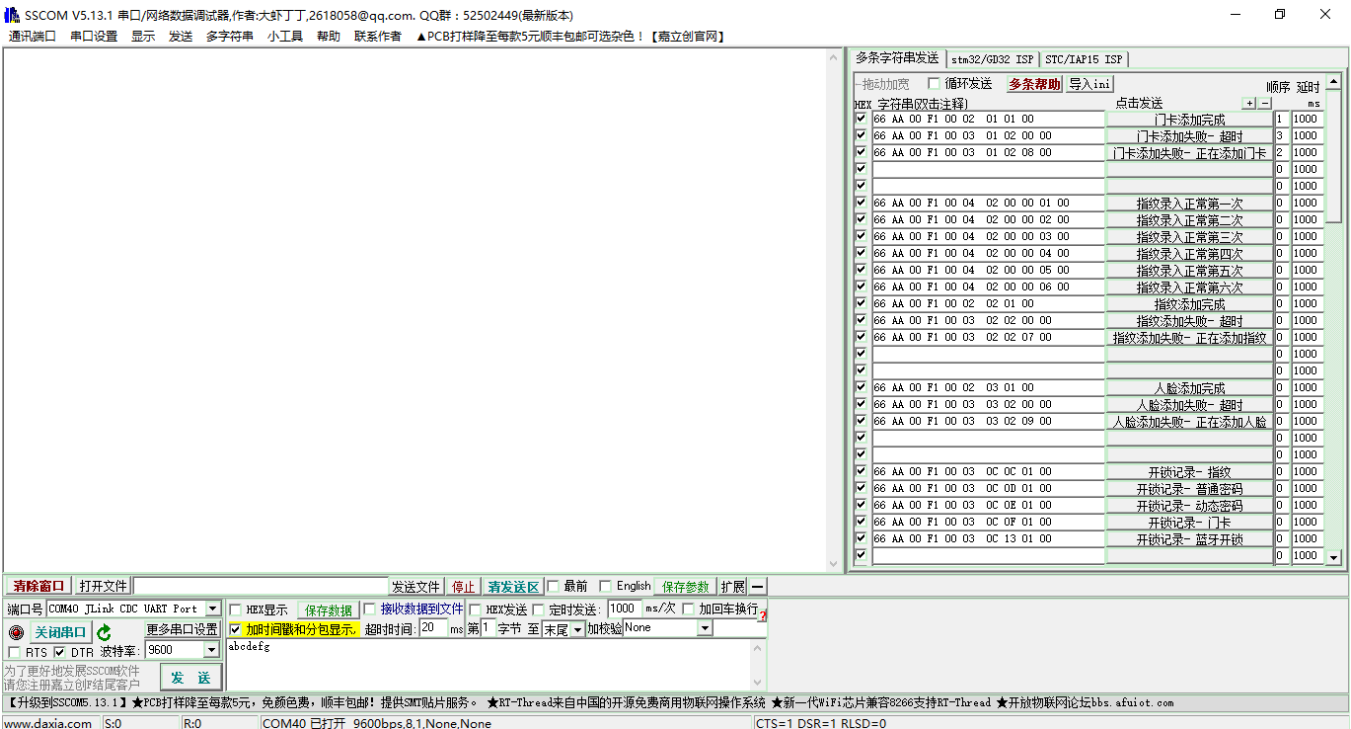
1. 准备一块 Nordic 官方开发板 PCA10040（推荐）或其他载有 nRF52832 芯片的开发板（串口保持跟 PCA10040 开发板一致，RX=8，TX=6），将开发板通过 j-link 连接电脑，打开 RTT-viewer 调试窗口；
2. 打开路径
“\nRF5_SDK_15.3.0_59ac345\examples\ble_peripheral\ble_app_uart\pca10040\s132\arm5_no_packs\hex”，双击“load_softdevice_bootloader_app.bat”下载固件（若脚本运行失败，请打开 readme 文档，下载安装“JLink_Windows_V652e.exe”和“nrfconnectsetup260.exe”后尝试），等待指令执行完成，调试窗口会出现 log；
3. 打开路径“\tuya_ble_lock_sdk\tools\uart_simulate_hard\SSCOM”内的“sscom5.13.1.exe”串口软件；
4. 点击“多字符串”按钮，会发现上节所述的**部分**串口指令已经在软件右侧文本框输入完毕；
5. 调整串口波特率为“9600”，即可开始串口模拟硬件，如下图所示：



6. 此时，打开涂鸦智能 app → 右上角+号 → 自动发现 → 开始搜索 → 找到“涂鸦公版门锁” → 下一步 → 添加（可选） → 等待绑定完成即可；
7. 进入“涂鸦公版门锁”界面 → 点击“手机开门”，即可看到手机开门的 log → 点击串口调试助手上的“开门记录- 指纹”，即可看到涂鸦智能 app 上提示“指纹解锁”字样 → 其他功能可自行探索。

5.3 示例 (bk3431q)

1. 准备一块 bk 官方 Demo 板，阅读《涂鸦_bk3431q 开发指南》；
2. 使用任一种方式烧录固件，烧录成功后需焊接上 Demo 板上的 P16、P17 引脚排针，使之接入 usb 转串口工具对应的 RX、TX 引脚，打开串口调试助手（921600），复位 Demo 板，即可观察到 log；
3. 打开路径“\tuya_ble_lock_sdk\tools\uart_simulate_hard\SSCOM”内的“sscom5.13.1.exe”串口软件；
4. 点击“多字符串”按钮，会发现上节所述的部分串口指令已经在软件右侧文本框输入完毕；
5. 调整串口波特率为“9600”，即可开始串口模拟硬件，如下图所示：



6. 此时，打开涂鸦智能 app → 右上角+号 → 自动发现 → 开始搜索 → 找到“涂鸦公版门锁” → 下一步 → 添加（可选） → 等待绑定完成即可；
7. 进入“涂鸦公版门锁”界面 → 点击“手机开门”，即可看到手机开门的 log → 点击串口调试助手上的“开门记录- 指纹”，即可看到涂鸦智能 app 上提示“指纹解锁”字样 → 其他功能可自行探索。