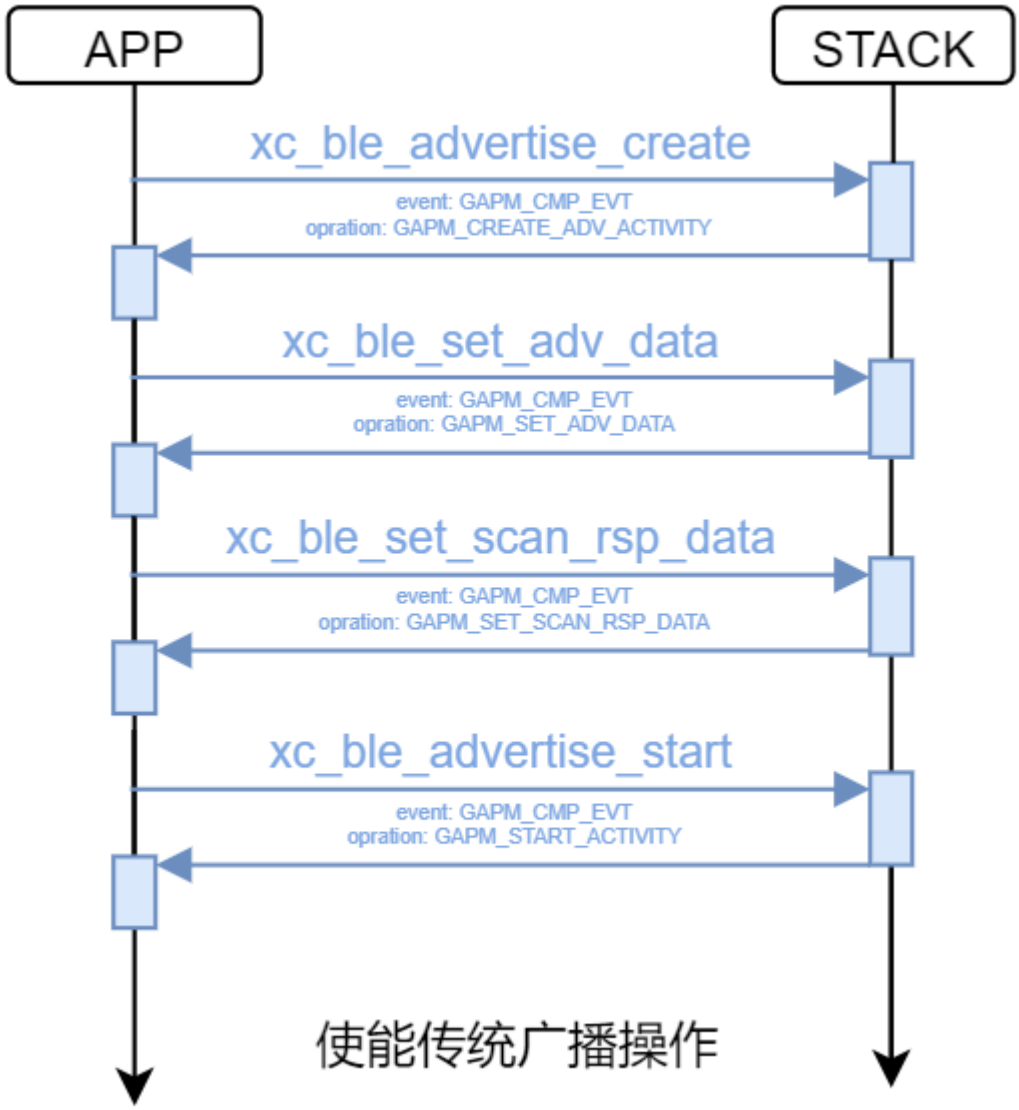
# ble用户手册

## 2.1 广播

### 2.2.1. 传统广播

当设备开启传统广播流程时，应用层与 BLE 协议栈之间的交互流程如下所示：




#### 2.2.1.1. 设置广播参数

比如配置一个可被发现、可被链接、广播间隔为 40ms 的普通广播为例介绍如何设置广播参数：

```
struct gapm_activity_create_adv_cmd param = {0};
param.own_addr_type                         = GAPM_STATIC_ADDR;
param.adv_param.type                        = GAPM_ADV_TYPE_LEGACY;
param.adv_param.disc_mode                   = GAPM_ADV_MODE_GEN_DISC;
param.adv_param.prop                    = GAPM_ADV_PROP_UNDIR_CONN_MASK;
param.adv_param.max_tx_pwr              = APP_MAX_TX_POWER;
```

```
param.adv_param.filter_pol            = ADV_ALLOW_SCAN_ANY_CON_ANY;
param.adv_param.prim_cfg.adv_intv_min = APP_ADV_INT_MIN;
param.adv_param.prim_cfg.adv_intv_max = APP_ADV_INT_MAX;
param.adv_param.prim_cfg.chnl_map     = APP_ADV_CHMAP;
param.adv_param.prim_cfg.phy          = GAP_PHY_1MBPS;

xc_ble_advertise_create(&param);
```

#### 2.2.1.2. 设置广播内容

当广播类型为定向广播时，不需要设置广播数据，其他情况均需要对广播内容进行设置。

传统广播最大数据段为 31 字节，其中 flag 字段由 host 协议栈内部根据广播参数 生成，占据三个字节，因此留给用户配置的数据长度为 28 字节。广播的内容需要严格按照蓝 牙协议规定的 length，type，data 格式进行填写，否则将会设置失败。

比如设置广播内容包含完整的设备名(APP_DFLT_DEVICE_NAME)。

```
uint8_t adv_data[ADV_DATA_MAX_LENGTH] = {0};
adv_data[0]                           = APP_DFLT_DEVICE_NAME_LEN + 1;
adv_data[1]                           = GAP_AD_TYPE_COMPLETE_NAME;
memcpy(&adv_data[2], APP_DFLT_DEVICE_NAME, APP_DFLT_DEVICE_NAME_LEN);
xc_ble_set_adv_data(app_env.adv_actv_idx, APP_DFLT_DEVICE_NAME_LEN + 2, adv_data);
```

在需要更新广播内容时，用户无需暂停当前广播，可以直接调用该函数对内容进行更新。

#### 2.2.1.3. 设置扫描回复内容

扫描回复（scan response）内容为设备在广播过程中收到扫描请求（scan request）时做 出的回复。当广播参数设置为可扫描时，需要设置扫描响应数据，其他情况不需要设 置。在传统广播中，扫描回复内容最长可以配置 31 字节。扫描回复的内容需要严格按照蓝牙 协议规定的 length，type，data 格式进行填写，否则将会设置失败。

比如设置扫描回复的数据为 MANUFACTURER_DATA。

```
uint8_t scan_rsp_data[ADV_DATA_MAX_LENGTH] = {0};
scan_rsp_data[0]                           = MANUFACTURER_DATA_LEN + 1;
scan_rsp_data[1] = GAP_AD_TYPE_MANU_SPECIFIC_DATA;
memcpy(&scan_rsp_data[2], MANUFACTURER_DATA, MANUFACTURER_DATA_LEN);
xc_ble_set_scan_rsp_data(app_env.adv_actv_idx,
                         MANUFACTURER_DATA_LEN + 2, scan_rsp_data);
```

在需要更新扫描回复内容时，用户无需暂停当前广播，可以直接调用该函数对内容进行更 新。

#### 2.2.1.4. 开启广播

调用 gap_start_advertising 函数用于开启广播，输入参数是广播持续时间，单位为 10ms， 设置为 0 时表示广播不会自动停。

比如设置一直持续的广播，duration的值为0，可连接广播在建立连接后会自动停止。

```
struct gapm_activity_start_cmd param = {0};
param.actv_idx                       = app_env.adv_actv_idx;
param.u_param.adv_add_param.duration = ADV_DURATION;
xc_ble_advertise_start(&param);
```

## 3.1 Profile 定义

通用属性规范 GATT（Generic Attribute Profile）用于两个连接设备之间的数据通信。在 BLE GATT 层中，数据以特征（Characteristic）的形式进行传输和存储。

## 3.1.1 GATT 角色

从 GATT 的角度来看，当两个设备处于连接状态时，一个设备作为 GATT 服务端，另一个 设备作为 GATT 客户端。这种角色的分配与 GAP 层设备角色（外围设备、中央设备）无关。 一个外围设备既可以充当 GATT 客户端，也可以充当 GATT 服务端；一个中央设备同样既可 以充当 GATT 客户端，也可以充当 GATT 服务端。

- GATT 客户端：设备发起命令、请求并接收响应、通知和指示。
- GATT 服务端：设备接收命令、请求并发出响应、通知和指示。

## 3.1.2 GATT Profile 层级

一个 profile 中可包含一个或多个服务；一个服务可包含一个或者多个特征；一个特征至 少包含两个属性，包括特征声明和特征值。下面章节将具体描述属性、特征和服务.

## 3.1.2.1. 属性（Attribute）

在低功耗蓝牙中，特征可以看作是一组属性信息的集合，包括特征声明、特征值以及特 征描述符，数据以属性的形式进行交互。属性主要包括以下几个部分：

- 属性句柄：句柄是属性在 GATT 属性表中的索引，每个属性都有唯一的句柄。
- 属性类型：表示数据代表的事物，通常是 Bluetooth SIG 规定或由用户自定义的 UUID （通用唯一标识符）。
- 属性值：属性的数据值。
- 属性权限：规定了 GATT 客户端设备对属性的访问权限，包括是否能访问和怎样访问。

## 3.1.2.2. 特征（Characteristic）

一个典型的特征由以下属性组成：

- 特征声明：描述特征的性质、存储位置（句柄）、类型。
- 特征值：特征的数据值。
- 特征描述符：描述特征的附加信息或配置。

## 3.1.2.3. 服务（Service）

一个 profile 包含一个或多个服务，GATT 服务是一系列特征的集合，例如，心率服务包 括心率测量特征和身体位置特征等。常见的服务有：

- 通用访问服务（Generic Access Service）：该服务包括设备及访问信息等，比如设备 名称、供应商标志、产品标志等。该服务定义的特征有：设备名称（Device Name）、外观（Appearance）、外设优先连接参数（Peripheral Preferred Connection Parameters）等。
- 通用属性服务（Generic Attribute Service）：该服务主要用于服务端通知所连接的客 户端其提供的服务发生了变化。该服务包含 Service changed 特征。 上述两个服务在 BLE 协议栈初始化时，默认添加到服务端的数据库中。

## 3.1.3. 属性表

低功耗蓝牙的通信就是通过对属性进行读写等操作来实现的，本节介绍如果定义这些属 性来构成一个完整的服务。

## 3.1.3.1. 添加自定义服务

每个 Service 必须定义一个属性表并通过 profile 的初始化函数将其传递给协议栈。以一 个 batt 服务的实例来解释属性表的定义。该服务代码在

```
static const gatt_att_desc_t custom_server_atts[] = {
    [CUSTOM_SVC_DECL] =
        {
            .uuid   = 0x00,0x28,
            .info     = GATT_ATT_RD_BIT | ATT_UUID(16),
            .ext_info = 0,
        },
    [CUSTOM_SVC_RX_CHAR_DECL_CHAR] =
        {
            .uuid   = 0x03, 0x28,
            .info     = GATT_ATT_RD_BIT| ATT_UUID(16),
            .ext_info = 0,
        },
    [CUSTOM_SVC_RX_CHAR_VAL] =
        {
            .uuid = CUSTOM_SVC_RX_CHAR_UUID,
            .info   = GATT_ATT_WC_BIT | GATT_ATT_RD_BIT | ATT_UUID(128),
            .ext_info =
                GATT_ATT_NO_OFFSET_BIT | CUSTOM_SVC_RX_CHAR_VAL_MAX_LENGTH ,
        },
    [CUSTOM_SVC_TX_CHAR_DECL_CHAR] =
        {
            .uuid   = 0x03, 0x28,
            .info     = GATT_ATT_RD_BIT| ATT_UUID(16),
            .ext_info = 0,
        },
    [CUSTOM_SVC_TX_CHAR_VAL] =
        {
            .uuid = CUSTOM_SVC_TX_CHAR_UUID,
            .info   =  GATT_ATT_N_BIT | ATT_UUID(128),
            .ext_info =
```

```
                GATT_ATT_NO_OFFSET_BIT | CUSTOM_SVC_TX_CHAR_VAL_MAX_LENGTH,
        },
    [CUSTOM_SVC_TX_CHAR_CFG] =
        {
            .uuid   = 0x02, 0x29,
            .info     = GATT_ATT_RD_BIT | GATT_ATT_WR_BIT | ATT_UUID(16),
            .ext_info = 0,
        },
};
```

TODO: 解释一下是如何进行填充的。

## 3.1.3.2. 建立属性表

当设备上电或者重启时，应用层需在初始化阶段构建自己的服务列表。每个服务都包含 一些属性，并根据自己的功能需求定义服务的回调函数。属性表和回调函数在调用添加服务 函数时传递给协议栈。

例如，在应用层初始化函数中，依次调用 svcA_gatt_add_service 以及 svcB_gatt_add_service 函数添加两个自定义的服务，执行流程如下图所示。 APP BLE Stack 调用 ble_stack_init( ) 初始化协议栈 Profile A Profile B GATT

## 3.1.4. GATT 服务端基础流程

### 3.1.4.1. 服务端

GATT 服务端负责接收对端设备 GATT 客户端发送的命令以及请求，并且根据收到的命 令以及请求回复对端设备响应，或者向对端设备发送通知或指示。GATT 服务端的功能需要 和各个 profile 紧密配合才能够完成。 下面章节将描述如何在 GATT 服务端添加 profile 和注册 profile 回调函数，以及 GATT 服务端如何处理客户端的读/写请求。关于 API 的详细描述，请参考 gatt_api.h 内函数定义说 明。服务端 GATT 基础流程如下图所示。

### 3.1.4.2. 增加服务端 profile

针对服务端 profile，用户需要给出一个属性数组 att_table 和 profile 的回调函数，注册 一个服务端的 profile，便于协议栈上传消息，和侦听特定的属性的消息。以一个服务端 profile 注册的实例来解释该过程。

### 3.1.4.3. 等待客户端发送 notify/indicate 使能

链接建立后，添加了服务端 profile 的设备，在收到客户端发送的 notification/indication enable 的消息后，就可以向客户端发送 ntf/ind 的消息。见如下示例代码：

说明：

- 链接建立后，通过判断 GATT 事件类型为 GATTC_MSG_WRITE_REQ 来获取客户端写 数据的事件处理入口。通过判断写入数据的第一个字节的置位判断 ntf/ind 使能。
- 服务端 profile 对应的 GATT 事件回调处理参见下一节。

### 3.1.4.4. 服务端 profile 回调函数

# 4 安全管理

## 4.1 配对

配对过程，即生成和分发密钥的过程，可以分为 3 个阶段：

1. 交换配对信息。
2. 生成链路加密密钥。
3. 在加密链路上分发其它指定的密钥信息，并根据设备是否支持绑定决定是否需要在安全数据库中存储分发的密钥信息。

配对流程图如下所示，配对方式分成 LE Legacy Pairing 和 LE Secure Connection（从 core 4.2 版本添加进来）两种，二者区别主要在于 step2-生成密钥的过程，另外在该步骤， LE Legacy Pairing 生成 STK， LE Secure Connection 生成 LTK。

LE Secure Connection 中引入了 Elliptic Curve Diffie-Hellman 加密算法，使得安全性得到 了很大的提升。根据设备所支持的安全特性，配对方法可分为如下四种：

- Just Works（Secure Connections or Legacy），适合于配对双方没有输入输出 io 设备（比如：键盘输入或显示器输出）的情况，由于在 Just Works 配对过程中无需 MITM 认证，因此也就无法抵御 MITM 攻击。Just Works 配对既可以用于 Legacy Pairing， 也可以用于 Secure Connection。
- Passkey Entry（Secure Connections or Legacy），配对流程支持 MITM 认证，既可用于 Legacy Pairing，也可以用于 Secure Connection。配对过程中需要用户输入 6 位十进 制数的密码，作为配对过程所需的输入参数。
- Numeric Comparison（Secure Connections），如果双方设备都支持 Secure Connection， IO 能力都具有显示和输入功能，且需带 MITM 认证，则可以使用 Numeric Comparison 配对流程。
- Out of Band（Secure Connections or Legacy），该方式通过 NFC 等带外方式交互配对过 程需要的中间值。目前版本的 SDK 中未支持该配对方式。

### 4.1.1. JustWorks 配对

如果双方设备都无需带中间人（Man-in-the-middle， MITM）认证，那么就可以使用 Just Works 配对流程。由于在 Just Works 配对过程中无需 MITM 认证，因此也就无法抵御 MITM 攻 击。Just Works 配对既可以是 Legacy Pairing 也可以是 Secure Connection，用户只需在配对 流程开始之前配置好安全参数，配对流程启动后，配对过程中无需用户的任何交互。 下面以 Legacy Pairing、Justworks 配对方式为例，介绍参数如何配置：

在上电初始化的时候，字段 pairing_mode 用于设置采用安全配对，传统配对，或者是禁止配对。

```
struct gapm_set_dev_config_cmd cfg_param = {
    .role               = GAP_ROLE_ALL,
    .sugg_max_tx_octets = BLE_MAX_OCTETS,
    // .sugg_max_tx_time   = BLE_MAX_TIME,
    .sugg_max_tx_time   = 2120,
    #if (APP_SEC_CON)
    .pairing_mode = GAPM_PAIRING_SEC_CON ,
    #else // !(APP_SEC_CON)
    .pairing_mode = GAPM_PAIRING_LEGACY ,
    #endif // (APP_SEC_CON)
};
xc_ble_set_dev_config(&cfg_param);
```

在收到主设备的 配对请求后，会触发 函数 gapc_bond_req_ind_handler 中 GAPC_PAIRING_REQ 事件的处理。在这里会配置设备的 io 能力为无输入无输出，是否使用安全配置等。

```
cfm->request = GAPC_PAIRING_RSP;
cfm->accept  = true;
#if (APP_SEC_CON)
cfm->data.pairing_feat.auth      = GAP_AUTH_REQ_SEC_CON_BOND;
#else // !(APP_SEC_CON)
cfm->data.pairing_feat.auth      = GAP_AUTH_REQ_NO_MITM_BOND;
#endif // (APP_SEC_CON)
app_env.sec_con_enabled = true;
cfm->data.pairing_feat.iocap     = GAP_IO_CAP_NO_INPUT_NO_OUTPUT;
cfm->data.pairing_feat.key_size  = 16;
cfm->data.pairing_feat.oob       = GAP_OOB_AUTH_DATA_NOT_PRESENT;
cfm->data.pairing_feat.sec_req   = GAP_SEC1_NOAUTH_PAIR_ENC;
cfm->data.pairing_feat.rkey_dist = GAP_KDIST_ENCKEY | GAP_KDIST_IDKEY;
cfm->data.pairing_feat.ikey_dist = GAP_KDIST_ENCKEY | GAP_KDIST_IDKEY;
ke_msg_send(cfm);
```

#### 4.1.1.2. Passkey Entry 配对

TODO

#### 4.1.1.3. Security Connections 配对

TODO

#### 4.1.1.4. 配对 key 的保存

如果用户程序需要保存绑定信息、对端服务信息等内容，在协议栈运行之前需要先调用 绑定管理初始化函数。

#### 4.1.1.5.从机进行配对

#### 4.1.1.5.1 从机开启配对请求流程的具体步骤如下：

说明 ：

- 在 GAP 事件回调函数内，通过判断事件类型为 GAP_EVT_SLAVE_CONNECT 来获 取从机连接事件处理的入口。在该事件下面调用 gap_security_req 函数请求主机发起 绑定动作。
- 绑定之后会自动对链接进行加密，加密成功后，BLE 协议栈会上传 GAP 事件 GAP_SEC_EVT_SLAVE_ENCRYPT。

2.6.2. 加密 设备进行做配对之后，只要开启了绑定管理功能。可以直接调用加密函数对已建立的链 接进行加密操作。加密操作是主机在链接建立后发起的。

2.6.2.2. 从机响应加密

说明 ：

- 在 GAP 事件回调函数内，通过判断事件类型为 GAP_EVT_SLAVE_CONNECT 来获 取从机连接事件处理的入口。
- 加密成功后，BLE 协议栈会上传 GAP 事件 GAP_SEC_EVT_SLAVE_ENCRYPT。

### 2.6.3. 隐私管理

BLE 中的隐私管理，是指已认证设备可以跟踪识别目标设备，而其他非认证设备无法跟踪 识别目标设备。隐私管理使得已认证的设备可以正常地与目标设备进行连接和通讯，同时防 止其他非认证设备、恶意破坏设备对目标设备的跟踪。

### 2.6.3.1. 开启隐私管理功能

系统初始化时可以设置白名单，可以在广播和主动链接时过滤掉白名单以外的设备。开 启白名单管理功能的具体步骤如下：

1. 主设备与从设备进行连接并绑定，绑定过程中需要交换 IRK 以及身份地址信息，绑 定之后主设备与从设备断开连接。

2. 白名单配置。用户可以在系统初始化过程中，发送广播、发起主动连接之前设置白名 单。

3. 广播设备使用白名单，广播可以通过设置广播参数中的过滤政策来使能白名单过滤功 能。

2.6.3.2. 地址配置说明 当开启广播、扫描、建立连接时，controller 会使用用户设置的本机地址发送空口数据包。 本机地址在初始化时进行设置一次即可。广播、扫描、主动连接时均采用初始化时设置的本 机地址做为设备地址。

说明：

- PUBLIC 类型，表示设备的 mac 地址会一直不变，该类型地址需要向 IEEE 申请。
- PRIVATE 类型地址可以用户自行指定，在一个上电周期类保持不变或者一直不变， 该类型地址的最高 2bit 要设为 11b。
- RANDOM_RESOVABLE 类型地址通过一个随机数和 IRK（identity resolving key） 来生成，采用这种类型地址的设备可以防止被未知设备扫描和追踪。
- RANDOM_NONE_RESOVABLE 类型地址与 PRIVATE 类型地址类似，但是 RANDOM_NONE_RESOVABLE 类型地址会定时更新，更新周期由 GAP 来设定。

2. 设置设备本地地址，在 user_main()初始化时，需要给 mac 地址赋值的同时，指定本 地 mac 地址的类型，本地 mac 地址类型一旦设定后续不可修改，扫描，广播，和主 动发起连接的动作，均使用此时设定的本地地址类型进行操作。