# BLE广播方式实现

### 门禁端:



### 移动端:



## 相关说明:

门口机/围墙机作为中心设备,扫描指定UUID服务。

16bit的UUID格式: 0000XXXX-0000-1000-8000-00805f9b34fb, 其中XXXX可变。

• 门口机: 如楼栋号0001

扫描指定UUID:00000001-0000-1000-8000-00805f9b34fb

● 围墙机: 000a

扫描指定UUID:0000000a-0000-1000-8000-00805f9b34fb

广播数据: (注意,以下做数据校验的时候均为大写)

## Android

指定ServerUUID用于门口机过滤: 门口机UUID指定ServerUUID用于围墙机过滤: 围墙机UUID

• 厂商自定义数据:

。 厂商id: 0x5A57(即ZW)

○ 数据: 门口机+围墙机临时密码

Server data :

。 UUID 使用: 0000aaaa-0000-1000-8000-00805f9b34fb

○ 数据: 门口机+围墙机临时密码

### 注:厂商数据与Serverdata数据相同,目的为方便解析与加强校验减少数据结构改动

iOS

ios通过ibeacon发送数据。

- UUID,格式如:000A0001-0000-1000-8000-00805F9B34FB
  - 。 前八位中的前四位为围墙机位000A, 后四位为门口机楼栋号(如0001),其后面由于新增手机号原因, 所以不定, 只需要校验前8位科
- 名称Name为: 门口机+围墙机临时密码
- major+minor:为门口机+围墙机临时密码

注: Name与major+minor数据相同,目的为方便解析与加强校验及减少数据结构改动

### UUID校验(20171207改)

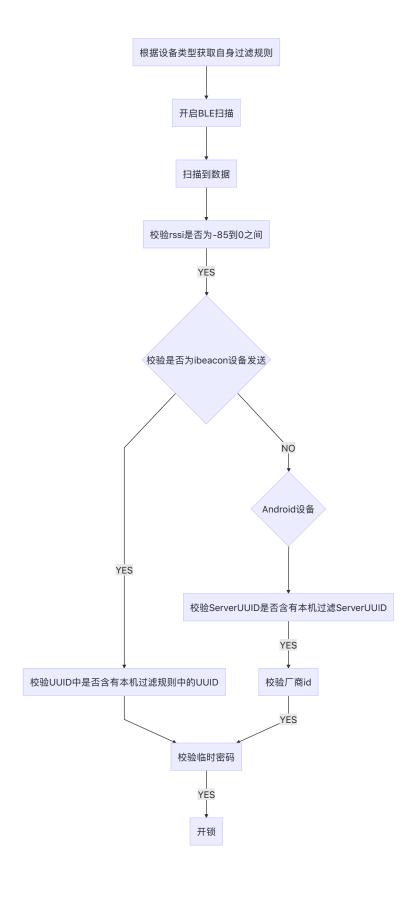
• 蓝牙门禁-围墙机(全部楼栋)



● 蓝牙门禁-门口机 (只有1个楼栋)

过滤广播楼栋号是否与本机楼栋号相等

### 接收端过滤验证步骤:



#### 接收端接收到数据例子如下:

//接收数据.

000A000100001000800000805F9B34FB//UUID014D//Major029A//MinorC5//iBeacon的信号发出强度值TxPower。这里是补码,转化为原码就是-59,,RSSI一起测距的基准07//后面7个字节09//完整设备名称标识3636363636//设备名称Name/\*数据说明:\*/Name: 666666//门口机临时密码Mac:

**7D:**B6:18:6F:7C:2FUUID: 000A0001-0000-1000-8000-00805F9B34FBMajor: 333Minor: 666TxPower: -59//iBeacon的信号发出强度値TxPowerr (接收信号强度) Received Signal Strength Indicator,低功耗蓝牙分四級//-60 ~ 0 4//-70 ~ -60 3//-80 ~ -70 2//<-80 1distance: 4.756481003459447

- 匹配UUID中前八位、解析出前四位和后四位、中含有该门口机过滤0001、则判定为ves
- Name为门口机临时密码: 666666
- Major+Minor为围墙机临时密码:333666

android 例

//android端接收数据例子: address:51:AD:E0:D9:A5:25,

name:null,

//解析说明02//后面2个字节01//flags 类型标识02//flags 类型数据09//后面9个字节FF //厂商设备类型标识777A //厂商id, 注意这是倒叙303634333 据05//后面5个字节03//服务如uid类型标识0100//服务如uid ,注意这里是倒叙,即实际是00010A00 //服务如uid ,注意这里是倒叙,即实际是000A09//后i16//服务数据server data 类型标识AAAA //服务数据uid393837363534//服务数据/\*数据说明:\*/厂商id:777A -> ZW 厂商数据:303634333231-> 064321//围墙机临时密码ServerUUid:01000A00 -> 0001000A,分割出来即为0001,000A

ServerUUIDData: AAAA393837363534 -> serveruuid AAAA ,数据987654//门口机临时密码

- 匹配UUID列表,中含有该门口机过滤0001,则判定为yes
- 厂商数据:为围墙机临时密码
- ServerUUIDData:为门口机临时密码

注意:临时密码组合后(因为转换为int类型的原因),如果低于6位,需要在前面补0到6位后比较(是否需要转换后比较取决与比较方式,如需要,如果转换为string则需要补0),ios的major和minor都为3位,不足需要在前补0到3位后再拼接成6位。

### 广播数据类型定义参考

此门禁中主要用到:

- Android
  - 。 FF (厂商设备类型标识)
  - 。 03 (服务uuid类型标识)
  - 16 (server data 类型标识)
  - 。 以下给出一些类型标识参考。
- ios
  - o FF
  - 。 4C00 (组织标识)
  - o 02 (ibeacon标识位)
  - 15 (22个字节标识长度)
  - 09 (完整设备名称标识)

以下是部分表示类型参考(仅供参考,ibeacon的是苹果自己定过的,与andorid的蓝牙联盟使用的协议不尽相同,所以类型标识不尽相同):

```
publicstaticfinalintBLE_GAP_AD_TYPE_FLAGS = 0x01;//< Flags for</pre>
discoverAbility.publicstaticfinalintBLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_MORE_AVAILABLE = 0x02;//< Partial list of 16 bit
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_COMPLETE = 0x03;//< Complete list of 16 bit service
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_MORE_AVAILABLE = 0x04;//< Partial list of 32 bit service
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_COMPLETE = 0x05;//< Complete list of 32 bit service
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_MORE_AVAILABLE = 0x06;//< Partial list of 128 bit service
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_COMPLETE = 0x07;//< Complete list of 128 bit service
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_SHORT_LOCAL_NAME = 0x08;//< Short local device</pre>
name.publicstaticfinalintBLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME = 0x09;//< Complete local device</pre>
\textit{name.} \textbf{publicstaticfinalint} \textbf{BLE\_GAP\_AD\_TYPE\_TX\_POWER\_LEVEL} = \textbf{0x0A;} // < \textit{Transmit power}
level. {\tt public static final int BLE\_GAP\_AD\_TYPE\_CLASS\_OF\_DEVICE} = {\tt 0x0D;} // < {\tt Class\ of\ Class\ of\
device.publicstaticfinalintBLE_GAP_AD_TYPE_SIMPLE_PAIRING_HASH_C = 0x0E;//< Simple Pairing Hash
C.publicstaticfinalintBLE_GAP_AD_TYPE_SIMPLE_PAIRING_RANDOMIZER_R = 0x0F;//< Simple Pairing Randomizer
\textit{R.} \textbf{publicstaticfinalint} \textbf{BLE\_GAP\_AD\_TYPE\_SECURITY\_MANAGER\_TK\_VALUE} = \textbf{0x10;} // < \textit{Security Manager TK} \textbf{TK} \textbf{NAMAGER\_TK\_VALUE} = \textbf{0x10;} // < \textbf{
Value.publicstaticfinalintBLE_GAP_AD_TYPE_SECURITY_MANAGER_00B_FLAGS = 0x11;//< Security Manager Out Of Band
Flags.publicstaticfinalintBLE_GAP_AD_TYPE_SLAVE_CONNECTION_INTERVAL_RANGE = 0x12;//< Slave Connection Interval
Range.publicstaticfinalintBLE_GAP_AD_TYPE_SOLICITED_SERVICE_UUIDS_16BIT = 0x14;//< List of 16-bit Service Solicitation
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_SOLICITED_SERVICE_UUIDS_128BIT = 0x15;//< List of 128-bit Service Solicitation
UUIDs.publicstaticfinalintBLE_GAP_AD_TYPE_SERVICE_DATA = 0x16;//< Service Data - 16-bit</pre>
UUID.publicstaticfinalintBLE_GAP_AD_TYPE_PUBLIC_TARGET_ADDRESS = 0x17;//< Public Target</pre>
Address.publicstaticfinalintBLE_GAP_AD_TYPE_RANDOM_TARGET_ADDRESS = 0x18;//< Random Target
 Address.publicstaticfinalintBLE_GAP_AD_TYPE_APPEARANCE = 0x19;//
Appearance.publicstaticfinalintBLE_GAP_AD_TYPE_ADVERTISING_INTERVAL = 0x1A;//< Advertising
 Interval.publicstaticfinalintBLE_GAP_AD_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS = 0x1B;//< LE Bluetooth Device
Address.publicstaticfinalintBLE_GAP_AD_TYPE_LE_ROLE = 0x1C;//< LE
Role.publicstaticfinalintBLE_GAP_AD_TYPE_SIMPLE_PAIRING_HASH_C256 = 0x1D;//< Simple Pairing Hash C-
 256.publicstaticfinalintBLE_GAP_AD_TYPE_SIMPLE_PAIRING_RANDOMIZER_R256 = 0x1E;//< Simple Pairing Randomizer R-
256.publicstaticfinalintBLE_GAP_AD_TYPE_SERVICE_DATA_32BIT_UUID = 0x20;//< Service Data - 32-bit
UUID.publicstaticfinalintBLE_GAP_AD_TYPE_SERVICE_DATA_128BIT_UUID = 0x21;//< Service Data - 128-bit
UUID.publicstaticfinalintBLE_GAP_AD_TYPE_3D_INFORMATION_DATA = 0x3D;//< 3D Information
Data.publicstaticfinalintBLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA = 0xFF;//< Manufacturer Specific Data.
```

### 临时密码规则(20171211改)

因MCU没有网络,扩大容错率,临时密码校验除了当前时间及以前时间的临时密码外,增加往后3分钟的临时密码。

- 临时密码方式生成参考: http://note.youdao.com/noteshare?id=69569dfb713175abd7604eb5863e9515
- 蓝牙声波门禁验证密码特别说明(有效密码个数以此为准):
  - o 蓝牙: 收到广播后,解析数据中的临时密码,用5个临时密码(前1分钟+当前分钟+后3分钟一共5个)进行比较,成功则开锁,失败
  - 。 声波/按键输入: 收到声波或按键输入后,解析数据中的临时密码,用63个临时密码(前59分钟+当前分钟+后3分钟)进行比较,成功则不处理。

### 20171027增加时间广播发送规则

由于模块需要较时,故在开锁广播发送完成之后,添加两秒时间广播,用于MCU较时。

注意: 此广播不一定会收到。此广播与前一个门锁广播无任何关联关系

#### Android

广播数据只包含厂商自定义数据,如下:

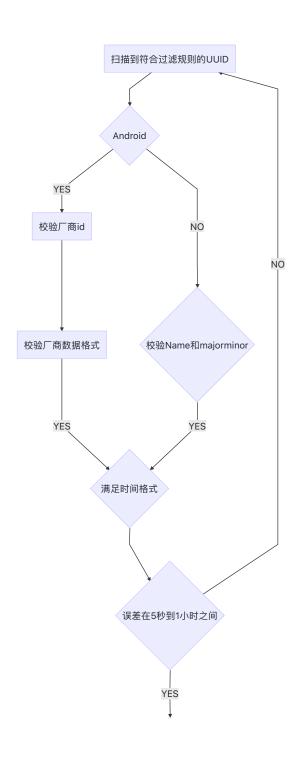
- 服务UUID
  - 。 0001000A 用于做过滤。
- 厂商自定义数据(两种可能性,可能是网络时间或者本地时间):
  - 。 网络时间:
    - 厂商id: 0x5449(即TI)
    - 数据:网络时间,格式:yyyyMMddHHmmss
  - 。 二 本地时间:
    - 厂商id: 0x4C54(即LT)
    - 数据: 本地时间,格式: yyyyMMddHHmmss

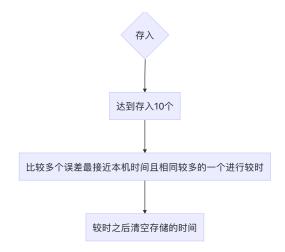
ios

ios通过ibeacon发送数据。

- UUID,格式如: 000A0001-0000-1000-8000-00805F9B34FB
  - 。 前八位中的前四位为围墙机位000A,后四位为门口机楼栋号(如0001),用于过滤。
- Name(两种可能性,可能是网络时间或者本地时间):
  - 。 网络时间:
    - TI+网络时间(格式同上)。如: TI20171027172122
  - 。 二 本地时间:
    - LT+本地时间(格式同上)。如: LT20171027172122
- major+minor: 000000,即全0

### 校验





### 例子

Android

11//11字节FF //厂商type4954//TI3230313731303237313732313232//2017102717212205//5字节03//服务uuid类型标识0100//0001000A //0

### 20170922广播信息添加手机号说明

此说明携带数据,仅门口机解析需要使用。假设手机号加密后16进制string为: 5251504F4E4D4C4B4A4742

### Android

- 广播中添加被扫描响应数据:
  - 。 厂商自定义数据:
    - 厂商id: 0x7A77(即zw)
    - 数据:加密的手机号字段:52574E534C47484A47413F

ios

- 广播中改UUID为: 000A0001-5251-504F-4E4D-4C4B4A474200
  - 。 前八位中的前四位为围墙机位000A,后四位为门口机楼栋号(如0001),后面为加密的手机字段(补足UUID32位,不足补0),修改
    - 000A0001-5251-504F-4E4D-4C4B4A00

### 手机号加密方式

- 1. 取出手机每一位转换为16进制字符串
- 2. 将得到的每一位16进制字符串转为10进制数字
- 3. 第一位数字+33, 第二位+31, 第三位加29···第十一位加11
- 4. 将每一位加后的数字转换为16进制字符串
- 5. 拼接数字得到最终加密后的16进制字符串

如:

java 端:

```
* 传入手机号得到加密后的byte数绝*/publicbyte[] encode(Stringphone) {
    String phone = "12345678985";char[] chars = phone.toCharArray();
    Stringencode = "";
   byte[] bytes;
   intaddNum = 33;//从33开始加//遍历每一位,计算每一位for(inti = 0; i < chars.length; i++, addNum -= 2) {
       StringcharStr = String.valueOf(chars[i]);
       System.out.println("每一位转换位string后:"+ charStr);
       // 获取手机号位的hex值StringhexStr = Useful.str2HexStr(charStr);
       System.out.println("hexStr:"+ hexStr);
       //手机号位转10进制值intstr10 = Integer.parseInt(hexStr, 16);
       //+数字intintEned = str10 + addNum;
       System.out.println("addNum之后:"+ intEned);
       StringstrEnd = Integer.toHexString(intEned);
       System.out.println("addNum之后转Hex:"+ strEned);
       encode = encode + strEned;//拼接每一位计算后的数字}
   System.out.println("加密后16进制byte:"+ encode);
   //这个就是我们要传入的bytes了bytes = Useful.hexStringToBytes(encode);
   returnbytes;
```