


Dialog SDK 5.0.3 Training Materials – Pairing, Bonding and Security

2016 June

A large, abstract graphic consisting of multiple overlapping, semi-transparent, wavy bands in shades of blue, purple, magenta, red, orange, and green, creating a sense of motion and depth.

...personal
...portable
...connected

BLE Custom profile with security



BLE security overview

Custom profile service Source code discussion

What would you see as output

BLE Custom profile with security

Let's build a demo together ...



- **Before we start, we recommend you to ...**
 - Take a look at Training material 2 custom profile application

- **What are you going to learn from this training ...**
 - Basic understanding of BLE Security and issues
 - What is Pairing? What is Bonding?
 - 'Just-Works' pairing
 - Single-device bonding
 - Basic understanding of multi-device bonding
 - Small assignment to add pairing in the custom service database

- **What's next ...**
 - See **Reference** section of this training slide

GAP LE Security



Generic Access Profile Low Energy (LE) Security Considerations

- Several security concerns exist with Bluetooth LE communications:
- **Man-in-the-Middle (MITM)**

A MITM requires an attacker to have the ability to both monitor and alter messages into a communication channel. One example is active eavesdropping, in which the attacker makes independent connections with the victims and relays messages between them to make them believe they are talking directly to each other over a private connection. The attacker must be able to intercept all relevant messages passing between the two victims and inject new ones. Protection against MITM attack is obtained by using the passkey entry pairing method or may be obtained using the out of band pairing method.

GAP LE Security



Generic Access Profile Low Energy (LE) Security Considerations

- Several security concerns exist with Bluetooth LE communications:
- **Passive Eavesdropping**

Passive Eavesdropping is secretly capturing pairing data, or normal communications if no pairing has been done, and then listening (by using a sniffing device) to the private communication of others without consent. BLE v4.0 offers no protection against such attacks. (In BLE v4.2, LE Secure Connection uses ECDH public key cryptography as a means to thwart passive eavesdropping attacks – not considered further here.)
- **Privacy/Identity Tracking**

BLE supports the Privacy Feature that reduces the ability to track a LE device over a period of time by changing the Bluetooth device address on a frequent basis. The frequently changing address is called the private address and the trusted devices can resolve it.

GAP LE Security

Generic Access Profile Low Energy (LE) Bonding

- To remove the risk of MITM and passive eavesdropping attacks, two BLE devices may be paired.
- Pairing is the process in which two devices **exchange** security and identity information, to create a trusted relationship.
- This security and identity information is also known as the bonding information. When the devices **store** the bonding information, 'a bond is created' or 'the devices have bonded.'
- Once keys are exchanged, each end of the connection may store the keys. This 'bonds' the devices, and allows them to communicate securely, or to re-pair in future, without re-exchanging keys.
- BLE devices can be configured in non-bondable mode or bondable mode.
 - Pairing is only possible in bondable mode .
- Bonding can only be initiated by the master device.

BLE Custom Profile - Bonding

Authentication Options

- The devices first exchange IO capabilities in the Pairing Feature Exchange to determine which of the following methods are available to pair:
 - ‘Just Works Pairing.’ Both devices set the Temporary Key value to zero
 - Passkey Entry. Uses 6 numeric characters
 - Out Of Band (OOB). Offers greater security than using the Passkey Entry or Just Works methods. However, both devices need to have matching OOB interfaces (e.g. near-field or infra-red communications, etc.)
- The information exchanged is used to select which key generation method is used.
- Note that any Master can authenticate with a BLE device and exchange bonding information. To limit access, some further authorization must be performed (although this can be done after authentication).
- The recommended pairing scheme depends on the IO capabilities of the device; this is described in Section 2.3.5.1 of Volume 3 Part H of the Bluetooth Specification



BLE Custom Profile - Bonding

Authentication of Characteristics

- Authentication in the GATT Profile is applied to each characteristic independently
- To enable authentication and subsequent encryption of a characteristic, the following steps must occur:
 1. The GATT Profile procedures that are used to access information can be configured to require the client to be authenticated and have an encrypted connection before a characteristic can be read or written.
 2. In this case, and where the physical link is unauthenticated or unencrypted, the server shall send an Error Response with the status code set to Insufficient Authentication.
 3. The client wanting to read or write this characteristic can then request that the physical link be authenticated using the GAP authentication procedure, and once this has been completed, send the request again.

BLE Custom Profile - Bonding



Single Device Bonding

- By default, the latest key is always stored in retention RAM.
 - This is retained in all sleep modes and will remain unless the device is completely powered down.
 - If using a single master, no extra bonding needs to be performed.
 - Connecting with a different master will erase the existing bonding information.
- The "just works" version of the pairing mechanism is insecure if the air traffic is sniffed by someone during pairing since the long term key is sent in plain text over the air (but only during the pairing phase), so avoid this mode if possible.
- Passkey entry (MITM) example is shown in the next slides. The example shows how to enable security at connection.

The example for simple bonding is based on the custom profile example from Training Material 2.

BLE Custom Profile - Bonding

Exercise: how to enable MITM passkey entry

- In da1458x_config_basic.h, enable **CFG_APP_SECURITY**

```
#define CFG_APP_SECURITY
```

- In user_config.h, change line 44 to 58. (tk.key = PIN code in hex: 0x01E240 = “123456”). We are using a fixed PIN code in this example, but normally the PIN is randomly generated.

```
static const struct security_configuration user_security_configuration = {
    .oob                = GAP_OOB_AUTH_DATA_NOT_PRESENT,
    .key_size           = KEY_LEN,
    .iocup              = 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
    },
    .csr={
        .key={0xAB, 0xAB, 0x45, 0x55, 0x23, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
    },
};
```

BLE Custom Profile - Bonding

Exercise: how to enable MITM passkey entry

- In `app_default_handlers.c` disable the first two lines in function `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);
}
```

- In `user_config.h` (almost at the end of the file) change this line

```
.security_request_scenario=DEF_SEC_REQ_NEVER
```

into

```
.security_request_scenario=DEF_SEC_REQ_ON_CONNECT
```

- In `user_modules_config.h` change this define into (0)

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

BLE Custom Profile - Bonding

Exercise: how to enable MITM passkey entry

- In `user_callback_config.h` check that the default handlers are set, the default project should be correct.

```
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_succeeded     = NULL,
    .app_on_encrypt_ind          = NULL,
    .app_on_mitm_passcode_req    = NULL,
    .app_on_encrypt_req_ind      = default_app_on_encrypt_req_ind,
};
```

BLE Custom Profile - Bonding

GATT Characteristic Value permissions



- The following example shows how to enable security per characteristic
- This can be achieved by changing the permissions from UNAUTH to AUTH
- Using this setting `.security_request_scenario=DEF_SEC_REQ_ON_CONNECT` in `user_config.h` you can select when authorization is required: during connection or during read/write of a characteristic.

BLE Custom Profile - Bonding

Single Device Bonding Example

- To convert an existing read or write characteristic to require pairing change the Characteristic Value permissions in the Database Description change the permission flag:

```
/// 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},

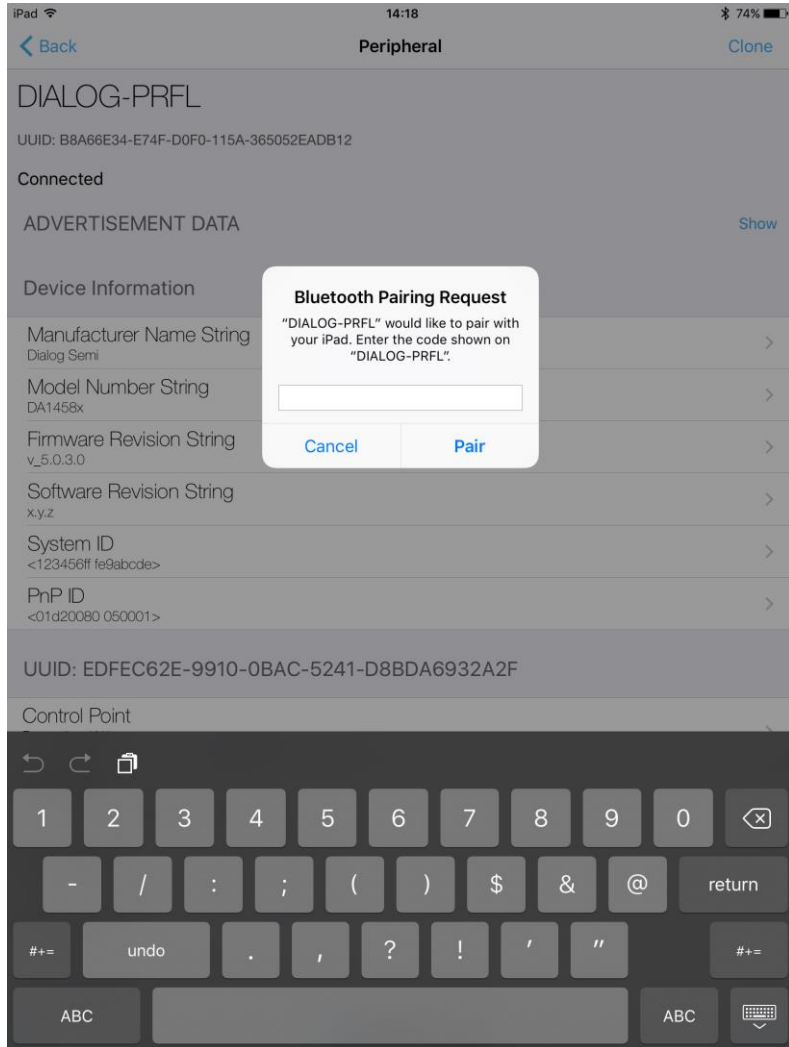
    // Custom Write Characteristic Declaration
    [CUST_IDX_WRITE_CHAR] = {(uint8_t*)&att_decl_char, ATT_UUID_16_LEN,
                              PERM(RD, ENABLE),
                              sizeof(custom_write_char),
                              sizeof(custom_write_char),
                       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

- This is the only change required to support bonding with a single Master.

BLE Custom Profile - Bonding

What would you see as output



BLE Custom Profile - Bonding



- **Note:** The devices will be connectable in this and future examples. Connecting to a device will mean that other scanners won't be able to locate the device – it is recommended that you only connect to your own device.
- **Note:** Some scanners (notably Apple devices) may not update the name of device if it is changed – to correct this, it is necessary to disable then re-enable Bluetooth.

BLE Custom Profile - Bonding



Multiple Device Bonding

- To support multiple devices, the bonding information must be stored for each device.
- The simplest way is to store the bonding information in retained memory, using the attribute `__attribute__((section("retention_mem_area0"), zero_init))`.
- The SDK **app_sec** module provides a structure `app_sec_env` in this retained memory in which bonding information may be stored.
 - This module also provides helper functions to generate PINs and the Long Term Key

BLE Custom Profile - Bonding



Multiple Device Bonding

- Bonding information may be stored and used using the following procedure:
 - On successful pairing, the callback `.app_on_pairing_succeeded` will be called. At this point you may store the `app_sec_env.rand_nb`, `app_sec_env.ediv`, `app_sec_env.ltk` and `app_sec_env.key_size` values to your permanent store.
 - When the callback `.app_on_encrypt_req_ind` is called, do a lookup on the `app_sec_env.rand_nb` and `app_sec_env.ediv` variables stored previously. If a match is found, write the values to `app_sec_env.rand_nb`, `app_sec_env.ediv`, `app_sec_env.ltk` and `app_sec_env.key_size` and return true.
- Dialog's IoT Sensor and Keyboard reference designs (available through support.dialog-semiconductor.com) include example code dealing with storing bonding information to EEPROM.

BLE Custom Profile - Bonding



Further Considerations

- Other pairing modes:
 - Other pairing methods like Pass Key Entry and Out Of Band are supported by the Dialog platform and SDK
- Private addresses:
 - Generally, a peripheral will broadcast its presence to all listeners using the same address every time. It is possible to obfuscate the identity of a peripheral using a 'Private Address'
 - Only devices which are bonded to the client can resolve the address, using their stored keys.
- Bondable / non-bondable:
 - When a master connects to a BLE slave, it may only pair if the slave allows it.
 - Typically, bonding can be controlled using a user interaction on the device – for example, pressing a specific button will start the device advertising in a mode that allows bonding.

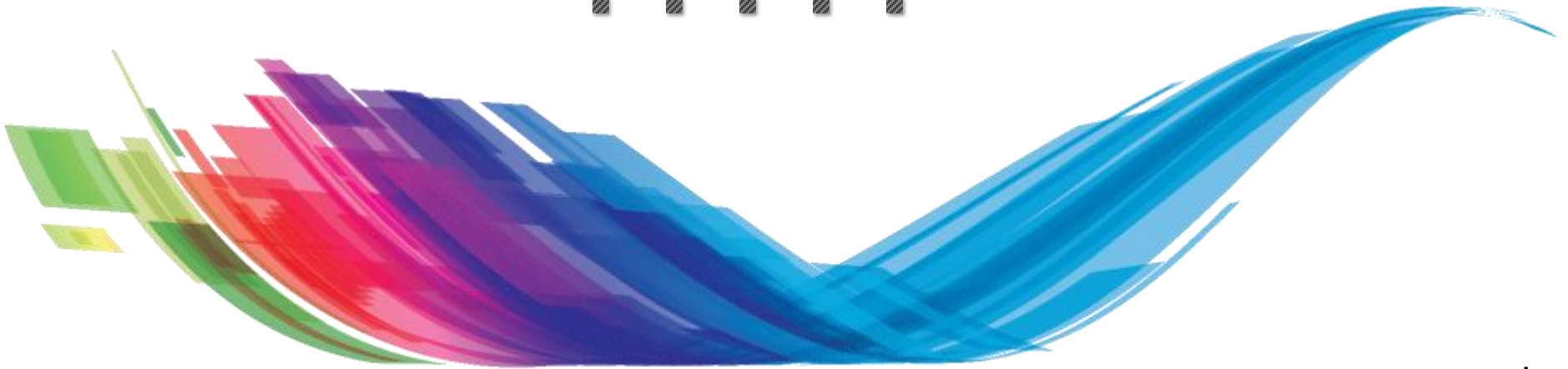
BLE Contents

Reference

- <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