"Schrödingers Alarm"
Projektdokumentation

Lucas Hanisch
Seifeddine Mhiri
Paul Krause
Monique Golnik

Master Informations- und Kommunikationstechnik WS 2021/22

21. Februar 2022

Inhaltsverzeichnis

Kapitel 1

Start

1.1 Zielsetzung

Seif

1.2 Ideensammlung

1.2.1 Starkstrom Taser Auto

Moni

1.2.2 Autoleere Stadt

Eine unserer Ideen war es, anstatt eine direkte Lösung für das Problem zu finden, das Problem selbst zu beseitigen. Hierfür haben wir uns vorgestellt, dass wir einfach Autos generell abschaffen. Die Idee ist, dass wenn es keine Autos mehr gibt, können sie auch nicht mehr geklaut werden. Diese Idee ist jedoch an der Umsetzbarkeit gescheitert. Es gibt zwar bereits Innenstädte, wo Autos verboten sind, jedoch ist dies nicht gleichzusetzen mit dem kompletten Abschaffen von Autos, da diese zumeist nur außerhalb der Stadt geparkt sind. Die Idee wurde letztendlich verworfen, da die Umsetzung dieses Projekt unseren Projektrahmen gesprengt hätte.

1.2.3 Gesichtserkennung und Startunterbrechung

Bei dieser Lösung hatten wir uns überlegt, mittels Gesichtserkennung zu überprüfen, ob die sich auf dem Fahrersitz befindliche Person der Besitzer oder ein berechtigter Fahrer des Fahrzeugs ist. Die Zündung des Autos soll hier nur dann möglich sein, wenn der Fahrer erkannt wird. Das Problem an dieser Modifikation und der Grund, warum wir uns gegen diese Idee entschieden haben, ist die Zulassungspflicht durch den TÜV. Um den Motor am

Starten zu hindern, müssten wir uns entweder auf das Steuergerät des Autos mit aufschalten oder direkt eine Fernsteuerung für die Motorklemme mit einbauen. Die erste Möglichkeit scheitert daran, dass sich der Zugang zu den Steuersystemen des Autos schwer gestaltet, da je nach Modell die Steuersoftware unterschiedlich ist, womit allgemeine Ansätze ausgeschlossen sind. Des Weiteren erlauben Hersteller entweder generell nicht, dass Drittanbieter ihre Software nutzen oder stimmen nur gegen hohe Lizenzgebühren einer Nutzung zu. Ein weiteres Problem ist, dass beide Lösungen eine professionelle Installation bedürfen, was unserem Konzept widerspricht. Abgesehen davon ist es wie, bereits erwähnt, notwendig, für jedes Fahrzeugmodell die entsprechende Straßenzulassung für die Modifikation zu erhalten. Entsprechend ist der Arbeitsaufwand für unseren Projektzeitraum zu groß.

1.3 Projektmanagement

Für das Projekt haben wir nach der agilen Methode SCRUM gearbeitet - daher wird man auf der Suche nach einem historischen Pflichten- und Lastenheft bei uns nicht fündig. Die in unserem Projektstrukturplan erfassten Arbeitspakete haben wir in einem sogenannten Trello-Board festgehalten. Dies ermöglichte uns dank einer selbstgewählten Kartenübersicht auf einem Blick schnell erfassen zu können, wo wir derzeit stehen und was noch unbedingt erledigt werden sollte. Ebenfalls haben wir unvorhergesehene Probleme dort festgehalten, um zum einem Stichpunkte für die Dokumentation festzuhalten und zum anderen nachvollziehen zu können, weshalb wir manche Tickets nicht so schnell fertig bearbeiten konnten, wie ursprünglich angenommen.

SCRUM – „Pläne sind nichts. Planung ist alles“

Die kurze zyklische Struktur von Scrum mit seinen wiederkehrenden Meetings bietet eine ständige Möglichkeit zur Überprüfung und Anpassung. Positiv ist hierbei auch, dass auch der Fortschritt zeitnah sichtbar und dementsprechend das Team motiviert wird. Anfänglich haben wir Sprints von 14 Tagen ausprobiert in der Retrospektive jedoch festgestellt, dass ein 7-tägiger Rhythmus, um schneller Probleme analysieren zu können und aufgrund der recht kurzen Gesamtbearbeitungszeit im ständigen Austausch zu sein, sinnvoller ist. Dies hat sich auch sehr bald als richtige Entscheidung herausgestellt, da das Gefühl “dass jeder nur seine Insel bearbeitet” minimiert werden konnte.

1.4 Konzept

1.4.1 Umfrage

Im Rahmen der Projektvorbereitung soll auch eine Umfrage unter potenziellen Kunden erfolgen. So sollen möglichst früh in der Entwicklung Wünsche

und die Sicht der Kunden auf ein Produkt oder ein Problem eingeholt werden. Hierbei handelt es sich um den sogenannten Customer Insight, welcher uns neue Perspektiven eröffnen soll. Damit soll verhindert werden, dass wir in unseren Ideen möglicherweise etwas ganz Entscheidendes vergessen, was aber aus der Sicht der Kunden besonders wichtig ist. Es sollen außerdem auch möglichst viele verschiedene Anforderungen der einzelnen Personen zu erhalten. Dafür haben wir eine Umfrage mit insgesamt 19 Fragen entwickelt. Am Ende konnten wir immerhin 44 Teilnehmende aufweisen.

Zuerst kamen ein paar Fragen zum Fahrzeug der Teilnehmer, z.B. von welcher Marke und wie alt es ist. Im zweiten Abschnitt stellten wir einige Frage zur Sicherheit der Fahrzeuge der Teilnehmer, z.B. wie sie gegen Diebstahl geschützt sind und ob schonmal etwas gestohlen wurde. Als nächstes folgten Fragen, mit denen wir direkte Bedürfnisse und Anforderungen für unser Produkt gewinnen wollten, unter anderem wurde gefragt, was ihn/sie an aktuell zu erwerbenden Autoalarmsystemen stört und wieviel er/sie maximal für ein solches System ausgeben würde. Zum Abschluss wurden noch ein paar persönliche Informationen für mögliche statistische Zusammenhänge abgefragt, wie Geschlecht, Alter und Beschäftigungsverhältnis. Konkret wurden die folgenden Fragen gestellt:

- Von welcher Marke ist dein Auto?
- Zu welcher Fahrzeugklasse gehört dein Auto?
- Welches Baujahr ist dein Auto?
- Wie teuer war dein Auto ungefähr und wieviel ist es jetzt noch wert?
- Wie wichtig ist dir dein Auto? (Emotionale Bindung usw.)
- Wie ist dein Auto gegen Einbruch und Diebstahl geschützt?
- Wie zufrieden bist du mit deiner Sicherheitslösung?
- Wurde dir bereits ein Auto geklaut oder es versucht?
- Wurde schon einmal in dein Auto eingebrochen oder es versucht?
- Wenn ja, was wurde geklaut?
- Wie ist dein Auto versichert?
- Wo parkst du dein Auto normalerweise?
- Um was hast du Angst, wenn dein Auto unbeaufsichtigt ist?
- Was stört dich an bisher käuflich erwerblichen Alarmsystemen für Autos bzw. was würdest du an diesen ändern?

- Wie viel würdest du für ein neues Alarmsystem für dein Auto ausgeben?
- Du bist... (männlich, weiblich, divers)
- Wie alt bist du?
- Wie ist dein derzeitiges Beschäftigungsverhältnis?
- Wie hoch ist dein monatliches Nettoeinkommen?

Die Umfrage wurde mit Hilfe von Google Forms realisiert und noch unter folgendem Link abrufbar: Umfrage

1.4.2 Persona

Moni

1.4.3 Konzept

Nachdem wir also mit Hilfe der Umfrage und der Persona schon einen groben Eindruck bekommen haben, auf welche Zielgruppe unser Produkt passen könnte und welche Eigenschaften und Funktionen es erfüllen soll, erstellten wir ein erstes, grobes Konzept. Dieses Konzept beinhaltet alle wesentlichen Komponenten unseres Produkts und ist in der folgenden Abbildung schematisch dargestellt.

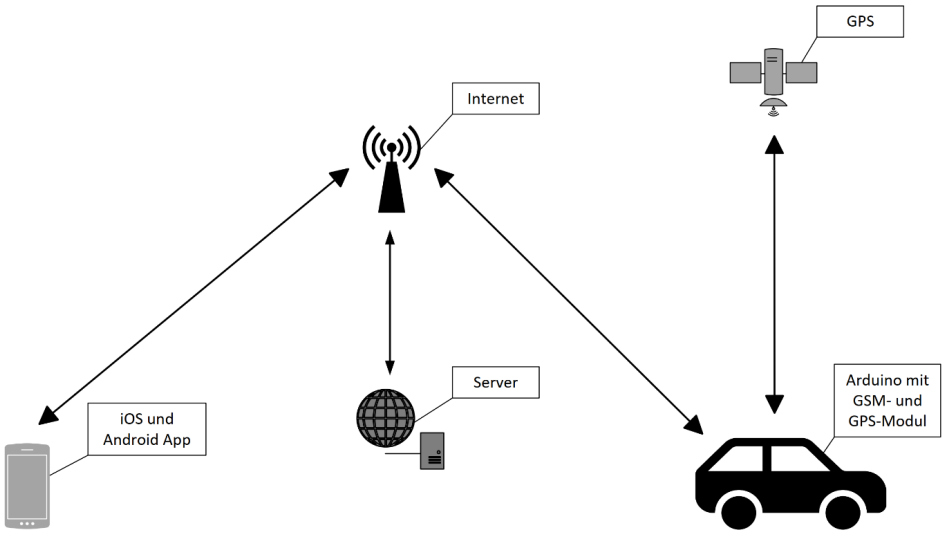

Abbildung 1.1: Systemkonzept

Im Fahrzeug wollen wir einen Arduino Uno zusammen mit einem passenden GSM- und GPS-Modul platzieren. Dieser ermittelt in bestimmten Zeitintervallen die aktuelle Position des Fahrzeugs und schickt die Koordinaten über die GSM-Verbindung an unseren Webserver. Dieser verarbeitet diese Daten und stellt sie anschließend so bereit, dass die Apps beider Betriebssysteme die aktuellen Koordinaten abfragen können.

Kapitel 2

Entwicklung

2.1 Arduino

2.1.1 Ausstattung

Um den Zielen, die wir uns für unseren Prototypen gesetzt haben gerecht zu werden, ist der Arduino mit einem Sim 808 Module sowie entsprechenden GPS und GSM Antennen ausgerüstet. Angebracht sind das Modul und die Antennen durch einen Shield. Es handelt sich um ein Shield des Models OAS808SIM und ist von der Firma MakerFabs.

Die Stromversorgung erfolgt mittels vier BRC 18650 4200mAh Batterien, die in Reihe geschaltet sind.

2.1.2 Software

Die Kontrolle des Shields erfolgt mittels UART und entsprechenden AT-Befehlen. Die erforderlichen Befehle sind zusammen mit einer kurzen Erklärung nachfolgend aufgelistet.

AT+CPIN=<PIN>	Konfiguriert die Pin für die Sim-Karte
AT+SAPBR=3,1,\"Contype\",\"GPRS\"	Definiert die Art der übertragenen Daten als GPRS Daten
AT+SAPBR=3,1,\"APN\",\"TM\"	Gibt an, welcher APN verwendet wird. Wir verwenden TM was für Things Mobile steht und dem Provider der Sim Karte entspricht
AT+SAPBR=1,1	Startet die Verbindung mit dem Carrier mit den angegebenen Parametern
AT+SAPBR=2,1	Fragt die Statusinformationen vom Carrier ab
AT+HTTPPARA=\"CID\",1	Definiert, welches Profil an Trägerdiensten verwendet werden soll
AT+HTTPPARA=URL, URL	Legt fest, an welche URL gesendet werden soll
AT+HTTPPARA=CONTENT,application/x-www-form-urlencoded	Informiert den Server, um welchen Datentyp es sich bei den gesendeten Daten handelt
AT+HTTPDATA=192,5000	Definiert im ersten Parameter die Größe des gesendeten Datenpaketes und im zweiten Parameter, wie lange auf eine Antwort vom Server gewartet werden soll, bevor die Übertragung abgebrochen werden soll
param={\"device\":\""+device+"\",\"Latitude\":"+Feld[3]+",\"Longitude\":"+Feld[4]+"}	Sendet die gewünschten Daten im JSON Format an den Server, wie im vorhinein definiert wurde
AT+HTTPACTION=1	Gibt die Art des Http Befehls an GET=0 , POST=1 und HEAD=2
AT+HTTPREAD	Liest die Antwort vom Server
AT+HTTPTERM	Bricht laufende Http Anfrage ab
AT+SAPBR=0,1	Beendet die Verbindung zum Carrier
AT+HTTPINIT	Startet den HTTP Service auf dem Sim 808 Modul

Abbildung 2.1: GSM und Http-Befehle

AT+CGNSPWR=1	Startet GPS Module
AT+CGNSSEQ=RMC	Definiert die Schreibweise der Zurückgegebenen GPS Informationen
AT+CGNSINF	Fragt die momentane GPS-Position ab

Abbildung 2.2: GPS-Befehle

2.1.3 Aufbau Code

Der Code ist wie ein Standard-Mikrocontroller-Code aufgebaut und besitzt zwei Unterfunktionen, die hier kurz grafisch dargestellt sind.

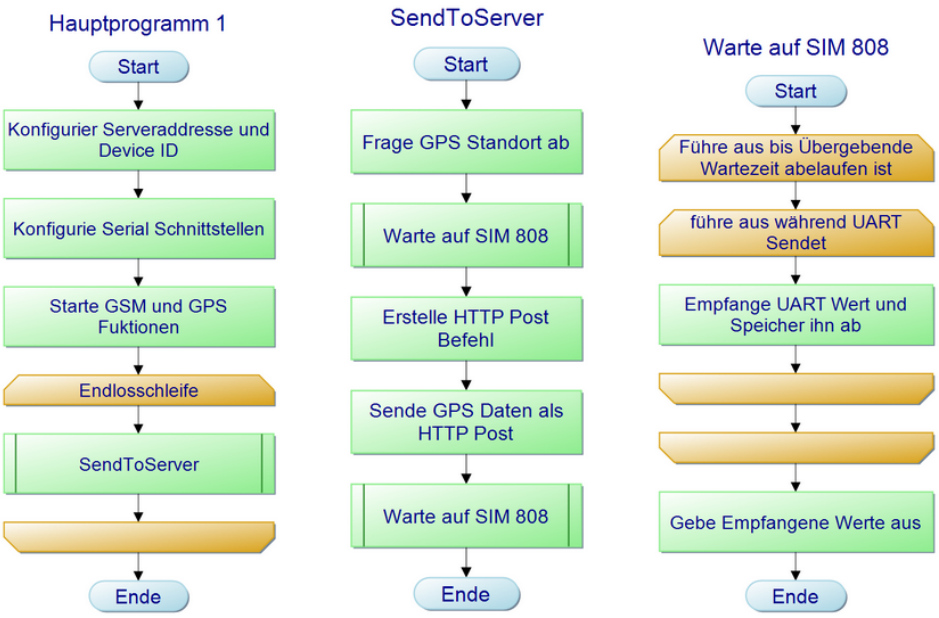

Abbildung 2.3: Programmablaufplan

Der Arduino macht in der momentanen Prototyp Version nichts anderes, als das Sim 808 Modul zu steuern. Dafür gibt es drei Funktionen: Eine Funktion, welche die Kommunikation mit dem Sim Modul erleichtert, eine weitere Funktion für die Abfrage des momentanen Standortes und eine dritte um die Kommunikation mit dem Server zu erlauben. Mit der Funktion wart() wird zum einen das Pausieren für die Bearbeitungszeiträume geliefert und zum anderen die Ausgabe des Sim Moduls an den PC zum Debuggen geschickt. Die Funktion gpsDaten fragt den aktuellen Standort ab und die Funktion SendToServer sendet die zu übermittelnden GPS Daten an den Server. Der

Code ist in C geschrieben, da diese Programmiersprache für unsere Zwecke ausreichend performant ist.

2.1.4 Entwicklungsprozess

Der Entwicklungsprozess ist in mehreren Stufen erfolgt. Zunächst wurde mittels eines einfachen Programms die Funktionalität der GPS-Antenne überprüft, da wir dafür noch keine Sim-Karte benötigten. Zur Implementierung der Standortabfrage soll zum ersten Mal die UART-Schnittstelle zum Board konfiguriert werden. Nachdem die UART-Schnittstelle konfiguriert wurde, haben wir die Funktion wart() entwickelt, um die Kommunikation mit der UART-Schnittstelle zu erleichtern. Mit Hilfe der wart() Funktion konnten wir nun auf die Antwort von der UART-Schnittstelle warten und dies auf dem Computer zum Debuggen ausgeben. Das Senden und Empfangen von Informationen über UART funktionierte also und anschließend haben wir die Abfrage der GPS-Daten realisiert. Nachdem wir die entsprechenden Befehle gefunden hatten, war dies einfach zu realisieren.

Da die ersten Tests jedoch auf dem Dachboden ausgeführt wurden und das Dach aus Metall ist, konnte zunächst kein Signal empfangen werden. Nachdem wir das Problem erkannt hatten, haben wir alle zukünftigen Test an einer besser geeigneten Position durchgeführt.

Die Standortabfrage funktionierte nun, sodass wir im nächsten Schritt zum Testen des GSM-Moduls und der Sim-Karte das Senden von SMS realisiert hatten. An dieser Stelle haben wir lediglich die bereits erstellten Funktionen genutzt und die Befehle für die SMS-Kommunikation hinzugefügt. Dabei ist aufgefallen, dass beim erstmaligen Nutzen des GSM-Moduls nach dem Einschalten immer der PIN zum Entsperren erforderlich ist.

Nachdem wir die Kommunikation mit dem GSM Modul ermöglicht hatten, haben wir die Kommunikation mit dem Server realisiert. Nachdem wir hierfür die richtigen Befehle gefunden hatten, haben wir zunächst Testdaten per HTTP POST an einen Test-Server gesendet, der einfach nur seine empfangen HTTP POSTs anzeigt. Danach haben wir das Senden von JSONs an unseren Server realisiert. Aus Netzwerksicht mussten wir nur die Serveradresse ändern. Damit der Server die empfangenden Daten nutzen kann, mussten wir dann jedoch noch den Daten-Sendebefehl neu Konfigurieren. Da wir über die UART-Schnittstelle nur Strings schicken können, mussten wir die Formatierung für die zu sendenden Daten anders einstellen und diese dann beim versendeten String berücksichtigen.

Nachdem die Kommunikation mit dem Server erfolgreich funktioniert hatte, haben wir die einzelnen Teile zusammengefügt. Hierfür mussten wir den GPS-Antwort String verarbeiten. Um die benötigten Werte zu bekommen, haben wir den String immer an den Kommas in kleinere Strings unterteilt und über die vordefinierte Struktur des Strings ermittelt, welche Werte der Längengrad und der Breitengrad sind. Dies Werte haben wir für zukünf-

tige Erweiterungen bzw. Funktionalitäten zwischengespeichert und binden sie dann in den zu sendenden String ein.

Wir haben dabei den Datentyp String der erhaltenen Daten bewusst nicht geändert, da dies sonst dazu führt, dass Nachkommastellen weggelassen werden, was die Standortbestimmung verzerrt.

2.1.5 Aufbau Hardware

Für die Stromversorgung müssen die Akkus an den Vin (Eingangsspannung) Pin und den GND (Masse) Pin angeschlossen werden. Da das Schild diese Pins einfach durch schaltet, können wir die Konstruktion trotz Schild so verwenden.

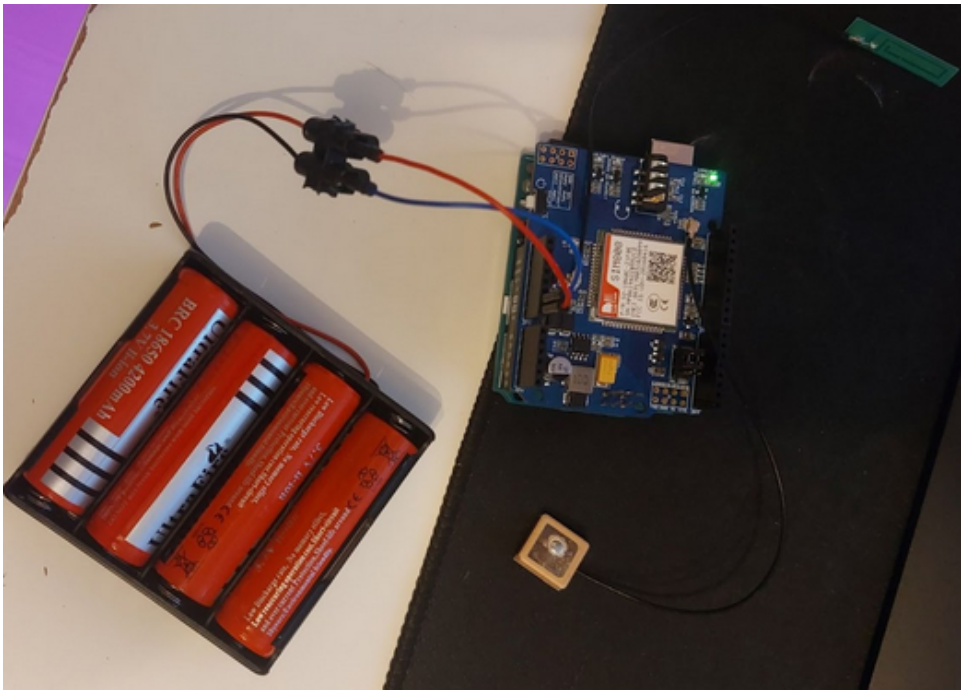

Abbildung 2.4: Systemaufbau für das KFZ

Der Vorteil dieser Konstruktion ist, dass sie sehr kompakt ist und somit leichter im Auto versteckt werden kann. Ein weiterer Vorteil dieser Konstruktion ist, dass für den Fall, dass später durch Erweiterungen die Akkulaufzeit nicht ausreicht, einfach weitere Akkus parallel hinzugefügt werden können.

2.1.6 Probleme

Die Probleme, die bei der Entwicklung des Controllers aufgetreten sind, lassen sich in zwei Kategorien unterteilen. Zum einen gab es technische Pro-

bleme und zum anderen logistische Schwierigkeiten. Das Problem bei der Beschaffung der GSM-Module war, dass diese meist aus China importiert werden mussten, was dazu geführt hat, dass wir unter langen Wartezeiten gelitten haben. Von den vier bestellten Modulen wurden zwei nicht geliefert (die Bestellung wurde vom Lieferanten gecancelt). Des Weiteren konnte eines der Module nur über eine bereitgestellte Software verwendet werden.

Die technischen Probleme sind zum Teil auf die schlechte Dokumentation des Arduino Shields zurückzuführen. Zum einen fehlt die Information für die richtige Baudrate für die UART Schnittstelle, welche über strukturiertes Testen herausgefunden werden musste. Das Problem hierbei war, dass das Modul entsprechend keine Fehlermeldung zurückgeben konnte, da die Kommunikation zwischen Modul und Arduino selbst gestört war und nur nichts zurückgeben wurde:

```
Test
;;;;;;;;;;;;;;;;;;;;

Test
AT+CMEE=1
OK
AT+CPIN=<1503>
+CME ERROR: 3
AT+CGNSPWR=1
OK
AT+CGNSSEQ=RMC
OK
AT+CGNSINF
+CGNSINF: 1;0;19800106000003.000;;;;0.00;0.0;0;;;;;;0;0;;;;;AT+CGNSINF
+CGNSINF: 1,0,19800106000003.000,,,,0.00,0.0,0,,,,,,0,0,,,,,
```

Abbildung 2.5: fehlerhafte Ausgabe

In dem obenstehenden Bild sieht man zunächst, wie die fehlerhafte Ausgabe aussieht. Unter der fehlerhaften Ausgabe ist dann die richtige Ausgabe zum Vergleich abgebildet.

Auch die AT-Befehle, welche zur Kommunikation mittels HTTP dienen sollten, haben nicht entsprechend funktioniert. Gelöst werden konnte das Problem, indem die Befehlskette verwendet wird, mit der jedes Datenpaket einzeln und manuell konfiguriert werden kann. Entsprechend mussten die Pakete als IOT-HTTP-Paket definiert werden und die Netzbetreiber-Daten sowie Paketinformationen für die Verbindung eingestellt werden.

Des weiteren hat das GSM-Modul die Kommunikation mit einer normalen Vodafon Sim-Karte nicht aufbauen können. Die Verzögerung, eine geeignete Prepaid Karte zu bekommen, hat den Zeitrahmen für diesen Teil erheblich

nach hinten verschoben. Das Problem an diesen Fehlern war, dass das Modul nur ERROR 3 zurückgegeben hat, was als "Operation nicht erlaubt" definiert ist. Es wird dabei aber nicht genauer spezifiziert, warum diese Operation nicht erlaubt ist. Dies hat das Troubleshooting weiter erschwert.

Für die GPS-Abfrage haben sich zwei wesentliche Probleme ergeben. Das am schwierigsten zu lösende Problem war, dass die benötigten Befehle nicht in der richtigen Dokumentation zu finden sind, sondern nur in einigen Beispiel-Programmen vom Shield Hersteller. Das andere Problem ist, dass das GPS-Signal beim ersten Konfigurieren (Einstellen der internen Uhr) guten Empfang braucht.

2.2 Webserver

2.2.1 Einleitung / Notwendigkeit

Damit die vom Arduino ermittelten Standortdaten des Fahrzeugs an die Apps weitergeleitet werden können, benötigt es eine Zwischeninstanz. Diese soll die empfangenen Daten aufbereiten und sie anschließend so bereitstellen, dass sie einfach von den Apps abgefragt werden können. In einem zweiten Entwicklungsschritt soll dann der Sicherheitsaspekt der Daten berücksichtigt werden. Hierfür sollen Verschlüsselungsalgorithmen implementiert werden, sodass die Daten nur von den Apps entschlüsselt und ausgewertet werden können.

2.2.2 Einrichtung des Webservers auf einem Raspberry Pi

Der HTTP-Webserver soll auf einem Raspberry Pi mit Hilfe des „lighttpd"-Webservers erstellt werden. Die Installation und Einrichtung desselbigen gestaltete sich relativ einfach. Zunächst musste das entsprechende „lighttpd"-Paket auf dem Raspberry Pi installiert werden. Im nächsten Schritt mussten noch ein paar Berechtigungen erteilt werden. Nach einem Neustart von lighttpd war die Einrichtung soweit fertig und der Webserver war aktiv. Alle Dateien und Skripte, die der Webserver bereitstellt und bearbeitet liegen im Verzeichnis „/var/www/html".

Damit der Server später auch von außerhalb erreichbar ist, waren noch einige Arbeitsschritte zu erledigen. Zuerst musste der Raspberry Pi eine statische IPv4-Adresse erhalten, wir setzten diese auf 192.168.178.215. Die .215 wurde deshalb ausgewählt, weil der Bereich der automatischen Adresszuweisung von den meisten DHCP-Servern nur von .0 bis .200 geht und somit eine doppelte Vergabe sehr unwahrscheinlich ist.

Bei DYNDNS handelt es sich um einen Service, der es ermöglicht, Domains mit dynamisch wechselnden IP-Adressen im Internet zu verwalten. Dieser Service ist notwendig, da der heimische Router vom Provider in regelmäßigen Abständen eine neue, externe IP-Adresse zugeordnet bekommt

und somit der Raspberry Pi bzw. der darauf aktive Webserver nach einem ÏP-Wechsel"nicht mehr über das Internet erreichbar wäre. Bei Nutzung eines DYNDNS Anbieters, bleibt der Raspberry Pi bzw. der Webserver weiterhin unter seinem vergebenen Hostnamen erreichbar. Damit ein solcher Service genutzt werden kann, ist es notwendig, sich bei einem DYNDNS-Anbieter einen Account anzulegen. Für diese Arbeit wurde sich aufgrund einer persönlichen Empfehlung für Securepoint DYNDNS entschieden. Auf deren Webseite konnte nach der Erstellung eines Benutzerkontos ein IPv4-Host hinzugefügt werden. Dafür wurde neben der IP-Adresse des Raspberry Pi der Hostname eingetragen, welcher dann automatisch eine Domainerweiterung des DYNDNS Anbieters erhalten hat.

Im letzten Schritt musste nun der heimische Router so konfiguriert werden, dass ein Zugriff auf den Webserver von außerhalb des Heimnetzwerks erlaubt ist. In unserem Fall handelt es sich beim Router um eine FRITZ!Box, bei welcher diese Einstellungen einfach vorgenommen werden konnten. Dafür musste zunächst eine neue Freigabe für das Gerät „Raspberry Pi“ hinzugefügt werden. Anschließend wird als Anwendung „HTTP-Server“ und damit Port 80 ausgewählt. Zusätzlich musste dann noch das DYNDNS in der FRITZ!Box konfiguriert werden. Dafür wird eine Update-URL des DYNDNS-Anbieters sowie der Domainname und die Benutzerinformationen benötigt. Damit waren alle notwendigen Einstellungen vorgenommen und er Webserver ist nun von überall erreichbar.

Zusätzlich haben wir die Software „VNC Server“ auf dem Raspberry Pi installiert. Mit Hilfe dieser Software und dem entsprechenden Client „VNC Viewer“ auf einem PC oder Laptop kann eine Remotedesktopverbindung zum Raspberry Pi hergestellt werden. Dies ist besonders auch in Zeiten von Lockdowns und Online-Vorlesungen sowie digitalen Gruppenmeetings eine wichtige Funktion, da so von überall an dem Raspberry Pi gearbeitet werden kann.

2.2.3 Probleme mit dem Raspberry Pi

Während der ersten Wochen der Projektarbeit mussten wir leider alle paar Tage feststellen, dass keine Verbindung per VNC zum Raspberry Pi hergestellt werden konnte. Wir vermuteten zunächst keinen dramatischen Fehler, denn nach einem Neustart funktionierte wieder alles. Dieser Fehler trat allerdings doch mit einer gewissen Regelmäßigkeit auf und nachdem ein erstes PHP-Skript auf unserem Webserver lief, konnten wir feststellen, dass dieses auch regelmäßig nicht erreichbar war. Der Fehler lag also eindeutig nicht nur an der VNC-Software, sondern es scheint generelle Verbindungsprobleme des Raspberry Pi zu geben. Deshalb versuchten wir doch genauer herauszufinden, wo der Ursprung des Verbindungsverlustes ist. Nach einigem Recherchieren und dem Sichten diverser Log-Files auf dem Raspberry Pi konnten wir die Fehlermeldung „brcmf_cfg80211_scan: scan error (-110)“

für die Verbindungsabbrüche ausfindig machen. Bei diesem Fehler handelt es sich um ein Verbindungsproblem des WiFis des Raspberry Pi's. Dieser Fehler ist nicht unbekannt und mögliche Lösungen werden in diversen Internetforen diskutiert. Allerdings scheint keine der vorgeschlagenen Workarounds generell zu funktionieren, außer ein wie auch von uns praktiziertes, regelmäßiges Neustarten. Bevor wir diese alle ausprobieren und im ungünstigsten Fall eine Menge Zeit ohne Ergebnis verschwenden würden, entschlossen wir uns einfach dazu, den Raspberry Pi per LAN-Kabel direkt mit der Fritzbox zu verbinden. Nachdem dann auch nach über zwei Wochen keine Verbindungsabbrüche registriert wurden, konnten wir dieses Problem als gelöst ansehen und uns wieder anderen Aufgaben des Projekts widmen.

Im Verlauf der weiteren Programmierung der PHP-Skripte stellte sich heraus, dass die PHP-Version 7.3 auf dem Raspberry Pi veraltet war und einige nützliche Funktionen nicht unterstützt. Deshalb wurde die Version 7.3 deinstalliert und anschließend mit PHP 8.0 eine der aktuellen Versionen installiert. Allerdings lief entweder die Deinstallation der alten oder die Installation der neuen Version nicht fehlerfrei. Nach einigem Suchen in verschiedenen Log-Files stellte sich heraus, dass fälschlicherweise die Socket-Datei „php7.3-fpm.sock“ der Version 7.3 vom lighttpd-Server versucht wurde zu öffnen. Diese existierte aber natürlich nach der Deinstallation nicht mehr. Mit PHP 8.0 wurde eine entsprechende, neue Datei „php8.0-fpm.sock“ im gleichen Dateipfad „/var/run/php“ bereits erzeugt. In der Konfigurations-Datei „15-fastcgi-php.conf“ im Verzeichnis „/etc/lighttpd/conf-enabled“ ist der Pfad der vom lighttpd-Server zu nutzenden Socket-Datei hinterlegt. Allerdings ist es nicht möglich, diese Datei einfach mit Hilfe eines Texteditors auf dem Raspberry Pi zu verändern. Dafür benötigt es höhere Benutzerrechte. Die Datei lässt sich auch über die Kommandozeile öffnen und bearbeiten. Dabei ist es wichtig den Zusatz „sudo“ vor jeden Befehl zu schreiben, da somit für den folgenden Befehl die notwendigen höheren Rechte temporär gewährt werden. So war es möglich, in der Datei „15-fastcgi-php.conf“ den Pfad der Socket-Datei auf die aktuelle Datei von PHP-Version 8.0 zu ändern. Im Anschluss musste der lighttpd-Server noch einmal neu gestartet werden, sodass alle Änderungen übernommen werden konnten.

2.2.4 Grundsätzliche Überlegungen

Bevor es ans konkrete Programmieren verschiedener HTML- oder PHP-Skripte geht, wollte wir uns zunächst überlegen, was denn grundsätzlich auf dem Server passieren soll. Auf jeden Fall benötigen wir ein Skript, das Daten nutzen und weiterverarbeiten kann, die an seine URL (Uniform Resource Locator) übermittelt wurden. Darüber hinaus ist es sicherlich sinnvoll und möglicherweise sogar notwendig, das Bereitstellen von Daten für die Apps in einem separaten Skript zu realisieren. Zusätzliche Skripte für zum Beispiel eine Verschlüsselung der Daten sind gegebenenfalls zu einem späteren Zeit-

punkt noch interessant. Auch wenn der GPS-Sensor am Arduino neben den Längen- und Breitengraden noch viele andere Informationen erfasst, reicht es für unser primäres Projektziel aus, nur die Koordinaten an den Webserver zu senden. Um die Koordinaten-Paare bei der Weiterverarbeitung in den Apps unterscheiden zu können, soll der Webserver die Daten noch mit dem aktuellen Zeitstempel verknüpfen.

Sowohl die Daten, die der Arduino an den Webserver übermittelt als auch die Daten, die der Webserver für die Apps wieder bereitstellt, sollen im JSON-Format (JSON = JavaScript Object Notation) übertragen werden. Dieses Format ist einerseits leicht lesbar und andererseits ist es unabhängig von Programmiersprachen und somit universell nutzbar.

Über mögliche Sicherheitsaspekte unserer Datenübertragung dachten wir anfangs nicht wirklich nach, da eine an sich funktionierende Kommunikation zwischen den verschiedenen Teilnehmern erstmal im Vordergrund stand. Lediglich die Überlegung, wie man verhindern kann, dass nicht jeder beliebige Internetnutzer irgendwelche Daten an unseren Webserver schickt, beschäftigte uns am Anfang des Projekts. Es lässt sich natürlich nicht verhindern, dass andere Geräte versehentlich oder auch absichtlich irgendwelche Daten an unseren Webserver bzw. an ein dort laufendes PHP-Skript senden. Aber wir können dieses Skript so programmieren, dass nur bestimmte empfangene Daten auch weiterverarbeitet werden. Um dies zu realisieren, muss jedes Gerät eine von uns manuell vergebenen Geräte-Identifikationscode (Geräte-ID) zusammen mit den Nutzdaten verschicken. Diese Geräte-IDs sind 20-stellige, alphanumerische Werte, die wir einmal zufällig generiert haben und die in der txt-Datei „allowed_devices.txt“ lokal auf dem Raspberry Pi gespeichert sind. So kann später darauf zugegriffen werden und ein Abgleich erfolgen, ob die mitgesendete Geräte-ID bei uns bekannt ist. Wenn dies nicht der Fall ist, sollen diese Daten nicht weiterbearbeitet werden.

Es ist klar, dass auch diese Sicherheitsvorkehrung nicht besonders stark ist, solange die Geräte-IDs unverschlüsselt und quasi frei zugänglich auf dem Server gespeichert sind. Aber es ist ein erstes Mittel, um zumindest einfache Störversuche zu unterbinden und auch uns selbst vor möglichen Übertragungsfehlern zu schützen.

Es ist sehr wahrscheinlich, dass nicht alle Projektteile mit der gleichen Geschwindigkeit bearbeitet werden und somit einzelne Teile unterschiedlich schnell in einem testfähigen Stadium sind. In diesem Fall betrifft das die GSM-Sendeeinheit des Arduinos und die einzelnen Skripte des Webservers. Damit grundsätzliche Funktionalitäten der Skripte möglichst schon während der Entwicklung getestet werden können, ohne dass erst auf den Projektteil des Arduinos gewartet werden muss, wollten wir eine einfache Anwendung für einen Windows-PC entwickeln. Die Entscheidung fiel hier relativ schnell auf die Programmiersprache Python, da wir diese auch schon zuvor in einigen Studienprojekten benutzt hatten und sie für den hier gedachten Einsatzzweck eine einfache und schnelle Umsetzung verspricht.

2.2.5 Testprogramme

Im Verlauf der Programmierung der PHP-Skripte auf dem Webserver wurden insgesamt drei Python-Skripte geschrieben, um die Funktionen auf dem Webserver zu testen.

Das erste Python-Skript heißt „Senden.py“. Mit ihm soll getestet werden, ob gesendete Daten beim Webserver ankommen und auch weiterverarbeitet werden können. Dafür werden neben einer gültigen Geräte-ID auch je eine Variable für den Breitengrad (Latitude) und den Längengrad (Longitude) mit frei gewählten Startwerten deklariert. Ebenso wird die URL des empfangenden Skripts auf dem Webserver („http://retrorbit.spdns.de/get_Data.php“) als Variable gespeichert. Der restliche Code ist eine endlose while-Schleife, in der die aktuellen Daten ins JSON-Format gebracht und dann mittels HTTP-POST an die gespeicherte URL gesendet werden. Danach folgt noch eine kurze Wartezeit von 5 Sekunden sowie eine Inkrementierung der Latitude und eine Dekrementierung der Longitude jeweils um den Wert 1.

Im zweiten Python-Programm mit dem Namen „Abfragen.py“ werden innerhalb einer endlosen while-Schleife die aktuellen Werte vom Webserver bzw. von der Webseite „http://retrorbit.spdns.de/provide_Data.php“ per HTTP-GET abgefragt und die Daten im JSON-Format ausgegeben.

Das dritte Python-Skript heißt „Störer.py“ und gleicht dem „Senden.py“-Skript. Mit diesem sollten zeitgleich entweder beliebige oder auch vom Format her gleiche Werte an das „get_Data.php“-Skript gesendet werden. So wollten wir überprüfen, ob die Sicherheitseinrichtung mit den Geräte-IDs tatsächlich funktioniert.

2.2.6 Erstellen der PHP-Skripte zum Empfangen und Bereitstellen von Daten

Erläuerungen zum Code Bei allen drei übermittelten Parametern, also Geräte-ID, Latitude und Longitude, wird immer eine Prüfung durchgeführt, bevor deren Werte tatsächlich in Variablen des PHP-Skripts geschrieben werden. Die Struktur dieser Prüfung ist bei allen drei Parametern identisch, bis auf den entsprechenden Bezeichner und den Variablennamen. Die Variable übernimmt nur dann den Wert der $POST Daten, wenn diese einerseits von „null“ verschieden sind und andererseits nicht leer sind. Dies wird realisiert, indem in einer if-Abfrage das Ergebnis der „isset“-Funktion mit der Bedingung, dass der $POST Wert ungleich leer ist, über ein logisches UND verknüpft sind. Die „isset“-Funktion prüft dabei, ob die Variable existiert und sich von „null“ unterscheidet. Wenn diese Bedingungen allerdings nicht erfüllt sind, wird im else-Zweig mittels goto-Anweisung zum Ende des Skriptes gesprungen und eine Fehlermeldung ausgegeben. Die aktuell empfangenen Daten werden dann nicht weiterverarbeitet, sodass immer noch der vorherige Standort von der Seite „http://retrorbit.spdns.de/provide_Data.php“

abgefragt werden kann.

Mit der Funktion str_contains kann überprüft werden, ob ein String in einem anderen String enthalten ist. In unserem Fall wird so gecheckt, ob die Geräte-ID ($device_id) im String $all_devices enthalten ist, welcher alle Geräte-IDs aus der Datei „allowed_devices.txt“ beinhaltet.

Die aus den $POST Variablen übernommenen Werte für Latitude und Longitude werden dann zusammen mit den aktuellen Datumsinformationen in ein Array mit der Bezeichnung $koor kopiert. Anschließend wird dieses Array noch in JSON codiert und als $json_koor gespeichert. Wichtig ist an dieser Stelle der zweite Parameter des json_encode Befehls. Wenn hier nichts angegeben wird, werden alle enthaltenen Werte als String interpretiert. Für das Datum und die Uhrzeit ist das sinnvoll, für die Koordinaten hingegen eher nicht. Damit deren Werte auch als Zahlenwerte interpretiert werden, ist der Parameter „JSON_NUMERIC_CHECK“ zu setzen, welcher automatisch Zahlenformate erkennt und ins JSON Format übernimmt. Im letzten Schritt des „get_Data.php“ Skripts wird der aktuelle Inhalt der Datei „Koordinaten.txt“ mit den Daten der JSON-codierten Variable $json_koor überschrieben.

Die Trennung für Datenempfang und Datenbereitstellung hat folgenden wichtigen, technischen Hintergrund. In dem Moment, in dem ein http-Request an das „get_Data.php“ Skript gestellt wird – egal ob Daten mittels POST dorthin übermittelt werden oder nur die bereitgestellten Daten mittels GET abgefragt werden sollen – wird es einmal ausgeführt. Da bei einer reinen GET-Abfrage aber natürlich keine Daten mitgeschickt werden, wird der entsprechende PHP-Code zur Datenverarbeitung übersprungen und die Koordinaten mit „null“ gefüllt. Genau dies wird danach auch ausgegeben und von dem http-GET-Request ausgelesen. Somit ist es nicht möglich innerhalb eines PHP-Skripts Daten zu empfangen und bereitzustellen.

In der nachfolgenden Abbildung ist der grundsätzliche Programmablauf der beiden PHP-Skripte auf dem Webserver dargestellt, die die Datenverarbeitung sowie die Datenbereitstellung realisieren.

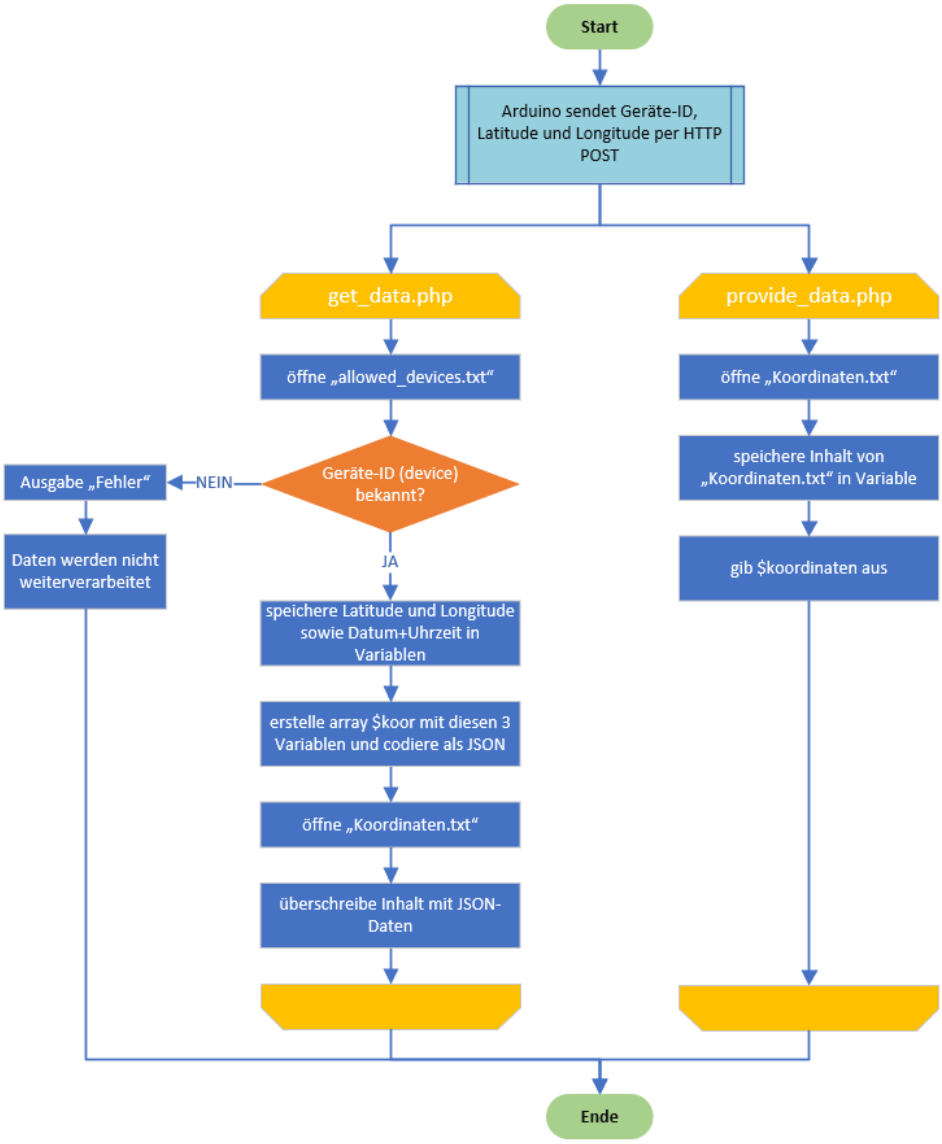

Abbildung 2.6: Programmablaufplan Webserver

Probleme beim Senden vom Arduino Nachdem das Skript „get_Data.php“ in einem funktionsfähigen Zustand war und auch das Testen mit dem Python-Programm „Senden.py“ keine Fehler ergab, konnte zum nächsten Entwicklungsschritt übergegangen werden. Dieser besteht darin, die Kommunikation vom Arduino über das GSM-Modul zum Webserver herzustellen, sodass dort die gesendeten Daten auch weiterverarbeitet werden können. Im Prinzip müssen im Programm auf dem Arduino nur die Koordinaten ins JSON Format gebracht werden und per http-POST an die URL „http://retrorbit.spdns.de/get_Data.php“

gesendet werden. Es erforderte einige Versuche, bis das überhaupt funktionierte, allerdings konnten die Daten auf dem Webserver nicht richtig verarbeitet werden. Dies äußerte sich darin, dass in der Datei „Koordinaten.txt“ für Latitude und Longitude der Wert „null“ eingetragen war. Das Auslesen der Daten mittels POST auf der Seite des Webservers hat also nicht funktioniert. Die Ursache hierfür lag bei der Datenformatierung des Arduinos bzw. dem AT-Befehl zum Senden der Daten via GSM. Dabei werden die Daten immer als String versendet unabhängig von der ursprünglichen Form, welche dann als String angesehen wird. Deshalb funktionierten im PHP-Skript die $_POST Befehle für die einzelnen Parameter nicht mehr. Mit diesem Wissen wurde das „get_Data.php“ Skript nochmals angepasst und als „get_Data2.php“ abgespeichert. Dieses ist im Prinzip genauso aufgebaut wie die ursprüngliche Version, jedoch wurden die einzelnen $_POST Befehle gegen einen normale Arraybezeichner ($POST_data) ausgetauscht. Dafür wird nach den Deklarationen der Dateinamen in Zeile 6 mit nur einem $_POST der vom Arduino gesendete String ausgelesen und in der Variable $Arduino_Daten gespeichert. Es folgt eine JSON-Decodierung, sodass nun die einzelnen Parameter (Geräte-ID und die beiden Koordinaten) im Array $POST_data gespeichert und referenzierbar sind. Der Zugriff auf die einzelnen Werte geschieht jetzt also nicht mehr über die $_POST Befehle, sondern per Referenzierung auf das Array.

2.2.7 Verschlüsselung der Daten

Hintergrund Im aktuellen Zustand des Webservers und unserer Kommunikation allgemein werden die Koordinaten unverschlüsselt und damit für jedes Gerät bzw. jeden Nutzer klar lesbar auf der Seite „http://retrorbit.spdns.de/provide_Data.php“ dargestellt. Sowohl aus datenschutztechnischer Sicht als auch im Interesse des späteren Kunden ist dies natürlich nicht. Deshalb ist es nötig, die bereitgestellten Daten zu verschlüsseln. Der wichtigste Punkt an dieser Stelle ist, dass die verwendeten Chiffrier- und Dechiffrieralgorithmen bzw. Bibliotheken sowohl von PHP als auch von den beiden Programmiersprachen der Apps unterstützt werden.

Symmetrische und asymmetrische Verschlüsselungsverfahren Die Verschlüsselung unserer Nutzdaten, also Latitude, Longitude sowie die dazugehörigen Datumsinformationen, soll mittels eines symmetrischen Verschlüsselungsverfahren realisiert werden. Symmetrische Verschlüsselungs-verfahren sind einerseits sehr performant, selbst bei großen Datenmengen, und andererseits sind sie nur unter unverhältnismäßig großem Aufwand zu entziffern. Ein weit verbreiteter, frei verfügbarer Standard stellt dabei der „Advanced Encryption Standard“, kurz AES, den auch wir bei unserem Projekt nutzen wollen. Bei symmetrischen Verschlüsselungsverfahren wird für das Ver- und Entschlüsseln ein und derselbe Schlüssel benutzt. Beim AES kann dieser Schlüssel wahlweise 128, 192 oder 256 Bit lang sein, wobei gilt: Je länger

der Schlüssel ist, desto höher ist auch die Sicherheit der verschlüsselten Daten. Bei der symmetrischen Verschlüsselung besteht aber ein wesentlicher Nachteil, denn der Schlüssel muss beiden Teilnehmern der Kommunikation bekannt sein. Wenn die Chiffrierung also auf Gerät A durchgeführt wird, muss neben den nun verschlüsselten Daten auch der AES-Schlüssel zu Gerät B übertragen werden. Je nach individuellen Gegebenheiten lässt sich dies ggf. auf einem zweiten Kommunikationsweg manuell durch den Menschen realisieren. In der Praxis ist das aber meist nicht möglich. Deshalb wird hier zusätzlich noch ein asymmetrisches Verschlüsselungsverfahren angewandt, mit welchem der AES-Schlüssel chiffriert wird. Diese Kombination beider Verschlüsselungsverfahren wird auch als Hybride Verschlüsselung bezeichnet. Bei asymmetrischen Verschlüsselungsverfahren hat jeder Kommunikationspartner ein eigenes Schlüsselpaar bestehend aus einem öffentlichen und einem geheimen Schlüssel. Der öffentliche Schlüssel wird immer für die Verschlüsselung von Daten beim jeweilig anderen Teilnehmer genutzt und der geheime Schlüssel für die Entschlüsselung der empfangenen Daten auf dem eigenen Gerät. Der Vorteil ist hier, dass der öffentliche Schlüssel bei der Übermittlung an den Kommunikationspartner nicht extra geschützt werden muss. Eines der bekanntesten asymmetrischen Verschlüsselungsverfahren ist das nach seinen Entwicklern benannte RSA-Verfahren. Da dieses aber schon veraltet ist, werden wir mit der Elliptic Curve Cryptography ein zeitgemäßeres Verfahren nutzen.

Implementierung der Verschlüsselung auf dem Webserver Der komplette Verschlüsselungsprozess ist in „encryption.php“ realisiert, also sowohl die Verschlüsselung der Nutzdaten als auch die Chiffrierung des AES-Schlüssels. Für die Übermittlung des öffentlichen Schlüssels wird zusätzlich das Skript „key_from_app.php“ erstellt. Allerdings ist die Implementierung noch nicht im Zusammenspiel mit einer der Apps getestet worden, da es hier Verzögerungen im Projekt gab, sodass das ganze Verschlüsselungsthema erst in naher Zukunft tatsächlich funktionsfähig sein wird. Für die symmetrische Verschlüsselung nutzen wir in PHP die Funktionen der „OpenSSL“ Bibliothek und für die asymmetrische Verschlüsselung kommen die Funktionen der „Sodium“ Bibliothek zum Einsatz, welche auch für die von uns verwendeten Programmiersprachen der Apps zur Verfügung stehen.

Als erstes werden die zu verschlüsselnden Nutzdaten (Koordinaten und Datum mit Uhrzeit) aus der Datei „Koordinaten.txt“ geladen. Anschließend wird das Verschlüsselungsverfahren auf AES-192-CTR festgelegt und ein entsprechend langer AES-Schlüssel generiert. CTR steht dabei für Counter Mode, welcher zusätzlich einen Initialisierungsvektor erfordert, welcher die gleiche Länge wie der AES-Schlüssel haben muss. Nachdem dieser Initialisierungsvektor generiert wurde, folgt die eigentliche Verschlüsselung unserer Nutzdaten mit der Funktion „openssl_encrypt“. Weiter geht es mit der Generierung des Schlüsselpaares des Webservers für die asymmetrische Verschlüsselung mit der Funktion „sodium_crypto_box_keypair()“.

Nun fehlt noch der öffentliche Schlüssel aus der App. Dieser wird aus der Datei „app_key.txt“ geladen. Diese Datei wurde im Skript „key_from_app.php“ generiert und lokal auf dem Server gespeichert. Dieses Skript ist im Prinzip genau so aufgebaut wie das „get_Data.php“ Skript, nur dass es entsprechend nur auf die beiden Parameter „Geräte-ID“ und „keypair2_public“ abgestimmt ist. So findet auch hier zuerst eine Überprüfung statt, ob die gesendeten Daten von einem bekannten, erlaubten Gerät stammen.

Nachdem dann also alle notwendigen Schlüssel auf dem Webserver vorhanden sind, kann der AES-Schlüssel mit der Funktion „sodium_crypto_box()“, welche noch eine zusätzliche Zufallszahl (nonce) benötigt, verschlüsselt werden. Diese Zufallszahl wurde zuvor erzeugt und muss auch an den Empfänger für die Entschlüsselung übermittelt werden. Im nächsten Schritt werden wieder alle zu sendenden Variablen in ein Array ($datenpaket) kopiert, siehe untenstehende Abbildung. Dabei ist es wichtig, dass alle Variablen bis auf die verschlüsselten Nutzdaten noch einmal mit base64 codiert werden, da sonst bei der JSON-Codierung des Arrays im nächsten Schritt nicht alle Zeichen der einzelnen Strings korrekt verarbeitet werden können. Bevor diese Strings dann allerdings in den Apps genutzt werden können, muss dort wieder eine base64 Decodierung erfolgen, um wieder das BYTES Format zu erhalten.

Im letzten Schritt werden nun die JSON codierte Variable $json_data in der Datei „Datenpaket.txt“ lokal abgespeichert. Diese Datei kann dann genau wie zuvor die Datei „Koordinaten.txt“ von dem Skript „provide_Data.php“ geöffnet und deren Inhalt ausgegeben werden. So können die Entschlüsselungsparameter zusammen mit den verschlüsselten Daten von den Apps abgefragt werden.

```
46  $datenpaket = array(
47      "key1pub" => base64_encode($keypair1_public),
48      "nonce" => base64_encode($nonce),
49      "AES_key" => base64_encode($encrypted_AES_key),
50      "AES_iv" => base64_encode($iv),
51      "Nutzdaten" => $encrypted_Nutzdaten
```

Abbildung 2.7: Datenpaket Webserver

Der gesamte Ablauf der Skripte auf dem Webserver ist in der nachfolgenden Übersicht zur besseren Nachvollziehbarkeit noch einmal schematisch dargestellt.

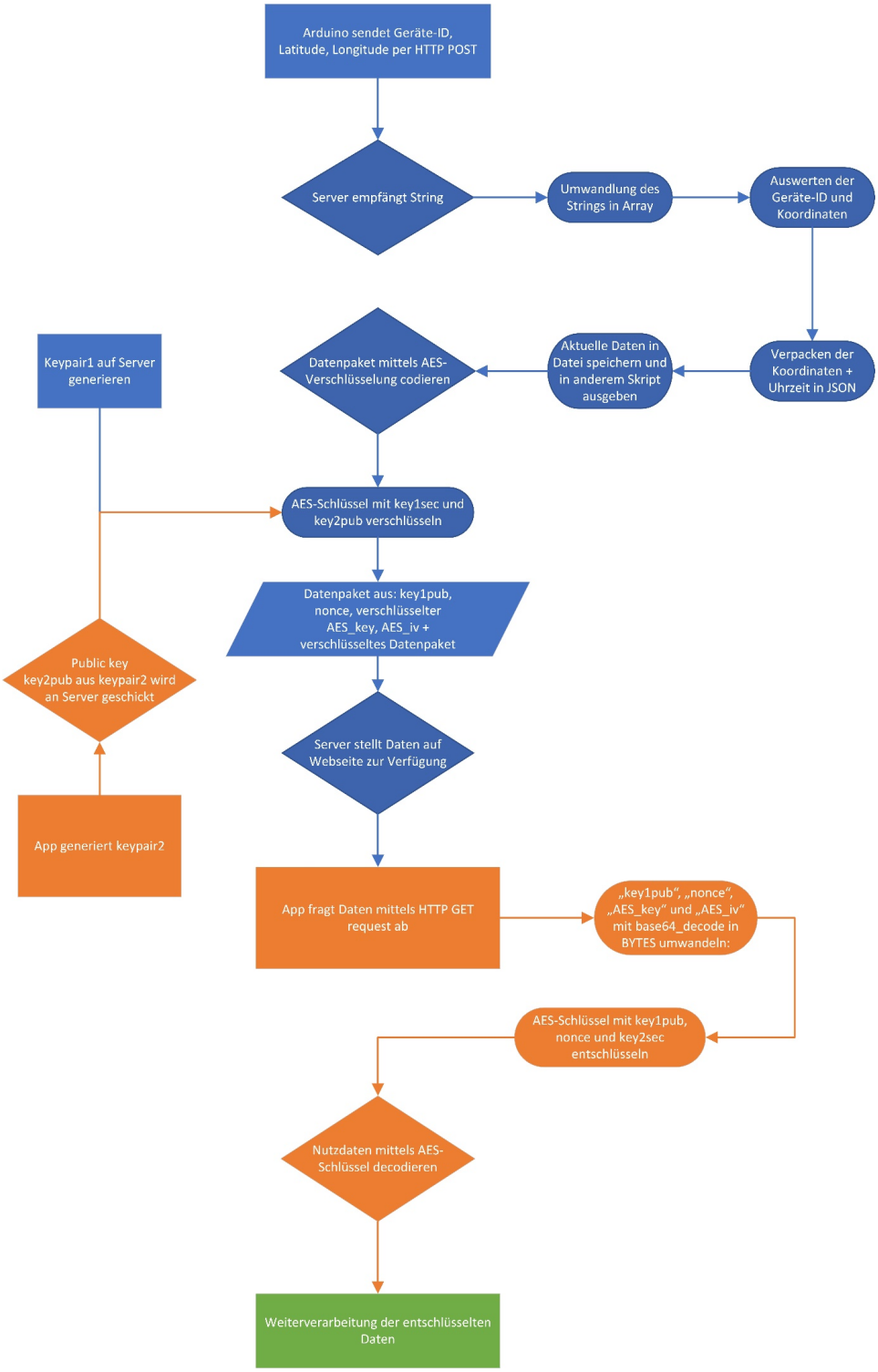

Abbildung 2.8: Skriptablauf Webserver

2.3 App - Android

2.4 App - iOS

Kapitel 3

Produkt

3.1 Produkt

3.1.1 zukünftiges Produktangebot

Wir haben uns zwei Produktkategorien überlegt, mit denen wir Gewinn generieren können. Zum einen ist das ein monatliches Bezahlkonzept. Hierbei haben wir uns an dem Konzept einiger Mobilfunkanbieter orientiert. Die Überlegung ist, dass wir zwei hinzu buchbare Optionen anbieten und man nur das bezahlt, was wirklich gebraucht wird. Eine Optionen ist die Google Standort API für eine genauere und zuverlässigere Standorterkennung. Die andere Option ist das Bestellen einer Sim-Karte mit Mobilfunkvertrag. Die Idee ist hier, dass so gegen einen kleinen Aufpreis der Ease-of-Use unseres Produkts weiter gesteigert wird.

Für das Endgerät haben wir uns neben der aktuellen Basis Variante für eine Base+ und eine Pro Variante entschieden. Bei der Base+ Variante ergänzen wir die Basis Variante lediglich um eine Sirene und etwas mehr Akkuleistung. Für die Pro Variante erweitern wir das Basismodell um einen WLAN Empfänger. Des Weiteren überlegen wir unseren modularen Aufbau für eine weitere Umsatzsteigerung zu nutzen und Akkumodule zu verkaufen, mit denen die Akkulaufzeit unserer Produkte vom Nutzer beliebig verlängert werden kann. Ein weiterer Vorteil ist, dass so bei nachlassender Akkuleistung unser Produkt nicht entsorgt werden muss, sondern der Nutzer einfach mit offiziellen Produkten den Akku wechseln kann. Dies soll verhindern, dass der Kunde auf Drittanbieter zurückgreift. Das Ziel ist, dass wir unsere Apps und den Webserver unabhängig von der Produktversion verwenden können.

3.1.2 Vergleich

Moni

Kapitel 4

Rück- und Ausblick

4.1 Rück- und Ausblick

Soll/Ist Vergleich Lessons Learned (User Umfrage nutzen)

4.1.1 Zukünftige Entwicklung

Arduino: Als Erweiterungen für den Arduino sollen in Zukunft noch eine Alarmsirene vom Typen BSE128 verbaut werden und für die Standorterkennung noch die Google API zur beseren Lokalisierung genutzen werden. Für die Sirene muss das Gesamtsystem noch dahingehend erweitert werden, dass eine Kommunikation zum Arduino hin möglich wird. Auf dem Arduino muss hierfür noch realisiert werden, dass der Rückgabewert vom Server abgespeichert wird und in regelmäßigen Abständen geprüft wird, dass sich das Gerät nicht aus dem vom Anwender definierten Bereich entfernt hat. Es muss also noch Geofencing ermöglicht werden, um zu erkennen wann die Sirene auslösen soll. Es ist aber auch eine einfache Lösung denkbar, bei der der Benutzer den Alarm aus der Entfernung selbst steuern muss bzw kann.

Aus Hardware Sicht muss dann nur die Sirene mittels Transistorschaltung an den Arduino angeschlossen und mit Strom versorgt werden, damit der Arduino das Alarmsignal im Fall einer Entwendung geben kann. Die Schaltung dafür sieht wie folgt aus:

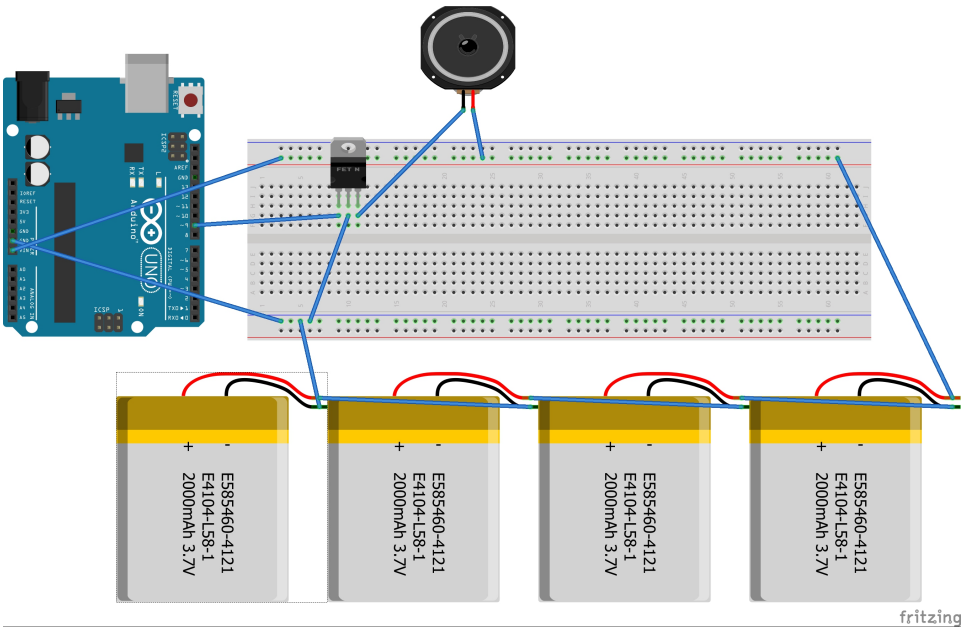

Abbildung 4.1: Systemerweiterung um Sirene

Dabei kann die Sirene, da sie 12-24V Eingangsspannung akzeptiert, an die Stromversorgung des Arduinos mit angeschlossen werden. Es muss dafür jedoch eventuell die Akkuleistung ebenfalls erweitert werden. Dies muss jedoch erst noch getestet werden.

Die Sirene soll zukünftig durch eine einfache N-Channel Mosfet Schaltung gesteuert werden. Um den Google Standort Service effektiv verwenden zu können, ist die Erweiterung um einen WLAN Empfänger sinnvoll. Somit kann neben den Informationen, die durch GSM und GPS zur Verfügung stehen, auch die lokalen WLAN-Netze zur Standortermittlung genutzt werden. Dies erhöht weiter die Genauigkeit kann aber theoretisch auch wegfallen. Die Verwendung der API ist jedoch kostenpflichtig und dementsprechend ist das Geschäftskonzept eventuell weiter zu überdenken. Eine mögliche Lösung hierfür wäre ein monatliches Bezahlkonzept, über das man je nach Bedarf die API Standorterkennung hinzu buchen kann.

Webserver

Finalisierung der Verschlüsselung Der erste Punkt, der in Zukunft bearbeitet werden soll, ist die vollständige Implementierung und ein anschließendes Testen der Verschlüsselung der Nutzdaten. Für den Webserver ist wie zuvor beschrieben bereits alles fertig. Es gilt nun also die entsprechend umgekehrten Abläufe in den beiden Apps zu realisieren.

Darüber hinaus sollte geprüft werden, ob es auch möglich ist, die Koordinaten bereits auf dem Arduino zu verschlüsseln. So könnte zusätzliche Sicherheit der Daten gewonnen werden.

Umstellung auf HTTPS Bei der Entwicklung der iOS-App trat an der Stelle des Auslesens der JSON codierten Daten ein Problem auf. Die Fehlermeldungen waren zunächst nicht eindeutig zu interpretieren, doch es stellte sich heraus, dass mit der von uns gewählten Version nur von HTTPS-Seiten JSONs mittels HTTP-GET abgefragt werden können.

Um derartige Probleme in Zukunft zu umgehen, soll der Webserver von HTTP auf HTTPS umgestellt werden. Wir hatten auch erst überlegt, dies im aktuellen Bearbeitungszeitraum umzusetzen, jedoch fand sich für die iOS-App eine Alternativlösung. Auch erste Recherchen zur Umstellung auf HTTPS zeigten uns, dass dies doch ziemlich aufwendig werden und wieder neue Probleme mit sich bringen könnte.

Bidirektionale Kommunikation ermöglichen Ein weiteres Feature, das in Zukunft realisiert werden soll, ist eine Kommunikation auch in die andere Richtung. Es soll also dann möglich sein, dass die App Befehle oder sonstige Daten an den Webserver schickt und der Arduino diese dann auch abfragen kann. Eine solche Kommunikation wäre zum Beispiel für das Aktivieren der ebenfalls noch geplanten Sirene im Innenraum des Fahrzeugs erforderlich.

MQTT als redundanten Übertragungsweg einrichten Um nicht nur den bisherigen Übertragungsweg über den HTTP-Server nutzen zu können und somit anfällig für Systemausfälle zu sein, soll ein zweiter, redundanter Übertragungsweg mit Hilfe des MQTT-Protokolls errichtet werden. Hierfür ist als erstes das Bereitstellen eines eigenen MQTT-Brokers erforderlich, um direkt unabhängig von Dritten zu sein. Auch muss ein MQTT-Topic definiert werden, in welchem die Kommunikation stattfinden soll. Im nächsten Schritt muss auf dem Arduino ein entsprechendes Skript programmiert werden, mit welchem die Koordinaten mittels PUBLISH zum MQTT-Broker übermittelt werden können. Dieses muss dann entweder parallel zum bisherigen Skript ausgeführt werden oder es werden beide in einem neuen Skript vereinigt.

Anschließend muss in den beiden Apps ein MQTT-SUBSCRIBE implementiert werden, um die vom Arduino veröffentlichten Daten zu erhalten. Auch dies muss parallel zur Datenabfrage vom Webserver geschehen. Zusätzlich soll in den Apps dann ein Abgleich beider Daten (HTTP und MQTT) implementiert werden. Sind beide Daten gleich werden die neuen Koordinaten übernommen. Für den anderen Fall, dass die beiden Datenpakete unterschiedlich sind, muss noch überprüft werden, welche Daten dann gespeichert werden.

Android Seif

iOS Moni

Abbildungsverzeichnis