TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Fakultät für Mathematik und Informatik
Institut für Informatik
Lehrstuhl für Betriebssysteme und Kommunikationstechnologien

Bakkalaureatsarbeit

Entwicklung einer Sicherheitsinfrastruktur zur Bluetooth-Kommunikation zwischen Smartphone und Mikrocontroller

Marian Käsemodel

Angewandte Informatik
Vertiefung: Technik

Matrikel: 62 412

7. Juli 2021

Betreuer/1. Korrektor:
Prof. Dr. Konrad Froitzheim

2. Korrektor:
M.Sc. Jonas Treumer

Eidesstattliche Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Versicherung bezieht sich auch auf die bildlichen Darstellungen.

7. Juli 2021 Marian Käsemodel

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Bluetooth ist eine weit verbreitete Kommunikationstechnologie, die von vielen Geräten wie Smartphones, Laptops und eingebetteten Systemen unterstützt wird. Seit 2010 unterteilt sich *Bluetooth* in die Varianten *Bluetooth Classic* und *Bluetooth Low Energy*. Zweiteres ist eine geeignete Alternative für Geräte, die einen niedrigen Energieverbrauch anstreben.

Viele Infrastrukturen, die auf *Bluetooth* basieren, transportieren sensible Daten. Bekannte Anwendungen sind Smartwatches, die mit Smartphones kommunizieren, oder die Vernetzung von eingebetteten Systemen in der Industrie. Ein weiterer Anwendungsfall sind autonome Verleihdienste für Kleinfahrzeuge innerhalb von Städten. So können Personen Fahrräder oder Roller ausleihen, um flexibel am Individualverkehr teilzunehmen. Einige dieser Dienste verwenden dabei *Bluetooth*, um zwischen dem Fahrzeug und dem Smartphone des Nutzers Daten zu übertragen. Für diese Anwendungen, stellt sich die Frage, welche Sicherheiten *Bluetooth* bietet.

Hintergrund dieser Arbeit ist das Projekt *SteigtUM*, das einen solchen autonomen Verleih umsetzen soll. Es unterscheidet sich von anderen Verleihdiensten durch die Nutzung von Lastenfahrrädern. Dabei soll ein neuer Anwendungsfall berücksichtigt werden, der es dem Nutzer ermöglicht, das Fahrrad abzustellen und zu einem späteren Zeitpunkt wiederzuverwenden. Auf diese Weise kann der Nutzer bspw. zu einem Geschäft gelangen, Einkäufe tätigen und diese anschließend nach Hause transportieren. Im Rahmen dieser Arbeit wird die Idee verfolgt, dass das Smartphone des Nutzers mit dem Mikrocontroller des Fahrzeugs über *Bluetooth Low Energy* kommuniziert.

1.1 Problemstellung

Ziel der Arbeit ist zunächst die Untersuchung der Schwachstellen innerhalb von *Bluetooth Low Energy*. Dazu muss ein tieferes Verständnis für dessen Funktionsweise und die zur Verfügung gestellten Sicherheitsfunktionen gewonnen werden. Auf diesem Wissen aufbauend wird eine Infrastruktur entworfen, durch die ein Smartphone und ein Mikrocontroller mittels *Bluetooth Low Energy* sicher miteinander kommunizieren können. Weder bei der Übertragung noch bei der Verarbeitung innerhalb des Systems soll eine außenstehende Instanz auf die übertragenen Nachrichten zugreifen können. Demnach ist eine Lösung für die Probleme der Authentizität, Datenintegrität und Vertraulichkeit erforderlich. Im Idealfall ist die Infrastruktur nicht nur auf die Geräte Smartphone und Mikrocontroller beschränkt, sondern wäre für jede Art von Bluetooth-Gerät denkbar.

Desweiteren ist es Ziel der Arbeit, mithilfe dieser Infrastruktur eine Implementierung umzusetzen, die sich auf das Projekt *SteigtUM* bezieht. Da bei dem vorgestellten Verleihdienst sensible Daten zwischen Smartphone und Fahrzeug ausgetauscht werden, wird die Infrastruktur als Grundlage für die sichere Kommunikation genutzt. Dennoch sind weitere Lösungen gefordert. Nur der Nutzer darf Zugriff auf die Funktionen des ausgeliehenen Fahrzeugs erlangen. Demnach muss zu Beginn eines Ausleihprozesses sichergestellt werden, dass der Nutzer berechtigt ist, das Fahrzeug auszuleihen. Verlässt er das Fahrzeug und möchte es nach einer bestimmten Zeit wieder nutzen, soll kein anderer in der Lage sein, das Fahrzeug in der Zwischenzeit zu entsperren und zu entwenden.

2 Grundlagen zu Bluetooth Low Energy

2.1 Überblick

Bluetooth ist ein Industriestandard für die Übertragung von Daten per Funk, dessen Intention die Reduzierung von Kabelverbindungen an mobilen sowie stationären Geräten ist. Wichtige Eigenschaften der Technologie sind vor allem ein niedriger Energieverbrauch und günstig herstellbare Hardware. Seit 1998 wird Bluetooth von der *Bluetooth Special Interest Group* (SIG) entwickelt und ist seit 2002 von der *Organisation Institute of Electrical and Electronics Engineers* (IEEE) standardisiert [1].

Es agiert im lizenzfreien ISM-Band (Industrial, Scientific and Medical Band) von 2,4 GHz. Zur Reichweite kann keine genaue Aussage getroffen werden, da sie von vielen Parametern wie beispielsweise der Sendeleistung und Einflüssen aus der Umwelt abhängt. Um trotzdem einen Eindruck zu gewinnen, kann für bestimmte Bedingungen und Konfigurationen die maximale Reichweite mithilfe eines Tools [2] der SIG ermittelt werden. Dabei variieren die Ergebnisse von ca. einem Meter bis hin zu mehr als 1000 Metern.

Grundlegend wird Bluetooth seit der Version 4.0 von 2010 in die zwei Systeme *Basic Rate* (BR) und *Low Energy* (LE) unterteilt, wobei LE darauf ausgelegt ist, weniger Energie als BR zu benötigen. Die neueste Bluetooth-Version ist die Version 5.2, die wie jede Version abwärtskompatibel ist. Jedoch sind beide Systeme (BR und LE) bezüglich der Kommunikation miteinander inkompatibel: implementiert ein Gerät nur das BR-System, kann es keine Daten mit einem Gerät austauschen, das nur das LE-System unterstützt. Demnach ist für LE die Abwärtskompatibilität nur bis zur Version 4.0 gegeben. Desweiteren ist es möglich, dass ein Gerät über beide Systeme verfügt und so die meisten Nutzungsfälle abdeckt. Das BR-System kann mit den Erweiterungen *Enhanced Data Rate* (EDR) und *Alternate Media Access Control and Physical Layer* (AMP) genutzt werden, um eine höhere Datenrate zu erzielen. Die einzelnen Systeme und Erweiterungen können entsprechend ihrer Bluetooth-Version die in Tabelle 1 dargestellten Datenraten erreichen.

System/Erweiterung	**max. Bitrate (Version 4.0)**	**max. Bitrate (Version 5.2)**
BR	1 Mbit/s [3]	1 Mbit/s [4]
BR/EDR	2 Mbit/s bis 3 Mb/s [3]	2 Mbit/s bis 3Mb/s [4]
802.11 AMP	24 Mbit/s [5]	52 Mbit/s [6]
LE	1 Mbit/s [7]	2 Mbit/s [8]

Tab. 1: Maximale Bitraten der Bluetooth-Systeme

Da Bluetooth Low Energy (BLE) ein zentraler Bestandteil dieser Arbeit ist, bezieht sich der Autor von nun an nur darauf und nicht mehr auf Bluetooth im Allgemeinen. D.h., dass Bluetooth Classic, welches BR/EDR und AMP beschreibt, nur noch behandelt wird, wenn das BR-System oder eine seiner Erweiterungen explizit erwähnt werden.

Die Architektur eines Bluetooth-Systems unterteilt sich in einen Host und in einen oder mehrere Controller. Ein Host ist eine logische Entität, definiert als alle Schichten unterhalb der nicht zu Bluetooth gehörigen Profile (Protokolle) und oberhalb des Host-Controller-Interface (HCI). Ein Controller ist eine logische Entität, definiert als alle Schichten bzw. Funktionsblöcke unterhalb des HCI. Der Aufbau setzt sich immer aus genau einem primären Control-

ler und optional aus sekundären Controllern zusammen. Dabei kann die Rolle des primären Controllers entweder durch einen BR/EDR-Controller, einen LE-Controller oder durch eine Kombination aus BR/EDR- und LE-Controller eingenommen werden, während die Rolle eines sekundären Controllers nur durch einen AMP-Controller besetzt werden kann. In Abb. 1 sind einige Varianten skizziert.

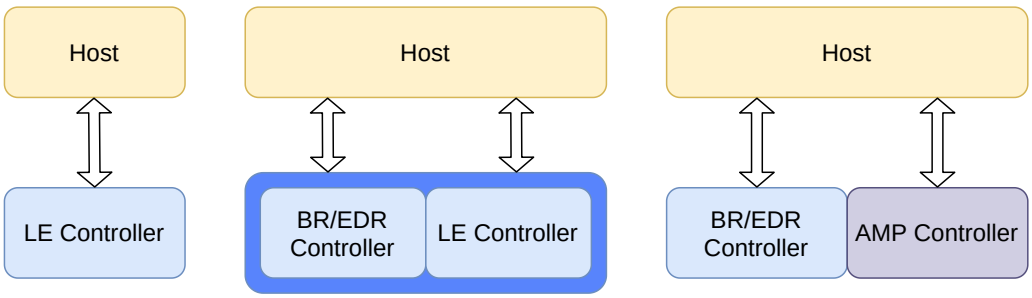

Abb. 1: Kombinationen aus Host und Controller; in Anlehnung an [9]

Zur Veranschaulichung der Architektur bezüglich eines LE-Systems ist in der Abb. 2 die Zusammensetzung aus Host und Controller mit deren Schichten bzw. Protokollen festgehalten, die in den Sektionen 2.4 und 2.5 thematisiert werden. Über dem Host befindet sich die Anwendungsschicht.

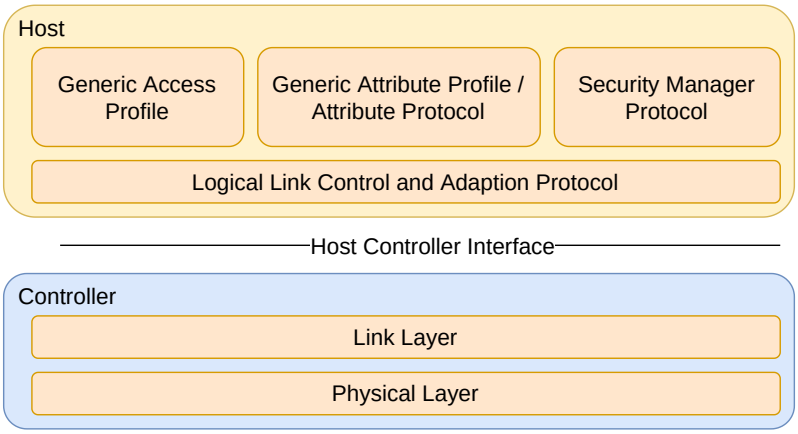

Abb. 2: BLE-Architektur von Host und Controller; in Anlehnung an [10]

2.2 Topologie

Ein Ad-Hoc-Netzwerk bestehend aus Bluetooth-Geräten wird Piconet genannt. Dabei existiert in einem Piconet immer ein Master-Gerät, das sich mit einem oder mehreren Slave-Geräten verbinden kann. Ein Gerät kann zur selben Zeit innerhalb mehrerer Piconets agieren, wobei die Rollen unabhängig sind. Z.B. könnte ein Gerät A der Master eines Piconet A sein, während es ein Slave in einem Piconet B ist (siehe Abb. 3). [11]

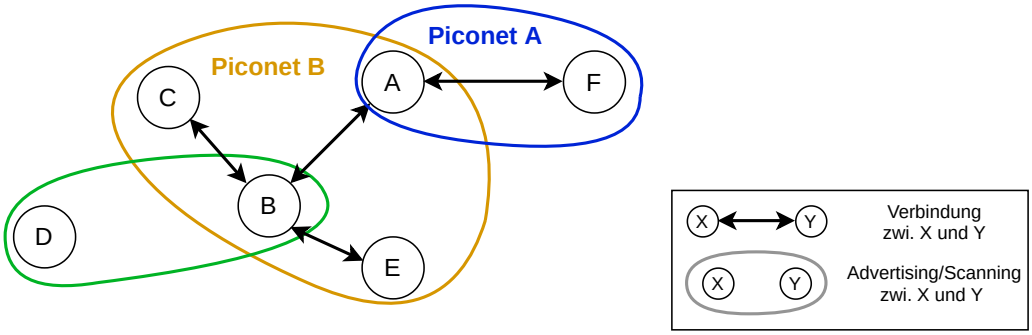

Abb. 3: Topologie in LE

2.3 Verbindungsaufbau

Um mittels BLE ein Piconet zu bilden, ist ein Advertiser notwendig. Auf drei vorgegebenen Frequenzen, den Advertising Channels (siehe Sektion 2.4.1.1), sendet er Daten, mit denen er sich für andere Geräte bemerkbar macht (Advertisements). Dabei können Advertisements auch genutzt werden, um Nutzdaten zu senden. Jedes Advertisement-Paket beinhaltet eine Bluetooth-Adresse des Senders, die 48 Bit lang ist.

Geräte, die Daten auf den Advertising Channels empfangen, werden Scanner bzw. Initiator genannt. Auf diesem Weg finden sich die Geräte (Discovering). Der Initiator unterscheidet sich vom Scanner, da er in der Lage ist, sich zu einem Advertiser zu verbinden, von dem er ein Advertisement erhalten hat, das eine Verbindung ermöglicht. Sind zwei Geräte verbunden, senden und empfangen sie ihre Pakete auf den Data Channels (siehe Sektion 2.4.1.1). Verbinden sich zwei Geräte, wird der Initiator als Master und der Advertiser als Slave angesehen.

Durch die Anwendung von Zeitmultiplexing senden die Geräte ihre Pakete immer zu festgelegten Zeitpunkten. Dabei ist ein Event ein zeitlicher Abschnitt, in dem zusammenhängende Daten in Form von Paketen gesendet bzw. empfangen werden.

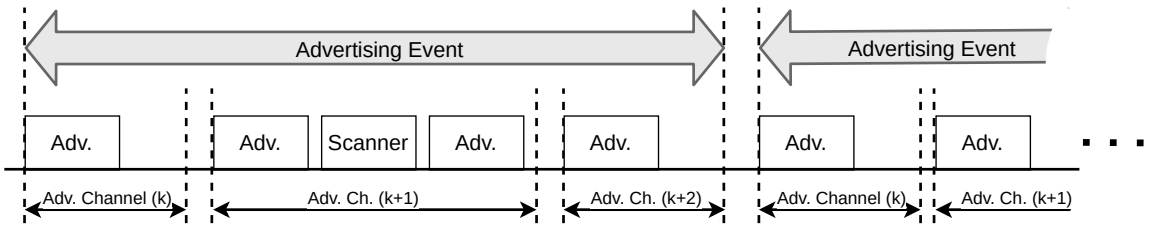

Abb. 4: Advertising Event [12]

In Abb. 4 ist ein Advertising Event dargestellt, bei dem ein Advertiser auf allen drei Advertising Channels nacheinander Advertisement-Pakete sendet. Auf dem zweiten Kanal empfängt der Advertiser -direkt im Anschluss auf sein erstes Advertisement-Paket- in diesem Kanal ein Paket eines Scanners, auf das er mit einem weiteren Advertisement antwortet.

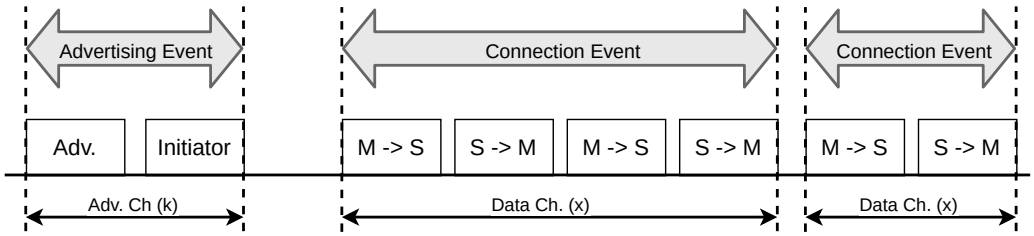

Abb. 5: Advertising und Connection Event [12]

In Abbildung 5 ist ein Advertising Event dargestellt, bei dem ein Initiator auf das Advertisement-Paket eines Advertiser antwortet, um eine Verbindung aufzubauen. Darauf folgt ein Connection Event, bei dem Master (ursprünglich Initiator) und Slave (ursprünglich Advertiser) einander Pakete auf einem Data Channel senden. Danach folgt ein weiteres Connection Event auf einem anderen Data Channel.

2.4 Controller

Wie in Abbildung 6 zu sehen ist, umfasst der Controller die Schichten Physical Layer (PHY) und Link Layer (LL). Über dem Controller ist der Logical Link Control And Adaption Protocol (L2CAP) Layer in grau dargestellt, da er nicht Teil des Controllers sondern des Hosts ist.

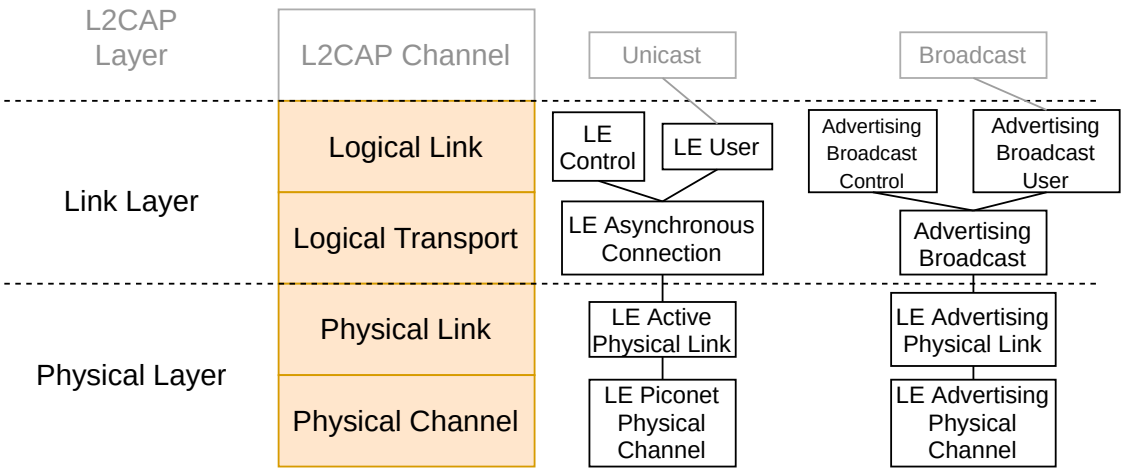

Abb. 6: Architektur des Physical Layers und des Link Layers; in Anlehnung an [13] und [14]

Der Physical Layer unterteilt sich weiter in die Physical Channels und die Physical Links, während der Link Layer sich in Logical Transports und Logical Links aufteilt. In den Schichten bzw. ihren Untergliederungen wird definiert, wie Anwendungsdaten und Kontrollsignale innerhalb Verbindungen bzw. Advertisements übertragen werden. Bezüglich des L2CAP Layers werden Anwendungsdaten entweder in Form eines Unicasts oder Broadcasts übertragen.

2.4.1 Physical Layer

2.4.1.1 Physical Channel

Um miteinander zu kommunizieren, müssen zwei Bluetooth-Geräte (ein Sender und ein Empfänger) zur selben Zeit den selben Kanal nutzen, wobei sich der Empfänger in der Reichweite des Senders befinden muss. Da mehrere Piconets zur selben Zeit im selben lokalen Bereich agieren können, besteht die Wahrscheinlichkeit, dass zwei Sender zweier verschiedener Gerätepaare in Reichweite den selben Kanal zur selben Zeit nutzen und eine Kollision verursachen.

Mittles des Frequenzmultiplexverfahrens ist das ISM-Band eines LE-Systems von 2400 MHz bis 2483,5 MHz in 40 Funkkanäle unterteilt. Beginnend bei 2402 MHz nutzt jeder Kanal eine Frequenz, die 2 MHz über der Frequenz des Vorgängers liegt. Das Trägersignal wird mithilfe des Gaussian Frequency Shift Keying moduliert. [15]

Somit bildet der Physical Channel die niedrigste Ebene der Architektur. 37 der 40 Kanäle werden als LE Piconet Channel (entsprechend S. 11 Abb. 6 als LE Piconet Physical Channel) bezeichnet, die mit einem Piconet assoziiert werden und zur Kommunikation zwischen zwei bereits verbundenen Geräten dienen. Die verbleibenden drei Kanäle werden Advertisement Broadcast Channel (entsprechend S. 11 Abb. 6 als LE Advertising Physical Channel) genannt und befinden sich auf den Frequenzen 2402 MHz, 2426 MHz sowie 2480 MHz. [16]

Mittels Advertisements können Geräte in diesen drei Kanälen auf sich aufmerksam machen, um von anderen Geräten entdeckt zu werden. Zudem werden sie genutzt, um Geräte miteinander zu verbinden oder Anwendungsdaten an Scanner bzw. Initiatoren zu senden. Ein Gerät kann nur einen Kanal zur selben Zeit nutzen, weswegen das Zeitmultiplexverfahren verwendet wird, das bereits verbundenen Geräten ermöglicht, zusätzlich das Advertisement zu betreiben.

Um Interferenzen innerhalb des genutzten Frequenzbands z.B. mit *Wi-Fi* zu vermeiden, wird das Adaptive Frequency Hopping [17] genutzt (eine Form des Frequency Hopping Spread Spectrum). Dabei wechseln Sender und Empfänger in kurzen Zeitabständen den Kanal und passen die Menge der zu nutzenden Kanäle (Channel Map) an, indem sie dynmaisch ermittleln, in welchen Kanälen häufiger Kollisionen auftreten. Treten in einem Kanal häufig Interferenzen auf, wird er für eine bestimmte Zeitspanne aus der Channel Map entfernt und vorerst nicht mehr genutzt.

2.4.1.2 Physical Link

Ein Physical Link wird immer mit genau einem Physical Channel assoziiert. Dagegen kann ein Physical Channel mehrere Physical Links unterstützen. Bezüglich Bluetooth wird der Physical Link nicht in der Struktur eines Paketes repräsentiert, kann aber innerhalb eines LE-Paketes anhand der Access Address identifiziert werden. [18]

Active Physical Links sind die Punkt-zu-Punkt-Verbindungen zwischen Master und Slave über einen Piconet Physical Channel und gelten nur als aktiv, wenn ein Asynchronous Connection (ACL) Logical Transport zwischen den Geräten existiert. [19]

Advertising Physical Links dienen dazu, zwischen einem Advertiser und einem Initiator einen Active Physical Link aufzubauen, und existieren nur für einen kurzen Zeitraum. Zwischen Advertiser und Scanner existieren sie für längere Zeit und dienen dem Broadcast von Nutzdaten.

[19]

2.4.2 Link Layer

2.4.2.1 Paketstruktur

Der Link Layer nutzt ein gemeinsames Paketformat für das Übertragen von Advertising-Paketen und Nutzdaten-Pakten, das in Abb. 7 dargestellt ist.

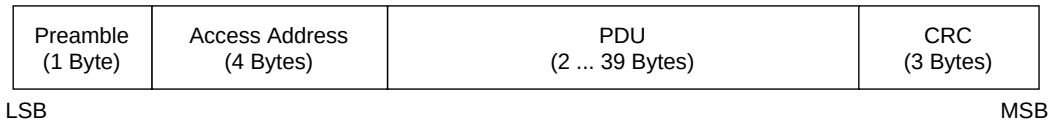

Abb. 7: Paketstruktur des Link Layers; in Anlehnung an [20]

Die Preamble hat eine Größe von acht Bit und wird genutzt, um auf Empfängerseite die Frequenz zu synchronisieren, die Zeiteinteilung der Symbole zu schätzen und um die Automatic Gain Control zu trainieren. Die Preamble beträgt immer 0b01010101, falls das Bit mit dem niedrigsten Stellenwert (LSB für Least Significant Bit) der Access Address 1 ist. Anderenfalls beträgt die Preamble 0b10101010. [21]

Die Access Address hat eine Größe von 32 Bit und identifziert eine Verbindung über den Link Layer bzw. dient dazu, Pakete mittels des festgelegten Wertes 0x8E89BED6 als Advertisement-Pakete zu identifzieren. Bevor ein Initiator eine Verbindung zu einem Advertiser aufbaut, erstellt er eine zufällige Access Address, die neben weiteren Bedingungen nicht der des Advertisement-Pakets gleichen oder sich nur um ein Bit von ihr unterscheiden darf. Diese Access Address sendet er dann innerhalb der Verbindungsanfrage an den Advertiser. [21]

Das letzte Feld des Link-Layer-Pakets ist der 24 Bit lange Cyclic Redundancy Check (CRC), der über das PDU-Feld berechnet wird. Im Fall, dass auf Ebene des Link Layers die PDU verschlüsselt wird, generiert man den CRC erst nach der Verschlüsselung. Unabhängig davon, ob die Verschlüsselung aktiv ist, wird anschließend ein Whitening [22] durchgeführt, um Sequenzen vieler gleichbleibender Bit (bspw. 0b00000000) zu verhindern. [21]

Die Protocol Data Unit (PDU) unterteilt sich in die Advertising Channel PDU und Data Channel PDU.

Advertising Channel PDU

Wie in Abb. 8 gezeigt wird, besteht die Advertising Channel PDU aus einem 16 Bit langen Header und einem Payload variabler Länge. RFU steht dabei für Reserved for Future Use.

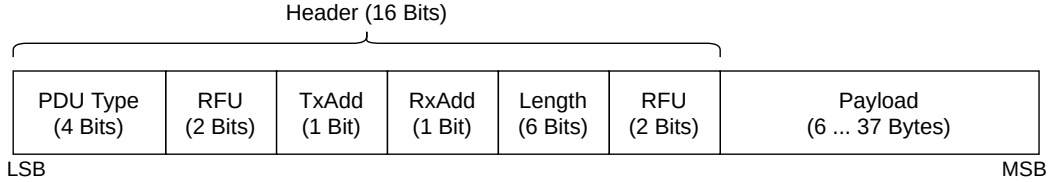

Abb. 8: Link Layer Advertising Channel PDU; in Anlehnung an [23] und [24]

Dabei beinhaltet der Header unter anderem ein 4 Bit langes Feld für den PDU Type (bspw. Connectable Undirected Advertising Event oder Scan Request) und zwei Flags TxAdd und RxAdd für zusätzliche Informationen bezüglich des PDU Type. Die Bedeutungen von TxAdd und RxAdd hängen vom PDU Type ab. Die Menge aller PDU Types lässt sich folgendermaßen untergliedern:

- Advertising PDU
- Scanning PDU
- Initiating PDU

Bei allen bilden die ersten 6 Byte des Payload die Adresse des Senders (Advertiser, Scanner oder Initiator). Hier sagt TxAdd bei jedem PDU Type aus, ob die angegebene Adresse des Senders öffentlich (TxAdd = 0) oder zufällig generiert (TxAdd = 1) ist. Diese Funktion wird Privacy Feature gennant und ist in Sektion 2.5.3.1 genauer beschrieben. RxAdd dagegen ist nur bei PDU Types von Bedeutung, die in ihrem Payload eine zweite Adresse enthalten, nämlich die des Empfängers. Analog zu TxAdd sagt RxAdd aus, ob die Adresse des Empfängers öffentlich (RxAdd = 0) oder zufällig (RxAdd = 1) ist.

Ein weiteres Feld im Header der Advertising Channel PDU ist das 6 Bit lange Feld für die Länge des Payloads in Byte, dessen Wert eine Spanne von 6 bis 37 Byte abdeckt. [25]

Data Channel PDU

Die Data Channel PDU nutzt entsprechend der Abb. 9 einen 16 Bit langen Header, einen Payload variabler Länge und optional einen 32 Bit langen Message Integry Check (MIC), der die Integrität des Payload sicherstellt. Das MIC-Feld entfällt bei einer unverschlüsselten Link-Layer-Verbindung und bei einer Data Channel PDU, deren Payload leer ist.

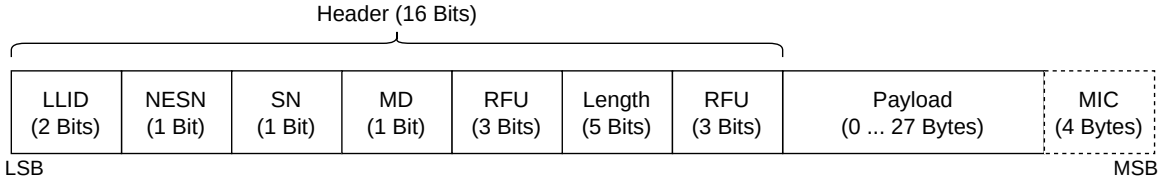

Abb. 9: Link Layer Data Channel PDU; in Anlehnung an [26] und [27]

Das erste Feld des Headers ist der zwei Bit lange Link Layer Identifier (LLID), der mit 0b01 sowie 0b10 aussagt, dass es sich um eine LL Data PDU handelt und mit 0b11, dass es sich um eine LL Control PDU handelt. Der Wert 0b00 ist reserviert. Die LL Control PDU dient dazu, die LL-Verbindung zu steuern. Dazu gehören unter anderem Anfragen zum Ändern der Verbindungsparameter (z.B. Window Size oder Timeout), zum Ändern der Channel Map oder zum Verschlüsseln.

Auf die LLID folgt das Feld der Next Expected Sequence Number (NESN) und das Feld der Sequence Number (SN) mit jeweils einem Bit Länge, die innerhalb Sektion 2.4.2.2 näher erläutert werden.

Unter anderem beinhaltet der Header ein fünf Bit langes Feld für die Länge des Payload in Byte und ggf. einschließlich der Länge des MIC. Der maximale Wert der Länge beträgt 31 Byte, wobei sich der Payload in jedem Fall auf eine maximale Länge von 27 Byte bemisst. [28]

In der Bluetooth-Version 4.2 ist das Feld der Länge auf acht Bit erweitert, wodurch der Payload eine maximale Größe von 251 Byte annehmen kann. [29]

2.4.2.2 Logical Transport

Über dem Physical Layer ist der Link Layer angesiedelt, beginnend mit dem Logical Transport, der in die zwei Betriebsmodi LE Asynchronous Connection (LE ACL) und LE Advertising Broadcast (ADVB) unterteilt.

Die LE ACL transportiert Kontrollsignale des über ihr befindlichen Logical Link und Logical Link Control and Adaption Protocol (L2CAP). Außerdem überträgt die LE ACL asynchrone Anwendungsdaten nach dem Best-Effort-Prinzip.

Mithilfe der Next Expected Sequence Number bzw. Sequence Number (NESN/SN), die jeweils nur die Größe eines Bits besitzen, wird eine einfache Zuverlässigkeit gewährleistet. Empfängt ein Gerät ein Paket B, vergleicht es dessen NESN mit der SN, die es innerhalb des vorherigen Pakets A abgesendet hat. Wenn diese unterschiedlich sind, wurde das vorherige Paket A vom Gegenüber vollständig und korrekt empfangen (ACK für Acknowledgement). Anderenfalls sind die Nummern gleich, was bedeutet, dass das vorherige Paket A nicht vollständig bzw. korrekt empfangen wurde (NACK/NAK für Negative Acknowledgement) und nun erneut an das Gegenüber gesendet werden muss. Zudem prüft das Gerät die SN des empfangenen Pakets B mit der NESN seines vorher gesendeten Pakets A. Sind diese Nummern gleich, wurde das empfangene Paket B vom Gerät erwartet. Anderenfalls sind die Nummern verschieden und somit wurde das Paket B nicht erwartet, weswegen es ignoriert wird. Ein ausführliches Beispiel ist im Anhang 6.1 (S. 44) angegeben. Folglich wird auch die Flusskontrolle ermöglicht, da ein Empfänger bei nicht ausreichend freiem Speicher im Buffer ein NACK mithilfe der Sequenznummern zurücksenden kann. [30]

Wenn ein Gerät einem Piconet beitritt, wird zwischem dem Master und dem Slave eine Default LE ACL über einen Active Physical Link gebildet. Die Default LE ACL ist einer Access Address zugeordnet. Wird die Default LE ACL getrennt, werden alle Logical Transports zwischen Master und Slave ebenfalls unterbrochen. Bei einem unerwarteten Synchronisationsverlust zum LE Piconet Physical Channel werden der LE Physical Link und alle LE Logical Transports und LE Logical Links unterbrochen.

Der ADVB transportiert ohne Verwendung von Acknowledgements Kontrollsignale und Anwendungsdaten bezüglich des Broadcasts über den darunter gelegenen LE Advertising Broadcast Link. Der Datenverkehr ist überwiegend unidirektional, ausgehend vom Advertiser zu allen in Reichweite befindlichen Scannern. Scanner können eine Anfrage an den Advertiser senden, um weitere Anwendungsdaten über den Broadcast zu empfangen. Initiator können eine Anfrage an den Advertiser senden, um eine LE ACL zu bilden. Aufgrund des Verzichts auf Acknowledgements ist der ADVB unzuverlässig, weswegen Pakete redundant übertragen werden. Sobald ein Gerät mit dem Advertising beginnt, wird ein ADVB erzeugt, der anhand der Adresse des Gerätes identifiziert wird. [31]

2.4.2.3 Logical Link

Ein Logical Link unterscheidet sich abhängig davon, ob er auf einem LE ACL Logical Transport oder einem ADVB Logical Transport aufbaut und ob er zur Übertragung von Kontrollsignalen oder Anwendungsdaten genutzt wird. Anhand des Logical Link Identifier (LLID) im Header des Basisbandpakets (siehe S. 14 Abb. 9) wird unterschieden, ob es sich bei der zu übertragenden PDU um Anwendungsdaten oder Kontrollsignale handelt.

Der Control Logical Link (LE-C) nutzt den darunter liegenden LE ACL Logical Transport, um Kontrollsignale zwischen Geräten im Piconet zu übertragen.

Der User Asynchronous Logical Link (LE-U) nutzt den darunter liegenden LE ACL Logical Transport, um alle asynchronen Anwendungsdaten zu übertragen. Über dem Link Layer agiert das Protokoll L2CAP, dessen Frames für den Link Layer fragmentiert werden müssen. Mithilfe des LLID-Wertes 0b10 wird der Beginn eines L2CAP-Frames (das erste Fragment eines L2CAP-Frame) und der Wert 0b01 die Fortsetzung eines L2CAP-Frames (die folgenden Fragmente des L2CAP-Frame) gekennzeichnet. Somit wird der Header des Protokolls L2CAP einfach gehalten und eine korrekte Synchronisation bei der Zusammensetzung der Fragmente zu einem L2CAP-Frame garantiert. Folglich muss ein L2CAP-Frame zunächst vollständig übertragen werden, bevor ein neuer Frame begonnen werden kann. [32]

Der Advertising Broadcast Control Logical Link (ADVB-C) nutzt den darunter liegenden Default ADVB, um Kontrollsignale für Verbindungsanfragen oder Anfragen für weitere Broadcast-Anwendungsdaten zu übertragen.

Der Advertising Broadcast User Data Logical Link (ADVB-U) nutzt den darunter liegenden Default ADVB, um verbindungslos und ohne den Gebrauch von LE-U Anwendungsdaten als Broadcast zu senden. [32]

2.5 Host

Im Wesentlichen umfasst der Host das Logical Link Control and Adaption Protocol (L2CAP), das Generic Access Profile (GAP) und Generic Attribute Profile (GATT) sowie das Security Manager Protocol (SMP). Er ist unter der Anwendungsebene angesiedelt und steuert den Datentransport als auch den Verbindungsaufbau sowie Sicherheitsaspekte. Je nachdem wie Host und Controller implementiert sind, kann das standarisierte Host Controller Interface (HCI) als Schnittstelle zum Datenaustausch zwischen Host und Controller dienen. Es ist optional [33], da es Implementierungen bestehend aus Host und Controller gibt, die direkt verbunden sind, wodurch ein HCI nicht nötig ist. Da das HCI den allgemeinen Kommunikationsprozess zwischen Bluetooth-Geräten nicht nennenswert beeinflusst, wird es nicht weiter behandelt.

2.5.1 Logical Link Control and Adaption Protocol

Das Logical Link Control and Adaption Protocol (L2CAP) bildet die unterste Schicht im Host (siehe S. 9 Abb. 2) und dient je nach Konfiguration dazu, den Datenverkehr zu steuern und zwischen höheren und niedrigeren Schichten zu vermitteln. Mittels Multiplexing können mehrere Anwendungen einen LE-U (also Logical Link) nutzen. Es verfügt über fünf Modi:

- Basic L2CAP Mode
- Flow Control Mode

- Retransmission Mode
- Enhanced Retransmission Mode
- Streaming Mode
- LE Credit Based Flow Control Mode (seit Bluetooth 4.2)

Für Bluetooth allgemein (BR/EDR und LE) wird immer der Basic L2CAP Mode genutzt, wenn kein anderer festgelegt wird. Der LE Credit Based Flow Control Mode soll [34] zufolge als einziger Modus für verbindungsorientierte LE-Kanäle genutzt werden („This is the only mode that shall be used for LE L2CAP connection oriented channels“ [34]). Da diese Aussage nicht ausschließt, dass eine verbindungsorientierte LE-Verbindung über den standardmäßig festgelegten Basic L2CAP Mode erfolgen kann, und der LE Credit Based Flow Control Mode noch nicht in der Bluetooth-Version 4.0 vertreten war [35], ist anzunehmen, dass eine verbindungsorientierte LE-Verbindung auch mit dem Basic L2CAP Mode möglich ist.

Logische Kanäle, genannt L2CAP Channels, dienen innerhalb eines Geräts als Endpunkt für höher gelegene Protokolle oder direkt für die Anwendung und sind für jedes Gerät individuell an dem Channel Identifier (CID) unterscheidbar. Folglich muss der L2CAP Channel einer Verbindung zwischen zwei Geräten von diesen nicht zwingend mit der gleichen CID gekennzeichnet sein.

Der L2CAP Layer wird von zwei Modulen gesteuert: dem Resource Manager und dem Channel Manager (siehe Abb. 10).

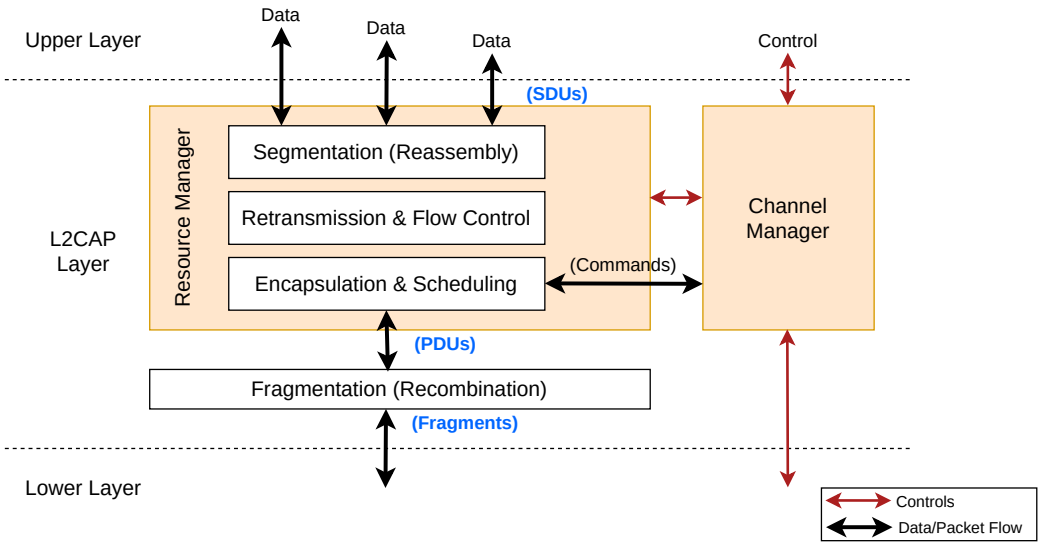

Abb. 10: Architektur des L2CAP Layers [36]

Der Channel Manager ist für die Verwaltung und Steuerung der L2CAP Channels zuständig. Das umfasst die auf L2CAP bezogene Signalübertragung intern, Peer-to-Peer, und zu höheren und niedrigeren Schichten [37]. Die Signale, die zwischen den zwei L2CAP-Entitäten zweier verbundenen Geräte übertragen werden, sind Kommandos wie bspw. der LE Credit Based Connection Request und die entsprechende Response. Dafür wird ein separater L2CAP Channel mit der CID 0x0005 genutzt. Zudem betreibt der Channel Manager den L2CAP-

Zustandsautomaten, auf den hier nicht näher eingegangen werden soll.

Da der L2CAP Layer nach unten mit dem Link Layer verknüpft ist (ggf. über das HCI), müssen die L2CAP PDUs dem Paketformat des Link Layers (bzw. dem HCI) gerecht werden. Dementsprechend werden die L2CAP PDUs fragmentiert bzw. wieder zusammengesetzt. Die Maximal PDU Payload Size (MPS) bezeichnet die maximale Größe des Payload einer PDU in Byte, die eine L2CAP-Entität verarbeiten kann.

Ein Datenaustausch zwischen L2CAP und dessen höhergelegenen Schichten / Protokollen erfolgt in der Form von Service Data Units (SDU), wobei deren Ursprung immer aus einer höheren Schicht stammt. Die Maximum Transmission Unit (MTU) bezeichnet die maximale Größe einer SDU in Byte, die die höhergelegene Schicht verarbeiten kann. Die Segmentierung (bzw. Zusammensetzung) dieser SDUs wird vom Resource Manager ausgeführt und ist für alle Modi außer dem Basic L2CAP Mode relevant. Wenn keine Segementierung angewandt wird, ist die MTU gleich der MPS [38]. Falls ein L2CAP Channel in einem anderen Modus als dem Basic L2CAP Modus agiert, ist der Resource Manager auch für die erneute Übertragung von PDUs und Flusskontrolle zuständig. Außerdem plant der Resource Manager zu welchem Zeitpunkt L2CAP Channel PDUs versenden können, um den L2CAP Channels mit Quality-of-Service-Optionen genügend Ressourcen bezüglich der Buffer des Controllers freizugeben. [39] [40]

2.5.1.1 Struktur eines Datenpakets

Es existieren verschiedene Arten von L2CAP-Datenpaketen, die den gennanten Modi zugeordnet sind. Der Basic L2CAP Mode unterstützt verbindungsorientierte und verbindungslose L2CAP Channel, während die restlichen Modi nur verbindungsorientierte L2CAP Channels nutzen. In Abb. 11 ist die Paketstruktur einer L2CAP PDU im Basic L2CAP Mode (verbindungsorientiert) dargestellt.

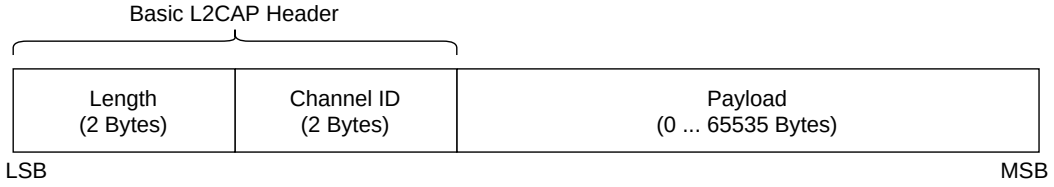

Abb. 11: Struktur einer L2CAP PDU (Basic L2CAP Mode) [41]

LSB steht für Least Significant Bit, also das Bit mit niedrigstem Stellenwert, und MSB für Most Significant Bit, also das Bit mit höchstem Stellenwert. Das Längenfeld beschreibt die Länge des Payload, der eine maximale Länge von 65535 Byte besitzt.

Abb. 12 zeigt die Paketstruktur einer L2CAP PDU im LE Credit Based Flow Control Mode.

Das SDU-Längenfeld ist nur in der ersten PDU einer SDU enthalten, um die Gesamtlänge der SDU in Byte zu beschreiben. Dadurch kann die Länge des Payloads einer solchen ersten PDU nur maximal 65533 Byte betragen, da der Wert des Längenfelds des Basic L2CAP Header auch das SDU-Längenfeld umfasst. Alle zur SDU zugehörigen folgenden PDUs enthalten das SDU-Längenfeld nicht, wodurch deren maximale Payload-Länge 65535 Byte beträgt.

Im LE Credit Based Flow Control Mode regulieren die beiden Endpunkte mithilfe von Credits wie viele PDUs der jeweilige Endpunkt empfangen kann. Sind keine Credits mehr für

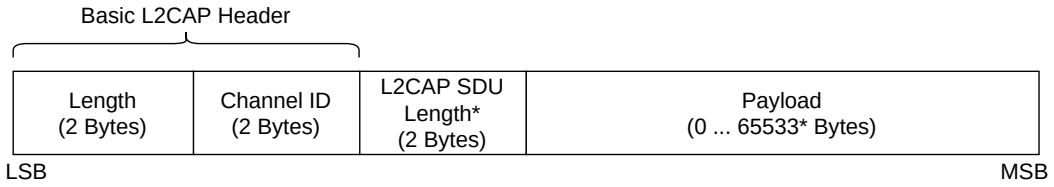

Abb. 12: Struktur einer L2CAP PDU (LE Credit Base Flow Control Mode); *nur in erster PDU einer SDU; [42]

einen Endpunkt (Empfänger) vorhanden, kann der andere Endpunkt (Sender) keine PDUs mehr an den Empfänger übermitteln und muss warten bis der Empfänger ein Credit-Signalpaket sendet, das neue Credits freigibt. [43]

2.5.2 Generic Attribute Profile

Das Generic Attribute Profile (GATT) dient zur Kommunikation zwischen Client und Server. Es basiert auf dem Attribute Protocol (ATT), welches ausgehend vom Server Attribute bereitstellt, die von einem oder mehreren Clients „entdeckt“ werden können. Ein Attribut besteht aus einem Typ, der anhand des Universal Unique Identifier (UUID) identifiziert wird, einem Attribute Handle, um auf das Attribut zuzugreifen, und einem Wert, der von Server und Client ausgelesen bzw. überschrieben werden kann. Zudem ist es möglich, für Attribute Berechtigungen festzulegen (bspw. nur Lesen, nicht Schreiben). [44]

Entsprechend der S. 9 Abb. 2 ordnet sich ATT über L2CAP in den Protokollstapel ein.

Möchte ein Client einen Attributwert lesen bzw. schreiben, dann sendet er einen Read Request bzw. einen Write Request an den Server. Dieser reagiert, indem er bei einem Read Request das Attribut an den Client sendet bzw. bei einem Write Request den Attributwert entsprechend ändert und dem Client eine Bestätigung sendet. Im Fall, dass ausgehend vom Server ein Attribut geändert werden soll, sendet dieser eine Notification oder eine Indication an den Client. Im Gegensatz zur Notification bestätigt der Client eine empfangene Indication. [45] [46]

Die Daten werden in Form von Attribute Protocol PDUs (siehe Abb. 13) übertragen.

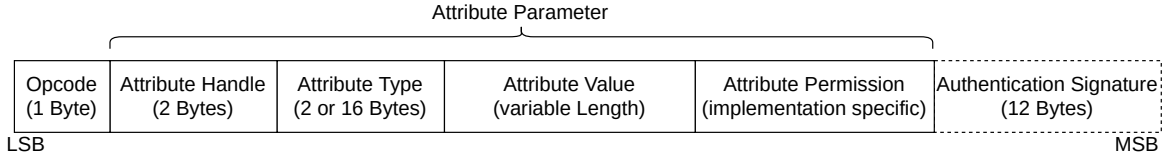

Abb. 13: (Generic) Attribute Protocol PDU; in Anlehnung an [47] und [48]

Der ein Byte lange Opcode sagt aus, ob die PDU entweder ein Request, eine Response, Notification, Indication oder Bestätigung ist, und enthält ein Flag für die Authentifizierung. Die Attributparameter unterteilen sich in zwei Byte für das Attribute Handle, zwei oder 16 Byte für den Attribute Type (die UUID), den Attributwert variabler Länge und die Attributberechtigungen, deren Länge von der Implementierung abhängig ist. Auf die Attributparameter folgt die 12 Byte lange Signatur für die Authentifizierung, falls gefordert. [49]

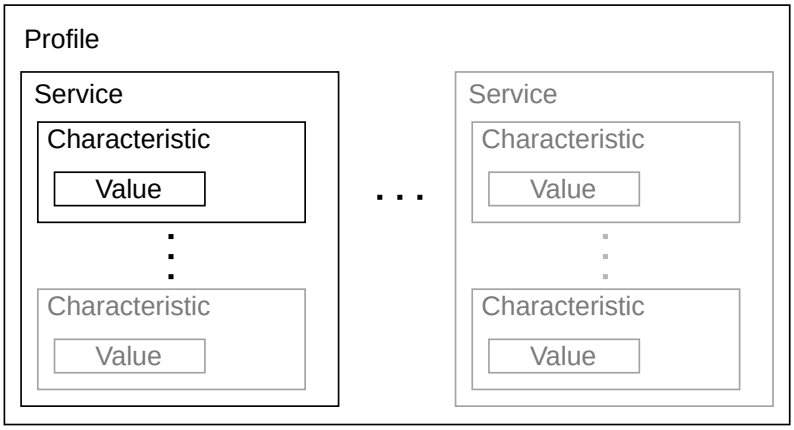

Abb. 14: Hierarchie der Attribute in GATT; in Anlehnung an [50]

GATT bildet eine Hierachie (siehe Abb. 14) bestehend aus den grundlegenden Elementen Profile, Service und Characteristic, die alle als Attribute definiert werden. An oberster Stelle befindet sich das Profile. Es enthält einen oder mehrere Services. Ein Service kann wiederum eine oder mehrere Characteristics enthalten, die sich aus Properties, einem Wert und Descriptors zusammensetzen.

2.5.3 Generic Access Profile

Das Generic Access Profile (GAP) definiert verschiedene Rollen und Modi bzw. Prozeduren für Broadcasts und den Aufbau von Verbingungen.

Für LE existieren die vier Rollen:

- Broadcaster
- Observer
- Peripheral
- Central

Ein „Broadcaster“ ist aus Sicht des Link Layers (LL) ein „Advertiser“, da er verbindungslos Daten in Form von Advertising Events sendet. Der zugehörige Modus ist der Broadcast Mode. Diese Advertising Events können von „Observern“, die die Observation Procedure ausführen, empfangen werden, weswegen sie aus Sicht des LL als Scanner bezeichnet werden. Die Rolle „Peripheral“ wird einem Gerät zugewiesen, wenn es den Aufbau eines LE Physical Link akzeptiert. Dabei nimmt es in Bezug auf den Link Layer die Rolle des „Slaves“ ein. Die Rolle „Central“ wird einem Gerät zugewiesen, wenn es den Aufbau einer physischen Verbindung einleitet. Dabei nimmt es in Bezug auf den Link Layer die Rolle des „Masters“ ein. Ein Gerät kann mehrere Rollen zur selben Zeit einnehmen. [51] [52]

Jedes Gerät befindet sich entweder im Non-discoverable Mode, in dem es nicht von anderen Geräten entdeckt werden kann, oder im General Discoverable Mode bzw. im Limited

Discoverable Mode, in denen es entdeckbar ist. Im Letzteren ist ein Gerät nur für eine bestimmte Dauer entdeckbar. Geräte müssen die General Discovery Procedure bzw. Limited Discovery Procedure ausführen, um andere Geräte zu entdecken. [53]

Um Verbindungen und deren Aufbau zu steuern, gibt es mehrere Modi und Prozeduren, von denen einige in der Tabelle 2 zusammengefasst werden.

Modus/Prozedur	**Beschreibung**
Non-connectable Mode	keine Verbindungen akzeptieren
Directed Connectable Mode	nur Verbindungen von bekannten Peer-Geräten (Bluetooth-Adresse bekannt) akzeptieren, die die Auto oder General Connection Establishment Procedure ausführen
Undirected Connectable Mode	nur Verbindungen von Geräten akzeptieren, die die Auto oder General Connection Establishment Procedure ausführen
Auto Connection Establishment Procedure	Aufbau von Verbindungen zu Geräten, die in einem Connectable Mode sind und deren Adresse auf der Whitelist eingetragen ist
General Connection Establishment Procedure	Aufbau von Verbindungen zu bekannten Peer-Geräten, die in einem Connectable Mode sind
Connection Parameter Update Procedure	Peripheral oder Central kann Link-Layer-Parameter einer Verbindung ändern

Tab. 2: Modi und Prozeduren für Verbindungen [54]

Bonding ist das Speichern von Sicherheits- und Identitätsinformationen wie dem Identity Resolving Key (IRK) oder dem Long Term Key (LTK). Der IRK generiert zufällige Adressen oder löst diese auf. Der LTK wird beim Pairing generiert und dient zur Verschlüsselung einer Verbindung (siehe Sektion 2.5.4).

Ein Gerät im Non-Bondable Mode blockiert die Erstellung eines sogenannten Bonds mit einem Peer-Gerät. Dagegen kann ein Gerät im Bondable Mode einen Bond mit einem Peer-Gerät erstellen, das nicht im Non-Bondable Mode agiert. Haben ein Central und ein Periphal zusammen einen Bond erstellt, können sie beim erneuten Verbinden mithilfe der gespeicherten Sicherheits- und Identitätsinformationen das Pairing überspringen. Broadcaster und Observer sollten das Bonding nicht unterstützen. [55]

2.5.3.1 Sicherheitsaspekte

Zusätzlich verfügt GAP über Sicherheitsaspekte, die unter anderem das Privacy Feature und die Random Device Address umfassen. Das Privacy Feature erschwert das Verfolgen eines Gerätes durch das Ändern seiner Adresse.

Entsprechend Abb. 15 existieren verschiedene Arten von Bluetooth-Adressen.

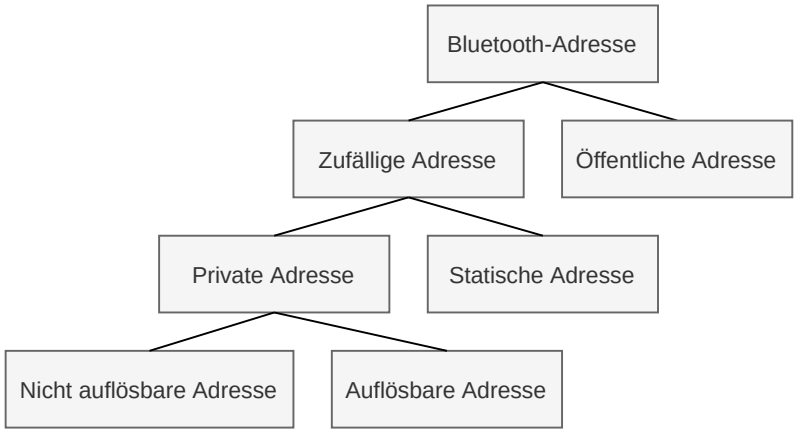

Abb. 15: Arten der Bluetooth-Adressen

Die öffentliche Bluetooth-Adresse ist die vom Hersteller bzw. IEEE festgelegte Adresse mit einer Länge von 48 Bit. Die 24 Bit mit dem höchsten Stellenwert bilden die von der IEEE vergebene Hersteller-ID und die 24 Bit mit dem niedrigsten Stellenwert bilden die vom Hersteller festgelegte Geräts-ID. [56]

Eine zufällig generierte Adresse kann entweder statisch oder privat sein. Eine zufällige statische Adresse kann bei jedem Neustart des Gerätes geändert werden (aber nicht während der Laufzeit). Die zwei Bit mit dem höchsten Stellenwert sind jeweils auf 1 gesetzt, während die restlichen 46 Bit zufällig generiert werden, ohne dass alle nur auf 0 oder nur auf 1 gesetzt sind.

Zufällig generierte private Adressen werden in auflösbare und nicht auflösbare Adressen unterschieden. Bei einer nicht auflösebaren Adresse sind die zwei höchstwertigen Bit auf 0 gesetzt und die restlichen 46 Bit werden zufällig generiert, ohne dass alle nur auf 0 oder nur auf 1 gesetzt sind. Zudem darf diese Adresse nicht der öffentlichen Adresse gleichen.

Mittels eines IRK (der lokale oder der eines Peer-Geräts) und einer Zufallszahl (24 Bit Länge) kann ein Gerät eine auflösbare Adresse generieren. Die zwei Bit der zufällig generierten Zahl mit dem höchsten Stellenwert sind auf 0 (höchster Stellenwert) und 1 (zweithöchster Stellenwert) gesetzt, während die restlichen 22 Bit zufällig generiert werden, ohne dass alle nur auf 0 oder nur auf 1 gesetzt sind. Diese zufällig generierte Zahl nimmt die 24 höchstwertigen Bit ein. Die restlichen niederwertigen 24 Bit der Adresse repräsentieren einen Hashwert, der mit der Funktion ah [57] und den Eingabewerten IRK und der Zufallszahl ermittelt wird. [58]

Nutzt ein Broadcaster oder Observer das Privacy Feature, dann verwendet er eine auflösbare oder nicht auflösbare Adresse, die er zeitlich periodisch ändert, solange er das Advertising bzw. Scanning betreibt. Jedoch wird die Adresse nur dann geändert, wenn der Controller die Auflösung von Adressen nicht unterstützt. Beim Advertising sollte der Broadcaster (genauso Peripherals) nicht seinen Gerätenamen oder Daten preisgeben, die zu dessen Identifikation verhelfen. [59]

Ein Peripheral mit aktiviertem Privacy Feature nutzt im Connectable Mode eine auflösbare Adresse und im Non-Connectable Mode eine auflösbare oder nicht auflösbare Adresse. Empfängt das Peripheral eine auflösbare Adresse und verfügt über Bonding-Information, löst

es damit die empfangene Adresse auf. Konnte die empfangene Adresse erfolgreich aufgelöst werden, kann das Peripheral die Verbindung akzeptieren. Anderenfalls wird die Verbindung abgelehnt oder das Pairing ausgeführt. Basiert das Privacy Feature des Peripheral auf dem Host, dann wird die auflösbare bzw. nicht auflösbare Adresse nur während des Advertising zeitlich periodisch geändert. [60]

Während des Scannings nutzt ein Central eine auflösbare oder nicht auflösebare Adresse, ein Initiator dagegen nur eine auflösbare Adresse. Empfängt das Central eine auflösbare Adresse und verfügt über Bonding-Information, löst es damit die empfangene Adresse auf. Konnte die empfangene Adresse erfolgreich aufgelöst werden, kann das Central die Verbindung akzeptieren. Anderenfalls wird die enpfangene Adresse nicht beachtet. Basiert das Privacy Feature des Central auf dem Host, dann wird die auflösbare bzw. nicht auflösbare Adresse nur während des Scanning zeitlich periodisch geändert. [61]

2.5.4 Security Manager

Der Security Manager (SM) bzw. das Security Manager Protocol (SMP) ist dafür zuständig, eine sichere Verbindung zwischen zwei Geräten (Master und Slave) aufzubauen. Dies umfasst im Wesentlichen das Pairing, das zur Authentifizierung und Generierung eines Schlüssels dient, und die Verteilung von Schlüsseln. Entsprechend der Abb. 16 lässt sich der Security Manager bzw. das SMP in die BLE-Architektur einordnen.

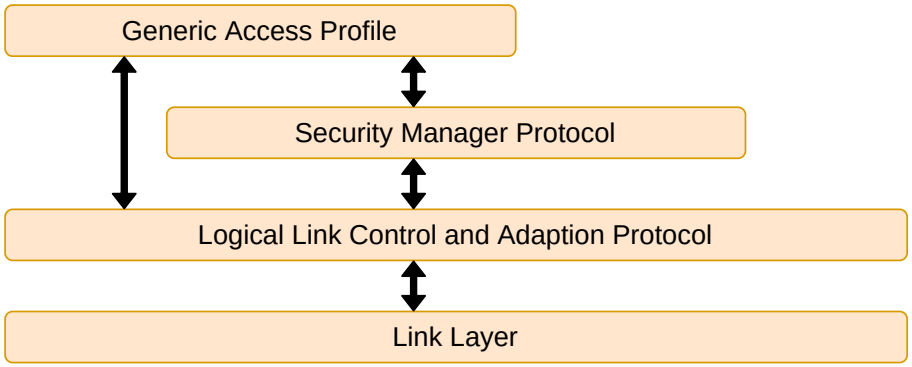

Abb. 16: Beziehungen des Security Managers zu anderen Host-Komponenten; in Anlehnung an [62]

Um mit dem Link Layer zu interagieren, nutzt der SM einen L2CAP Channel mit einer festgelegten CID. Zudem kann er mit GAP direkt kommunizieren.

Das Pairing lässt sich in drei Phasen aufteilen. Phase 1 und 2 dienen der Generierung eines Schlüssels, den beide Geräte zur Verschlüsselung und Authentifizierung im Link Layer nutzen. Dafür wird der Cipher Block Chaining - Message Authentication Code (CCM) Mode in Verbindung mit dem Advanced Encryption Standard (AES), genauer dem AES-128 Block Cipher, genutzt [63].

2.5.4.1 Pairing: Phase 1

Die erste Phase ist der Pairing Feature Exchange, bei dem die beiden Geräte ihre für den Nutzer zugänglichen Ein- und Ausgabemöglichkeiten (IO Capabilities) austauschen. Die Tabellen 3 und 4 listen die entsprechenden Bezeichnungen dieser Möglichkeiten auf.

Eingabemöglichkeit	Beschreibung
Keine Eingabe	Gerät kann keine Eingaben entgegennehmen
Ja / Nein	Gerät kann zwei verschiedene Eingaben verarbeiten, die der Bedeutung Ja bzw. Nein zugewiesen werden können (bspw. zwei Tasten)
Tastatur	Eingabe der Ziffern 0 bis 9 möglich und die Möglichkeit Ja und Nein einzugeben

Tab. 3: Eingabemöglichkeiten eines Gerätes [64]

Ausgabemöglichkeiten	Beschreibung
Keine Ausgabe	Gerät kann dem Nutzer keine sechsstellige Dezimalzahl anzeigen bzw. kommunizieren
Numerische Ausgabe	Gerät kann dem Nutzer eine sechsstellige Dezimalzahl anzeigen bzw. kommunizieren

Tab. 4: Ausgabemöglichkeiten eines Gerätes [65]

Desweiteren tauschen die Geräte in der ersten Phase Informationen darüber aus, ob eine Authentifizierung zum Schutz vor einem Man-In-The-Middle-Angriff nötig ist (MITM Flag), und ob Daten für die Authentifizierung über die Pairing-Methode Out Of Band (OOB), d.h. mittels einer anderen Technologie (z.B. Near Field Communication), übertragen werden können (OOB Flag). Außerdem werden die minimalen und maximalen Größen für die Schlüssel ausgetauscht, die sich in 8 Bit großen Schritten zwischen 56 Bit (7 Byte) und 128 Bit (16 Byte) befinden. Dabei wird das Minimum beider maximaler Größen übernommen. Falls die beiden Spannen sich nicht überschneiden, wird das Pairing abgebrochen.

2.5.4.2 Pairing: Phase 2

Anhand der ausgetauschten Informationen aus der ersten Phase wird in der zweiten Phase entschieden, welche der folgenden Methoden zur Generierung des Short Term Key (STK) bzw. Long Term Key (LTK) zu verwenden ist:

- Numeric Comparison (für LE erst ab Bluetooth-Version 4.2)
- Just Works
- Out of Band (OOB)
- Passkey Entry

Da das Pairing (speziell Phase 2) für LE in der Bluetooth-Version 4.0 funktionale Unterschiede zum Pairing für LE ab Version 4.2 aufweist, wird das Pairing für LE in Version 4.0 als **LE Legacy Pairing** und das Pairing für LE ab Version 4.2 als **LE Secure Connections Pairing** bezeichnet. Aus Sicht des Nutzers sind beide Verfahren jedoch gleich und jede Bluetooth-Version, die LE unterstützt, unterstützt das LE Legacy Pairing. Im Gegensatz zum LE Secure Connections Pairing bietet LE Legacy Pairing für die Methoden Just Works

und Passkey Entry keinen Schutz vor passivem Abhören während des Pairings, da LE Secure Connections Elliptic Curve Diffie-Hellman (ECDH) für den Schlüsselaustausch nutzt und LE Legacy Pairing nicht. [66]

Verfügen beide Geräte über OOB-Authentifizierungsdaten für das LE Legacy Pairing, wird unabhängig vom jeweiligen MITM-Flag die Methode OOB gewählt. In LE Secure Connections Pairing wird ebenso verfahren, nur dass hier nicht die OOB-Flags beider Geräte gesetzt sein müssen (aber können), da ein gesetztes OOB-Flag genügt. Sind die MITM-Flags beider Geräte nicht gesetzt, wird die Methode Just Works ausgeführt. Anderenfalls werden die Ein- und Ausgabemöglichkeiten der Geräte entsprechend [67] in die Wahl der Methode einbezogen.

Pairing Methoden

Bei der **Numeric Comparison** wird dem Nutzer auf beiden Geräten jeweils eine zufällig generierte sechsstellige Dezimalzahl angezeigt. Diese muss der Nutzer vergleichen und im Falle der Übereinstimmung auf beiden Geräten bestätigen oder anderenfalls ablehnen. Somit kann der Nutzer unabhängig von der Namensgegebung der Geräte sicherstellen, die richtigen Geräte ausgewählt zu haben. Zudem bietet diese Methode Schutz vor MITM-Angriffen. Außenstehende, die Kenntnis über diese Zahl gewonnen haben, können laut [68] damit keinen Vorteil zur Entschlüsselung der zwischen den beiden Geräten ausgetauschten Daten erlangen, da die Zahl nicht als Eingabe zur Generierung eines Schlüssels verwendet wird.

Just Works basiert auf der Funktionsweise von Numeric Comparison mit dem Unterschied, dass hier dem Nutzer keine sechsstellige Dezimalzahl ausgegeben wird und er die Verbindung nur bestätigen muss. Dadurch bietet Just Works keinen Schutz vor MITM-Angriffen, aber einen Schutz gegen passives Abhören (außer für LE Legacy Pairing). [69]

Out of Band ist die Nutzung einer anderen Technologie (z.B. Near Field Communication), um Geräte zu entdecken oder um kryptographische Informationen für den Pairing-Prozess auszutauschen. Dabei sollte die Technologie Schutz vor MITM-Angriffen bieten. [70]

Die Methode **Passkey Entry** definiert, dass ein Gerät die zufällig generierte sechstellige Dezimalzahl ausgibt und der Nutzer sie auf dem anderen Gerät eingeben muss. Ein Schutz gegen MITM-Angriffe ist gegeben, da diese nur mit einer Wahrscheinlichtkeit von 0,0001% für jede Durchführung der Methode möglich sind. Schutz gegen passives Abhören bietet Passkey Entry nur in LE Secure Connections Pairing und nicht in LE Legacy Pairing. [71] [72]

LE Legacy Pairing: Schlüssel und deren Generierung

Beim LE Legacy Pairing zweier Geräte generieren beide einen 128 Bit langen Temporary Key (TK), der bei der Authentifizierung genutzt wird, um den STK zu generieren und die Verbindung zu verschlüsseln. In Just Works wird der TK auf null gesetzt. Bei der Methode Passkey Entry ist der TK die besagte zufällig generierte sechstellige Dezimalzahl, die bereits mit 20 Bit dargestellt werden kann, weswegen die restlichen Bit des TK auf null gesetzt werden müssen. Dagegen kann bei OOB auf diese Einschränkung verzichtet werden, wodurch der TK tatsächlich eine Länge von 128 Bit aufweist.

Das Gerät, welches das Pairing einleitet (Master), generiert eine zufällige 128 Bit lange Nummer *Mrand* und ermittelt den gleichlangen Bestätigungswert *Mconfirm* mit der Confirm Value Generation Function c1 [73]. Zur Berechnung von *Mconfirm* erhält die Funktion c1 folgende

Eingabewerte entsprechend Gl. 1 [74]:

$$\begin{aligned} \text{Mconfirm} = \text{c1}(&\text{TK, Mrand,} \\ &\text{Pairing Request Command, Pairing Response Command,} \\ &\text{Adresstyp des Masters, Adresse des Masters,} \\ &\text{Adresstyp des Slaves, Adresse des Slaves)} \end{aligned} \tag{1}$$

Ebenso führt das antwortende Gerät (Slave) diese Schritte durch, wobei *Mrand* als *Srand* bezeichnet wird und *Mconfirm* als *Sconfirm*. Die Eingabewerte der Funktion c1 zur Berechnung von *Sconfirm* sind demnach analog zu denen von *Mconfirm*. Danach findet gemäß Abb. 17 folgender Austausch statt:

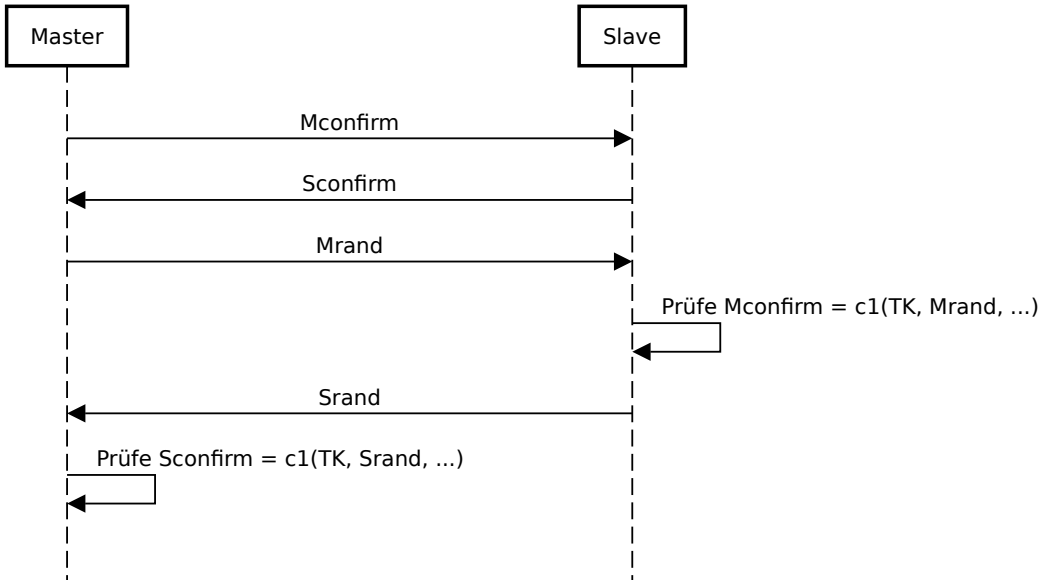

Abb. 17: Austausch von *Mconfirm*, *Sconfirm*, *Mrand* und *Srand* zwischen Master und Slave

Master und Slave tauschen *Mconfirm* und *Sconfirm* aus. Danach überträgt der Master *Mrand*, damit der Slave *Mconfirm* entsprechend Gl. 1 berechnen und so den empfangenen *Mconfirm* verifizieren kann. Nach einer erfolgreichen Prüfung von *Mconfirm* überträgt der Slave *Srand*, damit der Master *Sconfirm* analog zu Gl. 1 berechnen und somit den empfangenen *Sconfirm* verifizieren kann. Wenn einer der Bestätigungswerte (*Mconfirm*, *Sconfirm*) nicht erfolgreich verifiziert wird, wird die Verbindung sofort beendet. [74]

Anschließend wird der STK mit der Funktion s1 [75] entsprechend Gl 2 berechnet. [74]

$$\text{STK} = \text{s1}(\text{TK, Srand, Mrand}) \tag{2}$$

Demnach kann bei der Methode Passkey Entry kein ausreichender Schutz gegen passives Abhören geboten werden, da der TK nur wenig mögliche Werte annehmen kann. Ist die vereinbarte Schlüsselgröße kleiner als 128 Bit, werden die überschüssigen Bit beginnend bei dem MSB auf null gesetzt. Der STK wird nun zur Verschlüsselung der Verbindung genutzt. [74]

LE Secure Connections Pairing: Schlüssel und deren Generierung

Beim LE Secure Connections Pairing wird ein Long Term Key (LTK) erstellt. Zuvor generieren beide Geräte jeweils ein ECDH-Schlüsselpaar (PK - Public Key, SK - Private Key) und tauschen ihre Public Keys aus. Danach berechnet jedes Gerät den Diffie-Hellman-Schlüssel aus seinem Private Key und dem Public Key des Gegenübers. Durch den Diffie-Hellman-Schlüssel kennen beide Parteien ein gemeinsames Geheimnis, mit dem sie den weiteren Datenaustausch zur Authentifizierung verschlüsseln können. Die Authentifizierung ist notwendig, da der ECDH-Schlüsselaustausch zwar resistent gegen passives Abhören ist, jedoch nicht gegen MITM-Angriffe. [76]

Diese Authentifizierung wird mit den Pairing Methoden Numeric Comparison, Just Works, OOB und Passkey Entry ermöglicht. Jedoch unterscheiden diese sich aus funktionaler Sicht (nicht aus Nutzersicht) vom LE Legacy Pairing durch komplexere Verfahren. Letztendlich lässt sich für die vier Pairing-Methoden Folgendes zusammenfassen. Numeric Comparison signalisiert dem Nutzer mit einer Wahrscheinlichkeit von 99,9999% einen stattfindenden MITM-Angriff [77]. Just Works bietet keinen Schutz vor einem MITM-Angriff [69]. Ein MITM-Angriff während des Passkey Entry gelingt nur mit einer Wahrscheinlichkeit von 0,0001% [78]. Die Anfälligkeit der OOB-Methode ist von der verwendeten OOB-Technologie abhängig [79].

Nach der Ausführung einer Pairing-Methode wird der LTK als Teilergebnis der Funktion f5 [80] mit den Eingabewerten Diffie-Hellman-Key, einer Nonce des Masters, einer Nonce des Slaves und der Adresse des Masters und Slaves ermittelt [81].

2.5.4.3 Pairing: Phase 3

Wurde der STK bzw. LTK generiert, wird er genutzt, um die Verbindung zu verschlüsseln. Nun können in der dritten Phase transportspezifische Schlüssel ausgetauscht werden. Z.B. wird der Identity Resolving Key (IRK) zur Generierung und Auflösung von zufälligen Adressen (Privacy Feature) verwendet und der Connection Signature Resolving Key (CSRK) zur Signatur von Daten und Überprüfung von Signaturen.

2.6 Sicherheit

Obwohl Bluetooth LE mit dem Privacy Feature und dem Security Manager (vor allem ab Bluetooth 4.2) einige Optionen für eine sichere Infrastruktur bietet, bleiben Probleme offen, die betrachtet werden müssen.

Ein erfolgreicher Angriff auf zwei Geräte, die mittels BLE verschlüsselt kommunizieren, in Form von passivem Abhören oder eines MITM-Angriffs kann beim Aufbau einer Verbindung, also dem Pairing (siehe Sektion 2.5.4) auftreten, da hier die Schlüssel ausgetauscht werden. In Tabelle 5 wird dargestellt, welchen Schutz die Pairing-Methoden des LE Legacy Pairing und des LE Secure Connections Pairing gegen passives Abhören und MITM-Angriffe bieten.

Dabei ist zu beachten, dass die Methode Numeric Comparison dem Nutzer mit einer Wahrscheinlichkeit von 99,9999% einen stattfindenden MITM-Angriff signalisiert [77], bevor er das Pairing fortsetzt, und dass die Methode Passkey Entry nur mit einer Wahrscheinlichkeit von 0,0001% anfällig für einen MITM-Angriff ist [72] [78]. Die Methode OOB des LE Legacy Pairing ist bei einer sicheren OOB-Technologie ebenfalls mit einer Wahrscheinlichkeit von 0,0001% oder kleiner (je nach Schlüsselgröße) anfällig gegen MITM-Angriffe [83].

Obwohl die Methoden Numeric Comparison und Passkey Entry des LE Secure Connections

	Schutz gegen Passives Abhören		Schutz gegen MITM-Angriff	
	LE Legacy	**LE Secure Connections**	**LE Legacy**	**LE Secure Connections**
Numeric Comparison	-	Ja	-	Ja [77]
Just Works	Nein [82]	Ja [69]	Nein [82]	Nein [69]
Out of Band	abhängig von OOB-Technologie [83] [79]			
Passkey Entry	Nein [72]	Ja	Ja [72]	Ja [78]

Tab. 5: Schutz durch Pairing-Methoden vor passivem Abhören und MITM-Angriffen; Der Bindestrich symbolisiert, dass diese Methode nicht verfügbar ist. LE Secure Connections ist aufgrund des ECDH-Schlüsselaustausches [76] generell vor passivem Abhören geschützt.

Pairing eine beachtliche Sicherheit bieten, ist laut den Bluetooth-Spezifikationen 4.0 bis 5.2 jede dieser Bluetooth-Versionen in der Lage, das LE Legacy Pairing auszuführen [84] [85], das wiederum anfälliger ist (OOB ausgenommen). Vermutlich kann dies durch die letztendlichen Entwickler einer Bluetooth-Software bzw. -Hardware oder durch den Anwender selbst eingeschränkt werden. Falls dies nicht umgesetzt wird, stellt die Möglichkeit einer Rückstufung auf LE Legacy Pairing ein Sicherheitsrisiko dar.

Dabei sei zu erwähnen, dass jede Methode bestimmte Möglichkeiten zur Ein- und Ausgabe von den zu verwendenten Geräten voraussetzt. Ist es für den Anwender nicht möglich, diese Voraussetzungen in den Geräten zu implementieren, kann BLE selbst keine Schutz bieten.

Ein weiteres Problem ist die Sicherheit auf der Anwendungsebene. Unterstützt ein System weitere Anwendungen, könnte es möglich sein, dass eine solche fremde Anwendung auf Daten zugreifen kann, die nicht für sie bestimmt sind. Die Möglichkeit dazu besteht, da BLE zu übertragende Daten innerhalb des Controllers auf der Ebene des Link Layer verschlüsselt [86] [63]. Ein Beispiel für ein solches System ist das Betriebssystem *Android* von der *Open Handset Alliance*. Auf Androids Webpräsenz für BLE wird darauf hingewiesen, dass BLE keine Sicherheit auf Anwendungsebene liefert: „When a user pairs their device with another device using BLE, the data that's communicated between the two devices is accessible to all apps on the user's device“ [87]. Den Beweis dafür, dass Daten, die durch BLE über den Link Layer verschlüsselt oder unverschlüsselt übertragen werden, von anderen Apps ausgelesen werden können, liefert eine Studie der *Royal Holloway University of London* [88]. Sie zeigt, wie innerhalb des Betriebssystems *Android* eine Anwendung Daten über das Protokoll ATT bzw. GATT empfangen kann, die theoretisch für eine andere Anwendung bestimmt sind.

3 Infrastruktur

Die Infrastruktur beschreibt eine allgemeine Lösung, um eine sichere Kommunikation mittels Bluetooth Low Energy (BLE) zwischen einem Mikrocontroller und einem Smartphone zu gewährleisten. Bis auf die Tatsache, dass ein Mikrocontroller und ein Smartphone über BLE miteinander kommunizieren, sind die spezifischen Systeme der beiden Kommunikationsparteien nicht vorgegeben. Dennoch ist die Lösung universell einsetzbar, solange jedes System über ein Bluetooth-Modul (mind. der Bluetooth-Version 4.0) verfügt. Da das Protokoll *Transport Layer Security* (TLS) ein wesentlicher Bestandteil des Lösungsansatzes ist, müssen die Systeme eine Software-Bibliothek für TLS unterstützen. Somit ist die Infrastruktur unabhängig von dem in der Sektion 1 beschriebenen Projekt *SteigtUM*.

Da das Ziel dieser Infrastruktur die sichere Datenübertragung zwischen zwei Kommunikationsparteien ist, sollte diesbezüglich „Sicherheit“ definiert werden. Nach [89] werden mehrere Sicherheitsziele definiert, um ein sicheres Netzwerk zu schaffen. Daraus ergeben sich die für diese Infrastruktur wesentlichen Sicherheitsziele der Vertraulichkeit, Datenintegrität und Authentizität.

Vertraulichkeit bedeutet, dass „Übertragene Daten [...] nur berechtigten Instanzen zugänglich sein [sollen], d.h. keine unbefugte dritte Partei soll an den Inhalt von übertragenen Nachrichten gelangen können“ [90]. Die Datenintegrität trägt folgende Bedeutung. „Für den Empfänger muss eindeutig erkennbar sein, ob Daten während ihrer Übertragung unbefugt geändert wurden“ [90]. Authentizität teilt sich in zwei Punkte auf. Zum einen soll „eine Instanz [...] einer anderen ihre Identität zweifelsfrei nachweisen können (Identitätsnachweis bzw. Authentifizierung der Instanz)“ [90]. Zum anderen „soll überprüft werden können, ob eine Nachricht von einer bestimmten Instanz stammt (Authentizität der Daten)“ [90].

3.1 Topologie

Die Topologie der Infrastruktur beschränkt sich auf das Minimum an Kommunikationsparteien. Dem Thema zufolge sollen ein Mikrocontroller und ein Smartphone sicher Daten austauschen. Um dies zu bewerkstelligen, wird über dem Transport der Daten durch Bluetooth Low Energy (BLE) das Protokoll TLS verwendet (siehe Sektion 3.3). Dementsprechend ist, wie in Abb. 18 zu sehen, neben dem Mikrocontroller und dem Smartphone eine Zertifizierungsstelle notwendig. In welcher Form diese auftritt, ist von der Anwendung abhängig.

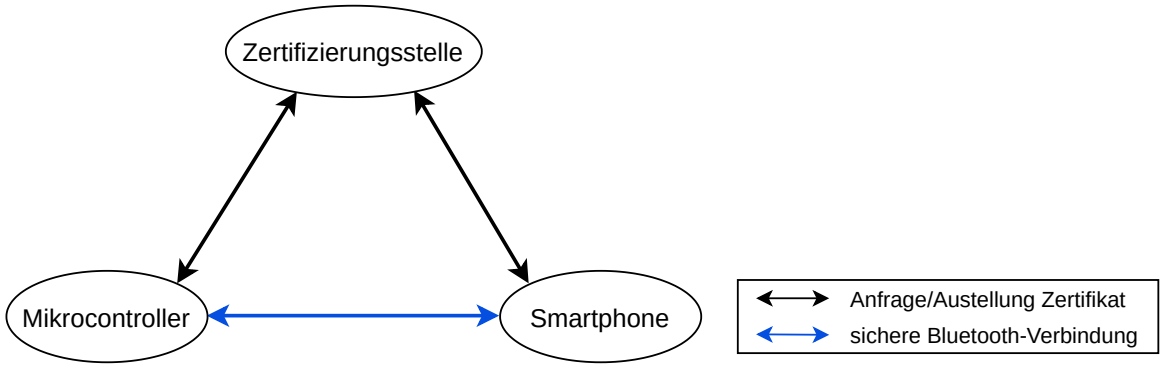

Abb. 18: Topologie der Infrastruktur

Für einen öffentlich zugänglichen Anwendungsfall sollte die Zertifizierungsstelle dem Mikrocontroller und Smartphone in regelmäßigen Abständen (z.B. jährlich) jeweils ein Zertifikat ausstellen. Mit diesen Zertifikaten und der Kenntnis über das Root-Zertifikat (das Zertifikat der Zertifizierungsstelle) können Mikrocontroller und Smartphone sich gegenseitig authentifizieren und somit die Grundlage für eine sichere Kommunikation bilden.

Für einen einfachen privaten Anwendungsfall, in dem z.B. nur ein einziger Mikrocontroller und ein einziges Smartphone diese Infrastruktur verwenden, würde es genügen, die Schlüsselpaare und Zertifikate auf einer dritten Instanz (z.B. einem Personal Computer) zu generieren und über eine sichere Verbindung (z.B. eine direkte Kabelverbindung) an den Mikrocontroller und das Smartphone zu übertragen.

Die Zuweisung der Rollen ist nicht vorgeschrieben. Jedoch wäre es üblich, den Mikrocontroller in Bezug auf BLE als Slave bzw. Peripheral (siehe Sektion 2.5.3) und in Bezug auf TLS als Server zu definieren. Folglich würde das Smartphone bzgl. BLE als Master bzw. Central und bzgl. TLS als Client festgelegt werden.

3.2 Transport

Einer der grundlegenden Aspekte dieser Arbeit ist die Tatsache, dass BLE als Technologie zum Übertragen der Daten zwischen Smartphone und Mikrocontroller genutzt werden soll. Demnach stellt sich die Frage nach einem geeigneten Transport der Daten innerhalb der BLE-Architektur.

Es existieren mehrere Referenzmodelle für die Kommunikation innerhalb von Rechnernetzen. Geläufig sind das TCP/IP-Referenzmodell (Transport Control Protocol / Internet Protocol), das OSI-Referenzmodell (Open Systems Interconnection) und ein hybrides Referenzmodell aus diesen beiden. Alle drei legen sich auf unterschiedliche Anzahlen von Schichten fest, innerhalb derer einige Überschneidungen auftreten. Die Eigenschaften der Transportschicht sind bei den drei Referenzmodellen identisch. Die zu übertragenden Daten der darüberliegenden Anwendungsschicht werden in Segmente aufgeteilt und von einem Endpunkt über die niedrigeren Schichten zum anderen Endpunkt gesendet. Auf Empfängerseite erhält die Transportentität diese Segmente, setzt sie wieder zusammen und übergibt sie der zugehörigen Anwendung, die mithilfe von Ports identifiziert wird. Demnach kann die Transportschicht eine verbindungslose oder verbindungsorientierte Datenübertragung unterstützen, wobei die verbindungsorientier-

te Übertragung Fluss- und Verstopfungskontrolle, verlustfreie Übertragung und die korrekte Reihenfolge der Segmente gewährleisten kann. [91]

Die BLE-Architektur lässt sich entsprechend der Abb. 19 am besten dem hybriden Referenzmodell zuordnen, da es im Gegensatz zum OSI-Referenzmodell die Anwendungsschicht nicht in drei weitere Schichten unterteilt und abgesehen von dieser Differenz dem OSI-Referenzmodell gleicht.

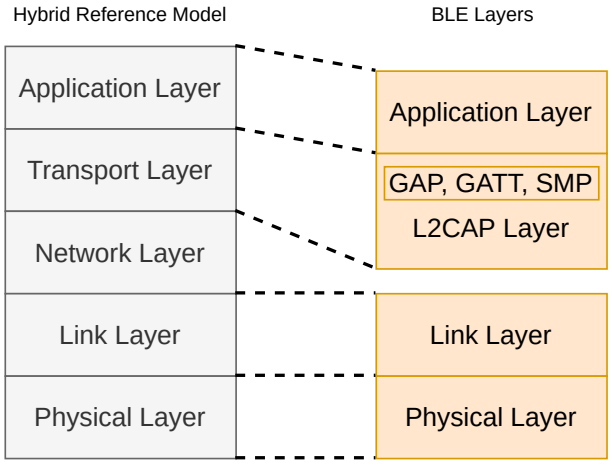

Abb. 19: Zuordnung der BLE Layer zum hybriden Referenzmodell

Der Physical Layer bzgl. BLE stimmt mit dem Physical Layer des hybriden Referenzmodells überein, da hier die physikalische Bitübertragung definiert wird. Der Link Layer bzgl. BLE unterstützt wie der Link Layer des hybriden Referenzmodells den Zugriff auf das Übertragungsmedium mittels der Access Address und das Erkennen von fehlerhaft übertragenen Paketen (BLE) bzw. Frames (hybrides Referenzmodell) mittels einer Prüfsumme. Auch physische Adressen (zumindest die öffentliche Bluetooth-Adresse) finden sich im Link Layer von BLE beim Advertising und Verbindungsaufbau wieder. Ein Äquivalent zum Network Layer des hybriden Referenzmodells existiert nicht, da nur Punkt-zu-Punkt-Verbindungen existieren. Folglich ist kein Routing der Pakete notwendig.

L2CAP bzw. der L2CAP-Layer kann als Äquivalent zum Transport Layer des hybriden Referenzmodells angesehen werden. Die Nutzdaten der Anwendungsschicht werden in Segmente aufgeteilt und über die unteren Schichten an den L2CAP-Layer des Empfängers übertragen. Über die L2CAP-Channels und deren CIDs können diese Nutzdaten den Anwendungen zugeordnet werden. L2CAP ist in der Lage, die Frames/PDUs verbindungsorientiert oder verbindungslos zu kommunizieren. Im LE Credit Based Flow Control Mode unterstützt BLE ab Bluetooth-Version 4.2 Flusskontrolle. Eine verlustfreie Übertragung wird bereits im Link Layer sichergestellt, da jedes Link-Layer-Paket vom Empfänger positiv (ACK) oder negativ (NAK) bestätigt wird. Wird ein Paket negativ bestätigt, wird es erneut gesendet.

Alle weiteren Protokolle wie GAP, SMP und GATT nutzen den Transport durch L2CAP. GAP dient dem Verbindungsaufbau und SMP bietet Sicherheitsfunktionen. Sie gehören im hybriden Referenzmodell dem „oberen Teil“ des Transport Layers an.

Ein wichtiger Schwerpunkt von BLE liegt in der Übertragung von Sensordaten [92]. Da GATT

für deren Übertragung von Sensordaten geeignet ist, tritt es häufig bei der Recherche nach der Funktionsweise und den Anwendungen von BLE auf. Jedoch basiert GATT seinem Namen entsprechend auf hierarchisch gegliederten Attributen, die unter anderem veränderbare Werte beinhalten. Ein Server erstellt solche Attribute und gibt sei über das Advertising oder erst über eine Verbindung zu einem Client bekannt. Der Client kann die Werte der Attribute mittels Anfragen lesen und schreiben. Der Server verwaltet die Attribute, weswegen er sie zu jeder Zeit lesen und schreiben kann. Zudem kann der Server Attribute/Werte direkt an einen Client senden, ohne eine vorherige Anfrage von ihm erhalten zu haben.

Ein geeigneter Anwendungsfall wäre demnach ein Server, der über einen oder mehrere Sensoren verfügt, deren Ausgabewerte er zeitlich periodisch in seine vorher erstellten Attribute schreibt. Ein oder mehrere Clients können dann über die Attribute spezifische Sensordaten anfragen und empfangen.

Daraus kann gefolgert werden, dass GATT im Gegensatz zum reinen L2CAP einen Overhead benötigt und aufgrund seiner Architektur für diese Infrastruktur als Transportprotokoll ungeeignet ist. Allerdings zeichnet GATT sich durch eine breite Unterstützung in den meisten BLE-Programmierschnittstellen aus.

Der Zugang zu L2CAP ist dagegen nicht überall verfügbar. Ein Beispiel dafür sind die beiden gängigen Betriebssysteme *Android* und *iOS*. *Android* unterstützt BLE allgemein seit Android 4.3 [87], doch ermöglicht den Zugang zu L2CAP (also LE Connection-Oriented Channels) erst ab Android 10 [93]. *iOS* unterstützt BLE allgemein seit iOS 5.0 [94] und ermöglicht den Zugang zu L2CAP erst ab iOS 11.0 [95].

Trotz des eingeschränkten Zugangs zu L2CAP in vielen Programmierschnittstellen wird es als Transportprotokoll für diese Infrastruktur gewählt. Die Nutzung von GATT würde einen deutlichen Overhead verursachen, da es auf L2CAP aufsetzt und die Attributsemantik nicht mit dem Anwendungsfall dieser Arbeit kompatibel ist.

3.3 Sicherheit

BLE bietet einige Sicherheitsfunktionen, die durch den Security Manager bzw. das Security Manager Protocol (siehe Sektion 2.5.4) zur Verfügung gestellt werden. Jedoch sind diese Maßnahmen nicht ausreichend, um die vorliegende Infrastruktur zu schützen. Zudem sind sie abhängig von den Ein- und Ausgabeoptionen, die die kommunizierenden Geräte (Mikrocontroller und Smartphone) unterstützen. In Sektion 2.6 werden die konkreten Mängel hervorgehoben. Dabei ist eine der kritischsten Sicherheitslücken die nicht vorhandene Sicherheit auf Anwendungsebene, die für die Infrastruktur unbedingt bestehen muss, da sonst im System des Smartphones Software Dritter auf die Daten der eigenen Anwendung zugreifen kann.

3.3.1 Transport Layer Security

Eine selbsterstellte kryptographische Implementierung für den Schutz des Transports wäre sehr aufwendig und aufgrund der hohen Komplexität kryptographischer Verfahren inhärent fehleranfällig. Stattdessen sollte auf bewährte Lösungen von professionellen Organisationen gesetzt werden. Ein äußerst weitverbreitetes Protokoll zum Schutz der Transportschicht ist das *Transport Layer Security* (TLS) Protocol. Es wird von der *Internet Engineering Task Force* (IETF) entwickelt, wobei die neueste Version 1.3 im August 2018 veröffentlicht wurde.

TLS lässt sich im Wesentlichen in zwei Teilprotokolle unterteilen: das Handshake Protocol

und das Record Protocol. Das Handshake Protocol dient dem Austausch von Sicherheitsparametern zwischen Client und Server. Dabei einigen sich beide Parteien, welche der gemeinsam unterstützten Cipher Suites (Mengen aus Algorithmen für Verschlüsselung, Schlüsselaustausch, Authentifizierung und Hash-Funktion) genutzt wird. Zudem authentifiziert der Client den Server, indem er dessen empfangenes Zertifikat verifiziert. Eine beidseitige Authentifikation ist möglich und kann vom Server angefordert werden. Wurde das Handshake Protocol erfolgreich abgeschlossen, können Server und Client mithilfe des Record Protocols durch den entsprechenden Algorithmus aus der vereinbarten Cipher Suite nun die Anwendungsdaten verschlüsseln und deren Datenintegrität sicherstellen. Somit erfüllt TLS die in Sektion 3 gesetzten Sicherheitsziele Vertraulichkeit, Datenintegrität und Authentizität / Authentifizierung. [96]

Weitere Argumente für die Verwendung von TLS sind folgende: TLS ist unabhängig vom Transport Layer, weshalb nicht zwingend Sockets verwendet werden müssen. Zudem existieren viele Software-Bibliotheken zu TLS, wodurch mehrere Platformen unterstützt werden. Die Ciphersuites können je nach verwendeter Programmierschnittstelle vor der Laufzeit für das Zielsystem beschränkt werden. Außerdem wird für diese Infrastruktur mit TLS der Mangel an Sicherheit auf Anwendungsebene beseitigt, da eine Ende-zu-Ende-Verschlüsselung vorliegt.

Die Versionen TLS 1.0 und TLS 1.1 sind veraltet und sollten nicht mehr genutzt werden [97]. Demnach sind die Versionen TLS 1.2 oder bevorzugt TLS 1.3 zu nutzen.

3.3.2 BLE-Sicherheitsfunktionen

Das Privacy Feature von BLE erschwert das Verfolgen eines Geräts, indem dessen Adresse zeitlich periodisch geändert wird. Es wäre als zusätzliche Funktion denkbar, da TLS von der BLE-Architektur separiert ist und somit keinen Einfluss auf auf die Darstellung der Bluetooth-Adressen nehmen kann. Jedoch ist das Privacy Feature Teil der dritten Phase des Pairings (siehe Sektion 2.5.4.3) und das Pairing sollte nicht verwendet werden. Unabhängig davon, welche Pairing-Methode verwendet wird, würden die Anwendungsdaten der Pakete im Link Layer verschlüsselt werden. Jedoch bietet eine weitere Verschlüsselung neben TLS keinen Mehrwert für die Sicherheit der Infrastruktur und würde nur mehr Leistung beanspruchen.

3.4 Verbindungsaufbau

Vor einem Verbindungsaufbau müssen die Rollen zugewiesen sein. Wie bereits in Sektion 3.1 erwähnt, ist es naheliegend, den Mikrocontroller als Slave bzw. Peripheral (bzgl. BLE) und Server (bzgl. TLS) zu definieren und das Smartphone als Master bzw. Central (bzgl. BLE) und Client (bzgl. TLS).

Zur Darstellung des Verbindungsaufbaus ist im Anhang 6.2 S. 46 ein Sequenzdiagramm abgebildet.

3.4.1 Advertising/Scanning

Zunächst muss das Peripheral mit dem Advertising beginnen. Während das Peripheral auf den drei Advertising Physical Channels periodisch Advertisments in Form eines Broadcasts sendet, scannt das Central diese Channels. Empfängt das Central ein Advertisement, muss es prüfen, ob es sich mit dem Sender des Advertisements verbinden soll. Dazu kann die Entscheidung entweder dem Nutzer oder der Anwendung überlassen werden. Je nachdem, welche

Informationen das Peripheral im Advertising-Paket angibt, kann diese Entscheidung getroffen werden. Es könnte neben der Bluetooth-Adresse ein Geräte-Name (Local Name) und herstellerspezifische Daten angegeben werden. Diese Informationen bezeugen nicht die Identität des Peripherals. Soll sich das Central nun zum Peripheral verbinden, sendet es einen Connection Request an das Peripheral. Das Peripheral akzeptiert die Verbindung und beide Parteien können nun über diese Verbindung auf den Piconet Physical Channels Daten übertragen. GAP steuert den Verbindungsaufbau auf Ebene des Hosts und stellt der Anwendung die verbindungsspezifischen Daten zur Verfügung. Mit diesen Daten baut die Anwendung nun eine Verbindung über L2CAP mittels eines L2CAP Connection-Oriented Channels auf, der für den weiteren Verlauf des Verbindungsaufbaus und der Übertragung der Anwendungsdaten dient. Wie bereits in Sektion 3.3.2 erörtert, sollte kein Pairing und somit keine Verschlüsselung durch die BLE-Architektur angewandt werden.

3.4.2 TLS-Handshake

Nun authentifizieren sich Server (Peripheral) und Client (Central) mit dem TLS-Handshake und führen ein Schlüsselaustauschverfahren durch, um mit dem generierten Schlüssel die Anwendungsdaten zu verschlüsseln. Zu Beginn sendet der Client ein „Client Hello“, das unter anderem die höchste unterstützte TLS-Version, eine Zufallszahl und eine Liste der möglichen Cipher Suites enthält. Falls der Server die Version unterstützt, wählt er eine der Cipher Suites, die er auch selbst unterstützt, und sendet sie mit einer Zufallszahl an den Client („Server Hello“).

Um eine beidseitige Authentifizierung zu gewährleisten, kann der Server einen „Certificate Request“ senden. Daraufhin übermittelt der Server sein Zertifikat, das der Client mit dem Zertifikat der Zertifizierungstelle bzw. mit dessen öffentlichem Schlüssel verifiziert. Nachdem der Server sein Zertifikat übertragen hat, sendet er ein „Certificate Verify“, das ebenfalls vom Client verifiziert wird. „Certificate Verify“ ist eine Signatur der bisher ausgetauschten Nachrichten, die unter anderem mit dem privaten Schlüssel des Server-Zertifikats erstellt wurde. So kann der Client mit dem öffentlichen Schlüssel des Server-Zertifikats prüfen, ob der Server wirklich der Besitzer des besagten privaten Schlüssels ist. Da der Server zuvor einen „Certificate Request“ gestellt hat, muss der Client nun sein Zertifikat und anschließend sein „Certificate Verify“ senden, damit der Server beide Datensätze verifizieren kann.

Nun sind beide Parteien authentifiziert und es wird mittels des vereinbarten Schlüsselaustauschverfahrens aus der gewählten Cipher Suite ein symmetrischer Schlüssel generiert. Anschließend werden die Anwendungsdaten mit dem vereinbarten Algorithmus aus der Cipher Suite und dem soeben generierten symmetrischen Schlüssel verschlüsselt und übertragen.

4 Implementierung

Mithilfe der definierten Infrastruktur kann nun eine Implementierung für einen spezifischen Anwendungsfall erstellt werden. In dieser Arbeit wird die Infrastruktur auf einen autonomen Verleih von elektrischen Kleinfahrzeugen in Bezug auf das Projekt *SteigtUM* (siehe Sektion 1) angewandt. Der Verleihdienst sieht vor, dass ein Benutzer mit einem Smartphone ein Kleinfahrzeug ausleihen kann. Smartphone und Kleinfahrzeug (Mikrocontroller) kommunizieren mittels BLE miteinander. Um einen Ausleihprozess einzuleiten wird ein Back End in Form eines Servers benötigt, damit ein Verwaltungsmodell realisiert werden kann. Dieses Modell ist jedoch nicht Bestandteil der Arbeit. Dennoch wird für den Prototyp (und für die spätere reale Implementierung) ein Back End benötigt, um den Benutzer bzw. die Smartphone-Anwendung (App) zum Ausleihen eines Fahrzeugs zu autorisieren.

Innerhalb des Prototyps kommunizieren anstelle eines Smartphones und eines Mikrocontrollers vorerst nur zwei Mikrocontroller. Ein Mikrocontroller wird mit dem Verleihfahrzeug assoziiert, während der andere das Smartphone des Benutzers vertritt. Die weiteren Kommunikationsparteien Back End und Zertifizierungsstelle werden simuliert.

Für die Implementierung wurden einige Bedingungen aufgestellt, damit sie das konkrete Modell eines Verleihdienstes nicht einschränkt. Eine Bedingung ist, dass zu Beginn eines Ausleihprozesses die App mit dem Back End kommunizieren kann, bis der Benutzer das Fahrzeug nutzen kann. Danach darf keine weiter bestehende Verbindung zwischen App und Back End vorausgesetzt werden. Zwischen dem Fahrzeug und dem Back End darf nie eine Verbindung vorausgesetzt werden. Wird die Bluetooth-Verbindung zwischen App und Fahrzeug unterbrochen, darf nicht davon ausgegangen werden, dass zwischen App und Back End eine (sichere) Verbindung hergestellt werden kann. Somit bleibt die Implementierung weitestgehend unabhängig von äußeren Einflüssen, die die Verbindungen zum Back End beeinträchtigen können. Außerdem gibt sie nicht vor, welche technischen Voraussetzungen das Fahrzeug für die Kommunikation benötigt (mit Ausnahme von BLE).

4.1 Topologie

Die Topologie entspricht der vorgestellten Topologie der Infrastruktur (siehe Sektion 3.1). Zusätzlich wird ein Back End in Form eines Servers benötigt, um den autonomen Verleih zu steuern. Die Abb. 20 zeigt die Topologie der Implementierung.

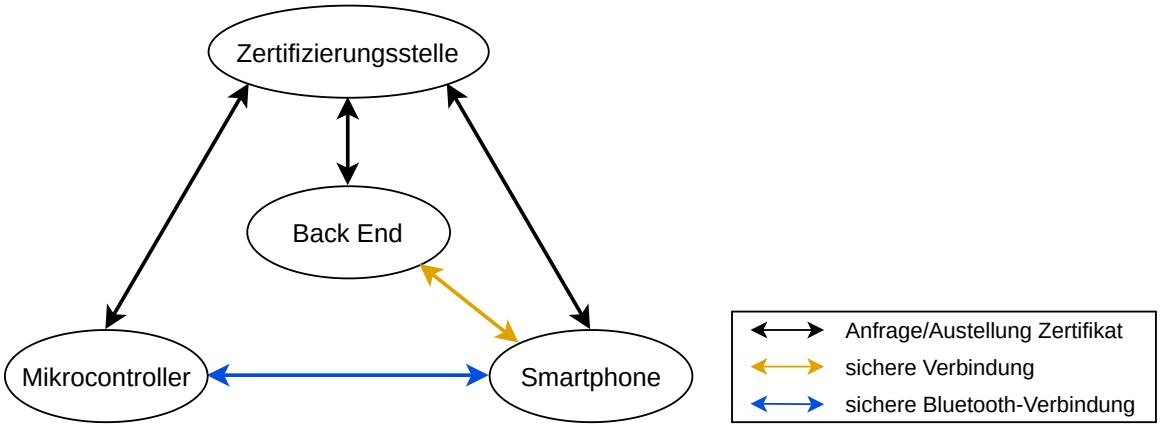

Abb. 20: Topologie der Implementierung

Bevor ein Ausleihprozess stattfinden kann, müssen Back End, Mikrocontroller und Smartphone jeweils über ein von der Zertifizierungsstelle ausgestelltes Zertifikat verfügen. Jeder Partei außer der Zertifizierungsstelle sollte periodisch (z.B. jährlich) ein Zertifikat ausgestellt werden.

4.2 Ausleihprozess

Mit der Anwendung der Infrastruktur wird bereits der Aufbau einer sicheren Verbindung zwischen Smartphone und Mikrocontroller ermöglicht. Ab diesen Punkt muss der Mikrocontroller sicherstellen, dass der Benutzer bzw. das Smartphone dazu autorisiert ist, das Fahrzeug auszuleihen. Dieses Problem wird mit einer vom Back End signierten Bestätigung gelöst, die im Rahmen dieser Arbeit als „Subscription“ bezeichnet wird.

Die Subscription muss folgende Anforderungen erfüllen. Sie muss beweisen, dass sie ausschließlich vom Back End angefertigt wurde (Authentizität). Falls ihr Inhalt von einer anderen Instanz verändert wurde, muss dies bei der Prüfung der Subscription ersichtlich werden (Datenintegrität). Die Informationen innerhalb der Subscription beziehen sich auf den zugehörigen Ausleihvorgang (z.B. Identität des Fahrzeugs/Mikrocontrollers, Zeitstempel).

4.2.1 Verbindungsaufbau und Autorisierung

Bevor die Subscription erstellt bzw. vom Smartphone angefordert wird, herrscht folgende Ausgangssituation. Vor der Inbetriebnahme des Verleihdienstes müssen Smartphone, Back End und Mikrocontroller jeweils über ein individuelles Zertifikat (nach X.509-Standard) und dessen privaten Schlüssel verfügen. Die Zertifikate wurden jeweils von der Zertifizierungsstelle ausgestellt. Zudem kennt auch jede der drei Parteien (Smartphone, Back End, Mikrocontroller) das Root-Zertifikat (das Zertifikat der Zertifizierungsstelle), um damit die anderen Parteien authentifizieren zu können. Außerdem benötigt das Back End ein weiteres Zertifikat mit zugehörigem privaten Schlüssel, das Subscription-Zertifikat genannt wird.

Nun möchte ein Nutzer mit der Smartphone-Anwendung (App) ein Fahrzeug ausleihen. Dafür sollte das Fahrzeug signalisieren, dass es für einen Ausleihvorgang zur Verfügung steht. Ab diesen Punkt ist der weitere Verlauf des Ausleihprozesses in Abb. 21 dargestellt.

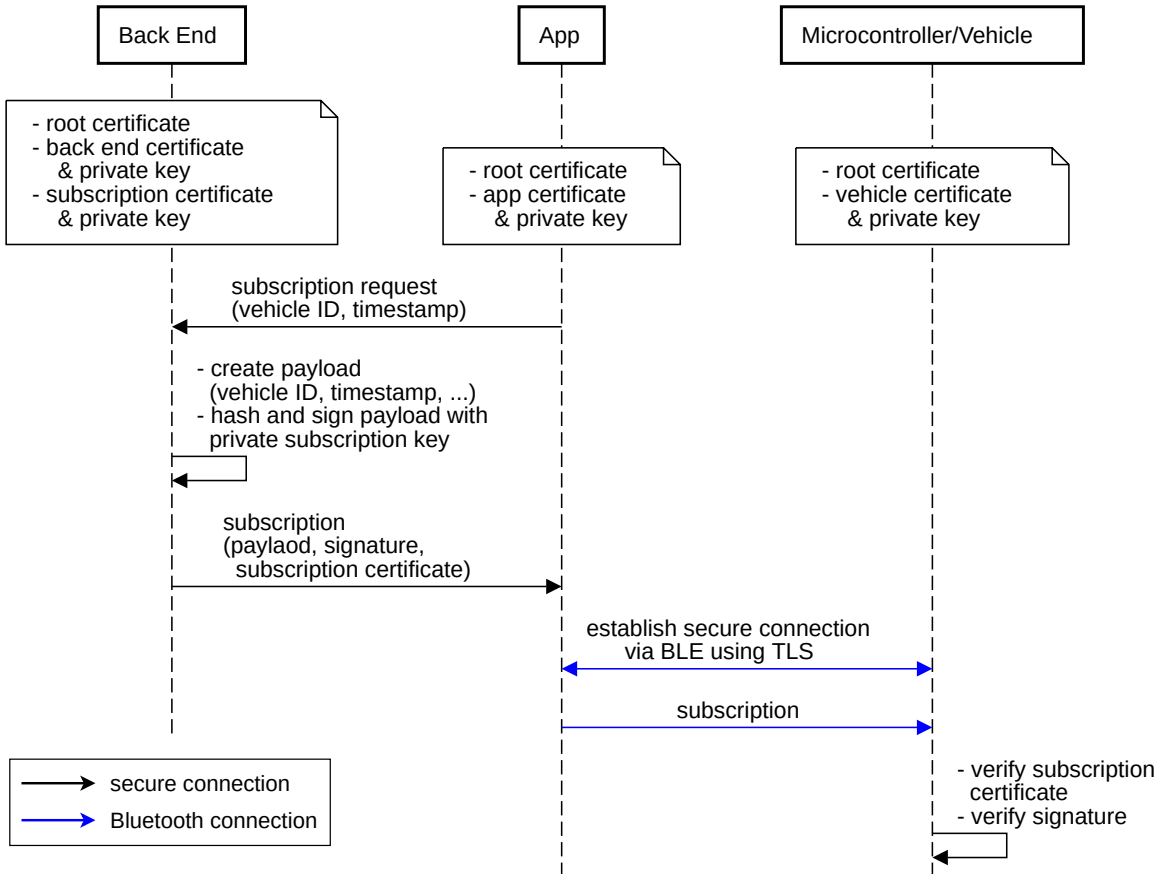

Abb. 21: Verlauf des Ausleihprozzeses

Zu Beginn muss die App die Identität und Bluetooth-Adresse des Fahrzeugs feststellen (z.B. über einen Quick Response Code, kurz QR-Code). Die App baut eine sichere Verbindung zum Back End auf und verlangt nach einer Subscription. Dabei sendet sie die Identität des Fahrzeugs, einen aktuellen Zeitstempel und evtl. weitere Informationen (z.B. bzgl. des Bezahlmodells). Das Back End fügt zu diesen Informationen evtl. noch weitere hinzu (z.B. bzgl. des Bezahlmodells) und formt daraus den „Payload“. Danach wird der Hashwert des Payloads gebildet und mit dem privaten Schlüssel signiert. Die Subscription setzt sich nun entsprechend Abb. 22 zusammen.

Length of Payload (2 Bytes)	Payload (... 65536 Bytes)
Length of Signature (2 Bytes)	Signature (... 65536 Bytes)
Length of Subscription Certificate (2 Bytes)	Subscription Certificate (... 65536 Bytes)

Abb. 22: Aufbau der Subscription

Das erste Längenfeld gibt die Länge des darauffolgenden Payloads an. Das zweite Längenfeld gibt die Länge der Signatur an, worauf die Signatur folgt. Das letzte Längenfeld steht für die Länge des Subscription-Zertifikats und ist gefolgt vom Subscription-Zertifikat. Schließlich überträgt das Back End die Subscription an die App und trennt die Verbindung.

Entsprechend der Sektion 3.4 verbindet sich die App über BLE mit dem Fahrzeug und stellt mittels TLS eine sichere Verbindung her. Daraufhin beendet das Fahrzeug das Advertising. Anschließend sendet die App die Subscription an das Fahrzeug. Zuerst verifziert das Fahrzeug das Subscription-Zertifikat, indem es dessen Signatur mit dem Root-Zertifikat prüft. Danach verifiziert es die Signatur des Payloads mit der gleichen Hash-Funktion, die zur Erstellung der Signatur genutzt wurde, und dem öffentlichen Schlüssel des Subscription-Zertifikats. Sind die Verifikationen erfolgreich, konnte das Fahrzeug sicherstellen, dass die Subscription vom Back End erstellt wurde. Nun müssen noch die Inhalte des Payloads geprüft werden. Demnach muss die angegebene Identität mit der des Fahrzeugs übereinstimmen und der Zeitstempel aktuell sein. Wurden weitere Informationen angegeben, sollte deren Plausibilität geprüft deren Informationsgehalt verarbeitet werden. Somit ist der erste Teil des Ausleihprozesses abgeschlossen. Sollte eine der Verifikationen fehlschlagen sowohl bei der Prüfung der Subscription als auch vorher auf Ebene von TLS, wird die Verbindung abgebrochen.

4.2.2 Erneuter Verbindungsaufbau

Ab diesen Punkt muss ein weiterer Fall für den Verlauf des Ausleihprozesses berücksichtigt werden: der Abbruch der Verbindung zwischen Smartphone und Fahrzeug sowie die darauffolgende Wiederherstellung der Verbindung. Der Verbindungsabbruch kann zum einen durch technische Einflüsse (z.B. Interferenzen) verursacht werden und nicht vom Benutzer beabsichtigt sein. Zum anderen ist es denkbar, dass der Nutzer sich für eine bestimmte Zeitspanne vom Fahrzeug entfernt, wodurch die Verbindung aufgrund der begrenzten Bluetooth-Reichweite abbricht. Evtl. sollte hier der Benutzer dem Fahrzeug signalisieren, dass er für eine bestimmte Zeitspanne abwesend ist. Danach begibt sich der Nutzer wieder zum Fahzeug und möchte es weiter nutzen.

Wenn die Verbindung nun abbricht, beginnt das Fahrzeug mit dem Advertising. Sobald der Nutzer bzw. das Smartphone in Reichweite ist, verbindet sich die App per BLE mit dem Fahrzeug, da es die Bluetooth-Adresse des Fahrzeugs bereits kennt, und baut wie oben beschrieben eine sichere Verbindung auf. Nun sendet die App die alte Subscription erneut an das Fahrzeug. Das Fahrzeug verifiziert erneut die Subscription und vergleicht sie mit der Subscription, die es zu Beginn des Ausleihprozesses empfangen hat. Stimmt sie mit der ursprünglichen Subscription überein, kann das Fahrzeug wieder genutzt werden. Anderenfalls trennt das Fahrzeug die Verbindung und beginnt das Advertising.

4.2.3 Beenden des Ausleihprozesses

Das Beenden des Ausleihprozesses ist mitunter abhängig vom Bezahlmodell des Verleihdienstes. Im Wesentlichen könnte die App dem Fahrzeug eine Nachricht senden, die besagt, dass der Ausleihprozess nun beendet wird. Zudem sollte auch der Fall berücksichtigt werden, wenn keine ordnungsgemäße Beendigung stattfindet. Dabei sollte sich das Fahrzeug nach einer bestimmten Zeitspanne für neue Ausleihprozesse automatisch freigegeben.

4.2.4 Einfluss der Infrastruktur

Da die Zertifikate der Parteien (Back End, App, Fahrzeug) von der Zertifizierungsstelle des Verleihdienstes ausgestellt wurden und die Parteien nur das Root-Zertifikat des Verleihdienstes nutzen, um das Zertifikat eines Gegenüber zu verifizieren, können nur die Parteien TLS-Verbindungen zueinander aufbauen. Jede außenstehende Instanz, die versucht eine Verbindung zu einer der Parteien (Back End, App, Fahrzeug) aufzubauen, wird bereits beim TLS-Handshake abgewiesen. Grund dafür ist, dass die außenstehende Instanz kein von der Zertifizierungsstelle ausgestelltes Zertifikat besitzt. Sollte sie doch an ein solches Zertifikat gelangen, muss sie beim TLS-Handshake immer noch beweisen, dass sie den zugehörigen privaten Schlüssel besitzt. Aus diesem Grund muss jede der Parteien (Back End, App, Fahrzeug) den privaten Schlüssel geheim halten. Das bedeutet auch, dass ein Angreifer nicht in der Lage sein darf, den privaten Schlüssel aus einer der Parteien auszulesen (z.B. bei physischen Zugang). Deshalb sollten Zertifikate und private Schlüssel in Keystores gespeichert werden.

4.3 Prototyp

In diesem Abschnitt werden Details zum Prototyp angegeben. Wie bereits erwähnt, ist der Prototyp nicht für eine Smartphone-Anwendung und einen Mikrocontroller umgesetzt, sondern vorerst nur für zwei Mikrocontroller. Dabei vertritt ein Mikrocontroller die Rolle der Smartphone-Anwendung (Client-Mikrocontroller) und der andere die Rolle des Fahrzeugs (Server-Mikrocontroller)).

Im Anhang 6.3 ist das zugehörige Repository verlinkt.

4.3.1 Hardware und Software

Zur Entwicklung des Prototyps wurde neben einem Personal Computer (PC) folgende Hardware verwendet:

- Mikrocontroller *ESP32-WROOM-32*
- Smartphone mit Android 11.0

Obwohl das Smartphone vorerst nicht Teil des Prototyps ist, wird es bzw. das Betriebssystem *Android* für die Weiterführung berücksichtigt.

4.3.1.1 ESP32-WROOM-32 (Mikrocontroller)

Der *ESP32-WROOM-32* unterstützt Bluetooth 4.2 für BR/EDR und BLE [98]. Der Hersteller *Espressif* stellt das *Espressif Internet of Things Development Framework* (ESP-IDF) [99], mit dem Anwendungen für den *ESP32-WROOM-32* entwickelt werden können. Das ESP-IDF unterstützt unter anderem folgende Software-Komponenten:

- *Free Real Time Operating System* (FreeRTOS)
- *Serial Peripheral Interface Flash Files System* (SPIFFS)
- *nimBLE*
- *mbedTLS*

FreeRTOS dient als Betriebssystem für den *ESP32-WROOM-32*. SPIFFS wird benötigt, um Dateien auf den *ESP32-WROOM-32* zu „flashen“ bzw. um auf diese während der Laufzeit zugreifen zu können.

nimBLE ist eine Software-Bibliothek für den BLE-Host von der *Apache Software Foundation* und bietet Programmierschnittstellen für GAP und L2CAP [100]. Für GAP und L2CAP wird jeweils ein Event Handling ausgeführt. So kann bspw. festgelegt werden, welche Reaktion auf das Entdecken eines anderen Bluetooth-Geräts folgt oder wie mit über L2CAP empfangenen Anwendungsdaten verfahren wird.

mbedTLS ist eine Software-Bibliothek für TLS und eignet sich aufgrund niedriger Anforderungen an den Speicher für eingebettete Systeme. Aktuell wird höchstens TLS 1.2 unterstützt [101].

4.3.1.2 Smartphone/Android

Das Smartphone unterstützt Bluetooth 5.0 und wird mit Android 11.0 betrieben. Wenn es mit dem *ESP32-WROOM-32* per Bluetooth kommuniziert, wird also die Bluetooth-Version 4.2 verwendet. Android unterstützt mit den Programmpaketen „android.bluetooth.le“ [102] BLE und mit „javax.net.ssl“ [103] SSL/TLS. Wie in Sektion 3.3 erläutert, sollten nur TLS 1.2 oder TLS 1.3 genutzt werden. TLS 1.2 wird ab Android 4.1 und TLS 1.3 ab Android 10.0 unterstützt [104].

4.3.2 Simulierung der Zertifizierungsstelle

Die Zertifizierungstelle wird mittels eines Personal Computer (PC) simuliert. Die Zertifikate und Schlüssel werden vom PC mittels *mbedTLS* generiert und beim „Flashen“ des jeweiligen Programmes auf die Mikrocontroller als Dateien übertragen. Jedoch werden sie nicht in Keystores gespeichert. Die Zertifkate werden nach dem Standard X.509 erstellt und die Schlüssel mittels des RSA-Verfahrens generiert. In Abb. 23 wird gezeigt, wie sie erstellt bzw. generiert werden.

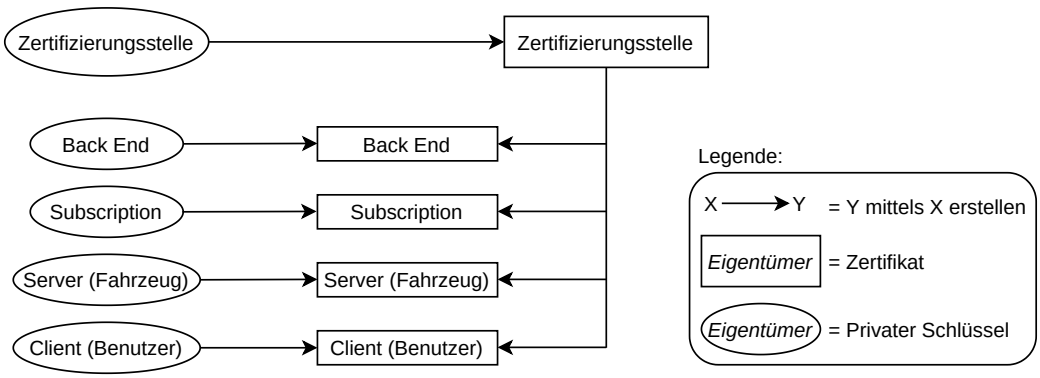

Abb. 23: Generierung der Schlüssel und Erstellung der Zertifikate

Zuerst wird ein 4096 Bit langer privater Schlüssel für die Zertifizierungsstelle generiert. Danach wird das zugehörige Root-Zertifikat mit diesem Schlüssel erstellt. Dabei wird angegeben, dass es ein selbstunterzeichnetes Zertifikat ist und für eine Zertifizierungsstelle ausgestellt wird. Der Name des Ausstellers ist in diesem Fall auch der Name des Eigentümers. Alle weiteren privaten Schlüssel sind ebenfalls 4096 Bit lang. Wurden sie generiert, wird mit jedem privaten Schlüssel genau ein Zertifikat erstellt, dessen Ausstellerzertifikat das Root-Zertifikat

ist.

Nun liegen alle Zertifikate und Schlüssel vor und werden beim „Flashen“ des jeweiligen Programmes folgendermaßen auf die beiden Mikrocontroller übertragen:

- Server-Mikrocontroller (Fahrzeug):
  - privater Server-Schlüssel (Fahrzeug)
  - Server-Zertifikat (Fahrzeug)
  - Root-Zertifikat
- Client-Mikrocontroller (stellvertretend für die App):
  - privater Client-Schlüssel (App)
  - Client-Zertifikat (App)
  - privater Subscription-Schlüssel
  - Subscription-Zertifikat
  - Root-Zertifikat

Der Client erhält zusätzlich den Schlüssel und das Zertifikat zum Erstellen der Subscriptions, um so das Back End zu simulieren.

4.3.3 Ablauf der Anwendung für Client und Server

Entsprechend Abb. 24 ist der erste Teil des Ablaufs der Anwendungen für Client-Mikrocontroller bzw. Server-Mikrocontroller dargestellt. Er ist für beide Anwendungen identisch.

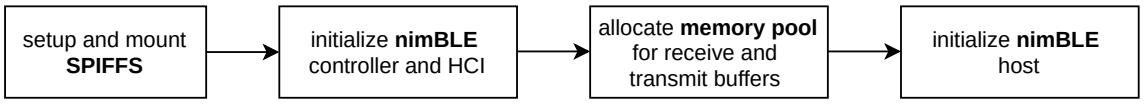

Abb. 24: Ablauf beider Anwendungen (Teil 1)

Zu Beginn wird das Dateisystem SPIFFS konfiguriert und initialisiert. Vor dem „Flashen“ wird eine Partition erstellt, die die jeweiligen Schlüssel und Zertifikate enthält (siehe Sektion 4.3.2). Danach wird über *nimBLE* der Bluetooth-Controller im Modus Low Energy und das Host Controller Interface initialisert. Danach wird Speicher für zwei Memory Pools reserviert: einen Memory Pool für empfangene Anwendungsdaten und einen für zu sendende Anwendungsdaten. Schließlich wird der nimBLE-Host in einem neuen Task initialisiert, um das Event Handling für L2CAP und GAP separiert von der Anwendung zu betreiben.

Die Abb. 25 zeigt den weiteren Ablauf der Server-Anwendung.

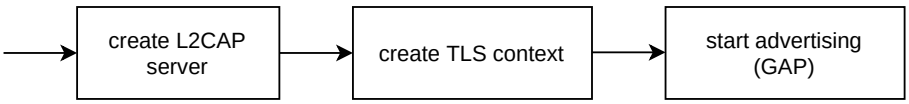

Abb. 25: Ablauf der Server-Anwendung (Teil 2)

Zunächst wird der L2CAP-Server erstellt. Somit wird das L2CAP Event Handling aktiv. Anschließend wird der TLS-Kontext erstellt. Er benötigt den privaten Schlüssel und das Zertifikat des Fahrzeugs sowie das Root-Zertifikat. Zudem verlangt er nach den Funktionen für das Senden bzw. Empfangen von Daten durch den Transport (L2CAP). Danach beginnt die Server-Anwendung mit dem Advertising mithilfe von GAP. Dabei werden Connectable Undirected Advertising Events als Broadcast gesendet, in denen die öffentliche Bluetooth-Adresse des Server-Mikrocontrollers angegeben wird. Nun ist auch das GAP Event Handling aktiv.

In Abb. 26 ist der weitere Ablauf der Client-Anwendung dargestellt.

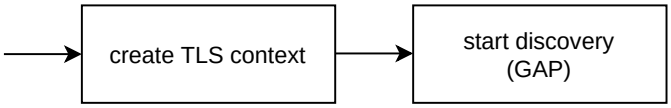

Abb. 26: Ablauf der Client-Anwendung (Teil 2)

Die Client-Anwendung erstellt ebenfalls einen TLS-Kontext, nur dass hier der private Schlüssel und das Zertifikat für den Client und nicht für das Fahrzeug benötigt wird. Danach wird mittels GAP nach Advertising Events gescannt (General Discovery Mode). Somit wird das GAP Event Handling aktiv.

Mittels der beiden Event Handlings bauen Server und Client erst eine Verbindung über GAP auf und anschließend eine Verbindung über einen neuen L2CAP-Channel. Daraufhin wird der TLS-Handshake ausgeführt. Dabei senden die Anwendungen über L2CAP Service Data Units, deren maximale Länge der Maximum Transmission Unit (MTU) entspricht. Empfängt eine Anwendung eine SDU, speichert sie die SDU im Memory Pool für empfangene Anwendungsdaten (RX-Buffer). Der RX-Buffer ist in Blöcke gleicher Länge aufgeteilt, die jeweils eine MTU speichern können. Jede im RX-Buffer gespeicherte SDU nimmt einen dieser Blöcke ein. Möchte nun die Anwendung eine bestimmte Anzahl an Bytes empfangen, prüft sie zunächst, ob sich eine oder mehrere SDUs im RX-Buffer befinden. Sie liest ggf. die geforderte Byteanzahl aus der SDU und lässt einen Pointer auf das nächst zu lesende Byte zeigen. Für den Fall eines leeren RX-Buffers, wartet die Anwendung (mittels Semaphores) bis eine SDU empfangen und in den RX-Buffer geschrieben wurde.

Schlägt der TLS-Handshake fehl, stoppen beide Anwendungen. Anderenfalls erstellt der Client die Subscription mit einem vorgefertigtem Payload. Hat er den Hashwert des Payloads gebildet und signiert, sendet er die Subscription mithilfe des TLS-Kontextes an den Server. Dieser verifiziert die Subscription (siehe Sektion 4.2.1). Wurde die Subscription erfolgreich verifiziert, erfolgt die Bestätigung auf der Konsole. Anderenfalls stoppt die jeweilige Anwendung.

5 Zusammenfassung und Ausblick

Zu Beginn der Arbeit wurde die Funktionsweise von *Bluetooth Low Energy* (BLE) vorgestellt. Die Recherche beschränkte sich auf die Spezifikationen und Webauftritte der Bluetooth SIG, um aus erster Hand ein Verständnis für die BLE-Architektur zu gewinnen. Durch die Untersuchung der Sicherheitsfunktionen und weitere Recherche, konnten die Schwachstellen von BLE aufgedeckt werden.

Zum Entwurf der Infrastruktur musste zunächst ein geeigneter Transport innerhalb der BLE-Architektur bestimmt werden. Die Tatsache, dass sich viele Quellen zu dieser Frage auf das Protokoll GATT beziehen, war irreführend. GATT ist aufgrund seiner Attributsemantik und dem einhergehenden Overhead in der Übertragung ungeeignet für den effizienten Transport von Daten. Stattdessen stellt sich L2CAP als geeignetes Transportprotokoll heraus. Desweiteren war eine Recherche bzgl. der Sicherheitprobleme der Kommunikation und deren Lösungen notwendig. Das TLS-Protokoll stellte sich dabei als hervorragende Lösung heraus, um über einen beliebigen Transport sicher Daten zu übertragen.

Die Implementierung der Infrastruktur für den Verleihdienst wies das Problem der Autorisierung auf. Als Lösung wurde die „Subscription“ entworfen, die sicherstellt, dass ein Nutzer berechtigt ist, ein Fahrzeug auszuleihen, und es nach kurzer Abwesenheit wieder zu nutzen. Der entwickelte Prototyp für die Implementierung zeigt, wie für zwei Mikrocontroller die Infrastruktur angewandt wird und eine Subscription sicher übertragen und verifiziert wird.

Eine Weiterführung der Arbeit wäre in der Implementierung denkbar. Dabei könnte der Prototyp zunächst unter Einbezug eines Smartphones weiterentwickelt werden. Desweiteren könnten die Zertifizierungsstelle und der Back End Server physisch eingebunden werden, um einen realistischen Prototypen zu testen.

Außerdem kann die Arbeit mit einer genauen Kryptoanalyse des vorgestellten Konzepts der „Subscription“ weitergeführt werden, um eventuelle Schwachstellen aufzudecken und auszuschließen.

6 Anhang

6.1 Beispiel: Sequence Number / Next Expected Sequence Number

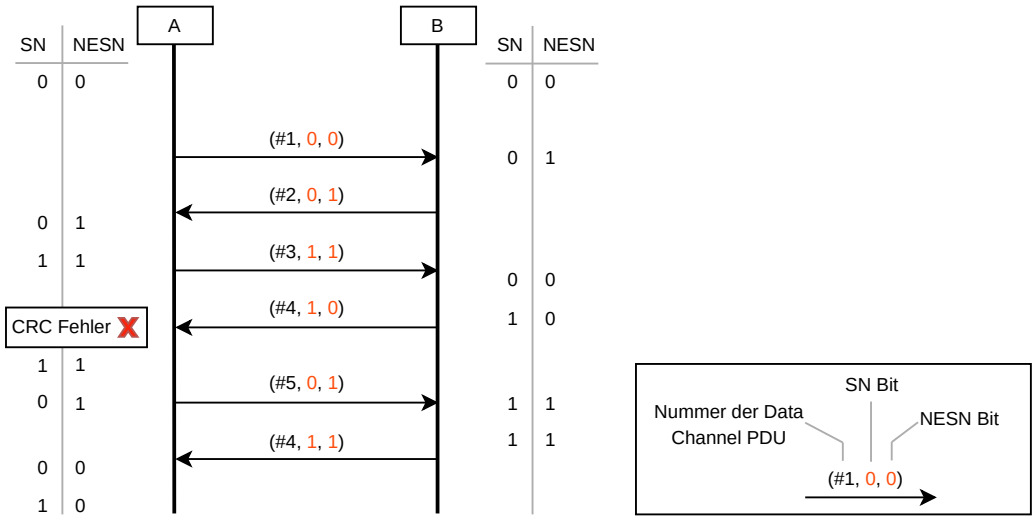

Abb. 27: Beispiel für SN/NESN

Gerät A und B verfügen jeweils über eigene Werte für SN/NESN und setzten diese jeweils auf 0 zu Beginn der Verbindung. Nun senden A und B gegenseitig Data Channel PDUs und nutzen folgenden Algorithmus für die SN/NESN. Die SN/NESN einer Data Channel PDU werden zur SN Bit bzw. NESN Bit genannt. Bei dem erstmaligen Empfang von Data Channel PDU #4 gibt der CRC einen Fehler aus, also wurde das Paket nicht korrekt übertragen.

Algorithmus zum Setzen der SN/NESN [30] (bezieht sich immer nur auf das ausführende Gerät):

Sei „neue Data Channel PDU“ eine Data Channel PDU, die erstmalig gesendet wird (also keine erneute Sendung einer bereits gesendeten Data Channel PDU).
Sei „alte Data Channel PDU“ eine Data Channel PDU, die bereits von dem Gerät gesendet bzw. empfangen wurde.
Sei „letzte Data Channel PDU“ eine Data Channel PDU, die das Gerät zuletzt gesendet hat.

- Senden einer Data Channel PDU:
  1. Setze NESN Bit auf NESN. Falls Data Channel PDU eine neue Data Channel PDU ist, setze SN Bit auf SN
- Empfangen einer Data Channel PDU:
  1. Ist SN Bit gleich NESN, dann wurde eine neue Data Channel PDU empfangen und NESN wird inkrementiert. Anderenfalls ist SN Bit ungleich NESN, also wurde eine alte Data Channel PDU empfangen, die ignoriert wird.
  2. Ist NESN Bit ungleich SN, dann wurde die letzte Data Channel PDU vom gegenüber bestätigt (ACK) und es wird SN inkrementiert. Anderenfalls ist NESN Bit gleich SN, also wurde die letzte Data Channel PDU nicht bestätigt (NAK) und

muss erneut gesendet werden. Dabei muss SN Bit den Wert des SN Bit der letzten Data Channel PDU (also die zuletzt vom Gerät gesendet wurde) annehmen.

6.2 Sequenzdiagramm: Verbindungsaufbau der Infrastruktur

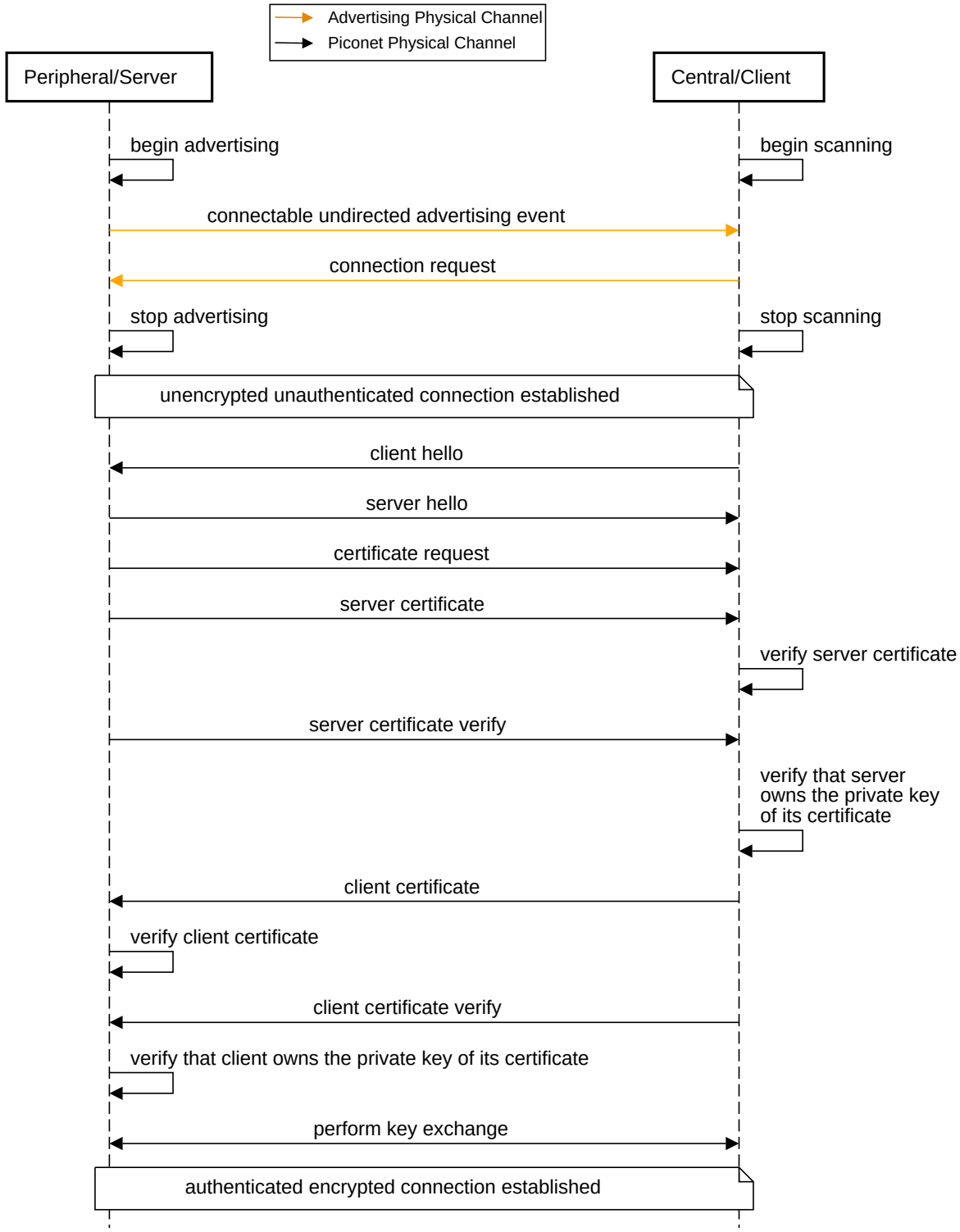

Abb. 28: Verbindungsaufbau der Infrastruktur

6.3 Referenz zum Repository

Unter folgendem Link befindet sich das Repository des Prototypen:
`https://github.com/immrn/secure-ble-for-esp32`

Literatur

[1] Institute of Electrical und Electronics Engineers. *IEEE 802.15.1-2002 - IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*. URL: https://standards.ieee.org/standard/802_15_1-2002.html (besucht am 28.06.2021).

[2] Bluetooth Special Interest Group. *The Bluetooth Range Estimator*. URL: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/ (besucht am 28.06.2021).

[3] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 1.1 Overview of BR/EDR Operation, S. 18 (PDF S. 124). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[4] Bluetooth Special Interest Group. „Bluetooth Core Specification Version 5.2“. In: Bd. Vol 1. 31. Dez. 2019. Kap. Part A, 1.1 Overview of BR/EDR Operation, S. 188 (PDF S. 188). URL: https://www.bluetooth.com/specifications/specs/core-specification/.

[5] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 1 General Description, S. 17 (PDF S. 123). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[6] Bluetooth Special Interest Group. „Bluetooth Core Specification Version 5.2“. In: Bd. Vol 1. 31. Dez. 2019. Kap. Part A, 1 General Description, S. 187 (PDF S. 187). URL: https://www.bluetooth.com/specifications/specs/core-specification/.

[7] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 1.2 Overview of Bluetooth Low Energy Operation, S. 20 (PDF S. 126). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[8] Bluetooth Special Interest Group. „Bluetooth Core Specification Version 5.2“. In: Bd. Vol 1. 31. Dez. 2019. Kap. Part A, 1.2 Overview of Bluetooth Low Energy Operation, S. 190 (PDF S. 190). URL: https://www.bluetooth.com/specifications/specs/core-specification/.

[9] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 1 General Description, Figure 1.1, Figure 1.2, S. 18 (PDF S. 124). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[10] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 2 Core System Architecture, Figure 2.1, S. 31 (PDF S. 137). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[11] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 4.1.2 LE Topology, S. 75 (PDF S. 181). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[12] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 1.2 Overview of Bluetooth Low Energy Operation, S. 21 (PDF S. 127). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[13] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3 Data Transport Architecture, Figure 3.1, S. 39 (PDF S. 145). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[14] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3.2 Transport Architecture Entities, Figure 3.3, S. 45 (PDF S. 151). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[15] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part A, 2 Frequency Bands and Channel Arrangement, S. 16–17 (PDF S. 2180–2181). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[16] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 1.4.1 Advertising and Data Channel Indexes, S. 35 (PDF S. 2199). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[17] Bluetooth Special Interest Group. *How Bluetooth Technology Uses Adaptive Frequency Hopping to Overcome Packet Interference*. URL: how-bluetooth-technology-uses-adaptive-frequency-hopping-to-overcome-packet-interference/ (besucht am 28. 06. 2021).

[18] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3.4 Physical Links, S. 58 (PDF S. 164). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[19] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3.4.3 LE Links Supported by the LE Physical Channels, S. 60–61 (PDF S. 166–167). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[20] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.1 Packet Format, Figure 2.1, S. 38 (PDF S. 2202). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[21] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.1 Packet Format, S. 36–37 (PDF S. 2200–2201). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[22] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 3.2 Data Whitening, S. 53–54 (PDF S. 2217–2218). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[23] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.3 Advertising Channel PDU, Figure 2.2, S. 37 (PDF S. 2201). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[24] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.3 Advertising Channel PDU, Figure 2.3, S. 38 (PDF S. 2202). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[25] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.3 Advertising Channel PDU, S. 37–44 (PDF S. 2201–2208). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[26] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.4 Data Channel PDU, Figure 2.12, S. 44 (PDF S. 2208). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[27] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.4 Data Channel PDU, Figure 2.13, S. 44 (PDF S. 2208). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[28] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 2.4 Data Channel PDU, S. 44–45 (PDF S. 2208–2209). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[29] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 6. 2. Dez. 2014. Kap. Part B, 2.4 Data Channel PDU, S. 46–47 (PDF S. 2589–2590). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[30] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part B, 4.5.9 Acknowledgement and Flow Control, S. 75–77 (PDF S. 2239–2241). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[31] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3.5.4.6 LE Asynchronous Connection (LE ACL), 3.5.4.7 LE Advertising Broadcast (ADVB), S. 68–69 (PDF S. 174–175). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[32] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 3.5.5.2 LE Logical Links, S. 70–71 (PDF S. 176–177). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[33] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 2 Core System Architecture, S. 32 (PDF S. 138). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[34] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part A, 2.4 Modes of Operation, S. 43 (PDF S. 1735). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[35] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part A, 2.4 Modes of Operation, S. 41 (PDF S. 1401). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[36] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part A, 1.1 L2CAP Features, Figure 1.1, S. 31 (PDF S. 1391). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[37] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part A, 1.1 L2CAP Features, S. 30 (PDF S. 1390). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[38] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part A, 1.4 Terminology, Table 1.1, S. 35 (PDF S. 1727). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[39] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 2.1.1 Host Architectural Blocks, S. 30 (PDF S. 185). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[40] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part A, 1.4 Terminology, S. 33–34 (PDF S. 1725–1726). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[41] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 3.1 Connection-Oriented Channels in Basic L2CAP Mode, Figure 3.1, S. 45 (PDF S. 1737). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[42] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 3.4 Connection-Oriented Channels in LE Credit Based Flow Controle Mode, Figure 3.6, S. 55 (PDF S. 1747). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[43] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part A, 4.24 LE Flow Control Credit (Code 0x16), S. 88 (PDF S. 1780). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[44] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part F, 3.1 Introduction, S. 475 (PDF S. 1835). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[45] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part F, 3.4.4.3 Read Request, 3.4.4.4 Read Response, S. 494–495 (PDF S. 1854–1855). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[46] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part F, 3.4.5.1 Write Request, 3.4.5.2 Write Response, S. 501–503 (PDF S. 1861–1863). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[47] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part G, 2.5.1 Overview, Figure 2.3, S. 528 (PDF S. 1888). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[48] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part G, 2.5.1 Overview, Figure 2.4, S. 529 (PDF S. 1889). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[49] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part G, 2.5 Attribute Protocol, S. 528–529 (PDF S. 1888–1889). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[50] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part G, 2.6.1 Overview, Figure 2.5, S. 532 (PDF S. 1892). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[51] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part C, 2.2.2 Roles when Operating over an LE Physical Channel, S. 278–279 (PDF S. 1638–1639). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[52] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part C, 9.1 Broadcast Mode and Observation Procedure, S. 335–337 (PDF S. 1695–1697). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[53] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part C, 9.2 Discovery Modes and Procedures, S. 337 (PDF S. 1697). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[54] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part C, 9.3 Connection Modes and Procedures, S. 344–359 (PDF S. 1704–1718). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[55] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part C, 9.4 Bonding Modes and Procedures, S. 368–370 (PDF S. 2060–2062). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[56] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 6. 2. Dez. 2014. Kap. Part B, 1.3.1 Public Device Address, S. 33 (PDF S. 2576). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[57] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.2.2 Random Address Hash function ah, S. 595 (PDF S. 2287). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[58] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 6. 2. Dez. 2014. Kap. Part B, 1.3.2 Random Device Address, S. 34–36 (PDF S. 2577–2579). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[59] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part C, 10.7.3 Privacy Feature in a Broadcaster, 10.7.4 Privacy Feature in an Observer, S. 386–387 (PDF S. 2078–2079). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[60] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part C, 10.7.1 Privacy Feature in a Peripheral, S. 385 (PDF S. 2077). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[61] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part C, 10.7.2 Privacy Feature in a Central, S. 386 (PDF S. 2078). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[62] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part H, 1.1 Scope, Figure 1.1, S. 598 (PDF S. 1958). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[63] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 6. 30. Juni 2010. Kap. Part E, 1 Encryption and Authentication Overview, S. 121 (PDF S. 2285). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[64] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part H, 2.3.2 IO Capabilities, Table 2.1, S. 605 (PDF S. 1965). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[65] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 3. 30. Juni 2010. Kap. Part H, 2.3.2 IO Capabilities, Table 2.2, S. 605 (PDF S. 1965). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[66] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.4.1 Association Models, S. 93 (PDF S. 248). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[67] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.1 Selecting Key Generation Method, Table 2.8, S. 609 (PDF S. 2301). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[68] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.2.4.1 Numeric Comparison, S. 89–90 (PDF S. 244–245). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[69] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.2.4.2 Just Works, S. 90 (PDF S. 245). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[70] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.2.4.3 Out of Band, S. 91 (PDF S. 246). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[71] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.2.4.4 Passkey Entry, S. 91–92 (PDF S. 246–247). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[72] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.3 LE Legacy Pairing - Passkey Entry, S. 612 (PDF S. 2304). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[73] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.2.3 Confirm value generation function c1 for LE Legacy Pairing, S. 596 (PDF S. 2288). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[74] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.5 LE Legacy Pairing Phase 2, S. 613–614 (PDF S. 2305–2306). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[75] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.2.4 Key generation function s1 for LE Legacy Pairing, S. 598 (PDF S. 2290). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[76] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.6.1 Public Key Exchange, S. 615 (PDF S. 2307). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[77] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.6.2 Authentication Stage 1 – Just Works or Numeric Comparison, S. 617 (PDF S. 2309). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[78] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.6.3 Authentication Stage 1 – Passkey Entry, S. 619 (PDF S. 2311). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[79] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.6.4 Authentication Stage 1 – Out of Band, S. 620–621 (PDF S. 2312–2313). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[80] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.2.7 LE Secure Connections Key Generation Function f5, S. 600–601 (PDF S. 2292–2293). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[81] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.6.5 Authentication Stage 2 and Long Term Key Calculation, S. 622 (PDF S. 2314). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[82] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.2 LE Legacy Pairing - Just Works, S. 612 (PDF S. 2304). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[83] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 3. 2. Dez. 2014. Kap. Part H, 2.3.5.4 Out of Band, S. 613 (PDF S. 2305). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[84] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.2“. In: Bd. Vol 1. 2. Dez. 2014. Kap. Part A, 5.4 LE SECURITY, S. 613 (PDF S. 2305). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-2/.

[85] Bluetooth Special Interest Group. „Bluetooth Core Specification Version 5.2“. In: Bd. Vol 1. 31. Dez. 2019. Kap. Part A, 5.4 LE Security, S. 277 (PDF S. 277). URL: https://www.bluetooth.com/specifications/specs/core-specification/.

[86] Bluetooth Special Interest Group. „Bluetooth Specification Version 4.0“. In: Bd. Vol 1. 30. Juni 2010. Kap. Part A, 5.2.3 Encryption, S. 90 (PDF S. 196). URL: https://www.bluetooth.com/specifications/specs/core-specification-4-0/.

[87] Open Handset Alliance. *Bluetooth low energy.* URL: https://developer.android.com/guide/topics/connectivity/bluetooth-le (besucht am 28. 06. 2021).

[88] Pallavi Sivakumaran und Jorge Blasco. „A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape“. In: *28th USENIX Security Symposium (USENIX Security 19).* Santa Clara, CA: USENIX Association, Aug. 2019, S. 3–5. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/sivakumaran (besucht am 28. 06. 2021).

[89] Roland Bless u. a. *Sichere Netzwerkkommunikation. Grundlagen, Protokolle und Architekturen.* 1. Aufl. 2005. Kap. 2.6 Sicherheitsziele in Netzwerken, S. 19–20. ISBN: 978-3-540-27896-2. DOI: https://doi.org/10.1007/3-540-27896-6.

[90] Roland Bless u. a. *Sichere Netzwerkkommunikation. Grundlagen, Protokolle und Architekturen.* 1. Aufl. 2005. Kap. 2.6 Sicherheitsziele in Netzwerken, S. 19. ISBN: 978-3-540-27896-2. DOI: https://doi.org/10.1007/3-540-27896-6.

[91] Christian Baun. *Computer Networks / Computernetze. Bilingual Edition: English – German / Zweisprachige Ausgabe: Englisch – Deutsch.* 1. Aufl. 2019. Kap. 4 Protokolle und Schichtenmodelle, S. 36–40. ISBN: 978-3-658-26356-0. DOI: https://doi.org/10.1007/978-3-658-26356-0.

[92] Bluetooth Special Interest Group. *Erfahren Sie Alles Über Bluetooth Datenübertragung.* URL: https://www.bluetooth.com/de/learn-about-bluetooth/solutions/data-transfer/ (besucht am 01. 07. 2021).

[93] Open Handset Alliance. *Bluetooth LE Connection Oriented Channels (CoC).* URL: https://developer.android.com/about/versions/10/features#bluetooth-le-coc (besucht am 02. 07. 2021).

[94] Apple. *Core Bluetooth.* URL: https://developer.apple.com/documentation/corebluetooth (besucht am 02. 07. 2021).

[95] Apple. *CBL2CAPChannel.* URL: https://developer.apple.com/documentation/corebluetooth/cbl2capchannel (besucht am 02. 07. 2021).

[96] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3.* Techn. Ber. RFC Editor, Aug. 2018, S. 24–33, S. 61–62, S. 77–78. DOI: 10.17487/RFC8446. URL: https://rfc-editor.org/rfc/rfc8446.txt (besucht am 04. 07. 2021).

[97] Kathleen Moriarty und Stephen Farrell. *Deprecating TLS 1.0 and TLS 1.1.* Techn. Ber. RFC Editor, März 2021. DOI: 10.17487/RFC8996. URL: https://datatracker.ietf.org/doc/rfc8996/ (besucht am 05. 07. 2021).

[98] Ltd. Espressif Systems (Shanghai) Co. *ESP32-WROOM-32 Datasheet.* Version 3.1. 2021, S. 6. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (besucht am 05. 07. 2021).

[99] Ltd. Espressif Systems (Shanghai) Co. *ESP-IDF Programming Guide.* URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/ (besucht am 05. 07. 2021).

[100] Apache Software Foundation. *Repository zu apache/mynewt-nimble.* URL: https://github.com/apache/mynewt-nimble (besucht am 05. 07. 2021).

[101] Arm Limited. *Core Features*. URL: https://tls.mbed.org/core-features (besucht am 05. 07. 2021).

[102] Open Handset Alliance. *android.bluetooth.le*. URL: https://developer.android.com/reference/android/bluetooth/le/package-summary?hl=en (besucht am 05. 07. 2021).

[103] Open Handset Alliance. *javax.net.ssl*. URL: https://developer.android.com/reference/javax/net/ssl/package-summary?hl=en (besucht am 05. 07. 2021).

[104] Open Handset Alliance. *SSLContext*. URL: https://developer.android.com/reference/javax/net/ssl/SSLContext (besucht am 05. 07. 2021).