Bluetooth Security Features Labmanual – 90 min

Using GATT permissions and different pairing processes in practice.

Introduction

This lab demonstrates the practical usage of GATT permissions and pairing processes so application developers can use the security features that Bluetooth provides.

KEY FEATURES

- Demonstrates the usage of GATT permission
- Demonstrates the different pairing processes
- Provides code examples for GATT and security manager settings

GATT permissions

In GATT database different permissions can be granted for the characteristics and descriptors. The lab demonstrates what happens if users read or write data with different permissions and how the permission can be granted via pairing processes.

Pairing Processes

This lab also demonstrates the different pairing processes and provides code snippets for security manager configurations.

Demo Application

This lab requires the *security demo application* firmware and the *Blue Gecko* app. The demo firmware uses different pairing methods and shows the used security manager settings on the WSTKs LCD. The *Blue Gecko* app is required to read and write the GATT of the demo firmware.

.

Hands-on

Preparation

1. Flash the *security demo application* firmware to the WSTK. The binary can be found in the shared training folder next to the project source.

   The firmware is supporting the BRD4153A Mighty Gecko radio board.

2. Install the *Blue Gecko* app to your phone. Alternatively, you can use *LightBlue* or *nRF Connect*.

Exercise 1 - Browsing the GATT in case of just works pairing used

Goals:

- Learn how to use the Blue Gecko app for browsing the GATT
- Understand the GATT permissions
- Try out just works pairing

1. Select the ***just works*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

just works security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
flags                 = 0x08, //Bonding requests need to be confirmed
io_capability         = sm_io_capability_noinputnooutput
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
bondable              = 1
```

You can find more about the API commands and events at Appendix – Security manager chapter

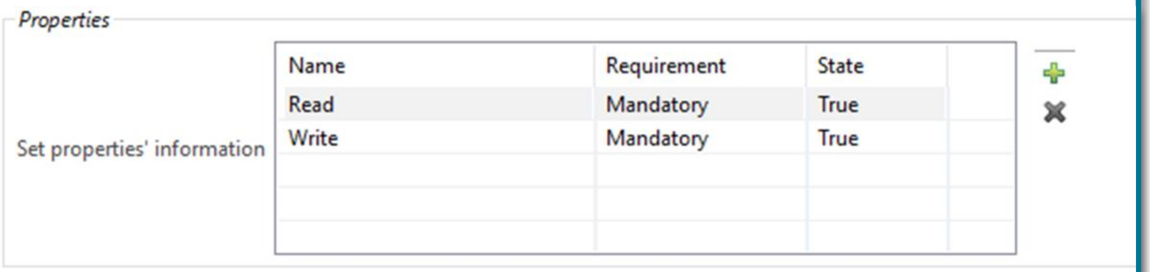
2. Start the *Blue Gecko* app and connect it to the WSTK via Bluetooth. The demo application advertise itself as *Security Example*

3. Open the GATT browser and select *Unknown service – a5559637-bbb5-4eaf-8625-45aa0cd892eb*. You must see the following characteristic.

• Public Data	ef9af9bc-6b2d-46c2-b3c6-c70ee33a47d1
• Encrypted Data	dfa037d5-cd4b-4a95-b00f-ed8693d0483e
• Authenticated Data	35b1c7e4-e062-405a-a9d2-adb9cd562d56

GATT definition

The *Public Data* characteristic has the following properties:



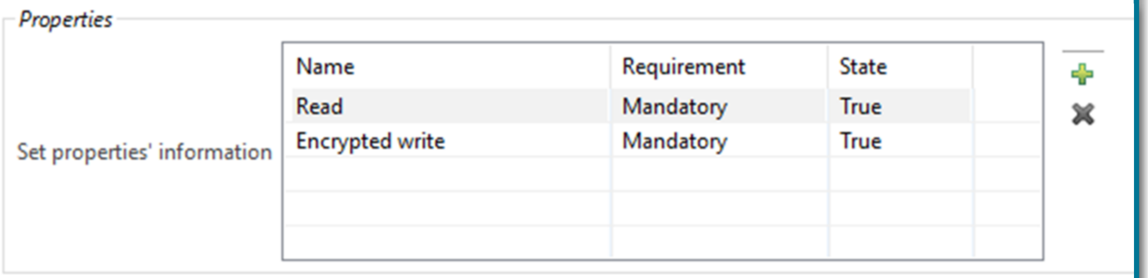
Properties

Set properties' information

Name	Requirement	State
Read	Mandatory	True
Write	Mandatory	True

These properties ensure that reading and writing the *Public Data* characteristic is without any restriction. It does not require Authentication or Encrypted link. Any GATT client can read or write it.

The *Encrypted Data* characteristic has the following properties:



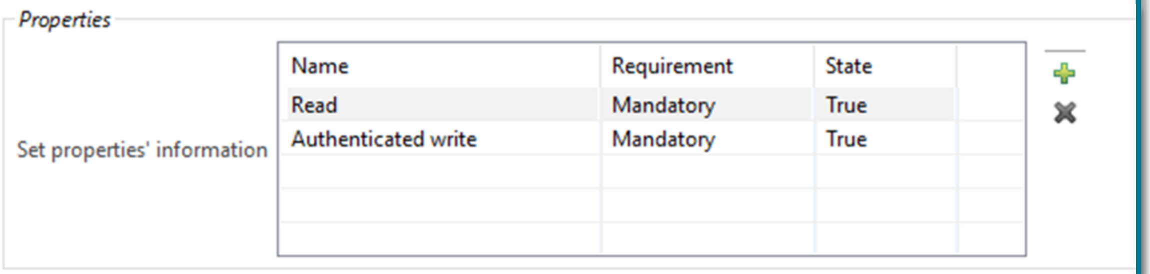
Properties

Set properties' information

Name	Requirement	State
Read	Mandatory	True
Encrypted write	Mandatory	True

Reading the *Encrypted Data* characteristic is without any restriction. Writing the *Encrypted Data* characteristic requires encrypted link. On some newer devices the device must also be bonded at least with Just Works pairing. Writing this characteristic in the first time will trigger the pairing process. The actual pairing process depends on the IO capabilities of the both client and server devices.

The *Authenticated Data* characteristic has the following properties:



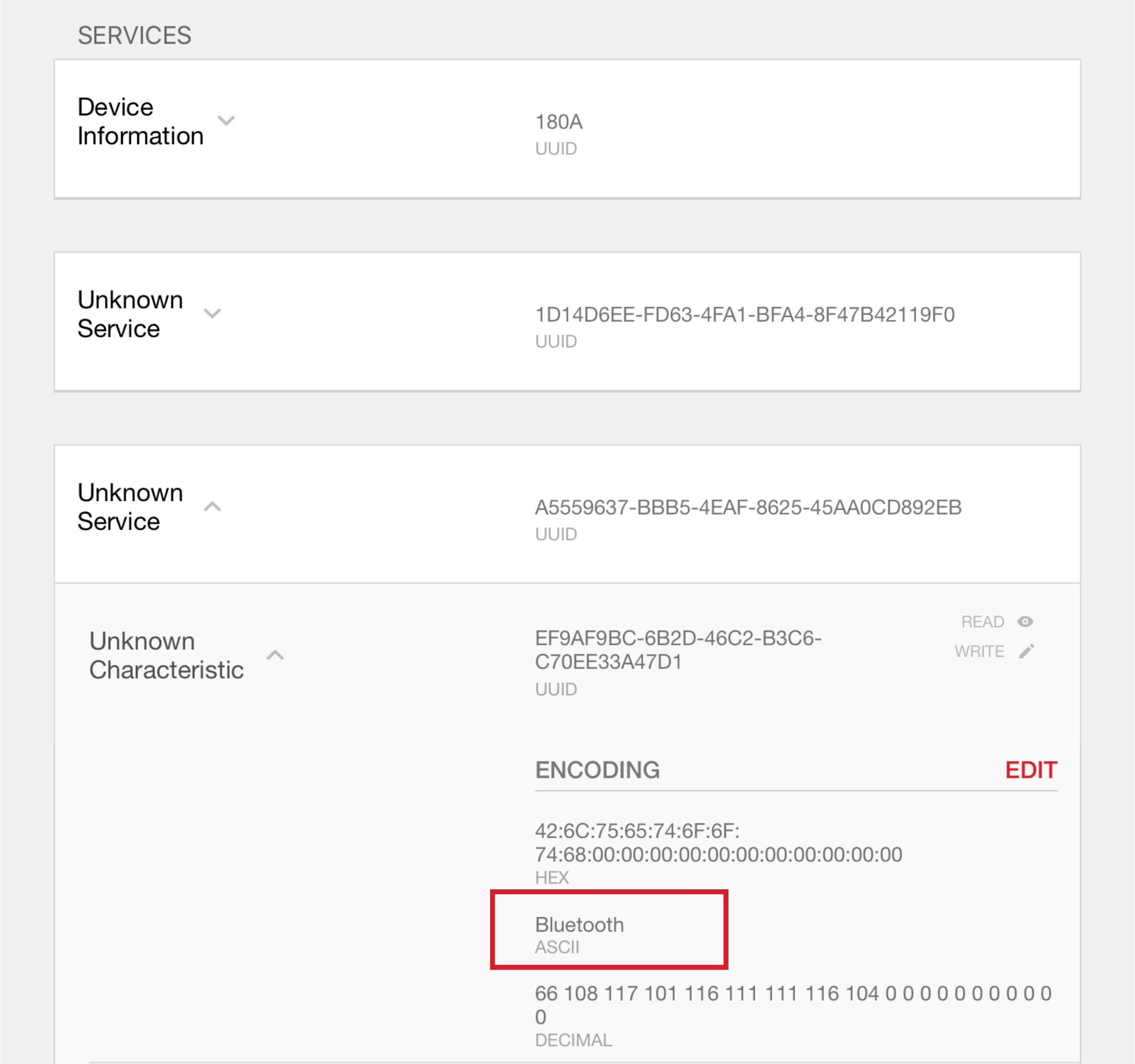
Properties

Set properties' information

Name	Requirement	State
Read	Mandatory	True
Authenticated write	Mandatory	True

Reading the *Authenticated Data* characteristic is without any restriction. But writing requires Authentication. It means the remote device must be bonded with man in the middle (MITM) protection enabled. Writing this characteristic in the first time will trigger the pairing process.

4. Read these characteristics. What is the stored data in the characteristics? Check the ASCII field in the app.



SERVICES

Device Information
180A
UUID

Unknown Service
1D14D6EE-FD63-4FA1-BFA4-8F47B42119F0
UUID

Unknown Service
A5559637-BBB5-4EAF-8625-45AA0CD892EB
UUID

Unknown Characteristic
EF9AF9BC-6B2D-46C2-B3C6-C70EE33A47D1
UUID

READ
WRITE

ENCODING EDIT

42:6C:75:65:74:6F:6F:74:68:00:00:00:00:00:00:00:00:00:00:00
HEX

Bluetooth
ASCII

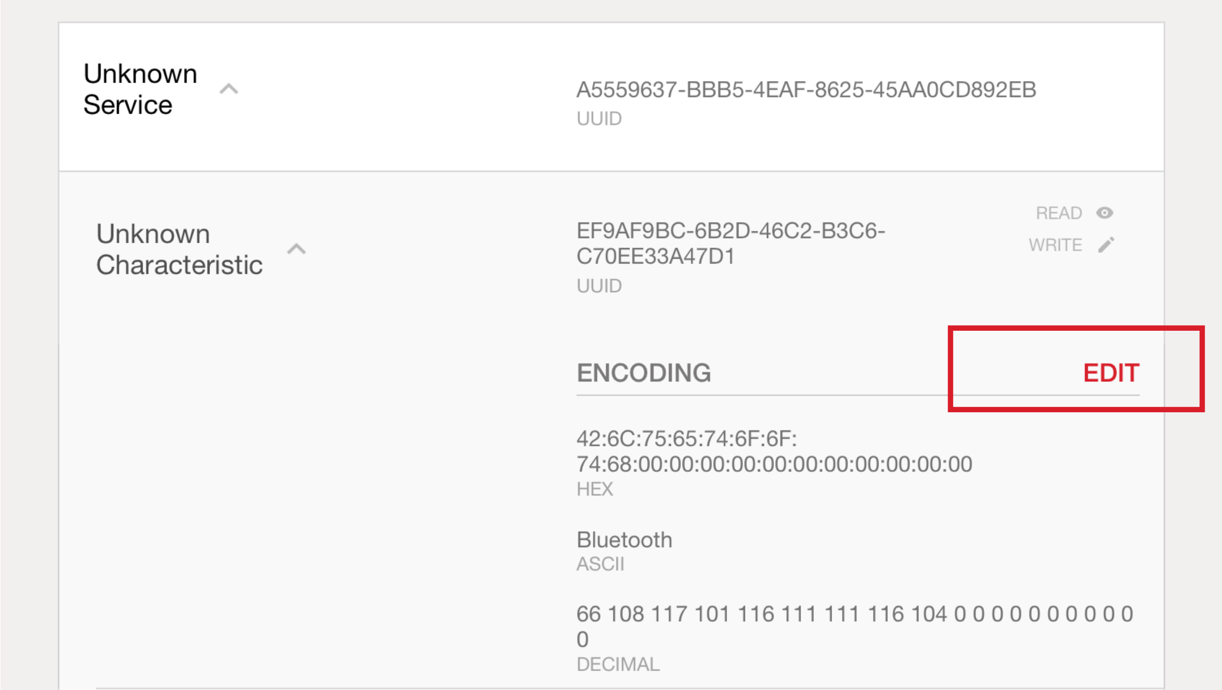
66 108 117 101 116 111 111 116 104 0 0 0 0 0 0 0 0 0 0 0
DECIMAL

Public Data *ef9af9bc-6b2d-46c2-b3c6-c70ee33a47d1* is ______________________________

Encrypted Data *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* is ______________________________

Authenticated Data *35b1c7e4-e062-405a-a9d2-adb9cd562d56* is ______________________________

5. Write these characteristics. Tap to Edit.



Unknown Service

A5559637-BBB5-4EAF-8625-45AA0CD892EB
UUID

Unknown Characteristic

EF9AF9BC-6B2D-46C2-B3C6-C70EE33A47D1
UUID

ENCODING

42:6C:75:65:74:6F:6F:74:68:00:00:00:00:00:00:00:00:00:00:00
HEX

Bluetooth
ASCII

66 108 117 101 116 111 111 116 104 0 0 0 0 0 0 0 0 0 0 0
DECIMAL

Question - Were you able to write the characteristic data? Check the access counters on the bottom of the LCD for help! **In case of pairing request, you need to accept pairing with PB0 button on the WSTK.**

Public Data	*ef9af9bc-6b2d-46c2-b3c6-c70ee33a47d1*	**yes**/no
Encrypted Data	*dfa037d5-cd4b-4a95-b00f-ed8693d0483e*	**yes**/no
Authenticated Data	*35b1c7e4-e062-405a-a9d2-adb9cd562d56*	yes/**no**

Question - Which characteristic writes triggered the pairing?

Public Data	*ef9af9bc-6b2d-46c2-b3c6-c70ee33a47d1*	yes/**no**
Encrypted Data	*dfa037d5-cd4b-4a95-b00f-ed8693d0483e*	**yes**/no
Authenticated Data	*35b1c7e4-e062-405a-a9d2-adb9cd562d56*	**yes**/no

Take away

#1 In case of *Just works* pairing there is no possibility to confirm the identity of the connecting devices. Devices will pair with encryption but without authentication.

#2 Characteristics which requires encryption cannot be accessed without pairing

#3 Characteristics which requires authentication cannot be accessed in case of just works pairing used.

#4 Android does not ask the user for pairing confirmation in case of just works pairing used

Exercise 2 - Browsing the GATT in case of passkey entry used

Goals:

- Understand the GATT permissions
- Try out passkey entry pairing

1. Reset the WSTK.
2. Select the ***passkey entry*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

passkey entry security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
        flags                 = 0x00
        io_capability         = sm_io_capability_displayonly
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
        bondable              = 1
```

You can find more about the API commands and events at Appendix – Security manager chapter

3. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device. On the EFR32 the bondings automatically deleted every time when the security configuration changed.

Warning!

Bondings created with different security configurations other than the active security configuration lead to errors in the pairing process.

4. Connect with your phone and try to write the characteristics in the GATT. Check the access counters on the bottom of the LCD for help! In case of pairing request, you need to accept pairing with PB0 button on the WSTK.

Question - Were you able to write the characteristic data?

Public Data	*ef9af9bc-6b2d-46c2-b3c6-c70ee33a47d1*	**yes**/no
Encrypted Data	*dfa037d5-cd4b-4a95-b00f-ed8693d0483e*	**yes**/no
Authenticated Data	*35b1c7e4-e062-405a-a9d2-adb9cd562d56*	**yes**/no

Take away

#1 In case of passkey entry pairing a passkey is displayed, which has to be input on the other device to confirm authentication.

#2 Characteristics which are requires authentication can be accessed in case of passkey entry pairing used.

Exercise 3 – BLE 4.2 secure connections and numeric comparison

Goals:

- Check that your phone supports LE secure connections
- Try out the numeric comparison pairing

1. Reset the WSTK.
2. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device. On the EFR32 the bondings automatically deleted every time when the security configuration changed.
3. Select the ***secure connections*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

secure connections security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
      flags               = 0x04 //force secure connections
      io_capability       = sm_io_capability_displayyesno
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
      bondable            = 1
```

You can find more about the API commands and events at Appendix – Security manager chapter

4. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device.
5. Try to write the *Authenticated Data* *35b1c7e4-e062-405a-a9d2-adb9cd562d56* characteristic or the
   Encrypted Data *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* characteristic

6. **Question** - Is it possible with your phone?

   YES – probably you have IOS / Android 7 / windows 10?

   NO – your phone or Android build does not support the BLE 4.2 LE secure connections feature

7. Reset the WSTK.
8. Select the ***numeric comparison*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.
9. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device.

numeric comparison security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
        flags                   = 0x00
        io_capability           = sm_io_capability_displayyesno
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
        bondable                = 1
```

You can find more about the API commands and events at Appendix – Security manager chapter

10. Try to write the *Authenticated Data* *35b1c7e4-e062-405a-a9d2-adb9cd562d56* characteristic. or the

    Encrypted Data *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* characteristic

After the passkey entry it should work now.

Take away

#1 BLE 4.2 LE Secure Connections feature using Elliptic Curve Diffie-Hellman (ECDH) public-private key pairs for numeric comparison paring method.

#2 In numeric comparison paring method both devices will display a 6-digit passkey. The user must confirm, that the two devices display the same passkey by pressing a button.

#3 Not all the phone supports the BLE 4.2 LE Secure Connections feature.

#4 If the phone does not support the BLE 4.2 LE Secure Connections feature numeric comparison pairing cannot be used. In this case the same io capability setting will trigger passkey entry pairing.

Exercise 4 – Preshared keys

Goals:

- Using passkey entry pairing with pre-shared keys

1. Reset the WSTK.
2. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device. On the EFR32 the bondings automatically deleted every time when the security configuration changed.
3. Select the ***preshared keys*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

Implementation details – *preshared keys* security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
flags              = 0x00
io_capability      = sm_io_capability_displayyesno
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
bondable           = 1
```

Additionally, this configuration calls the **gecko_cmd_sm_set_passkey(123456)** API to set the pre-shared key to 123456.

You can find more about the API commands and events at Appendix – Security manager chapter

4. Try to write the *Authenticated Data* *35b1c7e4-e062-405a-a9d2-adb9cd562d56* characteristic. or the *Encrypted Data* *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* characteristic
5. The key is **123456**.

Take away

However, the BLE not supports pre-shared keys directly this is possible with **gecko_cmd_sm_set_passkey** API

Exercise 5 – not bondable

Goals:

- Learn how to forbid bonding

1. Reset the WSTK.
2. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device. On the EFR32 the bondings automatically deleted every time when the security configuration changed.
3. Select the ***not bondable 1*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

not bondable 1 security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
flags               = 0x00
io_capability       = sm_io_capability_displayyesno
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
bondable            = 0
```

You can find more about the API commands and events at Appendix – Security manager chapter

4. Try to write the *Authenticated Data* *35b1c7e4-e062-405a-a9d2-adb9cd562d56* characteristic. or the
   Encrypted Data *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* characteristic

Question – Does it bond? **yes**/no

5. Reset the WSTK.
6. Open the Bluetooth Settings on your phone and delete the bonding with *Security Example* device. On the EFR32 the bondings automatically deleted every time when the security configuration changed.
7. Select the ***not bondable 2*** security configuration on the WSTK. You can change the security configuration with the PB0 and PB1 buttons.

not bondable 2 security configuration

This configuration uses **gecko_cmd_sm_configure (flags , io_capabilities)** API with the following parameters

```
flags               = 0x08 //Bonding requests need to be confirmed
io_capability       = sm_io_capability_displayyesno
```

This configuration calling **gecko_cmd_sm_set_bondable_mode(bondable)** API with the following parameter.

```
bondable            = 1
```

You can find more about the API commands and events at Appendix – Security manager chapter

8. Try to write the *Authenticated Data* *35b1c7e4-e062-405a-a9d2-adb9cd562d56* characteristic. or the
   Encrypted Data *dfa037d5-cd4b-4a95-b00f-ed8693d0483e* characteristic

   It leads to the same paring but in this case the bondings have to be accepted by the user.

Take away

#1 gecko_cmd_sm_set_bondable_mode(bondable) API has no effect.

#2 Bondings can be accepted or rejected on application level

Appendix

GATT permissions

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. Attributes have a set of permissions that controls whether they can be read or written, or whether the attribute value shall be sent over an encrypted link. Attribute permissions are used by the server to determine whether read or write access is permitted for a given attribute. Attribute permissions are established by the GATT Profile.

Authentication

Authentication is defined as a way to prove that the device with which you are connecting is actually the device it claims to be and not a third-party attacker.

Authentication in the GATT Profile is applied to each characteristic independently. The GATT Profile procedures are used to access information that may require the client to be authenticated.

A characteristic may be allowed to be read by any device, but only written by an authenticated device. Similarly, if a characteristic can be written, it does not mean the characteristic can also be read. Each individual characteristic could have different security properties.

Encryption

Encryption is the process of encoding a message or information in such a way that only authorized parties can access it.

The requirement to have encrypted link before reading or writing a characteristic stored in GATT Profile. This requirement is applied to each characteristic independently.

Services and Characteristics

The list of services and characteristics that a device supports is not considered private or confidential information, and therefore the Service and Characteristic Discovery procedures are always permitted.

Defining GATT permissions

The characteristics access and security properties are defined in the *gatt.xml* file by the XML attribute **<properties>** and its parameters, which must be used inside the **<characteristic>** XML attribute tags. Alternatively, Visual GATT Editor can be also used for defining the GATT elements and their permissions.

The table below describes the parameters that can be used for defining the related values.

Parameter	Description
read	Characteristic can be read by a remote device true: Characteristic can be read false: Characteristic cannot be read
write	Characteristic can be written by a remote device true: Characteristic can be written false: Characteristic cannot be written
bonded_read	Reading the characteristic value requires an encrypted link. Devices must also be bonded at least with Just Works pairing. true: Bonding and encryption are required false: Bonding is not required
bonded_write	Writing the characteristic value requires an encrypted link. Devices must also be bonded at least with Just Works pairing. true: Bonding and encryption are required false: Bonding is not required
authenticated_read	Reading the characteristic value requires an authentication. In order to read the characteristic with this property the remote device has to be bonded using MITM protection and the connection must be also encrypted. true: Authentication is required false: Authentication is not required
authenticated_write	Writing the characteristic value requires an authentication. In order to write the characteristic with this property the remote device has to be bonded using MITM protection and the connection must be also encrypted. true: Authentication is required false: Authentication is not required
encrypted_read	Reading the characteristic value requires an encrypted link. With iOS 9.1 and newer devices must also be bonded at least with Just Works pairing. true: Encryption is required false: Encryption is not required
encrypted_write	Writing the characteristic value requires an encrypted link. With iOS 9.1 and newer devices must also be bonded at least with Just Works pairing. Values: true: Encryption is required false: Encryption is not required

*bonded_read and bonded_write not supported at the moment -> create JIRA ticket about this

Security manager

The Bluetooth SDK provides APIs for configuring the Security Manager and to let the user interact during the pairing process.

Configuration

Before pairing, the Security Manager (SM) has to be configured based on the IO capability of the device and on the requirements of the application. This allows the SM decide how to pair with the remote device. The configuration can be done with the following API calls:

gecko_cmd_sm_configure (flags , io_capabilities)

This command can be used to configure security requirements and I/O capabilities of the system. The security requirements are signaled with flags.

Flags	Usage
bit 0	0: Allow bonding without MITM protection 1: Bonding requires MITM protection
bit 1	0: Allow encryption without bonding 1: Encryption requires bonding
bit 2	0: Allow bonding with legacy pairing 1: Secure connections only
bit 3	0: Bonding request does not need to be confirmed 1: Bonding requests need to be confirmed. Received bonding requests are notified with sm_confirm_bonding events
bit 2 to 7	Reserved

The capabilities parameter tells what kind of user input and output methods are available on our device. The different IO capabilities leads to different pairing methods.

IO_capabilities	Available input/output on the device
sm_io_capability_displayonly (= 0)	6 digit numeric display, no keyboard
sm_io_capability_displayyesno (= 1)	6 digit numeric display, two buttons or equal to confirm yes/no
sm_io_capability_keyboardonly (= 2)	No display, numeric keyboard
sm_io_capability_noinputnooutput (= 3)	No display, no keyboard
sm_io_capability_keyboarddisplay (= 4)	6 digit numeric display, numeric keyboard

gecko_cmd_sm_set_bondable_mode (bondable)

This command can be used to set whether the device accepts new bondings or not.

Values:

- **0:** New bondings not accepted
- **1:** Bondings allowed

Note: This API has no effect in SDK2.3.1. or later

Pairing process

The pairing process is an interactive process, since the user has to confirm the identity of the connecting devices. This interaction is realized with events raised by the stack and commands sent to the stack.

Based on the IO capabilities of the two devices, there are five types of pairing processes. In each case there is a different sequence of events and commands which has to be followed. When implementing an application, the developer has to consider all scenarios that may happen with the IO capabilities of the device.

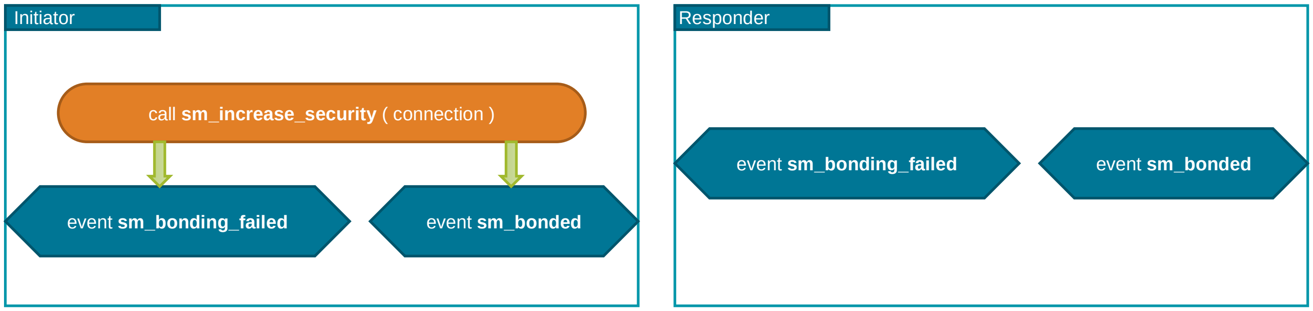
		Initiator				
		DisplayOnly	**DisplayYesNo**	**KeyboardOnly**	**NoInputNoOutput**	**KeyboardDisplay**
Responder	**DisplayOnly**	Just Works	Just Works	Passkey Entry (R displays, I inputs)	Just Works	Passkey Entry (R displays, I inputs)
	DisplayYesNo	Just Works	Numeric Comparison	Passkey Entry (R displays, I inputs)	Just Works	Numeric Comparison
	KeyboardOnly	Passkey Entry (I displays, R inputs)	Passkey Entry (I displays, R inputs)	Passkey Entry (R and I inputs)	Just Works	Passkey Entry (I displays, R inputs)
	NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
	KeyboardDisplay	Passkey Entry (I displays, R inputs)	Numeric Comparison	Passkey Entry (R displays, I inputs)	Just Works	Numeric Comparison

The different IO capabilities leads to different pairing methods

Just works

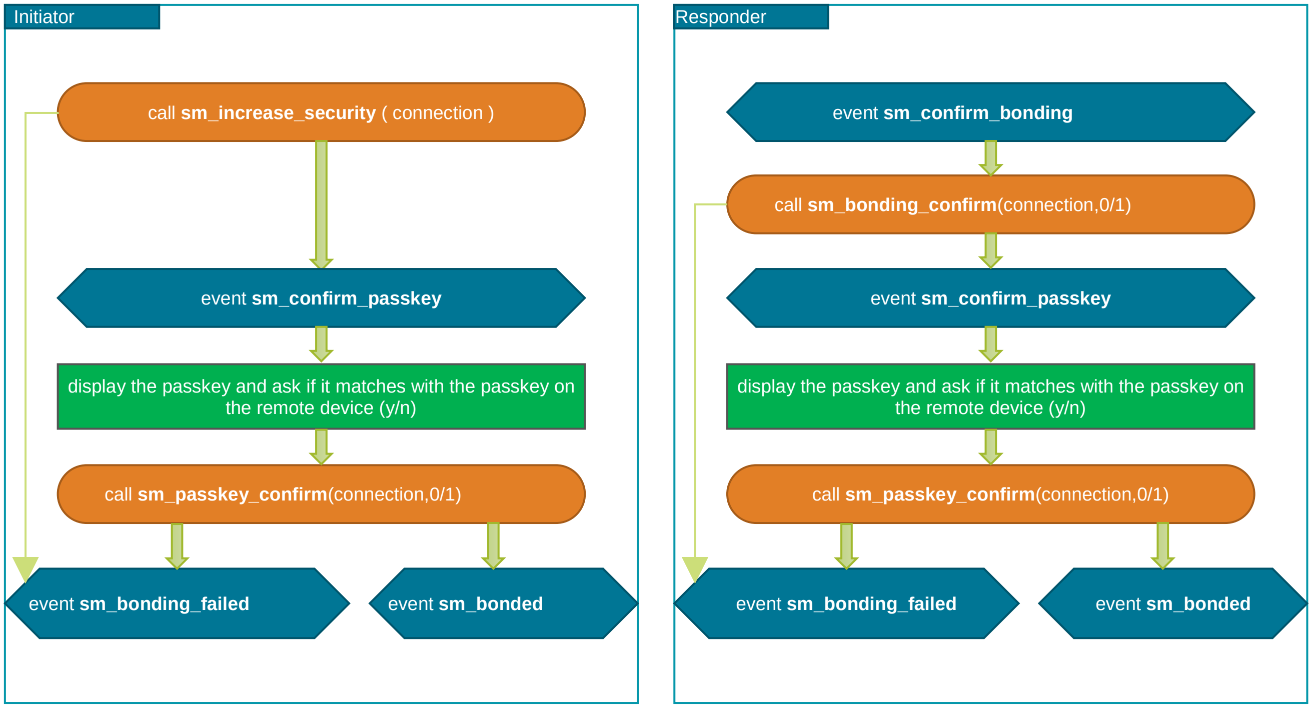
In this case there is no possibility to confirm the identity of the connecting devices, so no interaction is needed. Devices will pair with encryption but without authentication.

The just works method provides no protection against eavesdroppers or man in the middle attacks during the pairing process. If the attack is not present during the pairing process, then confidentiality can be established by using encryption on a future connection.



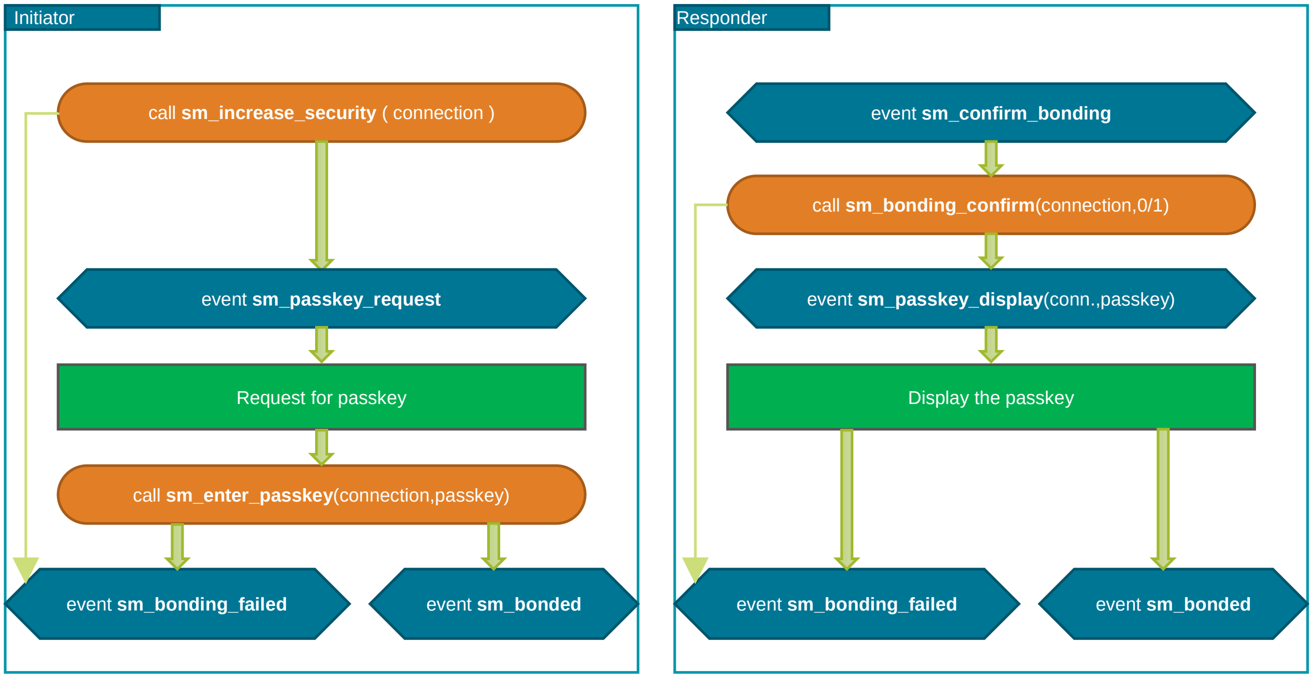
Numeric Comparison

In this case both devices will display a 6-digit passkey. The user has to confirm, that the two devices display the same passkey by pressing a button. If the passkeys match, it means that there were no Man-In-The-Middle, and the devices could exchange keys securely


Initiator
call sm_increase_security (connection)
event sm_confirm_passkey
display the passkey and ask if it matches with the passkey on the remote device (y/n)
call sm_passkey_confirm(connection,0/1)
event sm_bonding_failed
event sm_bonded
Responder
event sm_confirm_bonding
call sm_bonding_confirm(connection,0/1)
event sm_confirm_passkey
display the passkey and ask if it matches with the passkey on the remote device (y/n)
call sm_passkey_confirm(connection,0/1)
event sm_bonding_failed
event sm_bonded

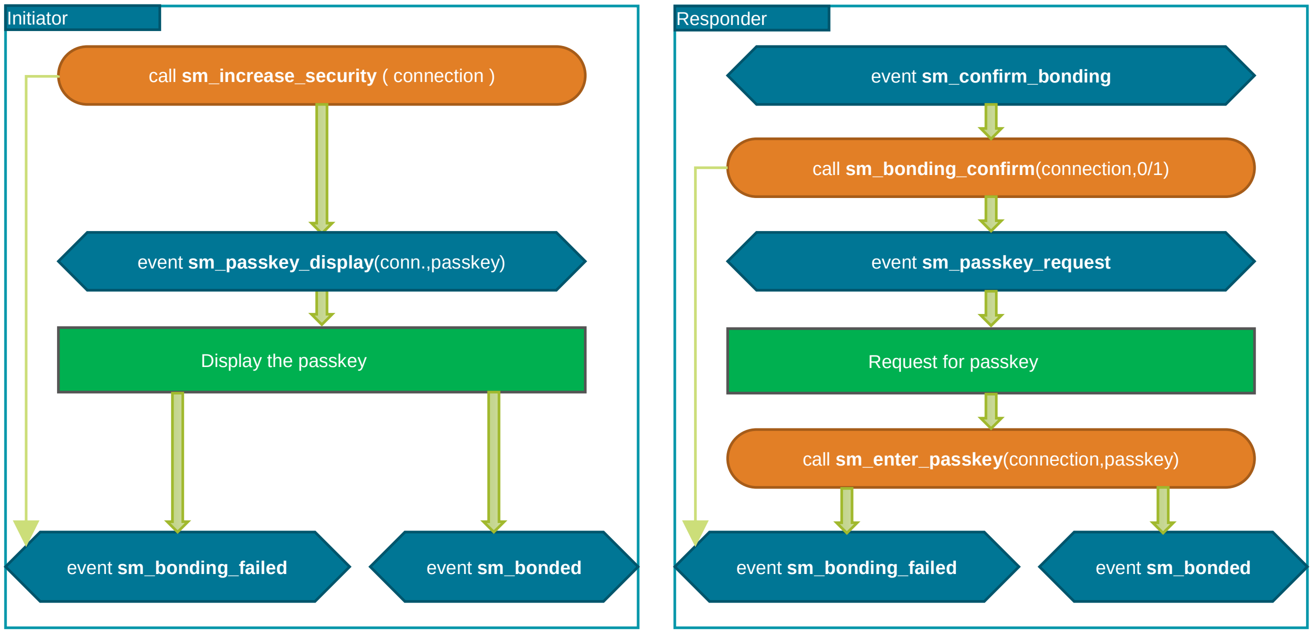
Passkey entry: responder displays, initiator inputs

A passkey is displayed on the responder device, which must be typed in with the use of the numeric keyboard on the initiator device.



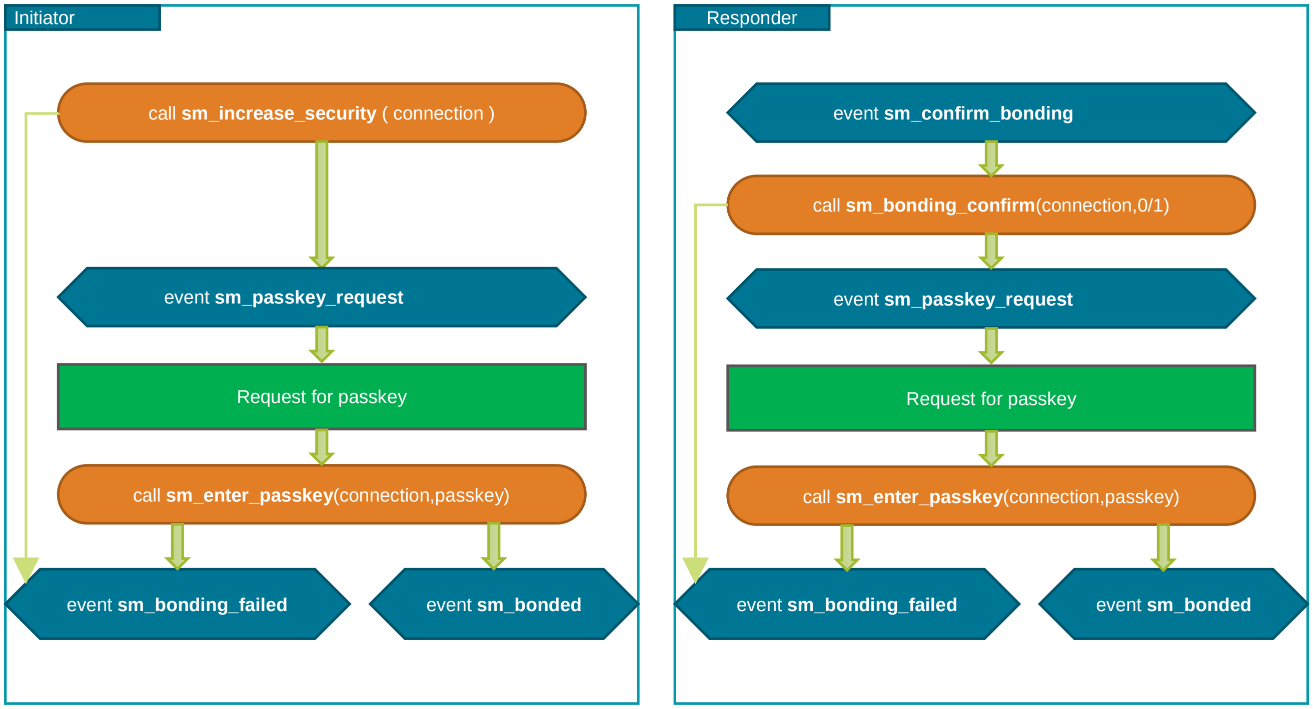
Passkey entry: initiator displays, responder inputs

A passkey is displayed on the initiator, which must be input on the responder device to confirm authentication.



Passkey entry: initiator and responder inputs

In this scenario, both devices must input a passkey. In contrast to other scenarios where the passkey was generated with either one of the devices, in this case the user must find out a key, and enter the same key in both devices.



Initiator
call **sm_increase_security** (connection)
event **sm_passkey_request**
Request for passkey
call **sm_enter_passkey**(connection,passkey)
event **sm_bonding_failed**
event **sm_bonded**
Responder
event **sm_confirm_bonding**
call **sm_bonding_confirm**(connection,0/1)
event **sm_passkey_request**
Request for passkey
call **sm_enter_passkey**(connection,passkey)
event **sm_bonding_failed**
event **sm_bonded**

References

[1] Bluetooth Smart Software API Reference Manual

[2] BLUETOOTH SPECIFICATION Version 4.2