BLE

1

INTRODUCTION

Specs and usage

BLUETOOTH LOW ENERY

What is **Bluetooth Low Energy** technology?

- Wireless personal area network
- Improved Bluetooth technology
  - Reduce power consumption
  - Reduce cost
  - Similar communication range

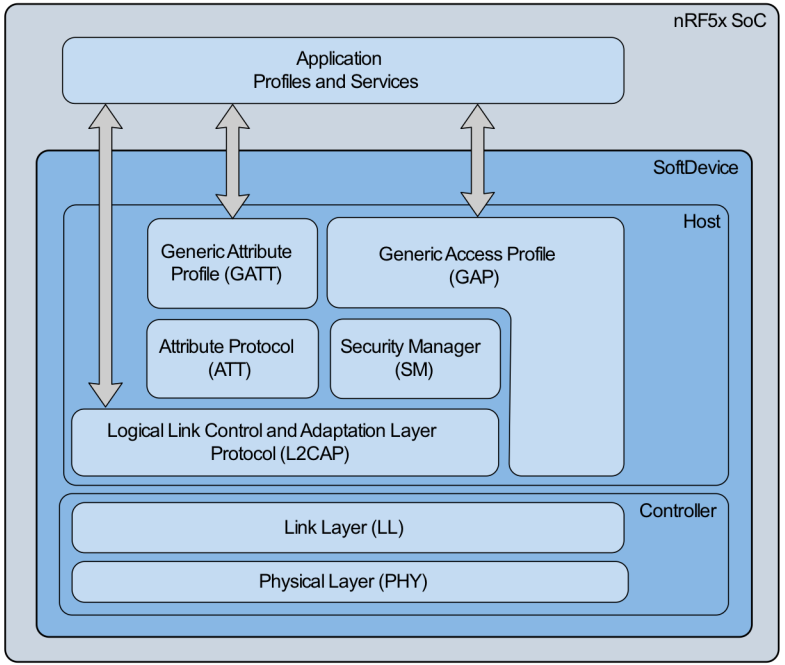
TECHNICAL SPECIFICATION

	Classic Bluetooh	Bluetooth Low Energy
Range	100m	>100m
Data rate	2.1 Mbit/s	0.27 Mbit/s
Power consumption	1 W (reference)	0.01 – 0.50 W

APPLICATIONS

- Proximity sensing
- HID connectivity
- Health care
- Sport and fitness
- Internet connectivity
- IoT – Smart device – Game – …

BLE PROTOCOL STACK

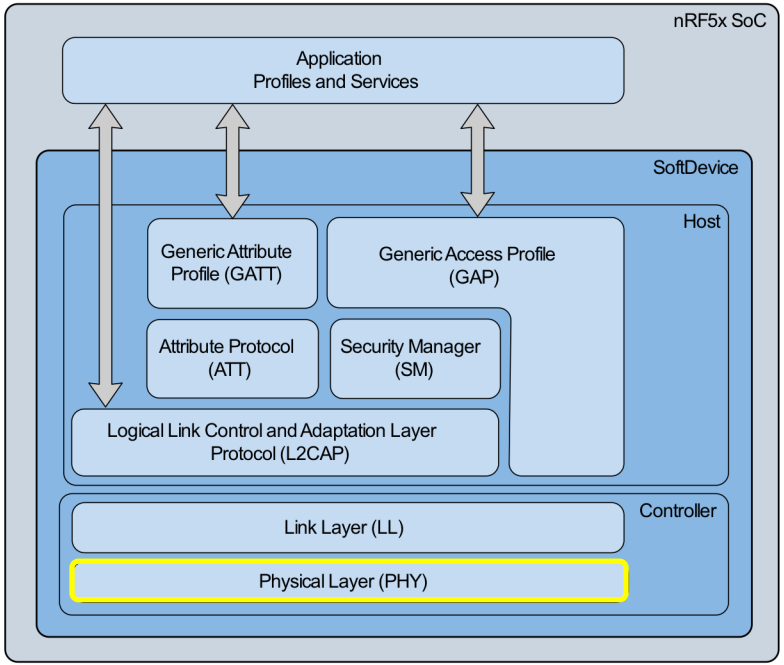


2

PHYSICAL

Radio, modulation, band, etc

PHYSICAL LAYER

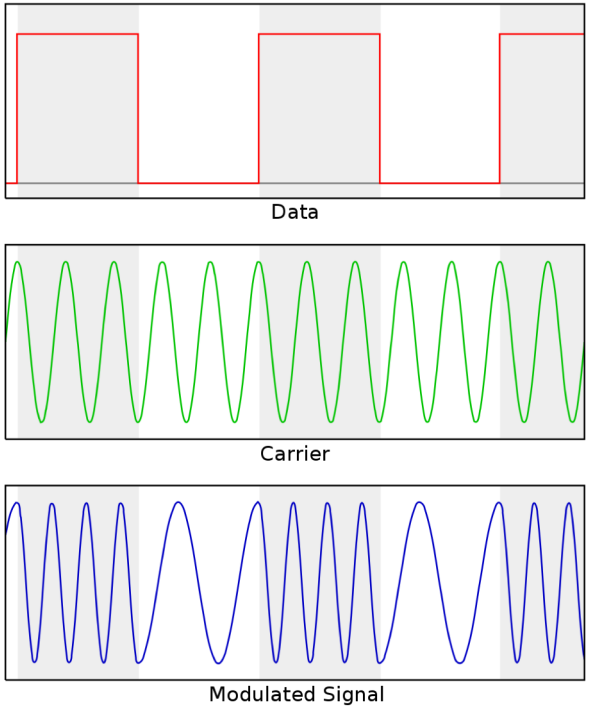


RADIO FREQUENCY

2.4 GHz ISM band

- Industrial Scientific Medical (ISM) band
- Licence free (with certain rules)
- 2400 MHz to 2483.5 MHz
- Also used by 802.11 (WiFi) and microwave,

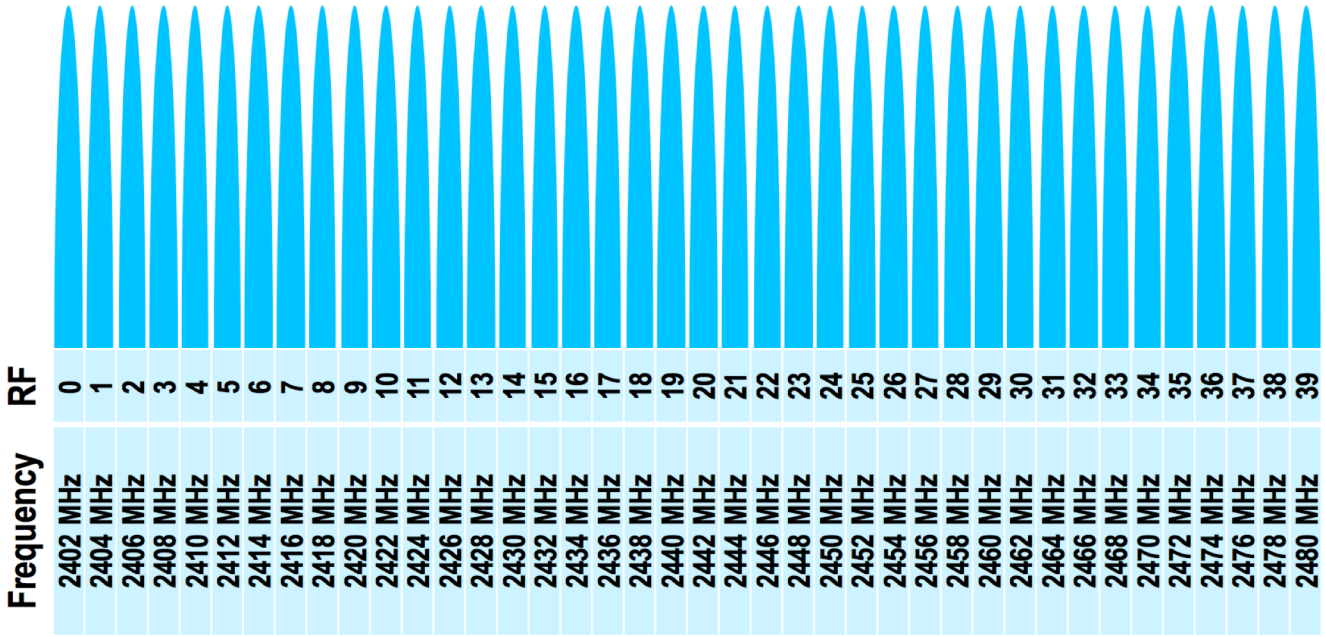
MODULATION



Gaussian frequency-shift keying

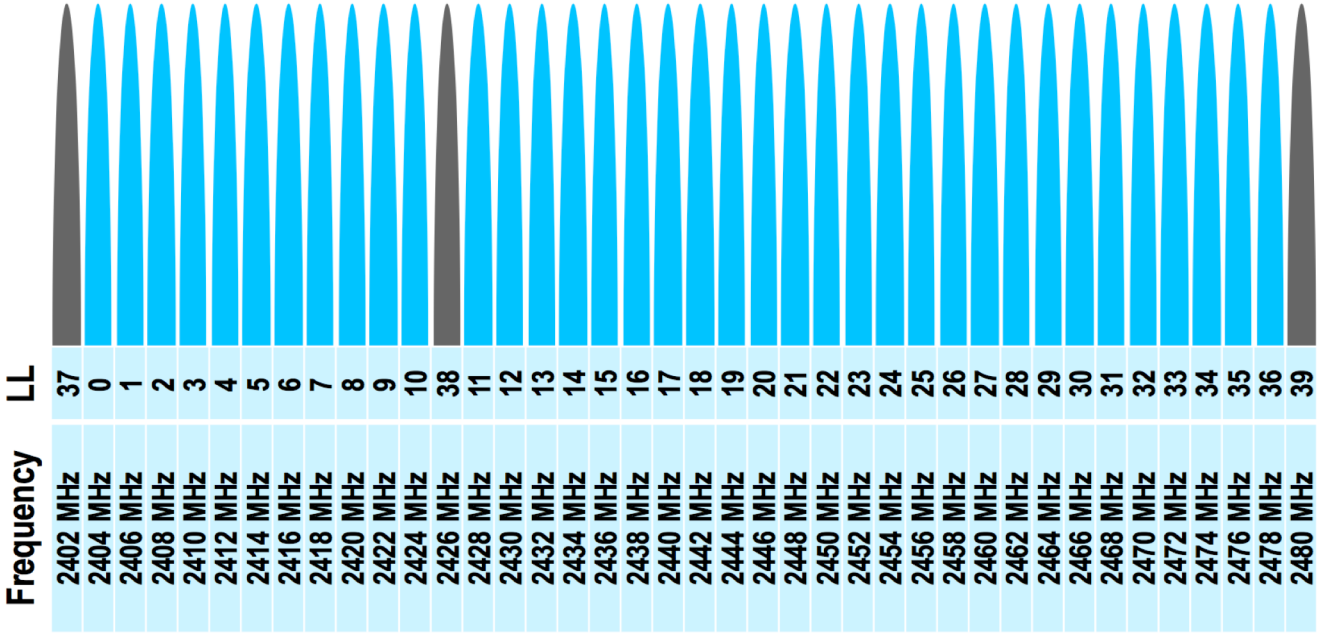
- Data transmitted through frequency changes
- Filters data with Gaussian filter prior to modulation

CHANNELS



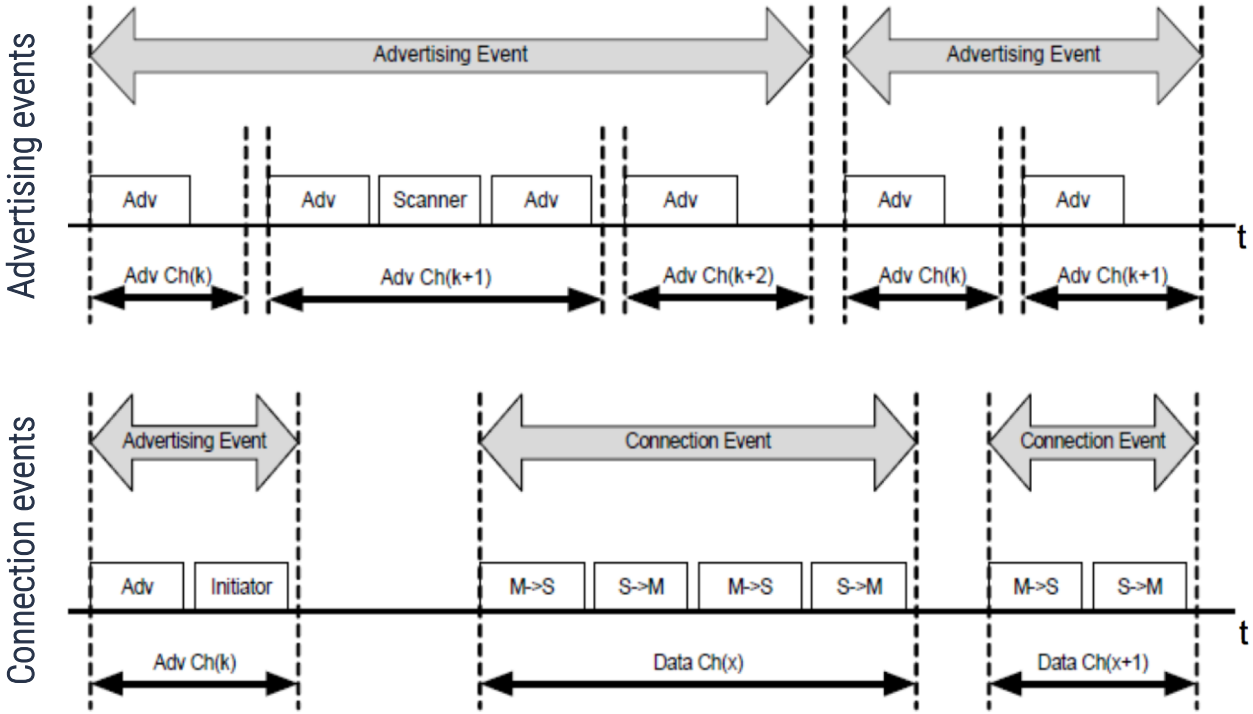
- 40 channels
- 2402 MHz – 2480 MHz
- 2 MHz spacing

CHANNELS



- 3 advertising channel
- 37 data channels

CHANNELS HOPPING

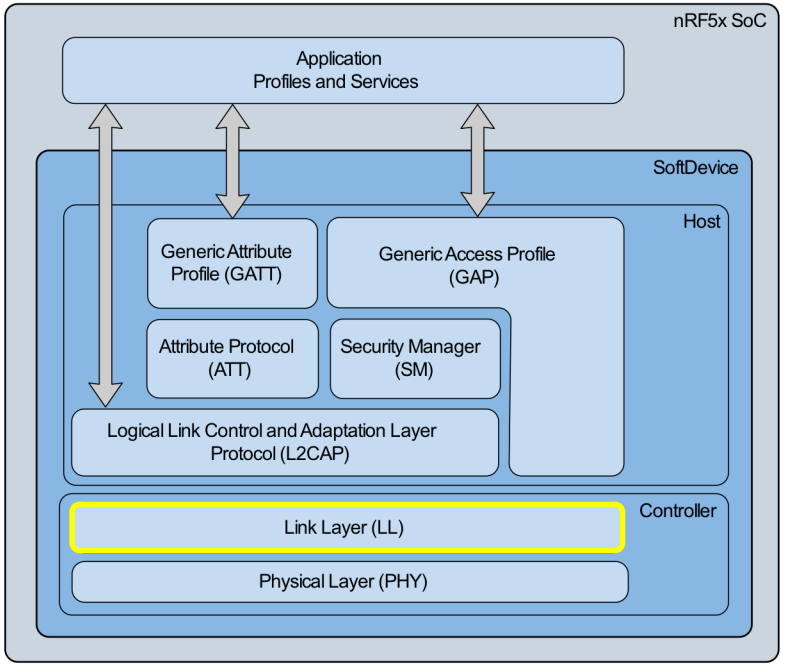


3

LINK LAYER

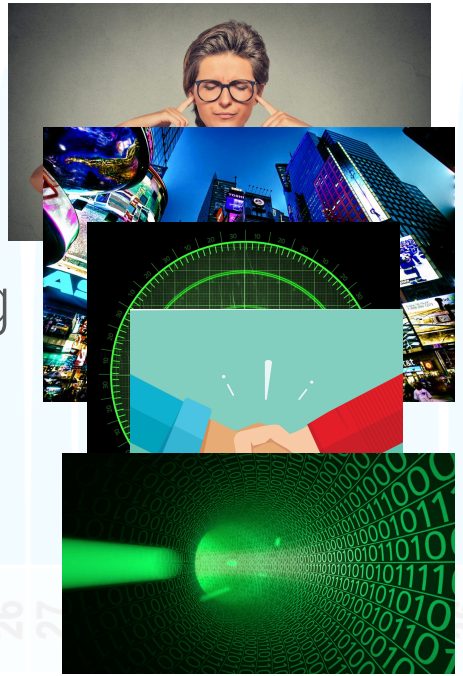
Protocol and format

LINK LAYER


nRF5x SoC
Application
Profiles and Services
SoftDevice
Host
Generic Attribute
Profile (GATT)
Generic Access Profile
(GAP)
Attribute Protocol
(ATT)
Security Manager
(SM)
Logical Link Control and Adaptation Layer
Protocol (L2CAP)
Controller
Link Layer (LL)
Physical Layer (PHY)

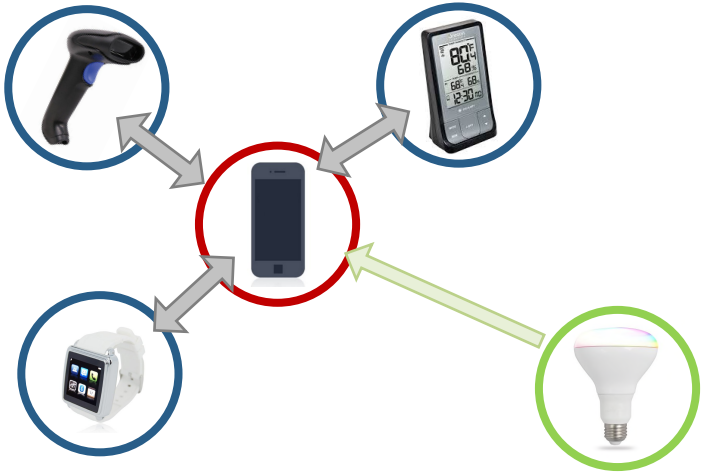
OPERATING STATES

- Standby: No commincation
- Advertisement (beacon): 1-way broadcasting
- Scanning: Listen to advertising packets
- Initiating: Initiate connection to advertiser
- Connection: Slave – Master communication



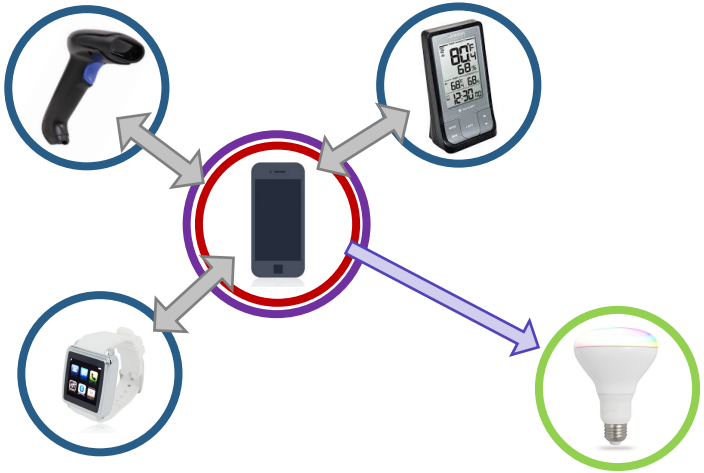
CONNECTION TOPOLOGY AND ROLES

- Master can have multiple link layer connections
- Slave can have only one link layer connection



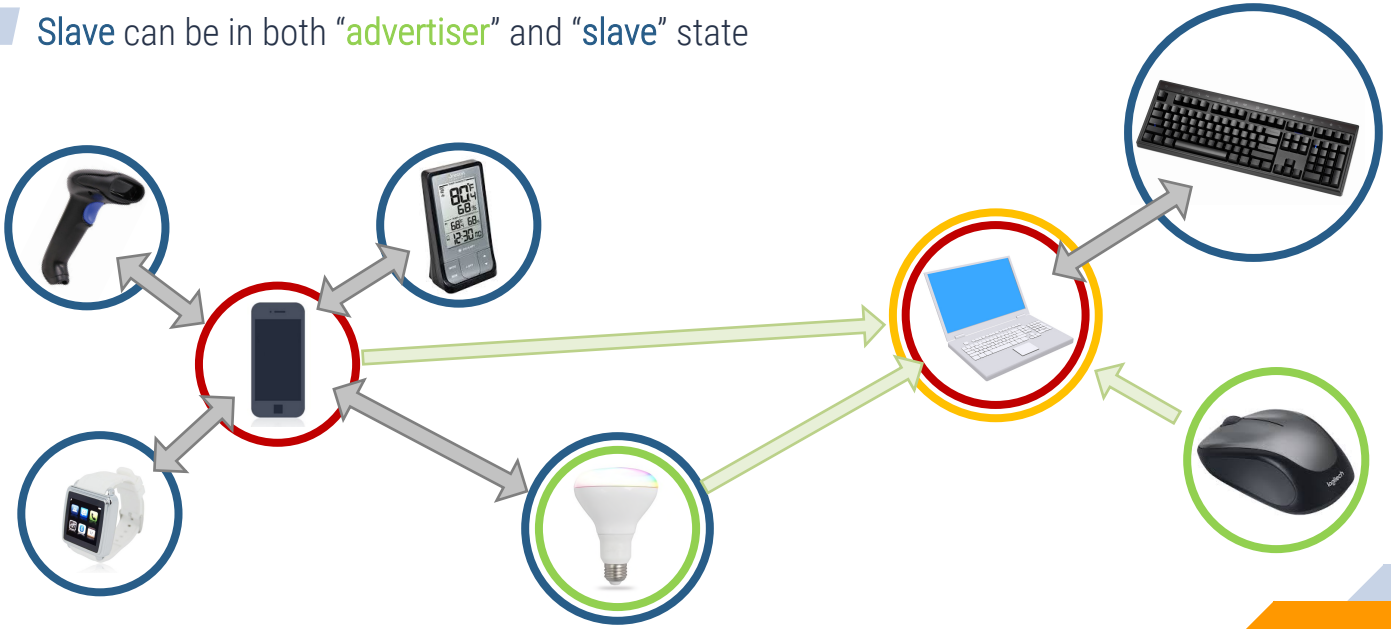
CONNECTION TOPOLOGY AND ROLES

- Master can be in both “scanning” and “master” state
- Master can be in both “initiator” and “master” state



CONNECTION TOPOLOGY AND ROLES

- Master can be in both “advertiser” and “master” state
- Slave can be in both “advertiser” and “slave” state

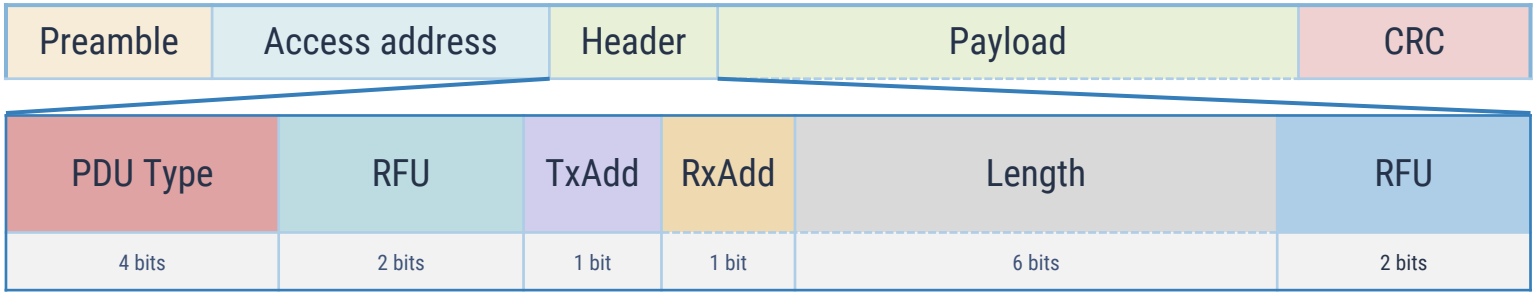


ADVERTISING PACKET STRUCTURE

Preamble	Access address	PDU	CRC
1 Byte	4 Bytes	2 - 37 Byte(s)	3 Bytes

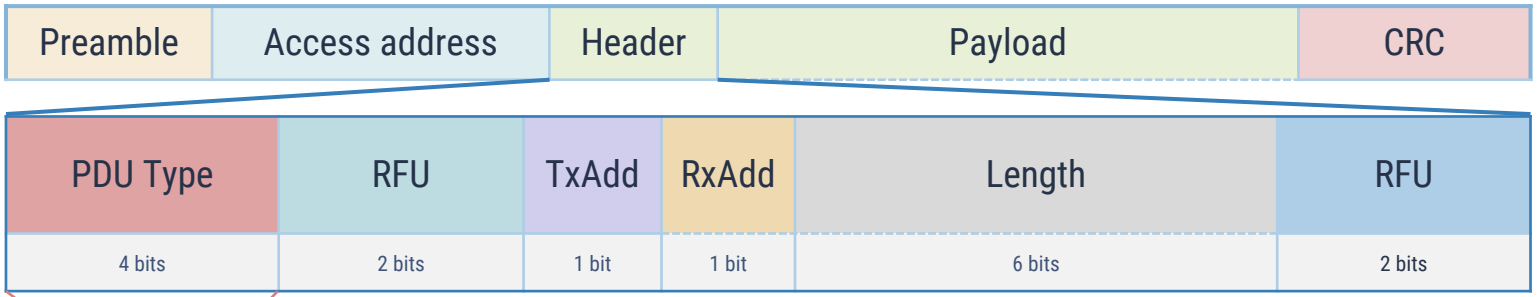
- **Preamble**: Used for sync and timing estimation (broadcast = `0xAA`)
- **Access address**: Broadcast (`0x8E89BED6`)
- **PDU**: Protocol Data Unit (header + payload)
- **CRC**: Cyclic Redundancy Check (error-detection)

ADVERTISING PDU



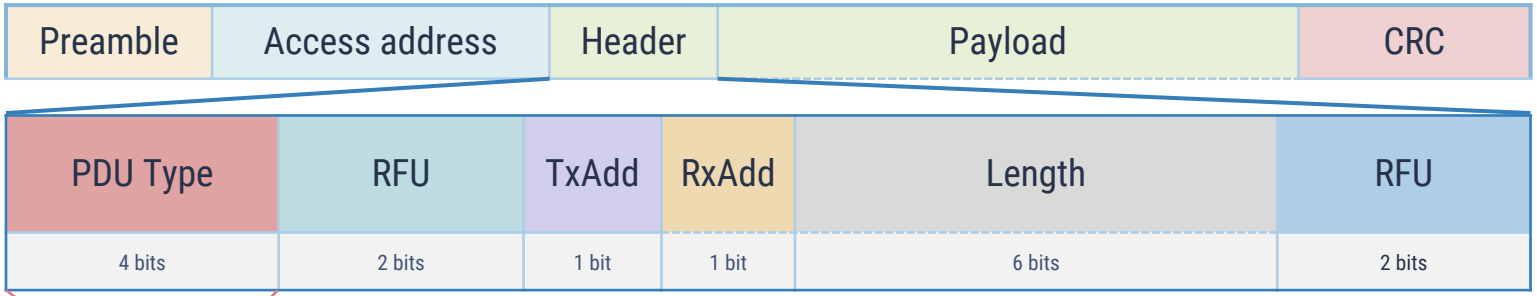
- **PDU type**: Indicate PDU type of advertisement
- **RFU**: Reserved for Future Use
- **TxAdd**: Indicates whether the advertiser‘s address (in payload) is public or random
- **RxAdd**: N/A for advertising
- **Length**: Define the size of the payload in Bytes (6 – 37 Bytes)

ADVERTISING PDU TYPES



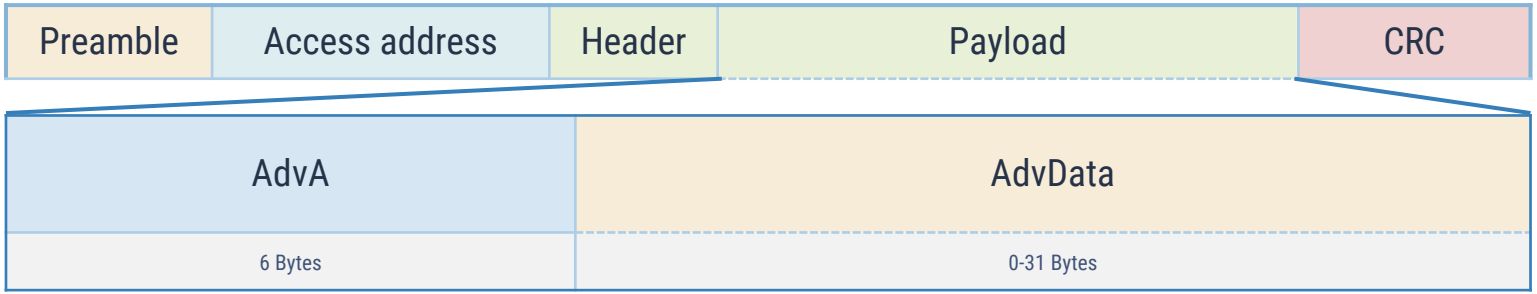
- 0000 - `ADV_IND`: *Advertising Indications*, a peripheral device requests connection to any central device
- 0001 - `ADV_DIRECT_IND`: Similar to ADV_IND, yet the connection request is directed at a specific central device.
- 0010 - `ADV_NONCONN_IND`: Non connectable devices, advertising information to any listening device.
- 0110 - `ADV_SCAN_IND`: Similar to ADV_NONCONN_IND, with the option additional information via scan responses.

ADVERTISING PDU TYPES



- 0011 - `SCAN_REQ`: Sent by *scanner* in active scanning
- 0100 - `SCAN_RSP`: Sent by *advertiser* in response to `SCAN_REQ`
- 0101 - `CONNECT_REQ`: Sent by *initiator* (*master*) to initiate connection
- other - `RESERVED`

ADVERTISING PDU



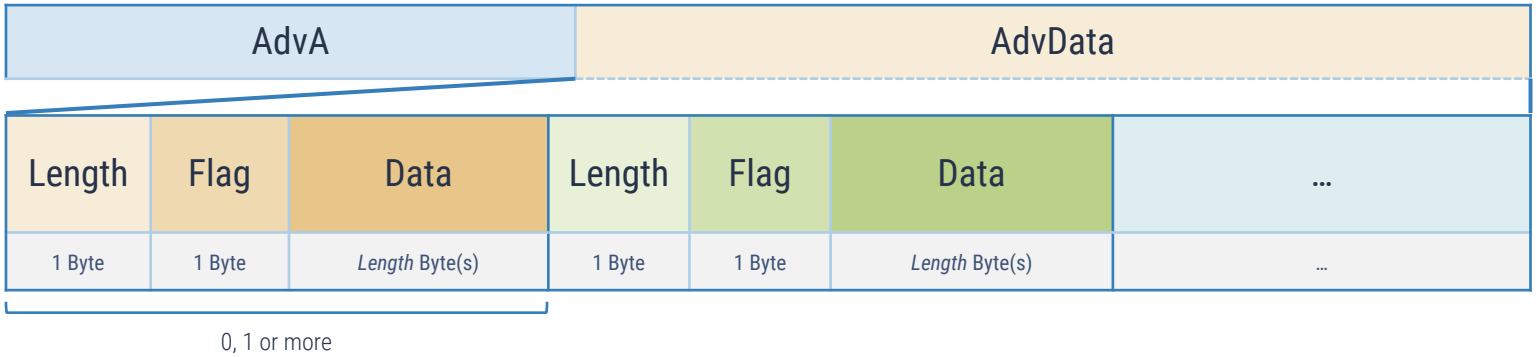
- **AdvA**: Advertising address (device address)
- **AdvData**: Data

ADVERTISING PDU

Advertising address (AdvA)

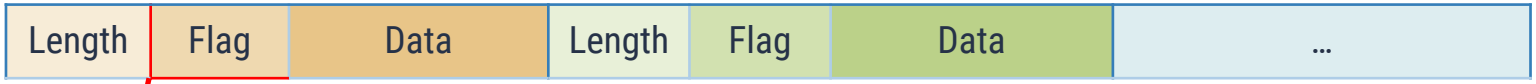
- **Public**: Like MAC address, manufacturer receive an Organizational Unique Identifer (OUI) than append a unique number for the device
- **Random**: In order to prevent devices to be traceable, Advertising address can be random
  - Static: The address is randomly generated when the device is powered on
  - Private non-resolvable: The address is random and can change over time (withing the same power cyble)
  - Private resolvable: The address is random and can change over time (within the same power cyble). The generated address can be used to derive the true address

ADVERTISING PDU



- **Length**: Length of this data (including flag)
- **Flag**: Advertisement data type
- **Data**: Advertisement data

ADVERTISING PDU



- `0x01` **Flags**: Device discovery capabilities
- `0x02–0x07` **Service UUID**: Device GATT services
- `0x08–0x09` **Local name**: Device name
- `0x0A` **TX power level**: Device output power
- `0xFF` **Manufacturer specific data**: User defined

ADVERTISING PDU - FLAGS

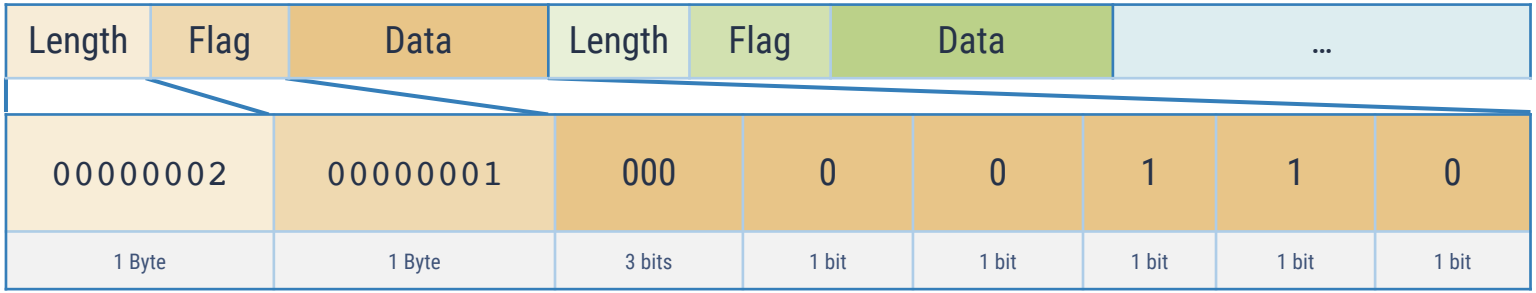
Length	Flag	Data	Length	Flag	Data	...

0x02	0x01	RFU	*SLEBRH**	*SLEBRC**	*BRNS**	*LEGDM**	*LELDM**
1 Byte	1 Byte	3 bits	1 bit	1 bit	1 bit	1 bit	1 bit

* *not official name*

- RFU: Reserved for future use
- SLEBRH: Simultaneous LE and BR/EDR (Host)
- SLEBRC: Simultaneous LE and BR/EDR (Controller)
- BRNS: BR/EDR Not Supported
- LEGDM: LE General Discoverable Mode
- LELDM: LE Limited Discoverable Mode

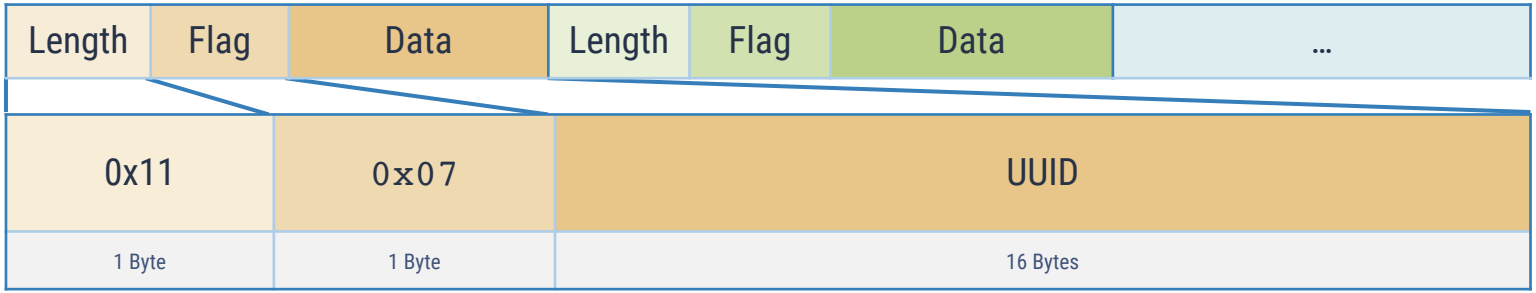
ADVERTISING PDU



* not official name

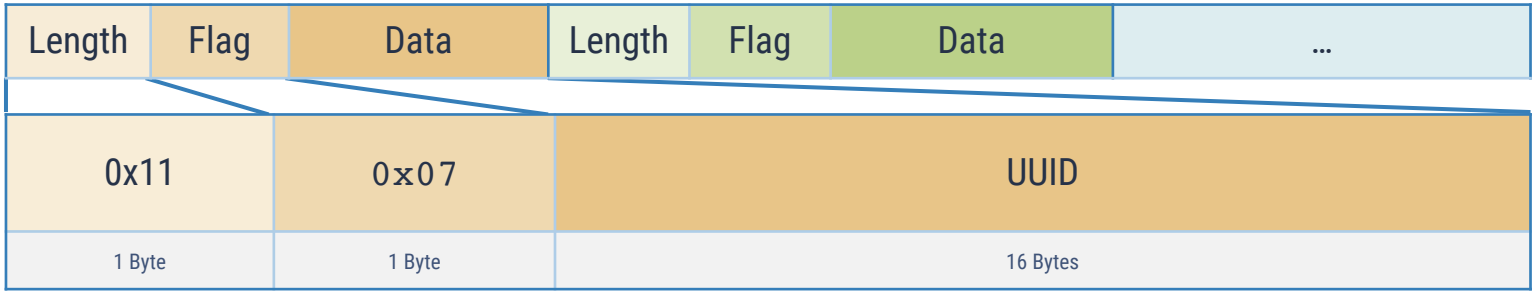
```
▾ Flags
    Length: 2
    Type: Flags (0x01)
    000. .... = Reserved: 0x0
    ...0 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): false (0x0)
    .... 0... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): false (0x0)
    .... .1.. = BR/EDR Not Supported: true (0x1)
    .... ..1. = LE General Discoverable Mode: true (0x1)
    .... ...0 = LE Limited Discoverable Mode: false (0x0)
```

ADVERTISING PDU – SERVICE UUID



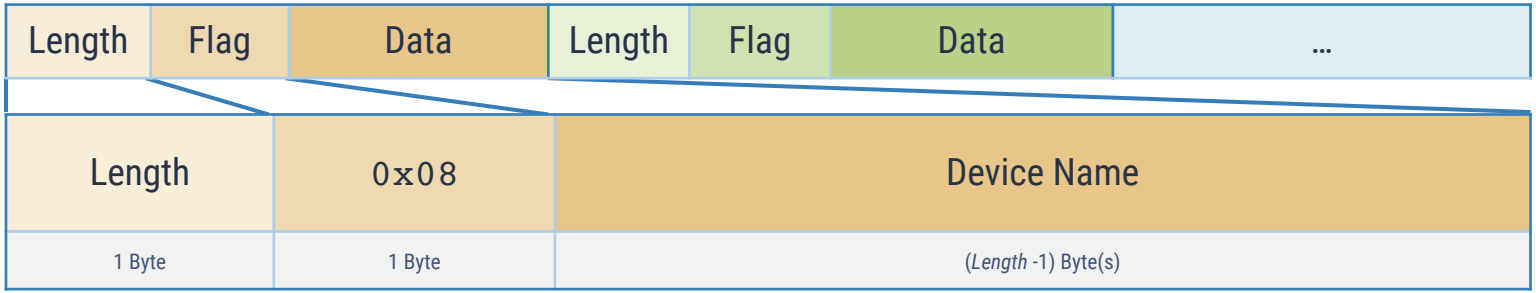
- **UUID**: Universally Unique Identifier of a GATT service (explained later)

ADVERTISING PDU – SERVICE UUID



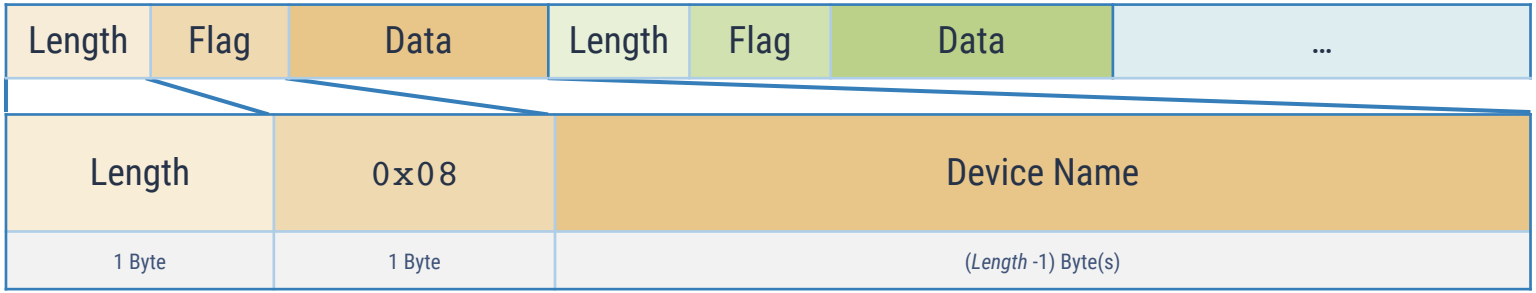

```
▾ 128-bit Service Class UUIDs
    Length: 17
    Type: 128-bit Service Class UUIDs (0x07)
    Custom UUID: dab91435-b5a1-e29c-b041-bcd562613bde (Unknown)
```

ADVERTISING PDU – DEVICE NAME



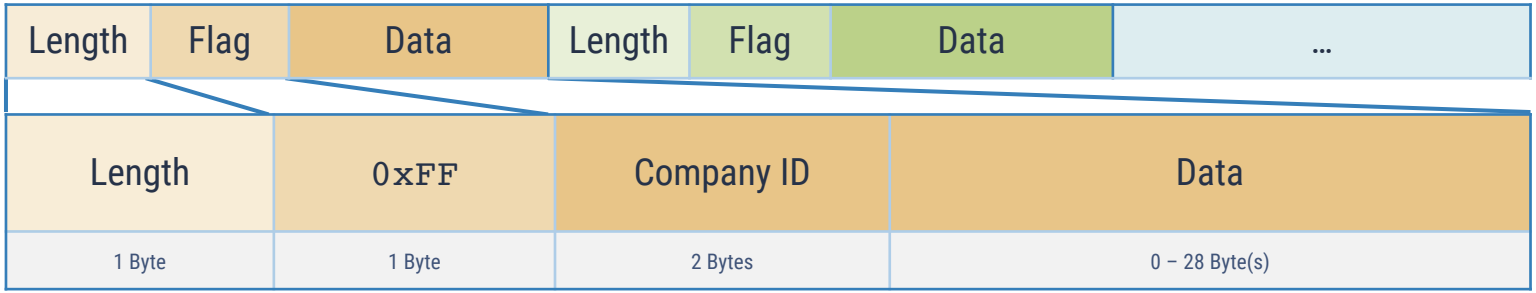
- **Device Name**: Name of the device (as listed when scanned). In this case (flag = 0x08), it's a shortened version. The name is not `NULL` terminated.

ADVERTISING PDU – DEVICE NAME



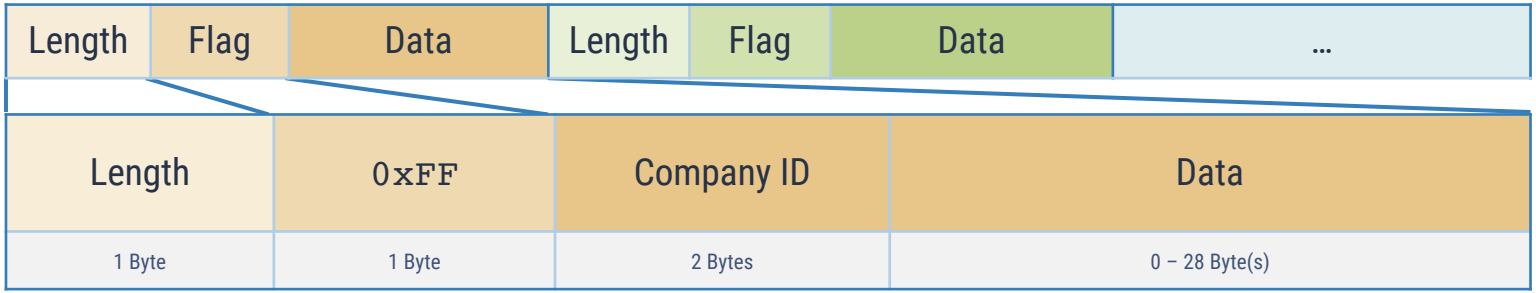

```
Device Name (shortened): Hello
  Length: 6
  Type: Device Name (shortened) (0x08)
  Device Name: Hello
```

ADVERTISING PDU – MANUFACTURER SPECIFIC



- **Company ID**: Company identifiers are unique numbers assigned by the Bluetooth SIG to member companies requesting one. E.g. Apple = `0x004c`, Samsung = `0x0075`.
- **Data**: The Manufacturer-Specific Data can contain any user-defined information

ADVERTISING PDU – MANUFACTURER SPECIFIC

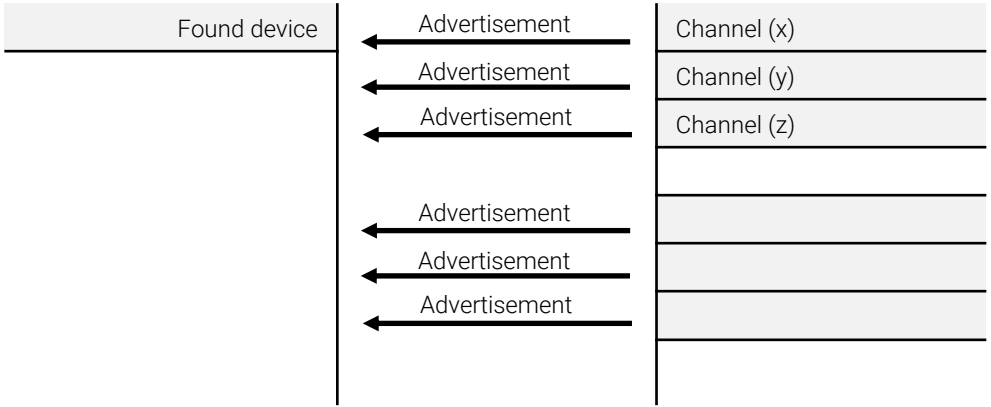



```
Advertising Data
  Manufacturer Specific
      Length: 27
      Type: Manufacturer Specific (0xff)
      Company ID: Samsung Electronics Co. Ltd. (0x0075)
    ▸ Data: 42040180aef8042ec1f99dfa042ec1f99c01000000000000
```

PASSIVE SCANNING

Scanner (e.g. smartphone)

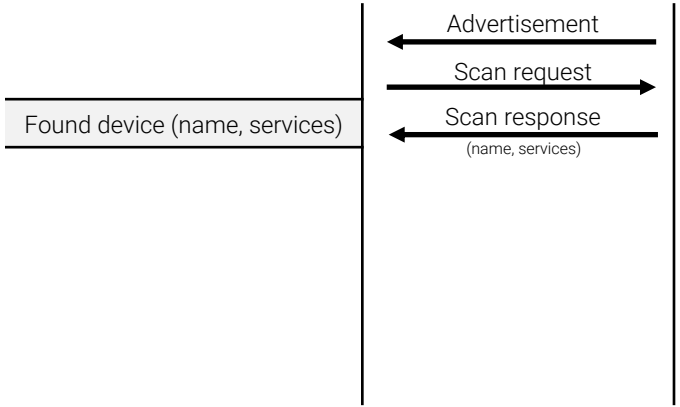
Advertiser (e.g. smart watch)



ACTIVE SCANNING

Scanner (e.g. smartphone)

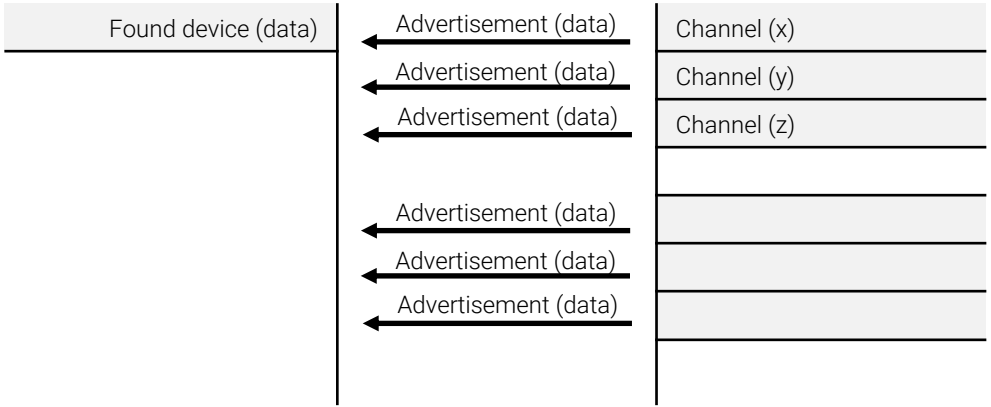
Advertiser (e.g. smart watch)



BROADCASTING DATA

Scanner (e.g. smartphone)

Advertiser (e.g. smart watch)

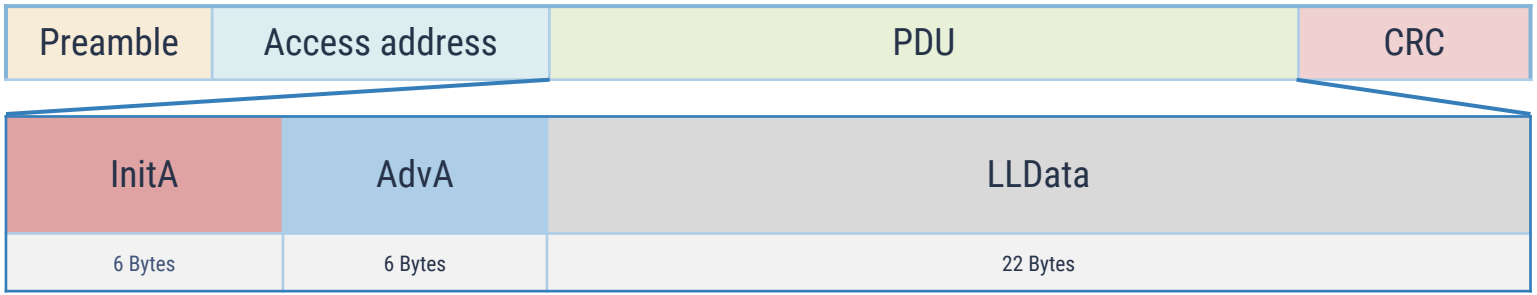


CONNECTION REQUEST PACKET STRUCTURE

Preamble	Access address	PDU	CRC
1 Byte	4 Bytes	0 - 37 Byte(s)	3 Bytes

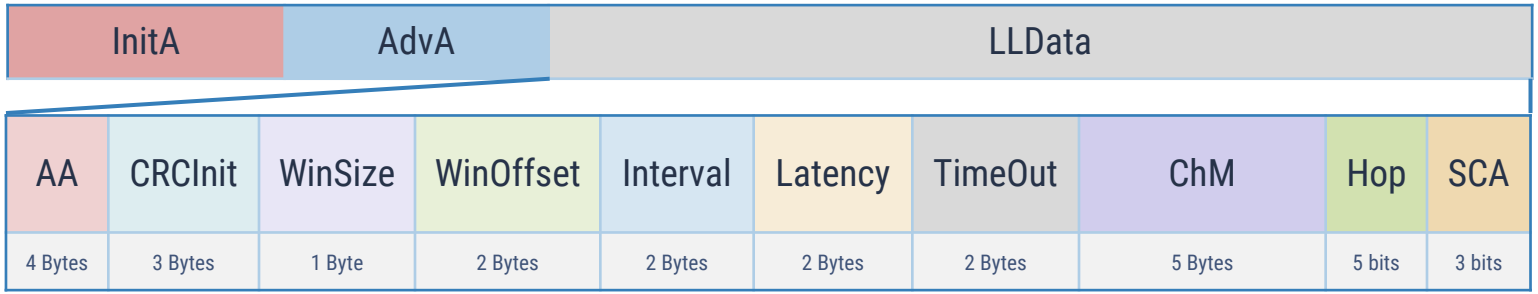
- **Preamble**: Used for sync and timing estimation (broadcast = `0xAA`)
- **Access address**: Broadcast (`0x8E89BED6`)
- **PDU**: Protocol Data Unit
- **CRC**: Cyclic Redundancy Check (error-detection)

CONNECTION REQUEST PACKET STRUCTURE



- InitA: Initiators public or random device address
- AdvA: Advertising devices public or random device address
- LLData: Link Layer Data

CONNECTION REQUEST PACKET STRUCTURE

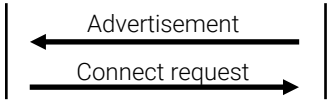


- **AA**: The link layer connection's access address
- **CRCInit**: Initialization value for the CRC calculation
- **WinSize**: The *transmitWindowSize* value
- **WinOffset**: The *transmitWindowOffset* value
- **Interval**: Connection interval of data connection
- **Latency**: The *connSlaveLatency* value
- **Timeout**: The *connSupervisionTimeout* value
- **ChM**: Channel map which indicates Used and Unused data channels. LSB represents data channel index 0. A bit value of 0 indicates that the channel is Unused and a bit value of 1 indicates that the channel is Used.
- **Hop**: hopIncrement used in the data channel selection algorithm. Random value in the range of 5 to 16.
- **SCA**: used to determine the worst case Master's sleep clock accuracy.

INITIATING CONNECTION

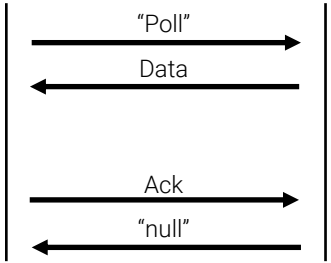
Initiator (e.g. smartphone)

Advertiser (e.g. smart watch)



Master (e.g. smartphone)

Slave (e.g. smart watch)

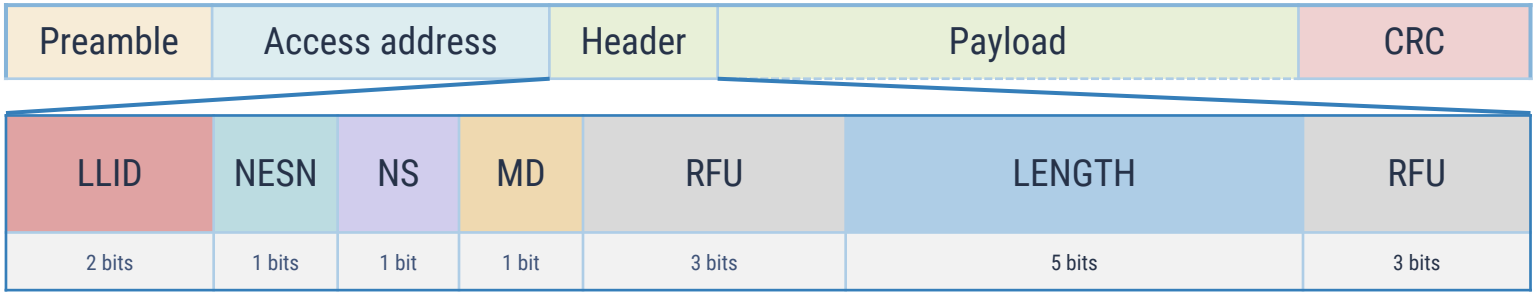


DATA CHANNEL PACKET STRUCTURE

Preamble	Access address	PDU	CRC
1 Byte	4 Bytes	0 - 37 Byte(s)	3 Bytes

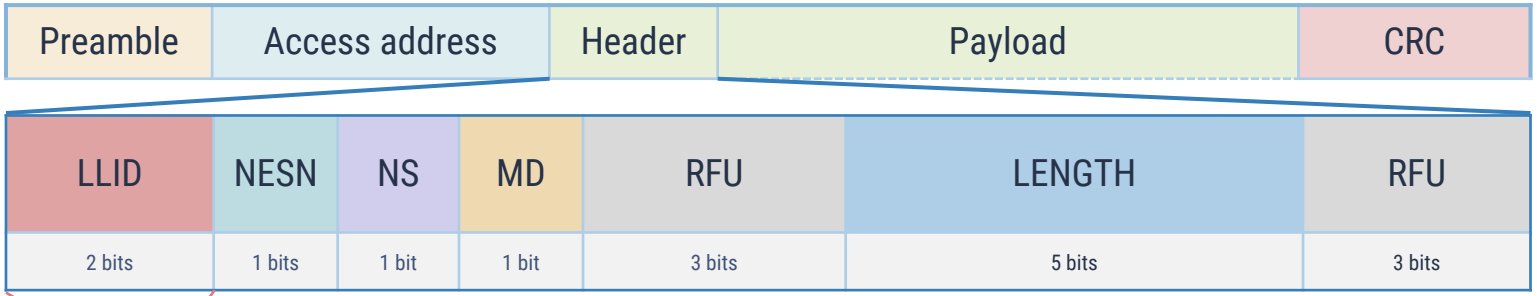
- **Preamble**: Used for sync and timing estimation (`0xAA` or `0x55`)
- **Access address**: Different for each link layer connection (see connection packet)
- **PDU**: Protocol Data Unit
- **CRC**: Cyclic Redundancy Check (error-detection)

DATA CHANNEL PACKET STRUCTURE



- **LLID**: Indicates whether the packet is an LL Data PDU or an LL Control PDU
- **NESN**: Next expected Sequence Number
- **NS**: Sequence Number
- **MD**: More Data
- **RFU**: Reserved for Future Use
- **LENGTH**: Indicates the length of the payload and MIC (if included) in Bytes

DATA CHANNEL PACKET STRUCTURE



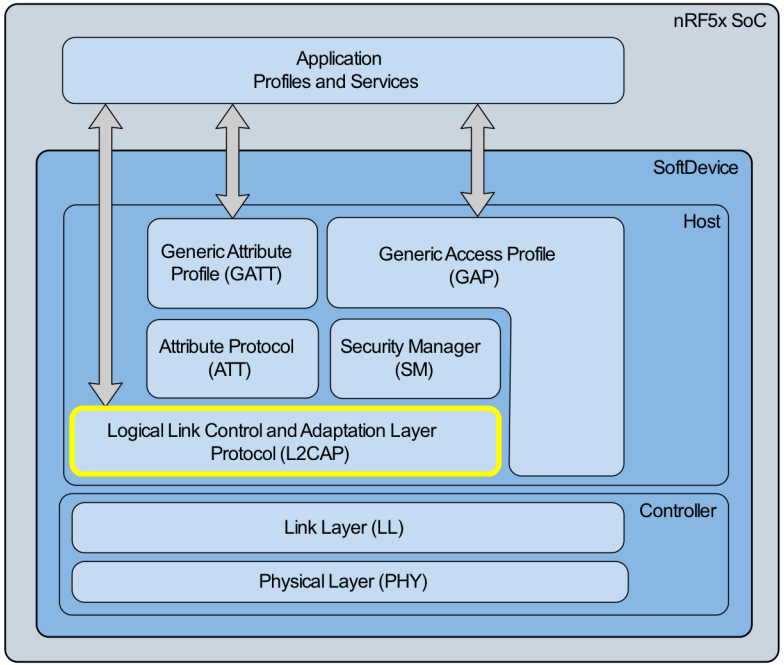
- 00 - `RESERVED`
- 01 - `LL Data PDU`: Continuation fragment of an L2CAP message or empty PDU
- 10 - `LL Data PDU`: Start of an L2CAP message
- 11 - `LL Control PDU`

4

LOGICAL LINK CONTROL & ADAPTATION PROTOCOL

Protocol and format

LOGICAL LINK CONTROL & ADAPTATION PROTOCOL

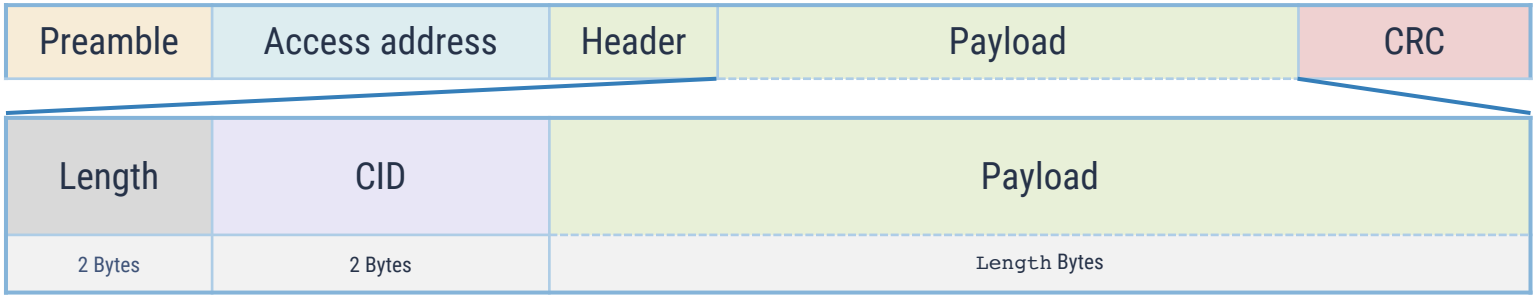


L2CAP

Logical Link Control and Adaptation Protocol (L2CAP)

- All application data is sent using L2CAP
- Communication model based on *channels*
  - A channel represents a data flow between L2CAP entities in remote devices
  - Logical channels multiplexes various protocols/services over the same physical link
  - Channels could have fixed specification or dynamic on the need basis
- *Connection-oriented* and *connection-less* data services
  - *Connection-oriented*: connection between two devices, where a CID identifies each endpoint of the channel
  - *Connection-less*: data flow restricted to a single direction
- Limited MTU (Max Transmission Unit): segmentation and re-assembly

L2CAP PACKET FORMAT



- **Length**: Length of the L2CAP Payload
- **CID**: Either fixed channels or connection oriented channels
- **Payload**: Protocol Data Unit

L2CAP – CHANNEL IDENTIFIER

Channel IDentifier (CID)

- Sort of TCP/UDP port
- Local name representing a channel end-point
- One device cannot use the same CID for two different channels
- One CID could be use by multiple devices at the same time
- CIDs from 0x0001 to 0x003F have a fixed purpose
- CIDs from 0c0040 to 0xFFFF are dynamically allocated

CHANNEL IDENTIFIER

Main CID for BLE

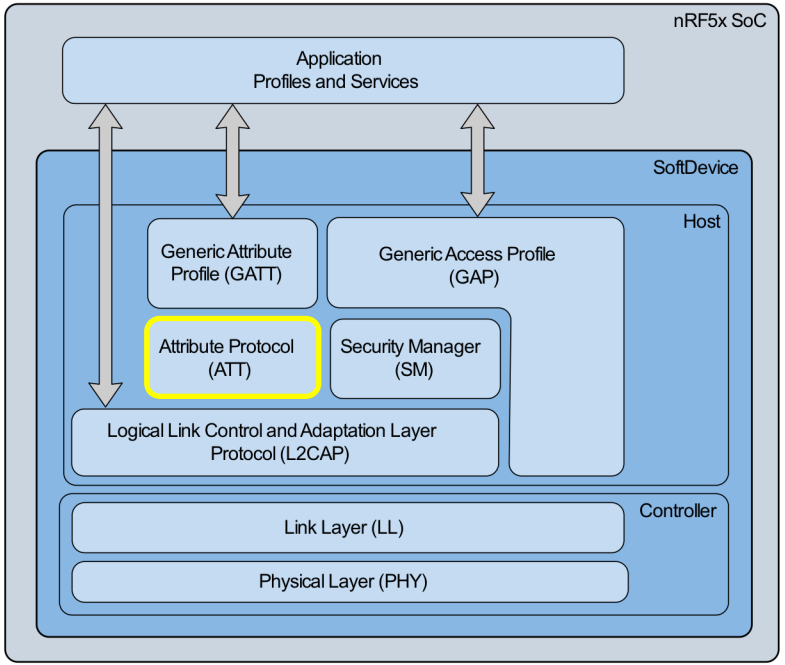
CID	Description
0x0004	Attribute protocol
0x0005	LE L2CAP signaling channel
0x0006	Security manager protocol

5

ATTRIBUTE PROTOCOL

Protocol and format

ATTRIBUTE PROTOCOL



ATTRIBUTE PROTOCOL

Attribute Protocol (ATT)

- Peer-to-peer protocol between *server* and *client*
  - *Server*: contains data (attributes)
  - Client: request data
- A device can implement both client and server roles
- Attribute structure
  - Handle: The ID used to select the attribute (when reading/writing/etc)
  - Type: The type of value (https://www.bluetooth.com/specifications/gatt/characteristics)
  - Value: Content of the attribue
  - Permission: controls whether they can be read or written or shall be sent over encrypted channel

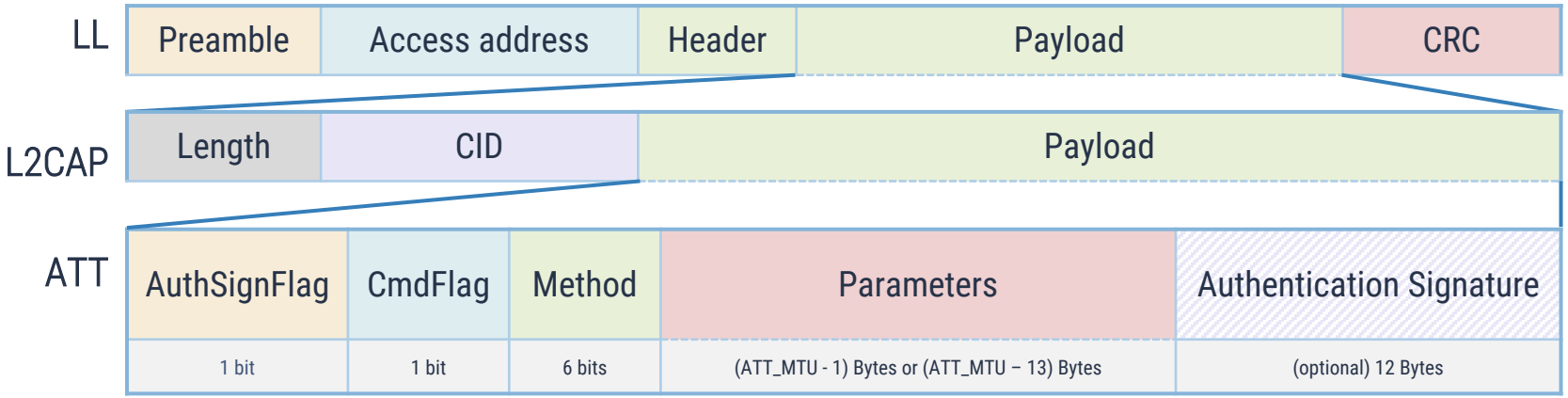
Handle	Type	Value	Permission
2 Bytes	2 or 16 Bytes	0 – 512 Byte(s)	Implementation specific

ATTRIBUTE TYPE

- Universal Unique IDentifier (UUID)
  - Bluetooth Base UUID: `0000xxxx-0000-1000-8000-00805F9B34FB`
  - `xxxx`: Assigned number (2 Bytes) officially adopted BLE Services
  - In order to save bits in ATT packet, types officially adopted by BLE Services are sent in its compressed 2 Bytes form
  - When custom types are used, the full UUID is sent

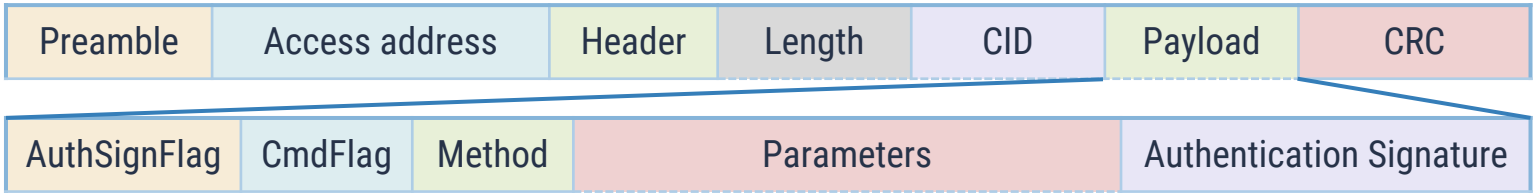
Name	Uniform Type Identifier	Assigned Number	Specification
Device Name	org.bluetooth.characteristic.gap.device_name	0x2A00	GSS
Appearance	org.bluetooth.characteristic.gap.appearance	0x2A01	GSS
Peripheral Privacy Flag	org.bluetooth.characteristic.gap.peripheral_privacy_flag	0x2A02	GSS
Reconnection Address	org.bluetooth.characteristic.gap.reconnection_address	0x2A03	GSS
Peripheral Preferred	org.bluetooth.characteristic.gap.peripheral_preferred_connection_parameters	0x2A04	GSS

ATT PACKET FORMAT



- **AuthSignFlag**: Presence of authentication signature
- **CmdFlag**: Command flag
- **Method**: Define the request type (find, read, write, etc)
- **Parameters**: Parameters for the request/response/…
- **Authentication signature** (optional): Only for write command

ATT METHOD TYPE



Methods

- **Request**: exchange MTU, read/write/find attribute
- **Response**: exchange MTU, read/write/find attribute
- **Command**: write attribute
- **Indication**: Indicate attribute‘s value
- **Confirmation**: Confirm that the client has received the indication
- **Notification**: attribute value (without prior request)

EXAMPLE OF METHODS

Methods

- **Find information**: obtain the mapping of all attribute handles between a given starting handle and ending handle
- **Read by group type**: obtain the values of attributes that matches a given group type. The grouping is defined by a higher layer specification
- **Read**: read an attribute‘s value that matches a given handle
- **Write**: write an attribute‘s value that matches a given handle
- **Signed write command**: write an attribute‘s value that matches a given handle (typically into a control-point attribute) with an authentication signature
- **Handle value**: Indicating attribute‘s value without client request

MORE EXAMPLE OF METHODS

Methods

- MTU exchange
- Find by type value
- Read blob
- Read multiple
- Read by type
- Signed write command
- Write command
- Prepare write
- Execute write
- Error

http://mutsughost1.github.io/2015/02/26/ATT-Spec/#Attribute_Protocol_PDUs

ATT PACKET FORMAT

	AuthSign Flag	Cmd Flag	Method	Parameters					
Find info req.	0	0	000100	First handle (2 Bytes)	Ending handle (2 Bytes)				
Find info resp.	0	0	000101	Format (1 Byte)	Attribute type (2 or 16 Bytes)				
Read by group type req.	0	0	010000	First handle	Ending handle	Attribute type			
Read by group type resp.	0	0	010001	Length (1 Byte)	Handle	End handle	Value	Handle	…	…
Read req.	0	0	001010	Handle (2 Bytes)					
Read req.	0	0	001011	Value					

ATT PACKET FORMAT

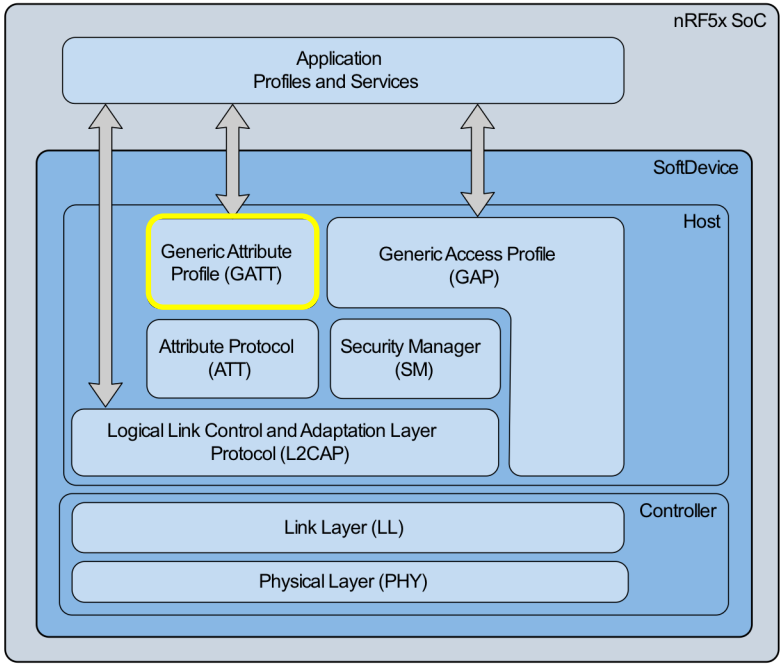
	AuthSign Flag	Cmd Flag	Method	Parameters		
Write request	0	0	010010	Handle (2 Bytes)		Value
Write response	0	0	010011			
Signed write command	1	1	010010	Handle	Value	Auth. signature (12 Bytes)
Handle value indication	0	0	011101	Handle		Value
Handle value notification	0	0	011110			

6

GENERIC ATTRIBUTE PROFILE

Protocol and format

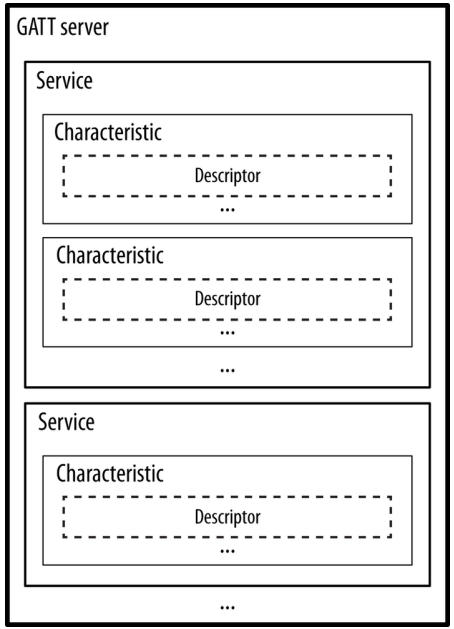
GENERIC ATTRIBUTE PROFILE (GATT)



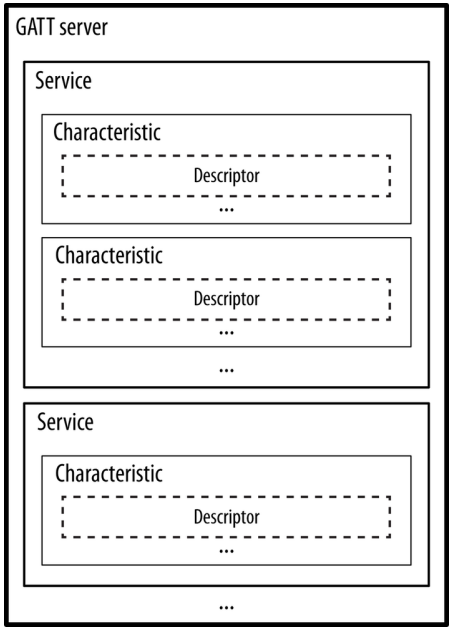
GATT DESCRIPTION

Generic Attribute Profile (GATT)

ATT protocol allows us to read/write/find attributes. Attributes can be organised in „services“, which group related data together. This faciliate the discoverability of attributes and the data structure. GATT defines the way the service are organized.

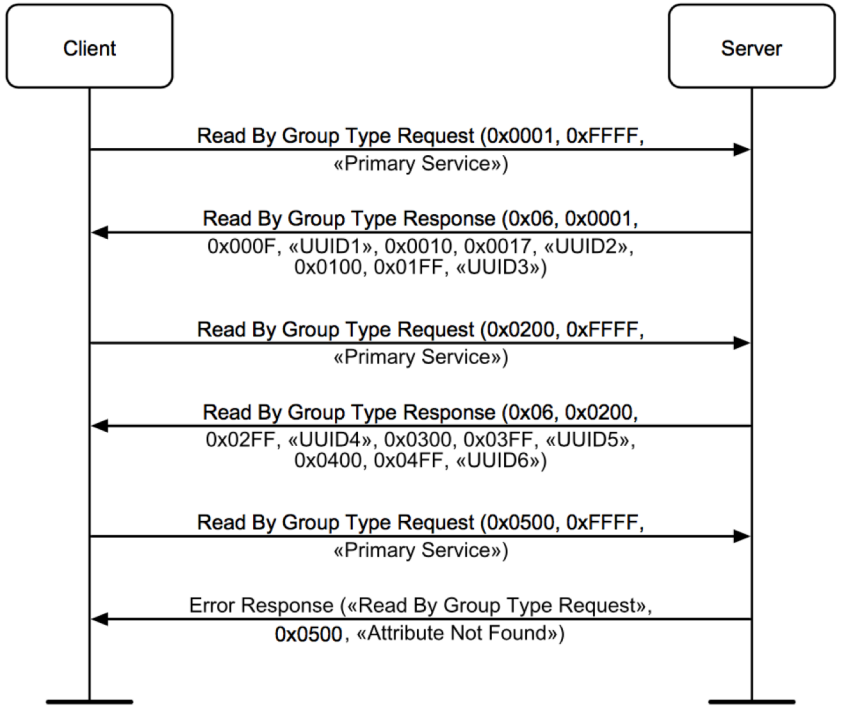


GATT STRUCTURE



- **Service**: GATT services group conceptually related attributes in one common section of the attribute information set in the GATT server
- **Characteristics**: You can understand characteristics as containers for user data. They always include at least two attributes: the *characteristic declaration* and the *characteristic value*
  - **Characteristics declaration**: standardized unique UUID used exclusively to denote the beginning of characteristics
  - **Characteristics value**: the actual user data
- **Descriptor**: mostly used to provide the client with metadata (additional information about the characteristic and its value)

GATT – FIND PRIMARY SERVICES



GATT - ATTRIBUTES LIST

Handle	UUID	Value		
0x0001	0x2800 (service)	0x180D (Heart rate)		
0x0003	0x2803 (char)	NOT	0x0004	0x2a37 (HRM)
0x004	0x2A37 (HRM)	0x1234 (BPM)		
0x009	0x2902 (CCCD)	0x0001 (NOT)		
0x00D	0x2803 (char)	RO	0x000f	0x2a38 (BSL)
0x000f	0x2a38 (BSL)	0x4321 (finger)		
0x0010	0x2800 (service)	0x1810 (Blood pressure)		
0x0012	0x2803 (char)	NOT	0x0013	0x2a35 (BPM)
…	…	…	…	…

GATT - ATTRIBUTES LIST

Handle	UUID	Value		
0x0001	0x2800 (service)	0x180D (Heart rate)		
0x0003	0x2803 (char)	NOT	0x0004	0x2a37 (HRM)
0x004	0x2A37 (HRM)	0x1234 (BPM)		
0x009	0x2902 (CCCD)	0x0001 (NOT)		
0x00D	0x2803 (char)	RO	0x000f	0x2a38 (BSL)
0x000f	0x2a38 (BSL)	0x4321 (finger)		
0x0010	0x2800 (service)	0x1810 (Blood pressure)		
0x0012	0x2803 (char)	NOT	0x0013	0x2a35 (BPM)
…	…	…	…	…

Service

GATT - ATTRIBUTES LIST

Handle	UUID	Value		
0x0001	0x2800 (service)	0x180D (Heart rate)		
0x0003	0x2803 (char)	NOT	0x0004	0x2a37 (HRM)
0x004	0x2A37 (HRM)	0x1234 (BPM)		
0x009	0x2902 (CCCD)	0x0001 (NOT)		
0x00D	0x2803 (char)	RO	0x000f	0x2a38 (BSL)
0x000f	0x2a38 (BSL)	0x4321 (finger)		
0x0010	0x2800 (service)	0x1810 (Blood pressure)		
0x0012	0x2803 (char)	NOT	0x0013	0x2a35 (BPM)
...	...	...	...	...

Service

Characteristic

GENERIC ATTRIBUTE PROFILE

Handle	UUID	Value		
0x0001	0x2800 (service)	0x180D (Heart rate)		
0x0003	0x2803 (char)	NOT	0x0004	0x2a37 (HRM)
0x004	0x2A37 (HRM)	0x1234 (BPM)		
0x009	0x2902 (CCCD)	0x0001 (NOT)		
0x00D	0x2803 (char)	RO	0x000f	0x2a38 (BSL)
0x000f	0x2a38 (BSL)	0x4321 (finger)		
0x0010	0x2800 (service)	0x1810 (Blood pressure)		
0x0012	0x2803 (char)	NOT	0x0013	0x2a35 (BPM)
...	...	...	...	...

Service

Characteristic

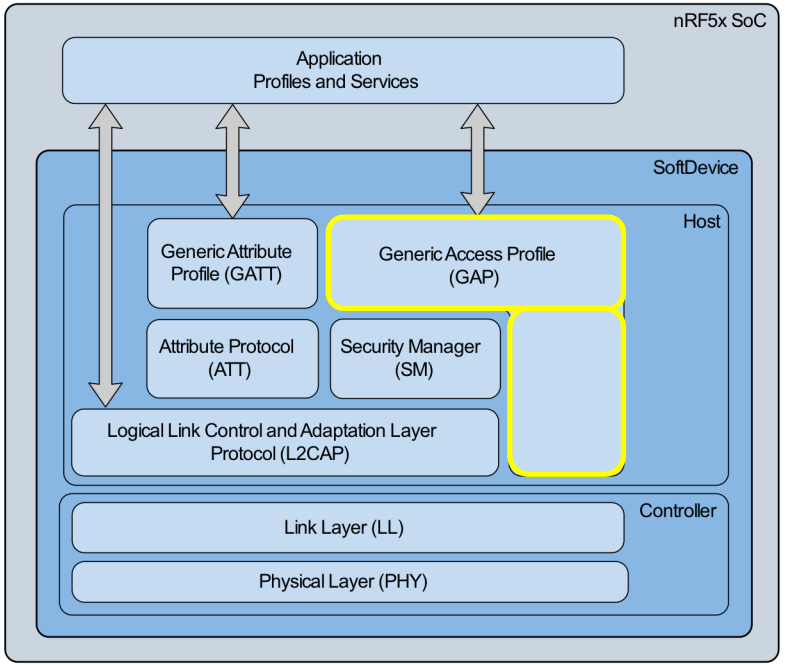
Descriptor

7

GENERIC ACCESS PROFILE

Protocol and format

GENERIC ACCESS PROFILE



GENERIC ACCESS PROFILE

Generic access profile (GAP)

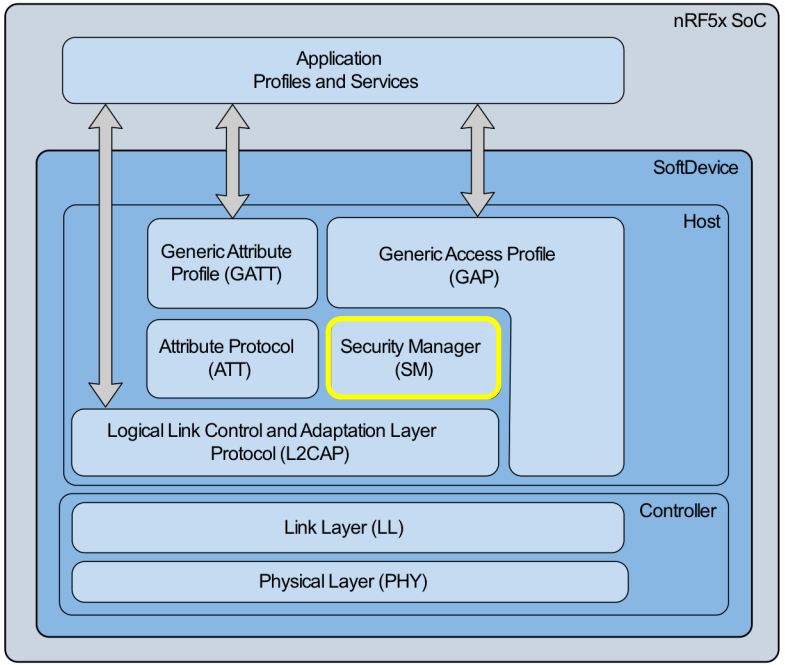
- All Bluetooth device should have a GAP service in their GATT server
- UUID for GAP is `0x1800`
- GAP contains the following characteristics
  - **Device name**: name of the device as a UTF-8 string
  - **Appearance**: define how to represent the device to the user, i.e. using an icon, a string, or similar
  - **Peripheral preferred connection parameters**: connection parameters such as connection interval, timeout, etc
  - **Central address resolution**: Used for Resolvable Private Address
  - **Resolvable private address only**: Status whether it is using RPA

8

SECURITY MANAGER

Protocol and format

SECURITY MANAGER



SECUITY MANAGER

Security in Bluetooth Low Energy

- Eavesdropping protection
- Man-in-the-middle protection
- Privacy of devices
- Security shared
  - Link Layer: encryption and authentication
  - Security Manager: security protocol (L2CAP CID = `0x0006`)

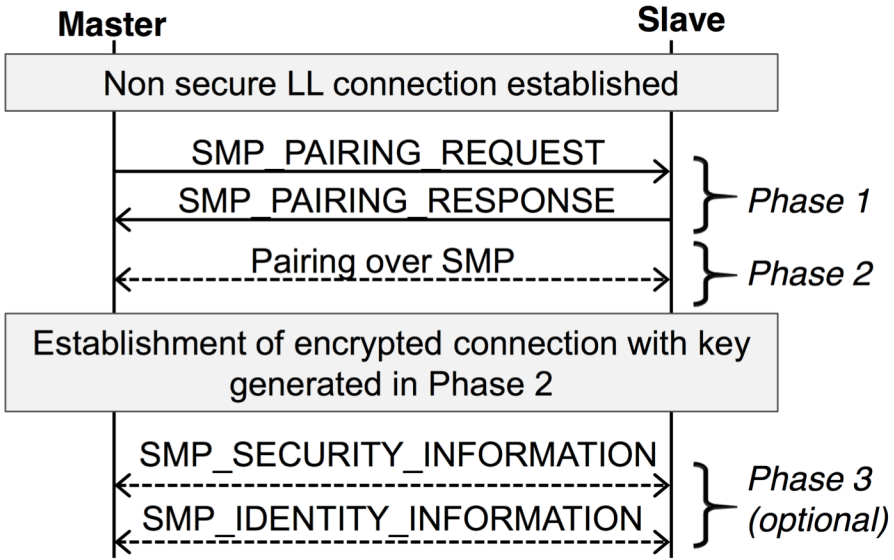
SECUITY MANAGER – SECURITY LEVEL

- **Security Level 1** without security at all
- **Security Level 2** AES-CMAC encryption (aka AES-128 via RFC 4493, which is FIPS-compliant) during communications when the devices are unpaired
- **Security Level 3** supports encryption and requires pairing
- **Security Level 4** supports encryption and requires pairing but instead of AES-CMAC for encryption, ECDHE (aka Elliptic Curve Diffie-Hellman aka P-256, which is also FIPS-compliant) is used instead

SECUITY MANAGER – SECURITY MODE

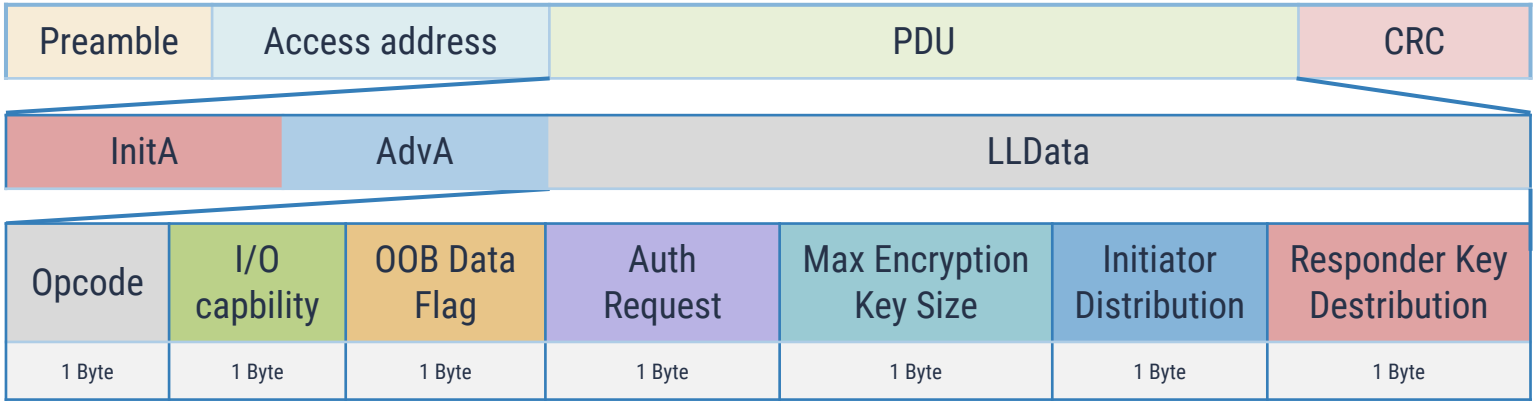
- Security Mode 1 securiy levels without signing of data
- Security Mode 2 security levels with signing of data, including both paired and unpaired communications.
- Mixed Security Mode is when a device is required to support both Security Mode 1 and 2, i.e., it needs to support signed and unsigned data.

SECUITY MANAGER – PAIRING



- **Phase 1:** exchange security capabilities.
- **Phase 2:** devices agree on a Temporary Key (TK) in order to generate the Short Term Key (STK)
- **Phase 3**: generate Long Term Key (LTK), Connection Signature Resolving Key (CSRK) for signing and Identity Resolving Key (IRK) for address generation and lookup. STK is used to distribute the new keys.

PAIRING – PHASE 1



Phase One

Exchange security capabilities.

PAIRING – PHASE 1

Opcode	I/O capbility	OOB Data Flag	Auth Request	Max Encryption Key Size	Initiator Distribution	Responder Key Destribution
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Opcode

- `0x01` pairing request
- `0x02` pairing response

PAIRING – PHASE 1

Opcode	I/O capbility	OOB Data Flag	Auth Request	Max Encryption Key Size	Initiator Distribution	Responder Key Destribution
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

I/O capability

- `0x00` Display Only
- `0x01` Display Yes/No (both a display and a way to designate yes or no)
- `0x02` Keyboard Only
- `0x03` No Input/No Output (e.g. headphones)
- `0x04` Keyboard Display (both a keyboard and a display screen)
- `0x05` - `0xFF` Reserved

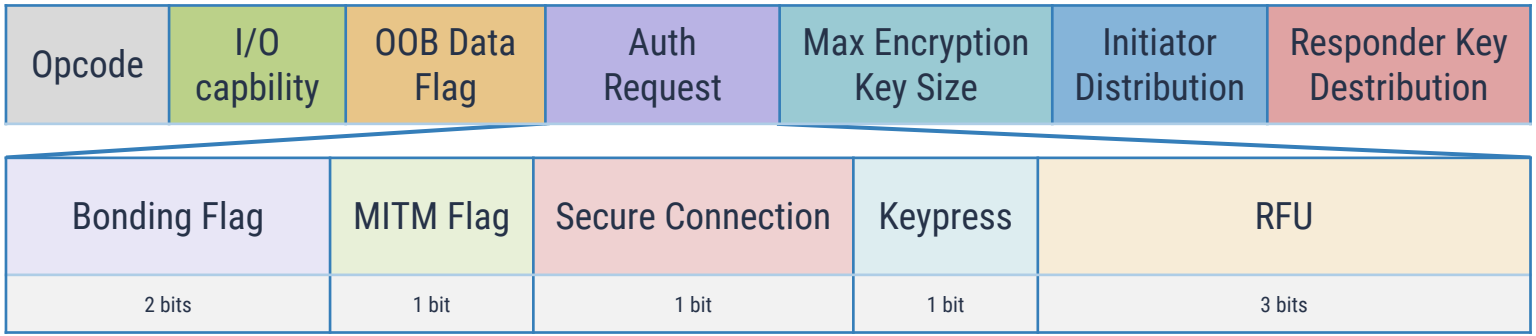
PAIRING – PHASE 1

Opcode	I/O capbility	OOB Data Flag	Auth Request	Max Encryption Key Size	Initiator Distribution	Responder Key Destribution
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

OOB (out-of-band) Data Flag

- `0x00` no OOB data
- `0x01` OOB data presence

PAIRING – PHASE 1



- **Bonding flags**: bonding is the exchange of long-term keys after pairing occurs. `0x00` is for no bonding, `0x01` is for bonding. If bonding is used, it means two devices can be paired, and a reboot or sleep mode will not unpair the devices.
- **MITM flag**: `0x00` is that MITM protection is not requested, `0x01` is that MITM protection is requested.
- **Secure connection:** if this is set to 1, the device is requesting to do Secure Connection Only Mode, otherwise it is set to 0.
- **Keypress**: if set to 1 it means Passkey Entry needs to be used, otherwise it is ignored.
- **RFU**: Reserved for future use

PAIRING – PHASE 1

Opcode	I/O capbility	OOB Data Flag	Auth Request	Max Encryption Key Size	Initiator Distribution	Responder Key Destribution
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

- **Encryption Key Size**: size of the encryption key in octets
- **Initiator Distribution**: contains flags that states what keys will be distributed
- **Responder Kez Distribution**: contains flags that states what keys will be distributed

PAIRING – PHASE 1 – PAIRING METHODS

- Numeric Comparison. Both devices indicate the same six digit identifier on their screens/LCS displays. If it matches, you confirm the connection. This is not meant to prevent MitM attack, but only verify you are pairing the intended devices together.
- Just Works. Whenever you don't have a screen/display (e.g. headphone), Just Works method will use the numeric comparison with the six digits set to `000000`. While Numeric Comparison requires some on-the-fly math if you are performing a MITM attack, there is no MITM protection with Just Works.

PAIRING – PHASE 1 – PAIRING METHODS

- Passkey Entry. Instead of just validating if the two identifier match, with passkey, the user is requested to enter manually the six digit code, which should be displayed on the other device.
- Out Of Band (OOB). A communication method outside of the Bluetooth communication channel is not used, but the information is still secured. The Apple Watch is a good example of this workflow. During the Apple Watch pairing method, a swirling display of dots is displayed on the watch face, and you point the pairing iPhone's camera at the watch face while (according to Apple) bits of information are transmitted via this process. Another example is using Near Field Communication (NFC) between NFC-capable headphones and a pairing phone.

PAIRING – PHASE 1 – PAIRING METHODS

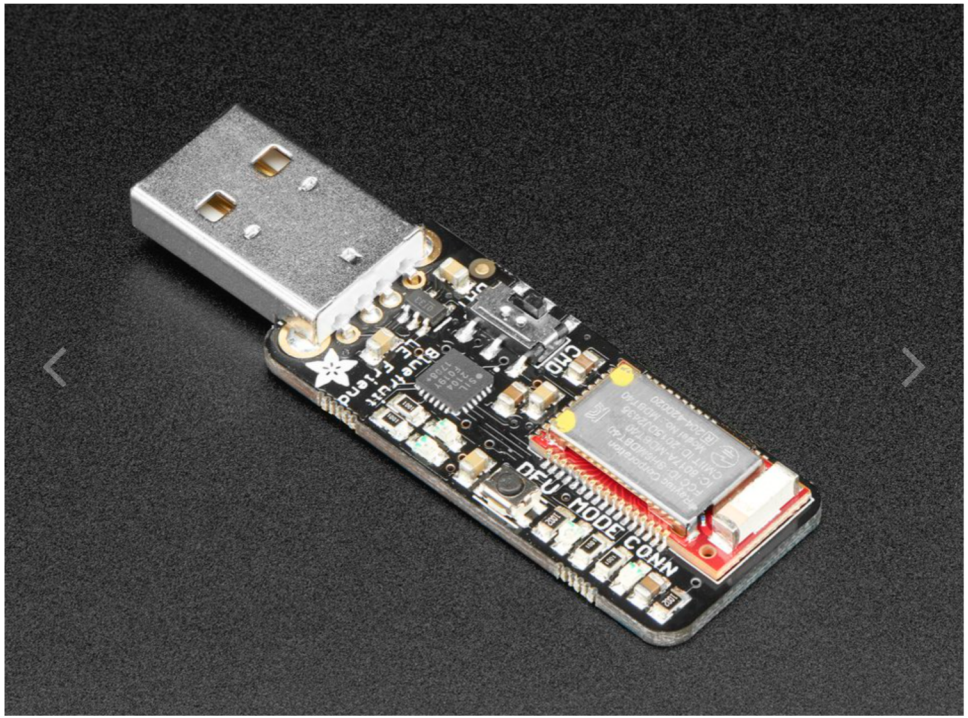
- Passkey Entry. Instead of just validating if the two identifier match, with passkey, the user is requested to enter manually the six digit code, which should be displayed on the other device.
- Out Of Band (OOB). A communication method outside of the Bluetooth communication channel is not used, but the information is still secured. The Apple Watch is a good example of this workflow. During the Apple Watch pairing method, a swirling display of dots is displayed on the watch face, and you point the pairing iPhone's camera at the watch face while (according to Apple) bits of information are transmitted via this process. Another example is using Near Field Communication (NFC) between NFC-capable headphones and a pairing phone.

9

DEMO

Let’s sniff and RE some BLE

BLE SNIFFER



Bluefruit LE Sniffer - Bluetooth Low Energy (BLE 4.0) - nRF51822 - Firmware Version 2

PRODUCT ID: 2269

$24.95 (34€ on Amazon)

IN STOCK

1 ADD TO CART

QTY	DISCOUNT
1-9	$24.95
10-99	$22.46
100+	$19.96

ADD TO WISHLIST

DESCRIPTION

TECHNICAL DETAILS

LEARN

RESOURCES

- http://blog.bluetooth.com/bluetooth-low-energy-it-starts-with-advertising
- http://www.ti.com/lit/an/swra475a/swra475a.pdf
- http://www.fte.com/docs/Ble_101_frontline.pps
- https://github.com/riotrf/RIOT.RF/wiki/Link-Layer-Packet-Format
- https://e2echina.ti.com/cfs-file/__key/telligent-evolution-components-attachments/00-103-01-00-00-03-93-04/handouts_5F00_WSNs_5F00_BT_2D00_LE.pdf
- https://datatracker.ietf.org/meeting/interim-2016-t2trg-02/materials/slides-interim-2016-t2trg-2-7
- http://mutsughost1.github.io/2015/02/26/ATT-Spec/
- Getting Started with Bluetooth Low Energy (O'Reilly)
- https://duo.com/decipher/understanding-bluetooth-security