Version	Description	Date	Author
2.0.0	Telink Bluetooth SDK on iOS platform	2016/5/20	Shiqinglu
3.0.0	Fix some bugs	2017/7/13	Shiqinglu

目录

iOS SDK开发文档的思路简介

泰凌微电子（上海）有限公司

前言

Telink mesh是基于单一BLE连接，多个低功耗蓝牙设备基于mesh通信协议组成的网络；每个单一设备均有网络属性，属性用来标识mesh网络，该属性的主要构成有mesh name 、mesh password、ltk(ltk通常会设置成默认值，不建议外界修改)，并且该属性可被修改；

mesh这些属性提高了登录的隐私性，可以理解成是一个网络登陆一个登录许可，出厂默认name：“telink-mesh1”，password：“123”，ltk则作用于通信过程；当连接上符合要求的设备后，会请求登录，只有登录成功过后才能对设备指令操作；

由于mesh通信范围较蓝牙通信范围广(mesh为多跳中继网络)，通信过程均是由设备地址(u_DevAdress，通过Online Status notify获取)来唯一标示设备，而设备地址(u_DevAdress)也可被修改；为了合理管理设备，通常会建议修改设备mesh信息(name & password)，同时合理设置每个设备的地址(u_DevAdress);

一.Telink Mesh 工作流程

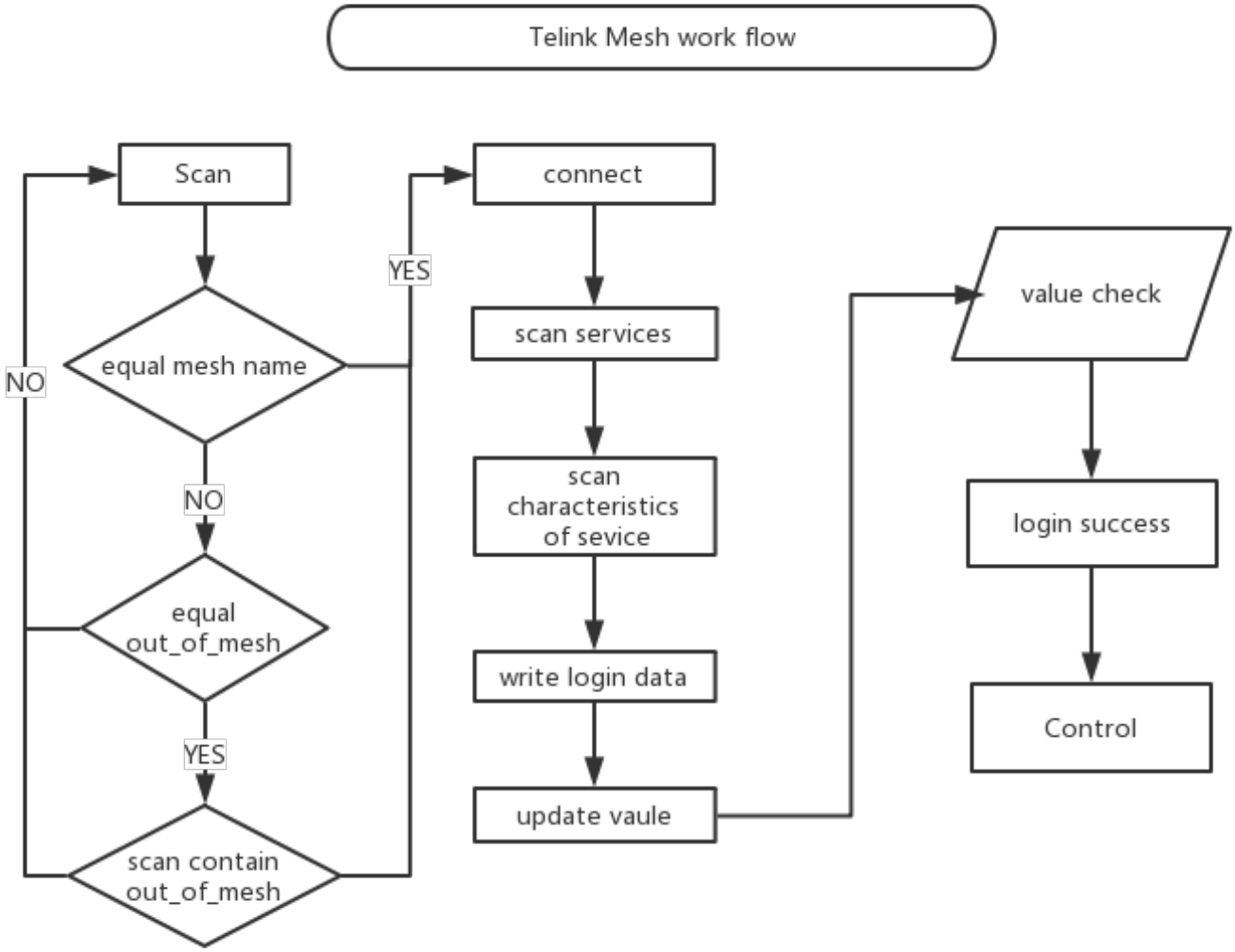

图 1

注：

1.扫描过程，会用传进去的mesh name进行过滤，由于广播包中看不到密码，无法校验密码，当扫描到符合要求（mesh name一致）的设备；

2.如果参数许可时，才会自动连接登录，其中连接后，会自动扫描服务ATT列表，以及服务中的特征值列表，当扫到目标特征值时，会默认给特征值write登录数据(发起登录请求)，当登录成功后，才能控制设备；

3.当设备登陆成功后，会每隔500ms，连续请求3次以获取Online Status，即执行方法

- (void)setNotifyOpenPro;(获取online status)

mesh中所有设备的u_DevAdress和light_Brightness以及light_Stata都是通过此方式获取，并且是通过u_DevAdress来唯一标识，后续是通过该标识来发起控制指令

二. SDK 介绍

在SDK中，有一个单例类“BTCentralManager”，该单例类中有一个私有的CBCentralManager属性作为蓝牙管理中心(管理中心和外设组成的Bluetooth mesh)，当该私有属性CBCentralManager被初始化时(同时设置单例类作为管理中心代理)，蓝牙会检查蓝牙开启状态，如果是开启状态，其中有一个提供一个 成员变量参数isNeedScan，供外界选择是否需要扫描，如果isNeedScan是YES，即开始扫描信号，反之亦然;

a. 其静态方法，生成单例管理

会生成一个单例，控制整个代理回调

```
+ (BTCentralManager*) shareBTCentralManager {
    static BTCentralManager *shareBTCentralManager = nil;
    static dispatch_once_t tempOnce=0;
    dispatch_once(&tempOnce, ^{
        shareBTCentralManager = [[BTCentralManager alloc] init];
        [shareBTCentralManager initData];
    });
    return shareBTCentralManager;
}
```

b.蓝牙管理中心初始化

初始化时，会检查蓝牙状态，调用下面代理方法，告知外界centralManager发生变化

```
- (void)centralManagerDidUpdateState:(CBCentralManager *)central {
    _centerState=central.state;
//whether central'state is on
    if (central.state == CBCentralManagerStatePoweredOn) {
        if (isNeedScan)
            [self startScanWithName:self.userName Pwd:self.userPassword];
    }else if (central.state==CBCentralManagerStatePoweredOff){
        [self stopConnected];
        [self stopScan];
    }
//call bcak state of central
    if (_delegate && [_delegate respondsToSelector:@selector(OnCenterStatusChange:)]) {
        [_delegate OnCenterStatusChange:self];
    }
}
```

c.发起扫描请求

当蓝牙管理中心被创建完成后，蓝牙处于开启状态，外界可通过此方法扫描设备

```
- (void)startScanWithName:(NSString *)nStr Pwd:(NSString *)pwd AutoLogin:(BOOL)autoLogin;
```

扫描回调，当发现了设备会回调下面方法，并把对应的参数回调出来

- (void)centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral *)peripheral advertisementData: (NSDictionary<NSString *,id> *)advertisementData RSSI:(NSNumber *)RSSI

当获取到符合要求的设备广播信息后，用模型BTDevice保存接收，并保存在_srcDevArrs中

数据结构如：<11021102 2211ffff 11022211 ffff0500 010f0000 01020304 05060708 090a0b0c 0d0e0f>

- (void)scanResult:(BTDevItem *)item;//代理方法

//flag 为DevChangeFlag_Add

- (void)OnDevChange:(id)sender Item:(BTDevItem *)item Flag:(DevChangeFlag)flag;

d.连接

蓝牙一经发现了设备，发起连接请求后，会可能有下面回调

连接成功

- (void)centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral *)peripheral

连接断开

- (void)centralManager:(CBCentralManager *)central didFailToConnectPeripheral:(CBPeripheral *)peripheral error:(NSError *)error

连接失败

- (void)centralManager:(CBCentralManager *)central didDisconnectPeripheral:(CBPeripheral *)peripheral error:(NSError *)error

均会通过下面代理回调出去

-(void)OnDevChange:(id)sender Item:(BTDevItem *)item Flag:(DevChangeFlag)flag;

涉及的 API

-(void)connectWithItem:(BTDevItem *)cItem

e.搜索服务特征值列表

当找到设备的 service 时，通过 uuid 订阅 services 中的 characteritics，保存目标 characteristics

- (void)peripheral:(CBPeripheral *)peripheral didDiscoverServices:(NSError *)error
- (void)peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:(CBService *)service error:(NSError *)error

f.登录模块

当获取到目标登录操作的 characteristic 时，可进行登录

- (void)loginWithPwd:(NSString *)pStr;

//给 characteristic 写数据后，如果设备有相应的回应，通常会通过下面 API 回调上来

- (void)peripheral:(CBPeripheral *)peripheral didUpdateValueForCharacteristic:(CBCharacteristic *)characteristic error:(NSError *)error

h.数据解析

- (void)pasterData:(uint8_t *)buffer IsNotify:(BOOL)isNotify;

当开启了 online status，有 notify 回来时，则会通过代理方法回调获取到的 model

- (void)notifyBackWithDevice:(DeviceModel *)model;

下面方法是有 feature UpdateValue 回来时

- (void)OnDevNofify:(id)sender Byte:(uint8_t *)byte;// notifyFeature update

- (void)OnDevCommandReport:(id)sender Byte:(uint8_t *)byte;// commandFeature update

解密回来的的数据解析，请参考附 1 文档 1

i.其他 API:

设置新的网络->mesh name & password 以及 ltk，但是 ltk 设置成默认值，不改变

uint8_t tlkBuffer[20]=
{0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,0x0,0x0,0x0,0x0};

类似的方法有 3 个，如下:

- (void)setNewNetworkName:(NSString *)nName Pwd:(NSString *)nPwd ltkBuffer:(uint8_t *)buffer;

- (void)setNewNetworkName:(NSString *)nName Pwd:(NSString *)nPwd WithItem:(BTDevItem *)item ltkBuffer:(uint8_t *)buffer;

- (void)setOut_Of_MeshWithName:(NSString *)addName PassWord:(NSString *)addPassWord NewNetWorkName:(NSString *)nName Pwd:(NSString *)nPwd ltkBuffer:(uint8_t *)buffer ForCertainItem:(BTDevItem *)item;

上述配置方法中有连接登录，连接登录前标定为配置网络，当登录成功后，执行

- (void)setNewNetworkDataPro;//私有方法

才会进行真正的配置工作—>发送指令告知设备修改网络

当配置成功后会有回调成功，pairFeature 会有 updatevalue back

```
- (void)sendPack:(NSData *)data; //发包

- (void)readFeatureOfselConnectedItem;// 获取直连灯属性

- (void)stopConnected;
```

j.指令的定制

所有的指令均会走到下面方法

```
- (void)sendCommand:(uint8_t *)cmd Len:(int)len;
```

参考附 1 文档 1

指令案例如

```
/**
 *  turn on / off all peripherals in mesh
 */
- (void)turnOffAllLight;//

- (void)turnOnAllLight;

/**
 *  turn on/off single peipheral
 *
 *  @param u_DevAddress
 */
- (void)turnOnCertainLightWithAddress:(uint32_t)u_DevAddress;//

- (void)turnOffCertainLightWithAddress:(uint32_t)u_DevAddress;

/**
 *  turn off/on single peipheral
 *
 *  @param u_DevAddress
 */
- (void)turnOffCertainLightWithAddress:(uint32_t)u_DevAddress; //

- (void)turnOnCertainGroupWithAddress:(uint32_t)u_GroupAddress; //

/**
 *  add / delete to group
 *
 *  @param targetDeviceAddress address of peripheral being added to group
 *  @param groupAddress        address of group
```

```
 */
- (void)addDevice:(uint32_t)targetDeviceAddress ToDestinateGroupAddress:(uint32_t)groupAddress;

- (void)deleteDevice:(uint32_t)deviceAddress ToDestinateGroupAddress:(uint32_t)groupAddress; //

/**
 *  set luminance of peripheral in group or single
 *
 *  @param lum
 */
- (void)setLightOrGroupLumWithDestinateAddress:(uint32_t)destinateAddress
WithLum:(NSInteger)lum; //

/**
 *  setting RGB of peripheral
 *
 *  @param destinateAddress address of single peripheral or group peripherals
 *  @param R
 *  @param G
 *  @param B
 */
- (void)setLightOrGroupRGBWithDestinateAddress:(uint32_t)destinateAddress WithColorR:(float)R
WithColorG:(float)G WithB:(float)B; // RGB

/**
 *  kick out peipheral (or peripherals, for group edit recommendation)
 *  resset all parameters(like ltk/password/) of peripheral to the state of factory set
 *  and mesh name is resset "out_of_mesh"
 *
 *  @param destinateAddress
 */
- (void)kickoutLightFromMeshWithDestinateAddress:(uint32_t)destinateAddress; //

/**
 *  set CT(0~1) value of peripheral
 *  @param destinationAddress address of peripheral
 */
- (void)setCTOfLightWithDestinationAddress:(uint32_t)destinationAddress AndCT:(float)CT;
```

k.加解密

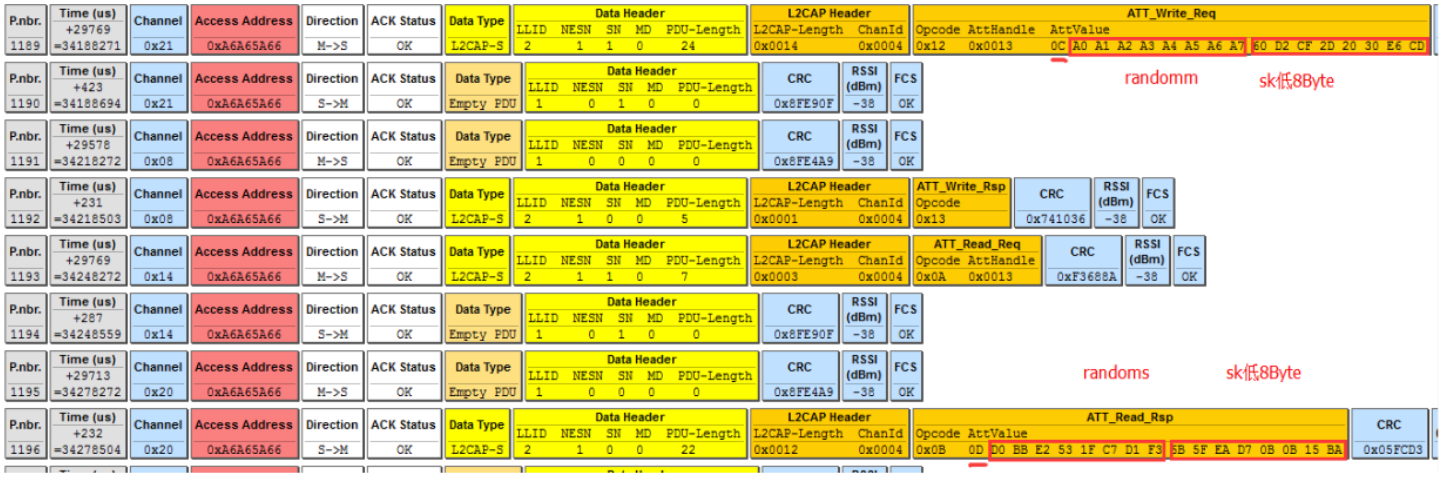

图 2

每次login时，都要求生成一个8个bytes的随机数，由Byte[8]表征

```
+ (BOOL)getRandPro:(uint8_t *)prand Len:(int)len {
    srand((int)time(0));
    memset(prand, 0, len);
    for (int i=0;i<len;i++)
        prand[i]=(uint8_t)random(255);
    return YES;
}
```

根据 meshName、password、loginRand这 3 个参数生成一个 sk，然后把 loginRand和生成的 sk 的低 8 个 byte(校验用)一起发送给模块

aes_att_er (meshName, password, loginRand, pcmd + 1)转化成oc写法，如下:

```
+ (BOOL)encryptPair:(NSString *)uName Pas:(NSString *)uPas Prand:(uint8_t *)prand
PResult:(uint8_t *)presult {
    uint8_t *tmpNetworkName = (uint8_t *)[uName cStringUsingEncoding:NSUTF8StringEncoding];
    uint8_t *tmpPassword = (uint8_t *)[uPas cStringUsingEncoding:NSUTF8StringEncoding];
    unsigned char               pNetworkName[16];
    unsigned char               pPassword[16];
    memset(pNetworkName, 0, 16);
    memset(pPassword, 0, 16);
    memcpy(pNetworkName, tmpNetworkName, strlen((char *)tmpNetworkName));
    memcpy(pPassword, tmpPassword, strlen((char *)tmpPassword));
    unsigned char sk[16], d[16], r[16];
```

```
    for (int i=0; i<16; i++) {
        d[i] = pNetworkName[i] ^ pPassword[i];
    }
    memcpy (sk, prand, 8);
    memset (sk + 8, 0, 8);
    aes_att_encryption (sk, d, r);
    memcpy (presult, prand, 8);
    memcpy (presult+8, r, 8);
    if (!(memcmp (prand, presult, 16))) return YES;
    return NO;
}
```

当手机收到模块返回的response后，结合response的data和meshName、password，以及loginRand生成加解密所需要的sectionKey，

aes_att_get_sk (meshName, password, loginRand, data, sectionKey) 转换成OC写法：

```
+ (BOOL)getSectionKey:(NSString *)uName Pas:(NSString *)uPas Prandm:(uint8_t *)prandm Prands:(uint8_t *)prands PResult:(uint8_t *)presult {
    uint8_t *tmpNetworkName = (uint8_t  *)[uName cStringUsingEncoding:NSUTF8StringEncoding];
    uint8_t *tmpPassword = (uint8_t *)[uPas cStringUsingEncoding:NSUTF8StringEncoding];
    unsigned char           pNetworkName[16];
    unsigned char           pPassword[16];
    memset(pNetworkName, 0, 16);
    memset(pPassword, 0, 16);
    memcpy(pNetworkName, tmpNetworkName, strlen((char *)tmpNetworkName));
    memcpy(pPassword, tmpPassword, strlen((char *)tmpPassword));
    unsigned char sk[16], d[16], r[16];
    for (int i=0; i<16; i++) {
        d[i] = pNetworkName[i] ^ pPassword[i];
    }
    memcpy (sk, prandm, 8);
    memcpy (sk + 8, prands, 8);
    aes_att_encryption (d, sk, r);
    memcpy (presult, r, 16);
    return YES;
}
```

固定写法

```
sec_ivm [8]数据结构如下：
//        phone: command
//        sec_ivm [0] = slave_mac_address[0];
//        sec_ivm [1] = slave_mac_address[1];
//        sec_ivm[2] = slave_mac_address [2];
//        sec_ivm[3] = slave_mac_address [3];
//        sec_ivm[4] = 1
//        sec_ivm[5] = sno[0];
//        sec_ivm[6] = sno[1];;
//        sec_ivm[7] = sno[2];

sec_ivs [8]数据结构如下：
//        sec_ivs[0] = slave_mac_address[0];
//        sec_ivs[1] = slave_mac_address[1];
//        sec_ivs[2] = slave_mac_address[2];
//        sec_ivs[3] = sno[0];
//        sec_ivs[4] = sno[1];
//        sec_ivs[5] = sno[2];
//        sec_ivs[6] = src[0]; //从发送的命令中获取，在 sno 后的第一个 byte
//        sec_ivs[7] = src[1]; //从发送的命令中获取，在 sno 后的第二个 byte
```

图3

1.加密

当指令buffer经过上图固定转换后，执行下面方法，进行加密

aes_att_encryption_packet(pair_sk, sec_ivm, buff + 13, 2, buff + 15, n - 15)

传参解析:pair_sk:就是在 login 成功后得到的 sk;

sec_ivm:详见图3;

buff+13:待发送的命令的 src 字段的第一个字节的地址;

2:此处恒为 2;

buff + 15:待发送的命令的 dst 字段的第一个字节的地址;

n – 15:需要加密的数据长度，从 dst 字段开始到命令结束的总共的长度。

对于 app 开发来说，因为 app 能看到的数据是从 sno 开始的，即

CMD-Status_All = u8 cmd[] = { 11,11,51,00,00,ff,ff,da,11,02,10};

所以: buff+13:应该等于“00 00” 的首地址，即 cmd+3buff+15:应该等于“ff ff” 的首地址，即 cmd+5

n – 15: 应该等于 总的数据长度 – 5;即 sizeof(cmd) – 5;

转换为oc写法

[CryptoAction encryptionPpacket:sectionKey Iv:sec_ivm Mic:buffer+3 MicLen:2 Ps:buffer+5 Len:15];

```
+ (BOOL)encryptionPpacket:(uint8_t *)key Iv:(uint8_t *)iv Mic:(uint8_t *)mic MicLen:(int)mic_len
Ps:(uint8_t *)ps Len:(int)len {
    uint8_t       e[16], r[16], i;
    ////////////// calculate mic //////////////////////
    memset (r, 0, 16);
    memcpy (r, iv, 8);
    r[8] = len;
    aes_att_encryption (key, r, r);
    for (i=0; i<len; i++) {
        r[i & 15] ^= ps[i];
        if ((i&15) == 15 || i == len - 1) {
            aes_att_encryption (key, r, r);
        }
    }
    for (i=0; i<mic_len; i++) {
        mic[i] = r[i];
    }
    ///////////////// calculate enc ////////////////////////
    memset (r, 0, 16);
    memcpy (r+1, iv, 8);
    for (i=0; i<len; i++) {
        if ((i&15) == 0) {
            aes_att_encryption (key, r, e);
            r[0]++;
        }
        ps[i] ^= e[i & 15];
    }
    return YES;
}
```

2.解密

aes_att_decryption_packet(pair_sk, sec_ivs, p + 14, 2, p + 16, p[2] - 10); 解密数据 传参解析:

pair_sk:就是在 login 成功后得到的 sk;

sec_ivs:详见图3;

p + 14:待解密命令的 dst 字段的第一个字节的地址;

2:此处恒为 2;

p + 16:待解密的命令的 op code 字段的第一个字节的地址;

p[2] - 10:待解密的数据长度，从 op code 字段开始到命令结束的总共的长度。

P[2]就是收到的命令的 L2cap_length

对于 app开发来说，因为 app 能看到的数据是从 sno 开始的，即 CMD-Status_All_Response =

u8 response[] = { 11,11,51,02,00, 02,00,db,11,02,ff,ff,ff,ff,ff,ff,00,00,04,01 };

所以:

p + 14:应该等于第二个“02 00” 的首地址，即 response +5

p + 16:应该等于“db”的地址，即 response +7

p[2] - 10: 应该等于 总长度 – 7;即 sizeof(response) – 7;

[CryptoAction decryptionPpacket:sectionKey Iv:sec_ivm Mic:buffer+5 MicLen:2 Ps:buffer+7 Len:13];

```
+ (BOOL)decryptionPpacket:(uint8_t *)key Iv:(uint8_t *)iv Mic:(uint8_t *)mic MicLen:(int)mic_len
Ps:(uint8_t *)ps Len:(int)len {
    uint8_t   e[16], r[16], i;
    ////////////////// calculate enc ///////////////////////
    memset (r, 0, 16);
    memcpy (r+1, iv, 8);
    for (i=0; i<len; i++) {
        if ((i&15) == 0) {
            aes_att_encryption (key, r, e);
            r[0]++;
        }
        ps[i] ^= e[i & 15];
    }
    ////////////// calculate mic ///////////////////////
    memset (r, 0, 16);
    memcpy (r, iv, 8);
    r[8] = len;
    aes_att_encryption (key, r, r);
    for (i=0; i<len; i++) {
        r[i & 15] ^= ps[i];
        if ((i&15) == 15 || i == len - 1) {
            aes_att_encryption (key, r, r);
        }
    }
    for (i=0; i<mic_len; i++) {
        if (mic[i] != r[i]) {
            return NO;                    //Failed
        }
```

```
    }
    return YES;
}
```

三.SDK 修改记录

1

修改时间：2017/05/22，修改人：石晴露

修复之前因错误修改手机时间造成命令延时错误的问题；

修改详情：

在Class BTCentralManager.m文件中

```
- (void)sendCommand:(uint8_t *)cmd Len:(int)len//对此方法做相应调整
```

2

修改时间：2017/7/13 修改人：石晴露

修改详情：修改文档

附 1

1：AN_BLE-15120203-C2_Communication Protocol for Telink BLE Mesh Light APP.pdf