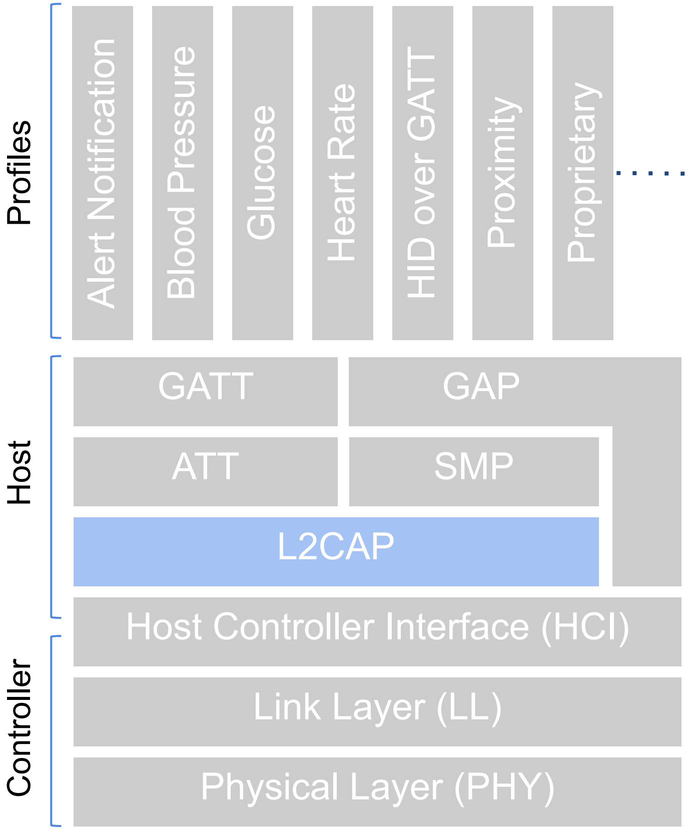
UNIVERSIDAD
COMPLUTENSE
MADRID

Bluetooth Low Energy - 2

Networks and Protocols 1

Facultad de Informática

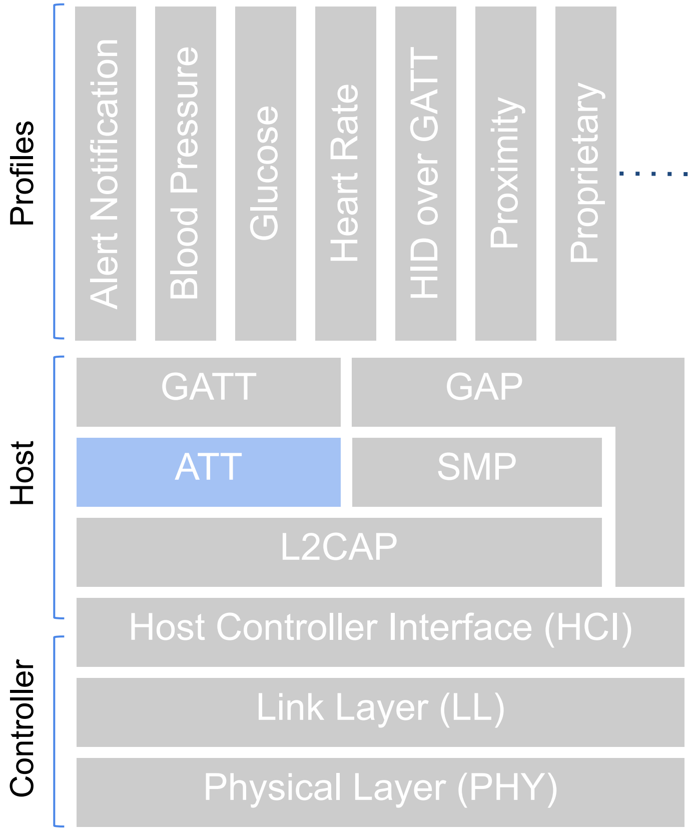
L2CAP


Profiles
Alert Notification
Blood Pressure
Glucose
Heart Rate
HID over GATT
Proximity
Proprietary
......
Host
GATT
GAP
ATT
SMP
L2CAP
Host Controller Interface (HCI)
Controller
Link Layer (LL)
Physical Layer (PHY)

L2CAP

- Creates logical channels
  - From/To a pair of services in a pair of devices
  - Two devices can maintain several active channels at the same time
- Static channels
  - Exist once two devices create a connection
  - Do not require additional configuration
- Dynamic channels
  - Have a channel id per connection
- 16-bit channel ids
  - 0x0004 used for the ATT protocol (static)
  - 0x0006 used for the SMP protocol (static)
  - 0x0040 – 0xFFFF for connection oriented channels (dynamic)

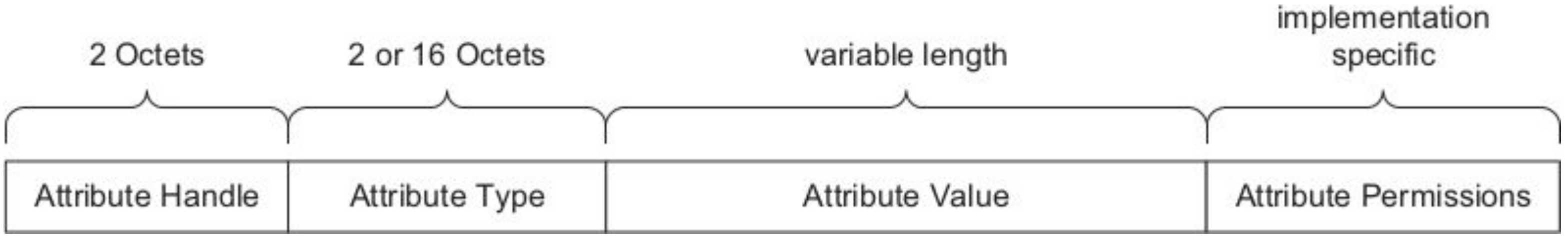
Attribute Protocol (ATT)



Attribute Protocol (ATT)

- Provides an attribute interchange service in a connection
- Client-server protocol model
  - The server exposes an attribute table
  - The client can discover, read and/or write those attributes
  - The server can also send notifications or indications of his attributes
- The ATT client sends commands, requests and confirmations to the ATT server
- The ATT server sends responses, notifications or indications to the ATT client
- A command can be used to read, write or discover the attributes of the server
- The maximum payload size is 23 bytes

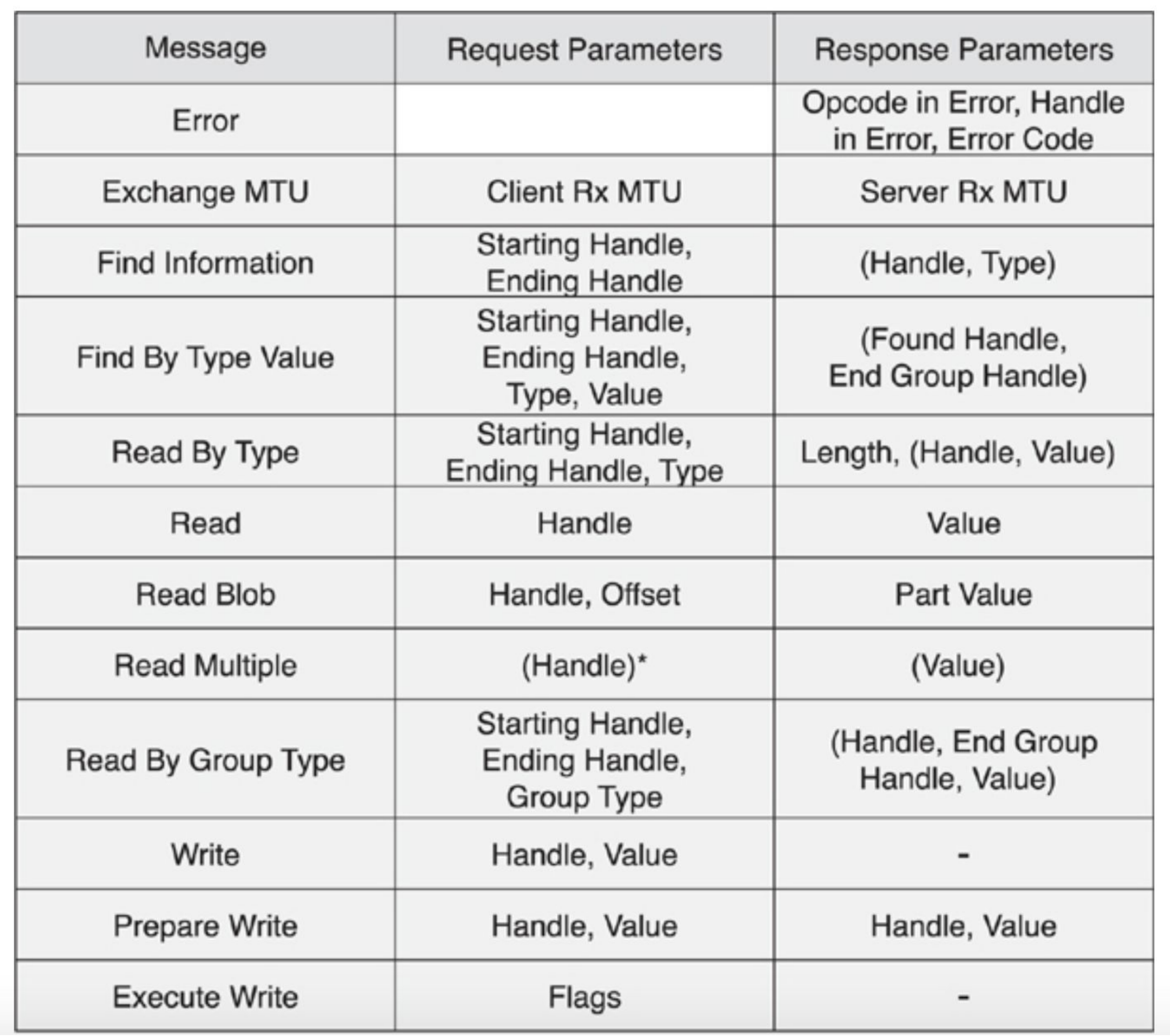
Attributes



2 Octets	2 or 16 Octets	variable length	implementation specific
Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions

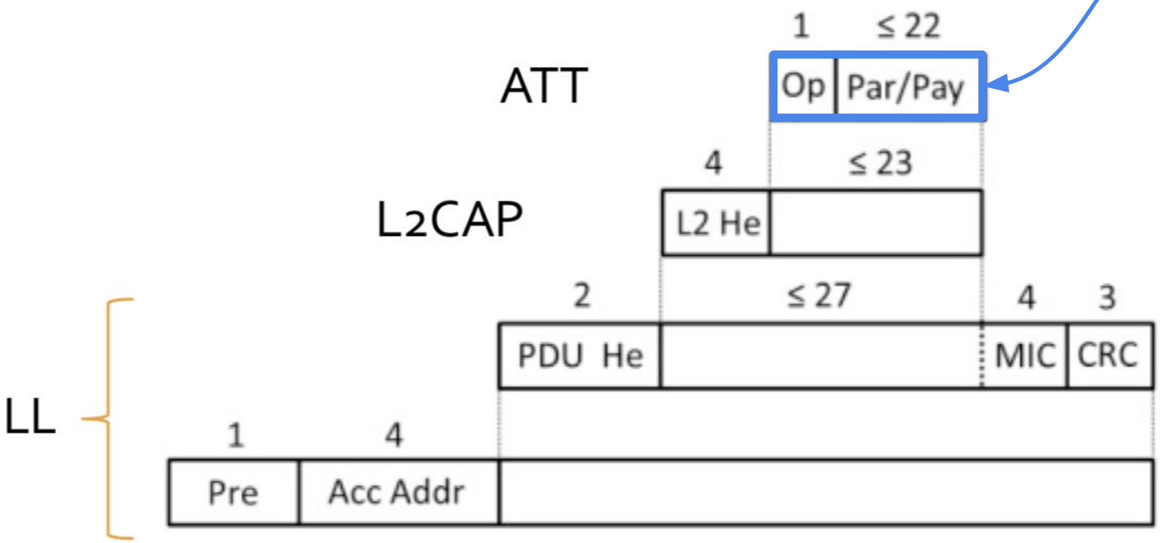
- An attribute is a addressable labeled value
  - Each device has several attributes stored as a table
  - GATT services and characteristics are represented as attributes
- *Value:* represents the exposed data
- *Handle*: is the attribute identifier/address in the device
- *Type*: encodes what type of data is the Value (temp, pressure…)
  - Expressed with a Universally Unique Identifier (UUID)
  - Registered types can be encoded with only 2 bytes
- *Permissions*: permitted accesses and security requirements
  - Cannot be accessed using ATT
  - Write only Attributes are control points

ATT: some available messages



Message	Request Parameters	Response Parameters
Error		Opcode in Error, Handle in Error, Error Code
Exchange MTU	Client Rx MTU	Server Rx MTU
Find Information	Starting Handle, Ending Handle	(Handle, Type)
Find By Type Value	Starting Handle, Ending Handle, Type, Value	(Found Handle, End Group Handle)
Read By Type	Starting Handle, Ending Handle, Type	Length, (Handle, Value)
Read	Handle	Value
Read Blob	Handle, Offset	Part Value
Read Multiple	(Handle)*	(Value)
Read By Group Type	Starting Handle, Ending Handle, Group Type	(Handle, End Group Handle, Value)
Write	Handle, Value	-
Prepare Write	Handle, Value	Handle, Value
Execute Write	Flags	-

Packet formation

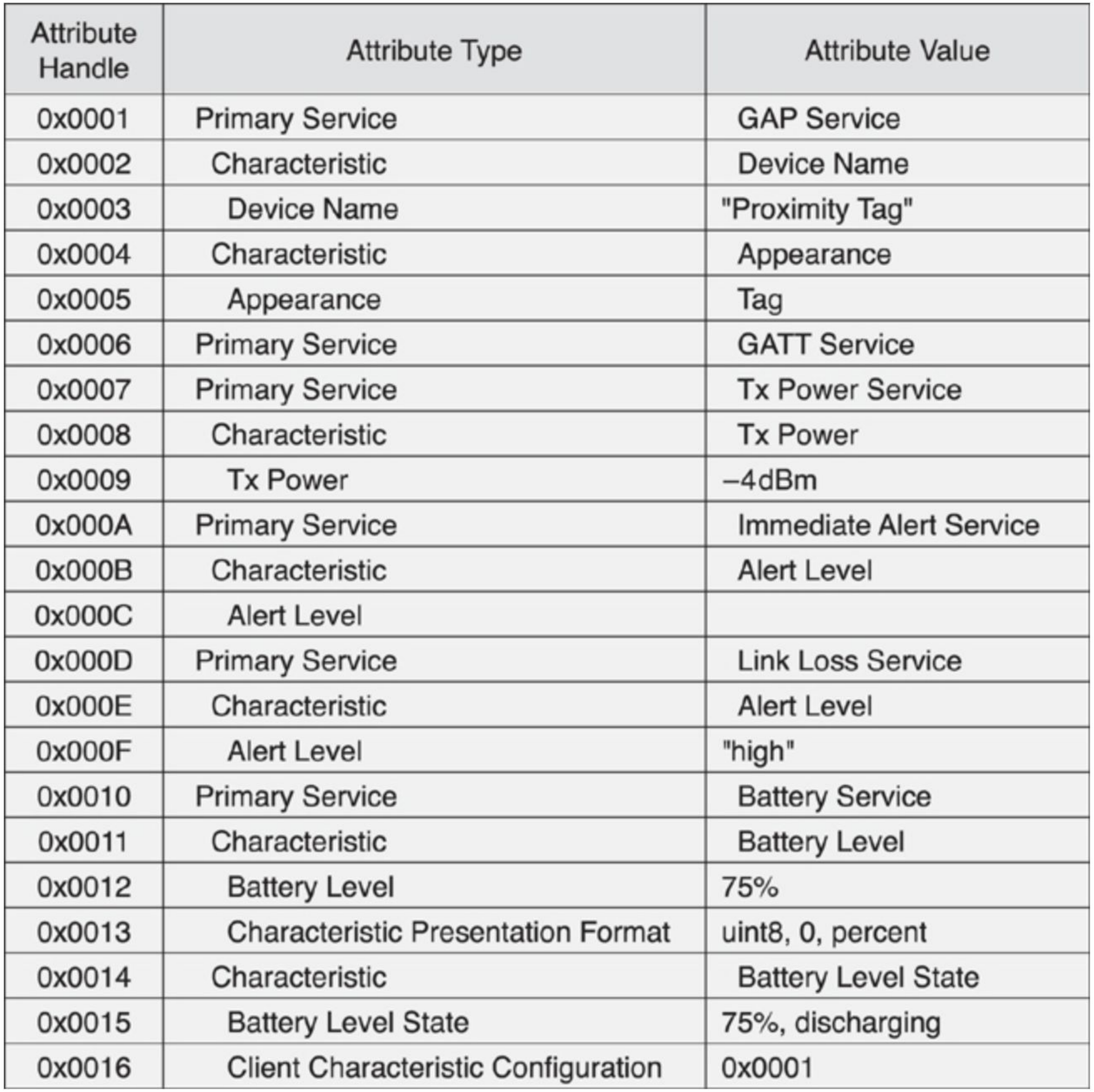


Formatos defined in section 3.4 vol 3 of the BT5.0 standard

Par/Pay:	Parameters and Payload
Op:	ATT Opcode
PDU He:	PDU Header
L2 He:	L2CAP Header
Acc Addr:	Access Address
Pre:	Preamble
MIC:	Message Integrity Check
CRC:	Cyclic Redundancy Check

Structure of the attribute table

- The ATT server stores the attributes as a table
- The first 6 rows are mandatory, available in all ble servers



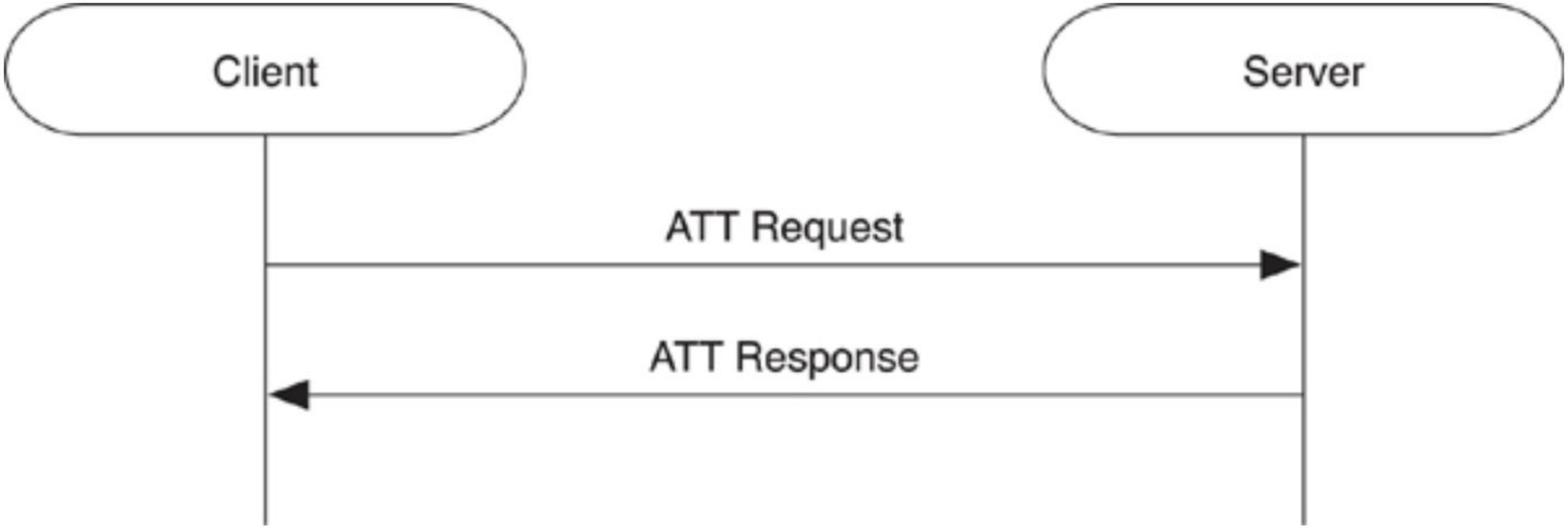
Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	Appearance
0x0005	Appearance	Tag
0x0006	Primary Service	GATT Service
0x0007	Primary Service	Tx Power Service
0x0008	Characteristic	Tx Power
0x0009	Tx Power	−4dBm
0x000A	Primary Service	Immediate Alert Service
0x000B	Characteristic	Alert Level
0x000C	Alert Level	
0x000D	Primary Service	Link Loss Service
0x000E	Characteristic	Alert Level
0x000F	Alert Level	"high"
0x0010	Primary Service	Battery Service
0x0011	Characteristic	Battery Level
0x0012	Battery Level	75%
0x0013	Characteristic Presentation Format	uint8, 0, percent
0x0014	Characteristic	Battery Level State
0x0015	Battery Level State	75%, discharging
0x0016	Client Characteristic Configuration	0x0001

ATT Protocol: basic operations

- Request
- Response
- Command
- Indication
- Confirmation
- Notification

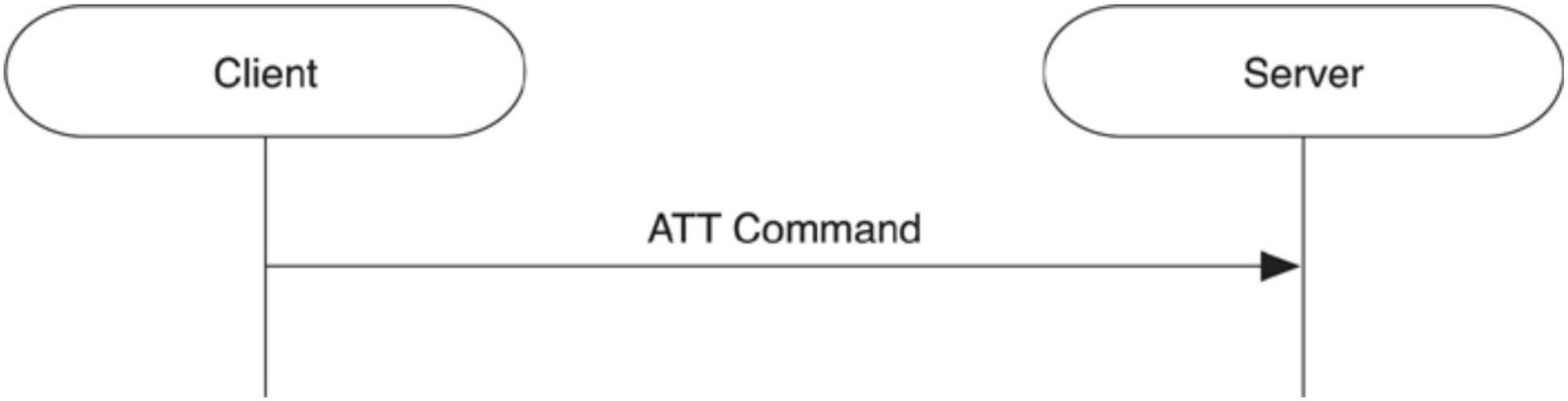
ATT: Request

- A client sends a Request to request some information from the server, and waits for his response
  - Requests are sent one by one
  - Only two responses possible: the result of the operation requested or error and the reason for it



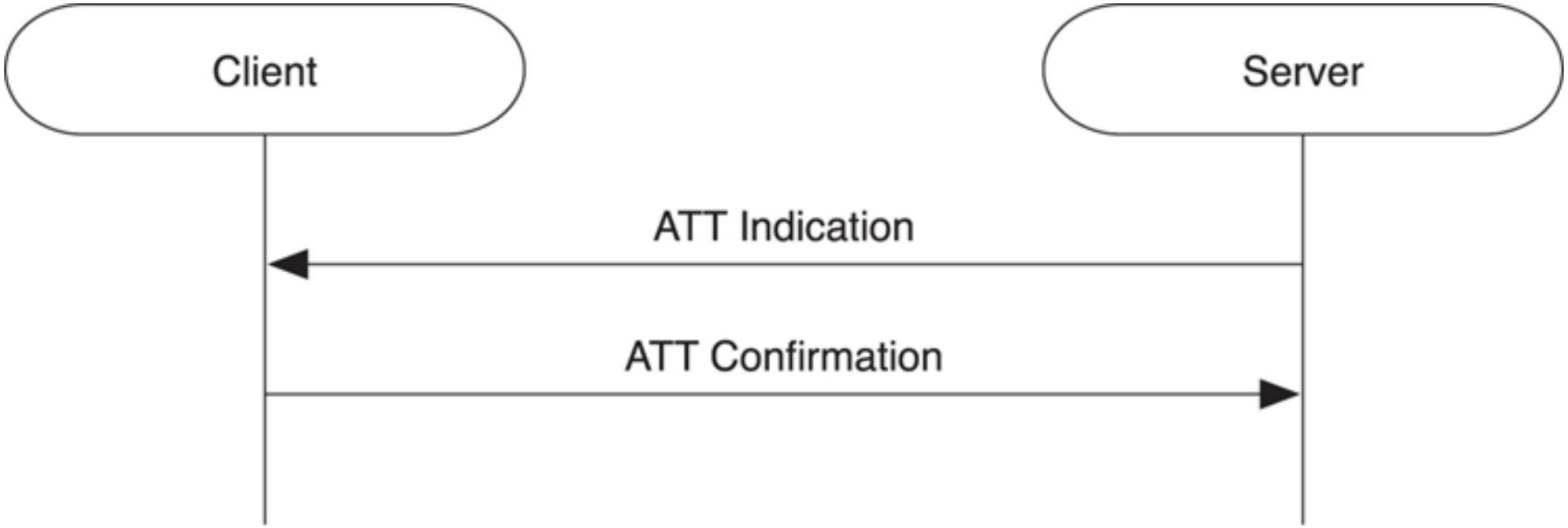
ATT: Command

- Requests to perform some operation/action on the server



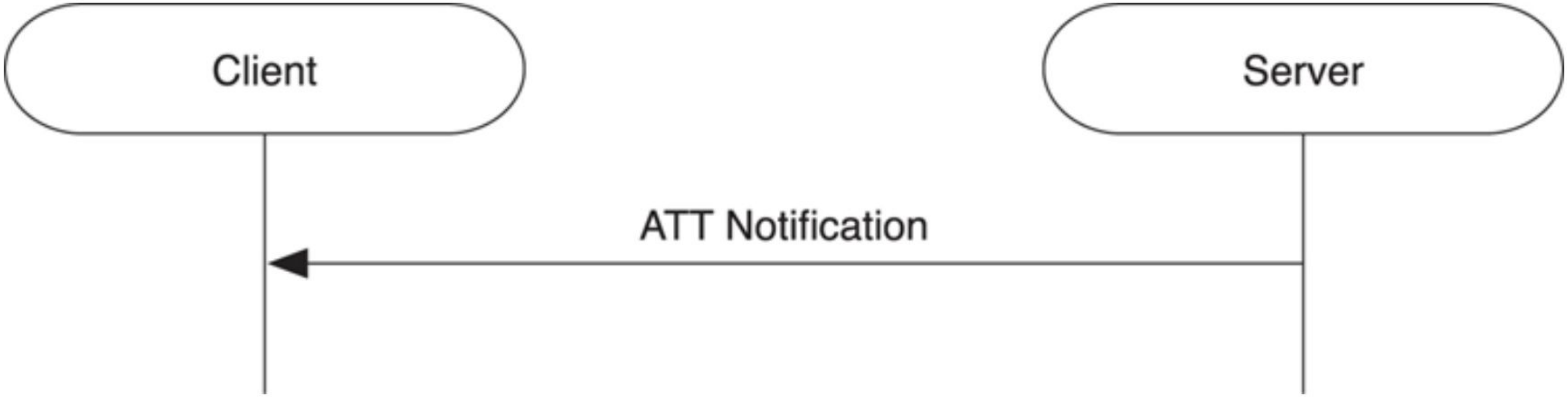
ATT: Indication

- The server indicates to the client that an attribute has changed its value, without an explicit request from the client
  - Indications must be confirmed by the client



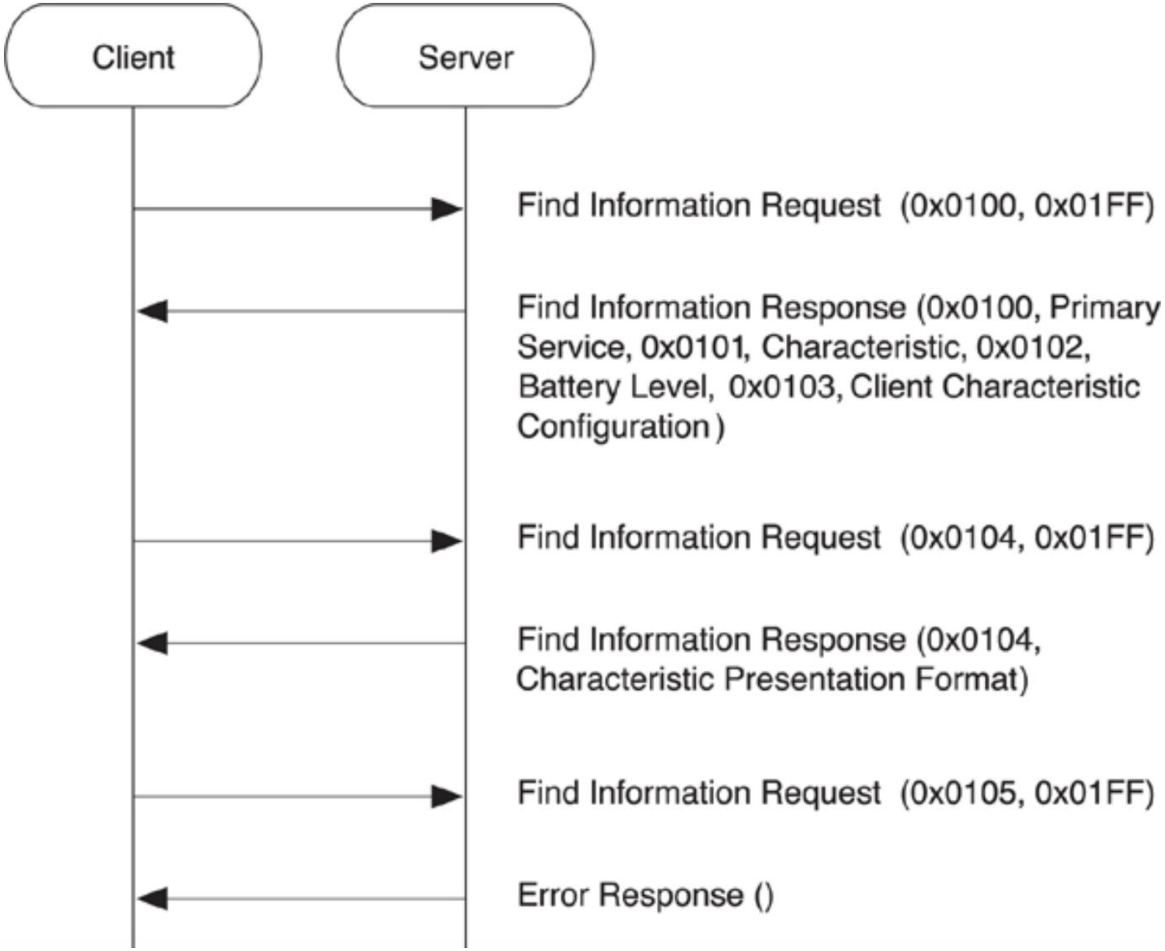
ATT: Notification

- Similar to an *Indication* but the client does not send confirmations for the notifications



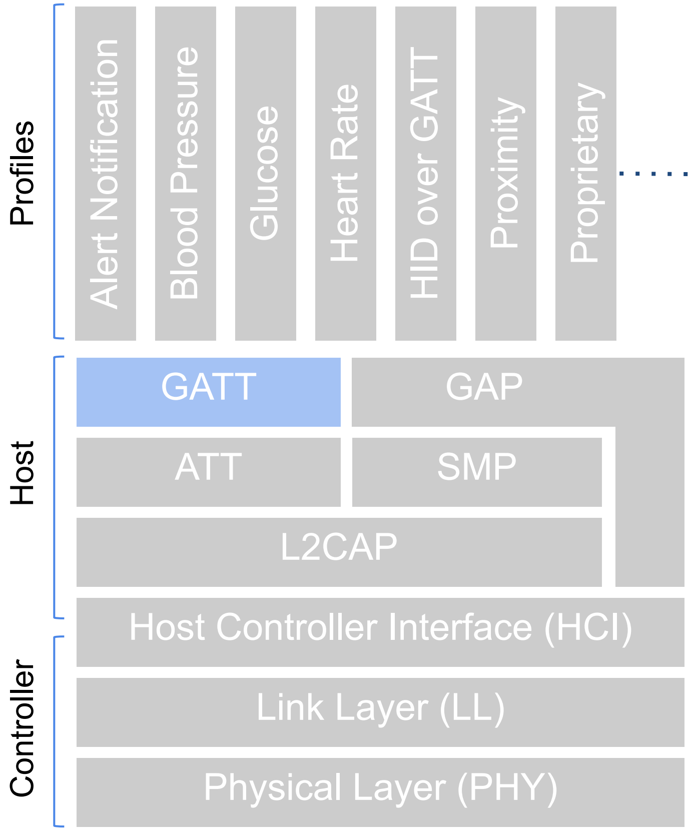
ATT: example

- Find Information Request
  - Indicates the range of *handles* for which the information is requested



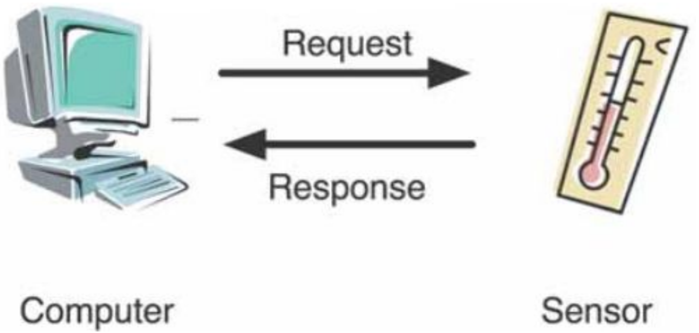
Client
Server
Find Information Request (0x0100, 0x01FF)
Find Information Response (0x0100, Primary Service, 0x0101, Characteristic, 0x0102, Battery Level, 0x0103, Client Characteristic Configuration)
Find Information Request (0x0104, 0x01FF)
Find Information Response (0x0104, Characteristic Presentation Format)
Find Information Request (0x0105, 0x01FF)
Error Response ()

Generic Attribute Profile (GATT)



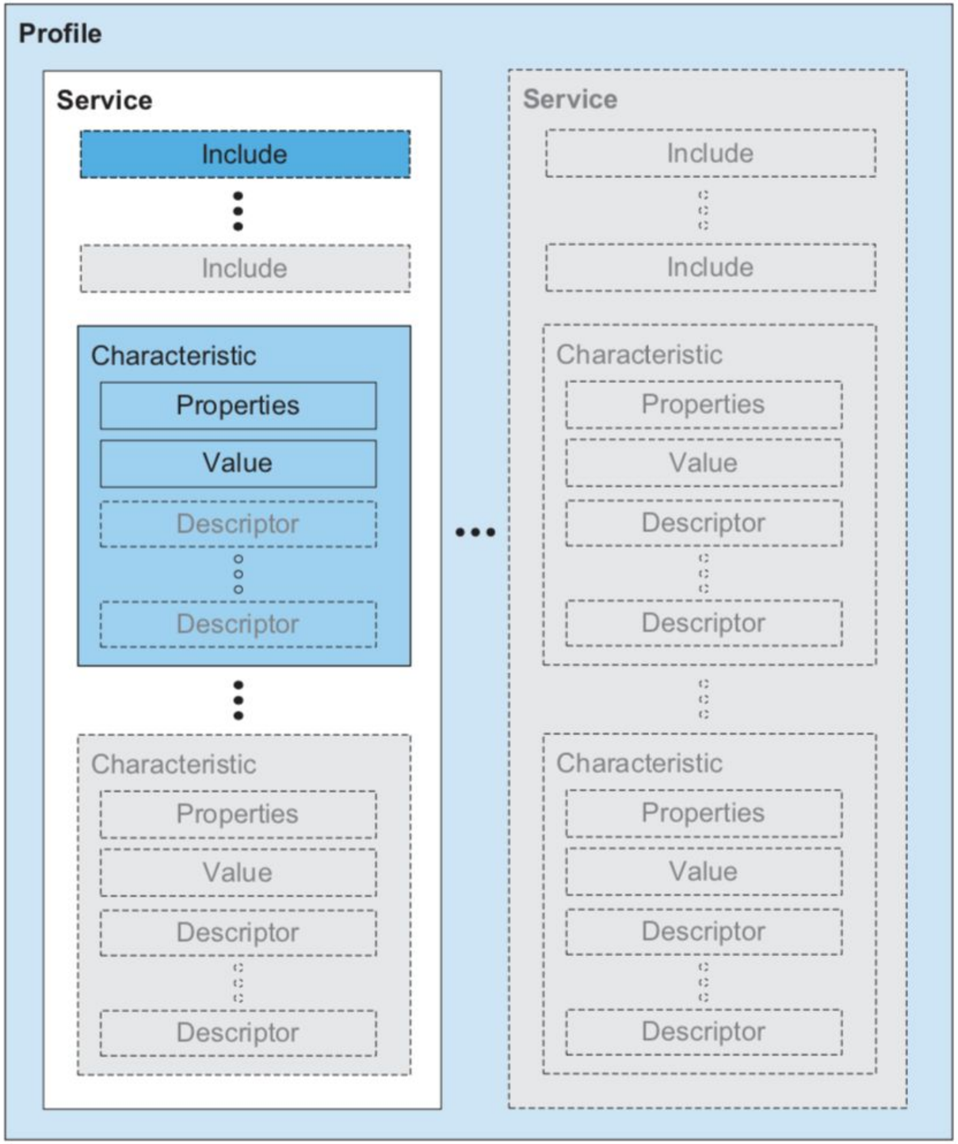
Generic Attribute Profile (GATT)

- GATT defines:
  - The way the services and information of a server has to be encoded as a hierarchy of Attributes
  - A simple API to manage the ATT protocol to work with these attributes
    - Specifies how to advertise, read, write and notify the attributes exposed by a server
- GATT Roles:
  - Client:
    - Sends commands and requests to the server
    - Receives responses, indications and notifications from the server
  - Server
    - Sends responses, indications and notifications to the client
    - Receives commands and requests from the client



GATT: attribute hierarchy

- **Profile**
  - contains one or more services
- **Service**
  - a set of characteristics
  - optionally include other services
- **Characteristic**
  - a value
  - a set of descriptors (additional information)

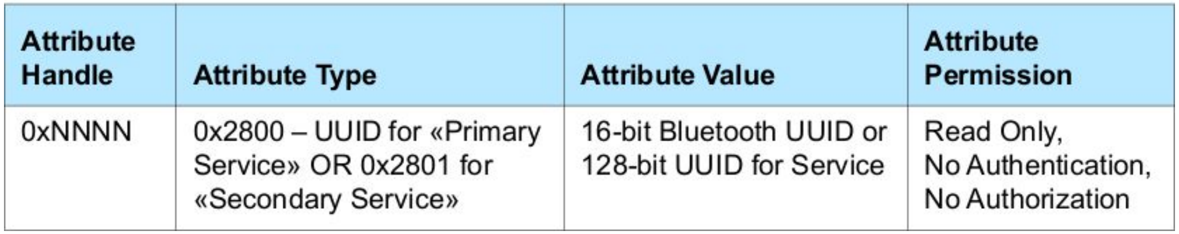


Service

- Is a collection of characteristics and a specified behaviour exposed through these characteristics
- The set of characteristics and the associated behaviour form the interface of the service
  - The behaviour may not be altered in future versions
- To *extend* a service, we can build another service that includes the former
  - It is a form of inheritance
  - This way the backward compatibility is assured
- Two types
  - Primary: service that is not included by other services, exposes the functionality of a device
    - A list of primary services can be requested through GATT and ATT
  - Secondary: referenced by other services

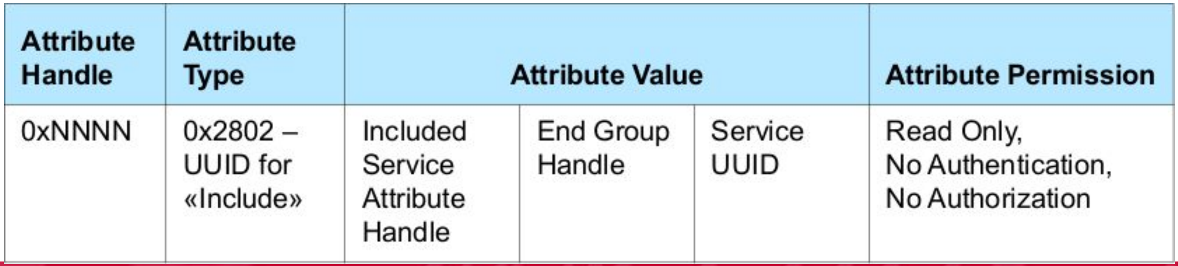
Define the service

- It is composed of:
  - a service declaration
  - optional inclusion definitions
  - optional characteristics
- They must appear in order in the ATT table
  - the definition attribute of a service ends the previous service
- The service definition attribute contains:
  - Type: UUID of primary or secondary service declaration
  - Value: UUID of the service



Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

- The service inclusion attribute contains:
  - Type: include UUID
  - Value: the *Handle* of the included service, End Group Handle and Service UUID

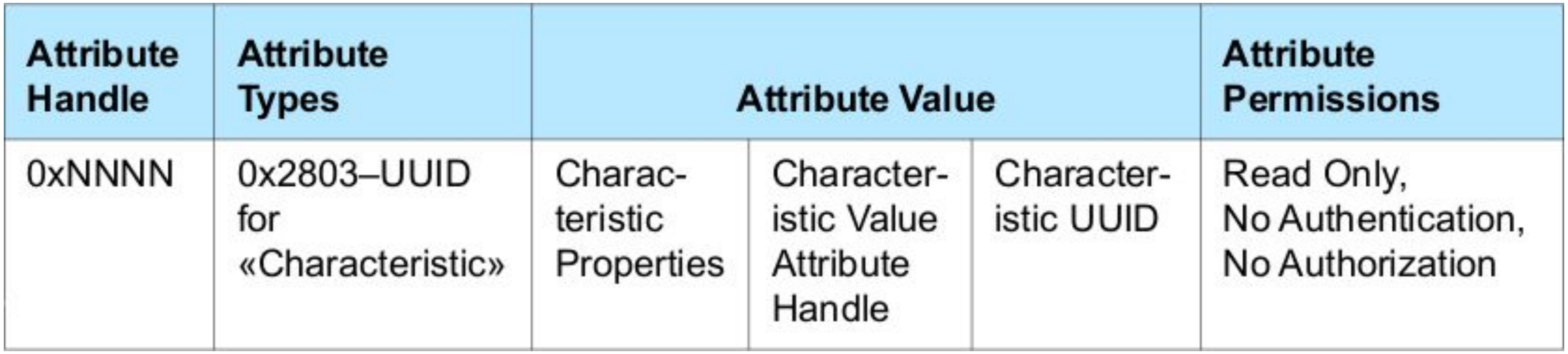


Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

Characteristics

- A characteristic represents a value:
  - E.g.: temperature, heart rate,…
- Composed of 3 basic elements:
  - Declaration attribute
    - starts the characteristics and contains its properties
  - Value attribute
    - contains the value of the characteristic
  - Descriptor attributes
    - Optional, contain additional information about the characteristic

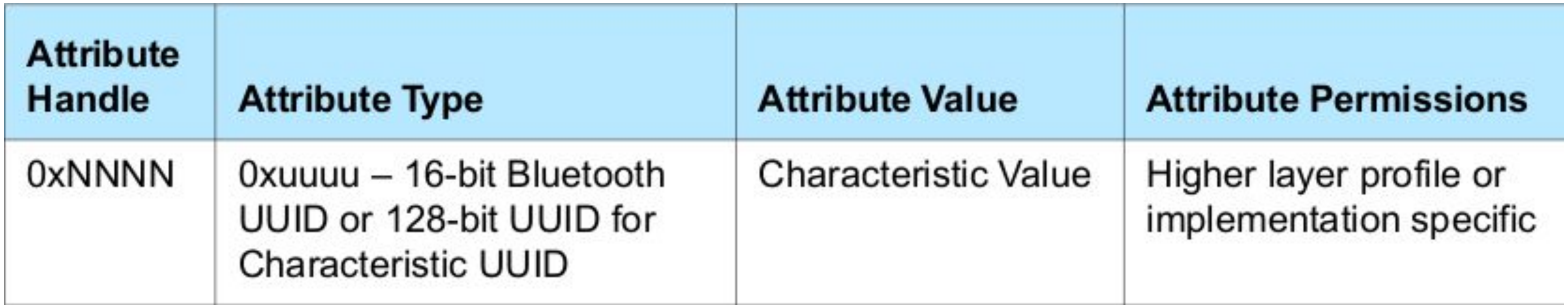
Characteristic Declaration Attribute



Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Charac-teristic Properties	Character-istic Value Attribute Handle	Character-istic UUID	Read Only, No Authentication, No Authorization

- An attribute with
  - Type: UUID for *Characteristic* (0x2803)
  - Value: composed of
    - Properties: if the value attribute can be read, written, notifications…
    - *handle* the handle of the characteristic value attribute
    - UUID for the type of value
      - It will be the same UUID used in the value attribute

Characteristic Value Attribute



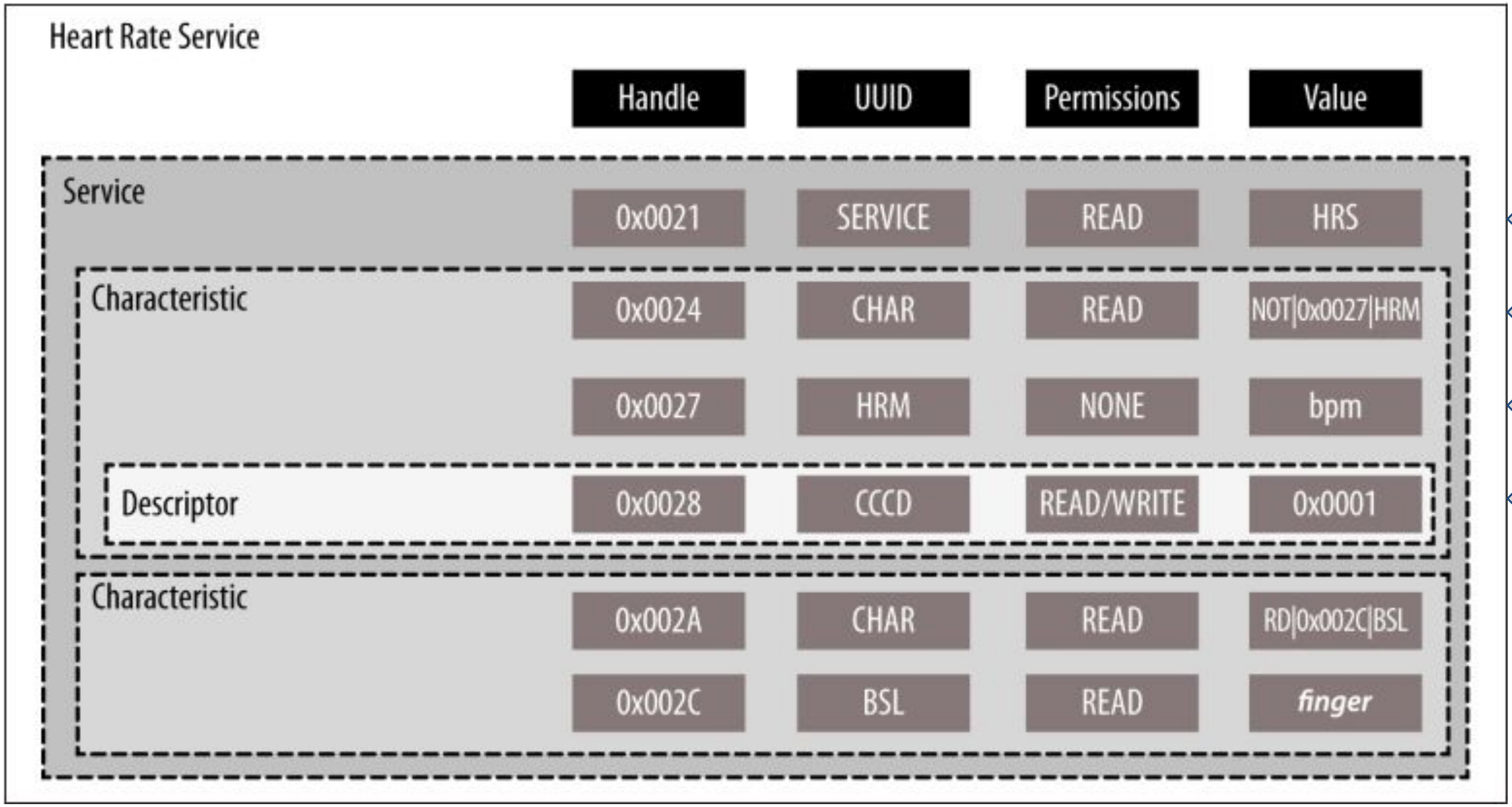
Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

- Contains:
  - Type: the UUID indicated in its declaration
  - Value: the characteristic value
- Must come just after the characteristic declaration
- The access permissions are exposed in the properties field of the characteristic declaration attribute, not in the attribute permissions field of this attribute

Characteristic Descriptor Attributes

- Provide additional information about the value
  - For instance, how to display it, where the sensor is located, …
- Some characteristics may require the presence of some descriptors (indicated in its properties)
  - Characteristic Extended Properties
  - Characteristic User Description
  - **Client Characteristic Configuration**
  - Server Characteristic Configuration
  - Characteristic Presentation Format
  - Characteristic Aggregation Format

An example: Heart Rate Service



Heart Rate Service

	Handle	UUID	Permissions	Value
Service	0x0021	SERVICE	READ	HRS
Characteristic	0x0024	CHAR	READ	NOT|0x0027|HRM
	0x0027	HRM	NONE	bpm
Descriptor	0x0028	CCCD	READ/WRITE	0x0001
Characteristic	0x002A	CHAR	READ	RD|0x002C|BSL
	0x002C	BSL	READ	*finger*

Service declaration

Characteristic declaration

Characteristic value

Characteristic descriptor

GATT: discovery procedures

- When a client connects to a server for the first time it does not know its structure
  - The handles may be cached for the conection
  - If the server has the *ServiceChanged* characteristic, the client must subscribe itself to notifications
  - If the devices are bonded, the cached information can be kept for future connections
- The client should first discover the primary services, and use their range of handles to discover:
  - *Secondary* services referenced by the primary service
  - The characteristics of the service and their descriptors
- Once the structure is known, the client can proceed to read/write the characteristic values

GATT: discovery procedures

- Procedures to discover services
  - Discover All Primary Services
    - *ReadByGroupTypeRequest* with *handles* 0x0001 to 0xFFFF and type *Primary Service*
  - Discover Primary Service By Service UUID
    - *FindByTypeValueRequest* with *handles 0x0001 to* 0xFFFF, type *Primary Service*, and the UUID of the service
  - Find Included Services
    - To get the secondary services included by other service
- Procedures to discover characteristics
  - Discover all characteristics of a service
    - ReadByTypeRequest(HandleRange, “Characteristic”)
  - Discover all characteristic descriptors
    - FindInformationRequest(HandleRange of declaration)

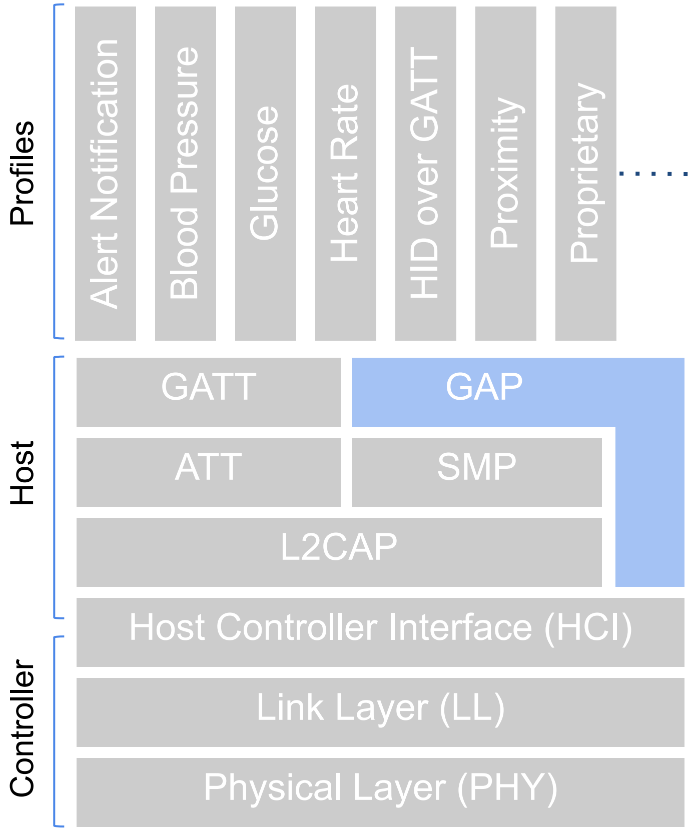
GATT: procedures on characteristics

- A cliente que initiate 2 procedures on characteristics and their descriptors (Vol3, partG, 4.2):
  - Read its value: *ReadRequest*
    - Params: characteristic *handle* and type (UUID)
    - *ReadBlobRequest* for attributes larger than 22 bytes (or variable size)
  - Write its value: *WriteRequest*
    - Params: characteristic *handle* and type (UUID)
    - Sequence of *PrepareWriteRequest* followed by one *ExecuteWriteRequest* for large values

GATT: server characteristic procedures

- Two procedures can be initiated by the GATT server:
  - *Notifications*
    - No flow control (no confirmation)
    - The client may miss the notification
    - Not reliable
    - ATT message: *HandleValueNotification*
  - *Indications*
    - Have control flow: the server cannot send a new indication until the previous one has been confirmed by the client
    - ATT message: *HandleValueIndication*
      - Confirmed by the client with a *HandleValueConfirmation* ATT message
- Enabled by each characteristic on its *Client Characteristic Configuration Descriptor (CCCD)*
  - The CCCD value is preserved between connections for bonded devices
  - Each client obtains its own CCCD instance

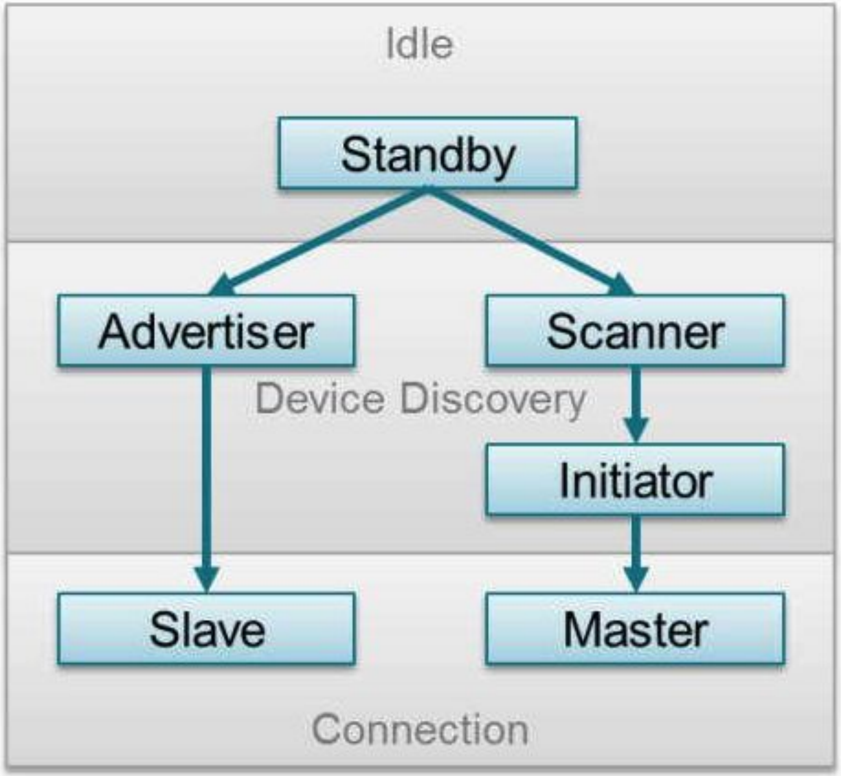
Generic Access Profile (GAP)



GAP: Generic Access Profile

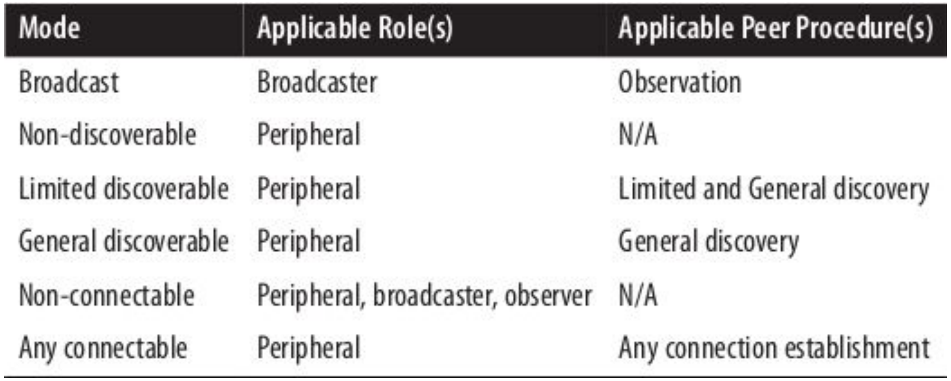
- Defines how a device can discover other devices and connect to them
- Defines the roles for the devices
  - Broadcaster
    - A device that sends *Advertising Packets*
    - Can *broadcast* data
    - Only needs an emitter
  - Observer
    - Listens to *broadcasters* and transmits the information to the application
  - Peripheral
    - A device that announces it self with *Connectable Advertising Packets*
    - Becomes the slave once connected
    - Needs emitter and receiver
  - Central
    - Device that initiates connections with a *Peripheral*
    - Becomes the master once connected
- A device can manage several GAP profiles simultaneously
  - Can be a *broadcaster* and *peripheral* at the same time

GAP: states

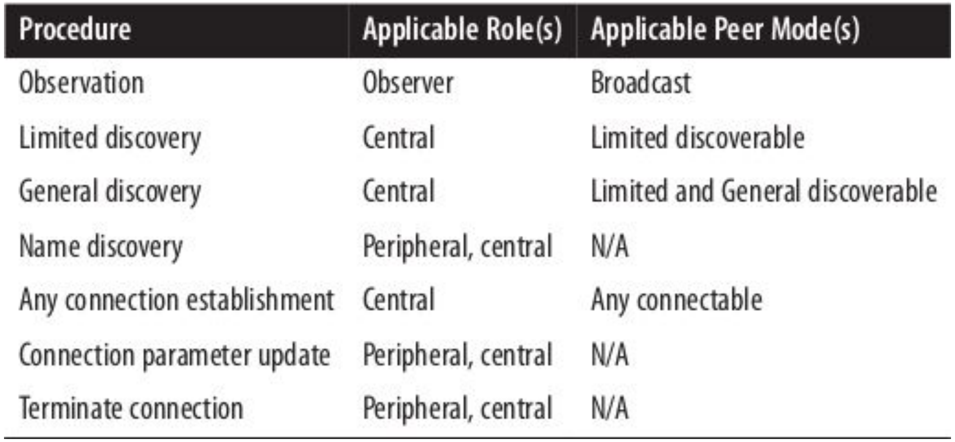

Idle
Standby
Advertiser
Scanner
Device Discovery
Initiator
Slave
Master
Connection

GAP: modes and procedures

- Mode: a state on which the device can be temporally
  - Achieve a certain goal, or
  - Allow another device to complete a procedure
- Procedure: sequence of actions to achieve a goal
  - Send a sequence of packets from the LL
  - The other device must be in a certain state



Mode	Applicable Role(s)	Applicable Peer Procedure(s)
Broadcast	Broadcaster	Observation
Non-discoverable	Peripheral	N/A
Limited discoverable	Peripheral	Limited and General discovery
General discoverable	Peripheral	General discovery
Non-connectable	Peripheral, broadcaster, observer	N/A
Any connectable	Peripheral	Any connection establishment



Procedure	Applicable Role(s)	Applicable Peer Mode(s)
Observation	Observer	Broadcast
Limited discovery	Central	Limited discoverable
General discovery	Central	Limited and General discoverable
Name discovery	Peripheral, central	N/A
Any connection establishment	Central	Any connectable
Connection parameter update	Peripheral, central	N/A
Terminate connection	Peripheral, central	N/A

Broadcast and Observation

A device can send information (broadcaster) to one or more listening devices (observers)

- The observers do not acknowledge the packets
- The observer listens without knowing if he is going to receive something

Each advertising packet contains as much as 31 bytes

- It can indicate if Scan Requests are allowed
- Can indicate that it is an extended advertisement (BT 5.0)

Discoverability modes

- Used in the *Peripheral* role
- Three modes
  - ***Non-discoverable***: devices that do not require its presence to be perceived and do not want to receive requests
    - Only send ADV_NONCONN_IND o ADV_SCAN_IND
  - ***Limited discoverable***: devices that want to be discovered in a limited time period
    - Must activate the *Limited Discoverable* flag in the adverisement
    - The centra has to complete the *Limited Discovery o General Dicsovery* procedure
  - ***General discoverable***: devices that want to be discovered at any moment, to establish a connection with a central device.
    - They have to activate the General Discoverable flag in the advertisements
    - The central has to complete the *General Discovery* procedure

Discoverability procedures

- Limited Discovery
  - The central initiates this procedure without a *white list*
  - GAP analyses all the received advertisements and only communicates to the application those that have the *Limited Discovery Flag* active
- General Discovery
  - The central initiates this procedure without a *white list*
  - GAP analyses all the received advertisements and only communicates to the application those that have the *Limited Discovery* or *General Discovery Flags* activated

Connection establishment

- The peripheral can be in one of the following modes:
  - ***Non-connectable***: a peripheral in this mode can only send ADV_NONCONN_IND or ADV_SCAN_IND advertisements
    - No central can connect to these peripherals
  - ***Directed connectable***: a device in this mode sends ADV_DIRECT_IND with high frequency, indicating the central they want to connect to.
    - Fast connection mode
  - ***Undirected connectable***: standard mode for a peripheral that wants to connect to a central, sends ADV_IND advertisements

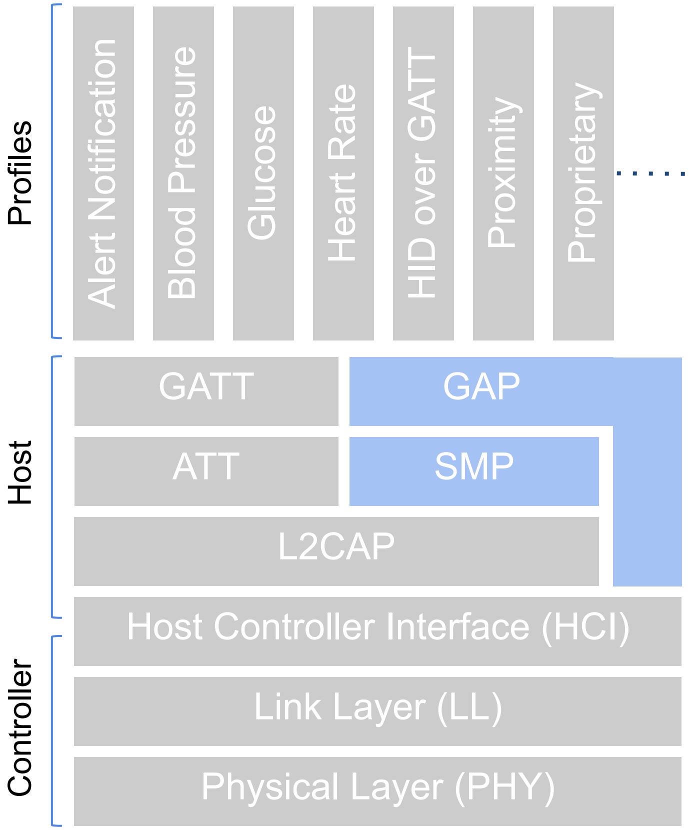
Connection establishment

- The central can initiate the following procedures:
  - ***Auto-connection establishment***: to connect to an unknown device, with no priorities.
    - The application provides a white list of known devices to which it wants to connect.
    - GAP initiates the connection with the first device from the list discovered.
  - ***General connection establishment***: used to connect to an unknown device
    - The centra first scans to detect devices sending advertisements
    - The application must then select the device for the connection
  - ***Selective connection establishment***: used to connect to a known device
    - Like the general, but using a white list to filter out devices
  - ***Direct connection establishment***: initiates the connection procedure with a known device
    - Can fail if the device is not present or is not in a connectable mode

Other GAP procedures

- Name discovery:
  - Allows the central to obtain the name of a connected device (string of utf-8 characters)
- Connection parameter update:
  - Changes the parameters of the connection (connection interval, slave latency, etc.)
  - The peripheral can only request it, the central is the one that has to make the changes
- Terminate connection

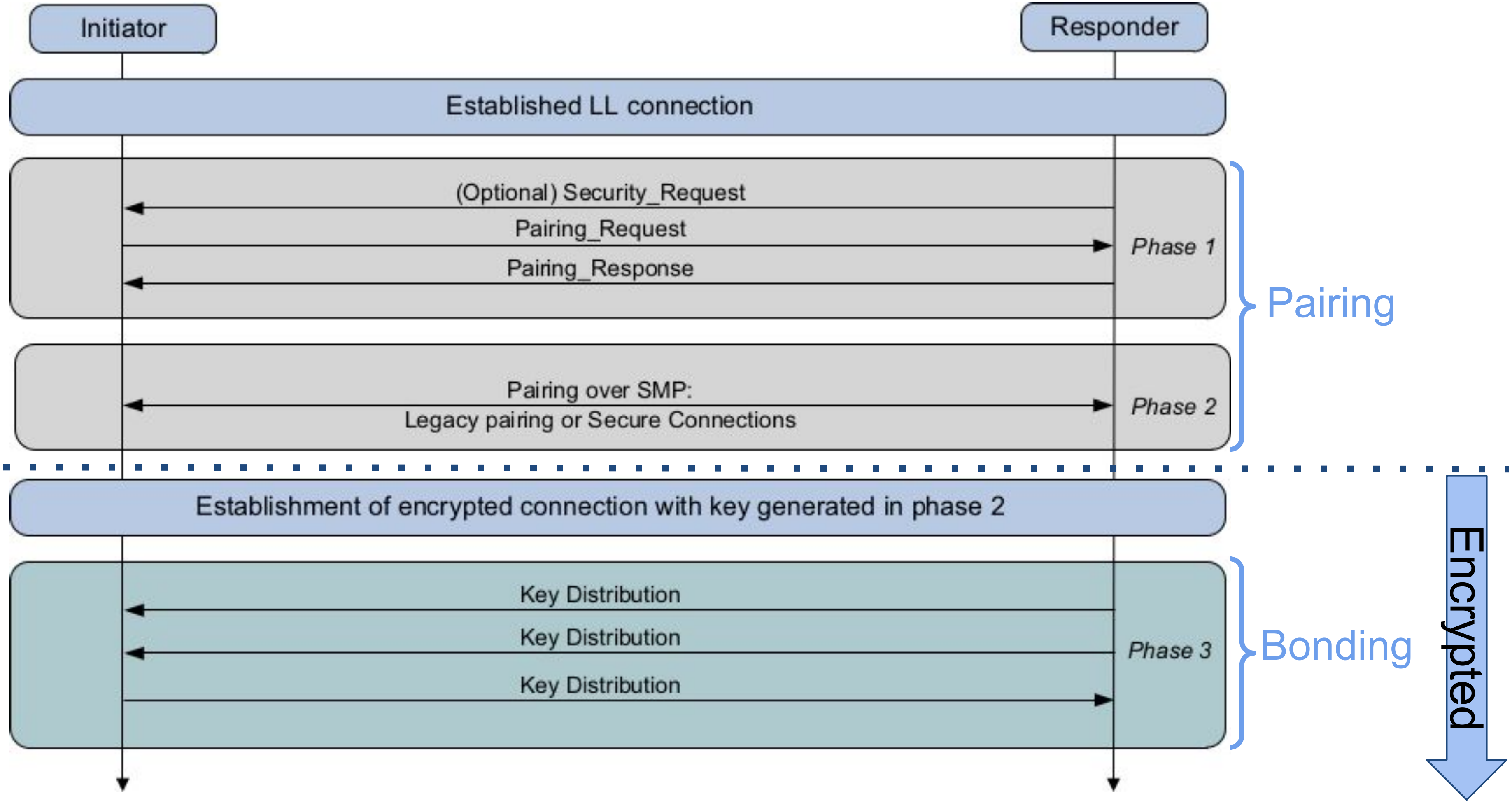
GAP + Security Manager (SMP)



Security aspects in BLE

- Authentication
  - That the other is who it says
  - BLE: PassKey, OOB or numeric
- Integrity
  - Others cannot modify the message
  - BLE: Message Authentication Code (MAC), 64 bits with AES-128
- Confidentiality
  - Others cannot read the transferred messages
  - BLE: AES-CCM encryption
- Privacy
  - Others cannot identify the node
  - BLE: random addresses

BLE: phases, pairing and bonding



Pairing

Phase 1:

- Interchange of I/O capacities and security requirements

Phase 2:

- Legacy connections (BT 4.0-4.1): weaker
  - Short Term Key (STK) generated from a shared Temporal Key (TK) and other parameters (device addresses, type of devices, …)
  - The STK is used to encrypt the communication afterwards
- Secure connections (from BT 4.2): much more robust
  - A Long Term Key is generated with an Elliptic-curve Diffie–Hellman (ECDH) procedure (uses public-private key pairs)
  - The LTK is used to encrypt the communication afterwards

Pairing methods

Legacy connections (BT 4.0-4.1):

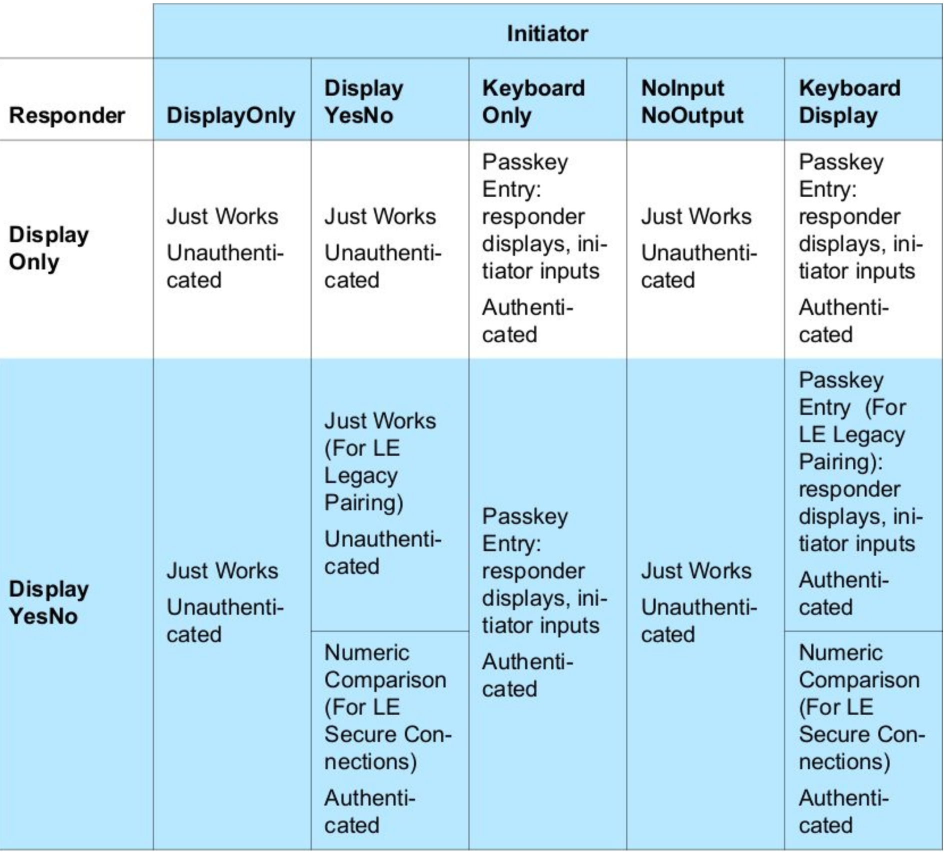
- Just Works
- PassKey -- Authenticated
- Out of Band (OOB) -- Authenticated

Secure connections (from BT 4.2): much more robust

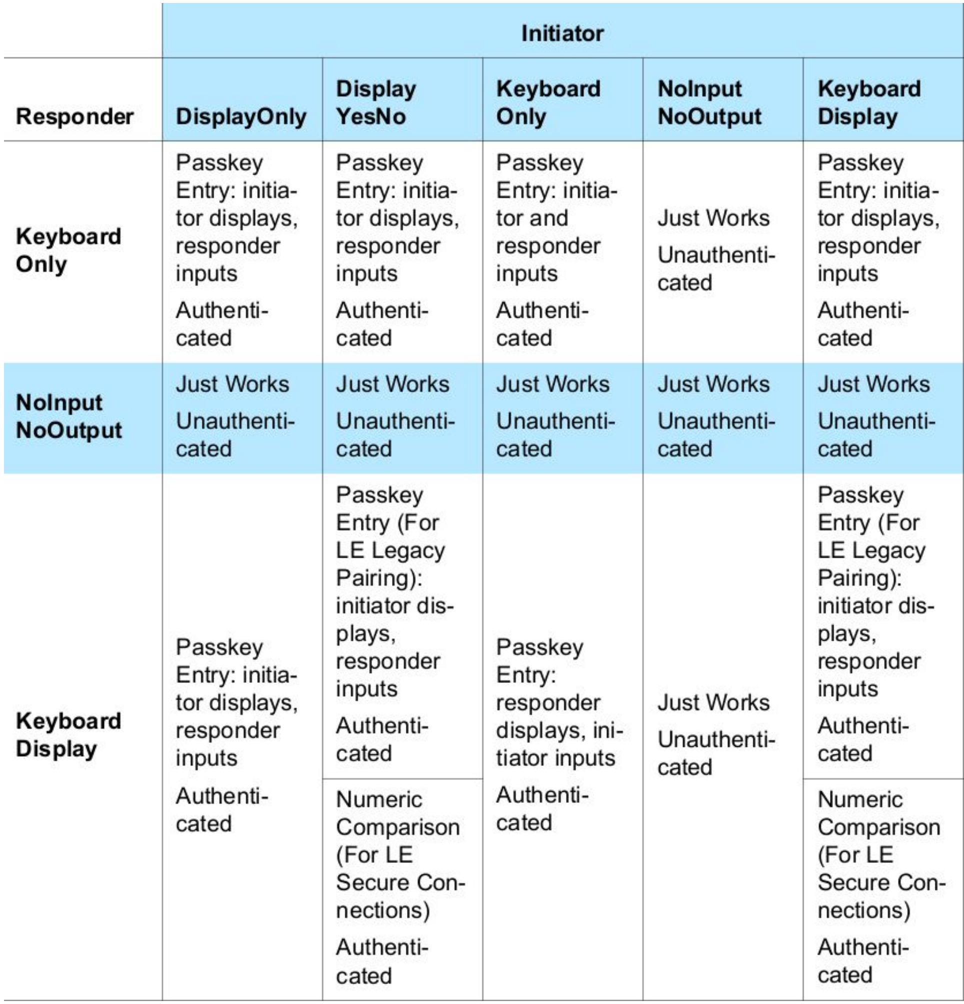
- Just Works
- PassKey -- Authenticated
- Out of Band (OOB) -- Authenticated
- Numeric -- Authenticated

The methods for legacy and secure connections are different

IO capabilities -> authentication method



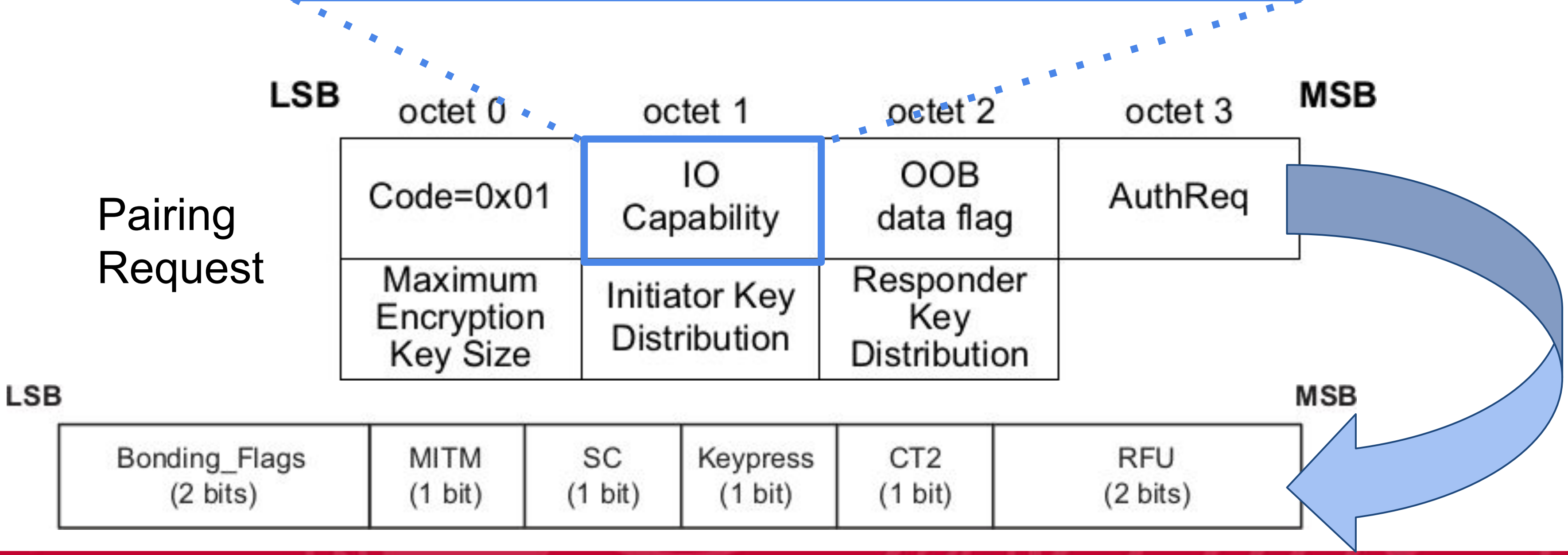
	Initiator				
Responder	**DisplayOnly**	**Display YesNo**	**Keyboard Only**	**NoInput NoOutput**	**Keyboard Display**
Display Only	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated
Display YesNo	Just Works Unauthenticated	Just Works (For LE Legacy Pairing) Unauthenticated Numeric Comparison (For LE Secure Connections) Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): responder displays, initiator inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated



	Initiator				
Responder	**DisplayOnly**	**Display YesNo**	**Keyboard Only**	**NoInput NoOutput**	**Keyboard Display**
Keyboard Only	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator and responder inputs Authenticated	Just Works Unauthenticated	Passkey Entry: initiator displays, responder inputs Authenticated
NoInput NoOutput	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated
Keyboard Display	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated Numeric Comparison (For LE Secure Connections) Authenticated

Pairing Request and Pairing response

Value	Description
0x00	DisplayOnly
0x01	DisplayYesNo
0x02	KeyboardOnly
0x03	NoInputNoOutput
0x04	KeyboardDisplay
0x05-0xFF	Reserved for future use



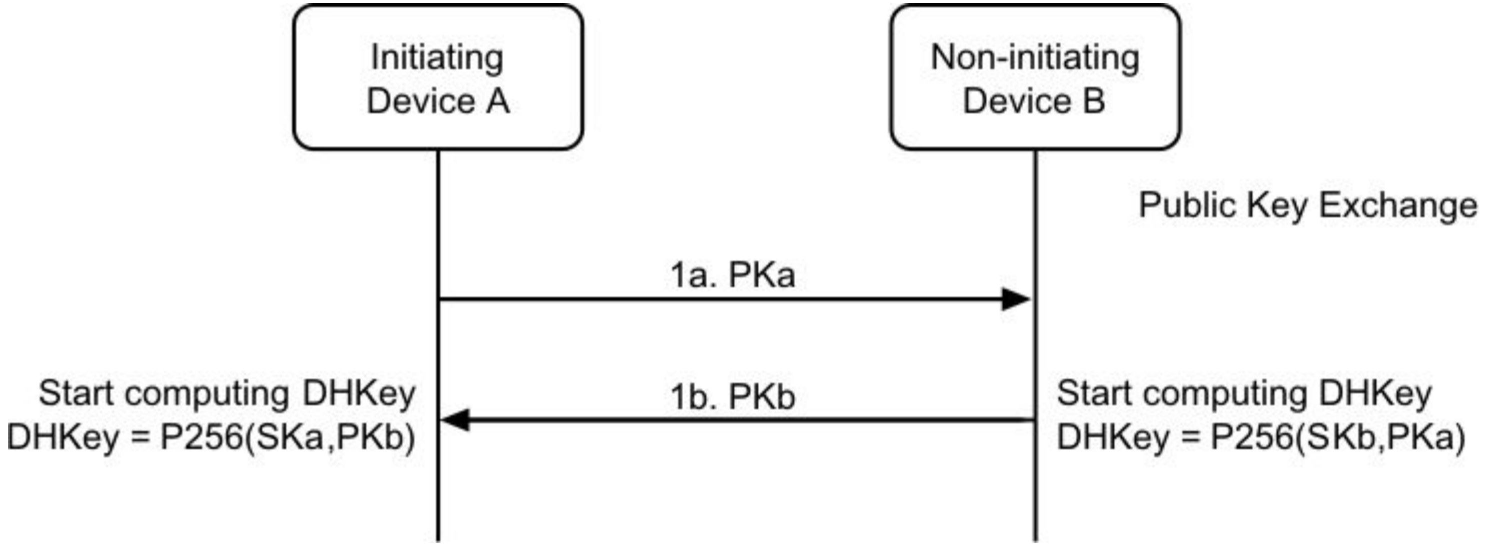
Pairing Request

LSB octet 0	octet 1	octet 2	octet 3 MSB
Code=0x01	IO Capability	OOB data flag	AuthReq
Maximum Encryption Key Size	Initiator Key Distribution	Responder Key Distribution	

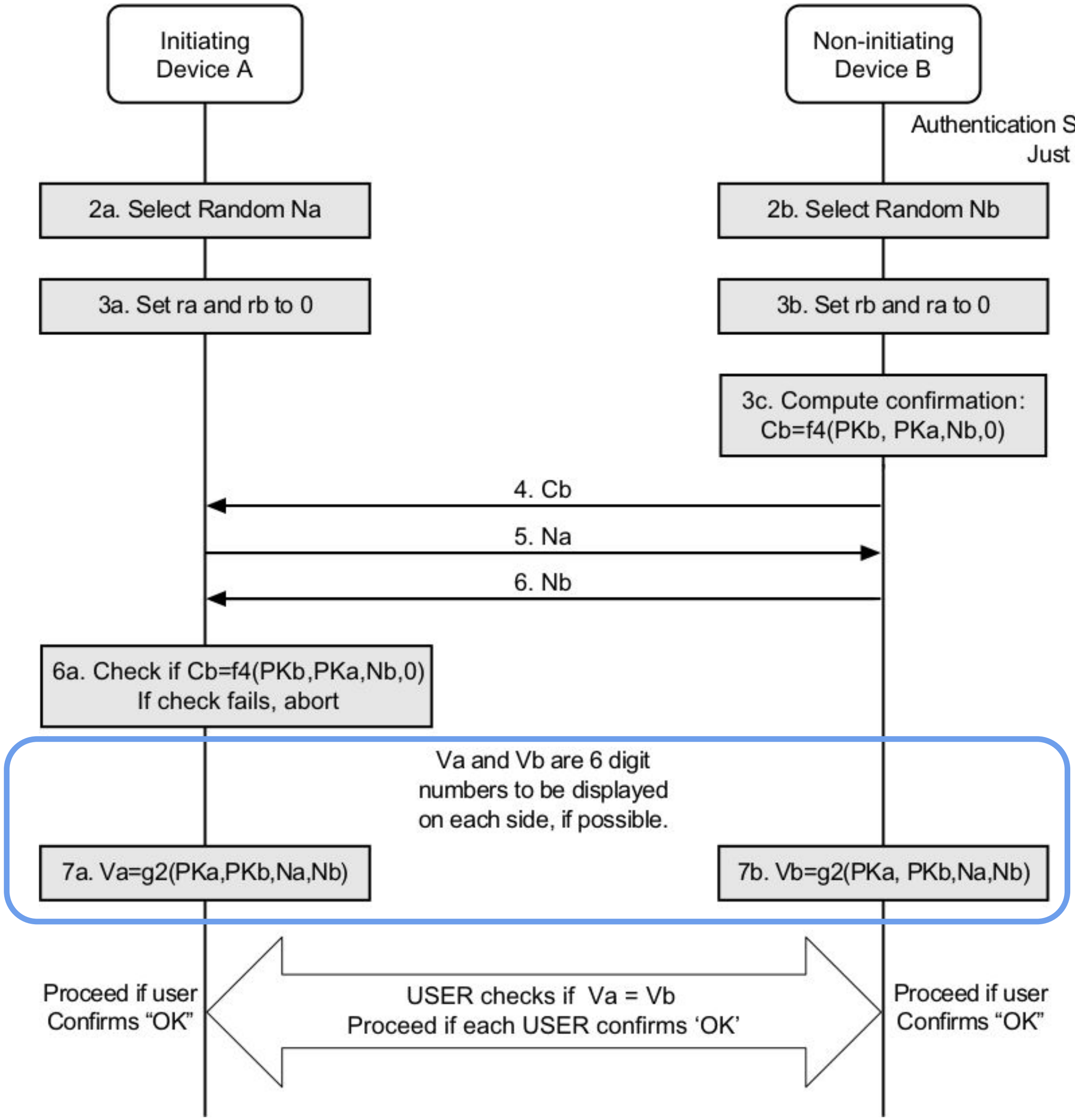
LSB Bonding_Flags (2 bits)	MITM (1 bit)	SC (1 bit)	Keypress (1 bit)	CT2 (1 bit)	RFU (2 bits) MSB

E.g.: pairing in LE secure connections

Initial public key interchange and generation of the DHKey shared secret

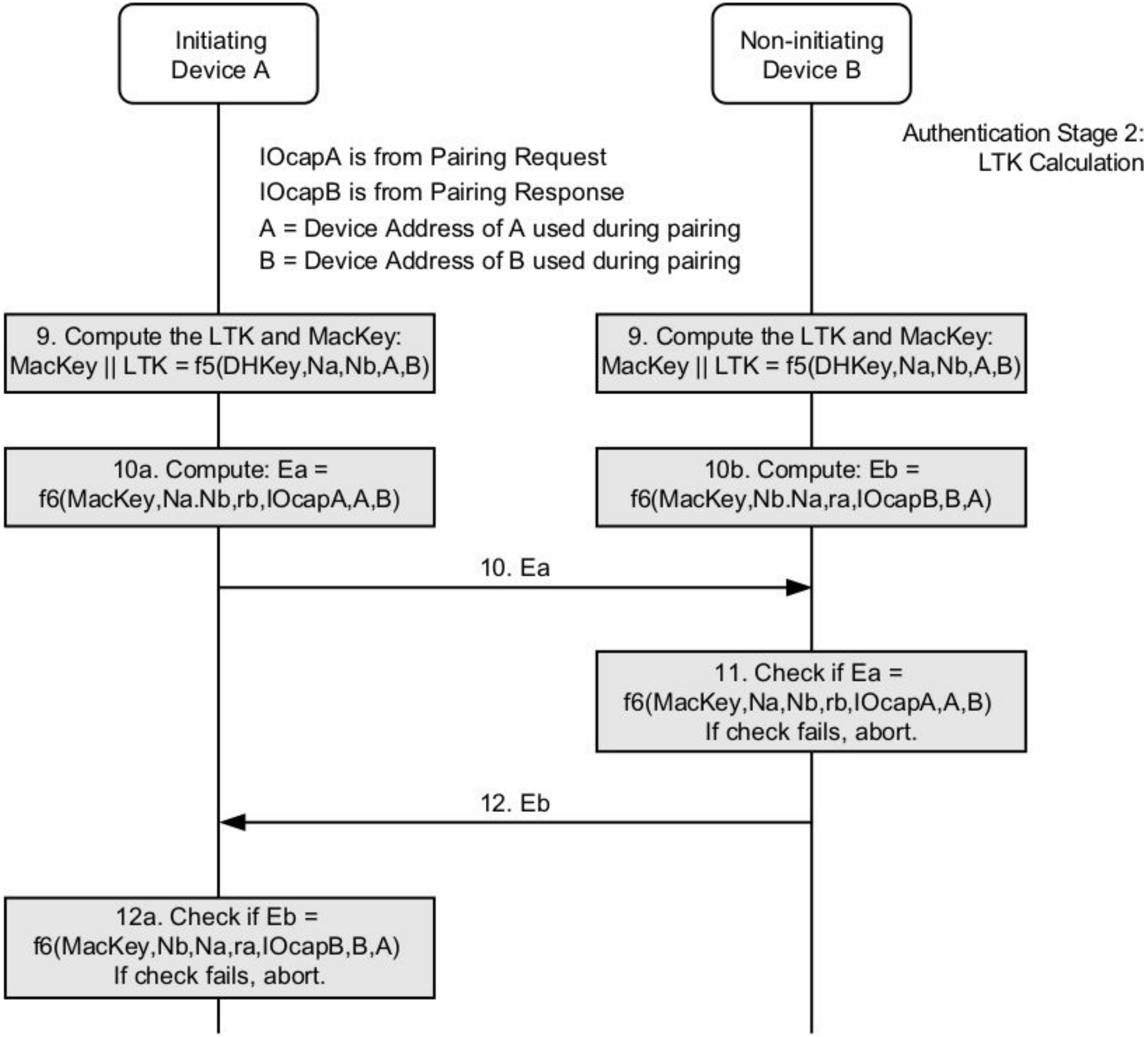


E.g.: LE secure pairing j. works & numeric


Initiating Device A
Non-initiating Device B
Authentication Stage1: Just Works
2a. Select Random Na
2b. Select Random Nb
3a. Set ra and rb to 0
3b. Set rb and ra to 0
3c. Compute confirmation: Cb=f4(PKb, PKa,Nb,0)
4. Cb
5. Na
6. Nb
6a. Check if Cb=f4(PKb,PKa,Nb,0) If check fails, abort
Va and Vb are 6 digit numbers to be displayed on each side, if possible.
7a. Va=g2(PKa,PKb,Na,Nb)
7b. Vb=g2(PKa, PKb,Na,Nb)
Proceed if user Confirms "OK"
USER checks if Va = Vb Proceed if each USER confirms 'OK'
Proceed if user Confirms "OK"

Numeric: Va & Vb are shown to the user that can confirm them

E.g.: LE secure LTK generation



Initiating Device A

Non-initiating Device B

Authentication Stage 2:
LTK Calculation

IOcapA is from Pairing Request
IOcapB is from Pairing Response
A = Device Address of A used during pairing
B = Device Address of B used during pairing

9. Compute the LTK and MacKey:
MacKey || LTK = f5(DHKey,Na,Nb,A,B)

9. Compute the LTK and MacKey:
MacKey || LTK = f5(DHKey,Na,Nb,A,B)

10a. Compute: Ea =
f6(MacKey,Na.Nb,rb,IOcapA,A,B)

10b. Compute: Eb =
f6(MacKey,Nb.Na,ra,IOcapB,B,A)

10. Ea

11. Check if Ea =
f6(MacKey,Na,Nb,rb,IOcapA,A,B)
If check fails, abort.

12. Eb

12a. Check if Eb =
f6(MacKey,Nb,Na,ra,IOcapB,B,A)
If check fails, abort.

Bonding

In phase 1 the bonding flag is checked

- Extra keys are generated for future connections
  - Identity Resolving Key (IRK): 128 bits for random address generation and resolution
  - Connection Signature Resolving Key (CSRK): 128 bits for signature verification
  - Long Term Key (LTK): 128 bits to encrypt communications (legacy)
  - Encrypted Diversifier (EDIV): 16 bits used to identify the LTK (legacy)
  - Random Number (Rand): 64 bits used to identify the LTK (legacy).
- If bonded, the devices can omit the pairing and use the stored keys

Security Modes

- Required by devices and/or services
- Security Mode 1:
  - Level 1 -> security is not required
  - Level 2 -> no authentication required, but we want confidentiality
  - Level 3 -> authentication and confidentiality are required
  - Level 4 -> LE secure connection required, with autenticación and confidentiality
- Security Mode 2:
  - Level 1 -> unauthenticated with signatures
  - Level 2 -> authenticated with signatures

References

- Bluetooth core specification
  - https://www.bluetooth.com/specifications/bluetooth-core-specification/
- Kevin Townsed, Carles Cufí, Akiba & Robert Davidson, “Getting Started with Bluetooth Low Energy”, 2014, O’Reilly.
- Robin Heydon, “Bluetooth Low Energy: The Developer's Handbook”, 2013, Prentice Hall
- SDK Texas instruments
  - http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/index.html