Design of a secure e-bike control based on Bluetooth Low Energy

Eric Jedermann

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Master Thesis

Design of a secure e-bike control based on Bluetooth Low Energy

by

Eric Jedermann

July 1, 2019

Technische Universität Kaiserslautern
Department of Computer Science
Distributed Systems Lab

Supervisor: Prof. Dr.-Ing. Jens B. Schmitt
Examiner: M. Sc. Lukas Tremper

Abstract

In this master thesis, an e-bike control based on Bluetooth Low Energy (BLE) is designed. To estimate the suitability of the protocol, the power consumption and robustness against interferences of BLE is compared to common wireless communication technologies, such as WiFi and ZigBee. Afterwards the BLE architecture and the different paring methods are introduced. To design a secure interface, possible attacks against BLE are regarded and appropriate countermeasures are mentioned. Based on the derived security requirements and the functional requirements, that come from the bicycle industry, the interface is designed. It is implemented in two android applications to demonstrate the functionality of the services. In the end the transmitted data are sniffed and analyzed for their security.

Zusammenfassung

In dieser Masterarbeit wird eine E-Bike Steuerung auf Basis von Bluetooth Low Energy (BLE) entwickelt. Um die Eignung des Protokolls abzuschätzen, wird die benötigte Leistung und Robustheit gegenüber Signalstörungen von BLE mit anderen Technologien zur drahtlosen Datenübertragung, wie WiFi und ZigBee, verglichen. Anschließend werden die BLE Architektur und die unterschiedlichen Kupplungsmethoden erklärt. Um eine sichere Schnittstelle zu entwickeln, werden mögliche Angriffe gegen BLE betrachtet und entsprechende Gegenmaßnahmen aufgeführt. Basierend auf den Sicherheits- und Funktionsanforderungen, kommend aus der Fahrradindustrie, wird die Schnittstelle entwickelt. Diese wird in zwei Android Anwendungen implementiert um die Funktionalität der Services zu demonstrieren. Abschließend werden die übertragenen Daten abgehorcht und auf ihre Sicherheit analysiert.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Alle wörtlich oder sinngemäß übernommenen Zitate sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Kaiserslautern, den July 1, 2019

_____

Eric Jedermann

Contents

List of Figures

List of Tables

1 Introduction

In the last ten years about 4.8 million e-bikes have been sold in Germany. The nearly 1 million sold e-bikes in the year 2018 show that the trend of using e-bikes is continuously growing[ZIV 13b][ZIV 13a][ZIV 19].

The e-bikes are equipped with a battery and an electrical powered engine to support the driver. The engine supports the driver while he pedals until the vehicle reaches a speed of typically 25 km/h, than the support stops.

A second type of electric supported bikes are the pedelecs. They support the driver until a speed of 45 km/h.

Currently the engine is controlled via a control panel, that is permanently mounted on the handlebar.

At the same time, many bikes tend to attach additional bike computers or their smartphones to the bike to record their own bike tours or follow a guided tour. To do so, the attached device can use its own build in sensors or the bike can be equipped with additional sensors that can be accessed by the device[Kiefer 16][Herrero 15].

Although the e-bike is already equipped with many useful sensors. But those internal sensors of the e-bikes can not be used by the attached devices. The internal sensors are capsuled in a closed system that communicates only via wired connections, as the CAN bus. To open the closed systems and enable a utilization of the internal e-bikes sensors, this thesis designs a communication interface based on Bluetooth Low Energy (BLE). Additionally it is considered to control the electric support engine of the bike with this interface.

Therefore chapter 2 will examine the energy consumption of Bluetooth Low Energy and compare it with other wireless communication standards to get an overview of the current situation and to estimate if Bluetooth Low Energy is a suitable candidate for this purpose. After clarifying the energy consumption, the BLE architecture will be explained in chapter 3.

Designing a new interface to connect a device with the rest of the world, always offers the possibility to abuse this connection. In the past vulnerabilities in the communication interfaces and data management systems of several systems like fitness trackers, health-care monitors or smart home devices were found [Cyr 14] [Sivakumaran 18].

Often the devices use weak static PIN codes for authentication, user specific settings can be overwritten or the implemented safety mechanism is easy to overcome. A par-

ticularly precarious security vulnerability existed in 2011 where the 'Sexual activity tracked by fitbit shows up in google search results'[Rao 11].

To avoid such security flaws in the e-bike interface, additional attention is being paid to already known attacks against BLE and possible counter measures. Chapter 4 gives an overview of the possible attacks against BLE and how suitable counter measures can be applied. This will be used to derive security requirements for the interface.

To capture the functional requirements and create a beneficial communication interface, multiple companies from the bike industry contributed their requirements on such an interface. The companies came from various areas, for example component supplier, bike computer manufacturer and smartphone application developers.

The functional requirements from the partner companies are included in chapter 5. Additionally this chapter will include the security requirements to prevent the attacks from the previous chapter.

Knowing all the required aspects, chapter 6 will introduce the designed communication interface and the mechanisms to meet the raised requirements.

Chapter 7 describes a setup where the introduced BLE interface is implemented in two android applications. One application emulates an e-bike, the second application connects to the emulated bike, read the provided values and control the bike.

The communication is recorded and analyzed for the security. The results of the tests are presented in chapter 8.

In the end, chapter 9 summarizes the work, brings up some possible future improvements for the designed services and describes the planned proceeding with the developed profile.

2 Comparison of radio technologies

To remote control a device, multiple technologies are available. Some technologies as ANT or Bluetooth are already used in the bicycle industry. Other technologies as ZigBee or WiFi are known from home automation. The goal of this chapter is to examine if Bluetooth Low Energy (BLE) is a suitable technology for controlling an e-bike. For this purpose different technologies and their setups are introduced. The most important aspect is the power consumption of the communication technology, since an e-bike is a battery powered device. So different experimental setups will be used to get an overview of the power consumption. Additionally the coexistence strategies of each technology will be mentioned, since most of the technologies are working in the free 2.4 GHz band. As a third factor the possibility of an encrypted communication will be considered.

To compare the different technologies, each technology will be introduced and the results of multiple projects, using this technology, are introduced. In the end a brief summary will conclude the overview and give a statement if BLE is suitable for controlling an e-bike.

2.1 ZigBee

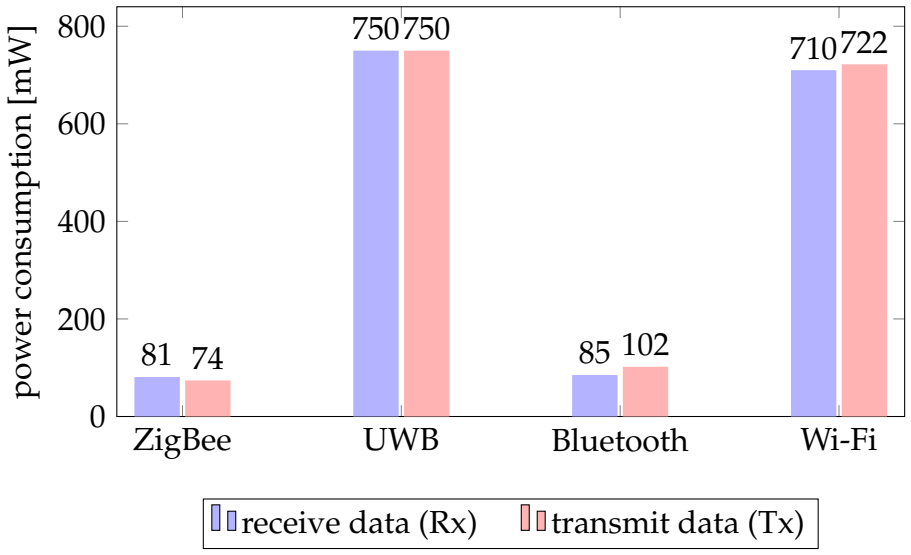
ZigBee is an open standard, which corresponds to the IEEE 802.15.4 standard. The typical communication range of ZigBee devices are 10 to 100 meter, while the maximal data rate is 250 kb/s. [Smith 11]

It supports different network topologies: from a complex mesh network, over a star topology, to a simple P2P network topology. Also broadcasting and scanning is an option.

2.1.1 Power consumption

Lee et al [Lee 07] captured the energy consumption of multiple technologies and normalized them by the transfered data. For ZigBee they measured a current of 24.7 mA during receiving and 27 mA at transmitting at 3V. With this measurements the power consumption can be calculated. Sending needs 74 mW and receiving consumes 81 mW, this is visualized in figure 2.1. Since the data rate of ZigBee is 0.25Mb/s, the

Figure 2.1: Power consumption of sending and receiving [Lee 07]



normalized power consumption can be calculated. For transmitting data it is 0.2964 μW/b and 0.324 μW/b for receiving. The results are also shown in figure 2.4. Additionally the efficiency of the protocol was examined: In ZigBee the maximal data payload of a packet is 102 byte, while the maximal overhead of a packet are 31 byte, which leads to a maximal total package length of 133 byte. Dividing the payload by the total length leads to a efficiency of 77% for this protocol.

Smith et al [Smith 11] also focused on the consumed power per bit. For ZigBee he uses a power consumption of 35.7 mW, visible in in figure 2.2. Based on this, he calculates a efficiency of 185.9 μW/bit, shown in figure 2.5. Comparing the power consumption, Lee's calculation is more than 600 lower. Also this value differs so much from the rest of Smith's measured values, that this result is not shown in graph 2.5. Smith calculated a peak current of 40 mA, so a common CR2032 battery cell would not be able to power the device, since this battery only delivers 15 mA.

A third study a measurement with different cyclic sleep intervals was performed over a longer time. Dementyev and his team compared ZigBee, Bluetooth Low Energy and ANT sensors to get a reliable result[Dementyev 13]. The sensors were powered at 3.3 V. They transfered 8 byte random data in intervals between 5 and 120 seconds. For the 5 second intervals a average current of 0.45 mA was measured on the ZigBee node. At the 120 seconds interval the average current decreased to 16 uA. Their results are visualized in figure 2.3. Each ZigBee connection event took about 250 ms to wake up, reestablish a new connection and transmit the data.

In a last study, regarding the power consumption Georgakakis et al [Georgakakis 10] compared ZigBee, Bluetooth and Bluetooth Low Energy. Looking at their first exam-

ined technology, they specify a power consumption of 30 mW for ZigBee.

2.1.2 Security

To archive security in a network, ZigBee uses an 128-bit AES based encryption in CCM* mode. This mode enables the usage of one key for all security levels. So all layers (MAC, Network and Application layer) use the same encryption key[Li 10].

Despite this shared key, in ZigBee networks exist two important types of keys to encrypt the communication [Fredriksson 17]: a network key and a link key. The first is used to encrypt messages that are broadcasted to many devices, so this key is equal in all devices of a network. The link key is used for end-to-end encryption of between two devices.

To manage these keys, ZigBee supports a distributed and a centralized security model [Li 10]. In the distributed model the routing devices handle the key distribution themselves and new devices adopt the keys from the existing network. The centralized model is managed by a trusted center that handles key management and the integration of new nodes. In this model new devices often use a pre-installed default key to encrypt the initial communication with the trusted center.

Fan et al [Fan 17] argued that the usage of a pre-installed key is a moment of insecurity, but it is a good trade off between security and simplicity. Besides the Zigbee 3.0 specification added the possibility of out of band key pre installation. This can be realized with individual qr-codes for each device or via NFC communication to generates a unique joining key.

In this context, Fan [Fan 17] also referred to a worm that was infecting Philips Hue Light Bulbs, using this weakness. The bulbs had a ZigBee interface with hard coded keys to access their control. In this way the worm was able to took over the lights and locally spread to nearby bulbs up to 400 meters away. A firmware update from Philips solved the problem.

2.1.3 Robustness against interferences

As already mentioned, ZigBee is primary operating in the free 2.4 GHz band. Other possible frequency bands for ZigBee are at 868 MHz and 915 MHz [Domínguez 12]. The ZigBee specification defines eleven communication channels in those lower frequency ranges and 16 channels in the 2.4 GHz range. The focus of this examination is on the 2.4 GHz band, where all ZigBee channels are overlapping with WiFi channels. Since WiFi is a widely spread wireless technology with high transmission power, all other technologies have to deal with the presence of WiFi and measurements often focus on interferences that come along with this.

To find out how much WiFi influences ZigBee Dominguez et al [Domínguez 12] analyzed the interference. They found some safe zones where ZigBee works relatively stable, since the overlapping WiFi channels are almost never used. Additionally they measured the influence of the different WiFi channels to ZigBee channel 26, at 2.480 GHz. The three upper WiFi channels 11, 12 and 13 (at 2.462, 2.467 and 2.472 GHz) are influencing ZigBee. With increasing distance to the WiFi device, the ZigBee device had a better packet reception rate. At five meter distance the packet reception rate was at 30%, increasing to 15 meter with no influence from the WiFi.

Huang et al [Huang 10] analyzed the CSMA mechanisms of WiFi devices and measured that their transmitters cannot detect ZigBee signals due to the unequal power levels of both technologies. So they analyzed WiFi communication and found that the WiFi transmissions are highly clustered. Based on this knowledge, they created a model that characterizes the white space between WiFi frame clusters. This model was used to design a ZigBee control protocol, that is able to predict the size of white space in WiFi traffic and adapt the size of ZigBee frames to optimize the throughput. With this control protocol they were able to gain a four time higher throughput, compared to the default ZigBee MAC protocol.

2.2 ANT and ANT+

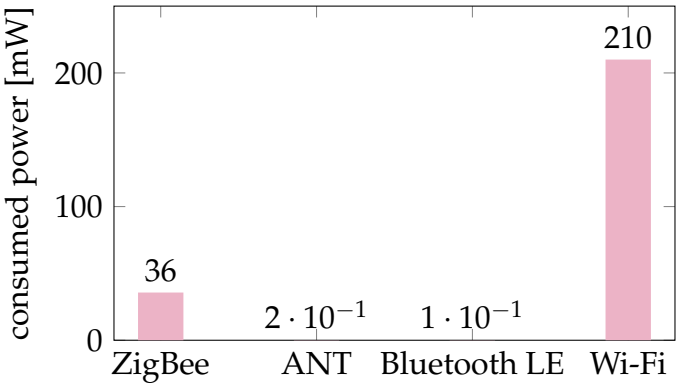
The Adaptive Network Technology (ANT) was published by the sensor company Dynastream [Smith 11]. ANT was introduced 2003, 8 years later the improvement ANT+ was published. The enhanced protocol supports the same network topologies as ZigBee: mesh, star, peer to peer, broadcasting and scanning. But it allows only a maximal throughput of 20 kb/s, which is about 13% of the throughput of ZigBee.

2.2.1 Power consumption

Smith measured the power consumption of ANT devices. The device were powered at 3 V and consumed 61 μA, which leads to 183 μW. Since the devices transmited 256 bits of data, the power consumption per bit is 710 nW, as visible in 2.5. Further he determined the protocol efficiency at 47%, which is the ratio of payload to the over all package length. [Smith 11]

Demetyev measured in his cyclic sleep scenario a power consumption that was comparable to the consumption of the ZigBee technology. Figure 2.5 shows the average current at the 5 seconds interval was at 0.43 mA, and reduced to 28 μA at the 120 seconds interval, while the devices were powered at 3.3 V. These values are roughly on the same level as his measurements of the ZigBee node. Additional he measured a increased connection time of the ANT protocol. It needs about 930 ms for a connection event, which is about 3.7 times longer than ZigBee.

Figure 2.2: Power consumption of different technologies [Smith 11]



Another project designed an experimental sensor network for bicycle torque performance measurements[Gharghan 15]. They compared two communication technologies to transfer their data from the bike to a base station: ZigBee and ANT. While using the ANT technology, power savings of 98% were achieved, regarding only the communication. Referring to the whole system, power savings of 30% were measured.

2.2.2 Security

As ZigBee, ANT works in the 2.4 GHz band. It devices the frequency band into 125 channels with a bandwidth of 1 MHz. Each established connection is encrypted by a unique connection key. AES with 128-bit is used for the encryption. [Mehmood 15]

Further studies about interference tests with ANT or security examinations are not available. Which makes it unpredictable how ANT would react on interfering signals.

2.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) belongs to the IEEE 802.15.1 standard. As all technologies above, BLE is working in the 2.4 GHz spectrum, with a maximum throughput 2 Mb/s [BluetoothSIG 16]. In contrast to the previous technologies, BLE does not natively implement mesh networking [BluetoothSIG 10]. In 2017, seven years after the first BLE specification was released, the Bluetooth Mesh Devices specification [BluetoothSIG 17] started to define a mesh networking on top of BLE. Network topologies as star-network, peer to peer, scanning or broadcast networks are supported by native BLE devices [BluetoothSIG 10].

2.3.1 Power consumption

Smith et al [Smith 11] calculated the efficiency of BLE advertising packets, at 66%. The efficiency of a data packet ranges from 6% to 96%. In the worse case a package with 1 byte payload and 15 byte total length has an efficiency of 6%. In the best case a package with 255 byte payload and 265 byte total length has an efficiency of 96%. How this different package sizes are possible is described in chapter 3.2. Furthermore Smith calculated a power consumption of 0.147 mW during advertising, visible in figure 2.2. Based on this and 960 transmitted bits he determined the power efficiency of BLE advertising at 0.153 μW/bit, which is also shown in figure 2.5. During transmission a peak current of 12.5 mA was measured. This enables a common CR2032 coin cell with maximal 15 mA to supply the chip with enough energy.

Based on this measurements, Kamath et al [Kamath 10] made an experiment with a CC2541 BLE-system-on-a-chip and measured a average current of 0.024 mA while the chip had a connection to another BLE module. A CR2032 battery has a capacity of 230 mAh, which enables the chip to stay connected for about 400 days.

Dementyev et al [Dementyev 13] measured in his cyclic sleep scenario a current between 0.18 mA, in the 5 seconds sleep cycle, and 10 uA in the 120 seconds sleep cycle, visible in figure 2.3. Each connection event took about 1.15 seconds, to transmit all the data. This is more than four times longer than the ZigBee connection events, as explained in chapter 2.1. The increased connection time might be caused by the frequency hopping of BLE which takes some time to connect, compared to the fixed frequency of ZigBee.

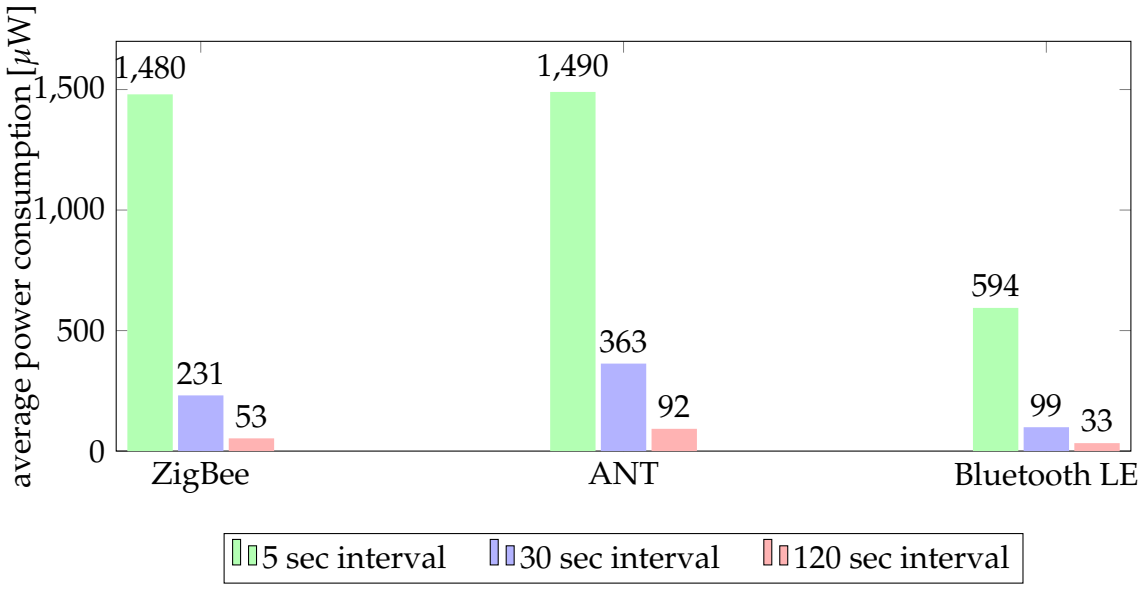
Bluetooth low energy was designed to consume a maximal power of 10 mW, as Georgakakis et al [Georgakakis 10] mentioned. Comparing this with the experimental measurements above, BLE seems to consume even less energy than allowed.

2.3.2 Security

To evaluate the security of an established, transmitting connection, Fredriksson et al [Fredriksson 17] analyzed different attacks against BLE. They focused on Bluetooth specification 5.0 [BluetoothSIG 16], which is equal to specification 4.2 from the security point of view [BluetoothSIG 14]. BLE prevents attacks as a replay attacks or eavesdropping by singing and encrypting via the AES-CCM algorithm, described in chapter 3.7.3. In the older specifications 4.0 and 4.1 man in the middle attacks were possible due to insecure paring modes, further described in chapter 3.7.1. With specification 4.2 this problem was solved. So Fredriksson identifies a denial of service attack as the most powerful attack against BLE.

One point to attack BLE is before a connection is established and a data transmission can take place. This phase is the advertising phase. In this phase a BLE device is broadcasting its ability to establish connections and share informations. Brauer et al

Figure 2.3: Power consumption in a cyclic sleep scenario [Dementyev 13]



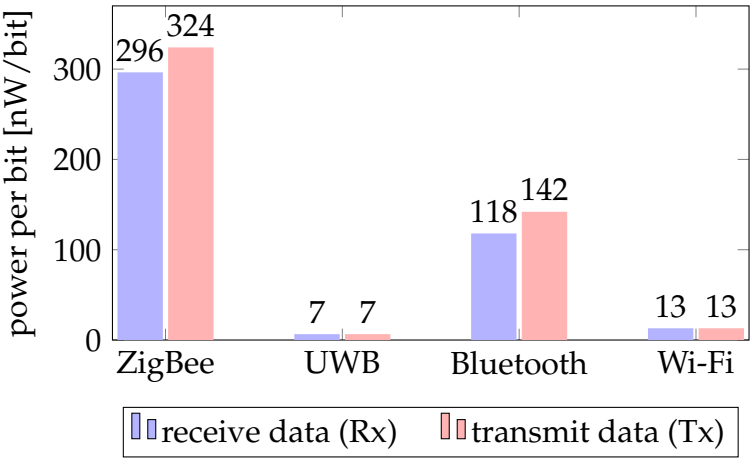
[Brauer 16] tested how good this advertising can be jammed. They designed a simple and cheap reactive jammer to jam only on the BLE advertising channels 37, 38 and 39. The advertising BLE device was sending 46 Byte at 0 dBm power, while the jammer transmitted pseudo random bits at 4 dBm. Depending on the distance between jammer and advertiser the rate of undamaged advertising packets was increasing. At a distance less than 1 m all packets were jammed. Between 1 and 5 meter the rate was increasing from 10% to 90%.

If a connection is established, BLE uses channel hopping to avoid accidental interferences. But a jammer could follow this hops and deny a communication, further described in chapter 4.1. With Bluetooth specification 5.0 [BluetoothSIG 16] in December 2016, the channel selection algorithm number 2 was introduced. It pseudo randomly selects a communication channel, which makes it much harder for an attacker to follow a specific BLE connection.

2.3.3 Robustness against interferences

BLE uses frequency hopping during a connection to avoid accidental interferences. To evaluate the usability of Bluetooth Low Energy in realistic environments, Al Kalaa et al [Al Kalaa 16] measured the strength of interfering signals on the 2.4 GHz band. The measurements were done during a sport event, in a university food court and in a Hospital intense care unit. They analyzed the power of the measured signals at the BLE channel frequencies. 30% of the BLE channels are usable with a transmis-

Figure 2.4: Efficiency of sending and receiving [Lee 07]



sion power of -105 dBm, with exception of the hospital environment with only 1% channel availability. An increased transmission power of -95 dBm also increased the channel availability to more than 50% in all environments. Adding 10 additional dBm transmission power, to -85 dBm, boosts the channel availability to more than 90%.

2.4 Bluetooth

Standard Bluetooth refers to the IEEE 802.15.1 standard, as well. To differentiate the normal Bluetooth from BLE the term BR/EDR Bluetooth is used, which is an abbreviation for basic rate / enhanced data rate. This refers to the first version of Bluetooth from the year 1999 [BluetoothSIG 99] where a basic data rate of 732,2 kb/s was introduced. In 2004 the core specification 2.0 with the enhanced data rate of 2.1 Mb/s was released [BluetoothSIG 04].

2.4.1 Power consumption

Lee et al [Lee 07] measured in his comparative study the power consumption of the devices during sending and receiving. It is 102 and 85 mW, as visible in figure 2.1. Furthermore the power consumption per bit can be calculated: 142 nW/b for transmitting and 118 nW/b for receiving. This is shown in figure 2.4. Additional Lee calculated an encoding efficiency of 94%.

To complete the analysis of power consumption, Georgakakis [Georgakakis 10] argues that Bluetooth was mentioned to consume about 100 mW. The measurements above confirm this value.

2.4.2 Security

Since core specification 2.1 [BluetoothSIG 10] Bluetooth uses a security scheme called 'secure simple paring'. This introduced paring methods as numeric comparison or passkey entry. With core specification 4.2 these paring methods were introduced in BLE with the 'secure connections only mode'. They are further explained in chapter 3.7.1.

BR/EDR Bluetooth and BLE use the same AES-CCM encryption to archive integrity and confidentiality. In terms of safety, these two technologies are at the same security level. Due to the use of the same security mechanisms as paring, signing and encryption Fredrikssons [Fredriksson 17] investigations can be transfered to this technology.

2.4.3 Robustness against interferences

BR/EDR Bluetooth uses the same frequency hopping as Bluetooth Low Energy, with more available transmission power. So Al Kalaas [Al Kalaa 16] measurements of the noise power at the used frequency band is also informative for this technology. BR/EDR Bluetooth has the advantage of higher transmission power, which increases the availability of channels.

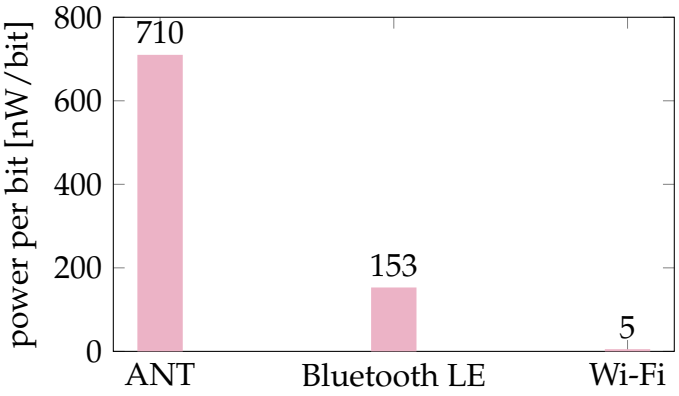
As Fredriksson already described, a denial of service attack is considered to be the most effective. Koppel et al [Köppel 13] described the procedure to start a denial of service attack on BR/EDR Bluetooth based on specification 4.0. To determine the hopping sequence, he identifies the Upper Address Part (UAP) and Lower Address Part (LAP) of the Bluetooth master device. With this, it was possible follow the connections and transmit pseudo random bits to jam a specific connection without prior knowledge.

This method to jam a BR/EDR Bluetooth connection should still work, since the current specification 5.1 [BluetoothSIG 19] defines no additional channel selection algorithm for this technology.

2.5 WiFi

WiFi might be the most popular wireless communication technology. It refers to multiple standards as IEEE 802.11 a/b/g. This technology was designed for devices with a substantial power supply that need high data rates, up to 54 Mb/s. WiFi only allows peer to peer connections and creates a star topology with multiple peer to peer connections to different devices.

Figure 2.5: Efficiency of different technologies [Smith 11]



2.5.1 Power consumption

Smith et al [Smith 11] used the standard IEEE 802.11v, which provides a reduced power consumption, while archiving a throughput of 6 Mb/s in their setup. The reduced power consumption still needs 210 mW, as figure 2.2 shows. They were able to reach a very high efficiency of 5.25 nW/bit, as visible in figure 2.5. The power efficiency per bit is 30 times higher than the efficiency of BLE. At the same time the peak current of 116 mA is more than 9 times higher.

Also Lee et al [Lee 07] analyzed the WiFi technology. They measured a power consumption of 722 mW during sending and 710 mW while receiving. This is shown in figure 2.1. The normalized consumption for receiving and transmitting data is 13 nW/b, visualized in figure 2.5. This agrees with the researches of Smith, that WiFi is much more energy efficient per bit than ZigBee, Bluetooth or BLE. But with the high data rates of 54 Mb/s a high power consumption during communication comes ahead. Additionally Lee calculated the WiFi encoding efficiency at 97%.

2.5.2 Security

For WiFi the four security protocols are available to encrypt the communication. The first developed protocol is WEP, Wired Equivalent Privacy. This protocol is already known to be broken. The major weaknesses are the short 40 bit key, a too short internal used initialization vector of 24 bit and the broken algorithm itself. Today encryption can be broken in some minutes. [Rowan 10][Katz 10]

The successor protocol WPA, WiFi Protected Access, was designed as a short-term fix for the problems of WEP. It can be updated on the old WEP using devices via a firmware upgrade [Rowan 10]. It has an improved encryption with a 128 bit key

and uses the Temporal Key Integrity Protocol (TKIP) for encryption. This protocol generates individual keys for each data packet to encrypt. Still, WPA can be cracked in about 15 minutes[Tews 09].

WPA2 was designed more carefully. It uses AES encryption for the data packets instead of TKIP. The devices need additional hardware to implement the AES encryption, so the old devices have to be replaced [Rowan 10][Katz 10]. In 2017 scientists from Belgium fround a key reinstallation attack, called KRACK, that can be applied during the handshake. This attack fools the connecting client and persuades him to reuse some already used encryption keys [Vanhoef 17].

The newest protocol WPA3 was released in 2018. WPA3 uses a 192 bit encryption and a new handshake protocol called Simultaneous Authentication of Equals (SAE). SAE is also know as Dragonfly Key Exchange Protocol. It fixes the KRACK attack from WPA2. Additional WPA3 implements forward secrecy and prevents offline dictionary attacks [Sahinaslan 19]. This year scientists found multiple attacks against the dragonfly protocol. The attacks are summarized by the code name 'dragonblood'. They enable an attacker to recover the password by abusing timing or cache-based side-channel attacks [Vanhoef 19].

2.5.3 Robustness against interferences

Since WiFi is the technology with the highest transmission powers, except UWB, there are no serious impacts on the WiFi communication due to interferences. Huang [Huang 10] found out that the transmitters in WiFi devices cannot detect ZigBee communication, as already mentioned in the previous chapter 2.1.

2.6 UWB

The standard IEEE 802.15.3 referees to ultra wide band technology. It is recommended for short range and high data rate. UWB works in a wide frequency space between 3.1 and 10.6 GHz with a channel width between 500 MHz and 7.5 GHz. It allows only peer to peer network topologies.

2.6.1 Power consumption

Lee et al [Lee 07] describe possible bandwidths of 110 Mb/s and even more. Lee calculated an encoding efficiency of 98% and measured a power consumption of 750 mW, shown in figure 2.1. It is roughly at the same level as the power consumption of WiFi and more than 7 times higher than the power consumption of Bluetooth or ZigBee. Based on the high data rates the normalized power consumption is 6.5 nW/b for receiving and transmitting data packets, visible in figure 2.4.

2.6.2 Security

Ko et al proposes a signaling model to utilize physical properties of UWB to enhance the security of UWB communication. They use a shared key to enable time-hopping and binary pulse modulation schemes. In combination with a lightweight cryptographic protocol, it should archive a high level of security [Ko 10].

In so far as known, there are no further investigations about UWB supporting higher level cryptographic protocols.

2.6.3 Robustness against interferences

To estimate the influence of UWB Choliz et al [Chóliz 11] made investigations about the coexistence of UWB radio systems to other wireless communication systems, including Bluetooth, WiFi and ZigBee.

To measure the influence on ZigBee they describe a setup, using ZigBee channel 20 at 2450 MHz and emitting at 0 dBm. The UWB was transmitting on the band-group between 3.168 - 4.752 GHz. There was no interaction between both technologies.

But they found an influence of a Bluetooth device on the UWB channel from 3.168 to 3.696 GHz. The Bluetooth device was transmitting at maximum power of +20 dBm. An influence from UWB to a Bluetooth device was not measured. This might be based on the fact that the maximal allowed emission of UWB in the 2.4 GHz band is -85dBm/MHz.

WiFi working in the 2.4 GHz band did not affect the UWB communication and was not affected by it, vice versa. When switching WiFi to the 5 GHz band, the UWB throughput was reduced by more than 70%. Still the WiFi connection was unaffected by ongoing UWB traffic.

2.7 Summary

Comparing the different efficiencies of the introduced technologies, WiFi and UWB reached the lowest power per bit values. For transmitting large data those technologies are unbeatable. But comparing the total power consumption of the technologies the advantage of the low power technologies comes ahead. They only consume a fraction of the power, that is needed by the high data rate technologies. Therefore the efficiency of the low power technologies is far behind.

In this case, the controlling of the e-bike will only generate a small amount of data so the efficiency is not the most important factor. The overall power consumption of WiFi and UWB very high, so both technologies are not applicable for a battery powered system as an e-bike.

Summarizing the power consumption of the four remaining low power technologies a relatively clear trend is visible:

- Bluetooth consumes about 100 mW, verified by Lee and Georgakakis.
- ZigBee consumes in average 30 mW, validated by Smith and Georgakakis. Lee measured a significantly higher power of 81 mW, while Demetyev measured only 1.48 mW. The second, much lower, measurement may be caused by the sleeping interval of the experimental setup.
- The ANT power consumption level is between ZigBee and BLE. Demetyev had nearly equal values in his experiment for ZigBee and ANT. While Smith calculated similar values for ANT and Bluetooth Low Energy. Additional Gharghan archived immense power savings when switching from ZigBee to ANT.
- Bluetooth Low Energy has the smallest power consumption. Smith and Demetyev got the lowest values in their experiments with this technology. Georgakakis adds that BLE was designed to consume less power than Bluetooth and ZigBee, which encourages the observations.

Based the power consumption, Bluetooth low energy and ANT are the favorites of this selection.

Having a look at the interference robustness and the security, the WiFi and UWB technologies can be missed out, since their power consumption is too high:

- ANT has the disadvantage of an underestimated underdog, so only a few references were found. It was not possible to compare the robustness of a ANT connection in case of interferences. Additionally it seems the security architecture of ANT+ was never examined, which makes it a blank slate.
- ZigBee is well known and analyzed multiple times. It is possible to use different techniques to improve the resistance and throughput in case of interfering signals. At the same time the security architecture of ZigBee is reliable and proven. With the right setup mechanisms it is possible to archive good security.
- BR/EDR Bluetooth is very stable against accidentally interferences and archives good security by a minimal user interaction.
- Bluetooth low energy seems to be roughly as robust to interferences as BR/EDR Bluetooth. Due to the random channel selection after paring, it can be more stable against jamming than BR/EDR Bluetooth. Using Bluetooth specification 4.2 or higher archives the same level of security as BR/EDR Bluetooth since the same paring and encryption mechanisms are available.

From the interference and security point of view, Bluetooth low energy and Bluetooth are recommended.

Combining the power consumption and interference/security comparisons, Bluetooth low energy is a good choice for setups with a reduced energy availability, a small required communication range and a high security level.

3 Bluetooth Low Energy architecture

Bluetooth Low Energy (BLE), also called Bluetooth smart, was specified by the Bluetooth Special Interest Group (Bluetooth SIG) [1]. This is a consortium of more than 30.000 companies to carry the development of the widely-used communication standard. The first time Bluetooth Low Energy was introduced in the Bluetooth Core Specification 4.0 in June 2010. Until today, four new specifications were released. The newest specification 5.1 was released in January 2019.

The Bluetooth SIG also developed the "classic" Bluetooth protocol. This is called the basic-rate/enhanced-data-rate (BR/EDR) Bluetooth. Disregarding of the name, Bluetooth Low Energy has many differences to the BR/EDR Bluetooth technology. Both technologies are not compatible with each other.

Both technologies communicate in the license free ISM frequency band from 2.4 to 2.483 GHz and both divide the band in multiple frequency bands. BE/EDR Bluetooth divides the band in 79 channels with a bandwidth of 1-MHz and hops from one channel to another 1600 times a second. At the same time it archives data rates up to 3 Mbit/second. Bluetooth Low Energy divides the frequency band into 40 channels, each 2-MHz wide and jumps up to 133 times per second to another channel. The maximal data rate of BLE is at 2 Mbit/second.

Devices that offer both communication interfaces are called dual mode devices. The Bluetooth chips of these devices are enhanced to support both standards at the same time. Most up to date smartphones that run at least android 4.3 or iOS 6 are able to support Bluetooth Low Energy[Google 13][Apple 13].

In this chapter the architecture of Bluetooth Low Energy will be explained.

Bluetooth Low Energy is organized in layers, which are shown in figure 3.1. The lowest two layers, the physical and the link layer are implemented in the hardware of the controller. On top of the controller hardware, the software of the host system implements the next five layers.

Chapter 3.1 will start with the physical layer. Based on this, the following chapters will describe the architecture in a bottom-up structure.

[1] Bluetooth SIG

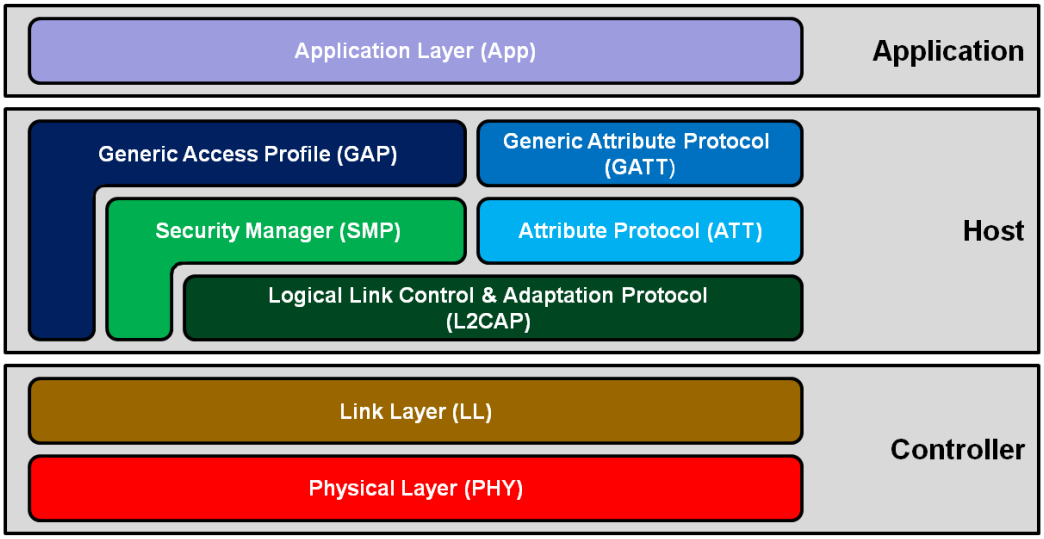


Figure 3.1: The layered architecture of BLE
Source: microchipdeveloper.com

3.1 Physical Layer

As already mentioned, Bluetooth Low Energy uses the 2.4 GHz frequency band. It uses the frequencies from 2.400 GHz up to 2.480 GHz. This band is divided into 40 channels with 2 MHz spacing for every channel. Figure 3.2 illustrates the subdivision of the frequency band.

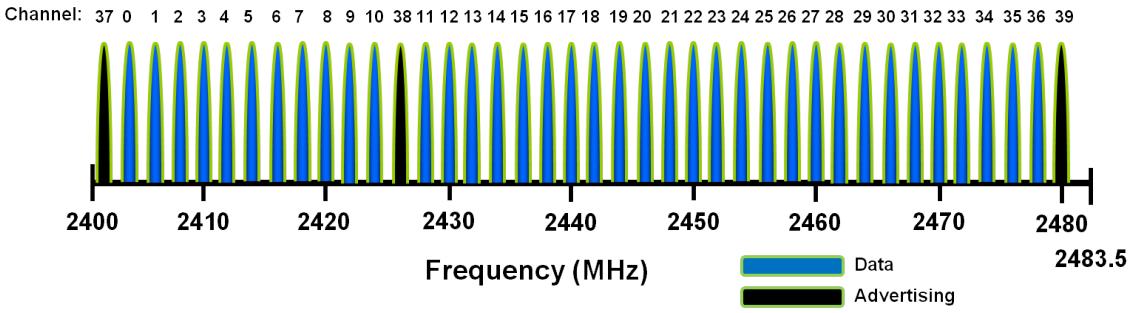


Figure 3.2: The physical channels of BLE in the 2.4 GHz band
Source: microchipdeveloper.com

The three channels with the numbers 37, 38 and 39 (at 2.402, 2.426 and 2.480 GHz) are used as advertising channels. They are spread across the whole spectrum to prevent them from being accidentally disturbed all at one time. These channels are used by BLE devices to send advertising messages. These messages are used for initiating a communication. A device that offers a possibility to connect will send an advertising message on one of the advertising channels. It will wait for possible responses and

move to the next advertising channel in case no response is received. So all advertising channels are used at the same workload.

The remaining 37 channels are data transfer channels. These channels are used by already established connections to exchange the data. Bluetooth Low Energy uses adaptive frequency hopping for established connections. During the connection setup informations about the time spent on each channel and the channel hopping sequence are exchanged. So every connection between two BLE devices has an individual sequence of the utilized channels and an individual time spent on a channel. This frequency hopping is used to reduce the impact of possible interferences.

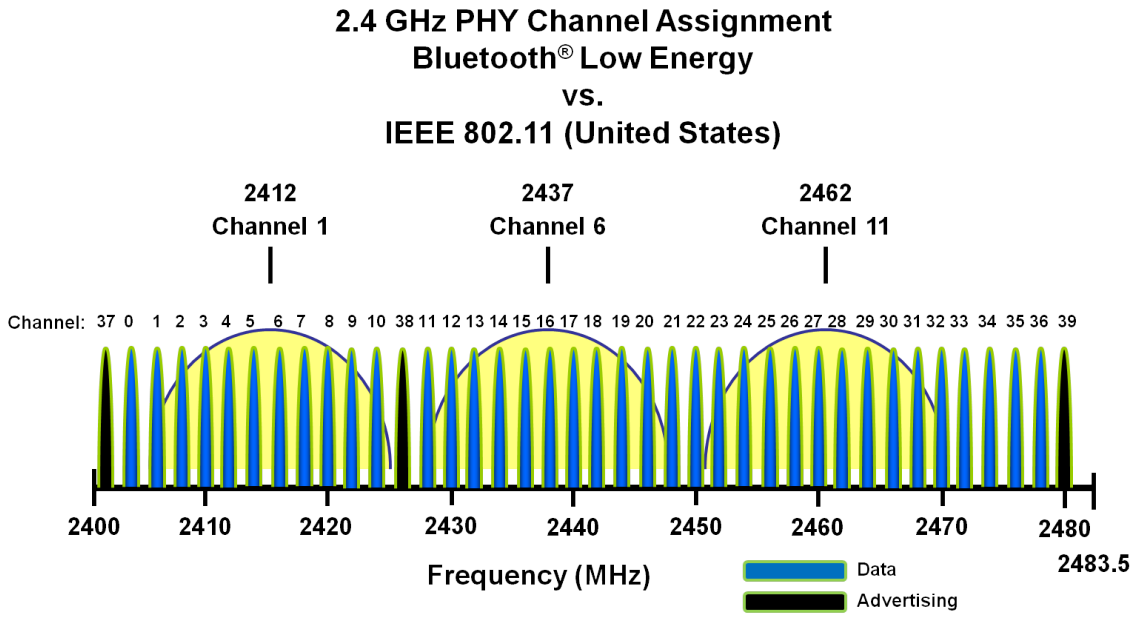


Figure 3.3: Coexistence of BLE with WiFi (IEEE 802.11) in the 2.4 GHz band
Source: microchipdeveloper.com

One possible source of interferences is WiFi, which operates in the same frequency band. Graphic 3.3 visualizes the used frequencies of both technologies. With the adaptive frequency hopping of Bluetooth Low Energy it is possible to select the frequencies with minimal interference. In this example the data channels 0, 22 and 33 to 36 are free from interfering WiFi signals. The channel selection is automatically applied by the channel selection algorithm.

Additionally BLE offers a Slot Availability Masks to predefine which channels should be used. It can be used to generally prevent a BLE device from communicating on heavily used channels. This feature was introduced in December 2016 with Bluetooth Core Specification 5.0 [BluetoothSIG 16].

An established connection allows a data transfer with up to 1 MB/s. With Bluetooth Core Specification 5.0 the possibility of a higher data rate of 2 MB/s was introduced [BluetoothSIG 16].

3.2 Link Layer

The Link Layer is the second layer, and is also implemented in the controller itself. This layer handles the packet processing, device discovery and connection procedures.

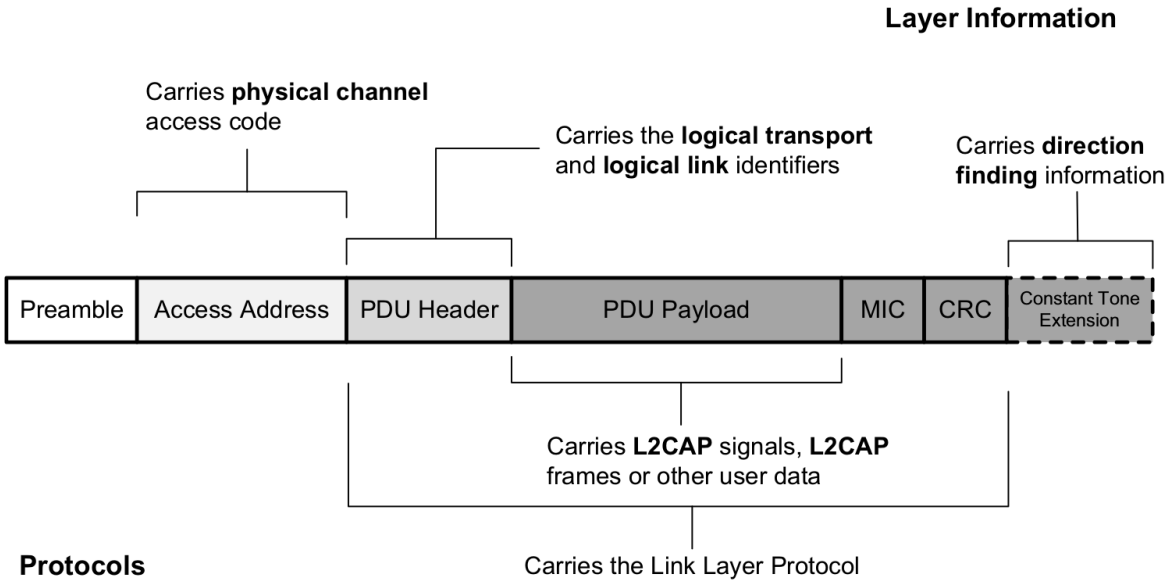


Figure 3.4: The structure of a BLE data packet
Source: Bluetooth Core Specification 5.1[BluetoothSIG 19]

Figure 3.4 shows a schematic of a Bluetooth Low Energy packet. The length of a packet can vary between 10 byte and 267 byte, exclusive the optional constant tone extension at the end. The transmission of a package takes between 44 μs and 2120 μs, depending on the size and the throughput. To understand how this layer works, the different parts of the data packet will be explained:

- The preamble contains a fixed sequence of alternating 0 and 1 bits. In a 1 Mbit/s connection, the preamble has a length of 1 byte. In a 2 Mbit/s connection the preamble has a length of 2 bytes[BluetoothSIG 16].
- The access address has a fixed size of 4 bytes. Every established link layer connection between two devices and each advertising message has an individual access address. This enables the classification, which data packet belongs to which connection.
- The third part of the BLE packet in figure 3.4 is the protocol data unit header (PDU header) with a length of 2 bytes. The header is used to define the function of the package. The basic package types are:

  - ADV IND: This package indicates a common advertising message. It is used for connectable and scannable devices, which send undirected advertising events. Those devices want to establish a connection to a host to reveal their data.
  - CONNECT IND: This header is sent by the Link Layer of a device, initiating a connection. The payload of this package contains information about the access address of the initiating device, the transmit widow size (the time window how long a connection stays on at one channel), the channel map (which of the 37 data channels are used), and additional information for a successful connection setup. The Link Layer of the advertising device will receive this message and create a connection to the remote device.
  - LL Data PDU: This package types contain a L2CAP-message. Those messages are forwarded to the L2CAP-layer, which is above the link layer, as figure 3.1 shows. This layer will be handled in the following chapter 3.3.
  - LL CONNECTION UPDATE IND: This packet type is used to update the settings of a connection. A package with this header contains information about the transmit window size, the timeout and additional setup information.
  - LL TERMINATE IND: This package type has a field for an ErrorCode, which shall be used to inform the remote device why the connection is about to be terminated. This could be one of 66 possible error codes. Some of the most significant codes are 'Authentication Failure (0x03)' and 'Insufficient security (0x2F)'.

- The payload: This field heavily depends on the PDU header. Depending on the header the size of the payload differs between 0 bytes and 255 bytes. Also the target of the contained information depends on the header. Possible targets are the L2CAP-layer, the physical layer, the security manager or the link layer itself.
- The message integrity check (MIC) is optional. The security manager can activate this field. As an example it is used in an encrypted connection. The message integrity check has a size of 4 bytes.
- The sixth part of a BLE packet is the cyclic redundancy check (CRC). This field is used to validate the previous received bits of the message. The CRC has a length of 3 bytes.
- The last part is the constant tone extension. This optional field was added with core specification 5.1 [BluetoothSIG 19]. With this feature it is possible to determine the angle of arrival of a package and thereby estimate the position of the device. It can be used in a IOT scenario, where Bluetooth Low Energy beacons are distributed in a room to triangulate the position of a device. A sender can use this field to transmit a sine-wave that holds between 16 μs and 160 μs. A receiver needs multiple antennas to use this feature. Every antenna would

receive the signal with small offset, that depends on the distance between the antennas and the angle of the arriving sine-signal. With this offset it is possible to calculate the direction of the signal source.

3.3 Logical Link Control and Adaptation Layer Protocol (L2CAP)

The L2CAP-layer is on top of the link layer, which is described in the previous chapter. The primary tasks of the L2CAP-layer are protocol multiplexing, the segmentation and reassembly of data packets for the upper layers.

The protocol multiplexing is necessary to enable the usage of multiple communication protocols. The most common protocols are the security-manager-protocol and the attribute-protocol, which are described in the following chapters 3.4 and 3.7. But it is also possible to design a custom protocol, which uses the L2CAP layer as access to the BLE architecture. The L2CAP layer provides an individual communication channel to every protocol above. Those channels are distinguished by channel identifiers (CID). The attribute protocol uses channel 0x0004, while the Security Manager protocol uses channel 0x0006. Custom protocols can use the channels 0x0040 to 0x007F.

The segmentation and reassembly of data is the second primary task of this layer. This operations are used if a data-set should be transmitted, which is larger than the current packet-size. The default payload size for a BLE data packet is 23 byte. If a larger data set should be transmitted, the L2CAP layer can split the large data set in multiple smaller data sets. Each of the smaller data sets will be transmitted in a single BLE data packet. Every data packet will be constructed according to the explanation in chapter 3.2. The L2CAP layer of the target device will reassemble the segmented data set and forward it to the upper layers. This segmentation mechanism enables transmission of data packets up to 64 KByte.

With Bluetooth Core Specification 4.2[BluetoothSIG 14] the maximum transmission unit (MTU) can be increased up to 251 byte, depending on the memory capabilities of the communicating devices. This enables more efficient transmission of larger data packets without the overhead of additional link-layer headers.

3.4 Attribute protocol

The Attribute protocol (ATT) is strongly connected to the Generic Attribute Profile (GATT), which is the layer above. While the attribute protocol provides basic data structures and operations to access the structures, the GATT defines how the attribute protocol is applied and how the data has to be interpreted.

The core element of the attribute protocol are the attributes. Each attribute consists of four elements:

- a 16 bit handle: The handle is a device internal unique identifier to identify a specific attribute.
- a type-UUID: This Universally Unique Identifier (UUID) defines which type the attribute has. It is important for upper layers to know how to interpret the data of the attribute. Since it is possible to have multiple attributes of the same type on one device, the handle has to ensure the identifiability. The type-UUID is not the same UUID that is used to identify a service. Some important type-UUID are:
  - 0x2800: this attribute is a primary service
  - 0x2801: this attribute is a secondary service
  - 0x2802: this attribute is a included service
  - 0x2803: this attribute is a characteristic
  - 0x2902: this attribute is a Client Characteristic Configuration Descriptor
  - 0x2A19: this attribute is a value of the type Battery Level

  The meaning of these types is described in the following GATT chapter 3.5. In chapter 3.5.1 an example data structure will be explained. This may help to understand the difference between the handle, the type-UUID and the UUID.
- a value: This field stores the information. The value is basically an array of bytes of an individual size. The attribute protocol does not determine the size of the value based on the type-UUID. The size management is done by the upper layer protocols, because the size often depends on internal configuration, stored inside the value. But the ATT layer does not interpret the value.
- security permissions: Each attribute has some permissions, that define the possible access to the attribute. The attribute protocol does not set the permissions, they have to be set from upper layer protocols. The attribute protocol has a fixed set of permissions:
  - Access permissions define how the attribute can be accessed:
    - None / readable / writeable / readable and writeable
  - Encryption permissions define if an encrypted connection is necessary to access this attribute:
    - Encryption required / no encryption required
  - Authentication permissions define if the client has to communicate via an authenticated link to access this attribute:

    - Authentication required / no authentication required
  - Authorization permissions introduce the concept of trusted devices. A trusted device has to be authenticated, but not all authenticated devices may be trusted devices. If a unauthorized but authenticated client wants to gain authorization to access an attribute, the user has to confirm the operation. This authorized the client for this one attribute:
    - Authorization required / no authorization required

The attribute protocol has two roles: an ATT server and an ATT client. The server stores the attributes. The client uses the attribute protocol to perform read and write operations at the attributes of the server. Most of the time the server has the passive role, while the client is the active part.

Only at notification and indication operations the server has the active role. Those operations give the responsibility to the server, to notify the client when the value of an attribute has changed. In case of an indication the client sends an acknowledgement when a value update is received, while a notification needs no acknowledgement.

The attribute profile has some improvements to save energy during the communication. The attribute profile never sends some length informations aside with the values. The receiver has to know the individual size and layout of the values, based on the UUID. For attributes with large values, the attribute profile offers long-read and long-write operations in cooperation with the L2CAP layer. These operations split the large attributes in chunks so they match the ATT-MTU, as mentioned in chapter 3.3.

3.5 Generic Attribute Profile

The Generic Attribute Profile (GATT) is the highest host layer in the BLE architecture. The GATT describes how attributes from the attribute protocol in chapter 3.4 are combined to a profile. Every element of a profile has its own UUID and is stored as an attribute.

The profile is the top level of the data structure hierarchy. A profile is the composition of all services a device implements. So a profile may have multiple services, as figure 3.5 illustrates. A service is a collection of data to implement a function or a feature of the device. So a device with multiple functions may have a BLE profile with multiple services, one service for each function of the device.

A service that implements a main functionality of the device is called a primary service. Other services that are used to support the primary service are called secondary services. A BLE device may have multiple primary services. Every service is identified by an UUID. The Bluetooth SIG has predefined some 16 bit UUIDs for elected

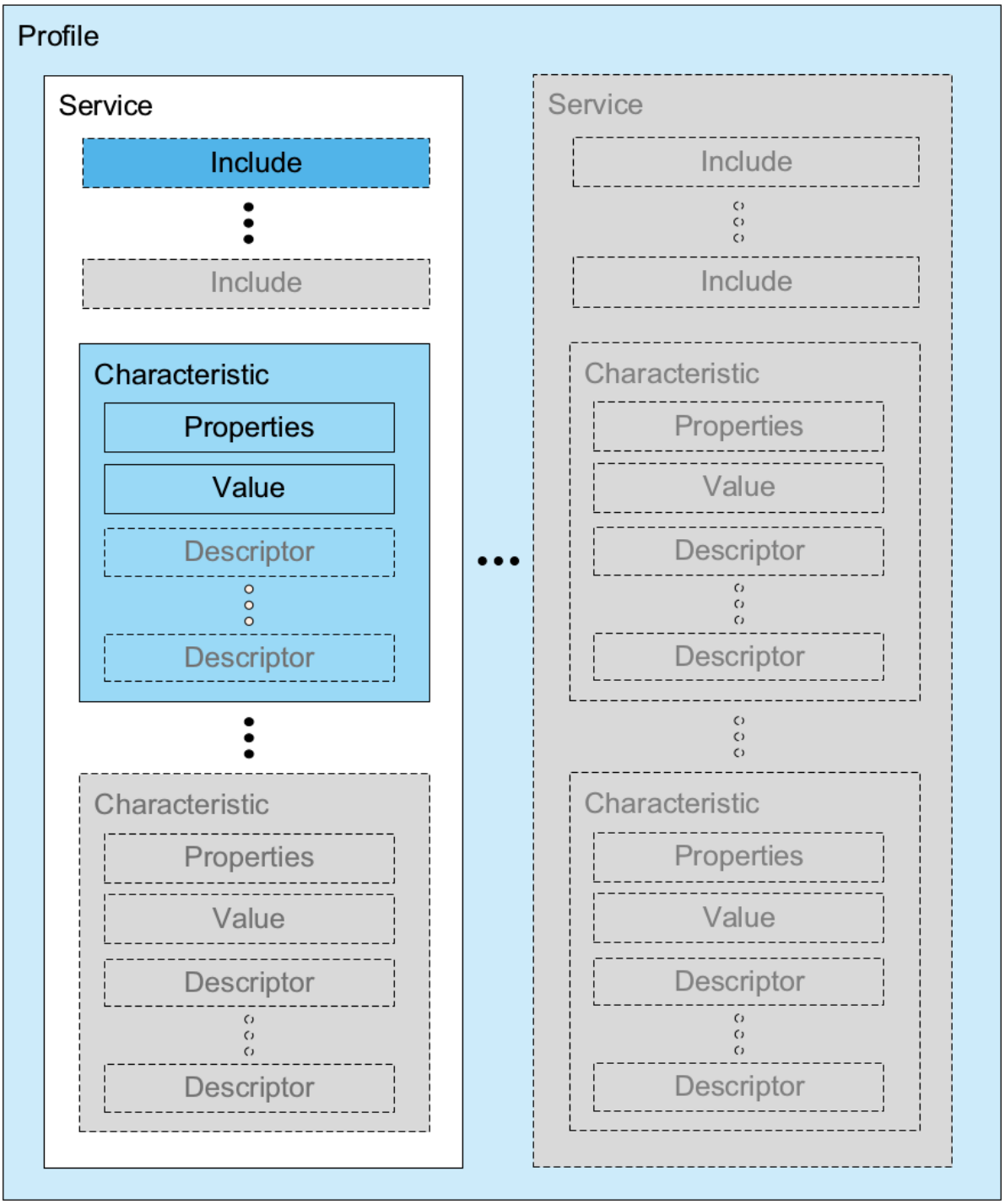


Figure 3.5: The structure of a BLE profile
Source: Bluetooth Core Specification 5.1 [BluetoothSIG 19]

services. The predefined services are associated with a particular feature, that they may implement. Custom designed services must have a 128 bit UUID. A service consists of some optional and some mandatory parts:

- Inclusions: It is possible that a service may include other services. This is optional. The included service will still be implemented as an independent service on the device. The inclusion of a service can be used to clarify that the functionality of the included service supports functionality of the including service. A primary services can include secondary services, but a primary service is never included by another service.
- Characteristics: A service has to implement at least one characteristic. A characteristic describes one part or property of the feature, the service implements. For the predefined services of the Bluetooth SIG, some corresponding characteristics are predefined. Every characteristic consists of multiple parts:
  - Properties: The properties describe how the characteristic can be accessed. There are seven primary properties that can be set:
    1. Broadcast (0x01)
    2. Read (0x02)
    3. Write without response (0x04)
    4. Write (with response) (0x08)
    5. Notify (0x10)
    6. Indicate (0x20)
    7. Authenticated Signed Writes (0x80)

    Some properties require a special descriptor. The broadcast property requires a Server Characteristic Configuration Descriptor, while notify and indicate require a Client Characteristic Configuration Descriptor. Those special descriptors are used to activate or deactivate the functionality, comparable to a subscribe/unsubscribe.
  - A value: The value is the raw information that should be communicated. The maximal length of the value is 512 byte.
  - Descriptors: A characteristic may contain one or more descriptors, which describe the value or enable some configurations for this characteristic.

  It is possible to set individual permissions to access a characteristic. This enables individual security levels, which are further described in the security manager in chapter 3.7.

3.5.1 Example Data Structure

This chapter shows an example that illustrates the interaction of the ATT layer and the GATT layer to build up a data structure. Table 3.1 shows in which structure the data of a battery service and a health thermometer service are stored. Both services are predefined by the Bluetooth SIG. The definitions of the services can be found online[2].

Handle	type-UUID	type-description	value	value-description	raw data
0x0100	0x2800	Primary Service	0x180F	Battery Service UUID	00 01 00 28 0F 18
0x0101	0x2803	Characteristic	0x12, 0x0102, 0x2A19	read+notify, value-handle, Battery Level UUID	01 01 03 28 12 02 01 19 2A
0x0102	0x2A19	Battery Level Value	0x0028	value 40%	02 01 19 2A 28 00
0x0103	0x2902	Client Characteristic Configuration Descriptor	0x0000	notification disabled	03 01 02 29 00 00 00 00
0x0150	0x2800	Primary Service	0x180D	Heart Rate Service UUID	50 01 00 28 0D 18
0x0151	0x2803	Characteristic	0x10, 0x0152, 0x2A37	notify, value-handle, Heart Rate Measurement UUID	51 01 03 28 10 52 01 37 2A
0x0152	0x2A37	Heart Rate Measurement Value	0x00, 0x4B	Format UINT8, value 75 bpm	52 01 37 2A 00 4B
0x0153	0x2902	Descriptor Client Characteristic Configuration	0x0001	notification enabled	03 01 02 29 00 00 01 00

Table 3.1: An example how two services are encoded, based on the definitions of the ATT and the GATT.

Every row in Table 3.1 is a single attribute of the ATT layer. As mentioned in chapter 3.4, every attribute consists of a handle, a type-UUID and a value. The security permissions are skipped in favor of readability.

The first column shows the handle, given by the ATT layer to identify each attribute. The second column holds the type-UUIDs. They are used by the GATT layer to interpret the value. The type-description column is introduced to clarify the meaning of the type-UUID. The fourth colum holds the value, while the fifth column describes the interpretation of the data in the value field. The last column contains the raw data of the attributes in hex format. This is the ordering how the data are stored and also

[2]Bluetooth SIG predefined services

send. The data are in little endian ordering, so the hex value 0x1234 is stored in order 0x34 12.

The first line with the handle 0x0100 is a primary service, which is indicated by the type-UUID 0x2800. Some type-UUIDs were introduced in chapter 3.4. The UUID that indicates which service it is, is stored in the value field. In this example the UUID 0x180F indicates a Battery Service. All following lines belong to this service, until the next attribute with a primary or secondary service type-UUID appears.

Line 0x0101 holds a characteristic with the corresponding type-UUID. The value of a characteristic always stores in the first byte the properties how to access the characteristic. According to the property enumeration in chapter 3.5 the read-property has the value 0x02 and the notify-property has the value 0x10. These values are combined by an or-operation, so the result is the value 0x12. In this example the characteristic can be accessed via a read or a notify. The next two byte of the value indicate the handle with the value of the characteristic and the UUID describing the type of the characteristics value. In this example the value of the characteristic is stored in handle 0x0102 and the type of the value is the Battery Level.

Attribute 0x0102 is a battery level value type. This attribute holds the information of the characteristic. The information how to interpret the value of this attributes depends on the type-UUID of the attribute. The type-UUID holds the information for upper layers how construe the information of this attribute. In this case, the information is encoded as an unsigned 8 bit integer. The values from 0 to 100 are used to indicate the battery level in percent, values from 101 to 255 are invalid.

The following attribute 0x0103 has the type-UUID 0x2902, which indicates that this attribute is a client characteristic configuration descriptor. This descriptor is used to enable the notify property of the characteristic. The value 0x0000 shows that the notification is turned off.

The next service is in attribute 0x0150. The UUID, identifying it as a heart rate service is stored in the value of this attribute. Again all following attributes belong to this service, until the next service attribute is reached.

The characteristic in attribute 0x0151 can only be accessed via notify as the first byte of the value indicates. The next bytes are the handle of the attribute holding the value and the type of the attribute.

Attribute 0x0152 is the heart rate measurement value that belongs to the characteristic attribute above. The values encoding is a bit more complex. The first byte indicates that the following measurement is encoded as 8 bit unsigned integer. Flipping the last bit would indicate that the measurement is encoded as 16 bit unsigned integer. Additional bit flips could be used to enable features as a field with the expended energy in kilo joule. The exact definition can be read at the Bluetooth website[3]. All the additional features are turned off for this example. So the following byte is the

[3]Bluetooth Characteristic Heart Rate Measurement Specification

unsigned integer which holds the information of the current measurement, which is a heart rate of 75 beats per minute in this example.

The last attribute 0x0153 is a client characteristic configuration descriptor that is used to access the characteristic via a notify. In this example the notification is turned on.

3.6 Generic Access Profile

The Bluetooth Low Energy Generic Access Profile (GAP) defines the different roles a BLE device can have. Each role defines the basic behavior of a device. The GAP influences all other layers in the host, so it is drawn across all other layers in figure 3.1. There are four basic roles:

1. Broadcaster: Together with an observer, a broadcaster implements a connectionless unidirectional communication. A broadcaster periodically sends advertising messages with data. It is not possible to establish a connection to a broadcaster.
2. Observer: An observer scans for broadcasters and listens for advertising messages containing data.
3. Peripheral: Together with a central, a peripheral implements a connection oriented bidirectional communication. A peripheral sends advertising messages with all necessary information for a connection establishment. It waits for a central to establish a connection, so the peripheral is in a slave role. A peripheral can handle connections of multiple centrals if the device implements Bluetooth specification 4.1. Peripherals are optimized to consume at least processing power and memory as possible.
4. Central: A central is in a master role. It scans for advertising messages of peripherals, to extract their communication parameters. With this knowledge a central can start a connection establishment. A central can be connected to multiple peripherals at a time.

Devices can implement multiple roles at the same time. In figure 3.6 five different BLE networks are shown.

In the first network device A is a master of the two devices B and C. So device A implements a central while the devices B and C are implementing the roles of peripherals. Meanwhile device D is advertising (represented by the star) and device A is establishing a connection with this device, to become the master of a third device. At the same time device C is advertising, which is recognized by device E.

The devices G and F create their own network, with F as the central and G as the peripheral.

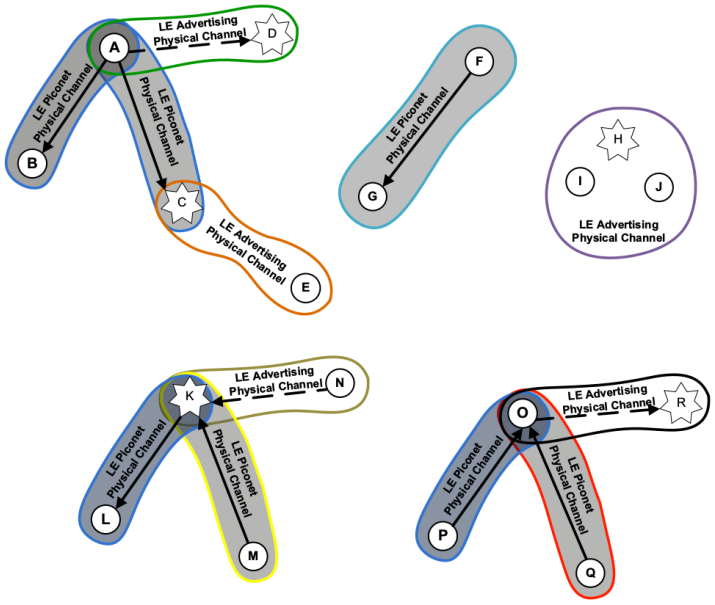


Figure 3.6: Different Bluetooth Low Energy network topologies
Source: Bluetooth Core Specification 5.1[BluetoothSIG 19]

The two devices I and J are recognizing the advertising of device H. But they are not attempting to connect to the device. So device H is a broadcaster, while the devices I and J are observers.

Device L is the slave of device K, and device M is the master of device K. So K implements the roles of a central and a peripheral at the same time. Device K is advertising, and device N initiates a connection with K to become another master of K.

In the last composition devices P and Q are the masters of device O, which is the slave of both devices. Device R is currently advertising and device O establishes a connection with R. So R would become the slave of O, and device O would become the master of R. In this network device O implements the roles of a central and peripheral at the same time.

This illustrates some of the possible network topologies in Bluetooth Low Energy. However BLE does not natively support mesh networking. The informations are only communicated from a peripheral to the master or vice versa. There are no additional network layers to enable advanced routing or package forwarding to support communication between devices that are not directly connected. Accordingly there is no possible direct communication between the devices P and Q in the last example of figure 3.6.

In 2017 the Bluetooth SIG introduced a specification for Bluetooth mesh devices. 2019 the specification of a mesh profile followed. So the Bluetooth SIG reacted on the missing mesh networking and provided a possibility to enable the mesh networks.

Furthermore the GAP defines security modes and operations to ensure certain security levels. These modes and operations are implemented in the Security Manager, which will be described in the following chapter 3.7.

3.7 Security Manager

The Security Manager implements the security modes and operations, that are required by the GAP and the ATT protocol. As figure 3.1 illustrates, the Security Manager uses some functions of the L2CAP layer. It has direct access to the link layer to handle encryption keys and manage authentication procedures. In the following, the primary security architecture and the procedures for paring and bonding are described.

The rules of the GAP define two security modes, where each mode has multiple security levels:

- LE Security mode 1. This security mode has four security levels:
    1. No security (no encryption or authentication)
    2. Unauthenticated pairing with encryption
    3. Authenticated pairing with encryption
    4. Authenticated LE Secure Connections pairing with a 128-bit encryption key
- LE Security mode 2. This security mode is used for data exchange with signed data. It has two security levels:
    1. Unauthenticated pairing with data signing
    2. Authenticated pairing with data signing

When two devices establish a new connection they initially start at security mode 1 level 1, this means the connection offers no security. To improve the security of the connection the two devices must pair. The paring can be started by a central device by reading a characteristic on a peripheral that needs authenticated access. Alternatively, one of the devices could actively send a paring or bonding request. Paring causes the two devices to authenticate their identity before they negotiate a short term key (STK) to encrypt the communication.

Paring is a pre-stage of bonding. Bonding describes the exchange of a long term key (LTK) that will be stored on both devices. After the communication channel is encrypted with the STK, the LTK will be exchanged. The LTK enables a simpler security setup when reconnecting the devices in the future, so both devices will be able to create encrypted communication links without passing through the paring procedure again.

3.7.1 The paring methods

BLE offers different possibilities to pair two devices. The paring modes differ in the way the devices authenticate their identity and how the keys are generated. The Bluetooth SIG defined some paring modes in core specification 4.0 that are shown to be insecure, chapter 4.3 explains the problems in more detail. These paring modes are summarized under the term 'legacy paring'. As a reaction, they released the core specification 4.2 with new paring modes for the Bluetooth Low Energy architecture. These paring modes are considered to be secure. They are summarized under the term 'secure connections'.

There are three keys that are exchanged during the paring and bonding procedure:

- a temporary key (TK): the TK is a 128 bit AES key whose value depends on the selected pairing mode.
- a short term key (STK): the messages to exchange the STK are encrypted with the TK. The STK is used to establish the link-layer encryption.
- and a long term key (LTK): the LTK is stored on both devices to encrypt connections in the future. The LTK is exchanged via an link that is encrypted with the STK.

The three 'legacy paring' modes are [BluetoothSIG 10]:

- Just Works: This paring mode does not check the identity of the paring devices. It assumes that the devices are correct and automatically sets the TK to the static value 0 (zero).
- Passkey entry: This paring mode requires that a 6-digit pin can be transfered from one device to another. Typically this is done by using a display to show the six digits on one device and a keyboard to type in these digits in the other device. Both devices use this 6-digit pin as a TK.
- Out of Band (OOB): This paring mode uses another communication channel outside the common BLE frequencies which is supposed to be secure. The secure channel is used to distribute a random 128-bit value that is used as the TK. As an example NFC can be used as a secure out of band channel.

In core specification 4.2 the problem of the insecure key exchange was addressed by introducing a 'secure connections only mode' to BLE. This mode allows to use the four paring methods of the standard BR/EDR Bluetooth to establish more secure connections. These four paring methods are using an Elliptic Curve Diffie Hellman (ECDH) key exchange algorithm with a key size of 256 bit [BluetoothSIG 14]. The Diffie Hellman algorithm is a well known and approved key exchange. The Elliptic Curve Diffie Hellman is an enhanced version of the standard DH, which is more efficient and harder to break [Schmitt 16]. Old devices which implement only the BLE specification 4.0 or 4.1 are not compatible with the 'secure connections only mode' in

new devices due to the new hardware that is necessary to implement the ECDH. The four paring methods in 'secure connections' mode are [BluetoothSIG 14]:

- Just Works: In this paring mode does not check the identity of both devices. The devices exchange the key by using the ECDH algorithm. To verify that both devices have the same key, the peripheral generates a confirmation value C_a based on a random value and his key. The random value and C_a are send to the central device. The central device uses the received random value and his own key to generates a second confirmation value C_b which should be equal to C_a. Simultaneously the central generates his own random value, and a confirmation value and sends both to the peripheral, so the peripheral can validate the correctness of the keys, too.
- Numeric Comparison: This paring mode uses the same procedure as the Just Works but adds an additional step at the end to authenticate both devices: when both devices agree that their keys match, they independently generate a 6 digit value using the two random values. Both devices show the values to the user, which has to manually compare the values and admits the connection.
- Passkey entry: this paring mode exchanges the key via ECDH, as the previous methods. Additionally a 6 digit number must be known on both devices, this is the passkey. Usually one device displays the digits to the user, which types it into the second device. The devices use the passkey, a generated random value and the shared key to authenticate the connection. The initiator of the pairing generates a confirmation value based on the random value, one bit of the passkey and the shared key and sends it to the peripheral. The same is done by the peripheral. In a second step the central sends the random value to the peripheral, so this one can validate the first bit of the passkey. If it is accepted, the peripheral sends its random value to the central, so this one can validate the first bit. This procedure is done bitwise for all bits of the passkey.
- Out Of Band: In this paring mode another communication band outside the BLE frequencies is used to exchange all security informations. As in BLE legacy paring the security of this paring mode depends on the security of the used communication channel.

Chapter 4.3 looks at the weaknesses of connections and takes a deeper look at the strength of the different paring methods. This chapter continues by determining in which situation a paring method is used. This choice, depends primary on the input and output capabilities of the pairing devices.

The paring modes, shown in table 3.2, demonstrate that in 13 of 25 cases an authenticated connection is possible. The only setups that make it impossible to create a authenticated connection use devices with no input-output capability.

Using only the legacy paring methods, would change the table a bit:

- The 'Numeric Comparison[1]' would be replaced by a unauthenticated Just Works paring.

Device B (responder)	Device A (initiator)				
	No input, no output	Display only	Display Yes/No	Display with keyboard	Keyboard only
No input, no output	Just works (unauthenticated)	Just works (unauthenticated)	Just works (unauthenticated)	Just works (unauthenticated)	Just works (unauthenticated)
Display only	Just works (unauthenticated)	Just works (unauthenticated)	Just works (unauthenticated)	Passkey Entry (authenticated)	Passkey Entry (authenticated)
Display Yes/No	Just works (unauthenticated)	Just works (unauthenticated)	Numeric Comparison[1] (authenticated)	Numeric Comparison[2] (authenticated)	Passkey entry (authenticated)
Display with keyboard	Just works (unauthenticated)	Passkey entry (authenticated)	Numeric Comparison[2] (authenticated)	Numeric Comparison[2] (authenticated)	Passkey entry (authenticated)
Keyboard only	Just works (unauthenticated)	Passkey entry (authenticated)	Passkey entry (authenticated)	Passkey entry (authenticated)	Passkey entry (authenticated)

Table 3.2: Possible secure connections paring methods, depending on the input and output capabilities of the devices
Source: Bluetooth Core Specification 5.1[BluetoothSIG 19]

- The 'Numeric Comparison[2]' would be replaced by a authenticated Passkey Entry.

If it is possible to select between the Numeric Comparison or the Passkey Entry the Bluetooth SIG prefers the Numeric Comparison since it requires a simpler user interaction.

3.7.2 The privacy feature

Since Bluetooth Core Specification 4.1 BLE offers a privacy feature. It allows bonded devices to change their revealed MAC address frequently [BluetoothSIG 13]. This feature was introduced to prohibit the simple tracking of BLE devices. A peripheral which supports this feature exchanges during the bonding procedure an identity resolving key (IRK) with the central device. This IRK can be used to generate new MAC addresses. In the future the advertising messages of the peripheral will use those new generated MAC addresses instead of its own address. The bonded central is able to map the generated MAC addresses back to the real address of the peripheral.

3.7.3 The encryption

BLE uses the AES-CCM algorithm to encrypt data [BluetoothSIG 19].

In CCM mode AES combines a counter mode for the encryption and a CBC-MAC for integrity check. In this way, the encryption and the authentication check is done with one key.

Each connection has a individual initialization value, which is generated during the pairing. The central and the peripheral both randomly generate a half of the value and share it with the other device.

The CBC-MAC mode uses the cypher block chaining (CBC) to generate a message authentication code (MAC). With this MAC the authentication of the message sender can be verified and the integrity of the transmitted data can be checked. To avoid confusion in the BLE specification, the message authentication code is called a message integrity check (MIC), because the abbreviation MAC is already used for another component.

To generate a MIC the plain text, the encryption key and the initialization vector are used. A CCM nonce is uses as initialization vector. The CCM nonce contains the shared initialization value and a 39 bit packet counter. The counter starts with zero for a new connection and is incremented for each encrypted package data unit (PDU). The nonce is individual for each data packet that is encrypted, so equal data packets will be encrypted to different cypher texts.

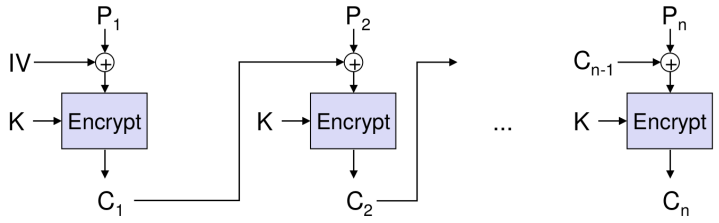


Figure 3.7: Schematic of the AES-CBC mode
Source: Lecture protocols and algorithms of network security[Schmitt 16]

In figure 3.7 the CBC mode of AES is shown. Every generated cypher text block (C_i) is combined with the next plain text block (P_{i+1}) via a x-or. This modified plain text block is encrypted with the key (K) to generate the next cypher text block (C_{i+1}). In this way, the last cypher text block (C_n) depends on all plain text blocks and on the CCM nonce. This last cypher text block is the MIC. It is added to the plain text to authenticate the sender and to prevent replay attacks.

The counter mode is used to convert a block cypher, as AES, into a stream cypher. Block cyphers always encrypts data blocks of a specific size and so the output is generated block wise. This may blow up the data that shall be transmitted. Stream cyphers are able to encrypt data packages of every size without increasing the length of the encrypted data. To avoid increasing the size of transmitted data packets BLE uses a stream cypher to encrypt the transmitted data.

The counter mode generates cypher blocks that are combined with the plain text via x-or operations. In this way the plain text and the appended MIC are encrypted. The cypher blocks are generated by using the CCM nonce and the encryption key.

3.8 Application Layer

The application layer is above all other introduced layers. It represents the application that uses Bluetooth Low Energy for communication and runs on a smartphones or in embedded hardware.

On a peripheral device this layer uses the GATT and ATT layer to implement the services and update the values in this data structure at runtime.

A central device uses the underlying layers to find the peripheral and connect to the device. Received data from the peripheral is passed through the L2CAP, in the ATT and the GATT layer to offer a usable data structure.

4 Possible attacks against Bluetooth Low Energy

This chapter will introduce possible attacks against BLE communication. Each section will handle a specific class of attacks. Therefore a brief introduction explains how the attacks influence the communication. Based on this the possible counteractive measures are described.

Some classical attacks as cross side scripting or ARP poisoning are not adaptable to the BLE communication stack, so these assaults will not be explained.

The list starts at the physical layer, where the raw signals are transmitted. In this layer jamming is a common attack. The description and countermeasures are described in chapter 4.1. It continues on the link layer with gathering informations about potential connection devices and a possible control mechanism in chapter 4.2. Chapter 4.3 goes a step further and assumes that two devices start communicating with each other to discuss how to eavesdrop on them and how to prevent eavesdropping. At the same time chapter 4.4 becomes relevant, when the communicating devices authenticate to each other, therefore spoofing is a serious issue. After discussing protections against spoofing, the following chapter 4.5 continues with the possibility of packet injection in a established connection. After packet injection the class of denial of service (DOS) attacks is introduced in chapter 4.6. After all this active attacks, chapter 4.7 covers the passive attack of traffic analysis. Chapter 4.8 arrives at the application layer and deals with function level access without permission. In the end chapter 4.9 labels some possible attacks against BLE devices, which are performed outside the BLE communication.

4.1 Jamming

Jamming describes the disturbing of a communication by sending interfering signals into the communication channel [Schmitt 15][Healy 09][Barcena 15]. It is possible to lower the throughput of a network or to archive a full denial of service. In case of wireless communication the communication channel is a electro magnetic frequency band in a specific frequency range. Bluetooth low Energy uses the frequency range from 2.402 GHz to 2.480 GHz [BluetoothSIG 10]. Since everyone can access this frequencies they are a public available shared medium. Such a public available shared

medium is highly prone to disturbances. A potential attacker does not even have to be a member of the network to get access to the communication channel.

Due to the public availability there is no real algorithmic defense against an attacking jammer. It is possible to increase the distance between the jammer and the communicating partners, so that the attacker would need much more energy to send strong enough interfering signals. Another mitigation is to change the communication channel. In the case of electro magnetic waves this can be realized by changing the used communication frequency. This technique is called channel switching or channel hopping. [Schmitt 15]

Bluetooth Low Energy uses automatic channel hopping to avoid constantly interfering with other signals. The used frequency ranges from 2402 MHz to 2480 MHz is split into 40 channels. As already mentioned in chapter 3.1. 37 of this 40 channels are used for data transfer, 3 channels are used for setting up new connections. An already established connection jumps across the 37 data channels after a predefined interval. This interval ranges from 7.5 ms until 4 s with increasing 1.25 ms steps. The individual interval is determined during the connection establishment. [BluetoothSIG 10]

Based on the channel selection algorithm Ryan et al [Ryan 13] designed an attack to find out the channel hopping sequence and follow the used communication channels. Therefore Ryan listened to a single data channel and measured the time between a usage of this channel and the next usage of this channel. The algorithm uses all available communication channels, so a complete cycle over all 37 channels is performed until a channel is used again. The time for a complete cycle is given by $37 * 1.25ms * hopInterval$. By measuring the cycle time, the hop interval can be calculated with equation 4.1.

With the known hop interval it is possible to find out how many channels are used in a specific time by equation 4.2. The next communication channel is calculated by $nextChannel = currentChannel + hopIncrement(mod37)$. The hop increment of a connection does not chance, so equation 4.3 must hold. This can be transformed to equation 4.4 to get the hop increment. Since the hopping sequence is isomorphic to the field of $\mathbb{Z}_{37}$, the transformation 4.5 with Fermat's little theorem can be applied. So it is possible to calculate the hop increment by formula 4.6. Based on this the next used communication channel can be determined. [Ryan 13]

$$hopInterval = \frac{\Delta t}{37 * 1.25ms} \tag{4.1}$$

$$channelsHopped = \frac{\Delta t}{1.25ms * hopInterval} \tag{4.2}$$

$$0 + hopIncrement * channelsHopped = 1(mod37) \tag{4.3}$$

$$hopIncrement = channelsHopped^{-1}(mod37) \tag{4.4}$$

$$channelsHopped^{-1} = channelsHopped^{37-2}(mod37) \tag{4.5}$$

$$hopIncrement = channelsHopped^{35}(mod37) \tag{4.6}$$

With these calculations it is possible to recover the sequence of used channels and the time spend on each channel. This makes it easy to follow the communication and jam the signals.

In Bluetooth specification 5.0 a second channel selection algorithm was introduced, to pseudo random select the next channels [BluetoothSIG 16]. This makes it harder for an attacker to follow the used communication channels.

Nevertheless there exists another possibility to determine the used channels: during the connection establishment at the beginning, some initial messages are exchanged. These messages contain informations about the channel selection algorithm. If those messages are unencrypted and an attacker can listen to them he has all informations to follow the connection.

Still there exist some strategies to jam signals that use multiple channels: Sampath et al [Sampath 07] used a cognitive radio to jam multiple 802.11 network (WLAN) channels. This approach could be used to interrupt the advertising channels.

Beside the planned attacks there are multiple interference sources that send electro magnetic signals as a side effect of their normal function. Transformators or microwave ovens are examples for such sources.

Barcena et al [Barcena 15] proposes the implementation of a fall-back mechanism that ensures the correctness of the system in case of a disturbed communication. To meet the requirements of the functional safety this has to be considered during implementing a secure communication. Chapter 5 deals with the necessary security requirements.

4.2 Access Control

To accomplish full access control in wireless networks is nearly impossible, since the used communication medium is available for everyone. Still there are some techniques to complicate the uncontrolled access to a network. In WLANs it is possible to use hidden SSIDs (Service Set Identifiers). This ensures that the network is not listed while searching for networks in the common way [Schmitt 15].

BLE has a similar approach where the advertiser sends no advertising messages and keeps undetected for a scanning device. The Out Of Band (OOB) association model is designed for scenarios where an out of band communication is used to discover a device and to exchange informations for a connection setup. Depending on the security of the OOB communication channel the created BLE connection is protected against eavesdropping and MITM attacks [BluetoothSIG 10][Barcena 15].

4.3 Eavesdropping

Eavesdropping describes the listening to a communication without the consent of the communicating entities. In wireless networks this is possible due to the uncontrollable spreading of the signal waves. The attacker does not have to be a member of the network to passively listen to the communication. To maintain confidentiality eavesdropping must be prevent. This can be archived with a strong encryption [Schmitt 15][Healy 09].

BLE uses AES with a 128 bit key to ensure the confidentiality and integrity of the messages [BluetoothSIG 10]. AES is a symmetric encryption algorithm, so sender and receiver must have the same keys for encryption and decryption. A key-exchange algorithm is needed to ensure that only the communicating partners have the required key and no possible eavesdropper can get the key.

With the introduction of BLE in the Bluetooth Core Specification 4.0 a 'legacy paring' key exchange algorithm was introduced. The algorithm uses specific values depending on the used paring procedure. The paring of two devices and key exchange for connection establishment belong together. The 'legacy paring' method is still the most widely used [Sivakumaran 18], despite its weaknesses. Ryan et al [Ryan 13] described how to recover the 128-bit AES key from a eavesdropped connection setup.

To fully understand the problematic it is important to remember the different keys and paring modes BLE uses for paring as described in chapter 3.7.1. The three keys are a temporary key (TK), a short term key (STK) and a long term key (LTK).

The 'legacy paring' modes are [BluetoothSIG 10] Just Works, Passkey entry and Out of Band (OOB). In case of a Just Works paring the TK is already known, it is the static value 0. The passkey entry paring uses the exchanged 6-digit value as the TK. So the TK will be a value between 0 and 999.999, which leads to approximately 20 bits of entropy, which is still very weak. This key can be cracked in less than one second on a single core of a common Intel Core i7 CPU as Ryan et al [Ryan 13] wrote. So these two paring modes provide no protection against an eavesdropper that is listening from the beginning [Padgette 12]. Only the OOB paring archives a acceptable security by shifting the security problem to another technology. NFC is a suitable candidate, due to its small communication range of just a few centimeter, which makes it hard for an eavesdropper to remain undiscovered.

The Bluetooth core specification 4.0 holds a note that those paring methods are only safe if an attacker starts listening after the STK was exchanged [BluetoothSIG 10]. So the Bluetooth SIG was has known the insecurities of the paring modes from the beginning.

Four years later in core specification 4.2 the problem of the insecure key exchange was addressed by introducing a 'secure connections only mode' to BLE, with four possible paring methods: Just Works, Numeric Comparison, Passkey entry and Out Of Band.

Assuming that the selected communication channel for OOB-paring is secure, all of the secure connections only paring modes archive a good protection against eavesdropping. All of the paring methods that are working in BLE frequencies are using ECDH key exchanges with 256 bit keys, which is an adequate protection for a 128 bit AES key [Schmitt 16]. So the communication is protected against eavesdropping from the beginning.

4.4 Spoofing

Spoofing describes a man-in-the-middle (MITM) attack, where a communication is redirected over an attacking participant. It takes place if an entity A wants to communicate with a second entity B. An attacking entity C pretends to be B, so A starts communicating with him. C wants to stay undetected to listen to the communication. So he can either fake the behavior of B and continue communicating or C could forward the messages to B pretending to be A. In the second case C would be able to read the communication between A and B even though they start using link layer encryption because both are unintended communicating with C [Schmitt 15].

In case of BLE this attack has to take place during the paring, before the communicating entities exchanged their encryption keys. The different paring modes are more or less vulnerable to man in the middle attacks. In legacy paring [BluetoothSIG 10]:

- Just Works: in this paring mode only the TK is determined. No authentication of the paring devices is performed, so this is very vulnerable to spoofing.
- Passkey Entry: this paring mode uses a 6-digit pin as a TK. The pin is usually displayed on one device, so that the user can type it into the second device. The 6 digit pin allows numbers from 0 to 999.999. Assuming that both the displaying device and the input device are not corrupted by the attacker, the chance to guess the right pin for an MITM attacker is 1:1.000.000.
- Out Of Band: this paring mode uses a communication outside BLE to exchange informations. The vulnerability depends on the used communication technique. Additional procedures to ensure the identity of the devices are not available.

The situation in 'secure connections only mode' is similar:

- Just Works: this paring mode exchanges keys via ECDH and both devices validate their keys. In this procedure is no possibility to authenticate the devices to ensure the identity of the devices, so it is vulnerable to MITM attacks.
- Numeric Comparison: in this paring mode both devices independently generate 6-digit numbers, based on the keys. Those numbers are shown to the user. The user ensures that the numbers on both devices are equal. With this independent generation and user made comparison the chance for an attacker to

generate a fitting number is 1:1.000.000. So this paring mode inhibits the spoofing of the communication relatively good.

- Passkey Entry: this mode uses the bitwise comparing of a 6-digit number to authenticate both devices to have the same number. The distribution of the number via a encrypted channel ensures the correctness of the authentication. Typically one device displays the number and the user, which is used as the secure channel, enters this number into the second device. The chance of an attacker to guess the right number and stay undetected in a communication is 1:1.000.000 and which a relative good protection.
- Out Of Band: in this paring mode the MITM protection depends on the used out of band communication channel.

Nearly all paring modes offer some protection against MITM attacks. Only the two Just Works paring modes in 'BLE legacy paring' and in 'secure connections only' mode provide no possibility to authenticate the two devices [Padgette 12].

The Numeric Comparison and Passkey Entry modes provide a chance of 1:1.000.000 for an attacker to successfully spoof the communication. This probability is limited by the maximal digits of the values used to authenticate the devices. The number of digits were limited by six to minimize the usability impact [BluetoothSIG 10]. This is acceptable in this case of a e-bike control, since the hacking of an e-bike does not directly cause injuries or fatal financial impacts.

Still these paring modes need at least the possibility to display the validation numbers on the e-bike. This is impractical for an e-bike without an display. A good alternative would be the Out Of Band paring mode, where no additional displays are used. Still OOB paring is not widely seen in practice [Sivakumaran 18], which might be caused by the additional hardware for the additional communication channel.

4.5 Packet-injection

Packet injection describes the threat of an attacker inserting messages into a established communication, in a way the communicating entities do not realize that the inserted message is from an attacker [Barcena 15][Schmitt 15]. Different attacks can be summarized as injection attacks:

- deauthentication attack: an attacker inserts a message so that at least one of the communicating entities assumes the communication channel is closed. So further communication is prevented.
- replay attack: An attacker listens to communication and resends some of the messages recorded. The attacker does not even have to understand the messages to perform this attack, which would be the case in an encrypted communication channel. The goal of this attack is to bring the communicating entities in an undesired state, to potentially harm the system or the environment.

To prevent packet-injection Healy et al [Healy 09] recommends to use communication protocols which provide confidentiality, enforce authentication and ensure integrity. In BLE this can be accomplished by using an encrypted communication, in combination with an paring mode that ensures authentication [BluetoothSIG 10]. The encryption of communication links is realized with the AES-CCM algorithm. This algorithm provides protection against packet injection by using a packet counter, as chapter 3.7.3 describes. Additionally BLE provides a signing funcion for unencrypted communication links. The signing algorithm uses a signing key and a 32-bit counter to sing the data. The counter protects against simple replay attacks from unauthorized intruders.

4.6 Denial Of Service Attacks

A Denial Of Service (DOS) attack describes a class of attacks where malicious requests from an attacker, lead to the crash of the system [O'Sullivan 15] [Healy 09]. In such a scenario, an attacker exploits a bad implementation, overloads limited resources or abuses protocol flaws to bring the service provider in an undesired state.

Classic attacks like the Ping of death or the Teardrop use bad implementations of the TCP/IP stack to crash the machine. A Ping Flood or a Smurf Attack abuse the limited bandwidth of networks to send as many ping requests as possible to a victim. This overloads the bandwidth and reduces his successful normal traffic to a minimum [Schmitt 15]. These classical attacks are not applicable in a BLE network due to the missing network layers and the small number of participants in BLE networks.

Devices using BLE are normally limited in their available energy, so a new group of DOS attacks occurred: Battery Draining Denial of Service attacks [O'Sullivan 15] [Barcena 15].

Researchers from the university of Utah used repeating requests to reduce a phones battery life by 97 % [Premnath 08]. The experiment was performed in 2008 so the current specification was Bluetooth 2.0 and the result might not be completely applicable to current BLE standards. But it shows a new field of potential attacks. To mitigate such an attack, it was recommended to run the bluetooth devices in non-discoverable mode. This approach is comparable to the OOB-association model [BluetoothSIG 10], which was already mentioned during discussing possible access control to BLE connections in chapter 4.2.

4.7 Traffic Analysis

Traffic analysis describes a passive attack, where an attacker listens to communication and collects sensitive metadata. This is independent from the content of the messages

[Schmitt 15]. In larger networks, with more communicating entities traffic analysis is able to gather information about which entities communicate with each other, which route a message takes, how often and at witch time entities communicate. This information can be used to figure out the structure of a network or to rank entities in a network. This is useful to identify specific participants, so that attacks have the highest impact [Healy 09].

In case of direct BLE connections many informations like the structure or the identity of the communicating entities are trivial to figure out. Other metrics as the communication time and duration can reveal a lot about the usage of a device.

The best prevention of BLE against traffic analysis is the limited range of the signals. This enforces an attacker to be close to the communicating devices. To fully prevent the collection of metadata would require very high effort [Schmitt 15]. As an example it would require to send data and keep multiple connections all the time to hide the real using times of the bike. In an environment with limited energy, like an e-bike, this is impossible to handle.

A second possible scenario is tracking a BLE device by the used device address during a communication. This could be possible in a environment with many devices, listening for BLE traffic of nearby devices. By combining recorded device addresses with the locations of the recording devices it is possible to track communicating BLE devices and create a movement profile of their owners [Padgette 12]. This was possible for Fitbit devices, which were analyzed by Cyr et al [Cyr 14]. BLE offers a privacy feature which allows bonded devices to change their device addresses frequently, which makes it much harder to track the devices over a long time [BluetoothSIG 10].

4.8 Unauthenticated function-level access

Missing function-level access control allows unauthenticated entities to perform actions. Reconfiguration of devices, access to sensitive data or execution of critical functions could lead to harmful behavior of the system[Barcena 15].

The standard specification for e-bikes DIN-EN-15149 requires the protection against unauthorized access to the motor. Also manipulation on the control software has to be avoided. Chapter 5.1 lists the requirements in detail.

In BLE the data are capsuled in characteristics. These characteristics define the possible access properties to manipulate the data. The seven possible access types are Broadcast, Read, Write without response, Write with response, Notify, Indicate and Authenticated Signed Writes. Characteristics with a write property are the most critical, since their values can be overwritten and thereby influence the behavior of the system. The encryption and authentication permissions of the ATT protocol can be used to protect a characteristic from unauthorized access. A protected

characteristic enforces the communication over an encrypted and authenticated link [Sivakumaran 18]. This ensures that only paired devices can access the critical values.

4.9 Attacks outside BLE

As a perspective some additional vulnerabilities outside BLE are mentioned here. They will not be discussed further more but still may influence the operating of communicating devices. At first some hardware related attacks are mentioned. In the second part maintenance related weaknesses are labeled.

The influence on a communication starts with the design and manufacturing of the hardware. Already in this stage it is possible to create flaws. Those attacks, where manufacturers deliver corrupted hardware, are called supply-chain attacks or forgery [Barcena 15].

The next stage that creates weak points is the selection of a component with already known vulnerabilities [Barcena 15]. An example is the selection of a device at only implements the Bluetooth 4.0 standard, where most of the paring modes are vulnerable to eavesdropping.

At last hardware related attacks are possible if an attacker has physical access to the device. In this case he could reset the device to factory settings and reconfigure it. Alternatively an attacker can analyze the hardware and get access to secrets inside the memory or he can physically destroy the node [Healy 09].

The most important maintenance related weakness is the potential of updating corrupted firmware. If firmware updates are neither encrypted nor signed it is possible to generate a valid and malicious firmware update[Barcena 15].

5 Requirements on a BLE e-bike profile

This chapter describes how the requirements for the profile were brought together. Basic interaction with the e-bike is a fundamental requirement. This includes the activation of the start-up assistance, controlling of the electrical support engine, getting informations about the current speed of the vehicle and the current battery level of the e-bike. This controlling abilities will be capsuled in one BLE service.

Additional to the control service, it is planned to design a software bridge to access the bike internal CAN-bus via BLE. This enables a better maintainability of the e-bike. Of course this bridge will need additional protection against unauthorized access. Only authorized engineers are allowed to access this nervous system of the bike. The BLE to CAN bridge will be capsuled in a second service.

Further requirements will be captured during this chapter. In the beginning the European specification for e-bikes 'DIN EN 15194' will be considered. This will be completed by some requirements from different literatures. Next, the requirements of Sigma and various partners are mentioned.

5.1 DIN EN 15194

This is an abstract of the European specification for electrical supported bikes DIN EN 15194 [DIN-NASport 18]. The translation is not legally enforceable since it is not verified and some parts are skipped to increase the readability.

- Start-up assistance: there must be an aware and continuous interaction of the user to trigger the assistance. Either during driving without using the pedal or during scooping the bike.
- Protection against manipulation:
  - Procedures against manipulation have to be applied to protect against unauthorized access. Usually users try to manipulate controlling elements or power units to improve the performance. The protection has to prevent the manipulation with commercial tools or normal equipment.
  - Prohibit the unauthorized access to the engine: the following parameters may only be changed by the manufacturer or by authorized personal. Software parameters must be protected from manipulation with programming tools that are not public available.

        * Maximal speed with the support engine
        * Parameters that influence the maximal speed and are limited by design
        * Maximal gear ratio (for systems with a engine before the gearshift)
        * Maximal engine power
        * Maximal speed of the start up assistance
      - Predictable manipulations of relevant configurations must be prevented or protected by appropriate countermeasures (validity checks for detection of sensor manipulation)
      - Connected elements have to be checked for their permission
      - Protection against trackless opening of relevant elements
  - Lighting systems and reflectors must conform the national specifications
  - Appendix C: electro-magnetic tolerances for EPACs (electronic pedelecs) and EUB (electronic sub modules):
      - C.1: Electrical components have a defined limit of allowed electromagnetic emissions up to a frequency of 1 GHz.
      - C.4: The tolerance of an EPAC against electromagnetic emission are measured in a frequency space from 20 to 2000 MHz.
      - C.7: The tolerance of an EUB against electromagnetic emission are measured in a frequency space from 20 to 2000 MHz.

The specification combines requirements from multiple areas that are also important for software design: In case of the user interaction it requires a continuous action of the user to enable the start-up assistance. The protection against manipulation must be considered, including the protection against unauthorized usage of the BLE to CAN bridge.

Since the BLE profile will offer a possibility to enable the start-up assistance, the problem to ensure an aware and continuous interaction to trigger the assistance has to be considered. A possible solution is the usage of a watchdog timer, that stops the assistance automatically if a repeating signal is not received in a specific time. This enforces a connected master device to continuously send messages so that the start-up assistance remains active. The central device may use a button that has to be pressed continuously to identify the will of the user to activate the assistance. So a part of the responsibility will be in the devices, that connects to this profile. The profile has to implement a method to start and stop the assistance. To verify the continuous will of the user is a task of the connected devices.

The protection against manipulation can be understood in two different ways: As protection against access of the bike owner to forbidden components, as the motor.

Or as protection against influences from unauthenticated devices during riding the bike. As an example it is not allowed that an unauthenticated device triggers the start-up assistance.

Both scenarios will be considered: To protect the driver during using the bike, all data transmitted to the bike need an encrypted, authenticated, and MITM protected connection to be accepted. To archive this, the communicating BLE devices have to pair, as described in chapter 3.7. With this measures it can be guaranteed that only information from a authenticated device can influence the driving.

The BLE to CAN service has a very higher potential of abuse, since it is intended to access the bike internal CAN bus to reconfigure the settings. To inhibit a normal user to manipulate the bike configurations via this bridge, an additional encryption is planed. The key for this additional encryption is only known by authorized engineers that are allowed to manipulate the settings.

The electro magnetic tolerances of the bike are defined up to a frequency of 1 GHz for emission. Therefore using the 2.4 GHz band for BLE will be no problem. In case of incoming frequencies the bike has to tolerate frequencies up to 2 GHz. The frequency hopping of BLE, which uses the whole 2.4 GHz band makes it very robust against interferences. This makes it unlikely to block the whole connection by accident.

To be prepared for a jamming attack the service implementation must have a fall back strategy that applies if the connection is lost. The current plan for this strategy is to turn off the support engine and the start-up assistance, while turning on the lights of the bike. So the bike is in a safe state in case of connection loss.

5.2 Security requirements

When analyzing communication systems, three major threats exist: the stealing of sensitive informations, their manipulation and influencing the availability of the service. Depending on the impact of a system malfunction the system may be classified as a safety critical system, which adds additional requirements, as explained later. Therefore, it is important to capture security requirements in a early planning phase. Possible threads and appropriate countermeasures are listed below. Based on the countermeasures the security requirements of the e-bike profile will be derived. [El-Hadary 14] [Aven 09] [Healy 09]

Aven et al [Aven 09] discusses various definitions of critical systems, including this definition: 'A system is considered critical if its failure or malfunction may result in severe consequences, for example loss of lives, environmental damage or economic loss'. This is a fuzzy definition, since 'environmental damage' and 'economic loss' describe a wide range of possible consequences.

Driving an e-bike that is remote controlled by another person could have unexpected behavior as a suddenly starting motor. This increases the chance of an accident dramatically. The accident can lead to serious injury of the bike driver or involved people. Also it may implicate personal economic loss based on damaged vehicles. So the introduced definition can be applied to the given scenario.

The origin of the scenario was an e-bike that was attacked and controlled by another person. To prevent such scenarios Healy et al [Healy 09] describe various approaches, from preventative to reactive measures. Cryptographic primitives to provide authentication, integrity and confidentiality are the most common preventative measures. They make it harder for an attacker to successfully perform an attack. Reactive measures as controlled shutting down the network intervene if an attack happens. Both types of measures are used to derive the security requirements.

Hu et al [Hu 04] point out that the applied security mechanisms have some resource constraints in energy, memory and computing power. Therefore the utilization of already build in mechanisms are essential to avoid overhead and save energy. Chapters 3 and 4 describe attacks against BLE and mechanisms to counter those attacks. These mechanisms will be used in the designed interface to receive the required security levels.

Hereafter, possible attacks against wireless networks will be listed. They are selected from different sources to cover a wide range [Healy 09] [Hu 04] [Sivakumaran 18] [Barcena 15] [Köppel 13]. In a few sentences each attack will be discussed and which measures can be applied to prevent them.

- eavesdropping: Listening to transmitted messages is easy in a wireless network, due to its broadcast nature. BLE offers a build-in 128 bit AES encryption to protect the confidentiality of sensitive information. For private data it is required to encrypt the messages.
- spoofing: Masquerading allows an attacker to get the rights of another entity, for instance controlling the e-bike engine. In BLE this could happen before a communication is encrypted, during the paring procedure. To prevent this, the devices must authenticate to each other while setting up a connection. For all actions controlling the e-bike it is required that the devices communicate via an encrypted and authenticated link.
- jamming: This type of denial of service attacks generates interferences of the transmitted messages. To prevent this, the interference protection of BLE, the channel hopping, can be used. In new devices, implementing Bluetooth specification 5.0 the channel selection algorithm 2 shall be used to make it harder for an attacker to follow the connection. Due to the fact that it is not possible to absolutely prevent jamming, a fall back mechanism is recommended to ensure safe behavior. It is required to implement a secure fall back state and to use channel selection algorithm 2, if available.

- packet-injection: The injection of packets can be prohibit by ensuring authentication and integrity, while modification of transmitted packages can be prevent by providing confidentiality and authentication. BLE offers confidentiality, integrity and authentication by using a AES-CCM encrypted and authenticated communication channel.
- node replication: In BLE this is equal to spoofing. The considered BLE connections are point to point connections and an attacker would have to masquerade as one of both end-devices.
- physically destroying: Against this attack, no protection in software is applicable. As a reactive measure a secure fail state can be applied to protect against injury.
- side channel attacks: These type of attacks try to obtain an increased power consumption or unfavorable behavior of components to damage the system or the user experience. It is hard to give a protection against these attacks, since often they use hardware specific vulnerabilities or unexpected side effects to reach their goal.
- Traffic analysis: The collection of all meta data can't be prevented without a high effort. Nevertheless BLE specification 4.1 offers a privacy feature, which makes it harder for an attacker to track a device over a longer period of time.
- Routing Attacks: This type of attacks is applicable in networks with complex routing. The considered BLE connections are direct point to point connections without advanced routing. So this type of attacks are not applicable in this setup.
- Unauthenticated access: To prevent unauthenticated devices to influence the e-bike behavior, all functions sending information to the e-bike are only usable over an encrypted and authenticated connection. Encryption and authentication is a requirement for all e-bike controlling functions to forbid unauthenticated access.
- Privacy concerns: To reduce doubts about the revealed informations, no user specific informations should be accessible without authentication. The public available informations only reveal the current state of the bike. In combination with the BLE privacy feature, introduced in specification 4.1, it is not possible to gather reliable informations about the user. So the usage of the BLE privacy feature and a limitation of available informations without authentication are additional requirements.

Some of the possible threads had multiple requirements, while some different threads referred to one requirement. To clarify the security requirements, table 5.1 will summarize all requirements that came up. The first column will hold an ID, to enable specific references. The second column holds the requirement and the third column holds the thread that caused the requirement.

Table 5.1: Security Requirements

ID	Requirement	Threads
S1	encrypted communication	eavesdropping, packet-injection, unauthenticated access
S2	authenticated communication	spoofing, packet-injection, unauthenticated access
S3	secure fall back state	jamming, physically destroying
S4	channel selection algorithm 2	jamming
S5	packet signing	packet-injection
S6	BLE privacy feature	traffic analysis, privacy concerns
S7	limited unauthenticated access	privacy concerns

5.3 Functional requirements

The following table hold the requirements raised by different partners. The first column of the table holds an ID, to enable referring a specific requirement. The second column is the requirement itself. The type of information is stored in column three. The fourth column holds the priority of the requirement. The priority is important since it identifies which functionality is essential and which functionality could or should be available. The type classifies the requirement and gives information about the update rate of the feature. All 'live information' will be updated automatically and are intended to notify a connected device. The 'diagnostic information' will not change so frequently and are intended to be sent only on request. 'bike interaction' indicates that these informations will be sent to the bike.

Most of the required data are live informations. To organize this group they are split into sub groups, divided by the respective component. So the sub groups 'live battery information', 'live motor information', 'live messages' and 'live ride and user information' are introduced.

It is important to note that the bike is treated as a sensor system for this service. This means the service is designed to reveal internal informations from the bike and set controlling values to regulate the bike behavior. It is not planed to use this service to manage a display of the e-bike. A display control would be implemented in a separate display service, which is not be considered in this thesis.

Table 5.2 holds the combined requirements from Sigma and multiple partners. Overlapping requirements are already eliminated, for example the current speed was required by multiple companies. Also requirements targeting the user interface are removed, since the target of this service is to handle the e-bike as a multi-sensor system without display.

These requirements are used to design the e-bike service in the following chapter.

Table 5.2: Functional Requirements

ID	Requirement	Type	Priority
F1	get ride time	live information (ride & user)	must
F2	get trip distance	live information (ride & user)	must
F3	get human power	live information (ride & user)	must
F4	get speed	live information (ride & user)	must
F5	get cadence	live information (ride & user)	should
F6	get calories	live information (ride & user)	should
F7	get altitude	live information (ride & user)	should
F8	get heart rate	live information (ride & user)	should
F9	get assist mode	live information (motor)	must
F10	get GPS signal	live information (ride & user)	should
F11	set assist mode	bike interaction (motor)	must
F12	get remaining battery	live information (battery)	must
F13	get gear status	live information	should
F14	get battery health	diagnostic information	must
F15	get light status	live information	must
F16	set light status	bike interaction	should
F17	get battery temperature	live information (battery)	could
F18	get remaining battery at destination	live information (battery)	could
F19	get total distance	diagnostic information	must
F20	get maintenance alert	diagnostic information	must
F21	set maintenance alert	bike interaction	could
F22	trigger walk mode	bike interaction	should
F23	safe walk mode	functional safety (DIN EN 15194)	must
F24	get % of assistance	live information (motor)	should
F25	get motor power	live information (motor)	must
F26	get motor temperature	live information (motor)	could
F27	get charging counter	diagnostic information	should
F28	get error messages	live information (messages)	could
F29	set the logging priority	bike interaction	could
F29	get the logging priority	diagnostic information	should
F30	get the full battery capacity	diagnostic information	should
F31	get the amount of assist modes	diagnostic information	should
F32	get a maintenance reminder	diagnostic information	should

6 The BLE e-bike profile

In this chapter the BLE e-bike profile is described based on the requirements from chapter 5.

Each BLE peripheral device implements one or more services. This combination of BLE services is called a BLE profile. The BLE e-bike profile will consist of multiple services: a generic access profile, a device information service, an e-bike service, a Ble2Can service and an object transfer service.

The generic access, device information and object transfer service are predefined services from the Bluetooth SIG. The first two services are used to exchange general data as the device name, manufacturer name, model number, firmware revision and more. The object transfer service will be used to transfer the log files, which hold information about occurred errors and usage of the bike.

The e-bike and ble2can service will be described in chapters 6.1 and 6.2. The e-bike service will hold the functionality to interact with the bike and provide the data requested by the functional requirements. The Ble2Can service is used to implement the BLE to CAN bridge, which is planed to increase the maintainability.

6.1 Determining the e-bike service

The requested data in table 5.2 from the previous chapter are capsuled in characteristics. Each characteristic holds multiple informations about one component or aspect of the e-bike. The table already divided the requirements in different types as motor live information, bike interaction, diagnostic information and more. These types are used to cluster the information and combine them to groups, which are translated to characteristics.

The following groups are available:

- live ride information: This group reveals data about the current trip, for example the speed and distance. The requirements F2 to F5 are captured. Requirement F1 is not included, since the trip time can also be measured by receiving update-messages of this characteristic. F6 is based on a heart rate measurement, this value would be delivered by an external sensor, so this requirement is excluded. Currently there is no e-bike with the sensors required for the GPS

signal (F10) and the altitude (F8) planned, so those values are not included in this service.

- live motor information: All information about the current state of the motor are combined in this group. The requirements F24 to F26 (motor power, motor temperature and % of assistance) form this group.
- live battery information: The actual state of the battery is revealed by this group. The requirements F12, F17 and F18 are met. The remaining battery at destination (F18) is not possible to calculate, since the bike does not know the destination of the driver. Nevertheless it is possible to calculate the remaining distance, based on the actual battery usage. This information will be send instead of the remaining battery at destination.
- bike light: This group holds all data about the connected lights. Three lights are planned: a common front light, a common back light and a high beam for the front light. Each light can be controlled by the service, so this characteristic needs to be read and write-able to fulfill the requirements F15 and F16.
- bike assistance: The bike assistance handles the control of the support engine and trigger the walk mode. To fulfill the requirements F9, F11 and F22 it is read and write-able. Requirement F23 targets the functional safety, which has to be considered in the implementation as already discussed in chapter 5.3.
- bike messages: Messages are notifications sent by the e-bike. This might be an error message, a gear shifting advice or any generic message that is important for the driver. The characteristic is able to distinguish between different types of messages, as shifting advices or errors. Additional to the message type an arbitrary UTF-8 string is allowed to further specify the information.
- bike status: This group summarizes the remaining live information that are given by the bike. The only requirement fulfilled in this group is F13, the gear status. Additional this group includes some flags to control the current bike status:
  - forced eco: it is triggered if the battery is low and the bike switches to a economic mode
  - charging: this flag indicates that the battery is charging. This can be used to give a feedback to the connected battery charging tool
  - service tool: this flag is for maintenance work, to get a feedback if the bike accepts a connected service tool
- bike diagnostics: Diagnostic data as the total ride time and distance, the battery health or the charging cycle counter is contained in this group. So the requirements F14, F19 and F27 are fulfilled.
- bike setup information: Information about the initial setup of the bike are summarized here. The battery design capacity, the number of assist modes and the

current version of this service are some examples of available data of this group. The requirements F30 and F31 are fulfilled by this.

- bike log information: Details about the internal logging of the bike are available in this block. Requirement F29 is satisfied. But F28, the requirement to set the logging, is not implemented, due to the fact that not every user should be able to manipulate the internal logging.
- bike maintenance interval: With the maintenance interval, it is possible to remind the driver to do the next service. For this purpose, the user or the technician can set the date for the next maintenance and for the reminder. This complies requirement F32.

Depending on the revealed data and the access operations of a characteristic, it must fulfill multiple security requirements from table 5.1.

The device implementing the services has to fulfill some security standards independent of the characteristic specific security requirements:

The device has to implement a fall back strategy to react on interrupted connections to fulfill requirement S3. Corresponding to S4 the device should use the channel selection algorithm number 2, which is available since specification 5.0. The third security requirement the implementing device has to fulfill is S6, the usage of the BLE privacy feature, which is available since specification 4.1.

To match requirement S7, the limited unauthenticated access, only readable characteristics are available without authentication. Controlling characteristics that influence the behavior are only write-able with secure connections that are encrypted and authenticated.

- live ride information, live motor information, live battery information, bike status, bike messages: These five characteristics reveal data about the current state of the e-bike. The values may change frequently, so the characteristics can be accessed via read and notify. This reduces the interaction with the bike to a minimum. Security concerns as requirements S1, S2 or S5 are not necessary for this characteristics.
- bike diagnostics, bike setup information, bike log information: The information revealed in these characteristics are not changing frequently, and are not so important during a ride. The access to the characteristics is only possible via a read operation. This reduces the interaction to a minimum, without any security threat for the bike. So they do not have to fulfill the requirements S1, S2 or S5.
- bike light, bike assistance: These two characteristics are used to control the behavior of the e-bike and they may change multiple times during a ride. They will be accessible via read, notify and write operations. To perform a write operation, the connection has to fulfill all security requirements. So it has to be

an encrypted (S1), authenticated (S2) communication channel, and the devices have to sign the packages (S5).

- bike maintenance interval: The maintenance interval will not change very often, so a read operation once in a while will be enough to remind the user of a recommended maintenance. The users should be able to handle the reminder themselves. This necessitates to give the users write access to this characteristic. Since only authorized devices should be able to write data on the bike, the write is only allowed via an encrypted (S1), authenticated (S2) communication channel, while the communicating devices sign the date (S5).

The listed security requirements ensure that no uncontrolled access to the e-bike is possible. To improve the privacy of the e-bike, encryption may be applied to all characteristics (S1). But this would enforce more user-interaction to connect even simple devices as bike computers, that only require read access. The current state is a trade-off between usability and security. The readable data of the characteristics are considered to be uncritical for the privacy of the driver. So it seems to be a good balance to restrict the access only for writing functions, that influence the safety and experience of the users.

6.2 Determining the Ble2Can service

The BLE to CAN bridge is considered to be a maintenance tool for technicians to access the inner functions via the CAN bus. Currently some e-bikes have USB-slots as service points. Their usage requires a specific reader with the appropriate software. A BLE interface reduces the requirements to an appropriate software. But not every user should be able to use this feature, which rises the requirement of an additional access restriction.

This restriction is archived by an additional encryption. Symmetric encryption is much more efficient than asymmetric [Schmitt 16] and many micro controllers have hardware accelerators for AES encryption.

Currently it is planned to use a KW35 micro-controller [1] for implementing the peripheral device in future e-bikes. It supports BLE version 5.0, has an AES accelerator and an integrated CAN controller, which makes it well suited for this project.

So AES encryption is no problem for this controller, which makes it a suitable candidate for controlling the access rights to the CAN bus. The key to decrypt the message has to be stored inside the peripheral.

Soring a key in the memory of a micro controller always leads to the possibility that an attacker could open the device, read out the memory and reveal the key. On the

[1] micro-controller KW35/36 on www.nxp.com

other hand it is more easy for an attacker to open the case of the e-bike and access the CAN bus directly via a plug-in connector. In case a key was revealed, the firmware of the peripheral should be updated to replace the revealed key.

The service is panned to contain three characteristics:

- A salt: The salt is intended to change every time a possible CAN message is sent to the service. Without the salt the peripheral would have no possibility to check if the received bytes are a valid CAN message. This may be used by an attacker to bypass the additional encryption. A random generated salt reduces this possibility dramatically. The salt can be accessed only via read operations, so no additional security requirements are necessary for this characteristic.
- The CAN message: The message that shall be send to the CAN bus is written to this characteristic. To ensure that the sender of the message has the rights to use the CAN bus, the message has to be encrypted with the key in a cypher. The cypher contains the CAN bus message and the salt. The peripheral decrypts the cypher and compares the decrypted salt with the actual salt. Only if the salts are equal, the decrypted CAN bus message is forwarded to the CAN bus. Independent of the validity of the received cypher, the peripheral generates a new salt.

  Additional to the cypher this characteristic will also hold a CAN response ID. The peripheral uses this ID to listen on the CAN bus for a message with this ID. This message is the response that belongs to the forwarded CAN bus message. The response is published via the third characteristic of this service.

  This CAN message characteristic is only write-able via an encrypted (S1) and authenticated (S2) communication channel.
- The CAN response: This characteristic is used to reveal the received response from the CAN bus. The CAN message characteristic is given a CAN response ID. The device listens in the CAN bus for a message with a corresponding ID and forwards the message to this characteristic, so the technician can read the response. To enable this interaction, this characteristic must be readable and notify-able.

Figure 6.1 visualizes the desired interaction with the Ble2Can service. It is assumed that the connection is already encrypted and authenticated. In the beginning the central device subscribes to the notify function of the CAN response characteristic.

The following request attempts to read the actual salt of the peripheral. This is necessary to encrypt the CAN message correctly. After the encryption of the CAN message the central writes the CAN message characteristic to the peripheral. When the peripheral has received the message, it decrypts the CAN message and checks the validity by comparing the decrypted salt with its actual salt. The peripheral generates a new salt, independent of the correctness of the decrypted message. If the decrypted message is correct, the CAN message is forwarded to the CAN bus of the bike.

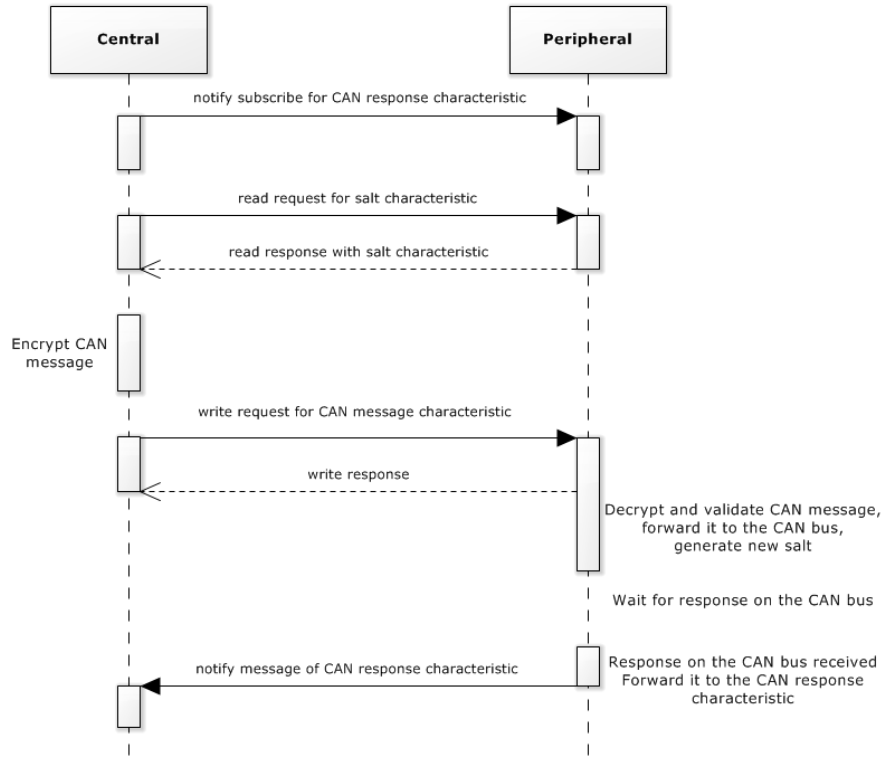


Figure 6.1: A sequence diagram of the Ble2Can bridge usage

The CAN controller of the peripheral is listening on the CAN bus for a response with a ID equal to the transmitted response ID. If the controller receives a suitable response, the response is forwarded to the CAN response characteristic. The central automatically gets a notify message that the CAN response characteristic is updated.

An alternative interaction is that the central writes additional CAN messages to the peripheral, before a response is expected. In this case, the sequence diagram repeats from the second message on. The central reads the actual salt, encrypts the next CAN message and writes the new CAN message characteristic. The peripheral checks the validity, generates a new salt and forwards the received CAN message. This may repeat multiple times. The central can use a zero response ID to indicate the peripheral that no CAN response is expected, so the CAN controller does not listen to the bus.

6.3 Formal definition of the profile

The BLE profile of the peripheral consists of multiple services. In context of the development of an e-bike service the entire profile, surrounding the designed service, is presented briefly. In the following sub-chapters the planned services are described. For predefined services, a detailed documentation is available on the official blue-

Table 6.1: General Access service

	Name	UUID	Requirement
Service	General Access	0x1800	
Characteristic	Device Name	0x2A00	Mandatory
Characteristic	Appearance	0x2A01	Mandatory
Characteristic	Peripheral Privacy Flag	0x2A02	Optional*
Characteristic	Reconnection Address	0x2A03	Conditional on 0x2A02
Characteristic	Peripheral Preferred Connection Parameters	0x2A04	Optional

tooth website[2]. Custom designed services will be fully described in this section.

6.3.1 General access service

The general access service is a default service, that is available on all BLE devices. This service is predefined by the Bluetooth SIG, so the UUIDs in table 6.1 only have a size of 16 byte. Two characteristics are mandatory: the device name and the appearance. The appearance is used to classify the device, example classifications are 'Generic Phone', 'Generic Computer', 'Generic Watch' or 'Generic Remote Control'. The last one might be the best fitting for this purpose.

The characteristic 'peripheral privacy flag' is marked as 'Optional*', because it is only optional if the peripheral supports only 1 bond, otherwise this characteristic is excluded. If the privacy flag is implemented the reconnection address must be implemented, too.

It is recommended to use this standardized service to reveal the device name and classify the device via a standardized service.

6.3.2 E-Bike service

This service is designed to enable secure interaction with an e-bike. The e-bike is treated as a basic sensor- actuator system, where the analysis and evaluation of the collected data is done in connected higher level devices.

Table 6.2 gives an overview of the characteristics of the E-Bike service. The service has a custom 128 bit UUID, because it is a custom designed service and not adopted by the Bluetooth SIG. The 3rd and 4th most significant bit of the service UUID is used as a counter and is incremented for every custom characteristic of the service.

Each characteristic has an individual table with details of the characteristic. As an example table 6.3 shows details about the Live ride & user information characteristic. The table is split in two parts:

[2]predefined services on www.bluetooth.com

Table 6.2: E-Bike service

	Name	UUID	Details
Service	E-Bike	00000000-abab-4499-b9d7-d0efc0d06477	
Characteristic	Live Ride & User Information	00000001-abab-4499-b9d7-d0efc0d06477	Table 6.3
Characteristic	Live Motor Information	00000002-abab-4499-b9d7-d0efc0d06477	Table 6.4
Characteristic	Live Bettery Information	00000003-abab-4499-b9d7-d0efc0d06477	Table 6.5
Characteristic	Bike Status	00000004-abab-4499-b9d7-d0efc0d06477	Table 6.6
Characteristic	Bike Diagnostic Information	00000005-abab-4499-b9d7-d0efc0d06477	Table 6.7
Characteristic	Bike Setup Information	00000006-abab-4499-b9d7-d0efc0d06477	Table 6.8
Characteristic	Bike Assistance	00000007-abab-4499-b9d7-d0efc0d06477	Table 6.9
Characteristic	Bike Light	00000008-abab-4499-b9d7-d0efc0d06477	Table 6.10
Characteristic	Bike Messages	00000009-abab-4499-b9d7-d0efc0d06477	Table 6.11
Characteristic	Bike Log Information	0000000a-abab-4499-b9d7-d0efc0d06477	Table 6.12
Characteristic	Bike Maintenance Interval	0000000b-abab-4499-b9d7-d0efc0d06477	Table 6.13

The upper part holds informations about the characteristics UUID, required descriptors and necessary access properties. In this example the characteristic has a client characteristic descriptor to enable the notify functionality. The properties regulate how the characteristic can be accessed via BLE, as already explained in chapter 3.5.

The lower part of the table shows detailed information how the data array of the characteristic is composed. In case of this characteristic, the data array has a size of 10 byte, where the first 2 bytes are interpreted as a unsigned 16-bit integer value to revive the speed value. The following 4 bytes are an 32 bit unsigned integer value for the traveled distance.

In this way all characteristics are composed by multiple values. The size of each value is determined by the required resolution and possible maximal values. For the distance it is mandatory to have a resolution in the order of a meter, so a 16 bit value allows the maximal distance of 65 km, which is not enough. The next available standard value is the 32 bit value, this allows a maximal distance of 4 million kilometers. Of course it would be possible to make a proprietary data type with a 24 bit integer (maximal distance of 16.777 km), but this is not favorable since it uses non standard types, which makes it more difficult to use.

6.3.3 Ble2Can service

This service is the Bluetooth interface of the Bluetooth to CAN bridge. Table 6.14 shows the UUID of the Ble2Can service and the three contained characteristics. Again, the 4th most significant bit is used as a counter for the UUIDs of the characteristics. Each characteristic has a individual table that reveals more details about the corresponding characteristic.

For the CAN message characteristic it is important to handle the data format in the right way, so that authorized persons are able to access the CAN bus. Table 6.15 reveals that the first four byte are reserved for the CAN response ID.

Table 6.3: Live Ride & User Information Characteristic

UUID	00000001-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, notify		
Name	Unit	Format	Definition
Speed	0.1 km/h	uint16	0 - 65534: current speed in 0.1 km/h steps 65535: no information available
Distance	m	uint32	0 - 4294967294: traveled trip distance 4294967295: no information available
Tredale power	W	uint16	0 - 65534: power generated by the driver 65535: no information available
Tredale torque	Nm	uint8	0 - 254: torque generated by the driver 255: no information available
Cadence	1/min	uint8	0 - 254: cadence of the tredale 255: no information available

Table 6.4: Live Motor Information Characteristic

UUID	00000002-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, notify		
Name	Unit	Format	Definition
Motor power	W	uint16	0 - 65534: power consumed by the motor 65535: no information available
Motor torque	Nm	uint16	0 - 65534: torque generated by the motor 65535: no information available
Motor temperature	°C	uint8	0 - 254: temperature of the motor, with a -50 °C offset 255: no information available
Motor assistance	%	uint8	0 - 100: actual support at this assistance level 101 - 254: not used 255: no information available

Table 6.5: Live Battery Information Characteristic

UUID	00000003-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, notify		
Name	Unit	Format	Definition
Battery level	%	uint8	0 - 100: actual battery level 101 - 254: not used 255: no information available
Battery current	mA	sint16	-32768 - -1: electrical current, going into the battery 0 - 32766: electrical current, going out of the battery 32767: no information available
Battery voltage	mV	uint16	0 - 65534: actual battery voltage in mV 65535: no information available
Battery temperature	°C	uint8	0 - 254: temperature of the battery, with a -50 °C offset 255: no information available
Estimated range	km	uint16	0 - 65534: internal calculated range 65535: no information available

Table 6.6: Bike Status Characteristic

UUID	00000004-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, notify		
Name	Unit	Format	Definition
Front gear	-	uint8	The chain-ring with the least gear tooths is at 1. 0: system starting 1 - 254: current gear of the front shifter 255: no information available
Rear gear	-	uint8	The chain-ring with the most gear tooths is at 1. 0: system starting 1 - 254: current gear of the rear shifter 255: no information available
System state	-	1 byte	1st bit: 1 = forced eco is active 2nd bit: 1 = system is charging 3rd bit: 1 = service tool connected 4th - 8th bit: reserved for future use

Table 6.7: Bike Diagnostic Information Characteristic

UUID	00000005-abab-4499-b9d7-d0efc0d06477		
Descriptors	-		
Access properties	read		
Name	Unit	Format	Definition
Total ride distance	m	uint32	0 - 4294967294: total driven distance (odometer) 4294967295: no information available
Total ride time	sec	uint32	0 - 4294967294: total used time 4294967295: no information available
Battery state of health	%	uint8	0 - 100: health of the battery 101 - 254: not used 255: no information available
Charging cycle counter	-	uint16	0 - 65534: how often the battery was charged 65535: no information available

Table 6.8: Bike Setup Information Characteristic

UUID	00000006-abab-4499-b9d7-d0efc0d06477		
Descriptors	-		
Access properties	read		
Name	Unit	Format	Definition
Number of assist modes	m	uint8	0 - 254: the number of supported assist modes 255: no information available
Battery design capacity	0.1 Ah	uint16	0 - 65534: capacity of the battery at design time 65535: no information available
Battery design voltage	V	uint8	0 - 254: voltage of the battery at design time 255: no information available
Light availability	-	1 byte	1st bit: front light: 1 = available 2nd bit: front light high beam: 1 = available 3rd bit: rear light: 1 = available 4th - 8th bit: reserved for future use
BLE service major version	-	uint8	0 - 254: major version of the service 255: no information available
BLE service minor version	-	uint8	0 - 254: minor version of the service 255: no information available

Table 6.9: Bike Assistance Characteristic

UUID	00000007-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, signed write, notify		
Name	Unit	Format	Definition
Assist mode	-	uint8	0: no assistance 1 - 254: current active assist mode 255: no information available
Start up assistance	-	1 byte	1st bit: start up assistance: 1 = assistance is active 2nd - 8th bit: reserved for future use

Table 6.10: Bike Light Characteristic

UUID	00000008-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, signed write, notify		
Name	Unit	Format	Definition
Light modes	-	1 byte	1st bit: front light flash mode: 1 = flash mode on 2nd bit: front light high beam flash mode: 1 = flash mode on 3rd bit: rear light flash mode: 1 = flash mode on 4th bit: auto mode: 1 = brightness sensor controls the lights 5th - 8th bit: reserved for future use
Front light brightness	%	uint8	0 - 100: brightness: 0 = light is off, 100 = light is fully on 101 - 254: not used 255: no information available
Front light high beam brightness	%	uint8	0 - 100: brightness: 0 = light is off, 100 = light is fully on 101 - 254: not used 255: no information available
Rear light brightness	%	uint8	0 - 100: brightness: 0 = light is off, 100 = light is fully on 101 - 254: not used 255: no information available

Table 6.11: Bike Messages Characteristic

UUID	00000009-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Descriptor		
Access properties	read, notify		
Name	Unit	Format	Definition
Category	-	uint8	0 - 4: error message with sevirity level 0 to 4 5: Gear shifting advice front up 6: Gear shifting advice front down 7: Gear shifting advice rear up 8: Gear shifting advice rear down 9: Assist mode shifting advice up 10: Assist mode shifting advice down 11 - 254: reserved for future use 255: no information available
Message length	-	uint8	0 - 255: length of the message in bytes
Message	UTF-8 string	byte[ ]	UTF-8 encoded string with an error message

Table 6.12: Bike Log Information Characteristic

UUID	0000000a-abab-4499-b9d7-d0efc0d06477		
Descriptors	-		
Access properties	read		
Name	Unit	Format	Definition
Logging interval	sec	uint16	0: no logging 1 - 65535: number of seconds between two data-logs
Data-log amount	-	uint16	0 - 65534: number of log messages, that is stored 65535: no information available
Error-log amount	-	uint8	0 - 254: number of error messages, that is stored 255: no information available
Minimal error severity	-	uint8	0 - 4: minimal stored severity level 5 - 254: not used 255: no information available

Table 6.13: Bike Maintenance Interval Characteristic

UUID	0000000b-abab-4499-b9d7-d0efc0d06477		
Descriptors	-		
Access properties	read, signed write		
Name	Unit	Format	Definition
Day of next maintenance	day	uint8	0: no next maintenance 1 - 31: day of a month 32 - 255: reserved for future use
Month of next maintenance	month	uint8	0: no next maintenance 1 - 12: month 13 - 255: reserved for future use
Year of next maintenance	year	uint16	0 - 2018: reserved for future use 2019 - 65535: year
Date reminder	days	uint8	0: no reminder 1 - 254: reminder X days before next maintenance 255: no information available
Odometer at next maintenance	km	uint16	0 - 65534: odometer where the next maintenance is due 65535: no information available
Distance reminder	km	uint8	0: no reminder 1 - 254: reminder X km before the next maintenance 255: no information available

Table 6.14: Ble2Can service

	Name	UUID	Details
Service	Ble2Can	00000100-abab-4499-b9d7-d0efc0d06477	
Characteristic	CAN message	00000101-abab-4499-b9d7-d0efc0d06477	Table 6.15
Characteristic	CAN response	00000102-abab-4499-b9d7-d0efc0d06477	Table 6.16
Characteristic	Salt	00000103-abab-4499-b9d7-d0efc0d06477	Table 6.17

The identifier of a can frame can have 11 bits or 31 bits. The first bit in the 4 byte array indicates which type of ID is available. In case of an standard ID only the last 11 bit of the 4 byte array hold the ID. If the ID is an extended ID the all remaining 31 bit are holding the ID.

The second part of the CAN message characteristic holds informations about the length of the raw CAN message in bit. It is important to notice that BLE stores and processes data in byte-packages. So it may be necessary to append some bits at the end of the raw CAN message, so it fits into the next larger byte array. With the exact length in bits, the receiving peripheral is able to ignore appended bits and forward the original raw CAN message to the CAN bus.

In the last part is the encrypted CAN message. To handle the format correctly, the salt has to be appended to the CAN message, so that the first n bytes belong to the message and the last 16 bytes belong to the salt. This combined byte array has to be encrypted. Figure 6.2 visualizes the required structure of the CAN message.

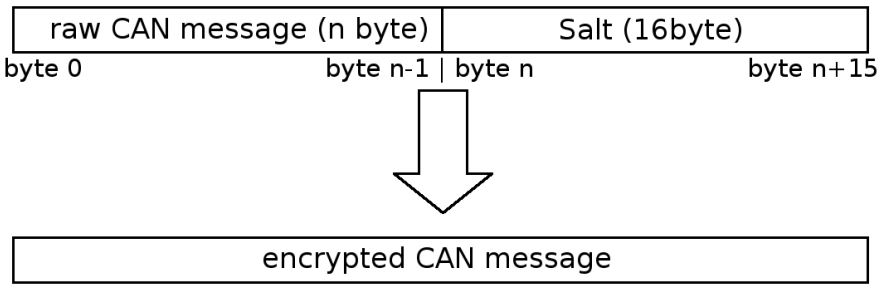


Figure 6.2: The construction of the CAN message

The maximal length of a characteristic is 512 byte, as mentioned in chapter 3.5. This allows a maximal size of the encrypted string of 506 byte. Unfortunately 128 bit AES-CBC generates cyphers with a multiple length of 16 byte, which leads to a maximal cypher length of 496 byte. The plain text can be at the same size or smaller, so maximal 496 byte plain text are possible. Subtracting the 16 byte of the salt leaves a maximal message length of 480 byte, 3840 bit.

Table 6.15: CAN message Characteristic

UUID	00000101-abab-4499-b9d7-d0efc0d06477		
Descriptors	-		
Access properties	read, signed write		
Name	Unit	Format	Definition
CAN Response ID	-	4 byte	1st bit: 0 = standard ID; 1 = extended ID standard ID: 2nd - 21th bit: 0; 22st - 32th bit: ID extended ID: 2nd - 32th bit: ID all bits 0: no response expected
Message length	-	uint16	0: no message and no salt 1 - 3840: length of the message in bit, exclusive 128 bit salt 3841 - 65535: not used
Message	-	byte [ ]	encrypted(raw CAN message, salt)

Table 6.16: CAN response Characteristic

UUID	00000102-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Configuration		
Access properties	read, indicate		
Name	Unit	Format	Definition
Response length	-	uint16	0: invalid CAN message, no response 1 - 4080: length of the message in bit 4081 - 65535: not used
CAN Response	-	byte [ ]	response of the can message

Table 6.17: Salt Characteristic

UUID	00000103-abab-4499-b9d7-d0efc0d06477		
Descriptors	Client Characteristic Configuration		
Access properties	read, indicate		
Name	Unit	Format	Definition
Salt	-	byte [16]	128 bit random value, changes each CAN message write

Table 6.18: Device Information service

	Name	UUID
Service	Device Information	0x180A
Characteristic	System ID	0x2A23
Characteristic	Model Number	0x2A24
Characteristic	Serial Number	0x2A25
Characteristic	Firmware Revision	0x2A26
Characteristic	Hardware Revision	0x2A27
Characteristic	Manufacturer Name	0x2A29

6.3.4 Device information service

The Device Information is a Bluetooth SIG adopted service. Table 6.18 shows some possible characteristics for this service. It only shows some possible characteristics, because all characteristics in this service are optional, so it can be fully adapted to the individual requirements.

It is recommendable to use this standardized service to store as many information about the e-bike in the device as possible. For example the manufacturer name and the model number can be used to get a instruction manual for this specific bike. The firmware and hardware revisions may reveal informations about the integrated components.

6.3.5 Object transfer service

The Object transfer service is the third Bluetooth SIG adopted service in this profile. In the e-bike profile, this service is used to transfer the log files of the e-bike internal data and error log.

Table 6.19 shows a possible composition of the Object transfer service. Again, the UUIDs are only 16 bit, due to the Bluetooth SIG conventions. Some of the characteristics, as the object size, are mandatory. Multiple characteristics are marked with 'Optional*', to indicate that they are only optional if the host device transfers only one object. In case of multiple transferable objects these characteristics are mandatory. Only the 'Object Last Modified' characteristic is always optional.

This service requires an encrypted connection. It is recommended to add authorization to the requirements for this service to ensure the correct identity of the connected device.

For more detailed information about the service or the characteristics, the official Bluetooth website has a list of predefined services[3].

[3]Bluetooth SIG predefined services

Table 6.19: Object Transfer service

	Name	UUID	Requirement
Service	Object Transfer	0x1825	
Characteristic	OTS Feature	0x2ABD	Mandatory
Characteristic	Object Name	0x2ABE	Optional*
Characteristic	Object Type	0x2ABF	Mandatory
Characteristic	Object Size	0x2AC0	Mandatory
Characteristic	Object Last Modified	0x2AC2	Optional
Characteristic	Object ID	0x2AC3	Optional*
Characteristic	Object Properties	0x2AC4	Mandatory
Characteristic	Object Action Control Point	0x2AC5	Optional*

7 The experimental setup

This chapter describes an example setup that implements the designed BLE e-bike and Ble2Can service. This setup consists of two android applications. One app emulates an e-bike and implements the BLE peripheral. The other app implements a central and connects to the peripheral to control the emulated e-bike.

The security permissions of the peripheral are according to the requirements, so the necessary traffic will be encrypted and the devices will have to authenticate to perform write requests on the peripheral.

In the following the architecture and functionality of both applications are described.

7.1 The e-bike emulator application

The task of the e-bike emulator is to implement a BLE peripheral to provide an interface that can be utilized by a connecting device.

The app does not implement the whole e-bike BLE profile. It only implements the two new designed services. The single values in the characteristics of the services will not be realistic, since the emulator has no ambition to simulate the behavior of an e-bike.

The Application is designed to be as simple as possible. Figure 7.1 shows a screenshot of the emulator application. When starting the app, the BLE service is created and the values of the e-bike change automatically.

There are just a few possible interactions for the user:

- Enable and disable advertising: This might be the most important user interaction. By pressing the BLE advertising button, the advertising of the device will be turned on and off. In the screenshot the advertising is currently turned on.
- The NFC button was intended to start an OOB paring. But the paring is not jet implemented, so the button is deactivated.
- Three check boxes are used to regulate the availability of the front lights, front high beam and the rear light. The check boxes are not visible in the screenshot, the user has to scroll down to find them.

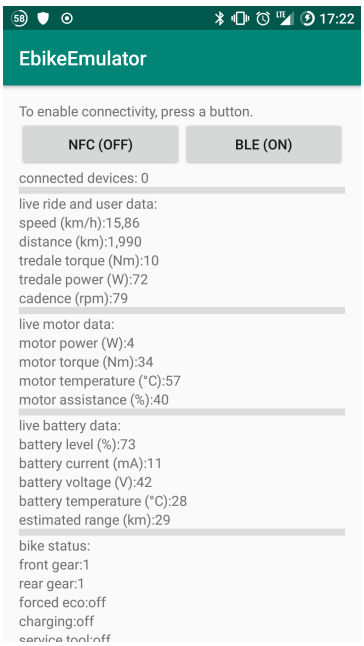


Figure 7.1: A screenshot of the emulator application

The values shown in the user interface are clustered, as the screenshot in figure 7.1 indicates. The clustering was introduced to increase the readability. Each group holds the values of one characteristic.

7.1.1 The class structure

The graphical interaction is enabled by the UIhandler. This class implements the whole functionality of the user interface.

Figure 7.2 shows a class diagram of the classes, involved in the e-bike emulator. Central in the diagram is the MainActivity. This class is the core of the emulator. It sets up the other classes, it is the interface for the operation system to interact with the application and it creates the BLE GATT server to enable the BLE communication. The GATT server provides a connection interface for the services to the underlying BLE structure.

At the top of the figure the GattServerCallback is visible. It is created by the MainActivity to react on incoming requests on the GATT server. The request can be a connection request, a read or a write request. Depending on the type of the request, the callback may forward the request to the e-bike service or the Ble2Can service.

This two classes, on the left and right side of the class diagram, are the heart of the services. They implement the data structure and operations of the BLE services.

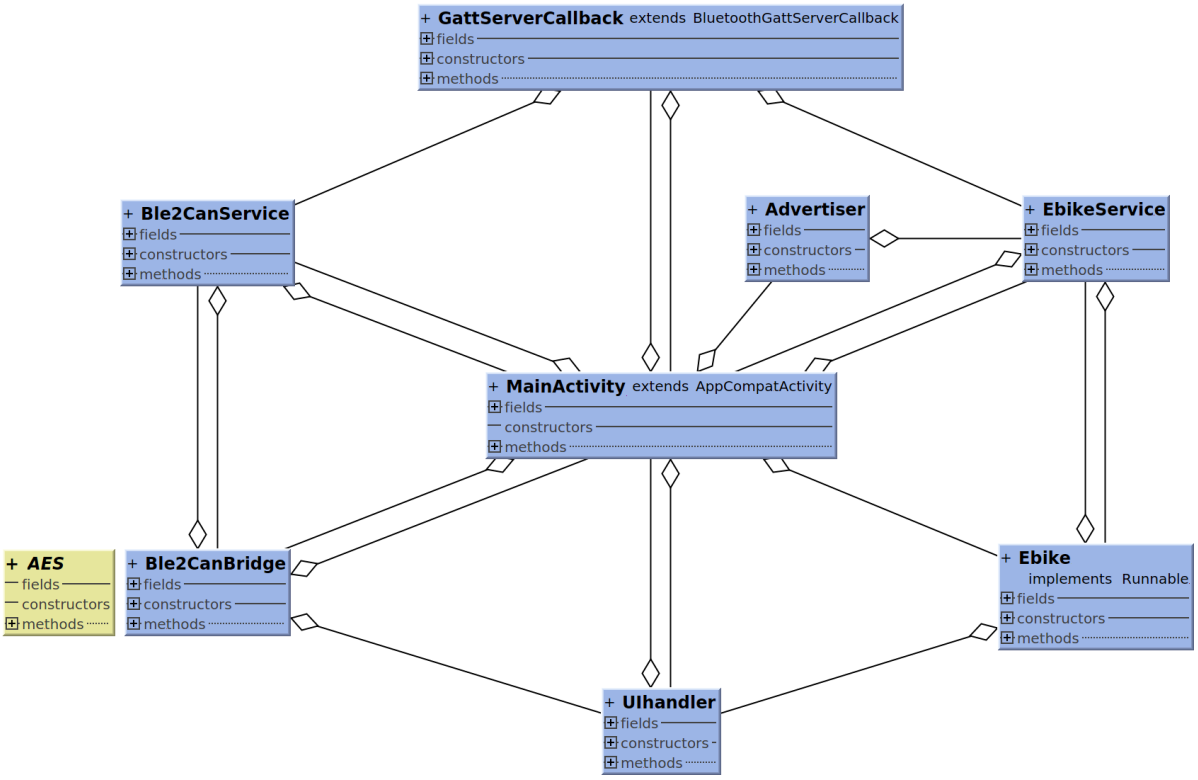


Figure 7.2: The class structure of the e-bike emulator.

They are strongly connected to the Ble2CanBridge-class and the Ebike-class, which are printed below them. This two classes hold the logic of the services. The Ebike-class additionally updates the values of the emulated e-bike each second. This is done in a separate thread, to be independent of the rest of the application. The Ble2CanBridge-class utilizes the abstract AES-class on its left side to decrypt the CAN-message.

The Advertiser-class is only used to start and stop the BLE advertising. For advertising it uses the UUID of the e-bike service. To get this UUID the advertiser has a connection to the EbikeService-class.

To explain how the classes interact, two scenarios are mentioned:

- In case a connected device writes a value to the e-bike service, the request is received by the GattServerCallback. It will forward the write-request to the EbikeService. The service will update the characteristic and the values of the Ebike-class. This class will forward the changes to the UIhandler to display the new value.
- Once a second, the Ebike-class updates the values of the emulated e-bike. The updated values are passed to the EbikeService-class to write the new values into the characteristic. The EbikeService-class checks if a connected device has subscribed to this characteristic to send a notify message with the new values. After this the Ebike-class calls the UIhandler to update the values of the UI.

7.1.2 The Ble2Can service

The implementation of the emulated Ble2Can service is more than just generating random values and displaying them, as in the e-bike service. The Ble2Can service implementation contains additional encryption and validity checks as recommended for the final implementation of this service.

The salt of the salt characteristic is random generated and changes on every write operation on the CAN-message characteristic.

The CAN-message characteristic contains three values: a 32 bit field for the response ID, an 8 bit field for the length of the CAN-message and an arbitrary encrypted can-message. The 32 bit for the response ID are implemented as a 32 bit unsigned integer. An interpretation of the first bit to distinguish between standard and enhanced identifier is not available. The eight bit for the length of the CAN-message are used by an eight bit unsigned integer to represent the length of the raw message, just as intended by the specification. The CAN-message field is treated as an encrypted bit array. It is decrypted by the emulator and the decrypted salt, the last 16 byte of the decrypted message, are compared to the actual salt value. If the values are equal, the message is accepted.

The CAN-response characteristic depends heavily on the CAN-message characteristic. If the message received in the CAN-message characteristic is accepted, a new CAN-response value is generated. Since there is no CAN-bus to listen for messages, the response is generated by using the UTF-8 string 'myResponse' and appending the response ID given in the CAN-message characteristic. In case of an invalid CAN-message characteristic, the response is set to an empty string.

The specification does not jet specify which encryption is used to protect the service against unauthorized usage or which encryption parameters are used. The emulator uses AES-CBC with the UTF-8 string 'theBestSecretKey' as a 128 bit key. It is not recommended to use this key in a later implementation. The salt is used as initialization vector.

The AES-CBC encryption is a block cypher. The cypher text is generated in blocks with 16 bytes length. This may increase the size of the encrypted data to a multiple of 16 bytes. During decryption AES-CBC recovers the original string, regardless of an increased cypher text length.

7.2 The control application

The second application is able to control the emulated e-bike. It communicates with the peripheral via the BLE interface and interacts with the emulated e-bike, using the new designed services. It is the counterpart of the emulator application of the previous chapter.

This application consists of two parts: The first part to find other BLE devices is shown in the left screenshot of figure 7.3. The second part connects to one BLE device and accesses the e-bike service, it is shown in the middle and the right screenshot of figure 7.3.

The first part scans for connectable BLE devices. To do so, the device is listening on the BLE advertising channels for advertising messages. Devices sending those messages are added to a list of connectable devices. The user can connect to one of the listed devices to activate the second part of the application.

To implement the scanning multiple classes are used, they are shown by figure 7.4.

Again, the MainActivity is the heart of the application. It is the access point of the operating system and sets up the other classes.

The BLE_Scanner is controlling the BLE operations to search for new devices and it creates the BLE_Scanner_Callback. The callback is executed if a new device is found. In this case, the callback extracts the device name and the device address from the advertising message. With this information a new BLE_Device object is created. This object is added to the list of available devices via the Devices_List_Adapter class.

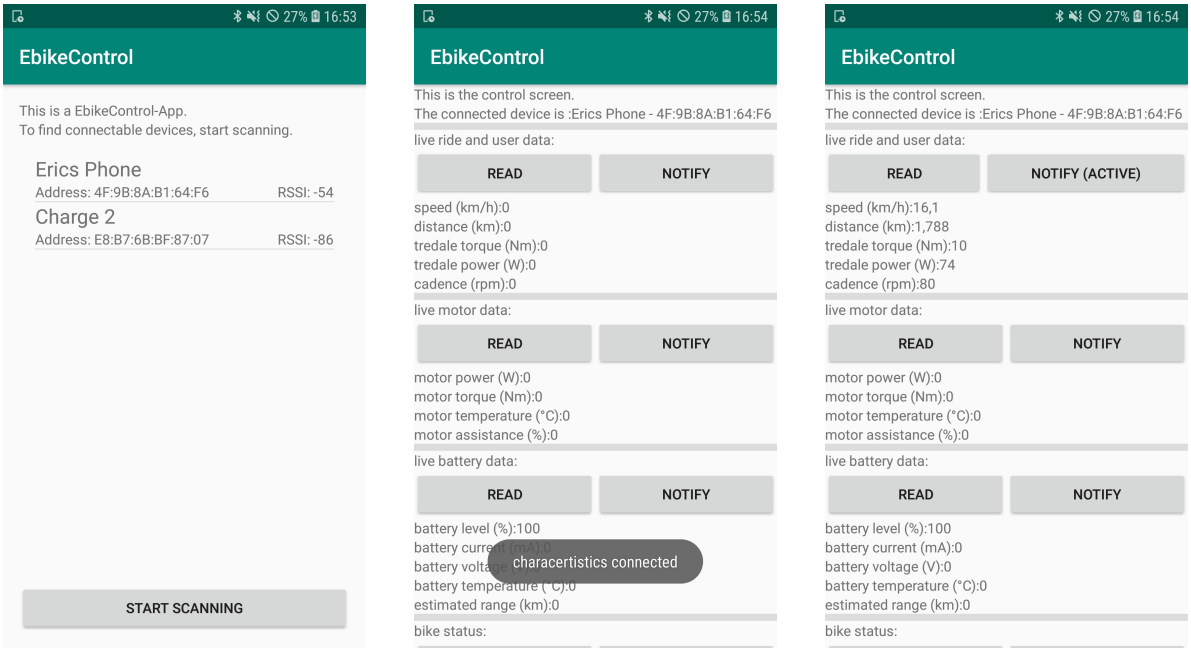


Figure 7.3: Screenshots of the control application

The screenshots show three important functionalities of the control application: Search for other BLE devices (left), automatically connect to the characteristics (middle), activated notify on the live ride and user characteristic (right).

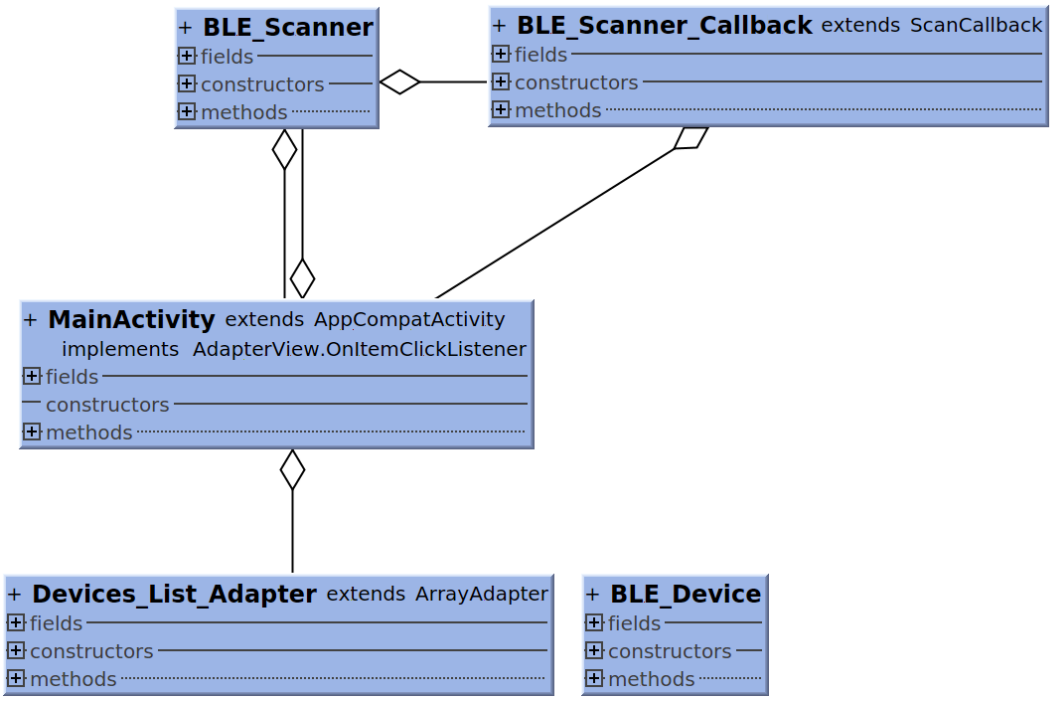


Figure 7.4: The class diagram of the first part of the control application, used to find other BLE devices

The updated list is shown to the user. The left screenshot of figure 7.3 shows a device list, containing two BLE devices.

To connect with a device, the user has to select the desired device in the list. This activates the second part of the application, which will connect to the selected device. This is done in a new screen, that is already prepared to control the e-bike and Ble2Can services.

In the background the application will search for the necessary service UUIDs and characteristic UUIDs on the connected peripheral. If the UUIDs are found the message 'characteristics connected' indicates that the application is ready to control the e-bike. The middle screenshot in figure 7.3 shows this message.

The user can access each characteristic separately. The buttons are spread according to the designated access possibilities of the characteristics. According to the access permissions of the characteristics, the read and notify accesses are possible without encrypt or authenticate the connection.

Sending a write request to a characteristic, the request will trigger the Bluetooth paring procedure. The paring is done to encrypt and authenticate the connection between the two devices. If the paring is successful the write is executed, in case of unsuccessful paring the write is aborted.

The paring procedure is controlled by the operating system, so the programmer has no influence to this procedure. Paring two mobile phones is done by the numeric comparison method. The possible paring modes are described in chapter 3.7.1.

One possibility to influence the paring mode is in case of an OOB paring. Here the key is transmitted via another communication channel and the programmer forwards the received key to the BLE paring procedure. This feature is not yet implemented in this application.

To give an overview over the class structure inside the second part of the application, figure 7.5 is introduced. The ControlActivity is the center of this application part. It is created by the MainActivity of the first part, if the user selects a BLE device from the list. The ControlAtivity creates a BLE_GATT_Service object with the data of the user selected device. This object is given to the BLE_GATT_ServiceConnection to establish a connection with the device.

The BLE_GATT_Callback is created at the same time to react on incoming BLE messages via the BLE_GATT_Service. The processing of the messages is decoupled from the message receiving by internal broadcasts. This is necessary due to the restricted access on the UI elements in Android. Only the UI thread is allowed to manipulate the screen content.

The BLE_GATT_BroadcastReceiver reacts on broadcasted messages. This component is executed by the UI thread, so it can manipulate the screen. The BroadcastReveicer interprets the message and uses the UIhandler to update the values.

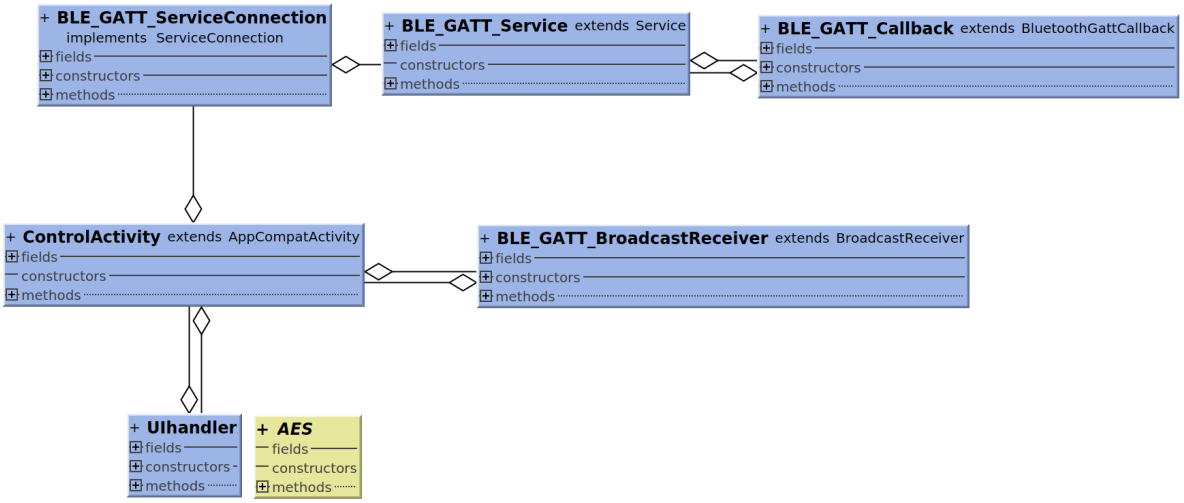


Figure 7.5: The class diagram of the second part of the control application, used to control the characteristics of the connected e-bike

The UIhandler uses the AES class to encrypt the CAN-message in the CAN-message characteristic. Since AES is a symmetric cypher, the key is the UTF-8 string 'theBestSecretKey' and the initialization vector is the current salt.

To clarify the interaction of the classes, two scenarios are explained:

- The user presses a button: The UIhandler forwards the action via the ControlActivity to the ServiceConnection. It sends the correspondent message via the BLE_GATT_Service to the connected device.
- The device receives a notify message: The BLE_GATT_Callback receives the message from the connected device. It extracts the UUID and the value of the characteristic from the notify message. The extracted data are broadcasted inside the application. The BLE_GATT_BroadcastReceiver receives the broadcast and updates the values of the characteristic via the UIhandler.

8 The experiment

In this chapter the applications introduced in the previous chapter are used to perform some experiments. In different situations their communication is eavesdropped. The recorded data is analyzed for the revealed information.

The eavesdropping is done with btlejack [Ries 18][Cauquil 18] in combination with micro:bit [Foundation 18] micro-controllers. The controllers have a internal BLE module, so they are ideal for this experiment. In total three micro:bits are used to observe all three advertising channels at the same time, so new established connections are always captured. Btlejack is a python program that was written by Damien Cauquil [Cauquil 18] to search for BLE vulnerabilities.

To evaluate the security of the designed BLE services, three experiments will be made:

1. Eavesdropping on an unencrypted connection while reading some characteristics.
2. Eavesdropping on an unencrypted connection that becomes an encrypted connection due to writing a characteristic.
3. Eavesdropping on an encrypted connection to compare the specified Ble2Can service with a modified version of the service.

8.1 Eavesdrop unencrypted reads

The goal of the first experiment is to verify the correct functionality of the devices and getting an overview how the raw data are arranged in lower layers of the BLE stack.

For this purpose the received values are logged in the development environment of the controlling device. At the same time btlejack recorded the traffic.

Table 8.1 shows the data that are received by btlejack. The first column contains the received raw link layer data in hex format. The column in the middle indicates which command belongs the received data. The last column holds a possible interpretation of the received data. The interpreted values are presented in tuples, containing the value in decimal representation and the number of bits considered for the interpretation. The number of the bits considered for each value is taken from the profile specification in chapter 6.3.2.

Table 8.1: The sniffed packets of btlejack

sniffed link layer data	command	interpreted values [considered bits]
0e 07 03 00 04 00 0a 33 00 06 10 0c 00 04 00 0b b3 00 36 01 00 00 25 00 1f 47	read request live ride characteristic read response	- - [56]; 179 [16]; 310 [32]; 37 [16]; 31 [8]; 71 [8]
0e 07 03 00 04 00 0a 33 00 06 10 0c 00 04 00 0b b3 00 44 02 00 00 20 00 1a 48	read request live ride characteristic read response	- - [56]; 179 [16]; 580 [32]; 32 [16]; 26 [8]; 72 [8]
02 07 03 00 04 00 0a 36 00 0a 0b 07 00 04 00 0b 00 00 00 00 33 00	read request live motor characteristic read response	- - [56]; 0 [16]; 0 [16]; 51 [8]; 0 [8]
02 07 03 00 04 00 0a 39 00 0a 0d 09 00 04 00 0b 64 00 00 30 00 32 ff ff	read request live battery characteristic read response	- - [56]; 100 [8]; 0 [16]; 48 [8]; 50 [8]; 65535 [16]
0e 07 03 00 04 00 0a 3f 00 06 10 0c 00 04 00 0b 75 04 00 00 00 01 00 00 64 01 00	read request diagnostic characteristic read response	- 1141 [32]; 256 [32] 100 [8]; 1 [16]

Table 8.2: The recorded values of the control application

action performed by the application	shown values
send read request for live ride characteristic receive read response	- 17,9 km/h; 0,31 km; 37 W; 31 Nm; 71 rpm
send read request for live ride characteristic receive read response	- 17,9 km/h; 0,58 km; 32 W; 26 Nm; 72 rpm
send read request for live motor characteristic receive read response	- 0 W; 0 Nm; 1 °C; 0 %
send read request for live battery characteristic receive read response	- 100 %; 0 mA; 48 V; 0 °C; 65535 km
send read request for diagnostic characteristic receive read response	- 1141 m; 256 sec; 100 %; 1 cycle

It has to be mentioned that BLE uses little-endian byte ordering for transmitted messages, which means the least significant byte is transmitted first, than the more significant byte is transmitted. This has an influence on the interpretation of the received data: In the second row of table 8.1, the bytes 10 to 13 are interpreted as a 32 bit value. The received bytes 0x 36 01 00 00 have to be rearranged to 0x 00 00 01 36 to interpret them as a common hex value. In this case the decimal value is 310.

The value can be interpreted as the second value of the live-ride-and-user characteristic, given in table 6.3. Correspondingly the traveled distance on this ride is 310 meters.

Comparing the sniffed communication with the logged data of the control application in table 8.2 it is obvious that the reconstructed interpretation of the sniffed value is correct. In the second row of the table the response of the read request is shown. The traveled distance is given with 0.31 km, which is equal to the 310 m calculated by the sniffed data.

When comparing further values of the tables, some differences on the temperature values occur. The sniffed data indicate a values of 50 and 51, while the logged data in the application are at 0 °C and 1 °C. Both values are correct. The specification for the temperature values in the live-motor and live- battery characteristic (tables 6.4 and 6.5) show that on both values a offset of -50 °C has to be applied. Regarding the offset the sniffed values match the logged data.

Table 8.3: The recorded packets of the unencrypted and encrypted communication

sniffed link layer data	command	interpreted values
0e 07 03 00 04 00 0a 33 00	read request live ride characteristic	-
06 10 0c 00 04 00 0b 90 00 39 00 00 00 27 00 29 3a	read response	144; 57; 39; 41; 58
06 09 05 00 04 00 01 12 46 00 05	write request on light characteristic	-
...	enabling encryption and the write response	-
02 0b 58 e2 d6 32 70 8d 94 96 84 45 20	encrypted read request on live ride characteristic	-
06 14 a8 10 77 42 99 da 14 77 b7 fe 4e f6 ca 12 ae 00 1f 7a 27 d1	encrypted read response	-
02 0b b4 68 cb b0 31 79 94 df eb a9 86	2nd encrypted read request on live ride characteristic	-
06 14 72 b7 d2 bf f3 20 28 ba 49 ba da 76 5d cf 48 18 ef 2a 87 30	2nd encrypted read response	-

This first experiment has shown that it is possible to eavesdrop a unencrypted connection and fully recover the information send by the bike.

8.2 Eavesdrop encrypted writes

In the second experiment, the encryption process is captured. In the beginning some read requests on an unencrypted connection are performed. In the next step, a write operation is requested and the connection becomes encrypted. The communicating devices use ECDH to exchange the keys and authenticate by using the Numeric-Comparison method, which was introduced in chapter 3.7.1. In the end some read requests on the encrypted connection are eavesdropped.

The complete eavesdropped communication is available in appendix 10.1. The following table 8.3 only contains an abstract of the communication.

The difference between the unencrypted and the encrypted communication is visualized in table 8.3. In the first two rows, messages of the unencrypted communication are shown. The read request is similar to the read requests of the previous experiment. Also the read response can be interpreted in the same scheme as in the previous experiment.

The third row holds the captured communication of the write request. This is also the start of the paring process. The next captured messages were used to exchange the encryption keys and authenticate the devices. These messages are abbreviated by the '...' in the fourth line, since there was a lot communication and it was not possible to find out the function of each packet.

In the fifth and seventh row, two encrypted read requests on the same characteristic are displayed. The first two bytes of the messages are identical. These bytes are the PDU header, which stays unencrypted. The rest of the Link Layer data are encrypted. The paring and the key exchange is done by the Security Manager, which can not be accessed easily by an application running in the Application Layer. So it was not possible to extract the encryption key via the Android application and the encrypted data stays confidential.

The sixth and eighth row show the encrypted answers that belong to the encrypted requests. Analogous to the encrypted requests, the PDU header is unencrypted while the rest of the message is unreadable.

Due to the lack of the encryption key, further investigations in the transmitted data are very hard and are skipped in this thesis. Nevertheless the experiment has shown that the PDU header stays unencrypted, while the PDU payload is encrypted automatically. As expected, the fifth and seventh row show that repeating requests that hold the same data are encrypted in different cyphers.

8.3 Eavesdrop the Ble2Can service

The goal of this experiment was to measure the impact of a refreshing salt on the security. This experiment consists of two setups: one setup with a changing salt, as specified in chapter 6.2, and one setup with a static salt.

In both setups the communication are captured. The commands are send in the same order, with the same arguments for the write operations.

The exchanged messages of the static salt setup are shown in table 8.4. In the beginning, the salt was read. This static salt is used for all following write accesses.

In the third row a read requests on the CAN-response is captured to get the initial value of this characteristic. The result is the encrypted UTF-8 string 'initial response'. The 16 bytes of the string in combination with one byte for the length, a four byte message integrity check (MIC) and a 7 byte header leads to the 28 bytes of the recorded message.

A first write request attempts to write the CAN-message characteristic with the response ID '123' and a raw message 'qwer'. The CAN-message consists of a raw message and the appended salt. In this case, the raw message consists of four UTF-8 characters. In combination with the 16 bytes of the salt, the raw message holds 20 bytes. The AES-CBC encryption of the emulator application encrypts the 20 byte message in a 32 byte cypher. The whole transmitted characteristic consists of 38 bytes: four bytes for the response ID, two bytes for the length of the message and 32 bytes of the cypher.

A normal write operation can transmit 23 byte of data, which is the default the maximum transmission unit (MTU). This is not enough for this characteristic with 38 bytes. So the characteristic is transfered via multiple consecutive write operations. This is called a long write. Each write operation of the long writes can handle 20 bytes of data. So the characteristic is split into chunks of 20 byte.

The first long write operation has a link layer packet size of 33 byte, the second operation has 31 byte. Each of the operations has an individual MIC of 4 bytes and headers

Table 8.4: The captured packets of the static salt setup

sniffed link layer data	command
0e 0b 7f e5 c9 f3 51 70 c4 04 ee 31 b3 06 19 f1 2f b8 1f e0 d3 70 14 57 ca cb ad b7 0a 57 94 a8 d0 8e 49 a1 39 43 87 25	read request on salt encrypted salt
0e 0b 31 de 2a 99 ca 03 5a 0b 34 63 a3 0a 1a 72 93 ec 1b d7 79 3d 74 6b c3 26 4b ce cf fd 42 e5 70 38 1c 3e a5 29 0b f2 c0	read request on initial CAN-response initial CAN-response
0e 1f 7a fc b2 78 8b 2a 48 ab 8e 64 72 51 61 61 a6 d4 b6 1a 6c d9 9c 57 3d fa c8 e2 ab 78 70 78 23 0a 09 55 1b d7 30 2d 7c 79 1f 62 02 1d c1 10 eb 57 6f 90 78 65 52 4d 21 f5 b0 89 6f 01 a9 13 ba f7 15 38 25 63 3d 35 db d0 a4 0a 09 28 27 d1 94 95 11 5e d1 c7	long write on CAN-message (1) request next part of long write long write on CAN-message (2) acknowledge long write
0e 0b 84 a6 d5 62 59 85 0a 37 4e 4e b9 06 17 5a 7b f8 20 5f c3 83 39 9d b5 dc bd a3 d5 22 87 47 a3 1d 60 be fa 2d	read on CAN-response read response
0e 1f 60 b7 35 c4 85 58 b3 ba 5b c0 1a 37 6b de 7c 7c 1d 8a 69 e2 4a 55 ec 73 63 c9 ef bb 4f f2 69 0a 09 bf 99 ec 80 7d fe f9 39 cf 02 1d 9b 15 00 cf 84 4b 50 26 55 98 24 ea 4d e0 cd 54 dc 1a 0c 17 84 c7 72 65 66 14 ed ed 19 0a 09 d8 c5 63 66 3b 83 3c 18 d7	2nd long write on CAN-message (1) request next part of long write 2nd long write on CAN-message (2) acknowledge long write
02 0b 8d 5d 22 b0 78 13 65 45 c6 a5 ce 0a 17 7b 41 55 e9 57 59 d1 94 c1 ff a8 c5 de 87 5a bf 0a da ee 95 bf 5c 74	2nd read on CAN-response 2nd read response

of 9 byte, which leads to effective payload of 20 and 18 byte. This is the size of the transfered characteristic.

The following read access on the CAN-response characteristic reveals a characteristic length of 25 byte. The response contains the response ID of the previous CAN-message write. The emulator creates a response with the UTF-8 string "myResponse" and the appended value of the response ID, in this case '123'. The 13 bytes of the response string, one byte for the length of the response string, four byte for the MIC and seven byte for the header are the 25 recorded byte.

The second write access to the CAN-message characteristic transports identical information, a response ID of '123' and a raw message of 'qwer'. Still the long-write operations are encrypted in different values. This is based on the AES-CCM encryption, described in chapter 3.7.3.

Also a second read operation on the CAN-response characteristic, that holds same data as the previous read, is encrypted with different values.

Table 8.5 holds the sniffed messages of the changing salt setup.

Similar to the previous setup, in the beginning the salt and the initial CAN-response was read. The following long-write of the CAN-message characteristic transmits the response ID '123' and the can-message 'qwer' to the peripheral. In this setup the peripheral creates a new salt, so the old salt was only valid for this one write access.

The read operation on the CAN-response contains the response UTF-8 string 'myResponse123' and leads to the 25 captured byte in the link layer.

As in the previous setup, the next operation attempts to write the CAN-message characteristic with the old salt. The data is fully transmitted via two long-write operations. But the message is rejected, as the following read of the CAN-response characteristic reveals.

The read operation returns a small data package of 12 byte. According to the specification, the response of an invalid write attempt is an empty string with a length of

Table 8.5: The recorded communication of the changing salt setup

sniffed link layer data	command
0e 0b aa 94 71 f3 66 69 11 c3 20 e7 a6 06 19 7e 9b a5 73 d4 90 5b c8 55 7f c6 e5 8b e1 88 c8 9a 2c af a5 dc 94 f5 70 91	read request on salt encrypted salt
02 0b 4d b2 4e eb 4b 38 31 cc 82 1c a9 0a 1a b3 27 16 33 3b 54 5b b1 98 a3 35 c3 19 3a 76 31 ae 15 a2 0e 0e f1 0d 9b 3d b6	read request on initial CAN-response initial CAN-response
02 1f 4c f9 bc a8 f4 d2 a9 ea 82 b2 bf 44 28 f8 63 a5 7f ad 55 86 d5 65 94 d3 30 2c 9b 87 b5 d6 ff 0a 09 6e 5b d3 aa ce 9e e6 de 2f 02 1d 4d 1f 9e 48 66 f3 fb 99 1b 94 a1 82 15 c9 f8 d0 7c a2 29 01 45 50 09 94 f6 7a 89 41 98 06 09 0c 1f 47 ff 28 0b ea d0 10	long write on CAN-message (1) request next part of long write long write on CAN-message (2) acknowledge long write
0e 0b 28 07 dd 7f 5c 2e 1d 98 b7 b1 54 0a 17 a3 7a 6c e9 5f 96 0e b0 80 77 5c b5 4c 0c 53 03 6e d0 71 66 58 7a fc	read on CAN-response read response
02 1f 1b 58 44 63 a9 8b 41 ad 42 23 e9 b3 61 b3 17 c7 f9 79 d9 f1 87 9a 0d b6 95 62 53 a6 dc 35 ef 0a 09 ba 8d 0e 78 41 68 19 e9 fb 02 1d f3 b9 4d a9 42 8c ed 84 07 3a ae 2f 49 d6 12 ee d9 cb 0c ec a7 f7 2d 86 85 ee 04 16 85 06 09 1c 91 6f 04 60 c0 0c c0 f2	2nd long write on CAN-message (1) request next part of long write 2nd long write on CAN-message (2) acknowledge long write
0e 0b 50 26 4d 51 eb 70 a6 8b 8f be 88 06 0a 09 84 bc 78 92 d4 5c 0d 28 1c	2nd read on CAN-response 2nd read response

zero. In this setup the characteristic is set to this empty string. So the received data packet consists of a 7 byte header, 1 byte that indicates the length of the response and four byte MIC.

The goal of this experiment was to examine if different salts have an influence on the encryption. The consecutive read and write operations in both setups contained the same values. Still they were encrypted to individual cyphers. This is based on the encryption algorithm. The AES-CCM encryption ensures confidentiality and authentication of the transfered data.

The usage of a static or dynamic salt has an influence on the behavior of the connected devices. Using a static salt, enables multiple consecutive CAN-message characteristic writes. A dynamic salt requires a update of the salt value after each CAN-message write access. This communication overhead is negligible, since the Ble2Can bridge is considered to be used during maintenance and not every day.

The different salts also influence the security of the encrypted CAN-message inside the characteristic. For encryption AES needs a secret key and an initialization vector (IV). In the Ble2Can bridge the transmitted CAN-messages should be independent from each other, so the IV is based on the salt. Encrypting equal messages with a static salt always returns the same cypher. This weakness could be used by an attacker to find an encryption for an arbitrary CAN message. To compensate this weakness, the changing salt was introduced. An attacker will have only one guess to find a cypher, than the salt changes and previous attempts of the attacker are invalid.

9 Conclusion

In this master thesis a new e-bike control interface based on Bluetooth Low Energy (BLE) is designed.

For this purpose, the power consumption of different wireless communication technologies was compared. High data rate technologies as WiFi or UWB are the most efficient technologies, regarding the power per bit. Regarding the overall consumed power, ANT and BLE are the most economical technologies. With attention to the small data rates that are necessary to communicate with an e-bike, the power saving technologies are more suitable for this task.

Considering robustness and encryption of the communication technologies, BR/EDR Bluetooth and BLE are the best candidates in the low energy sector.

With BLE as the favorite, the architecture of this standard was analyzed to use the full potential of this technology. One of the considered layers was the security manager, that is responsible for the key exchange and the authentication, which is implemented in different paring methods.

Some of the paring methods allow an attacker to perform man-in-the-middle attacks or to recover the exchanged keys and violate the confidentiality and integrity of the communication. Those attacks are considered and counter measures were introduced.

Based on the attacks and the counter measures the security requirements for the new communication interface were derived. Various companies from the bicycle industry provided the functional requirements. Also the legal requirements of the DIN-EN-15194 standard are considered.

Based on the captured requirements, the e-bike profile was designed. The core of the profile is the e-bike service, which provides access to the sensor data of the e-bike and enables the control of the electrical support engine. An additional Ble2Can service enables a comfortable access to the internal CAN bus of the bike to control the internal settings of the bike. This service is protected by an additional encryption, that regulates the access to the CAN bus. The two designed services are surrounded by several Bluetooth SIG adapted services to share informations about the bike and exchange logging informations.

The new designed services are implemented in an e-bike emulator application and a control application. The two applications were introduced to demonstrate the functionality of the services and enable some measurements on the transmitted data.

The measurements have shown that it is quiet easy to eavesdrop a BLE connection and recover the informations on an unencrypted communication. The specification requires an encrypted and authenticated link for critical information to protect against manipulation. The experiments have shown it is possible to gather metadata and guess which type of information was transmitted, based on the size of the transmitted data package. The content of the data package was protected by the encryption.

It is aspired to implement the designed services in e-bike components and spread the profile in the bicycle industry. For an implementation a further validation of the functional safety is necessary. A reliability analysis of the whole start-up assistance process has to be made. Also the Bluetooth stack and the probability of a false write transmission has to be considered in this analysis. Constructing e-bikes with a wireless interface, creates new challenges in development and new possibilities of interaction with them.

Bibliography

[Al Kalaa 16] M. O. Al Kalaa, W. Balid, N. Bitar, H. H. Refai, "Evaluating bluetooth low energy in realistic wireless environments", in *2016 IEEE Wireless Communications and Networking Conference*, IEEE. 2016, pp. 1–6.

[Apple 13] Apple, "Core Bluetooth Programming Guide: About Core Bluetooth", Website, 2013. Online available at `https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html`; on 31st March 2019.

[Aven 09] T. Aven, "Identification of safety and security critical systems and activities", *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 404–411, 2009.

[Barcena 15] M. B. Barcena, C. Wueest, "Insecurity in the Internet of Things", *Security Response, Symantec*, 2015.

[BluetoothSIG 04] BluetoothSIG, "Bluetooth Core Specification Version 2.0", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2004.

[BluetoothSIG 10] BluetoothSIG, "Bluetooth Core Specification Version 4.0", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2010.

[BluetoothSIG 13] BluetoothSIG, "Bluetooth Core Specification Version 4.1", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2013.

[BluetoothSIG 14] BluetoothSIG, "Bluetooth Core Specification Version 4.2", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2014.

[BluetoothSIG 16] BluetoothSIG, "Bluetooth Core Specification Version 5.0", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2016.

[BluetoothSIG 17] BluetoothSIG, "Mesh Working Group of the Bluetooth Special Interest Group: Bluetooth Mesh Device Properties 1.0", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2017.

[BluetoothSIG 19] BluetoothSIG, "Bluetooth Core Specification Version 5.1", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 2019.

[BluetoothSIG 99] BluetoothSIG, "Bluetooth Core Specification Version 1.0", in *Specification of the Bluetooth System*, Bluetooth Special Interest Group. 1999.

[Brauer 16] S. Brauer, A. Zubow, S. Zehl, M. Roshandel, S. Mashhadi-Sohi, "On practical selective jamming of Bluetooth Low Energy advertising", in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, IEEE. 2016, pp. 1–6.

[Cauquil 18] D. Cauquil, "BtleJack: a new Bluetooth Low Energy swiss-army knife", Online, GitHub, 2018. Available at https://github.com/virtualabs/btlejack; on 18th June 2019.

[Chóliz 11] J. Chóliz, Á. Hernández-Solana, A. Sierra, P. Cluzeaud, "Coexistence of MB-OFDM UWB with Impulse Radio UWB and other radio systems", in *2011 IEEE International Conference on Ultra-Wideband (ICUWB)*, IEEE. 2011, pp. 410–414.

[Cyr 14] B. Cyr, W. Horn, D. Miao, M. Specter, "Security analysis of wearable fitness devices (fitbit)", *Massachusets Institute of Technology*, p.1̃, 2014.

[Dementyev 13] A. Dementyev, S. Hodges, S. Taylor, J. Smith, "Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario", in *2013 IEEE International Wireless Symposium (IWS)*, IEEE. 2013, pp. 1–4.

[DIN-NASport 18] DIN-NASport, "DIN-Normenausschuss Sport- und Freizeitgerät (NASport): DIN-EN-15194 Fahrräder - Elektromotorisch unterstützte Räder - EPAC", November 2018.

[Domínguez 12] F. Domínguez, A. Touhafi, J. Tiete, K. Steenhaut, "Coexistence with WiFi for a home automation ZigBee product", in *2012 19th IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, IEEE. 2012, pp. 1–6.

[El-Hadary 14] H. El-Hadary, S. El-Kassas, "Capturing security requirements for software systems", *Journal of advanced research*, vol. 5, no. 4, pp. 463–472, 2014.

[Fan 17] X. Fan, F. Susan, W. Long, S. Li, "Security analysis of zigbee", 2017.

[Foundation 18] M. E. Foundation, "microbit.org", Online, 2018. Available at https://microbit.org/; on 18th June 2019.

[Fredriksson 17] T. Fredriksson, N. Ljungberg, "Security in low power wireless networks: Evaluating and mitigating routing attacks in a reactive, on demand ad-hoc routing protocol", 2017.

[Georgakakis 10] E. Georgakakis, S. A. Nikolidakis, D. D. Vergados, C. Douligeris, "An analysis of bluetooth, zigbee and bluetooth low energy and their use in wbans", in *International Conference on Wireless Mobile Communication and Healthcare*, Springer. 2010, pp. 168–175.

[Gharghan 15] S. Gharghan, R. Nordin, M. Ismail, "An ultra-low power wireless sensor network for bicycle torque performance measurements", *Sensors*, vol. 15, no. 5, pp. 11741–11768, 2015.

[Google 13] Google, "Android Developers: Jelly Bean", Website, 2013. Online available at https://developer.android.com/about/versions/jelly-bean.html; on 31st March 2019.

[Healy 09] M. Healy, T. Newe, E. Lewis, "Security for wireless sensor networks: A review", in *Sensors Applications Symposium, 2009. SAS 2009. IEEE*, IEEE. 2009, pp. 80–85.

[Herrero 15] J. R. Herrero, G. Villarrubia, A. L. Barriuso, D. Hernández, Á. Lozano, M. A. D. L. S. GONZÁLEZ, "Wireless controller and smartphone based interaction system for electric bicycles.", *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 4, no. 4, pp. 59–68, 2015.

[Hu 04] F. Hu, J. Ziobro, J. Tillett, N. K. Sharma, "Secure wireless sensor networks: Problems and solutions", *Rochester Institute of Technology, Rochester, New York, USA*, 2004.

[Huang 10] J. Huang, G. Xing, G. Zhou, R. Zhou, "Beyond co-existence: Exploiting WiFi white space for Zigbee performance assurance", in *The 18th IEEE International Conference on Network Protocols*, IEEE. 2010, pp. 305–314.

[Kamath 10] S. Kamath, J. Lindh, "Measuring bluetooth low energy power consumption", *Texas instruments application note AN092, Dallas*, 2010.

[Katz 10] F. H. Katz, "WPA vs. WPA2: Is WPA2 Really an Improvement on WPA?", in *2010 4th Annual Computer Security Conference (CSC 2010), Coastal Carolina University, Myrtle Beach, SC*. 2010.

[Kiefer 16] C. Kiefer, F. Behrendt, "Smart e-bike monitoring system: real-time open source and open hardware GPS assistance and sensor data for electrically-assisted bicycles", *IET Intelligent Transport Systems*, vol. 10, no. 2, pp. 79–88, 2016.

[Ko 10] M. Ko, D. L. Goeckel, "Wireless physical-layer security performance of UWB systems", in *2010-MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*, IEEE. 2010, pp. 2143–2148.

[Köppel 13] S. Köppel, "Bluetooth jamming", *ETH Zürich, Switzerland*, 2013.

[Lee 07] J.-S. Lee, Y.-W. Su, C.-C. Shen et al, "A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi", *Industrial electronics society*, vol. 5, pp. 46–51, 2007.

[Li 10] H. Li, Z. Jia, X. Xue, "Application and analysis of ZigBee security services specification", in *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 2, IEEE. 2010, pp. 494–497.

[Mehmood 15] N. Q. Mehmood, R. Culmone, "An ANT+ protocol based health care system", in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, IEEE. 2015, pp. 193–198.

[O'Sullivan 15] H. O'Sullivan, "Security Vulnerabilities of Bluetooth Low Energy Technology (BLE)", *Tufts University*, 2015.

[Padgette 12] J. Padgette, K. Scarfone, L. Chen, "Guide to bluetooth security", *NIST special publication*, vol. 800, no. 121, p.5̃, 2012.

[Premnath 08] S. N. Premnath, S. K. Kasera, "Battery-Draining-Denial-of-Service Attack on Bluetooth Devices", *Thanks to*, p.3̃, 2008.

[Rao 11] L. Rao, "Sexual activity tracked by Fitbit shows up in Google search results", Online, 2011. Available at https://techcrunch.com/2011/07/03/sexual-activity-tracked-by-fitbit-shows-up-in-google-search-results/; on 30th May 2019.

[Ries 18] U. Ries, "Btlejack: Neues Gratis-Tool zum Belauschen von Bluetooth-Verbindungen", Online, heise.de, 2018. Available at https://www.heise.de/security/meldung/Btlejack-Neues-Gratis-Tool-zum-Belauschen-von-Bluetooth-Verbindungen-4134142.html; on 18th June 2019.

[Rowan 10] T. Rowan, "Negotiating wifi security", *Network Security*, vol. 2010, no. 2, pp. 8–12, 2010.

[Ryan 13] M. Ryan et al, "Bluetooth: With Low Energy Comes Low Security.", *WOOT*, vol. 13, pp. 4–4, 2013.

[Sahinaslan 19] O. Sahinaslan, "Encryption protocols on wireless IoT tools", in *AIP Conference Proceedings*, vol. 2086, no. 1, AIP Publishing. 2019, p. 030036.

[Sampath 07] A. Sampath, H. Dai, H. Zheng, B. Y. Zhao, "Multi-channel jamming attacks using cognitive radios", in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, IEEE. 2007, pp. 352–357.

[Schmitt 15] J. Schmitt, "Network Security", 2015. University of Kaiserslautern.

[Schmitt 16] J. Schmitt, "Protocols and Algorithms for Network Security", 2016. University of Kaiserslautern.

[Sivakumaran 18] P. Sivakumaran, J. Blasco Alis, "A Low Energy Profile: Analysing Characteristic Security on BLE Peripherals", in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, ACM. 2018, pp. 152–154.

[Smith 11] P. Smith, "Comparisons between low power wireless technologies", *US Patent CS-213199-AN*, 2011.

[Tews 09] E. Tews, M. Beck, "Practical attacks against WEP and WPA", in *Proceedings of the second ACM conference on Wireless network security*, ACM. 2009, pp. 79–86.

[Vanhoef 17] M. Vanhoef, F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM. 2017, pp. 1313–1328.

[Vanhoef 19] M. Vanhoef, E. Ronen, "Dragonblood: A Security Analysis of WPA3's SAE Handshake.", *IACR Cryptology ePrint Archive*, vol. 2019, p. 383, 2019.

[ZIV 13a] ZIV, "Zweirad Industrie Verband Wirtschaftspressekonferenz am 08. März 2016 in Berlin Zahlen - Daten - Fakten zum Fahrradmarkt in Deutschland 2015", Website, 2013. Online available at https://www.ziv-zweirad.de/fileadmin/redakteure/Downloads/Marktdaten/PK-2016_08-03-2016_Praesentation.pdf; on 30th May 2019.

[ZIV 13b] ZIV, "Zweirad Industrie Verband Wirtschaftspressekonferenz am 20. März 2013 in Berlin Zahlen - Daten - Fakten zum Fahrradmarkt in Deutschland", Website, 2013. Online available at `https://www.ziv-zweirad.de/fileadmin/redakteure/Downloads/Marktdaten/PK_2013-ZIV_Praesentation_20-03-2013_oT.pdf`; on 30th May 2019.

[ZIV 19] ZIV, "Zweirad Industrie Verband Wirtschaftspressekonferenz am 21. März 2019 in Berlin Zahlen - Daten - Fakten zum Fahrradmarkt in Deutschland 2018", Website, 2019. Online available at `https://www.ziv-zweirad.de/fileadmin/redakteure/Downloads/PDFs/PK-2019_21-03-2019_Praesentation.pdf`; on 30th May 2019.

10 Appendix

10.1 Complete eavesdropped communication of the second experiment

```
[i] Got CONNECT_REQ packet from 55:6f:3b:36:42:97 to 6d:d5:17:7e:19:63
 |-- Access Address: 0xb142c518
 |-- CRC Init value: 0x2c9fee
 |-- Hop interval: 36
 |-- Hop increment: 15
 |-- Channel Map: 0x1fffffffff
 |-- Timeout: 5000 ms

LL Data: 03 09 08 3f 00 00 00 00 00 00 00
LL Data: 0f 09 09 3f 00 00 00 00 00 00 00
LL Data: 0b 09 09 3f 00 00 00 00 00 00 00
LL Data: 03 09 14 7b 00 48 04 64 00 90 03
LL Data: 03 09 14 7b 00 48 04 64 00 90 03
LL Data: 0f 09 15 7b 00 48 04 64 00 90 03
LL Data: 0b 06 0c 08 1d 00 5a 02
LL Data: 03 06 0c 08 1d 00 5a 02
LL Data: 00 90 03 7b 00 48 04 64 00 90 03 17 d5 6d 18 c5 42 b1 ee 9f 2c 02 21 00 24 00 00 00 f4 01 ff ff ff ff 1f 0f 40 41
         34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
         00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
LL Data: 07 09 15 7b 00 48 04 64 00 90 03
LL Data: 0f 18 0f 06 00 06 00 00 00 f4 01 00 0b 00 00 00 01 00 02 00 03 00 04 00 05 00
LL Data: 02 0b 07 00 04 00 10 01 00 ff ff 00 28
LL Data: 0f 0c 00 02 05 00 06 00 00 00 f4 01 0e 00
LL Data: 01 18 14 00 1c 00 00 18 01 00 05 00 0e 00 00 00 01 00 02 00 03 00 04 00 05 00
LL Data: 02 0b 07 00 04 00 10 1d 00 ff ff 00 28
LL Data: 0a 1a 16 00 04 00 11 14 28 00 30 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 00 01 00 00          (Ble2Can service UUID)
LL Data: 02 0b 07 00 04 00 10 31 00 ff ff 00 28
LL Data: 0a 1a 16 00 04 00 11 14 31 00 ff ff 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 00 00 00 00           (e-bike service UUID)
LL Data: 02 0b 07 00 04 00 08 01 00 05 00 02 28
LL Data: 0b 09 14 7b 00 48 04 64 00 90 03
LL Data: 03 09 15 7b 00 48 04 64 00 90 03
LL Data: 06 09 05 00 04 00 01 08 01 00 0a
LL Data: 0f 09 14 7b 00 48 04 64 00 90 03
LL Data: 02 0b 07 00 04 00 08 01 00 05 00 03 28
LL Data: 07 09 15 7b 00 48 04 64 00 90 03
LL Data: 0a 0d 09 00 04 00 09 07 02 00 20 03 00 05 2a
LL Data: 02 0b 07 00 04 00 08 03 00 05 00 03 28
LL Data: 0a 09 05 00 04 00 01 08 03 00 0a
LL Data: 02 09 05 00 04 00 04 04 00 05 00
LL Data: 0a 09 05 00 04 00 01 04 04 00 0a
LL Data: 02 0b 07 00 04 00 08 14 00 1c 00 02 28
LL Data: 0a 09 05 00 04 00 01 08 14 00 0a
LL Data: 02 0b 07 00 04 00 08 14 00 1c 00 03 28
LL Data: 0a 1b 17 00 04 00 09 07 15 00 02 16 00 00 2a 17 00 02 18 00 01 2a 19 00 02 1a 00 a6 2a
LL Data: 02 0b 07 00 04 00 08 1a 00 1c 00 03 28
LL Data: 0a 09 05 00 04 00 01 08 1a 00 0a
LL Data: 02 09 05 00 04 00 04 1b 00 1c 00
LL Data: 0a 09 05 00 04 00 01 04 1b 00 0a
LL Data: 02 0b 07 00 04 00 08 28 00 30 00 02 28
LL Data: 0a 09 05 00 04 00 01 08 28 00 0a
LL Data: 02 0b 07 00 04 00 08 28 00 30 00 03 28
LL Data: 0a 1b 17 00 04 00 09 15 29 00 0a 2a 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 01 01 00 0     (Ble2Can characteristics)
LL Data: 02 0b 07 00 04 00 08 2a 00 30 00 03 28
LL Data: 0a 1b 17 00 04 00 09 15 2b 00 22 2c 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 02 01 00 00
LL Data: 02 0b 07 00 04 00 08 2c 00 30 00 03 28
LL Data: 0a 1b 17 00 04 00 09 15 2e 00 02 2f 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 03 01 00 00
LL Data: 02 0b 07 00 04 00 08 2f 00 30 00 03 28
LL Data: 0a 09 05 00 04 00 01 08 2f 00 0a
LL Data: 02 09 05 00 04 00 04 2d 00 2d 00
LL Data: 0a 0a 06 00 04 00 05 01 2d 00 02 29
LL Data: 02 09 05 00 04 00 04 30 00 30 00
LL Data: 0a 0a 06 00 04 00 05 01 30 00 02 29
LL Data: 02 0b 07 00 04 00 08 31 00 ff ff 02 28
```

```
LL Data: 0a 09 05 00 04 00 01 08 31 00 0a
LL Data: 02 0b 07 00 04 00 08 31 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 32 00 12 33 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 01 00 00 00     (e-bike characteristics)
LL Data: 02 0b 07 00 04 00 08 33 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 35 00 12 36 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 02 00 00 00
LL Data: 02 0b 07 00 04 00 08 36 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 38 00 12 39 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 03 00 00 00
LL Data: 02 0b 07 00 04 00 08 39 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 3b 00 12 3c 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 04 00 00 00
LL Data: 02 0b 07 00 04 00 08 3c 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 3e 00 02 3f 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 05 00 00 00
LL Data: 02 0b 07 00 04 00 08 3f 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 40 00 02 41 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 06 00 00 00
LL Data: 02 0b 07 00 04 00 08 41 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 42 00 1a 43 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 07 00 00 00
LL Data: 02 0b 07 00 04 00 08 43 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 45 00 1a 46 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 08 00 00 00
LL Data: 02 0b 07 00 04 00 08 46 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 48 00 12 49 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 09 00 00 00
LL Data: 02 0b 07 00 04 00 08 49 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 4b 00 02 4c 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 0a 00 00 00
LL Data: 02 0b 07 00 04 00 08 4c 00 ff ff 03 28
LL Data: 0a 1b 17 00 04 00 09 15 4d 00 0a 4e 00 77 64 d0 c0 ef d0 d7 b9 99 44 ab ab 0b 00 00 00
LL Data: 02 0b 07 00 04 00 08 4e 00 ff ff 03 28
LL Data: 0a 09 05 00 04 00 01 08 4e 00 0a
LL Data: 02 09 05 00 04 00 04 34 00 34 00
LL Data: 0a 0a 06 00 04 00 05 01 34 00 02 29
LL Data: 02 09 05 00 04 00 04 37 00 37 00
LL Data: 0a 0a 06 00 04 00 05 01 37 00 02 29
LL Data: 02 09 05 00 04 00 04 3a 00 3a 00
LL Data: 0a 0a 06 00 04 00 05 01 3a 00 02 29
LL Data: 02 09 05 00 04 00 04 3d 00 3d 00
LL Data: 0a 0a 06 00 04 00 05 01 3d 00 02 29
LL Data: 02 09 05 00 04 00 04 44 00 44 00
LL Data: 0a 0a 06 00 04 00 05 01 44 00 02 29
LL Data: 02 09 05 00 04 00 04 47 00 47 00
LL Data: 0a 0a 06 00 04 00 05 01 47 00 02 29
LL Data: 02 09 05 00 04 00 04 4a 00 4a 00
LL Data: 0a 0a 06 00 04 00 05 01 4a 00 02 29
LL Data: 02 09 05 00 04 00 04 4f 00 ff ff
LL Data: 0a 09 05 00 04 00 01 04 4f 00 0a
LL Data: 03 18 0f 24 00 24 00 00 00 f4 01 00 5d 00 00 00 01 00 02 00 03 00 04 00 05 00
LL Data: 0b 18 10 24 00 24 00 00 00 f4 01 00 5e 00 00 00 01 00 02 00 03 00 04 00 05 00
LL Data: 03 0c 00 02 0b 00 24 00 00 00 f4 01 5f 00
LL Data: 07 09 14 7b 00 48 04 64 00 90 03
LL Data: 0f 09 15 7b 00 48 04 64 00 90 03
LL Data: 03 09 14 7b 00 48 04 64 00 90 03
LL Data: 0b 09 15 7b 00 48 04 64 00 90 03

LL Data: 0e 07 03 00 04 00 0a 33 00                                          (read request live ride characteristic (unencrypted))
LL Data: 06 10 0c 00 04 00 0b 90 00 39 00 00 00 27 00 29 3a 6d                                  (read response (unencrypted))
LL Data: 02 07 03 00 04 00 0a 33 00                                  (2nd read request live ride characteristic (unencrypted))
LL Data: 0a 10 0c 00 04 00 0b 95 00 42 00 00 00 2e 00 2e 3b 6e                                  (2nd read response (unencrypted))
LL Data: 0e 07 03 00 04 00 0a 36 00                                    (read request live motor characteristic (unencrypted))
LL Data: 06 0b 07 00 04 00 0b 00 00 00 00 32 00                                 (read response (unencrypted))
LL Data: 02 07 03 00 04 00 0a 39 00                                  (read request live battery characteristic (unencrypted))
LL Data: 0a 0d 09 00 04 00 0b 18 00 00 30 00 32 00 00                           (read response (unencrypted))

LL Data: 0e 07 03 00 04 00 0a 46 00                                    (read request light info characteristic (unencrypted))
LL Data: 06 09 05 00 04 00 0b 00 00 00 00
LL Data: 0e 0b 07 00 04 00 12 46 00 00 00 00 00 00                              (read response (unencrypted))
LL Data: 06 09 05 00 04 00 01 12 46 00 05      (write request light information characteristic (unencrypted, paring starts))

LL Data: 0f 18 0f 06 00 06 00 00 00 f4 01 00 d1 02 00 00 01 00 02 00 03 00 04 00 05 00
LL Data: 02 0b 07 00 06 00 01 04 00 2d 10 0f 0f
LL Data: d2 02 00 00
LL Data: 0f 0c 00 02 05 00 06 00 00 00 f4 01 d3 02
LL Data: 0b 09 14 7b 00 48 04 64 00 90 03
LL Data: 03 09 15 7b 00 48 04 64 00 90 03
LL Data: 0f 09 14 7b 00 48 04 64 00 90 03
LL Data: 07 09 15 7b 00 48 04 64 00 90 03
LL Data: 0a 0b 07 00 06 00 02 04 00 0d 10 0f 0f
LL Data: 0e 45 41 00 06 00 0c eb 65 67 b5 e0 b9 dd 9d 6d f8 d8 43 aa c4 61 b3 67 2c c2 c5 35 b6 6d 36 2b 95 b7 65 70 a2 76
         d2 52 5f af ef b6 d1 31 58 f0 cc 99 9f c6 ea f4 35 88 4b ce 9e 73 31 0f eb 76 0f 22 fd c4 6a 6c e6
LL Data: 0a 45 41 00 06 00 0c 94 64 a5 5e cd f8 97 0a 49 71 c8 1b 70 55 bd 81 f0 7c c5 fe 8a 82 c8 5a 74 e7 4d 5d d9 6a dc
         34 25 5b 61 5f 85 da c3 d0 55 81 3f 4a 2c 72 ef 42 c7 8e 00 11 b1 e0 35 c3 8c d3 2e 7e 81 3c 7d c3
LL Data: 0a 15 11 00 06 00 03 db d5 fc 6f da cc c5 cd 02 8d b5 5a 53 4e 55 68
LL Data: 02 15 11 00 06 00 04 24 b9 73 a3 a4 d8 75 79 27 a0 df a6 7a a5 98 ff
LL Data: 0a 15 11 00 06 00 04 eb 9e 66 32 4e 7e df db b5 5d 5f 94 90 cb eb 7e
LL Data: 0e 15 11 00 06 00 0d cc 2b 2b 00 33 64 2e 67 68 62 37 11 bd 3a a3 10
LL Data: 0a 15 11 00 06 00 0d f1 17 22 a0 9e 69 6a c6 01 24 1b 70 46 24 51 6d
LL Data: 0f 17 03 00 00 00 00 00 00 00 00 00 00 79 7e 60 ce cb 06 f0 71 f8 ff ea 0e
LL Data: 07 0d 04 72 05 52 14 3a 76 88 84 ca 1d 46 5b
LL Data: 0b 01 05
LL Data: 03 05 d4 e3 30 84 47
LL Data: 0b 05 52 15 7a b2 f4
LL Data: 16 19 30 1d 3d 7d 86 9b d4 da 26 d6 1e 72 d0 47 d0 88 87 e6 61 69 70 22 d8 12 7c
LL Data: 1a 10 12 b2 ff d1 16 9d 31 a6 fe 74 1f 1d e2 af f0 2a
LL Data: 06 19 67 cd 2f d1 2d d8 e1 f1 ef 02 3e 77 56 28 f3 6a 79 91 2f 46 30 ab 49 42 29
LL Data: 1e 19 8e 6f 0e 24 bf a9 84 4a 21 44 74 13 38 bb 4c 32 ca 4f a3 0e 89 1b 2d a1 ba
```

```
LL Data: 02 10 3a a5 f7 80 41 5b 3d c8 87 63 e3 d8 d8 f3 0e 80
LL Data: 0e 19 2e a7 af 05 d3 a4 08 0c 0c 42 b4 39 c0 41 75 48 c6 82 ef 95 f4 3f 2b 35 ec
LL Data: 02 0f 0d 06 0c 6c 03 f6 d7 cc 9e d3 3d 07 0a b5 11
LL Data: 0b 0d bc b8 7e 7e 11 9c 94 b5 1c d8 ed b6 e7
LL Data: 03 0d c1 dd b2 02 a7 29 2e 86 f0 bc ce fe b4
LL Data: 03 0d 7c 70 ab 31 98 15 d8 f4 de 3a 0c bf 6d
LL Data: 0b 0d 62 0d c8 65 fa 29 53 82 57 a7 fe 46 c1
LL Data: 0a 09 5b 6b 13 ff 02 c8 46 18 e2
LL Data: 02 0f e8 ef 30 f7 3c 62 e0 8b 8d db 04 54 d7 40 7e
LL Data: 0a 16 09 5c fe 49 00 3f b9 54 c1 a5 24 e1 a6 4f 6a d9 fd d8 85 9e 62 ad
LL Data: 02 0f aa 65 92 2a f1 b8 85 d7 a8 7f ba 73 24 ef 21
LL Data: 07 0d 85 82 93 35 4c 89 39 83 06 2d e3 40 4e
LL Data: 0f 0d 45 56 c8 6d 3c 4a de 14 04 04 7c b5 51
LL Data: 3a 57 66 8e b7 ed 7f c9 27 08 cb 77 e1 65 35 b4 a5 18 e7 d8 85 9e 62 ad 2b 35 ec 8a 82 c8 5a 74 e7 4d 5d d9 6a dc
         34 25 5b 61 5f 85 da c3 d0 55 81 3f 4a 2c 72 ef 42 c7 8e 00 11 b1 e0 35 c3 8c d3 2e 7e 81 3c 7d c3 00 00 00 00 00
         00 00 00 00 00 00 00 00 00 00 00 00 00 00
LL Data: 02 0f 1d 23 53 69 fd ff f8 76 ed ec 77 02 99 46 36
LL Data: 0a 1e e3 e5 82 0f 63 6d ef c1 61 ad de 79 87 cf f1 26 43 27 a6 9d 00 86 68 d4 1e 03 2f 14 e2 a8
LL Data: 02 0f fc da 03 0b f0 b2 cc 87 60 bf 58 97 fd bb 79
LL Data: 0a 0d 40 c5 60 6b af 83 41 cc 58 78 cd 79 45
LL Data: 02 0f 7b 93 74 5c 71 4c ff 5e 3c 86 e8 54 3b e1 39
LL Data: 0a 11 49 8f 9a a3 b0 a0 06 2b 27 08 bf 90 5c 3d d1 c8 83
LL Data: 02 0f a0 22 d2 43 5a 06 32 61 f2 54 f8 21 83 28 a0
LL Data: 0a 0d 46 dd 0c b1 90 70 d8 16 40 82 40 23 f5
LL Data: 02 0d 9f be 8a 61 49 fd 5e 66 fa f5 ae 55 fa
LL Data: 0a 0d a6 3d bb 89 de f1 1b b5 ce 7b 44 85 b1
LL Data: 02 0f 3f 6b 7a bf d6 07 e5 3d 98 45 dd a7 f9 40 0b
LL Data: 0a 0d 08 91 d9 1a 04 4a 40 2a a2 cf 15 33 d7
LL Data: 02 0f a3 9e 98 7f 43 26 66 ca c6 08 84 ff fc a9 bb
LL Data: 0a 1f a6 c3 39 91 7f 0c cc 78 5e 33 a1 24 14 a1 26 67 09 06 bc 58 40 77 d1 11 be ba dd 7c 7e c8 11
LL Data: 02 0f 8b 4f 1e 07 3e 19 5e 3d 74 22 a4 4b 66 a8 5e
LL Data: 0a 0d fc 61 ad 1a 9a a7 d7 b7 0a 31 bc 9f 80
LL Data: 02 0d 12 e8 fa 1a 43 c3 cb dc 86 21 36 7d ed
LL Data: 0a 0d bf 73 ce fd 72 49 19 e0 ff 82 69 b3 7d
LL Data: 02 0f f3 46 ea e0 c9 a2 c6 46 f3 4a 5a 32 a6 5f 90
LL Data: 0a 0d 8e fb 26 4c fb 84 31 e0 ec 2e 82 93 3b
LL Data: 02 0f ac b6 a6 d5 d5 24 38 27 dc 99 67 9c ee e4 09
LL Data: 0a 1f a7 a6 37 04 11 0f f2 7a 6a 86 13 39 87 75 20 cb ce f8 ee 38 3f 3e cc 9b 38 17 ab 69 94 a6 f1
LL Data: 02 0f ae cf c1 25 cf 79 27 64 7c 8c 3d 4c 39 da b1
LL Data: 0a 1f 24 98 cf fc 0b ac 09 0c 56 90 94 c7 49 31 d5 a7 f8 7a 34 f7 d9 4c 1b 53 3a 3f 10 15 27 55 30
LL Data: 02 0f 8a d3 e8 54 51 75 70 96 17 84 64 d6 ca a0 af
LL Data: 0a 1f 7d a8 32 05 d4 6d 64 d9 77 d6 ab dc 5e bc 5d 8a 76 5e 46 64 e4 b7 b6 b8 f6 90 5f 74 63 aa 75
LL Data: 02 0f ca ed 35 32 3e 0a 1d fd c2 52 76 97 53 0a c5
LL Data: 0a 0d 7f c1 36 5e db 4d 5f 1f 81 91 7d 68 ca
LL Data: 02 0d 4d f7 0b 45 2f 29 db a2 49 7c 68 50 91
LL Data: 0a 0e b6 e0 3e 74 70 50 f6 6c 38 68 2b 9a 20 b9
LL Data: 02 0d 95 0d 74 14 03 4b 36 2f 70 51 95 d2 00
LL Data: 0a 0e ec 87 60 f8 1e e2 43 3a c7 d1 f3 77 c7 71
LL Data: 02 0f 1f b7 37 b3 14 8e 57 6f 42 3e b5 ce 1e a1 71
LL Data: 0a 0d bc cd b4 5f 18 eb 93 eb 1d af fc 8f b2
LL Data: 02 0f 66 9a 56 f5 73 17 00 60 c0 94 75 5b c1 94 71
LL Data: 0a 1f cf a0 3f b2 c8 3f 2a d1 0a 94 3b 4c 08 c4 67 7e c7 14 92 a0 85 eb 7f 14 31 f0 48 a1 e4 3c 9b
LL Data: 02 0f a2 15 a4 83 30 d6 dd 4b e7 0f 9d a3 25 cb 57
LL Data: 0a 1f 5b 0e 5f 89 eb e6 19 11 ed 3b 80 7a 78 f0 38 90 b5 65 6d 6c 93 f9 ba 04 81 9a 7d cf 8a b7 14
LL Data: 02 0f 43 59 2a 3e d5 ae bd 42 61 60 3a f8 4b 3a d8
LL Data: 0a 1f 67 1e f4 e1 7f c4 81 c2 1c 95 cb 7e 04 05 a6 d0 97 5b cb 3d d2 82 48 7d b4 70 60 39 77 0b a1
LL Data: 02 0f 44 e8 4b 09 02 ca 48 bb 33 17 c2 e9 67 cb 04
LL Data: 0a 1f f5 d5 87 a9 98 bb 98 ae 47 69 c2 a3 17 00 a2 a2 8b 79 06 d3 ec 71 3a dd 42 35 d1 89 54 5b 44
LL Data: 02 0f 33 82 13 00 ce ed 62 74 44 52 6d c8 8d ed 40
LL Data: 0a 1f d0 75 19 a7 42 c1 1b 91 a0 a7 36 6e 8d 1f 41 83 dc f8 08 01 f0 26 1c 27 f2 46 9e 4d 1b 6f 43
LL Data: 02 0f ad f7 d9 97 fb c2 73 81 5b fd 29 cc 8b fa 68
LL Data: 0a 1f 1b ba a6 48 fb 64 e2 7b 37 e9 c9 81 0b b8 73 bf 72 77 03 b3 0b 01 3b 36 29 71 2b 34 aa 18 b6
LL Data: 02 0f 5a 40 0b 85 5a 7b a9 3d 8c 70 dd 5d 0a 3d 11
LL Data: 0a 1f f7 23 76 73 4c 67 61 88 f1 d9 28 f3 b1 df 42 f6 94 5d f2 f2 76 60 b1 31 24 92 34 1b 1a fb 31
LL Data: 02 0f 30 08 d0 33 f3 63 92 b7 24 ec 61 9d 3a 58 cc
LL Data: 0a 1f 03 03 81 d1 42 57 f8 56 c4 f0 96 32 a5 a0 43 27 54 53 12 fc 23 f2 a1 30 7b 0c 2a 05 93 69 1d
LL Data: 02 0f 5e 0a 3e 94 46 b6 72 b1 fe 77 9e d2 c3 06 a8
LL Data: 0a 1f ca 90 ae cc e0 53 2f 51 21 12 4b b1 e2 ca c1 72 8e 23 cb 4e b3 2d 36 76 b4 9e 72 e9 91 56 a9
LL Data: 02 0f 63 02 0f d5 ef 1e 33 f1 cf 27 ac 37 aa a4 46
LL Data: 0a 1f 37 df a8 93 80 d6 54 7a 92 7d 49 a9 9b ca 51 f5 81 ef b6 d3 c5 94 b2 17 91 11 00 da 56 09 00
LL Data: 02 0f 72 87 97 9a 96 3c 35 e9 cb b8 0a 31 41 40 5f
LL Data: 0a 1f e2 9c 25 f3 d0 aa 35 4b f0 fd e5 d7 cc a6 15 7e 9c 85 3e f9 b6 d4 f4 08 2f 8a 15 5b a0 21 53
LL Data: 02 0f 72 47 09 9b 95 53 ff 20 4d 1e f9 fb 99 42 b6
LL Data: 0a 0d e7 7a 67 12 02 65 b2 1a 71 e9 aa e9 b4
LL Data: 02 0d 02 61 f5 46 3b 54 23 08 cf 1b 7e fd c7
LL Data: 0a 0e 9b 24 b7 f1 40 30 a0 1f 1d ba 4b f7 3c 91
LL Data: 02 0d f7 f9 8d 51 ce 50 5e 32 b7 27 9d ec 46
LL Data: 0a 0e 10 0f bf e4 70 07 a0 1b c1 f3 c6 e9 e4 53
LL Data: 02 0d 46 20 d9 42 ca 42 16 16 29 47 ba 29 af
LL Data: 0a 0e 33 f5 b9 b7 d4 29 2c a2 82 8f 46 24 9f 18
LL Data: 02 0d 65 93 27 f3 7b ed f4 04 45 35 33 6c b2
LL Data: 0a 0e e5 9a 51 2a 24 c7 2f 32 f7 0a c1 50 e4 12
LL Data: 02 0d fa cb c5 7e 8f c4 92 be 61 87 26 6d 51
LL Data: 0a 0e 2d c1 da 1c 6a 0b fe 27 bc fa ea af 6d 51
LL Data: 02 0d 00 10 6a d7 3d af 2f b1 12 31 b4 2b 09
LL Data: 0a 0e 78 cd c2 9a ed 67 5f f6 44 8c cc e8 89 0d
LL Data: 02 0d 13 2b 5b 28 27 06 a1 8f c9 99 01 0c 07
LL Data: 0a 0e 89 69 71 78 ec 9a 9e 25 86 b2 23 43 c0 10
LL Data: 02 0d b1 ce 5e 8a c4 8c ca a8 20 42 61 97 a8
LL Data: 0a 0d bf c8 3f 88 96 40 90 09 67 37 69 00 2a
```

```
LL Data: 03 1c 59 d3 c0 83 ef 2c 0f 87 95 99 52 c7 52 4f 3e 77 14 e3 48 c9 1f 29 85 5e 8c a5 d6 2d
LL Data: 0b 07 31 e2 7f 5f b0 79 88
LL Data: 03 0d f3 73 64 40 08 1e c1 06 60 50 0d 10 16
LL Data: 0b 0d 5a 4e 75 8f 4b 5f fb f6 87 4c d5 04 38
LL Data: 0f 0d 42 09 61 84 99 f2 4f 72 13 01 25 82 99
LL Data: 07 0d 40 61 21 c2 5c 38 d8 5c 70 0a 72 5c 1b                  (somewhere above the write was executed successfully)

LL Data: 07 0d 53 f4 e9 e5 d5 d3 ac 6e 53 53 33 ad 41
LL Data: 0f 0d 1c 9b a9 15 52 b4 4a 39 4f 5b e4 1c 42
LL Data: 03 0d ef 1d 85 16 09 78 2a 59 f7 0c d5 f9 57
LL Data: 0b 0d 78 f4 5b 57 e0 fb de 32 cc 78 83 e2 6c
[!] Connection lost, sniffing again...
^C[i] Quitting


$ sudo btlejack -f 0xb142c518                                          (restart sniffing on connection 0xb142c518)
BtleJack version 1.3

[i] Using cached parameters (created on 2019-06-13 21:07:06)
[i] Detected sniffers:
 > Sniffer #0: fw version 1.3
 > Sniffer #1: fw version 1.3
 > Sniffer #2: fw version 1.3

[i] Synchronizing with connection 0xb142c518 ...
âIJŞ CRCInit: 0x2c9fee
âIJŞ Channel Map = 0x1fffffffff
âIJŞ Hop interval = 6
âIJŞ Hop increment = 15
[i] Synchronized, packet capture in progress ...
LL Data: 02 0b 58 e2 d6 32 70 8d 94 96 84 45 20                    (encrypted read request on live ride characteristic)
LL Data: 06 14 a8 10 77 42 99 da 14 77 b7 fe 4e f6 ca 12 ae 00 1f 7a 27 d1          (encrypted response on the read request)
LL Data: 02 0b b4 68 cb b0 31 79 94 df eb a9 86                   (2nd encrypted read request on live ride characteristic)
LL Data: 06 14 72 b7 d2 bf f3 20 28 ba 49 ba da 76 5d cf 48 18 ef 2a 87 30      (encrypted response on the 2nd read request)

LL Data: 02 0b f0 fc 43 e4 5b 40 ec f7 57 00 93                   (3rd encrypted read request on live ride characteristic)
LL Data: 0a 14 9e 4f 49 33 fb 88 35 3e 2a 10 cb a3 86 87 93 a2 20 63 53 a5      (encrypted response on the 3rd read request)
LL Data: 02 0b 28 2c b9 2e d9 93 b9 96 7a 2e d2                   (4th encrypted read request on live ride characteristic)
LL Data: 06 14 c2 27 e5 7d 37 d3 44 66 31 fa 80 22 03 cb 76 d9 ff 02 9c 89      (encrypted response on the 4th read request)

LL Data: 02 0d 65 4f 87 99 e6 98 7c 43 4f ea 3a db 1d        (activate live ride characteristic notify messages (encrypted))
LL Data: 06 09 77 5d ca 85 85 f6 27 1d 03
LL Data: 06 16 8a 37 2a 00 2e 59 e7 49 43 fe 82 55 5d cb ee 43 7b e1 c1 04 4f 97     (encrypted notify messages of live ride
characteristic)
LL Data: 0a 16 6b 01 68 f5 98 14 60 f9 27 3f dd 0a e5 7d 3b b9 74 02 fb 78 e6 07
LL Data: 0a 16 35 da c9 68 4b eb b7 40 1f 92 71 6f 34 a6 c5 80 67 c0 d4 e8 3c 0b
LL Data: 0a 16 0e f3 2c 6a 12 23 a0 aa 43 5f 35 5a f2 a5 10 56 77 1a b4 67 49 90
LL Data: 0e 0d 38 e3 28 0f 70 77 c6 16 64 dc d6 ed 24      (deactivate live ride characteristic notify messages (encrypted))
LL Data: 06 09 36 5d 83 f8 1c 31 62 22 fe

LL Data: 0e 0d 99 55 7c 12 41 97 33 1b e7 24 a5 46 71        (activate live ride characteristic notify messages (encrypted))
LL Data: 06 09 b3 76 db 21 f0 73 c5 16 3f
LL Data: 0a 16 a8 82 e3 1a f1 2f 04 a1 7e ba 4a 28 7c f6 9d 70 dd 70 79 0d c6 82     (encrypted notify messages of live ride
characteristic)
LL Data: 06 16 0e 58 09 b6 ce 7d d3 e4 2e 00 9b a1 62 00 76 5b f6 bf 4f 64 e5 be
LL Data: 06 16 fc f6 d3 5d 64 a2 e1 cb 0a ae 3d 5e 36 1c 13 89 f6 e5 23 52 3d 6f
LL Data: 0a 16 aa 04 6c df c3 c2 af aa e6 63 29 f1 fd 18 6e f7 20 ba 2e 86 fd e4
LL Data: 02 0d cc ef 0b 7f 02 95 22 31 70 44 07 32 dd      (deactivate live ride characteristic notify messages (encrypted))
LL Data: 0a 09 89 bd c1 b4 46 a3 f2 da 12
```