Bluetooth Low Energy security analysis framework

Jennifer Ann Janesko

Technical Report

RHUL–ISG–2018–5

5 April 2018

Information Security Group
Royal Holloway University of London
Egham, Surrey, TW20 0EX
United Kingdom

Student Number: 120232774
Janesko, Jennifer Ann

Title: Bluetooth Low Energy Security Analysis Framework

Supervisor: Jorge Blasco Alis

Submitted as part of the requirements for the award of the
Msc in Information Security
at Royal Holloway, University of London.

This page is intentionally left blank.

Acknowledgements

I would like to thank my thesis advisor, Dr. Jorge Blasco Alis, of the Information Security Group at Royal Holloway, University of London. He provided exactly the right amount of guidance and direction to keep me on track. He has read multiple drafts of this thesis and provided both constructive and encouraging feedback. I hope that this thesis meets his expectations in terms of quality and deliverables.

I would also like to acknowledge and thank those professors and instructors whose courses gave me the background knowledge to work on (and enjoy) this project: Dr. David Evans, Dr. Dan Boneh, Dr. Keith Martin and Mr. Po Yau.

I would also like to thank my running partner, Aygul Shugeva. During our jogs she listened to my discoveries and stresses surrounding BLE testing and provided technical feedback and ideas that I had not yet considered. It is great to have an embedded pentester as a friend.

I would like to thank Mike Ryan for taking a minute from his hectic schedule to clarify the vocabulary pertaining to the BLE's description of encryption and MACs. Without his prompt feedback, I would not have been able to move forward in a timely fashion.

And, last but not least, I want to thank my husband and partner in all things, Tim Thomas. He gave me the time, space and (some) gadgets for this project, he listened when I prattled on about specifications, writing and scripting, he booked the hotel room with a balcony and wifi overlooking the ocean and he gave me encouragement and chocolate as needed. I'm done, Schatz. Let's go climb a mountain.

Table of Contents

Index of Figures

1 Project Introduction

1.1 Bluetooth Low Energy

Bluetooth low energy (BLE) is a wireless communication technology that allows the short range, wireless exchange of communication between devices. It is a special type of Bluetooth that was developed by the Bluetooth SIG with the specific goal to operate as efficiently as possible so that devices conserve battery power. It was introduced in version 4.0 of the Bluetooth specification. At the start of this project, the Bluetooth specification is at version 4.2. At the time of writing, the Bluetooth SIG started promoting version 5.0, although it had not officially been released.

Traditionally, Bluetooth has been used as a technology to organize devices wirelessly into networks called “piconets”. These piconets are often described as “ad hoc” because they can be built up and torn down quickly. “Ad hoc” has the connotation that the use of Bluetooth are impermanent and possibly only relevant for non-serious applications such as exchanging pictures between to mobile phones. But, the Bluetooth SIG has been marketing its Bluetooth low energy technology as a solution for Internet of things communications [B516].

Internet of things, or “IoT”, is a term that is used in the press, but is not clearly defined. Zielgedorf, et. al., outline a reference model by which IoT components can be categorized. IoT is described as “anyone and anything [that] is interconnected anywhere at any time via any network participating in any service.” While this definition seems overly broad, the authors refine it by describing five[1] types of entities that participate in an IoT application and their related information flows.

1 Ziegeldorf, et.al., actually introduce four entities where “subjects” and “recipients” are combined into one entity under the category of “humans”. This is appropriate for their paper because their research focuses on the privacy challenges embedded in the use of IoT devices by human individuals in everyday life. This paper will use IoT in a broader sense to include subjects and recipients that are non-human which is fitting to smart home technologies and manufacturing automation technologies.

- Smart things: These are everyday devices that have been augmented with ICT components to collect data and share this data via services.
- Subjects: These are the entities from or about which smart things collect data for reporting.
- Services hosted on backends: These are services that collect data from the smart things and process that data for use in decision making.
- Recipients: These are the entities that receive feedback and information from the services on the backend.
- Infrastructure: These are the networks that allow communication between smart devices and their backend services. [JHZ13]

Take as a simple example, the logical representation of a smart climate management system in figure 1.

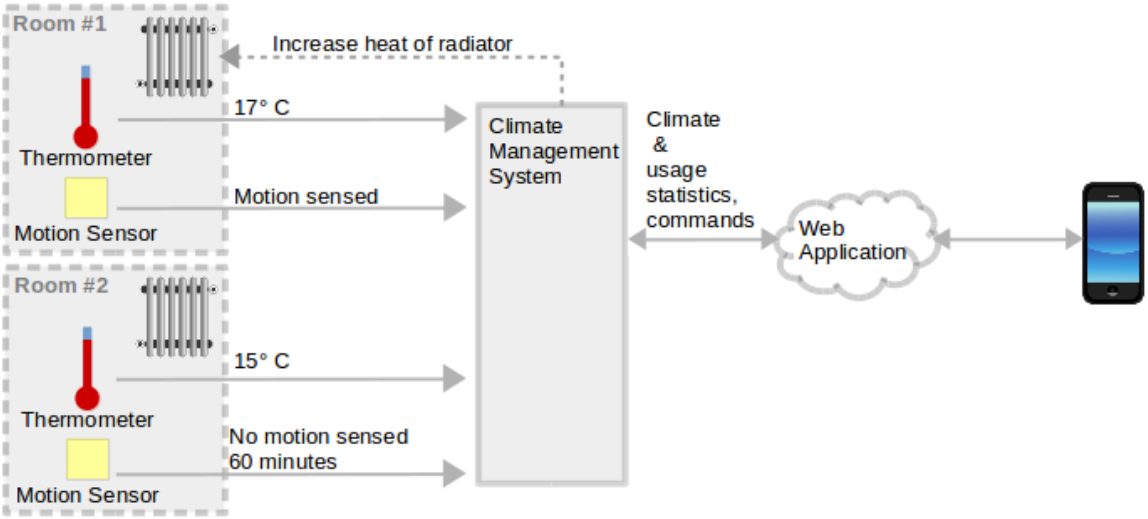

Figure 1: Climate Management Example

This could be considered an IoT example. In the example there are smart things, the thermometers and motion sensors, that collect data from their subjects. In this case, the thermometers' subject is the room temperature. The motion sensors' subject is the (human) movement in the room. These smart devices communicate their data to the client, i.e., the backend client

management system. This system reports statistics to an on-line web application which is viewable by the owner of the building. Based on input from the sensors and the climate and usage statistics over time, the climate management system determines whether to modify the climate controls in each individual room. This type of system has the advantages of being environmentally friendly as well as cutting heating costs.

In the example above, the infrastructure is the network that is set up between the sensors and the server and the server and the external web application. These infrastructures most likely rely on different protocols for communication. The infrastructure that provides a connection to the Internet is often TCP/IP-based. For manufacturing automation, the long-distance communication protocol may be specific to the application. The communication between the smart things and the server usually take place over a wireless protocol.

Wireless protocols provide a few advantages over cabled protocols. Setting up a wireless communications network is usually less costly and requires less effort for physical installation. In addition to this, wireless devices can be installed in areas where it might be a challenge to install cabling. In the IoT market, there are a small number of wireless protocols that are often mentioned: Zigbee, BLE and Thread [VT14]. Zigbee is a proprietary protocol that has been on the market since 2004. Thread is a newcomer (2014) and is backed by IT-giants like Google and NXP. BLE came to the market in 2013, was delivered on 165 million devices by 2014 and is expected to reach 1.2 billion devices by 2018. [RQ15]

Initially, BLE was known for being the communication protocol for smart devices like Apple's iBeacon or for fitness trackers such as a Fitbit. But, BLE's low power consumption has also made it ideal for health monitoring devices and home automation. The fact that BLE is an open standard coupled with its prevalence on users' mobile devices has helped its proliferation [PMGL16][GL16]. In 2016, there was a big move in the controller industry to incorporate BLE for use in applications where reliability is required. Silicon Labs, NXP and Cypress have incorporated BLE into their PSoC offerings

[SLG16][JY16][PM16][GL16]. Open RTOS implemented a BLE stack [RM16]. Telit, an important competitor in the automation market, acquired a BLE stack in January 2016 [TAW16]. They provide equipment in the fields of condition monitoring, industrial automation, predictive maintenance, asset tracking, supply chain management and telematics and fleet management.

BLE is positioned to be a core IoT infrastructure component. And, whether is it used for tracking personal fitness, manging a home or managing factory automation, the question of BLE's security must be considered when developing a device. At the time of this project's writing, there has been little released to provide guidance with respect to IoT. OWASP has an emerging security testing guide for IoT, but at the time of writing, it focuses on the communication between an IoT device and an Internet service. The NIST has provided a set of security guidelines for the use of Bluetooth. These guidelines do include BLE, but the guidelines refer to BLE 4.0 [MSKS13]. In addition to this, there has been piecemeal research into the security aspects of BLE (see Chapters 4), but no structured approach to testing the security of a BLE device has been developed.

1.2 Project scope

The goal of this project to to provide a security analyst with the necessary information to perform a comprehensive security analysis of a device that uses Bluetooth Low Energy (BLE) for communication. To accomplish this task the paper is broken down into the following major sections:

- Introduction to the Bluetooth low energy concepts
- Enumeration of the generic Bluetooth low energy attack surface
- Development of a generic Bluetooth low energy threat model
- Outlining of an approach to BLE security testing

Although the Bluetooth Low Energy communication can be regarded as relatively simple, there is a considerable amount of detail in the specification that is relevant for a security review. To preserve the readability of this paper,

the core body of the text in chapter 2 will be dedicated to elucidating core BLE concepts to the reader that were deemed relevant for security. The appendix of this document will contain a series of tables and listings that will provide necessary detail if a reader plans to use this document as a framework for an actual testing scenario.

1.3 Project Limitations

There are three modes of Bluetooth: BLE, EDR/BR and EDR/BR/BLE.

- BLE, as mentioned above, is Bluetooth Low Energy. BLE was introduced in the 4.0 Bluetooth specification. BLE is also marketed as “Bluetooth Smart”.
- EDR/BR is the version of Bluetooth that has been available since the first release of the Bluetooth specification. EDR/BR is often simply referred to as “Bluetooth” or “classic Bluetooth”. EDR/BR has enough differences from BLE on the host and controller that its communication is not compatible with BLE.
- EDR/BR/BLE devices have both the EDR/BR and BLE stacks built into them. These are also referred to as “dual mode” devices.

This project will only focus on providing a framework for BLE mode testing. EDR/BR and EDR/BR/BLE modes are not in scope for this project. That being said, there is a great deal of overlap in the specification between BLE and EDR/BR modes, and there has been significant research into the security of EDR/BR. Where applicable, results from EDR/BR research will be taken into consideration for this paper.

In addition to this, the version of the Bluetooth specification that was available at the beginning of this project was version 4.2. This project will focus on the analysis of the contents of the 4.2 specification.

1.4 Note on Referencing

Throughout the course of this work, the type of in-text referencing is used where the first 3 letters of the author's name is referenced plus the last two digits of the year of the referenced publication. No page number is provided with this type of referencing.

It is the author's intent to make the Bluetooth specification more accessible for others, and when referencing the Bluetooth specification, the following conventions will be used.

- BLE-LL refers to volume 6 part E of the specification.
- BLE-GAP refers to volume 3 part C of the specification.
- BLE-ATT refers to volume 3 part F of the specification.
- BLE-GATT refers to volume 3 part G of the specification.
- BLE-SMP refers to volume 3 part H of the specification.
- BLE-Supp refers to the Core Specification Supplement version 4 (CSS)

When the specification is referenced, the page numbers of the respective volumes will be included so that it is easier to locate the information and gain further background information.

2 BLE Technology Concepts

2.1 Chapter Scope

The first step to performing a security analysis of an application built on top of BLE is to understand the functional behavior of the application. Further steps involve evaluating how that application interacts with the underlying software and hardware for communication and data processing. A thorough understanding of how data is stored, accessed, manipulated and communicated provides an analyst with the tools to know where to look for security vulnerabilities in both the application logic and the underlying technologies.

The BLE specification is embedded in a document of over 1000 pages. It is broken up into different sections corresponding primarily to the communication stack. In the specification, both BLE and EDR/BR concepts are interspersed. It is common to read a section dedicated to one part of the stack only to find that it references and relies upon other parts of the stack.

The sheer volume of information and its interdependencies in the specification makes it difficult to obtain a quick overview of the technology. For a security analyst, time is often a luxury that is not available when a customer has time pressures of getting a product out to the market. There are plenty of tools that can assist with and expedite analysis, but they produce data like that in the packet capture in figure 2.

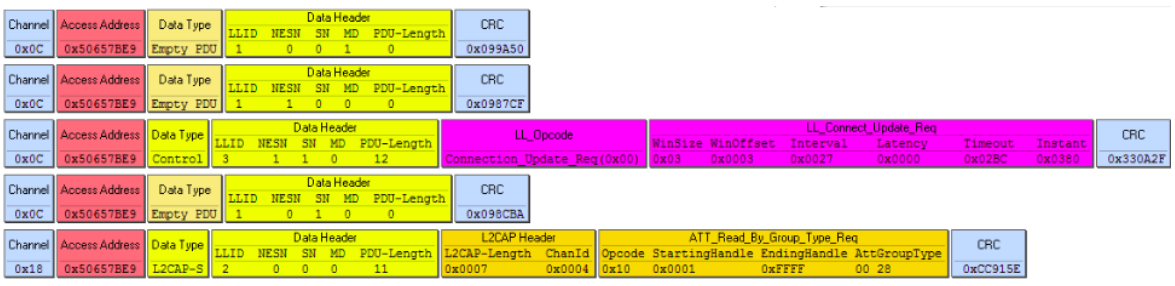

Figure 2: Screen capture of packet exchange using Texas Instruments SmartRF Packet Sniffer

While there is a great deal of information here, and it is helpfully highlighted, it still needs to be interpreted so that an effective security analysis can occur.

This chapter provides an overview of BLE technology to the extent that the different components of of the packet capture above can be interpreted. This requires an overview of the BLE protocol stack, the interdependencies between the “layers” in the stack and the communication modes and procedures that facilitate the exchange of data. Special care is given to the introduction and utilization of key concepts and vocabulary from the specification to assist the analyst in further work with other BLE information sources.

2.2 Overview of the BLE Stack

BLE is a technology that supports the processing and communication of data. It is organized into a stack that is reminiscent of the OSI stack. The BLE stack is comprised of eight separate, but interdependent components, that can be logically represented as shown in figure 3.

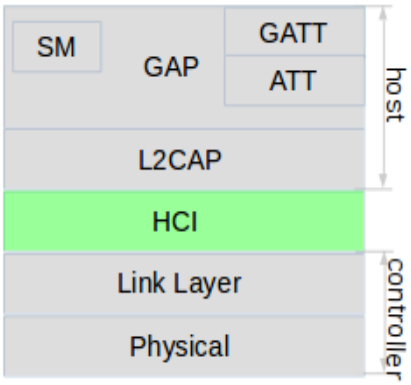

Figure 3: BLE Stack

Typically, as will be discussed later in the chapter, the link layer and physical layers reside on one chip called the “controller. The L2CAP, GAP, SM, ATT and GATT layers reside on a separate chipset called the host. A brief description of each layer is specified in the following table. The vocabulary

that is used in the table is typical of the BLE specification, and it will be clarified throughout the remainder of this chapter.

Layer	**Description**
Physical	The physical layer describes the technical radio components needed for physical, over-the-air communication.
Link Layer	The link layer defines the packet structure for over-the-air communication, defines core PDUs for data and control communication, performs cryptographic functions and supports whitelists and filtering policies. This part of the specification introduces the following states: standby, scanning, advertising, initiating, connection. It also introduces the following roles: master, slave.
HCI	HCI stands for host controller interface. The HCI provides a standard interface to communicate between the host and the controller.
L2CAP	L2CAP stands for logical link control and adaptation protocol. L2CAP establishes advertising and data channels, packages PDUs from the host for processing by the controller and unpackages packets from the controller for use by the host.
GAP	GAP stands for generic access protocol. GAP provides a foundation for the communication of attributes, defines control modes and procedures for advertising communications and serves as the foundation for security. GAP introduces the follow roles: broadcaster, observer, peripheral, central.
SMP	SMP stands for security manager protocol. SMP defines pairing, bonding and the requisite cryptographic key material and key distributions. Additionally, it specifies how private resolvable addresses function.
ATT	ATT stands for attribute protocol. ATT defines attributes and describes the communication of attributes via attribute protocol PDUs. ATT introduces the following roles: server, client.

Layer	Description
GATT	GATT stands for generic attribute profile. GATT defines how attributes are organized into primary and secondary services. It also specifies core attributes. Further it specifies features that facilitate attribute communication and their relationships to ATT attribute protocol PDUs.

As seen in the diagram above, the stack is split up into host and controller. The controller normally supports operations that have real-time requirements. The host supports the application and its supporting data. It is the layer of the stack that is closest to the user. Operations in the host are often implemented in software, rather than hardware.

There are several places in the specification where the host sends control data to the controller. This allows the controller to make decisions about control communications. This strategy is used because controllers are implemented to generally perform more efficiently than the host. This strategy helps to reduce the energy consumption requirements involved in communication.

According to Townsend, et. al., there are three primary physical relationships that can be set up between host and controller: system on chip (SOC), dual IC over HCI and dual IC connectivity device. These are logically represented in Figure 4.

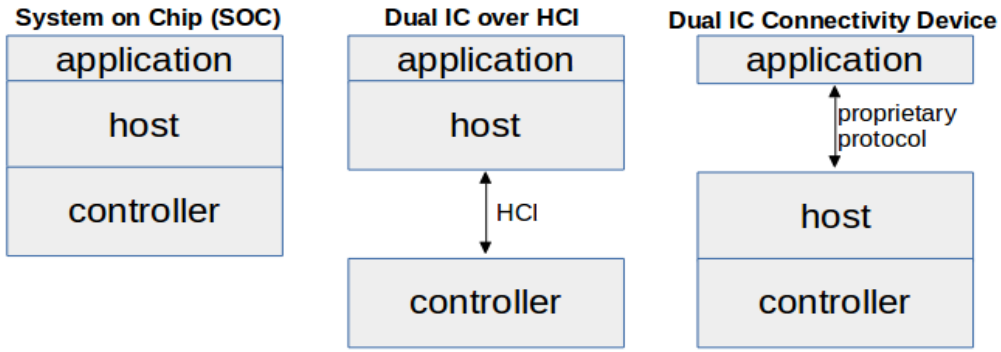

Figure 4: Physical host-controller configurations [Townsend]

In sytem on chip configurations the entire Bluetooth stack and the application are located on one chip. This type of configuration is inexpensive, and normally built on low-powered hardware. This type of configuration is common in devices such as sensors.

Dual IC over HCI is a configuration where the host and the controller are separated on two different integrated circuits, and communication takes place over the standard HCI interface. HCI is defined as part of the Bluetooth specification and decouples the communication between host and controller so that different controllers can be paired with different hosts without modification. This provides flexibility in component selection and allows the host to be tied a more resource-intensive CPU. This is a configuration common in mobile phones and tablets.

Dual IC connectivity device is a configuration where the application communicates with the host over a proprietary protocol. This tends to be used in cases where BLE needs to be added to a device, but the designers want to do this without disrupting the architecture of the device.

2.3 Application Data Storage

BLE is a protocol that is primarily concerned with the exchange of application data. This data resides in the host and is either shared via a BLE channel, over a user interface or transferred via some other network connection to another process interface. This chapter describes how data is stored in the host.

BLE data are stored as “attributes” on a BLE device. Attributes are defined in the ATT layer, and they are comprised of four different components: an attribute handle, an attribute UUID, the attribute value and a set of attribute permissions.

- Attribute handle: This is a two octet, unique ID for an attribute on a BLE device.

- Attribute type: This is a 2 or 16 octet (unique universal identifier) UUID that is specified by a definition of the attribute by a higher level protocol or, in special cases, by the BLE specification itself.
- Value: This is the value of the attribute.
- Permissions: Specifies the permissions for the attribute: broadcast, read, write without response, write, notify, indicate, authenticated signed writes and extended properties.

Attributes may contain the values that are communicated over-the-air with peer devices. They may also contain values that are useful only for the local application.

The GATT layer provides more meaning behind the different kin,ds of attributes that reside on a device that contains attributes, i.e. a server. There three major types of attributes: services and characteristics. GATT services can have three major components.

- Service declaration- an attribute that indicates the kind of service.
- Includes (optional): an attribute with a UUID that references other supporting services on the server.
- Characteristics

Characteristics are defined by a variety of attributes: characteristic declarations, characteristic values and, optionally, characteristic descriptors.

- Characteristic declaration – attribute that indicates the type of the characteristic. A characteristic declaration contains:
  - the handle of the characteristic value,
  - the UUID that describes the characteristic value,
  - charactertistic permissions flags that indicate how the characteristic may be used: broadcast, read, write without response, write, notify, indicate, authenticated signed writes, extended properties.

- Characteristic value – attribute that contains the value of the characteristic.
- Characteristic descriptors (optional) – attribute that specifies further permissions and/or formatting requirements for the characteristic value.

In figure 5 there is an example of an instance of the generic access service and its associated characteristics enumerated with a script based on the pygattlib Python library.

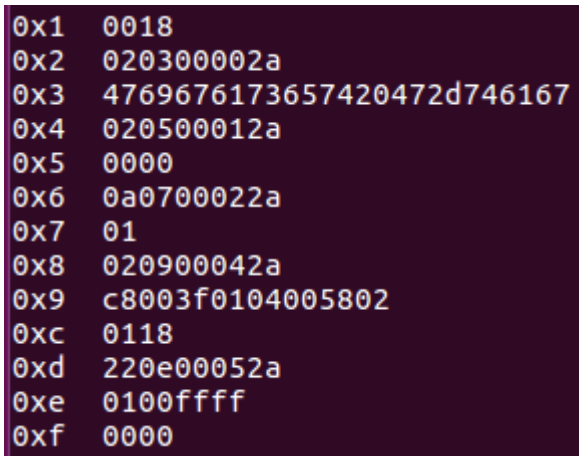

Figure 5: Example of a BLE service enumerated using a pygatt script

In the left column are the handles which provide an index for looking up the different attributes, and in the right column the contents of the attributes are represented in little endian format. The following list provides some examples of the service contents.

- Handle 0x1 – This attribute contains a UUID of 1800. This indicates the start of a generic access service.
- Handle 0x6 – This attribute is a characteristic declaration (or descriptor) and contains 3 parts.
    - 022a – At the end of the characteristic declaration is a UUID of 2a02. This UUID represents the peripheral privacy value attribute.

  - 0700 – In the middle of the characteristic declaration is the value 0007. This value represents the handle value of the characteristic being defined by the declaration.
  - 0a – At the beginning of the hex value is 0a. This value represents the permissions for the value stored at the handle 0x7. In this case there are read and write permissions.

- Handle 0x7 – This is the actual value of the characteristic.

There are two types of services: primary and secondary. Primary services are services that are exposed to peer devices. Secondary services are typically services that are used by other other services.

GATT services are typically specified outside of the BLE specification. For compatibility between devices, there are publicly available service definitions maintained under https://www.bluetooth.com/specifications/adopted-specifications.

The Phone Alert Status Service (PASS), for example, specifies a primary service for the exchange of data concerning the ringer state of a mobile phone and allows the update of the ringer. It contains no included services, but it specifies three characteristics as represented in figure 6.

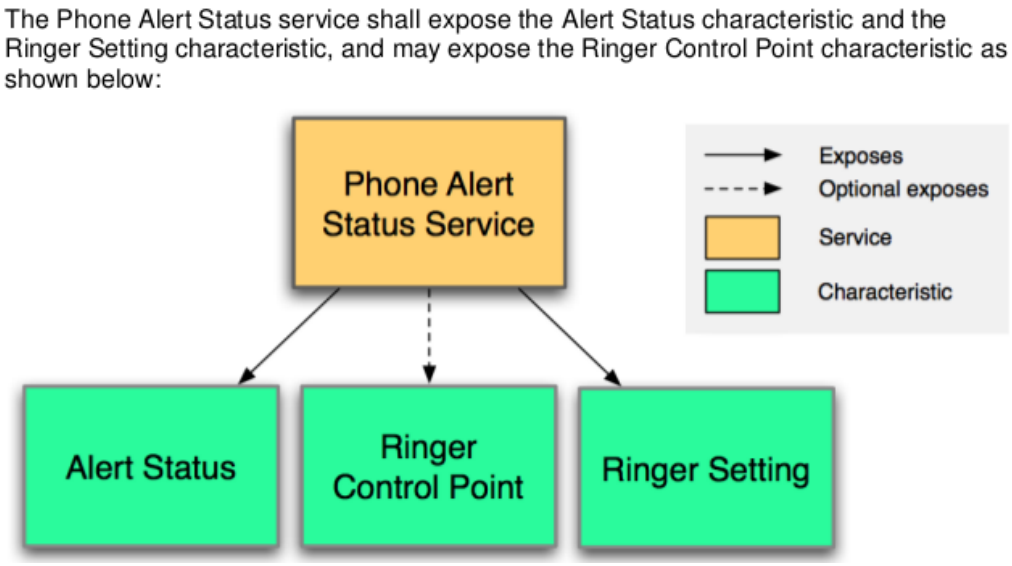

Figure 6: PASS Characteristics [KOY15]

As can be seen in figure 6, two of the characteristics, alert status and ringer setting, are required. Ringer control point is optional. For each characteristic, the PASS specification also defines the characteristic property and the characteristic descriptor attributes.

To gather the permissions for this service and its accompanying characteristics, it is necessary to review the PASS specification and its accompanying document, the “Phone Alert Status Profile” (PASP). In the PASS specification, the alert status and ringer setting are defined as read only. Ringer control point is defined as writable. In the PASP document, authentication is required to access service and its characteristics. Authorization is not specified.

The service specifications stop just short of providing all of the information needed for implementing the attributes that make up the service and its characteristics. As specified above, attributes must have a type which is a UUID. For standardized service and characteristic specifications, the UUIDs are fixed. An up-to-date listing can be found under the following web addresses.

- Services: https://www.bluetooth.com/specifications/gatt/services
- Characteristics: https://www.bluetooth.com/specifications/gatt/characteristics

Appendix 7.1 provides a list of common service and characteristic UUIDs and a description of each.

For companies that define their own, proprietary services, they can define their own UUIDs for their services and their characteristics. These new UUIDs must be requested from the ITU [ITU16].

2.4 BLE Communication

One of the key functionalities of BLE is the definition of how data is communicated over-the-air. This requires the collaboration of all layers of the stack.

Data transmission and reception occurs at the physical layer. BLE operates in the 2.4000 GHz-2.4835GHz frequency ranges. BLE uses Gaussian shift keying modulation (GFSK) at a 1MBit/second data rate. To withstand interference, BLE supports frequency hopping over 40 physical channels.

Information is exchanged between two devices in the form of “air interface packets”. The link layer defines the air interface packet structure which is represented in figure 7.

Figure 7: Link Layer Packet Format [BLE-LL,38]

These packets are exchanged over two type of channels: advertising or data. Each type of channel has its own dedicated RF channels. Advertising channels are RF channels 0, 12 and 39. The data channels are RF channels 1-11 and 13-38. These channels have “channel indexes” which often show up in literature. The data channel index range goes from 0-37. The advertising channel range is from 37-39.

The air interface packet contains four components. These components are populated based on which channel they are being sent on.

- Preamble – This is a fixed series of 1s and 0s that indicates to a BLE device that a message is starting.
- Access Address – This is a field that can take two types of values.
    - On the advertising channels, the address is a fixed value of 0x8E89BED6.
    - On data channels the address represents a connection identifier and is generated at the point that a connection is established.

- PDU – This is a protocol data unit. This is the payload of the packet. There are two types of PDUs supported by the linked layer: LL data PDUs and LL control PDUs.
    - LL data PDUs contain data from attributes stored in GATT services.
    - LL control PDUs contain data to control the data flow between two peer devices.
- CRC – This is a cyclical redundancy check value. The CRC polynomial is $x^24+x^9+x^6+x^4+x^3+x+1$.
    - The CRC's shift register is 0x555555 when a device is communicating on advertising channels.
    - This value is set to a different value during a CONNECT_REQ LL control PDU when two devices enter into a connection and communicate over data channels.

The characteristics and communication exchanges that occur over the advertising and data channels are quite different. The next two major sections will outline the kinds of communication that can occur over these channels.

2.4.1 Advertising Channel Communication

Advertising channels support three different kinds of activities: advertising, scanning and initiating.

- Advertising is where a peer device called an "advertiser" transmits an air interface packet without an established connection to another peer. These packets can either be broadcast messages meant for any listener, or they can be directed advertisements meant for specific peer devices. This packet will possibly contain one or two different types of data:
    - data that can be received and processed by another peer, and/or
    - an indication that a connection and/or a scan can be initiated.

- Scanning is where a peer device, called a scanner, listens for advertisement information. A scanner can scan actively or passively.
    - Passive scanning is when the device listens for advertising packets, but it does not respond to those packets in any way.
    - Active scanning is when the device listens for advertising packets, and when it receives advertising packets, it returns a scan request for more information. An advertiser can only respond to one scan request.
- Initiating is where a peer device has received an advertisement that indicates a connection can be established, and it sends a request to the advertiser to establish a connection. The device that initiates a connection is called the "initiator".

The remainder of this subsection will focus on describing the different formats of advertising channel air interface packets and the related BLE device configurations required to support these activities.

As outlined in section 2.4, the air interface packet is composed of four fields: preamble, access address, PDU and CRC. The access address of all air interface packets on the advertising channel for all activities is 0x8E89BED6. All advertising PDUs of air interface packets have the same general structure which is represented in figure 8. The activity that is being executed over the advertising channel is determined by the advertising PDU itself.

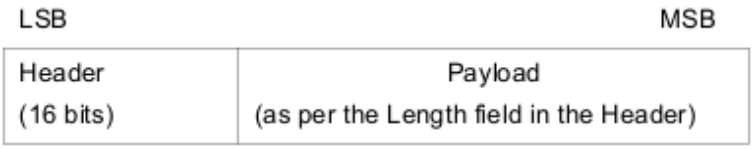

Figure 8: Advertising PDU Structure [BLE-LL, 39]

The advertising PDU has a header and a payload. The header has a fixed format as represented in Figure 9.

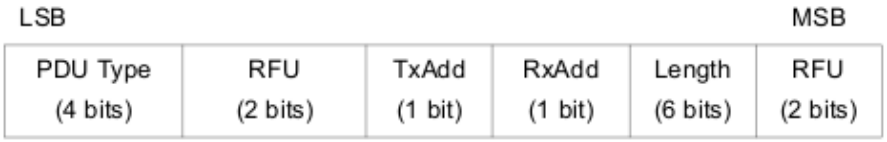

Figure 9: Advertising PDU Header Structure [BLE-LL, 40]

As can be seen in the figure above, the header has six discrete sections. These are described in the table below.

PDU Component	**Description**
PDU Type	This a bit description that indicates what sort of advertising PDU is being transmitted.
RFU	Reserved for future use.
TxAdd	This value of TxAdd is relative to the PDU type. It usually refers to an address field payload and differentiates between a public or a random address. • 0=public • 1=random
RxAdd	This value of RxAdd is relative to the PDU type. It usually refers to an address field payload and differentiates between a public or a random address. • 0=public • 1=random
Length	This is the number of octets contained in the payload.
RFU	Reserved for future use.

[BLE-LL, 40]

There are only a handful of PDU types. These are summarized in the next table. Appendix 7.2.6 provides more detail for the specific contents of the each PDU type payload.

PDU Type	**PDU Name**	**Description**
0000	ADV_IND	• Advertiser allows connections. • Payload contains advertiser's device address and advertising data.
0001	ADV_DIRECT_IND	• Advertiser allows connection connections from a specific peer. • Payload contains advertiser's device address and the desired initiator's device address.
0010	ADV_NONCONN_IND	• Advertiser does not allow connections. • Payload contains the advertiser's device address and advertising data.
0101	CONNECT_REQ	• This is an initiator's request to establish a connection with the advertising device. • Payload contains initiator's device address, the advertiser's device address and logical link connection information.
0110	ADV_SCAN_IND	• Advertiser allows scans. • Payload contains advertiser's device address and advertising data.
0011	SCAN_REQ	• This is a scanner's request to get more information after receiving an advertisement. • Payload contains scanner's device address and the advertiser's device address.
0100	SCAN_RSP	• This is an advertiser's response to a SCAN_REQ. • Payload contains advertiser's device address and advertising data (the scan response data).

[BLE-LL, 41-45]

As can be seen in each payload description, there is at least one device address that is specified. The device addresses, as discussed earlier, can be either public or private. These addresses serve four different purposes:

- to allow two peer devices to identify each other for connection purposes
- to allow a peer to direct advertising events to specific peer devices
- to allow a scanner to request scan data from a specific peer
- to and to allow whitelist filtering in the link layer.

Whitelists are lists are lists of target device addresses that are stored in the link layer of the BLE device. They are populated by the host, and they provide a way for the device to identify which devices can perform certain events over the advertising channel. Whitelists are used in conjunction with policies. Policies specify the events that the peer devices in the whitelist may perform. There are three possible whitelist policies:

- Advertiser policy: relevant to advertiser role. This specifies which devices may initiate a connection with the advertiser.
- Scanner policy: relevant to the scanner role. This specifies which advertisers' packets may be processed by the scanner.
- Initiator policy: relevant to the the initiate role. This specifies which advertisers' connectible advertisements can be processed to establish a connection.

In addition to the devices addresses contained in the payload, there can also be advertising data present in the ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND and the ADV_SCAN_IND PDUs. Advertising data comes from the host, and it is defined by the GAP layer. A payload can contain more than one piece of information in the advertising data, although it is limited to a total number of 31 octets of total data. An advertising data field is represented in figure 10.

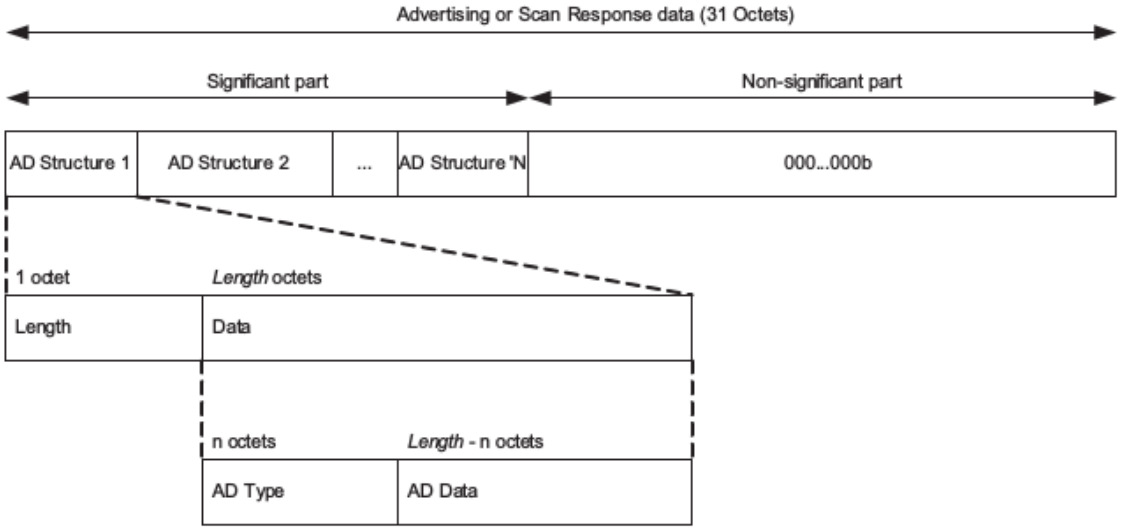

Figure 10: Structure of Advertising Data [BLE-GAP, 389]

As show in figure 10, there are two major parts to advertising data: a significant part and an insignificant part. Advertising data is a fixed length of 31 octets. The significant part of advertising data is the part that is populated with actual data. The insignificant part is populated with zeros.

The significant part contains “AD structures”. These are simply length and data field pairs. The length field specifies how many octets the data in the next bits take up, and the data follows the length declaration directly. The data is then broken down into an AD Type field and an AD Data field pair. The AD Type field indicates the type of data that is contained in the field. The AD Data is the actual data value.

Advertising data has a core set of possible values defined in a supplement to the Bluetooth specification. Additional types can be defined outside of the specification in a different profile. The AD Type field is a hex ID of one octet in length. An up-to-date list of these values can be found under the following web address:

- https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

A list of the core advertising data types can be found in appendix 7.2.8 .

In addition to providing the format of the advertising data, the GAP layer also introduces new vocabulary to represent the roles played by devices in the link layer advertising, scanning and initiating states. These new roles are broadcaster and observer. Broadcaster roughly corresponds to advertiser and observer corresponds to scanner. A device that advertises information is said to be in “broadcast mode” and a device that actively or passively scans for data is said to be performing the “observation procedure”.

In addition to the broadcast mode and observation procedure, there are also “discovery” modes and procedures. These modes and procedures have to do with the visibility of one peer device to other peer devices. A BLE device normally only supports a subset of these modes and procedures.

To describe these modes and procedures, the specification introduces two additional roles: “peripheral” and “central”. On the advertising channel, peripherals are devices that can accept connections, and they typically correspond to the advertiser in the link layer. On the advertising channel, centrals are devices that scan for peer device advertisements, issue scan requests or initiate connections.

The following table summarizes the discovery modes and procedures that are available on the advertising channel.

Mode / Procedure	**Description**
Non-discoverable-mode	• Peripheral sends: non-connectible advertising events, scannable undirected advertising events or no advertising events.
Limited-discoverable-mode	• Peripheral sends: non-connectible advertising events, scannable undirected advertising events or connectible undirected advertising events. • Peripheral is only available for a limited time. • Receives requests from central performing: limited-discovery-procedure or general discovery procedure.
General-	• Peripheral sends: non-connectible advertising events,

discoverable-mode	scannable undirected advertising events or connectible undirected advertising events. • Peripheral generally available for an extended period of time. • Receives requests from central performing: general discovery procedure.
Limited-discovery-procedure	• Central can receive advertising and scan response data only from peripherals in the limited-discoverable-mode.
General-discovery-procedure	• Centrals can receive advertising and scan response data from peripherals in either limited-discoverable-mode or general-discoverable-mode.

[BLE-GAP, 350]

Once an peripheral has been discovered by a central, the central must make the decision whether or not to initiate a connection. As with discovery, there are set of “connection” modes and procedures defined in the GAP layer to support connection initiation. The following table provides a summary of the connection modes and procedures.

Mode / Procedure	**Description**
Non-Connectible Mode	• A peripheral in this mode will not accept connections from peer devices. Peripheral in this mode may send non-connectible undirected advertising events or scannable undirected advertising events.
Directed Connectible Mode	• A peripheral in this mode accepts connection requests from a known central. • To initiate a connection, a central will perform either the auto connection establishment procedure or the general connection establishment procedure.
Undirected Connectible	• A peripheral in this mode accepts connection requests from any central.

Mode / Procedure	**Description**
Mode	• To initiate a connection, a central will perform either the auto connection establishment procedure or the general connection establishment procedure.
Auto connection Establishment Procedure	• The host of the central configures the link layer to automatically establish a connection to a specific set of peripherals specified in the link layer whitelist which are matched with the initiator policy.
General Connection Establishment Procedure	• The central establishes a connection to a specific set of peripherals specified in the link layer whitelist. In this case, the initiation of the connection originates in the host.
Selective Connection Establishment Procedure	• The central establishes a connection with a specific set of peripherals specified in the link layer white list. In this case, the initiation of the connection originates in the host, and the host specifies the connection settings.
Direct Connection Establishment Procedure	• The central establishes a connection to a specific peripheral. The host determines the peripheral address and ignores the whitelist.

[BLE-GAP, 357]

Connection procedures are relatively simple, and as can be seen in the table above, vary in only minimal ways. Figure 11 shows an example communication flow where a connection procedure is initiated, and a connection is established.

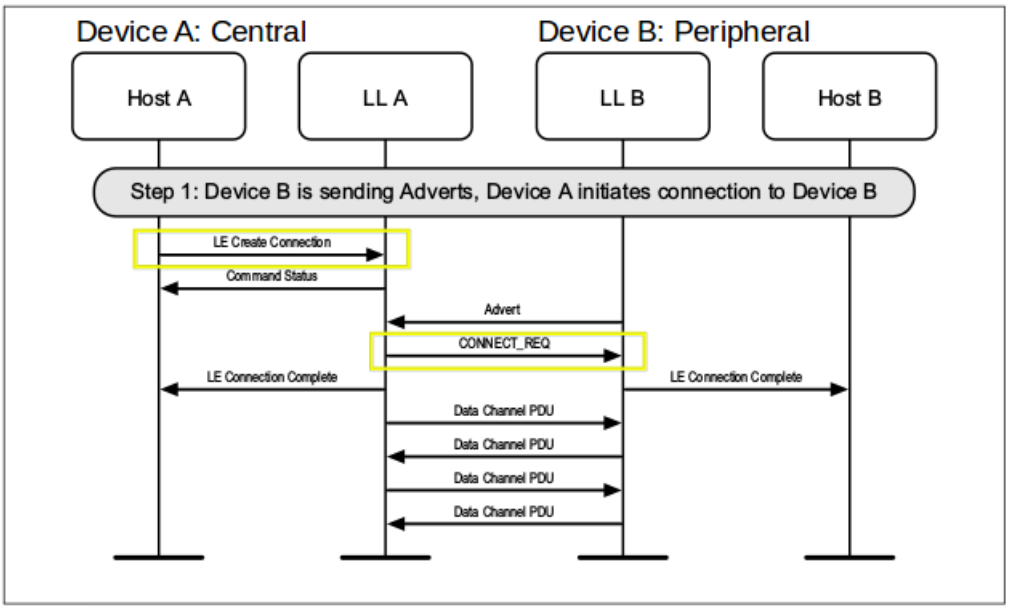

Figure 11: Example of a connection procedure [BLE-LL, 144]

The connection procedure components are highlighted yellow. They consist of the host's interaction with the link layer to prepare for the appropriate type of connections, the central link layer's processing of an advertisement with connection parameters and the transmission of an air interface packet with a CONNECT_REQ PDU.

Once this air interface packet has been issued, the central goes from the initiated state to the the connected state. Upon receipt of the air interface packet, the peripheral processes the connection parameters contained in the PDU, and moves from the advertising state to the connected state. All communication between the two peers then occurs over the data channels.

2.4.2 Data Channel Communication

Once a connection has been established between two peer devices, all communication follows over the data channels. The conditions for this communication are established by the CONN_REQ PDU on the advertising channel. Once a connection has been established, the two devices move to the link layer state "connected". On the link layer, the device that initiated the connection is called the "master" and the device that accepted the connection is called the "slave".

Ten values are sent via the CONN_REQ PDU that define the connection parameters. These values define timing for communication and the channels that will be used. Six of those values bear mentioning at this point to help with the understanding of how the data channel functions and are described in the table below.

Para-meter	**Description**
AA	Access address. This is the address that the slave will use to identify the master on the link.
CRCinit	This is the new initialization value that will be used to calculate the CRC of the air interface packet.
Timeout	This is the length of time that the master will wait for communication from the slave before the connection is considered to be terminated.
Channel map	This provides a list of the valid data channels for the master on that link.
Hop	The link layer uses frequency hopping to help prevent communication interference. Once a packet has been sent over a data channel, the communicating device will move to the next channel to send the next packet. The hop, or hopIncrement value provides the slave with a value needed to calculate the next data channel that will be used for communication. To calculate the value, the following formula is used: nextChannelIndex = (usedChannelIndex + hopIncrement) % 37 If there are fewer than the full 37 channels mapped in the channel map, then an extra piece of logic is needed to determine the next channel index.

Para-meter	Description
	remappedNextChannelIndex = nextChannelIndex % countOfMappedChannels [BLE-LL, 82][2]
Connection Interval	This value helps determines the timing of how often the two devices should check the next channel for communication.

[BLE-LL, 44-45]

On the data channel there are two types of communication that can occur: link layer control communication and data communication. Link layer control communication allows the exchange of requests and commands concerning the connection itself. Over the air packets that perform these activities contain PDUs called “LL control PDUs”. Data communication describes the manner in which information in exchanged between two peers. Over the air packets that provide data exchange contain PDUs called “LL data PDUs”.

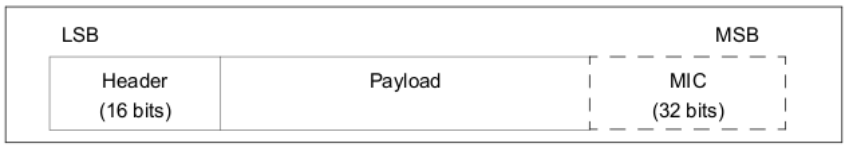

Figure 12: Logical representation of a data channel PDU [BLE-LL, 46]

The specification represents a PDU on the data channel as represented in figure 12. The PDU is made up of three fields: a header, the payload and a message integrity check (MIC). The following table provides an overview of these fields.

2 The variable names in the specification differ from the variable names used in this document. The names have been changed here to provide more transparency about their roles in the calculations. The variables names can be mapped in the following way:
nextChannelIndex = unmappedChannel
usedChannelIndex = lastUnmappedChannel
hopIncrement = hopIncrement
remappedNextChannelIndex = remappingIndex
countOfMappedChannels = numUsedChannels

Field	Description
Header	The header contains five fields: • LLID (2 bits) : ◦ 01= Continuation of an LL data PDU ◦ 10= Start of an LL data PDU ◦ 11= LL control PDU • NESN (1 bit): Next expected sequence number • SN (1 bit): Sequence number • MD (1 bit): More data flag • RFU (3 bits): Reserved for future use • Length (8 bits): Length of the payload and MIC in octets.
Payload	Varies in length and is dependent on the type of PDU.
MIC	Message integrity check: The message integrity check is optional and dependent on whether or not integrity is a security requirement of the application.

[BLE-LL, 46]

2.4.2.1 LL Control PDUs

An LL control PD contains two main fields. These fields are represented in figure 13. In total there are twenty-eight separate LL control PDUs defined. Each one has an opcode of one octet in length. The data contained is specific to the functionality being requested or commanded in the PDU.

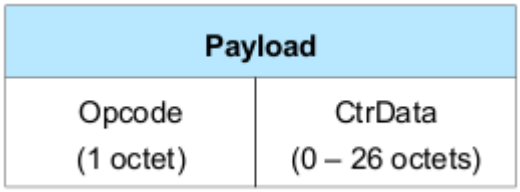

Figure 13: Logical representation of LL control PDU [BLE-LL, 48]

The following table provides a summary of the LL control PDUs defined in the specification. See appendix 7.3.1.2 for additional technical detail about the PDUs.

OP Code	**LL Control PDU Name**	**Description**	**Sender**
00	LL_CONNECTION_UPDATE_REQ	Requests update of WinSize, WinOffset, Interval, latency, timeout and instant values of connection.	master
0F	LL_CONNECTION_PARAM_REQ	This is a request to change connection parameters. Information is contained in the PDU: interval_min, interval_max, latency, timeout, preferredPeriodicity, referencConnEventCount, Offset0, Offset1, Offset2, Offset3, Offset4 and Offset5.	master
10	LL_CONNECTION_PARAM_RSP	This response to the preceding PDU and defines the connection parameters.	slave
01	LL_CHANNEL_MAP_REQ	Requests an update to the list of data channels to be used for a connection.	master
02	LL_TERMINATE_IND	This is an indicator that the connection is about to be ended, and it provides an error code that specifies the reason.	master or slave
03	LL_ENC_REQ	This requests that encryption material be established with the supplied Rand, EDIV, SKDm and IVm values from the master.	master
04	LL_ENC_RSP	This is the response to the preceding PDU from the slave with its respective SKD and IV values.	slave
05	LL_START_ENC_REQ	This is a request that further communication be encrypted with established cryptographic material.	master
06	LL_START_ENC_RSP	This is a response to the preceding PDU.	slave
07	LL_UNKNOW	This is a response that indicates the	master or

OP Code	**LL Control PDU Name**	**Description**	**Sender**
	N_RSP	device received a PDU type that it did not support. The unknown type OP code is returned.	slave
08	LL_FEATURE_REQ	This provides the set of features supported by the master.	master
0E	LL_SLAVE_FEATURE_REQ	Contains the set of supported features in the slave's link layer.	slave
09	LL_FEATURE_RSP	This is the slave's response with its list of supported features.	master or slave
0A	LL_PAUSE_ENC_REQ	This is a request to pause encrypted communication. This is usually to reestablish cryptographic material.	master
0B	LL_PAUSE_ENC_RSP	This is the response to the preceding PDU.	slave and master
0C	LL_VERSION_IND	This message contains the version of the Bluetooth controller, the company identifier of the manufacturer and the revision number of the controller.	master or slave
0D	LL_REJECT_IND	In the case that a request must be rejected, this PDU will be returned with an error code that indicates the reason for rejection.	master or slave
11	LL_REJECT_IND_EXT	This is a PDU that is sent when an operation is rejected. The OP code of the PDU being rejected and an error code indicating the reason are returned.	master or slave
12	LL_PING_REQ	This is a PDU that is sent to check the presence of a communicating peer.	master or slave
13	LL_PING_RSP	This is the response to the preceding PDU.	master or slave
14	LL_LENGTH_	This requests a change in the length of	master or

OP Code	LL Control PDU Name	Description	Sender
	REQ	transmission and reception time and/or the length of an air to interface packet.	slave
15	LL_LENGTH_ RSP	This is a response to the preceding packet with the values for the connection.	master or slave

[BLE-LL, 48-50]

These operations are the building blocks for a series of link layer procedures. The procedures describe how the LL control PDUs are used. There are 12 total procedures:

- Connection update procedure / Connection parameters request procedure:
  - Allows a peer device to request updates to connection parameters.
  - PDUs used: LL_CONNECTION_UPDATE_REQ or LL_CONNECTION_PARAM_REQ / LL_CONNECTION_PARAM_RSP
- Channel map update procedure:
  - Updates the channel map.
  - PDU used: LL_CHANNEL_MAP_REQ
- Encryption procedure:
  - Supports the exchange of cryptographic material and the initiation of encryption on the link.
  - PDUs used: LL_ENC_REQ / LL_ENC_RSP and LL_START_ENC_REQ / LL_START_ENC_RSP
- Encryption pause procedure:

  - Allows encryption to pause for the update of key material. During this time, no data should be exchanged between the two communicating devices.
  - PDUs used: LL_PAUSE_ENC_REQ (sent encrypted by master) / LL_PAUSE_ENC_RSP (sent encrypted by slave) / LL_PAUSE_ENC_RSP (sent unencrypted by master)

- Features exchange procedure:
  - Allows link layers to provide information about supported features.
  - PDUs used: LL_FEATURE_REQ (from master) / LL_FEATURE_RSP or LL_SLAVE_FEATURE_REQ (from slave) / LL_FEATURE_RSP

- Version exchange procedure:
  - Allows a link layer to provide information about device version information.
  - PDUs used: LL_VERSION_IND

- Termination procedure:
  - Facilitates the communication that a link is being terminated.
  - PDUs used: LL_TERMINATION_IND

- LE ping procedure:
  - Allows one peer device to check to see if the link layer of its peer device is still present.
  - PDUs used: LL_PING_REQ / LL_PING_RSP

- Data length update:
  - Allows the length of time that transmission of a signal can occur and/or the length of the transmitted signal to be updated.
  - PDUs used: LL_LENGTH_REQ / LL_LENGTH_RSP

2.4.2.2 LL Data PDUs

Data exchange is defined in the ATT, GATT specifications and mentioned in the link layer specification. ATT and GATT specifications outline the operations that can be performed on the data. At these layers, two new roles are introduced: client and server. The client is the device that makes requests of data. These requests can be read or write requests. The server is the device the provides the data store.

Unlike other PDUs, the the LL Data PDU structure is not defined in the link layer specification. It is instead defined in the ATT specification. The LL data PDU is called the "attribute PDU" at the ATT layer, and it is broken down into three fields. Figure 14 defines its characteristics.

Name	Size (octets)	Description
Attribute Opcode	1	The attribute PDU operation code bit7: Authentication Signature Flag bit6: Command Flag bit5-0: Method
Attribute Parameters	0 to (ATT_MTU - X)	The attribute PDU parameters X = 1 if Authentication Signature Flag of the Attribute Opcode is 0 X = 13 if Authentication Signature Flag of the Attribute Opcode is 1
Authentication Signature	0 or 12	Optional authentication signature for the Attribute Opcode and Attribute Parameters

Figure 14: Attribute PDU Format [BLE-ATT, 478]

As can be seen in the PDU, there is an Opcode with a set of values. Bits 0-5 correspond to the different attribute PDU types supported by Bluetooth LE. Bit 6 indicates that the PDU is a command, and bit seven indicates whether or not the authentication signature is required. The data PDU types specified in the ATT layer are summarized in the following table.

Opcode	**Attribute PDU Name**	**Parameters**	**Description**
0x01	Error Response	Erroneous opcode, erroneous attribute handle, error code	Response to a request that cannot be performed. Includes a reason.
0x02	Exchange	Client Rx MTU	Informs the server of the

Opcode	**Attribute PDU Name**	**Parameters**	**Description**
	MTU Request		client's maximum transmission unit size.
0x03	Exchange MTU Response	Server Rx MTU	Response to Exchange MTU request and informs client of MTU that will be used.
0x04	Find Information Request	Starting handle, ending handle, UUID	Request from client to server to gain mapping of attribute handles for specified attribute types.
0x05	Find Information Response	Format, information data	Response to find information request with handle-UUID pairs.
0x06	Find by Type Value Request	Starting handle, ending handle, attribute type, attribute value	Request to the server from the client for handles that have specified attribute types with 16-bit UUIDs.
0x07	Find by Type Value Response	Handles of information list	Response to Find by Type Value Request and contains a list of corresponding handles.
0x08	Read by Type Request	Starting handle, ending handle, UUID	Request to the server from the client for handles that have specified attribute types.
0x09	Read by Type Response	Length, attribute data list	Response to Read by Type Request and contains handle-value pairs.
0x0A	Read Request	Attribute handle	Request from client to server for the value of an attribute for a specific handle.
0x0B	Read Response	Attribute value	Response to Read Request and contains the value of the associated handle.

Opcode	**Attribute PDU Name**	**Parameters**	**Description**
0x0C	Read Blob Request	Attribute handle, value offset	Request from client to server for part of a value specified by the handle and the octet offset.
0x0D	Read Blob Response	Part attribute value	Response to the Read Blob Request with the value associated with the requested handle starting from the provided offset.
0x0E	Read Multiple Request	Set of handles	Request from client to server for the values of more than one handle.
0x0F	Read Multiple Response	Set of values	Response to Read Multiple Request. Values are concatenated in the order requested.
0x10	Read by Group Type Request	Start handle, ending handle, UUID	Request from client to server where values are request for a range of handles where for a specified type.
0x11	Read by Group Type Response	Length, attribute data list	Response to Read by Group Type Request. Returns a list of values in the order of the handles. Values will be contained in their own octets.
0x12	Write Request	Attribute handle, attribute value	This is a request from a client to the server to update a value based on the provided handle.
0x13	Write response	--	This is a response to the Write Request indicating that the write was successful.
0x52	Write	Attribute handle,	This is a command from a

Opcode	Attribute PDU Name	Parameters	Description
	Command	attribute value	client to a server to update a value based on the attribute handle. No response from server needed.
0x16	Prepare Write Request	Attribute handle, value offset, part attribute value	This is a request from the client to the server to queue a value to be written on the server at the specified handle.
0x17	Prepare Write Response	Attribute handle, value offset, part attribute value	This is a response to the Prepare Write Request indicating that the submitted value has been queued for the provided handle.
0x18	Execute Write Request	flags	This is a request from the client to the server to perform one of two actions with queued values: 00-cancel all queued writes or 01- write all pending values.
0x19	Execute Write Response	-	This is a response to the Execute Write Request signifying the successful completion of the request by the server.
0x1B	Handle Value Notification	Attribute handle, attribute value	The server transmits a value (unsolicited) to the client. If the value exceeds MTU-3 octets in length, the client will need to send a Read Blob Request to get the whole value.

Opcode	Attribute PDU Name	Parameters	Description
0x1D	Handle Value Indication	Attribute opcode, attribute handle, attribute value	The server transmits a value (unsolicited) to the client. If the value exceeds MTU-3 octets in length, the client will need to send a Read Blob Request to get the whole value.
0x1E	Handle value confirmation	--	This is a confirmation of receipt of the Handle Value Indication.
0xD2	Signed write command	Attribute handle, attribute value, authentication signature	This is a command from the client to the server to write a value to the server based on the attribute handle only after verifying the provided signature. No response from server is required.

[BLE-ATT, 481-482]

The PDU types can be categorized into one of 6 types. These are:

- Requests and responses
    - Client sends a request for some operation.
    - Server responds.
- Commands
    - One-way communication where client sends unsolicited command to server.
- Notifications

    - One-way communication where the server sends an unsolicited message to the client.

- Indications and confirmations
    - Server sends an unsolicited message to the client with the expectation of confirmation.
    - The client sends a confirmation to the server.

As can be seen in the list, there are two types of PDUs that are paired together: request / response and indications / confirmations. These types of PDUs must be executed as transactions. This means, when a server issues a request or an indication, it must wait for the answer from the client. A server may have only one request and one indication active at any one time. It must wait for the answering attribute PDU or wait for a timeout before it can issue another one of the same type. For example, if a client sends a request to a server, it must wait for the server's response before it can issue any further requests. Likewise, if a server issues an indication to a client, it must wait for the client's confirmation before it can issue any further notifications.

Exactly how these PDUs are used is defined at the GATT layer. The GATT layer provides a set of 11 data communication "features" that are dependent on the attribute PDUs from the ATT layer. Each of these GATT features are comprised of a set of sub-procedures that define how the ATT PDUs are utilized during data exchange between client and a server. The following table provides a summary of the GATT features, their sub-procedures, and the attribute PDUs related to those sub-procedures.

Feature	**Sub-Procedure**	**Attribute PDUs**
Server Configuration	Exchange MTU: Two-way exchange to identify the largest attribute PDU supported by both client and server.	Exchange MTU Request Exchange MTU Response Error Response
Primary	Discover All Primary Services:	Read by Group Type

Feature	**Sub-Procedure**	**Attribute PDUs**
Service Discovery	Used by client to request primary services available in a handle range.	Request Read by Group Type Response Error Response
	Discover Primary Services by Service UUID: Used by client to find a primary service on the server when only the UUID (type) is known.	Find by Type Value Request Find by Type Value Response Error Response
Relationship Discovery	Find Included Services: Used by client to find included services inside of a service declaration within a handle range.	Read by Type Request Read by Type Response Error Response
Characteristic Discovery	Discover All Characteristics of a Service: Used by client to find all characteristic declarations of a service within a handle range.	Read by TypeRequest Read by Type Response Error Response
	Discover Characteristic by UUID: Used by client to look up characteristics of a service when only the service hande range is known and the characteristic UUID.	Read by TypeRequest Read by Type Response Error Response
Characteristic Descriptor Discovery	Discover All Characteristic Descriptors: Used by client to find all of the characteristic descriptor's handles and types (UUID) within a given handle range.	Find Information Request Find Information Response Error Response

Feature	**Sub-Procedure**	**Attribute PDUs**
Characteristic Value Read	Read Characteristic Value: Used by client to read a characteristic value from a service by the handle.	Read Request Read Response Error Response
	Read Using Characteristic UUID: Used by client to read a characteristic value from a service when only the UUID is known.	Read by TypeRequest Read by Type Response Error Response
	Read Long Characteristic Value: Used by client to request a value by handle, and the length of the value will be longer than what can be returned in the Read_Response PDU.	Read Blob Request Read Blob Response Error Response
	Read Multiple Characteristic Values: Used by client to request multiple characteristic values. The client provides a set of handles.	Read Multiple Request Read Multiple Response Error Response
Characteristic Value Write	Write Without Response: Used by client to update characteristic value by handle.	Write Command
	Signed Write Without Response: Used by client to update a characteristic value by handle where the link is not encrypted.	Signed Write Command
	Write Characteristic Value: Used by client to update a characteristic value by handle.	Write Request Write Response Error Response

Feature	Sub-Procedure	Attribute PDUs
	Server must provide a response.	
	Write Long Characteristic Values: Used by client to update a characteristic value to by handle where the value to be updated is longer than what can be contained in the Write Request PDU. Server must provide responses.	Prepare Write Request Prepare Write Response Execute Write Request Execute Write Response Error Response
	Characteristic Value Reliable Writes: Used by client to update several characteristic values for a service at one time. Values are queued by handle on the server until Execute Write Request is issued.. Server must provide responses.	Prepare Write Request Prepare Write Response Execute Write Request Execute Write Response Error Response
Characteristic Value Notification	Notifications: Server sends a notification of a characteristic value to a client without the client requesting it.	Handle Value Notification
Characteristic Value Indication	Indications: Server sends a characteristic value to a client without a request from the client. The client must confirm receipt of the indication.	Handle Value Indication Handle Value Confirmation
Characteristic Descriptor Value Read	Read Characteristic Descriptors: Used by client to read a characteristic descriptor by handle.	Read Request Read Response Error Response

Feature	Sub-Procedure	Attribute PDUs
	Read Long Characteristic Descriptors: Used by client to read a characteristic descriptor by handle where the value of the characteristic descriptor is longer than what can be contained in the Read Response PDU.	Read Blob Request Read Blob Response Error Response
Characteristic Value Write	Write Characteristic Descriptors: Used by client to update a characteristic descriptor based on the characteristic descriptor's handle. Server must provide a response.	Write Request Write Response Error Response
	Write Long Characteristic Descriptors: Used by client to update a characteristic descriptor based on the characteristic descriptor's handle where the value to be updated is longer than what can be contained in the Write Request PDU. Server must provide responses.	Prepare Write Request Prepare Write Response Execute Write Request Execute Write Response Error Response

[BLE-GATT, 544-571]

These data exposed via operations and features at the BLE ATT and GATT layers are the same data that are processed by the BLE applications that sit on top of the BLE host. In some systems, this data can be forwarded to upstream processes by the BLE application. Access and modification of these data is an ideal attack vector and will be discussed later in chapters 3 and 4.

2.5 Bluetooth LE Security Features

The Bluetooth LE specification explicitly addresses security with the introduction of features such as private address and security modes and procedures. These features provide a foundation for addressing security requirements of over-the-air communication. Only a high level overview of these features will be provided here, and more detail will be provided in later chapters 4 and 5 where the topic of security is more thoroughly analyzed.

2.5.1 Device address privacy

The way that one peer device recognizes another is via the device address in the air interface packet or the AdvA, ScanA or InitA addresses in advertising packet payloads. The addresses can either be public, random static or private (resolvable or non-resolvable) and are broadcast in plaintext in advertisement packets. The Bluetooth LE specification includes mechanisms to mask device addresses for the provision of the security goal of privacy. There are four different forms that an device address can take:

- public (Bluetooth device ID)
- random static
- random private non-resolvable
- random private resolvable

The first two types are not designed to provide privacy, whereas the second two are.

An device address has different formats based on whether is a Bluetooth device ID, a random static address, a random private non-resolvable address or a random private resolvable address. These different formats are represented in the following figure 15.

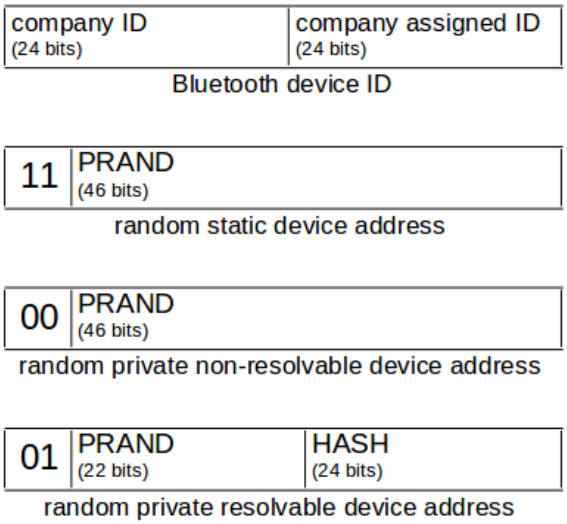

Figure 15: Device Address Formats [BLE-LL, 43-46]

Privacy is addressed primarily in the GAP, SMP and link layer specifications. GAP introduces the concept of private addresses, the link layer specifies the format of private device addresses, and the SMP specification describes the algorithms for the generation of pseudorandom values and the creation of the private device addresses.

The following table provides a summary of the four different types of device addresses.

Address Type	**Description**
public	This is the device's Bluetooth device address. This type of address is trackable because it persists over time across controller resets. This address provides no privacy. Further, the manufacturer of the device can be determined because the first half of the Bluetooth device address indicates the manufacturing company.
random static	When the controller of a device is reset, it pseudorandomly generates a 46 bit number. This number is concatenated to a

Address Type	**Description**
	two bit value “11”, and this is the device’s address. The device uses this address rather than the public device address for communication identification over a link. This type of address is trackable and not private.
random private non-resolvable	The last 46 bits of this type of access address are pseudorandomly generated either each time that a packet is sent or in accordance with a timer that indicates when the address should be renewed. The 46 bit value is appended to a two bit value of “00”, and this is the full address. This type of address can only be used on the advertising channel because it cannot be resolved for two-way communication purposes. The pseudorandomness of this type of address makes the value untrackable by peers.
random private resolvable	This type of access address contains a 22 bit pseudorandomly generated number (PRAND). This number is “hashed” with the assistance of an identity resolution key (IRK). A 24 bit value is extracted from the hash. This value is appended to the PRAND value. These two values together are appended to a two bit value, “01”. The address’s persistence is tied to a timer. When the timer runs out, a new address is generated. This type of address can be used for connection purposes because two peers have the correct information to resolve the generated addresses (PRAND and a previously shared IRK). Assuming the timer for address renewal is set appropriately, the pseudorandomness makes the access address value untrackable because peers that do not have the corresponding IRK cannot determine which address belongs to which device.

2.5.2 Pairing and bonding

Pairing and bonding are functions that attempt to satisfy or support security requirements such as confidentiality, integrity, authenticity and/or privacy security requirements in the over-the-air communication between two peers. The communication path between two devices is often referred to as the "link".

Pairing is the act of peers exchanging information to establish a set of shared cryptographic keys with the goals to encrypt the link to preserve confidentiality of exchanged data. Pairing is not permanent, and if two devices simply pair with one another, the shared keys are not persisted after the connection is lost.

Bonding takes pairing one step further and allows the exchange of cryptographic key material to provide a means by which the integrity and authenticity of the data exchanged over the link can be verified at a later time. Bonding, unlike pairing, is permanent. When two devices are bonded, the relevant necessary key material for confidentiality, integrity and privacy are stored.

Devices may have different security requirements. For example, some devices may require encryption and no verification of authenticity and integrity. Some may require verification of authenticity and integrity.

Further, there may also be differing demands on the strength of assurance of confidentiality, integrity and authenticity. These varying levels of requirements are supported in the types of pairing that are available. In the 4.2 specification, there are four types of pairing which are partially dependent on the available input and output capabilities of the communicating devices.

- Just works: This is a pairing approach where a known, fixed value is used to generate cryptographic material to secure the link. For this type of pairing, no input or output devices are needed.
- PIN/Passkey entry: This is a type of pairing where a number with six positions (PIN/passkey) is displayed by the peripheral. A user must

then enter the value on the central. Once the value has been entered into the central, a handshake procedure takes place over the air interface. It is important to note that the handshake does not exchange the PIN at any time.

- Numeric comparison (LE secure): This is a type of pairing where the two BLE devices use a Diffie-Hellman exchange to establish a connection and to generate a set of values. The generated values are displayed on both devices. If the displayed values on the devices match, the user can indicate that the values do match via the user interface, and the pairing can finish.
- OOB: This is a type of pairing where values are exchanged via other channels other than over LE channels (NFC, for example). The values that are exchanged over the non-LE channels are then used to generate cryptographic materials to secure the link. This does not require input and output devices for use by a user, however it requires some other way of exchanging the OOB values other that BLE itself.

Pairing and bonding are introduced in the GAP specification, and the details surrounding the cryptographic material exchange are explicitly outlined in the SMP specification.

How pairing occurs and which values are stored at bonding are dependent on the specification being used for development. Starting from the Bluetooth 4.2 specification, the recommendation is to use a pairing method called “LE Secure”. Pairing according to methods prior to the Bluetooth 4.2 specification are referred to as “LE Legacy Pairing”.

Legacy pairing uses a proprietary key exchange protocol to establish keys to secure the link. Key material is generated using a 128 bit AES block cipher. LE secure uses an asymmetric Diffie-Hellman protocol based on ECC with a NIST P-256 curve to ensure confidentiality when establishing cryptographic material. AES-MAC is used to generate some key material. The controller is required to support, at the minimum, AES encryption.

Prior to 4.2 cryptographic processing was expected to take place in the controller. As of 4.2, cryptographic processing may take place in the host. This was done to support the need for devices to be upgradeable. Cryptography at the controller level is normally implemented in hardware rather than software. Cryptographic processing in the host tends to be implemented in software, so it makes it simpler to be able to upgrade cryptographic routines as needed.

2.5.3 Security Modes and Procedures

The GAP specification specifies a set of security modes and procedures for Bluetooth LE. The following table provides a summary of each mode and procedure.

Mode / Procedure	**Description**

Security mode 1	Security mode 1 refers to the strength of pairing that a device supports. There are four levels identified:
	• No security: no authentication and no encryption
	• Unauthenticated pairing with encryption:
	  ○ No MITM protection
	  ○ Provided by just works pairing.
	• Authenticated pairing with encryption:
	  ○ Assumption of MITM protection
	  ○ Provided by LE legacy passkey entry and OOB pairing.
	• Authenticated LE secure connections pairing with encryption
	  ○ Assumption of MITM protection
	  ○ Provided by LE secure passkey, numeric comparison and OOB pairing.

Security mode 2	Security mode 2 refers to the authenticity strength of a message integrity check. • Unauthenticated pairing with data signing

Mode / Procedure	Description
	• Authenticated pairing with data signing
Authorization procedure	An authorization procedure usually requires some action on the part of the user through the user interface. The device will ask the user if it is acceptable for some activity to take place based on information exchanged between two peer devices. The user must indicate yes or no (provide authorization) before the activity is allowed to take place.
Authentication procedure	This procedure provides a decision matrix for two devices negotiating security mode 1.
Connection data signing procedure	This procedure specifies that a signature key and signature counter must be established to support security mode 2 in the case that data signature is required by a peer during communication.
Authenticate signed data procedure	This procedure describes how to handle signatures for devices that have already exchanged signature material.
Encryption procedure	This procedures describes that a central is responsible for initiating encryption on a link, while the peripheral may make a request to the central to start encryption initiation.

[BLE-GAP, 348-370]

2.6 Chapter Summary

This chapter attempts to provide a compact summary of the vocabulary and the components that are essential to understand the Bluetooth low energy specification. This type of overview is necessary because, as highlighted in section 2.1, an analyst will need to be able to make sense of data being captured by sniffing devices such as the Texas Instruments CC2540 Dongle. This device sniffs both advertising channels and data channels, and produces output similar to that in figure 16.

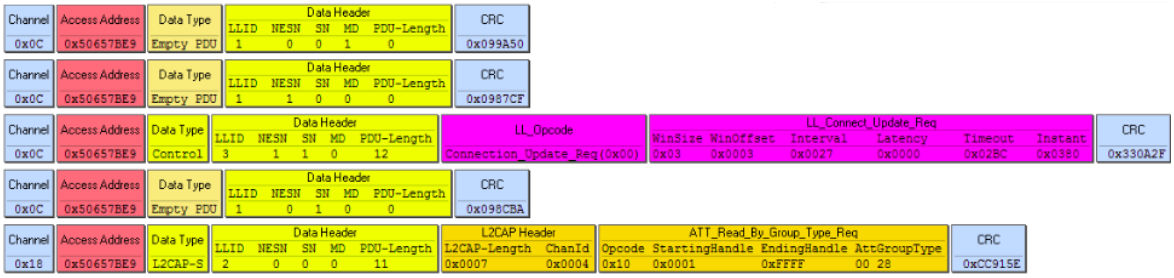

Figure 16: Capture of sniffed BLE packets

From this capture, the following aspects should be clear from the information provided in this chapter.

1. From the access address, 0x50657BE9 and from the channel values, it is clear that the communicating devices are communicating over a link (or a connection).
2. The devices first communicated on channel 0x0C and then hopped to 0x18.
3. Based on the master-slave, master-slave pattern of communication, it can be determined that the first packet is from the master. This is because the third packet is a LL_CONNNECTION_UPDATE_REQ (opcode 0x00) packet, and only a master can send this type of request. So th exchange on channel 0x0C that we have show in figure 16 is:
   a) Master → Slave: empty packet
   b) Slave → Master: empty packet
   c) Master → Slave: LL_CONNECTION_UPDATE_REQUEST
   d) Slave → Master: empty packet
4. When the devices hop to the next channel, we can see that the master, which is always first to send packets on a new channel, issues an ATT_Read_By_Group_Type_Req (opcode 0x10) packet. This is a request that allows the master to request all of the attributes of a specific type between a start handle and an end handle from the GATT server. In this case, the request has the following values:

a) Start Handle: 0x0001

b) End Handle: 0xFFFF

c) Type of attribute being requested: 2800 which is the UUID for primary service declarations

There is even more detail that can be derived from the packet information represented above, but hopefully the reader can begin to understand how having a basic understanding of the specification can be helpful for analysis. The detail in this chapter has been supplemented with a series of more detail which can be found in the appendix. Hopefully, the information captured here and the appendix should provide a solid starting point for analyzing BLE communication.

3 Generic BLE Attack Surface

This chapter maps the full attack surface of a BLE application including non-over-the-air surfaces. The full attack surface, rather that just the over-the-air interface, will be explored for two reasons.

The first reason is so the analyst has a full overview of the scope of the attack surface. When a customer comes with a request for analysis, an analyst must be able to be clear about the scope of testing. Having an overview of the possible attack surfaces provides an important point for discussion and clarification with the customer.

The second reason why the full attack surface will be explored is because attacks on the air interface may not be observable until they can be observed in a downstream process several processing steps from the over-the-air communication exchange. For example, an attacker may forge a message and transmit it to BLE device. This message is received by the controller, forwarded to the host and then processed in the application. From there the processed value can be forwarded to a cloud based service. It is only via the cloud service's user interface that it becomes apparent that the attacker's value has been processed. There are several points at which the forged value could have been caught and handled. The analyst must understand the relationships between the system components to be able to make useful recommendations for remediation.

This part of the analysis will leverage the approach outlined by Burns in his "Threat Modeling: A Process To Ensure Application Security". Originally this approach was developed to model threats for specific applications, so some of the points in his approach will not be applicable to this generic analysis. The main activities from Burns' approach used in this analysis are:

- mapping entry and exit points
- identifying assets
- identifying external dependencies

- identifying (generic) use scenarios
- and modeling the system [BU05].

3.1 Exit and Entry Point Identification

The first step in developing mapping the attack surface for a Bluetooth low energy application is to identify data entry and exit points. These are the pathways by which data flows in an out of the application. Figure 17 shows a set of exits and entry points that could be present in an BLE application[3].

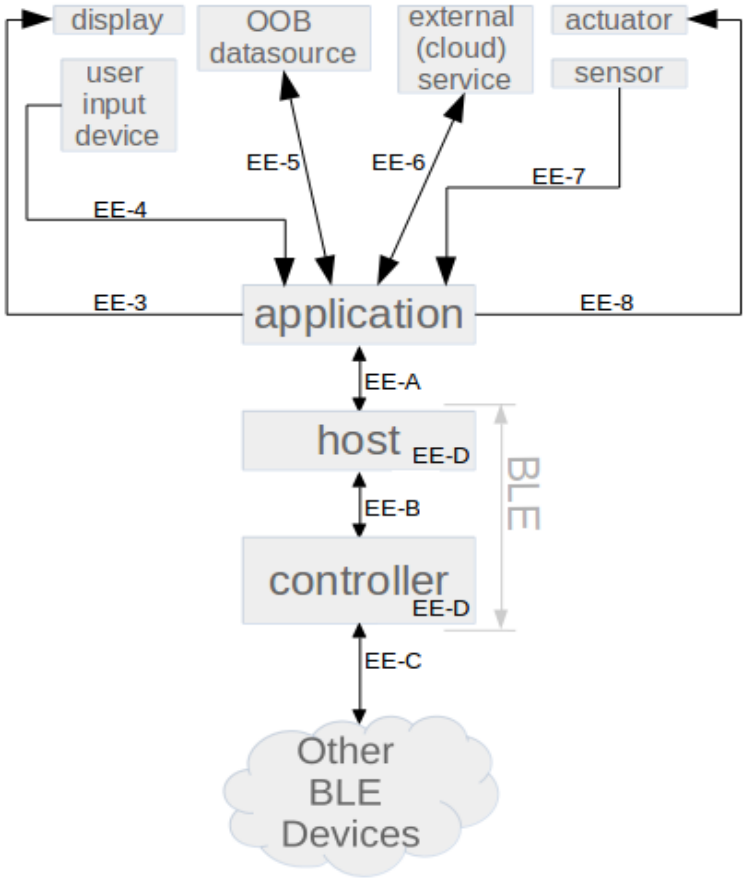

Figure 17: BLE Exit and Entry Points

3 The components listed here are common in Bluetooth low energy applications. It can be the case that there other possibilities for exit and entry points at the application level. If this is the case for a target of evaluation, then those components should be considered when performing a security evaluation of that specific application.

Each exit and entry point in the figure above has a unique label in the format EE-<ID>. Data that can be entered through these points can have an influence on BLE communication. Data that is sent out from the application have an influence of upstream processes. The table below provides a summary of the entry and exit points.

ID	Label	Description
EE-3	Application → Display	Data that is transferred from the application to some sort of output for display purposes must travel over some medium over a certain distance to the point of display. There is a difference, for example, between a mobile phone display and a monitor that is connected to a device via an HDMI cable. The communication medium and distance from the application are candidates for evaluation in a security analysis.
EE-4	User Input Device → Application	There are a variety of user input devices: keyboards, mice, switches, scanners, etc. These devices communicate over a variety of media, for example, USB cables, wireless, Bluetooth EDR, etc. The nature of the input device, the medium, and the extent to which BLE communication can be controlled with these devices are candidates for evaluation in a security analysis.
EE-5	OOB ↔ Application	OOB communication is used for pairing two BLE devices. This communication can take place over any other channel aside from BLE itself. NFC, an over-the-air radio technology, is often cited as a viable NFC option. If OOB pairing is used, the protocol and its communication medium must be taken into consideration for a security analysis.
EE-6	External (Cloud) Services ↔	It is common for applications to communicate with external services over TCP/IP. The communication can be unidirectional or bidirectional. These services

ID	Label	Description
	Application	perform actions such as data processing, data storage and system updates. The medium of communication, the protocols used and the extent to which Bluetooth low energy communication can be influenced by the interaction with the external service are candidates for evaluation in a security analysis.
EE-7	Sensor → Application	Sensors collect data that can be transferred over BLE channels to other upstream processes. They can be connected physically or wirelessly to a BLE device. The sensor, its communication medium and its communication protocol are candidates for evaluation in a security analysis.
EE-8	Application → Actuator	Actuators perform process-regulating actions based on commands from a BLE device. The commands may be generated from the device itself or they may be generated on a further upstream system and communicated to the BLE device over a BLE channel. As with sensors, actuators can be physically or wirelessly to the BLE device. The actuator, its communication medium and its communication protocol are candidates for evaluation in a security analysis.
EE-A	BLE App ↔ Host	As discussed in section 2.2, Bluetooth low energy applications are two possibilities of how the application is physically connected to the host. In some cases, the host and the application reside on the same chipset. In other cases, the application is run on a different chipset. In this case the communication medium can vary, and the protocol by which the communication occurs is proprietary. If the application is separated from the host, the communication medium and the communication protocol between the application and

ID	Label	Description
		the host are candidates for evaluation in a security analysis.
EE-B	BLE HCI (Host ↔ Controller)	As discussed in chapter 2, Bluetooth low energy devices can have both the host and the controller on the same chip as in ISOCs. There are cases, though, when the host and the controller are physically separated. The BLE specification outlines standard HCI definitions for communications between the host and the controller. If the BLE device utilizes communication via HCI, then the communication medium and HCI communication are candidates for evaluation in a security analysis.
EE-C	BLE Over-The-Air	Over-the-air communication is the core of Bluetooth low energy. Communication takes place over two different types of channels and can come in the form of advertisements, scan requests and responses, connection initiations, connection configuration, control messages and data exchange. This communication must be part of the security analysis.
EE-D	On-Board Update Interface	This is an interface, such as JTAG, that allows an individual to connect directly to the board and flash the firmware on the device. If this type of functionality is present, it is a candidate for consideration in a security analysis.

3.2 Asset Identification

An attacker will attempt to exploit exit and entry points with the goal of attacking the assets of the system. Assets are the tangible and intangible items that the system should protect. In general, assets should be defined within the context of a specific application. This section will identify the types of assets that are common in Bluetooth low energy applications. The following table

lists these assets and provides a description of the security requirement considerations for each asset.

ID	Name	Description	Security Requirement Considerations
A1	Bluetooth low energy device	Includes the host, controller and HCI interface (if present)	• Physical availability of device. • Confidentiality, integrity, authenticity of data stored and processed in host and controller. • Authenticity of host and controller firmware • Availability of device functionality. • Privacy and trackability of device.
A2	Application	Includes the software that interfaces between BLE device and other components	• Confidentiality, integrity, authenticity of data stored and processed in the application. • Authenticity of application software. • Availability of application.
A3	Downstream process(es)	Processes that are performed based on data collected via exchanges over BLE channels.	• Protect integrity and authenticity of information used in these processes with considerations made based on: ◦ human safety ◦ environmental impacts ◦ financial impact from process breakdown ◦ availability of service. • Confidentiality of data involved in downstream processes.

ID	Name	Description	Security Requirement Considerations
A4	Display	Display tied with application. Information in display may have impact on decision making in downstream processes.	• Confidentiality of displayed data. • Integrity and authenticity of displayed data. • Availability of displayed data. • Availability of display device.
A5	Actuator	Mechanism that physically regulates a process.	• Availability of actuator. • Integrity and authenticity of data that provides instructions to actuator. • Confidentiality of data that is transferred to actuator may also be a consideration. • Authenticity of actuator firmware.
A6	External (cloud) service	Service that gathers data from application over time for purposes of process control or decision making	• Confidentiality of data stored in service. • Integrity and authenticity of data stored and processed in service. • Availability of service. • Authenticity of service software.
A7	User Input Device	Device that allows the	• Physical availability to user input device.

ID	Name	Description	Security Requirement Considerations
		input of data into the application directly by the user.	• Integrity and authenticity of inputted data. • Authenticity of device firmware.
A8	Sensor	Mechanism that (automatically) collects data from an environment for use in a downstream process.	• Physical availability of sensor. • Integrity and authenticity of collected data. • Authenticity of sensor firmware.
A9	OOB Data Source	Data source that provides key material in OOB pairing.	• Confidentiality of key material values. • Integrity and authenticity of transmitted key material values.

3.3 Identification of External Dependencies

A target of evaluation's external dependencies are components or conditions that have an influence on the overall system, even though they may not have direct inputs to the data flow of the systems. In the case of Bluetooth low energy, there are three generic external dependencies that can be identified that could be points of exploitation by an attacker. These are identified in table below.

ID	Name	Description
D1	Battery	BLE devices are dependent on a source of power. BLE is specifically designed to consume as little power as possible, but if the battery could be attacked, the

ID	Name	Description
		availability of the device could be compromised.
D2	Physical Operating Conditions	The manufacturer of a target of interest may have specific assumptions about the environmental conditions surrounding the use of the device. Factors such as heat, moisture or the presence of dust could impact a device's availability or the integrity of the data that is processed in the device.
D3	BLE Channel Availability	A BLE target of evaluation assumes the availability of the BLE advertising and data channels. The specification allows for flexibility if some channels are not available due to interference. This being said, strong electromagnetic interference and/or intentional signal jamming could lead to a loss of availability of data communication on BLE over-the-air channels.

3.4 Use Scenario Definition

Use scenarios primarily provide discrete descriptions how the system will be used and will not be used. Defining use scenarios is an essential part of threat modeling for a target of evaluation. This helps build later data flows for a specific security analysis and provides the foundation of scope for a security analysis.

Use scenarios are strongly tied to the actual target of evaluation, however three use scenarios are likely to be common across many BLE targets of evaluation. They are defined in table below.

ID	Name	Description
US1	System Startup	In order for a BLE device to be used, it must go through an initialization process. This establishes the BLE functionalities supported by the host and the controller. Data that is needed by the controller is moved from the host to the controller.
US2	BLE Application	This is the software update process for the BLE

	Software Update	application which can include things like bug fixes, patches for security flaws and new functionality.
US3	BLE Firmware Update	This is the firmware update process which can include things like bug fixes, patches for security flaws and new functionalities.
US4	BLE Pairing & Bonding	Pairing and bonding is the process of two devices exchanging cryptographic key material to support security functions such as providing confidentiality over a connection, providing authenticity of exchanged data and protecting the trackability of a device over an extended period of time.

3.5 System-Specific Points for Analysis

In Burn's threat modeling process, there are three system-specific points of analysis that are applicable for specific targets of evaluation, but do not provide any value to analyze at a generic BLE security level. These are: developing external security notes, determining internal security notes and identifying implementation assumptions.

External and internal security notes deal respectively with risk deference and risk acceptance. External security notes specify the terms of use that a manufacturer expects from a user when employing the target of evaluation. Internal security notes explicitly list the security concessions that have been made in the development of the product. Implementation assumptions identify the further development paths for the target of evaluation. These plans may have an impact on security decisions that are made for the current version of the product under evaluation.

Because these points of analysis are tightly coupled with the context of actual applications, these points will not be further analyzed here.

3.6 Modeling the System

The next step to developing a threat model is to identify the process flows of the application. This identifies process components such as inputs, outputs, data stores, and processing. This allows the security analyst to analyze points of attack more concretely. Normally this would be performed for a specific target of interest.

For the purposes of this generic analysis, common data flows found in BLE communication will be abstracted and presented to support the identification of possible attack vectors. The examples analyzed in this section will not cover every single, possible BLE data flow. For efficiency's sake, eleven categories of data flows will be introduced, and representative examples of those data flows will be examined.

3.6.1 Process Flow Type #1: System Initialization

At the reset of a BLE device, data storage on the controller is cleared. The host and the controller send several messages back and forth specifying which functionalities are supported by the link layer in the controller and which commands are supported by the host. In addition to this, the host has the chance to set a random device address, add entries to the whitelist and add IRKs to the resolving list. Figures 18-20 shows examples of the data flow between the host and the link layer to perform these actions.

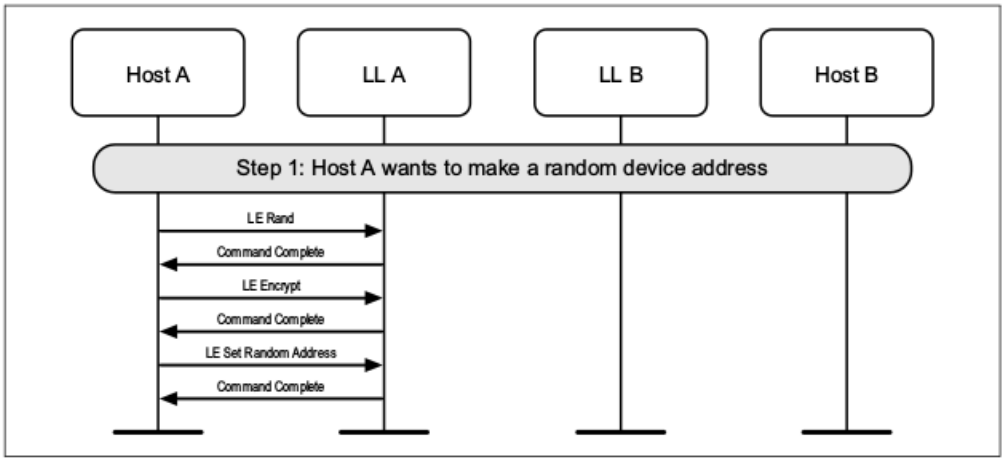

Figure 18: Initialization: Setting a random address [BLE-LL,134]

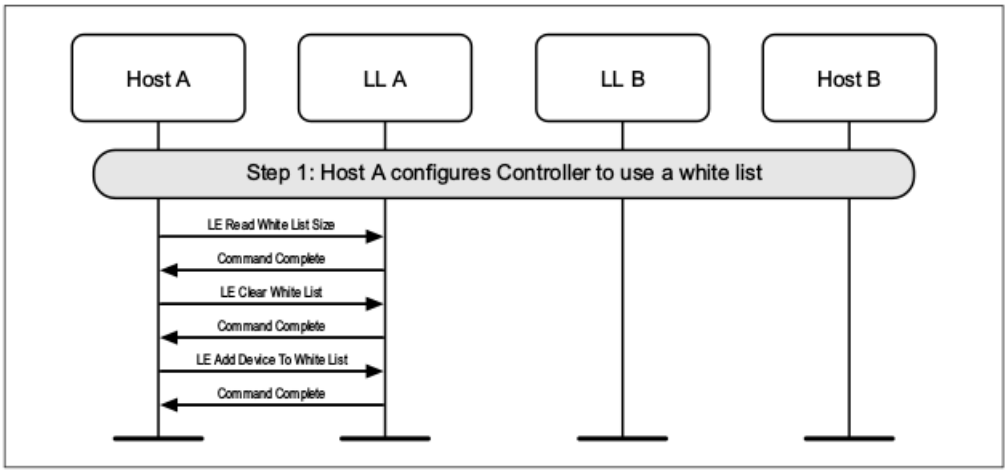

Figure 19: Initialization: Adding entries to whitelist [BLE-LL,134]

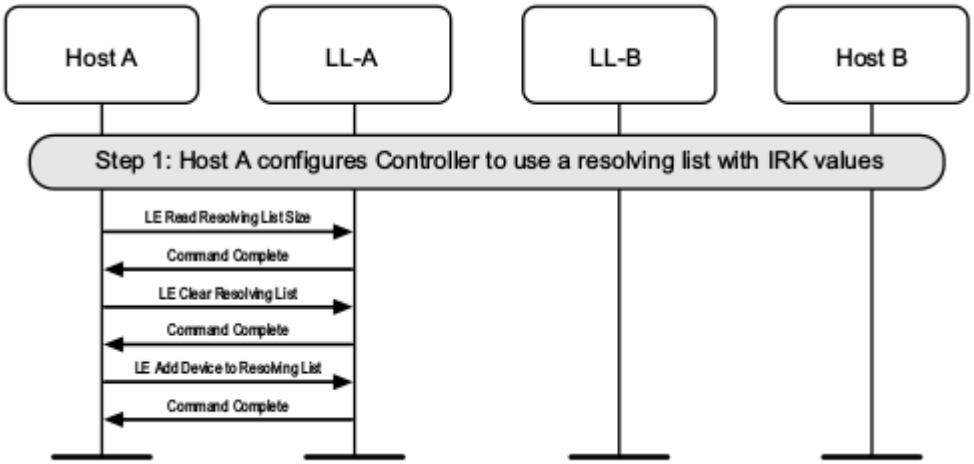

Figure 20: Initialization: Configuration of a resolving list [BLE-LL,135]

3.6.2 Process Flow Type #2: Advertising

Advertising is the act of a device sending out broadcast messages on one of three advertising channels. This may have the purpose of broadcasting data for consumption by target devices. It may have the purpose of letting other devices know that the advertiser is accepting scan requests or connection requests. Important to know in this flow is that the data being broadcast is accessible to all devices scanning on the advertising channels as illustrated in figure 21.

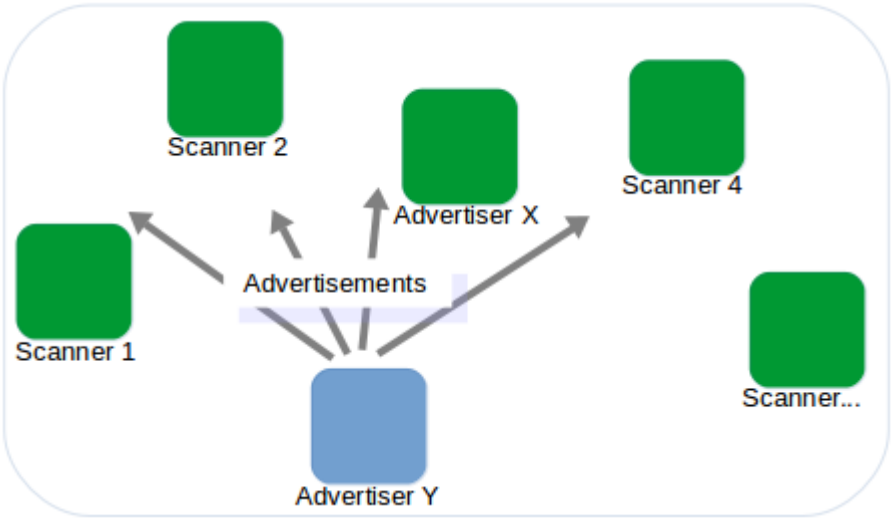

Figure 21: Advertising: One to Many Communication

In order to prepare an advertisement, multiple exchanges are made between the host and controller to establish the way that values will be broadcast. An example of an undirected advertisement can bee seen in figure 22.

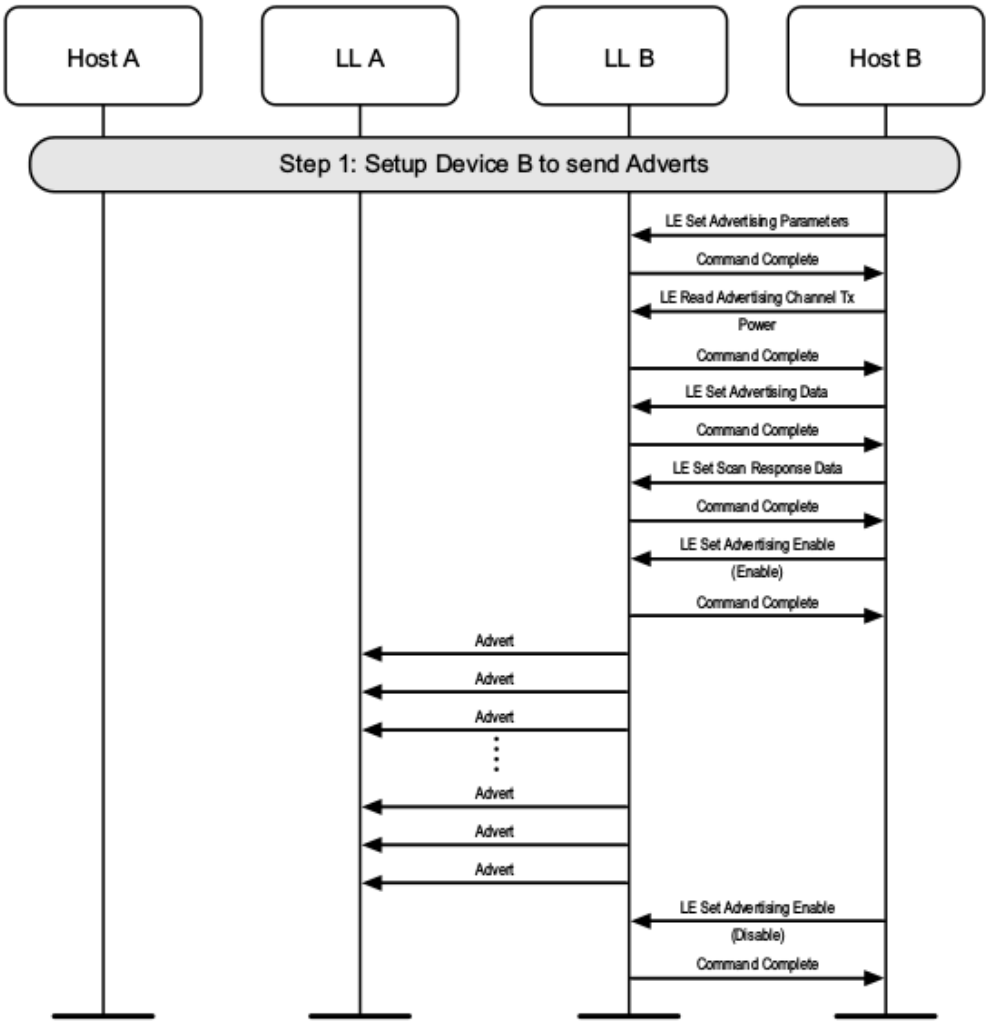

Figure 22: Advertising: Internal Flows in Advertiser for Undirected Advertisements [BLE-LL, 137]

3.6.3 Process Flow Type #3: Scanning

In Figure 19, in addition to the advertiser, there are multiple scanners. Scanners listen for advertisement and decide what to do, if anything, with them. A scanner has ultimately four options: ignore th advertisement, receive the data and pass it to the host for further processing, issue a scan request for additional data or initiate a connection. It is important to note, that scan requests are also broadcast on advertising channels and can be observed by devices listening on those channels.

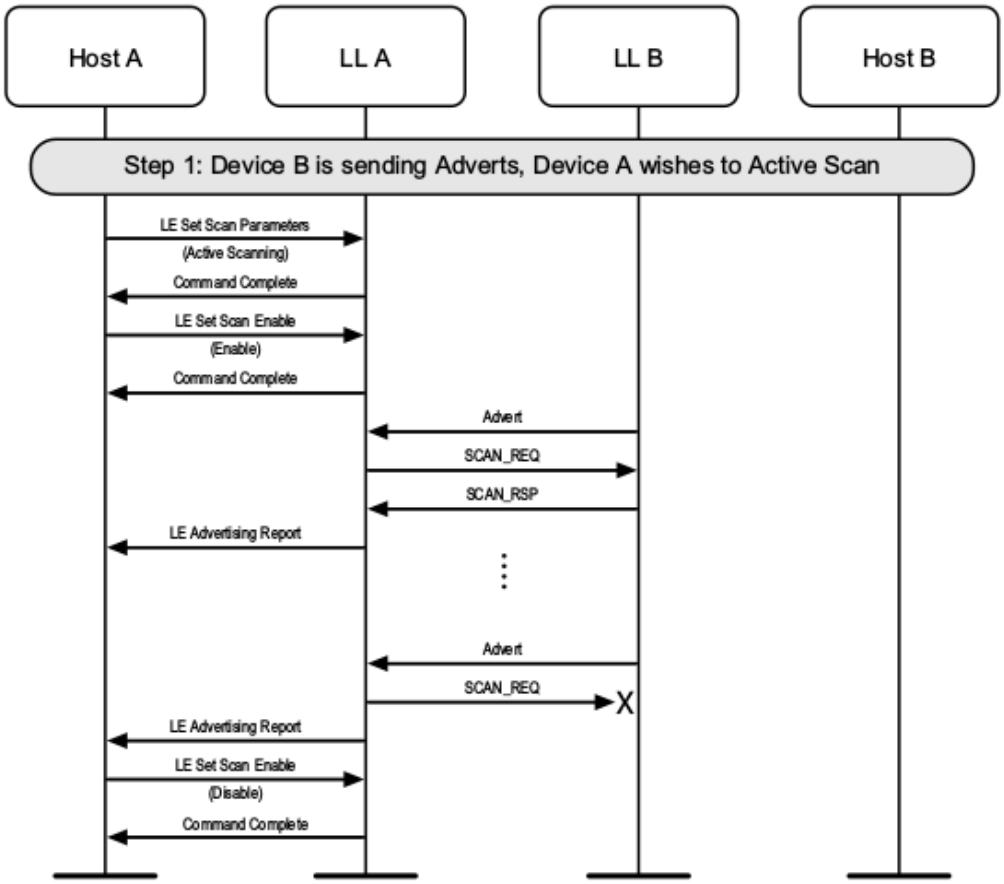

Figure 23: Scanning: Scanner Issues Scan Requests in Response to Advertisements [BLE-LL, 140]

Figure 23 shows the process flow for a scanner listening for advertisements and issuing scan requests.

3.6.4 Process Flow #4: Initiating Connection

As mentioned above, one of the actions that a scanner can take is to respond to an advertisement that indicates the advertiser is accepting connection requests. The scanner responds by moving from a scanning state to an initiator state and sending an initiate connection request as shown in figure 24.

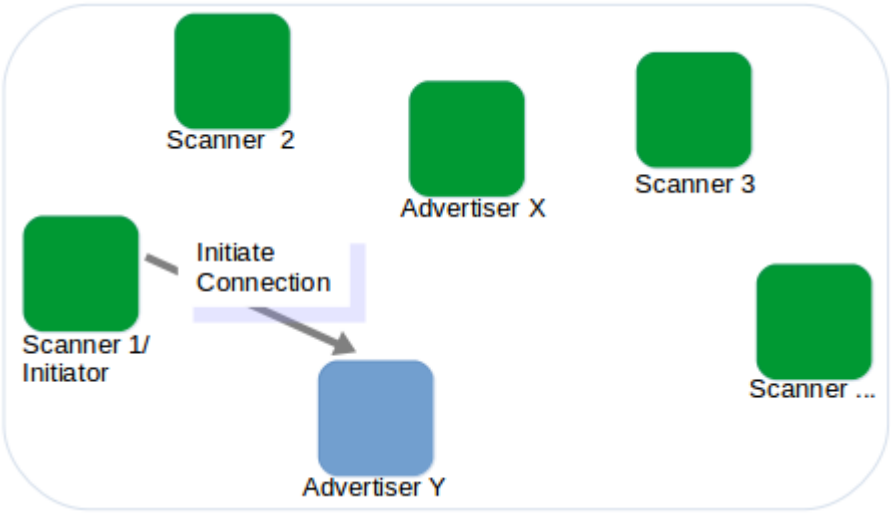

Figure 24: Initiating Connection: Scanner Moves to Initiator State

This initiate connection request is broadcast over the advertisement channel and can be heard by any device listening on the advertising channels. The original advertiser can choose not to accept the connection request. Figure 25 shows a more detailed process flow of connection initiation.

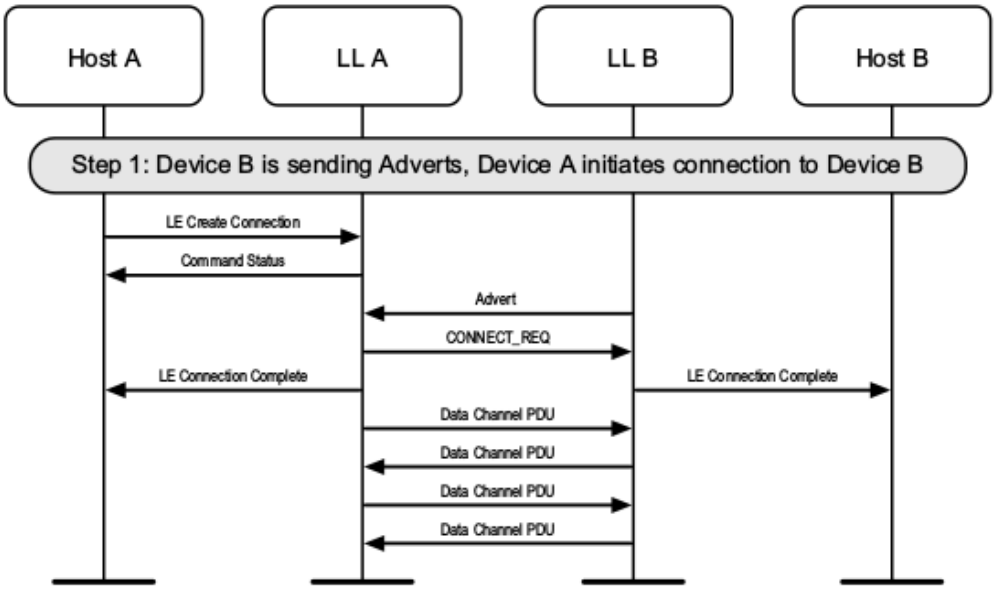

Figure 25: Initiating Connection: Connection Request and Data Channel Establishment [BLE-LL, 144]

After connection establishment, the initiator becomes the master and advertiser becomes the slave. It is important to note that devices are either masters or slaves. They cannot be both at the same time. And, further, masters may be connected to many slaves, but a slave may only have one master as show in figure 26.

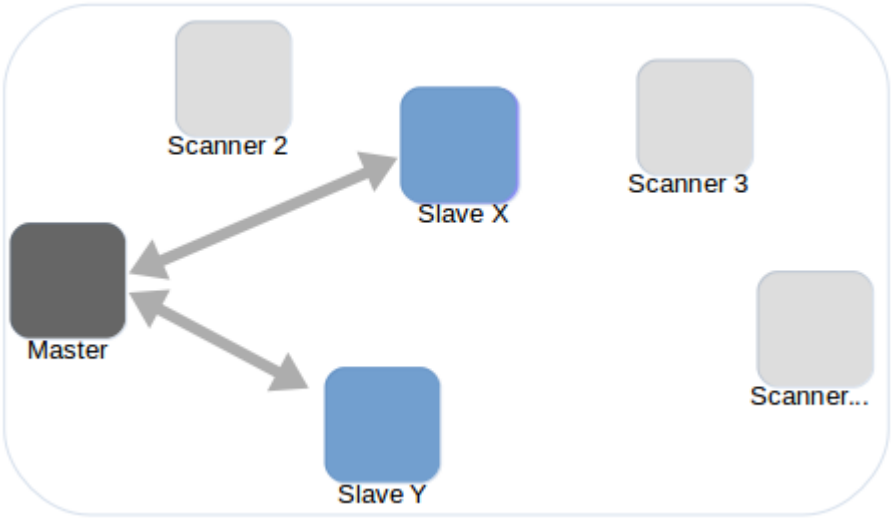

Figure 26: Initiating Connection: Master with Multiple Slaves

3.6.5 Process Flow #5: Exchanging Data over Data Channels

After the establishment of a connection between a master and slave, data can be exchanged in a client-server format between the two peripherals. The client makes requests of the server for data. The server responds with either the data or an error message. In this scenario, a master can be either a client or a server. The same holds true for the slave. Client and server are completely separate from the master and slave roles. Figure 27 shows the data flow for the exchange of data over a connection between two BLE peripherals. Read requests and their responses are issued over LL Data Packets.

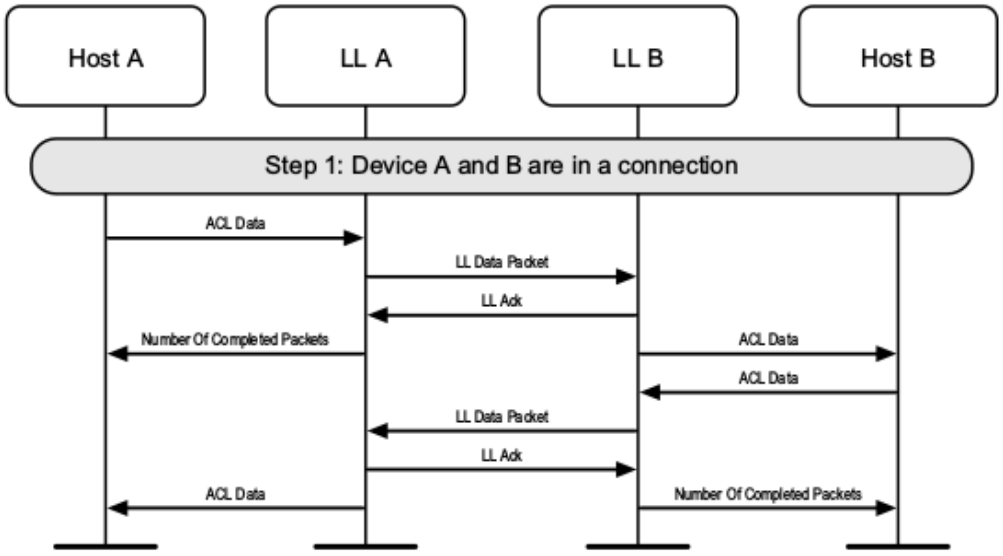

Figure 27: Data Exchange: Both Peripherals Act as Client and Server [BLE-LL, 148]

3.6.6 Process Flow #6: Control Messages

There are a number of control messages that can be sent between two peripherals that are communicating in a connection on data channels. These control messages have the potential to reset the parameters of communication such as the selection of channels, redefining the MTU or reseting the timing parameters for communication. Figure 28 shows an example of the master specifying new connection parameters for a peripheral.

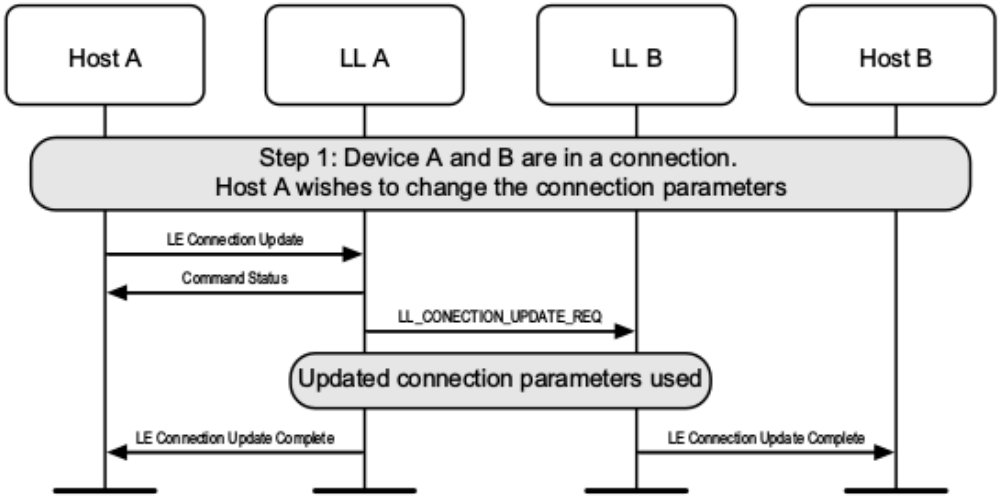

Figure 28: Control Messages: Master Resets Connection Parameters [BLE-LL, 149]

An important control message that can be issued by both the master or the slave is the command to disconnect a connection. This flow is represented in figure 29.

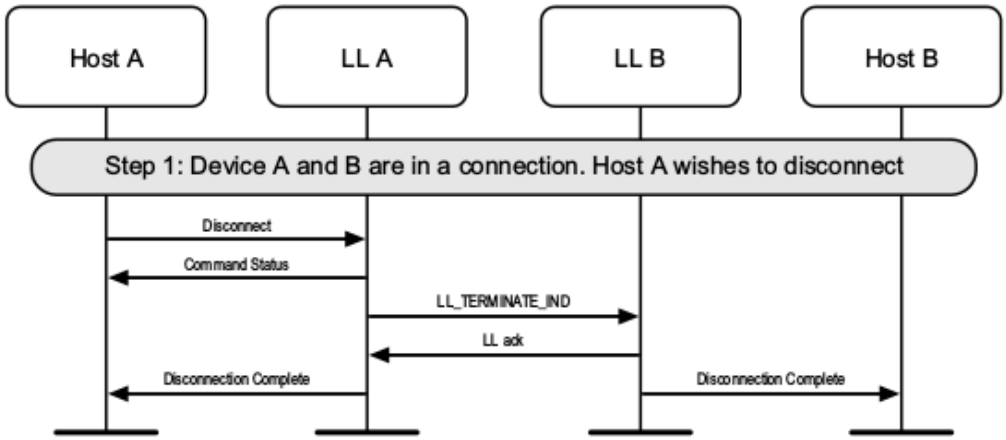

Figure 29: Example of a Disconnect Flow (Initiated by Either Master or Slave) [BLE-LL, 157]

3.6.7 Process Flow #7: LE Legacy Pairing/Bonding

As discussed in chapter 2, there are two types of pairing/bonding in Bluetooth Low Energy: LE Legacy and LE Secure. Both have the aim to provide mechanisms for the provision of confidentiality and authenticity for the exchange of key material, but they accomplish the key material exchange for this in extremely different ways. LE Legacy pairing will be addressed in this section, and LE Secure pairing will be addressed in section 3.6.8.

In general, the specification breaks down pairing/bonding for both LE Legacy into three phases:

1. *Phase 1: Pairing Feature Exchanges*

   This is the phase where the master sends a pairing request to the slave. The master may be doing this autonomously, or it may be responding to a security request from the slave. In response, the slave responds with a pairing response or an error. The type of data that is exchanged includes:

   a. user input/output capabilities

b. whether or not OOB data are available

c. whether or not LE secure is required

d. which cryptographic keys are requested (options: LTK, CSRK, IRK)

e. whether or not bonding is required

f. maximum encryption key size (58 bits – 128 bits)

The exchanged data allows the two devices to select the appropriate pairing method based on user input/output and the presence of OOB data.

2. *Phase 2: Establish keys to set up encryption*
   During this phase, the two devices exchange values to establish a set of common keys to encrypt the connection for key exchange in phase 3.

3. *Phase3: Encrypt link and exchange key material*
   In Phase 3 connection, otherwise referred to as “link”, encryption is set up between the master and the slave devices. The keys that were specified in phase 1 are subsequently generated/distributed along with other key material over the encrypted link.
   [BLE-SMP, 593]

3.6.7.1 LE Legacy Pairing Phase 1

The goal of phase 1 is to share information needed to start pairing. Figure 30 shows the message flow between the master and the slave to start pairing.

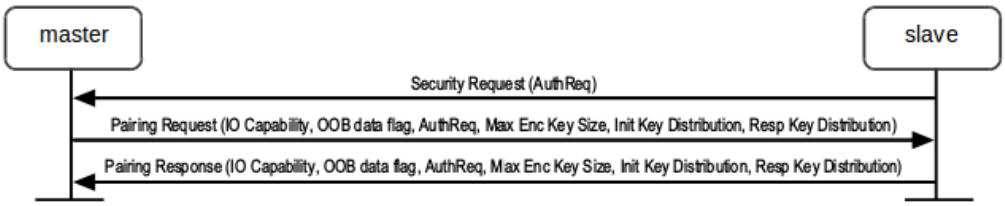

Figure 30: Phase 1 pairing message exchange [BLE-SMP,661]

In this exchange, there are at a minimum two messages and an optional third message. The essential two messages are the pairing request and pairing

response messages. The master always initiates pairing. The slave may request that pairing be started with the security request message.

3.6.7.2 LE Legacy Pairing Phase 2

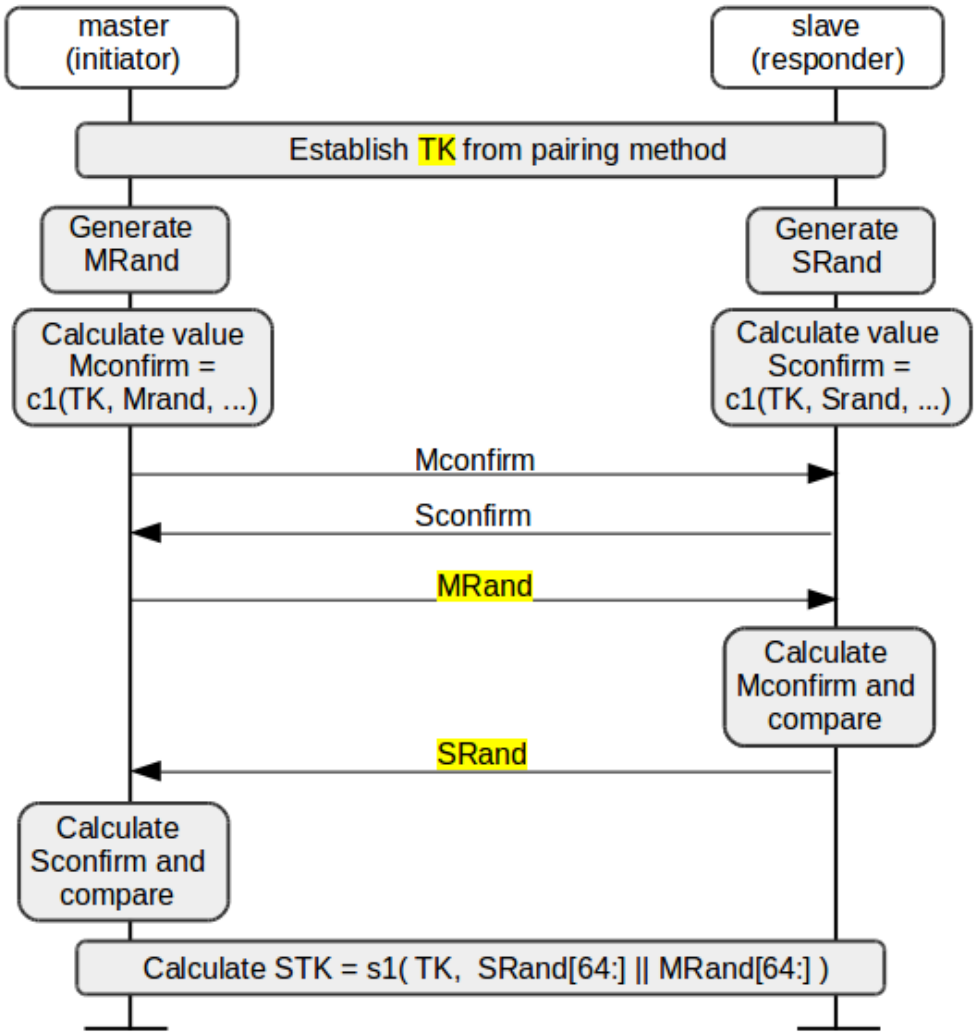

Figure 31: LE Legacy Phase 2 Pairing Process Flow [BLE-SMP, 661-663]

The goal of phase 2 of LE Legacy pairing and key exchange is to establish short term keys (STK) for both the master and the slave. The STK is calculated using three values: a temporary key (TK), a random value generated by the master (MRand) and a random value generated by the slave (SRand). Figure 31 shows how these values are shared and distributed.

In legacy pairing, the TK value is derived from the type of pairing. There are three types of pairing available in legacy.

1. Just works: In this case, the TK value is 0x00000000000000000000000000000000.
2. Passkey entry: In this case, the TK value is the hexadecimal value of the entered 6 digit PIN prepended by enough zeros to make it a 128 bit number. The value can range from: 0x00000000000000000000000000000000 to 0x0000000000000000000000000F423F.
3. OOB: In this case, this is the value that is transmitted to the BLE device via an OOB channel. This value is 128 bits. The value can range from 0x00000000000000000000000000000000 to 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF.

It is important to note that the TK is not exchanged over a BLE data channel. This value is either fixed (as in just works), generated in one device and manually entered on the other (passkey entry) or sent over a different communication (OOB).

After the TK has been established, the devices each generate a RAND value, and they use a variety of shared values to calculate confirm values (Mconfirm and Sconfirm) using a function called “C1”. After this, the master and the slave each exchange their confirm values.

The master then sends its MRand value to the slave. The slave uses the same C1 function along with the same inputs and the MRand value to see if the result matches the Mconfirm value sent from the master. If this is successful, the slave sends the SRand value to the master. The master performs the same type of calculation using SRand and checks the result against the Sconfirm value received from the slave.

If both the Mconfirm values and the Sconfirm values are verified, both the master and the slave use the original TK value along with the exchanged

MRand and SRand values to generate the STK using the key generating function “S1”. As an input to this process, the TK is entered as the key to the function, and the least significant 64 bits of both the SRand and MRand values concatenated together are entered as the value to the function.

At this point, encryption is established on the link using the STK (see chapter 3.6.9.2 for the link encryption process flow), and cryptographic material can be exchanged.

3.6.7.3 LE Legacy Pairing Phase 3

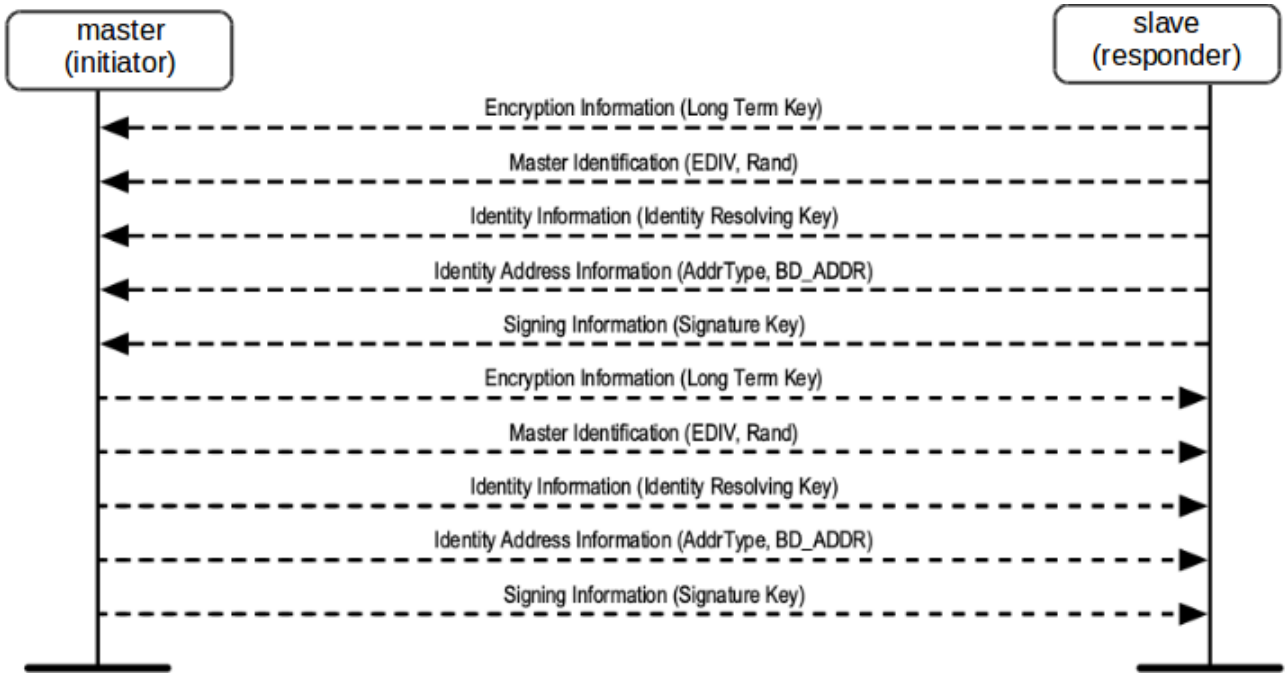

Figure 32: Phase 4 LE Legacy Pairing - Exchange of Cryptographic Material [BLE-SMP,678]

Figure 32 shows all of the possible cryptographic material that can be exchanged in LE Legacy pairing. For both the master and the slave, the long term key, or the LTK, is a key that will be stored long-term for the encryption of future links. The EDIV and Rand values are stored with the LTK in the recipient device for the later identification of the correct LTK. LTKs can be generated pseudorandomly or they can be derived from another key (ER) using the key diversifying function, d11. The IRK is used to resolve random

resolvable addresses on the advertising channel. And, the signature key, or CSRK, is used to generate MIC (message integrity codes) in an attempt to provide assurance of authenticity of the contents of a transmitted PDU.

3.6.8 Process Flow #8: LE Secure Pairing/Bonding

The only similarity between LE Legacy pairing/bonding and LE Security pairing/bonding is that it provides a means to exchange an LTK, IRK and CSRK. The mechanism used by which confidentiality and authenticity are ensured in LE Secure is via elliptic curve cryptography using the NIST P-256 curve.

Unlike LE Legacy, the first part of the LE Secure process flows for pairing/bonding differ based on the pairing method chosen; just works and numeric comparison, passkey entry and OOB pairing. The pairing methods will be addressed separately in the next subsections. In the final subsection, the establishment of the LTK and the exchange of the IRK and CSRK will be addressed.

3.6.8.1 Process Flow #8.1: LE Secure Just Works and Numeric Comparison

The two methods in LE Secure, just works and numeric comparison, share similar process flows. Figure 33 provides an overview of the exchanges that take place to establish key material to generate the LTK and to prepare for the exchange of the IRK and/or CSRK.

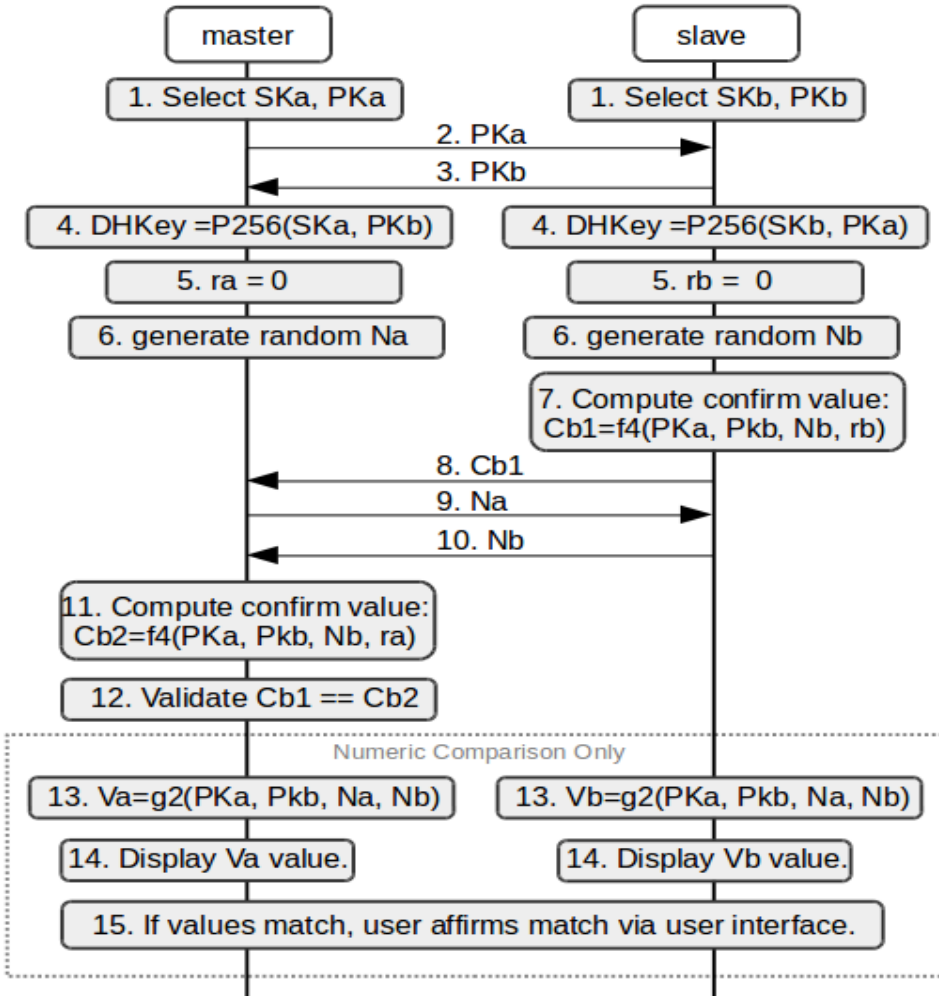

Figure 33: LE Secure Just Works and Numeric Comparison [BLE-SMP,666]

The steps below explain the figure in more detail.

1. Steps 1-4 represents the exchange of public keys and the generation of a shared Diffie-Hellman key.
2. Each device has its own r variable which is set to 0 as seen in step 5.
3. After this, each device generates its own 128 bit nonce.
4. In step 7, the slave device calculates a confirmation value using the exchanged public keys, the nonce it has generated, and the slave's r value. It then sends the confirmation value to the master.

5. The master then sends its nonce value to the slave, and the slave follows by sending its nonce value to the master.
6. The master than proceeds to calculate a second confirmation value using the nonce forwarded from the slave. If this confirm value matches that previously sent by the slave, then it moves on to the next step.
7. If the pairing method is numeric comparison, each device calculates a value based on the over-the-air exchanged nonces and public key values via a function “g2”. This value is displayed on each device, and the users have the chance to compare values and verify that they match through a user interface.

3.6.8.2 Process Flow #8.2: LE Secure Passkey Entry

The passkey pairing method is similar to just works and numeric key entry, although the difference is significant enough such that it warrants its own process flow. Figure 34 provides an overview of the exchanges that take place between master and slave to e establish key material to generate the LTK and to prepare for the exchange of the IRK and/or CSRK.

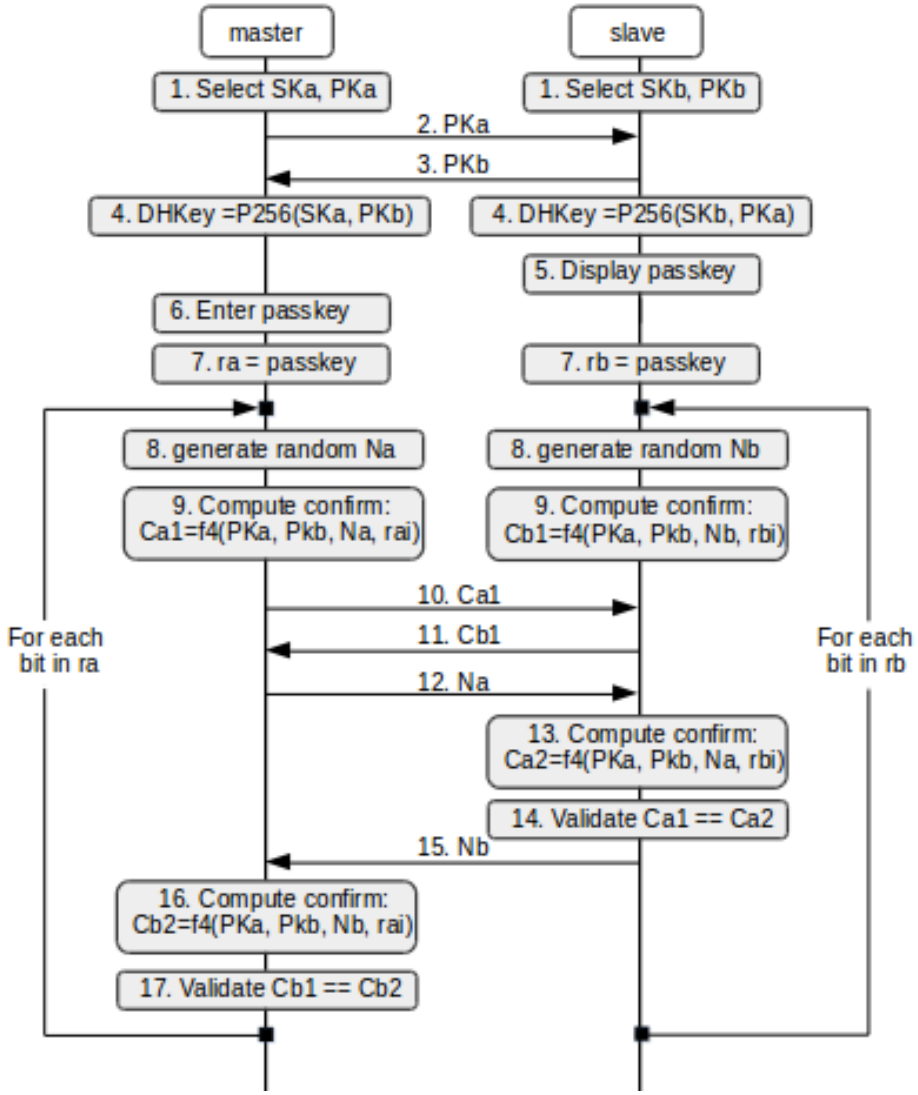

Figure 34: LE Secure Passkey Entry [BLE-SMP, 670]

The steps below explain figure 34 in more detail.

1. Steps 1-4 represents the exchange of public keys and the generation of a shared Diffie-Hellman key.

2. The passkey is displayed on one device, and it is entered via user interface on the other device. The passkey value is never exchanged over-the air.

3. Both devices store the entered values in an r variable. Then, for each bit in the passkey value, a series of steps are undertaken:

1. A nonce is generated in each device and passed along with the two public key values and the bit value from the passkey to the f4 function to calculate a confirm value.
2. The confirm values are exchanged.
3. The master sends its nonce.
4. The slave calculates the confirm value based on the the exchanged public keys, the bit value from the passkey and the nonce sent from the master. If the confirm value matches the confirm value from the master, then it forwards its nonce to the master.
5. The master calculates the confirm value in a similar manner using the nonce sent from the slave. If it calculates the same confirm value as was sent by the slave, then the process moves to the next step.
6. If end of the passkey has been reached, then the next step is to calculate the LTK and prepare for the exchange of the CSRK and IRK. Otherwise, the next bit is selected from the passkey, and the process goes back to step 8.

3.6.8.3 Process Flow #8.3: LE Secure OOB

The out of band communication method of pairing is similar to the other LE Secure process flows. Interestingly, it provides more structure about which data is exchanged for the purposes of pairing than in LE Legacy pairing.

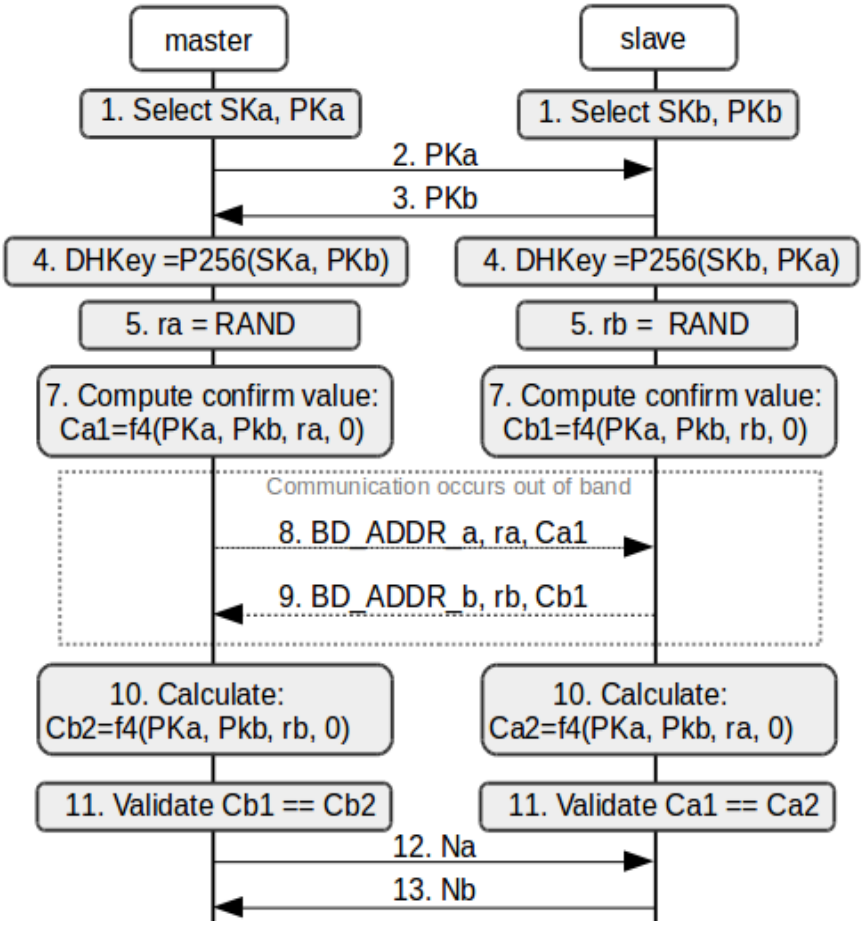

Figure 35: LE Secure OOB [BLE-SMP, 674]

Figure 35 provides an overview of the exchanges that take place between master and slave to e establish key material to generate the LTK and to prepare for the exchange of the IRK and/or CSRK.

The steps below explain figure 35 in more detail.

1. Steps 1-4 represents the exchange of public keys and the generation of a shared Diffie-Hellman key.
2. Both devices generate random r values. These are used to calculate confirm values on both devices.
3. Each device sends its device address, its r value and is confirm value to the other device. This exchange does not take place over a BLE

channel. It instead is transferred over some alternate channel and is managed by the security manager.

4. Each device calculates the confirm values with the received r values and validate that the match the transferred confirm values. If this is the case, the devices then exchange nonces for later use in generating the LTK and the exchange of the IRK and CSRK.

3.6.8.4 Process Flow #8.4: LE Secure LTK, CSRK and IRK

The LTK, just as in LE Legacy pairing, is the key that is used to generate the session keys used to encrypt the link to ensure confidentiality in communication. The generation of the key is represented in figure 36.

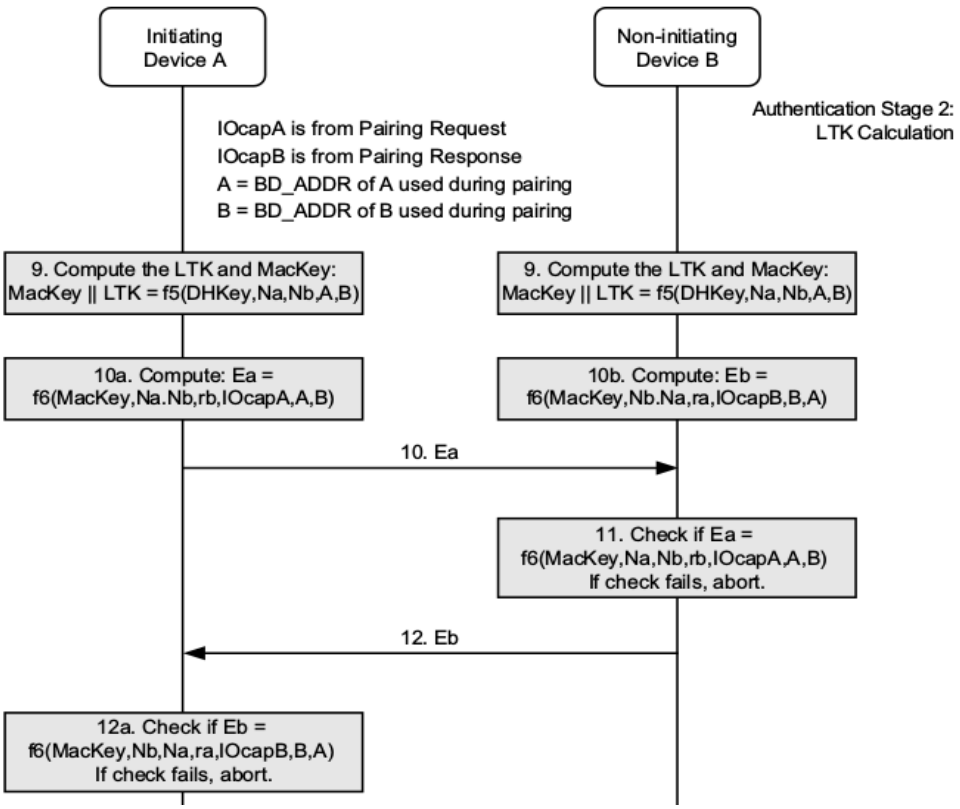

Figure 36: LE Secure LTK establishment [BLE-SMP, 676-677]

The steps below provide more detail about flow represented in figure 36.

1. The two devices calculate a MAC key and the LTK using the shared Diffie-Hellman key and data exchanged during the previous steps of the pairing process. The LTK is stored with the device address (BD_ADDR). The MAC and LTK values are 128 bit values that can be used as keys for use with AES.
2. Each device calculates a confirmation value using the f6 function.
3. The master sends its confirmation value, Ea to the slave.
4. The slave validates the calculation of Ea. If its value matches the value calculated by the master, it sends its confirmation value, Eb, to the master.
5. The master then validates the sent Eb value. If it, too, matches the value it calculates, then the LTK value has been established.

Once the LTK value has been established, link encryption can begin. The EDIV, Rand and, if required, the CSRK and/or IRK can be exchanged over the encrypted channel. A session key can be established, and the encrypted data exchange can begin.

3.6.9 Process Flow #9: Link Encryption Process Flow (Encryption and Authentication)

Encryption only takes place on the data channel and is not part of the specification for advertising, scanning and initiating packets. In other words, encryption can only take place on an established BLE connection. Figure 37 shows how link encryption is started on a BLE link.

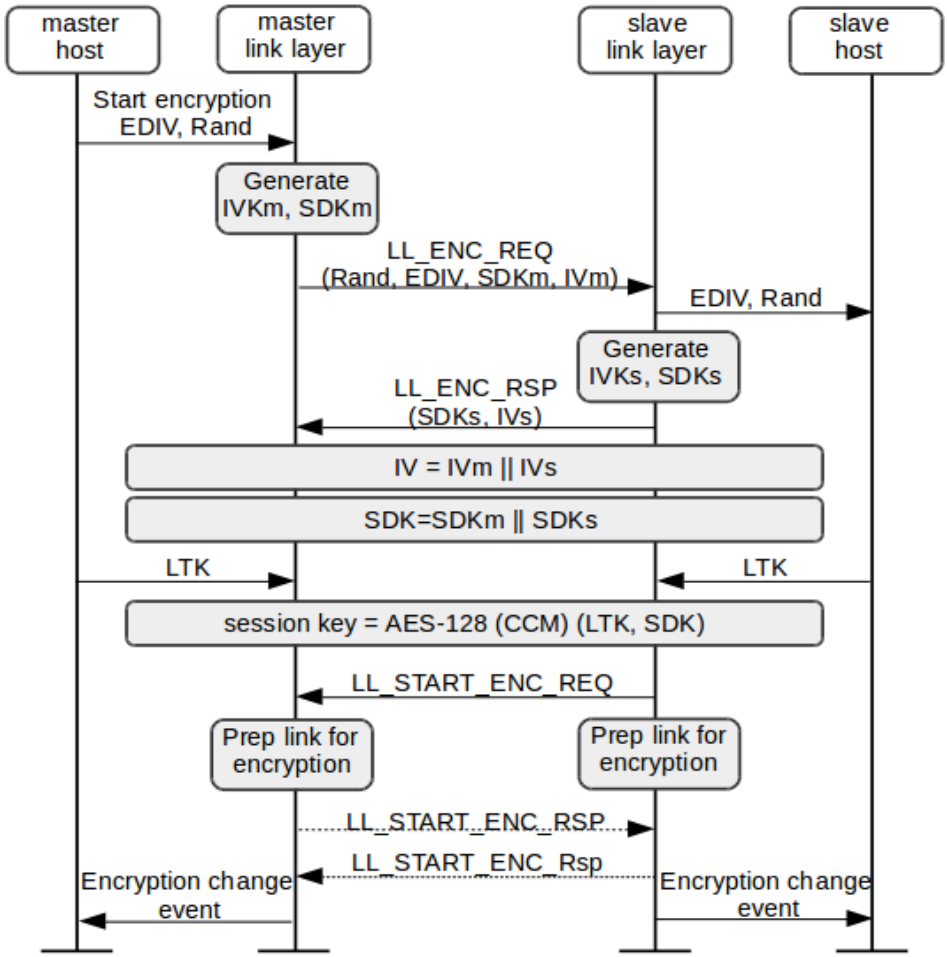

Figure 37: Link session encryption establishment [BLE-LL, 95-98]

Connection, otherwise known as “link”, encryption starts in the host of the master. The host sends the EDIV and the Rand value that belong to the LTK exchanged previously with the slave to the master’s link layer. This is the flag to the link layer that the link encryption process has started. The following steps then occur:

1. The master link layer generates 2 random values: the master’s part of the session key diversifier (SKDm) which is 64 bits long and the master’s portion of the initialization vector (IVm) which is 32 bits long. If the original pairing when the LTK was established was an LE Secure pairing, the EDIV and Rand values are set to 0.

2. The master sends an LL_ENC_REQ packet to the slave which includes the Rand and EDIV values from the master's host along with the SKDm and the IVm.

3. The link layer of the slave receives this request and the sends a message to its host with the EDIV and Rand value to look up and validate the LTK.

4. The link layer of the slave then generates 2 random values: the slave's part of the session key diversifier (SKDs) which is 64 bits long and the slave's part of the initialization vector (IVs) which is 32 bits long.

5. The slave then sends an LL_ENC_RSP packet to the master which includes the SKDs and the IVs.

6. Both the master and the slave then calculate the initialization vector and the session key diversifier.
   A. The initialization vector is the concatenation of the IVm and IVs values.
   B. The session key diversifier is the concatentation of the SKDm and SKDs values.

7. At this point, the LTK is requested from the hosts of both the slave and the master.

8. The master and the slave then calculate the session key. The calculation of the session key is performed using AES-128, the LTK as the key and the SKD as the plaintext value.

9. Once the session key has been established, the slave sends an unencrypted LL_START_ENC_REQ packet. This packet has an empty data channel payload.

10. The link layer of the slave then immediately starts preparing itself for encryption. It creates a 5 byte packet counter value and sets it to 0.

11. Upon receipt of the LL_START_ENC_REQ packet, the master's link layer prepares itself for encryption. It, too, creates a 5 byte packet counter and sets the value to 0.

12. The master then sends an encrypted LL_START_ENC_RSP value to the slave.
13. The slave returns an encrypted LL_START_ENC_RSP value to the master.

After this all packets that have a non-zero length data channel PDU that are exchanged will have two cryptographic operations performed on them. First, a MIC (CMAC) will be generated based on the first eight bits of the data channel PDU header and the data channel PDU payload. This MIC will be appended to the data channel PDU. After this, the data channel PDU value and the MIC will be encrypted. The next two sections will address the mechanics of these two cryptographic operations.

3.6.9.1 MIC (message integrity check)

A MIC is a cryptographic, 4 byte MAC based on the AES-128 CBC-MAC algorithm. The MAC is calculated with the following values:

- The current session key established at encryption setup.
- A nonce that is made out of 39 bits of the packetCounter, 1 directional bit which corresponds to the device sending the packet, specifically master (1) or slave (0), and an 8 byte IV.
- The payload of the data PDU and the first byte of the PDU header where the NESN, SN and MD fields are set to 0.

Figure 38 shows the organization of a non-encrypted BLE packet and the components that are included in the MIC.

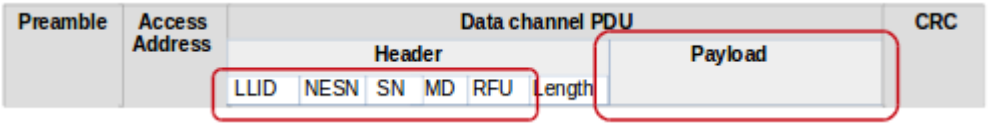

Figure 38: Fields used to calculate the MIC [BLE-SMP,167]

The MIC is the affixed to the end of the data channel PDU. This means that if the PDU payload is not empty, it goes directly after the payload. If the PDU payload is empty, it is appended to the header.

3.6.9.2 Link Encryption

Encryption takes place with the AES 128 CBC-MAC. The encryption is performed using the following values:

- The current session key exchanged during encryption setup.
- The nonce used to generate the MIC. Each time a new encryption occurs, the packetCounter part of the nonce is incremented by 1.
- The data channel PDU payload value and the MIC.

Figure 39 shows the organization of the a non-encrypted BLE packet and the components that are encrypted.

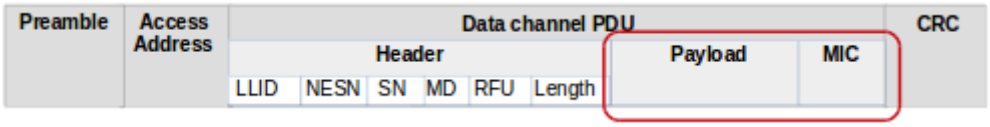

Figure 39: Fields to be encrypted [BLE-LL, 167]

3.6.10 Process Flow #10 Authenticating Packets without Encryption

According to the specification, unencrypted packets can be exchanged, and authenticity can still be assured through the use of a MAC. The MAC is generated through the use AES-128 CBC-MAC and takes in three values:

- CSRK: This is a key that was exchanged via an earlier pairing/bonding exchange.
- M: This is a concatenation of the data PDU that needs to be signed and the SignCounter. The SignCounter is a 4 byte counter value that is set to 0 when the CSRK is received. For each message that is “signed”, the SignCounter is incremented by one.

- Tlen: This value is fixed at 64 and represents the size in bits that the resulting MAC should be.

The MAC is then appended to the data PDU. Figure 40 shows a logical representation of the data PDU to the signature.

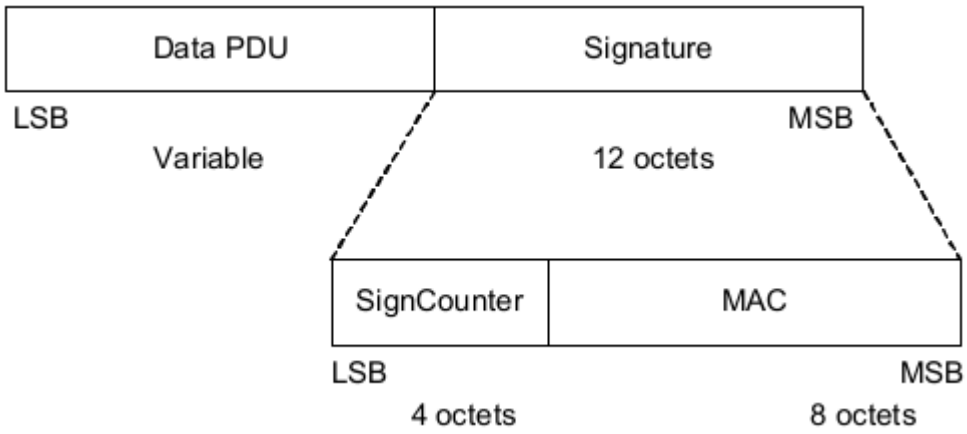

Figure 40: Relationship of SignCounter and PDU Payload [BLE-GAP, 381]

3.6.11 Process Flow #11: Private Address Generation and Resolution

To make a device less trackable when in the advertising state, BLE provides the opportunity to randomize the AdvA address in an advertising packet. As highlighted in chapter 2, this address can be completely random and not resolvable by a receiving device, and it can appear to be random but still resolvable by a receiving device that has the correct key, IRK. Exchange of the key IRK can be found in section 3.8.6.4.

Resolvable private addresses have the following format:

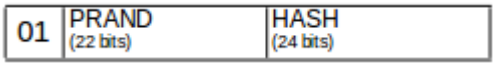

Figure 41: Resolvable private address AdvA format [BLE-SMP, 45]

To generate a private, resolvable address, a 24 bit pseudorandom number PRAND is generated by the link layer. The most significant values of this number will be binary 01. The PRAND value is hashed using AES-128 CCM with the CSRK as the key and the PRAND as the value to be hashed. The hash is divided by 2^24, and the remainder of that division is the HASH value. The PRAND value and the HASH value are concatenated and become the resolvable private address for AdvA. This set of operations is called “ah” in the documentation.

A device will use this private resolvable for only a limited time. There are two conditions that prompt the change of a resolvable private address. First, the device keeps an internal timer that specifies the lifetime of the resolvable private address. If the timer reaches 0, a new resolvable, private address is generated with the same procedure and is broadcast in the AdvA address on the advertising channels. The other way that a new resolvable, private address is generated is if the device is reset, and the host resets the IRK and prompts for the use of a resolvable, private address.

A device that receives the resolvable, private address takes the 24 most significant bits and uses the ah function and the IRK value exchanged during pairing to generate the HASH value. If the generated HASH value matches the 24 least significant bits of the sent AdvA value, the address is accepted as valid, and the packet can be processed further. Interestingly, the IRK is not associated with a specific device. If a receiving device has more than one IRK stored, it will test each IRK with the sent PRAND value until it finds a match to the HASH value.

4 Over-the-Air Threat Model

Taking into consideration all of the different characteristics identified in the attack surface in chapter 3, the next step is to define the generic threats to a BLE application from the over-the-air interface. For this part of the analysis, Burns recommends the use of STRIDE. STRIDE is an acronym for a set of threat modeling categories first published by LeBlanc and Howard in 2002. The categories include:

- Spoofing identity: Spoofing an identity is when an attacker is able to provide credentials of a different user.
- Tampering data: Tampering with data is the act of modifying the data of the application or modifying the data as it is in transit over a network.
- Repudiation: Repudiation refers to attacks where the application is not able to verify that the origin of received data comes from a trusted source.
- Information disclosure: Information disclosure is a type of attack where an attacker has access to application information.
- Denial of service: Denial of service is where an attacker is able to disrupt a service so as to prevent a user from utilizing that service.
- Elevation of privileges: Elevation of privileges is where an attacker is able to access the application and then gain further, unauthorized access to additional parts of the application.

In the next chapter there will be a section dedicated to each of these threat model categories. The order by which they are presented will device from the S-T-R-I-D-E order. The most relevant categories will be presented first. Relevant threats in these categories will be introduced and briefly described. It may be the case that some threats will belong in more than one category, but

to conserve space, if a threat is introduced in an earlier threat category, it will not be repeated in a later category.

4.1 Spoofing

The act of spoofing BLE communication would mean that an attacker is able to pose falsely as another device. There are multiple opportunities for spoofing because BLE devices identify themselves in a variety of ways. These include:

- via device addresses on the advertising channel, including:
  - AdvA (for advertising devices)
  - ScanA (for scanning devices)
  - InitA (for initializing devices)
- via the access address on the data channel
- via the signature on the data channel
- via service selection offerings which can be communicated on both the advertising channels and the data channels
- via an application-specific mechanism

Each subsection below will describe each of the spoofing opportunities in more detail.

4.1.1.1 Spoofing Device Addresses on Advertising Channel

Device addresses on the advertising channel provide a way for other BLE devices to identify which devices are active on the channel.

- The AdvA address tells scanners which devices are broadcasting. BLE scanners can make decisions on whether or not to send a scan request or connection initiation requests based on an AdvA address.
- The ScanA address can be used by an advertiser to determine whether or not to respond to a scan request.

- The InitA address can be used to evaluate whether or not an advertiser should establish a connection with the initiating device.

As highlighted in chapter 2, there are four different methods of establishing these addresses on the advertising channel. Address are either: assigned Bluetooth device IDs, random static addresses, random private unresolvable addresses or random private resolvable addresses.

In the first two cases, spoofing the addresses is trivial because the addresses are static. A sniffer can be used to eavesdrop the advertising channels and observe these addresses. These addresses can then be programmed into a Bluetooth device and messages can be broadcast on behalf of the device which has the fixed address. In classic Bluetooth, this is known as “MAC spoofing” [MIN12]

This can be achieved at low cost with a normal BLE dongle and the Bluez stack. It can also be accomplished by installing an app on a mobile device like Nordic’s nRF Connect or iOS’s LightBlue Explorer.

In the third case where the address is random private non-resolvable, the randomness of the address does not detract from its vulnerability to spoofing. This type of address, however, is not meant to be resolved, so it is unlikely that another device will use this type of address to identify uniquely another communicating device.

In the last case, the case of the address’ being a private resolvable address, spoofing is a little trickier but not insurmountable. Private resolvable addresses are temporary addresses that are generated with a nonce and an IRK. New addresses are generated when an internal timer runs out or when the device is reset. In the case of AdvA addresses, this means that if an attacker is able to eavesdrop on the advertising channel and learn one of its resolvable AdvAs, it will be able to spoof that AdvA and continue to use it even if the original device’s timer runs out and it changes its AdvA. There is no replay protection built into the generation or validation of a private, resolvable address.

Both the ScanA and InitA addresses are a bit more difficult to spoof due to the infrequency of their broadcasts. A ScanA address is only present in scan requests. An InitA address is only present in initiating requests. This means that an eavesdropper must be present at the time of these requests to be able to sniff these values and use them. To facilitate this, an active attacker could use a previously sniffed AdvA address to spoof an advertisement that indicates that it can be scanned or connected to. This would prompt a scanning device to either send a scan request or a connection request and reveals its ScanA or InitA addresses respectively.

Another way that an attacker could spoof a resolvable private address is to gain access to the target device's IRK used to generate these resolve these addresses. There are a handful of ways to gain access to the IRK, and they are discussed in section 4.3.1.2 below.

4.1.1.2 Spoofing Access Addresses on the Data Channel

Checking the access address in packets on the data channel is one of the methods used to identify a peer device in a connection on a data channel. This value is exchanged in plaintext at connection establishment, and it continues to be sent in the clear even if the data PDU is encrypted. An attacker can sniff this address either by eavesdropping during connection establishment or by listening for the address on one of the data channels after the connection has been established.

There are a few challenges that an attacker must overcome in order to be able to use the access address consistently in a connection. The access address can only be utilized if the attacker knows the connection configuration of the communicating devices and can follow the connection. This information includes connection values like:

- the channel map (which channels will be used),
- the hop interval (how long a connection event will last),
- and the hop increment (which determines the order of channel hopping).

These types of values are established at one of two possible times. They can be established at connection initiation where all values are exchanged in plaintext. They can also be updated at any time during a connection via a connection update request from either the master or slave. Connection information can be updated either in plaintext or over an encrypted link.

The reason why it is essential for the attacker to know the connection parameters to be able to spoof on the data channels is due to BLE's channel hopping strategy and the limitations of sniffing hardware. It is challenging to intercept data channel messages because the messages can be sent over 37 different channels. There are devices that have the capability to scan all channels at one time such as the Ellisys Bluetooth Explorer. In academic research, Spill and Bittau showed that two USRP boards could be used to eavesdropped over 5 classic Bluetooth channels at a time [SPI07]. These solutions are cost prohibitive. Ellisys devices cost upwards of $10,000. The USRP solution proposed in the academic paper would require multiple USRPs which would also push costs upwards of $2000 These options will not be further discussed in this project.

In the lower-cost, ~$100 or less, more practical range are sniffers such as the Texas Instruments CC2540 Dongle, the Nordic nRF51-Dongle, the Adafruit Bluefruit LE Sniffer and Ubertooth. These devices only have the capability to listen to one channel at at time. These sniffers listen for connection establishment, gather the information exchanged at connection initiation to follow one single connection. This allows a sniffer to follow data transmissions over the different channels and eavesdrop *on one single connection between two BLE devices.* No other advertisements or connections can be overheard by the device at the time that a connection is being sniffed.

If a connection is already established and the initiating phase was missed, a sniffer would need to sample transmissions on a select set of channels and then rebuild the connection information from observed activity. The Ubertooth sniffer, for example, has the ability to rebuild connection information in the case that the original connection initiating phase has been missed [RYA13].

To summarize the process:

1. The BLE sniffer selects a data channel and sniffs packets only on that channel.
2. Once it receives packets, it reverses the CRC init value.
3. It then sniffs the same channel and measures the time gap between CRC-validated, captured packets.
4. From there the hop interval is determined where it is assumed that all 37 channels are being used. This is completed by solving the equation:

$$hopInterval = \frac{\Delta t}{37 \times 1.25 \text{ ms}}$$

Figure 42: hopInterval Calculation[RYA13b]

5. Once the hop interval is determined, a second channel is selected. The sniffer will listen to the first channel, when it detects a packet on the first channel it will hop to the second channel and wait for a packet. It then determines the amount of time that has passed between receiving the packets on the first and second channels.
6. From there the number of channels that were hopped between the first and second channel can be calculated with the following equation:

$$channelsHopped = \frac{\Delta t}{1.25 \text{ ms} \times hopInterval}$$

Figure 43: channelsHopped Calculation[RYA13b]

7. From there a lookup table can be used to determine find the hop increment used, and all of the variables needed now to follow the connection are known.

The weakness in the procedure above is that the Ubertooth only supports connections that use all 37 of the available data channels. According to Ryan,

all the devices that had been tested prior to his paper in 2013 had utilized all data channels in spite of the fact that the specification allows for fewer. A possible extension to the Ubertooth sniffer would be to assume the 37 channels by default, and after getting the communication parameters, follow the connection to validate that the parameters have been calculated correctly. If the Ubertooth is unable to sniff successfully and consistently over the mapped channels, then perform the calculation again with a modulus of 36 channels, and do the same type of verification as with the the 37 channels. The complicating factor about doing this type of verification is that any of the 37 data channels can be the channel that is not used. This means that for a connection that uses 36 channels, in the worst case scenario it would take 37 attempts to calculate and verify a connection.

Once the connection parameters have been identified, an attacker could theoretically send messages on behalf of either the master or the slave by spoofing the access address. Given the limitations of current BLE sniffing/transmission technology and the timing requirements of this type of attack (see section 4.2.1.2), this type of attack would require a significant amount of effort to develop.

4.1.1.3 Spoofing the Signature on the Data Channel

During pairing, two devices can agree to exchange connection signature resolving keys (CSRKs). The CSRKs are used to generate MACs over a counter and a data PDU to ensure authenticity of messages transferred from one device to the other. The receiving device verifies the MAC, and if validation is successful, it continues to process the message and stores the latest counter value.

According to the NIST publication, “Guide to Bluetooth Security”, there are two ways that the CSRK can be established. It can be established by the generation of 128 bit random numbers which are then stored in a database on the device or derived from a 16 bit key diversifier and a stored ER key [SOU13]. According to the documentation, however, the CSRK can be assigned at time of manufacturing [BLE-SMP, 40], or it can be generated

from a key diversifier function, d1, that takes a stored, ER value and a key diversifier value [BLE-SMP, 71]. The difference between the NIST description and the BLE specification description is that in the first case of key assignment, NIST assumes that there will be multiple possible CSRKs assigned whereas the BLE specification only specifies one key. Assuming the specification is correct, the two different methods of establishing the CSRK provides at least two different methods of acquiring the CSRK for spoofing purposes.

- In the case that the CSRK was established at time of manufacturing, and it is a fixed, single value, an attacker could conceivably pair with the target device, request the exchange of the CSRK and then have the CSRK at his or her disposal. Then the attacker would be able to use that CSRK to create and sign messages.
- In the case that the device generates a new CSRK from the ER and a key diversifier value, the procedure that generates the CSRK could be poorly managed. For example, the device could mistakenly use the same diversifier for each, new CSRK exchange. In this case, the attacker would use the same method as above where the CSRK was fixed at the time of manufacturing. Pair, request CSRK, collect and use key.
- If each CSRK is generated with a new diversifier, however, this makes the task of an attacker much harder. Unless there were an obvious way to predict the key from the device, the attacker would need to be able to eavesdrop on the key exchange at the time of pairing. There is limited opportunity to do this, and it will be addressed in the section on confidentiality.

4.1.1.4 Spoofing via Service Offerings

When an advertiser broadcasts that it is accepting connection requests, a scanner device can determine if it wants to establish a connection by reviewing service offerings being broadcasted by the advertiser. There is the

possibility of an advertiser to specif which services it desires from the master, and which services it can offer a master. Based on this information, a scanner can make a decision as to whether or not it will initiate a connection with the advertiser [TOW14].

A way that an initiator can determine the service offerings prior to establishing a full pairing session to a slave is to establish a non-paired connection. In this respect the initiator becomes the master and the advertiser becomes the slave. Then, the master can request a list of the available GATT services of the target slave. If the slave does not offer the services the master is interested in, then the master can opt not to pursue further communication with the slave. To spoof the slave, then, it may be necessary to also clone the slave's services.

4.1.1.5 Spoofing via Application-Specific Mechanisms

Specific applications may add additional mechanisms for the provision of authenticity that go beyond the specification of Bluetooth low energy. These will need to be evaluated on a case-by-case basis to ensure that they provide appropriate protections against spoofing.

4.2 Tampering

Tampering with data over-the-air by manipulating the signal directly is not feasible with the equipment available to a standard security analyst. Instead, to tamper with data, an attacker would need to spoof the identifying characteristics of the target device's communicating partner and overcome the data integrity mechanisms in use. The data integrity on BLE channels is protected by four different mechanisms:

- via the BLE packet exchange protocol itself
- via a BLE packet CRC
- via a BLE data channel PDU signature
- via BLE data channel PDU encryption/MIC

Each of these mechanisms require an attacker to use different strategies to be able to tamper with communication.

4.2.1 Tampering and BLE Packet Exchange Protocol

The packet exchange protocol can facilitate and also inhibit message tampering depending on whether data is being exchanged over the advertising channel or the data channel.

4.2.1.1 Advertising Channel

Over the advertising channel, if a device spoofs the AdvA address of a device, it could:

- broadcast false data values for consumption by a scanner,
- broadcast connection availability and trick an initiator into establishing a connection with the wrong advertiser.

In general, if two devices broadcast on a channel at the same time, their broadcasts will most likely cancel each other out. This, however, is not a huge problem on the advertising channels. This is because there are no timing requirements that inhibit communication over the advertising channels. There is a certain degree of coincidence that an advertiser will be transmitting and scanner or an initiator will be listening at just the right times [TOW14]. Further, Bluetooth devices, especially advertisers, are created to conserve energy. Hence, advertising devices will limit the rate of broadcasting to save battery power. An attacker can take advantage of the unused air time to spoof the advertiser and broadcast packets more frequently on its behalf thereby increasing chances that it will be heard by scanners and potential initiators [JAS16].

4.2.1.2 Data Channel

The packet exchange protocol on the data channel is more challenging because of the timing requirements for packet exchange. The exchange of packets is organized by “connection intervals”. A connection interval is the amount of

time that communication will take place on one channel before the hop to the next channel. At the beginning of a connection interval, there is what is called a “connection event”. This is a point of synchronization between the master and the slave. The master sends the first packet on a new data channel, and before it can send any further packets on that channel, the slave must respond. A slave has the opportunity to respond or ignore the packet sent at the connection event. A value called “slave latency” defines the number of connection events that a slave can skip before the connection times out. If an exchange of data does occur on a channel, it does so in a master-slave, master-slave fashion. Between two packets being sent between master and slave, there is a short wait time called “interframe space” or “T_IFS”. This is a 150 microsecond window of time.

A connection interval ends, and the master and slave hop to the next channel when one of the following conditions hold true.

- Time has run out on the connection interval, and a new connection event must be started.
- Both the master and the slave have set the MD field in the data channel PDU header to 0 which indicates that there are no further messages for this channel.

To be able to tamper with the data on the channel, an attacker would spoof one of the devices in the connection by using the connection’s access address, determine the connection parameters, follow the connection hops and then transmit data during a connection interval. Because the timing of the messages and the back-and-forth nature of packet exchange, an attacker needs to find a slot for transmission that is empty. Otherwise it runs the risk of transmitting concurrently with the master or slave and creating a denial of service condition.

To compromise the integrity of communication over the data channel, an attacker would need to utilize the space between packet transmissions. Figure 44 shows a logical representation of master and slave packet exchanges over two channels.

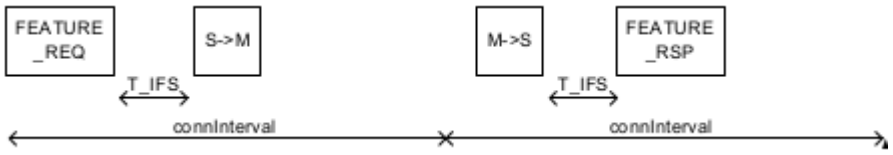

Figure 44: Example of Packet Exchanges on a Connection [BLE-LL, 100]

As can be observed in the figure, there can be some empty space after the master and/or slave packet transmissions. This can be due to a variety of factors:

- The slave may be in a mode to ignore the master on the connection interval. In that case, the slave would not respond to the master's connection event transmission.
- A transmitted packet may not leave enough space on the connection interval for the next device in the communication to be able to fit the next packet in the space, so it must wait to the next interval.
- The master and the slave may have been finished communicating on a particular connection interval, but they mistakenly did not transmit packets with the MD header bit set to 0.

An attacker can spoof the connection access address, and then send messages in these spaces. These messages would need to fit the context of messages being exchanged at that point and time or run the risk of being rejected.

There are a couple of strategies that an attacker could use to take over the connection context so that it would not have to compete with the device that it has spoofed.

- An attacker could spoof a slave device, sniff the connection, and then forge a connection change request. The master would then update the connection parameters and start communication directly with the spoofed slave device instead of the original slave. The attacker would then be the slave and send data packets to the master.

- An attacker could spoof a slave packet by eavesdropping the access address, and then forge an end connection request on behalf of the slave. The master packet would then end the connection. The attacker could then do one of two things:
    - He or she could spoof the slave's AdvA address, and flood the advertising channel with advertisements that support connections. The master could then establish a connection to the attacker's slave device, and the attacker could communicate exclusively with the master.
    - He or she could spoof the master's AdvA address and listen for advertisements from the original advertiser. The attacker could then establish a connection with the advertiser on behalf of the original master and send packets to the original slave.

4.2.2 Tampering and BLE Packet CRC

BLE packets have a cyclical redundancy check value that is appended to provide some assurance that packets are not modified during transmission. The equation and input values are generally know with one exception, namely the CRCInit value. On the advertising channel, the CRCInit is 0x555555. On the data channel, the CRCInit value is negotiated via the CONNECT_REQUEST PDU.

The CRCInit value that is used on the data channel can be ascertained in one of two ways. The first is to sniff the value at connection establishment. This value is transmitted at this time in plaintext. The second way to do this is to brute force the the CRCInit value. This can be performed quickly with CRC reverse engineering software such as reveng. reveng has, in fact, a Bluetooth low energy configuration [COO17].

4.2.3 Tampering and the Data Channel PDU BLE Signature

The BLE signature is used to provide assurance of authenticity on a connection that is not encrypted. As highlighted in the spoofing section, the signature is created with a combination of the contents of a data PDU, a signature counter and the CSRK. The signature counter is set to 0 when the CSRK is established, and it is incremented each time that a message is signed. The counter is employed to provide a level of assurance of freshness to sent messages. There are two ways that the signature can be compromised so that an attacker can tamper with messages sent.

The first tampering method is to gain access to the CSRK. These options are addressed the spoofing section. Once an attacker has access to the CSRK, all that the attacker must do is eavesdrop on the last packet exchange where the CSRK was in use to gain access to the latest counter value. Once the attacker has access to both the CSRK and the correct counter value, then the attacker would be able to tamper with message transmission on a connection.

The second potential tampering threat involves three devices. Suppose device A and device B have paired, during which time device B had sent its CSRK to device A. An attacker then uses device C to spoof device A and to establish a connection with device B. Device C then engages in packet exchange with device B where device B sends a series of signed messages that are recorded by device C. Device C then disconnects from device B, spoofs the identity of device B and establishes a connection with device A at a later time. Device C can then replay the recorded messages from device B.

The first threat is the more dangerous and practical threat of the two. An attacker that has access to a CSRK can create desired messages and the receiving device will accept them without the knowledge that they have been forged. The second threat is more theoretical, and its success is dependent on the application logic and the kinds of signed messages that could be elicited and recorded by an attacker.

4.2.4 Tampering and Data Channel PDU Encryption/MIC

As discussed in chapter 2 and 3, an encrypted session can be established on over a BLE connection. This link's encryption and MIC are employed to provide confidentiality and authenticity of communication. If an attacker can compromise the cryptographic mechanisms as described in section 4.3.1.2, then it is possible to tamper with transmitted data. There are three attacks feasible on an encrypted channel.

The first attack that is feasible is for an attacker to gain access to the LTK established during pairing. This key is used to derive the session key that is used to encrypt the data channel PDU and to create a MIC over the PDU. With this key an attacker has two possibilities. First and foremost, if the attacker is able to spoof the access address of the device and use the LTK, the attacker would be able to establish a connection with a target device and send forged messages to the device. The other possibility for an attacker would be to listen during connection initiation and eavesdrop during session key establishment for the initialization vector and key diversification value exchange, and then inject packets on the connection as described in section 4.2.1.2. Methods of gaining access to the LTK are discussed in sections 4.2.5 and 4.3.1.3.

4.2.5 Tampering and Man-in-the-Middle

The final threat under the category of tampering is the threat of a man-in-the-middle attack. In a man-in-the-middle attack, the attacker must have a computing device and two BLE dongles. Figure 45 shows the basic anatomy of the attack.

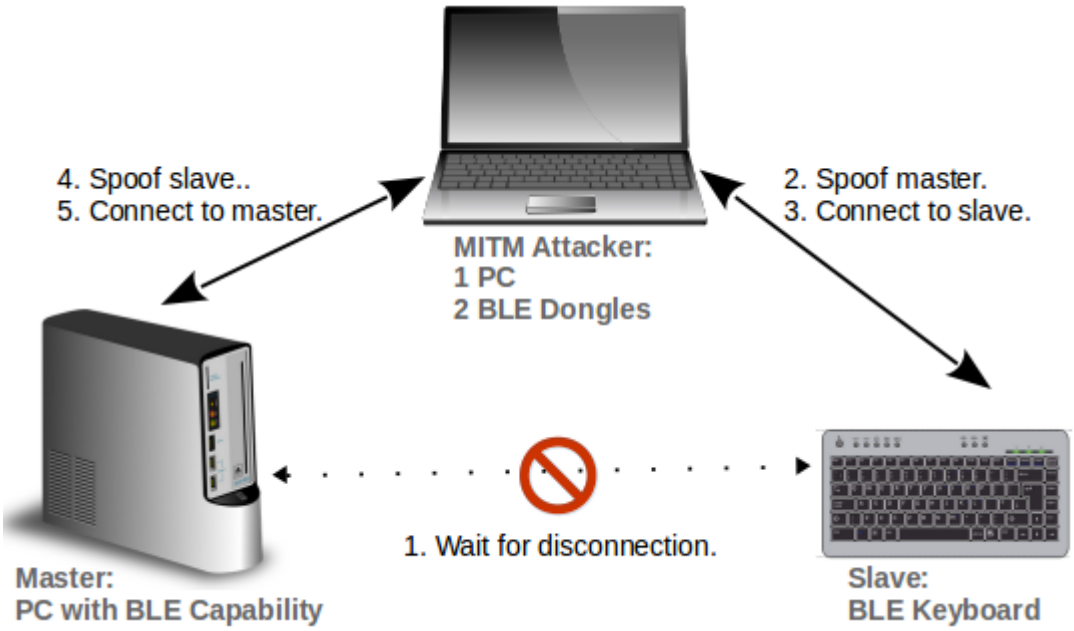

Figure 45: Logical Representation of a BLE Man-in-the-Middle Attack

The setup of the attack is partially dictated by the packet exchange protocol on the data channel. The following steps describe the execution of the attack in more detail:

1. First, the attacker must be able to identify two communicating devices and clone those devices.
2. Second, the attacker must wait for the connection between the devices to be broken, instigate the end of a connection as discussed in section 4.1.1.3, or inject interference on the data channels and thereby cause the connection to break down.. In the example above, if the PC were powered down, the connection would be broken.
3. Once the connection is broken, the slave device will periodically send advertisements with connection information. The attacker will then establish a connection with the slave using the cloned information from the master. Since slave devices can only be connected to one device at a time, there is no way that the slave can pair with the original master once it is connected to the attacker's device.

4. Once the attacker has connected to the slave device on its first BLE interface, it will begin sending out advertisements using the cloned data from the slave on its other BLE interface.
5. Once the master device is started again, it will receive the advertisements from the attacker's clone and initiate a connection.

Once the connection is established, the attacker will have to forward the communication between the two BLE attack interfaces. In doing this, the attacker is able to eavesdrop on all of the communication between the master and the slave. The attacker will also be able to manipulate the data being passed back and forth. There are a handful of notable researchers who have demonstrated this is possible. Ryan demonstrated this type of Man-in-the-Middle attack in his talk, "NSA Playset: Bluetooth Smart". In this talk he demonstrated how a man-in-the-middle attack could be performed on a BLE mouse (the slave) and a laptop (the master) [RYA14]. Jasek has introduced a tool called GATTACK which supports man-in-the-middle attacks by facilitating communication between the attacker's BLE interfaces via websockets and node.js [JAS16]. Cauquil has also introduced the Btlejuice tool that also leverages node.js for BLE man-in-the-middle manipulation [CAU16].

4.3 Information Disclosure

Confidentiality for Bluetooth low energy is ensured via the use of encryption and private, resolvable addresses. There has been a significant amount of work in the field concerning confidentiality and privacy issues in the area of BLE. Information disclosure, a breach of confidentiality and privacy, can be broken up neatly into two main categories:

- data exposure: This has to do with the exposure of confidential data during transmission.
- trackability: This has to do with the how easily a device in use can be used to track individuals over time.

Underneath each of these two categories are a number of options for attack.

4.3.1 Data Exposure

4.3.1.1 Unencrypted Transmissions

The BLE specification allows for the transmission of data in plaintext. The specification does not natively support encryption on the advertising channel. If the data is encrypted, the encryption is being performed at the application layer.

One type of related attack that is common in classic Bluetooth is called "Blueprinting". This is where information such as manufacturer, device version and firmware versions can be collected from a Bluetooth device without any authentication [MIN12]. This type of information can be used by an attacker to determine whether a device is susceptible to known, published attacks. And, while this type of attack was developed classic Bluetooth, the attack is also available, at least in part, for BLE. During advertising, there is an optional field defined for manufacturer-specific data in the advertising data payload. This, along with the listing of services advertised, could be used to determine the manufacturer and other implementation-specific details about the device. These features are largely available in the RaMBLE app for Android [LES16].

On the data channel, data can also be exchanged also in plaintext. If an attacker is able to sniff a connection as described in section 4.1.1.2, then the attacker will have full access to the data exchanged.

4.3.1.2 Pairing BLE Legacy

The purpose of pairing is the establishment of an LTK and the possible exchange of an IRK and CSRK. In his paper, "With Low Energy Comes Low Security", Mike Ryan provides an analysis of the pairing exchange for all of the different type LE Legacy pairing, ie. just works, passkey entry and OOB [RYA 13b]. His analysis points out that almost all of the key material for the establishment of the LTK is exchanged in plaintext between two devices.

There is one value that is unknown to an attacker, and this value's characteristics is dependent on the pairing method.

In the case of just works and passkey entry, the unknown value is a six digit value. In the case of OOB paring, the value is up to a 128 bit value. The six digit value is easily brute forced, and it is therefore trivial to be able to gain access to an LTK that has been established using passkey entry or just works. If during OOB pairing, an unpredictable, 128 bit value is used, then the LTK is safe because it would take too long to brute force 2^128 values. If the value is weak, predictable or the transmission channel is compromised, then the attacker would then be able to establish the LTK value.

If an attacker is successful gaining access to the LTK at the time of pairing, then it is possible to also be able to decrypt the data exchange when the IRK and CSRK are also exchanged. This has implications for tampering as discussed above in sections 4.2.2-4, and it also has implications for privacy which is discussed in section 4.3.2.3.

Access to an LTK is not sufficient to be able to sniff further, new sessions on the BLE link. This is due to the fact that when a new session is established, a new IV and new key diversifiers are exchanged to establish a session key. If an attacker misses the values at session establishment, then the attacker will not be have access to the session key. The attacker can get by this by interfering with communication and forcing the connection to drop and restart. On reconnection a new session will be established, and the attacker will be able to sniff the new session key values.

4.3.1.3 Pairing and Bonding BLE Secure

Accessing the LTK over-the-air with LE Secure is more challenging than in LE Legacy. In LE Secure, an ECC Diffie-Hellman exchange based on a NIST P-256 curve is used to establish the shared secret used in the generation of the LTK. Due to the size of the shared secret, 2^256 bits in length, it cannot be reasonably brute forced. That being said, the NSA dropped its support of the NIST P-256, although the reasoning behind its move away from the standard is not wholly clear at this time [KOB16].

Although ECC is used, there are two weaknesses to which BLE Secure pairing and bonding are subjected. The first weakness has to do with a debugging feature of BLE. BLE has a debug mode, and in this mode, both devices use a predefined set of Diffie-Hellman values for pairing (see figure 46).

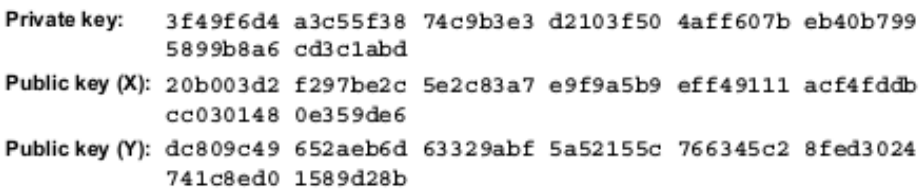

Figure 46: Debug Diffie-Hellman Values for LE Secure [BLE-SMP, 615]

If these values are used, the LTK can be recovered by eavesdropping on the rest of the publicly-exchanged values at the time of pairing.

The second weakness can be found in just-works, passkey and numeric comparison pairing. This type of pairing does not protect against man-in-the-middle attacks. The values in these authentication protocols are exchanged in plaintext. An attacker could potentially capture these values and calculate the correct confirmation values. Using this technique, any keys exchanged during pairing can be eavesdropped.

4.3.1.4 Side Channel Attacks

Even if the data PDUs are encrypted, an attacker can gain information about the information being exchanged by looking at the characteristics of the packets that are exchanged. The headers of the packets and the control data are never encrypted. This information can provide a great deal of information about what is contained in the BLE packets. In addition to this, there is packet length, rate of packet exchange and patterns of packet exchange that could provide information for an attacker. For example, Das et.al. performed a study that looked at various BLE devices and the data that could be discerned via these side-channel methods. In one experiment, they sniffed the packets transmitted by a fitness tracker. The size of the packets were enough for the

researchers to determine which activity was being reported to the connected application [DAS16].

4.3.1.5 Poor Key Management

The NIST guide provides list of common weaknesses in key management with respect to BLE. The first is that link keys can be stored improperly. If they are easily reachable via an app interface or via request over-the-air, then they are susceptible to theft and use. Another weakness in key management is that a BLE device's pseudorandom number generator may not provide a sufficient amount of entropy. This is likely due to the fact that many BLE devices are working with resource-constrained hardware. If the randomness can be predicted or is periodic, then the the keys can be compromised.

And, last but not least, the STK key lengths themselves are negotiable. The length of the key that is used in cryptographic calculations will always be 128 bits. But, the negotiated key value can be as small as 7 bytes (56 bits). If a 7 byte value is negotiated between two devices, then at the point of encryption it is prepended with 0 values until it is 128 bits long to satisfy the required 128 bits for AES. Naturally, the smaller the key length, the easier it is to crack the key value [PAD13].

4.3.2 Privacy

Privacy is one of the most scrutinized points of Bluetooth low energy. This is most likely due to the fact that privacy is the easiest aspect of BLE to analyze. There are four aspects of privacy that need to be explored with respect to data exposure [ALB16][DAS16][FAW16][HIL16][LIN16][WOO15][ZIE14].

4.3.2.1 Data on the Advertising Channel

As people move around with their headphones, beacons and smart watches, these devices are often in broadcast mode on the advertising channel. When a device is not connected to its master device, it advertises from time to time with a message indicating that connection is possible. In addition to this, other data can be transmitted such as the name of the device, the manufacturer, the

name of the user, services offered etc. If the transmitted information is unique in nature, it can be tracked for up to 100 meters with only a BLE dongle. A device does not have to be in eyesight to be able to be tracked or located.

4.3.2.2 Public and Random Static Device Addresses

The most common way of tracking a device, however is via the public device address or the static device address on the advertising channel. This address is fixed and unique to the device, and it is broadcast with every packet on the advertising channel. The Android app, RaMBLE, was developed to provide an inventory of the BLE devices broadcasting on the advertising channel. For tracking purposes the app is useful in that it keeps a history of the times that a device has been detected, and a map of where devices have been detected [LES16].

4.3.2.3 Private, Resolvable Addresses

Resolvable, private addresses, introduced in chapter 2, are used to reduce the trackability of a device. And, although privacy has been a key point of criticism of BLE and Bluetooth use, not all manufacturers have implemented it. For example, manufacturers Fitbit and Basis have indicated that they would like to implement LE privacy, but due to the “fragmented Android ecosystem” they could not implement it because of lack of systematic LE Privacy support [HIL16].

Resolvable, trackable addresses are used by a device for a limited time period, and then a new one is generated. The variables that are used to generate the address are the IRK and a 24 bit PRAND which always begins with 01. The algorithm that generates the resolvable private address generates a 24 bit MAC, and then it concatenates the RAND and the MAC together and sends it as the device’s address. Resolvable private addresses can be used for AdvAs, ScanAs and InitAs.

The private, resolvable address can be attacked in a variety of ways. The first way is to exploit the weaknesses identified in both LE Legacy and LE Secure

pairing to gain access to the IRK. If an attacker has the IRK, then it can be used to identify a device from all others.

The second way to attack the resolvable address is to take advantage of the poor key management in the specification. The IRK can be established in one of two ways. It can be hard-coded at the time of manufacturing, or it can be derived from a hard-coded IR value and a pseudorandomly generated key diversifier. If the key is fixed, then any device that engages in pairing and exchanges IRKs will have access to the IRK value.

The third way to attack attack a private, resolvable address is due to poor implementation of the address itself. The device should keep an internal timer that tracks the amount of time that a device has used its current address. If an attacker could find a way to attack the timer, or if the the timer is set for a longer period of time, an attacker would be able to track the fixed address.

4.3.2.4 Long Range Surveillance

The maximum transmission range of BLE in the 4.2 specification is 100 meters. The information that is broadcasted can be used to track a target. Tracking a target by BLE may not seem too advantageous in comparison to visually tracking. That is, if the target is in eyesight. If the target is behind a wall or in a building, then confirmation of presence via the detection of a person's BLE device could provide an attacker key information.

To extend the range a standard Bluetooth signal, Cheung and his team built a device called the "Bluesniper" [CHE 05]. His team was successful in establishing communication over a .5 mile distance by modifying a Bluetooth dongle and adding a long-range directional antenna. This allowed signals to be received over extended distances, but active transmission of signals was tricky because of timing. In order to be able to get known, active Bluetooth exploits to work, the exploit software had to be modified to take into consideration the timing limitations introduced by the long distances [CHE04].

At this point and time, no one has made any devices that are tailor-made for BLE public that perform the same type of amplification and range. But, it is

conceivable that such a device could be developed. This device could then be used to passively scan for target devices advertising either identifying address or unique information in an attempt to locate an individual.

4.4 Elevation of Privileges

Elevation of privileges is the act of gaining access to an application and then exploiting vulnerabilities to gain further unauthorized access to the application, the underlying system or resources provided by the system. There is a variety of approaches that an attacker can use to find and exploit these vulnerabilities.

4.4.1 Exploitation of Exposed Attributes

Each attribute that is stored on a BLE device and is accessible via the GATT layer can require a specific level of security: encryption, encryption and authentication, authentication with MITM protection and authentication without MITM protection. Each attribute can have read and/or write characteristics. An attacker can establish a basic connection to another BLE device and the iterate through all of the handles or UUID values that are available on the connected device to determine actions that can be performed on the discovered attributes. This can be accomplished via a simple python script or the gatttool and a BLE dongle. If an attribute is not properly assigned the appropriate security level, an attacker could access and even modify values that should not otherwise be available.

4.4.2 Fuzzing Attacks

A second way that attackers often find ways to escalate privileges is through the process of fuzzing. Fuzzing is the act of sending messages to an interface with unexpected values in an attempt to see how the application behind that interface reacts. Fuzzing can result in unexpected application behavior, buffer overflows and the discovery of application backdoors. BLE fuzzing can be broken down into two levels: GATT/ATT profile fuzzing and BLE protocol fuzzing.

4.4.2.1 GATT/ATT Profile Fuzzing

At the GATT/ATT level of the BLE stack, data is defined and organized into services as described in chapter 2. As with all services, the applications expect that the data exchanged at the service level have a certain type, length, range of values, etc, and these values can be fuzzed.

4.4.2.2 BLE Protocol Fuzzing

Theoretically, the BLE protocol layer could be fuzzed. In the specification several value ranges are defined for different control bits in a packet. Sometimes the value ranges do not cover the entire set of possible numeric values that could be represented by the defined bit range. Values outside of these ranges could be submitted to see how the controller or the host react. Relatedly, there are several points in the specification where there are bit values which are designed as “reserved for future use” (RFU) fields and should contain 0 values. Non-zero values could also be submitted in the RFU fields to see how the controller or host responds.

Fuzzing at this level is complicated by the fact that a standard BLE dongle should only be able to communicate using well formed packets. For this type of attack at lower level layers, special hardware and firmware would be needed so that more control over the implemented protocol stack would be available. At the GATT an ATT levels, the PyBT python library has been developed to provide the opportunity to send packets that do not conform to the specification [RYA17].

4.4.3 Injection Attacks

Another way to escalate privileges is through typical injection attacks such as SQL injection, code injection, LDAP injection, etc. In this case, the attacker would transmit injection values over-the-air to a master device with the intent that this value would be passed further to upstream processes where the input would be interpreted by the application at that stage of the process. Example goals of this type of activity would be to execute commands in an application

further upstream or to cause the application to reveal information system details such as configuration information or login information.

4.4.4 Brute Force Attacks and Whitelists

On the advertising channel, a device has the possibility to set up a whitelist. The whitelist contains either the AdvA, InitA or ScanA addresses that are allowed to communicate with the device on the advertising channel. If an attacker wants to be able to communicate with a target device with a whitelist, then the attacker would need to acquire the device address saved in the whitelist so that it can be spoofed. This can be accomplished in one of two ways. This could be accomplished by sniffing traffic on the access channel and tracking the addresses exchanged there. Or, alternatively, if a device in the whitelist is a standard Bluetooth device ID, and the attacker knows the manufacturer of the device, the attacker might be able to brute force the address [MIN12]. The first 3 octets of a Bluetooth device ID are fixed for a manufacturer. The attacker would simply need to iterate through the second 3 octets until the Bluetooth device ID could be determined.

4.4.5 Replay Attack

In a replay attack, the attacker records communication between two devices. If the values exchanged are fixed each time, such as a password being passed in plaintext or even being passed in an encrypted form without replay protection, then the attacker can spoof one of the target devices and replay the information that has been captured during the exchanges. The NCC group has developed a set of Python scripts to facilitate this type of testing for mobile-app-based BLE systems. These scripts are dependent on a modified version of the pygatt python library and apps that can run in an Android device in developer mode.

4.4.6 Reflection/Relay Attack

In a reflection attack, an attacker simply relays signals from one BLE device to another without any knowledge of the application values that are being

exchanged [MIN12]. To clarify how the attack works, a theoretical scenario will be presented. Consider, for instance, a theoretical BLE-controlled lock. The lock has been paired using LE Secure with a mobile phone. The lock has two actions that can be performed upon it.

- Lock

  It can be locked via a button press from an app on the mobile phone.

- Unlock

  The lock automatically unlocks when the owner (with the mobile phone) gets within a 10 meter range of the lock. The lock recognizes the mobile phone via a cryptographically sound challenge-response protocol that is protected over an encrypted BLE link. The exchange takes place “hands free”, i.e. without intervention from the user.

An attacker could use reflection to open the lock without the owner of the lock being within the 10 meter range. In this scenario, the lock is the slave, and the mobile phone is the master device.

To mount a reflection attack, an attacker needs to be able to extend the transmission of the BLE signal in some way beyond the 10 meter range of the unlock functionality. Figure 47 shows a logical representation of the communication configuration of a reflection attack.

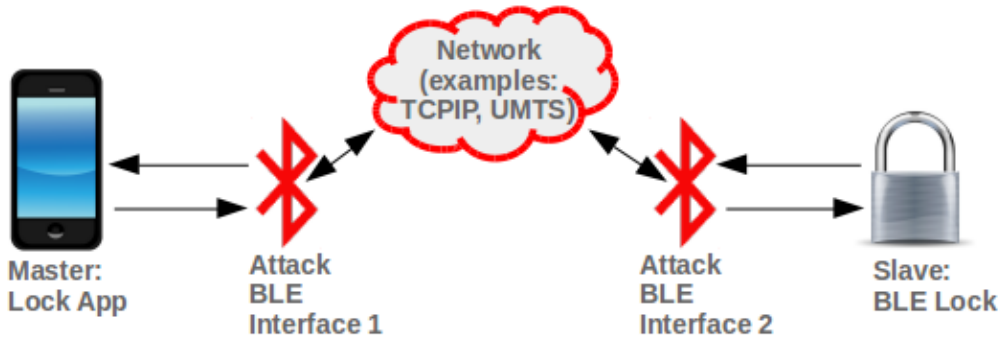

Figure 47: Logical Organization of a Reflection Attack

In this setup, the attacker will place an attack BLE interface near the target device (the lock) and an attack BLE interface near the victim (the mobile phone). A reflection attack can occur in the following way.

1. When the mobile phone is no longer in range of the lock, the connection is broken. The lock will begin to broadcast advertisements with invitations for connection.
2. The attack BLE interface in range of the lock (BLE interface 2) intercepts these advertisements.
3. This BLE interface then forwards the unmodified packets to the other attack BLE interface in the proximity of the the mobile phone (BLE interface 1).
4. BLE interface 1 broadcasts the advertisement unchanged.
5. The mobile phone in this case periodically listens for advertising packets from the lock. When it receives an advertisement from the lock that has been reflected by the attacker's BLE interfaces, it moves into the state initiating and transmits a connection request.
6. BLE interface 1 intercepts the request, collects the connection information and prepares to follow the channel hopping on the data channel.
7. After this, BLE interface 1 forwards the request over the longer-range network connection to BLE interface 2.
8. BLE interface 2 collects the connection information as well so it, too, can follow the channel hopping during the communication.
9. BLE interface 2 then forwards the connection request to the lock.

These types of exchanges will be successful for completing connection setup, establishing encryption and performing the challenge-response protocol. The attacker does not need to know any of the content of the messages. The attacker only needs to be able to follow the connection so that no message exchanges are missed. Eventually, once the challenge-response protocol has

been completed, the lock opens, and the attacker has gained access to whatever the lock was protecting.

In this type of attack timing will be a critical factor. Communication delays because of the time it takes to relay signals from one attacker BLE interface to the other may introduce too much lag for a successful attack.

4.4.7 Application Logic Exploitation

Elevation of privilege is most likely achieved through exploitation of application logic. Functionality that was supposed to be used for one purpose is then used for another.

An example of application logic exploitation can be found in a BLE application that manages customer “loyalty points”. To get loyalty points, customer of certain establishments installed an app on their BLE-enabled mobile phones. Then, when a customer would come into those target establishments, the mobile phone app would receive a message from a BLE iBeacon in the store, and it would increment the customer’s loyalty point counter. After a customer’s loyalty points reached a certain level, the customer would be eligible for special services, free of charge.

In this case, an attack was performed by recording the data that was being transmitted from the iBeacon. The device ID was then spoofed, and the same data was broadcast by the iBeacon was broadcast by an attack device. This was sufficient to increment the loyalty point counter and thereby gaining unauthorized access to services for free [JAS16].

Elevation of privileges through the exploitation of business logic is wholly dependent on the implementation of the application. And, while there are few examples of this type of exploitation that have been publicized for BLE applications, as the types of BLE application features expand, so will the attack surface of BLE applications.

4.5 Denial of Service

There are essentially three ways of creating a denial of service condition for a BLE application. An attacker could:

- cause electrical interference on a channel where BLE devices are broadcasting
- connect to a device with the intent to drain the battery
- user application/protocol logic to cause a connection to be interrupted.

4.5.1 Electrical Interference

Availability on the over the air interface can be impacted by electrical interference. To combat interference under general conditions, BLE employs frequency hopping and provides rules for retransmission in the case that a received packet's CRC check fails. This being said, an attacker can strategically create a denial of service condition. This, as with confidentiality and integrity, can be broken down into the two different types of channels: advertising and data.

On the advertising channels, an advertiser broadcasts a packet on one of three advertising channels, then moves to the next advertising channel and broadcasts again. It continues like this until it internally signals that broadcasting should stop, or until it receives a connection request. The rate at which the advertiser broadcasts will vary depending on the device. For a scanner to receive a packet, it must be coincidentally on the same channel at the same time as the advertising packet is broadcast.

An attacker can reduce the possibility of connections being established or advertising data being sent by simply transmitting a signal on one of the channels. If an attacker had three devices, then all three channels could be blocked.

On the data channels, as with confidentiality and integrity, the challenge to deny availability on the airwaves is greater. Once again, an attacker would need to capture the initiate connection request to gain knowledge of which

channels will be used and when for communication. When the devices with the connection move from one channel to the next, the attacking device will also need to move to the same channels and broadcast a signal on those channels to disturb any data communication.

4.5.2 Battery Drain

In classic Bluetooth, there is an attack called the “L2CAP guaranteed service attack” [MIN12]. This is an attack where during connection setup either the highest data rate is established or the lowest latency is established. This is to force the target device to actively communicate as much as possible to drain the battery. In theory, this attack is feasible for BLE as well. There are two fields in BLE connection establishment that correspond to classic Bluetooth and could have a negative impact on battery-life: interval and latency. Interval is a value that specifies how frequently channel hopping occurs. A high value would force a master to broadcast at every new connection event at the highest rate possible. And, if the slave latency is set to the lowest possible, then a slave would not be able to ignore the connection events at each connection event and would have to broadcast a response. This high rate of required communication could cause battery drain.

In addition to this, battery drain may be instigated by misuse of an application. If an attacker is able to create a set of communications to prompt at BLE application to execute a resource-intensive operation, it, too, will result in battery drain.

4.5.3 DoS via BLE Protocol/Application Logic

Either through the BLE protocol itself or through a BLE application’s logic, there is the risk that an attacker can exploit a vulnerability and create a DoS condition. This can occur, for instance, when two values are incompatibly set. For instance, there are two values that can be set at connection establishment: (master) timeout and (slave) latency. The master timeout specifies how long a master will wait for a response from a slave before considering the connection to be lost. The slave latency specifies how long a slave does not have to

respond to a master after the first message of a connection event. The specification states that the slave's latency period cannot exceed the master's timeout period. If, however, the BLE specification has not been implemented completely, and a check has not been built in to validate these values at connection establishment or renegotiation are not incompatible, then an attacker could inject an LL_CONNECTION_UPDATE request, and thereby create the conditions for a loss of connection.

4.6 Repudiation

Repudiation is the ability of one device to ensure that a communication originated from another device at a specific time and that the message was not tampered with in any way during transmission. Repudiation is usually initiated as a legal requirement rather than a technical requirement. The mechanisms that provide this type of assurance are usually digital signatures.

Within the BLE protocol, there is the provision for a "signature" for authentication which provides some assurance against tampering. In an encrypted link there is also the MIC that provides a cryptographic level of assurance that a message has not been tampered with during transmission. But, as identified in the tampering subsection above, each of these mechanisms leaves room for replay, man-in-the-middle attacks and key theft. If an attacker is able to gain access to the cryptographic keys using the methods identified in the information disclosure sections and the tampering sections above, then repudiation is not guaranteed.

What about, then, instances where an attacker is unable to access or guess the keys over the air link? Repudiation is not only about the protection of tampering over the air link, it is also a question of key management. If the keys involved in signing messages are not protected from external access, modification and loss, then provision of repudiation cannot be claimed. For example, key materials can be securely exchanged OOB without an attacker's gaining access to them. Messages can be signed and exchanged using keys derived from this key material. But, if the device is reset, the key store may be

reset with it and hence any derived keys. Reset is device dependent and can be initiated by things like a reset button, loss of power and device defect. Unless there is some additional way of associating a key with a device or preserving the key history of a device, the source of any message that is only protected with BLE mechanisms can be repudiated.

If a device has a requirement for the provision of repudiation, the BLE specification does not directly provide for it. Repudiation will have to be designed and maintained at the application level.

4.7 Other Threat Models

There has been little work done in modeling the threats against generic BLE devices. During research, there were three papers identified that provided a attempted to provide a comprehensive set of threat categories for Bluetooth and an analysis according to those categories. These threat models are reviewed here briefly analyze the problem of threat modeling from different angles and to identify any potential threats that did not surface in the STRIDE model above.

In Minar and Tarique's paper, "Bluetooth Security Threats and Solutions: A Survey", the researchers provide an analysis of the Bluetooth 4.0 attack surface and mention BLE. There they introduce three types of threats [MIN12].

- Disclosure threat
- Integrity threat
- Denial of service threat

After their analysis, they introduce a proof of concept for a BLE proxy that attempts to make provisions for protecting against the threats identified in their analysis. The threats that they identify are primarily known exploits for classic Bluetooth. They do not provide a deeper analysis how the threats could be adapted for a BLE context.

One type of threat that is introduced in the paper that was not identified in the STRIDE model above is the threat from malware. Malware will be addressed later in this chapter.

Jasek's paper, "GATTACKing Bluetooth Smart Devices", is up to this date the most thorough analysis of the BLE attack surfaces. In his paper, Jasek identifies seven different types of possible attack [JAS16].

- Attacks on advertisements
- Passive interception
- Active interception
- Attacks on exposed services
- Attacks on pairing
- Whitelisting bypass
- Privacy considerations

For each type of attack he goes on to provide a high level description as to how the attacks can be perpetrated. Rather than relying on the BLE specification primarily for his analysis, Jasek instead provides attack possibilities based on actual experience testing BLE devices. His analysis does not cover the full range of attack possibilities on the over-the-air interface, however the insightful input from his practical experience testing BLE devices has been incorporated into the STRIDE analysis above.

In John P. Dunning's thesis, "Bluetooth Threat Taxonomy," Dunning systematically analyses threats to classic Bluetooth and evaluates tools that facilitate those attacks. In his thesis, the introduces a set of threat categories for the evaluation of classic Bluetooth. His categories include [DUN10]:

- Obfuscation
- Surveillance
- Extended range

- Sniffing
- Man-in-the-Middle
- Unauthorized direct data access
- Denial of Service
- Malware
- Fuzzer

Dunning's threat model was considered for use in this paper. Ultimately, though, the STRIDE model was selected because it is better known and was the recommended approach for Burn's threat modeling methodology. On the one hand, the STRIDE model provided more generic categories by which the analysis could take place. This provided a level of freedom to identify simple threats and more complex threats under the same threat categories. But, as in the threat analysis performed by Minar and Tarique, Dunning also introduces malware. In the STRIDE framework, it is difficult to categorize malware.

Malware is a tricky type of threat to categorize because it is a complex type of threat. It can target one or more threat categories and can involve many layers of an application; from hardware to software. The examples of malware provided by the researchers were specific to classic Bluetooth and exploited common features of the Bluetooth stack. Examples inlcude Caribe, Skuller, CommWarrior and Lasco. At the time of this project, no BLE-specific malware has been publicized. However, in the case that malware were identified and made public, then it could most assuredly be categorized and analyzed under at least one of the STRIDE categories.

5 Performing a BLE Security Analysis

When faced with a Bluetooth low energy security analysis, an analyst must be armed with the correct information, approach and tools to achieve the best security testing coverage in the allotted time frame. This chapter will outline a security methodology that relies on the Penetration Execution Standard (PTES) for testing BLE systems.

PTES provides the backbone of a security analysis in the form of seven testing phases [PTES17]. These include:

- pre-engagement interactions
- intelligence gathering
- threat modeling
- vulnerability analysis
- exploitation
- post exploitation
- reporting

In the PTES standard, several concrete examples of common network pentest targets are provided as examples. In spite of the fact that the examples provided are traditional network testing targets, the overarching testing backbone fits to BLE testing as well.

The remainder of this chapter will address BLE testing specifics for each phase in the PTES. The stages of this testing methodology should be reviewed separately from this document and in their entirety because they provide very practical recommendations on carrying out a security analysis from the contract stage to the end report. In this chapter, only details peculiar to BLE testing will be presented to aid an analyst in preparing for and executing an analysis. At the end of this chapter, a review of existing tools will be provided

to aid the analyst in the practical aspects of intelligence gathering, exploitation and post exploitation.

5.1 Pre-Engagement Interactions

During pre-engagement interactions, the practical aspects of project management are discussed and agreed upon with the client. Among other activities, the scope, limitations, responsibilities and time frames are established for the security analysis.

Determining the scope and limitations in a security analysis is critical because, as identified in this paper's introduction and in chapter 3, BLE is often used in systems that involve a variety of assets and process flows that could have wide, physical distribution. At the point of planning the analysis, it is important to identify:

- The assets that are involved in the entire application.
- The general inputs, outputs and high-level process flows between the assets.

The knowledge of these components is essential because this information can be used to identify which assets can be used to tell if exploits have been successful or not. This information can also be used to identify which of the interfaces in the entire system should be tested and which are out of scope.

For those items in scope, the customer must provide assurance of ownership of those items and the rights to allow security testing of those items. System components that are hosted on infrastructures external to the customer, such as in the Amazon Cloud, should also be identified. Appropriate permission needs to be attained from external service providers before any tests begin.

Further, the following information should be gathered or requested.

1 Bluetooth Version

Which version of Bluetooth is being used in the system? The answer to this question determines the testing equipment needed for testing. Currently, most off-the-shelf Bluetooth dongles support version 4.0.

Mobile phones at the time of this writing support at best 4.1. For applications that rely on 4.1 or 4.2, a tester will need to have equipment that can support these versions. This restricts the tools that can be used for testing.

2 Master-Central-Scanner vs. Slave-Peripheral-Advertiser

Which BLE device(s) is/are being tested? Is it the master device, the slave device or both? This will have an influence on the tools and the methods that can be used for testing.

3 BLE Pairing

If BLE pairing is being used, additional information needs to be gathered. This will determine, in part, the tools needed for testing.

3.1 What sort of pairing is used: LE legacy or LE secure? What is the expected security level in pairing?

3.2 How can the device be reset after pairing has been established so that it can be re-paired with another device for testing?

3.3 If out of band pairing is being used, how can it be simulated for testing purposes? This will have implications of which hardware and software is necessary for testing.

3.4 If the device is reliant upon LE secure, it would be helpful to have test devices set in debug mode and test devices not set in debug mode. The devices set in debug mode, due to the default Diffie-Hellman keys used, will make it easier to perform certain types of testing.

4 Proprietary Security Mechanisms

Are any proprietary security mechanisms being used? This could include things such as authentication between the devices, proprietary encryption at the application level, integrity measures such as additional CRCs or MACs. It could be argued that this information should be identified in the course of the security analysis. Knowing this information ahead of time will assist in the selection of testing

tools, and it will save the tester (and the customer) time and money in the end.

5 BLE Configuration

Which chipset(s) is/are being used at the controller and, if applicable, the host level? How are the Bluetooth components, host, HCI and controller distributed? This information could be useful for research into known issues and practical test planning.

6 Functional Specification, User Documentation, Functional Demonstration

Documentation and/or a demonstration should be requested to give the analyst a jump start into understanding how the whole system interacts. Once again, it could be argued that a pentester should not have access to this type of information because an attacker would not necessarily have access to this type of information. The provision of the documentation is justified for a security analysis because of the limited time frame in which an analyst has to perform the tests. An attacker has unlimited time to acquire documentation, reverse engineer the communication and/or the system software. Without access to this information, the analyst could spend the entirety of the test reverse engineering the communication protocol rather than testing the security of the functionality.

7 Mobile Device Apps

Does the system rely on mobile device apps? Have the apps been released publicly? If not, then access to the apps will need to be organized. Additionally, it would be ideal to have access to at a minimum the Android version of the app because it is easier to gain access at a developer level to the Android device for application and Bluetooth log monitoring.

8 Monitoring Tools

Do the developers have a way to monitor the internal state of the parts of the system in use? This could include log files, debug traces, etc. If

this type of functionality is available in a production-ready device, this is a security finding unto itself. And, in addition to this, it would be one more tool for the security analyst to use to determine if his or her attacks are having effects or not.

9 Operating Context

If the application is dependent on a specific operating context, such as in a manufacturing control context, then a mechanism will need to be provided to simulate the operating context. Further, support for the simulated environment will need to be provided so that the least amount of time is lost in getting the simulation up and running stably for testing.

10 Number of Test Systems

It could be advantageous to have at least two sets of test systems so that certain aspects of testing can be parallelized to save time.

5.2 Intelligence Gathering

During the intelligence gathering phase, the analyst collects and analyses information that will be helpful for an analysis. The PTES identifies three different types of intelligence gathering: passive, semi-passive and active.

- Passive information gathering is where information is gathered about the target of interest without accessing the device/application or any of the vendors related to the target of interest.
- Semi-passive information gathering is where information is collected by interrogating resources of the vendors related to the target of interest. An example of this would be visiting a vendor's website. This activity should not raise any suspicion on the part of the vendors.
- Active intelligence gathering is where information is gathered without regard to covering the tester's tracks.

Once the scope has been clarified and the components of the target of interest are identified, intelligence gathering can begin. The following intelligence gathering activities should be considered during this phase.

5.2.1 Passive Information Gathering

1. Review Provided Documentation
   The documentation provided by the customer should be reviewed to gain a solid understanding of how the target of interest works. Any backdoor support functionalities discovered in the documentation should be noted for further investigation.

2. Review Publicly Available Information
   If the services provided by the devices are based on standardly supported services from the Bluetooth SIG, the service definitions should be acquired and reviewed from the customer or from: https://www.bluetooth.com/specifications/adopted-specifications.

3. Chipset/Infrastructure Research
   For the components used in the target of interest, research should be done to determine if any of the known components' software or hardware have existing security flaws.

4. Security Mechanisms
   If the customer has chosen to implement security measures outside of the BLE specification, then research should be performed to gather any information about these mechanisms. For example, if a login/password is used for authentication, check to see if default logins and passwords have been posted for this vendor in other products.

5. OWASP IoT Project [OWA16]
   Review the current state of the OWASP Internet of Things Project. OWASP is an organization that is best know for putting together comprehensive testing guidelines for web applications and web services. As of 2015, they started developing a set of guidelines for

IoT, and the information here could provide further insight into testing possibilities.

5.2.2 Semi-Passive Information Gathering

1 Monitor BLE Advertising Channels

A sniffer will be needed to monitor the advertising channels to determine what information is broadcast by the advertiser. Special care should be taken to ensure that all three channels are sniffed. While monitoring the advertising channel, the target of interest's behavior should be observed to determine the role of the advertisements and, if present, scan requests in system operation.

2 Monitoring Data Channels

While monitoring the advertising channels may be trivial, monitoring the data channels, as discussed in chapter 4 is not. Communication traffic should be captured for all identified data flows and notations on the impacts of the data flows in the system should be made. Successful monitoring is partially dependent on whether or not the exchanged information can be evaluated in plaintext.

2.1 If there is no encryption protection on the data channel, then a Bluetooth sniffer is sufficient to observe and map the communication exchanges on the data channels.

2.2 If LE legacy pairing is used with just works or pin entry, then the encryption can be broken by capturing and analyzing the pairing exchange. After this, the data traffic can be sniffed.

2.3 If LE secure pairing in developer mode is available, then a sniffer that has the LE secure debug mode keys enabled would be needed to sniff the traffic. (See section 4.3.1.3).

2.4 If other forms of pairing are used, or if proprietary encryption is used, then more active forms of information gathering are necessary.

5.2.3 Active Information Gathering

1. Service Discovery

   Establish a connection to a peripheral device and perform a service discovery request. Capture all of the identified services, characteristic descriptors, values, extended characteristics and permissions for the various characteristics.

2. Extended Handle Investigation

   During service discovery of a BLE device, there is a remote chance that not all services and characteristics will be returned. To ensure that this is not the case, a scan of the full range of BLE handles (0x01-0xFFFF) should be made.

3. Man-in-the-Middle (MITM)

   In the case that LE pairing is used, then a man-in-the-middle attack can be used to observe the data being exchanged on the data channel.

4. MITM-Resistant Encryption

   If proprietary encryption or man-in-the-middle-resistant LE secure pairing is used at the application level, and no workaround can be found to gain access to the raw data exchanged on the data channel, then the patterns of data exchanged should be observed and noted. Lengths of packets and the rate of packet exchange could provide some insight into what data is being exchanged. Additionally, if proprietary encryption is being used, this data could be captured for future replay testing. Further to this, it should be discussed with the customer whether or not the provision of key material would be beneficial for further testing.

5. Android Bluetooth Logs

   If the target of interest exchanges data with an Android master, then the Android device can be placed into developer mode, and the Bluetooth logs can be captured. This will provide information concerning what data is sent from and received by the host.

5.3 Threat Modeling

The PTES version of threat modeling focuses on assets and attackers. Assets include both the assets involved in testing and the processes associated with those assets. Attackers include the identification of attackers and the capabilities of those attackers.

For the steps of identifying assets and processes, the analysis in chapters 3 and 4 could be used as a starting point. The relevant aspects outlined there could be selected for the concrete analysis, and the irrelevant aspects can be left out. In addition to this, the target of interest's specific assets and identified process flows from the intelligence gathering phase should be taken into consideration.

In addition to the identification of assets and processes, an identification of the types of attackers who would have motivation to attack the target of interest should be performed. For example, the set of attackers against a smartwatch is likely to be different than the set of attackers against a building's climate control system. Further, the type of access that potential attackers would have should also be identified. Identifying the types of parties who would be interested in attacking the targets of interest can help later in the prioritization of vulnerability resolution.

5.4 Vulnerability Analysis

Vulnerability analysis is the activity of identifying flaws within the target of interest that could be leveraged by an attacker. This will take into consideration the identified assets, process flows, the attackers and the access to a target of interest. One of the goals in this phase for a BLE device is to methodically map vulnerabilities and attack vectors. This can be used to set up a concrete testing plan to make the most of the time remaining in the testing window.

Chapter 4 outlines a variety of attack vectors for Bluetooth low energy which can be used as a starting point.. At a minimum, the following attack vectors should be considered during a vulnerability analysis.

1 Pairing

1.1 Does the target of interest use BLE pairing?

1.1.1 Is it LE legacy…

1.1.1.1 with just works or PIN? If yes, then the STK can be recovered and used to decrypt traffic and make perform further attacks.

1.1.1.2 with OOB? If yes, then the length of the value being passed out of band should be investigated. Is it a length that can be brute forced for key recovery?

1.1.1.3 with a weak STK? The key length can be negotiated and be as little as 7 bits. The length is negotiated in the pairing request message in the maximum encryption key size field [BLE-SMP, 635].

1.1.2 Is it LE secure?

1.1.2.1 Are the Diffe-Hellman keys used in a production ready device the same keys used in debug mode?

1.1.2.2 Given the pairing implementation, how realistic is it to perform a man-in-the-middle attack?

1.1.3 Is the same STK or LTK used repeatedly or cyclically?

1.1.4 How much entropy does the pseudorandom number generator provide?

1.1.5 Is the STK or LTK easily readable from any component that makes up the target of interest?

2 Is proprietary encryption used? If so, information from the intelligence gathering stage should be used to identify potential vulnerabilities.

3 Data Disclosure

3.1 What information is revealed on the advertising channel in either advertisements or scan responses?

3.2 Data Channel

3.2.1 What information is revealed if the data channel is sniffed in clear text?

3.2.2 What information is revealed if the encryption on the data channel can be broken?

3.2.3 If encryption on the data channel cannot be broken, what information can be gathered through side channel means such as via packet lengths, rates at which packets are sent, etc.?

3.2.4 If encryption on the data channel is broken, is it possible to initiate the encryption pause procedure? If so, is it possible to prompt devices to exchange communications in plaintext which is contrary to the BLE specification? [BLELL-98]

3.3 By establishing a simple connection to a slave, what services are available and which confidential information is accessible via scans of the characteristics?

4 Trackability

4.1 Can the target of interest be tracked with the device ID, values broadcasted during advertising or via service offerings?

4.2 Is long-range tracking of a device a concern for the system?

4.3 Does the target of interest combine BLE functionalities with location services such as GPS such that it is susceptible to exploitation?

4.4 BLE Private, Resolvable Addresses

4.4.1 How often does the private resolvable address change? Is it frequent enough to satisfy security requirements for the target of interest? Can the timer be influenced somehow?

4.4.2 Can the IRK be recovered via weaknesses in pairing?

4.4.3 Are there weaknesses in the generation of the IRK? Is the same IRK used each time? Is the same IRK used periodically? How much entropy does the pseudorandom number generator provide?

5 Tampering and Privilege Escalation

5.1 On the Advertising Channel

5.1.1 How difficult is it to spoof an advertiser or a scanner? Can this spoofing be used to send values that control the behavior of the target of interest?

5.1.2 Are whitelists used? What is the effort needed to overcome the filtering?

5.2 On the Data Channel

5.2.1 Can the characteristic values of offered services be manipulated to control the behavior of the target of interest with a simple connection?

5.2.2 Is a CSRK used on the data channel?

5.2.2.1 Can it be recovered via weaknesses in pairing?

5.2.2.2 Are there weaknesses in the generation of the IRK? Is the same IRK used each time? Is the same IRK used periodically? How much entropy does the pseudorandom number generator provide?

5.2.2.3 To what extent can attacker prompt either a master or slave device to generate signed messages which can be captured for later replay?

5.2.3 Can the attacker spoof either a master or a slave and replay sniffed messages to manipulate the behavior of the target of interest?

5.2.4 Can the attacker spoof either a master or a slave and create and send system messages to manipulate the target of interest's behavior?

5.2.5 Can an attacker establish a man-in-the-middle attack with the goal of manipulating exchanged values?

5.2.6 Is a relay attack a threat for the security of the device?

5.2.7 Can the attacker send injection messages (command, SQL, etc.) to either gain further access to the application or upstream processes?

5.2.8 An analysis of the application logic and process flows should be performed. Are there any flaws in the logic that can be exploited for manipulation or privilege escalation purposes?

6 Denial of Service

6.1 Can a target of interest's connection be broken via electrical interference or flooding the data channels where communication takes place? Given the target of interest's context and usage, is this a security threat?

6.2 Can an attacker connect to the target of interest using inefficient connection parameters or instigate resource intensive actions that would cause the battery to drain more rapidly? Given the target of interest's context and usage, is this a security threat?

7 Repudiation

7.1 Are there requirements for repudiation? If so, to what extent are these requirements addressed by the application layer? Are there weaknesses that can be exploited?

5.5 Exploitation

Exploitation is the stage of a security analysis where the analyst attempts to bypass security restrictions to gain unauthorized access to the system or

system resources. In the exploitation phase, the vulnerabilities that were identified in the vulnerability analysis should be tested.

In addition to this targeted testing, fuzzing should be performed. Fuzzing should include: fuzzing at the application layer and fuzzing at the GATT/ATT layers. If possible, fuzzing at the lower level BLE layers should also be performed as well.

Before beginning fuzzing, it is important to identify how the system will be monitored. Some options include, but are not limited to:

- monitoring behavior of upstream systems
- capturing logs
- watching system responsiveness (lack of response indicates either a system interruption or restart)
- monitoring communication traffic.

5.6 Post-Exploitation

In the post-exploitation phase, the analyst identifies the next level of exploitation possible. The access that was achieved in the exploitation phase should be assessed to determine if further penetration into upstream processes or systems that contain the BLE components can be gained. Before proceeding with this next step in testing, the risks should be discussed with the customer; especially if the upstream processes support live systems.

Often, in post-exploitation artifacts are left on servers or in systems to allow further ingress into other parts of the system. These artifacts should be carefully tracked, and they should be removed at the end of the testing cycle.

5.7 Reporting

In the reporting phase, the findings from the analysis are presented in written format. The PTES breaks the report structure into two main sections: the executive summary and the technical report. The executive summary is

geared toward higher level IT and security managers. The technical report targets the individuals who will be implementing the changes to address the vulnerabilities identified during testing.

The executive summary should be brief and provide an overview of the most significant results from testing. Some method of prioritization and recommendations for next steps should also be provided. Some sort of rating that communicates the target of interest's production-readiness could also be provided.

The technical part of the report should provide detail concerning what vulnerabilities were found, what the risk is from those vulnerabilities and what should be done, technically or organizationally, to mitigate these vulnerabilities. Likelihood of exploit and a mitigation time frame should also be provided. These recommendations should be provided based on the work in the intelligence gathering threat modeling stages and exploitation stages.

5.8 BLE Security Testing Tools

This section will provide a list of existing research tools. Each tool will be evaluated for purpose of use, maturity and shortcomings. These tools, except where explicitly noted, were tested during the time of this masters project. Each tool was allotted at maximum a day for testing purposes.

5.8.1 BlueZ

BlueZ is an open source Bluetooth stack implementation for Linux. It comes pre-installed with popular versions of Linux such as Ubuntu and Kali. The packages can also be downloaded, compiled and installed from: http://www.bluez.org/. BlueZ works with a computer's built-in Bluetooth or with common, external Bluetooth dongles.

There are a variety of other tools that have been built on top of the BlueZ stack. This software is mature, and it is actively developed. Documentation for the BlueZ stack is limited. There is a great deal of non-obvious

functionality available in this stack. To be able to access these functionalities, an analyst would have to spend time going through the BlueZ source code.

5.8.2 hciconfig/gatttool

These are two programs that come with the BlueZ stack on Linux and can be used with a standard Bluetooth dongle.

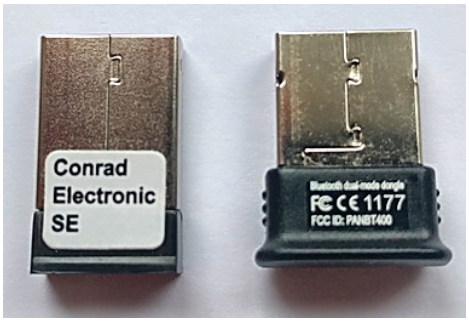

Figure 48: Examples of common Bluetooth dongles

These programs allow a user to stop and start the physical Bluetooth interface, scan for devices and take advantage ATT and GATT operations and features. For these tools there is in-program help, although not all of the features are well-explained. There are also a handful of tutorials online that demonstrate how to use the software.

The kind of ATT/GATT data that are returned by these tools tend to be raw, hex data that need to be interpreted. These tools are useful for rudimentary scanning and manual attribute testing if there is nothing else available.

5.8.3 Pygatt

Pygatt is an open source python library that is built on top of the BlueZ Linux Bluetooth stack. It provides the same types of functionality as hciconfig and gatttool, except that it allows an analyst to automate testing with scripts.

Pygatt and instructions on how to install it can be found under: https://github.com/peplin/pygatt. The library is actively updated, has example code and documentation. It is primarily useful for the simulation of a scanner/master/central device.

During the course of this project, this library was used to develop some reconnaissance scripts which can be found under: https://github.com/jennj/BLE-Scripts/tree/master/scripts-read.

5.8.4 NCC Group BLE Python Scripts

The NCC Group has published some open source scripts that are dependent on a modified version of the Pygatt library and the use of Android Bluetooth logs. These scripts have been specifically developed for BLE pentesting. Up to this point, there have been three BLE tools published from the NCC Group:

- BLESuite - https://github.com/nccgroup/BLESuite
- BLESuite-CLI - https://github.com/nccgroup/BLESuite-CLI
- BLE-Replay - https://github.com/nccgroup/BLE-Replay

Basic instructions can be found for these tools under: https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2016/september/introducing-blesuite-and-ble-replay-python-tools-for-rapid-assessment-of-bluetooth-low-energy-peripherals/. In addition to this, the tools have documentation and examples. At the time of this project, the scripts are up-to-date. But, the update activity for these scripts is low, and it is unclear if these projects will be further maintained.

5.8.5 noble/bleno

noble and bleno are two cross platform, open source node.js modules that run on Mac OSX, Linux and Windows.

- noble – simulates a BLE central: https://github.com/sandeepmistry/noble
- bleno – simulates a BLE peripheral: https://github.com/sandeepmistry/bleno

During this project, bleno was tested on a Linux (Ubuntu 16.04) system. To test these modules, a bleno peripheral was created based on the pizza peripheral example which can be found under:

https://github.com/sandeepmistry/bleno/tree/master/examples/pizza. Using this example it was possible to advertise packets, allow a connection from a central device and update GATT characteristic values. This would be ideal for spoofing a peripheral device for testing a central's application.

There is a significant amount of development activity surrounding the noble and bleno modules. There is little documentation, but there is a sufficient amount of example code and third-party tutorials that allows an analyst to quickly get up to speed. bleno and noble are ideal modules for testing the application layers and upstream processes of the centrals and peripherals of a BLE system.

5.8.6 gattacker[4]

gattacker is a security analysis tool that is dependent on noble, noble, json, web sockets and text files. The tool is advertised to run on a Raspberry Pi. It provides the opportunity for an analyst to set up a man-in-the-middle attack between two communicating BLE devices that do not use pairing. gattacker can be found under: https://github.com/securing/gattacker. A fundamental description of how it works can be found under: https://github.com/securing/gattacker/wiki/FAQ.

This tool was released shortly before this project started and is relatively young. There are instructions for installation, but beyond this there is very little documentation available, and no tutorials were found on-line. This tool could be extremely useful eavesdropping and fuzzing values that are exchanged between two BLE devices that do not use encryption. Because it is reliant on noble and bleno, it is only capable of testing values at the application and upstream process levels.

4 During the project a day was spent with the attempt to install gattack on two different Raspberry Pi devices: a Raspberry Pi B+ and a Raspberry Pi 3 using clean installations of the Raspbian operating system. In both cases, installation was not successful. The dependencies identified in the instructions could be installed. This was time-consuming because the default packages for Raspbian were too old, so the latest versions needed to be compiled. Ultimately, the final build failed and time ran out for experimentation with this tool.

5.8.7 BtleJuice[5]

Btlejuice is another security analysis tool that is dependent on noble and bleno and web sockets. The tool is also advertised to run on a Raspberry Pi. It provides an analyst the opportunity to perform a man-in-the-middle attack between two BLE devices and uses a web interface to help visualize communication. Btlejuice can be found under: https://github.com/DigitalSecurity/btlejuice.

This tool was released after the start of this project, and it is relatively young. In its README.md file, there are installation instructions and a minimal set of instructions on how to get the tool up and running. This tool could also be extremely useful in eavesdropping and fuzzing values communicated between two BLE devices. It is not clear if the tool supports man-in-the-middle attacks for devices that use pairing and encryption.

5.8.8 PyBT / Scapy

PyBT is a python Bluetooth library that is not dependent on pygatt. It was developed with the goal of being able to perform a security analysis at the application, BLE GATT and BLE ATT levels [RYA14]. Scapy is a pentesting tool that allows an analyst to quickly build communication packets using a variety of protocols for testing purposes. Scapy is flexible enough to allow packets that do not conform to the protocols which allows for fuzzing at protocol layers. The PyBT library, and a link to the Scapy version that supports PyBT can be found under: https://github.com/mikeryan/PyBT.

There is no public documentation for PyBT, and no tutorials could be found. During this project it was possible to write a script that created a peripheral and advertise packets (see: https://github.com/jennj/BLE-Scripts/blob/master/scripts-peripheral/pybtTestServer.py). An attempt was

5 During the project, as with gattack, a day was spent with the attempt to install Btlejuice one Raspberry Pi devices: a Raspberry Pi 3 using a clean installation of the Raspbian operating system. The installation was not successful. The dependencies identified in the instructions could be installed with some time investment, but the final build failed and time ran out for experimentation with this tool.

made to set up a GATT database with the goal of cloning a peripheral with writable values, but there was no success in establishing a connection to the cloned peripheral. More time is required than was alloted in this project to determine the source of the connection problem and to be able to leverage the full potential of this tool.

The PyBT library is immature in that it there is no documentation provided, and there is an assumed high-level of understanding about BLE. The project does not appear to be actively maintained as it was last updated in 2015.

5.8.9 Nordic NRF51 dongle

The Nordic NRF51 dongle is a piece of hardware that is dedicated to BLE development and sniffing, and it can be used on Windows, Linux and OSX. It relies on a special set of drivers that are separate from the BLE drivers for the different operating systems.

The software that runs on the device is broken down into two parts: a soft device and the application. The soft device provides the BLE protocol implementation. The application specifies the peripheral being built complete with the definition of ATT/GATT services and characteristics.

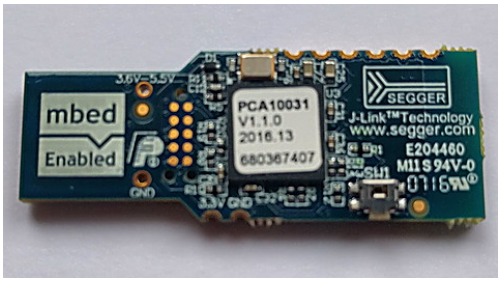

Figure 49: Nordic NRF51 Dongle

The original development software for the dongle was targeted for Windows systems, and there is a significant amount of documentation and information for development on Windows available online. Since the dongle release, developers have posted information about how to develop and flash the dongle with new software from Linux and OSX environments. Two tutorials that

were leveraged during this project can be found under the following web addresses.

- https://leavesified.wordpress.com/2016/03/24/setup-nrf51-development-on-linux/
- https://www.allaboutcircuits.com/projects/ble-using-nrf51-arm-gcc-build-environment/

Using a combination of the two tutorials listed above, it was possible to get a peripheral device configured and advertising. The peripheral device was configured to be a partial clone of a Gigaset Gtag (only a subset of the Gtag services were configured for testing). It was possible to connect to the "cloned" peripheral and modify a value that was configured to be writable.

Nordic, too, has begun offering development and analysis tools that work on other operating systems than Windows. Some important examples include:

- Nordic's Github account with multiple pages of BLE tools: https://github.com/NordicSemiconductor
- Tool to scan and clone peripherals: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Connect-for-desktop
- Wireshark sniffer plugin: https://devzone.nordicsemi.com/blogs/750/ble-sniffer-in-linux-using-wireshark/

The Nordic platform is extremely mature. It has actively-developed supporting software, active forums and wide variety of online tutorials. In addition to this, the version of BLE is not limited to the availability of the standard Bluetooth dongles on the market. To upgrade to BLE 4.2, for example, an analyst simply has to install the correct soft device and then adapt the application software to take advantage of this.

One question that is not entirely clear is whether or not the soft devices could be adapted to allow fuzzing at the BLE protocol level. If this were the case, it would make the NRF51 a powerful platform for BLE security testing.

5.8.10 Texas Instruments CC2540 Dongle

The Texas Instruments CC2540 dongle is a reliable BLE sniffing device that is accompanied by a sniffing software that color codes and labels BLE packet parts to reduce the complexity of interpretation and analysis of BLE traffic. The software is simple to set up, and it runs on Windows. Examples of screenshots from the sniffer can be seen in figure 2 in chapter 2.

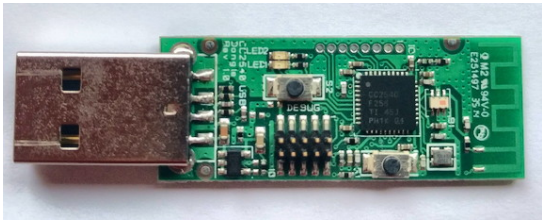

Figure 50: Texas Instruments 2540 Sniffer

Texas Instruments also offers a CC2540 & 2541 development kits (see: http://www.ti.com/tool/cc2541dk-mini and http://www.ti.com/tool/cc2540dk). These kits were not evaluated during the course of the project. From looking at the documentation, though, they provide a similar functionality to the Nordic dongle in so far as it is possible to select which version of the BLE protocol stack to flash on these devices for testing purposes. The development environment for BLE applications target the Windows operating system.

As with the Nordic NRF51 donge, if the BLE stack could be adapted to allow for fuzzing at the BLE protocol level, that would make this tool also a powerful platform for BLE security testing.

5.8.11 Ubertooth and Crackle

Ubertooth is an open source BLE sniffer. Its output can be piped into Wireshark.

- Ubertooth software and firmware: https://github.com/greatscottgadgets/ubertooth
- Wireshark instructions: https://github.com/greatscottgadgets/ubertooth

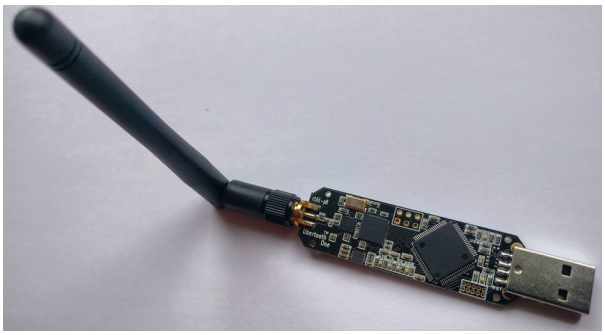

Figure 51: Ubertooth sniffer

In comparison to the TI sniffer discussed in section 5.8.10, the Ubertooth sniffer is much less reliable. That being said, the Ubertooth has the advantage that it can be combined with the Crackle. Crackle is a security analysis tool that can derive an LTK from an LE legacy pairing exchange. Crackle can be found under: https://github.com/mikeryan/crackle. These two tools are relatively simple, and there is enough information on-line to get them up and running quickly for testing purposes.

5.8.12 Nordic NRF Connect and NRF Toolbox

Nordic NRF Connect is a mobile app that is available for the iOS and Android platforms. NRF Connect scans the area for BLE peripherals. If connectible peripherals are found, a connection can be established with the app. The details of the services and characteristics will be displayed. Using this NRF Connect, values can be read and written (where allowed) from and to the peripheral. This app is excellent for reconnaissance of a peripheral device and for manual testing purposes. NRF Connect can be found under: https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Connect-for-mobile-previously-called-nRF-Master-Control-Panel.

The NRF Toolbox is an app that allows a user to turn a mobile phone into a BLE peripheral and is available for Android and iOS. In the app itself there are already a set of pre-programmed peripherals available. There is one peripheral, the UART peripheral, that allows the configuration of a custom peripheral. This app could be used to clone a peripheral and manually test an application. NRF Toolbox can be found under: https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Toolbox-App.

The usage of these mobile apps is self-explanatory from the user interface. From a maturity perspective, the apps have been actively maintained over the passed two years, and the source code for both of these mobile apps can be downloaded and adapted.

5.8.13 LightBlue Explorer [iOS]

LightBlue Explorer is an iOS app that provides the same type of functionality as NRF Connect. It can be used for reconnaissance purposes and manual testing of BLE peripherals. LightBlue Explorer can be found under: https://itunes.apple.com/us/app/lightblue-explorer-bluetooth-low-energy/id557428110?mt=8.

5.8.14 RamBLE [Android]

Ramble is an mobile app for Android that scans for advertising BLE peripherals. The goal of this app is to keep track of all of the unique peripherals identified in the wild and to maintain information about where the devices were seen and if they had been last seen. The mobile app allows visual mapping of the devices and export of the data for further analysis. Ramble can be found under: https://play.google.com/store/apps/details?id=com.contextis.android.BLEScanner&hl=en.

Usage of the app is self-explanatory from the user interface, and it would be ideal for a security analysis where the target of evaluation is a commonly available device. This would allow the analyst to collect data on how the device is actually used and the type of data that is exposed out in the field.

5.8.15 Android Bluetooth Developer Mode and the Bluetooth HCI Snoop Log

In addition to sniffing network traffic, it is possible to reverse engineer BLE communication for BLE systems that have an Android mobile app. This can be achieved by turning on developer mode on an Android device and activating the Bluetooth HCI snoop log. This will capture all Bluetooth data that goes over the HCI interface. An example of this type of reverse engineering can be found in this tutorial: https://medium.com/@urish/reverse-engineering-a-bluetooth-lightbulb-56580fcb7546#.v3rh4km0h.

The log that is captured can be exported from the phone to a computer for analysis in Wireshark. This type of analysis is advantageous in cases where sniffing is not a possibility due to interference or encryption on at the link layer level.

5.8.16 Testing Tool Summary

In the blog entry “Introducing BLESuite and BLE-Replay: Python Tools for Rapid Assessment of Bluetooth Low Energy Peripherals” the NCC Group describes the difficulties of testing Bluetooth low energy systems. There is a dearth of mature testing tools available for BLE security analysts that provide the level of control that is needed to perform a security review [FOR17]. After performing a review of available BLE tools at the time of this project, the author concurs with the NCC Group’s assessment. As can be seen from the review of the tools above, there are obvious gaps to the BLE security tool chain. With the increased push for BLE in context of IoT and industry 4.0 applications, the need for a reliable tool set is growing.

6 The BLE Security Testing Challenge

At the start of this project, the overarching goal was to construct a threat model for BLE systems so that a comprehensive security testing methodology could be developed and tool chain for over-the-air Bluetooth low energy systems could be identified. Only part of this goal was achieved.

After both review of the BLE 4.2 specification (see chapter 2) and hands-on experience with available BLE testing tools and BLE devices, it was possible to define a threat model (see chapters 3 and 4) and provide a testing roadmap for analysts (see chapter 5). This testing roadmap should serve as a starting point for BLE testing and should be extended as new versions BLE are developed and new threats are discovered.

During the course of this project it became apparent that the tools available for BLE security testing do not cover the range of testing needs defined in the testing methodology. Thus, the logical next step in this case would be to define the requirements for a flexible, comprehensive framework for testing. This should take into consideration factors such as differing versions of Bluetooth, pairing and encryption. A tool or a set of tools to cover missing testing functionality would then need to be developed.

We are at a tipping point in terms of BLE distribution. As discussed in Chapter 1, BLE is being pushed more and more often for IoT and industry 4.0 automation applications. If BLE becomes a widespread technology for wireless communication in these areas, then as security professionals we will need to be better prepared to support vendors' security testing needs. It is the author's hope that this analysis provides a reasonable starting point for further BLE security testing tool development.

7 Appendix

7.1 Appendix 1: GAP & GATT Attribute Definitions

Layer	Description	Type	Value	Read Permissions	Write Permissions
GAP	Device name	0x 2A00	Textual representation of device.	Yes	Optional, optional authentication/ authorization
GAP	appearance	0x 2A01	UUID that indicates which icon should represent this device.	Yes	Optional, optional authentication/ authorization
GAP	Peripheral preferred connection parameters characteristic (PPCP)	0x 2A04	See table 7.1.1below.	Yes, optional authentication/ authorization	No
GAP	Central address resolution	0x 2A06	0 – address resolution not supported 1 – address resolution supported 2-255 – Reserved.	Yes	No
GATT	Primary service	0x 2800	16 or 128 bit service UUID	Yes	No

Layer	**Description**	**Type**	**Value**	**Read Permissions**	**Write Permissions**
	declaration				
GATT	Secondary service declaration	0x 2801	16 or 128 bit service UUID	Yes	No
GATT	Include definition	0x 2802	-included service attribute handle -end group handle -16 bit service UUID	Yes	No
GATT	Characteristic declaration	0x 2803	-Characteristic properties (see Table 7.1.2 below) -Characteristic value attribute handle -Characteristic UUID	Yes	No
GATT	Characteristic extended properties descriptor	0x 2900	Character extended properties bit field (see table 7.1.3 below)	Yes	No
GATT	Characteristic user description descriptor	0x 2901	Textual description of characteristic value	Dependent on profile	Dependent on profile
GATT	Client characteristic configuration descriptor	0x 2902	Client configuration bits (see table 7.1.4 below)	Yes	Yes, optional authenticatio n/ authorization

Layer	Description	Type	Value	Read Permissions	Write Permissions
GATT	Server characteristic configuration descriptor	0x 2903	Server characteristic configuration bits (see table 7.1.6 below)	Yes	Yes, optional authenticatio n/ authorization
GATT	Characteristic presentation format description descriptor	0x 2904	-Format -Exponent -Unit -Name space -Description	Yes	No
GATT	Characteristic aggregate format descriptor	0x 2905	In the case there is more than one characteristic presentation format description, this provides a list of their attribute handles	Yes	No

[BLE-GAP][BLE-GATT]

7.1.1 Peripheral Preferred Connection Parameters

Name	Size	Description
Minimum connection interval	2 octets	Range: 0x0006 to 0x0C80 0xFFFF indicates no specific minimum interval = value * 1.25 ms
Maximum connection interval	2 octets	Range: 0x0006 to 0x0C80 0xFFFF indicates no specific maximum

		interval = value * 1.25
Slave latency	2 octets	Range: 0x0000 - 0x01F3
Connection supervision timeout multiplier	2 octets	Range: 0x000A to 0x0C80 0xFFFF indicates no specific value requested time = value * 10 msec

[BLE-GAP, 392]

7.1.2 Characteristic Properties

Name	**Bit Mask**
Broadcast	0x01
Read	0x02
Write Without Response	0x04
Write	0x08
Notify	0x10
Indicate	0x20
Authenticated Signed Writes	0x40
Extended Properties	0x89

[BLE-GATT, 533]

7.1.3 Extended Properties Bit Field

Name	**Value**
Reliable Write	0x0001
Writable Auxiliaries	0x0002
Reserved for Future Use	0xFFFC

[BLE-GATT, 535]

7.1.4 Client Characteristic Configuration Bit Fields

Name	**Value**
Default	0x0000
Notification	0x0001

Indication	0x0002
Reserved for Future Use	0xFFF4

[BLE-GATT, 536]

7.1.5 Character Configuration Bits

Name	**Value**
Broadcast	0x0001
Reserved for Future Use	0FFF2

[BLE-GATT, 537]

7.1.6 Characteristic Presentation Format Attribute Value Fields

Name	Size	Description
Format	1 octet	Format of the value.
Exponent	1 octet	Exponent field to determine how the value of this characteristic is further formatted.
Unit	2 octets	The unit.
Name Space	1 octet	The name space.
Description	2 octets	Description as defined in a higher layer profile.

[BLE-GATT, 539]

The full description of the contents of the fields can be found on pages 539-542 of the BLE GATT specification.

7.2 Appendix 2: Advertsing Channel Air Interface Packet Details

7.2.1 Specification References

Further details can be found in the BLE link layer specification on pages 39-45.

7.2.2 Access Address Value for the Advertising Channel

0x8E89BED6

7.2.3 Advertising Channels

RF Channel	0	12	39
BLE Channel Index	37	38	39

7.2.4 Advertising PDU Structure

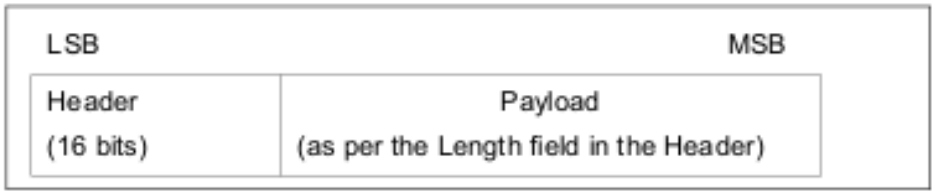

Figure 52: [BLE-LL, 39]

7.2.5 Advertising PDU Header Structure

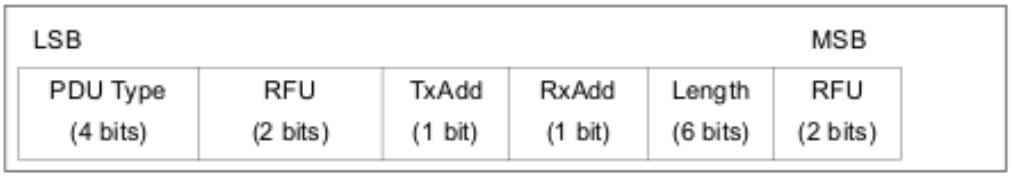

Figure 53: [BLE-LL, 40]

PDU Component	**Description**
PDU Type	This a bit description that indicates what sort of advertising PDU is being transmitted.
RFU	Reserved for future use.
TxAdd	This value of TxAdd is relative to the PDU type. It usually refers to an address field payload and differentiates between a public or a random address.

	• 0=public • 1=random
RxAdd	This value of TxAdd is relative to the PDU type. It usually refers to an address field payload and differentiates between a public or a random address. • 0=public • 1=random
Length	This is the number of octets contained in the payload.
RFU	Reserved for future.

[BLE-LL, 40]

7.2.6 Table Summary of Advertising PDU Type Descriptions

PDU Type	**PDU Name**	**Description**	**TxAdd refers to:**	**RxAdd refers to:**
0000	ADV_IND	Avertising packet for any scanner. Payload includes advertising data from advertiser's host. Connection can be initiated. Fields: • AdvA (6 octets): Advertiser's device address • AdvData (0-31 octets): Advertising data from host	AdvA	--
0001	ADV_DIRECT_IND	Advertising packet targets specific peer device. Connection can be initiated. Fields: • AdvA (6 octets): Advertiser's device address • InitA: (6 octets): Target peer's device	AdvA	InitA
0010	ADV_NONC	Payload includes adverting data from the advertiser's host. Advertiser does not allow	AdvA	--

PDU Type	PDU Name	Description	TxAdd refers to:	RxAdd refers to:
	ONN_IND	connections. Fields: • AdvA (6 octets): Advertiser's device address • AdvData (0-31 octets): Advertising data from host		
0011	CONNECT_REQ	This is an initiator's request to establish a connection with the advertising device. This request contains specific connection parameters. Fields: • InitA (6 octets): Initiator's device address • AdvA (6 octets): Advertiser's devices address • LLData (22 octets): Contains connection parameters (see table 7.2.6.1 for details)	InitA	AdvA
0110	ADV_SCAN_IND	Formerly known as ADV_DISCOVER_IND. Payload includes advertising data from the advertiser's host. Advertiser allows scans. Fields: • AdvA (6 octets): Advertiser's device address • AdvData (0-31 octets): Advertising data from host.	AdvA	--
0011	SCAN_REQ	This is a scanner's request to get more information after receiving an advertisement. Fields: • ScanA (6 octets): Scanner's device	ScanA	AdvA

PDU Type	PDU Name	Description	TxAdd refers to:	RxAdd refers to:
		address • AdvA (6 octets): Advertiser's device address		
0100	SCAN_RSP	Payload includes response data from the advertising host to a SCAN_REQ. Fields: • AdvA (6 octets): Advertiser's device address • ScanRspData (0-31 octets): Advertising data from host.	AdvA	

[BLE-LL, 41-45]

7.2.6.1 LLDATA of the CONNECT_REQ Advertising PDU

The LLDATA fields of the CONNECT_REQ PDU is structured as represented in Figure .

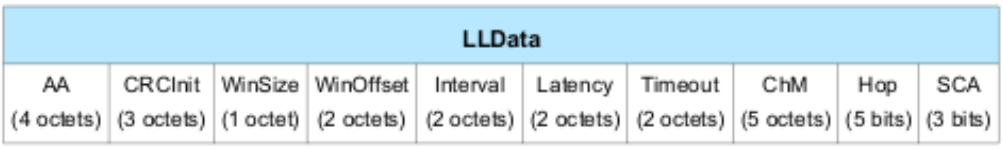

Figure 54: LLData Fields [BLE-LL, 44]

The values of the fields are described in the following table.

Field	Description
AA	Link Layer connections access address
CRCinit	Initialization value for the air interface packet's CRCs on the data channel.
WinSiz	Sets transmitWindowSize value.

Field	Description
e	transmitWindowSize=WinSize*1.25 ms.
WinOffset	Sets the transmitWindowOffset value. transmitWindowOffset=WinOffset*1.25ms.
Interval	Sets the connInterval value. connInterval=Interval*1.25ms.
Latency	Sets the connLatency value. connSlaveLatency=Latency
Timeout	Sets the connSupervisionTimeout value. ConnSupervisionTimeout = Timeout * 10 ms.
ChM	Shall contain the channel map indicating used and unused data channels. See the Bluetooth Specification, Volume 6, section 1.4.1 for further detail.
Hop	Set to indicate the hopIncrement used in the data channel selection algorithm. Value between 5 and 16.
SCA	Sets the masterSCA which determines the worst sleep clock accuracy of the master.

[BLE-LL, 44-45]

7.2.7 Advertising Data Payload Structure

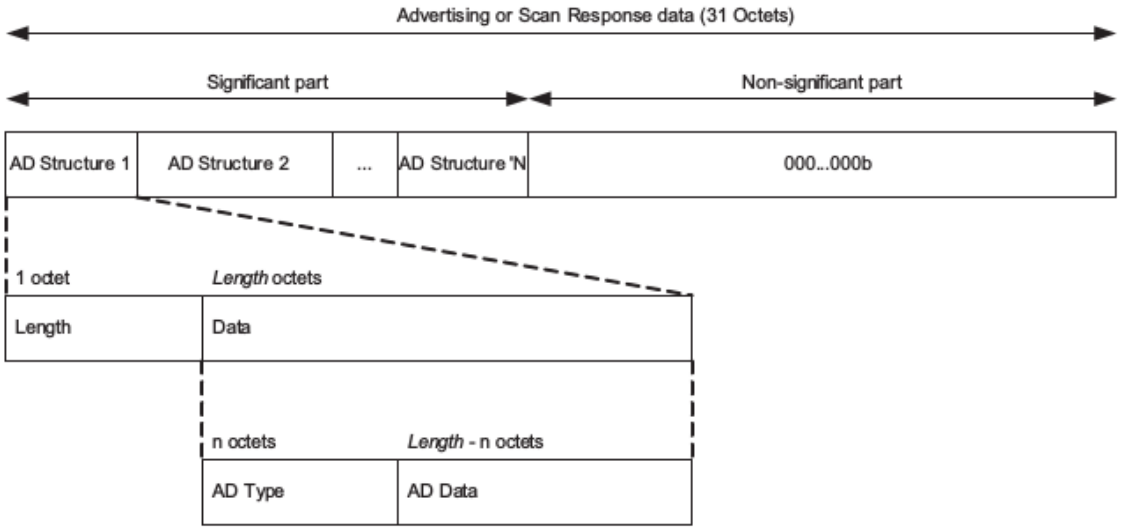

Figure 55: AD Structure [BLE-GAP, 389]

7.2.8 AD Types and AD Data Descriptions in AD Structure

An up-to-date listing can be found under:

- https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

Type	AD Type	Length (octets)	Description
Incomplete list of 16-bit Service UUIDs	0x02	Variable	List of 16-bit service or service class UUIDS
Complete List of 16-bit Service UUIDs	0x03	Variable	List of 16-bit service or service class UUIDS
Incomplete List of 32-bit Service UUIDs	0x04	Variable	List of 32-bit service or service class UUIDS
Complete List of 32-bit Service UUIDs	0x05	Variable	List of 32-bit service or service class UUIDS
Incomplete List of 128-bit Service UUIDs	0x06	Variable	List of 128-bit service or service class UUIDS
Complete List of 128-bit Service UUIDs	0x07	Variable	List of 128-bit service or service class UUIDS
Shortened Local Name	0x08	Variable	Local name assigned to device or a shortened version.
Complete Local Name	0x09	Variable	Local name assigned to the device.
Flags	0x01	0 or more octets	See table 7.2.8.1 below for the flag values.

Type	**AD Type**	**Length (octets)**	**Description**
Manufacturer Specific Data	0xFF	2 or more octets	First two octets contain the company identifier code and the remaining octets contain manufacturer-specific data.
TX Power Level	0x0A	1 octet	Hexadecimal value that represents a value from -127 to +127 dBm
Slave Connection Interval Range	0x12	4 octets	First two octets contain the peripheral's preferred minimum interval value (0x0006 -0x0C80). Second two octets contain the peripheral's maximum interval value 0x0006 to 0x0C80. 0XFFFF means there is no maximum interval value.
List of 16 bit Service Solicitation IDs	0x14	2 or more octets	This specifies the services that are desired by the peripheral from the central.
List of 32 bit Service Solicitation UUIDs	0x1F	4 or more octets	This specifies the services that are desired by the peripheral from the central.
List of 128 bit Service Solicitation UUIDs	0x15	16 or more octets	This specifies the services that are desired by the peripheral from the central.
Service Data – 16 bit UUID	0x14	2 or more octets	First two octets contain the service data UUID. The remaining octets contain the data.
Service Data – 32 bit UUID	0x1F	4 or more octets	First four octets contain the service data UUID. The remaining octets contain the data.
Service Data – 128 bit UUID	0x15	16 or more	First 16 octets contain the service data UUID. The remaining octets contain the

Type	AD Type	Length (octets)	Description
		octets	data.
Appearance	0x19		Appearance.
Public Target Address	0x17	6 octets for each address	Lists the of one or more public addresses of the intended recipient(s) of the advertisement. Addresses come from devices that were previously bound with a public address to the advertiser.
Random Target Address	0x18	6 octets for each address	Lists the of one or more public addresses of the intended recipient(s) of the advertisement. Addresses come from devices that were previously bound with a private address to the advertiser.
Advertising Interval	0x1A	2 octets	This is the adinterval value which is multiplied by 0.625 msec.
LE Bluetooth Device Address	0x1B	7 octets	The last bit of the first octet indicates public/private address: 0=public, 1=private. The remaining 6 octets contain the advertiser’s device address.
LE Role	0x1C	1 octet	See table 7.2.8.2 below for the value descriptions.

[BLE-Supp][BLE-GAP-AssignedNumbers]

7.2.8.1 Flag Values

The flags function such that if a bit in the octet is set to 1, the value for that flag is set.

Bit	Description
0	The device is in limited discoverable mode.
1	The device is in general discoverable mode.

2	BR/EDR is not supported.
3	LE & BR/EDR is supported in controller.
4	LE & BR/EDR is supported in host.
5..7	Reserved

[BLE-Supp, 11]

7.2.8.2 LE Role Values

Value	**Description**
0x00	Only peripheral role available.
0x01	Only central role available.
0x02	Peripheral and central role available. Peripheral preferred on connection establishment.
0x03	Peripheral and central roles available. Central role preferred on connection establishment.
0x04-0xFF	Reserved for future use.

[BLE-Supp, 25]

7.3 Appendix 3: Data Channel Air Interface Packet Details

7.3.1 LL Control PDU Details

7.3.1.1 LL Control PDU Structure

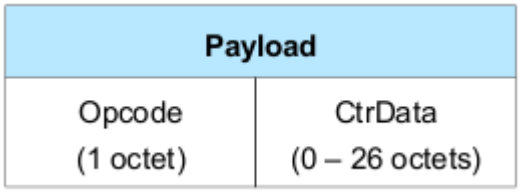

Figure 56: Logical representation of LL control PDU [BLE-LL, 48]

7.3.1.2 LL Control PDU Operations

OP Code	**LL Control PDU Name**	**CtrData**	**Sender**
00	LL_CONNECTION_UPDATE_REQ	WinSize: 1 octet, WinOffset: 2 octets, Interval: 2 octets, latency: 2 octets, timeout: 2 octets, instant: 2 octets	master
0F	LL_CONNECTION_PARAM_REQ	Interval_min: 2 octets, Interval_max: 2 octets, latency: 2 octets, timeout: 2 octets, preferredPeriodicity: 1 octet, referencConnEventCount: 2 octets, Offset0: 2 octets, Offset1: 2 octets, Offset2: 2 octets, Offset3: 2 octets, Offset4: 2 octets and Offset5: 2 octets	master
10	LL_CONNECTION_PARAM_RSP	Interval_min: 2 octets, Interval_max: 2 octets, latency: 2 octets, timeout: 2 octets, preferredPeriodicity: 1 octet, referencConnEventCount: 2 octets, Offset0: 2 octets, Offset1: 2 octets, Offset2: 2 octets, Offset3: 2 octets, Offset4: 2 octets and Offset5: 2 octets	slave
01	LL_CHANNEL_MAP_REQ	ChM: 5 octets, Instant: 2 octets	master

OP Code	**LL Control PDU Name**	**CtrData**	**Sender**
02	LL_TERMINATE_IND	CtrData: 1 octet	master or slave
03	LL_ENC_REQ	Rand: 8 octets, EDIV: 2 octets, SKDm: 8 octets, IVm 4 octets	master
04	LL_ENC_RSP	SKD: 8 octets, IV 4 octets	slave
05	LL_START_ENC_REQ	--	master
06	LL_START_ENC_RSP	--	slave
07	LL_UNKNOWN_RSP	UknownType: 1 octet	master or slave
08	LL_FEATURE_REQ	FeatureSet: 8 octets, see table 7.3.1.3	master
0E	LL_SLAVE_FEATURE_REQ	FeatureSet: 8 octets, see table 7.3.1.3	slave
09	LL_FEATURE_RSP	FeatureSet: 8 octets, see table 7.3.1.3	master or slave
0A	LL_PAUSE_ENC_REQ	--	master
0B	LL_PAUSE_ENC_RSP	--	slave and master
0C	LL_VERSION_IND	VersNr: 1 octet, CompID: 2 octets, SubVersNr: 2 octets	master or slave
0D	LL_REJECT_IND	ErrorCode: 1 octet	master or slave
11	LL_REJECT_IND_EXT	RejectOpCode: 1 octet, ErrorCode: 1 octet	master or slave
12	LL_PING_REQ	--	master or slave
13	LL_PING_RSP	--	master or slave
14	LL_LENGTH_REQ	MaxRxOctets: 2 octets, MaxRxTime: 2 octets,	master or slave

OP Code	LL Control PDU Name	CtrData	Sender
		MaxTxOctets: 2 octets, MaxTxTime: 2 octets	
15	LL_LENGTH_RSP	MaxRxOctets: 2 octets, MaxRxTime: 2 octets, MaxTxOctets: 2 octets, MaxTxTime: 2 octets	master or slave

[BLE-LL]

7.3.1.3 Features Supported in the Link Layer for Over-the-air Communication

Bit Position	Link Layer Feature
0	LE encryption
1	Connection parameter request procedure
2	Extended reject indication
3	Slave-initiated features exchange
4	LE ping
5	LE data packet length extension
6	LL privacy
7	Extended scanner filter policies
8-63	RFU

[BLE-LL, 87]

7.3.2 LL Data PDU (Attribute PDU) Details

7.3.2.1 Attribute PDU Structure

Name	Size (octets)	Description
Attribute Opcode	1	The attribute PDU operation code bit7: Authentication Signature Flag bit6: Command Flag bit5-0: Method
Attribute Parameters	0 to (ATT_MTU - X)	The attribute PDU parameters X = 1 if Authentication Signature Flag of the Attribute Opcode is 0 X = 13 if Authentication Signature Flag of the Attribute Opcode is 1
Authentication Signature	0 or 12	Optional authentication signature for the Attribute Opcode and Attribute Parameters

Figure 57: Attribute PDU Format [BLE-ATT, 478]

7.3.2.2 Attribute PDU Details

Opcode	Attribute PDU Name	Parameters
0x01	Error Response	Erroneous opcode: 1 octet Erroneous attribute handle: 2 octets Error code: 1 octet (see table 7.3.2.3)
0x02	Exchange MTU Request	Client Rx MTU: 2 octets
0x03	Exchange MTU Response	Server Rx MTU: 2 octets
0x04	Find Information Request	Starting handle: 2 octets Ending handle: 2 octets
0x05	Find Information Response	Format: 1 octet → 0x01 – 16-bit Bluetooth UUIDs in information data → 0x02 – 128-bit UUIDs in information data Information data: 4 to MTU-2 octets
0x06	Find by Type Value Request	Starting handle: 2 octets Ending handle: 2 octets Attribute type: 2 octets Attribute value: 0 to MTU-7 octets
0x07	Find by Type Value Response	Handles information list: 4 to MTU-1 octets
0x08	Read by Type Request	Starting handle: 2 octets Ending handle: 2 octets UUID: 2 or 16 octets
0x09	Read by Type Response	Length: 1 octet Attribute data list: 2 to MTU-2 octets
0x0A	Read Request	Attribute handle: 2 octets
0x0B	Read Response	Attribute value: MTU -1 octets
0x0C	Read Blob Request	Attribute handle: 2 octets Value offset: 2 octets
0x0D	Read Blob Response	Part attribute value: 0 to MTU-1

Opcode	Attribute PDU Name	Parameters
0x0E	Read Multiple Request	Set of handles: 4 to MTU-1 octets
0x0F	Read Multiple Response	Set of values: 0 to MTU-1 octets
0x10	Read by Group Type Request	Start handle: 2 octets Ending handle: 2 octets UUID: 2 or 16 octets
0x11	Read by Group Type Response	Length: 1 octet Attribute data list: 2 to MTU-2 octets
0x12	Write Request	Attribute handle: 2 octets Attribute value: 0 to MTU-3 octets
0x13	Write response	--
0x52	Write Command	Attribute handle: 2 octets Attribute value: 0 to MTU-3 octets
0x16	Prepare Write Request	Attribute handle: 2 octets Value offset: 2 octets Part attribute value: 0 to MTU-5 octets
0x17	Prepare Write Response	Attribute handle: 2 octets Value offset: 2 octets Part attribute value: 0 to MTU-5 octets
0x18	Execute Write Request	Flags: 1 octet (0x00 – cancel, 0x01 – execute write)
0x19	Execute Write Response	-
0x1B	Handle Value Notification	Attribute handle: 2 octets Attribute value: 0 to MTU-3 octets
0x1D	Handle Value Indication	Attribute handle: 2 octets Attribute value: 0 to MTU-3 octets
0x1E	Handle value confirmation	--
0xD2	Signed write command	Attribute handle: 2 octets Attribute value: 0 to MTU-15 octets Authentication signature: 12 octets

[BLE-ATT]

7.3.2.3 Error Codes for Attribute PDU Opcode 0x01

Error Code	**Description**
0x01	Invalid handle
0x02	Read not permitted
0x03	Write not permitted
0x04	Invalid PDU
0x05	Insufficient authentication
0x06	Request not supported
0x07	Invalid offset (offset out of bounds)
0x08	Insufficient authorization
0x09	Prepare queue full
0x0A	Attribute not found
0x0B	Attribute not long (Read Blob Request not needed)
0x0C	Insufficient encryption key size
0x0D	Invalid attribute value length
0x0E	Unlikely error (unexpected error occurred)
0x0F	Insufficient encryption
0x10	Unsupported group type (as defined by higher layer specification, i.e. GATT)
0x11	Insufficient resources
0x80-0x9F	Application errors (as defined in in higher layer specification like GATT profile)
0xFD	Client characteristic configuration descriptor improperly configured
0xFE	Procedure already in progress
0xFF	Out of range

[BLE-ATT, 481-482]

8 References

Albazrqaoe, W., Huang, J., Xing, G., 2016. Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications. Presented at the Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, ACM, pp. 333–345. doi:10.1145/2906388.2906403

averpix, 2011. Clipart - generic-office-cpu [WWW Document]. openclipart. URL https://openclipart.org/detail/127231/genericofficecpu (accessed 3.12.17).

Black Hat, 2016. Gattacking Bluetooth Smart Devices - Introducing a New BLE Proxy Tool byl Slawomir Jasek.

BLE, n.d. Adopted Specifications [WWW Document]. Bluetooth Technology Website. URL https://www.bluetooth.com/specifications/adopted-specifications (accessed 3.12.17a).

BLE, n.d. Deprecated Specifications [WWW Document]. Bluetooth Technology Website. URL https://www.bluetooth.com/specifications/adopted-specifications/deprecated-specifications (accessed 3.12.17b).

BLE-GAP-AssignedNumbers, n.d. Generic Access Profile | Bluetooth Technology Website [WWW Document]. URL https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile (accessed 3.13.17).

BlueZ: Official Linux Bluetooth protocol stack [WWW Document], 2017. BlueZ. URL http://www.bluez.org/ (accessed 3.12.17).

Burns, S.F., 2005. Threat modeling: A process to ensure application security. GIAC Security Essentials Certification (GSEC) Practical Assignment.

Cauquil, D., 2017. DigitalSecurity/btlejuice [WWW Document]. GitHub. URL https://github.com/DigitalSecurity/btlejuice (accessed 3.12.17).

Cheung, H., 2005. How To: Building a BlueSniper Rifle - Part 1 [WWW Document]. tom’s guide. URL http://www.tomsguide.com/us/how-to-bluesniper-pt1,review-408.html (accessed 3.12.17).

Choi, M., Lee, J., Kim, S., Jeong, Y.-S., Park, J.-H., 2016. Location based authentication scheme using BLE for high performance digital content management system. Neurocomputing 209, 25–38.

Cook, G., n.d. reveng-Catalogue of parametrised CRC algorithms [WWW Document]. Sourceforge. URL http://reveng.sourceforge.net/crc-catalogue/17plus.htm (accessed 3.12.17).

Cooper, 2016. Hack.lu 2016 BtleJuice: the Bluetooth Smart Man In The Middle Framework by Damiel Cauquil.

dannya, 2006. Clipart - Bluetooth [WWW Document]. URL https://openclipart.org/detail/197362/mono-kbluetoothd (accessed 12.4.16).

Das, A.K., Pathak, P.H., Chuah, C.-N., Mohapatra, P., 2016. Uncovering Privacy Leakage in BLE Network Traffic of Wearable Fitness Trackers. Presented at the Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, ACM, pp. 99–104. doi:10.1145/2873587.2873594

Davies, A., 2016. Bluetooth 5.0 debut imminent, mesh still just-round-corner - Rethink IoTRethink Internet of Things – IoT News and Analysis [WWW Document]. URL http://rethink-iot.com/2016/06/14/bluetooth-5-0-debut-imminent-mesh-still-just-round-corner/ (accessed 12.3.16).

Degeler, A., 2015. How To Overcome Security Issues In BLE [WWW Document]. Stanfy. URL https://stanfy.com/blog/bluetooth-low-energy-security-issues-and-how-to-overcome-them/ (accessed 11.28.16).

doctormo, 2007. Clipart - BTC6100C UK Compact Keyboard [WWW Document]. URL https://openclipart.org/detail/4946/btc6100c-uk-compact-keyboard (accessed 3.12.17).

dominicgs, mikeryan, diracdeltas, zbac, EMCP, 2016. greatscottgadgets/ubertooth/Ubertooth Build Guide [WWW Document]. GitHub. URL https://github.com/greatscottgadgets/ubertooth (accessed 3.12.17).

Dunning, J.P., 2010. Bluetooth Threat Taxonomy. Virginia Polytechnic Institute and State University.

Fagerness, T., 2015. BLE using nRF51: ARM-GCC Build Environment [WWW Document]. All About Circuits. URL https://www.allaboutcircuits.com/projects/ble-using-nrf51-arm-gcc-build-environment/ (accessed 3.18.17).

Fawaz, K., Kim, K.-H., Shin, K.G., 2016. Protecting Privacy of BLE Device Users. Presented at the 25th USENIX Security Symposium, USENIX Association, Austin, TX.

Foringer, G., 2017. Introducing BLESuite and BLE-Replay: Python Tools for Rapid Assessment of Bluetooth Low Energy Peripherals [WWW Document]. nccgroup. URL https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2016/september/introducing-blesuite-and-ble-replay-python-tools-for-rapid-assessment-of-bluetooth-low-energy-peripherals/ (accessed 2.11.17).

Hilts, A., Parsons, C., Knockel, J., 2016. Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security.

Ibn Minar, N.B.N., 2012. Bluetooth Security Threats And Solutions: A Survey. International Journal of Distributed and Parallel systems 3, 127–148. doi:10.5121/ijdps.2012.3110

Jasek, S., 2016. Gattacking Bluetooth Smart Devices. Presented at the Blackhat, Las Vegas.

jslawek, forte916, 2017. securing/gattacker [WWW Document]. GitHub. URL https://github.com/securing/gattacker (accessed 3.12.17).

Koblitz, N., Menezes, A., undefined, undefined, undefined, undefined, 2016. A Riddle Wrapped in an Enigma. IEEE Security & Privacy 14, 34–42.

Koyama, Shunsuke, Satoshi, O., Nagashima, S., Matsuoh, D., Bernsten, F., 2011. Phone Alert Status Service.

Leavesified, n.d. Setup nRF51 Development on Linux – Leavesified [Blog] [WWW Document]. URL https://leavesified.wordpress.com/2016/03/24/setup-nrf51-development-on-linux/ (accessed 3.12.17).

LeBlanc, D., Howard, M., 2002. Writing Secure Code (Developer Best Practices), 2 edition. ed. Microsoft Press.

Lester, S., 2015. The Emergence of Bluetooth Low Energy [WWW Document]. Context. URL https://www.contextis.com//resources/blog/emergence-bluetooth-low-energy/ (accessed 11.28.16).

Lester, S., Stone, P., 2016. Bluetooth LE - Increasingly popular, but still not very private [WWW Document]. Context. URL https://www.contextis.com/resources/blog/bluetooth-le-increasingly-popular-still-not-very-private/ (accessed 11.28.16).

LightBlue Explorer - Bluetooth Low Energy on the App Store [WWW Document], 2016. App Store. URL https://itunes.apple.com/us/app/lightblue-explorer-bluetooth-low-energy/id557428110?mt=8 (accessed 3.12.17).

Lin, H., Bergmann, N.W., 2016. IoT Privacy and Security Challenges for Smart Home Environments. Information 7, 44. doi:10.3390/info7030044

Lu, Y., Meier, W., Vaudenay, S., 2005. The conditional correlation attack: a practical attack on bluetooth encryption. Presented at the Proceedings of the 25th annual international conference on Advances in Cryptology, Springer-Verlag, pp. 97–117. doi:10.1007/11535218_7

Lu, Y., Vaudenay, S., 2004a. Faster Correlation Attack on Bluetooth Keystream Generator E0, in: SpringerLink. Presented at the Annual International Cryptology Conference, Springer Berlin Heidelberg, pp. 407–425. doi:10.1007/978-3-540-28628-8_25

Lu, Y., Vaudenay, S., 2004b. Cryptanalysis of Bluetooth Keystream Generator Two-Level E0, in: SpringerLink. Presented at the International Conference on the Theory and Application of Cryptology and Information Security, Springer Berlin Heidelberg, pp. 483–499. doi:10.1007/978-3-540-30539-2_34

Luthra, G., 2015. Embedded controllers for the Internet of Things [WWW Document]. EDN. URL http://www.edn.com/design/sensors/4440576/Embedded-controllers-for-the-Internet-of-Things (accessed 11.28.16).

Madaan, P., Luthra, G., 2016. IoT for the smarter home [WWW Document]. Electronics EETimes. URL http://www.electronics-eetimes.com/design-center/iot-smarter-home (accessed 11.28.16).

Merritt, R., 2016. Open RTOS Targets Net of Things [WWW Document]. EETimes. URL http://www.eetimes.com/document.asp?doc_id=1329158 (accessed 11.28.16).

mikeryan, pabigot, dsempsrott, jennj, jodypattison, 2017. greatscottgadgets/ubertooth/Capturing BLE in Wireshark [WWW Document]. GitHub. URL https://github.com/greatscottgadgets/ubertooth (accessed 3.12.17).

Nordic Semiconductor's Official Github Account [WWW Document], n.d. GitHub. URL https://github.com/NordicSemiconductor (accessed 3.18.17).

nRF Connect for desktop [WWW Document], n.d. Nordic Semiconductor. URL https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Connect-for-desktop (accessed 3.12.17).

nRF Connect for mobile (previously called nRF Master Control Panel) [WWW Document], n.d. Nordic Semiconductor. URL https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Connect-for-mobile-previously-called-nRF-Master-Control-Panel (accessed 3.12.17).

nRF Toolbox App / Nordic mobile Apps / Products / Home - Ultra Low Power Wireless Solutions from NORDIC SEMICONDUCTOR [WWW Document], n.d. URL https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Toolbox-App (accessed 3.18.17).

Ossmann, M., n.d. greatscottgadgets/ubertooth [WWW Document]. GitHub. URL https://github.com/greatscottgadgets/ubertooth (accessed 3.12.17).

OWASP, 2016. IoT Testing Guides [WWW Document]. OWASP Internet of Things Project. URL https://www.owasp.org/index.php/IoT_Testing_Guides (accessed 12.27.16).

PACKET-SNIFFER SmartRF Protocol Packet Sniffer [WWW Document], 2014. Texas Instruments. URL http://www.ti.com/tool/packet-sniffer (accessed 3.12.17).

peplin, 2017. peplin/pygatt [WWW Document]. GitHub. URL https://github.com/peplin/pygatt (accessed 3.18.17).

PTES, 2017. High Level Organization of the Standard [WWW Document]. The Penetration Testing Execution Standard. URL http://www.pentest-standard.org/index.php/Main_Page (accessed 3.12.17).

Quinnel, R., 2015. BLE Module Lowers IoT Development Costs [WWW Document]. EE Times. URL http://www.eetimes.com/document.asp?doc_id=1327916 (accessed 11.28.16).

RaMBLE - Bluetooth LE Mapper - Android Apps on Google Play [WWW Document], 2017. Google Play. URL https://play.google.com/store/apps/details?id=com.contextis.android.BLEScanner&hl=en (accessed 3.12.17).

Ryan, M., 2017. mikeryan/crackle [WWW Document]. GitHub. URL https://github.com/mikeryan/crackle (accessed 3.12.17).

Ryan, M., 2015a. mikeryan/PyBT [WWW Document]. GitHub. URL https://github.com/mikeryan/PyBT (accessed 3.12.17).

Ryan, M., 2015b. scapy (fork with Bluetooth) [WWW Document]. Bitbucket. URL https://bitbucket.org/mikeryan1/scapy/downloads/?tab=branches (accessed 3.12.17).

Ryan, M., 2014. NSA Playset: Bluetooth Smart.

Ryan, M., 2013a. Bluetooth: with low energy comes low security, in: Presented as Part of the 7th USENIX Workshop on Offensive Technologies.

Ryan, M., 2013b. Bluetooth Smart: The Good, the Bad, the Ugly, and the Fix!

sandeepmistry, 2017a. sandeepmistry/noble: A Node.js BLE (Bluetooth Low Energy) central module [WWW Document]. URL https://github.com/sandeepmistry/noble (accessed 3.12.17).

sandeepmistry, 2017b. sandeepmistry/bleno [WWW Document]. GitHub. URL https://github.com/sandeepmistry/bleno (accessed 3.12.17).

Sandhya, S., Devi, K.A.S., 2012. Analysis of Bluetooth threats and v4.0 security features. ResearchGate. doi:10.1109/ICCCA.2012.6179149

Shaked, U., 2016. Reverse Engineering a Bluetooth Lightbulb. Uri Shaked.

ShawnHymel, 2015. Bluetooth Low Energy Peripherals with JavaScript. Shawn Hymel.

shokunin, 2008. Clipart - modern touch phone mobile [WWW Document]. URL https://openclipart.org/detail/19480/modern-touch-phone-mobile (accessed 12.4.16).

Snapshot, n.d.

Souppaya, M., Scarfone, K., 2013. Guidelines for managing the security of mobile devices in the enterprise. NIST special publication 800, 124.

Spill, D., Bittau, A., 2007. BlueSniff: Eve Meets Alice and Bluetooth. WOOT 7, 1–10.

Telit, 2016a. Telit acquires wireless communications assets to boost capabilities in low-power Internet of Things market [WWW Document]. Telit. URL http://www.telit.com/press-media/press-releases/press-details/item/telit-acquires-wireless-communications-assets-to-boost-capabilities-in-low-power-internet-of-things/ (accessed 11.28.16).

Telit, 2016b. Enabling End-to-End IoT Solutions - Telit [WWW Document]. Telit. URL http://www.telit.com/ (accessed 11.28.16).

Townsend, K., Cufi, C., Akiba, Davidson, R., 2014. Getting Started with Bluetooth Low Energy. O'Reilly Media.

ttrabun, 2016a. nccgroup/BLESuite [WWW Document]. GitHub. URL https://github.com/nccgroup/BLESuite (accessed 3.12.17).

ttrabun, 2016b. nccgroup/BLESuite-CLI [WWW Document]. GitHub. URL https://github.com/nccgroup/BLESuite-CLI (accessed 3.12.17).

ttrabun, 2016c. nccgroup/BLE-Replay [WWW Document]. GitHub. URL https://github.com/nccgroup/BLE-Replay (accessed 3.12.17).

Turk, V., 2014. The Internet of Things Has a Language Problem [WWW Document]. Motherboard. URL http://motherboard.vice.com/read/the-internet-of-things-has-a-language-problem (accessed 11.28.16).

vectorace, 2011. Clipart - Vector laptop or notebook [WWW Document]. openclipart. URL https://openclipart.org/detail/129931/vector-laptop-or-notebook (accessed 3.12.17).

Veilleux, D., 2017. Intro to Application-level Security Using the ECB Peripheral [WWW Document]. Blogs - Nordic Developer Zone. URL https://devzone.nordicsemi.com/blogs/721/intro-to-application-level-security-using-the-ecb-/ (accessed 1.14.17).

Web Bluetooth [WWW Document], 2017. webbluetoothcg. URL https://webbluetoothcg.github.io/web-bluetooth/ (accessed 3.12.17).

Woolley, M., 2015. Bluetooth Technology – Protecting Your Privacy [WWW Document]. Bluetooth Blog. URL http://blog.bluetooth.com/bluetooth-technology-protecting-your-privacy/ (accessed 11.28.16).

Yoshida, J., 2016a. NXP Set to Demystify Smart Homes [WWW Document]. EETimes. URL http://www.eetimes.com/document.asp?doc_id=1328579 (accessed 11.28.16).

Yoshida, J., 2016b. Silicon Labs' Geckos Aim at IoT, Take on NXP [WWW Document]. EETimes. URL http://www.eetimes.com/document.asp?doc_id=1328988 (accessed 11.28.16).

Yoshida, J., 2016c. Bluetooth 4.2 Unveiled: No Mesh Yet, But Big on IoT | EE Times [WWW Document]. URL http://www.eetimes.com/document.asp?doc_id=1324835 (accessed 12.3.16).

Ziegeldorf, J.H., Morchon, O.G., Wehrle, K., 2014. Privacy in the Internet of Things: threats and challenges: Privacy in the Internet of Things: threats and challenges. Security and Communication Networks 7, 2728–2742. doi:10.1002/sec.795