Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 307

Entwurf und Implementierung von Schutzmechanismen zur sicheren Verfolgung privater Bluetooth Low Energy Geräte

Marius Maaß

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Kurt Rothermel
Betreuer/in:	Dr. Frank Dürr
Beginn am:	1. Februar 2016
Beendet am:	1. August 2016
CR-Nummer:	C.2.4, K.4.1

Kurzfassung

Die Verfolgung eines mobilen Gerätes ist eine nützliche Möglichkeit für den alltäglichen Gebrauch, allerdings sollte diese Verfolgung nur befugten Personen möglich sein. Bluetooth Low Energy-Beacons bieten eine Möglichkeit dies zu implementieren ohne die Mobilität eines mobilen Geräts einzuschränken. Da BLE-Beacons sehr energieeffizient arbeiten können, ist es möglich ein solches Beacon für mehrere Monate oder Jahre zu betrieben ohne die Batterie wechseln zu müssen. In dieser Arbeit wird ein System vorgestellt und implementiert, das dem Besitzer eines BLE-Beacons die Verfolgung dieser Geräte ermöglicht, ohne dass Dritten diese Verfolgung auch möglich ist. Hierfür verwendet ein Beacon Pseudonyme, die sich periodisch ändern, ohne dass ein Angreifer das nächste Pseudonym vorhersagen kann, obwohl der Besitzer des Beacons hierzu in der Lage ist. Diese Pseudonyme werden von einem Sensornetz aus Smartphones empfangen und an einen zentralen Server weitergeleitet, welcher es dem Besitzer ermöglicht die Position des eigenen Beacons abzufragen. Um die Energieeffizienz der Beaconimplementierungen zu testen wurde der Energieverbrauch der Beacons gemessen. Hierbei wurde eine Laufzeit von ca. einem Jahr auf einer üblichen 3 V Knopfzelle errechnet.

Abstract

Tracking a mobile device is a useful ability for day to day use. However, tracking the device should only be possible for people who are authorized to do so. Bluetooth Low Energy Beacons provide the means to do this without compromising the mobility of the device. Due to the low energy usage of BLE it is possible to operate a Beacon for multiple years without having to change the battery. This thesis presents and implements a system which allows the owner of a BLE beacon to track their devices without exposing this information to third parties. To do this it uses pseudonyms which change periodically to obfuscate the identity of a beacon. The used pseudonyms can be regenerated by the owner because only they posses the necessary information to do this. The BLE signals of the beacons containing the pseudonyms are received by a sensor net of smartphones which add their own position and relay this information to a central server which stores this data. This server allows the owner to retrieve the position of their beacon and track its movements. The energy efficiency of the beacons was analyzed by measuring the energy consumption. This showed that a beacon could operate for up to one year on a typical 3 V button cell.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AES Advanced Encryption Standard

API Application programming interface

BLE Bluetooth Low Energy

CBC Cipher Block Chaining

DK Development Kit

EID Ephemeral Identifiers

GAP Generic Access Profile

GATT Generic Attribute Profile

ID Identifikation

JSON JavaScript Object Notation

MAC Media Access Control

REST representational state transfer

SDK Software Development Kit

SdR Software defined Radio

SoC System on Chip

SPI Serial Peripheral Interface

UID Unique Identifier

URL Uniform Resource Locator

UUID Universally Unique Identifier

VANet Vehicular ad hoc network

Verzeichnis der Listings

Verzeichnis der Algorithmen

1 Einleitung

Bluetooth Low Energy (BLE) ist ein neuer, energiesparender Teil des Bluetooth Standards, der sehr schnell eine weite Verbreitung in der mobilen Branche erreicht hat. Dieser Standard ermöglicht es Geräte ohne externe Energieversorgung über einen längeren Zeitraum, ggf. auch über Jahre hinweg, zu betreiben. es Geräte zu betreiben, die über Jahre hinweg ohne externe Stromversorgung betrieben werden können.

BLE-Beacons benutzten diesen Standard, um Smartphones in ihrer Umgebung Dienste anzubieten, jedoch kann diese Technologie auch dafür verwendet werden, um die Position eines solches Beacons zu verfolgen. Dies ist für viele Anwendungsbereiche eine nützliche Funktion, allerdings kann dies auch Probleme für die Privatsphäre erzeugen, wenn ein Beacon verfolgt wird, ohne dass der Besitzer des Beacons dies bemerkt. Da die Signale eines BLE-Beacons von jedem empfangen werden können, müssen zusätzliche Maßnahmen ergriffen werden, um die Verfolgung zu verhindern. Gleichzeitig sollte es allerdings möglich sein, dass der Besitzer die Position des Beacons weiterhin verfolgen kann.

Um dies umzusetzen wird ein Sensornetz aus Smartphones benutzt, die die BLE-Signale der Beacons empfangen und an einen zentralen Server weiterleiten. Um die Position des Beacons bestimmen zu können, melden die Smartphones, die ein Signal des Beacons empfangen haben, ihre eigene Position an den Server wodurch dieser die ungefähre Position des Beacons bestimmen kann.

Der Server kennt durch diese Informationen zu jeder Zeit die ungefähre Position jedes Beacons und könnte so Standorte oder Bewegungen der Beacons verfolgen. Um dies zu verhindern werden von den Beacons Pseudonyme verwendet, die nur eine bestimmte Zeit lang gültig sind und danach nicht mehr verwendet werden.

Übersicht

Diese Arbeit ist in die folgenden Kapitel aufgeteilt:

Kapitel 2 – Grundlagen In diesem Kapitel werden die für die Arbeit notwendigen Themen und Konzepte erklärt.

Kapitel 3 – Verwandte Arbeiten In diesem Kapitel werden verwandte Arbeiten genannt und in Zusammenhang mit dieser Arbeit gesetzt.

Kapitel 4 – Systemmodell und Problemstellung Um die Rahmenbedingungen für den Entwurf und Implementierung zu schaffen wird ein System- und Angreifermodell entworfen, dass beschreibt welche Teilnehmer existieren, wie diese interagieren und wie ein möglicher Angreifer vorgehen könnte.

Kapitel 5 – Entwurf der Schutzmechanismen In diesem Kapitel wird das System entworfen und die benutzten Schutzmechanismen werden beschrieben.

Kapitel 6 – Implementierung Hier wird die Implementierung des zuvor beschriebenen Designs beschrieben und es wird auf die implementierungsspezifischen Eigenschaften der einzelnen Implementierungen eingegangen.

Kapitel 7 – Evaluation Die Implementierungen werden in diesem Kapitel in Bezug auf Stromverbrauch und Effektivität der Schutzmechanismen evaluiert.

Kapitel 8 – Zusammenfassung und zukünftige Arbeiten Die Arbeit wird zusammengefasst und mögliche zukünftige Arbeiten werden beschrieben.

2 Grundlagen

In dieser Arbeit werden verschiedene grundlegende Technologien und Konzepte benutzt, die in diesem Kapitel erklärt werden.

2.1 Bluetooth Low Energy

Bluetooth ist ein Standard zur drahtlosen Übertragung von Daten über kurze Distanzen. Das Protokoll ist sehr weit verbreitet und wird für viele Anwendungszwecke, wie zum Beispiel Freisprecheinrichtungen, Übertragung von Musik und viele andere Anwendungen benutzt. Als Teil der Version 4.0 wurde Bluetooth Low Energy (BLE) [Blu14] in den Bluetooth Standard aufgenommen. Dieser Teil stellt ein neues Protokoll bereit, welches deutlich weniger Strom verbrauchen soll als vorherige Versionen des Bluetooth-Standards.

Bei BLE wird zwischen zwei Arten von Geräten unterschieden. **Peripherals** sind kleine schwache Geräte, die Daten zur Verfügung stellen. **Centrals** sind relative leistungsstarke Geräte, wie zum Beispiel ein Smartphone oder ein Tablet, die als Empfänger und Verarbeiter dieser Daten fungieren. Centrals sind in der Lage, eine Verbindung mit einem Peripheral zu initiieren, wenn ein solches in der Umgebung gefunden wurde.

BLE benutzt das Generic Access Profile (GAP) und das Generic Attribute Profile (GATT) um Daten zu versenden. GAP wird in Abschnitt 2.1.1 näher beschrieben. Das GATT-Protokoll legt den Austausch von Daten zwischen Peripherals und Centrals fest. Da bei Bluetooth Beacons keine Zwei-Wege-Kommunikation stattfindet ist dieser Anwendungsfall für BLE-Beacons nicht relevant, weshalb hier nicht weiter auf dieses Protokoll eingegangen wird.

2.1.1 GAP

GAP beschreibt als Teil des BLE-Standards die Kommunikation von BLE-Geräten mit der Außenwelt. Ein Peripheral versendet periodisch Advertisment-Pakete, die von einem Central empfangen werden können. Diese Advertisment-Pakete können dazu benutzt werden, um anderen Geräten eine mögliche Verbindung anzuzeigen oder um generische Daten zu versenden, ohne dass hierzu eine Verbindung zwischen den Kommunikationsparteien hergestellt wird. Dies wird zum Beispiel bei BLE-Beacons benutzt, da diese keine Verbindungen benötigen, um ihre Dienste anzubieten. In diesem Fall wird das Payload-Feld in einem BLE-Advertisment-Packet

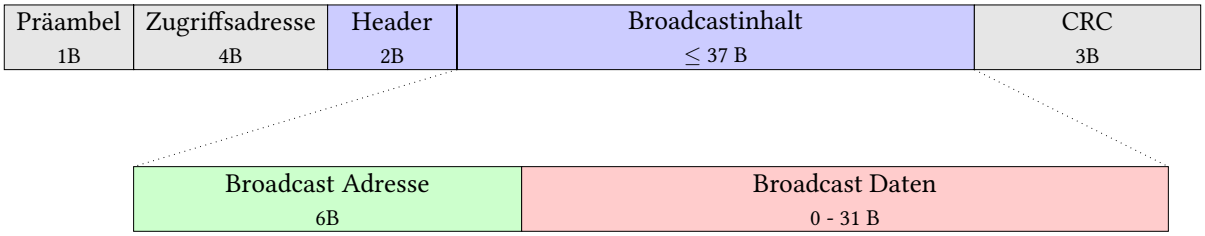


Abbildung 2.1: Format eines GAP-Broadcast Pakets. Ein Paket enthält mehrere Felder, die von der BLE-Implementierung gesetzt werden (hier grau dargestellt). Der Inhalt (hier blau dargestellt) eines Pakets besteht aus den Kopfdaten und dem eigentlichen Inhalt des Pakets. Der Inhalt ist abhängig von der Art des Pakets. Hier ist ein Broadcast-Paket dargestellt, dass aus einer Adresse (grün) und den Daten (rot) besteht.

benutzt, um die Nutzdaten zu kodieren. Diese Nutzdaten können vollkommen willkürlich formatiert werden, um den gewünschten Verwendungszweck zu verwirklichen. Wenn die Daten ohne eine Verbindung herzustellen versendet werden wird dies als „Broadcast“-Modus bezeichnet. In diesem Modus können bis zu 31 Byte Nutzdaten versendet werden. Der Aufbau eines Broadcast-Pakets ist in Abbildung 2.1 gezeigt. Alle Pakete haben einige Felder gemein, die von der BLE-Implementierung gesetzt werden, diese sind hier grau dargestellt. Der Inhalt besteht aus einer sechs Byte langen Media Access Control (MAC)-Adresse und den bis zu 31 Byte langen Nutzdaten.

2.2 Bluetooth Beacons

Bluetooth-Beacons sind Geräte, die mittels einer Bluetooth-Nachricht Geräten in der Umgebung mitteilen, dass sie existieren oder Dienste bereitstellen. Diese können zum Beispiel dafür benutzt werden, um auf einem Smartphone eine bestimmte Aktion auszuführen (zum Beispiel die Anzeige einer Meldung oder dem Aufruf einer Webseite). Es ist auch möglich, Bluetooth-Beacons zur Positionsbestimmung innerhalb von Gebäuden zu benutzen (wie z.B. in [Cha08] gezeigt). Hierbei konnte durch die geringe Reichweite von Bluetooth eine sehr genaue Position ermittelt werden.

Es gibt bisher keinen allgemeinen Standard zur Implementierung oder zum Betrieb von Bluetooth-Beacons, jedoch wurden bereits mehrere Spezifikationen mit Vorschlägen und Beispielen einer Implementierung vorgestellt. Die zwei am weitesten verbreiteten Spezifikationen sind derzeit „iBeacon“ von Apple und „Eddystone“ von Google. Beide Spezifikationen zielen vor Allem auf den mobilen Bereich ab, da hier Bluetooth-Beacons besonders interessant sind.

2.2.1 iBeacon

Die von Apple vorgestellte Spezifikation ist darauf ausgelegt, im iOS Smartphone-Betriebssystem benutzt zu werden. iOS unterstützt diese Beacon-Spezifikation auf dem Betriebssystemlevel und ermöglicht es einer App darauf zu reagieren, wenn ein solches Beacon in der Umgebung gefunden wurde. iBeacons benutzen BLE um per Funk ihre Identifikation (ID) zu versenden. Die Spezifikation ist nicht öffentlich zugänglich, jedoch wurde das Format der iBeacon-Pakete per Reverse-Engineering analysiert [War14]. Ein Beacon sendet eine bestimmte Universally Unique Identifier (UUID) aus, die den Besitzer des Beacons angibt. Dies könnte zum Beispiel eine Firma sein, die mit iBeacons einen Dienst anbietet. Anschließend gibt es noch zwei Byte Felder die „Major“ und „Minor“ genannt werden. „Major“ kann benutzt werden, um eine Untergruppe aller Beacons zu klassifizieren. Alle Beacons, die den gleichen „Major“ Wert haben gehören dann zu dieser Gruppe. Dies kann von App-Entwicklern genutzt werden, um die Informationen dieser Beacons zu gruppieren. „Minor“ ist innerhalb dieser Gruppe eindeutig und dient meist der eindeutigen Identifikation eines einzelnen Beacons. So könnten mit diesem Mechanismus spezielle Daten mit einem ganz bestimmten Beacon verknüpft werden. Zusätzlich ist in den versendeten Daten auch die Leistung des Senders (in dBm in nächster Nähe zum Beacon) enthalten. Diese ermöglicht es einem Empfänger, die Distanz zum Beacon abzuschätzen, da der BLE Empfänger die Stärke des empfangenen Signals kennt und somit mittels der versendeten und gemessenen Werte die Distanz berechnen kann.

2.2.2 Eddystone

Eddystone [Goo16] ist eine von Google vorgestellte Spezifikation für BLE-Beacons. Im Gegensatz zu Apples iBeacons ist diese Spezifikation offengelegt und auf GitHub verfügbar. Auch Google benutzt diese Beacons, um zusätzliche Dienste anzubieten. So verknüpft Google mit diesen Beacons Daten, die über die Nearby API [Goo] abgefragt werden können. Diese Dienste sind direkt in das Android Betriebssystem integriert. Die direkte Integration erlaubt es auf Beacons zu reagieren ohne eine zusätzliche App zu installieren. Es ist möglich mit einem Beacon eine Uniform Resource Locator (URL) zu verknüpfen, die aufgerufen wird wenn das Smartphone in der Nähe des Beacons ist. Diese URLs erlauben es zum Beispiel eine positionsbezogene Webseite aufzurufen oder eine App zu starten.

Im Gegensatz zur iBeacon-Spezifikation definiert Eddystone mehrere BLE Frame Typen, die unterschiedliche Bedeutungen haben. Alle Eddystone Frames werden im passiven Advertisment Modus versendet, weshalb alle für einen Frame relevanten Daten in ein BLE-Frame passen müssen. Die unterschiedlichen Arten der Frame-Typen werden im Folgenden näher erläutert.

UID-Frame Dies ist der zentrale Frame-Typ. In diesem Frame wird ein 16-Byte langer Unique Identifier (UID) versendet, der das Beacon eindeutig identifizieren soll. Diese UID besteht aus zehn Byte Namespace- und sechs Byte Instance-Werten. Dies ermöglicht

die Vergabe eines einheitlichen Namespace-Wert für eine Gruppe von Beacon, welche dadurch zur selben logischen Gruppe gehören. Wie bei den iBeacons enthält auch dieser Frame die Sendeleistung des Beacons, die dann eine Distanzabschätzung des Empfängers ermöglicht.

URL-Frame Der URL-Frame versendet eine speziell kodierte URL, auf die Klienten zugreifen können, sobald sie dieses Frame empfangen. Da der Platz innerhalb eines BLE Frames sehr begrenzt ist, ist diese URL speziell kodiert, wodurch auch längere URLs übertragen werden können. Diese spezielle Kodierung benutzt nicht druckbare ASCII-Zeichen, um Domänennamen zu verkürzen: („`.org`“ wird zum Beispiel durch den Byte-Wert 8 kodiert). Zusätzlich kann ein Internet-Dienst, wie zum Beispiel `goo.gl`, benutzt werden, der URLs verkürzt, um die unkodierte Form der URLs möglichst kurz zu halten.

TLM-Frame Eddystone sieht zusätzlich einen TLM-Frame für die Übermittlung von Telemetriedaten der Beacons vor, Dies beinhaltet die aktuelle Spannung der Batterie, womit die verbleibende Kapazität der Batterie und damit die verbleibende Laufzeit berechnet werden kann. Zusätzlich wird die aktuelle Temperatur, die Anzahl der versendeten Frames und die Zeit seit dem letzten Reset mitgeliefert. Dies kann zum Beispiel benutzt werden, um den Status mehrerer Beacons zu verwalten. Die TLM-Frames sind nicht direkt mit einem UID-Frame verbunden, da hierzu nicht genügend Platz in einem Frame vorhanden wäre, um alle Daten zu versenden. Stattdessen muss sichergestellt werden, dass ein TLM Frame möglichst zeitnah zu einem dazugehörigen UID-Frame versendet wird, so dass die beiden Frames miteinander verknüpft werden können. Diese Information kann dann dazu genutzt werden, um die verteilten Beacons zu überwachen und zum Beispiel die Batterie eines Beacons auszutauschen, wenn diese am Ende ihrer Lebensdauer angekommen ist.

EID-Frame Wenn ein Beacon eine statische UID benutzt, so kann jeder Benutzer diese empfangen und damit die, mit dem Beacon verknüpften Daten abzufragen. Es ist auch möglich, die UID des Beacons zu kopieren und an einem anderen Ort auszusenden. Für viele Anwendungsfälle stellt dies kein Problem oder sogar gewollt. Jedoch gibt es auch Anwendungsfälle, in denen es nützlich wäre den Kreis der erlaubten Klienten einzuschränken. Für diesen Anwendungsfall können Ephemeral Identifiers (EID) Frames [HMYZ16] benutzt werden. Diese spezielle Art der UID-Frames enthält ebenfalls eine ID, jedoch ändert sich diese periodisch und sollte nicht vorhersagbar sein. Dies wird über eine kryptografisch sichere Pseudozufallsfunktion realisiert. Das Beacon hat hierbei einen symmetrischen Schlüssel, mit dem neue IDs berechnet werden können. Dieser Schlüssel ist auch den Google-Servern bekannt, wodurch diese eine EID einem bestimmten Beacon zuordnen können. Mit diesen Informationen kann der Google-Server entscheiden, ob der Klient, der die Anfrage gestellt hat, auch berechtigt ist, die Daten des Beacons zu erhalten.

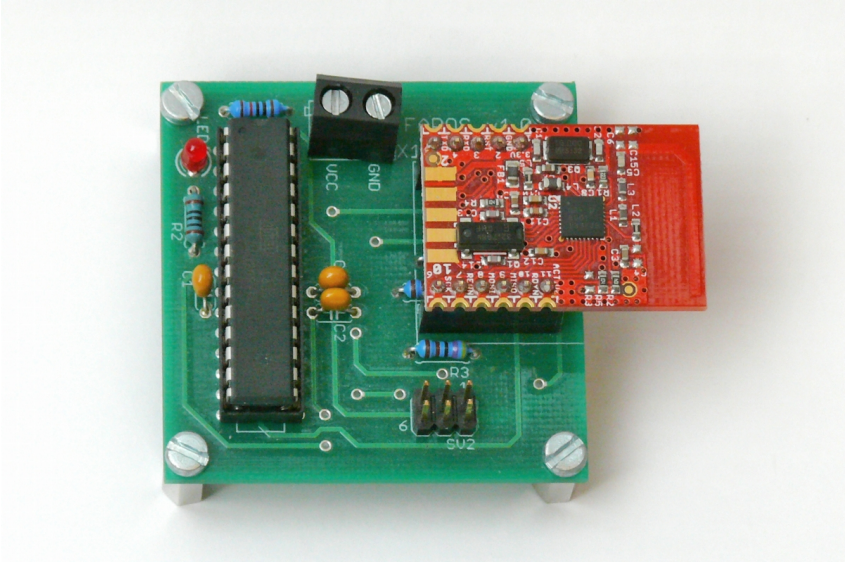


Abbildung 2.2: Faros-Beacon. Das BLE Modul ist das rötliche Bauteil auf der rechten Seite. Auf der linken Seite ist der Mikroprozessor zu sehen.

2.3 Zielplattformen

Da BLE-Beacons eine möglichst lange Laufzeit ohne externe Stromversorgung haben sollen, muss dementsprechend spezielle Hardware genutzt werden. Im Folgenden werden zwei Plattformen vorgestellt, die in dieser Arbeit verwendet werden.

2.3.1 Faros Beacon

Das Faros Beacon [Dür16a] (siehe Abbildung 2.2) ist eine für die Lehre an der Universität Stuttgart entwickelte BLE Beacon-Plattform. Sie besteht einem ATmega328P Mikroprozessor und dem über Serial Peripheral Interface (SPI) angeschlossenen nRF8001 BLE Chip. Der ATmega328P [Atm] ist eine 8-bit CPU mit 2 kB SRAM und einer maximalen Taktfrequenz von 20 MHz. Als persistenter Speicher sind 32 kB an Flash Speicher verfügbar. Der nRF8001 [Norb] ist ein von Nordic Semiconductors entwickelter BLE-Chip, der die volle Bluetooth Low Energy 4.0 Spezifikation unterstützt. Dieser Chip ist darauf ausgelegt, mit sehr wenig Strom betrieben zu werden und ist dadurch ideal für die Verwendung mit dem Faros Beacon.

2.3.2 nRF51 SoC

Die nRF51 Serie [Nora] ist eine System on Chip (SoC) Familie die auf funkgestützte Anwendungen mit einem sehr geringen Energieverbrauch ausgelegt ist. Sie benutzt eine ARM Cortex-M0 CPU, die auf einen geringen Stromverbrauch optimiert ist. Für die Funkanwendungen benutzt

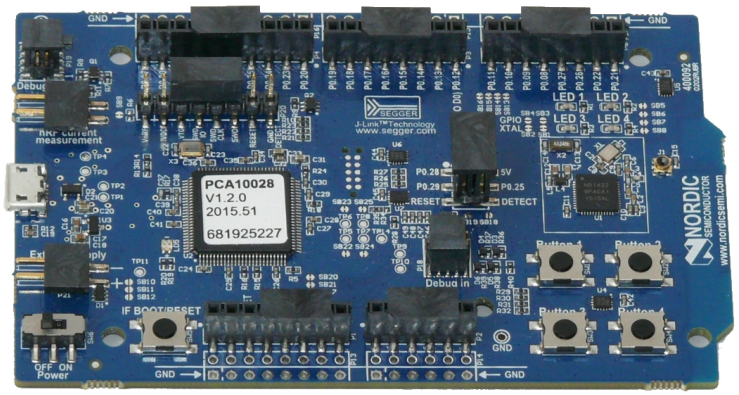


Abbildung 2.3: Das nRF51 Developer Kit, das für die Entwicklung benutzt wurde.

diese Plattform Software defined Radio (SdR) in Form von sogenannten „Softdevices“. Diese Softdevices sind ausführbarer Code, die ein bestimmtes Funkprotokoll implementieren. Für den nRF51 stehen mehrere Softdevices zur Verfügung, die, unter anderem, das BLE-Protokoll implementieren, weshalb diese Plattform für dieses Projekt interessant ist.

Für die Entwicklung gibt es eine spezielle Variante des SoC, die mehr Funktionen unterstützt als das Endprodukt. So unterstützt dieses Developers Kit zum Beispiel das Debugging des Codes über den J-Link Debugger von Segger [SEG] und hat zusätzlich vier LEDs und vier programmierbare Taster, die während der Entwicklung benutzt werden können. Ein solches Entwicklungskit ist in Abbildung 2.3 gezeigt.

2.4 AES

Für die Pseudonymgenerierung wird ein sichererer Zufallszahlengenerator benötigt. In dieser Arbeit wird hierfür der Advanced Encryption Standard (AES) benutzt.

AES [DR02] ist eine Spezifikation zur Verschlüsselung digitaler Daten. Sie benutzt das symmetrische Rijndael-Verschlüsselungsverfahren, das bedeutet, dass sowohl Ver- als auch Entschlüsselung mit dem gleichen Schlüssel ablaufen. In AES können drei unterschiedliche Schlüssellängen benutzt werden, wobei ein längerer Schlüssel auch zu einer höheren Sicherheit führt. Die möglichen Schlüssellängen sind 128 Bit, 192 Bit und 256 Bit. Eine Ver- oder Entschlüsselung ist nur auf vollen sechzehn Byte langen Blöcken möglich. Aus diesem Grund wird dieses Verfahren auch als Blockverschlüsselungsverfahren bezeichnet.

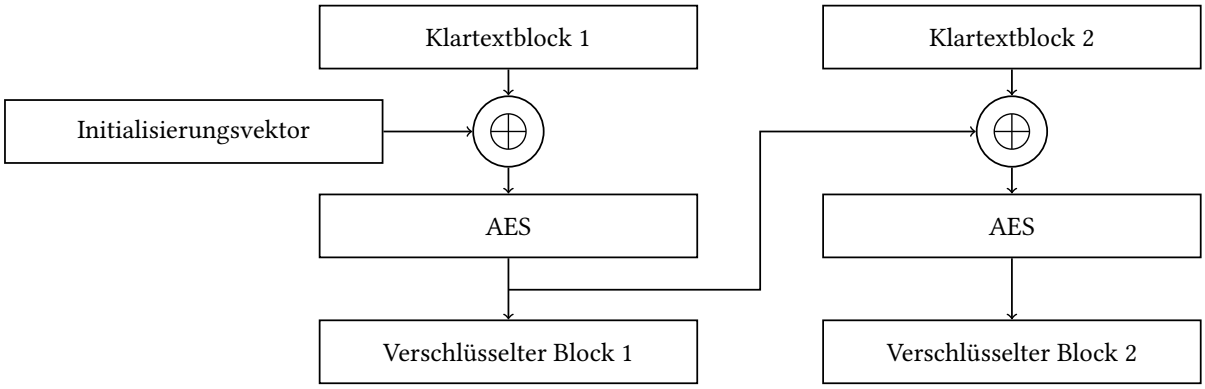


Abbildung 2.4: Der CBC-Betriebsmodus verschlüsselt mehrere Klartextblöcke, indem diese mit dem verschlüsselten Text der vorherigen Blocks mit dem logischen XOR verknüpft werden. Der erste dieser Blöcke ist der Initialisierungsvektor, der einer der Eingabeparameter dieses Verfahrens ist.

Der gerade beschriebene AES-Algorithmus ist nur in der Lage, volle Klartext-Blöcke zu verschlüsseln bzw. volle Blöcke an verschlüsselten Daten zu entschlüsseln. Es ist nicht möglich, mehr als sechzehn Byte Daten als Eingabe für den Algorithmus zu benutzen, weshalb es nicht möglich ist eine längere Sequenz an Daten zu ver- oder zu entschlüsseln. Aus diesem Grund gibt es in der Kryptographie die sogenannten *Betriebsmodi*, die ein Blockverschlüsselungsverfahren (wie zum Beispiel AES) und einen zusätzlichen Algorithmus benutzen um eine längere Sequenzen an Daten zu verschlüsseln. Es gibt mehrere dieser Modi, in dieser Arbeit wird der Cipher Block Chaining (CBC) Modus benutzt und im Folgenden genauer beschrieben. Der Verschlüsselungsvorgang dieses Verfahrens ist in Abbildung 2.4 bildlich dargestellt. Zum Start der Verschlüsselung mit CBC wird ein Initialisierungsvektor benutzt, dieser wird dafür benötigt, damit bei der wiederholten Verschlüsselung des gleichen Klartextes nicht der gleiche verschlüsselte Text ausgegeben wird. Der Initialisierungsvektor wird mit dem logischen XOR mit dem Klartext kombiniert und anschließend mit dem Blockverschlüsselungsverfahren verschlüsselt, die Ausgabe ist der verschlüsselte Text. Wenn der nächste Block verschlüsselt werden soll, so wird der letzte verschlüsselte Text als Initialisierungsvektor benutzt.

Bei der Entschlüsselung wird der verschlüsselte Text mit dem Blockverschlüsselungsverfahren entschlüsselt und anschließend per XOR mit dem gleichen Initialisierungsvektor wie bei der Verschlüsselung kombiniert. Dies ergibt dann wieder den zuvor verschlüsselten Klartext. Zur Entschlüsselung weiterer Blöcke wird wiederum der vorherige, verschlüsselte Block als Initialisierungsvektor benutzt, analog zum Verfahren bei der Verschlüsselung.

3 Verwandte Arbeiten

Zum Thema dieser Arbeit gibt es einige existierende wissenschaftliche Arbeiten, deren Inhalt verwandt mit dem Thema dieser Arbeit ist. Diese Arbeiten werden im Folgenden vorgestellt und kurz zusammengefasst.

3.1 Pseudonymisierung

Pseudonymisierung ist eine beliebte Methode zur Verschleierung der eigenen Identität. Hierbei wird statt der richtigen Identität ein erfundener Name benutzt. Ein externer Beobachter kann den erfundenen Namen nicht in Verbindung mit der realen Identität bringen, da diese Verknüpfung nicht öffentlich bekannt ist. Auch wenn ein Pseudonym verwendet wird, stellt dies nicht direkt sicher, dass die eigene Privatsphäre geschützt wird. Zwei Aktionen, die mit dem gleichen Pseudonym ausgeführt wurden, können direkt miteinander verknüpft werden, weshalb es nötig ist, zusätzliche Maßnahmen zu ergreifen, um die eigene Identität zu schützen. Eine beliebte Lösung für dieses Problem ist die Verwendung sich periodisch ändernder Pseudonyme, womit sichergestellt wird, dass nach einer bestimmten Zeit die Aktionen einer Person, mit unterschiedlichen Pseudonymen, nicht mehr verknüpft werden können. Dieses Konzept wird unter anderem in Vehicular ad hoc networks (VANets) benutzt, um die Identität der beobachteten Fahrzeuge zu verschleiern.

3.1.1 Pseudonyme in Vehicular Ad Hoc Networks

Ein Vehicular ad hoc network (VANet) ist ein Netzwerk, welches zwischen mehreren Fahrzeugen aufgebaut wird, um verkehrsrelevante Daten auszutauschen. Sie können z.B. zum Informationsaustausch über den umgebenden Verkehrsfluss genutzt werden, ohne einen zentralen Server zu benutzen. Dies reduziert die Latenz und eliminiert einen Single-Point-of-Failure, da das Netzwerk aus vielen einzelnen Fahrzeugen besteht. Die Daten in einem VANet können jedoch auch benutzt werden, um die Bewegungen eines Fahrzeugs nachzuverfolgen, da in einem Netzwerk die Teilnehmer eine eindeutige Identifikation benötigen, um miteinander kommunizieren zu können.

Um diesen Angriffsvektor zu eliminieren können Pseudonyme benutzt werden, wodurch es einem Angreifer erschwert wird, ein bestimmtes Fahrzeug zuverlässig nachzuverfolgen. In

[GG07] wird dieses Problem analysiert und es werden Lösungsvorschläge präsentiert, bei denen die Pseudonymänderung nur dann durchgeführt wird, wenn dadurch auch die Privatsphäre verbessert wird. Diese Entscheidung wird auf Grund der Umgebung getroffen. Wenn das Fahrzeug gerade in einer Fahrzeugmenge unterwegs ist, so ist es vorteilhaft das Pseudonym zu ändern, da die Verwechslung mit anderen Fahrzeugen hier am wahrscheinlichsten ist. Dies verringert die Wahrscheinlichkeit einer erfolgreichen Pseudonymänderung und minimiert die Generierung neuer Pseudonyme.

Buttyán, Holczer und Vajda [BHV07] haben gezeigt, wie ein Angreifer auch bei Benutzung von Pseudonymen einzelne Fahrzeuge identifizieren kann. Hierbei wurde ein Angreifer modelliert, welcher durch Abhören der Kommunikationen in einem VANet einen Pseudonymwechsel zu verfolgen. Das Modell der Untersuchung basierte auf dem „Mix-Zone"-Konzept, bei dem die Identität der Objekte vermischt wird, wenn sich diese innerhalb dieses Gebietes befinden. Der Angreifer benutzt mehrere Antennen, um die Kommunikation der Fahrzeuge abzuhören. Da der Angreifer nicht die gesamte Kommunikation in einem Netzwerk abhören kann, befinden sich die Fahrzeuge außerhalb der Reichweite des Angreifers in einer Mix-Zone. Durch dieses Modell wurden Simulationen mit mehreren Verkehrsdichten ausgeführt, die ergaben, dass die Wahrscheinlichkeit eines erfolgreichen Angriffs von der Anzahl der Antennen des Angreifers abhängt.

Da für die Schutzmechanismen der Beacons ein Mechanismus benötigt wird, der es einem Beobachter erschwert ein Beacon zu identifizieren, wird das Pseudonymkonzept benutzt, um die Identität eines Beacons zu verschleiern.

3.2 Bluetooth-Beacons zur Positionsbestimmung

Da übliche Positionierungssysteme wie GPS eine Sichtverbindung zu Satelliten in der Erdumlaufbahn benötigen, ist es nicht möglich, diese Systeme innerhalb von Gebäuden zu benutzten. Um dieses Problem zu lösen gibt es eine Reihe an wissenschaftlichen Arbeiten, die Bluetooth-Beacons benutzen, um innerhalb von Gebäuden die Position eines mobilen Geräts zu bestimmen. In [Cha08] wurde ein System vorgestellt, dass Informationen über die Ausbreitungseigenschaften der Bluetooth-Signale benutzt, um die Position des Empfängers zu berechnen. Hierfür wurde die Sichtbarkeit einzelner Beacons mit einem Graph modelliert. Mit dieser Information kann man die Position eines Geräts bestimmen, indem die sichtbaren Beacons mit dem Graph verglichen werden. Da die Sichtbarkeit eines Beacons ein räumliches Gebiet markiert, kann mit dem Graph die Schnittmenge der möglichen Positionen gebildet werden, wodurch eine mehr oder weniger genaue Position bestimmt werden kann.

3.3 Mix-Zones

Das sogenannte „Mix-Zone“-Modell [BS03] ist ein Konzept, bei dem bestimmte geographische Bereiche markiert werden, in denen die Identitäten aller in dieser Zone vorhandenen Personen vermischt wird. Dies wird üblicherweise mit Pseudonymen realisiert, die sich beim Betreten einer Mix-Zone ändern. Zusätzlich werden innerhalb einer Mix-Zone keine Positionsinformationen versendet, weshalb ein externer Beobachter nur eine limitierte Sicht auf die Bewegungen innerhalb einer Mix-Zone erhält, da er nur die Informationen über eingehende und ausgehende Benutzer erhält. Innerhalb der Mix-Zone sind keine Informationen über die Bewegungen vorhanden und müssten von einem Angreifer extrapoliert werden. Hierdurch soll es unmöglich oder zumindest sehr schwer gemacht werden, ein einzelnes mobiles Gerät zuverlässig zu verfolgen. Um dieses Konzept zu implementieren wird eine Middleware benötigt, die zwischen dem Nutzer und den angebotenen Diensten vermittelt. Die Nutzer melden ihre Position an diese Middleware, welche dann mit diesen Informationen das Mix-Zone Modell implementieren kann.

Dieses Konzept wird in dieser Arbeit benutzt um einen Pseudonymwechsel eines Beacons zu verschleiern, um es Angreifern zu erschweren auch nach einem Pseudonymwechsel ein Beacon zu verfolgen.

4 Systemmodell und Problemstellung

Im Folgenden wird das Systemmodell vorgestellt und unter welchen Annahmen dieses existiert. Es wird ein Angreifermodell eingeführt und analysiert, welche Angriffe auf das System möglich sind.

4.1 Systemmodell

Das System enthält vier Arten von Komponenten. Das **Beacon**, die **Beobachter**, den **Server** und den **Klienten**. Zusätzlich gibt es den *Besitzer* eines Beacons, jedoch interagiert dieser nicht direkt mit dem System. Im Folgenden wird mit *Besitzer* die Person bezeichnet, der ein Beacon gehört, der Klient ist eine Software, mit der der Besitzer Informationen über sein Beacon abfragen kann. In Abbildung 4.1 ist das Systemmodell beispielhaft dargestellt.

Das Beacon ist ein BLE-Gerät, welches Broadcast-Signale versendet. Dies kann zum Beispiel eine spezialisiertes Gerät sein, dass auf das Versenden der Beaconsignale ausgelegt ist, oder es kann ein Smartphone, Laptop oder Desktop-Computer sein, der so konfiguriert wurde, dass die Beaconsignale versendet werden.

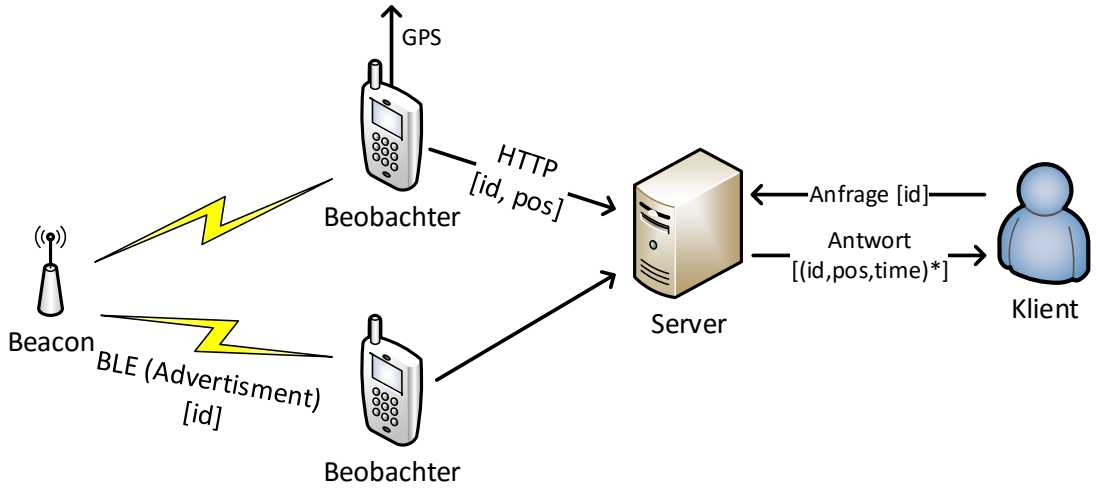


Abbildung 4.1: Das Beacon sendet per BLE seine Kennung aus. Die Beobachter empfangen dieses Signal und schicken es an den Server. Der Besitzer des Beacons kann von diesem Server die Position des Beacons abfragen.

Der Beobachter empfängt die Signale der Beacons in der Umgebung und sendet sie weiter an den Server. Typischerweise ist der Beobachter ein Smartphone, da diese mobil sind und per GPS ihre eigene Position leicht bestimmen können, allerdings ist es auch möglich, einen Desktop-Rechner mit der richtigen BLE-Hardware als Beobachter zu betreiben. Hier wird jedoch ein mobiles Sensornetz aus Smartphones angenommen.

Der Server stellt den Beobachtern eine HTTP/HTTPS-Schnittstelle zur Verfügung, über die die Daten der Beacons übertragen werden. Der Beobachter schickt die gefundenen Kennungen der Beacons und die eigene Position an den Server. Der Server stellt diese Daten über eine weiter HTTP/HTTPS-Schnittstelle den Klienten zur Verfügung.

Der Klient ist eine Software, die die HTTP/HTTPS Schnittstelle des Server benutzt, um die Bewegungen eines Beacons zu verfolgen. Diese Software wird vom Besitzer des Beacons betrieben und erhält von diesem alle nötigen Informationen über das Beacon, um dieses zu verfolgen.

Da sowohl Beobachter als auch Server nicht vom Besitzer gesteuert werden können, wird angenommen, dass diese nicht vertrauenswürdig sind. Aus diesem Grund wird im Folgenden ein Angreifermodell entwickelt, welches das Verhalten eines Angreifers und die möglichen Angriffe auf das System beschreiben.

4.2 Angriffe

Da das System möglichst sicher arbeiten soll, müssen mögliche Angreifer und deren Angriffe analysiert werden. In diesem Kapitel werden hierfür Modelle erstellt, die in der Analyse benutzt werden.

4.2.1 Angreifermodell

Um die Sicherheit des Designs zu untersuchen muss das mögliche Verhalten eines Angreifers modelliert werden. Hierbei wird zwischen zwei unterschiedlichen Angreiferarten unterschieden, da es mehrere mögliche Angriffspunkte gibt:

Lokal Ein Angreifer versucht die Daten eines Beacons in dessen näheren Umgebung zu verarbeiten. Zusätzlich gibt es hierbei zwei Angriffsvariationen, den *passiven* und den *aktiven* Angreifer. Passive Angreifer können die Nachrichten der Beacons mitlesen, speichern und verarbeiten, jedoch befolgen diese das entwickelte Protokoll und senden die Daten an den Server. Ein aktiver Angreifer erweitert diese Möglichkeiten mit der Fähigkeit das Protokoll vollkommen zu missachten. Dieser Angreifer ist in der Lage, die empfangenen Daten zu verändern oder Datensätze gar nicht an den Server zu schicken.

Server Ein Angreifer benutzt die Daten des Servers um ein Beacon nachzuverfolgen. So ein Angreifer könnte zum Beispiel auch der Betreiber des Servers sein, da dieser Zugriff auf alle Daten hat.

4.2.2 Mögliche Angriffe

Basierend auf dem beschriebenen Angreifermodell werden im Folgenden mögliche Angriffsvektoren gezeigt. Diese werden in Abschnitt 5.3 analysiert.

Lokale Angriffe

Bei lokalen Angriffen ist der Angreifer in unmittelbarer Nähe zum Beacon, da die Reichweite von BLE sehr begrenzt ist (ca. zehn Meter [GOP12]). Lokale Angriffe haben deshalb keine globale Sicht auf das Bewegungsprofil eines Beacons, jedoch können sie direkt per BLE mit dem Beacon kommunizieren. Die einzelnen möglichen Szenarien werden im Folgenden vorgestellt.

Preisgabe der Identität des Beacons Da BLE-Übertragungen zusätzliche Daten, wie zum Beispiel die MAC-Adresse, zu den Beacondaten enthalten kann, wäre es möglich, dass ein passiver Angreifer mit diesen Daten die Identität des Beacons feststellen könnte, ohne die Beacon-Identifikation zu benötigen.

Kompromittierung eines Beacons Sollte die Implementierung des Beacons eine Sicherheitslücke enthalten, könnte ein externer aktiver Angreifer per BLE das Beacon kompromittieren und dazu bringen, gegebenenfalls geheime Daten preiszugeben. Die Sicherheit der Geräte ist nicht das zentrale Thema dieser Arbeit, jedoch wurde dieses Problem beachtet, um mögliche zukünftige Erweiterungen des Entwurfs sicherer zu machen.

Angreifer kopiert Identifikation Da die Kennung eines Beacons im Klartext übertragen wird, könnte ein aktiver Angreifer diese kopieren und an einer anderen Stelle erneut benutzen.

Server Angriffe

Im Gegensatz zu den lokalen Angriffen haben Server Angriffe keinen direkten Zugriff auf die Beacons, sondern können nur auf die Daten zugreifen, die durch die Beobachter an den Server geschickt wurden. Ein Angreifer könnte an diese Daten gelangen, wenn dieser zum Beispiel auf die Daten des Servers zugreifen kann oder wenn der Serverbetreiber selbst die Daten benutzt, um Beacons zu verfolgen.

Das Hauptproblem bei einem zentralen Server ist die globale Sicht auf alle Beacons. Mit diesen Informationen könnte dieser Bewegungsprofile der Beacons anfertigen. Dies muss so weit wie möglich verhindert werden, um die Privatsphäre des Beaconbesitzers zu schützen.

4.3 Problemstellung

Da sowohl Beobachter als auch Server nicht als vertrauenswürdig angenommen werden können, ergibt sich das Problem, dass bei einer statischen Kennung eine Nachverfolgung trivial möglich ist. Aus diesem Grund müssen im Entwurf Schutzmechanismen vorhanden sein, die dies erschweren oder verhindern. Gleichzeitig sollte es allerdings dem Besitzer möglich sein, ein Beacon kontinuierlich zu verfolgen, ohne dadurch Angreifern zusätzliche Informationen über die Beacon-Identität preiszugeben.

5 Entwurf der Schutzmechanismen

Das in Kapitel 4 vorgestellte Systemmodell beschreibt die Umgebung, in der dieser Entwurf existiert. Auf dieser Grundlage kann nun das Verhalten der Beacons entworfen und gezeigt werden, wie die Schutzmechanismen funktionieren und wie effektiv diese sind.

5.1 Übersicht

Um Sicherheitslücken in der Software des Beacons zu vermeiden wird dieses ausschließlich verbindungslos betrieben. Das bedeutet, dass das Beacon keine Verbindung mit einem anderen Gerät aufnimmt, da dies ein mögliches Sicherheitsrisiko darstellen könnte, welches von einem Angreifer ausgenutzt werden kann. Durch eine vollkommen verbindungslose Implementierung kann ein Angreifer das Beacon zudem nicht von außen beeinflussen, wodurch es nicht möglich ist, z.B. Zugriff auf den internen Speicher des Beacons zu erlangen.

Durch die Benutzung verbindungsloser Beacons muss das Beacon ohne externe Einflüsse operieren können. Das Beacon kann Aktionen zeitbasiert ausführen, jedoch kann es nicht darauf reagieren, wenn ein Empfänger das Beacon gesehen hat, da keine Verbindung mit dem Beacon hergestellt wurde. Aus diesem Grund wird eine Methode benutzt, die bereits in VANets erforscht und benutzt wird: Pseudonyme. Anstatt einer festen Kennung sendet das Beacon eine sich ändernde Kennung aus, die für einen Angreifer wie zufällige Daten aussehen soll. Diese Kennung wird periodisch nach einer bestimmten Zeit geändert, um eine Nachverfolgung zu verhindern.

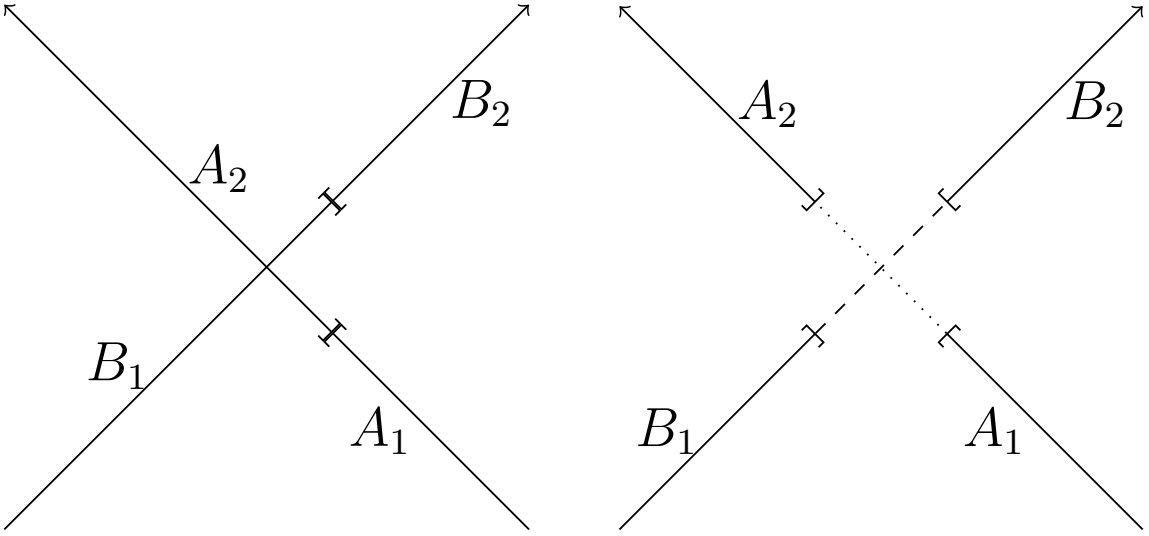
Zusätzlich soll es einem Angreifer erschwert werden, einen Pseudonymwechsel nachzuverfolgen, um zu verhindern, dass ein Beacon auch nach einem Pseudonymwechsel eindeutig identifizierbar bleibt. In Abbildung 5.1a wird gezeigt wie ein globaler Beobachter ein Pseudonymwechsel des Beacons A von A_1 auf A_2 nachvollziehen könnte, da die Änderung des Pseudonyms sofort geschieht. Für einen Beobachter sähe dies so aus, als ob ein Beacon verschwindet und sofort ein Neues auftauchen würde. Hierdurch könnte der Beobachter mit hoher Wahrscheinlichkeit schlussfolgern, dass diese beiden Kennung zum gleichen Beacon gehören. Um dies zu verhindern wird eine Adaption des „Mix-Zone“ Konzepts benutzt. Typischerweise ist diese Zone auf ein geografisches Gebiet limitiert, jedoch ist dies hier nicht möglich, da das Beacon seine Position nicht feststellen kann. Aus diesem Grund wird hierbei keine positionsbasierte, sondern eine zeitbasierte Mix-Zone verwendet. Diese zeitliche Mix-Zone wird umgesetzt, indem der BLE Sender vorübergehend ausgeschaltet wird, bevor ein neues

Pseudonym generiert wird. Dieses Konzept wird im Folgenden als „Funkstille“ bezeichnet. In Abbildung 5.1b ist zu sehen wie sich dies auf die Nachverfolgbarkeit auswirkt. Der globale Beobachter kann jetzt nicht mehr mit hoher Wahrscheinlichkeit erkennen, ob das Pseudonym A_2 zu Beacon A oder zu Beacon B gehört, wodurch die Nachverfolgung erschwert wird. Der globale Beobachter hat nach diesem Wechsel eine Wahrscheinlichkeit von $\frac{1}{2}$ dem richtigen Beacon das entsprechende Pseudonym zuzuordnen. Je mehr Beacons in der Nähe sind desto öfter kann das Beacon verwechselt werden wodurch die Wahrscheinlichkeit für eine sichere Nachverfolgung weiter sinkt. In der realen Welt ist diese Wahrscheinlichkeit jedoch höchstwahrscheinlich geringer, da durch Extrapolation der Beacon-Bewegung die Genauigkeit der Schätzung verbessert werden kann.

Algorithmus 5.1 zeigt den Algorithmus, der von einem Beacon benutzt wird, um diese Konzepte zu implementieren. Zu Beginn einer Periode wird ein neues Pseudonym generiert, welches von der aktuellen Zeit abhängt. Dieses Pseudonym wird anschließend als Kennung für die Beacon-Broadcast-Daten benutzt und, wenn dies durch die BLE-Hardware unterstützt wird, könnte zusätzlich die BLE-MAC-Adresse des Beacons geändert werden. Da Pseudonyme möglichst zufällig sein sollen, wäre ein generiertes Pseudonym eine gute Quelle von zufälligen Daten, die zur Generierung einer möglichst zufälligen MAC-Adresse genutzt werden kann. Nach der Initialisierung der Daten wird das Funkmodul aktiviert und das Beacon beginnt mit der Aussendung der Daten. Nach einer bestimmten Zeit wird das Funkmodul ausgeschaltet, um die Funkstille zu beginnen. Wenn die Funkstille zu Ende ist, wird der Zeitzähler des Beacons inkrementiert und der Algorithmus wird erneut ausgeführt. Wenn ein Beacon die Funkstille beginnt sieht es für einen Beobachter so aus, als wäre dieses Beacon verschwunden. Wenn das gleiche Beacon die nächste Periode beginnt, wird ein neues Pseudonym benutzen und der Beobachter entdeckt ein „neues“ Beacon. Für den Beobachter sollte es nicht möglich sein, nur aus diesen Daten einen Zusammenhang zwischen dem alten und dem neuen Pseudonym herzustellen. Dieses Verhalten ist allerdings nur dann effektiv, wenn mehrere Beacons in der Umgebung ihre ID ändern, da sonst ein Beobachter leicht erkennen kann, dass ein Beacon nur in der Funkstille war, wenn ein Beacon verschwindet und kurz darauf ein Neues erscheint. Das gleiche Problem tritt auch bei VANets auf [GG07] und ist eine Schwachstelle dieser Funktionalität. In Abschnitt 7.2 auf Seite 59 wird empirisch evaluiert, ob dies ein generelles Problem darstellen könnte.

5.2 Beacon

Da das Pseudonym periodisch geändert werden soll, muss das Beacon in der Lage sein, nach einer bestimmten Zeit eine Aktion auszuführen. Dies ist generell möglich, da die meisten Mikroprozessoren eine interne Uhr besitzen und damit die Möglichkeit bieten, periodisch bestimmte Funktionen auszuführen und damit z. B. das Pseudonym zu ändern. Allerdings sind die benutzten Uhren aus diversen Gründen nicht sehr genau, da sie mit wenig Hardwareressourcen auskommen müssen und gleichzeitig nur auf kurze Warteperioden ausgelegt sind. Es



(a) Ohne Funkstille **(b)** Mit Funkstille

Abbildung 5.1: Die Funkstille versucht die Identität eines Beacon zu verschleiern, indem der BLE-Sender vor der Pseudonymänderung ausgeschaltet wird. Ohne diese Vorkehrung wäre der Pseudonymwechsel deutlich weniger effektiv, da die Änderung sofort geschieht und eine Vermischung mit umgebenden Beacons deutlich unwahrscheinlicher wäre.

Algorithmus 5.1 Beacon Perioden Algorithmus

```
procedure DoPeriod(t)
    p ← GeneratePseudonym(t)
    SetBLEData(p)
    UpdateMACAddress(p)
    EnableRadio
    WaitFor(active_time)
    DisableRadio
    WaitFor(silent_time)
end procedure
```

ist auch nicht möglich, die Uhr nachträglich zu synchronisieren, da das Beacon nicht mit der Außenwelt interagieren soll, um die Angriffsfläche zu minimieren. Aus diesem Grund muss der Klient, der die Position des Beacons später herausfinden soll, in der Lage sein, den Clock-Drift auszugleichen, um sicherzustellen, dass eine unregelmäßig laufende Uhr nicht dazu führt, dass das Beacon nicht mehr gefunden werden kann.

Für die Pseudonymgenerierung muss ein Algorithmus benutzt werden, der unter Benutzung der Zeit des Beacons und einem internen Status, Zahlen ausgeben kann, die für einen externen Beobachter wie zufällige Zahlen aussehen. Gleichzeitig sollte es jedoch auch möglich sein, die generierten Daten erneut zu erzeugen, wenn die Parameter für den Algorithmus bekannt sind, damit der Besitzer die benutzten Pseudonyme nachverfolgen kann. Hierfür könnten Pseudozufallsgeneratoren [MVV96] wie zum Beispiel der Mersenne Twister [MN98] benutzt werden. Diese Algorithmen generieren basierend auf Eingabeparametern Zahlen, die wie Zufallszahlen aussehen, jedoch sind die meisten dieser Algorithmen nicht darauf ausgelegt, kryptografisch sicher zu sein. Das bedeutet, dass ein Angreifer, der genügend viele Zufallszahlen dieses Algorithmus gesehen hat, die Eingabeparameter des Algorithmus erraten oder berechnen kann. Aus diesem Grund gibt es eine weitere Klasse der Pseudozufallsgeneratoren, welche kryptografisch sicher sind [MVV96] und bei denen es nicht möglich oder zumindest sehr unwahrscheinlich ist, dass die Eingabeparameter erraten werden können. Diese Art der Algorithmen sind ideal für die Pseudonymgenerierung, da der Besitzer die Pseudonyme nachträglich generieren kann, jedoch ist es für einen Angreifer umgekehrt nicht möglich, die Pseudonyme zu benutzen, um die Eingabeparameter zu erraten. Ein solcher Algorithmus ist der Verschlüsselungsalgorithmus AES. Dieser ist für seine kryptografische Sicherheit bekannt [HW03] und kann als Pseudozufallsgenerator benutzt werden, indem die verschlüsselte Ausgabe zu Zahlen konvertiert wird. Der Besitzer kann die generierten Pseudonyme nachträglich generieren, da er den benutzten Schlüssel und den Initialisierungsvektor kennt, mit dem die Pseuodnyme generiert wurden.

5.3 Analyse möglicher Angriffe

In Abschnitt 4.2.2 auf Seite 31 wurden mögliche Angriffe aufgezählt. Diese können nun unter Benutzung des vorgestellten Designs analysiert werden. Wie in Abschnitt 4.2.2 wird die Diskussion in Lokale und Server Angriffe aufgeteilt.

5.3.1 Lokale Angriffe

Bei den lokalen Angriffen wird der Angreifer versuchen, in unmittelbarer Nähe zum Beacon eine Sicherheitslücke auszunutzen.

Nachverfolgung der Pseudonymänderung

Ein Angreifer könnte die empfangenen Daten benutzen, um ein Beacon über mehrere Pseudonymwechsel hinweg zu verfolgen, wenn der zur Pseudonymgenerierung verwendete Algorithmus nicht kryptografisch sicher ist, wodurch ein Angreifer durch Mitlesen mehrerer Pseudonyme das nächste Pseudonym berechnen oder erraten könnte. Aus diesem Grund muss die Implementierung einen kryptografisch sicheren Algorithmus, wie zum Beispiel AES, benutzten. Zusätzlich wird mit der Funkstille bei der Änderung des Pseudonyms eine Art Mix-Zone geschaffen, durch welche die Verfolgung der Pseudonymänderung erschweren werden soll.

Preisgabe der Identität des Beacons

Auch mit Pseudonymen könnten per BLE zusätzliche Daten übertragen werden, die das Beacon eindeutig identifizierbar machen. Die MAC Adresse des BLE-Moduls ist zum Beispiel global eindeutig und könnte dazu genutzt werden, das Beacon zu identifizieren falls diese nicht geändert wird. Wie bereits in Algorithmus 5.1 gezeigt wurde, kann dies implementiert werden, indem das zufällige Pseudonym zur Erzeugung einer neuen MAC-Adresse genutzt wird. Alternativ sieht BLE einen Modus vor, in dem die MAC-Adresse periodisch und zufällig geändert wird. Wenn dieser Modus verwendet wird, muss jedoch sichergestellt werden, dass diese Änderung mindestens ein Mal innerhalb der Funkstille geschieht, damit sichergestellt wird, dass vor und nach der Pseudonymänderung eine andere MAC-Adresse verwendet wird.

In Kapitel 6 wird für die einzelnen Beacon-Implementierungen gezeigt, ob und wie diese Änderung der MAC-Adresse realisiert wird.

Kompromittierung eines Beacons

Da das Beacon vollkommen verbindungslos arbeitet, ist es nicht möglich, das Beacon dazu zu bringen auf externe Einflüsse zu reagieren. Alle BLE Daten werden ausschließlich als Broadcast versendet und das Beacon reagiert nicht auf Verbindungsversuche. Sollte der Angreifer jedoch direkten physikalischen Zugriff auf das Beacon erlangen wäre es möglich, dass hierdurch die Daten des Beacons ausgelesen werden oder dass die Beacon-Software verändert wird. Dieser Art von Angriffen ist generell sehr schwierig vorzubeugen, sie wird deshalb hier nicht weiter betrachtet werden.

Angreifer kopiert Identifikation

Die Klartextübertragung der Kennung macht es technisch unmöglich, ein Kopieren der Identifikation zu verhindern. Aus diesem Grund müssen Server und Klient damit umgehen können,

dass Kennungen doppelt benutzt werden könnten. Dieser Fall könnte auch ohne einen Angreifer auftreten, falls es zu einer Kollision der Pseudonyme zweier Beacons kommt. Um zu erkennen, dass eine Kennung kopiert wurde könnten aus der Kryptografie übliche Verfahren benutzt werden, allerdings basieren diese meist auf einer Zwei-Wege-Kommunikation, die hier durch das verbindungslose Verhalten des Beacons nicht möglich ist. Aus diesem Grund wird dieser Ansatz in dieser Arbeit nicht weiter verfolgt.

5.3.2 Server Angriffe

Angriffe auf die Privatsphäre sind auch auf dem Server möglich. Hierbei sind nur die ID und die ungefähren Positionen der Beacons verfügbar, jedoch hat der Server eine globale Sicht auf die Position aller Beacons. Hierdurch ergeben sich andere Angriffsvektoren als bei lokalen Angriffen.

Angriff des Pseudonymalgorithmus

Da der Server Zugriff auf alle gefundenen Beacon-Kennungen hat ist er in der Lage, deutlich mehr Pseudonyme zu benutzen, um den Algorithmus anzugreifen, der für die Pseudonymgenerierung verwendet wird. Da kryptografisch sichere Algorithmen benutzt werden wird jedoch angenommen, dass der Algorithmus an sich in diesem Fall nicht angreifbar ist.

Zu wenig Beacons in einem Gebiet

Da die Funkstille darauf aufbaut, dass durch das Ausschalten des BLE-Senders die Identität eines Beacons mit anderen vermischt wird müssen natürlich genügend Beacons in einem Gebiet vorhanden sein. Ein ähnliches Problem tritt auch in VANets auf [GG07], wenn zu wenig Fahrzeuge in einem Gebiet unterwegs sind. In Abbildung 5.2 ist dieses Problem dargestellt, bei dem die Änderungen der Pseudonyme keine Verbesserung der Privatsphäre erreichen, da zu wenig anderen Beacons in der Nähe sind. Dieses Problem wird in Abschnitt 7.2 auf Seite 59 empirisch evaluiert.

Extrapolation der Beacon-Bewegung

Durch die globale Sicht könnte es auch möglich werden, durch Analyse des Bewegungsverhaltens der Beacons einen Zusammenhang zwischen einzelnen Pseudonymen herzustellen. Da die Bewegung eines Beacons meist sehr regelmäßig verläuft, da es z.B. einer Person folgt, die das Beacon bei sich trägt, könnten die weitere Bewegung extrapoliert werden und anschließend auch nach einem Pseudonymwechsel nachverfolgt werden. In Abbildung 5.3 ist dieses Problem visualisiert. Beacons A und B wären normalerweise verwechselt worden, da der globale Beobachter nicht mit Sicherheit sagen kann ob A_2 oder B_2 zu Beacon A gehört.

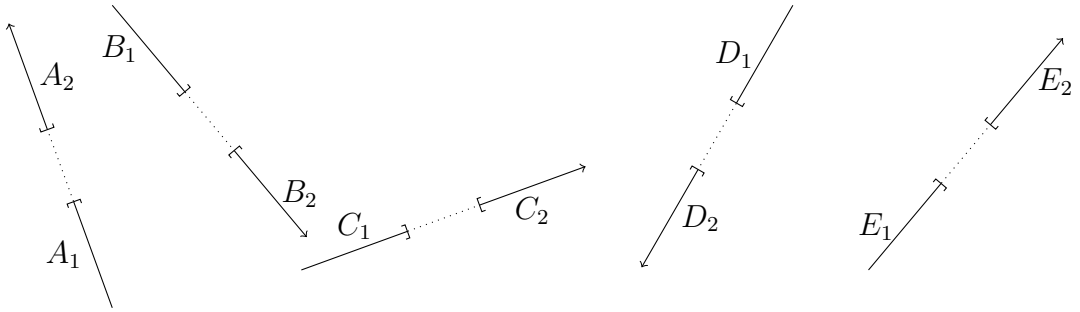


Abbildung 5.2: Sollten in einem Gebiet zu wenig Beacons unterwegs sein, so wird es für den Server sehr leicht, auch mit Pseudonymen und Funkstille ein Beacon nachzuverfolgen.

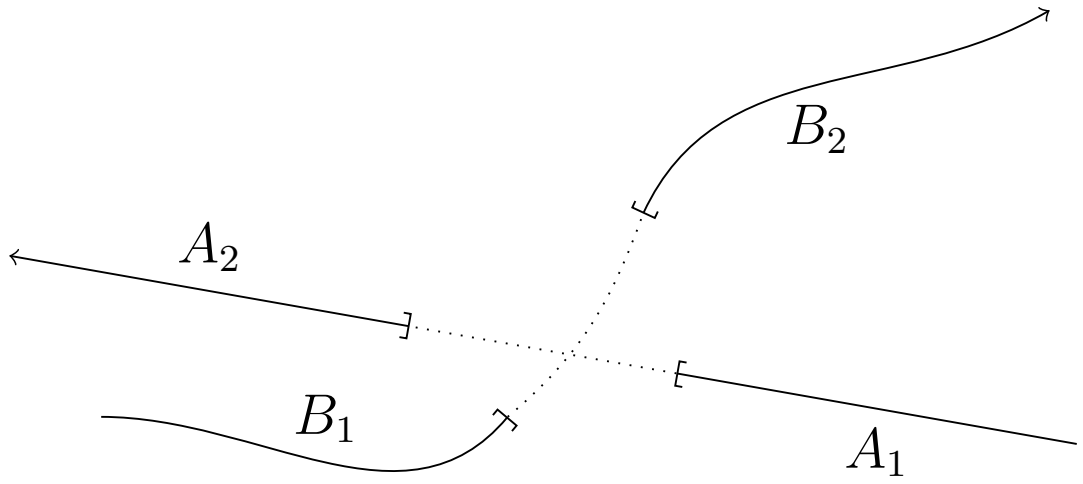


Abbildung 5.3: Wenn der Server durch Extrapolation die Bewegung eines Beacons vorhersagen kann, ist eine Nachverfolgung eines Beacons auch über ein Pseudonymwechsel hinweg möglich.

Jedoch bewegen sich A auf einer geraden Linie und entgegen von B weshalb der Beobachter mit hoher Wahrscheinlichkeit feststellen kann, dass A_2 zu A und B_2 zu B gehört.

Dieses Problem ist eins der größten Risiken für die Privatsphäre in diesem Entwurf, allerdings ist die Analyse und Behebung dieses Angriffsvektors eine sehr anspruchsvolle Problemstellung, die im Rahmen dieser Arbeit nicht weiter analysiert werden soll.

6 Implementierung

Das vorgestellte Beacon-Design wurde auf mehreren Plattformen implementiert, um die Portabilität des Algorithmus zu demonstrieren. Zusätzlich wurden noch zusätzliche Komponenten wie die Android-App, der Server sowie ein Web-Klient implementiert, die benötigt werden, um das Design in einem vollkommenen System umzusetzen.

6.1 Beacon

Für die Beacons wird der bereits existierende Eddystone Standard (siehe Abschnitt 2.2.2 auf Seite 19) von Google benutzt, da dieser bereits stark verbreitet ist und gut durch diverse Werkzeuge und Bibliotheken unterstützt wird. Der Standard unterstützt bereits einen Modus, in dem Beacons Pseudonyme anstelle von UIDs benutzen (siehe Abschnitt 2.2.2 auf Seite 19), allerdings benutzt dieser Modus nur acht anstelle der sechzehn Bytes in einem „UID“ Frame, wodurch die möglichen ID Werte unnötigerweise eingeschränkt werden. Aus diesem Grund wird hier der UID-Frame benutzt, allerdings wird die Kennung nicht in Namespace und Instance aufgeteilt, sondern die gesamten 16 Byte werden für das Pseudonym verwendet.

In Abschnitt 5.2 wurden bereits die grundlegenden Konzepte des Beacons vorgestellt. In diesem Kapitel wird nun auf die speziellen Eigenschaften der Implementierung eingegangen. Für die Pseudonymgenerierung wird AES mit einem 128-Bit langen Schlüssel im CBC-Modus verwendet. AES ist kryptografisch sicher [HW03] und sehr effizient implementierbar, wodurch der Algorithmus auch für die Verwendung in eingebetteten Systemen geeignet ist. Der CBC-Modus wird verwendet, da mehrere Blöcke hintereinander verschlüsselt werden müssen.

Das Beacon wurde auf der bereits existierenden BLE Beacon Platform Faros (siehe Abschnitt 2.3.1 auf Seite 21), auf der nRF51 Plattform (siehe Abschnitt 2.3.2 auf Seite 21) und als Android App implementiert. Für die Entwicklung des nRF51 Beacons wurde das nRF51 Developer Kit benutzt.

6.1.1 Faros Beacon

Das Faros Beacon ist eine existierende Eddystone Beacon-Plattform für den Arduino. Diese unterstützt bereits das Aussenden von Eddystone-Frames per BLE, jedoch ist die UID über die gesamt Laufzeit fest und kann nicht geändert werden. Dieser Code wurde abgeändert,

so dass periodisch die UID des Beacons geändert wird. Das Änderungsintervall kann zur Übersetzungszeit mit einer Konstante spezifiziert werden. Die existierende Plattform konnte bereits URL und TLM Frames versenden, jedoch werden diese Daten in diesem Anwendungsfall nicht gebraucht weshalb das Versenden dieser Daten abgeschaltet wurde.

Um Energie zu sparen benutzt das Faros Board den „Power-Down"-Modus des Arduinos, in dem so wenig Energie wie möglich verbraucht wird. Als Zeitgeber wird ein Interrupt benutzt, der jeder Sekunde ausgeführt wird. Sobald dieser ausgelöst wurde, wird überprüft ob die Funkstille begonnen werden soll oder ob ein neues Pseudonym erzeugt werden muss. Gleichzeitig wird ein Zeitzähler inkrementiert, der benutzt wird, um die Länge der Änderungsintervalls zu kontrollieren. Durch diese Vorkehrungen ist die CPU des Faros Boards für einen Großteil der Laufzeit inaktiv und benötigt dadurch sehr wenig Energie.

Mithilfe einer auf den Arduino spezialisierten AES-Implementierung [Lan] werden die Pseudonyme generiert. Der auf dem Faros-Beacon verwendete BLE Sender kann seine MAC-Adresse nicht auf Anfrage des Arduinos ändern, wodurch diese Plattform sich nicht für den tatsächlichen Betrieb eignet, da die MAC-Adresse des BLE-Moduls das Beacon eindeutig identifiziert, auch wenn die Eddystone UID geändert wird. Trotz dieser Einschränkung kann diese Plattform benutzt werden, um die Energieeffizienz der Entwurfs zu bewerten. Die Eingabedaten (Schlüssel und Initialisierungsvektor) von AES werden in dieser Implementierung fest in den Programmcode kodiert.

6.1.2 nRF51 Beacon

Die Implementierung auf nRF51 Basis benutzt die Zeitgeber, die mit dem Software Development Kit (SDK) des Entwicklungskits mitgeliefert wurden. Diese erlauben es, nach einer bestimmten Zeit eine Funktion aufzurufen. Intern werden dieser Zeitgeber über Interrupts in der CPU implementiert. Dies hat den Vorteil, dass während der Zeitgeber-Laufzeit die CPU in einen Energiesparmodus geschickt werden kann. Sobald ein Zeitgeber-Interrupt auftritt, wird die CPU reaktiviert und die Funktion wird ausgeführt. Durch diesen Energiesparmodus kann Energie gespart werden, da die CPU nur für aktive Aufgaben, nicht aber für den Betrieb der Zeitgeber aktiv sein muss und einen Großteil der Laufzeit im Energiesparmodus verbringt.

Um die Pseudonymgenerierung umzusetzen wurde eine Implementierung des AES-Algorithmus in reinem C benutzt [kok]. Diese ist nicht so stark optimiert wie die Assemblerimplementierung, welche mit dem Faros Board benutzt wurde, jedoch konnte die Bibliothek leicht in die Beacon-Implementierung integriert werden.

Der AES-Key und Initialisierungsvektor wurde wie beim Faros Beacon fest in den Quelltext kodiert. Für eine praktische Anwendung müssten diese Daten durch den Nutzer veränderbar sein. Diese Funktionalität könnte zum Beispiel durch einen Mechanismus implementiert werden, der es ermöglicht, eine BLE-Verbindung zum Beacon aufzubauen und über diese Verbindung die Daten zu setzen. Allerdings darf diese Verbindung nicht immer möglich sein. Zum einen verbraucht dies zusätzliche Energie und zum anderen wäre es eine ideale Einbruchsstelle für

einen Angreifer. Zur Sicherung dieser Schnittstelle könnte sie nur aktiviert werden, wenn der Benutzer, der üblicherweise physischen Zugang zum Gerät hat, z.B. einen Taster auf dem Beacon drückt. Danach wird das Beacon für eine kurze Zeit eine Verbindung bereitstellen, und nachdem die Verbindung beendet wurde, würde das Beacon in den normalen Aktiv-Inaktiv-Modus wechseln und als normales Beacon agieren, ohne auf eine eingehende Verbindung zu warten.

Das BLE Modul des nRF51 unterstützt die Änderung der MAC-Adresse zur Laufzeit. Dies wird benutzt, um die MAC-Adresse zur gleichen Zeit wie die Kennung zu ändern, um zu verhindern, dass ein lokaler Angreifer über die Adresse die Identität den Beacons feststellen kann. Für die MAC-Adresse werden die ersten 6 Bytes des Pseudonyms verwendet, wodurch die MAC-Adresse ausreichend zufällig wird und gleichzeitig energiesparend bereitgestellt werden kann, da das Pseudonym bereits berechnet wurde. Das Setzen der Adresse wird mit der `sd_ble_gap_address_set` Funktion bewerkstelligt, die das SDK des Softdevices bereitgestellt. Dieser Funktion wird eine Datenstruktur übergeben, die die neue Adresse enthält. Alle notwendigen Operationen, um die Adresse zu ändern, werden anschließend von der Softdevice selber übernommen.

6.1.3 Android Beacon

Die zwei bisher vorgestellten Beacon-Implementierungen benötigen beide spezielle Hardware, um das Beacon bereitzustellen, jedoch wäre auch eine Implementierung und Ausführung auf üblichen Smartphones sehr nützlich. Aus diesem Grund wurde eine Android-App entwickelt, welche das Beacon-Protokoll implementiert. Die Implementierung benutzt die von Android bereitgestellten Zeitgeber-Funktionen, um periodisch zwischen Broadcast-Modus und Funkstille zu wechseln. Da ein Smartphone üblicherweise eine synchronisierte Uhr hat, wird die gemeldete Uhrzeit benutzt, um die Länge des Broadcast-Modus und der Funkstille anzupassen, um damit eventuelle Ungenauigkeiten auszugleichen, die durch die Zeitgeberimplementierung in Android entstehen könnten.

Um die Pseudonyme zu generieren wird die AES-Implementierung im CBC Modus benutzt, die bereits in Java, beziehungsweise Android vorhanden ist. Das Versenden der BLE-Broadcasts wird mit der *Android Beacon Library* [alt] realisiert. Diese ermöglicht den Versand der Beacon-Daten in einem durch den Programmierer festgelegten Binärformat per BLE. Hier wurde ein abgewandeltes Eddystone-Format benutzt, bei dem die UID als zusammenhängender 16-Byte Block behandelt wird, anstatt sie in Namespace und Instance aufzuspalten. Die Android API stellt allerdings keine Funktionalität bereit, um die BLE-MAC-Adresse zu ändern. Die MAC-Adresse wird jedoch periodisch geändert, allerdings könnte dies zu Problemen führen, da die MAC-Adresse gleich bleiben könnte, während sich das Pseudonym ändert. Allerdings wären diese Informationen nur für einen lokalen Angreifer nützlich, da der Server normalerweise nicht die MAC-Adresse des Beacons erfährt.

Die nötigen Funktionen um als Beacon zu agieren stehen erst seit der Version 5.0 von Android bereit, weshalb diese Version die minimale Version ist, die von dieser App benötigt wird. Die vorherige Version 4.3 unterstützte die Central-Rolle, allerdings nicht die Peripheral-Rolle, die für Beacons benötigt wird.

6.2 Android Scanner Service

Damit die Beacons auch nachverfolgt werden können, werden Geräte benötigt, die die Kennungen empfangen und an den Server weiterleiten. Dies wurde durch die gleiche Android-App realisiert, die das Smartphone zu einem Beacon macht. Dies ist möglich, da während der Funkstille das BLE Modul nicht benutzt wird und somit für andere Aufgaben genutzt werden kann. Während dieser Zeit fungiert die App als Beobachter, die nach Beacons in der Umgebung sucht und Informationen über diese an einen zentralen Server weiterleitet. Diese Informationen enthalten die Position des Android-Geräts, die ungefähre Distanz zum Beacon (zusammengesetzt aus Genauigkeit der Position und berechneter Distanz zum Beacon), und die ID des Beacons. Die Position wird über die üblichen Android-APIs abgefragt. Hierbei wird die Quelle benutzt, die die höchste Genauigkeit bietet. Bei üblichen Smartphones gibt es eine GPS- und eine WLAN-Positionsquelle. Die GPS-Quelle bietet normalerweise eine bessere Genauigkeit, jedoch funktioniert diese nicht innerhalb von Gebäuden, weshalb in diesem Fall die WLAN-Quelle benutzt werden muss. Der Server, zu dem diese App die Daten überträgt (siehe Abschnitt 6.3), benötigt eine textuelle Darstellung der Beacon Kennung. Da allerding die ID in Binärform empfangen wird, muss diese zuerst in eine textuell darstellbare Form gebracht werden. Dies wird mittels der Base64 Kodierung [Jos06] umgesetzt. Diese Kodierung stellt beliebige Binärdaten und druckbaren ASCII-Zeichen dar und ist dadurch ideal für diese Anwendung geeignet. Im normalen Base64 Modus können jedoch Zeichen, wie zum Beispiel „/“, auftauchen, die für eine URLs von besonderer Bedeutung sind, weshalb die Standardkodierung nicht verwendet werden kann. Stattdessen gibt es noch einen URL-Modus für Base64, der diese Zeichen nicht verwendet und deshalb hier besser geeignet ist. Diese Vorkehrung ist nötig, da der Klient die IDs in einer speziell kodierten URL an den Server schickt. Um dem Nutzer eine Rückmeldung der gefundenen Beacons zu geben werden die Daten, die an den Server gesendet werden zusätzlich in einer Liste angezeigt. In Abbildung 6.1 ist ein Screenshot der App zu sehen. Am oberen Rand, markiert mit **1**, sind die Kontrollen des Tracker-Services zu sehen. Diese steuern den Android-Service, der den Beacon Modus und den Beobachter Modus implementiert. In der Mitte, markiert mit **2**, ist die Anzeige der Beacon-Daten zu sehen. Diese Daten wurden in JavaScript Object Notation (JSON)-Form (das genaue Format wird in Abschnitt 6.3 beschrieben) an den Server gesendet und enthalten UID, Position und Entfernung zu den einzelnen Beacons. Am unteren Rand, markiert mit **3**, ist eine Meldung zu sehen, die darauf hinweist, dass gerade die weiter oben stehenden Daten an den Server geschickt wurden.

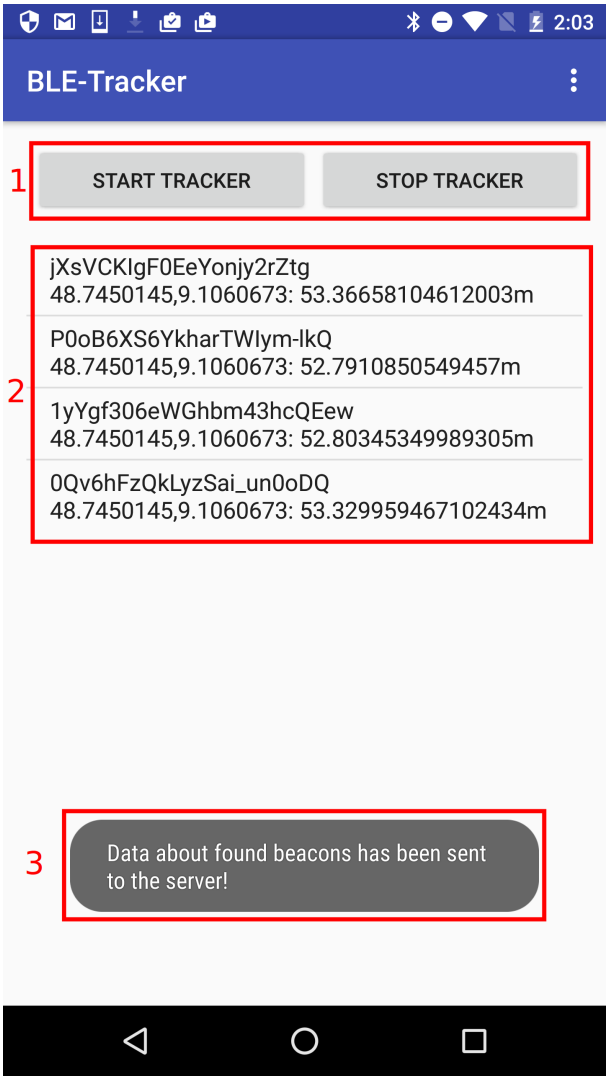


Abbildung 6.1: Die Android App hat ein rudimentäres Benutzerinterface um den Nutzer über gesendete Daten zu informieren. Die Kontrollen des Beacon und Scanner Services wurden mit **1**, **2** zeigt die Darstellung der versendeten Daten, mit **3** wurde die Meldung, dass diese Daten gerade an den Server geschickt wurde markiert.

Listing 6.1 Mögliche JSON-Daten, die an den Server geschickt werden könnten

```
[
      {
            "uid": "18LkZe12F606nPvDtKCeeA",
            "latitude": 48.745219,
            "longitude": 9.106640,
            "distance": 20.0
      },
      {
            "uid": "1yYgf306eWGhbm43hcQEew",
            "latitude": 28.561905,
            "longitude": -80.577203,
            "distance": 3.0
      }
]
```

6.3 Server

Zu Testzwecken wurde ein einfacher auf Node.js basierender Server implementiert. Dieser stellt eine representational state transfer (REST)-Application programming interface (API) zur Verfügung, über welche die Beobachter die Daten der gefundenen Beacons übermitteln können. REST [Fie00] ist eine Konvention, die eine spezielle Art von API bereitstellt, die im Web-Bereich sehr verbreitet ist. Hierbei werden Daten über speziell kodierte URLs an den Server geschickt oder von diesem abgefragt. Die URLs sind typischerweise hierarchisch aufgebaut und enthalten Server-Parameter in Form von URL Pfad Elementen. Auf dem hier vorgestellten Server ist diese REST-API unter dem Pfad „`/api/v1`“ verfügbar. Alle im Nachfolgenden genannten Pfade sind relativ zu diesem Pfad zu verstehen.

6.3.1 Daten hinzufügen

Um Positionsdaten zu einem Beacon dem Server hinzuzufügen wird eine HTTP POST Anfrage an den Server unter dem API-Pfad „`beacons`“ geschickt. Im Inhalt dieser Anfrage müssen die Daten des Beacons in einem JSON-Array an Objekten vorliegen. Jedes dieser Objekte enthält die Daten für ein Beacon. In Listing 6.1 ist beispielhaft gezeigt wie die JSON-Daten aussehen könnten. Die einzelnen Beacon-Daten werden mit der jetzigen Zeit versehen und in einer Datenbank gespeichert. Die im JSON Objekt angegebene UID wird ohne Änderungen, also in der Base64 Kodierung, in der Datenbank gespeichert, da übliche relationale Datenbanken (wie zum Beispiel MySQL) Text-Strings anders verarbeiten als binäre Daten. Sollte dies ein Problem darstellen könnten die Daten auch als Binärstring abgespeichert werden.

Listing 6.2 Mögliche JSON-Daten, die der Server auf eine Anfrage zurückschicken könnte

```
[
      {
            "uid": "18LkZe12F606nPvDtKCeeA",
            "latitude": 48.745219,
            "longitude": 9.106640,
            "distance": 20.0,
            "timestamp": "2016-05-15T20:23:24.000Z"
      }
]
```

6.3.2 Daten abfragen

Anfragen über die Position eines Beacons können in zwei Arten an den Server geschickt werden. Wenn nur die Daten zu einem einzelnen Pseudonym abgefragt werden sollen, kann dies mit einem HTTP GET und der „`beacons/:uid`“ API-Methode gemacht werden. `:uid` ist ein Platzhalter für die UID, die abgefragt werden soll. Der Server wird bei dieser Anfrage die Datenbank nach der gefragten UID durchsuchen und alle Beacon-Daten zurückgeben, die die gleiche UID haben. Um festzustellen ob ein Datensatz zu der gefragten UID gehört wird ein normaler Stringvergleich (bzw. eine `WHERE Uid = ':uid'` SQL-Anweisung) benutzt. Damit diese Operation für alle Beobachter und Klienten gleich verläuft müssen sich alle an die in Abschnitt 6.2 vorgestellten Regeln zur Formatierung der Binär-UIDs halten. Wenn ein Klient die Daten zum Pseudonym „`18LkZe12F606nPvDtKCeeA`“ anfragen möchte so würde dieser den Pfad „`beacons/18LkZe12F606nPvDtKCeeA`“ aufrufen. Der Server würde die Datenbank durchsuchen und wird mit den Daten der Beacons antworten. In Listing 6.2 ist eine mögliche Antwort zu sehen. Diese entspricht den Daten, die in Listing 6.1 für diese UID angegeben wurde, jedoch wurde zusätzlich noch der Zeitpunkt der Eintragung hinzugefügt. Es ist natürlich möglich, dass der Server keine übereinstimmende Datensätze findet. In diesem Falle würde ein leeres JSON-Array zurückgegeben werden.

Da je nach Änderungsperiode sehr viele Pseudonyme von einem Beacon verwendet werden können, gibt es eine weitere Möglichkeit die Daten abzufragen, die nicht für jede UID eine neue HTTP-Verbindung benötigt. Hierzu werden mehrere UIDs in einer Anfrage an den Server geschickt und der Server gibt alle Datensätze zurück, die zu einer dieser UIDs gehört. Dazu muss ein Klient eine HTTP PUT Anfrage an die „`beacons`“ Methode schicken. Im Inhalt der Anfrage sind alle UIDs in einem JSON-Array kodiert, die abgefragt werden sollen. Der Server antwortet anschließend mit dem gleichen Format wie für eine einzelne UID. Es wird nicht sichergestellt, dass die Daten sortiert oder in irgendeiner anderen Art nach UID gruppiert sind. Dieser Operation muss der Klient lokal ausführen, wenn dies vonnöten ist.

6.4 Web-Klient

Um die Server-Daten zu visualisieren wurde ein entsprechender Web-Klient geschrieben. Dieser benutzt die oben vorgestellte REST-API, um Informationen über die Beacons anzufragen und auf einer Karte darzustellen.

Der Algorithmus, den der Web-Klient benutzt, um ein Beacon zu finden, ist in Algorithmus 6.1 gezeigt. Der Web-Klient wird mit dem 128-bit langen Schlüssel und dem Initialisierungsvektor für AES im CBC-Modus initialisiert. Diese Daten werden zur Erzeugung der ersten zehn Pseudonyme genutzt und an den Server geschickt. Die zurückgegebenen Werte werden zur Bestimmung des Startpunkts der Suche benötigt. Wenn kein Beacon mit einem der ersten zehn Pseudonyme gefunden wird, so wird die Suche abgebrochen, da kein Anfangszeitpunkt gefunden werden konnte. Ohne diese Vorkehrung wäre es nicht möglich festzustellen, ob ein Beacon jemals gefunden wurde, da die erste gefundene UID zu einem deutlich späteren Zeitpunkt auftreten könnte. Dies sollte in der Praxis kein Problem darstellen, da ein Besitzer des Beacons einfach bei Erhalt des Beacons die ersten Pseudonyme mit der Android-App an den Server schicken könnte. Sobald die erste Position des Beacons gefunden wurde beginnt die normale Suche. Die Pseudonyme haben einen direkten Zusammenhang mit der tatsächlichen Zeit, da diese in einem festen Intervall geändert werden. Dies kann zur genutzt werden, um abzuschätzen zu welchem Zeitpunkt ein Pseudonym aktiv ist. Hierdurch werden nur Pseudonyme gesucht, die tatsächlich bereits gesehen werden konnten. Das Intervall wird zu Beginn mit fünf Minuten initialisiert, jedoch wird dieses Intervall neu berechnet, sobald ein Beacon vom Server gefunden wurde. Hierdurch kann sich der Klient auf beliebige Änderungsintervalle einstellen, ohne Vorwissen über die Implementierung des Beacons haben zu müssen. Zusätzlich wird hierdurch auch die Problematik der ungenauen Uhren gelöst, da ein Beacon mit einer ungenauen Uhr aus Sicht des Klienten wie ein Beacon mit einem anderem Änderungsintervall aussehen würde.

Nachdem der Klient festgestellt hat, dass keine weiteren Pseudonymwechsel stattfinden können, wird die Suche beendet und die gefundenen Positionen werden auf einer Karte im Browser dargestellt. Das Ende der Suche ist dann erreicht, wenn der berechnete Zeitpunkt, zu der ein Pseudonym aktiv wäre nach der aktuellen Zeit liegt. Dies kann berechnet werden, da im Verlauf des Algorithmus die Änderungdperiode des Beacons bestimmt wurde. In Abbildung 6.2 ist der Web-Klient in diesem Status gezeigt. In der Mitte, markiert mit **1**, sind mehrere gemeldete Positionen des Beacons sichtbar. Jede Markierung stellt eine Meldung eines Beobachters mit der Position und geschätzter Entfernung zum Beacon dar. In der unteren Hälfte, markiert mit **2**, ist die Kontrollleiste der Zeit zu sehen. Mit dieser kann der Nutzer kontrollieren zu welcher Zeit die Position des Beacons gezeigt werden soll. In dieser Implementierung wurde die Möglichkeit benutzt mehrere Kennungen auf einmal abzufragen. Hierzu werden bis zu 32 Kennungen gruppiert und an den Server geschickt, wodurch die Suche deutlich schneller ausgeführt werden kann.

Die Daten, die der Klient erhält, sind offensichtlich nicht kontinuierlich. Hierdurch ergeben sich Ungenauigkeiten in der Bestimmung des aktuellen Pseudonyms, wodurch es möglich wird,

Algorithmus 6.1 Algorithmus zum Finden eines Beacons

```
function FindBeacon
    beaconPeriod ← 300, timestamp ← 0                    // Standardwert ist 5 Minuten
    for i ∈ {1, ..., 10} do
        timestamp ← timestamp + 1
        firstData ← getBeaconData(timestamp)
        if firstData ≠ null then
            firstTimestamp ← timestamp
            break
        end if
    end for
    if firstData = null then
        return null
    end if
    while firstData.time + (timestamp + 1 − firstTimestamp) · beaconPeriod < NOW() do
        timestamp ← timestamp + 1
        data ← getBeaconData(timestamp)
        if data = null then
            continue
        end if
        lastData ← data
        beaconPeriod ← (data.time − firstData.time) / (firstTimestamp − timestamp)
    end while
    return lastData
end function
```

dass der Klient Anfragen zu einem Pseudonym stellt, die noch gar nicht in Benutzung sind. Hierdurch könnte der Server ein Beacon länger verfolgen, da ein Zusammenhang zwischen Pseudonymen erstellt werden kann. Je öfter ein Beacon von einem Beobachter gesehen wird desto genauer wird auch die Abschätzung des Änderungsintervalls, wodurch auch die Chance reduziert wird, dass mehr Pseudonyme als nötig abgefragt werden.

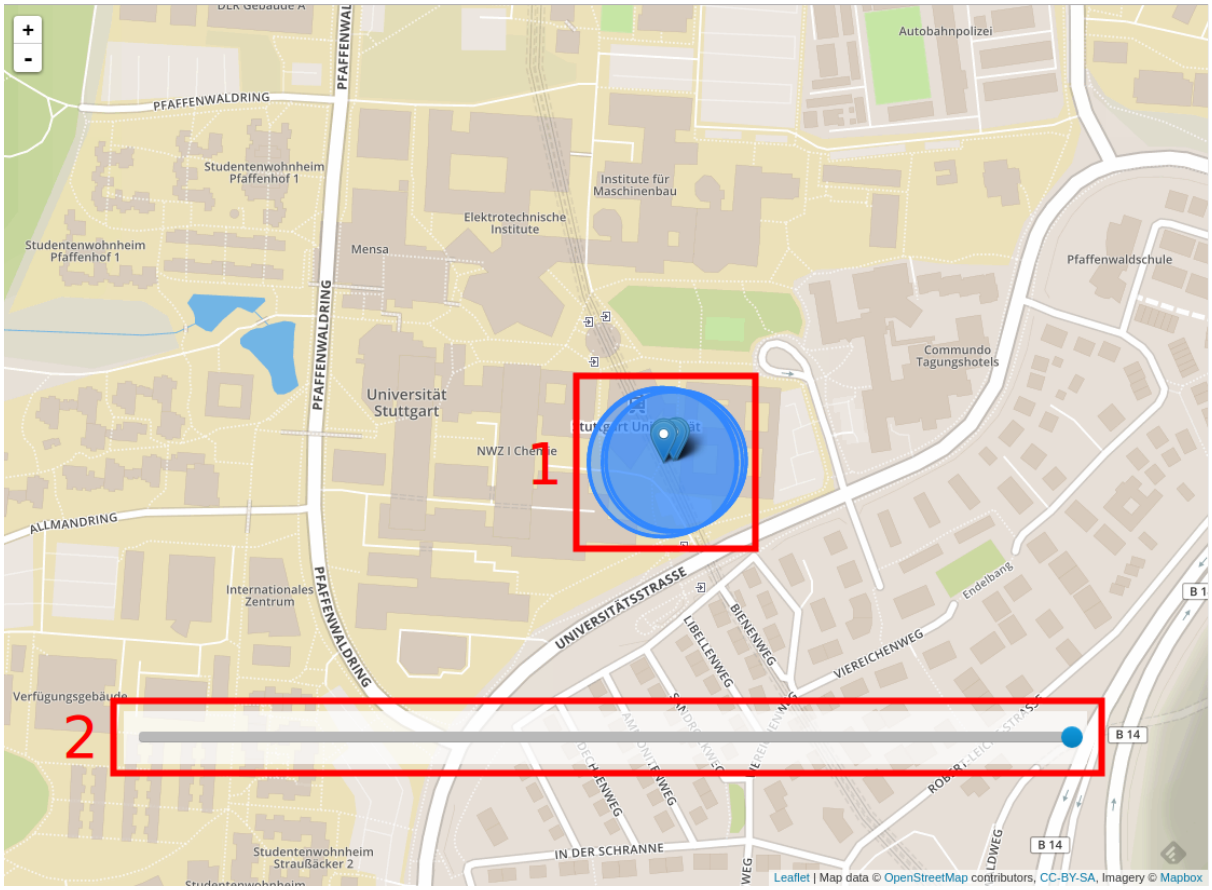


Abbildung 6.2: Web-Seite der Server REST-API zur Verfolgung der Position eines Beacons. Bei **1** sind die letzten Positionen des Beacons zu sehen. **2** zeigt das Kontrollelement, mit dem der anzuzeigende Zeitraum ausgewählt wird.

7 Evaluation

In diesem Kapitel werden die vorgestellten Kapitel bezüglich der Folgenden Eigenschaften bewertet

- Energieeffizienz
- Effektivität der Schutzmechanismen

7.1 Energieverbrauch

Der Energieverbrauch der beiden Beacon-Implementierungen wurde mit dem Low-Energy-Meter evaluiert. Die genaue Funktionsweise dieses Messaufbaus wird in Abschnitt 7.1.1 beschrieben. Die Beacons wurden an dieser Vorrichtung angeschlossen und über eine bestimmte Messzeit betrieben, um daraus den Energieverbrauch zu analysieren.

7.1.1 Evaluationsmethoden

Zur Bestimmung und Analyse des Energieverbrauch wurde der Low-Energy-Meter benutzt, der die Energieaufnahme messen kann. Die erzeugten Daten wurden anschließend mit Hilfe einiger selbst geschriebener Python-Skripten analysiert.

Low-Energy-Meter

Der Low-Energy-Meter [Dür16b] ist eine Kombination von Hardware und Software, die speziell auf die Messung von Geräten ausgelegt ist, die im Mittel nur sehr wenig Energie aufnehmen, jedoch gleichzeitig hohe Spitzenströme benötigen. Bei BLE-Beacons ist dies von Nutzen, da bei inaktivem BLE Modul relativ wenig Leistung benötigt (im Bereich von weniger als $0.5\,\mathrm{mW}$) wird, jedoch kann es bei einem Statuswechsel und dem An- bzw. Ausschalten des Moduls zu hohen Spitzenströmen (Leistungswerte im Bereich von $5\,\mathrm{mW}$) kommen. Aus diesem Grund ist ein üblicher Versuchsaufbau mit Multimeter oder Oszilloskop nicht gut geeignet, um den Energieverbrauch zu messen.

Um dieses Problem zu lösen benutzt der Low-Energy-Meter einen Kondensator, welcher die Energie für ein angeschlossenes Gerät speichert. Dieser Kondensator kann sowohl sehr kleine

Ströme wie auch die hohen Spitzenströme liefern, ohne dass es zu Spannungseinbrüchen oder anderen Problem kommt. Ein typisches BLE-Gerät kann mit dieser Energiequelle mehrere Minuten betrieben werden, womit der Energieverbrauch auch über längere Zeiträume gemessen werden kann. Um den Energieverbrauch zu messen wird die Spannung am Kondensator mit einem Analog/Digitalkonverter in einen Digitalwert übersetzt und per GPIO an einen Raspberry-Pi übergeben. Da die Messungen über einen längeren Zeitraum ausgeführt werden sollen, wird mit zwei Relais automatisch gesteuert, ob der Kondensator das Gerät versorgt, oder ob dieser aufgeladen wird. Bei der Versorgung des Geräts bilden Kondensator und Gerät einen geschlossenen Stromkreis, der nur durch den Kondensator versorgt wird. Wenn der Kondensator aufgeladen werden soll, so werden die zwei Relais umgeschaltet, die das Gerät und den Kondensator mit der externen Spannungsversorgung verbinden. Dadurch ist die Geräteversorgung sichergestellt und der Kondensator wird gleichzeitig aufgeladen.

Die Software des Raspberry-Pi kontrolliert die Ansteuerung der Relais und schreibt die aufgenommenen Spannungswerte in eine Datei. Diese Datei enthält den aktuellen Zeitpunkt und den gemessenen Spannungswert. Diese Werte können später ausgewertet werden um den Energieverbrauch zu berechnen. Wenn ein Test begonnen werden soll, wird dieses Programm auf dem Raspberry-Pi gestartet. Das Programm erhält mehrere Parameter, die das Laufzeitverhalten beeinflussen. Die wichtigsten Parameter sind die Ober- und Untergrenzen der Spannung, die am Kondensator anliegen dürfen. Die Obergrenze legt die Spannung des Kondensator zu Beginn einer Messperiode fest, typischerweise sind dies in diesem Anwendungsfall $3.3\,\mathrm{V}$. Die Untergrenze legt die Mindestspannung des Kondensators fest, ab welcher dieser wieder aufgeladen werden soll. In den nachfolgenden Versuchen beträgt diese Untergrenze $2.6\,\mathrm{V}$, um Unterspannungsprobleme mit dem Faros-Board zu verhindern (siehe Abschnitt 7.1.2 auf der nächsten Seite). Dieser Mechanismus ermöglicht es den Energieverbrauch über längere Perioden zu messen, ohne manuell eingreifen zu müssen. Die Hardware des Low-Energy-Meters ist in Abbildung 7.1 gezeigt. Auf der rechten Seite ist der Raspberry-Pi zu sehen, der die Messung mit der Software steuert. Auf der linken Seite ist die Messhardware gezeigt. Der Kondensator der Messhardware hatte für diese Versuche eine Kapazität von $10\,000\,\mu\mathrm{F}$.

Analyse der Messwerte

Nach der Aufnahme der Energiedaten wurden diese mittels einer Reihe von Python-Skripten ausgewertet. Um den durchschnittlichen Energieverbrauch zu berechnen wird die Spannung zu Beginn und Ende einer Messperiode mit der Standardformel $E = \frac{1}{2}CU^2$ in einen Energiewert umgerechnet. Der durchschnittliche Energieverbrauch wurde mit $P = \frac{\Delta E}{\Delta t}$ berechnet. ΔE ist der Unterschied der im Kondensator gespeicherten Energie zu Beginn und Ende einer Entladeperiode. Δt ist der Zeitunterschied zwischen Beginn und Ende der Entladeperiode. Mit diesen Werten wurde der durchschnittliche Energieverbrauch der Beacons bestimmt.

Zusätzlich wurden die Daten auch für die Erzeugung von Graphen benutzt, welche Aufschluss über das Laufzeitverhalten der Beacons geben. Hierzu wurde der Energieverbrauch innerhalb von zwei Sekunden berechnet und in einem Graphen aufgetragen. Um Fluktuationen zu Beginn

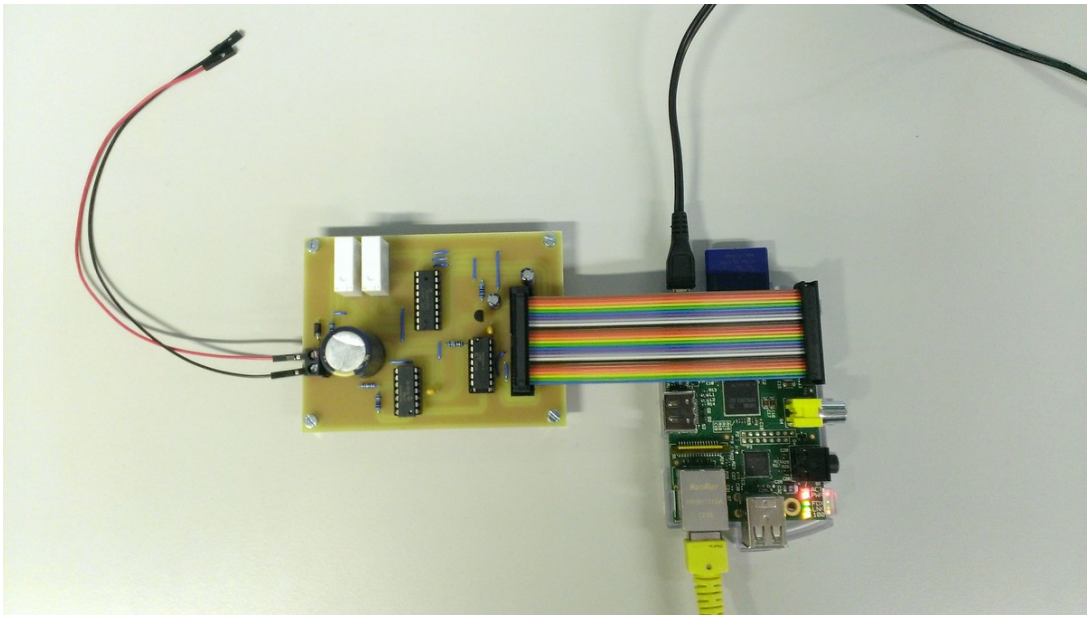


Abbildung 7.1: Aufbau des Low-Energy-Meters mit Raspberry-Pi und Mess-Hardware

einer Messperiode auszugleichen, wurden nur Werte unterhalb von 3 V in der Auswertung berücksichtigt. Durch den Einsatz des Analog-Digital-Konverters kam es gehäuft zu Messfehlern, da dieser zwischen zwei aufeinanderfolgenden Werten fluktuiert, wenn nur ein kleiner Strom fließt. Da bei diesen Versuchen selbst die kleinste Änderung des Digitalwerts eine signifikante Änderung des gemessenen Energieverbrauchs darstellte, mussten diese Werte angepasst werden, um die Fehler zu reduzieren. Aus diesem Grund wurden die Werte des Low-Energy-Meters zuerst durch eine Glättungsfunktion geleitet, um die größten Fehler auszugleichen. Dies hat die Qualität der Messdaten verbessert, jedoch kam es weiterhin zu Fluktuationen bei den Messungen. Dies ist bei Graphen des Energieverbrauchs zu sehen (Abbildung 7.3 und Abbildung 7.5), jedoch beeinflussen diese Fluktuationen nicht die Genauigkeit der Berechnung des durchschnittlichen Energieverbrauchs, da dieser durch die Spannungsdifferenz zu Beginn und Ende einer Messperiode berechnet wurde. Die Zeit und die Spannung dieser Werte liegen weit auseinander, weshalb bei diesen Messungen die Messungenauigkeiten keine Auswirkungen auf die Berechnung des Durchschnittsverbrauchs hatten.

7.1.2 Faros-Beacon

Mithilfe des in Abschnitt 7.1.1 vorgestellten Low-Energy-Meters wurde der Energieverbrauch der auf dem Faros-Beacon basierenden Implementierung evaluiert. Hierzu wurde das Faros Beacon an die Stromversorgung des Low-Energy-Meters angeschlossen (siehe Abbildung 7.2). Anschließend wurde der Energieverbrauch über eine Länge von 90 Minuten gemessen, um einen guten Mittelwert für den Energieverbrauch zu ermitteln. Aus diesen Werten ergab sich ein mittlerer Energieverbrauch von ca. 100 µW bei einer Änderungsperiode von 30 s.

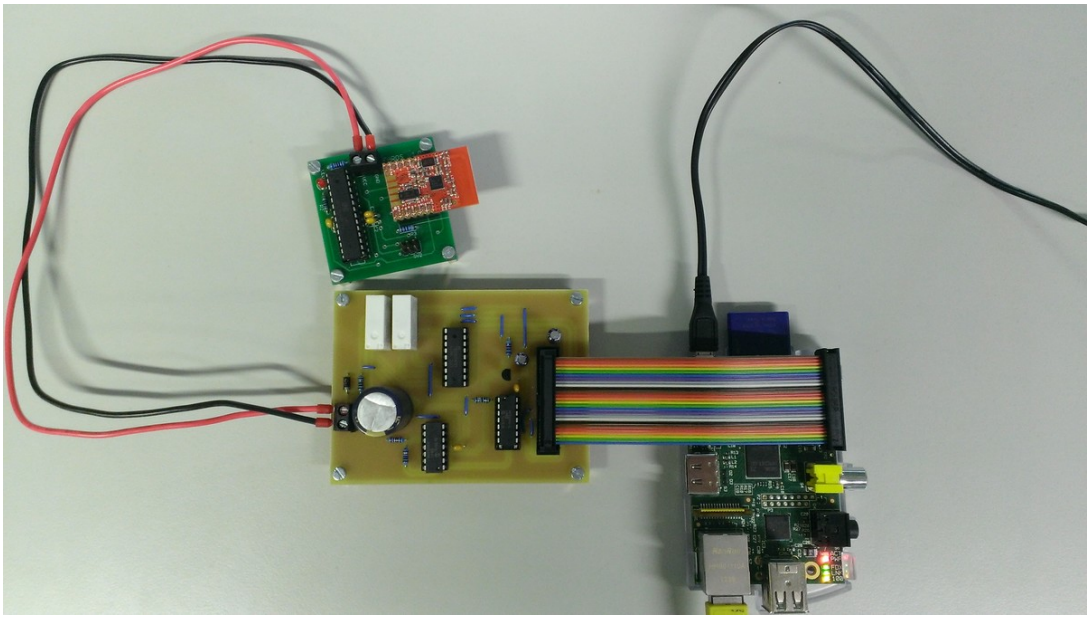


Abbildung 7.2: Versuchsaufbau des Faros-Beacons. Die Spannungsversorgung des Faros Boards wurde mit der Versorgung des Low-Energy-Meters verbunden.

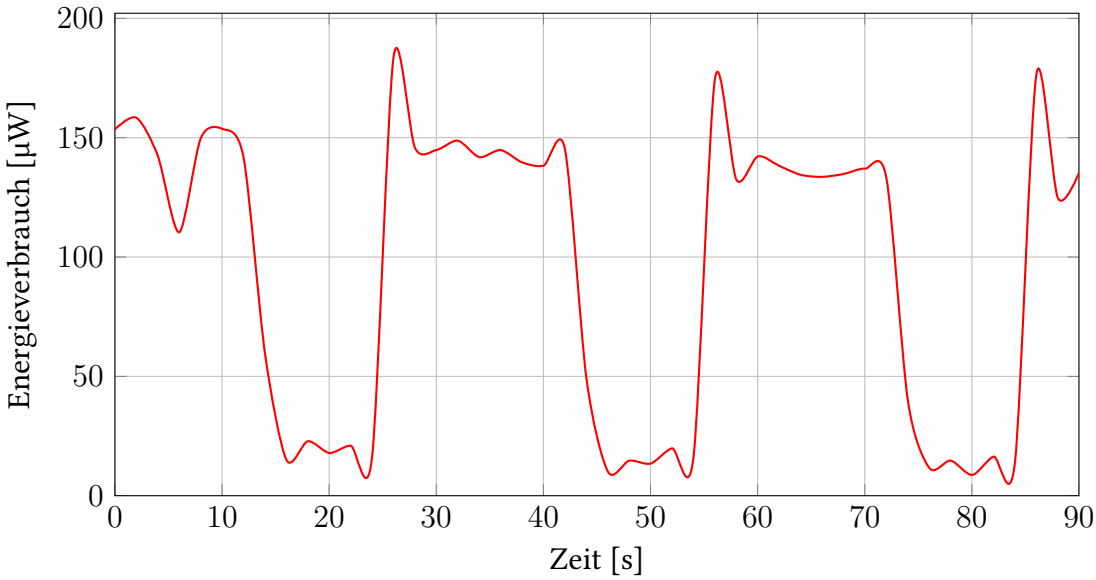


Abbildung 7.3: Energieverbrauch des Faros Beacons. Auf der X-Achse ist die Zeit in Sekunden aufgetragen. Auf der Y-Achse ist der Energieverbrauch in µW gezeigt.

Bei ersten Tests zum Energieverbrauch des Faros Beacons kam es zu Komplikationen, da beim Wiederaufladen des Kondensators des Low-Energy-Meters das Faros-Beacon in eine Fehlerroutine gesprungen ist. Diese Fehlerroutine wurde durch eine Endlosschleife implementiert, wodurch das Beacon seine eigentliche Arbeit nicht weiterführen konnte. Eine mögliche Ursache dieses Fehlers ist ein kurzer Spannungsabfall beim Umschalten der Relais am Kondensator, da hierbei eine Diode benutzt wurde, um sicherzustellen, dass kein Strom in die entgegengesetzte Richtung fließen kann. An dieser Diode fällt eine kleine Spannung ab (ca. $0.5\,\mathrm{V}$), wodurch die effektive Spannung an der Beacon Stromversorgung unter den minimalen Wert der CPU fiel. Hierdurch wurde höchstwahrscheinlich ein teilweiser Reset des Systems ausgelöst, wodurch beim wiederholten Starten des Faros-Prozessors die Fehlerroutine ausgelöst wurde. Durch Anheben der Minimalspannung des Low-Energy-Meters auf $2.6\,\mathrm{V}$ konnte dieses Problem umgangen werden.

In Abbildung 7.3 ist der Energieverbrauch in einer typischen Messperiode gezeigt. Der in Kapitel 5 auf Seite 33 beschriebene Änderungszyklus eines Beacons ist hier gut zu sehen. Zu Beginn einer Periode gibt es eine Spitze im Energieverbrauch, die durch die Berechnung des neuen Pseudonyms verursacht wird. Für $20\,\mathrm{s}$ ist anschließend das BLE-Modul aktiv und benötigt hierzu die entsprechende Energie. Die CPU ist in dieser Zeit kaum aktiv und überwacht nur die Zeit und die Operation des BLE-Moduls. Nach diesen $20\,\mathrm{s}$ beginnt die beschriebene Funkstille in der das BLE-Modul inaktiv ist. Dies ist gut durch den radikal gesunkenen Energieverbrauch zu erkennen. Hier ist auch der Basis-Energieverbrauch der CPU zu erkennen, welcher bei ca. $25\,\mu\mathrm{W}$ liegt. Hierdurch lässt sich auch ein ungefährer Energieverbrauch von $115\,\mu\mathrm{W}$ des BLE-Moduls errechnen.

Mit dem gemessenen Energieverbrauch kann auch eine Abschätzung der Laufzeit des Beacons berechnet werden, wenn dieses mit einer Batterie betrieben würde. Wenn eine Standard CR2032 Knopfzelle mit einer typischen Kapazität von $225\,\mathrm{mAh}$ benutzt werden würde, könnte das Beacon

$$t = \frac{225\,\mathrm{mA\,h}}{\frac{100\,\mu\mathrm{W}}{3\,\mathrm{V}}} = \frac{225\,\mathrm{mA\,h}}{33.33\,\mu\mathrm{A}} = 2.43 \times 10^7\,\mathrm{s}$$

betrieben werden. Dies entspricht ca. 281 Tagen, jedoch ist dieser Wert nicht genau, da mehrere Faktoren die Laufzeit beeinflussen. Der gemessene Energieverbrauch ist gemittelt über eine längere Zeit, bei der die Spannung zwischen $3\,\mathrm{V}$ und $2.6\,\mathrm{V}$ lag. Jedoch wurde in der Berechnung oben eine konstante Spannung von $3\,\mathrm{V}$ angenommen weshalb der tatsächliche Energieverbrauch bei Betrieb mit dieser Batterie höchstwahrscheinlich höher ausfallen würde. Auch ist die Spannung der Batterie während dessen Lebenszeit nicht konstant, sondern nimmt mit zunehmender Entladung ab. Dies hätte aber gleichzeitig Einfluss auf den Energieverbrauch, daher sollte das genannte Ergebnis nur als ungefähre Laufzeit angenommen werden.

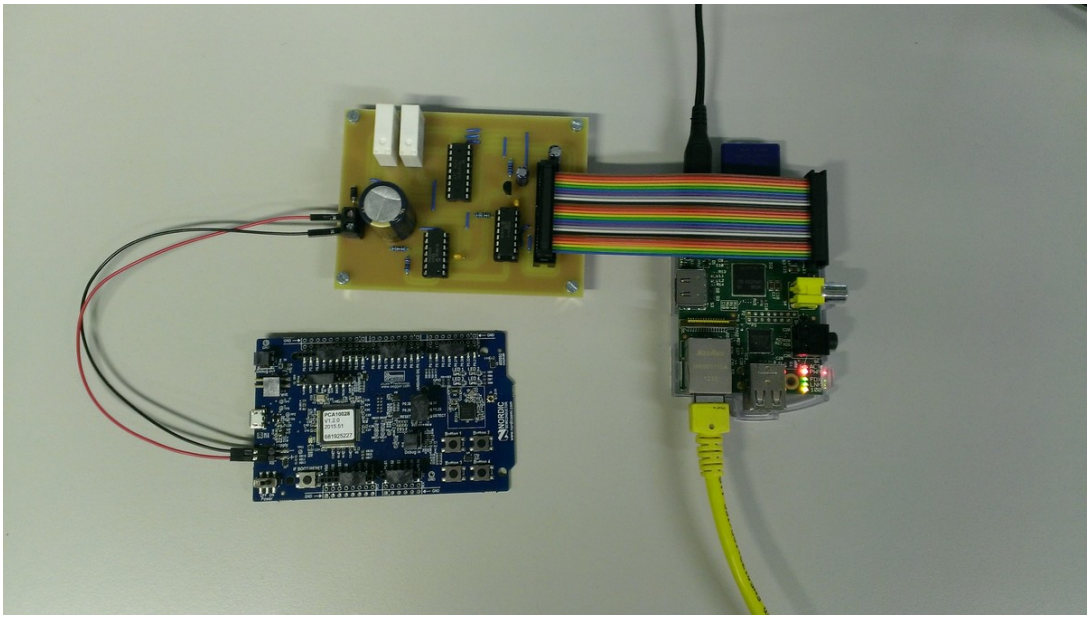


Abbildung 7.4: Versuchsaufbau des nRF51-Beacons. Die externe Spannungsversorgung des Entwicklerkits wurde mit der Spannungsversorgung des Low-Energy-Meters verbunden.

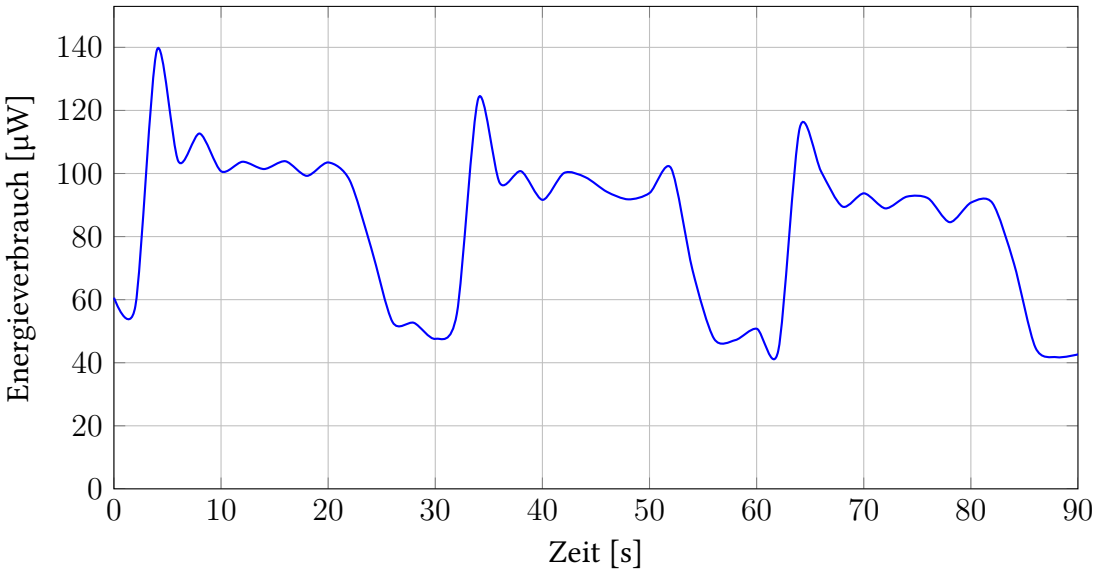


Abbildung 7.5: Energieverbrauch des nRF51 Entwicklerkits. Auf der X-Achse ist die Zeit in Sekunden aufgetragen. Auf der Y-Achse ist der Energieverbrauch in µW gezeigt.

7.1.3 Nrf51 Beacon

Die Beacon-Implementierung basierend auf dem nRF51 SoC wurde mit dem in Abbildung 7.4 gezeigten Versuchsaufbau evaluiert. Hierbei wurde die Stromversorgung des Low-Energy-Meters an die externe Stromversorgung des Entwicklerkits angeschlossen. Mit dem oben beschriebenen Low-Energy-Meters wurde anschließend der Energieverbrauch des Beacons gemessen.

In Abbildung 7.5 ist der berechnete Energieverbrauch des Beacons zu sehen. Hier ist wiederum zu erkennen, wie das Beacon abwechselnd zwischen dem sendenden und stillen Modus wechselt. Dies lässt sich, wie beim Faros Beacon, durch den reduzierten Energieverbrauch während der Funkstille erkennen. In diesem Testlauf war das Beacon auf ein Änderungsintervall von 30 Sekunden konfiguriert. Auch ist erkennbar, dass zu Anfang jeden Sendeperiode mehr Energie verbraucht wird als danach. Dies lässt sich, wie bei dem Faros-Beacon, auf die Generierung des neuen Pseudonyms per AES zurückführen. Allerdings ist beim nRF51 der Energieverbrauch dieser Aktion deutlich geringer als beim Faros-Beacon. Durch Analyse des Energieverbrauchs lässt sich ein Basisenergieverbrauch des Prozessors von ca. $50\,\mu W$ erkennen. Dies ist verglichen mit dem Faros Beacon etwa doppelt so viel, so dass man davon ausgehen kann, dass dessen CPU deutlich effizienter zu arbeiten scheint. Die Implementierung auf dem nRF51 wurde jedoch nicht ausdrücklich auf Energieeffizienz optimiert. So wären es noch möglich mehr Möglichkeiten auszuschöpfen, wie zum Beispiel die Benutzung des „System Off“ Modus, bei dem der Energieverbrauch signifikant reduziert wäre. Dieser Modus hat jedoch spezielle Anforderungen an die Implementierung und wurde deshalb in dieser Version der Beacons nicht benutzt. Mit den Energiedaten der Funkstille kann auch der ungefähre Energieverbrauch des BLE-Moduls berechnet werden, welcher bei ca. $45\,\mu W$ liegt, deutlich effizienter als das Modul des Faros Beacons. Im Mittel verbrauchte das nRF51 Beacon während der gesamten Messung, welche wiederum 90 Minuten lief, näherungsweise $80\,\mu W$. Mit der gleichen Berechnung wie oben könnte das Beacon mit diesem Energieverbrauch

$$t = \frac{225\,\text{mA h}}{\frac{80\,\mu\text{W}}{3\,\text{V}}} = \frac{225\,\text{mA h}}{26.67\,\mu\text{A}} = 3.038 \times 10^7\,\text{s}$$

betrieben werden. Dies entspricht 351 Tagen oder fast einem Jahr. Allerdings gelten hierbei wiederum die gleichen Einschränkungen wie bei Faros-Beacon, da der gemessene Energieverbrauch nicht genau dem tatsächlichen Energieverbrauch bei Betrieb mit einer Batterie entsprechen wird.

7.1.4 Vergleich

Aus den oben berechneten Durchschnittswerten lässt sich bereits feststellen, dass das Beacon auf nRF51 Basis energieeffizienter arbeitet als das Beacon auf Faros-Basis. Ein Großteil dieser Energieeffizienz lässt sich jedoch auf das BLE-Modul zurückführen, da dieses sowohl beim

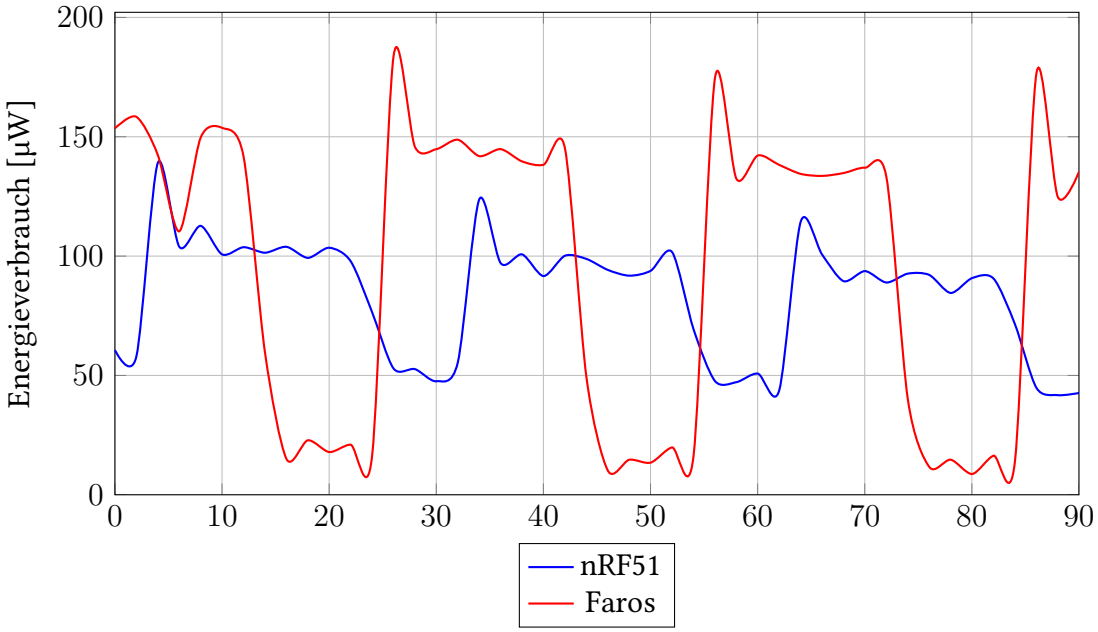


Abbildung 7.6: Dies zeigt einen Vergleich des Energieverbrauchs der beiden Beacon-Implementierungen. Auf der X-Achse ist die Zeit in Sekunden aufgetragen. Auf der Y-Achse ist der Energieverbrauch in µW aufgetragen.

Faros als auch beim nRF51 Beacon den Großteil des Energieverbrauchs ausmacht. Unter Berücksichtigung des Energieverbrauchs ließ sich eine ungefähre Abschätzung des Energieverbrauchs der einzelnen Module berechnen. Das Faros Beacon verbraucht demnach ca. 115 µW für das BLE-Modul wohingegen das Modul des nRF51 nur ca. 45 µW verbraucht. Zusätzlich ist auch zu erkennen, dass der Energiebedarf zur Berechnung eines neuen Pseudonyms beim Faros Beacon deutlich höher ausfiel als beim nRF51 Beacon, obwohl die Implementierung des Faros Beacons eine speziell auf diese CPU ausgerichtete Assembler-Implementierung von AES verwendete. Das nRF51 Beacon benutzt nur eine reine C-Implementierung, die höchstwahrscheinlich nicht so effizient arbeitet, wie eine äquivalente Assembler Implementierung. Dies lässt sich durch die grundlegenden Unterschiede zwischen den beiden CPU-Typen erklären. Das Faros Board benutzt eine 8-bit CPU, die als Mikrokontroller gedacht ist. Im Gegensatz hierzu wird von dem nRF51 Beacon eine 32-bit ARM Cortex-M0 CPU benutzt, die auch für Mikroprozessoren ausgelegt ist, jedoch basiert diese CPU auf der ARM-Architektur, die auf allgemeinere Aufgaben ausgelegt ist und deutlich effizienter hergestellt werden kann als die ATMega-Prozessoren, die beim Faros Beacon eingesetzt werden.

In Abbildung 7.6 ist ein direkter Vergleich des Energieverbrauchs gezeigt. Es ist gut zu erkennen, dass beide Beacons dem gleichen Aktiv-Inaktiv-Phasenrhythmus folgen, allerdings hat das Faros Beacon im aktiven Modus einen höheren Energieverbrauch als das nRF51 Beacon, jedoch ist der Energieverbrauch im inaktiven Modus geringer. Dies zeigt, dass die CPU des Faros Beacon im Moment deutlich energieeffizienter arbeiten kann als die CPU des nRF51 Beacons.

Durch zusätzliche Optimierungen könnte dieser Unterschied höchstwahrscheinlich noch reduziert werden.

7.2 Privatsphäre

Wie bereits in Abschnitt 5.3.2 auf Seite 38 angesprochen wurde kann es zu Privatsphärenproblemen kommen, wenn zu wenig Beacons innerhalb eines Gebiets vorhanden sind. Um dieses Problem zu analysieren wurde der Mobility Generator „BonnMotion“ [AEGS10] benutzt. BonnMotion ist ein Java-Projekt, das zufällige Bewegungsprofile erzeugen kann. Diese Bewegungsprofile werden anschließend zur Simulation der Beacon-Bewegungen benutzt, womit ein globaler Beobachter, wie zum Beispiel der Server, modelliert werden kann.

7.2.1 Evaluationsmethoden

Der „RandomWalk“ Generator von Bonnmotion wurde benutzt um Bewegungsprofile zu generieren. Um die Fluktuation der Ergebnisse zu reduzieren wurden 20 separate Bewegungsprofile pro Knotenanzahl angefertigt. Die Größe des analysierten Gebietes wurde bei BonnMotion auf einen Quadratkilometer festgelegt. Dies erlaubt es die Anzahl der Knoten direkt mit der durchschnittlichen Bevölkerungsdichte zu vergleichen. Die Knotenanzahl wurde zwischen 20 und 600 Knoten variiert, um eine Aussage über die Abhängigkeit der Nachverfolgbarkeit von der Knotendichte machen zu können.

Nachdem die Daten von BonnMotion generiert wurden, wurde ein Python Skript benutzt um diese zu analysieren. Da für die Evaluation interessant ist wie lange ein einzelnes Beacon verfolgbar ist, wurde alle 20 Zeiteinheiten die Position jedes Knotens erfasst. Zu diesem Zeitpunkt wird angenommen, dass das Beacon gerade in der Funkstille ist und sein Pseudonym geändert hat. Wenn ein Knoten innerhalb 20 Meter eines anderen Knotens war, wurde angenommen, dass diese beiden Knoten innerhalb der gemeinsamen Mix-Zone waren, wodurch sie nicht mehr voneinander unterscheidbar und damit nicht mehr verfolgbar wären. Diese Daten wurden zur Berechnung der Zeit genutzt, während der ein globaler Beobachter (zum Beispiel der Server) ein Beacon nachverfolgen könnte. Dies lässt Rückschlüsse auf die Nachverfolgbarkeit eines Beacons seitens des Servers zu. Die Daten wurden über eine Zeitperiode von 900 Zeiteinheiten analysiert, wodurch ein Nachverfolgbarkeitswert von 900 den schlechtesten Wert darstellt.

7.2.2 Ergebnisse

Mit den in Abschnitt 7.2.1 beschriebenen Methoden wurden die von BonnMotion generierten Bewegungsprofile analysiert. Die gewählte Beaconanzahl ist an der durchschnittlichen Bevölkerungsdichte orientiert. In Deutschland leben zum Beispiel in ländlichen Gebieten weniger als 120 Menschen pro Quadratkilometer, wohingegen in Stadtgebieten mehr als 1000

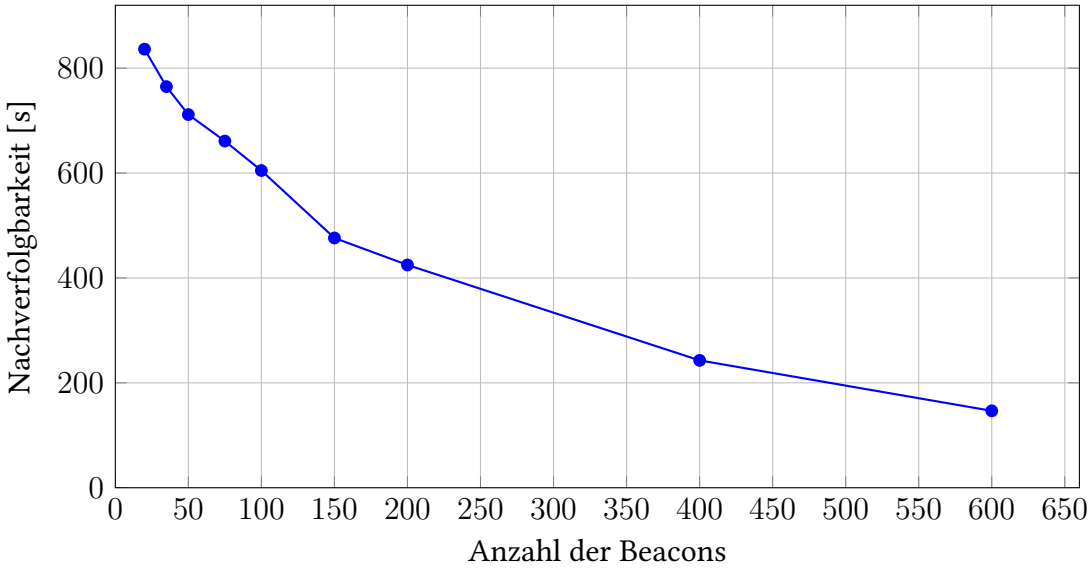


Abbildung 7.7: Privatsphäre in Relation zu Beacon-Dichte. Die X-Achse gibt die Anzahl der Beacons pro km^2 an. Die Y-Achse gibt an wie lange im Durchschnitt ein Beacon verfolgt werden konnte

Menschen pro Quadratkilometer wohnen. In Innenstädten ist diese Dichte noch höher [Sta13], weshalb die höchste Beaconanzahl (600) der Dichte einer kleinen Stadt oder einer Vorstadt entspricht. In Abbildung 7.7 sind die berechneten Daten zu sehen. Hierbei ist zu erkennen, dass die nachverfolgbare Zeit mit der Dichte der Beacons abnimmt. Bei 20 Beacons pro km^2 ist ein einzelnes Beacon ohne Probleme nachverfolgbar. Diese Nachverfolgbarkeit nimmt jedoch rapide ab, je mehr Beacons in einem Gebiet vorhanden sind. Bei 600 Beacon pro km^2 ist ein Beacon im Mittel nur noch ca. 170 s verfolgbar.

8 Zusammenfassung und zukünftige Arbeiten

Diese Arbeit hat ein System vorgestellt, welches Schutzmechanismen bei der Nachverfolgung von BLE-Geräten realisiert. Hierfür wurde ein Systemmodell aufgestellt, das beschreibt, welche Teilnehmer in diesem System vertreten sind und wie diese interagieren. Hierbei gibt es das BLE-Beacon, das eine Kennung aussendet, die dieses Beacon identifiziert. Die Beobachter empfangen diese Kennung und melden diese mit einer ungefähren Position an den Server. Dieser stellt eine Schnittstelle bereit, die es einem Besitzer einer Klienten-Software ermöglicht die Position eines Beacons abzufragen.

Aufbauend auf diesem Modell wurden die Schutzmechanismen entworfen. Hierbei wurden zwei Konzepte aus anderen Forschungsgebieten benutzt, zum einen wurden Pseudonyme eingeführt, die aus dem Bereich der VANets bekannt sind. Diese Pseudonyme werden periodisch geändert, um die Verfolgbarkeit eines Beacons zu reduzieren. Diese Pseudonyme sehen für die Beobachter wie zufällige Daten aus, jedoch wird zur Generierung dieser ein Algorithmus benutzt, der es dem Besitzer ermöglicht, alle Pseudonyme nachträglich erneut zu generieren. Dies ermöglicht es dem Benutzer die Nachverfolgung des Beacons, ohne dass Dritte dies auch tun könnten. Zum anderen wurde das „Mix-Zone“-Konzept benutzt, bei dem die Identität mehrere Nutzer in einem Gebiet vermischt wird, um eine Nachverfolgung zu erschweren. Dies wurde mit Hilfe einer „Funkstille“ umgesetzt, bei der das Beacon keine BLE-Signale aussendet und dadurch für Beobachter unsichtbar wird. Nach einer bestimmten Zeit werden die Übertragungen wieder mit einem neuem Pseudonym aufgenommen. Wenn mehrere Beacons in der Umgebung sind, kann ein Beobachter diese nach einem Pseudonymwechsel nicht mehr ohne Weiteres unterscheiden, wodurch eine Verfolgung deutlich erschwert wird.

Dieser Entwurf wurde auf mehreren Plattformen implementiert. Das Beacon wurde auf drei Geräteklassen umgesetzt 1. Das Faros Beacon. 2. Dem nRF51 SoC. 3. Der Android Plattform in der Form einer App. Der Beobachter wurde als Zusatzfunktion in der Android Beacon App implementiert. Dieser Beobachter benutzt die BLE-API von Android, um nach Beacons in der Umgebung zu suchen. Diese werden über eine REST Schnittstelle an den Server geschickt, welcher die Daten in einer Datenbank speichert und es dem Besitzer über die gleiche REST-API ermöglicht, die Position eines Beacons abzufragen. Diese Abfrage ist in der Form eines Web-Klienten implementiert, der die Position eines Beacons verfolgen kann und diese auf einer Karte in einem üblichen Web-Browser darstellt.

Außerdem wurde der Enegieverbrauch der jeweiligen Beacon-Implementierungen analysiert. Bei dieser Evaluation wurde ein durchschnittlicher Stromverbrauch berechnet, mit dem sich auf einer Knopfzelle das Faros Beacon für ca. ein $\frac{3}{4}$ Jahr betreiben lassen würde. Bei der nRF51 Implementierung ist der Betrieb für ca. ein ganzes Jahr ohne einen Batterietausch möglich. Zusätzliche wurde die Effektivität der Schutzmechanismen evaluiert. Dies ergab, dass die Schutzmechanismen bei einer moderaten Beacon-Dichte effektiv sind.

Zukünftige Arbeiten

Der jetzige Web-Klient benutzt die API des Servers, um die Position der einzelnen Pseudonyme abzufragen. Wenn der Server nur die Daten der Beobachter erhält, kann er nicht feststellen, welche Pseudonyme zu welchem Beacon gehören, da diese zufällig verteilt sind. Sobald jedoch der Web-Klient die Positionen abfragt weiß der Server, welche Pseudonyme zu welchem Beacon gehören, da der Web-Klient nur die berechneten Pseudonyme des Beacons abfragt. Um dieses Problem zu lösen könnten zusätzliche nicht-existente Pseudonyme in die Anfragen integriert werden, wodurch der Server nicht mehr sicher feststellen könnte, welches Pseudonym das des gesuchten Beacons ist. Einen ähnlichen Ansatz verfolgt das Vuvuzela-Projekt [VLZZ15], bei dem ein Nachrichtensystem entworfen wurde, welches zusätzlichen Netzwerkverkehr zur Verhinderung der Nachverfolgbarkeit der Kommunikation zwischen den Teilnehmern generiert. Dieses Konzept könnte auf den Klienten angewendet werden, um den Server daran zu hindern, die richtigen Pseudonyme aus der Liste der angefragten Pseudonyme herauszufiltern.

Die bestehenden Beacon-Implementierungen benutzen eine Funkstille mit fester Länge. Dies hat den Nachteil, dass ein Angreifer diese Information benutzen kann um Pseudonyme miteinander zu verknüpfen. Wenn ein Beacon mit Pseudonym A verschwindet und ein neues Beacon nach der festen Länge der Funkstille mit Pseudonym B erscheint, ist es sehr wahrscheinlich, dass diese Pseudonyme zu dem gleichen Beacon gehören. Um dies zu verhindern könnte die Länge der Funkstille zufällig gewählt werden, allerdings wird hierfür eine gute Zufallsquelle benötigt. Typische Betriebssysteme bieten Möglichkeiten, echte Zufallszahlen zu generieren, zum Beispiel mit `/dev/urandom` auf Linux, allerdings ist dies mit den Mikroprozessoren, die von den Beacons benutzt werden, nicht direkt möglich. Eine Möglichkeit wäre das Auslesen nicht belegter GPIO-Ports. Diese würden mehr oder weniger zufällige Werte zurückliefern, allerdings müsste analysiert werden, ob die Qualität der damit erzeugten Zufallszahlen tatsächlich ausreichend wäre.

Die Energieaufnahme des nRF51 Beacons könnte durch weitere Optimierungen weiter verbessert werden. Die CPU unterstützt den sogenannten „System-Off“-Modus, bei dem das System deutlich weniger Energie (ca. $\frac{1}{2}$ des jetzigen Verbrauchs) benötigt. Dieser Modus benötigt spezielle Vorkehrungen, da standardmäßig der komplette Programmstatus verloren geht, da der RAM die Daten nicht speichert. Es ist möglich, bestimmte RAM-Blöcke so zu konfigurieren, dass deren Daten nicht verloren gehen, allerdings erhöht dies den Stromverbrauch. Alternativ könnte der Programmstatus in den persistenten Flash-Speicher geschrieben werden, allerdings

benötigt auch dieser Vorgang zusätzliche Energie. In einer zukünftigen Arbeit könnten diese Alternativen implementiert und in Bezug auf die Energieaufnahme evaluiert werden. Ein nicht unerheblicher Anteil des Energieverbrauchs ist der Generierung neuer Pseudonyme zuzuschreiben, welches bisher mit einer Software-Implementierung des AES-Algorithmus erzeugt wird. Der nRF51 besitzt einen AES-Coprozessor, der allerdings nur den einfacheren ECB-Betriebsmodus anstatt des CBC-Modus unterstützt. Mit einer Anpassung des Pseudonymalgorithmus könnte dieser Coprozessor benutzt werden, um die Energieaufnahme weiter zu verringern. Allerdings muss hierbei darauf geachtet werden, dass durch die Benutzung des ECB-Modus keine Sicherheitslücken entstehen, da dieser dafür bekannt ist, dass bei falscher Benutzung Rückschlüsse auf den Inhalt der Nachrichten gezogen werden können.

Literaturverzeichnis

[AEGS10] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, M. Schwamborn. „BonnMotion: a mobility scenario generation and analysis tool“. In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics und Telecommunications Engineering). 2010, S. 51 (zitiert auf S. 59).

[alt] altbeacon. *Android Beacon Library*. URL: https://altbeacon.github.io/android-beacon-library/ (zitiert auf S. 43).

[Atm] Atmel Corporation. *ATmega328P*. URL: http://www.atmel.com/devices/atmega328p.aspx (zitiert auf S. 21).

[BHV07] L. Buttyán, T. Holczer, I. Vajda. „On the effectiveness of changing pseudonyms to provide location privacy in VANETs“. In: *Security and Privacy in Ad-hoc and Sensor Networks*. Springer, 2007, S. 129–141 (zitiert auf S. 26).

[Blu14] Bluetooth Special Interest Group. *Bluetooth Core Specification 4.2*. 2. Dez. 2014 (zitiert auf S. 17).

[BS03] A. R. Beresford, F. Stajano. „Location privacy in pervasive computing“. In: *IEEE Pervasive computing* 2.1 (2003), S. 46–55 (zitiert auf S. 27).

[Cha08] S. S. Chawathe. „Beacon placement for indoor localization using bluetooth“. In: *2008 11th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2008, S. 980–985 (zitiert auf S. 18, 26).

[DR02] J. Daemen, V. Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002, S. 238. ISBN: 3-540-42580-2 (zitiert auf S. 22).

[Dür16a] F. Dürr. *Faros*. 2016. URL: https://github.com/duerrfk/Faros (zitiert auf S. 21).

[Dür16b] F. Dürr. *Low-Energy-Meter*. 2016. URL: https://github.com/duerrfk/low-energy-meter (zitiert auf S. 51).

[Fie00] R. Fielding. „Fielding dissertation: Chapter 5: Representational state transfer (rest)“. In: *Recuperado el* 8 (2000) (zitiert auf S. 46).

[GG07] M. Gerlach, F. Guttler. „Privacy in VANETs using changing pseudonyms-ideal and real“. In: *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. IEEE. 2007, S. 2521–2525 (zitiert auf S. 26, 34, 38).

[Goo] Google. *Nearby API*. Google. URL: https://developers.google.com/nearby/ (zitiert auf S. 19).

[Goo16] Google. *Eddystone*. Google, 2016. URL: https://github.com/google/eddystone (zitiert auf S. 19).

[GOP12] C. Gomez, J. Oller, J. Paradells. „Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology“. In: *Sensors* 12.9 (2012), S. 11734–11753 (zitiert auf S. 31).

[HMYZ16] A. Hassidim, Y. Matias, M. Yung, A. Ziv. „Ephemeral Identifiers: Mitigating Tracking & Spoofing Threats to BLE Beacons“. In: (2016) (zitiert auf S. 20).

[HW03] P. Hellekalek, S. Wegenkittl. „Empirical evidence concerning AES“. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 13.4 (2003), S. 322–333 (zitiert auf S. 36, 41).

[Jos06] S. Josefsson. „The base16, base32, and base64 data encodings (rfc 4648)“. In: *Network Working Group* (2006) (zitiert auf S. 44).

[kok] kokke. *tiny-AES128-C*. URL: https://github.com/kokke/tiny-AES128-C (zitiert auf S. 42).

[Lan] D. Landman. *AESLib*. URL: https://github.com/DavyLandman/AESLib (zitiert auf S. 42).

[MN98] M. Matsumoto, T. Nishimura. „Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator“. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), S. 3–30 (zitiert auf S. 36).

[MVV96] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone. *Handbook of applied cryptography. Pseudorandom Bits and Sequences*. CRC press, 1996. Kap. 5 (zitiert auf S. 36).

[Nora] Nordic Semiconductor. *nRf51 Series SoC*. Nordic Semiconductor. URL: https://www.nordicsemi.com/eng/Products/nRF51-Series-SoC (zitiert auf S. 21).

[Norb] Nordic Semiconductor. *nRF8001*. Nordic Semiconductor. URL: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF8001 (zitiert auf S. 21).

[SEG] SEGGER. *Debug Probes - J-Link and J-Trace*. SEGGER. URL: https://www.segger.com/jlink-debug-probes.html (zitiert auf S. 22).

[Sta13] Statistisches Bundesamt Deutschland. *Oberlandes-, Landes- und Amtsgerichtsbezirke nach Fläche, Bevölkerung und Bevölkerungs dichte am 31.12.2013*. 2013. URL: https://www.destatis.de/DE/ZahlenFakten/LaenderRegionen/Regionales/Gemeindeverzeichnis/Administrativ/Archiv/Standardtabellen/12_GerichteVorjahr.html (zitiert auf S. 60).

[VLZZ15] J. Van Den Hooff, D. Lazar, M. Zaharia, N. Zeldovich. „Vuvuzela: Scalable private messaging resistant to traffic analysis“. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM. 2015, S. 137–152 (zitiert auf S. 62).

[War14] A. Warski. *How do iBeacons work?* 13. Jan. 2014. URL: http://www.warski.org/blog/2014/01/how-ibeacons-work/ (zitiert auf S. 19).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift