



Phillip Goriup, BSc

**Design and Implementation of an
Authenticated Presence Detection Protocol
based on Bluetooth Smart**

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisor

Ass.Prof. Dipl.Ing. Dr.techn. Christian Steger
Dipl.-Ing. Manfred Jantscher (CISC Semiconductor GmbH)

Graz, October 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, _____
Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am _____
Datum Unterschrift

Abstract

Detecting the presence of people opens up new applications for different domains. It can improve the user experience by detecting people's presence without physical interaction.

Using short range wireless technologies and their transmission ranges for implementing presence detection, is heavily investigated by researchers. They not only present applications based on presence detection, but also extend their researches with presence detection to add more information to it. Different technologies like Bluetooth, Bluetooth Smart or WiFi are used to generate this presence information. The most common way to identify the detected devices, is using their hardware addresses.

The aim of this thesis is to substitute this identification, based on hardware addresses, by cryptographic authentication. Therefore, the opportunities of using Bluetooth Smart for executing presence detection and combining it with authentication, have been investigated. This resulted in a protocol, which not only supports presence detection, but also continuously authenticates all detected devices.

An implementation has been provided, showing the practical opportunities of this authenticated presence detection based on Bluetooth Smart. Two applications for Android perform the presence detection and use the COYERO access environment as basis for the authentication. This implementation has been verified by conducting extensive experiments, which document its properties and limitations.

Keywords: Bluetooth Smart, Presence Detection, User Authentication, Protocol, Mobile Devices

Kurzfassung

Anwesenheitserkennung von Personen eröffnet neue Möglichkeiten von Anwendungen in verschiedenen Domänen. Diese kann das Nutzererlebnis verbessern, indem Personen erkannt werden ohne jeglicher Art von physischer Interaktion.

Die Verwendung von Kurzstanz-Drahtlostechnologien und ihrer Übertragungsreichweite, für die Implementierung von Anwesenheitserkennung, wird von diversen Forschern untersucht. Diese ermöglichen nicht nur eigene Anwendungen die auf Anwesenheitserkennung basieren, sondern erweitern ihre Forschungen auch um Informationen die mittels dieser gewonnen werden können. Unterschiedlichste Technologien, wie Bluetooth, Bluetooth Smart oder WiFi werden genutzt um diese Anwesenheits-Information zu generieren. Die Nutzung der Hardwareadressen der Geräte, ist der am weitesten verbreitetste Ansatz um diese zu identifizieren.

Das Ziel dieser Arbeit ist es, diese Identifizierung mittels Hardwareadressen, durch eine kryptographische Authentifizierung zu ersetzen. Dafür wurden die Möglichkeiten, Bluetooth Smart, für eine Anwesenheitserkennung, in Kombination mit einer Authentifizierung zu verwenden, untersucht. Dies resultierte in einem Protokoll, welches nicht nur eine Anwesenheitserkennung, sondern auch die durchgehende Authentifizierung aller erkannten Geräte unterstützt.

Eine Umsetzung dieses Protokolls wurde implementiert, welche die praktischen Möglichkeiten, dieser auf Bluetooth Smart basierenden authentifizierten Anwesenheitserkennung, zeigt. Zwei Applikationen für Android führen die Anwesenheitserkennung durch und nutzen die COYERO access Umgebung als Basis für die Authentifizierung. Diese Umsetzung wurde verifiziert durch die Durchführung mehrerer ausführlicher Experimente, welche die Eigenschaften und Einschränkung dieser zeigen.

Stichwörter: Bluetooth Smart, Anwesenheitserkennung, Nutzer Authentifizierung, Protokoll, Mobile Geräte

Acknowledgment

Firstly, I wish to express my honest gratitude to my supervisor Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger of the Institute for Technical Informatics at Graz University of Technology, who always found the time to do provide his support whenever I needed some guidance or had a question.

Besides my supervisor, I would like to thank the people from CISC, especially Dipl.-Ing. Manfred Jantscher and all the others from the office in Graz, for offering the opportunity of writing my thesis in corporation with them and providing place and material for conducting my practical experiments.

I am also grateful to my brother and my girlfriend for proofreading drafts of my work and for lending me a sympathetic ear, whenever I needed it. Finally, I would like to express my very profound gratitude to my mother for providing me with unfailing support throughout my years of study and my entire life. Thank you.

Graz, October 2017

Contents

Abstract	iii
Acknowledgment	v
List of Figures	viii
List of Tables	x
List of Listings	xi
List of Abbreviations	xii
1 Introduction	1
2 State of the Art	3
2.1 Overview of Wireless Technologies	3
2.2 Bluetooth and Bluetooth Smart	5
2.2.1 Architecture	5
2.2.2 Customizable Settings	13
2.3 Presence Detection	14
2.3.1 Related Work	15
2.3.2 Comparison of Presence Detection Systems	21
2.4 Cryptographic Authentication	24
3 Design of Authenticated Presence Detection	30
3.1 High Level Design	30
3.2 Recognition	34
3.3 Authentication	35
3.3.1 Authentication Protocol	35
3.3.2 BLE Interface	36
3.4 Authenticated Presence Detection	38
3.4.1 APD Challenge Response Protocol	39
3.4.2 Vulnerabilities of the Challenge Response Protocol	41
3.4.3 BLE Interface	42
3.4.4 Absence Timeout	42

4	Implementation for Android	44
4.1	Android	44
4.1.1	Application Development	45
4.1.2	Android and BLE	46
4.2	System overview	48
4.3	COYERO Detector Demo Application	49
4.4	COYERO SDK Extension and COYERO Client Demo Application	53
4.5	APD Server	57
4.5.1	Update Authentication Information	57
4.5.2	Visualize Information	57
4.6	Process Architecture	58
4.7	Scenarios	60
4.7.1	Entering the AoI	60
4.7.2	Leaving the AoI	62
5	Experiments and Results	63
5.1	Components	63
5.2	Basic Performance Measuring	65
5.2.1	Results for the Authentication Process	66
5.2.2	Results for the APD Challenge Response	68
5.2.3	Findings	71
5.3	Simultaneous Entering devices	71
5.4	System Test	72
5.4.1	Authentication Frequency	73
5.4.2	Absence Timeout	73
5.5	Unexpected Findings	75
6	Conclusion	80
6.1	Comparison to other Researches	81
6.2	Future Work	81
A	Custom Packet Format	83
B	APD Server RESTful Definition	85
	Bibliography	87

List of Figures

2.1	Data Rates and Transmission Distances. Adapted from Gupta [37, p. 3] and extended by information of Ribeiro [54], Bhandare [13], and Georgakakis et al. [36]	4
2.2	BLE Architecture adapted from Heydon [39, Chapter 3] and Townsend, Cufi and Davidson [64, p. 16].	6
2.3	The Link Layer States, adapted from the <i>Specification of the Bluetooth System 4.1</i> . [61, Vol. 6, pp. 16-19]	7
2.4	Link Layer Packets	7
2.5	Undirected Advertising and Scan Response Packets	8
2.6	L2CAP PDU	8
2.7	Example GATT Service	11
2.8	Advertising and Scan Response	12
2.9	AD Structure of Advertisement and Scan Response Data	12
2.10	Example GATT Device Profile Structure	13
2.11	Advertising Interval	13
2.12	Scan Interval and Scan Window	14
2.13	Connection Interval	14
2.14	Wireless Presence Detection	15
2.15	Location Context Nodes and Detectable Nodes of Wireless Network	16
2.16	Implicit and Explicit AoI Definition	18
2.17	Sub-Area Detection	18
2.18	Bi Directional Presence Detection	20
2.19	Different Nodes acting as Gateway in a Presence Detection System	22
2.20	CA as trusted third Party	26
2.21	Attacks on a Communication Channel	28
2.22	COYERO access environment, adapted from [28]	29
3.1	Use Case Diagrams	31
3.2	High Level Design	33
3.3	Authentication Protocol	37
3.4	Challenge Response Protocol for Authenticated Presence Detection	40
3.5	Limited Replay Vulnerability of the Challenge Response Protocol	41
3.6	Scan Procedure for Challenge Advertisement	42
3.7	Scan Procedure for Challenge Response Advertisement	42

4.1	Android Build Process, adapted from the documentation for <i>The Build Process</i> [63]	45
4.2	System Architecture	49
4.3	Logical Architecture of COYERO Detector	50
4.4	COYERO Detector Views	51
4.5	Layer System of the Authentication Gatt Server	52
4.6	Validity Period of Challenges	53
4.7	Logical Architecture of COYERO Client	55
4.8	COYERO Client UI Extensions	55
4.9	Web Server Information Visualization	58
4.10	Process Architecture	59
4.11	Scenario Entering the AoI	60
4.12	Scenario Leaving the AoI	62
5.1	Java Swing Tool for controlling BLE112 Dongles	64
5.2	Experiment setup of the BLE112 dongles	65
5.3	Basic Performance Measuring Experiment Setup	66
5.4	Average Times and Standard Deviations of Authentication Steps	67
5.5	SONY Performance compared to HTCN Performance	68
5.6	Error Ratio of Authentication Steps	69
5.7	Average Authentication Delay of all Measurements	70
5.8	Bad Performance of LGNE in L_tL_d compared to other Devices	70
5.9	Average Times of simultaneously entering Devices	72
5.10	Number of currently participating APD-clients and Authentication Frequencies of the Android Devices running COYERO Client for the different Android Devices running COYERO Detector	74
5.11	HTCN Absence Timeout Analysis	75
5.12	LGG4 Absence Timeout Analysis	76
5.13	LGNE Absence Timeout Analysis	76
5.14	SONY Absence Timeout Analysis	76
5.15	Signal Strength Differences with same Settings	79
A.1	Challenge Packet Format	83
A.2	Challenge Response Packet Format	84
A.3	Authentication Data Packet Format	84

List of Tables

2.1	Types of Bluetooth Devices	5
2.2	Comparison of Related Work in Presence Detection	25
3.1	Notation for the Authentication Protocol	35
3.2	Properties of the GATT Authentication Service	36
3.3	GATT Characteristic Client Authentication Data	38
3.4	GATT Characteristic Kiosk Authentication Data	38
3.5	GATT Characteristic Session ID	38
3.6	Notation Extension for the Challenge Response Protocol	39
4.1	Android BLE Scan Settings for API ≥ 21 , based on Information of [15]	46
4.2	Android BLE Advertising Mode Settings, based on Information of [15]	47
4.3	Android BLE TX Power Settings, based on Information of [15]	47
4.4	Android BLE Connection Priority Settings, based on Information of [15]	48
4.5	Properties of the COYERO Detector Application	50
4.6	Properties of the COYERO Client Application	54
5.1	Android Devices used for the Experiments	63
5.2	Different Experiment Distance Contexts	65
5.3	Different Experiment Traffic Contexts	66
5.4	Optimal Absence Timeouts for the Devices	75
5.5	Device specific Maxima of simultaneously performed BLE Operations	77
6.1	Embedding of the APD Design	81
B.1	RESTful Specification of the APD Server's Interface	86

List of Listings

4.1	Scan API Callback Difference, based on Information of [18] and [57]	46
4.2	Call for Checking Advertisement Support, based on Information of [17]	47
4.3	Connect to a GATT Server, based on Information of [19]	48
4.4	Open a GATT server, based on Information of [23]	48

List of Abbreviations

AAR	Android Archives
AoI	Area of Interest
APD	Authenticated Presence Detection
API	Application Programming Interface
APK	Android Application Package
BLE	Bluetooth Low Energy
CA	Certificate Authority
COM	Communication
CRC	Cyclic Redundancy Check
dBm	Decibel-Milliwatts
DH	Diffie-Hellman
DN	Detectable Node
ECC	Elliptic Curve Cryptography
GAP	Generic Access Profile
GAS6	Samsung Galaxy S6
GATT	Generic Attribute Profile
GPS	Global Positioning System
HTCN	HTC Nexus 9
HTML	Hypertext Markup Language
ID	Identifier
JAR	Java ARchive
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
LCN	Location Context Node
LGG4	LG-G4 H815
LGNE	LG Nexus 5x

MAC	Message Authentication Code
MIC	Message Integrity Check
MITM	Man-in-the-Middle
MTU	Maximum Transmission Unit
N/A	Not Available
NFC	Near-Field Communication
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
REST	Representational State Transfer
RN	Receiving Node
RSA	Rivest-Shamir-Adleman cryptosystem
RSSI	Received Signal Strength Indication
SDK	Software Development Kit
SN	Sending Node
SONY	Sony Xperia Z5
TX	Transmission
UI	User Interface
USB	Universal Serial Bus
UUID	Universally unique identifier
WiMAX	Worldwide Interoperability for Microwave Access

Chapter 1

Introduction

Detecting, that certain people are present within a specific area can be used for improving the performance of services and processes. For instance, a store, which is selling goods to customers and offering an online pre-ordering tool, could detect the presence of customers in their store and prepare the pre-ordered good before the customer even interacts with an employee. This would not only improve the customer's experience, because they do not need to wait for their order to be prepared, but also increase the efficiency of the employees. Another example are attendance systems of all kinds. An organizer of a meeting could get a notification when all invited employees are present in the meeting room, so that they know, when they can start the meeting. These examples show the value, which could be added by extending systems with presence detection.

Indoor presence detection systems are usually implemented by using wireless transmitters and their signal strength for defining a certain area, in which devices are detected. Many researchers investigate in the possibilities of presence detection with wireless technologies and try not only to create applications based on presence detection, but also to add value to their particular researches by extending it with presence detection. Most of the researchers rely on well established and wide spread wireless technologies, like WiFi, Bluetooth and Bluetooth Smart. Using WiFi and Bluetooth has the advantage, that both technologies are wide spread and that there is often an established infrastructure available. Bluetooth Smart often requires additional hardware in the form of beacons. However, beacons are very easy to set up and quite cost-effective. Another major advantage of Bluetooth Smart is its low energy nature, which makes it very interesting when using mobile devices that rely on a battery. The amount of mobile devices increases constantly and they support a variety of wireless technologies, which makes detecting people's mobile devices one of the best ways for detecting people themselves. This opens up possibilities of not only detecting people, but also enables authenticating them during the detection process, by using an environment like COYERO access, which supports authenticating people via their mobile devices. An introduction to Bluetooth Smart, state of the art presence detection research and well established cryptographic authentication systems are presented in [Chapter 2](#) of this work.

[Chapter 3](#) presents a design, which combines a presence detection via Bluetooth Smart and a cryptographic authentication for achieving an authenticated presence detection. This design consists of three phases. Firstly, the recognition of the device is done by receiving a Bluetooth Smart advertisement. The advertisement's signal strength defines the

area, within devices are detected. In phase two, the recognized device is authenticated. A Public Key Infrastructure builds the basis for the cryptographic authentication procedure. This authentication uses digital certificates, issued by the Public Key Infrastructure, and performs a key exchange to create an authenticated session. The last phase is performing a challenge response protocol, based on the previously created authenticated session. The challenger continuously broadcasts a challenge, which can be scanned, solved and responded by nearby devices. Each device, that is already in an authenticated session is able to solve the challenge. When a device is able to receive a challenge and send a correctly solved challenge response back to the challenger, this particular device is not only within a certain range, but also authenticated.

An implementation of the presented design has been carried out utilizing Android devices and the COYERO access environment. This implementation is shown in [Chapter 4](#). Two Android applications have been developed, which provide the required functionalities. One application defines the detection area, when installed on an Android device. This device is placed in a relevant location, for instance in a shop. It is able to detect people's presence in a certain area around the device. To do so, these people have to have the second application installed on their mobile devices. When both applications recognize each other, they perform a mutual authentication, create an authenticated session and start to perform the authenticated presence detection. Additionally, the information about the currently authenticated and present people is sent to a web server and can be requested by using a web browser.

The properties of the implementation and the possibilities for using it in a real life scenario have been investigated by conducting several experiments. The aim of these experiments was to determine the impact of the separate steps within the design onto the systems performance. Furthermore, insights into the performance of Android, when it is used for performing Bluetooth Smart operations, have been gained and device specific behaviour has been revealed and documented. It has been shown, that an authenticated presence detection via Bluetooth Smart is possible by only utilizing Android devices. The results of the experiments and identified limitations are shown in [Chapter 5](#).

Chapter 2

State of the Art

This chapter gives an overview of existing wireless technologies and shows the most important architectural features of Bluetooth Smart, which is the utilized wireless technology of the presented design in this work. Afterwards, the different fields of presence detection in science are presented and proposed systems of other researchers are discussed and compared. The last part of this chapter shows the currently available cryptographic concepts, which can be used for achieving a user authentication followed by a brief explanation of a state of the art cryptographic environment, namely COYERO access.

2.1 Overview of Wireless Technologies

Use cases for wireless technologies have different requirements regarding data rates or transfer distances. Some require high data rates and have to transfer their data over long distances, where others require low data rates over a small distances. There are various different wireless technologies available. Small distance applications like remote control of the TV or getting access to a building can utilize Near-Field Communication (NFC), Bluetooth, Infra-red or Zigbee. Navigation of ships requires a wide distance support, which would make satellite communication like Global Positioning System (GPS) a good choice. For transferring huge data junks Ultra-wideband and Wifi would be a good choice. [Figure 2.1](#) shows a set of wireless technologies, their typical data rates and transfer distances, and lets conclude their suitable fields of application.

Wireless technologies are usually allocated with a network category based on their operational distance. These network types are an artificial construct which enables classification of use cases. Based on the work of Gupta [37, p. 2], some of the categories are as follows:

- **Body Area Network:** A body area network is typically build by devices that are worn by people like a smart phone, smart watch or heart rate monitor. Representative technologies are Bluetooth, Bluetooth Smart, NFC.
- **Personal Area Network:** Generally build by personal devices, which are communicating within a few meters like smart phones, printers, and cameras. Representatives are Bluetooth, Bluetooth Smart, Zigbee.

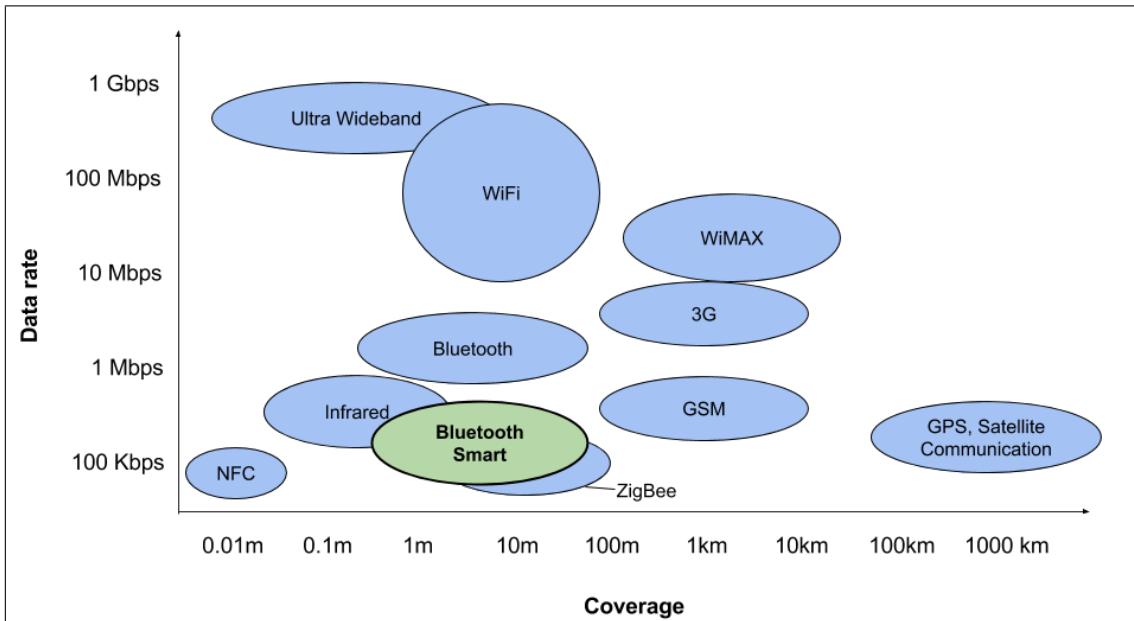


Figure 2.1: Data Rates and Transmission Distances. Adapted from Gupta [37, p. 3] and extended by information of Ribeiro [54], Bhandare [13], and Georgakakis et al. [36]

- **Local Area Network:** Covers networks, which operate within a few hundred meters. This could be a company building or a campus. A representative technology is WiFi.
- **Metropolitan Area Network:** Ranges up to several kilometres are supported. Could be used in a small town or a metropolis. A representative technology is Worldwide Interoperability for Microwave Access (WiMAX).
- **Wide Area Network:** Supports communication up to several thousand kilometres or in other words, worldwide communication. A representative technology is GPS.

There are more categorisations like a neighbourhood area network as mentioned by Reiter [53] or a home area network as mentioned by Movassaghi et al. [48]. Most of the additional network definitions can be classified with one of the previously mentioned categories as well. However, their name provides often additional domain specific context.

A lot of research is going on to link wireless technologies to different use cases, by comparing their data rates, transfer distances, power consumptions, and other properties. Mahmood, Javaid and Razzaq [45] review wireless technologies and their usability for home area networks and neighbourhood area networks. Other papers focus on comparing the lower range and low power wireless technologies, like the published works from Movassaghi et al. [48] and Dementyev et al. [29].

Presence detection is a low range application, which does not require huge data throughput and usually builds a personal area network. Bluetooth Smart is a suitable representative which has all these properties.

Name	Description	Mode
Bluetooth Classic	Supports only Bluetooth Classic connections Bluetooth specification < 4.0	N/A
Bluetooth Smart	Supports only Bluetooth Smart connections Bluetooth specification \geq 4.0	Single-Mode
Bluetooth Smart enabled	Supports Bluetooth Classic and Bluetooth Smart connections Bluetooth specification \geq 4.0	Dual-Mode

Table 2.1: Types of Bluetooth Devices

2.2 Bluetooth and Bluetooth Smart

Bluetooth Smart, also known as Bluetooth Low Energy (BLE), was first published in the *Specification of the Bluetooth System 4.0*. [60] by the Bluetooth Special Interest Group in 2010. It has extended the already established Bluetooth standard with a low energy solution, that focuses on long lifetime of devices for transmitting a limited amount of data. Despite the increasing speed in wireless technologies, as shown by Heydon [39, Chapter 1], the main target of BLE is low power consumption by sacrificing speed.

The increasing number of available BLE devices, makes it a widely deployed technology. The article *Driven by Standards, in 2016 Wi-Fi Direct and BLE Protocols Will Ship in 2 and 2.9 Billion Devices Respectively* [32] states, that about 2.9 Billion devices supporting BLE, which have been expected to be shipped in 2016. This and the focus on low power consumption makes BLE a good choice for a various applications.

As mentioned before, BLE has been introduced as a part of the *Specification of the Bluetooth System 4.0*. [60]. This BLE Specification has been extended several times since then. The experiments, which have been conducted in Chapter 5, have used devices with Bluetooth version 4.1. Therefore, all citations of the Bluetooth Specification that are carried out from now on, are done for the *Specification of the Bluetooth System 4.1*. [61].

The extension of classic Bluetooth with BLE resulted in three different types of Bluetooth enabled devices as shown in table Table 2.1. *Bluetooth Classic* devices can communicate with other *Bluetooth Classic* devices and with *Bluetooth Smart enabled* device. *Bluetooth Smart enabled* devices can communicate with all types of devices. *Bluetooth Smart* devices can communicate with other *Bluetooth Smart* devices and with *Bluetooth Smart enabled* devices. [64, pp. 3-4]

2.2.1 Architecture

To get an impression of how BLE is working, its architecture is explained in this section. This is only an introduction which covers the most important points. All details can be found the in *Specification of the Bluetooth System 4.1*. [61].

The architecture of BLE is divided into three parts: a controller, a host, and an application layer. The controller represents the lowest layer, that is responsible for receiving the radio waves and generates interpretable data packets. The host, which is usually implemented as a software stack, takes this data packets and adds logical management, like maintaining more than one connections at the same time. The application layer then

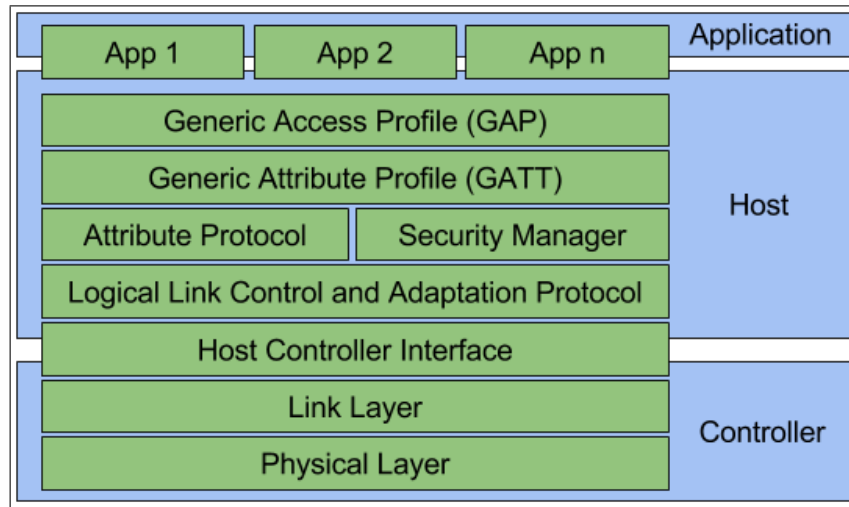


Figure 2.2: BLE Architecture adapted from Heydon [39, Chapter 3] and Townsend, Cuff and Davidson [64, p. 16].

make use of the host’s software stack to implement a use case. [39, Chapter 3]

Figure 2.2 shows the architecture and the participating layers within the three previously mentioned parts. These layers are explained in the following paragraphs.

Physical Layer: It handles the conversion of analogous signals to digital signals. To decrease interference by other radios, BLE works on the 2.4 GHz frequency band and divides it into 40 channels. Each of these channels has a range of 2MHz on which the data is transferred. The Transmission (TX) power of BLE is specified in Decibel-Milliwatts (dBm). The signal shall be transmitted with a dBm value ranging from -20 to +10. [39, Chapter 5, Section 5.7], [61, Vol. 6, pp. 16-19]

Link Layer: The link layers work flow can be seen as a state machine, which is illustrated in Figure 2.3. Advertising means emitting a signal for all BLE devices in a surrounding area. A scanning devices is looking for such advertisement signals. When the hardware supports it, a scanning device can change to the initiating state and start a connection with the advertising device for exchanging data. For distinguishing advertisement and connection signals, the 40 channels of the physical layer are divided into two logical groups. Three channels become advertising channels and 37 become data channels. Devices use the three advertising channels for sending advertisements, which can be scanned by surrounding devices. On the other hand, they use the 37 data channels for transferring data, when they are in an active connection. The actively used data channel is specified by an adaptive frequency hopping algorithm. There are two roles within an established connection. The so called master is always the initiator of a connection, whereas the slave is the one that has been connected to. A master can be connected to more than one slaves at the same time and a slave can be connected to more than one master at the same time. Additionally, a device can be a master and a slave at the same time. [61, Vol. 6, pp. 32-36]

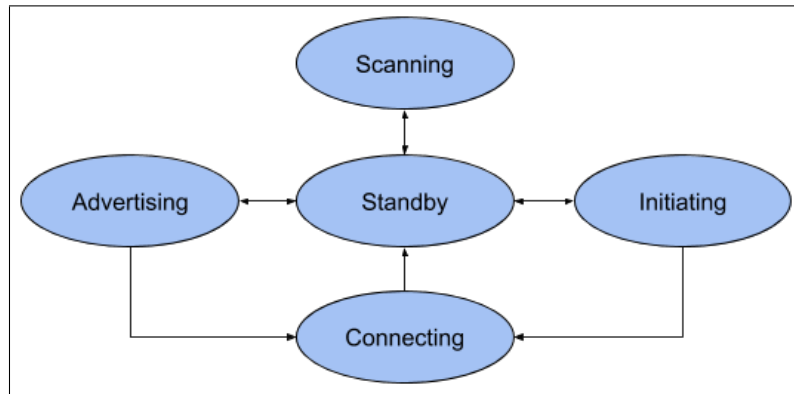


Figure 2.3: The Link Layer States, adapted from the *Specification of the Bluetooth System 4.1*. [61, Vol. 6, pp. 16-19]

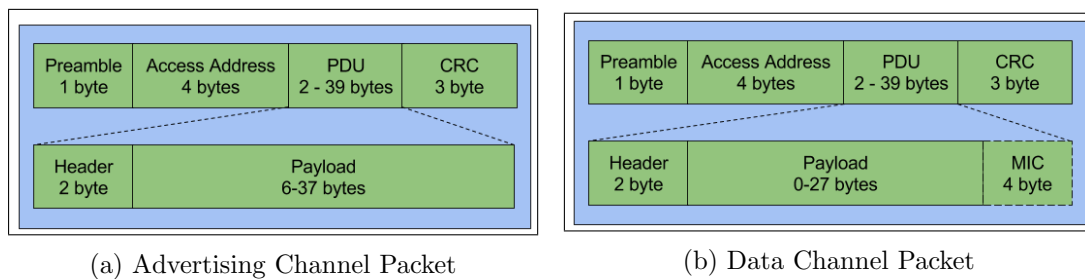


Figure 2.4: Link Layer Packets

For sending any information on either the advertising channels or the data channels a packet structure is defined by the link layer. The structure is the same for data and advertising packets. It consists of a preamble which is used for synchronizing the frequency, an access address for distinguishing between data and advertising packets, a Cyclic Redundancy Check (CRC) for ensuring that all received bits are valid and a Protocol Data Unit (PDU). The PDU contains either an advertisement or data. Advertising channel PDUs, as seen in Figure 2.4a, consist of a two byte header, followed by a 6 to 37 byte payload. This payload is divided into a 6 byte header followed by 0 - 31 bytes of custom data, which is illustrated in Figure 2.5. This means, a maximum of 31 bytes of custom data can be transferred with one advertisement. Data channel PDUs consist of a two byte header, then a 0-27 byte payload and an optional 4 byte Message Integrity Check (MIC). The message integrity check is only used in an encrypted channel. Figure 2.4b shows the format of a data channel packet. These data channel PDUs are the basis for the higher protocol's data transfer. [61, Vol. 6, pp. 37-39, pp. 45-46]

Host Controller Interface: The host controller interface provides a standardized way of communication between the host and the controller. This separation is very useful when deploying one host with controllers of different manufacturers. The host controllers logical interface definition contains commands plus their expected effects. The physical interface definition specifies how to physically transport the data on physical channels, like Universal Serial Bus (USB). [39, Chapter 3, Section 3.1.4]

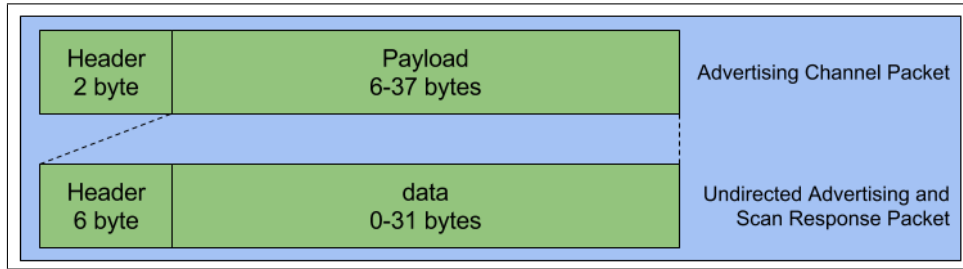


Figure 2.5: Undirected Advertising and Scan Response Packets

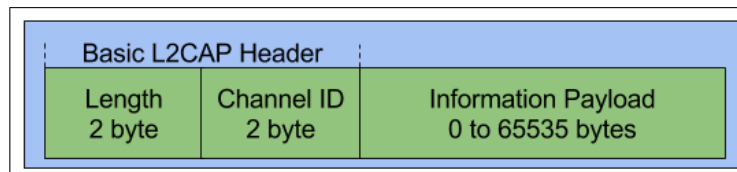


Figure 2.6: L2CAP PDU

Logical Link Control and Adaptation Protocol (L2CAP): It is basically responsible for two different things. It fragments big data payloads of higher protocol layers and encapsulates the fragments, so that they are transmittable via the underlying protocols. Additionally, it serves as a multiplexer, which redirects the received data to higher level protocols and vice versa. This is done with channels ids, that indicate the corresponding higher protocol. BLE uses three channels. Id 0x4 is used for the attribute protocol, 0x6 for the security manager, and 0x5 for the L2CAP's own signalling channel. The signalling channel is used when a received message can not be identified, is invalid or had a wrong length. Additionally, it is used when the slave wants to request an update of the connection parameters. [37, pp. 203-207], [64, p. 25]

The data, which is received by higher protocols is packed into a L2CAP PDU. This packet can be seen in Figure 2.6. The first two bytes contain the length of the payload, followed by 2 byte that contain the channel ID, which is used for identifying the endpoint of the data. This L2CAP PDU is then fragmented and transferred to the link layer via the host controller interface. [61, Vol. 3, p. 47], [37, p. 205]

The size of the data which is received by the higher protocols is limited to the size of the Maximum Transmission Unit (MTU). This value informs other BLE devices about the acceptable maximum size of L2CAP payloads. The default and minimum value of the MTU is 23 bytes. The theoretical maximum MTU is 65535 bytes. [37, p. 202], [61, Vol. 6, p. 46]

Security Manager: This paragraph contains security related information which makes use some terms explained in 2.4. BLE provides the processes of pairing and bonding for devices. When two devices perform a pairing or a bonding procedure, keys are exchanged for establishing security features. The security manager is responsible for performing this key distribution. Pairing means temporary exchanging a secret key, that enables switching to a secured connection link. Bonding must follow after a completed pairing procedure and stores secrets permanently. These permanent secrets are then used for

faster re-establishment of a secured connection. The previously mentioned keys are derived from a shared secret. This shared secret can be generated with three different modes. The mode *Just Works* sets the shared secret to zero, meaning the communication is not authenticated. This method is used, when at least one of the devices has no input and output capabilities. The mode *Passkey Entry* can be used, when one device is able to display a 6-digit numerical passkey or the user knows the 6-digit numerical passkey of this device and the second device is able to receive input by the user. This 6-digit numerical passkey is the basis for an authenticated and secured communication between the devices. *Out of Band* is the third mode for exchanging a shared secret. The *Out of Band* shared secret is transferred with a different technology, like NFC. This generates a 128 bit shared key, which is then used as basis for the security features. [37, pp. 214-216], [39, Chapter 3, Section 3.2.2]

Under the premise, that the used channel within an *Out of Band* pairing can not be eavesdropped, *Out of Band* is the only pairing method that is secure. *Just Works* and *Passkey Entry* are vulnerable to an passive eavesdropping attack as shown by Ryan [55].

Attribute Protocol: The attribute protocol is an additional abstraction layer, that organizes data in attributes. Three things are associated to an attribute. The *Attribute Handle* is a 16-bit value which is used for accessing the attributes value. The *Attribute Type* is uniquely identified by an 128-bit Universally unique identifier (UUID), as defined in *ITU-T Rec. X.667* [40], and can be specified by anybody without restrictions. The *Attribute Value* is the actual value linked to the attribute. Additionally, permissions, which handle the access, can be linked to an attributes and mark them as readable, writeable, or both. The attribute protocol defines the roles of a server and a client. When two devices are in a connection, they can become a client, a server, or both. These roles are not related to the roles of master or slave of the link layer, which are explained in 2.2.1. [61, Vol. 3, pp. 482-485]

There are six defined method types which are used for establishing a communication between server and client. The following list is based on the *Specification of the Bluetooth System 4.1*. [61, Vol. 3, p. 488] and contains the six define methods.

- A *request* is a command that is sent from a client to a server and requires an answer in form of a response.
- A *response* is the answer from a server sent to a client following a request.
- A *command* is send from a client to a server.
- A *notification* is a server initiated message sent to a client.
- An *indication* is a server initiated message sent to a client, which requires a confirmation of the client.
- A *confirmation* is an answer from a client, send after an indication.

There are several operations that are supported by the attribute protocol. All operations are implemented with one of the previously mentioned methods. The operations are used to perform server configurations, error handling, finding server information, reading

and writing values, and notifying clients about updates. A detailed list of these operations can be found in the *Specification of the Bluetooth System 4.1*. [61, Vol. 3, pp. 492-524] and an overview is published by Townsend, Cufi and Davidson [64, pp. 26-28].

Generic Attribute Profile (GATT): GATT is located on top of the attribute protocol. In contrast to the flat organized attributes of the attribute protocol, where no conceptual grouping is possible, GATT introduces functions for grouping and achieving a hierarchical organization. To do so, the concepts of services and characteristics are introduced. Services and characteristics are implemented via attributes of the attribute protocol. [64, pp. 32-33]

A service is a collection of data, which is related to the same feature or purpose. It can either be a primary or a secondary services. Primary services are intended to expose the main functionality of the devices and can be discovered by GATT defined discovery procedures, meaning a client can retrieve information about the structure of the service. Services can be included from other services to add additional context to it. Secondary services shall be included by other services and can only be discovered, when the client already knows about the service, which includes the secondary service. [61, Vol. 3, p. 542], [64, p. 65]

Characteristics represent the actual data that is provided by a service. It consists of a *declaration*, a *value*, and optional *descriptors*. The *declaration* holds the information about the characteristics, like its unique UUID. This UUID is used to identify the characteristic. The *value* represents the assigned data and the *optional descriptors* may contain additional meta data like a user friendly description string. Once a service is discovered, the characteristics of the service can be discovered as well. After discovering the characteristics, a client can read and write them, if the assigned permissions allow it. [37, p. 265, pp. 273-274]

GATT defines procedures, which use the operations defined by the attribute protocol, for enabling convenient usage of services and characteristics. Four different types of procedures can be distinguished within GATT. *Discovery procedures* are used for discovering information about service structures and characteristic properties. *Client initiated characteristic procedures* are used for reading and writing the servers characteristics and their descriptors. *Server initiated characteristic procedures* are used by the server to send updates about a characteristics to the client. The last are *server configurations*, which are used for configuring the server. An example service can be seen in Figure 2.7. [39, Chapter 10, Section 10.7], [61, Vol. 3, p. 558]

Only one procedure of is available for server configurations. This is the so called *exchange MTU* procedure. The client can use this procedure to exchange the MTU with the server for increasing or decreasing the performance of data transfers. The client is sending its MTU to the server and the server either responds with an error or with its own MTU. If the server responds with its MTU, the minimum of the client's and server's MTU will become the MTU of both. [61, Vol. 3, p. 560]

Generic Access Profile (GAP): In contrast to GATT, which defines the data structure and its procedures, GAP defines how a device discovers other devices, how it connects to them and how it presents useful information to other devices. Basically it defines how

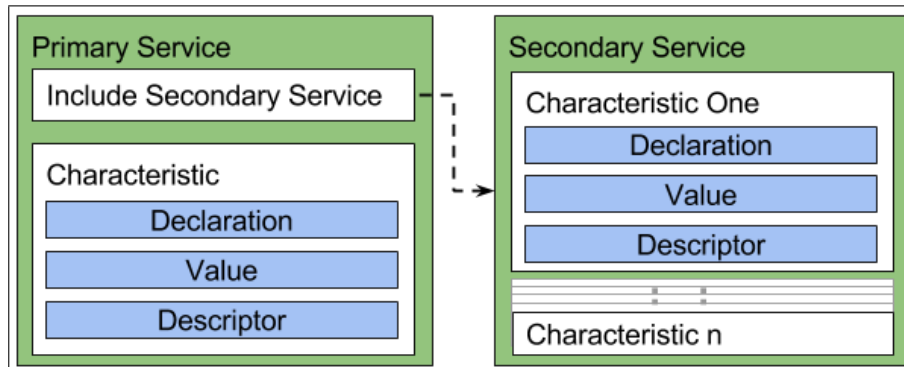


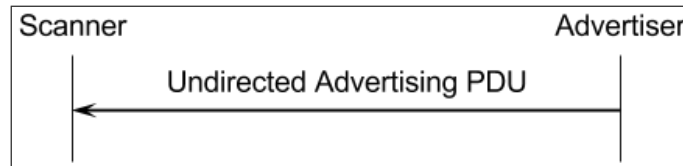
Figure 2.7: Example GATT Service

the devices have to use the previously introduced protocol layers and structures for communicating with each other. Furthermore, it provides a naming convention that shall be used within an User Interface (UI), to enable an easy and unified user experience for all BLE applications. One of these conventions is that on a UI the term “Bluetooth Passkey” shall be used for indicating the 6-digit numerical passkey. [37, pp. 305-306]

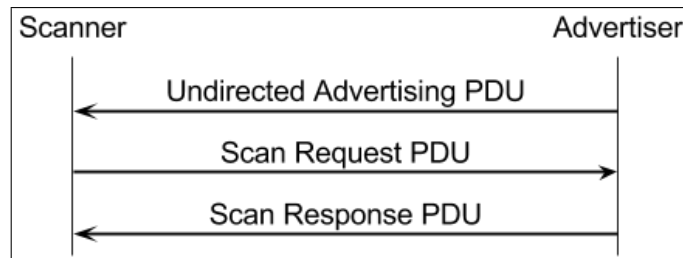
Devices can take over several roles in BLE. These roles usually come in pairs. Firstly, broadcaster and observer. A broadcaster is a device which advertises data. An observer, on the other hand, is scanning for advertisements. When the devices intend to establish a connection, they are called peripheral and central. A peripheral transmits a signal, like the broadcaster. However, it is connectable so that a central can initiate a connection to a peripheral. A central takes over the role of a master and a peripheral takes over the role of a slave in the link layer (see 2.2.1). The attribute protocol’s roles of a client and a server are not related to a peripheral and central. This means, a device can be a client or a server, regardless of the device being a peripheral or central. When the link layer supports it, devices can act in more than one GAP role at the same time. [61, Vol. 3, pp. 292-294], [64, p. 37]

When a peripheral or a broadcaster is transmitting data on the advertising channels of the link layer (see 2.2.1), they are called *advertiser*. Centrals and observer listening for such advertisements are called *scanner* and are able to receive data from advertisers. Additionally, scanner can send a scan request to an advertiser, which then is answered with a scan response. This enables sending two advertising packets to other devices without establishing a connection. Only scanning for advertisements without sending a scan request is called passive scanning, whereas sending a scan request is called active scanning. These two behaviours are illustrated in Figure 2.8. [37, p. 309]

The undirected advertising PDUs and the scan response carry the custom data that shall be sent to a scanner. Those two structures are very similar and can be seen in Figure 2.5. The space for custom data is 31 bytes. GAP defines a format for these advertising and scan response data. It fills up the remaining 31 byte front to back with so called AD-structures (see Figure 2.9). These AD-structures have a variable length. Therefore it may happen that not all of the 31 bytes are used. Unused space is filled with zeros. AD-structures start with one byte indicating the length. Then there is a one byte AD-type definition, followed by length-1 bytes of AD-data. [61, Vol. 3, p. 387]



(a) Passive Scanning



(b) Active Scanning

Figure 2.8: Advertising and Scan Response

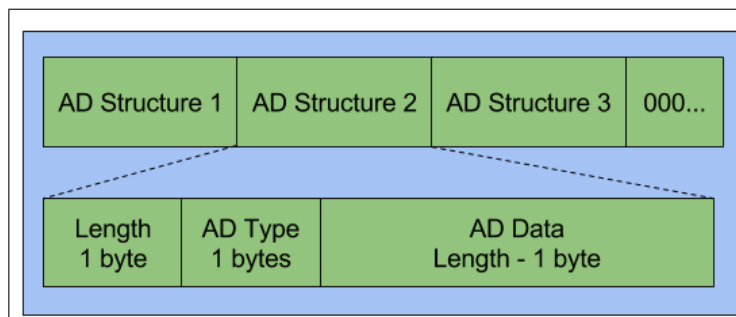


Figure 2.9: AD Structure of Advertisement and Scan Response Data

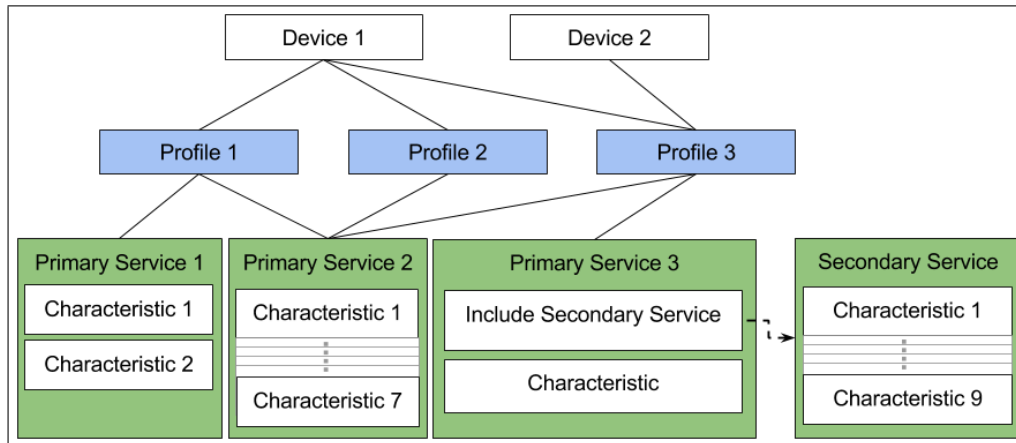


Figure 2.10: Example GATT Device Profile Structure

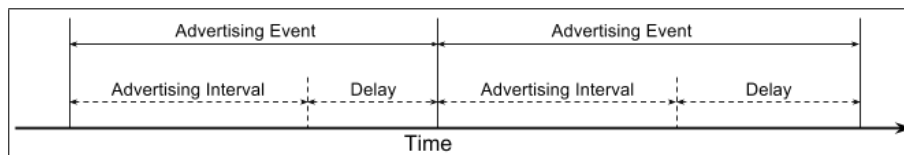


Figure 2.11: Advertising Interval

Application Layer: It is the highest layer in the protocol stack and represents applications, which use the underlying layers for implementing use cases. So called GATT based profiles are defined for making designing a BLE device easier. If two devices support the same profile, they are able to perform the required actions for the use case, which the profile was defined for. Profiles define the behaviour of devices like how to discover other devices and how to connect to them. Furthermore, they contain a definition of the used GATT services, a definition of the characteristics, and a detailed description of the use case. A profile may contain more than one service. However, a GATT service can be part of more than one profile as well. Additionally, multiple profiles can be supported from one device. This leads to a structure as illustrated in [Figure 2.10](#). [37, pp. 333-335], [39, Chapter 3, Section 3.3.3], [64, p. 14], [66]

2.2.2 Customizable Settings

It is possible to customize several settings of the BLE device. This section highlights the most important ones, which have been used by the presented design in [3](#).

Advertising Interval: For explaining the advertising interval, the concept of an advertising event has to be clarified. An advertising event is the process of sending one or more advertising PDUs on all used advertising channels. This is managed by the link layer. The length of such an advertising event is the combination of the advertising interval and a pseudo random delay as illustrated in [Figure 2.11](#). This advertising interval has to be a value within the range of 20 milliseconds up to 10.24 seconds and it has to be a multiple of 0.625 milliseconds. [61, Vol. 6, pp. 61-62]

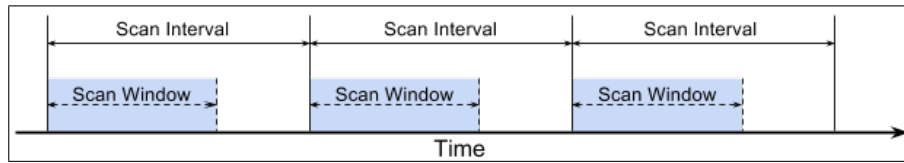


Figure 2.12: Scan Interval and Scan Window

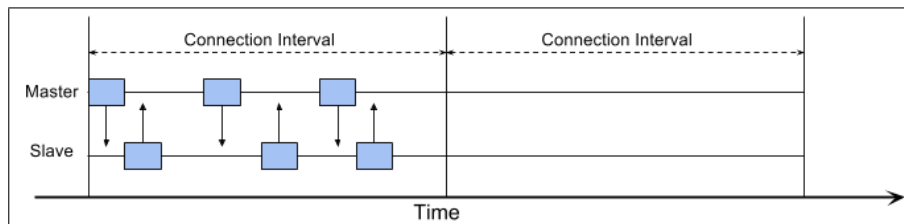


Figure 2.13: Connection Interval

Scan Interval and Scan Window: The scan interval defines, when the scanner starts scanning for advertisements. The scan window defines how long the scanner is active. This is illustrated in [Figure 2.12](#). [[39](#), Chapter 8, Section 8.5.2]

Connection Interval, Slave Latency and Supervision Timeout: The connection interval defines the length of so called connection events. Within one connection event the master and slave of the link layer transmit data on one of the data channels. When one connection event is over, both, master and slave, switch to the next channel and start a new connection event (see [Figure 2.13](#)). However, the slave is not required to listen and react to every connection event. It can skip as much connection events as defined by the slave latency setting. This means, when the slave latency is set to zero, the slave has to listen and react to every connection event. The supervision timeout setting is the indicator for a connection loss. When either the master or the slave do not recognize a packet for the time specified in the supervision timeout, it is considered as a loss of connection. [[37](#), pp. 158-159]

2.3 Presence Detection

The main task of wireless networks is to transfer data from one entity to another without having the requirement of a physical connection. However, another application, especially for wireless technologies with a smaller transmission range, like Bluetooth, BLE, and WiFi, is presence detection of devices.

A network consists of several nodes, that communicate with each other, either one directional or bidirectional. Presence detection requires only a one directional communication, meaning one node sends information to another. From now on, a node emitting a signal is called Sending Node (SN), whereas a node receiving these signals is called a Receiving Node (RN). When a RN receives a signal from a SN, it can be concluded, that the SN and the RN are within a certain range. This range is defined by the maximum transmission range of the SN's wireless technology. This maximum transmission range

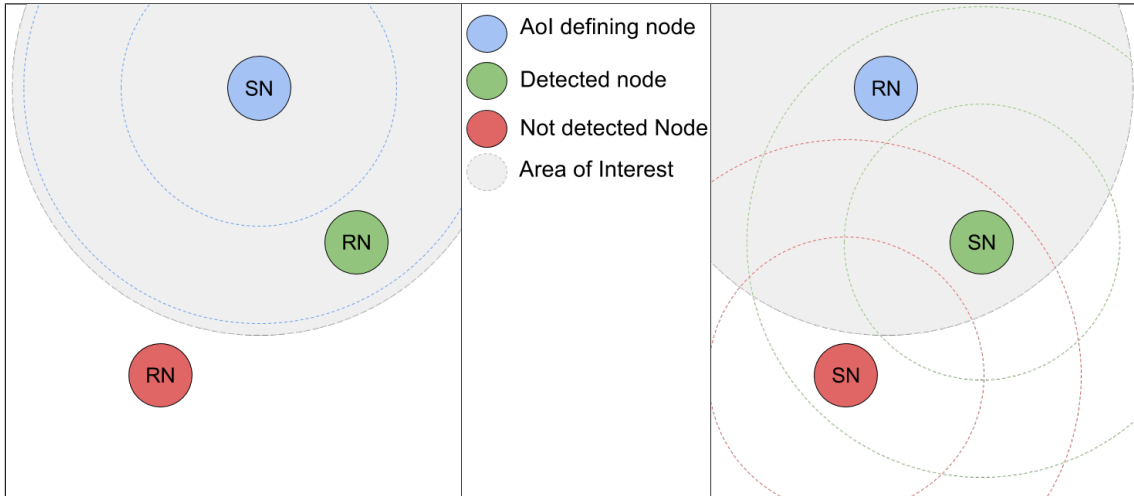


Figure 2.14: Wireless Presence Detection

defines the Area of Interest (AoI), in which nodes are able to be detected. It can either be an explicit or an implicit AoI as illustrated on the left and right in [Figure 2.14](#), respectively. Explicit means that the SN defines it with its transmission range, whereas implicit means, that a RN defines the AoI. This AoI can be narrowed down by thresholding the signal strength of the received signal.

Some use cases require location context that is linked to nodes. From now on a node of this kind is called Location Context Node (LCN). LCNs always define the AoI, because they can add location context to it. This context can be created in different ways. One way for adding location context to a node is, when a node is able to retrieve its GPS coordinates. Another would be deploying nodes at static positions. For instance, a node placed next to a playground adds the context ‘playground’ to it. A Detectable Node (DN), on the other hand, is a node, which changes its position and does not have any location information attached to it. However, when a DN can sense or can be sensed by a LCN, the location context of the LCN is valid for the DN as well. This is visualized in [Figure 2.15](#).

The categorizations into RN/SN and LCN/DN have no relationship to each other, meaning a RN can be a LCN or a DN as well as a SN can be a LCN or a DN. These two categorizations are defined to clarify the following explanations.

2.3.1 Related Work

The knowledge which is generated by presence detection with wireless technologies is useful in many use cases and is therefore focused by many researchers. Lodha et al. [44] present a way to use BLE for a student attendance management system. It tracks the attendance of students in lectures at a university. BLE transmitters are embedded into student cards and are used as SNs in the presence detection system. Each BLE chip has a unique id associated, which is linked to the student. A smart phone application, which shall be installed on the lecturer’s smart phone, is able to scan for the chips inside the student cards and therefore acts as RN. When the smart phone receives a signal of a BLE chip, it parses its unique id and sends this id and the information about the current lecture to

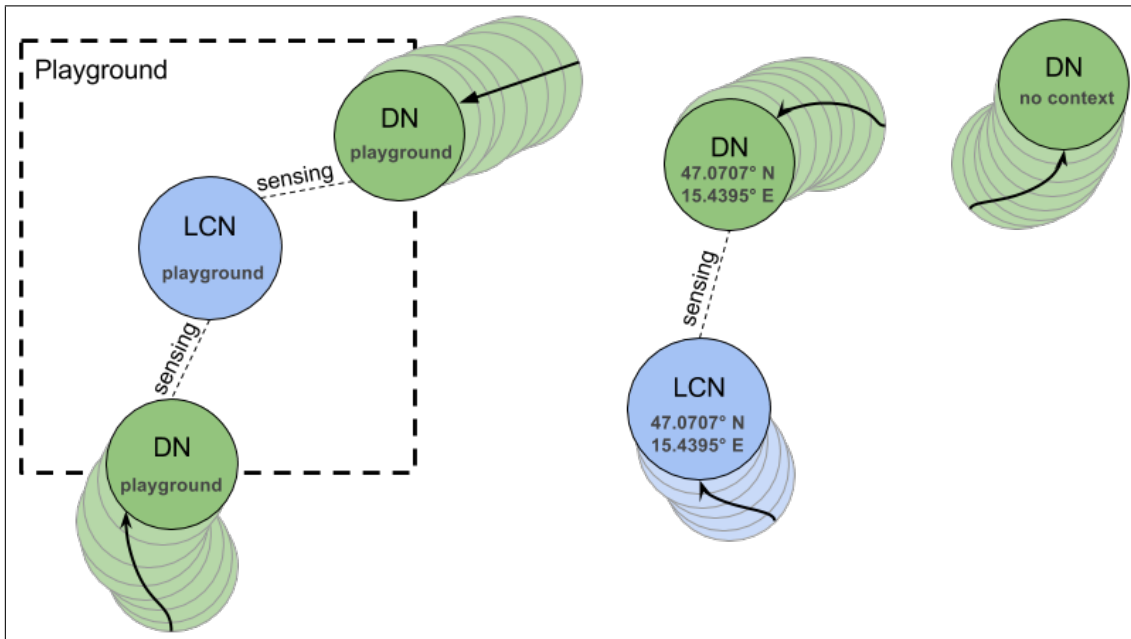


Figure 2.15: Location Context Nodes and Detectable Nodes of Wireless Network

an online system. The online system's database contains the enrolment information for each student. After recognizing a student, that is enrolled for that particular lecture, an attendance time stamp is written to the database. The smart phone can be seen as LCN that defines the AoI. Students have to enter this AoI with their student cards (DNs) for being recognized.

The work of Vinay and Savitha [69] proposes a system that can determine unexpected locations of pupils by tracking their presence in the class room as well as in the school bus. It is build for concerned parents who want to know where their children are. Each pupil gets an id card with an embedded BLE chip, which takes over the roles of a SN and a DN. The unique Bluetooth addresses of the chips are used for identifying the related pupil. There are three statically installed BLE readers per class room, which are connected to a server module. The school bus is equipped with an BLE reader, a Global System for Mobile Communications and a GPS to be able to provide the required location information. These setting enables tracking of pupils in school and on their way to school. The BLE readers within the class room and the one from the bus take over the role of RNs and LCNs. Therefore, they are defining the AoI. When pupils do not attend in class or do not enter the bus, when they are expected to do, the parents get informed via a smart phone application.

BLE beacons are BLE sender, which are places statically at a position and continuously transmit an advertisement. Barapatre et al. [9] present a solution for attendance tracking, which utilizes BLE beacons and the students smart phones for generating attendance information. The beacons are placed inside the lecture halls, and an Android application has to be installed on the students smart phone. This application is scanning for the beacons and sends the scanned information to a service in the internet. The service can then be used to look up a students lecture attendance. A similar system was proposed by

Deugo [30], which also utilizes BLE beacons and an Android application for attendance detection. In both proposals the BLE beacons become the SNs and LCNs, which define the AoI. Additionally, the smart phones take over the roles of a RNs and a DNs.

The presented system of Gavade et al. [35] uses Bluetooth classic to track the attendance of employees in a company. Bluetooth access points are placed all over the company building and scan for nearby Bluetooth devices. These access points define the AoI and take over the roles of the RNs and LCNs. The smart phones of the employees are acting as SNs and DNs. The identification of an employee is based on the Bluetooth address of their smart phone, which has to be registered in the companies infrastructure. Due to the continuous scanning of the Bluetooth access points, the smart phone get scanned eventually and its Bluetooth Address is then forwarded to a server. The server has access to a database where the relation between the employees and the Bluetooth addresses is stored.

An approach, for sensing crowds on events, using BLE, is presented by Basalamah [11]. Handing out 600 BLE tags and ten mobile devices to people, enabled tracking of crowd movements within an event area. The BLE tags were continuously transmitting a signal which could be identified by the mobile devices. Therefore, the BLE tags represent the DNs, by acting as SNs. The mobile devices on the other hand are scanning for the BLE tags, meaning, they were acting as RNs. When an advertisement from a BLE tag was received by a mobile device, the current coordinates of the mobile device, which were retrieved with an integrated GPS module, and a current timestamp were send to a server. Therefore, the mobile devices took over the role of a LCNs. The server stored all the received data in a database. With this approach, it is possible to make statements about the occupancy of different areas within the event as well as tracking streams of movements all over the event area.

All presented approaches so far, have the aim to detect the presence of wireless devices and assign a location context to them. There are only two different outcomes of a presence detection. A device is either present or not. LCNs are used for deriving the location context of DNs. These LCNs have either access to GPS or are placed statically at a known position. Each SN has an identifier, that is transmitted continuously. When a RN receives signals of a SN, it forwards the identifier to a server, which then processes or stores the information. Additionally, they use the LCN to either define the AoI explicitly or implicitly. An explicit AoI definition, where DNs are the RNs and LCNs are the SNs, is used from Barapatre et al. [9] and Deugo [30]. This explicit AoI definition is illustrated in [Figure 2.16b](#). An implicit AoI, where the LCNs are the RNs and the DNs are the SNs, is done by Lodha et al. [44], Vinay and Savitha [69], Gavade et al. [35], and Basalamah [11]. This implicit definition is visualized in [Figure 2.16a](#). Furthermore, they have an additional data transfer channel available on the RNs, which is used for redirecting the gathered information. Vinay and Savitha [69] use WiFi for storing the data on a server and the system of Gavade et al. [35] uses laptops, which either have a wired or a WiFi connection to the server, for storing the data. In cases where RN are not at a fixed location, as in the systems of Lodha et al. [44], Vinay and Savitha [69], Gavade et al. [35], and Basalamah [11], mobile devices are used for redirecting the gathered information. Mobile Devices, like smart phones, usually support many different wireless network technologies and are often used in a presence detection system.

Adding more LCNs to a network, extends the area, which can be searched for the

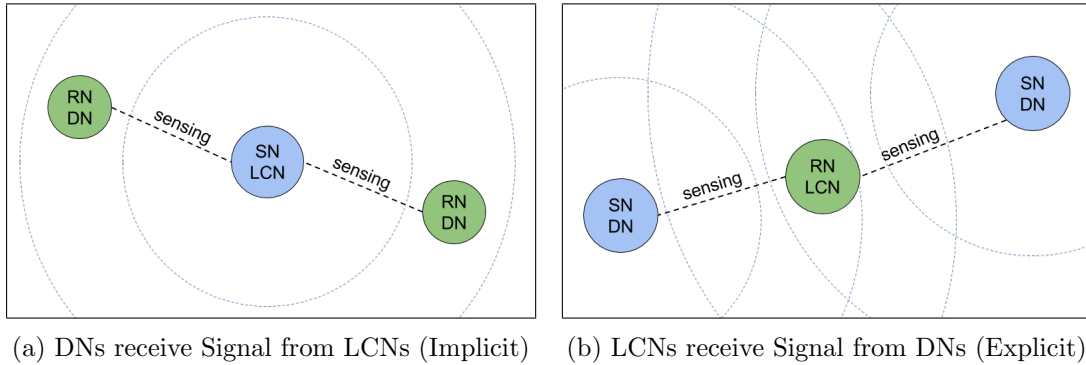


Figure 2.16: Implicit and Explicit AoI Definition

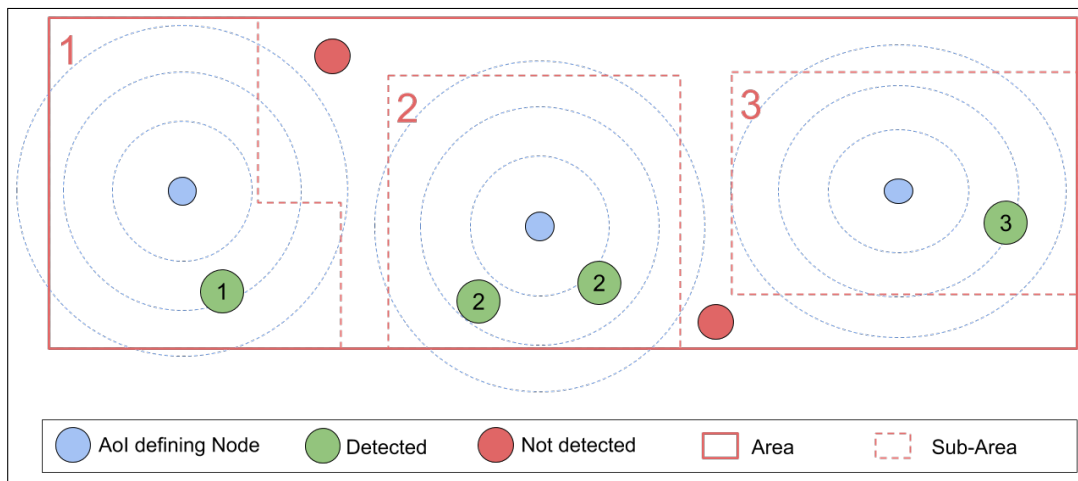


Figure 2.17: Sub-Area Detection

presence of DNs, as done by Basalamah [11]. However, using more than one LCN, enables a fragmentation of the AoI into sub-areas. The requirement of assigning nodes to such sub-areas, makes the whole process more challenging, but enables a system to generate additional knowledge, like the room, where a device is currently located. This is illustrated in Figure 2.17.

This knowledge of sub-area assignment, can be used to implement an occupancy detection systems, for improving the energy consumption of heating, ventilation, and air conditioning in smart buildings, by regulating the different settings of them according to the detected occupancies. Sentinel, a proposal from Balaji et al. [8], utilizes the existing WiFi network infrastructure of a building for deriving information about the room occupancy. The WiFi access points are the LCNs and the RNs in that system, whereas the smart phones of the employees are the DNs and the SNs. Additionally, they assign rooms of the building either to so called personal space or shared space. Sentinel can only make statements about rooms that are personal space like offices, because these rooms are directly linkable to people, which are usually located in that rooms. An example would be personal offices. Shared space, like the kitchens, can not be linked to particular people. The WiFi access points are aware of the personal spaces within their transmission area.

When a smart phone of a user connects to an WiFi access point that can reach the personal space of the user, this personal space is marked as occupied. When disconnecting the occupation is removed.

BLE based occupancy systems, which make use of the beacon technology, are presented from Conte et al. [26] and Corna et al. [27]. They conclude the occupancy of a room, by assigning people the rooms. Both systems create a wireless infrastructure by placing BLE beacons all over the building. This means, the beacons represent the LCNs and the SNs within the systems. The RNs, which scan for the beacons, are iPhones and Android phones in the researches of Conte et al. [26] and Corna et al. [27] respectively. The smart phones parse the unique identifiers of the beacons' signals and therefore act as DNs. Together with the current time, the unique identifier is forwarded to a server for further processing. Both systems utilize a classification algorithm to assign the smart phones to rooms. This is why both approaches require the execution of a preceding training and data collection.

Another approach, which uses a different classification algorithm for assigning smart phones to rooms of a building, is from Alhamoud et al. [3]. It uses the same setup as those from Conte et al. [26] and Corna et al. [27], with the difference of utilizing Bluetooth classic instead of BLE. They place Bluetooth beacons, acting as LCNs and SNs, in different rooms and perform periodically scans via smart phones. The smart phones, representing the DNs and RNs, extract the identifier of the beacons and forward them plus a timestamp, via a WiFi connection, to a server. The collected data is then analysed and the classification, which shall deduce the room assignment for the smart phone, is performed.

A system for room assignment without a classification algorithm is presented by Yang, Li and Pahlavan [79]. This system utilizes BLE beacons, acting as LCNs and SNs, for detecting smart phones and assigning rooms to them. The smart phones are the DNs and RNs, which forward the strength of the received signals to a server. The beacons are placed next to the entrance doors of the rooms. This leads to a peak in the signal strength, when entering a room. Two different settings have been tested. The first uses two beacons, where one is placed inside and the other outside of the room, but both of them next to the entrance door. A BLE device entering a room, creates a peak in the received signal strengths of both beacons. However, one of them is slightly higher due to the direct line of sight, which is existing only for one beacon due to the placement. Therefore, it is possible to conclude, whether a person has entered or left the room. The second setting uses one beacon, placed inside of the room, next to the entrance door. This works only with the prerequisite that the person closes the door after entering or leaving the room. This leads to a measurable difference in the signal strength when leaving or entering. The recognition, if a smart phone is inside or outside of a room, is done by using a threshold. When an enter or leave event is detected, the server updates the current room assignment of smart phone.

A large scale system to generate education related information about the students attendance at university lectures, by observing WiFi data, is presented by Zhou et al. [80]. The campus that is focused by the experiment, provides over 2700 WiFi access points, which were used for collecting the required information of nearby devices. These access points represent the LCNs and RNs. By mapping the hardware addresses of devices, like laptops, to over 2000 campus accounts of volunteers, information linked to people could be derived. These devices represent the DNs and SNs. They use the approach of Zhou et al. [81], to enable the creation of mobility traces for students. Comparing these traces with

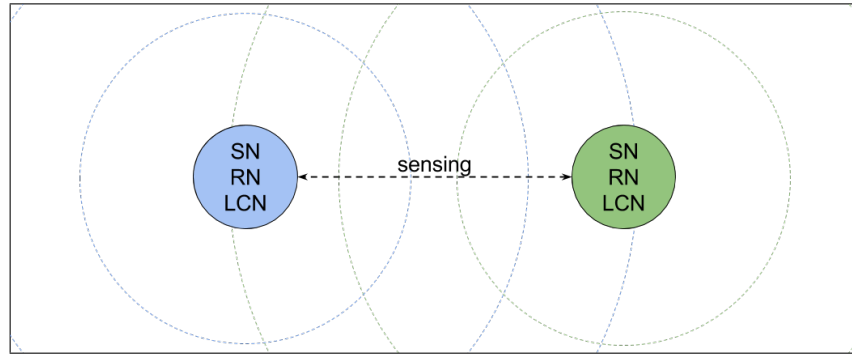


Figure 2.18: Bi Directional Presence Detection

Received Signal Strength Indication (RSSI) fingerprints¹ of the lecture rooms, enables deducing knowledge about the students attendance in lectures.

Another application of presence detection is social science. The fact that many people are in possession of a smart phone, is used by researchers to transform smart phones into data gathering devices, which are attached to people. The idea is, that smart phones scan for other smart phones via a wireless technology. This means, every node in this type of network takes over the role off a RN, a SN, and a LCN. This is illustrated in Figure 2.18

Miluzzo et al. [47] are presenting an approach, using smart phones, to gather information about social networks. They use a wide spread spectrum of data, that is collectable by smart phones, like acceleration, surrounding audio or location data via GPS. Additionally, they are generating proximity information by scanning for surrounding Bluetooth classic devices. The source of identification of the smart phones is the Bluetooth address. All gathered information is then sent to a server, where the data is processed. Eagle, Pentland and Lazer [34] created a similar approach for generating information about social networks. They utilizes cellular tower transitions of smart phones for generating big scale location information and Bluetooth classic scans for small scale proximity detection of people. A more recent paper, using proximity detection for mapping social networks, is from Boonstra, Larsen and Christensen [24]. Their research utilizes Android phones and iPhones, by providing an application for each of these platforms. They perform periodic Bluetooth scans for generating proximity information, as well. However, they only rely on information gathered by these Bluetooth scans for their research.

Townsend et al. [65], examined, whether smart phones can use BLE for detecting the proximity of other smart phones, by providing applications for Android and iOS. These applications continuously transmit and scan for BLE advertisements. They tested smart phones, two iOS devices and two Android devices, for their ability to detect other smart phones only by using BLE. They found, that it depends on the type of the phone, meaning if it is an iPhone or an Android phone, as well as on the phones current state. When participating smart phones are in an unlocked state, all smart phones, which were tested, were able to detect each other. The only combination of two smart phones, that were not able to receive a signal from each other, were two iOS smart phones, when in a locked state. This showed that iOS limits the BLE capabilities when the devices are locked.

¹A RSSI fingerprint is an identifiable combination of RSSI values of different access points used for localization [78]

2.3.2 Comparison of Presence Detection Systems

Comparing the presented systems of 2.3.1, provides a good overview of important aspects of presence detection. There are a lot of things, which have to be considered, when developing a system for a use case, that requires any kind of presence detection. A compact comparison, of the presented works, can be found in Table 2.2. The following paragraphs define the properties of the comparison and explain the values, which are filled into the table. When no information regarding one of the properties has been found in the presented works, the table is filled with N/A.

High Level Objectives: A categorization of the related works into three different high level objectives, can be done.

1. **Geographical Presence:** The first is detecting a device in an specific geographical area. The outcome of the systems is a binary classification, to either detected or not detected.
2. **Sub-area Presence:** This means, splitting up a bigger area into sub-areas and assigning devices, which are present within the area, to this sub-areas.
3. **Bi-directional Presence:** The last category is the detection of presence of another device, independent of the geographical context. This means, the relevant information is the proximity of two or more devices.

Gateway to another Channel: Each presented system, uses a certain device role as gateway to another channel, for forwarding the gathered information. When the used wireless technology for the presence detection is Wifi or Bluetooth classic, all participating devices are most likely to have the required computational power, for acting as gateway on their own. When it comes to BLE, most of the time not all of the participating devices are able to act as gateway, due to low computational power of many BLE devices. The different gateway systems are illustrated in Figure 2.19. A categorization, of the presented researches, into one of the following categories can be done:

1. **DN is the gateway:** The DN, which derives the location information from the LCN, forwards the data as illustrated in Figure 2.19a
2. **LCN is the gateway:** The node that provides the location information, forwards the gathered information to the server as well. This is illustrated in Figure 2.19b and Figure 2.19c

Up-to-dateness: The information which is collected for the presence detection is not in all presented approaches real time conform. It depends on the use case, whether up-to-dateness is important or not. A noticeable fact is that most researchers, using BLE beacons in combination with an iPhone, note that the iPhones do not detect beacon advertisings in a sufficient frequency, because the iPhones switch into a sleep mode. Some of the researchers do not make any statement about their up-to-dateness at all. Definitions of the assignable values for up-to-dateness are as follows:

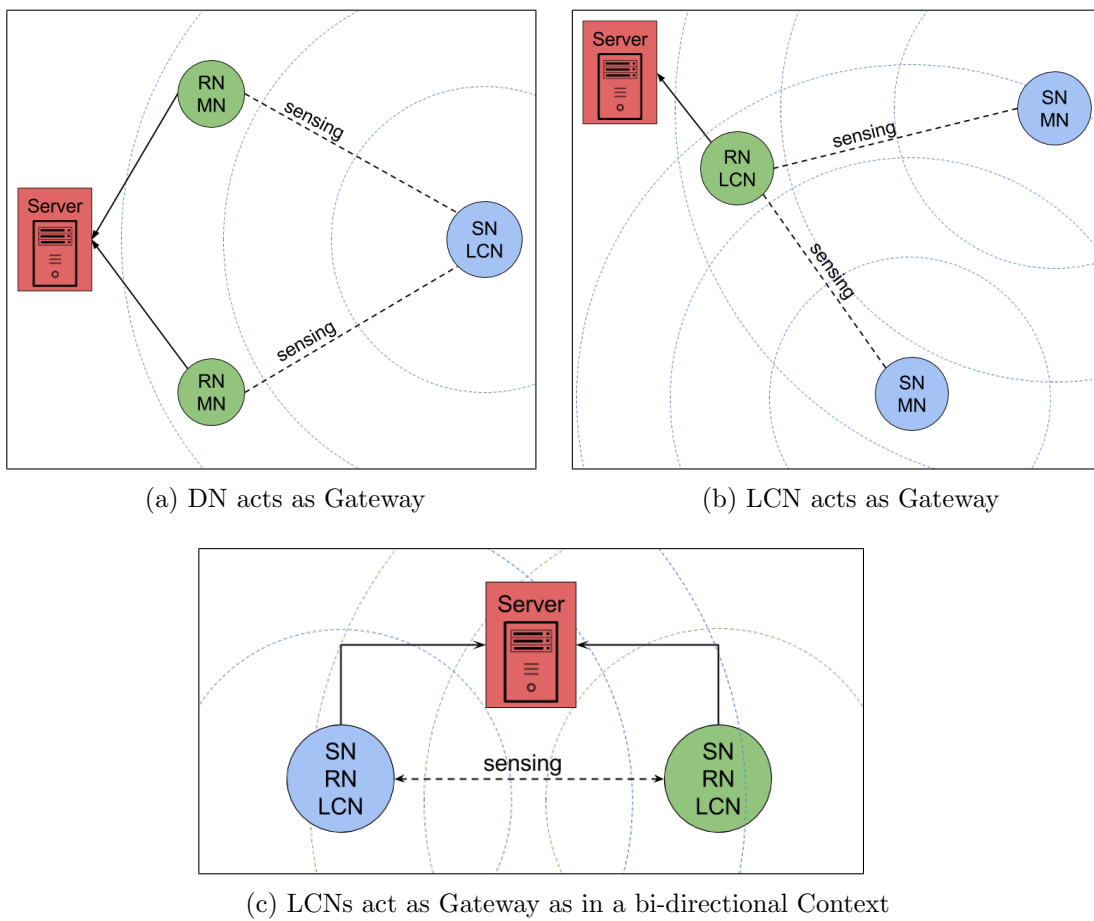


Figure 2.19: Different Nodes acting as Gateway in a Presence Detection System

1. **High:** Everything within a minute is considered as high frequent updates.
2. **Low:** The data is captured at a low rate, meaning there is a delay of several minutes between each data acquisition.
3. **Event based:** The time, when data is captured, is based on events like detecting the presence or the absence.

Wireless Technologies: The presented approaches use different wireless technologies to implement their systems. Most of them use *BLE*, however *Bluetooth classic* and *Wifi* are used as well. Some of the researchers use or extend already available infrastructures, whereas others build their own infrastructure.

Hardware: Considering real world environments, a Wifi infrastructure is usually already in place, which reduces the workload to implement a new system. BLE and Bluetooth classic on the other hand, often require the installation of additional hardware within the environment. WiFi and Bluetooth classic require all devices to have a solid power source like hard wired power supplies or a mobile device with a sufficient battery. BLE on the other hand, is build to be able to work for a long time, with a very small source of power, like a coin cell. This enables the provision of BLE tags and beacons without a lot of effort. Mobile devices tend to be a good hardware for presence detection. Due to the wide spreading of mobile devices and the variety of supported wireless networks, they are not only used for the presence detection itself, but also for acting as a gateway for forwarding the gathered information to another channel. The table indicates whether an additional hardware is required for the RNs or the SNs. Mobile devices are not considered as additional hardware.

Authentication Level: All of the presented approaches have some kind of identification of the devices, which are required to be detected. However, this identification is mostly done by using the unique device address, which is linked to an identity. Especially when it comes to BLE this is not a good approach for identifying an entity, because BLE provides a privacy feature which enables randomizing its retrievable Bluetooth address, as explained in the *Specification of the Bluetooth System 4.1*. [61, Vol. 3, pp. 381-386]. Additionally, adversaries could spoof² the address of a victims device and therefore impersonate them. None of the presented systems performs an adequate cryptographic authentication. However, it is important to note, that an secure authentication, during the presence detection process, is not the focus of any of the presented works. A categorization, which is used in [Table 2.2](#), is as follows:

1. **None:** The system can detect devices that are part of the system, however no distinction between them is possible.
2. **Identification:** The system can uniquely identify all corresponding devices, based on either their hardware address or an defined unique Identifier (ID).

²When an entity is masquerading another, by faking data, it is called a spoofing attack. [77]

3. **Authentication:** A proper authentication protocol is performed for identifying and authenticating the devices.

2.4 Cryptographic Authentication

Authenticating a user can be achieved by utilizing cryptographic operations and algorithms. These cryptographic algorithms are standardised and verified by organisations like the National Institute of Standards and Technology³, to prevent the usage of faulty and vulnerable algorithms. The National Institute of Standards and Technology is part of the United States Department of Commerce. One of their objectives is to publish standards that define technology, like cryptography. To do so, they create guidelines and release algorithm recommendations. [49]

User authentication is usually done by making use of asymmetric cryptography. Asymmetric cryptography utilizes two different keys, a so called key pair, for encrypting and decrypting information. When encrypting information with the first key the second key is required to decrypt it and vice versa. These two keys are generally named private key and public key. The private key has to be kept a secret, whereas the public key may be publicly available. This enables confidential communication with an entity by encrypting the messages with the entities public key. This encrypted message can only be decrypted by the possessor of the corresponding private key. Examples for asymmetric key algorithms are Elliptic Curve Cryptography (ECC) and the Rivest-Shamir-Adleman cryptosystem (RSA). [68, p. 11, p. 195, p. 532]

A key pair from asymmetric cryptography can be used for signing a message and therefore ensuring its authenticity. This signing procedure creates a digital signature. When an entity signs a message with its private key, everybody who receives the message, the digital signature, and the entities public key, can verify that the message was send by the entity, possessing the corresponding private key. When a digital signature of a document has been created, the party, which has signed the document, can not declined, that the document has been signed with its private key. [68, p. 171]

A digital signature is usually created by using of a cryptographic hash function, which generates a so called hash value from a message. Such a hash function shall provide three properties. *Pre image resistance* means, when a certain hash value is known, it shall be very hard to find a message, which would generate this certain hash value when used as input of the hash function. The second property, *second pre image resistance*, implies that for a given message, it shall be very hard finding a second message, which produces the same hash value, when used as input for the hash function. *Collision resistance* is the third property and means, that it has to be hard finding two messages, which would produce the same hash value, when using both messages as input for the same hash function. [68, pp. 256-267]

A Public Key Infrastructure (PKI), makes use of digital signatures and asymmetric cryptography for creating a system, which enables issuing so called digital certificates for users. These digital certificates link the public key of an asymmetric key pair to an identity. Therefore, they contain the information, which is required to uniquely identify a

³National Institute of Standards and Technology Website <https://www.nist.gov/>

Research	High Level Objectives	Gateway	Up-to-dateness
Lodha et al. [44]	Geographical Presence	LCN	N/A
Vinay et al. [69]	Geographical Presence	LCN	Event based
Barapatre et al. [9]	Geographical Presence	DN	Event based
Deugo [30]	Geographical Presence	DN	Event based
Gavade et al. [35]	Geographical Presence	LCN	N/A
Basalamah [11]	Geographical Presence	LCN	Low
Balaji et al. [8]	Sub-area Presence	LCN	Low
Conte et al. [26]	Sub-area Presence	DN	Low
Corna et al. [27]	Sub-area Presence	DN	N/A
Alhamoud et al. [3]	Sub-area Presence	DN	High
Yang et al. [79]	Sub-area Presence	DN	N/A
Zhou et al. [80]	Sub-area Presence	LCN	N/A
Miluzzo et al. [47]	Bi-directional Presence	LCN	High
Eagle et al. [34]	Bi-directional Presence	LCN	Low
Boonstra et al. [24]	Bi-directional Presence	LCN	Low
Townsend et al. [65]	Bi-directional Presence	LCN	N/A

Research	Technology	Additional Hardware		Authentication Level
		RN	SN	
Lodha et al. [44]	BLE	no	yes	Identification
Vinay et al. [69]	BLE	yes	yes	Identification
Barapatre et al. [9]	BLE	yes	no	Identification
Deugo [30]	BLE	yes	no	Identification
Gavade et al. [35]	Bluetooth classic	yes	no	Identification
Basalamah [11]	BLE	no	yes	None
Balaji et al. [8]	WiFi	no	no	Identification
Conte et al. [26]	BLE	no	yes	Identification
Corna et al. [27]	BLE	no	yes	Identification
Alhamoud et al. [3]	Bluetooth classic	no	yes	Identification
Yang et al. [79]	BLE	no	yes	Identification
Zhou et al. [80]	WiFi	no	no	Identification
Miluzzo et al. [47]	Bluetooth classic	no	no	Identification
Eagle et al. [34]	Bluetooth classic	no	no	Identification
Boonstra et al. [24]	Bluetooth classic	no	no	Identification
Townsend et al. [65]	BLE	no	no	Identification

Table 2.2: Comparison of Related Work in Presence Detection

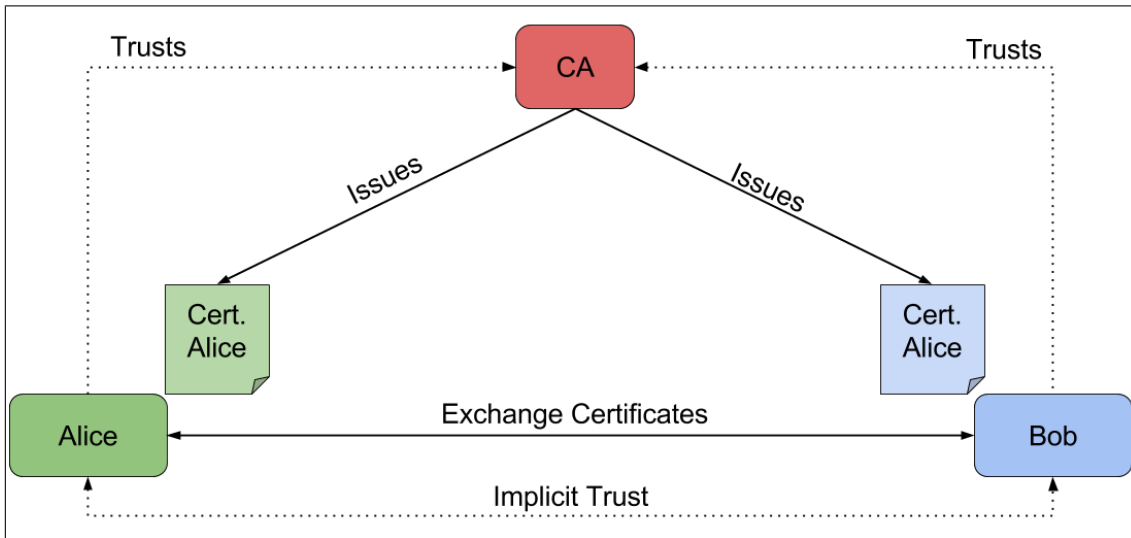


Figure 2.20: CA as trusted third Party

user, plus the public key of the user's key pair. To ensure that an issued digital certificate is valid, a Certificate Authority (CA) uses a so called root certificate to sign the issued digital certificates. This creates a trust relation between users, when using digital certificates, which are issued by the same CA. Therefore, when two entities are communicating with the help of their digital certificates, the CA acts as trusted third party and enables authenticated communication as illustrated in Figure 2.20. [68, p. 67, p. 488, pp. 628-637]

Such a PKI can be used for performing a key agreement. A key agreement is the process of creating a shared secret for two or more entities, by using an insecure data transfer channel, without exposing any sensible information. Algorithms like the Diffie-Hellman (DH) key exchange utilizes asymmetric cryptography for performing a key exchange. [68, p. 154, p. 325]

A key agreement is often performed, to enable symmetric cryptography, which is faster and can use smaller keys than asymmetric cryptography. Symmetric cryptography utilizes a symmetric key plus a symmetric key algorithm, to transfer information confidential from one entity to another. The symmetric key can be derived from a shared secret only known by a defined set of entities. The symmetric cryptographic algorithm can encrypt a plain text information and decrypt the encrypted information with the same symmetric key. Therefore, all entities, who know the shared secret and can derive the symmetric key, are able to participate in a secured communication. Examples of a symmetric key algorithm are the Advanced Encryption Standard and the Data Encryption Standard. [46, p. 32], [68, pp. 129-135, pp. 520-524, pp. 602-603], [76]

Symmetric cryptography, not only enables transferring information securely, but also can be used for authenticating messages with a so called Message Authentication Code (MAC). A MAC is generated by applying an algorithm that uses a secret key and a message as input. Considering a set of entities, which share a secret key, all members of that set can send messages plus the computed MAC to other members of the set. All other members can recompute the MAC from the received message and compare it to the received MAC. Doing so, they can verify that the message has been sent by someone, who knows the

shared secret and the message has not been tampered with. These properties enable building a challenge response protocol, by using a MAC. A challenge response protocol is executed, when one entity is asking for some kind of information, which can only be provided by a set of other entities. Therefore, by asking an entity for computing the MAC of a random number, is performing a challenge response protocol. An example for a MAC algorithm is the Cipher-based Message Authentication Code, which is recommended by the National Institute of Standards and Technology and uses the Advanced Encryption Standard to provide authenticity of a message. [33], [68, p. 73, p. 361]

Many of the previously mentioned algorithms rely on a source of randomness. This randomness is in software usually achieved by an pseudo random number generator, which is a deterministic algorithm for generating a sequence of numbers that have almost probabilities of random numbers. However, they are based on a so called seed, which defines the sequence of numbers during initialization. These pseudo random number generators are often used in software applications, because they are fast. [68, pp. 485-487]

All these cryptographic algorithms and systems are used for preventing adversaries getting access to sensible data. This is required, because there is a variety of known attacks which can be used for doing so. In a *Man-in-the-Middle (MITM)* attack, two parties think that they are communicating with each other, but in fact they do not. This takes often place when asymmetric key pairs are in use. Considering an example of Alice wanting to talk with Bob and both of them have an asymmetric key pair. They want to exchange their public keys and encrypt the information with the public key of the other party to ensure a secure link. However, the key exchange could be intercepted by an adversary named Eve. Eve has also an asymmetric key pair and intercepts the public keys of Alice and Bob. Instead of forwarding the public keys of Alice and Bob, Eve sends her own public key to the two entities. When she intercepts an encrypted message, she can decrypt it with her private key, read the plain message, encrypt it with the public key of actual destination and forward the newly encrypted message. *Eavesdropping* is done by a third party, which listens to an ongoing communication of two other parties. This third party is called an eavesdropper and is able to read the information without the other two parties knowing about it. Passive eavesdropping is only reading the information, whereas active eavesdropping is the attempt of exchanging the text before forwarding it. Another attack is called *replay attack*. When an adversary is performing a such an attack, they are recording communication sessions of two or more entities. Later in time, the adversary uses the recorded messages to replay the whole session, by sending the same messages again. This could be used to retrieve sensible data. All these attacks are illustrated in Figure 2.21. [68, p. 169, p. 368, p. 519]

COYERO access: It is a framework, enabling easy set up of a client-server environment, which supports authentication and authorization of people, using their personal mobile devices. Its environment consists of three different instances, namely client, kiosk, and server. The client and the kiosk are available as Software Development Kit (SDK) for Android and iOS. The server is provided as a hosted cloud service. [28]

The *COYERO access Cloud Service* provides a RESTful⁴ Application Programming

⁴Representational State Transfer (REST) can be used to provide a unified way of interoperability between web services. [75]

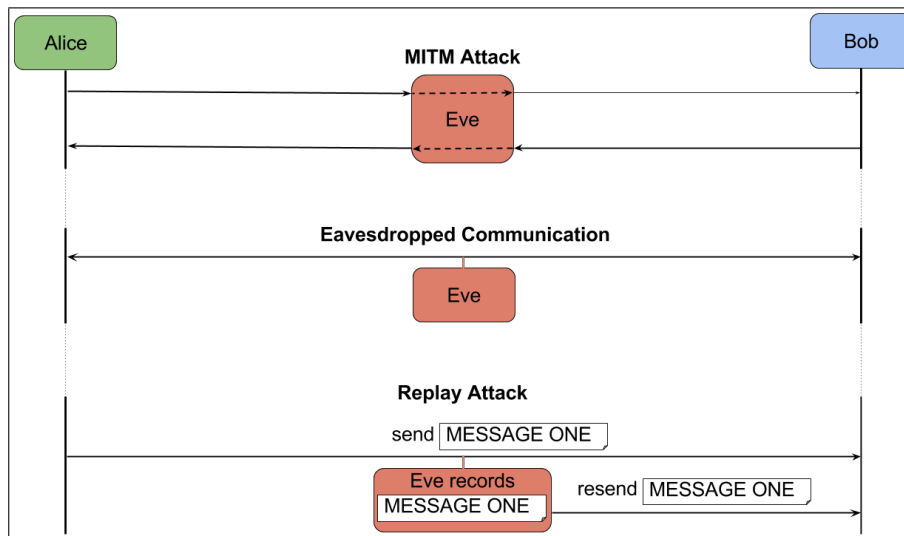


Figure 2.21: Attacks on a Communication Channel

Interface (API), which can be used for granting authentication rights to users and for creating authorizations for different tasks, like entering a garage or buying an article in a shop. Users are identified with a so called authentication token, which is signed by the COYERO access server. Therefore, the COYERO access server is acting as CA and the authentication token can be compared to a digital certificate. The authorization rights for users are issued as so called entitlement tokens. These entitlement tokens contain the purpose of the authorization, the linked authentication token id for identifying the user, and it is signed by the COYERO access server's private key. These tokens are then used for authenticating the user and verifying the user's authorizations. [28]

Mobile applications extended with the *COYERO access Client SDK*, transforms a user's personal mobile device into their authentication device, namely a COYERO access client. The SDK is able to request and store a unique authentication token, by performing a RESTful call to a COYERO access server. Once a valid authentication token has been received by the COYERO access client, it can be used to request entitlement tokens for getting authorization rights for specific goods or services. These tokens are stored on the user's mobile device and can later on be used without a connection to the COYERO access server. With the authentication token, COYERO access clients can authenticate themselves and redeem entitlement tokens at a COYERO access kiosk. This can be done, by using different physical channels. Based on the user's device, BLE, NFC, and QR-Code scanning are available. [28]

The *COYERO access Kiosk SDK* can transform a mobile device, which is physically placed in the area where the authentication or authorization operations are required, into a COYERO access kiosk, which serves as a validation instance for COYERO access clients. This means, a COYERO access kiosk can verify authentication tokens and entitlement tokens of COYERO access clients via BLE, NFC or QR-Code scanning. Additionally every COYERO access kiosk has its own authentication token, which can be verified by COYERO access clients to achieve a mutual authentication. [28]

The three instances, COYERO access server, COYERO access client, and COYERO

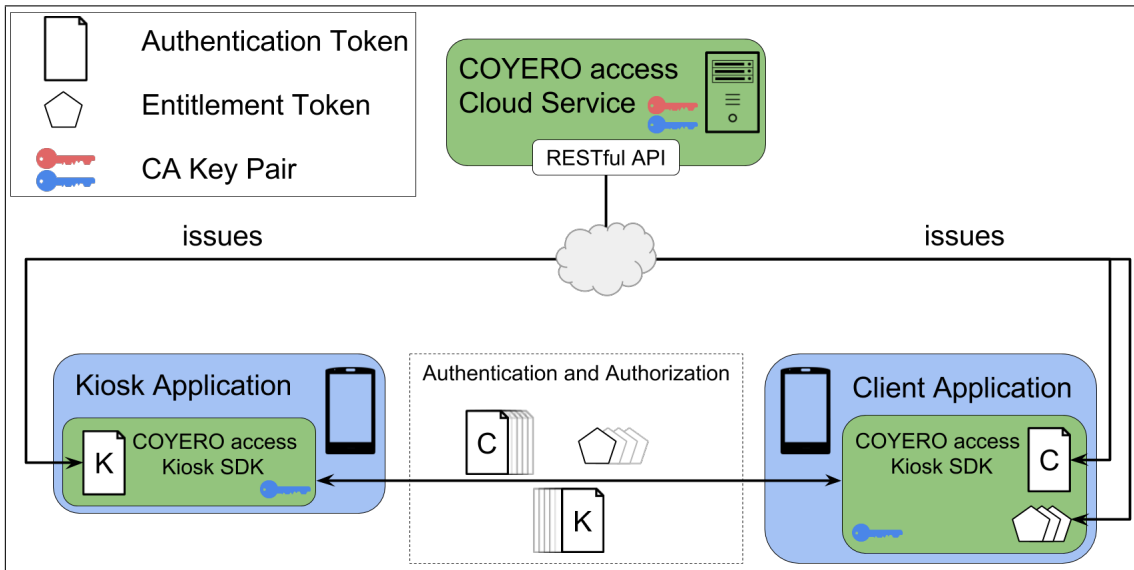


Figure 2.22: COYERO access environment, adapted from [28]

access kiosk, enable a COYERO access client to use already obtained authentication and entitlement tokens without having an active internet connection. This offline capability is enabled by installing the CA root certificate from the COYERO access server onto the mobile devices. The concept is illustrated in Figure 2.22. [28]

Chapter 3

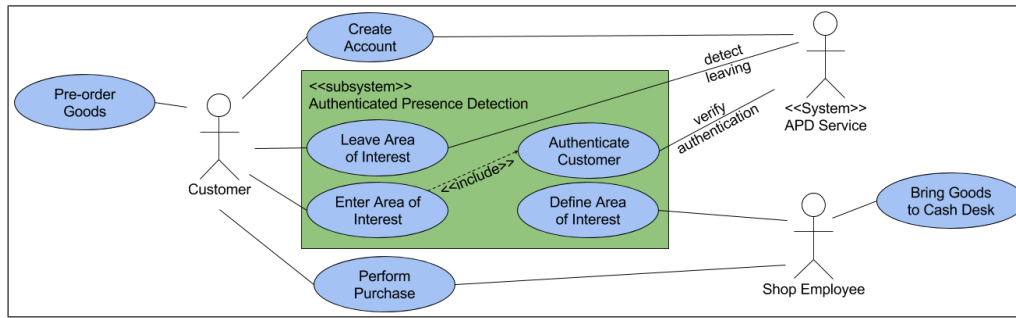
Design of Authenticated Presence Detection

This chapter presents the design of the newly created protocol for authenticated presence detection using BLE. It starts, by giving a high level overview of the design, where use cases, requirements and the different architectural views of the design are presented. Afterwards, the three main parts of the design, namely recognition, authentication, and Authenticated Presence Detection (APD) are then explained in detail.

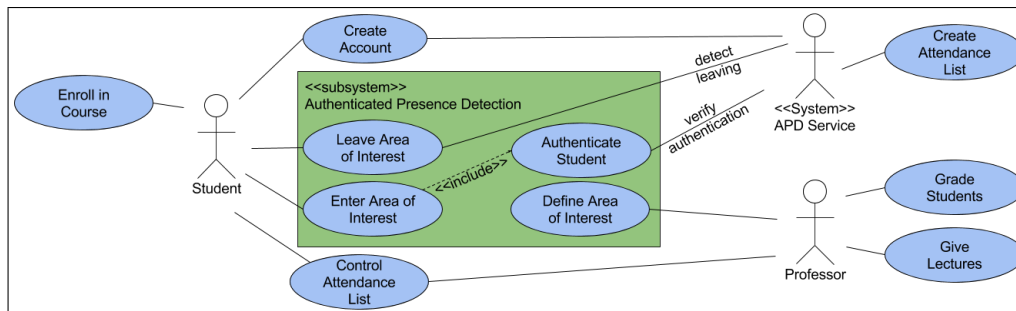
3.1 High Level Design

To get a first impression of the designs aim, two use cases are shown. The first is in the context of a salesman who wants to provide faster services for their customers. A customer has the possibility of creating an account and then can perform pre-orders with this account. The salesman's ambition is to have the pre-ordered good already available at the cash desk, as soon as the customer is in front of it. The shop's employees are able to define the shop as area of interest (AoI). When a customer enters this AoI, an authentication process is performed, which authenticates the customer. Now that the shop's employees know, that the customer is already in the shop, the pre-ordered good of the customer can be brought to the cash desk, so that the customer is able to obtain it immediately. When the customer leaves without purchasing their pre-ordered good, which can be detected as well, the goods can be removed from the cash desk. A second applications is an attendance system for students in lectures. Students shall have the possibility of creating an online account, which they can use for enrolling in courses and checking their attendance information of lectures. These attendance information can be retrieved by professors as well and be taken into account when it comes to grading of students. When a professor is giving a lecture, they are able to define the lecture room as AoI and can therefore detect students, which have entered or left the lecture room. A use case diagram for these applications can be seen in [Figure 3.1](#).

These use case diagrams have some use cases in common. First of all, some kind of user account is required, which is the basis of the identification of the user. Defining, entering and leaving the AoI are functionalities, which shall be provided for detecting the physical presence of the users within an specific location. Additionally, the user needs to proof



(a) Shop Pre-Order and Preparation



(b) Student Attendance System

Figure 3.1: Use Case Diagrams

who they are by authenticating themselves. Detecting the physical presence of users and authenticating them via BLE is covered by the presented design, whereas creating a user account which contains the users information is a prerequisite. The covered features are indicated by the subsystem in Figure 3.1. Furthermore, these use case diagrams are the basis for some identified requirements, which have to be fulfilled to achieve the subsystem's functionalities with BLE. These requirements are as follows:

- **False positives are the worst:** The worst case is when the subsystems claims, that a user is present, although it is not. These cases are called false positives and would lead to wrong information and actions. For the shop pre-order and prepare use case, it would lead to an unnecessary workload for the employees, because they would assume that a customer is present and they prepare their order, although they should not. False negatives would simply not improve the user experience, because the employees would start to prepare the order of the customers, when they were present at the cash desk. This would be the same behaviour as without the presence detection in place.

For the attendance system, false positives would improve the grading for students, which are wrongly detected as present. False negatives, on the other hand, would only change the required actions of the students from 'signing an attendance paper', to 'checking the attendance list and notifying the presence personally'.

- **Usage of well known authentication methods:** It is crucial when it comes to information technology security related tasks, that well known cryptographic al-

gorithms are used for authentication. Otherwise, adversaries could fake the presence of other users, which would lead to additional workload in the use case of [Figure 3.1a](#) and improved grading in the use case of [Figure 3.1b](#).

- **Minimal data transfer:** For presence detection and the authentication, minimal data shall be transferred. Minimal data transfer is very important when it comes to BLE, because BLE is not build for transferring a lot of data.
- **Up-to-date information of present devices:** All participating users within the AoI have to announce their presence as often as possible and additionally proof with every announcement that they are authenticated. This is important because a wireless presence detection system can only make statements about presence of users when it receives signals from them.
- **Ad-hoc presence detection:** The presence detection and authentication shall be performed without any required actions of the user. When the user enters the AoI, the authentication and presence detection shall start automatically.
- **Offline capability:** The whole process of authenticating and detecting the presence of devices has to be able without the devices, which are required to be detected, having access to the internet.

These requirements, lead to the high level architecture of the presented design, which is shown in [Figure 3.2](#). [Figure 3.2a](#) shows two entities, which are required to perform the authenticated presence detection. These two entities, namely the APD-kiosk and the APD-client, have to perform several actions in a certain order. These actions are recognition, authentication, and authenticated presence detection. This is shown in [Figure 3.2b](#). [Figure 3.2c](#) shows an exemplary process, which happens, when a user enters or leaves the AoI.

APD-kiosk: The APD-kiosk is the AoI defining entity. Within this AoI, other entities, namely the APD-clients, can be detected by either receiving their advertisements or exchanging messages with them. The APD-kiosk offers three different services via BLE, which can be used by APD-clients to perform the recognition, authentication and authenticated presence detection.

APD-client: APD-clients are the entities, which are recognized and authenticated after entering the AoI. The APD-client uses a state machine for performing the required procedures of recognition, authentication, and authenticated presence detection in the correct order.

Recognition: The recognition is the first contact between the APD-kiosk and the APD-client, or in other words, the APD-client entering the APD-kiosk's AoI. The APD-kiosk is transmitting a BLE advertisement, which indicates that it offers all the required functionalities to perform authentication and authenticated presence detection. When an APD-client scans such an advertisement, it has recognized the APD-kiosk and can start the authentication procedure. More information can be found in [3.2](#).

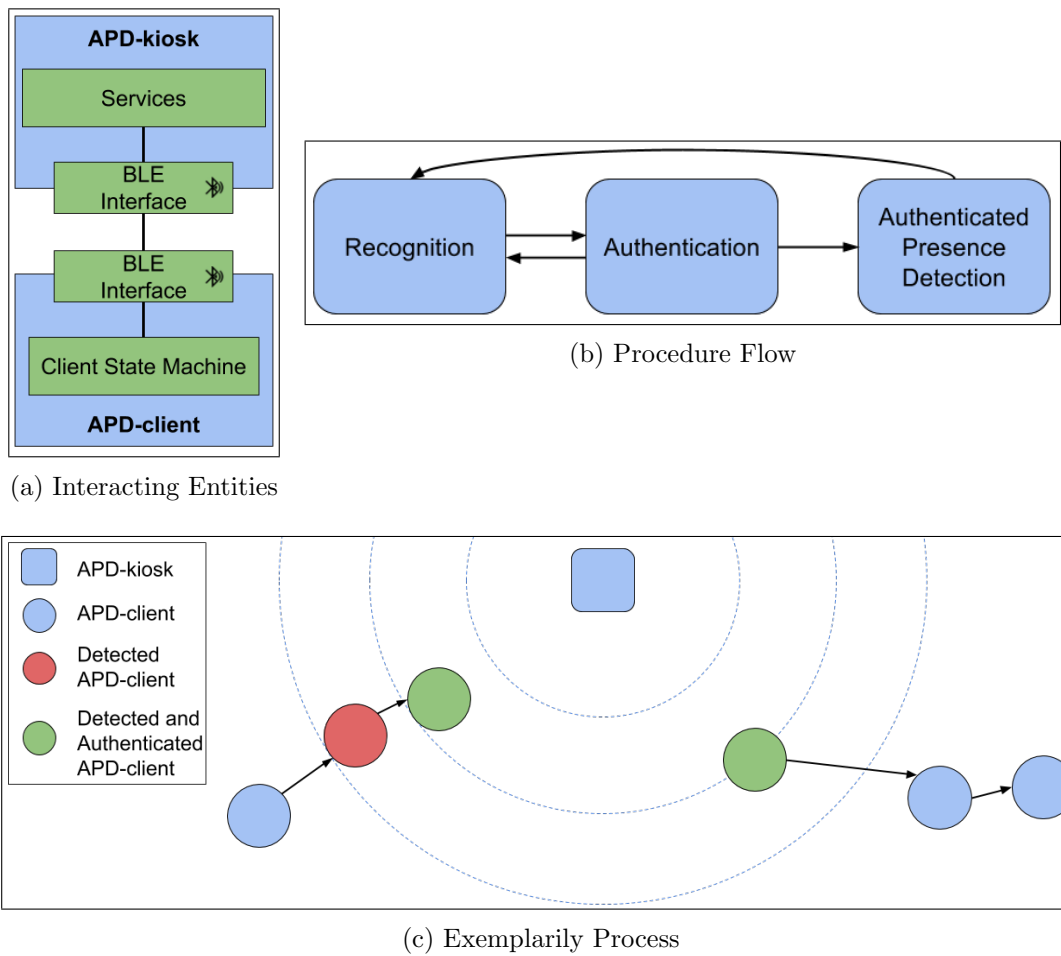


Figure 3.2: High Level Design

Authentication: After the APD-client has recognized the APD-kiosk and established a connection, both party perform an authentication protocol, which verifies the authenticity of the APD-client and the APD-kiosk. When the authenticity can not be verified, the ADP-client is disconnected and is not able to perform the authenticated presence detection. When the authenticity of the APD-client can be verified, an authenticated session is created. The authentication protocol is performed via a BLE GATT connection, utilizing a GATT Authentication Service (see 3.3.2). This service provides characteristics to write and read all the required data for performing the authentication protocol, which is presented in 3.3.1. The APD-client represents the GATT client and the APD-kiosk the GATT server. When an authenticated session is established, the BLE GATT connection is closed and the authenticated presence detection is triggered. The technical details for this process are carried out in 3.3.

Authenticated Presence Detection: The previous process created an authenticated session, which is then used to perform an easy and fast challenge response protocol for verifying the user’s authenticity. This challenge response protocol is simultaneously used for performing the presence detection. Therefore, the APD-kiosk starts acting as BLE broadcaster and advertises a challenge, which shall be solved by the APD-client. This challenge is periodically updated. The APD-client acts as BLE observer and scans for such challenge advertisements. When an APD-client receives a challenge, it solves the challenge and responds it by starting its own BLE advertisement. When the APD-kiosk’s challenge advertisement has been updated, the APD-client’s challenge response advertisement is updated as well. The APD-kiosk scans for challenge response advertisements and not only uses them for verifying the authenticity of APD-clients, but also generates the knowledge, that the APD-client is still present in the AoI. Details can be found in 3.4. When a APD-client leaves the AoI and is not detected by the APD-kiosk anymore, it switches back the recognition and again starts scanning for an APD-kiosk (see 3.4.4).

3.2 Recognition

The recognition is the first procedure, which is required to be performed. It is the most trivial one of the three. The APD-kiosk takes over the role of a BLE peripheral and uses a connectable advertisement which includes the UUID of the GATT Authentication Service, which can be seen in Table 3.2. This advertisement can be used from APD-clients, which act as a BLE peripheral, to establish a connection. The APD-client, then has to discover the available GATT service, which have to include a GATT Authentication Service, which is defined in 3.3.2. After the GATT Authentication Service has been successfully discovered, the recognition process is finished and the authentication protocol can be performed.

The signal strength of the APD-kiosk’s advertisement limits the defined AoI and therefore has to be the same as the signal strength from the challenge advertisement of the authenticated presence detection, which is explained in 3.4.3.

Notation	Meaning
Alice, Bob	Entities, which want to perform mutual authentication and create an authenticated session
Cert_K _{pub} (<entity>)	Public key of the certificate from <entity> which is signed by a CA
Cert_K _{pri} (<entity>)	Corresponding private key to Cert_K _{pub} (<entity>)
Cert_K _{pri,pub} (<entity>)	Key pair of the certificate from <entity>
Temp_K _{pub} (<entity>)	Temporary public key from <entity>
Temp_K _{pri} (<entity>)	Corresponding private key to Temp_K _{pub} (<entity>)
Temp_K _{pri,pub} (<entity>)	Temporary key pair of <entity>
hash(<data>[, <data>, ...])	Cryptographic hash function that hashes <data>. Optionally the input can be a concatenation of several input values.
Ses_K(<entity1>,<entity2>)	Symmetric key only known by <entity1> and <entity2>
Ses_ID(<entity1>,<entity2>)	Unique identifier which is linked to <entity1> and <entity2>
sign(<pri_key>,<data>)	Signature for <data>, generated by a private key (<pri_key>) of an asymmetric key pair
p_rng()	Pseudo random number generator

Table 3.1: Notation for the Authentication Protocol

3.3 Authentication

When an APD-client has recognized an APD-kiosk, which offers an GATT Authentication Service, it connects to it, discovers the service, and then starts the authentication. This section contains a detailed explanation of the used cryptographic authentication protocol in 3.3.1 and describes the BLE interface, which is used for performing the authentication in 3.3.2. Table 3.1 contains the notation, which has been used for the explanations.

3.3.1 Authentication Protocol

The precondition for implementing this authentication protocol, is an already in place PKI, where a CA has issued certificates for each participant, plus all of the participants are trusting the CA. Considering two parties, from now on called Alice and Bob, have certificates, which are issued by the CA of the PKI. Both are in possession of the corresponding private key. Alice acts as APD-kiosk which wants Bob, who is acting as APD-client, to authenticate himself.

Both, Alice and Bob, have to exchange their digital certificates. In other words, they exchange their Cert_K_{pub} with each other. The actual validation of the authenticity is performed, by validating the certificate of the other party with the stored and trusted CA certificate. After this, Alice and Bob create Temp_K_{pri,pub}(Alice) and Temp_K_{pri,pub}(Bob), respectively, using ECC. The decision for using ECC is based on several things. ECC requires a smaller key size than RSA for achieving the same security properties. This has been shown in the work of Barker [10, p. 53], which indicates the key sizes of algorithms,

Property	Definition
UUID	5b3e4f9e-d685-4458-a3c9-2e2baf0a8513
Characteristic	Client Authentication Data (defined in Table 3.3)
Characteristic	Kiosk Authentication Data (defined in Table 3.4)
Characteristic	Session ID (defined in Table 3.5)

Table 3.2: Properties of the GATT Authentication Service

that provide the same cryptographic strength. To have the same cryptographic strength as ECC, when using a 256 bit key, a RSA key with 3072 bit would be required. This means when using ECC, a lot less data transfer is required. Additionally, Gura et al. [38] found, that ECC outperforms RSA when it comes to speed as well. Therefore, ECC is faster than RSA and requires fewer data to be transferred, which makes ECC the better fit for the requirements defined in 3.1. ECC is therefore favoured over RSA. $\text{Temp_K}_{\text{pri, pub}}(\text{Alice})$ and $\text{Temp_K}_{\text{pri, pub}}(\text{Bob})$ shall then be used in a DH key exchange for generating a shared secret. To do so, both parties are required to have the $\text{Temp_K}_{\text{pub}}$ of the other party. Before exchanging their $\text{Temp_K}_{\text{pub}}$, both parties sign them with their $\text{Cert_K}_{\text{pri}}$. This enables the other party to verify the authenticity of the $\text{Temp_K}_{\text{pub}}$ and prevents MITM attacks. The purpose of using a separate $\text{Temp_K}_{\text{pri, pub}}$, is decoupling the protocol from the already in place PKI.

Alice is now in possession of $\text{Temp_K}_{\text{pri, pub}}(\text{Alice})$ and $\text{Temp_K}_{\text{pub}}(\text{Bob})$, whereas Bob has now access to $\text{Temp_K}_{\text{pri, pub}}(\text{Bob})$ and $\text{Temp_K}_{\text{pub}}(\text{Alice})$. Therefore, they can perform the DH key exchange. The generated shared secret plus both of the $\text{Temp_K}_{\text{pub}}$ are then hashed for deriving a symmetric key, which represents $\text{Ses_K}(\text{Alice, Bob})$. The hashing of this values is recommended, due to the work of Langley, Hamburg and Turner [43, p. 15], because different public keys can create the same shared secret, when using DH with ECC. The last step of the authentication protocol is Alice generating a unique session identifier, namely $\text{Ses_ID}(\text{Alice, Bob})$, with a `p_rng()` and sending it to Bob. The $\text{Ses_K}(\text{Alice, Bob})$ and $\text{Ses_ID}(\text{Alice, Bob})$ form the authenticated session, which is later on used for performing fast and high frequent authenticated presence updates. The protocol is illustrated in [Figure 3.3](#).

3.3.2 BLE Interface

The GATT Authentication Service is a custom BLE GATT service offered by an GATT server running on the APD-kiosk. APD-clients can connect to this GATT server as GATT client and use the offered GATT Authentication Service for performing the authentication protocol described in 3.3.1. The authentication protocol requires the possibility of exchanging data between both participating parties, which is then used for performing and verifying the authentication. The GATT Authentication Service defines characteristics which can be used to write data to and read data from the GATT server. The GATT Authentication Service definition can be seen in [Table 3.2](#).

Client Authentication Data: APD-clients use this characteristic for transferring their $\text{Cert_K}_{\text{pub}}$, their $\text{Temp_K}_{\text{pub}}$, and a signature, which is created by signing their $\text{Temp_K}_{\text{pub}}$ with their $\text{Cert_K}_{\text{pri}}$, to the APD-kiosk. For this, the data is packed into an authentication

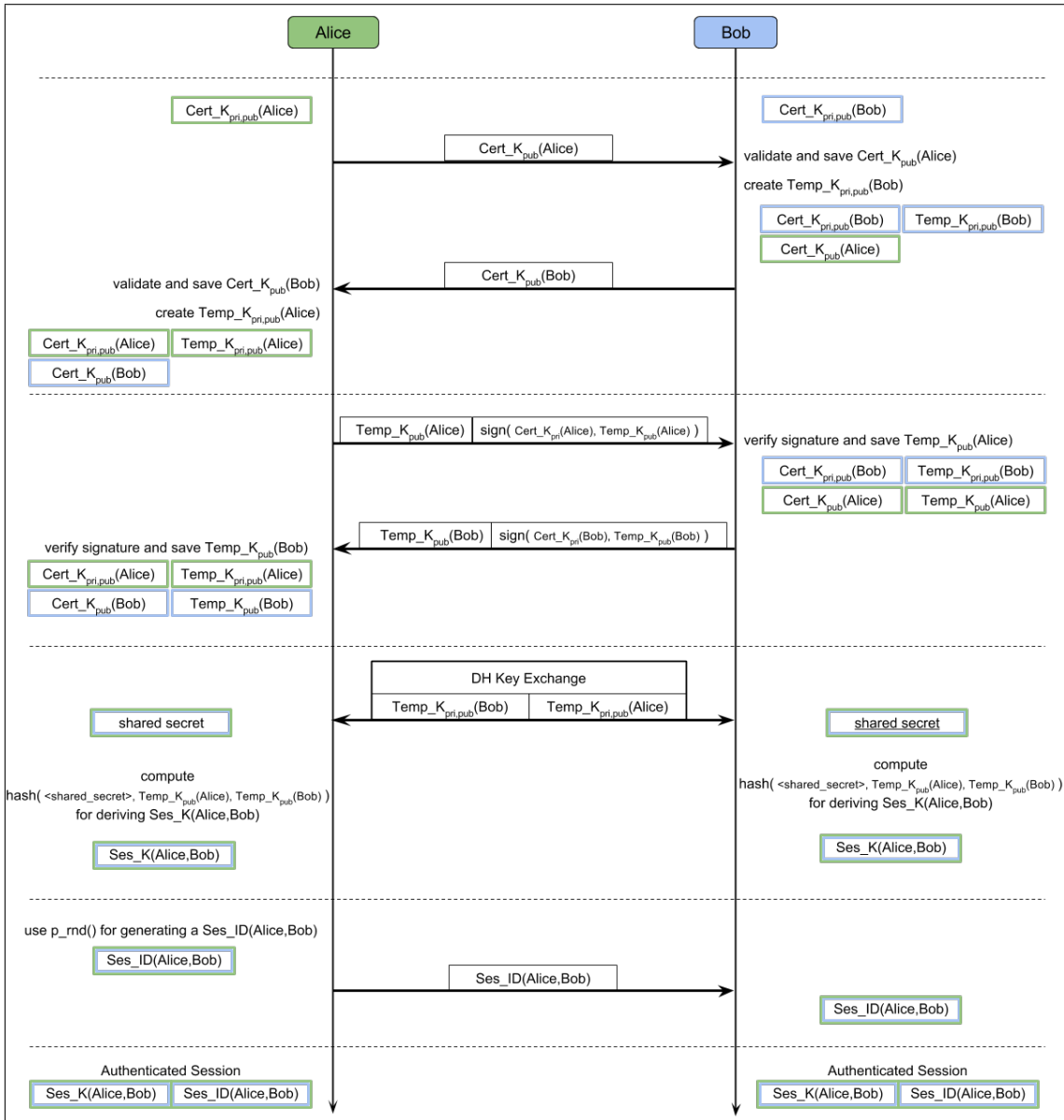


Figure 3.3: Authentication Protocol

Name	Client Authentication Data
UUID	e0628a04-8abc-46b7-b282-d0e0f247f531
Properties	Write

Table 3.3: GATT Characteristic Client Authentication Data

Name	Kiosk Authentication Data
UUID	b305eebb-eac6-4915-82b0-c724fb98659a
Properties	Read

Table 3.4: GATT Characteristic Kiosk Authentication Data

data packet, as defined in [Appendix A](#). Transferring the data within one characteristic, prevents the overhead of multiple write operations. The characteristic is marked as write only, as there is no need to read it. See [Table 3.3](#) for the characteristic definition.

Kiosk Authentication Data: The authentication data of the APD-kiosk, which includes the $\text{Cert}_{\text{K}_{\text{pub}}}$, the $\text{Temp}_{\text{K}_{\text{pub}}}$, and a signature created by signing its $\text{Temp}_{\text{K}_{\text{pub}}}$ with its $\text{Cert}_{\text{K}_{\text{pri}}}$, is provided by another characteristic. When an APD-client is reading this characteristic, the APD-kiosk sends its authentication data, which is packed into an authentication data packet, as defined in [Appendix A](#), to the APD-client. This is done with one characteristic, to minimize the amount of packets, which are required to be sent. The characteristic is marked as read only, because APD-clients shall not be able to change the value. It is defined in [Table 3.4](#).

Session ID: When an APD-kiosk receives a read operation onto the session id characteristic and the key agreement between the APD-client and the APD-kiosk has been finished successfully, a unique $\text{Ses_ID}(\text{APD-kiosk}, \text{APD-client})$ generated by using a `prng()`, is returned to the APD-client. If the key agreement has not been finished yet, a read operation will return zero. The definition of the Session ID characteristic can be found in [Table 3.5](#).

3.4 Authenticated Presence Detection

The authenticated presence detection is triggered, when an authentication has successfully initialized an authenticated session, or in other words the authenticity of the participating entities has been verified and a shared secret session key plus a unique session identifier have been created. For the authenticated presence detection, a challenge response protocol is used, which is explained in [3.4.1](#). The utilized BLE roles are outlined in [3.4.3](#). [Table 3.6](#)

Name	Session ID
UUID	0c811e1e-d244-4c9c-9850-d35a2508f512
Properties	Read

Table 3.5: GATT Characteristic Session ID

Notation	Meaning
Charlie	Entity in the same role as Bob
$\text{mac}(\langle \text{sym_key} \rangle, \langle \text{data} \rangle)$	Message authentication code generated with a symmetric key ($\langle \text{sym_key} \rangle$) of $\langle \text{data} \rangle$
challenge	Challenge, which is generated with a $\text{p_rng}()$
$\text{chal_solved}(\langle \text{entity} \rangle)$	Challenge response computed by $\langle \text{entity} \rangle$

Table 3.6: Notation Extension for the Challenge Response Protocol

extends the already presented protocol notation of [Table 3.1](#), which is then used for the following explanations.

3.4.1 APD Challenge Response Protocol

Within this explanations, Alice represents the APD-kiosk, acting as the challenging entity, whereas Bob and Charlie are representing APD-clients, who are challenged by Alice. Having a shared secret session key and the corresponding unique session identifiers for Bob and Charlie, enables Alice to challenge both with the same challenge value. Therefore, it is possible for Alice to broadcast the challenge. This challenge is packed into a challenge packet, as defined in [Appendix A](#). Bob and Charlie can then listen for this broadcasted packet, extract the challenge, and solve it by computing $\text{mac}(\text{Ses_K}(\text{Alice}, \text{Bob}), \text{challenge})$ and $\text{mac}(\text{Ses_K}(\text{Alice}, \text{Charlie}), \text{challenge})$ respectively. These computed values represent $\text{chal_solved}(\text{Bob})$ and $\text{chal_solved}(\text{Charlie})$. Considering Bob wanting to solve and answer the challenge of Alice, he is required to combine $\text{chal_solved}(\text{Bob})$ with the unique $\text{Ses_ID}(\text{Alice}, \text{Bob})$ in an challenge response packet, as defined in [Appendix A](#). This packet is then broadcasted by Bob, so that Alice can receive it. Alice receives the challenge response packet of Bob, extracts the $\text{Ses_ID}(\text{Alice}, \text{Bob})$, and uses it to identify Bob as sender. For verifying that the sender was indeed Bob, or in other words for authenticating Bob, Alice computes $\text{mac}(\text{Ses_K}(\text{Alice}, \text{Bob}), \text{challenge})$ on her own and compares it to the received $\text{chal_solved}(\text{Bob})$. When both values, $\text{chal_solved}(\text{Bob})$ and the computed value $\text{mac}(\text{Ses_K}(\text{Alice}, \text{Bob}), \text{challenge})$ from Alice, are equal, Alice can be sure, that the challenge has been solved by Bob, because he is the only entity that has the same $\text{Ses_K}(\text{Alice}, \text{Bob})$. Alice does this every time she receives a challenge response packet. After a defined time period, Alice renews the challenge that is broadcasted to the participating entities for mitigating replay attacks, which is explained in [3.4.2](#). Therefore, Bob has to listen to the broadcasts of Alice continuously and recompute his challenge response, when he receives a new challenge. The whole process for the authenticated presence detection is illustrated in [Figure 3.4](#). This challenge response protocol for the authenticated presence detection has following advantages:

- Using a connectionless broadcast model, for performing the challenge response protocol, removes the computational costs for organizing connections, which arise when using connected communication channels.
- The usage of shared authenticated sessions enable Alice to challenge all participating APD-clients with the same challenge, because APD-clients, which want to authentic-

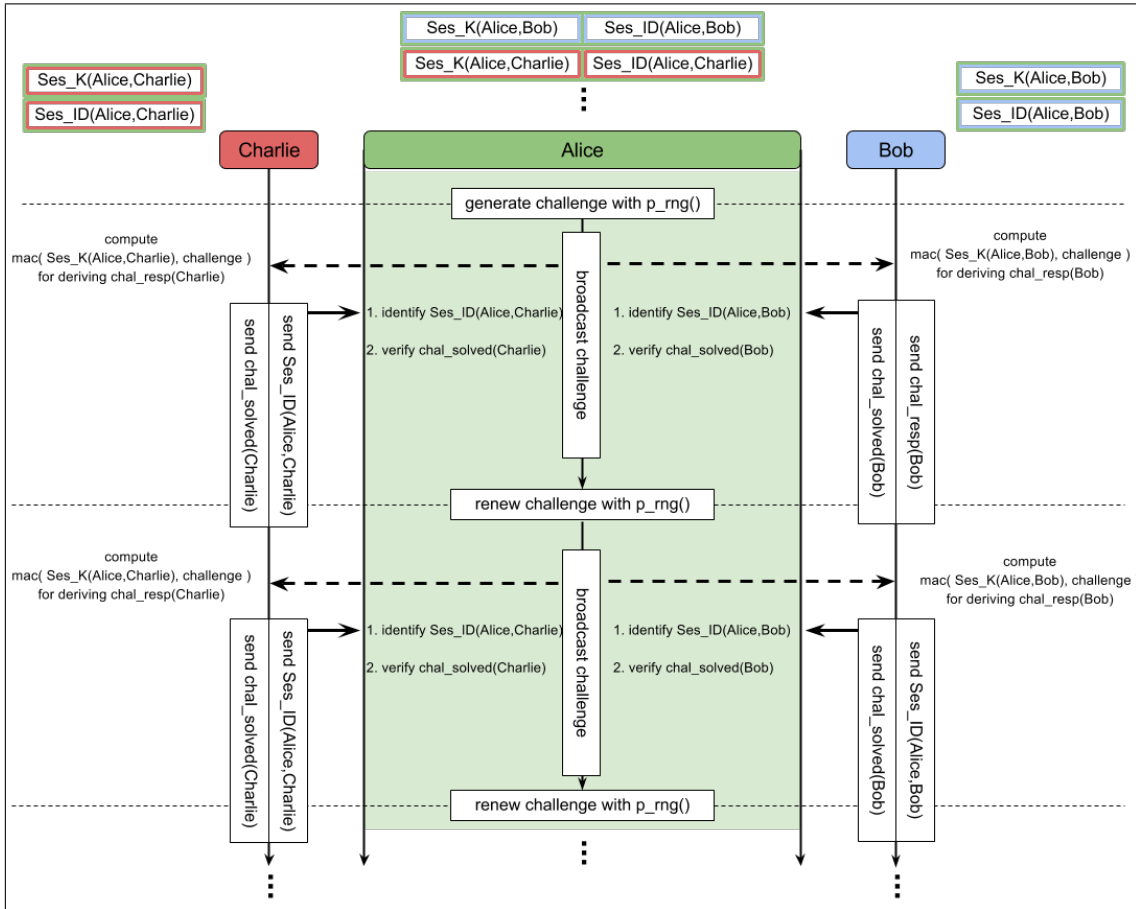


Figure 3.4: Challenge Response Protocol for Authenticated Presence Detection

ate themselves, have to solve that challenge, by using the secret session key created during the authentication process.

- The possibility of authenticating entities by computing a MAC, for instance with the AES-CMAC algorithm, which requires only 128 bit, as stated in the work of Song et al. [59], reduces the data of the solved challenge to a few bits, which makes it faster and more usable for BLE.
- Bob and Charlie are broadcasting their challenge response continuously. This enables Alice to authenticate Bob and Charlie with every received challenge response packet. This leads to a high frequently performed authentication. Additionally, the received challenge response packets are indications for the presence of Bob and Charlie as well, because their challenge response packets can only be received by Alice, when Alice is within a certain range of them.

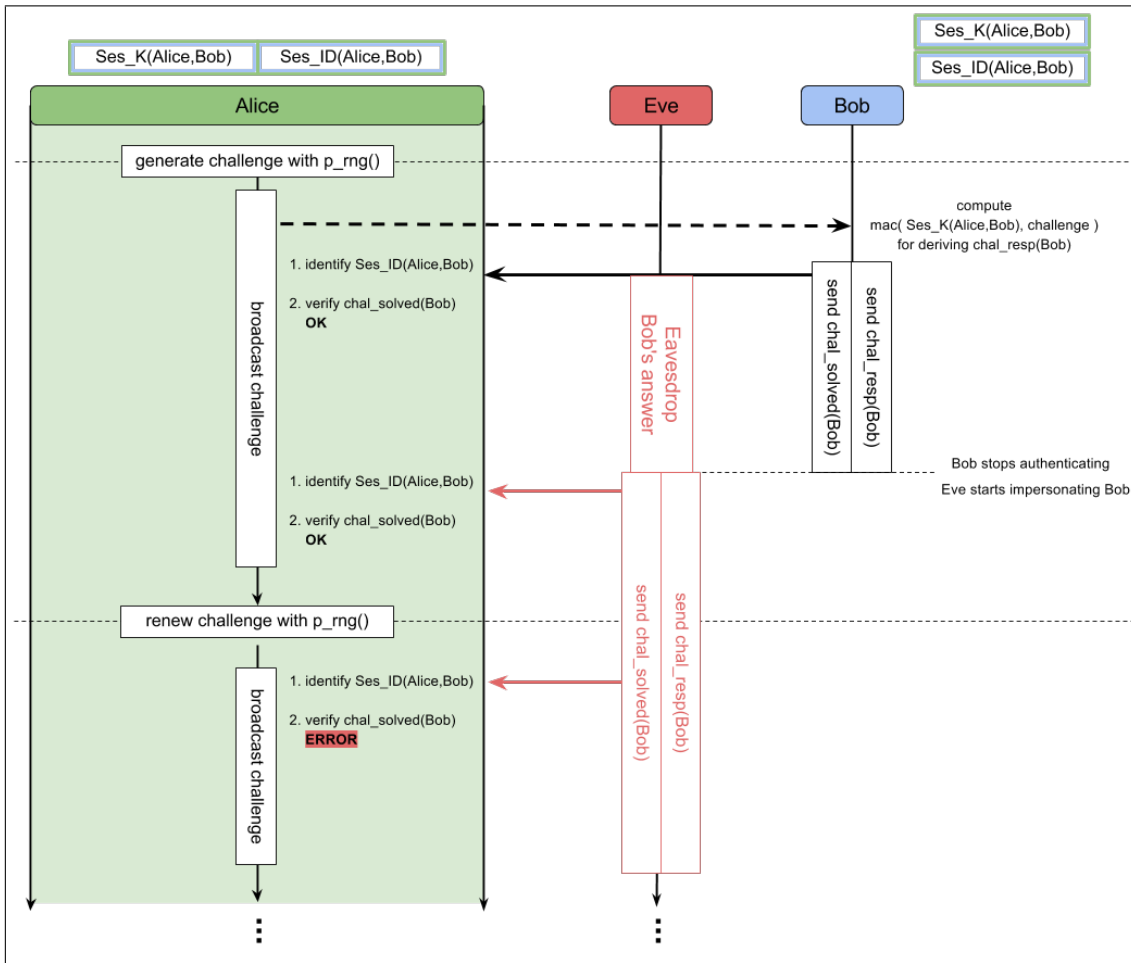


Figure 3.5: Limited Replay Vulnerability of the Challenge Response Protocol

3.4.2 Vulnerabilities of the Challenge Response Protocol

The presented challenge response protocol in 3.4 is vulnerable to some attacks. An adversary is able to perform a replay attack by eavesdropping the responses of the challenged entities and retransmitting the recorded messages. This lets the adversary fake the presence of an entity, even though the entity is already gone. However, the adversary can only fake the presence of an entity until the challenge is renewed. Therefore, the attack is limited to a defined period. This is illustrated in Figure 3.5.

No mitigates regarding relay attacks and denial of service attacks are carried out. A relay attack is performed, when an adversary uses the APD-clients service in an unauthorized way, by retransmitting the communication between APD-client and APD-kiosk over another channel. This results in a faked presence of the APD-client. A denial of service attack could be achieved by jamming the BLE signals or flooding the APD-kiosk with more requests than the APD-kiosk can handle. This would lead to an APD-kiosk, which is not working properly. [68, p. 143,519], [74]

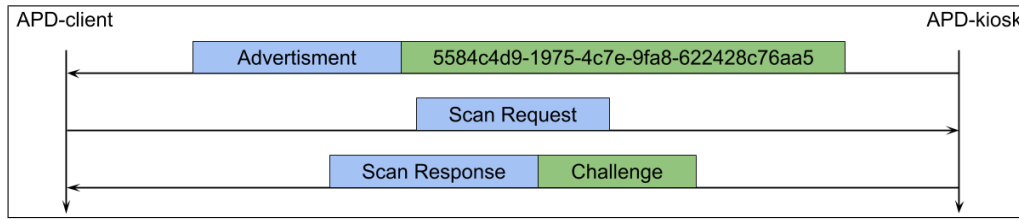


Figure 3.6: Scan Procedure for Challenge Advertisement

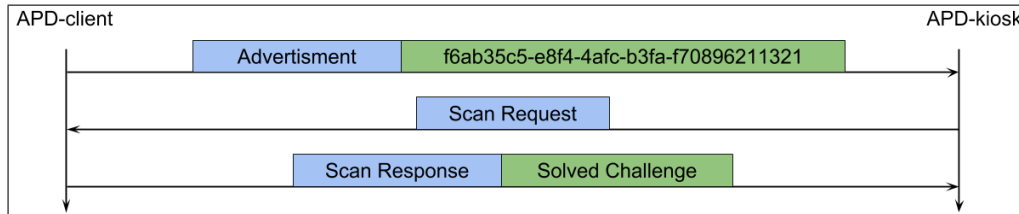


Figure 3.7: Scan Procedure for Challenge Response Advertisement

3.4.3 BLE Interface

Both devices, the APD-kiosk and APD-client, take over the role of a BLE broadcaster and a BLE observer at the same time. This is done to implement the APD challenge response protocol, explained in 3.4.1. The APD-kiosk advertises the UUID 5584c4d9-1975-4c7e-9fa8-622428c76aa5, which enables APD-clients to identify it as participating APD-kiosk. The APD-clients send scan requests to the APD-kiosk, which are answered by scan responses containing the current challenge. This is shown in Figure 3.6. The challenge is packed into a challenge packet, which is defined in Appendix A. When the challenge is updated, which is explained in 3.4.1, the scan responses for scan requests are updated as well, to transmit the new challenge. APD-clients extract the challenge from the challenge packet within the scan response and solve it as shown in 3.4.1.

The APD-clients announce their presence by advertising the UUID f6ab35c5-e8f4-4afc-b3fa-f70896211321. The APD-kiosk sends scan requests to the APD-clients, which are answered with scan responses, containing challenge response packets. The format of a challenge response packet is defined in Appendix A and contains the solved challenge. This process is illustrated in Figure 3.7

3.4.4 Absence Timeout

Detecting the absence of an APD-client, or in other words, detecting, when an APD-client has left the AoI, has to be done with a timeout, due to the utilization of a broadcaster-observer architecture for the presence detection. When the APD-kiosk has not received a signal from an APD-client for a defined period of time, the so called absence timeout, it is considered as gone. Defining this period of time is not that easy, due to the wireless nature of the presented design. There are many situations, where a present APD-client is potentially not detected, due to unfortunate coincidence. For instance the authentication via the GATT server could be aborted, due to interference signals. Another potential failing detection could happen, due to an overwhelming amount of APD-clients within the

AoI and an APD-kiosk which is not able to detect that much APD-clients simultaneously. [5.4](#) describes an experiment, which has been conducted with the aim of finding such an absence timeout, based on a experimental setup, that includes a lot of simultaneously present APD-clients.

Chapter 4

Implementation for Android

The implementation of the authenticated presence detection design, presented in [Chapter 3](#), has been done for Android mobile devices. This chapter starts with a brief introduction of Android and its support of BLE. Then, an overview of the implemented system is presented, followed by details of the implemented parts and how they work together to perform the authenticated presence detection. Last, the flow of actions, when performing two main scenarios is explained.

4.1 Android

Android is an open source operating system, based on the linux kernel, which originated in 2005. It has been developed by Android Inc. until bought by Google in 2005. [\[72\]](#)

When it comes to market share of mobile operating systems, Android and iOS are the two big players. Android had in the third quarter of 2016 a global market share of 88%, whereas Apples iOS had 12.1%, according to an article written by Bhattacharya [\[14\]](#). A different article, written by Vincent [\[70\]](#), states a global market share of 81.7% for Android and 17.9% for iOS in the fourth quarter of 2016. However, the situation is quite different in the United States. In January 2016, the United States market share of Android was at 52.8% and the iOS market share at 43.6%, according to the *comScore Reports January 2016 U.S. Smartphone Subscriber Market Share* [\[25\]](#). Nevertheless, all of these statistics indicate that Android and iOS are running on the majority of mobile devices.

An Android device usually comes with a pre-installed software, which enables users to download and install applications on their device. The most popular one is developed by Google and called *Play Store*. The information in the article *Number of Android applications* [\[52\]](#) shows, that about 3,171,376 applications are provided by the Play Store in 2017. However, there are no restrictions to the application's origin. Users can install and obtain applications from wherever they want. As a consequence, there exist several stores, which provide the feature of application installation, as documented by Wikipedia contributors [\[73\]](#).

The previously mentioned numbers like the market share and the number of available applications, highlights the importance of Android within the mobile device sector and indicates why it should be targeted from mobile application developers.

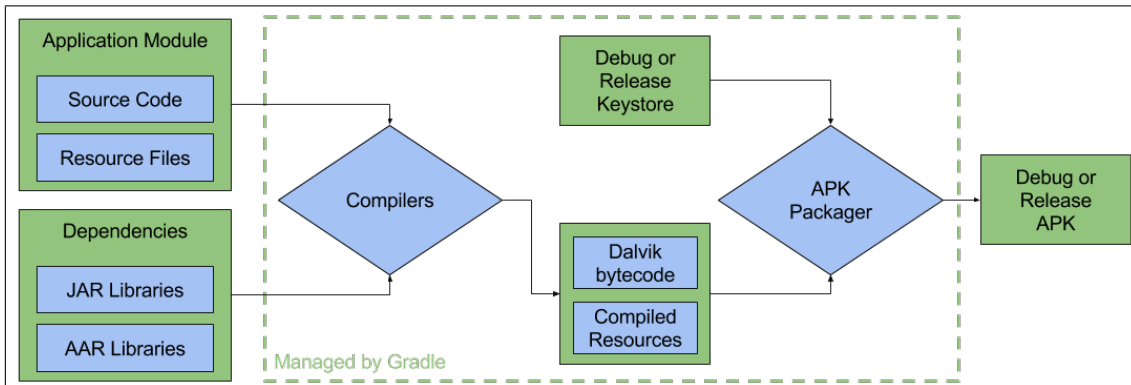


Figure 4.1: Android Build Process, adapted from the documentation for *The Build Process* [63]

4.1.1 Application Development

Application development for Android is supported by a huge information base, which is continuously improved and updated. This page for developers, contains all the information required for developing Android applications. A centralized collection of information is important, due to the high frequently performed extensions and improvements of the Android operating system and the provided API. The current level of the Android API is 26. This makes about 2.7 API updates per year and shows the rapidly applied changes to the operating system. Android distinguishes between Android version and API level. The Android version is presented to the public and publishes the features to the outside. The API level, on the other hand, defines the features, which are usable from applications. Android is forward compatible, however not backward compatible. That means, an application which is developed for a specific API does not necessarily work on an older API as well. [5], [31], [71]

Android Studio¹ is the official integrated development environment for Android. Android Studio offers not only an intelligent code editor and an Android device emulator, but also provides a build chain which lets developers easily transform their project files into a valid Android Application Package (APK). This build process can be seen in Figure 4.1. The application module contains the source code and the resource files. The source code is written in Java and the resources are represented as extensible markup language files. The dependencies of the application module are external libraries like Java ARchive (JAR) files, which pack Java files and other resources into one file. Android Archives (AAR) are similar to JAR files. However, they can contain Android resource files as well. Compilers transform the files from the application module and the dependencies into Dalvik bytecodes. This bytecode is later on executed on the phone by the Android Runtime. The generated bytecode is packed into an debug or release APK. The whole process is managed by the Gradle² android plugin. [6], [7], [50, p. 74], [63]

¹Android Studio Web Site: <https://developer.android.com/studio/index.html>

²Gradle Web Site: <https://gradle.org/>

Scan Mode Setting	Scan Interval in seconds	Scan Window in seconds
Low Latency	5	5
Balanced	5	2
Low Power	5	0.5

Table 4.1: Android BLE Scan Settings for API ≥ 21 , based on Information of [15]

4.1.2 Android and BLE

Due to the focus on mobile devices, Android supports a variety of wireless technologies, like Bluetooth classic, BLE, Wi-Fi or GPS. These technologies are used for exploring the opportunities in the fields of presence detection, crowd sensing, and social science as mentioned in 2.3. Not only the support of BLE, but also the combination of so many wireless technologies makes Android mobile devices valuable for researchers as well as for commercial use cases.

BLE Scanning Since API 18, it is possible to scan for BLE advertisements, by using the Android API. If an advertisement is scanned, the advertising Bluetooth device, in form of an object, the RSSI value and the advertised data are handed over to a callback. Up to API 20, the data of the scan record had to be parsed by the developer, to retrieve any details of the advertising packets. As of API 21, the parsing is done by the Android operating system itself. The callback of the API provides a *ScanResult* object, that contains the data in a well prepared form, as seen in Listing 4.1. The scanning itself has been improved by adding filter possibilities for the advertising packets. The developers can define filter parameters, which are applied to incoming advertisements by the Android operating system. Filter options are service UUIDs, names and addresses of remote BLE device and manufacturer specific data. Additionally, API 20 added the opportunity of choosing a scan mode setting, which defines the scan interval and the scan window. The three setting options, that were introduced are listed in Table 4.1. With API 23, an additional opportunistic scan mode has been added, which listens passively to other scan results without starting its own scans. Additional settings were added as well, which enables the developer to define the callback type, the match mode, and the number of matches per filter. The callback type defines, whether the callback should forward all, only the first, or the last received advertisement. With the match mode, developers can decide, whether the received advertisements must have a certain signal strength to be forwarded. The last additional setting, the number of matches per filter, controls the forwarded advertisements per filter. [17], [18], [57], [58]

```

1 // API 18-20
2 void BluetoothAdapter.LeScanCallback#onLeScan (BluetoothDevice device, int
   rssi, byte[] scanRecord)
3
4 // API 21+
5 void ScanCallback#onScanResult (int callbackType, ScanResult result)

```

Listing 4.1: Scan API Callback Difference, based on Information of [18] and [57]

BLE Advertising With API 21 the possibility of performing BLE advertisements has been introduced. However, not all devices with a matching API level are able to do so.

Advertise Mode	Advertising Interval in Milliseconds
Low Latency	100
Balanced	250
Low Power	1000

Table 4.2: Android BLE Advertising Mode Settings, based on Information of [15]

Advertise TX Power	TX Power in dBm
High	+1
Medium	-7
Low	-15
Ultra Low	-21

Table 4.3: Android BLE TX Power Settings, based on Information of [15]

There are some preconditions, when using the advertising API. The API offers a function call, which returns *true* when BLE advertising is supported. This call can be seen in Listing 4.2. Not all devices return the correct value, when using this function. [15], [17]

Advertisements can be initialized with different configurations. The developer has several options for the advertise mode. This setting changes the advertising interval and can be seen in Table 4.2. Another setting influences the signal strength, or in other words, the TX power. This can be seen in Table 4.3. [2]

```
1 boolean BluetoothAdapter#isMultipleAdvertisementSupported();
```

Listing 4.2: Call for Checking Advertisement Support, based on Information of [17]

The advertising data and the scan response data can be initialized with an *AdvertiseData* object. This object lets the developer define, whether the device's name or the TX power level shall be included. Furthermore, a service UUID, service data, and manufacturer data can be added. A downside is, that the developer has to use the *AdvertiseData* class and can not define the raw data, which shall be transmitted on its own. [1]

BLE Connection The *BluetoothDevice* object, which is received after a successful scan of an advertisement, can be used to establish a GATT connection with the remote BLE device. This can only be done under the premise, that the advertising remote device is connectable and running a GATT server instance. Connections with BLE are available since API 18 and transform an Android device into a GATT client. The call for doing so is shown in Listing 4.3. This method call requires a *BluetoothGattCallback* object as parameter, which receives callbacks, whenever an asynchronous GATT operation has been finished. The returned *BluetoothGatt* object can be used for performing GATT operations, like discovering the services, reading characteristics, and writing characteristics. However, this object shall not be used until a first callback in the *BluetoothGattCallback* object has been received, because this first callback indicates, that the connection has been properly established. With API 21, methods for setting a connection priority and requesting a MTU change have been added as well. The default MTU is 23 and the maximum MTU is 517. The call for setting the connection priority lets the developer use one of three different

Connection Priority	Connection Interval in Milliseconds		Slave Latency	Supervision Timeout in Seconds
	API 21	API 23+		
High	7.5-10	11.25-15	0	20
Balanced	30-50		0	20
Low Power	100-125		2	20

Table 4.4: Android BLE Connection Priority Settings, based on Information of [15]

settings, which define the connection interval, the latency, and the timeout. These settings can be seen in Table 4.4. However, the call for setting the connection priority has a known bug on API 21, which leads to not getting callbacks when using this functionality. [15], [18], [19], [20], [21], [57]

```
1 BluetoothGatt BluetoothDevice#connectGatt(Context, boolean autoConnect,
BluetoothGattCallback)
```

Listing 4.3: Connect to a GATT Server, based on Information of [19]

A GATT server can be added to a device, as of API 18. The developer can start a GATT server on an Android device by calling the method of the *BluetoothManager* object, which is shown in Listing 4.4. The *BluetoothGattServerCallback* object, is the counter part of the previously mention *BluetoothGattCallback* from the GATT client. It gets called, when GATT operations from a GATT client are received. API 18 receives callbacks, when a new connection is established and the services of the GATT server have been discovered. Additionally, there are callbacks received, when a GATT client performs reading or writing operations for characteristics. API 21 adds additional callbacks, which are triggered, when a notifications or an indication have been sent. With API 22 a callback for a MTU change request, when requested by a GATT client, has been added as well. [22], [23]

```
1 BluetoothGattServer BluetoothManager#openGattServer(Context,
BluetoothGattServerCallback);
```

Listing 4.4: Open a GATT server, based on Information of [23]

4.2 System overview

Figure 4.2 shows an overview of the implemented system. All parts in green have been implemented, the ones in blue have already existed. The Android COYERO access Client SDK for mobile applications has been extended with the APD-client’s functionalities. An already existing Android demo application, namely COYERO Client, has been adapted, so that it is using the extended COYERO access Client SDK for showing the functionality of the newly added APD-client features. An additional demo Android application, namely COYERO Detector, has been implemented, which offers the APD-kiosk’s features and uses the authentication functionalities, offered by the COYERO access Kiosk SDK. Therefore, Android devices with the COYERO Client application installed, are able to perform an

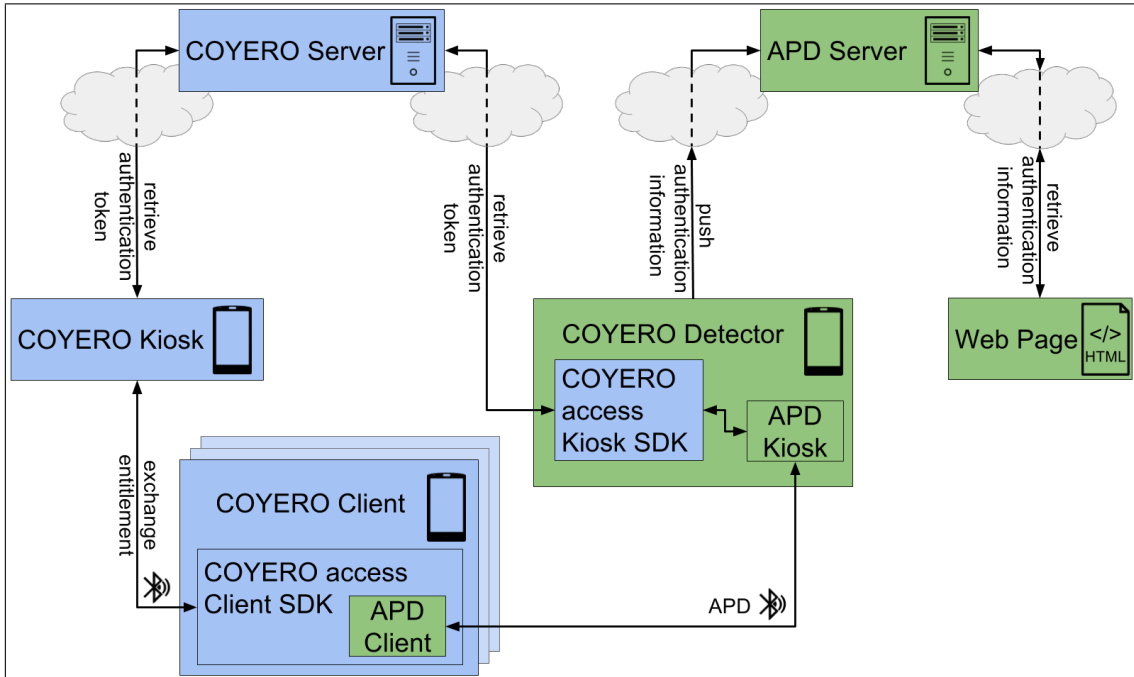


Figure 4.2: System Architecture

authenticated presence detection with Android devices having the COYERO Detector application installed. Additionally, COYERO Detector sends the information of detected nearby devices periodically to the APD Server, where the information is stored. The APD Server runs a python web server, which offers a web page, consisting of Hypertext Markup Language (HTML) and JavaScript files. This enables users to easily retrieve the currently present and authenticated nearby devices with a browser.

The COYERO access environment, which is presented in 2.4, is the foundation of the implementation. COYERO access offers a framework, which provides a PKI for authenticating COYERO kiosks and COYERO clients. Such a PKI is required for the APD design, presented in Chapter 3. The COYERO server, is able to issue so called authentication tokens, which are basically digital certificates issued by a CA. It can do this for COYERO kiosks and COYERO clients. This authentication tokens are therefore available within the COYERO Client application and can be used for authentication. The COYERO Detector application can be seen as a new feature within the COYERO environment. It requests an authentication token from the COYERO server, by performing a registration process and a login. Therefore, both, COYERO Client and COYERO Detector, have access to authentication tokens, which serve as cryptographic basis for performing the authenticated presence detection.

4.3 COYERO Detector Demo Application

COYERO Detector is an Android application, which utilizes the PKI of the COYERO environment, for implementing the functionalities of an APD-kiosk. These functionalities are required to perform an authenticated presence detection. An overview of COYERO


	
COYERO Detector	
Minimum API	API 21
Device Requirements	BLE Support Multiple BLE Advertisement Support
Software Dependencies	COYERO access Kiosk SDK Spongy Castle ³ Volley ⁴

Table 4.5: Properties of the COYERO Detector Application

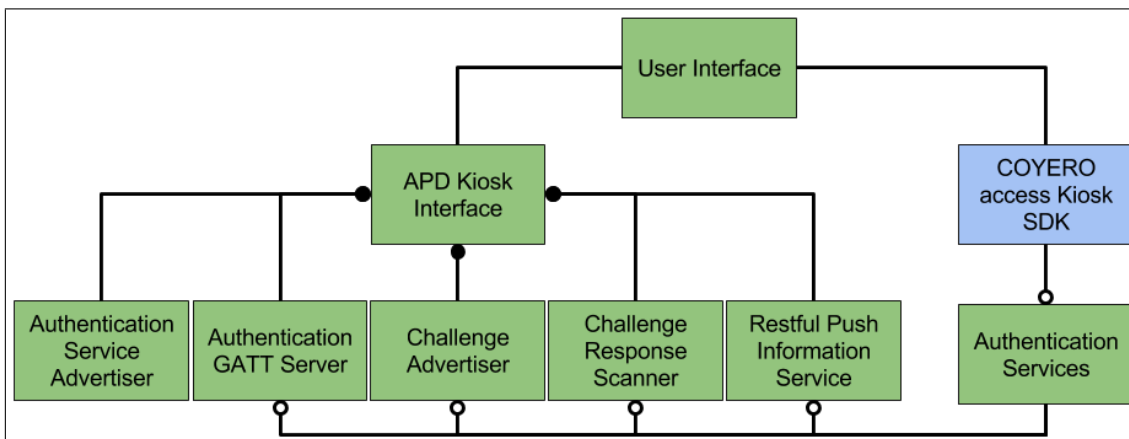


Figure 4.3: Logical Architecture of COYERO Detector

Detector’s properties can be found in Table 4.5. COYERO Detector can be seen as service provider, which is started once and offers the functionalities of the APD-kiosk until it is explicitly stopped. The supported minimal API is 21, which would cover about 75% of the Android devices in use, according to the *Android Dashboards* [4].

The logical architecture represents the separable functional parts within the COYERO Detector sources. Each of this components is responsible for performing a distinct task. This logical architecture is visualized in Figure 4.3, by using the notation presented in the work of Kruchten [42]. This notation has been extended with the colouring of the boxes indicating, whether the part was already existing (blue) or not (green).

User Interface: The UI provides a login view, which can be used, for entering COYERO user credentials. When the credentials are valid, the COYERO access Kiosk SDK is initialized and the application switches to the ‘APD view’. This view provides three possibilities for the user to interact with it. These are navigating back to the login view, starting the authenticated presence detection process, and stopping it. The UI can be seen in Figure 4.4

³Spongy Castle is a newly packet version of the security library Bouncy Castle, which is supported by Android. [62]

⁴Volley is a library for making asynchronous RESTful calls. [67]

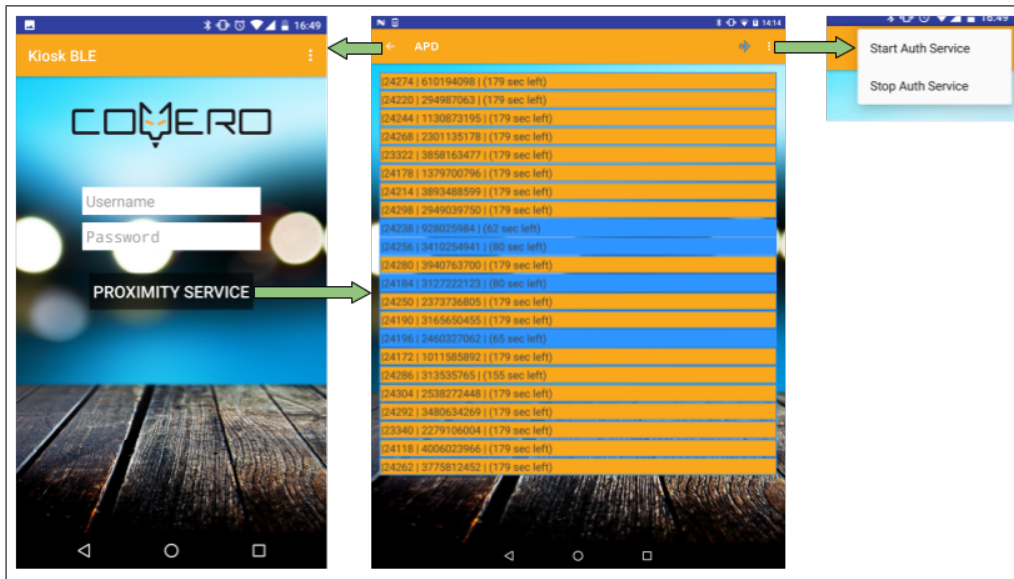


Figure 4.4: COYERO Detector Views

COYERO access Kiosk SDK: It is part of the existing COYERO framework and provides the authentication token, which is used as basis for the authentication protocol.

APD Kiosk Interface: It is the management element for the five components, which are implementing the APD-kiosk’s functionalities. It provides a simple interface, which is used from the UI for activating or deactivating this functionalities.

Authentication Service Advertiser: It announces the presence of an APD-kiosk to the outside. It does this by using a connectable BLE advertisement, which can be used by nearby APD-clients to establish a connection. It is set to a *low latency* advertisement with a *medium* TX power. These settings define an advertising interval of 100[ms] and a TX power of +1[dBm] as shown in [Table 4.2](#) and [Table 4.3](#), respectively. The data of the advertisement is the UUID of the GATT Authentication service, which is specified in [Table 3.2](#). This component is stopped once a connection to the Authentication GATT Server is active. It is restarted after the connection has been closed. This prevents more than one simultaneously active connections to APD-clients. The reason for doing this, is to have a unified behaviour for all devices running the COYERO Detector application, because the maximal supported simultaneously connections to GATT clients is potentially different for devices (see [5.5](#)). Additionally, the restart of the Authentication Service Advertiser is triggered three seconds after the connection has been successfully closed, to be able to receive challenge response advertisements, before starting a potential new connection. This is carried out due to the findings shown in [5.5](#).

Authentication GATT Server: The Authentication GATT Server implements callback functions for the Android GATT Server implementation and provides the GATT Authentication Service functionality presented in [3.3.2](#). Therefore, it receives callbacks

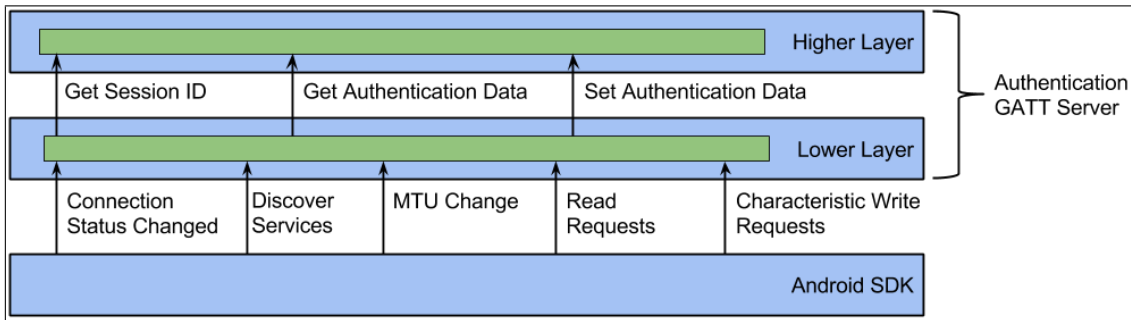


Figure 4.5: Layer System of the Authentication Gatt Server

for all incoming BLE GATT operations, which are related to the Authentication GATT Server. These BLE operations can be connection attempts, disconnections, discover operations, a request for a MTU change, or read and write operations for characteristics of the GATT Authentication Service. The Authentication GATT Server is divided into two layers. The lower layer is responsible for providing a controlled domain specific interface, by performing following tasks:

- Preventing concurrent connections, by disabling the Authentication Service Advertiser, when receiving a new connection attempt, and restarting it, delayed by three seconds, when the current connection is closed.
- Updating the settings when an MTU change is received.
- Redirecting the BLE read and write callbacks, for characteristics of the Gatt Authentication Service, to domain specific calls of the higher layer.

The higher layer, not only provides calls for reading the authentication data, writing the authentication data, and reading the session id, but also uses the Authentication Services for performing the cryptographic operations, which are required for a verified authentication. This layering is shown in [Figure 4.5](#).

Challenge Advertiser: The Challenge Advertiser registers itself at the Authentication Services. As a result, it gets notified whenever the challenge has been updated. This updated challenge is then advertised via BLE as described in [3.4.3](#). The advertising interval and TX power settings are the same as for the Authentication Service Advertiser, because these two advertisements define the AoI with their signal.

Challenge Response Scanner: It scans for challenge response advertisements of APD-clients as shown in [Figure 3.7](#). The extracted solved challenge is then, together with the received session identifier, passed over to the Authentication Services and verified. The BLE scanner is set to a *low latency* setting, which results, according to [Table 4.1](#), in a scan interval and a scan window of five seconds.

RESTful Push Information Service: It send periodically updates to the APD Server of currently present authenticated APD-clients, by using the APD Server's RESTful API, which is specified in [Appendix B](#). The update interval is set to one second.

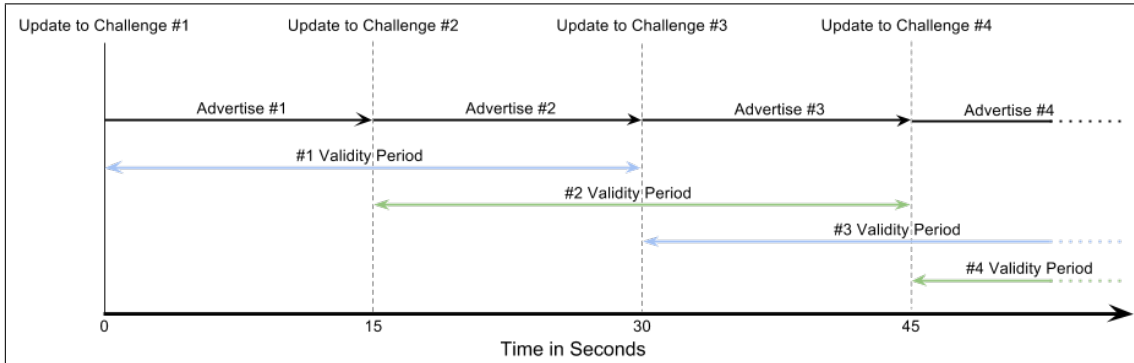


Figure 4.6: Validity Period of Challenges

Authentication Services: The Authentication Services manage and coordinate all important cryptographic tasks. They are the interface to the COYERO access Kiosk SDK and therefore to the COYERO Detector’s authentication token. They do basically three different things:

1. They are responsible for performing the cryptographic operations, which are required for creating the shared symmetric session keys with the APD-clients. The differentiation of the APD-clients is either done by the authentication token or the created session identifier, meaning either the authentication token or the session identifier is passed to the Authentication Services, when instructing them to verify an authenticity. This enables unambiguous identification and verified authentication of the APD-client.
2. The Authentication Services periodically update the challenge for the APD challenge response protocol. The update period is set to 15 seconds, meaning every 15 seconds there is a new challenge advertised. As of the first time a challenge has been advertised, it is accepted as solved challenge response for 30 seconds. Therefore, after the first challenge update, there are always two challenges which are valid simultaneously. This is done because it reduces the power consumption of the COYERO Client application, which is explained in 4.4. This so called validity period of 30 seconds per challenge, is illustrated in Figure 4.6.
3. They keep track of the currently nearby and authenticated APD-clients, or in other words, they manage the Absence Timeout, which is explained in 3.4.4.

4.4 COYERO SDK Extension and COYERO Client Demo Application

The COYERO client SDK provides all the functionalities, which are required to perform the COYERO authentication with their authentication tokens. This SDK has been extended with the features of an APD-client. The APD-client’s features are usable with a separate background service, so that the device’s display can be turned off, without stopping


COYERO Client		
Minimum API for the Application	API 17	
Minimum API for the APD-client's features	API 21	
Device Requirements	BLE Support Multiple Advertisement Support	
Software Dependencies	COYERO Client SDK Spongy Castle ⁵	

Table 4.6: Properties of the COYERO Client Application

the APD-client's functionalities. Therefore, acquiring a wake lock is required, to ensure that the Android device does not stop this background service, because it switches into a sleep mode, as explained in the article *Keeping the Device Awake* [41]. The COYERO Client application, which was already available and uses the COYERO client SDK, has been extended as well, so that the APD-client features are usable.

This application is the counterpart to COYERO Detector. All BLE interfaces, like advertisements, scanners, and GATT servers, which are offered by COYERO Detector, are used by COYERO Client. The properties of COYERO Client can be seen in Table 4.6. The newly added features of the extended COYERO access Client SDK require a minimum Android API of 21. However, the minimal supported API of the COYERO access Client SDK remains 17. The new features are simply not accessible using an API ≥ 17 and < 21 . Therefore the APD-client features would be available for 75% of the currently available Android devices, according to the *Android Dashboards* [4].

The logical architecture, shown in Figure 4.7, is like the architecture of COYERO Detector based on the notation of Kruchten [42], extended with a colouring, which indicates, whether a component was already existing (blue) or has been newly developed (green). Each component of Figure 4.7 is responsible for providing a distinct feature. As COYERO Client is the counterpart of COYERO Detector, following correlations between both logical architectures can be highlighted:

- The *Authentication Service Scanner* scans the advertisements of the *Authentication Service Advertiser*.
- The *Authentication GATT Client* uses the *Authentication GATT Server* for performing an authentication, which results in an authenticated session.
- The *Challenge Responder* scans the periodically updated challenge, which are advertised by the *Challenge Advertiser* and additionally advertises the solved challenge, which is then scanned by the *Challenge Response Scanner*.
- Both use a *COYERO access SDK*, which provides the authentication token.

⁵Spongy Castle is a newly packet version of the security library Bouncy Castle, which is supported by Android. [62]

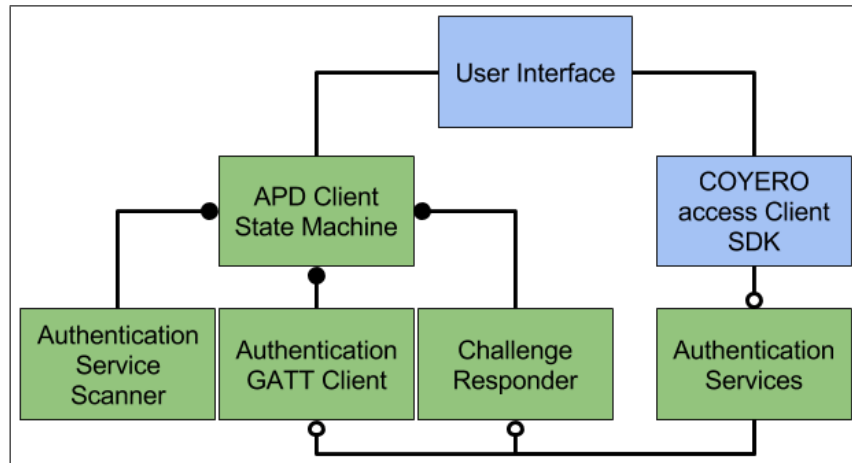


Figure 4.7: Logical Architecture of COYERO Client

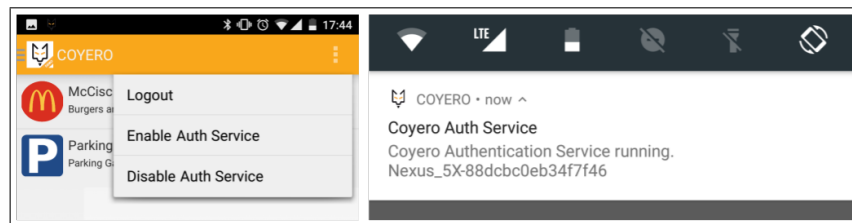


Figure 4.8: COYERO Client UI Extensions

User Interface: The UI was already available in the COYERO Client application. It provides a login, which initializes the COYERO access Client SDK, meaning all required information, for the authentication, is stored locally on the Android mobile device. The UI of COYERO Client has been extended with a menu entry, which allows starting and stopping the APD-client functionality. When this functionality is active, a notification in the status bar of the Android device is displayed. This UI extensions can be seen in Figure 4.8.

COYERO access Client SDK: It provides the authentication token of the COYERO environment, which is the basis for the authentication process.

APD Client State Machine: A state machine is implemented, which manages the correct order of the required procedures, for achieving an authenticated presence detection (see Figure 3.2b). It starts of by initializing the Authentication Services Scanner, which performs the recognition of an APD-kiosk. When an APD-kiosk is found, it uses the Authentication GATT Client for executing the authentication, which either fails or succeeds. When failing, it starts performing the recognition again with the Authentication Service Scanner. In the case of succeeding, it starts the Challenge Responder, which then performs the authenticated presence detection. When the APD-kiosk is not reachable anymore, the Challenge Responder is stopped and the recognition with the Authentication Service Scanner is started again.

Authentication Service Scanner: This scanner is looking for BLE advertisements which contain the UUID of a GATT Authentication Service. This UUID is defined in [Table 3.2](#). An advertising, containing this UUID, indicates the presence of a APD-kiosk. The scanner settings are set to a *low latency* mode, which is a five second scan interval and scan window according to [Table 4.1](#).

Authentication GATT Client: This component is used for performing the authentication protocol from [3.3.1](#), using a BLE GATT connection. Therefore, it implements a consecutive chain of small tasks, which depend on each other. Almost all of them are asynchronous and require a timeout mechanism to prevent getting stuck waiting for an answer. These timeouts have been defined in a way, allowing the attempts to be successful. In other words, the timeouts are quite high, to prevent aborting unfinished but still running attempts. When a task times out, the whole task chain is aborted and the authentication GATT client is closed. The only exception is the change of the MTU, because this task is optional and only increases the performance of following data transfers. The different tasks of the task chain, their required execution order and their defined timeouts are as follows:

1. **Connecting** [Timeout: 10 sec]: The connectable advertisement, which has been scanned from the Authentication Service Scanner, provided a Bluetooth address. This address is used for starting the GATT connection process.
2. **Discovering** [Timeout: 10 sec]: After a connection is successfully established to a GATT server, the Authentication GATT Client starts the discovery process, to retrieve GATT services, which are offered by the GATT server. This usually provides a GATT Authentication Service and its characteristics.
3. **Generating Client Authentication Data** [Timeout: N/A]: This is the only synchronous task within the task chain and has no timeout. It requests the required cryptographic information from the Authentication Services and packs it into the required packet format (see [Appendix A](#)).
4. **Requesting MTU change** [Timeout: 1 sec]: It is the only task, which does not terminate the whole task chain, when timing out. It computes the MTU based on the packet's size, which has to be transferred. When the packet size, in bytes, is smaller than the minimum MTU, the MTU is not changed. When it is bigger than the maximum MTU, the maximum MTU is requested. When it is a value in between the minimum and the maximum MTU, the newly requested MTU, is the packet's size in byte. The minimum and maximum MTU of Android are noted in [4.1.2](#).
5. **Sending Client Authentication Data** [Timeout: 10 sec]: By using a characteristic write operation, this task sends the previously prepared authentication data packet to the GATT server.
6. **Requesting Kiosk Authentication Data** [Timeout: 10 sec]: A characteristic read operation is used, for retrieving the authentication data of the APD-kiosk. The data is received as a authentication data packet, as defined in [Appendix A](#). This data is required for performing a key exchange, which creates the symmetric session key.

7. **Requesting Session ID** [Timeout: 2 sec]: The last task in the task chain is requesting the unique session id. The receiving of a valid session id marks the authentication process as successful. When this task is finished, the Authentication GATT Client is closed.

Challenge Responder: This component scans for challenges as shown in [Figure 3.6](#). The received challenge is then solved and advertised, as illustrated in [Figure 3.7](#). When a challenge is scanned, which has not been scanned yet, the advertisement of the solved challenge is updated. Scanning a new challenge leads to pausing of the BLE scanner for 15 seconds, to reduce the power consumption of COYERO Client. After 15 seconds the scanning for challenges is restarted. The advertisement of the solved challenge is never stopped, because a high frequent authenticated presence update is desired. The Challenge Responder has a timeout of ten seconds. When the challenge scanner is running and no challenge is scanned for a period of ten seconds, the Challenge Scanner stops scanning for the challenge and stops advertising the solved challenge. This means the APD-kiosk is considered as gone.

Authentication Service: It uses the authentication token offered by the COYERO access Client SDK for performing three cryptographic functions, which are as follows:

1. Authenticating the found APD-kiosk, by verifying the APD-kiosk's received authentication token.
2. Performing the key agreement for generating the authenticated session.
3. Solving received challenges.

4.5 APD Server

The APD Server has been hosted on an Amazon web service. It is a web service written in python 2.7, which offers basic functionality for visualizing the information of the currently authenticated nearby devices. Updating the information is done by the COYERO Detector application, using the APD Server's RESTful interface.

4.5.1 Update Authentication Information

It offers a RESTful PUT call, which can be used to update the information of the currently present and authenticated devices. Therefore, a PUT call to the host's address with content type *application/json* and a JavaScript Object Notation (JSON) representation of the information as body, stores the JSON body as file on the server. The definition of this PUT call, with example JSON data, can be seen in [Appendix B](#).

4.5.2 Visualize Information

A HTML page with JavaScript can be retrieved for visualizing the authentication information, which is stored on the APD server. The JavaScript polls the JSON file from the web server, which contains the device authentication information, with a RESTful

Authentication Information		
23 authenticated device(s) available - Last Update: 12/07/2017 @ 14:58:26.719		
Token ID of Client	User ID	Remaining Time
LG_G4-f677cdc4a3bd7684 (24118)	308473180	179 seconds
E9653-14f9e2449c6544a3 (23340)	2806132514	178 seconds
Nexus_5X-88dcb0eb347f16 (23322)	192580328	165 seconds
Token ID of Dongle	User ID	Remaining Time
24368	707470301	179 seconds
24344	4101999489	179 seconds
24440	2852561207	179 seconds
24386	159841789	179 seconds
24338	967717347	179 seconds
24392	2056392145	179 seconds
24416	1038280320	179 seconds

Figure 4.9: Web Server Information Visualization

GET call and updates the table in the HTML. Additionally, the JavaScript polls supplementary information for the APD-clients, which are stored on the server as well, with another RESTful GET call. This supplementary APD-client information, contains the APD-client’s name and the id of the authentication token. This enables mapping a human readable name to the authentication token id. The poll frequency is set to one second, meaning the visualized information are almost available in real-time. The RESTful calls, which are used from the JavaScript to fetch the data from the APD server, are shown in [Appendix B](#). An example of the visualization, is shown in [Figure 4.9](#).

4.6 Process Architecture

The process architecture shown in [Figure 4.10](#) is as well based on a notation presented by Kruchten [42] and extended by a colouring, where green indicates new features and blue already available ones. This process architecture shows the different concurrently running tasks for the COYERO Client, the COYERO Detector, the APD Server, and an arbitrary web browser. The two groupings COYERO Client and COYERO Detector highlight the different nature of the two processes. COYERO Detector has concurrently running tasks for almost all of its logical components. It acts like a service provider and is required to offer all its functionalities to more than one APD-client. These functionalities shall be concurrently usable. COYERO Client, on the other hand, has only one new task. This task is running a state machine, which uses the logical components for performing the required actions. This highlights the ‘client-like’ nature of the COYERO Client, as well as its responsibility to manage the authenticated presence detection related actions. The APD Server process is responsible for collecting the data, by offering a RESTful interface, which is used by the COYERO Detector. The web browser process updates its visualization of the currently present authenticated devices, by periodically requesting the current information from the APD Server.

The distribution of the processes onto different hardware platforms is provided by the process architecture, seen in [Figure 4.10](#), as well. Each mobile device shall have only one running instance of COYERO Detector or COYERO Client. The APD Server is hosted on a server which is accessible for retrieving the information. Only the web browser process does not require a separate device and can be started on any device, which supports web browser functionalities.

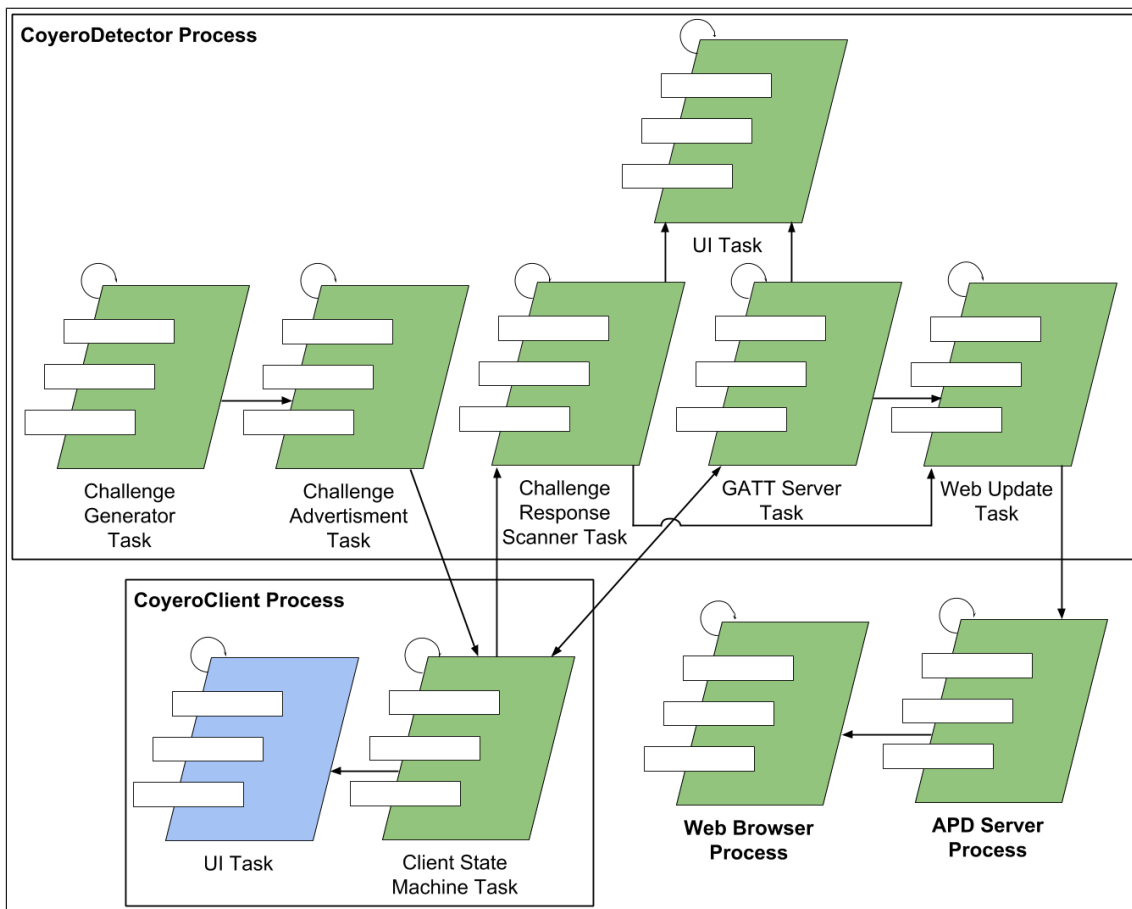


Figure 4.10: Process Architecture

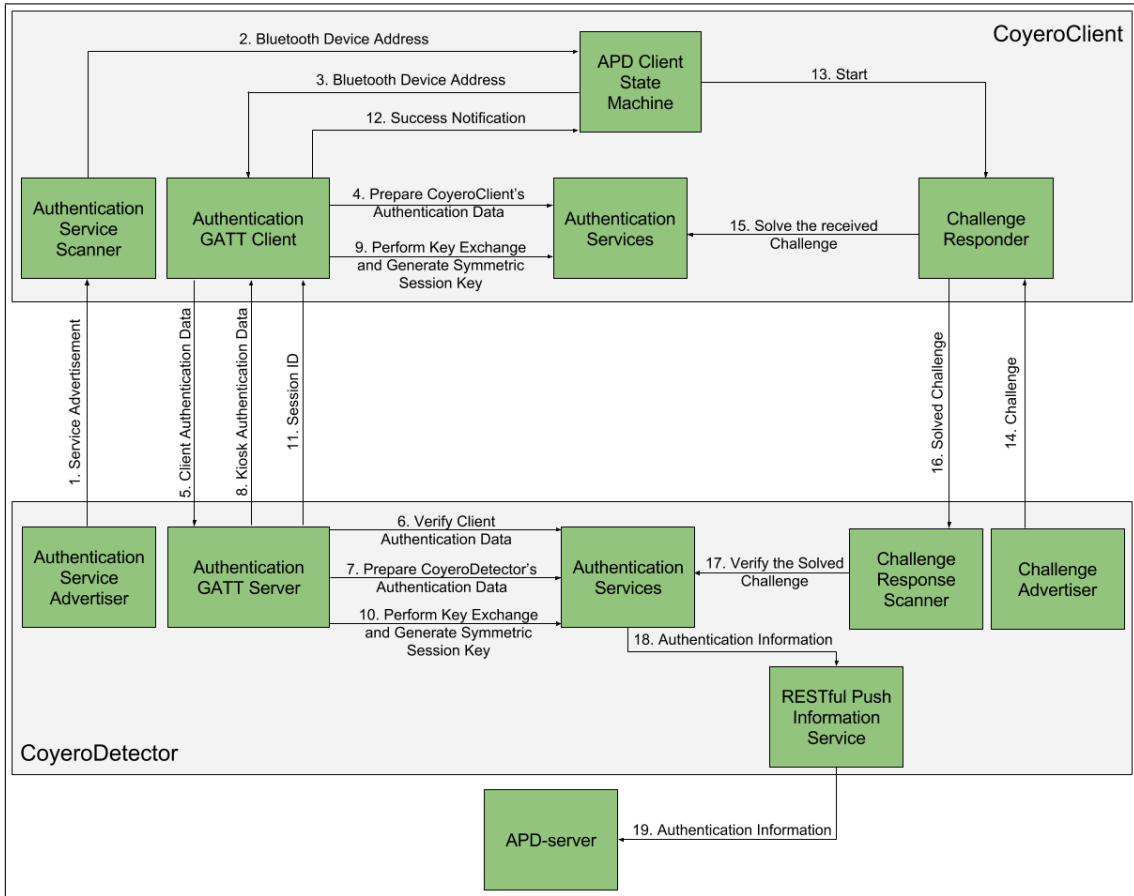


Figure 4.11: Scenario Entering the AoI

4.7 Scenarios

There are two main scenarios, which are important, when considering the use cases presented in 3.1. Both scenarios are presented by using a notation proposed by Kruchten [42] and are as follows:

1. Entering the AoI, which is defined by an Android device running COYERO Detector, with an Android devices running COYERO Client.
2. An Android devices running COYERO Client leaves the AoI of an Android device running COYERO Detector.

4.7.1 Entering the AoI

When COYERO Client enters the AoI of COYERO Detector, both have to perform several tasks in a certain order, for properly performing an authenticated presence detection. These steps are illustrated in Figure 4.11 and are as follows:

1. When a COYERO Client enters the AoI, it is able to receive the Service Advertisement from COYERO Detector’s Authentication Service Advertiser with its Authen-

tication Service Scanner.

2. The Authentication Service Scanner informs the APD Client State Machine about the Bluetooth address of the remote device.
3. Before starting the Authentication GATT Client and delivering the Bluetooth address to it, the APD-client stops the Authentication Service Scanner.
4. The Authentication GATT client uses the Authentication Services for preparing the authentication data of the COYERO Client.
5. The COYERO Client's authentication data is then sent to the Authentication GATT Server, by performing a BLE write operation to a characteristic of the GATT server.
6. The Authentication GATT Server of the COYERO Detector uses the Authentication Services for verifying the received client authentication data, or in other words, for authenticating the COYERO Client.
7. After the COYERO Client has requested the COYERO Detector's authentication data, by starting a GATT read operation, the Authentication GATT Server creates the COYERO Detector's authentication data with the COYERO Detector's Authentication Services.
8. The COYERO Client's read operation is answered by the Authentication GATT Server transmitting the generated authentication data back to the Authentication GATT Client.
9. The Authentication GATT Client verifies the the COYERO Detector's authentication data. Furthermore, it performs the key exchange and generates the symmetric session key by using the Authentication Services.
10. The Authentication Gatt Client performs a BLE read operation, which triggers the Authentication GATT Server to perform the key exchange and session key generation.
11. The BLE read operation is then answered by returning the unique session id to the Authentication GATT Client.
12. The successful authentication process is then notified from the Authentication GATT Client to the APD Client State Machine.
13. The APD Client State Machine starts the Challenge Responder, which results in a BLE scan for challenge advertisements.
14. The Challenge Responder is then able to receive the challenge, which is advertised by the Challenge Advertiser.
15. The received challenge is solved by the COYERO Client's Authentication Services.
16. Afterwards, the solved challenge is advertised from the Challenge Responder and scanned by the Challenge Response Scanner.

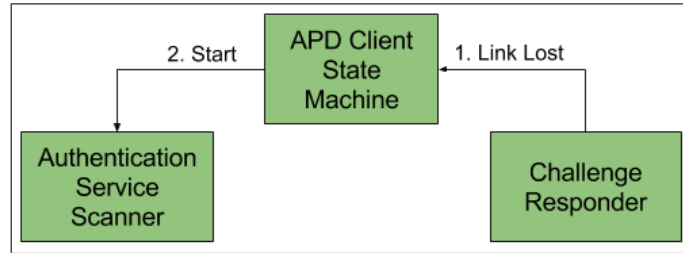


Figure 4.12: Scenario Leaving the AoI

17. The Challenge Response Scanner verifies the solved challenge with the Authentication Services.
18. The RESTful Push Information Service retrieves this successfully performed authenticated presence detection of the COYERO Client from the Authentication Services.
19. This information is finally pushed to the APD Server by the RESTful Push Information Service where it is stored.

4.7.2 Leaving the AoI

When a COYERO Client, which is currently performing the APD challenge response protocol, leaves the AoI, it can no longer detect the challenges, advertised by the COYERO Detector. This leads to the scenario illustrated in [Figure 4.12](#).

1. The Challenge Responder sends a link lost signal to the APD Client State Machine.
2. The APD Client State Machine starts the Authentication Service Scanner.

Chapter 5

Experiments and Results

The implementation presented in [Chapter 4](#), has been tested and analysed by conducting extensive experiments, which are presented in this chapter. Tests were carried out, regarding the overall performance of the implementation. Furthermore, experiments, testing the different parts of the system separately, have been conducted. The results of these experiments, some Android related pitfalls, and device specific behaviour, have been documented and are graphically illustrated for getting insights into the properties and limits of the implementation.

5.1 Components

Several different components were used, for conducting the experiments and collecting the required data. These include not only Android devices, but also BLE USB dongles and a web server.

Android devices There were five Android devices available for testing. These devices are listed in [Table 5.1](#). This is such a low number, because the devices had to support the central and peripheral role of BLE at the same time. Therefore, a minimal Android API of 21, had to be supported. Additionally, Android devices are required to support multiple advertisements, for being able to do advertising at all, as explained in [4.1.2](#). COYERO Detector and COYERO Client, which are presented in [Chapter 4](#), have been installed on all of these devices.

Abbreviation	Vendor	Type	Model	Android Version	API	BT Version
GAS6	Samsung	Galaxy S6	SM-G920	7.0	24	4.1
LGG4	LG	G4	LG-H815	6.0	23	4.1
SONY	Sony	Xperia Z5	E6653	7.0	24	4.1
LGNE	LG	Nexus	5x	7.1.1	25	4.1
HTCN	HTC	Nexus	9	7.1.2	25	4.2

Table 5.1: Android Devices used for the Experiments

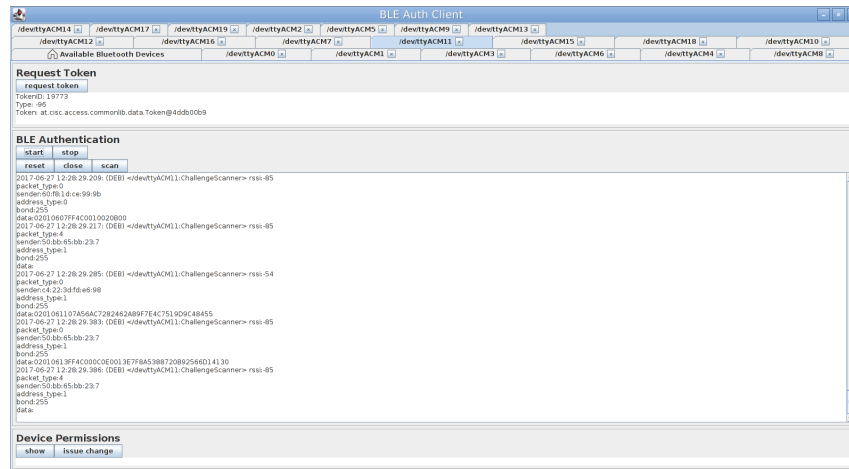


Figure 5.1: Java Swing Tool for controlling BLE112 Dongles

BLE USB dongles As the aim was to test the system under real conditions, simulating additional traffic was required. Therefore, the APD-client software, which was originally developed for Android, was ported, to be usable by another hardware platform as well.

This platform is the BLED112 USB Dongle. These dongles support a virtual COM Port, when connected to a computer, which can be used by programs for controlling the dongles' BLE functionalities. [16]

As the Android code for the APD-client is written in Java, a Java program, which uses the virtual Communication (COM) port of the BLED112, has been written. Only the Android specific parts of the Android Java code had to be rewritten. This included the UI and the BLE API.

For using the virtual COM port of the BLED112 dongles, the library `bglib`¹, has been used. This library wraps the required operations on the COM port and provides a simple API for using the BLE functionalities of the dongles. [12]

An adapted version of this library acted as new BLE API and a Java Swing UI replaced the Android UI. This Java Swing application enables users to control multiple BLED112 dongles at once and it was able to start an APD-client for connected BLED112 dongles. Furthermore, the application supports a feature, which enables starting an APD-client for all connected dongles, one after another. This application has been used in the experiments, for simulating additional APD-clients, which are detected by the COYERO Detector Android application. However, the APD-client implementation, for the BLED112 dongles, has been simplified, meaning there is no automatic restart of the APD-client, when the link to the COYERO Detector application is lost. Furthermore, the verification of the COYERO Detector's authentication token has been removed. A screenshot of the Java Swing application and a picture of the BLE112 dongle experiment setup, can be seen in Figure 5.1 and Figure 5.2, respectively.

Server Apart from the BLE devices, a server was used, which hosted a web service. This web service served as logging endpoint for the participating Android devices. Therefore,

¹`bglib` GitHub project: <https://github.com/SINTEF-9012/bglib>



Figure 5.2: Experiment setup of the BLE112 dongles

Abbr.	Distance
S_d	Almost no distance between COYERO Detector and COYERO Client.
M_d	Roughly half of the L_d distance
L_d	COYERO Client is barely receiving the signals from COYERO Detector.

Table 5.2: Different Experiment Distance Contexts

COYERO Client and COYERO Detector have been extended with a simple web logger module, which uses RESTful calls to store their relevant log and experiment information on that server. This server was first a laptop, locally connected to the network, but later on moved to an Amazon web service.

5.2 Basic Performance Measuring

This has been the most extensive experiment, which was conducted for collecting data, showing the influence of distance between APD-kiosk and APD-client, as well as the influence of other concurrently authenticating APD-clients. Different distance contexts are shown in [Table 5.2](#). These distance contexts are defined without any specifications of distance units, due to the differences in the signal emission strength, which is described in [5.5](#). The number of concurrently participating APD-clients, or in other words the traffic context, is specified in [Table 5.3](#). This traffic is simulated, by using BLE112 dongles. These dongles are not as reliable as the Android implementations. Therefore the two contexts of no traffic and as much traffic as possible, are carried out. The six possible combinations of the distance and traffic contexts, define the six experiment contexts, which have been used within the basic performance measuring experiment.

Each Android device, listed in [Table 5.1](#), has been tested as COYERO Detector, while the other Android devices were running COYERO Client. These tests have been repeated, for all six experiment contexts. However, there are no results for the GAS6, because it does not support the required BLE functionalities, although its data sheet states that it should. This is explained in [5.5](#). Therefore, only results for four Android devices are available.

Abbr.	Traffic
S_t	No other devices than the Android device under test is performing the APD-client's process.
L_t	As much BLED112 dongles as possible (maximum of 20) are performing the APD-client's process during the data measurements. On average there were 15 BLED112 dongles performing the authenticated presence detection concurrently during the tests with a L_t context. The BLED112 dongles are placed right next to the Android device, which was running COYERO Detector

Table 5.3: Different Experiment Traffic Contexts

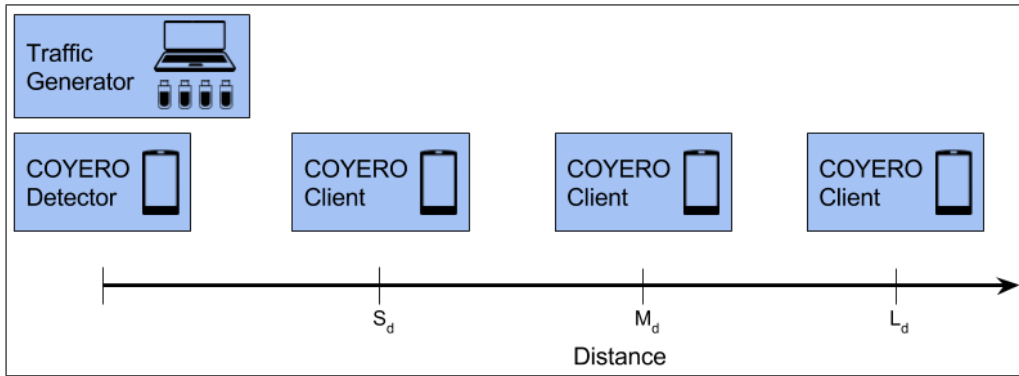


Figure 5.3: Basic Performance Measuring Experiment Setup

The setup of the experiment is illustrated in Figure 5.3. The Android device, which runs COYERO Detector was placed right next to the laptop, which controls the BLE112 dongles. The BLE112 dongles were either active or inactive, depending on current experiment context. The Android devices, running COYERO Client, are placed in position S_d , M_d , or L_d , depending on the current experiment context. Every COYERO Client performed several authentications via the GATT Authentication Service and the authenticated presence detection for about 50 seconds.

5.2.1 Results for the Authentication Process

The authentication is done by the COYERO Client performing seven consecutive tasks. The time each of these steps took has been measured, for showing the impact of each step onto the authentication process. The graph in Figure 5.4 shows the corresponding measurements of the six different contexts for the experiments. This graph contains measurements for all devices. The bars represent averaged times of each step and the averaged time of the complete authentication. The red line indicates the standard derivation of the measured values.

As expected, the time required for a complete authentication, which is shown by the grey bar, is higher for larger distances and when many APD-clients are concurrently active. Taking a closer look to the grey bars, shows that the increasing distance has a bigger influence than the traffic. This can be seen when comparing the contexts $S_t L_d$ and $L_t S_d$ with the baseline context $S_t S_d$. Expanding the distance, increases the time required

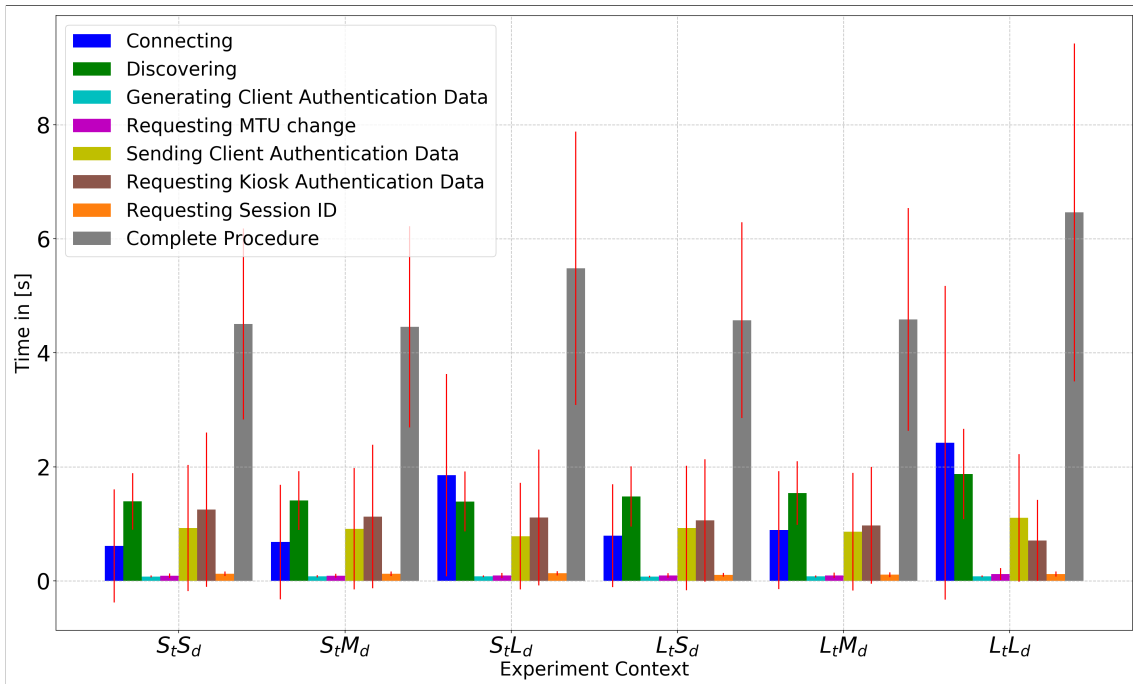


Figure 5.4: Average Times and Standard Deviations of Authentication Steps

for the process, by almost one second, whereas increasing only the traffic has almost no impact. However, the worst performance can be observed in context L_tL_d , which has the highest distance and highest traffic.

The increase of the required time, for performing an authentication process, is mainly due to the connecting step (blue bar). It is the only one, which significantly increases, when the distance and traffic are increasing. The other steps are quite constant in all six experiment contexts. Especially in experiment contexts with L_d , the required time for the connection step increased a lot and the standard derivation is broader as well. This means there is a loss in consistency, when a BLE connection attempt is performed with weaker signals. Furthermore, it can be observed, that steps, which require a lot of BLE operations, like connecting, discovering, sending authentication data, or requesting authentication data, have a broad standard derivation. This indicates inconsistent behaviour, when it comes to performing BLE operations.

There is an additional interesting devices depending behaviour, which is probably existing due to the build in Bluetooth chips, or custom implementations of the Bluetooth software stack for Android. When comparing the SONY device with other Android devices, the SONY device requires about two to three seconds more for transferring data to another device. This is happening even within the best context, namely S_tS_d . Figure 5.5 shows this behaviour by comparing the SONY and HTC devices. When both are acting as APD-kiosk, there is a huge difference in the step, which requests the kiosk authentication data. The SONY device requires much more time for sending the data back to the APD-clients, than the HTC device does. On the other hand, comparing both devices, when they act as APD-client, shows that the SONY device requires more time for the step, in which the client authentication data is send to the APD-kiosk. This lets conclude, that

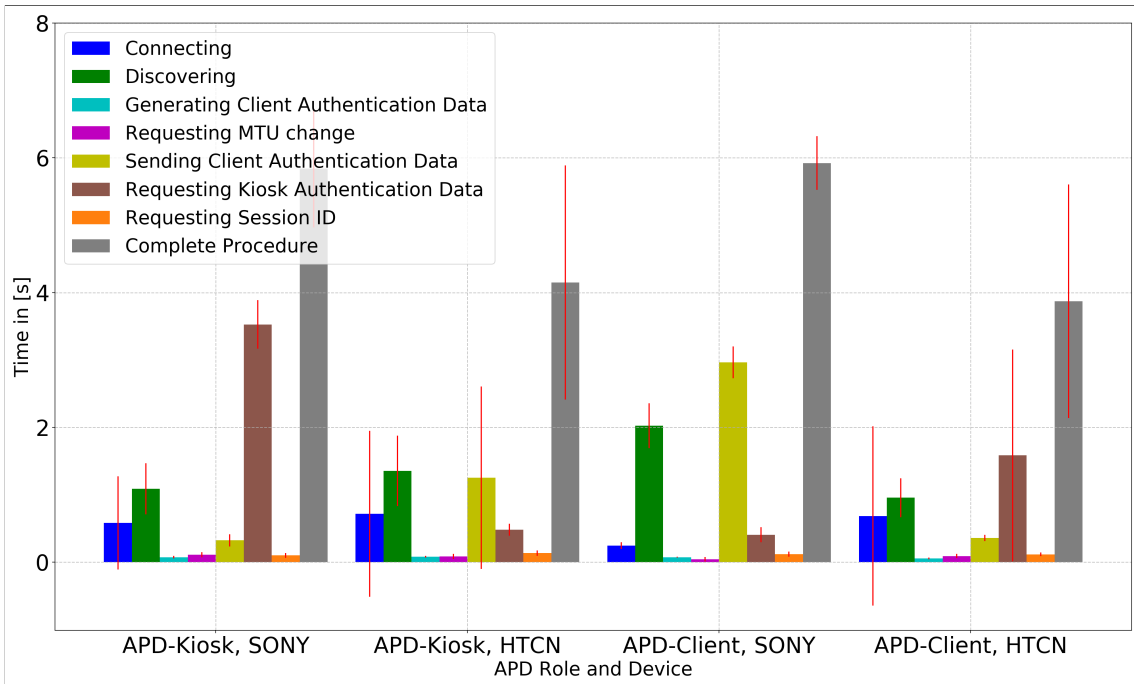


Figure 5.5: SONY Performance compared to HTCN Performance

the SONY device is slower than the others, when it comes to sending data to other devices, using BLE.

The previously shown graphs only include successful attempts of the specific authentication steps. But there were failing attempts as well. Figure 5.6 includes the measurements of all devices and shows the error ration of the authentication steps. The connection step, represented by the blue bar, failed the most during the experiment. The discovering step and the request MTU step have failed very rarely. All other steps did not fail at all during the experiment. Failing means, that the timeouts of the steps have been reached. These timeouts are explained in 4.4. This lets one conclude, that an established connection is very stable, however establishing one is inconsistent. This is another indicator for the connection step being the most fragile part of the authentication process. Additionally, this graph shows that an increased traffic does not make the connection step slower, as it is the case for an increased distance, but rather prevents it from being successful.

5.2.2 Results for the APD Challenge Response

For the APD challenge response, the delay between successive successful authentications has been measured. Meaning the frequency of the authentications, received by the APD-kiosk from an APD-client. The measured values are mainly the delays between successive scans of challenge response advertisements from APD-clients. However, there is one exception. The first successful authentication is the verification of the certificates, during the authentication process. Therefore, the delay between the successfully finished authentication and the first received challenge response advertisement, is included in the analysis as well. This first measured delay, includes the start up of the whole challenge response

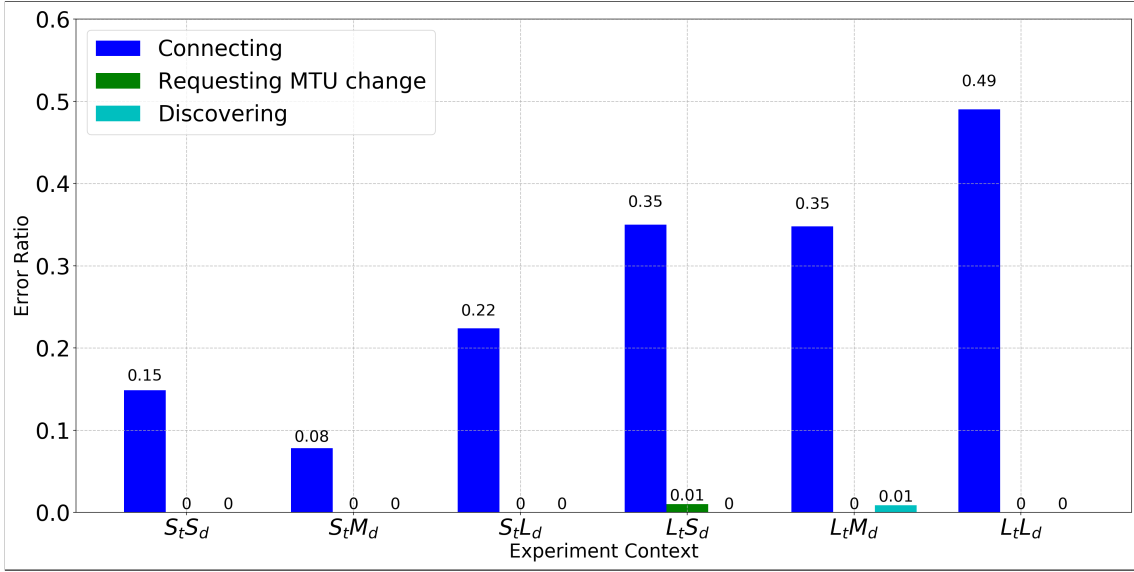


Figure 5.6: Error Ratio of Authentication Steps

BLE system and is therefore longer than the others. The analysed data is visualized in Figure 5.7. This graph contains three bars for each experiment context. The bars contain following information:

1. **Initial Delay:** Represents the average of all measured initial delays. An initial delay is the time between a successful authentication and the scan of the first challenge response advertisement.
2. **Challenge Response Delay:** This is the average of all delays between successive challenge response scans for the same APD-client.
3. **All Delays:** Represents the average of all initial delays and all challenge response delays.

Additionally the red line, which is visible for all the bars, shows the standard derivation and serves as indicator for consistency.

The distance has almost no impact onto the performance, when no traffic is active. All experiment contexts with S_t have a similar performance. When increasing the traffic, the distance has an impact. Comparing $L_t S_d$ with $L_t L_d$ shows an increase of the average delay about 100%. The standard derivation is quite high for all measured data, which indicates inconsistent delays. This could lead to wrong absence assumptions of APD-clients. The increase of the delays for $L_t L_d$ is a result of the bad performance of certain devices. Comparing the LGNE device, acting as APD-kiosk, with other devices acting as APD-kiosk, shows that the LGNE device performs terribly in the $L_t L_d$ experiment context. This behaviour is shown in Figure 5.8 and indicates that the LGNE device has troubles with finding newly started advertisements in the $L_t L_d$ context.

All together, it seems that finding newly started challenge response advertisements, is the task which takes the longest. As this is somehow comparable to the bad performance

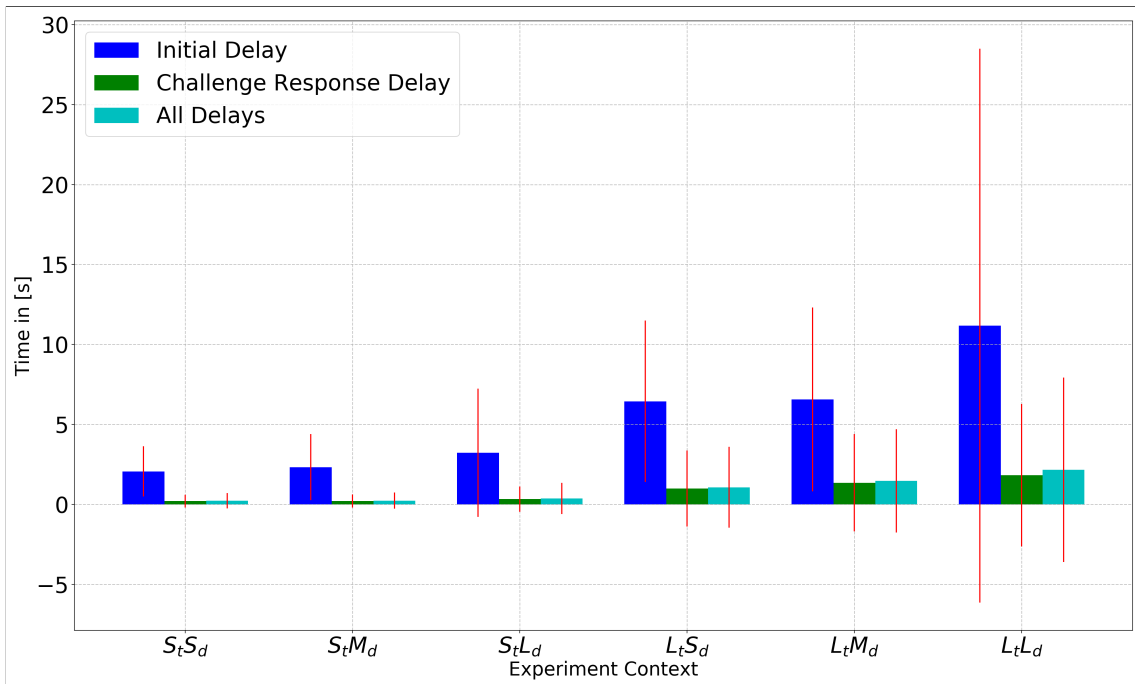


Figure 5.7: Average Authentication Delay of all Measurements

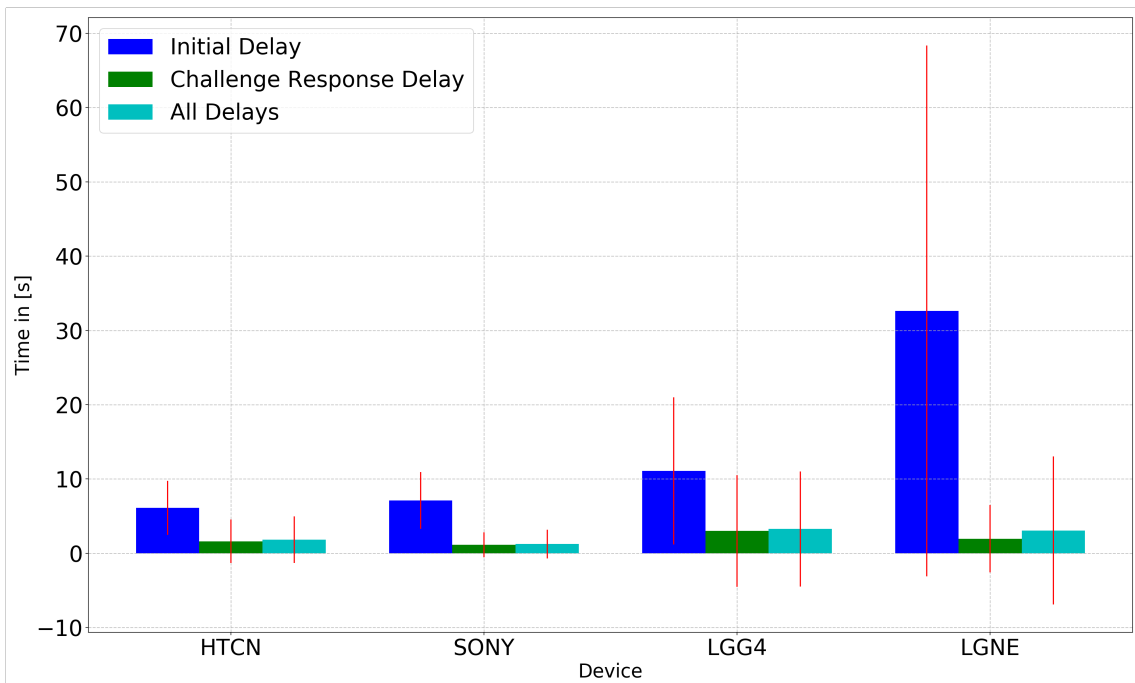


Figure 5.8: Bad Performance of LGNE in L_tL_d compared to other Devices

of the initial step of the authentication, one could conclude, that the bottleneck is the start up of a BLE communication. After this has been done, BLE performs quite well.

5.2.3 Findings

The most important finding is that the implementation works correctly. Android devices, which are running COYERO Client and enter the AoI, defined by an Android device running COYERO Detector, are detected correctly. For all tests within an L_d context, the entering was performed, by physically moving the Android device with an active COYERO Client, into the AoI. For all S_d and M_d contexts, the entering was simulated by activating COYERO Client on Android devices, which were already placed inside the AoI.

Furthermore, it has been shown, that there are device dependencies, when it comes to performance. Different devices running COYERO Client and COYERO Detector perform differently. This has been shown for the SONY device in the authentication steps and the LGNE device in the APD challenge response process, as seen in [Figure 5.5](#) and [Figure 5.8](#), respectively.

The bigger the distance between the COYERO Detector and the COYERO Client, the worse the performance. Additionally, the higher the number of participating APD-clients, the worse the performance. However, it is noteworthy, that the initial steps of the authentication and the authenticated presence detection, meaning the connection step and the recognition of the first challenge response advertisement, are the biggest contributors to the bad performance.

It is not possible to control the number of APD-clients within the AoI of the COYERO Detector, but it is possible to limit the bad impact of the L_d context, by limiting the AoI with a signal strength based threshold. Defining a threshold based on a certain dBm value, would eliminate the L_d context and therefore simulate the border of the AoI with the M_d context. The results of the measurements performed in M_d are much better, than those measured in L_d , which can be seen in the high error rates in [Figure 5.6](#) and the bad performance of devices as visualized in [Figure 5.7](#). This could be done in software, by only reacting to advertisements, with a signal strength bigger than the defined dBm threshold.

5.3 Simultaneous Entering devices

As the Authentication GATT Server is not connectable during an ongoing authentication (see [4.3](#)), the behaviour of simultaneously entering devices had to be analysed. The experiment was conducted with the distance context M_d and both traffic contexts. This decision has been made, due to the findings in [5.2.3](#). Every Android device was running COYERO Detector once, while the other three Android devices were running COYERO Client. All Android devices, running COYERO Client, have started the APD-client's process simultaneously and the time between the first COYERO Client application starting the authentication and the last one finishing the authentication has been measured. This was repeated several times.

The outcome of this experiment can be seen in [Figure 5.9](#). The bars show the average of the measured times, required for detecting and authenticating all three COYERO Client applications. The blue bars contain measurements, done in the context S_t , whereas the

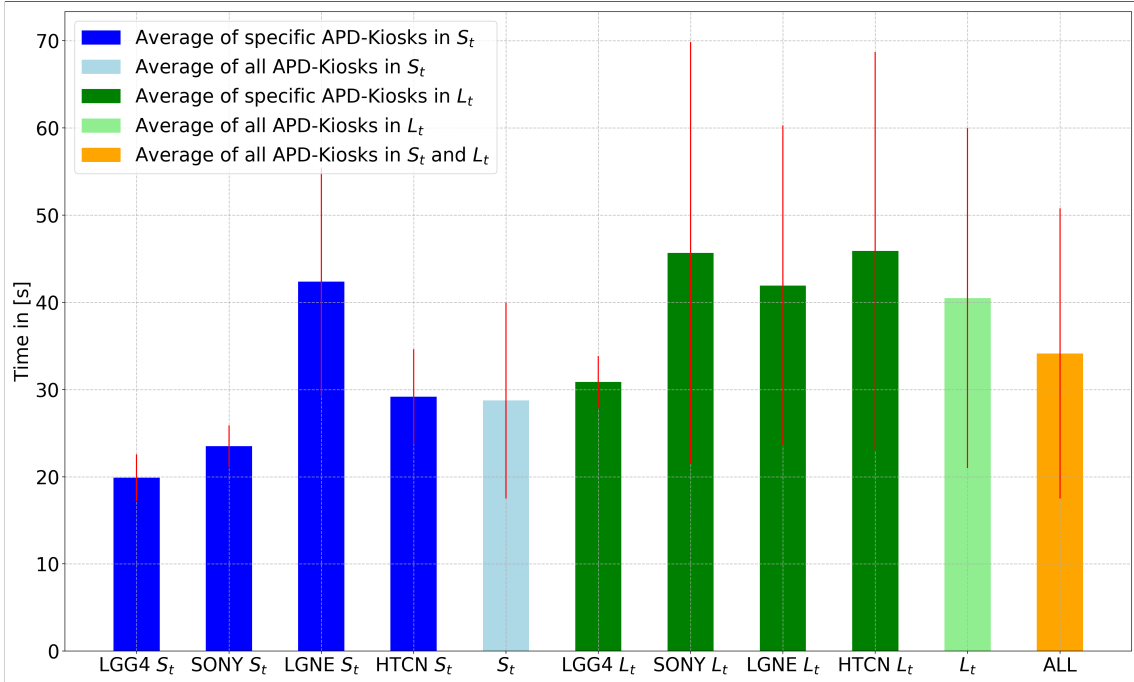


Figure 5.9: Average Times of simultaneously entering Devices

green bars contain the ones done in L_t . Dark blue and dark green bars contain only measurements of specific devices running COYERO Detector. Light blue and light green are the averages for all devices in the specific traffic contexts. The orange bar is the average of all measurements and the vertical red lines show the standard derivations, which indicate the consistency of the measured timings.

Comparing these measurements with the ones from Figure 5.4 shows that the simultaneous entering, of Android devices running COYERO Client, adds a lot of timing overhead. This is due to the high timeout of the connection step in the authentication process. When the COYERO Detectors’s authentication service advertisement is received by multiple COYERO Client applications at the same time, all of them start the connection step of the authentication process. This could lead to COYERO Client applications, getting stuck in the connection step until they time out, because they assume that they are performing a connection attempt, although the COYERO Detector application has already turned off the connectable advertisement and does not react to further connection attempts. This ten second timeout, plus the artificial delay before restarting the authentication service advertisements, lead to this overhead.

5.4 System Test

This test has been conducted, for analysing the overall performance of the implementation, when being active for about 15 minutes. The contexts varied during the experiment. The only context, which has not been tested, due to the findings in 5.2.3, was L_d . Each Android device was running COYERO Detector for 15 minutes with the other Android

devices running COYERO Client. Therefore, the whole process was performed four times. Devices running COYERO Client, were moved within the S_d and M_d contexts, meaning they constantly switched their physical position. The traffic was increased slowly from S_t to L_t during the experiment. Due to the instability of the BLE dongles (see 5.1), the traffic drops and raises within a certain range.

5.4.1 Authentication Frequency

The authentication frequency is a function, which shows the number of successful authentications and received correctly solved challenge response advertisements within a given interval, received by an APD-kiosk, for a particular APD-client. Comparing this authentication frequency to the information of how many devices are currently participating in the APD-process, leads to the visual representation in Figure 5.10. Figure 5.10 contains two graphs for each Android device. The first graph contains a function in black, which indicates the number of currently participating APD-clients over time. The second graph contains the authentication frequency function, for an interval of 25 seconds, of the three participating Android devices, which run the COYERO Client application.

The first thing that can be seen is, that the authentication frequencies drop, when the number of participating APD-clients increases. This is true for all Android devices running COYERO Detector. The three authentication frequencies, for individual devices, running COYERO Detector, look quite similar. However, comparing the authentication frequencies for different devices, which run COYERO Detector, there is a large gap between the these measured frequencies observable. The SONY device, running COYERO Detector, achieves, with the participating COYERO Client applications, a maximum authentication frequency of 125, whereas the authentication frequency maxima, of the COYERO Client applications, are values between 25 and 40, when the LGNE device is running COYERO Detector. Additionally, the function which indicates the number of currently participating APD-clients, is much more jittery for the LGNE, running COYERO Detector, than for the other devices. Furthermore, the received authentication frequency is for the LGNE quite often zero. This lets conclude, that some Android devices are better for acting as a APD-kiosk, via the COYERO Detector application, than others.

5.4.2 Absence Timeout

Finding an absence timeout for the APD-kiosk has been another aim of this experiment. The authentication frequency of the LGNE drops to zero several times, as seen in Figure 5.10. However, it does not drop to zero for other devices, which lets assume potential different optimal absence timeouts for different devices acting as APD-kiosk. Analysing the collected data regarding absence timeouts, confirmed this assumption.

As the Android devices, which run COYERO Client during the experiments, are always within the AoI of the COYERO Detector application, they should be detected as present all the time. This means, the COYERO Detector application shall receive a challenge response advertisement, for each Android device running COYERO Client, within the absence timeout. This is always the case, when the absence timeout is high enough. Some of the collected data is visualized in Figure 5.11, Figure 5.12, Figure 5.13, and Figure 5.14. Within these visualizations, a drop of a line, from ✓ to ✗, means that the absence timeout

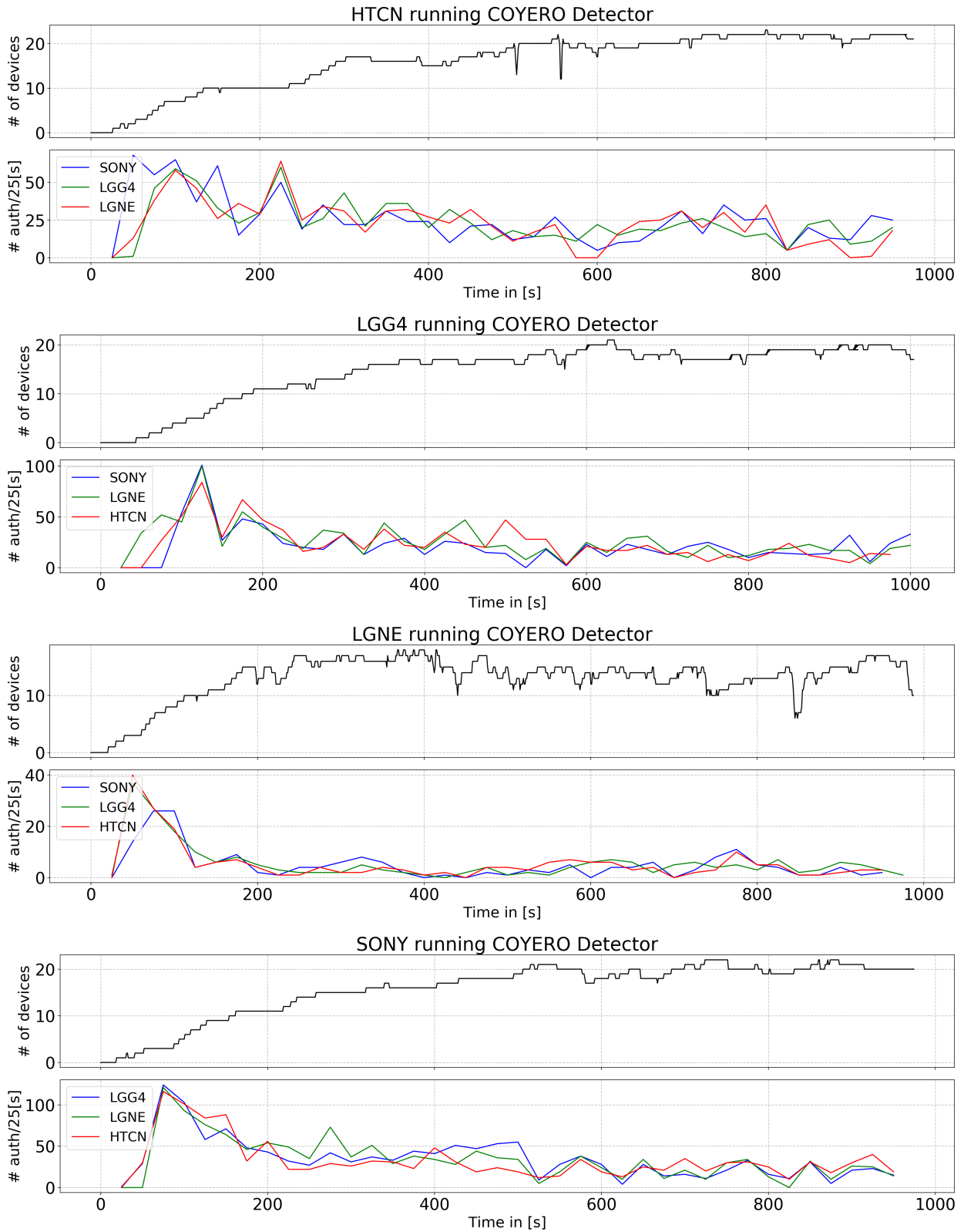


Figure 5.10: Number of currently participating APD-clients and Authentication Frequencies of the Android Devices running COYERO Client for the different Android Devices running COYERO Detector

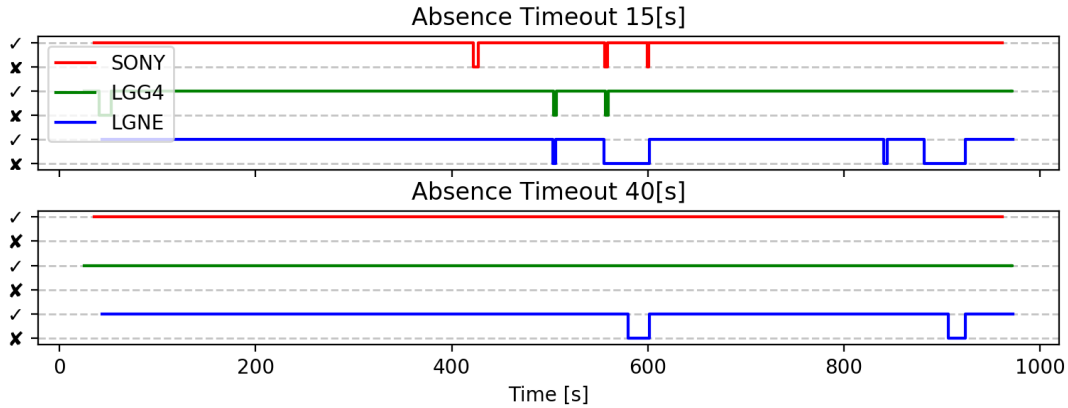


Figure 5.11: HTCN Absence Timeout Analysis

Device running COYERO Detector	LGG4	SONY	LGNE	HTCN
Optimal Absence Timeout	32 sec	43 sec	57 sec	62 sec

Table 5.4: Optimal Absence Timeouts for the Devices

has been reached by the corresponding Android devices at this point in time. The next scanned challenge response, of the corresponding Android devices, was received at a raise from \times to \checkmark . The optimal absence timeouts of the devices, shown in Table 5.4, are the absence timeouts in seconds, which have never been reached by any of the participating Android devices, running COYERO Client.

The tests of different absence timeouts for the HTCN device, as shown in Figure 5.11, highlight the bad performance of COYERO Client running on a LGNE device. The optimal absence timeout for the other two devices is 28 seconds. However, the LGNE pushes the optimal timeout up to 62 seconds, making it the highest of all computed optimal absence timeouts. The performance of the LGNE, running COYERO Client, for other Android devices, running COYERO Detector, is quite normal, as seen in Figure 5.14 and Figure 5.12. Its performance, when running COYERO Detector, on the other hand, is the worst of all four devices. It does not have the highest optimal absence timeout, but has the lowest authentication frequencies and therefore the highest number of absence timeout exceedances, when assuming a smaller absence timeout. It can not be guaranteed, that any absence timeout is able to detect the absence of devices with 100% accuracy. However, the detection rate seems to be very reliable with values for the absence timeouts, bigger than 60 seconds.

5.5 Unexpected Findings

During the implementation and the experiment conduction, some unexpected findings have been made. These are mostly related to the different devices and the usage of Android. These findings are explained within the following paragraphs.

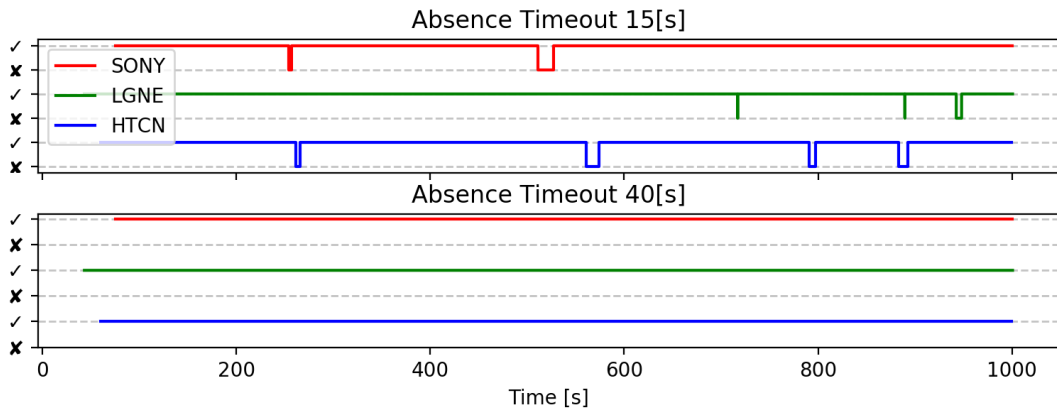


Figure 5.12: LGG4 Absence Timeout Analysis

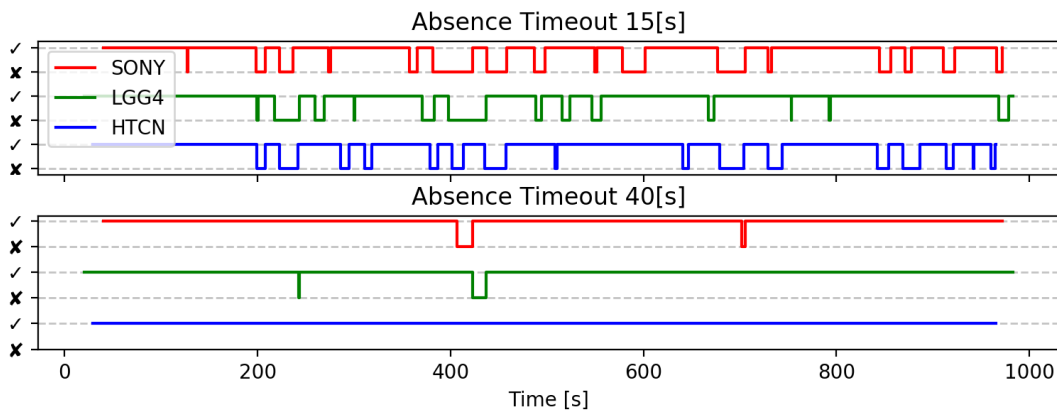


Figure 5.13: LGNE Absence Timeout Analysis

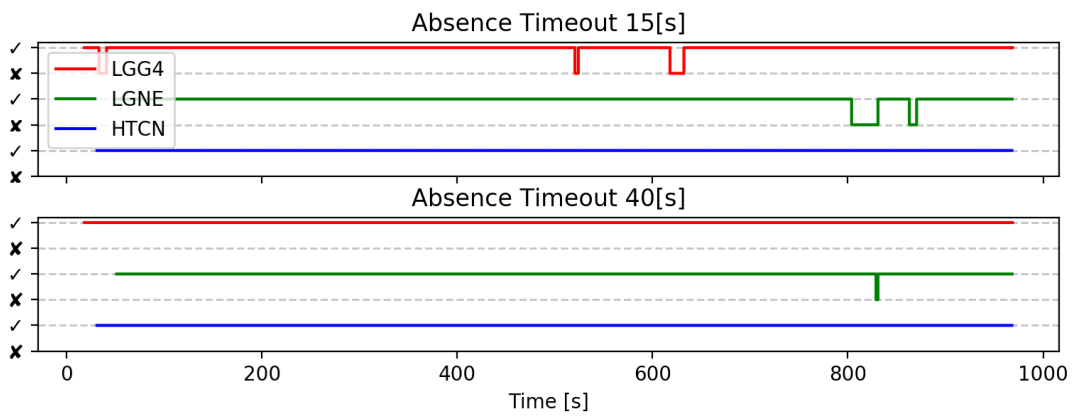


Figure 5.14: SONY Absence Timeout Analysis

Device	Max. Number of simultaneous Advertisements	Max. Number of simultaneous Connections to GATT clients
LGG4	4	5
SONY	4	5
LGNE	15	3
HTCN	4	5

Table 5.5: Device specific Maxima of simultaneously performed BLE Operations

Device specific BLE capabilities: The first thing that has been observed is that devices have different properties in regards to the numbers of simultaneously performable BLE actions. Simultaneously performed advertisements have been analysed as well as the maximal number of concurrent connections to different GATT clients for a GATT server. These are not the same numbers for all Android devices. The devices, which were tested after discovering differences, and the corresponding values are listed in Table 5.5. The LGNE is able to perform 15 simultaneous BLE advertisements, whereas the others can only perform four. When acting as a GATT server, the LGNE does only support three simultaneously active connections to GATT clients. A fourth GATT client can still receive the connectable advertisement of the LGNE, however is not able to establish a connection to it. The other three devices support five ongoing simultaneous connections to GATT clients. A sixth connection is not possible, because the advertisement, which has been used for establishing the connections, is terminated, when five connections are open. This makes it hard to implement optimized applications, because using the supported features of one device to optimize a procedure, makes potentially no difference on other devices.

Samsung Galaxy S6 does not work when running COYERO Client: A very unexpected behaviour has been shown from the Samsung Galaxy S6, when it is running the COYERO Client application, although it fulfils all BLE requirements for implementing the authenticated presence detection, according to its data sheet. [56]

The tests for the Samsung Galaxy S6, running COYERO Detector, have been successful. However, when running the COYERO Client application, it is only able to perform the authentication. After successfully authenticating as GATT client, by using the Authentication GATT Service, it switches to the APD challenge response protocol and starts scanning for challenge advertisements, of the device running COYERO Detector. However, it never receives any advertisements. It has not been clarified why it does not receive this advertisement, because the COYERO Client application works fine on all other tested Android devices. The first assumption was that the initially installed Android version 5.1.1 caused the wrong behaviour. However, even updating the Samsung Galaxy S6 to Android Version 7.0 did not solve the problem. Therefore, the Samsung Galaxy S6 has not been used for any experiment.

Skip scan of challenge responses: The HTCEN device, running COYERO Detector, does not receive challenge responses from its performed BLE scans, while processing GATT operations, as used done during an authentication. When many APD-clients want to establish a connection simultaneously, they used to be processed one after another, by the

COYERO Detector application, without any delay between finishing one authentication process and starting the next one. This resulted in exceeding the absence timeout of APD-clients, when many consecutive authentication processes of APD-clients have been performed. This behaviour was observed, when many BLE112 dongles have been used during the experiments. This problem has been solved, by adding a delay of three seconds between two successive authentications. This is long enough for the HTC device to scan the challenge response advertisements of surrounding APD-clients. As the authentication service advertisement is stopped, when a device has established a connection and restarted, when the device has been disconnected, the easiest way to implement the delay, was by delaying the restart of the authentication service advertisement.

Limitations in data transfer size: While executing some performance tests with a huge amount of data transferred during the authentication process, by using a GATT server, the data has not been received correctly. Therefore, investigations regarding this behaviour were carried out with the Android application nRF Connect for Mobile².

Read and write GATT operations have been performed. The LGG4 was used for being the GATT client and the LGNE was the GATT server. It was able to write and read up to 600 bytes correctly. Writing or reading data, which had more than 600 bytes, resulted in receiving only a truncated data stream of 600 bytes. Testing a read of 4500 bytes, lead to a restart of the LGNE's Bluetooth adapter, due to an occurring error.

Therefore an additional functionality had to be added to the implementation. Every data, which had to be transferred with a read or write GATT operation, was split up into chunks small enough, that they fit into one read or write operation. For a GATT read operation this is MTU-1 bytes and for a GATT write operation MTU-3 bytes, as stated in the *Specification of the Bluetooth System 4.1*. [61, Vol. 3, p. 570, p. 574]. Each chunk is then written or read one after another to prevent the previously mentioned errors.

Signal strength differences: During the conduction of the experiments the authenticated presence detection has been aborted unexpectedly. This was due to differences in the BLE signal strengths of the devices, although they were initialized with the same BLE signal strength settings. For highlighting the differences, an additional experiment was conducted. Four devices were placed at fix positions and pairs of them have been tested. Two devices started to advertise and scan at the same time with the same signal strength settings for a period of roughly two minutes. The scanned signals, including their strengths, have been collected, averaged, and visualized. The expected outcome would be, that both participating devices, receive the signal of the other device, with approximately the same signal strength. However, the differences of the received signal strength is unexpectedly high for the SONY device. Those differences are visualized in [Figure 5.15](#). The bar plot shows the six different combinations of the four devices, which were tested on the x-axis, and the measured dBm difference on the y-axis, for all four possible settings in Android. The black dashed line indicates the average value of all four Android settings combined. This graph highlights the higher signal strength of the SONY device, as all conducted tests, where the SONY is participating, show a higher difference than

²nRF Connect for Mobile is an Android application, which enables performing BLE operations, like advertising, scanning or GATT connections. [51]

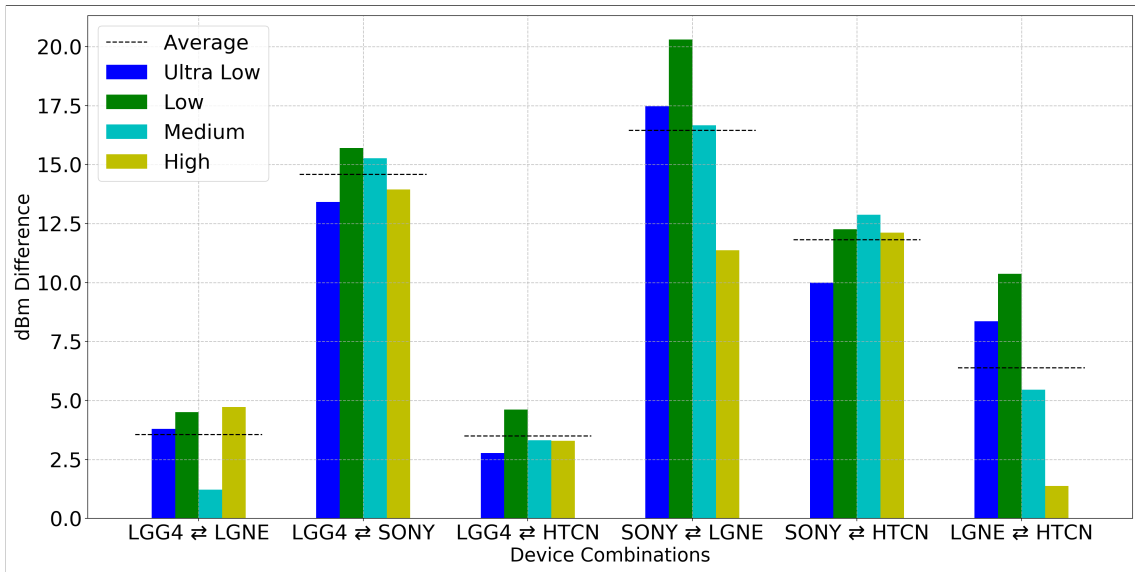


Figure 5.15: Signal Strength Differences with same Settings

the others. The huge difference of the SONY device, required a different setting for the conducted experiments, as otherwise the APD-process would not work properly. The data was collected with the Android application nRF Connect for Mobile³.

Power consumption: Android avoids draining the battery by sending idling devices into a sleep mode, which reduces the CPU usage to a minimum. There are cases where a task is required to be done as fast as possible. Therefore, an Android device can be prevented from switching to the sleep mode by acquiring a so called wake lock. When a wake lock is acquired, the CPU continues the computations until the wake lock is released again. [41]

The background service of the COYERO access Client SDK extension, which starts and performs the APD-client’s functionalities, used to acquire such a wake lock immediately after it has been activated, even when there is no APD-kiosk for performing an authenticated presence detection available. The recognition of APD-kiosks is done by continuously scanning for authentication service advertisements. It has been verified that the BLE scan works, even when no wake lock is acquired. Therefore, acquiring the wake lock after recognizing a APD-kiosk has been implemented to save battery. This has been tested on the LGNE, HTCN, and LGG4. All tested devices were still able to start and perform the APD process correctly.

³nRF Connect for Mobile is an Android application, which enables performing BLE operations, like advertising, scanning or GATT connections. [51]

Chapter 6

Conclusion

BLE, as state of the art wireless technology, is more and more used for presence detection systems. The low energy nature of BLE and its different modes for data transfer, makes it adaptable for different use cases in the field of presence detection. Many researchers are already focusing on building applications based on presence detection by using BLE, or extend their research with information gathered from it. However, none of them focuses on a proper cryptographic authentication within their presence detection.

The proposed design addresses this missing authentication. It uses a basic concept for detecting the presence of devices, which is inspired by works of other researchers. The innovative part of the design, is combining presence detection via BLE with a cryptographic authentication based on a PKI. This enables not only detecting the presence of devices continuously, but also authenticating them with every update of their presence. The design shows, that it is important to use certain communication modes of BLE for certain cryptographic operations. For performing a DH key agreement via BLE, a GATT connection between two devices is favoured, because a lot of data is required to be transferred. On the other hand, for executing a challenge response, utilizing BLE broadcasters and observers is recommended, because it increases the performance of the challenge response procedure and reduce the required computational resources of devices.

The design has been verified by implementing it for Android devices. This implementation provides all properties to be a reasonable extension for the COYERO access environment. The provided implementation uses two Android applications, namely COYERO Detector and COYERO Client. COYERO Detector defines the detection area, whereas COYERO Client has to be installed onto users mobile devices, which enables COYERO Detector noticing this mobile devices. Both use the COYERO access infrastructure to authenticated each other during the presence detection. An optional feature of the COYERO Detector can be activated, which sends information, about the currently present and authenticated devices, to a web server.

This implementation has been evaluated by conducting extensive experiments. The experiments showed that it is possible to achieve authenticated presence detection by only installing applications onto Android devices. The information, whether particular people are present, is updated at least every minute in a real life scenario, enabling potential service providers to react to this information within a reasonable time. Additionally, it has been shown that the variety of devices, which are available for Android, could be a problematic factor. Some of the devices perform worse than others or even do not support

High Level Objectives	Gateway	Up-to dateness	Technology	Additional HW required		Authentication Level
				RN	SN	
Geographical Presence	LCN	low - high	BLE	no	no	Authentication

Table 6.1: Embedding of the APD Design

the required BLE features, although they should according to their data sheets. This would lead to a lot of effort when it comes to building a system for commercial use.

6.1 Comparison to other Researches

An evaluation based on the conducted experiments, which addresses the identified common properties of the related work (see [Table 2.2](#)), is provided in [Table 6.1](#). This table embeds the presented design and its implementation into the current research field of presence detection.

- **High Level Objectives:** The high level object is detecting the *Geographical Presence* of devices.
- **Gateway to other Channel:** It uses the COYERO Detector application, or in other words the *LCN*, to forward the information of surrounding COYERO Client applications, to a web server. However, as the implementation is purely done in Android, the devices running COYERO Client, would easily be able to forward the data as well.
- **Up-to-dateness:** This was heavily focused by one experiment, which is shown in [5.4.2](#) and is related to the absence timeout explained in [3.4.4](#). The experiments showed an update frequency about one minute. Therefore the up-to-dateness is assigned to *low-high*.
- **Wireless Technologies:** Like most of the other presented presence detection approaches in [2.3](#), the newly created design uses *BLE* as well.
- **Additional Hardware:** Android devices are usually available, and therefore are not considered as additional hardware.
- **Authentication Level:** As the aim of this work was to provide an authenticated presence detection, this is the most important property of the new design's implementation. It supports a cryptographic *authentication* in the context of a presence detection, which none of the other presented works in [2.3.1](#) does.

6.2 Future Work

The most important objective for future work is to try splitting the work load of the presence detection to multiple devices, meaning having two or more devices, which can

be used for detecting surrounding people. This would not only increase the performance, but also open up the possibilities, for deriving information about the detected people's position within the detection area.

Furthermore, it is planned to exchange the COYERO Detector Android application with a COYERO communication controller¹. This would not only get rid of device dependencies on the APD-kiosk side and potentially increase the performance and stability, but also lead to a fixed installation of the APD-kiosk, which may be the preferred solution in many applications.

Finally, it is worth mentioning, that optimizations in the Android implementation itself will be carried out, like optimizing the timeouts of the GATT authentication process and adding a threshold for the signal strength to prevent the bad performance, at the border of the detection area.

¹Information about the COYERO communication controller can be found at <http://www.coyero.net/coyero-access>

Appendix A

Custom Packet Format

This chapter provides additional information for the custom packet format, which enables sending multiple parameters in one data block. This is an easy format that shall be used for packing the different parameters.

Challenge Packet

This packet contains the challenge, which is required to be solved by the receiving entities. [Figure A.1](#) shows the format for transmitting challenges. The first two bytes are the identification header of the packet, which is in the case of a challenge packet a 0x0001, followed by 16 bytes containing the challenge.

Challenge Response Packet

The challenge response packet combines the solved challenge with the session identifier. This is illustrated in [Figure A.2](#). The first two bytes are the identification of the challenge response packet, which is 0x0002, followed by four bytes containing the session id and then 16 bytes containing the solved challenge.

Authentication Data Packet

This packet is used for transferring the data, which is required for performing an authentication and a key exchange. It combines three different parameters into one block. These three parameters are as follows:

- **Certificate data:** This certificate data is required to authenticate the entity, which wants to be authenticated. It is a certificate issued by a trusted CA.

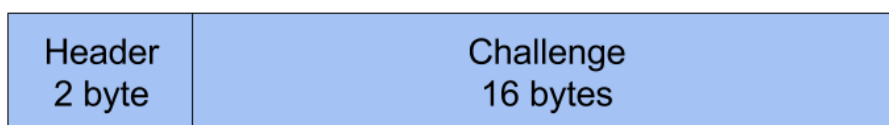


Figure A.1: Challenge Packet Format



Figure A.2: Challenge Response Packet Format

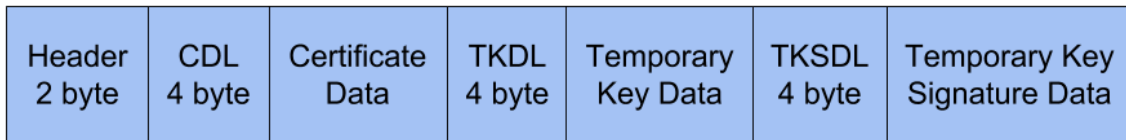


Figure A.3: Authentication Data Packet Format

- **Temporary key data:** This is the temporary public key of a key pair, which shall be used for performing the key exchange.
- **Temporary key signature data:** The signature is created by using the private key of the certificate and is computed for the temporary key data to ensure that it has not been tampered with.

To transmit these three parameters, the structure shown in [Figure A.3](#) shall be used. The authentication data packet starts with a two byte identification header, which is 0x0003, followed by the certificate data the temporary public key data and the signature of the temporary public key. All of the three appended parameters, have a leading four bytes, which define the length of the parameter. CDL defines the length of the certificate data, TKDL the length of the temporary key data, and TKSDL the length of the temporary key data's signature.

Appendix B

APD Server RESTful Definition

This chapter provides a definition of the APD Server's RESTful interface, which can be seen in [Table B.1](#). There are two different types of JSON files used within the interface. The first one contains the information of the authenticated and present APD-clients, detected by the APD-kiosk. The JSON parameters of this information can be interpreted as follows:

- **update_time**: Contains the time, when the APD-kiosk has sent this update to the server. This is date is defined by the APD-kiosk, to prevent wrong timing information due to network latency.
- **device_auth_information**: List of present and authenticated APD-clients, which can contain 0 to n entries. One entry contains following elements:
 - **remaining_time**: The remaining time in seconds before the APD-client is considered to be gone.
 - **token_id**: Unique id, of the authentication token from the COYERO access framework, to identify the APD-clients.
 - **session_id**: The unique session id.

The second JSON definition adds information, which is interpretable by users, to the authentication information of APD-clients. This separation enables the retrieving of readable information from other sources as well. It is a list which contains 0 to n device information entries. The JSON keys of the device information entries can be interpreted as follows:

- **device_name**: Human readable string representation of an APD-client.
- **token_id**: The unique id of the authentication token from the COYERO access framework, which links the readable string to the authentication information.

Update Authentication Information	
RESTful	PUT /
Body	<pre>{ "update_time" : "22/08/2017 @ 12:19:20.787", "user_auth_information" : [{ "remaining_time" : 178, "token_id" : 27711, "session_id" : 2857688042 }, { "remaining_time" : 177, "token_id" : 27716, "session_id" : 1579531574 }] }</pre>
Returns	HTTP Status 200 OK
Get Authentication Information	
RESTful	GET /authentication_devices.json
Returns	<pre>{ "update_time" : "22/08/2017 @ 12:19:20.787", "user_auth_information" : [{ "remaining_time" : 178, "token_id" : 27711, "session_id" : 2857688042 }, { "remaining_time" : 177, "token_id" : 27716, "session_id" : 1579531574 }] }</pre>
Get Device Information	
RESTful	GET /device_information.json
Returns	<pre>[{ "device_name" : "Nexus_9-ca56ed4f005e5488", "token_id": 27711 }, { "device_name" : "Nexus_5X-88dcbc0eb34f7f46", "token_id": 27716 }]</pre>

Table B.1: RESTful Specification of the APD Server's Interface

Bibliography

- [1] *AdvertiseData*. URL: <https://developer.android.com/reference/android/bluetooth/le/AdvertiseData.html> (visited on 10/08/2017) (cit. on p. 47).
- [2] *AdvertiseSettings*. URL: <https://developer.android.com/reference/android/bluetooth/le/AdvertiseSettings.html> (visited on 10/08/2017) (cit. on p. 47).
- [3] A. Alhamoud et al. ‘Presence detection, identification and tracking in smart homes utilizing bluetooth enabled smartphones’. In: *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks Workshops*. LCN ’14. Edmonton, Canada, 2014, pp. 784–789. DOI: [10.1109/LCNW.2014.6927735](https://doi.org/10.1109/LCNW.2014.6927735) (cit. on pp. 19, 25).
- [4] *Android Dashboards*. URL: <https://developer.android.com/about/dashboards/index.html> (visited on 10/08/2017) (cit. on pp. 50, 54).
- [5] *Android Developers*. URL: <https://developer.android.com/index.html> (visited on 10/08/2017) (cit. on p. 45).
- [6] *Android Studio*. URL: <https://developer.android.com/studio/index.html> (visited on 10/08/2017) (cit. on p. 45).
- [7] *ART and Dalvik*. URL: <https://source.android.com/devices/tech/dalvik/> (visited on 10/08/2017) (cit. on p. 45).
- [8] Bharathan Balaji et al. ‘Sentinel: Occupancy Based HVAC Actuation Using Existing WiFi Infrastructure Within Commercial Buildings’. In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’13. Roma, Italy, 2013, 17:1–17:14. DOI: [10.1145/2517351.2517370](https://doi.org/10.1145/2517351.2517370) (cit. on pp. 18, 25).
- [9] Ankit .S. Barapatre et al. ‘Smart College System using IoT BLE Beacons’. In: *International Journal of Advanced Research in Computer and Communication Engineering* (2017). ISSN: 2278-1021. DOI: [10.17148/IJARCCCE.2017.6485](https://doi.org/10.17148/IJARCCCE.2017.6485) (cit. on pp. 16, 17, 25).
- [10] Elaine Barker. ‘Recommendation for key management part 1: General (revision 4)’. In: *NIST special publication* 800.57 (2016) (cit. on p. 35).
- [11] A. Basalamah. ‘Sensing the Crowds Using Bluetooth Low Energy Tags’. In: *IEEE Access* 4 (2016), pp. 4225–4233. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2016.2594210](https://doi.org/10.1109/ACCESS.2016.2594210) (cit. on pp. 17, 18, 25).
- [12] *BGAPI protocol in Java*. URL: <https://github.com/SINTEF-9012/bglib> (visited on 28/08/2017) (cit. on p. 64).

- [13] Tejas Bhandare. ‘LTE and WiMAX Comparison’. In: (2008). URL: <https://pdfs.semanticscholar.org/666c/8a82672e23ccf1248f08c787ac413b09daaa.pdf> (cit. on p. 4).
- [14] Ananya Bhattacharya. *Android just hit a record 88% market share of all smartphones*. QUARZ. URL: <https://qz.com/826672/> (visited on 10/08/2017) (cit. on p. 44).
- [15] *BLE on Android v1.0.1*. Nordic Semiconductor ASA. URL: <https://devzone.nordicsemi.com/attachment/bdd561ff56924e10ea78057b91c5c642> (visited on 11/08/2017) (cit. on pp. 46–48).
- [16] *Bluegiga BLED112 Bluetooth® Low Energy Dongle*. URL: <https://www.silabs.com/products/wireless/bluetooth/bluetooth-low-energy-modules/bled112-bluetooth-smart-dongle> (visited on 28/08/2017) (cit. on p. 64).
- [17] *BluetoothAdapter*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html> (visited on 10/08/2017) (cit. on pp. 46, 47).
- [18] *BluetoothAdapter.LeScanCallback*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.LeScanCallback.html> (visited on 10/08/2017) (cit. on pp. 46, 48).
- [19] *BluetoothDevice*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html> (visited on 10/08/2017) (cit. on p. 48).
- [20] *BluetoothGatt*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothGatt.html> (visited on 10/08/2017) (cit. on p. 48).
- [21] *BluetoothGattCallback*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothGattCallback.html> (visited on 10/08/2017) (cit. on p. 48).
- [22] *BluetoothGattServerCallback*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothGattServerCallback.html> (visited on 10/08/2017) (cit. on p. 48).
- [23] *BluetoothManager*. URL: <https://developer.android.com/reference/android/bluetooth/BluetoothManager.html> (visited on 10/08/2017) (cit. on p. 48).
- [24] Tjeerd W Boonstra, Mark E Larsen and Helen Christensen. ‘Mapping dynamic social networks in real life using participants’ own smartphones’. In: *Heliyon* 1.3 (2015). DOI: [10.1016/j.heliyon.2015.e00037](https://doi.org/10.1016/j.heliyon.2015.e00037) (cit. on pp. 20, 25).
- [25] *comScore Reports January 2016 U.S. Smartphone Subscriber Market Share*. URL: <https://www.comscore.com/Insights/Rankings/comScore-Reports-January-2016-US-Smartphone-Subscriber-Market-Share> (visited on 10/08/2017) (cit. on p. 44).
- [26] Giorgio Conte et al. ‘BlueSentinel: A First Approach Using iBeacon for an Energy Efficient Occupancy Detection System’. In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. BuildSys ’14. Memphis, Tennessee, 2014, pp. 11–19. DOI: [10.1145/2676061.2674078](https://doi.org/10.1145/2676061.2674078) (cit. on pp. 19, 25).

- [27] A. Corna et al. ‘Occupancy Detection via iBeacon on Android Devices for Smart Building Management’. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. DATE ’15. Grenoble, France, 2015, pp. 629–632. DOI: [10.7873/DATE.2015.0753](https://doi.org/10.7873/DATE.2015.0753) (cit. on pp. 19, 25).
- [28] *COYERO access API*. URL: <https://www.coyero.biz/coyero/docs/api/> (visited on 10/08/2017) (cit. on pp. 27–29).
- [29] A. Dementyev et al. ‘Power Consumption Analysis of Bluetooth Low Energy, ZigBee, and ANT Sensor Nodes in a Cyclic Sleep Scenario’. In: *2013 IEEE International Wireless Symposium*. IWS ’13. Apr. 2013. DOI: [10.1109/IEEE-IWS.2013.6616827](https://doi.org/10.1109/IEEE-IWS.2013.6616827) (cit. on p. 4).
- [30] D Deugo. ‘Using Beacons for Attendance Tracking’. In: *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering*. FECS ’16. 2016, pp. 155–161 (cit. on pp. 17, 25).
- [31] *Development Considerations*. URL: <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#considerations> (visited on 10/08/2017) (cit. on p. 45).
- [32] *Driven by Standards, in 2016 Wi-Fi Direct and BLE Protocols Will Ship in 2 and 2.9 Billion Devices Respectively*. ABIResearch. URL: <https://www.abiresearch.com/press/driven-by-standards-in-2016-wi-fi-direct-and-ble-p/> (visited on 31/08/2017) (cit. on p. 5).
- [33] Morris J Dworkin. ‘Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication’. In: *Special Publication (NIST SP)-800-38B* (2016). DOI: [10.6028/NIST.SP.800-38B](https://doi.org/10.6028/NIST.SP.800-38B) (cit. on p. 27).
- [34] Nathan Eagle, Alex (Sandy) Pentland and David Lazer. ‘Inferring friendship network structure by using mobile phone data’. In: *Proceedings of the National Academy of Sciences* 106.36 (2009), pp. 15274–15278. DOI: [10.1073/pnas.0900282106](https://doi.org/10.1073/pnas.0900282106). URL: <http://www.pnas.org/content/106/36/15274.full.pdf> (cit. on pp. 20, 25).
- [35] Sourabh Gavade et al. ‘Automated Bluetooth Attendance Management System’. In: *International Journal of Computer Science Engineering and Information Technology Research* 5.2 (2015), pp. 7–14 (cit. on pp. 17, 25).
- [36] Emmanouil Georgakakis et al. ‘An Analysis of Bluetooth, Zigbee and Bluetooth Low Energy and Their Use in WBANs’. In: *Wireless Mobile Communication and Healthcare. Second International ICST Conference, MobiHealth 2010, Ayia Napa, Cyprus, October 18-20, 2010. Revised Selected Papers*. MobiHealth ’10. Springer. Ayia Napa, Cyprus, 2010, pp. 168–175. DOI: [10.1007/978-3-642-20865-2_22](https://doi.org/10.1007/978-3-642-20865-2_22) (cit. on p. 4).
- [37] N. Gupta. *Inside Bluetooth Low Energy*. Artech House, 2013. ISBN: 9781608075799 (cit. on pp. 3, 4, 8–11, 13, 14).
- [38] Nils Gura et al. ‘Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs’. In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by Marc Joye and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer, 2004, pp. 119–132. ISBN: 9783540286325. DOI: [10.1007/978-3-540-28632-5_9](https://doi.org/10.1007/978-3-540-28632-5_9) (cit. on p. 36).

- [39] R. Heydon. *Bluetooth Low Energy: The Developer's Handbook*. Pearson Education, 2012. ISBN: 9780132888363 (cit. on pp. 5–7, 9, 10, 13, 14).
- [40] *ITU-T Rec. X.667*. ITU-T. Sept. 2004. URL: <https://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf> (visited on 08/08/2017) (cit. on p. 9).
- [41] *Keeping the Device Awake*. URL: <https://developer.android.com/training/scheduling/wakeunlock.html> (visited on 10/08/2017) (cit. on pp. 54, 79).
- [42] Philippe Kruchten. ‘The 4+1 View Model of Architecture’. In: *IEEE Software* 12.6 (Nov. 1995), pp. 42–50. ISSN: 0740-7459. DOI: 10.1109/52.469759 (cit. on pp. 50, 54, 58, 60).
- [43] A. Langley, M. Hamburg and S. Turner. *Elliptic Curves for Security*. RFC 7748. RFC Editor, Jan. 2016. URL: <https://www.rfc-editor.org/rfc/rfc7748.txt> (cit. on p. 36).
- [44] Riya Lodha et al. ‘Bluetooth Smart Based Attendance Management System’. In: ICACTA '15. International Conference on Advanced Computing Technologies and Applications. 2015, pp. 524–527. DOI: 10.1016/j.procs.2015.03.094 (cit. on pp. 15, 17, 25).
- [45] Anzar Mahmood, Nadeem Javaid and Sohail Razzaq. ‘A review of wireless communications for smart grid’. In: *Renewable and Sustainable Energy Reviews* 41 (2015), pp. 248–260. ISSN: 1364-0321. DOI: 10.1016/j.rser.2014.08.036 (cit. on p. 4).
- [46] Alfred J Menezes, Paul C Van Oorschot and Scott A Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 9781439821916 (cit. on p. 26).
- [47] Emiliano Miluzzo et al. ‘Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application’. In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. Raleigh, NC, USA, 2008, pp. 337–350. DOI: 10.1145/1460412.1460445 (cit. on pp. 20, 25).
- [48] S. Movassaghi et al. ‘Wireless Body Area Networks: A Survey’. In: *IEEE Communications Surveys Tutorials* 16.3 (2014), pp. 1658–1686. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.121313.00064 (cit. on p. 4).
- [49] *National Institute of Standards and Technology*. URL: <https://www.nist.gov/> (visited on 10/08/2017) (cit. on p. 24).
- [50] P. Niemeyer and D. Leuck. *Learning Java: A Bestselling Hands-On Java Tutorial*. 4th ed. O'Reilly Media, Incorporated, 2013. ISBN: 9781449319243 (cit. on p. 45).
- [51] *nRF Connect for Mobile*. Nordic Semiconductor ASA. URL: <https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp> (visited on 28/08/2017) (cit. on pp. 78, 79).
- [52] *Number of Android applications*. URL: <https://www.appbrain.com/stats/number-of-android-apps> (visited on 10/08/2017) (cit. on p. 44).
- [53] Gil Reiter. *Wireless connectivity for the Internet of Things*. 2014. URL: <http://www.ti.com/lit/wp/swry010/swry010.pdf> (visited on 01/08/2017) (cit. on p. 4).

- [54] Connie Ribeiro. ‘Bringing Wireless Access to the Automobile: A Comparison of Wi-Fi, WiMAX, MBWA, and 3G’. In: *Proceedings of the 21st Annual Rensselaer at Hartford Computer Science Conference*. Hartford, USA, 2005. URL: <https://pdfs.semanticscholar.org/a6ff/bf930088c7ca698bf5ffe1cdc36b5b9af420.pdf> (cit. on p. 4).
- [55] Mike Ryan. ‘Bluetooth: With Low Energy Comes Low Security’. In: *Proceedings of the 7th USENIX Conference on Offensive Technologies*. WOOT’13. Washigton, D.C.: USENIX Association, 2013. URL: <https://www.usenix.org/conference/woot13/workshop-program/presentation/Ryan> (cit. on p. 9).
- [56] *Samsung Galaxy S6*. URL: http://www.gsmarena.com/samsung_galaxy_s6-6849.php (visited on 28/08/2017) (cit. on p. 77).
- [57] *ScanCallback*. URL: <https://developer.android.com/reference/android/bluetooth/le/ScanCallback.html> (visited on 10/08/2017) (cit. on pp. 46, 48).
- [58] *ScanResult*. URL: <https://developer.android.com/reference/android/bluetooth/le/ScanResult.html> (visited on 10/08/2017) (cit. on p. 46).
- [59] JH. Song et al. *The AES-CMAC Algorithm*. RFC 4493. RFC Editor, June 2006. URL: <https://www.rfc-editor.org/rfc/rfc4493.txt> (cit. on p. 40).
- [60] *Specification of the Bluetooth System 4.0*. The Bluetooth Special Interest Group. June 2010. URL: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737 (visited on 01/08/2017) (cit. on p. 5).
- [61] *Specification of the Bluetooth System 4.1*. The Bluetooth Special Interest Group. Dec. 2013. URL: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282159 (visited on 01/08/2017) (cit. on pp. 5–11, 13, 23, 78).
- [62] *Spongy Castle*. URL: <https://rtyley.github.io/spongycastle/> (visited on 10/08/2017) (cit. on pp. 50, 54).
- [63] *The Build Process*. URL: <https://developer.android.com/studio/build/index.html#build-process> (visited on 10/08/2017) (cit. on p. 45).
- [64] K. Townsend, C. Cufí and R. Davidson. *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. 1st ed. O’Reilly Media, 2014. ISBN: 9781491949511 (cit. on pp. 5, 6, 8, 10, 11, 13).
- [65] Samuel Townsend et al. ‘Using Bluetooth Low Energy in smartphones to map social networks’. In: *CoRR* abs/1508.03938 (2015). URL: <http://arxiv.org/abs/1508.03938> (cit. on pp. 20, 25).
- [66] *Traditional Profile Specifications*. The Bluetooth Special Interest Group. URL: <https://www.bluetooth.com/specifications/profiles-overview> (visited on 10/08/2017) (cit. on p. 13).
- [67] *Transmitting Network Data Using Volley*. URL: <https://developer.android.com/training/volley/index.html> (visited on 10/08/2017) (cit. on p. 50).
- [68] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*. Springer Science & Business Media, 2014. ISBN: 9780387234731 (cit. on pp. 24, 26, 27, 41).

- [69] Mahendraker Vinay and D Savitha. ‘Student Tracking System with Bluetooth Low Energy’. In: *International Journal of Engineering Technology Science and Research* 4.5 (2017), pp. 509–512. ISSN: 2394 – 3386. URL: http://www.ijetsr.com/images/short_pdf/1496055537_ietep311_ijetsr.pdf (cit. on pp. 16, 17, 25).
- [70] James Vincent. *99.6 percent of new smartphones run Android or iOS*. The Verge. URL: <https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016> (visited on 10/08/2017) (cit. on p. 44).
- [71] *What is API Level?* URL: <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels> (visited on 10/08/2017) (cit. on p. 45).
- [72] Wikipedia contributors. *Android*. Wikipedia, The Free Encyclopedia. URL: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)) (visited on 10/08/2017) (cit. on p. 44).
- [73] Wikipedia contributors. *List of Android app stores*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/List_of_Android_app_stores (visited on 10/08/2017) (cit. on p. 44).
- [74] Wikipedia contributors. *Relay Attack*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Relay_attack (visited on 14/08/2017) (cit. on p. 41).
- [75] Wikipedia contributors. *Representational state transfer*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Representational_state_transfer (visited on 04/08/2017) (cit. on p. 27).
- [76] Wikipedia contributors. *Shared Secret*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Shared_secret (visited on 10/08/2017) (cit. on p. 26).
- [77] Wikipedia contributors. *Spoofing Attack*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Spoofing_attack (visited on 04/08/2017) (cit. on p. 23).
- [78] Wikipedia contributors. *WiFi Fingerprinting*. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Wi-Fi_positioning_system#Fingerprinting_based (visited on 04/08/2017) (cit. on p. 20).
- [79] Yang Yang, Zhouchi Li and K. Pahlavan. ‘Using iBeacon for Intelligent In-Room Presence Detection’. In: *2016 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*. CogSIMA ’16. 2016, pp. 187–191. DOI: [10.1109/COGSIMA.2016.7497808](https://doi.org/10.1109/COGSIMA.2016.7497808) (cit. on pp. 19, 25).
- [80] Mengyu Zhou et al. ‘EDUM: Classroom Education Measurements via Large-scale WiFi Networks’. In: *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’16. Heidelberg, Germany, 2016, pp. 316–327. DOI: [10.1145/2971648.2971657](https://doi.org/10.1145/2971648.2971657) (cit. on pp. 19, 25).
- [81] Mengyu Zhou et al. ‘MobiCamp: A Campus-wide Testbed for Studying Mobile Physical Activities’. In: *Proceedings of the 3rd International on Workshop on Physical Analytics*. WPA ’16. Singapore, Singapore, 2016, pp. 1–6. DOI: [10.1145/2935651.2935654](https://doi.org/10.1145/2935651.2935654) (cit. on p. 19).