Dialog SDK 5.0.3 培训材料5– 配对，绑定和安全

2016.6

…personal
…portable
…connected

BLE 自定义服务的安全特性

BLE安全特性概览

自定义服务源码讨论

输出结果

BLE 自定义服务的安全特性

我们一起做一个demo…

- **在我们开始前，我们建议你 …**
  - 看一下培训材料2关于自定义服务的应用

- **你将会从这个培训里面学到 …**
  - 基本理解BLE的安全特性和问题
  - 什么是配对？什么是绑定？
  - ‘Just-Works（立即工作）’配对方式
  - 单设备绑定
  - 基本理解多设备绑定
  - 如何在自定义服务数据库里加入配对

- **接下来 …**
  - 看一下这份PPT的参考文档部分

GAP LE 安全

通用接入服务的低功耗安全特性和注意事项

- 在BLE通信中的相关安全特性考虑:

- **Man-in-the-Middle (MITM)**

  MITM模式要求攻击者有能力去监听和改变传往通信链路中的信息。一个例子是主动监听，攻击者会对被攻击设备发起独立主动的连接，并传递消息，使得设备相信自己是在一条私有连接上在进行通信。攻击者可以理解所有往来交互的信息，并有能力加入额外的消息。对应MITM的攻击，保护方式主要是使用万能钥匙的配对方式，或者使用带外配对的方式。

GAP LE 安全

通用接入服务的低功耗安全特性

- 在BLE通信中的相关安全特性考虑:

- **被动监听**

  被动监听是私下接收配对数据，或者没有配对下的普通交互数据；然后在没有得到同意的情况下（用sniffer工具）监听。BLE v4.0针对这种情况没有保护机制（在BLE v4.2的安全连接特性里会采用ECDH的公共密钥来应对这种被动监听攻击—不在此次讨论范围之内）

- **隐私/身份追踪**

  BLE支持隐私特性，会在一段时间内更新蓝牙地址，降低被跟踪的风险。这种修改后的地址叫做私有地址，被信任的设备可以解析地址。

GAP LE 安全

通入接入服务低功耗绑定

- 为了消除MITM和被动监听攻击带来的风险，两个BLE设备间需要进行配对.
- 配对是这样一个过程，两个设备间交互安全和身份信息，创建一种信任关系.
- 这些安全和身份信息就是所谓的绑定信息。当设备开始存储这些绑定信息，意味着绑定已经建立，或者设备间已经被绑定在一起
- 一旦秘钥交互后，每一次的连接结束之后都会存储这些绑定信息。这些信息可以让设备在以后安全通信，或者重新配对而无需再次交互秘钥。BLE设备可以被配置为非绑定模式和绑定模式.
  - 配对只有在绑定模式下才有效 .
- 只有主设备才能发起绑定.

BLE 自定义服务- 绑定

绑定选项

- 设备间首先交互IO 属性信息，用来决定选用哪种配对方式:
  - 立即工作‘Just Works Pairing.’ 两边的设备设置TK值为0
  - 万能钥匙进入’Passkey Entry’. 采用6位数字
  - 带外Out Of Band (OOB)， 提供更高的安全特性。然而，也需要两边设备间有匹配的带外接口 (比如近场通信或者红外通信等)
- 交互的信息用来选取采用哪种秘钥生成方式.
- 注意，任何主设备都可以认证BLE设备和交互绑定信息。为了限制这一点，会引入一些额外的授权流程（尽管这些都是在认证之后进行）.
- 推荐的配对方法依赖于设备的IO特性。这会在蓝牙协议的Section 2.3.5.1 of Volume 3 Part H里提到

BLE 自定义服务- 绑定

属性认证

- GATT服务里的认证会针对每个属性分别进行
- 为了使能认证和接下来的属性加密操作，会按照下面步骤开始进行操作:
  1. GATT服务流程会用来访问一些信息，这些信息在客户端属性读写操作之前，被用来配置认证和建立加密连接。.
  2. 在这个例子中，如果物理连接没有被认证或者被加密，服务器端会发送错误响应，状态字段设置为没有充分加密信息.
  3. 需要读写属性的客户端可以请求采用GAP的流程对物理链路进行认证。一旦这个步骤完成，重新发起请求。.

BLE 自定义服务- 绑定

单设备绑定

- 默认情况下，最新的秘钥总是存储在非易失性RAM里.
  - 在所有睡眠模式下，秘钥都会被保存。只有完全断电之后，秘钥才会清除.
  - 在单个主设备的应用环境中，不需要额外的绑定操作.
  - 与不同的主设备连接会擦除现有的绑定信息.
- “立即工作”的配对机制是不安全的。因为如果有人用sniffer工具去捕捉空中数据，会发现长期秘钥是以空中明文的形式来传输（只在配对阶段）。所以在可能的请况下尽量避免这种情况。
- “万能钥匙进入”的例子将会在下一页PPT里看到，会显示如何在连接阶段使能安全认证

简单绑定的例子将基于培训材料2的”用户自定义服务”的例子进行添加.

BLE 自定义服务 – 绑定

练习: 如何使能MITM万能钥匙接口

- 在文件da1458x_config_basic.h, 使能 CFG_APP_SECURITY

```
#define CFG_APP_SECURITY
```

- 在文件user_config.h, 修改44-48行 44 to 58. (tk.key = PIN code in hex: 0x01E240 = “123456”). 在这个例子里面我们采用固定的秘钥值，通常情况下秘钥值都是随机产生.

```
static const struct security_configuration user_security_configuration = {
            .oob            = GAP_OOB_AUTH_DATA_NOT_PRESENT,
            .key_size       = KEY_LEN,
            .iocap          = GAP_IO_CAP_DISPLAY_ONLY,
            .auth           = GAP_AUTH_REQ_MITM_BOND,
            .sec_req        = GAP_SEC1_AUTH_PAIR_ENC,
            .ikey_dist      = GAP_KDIST_SIGNKEY,
            .rkey_dist      = GAP_KDIST_ENCKEY,
            .tk={
                        .key={0x40,0xE2,0x01,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0},
                },
            .csrk={
                        .key={0xAB,0xAB,0x45,0x55,0x23,0x01,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0},
                 },
};
```

BLE 用户自定义服务 – 绑定

练习：如何使能MITM的秘钥传递接口

- 在 app_default_handlers.c 文件里，删除函数中的前两行 default_app_on_tk_exch_nomitm()

```
void default_app_on_tk_exch_nomitm(uint8_t connection_idx, struct gapc_bond_req_ind const * param)
{
            //    uint32_t pin_code = app_sec_gen_tk();
            //    app_easy_security_set_tk( connection_idx, (uint8_t*) &pin_code, 4 );
            app_easy_security_tk_exch( connection_idx);
}
```

- 在 user_config.h (几乎在文件的最后) 修改

```
.security_request_scenario=DEF_SEC_REQ_NEVER
```

到

```
.security_request_scenario=DEF_SEC_REQ_ON_CONNECT
```

- 在 user_modules_config.h 修改这个宏定义到 (0)

```
#define EXCLUDE_DLG_SEC             (0)
```

BLE 用户自定义服务 – 绑定

练习：如何使能MITM的秘钥传递接口

- 在 user_callback_config.h 文件里检查默认的接口函数是否被设置，默认的工程应该已经设置正确.

```
static const struct app_callbacks user_app_callbacks = {
    .app_on_connection               = default_app_on_connection,
    .app_on_disconnect               = default_app_on_disconnect,
    .app_on_update_params_rejected   = NULL,
    .app_on_update_params_complete   = NULL,
    .app_on_set_dev_config_complete = default_app_on_set_dev_config_complete,
    .app_on_adv_undirect_complete    = app_advertise_complete,
    .app_on_adv_direct_complete      = NULL,
    .app_on_db_init_complete         = default_app_on_db_init_complete,
    .app_on_scanning_completed       = NULL,
    .app_on_adv_report_ind           = NULL,
    .app_on_pairing_request          = default_app_on_pairing_request,
    .app_on_tk_exch_nomitm           = default_app_on_tk_exch_nomitm,
    .app_on_irk_exch                 = NULL,
    .app_on_csrk_exch                = default_app_on_csrk_exch,
    .app_on_ltk_exch                 = default_app_on_ltk_exch,
    .app_on_pairing_succeded         = NULL,
    .app_on_encrypt_ind              = NULL,
    .app_on_mitm_passcode_req        = NULL,
    .app_on_encrypt_req_ind          = default_app_on_encrypt_req_ind,
};
```

BLE 自定义服务 – 绑定

GATT 属性的权限值

- 下面的例子将说明在每个属性里如何使能安全特性
- 修改权限值：从UNAUTH到AUTH
- 使用文件user_config.h的 `.security_request_scenario=DEF_SEC_REQ_ON_CONNECT` 配置

- 使用文件user_config.h的`.security_request_scenario=DEF_SEC_REQ_ON_CONNECT` 配置，你可以选择在连接阶段或者读/写属性阶段使能授权。

BLE 自定义服务 – 绑定

单设备绑定实例

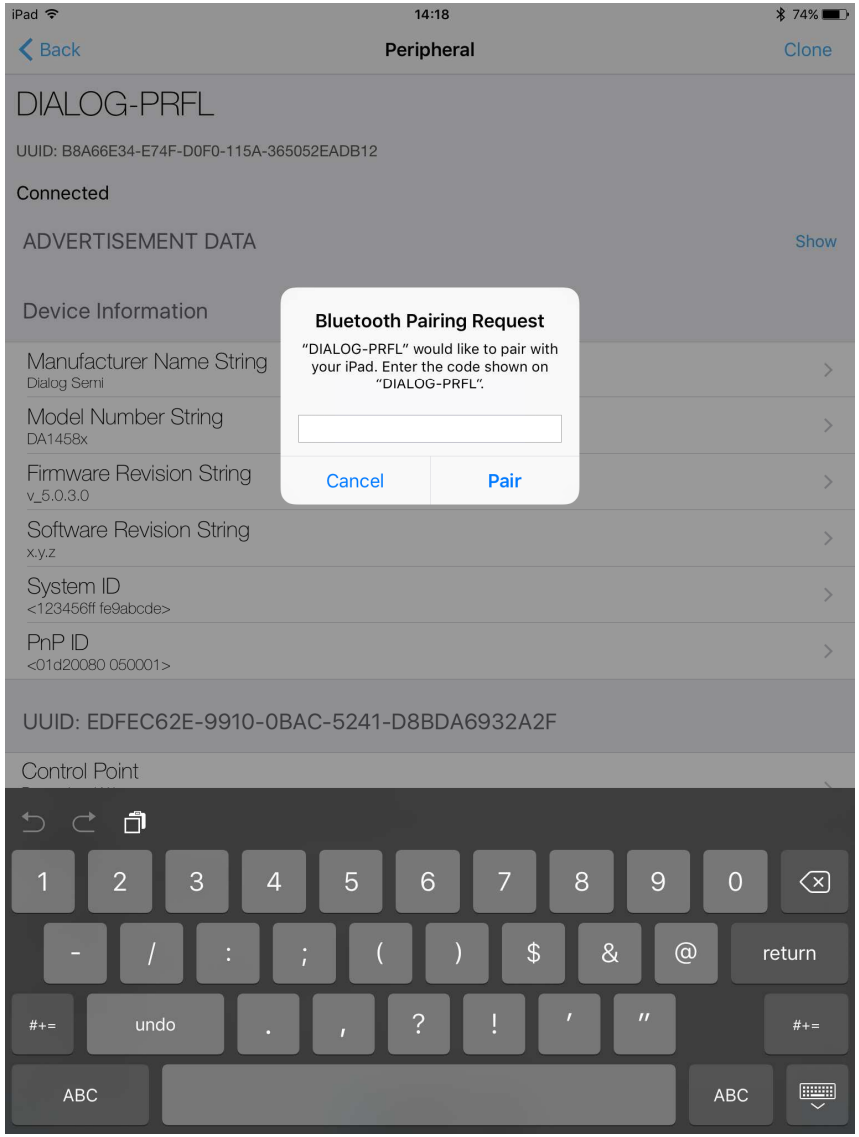
- 如果需要在一个已存在的读或写属性添加配对要求，可以修改数据库里的权限字段，

```
/// Full CUSTOM Database Description - Used to add attributes into the database
static const struct attm_desc_128 custs1_att_db[CUST_IDX_NB] =
{
    // CUSTOM Service Declaration
    [CUST_IDX_SVC]    = {(uint8_t*)&att_decl_svc,
                                 ATT_UUID_16_LEN,
                                 PERM(RD, ENABLE),
                                 sizeof(custom_svc),
                                 sizeof(custom_svc),
                                 (uint8_t*)&custom_svc},

    // Custom Write Characteristic Declaration
    [CUST_IDX_WRITE_CHAR] = {(uint8_t*)&att_decl_char, ATT_UUID_16_LEN,
                                 PERM(RD, ENABLE),
                                 sizeof(custom_write_char),
                                 sizeof(custom_write_char),
                                 (uint8_t*)&custom_write_char},

    // Custom Write Characteristic Value
    [CUST_IDX_WRITE_CHAR_VAL]     = {CUST_WRITE_CHAR_UUID_128,
                                        ATT_UUID_128_LEN,
                                        PERM(WR, UNAUTH),
                                        DEF_CUST_CHAR_LEN,
                                        0,
                                        NULL},
};
```

- 对于支持单设备绑定要求来说，这是唯一的修改.

BLE 自定义服务 – 绑定

你将会看到的输出



BLE 自定义服务 – 绑定

- **Note**: 在这个和之后的例子里设备可以被连接。设备连接后意味着别的扫描设备将无法定位这台设备。建议你只连接自己的设备。

- **注意**: 一些扫描设备（特别是Apple设备）也许不会在显示端更新获得的名字-为了修正这一点，有必要重启蓝牙功能。

BLE 自定义服务 – 绑定

多设备绑定

- 为了支持多设备绑定，每个设备的绑定信息都需要存储.

- 最简单的方式在非易失性RAM里存储绑定信息，添加属性字段说明
  `__attribute__((section("retention_mem_area0"),zero_init)).`

- SDK app_sec模块在存储绑定信息的非易失性RAM里提供了结构app_sec_env
  – 这个模块提供了用于生成密码和长期秘钥（LTK）的函数接口

BLE 用户自定义服务 – 绑定

多设备绑定

- 绑定信息可以按照如下模式进行存储和使用

  -- 在成功配对后，回调函数 `.app_on_pairing_succeded` 会被调用. 在此时，你可以存储 `app_sec_env.rand_nb, app_sec_env.ediv, app_sec_env.ltk` 和 `app_sec_env.key_size` 值到你的永久存储区域.

  – 当回调函数 `.app_on_encrypt_req_ind` 被调用，去查看之前存储的变量 `app_sec_env.rand_nb` 和 `app_sec_env.ediv` . 如果有符合，则写入新值到 `app_sec_env.rand_nb, app_sec_env.ediv, app_sec_env.ltk` 和 `app_sec_env.key_size` ，然后返回True.

- Dialog的IOT sensor和键盘参考设计（support.dialog-semiconductor.com) 里有关于存储绑定信息到EEPROM的代码实例.

BLE 自定义服务 – 绑定

进一步的考虑

- 其他配对模式:
  - 其他配对方式，Dialog平台和SDK都可以支持比如传递键值或者带外方式
- 私有地址:
  - 通常，一个外设会始终使用同一蓝牙地址来广播它的存在信息。但是也可以使用私有地址来模糊它的身份识别
  - 只有和这个设备绑定过，才能用存储的秘钥来解析获得蓝牙地址.
- 配对/非配对方式:
  - 当主设备连接BLE从设备，如果从设备允许配对，它将进行配对操作.
  - 一般来说，通过用户界面可以来操作绑定 – 举例来说，按某个特殊按键会开启广播，并允许绑定

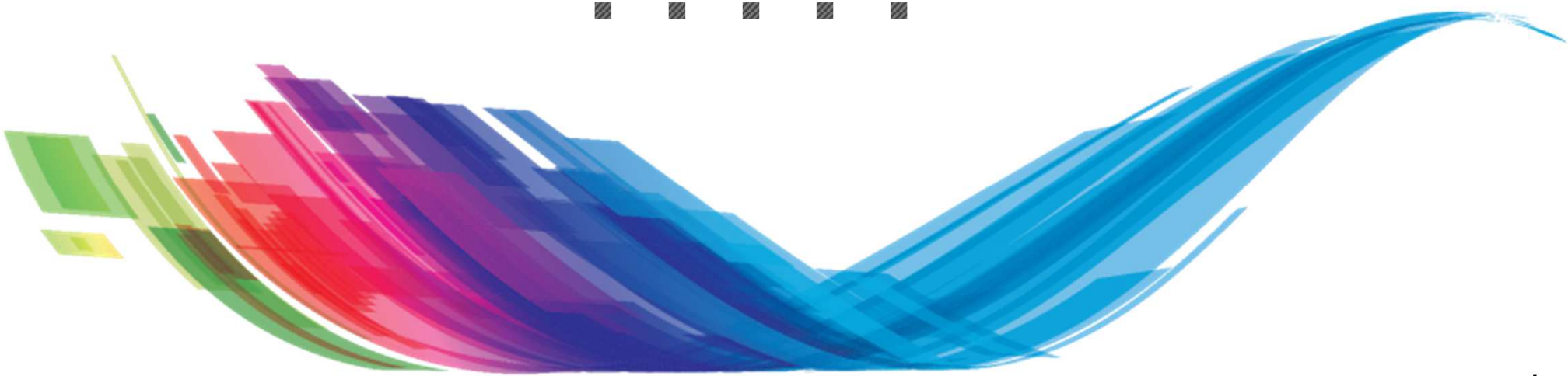
BLE 内容

参考文档

- https://developer.bluetooth.org/gatt/Pages/default.aspx
- https://www.bluetooth.com/specifications/adopted-specifications
- http://support.dialog-semiconductor.com/connectivity
- https://www.wikiwand.com/en/Universally_unique_identifier
- **Register with Dialog semiconductor to get enormous development support**
  - http://support.dialog-semiconductor.com/user/register

The Power To Be…

?????



…personal
…portable
…connected