

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

5.1	Security	2
5.1.1	Pairing.....	3
5.1.2	Bonding	5
5.1.3	Pairing & Bonding process summary	5
5.1.4	Authentication, Authorization and the GATT database.....	5
5.2	Privacy.....	6
5.3	Bluetooth® configurator settings for security and privacy.....	8
5.3.1	Security.....	8
5.3.2	Privacy	9
5.4	Firmware for security and privacy	10
5.4.1	Pairing.....	10
5.4.2	Pairing + Bonding.....	12
5.5	Exercises	16
Exercise 1:	Paring.....	16
Exercise 2:	Bonding.....	20
Exercise 3:	Passkey	25
Exercise 4:	Numeric comparison.....	27
Exercise 5:	Privacy with Resolvable Private Address (RPA).....	29
Exercise 6:	Store Bonding Information for Multiple Devices.....	32

Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	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).

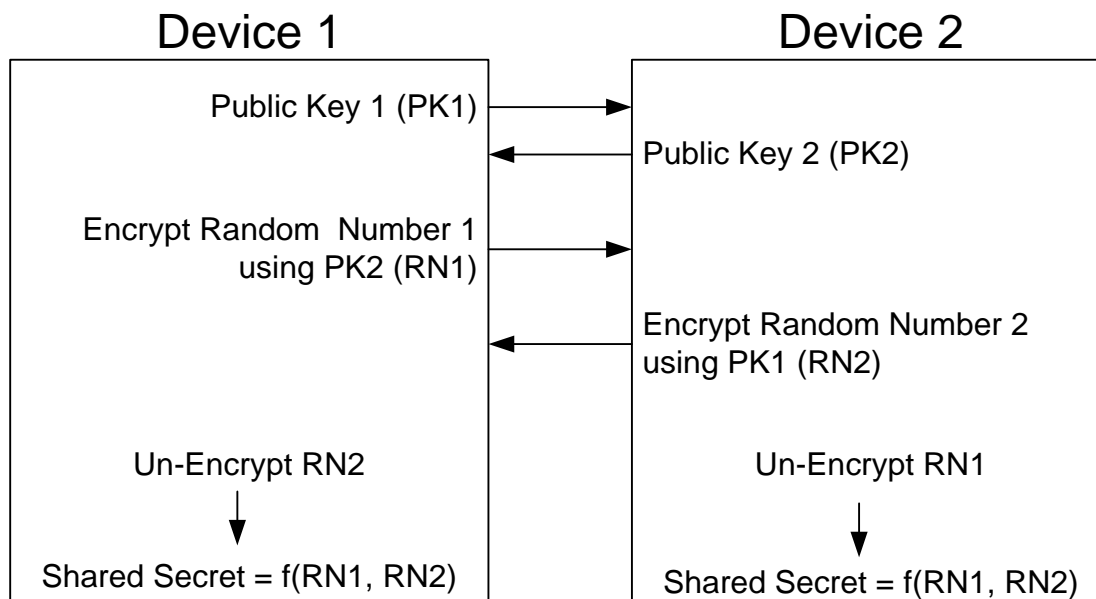
- $CypherText = F(SharedSecret, PlainText)$

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

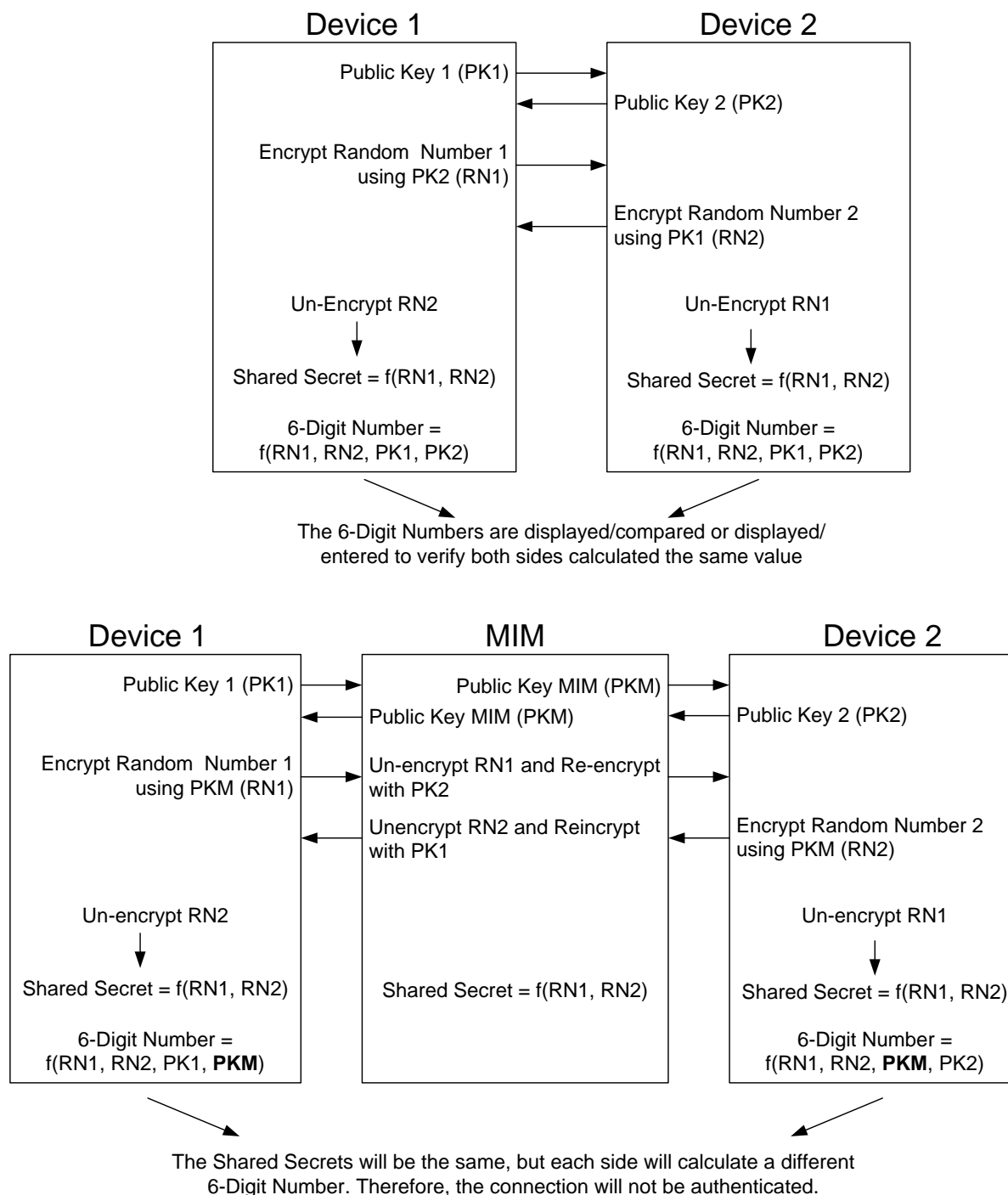
Pairing is the process of arriving at the shared secret. The basic problem continues to be how do you send a shared secret over the air, unencrypted and still have your shared secret stay a secret? The answer is that you use public key encryption. Both sides have a public/private key pair that is either 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.

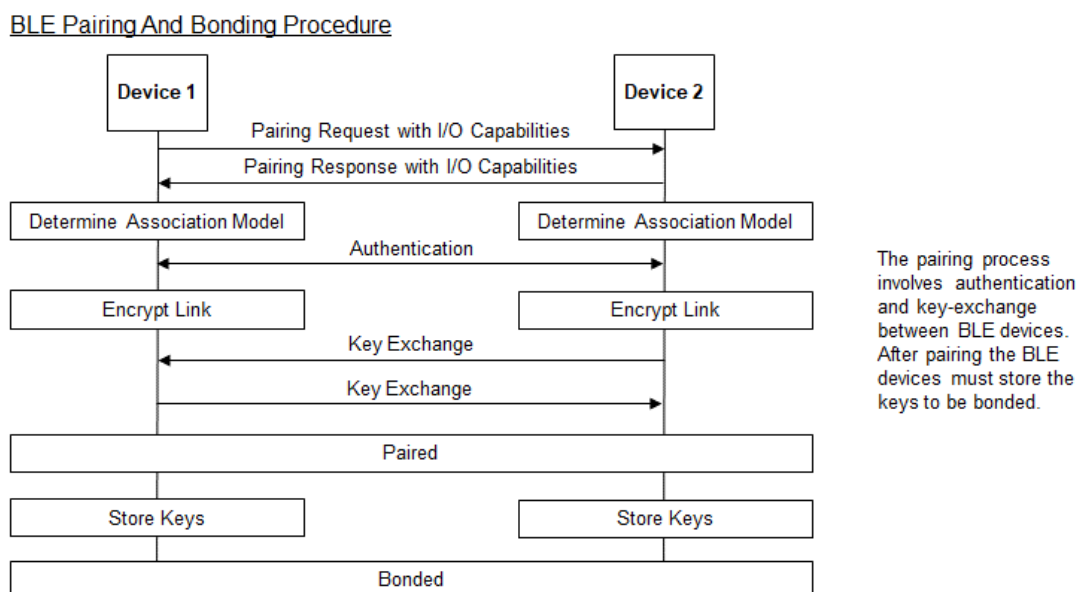


5.1.2 Bonding

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

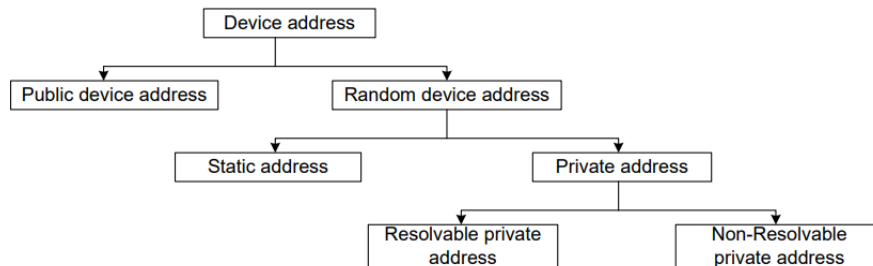
In prior chapters we talked about Attributes and the GATT Database. Each Attribute has a permissions bit field that includes bits for Encryption, Authentication, and Authorization. The stack will guarantee that you will not be able to access an Attribute that is marked for Encryption or Authentication 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. In ModusToolbox, 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 Bluetooth® configurator settings for security and privacy

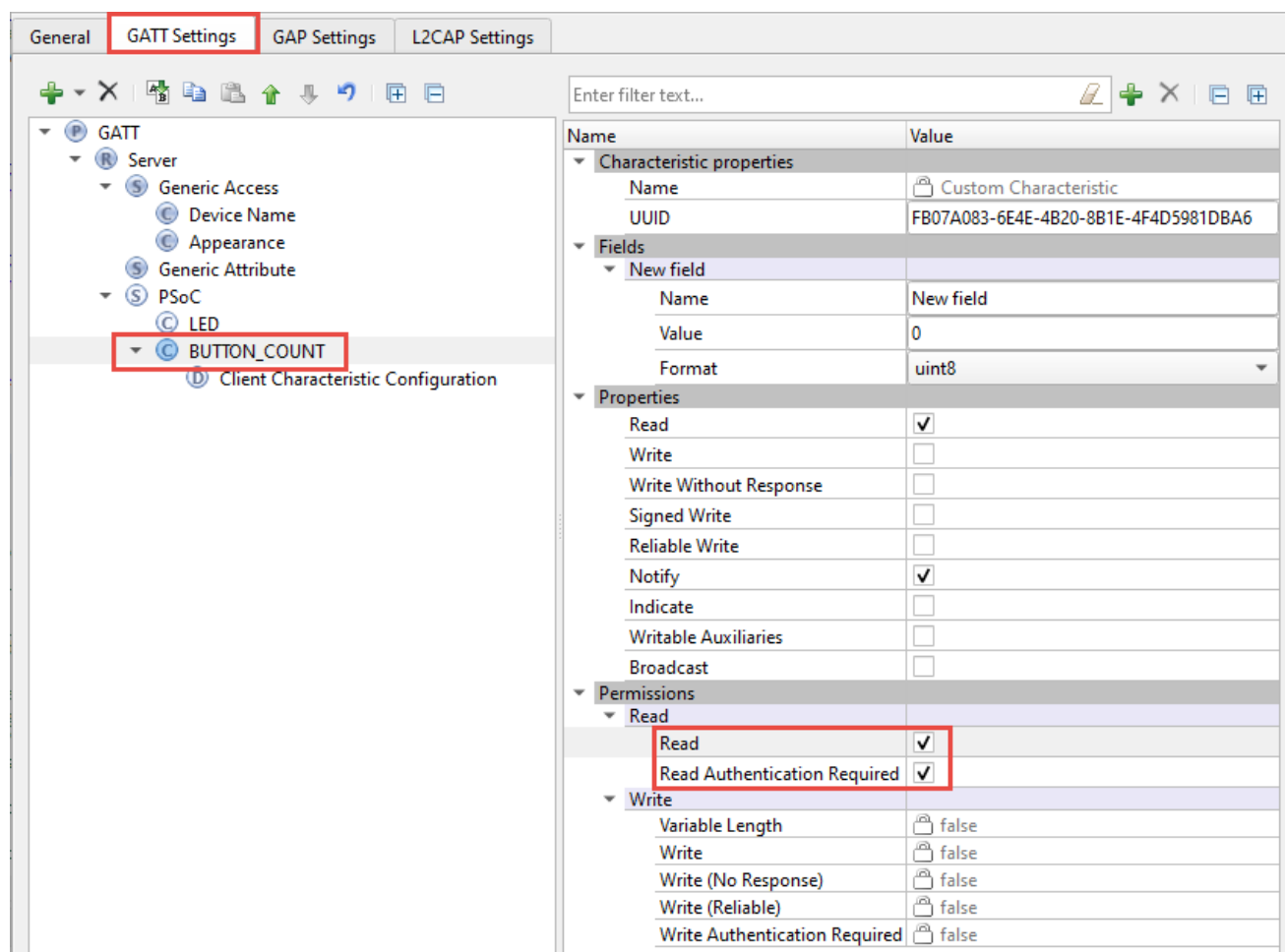
The ModusToolbox™ Bluetooth® configurator makes it simple to set up security and privacy. Examples of each are shown below.

5.3.1 Security

In order to enable security (i.e. to require pairing before allowing the client read or write of a characteristic) you just click the "Read Authentication Required" or "Write Authentication Required" button 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).

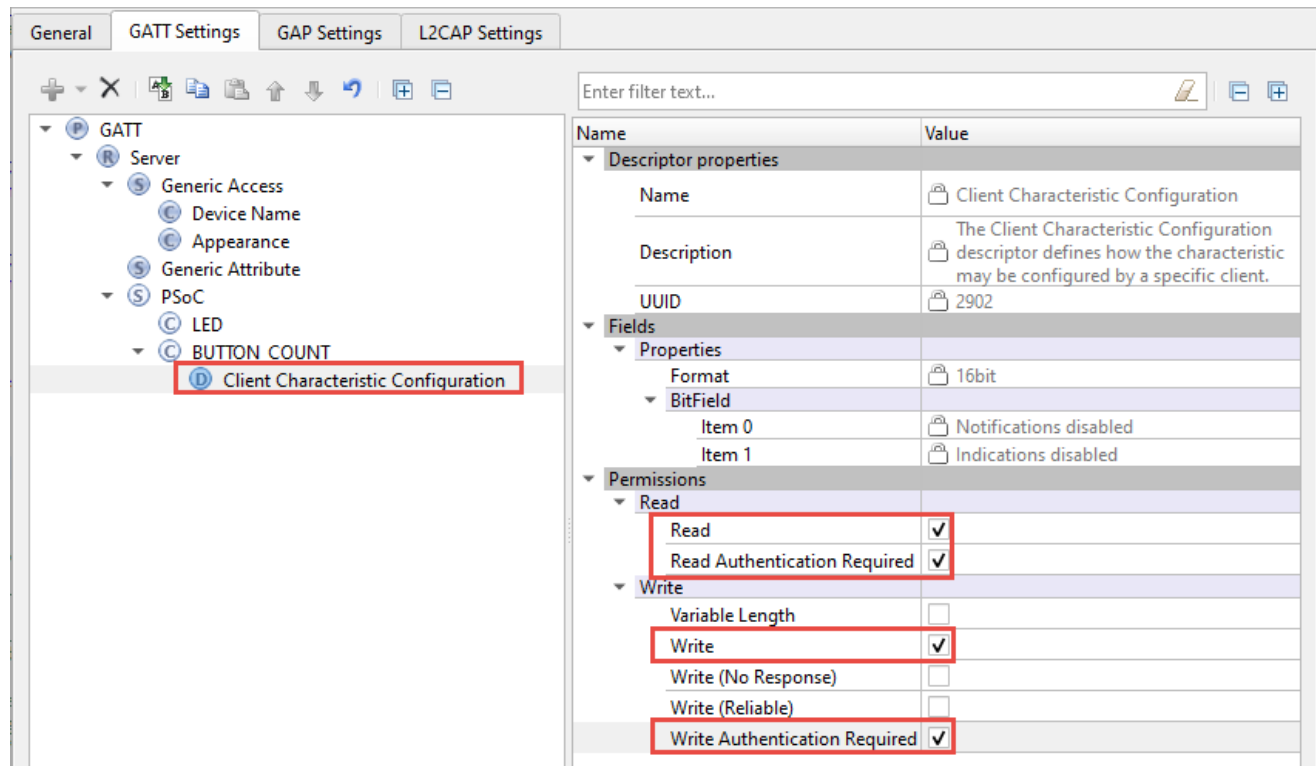
Note: The security settings for each Characteristic and its Descriptors are independent so make sure you set the permissions for everything the way you want them.

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



The screenshot shows the ModusToolbox Bluetooth configurator interface. The 'GATT Settings' tab is selected. In the left-hand tree view, the 'BUTTON_COUNT' characteristic is highlighted. The right-hand pane displays the configuration for this characteristic. The 'Permissions' section is expanded, and the 'Read' sub-section is also expanded. The 'Read Authentication Required' checkbox is checked, indicating that security is enabled for reading this characteristic. Other permissions like 'Read', 'Write', and 'Write Without Response' are also visible.

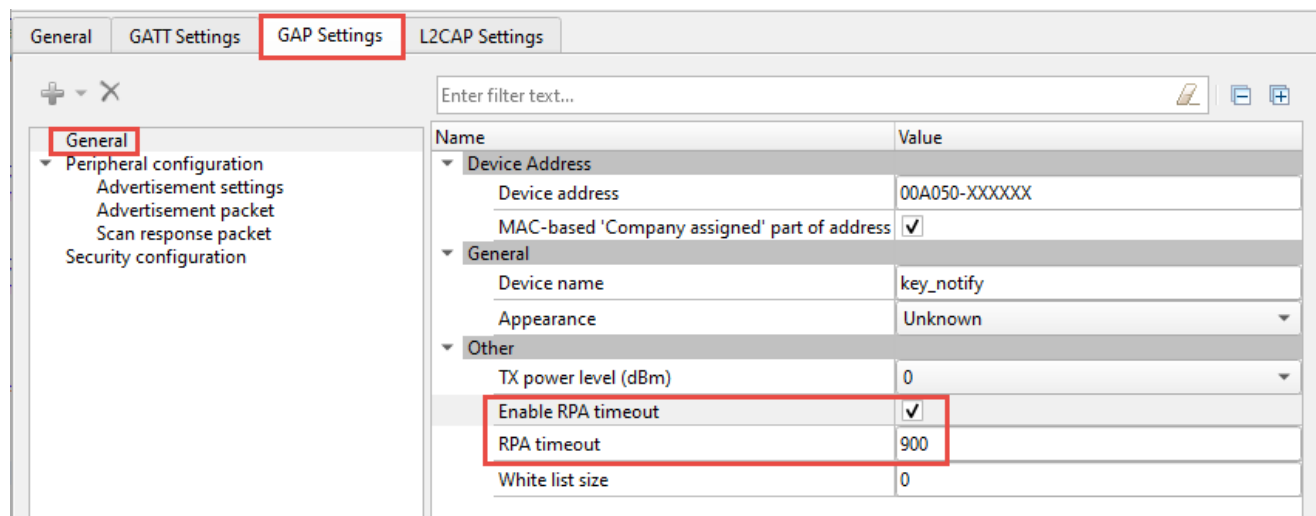
Name	Value
Characteristic properties	
Name	Custom Characteristic
UUID	FB07A083-6E4E-4B20-8B1E-4F4D5981DBA6
Fields	
New field	
Name	New field
Value	0
Format	uint8
Properties	
Read	<input checked="" type="checkbox"/>
Write	<input type="checkbox"/>
Write Without Response	<input type="checkbox"/>
Signed Write	<input type="checkbox"/>
Reliable Write	<input type="checkbox"/>
Notify	<input checked="" type="checkbox"/>
Indicate	<input type="checkbox"/>
Writable Auxiliaries	<input type="checkbox"/>
Broadcast	<input type="checkbox"/>
Permissions	
Read	
Read	<input checked="" type="checkbox"/>
Read Authentication Required	<input checked="" type="checkbox"/>
Write	
Variable Length	false
Write	false
Write (No Response)	false
Write (Reliable)	false
Write Authentication Required	false



Name	Value
Descriptor properties	
Name	Client Characteristic Configuration
Description	The Client Characteristic Configuration descriptor defines how the characteristic may be configured by a specific client.
UUID	2902
Fields	
Properties	
Format	16bit
BitField	
Item 0	Notifications disabled
Item 1	Indications disabled
Permissions	
Read	
Read	<input checked="" type="checkbox"/>
Read Authentication Required	<input checked="" type="checkbox"/>
Write	
Variable Length	<input type="checkbox"/>
Write	<input checked="" type="checkbox"/>
Write (No Response)	<input type="checkbox"/>
Write (Reliable)	<input type="checkbox"/>
Write Authentication Required	<input checked="" type="checkbox"/>

5.3.2 Privacy

A Resolvable Private Address (RPA) can be enabled for a device simply by checking the **Enable RPA timeout** box in the configurator's **GAP Settings** tab. Once the box is checked, there is also an **RPA timeout** value that can be changed if desired. The default is 900 seconds (i.e. 15 minutes).



Name	Value
Device Address	
Device address	00A050-XXXXXX
MAC-based 'Company assigned' part of address	<input checked="" type="checkbox"/>
General	
Device name	key_notify
Appearance	Unknown
Other	
TX power level (dBm)	0
Enable RPA timeout	<input checked="" type="checkbox"/>
RPA timeout	900
White list size	0

5.4 Firmware for security and privacy

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

5.4.1 Pairing

For a typical application that connects using a Paired link but does NOT use privacy, does NOT store bonding information in EEPROM, and does NOT 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. Since we are not implementing privacy yet, just return WICED_BT_SUCCESS so that the stack doesn't generate the privacy keys for no reason.
	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.
Connect	BTM_PAIRING_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 EEPROM so this state just returns WICED_BT_ERROR to tell the stack to generate new keys.
	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	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.
	BTM_PAIRING_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 EEPROM so this state just returns WICED_BT_ERROR.

Activity	Callback Event Name (both Stack and GATT)	Reason
Pair (if secure link is required)	BTM_SECURITY_REQUEST_EVT	The occurs when the client requests a secure connection. When this event happens, you need to call <code>wiced_bt_ble_security_grant</code> 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_ENCRYPTION_STATUS_EVT	This occurs when the secure link has been established.
	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to EEPROM. Again, in this case, we have not saved any keys in EEPROM so this state just returns <code>WICED_BT_ERROR</code> .
	BTM_PAIRING_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.
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 <code>GATT_CONNECTION_STATUS_EVENT</code> (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).

5.4.2 Pairing + Bonding

If bonding information is stored to EEPROM, 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
1 st Powerup	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT	When this event occurs, if privacy is enabled the firmware needs to load the privacy keys from EEPROM, provide them to the stack and return <code>WICED_BT_SUCCESS</code> . If keys have not been previously saved for the device then this state must return a value other than <code>WICED_BT_SUCCESS</code> such as <code>WICED_BT_ERROR</code> . The non-success return value causes the stack to generate new privacy keys. If privacy is not enabled, this state can just return <code>WICED_BT_SUCCESS</code> .
	BTM_ENABLED_EVT	<p>This occurs once the Bluetooth LE stack has completed initialization. Typically, you will start up the rest of your application here.</p> <p>During this event, the firmware needs to load keys (which also includes the <code>BD_ADDR</code>) for a previously bonded device from EEPROM and then call <code>wiced_bt_dev_add_device_to_address_resolution_db</code> to allow connecting to a bonded device. If a device has not been previously bonded, this will return values of all 0.</p>
	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 <code>BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT</code> was not <code>WICED_BT_SUCCESS</code>). During this event, the privacy keys must be saved to EEPROM.
	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
1 st Connect	BTM_PAIRING_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 EEPROM 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.
1 st Pair	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to EEPROM. Again, in this case, we have not saved any keys in EEPROM 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).

Activity	Callback Event Name	Reason
	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_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to EEPROM. Since this is the first pairing, we have not saved any keys in EEPROM yet so this state just returns WICED_BT_ERROR.
	BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT	During this event, the firmware needs to store the keys of the paired device (including the BD_ADDR) into EEPROM 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_PAIRING_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 EEPROM so this state returns WICED_BT_SUCCESS.
	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.

Activity	Callback Event Name	Reason
Re-Pair	BTM_ENCRYPTION_STATUS_EVT	<p>In this state, the firmware reads the state of the server from EEPROM. For example, the saved state of any notify settings may be read.</p> <p>Since the paired device BD_ADDR and keys were already available, no other steps are needed to complete pairing.</p>
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.
Reset	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT	Local keys are loaded from EEPROM.
	BTM_ENABLED_EVT	Stack is enabled. Paired device keys (including the BD_ADDR) are loaded from EEPROM and the device is added to the address resolution database.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising on.
Re-Connect	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	This event is called by the stack so that the firmware can load the paired device's keys from EEPROM. Since keys are available, this state must return WICED_BT_SUCCESS.
	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	<p>In this state, the firmware reads the state of the server from EEPROM. For example, the saved state of any notify settings may be read.</p> <p>Since the paired device BD_ADDR and keys were already available in EEPROM, no other steps are needed to complete pairing.</p>
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.

5.5 Exercises

Exercise 1: Paring

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

The following table shows the events that occur during this exercise. Arrows indicate the cause/effect of the stack events. New events introduced in this exercise are highlighted.

External Event	BLE Stack Event	Action
Board reset >	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT >	Not used yet.
	BTM_ENABLED_EVT >	Initialize application.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_UNDIRECTED_HIGH)	< Start advertising
CySmart will now see advertising packets		
Connect to device from CySmart >	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	Not used yet
	GATT_CONNECTION_STATUS_EVT >	Set the connection ID and enable pairing
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	
Pair >	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	Not used yet
	BTM_SECURITY_REQUEST_EVT >	Grant security
	BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT >	Capabilities are set
	BTM_ENCRYPTION_STATUS_EVT	Not used yet
	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	Not used yet
	BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT	Not used yet
	BTM_PAIRING_COMPLETE_EVT	Not used yet
Read BUTTON_COUNT characteristic >	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_READ >	Returns button count value
Read Button CCCD >	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_READ >	Returns button notification setting
Write 01:00 to Button CCCD >	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_WRITE >	Enables notifications

External Event	BLE Stack Event	Action
Press button >		Send notifications
Disconnect >	GATT_CONNECTION_STATUS_EVT >	Clear the connection ID
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_UNDIRECTED_HIGH)	Re-start advertising
Wait for timeout >	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_UNDIRECTED_LOW)	Stack switches to lower advertising rate to save power
Wait for timeout >	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	Stack stops advertising

Application Creation



1. Create a new application with the CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator 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. On the GAP Settings tab, change the device name to `<inits>_pair`.



4. On the GATT Settings tab and make the following changes:
 - a. In the BUTTON_COUNT characteristic, 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 *main.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 mobile CySmart 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 BUTTON_COUNT Characteristic (the one with Read and Notify Properties) and read the value. When requested, accept the invitation to pair the devices.

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



8. Disconnect from the mobile CySmart app.



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

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: The remaining steps use the Windows version of CySmart with a CY5677 dongle. If you do not have a CY5677 dongle and a Windows machine, skip the remaining steps.

- ☐ 10. Start the Windows CySmart app. Scan for your device. Once it appears in the list, stop scanning and connect to it.

Note: Don't forget to change the CySmart settings to increase the scan interval and scan window if you are in a congested environment.

- ☐ 11. Click on **Discover all Attributes** and then on **Enable Notifications**.

Notice that you get an authentication error because the BUTTON_COUNT notification requires authentication. If you press the button, no notification will be sent. Click **OK** to close the error dialog.

- ☐ 12. Try reading the BUTTON_COUNTER Characteristic value manually.

Notice that you again get an authentication error. Click **OK** to close the error window.

- ☐ 13. Click on **Pair**.

Click **No** if you are asked if you want to add the device to the resolving list since we haven't yet enabled privacy.

*Note: If you accidentally add the device to the resolving list, you can remove it once you disconnect. In the CySmart Device List, select the device from the list and then use the menu item **Clear > Resolving List**.*

- ☐ 14. Click on **Enable All Notifications** again.

Now when you press the button you will see the characteristic value change since notifications have been enabled.

- ☐ 15. Click on **Disable All Notifications** and then read the Button Characteristic Value manually.

It will work this time.

- ☐ 16. Click **Disconnect**.

- ☐ 17. From the Device List, click on any Device Address in the list and select **Clear > All**.

This step is necessary because we have not stored bonding information in the device yet.

Exercise 2: Bonding

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

User LED2 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 CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator 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 emulated EEPROM 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 mobile CySmart 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 BUTTON_COUNT 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 NOT 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.
- ☐ 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 CySmart 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.
- ☐ 22. Press [e] in the UART window to erase bonding information and reset the kit.

This forces it to restart advertising (it would restart advertising automatically if you waited long enough for the disallowed pairing operation to timeout).

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 bonding 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.

Note: The remaining steps use the Windows version of CySmart with a CY5677 dongle. If you do not have a CY5677 dongle and a Windows machine, skip the remaining steps.



26. Start the Windows CySmart app. Scan for your device. Once it appears in the list, stop scanning and connect to it.

Note: Don't forget to change the CySmart settings to increase the scan interval and scan window if you are in a congested environment.



27. Click on **Pair**. Note the stack events that occur during pairing.



28. Click **No** if you are asked if you want to add the device to the resolving list since we haven't yet enabled privacy.

*Note: If you accidentally add the device to the resolving list, you can remove it once you disconnect. In the CySmart Device List, select the device from the list and then use the menu item **Clear > Resolving List**.*



29. Once pairing completes, click on **Enable All Notifications**.

When you press the button, you will see the characteristic value change since notifications have been enabled.



30. Click **Disconnect**.

Note: In this case, since bonding information is saved, you should not remove it from the Device list after you disconnect so that you can reconnect and re-pair without going through the complete bonding process. You should however, remove it from the Device List list once you remove bonding information from the kit or reprogram the kit.



31. **Scan** again and the **Connect** to the device.



32. Click on **Pair**. Note that the number of stack events is greatly reduced.



33. Click **Disconnect**.



34. From the Device List, click on any Device Address in the list and select **Clear > All**.

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 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 to the UART at powerup. That would never be done in a real application.
- The emulated EEPROM library (emeeprom) is added as a dependency and its header file is included in main.c.
- The header file for the FreeRTOS Queue functions is included in main.c.
- 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 PWM operates the LED as follows:

Advertising?	Connected?	Bonded?	LED
No	No	N/A	OFF
No	Yes	N/A	ON
Yes	No	No	Blinking at 2 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) and the value of the Button CCCD. 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.
- Before initializing the stack, existing EEPROM data is loaded into the `bondinfo` structure. This gets used and updated later.
- In the `BTM_ENABLED_EVENT`, before initializing the GATT database, the link key data is examined from the `bondinfo` structure. If it is not all zeros, that means there is bonding information saved 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 bonded.
- In the Stack event `BTM_PAIRING_COMPLETE_EVT` if bonding was successful write the information from the `bondinfo` structure into the EEPROM and set bonded to `TRUE`.
 - This saves `bondinfo` upon initial pairing. This event is not called when bonded devices reconnect.
- In the Stack event `BTM_ENCRYPTION_STATUS_EVT`, if the device is bonded (i.e. `bond_mode` is `BONDED`), read bonding information from the EEPROM into the `bondinfo` structure.
 - This reads `bondinfo` upon a subsequent connection when devices were previously bonded.
- In the Stack event `BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT`, save the keys for the peer device to EEPROM.
- In the Stack event `BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT`, read the keys for the peer device from EEPROM.
- In the GATT connect event handler, for a disconnection, reset the CCCD to 0 so that if a new device is bonded it will not start out with notifications enabled.
- In the GATT write handler function, save the Button CCCD value to the `bondinfo` structure whenever it is updated and write the value into EEPROM.

- The UART is configured to accept input with a receive callback. This is done by reconfiguring the object that is created by the retarget-io library. The `rx_callback` function sends the received character to a UART task using an RTOS queue. This is done because you cannot call any Bluetooth® Stack functions from inside the ISR.
- The UART task looks for the key [e]. If it has been sent, 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 EEPROM.
- Because the UART does not function during Deep Sleep, a custom device configuration is included in `COMPONENT_CUSTOM_DESIGN_MODUS` to set the PSoC 6 **System > Power > System Idle Power Mode** to **CPU Sleep**. The *Makefile* is modified to include the custom configuration instead of the default configuration from the BSP.
- A `print_array` function is created to simplify printing out arrays of values such as keys.

Questions

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

☐

1. What items are stored in EEPROM?

☐

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 be displayed by the UART. The passkey will need to be entered in CySmart on the PC or in your Phone's Bluetooth® connection settings before Pairing/Bonding will be allowed.

Application Creation

- ☐ 1. Create a new application with the CY8CKIT-062S2-43012 BSP.
- ☐ 2. Use the Import functionality in project creator 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 `<init>_passkey`
 - b. Save the edits and close the configurator.
- ☐ 4. Open *main.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_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 CySmart mobile app.
- ☐ 4. Attempt to connect to the device and then navigate down to the button 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 button 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.
- ☐ 8. Press [**e**] in the UART terminal to put the kit into Bonding mode (i.e. erase the stored bonding information) and then reconnect.
Read the button characteristic and observe that the key must be entered again.
- ☐ 9. Disconnect again and remove the bonding information from the phone's Bluetooth settings.
- ☐ 10. Optional: Try the same thing using the Windows version of CySmart. It will pop up a window when the Passkey is needed.

*Note: Remember to put the kit into Bonding mode first by pressing [**e**] in the UART window to remove the phone's Bonding information from the kit. This is necessary since we only allow bonding information from one device to be stored in our firmware. A later exercise will fix that.*

*Note: Remember to click **No** if you are asked if you want to add the device to the resolving list since we haven't yet enabled privacy.*

Note: Remember to remove the device from the Device List once you are done with this exercise or when you otherwise remove bonding information from the device.

Questions

Here are some questions to answer to test your understanding.

- ☐ 4. Other than `BTM_IO_CAPABILITIES_NONE` and `BTM_IO_CAPABILITIES_DISPLAY_ONLY`, what other choices are available? What do they mean?
- ☐ 5. 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 CySmart on the PC or in your Phone's Bluetooth connection settings and type **[y]** in the UART terminal window for the kit before Pairing/Bonding will be allowed.

Note: The code to implement the passkey will not be removed since it is up to the central to decide which method to use. That is, we have the device capabilities set 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 CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator 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 `<init>_numeric`
 - b. Save the edits and close the configurator.



3. Open `main.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_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`
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 );  
wiced_bt_dev_confirm_req_reply( WICED_BT_ERROR, tempRemoteBDA );
```

Testing



1. Open a UART window.



2. Build the application and program your kit.



3. Open the CySmart mobile app.



4. Attempt to Connect and then Pair to the device.

You will see a notification from CySmart 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** in CySmart, and press [y] in the UART.



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



6. Disconnect and reconnect.



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.



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



10. Observe that the comparison must be done again to connect.



11. Disconnect again and clear the Device List in CySmart.



12. Optional: Try the same thing using the Windows version of CySmart. It will pop up a window when the Passkey is needed.

Note: Remember to put the kit into Bonding mode first by pressing [e] in the UART window to remove the phone's Bonding information from the kit. This is necessary since we only allow bonding information from one device to be stored in our firmware. A later exercise will fix that.

*Note: Remember to click **No** if you are asked if you want to add the device to the resolving list since we haven't yet enabled privacy.*

Note: Remember to remove the device from the Device List once you are done with this exercise or when you otherwise remove bonding information from the device.

Exercise 5: Privacy with Resolvable Private Address (RPA)

In this exercise, you will modify the previous exercise to use a Resolvable Private Address to provide link-layer privacy. Once this is done, we will have a fully functional BLE peripheral with the best possible security and privacy.

Application Creation



1. Create a new application with the CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator to start from the completed application for the previous exercise *ch05_ex04_numeric*. If you did not complete that exercise, the solution can be found in *Projects/key_ch05_ex04_numeric*.

Name the new application *ch05_ex05_rpa*.



2. Open the Bluetooth Configurator and go to the **GAP Settings** tab. In the General section:
 - a. Change the **Device Name** to `<init>_rpa`
 - b. Check the box for **Enable RPA timeout**.
 - c. Change the **RPA timeout** value to 20 seconds so that we will see frequent address changes.
 - d. Save the edits and close the configurator.



3. Open *main.c*.



4. Add new element to the `bondinfo` structure to hold the local identity keys. The entry in the structure will look like this:

```
wiced_bt_local_identity_keys_t  identity_keys;
```



5. Add a section to the EEPROM to save the local identity keys. It will look like this:

```
#define EEPROM_IDENTITY_KEYS      ((void *)&(bondinfo.identity_keys) -  
    (void *)&bondinfo)
```

6. In the Stack event `BTM_ENABLED_EVT`, find the section where the contents of the EEPROM are printed and all code to print out the local identity keys. For example:

```
printf("Identity Keys: ");  
print_array(&bondinfo.identity_keys, sizeof(bondinfo.identity_keys));
```



7. In the Stack event `BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT`, save the keys for the local device to EEPROM. For example:

```
printf( "Local Identity Key Update\n" );
memcpy(&bondinfo.identity_keys, &(p_event_data->local_identity_keys_update),
sizeof( wiced_bt_local_identity_keys_t));
eepromReturnValue = Cy_Em_EEPROM_Write(EEPROM_IDENTITY_KEYS,
&(p_event_data->local_identity_keys_update),
sizeof( wiced_bt_local_identity_keys_t), &Em_EEPROM_context);
if(CY_EM_EEPROM_SUCCESS == eepromReturnValue)
{
    printf( "Local identity Keys saved to EEPROM, result: %d:", eepromReturnValue);
    print_array(&(p_event_data->local_identity_keys_update),
sizeof( wiced_bt_local_identity_keys_t));
}
else
{
    printf("EEPROM Write Error: %d\n", eepromReturnValue);
}
result = WICED_BT_SUCCESS;
break;
```



8. In the Stack event `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT`, read the keys for the local device from EEPROM. Return `WICED_BT_ERROR` if keys were not found in the EEPROM and return `WICED_BT_SUCCESS` if they were found. For example:

```
printf( "Local Identity Key Request\n" );
/* If the key type is 0, return an error to cause the stack to generate
 * keys. The stack will call BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT
 * so the keys can be stored */
if(0 == bondinfo.identity_keys.key_type_mask)
{
    printf("New identity keys need to be generated by the stack.\n");
    result = WICED_BT_ERROR;
}
else
{
    memcpy(&(p_event_data->local_identity_keys_request),
&(bondinfo.identity_keys), sizeof(wiced_bt_local_identity_keys_t));
    printf("Identity keys are available in the database.\n");

    printf( "Local identity keys read from EEPROM: \n" );
    print_array(&bondinfo.identity_keys, sizeof( wiced_bt_local_identity_keys_t));
    result = WICED_BT_SUCCESS;
}

break;
```

Testing

Note: You can test the functionality of the application using the mobile version of CySmart, but if you want to see the address changes taking place, it is better to use the Windows version.



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



2. Program your application to your kit.



3. Open the Windows CySmart app.

- ☐ 4. Click on **Configure Master Settings** and change **Privacy 1.2 > Address generation** interval to match the timeout value from the firmware (20 s).

Note: Don't forget to also change the settings to increase the scan interval and scan window if you are in a congested environment.

- ☐ 5. Scan for your device.

- ☐ 6. Notice that the address is listed as a "Random" address and it is not the same as the Local Bluetooth Device Address that is listed in the UART during startup.

Note: If you wait more than 20 seconds, you will see an additional device show up with a different random address. If you do, you will need to connect to the new one since the device will no longer be advertising to the first address. This will happen every 20 seconds until advertising times out.

Note: If you forget which address is which, just stop and restart scanning so that only the address from the latest advertising packet will be listed.

- ☐ 7. Connect to your device, click pair, and complete pairing. Click **Yes** if you are asked if you want to add the device to the resolving list now that we have enabled privacy.

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

- ☐ 9. Disconnect from the device.

- ☐ 10. Start scanning again. Notice that the address is now listed as "Public Identity Address" and it matches the Local Bluetooth Device Address that is listed in the UART during startup.

- ☐ 11. Notice in the Device List pane there is an entry that shows random device address and Identity address that go together. Click on the **View...** button to see details of the Identity Resolving Keys.

- ☐ 12. Connect to your device and pair again. Notice that it pairs immediately without requiring the user to verify the code.

- ☐ 13. Disconnect again and clear the bonding information from the Device List (**Clear > All**).

Exercise 6: 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 EEPROM 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 bonding mode and bonded 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, if advertising has stopped, it will be restarted.

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 EEPROM 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 CY8CKIT-062S2-43012 BSP.

Use the Import functionality in project creator to start from the template provided in *Templates/ch05_ex06_multi*.

Keep the name as the default of *ch05_ex02_multi*.



2. Open the Bluetooth® Configurator.



3. On the GAP Settings tab Change the device name to <init>_multi.



4. Save your edits and close the configurator.



5. Review the code to see how bonding information is stored and retrieved using the emulated EEPROM 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 Windows CySmart app.

Note: Don't forget to change the settings to increase the scan interval and scan window if you are in a congested environment.

- ☐ 4. Scan for your device.
- ☐ 5. Connect to your device, click pair, and complete pairing. Click **Yes** if you are asked if you want to add the device to the resolving list now that we have enabled privacy.
- ☐ 6. Test the functionality of the application to make sure it still works the same way.
- ☐ 7. Disconnect from the device.
- ☐ 8. Press the [I] 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. Open the CySmart mobile app.
- ☐ 11. Scan for your device and connect to it.
- ☐ 12. Open the GATT browser, go to the BUTTON_COUNT 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 [I] key in the UART window to see that two devices are now bonded.
- ☐ 16. Scan and connect again from both CySmart clients to verify that bonding occurs without requiring you to do the numeric comparison. Verify that the functionality still works.
- ☐ 17. Disconnect and then remove bonding information from CySmart for one of the connected devices.
- ☐ 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. 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 [I] in the UART. Notice that the bonding information has been updated in the existing EEPROM 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.

Note: If you want to connect using the Windows version of CySmart from more than one computer, you must change the client's address on one of the instances so that they don't conflict. This setting can be found in **Configure Master Settings > Settings > Master Configuration > Device > Local Bluetooth Device Address > Public Address**. You can use any address as long as it doesn't conflict with an existing bonded device.

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

Overview of Changes

- The RPA timeout is set to the default value of 900 s.
- A `#define` is added for `BOND_MAX` which is the maximum number of devices that can be bonded at a time (default is set to 4).
- A `#define` is added for `DEVICE_NOT_FOUND` to use when a device does not have bonding information stored yet.
- The `bondinfo` structure is updated to have one entry for the number of devices currently bonded, one entry for the next location (slot) to use for new bonding information, one entry for local identity keys, and arrays to hold the device information for each bonded device (i.e. device link keys and CCCD value).
- A global variable `current_slot` is added to keep track of which device from the saved bonded devices list is currently connected.
- 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 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_PAIRING_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, set `new_device` to `WICED_FALSE`.
 - If not found, set the `current_slot` variable to the value of `next_slot` from the `bondinfo` structure and set `new_device` to `WICED_TRUE`.
 - If we are already at the max number of bonded devices and a new device is being added, remove the oldest device from the address resolution database and the bonded device list.
 - If in bonding mode:
 - Return `WICED_BT_ERROR` which causes the stack to generate new keys.
 - If not in bonding mode, return `WICED_BT_SUCCESS` so new keys will not be generated.
- Update `BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT` so that it stores the newly bonded device's encryption key information into the correct array location in EEPROM. It also increments the number of bonded devices and the next slot location to use if `new_device` is `WICED_TRUE`.
- Update GATT connection event so that it stores the remote BDA in the correct slot location.
- Update the GATT `app_bt_write_handler` function so that it stores any changes to the CCCD value to the correct slot location.

Trademarks

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

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2021 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.