The implementation of an MBAN gateway

Bachelor Thesis

Jeroen Vermeulen & Tarik Benaich

Technische Universiteit Delft

TU Delft

The implementation of an MBAN gateway

Bachelor Thesis

Students:	Jeroen Vermeulen	4694864
	Tarik Benaich	4609980
Supervisors:	Prof. Dr. Ir. Said Hamdioui	TU Delft
	Dr. Ir. Rajendra Bishnoi	TU Delft
Project proposer:	Dr. Ir. Christos Strydis	Erasmus MC
Daily supervisors:	Ir. Muhammed Ali Siddiqi	Erasmus MC
	Dr. Ir. Anteneh Gebregiorgis	TU Delft

June 18, 2021

TU Delft

Abstract

Epilepsy is a severe neurological disorder that affects every aspect of a patient's life. Unfortunately, there is no complete cure for everyone on the market yet. However, a lot of work has been done on seizure prevention. The entire project details the proof of concept implementation of a secure and reliable MBAN (Medical Body Area Network) used for seizure prevention. The principal objective of the MBAN system is to set up and maintain secure connections between the nodes of the MBAN system and store and analyze the received data in the cloud. Therefore, a suitable gateway is needed, which is created in this work. The gateway concerns a mobile application constructed with the Flutter SDK. The main ability of the applicaton is to communicate with the implantable medical device, which in the demonstration is the node the gateway is connected to using BLE. The application is designed for Android and iOS and is connected to the AWS cloud service in which the data is stored and analyzed with a simple function that checks whether the received heart rate is above a certain threshold. This function can be easily replaced by a more extensive function. In addition, the application displays user health metrics such as the heart rate, connection state, and it can update the firmware of the implantable medical device. The security measures taken in this project concern setting up the BLE connection with an OOB (Out Of Band) channel for key sharing, after which the key is used to encrypt the data streams. Additionally, the data in the cloud is encrypted.

Preface

This work is a thesis written in light of the Bachelor Graduation Project of Electrical Engineering. The project was commissioned by the Neuroscience department of the Erasmus University Medical Center Rotterdam. This project aims to design a proof of concept implementation of a secure and reliable MBAN system used for seizure prevention.

We want to show our appreciation to our daily supervisors Ir. Muhammad Ali Siddiqi and Dr. Ir. Anteneh Gebregiorgis, our project supervisors: Prof. Dr. Ir. Said Hamdioui and Dr. Ir. Rajendra Bishnoi and to Dr. Ir. Christos Strydis who was our project proposer. And we would also like to thank Prof.Dr.Ir. W.A. (Wouter) Serdijn for being our neutral assessor. At last we want to thank our collegues: Erik Speksnijder, Fedde van der Meer, Ismail Bourhial and Saul Pennings. It was a pleasure to work with all of you.

Jeroen Vermeulen & Tarik Benaich
Delft, June 2021

Contents

Nomenclature

API	Application Program Interface
AWS	Amazon Web Services
BaaS	Backend As A service
BLE	Bluetooth Low Energy
CRUD	Create,Read,Update,Delete
GATT profile	Generic Attribute profile
GUI	Graphical User interface
IMD	Implantable Medical Device
IoT	Internet of Things
MBAN	Medical Body Area Network
MVP	Minimum Viable Product
MPU	Memory Protection Unit
NFC	Near Field Communication
OOB channel	Out-Of-Band channel
RNG	Random Number Generator
REST	Representational State Transfer
UI	User Interface
UUID	Universally Unique Identifier
SQL	Structured Query Language

1 Introduction

Epilepsy refers to a collection of disorders in the brain where the activity becomes abnormal, this is affecting millions of people worldwide. The abnormal brain activity can lead to seizures which are a significant discomfort in a patients life and can even lead to dangerous situations, for example falling near a set of stairs. Fortunately, some medications can prevent seizures from taking place. These medications generally work for 7-8 out of 10 people [1]. A prevalent hypothesis in the field of neuroscience says that another solution is possible. With the help of a Wireless Medical Body Area Network (MBAN), consisting of different sensors and an actuator, seizures can be timely prevented. Through the actuator, the seizure can be prevented from happening [1].

The MBAN approach

Epileptic seizure is a system-wide phenomenon, and because of this, it can be observed with the help of different sensors. For example, sensors that measure: an elevated heart rate, sweating, rapid muscle tone and synchronized neuron firing. With this in mind, the solution of an MBAN was proposed by the Neuroscience department of the Erasmus MC. The designed MBAN for this project consists of a sensor network of implantable, wearable and portable wireless sensors, a gateway connecting to the sensors and a cloud application for data storage and analytics. These implantable, wearable or portable sensors can range from EEG sensors to heart rate sensors. The wireless communication between the nodes and the smartphones improves usability but introduces many security challenges. These challenges push for a design preventing malicious entities from accessing the data of the patient or controlling the implant [2].

State-of-the-art analysis

With the development of the Internet Of Things (IoT), which is a network of embedded sensors, the implementation of an MBAN has become feasible over the last few years. As a result of the technological development of IoT, it enabled the sensors to become smaller and consume significantly less energy than before [3], which the MBAN leverages. Another helpful development is the widespread use of smartphones by people. Therefore, smartphones serve as a useful gateway to interface with the sensor network. By using Bluetooth Low Energy (BLE) as the connection between the sensors nodes and the gateway, a network can be established [4]. Next to the research, there are also other implementations of MBAN systems, for example, the Hexoskin [5]. Which is a smart shirt for measuring vital signs of a user, it is also capable of real time monitoring of these vital signs on a smartphone.

Project goal

This thesis will describe the process of implementing a gateway in an MBAN. This can be done in a few different ways, but it is requested to be an (Android) application. The application should blend in with the rest of the MBAN system, which is being developed at the same time by other thesis groups. The system overview is depicted in Figure 1.1 (where the colours represent the different groups). Task 1 is implementing a secure pairing sequence between an MBAN node and the Gateway, task 2 is

[1] In other literature, the use of the words MBAN and WBAN are commonly interchanged

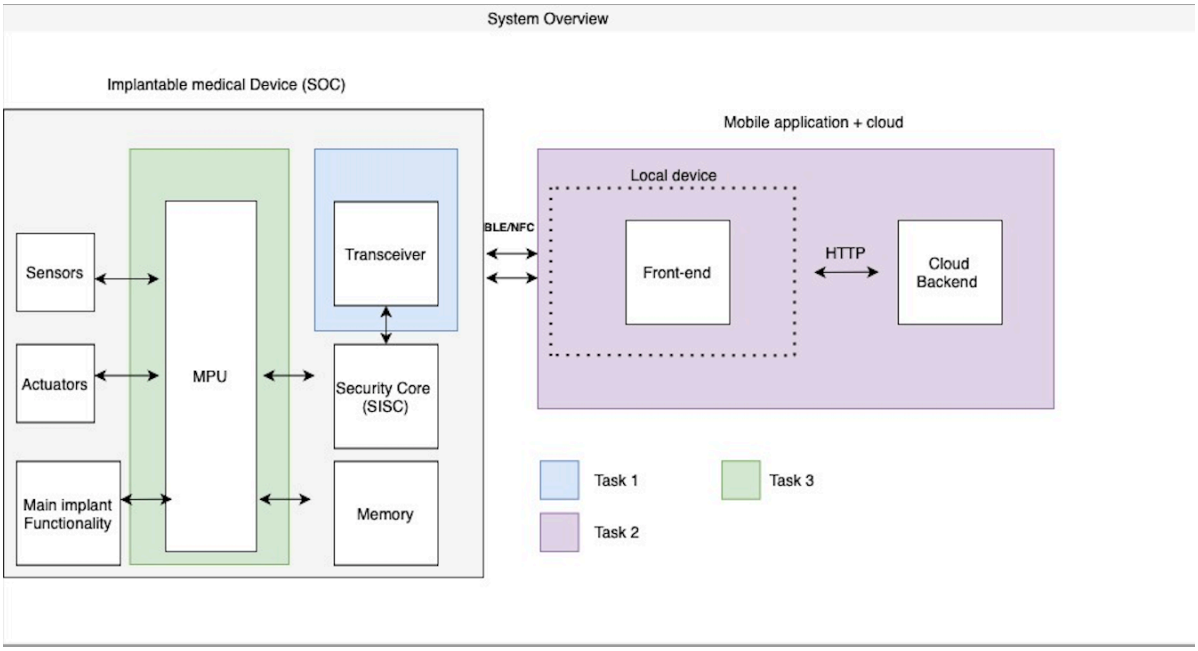

Figure 1.1: System overview

implementing an MBAN gateway and at last task 3 will be implementing a secure bus-architecture design for MBAN nodes. At the end of the project these three tasks will be combined to complete a proof of concept implementation of a secure and reliable MBAN.

The application can be split up into: the front end, the API and the cloud application. The front end is what the user interacts with and implements the Graphical User Interface (GUI). Next to this the front end manages the Bluetooth Low Energy (BLE) and Near Field Communication (NFC) connection. The API is the Application Programming Interface which is embedded inside the cloud and the front end which defines the interaction between the front end and the cloud. The cloud application which consists of the server which provides data on request, data processing applications and a database which organizes the information received from the sensors. All these different parts will be highlighted in this thesis and elaborated on how they were developed.

Thesis Structure

This thesis is structured as follows. First, the program of requirements is presented in chapter 2. Second, the front end and back end design processes are discussed in chapters 3 and 4, respectively. Subsequently, the prototype and the results are reviewed in chapter 5. Finally, the limitations of the development process and the most important conclusions are reviewed in chapters 6 and 7, respectively.

2

Program of Requirements

Mandatory requirements

The MBAN system must consist of the following requirements, otherwise the design is not acceptable. These requirements can be subdivided into two groups: functional requirements describe what the system offers in terms of features and functionalities. In contrast, non-functional requirements specify the quantifiable criteria to judge the operation of a system.

1. Functional requirements
   (a) A Graphical User interface Consisting of:
      - A display for Heart rate metrics of the user.
      - A display of the sensor battery levels.
      - An interface for the connection to the sensor.
      - Notifications in case of an error or a warning.
   (b) Ability to connect to a sensor via Bluetooth Low Energy (BLE).
   (c) Use OOB channel to receive an encryption key.
   (d) Ability to decrypt the received data and encrypt the send data.
   (e) Ability to update the firmware of the Memory Protection Unit on the implantable medical device.
   (f) A cloud service which stores and analyzes data.
   (g) A local back end which is in connection with a cloud service.
   (h) Ability to send data to the cloud.
2. Non-functional requirements
   (a) The project should be completed within 10 weeks.

Trade-off requirements

Trade of requirements will make the end user increasingly satisfied and the system should thus comply with these requirements as much as possible. Stated below are the trade-off requirements for the mobile application.

1. The GUI should be user friendly:
   (a) It should be intuitive.
   (b) it should be aesthetically pleasing
2. A minimal amount of user data should be used or transferred.

3

The front end

The front end is the presentation layer and concerns everything the user interacts with. It implements the GUI and manages the BLE and NFC connection. This chapter will first go over the construction of the front end and which tools are used, next to this it will be described how the application is designed.

3.1. Construction of the front end

3.1.1. The framework

Before the front end can be developed, a framework has to be chosen. A framework provides the foundation on which software can be developed, as it has predefined functions and classes. The front end framework had some requirements to which it should adhere. For instance the framework should be state of the art so that it will be supported for the coming years. As there was no prior knowledge about front end programming, it was required that the learning curve should be gentle and there should be sufficient documentation available. Three main options were found: Android SDK, React Native and Flutter. One of the latter two was chosen as Android SDK only provides support to Android and a cross-platform application will be able to support all users.

Flutter is created by Google and is an open-source UI software development kit for native Android and iOS application development. It is written in Dart, an object oriented programming language.

React Native is a JavaScript library used for building user interfaces. It uses React's framework capabilities in addition to some native platform capabilities.

Flutter was released in May of 2017, it is a fairly new framework when compared to React Native which was released in march of 2015. Flutter was chosen in the end, because the implementation of Bluetooth and NFC is easier due to the use of native Android and iOS development kit. Next to this Flutter is better in prototyping and delivering a final application [6].

3.1.2. Programming in Flutter

Flutter relies on widgets as building blocks, for instance UI elements, themes, styles and states are all managed in specific widgets. These widgets form a hierarchy based on composition, meaning that each widget is nested in its parent and can receive context from this parent. This structure is conserved in the whole application all the way to the root widget, which declares the application. Each widget has a build function which returns a new element tree (the visible part in the user interface). Next to that widgets also have a state, non mutable widgets are stateless while mutable widgets are stateful. These stateful widgets store the mutable state in a separate class, when this is mutated $setState()$ should be called. It should be noted that a stateless widget, so non-mutable, can have children which are stateful. This means only a small part of the parent widget will be updated instead of the whole widget.

3.2. Wireless connection

3.2.1. Bluetooth Low Energy

As stated in the requirements in Chapter 2, for the connection to a sensor, BLE needs to be used. BLE uses the Generic Attribute Profile (GATT), which is a set of rules to bundle, present and transfer data.

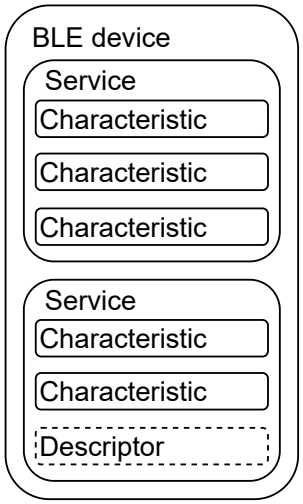

Figure 3.1: GATT profile of a BLE device

Figure 3.1 shows an example of the GATT profile for a BLE device. In the profile service characteristics and descriptors can be found. A service is a bundle of data accompanied by behaviors, designated for a particular function. Inside of a service, a characteristic and a descriptor can be found. The characteristic is where the data is presented and the descriptor gives additional details of the characteristics. A characteristic can be accessed through its service where each have a Universally Unique Identifier (UUID). There are two types of UUIDs: 16-bit versions which are predefined by the Bluetooth Special Interest Group and 128-bit versions which are custom made.

For the application two plugins were considered: flutter_blue [7] and flutter_reactive_ble [8]. For both plugins it was possible to read out data from a generic heart rate monitor, which proves that it is possible to read out sensor data. Flutter_reactive_ble is more extensive than flutter_blue, therefore it was harder to control the data model with the data streams coming in. As managing this data model is comprehensive while both extensions reach the same goal, flutter_blue was implemented. The available functions in Flutter_ble are: scan for devices, connect to a device and read, write or set notifications to listen for changes.

3.2.2. Out of Band Channel

BLE on itself is very vulnerable to eavesdropping, so sending the encryption key via BLE is not safe. To add an extra layer of protection an out-of-band channel is used. An out-of-band channel is a data transmission channel next to BLE and can take many forms. Three channels we considered were: galvanic coupling, ultrasound and NFC. As the first two channels would need a wired connection between the phone and a dedicated sensor these options are not ideal. An NFC reader however is implemented in every modern smartphone, this channel was thus chosen for its simple and time saving implementation. To read NFC tags the package nfc_manager is used [9]. The package is straightforward in use, it checks if NFC is turned on and then waits until a tag is read. It will then return the payload that is read.

3.3. Data model

To handle the data model created by the Bluetooth connection and the NFC reading, a state management plugin is necessary. Flutter provides some tools itself, but it was found that a plugin provides a few functions which save programming time. Following an example [10], which uses flutter_blue, $ScopedModel$ was chosen as the plugin as it was already proved to be working with the data from a Bluetooth connection. The plugin scoped_model [11] provides utilities to forward a data model from a parent widget down to its descendants. The model is incorporated in the root widget so that the data model is available in all widgets. In order to retrieve data the $ScopedModelDescendant$ widget is used. This widget provides the data and will rebuild the widget when notified. The use of this plugin simplifies how to forward data from a child widget through the root widget to another branch, this is referred to as app state management.

3.4. Channel encryption

To protect the data streams between the smartphone and the implantable medical device (IMD), encryption is used. Different types of encryption will be highlighted in Section 6.1. This encryption decrypts all the incoming data in the gateway and encrypts all the data sent out from the sensor node. The encryption had some requirements that need to be fulfilled, these are mainly related to the IMD as this has limited computing power. The encryption thus needs to be lightweight and small in size. It was found that the Simon block Cipher [12] is best suited for the use case. The code for the encryption is found in Appendix B.1. In combination with the block cipher, cipher block chaining is implemented. This will prevent hackers from seeing a pattern in the data send, the data will instead look like random noise. For the encryption to work, a few different functions were created: the key expansion, the encryption and the decryption. When the data is coming in it has the form of a list consisting of 8-bit integers of length eight, so in total 64-bits. These lists are first deconstructed and then they are used. The encryption and decryption use keys constructed in the key expansion, which uses predefined keys which need to be identical on both devices. Not all functions for the binary operations needed for the encryption are available in Dart, these were thus constructed to complete the encryption and decryption.

3.5. Front end design overview

To get a good understanding of the application's working, an understanding of all the widgets used is necessary. Section 3.1.2 describes how widgets work in Flutter and Figure 3.2 shows the implementation of this in the application.

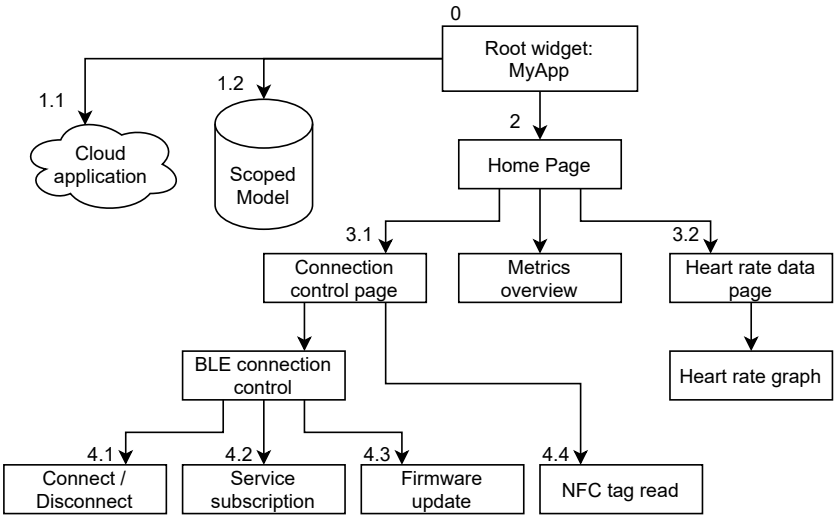

Figure 3.2: Overview of the widget hierarchy

An explanation of all the widgets shown in the widget hierarchy is given below:

0. The root widget, which is the base of the application.

1. The cloud application and scoped model, both linked to the root widget. Their data is available in all other widgets in the application.

2. The homepage, the first element of the GUI. Which shows some metrics and has the possibility to route the user to different pages.

3. Via the connection widget the connection control page is available and through heart rate metric widget the Heart rate data page is available.

4. In the connection control page the BLE connection control is available which consists of some sub widgets for connection management. In addition, the NFC tag read is also available in the same page.

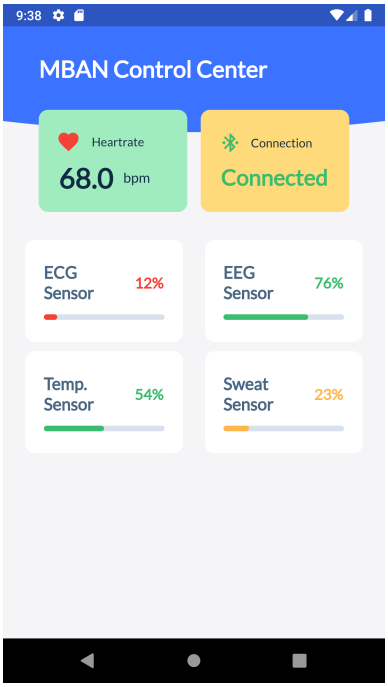

Figure 3.3: The homepage

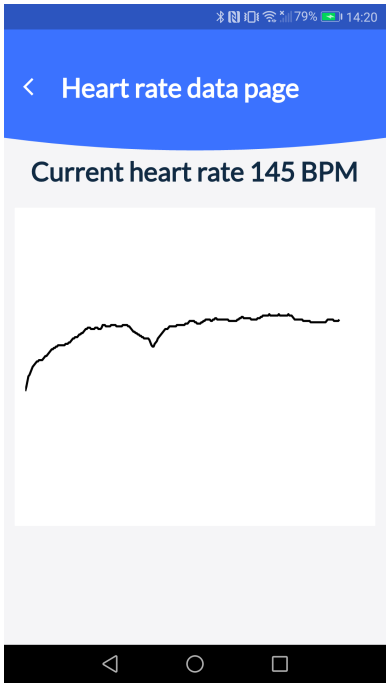

Figure 3.4: The heart rate data page

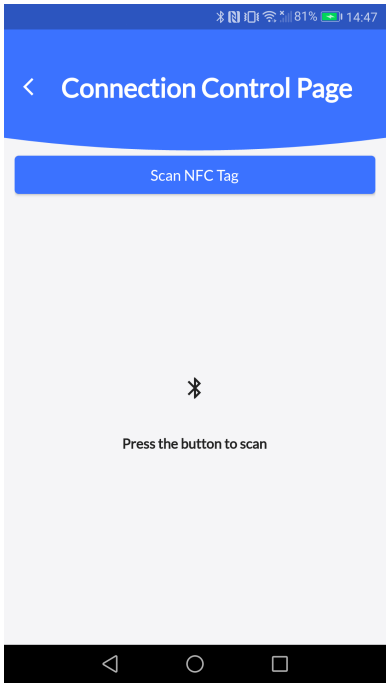

Figure 3.5: The connection control page

3.6. The Graphical User Interface

The requirements of the GUI are stated in Chapter 2. In short, the application must display certain metrics, can set up a BLE connection and show warnings or errors when they occur. Next to these functional requirements there are some trade-off requirements: it should be intuitive and aesthetically pleasing. As the project time is limited, an example of a medical application was used [13]. The outlines and colours of this example are used, but it is modified to fit the requirements.

The previews displayed in Figure 3.3 to 3.5 are screenshots of the Android version of the application. The screenshots of the iPhone version are shown in Appendix A.1.

The homepage, as shown in Figure 3.3, is presented when the application is opened. The homepage displays the following required metrics on the screen: the current heart rate, the connection status, and the battery levels of the sensors. The reason that they are shown on the homepage is to give a fast overview of a few of the data streams coming in via BLE and to keep the number of pages to a minimum.

The second page available in the application is the heart rate data page, as depicted in Figure 3.4. The page can be accessed by pressing the heart rate widget on the homepage. The page shows the current heart rate and a graph in the form of an oscilloscope [14] which plots the last 350 data points of the heart rate.

The third and last page, Figure 3.5, is the Connection Control Page. On this page, the user manages the Bluetooth connection. The user is able to: scan an NFC tag, scan for Bluetooth devices and connect to them, update the firmware of the IMD and disconnect from Bluetooth devices.

3.7. BLE connection setup

To set up a connection with a sensor, the connection page has to be opened and the button stating 'Press the button to scan' must be pressed. A list of found BLE devices will be shown and can be scrolled through, as visible in Figure 3.6. To connect to a specific device, its name should be pressed and the connection will be set up. There are three buttons at the bottom of the app, their functions from left to right: disconnect, firmware update and scan again. The two leftmost buttons will only be visible when connected to a device. The data on the home screen will only be presented when the predefined UUIDs match with the UUIDs of the sensor.

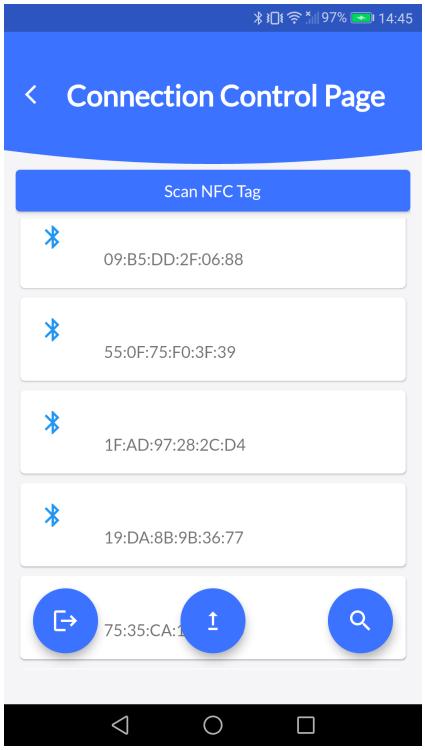

Figure 3.6: An overview of the connection control page when connected to a device

3.8. Firmware update

One of the requirements for the system is to be able to update the firmware on an IMD. The firmware update is an essential aspect of the BLE communication within the system as it is necessary to be able to update the protection table inside the memory protection unit (MPU). This is important as an implant cannot be taken out to update, so it has to be done wirelessly. In the communication with the IMD, 64-bit packages are used. To write to the MPU, a handshake is necessary so that only authorized users can update the firmware. This handshake is presented in the form of a diagram in Figure 3.7 where CMD stands for the 64-bit command containing the new description. $[CMD, NonceWrite]_k$ stands for the encrypted message, where the command sits in the bits 63 to 32 and the write nonce in the bits 31 to 0. The nonce, which is a random number generated by one side, is used as a reply by the other side. The nonce can only be sent once to ensure protection against replay attacks. With the nonce the handshake can thus not be controlled or redone by a third party. Both the reader and the writer have a nonce so that the handshake for both sides is secured. Next to the nonce to authenticate the other user, the X used in the handshake to authenticate the firmware update command. The handshake, as explained, is available in Appendix B.2. Because Dart works with signed integers, it is important to avoid problems due to large integers, for this BigInt is used as these can be called unsigned. Next to the channel encryption, the firmware update will be encrypted using an extra key only available to the system's admin so that the implant is maximally protected against unwanted actions.

3.9. Error management

Error management is done in a few different places. The smartphone needs to have NFC and Bluetooth enabled to scan for tags or devices. The application will issue a warning if the user has not enabled these functions. Figure 3.8 shows the alert dialog of an user not having enabled Bluetooth.

Next to this, the application will show an alert dialog when the firmware update is not completed correctly. It will show the reason for the error and in which step it occurred. This way, it is visible what went wrong. If the firmware update is completed correctly, the application will also send a notification. Finally, after data processing and if an anomaly occurs in the data, the application will receive a flag about the user's health from the cloud. After which, the application displays a notification that alarms

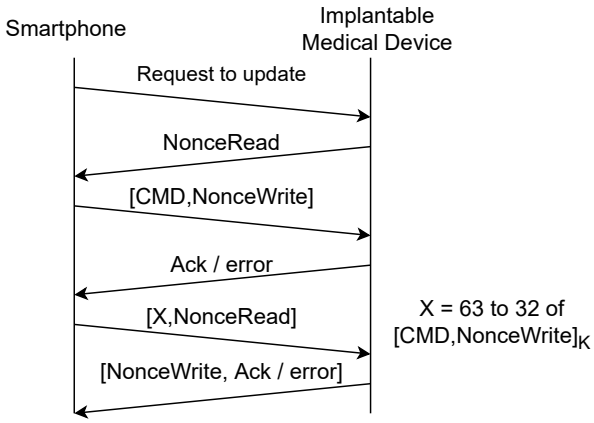

Figure 3.7: Diagram of the handshake needed to update the firmware on the IMD

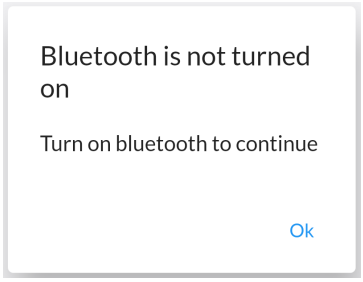

Figure 3.8: A warning notification given when Bluetooth is not turned on.

the user.

4

The back end

The function of a back end is to store and retrieve information from the database and enable business logic implementation. Business logic refers to the underlying processes within a system that carries out certain operations, such as heavy-duty data processing, calculations or storing information between the server and the user interface. In this section, the technology used for the back end will be discussed in detail.

4.1. The framework

There are several technologies available for designing the back end. In the scope of this project, we have limited the research only to the prevalent technologies, and these include the following frameworks:

- Flask/Django
- NodeJS
- BaaS Solutions (Google Cloud, Azure, AWS)

One of the requirements of the client was to perform data analytics. Therefore a framework written in Python was desired because Python has plenty of libraries and packages that facilitate data analytics implementations. Therefore, the NodeJS back end framework (written in JavaScript) was not an ideal option. As for Flask and Django frameworks (both written in Python), there is a steep learning curve that can be considered as an obstacle due to the limited time for the project. In contrast to a BaaS solution (Backend as a Service), which has a shallow learning curve because of its plug-and-play services. A Back end as a Service is a cloud service model in which the developer outsources all the underlying web/mobile application processes to the cloud provider. The preferred choice was to use a BaaS solution because it does not require advanced programming expertise, which is beneficial given the time scope for this project. In the following sections, the design of the back end with a BaaS solution will be discussed in depth.

4.2. Cloud computing

Cloud computing has been getting significant traction in the industry - and academic fields. There are multiple cloud definitions in different papers from different people. According to the European Union Agency for Cybersecurity ENISA publication: ”Cloud computing is an on-demand service model for IT-provision, often based on virtualization and distributed computing technologies” [15] According to the publication, the characteristics include:

- Highly abstracted resources
- Near instant scalability and flexibility
- Near instantaneous provisioning

- Shared resources (hardware, database, memory, etc)
- 'Service on demand', usually with a 'pay as you go' billing system
- programmatic management (e.g. through WS (Web Socket) API)

The three popular service cloud providers considered during this project are Azure, Google Cloud, and Amazon Web Services. Azure is a preferred choice for enterprises that already use Window's services, such as SharePoint and windows servers. Google cloud is relatively new to the market (2008), and with its experiences in machine learning tools and data centers, it was considered a good fit for this project. However, Amazon Web Services (AWS) has been on the market since 2002, and since then, it is the largest cloud provider in terms of market share (32%)[16]. Its popularity has another advantage; it is well documented and supported by the community. Given the benefits described earlier, it was decided to choose AWS as a back-end infrastructure. In figure 4.1 an overview of the back-end system is depicted.

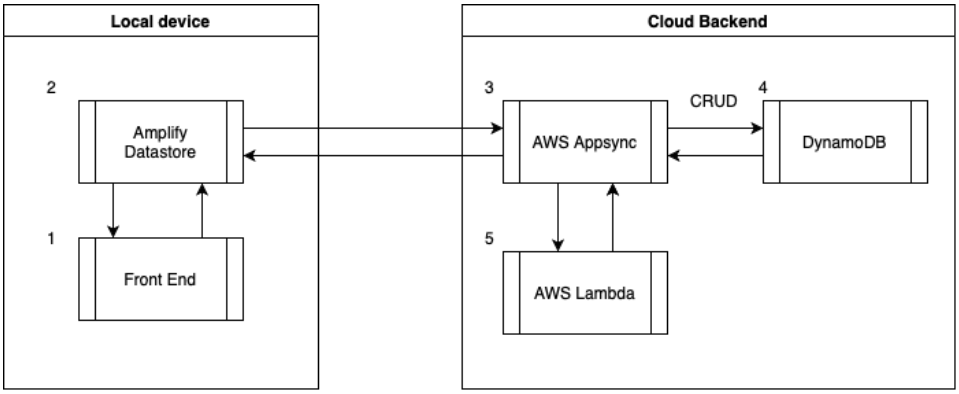

Figure 4.1: Overview of the back-end

1. The process begins when the mobile application receives data from another source, which can be, in this case, the sensor network.
2. The data from both online and offline users are stored locally in the Amplify Datastore component.
3. AWS AppSync receives the data, and the data is synchronized with other members, such as a doctor/relative.
4. Data is updated with CRUD operations into DynamoDB. CRUD operations are Create, Read, Update and Delete.
5. AWS Lambda provides the business logic, for example, a machine learning algorithm.

Amplify Datastore

Amplify DataStore is an on-device storage service that automatically synchronizes data between mobile/web applications and the database in the AWS cloud. It enables the programmer to create a data model which can be made via the AWS CLI or the AWS Admin UI (via the browser).

AWS Appsync

AWS Appsync is a service that allows the programmer to easily develop the GraphQL API which will be discussed in detail in 5.1

AWS Lambda

AWS Lambda is a serverless computing service that allows developers to create functions written in one of the supported languages. An example of a code that can be uploaded to the AWS Lambda environment is the seizure detection algorithm mentioned in the conclusion chapter. However, for this project, a simple function is created in which the heart rate stored in the database is fetched and used to check whether it exceeds a certain threshold integer value. The code can be found in appendix C.1.

4.3. Database

The two primary technologies when considering a database are relational or non-relational types. Both technologies have their advantages and disadvantages, which will be investigated in the following section.

4.3.1. Relational database

A relational database organizes and stores data into tables that can be linked to other tables through foreign keys (building relations). Relational databases are also known as SQL databases because they use SQL to query and manage the database. Relational databases are effective if the structure of the data is not changing.

4.3.2. Non-relational database

Unlike a relational database, a non-relational database allows storing unstructured data instead of using tables. A ”NoSQL” database originated in response to the limitations of existing (SQL) databases: NoSQL databases are capable of handling large amounts of structured, unstructured, semi-structured data with an amazing performance at reduced complexity and cost [17].

4.3.3. Choosing the database

AWS offers a fully managed NoSQL database, DynamoDB. Amazon DynamoDB is well-incorporated within Amazon’s cloud ecosystem, saving the developer time to set up the correct environment. Considering the time frame of this project, the decision was made to use DynamoDB as the database. After the decision of the database was made, a simple data model was created. The data model can be considered a model that organizes the database and standardizes how different data elements relate. For example, in table 4.2 the data model for this project is given, which stores information about the user.

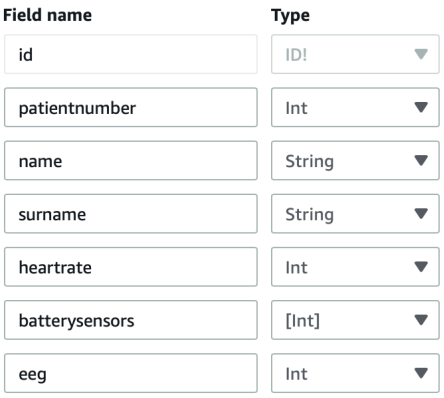

Figure 4.2: Structure of the data model

5 API

5.1. API

The Application Programming Interface (API) is necessary to establish a connection between the front end and back-end applications. The two applications communicate via a query architecture. Only the predefined queries designed by the programmer are sent to the server, and a response is sent from the server to the front-end. Therefore, an API can also be considered an extra layer toward enhancing data security; the data on the phone is never fully exposed to the server. Likewise, the data in the server is never fully exposed to a mobile phone. The two common architectures are GraphQL and REST API (Representational State Transfer). Facebook developed GraphQL to solve several difficulties when using conventional architectural styles such as REST. An example of a problem is that an application using the REST API cannot fetch a single entry in the database. Instead, with a REST API, the whole object is transmitted as a response. Over-fetching can introduce a high cost because unnecessary data bandwidth is utilized, which is undesirable when using a BaaS infrastructure with the ”Pay-as-you-go” model. GraphQL is a also well incorporated with AWS and therefore the GraphQL architecture is used for the communication between the front-end and the back-end.

6

Risk assessment Cloud Securtiy

Another enormous advantage of a BaaS infrastructure is partially outsourcing the implementation of data security. Within Amazon's ecosystem, security and compliances are shared responsibilities between AWS and the customer (developer). Shared responsibility means that the cloud provider is responsible for the security *of* the cloud (e.g., hardware infrastructure). At the same time, the customers (developers) are responsible for the security *in* the cloud. As shown in figure 6.1.

The primary data protection methods and approaches a customer can implement to mitigate risk and threats are implementing data encryption, elaborated in the following sections.

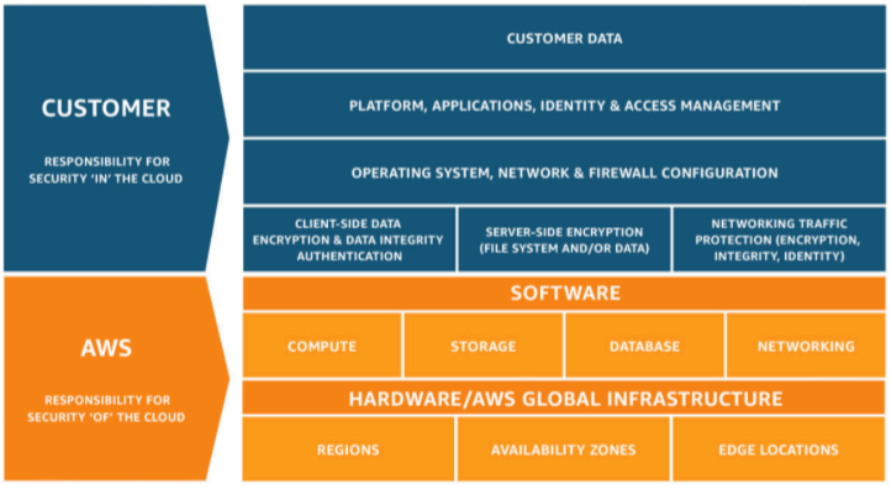

Figure 6.1: Application security is a shared responsibility between the cloud provider and the developer.[18]

6.1. Encryption in the back end

The cryptographic techniques are used for data encryption. The goal of encrypting data is to increase the level of data protection for assuring content integrity, authentication and availability [19]. The basic form of cryptography entails the transformation of plain text into cipher text using an encryption key. Then, at the receiver, the resulting cipher text is decrypted using a decryption key. There a different algorithms available such as:

- Advanced Encryption Standard (AES)
- Secure Hash Algorithm-3 (SHA-3)
- Rivest Cipher 4 (RC4)

- Riverst Shamir Adleman (RSA)

Each of these algorithms differs in computational complexity and key management approaches. The algorithms are either based on block ciphers, stream ciphers, or hashing techniques. In the following section, the principles behind these techniques are presented.

Block Ciphers

Block Ciphers is a method based on a cryptographic key and an algorithm applied to a block of data. Like many other encryption algorithms, AES is an example that uses the block ciphers technique to encrypt data. AES algorithm is considered the industry's standard because it is a highly secure algorithm, and at the time of writing this paper, it has not been breached[19].The only way this algorithm can be cracked is by brute force [20].

Stream Ciphers

In the stream ciphers technique one byte is encrypyted at a time. The stream ciphers' performance is faster than block ciphers because of the low hardware complexity. The RC4 algorithm is based upon stream ciphers, and it is regarded to be less safe than algorithms using block ciphers. For instance, RC4 is breakable if not implemented correctly. [19]

Hash

The hash function is a technique in which a mathematical function is implemented to convert the input data to an alphanumeric string [19]. In contrast to the encryption techniques described before, the hash function is a one-way encryption method if appropriately designed. Hashing serves a variety of purposes for example, password verification: Application with user login often requires a password which needs to be sent to the server for verification purposes. The sent password is converted to a hash value and is compared with the password stored in the database, which is also stored as a hash value. The SHA-3 is, at the time of writing, considered to be the recommended algorithm within the family of hash-based techniques. In comparison, the older version of the SHA familiy is either breached (SHA-1) or is expected to be breachable soon (SHA-2). [21].

6.1.1. Key management

Key exchange is an essential process within a system that is based on cryptographic techniques. A key can be an output of a mathematical function or essentially a random number produced by a random number generator. If a key is compromised, then the application is compromised as well. Typically, the length and the entropy of a key are positively proportional to the degree of resistance against attacks; the longer and more random a key, the more challenging it is to crack [22]. Therefore, a key should be created by a high-quality, ideally certified, Random Number Generator (RNG) for maximum protection. Generally, keys are distributed among a whole range of users to grant access to data stored in the back end, and that is not without risk. Therefore, it is essential to distinguish three types of keys that characterize how the keys are shared within a system these include:

- Symmetric Keys
- Asymmetric Keys
- Hash Keys

In the following paragraphs, these approaches will be discussed in detail.

Symmetric Keys

The Symmetric encryption methods require both the sender and receiver to use identical secret keys to decrypt data in the back end. This method works best for a closed system as it becomes harder for third parties to intervene. AES, is an example of an algorithm that is based upon the symmetric encryption method [22].

Asymmetric Keys

Public and private keys are involved in algorithms that are based on asymmetric encryption techniques. The asymmetric encryption technique uses the public key to encrypt data and the data can be decrypted by the owner of the private key. In terms of data privacy, anyone with a private key can impersonate the private-key-owner to decrypt data and gain unauthorized access to the application [22]. RSA is an example of an encryption algorithm that is based on assymetric keys.

Hash keys

As described earlier, hashing uses functions or algorithms to map object data (key) to an alphanumeric string (value) in a key-value pair stored in a hash table. Hash keys are used to ensure the integrity and authenticity stored in a database.[22]

6.1.2. Encryption used in the backend

All AWS cloud services used in this project encrypt the data by default with the AES-256 encryption technique. Nevertheless, the keys must be treated with care to prevent hacking. The client must ensure that key is only shared among selected designated individuals. Key distribution is possible with the AWS IAM (Identity & Access Management) service. This service enables the client to manage access to services and resources securely.

7

Prototype implementation and validation of the results

7.1. Overview of the prototype

The prototype will finalize the proof of concept implementation of a secure MBAN. This prototype will combine the three sub tasks the project consisted of. The first part of this prototype consists of secure pairing of the gateway and an MBAN node using an OOB channel. For the OOB channel NFC was chosen as this is readily available in modern smartphones and provides easy communication of the encryption key. To transfer the key an NFC card is emulated and read by the smartphone. As can be observed in this thesis a gateway will be delivered in the form of a smartphone application. At last a secure bus-architecture design for MBAN nodes is delivered in the form of an FPGA which simulates a sensor with a secure bus-architecture.

7.2. Functionalities of the prototype

The functionalities of the finalized prototype consist of the actions highlighted below. They focus mainly on the capabilities of the application running on the smartphone, as the development of which is described in this thesis. The smartphone application can:

- Connect to the FPGA using BLE.
- Decrypt the data stream, using the key read from the NFC card.
- Update the firmware of the Memory Protection Unit on the FPGA, using the admin key.
- Save and analyze the received data in the cloud.
- Notify users of anomalies in the analyzed data.
- Notify users of warnings and errors.

7.3. Connection

When connected to the FPGA, the application automatically connects to the UUID for the specified data. These UUIDs are custom because custom characteristics were needed for our data streams as standard charactersitics would not support our firmware update handshake. The services and characteristics have the following structure:

- Service 1: Data streams.
  - Characteristic 1: Heart rate value, Notifying channel.
  - Characteristic 2: Battery level of the heart rate monitor, Notifying channel.

- Service 2: Firmware Update.
  - Characteristic 1: Request and update data, Write channel.
  - Characteristic 2: Acknowledge and error values, Notifying channel.

For the data streams a notifying channel is best suited because a notification in the code will arrive when the value is updated instead of the need to actively read every few seconds. With this notification in the code some specific widgets will be updated with the new value. As for the firmware update a write channel is needed to write data to the IMD and a notifying channel will make sure the code reacts on the response received, so that the handshake can be completed.

7.4. Results

Prototype

After the firmware update the data streams start sending data, this is due to the fact that the MPU's protection table is now updated. This update gives the Bluetooth module read capabilities of the memory where the heart rate data and battery levels are stored. A few seconds after the data is received on the phone, it will be available in the cloud where it can be stored and processed. When a heart rate above 160 bpm is received the cloud will flag this as too high and a notification is send to the user.

If somewhere in the handshake, depicted in Figure 3.7, an error occurs, the application will give an alert dialog which shows in which step the error occurred. The alert dialog also shows what was received or what expected message was not received.

The gateway can perform create, delete, update and delete (CRUD) operations to the database which is stored in the cloud. The database consists of a simple data model which stores information of a patient, as illustrated in figure 4.2. The cloud can process the data from the database and forward the output to the front end via the GraphQL API. Additionally, the cloud is also capable of running heavy-duty algorithms such as a seizure detection algorithm presented in [23]. However, this detection function is not implemented in this project, as it will be explained in the discussion section. Instead, a function written in python is implemented in the cloud which examines whether the heart rate data exceeds a certain threshold and responds to the front end accordingly, the code can be found in appendix C.1. In terms of security, both the data at rest (stored in the cloud) and in transit (through the internet) are encrypted with the robust AES-256 algorithm by default.

8

Discussion

Evaluation of the requirements

In Section 2 a set of requirements was given to the system for it to be fully functional. As displayed in Section 7.4, the following requirements have been fulfilled by the prototype: a GUI is available displaying certain metrics, an interface for connecting to sensors and notifications will show in the event of an error. The ability to connect to sensors via BLE is available. Using an OOB channel an encryption key can be retrieved, which can be used to decrypt the incoming data streams from the sensors to the gateway and vice versa. The application has the ability to update the firmware of the protection table inside the MPU on the IMD. A cloud service is available which stores and analyzes data. This cloud service is connected to a local back end, from where the ability exists to send data to the cloud. Furthermore the application is build for Android and iOS which is an improvement over the requested Android application. Next to this all the GUI was requested to be user friendly, i.e. it should be intuitive and aesthetically pleasing. As these requirements are hard to quantify and will differ from person to person we asked our colleagues for feedback. It was found that the application is aesthetically pleasing but a manual when the application is first opened would give the support to every user to be able to use the application.

Security and Reliability

Security

As stated in the project title 'The proof of concept implementation of a secure and reliable MBAN for seizure prevention', the MBAN should be secure and reliable. The security in the system is realised in a few different ways, first of all the BLE data streams are encrypted and the encryption key is shared via an OOB channel. The combination of these two measurements renders man-in-the-middle attacks and eavesdropping impossible. The key sharing is done over NFC, this in itself is not safe as anybody is able to read NFC tags with their phone. But as NFC works only on a range of a few centimeters and the encryption uses the Simon cipher, which the hacker needs to know of before decrypting, the chance of being hacked is minimal. If a hack via this way occurs only the sensor data can be read, as for the firmware update a second key is necessary. Only the system administrator has access to this key. Next to the security on the BLE connection the cloud is encrypted with the robust AES-256 algorithm. The only way this algorithm can be cracked is by brute force.

Reliability

The reliability of the system is tackled in different ways. First, the code is debugged manually for many possible scenarios that can occur and will display errors when an error occurs. Examples of an error are when the heart rate exceeds a certain threshold or when the BLE connection with the FPGA is not established

Second, it is essential to allow data processing whenever the application is not displayed on the gateway. For instance, when the phone is locked, it would be beneficial to notify the user if a seizure might occur within a short time frame. In this prototype, background processing has not been implemented

due to time constraints. However, several options are available that allow the application to work in the background. This is discussed in the recommendation and future work section in chapter 9.

Cloud computing costs

Cloud computing is a rapidly developing and promising technology. It allows significantly accelerated application deployment, and it is characterized to be highly scalable. However, these resources are not free to use. Like many other cloud providers, Amazon Web Services comes with a pay-as-you-go model, which needs to be considered when this application may be further developed in the future. In this proof of concept, a free trial for a year was used and is limited to a certain amount of free usage.

9 Conclusion

In this report, the proof of concept implementation of an MBAN gateway is described. The gateway was implemented in the form of a mobile application, as required. The mobile application is connected with the sensors via BLE and by using NFC, key sharing necessary for the encryption is possible. Additional requirements are stated in Chapter 2; these requirements were the starting point from where the mobile application was developed. In Chapter 3 the design process of the front end is described by using the widget tree as displayed in figure 3.2. Then, chapter 4 explains the construction of the back end by using the AWS cloud platform. The backed services used are Amplify Datastore, AWS AppSync, DynamoDB and AWS Lambda. In the discussion, it is made clear that the implementation fills all the requirements stated in Chapter 2. Recognizing that all the mandatory requirements are fulfilled, it can be concluded that the constructed prototype is a working proof of concept implementation of an MBAN gateway.

Recommendations and future work

Scalability

The application is ready to be used by one user and can connect to one sensor as described in this thesis, a next step would be to include the possibility for multiple different users to use the application without any problems. For this a user profile inside the application is recommended which automatically links to an account in the cloud where only the personal data will be visible. Another recommendation is to be able to scale to multiple sensors, a single sensor is for the proof of concept enough, but in real life multiple sensors would be ideal. To be able to connect to multiple sensors, the data model needs to edited to keep connection with multiple sensors.

Reliability

A simple solution to improve reliability for this project is to use the Flutter Workmanager Plugin, which incorporates both operating systems: iOS and Android. It is essential to clarify the differences between the two systems on how they handle background processing. In the android environment, an API called Workmanager allows to schedule asynchronous tasks that are expected to run even if the app exits or the device restarts. The developer has several variables that can be adjusted, for example, the background process periodicity, unfortunately, it is not possible to have the application to run continuously in the background, the minimal interval is 15 minutes. Continuous background processing is not possible because the battery will be negatively impacted. In contrast to Android, iOS offers less degree of freedom, that is, only the the minimum background fetch interval can be adjusted.

Seizure Detection Algorithm

Various attempts have been made using conventional algorithms to accurately predict seizures however, so far, results are not that encouraging which is discussed in Joosse's thesis[23]. One way is by using the approximate entropy of a signal, which measures signal complexity in combination with a Discrete Wavelet Transform algorithm. Unfortunately, due to time constraints, this algorithm was not incorporated into this project. Instead, another simple functionality has been implemented in the cloud,

as described in 4.2. The operations of these algorithms are described in depth in [23]. For future work, it is possible to implement the algorithm in AWS Lambda.

A

Figures

A.1. iPhone screenshots

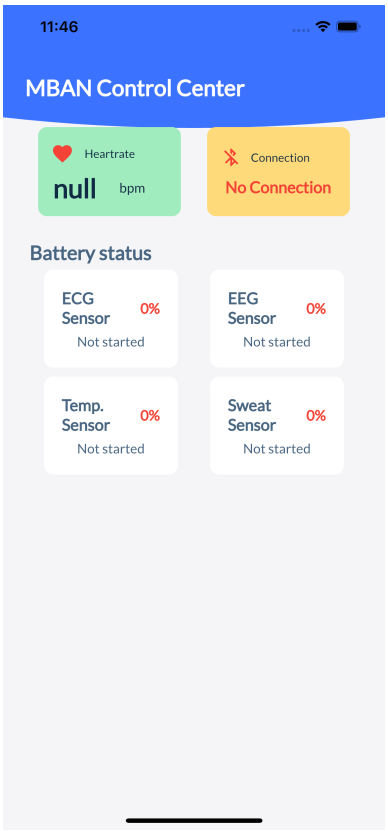

Figure A.1: The homepage, on an iPhone

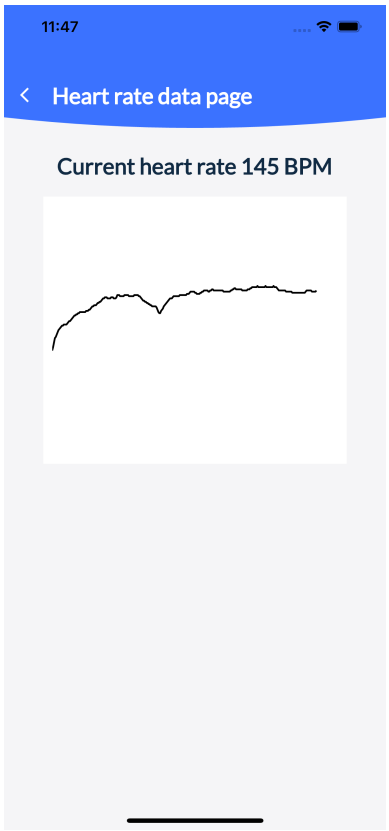

Figure A.2: The heart rate data page, on an iPhone

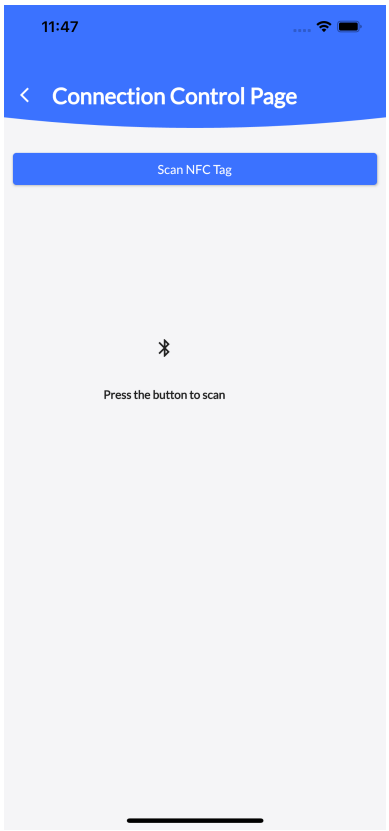

Figure A.3: The connection control page, on an iPhone

B

Dart code

B.1. Simon cypher

```
// Simon cipher encryption file
// Initialization of the used variables
int n = 32, m = 4, T = 44;
int z = 0x36EB19781229CD0F;
int c = 0xFFFFFFFC;
int xEnOldU = 0, xEnOldL = 0;
int xDecOldU = 0, xDecOldL = 0;
var k = List<int>.filled(T, 0);

// Function to incrypt the message to be send
// input should be a List with integers that the bluetooth communication
// is able to handle, output will also be a list of integers
List<int> encryption(List<int> list) {
  List<int> x, result;
  int xUp, xLow, temp;

  x = listtoX(list);
  xUp = x[0];
  xLow = x[1];

  // Cipher blockchaining
  xUp = xUp ^ xEnOldU;
  xLow = xLow ^ xEnOldL;

  // Encryption loop
  for (int i = 0; i <= (T - 1); i++) {
    temp = xUp;
    xUp =
        xLow ^ (_rotleft(xUp, 1) & _rotleft(xUp, 8)) ^ _rotleft(xUp, 2) ^ k[i];
    xLow = temp;
  }

  xEnOldU = xUp;
  xEnOldL = xLow;

  // Use bigint to avoid working with signed integers
  result = biginttolist(BigInt.from(xUp << 32 & xLow).toUnsigned(64));
  return result;
```

```
}

// Function to decrypt the message to be send
// input should be a List with integers that the bluetooth communication
// is able to handle, output will be a integer
int decryption(List<int> list) {
  List<int> x;
  int xUp, xLow, temp, tempU, tempL, result;

  x = listtoX(list);
  xUp = x[0];
  xLow = x[1];
  tempU = xUp;
  tempL = xLow;

  // Decryption loop
  for (int i = 0; i <= (T - 1); i++) {
    temp = xUp;
    xUp = xLow ^
        (_rotleft(xUp, 1) & _rotleft(xUp, 8)) ^
        _rotleft(xUp, 2) ^
        k[T - 1 - i];
    xLow = temp;
  }

  temp = xUp;
  xUp = xLow;
  xLow = temp;

  // Cipher block chaining
  xUp = xUp ^ xDecOldU;
  xLow = xLow ^ xDecOldL;

  xDecOldU = tempU;
  xDecOldL = tempL;

  result = xUp << 32 & xLow;
  return result;
}

// Key expansion function
keyexpans() {
  int temp, constant;
  // initialization of the beginning keys
  k[0] = 0;
  k[1] = 0;
  k[2] = 0;
  k[3] = 0;

  for (int i = m; i <= (T - 1); i++) {
    temp = _rotright(k[i - 1], 3);
    temp = temp ^ k[i - 3];
    temp = temp ^ _rotright(temp, 1);
    if (_bitSet(z, i - m)) {
      constant = 0x1;
    } else {
```

```
      constant = 0x0;
    }
    k[i] = k[i - m] ^ temp ^ constant ^ c;
  }
}

// 64-bit Bitwise rotation right, using unsigned shifting.
// Input intiger n is value, integer d is positions to be shifted
_rotright(int n, d) {
  return ((n >> d) & 0xFFFFFFFF) | ((n << 32 - d) & 0xFFFFFFFF);
}

// 64-bit Bitwise rotation left, using unsigned shifting.
// Input intiger n is value, integer d is positions to be shifted
_rotleft(int n, d) {
  return ((n << d) & 0xFFFFFFFF) | ((n >> 32 - d) & 0xFFFFFFFF);
}

// Bitset, return true if bit position pos in integer b is 1, otherwise false
bool _bitSet(int b, int pos) {
  return (b & (1 << pos)) != 0;
}

// Big integer to list, to make data ready for Bluetooth transmission.
// Big int makes sure no problems occurs with larges integers
// because the are signed in dart
List<int> biginttolist(BigInt begin) {
  var result = List<int>.filled(8, 0);
  BigInt temp = BigInt.zero, minus = BigInt.zero, rest = BigInt.zero;
  for (var i = 7; i >= 0; i--) {
    temp = begin >> 8 * i;
    rest = temp - minus;
    minus = temp << 8;
    result[7 - i] = rest.toInt();
  }
  return result;
}

// List to xUp and xLow to use in encryption en decryption
List<int> listtoX(List<int> list) {
  int shift1 = 0, shift2 = 0, xUp, xLow;
  xUp = 0;
  xLow = 0;

  for (var i = 0; i < list.length; i++) {
    if (i <= 3) {
      shift1 = list[i] << (8 * (3 - i));
      xUp = xUp + shift1;
    } else {
      shift2 = list[i] << (8 * (7 - i));
      xLow = xLow + shift2;
    }
  }
  return [xUp, xLow];
}
```

```
// Used to convert messages in List form to integers
int listtoint(List<int> list) {
  int shift = 0, x = 0;

  for (var i = 0; i < list.length; i++) {
    shift = list[i] << (8 * (7 - i));
    x = x + shift;
  }
  return x;
}
```

B.2. Firmware update

```
// Function to update the firmware on the implantable medical device
// Return statements make sure an alert dialog gets the right
// string to tell the user what happened

// The write() function called is a function to write to
// the designated Bluetooth channel

// Encryption and Decryption is not used as to make it more clear
// how the function works
firmwareUpdate() async {
  // Write firmware request
  write(Constants.firmwareReq);
  // make from stream a broadcaststreams so multiple listen
  // statements can be used
  broadcast = stream.asBroadcastStream();

  // await for first response
  await for (var response in broadcast) {
    // Usage of BigInt to avoid problems due to Dart using signed integers
    BigInt reaction = BigInt.from(listtoint(response)).toUnsigned(64);
    if (reaction == BigInt.zero) {
      // Flush out zero from Bluetooth notification channel
      print('response is zero');
    } else if (reaction == BigInt.one) {
      return '1st ack, not correct response';
    } else if (reaction == BigInt.two) {
      return '1st error, not correct response';
    } else {
      // Save nonce that is received
      BigInt nonceRead = reaction;
      // Create nonce to send
      var random = Random.secure();
      var nonceWrite = random.nextInt(0xFFFFFFFF);
      // Create command from the update and write nonce
      BigInt cmd =
          (BigInt.from(listtoint(Constants.firmwareUp)) << 32).toUnsigned(64) +
              nonceWrite;
      List<int> cmdnonceW = (biginttolist(cmd));
      // Save 32 most significant bits for later use
      List<int> signiflist = listtoX(cmdnonceW);
      int signif = signiflist[0];
      write(cmdnonceW);
      await for (var response in broadcast) {
```

```
        // See if reaction is an acknowledge
        BigInt reaction = BigInt.from(listtoint(response)).toUnsigned(64);
        if (reaction == BigInt.one) {
          // Create next message from 32 significant bits from last message
          // and the read nonce
          List<int> cmdnonceR = (biginttolist(
              (BigInt.from(signif) << 32).toUnsigned(64) + nonceRead));
          write(cmdnonceR);
          await for (var response in broadcast) {
            // See if reaction is an acknowledge
            BigInt reaction = BigInt.from(listtoint(response)).toUnsigned(64);
            if (reaction == BigInt.one) {
              return '3rd ack, not correct response';
            } else if (reaction == BigInt.two) {
              return '3rd error, not correct response';
            } else if (reaction ==
                (nonceWrite << 32).toUnsigned(64) + BigInt.one) {
              return 'succesful update';
            } else if (reaction ==
                (nonceWrite << 32).toUnsigned(64) + BigInt.two) {
              // Succesfully updated the firmware
              return '3rd unsuccesful update';
            } else {
              return '3rd not correct response';
            }
          }
        } else if (reaction == BigInt.two) {
          return '2nd ack, not correct response';
        } else {
          return '2nd not correct response';
        }
      }
    }
  }
}
```

B.3. Fetching

```
//this function fetches data from DB
  Future<MBANV4> fetchDB(name) async {
    try {
      final patients = await Amplify.DataStore.query(MBANV4.classType,
          where: MBANV4.NAME.eq('$name'));
      if (patients.isEmpty) {
        print("No objects with patientnumber: $name");
        return null;
      }
      _global = patients.first;
      print("testing fetchDB $_global");
    } catch (e) {
      print(e);
      throw (e);
    }
  }
```

C

Python code

C.1. simple lambda function

```
import boto3
import json
from boto3.dynamodb.conditions import Key, Attr


# read entry.
# determine if the heartrate is too high
# update the eeg value based on the given heartrate, 0 is low 1 is high.

def lambda_handler(event, context):
    # Get the service resource.
    dynamodb = boto3.resource('dynamodb')

    # Instantiate a table resource object without actually
    # creating a DynamoDB table. Note that the attributes of this table
    # are lazy-loaded: a request is not made nor are the attribute
    # values populated until the attributes
    # on the table resource are accessed or its load() method is called.
    table = dynamodb.Table('MBANV4-amuznha4jbh2pefcyfytt7s2xu-dev')

    entry_id = '8a2bd281-b5ba-4900-b0dd-6628078310a7'
    response=table.query(
        KeyConditionExpression=Key('id').eq(entry_id)
    )

    #accessing the JSON code.
    items = response['Items'][0]['heartrate']
    if items >= 160:
        updateheartrate(1,entry_id,table)
    else:
        updateheartrate(0,entry_id,table)

    return None


#This function updates the the given entry by id to a different name.
def updateheartrate(value,entry_id,table):
    # dynamodb = boto3.resource('dynamodb')
```

```
    # table = dynamodb.Table('MBANV4-amuznha4jbh2pefcyfytt7s2xu-dev')
    response = table.update_item(
    Key={
        "id": entry_id,
    },
    #ts is just a random name. It is necessary to modify an existing key, in this case eeg
    UpdateExpression='SET #ts = :val1',
    #assigning the value we get passed in.
    ExpressionAttributeValues={
      ":val1": value
    },
    #letting the code know what ts means.
    ExpressionAttributeNames={
      "#ts": "eeg"
    }
    )
```

Bibliography

[1] M. Goldenberg, "Overview of drugs used for epilepsy and seizures: Etiology, diagnosis, and treatment," *P T : a peer-reviewed journal for formulary management*, vol. 35, pp. 392–415, Jul. 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2912003/pdf/ptj35_7p392.pdf.

[2] S. Al-Janabi, I. Al-Shourbaji, M. Shojafar, and S. Shamshirband, "Survey of main challenges (security and privacy) in wireless body area networks for healthcare applications," *Egyptian Informatics Journal*, vol. 18, no. 2, pp. 113–122, 2017, ISSN: 1110-8665. DOI: https://doi.org/10.1016/j.eij.2016.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866516300482.

[3] P. Gope and T. Hwang, "Bsn-care: A secure iot-based modern healthcare system using body sensor network," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1368–1376, 2016. DOI: 10.1109/JSEN.2015.2502401.

[4] T. Wu, F. Wu, J.-M. Redouté, and M. R. Yuce, "An autonomous wireless body area network implementation towards iot connected healthcare applications," *IEEE Access*, vol. 5, pp. 11 413–11 422, 2017. DOI: 10.1109/ACCESS.2017.2716344.

[5] Carre Technologies inc. (2021). "Hexoskin smart shirts - cardiac, respiratory, sleep amp; activity metrics." Accessed: 20-May-2021, [Online]. Available: https://www.hexoskin.com/.

[6] K. Kolomiiets. (2021). "Flutter vs. react native: A developer's perspective." Accessed: 21-April-2021, [Online]. Available: https://gbksoft.com/blog/flutter-vs-react-native-a-developers-perspective.

[7] P. DeMarco. (2021). "Flutter blue." Github repository, A bluetooth plugin for Flutter, [Online]. Available: https://github.com/pauldemarco/flutter_blue.

[8] Philips Hue. (2021). "Flutter reacitve ble." Github repository, Flutter library that handles BLE operations for multiple devices., [Online]. Available: https://github.com/PhilipsHue/flutter_reactive_ble.

[9] Okadan. (2021). "Nfc manager." Github repository, Flutter plugin for accessing the NFC features on Android and iOS., [Online]. Available: https://github.com/okadan/flutter-nfc-manager.

[10] L. marceau. (2019). "Flutter-bluetooth." Github repository, A basic Flutter project to try the flutter_blue plugin, [Online]. Available: https://github.com/lmarceau/flutter-bluetooth.

[11] B. Egan. (2020). "Scoped model." Github repository, A set of utilities that allow you to easily pass a data Model from a parent Widget down to its descendants., [Online]. Available: https://github.com/brianegan/scoped_model.

[12] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck lightweight block ciphers," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15, San Francisco, California: Association for Computing Machinery, 2015, ISBN: 9781450335201. [Online]. Available: https://doi.org/10.1145/2744769.2747946.

[13] R. Valencia. (2020). "Medical app flutter design." Github repository, [Online]. Available: https://github.com/randyvalencia/mediapp.

[14] S. Rogers. (2021). "Oscilloscope widget package for flutter." Github repository, Oscilloscope is a graphical display similar to the trace on an oscilloscope that will display values as it scrolls across the screen., [Online]. Available: https://github.com/magnatronus/oscilloscope.

[15] E. Network and I. S. Agency. (2012). "Cloud computing." Accessed: 10-June-2021, [Online]. Available: https://resilience.enisa.europa.eu/cloud-security-and-resilience/publications/cloud-computing-benefits-risks-and-recommendations-for-information-security/.

[16] Statista. (2021). “Amazon leads $ 130-billion cloud market.” Accessed: 02-June-2021, [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[17] A. Gafaar, “Sql vs nosql,” May 2017.

[18] Amazon. (2021). “Shared responsibility model.” Accessed: 09-June-2021, [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/.

[19] O. Abood and S. Guirguis, “A survey on cryptography algorithms,” *International Journal of Scientific and Research Publications*, vol. 8, pp. 495–516, Jul. 2018. DOI: 10.29322/IJSRP.8.7.2018.p7978.

[20] A. Mamun, S. Rahman, T. Shaon, and M. A. Hossain, “Security analysis of aes and enhancing its security by modifying s-box with an additional byte,” *International journal of Computer Networks Communications*, vol. 9, pp. 69–88, Mar. 2017. DOI: 10.5121/ijcnc.2017.9206.

[21] G. Shaheen, “A robust review of sha: Featuring coherent characteristics,” Feb. 2020. DOI: 10.13140/RG.2.2.26608.69126.

[22] I. Kuzminykh, B. Ghita, and S. Shiaeles, “Comparative analysis of cryptographic key management systems,” in. Dec. 2020, pp. 80–94, ISBN: 978-3-030-65728-4. DOI: 10.1007/978-3-030-65729-1_8.

[23] C. Joosse, “Absence seizure prediction using recurrent neural networks,” p. 22, 2020. [Online]. Available: http://resolver.tudelft.nl/uuid:e752232f-cd6b-47c4-ba70-899a020b7e6d.