UNIVERSITÀ DEGLI STUDI DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale
in Communication Technologies and Multimedia

Bluetooth Security Evolution

Supervisor: Prof. Francesco Gringoli
Co-rapporteur: Dott. Marco Cominelli

Author:
Mohamed Yousif Elamin Abdelrahman
Matricola n. 730194

Anno Accademico 2021/2022

Abstract

Bluetooth is a popular wireless technology that comprises two different standards, Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and Bluetooth Low Energy (BLE). Bluetooth BR/EDR remains the effective connectivity technology for applications requiring high data rates, such as audio voice calling and streaming. On the other hand, BLE had remarkable success thanks to its low power requirements and its ubiquitous availability. BLE also enabled many new product types. Devices such as smartwatches and healthcare appliances would not be possible without such technology. Bluetooth 5.3 is the most recent version of the specification, a work resulting from various revisions, each one improving the available features and adding new characteristics and abilities, including more secure and robust pairing methods, like Secure Connection Pairing. Security was considered from the birth of Bluetooth and is being improved with each specification version.

In this thesis, we aim to extend the current body of knowledge regarding Bluetooth security. We first present a quick overview of both BR and BLE. Then, we present a more comprehensive description of BR and BLE security and privacy properties. We discuss several known vulnerabilities and attacks to both Bluetooth technologies. Finally, we validate a Man-in-the-Middle attack by experimenting with a *Micro:bit*-based sniffer that can monitor real-time BLE traffic. We show that, despite the limited funcionality, even a low-cost sniffer can easily detect active BLE sessions and infer their properties, including Access Addresses, CRC values, and hopping sequences.

Acknowledgements

Prophet Muhammad, peace and blessings be upon him, said, "He who does not thank people, does not thank Allah."
First and foremost, "Alhamdulillah." All praise is for Allah—Lord of all worlds, the Most Compassionate, and Most Merciful. For this blessing and all blessings.

My sincerest thanks to my supervisor Francesco Gringoli, and my co-rapporteur Marco Cominelli, without whom this work would not have been possible. Thank you for your supervision, guidance, and patience.
My big thanks to the University of **BRESCIA** for honoring me with this opportunity and opening my eyes to broader horizons.

Furthermore, I am so grateful to all my dear **FRIENDS**. You guys have made this journey smooth and remarkable. I am so lucky to have you. "Love you all <3."

Finally and most importantly, I thank my **FAMILY** for their love, patience, and support through this long and arduous journey. I can never pay back.

To be continued...

Table of Contents

List of Figures

List of Tables

List of Abbreviations

8DPSK	8-Phase Differential Phase-Shift Keying
ACL	Asynchronous Connectionless
ACO	Authenticated Ciphering OffseT
AES	Advanced Encryption Standard
AFH	Adaptive Frequency Hopping
ARQN	Automatic Repeat Request Number
BLE	Bluetooth Low Energy
BR	Basic Rate
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC mode
CE	Common Era
CLK	Clock
COF	Ciphering Offset Number
CRC	Cyclic Redundancy Check
CSRK	Connection Signature Resolving Key
CTKD	Cross-Transport Key Derivation
DHkey	Diffie-Hellman Key
DoS	Denial of Service
DQPSK	Differential Quaternary Phase-Shift Keying
DTM	Direct Test Mode
ECDH	Elliptic-Curve Diffie-Hellman
EDIV	Encrypted Diversifier
EDR	Enhanced Data Rate

FEC	Forward Error Correction
FHHS	Frequency Hopping Spread Spectrum
FIPS	Federal Information Processing Standards
FM	Frequency Modulation
GAP	Generic Access Profile
GATT	Generic Attribute Profil
GFSK	Gaussian Frequency-Shift Keying
GHz	Giga Hertz
HCI	Host Controller Interface
HEC	Header Error Check
HMAC	Hash-based Message Authentication Code
I/O	Input-Output
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers
ILK	Intermediate Link Key
ILTK	Intermediate Long Term Key
IoT	Internet of Things
IP	Internet Protocol
IRK	Identity Resolving Key
ISM	Industrial Scientific Medical
IV	Initialization Vector
JW	Just Works
L2CAP	Logical Link Control and Adaptation Protocol
LC	Link Control
LE	Low Energy
LL	Link Layer
LMP	(Link Manager Protocol
LSB	Least Significant Bit
LT_ADDR	Logical Transport Address

LTK	Long Term Key
MAC	Message Authentication Code
MAC [address]	Media Access Control [address]
MHz	Mega Hertz
MIC	Message Integrity Check
MITM	Man-in-the-Middle
MSB	Most Significant Bit
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
OOB	Out of Band
PC	Personal Computer
PDU	Protocol Data Unit
PHY	Physical Layer
PIN	Personal Identification Number
PK	Public Key
PKE	Passkey Entry
PSK	Phase-Shift Keying
Rand	Random Number
RF	Radio Frequency
RFC	Request for Comment
RFU	Reserved For Future
SC	Secure Connection
SCO	Synchronous Connection Orientated
SDP	Service Discovery Protocol
SDR	Software Defined Radio
SEQN	Sequence Number
SHA	Secure Hash Algorithm
SIG	Special Interest Group
SK	Secure Key

SKD	Security Key Diversifier
SM	Security Manager
SMP	Security Manager Protocol
SRES	Signed Response
SSP	Secure Simple Pairing
STK	Short Term Key
TDD	ime Division Duplex
TK	Temporary Key
UUID	Universally Unique Identifier
WPAN	wireless personal area networks

Chapter 1

Introduction

1.1 Thesis Objectives

This thesis aims at reviewing Bluetooth's security features, thus making the learning curve less steep for people who need to use such features and understand how they work. Moreover, in this thesis we want to test with a simple experiment the feasibility of breaking those security features using a low-cost Bluetooth sniffer.

1.2 Thesis Structure

The document is subdivided into seven chapters. The first chapter is this introduction. Chapter 2 delivers a quick overview of Bluetooth Classic (Basic Rate and Extended Data Rate, BR/EDR). Chapter 3 introduces Bluetooth Low Energy (BLE), its architecture, characteristics, and operation. Chapters 4 and 5 detail security measures and procedures (pairing, authentication, and encryption) for Bluetooth BR/EDR and BLE, respectively. Chapter 6 shows the possible attacks and vulnerabilities of Bluetooth BR/EDR and Bluetooth LE. Chapter 7 provides a detailed structure of the Bluetooth LE advertising packet and the feasibility of breaking the BLE security using BtleJack. In Chapter 8, we conclude our thesis and provide some recommendations.

1.3 Bluetooth Technology Overview

Bluetooth is a short-range radio frequency (RF) communication standard. It is a low-cost, low-power technology, mainly used to establish wireless personal area networks (WPANs) to replace the cables connecting devices. Bluetooth is used in many business and consumer devices, including cell phones, laptops, automobiles, keyboards, printers, headsets, and, more recently, medical appliances and personal devices (such as smartwatches, music speakers and home appliances). Bluetooth users can form ad-hoc networks between various devices to transfer both data and voice. Bluetooth devices incorporate to establish small wireless networks on an ad-hoc basis, known as Piconets.

Bluetooth is being managed by the Bluetooth Special Interest Group (SIG), which has over 35,000 member companies in telecommunication, computing, networking, and consumer electronics. Bluetooth technology has two forms: Basic Rate (BR) and Low Energy (LE). Both systems utilize device discovery, connection establishment, and connection mechanisms. Bluetooth Basic Rate (BR) includes an optional Enhanced Data Rate

(EDR) extension. Chapter 2 provides an overview of Bluetooth BR/EDR, where Chapter 3 discusses the Bluetooth Low Energy (BLE) case. Bluetooth 4.0 and later versions may have both BR/EDR and Low Energy as a "Dual Mode" Bluetooth device.

1.4 Brief History of Bluetooth

Ericsson initially conceived Bluetooth in 1994. It was named after King Harald "Bluetooth" Gormsson of Denmark, who helped unify warring factions in the 10th century CE [6]. Later, Ericsson, IBM, Intel, Nokia, and Toshiba formed what is known today as the Bluetooth Special Interest Group (SIG), a not-for-profit trade association developed to drive Bluetooth products' development and serve as the governing body for Bluetooth specifications.

Bluetooth is standardized within the IEEE 802.15 Working Group for Wireless Personal Area Networks that was formed in 1999 as IEEE 802.15.1-2002.4 [15]. Bluetooth Low Energy was introduced in Bluetooth version 4.0 and released in 2010. The latest Bluetooth core specification 5.3 combines the specifications of both Bluetooth BR/EDR and BLE.

Chapter 2

Bluetooth BR/EDR: Architecture and Operations

In this chapter, we present some details regarding Bluetooth Basic Rate technology that will be referenced in the remainder of the thesis. Here, we describe the architecture and operation of Bluetooth Basic Rate.

2.1 Bluetooth BR/EDR overview

Bluetooth versions 1.1 and 1.2 introduced what is known as Bluetooth Basic Rate (BR), supporting transmission speeds up to 1 megabit per second (Mbps) and achieving a payload throughput of roughly 720 kilobits per second (kbps). In Bluetooth 2.0, the Enhanced Data Rate (EDR) was introduced to provide data rates up to 3 Mbps and a payload throughput of approximately 2.1 Mbps. Bluetooth BR utilizes a binary Gaussian Frequency-Shift Keying (GFSK) modulation; on the other hand, EDR can use a $\pi/4$-rotated Differential Quaternary Phase-Shift Keying (DQPSK) and a 8-Phase Differential Phase-Shift Keying (8DPSK) modulation. EDR is supported in Bluetooth 2.0 and backward compatible with later versions. EDR support is not required for devices that comply with Bluetooth 2.0 specifications or later. Accordingly, some devices might be "Bluetooth 2.0 compliant" and others "Bluetooth 2.0 + EDR compliant."

2.2 Bluetooth BR/EDR Architecture

Bluetooth devices only communicate by establishing ad-hoc networks. Ad-hoc networks allow easy connection between devices in the same physical area without using any infrastructure. A Bluetooth Client is a device with a Bluetooth radio and software incorporating the Bluetooth protocol stack and interfaces. The Bluetooth specification conceptually defines a Host and a Controller, separating their duties to perform different stack functions. The Host handles the higher layer protocols, such as Logical Link Control and Adaptation Protocol (L2CAP) and Generic Access Profile (GAP). On the other hand, the Controller handles the lower layers, including the Radio, Baseband, and Link Control/Management. The Host and the Controller exchange data and commands using standardized communications over the Host Controller Interface (HCI). Figure 2.1 depicts BR/EDR system architecture [9].

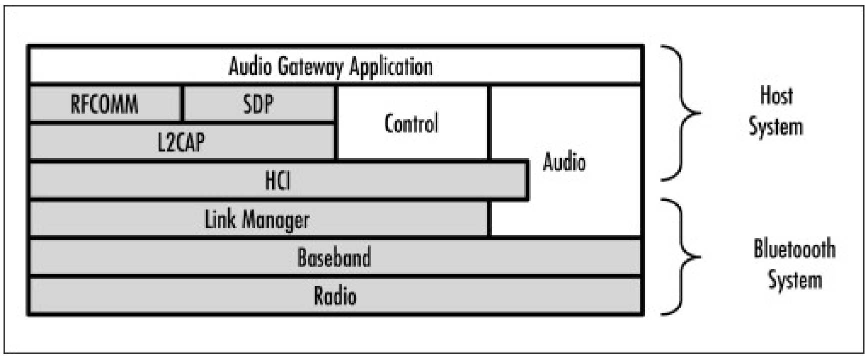


Figure 2.1: BR/EDR system architecture.

2.2.1 Controller

The Bluetooth Controller contains the radio frequency components that enable a Bluetooth device to transmit and receive data over the 2.4 GHz frequency band. In addition, it also performs cryptographic procedures. It will have access to physical sources of randomness and contain an encryption engine to enable the transmission of encrypted or authenticated data. Architecture-wise, the Controller comprises the Radio Layer, the Baseband Layer, Device Manager, Link Manager, and the Link Controller.

2.2.1.1 Radio Layer

Bluetooth operates in the 2.4 GHz ISM (Industrial Scientific Medical) band; the frequency range goes from 2400 MHz to 2483.5 MHz. RF channels are divided into 79 orthogonal narrow-band channels spaced 1 MHz. Bluetooth Radio utilizes a FHSS (Frequency Hopping Spread Spectrum) technique to reduce interference and fading, where it hops through the full spectrum of 79 channels using a pseudorandom hopping sequence. The Radio Layer has two modulation schemes: for Basic Rate, it uses a Gaussian-shaped, binary FM modulation to minimize transceiver complexity; for the Enhanced Data Rate, two types of PSK modulation are used: $\pi/4$-DQPSK and 8DPSK. The symbol rate for all modulation modes is 1 Msym/s. Thanks to the different modulation schemes, the data rate is 1 Mb/s for Basic Rate, 2 Mb/s for Enhanced Data Rate using $\pi/4$-DQPSK, and 3 Mb/s using 8DPSK. A Time Division Duplex (TDD) scheme is used in all modes.

2.2.1.2 Baseband Layer

The Baseband Layer performs several essential tasks to enable Bluetooth functionality. Some of the responsibilities of the Baseband Layer include defining packet structure, advertising, device discovery, connection initiation and management, and data transmission and reception to and from a connected device.

Packet structure: As shown in Figure 2.2 Bluetooth BR packet has three main fields. Every packet starts with an *Access Code*. If a packet header follows, the access code is 72-bits long; otherwise, the access code is 68-bits long and is known as a shortened access code. Its value differs across different connections, and it is used for synchronization and connection identification. The *Header* is 18-bits containing the Link Control (LC) information and comprises six fields shown in Figure 2.3 and detailed below.

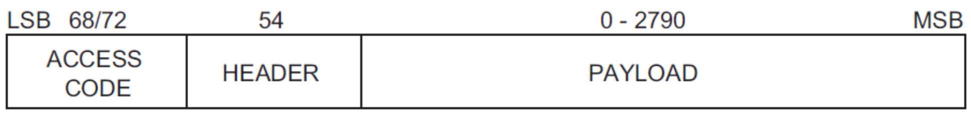


Figure 2.2: Bluetooth BR Packet Format

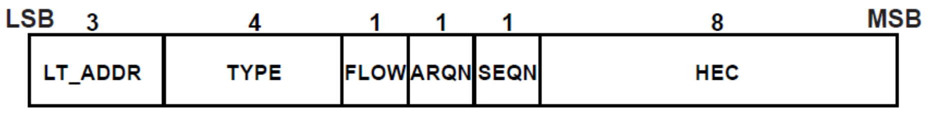


Figure 2.3: Bluetooth BR Packet Header

- *LT_ADDR*: 3-bits indicates the Peripheral source/destination of a Central-Peripheral transmission.
- *Type*: is a 4-bits that identifies the type of the packet.
- *FLOW*: implements the flow control of the packets over the ACL transport.
- ARQN: a 1-bit field to report a successful transfer of payload data with the CRC.
- *SEQN*: a 1-bit field to provide a sequential numbering scheme to order the data stream.
- *HEC*: 8-bits to check the packet header integrity.

The Header is encoded with a rate of 1/3 FEC—Forward Error Correction. The *Payload* contains the actual data and can fit up to 2790-bits of data of different packet types.

Asynchronous and Synchoronous Links: The Baseband layer provides adequate mechanisms for the data transfer over a Bluetooth link; the standard has several protocols and different links to ensure that the Bluetooth link is managed most effectively. Mainly there are two link types: the Asynchronous Connectionless communications link (ACL) and the Synchronous Connection Orientated communications link (SCO). ACL is the main link type, and it's used for exchanging framed data. An ACL link creates a packet-switching connection between the connection pair; packets are exchanged sporadically and only when data are available to be sent from the top levels of the Bluetooth stack. The slots are given to satisfy the quality of service requirement of each ACL. On the other hand, SCO is used to stream data through a symmetrical link between a connection pair; This data is typically an encoded voice stream. The data is transferred periodically in reserved time slots to be streamed without delay. Because voice transmissions are time-dependent, SCO packets are never retransmitted, so any packet that is not received correctly is lost.

2.2.2 Host Controller Interface (HCI)

The HCI layer is an element of the Bluetooth specification that allows the host layer to communicate with the controller layer. These two layers could exist in separate chipsets or live in the same chipset. The HCI provides interoperability between the two layers, so a device developer can choose two different Bluetooth-certified chipsets to implement the controller and the host, and be 100% confident that they are compatible with each other. The HCI layer has to be implemented over a physical communication interface if the host and controller are in separate chipsets. The HCI layer will be a logical interface if the host and the controller exist on the same chipset. The HCI layer relay commands from the host to the controller and sends events from the controller to the host.

2.2.3 Host

The Host is responsible for the following tasks: security, multiplexing, and disclosing a device's state data. It is composed of different protocols such as the Logical Link Control and Adaptation Protocol (L2CAP), the Attribute Protocol (ATT), and the Service Discovery Protocol (SDP). It also contains one main profile: the Generic Access Profile (GAP). We will see later that while the Controller operations are different in Bluetooth BR/EDR and BLE, the Host is roughly similar in BR/EDR and in BLE.

2.2.3.1 Logical Link Control and Adaptation Protocol (L2CAP) Layer

The Logical Link Control and Adaptation Protocol (L2CAP) layer is a protocol multiplexing layer. It takes multiple upper layers protocols and places them in standard Bluetooth packets passed to the lower layers. The L2CAP layer also fragments the larger upper layers packets so they can fit into the maximum-allowed payload size. On the receiver side, it takes multiple packets and combines them into one packet that the upper layers can handle.

2.2.3.2 Attribute Protocol (ATT)

In Bluetooth, the data is stored as discrete values called attributes and accessed through the Attribute Protocol (ATT). A Bluetooth device can use the ATT Protocol to read or write this data. The ATT uses a client-server model, in which the client accesses attributes from the server. Both the Master (Central) and the Slave (Peripheral) can act as both client and server. Each attribute consists of a value and its three properties: attribute type, attribute handle, and access permissions. The attribute type is described by a universally unique identifier (UUID). These UUIDs are specified either by the Bluetooth SIG or by the manufacturer. An attribute handle is a 16-bit number, which allows the client to specify the attribute in requests. An attribute's access permissions describe how the associated value can be accessed, e.g., if reading or writing is allowed. Security features like encryption, authentication, or authorization cannot be queried directly by the ATT.

2.2.3.3 Generic Access Profile

The Generic Access Profile (GAP) represents the base functionality of all Bluetooth devices. It describes profile roles and defines modes and procedures for the discoverability, connection, and security of Bluetooth devices. GAP utilizes the features provided by the other layers of the stack and guarantees interoperability between devices of different manufacturers.

2.3 Bluetooth Network Topology

Bluetooth devices are provided with point-to-point or point-to-multipoint, wherein the physical channel will be shared among the connection partners. Devices sharing the same physical channel form what is called a **Piconet**. Only one device will be the Piconet Central (which controls the channel access), whereas the other devices (up to seven) act as

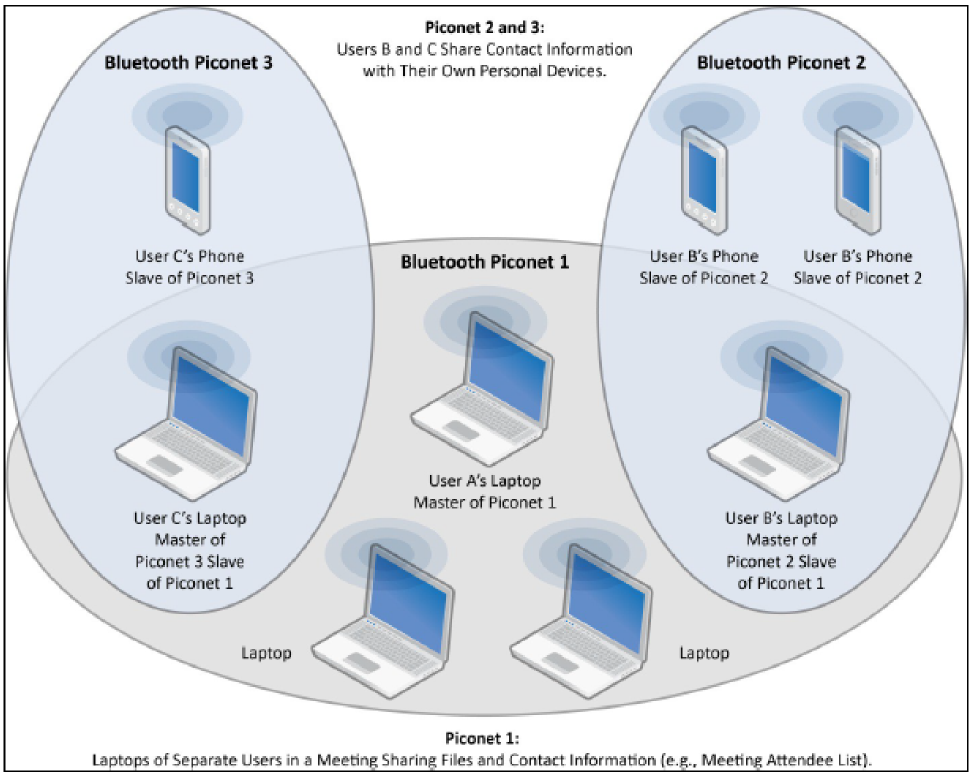

Piconet 2 and 3:
Users B and C Share Contact Information with Their Own Personal Devices.
Bluetooth Piconet 3
Bluetooth Piconet 2
User C's Phone Slave of Piconet 3
Bluetooth Piconet 1
User B's Phone Slave of Piconet 2
User B's Phone Slave of Piconet 2
User A's Laptop Master of Piconet 1
User C's Laptop Master of Piconet 3 Slave of Piconet 1
User B's Laptop Master of Piconet 2 Slave of Piconet 1
Laptop
Laptop
Piconet 1:
Laptops of Separate Users in a Meeting Sharing Files and Contact Information (e.g., Meeting Attendee List).

Figure 2.4: Bluetooth Networks (Multiple Scatternets)

Peripherals. All the Piconet devices communicate on the same channel by synchronizing to a common clock and hopping sequence. A Bluetooth device can participate concurrently in two or more Piconets "acting as a Peripheral" utilizing a time-division multiplexing basis. However, it can never be a Central in more than one piconet. More than two connected Piconets form what is known as a **Scatternet**. Figure 2.4 shows an example of three connected Piconets in which they form a Scatternet. Devices from Piconet 3 can communicate to devices in Piconet 2 through Piconet 1 [15].

Chapter 3

Bluetooth LE: Architecture and Operations

In this chapter, we present details regarding Bluetooth Low Energy (BLE). We describe its architecture and operation.

3.1 What is Bluetooth Low Energy?

Bluetooth Low Energy (BLE) is a new technology that has been designed as complementary to Bluetooth BR/EDR. Although it uses the Bluetooth brand and borrows much technology from its parent, Bluetooth Low Energy is in fact a different technology, addressing different design goals and different market segments [7]. BLE was introduced as Bluetooth v4.0 in 2010, but it was formerly known as "Wibree" and "Ultra Low Power Bluetooth". BLE is more prominent in applications where power consumption is critical (such as battery-powered devices) and small amounts of data are transferred infrequently (such as in sensor applications).
The two types of Bluetooth devices are incompatible even though they share the same brand and specification document. A Bluetooth BR/EDR device cannot communicate (directly) with a BLE device. That is why some devices, such as smartphones choose to implement both types (also called Dual Mode Bluetooth devices), allowing them to communicate with both types of devices. The key technology goals of BLE (compared with Bluetooth BR/EDR) include lower power consumption, reduced memory requirements, efficient discovery and connection procedures, short packet lengths, and simple protocols and services [15]. Like Bluetooth BR/EDR, BLE operate in the 2.4 GHz ISM (Industrial, Scientific, Medical) frequency spectrum [6].

3.2 BLE Architecture

Figure 3.1 shows the different layers within the architecture of BLE [6]. The three main layers in the architecture of a BLE device are the Application, the Host, and the Controller. The Host can be the same for BR/EDR and BLE, but the lowest layers of the stack shown in Figure 3.1 are not compatible with BR/EDR. For this reason, in this chapter we will briefly review only the link layer and physical layer definitions. A more detailed operational description of the connection procedure will be presented later in chapter 7.

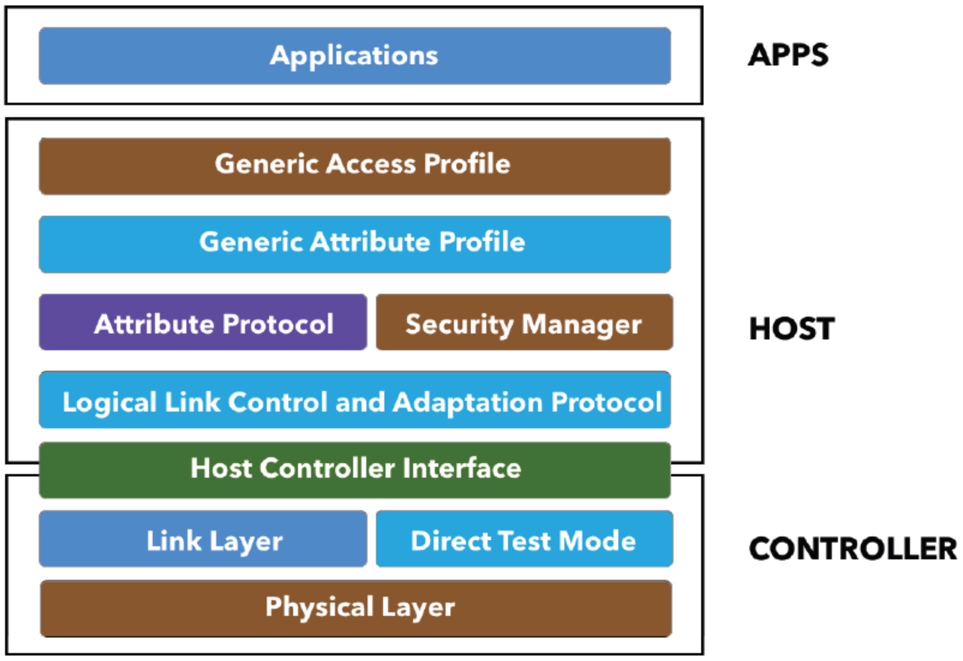


Figure 3.1: BLE Protocol Stack

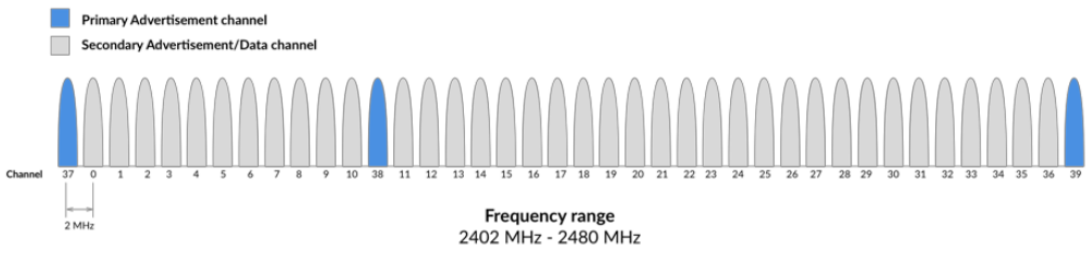


Figure 3.2: BLE frequency spectrum and RF channels

3.2.1 Physical Layer (PHY)

The physical layer (PHY) refers to the radio hardware used to communicate and modulate/demodulate the data. BLE operates in the 2.4 GHz ISM band, which is segmented into 40 RF channels, each separated by 2 MHz (center-to-center), as shown in Figure 3.2: Three of the 40 channels are named the Primary Advertising Channels, while the remaining 37 channels are used for Secondary Advertisements and data transfer during a connection. BLE employs Frequency Hopping Spread Spectrum (FHSS), which allows the two communicating devices to switch to randomly (agreed-on) selected frequencies for exchanging data. This dramatically improves reliability and allows the devices to avoid frequency channels that may be congested and used by other devices in the surrounding environment.
The minimum transmit power is 0.01 mW (-20 dBm) in all Bluetooth versions, where the maximum is 100mW (+20 dBm) starting from version 5 and later versions; and 10mW (+10 dBm) for v4.2 and older versions. BLE data rate was fixed at 1 Mbps in older versions of Bluetooth (4.0, 4.1, and 4.2); in this case, the physical layer radio (PHY) is the 1M PHY and is mandatory in all versions, including Bluetooth 5. With Bluetooth 5, however, two new optional PHYs were introduced: 2Mbps PHY, which achieves twice the speed of earlier versions of Bluetooth, and the Coded PHY to be used for longer-range communication.

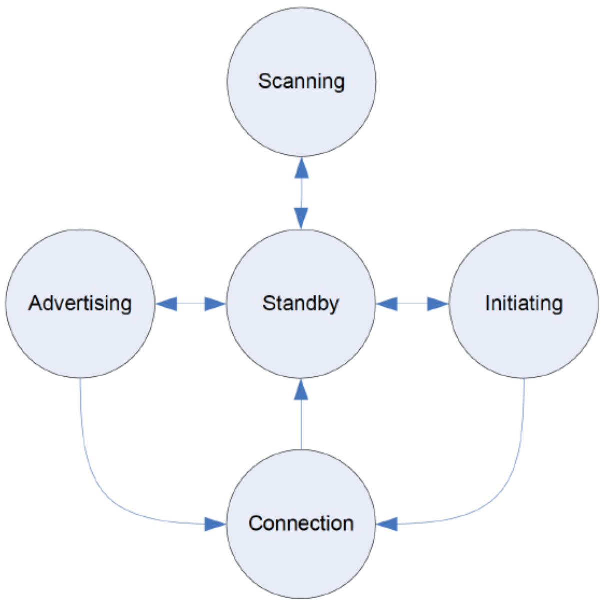


Figure 3.3: Link Layer States

3.2.2 Link Layer

The Link Layer (LL) defines the interface to the physical layer (radio). It provides the higher-level layers an abstraction and a way to interact with the radio (through the intermediary layer HCI—Host Controller Interface—which is on top of the Link Layer, while the radio (physical) is below the LL as shown in Figure 3.1). The Link Layer manages the radio's state and the timing requirements necessary for satisfying the BLE specification. It is also responsible for managing hardware-assisted operations such as CRC computation, random number generation, and encryption. A BLE device operates in three main states: Advertising state, Scanning state and Connected state. An **advertising** device allows other **scanning** devices to find it. If the advertising device allows connections and a scanning device decides to connect to it, they both enter into the **connected** state. Figure 3.3 shows how the Link Layer manages the different states of the radio.

- *Standby* is the default state in which the radio does not transmit or receive data.
- *Advertising* is the state in which the device sends out advertising packets for other devices to discover and read.
- *Scanning* is the state in which the device scans for devices that are advertising.
- *Initiating* is the state in which a scanning device decides to establish a connection with a device that is advertising.
- *Connected* is the state where a device has an established link with another device and regularly exchanges data with another device. This applies to both a device in the advertising state or one scanning for advertisements and then deciding to initiate a connection with the advertising device. In this connected state, the device that initiates the connection is called the **master**, and the advertising device is now called the **slave**.

BLE defines two transmission types: data and advertising. Three of BLE's 40 channels are dedicated to advertising (i.e., channels 37, 38, and 39), known as the Primary Advertising Channels, and they can fit up to 31-bytes of data. On the other hand, the remaining 37 channels, known as the Secondary Advertising Channels, can be used for data transmission and advertising; the transmitted data on these channels can be up to 254-bytes. All peripherals must go through advertising mode, whether sending beacons (e.g., transmitting location, weather, or other data) or a device (i.g., smartwatch) making a long-term connection with a host (e.g., phone).
The Link Layer maintains a unique device identifier (i.e., Bluetooth Device Address, or BD_ADDR), which distinguishes different communication partners, as we will see in the next section.

3.2.2.1 BLE Address

All Bluetooth devices (BR/EDR and BLE) are identified by a 48-bit address, similar to a MAC address. There are two main types of addresses: Public Addresses and Random Addresses. Bluetooth manufacturers have a choice of what type of address to use.

3.2.2.1.1 Public Address

It is a factory-programmed address. It is a fixed address and must be registered with the IEEE (similar to the WiFi and Ethernet device MAC address).

3.2.2.1.2 Random Address

A random address is either programmed on the device or generated at runtime. Random addresses are more popular than public addresses since they do not require IEEE registration. It can be one of two sub-types:

Static Address: It can be used as a replacement for Public addresses; it can be generated at boot up OR stay the same during the lifetime, and it cannot change until a power cycle.

Private Address: This type of address is grouped into two additional sub-types:

- **Non-resolvable Private Address:** Is random and temporary (for a specific amount of time). Not even paired devices can determine the corresponding identity address when in use, meaning that privacy is protected even from trusted devices. This type of address is not commonly used.
- **Resolvable Private Address:** Used for privacy, generated using Identity Resolving Key (IRK) and a random number. Changes periodically (even during the lifetime of the connection). This type is used to avoid being tracked by unknown scanners. Trusted devices (or Bonded) can resolve the address using the previously stored IRK (The procedure is detailed in Chapter 5, Section 5.6).

Chapter 4

Bluetooth BR/EDR Security

Security has always been one of the main concerns for technology manufacturers [1]. In this chapter, we summarize the Bluetooth BR/EDR security mechanisms as described in Volume 2, Part H of the Bluetooth Core Specifications [16], aiming to provide a better understanding to the reader.

4.1 BR/EDR Security Overview

Both Bluetooth BR/EDR and BLE offer some important security services, like:

Pairing/Bonding: The process of creating one or more shared secret keys and storing those keys to be used in subsequent connections to form a trusted device pair.

Authentication: Verifying that the connection pair have the right keys and verifying their identity based on the Bluetooth address.

Authorization: Controlling the resources by authorizing the right users to access the right services.

Confidentiality: Guaranteeing data delivery without a third (unauthorized) party being able to view its content.

Integrity: Guaranteeing that the message between the connection pair has been transmitted and received without being altered by forgeries.

Privacy: Privacy is indicated by how private the communication is, the identity of the communication pair, and whether the Bluetooth address is traceable.

In the following sections, we will only detail the most common Bluetooth security services, i.e., Pairing, Authentication, and Confidentiality. Bluetooth BR/EDR security services and modes of operation are explained in the following sections. Unfortunately, Bluetooth does not support audit and non-repudiation. Hence, if required, it has to be provided by other means.

4.2 BR/EDR Security Modes

The Bluetooth BR/EDR security model introduces the concept of security modes (from Mode 1 to 4). Each Bluetooth device must operate in one of these modes. These modes indicate when and how a Bluetooth device initiates security[15].

4.2.1 Security Mode-1

In this mode, security features (including authentication and encryption) are disabled. The device is considered non-secure and can participate in any connection. Mode-1 is supported up to version 2.0. From v2.1, Bluetooth devices can operate in Security Mode-1 for backward compatibility.

4.2.2 Security Mode-2

It is a service-level-enforced security mode. Security is initiated after link establishment but before logical channel establishment. The link manager maintains policies for access control and interfaces with other protocols and device users, granting access to some services and denying access to other services (authorization). Similar to Mode-1, Security Mode-2 is supported up to v2.0. From v2.1, Bluetooth devices use Security Mode-2 for backward compatibility.

4.2.3 Security Mode-3

Security Mode-3 is the link-level-enforced security mode. Bluetooth devices initiate security before the physical link is fully established. Authentication and Encryption are mandatory for all connections. Therefore, even service discovery cannot be held until authentication and Encryption have been initiated. All Bluetooth v2.0 and earlier versions support Security Mode-3, while v2.1 and later only keep it for backward compatibility.

4.2.4 Security Mode-4

Similar to Security Mode-2, it's a service-level-enforced security mode. The difference is that Security Mode-4 uses Secure Simple Pairing (SSP), in which ECDH key agreement is used for link key generation (see Section 4.4.2). Until Bluetooth v4.0, Elliptic Curve P-192 is used for the link key generation. Bluetooth v4.1 introduced the Secure Connections Pairing method, where the Elliptic Curve P-256 is used for link key generation. In v4.1, the authentication algorithm is upgraded to the FIPS-approved (HMAC-SHA-256). The encryption algorithm is also upgraded to the FIPS-approved (AES-CCM), which provides message integrity. Security requirements for services protected by Security Mode-4 are classified in different levels:

- Level-0: No security required. (Only allowed for SDP–Service Discovery Protocol)
- Level-1: No security required
- Level-2: Unauthenticated link key required
- Level-3: Authenticated link key required
- Level-4: Authenticated link key using Secure Connections required

4.3 Pairing

Pairing is the process of creating a shared secret key (the link key) between two Bluetooth devices that will be used to mutually authenticate the devices and establish the encryption key. Bluetooth BR/EDR can perform pairing in two ways:

- Personal Identification Number `(PIN)` Pairing (Legacy Pairing).
- Secure Simple Pairing `(SSP)`.

4.3.1 PIN/Legacy Pairing

PIN/legacy pairing is used in Security Modes 2 and 3. Two Bluetooth devices simultaneously derive and agree on a link key when the user(s) enter an identical secret PIN into one or both devices. The key derivation will be discussed in more detail in Section 4.4.1. The PIN length varies between 1 to 16-Bytes. If the PIN is less than 16-Bytes, the initiating device will add its BD_ADDR to the PIN value to generate the initialization key.

4.3.2 Secure Simple Pairing

Secure Simple Pairing `(SSP)` (i.e., Mode-4) was first introduced in Bluetooth v2.1 and then improved in v4.1. Compared to PIN/Legacy Pairing, `SSP` simplifies the pairing process by providing four flexible association models that translate the device input/output capabilities. `SSP` also improves security by adding ECDH public key cryptography (P-192 or P-256) for protection against passive eavesdropping and Man-in-the-Middle (MitM) attacks during pairing. `SSP` association models are Numeric Comparison, Passkey Entry, Out of Band, and Just Works.

4.3.2.1 Numeric Comparison

Numeric Comparison is designed for scenarios where both devices have a display and are capable of entering "yes" or "no." An example of this model is the cell phone /PC scenario. The user is shown six numeric digits on both displays and asked whether the numbers are identical. If "yes" selected on both devices, the pairing is **successful**, otherwise **failed**. Numeric Comparison provides protection against the MitM attack.

4.3.2.2 Passkey Entry

Passkey Entry is designed for scenarios where one of the connection pairs has only input capability (i.e., keyboard) and the other peer has only a display. The user is shown a six-digit number on the device with the display and is asked to enter it in the device with the input capability. If the value matches, the pairing is **successful**, otherwise **failed**.

4.3.2.3 Out of Band "OOB"

OOB is designed for scenarios where an Out of Band mechanism (e.g., Near Field Communication (NFC)) is used to discover the devices and exchange cryptographics values used in the pairing process. When NFC is in use, OOB model allows devices to pair by just "tapping" one device on the other, followed by the user accepting the pairing via a single push button. The OOB technology should be designed and configured to thwart eavesdropping and MitM attacks.

4.3.2.4 Just Works

As the name suggests, Just Works (`JW`) is the simplest and weakest association model. It's primarily designed for scenarios where one of the two devices does not provide either a display or keyboard. Just Works operating procedure is similar to the Numeric Comparison one, but the user is never shown a number and must accept the connection without verifying the calculated values on both devices. An example of this model is the cell phone/headset scenario, where most headsets do not have IO capability. (`JW`) does not offer protection against the MitM attack.

"Secure Connection"-Only Mode

A BR/EDR device using SSP is said to be in Secure Connection mode when it is required only to use FIPS-approved algorithms. The user is forced to use Elliptic-curve Diffie–Hellman (ECDH) P-256 algorithm during pairing. On the BR/EDR physical transport, secure authentication sequences are used, and AES-CCM is used for encryption. Secure Connections Only Mode is also called a "FIPS Mode."

4.4 Link Key Establishment

In section 4.3, we have seen that a link key is generated as a result of a successful pairing. The link key is a 128-bit random number that is shared between two or more Bluetooth devices and is known to be the base for all security transactions between these parties. In the following sections, we will have more details on how the link key is generated using the two pairing methods.

4.4.1 Legacy Pairing/PIN

In the case of Legacy Pairing, the link keys are either semi-permanent or temporary. A semi-permanent key is saved in the device memory and can be used for several sessions, while the temporary is only used in one session. The link key can be sorted into one of three types:

- Combination key K_{COMB}: is derived from information related to a pair of devices, $Device_1$ and $Device_2$. It is derived for every new combination of two devices.
- Temporary key K_{temp}: only used during the current session. It temporarily replaces the original link key (e.g., to broadcast data).
- Initialization key K_{init}: is used as the link key only during the initialization process, when the combination key is not yet generated, or a link key has been lost.

4.4.1.1 Generation of the initialization key K_{init}

The initialization key is derived using the E_{22} algorithm from a 128-bit random number `IN_RAND` and a `PIN` code of L bytes. The E_{22} algorithm is a custom version of the SAFER+ block cipher, which has a block size of 128-bit keys. `PIN`s with a length L shorter than the key length will be augmented using the least significant bits of the `BD_ADDR`, thus making $\texttt{PIN}'$ with the size of 128-bits.

$$K_{init} = E_{22}(\texttt{PIN}', \texttt{IN\_RAND}, L) \tag{4.1}$$

4.4.1.2 Generation of a combination key

The key combines two numbers generated in $Device_1$ and $Device_2$, respectively. The process starts with both devices generating a random number LK_{RAND_1} and LK_{RAND_2}; those values are going to be XORed with the initialization key (K_{init}) that have been shared in the preceding stage in both devices. The result will be K_{COMB_1} and K_{COMB_2}.

$$K_{COMB_1} = K_{init} \oplus LK_{RAND_1} \tag{4.2}$$

$$K_{COMB_2} = K_{init} \oplus LK_{RAND_2} \tag{4.3}$$

Since the initialization key is already shared between the pair, so to know each other random number (LK_{RAND}). The devices will exchange their K_{COMB} securely so that both of them will be able to extract LK_{RAND} as follows:

$$LK_{RAND_1} = K_{init} \oplus K_{COMB_1} \tag{4.4}$$

$$LK_{RAND_2} = K_{init} \oplus K_{COMB_2} \tag{4.5}$$

Then, utilizing E_{21} algorithm (E_{21} is custom algorithm based on SAFER+ for link key derivation, used when generating a key from the 48-bit address) with the two random numbers and their own BD_ADDRs, the following values are:

$$K_1 = BD\_ADDR_1 \oplus K_{COMB_1} \tag{4.6}$$

$$K_2 = BD\_ADDR_2 \oplus K_{COMB_2} \tag{4.7}$$

The pair knows the Bluetooth Device Address of each other. Thus, both devices can re-compute K_1 and K_2. Hence, the final step will be generating the combination key (i.e., the link key), resulting from XORing both K_1 and K_2.

$$K_{LINK} = K_1 \oplus K_2 \tag{4.8}$$

If K_{LINK} is identical in both devices, then the devices should terminate the process of generating the combination key. After link key generation is complete, the devices complete pairing by mutually authenticating each other to verify that they have the same link key. The old link key shall be discarded after successfully exchanging a new combination key. The message flow between **Device1** and **Device2** and the principle for creating the combination key illustrated in Figure 4.1.

4.4.2 Secure Simple Pairing SSP

The Secure Simple Pairing procedure is an alternative and significantly more secure approach for pairing Bluetooth BR devices. Secure Simple Pairing (i.e., link key derivation) is carried out in four phases as follows:

- Phase-1: Public key exchange

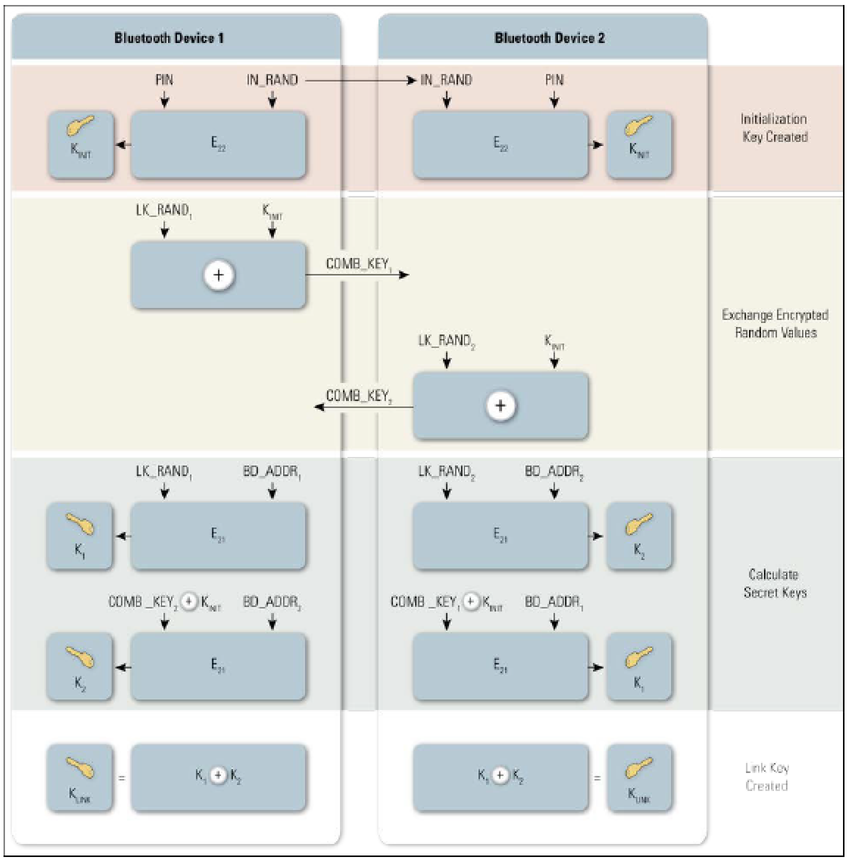


Figure 4.1: Link Key Generation from PIN (Legacy Pairing)

- Phase-2: Authentication stage-1
- Phase-3: Authentication stage-2
- Phase-4: Link key calculation

The steps to generate the link key in Secure Simple Pairing (using ECDH public/private key pairs) are illustrated in Figure 4.2.

4.4.2.1 Phase-1: Public key exchange

- Initially, each device generates its own Elliptic Curve Diffie-Hellman (ECDH) public-private key pair.
- Pairing is initiated by the initiating device (**Device1**) sending its public key (PK_1) to the responding device (**Device2**).
- **Device2** has to respond with its public key (PK_2). all the public keys must be validated against the correct curve (P-192 or P-256).

4.4.2.2 Phase-2: Authentication stage-1

Authentication stage-1 proceeds according to the association model: Numeric Comparison, Out-of-Band, and Passkey Entry. The Just Works association model shares the Numeric Comparison protocol and does not have a particular one. The association model is selected

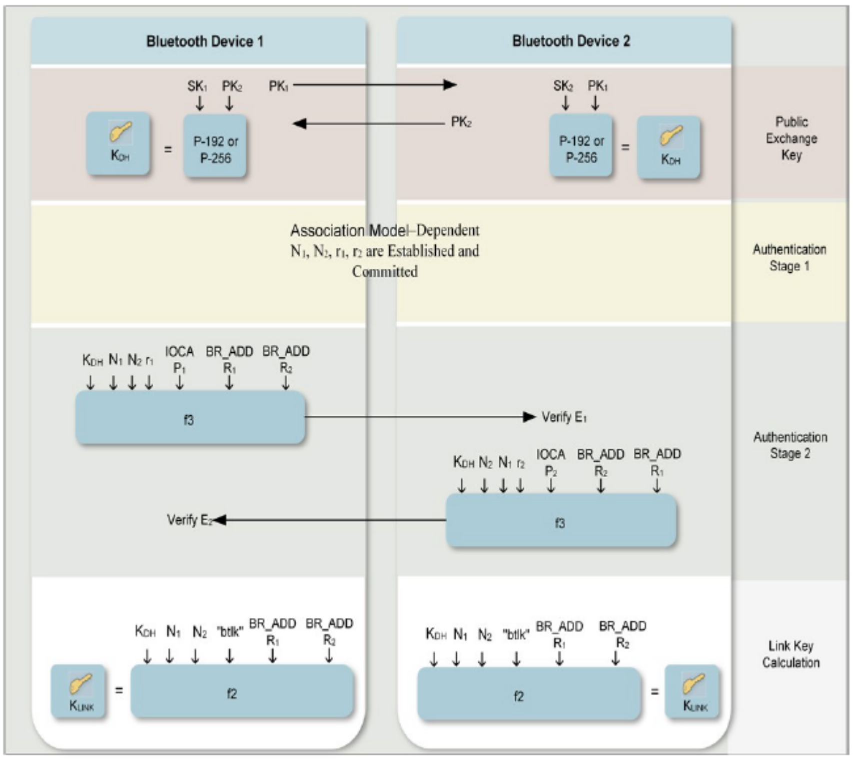


Figure 4.2: Link Key Establishment for Secure Simple Pairing

based on the IO capabilities of the two devices. In this section we will refer to $Device_1$ as $Device_A$ and $Device_2$ as $Device_B$.

Authentication stage-1: Numeric Comparison

Numeric Comparison is secure from MitM and passive eavesdroppers that may have been present during the pairing. Figure 4.3 depicts the details of the Numeric Comparison association model.

- After public keys exchange, each device selects a random 128-bit nonce (N_a and N_b). The value should be fresh with each instantiation of the pairing protocol.
- Using the f_1 function the responding device ($Device_B$) computes a commitment (C_b) as in EQ 4.9. For details of f_1 function, see Bluetooth Core Specification, Vol 2, Part H, Section 7.7.1 [16].
$$C_b = f_1(PK_b, PK_a, N_b, 0) \tag{4.9}$$
- The commitment is then transmitted to the initiating device ($Device_A$). The commitment prevents an attacker from changing these values at a later time.
- $Device_A$ and $Device_B$ exchange their nonce values (N_a and N_b).
- $Device_A$ confirms the commitment by recomputing C_b. A failure at this point causes the protocol to abort.
- A successful check at $Device_A$ results in both devices will use the g function to compute a 6-digit confirmation value (V_a and V_b) to be displayed to the user in both

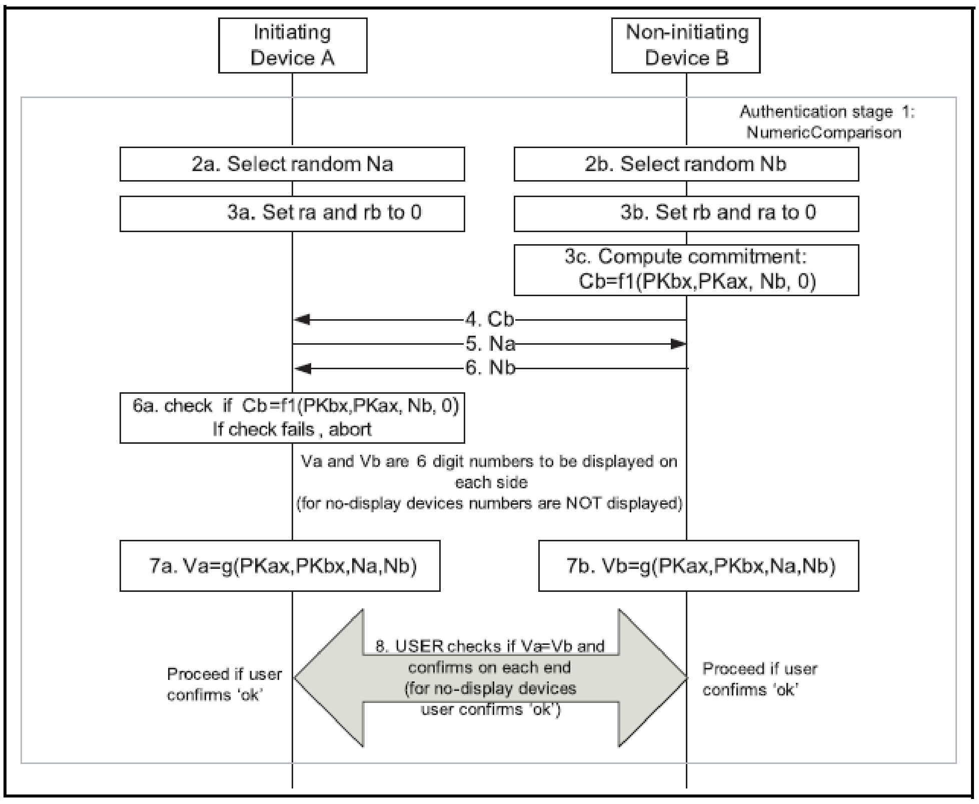

Initiating Device A
Non-initiating Device B
Authentication stage 1: NumericComparison
2a. Select random Na
2b. Select random Nb
3a. Set ra and rb to 0
3b. Set rb and ra to 0
3c. Compute commitment: Cb=f1(PKbx,PKax, Nb, 0)
4. Cb
5. Na
6. Nb
6a. check if Cb=f1(PKbx,PKax, Nb, 0) If check fails , abort
Va and Vb are 6 digit numbers to be displayed on each side (for no-display devices numbers are NOT displayed)
7a. Va=g(PKax,PKbx,Na,Nb)
7b. Vb=g(PKax,PKbx,Na,Nb)
Proceed if user confirms 'ok'
8. USER checks if Va=Vb and confirms on each end (for no-display devices user confirms 'ok')
Proceed if user confirms 'ok'

Figure 4.3: Authentication stage1: Numeric Comparison

devices. The user checks and confirms if the value matches on both devices. If not, the protocol aborts. For details of fg function, see Bluetooth Core Specification, Vol 2, Part H, Section 7.7.2 [16].

$$V_a = g(PK_a, PK_b, N_a, N_b) \tag{4.10}$$

$$V_b = g(PK_a, PK_b, N_a, N_b) \tag{4.11}$$

- When Just Works, the checks are not performed, and the user is never shown the 6-digit values.

Authentication stage-1: Passkey Entry

The Passkey Entry protocol is used when LMP–Link Manager Protocol– IO capability indicates that Passkey Entry shall be used. The user might enter the same six-digit Passkey into both devices, or one device displays a six-digit random number, and the user enters it into the other device. The Passkey will be the basis of the authentication of both devices. The steps of the link derivation are shown in Figure 4.4.

- The user enters an identical Passkey into both devices.
- A 6-digit Passkey is 20-bits long represented with r_a for $Device_A$ which is the concatenation of the 20-bits r_{a_i}. For $Device_B$ it's r_b and it's the concatenation of the 20-bits of r_{b_i}.

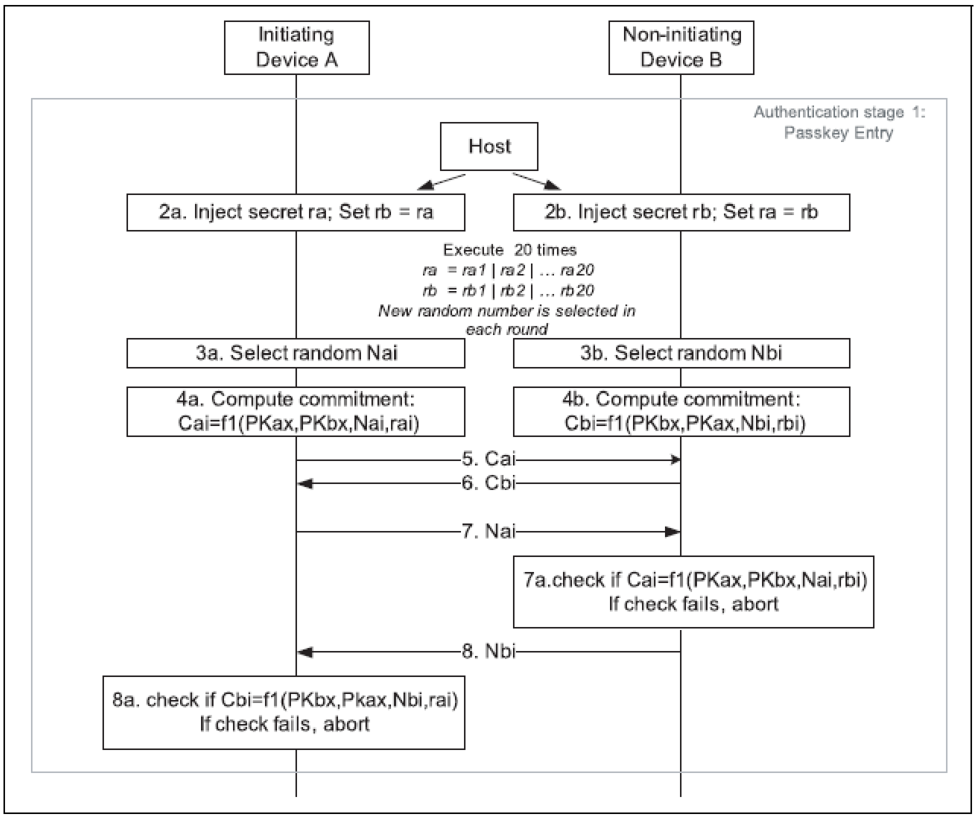

Initiating Device A
Non-initiating Device B
Authentication stage 1: Passkey Entry
Host
2a. Inject secret ra; Set rb = ra
2b. Inject secret rb; Set ra = rb
Execute 20 times
ra = ra1 | ra2 | ... ra20
rb = rb1 | rb2 | ... rb20
New random number is selected in each round
3a. Select random Nai
3b. Select random Nbi
4a. Compute commitment: Cai=f1(PKax,PKbx,Nai,rai)
4b. Compute commitment: Cbi=f1(PKbx,PKax,Nbi,rbi)
5. Cai
6. Cbi
7. Nai
7a.check if Cai=f1(PKax,PKbx,Nai,rbi) If check fails, abort
8. Nbi
8a. check if Cbi=f1(PKbx,Pkax,Nbi,rai) If check fails, abort

Figure 4.4: Authentication stage1: PassKey Entry

- Then, both devices will generate a nonce denoted by N_{a_i} and N_{b_i}.
- After, $Device_A$ and $Device_B$ will calculate the commitment values C_{a_i} and C_{b_i}. $Device_A$ and $Device_B$ exchange their C_{a_i} and C_{b_i} respectively.

$$C_{a_i} = f_1(PK_a, PK_b, N_{a_i}, r_{a_i}) \tag{4.12}$$

$$C_{b_i} = f_1(PK_b, PK_a, N_{b_i}, r_{b_i}) \tag{4.13}$$

- $Device_A$ sends it's N_{a_i}, while $Device_B$ checks C_{a_i} and if check succeed it replies with it's N_{b_i}.
- As a final step $Device_A$ checks the received C_{b_i}.
- The steps are repeated 20 times as the Passkey is 20-bits. At the end of this stage, N_a is set to $N_{a_{20}}$ and N_b is set to $N_{b_{20}}$ to be used in Authentication stage-2.

Authentication stage-1: Out of Band

The Out-of-Band protocol is used when at least one of the devices indicates the presence of the OOB parameter in the LMP–Link Manager Protocol– IO capability. In OOB, the discovery of the peer device is carried out in-band, and authentication parameters are transmitted through the OOB interface. Figure 4.5 shows the sequence diagram for Authentication stage1 for Out of Band.

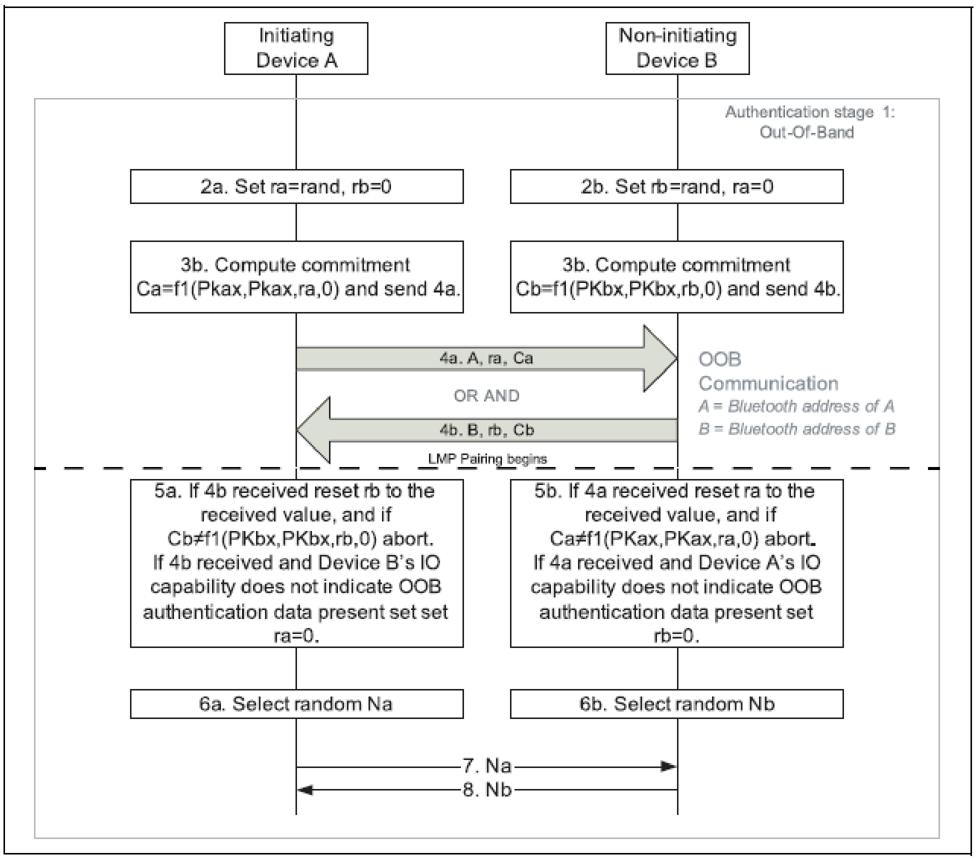

Initiating Device A
Non-initiating Device B
Authentication stage 1: Out-Of-Band
2a. Set ra=rand, rb=0
2b. Set rb=rand, ra=0
3b. Compute commitment Ca=f1(Pkax,Pkax,ra,0) and send 4a.
3b. Compute commitment Cb=f1(PKbx,PKbx,rb,0) and send 4b.
4a. A, ra, Ca
OR AND
4b. B, rb, Cb
OOB Communication
A = Bluetooth address of A
B = Bluetooth address of B
LMP Pairing begins
5a. If 4b received reset rb to the received value, and if Cb≠f1(PKbx,PKbx,rb,0) abort. If 4b received and Device B's IO capability does not indicate OOB authentication data present set set ra=0.
5b. If 4a received reset ra to the received value, and if Ca≠f1(PKax,PKax,ra,0) abort. If 4a received and Device A's IO capability does not indicate OOB authentication data present set rb=0.
6a. Select random Na
6b. Select random Nb
7. Na
8. Nb

Figure 4.5: Authentication stage1: Out of Band

- Each device generates a random number, (r_a and r_b), and set the other pair random number equal to zero ($r_b = 0$ in $Device_A$ and $r_a = 0$ in $Device_B$).
- They each calculate a commitment value,(C_a and C_b) as follows:

$$C_a = f_1(PK_a, PK_a, r_b, 0) \tag{4.14}$$

$$C_b = f_1(PK_b, PK_b, r_a, 0) \tag{4.15}$$

- $Device_A$ now sends its Bluetooth device address `(A)` (that will be used when in-band pairing proceeds) together with r_a and C_a to $Device_B$ through OOB.
- $Device_B$ sends it's corresponding data values (Bluetooth device address `(B)`, C_b, and r_b) back to $Device_A$.
- In this step, the connection pair communicate through `In-Band`, and they share their public keys (PK_a, and PK_b) in public.
- Both devices will use the received values `Out-of-Band` and `In-Band`. $Device_A$ recompute C_b and $Device_B$ recompute C_a as in EQ 4.14, and 4.15. If the values are equal to what they have received, they authenticate each other; otherwise, the process is aborted. As a result of successful authentication, they generate new random numbers (N_a and N_b) and exchange them for later procedures.

When only one device has OOB authentication data available, the communication of authentication data will be from one device to the other and not in the other direction. If both devices can transmit or receive data over an OOB channel, mutual authentication will be based on the commitment values (C_a and C_b) exchanged in OOB Authentication.

4.4.2.3 Phase-3: Authentication stage-2

In Authentication Stage-2, further checks are made to ensure that the exchange of public keys, random numbers, and device addresses was completed correctly in the earlier steps. This stage is identical in all the association models. The the sequence diagram for Authentication stage-2 is shown in Figure 4.6.

- Each device will employ the f_3 function to compute a new confirmation value (E_a and E_b) that includes the previously exchanged values and the newly derived shared key. For details of f_3 function see Bluetooth Core Specification, Vol 2, Part H, Section 7.7.4 [16].

$$E_a = f_3(DHKey, N_a, N_b, r_b, IOcapA, A, B) \tag{4.16}$$

$$E_b = f_3(DHKey, N_b, N_a, r_a, IOcapB, B, A) \tag{4.17}$$

- $Device_A$ transmits E_a to $Device_b$, where it is being checked. If this check fails, it indicates no pairing, and the process is aborted.
- $Device_b$ then transmits its confirmation value E_b, which is checked by $Device_A$. If this check fails, it indicates no pairing, and the process is aborted.

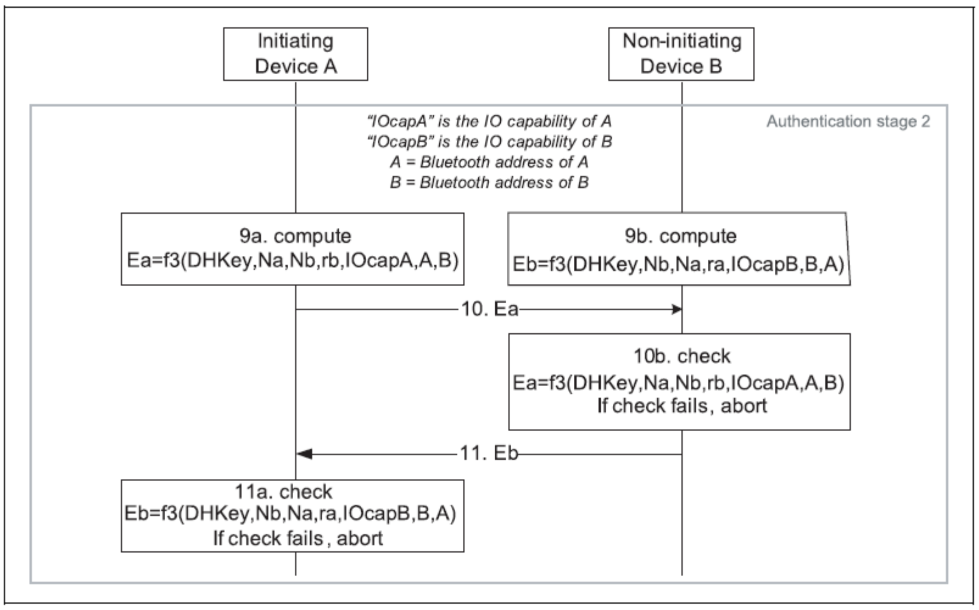

Initiating Device A
Non-initiating Device B
Authentication stage 2
"IOcapA" is the IO capability of A
"IOcapB" is the IO capability of B
A = Bluetooth address of A
B = Bluetooth address of B
9a. compute Ea=f3(DHKey,Na,Nb,rb,IOcapA,A,B)
9b. compute Eb=f3(DHKey,Nb,Na,ra,IOcapB,B,A)
10. Ea
10b. check Ea=f3(DHKey,Na,Nb,rb,IOcapA,A,B) If check fails, abort
11. Eb
11a. check Eb=f3(DHKey,Nb,Na,ra,IOcapB,B,A) If check fails, abort

Figure 4.6: Authentication Stage2 SSP

4.4.2.4 Phase 4: Link Key Calculation

Now both devices are paired with each other. As a final pairing step, the link key is computed using the f_2 function with the Diffie-Hellman shared key and the exchanged data. f_2 function uses the AES-CMAC algorithm with DHKey as its 128-bit key. The link key will be calculated in the same order on both sides as in EQ 4.18.

$$K_{LINK} = f_2(DHkey, N_a, N_b, \text{“}btlk\text{”}, BD\_ADDR_A, BD\_ADDR_B) \tag{4.18}$$

4.5 Authentication

BR/EDR authentication procedure is of two types; either Legacy Authentication (Section 4.5.1) or Secure Authentication (Section 4.5.2). If the authentication fails, a Bluetooth device waits an interval of time before making new attempts. This time interval increases exponentially to prevent intruders from repeating the authentication procedure with different keys (*bruteforce* attack). This technique does not provide any security against offline attacks attempting to know the link key using eavesdropped pairing frames and exhaustively guessing PINs[16].

4.5.1 Legacy Authentication

Legacy Authentication is used when at least one device does not support Secure Connections. The devices interacting in an authentication procedure are referred to as the claimant and verifier. Legacy Authentication uses the challenge-response scheme in which the verifier checks the claimant's knowledge of a secret key in a 2-move protocol using symmetric secret keys. The LM will be the responsible entity processing the authentication preferences from the application layer to determine in which direction the authentication takes place. Legacy Authentication is used when either Legacy Pairing or Secure Simple Pairing with the P-192 Elliptic Curve has been used to generate the link key[15]. Figure

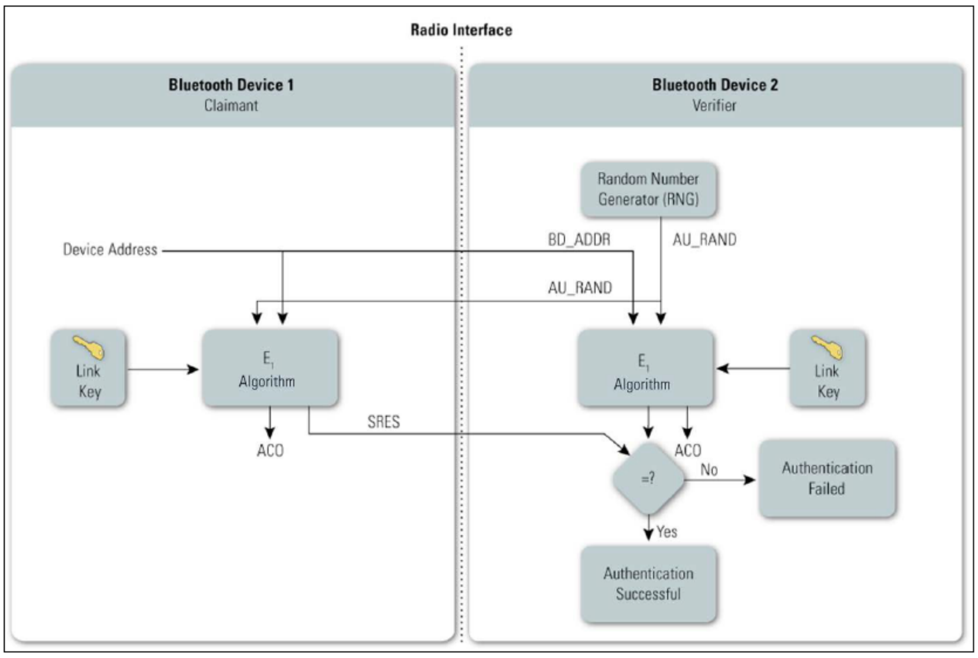


Figure 4.7: Bluetooth BR Legacy Authentication

4.7 summarize the Legacy Authentication scheme.
The steps in order to authenticate users in Legacy Authentication are:

1. The verifier generates and transmits a 128-bit random challenge (`AU_RAND`) to the claimant.
2. The claimant utilizes the E_1 algorithm (SAFER+ algorithm) to compute an authentication response (`SRES`) using his or her unique 48-bit Bluetooth device address (`BD_ADDR`), the link key (which has been shared between the devices in the pairing stage), and the `AU_RAND` (received from the verifier) as inputs.
3. The verifier performs the exact computation. As a result, only the 32-MS-bits of E_1 output are used for authentication purposes. The remaining 96-bit of the 128-bit output is known as the Authenticated Ciphering Offset (`ACO`) value, which will be incorporated later in the encryption key generation.
4. The claimant returns the 32 Most Significant bits of the E_1 output as the computed response, the Signed Response (`SRES`), to the verifier.
5. The verifier compares the `SRES` from the claimant with the value it computed.
6. If the two 32-bit values are equal, the authentication is considered **successful**. Otherwise, the authentication **fails**.

4.5.2 Secure Authentication

Secure Authentication is used when both devices support Secure Connections. Secure Authentication is mutual; hence, the challenge-response scheme will be in the way that both devices act as a verifier and claimant in the same sequence where the knowledge of a secret

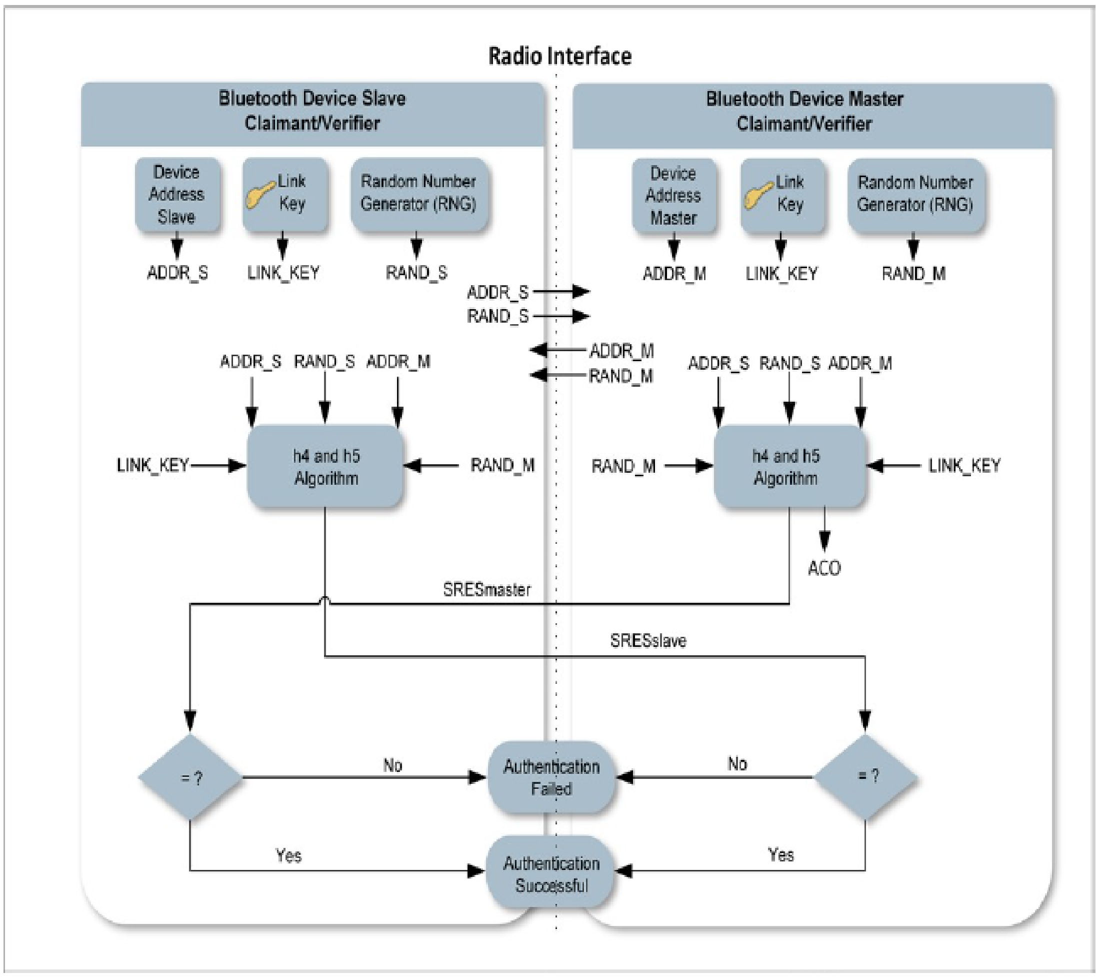


Figure 4.8: Bluetooth BR Secure Authentication[15]

key is checked through a 4-move protocol using symmetric secret keys. Secure Authentication is used when Secure Simple Pairing with the P-256 Elliptic Curve has been used to generate the link key. Since both devices are the claimant and verifier, for better understanding in the upcoming sections, we will refer to the authentication pair as a master and slave (in [16], the terms Central and Peripheral have been used instead of master and slave). Figure 4.8 summarize the Secure Authentication scheme.

Since the authentication is mutual in Secure Authentication, the connection pair have to authenticate each other; Therefore, the following steps will be generated on both sides. We will explain the case when the master initiates the authentication, and it is the same when the slave initiates the authentication, except that the first two steps are swapped.

1. The slave initiates the authentication by generating and transmitting a 128-bit random challenge (`RAND_S`) along with the Bluetooth device address (`ADDR_S`) to the master.

2. The master does the same by generating and transmitting a 128-bit random challenge `RAND_M` along with its Bluetooth device address (`ADDR_M`) back to the slave.

3. The pair will use h_4 and h_5 algorithms to to compute their authentication responses `SRES_S` and `SRES_M` using the following as inputs:

- the 48-bit Bluetooth device address of the master `ADDR_M`,
- the 48-bit Bluetooth device address of the slave, `ADDR_S`,
- the shared link key,
- the `RAND_M`,
- the `RAND_S`.

As a result, only the 32-Most Significant bits of h_5 output is used for authentication purposes. The remaining 96-bits of the 128-bit output are known as the Authenticated Ciphering Offset (`ACO`) value, which will be incorporated later in the encryption key generation.

4. The slave returns the 32-MS-bits of the h_5 output as its computed response `SRESslave` to the master.

5. After receiving `SRESslave`, the master returns the 32-MS-bits of his h_5 output as the computed response `SRESmaster`, back to the slave.

6. The master and slave now will compare the received SRES from each other with the value that they have computed.

7. If the two 32-bit values are equal on both sides, the authentication is considered **successful**. If the two 32-bit values are different on one or both sides, the authentication **fails**[15].

4.6 Confidentiality

In Section 4.2, we have seen the Security Modes of Operation for Bluetooth to provide pairing and authentication. Bluetooth provides Encryption to address the need for confidentiality by thwarting attackers and forgeries. Bluetooth offers three encryption modes as follows:

- Encryption Mode 1: No encryption at all.
- Encryption Mode 2: Individually addressed traffic is encrypted, while broadcast traffic is not encrypted.
- Encryption Mode 3: All traffic is encrypted.

Encryption

Encryption is the general application of cryptography. Using encryption, one can encode information so that an unauthorized third party in possession of the encoded data cannot decode it and access the original information [6]. Bluetooth protects users' information by encrypting the packet payload. Bluetooth payload Encryption is handled in one of two ways:

- A stream cipher encryption algorithm E_0
- AES-CCM encryption algorithm

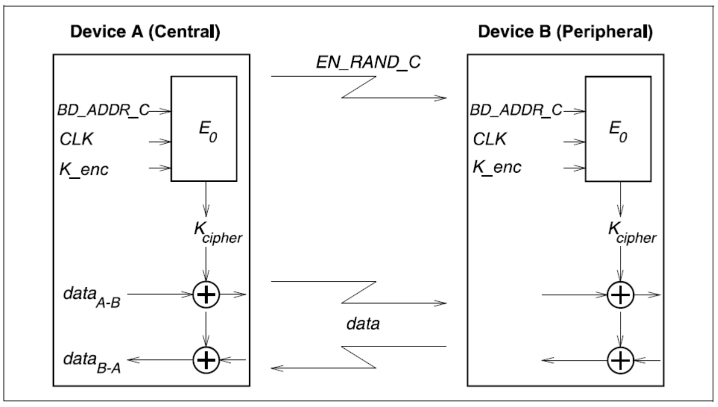


Figure 4.9: Bluetooth E0 Encryption Procedure

4.6.1 E0 Encryption Algorithm

E_0 is a stream cipher that generates a sequence of pseudo-random numbers and combines it with the payload data using the XOR operator. E_0 is used for Encryption when the device only offers Legacy Pairing or Secure Simple Pairing with ECDH P-192. The cipher is symmetric, meaning the decryption will be exactly as the Encryption using the same key. Figure 4.9 Summarize the Bluetooth E_0 Encryption procedure.

4.6.1.1 Encryption Concept

E_0 will cipher Bluetooth payload data. The payload is ciphered after the CRC bits have been appended and before the FEC encoding. E_0 uses four linear feedback shift registers ($LFSR_1, ..., LFSR_4$) whose output will be combined using a 16-states finite state machine (known as the summation combiner[20]). The output of the summation combiner is the keystream sequence. E_0 takes the following as its input:

- The Central's Bluetooth Device Address (BD_ADDR_C) (in previous sections, $Device_A$, $Device_1$ or the initiating device).
- The second input is 26-bits of the Central real-time clock (CLK): the clock will be incremented for each slot. Using CLK, at least one bit is changed between two transmissions. Thus, a new key stream is generated after each re-initialization.
- The encryption key K_{enc}: derived from the current link key, COF, and a random number EN_RAND_C. EN_RAND_C will be issued by the Central before entering encryption mode and transmitted in plain text. The key length used in Bluetooth is set to be 128-bits.

Within the E_0 algorithm, the encryption key K_enc is modified into another key denoted $K_{session}$ in which its size varies between 8 to 128-bit. For packets covering more than a single slot, the Bluetooth clock in the first slot will be used for the entire packet. E_0 perform the Encryption procedure in three phases as follow:

- Firstly, it performs the initialization phase, in which it combines the input bits in an appropriate order, giving an output known as session key ($K_{session}$).
- In the second phase, the output of the first phase will be fed into the four LFSRs, where the keystream bits will be generated (keystream generator).

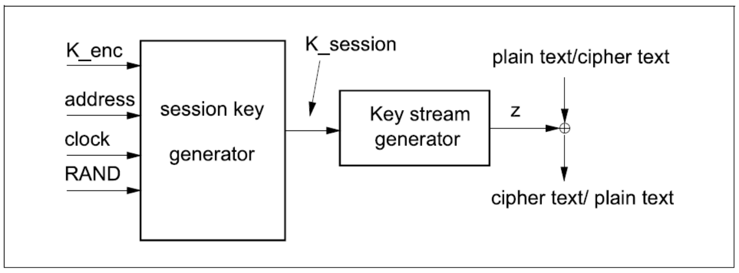


Figure 4.10: Stream ciphering for Bluetooth with E_0.

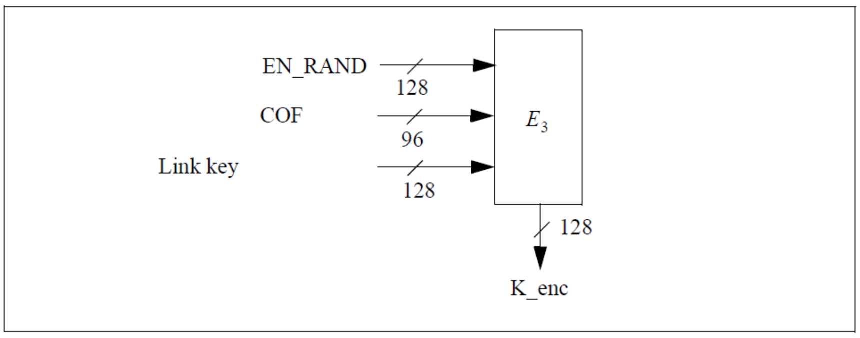


Figure 4.11: Generating the encryption key using E_3

- The third phase performs encryption/decryption by XORing the key stream bits with the plain text/cipher text. Figure 4.10 shows the detailed ciphering procedure with E_0.

4.6.1.2 Generating the encryption key

The encryption key K_{enc} is derived from the current link key using the algorithm E_3. The key length produced is 128-bits. However, before E_0, K_{enc} is shortened to the agreed encryption key length. The function is constructed as in Figure 4.11:

$$K_{enc} = E_3(EN\_RAND, LinkKey, COF) \tag{4.19}$$

COF value is set to one of two values:

- If the current link key is a temporary link key: $COF = BD\_ADDR \,||\, BD\_ADDR$.
- Otherwise $COF = ACO$.

4.6.2 AES-CCM Encryption Algorithm

AES-CCM is used for encryption in the case of Secure Simple Pairing with ECDH P-256. More details about the AES-CCM Encryption procedure are in Request for Comment (RFC) 5048 and 3610. Bluetooth AES-CCM encryption function takes the following as inputs:

- The encryption key k_{enc}
- The encryption nonce, which is one of two: either the payload counter format in the case of ACL packets or the clock format in the case of eSCO packets

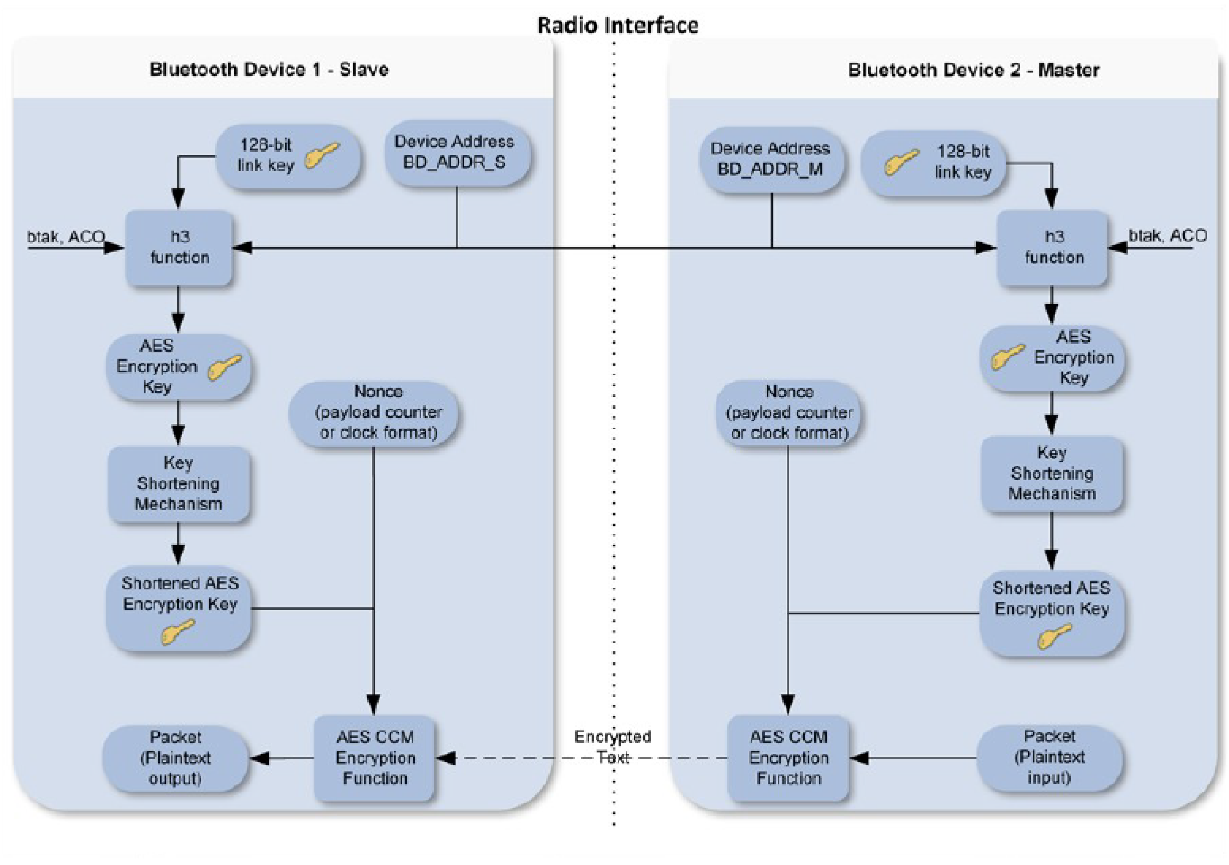


Figure 4.12: Bluetooth AES-CCM Encryption Procedure

- The payload bits

AES-CCM output will be the Encrypted text/Plain text. When AES-CCM encryption is enabled, ACL packets are included with a 4-Byte Message Integrity Check (MIC). while eSCO packets do not[15]. Figure 4.12 shows the procedure of Bluetooth encryption using AES-CCM.

4.6.2.1 AES encryption key generation

The key is created using the AES encryption key generation function h_3. the function (h_3) makes use of the Message Authentication Function HMAC-SHA256 which is denoted as $(\texttt{HMAC\_SHA\_256}_T)$ where T is 128-bit link key. The inputs to h_3 are the following:

- A_1, The BD_ADDR of the Central (Noted as Master in Figure 4.12)
- A_2, The BD_ADDR of the Peripheral (Noted as Slave in Figure 4.12)
- 64-bit of ACO (the output of h5 in Secure Authentication procedure)
- T, the 128-bit Bluetooth Link Key (derived from f2 in Secure Authentication)

The output of the h_3 function is:

$$AES\_K_{enc} = \texttt{HMAC\_SHA\_256}_T(Link\_key \parallel A_1 \parallel A_2 \parallel ACO) \tag{4.20}$$

The output is 256-bit, AES k_{enc} will be shortened, and only the most significant 128-bits will be considered the AES key.

Chapter 5

Bluetooth LE Security

BLE security concept is relatively similar to Bluetooth BR/EDR. However, the methods and operations are somehow different from Bluetooth BR/EDR security. In this chapter, we will go through the security and privacy features defined within the BLE specification.

5.1 BLE Security Overview

Version 4.0 of Bluetooth specification has seen for the first time the introduction of Advanced Encryption Standard–Counter with CBC-MAC (AES-CCM) encryption, which provides strong encryption. Up to Bluetooth v4.1, there was only one pairing mechanism: Low Energy Pairing. Version 4.2 was a security mutation for BLE, where BLE Pairing can be handled in two methods. The first is Legacy Pairing (referring to the old Low Energy Pairing), whereas the second method is a new introduction to BLE, known as Secure Connection. The newly added Secure Connection utilizes FIPS-approved algorithms (AES-CMAC and Elliptic curve P-256). Version 4.2 also added a feature of reusing/sharing the encryption keys generated by either BLE or Bluetooth BR/EDR if operating together in the same device. The BR/EDR link key can be derived from the BLE Long Term Key, and BLE Long Term Key can be derived from the BR/EDR Link key. Thus, the user pairing to one physical transport can be connected to the other automatically[[15]. The Security Manager Protocol (SMP) handles the BLE security procedure, setting and defining the algorithms and protocols for the main security procedure (e.g., generating and exchanging keys between BLE devices). BLE security is detailed in Bluetooth Core Specification, Vol 3, Part H, Sections 2 and 3 [16].

Similar to Bluetooth BR/EDR, BLE offers the main security services when requested, as discussed in Section 4.2. Unlike BR/EDR, BLE handles the authentication procedure as one of its stages and does not have an individual procedure. BLE Pairing is the main security procedure, where the device gets paired, bonded, and authenticated. The link encryption starts at the end of the Pairing procedure, but it does have a separate procedure. The Pairing is discussed in more detail in section 5.3.

5.2 BLE Security Modes

The security modes and levels refer to a combination of security attributes and requirements, where services and service requests of BLE will have their security requirements. A device will follow the mode and level that meets its security requirements. More robust security requirements prevail in the case of a service request and the associated service having different security requirements. BLE has two security modes:

- BLE Security Mode-1: has four different encryption levels as follows:
  - Level-1: No security (No authentication and no encryption)
  - Level-2: Unauthenticated pairing with encryption
  - Level-3:Authenticated pairing with encryption
  - Level-4: Authenticated Secure Connections pairing with encryption
- BLE Security Mode-2: is in two levels of data signing:
  - Level-1: Unauthenticated pairing with data signing
  - Level-2: Authenticated pairing with data signing

BLE security Mode-1 Level-4 is the most secure of all modes/levels since it offers Secure Connections authenticated pairing and encryption using AES-CMAC and Elliptic Curve P-256, for v4.2 or later devices[15].NIST recommends the use of Mode-1 Level-4 if possible. For version 4.1 devices and earlier versions, NIST recommends using Mode-1 Level-3. BLE security Mode-1 Level-1 is the least secure and should be avoided. Security Mode-2 does not provide encryption, and it's only meant with data signing.

5.3 Pairing

Pairing is the foundation of BLE security, where devices form a secure, trusted relationship. The devices are said to be bonded if they agree to store the shared keys at the Pairing procedure for future reuse. If not bonded, the devices have to go through the Pairing procedure every time they reconnect. Pairing is established to generate different keys, and one of those keys shall be used to encrypt the shared link. BLE Pairing is established in three phases as follows:

- **Phase-1**: Pairing Feature Exchange
- **Phase-2**:
  - Legacy Pairing: Short Term Key (STK) Generation
  - Secure Connections: Long Term Key (LTK) Generation
- **Phase-3**: Keys Distribution

BLE pairing Phase-1 and 3 are identical in both Legacy Pairing and Secure Connection. The only difference is in Phase-2. Figure 5.1 shows the different Pairing phases.

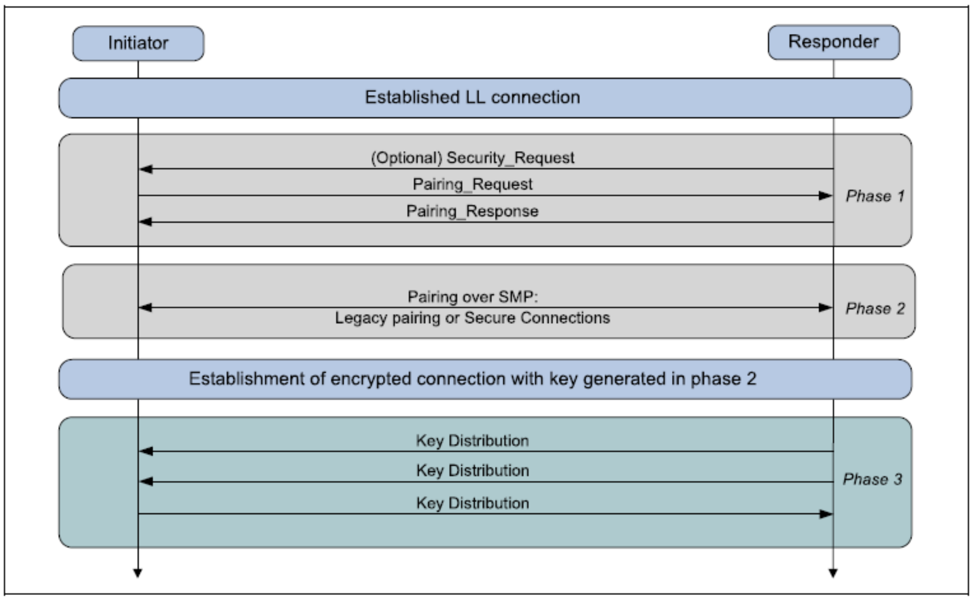
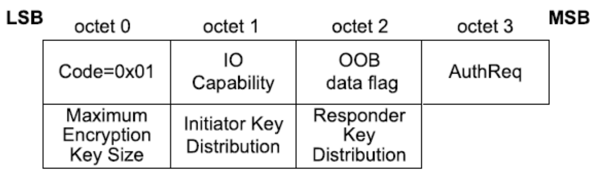

Initiator
Responder
Established LL connection
(Optional) Security_Request
Pairing_Request
Pairing_Response
Phase 1
Pairing over SMP:
Legacy pairing or Secure Connections
Phase 2
Establishment of encrypted connection with key generated in phase 2
Key Distribution
Key Distribution
Key Distribution
Phase 3

Figure 5.1: BLE Pairing Phases



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	

Figure 5.2: SMP Pairing Request/Response PDU

5.3.1 Phase-1

Phase-1 is the base for the BLE Pairing procedure. The procedure starts with the Initiating device sending an SMP Pairing Request to the Responder device; the Responder replies with an SMP Pairing Response. This step is called the Pairing Feature Exchange. The exchange provides the devices with the information needed to Decide/Determine:

- The Pairing method, whether Legacy Pairing or Secure Connections.
- The association model, whether Just Work, OOB, Passkey Entry, or Numeric Comparison (in Secure Connection Only).
- The device authentication and what form the authentication step should take.
- Key types to be generated and distributed.
- The encryption key length (Long Term Key).

Pairing Feature Exchange

The Initiator starts the Pairing Feature Exchange by sending a Pairing Request command to the responding device. Figure 5.2 shows the Pairing Request/Response fields.

- IO Capability: where the devices acknowledge their input and output capabilities as follows:

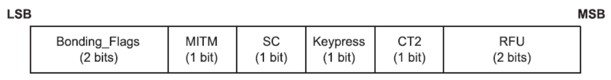


Figure 5.3: Authentication requirements flags

- **DisplayOnly**: The device can display numbers or text but cannot accept input.
- **KeyboardOnly**: The device can accept text or numeric input from the user.
- **DisplayYesNo**: The device allows the user to respond with YES or NO.
- **KeyboardDisplay**: The device has both keyboard and display.
- **NoInputNoOutput**: The device has neither input nor output capabilities.

More details about the IO Capability are found in Bluetooth Core Specification, Vol 3, Part H, Section 2.3.5.1 [16].

- OOB data flag: Defines whether or not the device is supporting OOB. If OOB is present, the flag is set to 1; otherwise, it is 0. In legacy Pairing, the two devices should support OOB to use this feature. In Secure Connection, if only one of the pairs has set the flag to 1, then OOB can be used.
- Responder/Initiator Key Distribution: The field indicates the requested keys by Responder/Initiator from the other pair to be generated and distributed in Phase-3.
- Maximum key size: This field defines the maximum encryption key size that the device can support, which is between 7 to 16-Bytes.
- AuthReq: Contains the requested security properties and is composed of the following sub-fields:
  - Bonding Flag: Used to indicate whether the device wishes to bond (i.e., storing the keys for later use.) or not.
  - MITM: Is set to 1 if the user requests Man In The Middle protection. If one or both devices set the flag to 1, MitM protection will be provided by any association model except Just Work.
  - SC: Is the most important field, where setting the flag to 1 in both devices means that the pair will use Secure Connection as the Pairing method. Otherwise, it is Legacy Pairing.

Figure 5.3 shows how AuthReq flag is composed.

5.3.2 Phase-2

Phase-2 proceed based on the information provided in Phase-1 Feature Exchange. As mentioned previously, Phase-2 differs on which method is selected for the Pairing: Legacy Pairing or Secure Connection.

Association Model	TK Value
Just Work	128-bits of Zeros.
Passkey Entry	The 6-digit passkey is 20-bits, will be padded with leading zeroes to give a 128-bits.
OOB	128-bit value is generated and exchanged Out-Of-Band between the devices.

Table 5.1: TK value based on the association model.

5.3.2.1 Phase-2 Legacy Pairing

Legacy Pairing Phase-2 is concerned with two main points:

1. To generate a key known as Short Term Key (STK), which will be used in link encryption to securely distribute other encryption materials in Phase-3.
2. To offer protection against MitM attacks by authentication.

Legacy Connection is selected if one or both devices have set the SC (Secure Connections) flag equal to zero. Phase-2 proceeds in four steps:

Step-1: Generating a Temporary Key (TK)

The Temporary Key (TK) is a 128-bit that will be used to derive the Short Term Key (STK). The method to generate TK depends on the selected association model in Phase-1 based on both devices' IO capabilities at the Feature Exchange. The association models have been discussed in Section 4.3.2

- **Just Work**: Will be selected if:
  - Both devices have set the MITM (Man In The Middle) flag to 0
  - Or, Both devices set the OOB Data Flag to 0. Furthermore, one or both devices have set the MITM flag to 1. Moreover, IO capabilities indicate that passkey entry cannot be supported.
- **Passkey Entry**: Will be selected if:
  - Both devices set the OOB Data Flag to 0. Furthermore, one or both devices have set the MITM flag to 1. Moreover, IO capabilities indicate that passkey entry can be supported.
- **OOB**: Will be selected if both devices have set the OOB Data Flag to 1.

TK value is different in each of the association models. Thus, each device will calculate TK as explained in Table 5.1, and the value is never shared over the air.

Step-2: Authentication

Up to the previous point, both devices have the same Temporary Key (TK). TK will be part of a new value that both devices will use to authenticate each other. The value is 128-bits, known as confirm value (LP_CONFIRM) generated using the c_1 algorithm. See Bluetooth Core Specification, Vol 3, Part H 2.2.3 for details of s_1 [16]. The input to the c_1 algorithm is as follows:

- k: 128-bits TK value
- r: 128-bits random number (LP_RAND$_I$ and LP_RAND$_R$) generated by both devices
- preq: 56-bits Pairing Request command
- pres: 56-bits Pairing Response command
- iat: 1-bit initiating device address type
- rat: 1-bit responding device address type
- ia: 48-bits initiating device address
- ra: 48-bits responding device address
- padding: 32- bits of zeros

The initiating device calculates the 128-bit confirm value (LP_CONFIRM$_I$) as follows:

$$LP\_CONFIRM_I = c_1(k, r_I, prq, pres, iat, rat, ia, ra) \tag{5.1}$$

While the Responder device will generate the 128-bits confirm value (LP_CONFIRM$_R$) as follows:

$$LP\_CONFIRM_R = c_1(k, r_R, prq, pres, iat, rat, ia, ra) \tag{5.2}$$

After both devices have derived their confirmation value, they will start the Authentication procedure by exchanging the values as follows:

1. The initiating device transmits LP_CONFIRM$_I$ to the responding device.
2. Upon reception, the responding device transmits LP_CONFIRM$_R$ to the initiating device.
3. When the initiating device receives LP_CONFIRM$_R$, it transmits LP_RAND$_I$ to the responding device.
4. The responding device verifies the LP_CONFIRM$_I$ by repeating the process, using the received LP_RAND$_I$.
5. If the values do not match, the pairing process shall be aborted.
6. If the values match. The responding device **authenticated** the initiating device. Upon authentication, the responding device transmits LP_RAND$_R$ to the initiating device.

7. The Initiating re-calculates LP_CONFIRM$_R$ value and compares it with the value it previously received. If they match, then the initiating device has **authenticated** the responding device.

Authentication has been achieved, and both devices have proved their knowledge of the TK. This stage ends here, and the devices will proceed to the next step, where they will generate the STK.

Step-3: Generating a Short Term Key (STK)

The STK is a 128-bits value to be used in encrypting the link. STK will be generated using the key generation function s_1. See Bluetooth Core Specification, Vol 3, Part H 2.2.4 for details of s_1 [16]. s_1 takes the following input:

- k: 128-bits TK value
- r_1: 128-bits random number (LP_RAND$_R$) generated by the responding device
- r_1: 128-bits random number (LP_RAND$_I$) generated by the initiating device

$$STK = s_1(k, r_1, r_2) \tag{5.3}$$

Step-4: Encrypting the link

The last step of Legacy Pairing Phase-2 is that the initiating device uses STK to encrypt the link. After link encryption, the pair will derive a session key as in EQ 5.18 in Section 5.4.1.1, to be used in the following sessions as the link key. From this point, all the packets are encrypted at the BLE Link Layer.

5.3.2.2 Phase-2 Secure Connection

Secure Connection (SC) is the alternative method in BLE Pairing. As the name suggests, Secure Connection is more secure than Legacy pairing because it added the Elliptic Curve Diffie-Hellman (ECDH) public key cryptography. Secure Connection Phase-2 is concerned with the following:

- To generate a key known as the Long Term Key (LTK), which will be used to encrypt the link for secure key distribution in Phase-3 and forthcoming sessions.
- To provide protection against MitM attacks by offering two steps authentication.

Secure Connection is the Pairing method if both devices set the SC (Secure Connections) flag to 1. Secure Connections Phase-2 breaks down into steps:

Step-1: Public Key Exchange

- Each device generates its own Elliptic Curve Diffie-Hellman (ECDH) public-private key pair. Public keys are denoted (Pk), and Private/Secret keys are denoted (SK).
- The initiating device ($Device_A$) sends its public key (PK_a) to the responding device ($Device_B$).

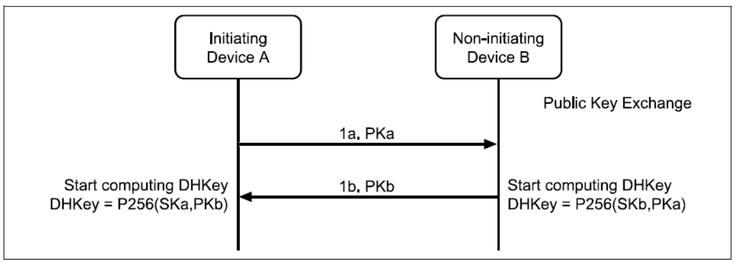


Figure 5.4: BLE Public key exchange

- The responding device replies with its public key (PK_b). The two keys should not be identical. Otherwise, the process fails.
- Both devices validate the received public keys on the right curve (P-256). An invalid check leads to a process failure.
- After the exchange, both devices generate a DHkey where each device uses its private key (SK) and the public key (PK) of the other device.

$$DHkey_a = P256(SK_a, PK_b) \tag{5.4}$$

$$DHkey_b = P256(SK_b, PK_a) \tag{5.5}$$

Figure 5.4 shows the procedure have been followed in the public key exchange.

Step-2: Authentication Stage-1

Authentication in Secure Connection Pairing is one of its strengths. Thus, it is being established in a long and complicated procedure, making it more difficult for the attackers to compromise the shared information. Authentication stage-1 proceeds according to the selected association model in Phase-1 Feature Exchange.

Just Works, and Numeric Comparison: Both Just Works and Numeric Comparison share the same protocol. Numeric Comparison is considered the most secure among Secure Connection and Legacy Pairing association models. Just work will be selected as described in Section 5.3.2.1 (Just Work). While Numeric Comparison will be selected if one or both devices set the MITM flag to 1, OOB is not supported by both devices, and both devices have indicated in their IO capabilities DisplayYesNo. The authentication procedure goes as follows:

- After a successful public key exchange, both device generates a random 128-bit nonce (N_a and N_b). The values should be fresh with each instantiation of the pairing procedure.
- The responding device ($Device_B$) computes a commitment value (C_b) using a function named f_1. See Bluetooth Core Specification, Vol 2, Part H 7.7.1 for details of f_1 [16].

$$C_b = f_1(PK_b, PK_a, N_b, 0) \tag{5.6}$$

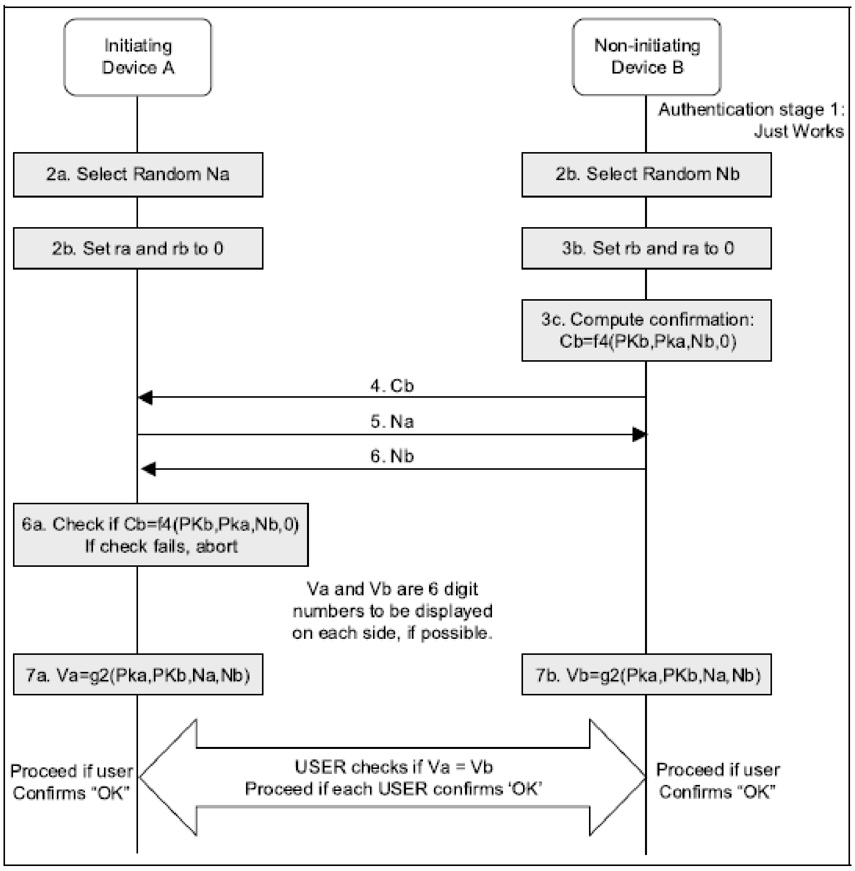

Initiating Device A
Non-initiating Device B
Authentication stage 1: Just Works
2a. Select Random Na
2b. Select Random Nb
2b. Set ra and rb to 0
3b. Set rb and ra to 0
3c. Compute confirmation: Cb=f4(PKb,Pka,Nb,0)
4. Cb
5. Na
6. Nb
6a. Check if Cb=f4(PKb,Pka,Nb,0) If check fails, abort
Va and Vb are 6 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"

Figure 5.5: BLE Authentication stage-1: Numeric Comparison

- $(Device_B)$ transmits C_b to the initiating device $(Device_A)$. The commitment prevents an attacker from altering any value at a later time.
- $Device_A$ and $Device_B$ exchange their nonce values (N_a and N_b).
- $Device_A$ confirms the commitment by recomputing C_b. A failure at this point causes the procedure to abort.
- A successful check at $Device_A$ results in both devices computing a 6-digit confirmation value (V_a and V_b) displayed to the user on both devices. The value is generated using the Bluetooth g_2 function. See Bluetooth Core Specification, Vol 3, Part H 2.2.9 for details of f_1 [16]. The user checks and confirms if the values match in both devices. If not, the procedure is aborted.

$$V_a = g_2(PK_a, PK_b, N_a, N_b) \tag{5.7}$$

$$V_b = g_2(PK_a, PK_b, N_a, N_b) \tag{5.8}$$

- When Just Works, the checks are not performed, and the user is never shown the 6-digit values.

Figure 5.5 depicts the details of the Just Work and Numeric Comparison association models. After both devices are authenticated, the procedure continues at Authentication Stage-2.

Passkey Entry: In Passkey Entry, The user can be in one of two situations: Either the user enters an identical Passkey into both devices. Alternatively, the Passkey is generated and displayed on one device, and the user enters it into the other device. Passkey Entry will be selected as described in Section 5.3.2.1 (Passkey Entry).
The 6-digits (20-bits) Passkey will be the basis of the authentication of the devices. The procedure goes as follows:

- The user enters an identical Passkey into both devices. A 6-digit Passkey is 20-bits long represented with r_a for $Device_A$ which is the concatenation of the 20-bits r_{a_i}. For $Device_B$ it's r_b and it's the concatenation of the 20-bits of r_{b_i}.
- Then, both devices generate a random 128-bits nonce (N_{a_i} and N_{b_i}).
- Each device compute a commitment value (C_{a_i} and C_{b_i}) and exchange it with each other. The commitment value is generated using the f_4 function. f_4 uses the AES-CMAC algorithm with the nonce as its 128-bit key, where the input is the concatenation of the other function arguments (PK_a,PK_b, and r_{a_i}/r_{b_i}).

$$C_{a_i} = f_4(PK_a, PK_b, N_{a_i}, r_{a_i}) \tag{5.9}$$

$$C_{b_i} = f_4(PK_b, PK_a, N_{b_i}, r_{b_i}) \tag{5.10}$$

- $Device_A$ sends it's N_{a_i} to $Device_B$. Upon reception of $Device_B$ checks C_{a_i} and if check succeed $Device_B$ replies with it's N_{b_i}.
- As a final step $Device_A$ checks the received C_{b_i}. If the check fails, the procedure is aborted.
- The steps are repeated 20 times as the Passkey is 20-bits. At the end of this stage, N_a is set to $N_{a_{20}}$ and N_b is set to $N_{b_{20}}$, which will be used in Authentication stage-2.

The iterative procedure used in generating the commitment value in Passkey Authentication is known as Gradual Disclosure, where an MitM attacker cannot receive more than 1-bit during each exchange and has to guess the remaining bits, making it more difficult in practical terms for the attackers. Figure 5.6 shows the sequence diagram for Passkey Entry Authentication stage-1.

OOB: The Out-of-Band protocol is used when one device sets the OOB Data Flag to 1. In OOB, the discovery of the peer device is carried out in-band, and authentication parameters are transmitted through the OOB interface. The procedure is detailed below:

- Each device generates a random number, (r_a and r_b), and set the other pair random number equal to zero ($r_b = 0$ in $Device_A$ and $r_a = 0$ in $Device_B$).
- They each calculate a commitment value (C_a and C_b) as follow:

$$C_a = f_4(PK_a, PK_a, r_a, 0) \tag{5.11}$$

$$C_b = f_4(PK_b, PK_b, r_b, 0) \tag{5.12}$$

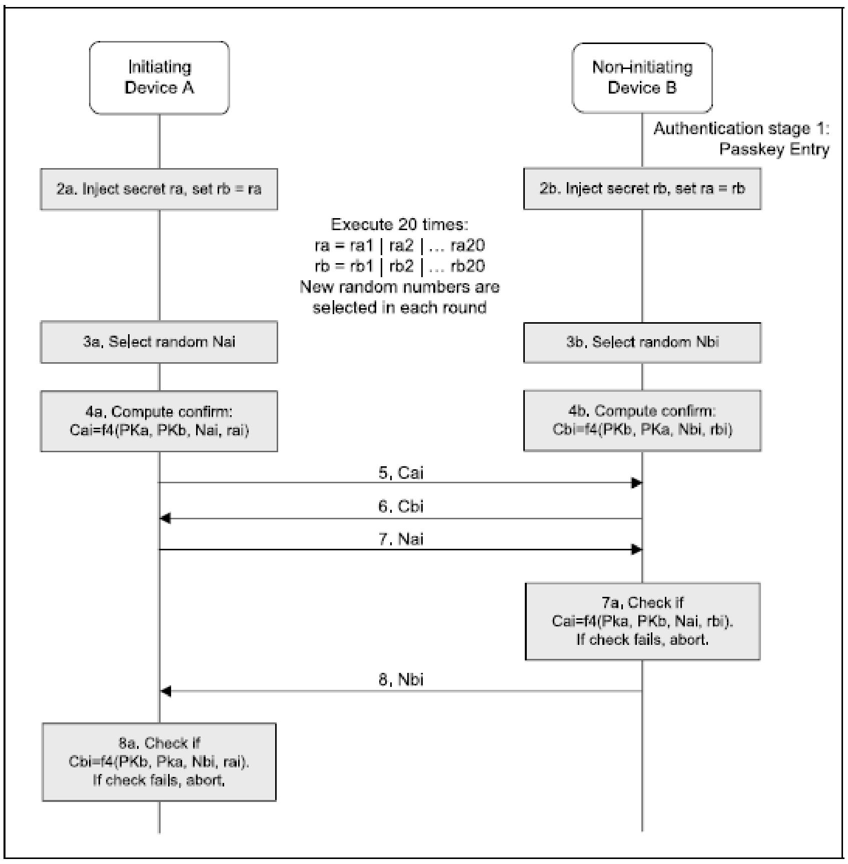

Initiating Device A
Non-initiating Device B
Authentication stage 1: Passkey Entry
2a. Inject secret ra, set rb = ra
2b. Inject secret rb, set ra = rb
Execute 20 times: ra = ra1 | ra2 | ... ra20 rb = rb1 | rb2 | ... rb20 New random numbers are selected in each round
3a. Select random Nai
3b. Select random Nbi
4a. Compute confirm: Cai=f4(PKa, PKb, Nai, rai)
4b. Compute confirm: Cbi=f4(PKb, PKa, Nbi, rbi)
5. Cai
6. Cbi
7. Nai
7a. Check if Cai=f4(Pka, PKb, Nai, rbi). If check fails, abort.
8. Nbi
8a. Check if Cbi=f4(PKb, Pka, Nbi, rai). If check fails, abort.

Figure 5.6: BLE Authentication stage-1: Passkey Entry

- $Device_A$ now sends its Bluetooth device address `(A)` (to be used with the in-band part) together with r_a and C_a to $Device_B$ through OOB.
- If $Device_B$ supports OOB, it sends its corresponding data values (Bluetooth device address `(B)`, C_b, and r_b) back to $Device_A$.
- In this step, both devices communicate using the `In-Band` and share their public keys (PK_a, and PK_b) in public.
- Both devices will use the received values `Out-of-Band` and `In-Band`. $Device_A$ recomputes C_b, and $Device_B$ recompute C_a as in EQ 5.11 and 5.12. If the values are equal to what they have received, they authenticate each other; otherwise, the process is aborted. Successful authentication results in both devices generating new random numbers (N_a and N_b) and exchanging them to be used in later procedures.

Figure 5.7 shows the sequence diagram for OOB Authentication stage-1.

Step-3: Authentication Stage-2

In Authentication Stage-2, more checks will be carried out to ensure the exchange of the public keys and other values from earlier steps. Furthermore, two important keys will be generated in this stage: The Long Term Key (LTK) and another key known as the *Mackey*. This stage is identical in all the association models. The procedure is shown in Figure 5.8 and illustrated below:

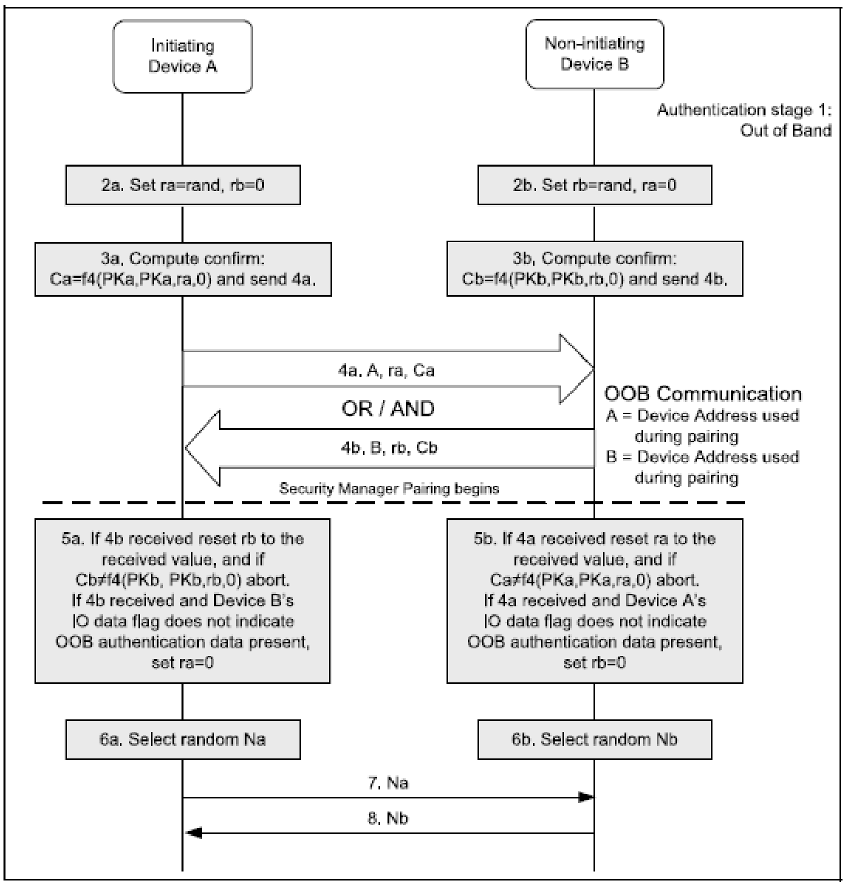

Initiating Device A
Non-initiating Device B
Authentication stage 1: Out of Band
2a. Set ra=rand, rb=0
2b. Set rb=rand, ra=0
3a. Compute confirm: Ca=f4(PKa,PKa,ra,0) and send 4a.
3b. Compute confirm: Cb=f4(PKb,PKb,rb,0) and send 4b.
4a. A, ra, Ca
OR / AND
4b. B, rb, Cb
OOB Communication
A = Device Address used during pairing
B = Device Address used during pairing
Security Manager Pairing begins
5a. If 4b received reset rb to the received value, and if Cb≠f4(PKb, PKb,rb,0) abort. If 4b received and Device B's IO data flag does not indicate OOB authentication data present, set ra=0
5b. If 4a received reset ra to the received value, and if Ca≠f4(PKa,PKa,ra,0) abort. If 4a received and Device A's IO data flag does not indicate OOB authentication data present, set rb=0
6a. Select random Na
6b. Select random Nb
7. Na
8. Nb

Figure 5.7: BLE Authentication stage-1: Out Of Band

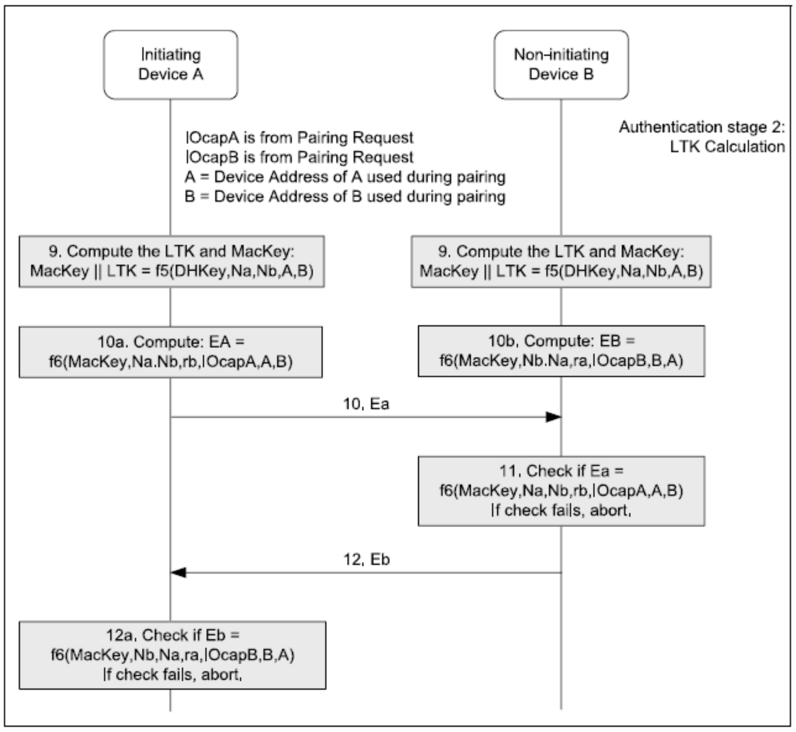

Initiating Device A
Non-initiating Device B
Authentication stage 2: LTK Calculation
IOcapA is from Pairing Request
IOcapB is from Pairing Request
A = Device Address of A used during pairing
B = Device Address of B used during pairing
9. Compute the LTK and MacKey: MacKey || LTK = f5(DHKey,Na,Nb,A,B)
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.

Figure 5.8: BLE Authentication stage-2 and LTK calculation

- The procedure starts with both devices utilizing the f_5 function to generate the exact value of LTK and $Mackey$ as in EQ 5.13. f_5 uses the AES-CMAC algorithm with DHKey as its 128-bit key.

$$MacKey \parallel LTK = f_5(DHkey, N_a, N_b, A, B) \tag{5.13}$$

- Then, each device uses the f_6 function to compute a new confirmation value (E_a and E_b) that includes the previously exchanged values and the newly derived key (i.e., $Mackey$). f_6 uses the AES-CMAC algorithm with $MacKey$ as its 128-bit key.

$$E_a = f_6(MacKey, N_a, N_b, r_b, IOcapA, A, B) \tag{5.14}$$

$$E_b = f_6(MacKey, N_b, N_a, r_a, IOcapB, B, A) \tag{5.15}$$

- $Device_A$ transmits E_a to $Device_b$ where it is checked. If the check fails, pairing is aborted.
- $Device_b$ then transmits its confirmation value E_b, which $Device_A$ checks. If the check fails, pairing is aborted.

As a final step in Authentication Stage-2, the initiating device will start encrypting the link using the lately derived LTK. Later, in Phase-3, LTK will contribute to creating a session key to be used in the upcoming sessions.

5.3.3 Phase-3

Phase-3 is known as the key distribution phase. Similar to Phase-1, Phase-3 is identical in both Legacy Pairing and Secure Connection and is considered an optional phase (only if the pair wish to bond). The keys to being distributed in the two pairing methods might not be the same. Phase-3 will proceed on an encrypted link, as the link has been encrypted in Phase-2 using either STK for Legacy Pairing or LTK for Secure Connection.

5.3.3.1 Legacy Pairing

In Legacy Pairing, the Central and the Peripheral will exchange the following keys:

- LTK, EDIV, and Rand
  - **LTK:** Long Term Key is a 128-bits used to generate the session key for session encryption.
  - **EDIV:** Encrypted Diversifier is a 16-bits key used as an index to identify the LTK when stored in the database. It's only used in Legacy Pairing. EDIV is generated each time an LTK is distributed and is transmitted in clear text.
  - **Rand:** Random Number is a 64-bits key used to identify the LTK. It's only used in Legacy Pairing. Rand is generated each time an LTK is distributed and is transmitted in clear text.
- IRK: Identity Resolving Key is a 128-bit key used to generate and resolve random addresses.

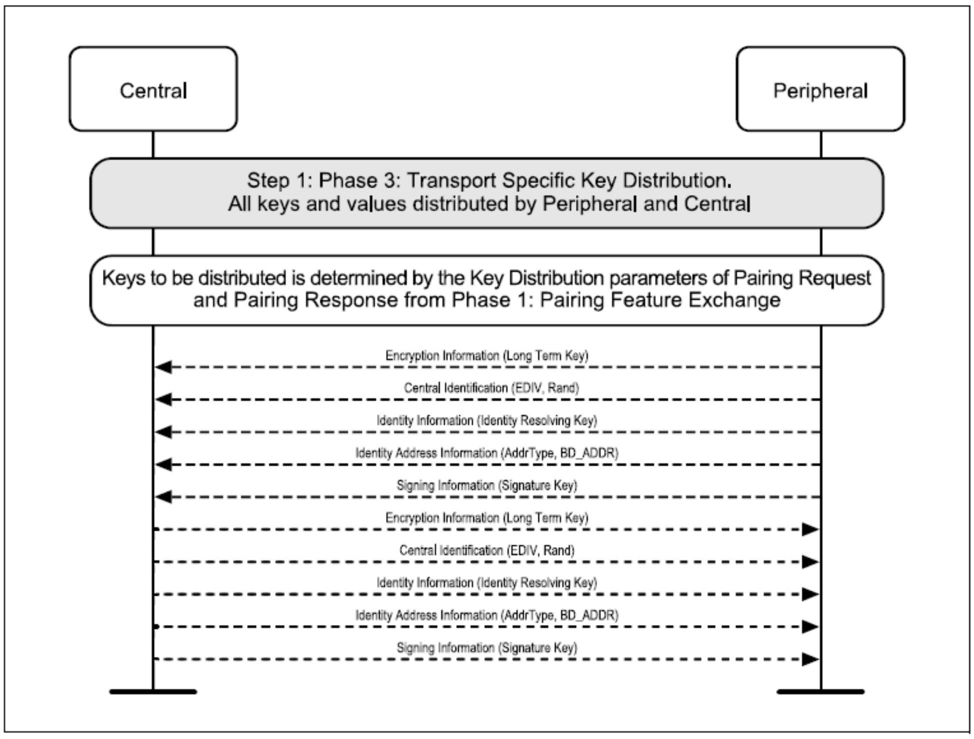

Central
Peripheral
Step 1: Phase 3: Transport Specific Key Distribution.
All keys and values distributed by Peripheral and Central
Keys to be distributed is determined by the Key Distribution parameters of Pairing Request and Pairing Response from Phase 1: Pairing Feature Exchange
Encryption Information (Long Term Key)
Central Identification (EDIV, Rand)
Identity Information (Identity Resolving Key)
Identity Address Information (AddrType, BD_ADDR)
Signing Information (Signature Key)
Encryption Information (Long Term Key)
Central Identification (EDIV, Rand)
Identity Information (Identity Resolving Key)
Identity Address Information (AddrType, BD_ADDR)
Signing Information (Signature Key)

Figure 5.9: BLE Phase-3 key distribution

- CSRK: is a 128-bit key used to sign data and verify signatures on the receiving device.

Once the exchange is successful, the distributed keys will be linked to the Bluetooth Device Address.

5.3.3.2 Secure Connection

Secure Connection exchanged keys are different from Legacy Pairing. The LTK will not be distributed this time; once it is generated, it will be stored locally by both devices. After that, the link is encrypted using an encryption key derived from the LTK. Thus, EDIV and Rand values will be set to zero, and the Bluetooth Device Address will identify the LTK. The Central and the Peripheral will only exchange **IRK** and **CSRK**.

In both cases (Legacy Pairing and Secure Connection), the keys are only distributed if requested in the “Initiator/Responder Key Distribution” field. Figure 5.9 summarizes the keys distribution procedure.

5.4 Confidentiality

BLE devices exchange data with each other in the form of attributes. This data should not be accessible by unauthorized entities, especially if the data is sensitive (e.g., about a user’s health or lifestyle, or relating to security or safety). So far, we have seen BLE Pairing, a long and complicated procedure. The LTK has been generated or exchanged in Pairing phase-3, allowing the user to activate the link encryption using this key. Unlike Pairing, BLE encryption is a Link Layer task.

5.4.1 Encryption

Even though BR/EDR uses AES-CCM as the encryption algorithm in the case of Secure Simple Pairing with ECDH P-256, the E_0 remains the main encryption algorithm. In contrast, BLE uses AES-CCM with a 128-bit key for all encryption operations. The encryption procedure is initiated after connection establishment at Pairing Phase-2. The Central Host will request its Controller to start the encryption on the link. The Peripheral will trigger the encryption in one of the following possibilities:

- Sending an SMP Security Request to the Central.
- Attempting to access an attribute with encryption security requirements.

The encryption is enabled by the Link Layer through the Encryption Start Procedure, as detailed in the following section.

5.4.1.1 Encryption Start Procedure.

The procedure starts with both the Central and Peripheral exchanging two parameters known as:

- IV: Initialization Vector
- SKD: Security Key Diversifier

IV and SKD will be exchanged in LL_ENC_REQ and LL_ENC_RES PDUs. If the bonding flag was set to 1 in Pairing Feature Exchange, EDIV and Rand values would also be exchanged as an identifier for the LTK. Following the parameters exchange, The Peripheral Host will notify the Link Layer to prepare for the link encryption. The encryption procedure will be finalized in a three-way handshake using LL_START_ENC_REQ and LL_START_ENC_RSP PDUs.The procedure goes as follows:

1. The Central generates its part of IV (IV_C a 32-bits random number) and SKD (SKD_C a 64-bits random number).
2. The Central transmit LL_ENC_REQ contains IV_C, SKD_C, EDIV, and Rand to the Peripheral.
3. Upon reception, the Peripheral generates its part of IV (IV_P a 32-bits random number) and SKD (SKD_P a 64-bits random number).
4. The Peripheral transmit LL_ENC_RES contains both IV_P and SKD_P to the Central.
5. Each device's Link Layer will generate the following:

$$SKD = SKD_P \parallel SKD_C \tag{5.16}$$

$$IV = IV_P \parallel IV_C \tag{5.17}$$

6. One or both devices have derived the Long Term Key in Pairing Phase-3. Thus, the LTK will be passed from the Host to the Link Layer, where the session key will be generated as follow:

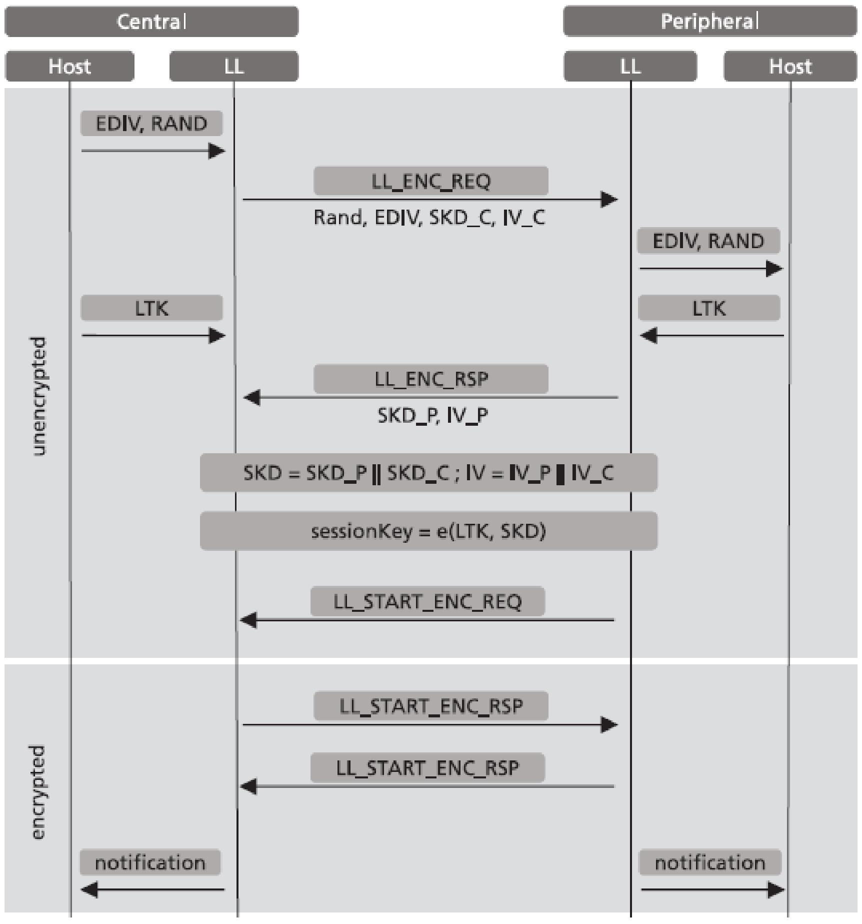


Figure 5.10: BLE Encryption Procedure.

- The Link Layer will utilize the Bluetooth security function e, which uses the AES-128 encryption algorithm to generate the 128-bits encrypted data from the 128-bits key and 128-bits plaintext.
- The input key will be the LTK/STK, and SKD will be used as the plaintext.

$$K_{session} = e(LTK/STK, SKD) \tag{5.18}$$

7. Three-way hand-shake starts from this point. The Peripheral transmits an unencrypted LL_START_ENC_REQ.

8. Upon reception, the Central sends an encrypted LL_START_ENC_RES.

9. The Peripheral then responds with an encrypted LL_START_ENC_RES.

10. When the central receives the last LL_START_ENC_RES, the connection will be encrypted. Figure 5.10 shows the sequence diagram of the BLE encryption procedure [18].

Encryption is only applied to data when in a connection. Connectionless sessions do not get encrypted. Once the link is encrypted, all the PDUs are appended by Message Integrity Code (MIC). The receiving device will decrypt the MIC and verify the message's genuinity. Figure 5.11 shows (in Red color) the encrypted part of BLE packet[6].

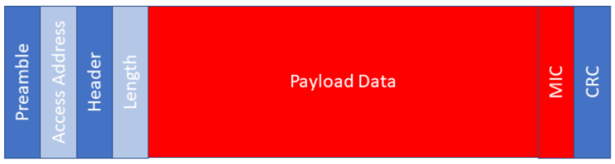


Figure 5.11: BLE Encrypted PDU

5.5 Cross-Transport Key Derivation (CTKD)

We have seen in both Chapters [4 and 5] the Pairing procedure ends with a key (i.e., Link Key or LTK) that is used in the encryption procedure. Those keys can be derived from each other. A device supports Bluetooth BR/EDR and BLE, and both support secure connection on a transport; the device can generate the encryption key for the other transport. There are two scenarios: either the Pairing procedure have been initiated in the BLE, then the BR/EDR link key will be derived from LTK, or vice versa. Both scenarios are discussed in the following sections.

5.5.1 BR/EDR link key from BLE LTK

BR/EDR Link Key is derived from the LTK in two steps as follows:

1. First, generating a new value known as the Intermediate Link Key (ILK), by utilizing the h_7 function (AES-CMAC-128), by providing the LTK as the key and an extended ASCII key identifier (keyID) of “tmp1” as the input.

$$ILK = h_7(LTK, \text{“}tmp1\text{”}) \tag{5.19}$$

2. The Bluetooth Link Key is derived using the h_6 function (AES-CMAC-128), using the ILK as the key and keyID of “lebr” as input.

$$K_{Link} = h_6(ILK, \text{“}lebr\text{”}) \tag{5.20}$$

5.5.2 BLE LTK from BR/EDR link key

LTK is derived from BR/EDR Link Key as follows:

1. A new value known as the Intermediate Long Term Key ($ILTK$) is computed using the h_7 function (AES-CMAC-128) by providing the BR/EDR link key as the key and the extended ASCII key identifier (keyID) of “tmp2”.

$$ILTK = h_7(K_{Link}, \text{“}tmp2\text{”}) \tag{5.21}$$

2. BLE LTK will be derived through the h_6 function (AES-CMAC-128), giving the $ILTK$ and keyID of “brle” as inputs.

$$LTK = h_6(ILTK, \text{“}brle\text{”}) \tag{5.22}$$

5.6 Privacy

In telecommunications and security, privacy ensures that the users and their corresponding devices cannot be tracked by unauthorized parties [6]. The Bluetooth device's identity is BD_ADDR, and it appears over the air in several scenarios.

- when scanning: in the `ScanA` field.
- when advertising: in the `AdvA` field.
- when it is a directed advertisement: in the `TargetA` field.

Unlike Bluetooth BR/EDR, BLE has a unique feature (i.e., Private Address), as we mentioned in Chapter 3, Section 3.2.2.1, that helps to form a periodically changing address, which will help protect the users' privacy and mislead the attackers. The privacy feature allows the Bluetooth device to have two types of addresses. One is the identity address (i.e., BD_ADDR), a static address. The new one is the private device address, which changes periodically. When Bluetooth activates, the private address is said to be in privacy mode. The Link Layer is the entity controlling the address change. Thus, there is a timer between 1 second and 11.5 hours. The link layer generates a new private address when the timer expires. The Bluetooth core specification recommends 15 minutes for the timer. The private address is either resolvable or non-resolvable. A resolvable address is used only if the connection pair decides to distribute the keys in the pairing procedure. IRK will be used to create and resolve the private address. Other devices will see a train of different BD_ADDR over different periods. In the non-resolvable case, no one (even a paired device) can determine the device's identity.

Chapter 6

Vulnerabilities and Attacks

In previous chapters, we have seen all about Bluetooth BR/EDR and Bluetooth LE security. Despite all these measures, Bluetooth has been vulnerable to many attacks. This chapter describes the most common vulnerabilities and attacks against Bluetooth BR/EDR and Bluetooth LE.

6.1 Vulnerabilities

In this section, we will summarize the major Bluetooth's vulnerabilities and threats. The work in this section is referred to the National Institute of Standards and Technology "NIST" [15]. The following sections will overview some vulnerabilities in the latest Bluetooth versions related to pairing, authentication, key, and encryption in Bluetooth BR/EDR and BLE.

6.1.1 BR/EDR

1. **Devices with Security Mode-1 do not initiate any security mechanisms**; thus, it is recommended to use Security Mode-3 for Bluetooth v2.0 devices and earlier versions.

2. **The use of short PINs in v2.0 and earlier versions**. Short PINs can be guessed easily.

3. **The encryption keystream is repeated every 23.3 hours in Bluetooth v2.0 and earlier versions**. One of the parameters used as input to the key-stream generator is the master's clock (CLK). If a connection lasts more than 23.3 hours, the CLK value will be repeated, resulting in a series of repeating keystreams, allowing the attacker to guess the plaintext.

4. **Just Works does not offer MitM protection during pairing**, Resulting in an unauthenticated link key. It's highly recommended to reject Just Work pairing requests.

5. **The use of weak and static ECDH key pairs**. Weak keys downgrade eavesdropping protection in SSP, allowing attackers to guess the link keys. Devices are urged to generate strong ECDH key pairs changing periodically.

6. **Static Passkeys are prone to MitM attacks**. MitM protection during pairing can be provided using fresh and random Passkeys for each pairing procedure.

7. **Security Mode-4 devices (v2.1 or later) are backward compatible with other security modes when connecting to devices do not support Security Mode-4 (v2.0 and earlier versions)**. Falling back to Security mode-1, the device would be stripped of protection. It's recommended to only fall back to Security mode-3; otherwise, never.

8. **The secret key is shared amongst all Piconet devices.** High risk of impersonation attacks when sharing the secret keys amongst more than two devices.

9. **BR/EDR encryption algorithm E_0 is relatively weak**. It is recommended to use FIPS-approved algorithms, as in the case of SSP with ECDH P-256; AES-CCM is used as the encryption algorithm.

10. **BR/EDR device privacy is compromised if BD_ADDR is captured.** Knowing the full BD_ADDR of a specific device allows lifetime tracking of its location and activity.

6.1.2 BLE

1. **BLE legacy pairing provides no protection against passive eavesdropping.** A successful eavesdropping attack during the pairing process can capture the shared secret keys (i.e., LTK, CSRK, IRK).

2. **BLE Security Mode-1 Level-1 is a no-security mechanism.** It is similar to BR/EDR Security mode-1. It's recommended to use Mode-1 Level-4 for high security.

3. **Just Works does not offer MitM protection during pairing**; like in BR/EDR, rejecting Just Works pairing requests is highly recommended.

4. **There is a design flaw in Feature Exchange during Pairing Phase-1:** The exchanged features are not authenticated. This allows a MitM attacker to perform the pairing procedure using different IO capabilities (e.g., forcing one side to use NC and the other side to use PKE with the same value) [18].

6.1.3 Other Vulnerabilities

1. **Negotiating encryption key length.** In BR/EDR, the key can be negotiated up to 1-Byte. Whereas in BLE, the minimum key length is 7-Bytes. It's recommended to use Secure Connection mode in BR/EDR and BLE, in which the key size is 128-bit.

2. **Bluetooth does not support audit and non-repudiation.** If such a service is required, it should be provided by other security means.

It is important to notice that a BLE device which supports also BR/EDR might be vulnerable through its BR/EDR interface.

6.2 Attacks

In this section, we mainly focus on the attacks implemented against BLE, but we also refer to some attacks against Bluetooth BR/EDR. As there are an enormous number of Bluetooth attacks, we will group the attacks into four main sub-categories in the subsequent sections.

6.2.1 Unauthorized Acquisition of Data

This type of attack happens when an unauthorized party accesses Bluetooth characteristic data values communicated between two Bluetooth devices. This usually happens through passive eavesdropping, also known as sniffing.

Sniffing

Sniffing refers to only listening to ongoing communication without intervening in the connection. A Bluetooth sniffer is a combination of software and radio hardware. BLE sniffing won't require a high capabilities tool; the attacker has only to monitor the three primary advertising channels where all connections are initiated. After that, the attacker must follow the connection hopping sequence and hopping interval related to a specific packet's access address of the connection events to capture packets correctly.
To detect the start of a Bluetooth packet in the middle of a connection data stream. One has to recognize the packet preamble and the unique access address (AA), as every Bluetooth packet starts with them. The easiest way to sniff a packet is to be present at the time of the connection initiation because a packet's preamble and access address are only sent during the connection setup. Otherwise, capturing the packet of an already established connection is complicated but possible. In order to sniff ongoing connections, one has to recover the Access Address, CRCInit, hopping interval, and hopping sequence. An enormous number of empty packets is being exchanged between a connection pair—those empty packets can be utilized to filter correct Access Addresses. To verify each packet's CRC, one must recover the CRCInit (initialization value for CRC). The CRC calculation is reversible, so the correct CRCInit value can be identified with enough valid packets. The hop interval can be recovered by waiting on one channel and estimating the period between possible connection events [18]. One of the available tools to capture ongoing and upcoming connections is the modified version Ubertooth project by Mike Ryan in 2013 [14]. A few years later, in 2018, Damien Cauquil presented another similar project "Btlejack", to sniff the ongoing connections [3].

6.2.2 Spoofing

Spoofing is that type of attack when a legitimate device is spoofed due to authentication failure, and the other side of the connection is deceived into accepting the spoofed device as the legitimate peer. A spoofing attack is possible if the attacker can clone the victim's Bluetooth device address, the victim's list of services and characteristics, and the BLE data (i.e., characteristics values) [17]. One or two of the mentioned possibilities would be enough depending on the attack's purpose. The most known attack in this category is the MitM attack.

MitM

A successful MitM attacker has the capability to sniff, forward, and tamper with the Bluetooth connection. The attacker is impersonating two legitimate Bluetooth devices at the same time (i.e., Central and Peripheral). The victim unknowingly connects to the malicious device, which relays the connection to the other legitimate device. Thus, the attacker can drop or inject some data into the devices. In 2016, at the DefCon 24 conference, Damien Cauquil published the famous “BTLEjuice” [5] as a BLE MitM tool. In the same year, Slawomir Jasek presented another MitM tool known as GATT-Tracker [8]. Both tools roughly have the same functionality. Mainly and most importantly, they clone the victim’s BD_ADDR and its Peripheral characteristics and services, broadcasting advertising packets that are indistinguishable from the legitimate ones.

6.2.3 Denial of Service (DoS)

It is that type of attack denying the legitimate device from connecting to its peer, temporarily or permanently. DoS can be in one of the following forms:

- Both devices are up but cannot communicate because the attacker blocks the connection channel (e.g., jamming);
- Making one of the devices unavailable by exhausting its capabilities/resources (e.g., battery drain, crashing);
- Dropping the packets during a MitM attack.

DoS, in its simplest form, is known by the jamming attacks, which we will see in the next section.

Jamming

Bluetooth utilizes the free, unlicensed, and crowded ISM band at 2.4 GHz. The interference and collision of the ISM band are bypassed by using the Adaptive FHSS (AFH) at the PHY layer. Even though, with all these measures, Bluetooth signals can be interfered and forced to collide, meaning AFH is not considered as a security mechanism, as also stated in [16] “Bluetooth system employs a frequency hopping to combat interference and fading”. Jamming Bluetooth BR/EDR is considered more complicated and overwrought, since there are 79 Bluetooth channels. For BLE, it is easier to jam only the three primary advertising channels. In this way, the attacker will easily deny and drop the connection initiation and the advertising packets. During connection initiation, the central shall set a timeout value for the received advertising packets from the connection peer. By blocking the connection, the packets timer will be exceeded, and the central will assume the connection to be lost. The attacker will jam the packets transmitted from the peripheral to the central device. Thus the timer at the central device is to be exceeded. After that, the attacker will impersonate the central device and initiate a new connection to the legitimate peripheral. In 2018, Damien Cauquil implemented the Btlejack tool for jamming BLE [3]. Recently, In 2021 Romain Cayre et al. demonstrated a possible method to inject a packet into an ongoing connection by publishing their work “InjectaBLE” [11].

6.2.4 Privacy attacks

Even though Bluetooth LE employs the privacy feature, the users can still be tracked if a device is tracked. A device that advertises packets periodically with its static address is more likely prone to attacks. However, many vendors still prefer to use the static address [18]. Re-identification of a Bluetooth device can be applied in one of three forms of attacks [13]: User tracking and surveillance, stalking and espionage, and compromising physical assets.

BLE resolvable private address, as discussed in Section 5.6, will prevent the device from tracking only if the pairing procedure has been implemented in one of the secure modes, where the distributed keys (i.e., IRK) cannot be extracted. If the device allows unauthenticated pairing (e.g., Just Works), an attacker can pair with the victim's device and exploit the IRK. Once IRK has been exposed, the attacker fully controls the device's privacy. It's mentioned in Bluetooth core specification v5.3 [16, page 1657] that the IRK can be assigned once to a device during manufacturing, meaning a distributed IRK results in lifetime traceability for that device. In [19], and [12], successful attacks have been implemented against BLE devices in the absence of the privacy feature (i.e., static address). In [10] Zhang et al. carried out a BLE attack in the presence of the privacy feature; they spoofed a legitimate Peripheral and connected to the legitimate Central; the spoofed peripheral generated a Link Layer encryption error (i.e., 0x06 Key Not Found), which forces the communication to be in insecure authentication mode by triggering Just Work association model.

In 2020, Gringoli et al. utilized a full-band SDR system [13] to sniff the 79 channels of Bluetooth BR/EDR with the purpose of reidentification and tracking a BD_ADDR by exploiting weaknesses in the design of the header error check (HEC) and header whitening mechanisms. They proved that it is possible to de-anonymize Bluetooth identity within 85 meters, achieving 100% identification in only 4-seconds.

Chapter 7

Sniffing BLE connections with Btlejack

In this chapter we discuss an attempt to break BLE security using a low-cost device, a Micro:bit equipped with the Btlejack software. However, to better understand the attack, we first need to describe more in-depth how a BLE connection begins. For this reason we will focus on the Advertising state of a BLE device and analyze the packets exchanged at the beginning of a new connection.

7.1 Introduction

In chapter 3, Section 3.2.2, we have seen that a BLE device can operate in three main states: Advertising, Scanning, and Connected state. Advertising allows devices to broadcast information defining their intentions. The **Advertiser**, usually a Peripheral device, broadcasts its presence allowing other devices to find it and, if possible, to connect to it. By contrast, the **Scanner**, which usually is the Central device, listens to the primary advertising channels for broadcast transmissions and eventually sends connection requests to a target advertiser. If the advertiser allows connections, they both enter into the connected state. Understanding these states is essential for anyone who wishes to sniff a BLE connection. More specifically, the advertising and the connection request packets are key packets, because there we can find the connection's most important parameters, such as CRCInit, channel map, and the hop interval. More details will be covered as we go through the chapter.

7.2 BLE Advertising Packets

Bluetooth Low Energy defines a unique packet format for advertising and data transmissions. Every BLE packet contains four fields: Preamble (1-byte), Access Address (4-bytes), Protocol Data Unit (PDU, 2-258 bytes), and Cyclic Redundancy Check (CRC, 3-bytes); Figure 7.1 shows the format of a generic BLE packet.
The PDU field defines whether the packet is an advertising or data packet. Here we focus our attention on advertising PDUs only, which are divided into two main fields: a 2-byte Header and a variable size payload, as shown in Figure 7.2. The advertising Header contains six fields, while the PDU Payload has a different structure depending on the advertisement type.

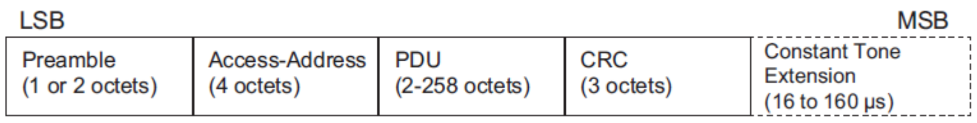


Figure 7.1: Link Layer packet format for the LE Uncoded PHYs

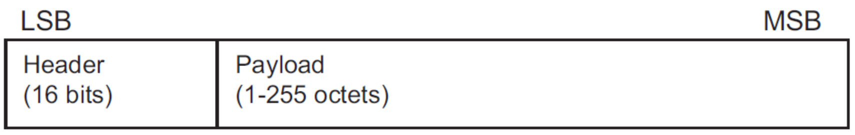


Figure 7.2: BLE Advertising PDU structure

7.2.1 BLE Advertising PDU Header

Figure 7.3 shows a BLE advertising PDU header's fields. The **PDU Type** field indicates the PDU type; different PDU types are explained in Bluetooth Core Specification Version 5.3 — Vol 6, Part B, Section 2.3, Table 2.3 [16]. The **ChSel** field indicates whether or not the BLE Channel Selection Algorithm #2 is supported if supported (CheSel = 1); otherwise set to 0. **RFU** is Reserved for Future Use. **TxAdd** field indicates whether the advertiser's address is public (TxAdd = 0) or random (TxAdd = 1); the difference between public and random address is discussed in Chapter 3, Section 3.2.2.1. The **RxAdd** indicates whether the target's address in the TargetA "the initiator" field is public (RxAdd = 0) or random (RxAdd = 1). The field is only used in the case of direct advertisements "ADV_DIRECT_IND" that we do not treat in this document. The **Length** field indicates the payload length in bytes (1 to 255 bytes).

7.3 BLE Advertising State

In order to understand how to capture a new BLE connection, one must understand not only the packets' structure but also how the states of the devices evolve during the connection initiation.

BLE has seven different types of Advertising packets, but the two main ones are *connectable scannable* and *nonconnectable scannable* undirected advertisements.

1. Nonconnectable scannable undirected advertisements:

The Advertiser broadcasts data to any other BLE device, without setting up a new connection, i.e., it cannot respond to any Connection Request. BLE beacons often employ this type of packets, and this category of devices is also called *connectionless.*

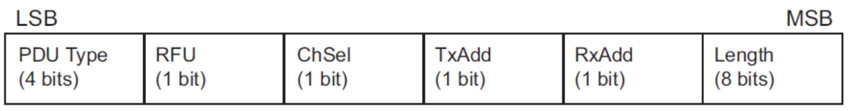


Figure 7.3: Fields of the BLE Advertising PDU Header

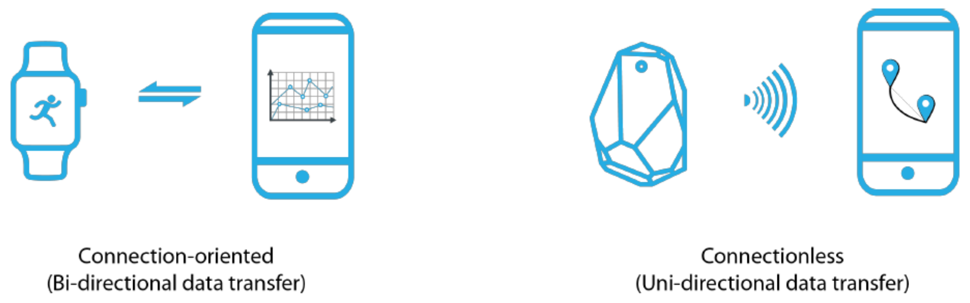

Connection-oriented
(Bi-directional data transfer)
Connectionless
(Uni-directional data transfer)

Figure 7.4: Connection-oriented vs. connectionless transmissions

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Figure 7.5: ADV_IND PDU Payload

2. Connectable scannable undirected advertisements:

The Advertiser can accept connections from a Scanner listening to the advertisements. Devices from this category are also named *connection-oriented*. Figure 7.4 shows an example of the two category [6]. The “PDU Type” field in the advertising PDU header indicates the type of data contained in the following payload. Different PDU types are explained in Bluetooth Core Specification Version 5.3 — Vol 6, Part B, Section 2.3, Table 2.3 [16].

In this Chapter, we will focus only on the type of advertisements that result in a long-term connection. The main advertising packet is ADV_IND; therefore will only give details to this one. Advertisements of this type are sent with the fixed Access Address `0x8E89BED6`.

7.3.1 ADV_IND

Advertising Indications (ADV_IND) are packets sent when a peripheral device is willing to accept connection requests from any central device. An example is a smartwatch requesting a connection to a smartphone. As shown in Figure 7.5, ADV_IND payload consists of AdvA and AdvData fields. The **AdvA** field contains the advertiser’s device address (public or random). The **AdvData** field includes the Advertised Data; otherwise, it is empty.

7.3.1.1 Advertising Data “AdvData”

The advertising data **AdvData** follows the well-known LTV (length-type-value) format, which contains multiple structures and different fields. Each of these has 1-byte of **AD Length**, 1-byte of **AD type** (different AD types are defined in Bluetooth SIG GAP Data Types [2]), and 29-bytes of AD Data as shown in figure 7.6.

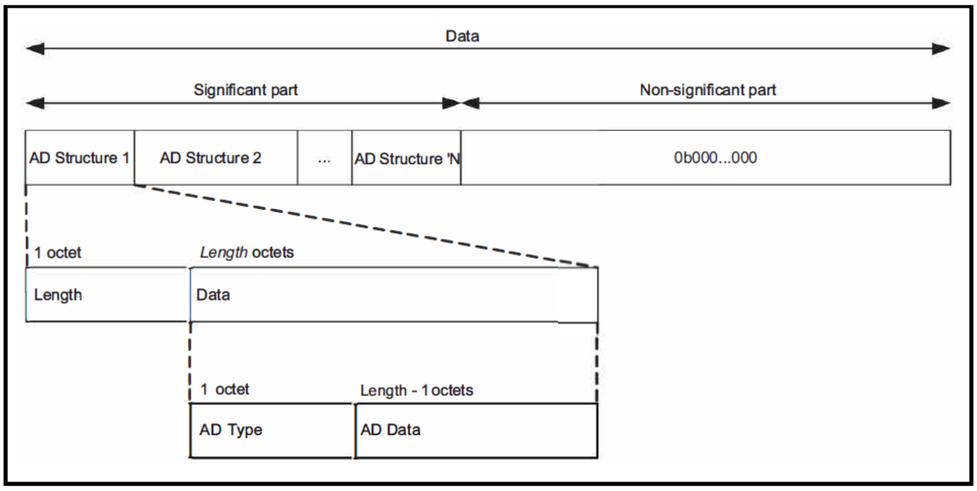


Figure 7.6: ADV_IND PDU Payload Advertised Data

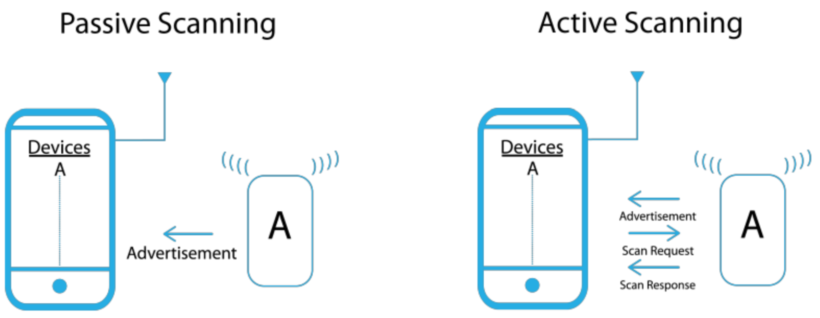


Figure 7.7: BLE Scanning Types

7.4 BLE Scanning State

If a Central "Scanner" wants to discover a Peripheral, it has to be tuned to the same channel on which the Peripheral is advertising at that given point. A device that is listening for advertisements and then sends "Scan Requests" is said to be in *active scanning mode*. In contrast, a device that passively listens to advertisements and does not transmit "Scan Request" is said to be in *passive scanning mode*. Figure 7.7 shows an example of both modes: the device A is advertising its presence, which is detected by the scanning smartphone.

7.4.1 Scanning Parameters

The main scanning parameters are the Scanning Type, Scanning Window, and Scanning Interval. The **Scanning Type** can be either Passive or Active as we have just seen. **Scan Window** tells how long to scan for advertisements. And finally, **Scan Interval** indicates how frequently to scan for advertisements. The scanner (i.e., the Central) will listen for the entire scan window at every scan interval and on a different Primary Advertising Channel in each scan window. Scan window and scan interval are configurable aspects of a scanner's behavior. Figure 7.8 shows the a scanning device parameters.

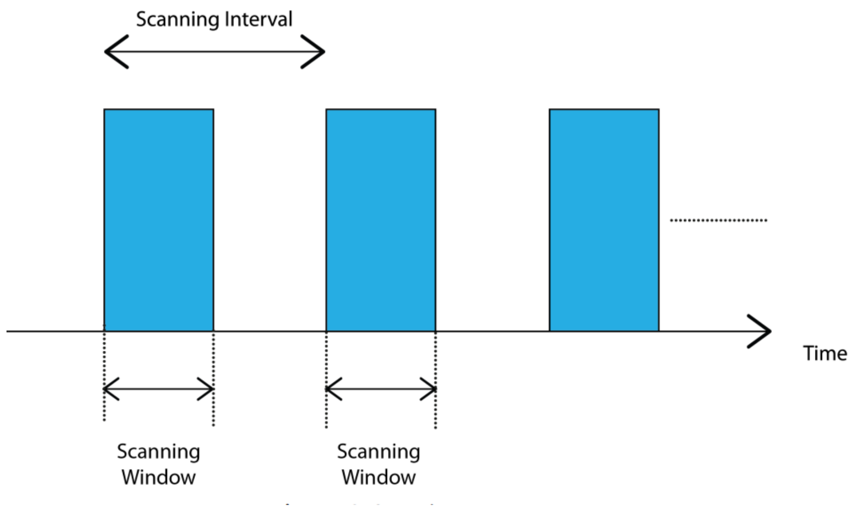


Figure 7.8: BLE Scanning Parameters

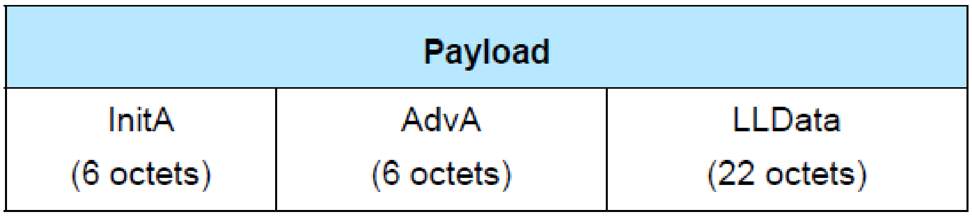
7.5 BLE Connection State

For two BLE devices to establish a connection, they have to go through the following procedure. First, the peripheral needs to start advertising and send out connectable advertisement packets that we have seen in Section 7.3. The Central device needs to scan for advertisements as described in Section 7.4. If the Central device is scanning on the same advertising channel on which the Peripheral is advertising, then the Central device discovers the Peripheral. It can then read the advertisement packet and all the necessary information to establish a connection. The Central then sends a CONNECT_IND packet“connection request.” The Peripheral always listens for a short interval on the same advertising channel after it sends out the advertising packet. This allows it to receive the connection request packet from the Central device — which activates the connection initiation between the two devices. Afterward, the connection is assumed to be created, but not yet established. A connection is established only when the device receives a packet from its peer. After the connection is established, the Central becomes the Master, and the Peripheral becomes the Slave. The Master manages the connection, controls the connection parameters, and the timing of connection events.

7.5.1 CONNECT_IND

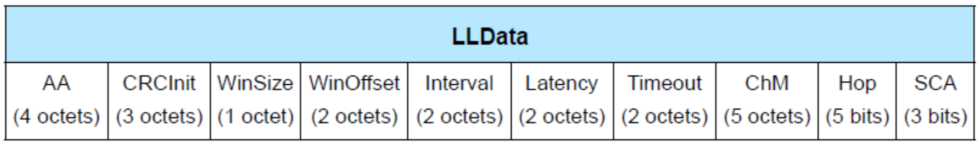
CONNECT_IND Payload fields are shown in Figure 7.9. **InitA** represents the initiator’s device address, whereas **AdvA** indicates the advertiser’s device address. **LLData** field contains important information as detailed in Figure 7.10. **LLData** is constructed in 10 fields:

- *AA* field contains the ACL connection’s Access Address defined by the Link Layer.
- *CRCInit* field contains the initialization value for the CRC calculation for the ACL connection. It’s a random value generated by the LL.
- *WinSize* field indicates the transmit window size value, as defined in the following manner: $Transmitwindowsize = WinSize * 1.25ms$.



Payload		
InitA (6 octets)	AdvA (6 octets)	LLData (22 octets)

Figure 7.9: CONNECT_IND Payload Fields



LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

Figure 7.10: CONNECT_IND LLData Field Structure

- *WinOffset* field indicates the transmit Window Offset value, as defined in the following manner: $transmitWindowOffset = WinOffset * 1.25ms$.
- *Interval* field indicates the connInterval as defined in the following manner: $connInterval = Interval * 1.25ms$.
- *Latency* field indicates the connPeripheralLatency value, as defined in the following manner: $connPeripheralLatency = Latency$.
- *Timeout* field indicates the connSupervisionTimeout value, as defined in the following manner: $connSupervisionTimeout = Timeout * 10ms$.
- *ChM* field contains the channel map indicating Used and Unused data channels. Every channel is represented with a bit positioned as per the data channel index.
- *Hop* field indicates the hopIncrement used in the data channel selection algorithm. It shall have a random value in the range of 5 to 16.
- *SCA* indicates the centralSCA used to determine the Central's worst-case sleep clock accuracy. The value of the SCA field shall is defined in [16] Vol 6, Part B, Table 2.9.

7.6 Btlejack

Btlejack is a tool developed by Damien "virtualabs" Cauquil to perform BLE attacks (i.e., sniffing, jamming, and hijacking). Btlejack's current version is 2.0, which supports BLE 4.x and 5.x. The BLE 5.x support is limited, as it only supports the 1 Mbps Uncoded PHY and does not support channel map updates [4].

7.6.1 Experimental setup

For the investigated experiment, we used a laptop with a Linux OS, one BBC Micro:Bit, one nRF52 development board that was programmed to emulate a HeartRate sensor to serve as the **Advertiser**, and a VIVO X60 mobile phone to serve as the **Central**. The experiment setup is shown in Figure 7.11. In this work, we only used one Micro:Bit,

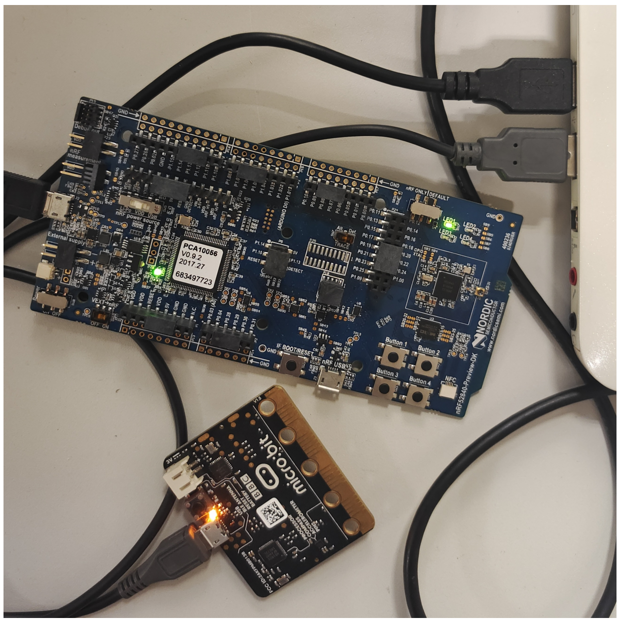


Figure 7.11: Btlejack Experimental Setup

allowing Btlejack to work only in one of the three Primary Advertising Channels, leaving us with a 0.33 probability of sniffing the packets. Three Micro:Bits would be enough to detect new connections with 100% proability; however, for the sake of this experiment, if the Micro:Bit fails to detect a new connection we will simply disconnect and reconnect the devices until the connection is captured.

7.6.1.1 Installation and Preparation

Btlejack can be installed following the guide in [4]. We already have Python3 installed in our system. Accordingly, we will use the following command to install Btlejack on the syste,/

```
$ sudo pip3 install btlejack
```

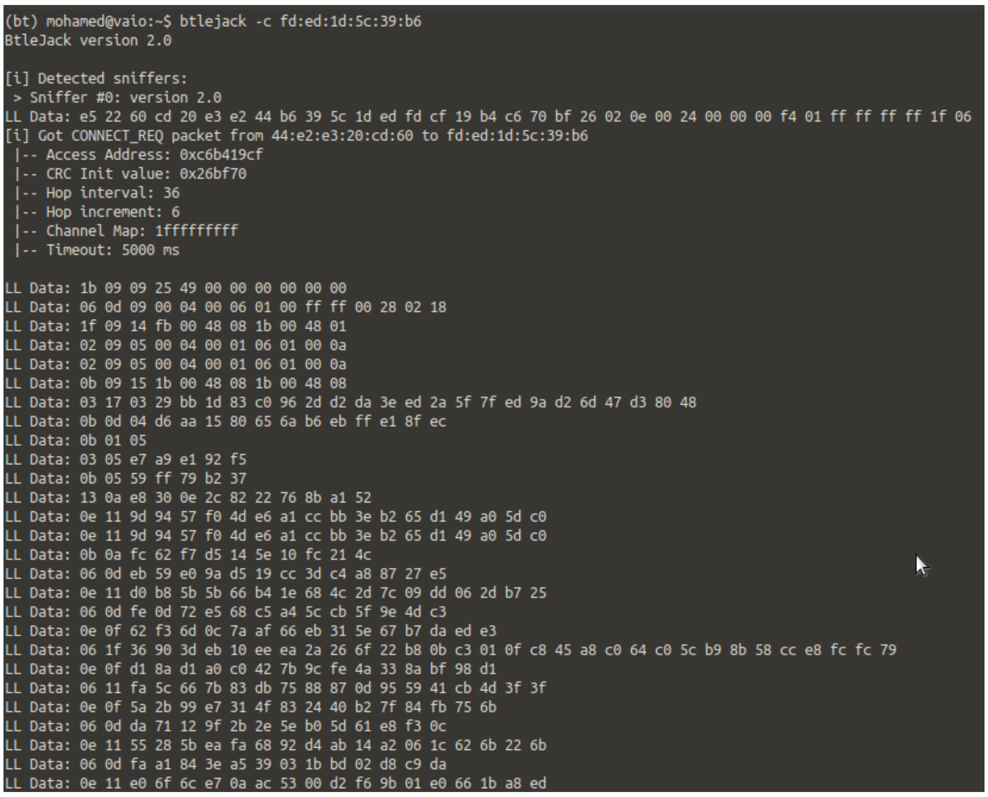
The Micro:Bit device is connected to the host laptop via a USB cable. To load (install) the BtleJack firmware on the Micro:bit, we run:

```
$ btlejack -i
```

The command will program the Micro:Bit with the correct firmware version for the current client software. At this point we have our sniffer up and ready, so we will start sniffing the packets on one of the Primary Advertising Channels.

7.7 Sniffing with Btlejack

To identify a BLE device, one has to capture the following: The advertising data that is sent in the advertising packet (ADV_IND); The channel map used by the Master device; and the used hop interval. The Advertised data is a collection of advertising records, as we have seen in Section 7.3.1.1. Each record contains a 1-byte length, 1-byte AD Type,



```
(bt) mohamed@vaio:~$ btlejack -c fd:ed:1d:5c:39:b6
BtleJack version 2.0

[i] Detected sniffers:
 > Sniffer #0: version 2.0
LL Data: e5 22 60 cd 20 e3 e2 44 b6 39 5c 1d ed fd cf 19 b4 c6 70 bf 26 02 0e 00 24 00 00 00 f4 01 ff ff ff ff 1f 06
[i] Got CONNECT_REQ packet from 44:e2:e3:20:cd:60 to fd:ed:1d:5c:39:b6
 |-- Access Address: 0xc6b419cf
 |-- CRC Init value: 0x26bf70
 |-- Hop interval: 36
 |-- Hop increment: 6
 |-- Channel Map: 1ffffffff
 |-- Timeout: 5000 ms

LL Data: 1b 09 09 25 49 00 00 00 00 00 00
LL Data: 06 0d 09 00 04 00 06 01 00 ff ff 00 28 02 18
LL Data: 1f 09 14 fb 00 48 08 1b 00 48 01
LL Data: 02 09 05 00 04 00 01 06 01 00 0a
LL Data: 02 09 05 00 04 00 01 06 01 00 0a
LL Data: 0b 09 15 1b 00 48 08 1b 00 48 08
LL Data: 03 17 03 29 bb 1d 83 c0 96 2d d2 da 3e ed 2a 5f 7f ed 9a d2 6d 47 d3 80 48
LL Data: 0b 0d 04 d6 aa 15 80 65 6a b6 eb ff e1 8f ec
LL Data: 0b 01 05
LL Data: 03 05 e7 a9 e1 92 f5
LL Data: 0b 05 59 ff 79 b2 37
LL Data: 13 0a e8 30 0e 2c 82 22 76 8b a1 52
LL Data: 0e 11 9d 94 57 f0 4d e6 a1 cc bb 3e b2 65 d1 49 a0 5d c0
LL Data: 0e 11 9d 94 57 f0 4d e6 a1 cc bb 3e b2 65 d1 49 a0 5d c0
LL Data: 0b 0a fc 62 f7 d5 14 5e 10 fc 21 4c
LL Data: 06 0d eb 59 e0 9a d5 19 cc 3d c4 a8 87 27 e5
LL Data: 0e 11 d0 b8 5b 5b 66 b4 1e 68 4c 2d 7c 09 dd 06 2d b7 25
LL Data: 06 0d fe 0d 72 e5 68 c5 a4 5c cb 5f 9e 4d c3
LL Data: 0e 0f 62 f3 6d 0c 7a af 66 eb 31 5e 67 b7 da ed e3
LL Data: 06 1f 36 90 3d eb 10 ee ea 2a 26 6f 22 b8 0b c3 01 0f c8 45 a8 c0 64 c0 5c b9 8b 58 cc e8 fc fc 79
LL Data: 0e 0f d1 8a d1 a0 c0 42 7b 9c fe 4a 33 8a bf 98 d1
LL Data: 06 11 fa 5c 66 7b 83 db 75 88 87 0d 95 59 41 cb 4d 3f 3f
LL Data: 0e 0f 5a 2b 99 e7 31 4f 83 24 40 b2 7f 84 fb 75 6b
LL Data: 06 0d da 71 12 9f 2b 2e 5e b0 5d 61 e8 f3 0c
LL Data: 0e 11 55 28 5b ea fa 68 92 d4 ab 14 a2 06 1c 62 6b 22 6b
LL Data: 06 0d fa a1 84 3e a5 39 03 1b bd 02 d8 c9 da
LL Data: 0e 11 e0 6f 6c e7 0a ac 53 00 d2 f6 9b 01 e0 66 1b a8 ed
```

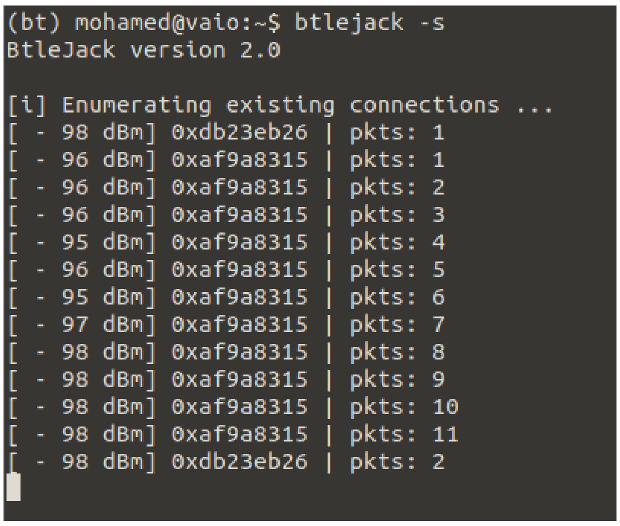
Figure 7.12: Captured Packets for Heart-Rate Sensor

and up to 29-bytes data.
Sniffing a BLE packet can be of two types: sniffing new connections or an ongoing connection.

7.7.1 Sniffing new connections

Sniffing a BLE connection from the start is a straightforward task: One has to first wait on one of the advertising channels (37, 38, or 39) for a CONN_IND connection request packet; then synchronize with both devices and use the provided parameters to follow and sniff packets. Btlejack provides this via the `-c` option by specifying the target's BD_ADDR; Btlejack will filter the connection requests and only sniff connections to this target. Another option is to capture all newly created connections using `-c any`.

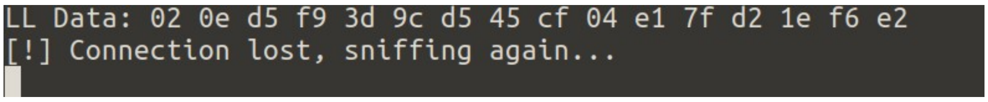
In Figure 7.13, we provide a capture for new connections in the surrounding environment. The first field represents Signal strength indication in dBm; the second field shows the Access Address identifying a link between a connection pair; the last field defines the number of received packets corresponding to the given Access Address. Figure 7.12 shows captured packets only from the device with the following BD_ADDR `FD:ED:1D:5C:39:B6` (i.e., the heart-rate sensor). Unfortunately, we could not sniff the whole connection due to resource limitations. The utilized Micro:Bit was unable to cope with the Central device (i.e., the VIVO X60 smartphone) channel hopping, as the channel map changes regularly, meaning that the Central device employs some SNR estimation to avoid congested channels. Therefore, we were only able to capture the connection initiation packets where after that, the connection is lost. Then Btlejack starts sniffing from the beginning, as shown in Figure 7.14.



```
(bt) mohamed@vaio:~$ btlejack -s
BtleJack version 2.0

[i] Enumerating existing connections ...
[ - 98 dBm] 0xdb23eb26 | pkts: 1
[ - 96 dBm] 0xaf9a8315 | pkts: 1
[ - 96 dBm] 0xaf9a8315 | pkts: 2
[ - 96 dBm] 0xaf9a8315 | pkts: 3
[ - 95 dBm] 0xaf9a8315 | pkts: 4
[ - 96 dBm] 0xaf9a8315 | pkts: 5
[ - 95 dBm] 0xaf9a8315 | pkts: 6
[ - 97 dBm] 0xaf9a8315 | pkts: 7
[ - 98 dBm] 0xaf9a8315 | pkts: 8
[ - 98 dBm] 0xaf9a8315 | pkts: 9
[ - 98 dBm] 0xaf9a8315 | pkts: 10
[ - 98 dBm] 0xaf9a8315 | pkts: 11
[ - 98 dBm] 0xdb23eb26 | pkts: 2
```

Figure 7.13: Finding Existing Connections



```
LL Data: 02 0e d5 f9 3d 9c d5 45 cf 04 e1 7f d2 1e f6 e2
[!] Connection lost, sniffing again...
```

Figure 7.14: Btlejack Connection Lost Message

7.7.2 Sniffing an ongoing connection

In the case of an already established connection, we cannot capture the required parameters from the CONNECT_REQ PDU as the Master device has already sent it. The following parameters should be calculated: CRCInit, channel map, hop interval and hop increment. In the case of new connections, we have only provided the BD_ADDR; in contrast, for sniffing an ongoing connection, one has to provide the target's connection **Access Address**. For this to work and to know the target's Access Address, we first have to use the Btlejack `-s` option to listen to all the ongoing connections. Therefore we can target one device from its connection Access Address. Once the target's Access Address is determined, Btlejack can be used with the `-f` (follow) option:

```
$ btleJack -f <access_address>
```

Having set this, Btlejack will recover all the required parameters. However, as in the case of sniffing new connections, the hardware limitation of the Micro:Bit makes this process very slow and prone to errors.

Chapter 8

Conclusion and Prospective

In this chapter, we revisit our thesis questions. Likewise, we provide some suggestions to the interested parties about Bluetooth weaknesses, based on our observations, in order to improve the overall security of Bluetooth.

8.1 Conclusion and Prospective

In this thesis, we wanted to detail different Bluetooth security aspects, both for BR/EDR and BLE; we have seen several studies uncovering threats and vulnerabilities and implementing some attacks exploiting these vulnerabilities. With this knowledge of Bluetooth's security, we verified how even a tool with limited resources can be used to mount simple attacks to users' privacy. Following the same principle, BR/EDR is susceptible to such attacks. We also discussed the advantages of sniffing the BLE Primary Advertising Channels with at least three Micro:Bits, or a more powerful SDR system covering the entire spectrum of BLE. The adoption of such systems can be considered for future works on this topic.

We have seen that it is a difficult task to have a secure Bluetooth device. Nevertheless, as we have also seen, the Bluetooth SIG provided many ways to mitigate potential threats. Therefore, high security can be achieved by employing all the security mechanisms provided by the Bluetooth core specification in the most current versions.

To have a secure data exchange, devices are recommended to perform bonding in order to store the security materials (i.e., LTK and Link Key). The use of Legacy Pairing and Just Works in both Pairing mechanisms must be avoided as it might expose the device's security. In contrast, devices should employ the Secure Connection mechanism which offers the maximum availbe security. Furthermore, user's privacy should be protected by regularly activating the Privacy feature in BLE. The CTKD–Cross-Transport Key Derivation–feature can be a threat if one of the two Blueetooth systems (BR and BLE) utilizes a weak security mechanism, allowing the attacker to breach the other system. Of course, users must also take care of security themselves, because even the strongest passkey or PIN is vulnearble to malicious visual eavesdropping.

Bibliography

[1] Mohammad Afaneh. *INTRO TO BLUETOOTH LOW ENERGY*. 2018.

[2] Bluetooth SIG. *Generic Access Profile*. `https://www.bluetooth.com/specifications/assigned-numbers/`. Accessed: 2022-09-23. 2022.

[3] Damien Cauquil. *You'd better secure your BLE devices or we'll kick your butts*. Tech. rep. DEF CON 26, 2018.

[4] D. Cauquil. *BtleJack: A new Bluetooth Low Energy Swiss-army knife*. `https://github.com/virtualabs/btlejack`. 2018.

[5] D. Cauquil. *BtleJuice framework*. `https://github.com/DigitalSecurity/btlejuice`. 2016.

[6] *Developer Study Guide: Bluetooth Low Energy Security*. 2021.

[7] Robin Heydon. *Bluetooth Low Energy; The Developer's Handbook*. 2012.

[8] Sławomir Jasek. *GATTacking Bluetooth Smart Devices*. Tech. rep. SecuRing, 2016. URL: `http://gattack.io/whitepaper.pdf`.

[9] "Chapter 9 - Designing an Audio Application". In: *Bluetooth Application Developer's Guide*. Ed. by David Kammer et al. Burlington: Syngress, 2002, pp. 379–417. ISBN: 978-1-928994-42-8. DOI: `https://doi.org/10.1016/B978-192899442-8/50012-4`. URL: `https://www.sciencedirect.com/science/article/pii/B9781928994428500124`.

[10] Y. Zhang; J.Weng; R. Dey; Y. Jin; Z. Lin; and X. Fu. "Breaking secure pairing of Bluetooth Low Energy using downgrade attacks". In: *29th USENIX Security Symposium* (2020), pp. 37–54. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/zhang-yue`.

[11] R. Cayre; F. Galtier; G. Auriol; V. Nicomette; M. Kaâniche; G. Marconato. *InjectaBLE; INjecting malicious traffic into established Bluetooth Low Energy connections6*. Tech. rep. Taipei (virtual); Taiwan: IEEE/IFIP International Conference On Dependable Systems and Networks; DSN, 2021. URL: `https://hal.laas.fr/hal-03193297`.

[12] A.K. Das; P.H. Pathak; C.-N. Chuah; P. Mohapatra. *Uncovering privacy leakage in BLE network traffic of wearable fitness trackers*. Tech. rep. Proceedings Of The 17th International Workshop On Mobile Computing Systems and Applications, 2016. URL: `https://dl.acm.org/doi/10.1145/2873587.2873594`.

[13] Marco Cominelli; Francesco Gringoli; Paul Patras; Margus Lind; Guevara Noubir. *Even Black Cats Cannot Stay Hidden in the Dark: Full-band De-anonymization of Bluetooth Classic Devices*. Tech. rep. 2020. URL: `https://ieeexplore.ieee.org/document/9152700`.

[14] Mike Ryan. *Bluetooth: with low energy comes low security.* Tech. rep. iSEC Partners, 2013. URL: `https://www.usenix.org/conference/woot13/workshop-program/presentation/ryan`.

[15] John Padgette; John Bahr; Mayank Batra; Marcel Holtmann; Rhonda Smithbey; Lily Chen; Karen Scarfone. *Guide to Bluetooth Security.* NIST, National Institute of Standards and Technology, 2022. DOI: `https://doi.org/10.6028/NIST.SP.800-121r2`.

[16] Bluetooth SIG. *Bluetooth Core Specification.* 2022. DOI: `https://www.bluetooth.com/specifications/adopted-specifications`.

[17] Pallavi Sivakumaran. "Security and Privacy in Bluetooth Low Energy". PhD thesis. Information Security Group; Royal Holloway University of London, 2021.

[18] Matthias Cäsar; Tobias Pawelke; Jan Steffan; Gabriel Terhorst. *A survey on Bluetooth Low Energy security and privacy.* Tech. rep. Rheinstraße 75, 64295 Darmstadt, Germany: Fraunhofer Institute for Secure Information Technology SIT, 2022.

[19] Taher Issoufaly; Pierre Ugo Tournoux. *BLEB: Bluetooth Low Energy Botnet for large scale individual tracking.* Tech. rep. 1st International Conference On Next Generation Computing Applications (NextComp), 2017. URL: `https://www.overleaf.com/project/62ff4672994575dc857f67b8`.

[20] Wikipedia. *Summation generator.* URL: `https://en.wikipedia.org/wiki/Summation_generato`. (accessed: 25.08.2022).