MBAN

How to securely connect an Android phone to the nodes of the MBAN and process it's data

by

Laurens Rutten (4450787)
Mick Koertshuis (4355059)

voor het bachelor afstudeer project MBAN
aan de Technische Universiteit Delft.

Supervised by Dr. Ir. A.J. van Genderen and Dr. Ir. I.E. Lager
Proposed by Dr. Ir. Christos Strydis and Ir. Muhammad Ali Siddiqi
Department of Electrical Engineering

Defended before Dr. Ir. A.J. van Genderen, Dr. Ir. I.E. Lager and Dr. Ir. Christos Strydis

Bachelor Graduation Thesis
January 15, 2020

Abstract-This project was commissioned by the department of neuroscience at Erasmus MC to help future research and on epilepsy and potentially develop a reliable way of prevention some seizures. In this project a Medical Body Area Network is proposed to accomplish this. With the MBAN proposed it will be possible to safely and securely collect data from sensor nodes on the body. Different methods will be discussed for this to be achieved and a prototype will be made.

Contents

1 Introduction

This project was commissioned by the department of neuroscience at Erasmus MC. They are doing research on epilepsy. In the commission they wrote that epilepsy is a medical condition which is identified as a period of health symptoms due to abnormally excessive or synchronous activity in the brain. Outward effects vary from uncontrolled shaking movements involving much of the body with loss of consciousness (tonic-clonic seizure), to shaking movements involving only part of the body with variable levels of consciousness (focal seizure), to a subtle momentary loss of awareness (abscence seizure). Most of the time these episodes last less than 2 minutes and it takes some time to return to normal. They are currently researching seizure prevention. In their latest experiments they concluded that timely seizure prevention is possible if sufficient sensory recordings are present. Epilepsy manifests itself across the whole body, Erasmus needs sensory data of multiple body functions. Some of these body functions are: sweating, elevated heart rate, rapid muscle tone, synchronized neuron firing etc. Because of this the department of neuroscience commissioned a proof-of-concept implementation of a secure and reliable Medical Body Area Network (MBAN) for seizure prevention. The seizure prevention itself will not be done in this project. This project focuses on implementing the medical body area network.

1.1. Body Area Network

A Body Area Network (BAN) is composed of multiple nodes sharing information between each other. The range of the Body Area Network should be, as the name suggest, in close proximity of the body with little to no outward transmissions. Body Area Networks have different uses, one of these uses is the monitoring of a patient's health. A Body Area Network needs to be comfortable to wear, because for the purpose of monitoring sensor data it needs to be applied for long durations of time. Most studies because of this focus on applying wireless body area networks (WBANs) [13][17][23]. The BAN also needs to work with implantable sensor nodes. Application of this is often called a Medical Body Area Network (MBAN). The MBAN commissioned needs to work with both wearables and implants. Some challenges are: the power consumption, connectivity and security.

1.2. Mobile Gateway

The project commissioned needs to have a mobile phone as a gateway for the network. This make the project technically a combination of a Body Area Network and a Personal Area Network because the communication of the nodes is not limited by the proximity of the body but also in close proximity of the person using the device. A mobile phone is not always on the body. With data transmissions being done outside the body, it will also bring some security issues. The wireless channel makes the data prone to being eavesdropped [20]. The Mobile Gateway is defined as an android phone in the commission. This means that the project will consist of both embedded software engineering and software engineering.

1.3. Focus of the thesis

The project is divided in two subgroups. One subgroup will focus on the implementation of the sensor-to-sensor network and the subgroup responsible for this thesis will focus on how to securely connect an Android

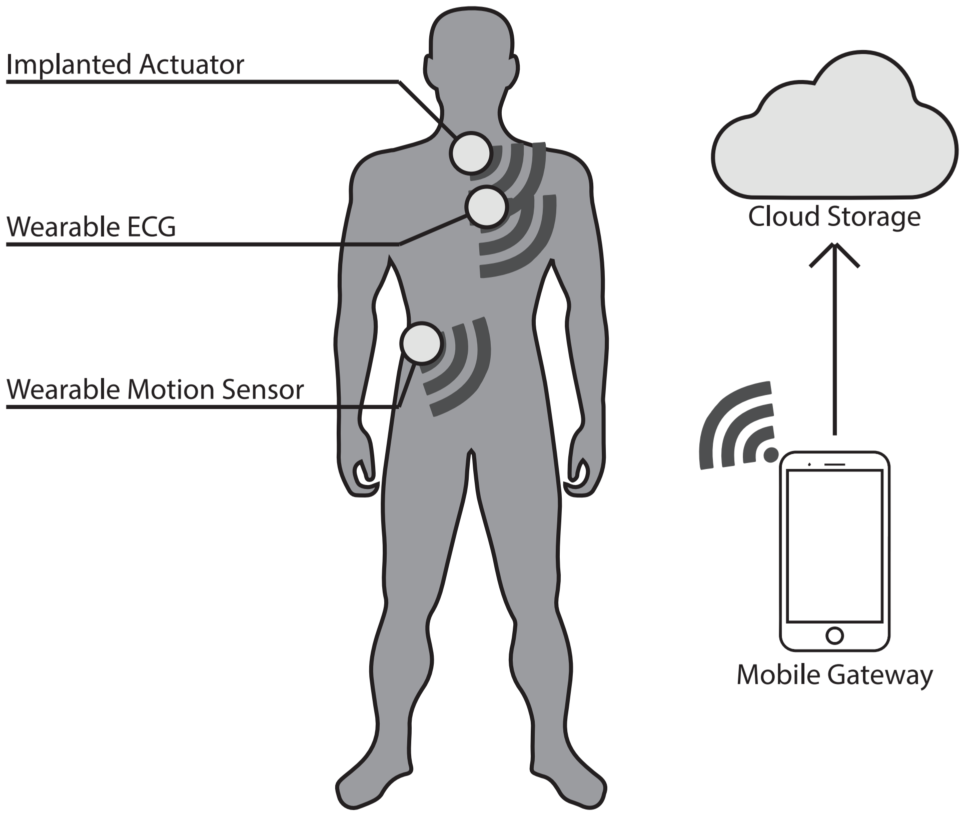


Figure 1.1: Overview of the BAN proposed

phone to the nodes of the MBAN and process its data. The requirements for this subgroup are defined in section 2.1. This subgroup will build the android application and implement a secure way of connecting the Android phone to the network presented by the other subgroup. In this thesis the wireless communication, wireless security, data storage, data management and key transfer will be discussed. This thesis will also have a discussion on the implementation in the proof-of-concept prototype presented. In the end a conclusion of the discussion will be made.

2 Program of Requirements

The purpose of the project is a proof-of-concept implementation of a secure and reliable Medical Body Area Network (MBAN) for seizure prevention. The project itself does not consist of the actual seizure prevention. It will merely be a framework to be used for the seizure prevention. If prevention is needed the network will set a flag that a seizure is detected and will activate an actuator so the user is informed that the detection was successful. The MBAN proposed needs to have the following features:

- Multi-sensor/-actuator human-body interaction; data fusion
- Real-time performance, mission-critical (QoS) functionality; mixed criticality
- End-to-end security (hardware, software, wireless)
- Dependability, graceful degradation (hardware, software, wireless)
- (Ultra-)low-power consumption

The complete project was divided over two subgroups. One subgroup will focus on the question "What robust network topology to use for a MBAN and have it operating even when nodes are not connected to a phone?", while this report will focus on the question "How to securely connect an Android phone to the nodes of the MBAN and process it's data".

2.1. Requirements

The following are the actual requirements that will be done by our subgroup:

- Secure communication using standard methods among MBAN nodes and gateway.
- Data collection, management, storage and transmission; on- (e.g. smartphone) and off-site (e.g. Fog, Cloud) data fusion.
- An android app on which the data of the sensors is presented with live measurements and a clear layout.
- Detection algorithm for measured data.
- Notifications and user interactions for processed data.
- The app gives a warning when nodes are disconnected.
- The MBAN is expandable with extra nodes.
- The minimum result of the project is a working prototype.

3
Wireless Communication

The implementation of a Body Area Network is nothing new [13][23][31][17]. The requirement for this project was "secure communication using standard methods among MBAN nodes and gateway" as described in section 2.1. The way the wireless communication is implemented differ in most studies. There is however a standard provided by the IEEE [16].

3.1. IEEE 802.15.6

IEEE 802 has established a Task Group called IEEE 802.15.6 for the standardization of WBAN [16]. The standard they proposed describes the PHY and MAC layers. The standard can be used for medical and non-medical applications. The IEEE 802.15.6 supports three different PHYs:

- Narrowband
- Ultra Wideband
- Human Body Communications

They network also operates in one of the following modes:

- Beacon mode with beacon period superframe boundaries
- Non-beacon mode with superframe boundaries
- Non-beacon mode without superframe boundaries

The standard defines the following three levels of security. Each security level has different security properties, protection levels and frame formats:

- Level 0 - unsecured communication
- Level 1 - authentication only
- Level 2 - authentication and encryption

This project tries to follow the standard as close as possible while working with existing wireless communication techniques. For the importance of securing medical data the project will need to achieve the highest level of security, therefor needs authentication and encryption.

3.2. Current applications

Monton et al. uses a BAN based on the use of ZigBee/IEEE 802.15.4 standard technology and off-the-shelf modules [23]. While others use Bluetooth Low-Energy for their WBAN [40]. Both have similar purposes in their study, mainly monitoring patients. Monton et al. writes that it is common to use radio frequency (RF) technologies, mainly Bluetooth, for most BANs. This communication is usually handled through a point-to-point connection between the sensor and the monitoring unit [23]. They also write that one of the main issues is with older applications is the power consumption of Bluetooth. But Bluetooth Low Energy would emerge later. Which in contrast with previous Bluetooth flavors, has been designed as a low-power solution for control and monitoring applications [9]. That being said, it won't completely out-rule the use of ZigBee. The differences will be discussed later.

3.3. Intra-body Communication Technologies

Another physical layer IEEE 802.15.6 proposed was the use of Human Body Communication. ZigBee and and BluetoothLE both use radio-frequencies but there are other methods of communication with the human body as medium [34][2][31]. The motivation for using intra-body communication is that solely limiting the placement of sensors on the surface of the body limits the types of biological information that can be obtained [34]. While radio frequency wireless technology has been successfully deployed in most BAN implementations, they consume a lot of battery power, are susceptible to electromagnetic interference and have security issues [31]. Using intra-body communication technologies would therefore by definition been safer to use. This is however if we consider that the communication is solely be done between the sensor nodes inside the body. The purpose of this project is to also have a telephone connected to the network. An possible solution would be an adapter for your phone connected with a wire to a node on or inside your body. Or with a wearable device that can access its data [11]. This would mean that for acquiring the data, people need to wear the device consistently. Because this paper uses a mobile phone this would be difficult because people don't have their phone on their body all the time. Intra-body communication technologies can however improve the security of the proposed BAN. It will be discussed in later sections.

3.4. ZigBee

ZigBee is one of the most widely utilized Wireless Sensor Network standards with low power, low data rate, low cost and short time delay characteristics, simple to develop and deploy and provides robust security and high data reliability [27]. ZigBee is a standard based network protocol supported solely by the ZigBee alliance that uses the transport services of the IEEE 802.15.4 network specification. IEEE 802.15.4 defines the operation of low-rate wireless *personal* area networks. ZigBee networks can have up to 653356 devices, the distance between ZigBee devices can be up to 50 meters, and each node can relay data to other nodes [27]. This means that ZigBee is really scalable. Which is one of the requirements of the project. ZigBee however is prone to some practical attacks [37].

3.5. BluetoothLE

"The advent of BLE has occurred while other low-power wireless solutions, such as ZigBee, have been steadily gaining momentum in application domains that require multihop networking. However, BLE constitutes a single-hop solution applicable to a different space of use cases in areas such as healthcare, consumer electronics, smart energy and security [9]." Gomez et al. writes that while ZigBee has already achieved a significant presence in several market segments. They do not have high deployment expectations in devices such as smartphones. That why this project is more tilted towards the use of Bluetooth Low Energy for the wireless communication. Bluetooth Low Energy however doesn't come without its own issues [28]. Ryan shows that packets can be intercepted and reassembled into connection streams. They also demonstrate that the key exchange protocol can be rendered useless against passive eavesdroppers. This eliminates any confidentially associated with the protocol [28]. If Bluetooth Low Energy is used these security issues need to be addressed. Bluetooth LE is also very close to Ultra Wideband. Which is in the standard provided by the IEEE. They have the same range, use the same frequencies, have around the same amount of RF channels. The main difference is the modulation technique, Bluetooth LE used GFSK and Ultra Wideband uses BPSK and QPSK.

3.6. Comparison

In the above sections both ZigBee and Bluetooth LE are briefly discussed. In this section the differences will be taken under the loop and a choice will be made. Georgakakis et al. gives a great overview what can be used in the discussion. The relevant information is put in table 3.1.

	ZigBee	Bluetooth LE
Advantages	A low-power alternative to Bluetooth, that offers significantly improved performance of 30mW compared to Bluetooth 100mW.	It offers high spectral efficiency and low power consumption.
Disadvantages	Low data rate.	Not supported by many devices
Range	100 meters	10 meters
Maximum Data transfer rate	250 Kbps	1 Mbps
Power Consumption	30 mW	10 mW

Table 3.1: Comparison of ZigBee and Bluetooth LE [8]

The disadvantage that Georgakakis et al. gives for Bluetooth LE is not a problem for this project. This is because this project utilizes nodes that support Bluetooth LE. A quick survey on the latest phone releases also show that Bluetooth LE is in most recent devices. ZigBee has a greater range than Bluetooth LE but that might also be a disadvantage. Because radio frequency (RF) waves can penetrate obstacles, wireless devices can communicate with no direct line-of-sight between them. This makes RF communication easier to use than wired or infrared communication, but it also makes eavesdropping easier [10]. While both ZigBee and Bluetooth LE have this problem, a greater range makes it easier to eavesdrop. According to the requirements the device needs to have (ultra-)low-power consumption. Bluetooth LE has a lower power consumption so this project will prefer Bluetooth LE.

3.7. Issues

The choice for Bluetooth LE has some security issues. The security issues consist of: eavesdropping [28] and man-in-the-middle attacks [10]. To uphold the security standards this project needs to implement counter measurements to these issues. By far the best way to stop the attacks is to use SSP's (Secure Simple Paring) OOB channel in such a way that an attacker cannot have visual contact to the victim devices [10]. While the communication is done with Bluetooth LE we need to render the data useless for potential eavesdroppers. Not everybody should have access to the data on the network. This project implements a way of pairing that is harder for attackers to access. It will be discussed in chapter 4.

4 Wireless Security

If Bluetooth LE is used it comes with a number of issues (3.7). This chapter will discus possible solutions for said issues. In section 3.1 the standard for a body area network was also briefly discussed. This standard also gives us insight for the security specifications. The MBAN will be working with personal data, so you want to have the highest level of security possible for the network. The highest level of security as discussed earlier is authentication and encryption [16]. The data transmitted needs to be authenticated and encrypted. The standard also discusses a way of securely transfering data. "A Master Key (MK) is activated for unicast secured communication. The MK may be preshared or established using unauthenticated association. Then, a Pairwise Temporal Key (PTK) is created for a single session. For multicast secured communication, a Group Temporal Key (GTK) is shared with the corresponding group using the unicast method [35]." The data needs to be encrypted with different keys and only authorized devices will be able to access the keys. One crucial problem is the transmission of these keys [3]. With keys you can encrypt the data so other people will have useless data if they intercept the transmission. But that means that the attacker must not be able to access to keys. The transfer of the keys will be discussed later. For now we will focus on the IEEE 802.15.6 security structure.

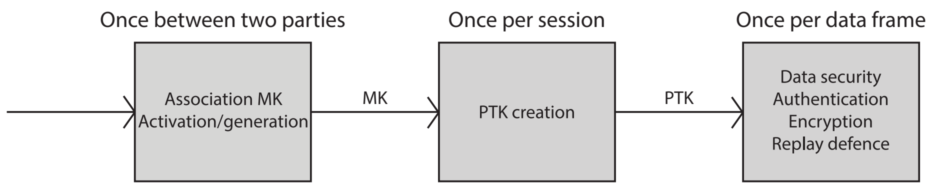


Figure 4.1: IEEE 802.15.6 security structure [35]

Figure 4.1 visualizes the structure earlier discussed. The project needs to follow this structure and build on it accordingly. The project will need a secure way of generating and/or sharing the Master Key. If the Master Key is established a Pairwise Temporal Key (PTK) or a Group Temporal Key (GTK) can be generated and shared via the Master Key. The PTK/GTK can be used to transfer the actual data. This report will discus each element of the security structure together with the choices made.

4.1. The Master Key

The Master Key is used for the authentication process and the connection establishment. The master key may be preshared according to the standard already discussed. However, stored data like a key can be leaked if a node is compromised [20]. A solution would be to store the key encrypted on the device and make it only accessible with, for example, a password. This will be doable for the application because than you will have a interface to enter the password. But on the nodes, which do not have an interface, will be much harder to encrypt. It is also worth noting that APKs (Android application packages) can be extracted and read. The Master Key will therefor not be pre-shared. Other solution this report will discus are the possibilities of generating the key and transferring of the key.

4.1.1. Key Agreement

Kompara and Hölbl divedes the key agreement schemes into four classes [15]:

- Traditional key agreement schemes
- Physiological value-based key agreement schemes
- Hybrid key agreement schemes
- Secret key generation schemes

The traditional key agreement schemes are agreement schemes most often implemented with pre-deployment of keying material. The key is usually stored somewhere on the device. If it is generated by the device itself or if the key is entered through external means doesn't really matter. The Psysiological value-based key agreement schemes are agreement schemes based on keys generated with body functions, like heart rate for example.

Traditional key agreement schemes

Traditional key agreement schemes are the least complex to implement. The key can be generated on the device and then saved in its memory so that the key can still be used after a reboot. The problem with this way of generating a key is that it needs to be saved on the device, which already established in section 4 that it would not be secure enough. The key can also be generated everytime the device starts. If the key is generated than it only needs a secure way of transmitting the key.

Physiological value-based key agreement schemes

In key negotiations based on physiological signals, two or more sensors measure (independently) the same physiological signal. This signal is then manipulated to get the common key [15]. This way the key doesn't have to be saved on the device because it can be generated every time the system reboots. There are a few examples of this. One example would be to use electrocardiogram signals to generate a key [36]. The system would create a key at a specific sampling rate for a fixed duration of time. Different nodes would acquire roughly the same data. In order to remove measurement artifacts the signal can than be smoothed. In the key agreement scheme proposed by Venkatasubramanian et al., none keys where identical between patients and the keys were random enough. Physiological value-based key agreement schemes have low computational cost, low memory storage, and low communication overhead [12]. This makes the use of Physiological value-based key agreement schemes interesting to use for an intra-body wireless body area network. Because one of the requirements is the (ultra-)low power consumption. The only problem with this scheme is that the different components need to measure the same overlapping physiological signal features. In the network proposed the different sensors are not always measuring the same attributes. There will probably exist different measuring nodes. One will measure heart rate and the other only motion or humidity. For this scheme to work the nodes need to measure (and compute) at least the same physiological signal feature. This will increase the power consumption. Another part of the problem is that the mobile phone does not measure anything by itself. This can be altered by adding a smart watch for example. The smart watch can measure the heart rate. For this project wearable extensions are used for additional data monitoring. These extensions only measure a single body function, therefor for this project using physiological value-based keys would not be possible. However if the sensors could measure multiple physiological functions these schemes would promise a valuable addition to the network. Because this project will not be using Physiological value-based key agreement schemes no measurements have be done to calculate the power consumption difference between transferring keys and computing keys on both platforms. We suggest that multimodal measurement devices need to be researched more for future research.

Secret key generation schemes

Is usually taken in the same category as traditional key agreement schemes [15]. Kompara and Hölbl writes that secret key generation is more similar to the physiological value-based key agreement than it is to the traditional approach. Except that instead of human vitals the key is generated with other overlapping signals. Signals close the the human body or even the signal of the communication itself.

4.1.2. Key Generation

Because of the shortcoming of the measurable data, the key can not be generated via physiological means. If a group master key would be established the project cannot use Secret key generation schemes. These schemes are very local to ensure security. If you make the scheme more acceptable to a wider range, attackers could potentially easier create a matching key from your environment. This project will use a traditional way of generating a key. For the MK the sensor node will need to generate a random key on startup. This key needs to be transferred securely to the phone so that it can be paired and a PTK/GTK can be generated to establish connection. The key needs to be random enough so that bruteforcing would not be efficiënt. For the project setup we use the arduino random() function to generate a random number from 0 to 255. This can be transformed to a byte. With these random bytes a 16 byte (128 bits) key can be generated. Using the traditional way of generating a key causes that the key needs to be transmitted for the pairing. The transmission of they key will be discussed in chapter 7.

4.2. The Pairwise Temporal Key and Group Temporal Key

According to the IEEE 802.15.6 proposed standard discussed in section 4 the PTK/GTK needs to be created once per session. After the MK is shared, the PTK needs to be generated. The phone will than send a PTK frame to the sensor. If the key is denied the sensor must not send data. This is the basic of the PTK and GTK Procedures [35]. Because the network consist of a variable number of nodes a GTK is used. The whole network will have the same key that can update regularly. The master keys are generated from the node and will be unique per node but the GTK will be generated from the phone. Generation of the GTK will be done with the build in cryptography provider from the .NET architecture. The GTK can then be transmitted to the sensors encrypted with the MK. The sensor will then send the data to the phone encrypted with the GTK.

4.3. Encryption

The standard proposed by IEEE 802.15.6 defines that the data transmitted needs to be encrypted [35]. There are different modes of Encryption/Decryptions [33]. In this section we will fist discuss each mode.

4.3.1. Modes of Encryption

Electronic Code Book (ECB)

The easiest to implement but also the weakest. The data is divided into 64-bit blocks and each block is encrypted one at a time [33].

Cipher Block Chaining (CBC)

In this mode each block of ECB encrypted ciphertext is XORed with the next plaintext block to be encrypted, thus making all the blocks dependent on all the previous blocks [33]. The first block is XORed with the Initialization Vector. This is more secure then ECB because you need the last message or the IV so this adds another layer of protection.

Cipher Feedback (CFB)

In this mode, blocks of plaintext that are less then 64 bits long can be encrypted [33]. Normally padding needs to be done to make blocks the correct length. The plaintext itself is not actually passed through the DES algrotihm, but merely XORed with an output block from it. This mode is similar to CBC and is very secure, but it is slower than ECB due to the added complexity.

Output Feedback (OFB)

This is similar to CFB mode, except that the ciphertext output of DES is fed back into the Shift Register, rather than the actual ciphertext [33]. This mode is less secure then CFB because only the real ciphertext and DES ciphertext output is needed to find the plaintext of the most recent block.

4.3.2. Algorithms

There are three different algorithms for encryption that will be discussed for implementation for this project.

Data Encryption Standard

DES can be used to encipher a 64-bit plaintext message block under a 56-bit key to create a 64-bit ciphered text [6]. There are however many known exploits and attacks that make it an insecure block cipher [33].

Blowfish
Blowfish is a symmetric block cipher that can be effectively used for encryption and safeguarding of data. It takes a variable-length key, from 32 bits to 448 bit, making it ideal for securing data [21]. The algorithm was first introduced in 1993, and as of 2011 has not been cracked yet [33]. Blowfish is also much faster than DES [24].

Advanced Encryption Standard
The key for DES is too small to be secure. Triple-DES can solve the key length problem but it is based on hardware encryption where most people use it today in software. That's why AES is introduced as a successor [14]. Brute force attack is the only effective attack known against this algorithm, it is however slower than Blowfish [33].

4.3.3. Implemented Encryption
For the project Blowfish will be used with CBC. DES is, as discussed, simply not secure enough in comparison with the other algorithms. Both AES and Blowfish would suffice for encrypting the data send, but because the execution time is shorter for Blowfish, this paper suggest using Blowfish. A shorter execution time would also mean that a little power is saved because less calculations need to be done. For this project an esp32[32] is used for simulating the sensor. Blowfish is also included in the esp32 arduino library and can be used in c# for the application.

4.4. Implementation

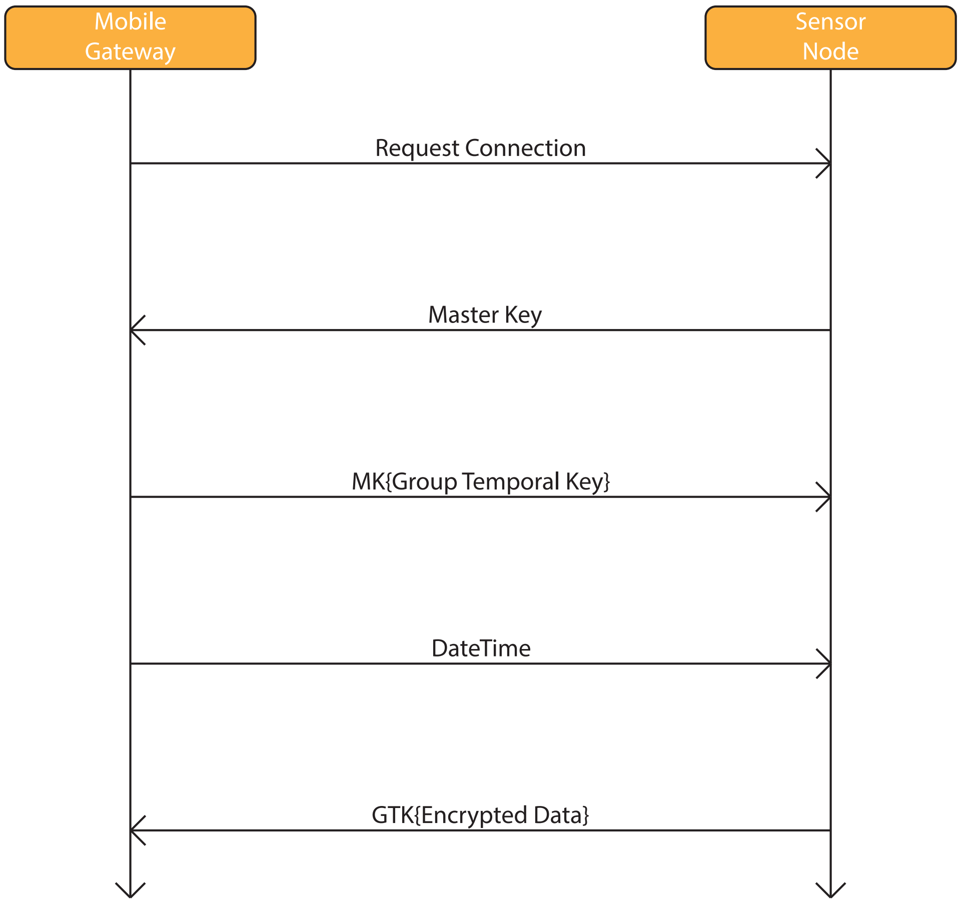


Figure 4.2: The authentication protocol

This project implements the authentication as follows. First the Sensor node will generate a key on startup that is used as the Master Key. If the Mobile Gateway wants to connect with the sensor the sensor will send the master key to the gateway. This needs to be done in a secure way to avoid eavesdropping. How the key will be transferred will be discussed in chapter 7. If the Mobile Gateway accepts the MK. It can establish a Bluetooth LE connection with the node. The gateway will than send the Group Temporal Key that it has generated to the node encrypted with the Master Key over the Bluetooth LE connection. The sensor can decrypt the GTK because it knows the MK. The GTK will then be used to encrypt the data that the sensor will send over the bluetooth connection. Now if the encrypted Data is intercepted the data will be useless because you need the key to decrypt it. The phone does not send anything to the sensor except the datetime and the GTK. So it will not be possible to corrupt the sensor by sending it wrong data. Only the datetime can be affected. The datetime is used to compare the time the data is measured versus the time the data is saved in the cloud. The authentication protocol is visualized in figure 4.2.

4.4.1. Possible Issues

Here a few possible issues will be discussed for the chosen implementation, together with potential fixes.

DateTime injection

Like discussed before. It is in theory possible to send the DateTime with another device to the sensors. However the sensor will than try to decrypt the DateTime with the GTK he has saved. This will result in corrupted data. The sensor needs to perform a check if the DateTime could be reliable. If it is not it needs to disrupt the connection and restart the authentication protocol.

Distributed Denial Of Service [19]

If the protocol starts by Requesting Connection over BLE it is possible to repeat the process to flood the network. This will deny access to other users. A fix would be to only start the protocol via the key transfer protocol that will be discussed in chapter 7.

5 Data Storage

When choosing how to handle storage of the data collected from the sensor network there are two main options; handling storage completely local and saving the data on the phone and when the phone runs out of storage transfer it to a PC (or even overwrite the oldest data on the phone and do not use a PC altogether), or using cloud storage where the data is stored over multiple servers which are owned by a hosting company.

5.1. Physical vs Cloud

In this section some of the differences between the two major options of data storage are discussed.

When considering security both physical and cloud storage have their advantages and disadvantages. Some of the major security requirements that need to be taken into account are [20]:

- Dependability
- Access control
- Accountability
- Revocability
- Scalability

Dependability means that the patient data should always be readily retrievable for the patient and the practicing physicians, so the data should always be reliably available. Physical storage is and advantage for the patient in this case, while cloud storage is better for the doctor. This is because with physical storage the data is saved on the phone, so for the patient it is already there, ready to be used, but for the doctor it is not. While for cloud storage everyone has the data when they want it (as long as there is an internet connection) and they can also have all the type of data they want, because everything is stored in the cloud because there are a lot fewer restraints than when physical storage is used. So in when only looking at this cloud would be the better option, but cloud has the disadvantage that is goes through hosting company, so it's only as dependable as the hosting company and if the servers go down it's not in the control of the consumer. Physical storage on the other hand is in control of the consumer, but if the phone is lost that means that all data on the phone is lost as well, while cloud backs up every save. So if it is acceptable to hand some the dependability over to a hosting company cloud would be a more dependable option in this scenario.

Access control implies that only the ones authorized (the patient and their doctor) have access to the patient data. Here the physical storage has the advantage that the data is only on the phone so only the one with the phone has access to the data, or the one allowed access the phone. But the problem of the lost phone arises again, when the phone is lost so is all access control since there is no way to prevent someone who really wants the patient data to get the data. Cloud storage on the other hand has the complication that the data is stored on the server of a hosting company and this company has to be trusted to not access that data and protect that data from unauthorized access. So if the patient can assure that the phone will stay in their possession, physical storage will have the better access control over cloud storage.

Accountability suggests that if a user of the MBAN abuses the patient data that users should be found and held accountable. If physical storage is used and the data is given to an other user (doctor or other patient) there is no control to check what this user actually does with the data, or how many times he/she accesses it. When using cloud storage the data can be pulled from the server by the authorized users, but after that there is the same problem as with physical storage; there is no control to see what the user actually does with this data. But cloud storage has the advantage that it is easy to keep a record of how many times users access certain data. So if a user accesses some specific data irregularly often this can maybe be further investigated. There also needs to non-repudation, this means that there there needs to be a service that ensures that the sender of a message is not able to deny its creation. For example, this would mean that a doctor who analyzes the data wrong should not be able to deny his/her involvement. Because there are more options to put some extra insight on who uses the data when using cloud storage this option takes the lead when it comes to accountability.

Revocability indicates that when a user is compromised or behaves maliciously that he/she is quickly identified and his/her privileges are taken away. This again ties in with the concept of accountability, because a users privileges can only be revoked if it is detected that the user misused the data. So again cloud storage is preferred in this situation since there is a little bit more surveillance over the data.

Scalability should also be considered when determining the method of storing data. In the future their will hopefully a lot of patients and multiple patients will have the same doctor, so these doctors will have a lot of data at their disposal. If this considered together with dependability, access control, accountability, revocability and non-repudation then cloud storage is the better option.

Because the amount of the data will increase by so much that it is more beneficial to have one central server where all data in saved instead of having multiple little 'servers'(phones) that each have the data of one individual patient and need to synchronized with the doctors.

Not only is cloud more scalable and as shown above overall more secure, but is also more cost effective, 'the costly systems and the people required to maintain them typically provides organizations with significant cost savings that more than offset the fees for storage. [41]'

5.2. Implementation

The final implementation is system that mainly relies on cloud storage with a little implementation of physical storage to make sure there is still data available even when the phone has no internet access.

When implementing the cloud storage a hosting company had to be chosen. There are a lot of possibilities; Amazon, Dropbox, Google and Alibaba just to name a few but eventually Microsoft was chosen as the cloud hosting company for the MBAN. This was mainly because the app itself was build on the Microsoft Xamarin architecture and since both Xamarin and Azure (the cloud platform from Microsoft) are Microsoft products they integrate very easily. Azure also has the advantage of not only being a storage system but more of a platform, so in the future more functions that are now done on the phone (like data management) could be transferred to the cloud. This is desirable since this will reduce computing necessary on the phone. Another reason why Microsoft Azure was chosen because its high scalability and disaster recovery to make sure that data can always be saved and retrieved which is important when it comes to patient data [4].

As stated earlier one of the disadvantages of cloud storage is that it is only accessible when the phone is connected to the internet. With large 4G networks now in the Netherlands (covers 98.8 percent of inhabitants [1]) and around the world and WiFi routers everywhere it isn't that big of an issue, but nevertheless as little patient data as possible should be lost due to lack of a internet connection. This is tackled by implemented the storage system so that when the phone is not connected to the internet the application will save the data on the phone itself until either the phone is connected to the internet again and dump the data in the cloud, or the phone can't save anymore in that case the oldest data is deleted and then and only then data is lost. When data is about to be lost this will also trigger a warning to the patient to find an internet connection as soon as possible to prevent data loss. It was also earlier discussed that patients should always have access to their own data, even when their is no internet connection, since it is their data. To minimize this disadvantage the cloud storage is implemented in such a way that when the phone is connected to the internet it grabs some of the most recently stored data from the cloud to make sure that the most recent data is also on the phone.

6 Data Management

The MBAN will produce a lot of patient data, what is done with this data and how it is analyzed will be discussed in this section.

6.1. Data Display

As stated earlier in this thesis the data is collected from the sensor by use of BLE. After the data is collected it needs to be displayed in the application in a pleasing and clear way. When implementing the display of the data on the phone one of the requirements in particular needs to be taken into account; 'the MBAN is expendable with extra nodes', because when nodes are added or taken away the display of the data should change as well.

To make sure that any type of sensor can be added or taken away the pages that are used to display the data are using a general class called Monitor. This means that the heart rate sensor and the orientation sensor are both from the monitor class, and if for example a temperature sensor is added this will also by from the monitor class. To keep track of what sensors are active in the system there is a list of all types of sensors. This list is displayed in the overview page where it displays all the sensors and their values. All sensors in the list also get there own page with the type of sensor and the value given by the sensor and more detailed information (like past values, and averages over time).

6.2. Calculations

The data should not only be displayed but there should also be some calculations done with this data, the main difference between the display and calculation is that the data should only be displayed when the patient wants to see it, while the calculations should be done all the time. These calculations should be done all the time to make sure that when an epileptic seizure occurs, it can be detected.

To make the app run calculations with the data it gets from the sensors all the time there are two options; either the app needs to be open all the time or there needs to be a way to perform tasks on the phone while the app is not open. The best method to use is looking for a way to perform tasks without the app needing to be to make sure the user can use the phone for other purposes. This again can be done in two ways; either by having the whole app run in the background or only run the specific tasks needed in the background. To have the whole app run in the background is easier to implement since the only thing that needs to be prevented is that the app is completely closed. This can be done by sending the user an alert when the app is completely closed. The downside to this is that it can deplete the battery very quickly [22]. The alternative is to use as Android calls it 'background tasks' these are tasks that start when the app is opened and keep running even when the app is closed, or the phone restarts. So these are the tasks that are used in the application to make sure that the data is still being analyzed and saved when the app is closed.

6.3. Alerts

Alerts or notifications are very important since they let the users know about everything that is going on. For instance, they alert the patient when an epileptic attack is coming, or when a node is disconnected. But there shouldn't be too many notifications since that the application might become to intrusive.

Alerts and notifications also make use of the background tasks stated earlier, since there should also be notifications shown to the user when the app is not open. They big issue to discuss was 'What is notified to the user and what is not'. There are obvious ones like the coming of an epileptic attack or the failing of a node, but should the user also be notified when there doctor is looking at their data? The final alerts notify the user when an attack is coming, when a node is disconnected, when the memory is running out and there is no way to upload it to the cloud and when an other user is accessing their data for the first time. The last one is to make sure that the patient knows who is accessing their data, they don't need to know when exactly they do, so that's why their only notified the first time. So when they don't trust the person they can immediately take action.

7 Key Transfer

For the authentication protocol discussed in chapter 4 it is for utter importance that the transfer of the Master Key is done in a secure way. The transfer of the key via Bluetooth Low Energy wouldn't be safe because it is very vulnerable to eavesdropping [28]. By far the best way to stop the attacks is to use an Out-Of-Band (OOB) channel in such a way that an attacker cannot have visual contact to the victim devices [10]. Out-of-band means that the activity is outside the defined activity that is used for the telecommunication. So in this case, outside the Bluetooth LE connection. The wireless connection protocol is chosen so that the phone will have a stable connection with the sensor network even when a few meters apart. But for establishing the connection a safer way would be to limit the range of the connection. That is why a OOB channel establishment with a protocol that limits this range would be more secure. In this chapter a few technologies will be discussed that can be used for the key transfer.

7.1. NFC

NFC is a candidate to be used for OOB key transfer because of its low range. The communication range is typically 0-20 centimeters and one clear advantage for using NFC technology is the availability of reasonable pricing of compact NFC devices [18]. Because of the short range, eavesdropping would be much harder. There are however two major concerns when using RF, tissue heating and energy replenishment once the sensor is implanted [34].

7.1.1. Tissue Safety

The Maximum Specific Absorption Rate of local exposure for the head is $2\,\mathrm{W\,kg^{-1}}$ for frequencies between 100 kHz and 6 GHz [30]. The Specific Absorption Rate produced by the NFC transmitter is far below the maximum [5].

7.1.2. Energy Consumption

For testing the power consumption a MFRC522 [25] is used. This is a development kit for NFC/RFID used for the esp32. While measuring this devices consumes 10 mW when idling and 15 mW when transmitting.

7.1.3. Complexity

NFC is a fairly simple concept of itself, the complexity kicks in when the position of the reader is discussed and the impact of RF signals on the skin as discussed earlier. Because there are already a lot of NFC or RFID devices on the market implementing it would be fairly easy. No adapter for the phone needs to be implemented because most phones have their own NFC/RFID adapter.

7.1.4. Security

NFC is pretty safe but like any technology there are some holes. One of these holes is that with very delicate and precise antennas it is possible to pick up NFC communication from a further away. This is however an issue with any type of RF signal, it is something that inevitable when using electromagnetic waves.

7.2. Ultrasound

Ultrasound has been used extensively for underwater communications due to the efficient propagation through media composed of mostly water [34]. Because the body consists of 65% water makes ultrasound an interesting candidate for the key transfer. For implantable applications, ultrasound waves can carry energy while propagating through tissue to an implanted device where it is converted to electrical energy using a piezoelectric transducer [2]. Because the waves need a medium using ultrasound would be secure for outside attackers.

7.2.1. Tissue Safety

Ultrasound does not use electromagnetic fields but propagation of waves. Ultrasound waves travel slower than electromagnetic waves. Because of this tissue is not heated as much than with electromagnetic waves. Ultrasound is therefor causing the least amount of stress to the tissue [34]. There is however an effect caused by ultrasound named cavitation. If the pressure of the ultrasound waves differentiate bubbles present in the medium will expand and retract. Bubbles might collapse when expansion is at its peak, damaging objects in close proximity [7]. According to Galluccio et al. no data collected on bioeffects of ultrasound waves in the last decade observed dangerous bioeffects to the tissue.

7.2.2. Energy Consumption

The energy consumption depends on the maximum range that is required. Because of the tissue safety when using ultrasound more power can be used to transmit these waves. Creating of those waves is done with piezoelectric transducers. For reference, Santagati and Melodia achieved a transmission in 19 cm of tissue with 10 mW for the transmitter and 16 mW for the receiver [29].

7.2.3. Complexity

When using ultrasound the transmitters and receivers need to be aimed. This can be really useful when you want to direct the signal only to where it is needed, but will require more precision with the implant. Ultrasound also works best when the medium is constant, if the medium changes reflections occur, thus reducing the amount of energy that actually passes through the boundary [7]. If communication is needed between an external source, like the phone used in this project, an adapter needs to be made. Skin to air signals can be achieved with pads with coupling gel and a piezoelectric transducer [2]. But because of the, with the gel yet slightly, change in medium it is unclear without testing if the signal is still recognizable with all the reflections.

7.2.4. Security

The need for aiming the ultrasound transmitters and receivers actually helps with making it more secure. Passive eavesdropping would be much harder to succeed if even able to succeed. Also because of the reflections in change of medium the signal will completely be contained inside the body. Ultrasound waves is because of this a completely secure way of wireless communication inside the body without any intervention from external means.

7.3. Galvanic Coupling

Galvanic Coupling is a way of communication by sending a low current through the body. Instead of the air or vacuum as medium for wireless communication, the body is used. This works really well for implanted devices because the conductivity of the muscle tissue is higher compared with the conductivity of the skin [38]. This will also mean that an attacker that has access to your skin will also have a weaker signal than other sensors inside your body.

7.3.1. Tissue Safety

If frequencies from 10 kHz to 1 MHz are used, the influence on body signals located below 10 kHz shall be omitted, while the drawback of signal loss above 1 MHz would be increasing [39]. By limiting the peak current to 1 mA the patient's safety is ensured [38]. Furthermore Galvanic Coupling has a Maximum Permissible Exposure of 1.8 $\frac{\text{mW}}{\text{cm}^2}$ [34].

7.3.2. Energy Consumption

The implantable Galvanic Coupled devices as proposed by Wegmueller et al. have a power consumption of 16 mW for the transmitter and 400 mW for the receiver [38]. This will make a distance variation of 35 cm feasible. While Tomlinson et al. claims that the power consumption for the transmitter can be as low as 2.0 mW [34]. For this project the higher power consumption for the receiver is not an issue because with the transfer of the master key the key is transmitted from node to the phone. The phone is in this case the receiver. The phone can have a bigger power supply than the sensor and it will also be able to get recharged more often.

7.3.3. Complexity

A mobile phone does not come with a build in receiver for currents emitted through the body. For galvanic coupling to work with the setup used extra steps need to be taken to implement it into the project. A relative simple way of doing this is the use ECG electrodes. Wegmueller et al. also utilizes this [39]. The signal received with the electrodes need to be amplified and filtered. After that the signal needs to be converted to a digital signal so a phone can read it with an USB adapter of some sort. The device only needs to be applied for the authentication protocol. After the authentication is complete it can be removed.

7.3.4. Security

Because you need to attach electrodes to your body close to the sensor you want to access, galvanic coupling is a secure way of authenticating intra-body communications.

7.4. Cable

If the devices are not implanted into the body they can be accessed with a wire. No wireless authentication is then necessary. With a wired connection no eavesdropping is possible because you are not transmitting an electromagnetic wave. A cable would also be able to provide power to the device and it can even be used to charge the battery. With a wired connection you can send the key in a secure way to the phone. The only problem being that the network also needs to work with implanted devices. It might in theory be possible to use intra-body wired communication [26], but more research need to be done. For this project a wired connection between the nodes and the phone for the authentication is only feasible if the nodes are placed on the body, not inside the body.

7.5. Implementation

For the prototype we use off-the-shelf components to give a proof concept of a safe way of implementing wireless authentication to be used for the wireless communication for the network. We chose to use NFC for this because NFC is present in most phones so a mobile adapter would not have been necessary. We use a esp32-wroom-32 [32] module as a sensor and a MFRC522 [25] for the transmission of the key. For the test set-up the esp32 will be sending mock-up data over the Bluetooth LE connection after the authentication protocol. This mock-up data could be sensor data. The MK must only be read by te phone, this would best to do with card emulation. But card emulation is not possible with the MFRC522 module. So for the prototype we have key card who gets the key from the sensor, the key can than be transferred to the phone. The key card is a passive NFC module and does not require power.

7.5.1. Authentication protocol

The sensor node will idle if no connection is established. When the card is held close to the node it will send the master key to the card after a time interval (see subsection 7.5.3). The node will then start advertising. The phone will than be able to request connection with the node over BLE but it will need the key first. The phone can read the key from the card and use it to send the GTK to the node over Bluetooth LE. The GTK will be encrypted with the MK as discussed in chapter 4. The phone and the node will now have established a connection via OOB pairing.

7.5.2. Power Consumption

The MFRC522 will only consume 10 mW when idling and use 15 mW when sending the key.

7.5.3. Security

The key must only be transferred to the user, you don't want potential attackers to access the key. With NFC normally the only requirement is to access the data if you are close enough. It would be not be secure if an attacker could get the key by only bumping into you. That's why a time delay is used before sending the key. The transmitter knows when a card is near and will than wait five seconds, if after the time delay the card is still present it will send the key. If the transmission fails the sensor node will create a new master key.

7.5.4. Discussion

Using the key card is purely chosen for the hardware we had available. When using NFC or RFID for the sensor nodes card emulation would be better. With card emulation you can get a new key directly to the phone by holding the phone close to the node. The time delay is a good addition to the authentication protocol. When quickly swiping past the node with a receiver would not get you the key. The setup shows that it is possible to use Out-Of-Band for authenticating the wireless connection. For this setup NFC is used but it is possible to use the other methods discussed in this section. This project relies on the Bluetooth LE network but it might be possible to exchange the BLE network with the methods discussed. More research is needed to implement a BAN with intra-body wireless communications if the requirement for accessing the network with a phone is not necessary.

8 Implementation

For this project a prototype was implemented. The prototype consist of two parts.

1. The Android Application that can access the MBAN
2. The key transfer for secure authentication

The application is written in C# and uses the xamarin framework. The application is able to connect to the network proposed by the other subgroup of the project. Because, during the project itself, it was not known what topology would be used the application can both connect to a star topology and a full mesh network topology. This is done by utilizing a SQL library with the known services and characteristics used by the network. When accessing the required sensor data the application searches the whole network for these characteristics, it does not matter what topology is used. When accessing the data on the network it can be encrypted using the technology discussed in section 4.3. The application is also able to notify the user if necessary. This happens if a sensor measurement is critical or if a node is disconnected. The user will than need to take precautions or extra steps to solve the problem. When the data is acquired it will be stored on the cloud. During the course of the project it was discovered that extra steps needed to be taken to reach the security goal of wireless transmission. That is why the second part of the prototype also includes a key transfer. In the next section (8.1) the implementation as described in this thesis will be discussed.

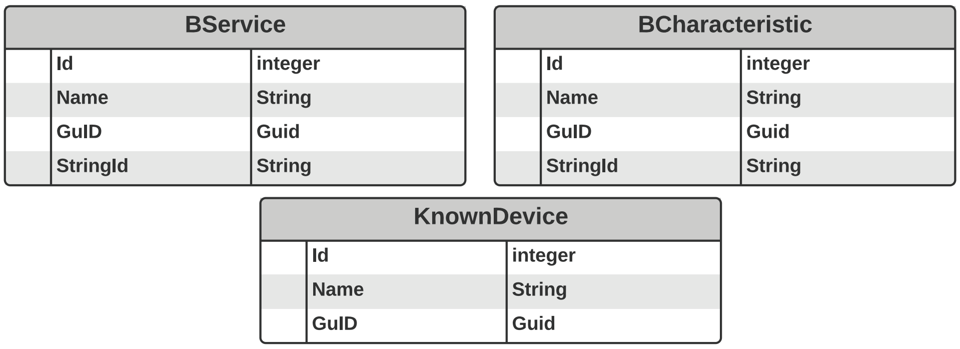
8.1. Implementations

8.1.1. BLE Connection

For the wireless communications this projected uses Bluetooth Low Energy because it supports off-the-shelf hardware. Developing the Bluetooth LE connection within Xamarin is relatively easy. No adapters are required and the documentation on Bluetooth development on Android phones is very in depth. For all the necessary functions needed a class was made named *BLE*. With the use of these class functions all the connectivity between the network and the application on the phone. The service and characteristic IDs necessary were provided by the network topology group. The code needs to be robust because it needs to be both configurable and be fail safe. If you get faulty authentication data the application must deal accordingly. In the code try-catch statements are often used together with throwing exceptions when data is incomplete. If necessary the user is informed via an alert display. A debug mode was also implemented, with this it was able to check the services and characteristics of all the connected devices. This was really useful for checking the correct services and characteristics that where included in the database. With a simple measurement setup the power consumption of the device was measured. The esp32 development kit used on average 150 mW while transferring data over BLE. This however does not represent an actual sensor node. If an actual sensor node will be implemented the power should be around 10 mW as discussed in section 3.6, the high power usage now is because of the development kit.

8.1.2. Security

The choice for using BLE did come with a few challenges. It did allow for connection between every node and the phone if necessary but it also brought security challenges with it. Solutions needed to be found for these



BService

	Id	integer
	Name	String
	GuID	Guid
	StringId	String

BCharacteristic

	Id	integer
	Name	String
	GuID	Guid
	StringId	String

KnownDevice

	Id	integer
	Name	String
	GuID	Guid

Figure 8.1: The structure of the SQL database used for searching the network for the correct characteristics and services.

challenges. That is why an advanced authentication protocol was designed for this project (see figure 4.2). This project implemented the NFC key transfer for the Out-Of-Band pairing. The pairing itself works. The time delay is a good addition because it adds another layer of security. The key transfer is able to fail however, this happens if the receiver is removed during transmitting. If this happens the transmitter throws away the old master key and generates a new one. This is done to counter potential eavesdroppers. Old devices connected will still be able to receive data because the GTK is unchanged. While testing we encountered that it is still possible to send a fake GTK to the sensor node. Making the pairing not reliable. To counter this a suitable change would be to send the Master Key (encrypted with itself) back to the sensor node together with the encrypted GTK. This way the sensor node must first check if the decrypted Master Key matches the Master Key it has send via Out-Of-Band. This would not be necessary for other implementations of the key transfer as discussed in chapter 7. Because those Out-Of-Band channel key transmission methods are more secure by default.

8.1.3. Cloud

One of the major challenges with using cloud is the security, since a hosting company is being used to save the data on their servers. The big issue with this is that there is no control on this hosting company, so it becomes an issue of trust that the company won't access the data. One way to prevent this issues would be to build a server yourself. This however costs a lot of money and time and the problem of scalability will come around again. Since, when the app and system scales then so does the amount of data retrieved from the sensor. This in turn will result in a increase in the need of storage space and when you're responsible for your own server you also have to scale it yourself. For a hosting company on the other hand the main goal is also scalability, since they want as many subscribers using their service as possible. One of things that could be implemented in the future would be to also transfer some of the computing to the cloud to even further reduce the computing load on the phone itself. Another possibility would be to look into other hosting companies since not all possibilities were looked into.

8.1.4. Management

The biggest challenge when working with the data management was working with the background events (having the phone still compute data even though the app is closed). The main reason why it was such a challenge was because, to test background events the app needs to be closed but if the app is terminated so does the debugger. This makes it a lot harder to test the background events. One thing that could be done to improve the data management could be to run more tests that actually track the battery life of the phone when running different solutions to make sure that the app does not consume as much.

9 Results/Discussion

Most of the implementation has already been discussed in chapter 8. In this chapter the program of requirements will again be discussed while looking at the implementation of the prototype. If some requirements where not fully satisfied the issues will also be discussed.

9.1. Requirements

Secure communication

Out-Of-Band pairing is the best way to handle a connection between the BAN and the mobile phone. It protects the network from eavesdroppers and man-in-the-middle attacks. The Out-Of-Band pairing could be done with any of the methods discussed in chapter 7. This project succeeded to exchange the key but NFC proves to be not preferred. NFC is really accessible for mobile devices but when considering a network without the use of the phone another method like ultrasound or galvanic coupling is preferred. If the mobile phone is still a requirement the network topology needs to be changed. A network with only a single sensor node that is used for gateway between the network and the phone would be a good place to start. That way it will only be necessary to apply an adapter to the skin once during setup, instead of multiple times like it would with using the topology used in this project.

Data collection, management, storage and transmission

The data collection is done via the BLE connection. The application searches all the connected devices for the right services and characteristics and updates the data it has collected. It collects data every time it has been changed on the sensor. Every time the data is collected the data is also saved to cloud. The data is given a timestamp in the filename to make it easier to find the data that is requested.

Android Application

It is possible to download the Android Package (APK) and install it on your Android phone. The application is written in xamarin.forms so it can also be ported to iOS with a bit of extra work. When opening the application your are greeted by the login page. After using the correct username and password it is possible to show all the data. There are a few default pages and a few generated pages. The default pages consist of:

- **Overview page:** Here is a list of all the sensor nodes that are available in the programming. If they are updating the measurements are also shown here.
- **Bluetooth page:** Here is all the Bluetooth LE connectivity done. You can view a list with all the bluetooth devices and connect to the accordingly. It is also possible to connect to all the known devices and debug the connected devices by viewing all their services and characteristics.
- **Settings page:** Used for debugging and basic functions like deleting all known devices from the database.

The generated pages are pages that are generated by the list containing the sensor nodes available, they are called Monitor.Page in the code. Every Monitor.Page has a start/stop button to enable or disable the live measurement. On the Monitor.Page the live measurement is also shown.

Detection Algorithm

The detection algorithm is not yet implemented into the prototype, but during the green light meeting the Erasmus research group said that there is a basic detection algorithm that can be used. The actuator however can be activated from the phone as well as from the sensor network itself, so when a detection algorithm is implemented it can immediately be used to trigger the actuator.

Notifications

Push alerts are implemented so that is user is notified when a sensor disconnects, when a new user accesses their data, or when the detection algorithm would be triggered. The notifications are also implemented is such a way that it can be easily ported which isn't normally the case because notifications are one of the things that are environment specific (different for iOS and android). This was done by implementing an INotificationManager which is shown in Appendix D

Handling Disconnection

The BLE plugin used has its own event handler for disconnecting devices. If a connection to a sensor is lost the user will be notified. This process does however take a few seconds to complete. The notification never failed during any of the tests, the user will always be notified if the phone is working correctly. How the user is notified if the phone itself will break or power down is outside our control.

Expandability

The expandability of the complete network is limited by the network topology. Unfortunately not enough sensors were available to test the maximum number of nodes. Some speculations are found that the maximum number of nodes that can be connected simultaneously is seven but no official sources are found. This could potentially be fixed if the phone only gets the data via a single BLE connection. One node will than need to transfer all the data it has acquired with other communication protocols, to the phone. Another solution would be if the application closes and opens the connection periodically but this will increase the power consumption. Application wise the network is fairly expandable. If a sensor is to be added a monitor object is created and added to the list. The monitor object has its own page which is also created accordingly. The main concern is the memory used when adding a lot of nodes.

Prototype

The prototype is an android application together with a sensor node that fulfills the requirements as discussed above. All the commits and branches of the application are saved on GitHub so it can be used for future research. With the prototype it is possible to connect to different nodes and collect their data as discussed in the implementation (see chapter 8).

9.2. Issues

During the project a few issues where discovered. Some have been acknowledged early on and potential fixes have been implemented. While other issues remain challenges for future research. In this section the issues will be discussed.

9.2.1. Software Heavy

One of the main issues that occurred that the project that was chosen was very software heavy. This was a problem because the Bachelor Graduation Thesis is set up the be a project that incorporates most, if not all, of the subject studied in the Bachelor. This project however has very little hardware design incorporated into it, while this is a major part of the curriculum. This does not concern the output of the project, only the background of the authors.

Android Development

Not only is the project very software heavy but it as also very focused on one particular software language and interface. The language that was used was C# which went well since there was some experience with C and C++ from the Bachelor and these skills translate well to C#. Even though developing in C# went pretty well there was still a steep learning curve since no one working on the project had any experience working with android, or developing phone applications.

9.2.2. The Mobile Phone Requirement

Like earlier discussed, the addition of the mobile phone requirement brought some challenges. Together with the uncertainty of the network topology some decisions where made that in retrospect could better be different. Right now the phone is able to connect to every node separately. This is really useful for debugging but not necessarily a secure way of a network topology considering that every signal that goes outside the proximity of the body can be intercepted. Also now the authentication protocol needs to be done between the phone and each node. If only one node will transfer information to the phone it will be possible to make an adapter to make a wired connection between the phone and the node. This will by far be the most secure method of data transfer between body and gateway.

9.2.3. Power Consumption

During the project only a handful of different hardware is used. Most research about power consumption of different methods is done by surveying. The NFC transmitter used is using only 10 mW when idling and 15 mW when transmitting. This can possibly be lower if own hardware is used.

9.2.4. Network Topology

This project is being build on a network topology that is been developed in parallel with this project. Working on this project parallel is in retrospect not optimal. The changing topology that this thesis needed to adapt to was inconsistent. It would be better for a comparable project to build on an existing network topology. That being said, it communication between the groups were good and progress was being made.

10 Conclusion

Using an Android phone to gather data from a body area network is user friendly. The patient can see his own data live on the application and will get notifications if something is wrong. The data that is acquired will be securely stored in the cloud and can be accessed by the patient and if necessary the doctor together with other authorized personnel. The way of connecting the phone to every node is not the best way to target the scalability of the network. It would be better and more secure to connect the phone to a single node. It is then the responsibility of the node to connect to other nodes. In this project BLE is used for the whole network but it is not optimal. We suggest using only BLE or even an wired connection from one sensor node or gateway node to the phone. The communication between the sensor nodes could potentially be done with body channel communication.

10.1. Future Research

When researching was done for the key transfer we acknowledged that the use of body channel communication techniques discussed in chapter 7 could potentially be a better way of wireless communication between the sensor nodes. The research done in this project about body channel communication is mostly done by surveying. We suggest more research need to be done about the implementation of these techniques.

Bibliography

[1] Vergelijk 4g dekking van alle providers. URL https://www.4gdekking.nl/.

[2] Kush Agarwal, Rangarajan Jegadeesan, Yong-Xin Guo, and Nitish V Thakor. Wireless power transfer strategies for implantable bioelectronics. *IEEE reviews in biomedical engineering*, 10:136–161, 2017.

[3] Erik-Oliver Blaß and Martina Zitterbart. An efficient key establishment scheme for secure aggregating sensor networks. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 303–310. ACM, 2006.

[4] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[5] S Cecil, G Schmid, K Lamedschwandner, J Morak, G Schreier, A Oberleitner, and M Bammer. Numerical assessment of specific absorption rate in the human body caused by nfc devices. In *2010 Second International Workshop on Near Field Communication*, pages 65–70. IEEE, 2010.

[6] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[7] Laura Galluccio, Tommaso Melodia, Sergio Palazzo, and Giuseppe Enrico Santagati. Challenges and implications of using ultrasonic communications in intra-body area networks. In *2012 9th Annual Conference on Wireless On-Demand Network Systems and Services (WONS)*, pages 182–189. IEEE, 2012.

[8] Emmanouil Georgakakis, Stefanos A Nikolidakis, Dimitrios D Vergados, and Christos Douligeris. An analysis of bluetooth, zigbee and bluetooth low energy and their use in wbans. In *International Conference on Wireless Mobile Communication and Healthcare*, pages 168–175. Springer, 2010.

[9] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[10] Keijo Haataja and Pekka Toivanen. Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. *IEEE Transactions on Wireless Communications*, 9(1):384–392, 2010.

[11] Keisuke Hachisuka, Azusa Nakata, Teruhito Takeda, Kenji Shiba, Ken Sasaki, Hiroshi Hosaka, and Kiyoshi Itao. Development of wearable intra-body communication devices. *Sensors and actuators A: physical*, 105(1):109–115, 2003.

[12] Chunqiang Hu, Xiuzhen Cheng, Fan Zhang, Dengyuan Wu, Xiaofeng Liao, and Dechang Chen. Opfka: Secure and efficient ordered-physiological-feature-based key agreement for wireless body area networks. In *2013 Proceedings IEEE INFOCOM*, pages 2274–2282. IEEE, 2013.

[13] Emil Jovanov, Aleksandar Milenkovic, Chris Otto, and Piet C De Groen. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of NeuroEngineering and rehabilitation*, 2(1):6, 2005.

[14] Ross Anderson1 Eli Biham2 Lars Knudsen. Serpent: A proposal for the advanced encryption standard. In *First Advanced Encryption Standard (AES) Conference, Ventura, CA*, 1998.

[15] Marko Kompara and Marko Hölbl. Survey on security in intra-body area network communication. *Ad Hoc Networks*, 70:23–43, 2018.

[16] Kyung Sup Kwak, Sana Ullah, and Niamat Ullah. An overview of ieee 802.15. 6 standard. In *2010 3rd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL 2010)*, pages 1–6. IEEE, 2010.

[17] Quang Duy La, Duong Nguyen-Nam, Mao V Ngo, Hieu T Hoang, and Tony QS Quek. Dense deployment of ble-based body area networks: A coexistence study. *IEEE Transactions on Green Communications and Networking*, 2(4):972–981, 2018.

[18] Antti Lahtela, Marko Hassinen, and Virpi Jylha. Rfid and nfc in healthcare: Safety of hospitals medication care. In *2008 Second International Conference on Pervasive Computing Technologies for Healthcare*, pages 241–244. IEEE, 2008.

[19] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2275–2280. IEEE, 2000.

[20] Ming Li, Wenjing Lou, and Kui Ren. Data security and privacy in wireless body area networks. *IEEE Wireless communications*, 17(1):51–58, 2010.

[21] Saikumar Manku and K Vasanth. Blowfish encryption algorithm for information security. *ARPN journal of engineering and applied sciences*, 10(10):4717–4719, 2015.

[22] Marcelo Martins, Justin Cappos, and Rodrigo Fonseca. Selectively taming background android apps to improve battery lifetime. In *2015 {USENIX} Annual Technical Conference ({USENIX} {ATC} 15)*, pages 563–575, 2015.

[23] E Monton, José F Hernandez, José Manuel Blasco, Thierry Hervé, Joseph Micallef, Ivan Grech, Andrea Brincat, and V Traver. Body area network for wireless patient monitoring. *IET communications*, 2(2): 215–222, 2008.

[24] Tingyuan Nie and Teng Zhang. A study of des and blowfish encryption algorithm. In *Tencon 2009-2009 IEEE Region 10 Conference*, pages 1–4. IEEE, 2009.

[25] NXP Ltd. MFRC522 Standard performance MIFARE and NTAG frontend. (3.9):95, 2016. URL `https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf`.

[26] Jihwan Park, Gi-Moon Hong, Mino Kim, Joo-hyung Chae, and Suhwan Kim. A 0.13 pj/bit, referenceless transceiver with clock edge modulation for a wired intra-ban communication. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2017.

[27] C Muthu Ramya, M Shanmugaraj, and R Prabakaran. Study on zigbee technology. In *2011 3rd International Conference on Electronics Computer Technology*, volume 6, pages 297–301. IEEE, 2011.

[28] Mike Ryan. Bluetooth: With low energy comes low security. In *Presented as part of the 7th {USENIX} Workshop on Offensive Technologies*, 2013.

[29] G Enrico Santagati and Tommaso Melodia. An implantable low-power ultrasonic platform for the internet of medical things. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[30] IEEE International Committee On Electromagnetic Safety (SCC39). *IEEE Standard for Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields , 3 kHz to 300 GHz (Amendment 1)*, volume 64. 2010. ISBN 9781504455480.

[31] MirHojjat Seyedi, Behailu Kibret, Daniel TH Lai, and Michael Faulkner. A survey on intrabody communications for body area network applications. *IEEE Transactions on Biomedical Engineering*, 60(8): 2067–2079, 2013.

[32] Espressif Systems. ESP32-WROOM-32. 2019. URL `https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf`.

[33] Jawahar Thakur and Nagesh Kumar. Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2):6–12, 2011.

[34] William J Tomlinson, Stella Banou, Christopher Yu, Milica Stojanovic, and Kaushik R Chowdhury. Comprehensive survey of galvanic coupling and alternative intra-body communication technologies. *IEEE Communications Surveys & Tutorials*, 21(2):1145–1164, 2018.

[35] Sana Ullah, Manar Mohaisen, and Mohammed A Alnuem. A review of ieee 802.15. 6 mac, phy, and security specifications. *International Journal of Distributed Sensor Networks*, 9(4):950704, 2013.

[36] Krishna Kumar Venkatasubramanian, Ayan Banerjee, Sandeep KS Gupta, et al. Ekg-based key agreement in body sensor networks. In *IEEE INFOCOM Workshops 2008*, pages 1–6. IEEE, 2008.

[37] Niko Vidgren, Keijo Haataja, Jose Luis Patino-Andres, Juan Jose Ramirez-Sanchis, and Pekka Toivanen. Security threats in zigbee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned. In *2013 46th Hawaii International Conference on System Sciences*, pages 5132–5138. IEEE, 2013.

[38] Marc Simon Wegmueller, Sonja Huclova, Juerg Froehlich, Michael Oberle, Norbert Felber, Niels Kuster, and Wolfgang Fichtner. Galvanic coupling enabling wireless implant communications. *IEEE Transactions on Instrumentation and Measurement*, 58(8):2618–2625, 2009.

[39] Marc Simon Wegmueller, Michael Oberle, Norbert Felber, Niels Kuster, and Wolfgang Fichtner. Signal transmission by galvanic coupling through the human body. *IEEE Transactions on Instrumentation and Measurement*, 59(4):963–969, 2009.

[40] Alan Chi Wai Wong, Mark Dawkins, Gabriele Devita, Nikolaos Kasparidis, Andreas Katsiamis, Oliver King, Franco Lauria, Johannes Schiff, and Alison J Burdett. A 1 v 5 ma multimode ieee 802.15. 6/bluetooth low-energy wban transceiver for biotelemetry applications. *IEEE Journal of Solid-State Circuits*, 48 (1):186–198, 2012.

[41] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *2010 International Conference on Intelligent Computing and Cognitive Informatics*, pages 380–383. IEEE, 2010.

A
The BLE class

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks;
using Plugin.BLE;
using Plugin.BLE.Abstractions.Contracts;
using Plugin.BLE.Abstractions.Exceptions;
using SQLite;
using Xamarin.Forms;

namespace BAP_App
{
    /// <summary>
    /// Most functions used for the bluetooth connections
    /// </summary>
    /// <permission cref="android.permission.ACCESS_COARSE_LOCATION"></permission>
    /// <permission cref="android.permission.ACCESS_FINE_LOCATION"></permission>
    /// <permission cref="android.permission.BLUETOOTH"></permission>
    /// <permission cref="android.permission.BLUETOOTH_ADMIN"></permission>
    class BLE
    {
        /// <summary>
        /// Gets the Bluetooth state of the device
        /// </summary>
        /// <param name="ble"></param>
        /// <returns>On,Off,Unavailable</returns>
        public static string GetBLEstate(IBluetoothLE ble)
        {
            if (ble == null)
            {
                throw new NotImplementedException("No IBluetoothLE device found.");
            }
            //The device checks what state the bluetooth adapter is in: on, off,
            ↪ unavailable, turningon, turning off etc and returns as a string
            var state = ble.State;

            return state.ToString();
        }
```

```
/// <summary>
/// Scans for available devices over bluetooth and returns on Observable
 ↪  connection
/// </summary>
/// <param name="ble"></param>
/// <param name="adapter">The current bluetooth adapter</param>
/// <returns>A list of discovered devices, it doens't matter if they are
 ↪  connected or not</returns>
public static ObservableCollection<IDevice> ScanForDevices(IBluetoothLE ble,
 ↪  IAdapter adapter)
{
    if (ble == null)
    {
        throw new NotImplementedException("No IBluetoothLE device found.");
    }
    if (adapter == null)
    {
        throw new NotImplementedException("No IAdapter found.");
    }
    //This function only puts the name of found devices in a list
    ObservableCollection<IDevice> deviceList = new
     ↪  ObservableCollection<IDevice>();
    var state = ble.State;  //gets the state of the bluetooth device
    if (state == BluetoothState.On) //checks if the bluetooth is turned on,
     ↪  if it isnt it skips searching
    {
        if (adapter.IsScanning)
        {
            adapter.StopScanningForDevicesAsync(); //stops the scan if the
             ↪  adapter is already scanning
        }
        adapter.ScanTimeout = 10000;   //adds a timeout to the scan
        adapter.DeviceDiscovered += (s, a) => deviceList.Add(a.Device);
         ↪  //adds the names of the discovered devices to the list
        adapter.StartScanningForDevicesAsync();
    }
    return deviceList;
}
/// <summary>
/// Connect to a selected device
/// </summary>
/// <param name="adapter">The bluetooth adapter</param>
/// <param name="device">A device, this can be selected in a list or just try
 ↪  already known devices</param>
public static async void ConnectToDevice(IAdapter adapter, IDevice device)
{
    if (device == null)
    {
        throw new NotImplementedException("No IDevice found.");
    }
    if (adapter == null)
    {
        throw new NotImplementedException("No IAdapter found.");
    }
    try
    {
        await adapter.ConnectToDeviceAsync(device);
        Debug.WriteLine("Connected to device: " + device.Id);
    }
    catch (Plugin.BLE.Abstractions.Exceptions.DeviceConnectionException e)
    {
        Debug.WriteLine(e);
    }
}
/// <summary>
/// Disconnect from a device
/// </summary>
/// <param name="adapter">The bluetooth adapter</param>
/// <param name="device">The device you want to disconnect from, can be
 ↪  selected in a list or just try one in connected devices</param>
public static async void DisconnectDevice(IAdapter adapter, IDevice device)
{
    if (device == null)
    {
        throw new NotImplementedException("No IDevice found.");
    }
    if (adapter == null)
    {
        throw new NotImplementedException("No IAdapter found.");
    }
    try
    {
        Debug.WriteLine("Disconnecting from device: " + device.Id);
        var connected = adapter.ConnectedDevices;
        foreach (IDevice i in connected)
        {
            if (i.Id == device.Id)                      //This prevents that
             ↪  the device settings are changed. Just get the Id
            {
                await adapter.DisconnectDeviceAsync(i);
                Debug.WriteLine("Disconnected from device: " + i.Id);
            }
        }
    }
    catch (Plugin.BLE.Abstractions.Exceptions.DeviceConnectionException e)
    {
        Debug.WriteLine("Disconnecting Failed with exception: " + e);
    }
}
/// <summary>
/// Starts updating a characteristic. It also creates a eventhandler for if
 ↪  the value is updated. This is now for HeartRate, it will be changed later
/// </summary>
/// <param name="characteristic">The characteristic you want to keep
 ↪  updating</param>
/// <param name="value">The value you want to start updating</param>
/// <returns>eventhandler that changes a label</returns>
public static async Task UpdateValueStart(ICharacteristic characteristic,
 ↪  Monitor monitor)
{
    if (characteristic == null)
    {
        throw new NotImplementedException("No ICharacteristic found.");
    }
    if (monitor == null)
    {
        throw new NotImplementedException("No Monitor found.");
    }
    ///Starts updating update_value for the given characteristic
    if (characteristic.CanUpdate == true)
    {
        SetUpdatedCharacteristic(characteristic, monitor);
        await characteristic.StartUpdatesAsync();
        monitor.updating = true; //sets the flag for updating true
    }
    else
    {
        Debug.WriteLine("The following characteristic doesn't except Updates:
         ↪  ", characteristic.Name);
    }
}
/// <summary>
/// Setting up the eventhandlers for updating the global values
/// </summary>
/// <param name="characteristic">the characteristic that is updating the
 ↪  value</param>
/// <param name="sensor">used in the switch statement for choosing the
 ↪  eventhandler</param>
public static void SetUpdatedCharacteristic(ICharacteristic characteristic,
 ↪  Monitor monitor)
{
    if (characteristic == null)
    {
        throw new NotImplementedException("No ICharacteristic found.");
    }
    if (monitor == null)
    {
        throw new NotImplementedException("No Monitor found.");
    }

    characteristic.ValueUpdated += (o, args) =>
    {
        monitor.Value = args.Characteristic.Value;
        Debug.WriteLine("Found value: " +
         ↪  System.Text.Encoding.ASCII.GetString(args.Characteristic.Value));
    };

}
/// <summary>
/// Stops updating a characteristic
/// </summary>
/// <param name="characteristic">The characteristic you want to stop from
 ↪  updating</param>
/// <returns></returns>
public static async Task UpdateValueStop(ICharacteristic characteristic,
 ↪  Monitor monitor)
{
    if (characteristic == null)
    {
        throw new NotImplementedException("No ICharacteristic found.");
    }
    try
    {
        await characteristic.StopUpdatesAsync();
        monitor.updating = false; //set a flag for updating false
        Debug.WriteLine("Succesfully stopped updating: " +
         ↪  characteristic.Id.ToString());
    }
    catch (Exception e)
    {
        Debug.WriteLine("Failed to stop updating: " +
         ↪  characteristic.Id.ToString());
        Debug.WriteLine(e);
    }
}
/// <summary>
/// Searches the characteristic for the given guid.
/// </summary>
/// <param name="service">The service you want to search in</param>
/// <param name="guid">The guid of the characteristic you are searching for.
 ↪  This is known in the network</param>
/// <returns>ICharacteristic</returns>
/// <seealso cref="SearchService(IAdapter,String)"></seealso>
public static async Task<ICharacteristic> SearchCharacteristic(IService
 ↪  service, string guid)
{
    if (service == null)
    {
        throw new NotImplementedException("No IService found.");
    }
    if (guid == null)
    {
        throw new NotImplementedException("No guid found.");
    }
    ///Searches the characteristic for the given guid
    try
    {
        var characteristic = await
         ↪  service.GetCharacteristicAsync(Guid.Parse(guid));
        return characteristic;
    }
    catch (Exception e)
    {
        Debug.WriteLine(e);
        return null;
    }
}
/// <summary>
/// Searches the characteristic for the given guid.
/// </summary>
/// <param name="service">The service you want to search in</param>
/// <param name="guid">The guid of the characteristic you are searching for.
 ↪  This is known in the network</param>
/// <returns>ICharacteristic</returns>
/// <seealso cref="SearchService(IAdapter,String)"></seealso>
public static async Task<ICharacteristic> SearchCharacteristic(IService
 ↪  service, Guid guid)
{
    if (service == null)
    {
        throw new NotImplementedException("No IService found.");
    }
    if (guid == null)
    {
        throw new NotImplementedException("No guid found.");
    }
    ///Searches the characteristic for the given guid
    try
    {
        var characteristic = await service.GetCharacteristicAsync(guid);
        return characteristic;
    }
    catch (Exception e)
    {
        Debug.WriteLine(e);
        return null;
    }
}
/// <summary>
/// Searches the service given the guid
/// </summary>
/// <param name="adapter">The bluetooth adapter</param>
/// <param name="guid">The guid of the service you are searching for. This is
 ↪  known in the network</param>
/// <returns>IService</returns>
/// <seealso cref="SearchCharacteristic(IService,String)"></seealso>
public static async Task<IService> SearchService(IAdapter adapter, string
 ↪  guid)
{
    if (adapter == null)
    {
        throw new NotImplementedException("No IAdapter found.");
    }
    if (guid == null)
    {
        throw new NotImplementedException("No guid found.");
    }
    ///Searches the service given the guid

    var connected_devices = adapter.ConnectedDevices;//gets the list of
     ↪  connected devices
    IService service = null;
    ObservableCollection<IService> service_list = new
     ↪  ObservableCollection<IService>();
    if (connected_devices != null)
    {
        foreach (IDevice i in connected_devices)//Searches inside the list of
         ↪  connected devices
        {
            try
```

```
                {
                    service_list.Add(await
                     ↪  i.GetServiceAsync(Guid.Parse(guid)));//Ads a found
                     ↪  service to the list
                }
                catch
                {
                    Debug.WriteLine(i.Name + " Does not contain service " +
                     ↪  guid);
                }
            }
            int maxrssi = -999;
            if (service_list.Count == 0 || service_list == null)
            {
                return service;
            }
            if (service_list.Count == 1)
            {
                return service_list[0];
            }
            foreach (IService j in service_list) //This will search for the
             ↪  device with the best RSSI so it connects to the best one
            {
                try
                {
                    if (j != null)
                    {
                        if (j.Device.Rssi > maxrssi)
                        {
                            service = j;//There can be only one!
                            maxrssi = j.Device.Rssi;
                        }
                    }
                }
                catch (NullReferenceException e)
                {
                    Debug.WriteLine("There are no services found with the
                     ↪  query");
                    Debug.WriteLine(e);
                }
            }
        }
        else
        {
            Debug.WriteLine("No connected devices");
        }
        return service;
    }
    /// <summary>
    /// Searches the service given the guid
    /// </summary>
    /// <param name="adapter">The bluetooth adapter</param>
    /// <param name="guid">The guid of the service you are searching for. This is
     ↪  known in the network</param>
    /// <returns>IService</returns>
    /// <seealso cref="SearchCharacteristic(IService,String)"></seealso>
```

```
public static async Task<IService> SearchService(IAdapter adapter, Guid guid)
{
    if (adapter == null)
    {
        throw new NotImplementedException("No IAdapter found.");
    }
    if (guid == null)
    {
        throw new NotImplementedException("No guid found.");
    }
    ///Searches the service given the guid

    var connected_devices = adapter.ConnectedDevices;//gets the list of
     ↪  connected devices
    IService service = null;
    ObservableCollection<IService> service_list = new
     ↪  ObservableCollection<IService>();
    if (connected_devices != null)
    {
        foreach (IDevice i in connected_devices)//Searches inside the list of
         ↪  connected devices
        {
            try
            {
                service_list.Add(await i.GetServiceAsync(guid));//Ads a found
                 ↪  service to the list
            }
            catch
            {
                Debug.WriteLine(i.Name + " Does not contain service " +
                 ↪  guid.ToString());
            }
        }
        int maxrssi = -999;
        if (service_list.Count == 0 || service_list == null)
        {
            return service;
        }
        if (service_list.Count == 1)
        {
            return service_list[0];
        }
        foreach (IService j in service_list) //This will search for the
         ↪  device with the best RSSI so it connects to the best one
        {
            try
            {
                if (j != null)
                {
                    if (j.Device.Rssi > maxrssi)
                    {
                        service = j;//There can be only one!
                        maxrssi = j.Device.Rssi;
                    }
                }
            }
```

```
                catch (Exception e)
                {
                    Debug.WriteLine("There are no services found with the
                     ↪  query");
                    Debug.WriteLine(e);
                }
            }
        }
        else
        {
            Debug.WriteLine("No connected devices");
        }
        return service;
    }
    /// <summary>
    /// Connects to all known devices it has found inside the database.
    /// </summary>
    /// <param name="adapter">The bluetooth adapter used to connect</param>
    public static async void ConnectToKnownDevices(IAdapter adapter)
    {
        if (adapter == null)
        {
            throw new NotImplementedException("No IAdapter found.");
        }
        Debug.WriteLine("Starting to connect to known devices");
        var result = await DBO.GetKnownDevices(); //Queries the known devices
        if (result == null)
        {
            Debug.WriteLine("There are no known devices!");
            return; //break if there are no known devices
        }
        Debug.WriteLine("I got a list of known devices");
        foreach (var s in result) //goes over each device
        {
            try
            {
                Debug.WriteLine("Connecting to device: " + s.GuID.ToString());
                await adapter.ConnectToKnownDeviceAsync(s.GuID);
                Debug.WriteLine("Connected to device: " + s.GuID.ToString());
            }
            catch (DeviceConnectionException e)
            {
                Debug.WriteLine("Couldn't connect to known Device");
                Debug.WriteLine(e);
            }
        }
    }
    /// <summary>
    /// Send the datetime to the device
    /// </summary>
    /// <param name="device">The device we use for the network</param>
    /// <returns></returns>
    public static async Task SendTime(IDevice device)
    {
        if (device == null)
        {
            throw new NotImplementedException("No IDevice found."); //throw not
             ↪  implemented, a device need to be used
        }
        var serviceguid = "00001805-0000-1000-8000-00805F9B34FB"; //The service
         ↪  id for the datetime
        var characteristicid = "00002A2B-0000-1000-8000-00805F9B34FB"; //The
         ↪  characteristic id for the datetime
        var time = DateTime.Now.ToLocalTime(); //Current time on the phone
        Debug.WriteLine("Current time: "+time.ToString());

        var characteristic = await GetCharacteristic(device, serviceguid,
         ↪  characteristicid);

        if(characteristic.CanWrite) //Only if we can write
        {
            var bytes = System.Text.Encoding.ASCII.GetBytes(time.ToString());
            await characteristic.WriteAsync(bytes);
            Debug.WriteLine("Succesfully written datetime to device!");
        }
        else
        {
            Debug.WriteLine("We can't write to this characteristic!");
        }
    }
    /// <summary>
    /// Searches the right characteristic on a device
    /// </summary>
    /// <param name="device">The device that he is searching</param>
    /// <param name="serviceid">The service id</param>
    /// <param name="characteristicid">The characteristic id</param>
    /// <returns></returns>
    public static async Task<ICharacteristic> GetCharacteristic(IDevice device,
     ↪  string serviceid, string characteristicid)
    {
        if (device == null)
        {
            throw new NotImplementedException("No IDevice found."); //throw not
             ↪  implemented, a device need to be used
        }
        if (serviceid == null)
        {
            throw new NotImplementedException("No IService found."); //throw not
             ↪  implemented, a device need to be used
        }
        if (characteristicid == null)
        {
            throw new NotImplementedException("No ICharacteristic found.");
             ↪  //throw not implemented, a device need to be used
        }
        ICharacteristic characteristic = null;
        IService service;
        try
        {
            service = await device.GetServiceAsync(Guid.Parse(serviceid)); //Get
             ↪  service if possible
        }
```

```
            catch (Exception e)
            {
                Debug.WriteLine("No Service found");
                Debug.WriteLine(e);
                return characteristic;
            }
            try
            {
                characteristic = await
                 ↪  service.GetCharacteristicAsync(Guid.Parse(characteristicid));
                 ↪  //Get characteristic if possible
            }
            catch (Exception e)
            {
                Debug.WriteLine("No characteristic found");
                Debug.WriteLine(e);
                return characteristic;
            }
            return characteristic;
        }
        public static async Task GetAcces(IDevice device)
        {
            if (device == null)
            {
                throw new NotImplementedException("No IDevice found."); //throw not
                 ↪  implemented, a device need to be used
            }
            var serviceguid = "0000180A-0000-1000-8000-00805F9B34FB"; //The service
             ↪  id
            var characteristicid = "00002A00-0000-1000-8000-00805F9B34FB"; //The
             ↪  characteristic
            var characteristic = await GetCharacteristic(device, serviceguid,
             ↪  characteristicid);
            var password = "Phone";

            if (characteristic.CanWrite) //Only if we can write
            {
                var bytes = System.Text.Encoding.ASCII.GetBytes(password);
                await characteristic.WriteAsync(bytes);
                Debug.WriteLine("Succesfully written password to device!");
            }
            else
            {
                Debug.WriteLine("We can't write to this characteristic!");
            }

        }
    }
}
```

B
The Monitor class

```
using Plugin.BLE;
using Plugin.BLE.Abstractions.Contracts;
using System;
using System.Diagnostics;
using Xamarin.Forms;

namespace BAP_App
{
    /// <summary>
    /// The heart and soul of the sensor data. All Monitors should be static. That
     ↪  way every monitor has single value and label.
    /// The data is used for measurements.
    /// The Label is used for visualization.
    /// </summary>
    public class Monitor
    {
        /// <summary>
        /// The label of said monitor. If it is public the layout can be changed for
         ↪  every monitor.
        /// </summary>
        public Label BigLabel = new Label();
        public Label SmallLabel = new Label();
        /// <summary>
        /// The constructor of the monitors. Should be used as a static
        /// </summary>
        /// <example>public static Monitor ExampleMonitor = new Monitor();</example>
        public Monitor(string name)
        {
            Name = name;                //The name is used for page creation
            ServiceName = name;
            CharacteristicName = name;
            App.MonitorList.Add(this); //Adding this monitor to the monitor list
            InitPage();
        }
        public Monitor(string name, string servicename, string characteristicname)
        {
            Name = name;                //The name is used for page creation
            ServiceName = servicename;
            CharacteristicName = characteristicname;
            App.MonitorList.Add(this); //Adding this monitor to the monitor list
            InitPage();
        }
        public readonly string Name;
        public string ServiceName; //This name will search the database for the
         ↪  service. So a GUID can be searched
        public DataPage Page;
        public string CharacteristicName; //This name will search the database for
         ↪  the characteristic
        private byte[] privatevalue;                                       //Private
         ↪  byte[] so it can be worked with only inside this class
        public byte[] Value                                            //the actual
         ↪  global variable HeartRateValue
        {
            get
            {
                if (privatevalue == null)
                    ↪  //It doesn't accept null so if it doesn't measure anything it
                    ↪  will return 0 (although this will probably never happen)
                {
                    return System.Text.Encoding.ASCII.GetBytes("0");
                }
                else
                {
                    return privatevalue;
                     ↪  //just return the normal value
                }
            }
            set
            {
                if (value == privatevalue)
                {
                    Debug.WriteLine("Value remains unchaged");             //The set
                     ↪  value is the same as the old value, no change is needed
                }
                else
                {
                    privatevalue = value;
                     ↪  //the internal value is now the measured vallue
                    Debug.Write("Value changed to: " +
                     ↪  System.Text.Encoding.ASCII.GetString(value)); //debugging
                    ChangeValue();
                }
            }
        }
        /// <summary>
        /// If the value is changed so should be the label. The label needs to be
         ↪  invoked on the main thread. Otherwise it would not update.
        /// </summary>
        /// <param name="label"></param>
        public void ChangeValue()
        {
            var text = System.Text.Encoding.ASCII.GetString(this.Value);
             ↪  //byte[] to string
            Device.BeginInvokeOnMainThread(() => this.BigLabel.Text = text);
            Device.BeginInvokeOnMainThread(() => this.SmallLabel.Text = text);
        }
```

```
/// <summary>
/// Initializes the page and the layout
/// </summary>
private void InitPage() //initialization of the new datapage
{
    Page = new DataPage(this.Name, this);
    BigLabel.FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label));
    BigLabel.VerticalOptions = LayoutOptions.CenterAndExpand;
    BigLabel.HorizontalOptions = LayoutOptions.Center;
    SmallLabel.Text = "Unavailable";
}
public bool updating;
/// <summary>
/// Gets the value using functions inside of the BLE class. This overload
 ↪  skips the check if the value is already updating. It just tries to update
 ↪  the value
/// </summary>
/// <see cref="BLE"/>
public void GetValue()
{
    GetValue(false);
}
/// <summary>
/// Gets the value using functions inside of the BLE class.
/// </summary>
/// <param name="alwaysstarting">If true: tries to start anyway, without
 ↪  looking if the app is already updating</param>
/// <see cref="BLE"/>
public async void GetValue(bool alwaysstarting)
{
    var adapter = CrossBluetoothLE.Current.Adapter; //Uses the bluetooth
     ↪  adapter
    var db = DBO.GetDatabase(); //get database
    var connected_devices = adapter.ConnectedDevices; //Get a list of
     ↪  connected devices
    if (connected_devices.Count == 0 || connected_devices == null) //Check if
     ↪  there are any
    {
        if (Page.Update.Text == "Stop")
        {
            Page.Update.Text = "Start";
        }
        else
        {
            await Page.DisplayAlert("Alert", "There are no connected
             ↪  devices!", "OK");
            Debug.WriteLine("There are no connected devices!");

        }
        return; //if there are no connected devices, close the action
    }
    BService bservice; //Get the service used in the BAN
    BCharacteristic bcharacteristic; //Get the characteristic used in the BAN
    IService service = null;
    ICharacteristic characteristic = null;
    try //If it doesn't find a service it exits the code
```

```
{
    var query1 = db.Table<BService>().Where(s => s.Name ==
     ↪  this.ServiceName);
    bservice = await query1.FirstAsync();
    Debug.WriteLine("Found bservice:
     ↪  "+bservice.Name+":"+bservice.StringId);
}
catch (Exception e)
{
    Debug.WriteLine("Didn't find a service for " + this.Name + "...");
    Debug.WriteLine(e);
    return;
}
try //If it doesn't find a characteristic it exits the code
{
    var query2 = db.Table<BCharacteristic>().Where(s => s.Name ==
     ↪  this.CharacteristicName);
    bcharacteristic = await query2.FirstAsync();
    Debug.WriteLine("Found bcharacteristic: " + bcharacteristic.Name +
     ↪  ":" + bcharacteristic.StringId);
}
catch (Exception e)
{
    Debug.WriteLine("Didn't find a characteristic for " + this.Name +
     ↪  "...");
    Debug.WriteLine(e);
    return;
}

Debug.WriteLine("Starting to get " + this.Name + " Value!");
 ↪  //Debugging
try
{
    service = await BLE.SearchService(adapter, bservice.GuID);
     ↪  //Searches for the service
}
catch(Exception e)
{
    Debug.WriteLine(e);
}
if(service == null)
{
    await Page.DisplayAlert("No Service found!","There are no services
     ↪  found. The database may be incomplete.","Cancel");
    return;
}
Debug.WriteLine("The service I found for " + this.Name + " is: " +
 ↪  service.Id.ToString());                            //Debuggin
characteristic = await BLE.SearchCharacteristic(service,
 ↪  bcharacteristic.GuID);   //Searches for the characteristic
if(characteristic == null)
{
    await Page.DisplayAlert("No Characteristic found!", "There are no
     ↪  characteristics found. The database may be incomplete.",
     ↪  "Cancel");
    return;
```

```
            }
            Debug.WriteLine("The characteristic I found for " + this.Name + " is: " +
             ↪  characteristic.Id.ToString());              //Debuggin
            if (alwaysstarting)
            {
                await BLE.UpdateValueStart(characteristic, this);
                Page.Update.Text = "Stop"; //If the value is starting to update it
                 ↪  will change the text to stop.
            }
            else
            {
                if (!updating)
                 ↪  //Checks if it needs to be started or stopped
                {
                    await BLE.UpdateValueStart(characteristic, this);
                     ↪  //Starts updating
                    Page.Update.Text = "Stop"; //If the value is starting to update
                     ↪  it will change the text to stop.
                }
                else
                {
                    await BLE.UpdateValueStop(characteristic, this);
                     ↪  //Stops updating
                    Page.Update.Text = "Start"; //If the value is stopping to update
                     ↪  it will change the text to start.
                }
            }
        }
    }
}
```

C
The SQLite database

```
using Plugin.BLE.Abstractions.Contracts;
using SQLite;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks;

namespace BAP_App
{
    /// <summary>
    /// Class used for database operations
    /// </summary>
    class DBO
    {
        /// <summary>
        /// Creates a list of known devices. It finds these inside the database.
        /// </summary>
        /// <returns>A list of known devices</returns>
        public static async Task<List<KnownDevice>> GetKnownDevices()
        {
            var db = GetDatabase();

            var query = db.Table<KnownDevice>(); //make a query for the knowndevice
             ↪ table

            var result = await query.ToListAsync(); //Make a list of the query
            return result;
        }
        /// <summary>
        /// Deletes all known devices inside the KnownDevice table in the database
        /// </summary>
        /// <returns></returns>
        public static async Task DeleteKnownDevices()
        {
            var db = GetDatabase();
            var query = "DELETE FROM KnownDevice";
            try
            {
                await db.ExecuteAsync(query);
                Debug.WriteLine("Succesfully deleted known devices.");
            }
            catch (Exception e)
            {
                Debug.WriteLine("Couldn't delete known devices.");
                Debug.WriteLine(e);
            }
        }
        /// <summary>
        /// Gets the database using SQLite, here is also the databasePath defined
        /// </summary>
        /// <returns>The database used for data operations</returns>
        public static SQLiteAsyncConnection GetDatabase()
        {
            // Get an absolute path to the database file
            var databasePath =
                ↪  Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),
                ↪  "MyData.db"); //Creates the folder or saves

            var db = new SQLiteAsyncConnection(databasePath); //creates the database
             ↪  at the folder
            return db;
        }
        /// <summary>
        /// Initializing of the database and creating the table "If not exists"
        /// </summary>
        public static async void InitBLEDatabase()
        {
            //Initializes the database and creates the first table used for bluetooth
            var db = GetDatabase();
            await db.CreateTableAsync<KnownDevice>(); //Creates a table for known
             ↪  devices
            await db.CreateTableAsync<BService>();
            await db.CreateTableAsync<BCharacteristic>();
            InitGUID(); //saves all the services and characteristics
        }
        /// <summary>
        /// Saves a device inside the database as "KnownDevice". This can be used
            ↪  later for reconnects. It saves both the name and guid. For now nothing
            ↪  else but that can be configurable.
        /// </summary>
        /// <param name="device">The Devices you want to be saved</param>
        public static async void SaveDevice(IDevice device)
        {
            //Saves a device into the knowndevice table
            var db = GetDatabase();
            bool check = await CheckKnownDevices(device);
            if (!check)
            {
                var new_device = new KnownDevice() //The device will be saved as a
                 ↪  KnownDevice
                {
                    Name = device.Name,
                    GuID = device.Id
                };
                await db.InsertAsync(new_device);
                Console.WriteLine("Auto Device id: {0}", new_device.Id);
            }
            else
            {
                Debug.WriteLine("Device is already known!");
            }
        }
        /// <summary>
        /// Checks if a device is already known. Returns a true if it is, false if it
         ↪  isn't
        /// </summary>
        /// <param name="device">The device you want to check</param>
        /// <returns>True if device is known, False if unknown</returns>
        public static async Task<bool> CheckKnownDevices(IDevice device)
        {
            //checks if a device is already saved in a database
            var result = await GetKnownDevices();

            foreach (var s in result) //check each item of the query
            {
                if (device.Id == s.GuID) //if the device Id is already inside the
                 ↪  known devices
                {
                    return true; //it is already known
                }
            }
            return false; //it is unkown
        }
        /// <summary>
        /// Initializes the known services and characteristics
        /// </summary>
        /// <param></param>
        public static async void InitGUID()
        {
            var db = GetDatabase();
            var bservice = new List<BService>();
            var bcharacteristic = new List<BCharacteristic>();
            bservice.Add(new BService()
            {
                Name = "Test",
                StringId = "00001809-0000-1000-8000-00805f9b34fb"
            });
            bcharacteristic.Add(new BCharacteristic()
            {
                Name = "Test",
                StringId = "00002a4e-0000-1000-8000-00805f9b34fb"
            });
            bservice.Add(new BService()
            {
                Name = "HeartRate",
                StringId = "0000180D-0000-1000-8000-00805F9B34FB"
            });
            bcharacteristic.Add(new BCharacteristic()
            {
                Name = "HeartRate",
                StringId = "00002A37-0000-1000-8000-00805F9B34FB"
            });
            bservice.Add(new BService()
            {
                Name = "Motion",
                StringId = "00001819-0000-1000-8000-00805F9B34FB"
            });
            bcharacteristic.Add(new BCharacteristic()
            {
                Name = "Motion",
                StringId = "00002A67-0000-1000-8000-00805F9B34FB"
            });

            await db.InsertAllAsync(bservice);
            await db.InsertAllAsync(bcharacteristic);
        }
    }
    /// <summary>
    /// The format for the saved devices, this is used for reconnects
    /// </summary>
    public class KnownDevice
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; } //This is the index id for the database NOT the
         ↪  id of the device
        public string Name { get; set; } //This is the name of the device
        public Guid GuID { get; set; } //This is the guid of the DEVICE this is
         ↪  constant
    }
    /// <summary>
    /// Our saved services
    /// </summary>
    public class BService
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; } //This is the index id for the database NOT the
         ↪  id for the device
        public string Name { get; set; } //The name of the service
        private Guid guid;
        public Guid GuID
        {
            get
            {
                return guid;
            }
            set
            {
                if (value != guid)
                {
                    guid = value;
                    StringId = guid.ToString(); //also sets the string ID. So you
                     ↪  don't need to convert is later
                }
            }
        }
        public string StringId
        {
            get
            {
                return stringid;
            }
            set
            {
                if(value!=stringid)
                {
                    stringid = value;
                    guid = Guid.Parse(value);
                }
            }
        }
        private string stringid;
    }
    /// <summary>
    /// Our saved characteristics
    /// </summary>
    public class BCharacteristic
    {
        [PrimaryKey, AutoIncrement]
        public int Id { get; set; } //This is the index id for the database NOT the
         ↪  id for the characteristic
        public string Name { get; set; } //The name of the characteristic
        private Guid guid;
        public Guid GuID
        {
            get
            {
                return guid;
            }
            set
            {
                if (value != guid)
                {
                    guid = value;
                    StringId = guid.ToString(); //also sets the string ID. So you
                     ↪  don't need to convert is later
                }
            }
        }
        public string StringId
        {
            get
            {
                return stringid;
            }
            set
            {
                if (value != stringid)
                {
                    stringid = value;
                    guid = Guid.Parse(value);
                }
            }
        }
        private string stringid;
        public byte[] Value { get; set; } //the saved value of the characteristic
    }
}
```

D
The Notification Manager

```
using System;
using Android.App;
using Android.Content;
using Android.Graphics;
using Android.OS;
using Xamarin.Forms;
using Android.Support.V4.App;
using AndroidApp = Android.App.Application;

[assembly: Dependency(typeof(BAP_App.Droid.AndroidNotificationManager))]
namespace BAP_App.Droid
{
    /// <summary>
    /// Set up notifications to be used in the Android application
    /// </summary>
    public class AndroidNotificationManager : INotificationManager
    {
        const string channelId = "default";
        const string channelName = "Default";
        const string channelDescription = "The default channel for notifications.";
        const int pendingIntentId = 0;

        public const string TitleKey = "title";
        public const string MessageKey = "message";

        bool channelInitialized = false;
        int messageId = -1;
        NotificationManager manager;

        public event EventHandler NotificationReceived;

        public void Initialize()
        {
            CreateNotificationChannel();
        }

        public int ScheduleNotification(string title, string message)
        // Triggered when the notifcation needs to be send
        {
            if(!channelInitialized)
            {
                CreateNotificationChannel();
            }

            messageId += 1;

            Intent intent = new Intent(AndroidApp.Context, typeof(MainActivity));
            intent.PutExtra(TitleKey, title);
            intent.PutExtra(MessageKey, message);

            PendingIntent pendingIntent =
             ↪  PendingIntent.GetActivity(AndroidApp.Context, pendingIntentId,
             ↪  intent, PendingIntentFlags.OneShot);

            NotificationCompat.Builder builder = new
             ↪  NotificationCompat.Builder(AndroidApp.Context, channelId);
            builder.SetContentIntent(pendingIntent);
            builder.SetContentTitle(title);
            builder.SetContentText(message);

             ↪  builder.SetLargeIcon(BitmapFactory.DecodeResource(AndroidApp.Context.Resources,
             ↪  Resource.Drawable.ic_stat_button_click));
            builder.SetSmallIcon(Resource.Drawable.ic_stat_button_click);
            builder.SetDefaults((int)NotificationDefaults.Sound |
             ↪  (int)NotificationDefaults.Vibrate);

            Notification notification = builder.Build();
            manager.Notify(messageId, notification);

            return messageId;
        }

        public void ReceiveNotification(string title, string message)
        //Triggerd when a notification is received
        {
            NotificationEventArgs args = new NotificationEventArgs();
            args.Title = title;
            args.Message = message;

            NotificationReceived?.Invoke(null, args);
        }

        void CreateNotificationChannel()
         //Setup channel through which notifications flow
        {
            manager =
             ↪  (NotificationManager)AndroidApp.Context.GetSystemService(AndroidApp.NotificationS

            if(Build.VERSION.SdkInt >= BuildVersionCodes.O)
            {
                Java.Lang.String channelNameJava = new Java.Lang.String(channelName);

                NotificationChannel channel = new NotificationChannel(channelId,
                 ↪  channelNameJava, NotificationImportance.Default);
                channel.Description = channelDescription;
                manager.CreateNotificationChannel(channel);
            }

            channelInitialized = true;
        }
    }
}
```