版权所有

BLE4.0

低功耗蓝牙协议总结

QQ 群 公众号 论坛

编写：刘权

作者保留本文的所有版权

（本文得以面世感谢长沙景嘉微电子股份有限公司）

目 录

版本控制......i
第一章 协议组成图......1
1.1、协议由 HOST 层和 CONTROLLER 层组成......1
1.1.1、CONTROLLER 组成......1
1.1.2、HOST 组成......2
第二章 控制器......2
2.1、PHY 物理层......2
2.1.1、频段......2
2.1.2、调制......2
2.1.3、射频信道......3
2.1.4、发射功率......3
2.2、链路层描述......4
2.2.1、链路层的状态机......4
2.2.2、状态描述......5
2.2.3、bit 数据流格式......7
2.2.4、Preamble 前导码......8
2.2.4、Access Address 接入地址......9
2.2.6、Cyclic Redundancy Check (CRC) 循环冗余码校验......11
2.2.7、DATA WHITENING 数据白化......11
2.3、链路层报文结构......13
2.3.1、PDU 的报头和长度......14
2.3.2、广播通道的 PDU 格式......15
2.3.3、数据通道的 PDU 格式......18
2.4、链路层设备滤波......20
2.5、NRF51822 的 Radio......21
2.5.1、EasyDMA......21
2.5.2、包结构......22
2.5.3、设备地址--白名单......26
2.5.4、Radio 的状态机和时间参数......27
2.5.5、Radio 的基本配置......32
2.6、信道......32
2.6.1、调频......34
2.6.2、自适应调频......35
2.7、非连接状态......40
2.7.1、就绪态......40
2.7.2、广播态......40
2.7.2.1、广播通道选择......40
2.7.2.2、广播间隔......41
2.7.2.3、非定向可连接广播事件(ADV_IND)......42

版权所有

2.7.2.4、定向可连接广播事件(ADV_DIRECT_IND) 44
2.7.2.5、非定向不可连接事件(ADV_NONCONN_IND) 45
2.7.2.6、可发现不可连事件(ADV_DISCOVER_IND/ADV_SCAN_IND) 46
2.7.3、扫描态 46
2.7.4、发起态 47
2.7.5、软件设计广播状态流程图 47
2.8、连接状态 48
2.8.1、连接事件 49
2.8.2、监管超时 50
2.8.3、连接事件传输窗口 50
2.8.4、连接状态--主机 54
2.8.5、连接状态--从机 55
2.8.6、连接事件关闭 57
2.8.7、窗口扩展 58
2.8.8、软件设计连接态流程图 60
2.9、空中接口包 62
2.9.1、广播通道 PDU 62
2.9.1.1、广播数据的结构 62
2.9.1.1.1、广播类型定义 AD Type 63
2.9.1.1.2、广播数据定义 AD Data 64
2.9.1.2、Advertising PDUs 70
2.9.1.2.1、ADV_IND、ADV_NONCONN_IND、ADV_SCAN_IND 70
2.9.1.2.2、ADV_DIRECT_IND 71
2.9.1.3、Scanning PDUs 71
2.9.1.3.1、SCAN_REQ 71
2.9.1.3.2、SCAN_RSP 72
2.9.1.4、Initialing PDUS 73
2.9.1.4.1、CONNECT_REQ 73
2.9.2、数据通道 PDU 73
2.9.2.1、LL Data PDU 74
2.9.2.2、LL Control PDU 75
2.9.2.2.1、LL_CONNECTION_UPDATE_REQ 77
2.9.2.2.2、LL_CHANNEL_MAP_REQ 78
2.9.2.2.3、LL_TERMINATE_IND 79
2.9.2.2.4、LL_ENC_REQ 80
2.9.2.2.5、LL_ENC_RSP 81
2.9.2.2.6、LL_START_ENC_REQ 82
2.9.2.2.7、LL_START_ENC_RSP 83
2.9.2.2.8、LL_UNKNOWN_RSP 83
2.9.2.2.9、LL_FEATURE_REQ 83
2.9.2.2.10、LL_FEATURE_RSP 84
2.9.2.2.11、LL_PAUSE_ENC_REQ 85
2.9.2.2.12、LL_PAUSE_ENC_RSP 85
2.9.2.2.13、LL_VERSION_IND 85

版权所有

2.9.2.2.14、LL_REJECT_IND 87
2.9.3、连接态的数据包确认和重发以及多数据发送标志 89
2.9.3.1、序列号(SN)....89
2.9.3.2、预期序列号(NESN) 90
2.9.3.3、更多数据(MD) 90
2.9.3.4、SN、NESN 和 MD 应用的例子....90
2.9.3.5、确认和重发的软件实现....94
2.10、直接测试单元(DTU) 96
2.10.1、UART 测试接口....96
2.10.2、测试模式 RADIO 配置 97
2.10.3、发射机测试....98
2.10.4、接收机测试....99
2.10.5、命令和事件....100
2.10.5.1、命令....100
2.10.5.2、事件....102
2.10.5.2.1、测试状态事件....103
2.10.5.2.2、测试报告报文事件....103
2.10.6、DTU 软件设计....104
2.10.7、NRF51822 的测试结果 106
2.10.8、测试结果对应的命令和事件....109
2.11、主机控制接口(HCI)....113
2.11.1、物理接口....113
2.11.1.1、UART....114
2.11.2、逻辑接口—HCI 包格式 115
2.11.2.1、命令数据包....115
2.11.2.2、事件数据包....117
2.11.2.3、数据包....119
2.11.3、命令和事件类型....120
2.11.4、HCI 软件设计 130
2.11.5、HCI 模拟数据传输 131
第三章 主机....135
3.1、逻辑链路控制和适配协议(L2CAP) 138
3.1.1、L2CAP 信道 138
3.1.2、L2CAP 数据包格式 139
3.1.3、低功耗信令信道包格式....140
3.1.3.1、命令拒绝....141
3.1.3.2、连接参数更新请求和响应....142
3.2、属性构成....145
3.2.1、属性句柄(Attribute Handle) 148
3.2.2、属性类型(Attribute Type) 149
3.2.3、属性值(Attribute Value)....151
3.2.4、属性许可(Attribute Permissions)....151
3.3、GATT 服务器构成 154

版权所有

3.3.1、服务.....155
3.3.1.1、服务声明.....156
3.3.1.1.1、服务声明格式.....158
3.3.2、包含服务«Include».....159
3.3.3、属性类型分组.....160
3.3.4、特性«Characteristic».....161
3.3.4.1、特性声明.....162
3.3.4.1.1、属性值—特性性质(Characteristic Properties).....162
3.3.4.1.2、属性值—特性的属性句柄(Characteristic Value Attribute Handle).....163
3.3.4.1.3、属性值—特性的属性类型(Characteristic UUID).....163
3.3.4.2、特性值声明.....163
3.3.4.3、特性描述符声明.....164
3.3.4.3.1、特性扩展性质描述符.....164
3.3.4.3.2、特性用户描述描述符.....165
3.3.4.3.3、客户端特性配置描述符.....165
3.3.4.3.4、服务器特性配置描述符.....166
3.3.4.3.5、特性表示格式描述符.....167
3.3.4.3.6、特性聚合格式描述符.....169
3.4、属性协议(ATT).....170
3.4.1、通信协议方法.....170
3.4.2、属性协议包格式.....171
3.4.3、属性协议 PDUs.....171
3.4.3.1、交换 MTU.....172
3.4.3.2、找信息请求\应答(Find Information Request\Response).....173
3.4.3.3、按类型值查找请求\应答(Find By Type Value Request\Response).....175
3.4.3.4、按类型读请求\应答(Read By Type Request\Response).....177
3.4.3.5、读请求\应答(Read Request\Response).....180
3.4.3.6、大对象读请求\应答(Read Blob Request\Response).....182
3.4.3.7、多重读取请求\应答(Read Multiple Request\Response).....183
3.4.3.8、按组类型读取请求\应答(Read By Group Type Request\Response).....184
3.4.3.9、写请求\应答(Write Request\Response).....187
3.4.3.10、写命令(*Write Command*).....188
3.4.3.11、签名写命令(Signed Write Command).....189
3.4.3.12、准备写请求\应答(Prepare Write Request\Response) 和执行写请求\应答(ExecuteWrite Request\Response).....190
3.4.3.13、句柄通知(Handle Value Notification).....194
3.4.3.14、句柄指示\确认(Handle Value Indication\Confirmation).....195
3.4.3.15、错误应答.....196
3.5、GATT 规程和 ATT 协议映射.....201
3.5.1、GATT 规程.....201
3.5.1.1、发现服务和特性.....201
3.5.2、ATT 协议与 GATT 映射表.....202
3.6、安全管理(Security Manager (SM)).....211

3.6.1、加密做了什么和加密需求......211
3.6.2、加密相关计算公式......213
3.6.3、加密配对绑定过程......215
3.6.3.1、配对特征交换得到临时密钥(TK)值......217
3.6.3.1.1、Input 和 Output 能力......219
3.6.3.1.2、Just Work:只工作......220
3.6.3.1.3、Passkey Entry:输入密码......221
3.6.3.1.4、Out of Band:带外......221
3.6.3.2、身份确认以及短期秘钥(STK)生产......222
3.6.3.2.1、身份确认值计算......222
3.6.3.2.2、短期秘钥(STK)值计算......223
3.6.3.3、特定密钥计算......224
3.6.3.3.1、长期密钥 LTK 计算......224
3.6.3.3.2、设备地址类型和身份解析密钥 IRK......226
3.6.3.3.3、连接签名解析密钥 CSRK......228
3.6.3.3.4、签名计算......228
3.6.4、加密标准 AES-CCM......229
3.6.5、完整加密过程图表......232
3.6.6、安全管理传输协议......235
3.6.6.1、安全管理命令包格式......235
3.6.6.2、配对请求 Pairing Request 和配对应答 Pairing Response......235
3.6.6.3、配对确认值 Pairing Confirm......238
3.6.6.4、配对随机数 Pairing Random......239
3.6.6.5、配对失败 Pairing Failed......240
3.6.6.6、加密信息 Encryption Information......241
3.6.6.7、主机鉴定 Master Identification......241
3.6.6.8、身份信息 Identity Information......242
3.6.6.9、身份地址信息 Identity Address Information......243
3.6.6.10、签名信息 Signing Information......244
3.6.6.11、安全请求 Security Request......244
3.6.7、NRF51822 加密硬件模块......245
3.6.7.1、电子密码本 AES Electronic Codebook mode encryption......245
3.6.7.1.1、ECB 程序设计......245
3.6.7.2、AES CCM Mode Encryption (CCM)......247
3.6.7.2.1、AES-CCM 模块工作流程......247
3.6.7.2.2、AES-CCM 模块加密过程......247
3.6.7.2.3、AES-CCM 模块解密过程......248
3.6.7.2.4、CCM 数据结构......249
2.6.7.2.5、AES-CCM 模块要求 RADIO 的配置......250
2.6.7.2.6、加密包在 RADIO 中传输模式......251
2.6.7.2.7、解密包在 RADIO 中接收模式......251
3.6.7.3、快速地址解析模块(Accelerated Address Resolver (AAR))......253
3.6.8、安全管理空中数据计算和分析......254
3.6.8.1 确认值计算......255

3.6.8.2、第 1 次连接加密----配对绑定 STK 和 SK 计算 259
3.6.8.3、第 2 次连接加密----LTK 和 SK 计算..262

版本控制

版本	修改内容	修改时间	修改人员
V1.0	初稿	2015/08/10	刘权
V1.1	添加广播数据结构 2.9.11 节	2017/09/08	刘权
V1.2	1、对属性许可补充 2、修改 ATT 错误应答码	2018/08/20	刘权

版权所有

低功耗蓝牙写笔记的论坛: 【bbs.codertown.cn】
低功耗蓝牙协议研究 QQ 群:【177341833】
低功耗蓝牙协议研究 微信公众号: 【Bluetooth-BLE】

第一章 协议组成图

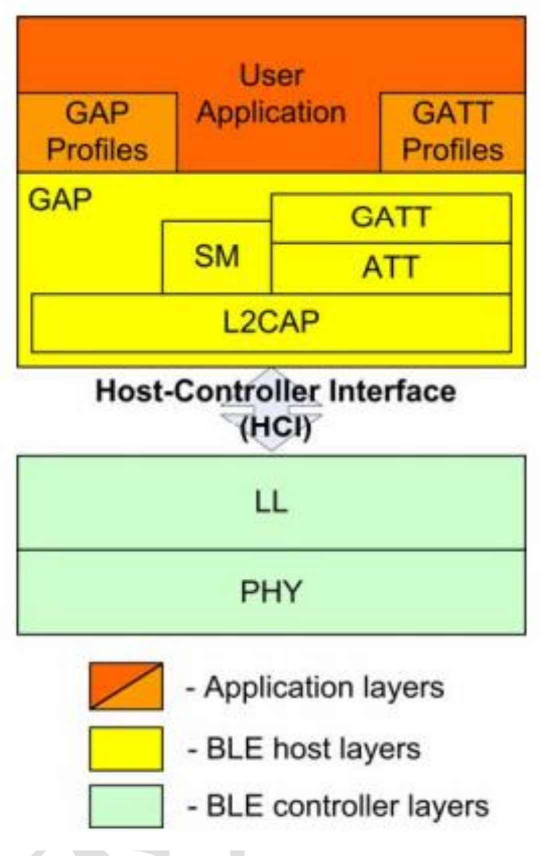


图 1-1 协议组成图

1.1、协议由 HOST 层和 CONTROLLER 层组成

1.1.1、CONTROLLER 组成

PHY：基带物理层

LL：链路层

HCI：主机控制接口

1.1.2、HOST 组成

L2CAP：逻辑链路控制和适配器

ATT：属性协议

GATT：属性协议配置规范

SM：安全管理

GAP：通用访问规范

再往上层就是应用层，不属于协议层。

第二章 控制器

2.1、PHY 物理层

2.1.1、频段

BLE 使用 2.4GHz 工业、科学及医疗(ISM)频段。

这个频段有两个特别之处：

- 它是一个无需授权的频段
- 它是唯一一个在任何国家都通用的频段，即对于 2.4GHz 的频段，从 2400MHz~2483.5MHz 约 83.5MHz 的频谱资源在任何地方都可以使用。

2.1.2、调制

BLE 采用的 GFSK 调制方式(高斯频移键控)，物理层的比特率为 1Mbit/s(1Mbps)。

版权所有

2.1.3、射频信道

一共 40 个通道，37 个自适应自动调频数据通道用于两个连接设备的通讯； 3 个固定广播通道分别是 37、 38、 39。通道的具体频带分布如图 2-1。

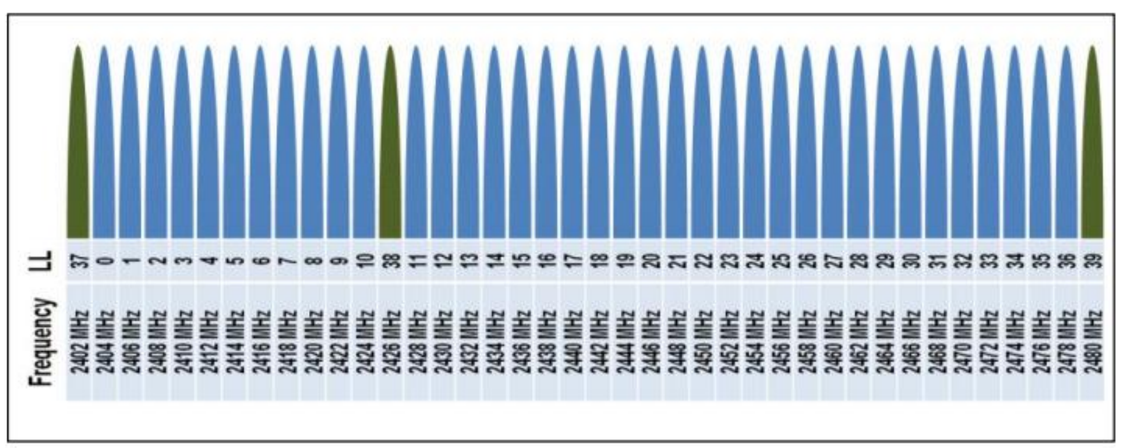


图 2-1 频率分布

每一个频率可以通过以下公式计算：

Regulatory Range	RF Channels
2.400-2.4835 GHz	f=2402+k*2 MHz, k=0, … ,39

表达式中 f 是无线信道 K 的中心频率。也意味着最小频率是 2402MHz，最大频率是 2480MHz。

2.1.4、发射功率

2.4GHz ISM 频段对无需授权的设备有最大发射功率的限制，对于 BLE，在 4.0 规范中有规定：

Minimum Output Power	Maximum Output Power
0.01 mW (-20 dBm)	10 mW (+10 dBm)

也就是最小发送功率不能低于-20dBm 即 10uW，最大的发送功率不能高于+10dBm 即 10mW。

2.2、链路层描述

链路层定义了两个设备如何利用无线电传输信息，包含了报文、广播、数据通道的详细定义，也规定了发现其他设备的流程、广播的数据、连接建立、连接管理以及连接中的数据传输。

2.2.1、链路层的状态机

链路层状态机定义了 5 种状态（图 2-2）：

- 就绪态(Standby)
- 广播态(Advertising)
- 扫描态(Scanning)
- 发起态(Initiating)
- 连接态(Connection)

在扫描状态中又有：主动扫描和被动扫描；连接态又有：主机和从机。

版权所有

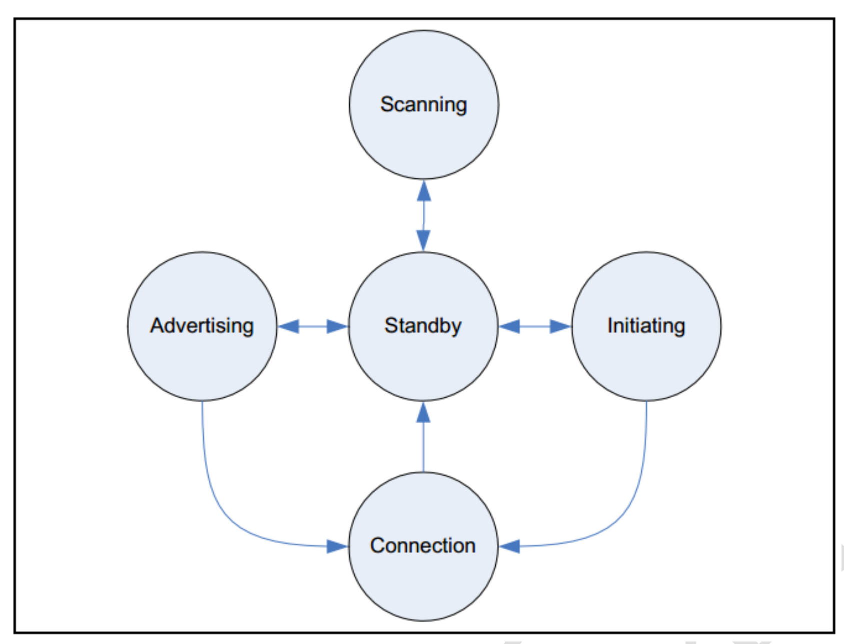


图 2-2 链路层状态机

2.2.2、状态描述

- 就绪态

上电后，链路层进入并保持就绪态，直到接收到主机的命令。就绪态可以进入广播态、扫描态或者发送态。从其他的任何状态都能进入就绪态。

- 广播态

在广播态链路层可以发送广播报文，也可发送扫描响应，用来回应主动扫描的设备。广播状态就是告诉周围的设备，我存在在这里，并在广播报文中发送自己的名字以及设备地址，等待别的设备进行连接。

在这注意，并不是所有的蓝牙一定要有发送和接收机。例如，当广播者仅仅是广播自己而不让其他设备进行连接时，就可以只需要发

送机而不需要接收机。

广播态的设备停止广播可以进入就绪态，也可以收到发起者的连接请求后进入连接态。

- 扫描态

扫描态能接收广播信道的报文，可用于简单的侦听哪些设备正在广播。它有两个子状态：被动扫描和主动扫描。

- 被动扫描

被动扫描态，设备只能被动的扫描，不能发送任何报文，因此，被动扫描可在只有接收机的设备中实现，以控制成本。

- 主动扫描中，链路层一旦发现了新的广播态设备，都会发送扫描请求，并等待该请求的响应。扫描请求和响应报文都是在广播通道中传输。

- 发起态

为了发起连接，链路层需要处于发起态，处于发起态的发起者，其接收机用于侦听自己试图连接的设备。如果收到了来自该设备的广播报文，链路层会向其发送连接请求并进入链接状态，并假设广播者也进入了连接状态，这个假设是什么意思呢？就是发起态发送连接请求后就会进入数据通道，不管和对方有没有真正建立连接，都假设已经建立，在数据通道过程中进行数据包发送，如果能应答说明连接建立，如果发送数据包应答超时，那么连接失败发起态进入就绪态。

- 连接态

连接态可由发起态和广播态进行，进入连接态分为主机和从机，

版权所有

如图 2-3。

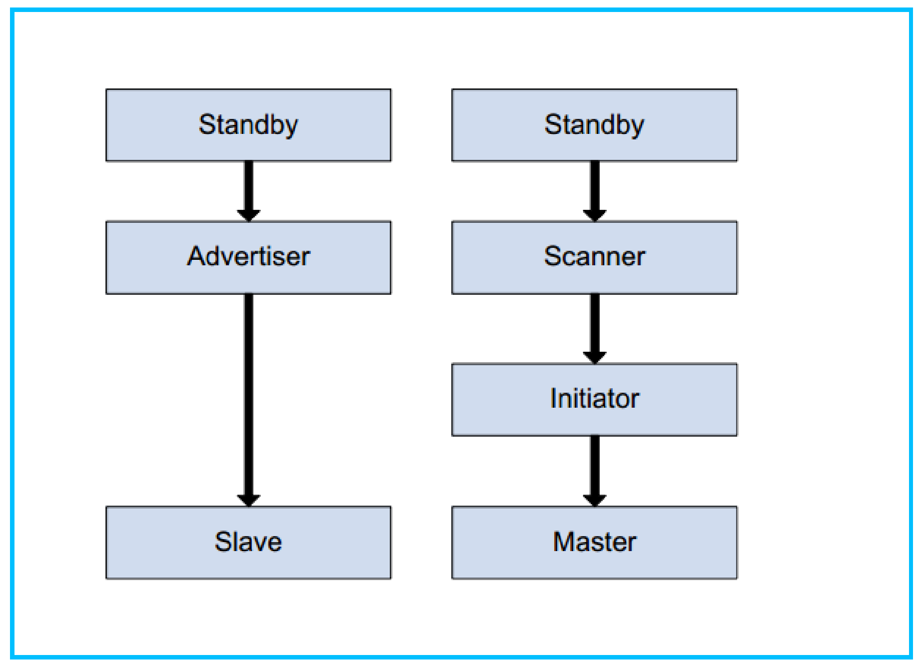


图 2-3 连接态主从机

- 主机只能从发起态进入，必须向对端设备发起连接，并且主设备必须定期向从设备发送报文(即链接事件)，从设备只能回复主机发送的报文时才能发送自己的数据。
- 从机只从广播状态进入，必须向对端设备进行广播，对于从设备是没有主动发送数据的能力，只有当主机发起了连接事件后，从机才能发送自己的数据，如果从机有更多的数据需要发送，那么将继续在同一个连接事件同一个信道上进行报文交互，当然如果从机没有数据需要发送，也可以忽略主机发送的连接事件以达到节能的目的。

2.2.3、bit 数据流格式

在讲数据报文之前，必须了解协议中对数据的 BIT 排序的规定，

在协议中规定数据包或者 PDU 都是以 Little Endian format(小端模式)存放，也就是在内存中低地址放低字节，高地址放高字节，并且 Least Significant Bit (LSB) 即最低 bit 位是第一个发送在空中的位，例如：发送 4 个 bit 的数 $b_0b_1b_2$ b_4= 1101，其中 b_0 为最低 bit 位，所以发送到空中先发送 1，接着发送 1，再发送 0，最后发送 1。也就是说低字节的最低 bit 先发送到空中。

然而并不是整个数据包的每个字节都是以小端模式存放，以小端模式发送数据，对于数据包中的 Cyclic Redundancy Check (CRC) 循环冗余码校验和 Message Integrity Check (MIC)信息完整性检查并不是先发送数据的低字节，而是先发送数据的高字节，例如 CRC 的值为 0x149f5e，那么发送到空中是先发送高字节，即先发送 0x14，在发送 0x9f，最后发送 0x5e。

2.2.4、Preamble 前导码

前导码为一个字节，它有两个值分别为：10101010b 和 01010101b。

实际上这个是干嘛的，具体我不是很清楚，它是由硬件实现，也就是由硬件添加到发送包中，接收到包解析也是由硬件检测，4.0 协议有说是频率同步和增益控制用，在协议中有这么一段话：

> All Link Layer packets have an eight bit preamble. The preamble is used in the receiver to perform frequency synchronization, symbol timing estimation, and Automatic Gain Control (AGC) training.

那么这前导码的两个值怎么使用呢？是根据 Access Address(接入

地址)的最低位确定的， 当接入地址的最低位为 0 时，那么前导码为 0xAA，即 10101010b，如果是 1 时，前导码为 01010101b，目的是保证报文的前 9 个 bit 都是交替位。实际上在 nrf51822 中，这些都是由硬件完成的。

2.2.4、Access Address 接入地址

接入地址是紧跟前导码的 32 个 bit，它分为两种类型：

- 广播接入地址
- 数据接入地址

广播接入地址是在广播数据，或是广播、扫描、发起连接时使用。数据接入地址是在连接建立之后两个设备的使用。

当控制器试图接收一个报文时，它要事先知道待接收报文的接入地址。接收机开启并调谐到正确的频率，就可以收到数据。即使附近没有发生数据的设备，接收机还是会收到背景辐射。考虑接收到纯随机噪声的概率，接收一段噪声与前导序列相符的可能性相对还是比较高的。通常，如果低功耗蓝牙接收机一直开着，每隔几分钟就能收到一个假的前导序列。因此，就需要利用接入地址以减少随机噪声造成伪报文接收的概率。

链路层也不知道其他设备什么时候发生报文，因此只能保留 40us 接收到的 bit，并在新的 bit 移入到寄存器的时候检查序列是否满足前导和接入地址，这一过程称为与接入地址求相关。(上面应该是硬件需要考虑的地方)。

版权所有

接入地址怎么确定呢？

● 广播接入地址来源

对于广播通道接入地址是一个固定值 0x8e89bed6。发送到空中时的二进制如下(从左到右)

LSB							MSB
0110	1011	0111	1101	1001	0001	0111	0001
6	d	e	b	9	8	e	8

从上面可了解到意味着广播报文的前导码是 01010101(低位在前)。之所以选择这个值作为接入地址，是因为它的相关性非常好。

● 数据接入地址来源

对于数据通道，接入地址是一个随机数，同一对设备不同连接访问地址也是不同的，当然这个随机值必须满足一定的条件：

- 没有超过连续的 6 个 1 或者 0；
- 不是广播通道的接入地址；
- 和广播通道接入地址至少有 1 个 bit 不同；
- 4 个字节必须都不相等；
- 不能超过 24bit 的翻转，即不能 010101010101010101010101；
- 地址的最高 6bits 至少有两次的翻转。

数据接入地址是有主机产生，通过连接请求发给从机，进而进入连接状态。

版权所有

2.2.6、Cyclic Redundancy Check (CRC) 循环冗余码校验

在报文的最后 3 个字节是 CRC 校验码。CRC 对整个 PDU 进行计算，不包括前导码和接入地址。

BLE 共有 24 位 CRC，它的生成项是：

$$CRC=x^{24}+x^{10}+x^{9}+x^{6}+x^{4}+x^{3}+x^{1}+x^{0}$$

在 nrf51822 中是采用硬件实现这个生成项，如图 2-4 线性反馈移位寄存器。

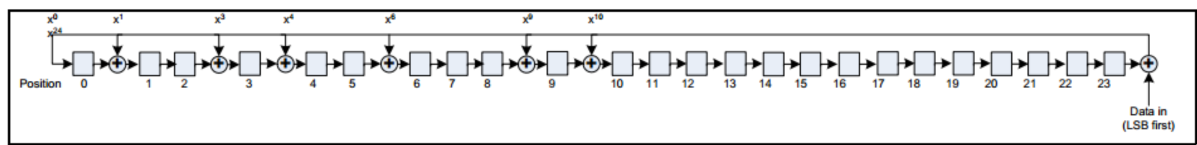


图 2-4 The LFSR circuit generating the CRC

而 CRC 的移位寄存器一定需要初值的，对于广播通道的 CRC 移位寄存器的初始值为 0x55555。而对于数据通道下的 CRC 初始值则是由主机发送连接请求时将 CRC 移位寄存器的初始值发送给从机，之后就一直使用这个值作为连接状态下的 CRC 移位寄存器的初始值。

2.2.7、DATA WHITENING 数据白化

数据白话是为了避免长序列的 0 或者 1，例如：00000000b 或者 11111111b，这是因为频移键控(FSK)接收机本身接收连续相同 bit 能力差的原因，比如，当接收一串“000000000000”的 bit 序列时，接收机会认为发射机的中心频率向左偏移，进而导致频率失锁。之后的 bit“1”会被错过，报文的接收也因此会失败，为了避免这一情况，BLE 规范中使用了“白化器”来随机化发送数据。

白化器是一个很短的输出“0”、“1”序列的随机发生器，目的是使发送的数据随机化。然而白化随机序列通过什么产生呢？类似于CRC，白化随机序列也通过生产项产生：

$$Whitener=x^7+x^4+x^0$$

它同样采用如图 2-5 所示的线性反馈移位寄存器实现。

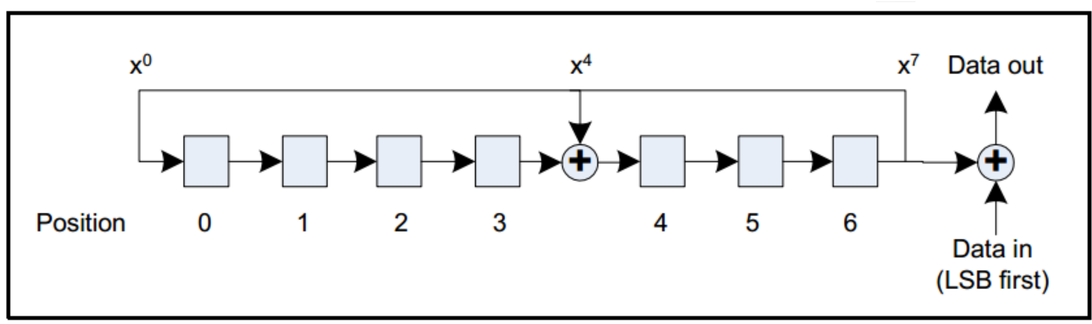


图 2-5 The LFSR circuit to generate data whitening

这里注意白化是不分广播状态或者是连接状态，它的移位寄存器的初始值为信道通道号，并且最高位一定置 1，也就是初始值的第 7bit 位一定是 1。例如，广播通道 channel index=38(0x26)，那么初始值就是 38。那么对应于图 2-5 中的 Position，是怎么放这个值得呢？

位置 0　Position 0 = 1 -----> bit6　//这里是硬件置 1

位置 1　Position 1 = 1 -----> bit5

位置 2　Position 2 = 1 -----> bit4

位置 3　Position 3 = 1 -----> bit3

位置 4　Position 4 = 0 -----> bit2

位置 5　Position 5 = 0 -----> bit1

位置 6　Position 6 = 0 -----> bit0

注意这里的位置和 bit 位的关系是反的，也就是移位寄存器按上图

的话，初始值的低位在右边，高位在左边。

下面在引用 4.0 规范中的图片图 2-6 Payload bit processes。

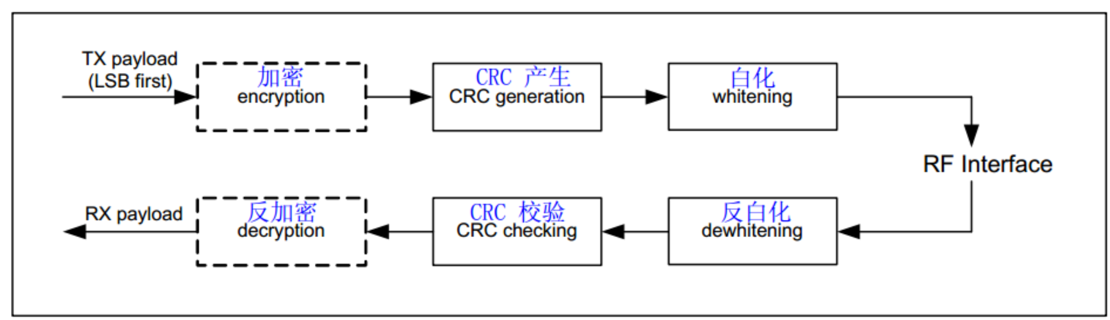
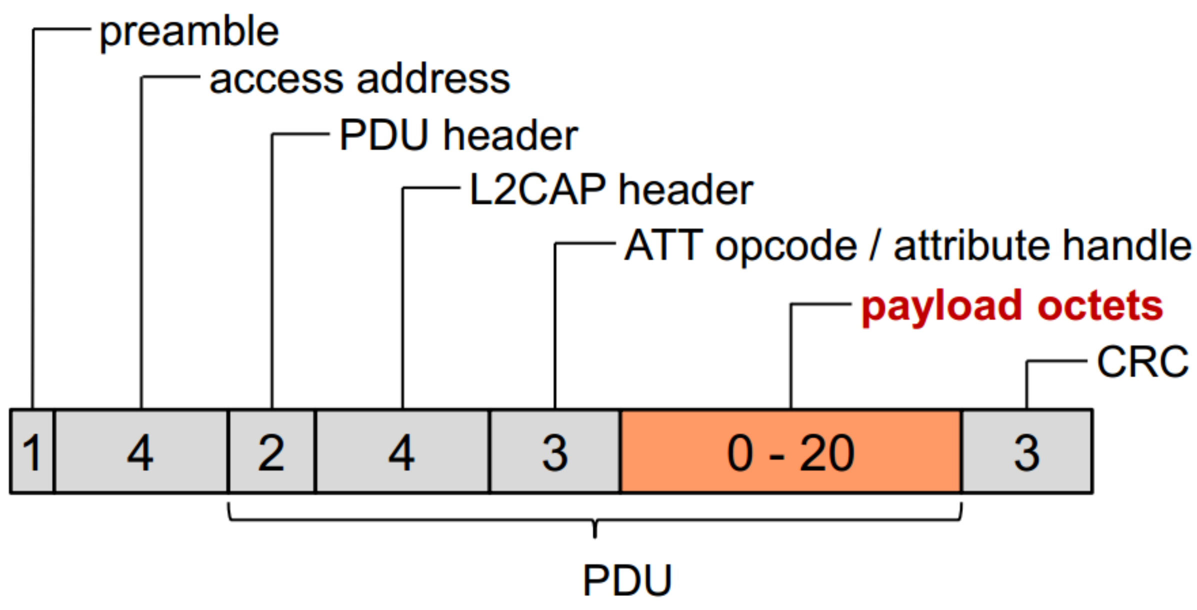


图 2-6 Payload bit processes

从上图可以知道，对于发送设备，白化是在 CRC 产生之后，发送之前的最后一道处理；对于接收设备，白化是在接收到数据的第一处理，进而进行 CRC 校验。

2.3、链路层报文结构

报文是协议的核心，而链路层的报文则是整个报文的基石，下面先给出协议所有报文的结构图 2-7。



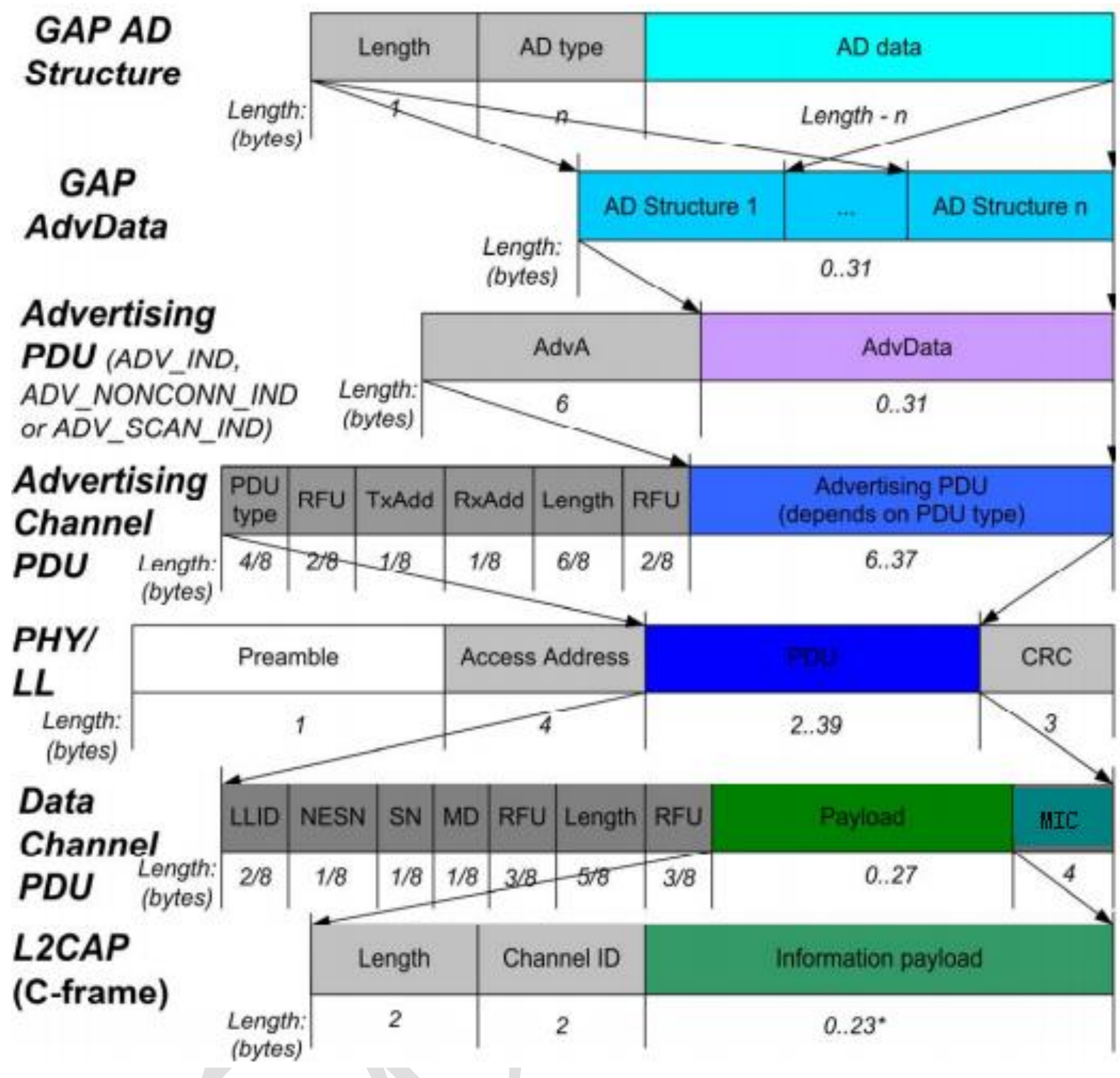


图 2-7 BLE 帧格式

由上图可以看到所有的数据都是指向 PHY/LL 数据帧结构中的 PDU(Protocol Data Unit)即协议数据单元。链路层的包格式如图 2-8。

LSB　　　　　　　　　　　　　　　　　　　　　　　　　　　　MSB

Preamble (1 octet)	Access Address (4 octets)	PDU (2 to 39 octets)	CRC (3 octets)

图 2-8 链路层包格式

2.3.1、PDU 的报头和长度

对于图 2-8 中的 PDU，又可以分为三部分，报头、长度和数据净

荷，报头和长度各占一个字节。如表 2-1。

表 2-1 报文结构

8	32	8	8	0~296 bit	24
前导	接入地址	报头	长度	数据净荷	CRC

2.3.2、广播通道的 PDU 格式

广播通道下的数据 PDU 的组成如表 2-2。

表 2-2 广播报文报头和长度

8	32	8	8	0~296 bit	24
前导	接入地址	报头	长度	数据净荷	CRC

LSB 4	2	1	1	6bit	MSB 2
报文类型	RFU 保留	发送地址	接收地址	净荷长度	RFU 保留

版权所有

- 报文类型：是最低 4bit，共有 7 种广播报文类型，每种都有不同的净荷格式以及行为：
  - ADV_IND——通用广播，我的程序上电就进行通用广播
  - ADV_DIRECT_IND——定向连接广播
  - ADV_NONCONN_IND——不可连接广播
  - ADV_SCAN_IND——可扫描广播
  - SCAN_REQ——主动扫描请求
  - SCAN_RSP——主动扫描应答
  - CONNECT_REQ——连接请求
- 发送地址(TXADD)和接收地址(RXADD)：两个都是说明设备地址的类型的，当某位为“1”时表示 Random Add(随机地址)，当为“0”时表示 Public Add(公共地址)。然而这个标记指定放在哪的地址是随机地址还是公共地址呢？在报文类型命令中对应的净荷带有 6bytes 地址，这些地址就紧跟在长度的后面，这些标志位就是标示这些地址类型的。如图 2-9 和图 2-10。

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

图 2-9 ADV_IND PDU Payload

版权所有

```
⊟ Bluetooth Low Energy
    Access Address: 0x8e89bed6
  ⊟ Packet Header
      .1.. .... = TX Address: random
      .... 0000 = TYPE: ADV_IND (0x00)
      Length: 25
    Advertising Address: f8:54:3c:59:b2:81 (f8:54:3c:59:b2:81)
  ⊞ Advertising Data: 0f094e6f726469635f4c515f55415254020105
    CRC: 0xfbfeea

0000  07 06 2c 01 38 0e 06 0a  01 27 49 00 00 90 02 00   ..,.8... .'I.....
0010  00 d6 be 89 8e 40 19 81  b2 59 3c 54 f8 0f 09 4e   .....@.. .Y<T...N
0020  6f 72 64 69 63 5f 4c 51  5f 55 41 52 54 02 01 05   ordic_LQ _UART...
0030  fb fe ea                                           ...
```

图 2-10 *sniffer 采样到的 ADV_IND 报文*

实际上广播通道下每一个报文类型的净荷都包含有一个 6 字节的地址。同时在图 2-10 中用红色框住的内容，可知 Advertising Address=0xf8543c59b281，在发送时是低字节先发送。但是 CRC=0xfbfeea，而在发送时先发送的是高字节，最后发送的是低字节。

- 净荷长度：这个长度是指在 PDU 中的数据除去报头和长度之外的有效净荷数据的长度，注意的是这里只用到了 6bit 的长度，也就是说最大只能有 64 个字节长度，够了吗？实际上在表 2-2 中净荷数据上面就标明了净荷的长度最大 296bits=37Bytes，也就是说在整个数据包中，在链路层真正的有效数据最大是 37 个字节，所以需要用也只需要 6bits 长度。再看图 2-10 可知实际的有效数据变为了最多 37-6=31 字节。
- RFU：它的意思是保留将来用(RESERVED FOR FUTURE USE)。

注意：在上面数据中依然是采用小端模式和低 bit 在左边，高 bit 在右边。也就是在报头中，报文类型的 4 个 bit 是在这个字节的低 4bit。

版权所有

2.3.3、数据通道的 PDU 格式

数据通道下的数据 PDU 的组成如表 2-3。

表 2-3 广播报文报头和长度

8	32	8	8	0~216 bit	32	24
前导	接入地址	报头	长度	数据净荷	MIC	CRC

LSB 2	1	1	1	3	5bit	3MSB
LLID	NESN	SN	MD	DFU	Length	DFU
链路标识符	期望下一报文序列号	报文序列	更多数据	保留	净荷长度	保留

直接引用 4.0 规范中的图片，解释各字段的意思，如图 2-11。

版权所有

Field name	Description
LLID	The LLID indicates whether the packet is an LL Data PDU or an LL Control PDU. 00b = Reserved 01b = LL Data PDU: Continuation fragment of an L2CAP message, or an Empty PDU. 10b = LL Data PDU: Start of an L2CAP message or a complete L2CAP message with no fragmentation. 11b = LL Control PDU
NESN	Next Expected Sequence Number
SN	Sequence Number
MD	More Data
Length	The Length field indicates the size, in octets, of the Payload and MIC, if included.

图 2-11 Data channel PDU Header field

在数据报文中有 Message Integrity Check (MIC)信息完整性检测。为了保证数据的正确性，它占有 32bit 即 4 个字节，这涉及到加密操作(见 3.6.3.3.4 节)，它是用虚线表示的，也就是不一定要有，因为并不是一定要加密才能工作，例如 MIC 在某些情况是**不需要 MIC** 的：

- 没有加密的链路层连接
- 有加密的链路连接，但是发送的是空包，也就是有效 PDU 是为空的包。

注意：**这里的有效净荷数据长度最大只有 216bits 了，也就是 27 个字节，加上 MIC 的 4 个字节，也才 31 字节，那么还有 6 字节去哪了？实际上协议规范就是这么规定的，在广播通道这 6 个字节是设备地址，在数据通道规定最大的净荷包不能超过 27 个字节。不管有没有 MIC 净荷只能是最多 27 字节**。

2.4、链路层设备滤波

这个全部是由硬件完成的，我也搞的不是很懂，这个滤波是指根据设备地址进行滤波。简单来说就是通过设备地址来确定这个数据是不是所需要的数据，而对于 nrf51822 来说，就是在连接请求的时候将对方的设备地址写到 **White List** 白名单中就行了，其余都是硬件搞定。避免我歪曲理解，将 4.0 规范中的原文贴到下面。在 4.0 协议的第 6 章 PART B 部分的 4.3 节。

4.3 LINK LAYER DEVICE FILTERING

The Link Layer may perform device filtering based on the device address of the peer device. Link Layer Device Filtering is used by the Link Layer to minimize the number of devices to which it responds.
A Link Layer shall support Link Layer Device Filtering unless it only supports non-connectable advertising.
The filter policies for the Advertising State, Scanning State and Initiating State are independent of each other. When the Link Layer is in the Advertising State, the advertising filter policy shall be used. When the Link Layer is in the Scanning State, the scanning filter policy shall be used. When the Link Layer is in the Initiating State, the initiator filter policy shall be used. If the Link Layer does not support the Advertising State, Scanning State, or Initiating State, the corresponding filter policy is not required to be supported.

4.3.1 White List

The set of devices that the Link Layer uses for device filtering is called the White List.
A White List contains a set of White List Records used for Link Layer Device Filtering. A White List Record contains both the device address and the device address type (public or random). All Link Layers supporting Link Layer Device Filtering shall support a White List capable of storing at least one White List Record.
On reset, the White List shall be empty.
The White List is configured by the Host and is used by the Link Layer to filter advertisers, scanners or initiators. This allows the Host to configure the Link Layer to act on a request without awakening the Host.
All the device filter policies shall use the same White List.

2.5、NRF51822 的 Radio

上面其实有许多问题，例如 CRC 和白化移位寄存器的初始值存放在哪？怎么用呢？设备地址的白名单是什么意思？对于 nrf51822 的 Radio 许多东西都通过硬件实现。如图 2-12 Radio 的整体框图。

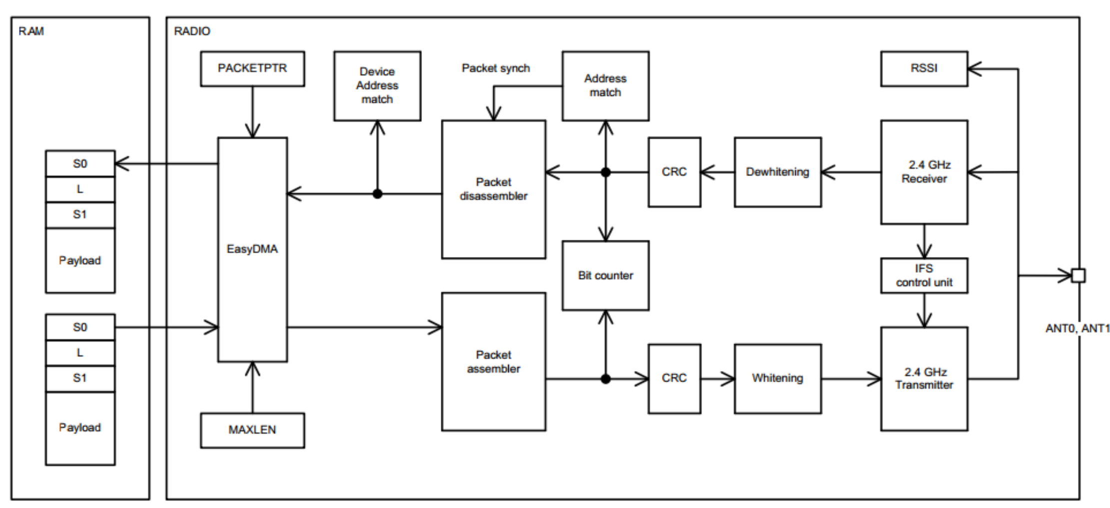


图 2-12 nrf51822 的 Radio 框图

其实从图 2-12 中可以看到，在 RAM 部分的就是图 2-8 中的 PDU，这就意味着，除了 PDU 这部分可以通过软件进行操作外，其余的前导码、接入地址、CRC 及数据白化都是通过硬件完成的。

2.5.1、EasyDMA

在 nrf51822 中的 radio 包含有一个简单的 DMA，DMA 的作用就是从 RAM 中的指定地址读写数据，而这个地址是有个寄存器的去存放的，实际上它还完成了一个事情，例如，每次取地址的上多少个数据呢？在软件中只是负责将数据准备好，然后将指针给寄存器，到底发送多少个数据也是硬件完成的。

2.5.2、包结构

NORDIC 公司在设计 radio 时，肯定是根据协议规范走的，所以它发送的包一定包含有前导码、访问地址、有效数据及 CRC 的部分。如图 2-13。

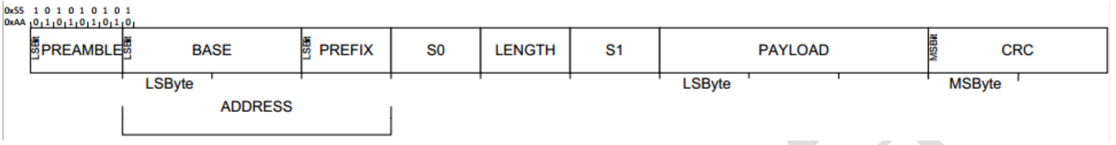


图 2-13 On-air packet layout

图中的前导码以及 CRC 就不讲了，都是硬件完成的。特别是前导码，软件根本就不用管它。

- 接入地址：如图 2-14，在 NORDIC 将接入地址分为了两部分，一个是地址基本部分和地址前缀部分。总共可以组成 8 个接入地址如图 2-15，通过另外两个寄存器 TXADDRESS 和 RXADDRESSES 分别指定哪个作为接入地址，选定了之后，在发送数据时，硬件会自动添加相应的接入地址到包中。还有一个只读寄存器 RXMATCH，当接收到数据时，这个寄存器可以读到是哪个接入地址匹配上。

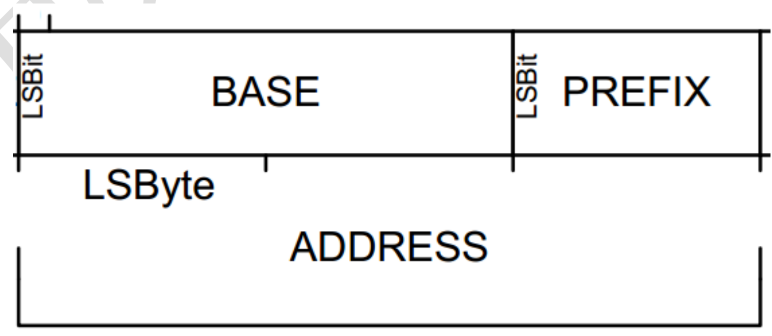


图 2-14 接入地址包

版权所有

Logical address	Base address	Prefix byte
0	BASE0	PREFIX0.AP0
1	BASE1	PREFIX0.AP1
2	BASE1	PREFIX0.AP2
3	BASE1	PREFIX0.AP3
4	BASE1	PREFIX1.AP4
5	BASE1	PREFIX1.AP5
6	BASE1	PREFIX1.AP6
7	BASE1	PREFIX1.AP7

图 2-15 接入地址组合

从图 2-15 中可以看出，共有 4 个寄存器组成这 8 个接入地址。图 2-16 为其中的两个寄存器。

BASE0

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID (Field ID)	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ID	RW	Field	Value ID	Value	Description
A	RW				Radio base address 0. Decision point: START task.

PREFIX0

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID (Field ID)	D	D	D	D	D	D	D	D	C	C	C	C	C	C	C	C	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ID	RW	Field	Value ID	Value	Description
A	RW	AP0			Address prefix 0. Decision point: START task.
B	RW	AP1		..	
C	RW	AP2		..	
D	RW	AP3		..	

图 2-16 BASE0 和 PREFIX0 寄存器

从图中可以知道，由 BASE0 和 PREFIX0 寄存器可以组成 5 字节的地址，但是在 ble 中，只需要 BASE0 的 3 个字节和 PREFIX0 的一个字节组成地址。然而这里注意，BASE0 的使用个数是在另个配置寄存器

中进行设置的，BLE 当然是设置成 3。只是还有一个有趣的问题是 BASE0 的 3 个字节是怎么放的，例如广播通道的访问地址是 0x8e89bed6，那么 BASE0=0x89bed6 和 PREFIX0=0x8e 就行了吗？实际上 nrf51822 的设计很有意思，BASE0 是从低字节开始截掉的，也就是说一定要设置 BASE0=0x89bed600。在我的程序中是如下设置：

```
Static   uint8_t   p_packetHeaderLFlen = 6;
static   uint8_t   p_packetHeaderS0len = 1;
static   uint8_t   p_packetHeaderS1len = 2;
static   uint8_t   p_static_length          = 0;
static   uint32_t   p_balen                 = 3;
static   uint32_t   Prefix0_add = 0x0000008e;      /**<  接入地址前缀 0 Address. */
static   uint32_t   Base0_add = 0x89bed600;         /**< Address. */
…
…
//  因为下面设置的基地址长度为 3，所以低字节被截除即取 Base0_add 的高 3 个字节
NRF_RADIO->BASE0    = Base0_add ;

//  逻辑地址  3  到  0  的前缀地址  最低字节为前缀地址 0
NRF_RADIO->PREFIX0    =
                            ( Prefix0_add & 0xff000000) |
                            ( Prefix0_add & 0x00ff0000) |
                            ( Prefix0_add & 0x0000ff00) |
                            ( Prefix0_add & 0x000000ff);

//  配置包  0  的设置
NRF_RADIO->PCNF0 =(p_packetHeaderS1len << RADIO_PCNF0_S1LEN_Pos) |    //0    s1 没有
                  (p_packetHeaderS0len << RADIO_PCNF0_S0LEN_Pos) |    //1 S0 为 1 字节
                  (p_packetHeaderLFlen<< RADIO_PCNF0_LFLEN_Pos);      //8bit 1 字节

//  配置包  1  的设置:使能白化,小端模式
NRF_RADIO->PCNF1 =(RADIO_PCNF1_WHITEEN_Enabled<<RADIO_PCNF1_WHITEEN_Pos)|
                  (RADIO_PCNF1_ENDIAN_Little<< RADIO_PCNF1_ENDIAN_Pos) |
                  (p_balen << RADIO_PCNF1_BALEN_Pos)      |      //基地址长度为 3
          (p_static_length<<RADIO_PCNF1_STATLEN_Pos) |//  这个值为 0  什么意思还不清楚
          (PHY_PAYLOAD_MAX_SIZE << RADIO_PCNF1_MAXLEN_Pos);       //净量长度 37 字节
……
```

- RAM 中的净荷包格式：前面提到，在 RAM 中的数据就是图 2-8 中的 PDU。在 2.3 节中可以知道，PDU 中的前 2 个字节其

实是固定了的，第 1 个字节在广播和数据通道中有各自的包头，第 2 个字节都是长度。如图 2-17 所示，NORDIC 也将这个两个字节进行定义，分别是可以配置的 S0、LENGTH 和 S1。

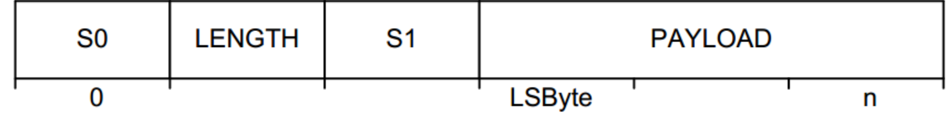


图 2-17 RAM 中的 PDU 包格式

然而因为 nrf51822 还可以作为通用的 2.4GHz 的无线收发器，才设计出可以配置的 S0、LENGTH 和 S1 区域。试图猜测，S0 必定对应于 4.0 协议规范中的报头，LENGTH 则为长度，那么 S1 是什么东西呢？看图 2-18 可知。S0 是由 S0LEN 配置的，如果是 1 就包含这个区域，这是以 Byte 为单位，LENGTH 和 S1 区域共同组成一个字节，所以在配置时是以 bit 为单位的，在广播通道下 LENGTH 区域为 6，S1 区域为 2，也就是配置 LFLEN=6，S1LEN=2，数据通道下 LENGTH 区域为 5，S1 区域为 3，也就是配置 LFLEN=5，S1LEN=3。在 NRF51822 的使用手册中有如下一段话：

> The sizes of the S0, LENGTH, and S1 fields can be individually configured through S0S, LS, and S1S in PCNF0 respectively. If any of these fields are configured to be less than 8 bits long, the least significant bits of the fields, as seen from the RAM representation, are used. 用字节低位
> If S0, LENGTH, or S1 are specified with zero length, their fields will be omitted in memory. Otherwise, each field is represented as a separate byte, regardless of the number of bits in their on-air counterpart. 意思是只要 S0 或者 S1 大于 0，那么在 RAM 中就占 1 字节

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID (Field ID)	-	-	-	-	-	-	-	-	-	-	-	-	C	C	C	C	-	-	-	-	-	-	-	B	-	-	-	-	A	A	A	A
Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ID	RW	Field	Value ID	Value	Description
A	RW	LFLEN		[0..8]	Length of length field in number of bits. Decision point: START task.
B	RW	S0LEN		[0..1]	Length of S0 field in number of bytes. Decision point: START task.
C	RW	S1LEN		[0..8]	Length of S1 field in number of bits. Decision point: START task.

图 2-18 Radio 的 PCNF0 配置寄存器

2.5.3、设备地址--白名单

这里要区分接入地址和设备地址，接入地址是一个 4 字节的随机数，而设备地址是由 Bluetooth SIG 分配的一个 6 字节的设备地址，是全球唯一的地址。这个设备地址在广播通道下的每一个报文都会被发送和接收，进入连接状态后，这个地址就会被放到白名单中存放着。对于 nrf51822 来说，有两个寄存器存放这 6 个字节，也就是在 4.0 规范中提到的白名单列表，至少要有一个能存放白名单的地方，对于 nrf51822 芯片中有 8 对寄存器可以存放 8 个设备地址，如图 2-19。DAP 为地址高 2 字节，DAB 为地址低 4 字节。

DAB[n] 设备基地址部分 0-7

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Id	RW	Field	Value Id	Value	Description
A	RW	DAB			Device address base segment n Device address base segment

DAP[n] 设备前缀地址部分 0-7

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id																	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Id	RW	Field	Value Id	Value	Description
A	RW	DAP			Device address prefix n Device address prefix

图 2-19 设备地址寄存器

在 nrf51822 的指导手册中有这么一句话：

> The RADIO is able to listen for 8 different device addresses at the same time.

那么能同时监听 8 个地址不一定一定就要监听 8 个地址，所以还有一个寄存器去配置哪个地址将使用。如图 2-20DACNF 寄存器，这个寄存器用两个字节来完成设置，第 1 个字节主要设置接收数据时，接收那个地址发送的数据，第 2 个字节是设置发送使用那个设备地址。

DACNF 设备地址匹配配置寄存器

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id																	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Id	RW	Field	Value Id	Value	Description
A	RW	ENA0 使能地址匹配设备地址0否？			Enable or disable device address matching using device address 0
			Disabled	0	Disabled
			Enabled	1	Enabled
B	RW	ENA1			Enable or disable device address matching using device address 1
			Disabled	0	Disabled
			Enabled	1	Enabled
C	RW	ENA2			Enable or disable device address matching using device address 2
			Disabled	0	Disabled
			Enabled	1	Enabled
D	RW	ENA3			Enable or disable device address matching using device address 3
			Disabled	0	Disabled
			Enabled	1	Enabled
E	RW	ENA4			Enable or disable device address matching using device address 4
			Disabled	0	Disabled
			Enabled	1	Enabled
F	RW	ENA5			Enable or disable device address matching using device address 5
			Disabled	0	Disabled
			Enabled	1	Enabled
G	RW	ENA6			Enable or disable device address matching using device address 6
			Disabled	0	Disabled
			Enabled	1	Enabled
H	RW	ENA7			Enable or disable device address matching using device address 7
			Disabled	0	Disabled
			Enabled	1	Enabled
I	RW	TXADD0 设备地址0的发送地址			TxAdd for device address 0
J	RW	TXADD1			TxAdd for device address 1
K	RW	TXADD2			TxAdd for device address 2
L	RW	TXADD3			TxAdd for device address 3
M	RW	TXADD4			TxAdd for device address 4
N	RW	TXADD5			TxAdd for device address 5
O	RW	TXADD6			TxAdd for device address 6
P	RW	TXADD7			TxAdd for device address 7

图 2-20 设备地址选择寄存器

2.5.4、Radio 的状态机和时间参数

图 2-21 为 Radio 的状态机

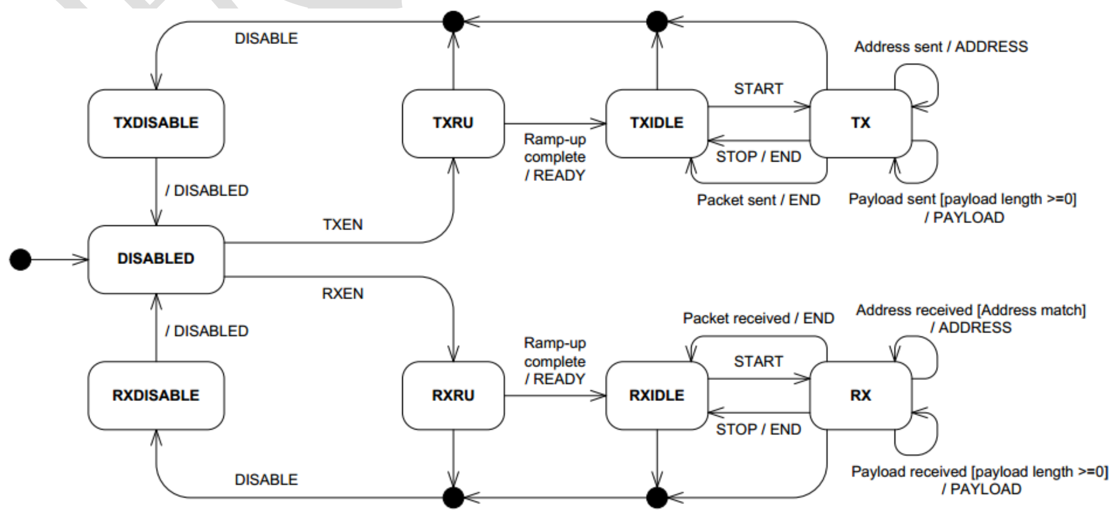


图 2-21 Radio 状态机

对于 Radio 的配置有几个过程，以发送为例。在发送前，先要将 Radio 转为发送模式，而转换的条件是 TXEN 任务触发，当转换成功后产生 REDAY 事件，这个事件产生说明可以开始传输数据，也就是已经具备了发送数据的能力，而真正的发送数据是触发 START 任务，也就是将数据往外面送，接着就是等待发送完毕事件产生，进而结束发送事件。

在软件中，用到 Radio 的主要几个状态：

- DISABLE：Radio 不使能状态
- RXIDLE ：Radio 使能 RXEN 从 DISABLE 转为准备 RX 状态
- TXIDLE ：Radio 使能 TXEN 从 DISABLE 转为准备 TX 状态
- RX：Radio 从 RXIDLE 状态进入 RX 状态
- TX：Radio 从 TXIDLE 状态进入 TX 状态
- Radio 接收到数据状态，等待接收完毕事件标志

为什么会有这么对状态，主要是 4.0 规范中规定，在同一个频道上的两个连续包必须 150±2 µs 完成，就是说主机发送一个包给从机，从机必须在 150±2 µs 给出应答给主机，否则数据接收不到。这里就不得不给出 nrf51822Radio 的时间参数，如图 2-22。

Symbol	Description	250 k	1 M	2 M	BLE	Jitter	Units
t_{TXEN}	Time between TXEN task and READY event	132	132	132	140	0	µs
$t_{TXDISABLE}$	Time between DISABLE task and DISABLED event when the radio was in TX	10	4	3	4	1	µs
t_{RXEN}	Time between the RXEN task and READY event	130	130	130	138	0	µs
$t_{RXDISABLE}$	Time between DISABLE task and DISABLED event when the radio was in RX	0	0	0	0	1	µs
$t_{TXCHAIN}$	TX chain delay	5	1	0.5	1	0	µs
$t_{RXCHAIN}$	RX chain delay	12	2	2.5	3	0	µs

图 2-22 Radio timing parameters

Radio 从不使能状态启动发送使能任务(TXEN)到使能转换完成进入准备接收状态的时间需要 140us，同样接收使能任务(RXEN)到使能转换完成进入准备接收状态的时间需要 138us。这是什么概念，如果按照这样的时间，我接收到数据处理数据和准备发送数据的时间仅仅 10 多 us。所以在软件中 Radio 控制采用如流程图 2-23 方式进行程序设计。

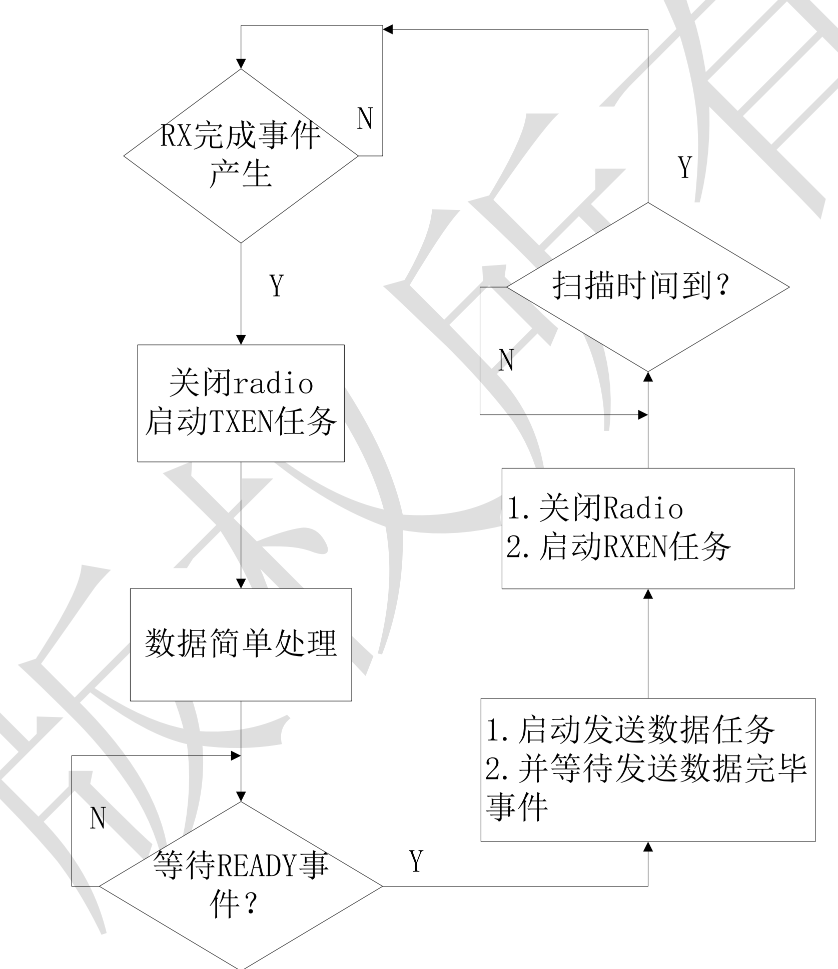


图 2-23 RADIO 收发转换流程图

在源码中的主要 Radio 的操作函数有：

```
/**函数名：Radio_Disable
*  作用  ：关闭 radio 同时复位某些寄存器
*/
void Radio_Disable(void)
```

```
-----------------------------------------------------

/**函数名：Radio_Configure_No_Change
* 作用 ：Radio 的无需改变的配置函数
*/
void Radio_Configure_No_Change(void)
-----------------------------------------------------

/**函数名：Radio_Configure
* 作用 ：Radio 的在广播或者连接状态下对 Radio 重新配置
*入口参数 ：Connect_Parameter，连接参数
*/
void Radio_Configure(Connect_Radio_Parameter * Connect_Parameter)
-----------------------------------------------------

/**函数名：Radio_Start_Task
* 作用 ：Radio 启动开始发送或者接收任务
*/
static void Radio_Start_Task(void)
-----------------------------------------------------

/**函数名：Radio_Wait_Ready_Event
* 作用 ：等待 Radio 的 TX 或者 RX 模式转换完毕
*/
static void Radio_Wait_Ready_Event(void)
-----------------------------------------------------

/**函数名：Radio_Wait_End_Event
* 作用 ：等待 Radio 的发送完毕或者接收完毕
*/
static void Radio_Wait_End_Event(void)
-----------------------------------------------------

/**函数名：Radio_RX_Mode_To_Ready
* 作用 ：Radio 启动 RXEN 的任务并等待 Ready 事件
*/
void Radio_RX_Mode_To_Ready(void)
-----------------------------------------------------

/**函数名：Radio_TX_Mode_Prepare
* 作用 ：Radio 启动 TXEN 的任务，仅仅启动任务
*/
void Radio_TX_Mode_Prepare(void)
-----------------------------------------------------
```

```
/**函数名：Radio_TX_Wait_Ready_To_Start
* 作用 ：Radio 等待 Ready 事件并启动开始发送
*/
static void Radio_TX_Wait_Ready_To_Start(void)
---------------------------------------------------

/**函数名：Radio_TX
* 作用 ：Radio 发送函数
*/
void Radio_TX(void)
---------------------------------------------------

/**函数名：Radio_RX
* 作用 ：Radio 接收函数
*入口参数：Status 是广播通道还是数据通道
*/
void Radio_RX(uint8_t Status)
```

流程图 2-23 的主要思想是：当接收数据事件(**Radio_Wait_End_Event()**)产生后立刻关闭 Radio(**调用 Radio_Disable()**)并进行 TXEN 任务(**调用 Radio_TX_Mode_Prepare()**)，然后 CPU 继续处理数据，当数据处理完后，再进行数据发送(**调用 Radio_TX()，这个函数中调用 Radio_Wait_Ready_Event() 和 Radio_TX_Wait_Ready_To_Start()**)，数据发送完毕关闭 Radio(**调用 Radio_Disable()**)，接着进行 RXEN 的任务(**调用 Radio_RX_Mode_To_Ready()**)等待下一次的数据接收。这样就充分利用了 Radio 的转换时间，然而又怎么保证刚刚好是 150±2 μs 呢？这不得不讲下面一个问题。

- T_IFS：The Inter Frame Space 帧间隔时间。在 4.0 协议中规定为 150us。这个值就是两个连续包之间的时间间隔。在 nrf51822 中是通过硬件来确定这个值的，在图 2-12 中的右边有“IFS control unit”，对于软件是有一个寄存器设置，单位为 us。也就是设置了这个寄存器，就不必担心在 150us 之前就

进行了数据的收发。

2.5.5、Radio 的基本配置

因为只有 PDU 是完全由软件支配，那么 Radio 必定是有对 CRC、白化初始值、频率、发射功率以及信号强度等参数的设置寄存器。这里不在一一讲解寄存器。对于 Radio 的配置函数，如图 2-24。这里仅仅是部分配置，在广播状态和连接状态都需要重新设置 Radio。

```
void Radio_Configure_No_Change(void)
{
    Radio_Disable();  // 关闭radio

     NRF_RADIO->TIFS=150;
  //无线速率: 00: 1Mbit, 01: 2Mbit, 02: 250Kbit, 03: 1Mbit（蓝牙） RADIO_MODE_MODE_Nrf_1Mbit
     NRF_RADIO->MODE       = ( RADIO_MODE_MODE_Ble_1Mbit<< RADIO_MODE_MODE_Pos);

     NRF_RADIO->PACKETPTR = (uint32_t)Radio_Packetptr.RTX_Radio_Packetptr;

     NRF_RADIO->TXPOWER   = (RADIO_TXPOWER_TXPOWER_0dBm << RADIO_TXPOWER_TXPOWER_Pos);  //无线功率 0分贝

    // Configure CRC calculation
     NRF_RADIO->CRCCNF = (RADIO_CRCCNF_SKIP_ADDR_Skip << RADIO_CRCCNF_SKIP_ADDR_Pos) |  //=1 不包括地址CRC
                         (RADIO_CRCCNF_LEN_Three << RADIO_CRCCNF_LEN_Pos);              //3bitCRC的校验

    //CRC的生成项
    NRF_RADIO->CRCPOLY     = 0x0000065b;/**<  CRC生成项x24 + x10 + x9 + x6 + x4 + x3 + x + 1 */

    //快捷键
    NRF_RADIO->SHORTS      = 0x00000010;   //RSSI快捷键

// 设置访问地址  逻辑地址的0
    NRF_RADIO->BASE0       = Base0_add ;      //访问地址或者逻辑地址0的基地址0 因为下面设置的基地址长度为3,
                                               //所以低字节被截除即取Base0_add的高3个字节
    NRF_RADIO->PREFIX0  =      // 逻辑地址 3 到 0 的前缀地址 最低字节为前缀地址0
                          ( Prefix0_add & 0xff000000)  // Prefix byte of address 3 converted to nRF51822
                        | ( Prefix0_add & 0x00ff0000)  // Prefix byte of address 2 converted to nRF51822
                        | ( Prefix0_add & 0x0000ff00)  // Prefix byte of address 1 converted to nRF51822
                        | ( Prefix0_add & 0x000000ff); // Prefix byte of address 0 converted to nRF51822

        // 配置包 0 的设置
     NRF_RADIO->PCNF0 =  (p_packetHeaderS1len  << RADIO_PCNF0_S1LEN_Pos) |   //0   s1没有
                         (p_packetHeaderS0len  << RADIO_PCNF0_S0LEN_Pos) |   //1   S0为1个字节
                         (p_packetHeaderLFlen  << RADIO_PCNF0_LFLEN_Pos);    //8   LFlen为8bit  也就是1字节

    // 配置包 1 的设置
     NRF_RADIO->PCNF1 =(RADIO_PCNF1_WHITEEN_Enabled         << RADIO_PCNF1_WHITEEN_Pos) |   //使能白化
                       (RADIO_PCNF1_ENDIAN_Little            << RADIO_PCNF1_ENDIAN_Pos)  |   //小端模式
                       (p_balen             << RADIO_PCNF1_BALEN_Pos)   |   //基地址长度为3
                       (p_static_length     << RADIO_PCNF1_STATLEN_Pos) |   //  这个值为0 什么意思还不清楚
                       (PHY_PAYLOAD_MAX_SIZE << RADIO_PCNF1_MAXLEN_Pos);    //净量长度37字节
}
```

图 2-24 Radio 的部分配置程序

2.6、信道

在前面已经提到过低功耗蓝牙使用 40 个信道。低功耗蓝牙的信

道宽度是 2MHz，在链路层，这些信道分为两种：广播信道和数据信道。信道类型对应于之前提到的广播报文和数据报文。如果在报文在广播信道传输，则为广播报文；反之，则为数据报文。

在图 2-1 中可知，低功耗蓝牙共有 3 个广播信道和 37 个数据信道。3 个广播信道在 SIM 频段的不同区域，如果他们集中在某个频段，则可能因为这个频段的深度衰落而造成所有广播无法进行。因此，各个广播信道之间至少相差 24MHz。数据信道在广播信道之间分布，间隔 2MHz。表 2-4 给出了广播信道和数据信道的完整列表，包括链路层信道编号以及中心频率。

广播信道的编号是 37、38 和 39。数据信道的编号是 0~36。这样的编号方式区分广播信道和数据信道使得调频算法的实现非常简单。

表 2-4 全部广播信道和数据信道的信道号及中心频率

频率/MHz	信道编号	类型	频率/MHz	信道编号	类型
2402	37	广播信道	2442	18	广播信道
2404	0	数据信道	2444	19	数据信道
2406	1	数据信道	2446	20	数据信道
2408	2	数据信道	2448	21	数据信道
2410	3	数据信道	2450	22	数据信道
2412	4	数据信道	2452	23	数据信道
2414	5	数据信道	2454	24	数据信道
2416	6	数据信道	2456	25	数据信道
2418	7	数据信道	2458	26	数据信道

2420	8	数据信道	2460	27	数据信道
2422	9	数据信道	2462	28	数据信道
2424	10	数据信道	2464	29	数据信道
2426	38	广播信道	2466	30	数据信道
2428	11	数据信道	2468	31	数据信道
2430	12	数据信道	2470	32	数据信道
2432	13	数据信道	2472	33	数据信道
2434	14	数据信道	2474	34	数据信道
2436	15	数据信道	2476	35	数据信道
2438	16	数据信道	2478	36	数据信道
2440	17	数据信道	2480	39	广播信道

2.6.1、调频

调频算法用于数据连接中，数据信道同 37 个，调频公式(2-1)如下：

$$unmappedChannel = (lastUnmappedChannel + hopIncrement) \bmod 37$$

$$\rightarrow f_{n+1}=(f_n+hop) \bmod 37 \qquad (2\text{-}1)$$

hop 是一个 5~16 的值，每次调频之后中心频率加后 hop 并模 37。因为都是正整数，这个表达式在软件中非常容易实现，软件中通过求余运算便能完成运算。而在应用中第一个 f_n 是多少呢？在 4.0 协议规范中规定：

The *lastUnmappedChannel* shall be ‘0’ for the first connection event of a connection.

也就是说在第一次连接事件中 f_n=0，那么实际上第一次连接事件

中的使用的频率是通过 $f_{n+1}=(0+hop)$ mod 37，也就是 hop 信道编号了。

2.6.2、自适应调频

在传输过程中，并不能保证每个信道都是好信道，自适应调频能够将一个已知的坏信道映射到一个已知的好信道，从而减少其他设备对数据报文传输干扰。为实现这一点，连接中的两个设备都要记录好、坏信道映射关系。如果经过调频公式计算出来的信道是一个坏信道，那么就要通过另一公式(2-2)进行再次映射：

$$remappingIndex = unmappedChannel \bmod numUsedChannels \quad (2\text{-}2)$$

这里的 *numUsedChannels* 是什么呢？*remappingIndex* 怎么用呢？

在记录好信道时，有一个这样的集合 *usedChannel*，这个集合从小到大存放着所有好的信道编号，*numUsedChannels* 则是所有好信道的个数，也就是 *usedChannel* 中元素的个数。得到上面的重映射索引号 *remappingIndex* 之后，通过公式(2-3)进行计算，得到能用的信道。

$$usedChannel[remappingIndex] \quad (2\text{-}3)$$

在低功耗蓝牙中，当发送连接请求时会发送信道图 *ChanelMap*。它用 5 个字节表示，每一个 bit 位表示一个信道，共 37 个信道，第 0bit 位表示第 0 信道，第 36bit 位表示第 36 信道。当某个 bit 位为 0 时表示这个信道不可用，当为 1 时表示这个信道是可用信道。例如，在请求连接时得到：

ChanelMap=00011110 00000000 11100000 00000110 00000000b

➔ *usedChannel[]={9,10,21,22,23,33,34,35,36}*

版权所有

➔*numUsedChannels=9*

假设 *hopIncrement* =7，那么就可以进行自适应调频计算了，第一连接时

$$f_{n+1}=(0+7) \bmod 37=7$$

而 7 信道不是一个可用的好信道，那么就要重映射

$$remappingIndex = 7 \bmod 9=7$$

再通过

$$usedChannel[7]=35$$

编号 35 信道一定是一个可用的信道了，因为它本身就从可用的信道集合中求出来的。当然这仅仅是求出了信道编号，这个编号当然可以直接赋值给白化初始值，但是频率还需要通过前面提到频率公式进行计算。在规范中有关于调频的流程图 2-25。

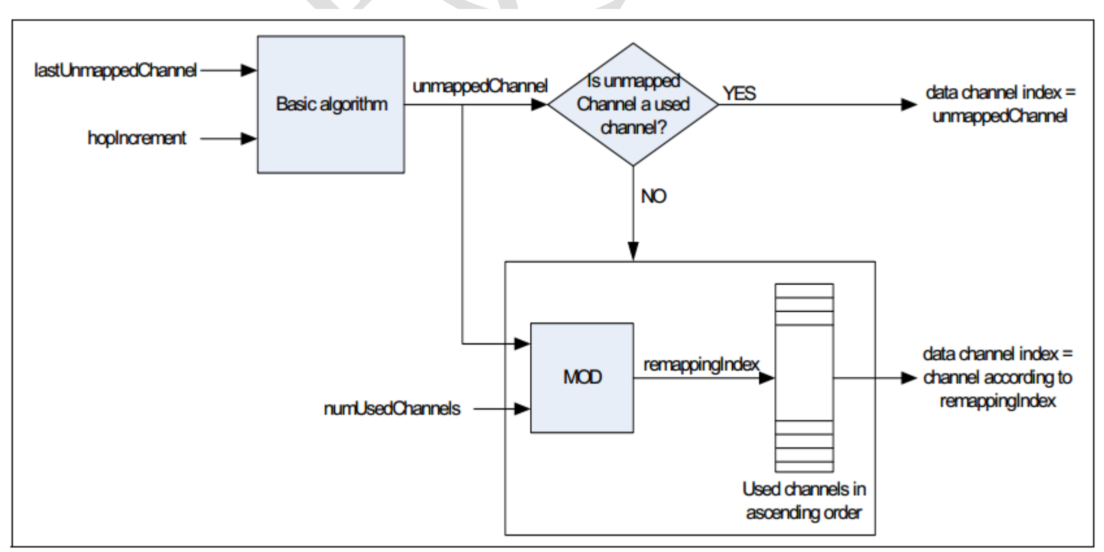


图 2-25 自适应调频流程

在程序中实现其实还蛮简单的(为了程序显示效果将其顶格了)。

```
/*
* 函数名：
```

```
*        Data_Frequency_Channel_Choose  -RADIO  调频计算函数  效果是计算出下一个频率通道和白化初始值
*  参数：
*        -无
*  返回值：
*        -无
*/
void Data_Frequency_Channel_Choose(void)
{
Data_frequency_map.Current_unmappedChannel                                                    =
(Data_frequency_map.Last_UnmappedChannel + Data_frequency_map.Hop_Increment) % 37    ;
//这里得到的是接下来的频率通道号，能不能用？需要和 MAP 进行比较
Data_frequency_map.Last_UnmappedChannel                                                       =
Data_frequency_map.Current_unmappedChannel ;    //上一次的频道等于当前频道
while(1)
{
if (Data_frequency_map.Current_unmappedChannel < 8) //从第 0 个频道字节确定那个能用
{
if(( Data_frequency_map.Channel_Map[0] >> Data_frequency_map.Current_unmappedChannel )
& 0x01 )    //如果这个频道是可用的
{
Data_frequency_map.Current_Frequency_value                                                    =
(Data_frequency_map.Current_unmappedChannel<<1)+4    ; //得到真正的频率，可以直接赋值给 NRF_RADIO->FREQUENCY 寄存器
break;     //  找到了可用的频道    并计算得到了实际寄存器的频率值  后跳出循环
}
else     //这个频道不可用  需要重新映射
{
Data_frequency_map.Current_unmappedChannel                                                    =
Data_frequency_map.Used_Channels[Data_frequency_map.Current_unmappedChannel          %
Data_frequency_map.Num_UsedChannels] ;//这里得到的是接下来的频率通道号，能不能用？需要和 MAP 进行比较
}
}
else if(Data_frequency_map.Current_unmappedChannel < 16)//从第  1  个频道字节确定那个频道能用
{
if(                    (                    Data_frequency_map.Channel_Map[1]                >>
(Data_frequency_map.Current_unmappedChannel- 8) ) & 0x01 )    //如果这个频道是可用的
{
if(Data_frequency_map.Current_unmappedChannel < 11)
{
Data_frequency_map.Current_Frequency_value                                                    =
(Data_frequency_map.Current_unmappedChannel<<1)+4    ; //得到真正的频率，可以直接赋值
```

```
给 NRF_RADIO->FREQUENCY 寄存器
}
else
{
Data_frequency_map.Current_Frequency_value                                    =
(Data_frequency_map.Current_unmappedChannel<<1)+6   ; //得到真正的频率，可以直接赋值
给 NRF_RADIO->FREQUENCY 寄存器
}
break;    //  找到了可用的频道   并计算得到了实际寄存器的频率值  后跳出循环
}
else    //这个频道不可用  需要重新映射
{
Data_frequency_map.Current_unmappedChannel                                    =
Data_frequency_map.Used_Channels[Data_frequency_map.Current_unmappedChannel        %
Data_frequency_map.Num_UsedChannels] ;//这里得到的是接下来的频率通道号，能不能用？
需要和 MAP 进行比较
}
}

else  if(Data_frequency_map.Current_unmappedChannel  <  24)  //从第  2  个频道字节确定那个
频道能用
{
if(                  (                  Data_frequency_map.Channel_Map[2]                >>
(Data_frequency_map.Current_unmappedChannel- 16) ) & 0x01 )   //如果这个频道是可用的
{
Data_frequency_map.Current_Frequency_value                                    =
(Data_frequency_map.Current_unmappedChannel<<1)+6   ; //得到真正的频率，可以直接赋值
给 NRF_RADIO->FREQUENCY 寄存器
break;    //  找到了可用的频道   并计算得到了实际寄存器的频率值  后跳出循环
}
else    //这个频道不可用  需要重新映射
{
Data_frequency_map.Current_unmappedChannel                                    =
Data_frequency_map.Used_Channels[Data_frequency_map.Current_unmappedChannel        %
Data_frequency_map.Num_UsedChannels] ;//这里得到的是接下来的频率通道号，能不能用？
需要和 MAP 进行比较
}
}

else if(Data_frequency_map.Current_unmappedChannel < 32)//从第  3  个频道字节确定那个频
道能用
{
if(                  (                  Data_frequency_map.Channel_Map[3]                >>
(Data_frequency_map.Current_unmappedChannel- 24) ) & 0x01 )   //如果这个频道是可用的
```

```
{
Data_frequency_map.Current_Frequency_value =
(Data_frequency_map.Current_unmappedChannel<<1)+6  ; //得到真正的频率，可以直接赋值给 NRF_RADIO->FREQUENCY 寄存器
break;   // 找到了可用的频道  并计算得到了实际寄存器的频率值 后跳出循环
}
else   //这个频道不可用 需要重新映射
{
Data_frequency_map.Current_unmappedChannel =
Data_frequency_map.Used_Channels[Data_frequency_map.Current_unmappedChannel %
Data_frequency_map.Num_UsedChannels] ;//这里得到的是接下来的频率通道号，能不能用？需要和 MAP 进行比较
}
}

else if(Data_frequency_map.Current_unmappedChannel < 40) //从第 4 个频道字节确定那个频道能用
{
if( ( Data_frequency_map.Channel_Map[4] >>
(Data_frequency_map.Current_unmappedChannel- 32) ) & 0x01 )  //如果这个频道是可用的
{
Data_frequency_map.Current_Frequency_value =
(Data_frequency_map.Current_unmappedChannel<<1)+6  ; //得到真正的频率，可以直接赋值给 NRF_RADIO->FREQUENCY 寄存器
break;   // 找到了可用的频道  并计算得到了实际寄存器的频率值 后跳出循环
}
else   //这个频道不可用 需要重新映射
{
Data_frequency_map.Current_unmappedChannel =
Data_frequency_map.Used_Channels[Data_frequency_map.Current_unmappedChannel %
Data_frequency_map.Num_UsedChannels] ;//这里得到的是接下来的频率通道号，能不能用？需要和 MAP 进行比较
}
}
}
NRF_RADIO->FREQUENCY = Data_frequency_map.Current_Frequency_value; //
Actual frequency (MHz): 2400 + register value
NRF_RADIO->DATAWHITEIV = Data_frequency_map.Current_unmappedChannel; //
白化数据寄存器初值为通道号
}
```

2.7、非连接状态

低功耗蓝牙设备通过广播信道发现其他设备或者通过广播信道让其他设备发送。非链接状态共有 4 类大状态，分别是就绪状态、广播状态、扫描状态和发起状态。

2.7.1、就绪态

就绪状态是一个默认的状态，在这个状态是不能进行数据收发的，它可以进入广播状态、扫描状态和发起状态。

2.7.2、广播态

在广播状态下，链路层在广播事件中发送广播 PDU 。广播事件共有 4 种：

- 非定向可连接事件(ADV_IND)
- 定向可连接事件(ADV_DIRECT_IND)
- 非定向不可连接事件(ADV_NONCONN_IND)
- 非定向扫描事件(ADV_DISCOVER_IND/ADV_SCAN_IND)

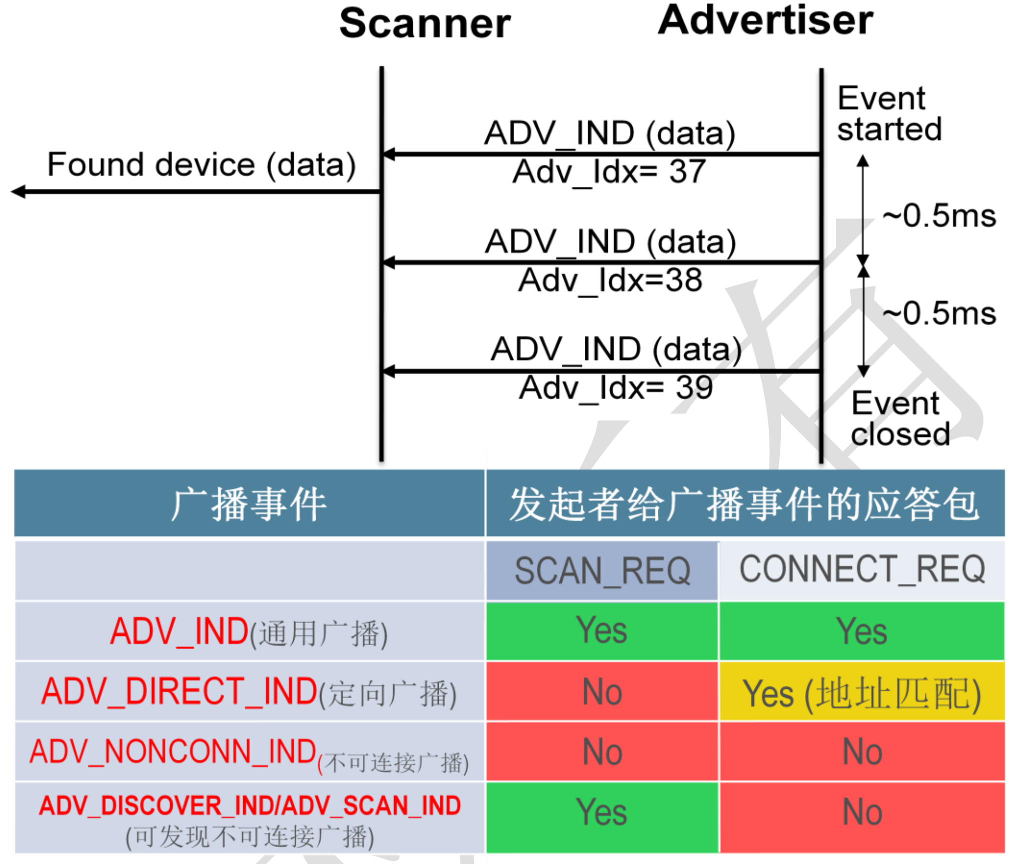
每一个事件都有向对应的 PDU 格式，具体的 PDU 格式见 2.9.1 章节。这 4 个广播事件的区别如图 2-26 所示中的表格。

2.7.2.1、广播通道选择

在广播事件中，每一个广播事件都会在 3 个广播信道中进行数据传输，而且每一个事件都是以最小的信道编号开始传输。也就是说当

版权所有

广播事件来了，这个 PDU 是依次从广播通道 37、38、39 中进行传输。如图 2-26 广播事件。



广播事件	发起者给广播事件的应答包	
	SCAN_REQ	CONNECT_REQ
ADV_IND(通用广播)	Yes	Yes
ADV_DIRECT_IND(定向广播)	No	Yes (地址匹配)
ADV_NONCONN_IND(不可连接广播)	No	No
ADV_DISCOVER_IND/ADV_SCAN_IND (可发现不可连接广播)	Yes	No

图 2-26 广播事件示意图和 4 种广播事件区别

2.7.2.2、广播间隔

所有的非定向广播事件，在两个连续广播事件间的时间即为广播时间间隔。这个间隔满足如下公式(2-4)。

$$T_advEvent = advInterval + advDelay \qquad (2\text{-}4)$$

advInterval 是 0.625ms 的倍数，在 20ms~10.24s 之间。如果广播类型是非定向扫描事件或者非定向不可连接广播事件，这个值不能小于 100ms。如果广播事件类型是非定向可连接事件，这个是只需大于

20ms 即可。

advDelay 是一个随机数，在 0ms~10ms 之间，在每一个广播事件中都有。由于设备间的时钟会有不同程度的漂移，所以这个随机延时的作用不但能消除设备之间时钟漂移，还能避免相同信道及时间点上的冲突。图 2-27 示意出了广播事件间的间隔。

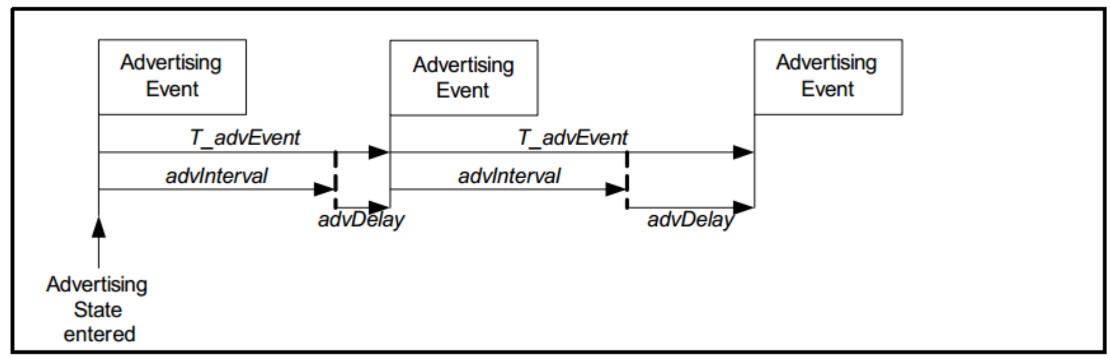


图 2-27 广播事件时间间隔

2.7.2.3、非定向可连接广播事件(ADV_IND)

这其实就是链路层通过广播信道发送(ADV_IND PDU)通用广播报文。这个报文发送之后可以接收由扫描者发送的(SCAN_REQ PDU)扫描请求，或者由发起者发送的(CONNECT_REQ PDU)连接请求。而接收后链路层需要早同一个信道上进行扫描者或者发起者的应答。当接收的数据报文没能通过广播滤波政策，要么就用下一个广播信道进行广播要么就关闭广播事件。

如果接收到的 SCAN_REQ PDU 通过了滤波政策，那么广播者需要在同一信道并且在接收到数据到发送 SCAN_RSP PDU 扫描应答报文的时间一定是 150±2 μs 完成。

如果接收到 CONNECT_REQ PDU，那么就进入连接状态，这个过

程比较复杂，这个时候并不需要进行应答。

上面有说道广播事件之间有时间间隔，那么广播本身里面有 3 个信道进行数据传输，这 3 个信道之间的时间间隔是多少？在协议中有规定：

> The time between the beginning of two consecutive ADV_IND PDUs within an advertising event shall be less than or equal to 10 ms. The advertising state shall be closed within the advertising interval.

也就是两个连续的通用广播之间的时间必须小于等于 10ms，如图 2-28。

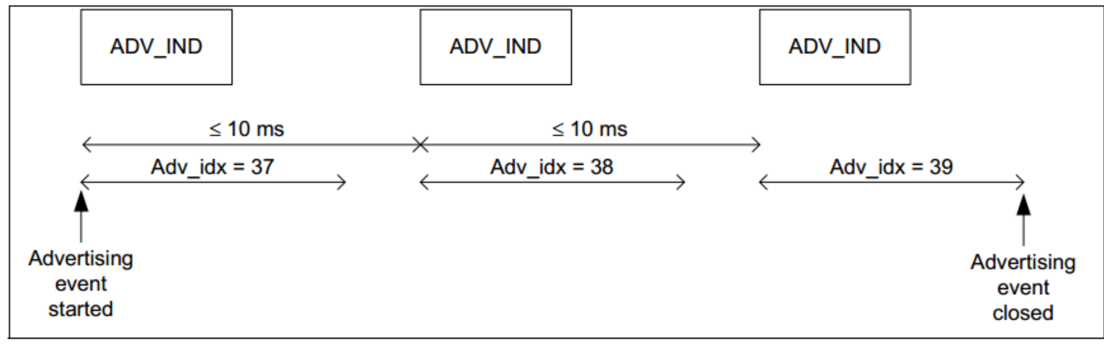


图 2-28 非定向可连接广播仅仅通用广播

我在软件中是通过定时器来处理广播事件时间间隔和广播事件中的信道之间的时间间隔的。

当有扫描请求包在广播事件中的中间信道上收到时，应答如图 2-29。注意图中的 T_IFS，这是帧间隔，它的时间是 150us。

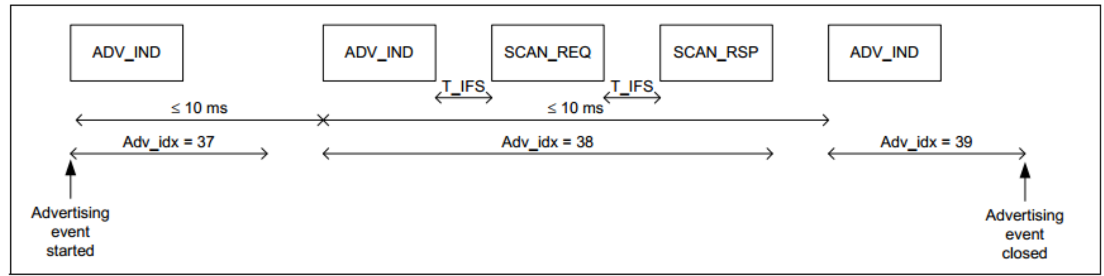


图 2-29 广播事件 38 信道的扫描请求和扫描应答

当有扫描请求包在广播事件中的最后信道上收到时，应答如图 2-30。

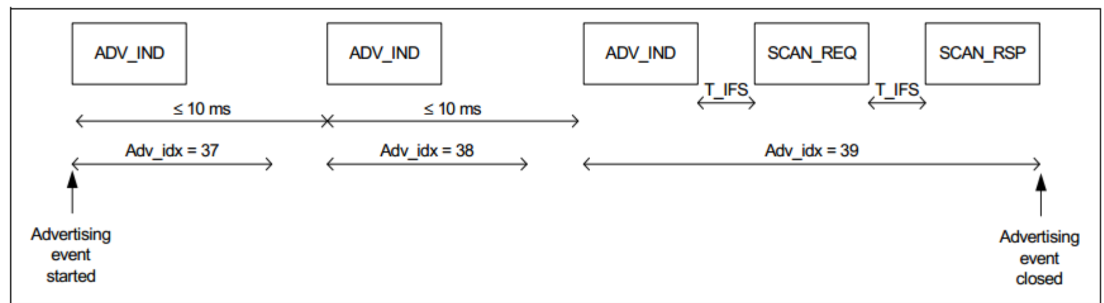


图 2-30 广播事件 39 信道的扫描请求和扫描应答

当有连接请求包在广播事件中的信道上收到时，没有应答如图 2-31。

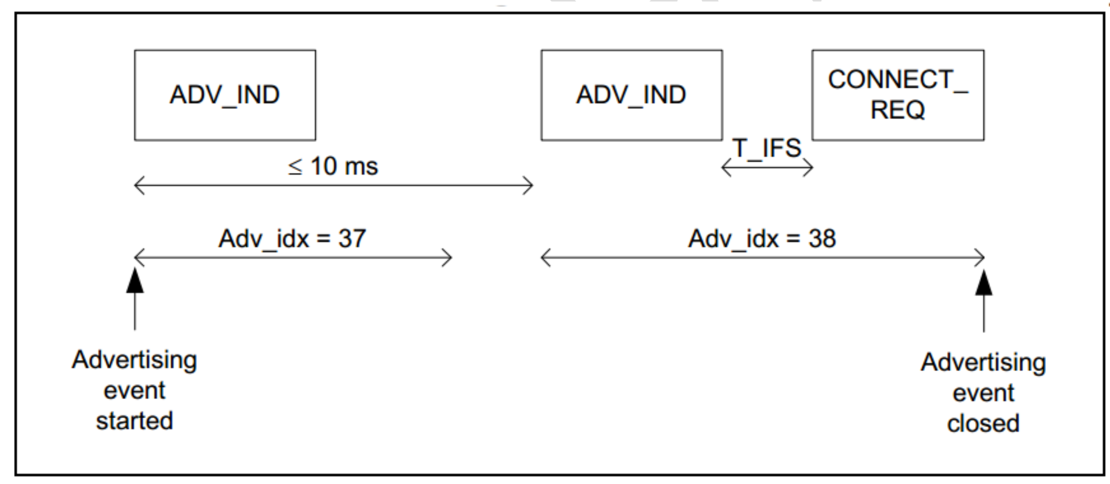


图 2-31 广播事件中收到连接请求包

2.7.2.4、定向可连接广播事件(ADV_DIRECT_IND)

这个广播是为了快速建立连接。这种报文包含两个地址：广播者地址和发起者的地址。发起设备收到发给自己的定向广播报文后，可以立刻发送连接请求事件作为回应，并进入连接状态。

定向广播事件有特殊的时序要求。完整的广播事件必须每 3.75ms 之内重复一次。这一要求似的扫描设备只需扫描 3.75ms 便可以收到定向广播设备的消息。如图 2-32 所示。

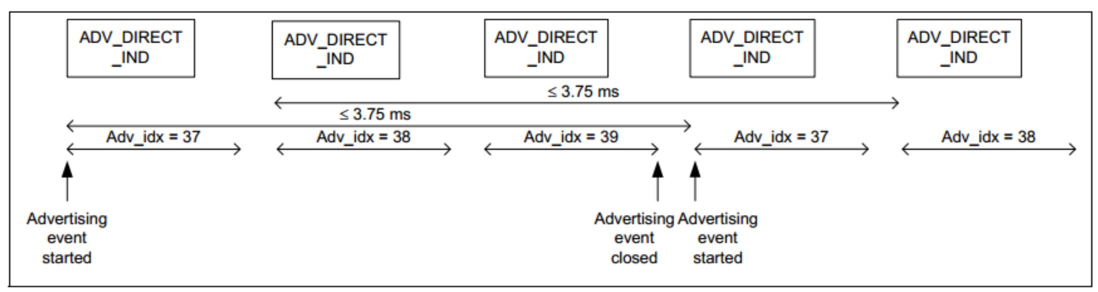


图 2-32　定向可连接广播

这么快的发送速度让周围充斥着广播信道，这使得该区域的其他广播事件无法进行广播。所以，协议规定：

> The Link Layer shall exit the Advertising State no later than 1.28 s after the Advertising State was entered.

也就是定向广播不能持续 1.28s 以上的时间。如果主机没有主动要求停止，或者连接没有建立，控制器都会自动停止广播。一旦超过 1.28s，主机只能使用通用广播让其他设备连接。

2.7.2.5、非定向不可连接事件(ADV_NONCONN_IND)

这是一个很奇葩的事件，它像是一个车模，大张旗鼓的告诉别人，但是不允许别人摸她，而这个事件比模特更加的奇葩，模特至少可以合个影，但是这个事件大声的告诉别人我在这里，之后就不搭理别人了，不接受任何信息，只管自己在每个广播事件中发送数据。时间要求和通用广播事件一样。它只能根据主机的要求在广播态和就绪态之

间切换，也是唯一可用于只有发射机而没有接收机设备的广播类型。

2.7.2.6、可发现不可连事件(ADV_DISCOVER_IND/ADV_SCAN_IND)

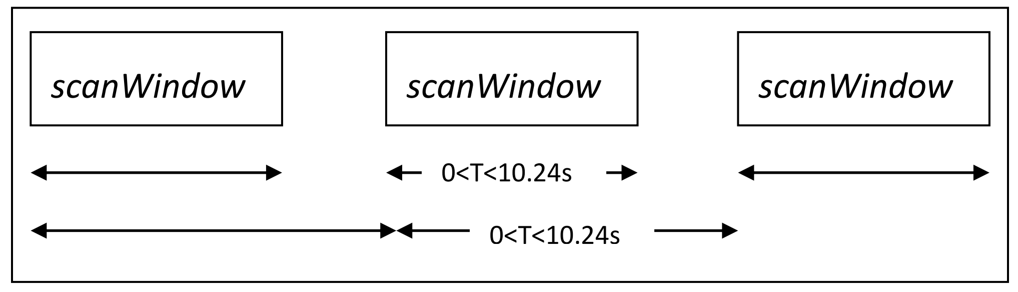
这个广播其实是一个非定向可发现的广播，它和通用广播的时间控制是一样的，应答也是 SCAN_REQ PDU 和 SCAN_RSP PDU，这个广播和通用广播的区别是，它不能建立连接，只能处于广播态或者就绪态。

这是一种适用于广播数据的广播形式，动态数据可以包含于广播数据中，而静态数据可以包含于扫描响应数据之中。

2.7.3、扫描态

在这里不再讲述主动扫描和被动扫描，这里提出两个概念：扫描窗口和扫描间隔。扫描窗口（*scanWindow*）是链路层侦听广播通道时持续的时间；扫描间隔（*scanInterval*）是两个连续的扫描窗口开始之间的时间。

在协议中规定：The *scanWindow* and *scanInterval* parameters shall be less than or equal to 10.24 s.也就是他们的最大值是 10.24s。而且扫描窗口必须小于等于扫描间隔，当扫描窗口等于扫描间隔时，相当于连续扫描。如图 2-33。



版权所有

图 2-33 扫描窗口和扫描间隔

2.7.4、发起态

发起态进入连接后成为主机设备。它做的工作是在扫描态的事情之后发送连接请求。当接收到了扫描应答后，发送连接请求事件，从而跳出发起态并进入连接状态成为主机。

2.7.5、软件设计广播状态流程图

我在软件设计中，广播状态值关注了通用广播和扫描应答以及连接请求事件。程序设计中需要注意几个细节：

- 广播事件的连接间隔控制
- 广播事件中 3 个信道之间的时间间隔控制
- 信道控制
- 白化控制
- 广播报文发送准备和接收解析
- Radio 的收发切换控制(前面已经讲过这个转换时间紧迫)

流程图 2-34 如下：

版权所有

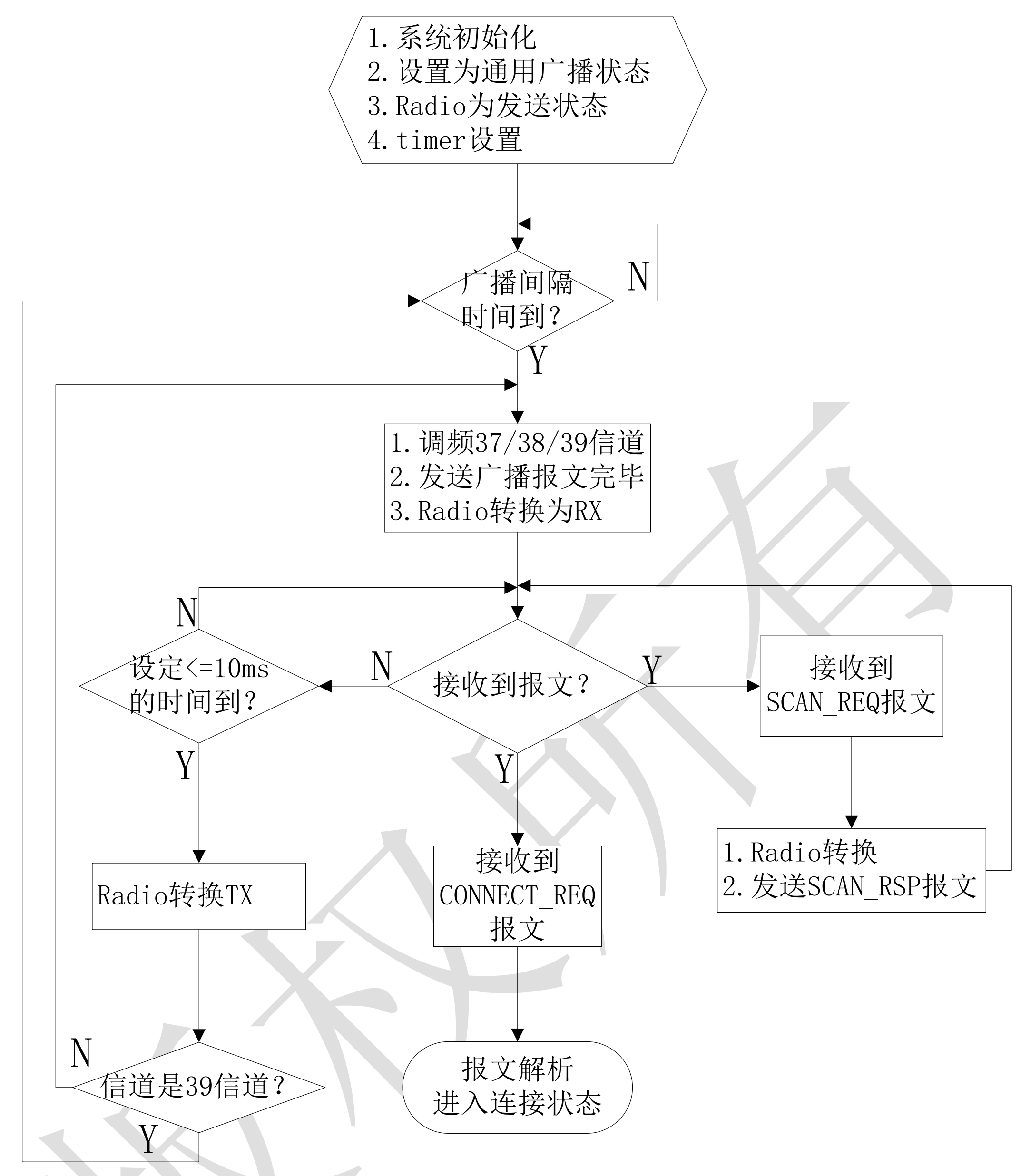


图 2-34 广播状态软件流程图

版权所有

2.8、连接状态

进入连接状态是广播状态下的 CONNECT_REQ PDU 的发送和接收。当进入连接状态后，两个设备就担任不同的角色，发送 CONNECT_REQ 的成为主机，接收 CONNECT_REQ 的成为从机。之后就会在这两个设备之间发生连接事件。

2.8.1、连接事件

和广播事件一样，每隔一定的时间连接事件由主机从数据信道发送一个数据包，从机接到数据如果有数据发送需要在 150±2 μs 做成应答。也就是说每一个连接事件中至少包含主机发送的一个包，从机可以不发送包。

- 连接事件的时间由两个参数决定：connection event interval (*connInterval*), and slave latency (*connSlaveLatency*).即连接间隔和从机潜伏期。
- *anchor point*

  连接事件开始的点叫做锚点(The start of a connection event is called an anchor point)。主机在锚点开始连接事件，从机需要在锚点前进入侦听状态。
- *connEventCounter*

  这是连接事件计数器，在协议中规定

Both the master and the slave shall have a 16-bit connection event counter (*connEventCounter*) for each Link Layer connection.

  主机和从机都有一个 16 位的连接事件计数器，这个值是为了这两个设备之间的同步。这个值只要是连接事件参数就会加 1，当然第一个连接事件时这个值为 0 而不是 1。无论从机潜伏期的值是多少，只有从机接收到主机的连接事件这个值在从机里面就要加 1。当这个值到了 0xFFFF 时，会翻转到 0x0000，

版权所有

重新开始计算。

2.8.2、监管超时

如果连接状态下，很长时间都没有连接事件发生，或者连接事件发送总是得不到从机的应答，是不是该考虑连接出了问题呢？在协议中也有规定一个参数——连接监管超时 Connection supervision timeout (*connSupervisionTimeout*)。这个值是 10ms 的倍数，并且在 100ms~30.0s 之间，同时它必须大于(1 + *connSlaveLatency*) * *connInterval*，这应该好理解，从机潜伏期是双方沟通和允许的事情，所以超时时间一定要大于上面表达式才行。这里有个特殊情况，在发送 CONNECT_REQ 之后，如果链路层的连接超时时间达到 6*connInterval*，那么将认为连接建立失败。我还是将协议的原文复制一下吧！避免歪曲理解。

> If the Link Layer connection supervision timer reaches 6 * *connInterval* before the connection is established, the connection shall be considered lost. This enables fast termination of connections that fail to establish.

2.8.3、连接事件传输窗口

这是一个头痛了我很久的问题。在这我还是先讲一下 CONNECT_REQ 的报文组成。如图 2-35 为 CONNECT_REQ PDU 的有效组成。

Payload		
InitA (6 octets)	AdvA (6 octets)	LLData (22 octets)

图 2-35 CONNECT_REQ PDU payload

InitA 和 AdvA 分别是发起者的设备地址和广播者的设备地址。图 2-36 为 LLData 的组成结构。

LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

图 2-36 LLData field structure in CONNECT_REQ PDU's payload

LLData 包含有 10 个区域：

- AA 区域共有 4 个字节。"AA"是 Access Address 接入地址的两个头字母，前面有提到连接状态下的接入地址是有主机在连接请求包中提供。
- CRCInit 区域共有 3 个字节。是连接状态下的 CRC 校验的移位寄存器的初始值。
- WinSize 区域共有 1 个字节。它为传输窗口的时间，而且这个值乘以 1.25ms 才是真正的传输窗口时间。即：
  - *transmitWindowSize* = WinSize * 1.25 ms

    它是 1.25ms 的倍数，范围：

    [1.25ms,MIN(10ms, *connInterval* - 1.25 ms)]
- WinOffset 区域共有 2 个字节。它是传输窗口偏移的时间，同样这个值需要乘以 1.25ms 才是真正的传输窗口偏移时间。**这个参数其实只在第一连接事件发生前或者连接参数更新时用**

一次，之后的时间主要就是连接间隔和扫描窗口控制。范围：

- *transmitWindowOffset* = WinOffset * 1.25 ms

  它是 1.25ms 的倍数，范围：

  [0, *connInterval*]

- Interval 区域共有 2 个字节。这个就是传说中的连接间隔时间了。时间为：

  - *connInterval* = Interval * 1.25 ms

    它是 1.25ms 的倍数，范围：

    [7.5ms,4.0S]

- Latency 区域共有 2 个字节。这个就是从机潜伏次数，这个是次数不是时间啊！什么作用呢？就是从机没有必要对于每一个主机的连接事件都进行应答，例如假设 *connSlaveLatency=5*，那么连续 5 个连接事件，如果从机本身没有数据需要发送，那么这个 5 个事件从机可以不用搭理，主机也不会认为连接出了问题。

  - *connSlaveLatency* = Latency

    范围：

    [0, MIN(((connSupervisionTimeout / connInterval) - 1),500)]

- Timeout 区域共有 2 个字节。这个时间前面有提到它是 10ms 的倍数。所以真正的时间为：

  - *connSupervisionTimeout* = Timeout * 10 ms

    范围：

版权所有

[MAX(100 ms, (1 + *connSlaveLatency*) * *connInterval*),32.0s]

- ChM 区域共有 5 个字节。它就是信道地图了，具体看信道章节。
- Hop 区域共有 5bits。这是 5bits，这是 5bits，重要的事情说 3 遍。这个在自适应调频中使用的调频增量。
- SCA 区域共有 3bits。这个东东就是主机的时钟精度，这个是最难用的一个参数了，在传输窗口扩展中会用到这个参数。这 3 个 bit 表达能力是有限的，所以在协议中有一张表，如图 2-37。

SCA	*masterSCA*
0	251 ppm to 500 ppm
1	151 ppm to 250 ppm
2	101 ppm to 150 ppm
3	76 ppm to 100 ppm
4	51 ppm to 75 ppm
5	31 ppm to 50 ppm
6	21 ppm to 30 ppm
7	0 ppm to 20 ppm

图 2-37 SCA 区域编码

也就是说传输窗口其实是在 CONNECT_REQ 包中的，并且窗口偏移(WinOffset)只有是从广播态进入连接态时使用，或者在参数更新时使用，而 WinSize 时间是每个连接事件都必须使用的。

版权所有

2.8.4、连接状态--主机

在第一个连接事件中，锚点是由 CONNECT_REQ PDU 中的参数决定的。当主机发送完 CONNECT_REQ 报文之后，接着发送第一个连接事件的报文，而从机接到 CONNECT_REQ 报文后，做好进入连接状态的准备工作后，根据 CONNECT_REQ 中的参数，决定开始侦听的时间，但是必须保证，主机在锚点前从机必须已经进入侦听状态。怎么保证这个时间点呢？ 来张图看看协议中是怎么规定的，如图 2-38。

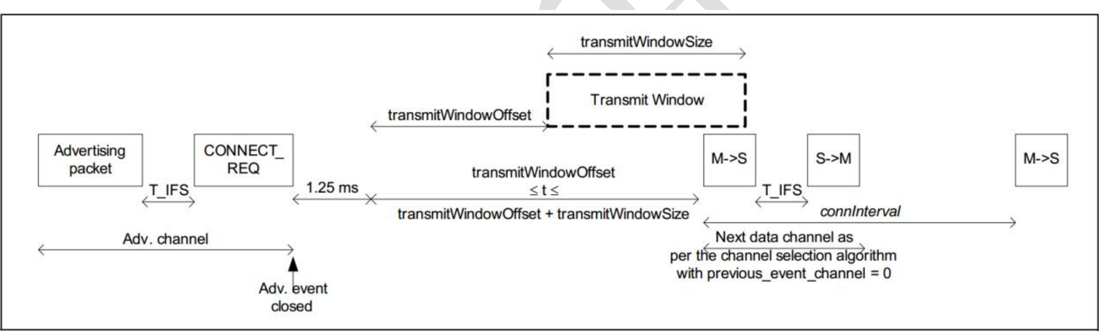


图 2-38 广播态进入连接态传输偏移不为 0

协议中有这么一段：

> The CONNECT_REQ PDU includes three parameters used to determine the transmit window. The transmit window starts at transmitWindowOffset+ 1.25 ms after the end of the CONNECT_REQ PDU, and the transmitWindowSize parameter shall define the size of the transmit window.The connIntervalis used in the calculation of the maximum offset and size of the transmit window.
> Therefore the start of the first packet will be no earlier than 1.25 ms + transmitWindowOffset and no later than 1.25 ms + transmitWindowOffset + transmitWindowSize after the end of the CONNECT_REQ PDU transmitted in the advertising channel.

从上可知传输偏移一定是在 1.25ms 之后。也就是说，连接状态的第一包数据传输一定是在**[1.25 ms + transmitWindowOffset, 1.25 ms + transmitWindowOffset + transmitWindowSize]**之间。

图 2-38 为传输偏移不为 0 的连接示意图，下图 2-39 为传输偏移为 0 的情况。

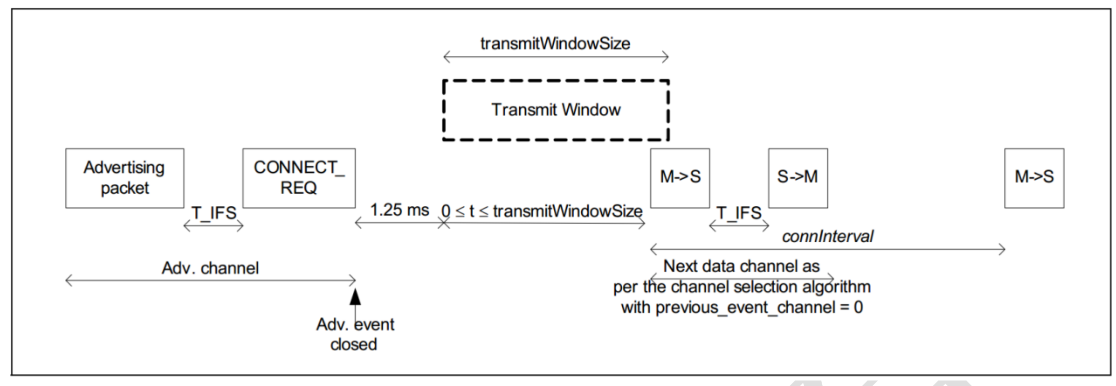


图 2-39 广播态进入连接态传输偏移为 0

2.8.5、连接状态--从机

对于主机来说控制着连接状态下的几乎所有权利，可是对于配合者的从机却是有些难办。如果理想状态，晶振没有任何偏差，那么有了这个时序，就很容易完成这个程序的编写。事实总是很残酷的，晶振总有一些漂移的，这样就会带来主机肯定是发送了连接事件中的第一个报文，但是从机并不一定能接收得到，所以看看从机如果在第一个连接事件中没有接收到数据报文的话，时间又是怎么控制的，有图有真相，见图 2-40。

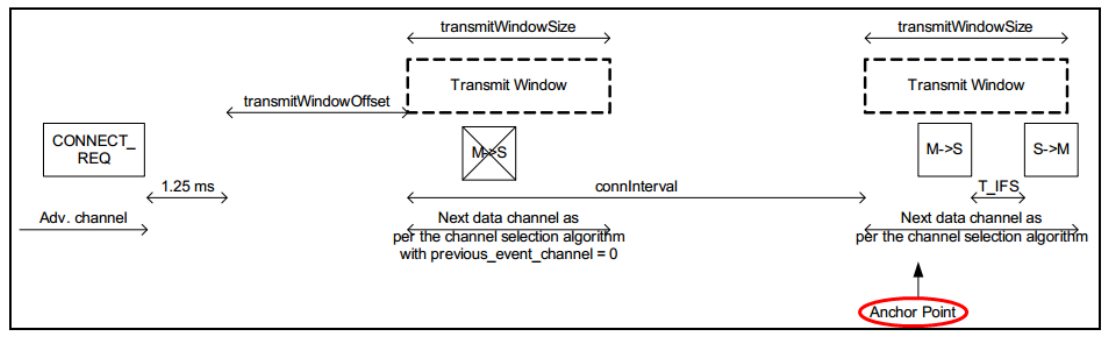


图 2-40 从机第一个连接事件接收失败处理

先讲述一下从机进入连接状态的过程：对于从机接收到 CONNECT_REQ 之后，协议留了至少 1.25ms 的时间让其准备进入连接状态，进入之后偏移一个传输窗口的时间，进而进入扫描窗口，开始侦听空中的包，看是否有自己需要的报文，所以在上面的图中传输窗口都是用虚线的，也就是说传输口规定了最大时间，但是并不一定要等这么久的时间，收到数据就可以去处理，并开始计时连接间隔时间，等到连接间隔时间到，再次进入扫描窗口，侦听报文。

在这再重点说几个问题：

- 什么情况表示从机接收到了第一个连接事件

这里我不敢乱翻译，还是来原文吧！

> The first packet received, regardless of a valid CRC match (i.e., only the access code matches), in the Connection State by the slave determines the anchor point for the first connection event, and therefore the timings of all future connection events in this connection.

我的理解是第一个包接收时，无论 CRC 是否正确，只需要接入地址正确，在连接状态下锚点由从机接收的第一个连接事件报文决定，并决定了将来的锚点。

- 连接间隔时间的起点在哪？

连接间隔时间的起点有两个位置：

  - 当从机接收报文成功，连接间隔时间起点，就是接收到报文的那个时刻，当然对于 nrf51822 来说，就是 Radio 的 END 事件产生的时候了。

- ➢ 当从机接收报文失败，连接间隔时间起点，就是扫描窗口的开始时刻。

- 从机在传输窗口中没有接收到第一个连接事件

这种情况，从机不得不等待连接间隔时间后再次进入传输窗口，等待接收下一个报文。虽然接收失败，但是还是有几点要注意的：

- ➢ 连接事件计数值 connEventCount 这个值还是得加 1 的。
- ➢ 数据通道的频率还得接着跳到下一个频率。

2.8.6、连接事件关闭

这里是连接事件关闭，不是连接关闭啊！

如果主机和从机之间没有过多的数据发送，那么连接事件仅仅是例行事件，主机发送一个报文，从机要么回应要么不回应，主机会关闭连接事件。但是如果主机或者从机在一个连接事件中有多个数据需要发送时，在前面数据报文中有提到 MD 这个标志(图 2-11)。这个标志将决定连接事件的关闭，如图 2-41。

		Master	
		MD = 0	**MD = 1**
Slave	**MD = 0**	Master shall not send another packet, closing the connection event. Slave does not need to listen after sending its packet.	Master may continue the connection event. Slave should listen after sending its packet.
	MD = 1	Master may continue the connection event. Slave should listen after sending its packet.	Master may continue the connection event. Slave should listen after sending its packet.

图 2-41 MD 位用于关闭连接事件。

2.8.7、窗口扩展

这是我在软件中最难处理的问题，到现在都还没有处理好。

前面有提到晶振总是有漂移的，那么从机希望的锚点，不一定按时出现，要么提前要么推迟，不管提前或者推迟，如果还是在传输窗口中也好，但是如果不在那就麻烦了，这个时候就要用到窗口扩展了。在 CONNECT_REQ PDU 报文中，主机会将自己的时钟精度发给从机，从机就需要通过主机和从机的时钟精度的偏差来计算出需要扩展的时间是多少。这个扩展的时间就是扩展给扫描窗口用的。相当于把扫描窗口的时间加大。既然要加大为什么一定要计算呢？这就是低功耗的魅力了，每一个环节都在进行功耗的节省，而射频是最耗能量的，所以得精打细算。

不过不幸运的是，我读不懂协议汇中的意思，这里贴出原文并留有空白页，如果有谁读到了这节，有自己的见解的话，希望能在下面做一下记录，或者告诉我。特别是红色部分，没明白什么意思。

后来在 QQ 群中有人说计算公式如下：

[(Master PPM + Slave PPM)/106] * 连接间隔时间 + 2

上式还没用用于程序，所以也还没有验证其正确性。

版权所有

Because of sleep clock accuracies, there is uncertainty in the slave of the exact timing of the master's anchor point. Therefore the slave is required to re-synchronize to the master's anchor point at each connection event where it listens for the master. If the slave receives a packet from the master regardless of a CRC match, the slave shall update its anchor point. The slave calculates the time when the master will send the first packet of a connection event (*slaveExpectedAnchorPoint*) taking clock jittering, and in the case of connection setup or a connection parameter update the transmit window, into account. The slave shall also use the masters sleep clock accuracy (*masterSCA*) from the CONNECT_REQ PDU, together with its own sleep clock accuracy (*slaveSCA*) and **the anchor point of the last connection event where it received a packet from the master (*timeSinceLastAnchor*)** to calculate the time it needs to receive. The increase in listening time is called the window widening. Assuming the clock inaccuracies are purely given in parts per million (ppm), it is calculated as follows:

windowWidening = ((*masterSCA* + *slaveSCA*) / 1000000) * *timeSinceLastAnchor*

During connection setup or during a connection parameter update, the slave should listen for windowWidening before the start of the transmit window and until windowWidening after the end of the transmitWindow for the master's anchor point. At each subsequent connection event, the slave should listen for *windowWidening* before the start of the *slaveExpectedAnchorPoint* and until *windowWidening* after *slaveExpectedAnchorPoint* for the master's anchor point.

2.8.8、软件设计连接态流程图

这里再说一下低功耗蓝牙中的应答事件并不一定要在同一个连接事件中立刻应答，例如主机在连接事件中发送了一个请求版本信息的请求，从机可以下回应一个空包，等到下一个连接事件时，从机再发送版本信息给主机。也就是说每次连接事件中，从机应答的都是主机在上一个连接事件中提出的问题，而主机在连接事件中发送的报文也有可能是从机上一个连接事件中提出的问题。之所以这样做是因为，在 150±2 μs 完成数据包的解析和发送包的准备是比较困难的。

连接状态下的状态机相对比广播状态并没有难多少。但是却也有些抓狂的事情：

- 扫描窗口的把握
- 连接间隔时间起点的把控
- 自适应调频和白化控制
- 报文上传到 HOST 层同时 HOST 准备发送报文

在连接状态，我用的时钟是 RTC 时钟，这里其实还有个问题，RTC 时钟最小的定时单位是 30us，而实际上主从的时钟精度加起来计算出来的时钟偏差都没有 30us，所以我在程序处理时，直接将扫描窗口扩展了两个单位，也就是 60us，这肯定是没有问题的，只是功耗增加。流程图 2-42 如下。

版权所有

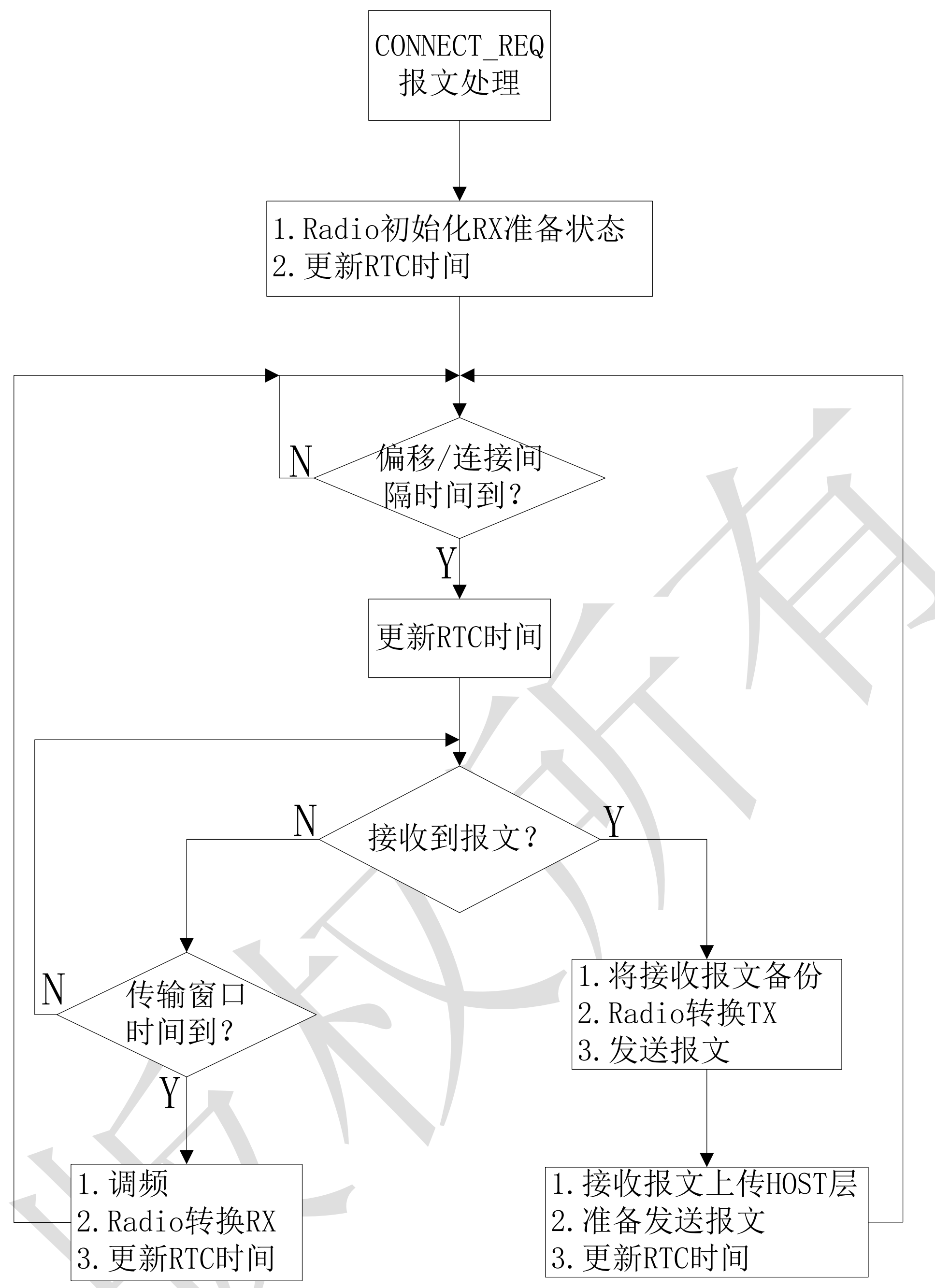


图 2-42 连接状态软件流程图

版权所有

2.9、空中接口包

在 2.3 节中已经讲了报文的格式，这一节讲解广播信道和数据信道下的具体的包内容，以及链路层的相关控制包。

2.9.1、广播通道 PDU

报文结构如表 2-2，广播包的类型共有 7 种，如图 2-43。

PDU Type $b_3b_2b_1b_0$	Packet Name
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111-1111	Reserved

图 2-43 广播信道 PDU 的包头类型区域代编码

2.9.1.1、广播数据的结构

首先讲广播数据的结构吧！在 4.0 规范中的第 3 卷 PART C 的第 11 节就有“Advertising and Scan Response data format”(page1735)这么一节。先来张图 2-44 吧。

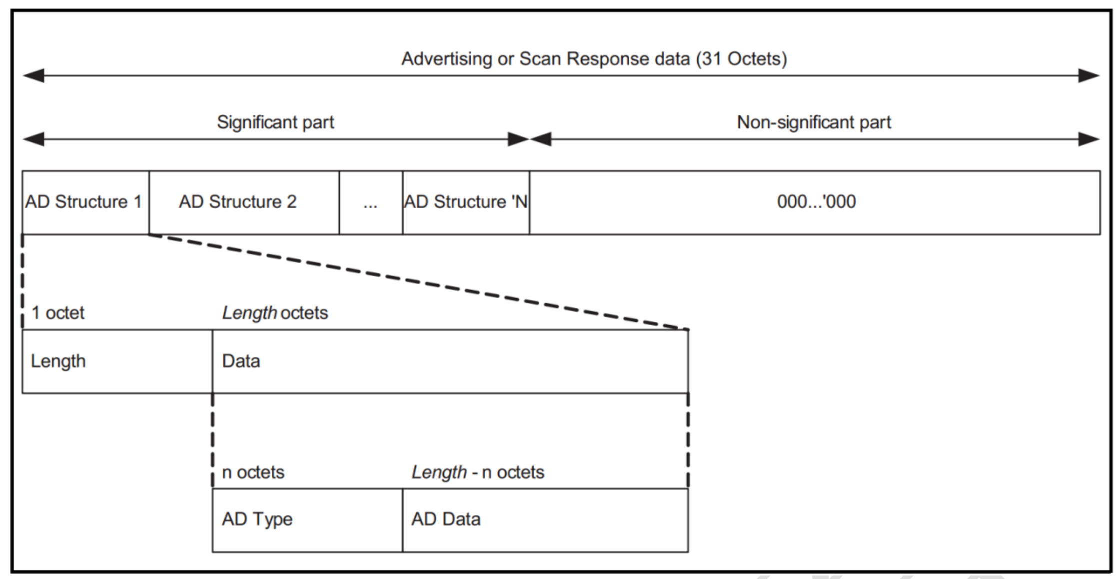


图 2-44 Advertising and Scan Response data format

图 2-44 中可以看到，对于 GAP 来说，广播和扫描应答数据的 31 个字节都用到了，包括有意义的部分和没有意义用 0 填充的部分，当然发送到空中只有有意义的部分，也就是所 GAP 发送到底层时是 31 字节，但是底层只需要有意义的部分。而有意义的部分是由许多相同的 AD Structure 组成，这个结构也有一定的格式：每个结构的第一个字节是长度，长度后面就是数据了，这个数据的长度就是前面的长度表示的多少个字节。而数据部分包括 AD Type 和 AD Data。在 4.0 规范手册的 1736 页有 AD Type 的定义，同时在 1761 页中"APPENDIX C (NORMATIVE): EIR AND AD FORMATS"中详细介绍每个 AD Type 的意义。下面详细介绍每个 AD Type 和 AD Data。

2.9.1.1.1、广播类型定义 AD Type

在 4.0 的规范中，AD Type 共有 11 种。如表 2-5

表 2-5 广播类型定义 AD Type

AD Type	Value	描述
Flags	0x01	广播出自己蓝牙某些特性
Service UUIDs	0x02~0x07	广播出自己服务的 UUID
Local Name	0x08/0x09	广播出自己的蓝牙名字
TX Power Level	0x0A	广播出自己的射频发射功率
Simple Pairing Option OOB Tags	0x0D~0x0F	广播出安全管理带外标签(本文忽略)
Security Manager TK Value	0x10	广播出带外方式配对绑定时的 TK(本文忽略)
Security Manager OOB Flags	0x11	广播出带外特性标志(本文忽略)
Slave Connection Interval Range	0x12	广播出自己希望的连接参数范围
Service Solicitation	0x14/0x15	广播出自己希望来接自己的主机有特定的服务
Service Data	0x16	服务数据
Manufacturer Specific Data	0xFF	广播出厂商信息（用户可以放自定义数据）

2.9.1.1.2、广播数据定义 AD Data

- **FLAGS**

FLAGS 类型的解释如图 2-45。

Value	Description	Bit	Information
0x01	Flags	0	LE Limited Discoverable Mode
		1	LE General Discoverable Mode
		2	BR/EDR Not Supported (i.e. bit 37 of LMP Extended Feature bits Page 0)
		3	Simultaneous LE and BR/EDR to Same Device Capable (Controller) (i.e. bit 49 of LMP Extended Feature bits Page 0)
		4	Simultaneous LE and BR/EDR to Same Device Capable (Host) (i.e. bit 66 of LMP Extended Feature bits Page 1)
		5..7	Reserved

图 2-45 Flag 广播类型数据

标志位是每个广播都必须发送的一个类型，这主要是告诉对方自己设备硬件能支持什么：是有限制发现模式还是通用发现模式、是否

支持常规蓝牙、作为从机同时支持常规蓝牙和 BLE 还是作为主机同时支持常规蓝牙和 BLE。

- **SERVICE**

服务 UUID 类型数据定义如图 4-46。

Value	Description	Information
0x02	16-bit Service UUIDs	More 16-bit UUIDs available
0x03	16-bit Service UUIDs	Complete list of 16-bit UUIDs available
0x04	32-bit Service UUIDs	More 32-bit UUIDs available
0x05	32-bit Service UUIDs	Complete list of 32-bit UUIDs available
0x06	128-bit Ser-vice UUIDs	More 128-bit UUIDs available
0x07	128-bit Ser-vice UUIDs	Complete list of 128-bit UUIDs available

图 4-46 SERVER UUID 广播类型数据

- **LOCAL NAME**

本地名称类型数据定义如图 4-47。

Value	Description	Information
0x08	Local Name	Shortened local name
0x09	Local Name	Complete local name

图 4-47 LOCAL NAME 广播类型数据

- **TX POWER LEVEL**

发射功率类型数据定义如图 4-48。

Value	Description	Information
0x0A	TX Power Level (1 byte)	0xXX:-127 to +127dBm Note: when the TX Power Level tag is not present, the TX power level of the packet is unknown.

图 4-48 TX POWER LEVEL 广播类型数据

如图 2-49 为采集到空中的广播报文，这个报文共包含了 4 个 AD 结构体。

```
Advertising Data: 0809696c736b73646a020106020af807030d180f180a18
  local name: ilsksdj
    length: 0x08
    type: Complete Local Name (0x09)
    local name: ilsksdj
  flags: 0x06
    length: 0x02
    type: Flags (0x01)
    ...0 .... = Simultaneous LE and BR/EDR (Host): False
    .... 0... = Simultaneous LE and BR/EDR (Controller): False
    .... .1.. = BR/EDR not supported: True
    .... ..1. = LE general discoverable: True
    .... ...0 = LE limited discoverable: False
  TX power level: -8
    length: 0x02
    type: Tx Power Level (0x0a)
    TX power level: 121
  16 bit uuids (complete): 0d180f180a18
    length: 0x07
    type: Complete List of 16-bit Service Class UUIDs (0x03)
    16 bit uuid: 0x180d
    16 bit uuid: 0x180f
    16 bit uuid: 0x180a
CRC: 0x8036ab

0000  04 06 30 01 fa 02 06 0a  01 25 48 00 00 dd ca 05   ..0..... .%H.....
0010  00 d6 be 89 8e c0 1d e8  39 7e 6e ed d2 08 09 69   ........ 9~n....i
0020  6c 73 6b 73 64 6a 02 01  06 02 0a f8 07 03 0d 18   lsksdj.. ........
0030  0f 18 0a 18 80 36 ab                               .....6.
```

图 2-49 sniffer 采集的广播包

- **SLAVE CONNECTION INTERVAL RANGE**

这个的作用是当主机连接时，可以参考从机所期望的连接参数，具体的参数设置可以参考 2.8.1 节。从机连接参数范围类型数据定义

如图 2-50。

Value	Description	Information
0x12	Slave Connection Interval Range	The first 2 octets defines the minimum value for the connection interval in the following manner: $connInterval_{min}$ = Conn_Interval_Min * 1.25 ms Conn_Interval_Min range: 0x0006 to 0x0C80 Value of 0xFFFF indicates no specific minimum. Values outside the range are reserved. (excluding 0xFFFF) The second 2 octets defines the maximum value for the connection interval in the following manner: $connInterval_{max}$ = Conn_Interval_Max * 1.25 ms Conn_Interval_Max range: 0x0006 to 0x0C80 Conn_Interval_Max shall be equal to or greater than the Conn_Interval_Min. Value of 0xFFFF indicates no specific maximum. Values outside the range are reserved (excluding 0xFFFF)

图 2-50 SLAVE CONNECTION INTERVAL RANGE 广播类型数据

- **SERVICE SOLICITATION**

这个用于从机希望连接它的主机具有某些服务，例如带 ancs 服务的从机希望是苹果手机连接，所以在广播中广播出这个祈求之后，安卓手机发现自己没有这个服务时，可以不去连接这个设备。期望服务类型数据定义如图 2-51。

Value	Description	Information
0x14	Service UUIDs	List of 16 bit Service UUIDs
0x15	Service UUIDs	List of 128 bit Service UUID

图 2-51 SERVICE SOLICITATION 广播类型数据

- **SERVICE DATA**

这个还不太懂是什么东西。定义如图 2-52 所示。

Value	Description	Information
0x16	Service Data (2 or more octets)	The first 2 octets contain the 16 bit Service UUID followed by additional service data

图 2-52 SERVICE DATA 广播类型数据

- **MANUFACTURER SPECIFIC DATA**

厂商数据可以放产品生产厂商的数据。例如自己公司名字，放蓝牙地址等等。厂商数据定义如下图 2-53。

Value	Description	Information
0xFF	Manufacturer Specific Data (2 or more octets)	The first 2 octets contain the Company Identifier Code followed by additional manufacturer specific data

图 2-53 MANUFACTURER SPECIFIC DATA 广播数据定义

注意：

上面所有类型都可以放在广播数据或者广播扫描应答数据中。图 2-54 为 sniffer 采集的广播扫描应答数据。图 2-55 为 sniffer 采集的广播数据。图 2-56 位手机扫描到的广播信息。

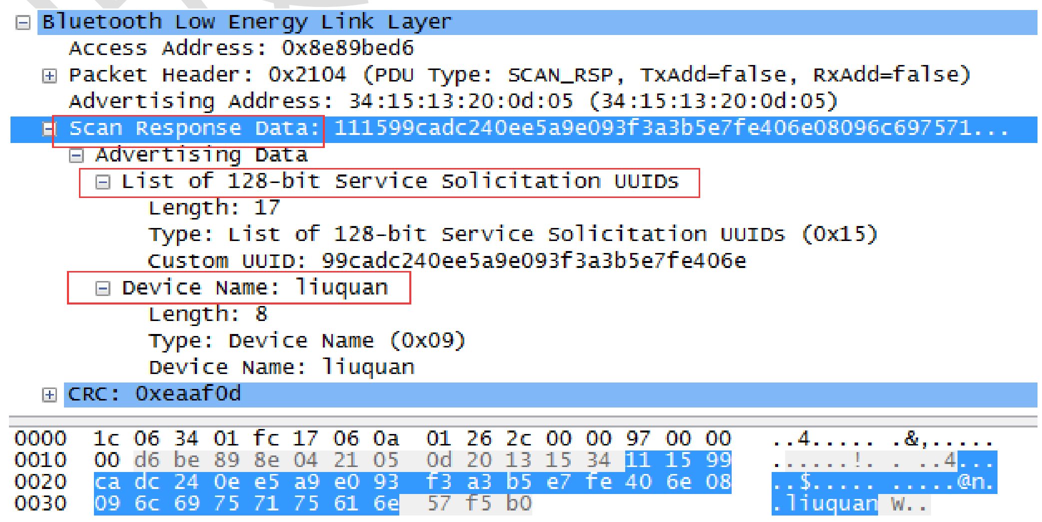


图 2-54 sniffer 采集的广播扫描应答数据

```
⊟ Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  ⊞ Packet Header: 0x2000 (PDU Type: ADV_IND, TxAdd=false, RxAdd=false)
    Advertising Address: 34:15:13:20:0d:05 (34:15:13:20:0d:05)
  ⊟ Advertising Data
    ⊟ Flags
        Length: 2
        Type: Flags (0x01)
        000. .... = Reserved: 0x00
        ...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x00)
        .... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x00)
        .... .1.. = BR/EDR Not Supported: true (0x01)
        .... ..1. = LE General Discoverable Mode: true (0x01)
        .... ...0 = LE Limited Discoverable Mode: false (0x00)
    ⊟ Tx Power Level
        Length: 2
        Type: Tx Power Level (0x0a)
        Power Level (dBm): 0
    ⊟ 16-bit Service Class UUIDs
        Length: 3
        Type: 16-bit Service Class UUIDs (0x03)
        UUID 16: Unknown (0xfee7)
    ⊟ Slave Connection Interval Range: 100 - 1000 msec
        Length: 5
        Type: Slave Connection Interval Range (0x12)
      Connection Interval Min: 80 (100 msec)
      Connection Interval Max: 800 (1000 msec)
    ⊟ Manufacturer Specific
        Length: 9
        Type: Manufacturer Specific (0xff)
        Company ID: Unknown (0x4c51)
      ⊞ Data: 341513200d05
  ⊞ CRC: 0x173da7

0000  1c 06 33 01 fa 17 06 0a  01 26 2c 00 00 88 02 00   ..3..... .&,.....
0010  00 d6 be 89 8e 00 20 05  0d 20 13 15 34 02 01 06   ...... . ..4...
0020  02 0a 00 03 03 e7 fe 05  12 50 00 20 03 09 ff 51   ........ .P. ...Q
0030  4c 34 15 13 20 0d 05 e8  bc e5                     L4.. ... ..
```

图 2-55 sniffer 采集的广播数据

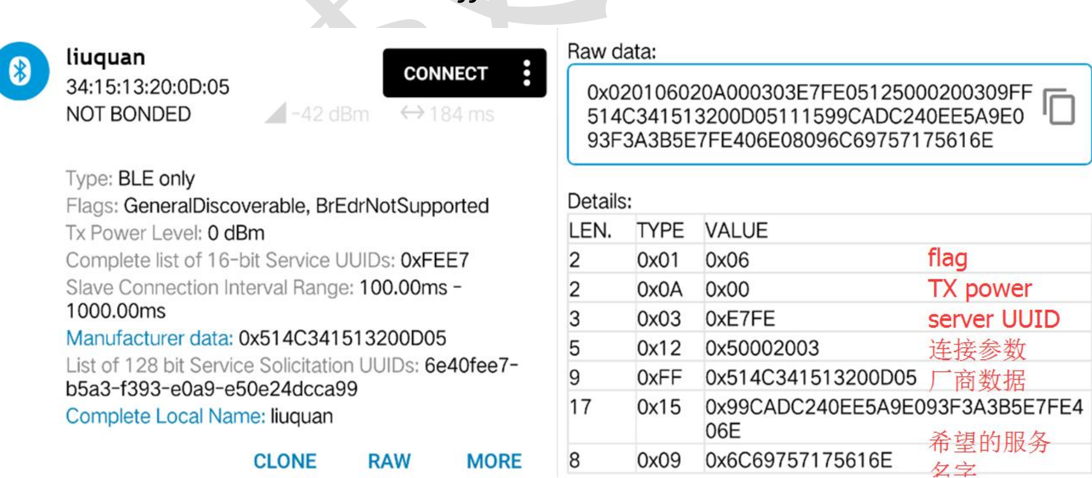


图 2-56 手机扫描到的广播和扫描应答数据

2.9.1.2、Advertising PDUs

2.9.1.2.1、ADV_IND、ADV_NONCONN_IND、ADV_SCAN_IND

上面的 3 种广播的 Payload 是一样的，如图 2-57。

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

图 2-57 ADV PDU Payload

前文中有提到,广播信道的 Payload 的首 6 个字节为,广播地址。它有报头中的 TxAdd 决定是公共地址还是随机地址。而其余的为 AdvData 广播数据，最多 31 个字节。广播数据是由广播者的 HOST 发送下来，其实这些数据是由 Generic Access Profile (GAP)也就是通用访问规范来设置的。说到这，我贴出下面的图 2-58。

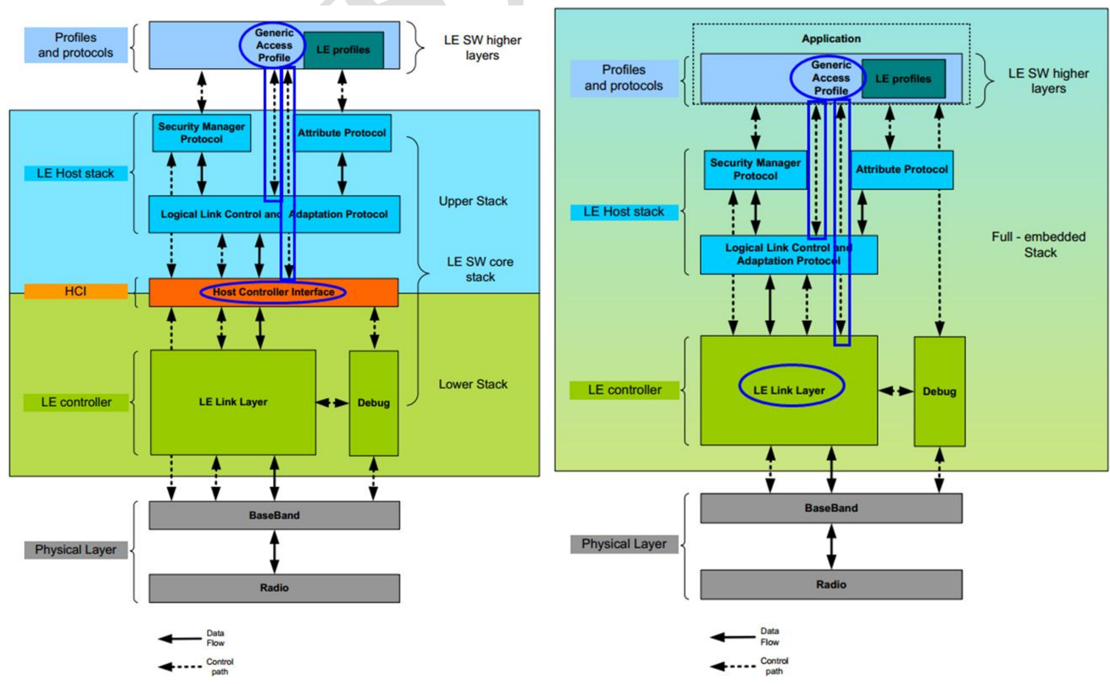


图 2-58 HOST 层数据到 CONTROLL 层的路径

在图 2-58 中，左边的为带有 HCI 的隔离式协议栈，右边是完整嵌入式协议栈。这里我主要想说明一个问题是，协议的每一层并不是像楼梯一样只能一步步的将数据向上或者向下传输，例如，GAP 其实差不多到了最高层了，但是他的数据有两种路径给底层，一种是经过 L2CAP 层，一种就是直接给到链路层，如果有 HCI 就直接给到了 HCI 接口。总之，不要局限于固定思维的以为，协议的每一层都是一层一层的将数据传输，有的层可能是平行的，有的层可以越过其他层到另外一层。

2.9.1.2.2、ADV_DIRECT_IND

这个报文只有两个地址，一个是广播者自己的设备地址还有一个就是希望连接的对方的设备地址，如图 2-59。

Payload	
AdvA (6 octets)	InitA (6 octets)

图 2-59 ADV_DIRECT_IND PDU Payload

2.9.1.3、Scanning PDUs

2.9.1.3.1、SCAN_REQ

这里不再累赘，和 ADV_DIRECT_IND 的包一样，如图 2-60。图 2-61 为采集空中的扫描请求包。

Payload	
ScanA (6 octets)	AdvA (6 octets)

图 2-60 SCAN_REQ PDU Payload

```
⊟ Bluetooth Low Energy
    Access Address: 0x8e89bed6
  ⊟ Packet Header
      1... .... = RX Address: random
      .1.. .... = TX Address: random
      .... 0011 = TYPE: SCAN_REQ (0x03)
      Length: 12
    Init Address: 72:e6:8e:bf:ff:5c (72:e6:8e:bf:ff:5c)
    Advertising Address: f8:54:3c:59:b2:81 (f8:54:3c:59:b2:81)
    CRC: 0x2b7782

0000  07 06 1f 01 39 0e 06 0a  01 27 32 00 00 96 00 00   ....9... .'2.....
0010  00 d6 be 89 8e c3 0c 5c  ff bf 8e e6 72 81 b2 59   .......\ ....r..Y
0020  3c 54 f8 2b 77 82                                  <T.+w.
```

图 2-61 Sniffer 采集空中扫描请求报文

2.9.1.3.2、SCAN_RSP

这个包和 ADV_IND 包一样，如图 2-62。 图 2-63 为空中采集的扫描应答包。

Payload	
AdvA (6 octets)	ScanRspData (0-31 octets)

图 2-62 SCAN_RSP PDU payload

在图 2-44 图中已经讲了 ScanRspData 的数据。

版权所有

```
⊟ Bluetooth Low Energy
    Access Address: 0x8e89bed6
  ⊟ Packet Header
      .1.. .... = TX Address: random
      .... 0100 = TYPE: SCAN_RSP (0x04)
      Length: 24
    Advertising Address: f8:54:3c:59:b2:81 (f8:54:3c:59:b2:81)
  ⊟ Scan Response Data: 11079ecadc240ee5a9e093f3a3b50100406e
    ⊟ 128 bit uuids (complete): 9ecadc240ee5a9e093f3a3b50100406e
        length: 0x11
        type: Complete List of 128-bit Service Class UUIDs (0x07)
        128 bit uuid: 6e400001b5a3f393e0a9e50e24dcca9e
    CRC: 0x179a3c

0000  07 06 2b 01 3a 0e 06 0a  01 27 49 00 00 97 00 00   ..+.:... .'I.....
0010  00 d6 be 89 8e 44 18 81  b2 59 3c 54 f8 11 07 9e   .....D.. .Y<T....
0020  ca dc 24 0e e5 a9 e0 93  f3 a3 b5 01 00 40 6e 17   ..$..... .....@n.
0030  9a 3c                                              .<
```

图 2-63 Sniffer 采集空中扫描应答报文

2.9.1.4、Initialing PDUS

2.9.1.4.1、CONNECT_REQ

连接请求包在 2.8.3 节中已经详细讲解，图 2-35 为连接请求包的包格式。

2.9.2、数据通道 PDU

数据报文的格式在 2.3 节中已经详细讲解。数据通道的报文分为两大类：链路层数据报文(LL Data PDU)和链路层控制报文。这里再将报头中的 LLID 字段截图如下图 2-64。

Field name	Description
LLID	The LLID indicates whether the packet is an LL Data PDU or an LL Control PDU. 00b = Reserved 01b = LL Data PDU: Continuation fragment of an L2CAP message, or an Empty PDU. 10b = LL Data PDU: Start of an L2CAP message or a complete L2CAP message with no fragmentation. 11b = LL Control PDU

图 2-64 数据 PDU 报头的 LLID 字段

2.9.2.1、LL Data PDU

链路层的数据通道是用来发送 L2CAP 的数据，也就是说只要是数据一定是从 HOST 层发送到下层的。而数据 PDU 的标示就是 LLID 为 01b 或者 10b。

- LLID=01b 这个数据要么是将 L2CAP 层的数据分解成多包发送的连续包，或者是一个空包。对于空包，数据 PDU 报头的长度域是需要设置成 00000b，而空包可被从机应答任何数据通道包。图 2-65 为空中采集的空包。
- LLID=10b 那么这包数据表示一个 L2CAP 层信息的起始包，当然如果 L2CAP 层的数据可以用链路层的一包数据发送完毕，即不需要分解成多个包发送，也是 10b 标示，相当于起始包一包就能将数据发送完毕。对于这个 LLID，数据 PDU 报头是的长度域是不能为 00000b 的。图 2-66 为空中采集的开始包。

版权所有

```
⊟ Bluetooth Low Energy
    Access Address: 0xaf9abd18
  ⊟ Data PDU Header: 0x0001
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .0.. = NESN: 0
      .... ..01 = LLID: Continuation fragment of an L2CAP message (1)
      Length: 0
    CRC: 0x8a3e75

0000  07 06 13 01 c4 18 06 0a  0b 1e 38 7c 02 fa 73 00   ........ ..8|..s.
0010  00 18 bd 9a af 01 00 8a  3e 75                     ........ >u
```

图 2-65 Sniffer 采集空中数据通道空包报文

```
⊟ Bluetooth Low Energy
    Access Address: 0xaf9abd18
  ⊟ Data PDU Header: 0x0d0e
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 1... = SN: 1
      .... .1.. = NESN: 1
      .... ..10 = LLID: Start of an L2CAP message or no fragmentation (2)
      Length: 13
    CRC: 0xe3e02f

0000  07 06 20 01 a6 18 06 0a  03 14 3a 6d 02 f8 73 00   .. ..... ..:m..s.
0010  00 18 bd 9a af 0e 0d 09  00 04 00 52 11 00 52 65   ........ ...R..Re
0020  77 69 6e 64 e3 e0 2f                               wind../
```

图 2-66 Sniffer 采集空中数据通道数据开始或非分裂报文

2.9.2.2、LL Control PDU

对于链路层的控制，LLID=11b。它的有效数据段还分为了两个区域，如图 2-67。

Payload	
Opcode (1 octet)	CtrData (0 – 22 octets)

图 2-67 LL control PDU payload

第一个字节为操作码，对于 BLE 共有 14 个控制操作，如图 2-68。

Opcode	Control PDU Name
0x00	LL_CONNECTION_UPDATE_REQ
0x01	LL_CHANNEL_MAP_REQ
0x02	LL_TERMINATE_IND
0x03	LL_ENC_REQ
0x04	LL_ENC_RSP
0x05	LL_START_ENC_REQ
0x06	LL_START_ENC_RSP
0x07	LL_UNKNOWN_RSP
0x08	LL_FEATURE_REQ
0x09	LL_FEATURE_RSP
0x0A	LL_PAUSE_ENC_REQ
0x0B	LL_PAUSE_ENC_RSP
0x0C	LL_VERSION_IND
0x0D	LL_REJECT_IND
0x0E-0xFF	Reserved for Future Use

图 2-68 LL Control PDU Opcodes

版权

2.9.2.2.1、LL_CONNECTION_UPDATE_REQ

连接参数第一次是主机发送的 CONNECT_REQ 中传递的。而这个命令的使用只限主机使用，也就是说主机根据需要随时都可以进行参数更新。从机接收到这个命令后要么使用参数，要么断开连接。然而从机如果需要更新连接参数呢？见 3.1.3.2 节。这个更新请求包的格式如图 2-69。

CtrData					
WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	Instant (2 octets)

图 2-69 CtrData field of the LL_CONNECTION_UPDATE_REQ PDU

这里不再一一讲解每个参数，详细见图 2-36。重点讲一下最后一个参数：Instant。

在连接更新参数时，并不是请求发送过去，这些参数就立刻生效，是在约定的时刻进行更新。在 2.8.1 节中有说到，BLE 的同步是通过一个连接事件计数器控制的。而这个数据包中的 Instant 参数就是一个未来的连接事件计数器的次数。当目前的连接事件的次数等于 Instant 的值的时候，连接参数开始投入使用，而对于 Instant 这次的连接事件就相当于一个新的连接建立的过程，只是这个过程都是在数据通道中完成。图 2-70 为实现过程示意图。

版权所有

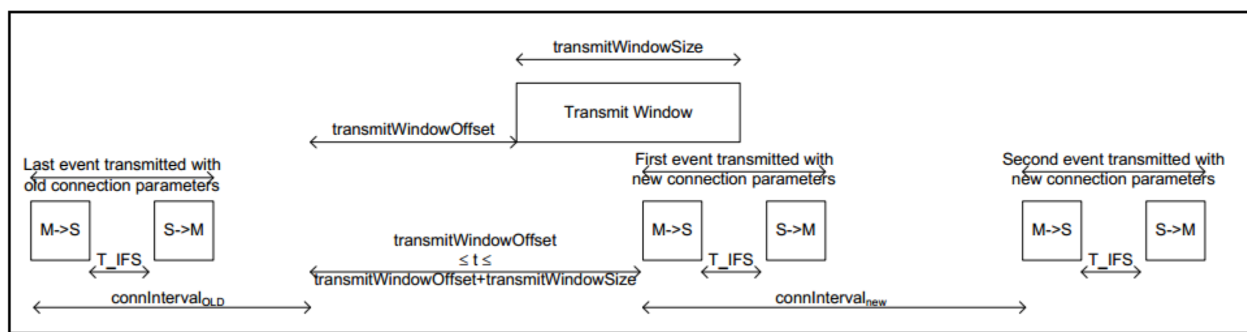


图 2-70 connection parameter update

图 2-71 为空中采集的参数更新包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x0c0f
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 1... = SN: 1
      .... .1.. = NESN: 1
      .... ..11 = LLID: LL Control PDU (3)
      Length: 12
  ⊟ LL Control PDU: LL_CONNECTION_UPDATE_REQ (0x00)
      LL Control Opcode: LL_CONNECTION_UPDATE_REQ (0x00)
      Window Size (ms): 3.75
      Window Offset (ms): 533.75
      Interval (ms): 997.5
      Latency: 0
      Timeout (ms): 4000
      Instant field: 0x00b4
    CRC: 0x168128

0000  07 06 1f 01 83 15 06 0a  3f 1f 36 a8 00 79 73 00   ........ ?.6..ys.
0010  00 d5 45 65 50 0f 0c 00  03 ab 01 1e 03 00 00 90   ..EeP... ........
0020  01 b4 00 16 81 28                                  .....(
```

图 2-71 Sniffer 采集空中连接参数更新报文

2.9.2.2.2、LL_CHANNEL_MAP_REQ

通道图第一次也是主机发送的 CONNECT_REQ 中传递的。包格式如图 2-72。

CtrData	
ChM (5 octets)	Instant (2 octets)

图 2-72 CtrData field of the LL_CHANNEL_MAP_REQ PDU

这个包中的 Instant 的意思和参数更新包中的意思是一样的。图 2-73 为空中采集的通道更新包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x50656b65
  ⊟ Data PDU Header: 0x0807
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .1.. = NESN: 1
      .... ..11 = LLID: LL Control PDU (3)
      Length: 8
  ⊟ LL Control PDU: LL_CHANNEL_MAP_REQ (0x01)
      LL Control Opcode: LL_CHANNEL_MAP_REQ (0x01)
    ⊟ Channel map: 00f8ffff1f
        Enabled channels:  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  , 11,
      Instant field: 0x00de
    CRC: 0x527db5

0000  04 06 1b 01 79 03 06 0a  01 1e 51 d4 00 98 00 00   ....y... ..Q.....
0010  00 65 6b 65 50 07 08 01  00 f8 ff ff 1f de 00 52   .ekeP... .......R
0020  7d b5                                              }.
```

图 2-73 *Sniffer 采集空中通道图更新报文*

2.9.2.2.3、LL_TERMINATE_IND

终止连接，就是将设备进入就绪态，任何一方可以给予任何原因在任何时候终止连接。要终止连接，设备首先发送一个终止报文，并等待对方进行报文确定。设备接收到终止连接报文后发送一个空包作为命令的确认，这样双方都进入就绪态。当然如果并没有发送空包进行确认呢？发送方将等待监管超时的时间，如果超时会自动断开连接。

还有两个原因会使连接终止：

- 监管超时

- MIC 失效

遇到上面的两个原因，不会发终止连接报文；链路层自动断开，双方主机随机得到通知。

连接终止报文如图 2-74。

CtrData
Error Code (1 octet)

图 2-74 CtrData field of the LL_TERMINATE_IND PDU

终止原因可多了，在 4.0 协议中有专门一个章节介绍错误原因。这里就不贴图，到 4.0 规范中的第 2 章节的 Part D-Error Codes 查看。

图 2-65 空中采集包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x0203
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .0.. = NESN: 0
      .... ..11 = LLID: LL Control PDU (3)
      Length: 2
  ⊟ LL Control PDU: LL_TERMINATE_IND (0x02)
      LL Control Opcode: LL_TERMINATE_IND (0x02)
      Error code: Remote User Terminated Connection (0x13)
    CRC: 0x5d78e9

0000  07 06 15 01 31 16 06 0a  3f 16 38 ff 00 27 37 0f   ....1... ?.8..'7.
0010  00 d5 45 65 50 03 02 02  13 5d 78 e9               ..EeP... .]x.
```

图 2-75 Sniffer 采集空中终止连接报文

2.9.2.2.4、LL_ENC_REQ

这是加密请求包，数据域如图 2-76。

CtrData			
Rand (8 octets)	EDIV (2 octets)	SKDm (8 octets)	IVm (4 octets)

图 2-76 CtrData field of the LL_ENC_REQ PDU

加密还没有涉及，之后再进行补充。.......

图 2-77 为空中采集包。

```
Bluetooth Low Energy
  Access Address: 0x506545d5
  Data PDU Header: 0x170f
    000. .... = RFU: 0
    ...0 .... = MD: 0
    .... 1... = SN: 1
    .... .1.. = NESN: 1
    .... ..11 = LLID: LL Control PDU (3)
    Length: 23
  LL Control PDU: LL_ENC_REQ (0x03)
    LL Control Opcode: LL_ENC_REQ (0x03)
    Rand: 0000000000000000
    EDIV: 0000
    SKDm: c7498d11c47cb05e
    IVm: 1f88a5ae
  CRC: 0x72682c

0000  07 06 2a 01 d7 14 06 0a  03 12 32 54 00 51 73 00   ..*..... ..2T.Qs.
0010  00 d5 45 65 50 0f 17 03  00 00 00 00 00 00 00 00   ..EeP... ........
0020  00 00 c7 49 8d 11 c4 7c  b0 5e 1f 88 a5 ae 72 68   ...I...| .^....rh
0030  2c                                                 ,
```

图 2-77 Sniffer 采集空中开始加密请求报文

2.9.2.2.5、LL_ENC_RSP

加密应答包，如图 2-78。

CtrData	
SKDs (8 octets)	IVs (4 octets)

图 2-78 CtrData field of the LL_ENC_ RSP PDU

加密还没有涉及，之后再进行补充。……

图 2-79 为空中采集包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x0d07
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .1.. = NESN: 1
      .... ..11 = LLID: LL Control PDU (3)
      Length: 13
  ⊟ LL Control PDU: LL_ENC_RSP (0x04)
      LL Control Opcode: LL_ENC_RSP (0x04)
      SKDs: 5fb1c066d8195091
      IVs: 15f7f875
    CRC: 0x4952ec

0000  07 06 20 01 da 14 06 0a  01 17 47 55 00 97 00 00   .. ..... ..GU....
0010  00 d5 45 65 50 07 0d 04  5f b1 c0 66 d8 19 50 91   ..EeP... _..f..P.
0020  15 f7 f8 75 49 52 ec                               ...uIR.
```

图 2-79 Sniffer 采集空中开始加密应答报文

2.9.2.2.6、LL_START_ENC_REQ

开始加密请求。这个报文没有数据域。图 2-80 为空中采集包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x0107
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .1.. = NESN: 1
      .... ..11 = LLID: LL Control PDU (3)
      Length: 1
  ⊟ LL Control PDU: LL_START_ENC_REQ (0x05)
      LL Control Opcode: LL_START_ENC_REQ (0x05)
    CRC: 0x03faf5

0000  07 06 14 01 de 14 06 0a  01 21 47 57 00 97 00 00   ........ .!GW....
0010  00 d5 45 65 50 07 01 05  03 fa f5                  ..EeP... ...
```

图 2-80 Sniffer 采集空中开始加密应答报文

2.9.2.2.7、LL_START_ENC_RSP

开始加密应答。这个报文没有数据域。图 2-81 为空中采集包。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x010f
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 1... = SN: 1
      .... .1.. = NESN: 1
      .... ..11 = LLID: LL Control PDU (3)
      Length: 1
  ⊟ LL Control PDU: LL_START_ENC_RSP (0x06)
      LL Control Opcode: LL_START_ENC_RSP (0x06)
    CRC: 0x8505be

0000  07 06 14 01 df 14 06 0a  3f 01 30 58 00 11 74 00   ........ ?.0X..t.
0010  00 d5 45 65 50 0f 01 06  85 05 be                  ..EeP... ...
```

图 2-81 Sniffer 采集空中开始加密应答报文

2.9.2.2.8、LL_UNKNOWN_RSP

这个命令是应答刚刚接收的包，告诉对方这个包不能识别的命令。返回值为不能执行或者识别的命令和数据。如图 2-82。

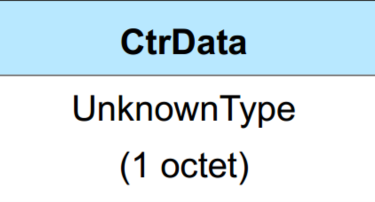


图 2-82 CtrData field of the LL_UNKNOWN_RSP PDU

一个字节的错误类型为刚刚接收的不认识的命名值。

2.9.2.2.9、LL_FEATURE_REQ

特征请求是由主机发送给从机的，如图 2-83 为 8 个字节的数据域，这 8 个字节每一个 bit 表示一个特征。对于 BLE4.0 协议中的特征

只有 1 个 bit 位是有效的，其余的全是保留。如图 2-84 所示。1 表示支持加密，0 表示不支持加密。

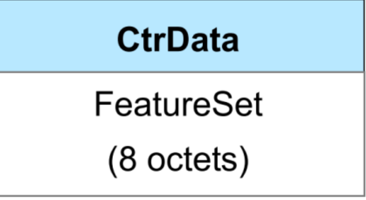


图 2-83 CtrData field of the LL_FEATURE_REQ/RSP PDU

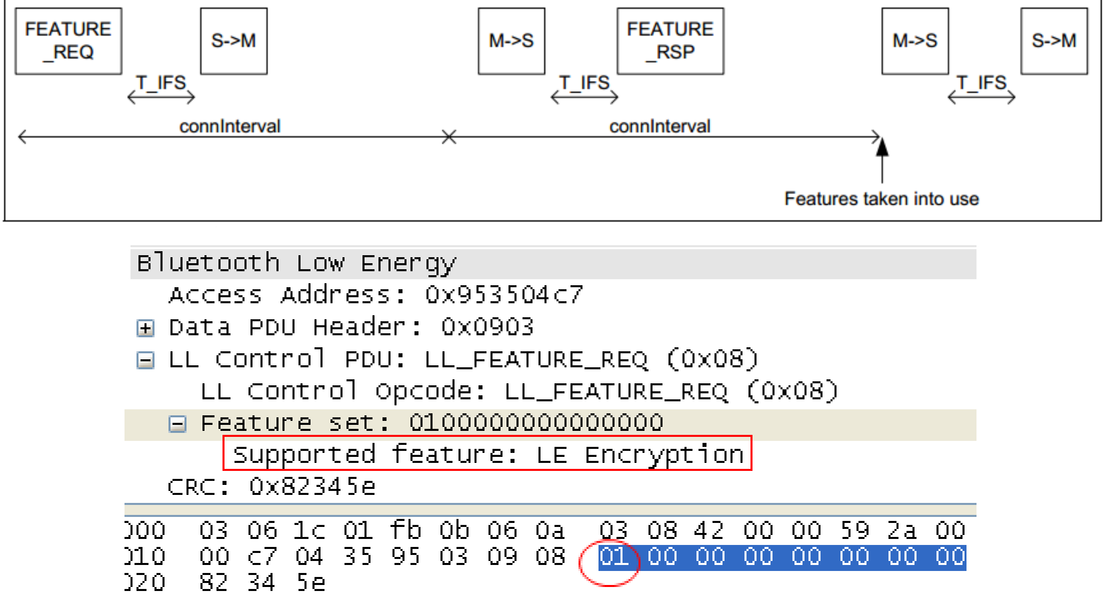
Bit position	Link Layer Feature	Valid from Controller to Host	Valid from Host to Controller	Valid from Controller to Controller
0	LE Encryption	Y 控制层传主机有效	Y 主机层传控制层有效	Y 控制层传控制层有效
1 – 63	RFU			

图 2-84 FeatureSet 区域位控制映射图

2.9.2.2.10、LL_FEATURE_RSP

特征应答是从机发送给主机的应答包，数据域和 *LL_FEATURE_REQ* 的一样。数据域的 8 个字节为从机本身的特征。

特征交换的模拟过程以及 Sniffer 空中采集的数据包如图 2-85。



```
Bluetooth Low Energy
  Access Address: 0x953504c7
⊞ Data PDU Header: 0x090b
⊟ LL Control PDU: LL_FEATURE_RSP (0x09)
    LL Control Opcode: LL_FEATURE_RSP (0x09)
  ⊟ Feature set: 0100000000000000
      Supported feature: LE Encryption
  CRC: 0xf6ace8

)00  03 06 1c 01 fd 0b 06 0a  01 10 57 01 00 97 00 00
)10  00 c7 04 35 95 0b 09 09  01 00 00 00 00 00 00 00
)20  f6 ac e8
```

图 2-85 特征交换过程

2.9.2.2.11、LL_PAUSE_ENC_REQ

暂停加密请求。这个报文没有数据域。

2.9.2.2.12、LL_PAUSE_ENC_RSP

暂停加密应答。这个报文没有数据域。

2.9.2.2.13、LL_VERSION_IND

版本交换命令数据域如图 2-86。

CtrData		
VersNr (1 octet)	CompId (2 octets)	SubVersNr (2 octets)

图 2-76 CtrData field of the LL_VERSION_IND PDU

这个数据域包含有 3 个区域，分别为：

- 版本号(VersNr)

它是为设备兼容的蓝牙规范版本所分配的编号。例如对于 BLE4.0 的编号就是 0x06。

- 公式标识符(CompId)

它是由蓝牙技术联盟(SIG)为生产这款控制器的公司所指定的代码。

- 子版本号(SubVersNr)

这个参数是生产厂商指定的一串数字，每一次实现或者修订控制器，子版本号都要进行相应的改变。

对于版本交换，在一次连接过程中最多进行一次版本交换，多次无效。这个命令可由任意一方发送。

如下图为nrf51822和iPhone4s之间的版本交换信息。其中图 2-87为4s作为主机发送的版本信息。图 2-88为nrf51822的版本信息。

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x0603
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 0... = SN: 0
      .... .0.. = NESN: 0
      .... ..11 = LLID: LL Control PDU (3)
      Length: 6
  ⊟ LL Control PDU: LL_VERSION_IND (0x0c)
      LL Control Opcode: LL_VERSION_IND (0x0c)
      Bletooth version: 6
      Company ID: 15
      Sub-version number: 16643
    CRC: 0xa6ea7a

0000  07 06 19 01 2b 14 06 0a  03 05 32 00 00 c5 41 00   ....+... ..2...A.
0010  00 d5 45 65 50 03 06 0c  06 0f 00 03 41 a6 ea 7a   ..EeP... ....A..z
```

图 2-87 Sniffer 采集空中 iphone4s 版本报文

```
⊟ Bluetooth Low Energy
    Access Address: 0x506545d5
  ⊟ Data PDU Header: 0x061b
      000. .... = RFU: 0
      ...1 .... = MD: 1
      .... 1... = SN: 1
      .... .0.. = NESN: 0
      .... ..11 = LLID: LL Control PDU (3)
      Length: 6
  ⊟ LL Control PDU: LL_VERSION_IND (0x0c)
      LL Control Opcode: LL_VERSION_IND (0x0c)
      Bletooth version: 6
      Company ID: 89
      Sub-version number: 73
    CRC: 0xf4af22

0000  07 06 19 01 2e 14 06 0a  01 0a 45 01 00 97 00 00   ........ ..E.....
0010  00 d5 45 65 50 1b 06 0c  06 59 00 49 00 f4 af 22   ..EeP... .Y.I..."
```

图 2-88 Sniffer 采集空中 nrf51822 版本报文

2.9.2.2.14、LL_REJECT_IND

图 2-89 为拒绝报文。

CtrData
Error Code (1 octet)

图 2-89 CtrData field of the LL_ REJECT_IND

错误编码同样也在 4.0 规范中的第 2 章节的 Part D-Error Codes 查看。

图 2-90 为采集的加密请求中的命令拒绝回复。

```
⊟ Bluetooth Low Energy
    Access Address: 0x50654267
  ⊟ Data PDU Header: 0x020b
      000. .... = RFU: 0
      ...0 .... = MD: 0
      .... 1... = SN: 1
      .... .0.. = NESN: 0
      .... ..11 = LLID: LL Control PDU (3)
      Length: 2
  ⊟ LL Control PDU: LL_REJECT_IND (0x0d)
      LL Control Opcode: LL_REJECT_IND (0x0d)
      Error code: PIN or Key Missing (0x06)
    CRC: 0xbdbb7c

0000  04 06 15 01 0a 4f 06 0a  01 24 48 05 00 97 00 00   .....O.. .$H.....
0010  00 67 42 65 50 0b 02 0d  06 bd bb 7c               .gBeP... ...|
```

图 2-90 Sniffer 采集空中命令拒绝报文

版权所有

2.9.3、连接态的数据包确认和重发以及多数据发送标志

在数据通道的报头中有 4 个字段，对于 LLID 字段已经在 2.9.2 节中讲过，那么还有 3 个字段或者说 3 个标志位将在本节分析。

在通信过程中怎么确定对方收到我发送的包呢？同时我接收包怎么确定是对方发送的新包还是重发的包呢？

毕竟是全球顶尖的人才设计的协议，在 BLE4.0 中仅仅采用了 2 个 bit 位就将确定和重发搞定。在 2.3.3 节中的表 2_3 中已经提到过报头的几个字段。这里针对其中的 3 个标志位进行说明。如图 2-91。

NESN	Next Expected Sequence Number
SN	Sequence Number
MD	More Data

图 2-91 NESN、SN 及 MD 标志位

2.9.3.1、序列号(SN)

这个标志位序列号。它是一个 bit 位，所以值是在 0 和 1 之间进行切换。为了是数据传输变得可靠，所有的数据报文都带有序列号。连接建立后，第一个数据包的序列号为 0；每次发送新的数据包时，其序列号与上个数据包的序列号不同。这使得接收装置能够判断接收的数据包的性质：如果序列号与之前的一样，则为重传报文，如果序列号和之前的不同则为新的报文。

2.9.3.2、预期序列号(NESN)

它是接收方希望接到的下一包的序列号，及逾期序列号，也是数据包的确认标志。NESN 的发送方用其通知对方自己期望接收的数据包的序列号。

当设备接收到序列(SN)为 0 的报文后，在发送给对方的数据包中，应将 NESN 设为 1，这样对方接收到这个包后，会发送一个新的数据包过来，否则就会重发上一次序列号为 0 的包。这个标志可以用来判断数据包是否被正确接收还是需要重传。

2.9.3.3、更多数据(MD)

这个标志位是用来通知对方设备自己还有其他数据准备发送。0 表示没有更多数据发送，1 标志有更多数据准备发送。如果收到标志为 1，即有更多数据需要发送，应当在当前连接事件中继续与对端设备通信。这样一来，只要还有数据要发送，连接事件会自动扩展。一旦不再有数据发送，连接事件立刻关闭。

2.9.3.4、SN、NESN 和 MD 应用的例子

如图 2-92 中讲述了一个关于序列号、下一个期望序列号和更多数据位的例子。

版权所有

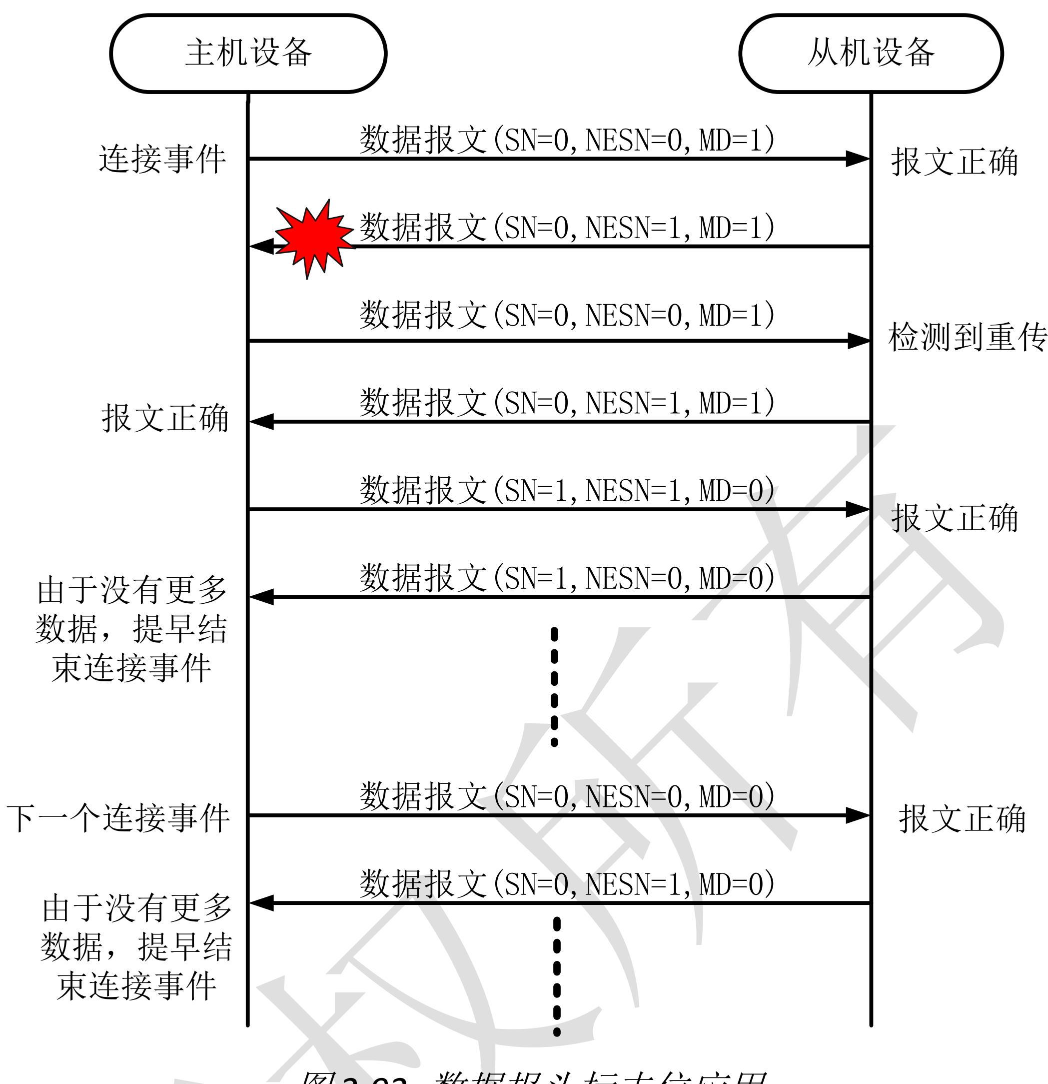


图 2-92 数据报头标志位应用

版权所有

1、主设备发送第一个数据包，默认的序列号为 0，下一个期望的序列号也为 0；同时由于有两个数据包要发送，更多数据位设为 1($SN_{master}=0$，$NESN_{master}=0$，$MD_{master}=1$)。从设备正确接收了该数据，更新下一个预期序列号($NESN_{slave}=1$)。

2、更新了下一个预期序列号($NESN_{slave}=1$)之后，从机发送它的第一个数据包($SN_{slave}=0$)。因为从设备还有一些数据要发送，该数据包也对更多数据位进行了设置($MD_{slave}=1$)。然而，主设备没有接收到这个包，所以主设备的下一个预期的序列号没有改变。从设备还将继续侦听主设备，因为二者都设置了更多数据位。

3、主设备再次发送第一个数据包($SN_{master}=0$，$NESN_{master}=0$，$MD_{master}=1$)，这是因为主设备没有收到从设备的数据包，因此必须重传上一个包。从设备接收该数据包后发现序列号和上一个包相同，判断是重传报文，于是不更新下一个预期序列号。由于看到主设备还要发送更多数据，从设备在当前连接事件里继续发送其他报文。

4、从设备重新发送它的第一个数据包($SN_{slave}=0$，$NESN_{slave}=0$，$MD_{slave}=1$)。主设备成功接收了该报文，于是更新它的下一个期望序列号($NESN_{slave}=1$)。主机发现从设备还有更多数据等待发送，于是发送其他报文继续该连接事件。

5、主设备第 3 次传输的是一个新的数据包，使用了一个新的序列号($SN_{master}=1$，$NESN_{master}=1$，$MD_{master}=0$)。由于该数据包含有所有的剩余数据，主机把更多数据位设置为 0。从设备在成功接收该数据包后更新它的下一个期望序列号($NESN_{slave}=0$)。此时，由于从设备任然有数据要发送，因此将继续连接事件。

6、从设备第 3 次传输的是一个新的数据包，使用了一个新的序列号($SN_{slave}=1$，$NESN_{slave}=0$，$MD_{slave}=0$)。从设备将更多数据位设为 0，表明没有其他数据需要发送。主设备在正确接收该数据包后更新它的下一个期望序列号($NESN_{master}=0$)。因为主从设备的最后一个数据包都表示没有更多数据，连接事件立刻关闭。

7、一段时间后，主设备在下一个连接事件醒来，此时从设备发送新的数据包，包含默认的新序列号和最后更新的下一个预期的序列号($SN_{master}=0$，$NESN_{master}=0$，$MD_{master}=0$)。这个包同时确认了从设备的

版权所有

上一个数据包。从设备成功接收了该数据包，并更新它的下一个期望的序列号($NESN_{master}=1$)。虽然没有要发送的数据，从设备仍然发送一个空包以做出响应。

8、从设备第 4 次传输的是一个空包，使用一个新的序列号($SN_{slave}=0$，$NESN_{slave}=1$，$MD_{slave}=0$)。由于没有更多的数据要发送，从设备将其更多的数据位设为 0。主机在成功接收该数据后更新它的下一个预期序列号($NESN_{slave}=1$)。因为主从设备的最后一个数据包都表示没有更多的数据，连接事件随即关闭。

在这提一下什么情况叫做数据包正确接收：CRC 校验值用于检验数据包，如果内容校验失败，数据包不算正确接收。

对于 NESN 还有一个作用是可以进行流量控制。设备一旦没有足够的缓冲区空间来处理消息，可以不更新下一个期望序列号。这将迫使对端设备重发当前消息，接收到的重发信息忽略。

对于连接事件而言还有一个注意事项。如果由于一些位的错误导致了 CRC 校验失败，数据包接收出错，这种情况在同一个连接事件里一旦发生 2 次，设备将立刻停止当前事件，并在下一次连接事件中重新同步和尝试传输。这样一来，如果某信道因干扰产生了拥堵，两设备很快能发现干扰，并停止使用该信道，下一个连接事件到来时将更新新的信道，干扰随即减轻，数据又可以快速地传给对方。

在协议中的原文如下

A Link Layer may not update nextExpectedSeqNum for reasons, including, but not limited to, lack of receive buffer space. This will cause the peer to resend the Data Channel PDU at a later time, thus enabling flow control.

版权所有

2.9.3.5、确认和重发的软件实现

这个其实困扰了好久，因为 4.0 规范中的原文这样写的

For each new Data Channel PDU that is sent, the SN bit of the Header shall be set to *transmitSeqNum*. If a Data Channel PDU is resent, then the SN bit shall not be changed.
Upon reception of a Data Channel PDU, the SN bit shall be compared to *nextExpectedSeqNum*. If the bits are different, then this is a resent Data Channel PDU, and *nextExpectedSeqNum* shall not be changed. If the bits are the same, then this is a new Data Channel PDU, and *nextExpectedSeqNum* may be incremented by one.
When a Data Channel PDU is sent, the NESN bit of the Header shall be set to *nextExpectedSeqNum*.
Upon receiving a Data Channel PDU, if the NESN bit of that Data Channel PDU is the same as *transmitSeqNum*, then the last sent Data Channel PDU has not been acknowledged and shall be resent. If the NESN bit of the Data Channel PDU is different from *transmitSeqNum*, then the last sent Data Channel PDU has been acknowledged, *transmitSeqNum* shall be incremented by one, and a new Data Channel PDU may be sent.

上面的红色的两个条件让我很是不理解，这两个条件貌似是一样。规范中还有一个流程图如下图 2-93。

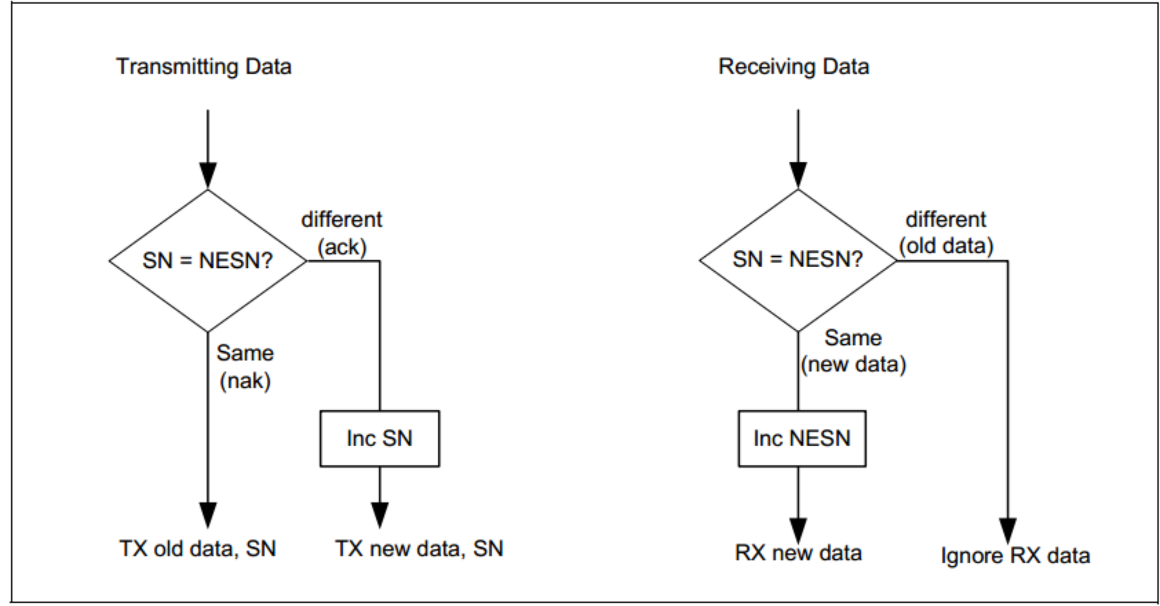


图 2-93 Transmit and Receive SN and NESN flow diagram

流程图和原文根本对应不上啊！所以我也是纠结了很久！才貌似写正确了这个程序。如图 2-94。

```
/*******************判断接收的包是不是对方发送的新包*******************/
if( rtx_data_pdu_header.Sequence_Number_Flag ==
    rtx_data_pdu_header.Next_Expected_SN_Flag )//刚接收的数据不是对方重发包 新包上报上层 否则丢弃
{
  New_Data_receive_Flag  =1;   //新数据接收标志, 0为接收对方重传的数据  1为 接收的对方发的新数据包
  rtx_data_pdu_header.Next_Expected_SN_Flag += 1; //新数据 下一个期望包+1 ;
  rtx_data_pdu_header.Next_Expected_SN_Flag &= 0x01;//  期望得到下一序列数据

  //发送的包是新包
  Data_retransmission_Flag  =1;   //数据重传标志,  0为重传            1为 发新数据包
  Radio_Packetptr.TX_Data_Prepare_Packetptr[0] &= 0xf3;  //低位 11001111 高位
  Radio_Packetptr.TX_Data_Prepare_Packetptr[0] |= (rtx_data_pdu_header.Sequence_Number_Flag<<3);
  Radio_Packetptr.TX_Data_Prepare_Packetptr[0] |= (rtx_data_pdu_header.Next_Expected_SN_Flag<<2);

  LQ_Memcpy(Radio_Packetptr.RTX_Radio_Packetptr,Radio_Packetptr.TX_Data_Prepare_Packetptr,
          Radio_Packetptr.TX_Data_Prepare_Packetptr[1]+PHY_HEADER_SIZE); //内存复制

}
else
{
   New_Data_receive_Flag  =0;   //新数据接收标志, 0为接收对方重传的数据 1为 接收的对方发的新数据包

  //重传
  Data_retransmission_Flag  =0;   //数据重传标志,  0为重传            1为 发新数据包
  LQ_Memcpy(Radio_Packetptr.RTX_Radio_Packetptr,Radio_Packetptr.Copy_TX_Data_Prepare_Packetptr ,
          Radio_Packetptr.Copy_TX_Data_Prepare_Packetptr[1]+PHY_HEADER_SIZE); //内存复制

}
```

图 2-94 SN 和 NESN 标志位软件设计

结合图 2-93 分析，接收到数据 CRC 校验正确后，判断 SN 和 NESN 是否相等。

- 如果相等说明接收到的是新的数据包，NESN 标志位加 1，并且准备发送新的数据包，发送的新数据包中的 NESN 位为刚刚加 1 后的值，而 SN 的值为刚刚接收包中的 SN 值一样，其他的标志位和数据是在上一个连接间隔中已经准备好了的。
- 如果不相等说明接收到的是重传的数据包，那么发送时也应该发送上一次发送的数据包。这里直接通过 Memcpy 函数将两个指针所指的内存进行指定长度的拷贝，其中被拷贝的对象的内容是在每次发送数据后，将发送的内容复制到其中，也就是说 Copy_TX_Data_Prepare_Packetptr 指针所指的内容就是上一次发送的数据。

版权所

2.10、直接测试单元(DTU)

对于 PHY 的测试，共有两种方法，一种的通过 HCI 或者是一个 2-wire UART 接口进行 RF 的测试（这里只讲 UART 接口测试）。直接测试模式需要 3 个设备(如图 2-95)。

- 待测设备 Device Under Test (DUT)
- 上位测试设备(UT)
- 下位测试设备(LT)

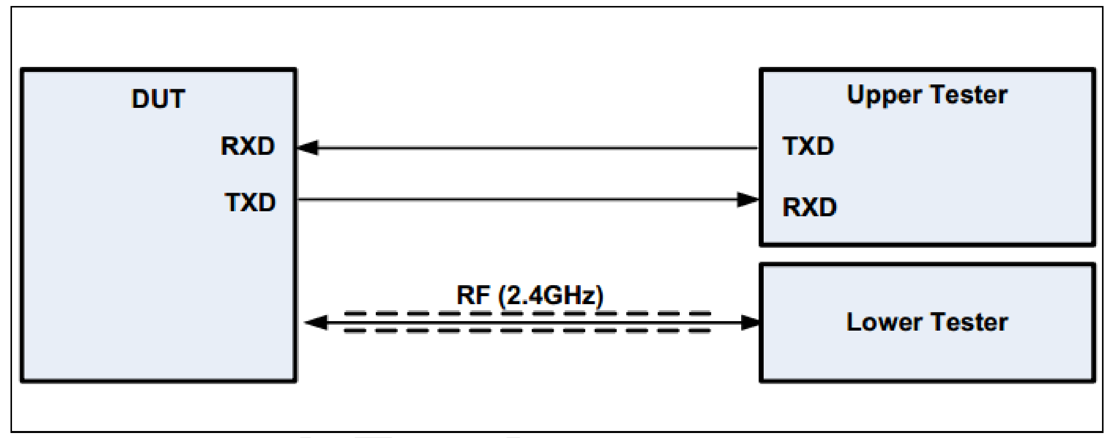


图 2-95 RF PHY test setup for Direct Test Mode (UART control)

2.10.1、UART 测试接口

串口测试接口的特征：

- 波特率

1200、2400、9600、14400、19200、38400、57600、115200 可选，一般采用 38400。

- 数据位数：8bit
- 无奇偶校验

- 1 位停止位
- 无流控

在发送命令和事件时，2 字节数据之间最大的时间间隔是 5ms。一个命令发送后，相应的事件必须在 50ms 内返回。这样为了保证待测设备能够及时开始和停止测试，进而保证计数的准确性。

2.10.2、测试模式 RADIO 配置

- 测试报文

测试报文与广播报文的格式一样(表 2-2)，前导码、接入地址、报头及报文长度，测试报文的前 4bit 表示报文类型，其余 4bit 为 0，报文类型见表 2-8。**这里注意，这节讲的报文和 2.10.5 命令和事件不同，测试报文指 DUT 发送到空中的数据包格式，而 2.10.5 节讲的是串口命令和事件。**长度也为报文长度的低 6bit。

- Radio 的模式

对于 nrf51822 芯片 Radio 配置为 Ble_1Mbit 模式。

- 白化

为了更好的测试，规定是发送不经过白发的数据报文。

- CRC

CRC 的生成项依然是 $x^{24} + x^{10} + x^{9} + x^{6} + x^{4} + x^{3} + x^{1} + x^{0}$，对于 nrf51822 就是设置 CRC 的相关寄存器为 0x0000065B。CRC 移位寄存器的初始值为 0x00555555。

- 接入地址

版权所有

测试模式的接入地址就是广播状态的地址按位求反的值，广播状态的接入地址为 0x8e89bed6， 二进制为：

1000　1110　1000　1001　1011　1110　1101　0110b

求反：0111　0001　0111　0110　0100　0001　0010　1001b

即：0x　7　1　7　6　4　1　2　9

0x71764129 即为测试状态下的接入地址。

2.10.3、发射机测试

发射机测试是为了获知待测设备发射机的性能。这一测试能得到频偏、频率漂移，以及其他规定的射频参数。图 2-96 为发射机测试的消息序列图(MESSAGE SEQUENCE CHARTS)。这里对于所有测试报文，都不进行白化处理，为了精确测量非随机比特序列的频偏。

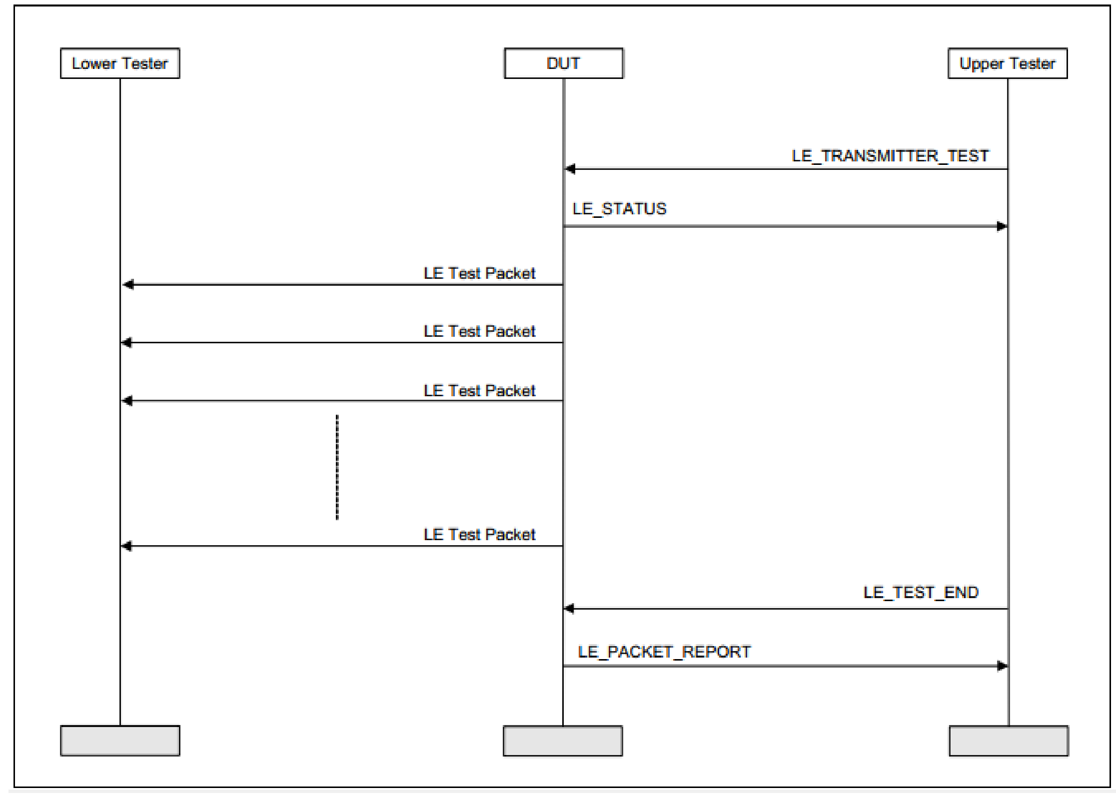


图 2-96 Transmitter Test MSC

2.10.4、接收机测试

接收机测试的目的是确定不同发射功率条件下接收机的误比特率。一般情况下，下位测试设备以已知的功率发射，待测设备接收，二者之间用射频线连接，从而避免不确定的路径损耗带来的影响。待测设备对正确接收的报文进行计数，并在测试结束时将这一信息上报上位测试设备。上位测试设备便可确定错误报文数量，从而估计特定信号强度情况下接收机的误比特率。图 2-97 为接收机测试的消息序列图。

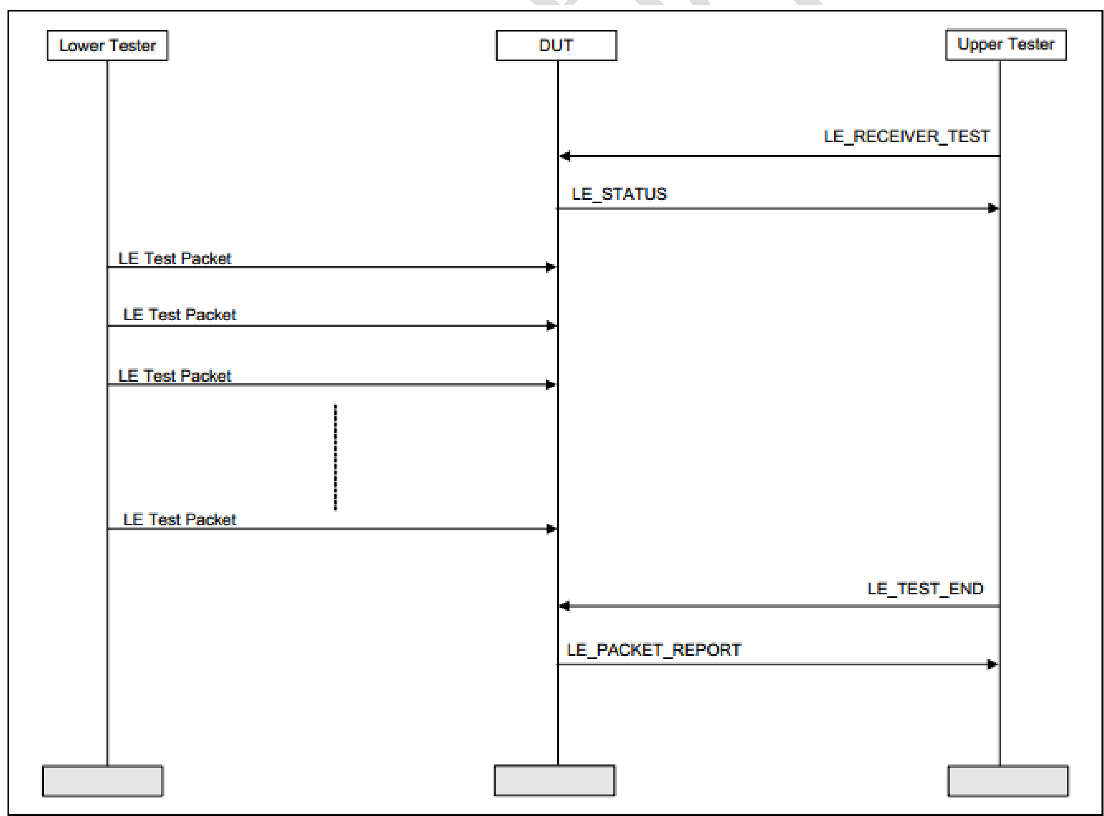


图 2-97 Receiver Test MSC

2.10.5、命令和事件

对于待测设备，命令是上位机发送，DUT 接收。事件是 DUT 发送，上位机接收。也就是说命令和事件是串口之间的通信，串口的命令和事件发送时都是以高字节先发送，然后发送低字节。测试的命令和事件如图 2-98。

Command (DUT RXD)	Event (DUT TXD)
LE_Reset	LE_Test_Status SUCCESS LE_Test_Status FAIL
LE_Receiver_Test	LE_Test_Status SUCCESS LE_Test_Status FAIL
LE_Transmitter_Test	LE_Test_Status SUCCESS LE_Test_Status FAIL
LE_Test_End	LE_Packet_Report LE_Test_Status FAIL

图 2-98 2-Wire command and event behavior

2.10.5.1、命令

命令格式如图 2-99。

15	MSByte	8	7	LSByte	0
CMD	FREQUENCY		LENGTH		PKT

图 2-99 命令信息格式

CMD(command)有 2bit。如表 2-6。

表 2-6 命令描述

命令：b_1b_0	描述
00	复位

01	接收测试
10	传输测试
11	测试结束

Frequency 共有 6bit。如表 2-7。

表 2-7 频率编号

频率编号	描述
N	N 为 0x00~0x27，对应的频率 F=2402+2N

Length 共有 6 比特。如表 2-8。

表 2-8 长度描述

长度值	描述
N	N=0x00~0x25:为每个包中的有效数据长度。 N=0x26~0x3f:保留。

PTK(Packet Type)包类型共有 2bit。如表 2-9。

表 2-9 包类型说明

包类型 b_1b_0	描述
00	PRBS9 报文
01	11110000 报文
10	10101010 报文
11	制造商用

- PRBS9 报文----用于发射功率测试

PRBS9 报文使用多个重复的 9 位伪随机比特序列。该随机机序列

由一个线性反馈移位寄存器生成。它的特性是它与进行过白化处理的报文有类似的随机特征。在我的软件设计中是一个宏如下：

```
#define PRBS9_CONTENT {0xff, 0xc1, 0xfb, 0xe8, 0x4c, 0x90, 0x72, 0x8b,
                       0xe7, 0xb3, 0x51, 0x89, 0x63, 0xab, 0x23, 0x23,
                       0x02, 0x84, 0x18, 0x72, 0xaa, 0x61, 0x2f, 0x3b,
                       0x51, 0xa8, 0xe5, 0x37, 0x49, 0xfb, 0xc9, 0xca,
                       0x0c, 0x18, 0x53, 0x2c, 0xfd}
```

- 11110000----用于频偏测试

“11110000”报文使用重复的 4 个连“1”和 4 个连“0”。该报文可用于连“0”或者连“1”情况下的频偏测试。为了能接收到连续的 0 或者 1，测试才禁止白化的。

- 10101010----用于载波频率和初始频率测试

“10101010”报文使用交替的“1”和“0”。该报文的“0”、“1”交替情况下的频偏测试，也可用于载波频率测试和发射初始频率测试。

2.10.5.2、事件

测试事件有两种类型：

- LE_Test_Status_Event： 测试状态事件
- LE_Test_Packet_Report_Event：测试报告报文事件

事件包格式如图 2-100。

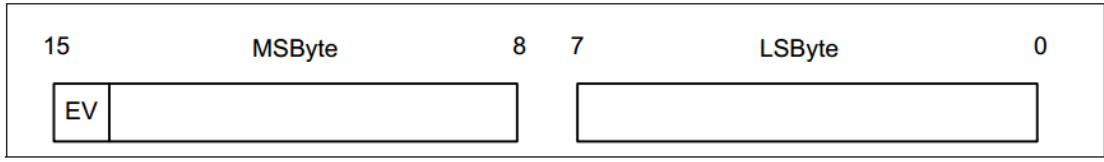


图 2-100 事件包格式

EV(Event)事件共 1bit。如表 2-10。

表 2-10 事件类型

事件值	描述
0	LE_Test_Status_Event： 测试状态事件
1	LE_Test_Packet_Report_Event：测试报告报文事件

2.10.5.2.1、测试状态事件

测试状态事件如图 2-101。

15	MSByte	8	7	LSByte	0
EV	DC			DC	ST

ST (status): *Size: 1 Bit*

Value	Parameter Description
0	Success
1	Error

DC (don't care): *Size: 14 Bits*

Value	Parameter Description
xxxxxxxxxxxxx	Value ignored

图 2-101 测试状态事件

2.10.5.2.2、测试报告报文事件

测试报告报文事件如图 2-102。

15	MSByte	8	7	LSByte	0
EV	PACKET COUNT			PACKET COUNT	

PACKET COUNT: *Size: 15 Bits*

Value	Parameter Description
N	N is the number of packets received Range = 0 to 32767.

图 2-102 测试报告报文事件

这里注意，如果测试时间过长发生溢出，DUT 是无法判断溢出的，也就是必须由上位测试设备保证测试时间不会长到导致计数器溢出。

2.10.6、DTU 软件设计

直接测试层的软件设计流程图如图 2-103。

版权所有

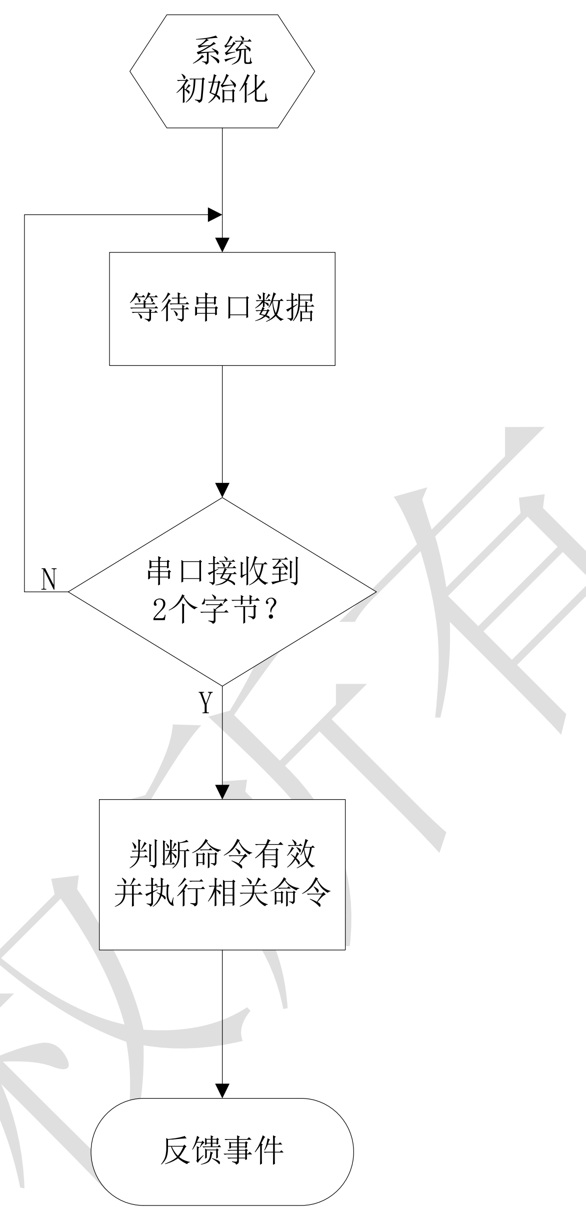


图 2-103 直接测试层软件流程图

版权所有

2.10.7、NRF51822 的测试结果

Anritsu

Bluetooth low energy Test Report

Test Set Serial Number: 001128004	Date: 2015-8-31
Test Set Firmware: 4.20.000	Time: 15:20:15

Overall Result: PASSED

Packet Length: 37

TRM-LE/CA/BV-01-C (Output Power)

	Low	Medium	High	Limits
Average Power	-0.32 dBm	-0.27 dBm	-0.04 dBm	
Max Power	-0.32 dBm	-0.27 dBm	-0.04 dBm	< 10 dBm
Min Power	-0.33 dBm	-0.27 dBm	-0.04 dBm	> -20 dBm

Peak to Average	0.13 dB	0.11 dB	0.11 dB	< 3 dB
Total Packets Failed	0	0	0	
Total Packets Tested	10	10	10	
Result	Passed	Passed	Passed	

TRM-LE/CA/BV-06-C (Carrier Frequency Offset and Drift)

	Low	Medium	High	Limits
Average Frequency Offset	17.3 kHz	19.3 kHz	8.5 kHz	
Max +ve Frequency Offset	22.3 kHz	31.8 kHz	11.3 kHz	≤ 150 kHz
Max -ve Frequency Offset	0.1 kHz	-7.4 kHz	-6.3 kHz	≤ 150 kHz
Drift Rate / 50 µs	10.99 kHz/50 µs	12.64 kHz/50 µs	16.84 kHz/50 µs	< 20 kHz / 50 µs
Max Drift	27 kHz	23 kHz	17 kHz	< 50 kHz
Average Drift	17 kHz	19 kHz	13 kHz	
Total Packets Failed	0	0	0	
Total Packets Tested	10	10	10	
Overall Result	Passed	Passed	Passed	

TRM-LE/CA/BV-05-C (Modulation Characteristics)

	Low	Medium	High	Limits
'Flavg'	234.6 kHz	236.7 kHz	233.7 kHz	225 kHz ≤ Flavg ≤ 275 kHz

'F1max'	242.9 kHz	243.5 kHz	242.1 kHz	
F1 Packets Failed	0	0	0	
'F2avg'	202.5 kHz	204.1 kHz	201.0 kHz	
'F2max'	188.3 kHz	189.5 kHz	187.0 kHz	≥ 185 kHz
'F2max' Pass Rate	100.00 %	100.00 %	100.00 %	
F1/F2 Ratio	0.86	0.86	0.86	≥ 0.8
Total Packets Tested	20	20	20	
Result	Passed	Passed	Passed	

RCV-LE/CA/BV-01-C (Receiver Sensitivity)

	Low	Medium	High	Limits
Frame Error Rate	0.13 %	0.00 %	0.07 %	≤ 30.8 %
Total Packets Received	1498	1500	1499	
Total Packets Tested	1500	1500	1500	
Result	Passed	Passed	Passed	

RCV-LE/CA/BV-07-C (PER Report Integrity)

	Cycle 1	Cycle 2	Cycle 3	Limits
Frame Error Rate	50.00 %	50.00 %	50.00 %	50.0 % ≤ FER ≤ 65.4 %
Total Packets Received	63	127	203	

Total Packets Transmitted	126	254	406	
Result	Passed	Passed	Passed	

RCV-LE/CA/BV-06-C (Maximum Input Power)

	Low	Medium	High	Limits
Frame Error Rate	0.00 %	0.00 %	0.00 %	≤ 30.8 %
Total Packets Received	1500	1500	1500	
Total Packets Tested	1500	1500	1500	
Result	Passed	Passed	Passed	

2.10.8、测试结果对应的命令和事件

蓝牙直接测试时采集的数据											
测试仪器发送的命令							待测试蓝牙发送的事件				
16 进制	2 进制				解释 注：传输测试是指待测蓝牙发送包，仪器接收 接收测试是指待测蓝牙接收包，仪器发送		16 进制	2 进制			解释
命令数 值	命令	频率	长度	包类型			事件数值	事件	数据域	状态	
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
01 00	01	000000	000000	00	接收测试		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误

80 94	10	000000	100101	00	传输测试 频率为 0 即 2402MHz 通道，空中包长度为 37 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
93 94	10	010011	100101	00	传输测试 频率为 19 即 2440MHz 通道，空中包长度为 37 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
A7 94	10	100111	100101	00	传输测试 频率为 39 即 2480MHz 通道，空中包长度为 37 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
80 96	10	000000	100101	10	传输测试 频率为 0 即 2402MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
93 96	10	010011	100101	10	传输测试 频率为 19 即 2440MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
A7 96	10	100111	100101	10	传输测试 频率为 39 即 2480MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
80 95	10	000000	100101	01	传输测试 频率为 0 即 2402MHz 通道，空中包长度为 37 字节，传输 11110000 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
80 96	10	000000	100101	10	传输测试 频率为 0 即 2402MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0

93 95	10	010011	100101	01	传输测试 频率为 19 即 2440MHz 通道，空中包长度为 37 字节，传输 11110000 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
93 96	10	010011	100101	10	传输测试 频率为 19 即 2440MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
A7 95	10	100111	100101	01	传输测试 频率为 39 即 2480MHz 通道，空中包长度为 37 字节，传输 11110000 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
A7 96	10	100111	100101	10	传输测试 频率为 39 即 2480MHz 通道，空中包长度为 37 字节，传输 10101010 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 00	1	000 0000 0000 000	0	包报告事件，包数为 0
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
40 00	01	000000	000000	00	接收测试		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 DB	1	000 0101 1101 101	1	包报告事件，包数为 1499
53 00	01	010011	000000	00	接收测试 频率为 19 即 2440MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 DB	1	000 0101 1101 101	1	包报告事件，包数为 1499
67 00	01	100111	000000	00	接收测试 频率为 39 即 2480MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 DC	1	000 0101 1101 110	0	包报告事件，包数为 1500
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
53 00	01	010011	000000	00	接收测试 频率为 19 即 2440MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 86	1	000 0000 1000 011	0	包报告事件，包数为 86

53 00	01	010011	000000	00	接收测试 频率为 19 即 2440MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		82 C6	1	000 0010 1100 011	0	包报告事件，包数为 710
53 00	01	010011	000000	00	接收测试 频率为 19 即 2440MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		80 E0	1	000 0000 1110 000	0	包报告事件，包数为 224
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误
40 00	01	000000	000000	00	接收测试		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 C	1	000 0101 1101 110	0	包报告事件，包数为 1500
53 00	01	010011	000000	00	接收测试 频率为 19 即 2440MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 C	1	000 0101 1101 110	0	包报告事件，包数为 1500
67 00	01	100111	000000	00	接收测试 频率为 39 即 2480MHz 通道，空中包长度为 0 字节，传输 PRBS9 格式数据		00 00	0	000 0000 0000 000	0	命令接收成功
C0 00	11	000000	000000	00	测试结束		85 DC	1	000 0101 1101 110	0	包报告事件，包数为 1500
C0 00	11	000000	000000	00	测试结束		00 01	0	000 0000 0000 000	1	命令错误

版权

2.11、主机控制接口(HCI)

在整个协议中最好理解的应该是 HCI 层了，HCI 就是一个接口，这个接口是连接主机和控制器的，既然是连接两个东西的接口，那么必定是控制器有部分 HCI 的程序，主机也有部分 HCI 的程序。所以在协议的框图中将HCI完全画在控制层有点不妥的，至少是误导了我的。

对于 nordic 的 nrf51822 这块芯片主机可控制器在同一个芯片中，所以我在程序设计中，虽然花了将近半个月的时间完成 HCI 控制层的程序，实际上在我的程序中并没有用到 HCI 层，直接将 L2CAP 层和 LL 层数据进行交换。

然而实际上，如果是 HOST 和 CONTROLLER 是分开在不同的芯片中的话，HCI 的接口包括了物理接口和逻辑接口。物理接口就是通过什么方式将数据发送到对端设备，例如 UART、USB 的等。逻辑接口是定义要传输的数据的组成方式，也就是数据包格式。

2.11.1、物理接口

在 4.0 的规范中花了一个卷(Volume 4-Host Controller Interface)专门讲了 HCI 的物理接口，共有 4 种物理接口：

- 通用异步收发器(UART)
- 3 线 UART
- USB
- SDIO

版权所有

我的程序中是采用的 UART 接口，也只研究了这个接口，所以下面只讲这个接口。

2.11.1.1、UART

- UART 的配置

UART 的波特率可以自己定义，协议中没有明确规定，8bit 的数据位，没有起始位，1bit 的停止位，同时通过 RTS/CTS 进行流控，RTS 连接对端的 CTS 的连接方式。

CTS=1，允许发送数据。

CTS=0，不允许发送数据。

这里注意，通过 RTS/CTS 进行的流控和逻辑接口协议中的流控不同，这里的流控是防止 UART 物理的缓冲区满而进行数据限制。

- UART 的协议

对于 HCI 的协议并没有给出发送的数据是什么类型，也就是对方如果仅仅根据 HCI 的数据包是无法判断出这个数据是命令包还是可用的数据包。所以需要在物理接口进行标识：

- 命令(HCI Command Packet) =0x01
- 异步无连接链路数据包(HCI ACL Data Packet) =0x02
- 同步面向连接链路数据包(HCI Synchronous Data Packet) =0x03
- 事件(HCI Event Packet) =0x04

对于低功耗蓝牙来说，没有用到 0x03 的同步数据链路的命令。那么到底是怎么标识的呢？就是在发送的数据前面加上上面中的某

版权所有

个字节即可，如果是命令包那么发送时在这个包前加上 0x01 这个字节。

2.11.2、逻辑接口—HCI 包格式

在整个 4.0 的协议规范中的 Volume 2 中的 Part E 部分共有 500 多页讲述 HCI 的功能，相当于整个协议规范的 1/5 还多的篇幅进行描述，可见这个接口是多么的复杂，当然仅仅 BLE 并没有这么多的数据包。在我的程序中仅仅实现了控制层的 HCI 接口程序，所以下面描述中都站在控制层的角色。

从上节的 UART 标识可以知道，对于 HCI 的逻辑接口应该也有 4 种包格式，而对于 BLE 应该是共有 3 种包格式：命令数据包、数据包、事件数据包。对于控制层来说，它接收的只有命令包和数据包，发送给主机层的是事件包。

2.11.2.1、命令数据包

只能从主机层发送给控制层。每个命令有两个字节的操作码，这个两个字节分成两个区域，OpCode Group Field (OGF) 和 OpCode Command Field (OCF)，即操作组和操作命令区域。HCI 的命令包格式如图 2-104。

版权所有

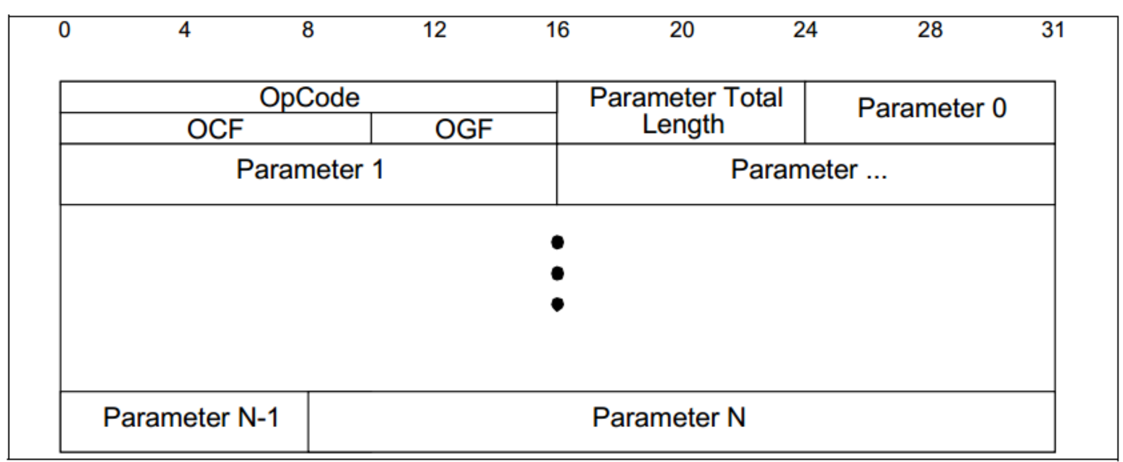


图 2-104 HCI 命令包格式

操作码： 长度：2 字节

值	参数描述
0xXXXX	OGF：6bit，0x00-0x3F 其中 0x3F 保留给厂商调试用 OCF：10bit，0x0000-0x03FF

参数总长： 长度：1 字节

值	参数描述
0xXX	这个长度是指这个命令包的所有参数的总字节数，是参数有多少个字节

参数 0-N： 长度：参数总长 字节

值	参数描述
0xXX	每一个命令都有自己特定的参数

对于操作组(OGF)，在 4.0 的协议中共有 7 个值可用，如表 2-11 命令操作组类型。

表 2-11 命令操作组类型

命令类型	OGF 值	BLE 支持	OCF 命令个数
Link Control commands	0x01	**Y**	2
Link Policy Commands	0x02	**N**	0
Controller & Baseband Commands	0x03	**Y**	6
Informational Parameters	0x04	**Y**	4
status parameters	0x05	**Y**	1
Testing Commands	0x06	**N**	0
LE Controller Commands	0x08	**Y**	30

上表中的测试命令并不是低功耗没有测试命令，而是将测试命令放在了 LE Controller Commands 中，也就是 Testing Commands 是测试经典蓝牙的 PHY。状态参数命令只有 1 个，就是 HOST 获取信号强度值(RSSI)。

2.11.2.2、事件数据包

事件包是应答主机发送的命令包，几乎每一个命令包都会伴随有事件包进行应答，仅仅只有极少数的命令是没有应答的，有的命令可能需要两个应答事件：一个是命令状态事件，用来表明命令接收到了；再一个是命令完成事件，表明命令执行完毕。在 BLE 中的事件包的总数为 14 个，因为有许多命令是用通用命令完成事件进行应答，事实上 BLE 的事件类型中主要分为 3 种：

- 通用命令完成事件

- 通用命令状态事件
- 特定命令完成事件

如图 2-105 所示，事件包是由事件码、参数总长及参数组成。

0	4	8	12	16	20	24	28	31
Event Code		Parameter Total Length		Event Parameter 0				
Event Parameter 1				Event Parameter 2		Event Parameter 3		
● ● ●								
Event Parameter N-1				Event Parameter N				

图 2-105 HCI 事件包格式

事件码：　　　　　　　　　　　　　　　　　　　　　　长度：1 字节

值	参数描述
0xXX	这是事件的唯一标识码，0-0xFF(0xFF 保留厂商用)

参数总长：　　　　　　　　　　　　　　　　　　　　　长度：1 字节

值	参数描述
0xXX	这个长度是指这个事件包的所有参数的总字节数，参数共有多少个字节

参数 0-N：　　　　　　　　　　　　　　　　　　长度：参数总长 字节

值	参数描述
0xXX	每一个事件码都有自己特定的参数

2.11.2.3、数据包

这里所讲的数据包为异步数据包，即只讲 BLE 支持的数据包格式。不过不幸的是我并没有理解数据包中的句柄和标志位，因为程序中没有用到 HCI，也就没有认真研究了。如图 2-106 所示。

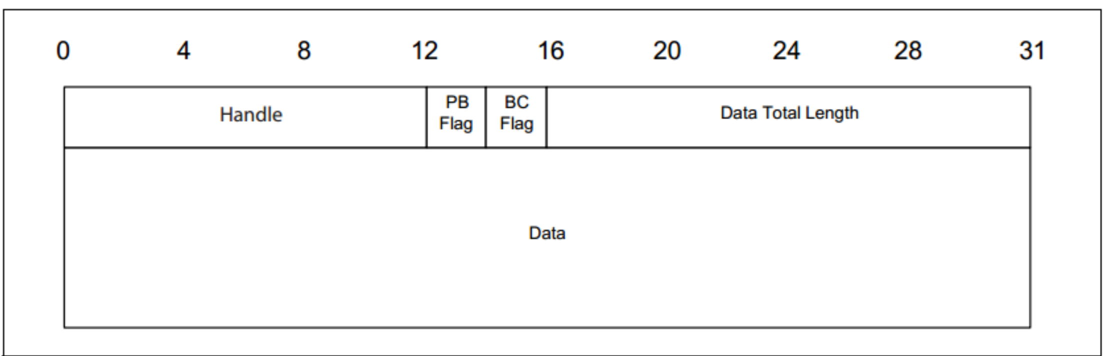


图 2-106 BLE 中 HCI 的数据包

这里预留一部分的空间留着日后补充吧！

版权所

2.11.3、命令和事件类型

低功耗蓝牙的所有 HCI 命令共计 43 个，通过 OGF 进行分组，分组数量分配见表 2-11。事件共计 14 个，其中重要的事件是命令完成事件 (Command_Complete_Event) 和命令状态事件 (Command_Status_Event)，这两个事件几乎要完成所有命令的回复。具体见下表 2-12HCI 命令格式和表 2-13HCI 事件格式，命令和事件总图为图 2-107。这里仅仅是总结出了 BLE 的命令和事件，参数的具体意义还需到协议规范中阅读，表的目的是为了编写程序时，可以一览所有命令的 OGF、OCF 以及参数和事件的参数，命令的返回参数是放在事件中返回的。再次明确本节中的 HCI 都是在控制层视角的命令和事件。

版权所有

Generic Events
Command Complete
Command Status
Hardware Error

Device Setup
Reset

Device Discovery
LE Advertising Report
LE Set Scan Enable
LE Set Scan Parameters

Host Flow Control
Host Buffer Size
Set Event Mask
Set Controller To Host Flow Control
Host Number of Completed Packets
Data Buffer Overflow
LE Add Device to White List
LE Clear White List
LE Read White List Size
LE Remove Device from White List
LE Set Event Mask

Controller Flow Control
Read Buffer Size
Number of Completed Packets
LE Read Buffer Size

Connection Setup
Disconnect Command
Disconnection Complete
LE Connection Complete
LE Create Connection Cancel
LE Create Connection

Controller Information
Read Local Version Information
Read Local Supported Commands
Read Local Supported Features
Read BDADDR
LE Read Local Supported Features
LE Read Supported States

Remote Information
Read Remote Version Information
Read Remote Version Information Complete
LE Read Remote Used Features
LE Read Remote Used Features Complete

Link Information
Read Transmit Power Level
Read RSSI
LE Read Advertising Channel TX Power
LE Read Channel Map

Controller Configuration
LE Set Advertise Enable
LE Set Advertising Data
LE Set Advertising Parameters
LE Set Random Address
LE Set Scan Response Data
LE Set Random Address

Connection State
LE Connection Update
LE Connection Update Complete

Physical Links
LE Set Host Channel Classification

Authentication and Encryption
Encryption Change
Encryption Key Refresh Complete
LE Encrypt
LE Long Term Key Requested
LE Long Term Key Requested Reply
LE Long Term Key Requested Negative Reply
LE Rand
LE Start Encryption

Test
LE Receiver Test
LE Transmitter Test
LE Test End

Black – existing commands
Black italicized – existing events
Red – new commands
Red italicized – new events

图 2-107 HCI 命令和事件总图

Command	OGF	OCF	Command_Parameters	Return_Parameters	返回事件
HCI_Disconnect	0x01	0x0006	Connection_Handle, Reason	返回特定参数	Command_Status_Event, Disconnection_Complete _Event
HCI_Read_Remote_Version _Information	0x01	0x001D	Connection_Handle	返回特定参数	Command_Status_Event, Read_Remote_Version _Information_Complete _Event
HCI_Set_Event_Mask	0x03	0x0001	Event_Mask	Status	Command_Complete_Evevt
HCI_Reset	0x03	0x0003		Status	Command_Complete_Evevt
HCI_Read_Transmit_Power _Level	0x03	0x002D	Connection_Handle, Type	Status, Connection_Handle, Transmit_Power_Level	Command_Complete_Evevt
HCI_Set_Controller_To_Host _Flow_Control	0x03	0x0031	Flow_Control_Enable	Status	Command_Complete_Evevt
HCI_Host_Buffer_Size	0x03	0x0033	Host_ACL_Data_Packet_Length, Host_Synchronous_Data_Packet_Length, Host_Total_Num_ACL_Data_Packets, Host_Total_Num_Synchronous_Data_Packets	Status	Command_Complete_Evevt
HCI_Host_Number_Of _Completed_Packets	0x03	0x0035	Number_Of_Handles Connection_Handle[i], Host_Num_Of_Completed_Packets[i]	Status	Command_Complete_Evevt

HCI_Read_Local_Version_Information	0x04	0x0001		Status, HCI_Version, HCI_Revision, LMP_Version, Manufacturer_Name, LMP_Subversion	Command_Complete_Evevt
HCI_Read_Local_Supported_Features	0x04	0x0003		Status, LMP_Features	Command_Complete_Evevt
HCI_Read_Buffer_Size	0x04	0x0005		Status, HC_ACL_Data_Packet_Length, HC_Synchronous_Data_Packet_Length, HC_Total_Num_ACL_Data_Packets, HC_Total_Num_Synchronous_Data_Packet	Command_Complete_Evevt
HCI_Read_BD_ADDR	0x04	0x0009		Status, BD_ADDR,	Command_Complete_Evevt
HCI_Read_RSSI	0x05	0x0005	Handle	Status, Handle, RSSI	Command_Complete_Evevt
HCI_LE_Set_Event_Mask	0x08	0x0001	LE_Event_Mask	Status,	Command_Complete_Evevt
HCI_LE_Read_Buffer_Size	0x08	0x0002		Status, HC_LE_ACL_Data_Packet_Length, HC_Total_Num_LE_ACL_Data_Packets	Command_Complete_Evevt
HCI_LE_Read_Local_Supported_Features	0x08	0x0003		Status, LE_Features,	Command_Complete_Evevt
HCI_LE_Set_Random_Address	0x08	0x0005	Random_Address	Status	Command_Complete_Evevt

HCI_LE_Set_Advertising _Parameters	0x08	0x0006	Advertising_Interval_Min, Advertising_Interval_Max, Advertising_Type, Own_Address_Type, Direct_Address_Type, Direct_Address, Advertising_Channel_Map, Advertising_Filter_Policy	Status	Command_Complete_Evevt
HCI_LE_Read_Advertising _Channel_Tx_Power	0x08	0x0007		Status Transmit_Power_Leve	Command_Complete_Evevt
HCI_LE_Set_Advertising _Data	0x08	0x0008	Advertising_Data_Length, Advertising_Data,	Status	Command_Complete_Evevt
HCI_LE_Set_Scan_Response _Data	0x08	0x0009	Scan_Response_Data_Length, Scan_Response_Data,	Status	Command_Complete_Evevt
HCI_LE_Set_Advertise_Enable	0x08	0x000A	Advertising_Enable	Status	Command_Complete_Evevt 有其他条件需要考虑
HCI_LE_Set_Scan_Parameters	0x08	0x000B	LE_Scan_Type, LE_Scan_Interval, LE_Scan_Window, Own_Address_Type, Scanning_Filter_Policy	Status	Command_Complete_Evevt
HCI_LE_Set_Scan_Enable	0x08	0x000C	LE_Scan_Enable, Filter_Duplicates,	Status	Command_Complete_Evevt 有其他条件需要考虑

HCI_LE_Create_Connection	0x08	0x000D	LE_Scan_Interval, LE_Scan_Window, Initiator_Filter_Policy Peer_Address_Type, Peer_Address, Own_Address_Type, Conn_Interval_Min, Conn_Interval_Max, Conn_Latency, Supervision_Timeout, Minimum_CE_Length, Maximum_CE_Length	返回特定参数	Command_Status_Event, LE_Command_Complete _Evevt
HCI_LE_Create_Connection _Cancel	0x08	0x000E		Status	Command_Complete_Evevt 有其他条件需要考虑
HCI_LE_Read_White_List_Size	0x08	0x000F		Status White_List_Size	Command_Complete_Evevt
HCI_LE_Clear_White_List _Size	0x08	0x0010		Status	Command_Complete_Evevt
HCI_LE_Add_Device_To _White_List	0x08	0x0011	Address_Type, Address	Status	Command_Complete_Evevt
HCI_LE_Remove_Device _From_White_List	0x08	0x0012	Address_Type, Address	Status	Command_Complete_Evevt

HCI_LE_Connection_Update	0x08	0x0013	Connection_Handle, Conn_Interval_Min, Conn_Interval_Max, Conn_Latency, Supervision_Timeout, Minimum_CE_Length, Maximum_CE_Length	返回特定参数	Command_Status_Event, LE_Connection_Update _Complete_Event
HCI_LE_Set_Host_Channel _Classification	0x08	0x0014	Channel_Map	Status	Command_Complete_Evevt
HCI_LE_Read_Channel_Map	0x08	0x0015	Connection_Handle	Status Connection_Handl Channel_Map	Command_Complete_Evevt
HCI_LE_Read_Remote_Used _Features	0x08	0x0016	Connection_Handle	返回特定参数	Command_Status_Event, LE_Read_Remote_Used _Features_Complete_Event
HCI_LE_Encrypt	0x08	0x0017	Key, Plaintext_Data	Status Encrypted_Data	Command_Complete_Evevt
HCI_LE_Rand	0x08	0x0018		Status Random_Number	Command_Complete_Evevt
HCI_LE_Start_Encryption	0x08	0x0019	Connection_Handle Random_Number, Encrypted_Diversifier, Long_Term_Key	返回特定参数	Command_Status_Event, 有条件判断回复之一 Encryption_Change_Event\ Encryption_Key_Refresh _Complete_Event
HCI_LE_Long_Term_Key _Request_Reply	0x08	0x001A	Connection_Handle, Long_Term_Key	Status Connection_Handle	Command_Complete_Evevt

HCI_LE_Long_Term_Key_Requested_Negative_Reply	0x08	0x001B	Connection_Handle	Status, Connection_Handle	Command_Complete_Evevt
HCI_LE_Read_Supported_States	0x08	0x001C		Status LE_States	Command_Complete_Evevt
HCI_LE_Receiver_Test	0x08	0x001D	RX_Frequency	Status	Command_Complete_Evevt
HCI_LE_Transmitter_Test	0x08	0x001E	TX_Frequency Length_Of_Test_Data Packet_Payload	Status	Command_Complete_Evevt
HCI_LE_Test_End	0x08	0x001F	TX_Frequency Length_Of_Test_Data Packet_Payload	Status Number_Of_Packets	Command_Complete_Evevt

表 2-12 HCI 命令格式

版权所有

Event	Event_Code	Event_Parameters
Disconnection_Complete_Event	0x05	Status, Connection_Handle, Reason
Encryption_Change_Event	0x08	Status, Connection_Handle, Encryption_Enabled,
Read_Remote_Version _Information_Complete_Event	0x0C	Status, Connection_Handle, Version, Manufacturer_Name, Subversion
Command_Complete_Event	0x0E	Num_HCI_Command_Packets, Command_Opcode, Return_Parameters
Command_Status_Event	0x0F	Status, Num_HCI_Command_Packets, Command_Opcode
Hardware_Error_Event	0x10	Hardware_Code
Number_Of_Completed_Packets_Event	0x13	Number_of_Handles, Connection_Handle[i], HCI_Num_Of_Completed_Packets[i]
Data_Buffer_Overflow_Event	0x1A	Link_Type
Encryption_Key _Refresh_Complete_Event	0x30	Status, Connection_Handle
LE_ Connection_ Complete_Event	0x3E	Subevent_Code, Status, Connection_Handle, Role, Peer_Address_Type, Peer_Address, Conn_Interval, Conn_Latency, Supervision_Timeout, Master_Clock_Accuracy
LE _Advertising_Report_Event	0x3E	Subevent_Code, Num_Reports, Event_Type[i], Address_Type[i], Address[i], Length[i], Data[i], RSSI[i]

LE_Connection_Update_Complete_Event	0x3E	Subevent_Code, Status, Connection_Handle, Conn_Interval, Conn_Latency, Supervision_Timeout
LE_Read_Remote_Used_Features_Complete_Event	0x3E	Subevent_Code, Status, Connection_Handle, LE_Features
LE_ Long_Term_ Key_Request_Event	0x3E	Subevent_Code, Connection_Handle, Random_Number, Encryption_Diversifier

表 2-13 HCI 事件格式

版权所有

2.11.4、HCI 软件设计

在 HCI 软件设计中，主要完成 3 件事情：

- UART 数据收发
- 接收命令后解析和执行
- 事件数据的组包和发送

所以它的流程图是非常简单的，如图 2-108。

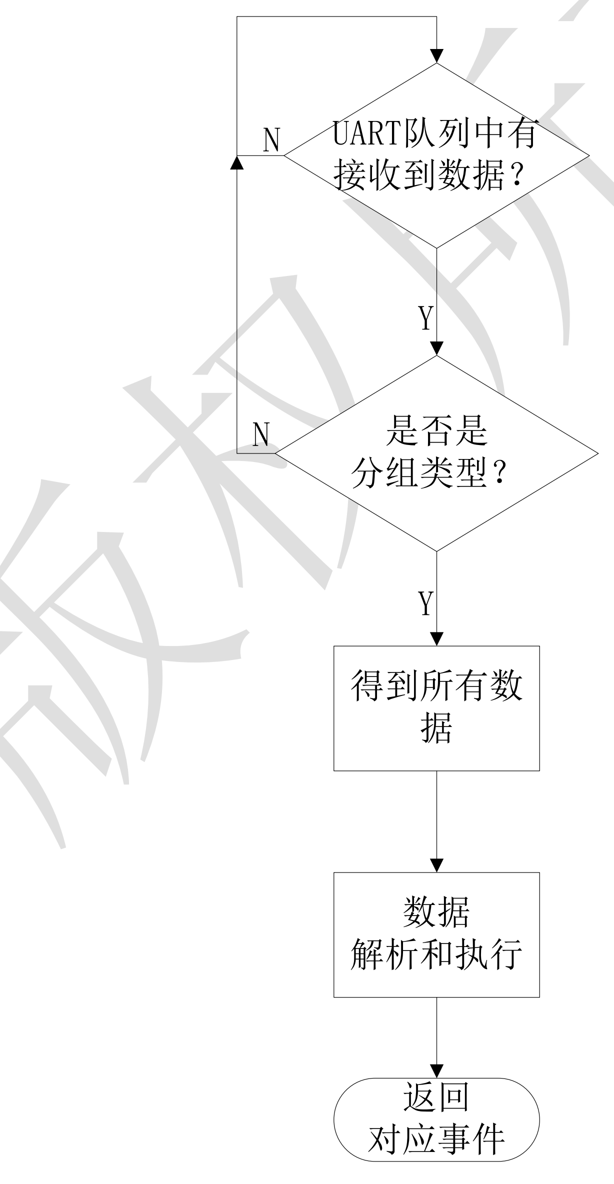


图 2-108 HCI 软件流程图

版权所有

在 UART 接收数据处理中，采用了一个环形缓冲器进行数据接收，并采用状态机将 UART 接收到的数据分为 5 个状态，并定义了如下枚举。

```
typedef enum
{
	HCI_WAIT_FOR_PACKET_TYPE=0,   //等待数据分组
	HCI_WAIT_FOR_COMMAND_HDR, //如果知道是命令分组那么就等接收到 3 个字节
	HCI_WAIT_FOR_COMMAND_DATA, //命令头接到后得到参数长度，等所有参数接收完毕
	HCI_WAIT_FOR_ACL_HDR,          //如果知道是数据分组那么就等接收到 4 个字节
	HCI_WAIT_FOR_ACL_DATA          //数据头接到后得到参数长度，等所有参数接收完毕
}S_BLE_HCI_State_t;
```

默认状态为等待数据分组类型，等到数据分组类型就可以判断出本次接收的数据是命令还是数据，并将状态进行相应的切换，接着继续判断接收到的是头部，如果接收到了头部，那么就可以计算出接下来这个包还有多少数据需要接收，当这个包的所有数据都接收到后，就可以进行数据解析了，然后回复响应的事件。整个软件看似很简单，但是面对这 40 多个命令和 10 多个事件需要进行解析执行和组包，真是一个大的工程。我软件中设计的不完整 HCI 代码就有差不多 3000 行代码。

2.11.5、HCI 模拟数据传输

这里模拟命令从主机传送到控制层以及控制层回复事件给主机层的例子以及数据包传输的例子。这里以 HCI_Disconnect 命令进行模拟传输，因为它需要两个返回事件，一个是命令状态事件(Command_Status_Event)，一个是关闭连接完成事件(Disconnection_Complete_Event)。命令和事件如下表 2-14。

表 2-14 HCI_Disconnect 命令和事件

Command	OGF	OCF	Command_Parameters	Return_Parameters	返回事件
HCI_Disconnect	0x01	0x0006	Connection_Handle, Reason	返回特定参数	Command_Status_Event, Disconnection_Complete_Event

Event	Event_Code	Event_Parameters
Disconnection_Complete_Event	0x05	Status, Connection_Handle, Reason
Command_Status_Event	0x0F	Status, Num_HCI_Command_Packets, Command_Opcode

- HOST 发送 HCI_Disconnect 命令给 CONTROLLER
  - 【0x01 0x06 0x04 0x03 0x29 0x00 0x13】关闭连接命令

解析：

【0x01】：分组类型为 HCI 命令分组

【0x0406】：L->MSB【0110000000 | 100000】即 OGF=0x01，OCF=0x0006

【0x03】：参数长度为 3 字节

【0x0029】：连接句柄

【0x13】：断开原因：REMOTE USER TERMINATED CONNECTION (0X13)

- CONTROLLER 发送事件给 HOST
  - 【0x04 0x0F 0x04 0x00 0x01 0x06 0x04】命令状态事件

解析：

【0x04】 1byte HCI_Packet_Type 分组类型：事件分组类型

【0x0F】 1byte Event_Code 事件代码：命令状态事件

【0x04】 1byte Parameter_Length 参数长度：4 字节

【0x00】 1byte Status 状态：成功

【0x01】 1byte Num_HCI_Command_Packets 控制层可接受 1 包数据

版权所有

【0x0406】 2byte Connection_Handle 句柄：0x0406

➢ 【0x04 0x05 0x04 0x00 0x29 0x00 0x16】关闭连接命令完成事件

解析：

【0x04】 1byte HCI_Packet_Type 分组类型：事件分组类型

【0x05】 1byte Event_Code 事件代码：命令状态事件

【0x04】 1byte Parameter_Length 参数长度：4 字节

【0x00】 1byte Status 状态：成功

【0x0029】 2byte Connection_Handle 句柄：0x0029

【0x16】 1byte Reason 原因:**CONNECTION TERMINATED BY LOCAL HOST (0X16)**

对于数据包的应答事件是

Event	Event_Code	Event_Parameters
Number_Of_Completed_Packets_Event	0x13	Number_of_Handles, Connection_Handle[i], HCI_Num_Of_Completed_Packets[i]

- HOST 发送"Hello"给 CONTROLLER

➢ 【0x02 0x29 0x20 0x0C 0x00 0x05 0x00 0x04 0x00 0x52 0x11 0x00 0x48 0x65 0x6C 0x6C 0x6F】

解释：

【0x02】 1byte HCI_Packet_Type 分组类型：数据分组类型

【0x2029】 2byte Connection_Handle&Flag 句柄=0x29,FB=0x02,BC=0x00

【0x000C】 2byte Data_Length 数据长度：12 字节

【0x0008】 2byte L2cap_ Data_Length L2cap 数据长度：8 字节

【0x0004】 2byte L2cap_CID L2cap 通道 ID：0x0004(ATT)

【0x52】 1byte ATT_Opcode ATT 命令码：写命令

【0x0011】 2byte Handle 属性句柄：0x0011

【0x676C6C6548】 5byte 内容"Hello"的 ASCII 码

在这个数据包中，对于 HCI 层的数据部分 0x05 0x00 0x04 0x00 0x52 0x11 0x00 0x48 0x65 0x6C 0x6C 0x6F，对于 BLE 来说，在 HOST 层所有应用都是基于 ATT 和 GATT 的，在后面章节中会讲，而 HCI 上层的接口又为 L2CAP 层，所以数据中必须包含有 L2CAP 层的数据，也就是说 ATT 是 L2CAP 的数据部分，而 L2CAP 又是 HCI 的数据部分。

- CONTROLLER 发送事件给 HOST
  - 【0x04 0x13 0x05 0x01 0x29 0x00 0x01 0x00】

解释：

【0x04】 1byte HCI_Packet_Type 分组类型：事件分组类型

【0x13】 1byte Event_Code 事件代码：完成包事件

【0x05】 1byte Parameter_Length 参数长度：5 字节

【0x01】 1byte Number_of_Handles 连接句柄数量：1 个

【0x0029】 2byte Connection_Handle 连接句柄

【0x0001】 2byte HCI_Num_Of_Completed_Packets 已完成 HCI 数据分组数

====================

值得一提另外一个 BLE 命令是：

Command	OGF	OCF	Command_Parameters	Return_Parameters	返回事件
HCI_LE_Set _Advertising_Data	0x08	0x0008	Advertising_Data_Length, Advertising_Data	Status	Command_Complete _Evevt

这是设置广播参数的命令，这个命令中的 Advertising_Data 是有讲究的，内容如图 2-44 所示。

第三章 主机

主机(HOST)层，是一个完全软件实现的层，这里不得不整体对 HOST 层进行一个说明，很多时候看不到山的全貌，“只缘身在此山中”。刚开始着手主机，一头扎到协议里面，完全摸不着头脑，最难搞懂的就是 ATT 和 GATT 了。这里引用图 1-1 中的主机部分，如图 3-1 所示。

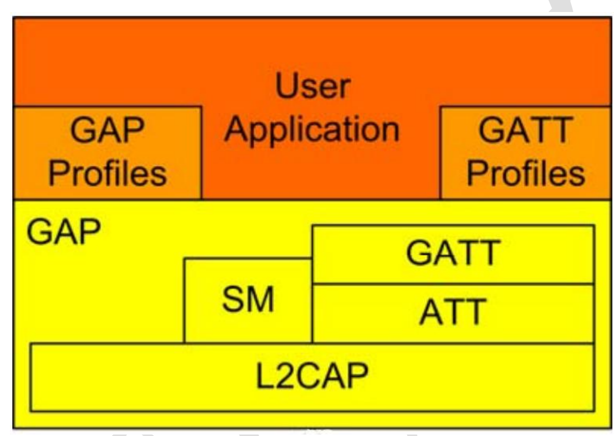


图 3-1 主机层和应用层框图

在图 3-1 中，黄色部分就是整个 HOST 层，再上面就是 Profile 和应用层了。图中可以看到 HOST 层由通用访问规范(GAP)、安全管理协议(SMP)、属性协议(ATT)、通用属性规范(GATT)、逻辑链路控制和适配协议(L2CAP)组成。有许多了 HOST 组成框图，为什么我想引用这张图呢？因为这个图画得很贴切，例如对 GAP 描述时就做得很好，从图中可以看到 GAP 相当于包含了整个黄色部分，在图 2-45 中有讲，GAP 可以有两种路径将数据传输给底层，一个是通过 L2CAP，另一个直接给底层，直接给底层一般是一些初始化的数据，而 L2CAP 层传的是动态数据，所以在框图中 GAP 包含了整个黄色部分。

在这里主要讲 ATT 和 GATT 的关系和对它们的整体理解，这样对于整个 HOST 层的理解会好一些，在后续中的章节中再详细讲述每一层的内容。如表 3-1 所示，简单将属性和煤球做类比。这里再说明属性(Attribute)、Profile、GATT 和 ATT。

- 属性：Profile 是制作煤球的模子，那么这个模子又是谁规定的呢？为什么有的蜂窝煤有 7 个孔，有的 14 个孔？这些肯定是有规定的。对于 Profile 它也必须有规定，每个 Profile 都应该遵循一定的规律，按照一定的规定进行设计。这就是属性。对于 GATT 来说，就是在数据库中的数据存放的位置都有特定的意义。也就是在 GATT 中的数据是有一定规则去存放的，这个规则就是属性。
- Profile：它是根据属性和应用需求制定的模子。
- GATT：它是用 Profile 制作成的数据库。
- ATT：规定了属性该怎么访问和运输，也就是怎么样去访问 GATT 数据库中的数据。

版权所有

表 3-1 ATT/GATT/Profile 理解

GATT Profile 通用属性配置	GATT 通用属性规范	ATT 属性协议
Profile 就像这个做煤球的模子。每个 SIG 组织成员都可以向 SIG 提交这个“模子”，如果审批通过，那么这个就成了全世界都通用的“模子”，你不用管这个“煤球”是在中国烧，还是在美国烧，不管是用的安卓系统还是苹果系统，全部都是通用的，所以这个叫“通用属性规范配置”，也就是“全球通用的蜂窝煤模子”	GATT 就是通过“模子”做出来的各种各样的“煤球”，它相当于一个“煤球供应商”，它这有各式各样已经做好的煤球，都存储在“库房”。对于软件来说，GATT 就相当于一个服务器或者是数据库，这个数据库中有各种各样由 Profile 模子制作出来的数据。这个数据存储在芯片中，软件实现的话就是一个链表，只是这个链表需要动态建立，也就相当于“煤球供应商”不能将所有不同种类的煤球都做出来，他们会根据客户的需求，动态去做哪种煤球。对于 GATT 数据库也是一样的，它会根据应用而在初始化系统时动态建立所用到的“模子”的数据库。	ATT 就相当于一个运输“煤球”工具。它在 4.0 的规范中定义了怎么去访问 GATT 数据库，并传输这些数据到相应层。

3.1、逻辑链路控制和适配协议(L2CAP)

L2CAP 相当于一个交通疏导员，控制器的数据来到了主机层，相当于一辆车来到了一个十字路口，驾驶员决定自己是直行还是左右拐弯(控制层的数据头中包含了数据是什么数据即包含了流向)，车往哪走需要看交通灯(数据到了 L2CAP 层后就由 L2CAP 层进行疏导，根据头部信息判断流向，再进行数据传输)。这样理解的话，意味着 L2CAP 层应该有许多数据道路咯！在低功耗蓝牙中，L2CAP 有 3 个固定的道路，而专用名词是----信道。

3.1.1、L2CAP 信道

信道就是数据传输时用的道路，L2CAP 可以随时修建道路为数据传输，但是对于 BLE 共有 3 个固定的信道，也就是 3 条固定的道路，这 3 条路是全世界都在用的，是协议中规定好的 3 个信道。那么道路都有名字，例如 107 国道，信道必定也要取个名字的，在低功耗蓝牙的信道标识符分配如下表 3-2。

表 3-2 L2CAP 信道标识符

信道标识符	Description	说明
0x0004	Attribute Protocol	属性协议
0x0005	Low Energy L2CAP Signaling channel	低功耗信令信道
0x0006	Security Manager Protocol	安全管理协议

这个信道标识符怎么用呢？在控制器传上来的数据中一定包含

版权所有

有某个信道标识符，那么 L2CAP 层判断是那个信道标识符就能将数据送到相应的上层进行数据处理。当然上层的数据向下发送时，上层需要告诉 L2CAP 层这包数据是 Attribute Protocol 还是 Security Manager Protocol 或者 Low Energy L2CAP Signaling channel，之后 L2CAP 加上相应的标识符后向下层发送数据。下层接收到数据后可以根据信道标识符进行不同的数据解析。

3.1.2、L2CAP 数据包格式

在协议规范中说了许多种的数据包格式，但是对于 BLE 只用到了基本方式数据包格式，如图 3-2。

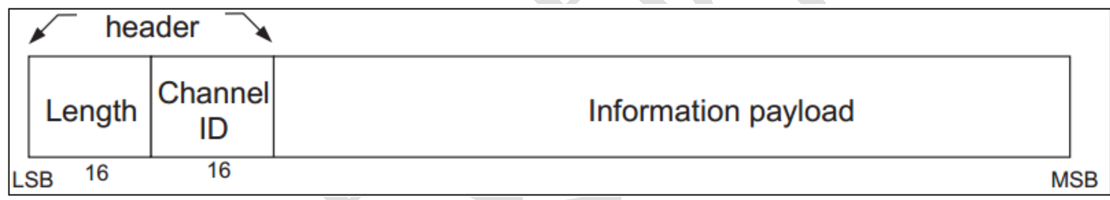


图 3-2 低功耗蓝牙 L2CAP 的数据包格式

- Length：为有效载荷的长度，对于 BLE 最大传输单元(Maximum Transmission Unit (MTU))为 23 字节，也就是 Information payload 的最大数量为 23 字节，那么如果需要发送的数据超过 23 字节怎么办呢？方法是剪短后分段发送，所有在 LL 层的报头中才有 MD 标志存在。Information payload 信息放的就是 L2CAP 上层 ATT 的数据包。
- Channel ID：通道 ID。0x0004、0x0005、0x0006 三个可选。

3.1.3、低功耗信令信道包格式

对于 L2CAP 层的 3 个信道，0x0004 为它的上层属性协议(ATT)使用，0x0006 为它的另外一个与 ATT 并列的一个上层安全管理协议(SMP)使用，而对于 0x0005 它是低功耗信令信道的固定信道，这个信道上传输的是命令包，那么它的上层是谁呢？数据给谁又由谁提供数据呢？答案是通用访问规范 Generic Access Profile(GAP)，只是有意思的是，GAP 并没有具体的数据包的格式，它只提供数据，真正的组包是在别的层，在图 2-45 中可以看到，GAP 的数据是有给 L2CAP 层的，所以 GAP 传给 L2CAP 数据的数据格式是定义在 L2CAP 中的。总体 L2CAP 的数据包格式还是图 3-2 所示，只是 Channel ID 并强制为 0x0005，而 Information payload 的命令格式如图 3-3 所示。

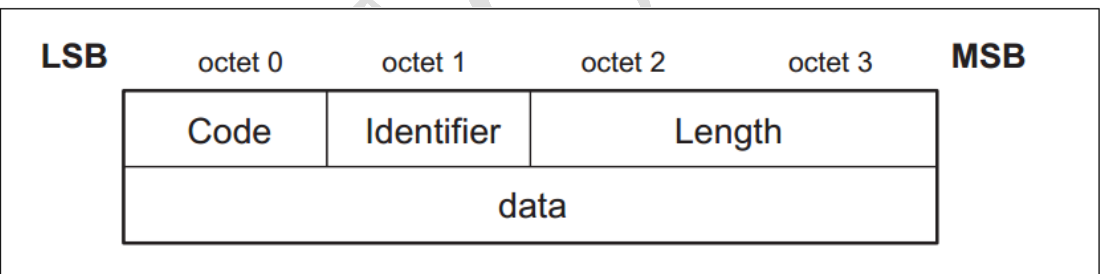


图 3-3 低功耗信令信道的命令格式

- Code：操作码，即 GAP 发送的命令或者接收的命令，对于 BLE 来说其实只有 3 个可用的命令，如表 3-3 所示。

表 3-3 信令命令码

操作码	Description	说明
0x00	Reserved	保留
0x01	Command reject	命令拒绝

0x12	Connection Parameter Update request	连接参数更新请求
0x13	Connection Parameter Update response	连接参数更新应答

- Identifier：标识符，这个的作用是确保应答包对应哪个请求包，发送一个请求包，接收到应答包中的 identity 值如果和请求包中的一致，说明是对这个请求包的应答。identity 的值发送每个请求命令都是不同的值，而对方设备的应答信息中的 ID 和发送请求中的 ID 是一样的，确保对其应答。0x00 是一个非法的值，不能用。
- Length：长度，这个长度表示的是后面 data 中的字节数。
- Data：数据，每个操作码都有固定格式的数据。

3.1.3.1、命令拒绝

命令拒绝操作码，用于对接收到的不支持的数据包的回应。这个命令是沿用经典蓝牙的，它的数据格式如图 3-4。

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x01	Identifier	Length		
	Reason		Data (optional)		

图 3-4 命令拒绝包格式

- Reason 占用两个字节，对于 BLE 只有 2 个值是可用的，如表 3-4 所示。

表 3-4 拒绝命令的原因码

原因代码	Description	说明

0x0000	Command not understood	命令不理解
0x0001	Signaling MTU exceeded	超出最大传输字节数

- Data：数据长度为 0 或者大于 0 字节的数，主要是对原因进行描述补充，而“命令不理解”是没有数据需要进行说明的。只有“Signaling MTU exceeded”有两个字节的数据去应答能接收的最大的字节数是多少，对于 BLE 就是 MTU=23bytes。

3.1.3.2、连接参数更新请求和响应

在 2.9.2.2.1 节提到主机通过 LL 层就可实现连接参数更新，那么从机也有更新连接参数的权利，通过这个命令即可，而且这个命令只能有从机发送给主机，如果从机接收到这个命令后，会回复命令拒绝。它的命令包格式如图 3-5 所示。

LSB octet 0	octet 1	octet 2	octet 3 MSB
Code=0x12	Identifier	Length	
Interval Min		Interval Max	
Slave Latency		Timeout Multiplier	

图 3-5 L2CAP 连接更新请求

参数不解释了，很简单。这里注意哦！里面的 4 个参数都是给的范围，也就是说从机不能最终决定具体连接参数值，从机只能给主机提出自己的想法，如果主机同意更新，那么需要主机发送“LL_CONNECTION_UPDATE_REQ”这个命令详细连接参数给从机，然后从机等到“瞬时”到后进行参数更新。

如果仅仅是从对等层来说，当从机发送 Connection Parameter Update request 命令，主机需要发送 Connection Parameter Update response 这个命令去回应。如图 3-6 所示。Result 为 2 个字节的结果代码。当主机同意修改连接参数的时候，Result 为 0x0000，也就是接受参数更新的建议。当主机不同意修改连接参数的时候，Result=0x0001，拒绝参数更新。

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x13	Identifier	Length		
	Result				

图 3-6 L2CAP 连接更新响应

实际中的参数更新，如图 3-7 所示，是通过 sniffe 采集的空中包，每个详细的包如图 3-8、图 3-9 和图 3-10 所示。

Slave	Master	L2CAP	42 Rcvd Connection Parameter Update Request
Master	Slave	LE LL	38 Control Opcode: LL_CONNECTION_UPDATE_REQ
Slave	Master	LE LL	26 Empty PDU
Master	Slave	L2CAP	36 Rcvd Connection Parameter Update Response (Accepted)

图 3-7 Sniffer 采集参数更新过程

```
⊟ Bluetooth L2CAP Protocol
    Length: 12
    CID: Low Energy L2CAP Signaling Channel (0x0005)
  ⊟ Command: Connection Parameter Update Request
      Command Code: Connection Parameter Update Request (0x12)
      Command Identifier: 0x02
      Command Length: 8
      Min. Interval: 400 (500 msec)
      Max. Interval: 800 (1000 msec)
      Slave Latency: 0 LL Connection Events
      Timeout Multiplier: 400 (4 sec)
```

图 3-8 Sniffer 采集从机 L2CAP 层连接参数更新请求

```
⊟ Bluetooth Low Energy Link Layer
    Access Address: 0x50654267
  ⊟ Data Header: 0x0c0f
      000. .... = RFU: 0
      ...0 .... = More Data: False
      .... 1... = Sequence Number: True
      .... .1.. = Next Expected Sequence Number: True
      .... ..11 = LLID: Control PDU (0x03)
      000. .... = RFU: 0
      ...0 1100 = Length: 12
    Control Opcode: LL_CONNECTION_UPDATE_REQ (0x00)
    Window Size: 3
    Window Offset: 534
    Interval: 798
    Latency: 0
    Timeout: 400
    Instant: 205
  ⊞ CRC: 0x026c67
```

图 3-9 Sniffer 采集主机 LL 层连接参数更新请求

```
⊟ Bluetooth L2CAP Protocol
    Length: 6
    CID: Low Energy L2CAP Signaling Channel (0x0005)
  ⊟ Command: Connection Parameter Update Response
      Command Code: Connection Parameter Update Response (0x13)
      Command Identifier: 0x02
      Command Length: 2
      Move Result: Accepted (0x0000)
```

图 3-10 Sniffer 采集主机 L2CAP 层连接参数应答

3.2、属性构成

先说一下在 GATT 中的两个角色：服务器和客户端。

- 服务器：提供数据的蓝牙设备
- 客户端：需求数据的蓝牙设备

这里注意，这两个角色在同一个蓝牙设备中可以都存在。

属性是一条公开的带有标签的，可以被寻址的数据。属性构成就是规定数据怎么组成。在 L2CAP 中可以知道，HOST 通往下层的数据只有 3 条路，并且如果是应用的话，只有一条 CID=0x0004 的通道，看来对于 BLE 所有应用数据都是通过 ATT 进行传输的。那么到沙子里面淘金子，为什么可以认出沙子里面的金子呢？因为金子发光，黄色的，这两个就是金子的基本属性。对于 ATT 跑到 GATT 哪里去找数据，肯定得规定数据的怎么放，也就是人为规定数据属性，这样才好找吧！在 4.0 协议规范中规定的这个格式就是属性，也就是数据按照一定规则存放的规定。如图 3-11 所示。

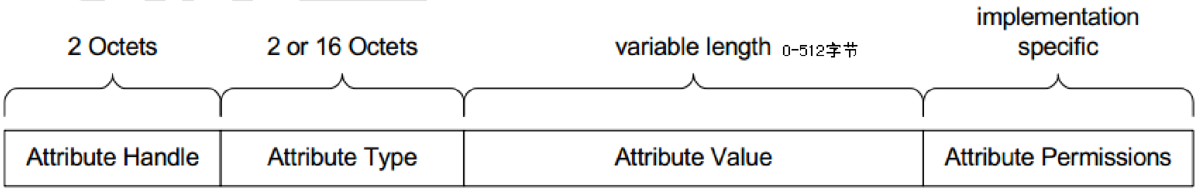


图 3-11 属性组成图

在实际的应用中，程序中存放的就是由多个上图属性组成的一个数据库，它由 4 个部分组成，分别是：属性句柄(Attribute Handle)、属性类型(Attribute Type)、属性长度(Attribute Value)以及属性许可(Attribute Permissions)。还是先贴一张数据库的属性表吧！如表 3-5

所示，这个表是 Sniffer 采集空中心率计的数据，我将其制作成的表，这个表并不是一个完整的心率计的数据库，仅仅是传到了空中的属性数据。在后文中会多次引用这张表，表中的斜体字为解释说明用。

表 3-5　心率计属性数据库

Handle	Attribute Type	Attribute **Value**
0x0001	0x2800 ***«Primary Service»***	0x1800　***«Generic Access Profile »***
0x0003	0x2A00 ***« Device Name Characteristic »***	"Liuquan_HRM"
0x0008	0x2800 ***«Primary Service»***	0x1801　***«Generic Attribute Profile »***
0x0009	0x2803 ***«Characteristic»***	0x20　*属性* 0x000a　*句柄* 0x2A05　*UUID*
0x000a	0x2A05 ***«Service Changed Characteristic»***	
0x000b	0x2902 ***«Client Characteristic Configuration»***	0x0002
0x000c	0x2800 ***«Primary Service»***	0x180D　***«Heart Rate service »***
0x000d	0x2803 ***«Characteristic»***	0x12　*属性* 0x000E　*句柄* 0x2A37　*UUID*
0x000e	0x2A37 ***«Heart Rate Measurement characteristic»***	0x00B4
0x000f	0x2902 ***«Client Characteristic Configuration»***	0x0001
0x0010	0x2803 ***«Characteristic»***	0x02　*属性* 0x0011　*句柄* 0x2A38　*UUID*
0x0011	0x2A38 ***«Body Sensor Location characteristic»***	0x03
0x0012	0x 2800 ***«Primary Service»***	0x180F　***«Battery service »***
0x0013	0x2803 ***«Characteristic»***	0x12　*属性* 0x0014　*句柄* 0x2A19　*UUID*
0x0014	0x2A19 ***«Battery Level characteristic»***	0x64
0x0015	0x2902 ***«Client Characteristic Configuration»***	0x0001
0x0016	0x2800 ***«Primary Service»***	0x180A　***«Device Information service»***

3.2.1、属性句柄(Attribute Handle)

它是两个字节的数据，简单的理解是它仅仅是一个序号。它有两个作用：

- 一方面是为了方便寻找属性的一种方式

举个例子，当去超市购物，将你的包存放在前台时，前台客服将你的包放到一个柜子里，然后给你一个牌子，这个牌子上有一个数字，当你取包裹时，将牌子给客服，他一看上面的数字就知道，你的包放在这个数字对应的柜子里。同样，对于属性放到内存中，也相当于包存放在柜子里，这个属性会自带一个数值，这个数值就是句柄，当其他层要数据时，可以只知道这个句柄，然后将这个句柄对应的 Attribute Type 和 Attribute Value 都找出来。

- 另一方面通过属性句柄操作多个同样属性类型

还是接着超市存包的例子讲，当牌子掉了，数字也不记得了，怎么向客服要你存的包，一般采用说出包的样子，里面放了什么东西，客服一看是对的，他会给你包。但是如果有两个包存放在前台，包和包里装的东西是一样的，怎么办？那么只能通过牌子的数值进行判断了。同样如果在一个蓝牙设备上有两个温度传感器，要读取其中的某个温度值，那么到底是读哪个呢？这时只能通过属性句柄进行判读，读取哪个温度传感器的值。

属性句柄是一个从 0x0001-0xFFFF 的值，0x0000 是保留不用的。属性句柄并没有规定那个值后面必须跟特定 Attribute Type 和

版权所有

Attribute Value，例如表 3-5 中的属性句柄 0x0001 和 0x0008 后面的值是可以交换的，就是看初始化建立数据库时这张表是怎么安排的，当然并不能随意的交换属性，因为单个属性其实是没有什么意义和作用的，就像一块砖头是没有什么用的，但是多个砖头搭建起了房子，作为一个整体就有用了，属性也是多个形成一个组合才对应用有意义，在后文中将会详细讲解。句柄值在数据库创建时，是按顺序递增的，但是并不需要连续递增，0x0003 后面可以是句柄 0x0008。

3.2.2、属性类型(Attribute Type)

属性类型其实就是对某个东西取一个别名，让机器可以理解，对于机器理解东西就是数字了，所以属性类型采用了 2bytes 或者是 16bytes 的长度的数字表示某个东西。例如心率计，我们人听到心率计都知道是什么，但是机器并不知道，同样对于一个不懂中文的人对他说“心率计”，他也不知道是什么。所以为了全球统一，心率计有一个数字代号：0x180D，这是唯一的识别码叫做通用唯一识别码(Universally Unique IDentifier (UUID))。在服务器的数据库中只要找到服务是 0x180D 的值，所有设备都知道这是一个含有心率计服务的蓝牙设备。

对于机器读到 0x180D 就知道是心率计服务，但是对于人来说还是喜欢形象的东西，所以又有一种方法，对于 2bytes 的 UUID，通常不直接用它的值，而是用一个名称并加上书名号，例如心率计通常用**«Heart Rate service »**表示数值为 0x180D 的 UUID。所以在表 3-5 中我

版权所有

都用斜体将 UUID 用书名号的方式进行说明。

那么 2bytes 或者是 16bytes 的 UUID 是什么意思？本来 UUID 是 128bit 长度的数值，但是为了提高传输效率，蓝牙技术联盟(SIG)定义了一个“蓝牙 UUID 基数”，作为 128bits 的 UUID，结合一个较短的 16bits 的数一起使用。这样当发送方发送 16bits 时，接收方收到后补上蓝牙 UUID 基数即可。世界上只要用到低功耗蓝牙的应用，那么必定有一个 UUID 用来给这个东西取一个别名，鞋子、袜子、温度、心率、压力、厘米等等，都需要有 UUID 与之对应。所以采用了 128bits 长度进行慢慢使用。

蓝牙 UUID 基数如下：

00000000-0000-1000-8000-00805F9B34FB

如果要发送心率计服务的 UUID=0x180D，完整的 128bits 的 UUID 为：

0000180D-0000-1000-8000-00805F9B34FB

UUID 的使用方法，对于低功耗蓝牙来说将 UUID 进行了范围规定：

- 0x1800~0x26FF 用于服务类 UUID
- 0x2700~0x27FF 用于标识计量单位
- 0x2800~0x28FF 用于区分属性类型
- 0x2900~0x29FF 用于特性描述
- 0x2A00~0x7FFF 用于区分特性类型

版权所有

3.2.3、属性值(Attribute Value)

属性值是一个 0~512 字节的数据，对于属性本身来说，属性值是没有用的，属性值是给应用用的。那么值可以有哪些呢？

- 服务通用唯一识别码(UUID)
- 单位
- 属性类型
- 特性描述符
- 特性类型

3.2.4、属性许可(Attribute Permissions)

属性许可是仅仅是对属性值的一种保护，对句柄和类型没有用。也就是说对方设备对这个属性值的操作具有什么样的权限，也就是规定了这个属性值的安全级别，例如读写是否需要认证或者需要。**注意：这个属性许可是不能通过属性发现协议获取到的**，只能是获取对方某个属性值时，如果需要什么权限，对方就会发一个状态过来，根据状态进行下一步操作。例如：

如果对安全属性的访问需要经过身份认证的连接，而客户端是在服务器上没有经过验证的安全性，这时服务器发送一个错误代码«Insufficient Authentication»(见表 3-13 错误代码表) 给客户端。当客户端收到这个错误代码，它可能尝试对连接进行身份验证，如果身份验证成功的，就可以访问这个属性值了。

如果对安全属性的访问需要加密连接，而连接不是加密的，服务

器发送错误代码«Insufficient Encryption»(加密不足)给客户端。当客户端收到这个错误代码时，它可能会尝试发起加密请求，如果加密成功，它就可以访问这个属性值。

属性许可本身分 3 种类型(属性权限是访问许可、身份认证许可和授权许可)：

- 访问许可

  访问许可定义有 3 种使用方式：

  - 可读
  - 可写
  - 可读且可写

  当读取属性值时，需要判断这个属性值是否可读。如果不可读。服务器会返回一个属性不可读的状态。当写入属性值时，也需要检测这个属性是否可写，否则会收到属性值不可写的状态信息。

- 身份认证许可

  认证许可有 2 种：

  - 需要认证
  - 不需要认证

  这个认证许可的意思是，是否允许客服端访问服务器，相当于演唱会的门票，要看演唱会，你得给验票员门票。但是如果你进去了上了个厕所再进行看，是不再需要门票的。当客户端访问服务器的属性值，如果需要认证，客户端发出认证请求。

版权所有

当通过了认证，下次访问需要认证的属性，是不需要认证的。如果强行访问需要认证的属性值，客户端会回复一个未认证的错误状态信息。

- 授权许可

授权许可有 2 种：

- ➢ 无授权
- ➢ 授权

这个授权许可的意思是，客服端有没有权利访问服务器的属性值。相当于开的一个粉丝演唱会，只有认为是粉丝的才能进去看。这个授权完全由服务器决定。

再次强调：属性许可不能通过属性协议访问到这个 handle 属性值的权限，这个权限只有当客户端访问服务器时才会将需要访问这个属性权限的条件(错误码)返回给服务器，进而服务器做出相应的处理。

下图采集的属性许可空中数据：

```
995 13.562117000  Slave   Master  ATT    35 Rcvd Error Response - Insufficient Authentication, Handle: 0x0014
996 13.591237000  Master  Slave   SMP    37 Rcvd Pairing Request: Bonding, MITM, Initiator Key(s): LTK IRK , Responder Key(s): LTK IRK
997 13.607478000  Slave   Master  LE LL  26 Empty PDU
998 13.621238000  Master  Slave   LE LL  26 Empty PDU
999 13.622383000  Slave   Master  SMP    37 Rcvd Pairing Response: Bonding, No MITM, Initiator Key(s): IRK , Responder Key(s): LTK

⊞ Frame 995: 35 bytes on wire (280 bits), 35 bytes captured (280 bits) on interface 0
⊞ Nordic BLE sniffer meta
⊞ Bluetooth Low Energy Link Layer
⊟ Bluetooth L2CAP Protocol
    Length: 5
    CID: Attribute Protocol (0x0004)
⊟ Bluetooth Attribute Protocol
    Opcode: Error Response (0x01)
    Request Opcode in Error: Read Request (0x0a)
    Handle in Error: 0x0014
    Error Code: Insufficient Authentication (0x05)

0000  04 06 1c 01 0e 69 06 0a  31 1e 3f 4b 01 97 00 00   .....i.. 1.?K....
0010  00 a1 7b 65 50 06 09 05  00 04 00 01 0a 14 00 05   ..{eP... ........
0020  ea 75 65                                           .ue
```

3.3、GATT 服务器构成

一个 BLE 的数据的库的组成如图 3-12 GATT Profile 分层。

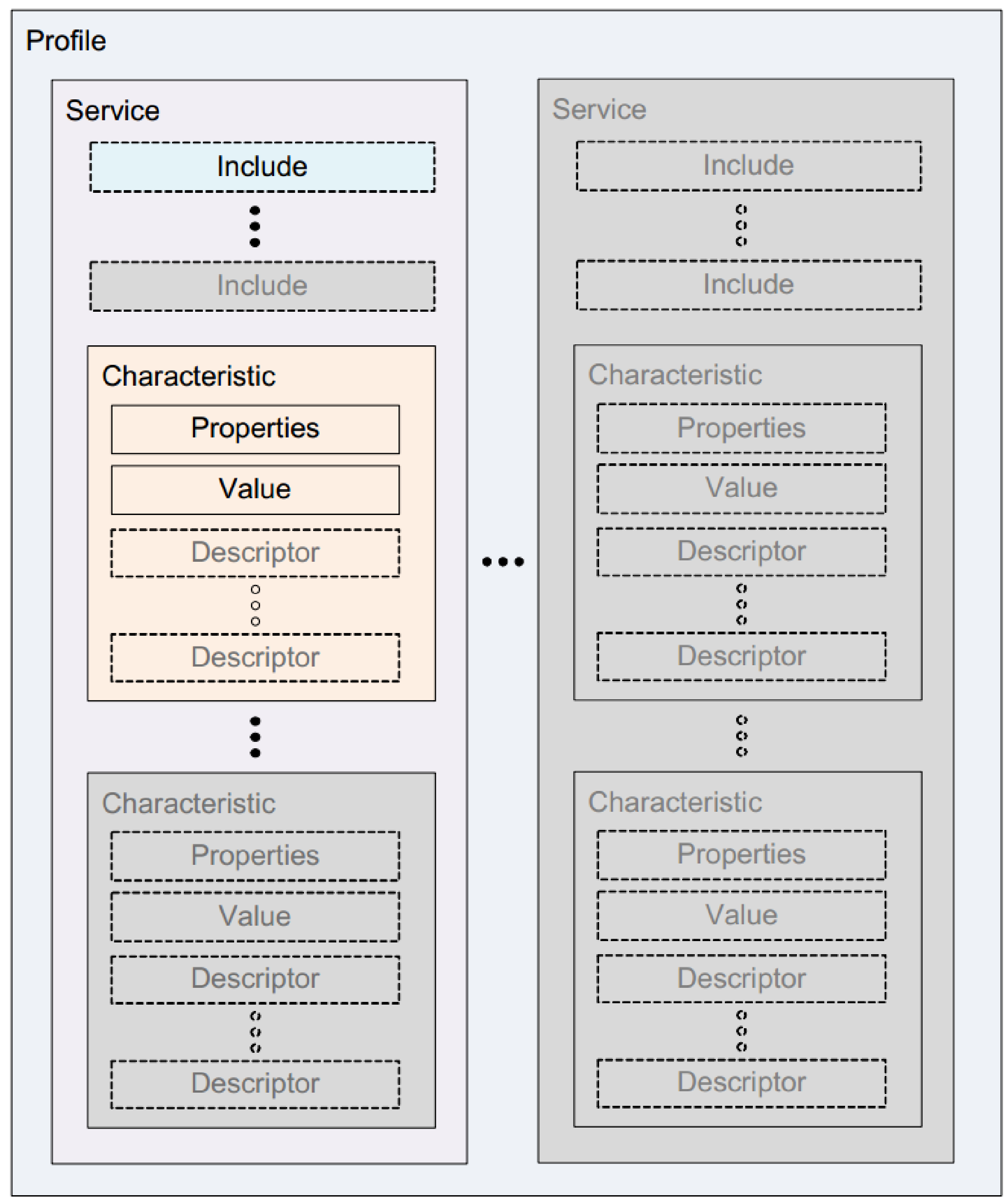


图 3-12 GATT Profile 分层

为什么先要讲这个图呢？目的是为了有一个对服务器有一个整体的了解。从图中可知，一个服务器中的数据库其实就是 Profile，在

表 3-1 中也有形象描述。Profile 中有多个 Server 组成，而 Server 的由 0 或多个《Include》和至少一个《Characteristic》组成。例如，在表 3-5 中的《Heart Rate service 》心率计服务，可以描述如表 3-6 所示。注意下图的 Value 并不是 Attribute Value，它是«Characteristic»的值。

表 3-6 心率计服务分层

0x180D***«Heart Rate service »*** ***心率计服务***

0x2803***«Characteristic»***

Properties	0x12	
Value	0x000E	*句柄*
	0x2A37	*UUID*
Declaration	0x2A37 ***«Heart Rate Measurement characteristic»***	0x00B4
Descriptor	0x2902 ***«Client Characteristic Configuration»***	0x0001

0x2803***«Characteristic»***

Properties	0x02	
Value	0x0011	*句柄*
	0x2A38	*UUID*
Declaration	0x2A38 ***«Body Sensor Location characteristic»***	0x03

3.3.1、服务

那么什么是服务呢？协议规范中是这么描述的：

A service is a collection of data and associated behaviors to accomplish a particular function or feature.In GATT,A service definition may contain referenced services, mandatory characteristics and optional characteristics.

我的理解是服务是指一系列由数据和相关行为组成的集合，为了去完成某个特定的功能或者特性。而一个服务可以包含引用服务即《Include》、强制性和可选的特征即《Characteristic》。

上面仅仅是对服务的抽象解释，具体形象描述是：一个服务将包含一个服务声明和 0 个或多个包含定义以及至少一个特性定义。服务定义的开始为服务声明，结束于下一个服务声明或者句柄达到 0xFFFF。所有的包含定义将紧跟服务声明并在特性定义之前，所有的特定定义将紧跟在最后一个包含定义，当没有包含定义时，特性定义紧跟在服务声明之后。

上面讲的是服务的定义，提到有服务声明、包含定义、特性定义。这到底是什么东西呢？在表 3-1 中有提到，Profile 就是做模子的，那么肯定对这些都有定义。

3.3.1.1、服务声明

服务声明是干什么用的？简单地讲，就是告诉其他蓝牙设备，我可以为你提供什么服务，对于心率计，就是告诉对方我可以提供心率的服务。

从表 3-5 中可以看出句柄 0x000C~0x0011 是一个**«Heart Rate service »**心率计服务；句柄 0x0012~0x0015 是**«Battery service »**电池服务。人很容易读懂表 3-5，因为**«Heart Rate service »**很容易认识，但是对于芯片，怎么让它知道一个服务的开始和结束呢？这就要规定当

读到什么样的值，这个值就一定代表某个服务，也就是服务声明。那么这个值是什么值呢？怎么定义的呢？从属性格式图 3-11 中可以知道在 GATT 层中能用的数据只有属性类型(Attribute Type)，因为属性值是给应用层使用的，也就是说如果要在 GATT 层能知道某个属性是干什么的，那么是不是应该有一个共识，那就是这个属性类型(Attribute Type)的 UUID 值应该是一个特定的值，这个值就代表着就是服务声明。在表 3-5 中句柄为 0x000C 的 UUID=0x2800，这个值就是服务声明，它声明一个**«Heart Rate service »**心率计服务。这就印证着"属性类型"的名字，这个 UUID 值是区分属性用的。实际上在 GATT Profile 中的属性类型有多个，如图 3-13 所示。

Attribute Type	UUID	Description
«Primary Service»	0x2800	Primary Service Declaration
«Secondary Service»	0x2801	Secondary Service Declaration
«Include»	0x2802	Include Declaration
«Characteristic»	0x2803	Characteristic Declaration
«Characteristic Extended Properties»	0x2900	Characteristic Extended Properties
«Characteristic User Description»	0x2901	Characteristic User Description Descriptor
«Client Characteristic Configuration»	0x2902	Client Characteristic Configuration Descriptor
«Server Characteristic Configuration»	0x2903	Server Characteristic Configuration Descriptor
«Characteristic For-mat»	0x2904	Characteristic Format Descriptor
«Characteristic Aggre-gate Format»	0x2905	Characteristic Aggregate Format Descriptor

图 3-13 Summary of GATT Profile Attribute types

上图中可以看到，在服务声明中有两个属性类型：《首要服务》«Primary Service»和《次要服务》«Secondary Service»，那么这两个有

什么区别呢？

- 《首要服务》«Primary Service»

公开设备功能的服务通常为首要服务。例如有一台支持心率计的蓝牙设备，那么心率计将被实例化为首要服务。

- 《次要服务》«Secondary Service»

不需要被客户端直接用或者不需要客户端理解的行为和特性的封装。次要服务是不需要被其他设备知道的，次要服务相当于公司的研发人员，而首要服务相当于销售人员，客户问销售要什么产品的报价什么，销售可以直接给客户，但是如果客户需要产品的具体性能指标参数，那么销售就只能找研发要了之后再给客户。也就是说次要服务只能被首要服务引用，也就是只能被首要服务用《Include》进行包含来引用。

3.3.1.1.1、服务声明格式

上面知道服务声明是通过特定的 UUID 值进行判断，那么到底服务声明有什么格式呢？如图 3-14 所示。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

图3-14 服务声明

为什么它的 Attribute Value 是一个 UUID 值呢？这个也好理解，它的工作就是进行服务声明，也就是说它的工作是专门告诉别的设备

自己具有什么服务，而真正的服务也是被 SIG 通过认证了的 UUID 或者是自己私有 UUID 值，通俗点就是说声明服务其实就是用用特定的 UUID 公开一个 UUID 值，这个 UUID 可以是 16bits 或者是 128bytes。在表 3-5 中，可以找到 5 个服务声明，并且都是首要服务，如表 3-7 所示。

表 3-7 心率计中首要服务声明

Handle	UUID	Value	
0x0001	0x2800 ***«Primary Service»***	0x1800	***«Generic Access Profile »***
0x0008	0x2800 ***«Primary Service»***	0x1801	***«Generic Attribute Profile »***
0x000c	0x2800 ***«Primary Service»***	0x180D	***«Heart Rate service »***
0x0012	0x 2800 ***«Primary Service»***	0x180F	***«Battery service »***
0x0016	0x2800 ***«Primary Service»***	0x180A	***«Device Information service»***

3.3.2、包含服务«Include»

包含服务，就相当于在写程序是要用别的文件中的函数，可以通过#include"XXXXX.h"进行别的头文件的包含。同样当一个服务需要需要用到别的服务里面的某些值的时候，也可以通过«Include»来完成。然而，«Include»一定是在服务声明之后的，那么服务声明有两种方式，首要服务可以引用另一个首要服务。首要服务也可以引用一个次要服务，从而使用次要服务公开的行为。次要服务可以引用一个次要服务或者首要服务。不过次要服务引用次要服务情况很少，次要引用主要服务就更少了。那么包含服务«Include»的声明格式是怎么样的呢？如图 3-15 所示。

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

图3-15 包含服务声明格式

从图 3-13 中可以知道包含服务《Include》本身的 UUID=0x2802。它的属性值共有 3 个部分组成：需要包含的服务的句柄、组结束的句柄和需要包含的服务的 UUID。当然最后一个参数并不是一定有的，当 UUID 为 128bits 时，服务 UUID 将不会作为声明值的一部分。如表 3-8 所示。句柄 0x0281 的属性类型为《Include》类型，它声明包含的服务是《Manufacturer Service》即厂商服务，服务句柄为 0x0500，组结束的句柄为 0x0504。

表3-8 包含声明实例

Handle	Attribute Type	Attribute Value
0x0280	«Primary Service»	«Weight Service»
0x0281	«Include»	{0x0505, 0x0509, «Manufacturer Service»}
0x0282	«Characteristic»	{0x02, 0x0283, «Weight Kg»}
0x0283	«Weight Kg»	0x00005582
0x0284	«Characteristic Format»	{0x08, 0xFD, «Kilogram», «Bluetooth SIG», «Hanging»}
0x0285	«Characteristic User Description»	“Rucksack Weight”
0x0500	«Secondary Service»	«Manufacturer Service»
0x0501	«Characteristic»	{0x02, 0x0502, «Manufacturer Name»}
0x0502	«Manufacturer Name»	“ACME Temperature Sensor”
0x0503	«Characteristic»	{0x02, 0x0504, «Serial Number»}
0x0504	«Serial Number»	“237495-3282-A”

3.3.3、属性类型分组

上面有提到包含服务《Include》中包含有组结束的句柄，那么什么是组呢？怎么定义的？在 GATT Profile 中将属性类型分为 3 组：«

首要服务»«Primary Service»,«次要服务» «Secondary Service»和«特性»«Characteristic»。一个组开始于一个声明，服务分组结束于下一个服务声明，特性分组结束于下一个特性的声明或者是下一个服务的声明。也就是服务声明对服务进行分组，特性声明对特性进行分组。服务是一种或多种特性的组合；特性则是由一种或多种属性组成。

3.3.4、特性«Characteristic»

应用时用的是属性的值，而应用中使用的属性值是特性分组中的，所以特性说明白真的很难。特性包含 3 个基本的要素：

- 特性声明
- 声明的属性的值的声明
- 特性描述符

表 3-9 是截取的表 3-5 中心率计的第一个特性分组。它刚好是一个最小的特性。从表中可以看到，特性声明后紧接着的是特性声明属性的值，怎么理解？可以发现特性声明中的属性值包含的句柄 0x000e 和属性类型 0x2A37 刚好是特性声明属性的值声明中的句柄和属性类型，也就是仅仅公开特性声明中属性的属性值。

表 3-9 心率计的特性

特性要素	Handle	Attribute Type	Attribute Value
特性声明	0x000d	0x2803 ***«Characteristic»***	0x12 *性质* 0x000E *句柄* 0x2A37 *UUID*
特性声明属性的值声明	0x000e	0x2A37 ***«Heart Rate Measurement characteristic»***	0x00B4
特性描述符	0x000f	0x2902 ***«Client Characteristic Configuration»***	0x0001

版权所有

3.3.4.1、特性声明

从图 3-13 中可以知道，特性声明本身的 UUID 值是 0x2803，特性声明中需要声明的特性是在属性值中的，属性值包含有 3 个字段：特性性质、属性句柄和属性类型，且仅为只读。如图 3-16 所示。

Attribute Handle	Attribute Type s	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Character-istic Proper-ties	Character-istic Value Attribute Handle	Characteris-tic UUID	Read Only, No Authentication, No Authorization

图 3-16 特性声明格式

特性声明中的属性值的 3 个字段中特性性质占 1 个字节，属性句柄占 2 个字节，属性类型要么是 2 字节 16bits 的 UUID 或者是 16 字节 128bits 的 UUID。如表 3-9 的特性声明。

3.3.4.1.1、属性值—特性性质(Characteristic Properties)

特性性质是一个 8 位字段，确定了特性数值属性对一系列操作的支持。也就是说通过这个字节可以知道声明的这个属性中的属性值可以被怎么操作，可以用什么“手段”进行操作，到底什么手段就在属性协议 ATT 中了。如表 3-10 所示。

表 3-10 特性性质

性质	值	描述
广播	0x01	如果设置该位，就必须有服务器的特性配置描述符«Server Characteristic Configuration»
读取	0x02	这个属性值是可通过读取规程进行读取的
写命令(无应答写)	0x04	这个属性值可以接受命令规程进行命令操作
写值(有应答)	0x08	这个属性值可以通过写值规程进行写操作，并有应答

通知	0x10	服务器可以主动通知属性值给客户端
指示	0x20	服务器将属性值指示给客户端，并得到客户端的确认
加密写命令	0x40	加密后写命令
扩展属性	0x80	说明这是一个扩展属性

3.3.4.1.2、属性值—特性的属性句柄(Characteristic Value Attribute Handle)

属性句柄就是特性声明中被声明的属性的句柄。

3.3.4.1.3、属性值—特性的属性类型(Characteristic UUID)

属性类型就是特性声明中被声明的属性的属性类型值即 UUID。

3.3.4.2、特性值声明

特性值的声明是在特性声明后的第一个属性，所有的特性定义中一定包含有一个特性值声明。它的声明格式如图 3-17 所示。例子见表 3-8 中的特性声明属性的值声明。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

图 3-17 特性值声明

对于每一个特性，对应的说明文档(Profile)描述了特性的格式。特性本身不包含行为，如果要确定某个特性实例公开的行为，必须查阅描述该特性的服务规格书即对于的 Profile。

3.3.4.3、特性描述符声明

特性描述是用来包含一些关于特性值的关联信息，特性描述有多种类型，一个特性的定义可以有任意多个的描述符，而所有描述符都是用来为特性值服务的，具体用到了哪些描述符需要根据相应的 Profile 进行确定。描述符跟随在特性值声明之后，如果有多种描述符进行声明，其次序是没有规定的。

特性可以包含如下 6 种描述符的组合，结合图 3-13 得到其属性类型的 UUID 值：

- 特性扩展性质 **«Characteristic Extended Properties»：0x2900**
- 特性用户描述 **«Characteristic User Description» ：0x2901**
- 客户端特性配置**«Client Characteristic Configuration» ：0x2902**
- 服务器特性配置**«Server Characteristic Configuration»：0x2903**
- 特性表示格式 **«Characteristic Presentation Format»：0x2904**
- 特性聚合格式 **«Characteristic Aggregate Format» ：0x2905**

3.3.4.3.1、特性扩展性质描述符

该描述符用来获得附加的扩展性质。声明格式如图 3-18 所示。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2900 – UUID for «Characteristic Extended Properties»	Characteristic Extended Properties Bit Field	Read Only, No Authentication, No Authorization

图 3-18 特性扩展性质描述符声明

属性值是 2 个字节的数据，它的每个 bit 表示一个扩展的性质，它主要用来描述怎么用属性值或者怎么访问特性描述。目前定义了两类扩展性质，如表 3-11 所示。

表 3-11 Characteristic Extended Properties bit field

性质	值	描述
Reliable	0x0001	可靠地写入数值的能力
Writable Auxiliaries	0x0002	写入“特性用户描述”描述符的能力

3.3.4.3.2、特性用户描述描述符

按字面意思理解：可以给用户进行配置的描述性的东西，它用一个字符串与某一特性相关联，让人活着其他设备更容易理解。例如用户可以配置恒温装置，使其用”在办公楼 20 楼”进行描述，那么人读到这样的描述一下就知道这个恒温装置在 20 楼。它的声明格式如图 3-19 所示。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2901 – UUID for «Characteristic User Description»	Characteristic User Description UTF-8 String	Higher layer profile or implementation specific

图 3-19 特性用户描述描述符声明

3.3.4.3.3、客户端特性配置描述符

支持通知或者指示的特性必须使用客户端配置描述符。它的声明格式如图 3-20 所示。该描述符是一个 2bit 的数值，分别用于设置通知与指示，但是不允许同时设置。关于如何通知或者指示规范中并没

有规定，而是在 Profile 中进行定义的。例如心率计中的心率是服务器通知给客户端的，那么是多长时间通知？还是心率变化了通知？都是由 Profile 规定的。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	02902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

图 3-20 客户端特性配置描述符声明

图 3-21 为属性值 bit 说明。

Configuration	Value	Description
Notification	0x0001	The Characteristic Value shall be notified.
Indication	0x0002	The Characteristic Value shall be indicated.
Reserved for Future Use	0xFFF4	Reserved for future use.

图 3-21 Client Characteristic Configuration bit field definition

在表 3-5 中就有对心率计值的通知。

3.3.4.3.4、服务器特性配置描述符

该描述符与客户端特性配置描述符类似，如图 3-22 声明结构所示，它包含的属性值是一个 bit 位(图 3-23)，意思是否使设备广播该特性所属服务的相关数据。广播的时间有服务器决定。

注意，单个的特性是不能进行广播的，只能由包含该特性的服务来定义所要广播的数据，那么所要广播的数据是什么呢？就是特性值，也就是特性值声明中的属性值。一些服务有可能将多种特性进行广播；

服务也同时定义了接收方如何识别广播中的各种特性。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2903 – UUID for «Server Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

图 3-22 服务器特性配置描述符

Configuration	Value	Description
Broadcast	0x0001	The Characteristic Value shall be broadcast when the server is in the broadcast procedure if advertising data resources are available.
Reserved for Future Use	0xFFF2	Reserved for future use.

图 3-23 Server Characteristic Configuration bit field definition

这里有个问题：为什么在特性声明«Characteristic»中也有一个 bit 用来启动该特性的广播？因为特性值本身是没有任何意义的，它脱离了服务，广播的特性值对其他设备没有任何意义。例如，假如广播特性值是“温度：37.5℃”这样的数据根本就没有什么意义，而如果放到“体温计”这个服务中，“体温：37.5℃”这样的数据才能说明温度服务的对象。

3.3.4.3.5、特性表示格式描述符

使用通用属性规范(GATT)的目的之一是支持通用客户端。通用客户端定义：它能读取某特性的数值，并在无须理解它们的含义的情况下向用户显示这些值。特性表示格式描述是对特性值进行深层次的解

释给通用客户端的用户用的。它的声明结构如图 3-24 所示。

Attribute Handle	Attribute Type	Attribute Value					Attribute Permissions
0xNNNN	0x2904 – UUID for «Character-istic Format»	Format	Exponent	Unit	Name Space	Description	Read only No Authentication, NO authorization

图 3-24 特性表示格式描述符

它的属性值具有 5 个字段：

- 格式：1 字节

这个字节说明特性值的数据类型，就像程序设计中 char 类型和 int 类型，只是这里的数据格式有非常多种。具体阅读协议规范。

- 指数：1 字节

指数可以很好的控制小数点的位置，实际值=特性值*$10^{\text{指数}}$。假设特性值是 3，表示格式是无符号的 16 位整数，指数为 2，那么实际值就是 300；如果指数为-2，特性值是 2345，那么实际值就是 2.345。

- 单位：2 字节

单位就是指特性值的单位，如果是温度计，都知道是℃。但是机器是不知道的，所以要告诉它。

- 命名空间：1 字节
- 描述：2 字节

上面两个字段是联合使用的。这两个字段携带了数值的附加信息。命名空间指明了控制描述字段的机构。描述字段则是一个 16 位的无符号数。这两个的字段数值的意思定义在 Assigned Numbers document 文档中。

版权所有

3.3.4.3.6、特性聚合格式描述符

有些特性值远比单一的数据复杂得多。譬如用标准的符号来表示地球上的某个位置，通常包含经度和纬度值，二者组合在一起便构成一个“坐标值”。为了构造类似的复杂的特性值，特性聚合格式描述符允许引用多个表示格式描述符来解释组合值的每个字段。它的声明格式如图 3-25 所示。

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2905 – UUID for «Characteristic Aggregate Format»	List of *Attribute Handles* for the Characteristic Presentation Format Declarations	Read only No authentication No authorization

图 3-25 特性聚合格式描述符

使用上面的例子，该坐标特性具有两个特性表示格式描述符(分别描述经度和纬度)和特性聚合格式描述符，后者用正确的顺序引用了前两者。于是通用客户端便能正确地解读该特性值的格式并将数值显示给用户。

要注意的是，特性聚合格式描述符所引用的多个特性表示格式描述符不必从属于同一个特性。它们甚至可以来自不同的服务和设备。引用它们只是为了表达特定的格式；与它们自身所属的特性和聚合格式无关。

3.4、属性协议(ATT)

上节是通过 Profile 搭建数据库，这节定义通信协议进行数据库的数据读写操作。通信协议主要 3 个方面：通信方式、通信包格式、具体通信数据包。

3.4.1、通信协议方法

属性协议主要用来发现、读写、通知和指示属性。具体如表 3-12。

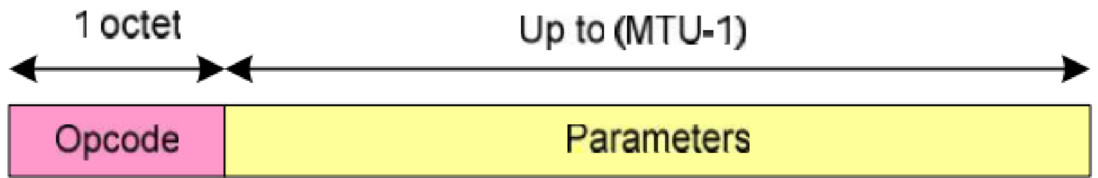
表 3-12 属性协议通信方式

PDU 类型		Send By	描述
请求	Request	Client	客户端向服务器请求数据
应答	Response	Server	服务器对上面请求的应答
命令	Command	Client	客户端向服务器发送命令--无应答
通知	Notification	Server	服务器对特性数值向客户端通知—无应答
指示	Indication	Server	服务器对特性数值指示给客户端
确认	Confirmation	Client	客户端对指示的应答

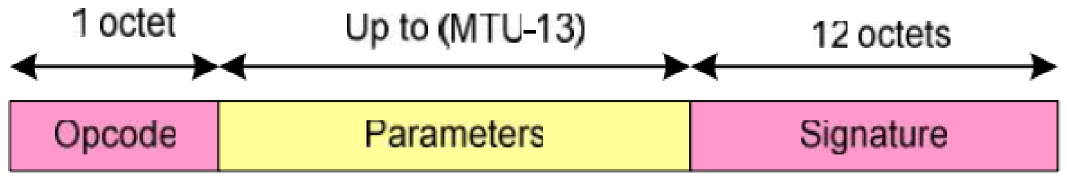
这里涉及到“原子操作”，上面的通信方式中 Request 和 Response 是一对，Indication 和 Confirmation 是一对，在没有得到对方的应答或者确认信息之前是不能进行第 2 包的请求或者指示数据。命令和通知因为没有应答信息，所以可以在任何时候进行包传输。

3.4.2、属性协议包格式

协议包格式如图 3-26 所示。包含有 1 字节的操作码和其他参数，其他参数有两种格式，一种是不包含加密认证信息的参数，另一种是带有 12 字节的加密认证信息的协议包。



ATT PDU if without signature



ATT PDU if with signature

图 3-26 属性协议包格式

需要注意的是，如果链路本身是加密的，那么包含认证标示的包将不被发送。因为加密链路已经进行一次安全认证，不需要再次进行认证。协议原文如下：

> An Attribute PDU that includes an Authentication Signature should not be sent on an encrypted link. Note: an encrypted link already includes authentication data on every packet and therefore adding more authentication data is not required.

3.4.3、属性协议 PDUs

属性协议数据包，是用来对属性数据库操作的实体。这一节几乎全部用图来描述。具体怎么发送数据库中的服务，下节中会讲到 ATT 协议怎么对应 GATT 服务发现。

3.4.3.1、交换 MTU

图 3-27 为交换 MTU 包格式。图 3-28 为交换 MTU 示意。图 3-29 为 Sniffer 采集空中交换 MTU 数据。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x02 = Exchange MTU Request
Client Rx MTU	2	Client receive MTU size

Format of Exchange MTU Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x03 = Exchange MTU Response
Server Rx MTU	2	Attribute server receive MTU size

Format of Exchange MTU Response

图 3-27 交换 MTU 包格式

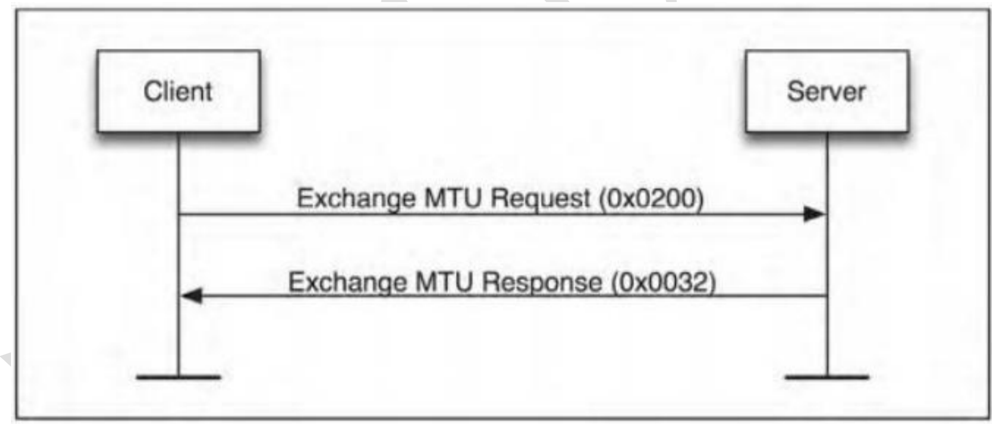


图 3-28 交换 MTU 示意

```
Bluetooth Attribute Protocol
    Opcode: Exchange MTU Request (0x02)
    Client Rx MTU: 158

Bluetooth Attribute Protocol
    Opcode: Exchange MTU Response (0x03)
    Server Rx MTU: 23
```

图 3-29 Sniffer 采集空中交换 MTU 数据

在低功耗蓝牙连接中，属性协议默认的 MTU 为 23 字节。一般客户端是不会发起这个请求的，它的默认值就是 23，当双方交换的值不同时，用较小的那个作为最终使用的值。

3.4.3.2、找信息请求\应答(Find Information Request\Response)

图 3-30 为找信息请求\应答包格式。图 3-31 为找信息请求\应答示意。图 3-32 为 Sniffer 采集空中找信息请求\应答数据。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x04 = Find Information Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number

Format of Find Information Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x05 = Find Information Response
Format	1	The format of the information data.
Information Data	4 to (ATT_MTU–2)	The information data whose format is determined by the Format field

Format of Find Information Response

Name	Format	Description
Handle(s) and 16-bit Bluetooth UUID(s)	0x01	A list of 1 or more handles with their 16-bit Bluetooth UUIDs
Handle(s) and 128-bit UUID(s)	0x02	A list of 1 or more handles with their 128-bit UUIDs

Format field values

Handle	16-bit Bluetooth UUID
2 octets	2 octets

Format 0x01 - handle and 16-bit Bluetooth UUIDs

Handle	128-bit UUID
2 octets	16 octets

Format 0x02 - handle and 128-bit UUIDs

图 3-30 找信息请求\应答包格式

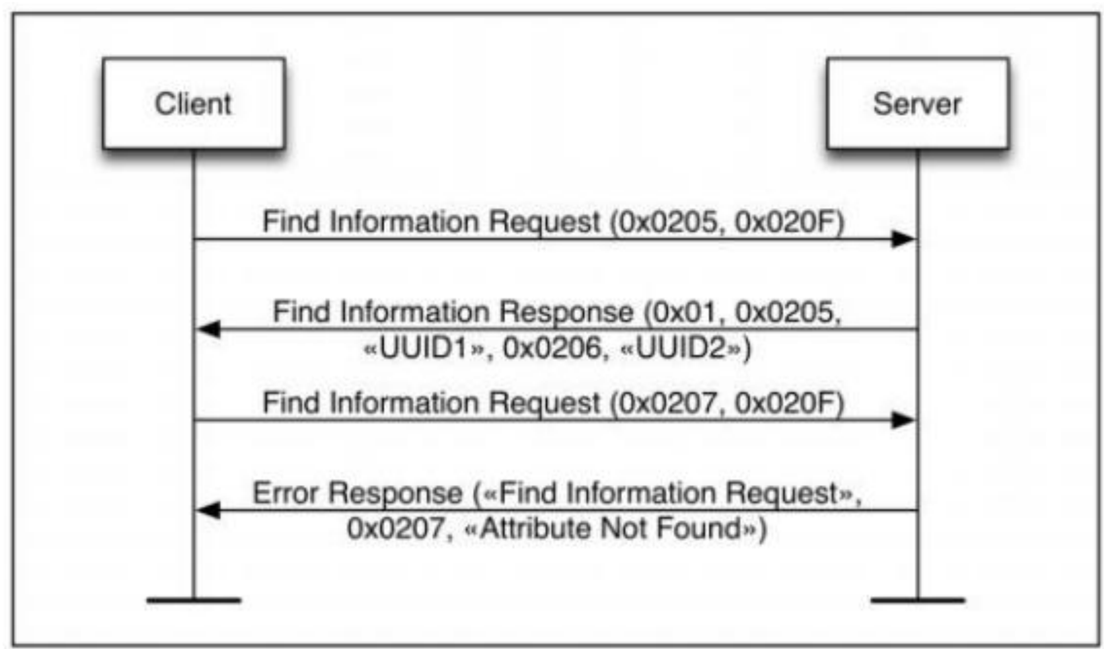


图 3-31 找信息请求\应答示意

```
Bluetooth Attribute Protocol
  Opcode: Find Information Request (0x04)
  Starting Handle: 0x000b
  Ending Handle: 0x000b
Bluetooth Attribute Protocol
  Opcode: Find Information Response (0x05)
  UUID Format: 16-bit UUIDs (0x01)
  Handle: 0x000b
  UUID: Client Characteristic Configuration (0x2902)

Bluetooth Attribute Protocol
  Opcode: Find Information Request (0x04)
  Starting Handle: 0x0018
  Ending Handle: 0xffff
Bluetooth Attribute Protocol
  Opcode: Find Information Response (0x05)
  UUID Format: 16-bit UUIDs (0x01)
  Handle: 0x0018
  UUID: Client Characteristic Configuration (0x2902)
```

图 3-32 Sniffer 采集空中找信息请求\应答数据

查找信息请求和回复用来查找一系列属性的句柄和类型信息。这是唯一一个能让客户端发现任意属性类型的消息。

查找信息请求包含有两个句柄：起始句柄和结束句柄。它们定义了该请求用到的属性句柄范围。为了找到所有数值的属性，该请求的起始句柄将是 0x0001，结束句柄设为 0xFFFF。但是回复的信息中因为长度限制，并不能包含所有的属性，所以需要再次请求，只是将起始句柄改为查找到的最大句柄的后一个句柄开始。

查找信息响应包含句柄-类型对。它有两种格式，一种是基于 16 位的 UUID 格式，允许在一个包中最多包含有 5 个属性句柄-类型对；另一种是基于 128 位的 UUID，允许包含有 1 个属性句柄-类型对。在同一应答包中不能包含有这两种格式。

3.4.3.3、按类型值查找请求\应答(Find By Type Value Request\Response)

图 3-33 为按类型值查找请求\应答包格式。图 3-34 为按类型值查找请求\应答示意。图 3-35 为 Sniffer 采集空中按类型值查找请求\应答数据。

按类型值查找请求和回复可以根据给定的类型与数值查找相应的属性。该请求包含有两个句柄：起始句柄和结束句柄，规定查找范围。对于这一范围类的所有属性，如果和请求中所指定的类型和数值一样，那么这个属性就要在响应中返回。

版权所有

Parameter	Size (octets)	Description
Attribute Opcode	1	0x06 = Find By Type Value Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Type	2	2 octet UUID to find
Attribute Value	0 to (ATT_MTU−7)	Attribute value to find

Format of Find By Type Value Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x07 = Find By Type Value Response
Handles Information List	to (ATT_MTU−1)	A list of 1 or more Handle Informations.

Format of Find By Type Value Response

Found Attribute Handle	Group End Handle
2 octets	2 octets

Format of the Handles Information

图 3-33 按类型值查找请求\应答包格式

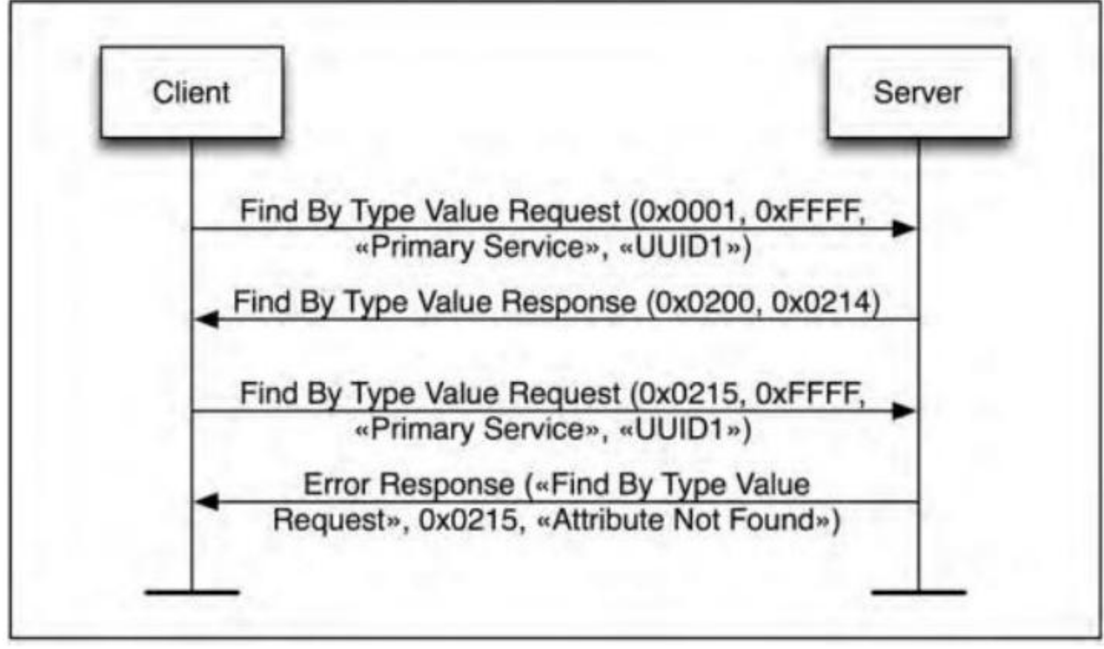


图 3-34 按类型值查找请求\应答示意

```
Bluetooth Attribute Protocol
  Opcode: Find By Type Value Request (0x06)
  Starting Handle: 0x0001
  Ending Handle: 0xffff
  UUID: GATT Primary Service Declaration (0x2800)
  Value: 0218
Bluetooth Attribute Protocol
  Opcode: Find By Type Value Response (0x07)
⊟ Handles Info, Handle: 0x0033, Group End Handle: 0x0036
    Handle: 0x0033
    Group End Handle: 0x0036
```

图 3-35 Sniffer 采集空中按类型值查找请求\应答数据

这个命令主要用来查找特定的首要服务。发送请求时，客户端将其中的类型设置为首要服务，并将数值设为该服务的 UUID。随后的响应将包含查找到的各个首要服务的实例的句柄范围。某些特殊的服务只会在服务器中实现一次，针对它们的响应只会包含一个句柄范围。

3.4.3.4、按类型读请求\应答(Read By Type Request\Response)

图 3-36 为按类型读请求\应答包格式。图 3-37 为按类型读请求\应答示意。图 3-38 为 Sniffer 采集空中按类型读请求\应答数据。

这个命令能在句柄范围内读取某个属性值。当客户端仅知道属性的类型而非句柄时可以使用该请求。请求包含有起始、结束句柄和需要读取的属性的类型。响应将给出符合的句柄和数值。

这个请求用于搜索被包含的服务，并通过特性类型来发现服务中的所有的特性。它也被用来读取已知类型的特性值。

版权所有

Parameter	Size (octets)	Description
Attribute Opcode	1	0x08 = Read By Type Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Type	2 or 16	2 or 16 octet UUID

Format of Read By Type Request

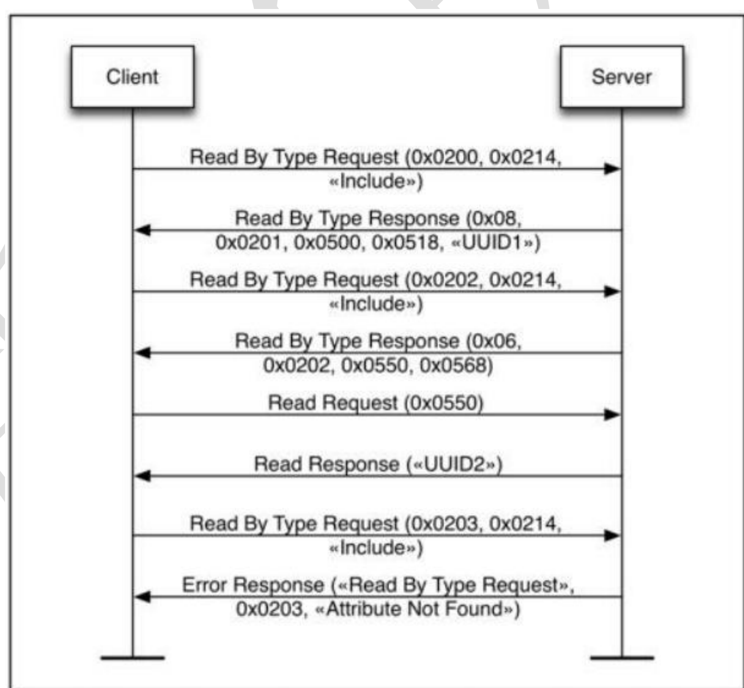
Parameter	Size (octets)	Description
Attribute Opcode	1	0x09 = Read By Type Response
Length	1	The size of each attribute handle-value pair
Attribute Data List	2 to (ATT_MTU– 2)	A list of Attribute Data.

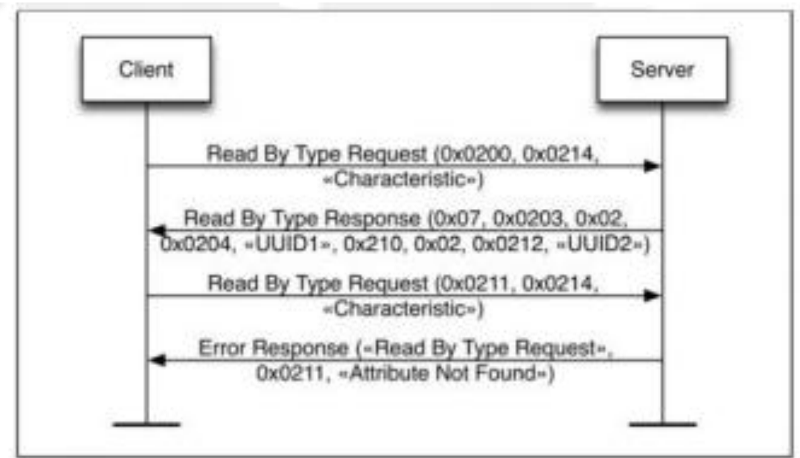
Format of Read By Type Response

Attribute Handle	Attribute Value
2 octets	(Length – 2) octets

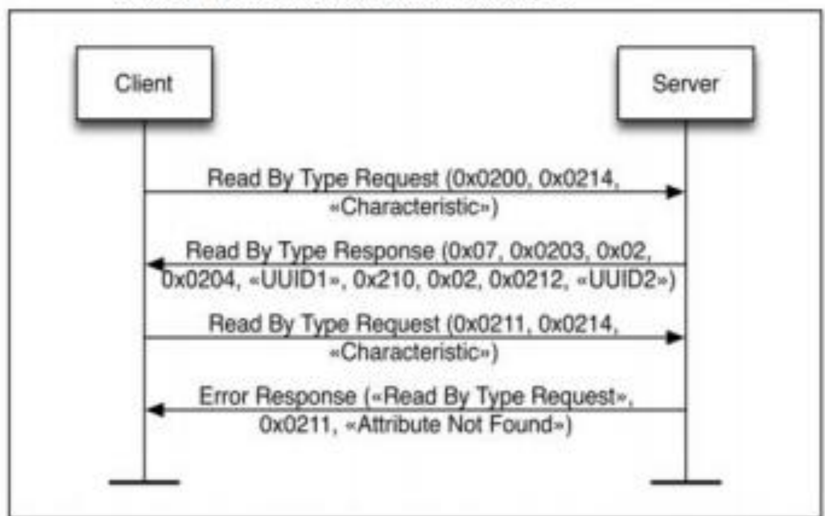
Format of the Attribute Data

图 3-36 按类型读请求\应答包格式

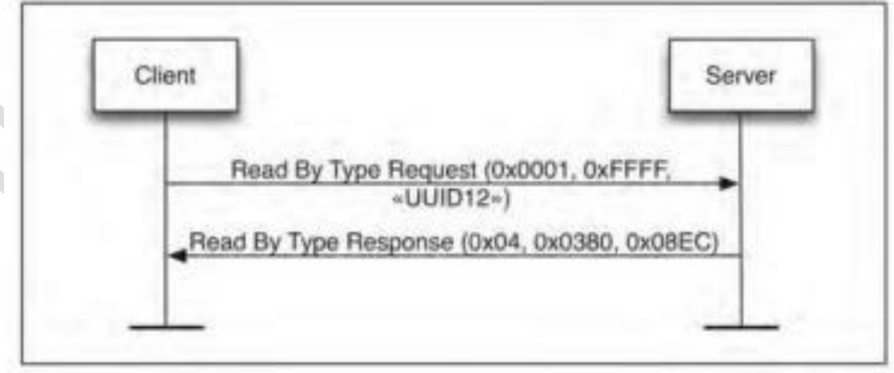




Discover All Characteristics of a Service example



Discover Characteristics by UUID example



Read Using Characteristic UUID example

图 3-37 按类型读请求\应答示意

```
Bluetooth Attribute Protocol
   Opcode: Read By Type Request (0x08)
   Starting Handle: 0x0018
   Ending Handle: 0xffff
   UUID: GATT Characteristic Declaration (0x2803)

Bluetooth Attribute Protocol
   Opcode: Error Response (0x01)
   Request Opcode in Error: Read By Type Request (0x08)
   Handle in Error: 0x0018
   Error Code: Attribute Not Found (0x0a)

Bluetooth Attribute Protocol
   Opcode: Read By Type Request (0x08)
   Starting Handle: 0x0001
   Ending Handle: 0x0007
   UUID: Device Name (0x2a00)

Bluetooth Attribute Protocol
   Opcode: Read By Type Response (0x09)
   Length: 13
⊟  Attribute Data, Handle: 0x0003
      Handle: 0x0003
      Value: 4e6f726469635f50726f78

00  07 06 26 01 f8 14 06 0a  3d 0e 47 62 00 b7 00 00   ..&..... =.Gb....
10  00 d5 45 65 50 0a 13 0f  00 04 00 09 0d 03 00 4e   ..EeP... .......N
20  6f 72 64 69 63 5f 50 72  6f 78 0a ae ab            ordic_Pr ox...
```

图 3-38 Sniffer 采集空中按类型读请求\应答数据

3.4.3.5、读请求\应答(Read Request\Response)

图 3-39 为读请求\应答格式。图 3-40 为读请求\应答示意。图 3-41 为 Sniffer 采集空中读请求\应答数据。

读取请求是属性协议总最简单的请求。该请求包含一个句柄，响应将返回该句柄对应的属性值。只有在客户端已知属性句柄的情况下，才能使用该请求读取属性值。

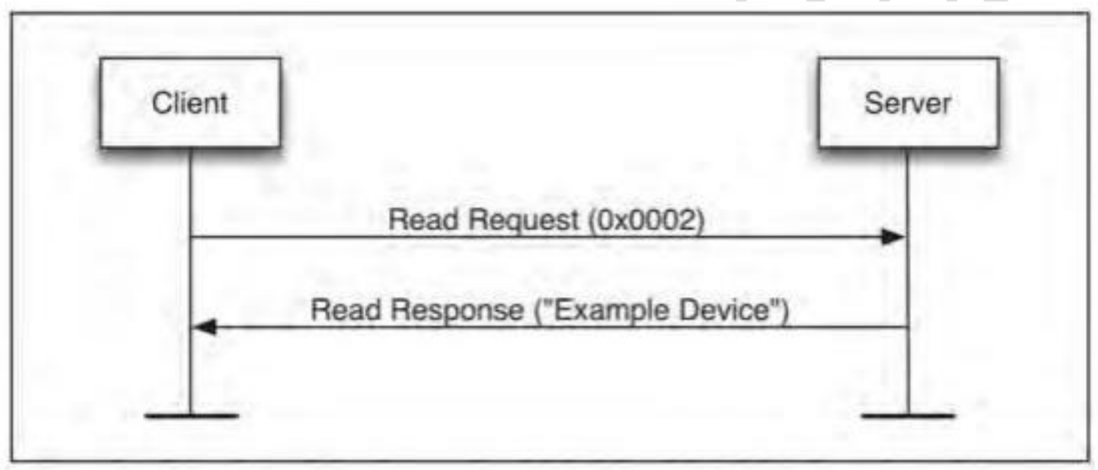
Parameter	Size (octets)	Description
Attribute Opcode	1	0x0A = Read Request
Attribute Handle	2	The handle of the attribute to be read

Format of Read Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0B = Read Response
Attribute Value	0 to (ATT_MTU−1)	The value of the attribute with the handle given

Format of Read Response

图 3-39 读请求\应答格式



Read Characteristic Value example

图 3-40 读请求\应答示意

```
Bluetooth Attribute Protocol
  Opcode: Read Request (0x0a)
  Handle: 0x0017
Bluetooth Attribute Protocol
  Opcode: Read Response (0x0b)
  Value: 64
```

图 3-41 Sniffer 采集空中读请求\应答数据

3.4.3.6、大对象读请求\应答(Read Blob Request\Response)

图 3-42 为大对象读请求\应答格式。图 3-43 为大对象读请求\应答示意。

有时属性值太长，无法装入一个读取响应，须使用大对象读取请求来获取剩余字节。大对象读取请求不光包含属性句柄，还包含属性值的这个数据中的偏移量。其响应将从属性偏移量开始，包含尽可能多的属性值。

在获得了属性值的前 22 个字节后，假如客户端还想获取后续的属性值，则使用大对象读取请求。下一条响应将返回第 23 个字节到 44 个字节；如此继续，直到客户端读取到完整的属性值。

该请求用于读取长特征值与长特征描述符。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0C = Read Blob Request
Attribute Handle	2	The handle of the attribute to be read
Value Offset	2	The offset of the first octet to be read

Format of Read Blob Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0D = Read Blob Response
Part Attribute Value	0 to (ATT_MTU – 1)	Part of the value of the attribute with the handle given

Format of Read Blob Response

图 3-42 大对象读请求\应答格式

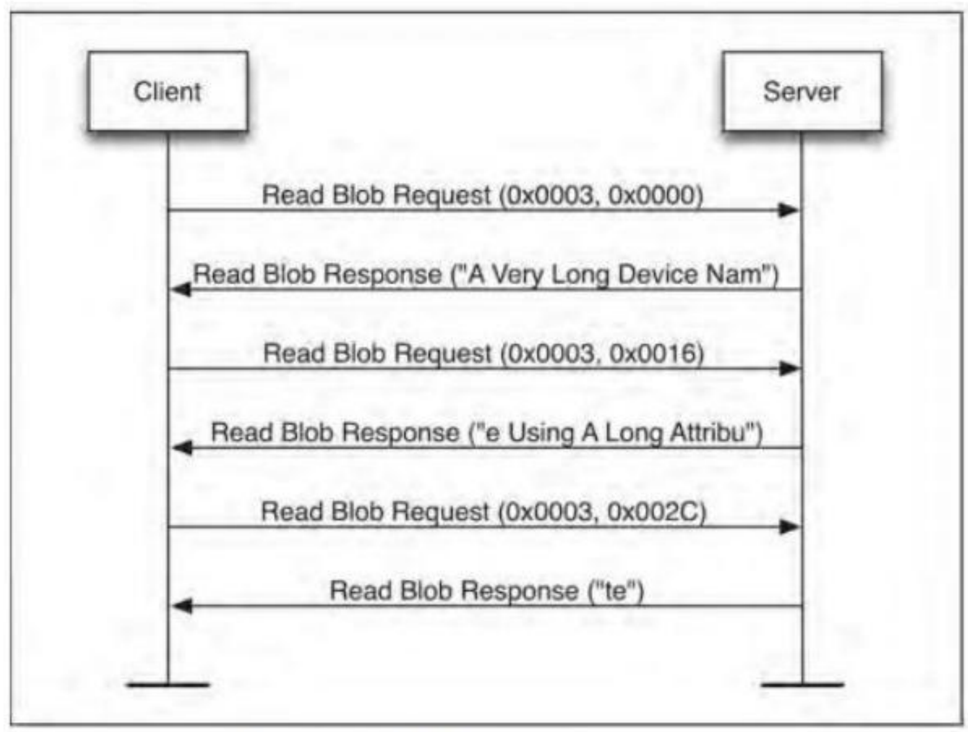


图 3-43 大对象读请求\应答示意

3.4.3.7、多重读取请求\应答(Read Multiple Request\Response)

图 3-44 为多重读取请求\应答包格式。图 3-45 为多重读取请求\应答示意。

用来在一个操作中读取多个属性值。该请求包含一个或多个属性句柄，响应则安之请求的顺序返回相应的属性值。然而因为响应中的数值之间没有界限，因此，在请求中除了最后一个属性允许可变的长度，其他属性必须为定长的属性值。也就是说，如果客户端用一个多重读取请求来读取三个属性，前两个属性的长度必须固定，最后一个属性的长度则可以变化。

如果客户端请求读取的属性值的长度超过了响应数据包所能承载的最大长度，那么无法放入响应数据包的数值将被丢弃。

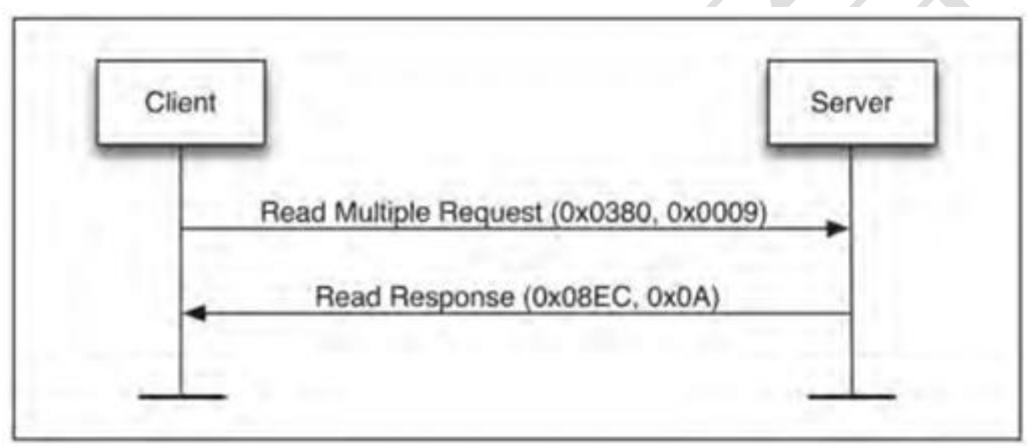
Parameter	Size (octets)	Description
Attribute Opcode	1	0x0E = Read Multiple Request
Set Of Handles	4 to (ATT_MTU–1)	A set of two or more attribute handles.

Format of Read Multiple Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x0F = Read Multiple Response
Set Of Values	0 to (ATT_MTU–1)	A set of two or more values.

Format of Read Multiple Response

图 3-44 多重读取请求\应答包格式



Read Multiple Characteristic Values example

图 3-45 多重读取请求\应答示意

3.4.3.8、按组类型读取请求\应答(Read By Group Type Request\Response)

图 3-46 为按组类型读取请求\应答包格式。图 3-47 为按组类型读取请求\应答示意。图 3-48 为 Sniffer 采集空中按组类型读取请求\应答数据。

它和按类型读取请求类型，也包含有一个句柄范围，读取时将其视为一个属性的类型来处理，只不过属性的类型必须为分组属性。其

响应包含所读取的属性句柄、属性分组中最后一个属性以及属性的数值。

这意味着，如果分组类型是首要服务，它将返回所有首要服务声明的属性句柄、该首要服务中最后一个属性以及首要服务声明的数值。因此，可以仅凭单个请求来发现设备上的所有首要服务、与之关联的属性句柄的范围以及这些服务的类型。

如同其他返回对个句柄-数值对的响应，如果返回的值长度可变，那么只有长度相同的属性值将在第一个响应中被返回。因此，客户端必须再次发起请求，更新起始句柄来发现想要获取的下一个属性。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x10 = Read By Group Type Request
Starting Handle	2	First requested handle number
Ending Handle	2	Last requested handle number
Attribute Group Type	2 or 16	2 or 16 octet UUID

Format of Read By Group Type Request

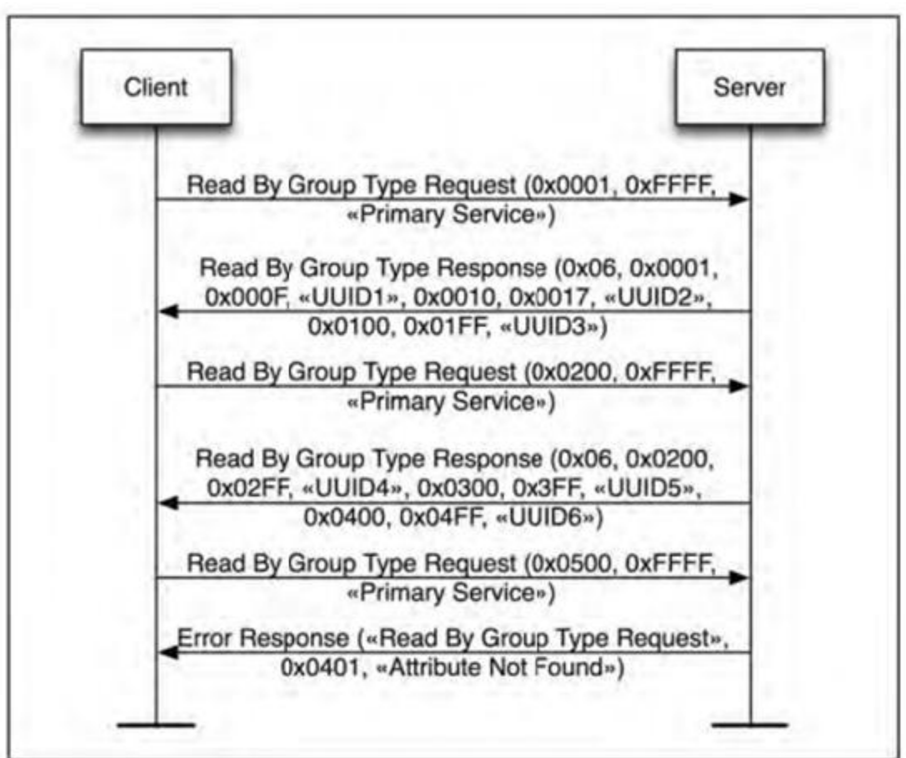
Parameter	Size (octets)	Description
Attribute Opcode	1	0x11 = Read By Group Type Response
Length	1	The size of each Attribute Data
Attribute Data List	2 to (ATT_MTU– 2)	A list of Attribute Data.

Format of Read By Group Type Response

Attribute Handle	End Group Handle	Attribute Value
2 octets	2 octets	(Length - 4) octets

Format of the Attribute Data

图 3-46 按组类型读取请求\应答包格式



Discover All Primary Services example

图 3-47 按组类型读取请求\应答示意

```
Bluetooth Attribute Protocol
  Opcode: Read By Group Type Request (0x10)
  Starting Handle: 0x0001
  Ending Handle: 0xffff
  UUID: GATT Primary Service Declaration (0x2800)

Bluetooth Attribute Protocol
  Opcode: Read By Group Type Response (0x11)
  Length: 6
⊟ Attribute Data, Handle: 0x0001, Group End Handle: 0x0007
    Handle: 0x0001
    Group End Handle: 0x0007
    Value: 0018
⊟ Attribute Data, Handle: 0x0008, Group End Handle: 0x000b
    Handle: 0x0008
    Group End Handle: 0x000b
    Value: 0118
⊟ Attribute Data, Handle: 0x000c, Group End Handle: 0x000e
    Handle: 0x000c
    Group End Handle: 0x000e
    Value: 0418
```

图 3-48 Sniffer 采集空中按组类型读取请求\应答数据

3.4.3.9、写请求\应答(Write Request\Response)

图 3-49 为写请求\应答包格式。图 3-50 为写请求\应答示意。图 3-51 为 Sniffer 采集空中写请求\应答数据。

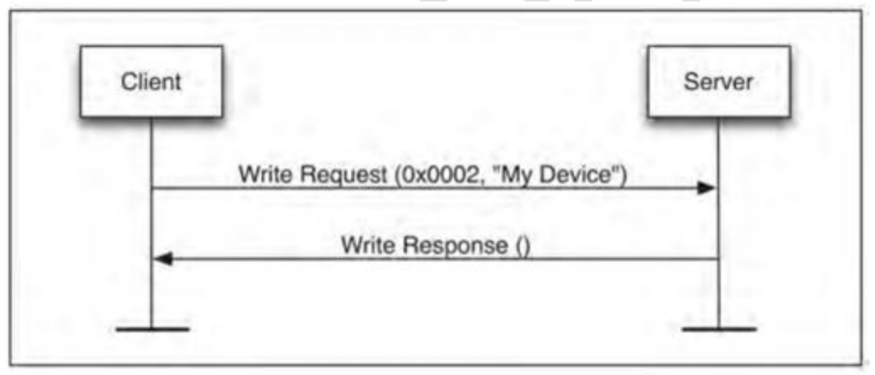
Parameter	Size (octets)	Description
Attribute Opcode	1	0x12 = Write Request
Attribute Handle	2	The handle of the attribute to be written
Attribute Value	0 to (ATT_MTU–3)	The value to be written to the attribute

Format of Write Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x13 = Write Response

Format of Write Response

图 3-49 写请求\应答包格式



Write Characteristic Value example

图 3-50 写请求\应答示意

```
Bluetooth Attribute Protocol
    Opcode: Write Request (0x12)
    Handle: 0x0014
    Value: 01

Bluetooth Attribute Protocol
    Opcode: Error Response (0x01)
    Request Opcode in Error: Write Request (0x12)
    Handle in Error: 0x0014
    Error Code: Insufficient Authentication (0x05)
```

图 3-51 为 Sniffer 采集空中写请求\应答数据

3.4.3.10、写命令(*Write Command*)

图 3-52 为写命令包格式。图 3-53 为写命令示意。图 3-54 为 Sniffer 采集空中写命令数据。

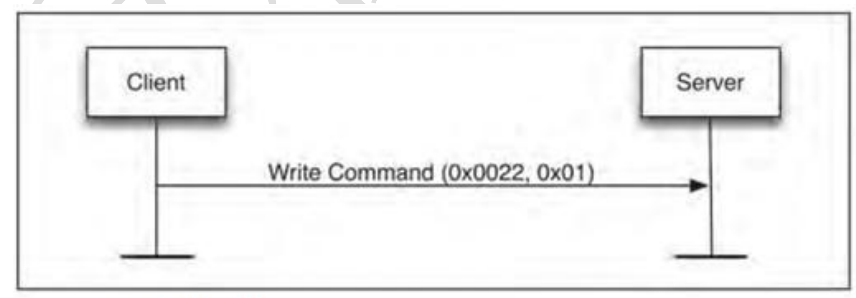
写入命令类似于读取请求，区别是写入命令没有响应。写入命令包含要写入的属性的句柄和要写入的数值。

当无需响应时，可以使用写入命令。此外，因为该命令可以在任何时刻发送，即使刚发送了一条请求还未收到相应的响应，对命令的发送时延有较高的要求时，该请求也适合。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x52 = Write Command
Attribute Handle	2	The handle of the attribute to be set
Attribute Value	0 to (ATT_MTU–3)	The value of be written to the attri-bute

Format of Write Command

图 3-52 写命令包格式



Write Without Response example

图 3-53 写命令示意

```
Bluetooth Attribute Protocol
   Opcode: Write Command (0x52)
   Handle: 0x0035
   Value: 02
```

图 3-54 Sniffer 采集空中写命令数据

3.4.3.11、签名写命令(Signed Write Command)

图 3-55 为签名写命令包格式。图 3-56 为签名写命令示意。

Parameter	Size (Octets)	Description
Attribute Opcode	1	0xD2 = Signed Write Command
Attribute Handle	2	The handle of the attribute to be set
Attribute Value	0 to (ATT_MTU - 13)	The value to be written to the attribute
Authentication Signa-ture	12	Authentication signature for the Attribute Upload, Attribute Handle and Attribute Value Parameters

Format of Signed Write Command

图 3-55 签名写命令包格式

Signed Write Without Response example

图 3-56 签名写命令示意

签名写命令和写命令类似，只是前者包含认证签名。通过这样的机制，发送端可以向服务器发送写入命令时认证自己，而无需加密通信连接。签名写命令适用于以下两种场合：

- 发起加密将显著增加数据连接的延迟
- 发起加密将显著增加简短且无需加密的数据的送达成本

认证签名由签名计数器和消息认证码构成。签名计数器对设备间发送的每一条消息赋予不同的值，不论消息发送的间隙连接中断与否。使用签名计数器可以抵御消息重放攻击，因此，假如签名写入命令的

签名计数器值和先前的值相同，该命令会被忽略。消息认证码长度为64 位，该码位于句柄、数值和签名计数器之后。注意，服务器需要为每一个客户端保存其最后使用的签名计数器。

3.4.3.12、准备写请求\应答(Prepare Write Request\Response) 和执行写请求\应答(ExecuteWrite Request\Response)

准备写请求和执行写请求实现两种功能：

- 它们提供了长属性值的写入功能
- 它们允许在一个单独执行的原子操作中写入多个值

属性服务器包含一个准备写入队列，其中保存有准备写入请求。队列的大小可独立配置，但通常它足够储存所需要准备写入的服务。只有在收到执行写入请求时，准备写入的值才会写入属性，也就是说执行写入请求给出了执行这些准备写入操作的开始信号。

准备写请求包含句柄、偏移量和部分属性值，这和大对象读取类似。这说明客户端即可以在队列中准备多个属性值，有可以准备一个长属性值的各个部分。这样，在真正执行准备队列之前，客户端可以确定某属性的所有部分都能写入服务器。

准备写入响应也包含请求中的句柄、偏移量和部分属性值。之所以这样做是为了数据传递的可靠性。客户端可以对比响应和请求的字段值，保证准备的数据被正确接收。

图 3-57 为准备写请求\应答包格式。图 3-58 为准备写请求\应答示意。

版权所有

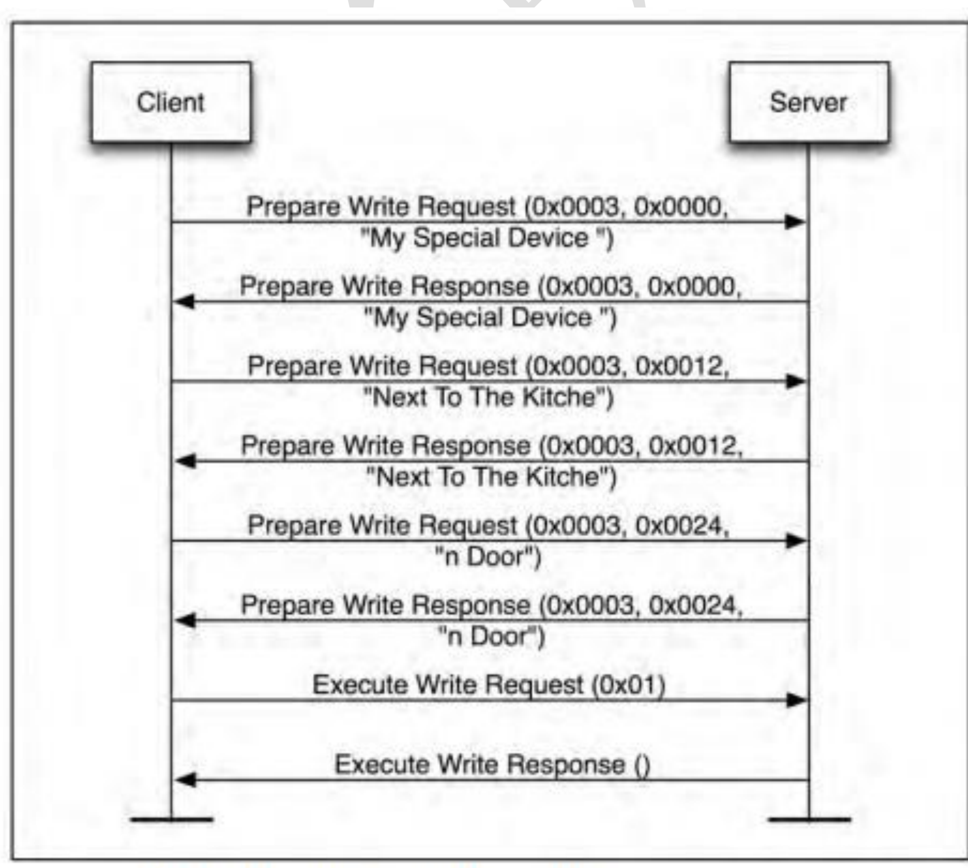
Parameter	Size (octets)	Description
Attribute Opcode	1	0x16 = Prepare Write Request
Attribute Handle	2	The handle of the attribute to be written
Value Offset	2	The offset of the first octet to be written
Part Attribute Value	0 to (ATT_MTU–5	The value of the attribute to be writ-ten

Format of Prepare Write Request

Parameter	Size (octets)	Description
Attribute Opcode	1	0x17 = Prepare Write Response
Attribute Handle	2	The handle of the attribute to be written
Value Offset	2	The offset of the first octet to be written
+Part Attribute Value	0 to (ATT_MTU–5))	The value of the attribute to be writ-ten

Format of Prepare Write Response

图 3-57 写请求\应答包格式



Write Long Characteristic Values example

图 3-58 写请求\应答示意

可以这么理解，准备写入请求，之所以是“准备写”，也就是没有真正的写到属性中去，上文提到准备写是写在服务器中的单独的一个缓冲器中，当所有需要写入的数据都发送给服务器后，那么就要下一个命令，即为执行写，“执行写”也就是真正的执行写入属性的操作。

完整过程是，一旦接收完所有的准备写入请求，服务器将拥有一个随时可以执行的准备写入队列。客户端发送表示位为“立刻写入”的执行写入请求，随后服务器将在一次原子操作中写入所有值。属性将按照其准备的顺序写入。如果客户端多次准备了同一个属性值，那么服务器将按照顺序向该属性写入这些值。这意味着如果使用准备队列配置硬件状态，例如硬件需要先后执行禁用、配置、重新启用操作，那么可以用一个准备队列来实现：在队列总先将对应的属性写入“禁用”，接着写入“配置”，之后在准备队列的末尾写入“启用”，随后执行该队列。这样便能在一个原子操作中实现该硬件的重新配置。

图 3-59 为执行写请求\应答包格式。图 3-60 为可靠写请求\应答示意。

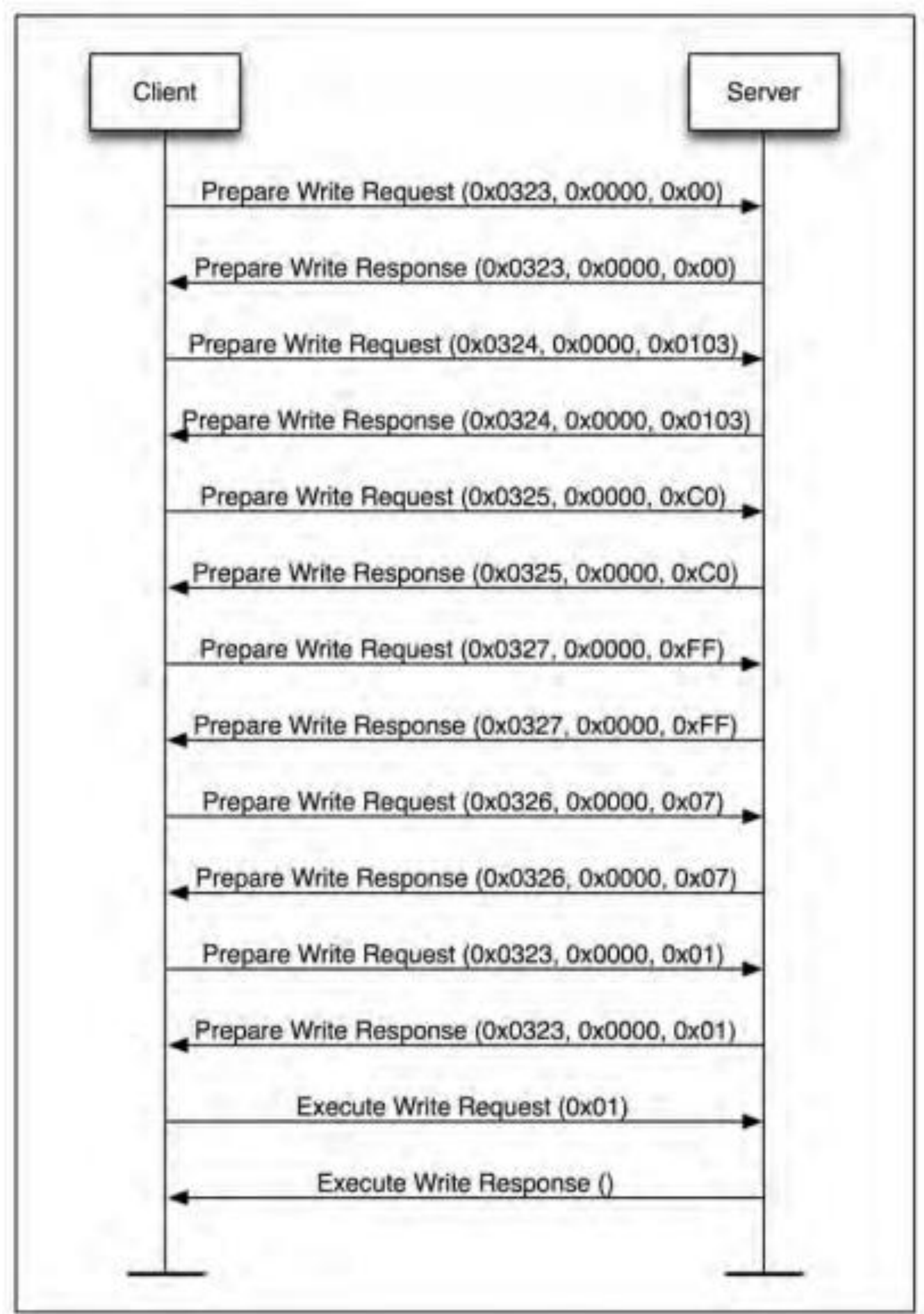
Parameter	Size (octets)	Description
Attribute Opcode	1	0x18 = Execute Write Request
Flags	1	0x00 – Cancel all prepared writes 0x01 – Immediately write all pending prepared values

Format of Execute Write Request

Parameter	Size	Description
Attribute Upload	1	0x19 - Execute Write Response

Format of Execute Write Response

图 3-59 执行写请求\应答包格式



Reliable Writes example

图 3-60 可靠写请求\应答示意

3.4.3.13、句柄通知(Handle Value Notification)

图 3-61 为句柄通知包格式。图 3-62 为句柄通知示意。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1B = Handle Value Notification
Attribute Handle	2	The handle of the attribute
Attribute Value	0 to (ATT_MTU–3)	The current value of the attribute

Format of Handle Value Notification

图 3-61 句柄通知包格式

Notifications example

图 3-62 句柄通知示意

当服务器想要向客户端发送快速得属性状态更新时，可以发送一条句柄值通知。这个是服务器能够发给客户端的两种消息中的一种，并且是不要求响应的那种。服务器可以在任何时刻发送该通知，同时该通知也是不可靠的。

句柄值通知包含属性句柄和数值。因此该通知是服务器发往客户端的一条消息，用来告知某属性的当前值。它是属性协议中最重要的消息之一。它不仅让客户端能够有效地从服务器获取当前属性数据库的更新，而且也被用来通知客户端有限状态机的变化。

一般来说，服务器会为所有的绑定设备配置通知。这样做的好处在于当某客户端重新与设备建立连接时，该设备上的服务能够立刻将自己当前的状态通知该客户端。例如，将某设备的电池电量配置为通知。

因为通知不具备任何确认机制，它们可以在任何时刻发送，而不管当时正在进行什么用的操作，一次，通知适用于必须立刻向客户端发送信息的情况。

3.4.3.14、句柄指示\确认(Handle Value Indication\Confirmation)

句柄值指示类似于句柄值通知。它有着相同的属性句柄字段和数值，不同的客户端收到指示以后应回复。服务器一次只能发送一条指示，并且只有收到确认响应后才能发起下一条指示。

句柄值确认不含任何数据，主要用于流控。因为具备了确认机制，指示被视为可靠传输。一旦服务器收到确认信息，它便能确定客户端收到了该信息。

图 3-63 为句柄指示\确认包格式。图 3-64 为句柄指示\确认示意。

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1D = Handle Value Indication
Attribute Handle	2	The handle of the attribute
Attribute Value	0 to (ATT_MTU–3)	The current value of the attribute

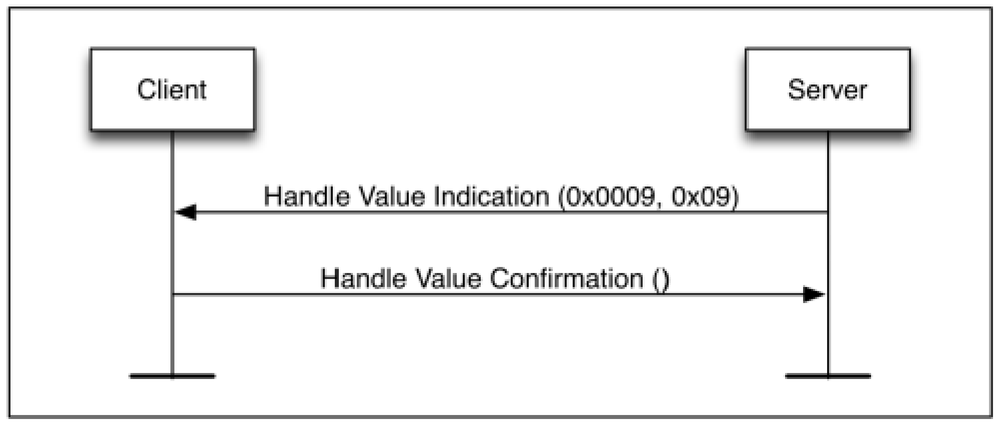
Format of Handle Value Indication

Parameter	Size (octets)	Description
Attribute Opcode	1	0x1E = Handle Value Confirmation

Format of Handle Value Confirmation

图 3-63 句柄指示\确认包格式

版权所有



Indications example

图 3-64 为句柄指示\确认示意

3.4.3.15、错误应答

当某设备无法完成请求所要求的操作时，它可以发送错误响应。例如某设备通过发送写入请求试图写入某个属性，但该属性是只读的，那么服务器应发送错误响应来给出请求失败的原因，而不是发送写入响应正常。

错误响应包含导致错误的与请求相关的所有信息，例如请求失败的属性、导致错误的首要原因等。一旦客户端收到错误响应，它会认为该响应与其发送的最后一条请求相对应。因此，错误响应是另一种利用响应消息来终止请求操作的方法。这也说明对每一条可以发送的请求都会有两种可能的响应：失败错误响应和成功响应。

图 3-65 为句柄指示\确认包格式。表 3-13 为错误代码表。

版权所有

Parameter	Size (octets)	Description
Attribute Opcode	1	0x01 = Error Response
Request Opcode In Error	1	The request that generated this error response
Attribute Handle In Error	2	The attribute handle that generated this error response
Error Code	1	The reason why the request has generated an error response

Format of Error Response

图3-65 句柄指示\确认包格式

表3-13 错误代码表

错误代码	书名号	描述
0x01	«Invalid Handle»	无效句柄
0x02	«Read Not Permitted»	不允许读取
0x03	«Write Not Permitted»	不允许写入
0x04	«Invalid PDU»	PDU 无效
0x05	«Insufficient Authentication»	认证不足
0x06	«Request Not Supported»	请求不支持
0x07	«Invalid Offset»	偏移量无效
0x08	«Insufficient Authorization»	授权不足
0x09	«Prepare Queue Full»	准备队列已满
0x0A	«Attribute Not Found»	属性不存在
0x0B	«Attribute Not Long»	属性非大对象
0x0C	«Insufficient Encryption Key Size»	秘钥长度不足
0x0D	«Invalid Attribute Value Length»	属性值长度无效

0x0 E	«Unlikely Error»	未知错误
0x0F	«Insufficient Encryption»	加密不足
0x10	«Unsupported Group Type»	组类型不支持
0x11	«Insufficient Resources»	资源不足
0x12:0x7F	Reserved	保留
0x80:0xFF	Application Error	应用自定义错误

下面对表中的代码进行一一说明：

- **无效句柄** 请求中的属性句柄是无线的，它在服务器中不存在。该错误是由于服务器不使用或者不具备请求读取或者写入的属性句柄。当属性句柄被设置为 0x0000 时也可以发送该错误。
- **不允许读取** 该属性不允许读取其属性值。例如当客户端试图读取某控制点的只写是属性时，可以发送该错误。
- **不允许昔写入** 该属性不允许写入其属性值。例如试图写入只读属性值将导致该错误。
- **PDU 无效** 服务器不理解收到的请求。通常当客户端发送了错误格式的请求时，将导致该错误。例如，读取请求必须有两字节的句柄作为其唯一参数。当读取请求不具备两字节的参数时将导致该错误。
- **认证不足** 当两设备缺少相互认证时，便无法执行某属性值的读取或写入操作。为了进行认证，可以将连接加密，或是在设备还未配对时使用安全管理配对规程对设备进行配对与连接。
- **请求不支持** 服务器理解发送的请求但当前无法执行。当服务器

版权所有

未实行某个已知的请求，或是不理解某个未来的请求时，可以使用该错误。该错误可以作为设备无法执行请求时的默认错误。

- **偏移量无效**　请求包含偏移量，但偏移量是无效的。偏移量和属性值的长度相同不会导致该错误，但会得到一个长度为 0 的值。
- **准备队列已满**　由于用于存储等待写入队列的空间已满，准备写入请求无法被接收。
- **属性不存在**　属性经遍历不存在。该错误仅用于在一个属性范围内查找一种或几种特定的属性类型时。只有查找信息请求、按类型读取请求与按组类型读取请求会导致该错误。对于查找信息请求而言，该错误意味着没有找到给定的句柄范围内的属性。对按类型值查找请求、按类型读取请求和按组类型读取请求而言，该错误说明在某句柄范围内没有找到给定类型的属性。
- **属性非大对象**　当大对象读取请求所要读取的属性不是大对象类型时，该请求被拒绝，客户端必须改用读取请求。该错误仅和定长的属性值相关，并且该长度小于大对象读取请求当前用到的 MTU 长度。由此简单的做法是先使用读取请求读取属性值的前 22 字节，随后使用大对象读取请求读取后续的字节。
- **秘钥长度不足**　该错误是由于在连接已经加密、认证与授权充足的情况下，配对期间协商的秘钥长度不满足服务的要求。有些属性值需要使用高强度的秘钥来充分保证数据的机密性。
- **未知错误**　这或许是个最棒的错误码了。如果导致该错误的情形事先没有考虑到过，便是“未知错误”。

版权所有

- **加密不足** 可以读取或写入属性值，但连接未加密。一些属性值需要加密连接来保证数据的机密性。
- **组类型不支持** 服务器不认为请求中包含的属性类型是组类型。按组类型读取请求只能使用服务器已知的组类型。
- **资源不足** 服务器的资源不足以接收和执行某特定的请求。例如，某服务配置某设备广播数据时，若当前广播的数据加上需要增加的广播的数据后，数据总长度过长，则会导致该错误。
- **应用错误** 某请求对服务中的属性执行了不允许的操作，服务分配了错误码来报告错误的所在。这是一个 128-255 范围内的错误，实际的含义有包含该属性的服务规格书来定义。

版权所有

3.5、GATT 规程和 ATT 协议映射

有了协议和数据库，相当于有了铁路和火车站，但是列车进出火车站还的有个规定是那个站台。

3.5.1、GATT 规程

对于 GATT 规程定义了如何用 ATT 协议发现和使用服务、特性与描述符的标准方法。GATT 规程有 3 中基本规程：发现规程、客户端发起规程和服务器发起规程。

3.5.1.1、发现服务和特性

有 3 种方法发现服务：发现所有首要服务、按服务 UUID 发现首要服务、查找包含服务。

在服务发现以后，便可以发现每个服务的特性。要取得完整的特性，需要发现特性和特性描述符

版权所有

3.5.2、ATT 协议与 GATT 映射表

表 3-14 为 GATT 规程映射到 ATT PDU。表 3-15 为 ATT PDU 映射到 GATT 规程。表 3-16 为 Sniffer 采集“心率计”服务的属性数据库。

表 3-14 GATT 映射到 ATT

作用	GATT 规程	ATT 协议	
Server Configuration	Exchange MTU	Exchange MTU Request	交换 MTU 请求
		Exchange MTU Response	
		Error Response	
Primary Service Discovery	Discover All Primary Services	Read By Group Type Request	按组类型读取
		Read By Group Type Response	
		Error Response	
	Discover Primary Services By Service UUID	Find By Type Value Request	按类型值查找
		Find By Type Value Response	
		Error Response	
Relationship Discovery	Find Included Services	Read By Type Request	按类型读取
		Read By Type Response	
		Error Response	
Characteristic Discovery	Discover All Characteristic of a Service	Read By Type Request	按类型读取
		Read By Type Response	
		Error Response	
	Discover Characteristic by UUID	Read By Type Request	按类型读取
		Read By Type Response	
		Error Response	
Characteristic Descriptor Discovery	Discover All Characteristic Descriptors	Find Information Request	查找信息请求
		Find Information Response	
		Error Response	
Characteristic Value Read	Read Characteristic Value	Read Request	读取
		Read Response	
		Error Response	
	Read Using Characteristic UUID	Read By Type Request	按类型读取
		Read By Type Response	
		Error Response	
	Read Long Characteristic Values	Read Blob Request	大对象读取
		Read Blob Response	
		Error Response	
	Read Multiple Characteristic Values	Read Multiple Request	多重读取
		Read Multiple Response	
		Error Response	

表 3-15 ATTPDU 映射到 GATT 规程

属性协议数据单元	GATT 规程
交换 MTU 请求	交换 MTU 规程
查找信息请求	发现所有特性描述符规程
按类型值查找请求	按服务 UUID 发现首要服务规程
按类型读取请求	查找包含服务 发现所有服务特性 按 UUID 发现特性 按特性 UUID 读取规程
读取请求	读取特性值 读取特性描述符规程
大对象读取请求	读取长特性值 读取长特性描述规程
多重读取请求	读取多重特性值规程
按组类型读取请求	发现所有首要服务规程
写入请求	写入特性值 写入特性描述符规程
写入命令	无需响应写规程
签名写入命令	无需响应的签名写入规程
准备写请求 执行写请求	写入长特性值 特性值可靠写入 写入长特性描述符规程
句柄值通知	通知规程
句柄值指示	指示规程

版权

表 3-16　心率计属性数据库

Handle	UUID		Value		Att 命令
0x0001	0x2800	**«Primary Service»**	0x1800	**«Generic Access Profile »**	Bluetooth Attribute Protocol Opcode: Read By Group Type Request (0x10) Starting Handle: 0x0001 Ending Handle: 0xffff UUID: GATT Primary Service Declaration (0x2800) Bluetooth Attribute Protocol Opcode: Read By Group Type Response (0x11) Length: 6 Attribute Data, Handle: 0x0001, Group End Handle: 0x0007 Handle: 0x0001 Group End Handle: 0x0007 Value: 0018 Attribute Data, Handle: 0x0008, Group End Handle: 0x000b Handle: 0x0008 Group End Handle: 0x000b Value: 0118 Attribute Data, Handle: 0x000c, Group End Handle: 0x0011 Handle: 0x000c Group End Handle: 0x0011 Value: 0d18

					Bluetooth Attribute Protocol Opcode: Read By Group Type Response (0x11) Length: 6 Attribute Data, Handle: 0x0012, Group End Handle: 0x0015 Handle: 0x0012 Group End Handle: 0x0015 Value: 0f18 Attribute Data, Handle: 0x0016, Group End Handle: 0xffff Handle: 0x0016 Group End Handle: 0xffff Value: 0a18
0x0003	0x2A00	**« Device Name Characteristic »**	"Nordic_HRM"		Bluetooth Attribute Protocol Opcode: Read By Type Request (0x08) Starting Handle: 0x0001 Ending Handle: 0x0007 UUID: Device Name (0x2a00) Bluetooth Attribute Protocol Opcode: Read By Type Response (0x09) Length: 12 Attribute Data, Handle: 0x0003 Handle: 0x0003 Value: 4e6f726469635f48524d
0x0008	0x2800	**«Primary Service»**	0x1801	**«Generic Attribute Profile »**	
0x0009	0x2803	**«Characteristic»**	0x20	性质	Bluetooth Attribute Protocol Opcode: Read By Type Request (0x08) Starting Handle: 0x0008 Ending Handle: 0x000b UUID: GATT Characteristic Declaration (0x2803)
			0x000a	句柄	
			0x2A05	UUID	

					Bluetooth Attribute Protocol Opcode: Read By Type Response (0x09) Length: 7 Attribute Data, Handle: 0x0009 Handle: 0x0009 Value: 200a00052a
0x000a	0x2A05	**«Service Changed Characteristic»**			
0x000b	0x2902	**«Client Characteristic Configuration»**	0x0002		Bluetooth Attribute Protocol Opcode: Find Information Request (0x04) Starting Handle: 0x000b Ending Handle: 0x000b Bluetooth Attribute Protocol Opcode: Find Information Response (0x05) UUID Format: 16-bit UUIDs (0x01) Handle: 0x000b UUID: Client Characteristic Configuration (0x2902) Bluetooth Attribute Protocol Opcode: Write Request (0x12) Handle: 0x000b Value: 0200

					Bluetooth Attribute Protocol Opcode: Write Response (0x13)
0x000c	0x2800	**«Primary Service»**	0x180D	**«Heart Rate service »**	
0x000d	0x2803	**«Characteristic»**	0x12	性质	Bluetooth Attribute Protocol Opcode: Read By Type Request (0x08) Starting Handle: 0x000c Ending Handle: 0x0011 UUID: GATT Characteristic Declaration (0x2803) Bluetooth Attribute Protocol Opcode: Read By Type Response (0x09) Length: 7 Attribute Data, Handle: 0x000d Handle: 0x000d Value: 100e00372a Attribute Data, Handle: 0x0010 Handle: 0x0010 Value: 021100382a
			0x000E	句柄	
			0x2A37	UUID	
0x000e	0x2A37	**«Heart Rate Measurement characteristic»**	0x00B4		Bluetooth Attribute Protocol Opcode: Handle Value Notification (0x1b) Handle: 0x000e Value: 00b4

<table>
<tr>
<td>0x000f</td>
<td>0x2902</td>
<td>«Client Characteristic Configuration»</td>
<td colspan="2">0x0001</td>
<td>Bluetooth Attribute Protocol
Opcode: Find Information Request (0x04)
Starting Handle: 0x000f
Ending Handle: 0x0010

Bluetooth Attribute Protocol
Opcode: Find Information Response (0x05)
UUID Format: 16-bit UUIDs (0x01)
Handle: 0x000f
UUID: Client Characteristic Configuration (0x2902)
Handle: 0x0010
UUID: GATT Characteristic Declaration (0x2803)

Bluetooth Attribute Protocol
Opcode: Write Request (0x12)
Handle: 0x000f
Value: 0100

Bluetooth Attribute Protocol
Opcode: Write Response (0x13)</td>
</tr>
<tr>
<td rowspan="3">0x0010</td>
<td rowspan="3">0x2803</td>
<td rowspan="3">«Characteristic»</td>
<td>0x02</td>
<td>性质</td>
<td rowspan="3"></td>
</tr>
<tr>
<td>0x0011</td>
<td>句柄</td>
</tr>
<tr>
<td>0x2A38</td>
<td>UUID</td>
</tr>
<tr>
<td>0x0011</td>
<td>0x2A38</td>
<td>«Body Sensor Location characteristic»</td>
<td colspan="2">0x03</td>
<td>Bluetooth Attribute Protocol
Opcode: Read Request (0x0a)
Handle: 0x0011</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td></td><td colspan="2"></td><td>Bluetooth Attribute Protocol
Opcode: Read Response (0x0b)
Value: 03</td></tr>
<tr><td>0x0012</td><td>0x 2800</td><td>«Primary Service»</td><td>0x180F</td><td>«Battery service »</td><td></td></tr>
<tr><td rowspan="3">0x0013</td><td rowspan="3">0x2803</td><td rowspan="3">«Characteristic»</td><td>0x12</td><td>性质</td><td rowspan="3">Bluetooth Attribute Protocol
Opcode: Read By Type Request (0x08)
Starting Handle: 0x0012
Ending Handle: 0x0015
UUID: GATT Characteristic Declaration (0x2803)

Bluetooth Attribute Protocol
Opcode: Read By Type Response (0x09)
Length: 7
Attribute Data, Handle: 0x0013
Handle: 0x0013
Value: 121400192a</td></tr>
<tr><td>0x0014</td><td>句柄</td></tr>
<tr><td>0x2A19</td><td>UUID</td></tr>
<tr><td>0x0014</td><td>0x2A19</td><td>«Battery Level characteristic»</td><td colspan="2">0x64</td><td>Bluetooth Attribute Protocol
Opcode: Read Request (0x0a)
Handle: 0x0014

Bluetooth Attribute Protocol
Opcode: Read Response (0x0b)
Value: 64</td></tr>
</table>

					Bluetooth Attribute Protocol Opcode: Handle Value Notification (0x1b) Handle: 0x0014 Value: 64
0x0015	0x2902	**«Client Characteristic Configuration»**	0x0001		Bluetooth Attribute Protocol Opcode: Find Information Request (0x04) Starting Handle: 0x0015 Ending Handle: 0x0015 Bluetooth Attribute Protocol Opcode: Find Information Response (0x05) UUID Format: 16-bit UUIDs (0x01) Handle: 0x0015 UUID: Client Characteristic Configuration (0x2902) Bluetooth Attribute Protocol Opcode: Write Request (0x12) Handle: 0x0015 Value: 0100 Bluetooth Attribute Protocol Opcode: Write Response (0x13)
0x0016	0x2800	**«Primary Service»**	0x180A	**«Device Information service»**	

3.6、安全管理(Security Manager (SM))

我的体会是安全管理是整个协议中最难处理理解和处理的事情，它的复杂程度有“这么”深，知道了吧！为了讲明白我也不知道该如何是好啊！分这么几点讲吧！

- 为什么加密：为什么呢？安全啊！谁都不想光屁股在外面跑吧！
- 加密干了什么，需求什么？
- 加密相关计算公式
- 加密配对绑定过程(密钥计算)
- 安全管理传输协议
- NRF51822 加密硬件模块介绍
- NRF51822 和 IPhone4s 的空中数据分析

3.6.1、加密做了什么和加密需求

在连接时，可通过加密来确保数据的机密性。机密性是指第三方“攻击者”由于没有加密链路的共享秘密，因此无法拦截、破译或读取消息原始内容。**注意：数据包的报头和长度字段不会被加密的，这句在 4.0 的规范中找了好久。原文如下：**

User information can be protected by encryption of the packet payload; **the access code and the packet header shall never be encrypted**.

这有个好处是，当接收到包时可以直接分析报头判断 SN 和 NESN 标志。

加密数据包含一个消息完整性校验值，表明该数据包已经过认证。

版权所有

为了验明发送方的有效身份，认证使用共享密钥为已加密的数据计算签名，可防止第三方篡改数据包中的任何内容。通过认证，消息的接收方能够确信收到的数据包来自一个可信设备。

加密数据包含一个数据包计数器，用来防止重放攻击。重放攻击是指攻击者截取一个消息，随后再次发送该消息以期望收到响应。

简单来说，1、加密了整个发送到空中的数据，防止窃取到原文；2、发送到空中的包中包含有 32 位的消息完整性校验(MIC)值，这个值能进行整个包的数据错位校验，CRC 只能检查有限的位，但是 MIC 可确保数据的正确性，在 3.6.3.3.4 节中可以知道，MIC 其实就是签名认证；3、加密数据包中的包计数器确保数据不是由第三方伪造。

那么要完成加密需要哪些东西呢？首先需要一个密码块，也就是密码本，接收到的密文，通过密码本一一翻译，变为明文，在 BLE 中的密码块，实际上是个单向函数，用于产生秘钥、加密和提供完整性检查。这种加密引擎被称为先进加密系统(AES)。低功耗蓝牙使用 AES 的 128 位版本，也就是明文、秘钥和密文都是 128 位的。它的表达式用式(3-1)表示：

$$密文=E_{key}(明文) \quad 或 \quad 密文=E(key,明文) \tag{3-1}$$

上式是加密的发动机，其中的密钥就是汽油了，在 BLE4.0 的协议中使用到大量的密钥，而**加密认证的整个过程几乎都是围绕怎么将两个设备使用到的秘钥能够安全的共享**，也就是当一方把密码告诉另一方时，始终要提防第三方也可能听得到这个密钥，一旦公共的密钥被第三方窃取了，那么又变成光屁股在路上跑了。所以对于加密来说，

版权所有

并不是加密数据有多难，而是把需要共享的密钥安全的送到正确的设备才是难点，这就引入了配对的复杂过程。

在 BLE 的 4.0 中主要有 5 个密钥需要共享，这些密钥传输时一次比一次安全，有些密钥共享时并不是赤裸裸的直接传输特定密钥，而是传输计算相应密钥时使用到的参数。这 5 个密钥如下：

- 临时密钥(Temporary Key:TK)
- 短期密钥(Short Term Key:STK)
- 长期秘钥(Long-Term Key:LTK)
- 身份解析秘钥(Identity Resolving Key:IRK)
- 连接签名解析秘钥(Connection Signature Resolving Key:CSRK)

版权所有

3.6.2、加密相关计算公式

在 4.0 的安全管理中设计到几个加密相关的计算，但是最终还是会归到式(3-1)中的安全函数 e 中。涉及的公式如下(应用在密钥计算章节)：

- ah 函数：这个函数是用来计算可解析地址中的哈希值(hash)的。

  表达式为式(3-2)

$$ah(k, r) = e(k, r') \bmod 2^{24} \tag{3-2}$$

  其中 k =128 bits 、r = 24 bits。而 r'是将 r 高位补 0 达到 128bits。表达式是 r' = padding || r，padding 就是 104bits 的 0，“||”为连接的意思。mod 2^{24} 目的是保留计算结果的低 24 位。

- c1 函数：这是计算确认值的函数，表达式为式(3-3)。

c1 (k, r, preq, pres, iat, rat, ia, ra) = e(k, e(k, r XOR p1) XOR p2)   (3-3)

其中

p1 = pres || preq || rat’ || iat’

p2 = padding || ia || ra

k = 128 bits： key 密钥

r =128 bits： rand 随机数

pres = 56 bits ： pairing response command 配对应答命令

preq= 56 bits ： pairing request command 配对请求命令

iat= 1 bit ： initiating device address type 发起者设备地址类型

ia = 48 bits ： initiating device address 发起者设备地址

rat =1 bit： responding device address type 响应者设备地址类型

ra = 48 bits ： responding device address 响应者设备地址

padding = 32 bits or 0： 补 0 用的掩码

- s1 函数：计算短期密钥 STK 用，表达式为式(3-4)。

s1(k, *r1*, *r2*) = e(k, r’)   (3-4)

其中

K= 128 bits

r1 =128 bits：这是蓝牙设备发送 128bits 的随机数

r2 = 128 bits：这是蓝牙设备发送 128bits 的随机数

r’ = *r1’* || *r2’*，而 *r1’* 和 *r2’*是截取 r1 和 r2 的低 64bits

- dm 函数：这个函数是计算长期秘钥 LTK 中的某个参数时用到，表达式为式(3-5)。

版权所有

$$dm(k, r) = e(k, r') \bmod 2^{16} \quad (3\text{-}5)$$

其中

k = 128 bits　　r= 64 bits

padding = 64 bits　　r' = padding || r

- d1 函数：这个函数用来计算 LTK、IRK、CSRK 和 DHK 等等密钥，表达式为式(3-6)。

$$d1(k, d, r) = e(k, d') \quad (3\text{-}6)$$

其中

k =128 bits　　d = 16 bits

r = 16 bits　　padding = 96 bits

d' = padding || r || d

3.6.3、加密配对绑定过程

这里先解释一下，什么是配对和绑定，配对是找到并确定需要和自己通信的设备，也就是身份确定，接着是将安全密钥共享，而这一过程仅仅是由启动加密到得到短期秘钥(STK)为止(图 3-66 中的 Phase1 和 Phase2)；而绑定就是将长期秘钥(LTK)、身份解析秘钥(IRK)和连接签名解析秘钥(CSRK)这 3 个密钥中的某个或者组合进行交换(图 3-66 中的 Phase3)后，将交换的这些密钥存储到数据库中的过程。

这里先讲明白一个事情：

配对绑定 (图 3-66 中的 Phase1、Phase2 和 Phase3)只有在两个设备之间第一次进行配对时才会发生，也就是说配对绑定过程只会发

版权所有

生在两个设备之间第一次连接，那么第二次和之后的连接呢？因为有在第一次配对绑定过程中有“绑定”过程，也就是说已经将第一次的密钥进行了存储，如果这个存储的数据库没有人为的清空，那么之后的连接就不会再出现配对过程。同时还有一个问题是：并不是所有的通信都需要加密进行数据保护，有时应用发送的数据不怕被别人听到，所以光着屁股跑也是没有关系。

配对绑定是一个三阶段的过程，如图 3-66。前两个阶段总是使用，第三阶段是一个可选的传输特定密钥分配的阶段，而这一个阶段的有无是由第一阶段决定，也就是没有第三阶段，相当于没有需要存储的密钥也就没有“绑定”过程。

阶段 1：配对特征交换得到临时密钥(TK)值

阶段 2：身份确认以及短期密钥(STK)生产

阶段 3：传输特定密钥

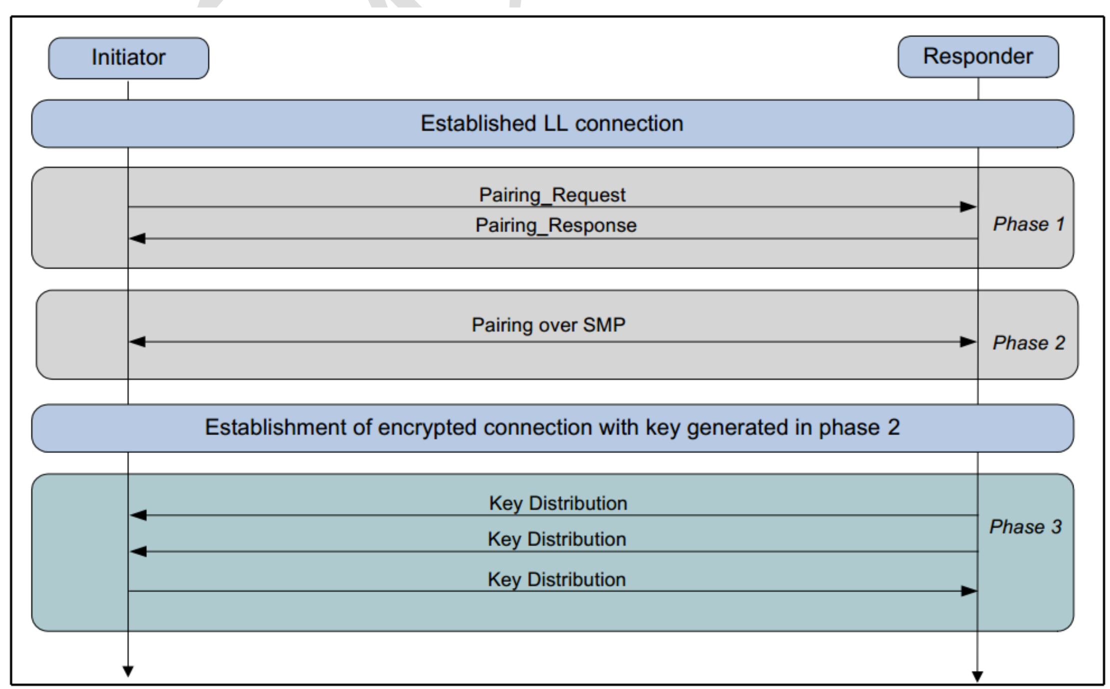


图 3-66 配对过程

这三个阶段都有关系，第 1 阶段决定了临时密钥(TK)的生产方式，同时也就是决定了第 2 阶段短期秘钥(STK)的生成，还决定着第 3 阶段是否存在；而第 2 阶段从英文“Pairing over SMP”理解就是通过安全管理协议进行配对，而实际上第二阶段过程的目的就是确定自己正在和一个真正想要通信的设备通信，而非第三方，简而言之就是确定对方身份。而阶段 3 实际上属于绑定的过程，因为要存储到安全数据库中的数据都是第 3 阶段发送的，这个阶段传输的数据通过第 2 阶段生成的密钥进行加密，确保共享的密钥受到保护。

3.6.3.1、配对特征交换得到临时密钥(TK)值

这个过程上面已经提到决定了临时密钥(TK)的生成。那么到底是怎么决定的呢？这里提到配对特征交换，为什么交换？交换了什么？交换的输入输出(IO)功能，这里拿 ATM 机取钱打个比方，插卡后你知道要输入取款密码，那么为什么可以输入密码，因为取款机有显示屏和输入密码的键盘。这里对于临时密钥(TK)也是一个密码，这个密码是不是也需要输入，但是如果有的蓝牙设备只有显示屏没有键盘或者只有键盘没有显示屏，甚至什么都没有，怎么办呢？所以需要两个设备进行基本输入输出(IO)功能的交换，以达到一个双方都能接收的方式实现 TK 共享，实际上反过来讲 TK 的共享又并不是完全由 IO 决定的，以信用卡刷卡为例，有的人不想对信用卡设置密码，这个也不犯法吧！POS 机上有输入键盘和显示屏，但是使用没有设置密码的信用卡，照样是可以的。也就是根据保护程度，协议中将安全分为 3 种特

版权所有

性：

- Authenticated MITM protection：可靠中间人保护
- Unauthenticated no MITM protection：不可靠无中间人保护
- No security requirements：无安全需求

MITM 为 man-in-the-middle 即中间人的意思，而这里的“人”应该是第三方蓝牙设备。可靠中间人保护就是在 TK 共享是不会有第三方设备知道共享的 TK 密钥；不可靠无中间人保护就是说 TK 共享时第三方蓝牙设备很容易知道共享的 TK 值，所以是不可靠的传输；而无保护就是光屁股在外面跑，也不怕被别人盗取数据。

然而事实上配对特征交换不只是简单的交换 IO 能力，同时 TK 也不单单由 IO 决定，在安全管理协议中的配对特征交换主要完成：交换 IO 能力、OOB(Out Of Band:带外)认证数据可用性、认证需求、秘钥大小以及图 3-66 中第 3 阶段将要分配的特定密钥。IO 能力、OOB 认证数据可用性以及认证需求用来决定图 3-66 中第 2 阶段计算 STK 生成的方法，其实就是决定 TK 的生成方法。

- 认证需求是指需不需绑定和需不需要防止 MITM。
- 密钥大小：在协议规范中有规定密钥的大小为[7Bytes,16Bytes]，也就是[56bits,128bits]，但是注意在 BLE 中所有密钥都是 128bits 长度的密钥，所以如果长度不够，那么就将高位补 0，从而达到 128bits。

版权所有

3.6.3.1.1、Input 和 Output 能力

输入和输出的组合决定采用什么方式生成 TK。输入能力见表 3-17；输出能力见表 3-18 所示。

表 3-17 输入能力

输入能力	说明
不能输入	设备没有任何输入功能
可输入 YES/NO	设备只是有两个按钮能选择 Y 或 N，无论是物理按键还是触摸屏；另一种方式是只有一个按钮，当在一个限制时间内按下，表示选择 Y，否则表示 N。
可使用键盘输入	设备有输入 0-9 的值和一个确定按钮，当然按键也就包含了输入 YES/NO 的能力

表 3-18 输出能力

输出能力	说明
不能输出	设备不能显示或者不能传达出一个 6 位数的数值
输出数值	设备可以显示或者能够传达出一个 6 位数的数值

在同一个设备上输入输出能力的组合才能决定这个设备到底具备什么样的能力实现 TK 值。组合能力见表 3-19。

表 3-19 设备 IO 综合能力

输出能力 / 输入能力	不能输出	输出数值
不能输入	无输入输出能力	只有显示
可输入 YES/NO	无输入输出能力	显示且能输入 YES/NO
可使用键盘输入	只有键盘	键盘和显示

特征交换时，将自己的综合能力发送给对方设备，对方设备也会把自己的 IO 综合能力发送过来，最后根据两个设备的能力最终选择那种方式实现 TK 值的共享，也就是说 TK 值虽然需要共享，但是这个密码不是两个设备通过无线进行通信得到的，而是通过人为输入或者默认的方式实现的。那么两个设备 IO 能力综合会有哪些产生 TK 的方式呢？见表 3-20。

表 3-20 临时密钥(TK)产生的方式选择

发起者 应答者	只能显示	显示且能输入 YES/NO	只有键盘	无输入输出	键盘和显示
只能显示	只工作：加密链路不可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠
显示且能输入 YES/NO	只工作：加密链路不可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠
只有键盘	输入密码：应答者显示，发起者输入。加密链路可靠	输入密码：应答者显示，发起者输入。加密链路可靠	输入密码：应答者显示，发起者输入。加密链路可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠
无输入输出	只工作：加密链路不可靠	只工作：加密链路不可靠	只工作：加密链路不可靠	只工作：加密链路不可靠	只工作：加密链路不可靠
键盘和显示	输入密码：应答者显示，发起者输入。加密链路可靠	输入密码：应答者显示，发起者输入。加密链路可靠	输入密码：应答者显示，发起者输入。加密链路可靠	只工作：加密链路不可靠	输入密码：应答者显示，发起者输入。加密链路可靠

从表 3-20 可以知道，IO 能力的交换决定的 TK 值的方式只有两种，而协议规范中其实是有三种方式决定 TK 值的：

- Just Work：只工作----这时 TK 双方默认为 6 个 0，就是默认使用 0
- Passkey Entry:输入密码----这时 TK 为输入的 6 位数 000000-999999
- Out Of Band(OOB):带外----这个不懂，是通过另一种无线接入两个设备的网络，同时将 TK 进行传送。

3.6.3.1.2、Just Work:只工作

仅工作方式两设备使用的是默认的 TK 值，即 6 个 0。对于这种

方式是一个不可靠的加密链路，它不能防止 MITM 攻击。这种方式使用时可靠的前提是，确保在配对绑定是能保证没有 MITM 攻击，那么在之后的连接中加密的数据是无法被其它设备窃听的，也就是说这种方式保护的是将来的加密链路安全，但是不能保护配对绑定过程。

3.6.3.1.3、Passkey Entry:输入密码

上面有提到 TK 的共享并不会是通过无线传输的，而是通过人为方式使两个设备使用的临时密钥一样。对于输入密钥来说，实现的方式是：两个设备中，一个蓝牙设备在自己的显示屏上显示[000000,999999]之间的**随机 6 位数**；而操作人员看到这 6 位数后，将这 6 位数在另一个蓝牙设备中输入，从而实现两个设备的 TK 值一样。

这个过程能防止 MITM 攻击，但是它的保护还是受限制的，因为毕竟这种方式中的 TK 值只有 6 位随机数，还是有概率被攻击者成功攻击。但是如果在配对绑定过程没有被攻击，那么在未来的加密链路中一定是安全可靠的。

这里的输入值是 6 位数，假设输入的是“019655”，那么实际使用的 TK 值为 0x00000000000000000000000000004CC7。也就是用 0 补满到 128bits。

3.6.3.1.4、Out of Band:带外

带外是使用另一无线方式将数据传给蓝牙设备，如果带外本身能防止 MITM 的攻击，那么传送的 TK 值肯定是受保护的。而且这种方

版权所有

式下的 TK 值是 128bits 的随机数，被虽然还是有概率被第三方猜中，但是猜中 128bits 随机数的概率远比输入密码时的 6bits 的随机数要小。

这个虽然更加安全，但是对蓝牙设备也有一定的要求：这两个设备需要有能匹配的 OOB 接口。所以在特征交换时，还会传输一个信息告诉对方设备自己具不具备带外认证数据的能力即 OOB(Out Of Band:带外)认证数据可用性。

3.6.3.2、身份确认以及短期秘钥(STK)生产

配对的第 1 阶段通过特征交换仅仅得到 TK 值，而 TK 值是用来做什么的呢？在第 2 阶段用来作为密钥进行计算两个重要的值：

- 身份确认值
- 短期秘钥(STK)值

3.6.3.2.1、身份确认值计算

得到了 TK 值，但是为了保证和自己通信的设备是自己需要连接的设备，必须通过某些计算从而确定对方的身份。两个设备都需要计算确认值，从而确定对方是所需要的连接的设备。所以分为主机确认值/发起者确认值(*Mconfirm*)计算和从机确认值/响应者确认值(*Sconfirm*)计算。确认值计算使用到的函数为 c1 函数见式(3-3)。式中所有的参数有的由自己设备提供，有的是由对方设备提供。具体为:

Mconfirm = c1(TK, ：*短期秘钥*

Mrand,：发起者发送给响应者的随机数

版权所有

Pairing Request command, ：配对请求命令

Pairing Response command, ：配对应答命令

initiating device address type, ：发起者设备地址类型

initiating device address, ：发起者设备地址

responding device address type, ：应答者设备地址类型

responding devices address)：应答者设备地址

从机的确认值/响应者的确认值的计算：

Sconfirm = *c1*(*TK*, *Srand*, Pairing Request command, Pairing Response command, initiating device address type, initiating devices address, responding device address type, responding device address)和主机的确认值是一样的，只是将随机数变为了 Srand。

3.6.3.2.2、短期秘钥(STK)值计算

得到 TK 后的另一个作用是计算短期秘钥，短期秘钥的使用的函数为 s1 函数见式(3-4)。具体的 STK 计算如下：

STK = s1(TK, Srand, Mrand)

其中 *Srand* 和 *Mrand* 和计算确认值时使用的值一样。

短期密钥 STK 计算了干什么呢？为什么叫做"短期秘钥"呢？STK 存在的目的在于配对绑定过程的第 3 阶段不再使用明文进行数据传输，而是使用 STK 作为长期秘钥 LTK 将需要交互的数据进行加密，第 3 阶段传输是在未来加密链路中使用到的 LTK、IRK 以及 CSRK 等等密钥，所以必须进行加密处理，也就是说在绑定配对过程中，第 3 阶段

版权所有

就已经使用加密的密文传输。

然而 STK 或者 LTK 并不能直接作为将要发送的数据包进行加密的密钥，为了传输的数据包更加的安全，加密数据包的密钥是会话密钥 Session Key(SK)，也就是说会话密钥 SK 是用 STK 或者 LTK 当做密钥通过加密引擎函数 e 计算得到的，计算公式如下：

SK=e(LTK,(SKDslave||SKDmaste))

参数 LTK 即为长期密钥，上文提到 STK 可以代替 LTK 作为密钥计算 SK，但是什么情况下可以使用 STK 代替 LTK 呢？当两个设备第一次进行配对绑定时，在第 3 阶段就需要进行加密链路传输数据，而此时长期密钥 LTK 是没有共享的，所以需要通过第 2 阶段计算得到的 STK 作为长期密钥 LTK 来计算会话密钥 SK。也就是说只要加密链路传输数据包那么必须使用会话秘钥 SK 进行链路加密。那么具体判断使用 STK 和 LTK 作为密钥见 3.6.5 章节。

3.6.3.3、特定密钥计算

在配对绑定的第 3 个阶段传输就是两个设备商量好了的特定的密钥，然而是不是有个问题，这些密钥到底是从哪儿来的呢？所有密钥都是通过计算得到。

3.6.3.3.1、长期密钥 LTK 计算

长期密钥 LTK ***使用的函数是 d1 函数见式(3-6)。计算如下：***

LTK=d1(ER,DIV,0)=e(ER,0||DIV)

然而这里最难搞定的还是 ER 和 DIV 这两个参数。***ER(Encryption Root)在 nrf51822 中其实是一个 128bits 的伪随机数***，并掩膜在 Flash 中。协议原文：

ER is used to generate LTK and CSRK. ER can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in ER having 128 bits of entropy

这参数就这一句话，但是我在协议中却找了好长时间啊！DIV 它是“*Diversifying*”这个单词的前 3 个字母吧！意思为“分散器”。目的是将一些数据分散然后通过计算将数据还原。DIV 的计算如下：

DIV=EDIV XOR Y 或 EDIV=DIV XOR Y

EDIV 即为加密分散器，这个值是在配对绑定过程中是由从机发送给主机，而配对绑定完之后的连接是由主机发送给从机，从而计算出长期密钥 LTK。一方面在未来的连接过程 LTK 不通过明文发送保证 LTK 的安全性；另一方面从机不存储配对过程中交换的某些密钥信息，可以减少对从机硬件要求。计算式中是个鸡生蛋蛋生鸡的过程，到底从机是怎么产生 EDIV 的，目前我也不清楚，现在我怀疑是伪随机数，但是到底是不是伪随机没有从协议规范中找到根据。

Y 也是通过计算得到的，使用的是 dm 函数见式(3-5)。计算如下：

Y=dm(DHK,Rand)=e(DHK,Rand’) mod 216

后面 mod 2^{16} 的目的是保留结果的低 16 字节。Rand 和 EDIV 是同一个命令传输的，所以它的传输和 EDIV 传输的有相同的特征。

DHK 由 d1 函数计算得到，计算如下：

DHK=d1(IR,3,0)=e(IR,0||3)

IR(Identity Root)和 ER 一样，也是一个固定的值，在 nrf51822 中

也被固定在 flash 中，协议原文：

IR can be used to generate IRK and other required keys. IR can be assigned, randomly generated by the device during manufacturing or some other method could be used, that results in IR having 128 bits of entropy.

3.6.3.3.2、设备地址类型和身份解析密钥 IRK

先讲一下本来属于 GAP 层描述的设备地址类型。设备地址共分为两类：公共地址和随机地址。

- 公共地址

公共地址的组成如下：

LSB 24bits	24bits MSB
公司分配	公司标识符

- 随机地址

随机设备地址又分为两类：静态地址和私有地址。

➢ 静态地址

静态地址是满足一些要求的伪随机数：

1.地址的最高 2bits 必须为 1;

2.不能全为 0 或者全为 1。

LSB 48Bits	MSB	
Random part of static address	1	1

➢ 私有地址

私有地址又分为两类：不可解析地址和解析地址。

◆ 不可解析私有随机地址

不可解析私有随机满足：1.最高 2bits 为 0；2.不能全为 0 或者 1；3.不能等于静态地址和公共地址。组成如下：

版权所有

LSB	48Bits	MSB	
Random part of no-resolvable address		0	0

◆ 可解析私有随机地址

可解析私有随机地址的最高两 bits 为“10”，最高 bit 为 0。组成如下：

LSB 24bits	24bits	MSB	
Hash(哈希码)	Rand(随机数)	1	0

可解析私有随机地址的生产和解析都需要知道设备的身份解析密钥(IRK)，如果知道对方的 IRK 就能解析出对方的身份，本地设备通过 IRK 也能计算出一个可解析的随机地址，这就是身份解析密钥的功能。

● 可解析私有随机地址的产生过程

从可解析私有随机地址的组成可以知道地址由两部分组成：一个随机部分和一个使用 IRK 对该随机部分进行哈希计算获得的，hash 使用的是 ah 函数见式(3-2)，计算如下：

Hash=ah(IRK,rand) = e(IRK, rand’) mod 224

Rand 是一个伪随机数，那么 IRK 又是怎么来的呢？IRK 使用的 d1 函数，计算如下：

IRK=d1(IR,1,0)=e(IR,0||1)

IR 一般是一个固定的值，也就是说对于某个蓝牙设备一般 IRK 是不会变的。

● 可解析私有随机地址的解析过程

解析一般是接收到这个地址的设备解析，目的是确定远程设备是不是自己需要连接的设备。计算依然采用 ah 函数，只是其中的参数，

版权所有

一个是接收地址中的伪随机数，另一个参数是 IRK，这个参数是在第一次配对绑定过程中获得的 IRK 并保存在数据库中。当计算出的 hash 和接收到的地址中的 hash 一样，则说明这个设备是自己需要连接的设备。

这里需要注意：在从设备中一般可以有存放多个 IRK，因为它可能和多个主机进行过配对绑定。所以在计算过程中需要遍历整个 IRK。

3.6.3.3.3、连接签名解析密钥 CSRK

在数据签名认证时需要使用到 CSRK 密钥，它使用的是 d1 函数见式(3-6)产生。计算如下：

CSRK=d1(ER,DIV,1) =e(ER,1||DIV)

它和 LTK 生成的只有一个参数不同。其他参数见 3.6.3.3.1 节。

3.6.3.3.4、签名计算

在属性协议 ATT 章节的 3.4.3.11 中的“签名写命令”中提到连接签名的使用场合。其实解释得也是不明不白，写到这节我也有个问题 Message Authentication Code (MAC)和 Message Integrity Check (MIC)到底是啥关系？在协议规范中有这么一段话：

> This specification uses the same notation and terminology as the IETF RFC except for the Message Authentication Code (MAC) that in this specification is called the Message Integrity Check (MIC) to avoid confusion with the term Media Access Controller.
> ……
> There is an optional Message Integrity Check (MIC) value that is used to authenticate the data PDU.

大概意思是为避免和 4.0 协议中 Media Access Controller 这个缩写

产生冲突，在 4.0 协议中的 Message Authentication Code (MAC)改名为 Message Integrity Check (MIC)。也就是说 MIC 就是 MAC，即签名认证所有计算其实在数据 PDU 中就是信息完整性检查(MIC)这 4 个字节。

那么这个 MAC 到底是怎么来的呢？协议中有如下公式(3-7)：

$$MAC = CMAC(K, M, Tlen) \quad (3\text{-}7)$$

其中 CMAC 函数是一个标准函数，在芯片中也是由具体硬件完成，所以不再细挖，后文中也会讲解。其参数 k 即为连接签名解析密钥 CSRK；Tlen 为固定的 64bits 也就是 8 字节长度；而 M 是一个可变长的数，它包括包计数器和具体数据这两个部分拼接而成，计数如下：

M = data || SignCounter

例如，假设需要签名的数据是 7 字节的数据*'3456789ABCDEF1'*，而计数器 *SignCounter* 为 67653874 (0x040850F2)，那么 M 的值为 '3456789ABCDEF1F2500804'。

从上文中可以知道，前文提到的数据包中包含有一个包计数值，实际上并不是直接发送一个包计数值，而是通过计算包含在 MAC 即 MIC 中的。在数据 PDU 中的 MIC 只有 4 字节，它是 MAC 的最高 32bits，也就是 MIC 使用 MAC 的最高 4 字节。

3.6.4、加密标准 AES-CCM

BLE 中使用的是 AES(Advanced Encyption Standard)即高级加密标准。在前文提到的加密函数 e(式(3-1))，它是 AES 标准中的 Electronic Codebook mode encryption (ECB)即电子密码本模式。计算 MAC 是用到

的函数 CMAC，它是由 Counter with Cipher Block Chaining-Message Authentication Code (CCM)实现。AES-CCM 是计数器密码块链消息认证码模式，融合有 3 种技术：AES 加密引擎、计数模式和消息认证。

在 nrf51822 中 AES ECB 和 AES-CCM 都是由硬件完成，那么 AES-CCM 需要哪些参数呢？共 3 个:CCM nonce(Nonce 是 Number used once 或 Number once 的缩写)、counte-mode blocks 和 encryption blocks。其中 counte-mode blocks 和 encryption blocks 是通过硬件计算得到，这两个块的参数也会用到 CCM nonce，所以只介绍 CCM nonce 的组成。

如图 3-67 所示，nonce 由包计数器、方向位和一个初始化向量(IV)构成：

$$nonce=PacketCount||Direction||IV$$

PacketCount 包计数器共有 39bits，方向位 1bit，而 IV 共有 64bits 即 8 字节，这 8 字节两个配对设备各提供 4 字节。这 3 部分总长为 13 字节。在 Nonce4 区字节中的低 7bits 属于包计数器，而最高 bit 为方向位：1 表示数据由主机发送给从机，0 表示数据由从机发给主机。关于包计数器 *PacketCount* 在协议中规定：

> The Link Layer shall maintain one *packetCounter* per Role for each connection. For each connection, the *packetCounter* shall be set to zero for the first encrypted Data Channel PDU sent during the encryption start procedure. The *packetCounter* shall then be incremented by one for each new Data Channel PDU that is encrypted. The *packetCounter* shall not be incremented for retransmissions.
> **实践证明：第一次加密启动时 *PacketCount* 为 0。只要发送的 PDU 是新的数据包，那么包计数器加加后再加密；只要解密的包是接收的对方发送的新的数据包，那么包计数器加加后再解密，不管是不是在同一个连接事件中的多个数据包通信，只要是新的数据包就进行加加。同时对于同一个设备加密包计数器和解密包计数器是两个独立的变量，并不是在同一个包计算器上进行加加。重发包和**

空包是不用加加的。

Octet	Field	Size (octets)	Value	Description
0	Nonce0	1	variable	Octet0 (LSO) of *packetCounter*
1	Nonce1	1	variable	Octet1 of *packetCounter*
2	Nonce2	1	variable	Octet2 of *packetCounter*
3	Nonce3	1	variable	Octet3 of *packetCounter*
4	Nonce4	1	variable	Bit 6 – Bit 0: Octet4 (7 most significant bits of *packetCounter*, with Bit 6 being the most significant bit) Bit7:*directionBit*
5	Nonce5	1	variable	Octet0 (LSO) of IV
6	Nonce6	1	variable	Octet1 of IV
7	Nonce7	1	variable	Octet2 of IV
8	Nonce8	1	variable	Octet3 of IV
9	Nonce9	1	variable	Octet4 of IV
10	Nonce10	1	variable	Octet5 of IV
11	Nonce11	1	variable	Octet6 of IV
12	Nonce12	1	variable	Octet7 (MSO) of IV

图 3-67 CCM nonce format

这里注意一个关于大小端模式的问题：**在 BLE 协议中规定，除了数据包中的 CRC 和 MIC 是先发送高字节再发送低字节外，发送到空中的数据都是先发送低字节最后发送高字节。这个规定在任何时候都不会改变的。对于 nrf51822 来说，其内核是使用小端模式，所有的数据都是低字节放低地址，高字节放高地址。而对于 AES 加密引擎函数 e 则有着相反的规定，所有的明文、密钥以及生成的密文都是按大端模式存放，也就是数组第 0 个元素存放的是明文、密钥以及生成的密文的最高字节，数组的最后一个元素存放的它们的最低字节。而对于 CCM nonce 的格式在 RAM 中还是按照如图 3-67 所示存储，即按**

照小端模式存储。

3.6.5、完整加密过程图表

加密过程不仅仅是 SM 层单独完成的，还有一些参数是由 LL 层进行传输的，像启动和停止加密都是有 LL 层进行控制的。为了使后续章节更容易理解，先来看看整个加密过程如图 3-68 所示。

图中将安全管理配对绑定过程全部描述，虚线部分发送的数据包表示根据具体情况判断是否需要发送该包数据。其中包含有第 1 阶段的交换特征值并得到 TK 值，第 2 阶段交换确认值以及计算确认值所用到的随机数，首先主机发送确认值给从机，从机也发送确认值给主机，之后主机发送随机数给从机，当从机接到随机数后，开始计算确认值，当计数的确认值和主机发过来的确认值一样，则从机也发送随机数给主机，否则就不发送随机数给主机，因为有可能受到攻击。如果主从都交换了计算确认值的随机数后，各自会计算出短期密钥 STK。

计算出 STK 后主机通过链路层使用 LL_ENC_REQ 发起加密请求，并将用来计算会话密钥 SK 的参数会话密钥分散值 SKDm 发送给从机，以及 CCM 使用的初始化向量 IVm 值、计算 LTK 的 EDIV 和 RAND 参数都发送给从机。那么 SKDm 和 IVm 值是怎么来的呢？在 4.1 协议规范中有如下一段话：IV 和 SKD 都为伪随机数。

IVm shall be a 32 bit random number generated by the Link Layer of the master. SKDm shall be a 64 bit random number generated by the Link Layer of the master. Both IVm and SKDm shall be generated using the requirements for random number generation

……

IVs shall be a 32 bit random number generated by the Link Layer of the slave. SKDs shall be a 64 bit random number generated by the Link Layer of the

slave. Both IVs and SKDs shall be generated using the requirements for random number generation

这里的 EDIV 和 Rand 参数就是决定 STK 作为 LTK 计算 SK 的评判标准，当 EDIV 和 Rand 都为 0 时，表示这个加密请求是两设备的第一次配对绑定，所以要用 STK 作为 LTK 使用；当 EDIV 和 Rand 不为 0 时，表示之前这两个设备已经配对过，这个值是在第一次配对时从机发送给主机的，从而是不是可以知道，只有第一次配对绑定过程才有安全管理协议参与，之后的连接只需链路层进行控制。

从机通过 LL_ENC_RSP 加密应答把计算 SK 相关的参数也发送给主机，这时主从之间通过 LL_START_ENC 进行 3 次加密握手。第 1 次，从机通过明文的方式将 LL_START_ENC_REQ 开始加密请求发送给主机，并将自己接收数据包方式设置为加密接收；第 2 次，当主机接收到从机的开始加密请求的明文后，主机发送加密的开始加密请求应答包 LL_START_ENC_RSP 给从机，并将自己的接收设置为加密接收；第 3 次，因为从机已经将接收设置为了加密模式，所以应该能成功接收到主机发送的密文 LL_START_ENC_RSP，之后从机发送加密的 LL_START_ENC_RSP 包给主机从而完成 3 次加密握手过程。

配对绑定的第 3 阶段是根据第 1 阶段的特征交换要求，选择发送那些密钥，所以图中都是用虚线进行表示的。

版权所有

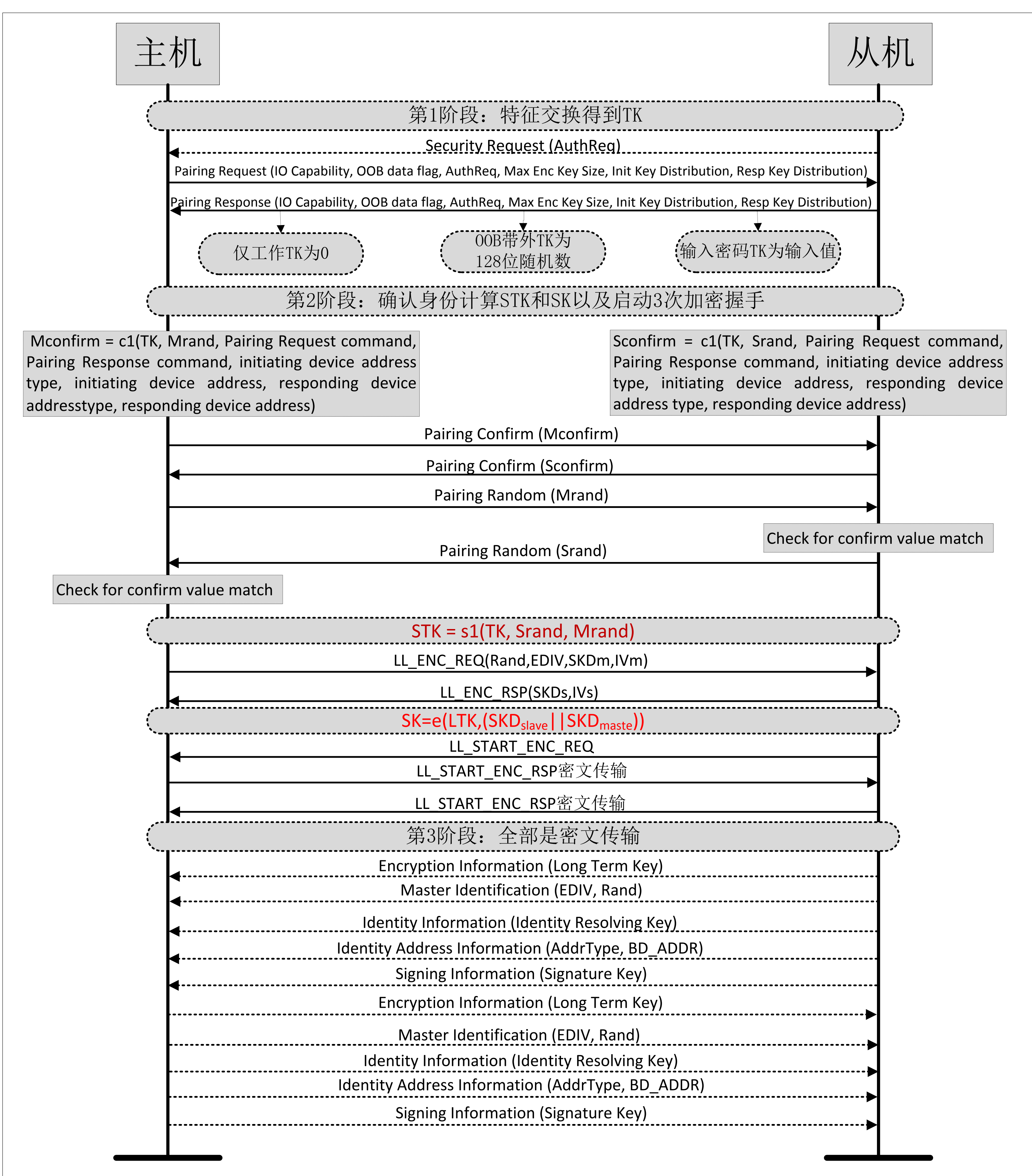


图 3-68 安全管理配对绑定数据包交互全过程

3.6.6、安全管理传输协议

安全管理协议就是两个设备之间传输需要共享数据时商量好了一种固定格式的命令或者数据包格式。

3.6.6.1、安全管理命令包格式

安全管理命令包格式如图 3-69 所示。

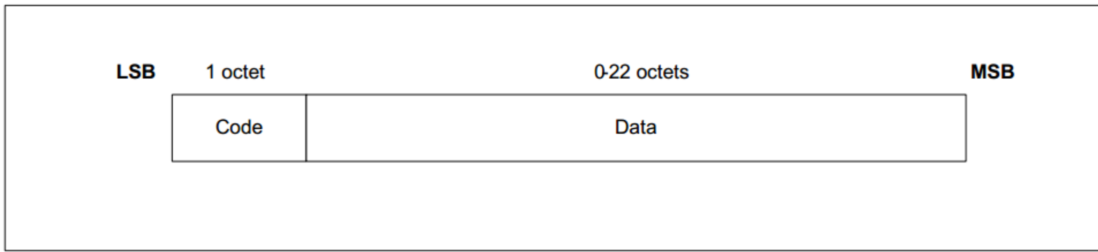


图 3-69 安全管理命令包格式

1 字节的命令码，0-22 字节的数据参数。其中命令码共有 10 个命令，如表 3-21 所示。

表 3-21 安全管理协议命令码

命令码	描述	命令码	描述
0x00	保留	**0x07**	Master Identification
0x01	Pairing Request	**0x08**	Identity Information
0x02	Pairing Response	**0x09**	Identity Address Information
0x03	Pairing Confirm	**0x0A**	Signing Information
0x04	Pairing Random	**0x0B**	Security Request
0x05	Pairing Failed	**0x0c-0xFF**	保留
0x06	Encryption Information		

3.6.6.2、配对请求 Pairing Request 和配对应答 Pairing Response

配对请求和应答的包格式一样如图 3-70 所示，只有命令码不同。空中采集的数据包如图 3-71 所示。

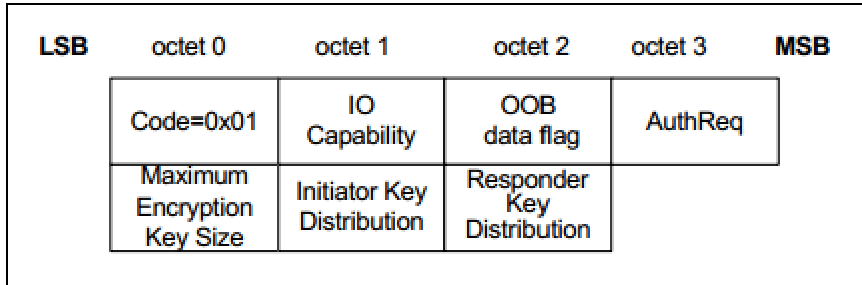


图 3-70 配对请求包

```
Bluetooth Security Manager Protocol
   Opcode: Pairing Request (0x01)
   IO Capability: Keyboard, Display (0x04)
   OOB Data Flags: OOB Auth. Data Not Present (0x00)
 ⊟ AuthReq: Bonding, MITM
      .... ..01 = Bonding Flags: Bonding (0x01)
      .... .1.. = MITM Flag: 1
   Max Encryption Key Size: 16
 ⊟ Initiator Key Distribution: LTK IRK
      .... ...1 = Encryption Key (LTK): 1
      .... ..1. = Id Key (IRK): 1
      .... .0.. = Signature Key (CSRK): 0
 ⊟ Responder Key Distribution: LTK IRK
      .... ...1 = Encryption Key (LTK): 1
      .... ..1. = Id Key (IRK): 1
      .... .0.. = Signature Key (CSRK): 0
```

图 3-71 Sniffer 空中采集的配对请求包

- IO Capability IO 能力 1 字节

具体哪些能力在 3.6.3.1.1 节已经讲解，这里将每一种 IO 能力对应到一个值，如表 3-22 所示。

表 3-22 IO 能力码

值	描述	值	描述
0x00	只显示	**0x03**	无输入输出
0x01	显示 yes/no	**0x04**	键盘和显示
0x02	只有按键	**0x05-0xFF**	保留

- OOB data flag 带外数据标志 1 字节

这个标志表示自己支不支持 OOB。当标志为 0 时表示不支持 OOB；当为 1 时表示支持 OOB，所需的 TK 由 OOB 提供。

- AuthReq 认值请求标志 1 字节

AuthReq 是 Authentication Requirements 的意思。它这个字节划分为 3 个区域，如图 3_72 图 3-72 所

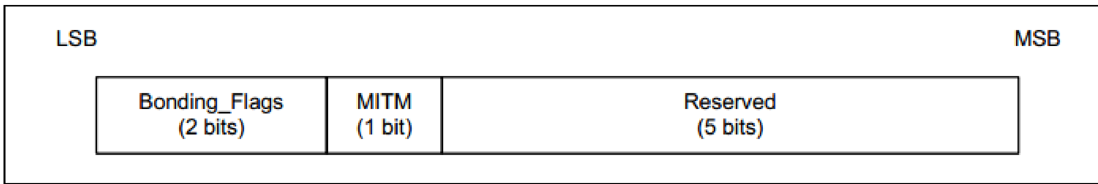


图 3-72 认值请求标志位

➢ Bonding_Flags 绑定标志共 2bits 组合如下表 3-23。

表 3-23 绑定类型

Bonding_Flags(b1b0)	绑定类型
00	无需绑定
01	需绑定
10	保留
11	保留

➢ MITM 防止中间人攻击标志 1bit

当 MITM=0 时，无需防止中间人攻击；当 MITM=1 时，认证需要防止中间人攻击。

- Maximum Encryption Key Size 最大密钥长度 1 字节

最大密钥长度大小为 7-16 字节，如果少于 128bits 的密钥，需要通过高位补 0 填充。

- Initiator Key Distribution 发起者能分配的密钥和 Responder Key Distribution 希望应答者能发送给自己的密钥

这两个区域各为 1 字节，它们有共同的格式如图 3-73 所示。

版权所有

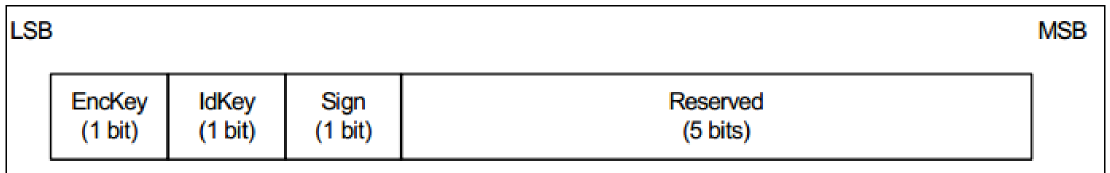


图 3-73 BLE 密钥分配格式

共 3 个密钥分配，各占 1bit。

- EncKey 加密密钥 LTK

为 1 时需提供或者能提供 LTK 给对方，通过 Encryption Information command 发送 LTK，同时还必须通过 Master Identification command 发送 EDIV 和 Rand 参数。

- IdKey 身份解析密钥 IRK

为 1 时需提供或者能提供 IRK 给对方，通过 Identity Information command 发送 IRK，同时还必须通过 Identity Address Information 发送公共地址或者静态随机地址。

- Sign 签名认证 CSRK

为 1 时需要提供或者能提供 CSRK 给对方，通过 Signing Information command 发送 CSRK。

3.6.6.3、配对确认值 Pairing Confirm

主从设备在交换特征后会发送配对确认值给对方，目的是验证身份，从而为后面计算 STK 做准备。它的包格式如图 3-74。空中采集的数据包如图 3-75 所示。

它的命令码是 0x03。后面紧跟的是 128bits 即 16 字节的确认值。这个确认值的计算见 3.6.3.2.1 节。

版权所有

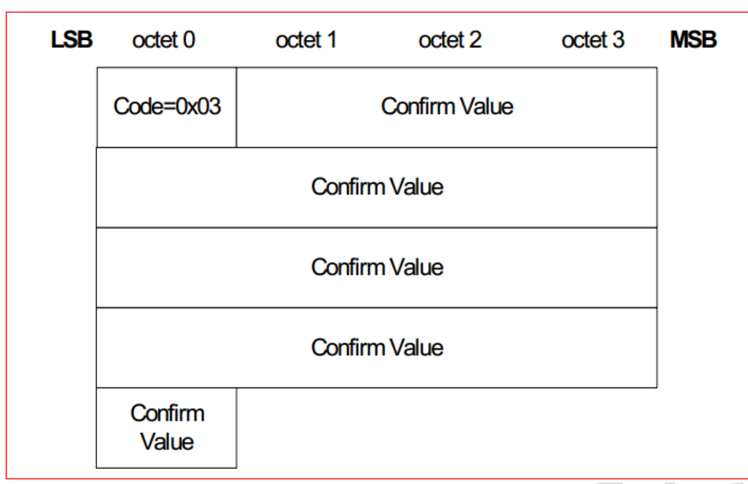


图 3-74 配对确认值包格式

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Confirm (0x03)
  Confirm Value: 52bc2ace1bc72111bcbb62239897a201
```

图 3-75 Sniffer 采集空中的确认值包

3.6.6.4、配对随机数 Pairing Random

交换确认值之后，需要交换配对随机数，这个随机数是一个128bits 的伪随机数。得到伪随机数就能算出确认值。包格式如图 3-76。空中采集的数据包如图 3-77。

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x04	Random value			
	Random value				
	Random value				
	Random value				
	Random value				

图 3-76 配对随机数包格式

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Random (0x04)
  Random Value: e47edf717dfc41502b0232f817f70a78
```

图 3-77 Sniffer 采集空中的随机数包

3.6.6.5、配对失败 Pairing Failed

配对的过程中因为各种原因会出现失败的情况，它的包格式如图 3-78 所示。命令码 1 字节，失败原因 1 字节。具体的原因代码见表 3-24 所示。

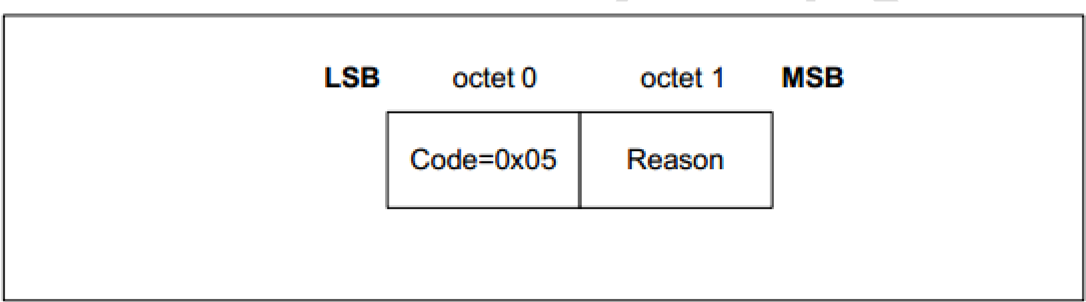


图 3-78 配对失败包格式

表 3-24 配对失败原因码

原因码	原因名称	说明
0x00	保留	保留
0x01	密码输入失败	用户输入密码失败，例如取消密码输入操作
0x02	OOB 不可用	OOB 数据不是有效的
0x03	认证需求	由于 IO 能力问题出现认证需求不能满足
0x04	确认值失败	收到的确认值和计算的确认值不一样
0x05	配对不支持	设备不支持配对
0x06	密钥大小	密钥大小不满足规定的密钥长度
0x07	命令不支持	接收到一个 SMP 不支持的命令
0x08	不明原因	配对失败由于某种不明原因
0x09	再次配对	配对或者认证不允许两次短时间内连续有同样的请求
0x0A	无效参数	对于 SMP 每一个命令有固定的参数格式，参数长度无效或超出指定的范围
0x0B-0xFF	保留	保留

3.6.6.6、加密信息 Encryption Information

加密信息用来传输特定密钥分配中分配未来连接使用到的 LTK。机密信息命令只有当链路使用已经生产的 STK 加密或者重新加密时传输。它是 128bits 的即 16 字节长度。它的命令包格式如图 3-79 所示。空中采集包如图 3-80。

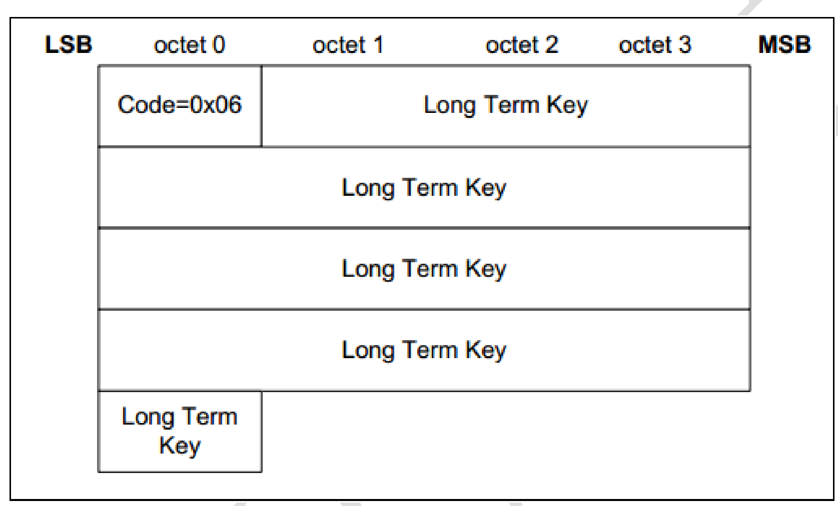


图 3-79 加密信息包格式

```
Bluetooth Security Manager Protocol
  Opcode: Encryption Information (0x06)
  Long Term Key: 0e0389f36884277f0f7c1164d20cca2b
```

图 3-80 Sniffer 采集空中 LTK 包

3.6.6.7、主机鉴定 Master Identification

这个命令其实是由从机发送给主机的，那为什么叫“主机鉴定”，其实这个命令第一次加密时有从机将 2 字节的 EDIV 和 8 字节的伪随机数 Rand 发送给主机，在之后的连接中，因为从机不保存 LTK，所以主机需要将这个两个值发送给从机，而从机接收到这两个值，计算出 LTK 和主机能进行加密通信，那么就从机就能鉴定这个主机。所以

叫做“主机鉴定”命令。它的包格式如图 3-81。空中采集包如图 3-82。

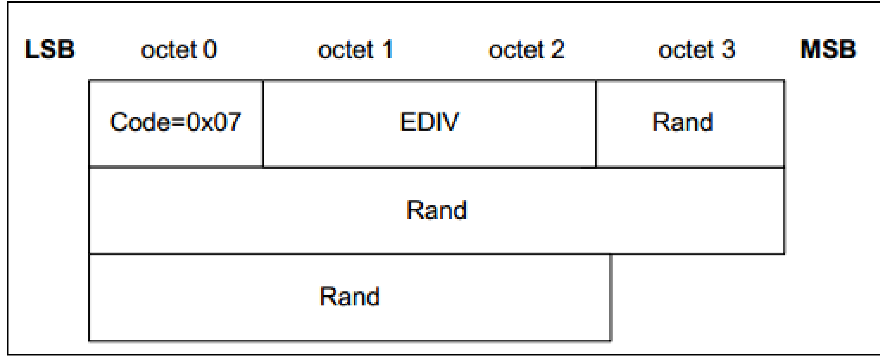


图 3-81 主机鉴定包格式

```
Bluetooth Security Manager Protocol
   Opcode: Master Identification (0x07)
   Encrypted Diversifier (EDIV): 0xe688
   Random Value: 0f622949fff5b96e
```

图 3-82 Sniffer 采集主机鉴定空中包

3.6.6.8、身份信息 Identity Information

身份信息发送的是身份解析密钥 IRK。它的命令包格式如图 3-83。空中采集包如图 3-84。

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x08	Identity Resolving Key			
	Identity Resolving Key				
	Identity Resolving Key				
	Identity Resolving Key				
	Identity Resolving Key				

图 3-83 身份信息包格式

```
Bluetooth Security Manager Protocol
  Opcode: Identity Information (0x08)
  Identity Resolving Key: b0aecb341ad8213a8ccc51a22d61edfb
```

图 3-84 Sniffer 采集空中身份信息包

3.6.6.9、身份地址信息 Identity Address Information

身份解析密钥 IRK 计算能识别对方设备的随机地址的原因是通过解析随机地址后再和对方真正的设备地址进行对比，从而进一步实现隐私。那么真正的设备地址就是通过这个身份地址信息传给对方的。包格式如图 3-85 所示。

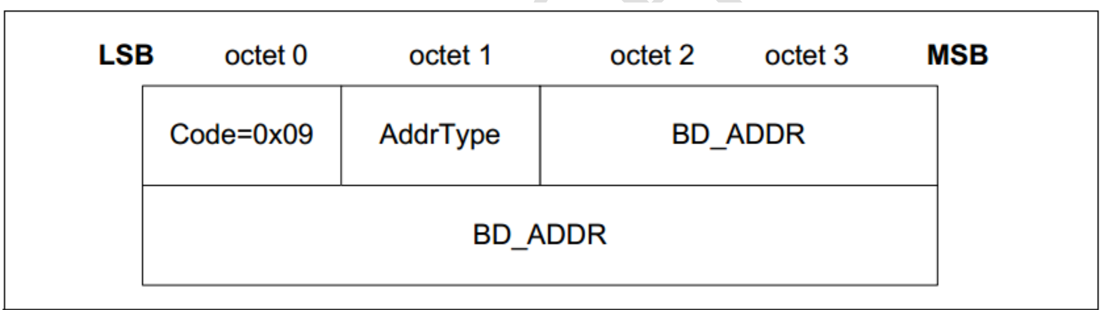


图 3-85 身份地址信息包格式

它包含有 1 字节的地址类型(见 3.6.3.3.2)和 6 字节的 BD_ADDR。如果 BD_ADDR 为公共地址或者全为 0，那么设置 AddrType 为 0x00；如果 BD_ADDR 为静态随机设备地址，那么设置 AddrType 为 0x01。空中采集的身份地址信息包如图 3-86 所示。

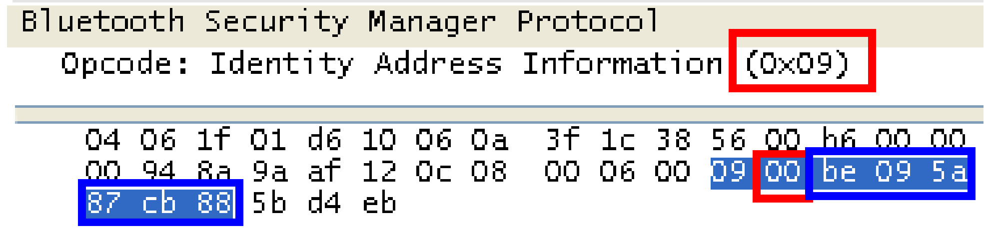


图 3-86 Sniffer 采集空中身份地址信息包

3.6.6.10、签名信息 Signing Information

签名信息就是传输连接签名解析密钥 CSRK 用。它的命令包格式如图 3-87 所示。

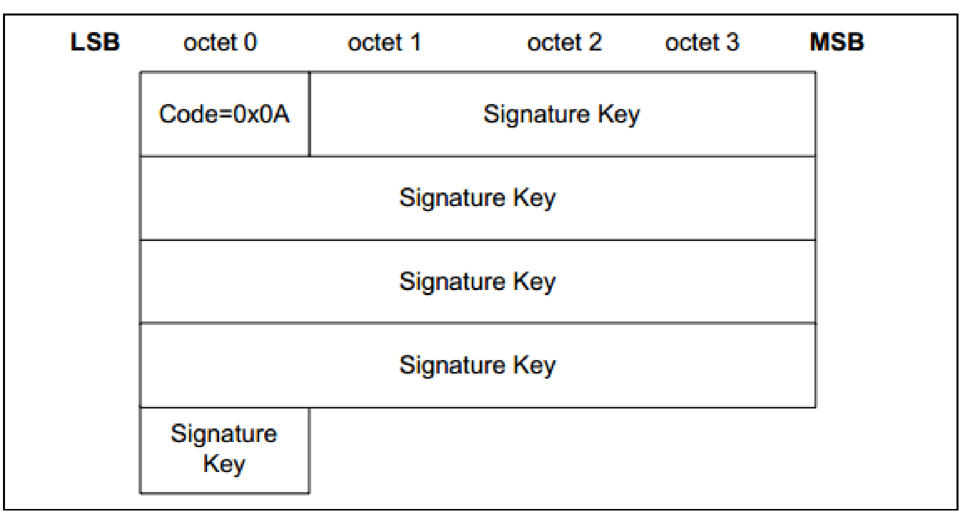


图 3-87 签名信息包格式

3.6.6.11、安全请求 Security Request

这个包是从机用来向主机请求安全属性初始化的命令。它的包格式如图 3-88 所示。包含有 1 字节的 AuthReq，这个和配对请求包(见 2.6.6.2 节)中的 AuthReq 字段一样。

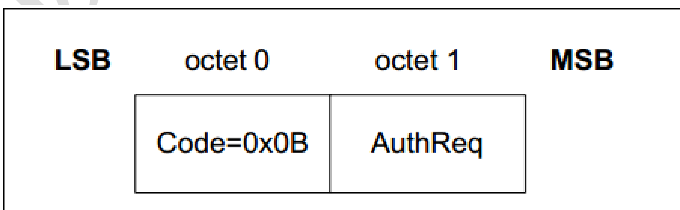


图 3-88 安全请求信息包

3.6.7、NRF51822 加密硬件模块

Nrf51822 的加密相关的硬件分为 3 个模块，分别为：AES ECB 电子密码本、AES_CCM 和快速地址解析模块 Accelerated Address Resolver (AAR)。

3.6.7.1、电子密码本 AES Electronic Codebook mode encryption

电子密码本 ECB 即为加密中用到的加密引擎函数 e。在 nrf51822 中的这个硬件模块需要的参数明文和密钥以及输出的密文都是保存在 RAM 中，通过 DMA 进行控制。也就是程序只需要将密钥、明文和密文指定某一地址传给模块的相关寄存器，启动加密后生产的密文可到指定的地址去取。在 RAM 中 ECB 的数据结构如图 3-89 所示。

Property	Address offset	Description
KEY	0	16 byte AES key
CLEARTEXT	16	16 byte AES cleartext input block
CIPHERTEXT	32	16 byte AES ciphertext output block

图 3-89 ECB 在 RAM 中的数据结构

3.6.7.1.1、ECB 程序设计

ECB 模块共有 3 个函数，数据结构初始化、密钥设置函数和加密函数。

```
/*ECB 数据结构定义*/
uint8_t    ecb_data[48];
uint8_t* ecb_key;
uint8_t* ecb_cleartext;
uint8_t* ecb_ciphertext;
```

```
/*ECB 数据结构初始化*/
bool nrf_ecb_init(void)
{
    ecb_key = ecb_data;
    ecb_cleartext    = ecb_data + 16;
    ecb_ciphertext = ecb_data + 32;

    NRF_ECB->ECBDATAPTR = (uint32_t)ecb_data;
    return true;
}
/*ECB 加密函数*/
bool nrf_ecb_crypt(uint8_t * dest_buf, const uint8_t * src_buf)
{
      uint32_t counter = 0x1000000;
      if(src_buf != ecb_cleartext)
      {
          memcpy(ecb_cleartext,src_buf,16);
      }
      NRF_ECB->EVENTS_ENDECB = 0;
      NRF_ECB->TASKS_STARTECB = 1;
      while(NRF_ECB->EVENTS_ENDECB == 0)
      {
        counter--;
        if(counter == 0)
        {
            return false;
        }
      }
      NRF_ECB->EVENTS_ENDECB = 0;
      if(dest_buf != ecb_ciphertext)
      {
          memcpy(dest_buf,ecb_ciphertext,16);
      }
      return true;
}
/*ECB 密钥设置*/
void nrf_ecb_set_key(const uint8_t * key)
{
    memcpy(ecb_key,key,16);
}
```

3.6.7.2、AES CCM Mode Encryption (CCM)

对于 AES-CCM 的原理可以百度，它是将上一个的密文当做下一个加密的密钥串接而成的加密方式。Nrf5188 已经将其模块化，主要的难点在于这个模块和 Radio 的配合进行数据包的解密加密。

3.6.7.2.1、AES-CCM 模块工作流程

AES-CCM 工作流程非常简单，在每个包的解密和加密首先需要计算密钥流，产生这次加密解密要用到的密钥，然后用密钥进行加解密。具体操作是通过 KSGEN 任务启动密钥流的生成，生成完毕会产生 ENDKSGEN 事件，然后启动 CRYPT 任务进行加解密，完成后产生 ENDCRYPT 事件。AES-CCM 工作流程示意如图 3-90 所示。

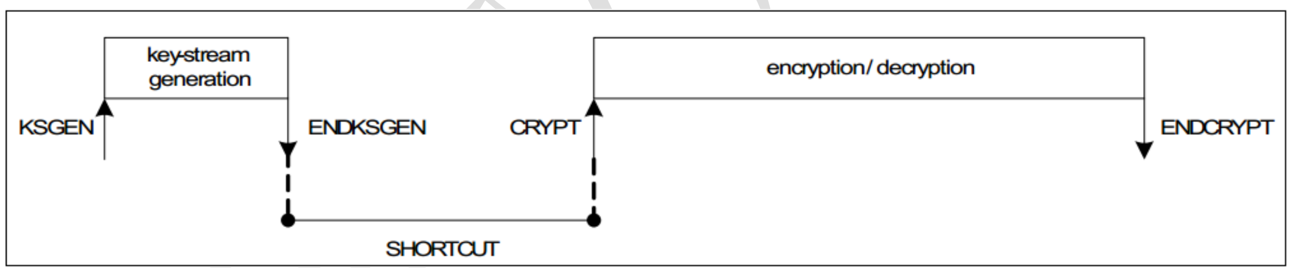


图 3-90 AES-CCM 工作流程示意

3.6.7.2.2、AES-CCM 模块加密过程

在数据包加密过程中，AES-CCM 读取待加密数据包在 RAM 中的地址，对于 AES-CCM 模块是读取 INPUT 寄存器所指向的地址，加密后的包会附加 4 字节的信息完整性检查(MIC)，同时 AES-CCM 会调整报头中长度区域的长度值，将这 4 字节添加到长度域中，最后将加密

后的数据包存放到 AES-CCM 模块中的 OUTPUT 寄存器所指向的地址。加密过程示意如图 3-91 所示。

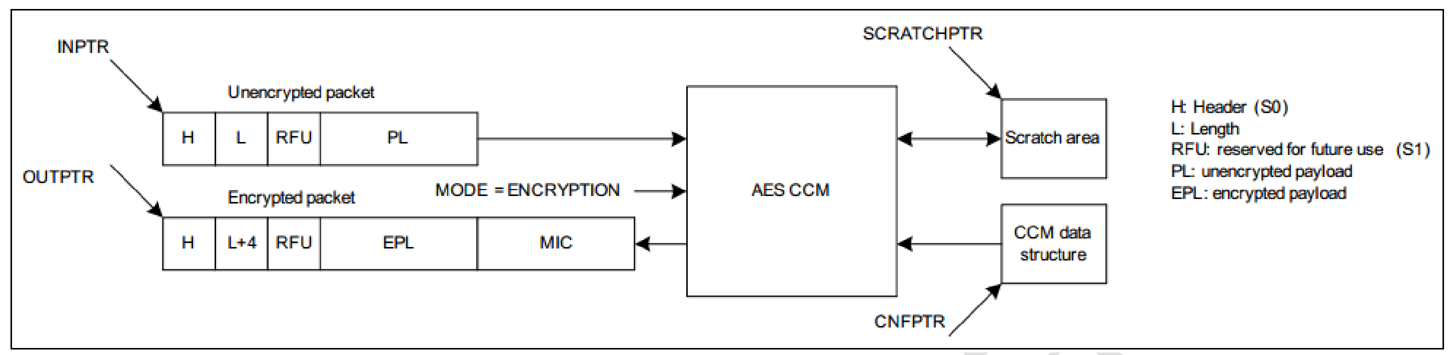


图 3-91 AES-CCM 模块加密过程

3.6.7.2.3、AES-CCM 模块解密过程

在数据包解密过程中，AES-CCM 读取待解密数据包在 RAM 中的地址，对于 AES-CCM 模块是读取 INPUT 寄存器所指向的地址，解密后的包会去掉 4 字节的信息完整性检查(MIC)并在 AES-CCM 模块中的 MIC 状态标志(MICSTATUS)中标示出 MIC 验证是否正确，同时 AES-CCM 会调整报头中长度区域的长度值，将这 4 字节在长度域中减掉，最后将解密密后的数据包存放到 AES-CCM 模块中的 OUTPUT 寄存器所指向的地址。解密过程示意如图 3-92 所示。

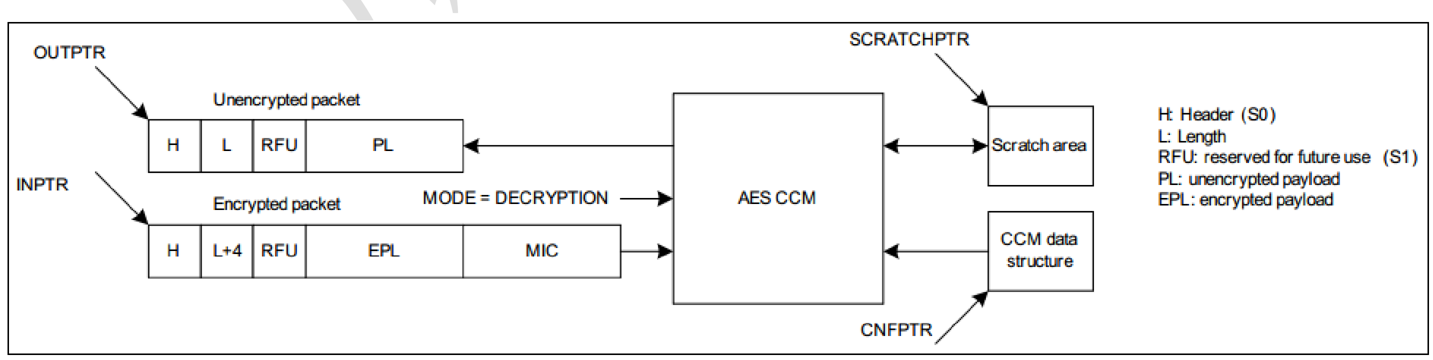


图 3-92 AES-CCM 模块解密过程

注意：空包通过 AES-CCM 模块后是不经过任何修改的，空包是不加解密的。AES-CCM 模块限制最大加密解密净荷的字节数为 27 字节。解密时报头的长度域必须大于 4，至少有 1 字节的净荷和 4 字节的 MIC，否则将产生 MIC 出错的状态标志。

3.6.7.2.4、CCM 数据结构

在图 3-91 和图 3-92 中有一个 CNFPTR 指向 CCM 数据结构，CNFPTR 是 AES-CCM 模块中的一个寄存器，这个寄存器中存放的数一个指向 RAM 中 CCM 结构体的地址。在 3.6.4 节中的图 3-67 就是 CNFPTR 所指向 CCM 数据结构。在 nrf51822 中定义的 nonce 的结构如图 3-93 所示。

Property	Address offset	Description
KEY	0	16 byte AES key
PKTCTR	16	Octet0 (LSO) of packet counter
	17	Octet1 of packet counter
	18	Octet2 of packet counter
	19	Octet3 of packet counter
	20	Bit 6 – Bit 0: Octet4 (7 most significant bits of packet counter, with Bit 6 being the most significant bit) Bit7: Ignored
	21	Ignored
	22	Ignored
	23	Ignored
DIRECTION	24	Bit 0: Direction bit Bit 7 – Bit 1: Zero padded
IV	25	8 byte initialization vector (IV) Octet0 (LSO) of IV, Octet1 of IV, … , Octet7 (MSO) of IV

图 3-93 nrf51822 中 CCM 数据结构

在图 3-93 中的结构和图 3-67 中的结构一样的，只是在 nrf51822 中将密钥 KEY 也放在了这个结构中，其余的也是包含包计数器、方向位和初始化向量 IV。

在图 3-91 中的 INPUT 和 OUTPUT 指向的结构分别如图 3-94 和图 3-95 所示。注意“Address offset”是地址偏移不是占有的字节数。

Property	Address offset	Description
HEADER	0	Packet Header
LENGTH	1	Number of bytes in unencrypted payload
RFU	2	Reserved Future Use
PAYLOAD	3	0 to 27 bytes unencrypted payload

图 3-94 AES-CCM 模块加密过程 INPUT 结构

Property	Address offset	Description
HEADER	0	Packet Header
LENGTH	1	Number of bytes in encrypted payload including length of MIC Note: LENGTH will be 0 for empty packets since the MIC is not added to empty packets
RFU	2	Reserved Future Use
PAYLOAD	3	0 to 27 bytes encrypted payload
MIC	3 + payload length	ENCRYPT: 4 bytes encrypted MIC Note: MIC is not added to empty packets

图 3-95 AES-CCM 模块加密过程 OUTPUT 结构

涉及到的 SCRATCHPTR 指针所指向的是这些模块计数时产生的中间数据，在程序中它是一个至少 43 字节的内存空间。

2.6.7.2.5、AES-CCM 模块要求 RADIO 的配置

AES-CCM 和 RADIO 是并行工作的，不需要 CPU 的参与，为了是这两个模块恰当配合，RADIO 需要进行如表 3-25 所示的配置。

表 3-25 启用加密 RADIO 的配置

RADIO 参数	值	说明
PCNF0.S0LEN	1	报头为 1 字节，对应图 3-94 中的 HEADER
PCNF0.LFLEN	5	连接状态下长度域为 5bit，对于 RAM 中占 1 字节，对应图 3-94 中的 LENGTH
PCNF0.S1LEN	3	连接状态下保留 3 比特，对于 RAM 中占 1 字节，对应图 3-94 中的 RFU
MODE	1_MBIT_QPSK	也就是蓝牙传输的 1Mbps 数据速率

PCNF1.BALEN	3	接入地址长度 32bits 即 4 字节
CRCCNF.LEN	3	CRC 长度为 24bits 即 3 字节

2.6.7.2.6、加密包在 RADIO 中传输模式

当 AES-CCM 加密并通过 RADIO 传输时，RADIO 必须从 AES-CCM 写入的内存地址读取加密包。所以 OUTPUT 指针和 RADIO 中的 PACKETPTR 指针指向的 RAM 中的同一地址。见图 3-96 所示。

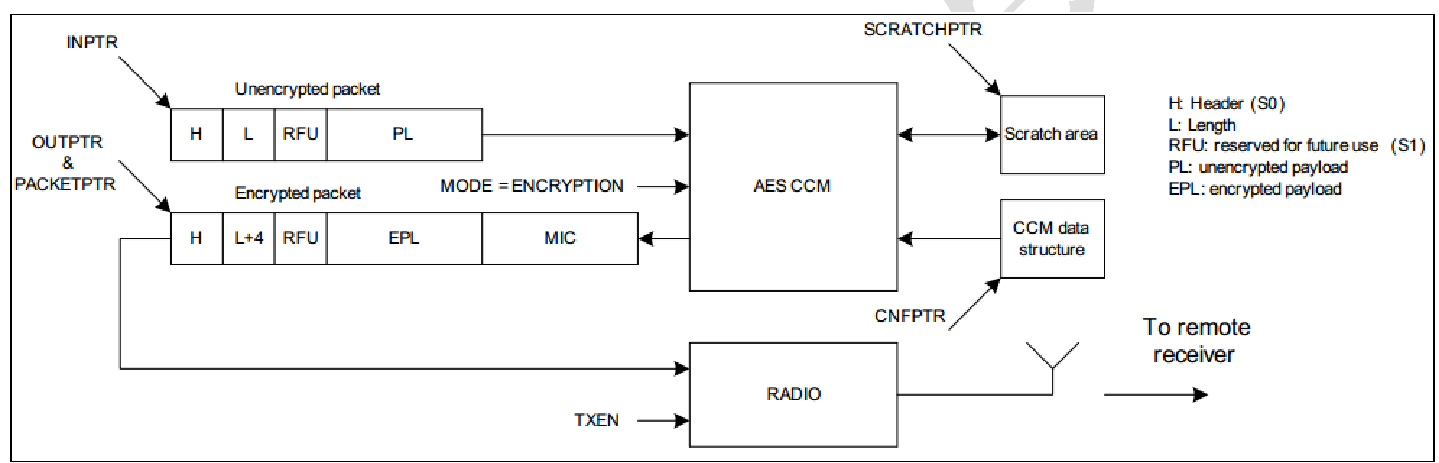


图 3-96 AES-CCM 和 RADIO 组合传输机密包

为了匹配 Radio 的时序要求，KSGEN 任务必须在 Radio 的 START 任务之前触发，ENDKSGEN 事件和 CRYPT 任务的快捷方式必须使能。如图 3-97 图 3_97 中用 PPI 将 Radio 中的 READY 事件和 AES-CCM 中的 KSGEN 任务进行连接。

2.6.7.2.7、解密包在 RADIO 中接收模式

当 AES-CCM 解密 RADIO 接收到的空中加密包时，AES-CCM 必须从 RADIO 接收的内存地址读取加密包。所以 AES-CCM 中 INPUT 指针和 RADIO 中的 PACKETPTR 指针指向的 RAM 中的同一地址。如图 3-98 所示。

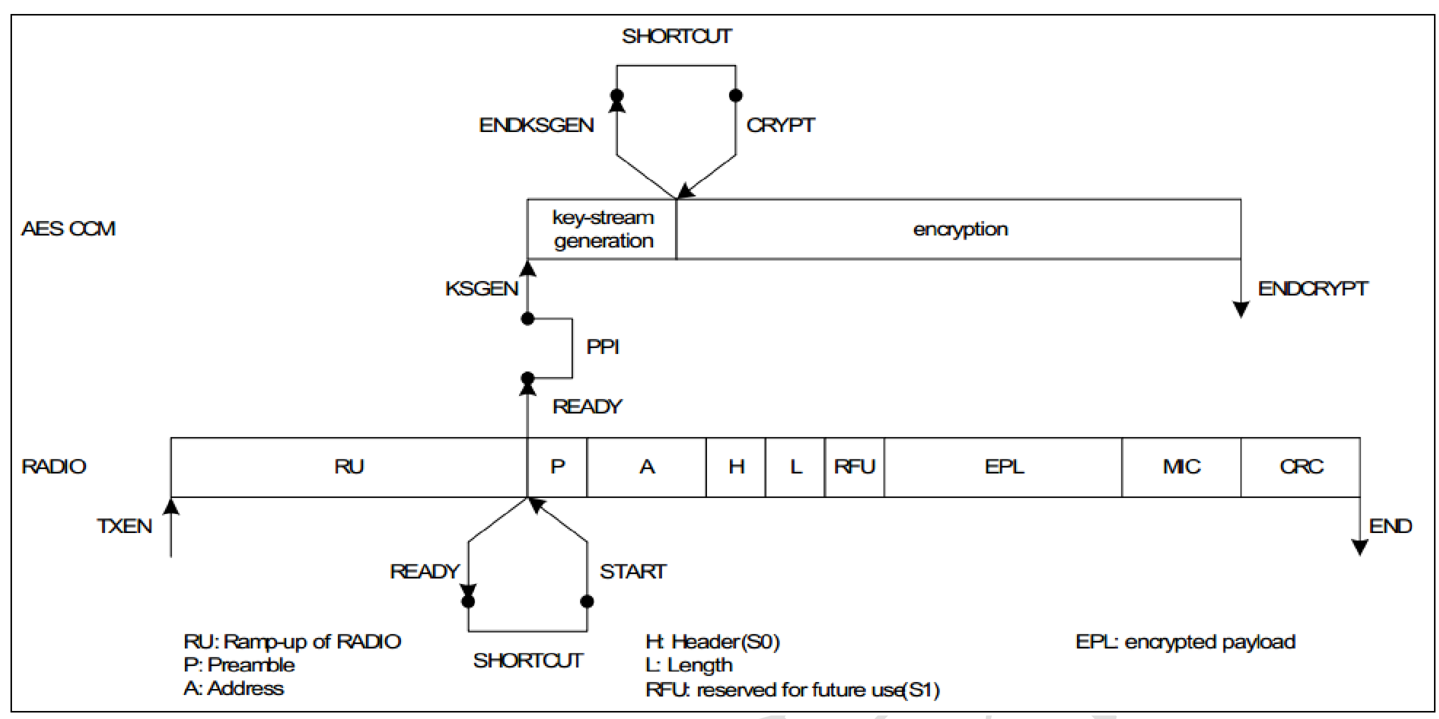


图 3-97 PPI 在 AES-CCM 和 RADIO 加密时使用

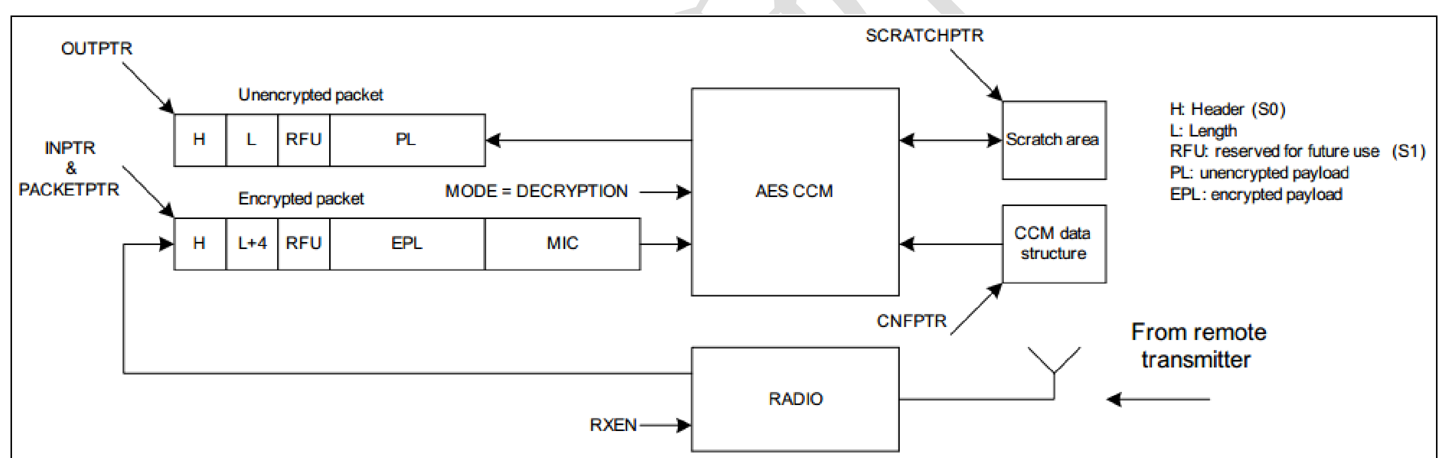


图 3-98 AES-CCM 和 RADIO 组合接收加密包

为了匹配 Radio 的时序要求，AES-CCM 的 KSGEN 任务必须在 Radio 的 START 任务之前触发，CRYPT 任务必须在 Radio 的 ADDRESS 事件之后触发。

如果 CRYPT 任务和 ADDRESS 事件恰好是在同一时间产生，那么 AES-CCM 必须在 Radio 的 END 事件之前完成数据包解密。图 3-99 中通过 PPI 将 Radio 的 ADDRESS 事件和 AES-CCM 的 KSGEN 任务连接。

而 AES-CCM 的 KSGEN 任务通过 Radio 的 READY 事件用 PPI 连接触发。

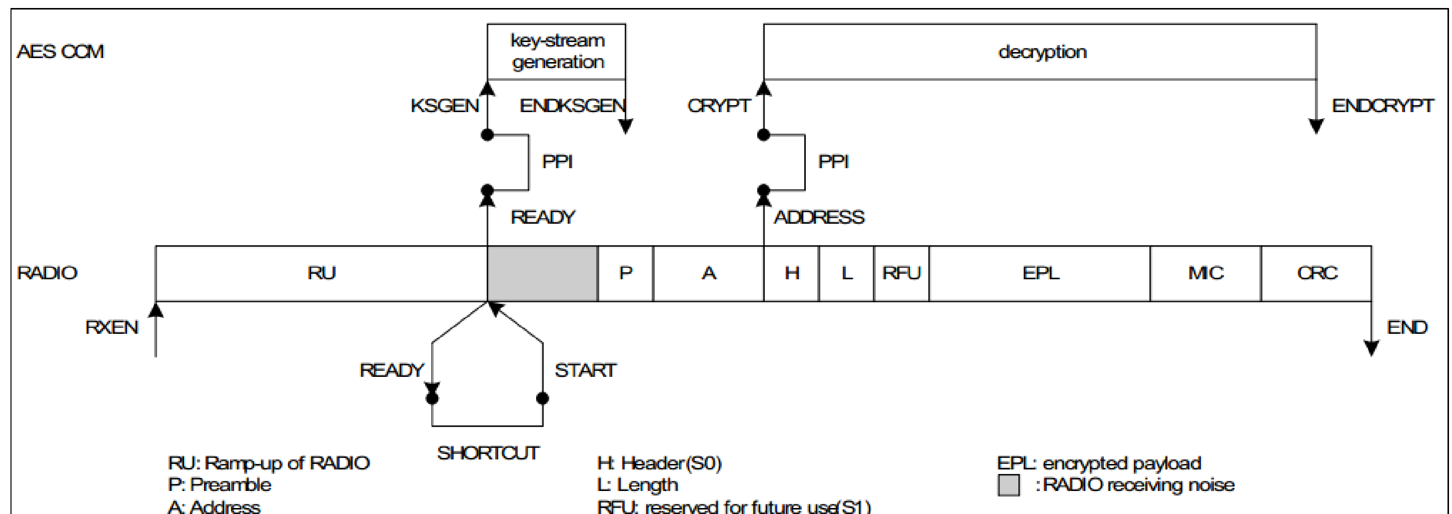


图 3-99 PPI 在 AES-CCM 和 RADIO 中解密时使用

3.6.7.3、快速地址解析模块(Accelerated Address Resolver (AAR))

对于可解析设备地址以及地址解析原理见 3.6.3.3.2 节中的设备地址类型。

AAR 解析一个地址时它的 ADDRPTR 指针必须指向地址之前 3 个字节的地址偏移量。解析开始时触发 START 任务，如果在 IRK 的集合中找到了能解析这个地址的 IRK，那么将产生 RESOLVED 事件。在 nrf51822 中最多可以存储 16 个 IRK 密钥(IRK0 到 IRK15)，AAR 将从 IRK0 开始使用。具体存储了多少个 IRK 用 AAR 模块中的 NIRK 寄存器记录。当地址不能被解析时，将产生 NOTRESOLVED 事件。如果解析完毕将产生 END 事件。

注意，AAR 虽然具有解析地址的能力，但是它不能区分地址是公共地址还是随机地址。也不能区分随机地址中的静态地址或私有地址，也不能区分私用地址中的可解析地址和不可解析地址。也就是它只能

做解析的工作，不能判断地址是否具备解析的属性。AAR 工作如图3-100 所示。图 3-101 为 AAR 和 RADIO 配合工作。

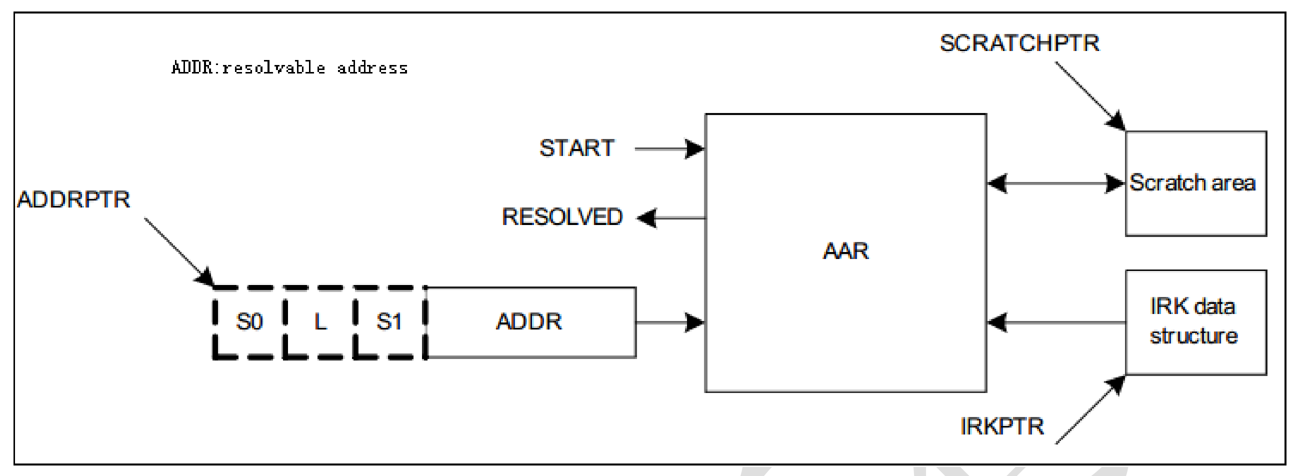


图 3-100 AAR 地址解析

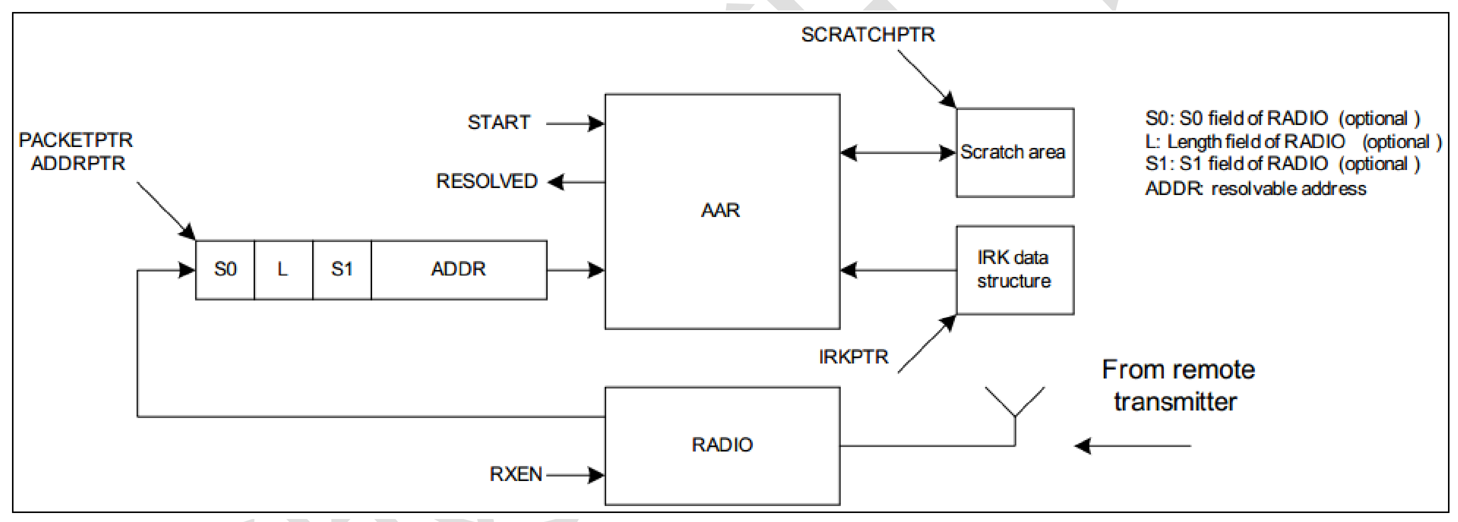


图 3-101 AAR 和 RADIO 配合工作

3.6.8、安全管理空中数据计算和分析

这一节通过采集空中的数据包进行分析和计算，计算确认值、短期密钥 STK 和会话密钥 SK 等。

3.6.8.1 确认值计算

具体采集数据和计算如下：

c1 (k, r, preq, pres, iat, rat, ia, ra) = e(k, e(k, r XOR p1) XOR p2)

p1 = pres || preq || rat' || iat'

p2 = padding || ia || ra

k is 128 bits 为短期秘钥，短期秘钥 3 种方式确定，当加密为仅仅工作方式时 TK 为 0，当为可输入密码时，为输入的 6 位数，再者就是 OOB 确认。

r: 128 bits　　即为 random　　为主机或者从机发送的随机数

pres: 56 bits　即为 pariring Reqest command

preq: 6 bits　　即为 pariring Response command　　如下图共 7 个字节的值

LSB	octet 0	octet 1	octet 2	octet 3	MSB
	Code=0x01	IO Capability	OOB data flag	AuthReq	
	Maximum Encryption Key Size	Initiator Key Distribution	Responder Key Distribution		

Iat:1 bit　即为 initiating device address type　主机的设备地址类型 iat'为 0x00 或者 0x01

ia: 48 bits 即为 initiating device address　　主机的设备地址

rat:1 bit　即为 responding device address type　从机的设备地址类型 rat' 为 0x00 或者 0x01

ra:48 bits　即为 responding device address　　从机的设备地址

padding:32 bits or 0　　这个是一个衬底　　为了满足 128bits 补 0 用的

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Request (0x01)
  IO Capability: Keyboard, Display (0x04)
  OOB Data Flags: OOB Auth. Data Not Present (0x00)
⊟ AuthReq: Bonding, MITM   preq=0x03031005000401
    .... ..01 = Bonding Flags: Bonding (0x01)
    .... .1.. = MITM Flag: 1
  Max Encryption Key Size: 16
⊟ Initiator Key Distribution: LTK IRK
    .... ...1 = Encryption Key (LTK): 1
    .... ..1. = Id Key (IRK): 1
    .... .0.. = Signature Key (CSRK): 0
⊟ Responder Key Distribution: LTK IRK
    .... ...1 = Encryption Key (LTK): 1
    .... ..1. = Id Key (IRK): 1
    .... .0.. = Signature Key (CSRK): 0

)00  04 06 1e 01 22 0e 06 0a  03 1b 2c 18 00 b1 73 00
)10  00 59 ab 9a af 0e 0b 07  00 06 00 01 04 00 05 10
)20  03 03 6d 0d 0c
```

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Response (0x02)
  IO Capability: No Input, No Output (0x03)
  OOB Data Flags: OOB Auth. Data Not Present (0x00)
⊟ AuthReq: Bonding, No MITM  pres=0x01021001000302
    .... ..01 = Bonding Flags: Bonding (0x01)
    .... .0.. = MITM Flag: 0
  Max Encryption Key Size: 16
⊟ Initiator Key Distribution: IRK
    .... ...0 = Encryption Key (LTK): 0
    .... ..1. = Id Key (IRK): 1
    .... .0.. = Signature Key (CSRK): 0
⊟ Responder Key Distribution: LTK
    .... ...1 = Encryption Key (LTK): 1
    .... ..0. = Id Key (IRK): 0
    .... .0.. = Signature Key (CSRK): 0

00  04 06 1e 01 25 0e 06 0a  01 22 39 19 00 98 00 00
10  00 59 ab 9a af 06 0b 07  00 06 00 02 03 00 01 10
20  02 01 af af 3d
```

```
Bluetooth Low Energy
  Access Address: 0x8e89bed6        rat'=0x01
⊟ Packet Header
    1... .... = RX Address: random      iat'=0x01
    .1.. .... = TX Address: random
    .... 0101 = TYPE: CONNECT_REQ (0x05)   ia=0x496ee4367636
    Length: 34                             ra=0xfa3a3c20d682
  Init Address: 49:6e:e4:36:76:36 (49:6e:e4:36:76:36)
  Advertising Address: fa:3a:3c:20:d6:82 (fa:3a:3c:20:d6:82)
```

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Confirm (0x03)    主机->从机 确认值
  Confirm Value: 087bfe422b6d3ea615df0e11df9768a4
```

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Confirm (0x03)   从机->主机确认值
  Confirm Value: 8a9521aeb80fd83ee4c1a8391e8052a4
```

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Random (0x04)   主机->从机随机数
  Random Value: b4c0b3c911d1faae8c42a11f4836256b
```

主机发给从机的随机数为

0x 6b 25 36 48 1f a1 42 8c ae fa d1 11 c9 b3 c0 b4

```
Bluetooth Security Manager Protocol
  Opcode: Pairing Random (0x04)  从机->主机随机数
  Random Value: e8c13c573ffe2ed0b809827 2e3cf06c9
```

从机发给主机的随机数为

0x c9 06 cf e3 72 82 09 b8 d0 2e fe 3f 57 3c c1 e8

也就是对于

c1 (k, r, preq, pres, iat, rat, ia, ra) = e(k, e(k, r XOR p1) XOR p2)中的

k :

0x00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

random :

主机发给从机的随机数为

0x 6b 25 36 48 1f a1 42 8c ae fa d1 11 c9 b3 c0 b4

从机发给主机的随机数为

0x c9 06 cf e3 72 82 09 b8 d0 2e fe 3f 57 3c c1 e8

preq :

0x03 03 10 05 00 04 01

pres:

0x01 02 10 01 00 03 02

iat’:

0x01

ia:

0x49 6e e4 36 76 36

rat’:

0x01

ra:

0xfa 3a 3c 20 d6 82

所以

p1 = pres || preq || rat’ || iat’

=0x01 02 10 01 00 03 02 03 03 10 05 00 04 01 01 01

p2 = padding || ia || ra

= 0x00 00 00 00 49 6e e4 36 76 36 fa 3a 3c 20 d6 82

通过程序先进行主机的确认值的计算:

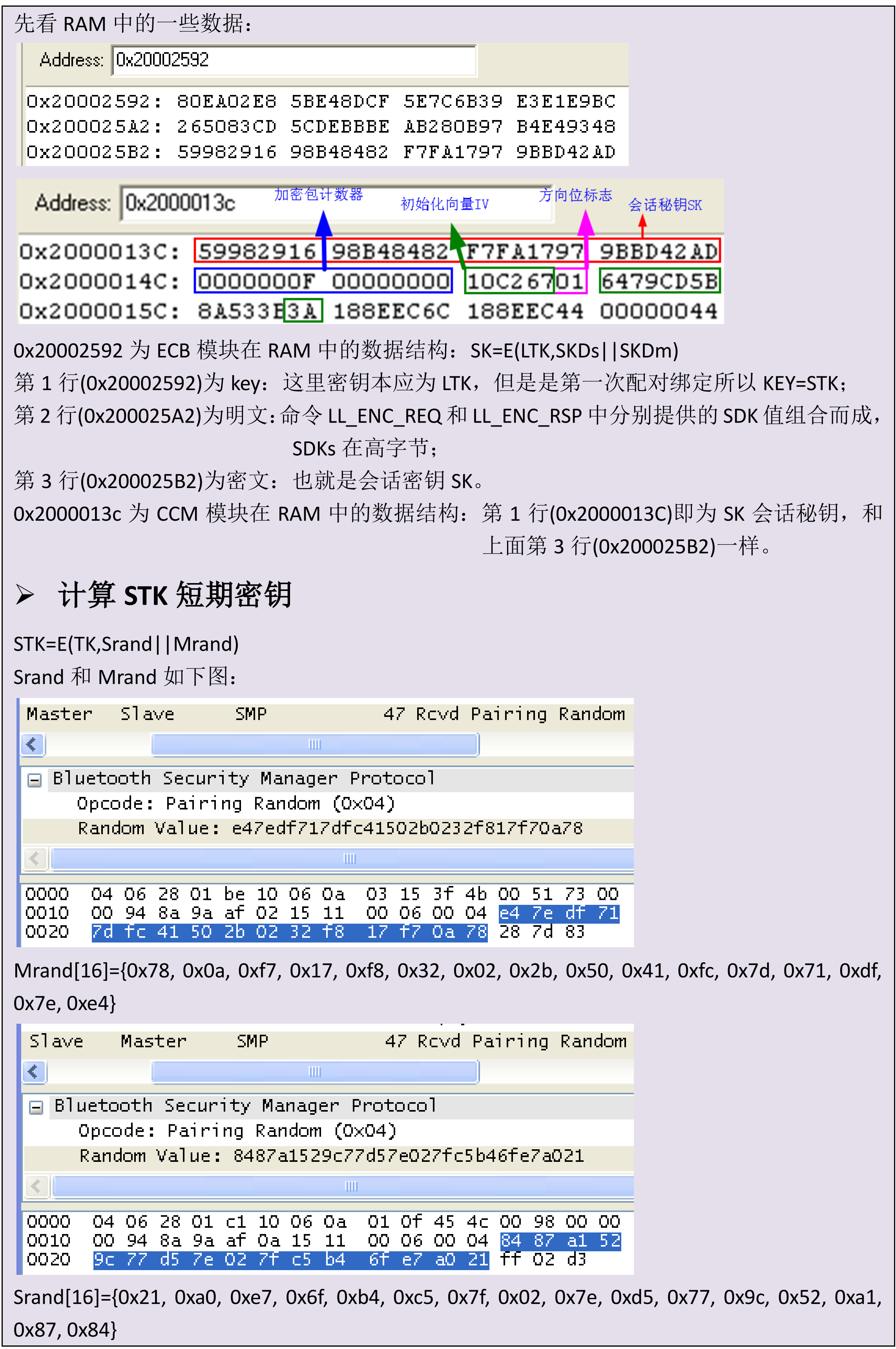
```
const uint8_t key[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t R1[16]={0x6b,0x25,0x36,0x48,0x1f,0xa1,0x42,0x8c,0xae,0xfa,0xd1,0x11,0xc9,0xb3,0xc0,0xb4};
uint8_t p1[16]={0x01,0x02,0x10,0x01,0x00,0x03,0x02,0x03,0x03,0x10,0x05,0x00,0x04,0x01,0x01,0x01};
uint8_t p2[16]={0x00,0x00,0x00,0x00,0x49,0x6e,0xe4,0x36,0x76,0x36,0xfa,0x3a,0x3c,0x20,0xd6,0x82};
uint8_t  dest_bur[16];
int main(void)
{
    uint8_t i=0;
    system_clk_16M();
    nrf_ecb_init();
    nrf_ecb_set_key(key);
    for(i=0;i<16;i++)
    {
      p1[i]^=R1[i];
    }

  while(nrf_ecb_crypt(dest_bur,p1)==false);

    for(i=0;i<16;i++)
    {
      p2[i]^=dest_bur[i];
    }

 while(nrf_ecb_crypt(dest_bur,p2)==false);

 while(1)
    {

    }
}
```

注意：最高字节 是放在数组的第0个元素
最低字节 是放在数组的最后一个元素

运行结果如下：

dest_bur 0x20000070
[0] 0xA4 '?
[1] 0x68 'h'
[2] 0x97 '?
[3] 0xDF '?
[4] 0x11
[5] 0x0E
[6] 0xDF '?
[7] 0x15
[8] 0xA6 '?
[9] 0x3E '>'
[10] 0x6D 'm'
[11] 0x2B '+'
[12] 0x42 'B'
[13] 0xFE '?
[14] 0x7B '{'
[15] 0x08
MSB
LSB
Bluetooth Security Manager Protocol
Opcode: Pairing Confirm (0x03)
Confirm Value: 087bfe422b6d3ea615df0e11df9768a4
主机->从机 确认值
dest_bur 0x20000070
[0] 0xA4 '?
[1] 0x52 'R'
[2] 0x80 '€'
[3] 0x1E
[4] 0x39 '9'
[5] 0xA8 '?
[6] 0xC1 '?
[7] 0xE4 '?
[8] 0x3E '>'
[9] 0xD8 '?
[10] 0x0F
[11] 0xB8 '?
[12] 0xAE '?
[13] 0x21 '!'
[14] 0x95 '?
[15] 0x8A '?
Bluetooth Security Manager Protocol
Opcode: Pairing Confirm (0x03)
Confirm Value: 8a9521aeb80fd83ee4c1a8391e8052a4
从机->主机确认值

3.6.8.2、第 1 次连接加密----配对绑定 STK 和 SK 计算



先看 RAM 中的一些数据：

0x20002592 为 ECB 模块在 RAM 中的数据结构：SK=E(LTK,SKDs||SKDm)

第 1 行(0x20002592)为 key：这里密钥本应为 LTK，但是是第一次配对绑定所以 KEY=STK；

第 2 行(0x200025A2)为明文：命令 LL_ENC_REQ 和 LL_ENC_RSP 中分别提供的 SDK 值组合而成，SDKs 在高字节；

第 3 行(0x200025B2)为密文：也就是会话密钥 SK。

0x2000013c 为 CCM 模块在 RAM 中的数据结构：第 1 行(0x2000013C)即为 SK 会话秘钥，和上面第 3 行(0x200025B2)一样。

➢ 计算 STK 短期密钥

STK=E(TK,Srand||Mrand)

Srand 和 Mrand 如下图：

Mrand[16]={0x78, 0x0a, 0xf7, 0x17, 0xf8, 0x32, 0x02, 0x2b, 0x50, 0x41, 0xfc, 0x7d, 0x71, 0xdf, 0x7e, 0xe4}

Srand[16]={0x21, 0xa0, 0xe7, 0x6f, 0xb4, 0xc5, 0x7f, 0x02, 0x7e, 0xd5, 0x77, 0x9c, 0x52, 0xa1, 0x87, 0x84}

所以 SMrand[16]={0x7e, 0xd5, 0x77, 0x9c, 0x52, 0xa1, 0x87, 0x84, 0x50, 0x41, 0xfc, 0x7d, 0x71, 0xdf, 0x7e, 0xe4}

计算结果如下：

SMrand	0x20000000
[0]	0x7E '~'
[1]	0xD5 '?'
[2]	0x77 'w'
[3]	0x9C '?'
[4]	0x52 'R'
[5]	0xA1 '?'
[6]	0x87 '?'
[7]	0x84 '?'
[8]	0x50 'P'
[9]	0x41 'A'
[10]	0xFC '?'
[11]	0x7D '}'
[12]	0x71 'q'
[13]	0xDF '?'
[14]	0x7E '~'
[15]	0xE4 '?'

STK	0x20000094
[0]	0xE8 '?'
[1]	0x02 '┐'
[2]	0xEA '?'
[3]	0x80 '€'
[4]	0xCF '?'
[5]	0x8D '?'
[6]	0xE4 '?'
[7]	0x5B '['
[8]	0x39 '9'
[9]	0x6B 'k'
[10]	0x7C '|'
[11]	0x5E '^'
[12]	0xBC '?'
[13]	0xE9 '?'
[14]	0xE1 '?'
[15]	0xE3 '?'

和 ECB 模块 0x20002592 地址的数据结构中的第 1 行的结果一样。这里注意 RAM 调试窗口中地址是左边高地址右边低地址。ECB 模块所有数据都是高字节数据放低地址，低字节数据放高地址。

所以 STK=0XE802EA80 CF8DE45B 396B7C5E BCE9E1E3 在 RAM 中调试窗口存放应为：

高地址(低字节)—低地址(高字节): 80EA02E8

高地址(低字节)—低地址(高字节): 5BE48DCF

高地址(低字节)—低地址(高字节): 5E7C6B39

高地址(低字节)—低地址(高字节): E3E1E9BC

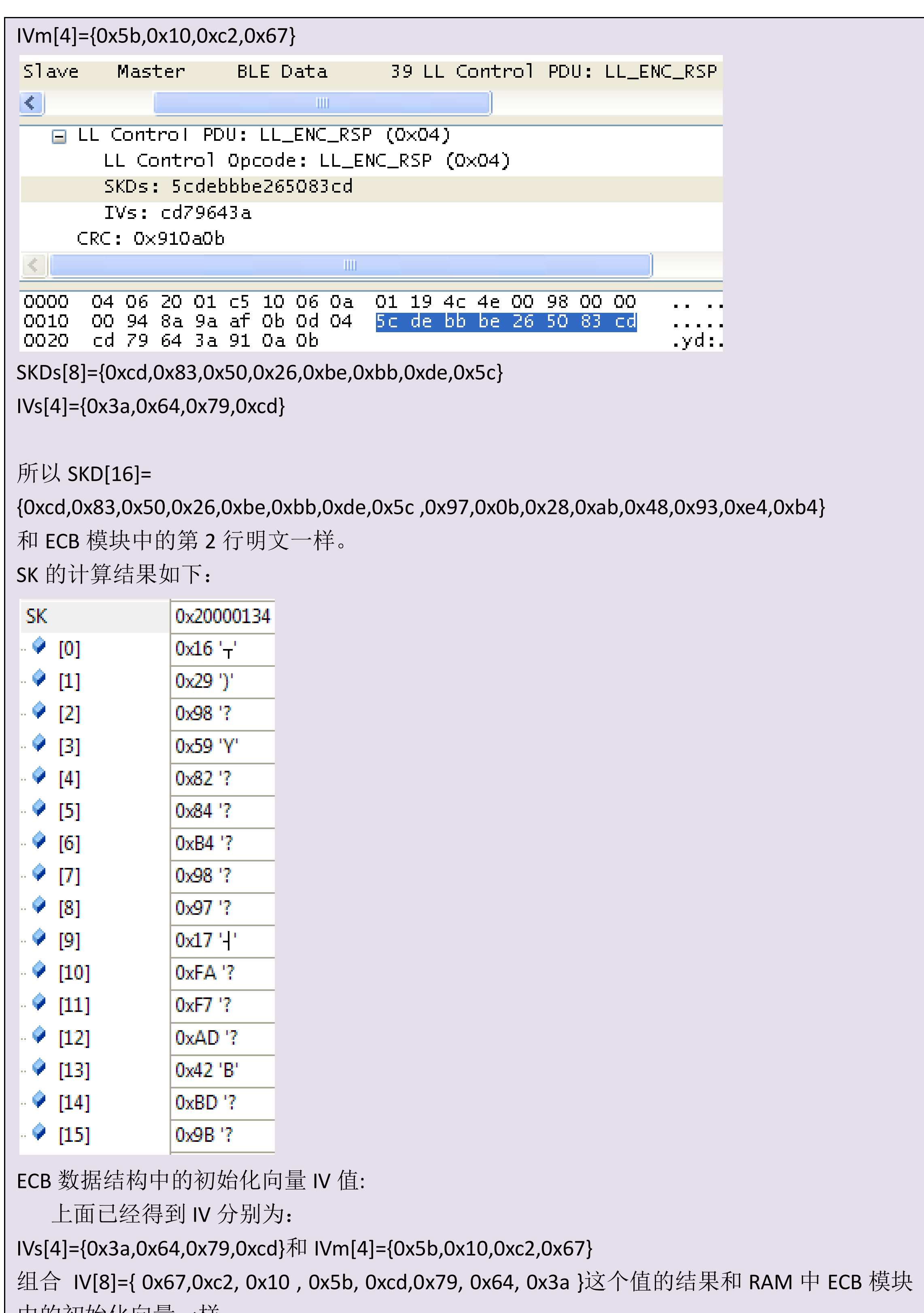
➢ 计算 SK 会话密钥

SK=E(LTK,SKDs||SKDm)，这里 LTK=STK，STK 上面已经算出。

SKDs 和 SKDm 如下图：

```
Master   Slave      BLE Data      49 LL Control PDU: LL_ENC_REQ

  ⊟ LL Control PDU: LL_ENC_REQ (0x03)
       LL Control Opcode: LL_ENC_REQ (0x03)
       Rand: 0000000000000000
       EDIV: 0000
       SKDm: b4e49348ab280b97
       IVm: 67c2105b

0000  04 06 2a 01 c2 10 06 0a  03 14 3f 4d 00 50 73 00   ..*..
0010  00 94 8a 9a af 03 17 03  00 00 00 00 00 00 00 00   .....
0020  00 00 b4 e4 93 48 ab 28  0b 97 67 c2 10 5b 40 25   .....
0030  2d                                                 -
```

SKDm[8]={0x97,0x0b,0x28,0xab,0x48,0x93,0xe4,0xb4}



IVm[4]={0x5b,0x10,0xc2,0x67}

SKDs[8]={0xcd,0x83,0x50,0x26,0xbe,0xbb,0xde,0x5c}

IVs[4]={0x3a,0x64,0x79,0xcd}

所以 SKD[16]=
{0xcd,0x83,0x50,0x26,0xbe,0xbb,0xde,0x5c ,0x97,0x0b,0x28,0xab,0x48,0x93,0xe4,0xb4}
和 ECB 模块中的第 2 行明文一样。
SK 的计算结果如下：

SK	0x20000134
[0]	0x16 '┬'
[1]	0x29 ')'
[2]	0x98 '?'
[3]	0x59 'Y'
[4]	0x82 '?'
[5]	0x84 '?'
[6]	0xB4 '?'
[7]	0x98 '?'
[8]	0x97 '?'
[9]	0x17 '┤'
[10]	0xFA '?'
[11]	0xF7 '?'
[12]	0xAD '?'
[13]	0x42 'B'
[14]	0xBD '?'
[15]	0x9B '?'

ECB 数据结构中的初始化向量 IV 值:

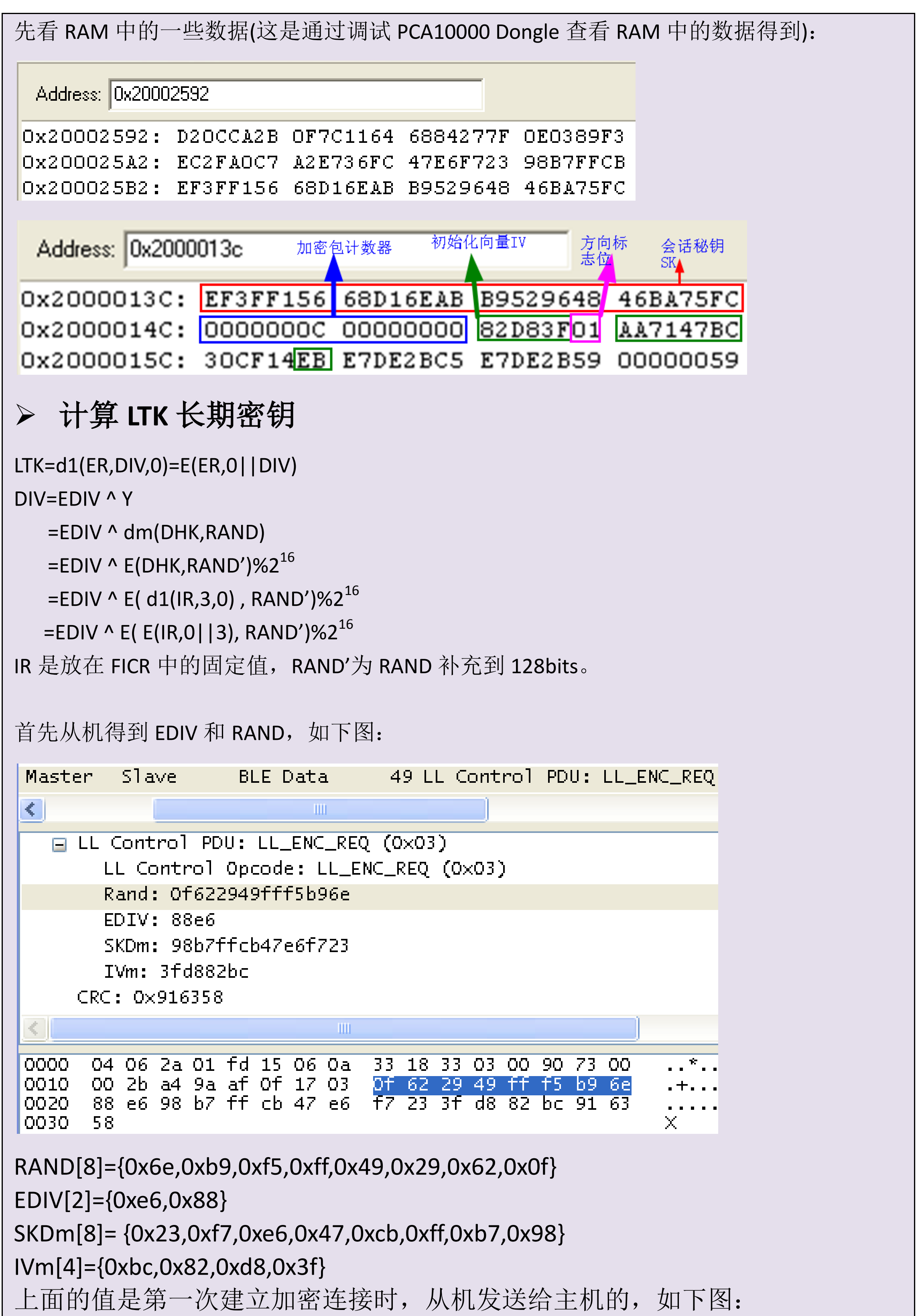
上面已经得到 IV 分别为：

IVs[4]={0x3a,0x64,0x79,0xcd}和 IVm[4]={0x5b,0x10,0xc2,0x67}

组合 IV[8]={ 0x67,0xc2, 0x10 , 0x5b, 0xcd,0x79, 0x64, 0x3a }这个值的结果和 RAM 中 ECB 模块中的初始化向量一样。

注意：只有用到 ECB 模块的才采用大端模式数据存储，CCM 模块中的数据结构中，除了 KEY 采用大端模式(因为这个秘钥是要用到 ECB 中去的)，其他加密计数器，方向标志，初始化向量都是采用小端模式，也就是低字节放低地址。

3.6.8.3、第 2 次连接加密----LTK 和 SK 计算



先看 RAM 中的一些数据(这是通过调试 PCA10000 Dongle 查看 RAM 中的数据得到):

➢ **计算 LTK 长期密钥**

LTK=d1(ER,DIV,0)=E(ER,0||DIV)

DIV=EDIV ^ Y

=EDIV ^ dm(DHK,RAND)

=EDIV ^ E(DHK,RAND')%2^{16}

=EDIV ^ E(d1(IR,3,0) , RAND')%2^{16}

=EDIV ^ E(E(IR,0||3), RAND')%2^{16}

IR 是放在 FICR 中的固定值，RAND'为 RAND 补充到 128bits。

首先从机得到 EDIV 和 RAND，如下图：

RAND[8]={0x6e,0xb9,0xf5,0xff,0x49,0x29,0x62,0x0f}

EDIV[2]={0xe6,0x88}

SKDm[8]= {0x23,0xf7,0xe6,0x47,0xcb,0xff,0xb7,0x98}

IVm[4]={0xbc,0x82,0xd8,0x3f}

上面的值是第一次建立加密连接时，从机发送给主机的，如下图：

```
Slave    Master    SMP         41 Rcvd Master Identification

Bluetooth Security Manager Protocol
  Opcode: Master Identification (0x07)
  Encrypted Diversifier (EDIV): 0xe688
  Random Value: 0f622949fff5b96e

00   04 06 22 01 d3 10 06 0a  3d 17 4c 55 00 b8 00 00   .."......
10   00 94 8a 9a af 06 0f 0b  00 06 00 07 88 e6 0f 62   ........
20   29 49 ff f5 b9 6e b3 cb  6e                        )I...n..
```

读取 FICR 可以得到 IR 和 ER，这里注意 IR 和 ER 存放的方式----小端模式。如下：

ER	
[0]	0x2927C47B
[1]	0x2A1682BB
[2]	0xCD6472AA
[3]	0x24C385D8
IR	
[0]	0x2CC5907B
[1]	0x2CE9D430
[2]	0x1D3C1570
[3]	0xD5624D44

```
Address: 0x10000080
0x10000080: 2927C47B 2A1682BB CD6472AA 24C385D8
0x10000090: 2CC5907B 2CE9D430 1D3C1570 D5624D44
```

```
ER[0]= *(uint32_t*)0x10000080;
ER[1]= *(uint32_t*)0x10000084;
ER[2]= *(uint32_t*)0x10000088;
ER[3]= *(uint32_t*)0x1000008c;

ER_key[0]=(uint8_t )(ER[3]>>24);
ER_key[1]=(uint8_t )(ER[3]>>16);
ER_key[2]=(uint8_t )(ER[3]>>8);
ER_key[3]=(uint8_t )(ER[3]>>0);
ER_key[4]=(uint8_t )(ER[2]>>24);
ER_key[5]=(uint8_t )(ER[2]>>16);
ER_key[6]=(uint8_t )(ER[2]>>8);
ER_key[7]=(uint8_t )(ER[2]>>0);
ER_key[8]=(uint8_t )(ER[1]>>24);
ER_key[9]=(uint8_t )(ER[1]>>16);
ER_key[10]=(uint8_t )(ER[1]>>8);
ER_key[11]=(uint8_t )(ER[1]>>0);
ER_key[12]=(uint8_t )(ER[0]>>24);
ER_key[13]=(uint8_t )(ER[0]>>16);
ER_key[14]=(uint8_t )(ER[0]>>8);
ER_key[15]=(uint8_t )(ER[0]>>0);
```

ER_key	0x200000D4
[0]	0x24 '$'
[1]	0xC3 '?'
[2]	0x85 '?'
[3]	0xD8 '?'
[4]	0xCD '?'
[5]	0x64 'd'
[6]	0x72 'r'
[7]	0xAA '?'
[8]	0x2A '*'
[9]	0x16 '┬'
[10]	0x82 '?'
[11]	0xBB '?'
[12]	0x29 ')'
[13]	0x27 '''
[14]	0xC4 '?'
[15]	0x7B '{'

```
IR[0]= *(uint32_t*)0x10000090;
IR[1]= *(uint32_t*)0x10000094;
IR[2]= *(uint32_t*)0x10000098;
IR[3]= *(uint32_t*)0x1000009c;

IR_key[0]=(uint8_t )(IR[3]>>24);
IR_key[1]=(uint8_t )(IR[3]>>16);
IR_key[2]=(uint8_t )(IR[3]>>8);
IR_key[3]=(uint8_t )(IR[3]>>0);
IR_key[4]=(uint8_t )(IR[2]>>24);
IR_key[5]=(uint8_t )(IR[2]>>16);
IR_key[6]=(uint8_t )(IR[2]>>8);
IR_key[7]=(uint8_t )(IR[2]>>0);
IR_key[8]=(uint8_t )(IR[1]>>24);
IR_key[9]=(uint8_t )(IR[1]>>16);
IR_key[10]=(uint8_t )(IR[1]>>8);
IR_key[11]=(uint8_t )(IR[1]>>0);
IR_key[12]=(uint8_t )(IR[0]>>24);
IR_key[13]=(uint8_t )(IR[0]>>16);
IR_key[14]=(uint8_t )(IR[0]>>8);
IR_key[15]=(uint8_t )(IR[0]>>0);
```

IR_key	0x200000B4
[0]	0xD5 '?'
[1]	0x62 'b'
[2]	0x4D 'M'
[3]	0x44 'D'
[4]	0x1D ''
[5]	0x3C '<'
[6]	0x15 '⊥'
[7]	0x70 'p'
[8]	0x2C ','
[9]	0xE9 '?'
[10]	0xD4 '?'
[11]	0x30 '0'
[12]	0x2C ','
[13]	0xC5 '?'
[14]	0x90 '?'
[15]	0x7B '{'

ER[16]={ 0x24, 0xc3 , 0x85,0xd8, 0xcd, 0x64, 0x72, 0xaa, 0x2a, 0x16, 0x82, 0xbb, 0x29, 0x27, 0xc4,0x7b }

IR[16]= {0xd5, 0x62, 0x4d, 0x44, 0x1d, 0x3c, 0x15, 0x70, 0x2c, 0xe9, 0xd4, 0x30, 0x2c ,0xc5, 0x90, 0x7b }

所以计算 DHK= E(IR,0||3),Y 和 DIV 为：其中 Y 取最低 2 字节：

DHK	0x200000E4
[0]	0xAD '?'
[1]	0x23 '#'
[2]	0x6A 'j'
[3]	0x08 '◘'
[4]	0x21 '!'
[5]	0xA6 '?'
[6]	0x9C '?'
[7]	0x12 '↕'
[8]	0x29 ')'
[9]	0xC4 '?'
[10]	0x86 '?'
[11]	0x1C ''
[12]	0x19 '├'
[13]	0xC5 '?'
[14]	0x94 '?'
[15]	0xA5 '?'

Y	0x20000104
[0]	0x00 '
[1]	0x00 '
[2]	0x00 '
[3]	0x00 '
[4]	0x00 '
[5]	0x00 '
[6]	0x00 '
[7]	0x00 '
[8]	0x00 '
[9]	0x00 '
[10]	0x00 '
[11]	0x00 '
[12]	0x00 '
[13]	0x00 '
[14]	0xEF '?
[15]	0x01 '┌'

DIV	0x20000114
[0]	0x00 '
[1]	0x00 '
[2]	0x00 '
[3]	0x00 '
[4]	0x00 '
[5]	0x00 '
[6]	0x00 '
[7]	0x00 '
[8]	0x00 '
[9]	0x00 '
[10]	0x00 '
[11]	0x00 '
[12]	0x00 '
[13]	0x00 '
[14]	0x09 ''
[15]	0x89 '?

计算 LTK=d1(ER,DIV,0)=E(ER,0||DIV)

LTK	0x20000124
[0]	0x2B '+'
[1]	0xCA '?'
[2]	0x0C '♀'
[3]	0xD2 '?'
[4]	0x64 'd'
[5]	0x11 '◄'
[6]	0x7C '|'
[7]	0x0F '☼'
[8]	0x7F ' '
[9]	0x27 '''
[10]	0x84 '?'
[11]	0x68 'h'
[12]	0xF3 '?'
[13]	0x89 '?'
[14]	0x03 '└'
[15]	0x0E '♫'

```
Slave    Master    SMP         47 Rcvd Encryption Information

Bluetooth Security Manager Protocol
  Opcode: Encryption Information (0x06)
  Long Term Key: 0e0389f36884277f0f7c1164d20cca2b

00   04 06 28 01 cf 10 06 0a  3d 0d 43 53 00 b8 00 00   ..(.....
10   00 94 8a 9a af 06 15 11  00 06 00 06 0e 03 89 f3   ........
20   68 84 27 7f 0f 7c 11 64  d2 0c ca 2b a7 6d 58      h.'..|.d
```

计算的结果和第一次连接是从机发送给主机的长期秘钥一样

➢ 计算 SK 会话密钥

SK=E(LTK,SKDs||SKDm)

上面已经得到了 SKDm。

SKDm[8]= {0x23,0xf7,0xe6,0x47,0xcb,0xff,0xb7,0x98}

SKDs 如下图：

```
Slave    Master    BLE Data    39 LL Control PDU: LL_ENC_RSP

⊟ LL Control PDU: LL_ENC_RSP (0x04)
    LL Control Opcode: LL_ENC_RSP (0x04)
    SKDs: a2e736fcec2fa0c7
    IVs: 4771aaeb
  CRC: 0x261558

00   04 06 20 01 00 16 06 0a  31 1e 4c 04 00 99 00 00   .. .....
10   00 2b a4 9a af 07 0d 04  a2 e7 36 fc ec 2f a0 c7   .+......
20   47 71 aa eb 26 15 58                               Gq..&.X
```

SKDs[8]= {0xc7,0xa0,0x2f,0xec,0xfc,0x36,0xe7,0xa2}

IVs[4]={0xeb,0xaa,0x71,0x47}

所以 SKD[16]=
{0xc7,0xa0,0x2f,0xec,0xfc,0x36,0xe7,0xa2,0x23,0xf7,0xe6,0x47,0xcb,0xff,0xb7,0x98}
计算得到的 SK 会话秘钥如下，其结果和实际的结果一样。

SK	0x20000134
[0]	0x56 'V'
[1]	0xF1 '?
[2]	0x3F '?'
[3]	0xEF '?
[4]	0xAB '?
[5]	0x6E 'n'
[6]	0xD1 '?
[7]	0x68 'h'
[8]	0x48 'H'
[9]	0x96 '?
[10]	0x52 'R'
[11]	0xB9 '?
[12]	0xFC '?
[13]	0x75 'u'
[14]	0xBA '?
[15]	0x46 'F'

ECB 数据结构中的初始化向量 IV 值:
上面已经得到 IV 分别为：
IVm[4]={0xbc,0x82,0xd8,0x3f}和 IVs[4]={0xeb,0xaa,0x71,0x47}
组合 IV[8]={ 0xeb,0xaa,0x71,0x47, 0xbc,0x82,0xd8,0x3f }
这个值的结果和 RAM 中 ECB 模块中的初始化向量一样。

版权