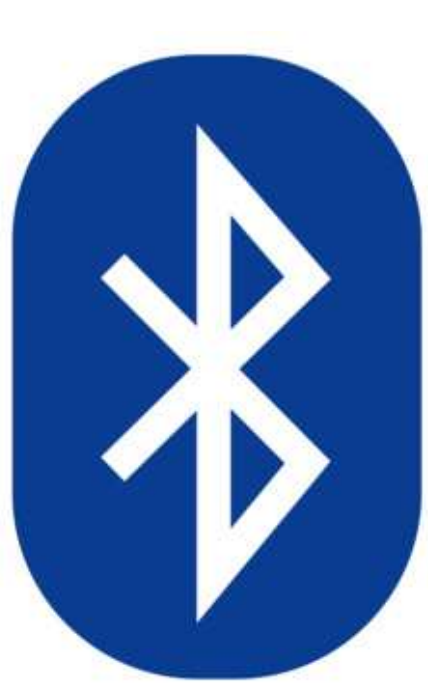
Payatu

IoT Security – Part 4 (Bluetooth Low Energy – 101)

Arun-Magesh

22/10/2018



Bluetooth Low Energy 101

If you haven't read through Part 1 to Part 3 of our IoT Security Blog series I would urge you to go through them first unless you are already familiar with the basics of IoT. Link to the previous blog – IoT security – Part 3

Bluetooth has been a buzz-word as people wanted all their devices to be smart and which basically implies that you get to control things across the devices and not needing to carry wire around. Bluetooth has been in the market for more than a decade. If you're a millennial, you would have used those classic fancy Nokia phone which has Bluetooth in it. Bluetooth was invented by Ericsson and other vendors have started using Bluetooth. Soon after that, all the major vendors created a consortium called as Bluetooth Special Interest Group – SIG which governs how the standard should be and the interoperability between different versions.

We are not going to talk about Bluetooth. Bluetooth by itself is a massive stack and their specification is around 2000+ pages. In this blog, I will be covering only the Bluetooth Low Energy more famously known as BLE.

With the advent of connecting all the things to the internet, there comes the problem of power and resource. As I mentioned early, Bluetooth is a huge stack. Implementing it in an end device like a fitness band would take more power and resource. So in the Bluetooth 4.0 standard, they introduced something called Low energy which is specially targeted for IoT and smart devices which runs on memory and power constrained devices.

Bluetooth SIG started selling the standard as Bluetooth Smart. Which has two components, Bluetooth smart devices are end devices which have only the Bluetooth Low Energy component and Bluetooth smart Ready are the device which is capable of doing both the Bluetooth LE and the EDR-Bluetooth classic component which could be your central device, ie, mobile phone or laptop.

Now let's look into the technical details of the Bluetooth specification

Technical Specification	Classic Bluetooth technology	Bluetooth low energy technology
Radio Frequency	2.4GHz	2.4GHz
Distance/Range	10m	10m
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.2 Mbit/s
Active slaves	7-16,777,184	Unlimited
Security	64/128bit and application layer user defined	128bit AES and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive fast frequency hopping
Latency (from a non-connected state)	Typically 100 ms	6 ms
Total time to send data	100 ms	<6 ms
Government Regulation	Worldwide	Worldwide
Certification Body	Bluetooth SIG	Bluetooth SIG
Voice capable	Yes	No
Network topology	Scatternet	Star-bus
Power consumption	1 as the reference	0.01 to 0.5 (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, automotive, PCs etc.	Mobile phones, gaming, PCs, watches, sports and fitness, healthcare, security & proximity, automotive, home electronics, automation, Industrial, etc.

Source:
https://archive.eetindia.co.in/www.eetindia.co.in/STATIC/ARTICLE_IMAGES/201312/EEIOL_2013DEC13_F

The table itself will give you a better insight into the specification, range and bandwidth has been reduced to withstand the low power and low resource.

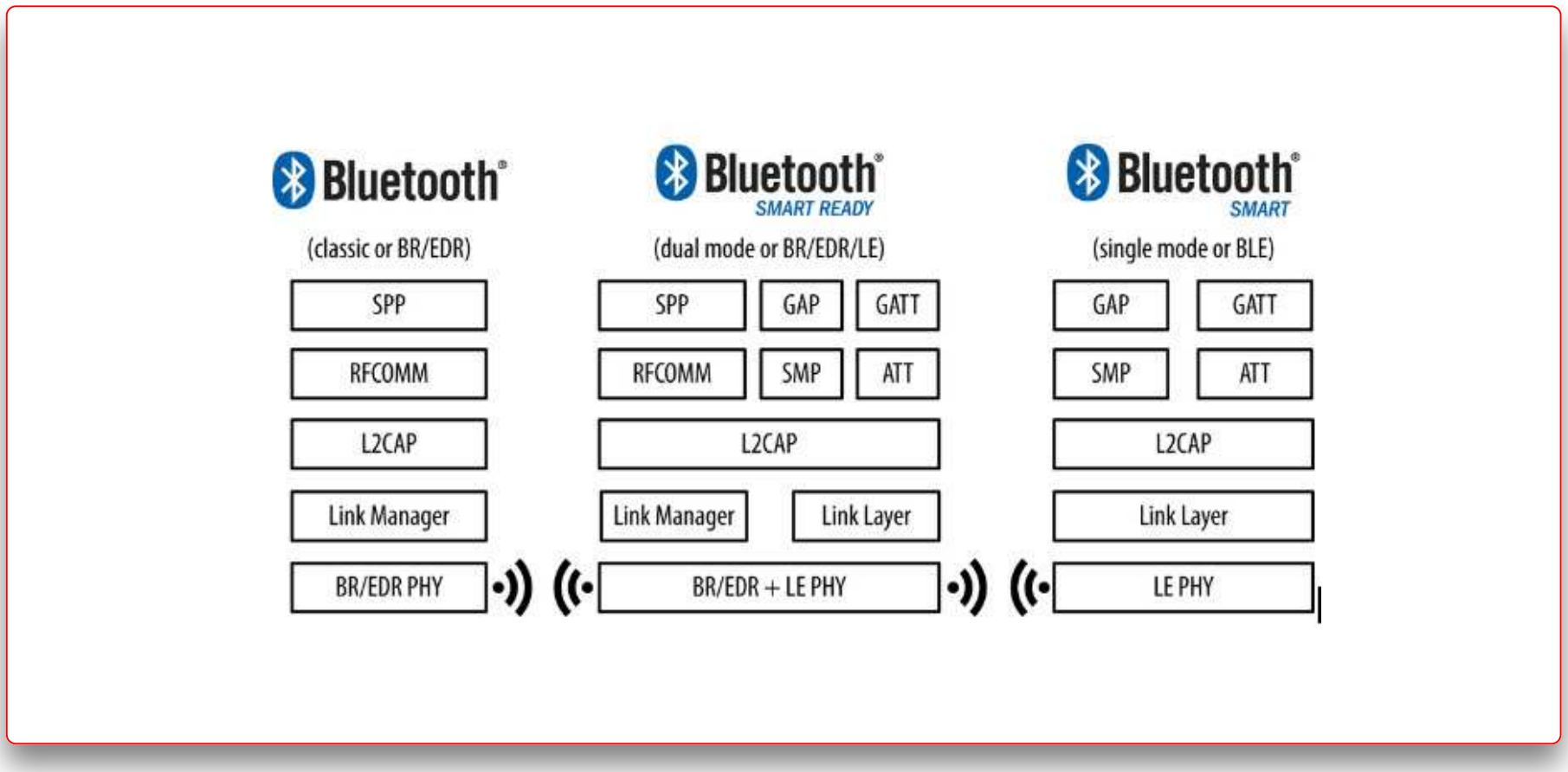
As I mentioned earlier, LE has two different types of devices.

Bluetooth Smart Ready – Which are the central device which is battery powered and high resource which is capable of running all the Bluetooth protocols. They are your laptops and a mobile phone.

Bluetooth Smart – They are your end devices like fitness tracker or baggage tracker or a smart dildo. They don't have to run an entire stack and they need to conserve power and resource. They run only the Bluetooth LE server. They are the peripheral device that the central device can connect to.

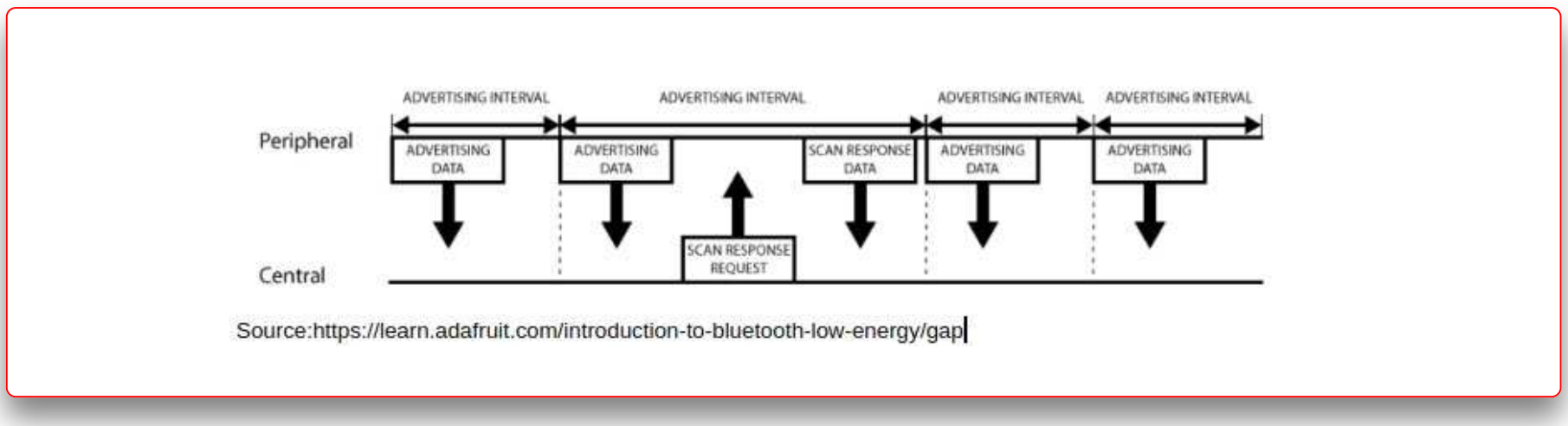
Bluetooth and LE stack details are out of the scope of this document.

But the two important components we will focus on are GAT and GAPP which are responsible for the operation of the BLE service.



Generic Access Profile (GAP)

GAP defines how your communication and connection to the central and peripheral should work.



Source:https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

Generic Attribute (GATT)

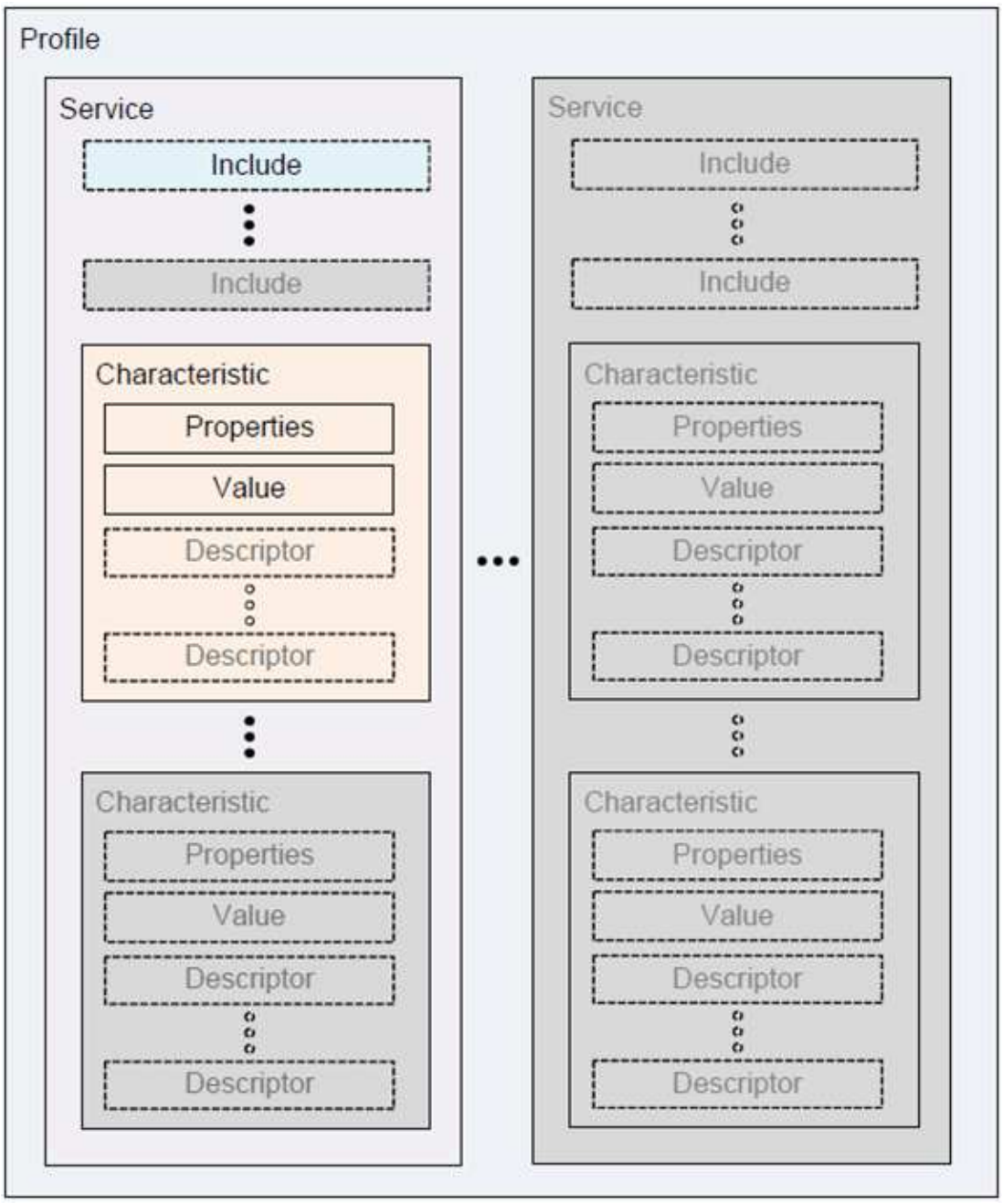
GATT is like a server which manages how your data needs to be treated.

Your Bluetooth LE devices work as a server-client principle. Here your end device/peripheral device acts as the server which runs the GATT server and your central device, acts as the client. So your end app or the tool connects to the GATT server and requests data from the device.

Inside your GATT server. There are three components.

1. **Profile** – Which is defined by the Bluetooth SIG, it could be based on the type of the device, be it a blood pressure device or temperature sensor or any most commonly used device which has an advantage of interoperability.
2. **Services** – Each device has multiple parameters inside it. Let's say a device could have a name, firmware version, OTA functionality, device operation. They are grouped into their specific datasets called as service.
3. **Characteristics** – inside your service is where your data is placed. It could be a 16 bit Bluetooth SIG derived characteristic or a vendor-specific 128-bit characteristic.

In short, service is like a folder and characteristics are the files which holds the data.



Now that we understood the basics of what is Bluetooth LE and how it functions. Let's go into some tools and methods on how to access the BLE devices.

If you are using windows, I would seriously suggest you use Ubuntu as it comes with all the necessary tools to access ble devices and get those cheap Bluetooth 4.0 dongles from Amazon. (some laptops don't come with it.)

Connecting your Bluetooth dongle:

1. Connect the Bluetooth USB Dongle to the free USB port of your laptop. (No need to install any driver from your host machine).
2. Once Connected, open your terminal and type "**sudo hciconfig**" You should be able to see this window which gives you the mac address(The USB dongle) and it should say UP and RUNNING.
3. If you encounter any issue restart the Bluetooth interface by "sudo hciconfig hci0 reset" → This will be handy a lot of time.

```
buzz@expliotable:~$ sudo hciconfig
[sudo] password for buzz:
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:628 acl:0 sco:0 events:39 errors:0
        TX bytes:1681 acl:0 sco:0 commands:39 errors:0
```

Scanning for Bluetooth devices

1. Once you have successfully connected your Bluetooth dongle to your machine
2. You can now scan for all the ble devices around you using **"sudo hcitool lescan"**

```
buzz@expliotable:~$ sudo hcitool lescan
LE Scan ...
0F:5E:0F:43:0D:22 (unknown)
2D:9A:AE:48:01:CA (unknown)
00:82:21:91:1F:A1 (unknown)
EF:CE:06:17:1A:EC Y5-1AEC
```

3. You will see a list of devices with their name and MAC address.
4. Figure out the mac of your device by turning it off and on and finding the difference.
5. Now to get more information about the device. Do a "sudo hcitool leinfo –random <mac>" – random depends on the type addressing.

```
buzz@expliotable:~$ sudo hcitool leinfo --random E8:77:6D:8B:09:96
Requesting information ...
        Handle: 70 (0x0046)
        LMP Version: 4.1 (0x7) LMP Subversion: 0x64
        Manufacturer: Nordic Semiconductor ASA (89)
        Features: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

6. You will get basic information like the manufacturer of radio.

Reading and writing data

1. Once you got the MAC address of your device. Save it in a file. It will be useful.
2. To connect to a smart device's GATT server. We use a tool called as gatttool.
3. Using this command " **sudo gatttool -I -b <mac> -t random**" you will get a CLI like this and type " connect" to it.

```
buzz@expliotable:~$ sudo gatttool -I -b E8:77:6D:8B:09:96 -t random
[E8:77:6D:8B:09:96][LE]> connect
Attempting to connect to E8:77:6D:8B:09:96
Connection successful
[E8:77:6D:8B:09:96][LE]>
```

4. Now you can see the characteristics and services running on the device by using **"primary**" , "**characteristics**" and "**char-desc**" to see all the UUIDs running in the device

```
[E8:77:6D:8B:09:96][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-00805f
9b34fb
attr handle: 0x0008, end grp handle: 0x000b uuid: 00001801-0000-1000-8000-00805f
9b34fb
attr handle: 0x000c, end grp handle: 0x0011 uuid: c3e6fea0-e966-1000-8000-be99c2
23df6a
attr handle: 0x0012, end grp handle: 0xffff uuid: 0000fee7-0000-1000-8000-00805f
9b34fb
[E8:77:6D:8B:09:96][LE]>
```

5.

```
[E8:77:6D:8B:09:96][LE]> characteristics
handle: 0x0002, char properties: 0x0a, char value handle: 0x0003, uuid: 00002a00
-0000-1000-8000-00805f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01
-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007, uuid: 00002a04
-0000-1000-8000-00805f9b34fb
handle: 0x0009, char properties: 0x20, char value handle: 0x000a, uuid: 00002a05
-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x10, char value handle: 0x000e, uuid: c3e6fea2
-e966-1000-8000-be99c223df6a
handle: 0x0010, char properties: 0x0c, char value handle: 0x0011, uuid: c3e6fea1
-e966-1000-8000-be99c223df6a
handle: 0x0013, char properties: 0x12, char value handle: 0x0014, uuid: 0000fea1
-0000-1000-8000-00805f9b34fb
[E8:77:6D:8B:09:96][LE]>
```

6. Now you can read and write to these handles using " **char-read-hnd <handle>**" and " **char-write-req <handle> <data>** " to read and write to it.

```
[E8:77:6D:8B:09:96][LE]> char-write-req 0x11 AABB
Characteristic value was written successfully
[E8:77:6D:8B:09:96][LE]>
```

```
[E8:77:6D:8B:09:96][LE]> char-read-hnd 0x0003
Characteristic value/descriptor: 46 34
[E8:77:6D:8B:09:96][LE]>
```

7. Here the char properties give you the permission of the handle like Read, Write, Notify, Indicate.
8.

characteristic, as defined by higher layer specifications, without regard to security requirements.

Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1.
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4.
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgment using the procedure defined in Section 4.10. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgment using the procedure defined in Section 4.11. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2.
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section

9.

```
[E8:77:6D:8B:09:96][LE]> char-desc
handle: 0x0001, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0002, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0007, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x0008, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0009, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x000a, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x000b, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x000c, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x000d, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x000e, uuid: c3e6fea2-e966-1000-8000-be99c223df6a
handle: 0x000f, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x0010, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0011, uuid: c3e6fea1-e966-1000-8000-be99c223df6a
handle: 0x0012, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0013, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0014, uuid: 0000fea1-0000-1000-8000-00805f9b34fb
handle: 0x0015, uuid: 00002902-0000-1000-8000-00805f9b34fb
```

10. You can enable notification by writing "01" to the handle too
11.

```
[E8:77:6D:8B:09:96][LE]> char-write-req 0x0f 0100
Characteristic value was written successfully
Notification handle = 0x000e value: ba 30 00 06 00 24 00 00 0a 00 ab 00 01 48
Notification handle = 0x000e value: ba 30 00 06 00 c5 00 00 0a 00 ab 00 01 4c
[E8:77:6D:8B:09:96][LE]>
```

You can check our other blogs on how to reverse a Bluetooth communication of a smart massager.

Continue to the next part - IoT Security – Part 5 (ZigBee Protocol - 101)

Reference:

1. https://www.nordicsemi.com/eng/News/ULP-Wireless-Update/A-short-history-of-Bluetooth
2. https://www.bluetooth.com/specifications/gatt/generic-attributes-overview
3. https://www.bluetooth.com/specifications
4. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt
5. https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch01.html
6. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
7. https://www.jaredwolff.com/blog/get-started-with-bluetooth-low-energy/
8. http://object-network.blogspot.com/2014/01/scanning-ble-adverts-from-linux.html
9. https://elinux.org/images/3/32/Doing_Bluetooth_Low_Energy_on_Linux.pdf
10. **https://www.digikey.com/Web%20Export/Supplier%20Content/Laird_776/PDF/laird-wireless-bluetooth-smart-ready.pdf**