Chapter 5:   Security and Privacy

In this chapter you will learn about security and privacy in Bluetooth® LE. First, the concepts for both security and privacy will be explained and then the method to implement them in the firmware will be described along with the stack events that will occur. Methods to store security information in non-volatile memory will also be explained.

Table of contents

Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

5.1   Security

In any type of system, to securely communicate between two devices you need to:

1. Authenticate that both sides know who they are talking to,

2. Ensure that all access to data is Authorized,

3. Encrypt all message that are transmitted, and

4. Verify the Integrity of those messages.

In Bluetooth® LE, this entire security framework is built around AES-128 symmetric key encryption. This type of encryption works by combining a shared secret code and the unencrypted data (typically called plain text) to create an encrypted message (typically called cypher text).

```
CypherText = F(SharedSecret,PlainText)
```

There is a bunch of math that goes into AES-128, but for all practical purposes if the shared secret code is kept secret, you can assume that it is very unlikely that someone can read the original message.

If this scheme depends on a shared secret, the next question is how do two devices that have never been connected get a shared secret that no one else can see? In Bluetooth® LE, the process for achieving this state is called Pairing. A device that is Paired is said to be Authenticated.
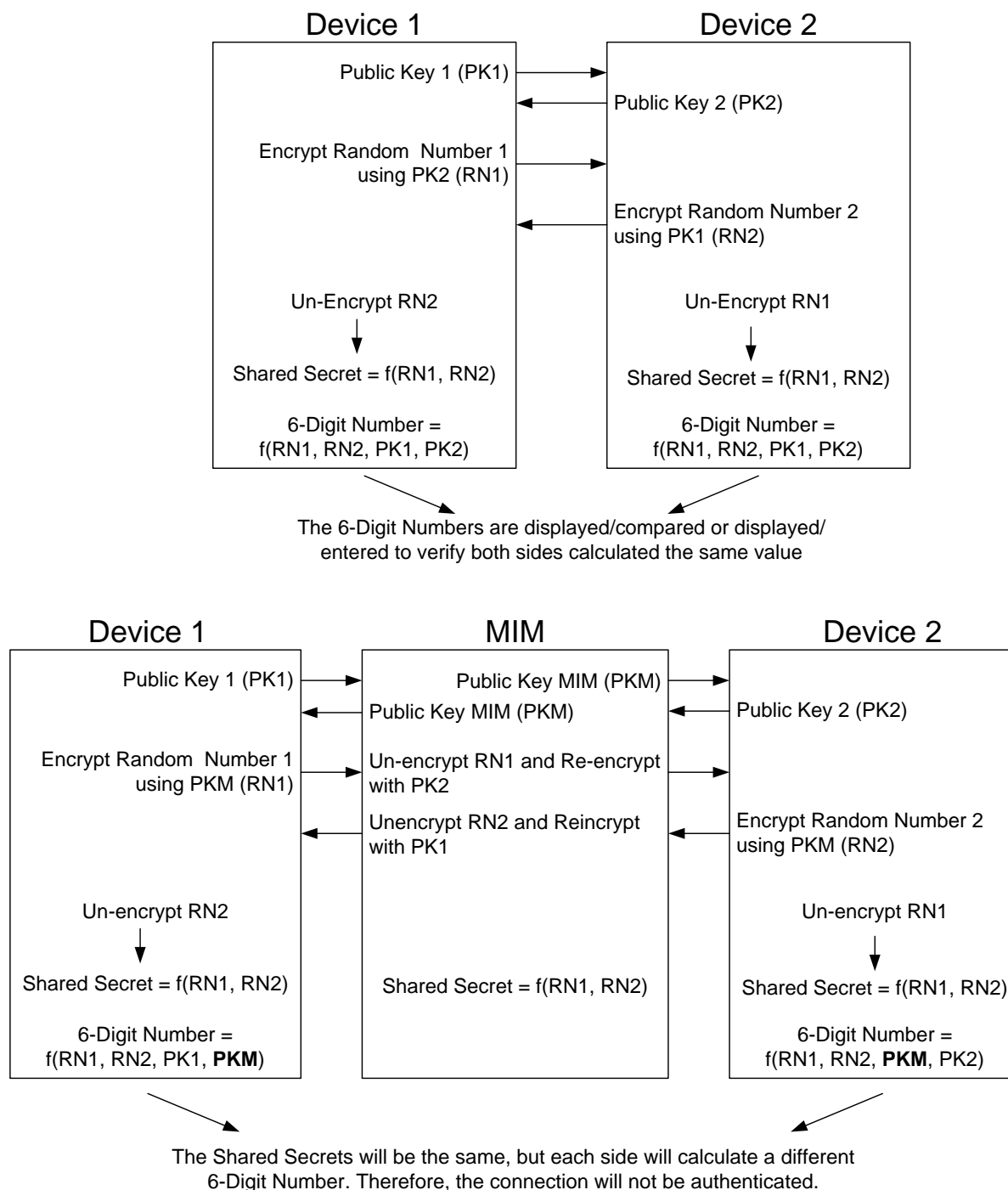
5.1.1 Pairing

Pairing is the process of arriving at the shared secret. The basic problem continues to be how do you send a shared secret over the air, unencrypted and still have your shared secret stay a secret? The answer is that you use public key encryption. Both sides have a public/private key pair that is either embedded in the device or calculated at startup. When you want to authenticate, both sides of the connection exchange public keys. Then both sides exchange encrypted random numbers that form the basis of the shared secret.
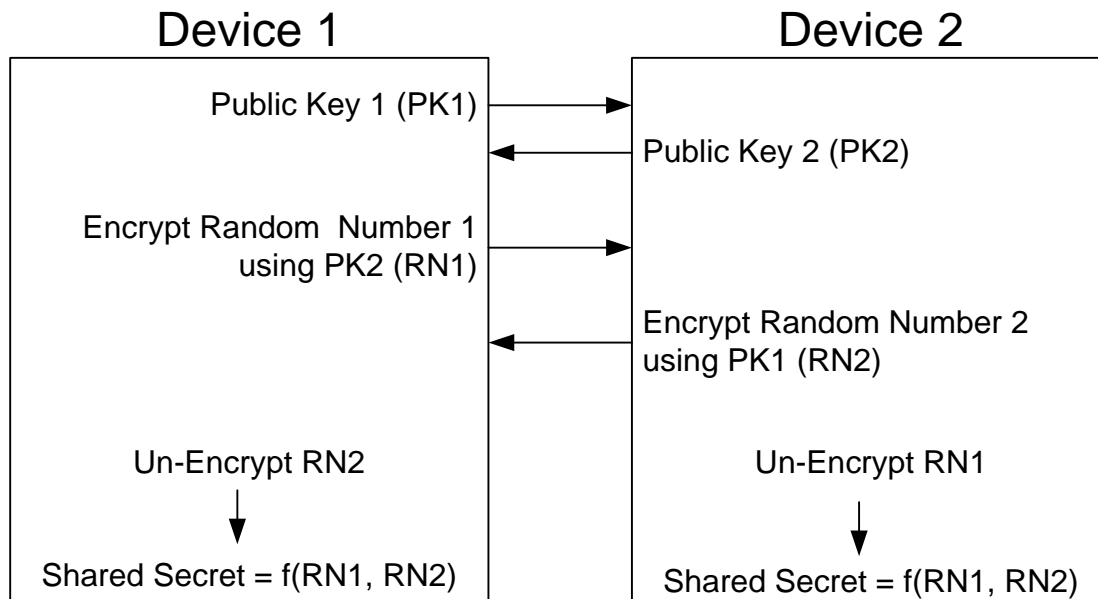
But how do you protect against Man-In-The-Middle (MIM)? There are four possible methods.

- Method 1 is called "Just works". In this mode you have no protection against MIM.

- Method 2 is called "Out of Band". Both sides of the connection need to be able to share the PIN via some other connection that is not Bluetooth® such as NFC.

- Method 3 is called "Numeric Comparison" (V2.PH.7.2.1). In this method, both sides display a 6-digit number that is calculated with a nasty cryptographic function based on the random numbers used to generate the shared key and the public keys of each side. The user observes both devices. If the number is the same on both, then the user confirms on one or both sides. If there is a MITM, then the random numbers on both sides will be different so the 6-digit codes would not match.

- Method 4 is called "Passkey Entry" (V2.PH.7.2.3). For this method to work, at least one side needs to be able to enter a 6-digit Passkey. The other side must be able to display the Passkey. One device displays the Passkey and the user is required to enter the Passkey on the other device. Then an exchange and comparison process happens with the Passkeys being divided up, encrypted, exchanged and compared.

Pictorially, the process with no MIM and with MIM for methods 3 and 4 is shown below. If there is a man in the middle, the two sides will calculate different numbers because the number is a function of the public keys used to encrypt the random numbers.

The 6-Digit Numbers are displayed/compared or displayed/ entered to verify both sides calculated the same value

The Shared Secrets will be the same, but each side will calculate a different 6-Digit Number. Therefore, the connection will not be authenticated.

5.1.2 Bonding

The whole process of Pairing is a bit painful and time consuming. It is also the most vulnerable part of establishing security, so it is beneficial to do it only once. Certainly, you don't want to have to repeat it every time two devices connect. This problem is solved by Bonding, which just saves all the relevant information into non-volatile memory. The information is stored by both devices that are part of the connection. This allows the next connection between those two devices to happen without repeating the pairing process.

5.1.3 Pairing & Bonding process summary

The process that occurs when you pair and bond can be seen pictorially below.
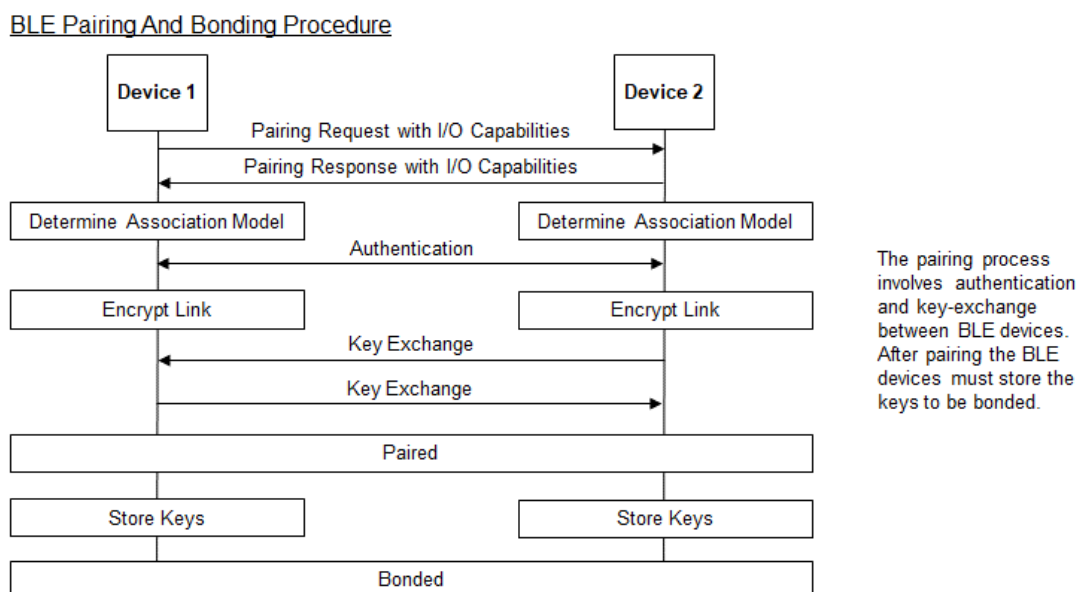
5.1.4 Authentication, Authorization and the GATT database

In prior chapters we talked about Attributes and the GATT Database. Each Attribute has a permissions bit field that includes bits for Encryption, Authentication, and Authorization. The stack will guarantee that you will not be able to access an Attribute that is marked for Encryption or Authentication unless the connection is Authenticated and/or Encrypted.

The Authorization flag is not enforced by the stack. The application is responsible for implementing the Authorization semantics. For example, you might not allow someone to turn off/on a switch without entering a password first.

5.2   Privacy

Bluetooth® LE devices are identified using a 48-bit device address. This device address is part of all the packets sent by the device in the advertising channels. A third device which listens on all three advertising channels can easily track the activities of a device by using its device address. Link Layer Privacy is a feature that reduces the ability to track a Bluetooth® LE device by using a private address that is generated and changed at regular intervals. Note that this is different than security (i.e. encrypting of messages).

There are a few different address types possible for Bluetooth® LE devices:
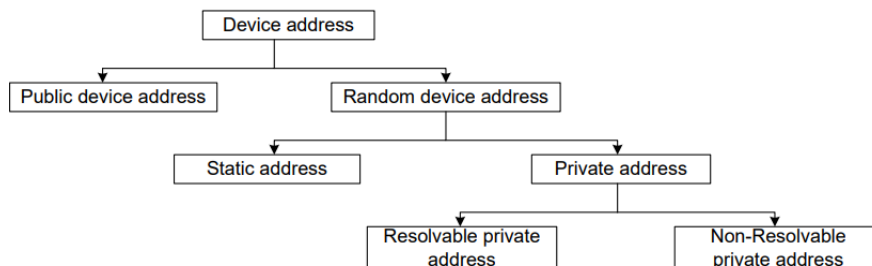
The device address can be a Public Device Address or a Random Device Address. The Public Device Addresses are comprised of a 24-bit company ID (an Organizationally Unique Identifier or OUI based on an IEEE standard) and a 24-bit company-assigned number (unique for each device); these addresses do not change over time.

There are two types of Random Addresses: Static Address and Private Address. The Static Address is a 48-bit randomly generated address with the two most significant bits set to 1. Static Addresses are generated on first power up or during manufacturing. A device using a Public Device Address or Static Address can be easily discovered and connected to by a peer device. Private Addresses change at some interval to ensure that the Bluetooth® LE device cannot be tracked. A Non-Resolvable Private Address cannot be resolved by any device so the peer cannot identify who it is connecting to. Resolvable Private Addresses (RPA) can be resolved and are used by Privacy-enabled devices.

Every Privacy-enabled Bluetooth® LE device has a unique address called the Identity Address and an Identity Resolving Key (IRK). The Identity Address is the Public Address or Static Address of the Bluetooth® LE device. The IRK is used by the Bluetooth® LE device to generate its RPA and is used by peer devices to resolve the RPA of the Bluetooth® LE device. Both the Identity Address and the IRK are exchanged during the third stage of the pairing process. Privacy-enabled Bluetooth® LE devices maintain a list that consists of the peer device's Identity Address, the local IRK used by the Bluetooth® LE device to generate its RPA, and the peer device's IRK used to resolve the peer device's RPA. This is called the Resolving List. Only peer devices that have the 128-bit identity resolving key (IRK) of a Bluetooth® LE device can determine the device's address.

A Privacy-enabled Bluetooth® LE device periodically changes its RPA to avoid tracking. The Bluetooth® LE Stack configures the Link Layer with a value called RPA Timeout that specifies the time after which the Link Layer must generate a new RPA. This value is set in *app_bt_cfg.c* and is called `rpa_refresh_timeout`. If the `rpa_refresh_timeout` is set to 0 (i.e. `WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_NEVER_CHANGE`), privacy is disabled, and a public device address will be used.

Apart from this, Bluetooth® 5.0 introduced more options in the form of privacy modes. There are two modes: device privacy mode and network privacy mode. A device in device privacy mode is only concerned about the privacy of the device itself and will accept advertising physical channel PDU's (Advertising, Scanning and Initiating packets) from peer devices that contain their identity address as well as ones that contain a private

address, even if the peer device has distributed its IRK in the past. In network privacy mode, a device will only accept advertising packets from peer devices that contain a private address. By default, network privacy mode is used when private addresses are resolved and generated by the Controller. The Host can specify the privacy mode to be used with each peer identity on the resolving list.

The following table shows the logical representation of the resolving list entries. Depending on the privacy mode entry in the resolving list, the device will behave differently with each peer device.

Device	Local IRK	Peer IRK	Peer Identity Address	Identity Address Type	Privacy Mode
1	Local IRK	Peer 1 IRK	Peer 1 Identity Address	Static/Public	Network/Device
2	Local IRK	Peer 2 IRK	Peer 2 Identity Address	Static/Public	Network/Device
3	Local IRK	Peer 3 IRK	Peer 3 Identity Address	Static/Public	Network/Device

## 5.3   Configuration settings for security and privacy

The ModusToolbox™ Bluetooth® Configurator makes it simple to set up security and privacy. Examples of each are shown below.

5.3.1   Security

In order to enable security (i.e. to require paring before allowing the client read or write of a characteristic) you just enable the **Read Authentication Required** or **Write Authentication Required** check box in the permissions for that characteristic. Note that this is a bitmask setting, so you must still keep **Read** and/or **Write** selected when you enable authentication. If not, reads/writes to that characteristic will fail. Enabling security works the same way for Descriptors such as the CCCD. That is, you can require authentication before allowing reads/writes of the CCCD (thereby preventing the client from turning Notifications on/off without pairing first).

Note: *The security settings for each Characteristic and its Descriptors are independent so make sure you set the permissions for everything the way you want them.*

As an example, for prior exercises, the Bluetooth® configuration settings for the Counter and its CCCD with security enabled look like this:
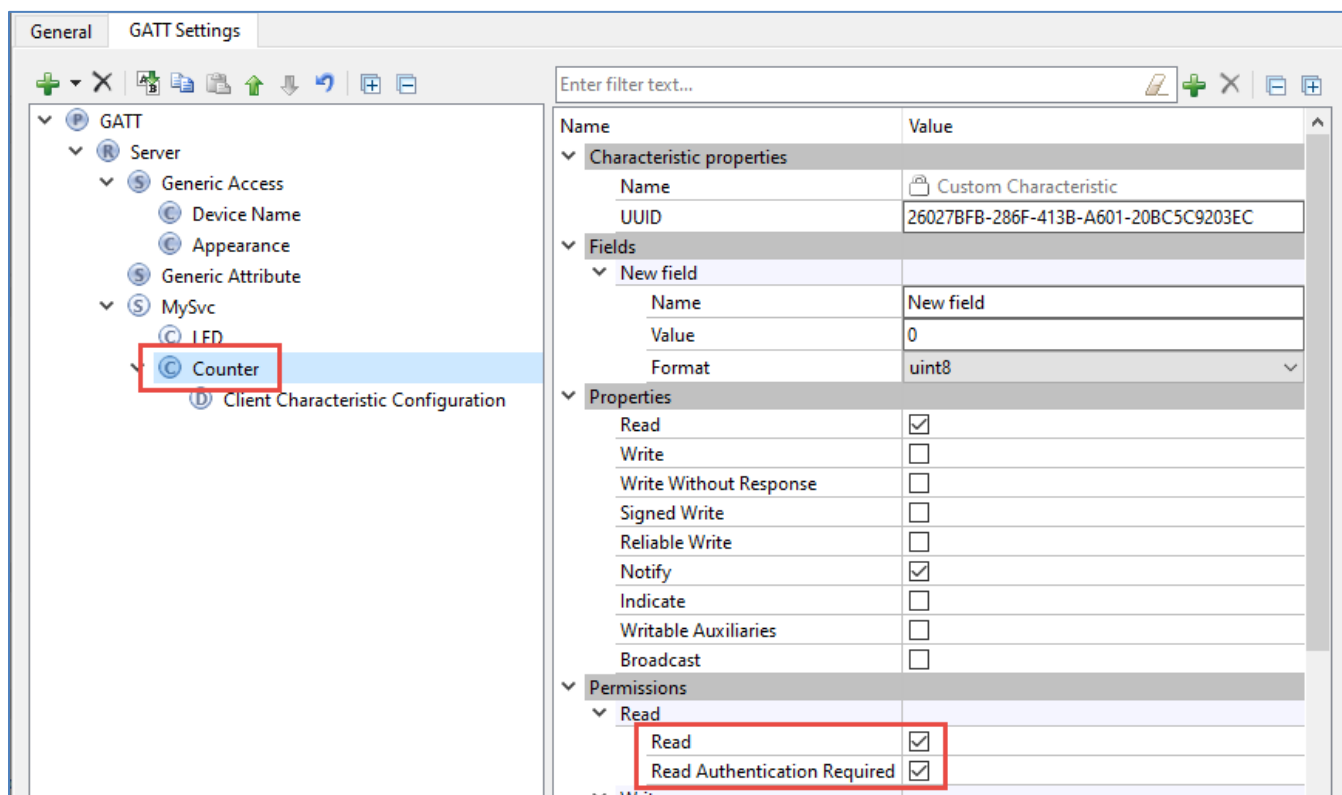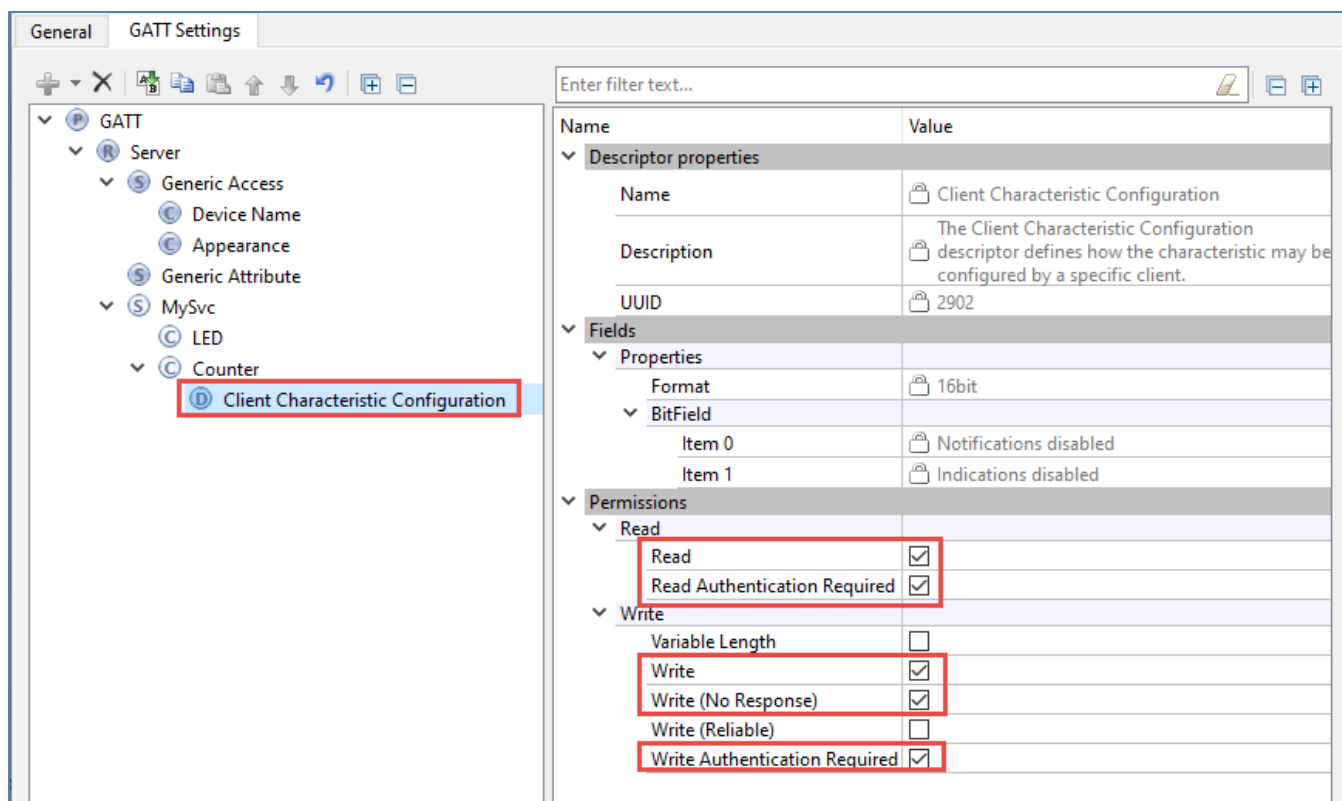
5.3.2   Privacy

A Resolvable Private Address (RPA) can be enabled for a device simply by changing the setting in the app_bt_cfg.c file for rpa_refressh_timeout in the wiced_bt_cfg_settings structure. You can add any value you want, but there is a macro for disabled (0) and for the default timeout of 900 seconds. For example:

RPA disabled:

```
.rpa_refresh_timeout = WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_NEVER_CHANGE,
```

RPA default timeout (900 seconds):

```
.rpa_refresh_timeout = WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_CHANGE_TIMEOUT,
```

Note: *A non-zero RPA will be used in all of our exercises because some iOS devices require that RPA is enabled for peripherals to function properly.*

5.4 Firmware for security and privacy

The only differences in firmware for security and privacy are additional Bluetooth® stack management events that your firmware needs to respond to.

5.4.1 Pairing

For a typical application that connects using a Paired link but does <u>NOT</u> use privacy, does <u>NOT</u> store bonding information in NVRAM, and does <u>NOT</u> require a passkey, the callback events will look like this:

Activity	Callback Event Name (both Stack and GATT)	Reason
Powerup	`BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT`	At initialization, the Bluetooth LE stack looks to see if the privacy keys are available. A return value of `WICED_BT_ERROR` indicates to the stack that it should generate new privacy keys since we are not storing values in EEPROM yet.
	`BTM_ENABLED_EVT`	This occurs once the Bluetooth LE stack has completed initialization. Typically, you will start up the rest of your application here.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	This occurs when you enable advertisements. You will see a return value of 3 for fast advertisements. After a timeout, you may see this again with a return value of 4 for slow advertisements. Eventually the state changes to 0 (off) if there have been no connections, giving you a chance to save power.
	`BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT`	This event is called if reading of the privacy keys from EEPROM failed (i.e. the return value from `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT` was not `WICED_BT_SUCESS`). Since we are not yet implementing bonding, this should just return `WICED_BT_SUCCESS`.
Connect	`GATT_CONNECTION_STATUS_EVT`	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	The Stack stops advertisements once the connection is established.
	`BTM_BLE_CONNECTION_PARAM_UPDATE`	The Client and Peripheral negotiate their connection requirements and capabilities to determine the connection parameters.
Pairing	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	The stack is requesting pairing keys for the connected peer. In this case, we have not saved any keys in NVRAM so this state just returns `WICED_BT_ERROR` to tell the stack to generate new keys.
	`BTM_SECURITY_REQUEST_EVT`	The occurs when the client requests a secure connection. When this event happens, you need to call `wiced_bt_ble_security_grant` to allow a secure connection to be established.

Activity	Callback Event Name (both Stack and GATT)	Reason
	`BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT`	This occurs when the client asks what type of capability your device has that will allow validation of the connection (e.g. screen, keyboard, etc.). You need to set the appropriate values when this event happens.
	`BTM_ENCRYPTION_STATUS_EVT`	This occurs when the secure link has been established.
	`BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT`	This event is used so that you can store the paired devices keys if you are storing bonding information. If not, then this state does not need to be implemented.
	`BTM_PAIRING_COMPLETE_EVT`	This event indicates that pairing has been completed successfully.
	`BTM_BLE_PHY_UPDATE_EVT`	This event indicates that the link has now been switched over to an encrypted link.
Read Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_READ`	The firmware must get the value from the correct location in the GATT database.
Write Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_WRITE`	The firmware must store the provided value in the correct location in the GATT database.
Notifications	`N/A`	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	`GATT_CONNECTION_STATUS_EVT`	For a disconnection, the connection ID is reset, all CCCD settings are cleared, and advertisements are restarted.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Upon a disconnect, the firmware will get a GATT event handler callback for the `GATT_CONNECTION_STATUS_EVENT`. At that time, it is the user's responsibility to determine if advertising should be re-started. If it is restarted, then you will get a Bluetooth® LE stack callback once advertisements have restarted with a return value of 3 (fast advertising) or 4 (slow advertising).

5.4.2 Pairing + Bonding + Privacy

If bonding information is stored to NVRAM, the event sequence will look like the following. The sequence is shown for three cases (each shaded differently):

1. First-time connection before bonding information is saved

2. Connection after bonding information has been saved for disconnect/re-connect without resetting the kit between connections.

3. Connection after bonding information has been saved for disconnect/reset/re-connect.

In the reconnect cases, you can see that the pairing sequence is greatly reduced since keys are already available.

Activity	Callback Event Name	Reason
1st Powerup	`BTM_LOCAL_IDENTITY_KEYS_REQUEST _EVT`	When this event occurs, if privacy is enabled the firmware needs to load the privacy keys from NVRAM, provide them to the stack and return `WICED_BT_SUCESS`. If keys have not been previously saved for the device then this state must return a value other than `WICED_BT_SUCESS` such as `WICED_BT_ERROR`. The non-success return value causes the stack to generate new privacy keys.
	`BTM_ENABLED_EVT`	This occurs once the Bluetooth LE stack has completed initialization. Typically, you will start up the rest of your application here. During this event, the firmware needs to load keys (which also includes the BD_ADDR) for a previously bonded device from NVRAM so that it will have the necessary keys to allow connecting to a bonded device. If a device has not been previously bonded, this will return values of all 0. If RPA is being used, you must also call `wiced_bt_dev_add_device_to_address_resolution _db` so that the random private addresses can be resolved.
	`BTM_BLE_ADVERT_STATE_CHANGED_EV T`	This occurs when you enable advertisements. You will see a return value of 3 for fast advertisements. After a timeout, you may see this again with a return value of 4 for slow advertisements. Eventually the state changes to 0 (off) if there have been no connections, giving you a chance to save power.
	`BTM_LOCAL_IDENTITY_KEYS_UPDATE_ EVT`	This event is called if reading of the privacy keys from NVRAM failed (i.e. the return value from `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT` was not `WICED_BT_SUCESS`). During this event, the privacy keys must be saved to NVRAM.
	`BTM_LOCAL_IDENTITY_KEYS_UPDATE_ EVT`	This is called twice to update both the IRK and the ER in two steps.

Activity	Callback Event Name	Reason
1st Connect	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	The stack looks at security information for the previous pairing (if any). This is done to prevent a peer from reconnecting with reduced security. In this case, we have not saved any keys in NVRAM so this state just returns `WICED_BT_ERROR`.
	`GATT_CONNECTION_STATUS_EVT`	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Once the connection happens, the stack stops advertisements which will result in this event. You will see a return value of 0 which means advertisements have stopped.
1st Pair	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	The latest keys are requested by the stack before requesting the application to write them to NVRAM. Again, in this case, we have not saved any keys in NVRAM yet so this state just returns `WICED_BT_ERROR`.
	`BTM_SECURITY_REQUEST_EVT`	The occurs when the client requests a secure connection. When this event happens, you need to call `wiced_bt_ble_security_grant` to allow a secure connection to be established.
	`BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT`	This occurs when the client asks what type of capability your device has that will allow validation of the connection (e.g. screen, keyboard, etc.). You need to set the appropriate values when this event happens.
	`BTM_PASSKEY_NOTIFICATION_EVT`	This event only occurs if the IO capabilities are set such that your device has the capability to display a value, such as `BTM_IO_CAPABILITIES_DISPLAY_ONLY`. In this event, the firmware should display the passkey so that it can be entered on the client to validate the connection.
	`BTM_USER_CONFIRMATION_REQUEST_EVT`	This event only occurs if the IO capabilities are set such that your device has the capability to display a value and accept Yes/No input, such as `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`. In this event, the firmware should display the passkey so that it can be compared with the value displayed on the Client. This state should also provide confirmation to the Stack (either with or without user input first).
	`BTM_ENCRYPTION_STATUS_EVT`	This occurs when the secure link has been established. Previously saved information such as paired device BD_ADDR and notify settings is read. If no device has been previously bonded, this will return all 0's.
	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	The latest keys are requested by the stack before requesting the application to write them to NVRAM. Since this is the first pairing, we have not saved any keys in NVRAM yet so this state just returns `WICED_BT_ERROR`.

Activity	Callback Event Name	Reason
	`BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT`	During this event, the firmware needs to store the keys of the paired device (including the BD_ADDR) into NVRAM so that they are available for the next time the devices connect.
	`BTM_PAIRING_COMPLETE_EVT`	This event indicates that pairing has been completed successfully.
Read Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_READ`	The firmware must get the value from the correct location in the GATT database.
Write Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_WRITE`	The firmware must store the provided value in the correct location in the GATT database.
Notifications	`N/A`	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Upon a disconnect, the firmware will get a GATT event handler callback for the `GATT_CONNECTION_STATUS_EVENT` (more on this later). At that time, it is the user's responsibility to determine if advertising should be re-started. If it is restarted, then you will get a Bluetooth LE stack callback once advertisements have restarted with a return value of 3 (fast advertising) or 4 (slow advertising).
Re-Connect	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	The stack looks at security information for the previous pairing (if any). This is done to prevent a peer from reconnecting with reduced security. In this case, we have saved keys in NVRAM so this state returns `WICED_BT_SUCCESS`. If RPA is being used, you must also call `wiced_bt_dev_add_device_to_address_resolution_db` so that the random private addresses can be resolved.
	`GATT_CONNECTION_STATUS_EVT`	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Advertising off.
Re-Pair	`BTM_ENCRYPTION_STATUS_EVT`	In this state, the firmware reads the state of the server from NVRAM. For example, the saved state of any notify settings may be read. Since the paired device BD_ADDR and keys were already available, no other steps are needed to complete pairing.
Read Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_READ`	The firmware must get the value from the correct location in the GATT database.
Write Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_WRITE`	The firmware must store the provided value in the correct location in the GATT database.

Activity	Callback Event Name	Reason
Notifications	`N/A`	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Advertising on.
Reset	`BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT`	Local keys are loaded from NVRAM.
	`BTM_ENABLED_EVT`	Stack is enabled. Paired device keys (including the BD_ADDR) are loaded from NVRAM and the device is added to the address resolution database.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Advertising on.
Re-Connect	`BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`	This event is called by the stack so that the firmware can load the paired device's keys from NVRAM. Since keys are available, this state must return `WICED_BT_SUCCESS`. If RPA is being used, you must also call `wiced_bt_dev_add_device_to_address_resolution_db` so that the random private addresses can be resolved.
	`GATT_CONNECTION_STATUS_EVT`	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Advertising off.
Re-Pair	`BTM_ENCRYPTION_STATUS_EVT`	In this state, the firmware reads the state of the server from NVRAM. For example, the saved state of any notify settings may be read. Since the paired device BD_ADDR and keys were already available in NVRAM, no other steps are needed to complete pairing.
Read Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_READ`	The firmware must get the value from the correct location in the GATT database.
Write Values	`GATT_ATTRIBUTE_REQUEST_EVT` → `GATTS_REQ_TYPE_WRITE`	The firmware must store the provided value in the correct location in the GATT database.
Notifications	`N/A`	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	`BTM_BLE_ADVERT_STATE_CHANGED_EVT`	Advertising on.

5.5 Exercises

Exercise 1: Paring

In this exercise, you will add Pairing and Security (Encryption) to the competed notification exercise from the previous chapter.

Application Creation

1. Create a new application with the CYW920835M2EVB-01 BSP.

   On the application template page, use the **Browse** button to start from the completed application for the *ch04_ex01_notify* exercise from the previous chapter. If you did not complete that exercise, the solution can be found in *Projects/key_ch04_ex01_notify*.
   Name the new application **ch05_ex01_pair**.

2. Open the Bluetooth® Configurator.

3. Change the device name to `<inits>_pair`.

4. Make the following changes to the Counter Characteristic:

   a. Set the "Read Authentication Required" permission, which will make the peripheral reject read requests unless the devices are paired first.

   b. Update the Client Characteristic Configuration Descriptor to require authenticated read and write. This will cause the application to require pairing to view or change the notification settings.

5. Save your edits and close the configurator.

6. In *app.c*, make the following changes.

   a. Look for the call to `wiced_bt_set_pairable_mode` mode and set the first argument to `WICED_TRUE` to allow pairing.

   Note: *Leave the second argument as `WICED_FALSE` - when set to true, this argument indicates that ONLY previously paired devices are allowed to connect.*

   b. In the `BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT` management case tell the central that you want MITM protection, but the device has no IO capabilities. The required code is shown below.

   ```
   p_event_data->pairing_io_capabilities_ble_request.auth_req =
   BTM_LE_AUTH_REQ_SC_MITM_BOND;
   p_event_data->pairing_io_capabilities_ble_request.init_keys =
   BTM_LE_KEY_PENC|BTM_LE_KEY_PID;
   p_event_data->pairing_io_capabilities_ble_request.local_io_cap =
   BTM_IO_CAPABILITIES_NONE;
   p_event_data->pairing_io_capabilities_ble_request.max_key_size = 0x10;
   p_event_data->pairing_io_capabilities_ble_request.resp_keys =
   BTM_LE_KEY_PENC|BTM_LE_KEY_PID;
   p_event_data->pairing_io_capabilities_ble_request.oob_data = BTM_OOB_NONE;
   ```

   c. In the `BTM_SECURITY_REQUEST_EVT` management case grant the authorization to the central by using the following code:

   ```
   wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr,
   WICED_BT_SUCCESS );
   ```

Testing

1. Open a UART terminal and connect to the kit.

2. Program your application to your kit.

3. Open the AIROC™ Connect app and connect to your device.

4. Watch the UART messages during the next steps to see which Bluetooth® events occur.

5. Connect to the device.

6. Open the GATT browser, navigate to the LED Characteristic (the one with Read and Write Properties) and read/write the value.

   Notice the events that occur. Remember that we did NOT require authentication for this Characteristic which is reflected in the events that you will see.

7. Now navigate to the Counter Characteristic (the one with Read and Notify Properties) and read the value. When requested, accept the request to pair the devices.

   Notice the events that occur. Remember that this Characteristic requires authentication.

8. Disconnect from the AIROC™ Connect app.

9. Go to the phone's Bluetooth® settings and remove the <inits>_pair device from the paired devices list.

   Note: *This is necessary so that when you reset or re-program the kit, the phone won't have stale bonding information stored which could prevent you from re-connecting. In the next exercise we'll store bonding information on the Bluetooth® LE device so that you will be able to leave the devices paired if you desire.*

   Note: *If you are using the Android version of AIROC™ Connect, you can remove bonding information by clicking on the "Paired" button next to your device's name rather than going to the phone's settings screen.*

Exercise 2: Bonding

In this exercise, you will try out and review an application that saves bonding information to the NVRAM. This exercise has been fully implemented in the template.

User LED1 has been updated to indicate whether bonding information has been saved or not. The states are:

OFF	Not advertising
Slow Blink	Advertising, not bonded
Fast Blink	Advertising, bonded
ON	Connected

Once bonding information has been stored, it can be erased from the device by entering [**e**] in the UART terminal window. This can only be done when the device is not connected. Erasing bonding information also causes advertising to restart if the device is not already advertising.

Application Creation

1. Create a new application with the CYW920835M2EVB-01 BSP.

   On the application template page, use the **Browse** button to start from the template provided in *Templates/ch05_ex02_bond*.
   Keep the name as the default of ***ch05_ex02_bond***.

2. Open the Bluetooth® Configurator.

3. On the **GAP Settings** tab, change the device name to <inits>_bond.

4. Save your edits and close the configurator.

5. Review the code to see how bonding information is stored and retrieved using the NVRAM library.

   Note: An overview of the changes made from the prior application is provided later in this exercise.

Testing

1. Open a UART terminal and connect to the kit.

2. Program your application to your kit.

3. Open the AIROC™ Connect app and connect to your device.

4. Watch the UART messages during the next steps to see which Bluetooth® events occur.

5. Connect to the device.

6. Connect to your device, open the GATT browser, click on the Service, and then on the Counter Characteristic. Click "Read".

7. If requested, accept the invitation to pair the devices.

8. Note down the Stack events that occur during pairing. This information is displayed in the UART.

9. Disconnect from the device. Do <u>NOT</u> remove the device from the phone's list of paired devices this time.

Note: *You will notice that the LED is blinking at 5 Hz. The firmware was written to do this when it is not connected but has bonding information stored.*

☐ 10. Re-scan and find your device in the list.

☐ 11. Re-connect to your device and read the Counter Characteristic.

☐ 12. Once again note down the Stack events that occur during pairing. You will notice that fewer steps are required this time.

☐ 13. Disconnect again.

☐ 14. Reset or power cycle the board.

Note: *If you power cycle the board, you will need to either reset or re-open the UART terminal window.*

☐ 15. Start a scan, find your device in the list, connect to your device for a third time and then read the Counter Characteristic.

☐ 16. Note down the Stack events that occur this time during pairing. Compare to the previous two connections.

☐ 17. Disconnect again.

☐ 18. Remove the device from the list of bonded devices in the Phone's Bluetooth® settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

☐ 19. Start a scan and find your device.

☐ 20. Connect to your device and try to read the Counter Characteristic.

Note: *Pairing will not complete because AIROC™ Connect no longer has the required keys to use. You will not be able to read the Counter value because it requires an authenticated connection.*

Note: *If you look in the UART window you will see a message about the security request being denied.*

☐ 21. Disconnect from the device. Wait until the pairing request times out so that advertising restarts or just reset the kit.

☐ 22. Press [**e**] in the UART window to erase bonding information and reset the kit.

Note that `CYBSP_USER_LED` begins blinking at 2 Hz. This indicates that the bonding information has been cleared from the device and it will now allow a new connection.

☐ 23. Scan, Connect, and attempt to read the Counter Characteristic again. Allow pairing if requested. This time it should work.

☐ 24. Note the steps that the firmware goes through this time.

☐ 25. Disconnect a final time and remove the device from the phone's paired Bluetooth® devices so that the saved boding information won't interfere with any future tests.

Note: *You should clear the bonding information anytime you are going to reprogram the kit or otherwise clear bonding information since the Bluetooth® LE device will no longer have the bonding information on its side.*

Overview of Changes

- There are a lot of UART messages printed in this example for learning purposes. In a real application, most if not all these messages would be removed. For example, the keys are printed at powerup. That would never be done in a real application.

- A global variable called bond_mode is created that can either be set to BONDED or BONDING to indicate the device's bonding status.

- The status LED is updated to indicate connection status. The LED operates as follows:

Advertising?	Connected?	Bonded?	LED
No	No	N/A	OFF
No	Yes	N/A	ON
Yes	No	No	Blinking at 1 Hz
Yes	No	Yes	Blinking at 5 Hz
Yes	Yes	N/A	N/A - this case doesn't occur

- A structure called `bondinfo` is created which holds the link keys for the bonded device (which also includes the BDA of the bonded device), the value of the Counter CCCD and the identity keys (which are used for RPA). The BDA of the bonded device is used to determine when we have reconnected to the same device while the CCCD value is saved so that the state of notifications can be retained across connections for bonded devices.

- In the `BTM_ENABLED_EVENT`, before initializing the GATT database, the data is read from NVRAM and placed in the `bondinfo` structure. If the read operation returns `WICED_SUCCESS`, that means bonding information was found so the remote device is added to the address resolution database and `bond_mode` is set to `BONDED`.

- In the `BTM_SECURITY_REQUEST_EVENT` look to see if `bond_mode` is `BONDING`. Security is only granted if the device is not already bonded.

- In the Stack event `BTM_PAIRING_COMPLETE_EVT` if paring was successful change the `bond_mode` to `BONDED`.

- In the Stack event `BTM_ENCRYPTION_STATUS_EVT`, if the device is bonded (i.e. `bond_mode` is `BONDED`), copy the CCCD information from the `bondinfo` structure into the GATT database. This is done so that when you reconnect to a device that was previously bonded, the CCCD will be set to the same value it was when you were last connected.

- In the Stack event `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT`, save the keys for the peer device to the `bondinfo` structure and write the bonding information to the NVRAM.

- In the Stack event `BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`, provide the keys that are in the `bondinfo` structure if `bond_mode` is `BONDED`. If no keys have been stored, return `WICED_BT_ERROR` so the stack will generate new keys.

- In the Stack event `BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT`, save the identity keys used for RPA to NVRAM.

- In the Stack event `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT`, read the identity keys used for RPA from NVRAM. If no keys have been stored, return `WICED_BT_ERROR` so the stack will generate new keys.

- In the GATT write handler function, save the Counter CCCD value to the `bondinfo` structure whenever it is updated and write the value into NVRAM.

- The UART is configured to accept input with a receive callback. The rx_cback function looks for the key [**e**]. If it has been pressed, it sets `bond_mode` to `BONDING`, removes the bonded device from the list of bonded devices, removes the device from the address resolution database, and clears out the bonding information stored in `bondinfo` and in the NVRAM.

Questions

Here are a few questions to answer to make sure you understand how the application works.

☐   1.   What items are stored in NVRAM?

☐   2.   Which event stores each piece of information?

☐   3.   Which event retrieves each piece of information?

Exercise 3: Passkey

In this exercise, you will modify the previous exercise to require a passkey to be entered to pair the devices the first time. The passkey will be randomly generated by the stack and will displayed by the UART. The passkey will need to be entered in your Phone's Bluetooth® connection settings before Pairing/Bonding will be allowed.

Application Creation

☐ 1. Create a new application with the CYW920835M2EVB-01 BSP.

☐   On the application template page, use the **Browse** button to start from the completed application for the previous exercise, *ch05_ex02_bond*. If you did not complete that exercise, you can use the template for it from *Templates/ch05_ex02_bond*.

   2. Name the new application **ch05_ex03_passkey**.

☐ 3. Open the Bluetooth Configurator.

   a. Change the **Device Name** to <inits>_passkey

   b. Save the edits and close the configurator.

☐ 4. Open *app.c* and change the `pairing_io_capabilities_ble_request.local_iop_cap` from `BTM_IO_CAPABILITIES_NONE` to `BTM_IO_CAPABILITIES_DISPLAY_ONLY`.

☐ 5. Create a new event case statement for `BTM_PASSKEY_NOTIFICATION_EVT` in `app_bt_management_callback` to print the passkey value provided by the stack to the UART. Be sure to set `result = WICED_BT_SUCCESS` inside the case.

   Note: `BTM_PASSKEY_NOTIFICATION_EVT` *is not in the template code.*

   Note: *Make sure you print something such as a string of asterisks around the value so that it is easy to find in the terminal window.*

   Note: *The passkey is 6 digits so print leading 0's if the value is less than 6 digits. (i.e. use `%06d`).*

   Note: *The passkey is passed to the callback event as:*
   `p_event_data->user_passkey_notification.passkey`

Testing

☐ 1. Open a UART terminal window.

☐ 2. Build the application and program your kit.

☐ 3. Open the AIROC™ Connect app.

☐ 4. Attempt to connect to the device and then navigate down to the Counter Characteristic in the GATT browser.

      You will see a notification from the Bluetooth system asking for the Passkey to be entered. Find the Passkey on the UART terminal window and enter it into the device.

☐ 5. Once Pairing and Bonding completes, verify that the application still works.

☐   6.   Disconnect, reconnect and read the Counter Characteristic again.

       Observe that the key does not need to be entered to Pair this time.

☐   7.   Disconnect, then remove the device from the phone's remembered Bluetooth device settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

☐   8.   Press [**e**] in the UART terminal to put the kit into Bonding mode (i.e. erase the stored bonding information) and then reconnect.

☐   9.   Connect again and read the Counter Characteristic. Observe that the key must be entered again.

☐   10.   Disconnect again and remove the bonding information from the phone's Bluetooth settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

Questions

Here are some questions to answer to test your understanding.

☐   1.   Other than `BTM_IO_CAPABILITIES_NONE` and `BTM_IO_CAPABILITIES_DISPLAY_ONLY`, what other choices are available? What do they mean?

☐   2.   What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

Exercise 4: Numeric comparison

In this exercise, you will modify the previous exercise to require the user to compare a 6-digit number on both devices to pair the first time. After comparing that both numbers are the same, the user needs to click **Yes** in in the Phone's Bluetooth connection settings and type [**y**] in the UART terminal window for the kit before Pairing/Bonding will bfe allowed.

Note: *The code to implement the passkey will <u>not</u> be removed since it is up to the central to decide which method to use. That is, we will set the device capabilities to*
`BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`. Since the device has both a display and Yes/No input, the central can choose to use either passkey or numeric comparison.

Application Creation

☐   1.   Create a new application with the CYW920835M2EVB-01 BSP.

On the application template page, use the **Browse** button to start from the completed application for the previous exercise *ch05_ex03_passkey*. If you did not complete that exercise, the solution can be found in *Projects/key_ch05_ex03_passkey*.
Name the new application ***ch05_ex04_numeric***.

☐   2.   Open the Bluetooth Configurator.

a.   Change the **Device Name** to <inits>_numeric

b.   Save the edits and close the configurator.

☐   3.   Open *app.c* and change the `pairing_io_capabilities_ble_request.local_iop_cap` to `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`.

☐   4.   Add a global variable of type `wiced_bt_device_address_t` to hold the remote BDA. This address will be needed once the user confirms the connection request.

☐   5.   Create a new event case statement for `BTM_USER_CONFIRMATION_REQUEST_EVT` in `app_bt_management_callback` to do the following:
a.   Print the value of the numeric code that is provided by the stack.
b.   Store the remote BDA to the temporary variable so that it is available when the response is sent back.
c.   Set `result = WICED_BT_SUCCESS`.

Note: *`BTM_USER_CONFIRMATION_REQUEST_EVT` is not in the template code.*

Note: *Make sure you print something such as a string of asterisks around the value so that it is easy to find in the terminal window.*

Note: *The numeric code is 6 digits so print leading 0's if the value is less than 6 digits. (i.e. use %06d).*

Note: *The numeric code is passed to the callback event as:*
`p_event_data->user_confirmation_request.numeric_value`

Note: *The remote_bda is an array so you must use `memcpy` to copy it to the temporary variable.*

Note: *The remote_bda is passed to the callback event as:*
`p_event_data->user_confirmation_request.bd_addr`

☐   6.   In the UART task, look for either a 'y' or a 'n' character and send the appropriate response back to the stack.

           If the response is 'y' send `WICED_BT_SUCCESS` and if the response is 'n' send `WICED_BT_ERROR`. You must provide the BDA of the remote device which was captured in the `BTM_USER_CONFIRMATION_REQUEST_EVT`

```
wiced_bt_dev_confirm_req_reply( WICED_BT_SUCCESS, tempRemoteBDA);
c;
```

Testing

☐   1.   Open a UART window.

☐   2.   Build the application and program your kit.

☐   3.   Open the AIROC™ Connect app.

☐   4.   Connect and then read the Counter Characteristic to initiate pairing.

           You will see a notification on the phone asking for you to verify the number printed by both devices is the same. Find the number on the UART terminal window. If the values match, click **Yes** on the phone, and press [**y**] in the UART.

☐   5.   Once Pairing and Bonding completes, verify that the application still works.

☐   6.   Disconnect, reconnect and read the Counter Characteristic.

☐   7.   Observe that the number does not need to be verified to pair this time.

☐   8.   Disconnect, then remove the device from the phone's remembered Bluetooth® device settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

☐   9.   Enter [**e**] in the UART window to put the kit into Bonding mode and then reconnect.

☐   10.   Connect again and read the Counter Characteristic. Observe that the numeric comparison must be done again to connect.

☐   11.   Disconnect again and remove the device from the phone's remembered Bluetooth® device settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

Exercise 5: Store Bonding Information for Multiple Devices

In this exercise, you will review and test an application that has all the features of the previous exercise, but it will allow up to 4 devices to be bonded. Note that while this application stores bonding information for multiple devices, it does NOT allow multiple simultaneous Bluetooth LE connections. That is also possible, but it's not demonstrated here.

Each new device that is bonded has its information stored in a separate NVRAM location. When a connection is made the device will search to see if the device is one of the stored devices.

When the application starts, it will be in bonding mode (just like the earlier applications). Once at least one device is bonded, the [**e**] key in the UART is used to toggle between bonded mode and bonding mode. You must enter bonding mode to bond a new device and you must be in bonded mode to connect to an existing device. If you have an existing device and delete its bonding information from the client, you can re-bond to it by entering bonding mode and then connecting to the device. This will replace the existing bonding entry for that device. When you change the mode advertising will be restarted if it is currently stopped.

If you try to bond a new device and all of the available slots are already taken, the bonding information for the oldest device will be removed and then replaced with the new device.

The [**l**] key in the UART can be used to print information on the number of bonded devices, the next NVRAM location that will be used to store bonding information, and the list of BDAs for the bonded devices.

Application Creation

1. Create a new application with the CYW920835M2EVB-01 BSP.

   On the application template page, use the **Browse** button to start from the template provided in *Templates/ch05_ex06_multi*.
   Keep the name as the default of ***ch05_ex06_multi***.

2. Open the Bluetooth® Configurator.

3. On the **GAP Settings tab**, change the device name to <inits>_multi.

4. Save your edits and close the configurator.

5. Review the code to see how bonding information is stored and retrieved using the NVRAM library.

   Note: An overview of the changes made from the prior application is provided later in this exercise.

Testing

1. Open a UART terminal and connect to the kit.

2. Program your application to your kit.

3. Open the AIROC™ Connect mobile app.

4. Scan for your device and connect to it.

5. Open the GATT browser, go to the Counter Characteristic, and click Read. Complete pairing by entering [**y**] in the UART and accepting pairing on the phone.

☐   6.   Test the functionality of the application to make sure it still works the same way.

☐   7.   Disconnect from the device.

☐   8.   Press the [**l**] key in the UART window to see that one device is bonded.

☐   9.   Press the [**e**] key in the UART window to put the device into bonding mode so that a second device can be bonded.

☐   10.   Use a different phone to open AIROC™ Connect.

☐   11.   Scan for your device and connect to it.

☐   12.   Open the GATT browser, go to the Counter Characteristic, and click Read. Complete pairing by entering [**y**] in the UART and accepting pairing on the phone.

☐   13.   Test the functionality of the application to make sure it still works the same way.

☐   14.   Disconnect from the phone.

☐   15.   Press the [**l**] key in the UART window to see that two devices are now bonded.

☐   16.   Scan and connect again from both AIROC™ Connect clients to verify that paring occurs without requiring you to do the numeric comparison. Verify that the functionality still works.

☐   17.   Disconnect and then remove bonding information from AIROC™ Connect on one of the phones.

☐   18.   Connect and then attempt to pair from the device and see that it fails because the device is not in bonding mode and the client does not have the bonding information available. Disconnect again.

☐   19.   Wait for advertising to restart (the bonding request must timeout first) and then press the [**e**] key in the UART window to put the device into bonding mode so that the prior device can be re-bonded.

☐   20.   Reconnect and pair. Notice that you are now required to do numeric comparison again since the bonding information is no longer available on the client.

☐   21.   Disconnect again and press [**l**] in the UART. Notice that the bonding information has been updated in the existing NVRAM slot rather than bonding a new device.

☐   22.   Disconnect again. Connect and pair additional devices if you desire to see them stored in unique locations. If you bond more than 4 devices, the oldest bonded device will be replaced.

☐   23.   Disconnect again and clear the bonding information from all devices.

Overview of Changes

- The RPA timeout is set to the default value `WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_CHANGE_TIMEOUT` which results in a new address every 900 seconds.

- Add a `#define` for `BOND_MAX` which is the maximum number of devices that can be bonded at a time (default is set to 4).

- Add a `#define` for `NONE` to use when a device does not have bonding information stored yet.

- Add a new NVRAM location (`VSID_MULTI_BOND_INFO`) and a structure in RAM (`multi_bond`) to hold the number of currently bonded devices and the next slot that will be used to store new bonding information.

- Add a variable (`current_slot`) to remember which device from the array of saved bonded devices is currently connected.

- Add a variable (`new_device`) to keep track of whether a device being bonded is going in a new slot or is overriding an existing device's information.

- The `bondinfo` structure is updated to be an array of structures to hold the device information for each bonded device (i.e. device link keys and CCCD value).

- The UART task is updated so that it just toggles whether it is in bonding mode or not when [**e**] is pressed. This can only be done when not connected.

- The UART task is updated so that when [**l**] is pressed, it will list information for the bonded devices.

- Update the `BTM_ENABLED_EVT` to load the `multi_bond` structure from NVRAM and to print out the bonding info for all bonded devices.

- Update the `BTM_ENABLED_EVT` to add all previously bonded devices to the address resolution database.

- Update `BTM_ENCRYPTION_STATUS_EVT` to update the CCCD information in the GATT database from the bonded device's last saved information.

- Update `BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT`:

  - Search for the device in the list of bonded devices:

    - If found, set the `current_slot` variable to the location containing the connected device's information, and provide the keys for that slot to the stack.

    - If not found, return `WICED_BT_ERROR` so new keys will be generated.

- Update `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT` so that it stores the newly bonded device's encryption key information into the correct array location in NVRAM. It also increments the number of bonded devices and the next slot location to use if this is a new device being bonded.

- Update the GATT `app_bt_write_handler` function so that it stores the bonded device information whenever the CCCD is written. The entire bondinfo structure for the current slot is written since the NVRAM API requires that you write to an entire VSID.

5.6 Appendix

Answers to the questions asked in the exercises above are provided here.

5.6.1 Exercise 2 answers

1. What items are stored in NVRAM?

    CCCD value (`bondinfo.cccd`)
    Security keys for the bonded device (`bondinfo.link_keys`)

2. Which event stores each piece of information?

    The CCCD value is stored in the GATT write handler whenever a new CCCD value is written.
    The link keys are stored by the `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT` event.

3. Which event retrieves each piece of information?

    All EEPROM data is read at powerup and put into the `bondinfo` structure. Newly generated values are stored in the structure and written to EEPROM as described above.

5.6.2 Exercise 3 answers

1. Other than `BTM_IO_CAPABILITIES_NONE` and `BTM_IO_CAPABILITIES_DISPLAY_ONLY`, what other choices are available? What do they mean?

    `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`
    The device can display values and can accept a Yes/No input
    `BTM_IO_CAPABILITIES_KEYBOARD_ONLY`
    The device cannot display values, but has a way for the user to enter numeric values
    `BTM_IO_CAPABILITIES_BLE_DISPLAY_AND_KEYBOARD_INPUT`
    The device can display values and has a way for the user to enter numeric values

2. What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

    `BTM_PASSKEY_NOTIFICATION_EVT`

    This event is called when the Stack has generated a 6-digit value that needs to be displayed to the user.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.