Alexander Rech, BSc.

Design and Implementation of a Secure Embedded Client and Gateway Application for Cloud Access Using Bluetooth Smart

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

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, May 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.

Date	Signature

Abstract

The Internet of Things is continuously advancing. Over the last decades more and more devices have been connected to the internet including PCs, handheld devices such as smartphones or tablets, and above all, embedded devices of all kinds. However, since embedded devices are usually constrained in terms of direct internet access other approaches are needed to provide them with the possibility of back-end communication. To overcome this problem, this thesis proposes the design and implementation of a secure embedded device and furthermore introduces a corresponding smartphone-based gateway solution which enables indirect communication between the constrained embedded client and the cloud.

The technology used to transfer data between the embedded client and the gateway is Bluetooth Low Energy (BLE). BLE is among the leading wireless communication standards and is especially useful for battery operated devices by reason of its power saving design. BLE specifies several roles which can be carried out by a device. Each role comes with certain requirements and restrictions and devices have to act in roles compatible to each other to exchange data. Therefore the embedded client was designed in such a way to maximize compatibility with other BLE devices by supporting multiple roles.

The application area of the developed prototype is the parking domain. It enables its users to gain access to parking lots and garages without relying on the constant interaction with a smartphone. To put it in other words, the user should no longer be involved in the process of retrieving parking tickets nor should he have to pull out his smartphone the moment he wants to access a parking space. This will enhance usability a lot. The prototype is implemented as part of the COYERO access platform which is specialized in providing their users with access to local infrastructures, services, and products via smartphone apps. High-end security mechanisms are a crucial aspect of COYERO access to overcome security faults when dealing with authentication and authorization. Hence, this thesis shall also investigate and discuss ways to securely exchange data between the embedded client and its communication participants.

Kurzfassung

Der Vormarsch des Internet of Things lässt sich über die letzten Jahrzehnte hinweg beobachten. Neben PCs und Handhelds wie Smartphones und Tablets, verbinden sich immer mehr Geräte mit dem Internet, darunter auch eingebettete Systeme verschiedenster Art. Da diese Gerätschaften jedoch oft nicht eigenständig Daten über das Internet austauschen können, müssen andere Wege aufbereitet werden, um Kommunikation zu Servern, bzw. zur Cloud zu ermöglichen. Um diesem Problem Abhilfe zu schaffen, präsentiert diese Arbeit das Design und die Implementierung eines sicheren, eingebetteten Gerätes und eines dazugehörigen Smartphone Gateways, welches zwischen dem eingebetteten Gerät und der Cloud vermittelt.

Die verwendete Technologie, welche die Kommunikation zwischen dem eingebetteten Client und dem Gateway ermöglicht, ist Bluetooth Low Energy (BLE). BLE gehört zu den führenden drahtlosen Kommunikationsstandards und ist vor allem aufgrund seiner energiesparenden Art für batteriebetriebene Anwendungen vorteilhaft. Der BLE Standard legt diverse Rollen fest, die ein Gerät einnehmen kann. Jede Rolle hat gewisse Anforderungen und Einschränkungen, die es zu beachten gilt, damit Geräte untereinander Daten austauschen können. Diese Tatsache wurde beim Design des eingebetteten Clients insofern berücksichtigt, als dass dieser verschiedene BLE Rollen unterstützt, um maximale Kompatibilität zu anderen BLE Geräten zu gewährleisten.

Das Anwendungsgebiet des entwickelten Prototyps ist Smart Parking. Er zielt darauf ab, seinen Benutzern Eintritt zu diversen Parkplätzen und Garagen zu verschaffen, ohne dass diese Zeit mit Parktickets verlieren oder eine bestimmte Smartphone App griffbereit halten müssen. Dies kommt wiederum der Benutzerfreundlichkeit zugute. Der Prototyp wurde als Teil der COYERO access Plattform implementiert, die darin spezialisiert ist, ihren Nutzern Zugang zu lokalen Infrastrukturen, Services und Produkten über Smartphone Apps zu ermöglichen. Hohe Sicherheitsstandards für Authentifizierung und Autorisierung sind dabei ein wichtiger Bestandteil von COYERO access. Deshalb liegt ein großes Anliegen dieser Arbeit auch darin, den Datenaustausch zwischen eingebettetem Client und seinen Kommunikationsteilnehmern vor betrügerischen Manipulationen abzusichern.

Acknowledgement

This master's thesis was written in 2017 at the Institute for Technical Informatics at Graz University of Technology in cooperation with CISC Semiconductor GmbH.

I would like to express my sincere gratitude to my supervisor Christian Steger who made it possible to write the thesis in this sophisticated way.

My thanks also go to the members of CISC Semiconductor GmbH for their backing in overcoming obstacles I faced during my research.

Last but not least, I would like to extend my heartfelt gratitude to my family and friends, especially to my parents, for their unfailing support and trust throughout my studies and life.

Graz, May 2017 Alexander Rech

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
API	Application Program Interface
ASN.1	Abstract Syntax Notation One
ATT	Attribute Protocol
AUTH	Authentication
BER	Basic Encoding Rules
BLE	Bluetooth Low Energy
CA	Certificate Authority
CCCD	Client Characteristic Configuration Descriptor
DER	Distinguished Encoding Rules
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EClient	Embedded Client
EEPROM	Electrically Erasable Programmable Read Only Memory
GATT	Generic Attribute Profile
GPIO	General Purpose Input/Output
GUI	Graphical User Interface

HCI	Host Controller Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IoT	Internet of Things
ISM	Industrial Scientific Medical
IFS	Inter Frame Space
I^2C	Inter Integrated Circuit
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaption Protocol
MITM	Man in the Middle
MCU	Micro Control Unit
NFC	Near Field Communication
OOB	Out of Band
OSGi	Open Services Gateway Initiative
PKI	Public Key Infrastructure
QR	Quick Response
REST	Representational State Transfer
RFID	Radio Frequency Identification
RNG	Random Number Generator
RSA	Rivest Shamir Adleman
RTLS	Real Time Locating System

SIG	Special Interest Group
SLOC	Source Lines of Code
SMP	Security Manager Protocol
SOC	System on a Chip
STK	Short Term Key
TDMA	Time Division Multiple Access
TK	Temporary Key
TLV	Tag Length Value
TTL	Transistor Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VA	Validation Authority
WSN	Wireless Sensor Network

Chapter 1

Introduction

This thesis proposes a way of providing back-end communication to an internet constrained device by using Bluetooth Smart also known as Bluetooth Low Energy (BLE). Fig. 1.1 gives an overview of the setup of the communication participants. A smartphone-based gateway is used to communicate with the embedded device over BLE. Consequently, the request is forwarded to the cloud by using HTTP calls. Finally, the corresponding response is processed and transferred back to the embedded device over BLE. The design and implementation of the embedded client, referred to as EClient in this thesis, and the corresponding gateway smartphone application is realized as part of the COYERO access platform.

The application area of the developed prototype is smart parking and it aims to enhance usability by minimizing interactions between the user, his mobile phone, and the EClient. The user basically concentrates on reaching the parking lot while the EClient takes care of providing access. Security goals like trusted authentication and authorization as well as data integrity are ensured via cryptographic standards in order to thwart fraudulent manipulations.

Figure 1.1: Communication Participants

Chapter 2 gives technical insights into the features and methodologies of the software platform COYERO access. Among other things, it is specialized on providing personalized and secure access to goods and services over BLE and NFC. In addition to its key features chapter 2 will also discuss security relevant aspects taken into account during the design phase of the embedded device. The second part of this chapter subsequently takes a closer look at the wireless technology Bluetooth Low Energy, focusing on the BLE Protocol Stack and how its layers work together in order to transfer data. Thanks to its power saving features BLE offers great benefits for Internet of Things (IoT) applications. For this reason, it is nowadays deployed in a large set of different applications. Therefore,

chapter 2 will additionally conduct a research on related work distinguishing between commercial and scientific approaches. Projects which utilize BLE in a similar way as the prototype developed in the course of this thesis will be examined. The focus will lie on BLE's multi-role behavior and on ways to provide internet constrained devices with the possibility to talk to servers over the internet. Additionally, projects which are already operating in the parking domain and offer their users smart parking without smartphone will be discussed.

The third chapter contains design aspects of the EClient and the corresponding smartphone gateway implemented as part of the COYERO Client Android app. First of all, the use case and requirements will be shown. Based on this information, the advantages and disadvantages of several eligible MCUs and hardware modules are compared with one another. The most suitable device will become the basis of the succeeding design discussion which gives an overview on how the embedded client works in conjunction with its gateway.

Chapter 4 gives a good insight into details regarding the implementation. First, information about the development environment including settings and used libraries will be given. Second, a closer look will be taken at the circuit schematic of the EClient. Afterwards, the focus of this chapter will lie on explaining the structure of the developed program on EClient and gateway side. Additionally, different encoding schemata that come into play during data transfer will be presented. Last but not least, the program flow of the implemented program is shown and explained.

Subsequently, results and findings of chapters 3 and 4 will be discussed, compared, and evaluated against other similar systems presented in chapter 2. Finally, the most important aspects of this thesis will be summarized and remaining issues will be addressed before proposing possible solutions.

Chapter 2

Related Work

Section 2.1 of this chapter describes the platform COYERO access, emphasizing on data handling and its key features and security aspects. Subsequently, a detailed overview of Bluetooth Low Energy will be provided in section 2.2.

In addition, this chapter is based on a research of projects and products that rely on the same or similar technologies used to design and implement the EClient and the gateway application. Embedded devices of all kinds are widely applied, most of the time without even being perceived by people. However, those devices often still lack the ability to talk to other devices over the internet. This happens either because they are distant from wide area networks such as Wi-Fi and the deployment of these networks in a sparse environment is difficult and expensive, or because the embedded devices do not have the possibility to rely on power consuming wireless technologies. Wi-Fi, for instance, is widespread but its large power requirements make it unsuitable for low-power applications.

An overview of several low power wireless standards as well as comparative experimental studies between BLE and other low power wireless technologies can be found in the literature [SC11, DHTS13, TMTG13]. The results showed that BLE is very energy efficient and one of the wireless technologies most robust to obstacles. Hence, it is a good candidate when power constrained embedded devices come into play. Its usage in the embedded sector is increasing rapidly.

Data collection with smart devices in different environments is becoming more and more important due to early danger detection [FMA+16] or for localization specific purposes [CLH16], to name a few examples. In these cases usually a network of different sensors capable of communicating amongst each other is deployed. In any case, devices with two way communication capabilities, either connection based or not, are necessary to forward signals over different paths. As a means of augmenting flexibility regarding the data transfer itself, it would be advantageous if each device was able to initiate the actual data transfer. To meet this requirement devices relying on BLE for data transmission have to support multiple BLE roles. Therefore, particular emphasis is placed on projects utilizing BLE in different roles to establish communication between several devices. See subsection 2.3.1 for further details on this topic.

Regarding the question on how to connect BLE devices to the internet there are basically two different approaches. In several cases a gateway that passes messages between embedded devices and the cloud is used. See subsection 2.3.2 for approaches discussed in academic papers or section 2.4 to get further information on products already deployed

commercially. Otherwise, there is the possibility of combining IPv6 and BLE, enabling BLE devices to communicate directly with other devices over the internet. More details on that aspect can be found in subsection 2.3.3.

Finally, section 2.5 names two already commercially deployed products with a use case related to the one of the prototype discussed in this thesis. To put it in a nutshell: it is all about smart parking without smartphone.

2.1 COYERO access

The software platform COYERO access is divided into three complementary parts: COYERO Client, COYERO Kiosk, and COYERO Server. Together they provide services for secure authentication, authorization, and accounting for applications of different kinds.

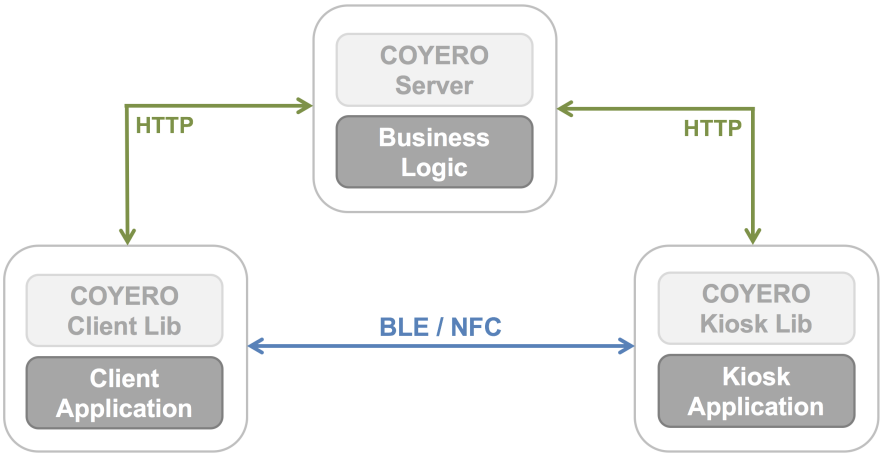

Figure 2.1: COYERO access Environment (Adapted from [CIS15])

Among other things, COYERO access provides several tools in the form of software libraries. Those libraries can be embedded into smartphone apps, for instance, enabling their users to gain access to local infrastructures, services, and products. Regarding the communication technologies used, COYERO Client and Kiosk devices transfer data between each other over BLE or NFC. They can also communicate with the server infrastructure over HTTP. The server infrastructure itself is divided into a business logic part responsible for application specific functionality and the COYERO Server module, which monitors and manages the entire framework and acts as secure cloud service [CIS15].

2.1.1 COYERO access Features

The following part explains key features of the COYERO Client and Kiosk libraries and the COYERO Server module [CIS15].

Features of the Client/Kiosk Library

The COYERO Client library supports iOS and Android, while the COYERO Kiosk library supports Android and was also implemented for the OSGi (Open Services Gateway initiative) platform. Both libraries use NFC and BLE to exchange data. Initial internet access is an essential requirement for this. Additionally, the COYERO Kiosk and Client libraries offer the following functions to the enclosing native applications through an API layer tailored for communicating to the COYERO Server module:

- **Authentication:**
  Devices using the COYERO Client and COYERO Kiosk libraries can prove their identity to each other by using Authentication-Tokens generated and signed by the COYERO Server. See subsection 2.1.2 for further details on Authentication-Tokens.
- **Authorization:**
  Kiosk devices can authorize the access to goods and services for client devices by redeeming their server signed Entitlement-Tokens. Refer to subsection 2.1.2 to gain a better overview of Entitlement-Tokens.
- **Device coupling:**
  Two devices can be coupled in case of close proximity using different communication technologies like NFC or BLE.
- **Transaction handling:**
  Connected devices can offer entitlements to or request entitlements from each other. In this case a challenge-response based authentication method is used before further data is exchanged between COYERO Client and Kiosk.
- **Accounting:**
  If an entitlement was redeemed, a transaction log is created and propagated back to the server once internet connectivity is available. This prevents duplicate transactions and other fraudulent manipulations and assures non-repudiation for the redemption process.

Features of the Server Module

The COYERO Server module provides the entry point for external applications and manages and monitors transactions between COYERO devices. Additionally, it operates in constant interaction with the Business Logic which is an application server responsible for providing COYERO Client and Kiosk devices with application specific functionality. Communication with the COYERO Server module is triggered over a REST webservice API in charge of the following services:

- **User and Device Handling:**
  The COYERO Server supplies devices containing the COYERO Client or Kiosk library with authentication. New users or devices have to register on the server initially. Afterwards, devices can communicate directly with the COYERO Server module using their server issued Authentication-Tokens. Subsection 2.1.3 explains the registration procedure in more detail.

- **Entitlement Management:**
  The creation of Entitlement-Tokens is triggered by the Business Logic through the REST webservice API. Consequently, these entitlements are distributed by the COYERO Server to authenticated devices.
- **Monitoring:**
  The COYERO Server maintains a transaction record database.

2.1.2 COYERO access Token Management

The COYERO access platform makes use of special data structures called tokens to handle user authentication and authorization to access products and services.[1] Each token has to be signed by the COYERO Server in order to make later modification of data impossible. For further details on security relevant information refer to subsection 2.1.4.

Authentication-Token

An Authentication-Token (Auth-Token) is an extended certificate which includes the public key of a client. It is tied to a particular user account and device and signed by the COYERO Server. An Auth-Token consists of all elements listed in table 2.1.

Element	Length (Bytes)	Description
Version	2	Current Version of Auth-Token
Id	8	Identifier of Auth-Token
IssuerId	4	Issuer of Auth-Token
ServiceStart	8	Start of Validity Period of Auth-Token
ServiceEnd	8	End of Validity Period of Auth-Token
PublicKey	91	Public secp256r1-ECC-Key (with Header)
Properties	2	Information to Control and Restrict Possible Use Cases for the Auth-Token
Signature	70-72	DER-Encoded Signature of Auth-Token

Table 2.1: Structure of an Authentication-Token

Entitlement-Token

An Entitlement-Token is issued and signed by the COYERO Server and bound to a specific Auth-Token. Therefore, it is only valid when presented in combination with the related user account and device. Entitlement-Tokens are divided into the following elements illustrated in table 2.2.

[1] Information on tokens was taken from the internal documentation of CISC Semiconductor GmbH.

Element	Length (Bytes)	Description
Version	2	Current Version of Entitlement-Token
Id	8	Identifier of Entitlement-Token
IssuerId	4	Issuer of Entitlement-Token
AuthTokenId	8	Identifier of Corresponding Auth-Token
ApplicationId	4	Identifier of Application
ServiceId	4	Identifier of Product/Service
CommonDataAmount	4	Payment Related Data
ServiceStart	8	Start of Validity Period of Entitlement-Token
ServiceEnd	8	End of Validity Period of Entitlement-Token
Properties	2	Information to Control and Restrict Possible Use Cases for the Entitlement-Token
ApplicationData	0-64	Application Specific Data Buffer
Signature	70-72	DER-Encoded Signature of Entitlement-Token

Table 2.2: Structure of an Entitlement-Token

2.1.3 Communication Between COYERO Entities

The communication between COYERO entities, including COYERO Client, COYERO Kiosk, and COYERO Server, can be classified into the following three procedures (see fig. 2.2). They will be described on the following pages based on [CIS15].

Figure 2.2: COYERO access Communication Sequence (Adapted from [CIS15])

Creation of Authentication-Tokens

Before client-side applications are authorized to use the services provided by the COYERO access platform, they have to gather a server issued Auth-Token by registering themselves at the COYERO Server. Figure 2.3 explains how an Auth-Token is retrieved.

First of all, after generating an ECC key-pair, the client application needs to provide login credentials to the Business Logic layer. In further consequence, a one-time registration code, also known as ticket, will be fetched by the Business Logic and sent back to the client application. Consequently, this ticket is sent to the COYERO Server in combination with the client device's public key. Finally, the server issues and signs the Auth-Token and sends it back to the client device.

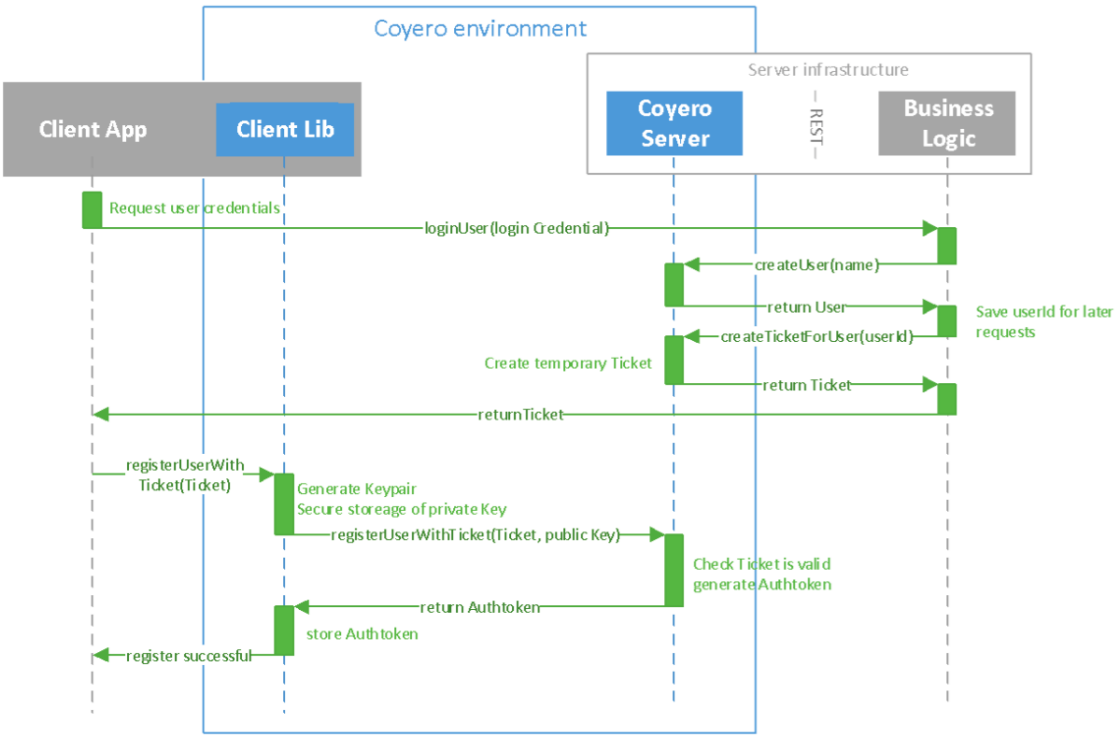

Figure 2.3: Login Procedure and Creation of an Authentication-Token [CIS15]

Creation of Entitlement-Tokens

After the login-process is finished an Authentication-Token is stored locally on the client device and entitlements can be retrieved. The creation of Entitlement-Tokens can be triggered directly by the Business Logic. In any case, the COYERO Server is responsible for issuing, signing, and sending them to the client where they are stored for later redemption (see figure 2.4).

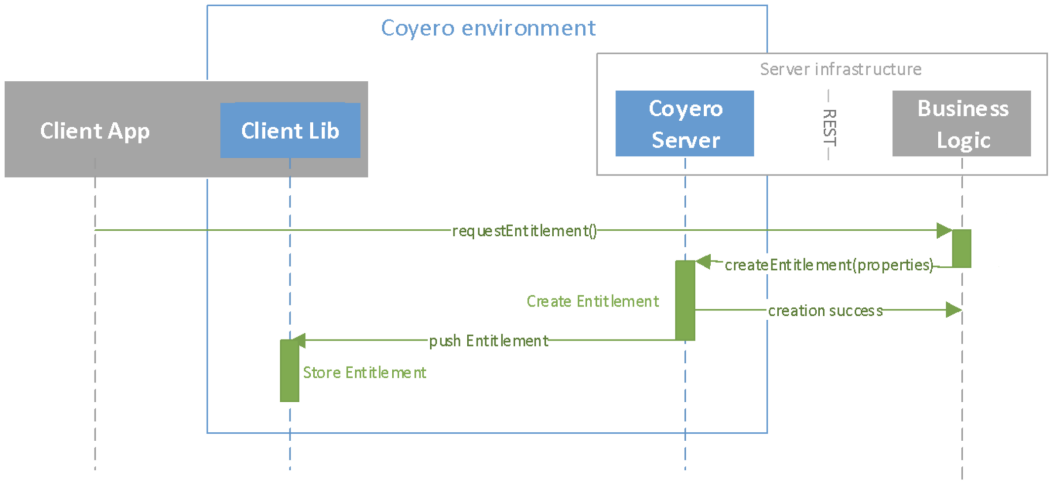

Figure 2.4: Creation of an Entitlement-Token [CIS15]

Redemption of Entitlement-Tokens

As soon as entitlements are available on the client device they can be redeemed at corresponding kiosk devices. A kiosk only accepts entitlements if their ApplicationID and ServiceID correspond to its predefined filter criteria. According to figure 2.5 the COYERO Client library and the COYERO Kiosk library perform an authentication procedure based on their Authentication-Tokens using a challenge-response method. Afterwards, the kiosk can check the received entitlements with the public key of the COYERO Server. If they are valid they are redeemed offline or online depending on the use case. In addition, a transaction log is generated and transmitted back to the server.

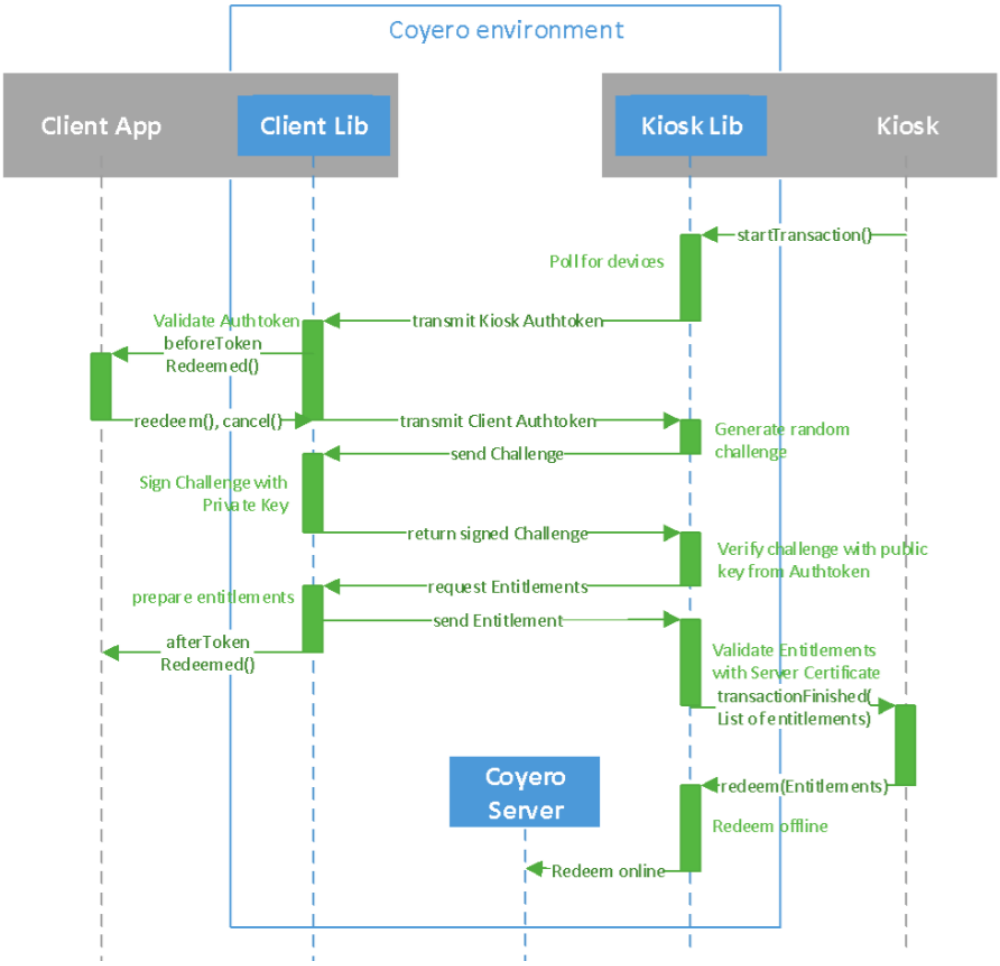

Figure 2.5: Redemption of an Entitlement-Token [CIS15]

2.1.4 Security

This subsection gives an insight into the security aspects of COYERO access. In order to provide security goals like data integrity and trusted authentication and authorization COYERO's security is based on Public Key Infrastructure (PKI) in combination with ECDSA. These cryptographic concepts are explained in the following paragraphs.

Public Key Infrastructure

A Public Key Infrastructure (PKI) is a security infrastructure. Its services rely on public-key concepts and its main goal is to ensure the identity of all communication participants. Generally, a PKI provides a system with the following services (see [AC99] for further information):

- **Authentication**
  On one hand, PKI provides the service of *data origin authentication*. An entity's private key may be used for the signing process, binding its identity to the signed data. On the other hand, also the service of *entity authentication* is offered. This is useful to check if the entity that sent the signature was the actual signer. Hence, the entity might be asked to sign an additional challenge which subsequently can be verified with its public key by the other communication participant.
- **Integrity**
  The signed data can be verified against the actual data to check if it was altered.
- **Confidentiality**
  Confidentiality can be provided with public encryption algorithms for instance. Data encrypted with the recipient's public key can only be decrypted by the one possessing the corresponding private key.

In case of COYERO access authentication and integrity is provided through server signed tokens and a challenge-response protocol between COYERO Client and Kiosk devices. However, confidentiality is not provided. This is due to the fact that tokens are only of benefit if the challenge-response protocol involving the private key of the right owner is passed. Therefore, as a means of speeding up data transmission, the actual data is not encrypted.

Figure 2.6 describes the trust relationship between all COYERO entities. The COYERO Server is a Certificate Authority (CA) responsible for issuing and signing Authentication- and Entitlement-Tokens for COYERO Client and Kiosk devices. In contrast to the COYERO Server, Kiosk devices are Validation Authorities (VAs). They validate the authentication and authorization of a client device based on its Authentication- and/or Entitlement-Tokens. Last but not least, there is the self signed root CA. It is the basis of trust for all COYERO Client and Kiosk devices since the signed data coming from the COYERO Server can be verified with their pre-installed root certificate [CIS15].

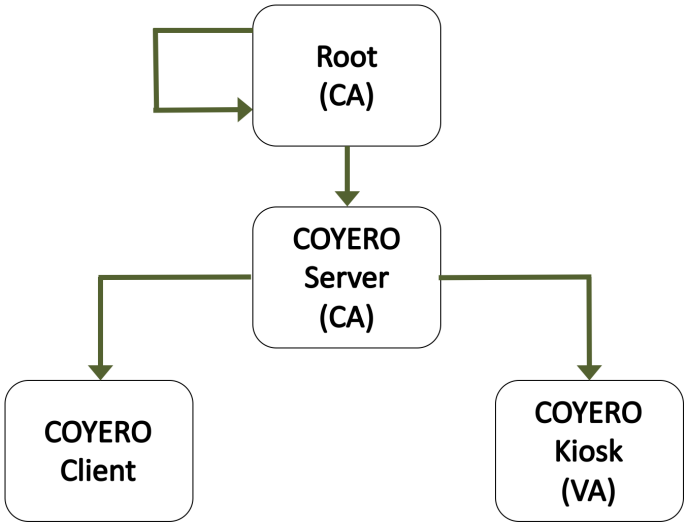

Figure 2.6: Trust Relationship Between COYERO Entities

ECDSA

COYERO access relies on the ECDSA algorithm for creating and verifying signatures. The elliptic curve secp256r1 (aka prime256v1, NIST P-256 [TIB+09]) is used for this purpose.

According to the NIST Recommendation of 2016 this type of curve is recommended at least until 2030. The major advantage of ECDSA lies in its cryptographic key length. Table 2.3 lists the cryptographic key sizes of ECC and RSA required for the same level of security [Blu16a]. It is expressly stated that RSA has to operate with significantly larger keys. This, in consequence, impacts the speed of the corresponding algorithm and affects memory requirements. In case of ECDSA this enables a quicker data transfer of keys and certificates since smaller data packets can be passed.

RSA Key Length (Bits)	160	224	256	384
ECC Key Length (Bits)	1024	2048	3072	7680

Table 2.3: Key Length Comparison Between RSA and ECC

2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a short range wireless technology. Its specifications are defined by the Bluetooth Special Interest Group (SIG). Compared to Bluetooth Classic it is less power hungry and was especially designed for power constrained devices and control and monitoring applications. All the information of this section was derived and adapted from [GOP12, Blu10, TCDD14].

2.2.1 Bluetooth Low Energy Protocol Stack

The BLE protocol stack consists of a Host and a Controller part. Communication between them is established over the Host Controller Interface (HCI). The Controller is composed of the Physical and the Link Layer, while the Host comprises the following parts:

- Logical Link Control and Adaption Protocol (L2CAP)
- Attribute Protocol (ATT)
- Generic Attribute Profile (GATT)
- Security Manager Protocol (SMP)
- Generic Access Profile (GAP)

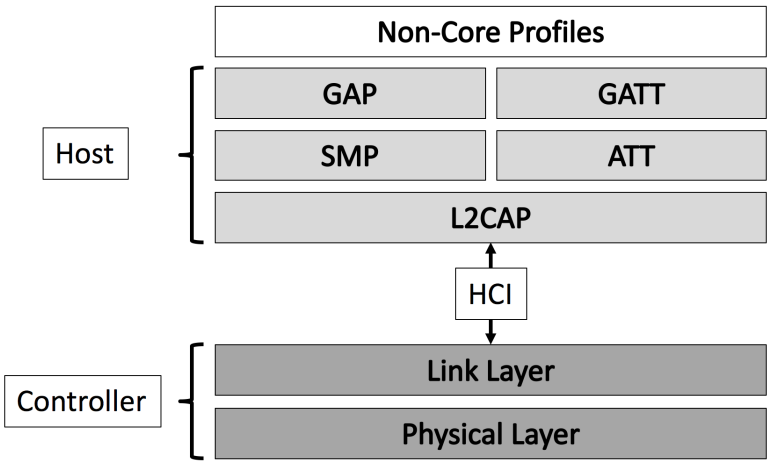

Figure 2.7: BLE Stack (Adapted from [GOP12])

Non-core profiles are features of the application layer which are not defined by the Bluetooth specification. They run on top of the Host. Figure 2.7 gives an overview of the BLE stack which will be described in the following paragraphs.

2.2.2 Physical Layer

BLE uses the 2.4 GHz ISM band for communication and divides it into 40 channels ranging from 2.4000 GHz to 2.4835 GHz. All available BLE channels and their frequencies are illustrated in figure 2.8. Channels 37, 38 and 39 are advertising channels responsible for

device discovery mechanisms, connection establishment, and sending broadcasts. Furthermore, the frequencies of the three advertising channels were defined in a way to minimize overlapping, with the commonly used IEEE 802.11 channels 1, 6 and 11. The remaining 37 channels (0-36) are data channels and responsible for bidirectional data transfer. They are switched via an adaptive frequency hopping mechanism in order to mitigate issues regarding signal interference and problems related to wireless propagation (e.g. multipath fading).

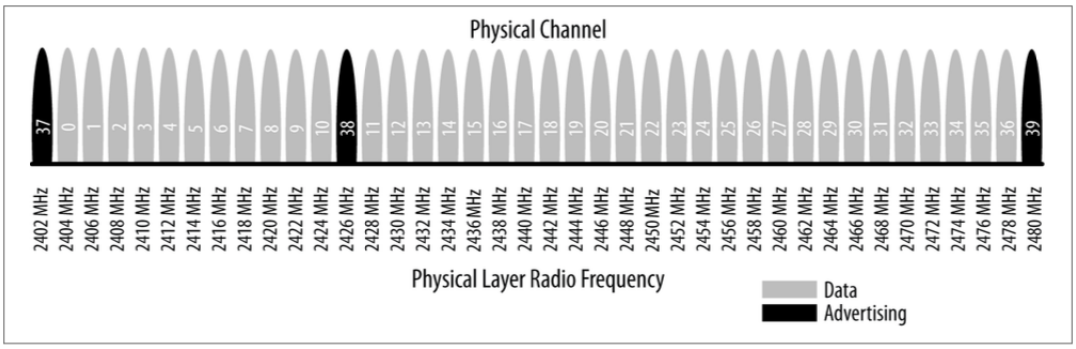

Figure 2.8: BLE Physical Layer Frequency Channels [TCDD14]

2.2.3 Link Layer

The Link Layer defines the following device roles:

Advertiser and Scanner (No Connection Required):
A device sending advertising packets through the advertising channels is called advertiser. In contrast, a scanner is a device scanning for advertising packets and receiving data through advertising channels. Data packets are advertised in time intervals. Within such an interval, called advertising event, the advertiser sequentially makes use of each advertising channel for packet transmission. Furthermore, an advertising packet can be classified according to three properties:

- **Connectability**: The scanner can or cannot initiate a connection upon receiving an advertising packet.
- **Scannability**: A scanner can or cannot issue a scan request upon reception of such an advertising packet.
- **Directability**: A *directed* packet contains only the Bluetooth addresses of the advertiser and the desired target scanner in its payload. No user data is allowed. All advertising packets of this kind are therefore connectable. An *undirected* packet is not targeted at any particular scanner and it can contain user data in its payload.

Master and Slave (Connection Required):
A master device listens for advertisements and transmits connection request messages, while a slave accepts connection requests and follows the timings specified by the master. A master can connect to multiple devices. In contrast, a slave can only be connected to

one master at a time. After establishing a point to point connection between a master and a slave, data can be transferred through the physical data channels. Consequently, the physical channel is divided into non-overlapping time units, called connection events. A frequency hopping algorithm is used to determine the data channel index (and frequency) for each connection event. Within a connection event, all packets are transmitted using the same frequency. Each connection event contains at least one packet sent by the master. If the slave received the packet, it must send a packet back as a response. An Inter Frame Space (IFS) of at least 150 μs must pass between the end of transmission of one packet and the start of the next one. For a new connection event, master and slave use a new data channel frequency provided by the frequency hopping algorithm.

Slaves are in sleep mode by default and wake up regularly to listen for possible packets from the master. The master decides when the slaves listen and coordinates the channel access by utilizing a Time Division Multiple Access (TDMA) scheme. Additionally, the master provides the slave with information required for the frequency hopping algorithm which is transmitted during the connection request message and updated during the connection. The following timing parameters have to be considered:

- **Connection Interval:** The time between two connection events is called connection interval. The advantage of a long connection interval is energy saving since the device can sleep most of the time between connection events. The disadvantage is that there are less opportunities for data to be sent or received.
- **Slave Latency:** The number of connection events during which a slave does not listen for the master is called slave latency. A reduction of the slave latency increases the power consumption but also reduces the amount of time for data to be sent and received.
- **Supervision Timeout:** The maximum time allowed between two data packets before the connection is considered lost is defined as supervision timeout.

2.2.4 L2CAP

The L2CAP layer manages and controls data coming from the ATT, SMP and Link Layer. It operates according to a best-effort approach. Mechanisms like retransmission or flow control do not exist. Neither segmentation nor reassembly is required since the upper layer protocols provide data packets that fit L2CAP's maximum payload size of 23 bytes.

2.2.5 SMP

The Security Manager is both a protocol and a series of security algorithms designed to provide the Bluetooth protocol stack with the ability to generate and exchange security keys for BLE pairing. In further consequence, this allows the peers to communicate securely over an encrypted link and trust the identity of the remote device. Additionally, the Security Manager offers the possibility to hide the public Bluetooth address from a medium to avoid being tracked by other devices.

Subsection 6.2.2 provides information on how the Security Manager could enhance the security of the developed prototype.

2.2.6 ATT and GATT

The Attribute Protocol (ATT) is a stateless protocol for discovering, reading, and modifying data on a peer device. Depending on the use case and the offered features of the Bluetooth unit, a device can act as client or server. Data is always stored in form of attributes on the server and can be accessed and modified by the client. An attribute is an addressable piece of information that can contain relevant data. It consists of the following elements:

- **Handle:** The Handle is a 16-bit long identifier to address and access an attribute value.
- **Type:** The Type is a 128-bit long UUID that specifies which kind of data is contained in the value of the attribute. For efficiency there also exist 16-bit and 32-bit formats used for UUIDs that are defined in the Bluetooth specification.
- **Permissions:** The Permissions element is metadata that specifies which operations can be executed on the attribute. It also incorporates security requirements.
- **Value:** The Value consists of the attribute's data content.

When a client wants to read an attribute or write to it, it will be addressed by its handle. Subsequently, the server will respond with the attribute value or an acknowledgement.

The Generic Attributes Profile (GATT) is built on top of ATT and uses it as its transport protocol. GATT basically defines a framework for transferring data between devices using concepts like *services*, *characteristics* and *descriptors* explained in the following paragraphs:

Services

Attributes on a GATT server are grouped into services. Conceptually, a GATT service could be compared to a class in object-oriented languages which breaks up data into logical entities. Services are read only and cannot be modified by the client.

Characteristics

Characteristics are data containers and can be contained in a service. A characteristic includes at least two attributes:

- **Characteristic declaration:** It provides metadata about the user data. Its value field consists of the following elements:
  - **Properties:** They describe which operations can be performed on this characteristic, e.g. Read / Write / Notify / Indicate.
  - **Value Handle:** It includes an attribute handle of the attribute that contains the value of the characteristic.
  - **UUID:** It contains the attribute type of the attribute which contains the value of the characteristic.

- **Characteristic value:** The Characteristic's value field holds the actual user data the client can read from or write to. Type and handle are the same as specified in the value field of the characteristic declaration attribute.

Descriptor

Characteristics may include descriptors. One one hand, a client may read a descriptor to get additional information about the characteristic and its value. On the other hand, a client may also write to a descriptor in order to configure a specific characteristic. Depending on its value field, notifications or indications can be switched on or off, for example.

GATT Service Example [TCDD14]

The attribute structure discussed above will now be explained by means of an example. Figure 2.9 illustrates a GATT service of a heart-rate-monitor. The attribute with *handle* 0x0021 contains the service declaration for the heart-rate service. Like all services it is read only. Its *value* field specifies that it is a heart-rate service (HRS).

The attribute with *handle* 0x0024 incorporates the *characteristic declaration*. Its *value* field consists of three elements: the *properties* field which is set to notify only, the *value handle* specified as 0x0027, and a *UUID* which is the identifier for the heart-rate measurement. In contrast, the attribute with *handle* 0x0027 contains the *characteristic value* which contains the actual heart-rate measurement value. The *UUID* of this attribute is the same as specified inside the *value* field of the *characteristic declaration*. Additionally, the *permissions* field defines that neither read nor write operations are permitted.

Nevertheless, a client can obtain the value through notifications sent by the heart-rate-device. This behavior is specified in the *value* field of the descriptor attribute with *handle* 0x0028. In this case it is a Characteristic Configuration Descriptor (CCCD) which is readable and writable. Its *value* field is set to 0x0001 indicating that notifications are enabled.

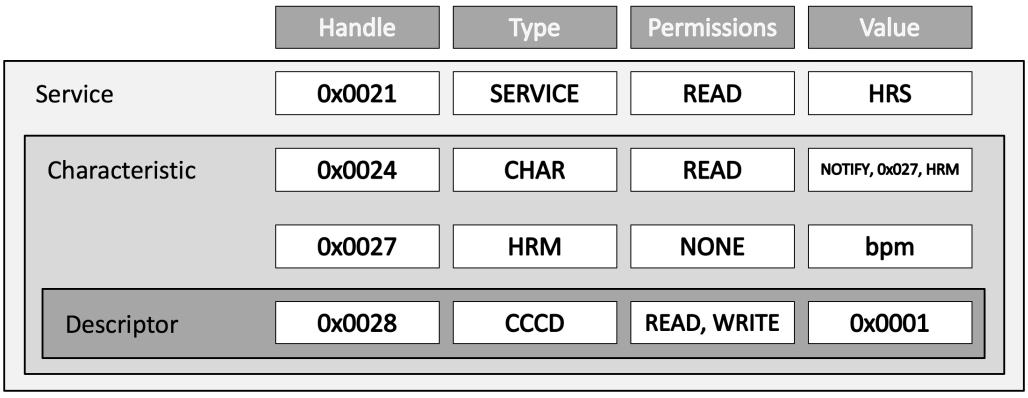

Figure 2.9: GATT Heart Rate Service (Adapted from [TCDD14])

2.2.7 GAP

The Generic Access Profile (GAP) is the highest level of the BLE stack and provides a framework consisting of roles, modes, and procedures which allow BLE devices to discover each other, broadcast data, establish secure connections, and perform other operations according to the predefined standard.

Roles

The GAP roles define the system's topology. Broadcaster and observer provide one directional communication, while peripheral and central roles are connection based and offer two-way communication.

- **Broadcaster and Observer:**
  A broadcaster only broadcasts data without requiring any confirmation or acknowledgment. It does not support connections to other devices. Public sensors like thermometers or humidity sensors that broadcast their sensor readings to any interested device are examples of a broadcaster. The observer role listens for advertising packets coming from broadcasters. They read the data without actually connecting to the device, like an alarm clock which displays the temperature data from a BLE broadcaster.
- **Peripheral and Central:**
  Similar to BLE broadcaster and observer roles, a peripheral sends advertising packets which may be received by a central device. Additionally, the central role is designed for initiating and managing multiple connections, whereas the complementary peripheral role is utilized by devices that wait for others to connect to it. After establishing a connection, further data can be transferred.

Modes and Procedures

Each role can be operated in different states, called modes or procedures. Depending on the actual goal of a device it can choose between different modes which in further consequence allow the other peer to perform a particular procedure. Among other things, these roles deal with the following tasks:

- **Discover devices:** How does a peripheral advertise its presence?
- **Connect to devices:** How does a central select the peripheral to communicate with?
- **Broadcast data:** How does a BLE device broadcast data to one or multiple observers?

2.3 Scientific Methodologies

The following subsections show ideas and approaches discussed in several papers including projects relying on multiple BLE roles or BLE gateways. Subsequently, there is also a part about BLE over IPv6.

2.3.1 Embedded Data Collection Based on BLE

The hop distance strategy in wireless sensor networks (WSNs) impacts the energy consumption of each device participating in the communication. Long-hop routing minimizes reception cost. However, routing over many short hops minimizes the transmission energy which increases with the communication distance. The optimal solution always depends on the environment where the sensor nodes are applied. In some cases the best approach might be a single hop solution, while in others a multi-hop solution is better suited. BLE enabled devices can be used for both scenarios.

In order to cope with the requirements of multi-hop networks the communication participants need to act as receiver and sender. To enable data transfer between nodes over several different paths the BLE unit should support multiple BLE roles. See subsection 2.2.7 for further details on BLE roles. Depending on the actual use case, the amount of data to be sent, and the available BLE roles, data can be transferred with or without an established connection between nodes. Examples of a connectionless and a connection-based BLE multi-hop network will be listed in the following paragraphs.

Connectionless BLE Multi-Hop WSN Approach

The first project, discussed in paper [CLH16] is about a WSN consisting of several devices, each acting as BLE broadcaster and receiver. They operate together to gain locational information of customers of a Taiwanese theme park in order to identify popular attractions and walking patterns of visitors. In the course of this project, visitors are provided with BLE bracelets that continuously broadcast data containing a unique ID. This information is detected by one of several deployed Bluetooth beacons. In addition to collecting these IDs, the BLE beacons forward the gathered information to other beacons until a dedicated PC is reached where all data is stored and analyzed. In the same way as the BLE beacons send data to the central PC, the PC which holds the information of each Bluetooth device may address specific beacons in order to change some settings, like the scan interval, for example. This request is sent again over the multi-hop network until it reaches the desired device.

Since data has to be relayed through several nodes until the desired destination is reached, all communication participants are being operated as BLE broadcaster and BLE receiver. In the applied scenario no connection between the nodes is established, so data that is transmitted further is embedded inside the advertisement packet. The packet has a limit of 32 Bytes at a time. Because of this limitation, timing relevant data, for instance, is only added when the data packet arrives at its destination. The BLE beacons consist of a CC2541 unit, enabling those devices to utilize broadcaster and receiver role. Each role remains active for five seconds before entering a 15 seconds long sleep mode to save energy. With this setup the Bluetooth beacons equipped with two 3000mAh batteries last for about 72 days according to this paper.

Connection-Oriented BLE Multi-Hop WSN Approach

The second project is called *Downhill* and is presented in paper [FMA+16]. It basically builds a Bluetooth Low Energy based multi-hop communication network using smartphones. The application area is early detection of landslides and slope failures. The *Downhill* smartphone application collects data from the phone's accelerometer and GPS unit and sends it to a server across a BLE network. Each smartphone is positioned in the range of another smartphone creating a wireless daisy chain. Data transmission is triggered the moment ground movements are detected.

This approach makes use of BLE's connection-based roles. The deployed smartphones utilize BLE central role to send the sensory data to a nearby phone acting as a peripheral device. As soon as the data is received as peripheral, the corresponding phone will additionally make use of BLE's central role to forward the data packet to the next phone in the chain. In this manner, the sensory data is propagated through the whole network until one central point is reached which collects and evaluates the data sets.

2.3.2 BLE Gateway Approaches

One way to enable embedded BLE devices to communicate over the internet is to use gateways. A gateway can transport data between two devices that would otherwise be unable to communicate with each other. In that sense, a gateway acting in between embedded BLE devices and servers has to support BLE and should have internet access.

One possible solution for providing connectivity to IoT devices is to provide a custom hardware gateway for each type of device. However, this approach seems unsuitable since the development of new hardware requires a long time to market and would lead to a massive deployment of new hardware in further consequence. Hence, companies like Intel, Wind River and McAfee have collaborated to provide services and tools to help IoT manufactures design dedicated multi-functional IoT hardware gateways that should support any specific IoT application [Int14]. However, this still involves the deployment of a new hardware ecosystem. Therefore, the gateway solution approaches described on the following pages were chosen based on the fact that they use hardware which is already widespread: smartphones.

Data Management in the Cloud for BLE Devices

The work illustrated by paper [TSPA16] presents a smartphone-based gateway solution responsible for retrieving data from wearable sensors over BLE as well as storing it to a central cloud storage in real-time. Data transmission from BLE devices to the gateway application takes 128ms on average. According to the paper, an application area suited for this real-time behavior is the healthcare domain. By constantly monitoring vital data such as pulse rate or blood oxygen saturation and providing feedback mechanisms via the Google Cloud Messaging (GCM) service the user always knows about his current situation. Additionally, third parties like doctors can make more precise diagnoses and carry out more appropriate treatments if they have access to these datasets. The service architecture is composed of the following four elements:

- **Producers:** Users who use BLE wearable sensors. These devices are responsible for continuous measurements of any kind. Examples are measurements of body temperature or pulse rate.
- **Gateway (Front-End):** The gateway part is implemented as an Android application and it automatically collects data produced by sensors, processes it, and stores it locally. Additionally, it monitors the measured values in order to notify the user in case of an emergency. The data encoded as JSON string is automatically sent to the back-end over HTTP on predefined time intervals or on demand. Moreover, the gateway allows the registration of new users and configuration of sensors with user dedicated rules. The process of adding new sensors is based on an XML scheme used to identify the BLE attribute values of the sensors' GATT characteristics. Furthermore, the smartphone gateway collects data from BLE devices based on this XML schema. The description of the XML schema consists of different tags such as the sensor-tag or device-tag used to identify the BLE devices and particular sensors that are registered in the system. Other tags are responsible for identifying parameters like measurement type, unit, or data format, for example.
- **Server/Cloud (Back-End):** The back-end is responsible for storing information regarding user devices and sensors as well as rules and historical data. Data is stored in JSON format based on the database MongoDB. Additionally, it features GCM for delivering push notifications to the phone.
- **Consumers:** In this context consumers are people who want to access the collected sensory data, e.g. medical personnel like doctors or the users themselves.

fabryq: Example of a Generic Gateway Approach

An even more generic gateway approach is shown by the project fabryq presented by paper [MERH15]. It addresses the problem of increased complexity which arises when developing systems responsible for establishing communication between embedded devices and servers over gateways. Most of the time, different programming languages complicate, hamper, and prolong the actual implementation. The fabryq architecture consists of three elements:

- **Embedded Device:** It provides sensory data over BLE.
- **Gateway (Front-End):** A smartphone (iOS based) acts as gateway as well as user interface.
- **Server/Cloud (Back-End):** The server collects, stores, and provides data, and manages commands, users, and devices.

A smartphones still acts as a middleman to connect devices supporting BLE to the internet. However, fabryq tries to simplify the development process of applications by using a protocol proxy programming model. Developers focus on writing application specific code with the provided fabryq JavaScript API, which is executed on the server side. In further consequence, the fabryq platform takes care of transferring data between embedded devices, the smartphone gateway, and the server.

A developer first has to register his phone in the fabryq iOS application. Subsequently, a scan for nearby BLE devices is initiated. The discovered entities can be selected and

information about their GATT tables is retrieved to learn about which data can be read or written to. Furthermore, either a new device type entry can be created on the server and phone or it is recognized as a previously created device type. Consequently, a developer can create an application by selecting his device and naming the new application. Code can be written in a local text editor or via the fabryq web editor. Fabryq programming consists primarily of issuing read and write requests to the content of the GATT attribute tables and writing callbacks to handle the returned data. The applications can be launched and controlled in a browser or on the fabryq iOS app using an agent interface. The application itself is constantly running in the cloud.

BLE commands that should be performed on the BLE devices are queued on the server by calls to the fabryq JavaScript API. The current implementation relies on constant polling to retrieve command logs for the smartphone app or the actual command results. If tasks are found on the server for the registered device of the user and the requested device is connected to the phone, the command is forwarded to the embedded device and the response is returned to the server over the gateway. To show data updates on the phone's display the API offers a method which triggers the server to send the phone an URL which basically consists of a webpage illustrating sensor specific data.

A user who wants to use the services fabryq offers has to install the mobile application and register his BLE devices. He then selects one of the available fabryc applications (programmed by a developer) and chooses which of his devices should be used for it. Depending on the selected application different use cases per device are possible.

2.3.3 IPv6 over BLE

Another possibility to connect BLE devices to the internet is to send IPv6 packets over BLE. In this case each device possesses its own globally unique IPv6 address, thus becoming recognizable as a single device over the internet and enabling true end-to-end connectivity between devices. Furthermore, no additional devices are needed to translate between different protocols.

The mechanisms for enabling an IP network on a BLE link were formalized by the Internet Engineering Task Force (IETF) and the BLE SIG. Finally, in October 2015, the IETF released the *RFC7668* standard which provides guidance on how to send IPv6 packets over BLE. The BLE link is similar to the IEEE 802.15.4 standard designed for resource constrained low power embedded systems. Additionally, many of the mechanisms defined for IPv6 over IEEE 802.15.4 can be applied to the transmission of IPv6 on BLE links. Among other things, this standardization requires header compression, fragmentation, and a reassembling process due to the smaller MTU size of BLE [NTS+15].

The flexibility of communicating over the IP layer is highly beneficial for supporting a wide variety of applications. The following project is a representative example where the end-to-end principle and the avoidance of an additional gateway device is advantageous:

Real-Time Acoustic Data Streaming over IPv6 and BLE:

The paper [YKKK15] proposes a way of streaming MP3 audio files sampled at 22050Hz with an IP enabled BLE headset. The approach is based on 6lo BLE. Additionally, the paper shows that it is also possible to provide continuous connectivity even when the device moves between different adjacent networks. IP relies on the end-to-end principle, hence session maintaining is possible. Finally, by using the handoff technology responsible for transferring sessions between different channels of a network, the BLE-IP based connection between streaming server and BLE headset can be maintained. However, this approach only works if enough IPv6 access points are available while the user with the BLE headset moves.

First open source implementations of communication stacks which fit all needs and requirements of the RFC 7668 standard are being elaborated. An example developed in the course of a thesis [Spo16] is the following:

IPv6 over Bluetooth Low Energy Using Contiki

Among other things, this thesis proposes the design of an IPv6 over BLE communication stack compliant to the RFC7668 standard. The stack fits the architecture of the Contiki OS and was implemented for the TI CC2650 SensorTag. Additionally, it is also shown that the communication stack developed in the course of this thesis is interoperable with devices supporting the RFC7668 standard and provides BLE nodes with network connectivity.

There are some challenges and problems which arise when dealing with IPv6 over BLE. The primary challenge is the complexity of communicating at the IP layer which enforces devices to run an IPv6 stack. In case of resource constrained embedded devices it may be difficult to support a full IP stack, especially for those devices which benefit from offloading work to more capable, less computationally and memory constrained devices. Sometimes an additional device is needed anyway acting inter alia as a user interface. Moreover, an IPv6 router/access point is required to enable IPv6 communication. The deployment of IPv6 is ongoing but still not nation-wide and it is not supported in every country. To estimate how widespread IPv6 actually is, Google is continuously measuring how many Google users access Google services over IPv6.[2] At the time of April 12, 2017, a total of 14,32% was measured.

[2]https://www.google.com/intl/en/ipv6/statistics.html (Accessed: 2017-04-12)

2.4 Commercial Products

There is a wide range of commercially deployed applications relying on the technologies discussed in section 2.3. A few of them will be discussed in the following paragraphs.

Wearable: Apple Watch

The Apple Watch features Wi-Fi and BLE functionality. Depending on the use case, one of the two modules is applied to enable retrieving data over the internet. If a known Wi-Fi network is in range the built-in Wi-Fi module enables direct internet connectivity. However, in order to enable data transfer over the internet even if there is no Wi-Fi network in range or to preserve energy, BLE is used. In this case an additional iOS device is required as a bridge between the Apple Watch and the internet. To be more specific, an iPhone or iPad with the corresponding iOS Version acts as gateway, retrieves data, and sends it back to the Apple Watch [App17a].

The Apple Watch and the iPhone are paired over BLE using an OOB pairing method. The watch displays an animated pattern which subsequently has to be captured by the camera of the user's iPhone. While doing so the pairing secret is exchanged. Once the BLE session is established and link layer encryption is enabled, the Apple Watch and the iPhone exchange an RSA 1280-bit key for encryption and ECDSA P-256 keys for creating signatures to offer an additional layer of security. For a more detailed explanation of security features please refer to [App17b].

Smarthome: Apple Homekit

Apple Homekit is a framework for communicating with home automation devices. iOS devices with the Homekit app installed are able to discover, configure, and control compatible devices. Both WiFi and BLE devices can be certified as HomeKit accessory by Apple. They have to support Apple's Homekit Accessory Protocol (HAP) that runs on top of the BLE or an HTTP/TCP/IP stack. On one hand, Homekit accessories may be controlled directly by an iOS device over BLE or WiFi. On the other hand, there is also the possibility of utilizing their services remotely. In this case an appleTV or iPad acts as gateway to forward data to the designated BLE or WiFi enabled device [WA14] [MA16].

Among other things, Homekits security relies on ECC Ed25519 for authentication between iOS devices as well as between iOS devices and accessories, and Secure Remote Password (3072-bit) protocol for exchanging keys which in further consequence are used to encrypt the session [App17b].

Gateway Solution: BluFi

BluFi is a Wi-Fi and BLE enabled gateway developed by Bluvision. Its BLE 4.1 chip supports BLE dual mode. Therefore, it can act as peripheral and central device. According to the BluFi specification sheet [Blu16b] it is able to handle multiple BLE connections at the same time with central devices when acting as peripheral. Additionally it supports multiple simultaneous BLE connections with peripherals when utilizing BLE central role. The gateway consists of an ARM Cortex M4 and M3 processor, on both running a RTOS.

Data collected over BLE is sent immediately to the Bluzone Cloud which enables real time data monitoring. The BluFi gateway comes in a handy format of 50mm x 38mm x 38mm and since it possesses an AC plug, it can be powered directly by a power socket. Regarding security it supports AES-128 Link Layer encryption of BLE. Additionally, communication from and to beacons is encrypted via RSA [Blu16b].

The initial setup and configuration can be done via the Bluzone App. BluFi is deployed at the Miami International Airport as a means of monitoring environmental conditions such as air conditioning temperature as well as other services like indoor navigation [Blu].

Real-Time Locating Service: Onyx Beacon

Onyx Beacon provides their customers with a Real Time Location Service (RTLS). The applied architecture consists of the following three elements:

- **BLE Beacon (Client):** The Enterprise Beacon of Onyx is a BLE device which permanently sends BLE broadcasts in a range of about 70m [Onya]. First of all, each asset that should be trackable has to be equipped with a BLE beacon. Furthermore, an accurate location estimation requires an additional grid of zone beacons placed, for instance, on walls and the ceiling. The more beacons are deployed, the more accurately assets can be located [Onyb]. Each beacon can be powered by four AA batteries which enable a continuous usage of the device of up to four years. Alternatively, the device can be powered over USB or a power plug. It has a waterproof enclosure that makes it suitable also for outdoor applications. Security is provided in form of AES-128 combined with Message Authentication Codes (MACs). Furthermore, it is possible to change the components of the beacon's advertising packets including UUID, major, and minor values [Onya].
- **Tracko App (Gateway):** The Tracko app, available for iOS and Android devices, can be used to track the location of the user's assets. The location of the assets equipped with BLE beacons is crowdsourced, meaning that each device containing the Tracko app autonomously scans for all available nearby beacons and reports their location to the cloud. If no internet connection is available, location relevant data is stored locally on the Android/iOS device and uploaded to the cloud as soon as the internet connection has been re-restablished [Onyb].
- **Tracko Cloud (Back-End):** The Tracko Cloud is the central data collection point. It provides the user with beacon related management services and analysis tools [Onya].

2.5 Smart Parking Without Smartphone

The application area of the EClient is smart parking. The less interaction between user and smart devices is required, the better and the more seamlessly the technology will be merged with the environment, improving usability. In this sense the EClient aims at improving usability by eliminating the need to pull a parking ticket or use ones smartphone to enter a parking lot. Hence, this section focuses on the explanation of smart parking solutions with a similar use case, or in other words: smart parking without smartphone. Both systems mentioned in the following use RFID technology. Specific information on how their access protocols are implemented is not public.

However, the common data transfer mechanism of projects based on RFID technology looks as follows. An RFID tag advertises object identifying data which a nearby RFID tag reader may receive. Consequently, the reader forwards the data to the back-end where different operations on a database are performed and application specific actions are triggered [Jec15].

Telepass

The Telepass is an electronic device used for collecting toll on motorways in Italy but it can also be used at some Italian car parks to gain access to the parking lot. The Telepass device is a semi-passive RFID transponder unit that can be mounted at the top of the vehicle's windscreen. The moment the car approaches one of the electronic booths optical procedures are used to recognize the type of the car and its numberplate. Subsequently, the booth emits a signal that is recognized by the Telepass which in turn replies to the request over a frequency of around 5,8 GHz. If the authentication process is successful, the gate opens and the driver can pass [P. 04, TEL].

Evopark

Evopark is a parking solution becoming more and more popular in Germany. It features parking at car parks and underground parking facilities. After an online registration over the website or the Evopark app, the user receives an Evopark parking-card. The card contains an RFID chip whose task it is to communicate with the MCUs of parking lot gates. When the gate opens the start or end time of the current parking session is recorded. To facilitate the search for a parking spot the corresponding smartphone app shows available parking spaces in realtime and offers direct navigation to the drivers [evo].

2.6 Related Work Conclusion

Table 2.4 summarizes several design aspects of the projects discussed in sections 2.3, 2.4, and 2.5. It shows the communication participants and what role they have to fill. Additionally, a rough survey of security relevant features is provided in summary form and the use case of each project is listed.

The project discussed in this thesis offers an embedded smart parking client. It makes the usage of smartphones obsolete when entering parking lots which support the COYERO access platform. There already exist a few smart parking systems which do not rely on the usage of smartphones. These, usually utilize RFID to transfer data wirelessly.

However, since RFID tags often only listen and respond regardless of who requested the signal, unauthorized access and modification of tag data is a major problem. Most of the time, these devices -especially passive RFID tags- are computationally constrained and also their memory is limited [Jec15].

In contrast, the EClient consists of a BLE chip and an additional dedicated MCU which makes it possible to implement more complex programs. Unlike the alternative RFID systems mentioned in section 2.5 the EClient in combination with COYERO access is a more generic system. This is due to the fact that the combined system does not only enable purchasing and redeeming parking entitlements, but also entitlements for other application areas, e.g. get access to hotels, redeem food vouchers at drive-through-restaurants, etc. A lot of different scenarios and use cases are possible.

A major advantage of using BLE is that it is compatible with smartphones. Direct interaction between them and the embedded device is hereby possible. Another distinguishing characteristic is that the smartphone the EClient is communicating with, acts as user interface, among other things. This allows managing several settings and keeping track of the user's entitlements. Furthermore, BLE supports connectionless and connection-based data transfer. Thanks to several security features on link layer level and the possibility to enhance security also on application layer, an adequate level of security can be reached.

Speaking of security, the EClient provides security goals such as data-integrity and authentication. In contrast to other commercially deployed distributed BLE systems, data confidentiality is not provided. However, this is not necessarily a drawback since challenge-response mechanisms are applied to provide means for authentication of the communication participants. If the authentication process cannot be verified the connection is simply aborted. Therefore, the absence of an additional encryption layer implies less computational effort and faster data transmission.

In BLE based wireless sensor networks it can be observed that in a few cases each BLE device supports multiple BLE roles. This behavior is especially useful to enable each device to initiate the communication to another device. This, in further consequence, raises the compatibility of BLE devices among each other and makes it possible to transfer data over multiple paths. Regarding the communication capabilities of the EClient, it also takes advantage of multiple BLE roles for maximizing compatibility to other BLE devices.

The EClient does not have internet access on its own. Hence, it initially needs a gateway device for retrieving relevant cloud data before being able to work autonomously. The gateway application itself is implemented as part of the COYERO Client Android application. Compared to other smartphone gateway applications, it has the disadvantage of only supporting devices with an integrated COYERO library. The gateway smartphone application acts as BLE central device. There are other gateway approaches which support more BLE roles, but in this specific scenario it is not required by the COYERO platform.

See chapters 3 and 4 to get an insight into the design and implementation aspects of the EClient and the gateway application. Results and findings will be evaluated in chapter 5 against projects of this chapter.

Project	**Communication Client**	**Communication Gateway**	**Security between Client and Gateway**	**Use Case**
Downhill [FMA+16]	BLE Peripheral, BLE Central	BLE Central	–	Slope Error Detection
Apple Watch [App17a]	BLE Peripheral	BLE Central	Confidentiality, Integrity, Authentication	Wearable/Smart Watch
Apple Homekit [MA16]	–	BLE Central	Confidentiality, Integrity, Authentication	Home Automation
IoT Cloud Data Management[TSPA16]	–	BLE Central	–	Generic Gateway App
Fabryq [MERH15]	–	BLE Central	–	Generic Gateway App
BluFi [Blu16b]	–	BLE Peripheral, BLE Central	Confidentiality	Generic Gateway
Onyx Beacon [Onya]	BLE Peripheral	BLE Central	Confidentiality, Authentication	Location Service
BLE beacon-based Network[CLH16]	BLE Broadcaster, BLE Receiver	BLE Central	–	Location Service
Telepass [P. 04]	RFID Tag	RFID Reader	–	Smart Parking
Evopark [evo]	RFID Tag	RFID Reader	–	Smart Parking
Prototype of this thesis	BLE Peripheral, BLE Central	BLE Central	Integrity, Authentication, Authorization	Smart Parking

Table 2.4: Overview of Projects Discussed in Chapter 2 [3]

[3]Entries marked as "–" are N/A or not mentioned in the corresponding references.

Chapter 3

Design

The following chapter will provide essential information on the design aspects of the EClient and the gateway application. First, the focus lies on the use case description, paying special attention to how the embedded client functions as part of COYERO access. In a second step several requirements are discussed. Subsequently, there is a part focusing on the analysis of several hardware pieces in order to find an appropriate embedded device that meets all requirements. Finally, a first concept for the prototype implementation will be presented. For more details regarding the implementation refer to chapter 4.

3.1 Use Case

The application area of COYERO devices is diverse. The COYERO Client device, on the one hand, acts as an interface for the user to purchase products or services in form of entitlements. This could be a shopping application, a restaurant app, or an app responsible for providing the user with access to infrastructures like buildings or parking lots. The COYERO Kiosk device, on the other hand, belongs to the entity responsible for providing the user of the client application with the actual item or service.

Regarding the topic of smart parking many mobile parking management apps already offer gated, valet, or street parking. However, there are only few solutions which do not depend on constant user-interaction with a mobile device. The embedded device designed and implemented during this thesis addresses this shortcoming. Although the device cannot connect to the internet itself and therefore occasionally needs a phone for synchronization purpose, interactions with the phone can be limited. A parking entitlement valid for a year, for example, would enable the embedded device to interact independently from the phone after an initial synchronization with it for that period of time. This, in further consequence, allows the user to get access to parking structures without any additional interaction.

This parking use case is described by figure 3.1. In the following example scenario the COYERO Client is implemented as a parking smartphone app used to purchase parking permits of any kind. The EClient could be mounted directly in the car, whereas the kiosk device could be in form of a smartphone carried by a parking guard or, in case of an automated parking environment, be part of the control unit of the gate itself.

In this sample scenario a user has purchased a long-time parking permit for a certain gated parking lot and wants to park his car at the dedicated garage. The moment he approaches his car and the EClient inside of the vehicle is in range of his smartphone, a synchronization process between the two devices is triggered, involving the COYERO Server. As soon as the car reaches the entrance of the garage, a corresponding kiosk device is responsible for checking if the EClient is in possession of the right entitlement to enter the parking lot. If the authentication of the device and the verification of the parking permit are successful the gate will finally open.

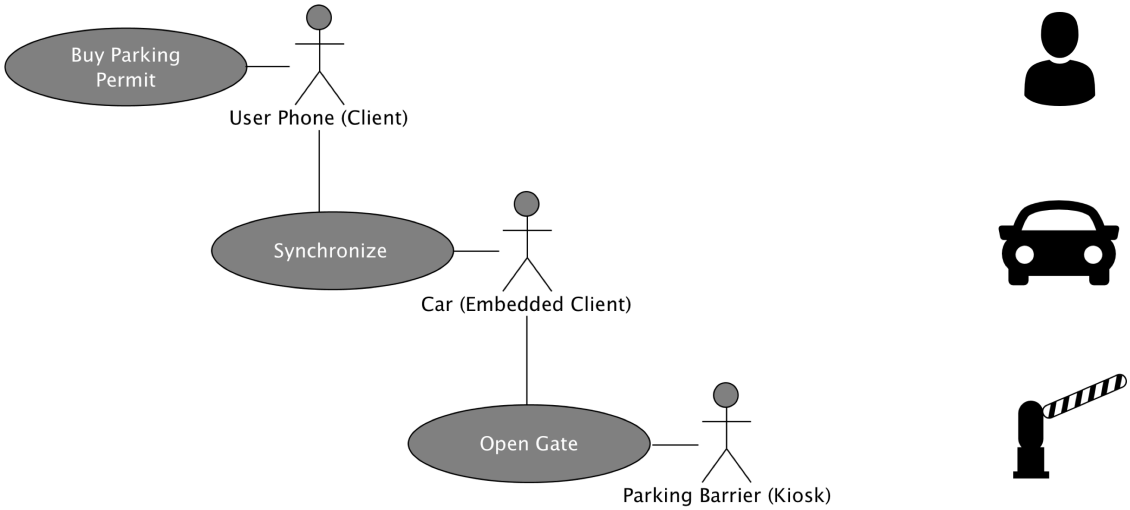

Figure 3.1: Use Case Description[1]

The procedure of getting access to the parking lot is processed offline, meaning that no further interaction between smartphone, COYERO Server, and EClient is required. The user basically just has to concentrate on reaching the garage without wasting time on pulling tickets or opening a specific app on his smartphone. The EClient takes care of the rest. The usage of BLE fits the use case. Since BLE supports a communication range of several meters, the entrance gate can be opened shortly before the car reaches the gate, enabling a smoother entry process. Thanks to the supported range, actions like leaning out of the car's window to operate a parking machine would become a thing of the past.

[1]Icons made by *Freepik* from *www.flaticon.com* (Accessed: 2017-04-27)

3.2 Requirements

All design aspects regarding the EClient and its gateway were selected in a way to fit the use case description discussed in section 3.1 as well as the requirements of the COYERO devices. Additionally, the COYERO Server had to be adapted in order to ensure compatibility with the gateway.

EClient Requirements:

- The application area of the EClient is smart parking. Therefore, only "Non-Consumable" Entitlement-Tokens should be stored on the EClient which are valid for a certain time and redeemable frequently during that period. As long as an Entitlement-Token is valid the EClient may function independently from the COYERO Client, avoiding continuous usage of a mobile phone which in further consequence would affect the usability in a negative way.
- Communication to COYERO Clients and COYERO Kiosks is established over BLE. Since clients operate in BLE central and kiosks in BLE peripheral role, both roles have to be supported by the EClient. With this setup it is basically possible to communicate to all kinds of BLE devices (see figure 3.2). First of all, communication with connection-oriented BLE devices operating in central or peripheral mode is feasible and, moreover, data can be received by BLE broadcasters or sent to BLE receivers via advertising packets without establishing a connection. For a further explanation on BLE roles see subsection 2.2.7.

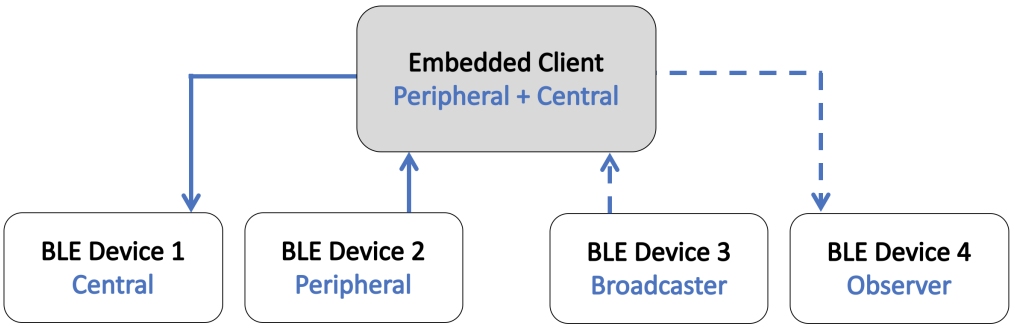

Figure 3.2: Support of Multiple BLE Roles

- The EClient should provide optical feedback depending on its current state, e.g. it should be possible for the user to see if redeemable entitlements are already stored on the device.
- Certain data packets like Auth-Tokens and Entitlement-Tokens have to be stored locally. It should be possible to store at least 5 Entitlements (610-940 Bytes each) and one Auth-Token (193-195 Bytes).
- A hardware reset mechanism that includes the deletion of stored data is required. It should be triggered by pressing a button.

- The device's compatibility should be limited to precisely one COYERO Client user device. The EClient should only accept Entitlement-Tokens from the specific client device which was responsible for retrieving the EClient's Auth-Token.
- There should be a mechanism to discard expired or about to expire Auth-Tokens and to renew them.
- In order to check the validity period of tokens, the EClient needs to know the current date and time.
- To avoid fraudulent data manipulation and to guarantee compatibility with the current COYERO protocol, the embedded client needs to support cryptographic primitives like ECDSA and SHA256.
- Offline mode: The moment tokens are on the device, it should work independently from the smartphone gateway for the validity period of the tokens.

Gateway Requirements:

- The gateway should be an Android phone with BLE functionality. The code responsible for acting in between EClient and server has to be implemented as part of the COYERO Client library.
- The gateway should support BLE central role to remain fully compatible to COYERO Kiosk devices, which should not be affected by any modifications.
- To support all necessary BLE features a minimum Android version of 5.0 is required.

COYERO Server Requirements:

- The provided server functionality has to be extended as a means of ensuring compatibility with the EClient and its gateway by providing calls responsible for the retrieval of Auth- and Entitlement-Tokens.

3.3 Architectural Description

To implement the use case described in section 3.1 the COYERO architecture, already explained in section 2.1, had to be extended. The red areas in figure 3.3 mark where additional features were added or where changes were applied:

- The embedded client is a secure BLE device capable of communicating to other devices irrespective of their BLE role. It acts in between COYERO Client (BLE central) and COYERO Kiosk (BLE peripheral).
- The COYERO Client library had to be extended in order to act as gateway for transmitting data between EClient and COYERO Server while preserving the compatibility to kiosk devices. Additionally, the client application had to be changed and updated in regard to its user interface.
- New methods on COYERO Server-side were implemented.

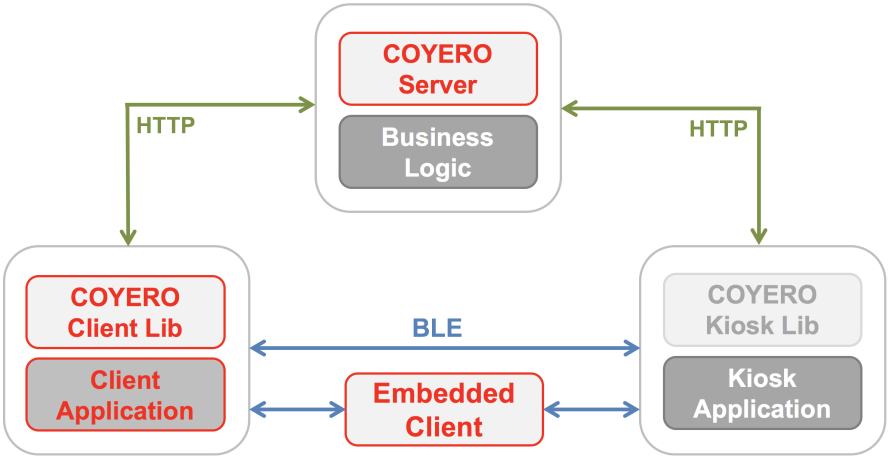

Figure 3.3: Extended COYERO access Architecture

From a more technical perspective and taking into account the just discussed extended COYERO architecture, the use case described in section 3.1 would include the following main tasks illustrated by figure 3.4. Users utilizing a COYERO Client application first have to authenticate themselves. Subsequently, the COYERO Server issues a device bound Auth-Token. Afterwards, users are allowed to buy digital vouchers for products and services. For each voucher purchased a dedicated Entitlement-Token is issued by the server. These entitlements can be sent by client devices to corresponding kiosk devices to redeem them by involving again the server.

The lower part of the figure 3.4 involves the EClient which acts in between COYERO Client, Kiosk, and Server. It basically talks to the server using a COYERO Client as gateway and sends data to kiosk devices like any regular COYERO Client device.

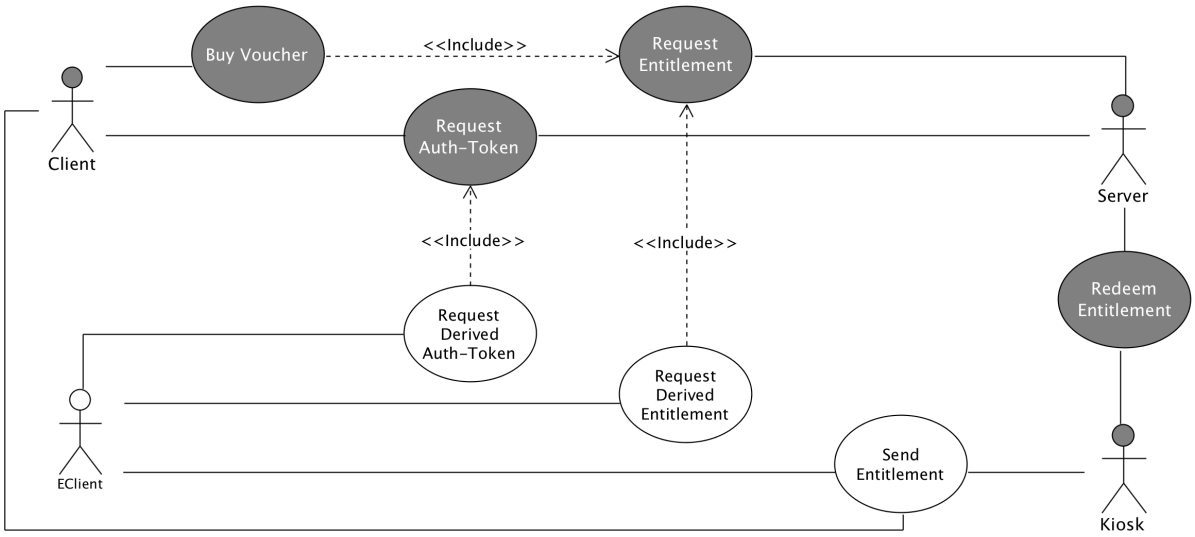

Figure 3.4: Overview Main Tasks

3.4 Hardware Selection

In the subsequent paragraphs different devices are analyzed in order to find the appropriate hardware that meets all requirements listed in section 3.2. Several development boards will be compared, including Silicon Labs' *Thunderboard React*, Texas Instrument's *CC2650*, and Adafruit's *Feather M0 Bluefruit LE*. Table 3.1 gives an overview on several specs of these devices. Subsequently, distinctive features of each board will be listed. In addition, the Bluetooth Low Energy module BLE112 from Bluegiga (see [Blu14a]) is named before summarizing all findings.

Attributes	**Thunderboard React** [Sil]	**CC2650 (Sensor Tag)** [Tex15] [Tex16b]	**Feather M0 Bluefruit LE** [Ada16a]
CPU	ARM Cortex M4	ARM Cortex M3	ARM Cortex M0+
Flash Memory	256 KB	128 KB	256 KB
BLE Version	4.2	4.2	4.1
Development Environment	Simplicity Studio	Code Composer Studio	Arduino IDE

Table 3.1: Development Boards Comparison

Thunderboard React [Sil]

- This board consists of a BGM111 BLE 4.2 module.
- The BGM111 module features a CRYPTO unit consisting of a cryptographic algorithm accelerator and a random number generator. It supports the usage of the following cryptographic primitives:
  - **SHA**: SHA-1 and SHA-2 (SHA-224 and SHA-256)
  - **AES**: 128bit or 256bit including ECB, CTR, CBC, PCBC, CFB, OFC, CBC-MAC, GMAC, and CCM mode
  - **ECC**: Available over both GF(P) and GF(2^m) P-192, P-224, P-256, K-163, K-233, B-163, and B-233
- Thunderboard React stores its cryptographic keys in flash. The exact location is handled by the stack itself.

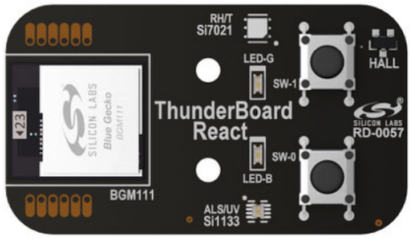

Figure 3.5: Thunderboard React [Sil]

CC2650 [Tex15], [Tex16b]

- It supports many wireless standards such as BLE (version 4.2), ZigBee, 6LoWPan, and RF4CE. Additionally, it is available as development kit (figure 3.6) or as sensor tag version (figure 3.7).
- **TI-RTOS:** It is Texas Instruments' real-time operating system usable within TIs Code Composer Studio with this board. It comes with networking stacks, power management tools, memory management, inter processor communication, scheduler, and device drivers, facilitating the development on the MCU.
- **AES Cryptography unit:** The AES-128 module of the CC2650 is implemented in form of a co-processor and offers additional security features. It is able to detect *key-load* and *Advanced High-performance Bus* errors and abort *Direct Memory Access* operations, for instance. The AES keys are stored securely on the hardware in the so-called *key store module* [Tex16a].
  However, since cryptographic primitives become weaker over time and the key length is an important security parameter, more and more companies encrypt their data by using 256 bit keys. NIST, the computer security resource centre, claims AES-128 bit is secure for the next decades [Blu16a]. Furthermore, compared to AES 256, the 128 bit variant has the advantage of enabling a faster encryption process due to its shorter key.

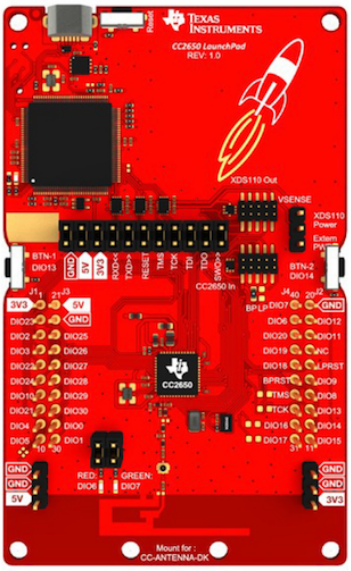

Figure 3.6: CC2650 Dev. Board [Tex16b]

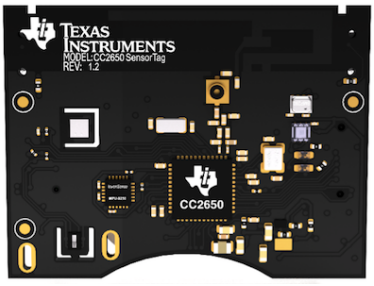

Figure 3.7: CC2650 Sensor Kit [Tex15]

Adafruit Feather M0 Bluefruit LE [Ada16a]

- This board supports the Arduino platform, which has a large community. Engineers, hobbyists, and professionals create their projects with Arduino. Due to this, several pre-existing Arduino software libraries facilitate the coding effort and safe time. Regarding software based cryptographic libraries *micro-ecc* [Mac16] features ECDSA, while *Cryptosuite* [Kni10] offers SHA based hashing, for example.
- There is the possibility to connect a lithium polymer or lithium ion battery to the JST jack and recharge it over USB. This in turn enables the device to last longer than when using a conventional CR2032 coin cell battery.
- The device consists of an additional Nordic nrf51 BLE chip which can be controlled via AT commands. Among other things, this enables controlling the BLE functionality of the on-board-chip and making use of the internal random number generator. However, Nordics firmware version S110 in cooperation with Adafruit's software revision version 0.7.0 supports BLE peripheral role only.

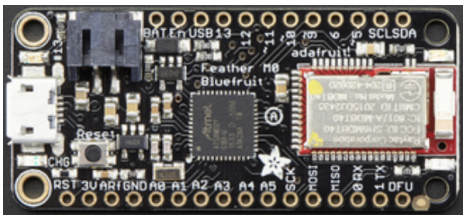

Figure 3.8: Feather M0 Bluefruit LE [Ada16a]

Bluegiga BLE112 Chip

The Bluegiga BLE112 is based on the TIs CC2540 chip. It integrates a BLE 4.0 module and processing unit into one chip. Additionally, its AES encryption core features 128 bit AES encryption and decryption. Furthermore, it also has hardware interfaces to connect to different peripherals and sensors. BLE central and peripheral roles are supported [Blu14a].

The event based programming language *BGScript* is used to interact with the module. On one hand, BGScript applications allow controlling the Bluegiga Bluetooth Smart module without the need of any external MCU. On the other hand, if the amount of available hardware interfaces, memory, or processing power limits the implementation of certain applications directly on the on-board MCU, there is the possibility of making use of the BGAPI protocol. It may be utilized by external devices to use features of the BLE112 module. In this case communication between the BLE112 module and the external device is established over the physical UART and USB interfaces [Sil15].

Figure 3.9: Bluegiga BLE112 Chip [Blu14a]

Selection Conclusion

The Bluegiga BLE112 chip was chosen as Bluetooth unit for the EClient. Not only is the chip operable as peripheral and central device, it also offers the possibility to be operated via the provided Bluegiga BGAPI and the UART interface by an external device less constrained in terms of computing power and memory.

The Feather M0 was picked as additional host device. One reason for this decision was that the implemented code should remain as portable as possible. Since the Feather M0 board is based on the widely spread Arduino platform, the code could be reused in the same or at least in similar form on other comparable Arduino boards. Additionally, thanks to Arduino's popularity lots of software libraries of different kind exist, facilitating the development of applications.

In order to control the BLE112 module from an Arduino based board like the Feather M0 the BGLib Arduino library (see [Row14]) was included in the project which uses C-wrappers to access specific functionalities of the BLE112. Communication between the two devices is established over the Serial UART interface.

Another advantage of the Feather M0 pertains to its small physical dimensions of 2in x 0.9in allowing it to be embedded almost everywhere. The possibility to attach an additional lithium polymer or lithium ion battery over the JST jack makes it even more portable. Another advantage of this setup is that the Arduino platform, including its compilers, is not subject to payment. In contrast, the Thunderboard React and the board of TI need the IAR Embedded Workbench compiler which are not free of charge (as of October 2016). Regarding the hardware of the Feather M0 board, its nRF51 chip offers

a hardware number generator that could be used for creation of cryptographic keys or nonces.

A downpoint of this setup is that the Feather M0 does not possess any internal secure memory section out of the box. Therefore data has to be saved on the internal flash memory. Additionally, there is no Arduino supported API to access the AES hardware engine of the BLE112 chip which could be used to wrap cryptographic keys generated in software. The possibility of extending the board with external secure hardware is addressed in subsection 6.2.3. In case of the COYERO protocol, data structures to hash, sign, and verify are small. Hence, the on-board Cortex M0+ processor of the Feather M0 is fast enough to handle cryptographic primitives in software without apparent delay.

With BLE version 4.1 BLE-multi-role behavior was introduced and added to the specification. It enables devices to act in different BLE roles at the same time [Blu10, Blu14b]. BLE chips that use a BLE 4.0 stack do not support this feature and even in newer BLE units this functionality is not always supported. In case of the EClient the absence of this particular feature represents no problem since the device does not have to communicate to COYERO Clients and COYERO Kiosks at the same time. Additionally, since not all BLE roles consume the same amount of energy, the simultaneous usage of different roles and multiple ongoing connections would probably raise the power demands of the EClient. A more detailed description on how the EClient utilizes different BLE roles can be found in the following section.

3.5 Embedded Device as State Machine

This section illustrates all possible states and transitions of the EClient as a means of getting a first overview of its behavior during program execution. As it was explained in section 3.4 the EClient basically consists of two complementary components, the Feather M0 board and the BLE112 module. While the Feather M0 acts as control unit and state machine, the main tasks of the BLE112 module are executing different BLE commands and forwarding their results and received responses back to the Feather M0.

Two different types of states are distinguished: Client-states and BLE-states. The Client-states are used to tell which kind of token the EClient is ready to receive, distinguishing between Auth-Tokens and Entitlement-Tokens. The BLE-states, on the other hand, provide a possibility of determining in which BLE role the device is currently being operated. Figure 3.10 shows all Client-states and transitions, while figure 3.11 presents all possible BLE-states and corresponding transitions.

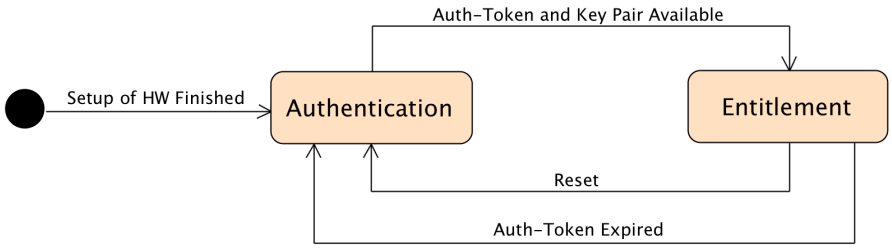

Figure 3.10: Client-states

The initial Client-state is always set to *Authentication* after booting and initializing the hardware. If there is no valid ECC keypair or Auth-Token on the device the Client-state is left unchanged, whereas the BLE-state is set to *Advertising*. This in turn makes the EClient advertise its presence with dedicated advertising packets, indicating that it is in need of an Auth-Token. Otherwise, if an Auth-Token and a valid keypair are already stored locally, the Client-state is set to *Entitlement*. This would lead the device to send another type of advertising packets than in *Authentication* state pointing out that it is ready to receive Entitlement-Tokens.

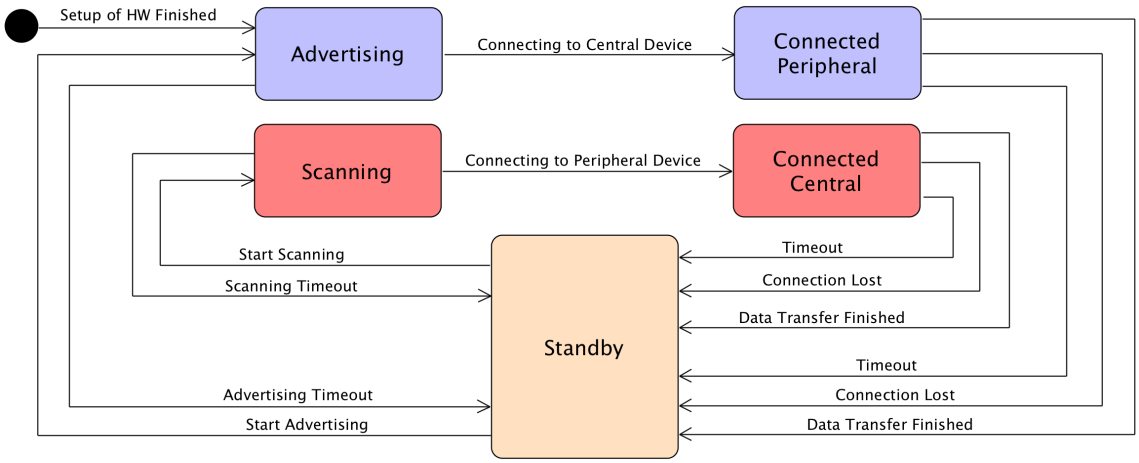

Figure 3.11: BLE-states

Each time a central device connects to the EClient acting as BLE peripheral, the BLE-state switches from *Advertising* to *ConnectedPeripheral* and data can be transferred between the two devices. Two scenarios are thereby distinguished:

- An Auth-Token can be transferred while the Client-state is set to *Authentication*. After receiving a valid Auth-Token it changes to *Entitlement* and the BLE-state is set back to *Advertising*.
- Entitlements can be transferred if the EClient is being operated in Client-state *Entitlement*. It only returns to state *Authentication* if the device has been reset or the stored Auth-Token is about to expire or has already expired.

As soon as the first entitlement is available on the device the EClient starts to constantly switch between the BLE-states *Advertising*, *Standby*, and *Scanning*. Thus, it can either be connected to a nearby central device (COYERO Client) to get additional entitlements, save energy, or connect to a peripheral device (COYERO Kiosk) to redeem its entitlements.

The Standby state can be considered a transition state reachable from the BLE-states *Advertising* and *Scanning* when no compatible COYERO Client or Kiosk device was found during a predefined time interval. Once in *Standby* state, the subsequent BLE state is immediately set, alternating between *Advertising* and *Scanning*. If no BLE connection is established for a longer period, the time the EClient remains in Standby state is increased. Since the BLE112 module is not used during Standby-state, the overall energy

consumption is reduced. Moreover, the *Standby* state can be reached from BLE states *ConnectedPeripheral* and *ConnectedCentral* if one of the following conditions is met:

- All data is transferred correctly and one of the devices performs a disconnect.
- The connection is lost.
- A data timeout occurs, meaning that in a certain time span no data packets were received by the EClient.

Two LEDs are used to indicate which Client-state and BLE-state the EClient is in. Depending on the specific situation an LED is constantly toggled or simply turned on or off. See table 3.2 to get an overview of all possible LED combinations.

LED Red	**LED Blue**	**Client State**	**BLE State**
Off	Toggles at 2Hz	Authentication	Advertising
Off	Toggles at 1Hz	Entitlement	Advertising
Toggle 1Hz	Off	Entitlement	Scanning
On	Off	Entitlement	Connected Central
Off	On	Authentication	Connected Peripheral
Off	Off	Entitlement	Standby
On	On	-	-

Table 3.2: Visual State Indication

Both LEDs will be turned on if the user presses the reset button. After pressing it for three seconds, a reset mechanism is triggered. Consequently, all tokens will be deleted before the program itself is reset.

3.6 Android Gateway Application

The BLE gateway is implemented in form of a Java application for the Android platform. The usage of Android devices as a gateway offers two big advantages. First, they are widely spread, hence a lot of different use cases can be derived and applied. Second, with regard to usability, this limits the difficulty to learn new UIs and commands for users since they are already familiar with their own devices.

Regarding Android's BLE stack there is to say that Android introduced a built-in platform support for the BLE central role with version 4.3. The possibility for scanning only for specific devices was introduced with Android 5.0. Additionally, since version 5.0, BLE peripheral role is supported by the Android BLE stack [Anda].

The gateway application is embedded into the COYERO Client library, which enables the Android device to act in BLE central role when communicating to the EClient which in turn utilizes BLE peripheral role. By doing so the COYERO Client application stays compatible to kiosk devices which utilize BLE peripheral role.

In order to distinguish between a COYERO Kiosk and an EClient device before even establishing a connection, the gateway application listens to different pre-defined advertisement UUIDs. Altogether there are three advertisements the COYERO Client application listens to, each of which includes a specific service in case of a successful connection establishment.

- **COYERO Kiosk advertisement:**
  The COYERO Client application redeems Entitlement-Tokens at the kiosk device. This functionality was already implemented.
- **EClient advertising in Client-state Authentication:**
  The COYERO Client application acts as gateway for the EClient and retrieves a server-issued Auth-Token for it.
- **EClient advertising in Client-state Entitlement:**
  The COYERO Client application acts as a gateway for the EClient and retrieves server-issued Entitlement-Tokens for it.

Chapter 4

Implementation

In the current chapter details about the prototype implementation are discussed. In addition to information regarding the development environment, including libraries and several settings, also details about the circuit schematic of the EClient will be given. In the next step the structure of the developed programs for the EClient written in C/C++ and the gateway application written in Java will be explained (see section 4.4) before taking a closer look at data structures which the EClient has to handle (refer to section 4.5). Additionally, different encoding schemata that come into play during data transfer will be presented in section 4.6. Finally, the program flow of the implemented program is explained in section 4.7 and shown by means of several sequence diagrams.

The software for the EClient and the Android gateway was developed on an Apple Macbook Air (2011) running macOS Sierra as operating system.

4.1 Development Environment – Embedded Client

The Eclipse based Arduino IDE Sloeber (version 3.1) for macOS was used to work on the Adafruit Feather M0 board. Adafruit boards are not supported by default. However, by adding the URL *https://adafruit.github.io/arduino-board-index/package_adafruit_index.json* in *Preferences>Arduino>Locations* new Adafruit boards and updates to existing boards are picked up automatically. The URLs point to index files that the IDE uses to build the list of available and installed boards.

After adding the URL, several Adafruit board packages will be available under *Preferences>Arduino>Platforms and Board*. Since the Feather M0 uses an ATSAMD21 chip, support for SAMD devices has to be added as well. This is done by ticking the checkbox in *Preferences>Arduino>Platforms and Boards>adafruit>Adafruit SAMD Boards>1.0.xx* (1.0.13 or higher). Once this initial setup is complete, it should be possible to upload code to the board. However, in some cases during the upload of the program the following error appears, indicating that the flash programming utility *bossac* cannot find the device.

```
Comport is not behaving as expected
Using comport /dev/cu.usbmodemFD121 from now onwards
Ending reset
Launching
/Applications/sloeber.app/Contents/Eclipse/arduinoPlugin/tools/arduino/bossac
    /1.6.1-arduino/bossac --port=cu.usbmodemFD121 -U true -i -e -w -v /
    Applications/sloeber.app/Contents/MacOS/workspace/BLEClient/Release/
    BLEClient.bin -R
Output:
No device found on cu.usbmodemFD121
bossac finished
```

The arduino webpage mentions the following in case of the Arduino Due:

> "*Opening and closing the "Native" port at the baud rate of 1200bps triggers a "soft erase" procedure: the flash memory is erased and the board is restarted with the bootloader*" ([Ard17]).

Unfortunately, it seems that the IDE fails to adapt the baud rate automatically in case of the Feather M0 board. Therefore, it is necessary to manually open the serial at 1200Bd before launching *bossac*. That will trigger the reboot and consequently the start of the bootloader on the chip. By doing so, *bossac* is able to find the bootloader. The terminal command to change the baudrate to 1200 for the port, mentioned in the error log above, looks as follows in macOS Sierra: *stty -f /dev/cu.usbmodemFD121 1200*. Eventually, it should be possible to upload programs to the board.

Additional Arduino Libraries

All external libraries included in the EClient project are listed and described in the next few paragraphs:

- **FlashStorage:**
  The non-volatile flash memory is generally used to store program code which cannot be changed during runtime. However, this library offers functions to store variables and C-structs to flash memory and retrieve the data during runtime. This library is especially useful for devices which do not feature additional storage units. However, a prerequisite to use it is that the MCU features an ATSAMD21 CPU [Mag16].
- **micro-ecc:**
  This library holds a lightweight ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit architectures. It supports secp160r1, secp192r1, secp224r1, secp256r1, and secp256k1. secp256r1 is used for the EClient's program [Mac16].
- **Cryptosuite:**
  It is a cryptographic library for Arduino specialized on secure hashing and hashed message authentication. It features SHA-1, SHA-256, HMAC-SHA-1 and HMAC-SHA-256. The EClients utilizes SHA-256 during program execution [Kni10].
- **BGLib:**
  This library acts as a C-wrapper for the event-driven BGLib protocol used to control the Bluegiga BLE112 module. BLE peripheral and central roles are supported [Row14].

- **Adafruit_BluefruitLE_nRF51:**
  This library provides methods for operating with the integrated nRF51 module of the Feather M0 board over AT-commands (e.g. for accessing the board's internal RNG) [Ada16b].

Additionally, the following changes were applied:

- The Arduino SoftwareSerial library does not work with the Feather M0. The problem is that the Feather M0 board is a SAMD device and the SoftwareSerial in the Arduino package currently only supports AVR devices. Hence, HardwareSerial is used to communicate with the BLE112 module over the BGLib library interface.
- In order to pass larger data packets over the serial interface the SERIAL_BUFFER_SIZE constant inside RingBuffer.h was increased to 1024.

4.2 Development Environment – Android

Android Studio IDE version 2.2.1 (for macOS) was used for the development of the gateway application. Table 4.1 shows the minimum and target SDK versions of the application. These parameters are specified inside the AndroidManifest.xml. It is a file in the Android platform that gives information of the app to the Android system, including a description of its components, different permissions, etc. [Andc].

Attribute	API level	Version	Code name
minimum SDK-version	21	5.0	Android Lollipop
target SDK-version	22	5.1	Android Lollipop

Table 4.1: AndroidManifest – SDK version

The gateway code makes use of the Android BLE stack. Furthermore, it utilizes the following external libraries:

- **com.google.code.gson:gson:2.3.1:**
  This is a Java serialization/deserialization library that can convert Java objects into JSON and back [Goo16].
- **com.android.volley:volley:1.0.9:**
  Volley is an HTTP library for easy and fast networking. Some of its advantages are its support for request prioritization, the possibility to connect to multiple concurrent networks, automatic scheduling of network requests, an out of the box support for strings, images JSON, etc. [Andb].

The gateway application was embedded into the COYERO Client Android library which provides the COYERO Client device with API calls for NFC/BLE/QR-Code communication with COYERO Kiosk devices. Additionally, the COYERO Client library holds methods responsible for communicating with the COYERO Server.

4.3 Circuit and Wiring

This section illustrates the circuitry of the EClient. It basically consists of a Bluegiga BLE112 module and an Adafruit Feather M0 board. Section 3.4 explains why this particular hardware setup was chosen for the EClient.

Figure 4.1 gives an overview on the pin layout of the Bluegiga BLE112 module, while table 4.2 describes the corresponding pins [Blu14a].

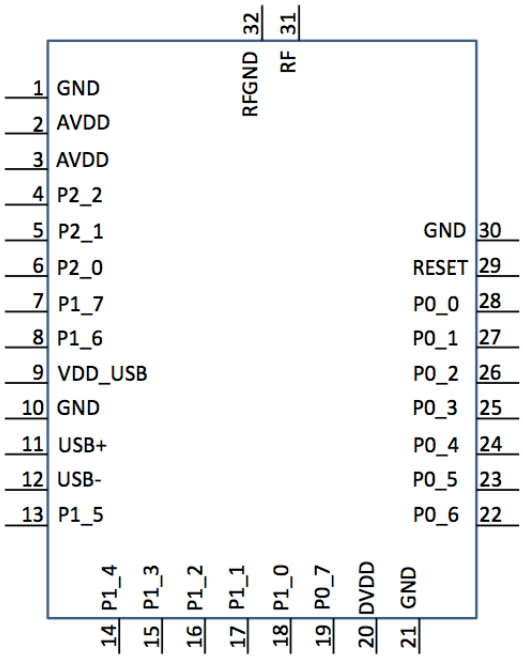

Figure 4.1: BLE112 Pinout [Blu14a]

Pin	Number	Description
RESET	29	Active low reset
GND	1, 10, 21, 30	Ground
DVDD	20	Supply voltage 2V - 3.6V
AVDD	2, 3	Supply voltage 2V - 3.6V
VDD_USB	9	Supply voltage 2V - 3.6V
USB+	11	USB data plus
USB-	12	USB data minus
PX_Y	4-8, 13-19, 22-28	Configurable I/O port

Table 4.2: Pinout Table BLE112 (Adapted from [Blu14a])

In order to make the BLE112 module more easily connectable to other devices like the Feather M0, it was used in form of a breakout board provided by CISC Semiconductor. The left part of figure 4.2 shows the BLE112 breakout board, while the right part consists of the Feather M0. It should be noted that the pins P0_5 (RX) and P0_4 (TX) of the BLE112 chip are responsible for the serial communication.

Two LEDs and a tactile switch were added to the board. The switch is used to trigger the deletion process of the user data. Subsequently, the GPIO pin connected to the RESET pin is set to LOW to restart the device. The LEDs provide a way to optically distinguish which state the EClient is currently in (see table 3.2 for further information).

Regarding the Feather board only those pins actually in use are illustrated by the figure. This is due to two reasons. First, this provides a better overview and ensures the illustration is not overloaded. Second, it puts an emphasis on the fact that also other similar Arduino boards could be used instead of the Feather M0.

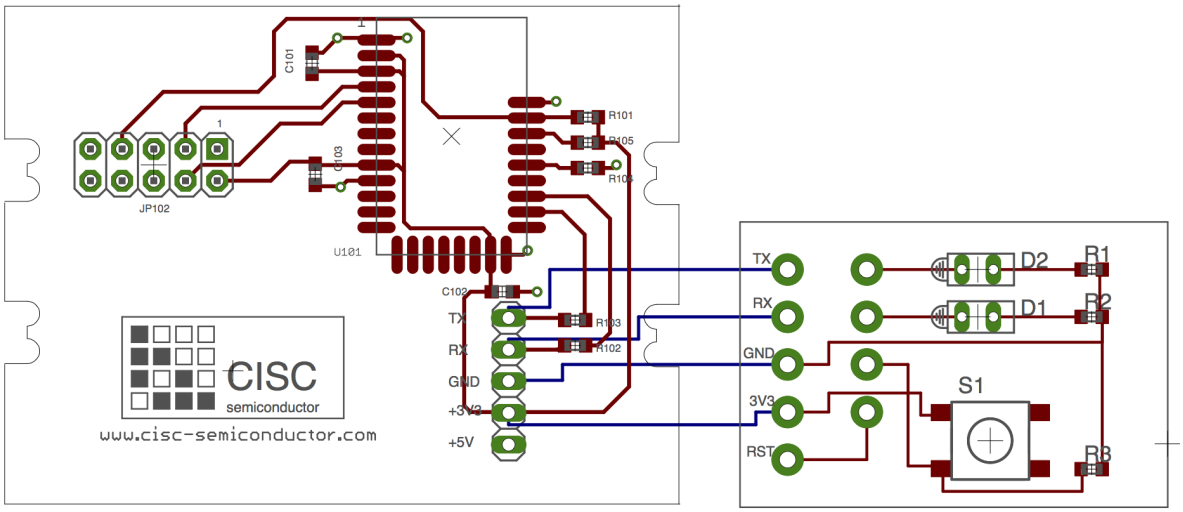

Figure 4.2: Prototype Board

Signals between the two devices are transferred over the Serial UART interface. They communicate serially at a transistor-transistor logic (TTL) level meaning that signals between the communication participants always remain between the limits of 0V and the device's power supply V_{CC}, which, in case of the Featherboard M0 and the BLE112, is 3.3V.

Figures 4.3 and 4.4 demonstrate what the transferred data between the devices might look like. The signals were measured with an oscilloscope[1] while the BLE112 was connected to the Feather M0 (as illustrated in figure 4.2) and some data was transmitted. Channel 1 measured the voltage at the TX pin and Channel 2 was responsible for measuring the voltage level at the RX pin of the BLE112. Both figures clearly demonstrate that the signals are always in between 0V and 3.3V.

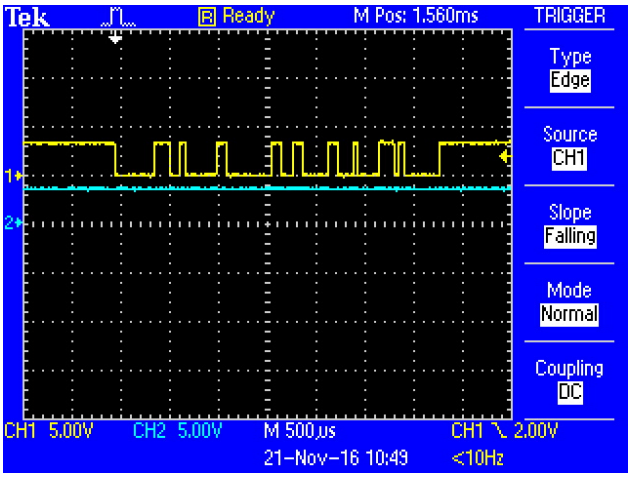

Figure 4.3: BLE112 Breakout board – TX pin (CH1)

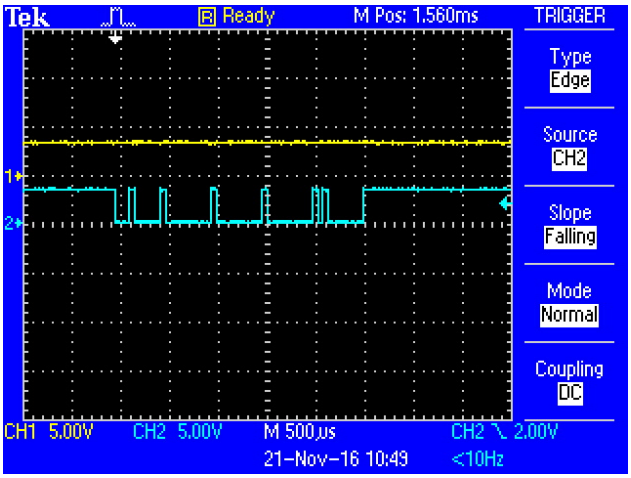

Figure 4.4: BLE112 Breakout board – RX pin (CH2)

[1] Tektronix TDS2024B

Figure 4.5 provides a schematic view of the BLE112 breakout board connected to the Feather M0 board, focusing on its individual parts and wiring.

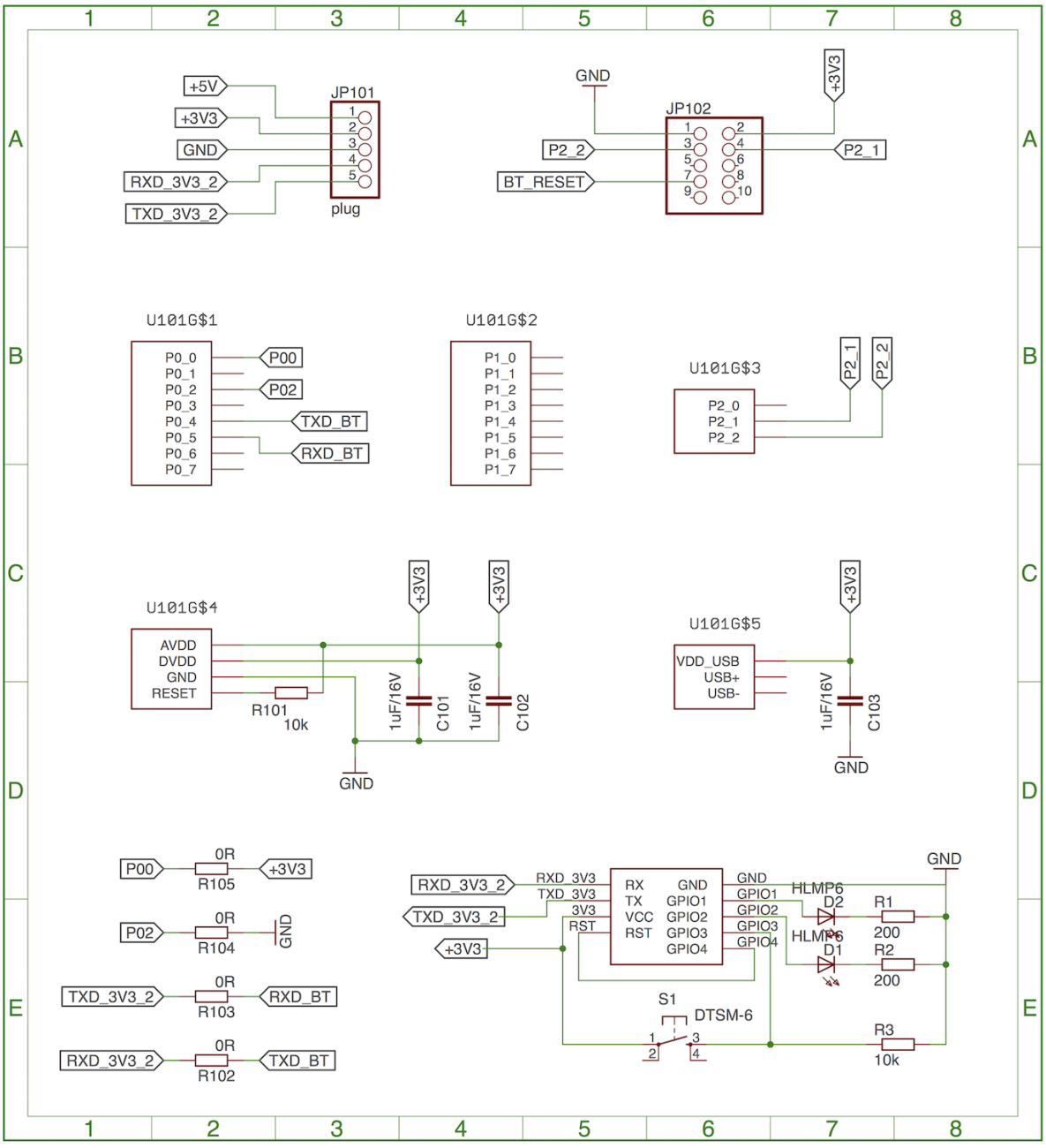

Figure 4.5: Electronic Schematic

4.4 Software Structure

This section provides information on the software of the developed prototype comprising charts of the embedded client and Android gateway application.

Embedded Client

The class diagram in figure 4.6 gives a brief overview of the most important files and their dependencies of the EClient-program written in C and C++. In order to maintain a better overview return types and parameters of functions are omitted.

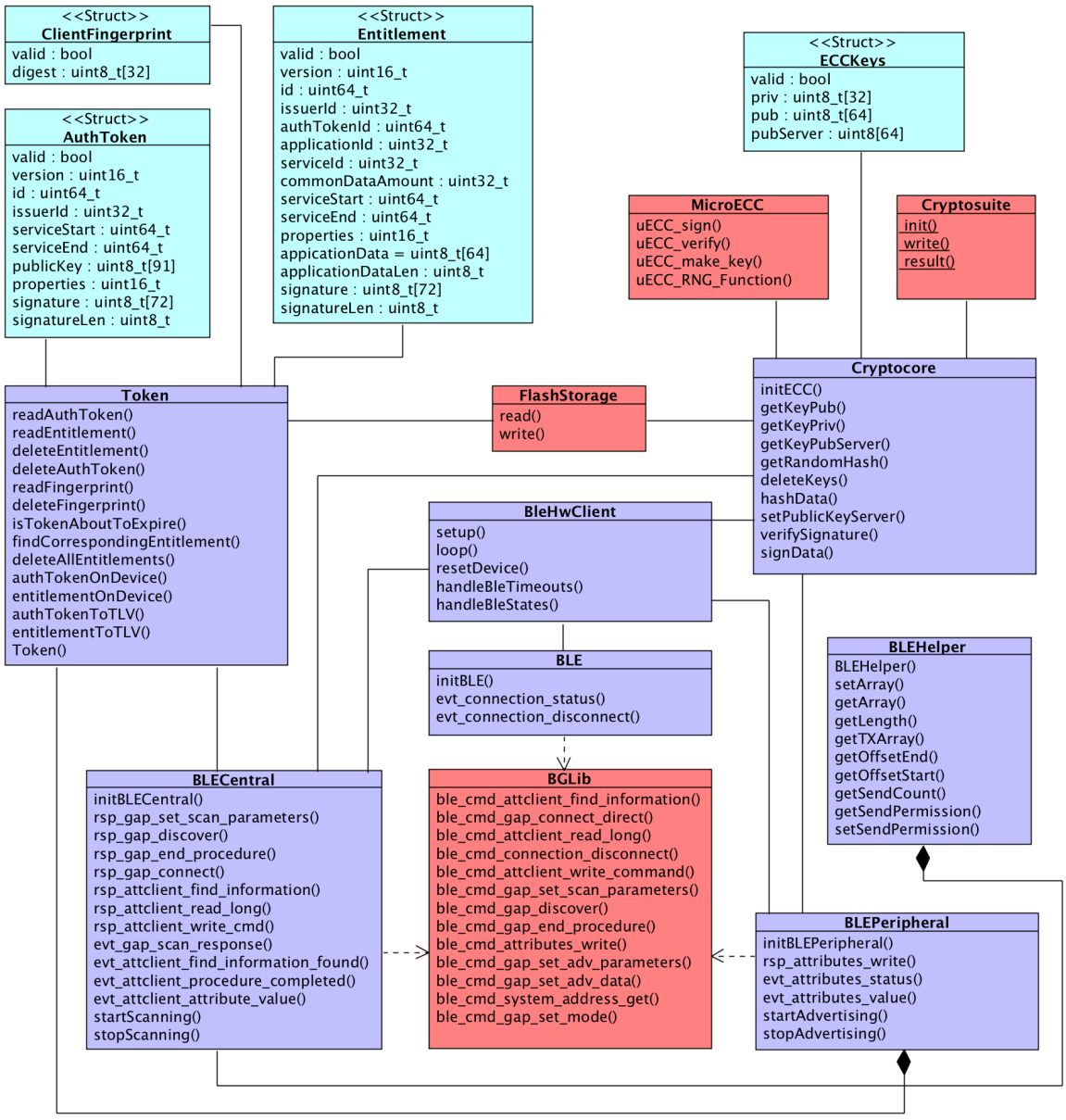

Figure 4.6: Class Diagram Embedded Client

The core of the program lies inside *BleHwClient*. When starting the device the setup function inside *BleHwClient* is called. Among other things, it is responsible for initializing the BLE module, setting up the board's pins, and checking if keys and an Auth-Token are already stored on the device. Depending on whether this data is already available in memory or not, the EClient may start acting as BLE peripheral in search of connectable clients to get an Auth-Token. Otherwise, it starts switching between peripheral and central mode in order to receive Entitlement-Tokens from COYERO Clients or redeem them at COYERO Kiosk devices. Altogether, *BleHwClient* can be considered a state machine. It sets the EClient's *BLE-states* and *Client-states* according to the situation the device is currently in. See section 3.5 for a more detailed description of all possible states of the EClient. Additionally, the device handles BLE timeouts which occur when it is in one *BLE-state* for a predefined time without finding any connectable devices or no data is transferred while it is connected to another device. In this case, the EClient's current BLE role will be stopped and switched. This switching mechanism between peripheral and central role occurs periodically until a connection to another device is established. Moreover, the moment an ongoing connection is terminated the switching between the BLE roles starts again.

Regarding Bluetooth functionality the *BGLib* library provides all BLE relevant methods the BLE112 module offers. To maintain an overview of all BLE methods and callbacks, three classes were introduced. When the EClient acts in peripheral mode, for instance, *BLEPeripheral* is responsible for advertising as well as for data transfer to a nearby BLE-Central device by managing the GATT characteristics and their values of the EClient. *BLECentral* takes over when the BLE central role is active. It involves scanning and data exchange between the communication participants. Since the effective user payload size is 23 bytes, the *BLEHelper* class was introduced which splits longer packets into 20 bytes long data chunks and sends them one by one. Last but not least, the *BLE* class is responsible for handling the connection and delivering BLE status updates.

Data transferred between the EClient and other COYERO devices is encoded in TLV format (explained in subsection 4.6.1). The *Token* class is used by *BLECentral* and *BLEPeripheral* to transform the received TLV-encoded byte stream into *Token*-objects which in further consequence are transformed either into an *AuthToken* or *Entitlement* C-struct. In addition to these tokens also a *ClientFingerprint* struct can be allocated by the *Token* class.

Since data should also be storable to devices without additional storage unit, like EEPROM, the *FlashStorage* library is used to handle writing and reading operations to the internal flash memory. No objects but only structs and primitive data types can be stored via this library. Therefore, *AuthToken*, *Entitlement*, *ClientFingerprint* and *ECCKeys* data is stored inside a struct before writing it to the flash. For further details on these C-structs refer to section 4.5.

Finally, *Cryptocore* makes use of the *Cryptosuite* and *MicroECC* libraries. *Cryptosuite* is responsible for hashing data with SHA-256, while *MicroECC* creates ECC keypairs and provides ECDSA functionality for creating signatures or verifying them. The elliptic curve used within COYERO is secp256r1.

Android Gateway Application

In the following, the software structure of the gateway code which acts in between EClient and COYERO Server is discussed. The code was embedded into the COYERO Android Client application which utilizes BLE central role when communicating to the EClient or to COYERO Kiosk devices. Figure 4.7 shows the corresponding class diagram of the gateway application. To provide a better overview only gateway relevant methods are shown, while function parameters and return values are omitted.

The *BLEInterfaceNew* class controls and manages BLE activities of the client application. It provides methods for scanning for other BLE devices, connecting to them, or aborting connections. It additionally acts as watchdog which aborts ongoing connections if no data is transferred in a certain time.

BLEHwModule contains gateway specific code. This class is invoked if an EClient was found during the scanning phase. In addition to the code responsible for connecting to the EClient, it also stores information about the embedded client's Auth-Token. This, in further consequence is needed to retrieve corresponding Entitlement-Tokens.

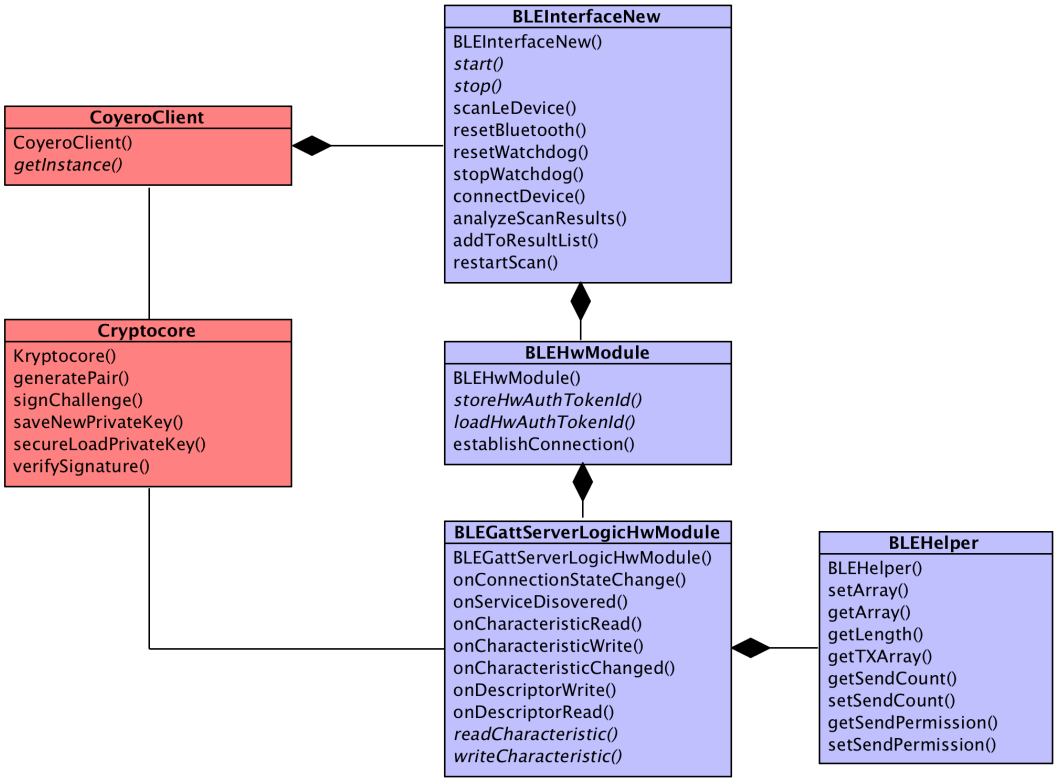

Figure 4.7: Class Diagram Android Gateway

The moment the connection establishment to an EClient was successful the *BLE-GattServerLogicHwModule* comes into play. It basically consists of the Android Bluetooth-Gatt-Callbacks making it possible to read from characteristics and descriptors on peripheral devices or write to them.

Just like the EClient when acting as central device, also the gateway uses a *BLEHelper* class to split longer packets into 20 Byte long data chunks before sending them one by one.

Aside from the mentioned BLE classes there is also the *CoyeroClient* and *Cryptocore*. While *CoyeroClient* is responsible for initializing and administering functionality of the COYERO Client library, *Cryptocore* offers methods for generating ECC key pairs as well as for signing data and verifying signatures.

4.5 Data Handling

Certain data sets need to be stored in the non-volatile flash-memory of the EClient as C-structs, including:

- **ECCKeys:**
  Holds the raw bytes for private and public ECC keys. Since secp256r1 is used, the private key is 32 bytes and the public key 64 bytes long. Additionally, the public server key is stored and used for verifying Auth-Tokens of COYERO Clients. All keys are stored in their raw form.
- **ClientFingerprint:**
  Consists of the SHA256 hashed Auth-Token of the COYERO Client which retrieved the EClient's Auth-Token. It serves as a reference for the EClient to determine from which device it received its Auth-Token. This ensures that the EClient does not accept any Entitlement-Tokens from devices whose Auth-Token digest differs from the stored one.
- **Auth-Token and Entitlement-Tokens:**
  Data structures for identifying entities and entitlements related to an entity. See subsection 2.1.2 for further explanation of these tokens.

4.6 Data Encoding

This chapter is about encoding formats that are applied to the data structures transferred between EClient and the gateway application. See subsection 4.6.1 to get more information about Type-Length-Value encoded data as well as subsection 4.6.2 to find out more about ASN.1-notation and DER-encoding.

4.6.1 Type-Length-Value (TLV)

All data sent over BLE is encoded in TLV-format. It allows the receiver to decode the information without requiring any pre-knowledge of the size or semantic meaning of the data. Each TLV-encoded data-unit conforms to the following format:

- **Type:** Unique byte-code of 1 byte length indicating the kind of data following next.
- **Length:** Holds the size of the value field. Since also the length has a size of 1 byte, the maximum possible length of a data-fragment is 255 bytes.
- **Value:** Actual data bytes.

Thanks to this simple syntax data can be interpreted easily with dedicated parsing functions. Additionally, TLV elements can be placed in any order inside the message body since they can be identified though their *Type/Tag* byte. Another advantage compared to static encoding mechanisms is that data of variable length can be transmitted because the actual length information is always part of the encoded package. However, it should be considered that the *Tag* and *Length* bytes cause an additional overhead which could lead to longer data transmissions, especially when dealing with bigger data sets.

4.6.2 ASN.1 DER-Encoding

Abstract Syntax Notation One (ASN.1) is a notation for structured data which describes rules for encoding and decoding data. ASN.1 specifies several TLV-based encoding standards, such as Basic Encoding Rules (BER) or Distinguished Encoding Rules (DER), which are sets of rules for transforming data structures described in ASN.1 into a sequence of bytes and back [ITU15].

Many widely used cryptographic software libraries make use of DER-encoding. *OpenSSL*, for instance, uses it for any binary output, e.g. cryptographic keys, certificates and signatures. So does the *Java SE Security* library. However, other libraries like *micro-ecc* require cryptographic keys to be in raw-byte-form. Hence, conversion mechanisms between DER-encoded data and raw bytes, or vice versa, are required.

In the following a short example will be given which explains how to extract the raw elliptic curve points from a DER encoded signature. Additionally, another example explains how an EC public key is represented when being DER encoded.

ECDSA Signature

This is an ASN.1 example describing the structure which an ECDSA signature exhibits:

```
ECDSASignature ::= SEQUENCE {
    r   INTEGER,
    s   INTEGER
}
```

When encoded into the DER format it is transformed to the following byte sequence:

I1 L1 I2 L2 R I3 L3 S
I1: Tag of 'SEQUENCE' (0x30)
  L1: Length of remaining bytes
  I2: Tag of 'INTEGER' (0x02)
    L2: Amount of bytes for r
    R: r converted into a big-endian encoded byte array
  I3: Tag of 'INTEGER' (0x02)
    L3: Amount of bytes for s
    S: s converted into a big-endian encoded byte array

The values r and s of an ECC secp256r1-signature, for instance, have a length of 32 or 33 bytes, respectively. If the leading bit is set, an additional zero byte has to be prepended. Therefore, the total length of a DER-encoded secp256r1-ECDSA signature varies between 70 to 72 bytes.

Elliptic Curve Public Key

The public key of a secp256r1 curve is defined in a 65 byte format. The first byte of value 0x04 indicates an uncompressed point [TIB+09]. This byte is followed by 32 bytes representing the X coordinate and another 32 bytes representing the Y coordinate of the EC point [Cer09].

Libraries such as *Java SE Security* apply the ASN.1 DER based encoding on the key-data by wrapping it into a structure that describes the used elliptic curve through the following symbolic identifiers.

There is a SEQUENCE whose contents are a nested sub-SEQUENCE and a BIT STRING. The nested sub-SEQUENCE in turn contains two OBJECT IDENTIFIER values. The first one points out that the key is an elliptic curve public key and the second one designates which kind of ECC curve it is (e.g. NIST P-256). In contrast, the BIT STRING holds the encoded curve point representing the public key itself. To sum it up: the encoded key has a total length of 91 bytes. The first 26 bytes are an identifier for the involved curve, while the remaining 65 bytes contain the encoding of X and Y.

With this in mind the following example demonstrates what a DER-encoded ECC secp256r1 public key (generated, for example, by the *Java SE Security* library) looks like:

I1 L1 I2 L2 I3 L3 O3 I4 L4 O4 I5 L5 0x00 0x04 X Y
I1: Tag of 'SEQUENCE' (0x30)
- **L1:** Length of remaining bytes (89 Bytes)
- **I2:** Tag of 'SEQUENCE' (0x30)
  - **L2:** Length of remaining bytes (19 Bytes)
  - **I3:** Tag of 'OBJECT IDENTIFIER' (0x06)
    - **L3:** Length byte of object identifier O1
    - **O3:** Object identifier bytes
  - **I4:** Tag of 'OBJECT IDENTIFIER' (0x06)
    - **L4:** Length byte of object identifier O2
    - **O4:** Object identifier bytes
- **I5:** Tag of 'BIT STRING' (0x03)
  - **L5:** Length byte of bit string
  - **0x00:** Leading zero byte, because the specified length requires it
  - **0x04:** First byte of public key, indicating an uncompressed point
  - **X:** X coordinate of EC point (32 bytes long when using secp256r1 curve)
  - **Y:** Y coordinate of EC point (32 bytes long when using secp256r1 curve)

The *micro-ecc* library, for instance, needs the public key to be in its 64 bytes raw format. In the example mentioned above this could be achieved by simply extracting the coordinates X and Y.

4.7 Program Flow

This section is about several explanations regarding the program flow of the EClient in conjunction with other COYERO entities, including COYERO Client, Kiosk, and Server. After describing initial setups of the EClient's program in subsection 4.7.1, subsection 4.7.2 will describe the program flow when the Client-state is set to *Authentication*, while subsection 4.7.3 will talk about the program flow when the Client-state is set to *Entitlement*. For a further explanation of available Client-states and BLE-states see section 3.5.

4.7.1 Sequence Setup

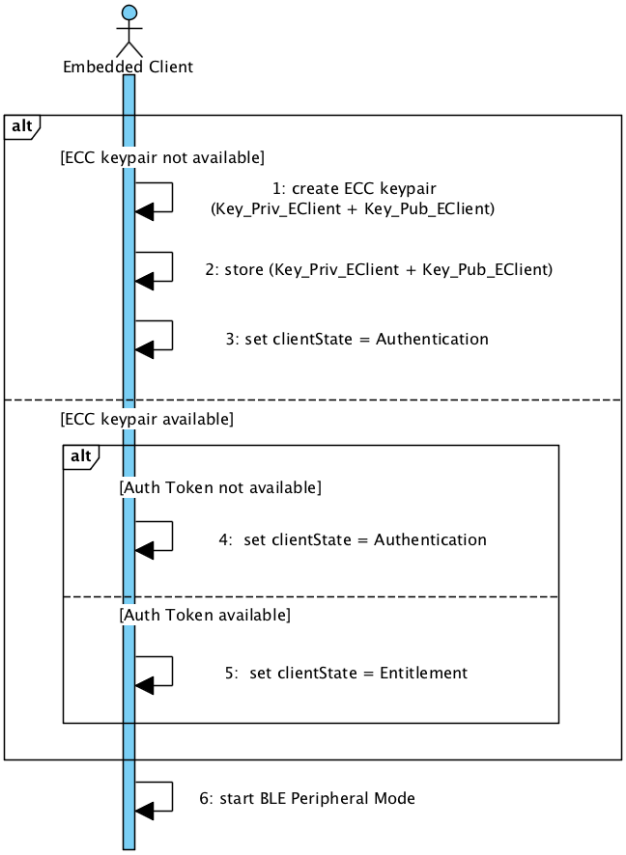

Figure 4.8: Sequence: Setup

Figure 4.8 shows the behavior of the EClient at program start. Whenever it is started it will operate in peripheral mode after doing some checks. First of all, it has to be verified if a valid ECC keypair already exists on the device. If not, it is generated and stored to memory. Since a new keypair requires a new Auth-Token, the Client-state will be set to *Authentication*. If, however, valid ECC keys are already available, there will be a second check to see if there is also an Auth-Token on the device. If so, no authentication step is needed and the Client-state is directly set to *Entitlement*.

After determining the right Client-state the EClient starts to advertise its presence enabling COYERO Client devices to establish a connection to it. The advertising data itself is set depending on the current Client-state and serves as information for a COYERO Client device if either an Auth-Token or an Entitlement-Token has to be sent.

4.7.2 Sequence Authentication State

Figure 4.9 gives an overview of the program flow if the device's Client-state is set to *Authentication*. In this case the goal of the EClient is to retrieve a server signed Auth-Token needed for further authentication. The EClient will advertise its presence with a specific advertising packet telling nearby COYERO Client devices that it is in need of an Auth-Token. As soon as a COYERO Client acting as BLE central device is in range, it will try to connect to the EClient. If a connection between these two devices is established successfully, the EClient will send a data packet containing its public ECC key and its MAC address to the COYERO Client.
In order to minimize the risk of other devices being connected to the COYERO Client and to prevent them from getting a server signed Auth-Token instead of the EClient, a specific dialog window will pop up on the Android app asking the user for permission to proceed. As an additional hint the EClient's blue LED will be turned on, implying that the device is connected and ready to receive an Auth-Token.

Subsequently, the COYERO Client will forward the data together with its own Auth-Token to the server. Its Auth-Token is used by the server to derive an additional Auth-Token for the EClient assigned to the same user. Since a token, among other things, consists of the public ECC key of the owner, the public key of the EClient is integrated into it. Its MAC address, by contrast, is only used for identifying it among other clients. Finally, the new Auth-Token is sent back to the COYERO Client together with the public server key required for further authentication purposes on EClient side.

The next step involves the client generating a unique fingerprint by hashing its Auth-Token with SHA256. The purpose of the gathered digest is to give the EClient the possibility to determine if the COYERO Client it is communicating with was also responsible for delivering its Auth-Token. If the check fails, no entitlements from this client will be accepted. There are two advantages of using the hashed Auth-Token as fingerprint:

- Each Auth-Token is bound to one specific device. Therefore, it is possible to uniquely identify a COYERO device by its Auth-Token.
- Since the Auth-Token is sent anyway before transmitting entitlements, the embedded client just has to hash the received Auth-Token and compare the result with the previous received digest in order to determine whether it is communicating with the right device. No further information has to be sent by the COYERO Client. This enables a faster throughput of the COYERO protocol.

Last but not least, the digest is sent to the EClient together with the public server key and the newly generated Auth-Token. If the delivered Auth-Token can be verified with the public key of the server, the whole dataset will be stored on the device.

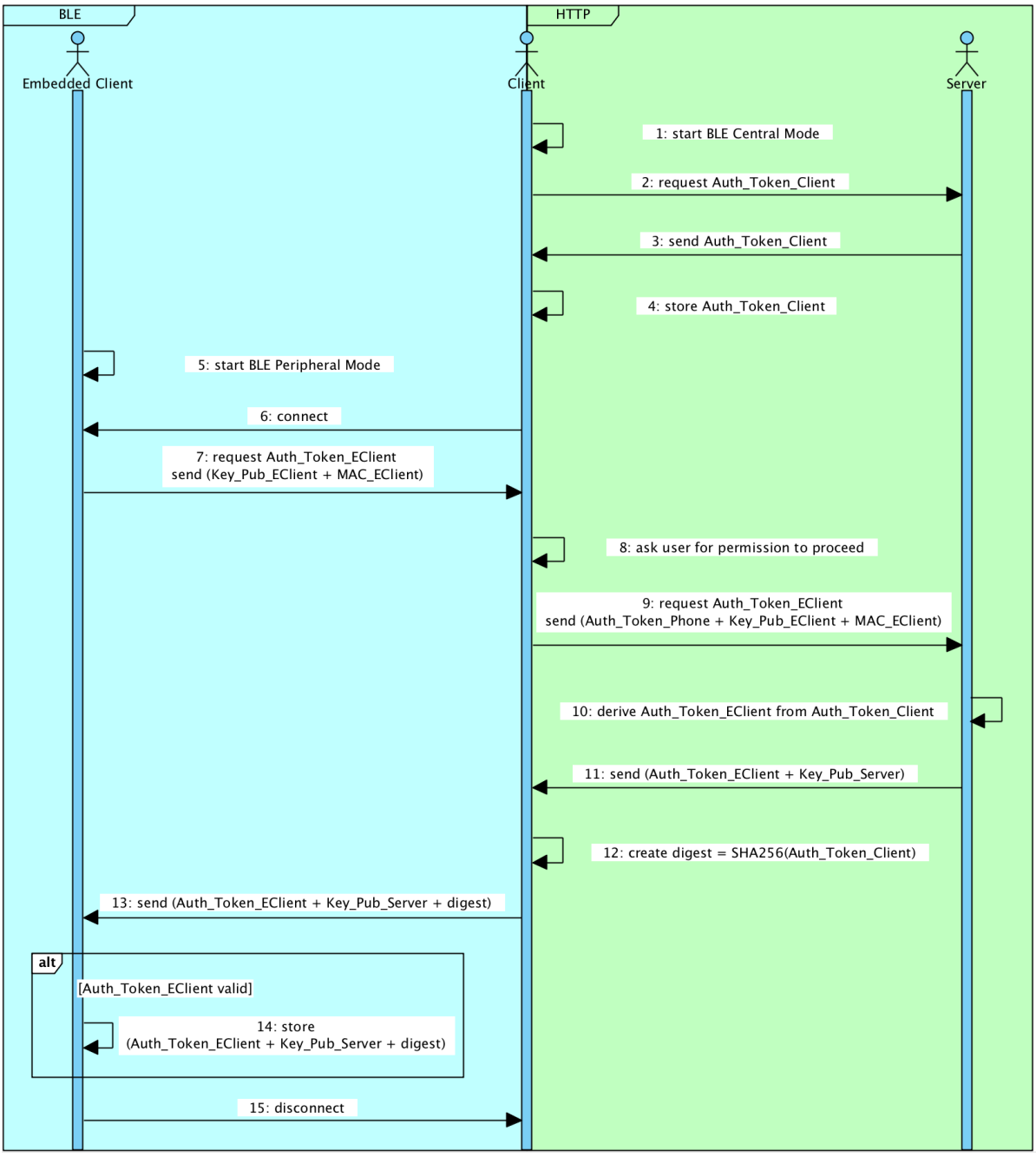

Figure 4.9: Sequence: Retrieval of Auth-Token

4.7.3 Sequence Entitlement State

In the scenario described by figure 4.10 the EClient acts as peripheral device with its Client-state set to *Entitlement*. By doing so, the advertising packet will change compared to when the EClient advertises in *Authentication* state. Hence, COYERO Clients know that the EClient is ready to receive entitlements. As soon as a COYERO Client connects to the EClient, the COYERO Client's Auth-Token and the current time is requested. The timestamp is required to check if the validation period of the EClient's tokens is still okay. Since embedded devices like the Feather M0 do not have internet access or a built-in real-time clock combined with a consistent power supply, the current date and time has to be gathered in this way. Consequently, the following four checks have to be undergone and passed before requesting entitlements:

- The Auth-Token of the COYERO Client has to be verifiable with the public server key received during the authentication process. If not, the connection is aborted.
- The already stored fingerprint must match the digest of the hashed Auth-Token of the COYERO Client. If so, the EClient knows that it is the same device which also sent its Auth-Token, otherwise the BLE connection is dropped.
- It has to be checked if the EClient's Auth-Token has already expired. This is done by comparing the received timestamp with the value inside the *ServiceEnd* field of the Auth-Token. If the token expired the EClient will disconnect from the COYERO Client and switch to *Authentication* state in order to renew the token.
- Last but not least, a challenge response protocol is applied to verify if the received Auth-Token of the COYERO Client really belongs to the device currently speaking to the EClient. First, the EClient generates a random number and challenges the COYERO Client to sign it with its private key. Subsequently, it is signed by the COYERO Client and returned to the EClient. Furthermore, the received signature has to be verified against the generated random number by using the public key of the COYERO Client which is embedded into its Auth-Token. If this check is successful, it was proven that the device the EClient is speaking to is the right owner of the Auth-Token.

If all checks mentioned above are passed, the EClient finally proceeds to the actual request of Entitlement-Tokens (figure 4.11). The request will be sent to the COYERO Client which forwards it to the server. From all entitlements the COYERO Client possesses, the server derives corresponding versions for the EClient bound to the EClient's Auth-Token. They will be sent back to the phone where they are filtered.

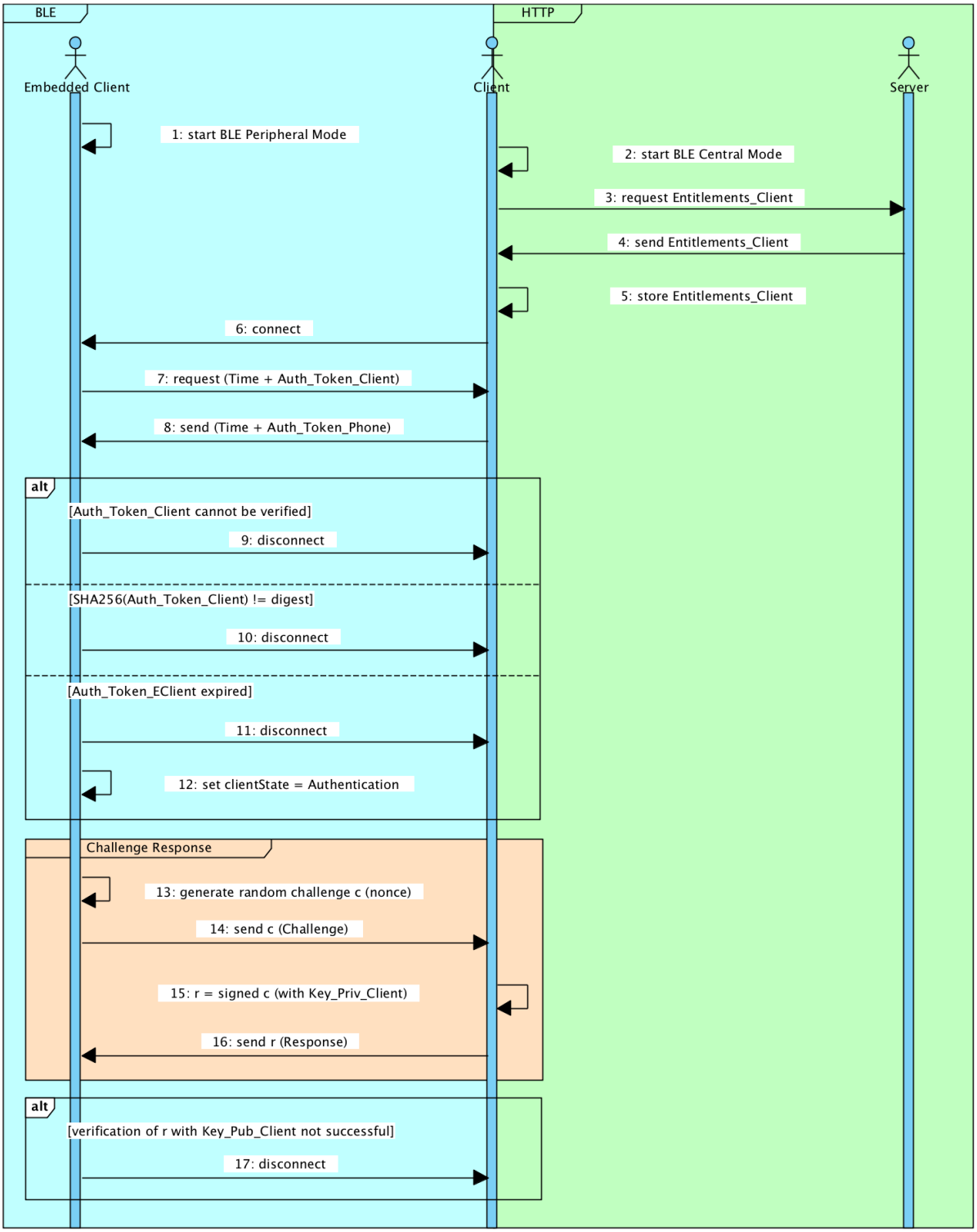

Figure 4.10: Sequence: Retrieval of Entitlement-Token (Part 1)

Only those entitlements which are “Non-Consumable” according to their *Properties* field are forwarded to the EClient. First, already locally stored Entitlement-Tokens are checked for their validity period on the EClient side. If they are expired, they are simply discarded. Second, all newly received entitlements are stored if they are not already on the device.

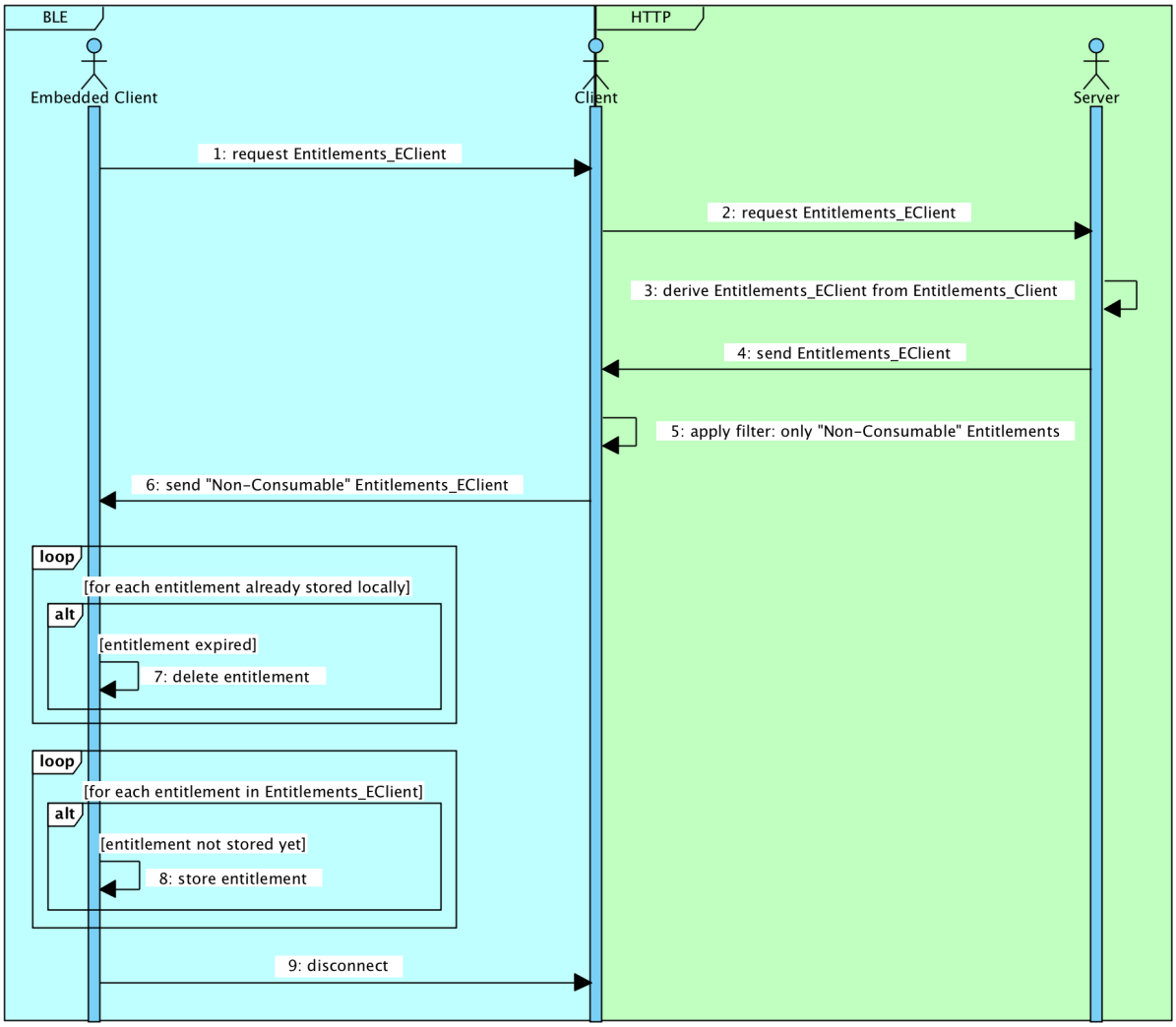

Figure 4.11: Sequence: Retrieval of Entitlement-Token (Part 2)

As soon as entitlements are available on the EClient and its Client-state is set to *Entitlement* it will continuously switch between peripheral and central role at cycle times of one second. This additionally enables communication with COYERO Kiosk devices which act as peripherals only. Figure 4.12 explains the program sequence for the scenario when the EClient connects to a COYERO Kiosk device specialized in the parking sector, to redeem its Entitlement-Tokens.

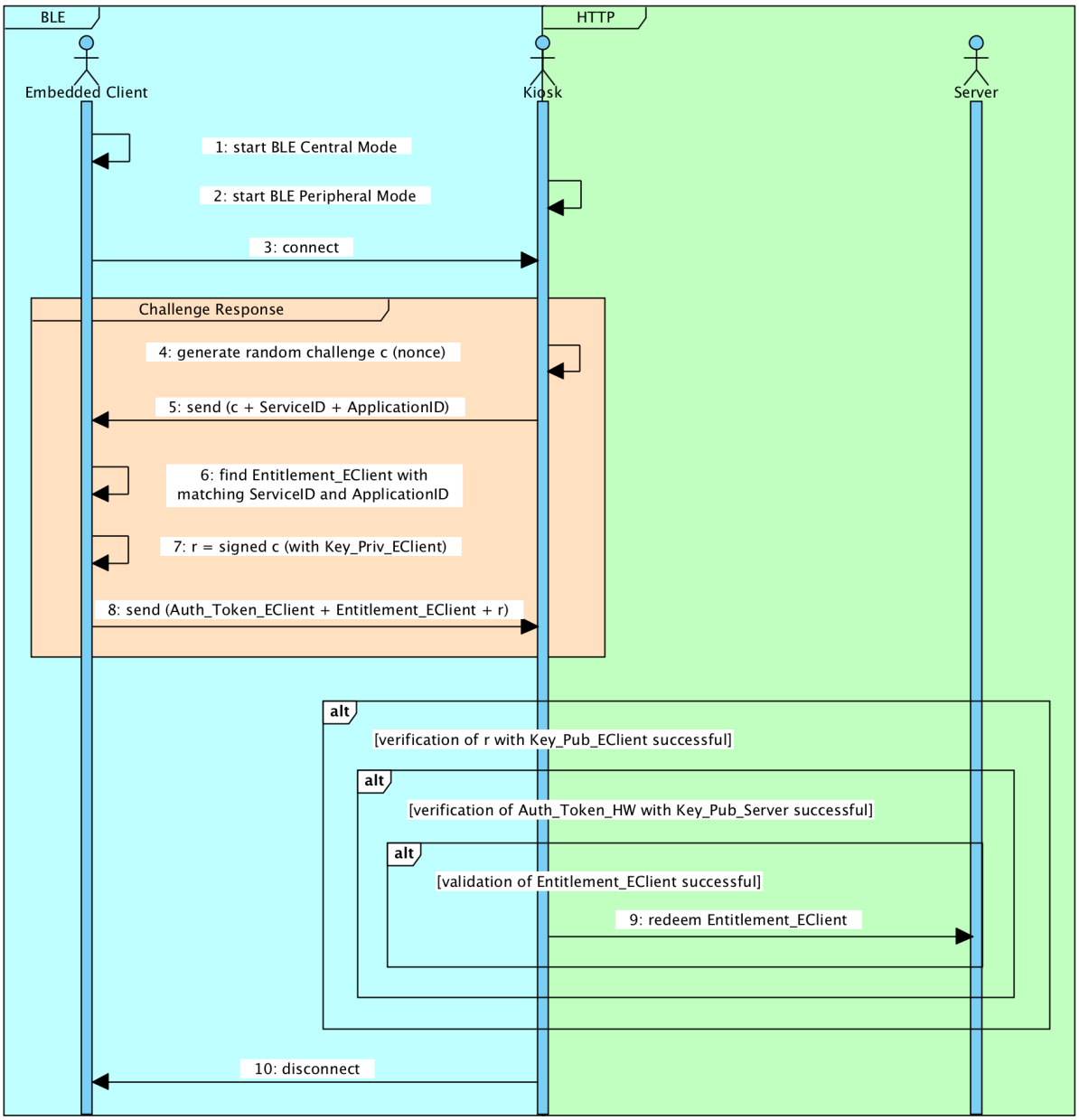

Figure 4.12: Sequence: Redemption of Entitlement-Token

Right after connecting to the kiosk, a random nonce will be generated and sent together with a specific service-ID and application-ID to the EClient. These IDs are used as filter criteria to specify which Entitlement-Tokens are redeemable by the kiosk device speaking to the EClient. Subsequently, on EClient side it will be checked if any of the locally stored entitlements match the given filter criteria. If so, the received challenge will be signed and sent back to the kiosk device together with the Entitlement-Token found and the EClient's Auth-Token. Afterwards, the response can be verified by the kiosk with the public key of the EClient's Auth-Token against the original challenge. If, furthermore, the verification process of the received Auth-Token turns out to be positive, also the Entitlement-Token will be validated by the kiosk. If it is valid, it will be redeemed and the user will finally obtain the actual product or service.

4.8 Communication Between Client and Server

The Coyero API library provides several methods for interaction between clients, kiosks, and the COYERO backend. It is organized around REST and is designed to have resource-oriented URLs. It uses HTTP response codes to indicate API errors. Authentication of the COYERO devices is necessary whenever they intend to communicate with the server. It is ensured by providing their secret API key for basic HTML authentication in the request. Additionally, all API server calls have to be done over HTTPS, otherwise they will fail [CIS15, COY16].

In the course of this thesis the API library had to be extended for new HTTPS calls customized for the EClient. Added features include:

- **getAdditionalAuthTokenForToken:** Creates an additional Auth-Token for a user who already possesses an Auth-Token. The new Auth-Token is linked with a new device entity and sent back.
- **getAllValidEntitlementTokens:** If an Auth-Token ID is submitted, additional entitlements are derived from the entitlements bound to the Auth-Token ID. Thereinafter, the list of additional entitlements is returned.

Chapter 5

Evaluation

Chapter 5 expands the initial assessment described in section 2.6 of the thesis's prototype. It evaluates the results of this thesis and compares them to similar projects discussed in chapter 2. While the results of several measurements are illustrated in section 5.1, the comparison-based part of this chapter can be found in section 5.2.

5.1 Measurements

Source Line of Code (SLOC) of Prototype Implementation

The prototype implementation consists of about 6500 SLOC split across the embedded client implementation written in C/C++ and Gateway-, GUI-, and server Code written in Java.

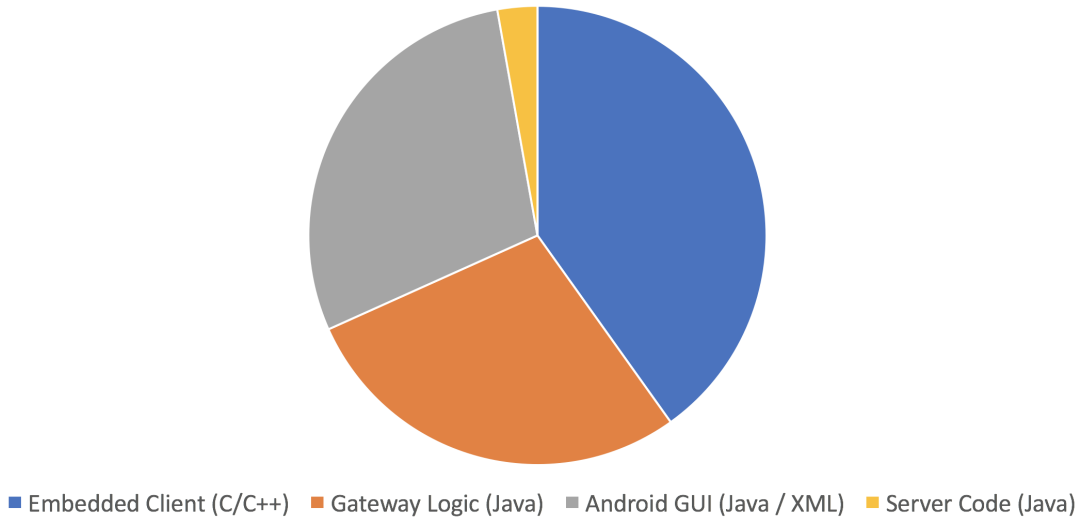

Figure 5.1: SLOC of Prototype Implementation

The distribution of the SLOC can be inferred from chart 5.1. These metrics were measured with the program SourceMonitor (Version 3.5) developed by Campwood Software.[1]

[1] http://www.campwoodsw.com/sourcemonitor.html (Accessed: 2017-04-07)

Timing

The following three tables provide timing-relevant information. Table 5.1 shows how much time the EClient takes to retrieve a server signed Auth-Token via the gateway and to verify and store it. Table 5.2 shows how much time passes to retrieve and verify an Entitlement-Token, while table 5.3 illustrates the timing behavior of the redemption process at corresponding COYERO Kiosk devices.

Action	**Time [ms]**
Retrieve Auth-Token via Gateway	1610
Verify Auth-Token	401
Store Auth-Token	70
Total	**2071**

Table 5.1: Time Measurement – Retrieve and Store Auth-Token

Action	**Time [ms]**
Get Timestamp + Auth-Token Client	410
Do Checks on Auth-Token	488
Challenge Response	831
Retrieve Entitlement	342
Store Entitlement	65
Total	**2136**

Table 5.2: Time Measurement – Retrieve and Store Entitlement-Token

Action	**Time [ms]**
Read Data from Kiosk	620
Prepare Data to Redeem	604
Redeem Entitlement	452
Total	**1676**

Table 5.3: Time Measurement – Redeem Entitlement-Token

In summary, the retrieval of Auth- and Entitlement-Token takes approximately two seconds each, while the entitlement redemption itself is processed in less than two seconds. These values represent short waiting times for the user underlining the usability aspect of the developed prototype.

Power Consumption

Depending on the active BLE-state the power requirements change. Table 5.4 illustrates how each BLE-State affects the overall power consumption if the Feather M0 board of the EClient is powered via a micro USB cable.[2] According to Adafruit's datasheet of the Feather M0 it will automatically regulate the 5V USB down to 3.3V [Ada16a]. To get a better insight into all BLE-states implemented and their remit please refer to section 3.5.

BLE-State	Power consumption [mA]
Standby	16.9 - 18.5
Advertising	24.7 - 25.8
Scanning	43.0 - 44.7
Connected Peripheral	26.0 - 26.5
Connected Central	27.5 - 28.1

Table 5.4: Measured Power Consumption

These measurements demonstrate clearly that a device acting as BLE central, especially when scanning, consumes much more power than when acting as BLE peripheral device. This, in further consequence, is one of the reasons why power constrained devices should utilize the peripheral role only. The higher power consumption presents no problem for the EClient since, according to the use case description of section 3.1, it could be powered directly by a car.

If, however, the EClient should be operated by battery, the integrated Feather M0 board offers the possibility to connect a lithium polymer or lithium ion battery to the JST jack. This will let the board run on a rechargeable battery which is recharged at 100mA when the device is powered by USB [Ada16a].

5.2 Comparison with Other Systems

The goal of this thesis was to design and implement an embedded client making use of several BLE roles and a smartphone-based gateway application in order to exchange data between the embedded client and the cloud. The prototype designed was embedded successfully into the COYERO access platform. The application area of the EClient and its gateway is the parking domain. Hence, the EClient can be used as a smart device to give its users access to parking lots or garages. Thanks to cryptographic concepts like PKI and ECDSA they do not need to worry about security.

The following pages compare the prototype implementation discussed in this thesis with related projects listed in sections 2.3 and 2.4 which rely on similar technical concepts. The comparison includes communication aspects, security relevant features, as well as information on compatibility and general setup. Subsequently, products that are being operated in the parking domain and enable their customers access without smartphone will be named for comparison purposes. They were already described in section 2.5.

[2] Measured with Voltcraft VC220 Multimeter

5.2.1 Communication

The project discussed in paper [FMA+16] makes use of BLE enabled smartphones to detect slope errors in advance. The devices are able to act as master and slave by supporting BLE central and peripheral roles in order to pass data through a multi-hop network. This BLE multi-role behavior is also utilized by the EClient, making it possible to talk to dedicated COYERO Client devices (centrals) and COYERO Kiosk devices (peripherals). In case of the EClient, besides the central role also the peripheral role enables data reception and transmission when connected since the allocated BLE characteristics are readable and writable. In both projects mentioned above a connection between dedicated devices is established before exchanging data.

The project illustrated by paper [CLH16], on the other hand, sends data without an actual connection. It utilizes BLE beacons to forward user data collected from BLE wearables to a central point. The beacons use the non connecting BLE roles broadcaster and receiver to exchange data. Compared to the connection enforced roles central and peripheral, this approach offers the advantage that data can be transferred faster. The reason for this is that relevant data is already inside the advertising packet. Hence, scanning devices may directly read from it without having to undergo a connection establishment procedure. However, the disadvantage of this approach is that only a limited set of data can be passed on and the data itself is being sent without knowing if it was actually received by other devices or not, without further ado. This, in further consequence, implies that a connection between devices is always required if more data has to be sent or if the implemented communication protocol between the BLE devices relies on synchronous messaging.

According to the BLE specification [Blu14b] the Link Layer BLE dual mode is only supported by BLE chips running a BLE 4.1 stack or newer, meaning that those devices are able to act as master and as slave simultaneously. Managing multiple BLE roles and connections at the same time is a complex and power consuming task. Therefore, many BLE devices today still do not support this concurrent multi-role behavior. Nevertheless, it is possible for devices not supporting link layer dual mode to act as master as well as slave by actively switching between BLE roles. The project discussed in [CLH16] utilizes the CC2541 SoC's BLE 4.0 stack to scan or broadcast in alternate sequence for five seconds. The Bluegiga BLE112 chip used by the EClient also uses a BLE 4.0 stack. Therefore, it is necessary to actively switch between different BLE modes in order to be able to talk to different BLE devices regardless of their active BLE role.

The gateway developed in form of an Android smartphone application was embedded into the COYERO Client app. Since it utilizes BLE central role, it is able to talk to the EClient functioning as BLE peripheral. The gateway accepts requests from the EClient, transmits them to the server, and sends back the server response. Depending on the request itself, data may be processed by the gateway before sending it back to the EClient. In any case, in order to meet all requirements of the COYERO protocol and to be able to provide the EClient with required server, data two-way communication between EClient and Gateway is necessary. There are other approaches that investigated smartphone based gateway solutions. The project described in [TSPA16] is one of them. Also in this case the smartphone acts as BLE central device and retrieves data from other BLE devices acting as BLE peripheral. Since the main task of the gateway is to retrieve and store sensory data

from BLE peripherals into one central online database, one-way communication from BLE device to gateway is necessary without further interaction between smartphone gateway and BLE device.

Smartphone gateways based on the fabryq platform [MERH15] support BLE read and write operations enabling one- or two-way communication between gateway and client depending on the application. Last but not least, BluFi's gateway is an example of a smartphone independent solution. Compared to all other gateway approaches discussed in this thesis it has the advantage of offering peripheral as well as central role in order to connect to BLE devices of all kinds [Blu16b].

5.2.2 Security Between BLE Devices and Gateway

The amount of security an application needs depends highly on its requirements and application area. Sometimes security goals like confidentiality are not required. The system discussed in [CLH16], for example, gathers anonymous positioning data of customers inside a theme park, while the project explained in [FMA+16] is utilized for early slope detection in mountains. In both cases, even if an attacker eavesdrops data, the benefit he gains remains limited.

The EClient/gateway project also does not provide data encryption mechanisms. All data including tokens are sent publicly. However, security goals like integrity, authentication, and authorization are provided by relying on a combination of cryptographic concepts like PKI and ECDSA. A COYERO Client device may only redeem purchased entitlements if it can authenticate itself in the context of several checks. Among other things, it needs to possess the correct private key corresponding to the public key integrated into all its tokens to pass a challenge-response authentication process with kiosk devices. Therefore, even if an attacker gains access to a token, he cannot do anything with it as long as he does not know the user's private key which never leaves the EClient. An additional layer of encryption would be redundant and just increase the computational effort and consequently the overall energy consumption of the device.

Other projects like the ones specified in papers [MERH15, TSPA16] are specialized in retrieving sensory data from BLE peripherals of different kinds. BLE sensors could provide values like temperature but also more privacy relevant information like vital data about their users. Pulse rate or blood oxygen saturation are just some examples. However, those papers do not mention how the communication channel between BLE devices and the gateway could be secured. To enable an encrypted connection to BLE devices, the BluFi gateway, for example, makes use of BLE AES 128 link layer encryption. Additionally, all communications from and to beacons are encrypted via RSA [Blu16b]. Just like BluFi also Onyx Beacon provides AES-128 link layer encryption. Additionally, MAC's are used to ensure message authentication [Onya]. Regarding the security features of the Apple Watch, an OOB pairing method is used to exchange the BLE link layer key for the AES-128 encryption. Subsequently, also public keys for further cryptographic operations are exchanged. In addition to the link layer encryption, messages transferred between the Apple Watch and an iOS device are also signed and encrypted on the application layer. Finally, Apple Homekit's security relies, among other things, on ECC Ed25519 for authentication between iOS devices as well as between iOS devices and accessories and Secure Remote Password (3072-bit) protocol for exchanging keys [MA16, App17a, App17b].

5.2.3 Compatibility

The prototype of the developed EClient was embedded into the COYERO access platform, while the gateway part was implemented as part of the COYERO Client library. In its current implementation state the EClient is compatible with other devices utilizing a COYERO library. The gateway itself was designed and customized in a way to fulfill all needs and requirements of the EClient and the security conscious COYERO protocol.

A more generic smartphone based gateway approach is shown in paper [TSPA16] where sensory data of different BLE devices can be retrieved by specifying the devices and their BLE properties via an XML schema.

The project Fabryq goes one step further by providing a framework for developers and end users to interact with BLE sensors of different kinds over the internet. Developers only have to concentrate on writing applications in form of a web client, while users can choose among those programs and use the one that best fits their desired use case and hardware [MERH15].

5.2.4 Setup

Table 5.5 shows the projects discussed in chapter 2 that rely on BLE to enable a client device to transfer data over the internet via a gateway.

Project/Product	**Client Device**	**Gateway Device**
Downhill [FMA+16]	Android phone	Android phone
IoT Cloud Data Management[TSPA16]	BLE wearable devices	Android phone
BLE beacon-based Network[CLH16]	CC2561 (SoC)	PC
Fabryq [MERH15]	BLE devices	iOS device
Apple Watch [App17a]	Apple Watch	iOS device
Apple Homekit [MA16]	BLE devices	Apple TV or iPad
Onyx Beacon [Onya]	Beacon device	iOS/Android device
Prototype of this thesis	EClient	Android device

Table 5.5: Client/Gateway Setup of Different BLE Projects and Products

Although the devices are employed in different application areas the technology running in the background is similar. It is notable that in a lot of projects smartphones are used as gateways. First of all, they provide a lot of different sensors and communication channels on their own. Furthermore, they are battery operated devices and hereby also suitable for outdoor applications. Additionally, they offer the possibility to act as interface between user and application. Thanks to their large price range Android phones are widespread.

Among the projects mentioned in table 5.5 the setup of the Apple Watch in combination with the iPhone is the most similar to the prototype developed during this thesis. Just as the Apple Watch incorporates many functions of the iPhone but needs it for synchronization purposes and internet access, the EClient acts as an independent COYERO

Client but needs a COYERO Client smartphone to retrieve data from it (e.g. current time) or to exchange data with the COYERO Server over the internet.

If BLE was combined with IPv6, no additional gateways would be needed to grant BLE devices the possibility to communicate over the internet [NTS+15]. The absence of a gateway in turn would provoke a higher computational effort and expanded memory requirements for embedded devices. First, program logic which otherwise would be executed by the gateway would have to be processed by the embedded devices itself. Second, BLE devices would need enough memory and processing power to support a BLE and an IPv6 stack. In the particular case of the EClient another problem would be associated with the absence of the smartphone gateway: Although the EClient might indicate its state via specific LED combinations (see table 3.2), it would not be able to display any details about available user data. Therefore, the COYERO Client gateway cannot be considered as redundant because, in addition to its gateway functionality, it also provides a user interface where data can be managed. Other projects [TSPA16] [MERH15] utilize smartphones functioning as gateway and user interface in a similar way.

5.2.5 Application Area

The application area of the EClient and its gateway embedded into the COYERO access platform is smart parking. After an initial synchronization process between EClient and COYERO Client acting as gateway, the EClient can operate autonomously for the validity period of its issued entitlements. Consequently, this enables its users to gain access to parking lots, garages or other places where secure access plays an important role without further interactions with their smartphones. This, in further consequence, increases usability a lot.

Regarding the use case of smart parking without smartphones, both Evopark and Telepass are already commercially deployed systems. Customers of those systems are equipped with additional devices. In case of Evopark it is a chip card, while in case of Telepass the user gets a small appliance that has to be attached on the car's windscreen. As soon as cars equipped with those systems approach a garage of a supported provider, the user gains access without smartphone interaction [P. 04, evo].

A distinguishing characteristic which affects the application area and use case is that the EClient uses BLE, while the other systems utilize RFID. Since most smartphones are BLE-enabled devices, direct communication between them and the embedded device is feasible. Not only does a smartphone act as internet-gateway for the EClient, it also serves as user interface for managing and keeping track of the user's entitlements. Speaking of different entitlements, the EClient in combination with the COYERO access platform has the advantage of being a more generic system than the other two. It allows purchasing and redeeming entitlements for a broader range of application areas. Possible scenarios are, for example, getting access to hotels or redeeming food vouchers at drive-through restaurants.

Chapter 6

Conclusion and Future Work

The first part of this chapter summarizes the thesis by giving an overview of the most important parts. The second part elaborates possible future extensions and improvements of the developed prototype and the COYERO access protocol. On one hand, security relevant issues are discussed and suggestions for improvement are given. On the other hand, additional hardware units for increased security are proposed.

6.1 Resume

This thesis is about the design and implementation of an embedded device with BLE 4.0 stack acting in multiple BLE roles and of a corresponding Android gateway application that enables the embedded device to talk to the cloud. The prototype was integrated into the COYERO access platform which grants its users access to local infrastructures and products.

Since embedded devices are power constrained, BLE, known for its power saving features, was chosen as the technology to transfer data wirelessly. The BLE code was implemented on the Arduino based Featherboard M0 which interacts with an attached BLE112 chip over the Serial UART interface to utilize BLE relevant features. This modular design enables the reuse of the code for similarly specced Arduino boards by just taking along the BLE112 unit. Regarding related work a distinction between scientific methodologies and commercial products was made. Systems and devices relying on the same or similar technologies were explained and served as a basis for discussion. The primary application area of the prototype developed in the course of this thesis is smart parking without smartphone. In this sense, also a few other projects which are already deployed in this field of application but use RFID were discussed and compared.

The embedded device significantly facilitates the process of gaining access to parking lots for the user by minimizing the interactions required between user and mobile phone. The user only needs to open the COYERO Client app once for the synchronization of his purchased entitlements. Afterwards, the embedded device can act autonomously during the validity period of the issued entitlement. It authenticates the user against a COYERO Kiosk device in charge of controlling the gate at the entrance of a garage, for instance.

The more connected our world becomes, the more important it is for technology to be secure against faults of different kinds. In order to be protected against fraudulent activities the embedded device and its gateway incorporate security mechanisms like PKI and ECDSA which are supported and demanded by all COYERO entities.

6.2 Future Work

This section includes possible improvements of the COYERO access platform, which in further consequence would also enhance the security level of the developed prototype. The authentication mechanism between EClient and kiosk is carried out on just one side to enable a faster data transmission. This is especially useful in the parking scenario to lower waiting times. However, mutual authentication would provide more security. Subsection 6.2.1 describes an improved and more secure authentication mechanism between a COYERO Client or an EClient and a COYERO Kiosk device. It was not implemented in the current version of the prototype since, according to the specified requirements (see section 3.2), compatibility to other COYERO devices should be maintained. In this sense the COYERO Kiosk should not be modified in any way.

Additionally, the current version of the COYERO protocol is vulnerable against relay attacks. See subsection 6.2.2 for further explanations. Countermeasures against this kind of attack were not implemented because this would have introduced major changes to the COYERO protocol, which would have involved also the modification of other COYERO devices.

6.2.1 Improved Authentication

The challenge-response part of the protocol between COYERO Client and Kiosk was designed in a way to enable fast data transfer with a minimum number of transactions. The main difference of the following approach compared to the implemented version (see figure 4.12) is that the communication participants now perform a mutual authentication based on their Auth-Tokens just by adding one additional transaction.

To put it in other words, also the kiosk device has to prove that it is the rightful owner of an Auth-Token. Therefore, also the EClient challenges the kiosk to sign a random challenge. If the Auth-Token of the kiosk device can be validated against the COYERO Server's public key and the signed response is verifiable with the public key of the kiosk device embedded into its Auth-Token, the protocol is processed as described in subsection 4.7.3. Figure 6.1 illustrates the proposed authentication procedure.

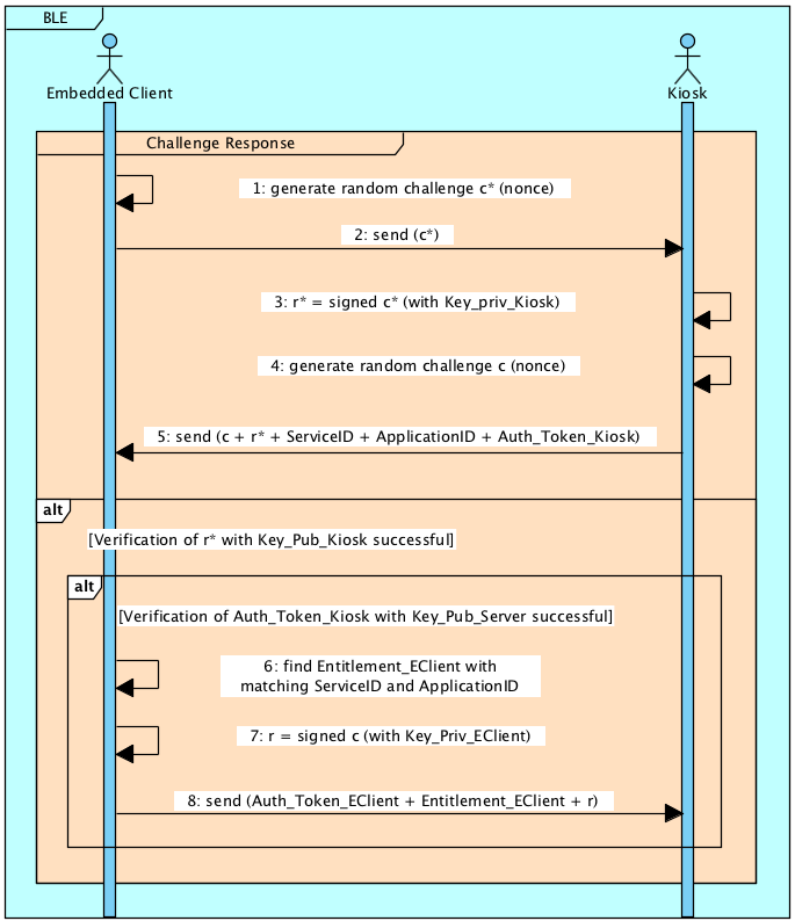

Figure 6.1: Improved Authentication Between EClient and Kiosk

6.2.2 Relay Attack

In IT-security a relay attack is a special form of a man-in-the-middle (MITM) attack. While during a classic man-in-the-middle attack an attacker intercepts and manipulates communications between two parties initiated by one of the parties, in a relay attack communication with both parties is initiated by the attacking device. Subsequently, the attacker is able to relay messages between the communication participants without manipulating or necessarily reading them.

Attack Scenario

A relay attack with the current version of the COYERO protocol would be possible, given the setup shown in figure 6.2.

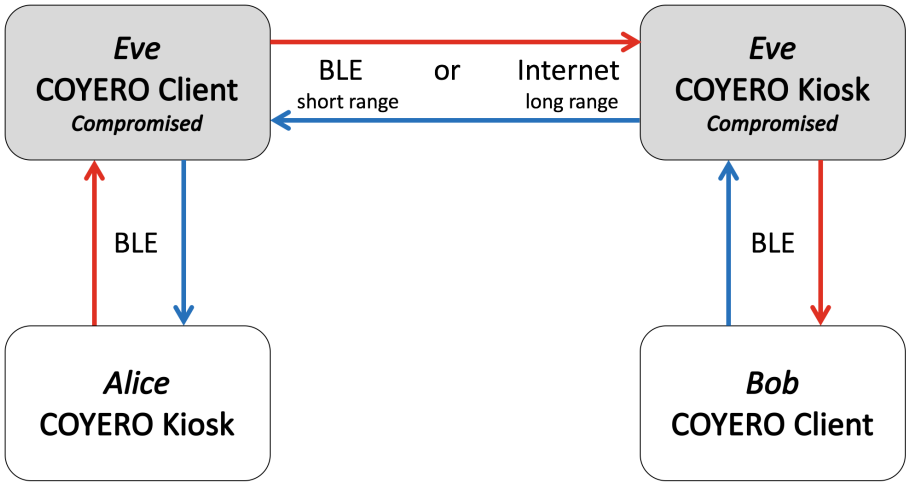

Figure 6.2: Relay Attack Scenario

Consider the following attack scenario. Bob is in possession of a smartphone with an installed COYERO Client app. Recently he acquired a one year permission to park at Alice's gated parking lot. Whenever Bob wants to park his car there, he first has to authenticate himself at Alice's kiosk device which is part of the control unit of the gate.

Eve has no intention of spending money on parking entitlements. Nevertheless, she wants to enter the garage. To do so Eve equips herself with two phones. On the first smartphone a compromised COYERO Client application is installed, while on the second one an altered COYERO Kiosk application is running. After completing all preparations, she finally approaches the parking gate. In the background her client smartphone and Alice's kiosk device start exchanging messages. Under normal circumstances Eve wouldn't be able to get inside the garage without being in possession of the right authentication and entitlement data.

To overcome this problem she impersonates Bob. For this, she forwards all requests coming from Alice's kiosk over the internet to her compromised COYERO Kiosk device which is positioned near to Bobs Client device. Now Eve's kiosk starts communicating to Bobs Client application by delivering all received requests originating from Alice's device. Bob now thinks that he is speaking to Alice. Subsequently, he passes his responses back to Eve's kiosk device which in further consequence sends them back to Alice through Eves COYERO Client. By doing so Alice's kiosk believes that it is talking to Bob and will grant Eve access to the garage. Eventually, the parking gate will open and Eve is free to enter.

Mitigation on Link Layer: BLE Pairing

In order to mitigate the risk of a relay or generally a MITM attack, the BLE link layer pairing could be used. Several key exchange protocols are provided for the pairing process. However, the exact implementation of the pairing protocols varies across different BLE versions. In case of the BLE 4.0 standard a 128-bit Temporary Key (TK) is exchanged. The TK is in turn utilized to generate a 128-bit Short Term Key (STK) for encrypting the connection [Blu10].

According to the core specifications of BLE 4.0 and 4.1, devices may choose between *Secure Simple Pairing* methods such as *Just Works*, *Passkey Entry*, or *Out of Band* Pairing (OOB) depending on their input/output capabilities. The following methods provide confidentiality if passive eavesdropping during the pairing process can be avoided [Blu10, Blu14b]:

- **OOB:**
  The TK is exchanged using a different wireless technology such as NFC, for instance. The security level highly depends on how secure the OOB channel is.
- **Passkey:**
  In this case the TK is a number consisting of six digits that is passed between the devices by the user. The exact way how this number is transferred can vary. One possible solution would be that one device generates a random number and shows it on its display. Subsequently, the user could read the number and enter it manually into the other device.

If the possibility of passive eavesdropping cannot be ruled out the *LE Secure Connections* pairing model supported as of BLE version 4.2 would be a possible remedy. In this case the ECDH key agreement protocol is used in order to mitigate the risk of passive eavesdropping. It provides a mechanism for exchanging a Long Term Key (LTK) over an unsecured channel to encrypt the connection [Blu14b].
Although BLE pairing methods provide concepts as a means of leveraging security, they negatively affect usability because each pairing method involves the interaction of the user, e.g. for manually entering a number in case of *Passkey*, for approaching a certain location in case of *OOB* combined with the short range technology NFC, etc.

Mitigation on Application Layer: Timeouts

In order to minimize the risk of MITM attacks without affecting usability, the COYERO protocol could introduce timeout mechanisms. If a transaction takes longer than an expected pre-defined threshold, the ongoing connection could be terminated. This measure is based on the assumption that a MITM would prolong the actual data transfer. In case of COYERO, if a malicious device intercepts the communication between two communication participants, the overall communication time would rise compared to a direct communication between COYERO Client and Kiosk since data had to be passed through at least one additional device.

6.2.3 Additional Secure Hardware

Only few devices deployed in IoT applications offer hardware-based security features like secure storage elements or cryptographic hardware accelerators. However, several manufacturers offer external hardware elements which can be attached to several development boards to enhance the security level. The outsourcing of confidential data such as cryptographic keys in case of the EClient would make it harder for attackers to gain access to this data.

Manufacturers like *Atmel* or *Maxim Integrated* offer secure hardware which can be attached to other boards. The Atmel ATECC508A [Atm15] and the Maxim Integrated

DS28C36 [Max16], for example, are cryptographic devices with a secure hardware based key storage that can be attached to other devices through the I^2C interface. Since they also offer elliptic curve and SHA-256 functionality, they would be suited to match the cryptographic needs of the EClient discussed in this thesis. Among other things, those chips offer the following features:

- Integration of elliptic curve Diffie-Hellman key agreement (ECDH) and elliptic curve digital signature algorithm (ECDSA)
- P256 elliptic curve support
- SHA-256 hash algorithm with HMAC option
- Secure key storage
- True random number generator (RNG)

If there is no need for an encrypting unit but only for secure storage, the DS3660 of Maxim Integrated could be used. It offers 1kB of secure SRAM storage and features to prevent tampering. It includes a seconds counter, watchdog timer, CPU supervisor, nonvolatile SRAM controller, and an on-chip temperature sensor. An external battery source is automatically used to keep the memory, time, and tamper-detection circuitry active if a power failure occurs [Max].

Bibliography

[AC99] L. Steve A. Carlisle. *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations*, pages 33–35, 24–30. New Riders Publishing, 1999.

[Ada16a] Adafruit Industries. *Adafruit Feather M0 Bluefruit LE*, Sep 2016. `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-m0-bluefruit-le.pdf`. Accessed: 2016-10-12.

[Ada16b] Adafruit Industries. Adafruit_BluefruitLE_nRF51. Dec 2016. `https://github.com/adafruit/Adafruit_BluefruitLE_nRF51`. Accessed: 2016-12-14.

[Anda] Google Android. Android 5.0 Bluetooth Low Energy. `https://developer.android.com/about/versions/android-5.0.html`. Accessed: 2017-03-24.

[Andb] Google Android. Android Volley. `https://developer.android.com/training/volley/index.html`. Accessed: 2017-03-24.

[Andc] Google Android. App Manifest. `https://developer.android.com/guide/topics/manifest/manifest-intro.html`. Accessed: 2017-04-19.

[App17a] Apple Inc. About Bluetooth and Wi-Fi on Apple Watch. Mar 2017. `https://support.apple.com/en-us/HT204562`. Accessed: 2017-04-01.

[App17b] Apple Inc. iOS Security Guide iOS10. pages 23–25, 29, 43–44. Mar 2017. `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`. Accessed: 2017-04-10.

[Ard17] Arduino. Getting started with the Arduino Due. 2017. `https://www.arduino.cc/en/Guide/ArduinoDue#toc4`. Accessed: 2017-03-30.

[Atm15] Atmel Corporation. *ATECC508A Atmel CryptoAuthentication Device SUMMARY DATASHEET*, Oct 2015. `http://www.atmel.com/Images/Atmel-8923S-CryptoAuth-ATECC508A-Datasheet-Summary.pdf`. Accessed: 2016-12-20.

[Blu] Bluvision Inc. *BLUFI FACT SHEET*. `https://bluvision.com/wp-content/uploads/2014/11/Bluvision-BLUFI.pdf`. Accessed: 2016-12-20.

[Blu10] Bluetooth SIG. BLUETOOTH SPECIFICATION Version 4.0. In *Specification of the Bluetooth System*, pages 86–89 (Volume 1), 473–478 (Volume 3), 524–525 (Volume 3), 603–610 (Volume 3), 68–70 (Volume 6), Jun 2010.

[Blu14a] Bluegiga Technologies. *BLE112 DATA SHEET*, 1.44 edition, Mar 2014.

[Blu14b] Bluetooth SIG. BLUETOOTH SPECIFICATION Version 4.2. In *Specification of the Bluetooth System*, pages 88 (Volume 1), 93–94 (Volume 1), 133–136 (Volume 1), 605–606 (Volume 3), Dec 2014.

[Blu16a] BlueKrypt. Cryptographic Key Length Recommendation - NIST Recommendations (2016). 2016. `https://www.keylength.com/en/4/`. Accessed: 2016-10-06.

[Blu16b] Bluvision Inc. *SPECIFICATION SHEET - BLUFI*, 1.3 edition, 2016. `http://bluvision.com/wp-content/uploads/2016/12/Specs-BluFi1.3.pdf`. Accessed: 2016-12-20.

[Cer09] Certicom Corporation and D. Brown. Standards for Efficient Cryptography 1 (SEC 1) – SEC 1: Elliptic Curve Cryptography. *Standards for Efficient Cryptography*, pages 10–11, May 2009.

[CIS15] CISC Semiconductor GmbH. COYERO Executive Summary. 2015.

[CLH16] C. Chang, S. Li, and Y. Huang. Building Bluetooth Beacon-based Network for Spatial-Temporal Data Collection. In *Proceedings of the 2016 International Conference on Communication and Information Systems*, ICCIS '16, pages 91–95. ACM, 2016.

[COY16] COYERO Inc. COYERO access – Docs API. 2016. `https://www.coyero.biz/coyero/docs/api/`. Accessed: 2017-03-07.

[DHTS13] A. Dementyev, S. Hodges, S. Taylor, and J. Smith. Power Consumption Analysis of Bluetooth Low Energy, ZigBee, and ANT Sensor Nodes in a Cyclic Sleep Scenario. In *Proceedings of IEEE International Wireless Symposium (IWS)*. IEEE, Apr 2013.

[evo] evopark GmbH. Evopark. `https://www.evopark.de`. Accessed: 2017-03-31.

[FMA+16] M. Fujimoto, S. Matsumoto, Y. Arakawa, H. Suwa, and K. Yasumoto. Development of BLE-Based Multi-hop Communication System for Detecting Slope Failure Using Smartphones. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 16–21, Aug 2016.

[Goo16] Google. Android Gson Library. Oct 2016. `https://github.com/google/gson`. Accessed: 2017-03-07.

[GOP12] C. Gomez, J. Oller, and J. Paradells. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. In *Sensors*, pages 11734–11741. MDPI AG, Aug 2012.

[Int14] Intel Corporation. *Developing Solutions for the Internet of Things*, 2014. `http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/developing-solutions-for-iot.pdf`. Accessed: 2017-01-20.

[ITU15] ITU Telecommunication Standardization Sector (ITU-T). Recommendation ITU-T X.690 – Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). *SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY*, pages v, 1–10, 18–19, Aug 2015.

[Jec15] C. Jechlitschek. Radio Frequency IDentification - RFID A Survey Paper on Radio Frequency IDentification (RFID) Trends. 2015.

[Kni10] Peter Knight. Cryptosuite. May 2010. `https://github.com/Cathedrow/Cryptosuite`. Accessed: 2016-11-22.

[MA16] Dennis Mathews and Apple Inc. System Frameworks Session – What's New in Homekit. *WWDC16 Session 710*, pages 1–60, 2016.

[Mac16] Ken MacKay. micro-ecc. Jul 2016. `https://github.com/kmackay/micro-ecc`. Accessed: 2016-10-27.

[Mag16] Cristian Maglie. FlashStorage library for Arduino. Oct 2016. `https://github.com/cmaglie/FlashStorage`. Accessed: 2016-10-27.

[Max] Maxim Integrated. Maxim Integrated DS3660. `https://www.maximintegrated.com/en/products/power/supervisors-voltage-monitors-sequencers/DS3660.html`. Accessed: 2017-03-30.

[Max16] Maxim Integrated. *DS28C36 Deep Cover Secure Authenticator (Abridged Data Sheet)*, Jun 2016. `https://datasheets.maximintegrated.com/en/ds/DS28C36.pdf`. Accessed: 2016-12-20.

[MERH15] W. McGrath, M. Etemadi, S. Roy, and B. Hartmann. fabryq: Using Phones as Gateways to Prototype Internet of Things Applications using Web Scripting. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 164–173. ACM, 2015.

[NTS+15] J. Nieminen, TeliaSonera, T. Savolainen, M. Isomaki, Nokia, B. Patil, AT&T, Z. Shelby, ARM, C. Gomez, and Universitat Politecnica de Catalunya/i2CAT. RFC 7668 – IPv6 over BLUETOOTH(R) Low Energy. Oct 2015. `https://tools.ietf.org/html/rfc7668`. Accessed: 2017-03-27.

[Onya] Onyx Beacon Ltd. Onyx Beacon: Beacon Hardware. `http://www.onyxbeacon.com/beacon-hardware`. Accessed: 2017-04-28.

[Onyb] Onyx Beacon Ltd. Onyx Beacon: Tracko. `http://www.onyxbeacon.com/tracko`. Accessed: 2017-03-28.

[P. 04] P. Bergamini and Autostrade per l'Italia S.P.A. Il transponder più conosciuto: il Telepass. Nov 2004. `http://www.ttsitalia.it/file/Infomobility/transponder%20telepass.pdf`. Accessed: 2017-03-11.

[Row14] Jeff Rowberg. BGLib - Arduino. Feb 2014. `https://github.com/jrowberg/bglib/tree/master/Arduino`. Accessed: 2016-10-20.

[SC11] Phil Smith and Convergence Promotions LLC. Power Comparison Wireless Technologies. Aug 2011. `https://www.digikey.com/en/articles/techzone/2011/aug/comparing-low-power-wireless-technologies`. Accessed: 2017-02-05.

[Sil] Silicon Labs. *UG164: Thunderboard React (RD-0057-0201) User's Guide*, 1.2 edition. `http://www.silabs.com/documents/public/user-guides/ug164-thunderboard-react.pdf`. Accessed: 2016-10-10.

[Sil15] Silicon Labs. *BGSCRIPT SCRIPTING LANGUAGE DEVELOPER GUIDE*, 4.1 edition, Dec 2015. `https://www.silabs.com/documents/login/user-guides/UG209.pdf`. Accessed: 2016-10-12.

[Spo16] Michael Spoerk. IPv6 over Bluetooth Low Energy using Contiki. Master's thesis, Institute for Technical Informatics, Graz University of Technology, Oct 2016.

[TCDD14] K. Townsend, C. Cuf, A. Davidson, and R. Davidson. *Getting started with Bluetooth Low Energy - TOOLS AND TECHNIQUES FOR LOW-POWER NETWORKING*, pages 15–44, 51–66. OReilly Media, 2014.

[TEL] TELEPASS S.P.A. Telepass Parcheggi in Struttura. `https://www.telepass.com/servizi/parcheggi-in-struttura`. Accessed: 2017-03-11.

[Tex15] Texas Instruments. *TI Designs Multi-Standard CC2650 SensorTag Design Guide*, Mar 2015. `http://www.ti.com/lit/ug/tidu862/tidu862.pdf`. Accessed: 2016-10-11.

[Tex16a] Texas Instruments. *CC13xx, CC26xx SimpleLink Wireless MCU Technical Reference Manual*, Jun 2016. `http://www.ti.com/lit/ug/swcu117f/swcu117f.pdf`. Accessed: 2016-10-10.

[Tex16b] Texas Instruments. *CC2650 SimpleLink Multistandard Wireless MCU*, Jul 2016. `http://www.ti.com/lit/ds/swrs158b/swrs158b.pdf`. Accessed: 2016-10-10.

[TIB+09] S. Turner, IECA, D. Brown, Certicom, K. Yiu, Microsoft, R. Housley, Vigil Security, T. Polk, and NIST. RFC 5480 – Elliptic Curve Cryptography Subject Public Key Information. 2009. `https://tools.ietf.org/html/rfc5480`. Accessed: 2017-03-27.

[TMTG13] R. Tabish, A. Ben Mnaouer, F. Touati, and A. M. Ghaleb. A comparative analysis of BLE and 6LoWPAN for U-HealthCare applications. In *2013 7th IEEE GCC Conference and Exhibition (GCC)*, pages 286–291, Nov 2013.

[TSPA16] S. Theodoros, S. Sotiriadis, E. Petrakis, and C. Amza. Internet of Things Data Management in the Cloud for Bluetooth Low Energy (BLE) Devices. In *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, ARMS-CC'16, pages 35–39. ACM, 2016.

[WA14] Robert Walsh and Apple Inc. Core OS – Designing Accessories for iOS and OS X. *WWDC14 Session 701*, pages 59–75, 97–98, 2014.

[YKKK15] J. Yim, S. Kim, N. K. Kim, and Y. B. Ko. IPv6 based real-time acoustic data streaming service over Bluetooth Low Energy. In *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 269–273, Aug 2015.