

---

# Sicherheitsarchitektur in einem drahtlosen Sensornetzwerk

---

Author:

Paul Machemehl

Betreuer:

Dipl.-Ing. (FH) Thomas Bartzsch HTW

Dipl.-Ing. Axel Schmidt IfN TUD

Verantwortlicher Hochschullehrer:

Prof. Dr.-Ing. habil. A. Finger

Ausgehändigt am:

02.04.2012

Einzureichen bis:

31.08.2012

---

Dieses Werk steht unter der Creative Commons Lizenz. Sämtlicher Inhalt, sowie die sich darin befindlichen vom Author selbst erstellten Grafiken dürfen frei verbreitet und wiederverwendet werden, auch unter dessen Abwandlung, sowie gewerbliche Zwecke. In jedem Fall ist allerdings anzugeben, woher das verwendete Material stammt und dass darauf stehts im vollem Umfang und kostenlos zugegriffen werden kann.



**creative  
commons**



# INHALTSVERZEICHNIS

<b>Akronyme</b>	<b>5</b>
<b>1. Einführung</b>	<b>7</b>
1.1. Motivation . . . . .	7
1.2. Aufgabenstellung . . . . .	9
<b>2. Grundlagen - Kryptographie und WPANs</b>	<b>11</b>
2.1. Ziele der Kryptographie . . . . .	11
2.2. Integrität und Authentizität mittels MAC/MIC . . . . .	12
2.3. Kryptosysteme . . . . .	13
2.3.1. Ziele und Eigenschaften . . . . .	13
2.3.2. Chiffrearten . . . . .	14
2.3.3. AES - Ein symmetrischer Blockchiffre . . . . .	17
2.4. Verschlüsselungsmodi . . . . .	17
2.4.1. Sinn und Zweck von Verschlüsselungsmodi . . . . .	17
2.4.2. ECB - Electronic Code Book Mode . . . . .	18
2.4.3. CBC - Cipher Block Chaining Mode . . . . .	18
2.4.4. CBC-MAC . . . . .	20
2.4.5. CTR - Counter Mode . . . . .	20
2.4.6. CCM/CCM* - Counter with CBC-MAC . . . . .	22
2.5. WPAN-Standards . . . . .	25
2.5.1. IEEE 802.15.4 . . . . .	25
2.5.2. ZigBee . . . . .	27
<b>3. Konzept und Szenarien</b>	<b>29</b>
3.1. Technische Randbedingungen und Angriffsszenarien . . . . .	29
3.2. Energieverbrauch . . . . .	32
3.3. Versuchsaufbau . . . . .	33

## *Inhaltsverzeichnis*

<b>4. Implementierung</b>	<b>35</b>
4.1. ATTEL MAC Stack 802.15.4 . . . . .	35
4.1.1. Verwendete Hardware . . . . .	35
4.1.2. Konfiguration der Software und der Entwicklungsumgebung . . . . .	36
4.1.3. Programmierung des MAC Stack . . . . .	38
4.1.4. Zugriff auf die Security-Schicht . . . . .	42
4.1.5. Programmablauf . . . . .	49
4.1.6. Aufbau der versendeten Daten . . . . .	54
4.1.7. Erstellung des Quellcodes . . . . .	56
4.1.8. Anpassen des MAC Quelltexts für die Energiemessung . . . . .	63
4.2. ZigBee . . . . .	67
<b>5. Funktionstest und Energiemessung</b>	<b>71</b>
5.1. ATTEL MAC Stack . . . . .	71
5.1.1. Aufbau des Sensornetzwerks für den Funktionstest . . . . .	71
5.1.2. Ergebnisse der Energiemessung . . . . .	75
5.2. BitCloud . . . . .	80
<b>6. Zusammenfassung und Ausblick</b>	<b>83</b>
<b>A. CD-Inhalt</b>	<b>85</b>
<b>B. Screenshots von Wireshark</b>	<b>87</b>
<b>C. Programm-Ablauf-Plan</b>	<b>91</b>
C.1. Device . . . . .	91
C.2. Coordinator . . . . .	100
<b>Formelverzeichnis</b>	<b>109</b>
<b>Abbildungsverzeichnis</b>	<b>112</b>
<b>Tabellenverzeichnis</b>	<b>113</b>
<b>Literaturverzeichnis</b>	<b>116</b>
<b>Index</b>	<b>120</b>
<b>Glossar</b>	<b>121</b>

# AKRONYME

AES	Advanced Encryption Standard.
API	Application Programming Interface (Applikations-Programmierungs-Schnittstelle).
CBC	Cipher Block Chaining.
CBC-MAC	CBC-Message Authentication Code.
CCM	Counter with CBC-MAC.
CCM*	Enhanced Counter with CBC-MAC.
CTR	Counter (Zähler).
ECB	Electronic Code Book.
FFD	Full Function Device (Gerät mit vollem Funktionsumfang).
HAL	Hardware Abstraction Layer (Hardware-Abstraktions-Schicht).
HMAC	Keyed-Hash Message Authentication Code.
IEEE	Institute of Electrical and Electronics Engineers.
ISO	International Standards Organization.
IV	Initialisierungsvektor.
LAN	Local Area Network.
LR-WPAN	low-rate WPAN (langsame WPAN).
MAC	Media Access Control (Medium-Zugriffs-Kontrolle).
MAC	Message Authentication Code.

## Akronyme

MCPS	MAC Common Part Layer.
MIC	Message Integrity Check.
MLME	MAC Layer Management Entity.
PAN	Personal Area Network (Persönliches Umgebungsnetzwerk).
PHY	Physical Layer (Physikalische Schicht).
PIB	PAN Information Base (PAN Informationsdatenbank).
RCB	Radio Controller Board.
RFD	Reduced Function Device (Gerät mit reduziertem Funktionsumfang).
SAP	Service Access Point (Zugriffspunkt für Dienste).
SIO	Serial I/O support.
SKKE	Symmetric-Key Key Establishment.
UART	Universal Asynchronous Receiver Transmitter.
WPAN	Wireless Personal Area Networks (Drahtloses Persönliches Umgebungsnetzwerk).
XOR	exklusives Oder.

# 1. EINFÜHRUNG

## 1.1. Motivation

### Datenfunk statt Datenleitungen - Warum eigentlich?

In den letzten Jahren sind die Kupferpreise deutlich nach oben gestiegen, der Rohstoff wird immer knapper und der Bedarf steigt stetig.<sup>1</sup> Dabei wächst die Nachfrage und der Wert so stark, dass selbst vor dem Diebstahl von Kupferleitungen zur Signalübertragung nicht mehr halt gemacht wird. So wurden in Berlin mehrere Meter Kupferkabel geklaut, deren Aufgabe das Steuern der Bahnanlagen war.<sup>2</sup>

Auch die Informationselektronik hat am Kupfermangel einen enormen Anteil, werden Kupferleitungen doch vornehmlich dafür genutzt, um Informationen von einem Punkt zum anderen zu transportieren und das möglichst schnell. Man denke an LAN<sup>3</sup>-Kabel, die den gesamten Haushalt miteinander vernetzen. Dazu kommen aber auch andere Elektronische Steuerelemente wie Lichtschalter, die mit dem gesamten Stromnetz verbunden sind, nur um eine Lampe an- oder auszuschalten. Dafür werden viele Meter kostbaren Kupfers verwendet, welches in anderen Bereichen wichtiger, weil unverzichtbar ist. Es stellt sich die Frage nach alternativen Übertragungsmedien, die kostengünstiger und besser verfügbar sind. Und was wäre da besser geeignet als der freie Raum, der uns in Unmengen umgibt?

Die Datenübertragung mit Hilfe elektromagnetischer Wellen, die durch den Raum geleitet werden, hat zweifelsohne enorme Vorteile: So sind die elektronischen Steuerelemente nicht an einen Ort gebunden, sondern können (bis zu gewissen Grenzen) frei in der Um-

---

<sup>1</sup><http://www.handelsblatt.com/finanzen/rohstoffe-devisen/rohstoffe/rohstoffe-warum-kupfer-immer-teurer-wird/3744360.html>

<sup>2</sup><http://www.bz-berlin.de/archiv/16-meter-kupferkabel-geklaut-zehntausende-im-s-bahn-stau-article1469836.html>

<sup>3</sup>Siehe Local Area Network (LAN) unter den Akronymen

## 1. Einführung

gebung platziert oder herum getragen werden. Außerdem können enorme Mengen an Kupfer gespart werden. Auch das nachträgliche Einbinden neuer Knoten und Geräte ist deutlich einfacher und flexibler, da keine neuen Leitungen gelegt und installiert werden müssen.

Dabei hat sich die Datenübertragung mittels Funk nicht nur in lokalen Arealen unserer täglichen Umgebung durchgesetzt, auch in viel kleineren Bereichen haben Funkwellen - zumindest in der Theorie - Einzug gehalten, so z.B. in Mikrochips, in denen einzelne Bestandteile mittels Funk über Strecken im Nanometer-Bereich kommunizieren, und das mit Geschwindigkeiten, die mit denen von Kupferleitungen gleichauf liegen.<sup>4</sup>

Allerdings ändern sich die technischen Randbedingungen enorm durch die Verwendung des Raums als sogenanntes *Shared Medium*, welches mehrere Teilnehmer zugleich benutzen können. Dies erfordert ein neues Sicherheitskonzept, in welches die neuen technischen Randbedingungen eingehen müssen.

## Sicherheitsmechanismen im sozialen Umfeld

Wer einmal lügt, dem glaubt man nicht und wenn er auch die Wahrheit spricht!

Hinter diesem allseits bekannten Sprichwort versteckt sich ein kompliziertes Geflecht aus sozialen und moralischen Mechanismen und Abläufen, die uns helfen in unserer Umwelt mit unseren Mitmenschen zurechtzukommen. Wir brauchen sie um zu entscheiden, wem wir vertrauen können und wem nicht. Dabei haben wir hier kein starres Geflecht vor uns. Mit der Zeit ändern sich unsere Beziehungen zu unseren Mitmenschen, da sich auch unsere Mitmenschen und wir selbst uns verändern. Das macht es notwendig uns immer wieder aufs Neue anzupassen und die Entscheidung, wen wir an uns heran lassen wollen und wen nicht, von Tag zu Tag neu zu treffen. Das oben erwähnte Sprichwort spiegelt einen dieser Mechanismen wider.

In technischen Sensor-Netzwerken, die aus mehreren Knoten (Routern und Endgeräten) und - zumindest durch unsere Randbedingungen erzwungen - einem Koordinator bestehen, ist es ebenfalls notwendig zu wissen, welchen Knoten wir in unser Netzwerk aufnehmen wollen und welchen nicht. Im Gegensatz zu dem menschlichen Pendant hat die von uns verwendete Hardware und Software nicht die Möglichkeit nach Gefühl zu entscheiden, sondern braucht einen Algorithmus um diese Entscheidung aufbauend auf verifizierbaren Daten zu treffen. Letztendlich wird auf mathematische Hilfsmittel zurückgegriffen, um darauf unser Sicherheitsmechanismen aufzubauen. Dabei findet eine Kombination mathematischer Verfahren Verwendung, um das gesamte Spektrum an Angriffsszenarien abdecken zu können. Sollte es also passiert sein, dass verschiedene Angriffsszenarien übersehen worden sind, muss das Sammelsurium unserer Sicherheitsmechanismen erweitert und angepasst werden. Diese Situation endet im allgemeinen in einem Katz- und Mausspiel zwischen Angreifer und Verteidiger.

---

<sup>4</sup><http://heise.de/-931165>

## *1.2. Aufgabenstellung*

### **1.2. Aufgabenstellung**

Laut Aufgabenstellung sollen der AVR2025 MAC Stack, welcher den offenen Standard IEEE 802.15.4 implementiert, und BitCloud, die Implementierung des ZigBee Standards, miteinander verglichen werden. Dabei stehen vor allem die Kriterien Energieeffizienz, Ressourcennutzung und Sicherheit im Vordergrund

Außerdem soll die Security Tool Box - STB evaluiert werden. Mit Hilfe eines Programm-Ablauf-Plans sollen die Implementierungen durch ein eigenes Softwaremodul dargestellt werden. Geplant ist außerdem, eine Messung des Energieverbrauchs in Abhängigkeit davon, ob das Netzwerk verschlüsselt wird oder nicht.

## 1. Einführung

# TECHNISCHE UNIVERSITÄT DRESDEN FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

## Aufgabenstellung für die Studienarbeit

von Herrn Paul Machemehl

**Thema:** Sicherheitsarchitektur in einem drahtlosen Sensornetzwerk

**Zielsetzung:**

In den weltweit offenen Standards IEEE 802.15.4 und ZigBee für die Funkdatenübertragung in Sensor- und Aktor-Netzwerken sind energieeffiziente Sicherheitsfunktionen erforderlich. Der Verschlüsselungsstandard AES bietet dazu verschiedene Möglichkeiten, die im Rahmen dieser Studienarbeit untersucht werden sollen.

Es ist eine Untersuchung und ein Vergleich der beiden Atmel-Stacks "Bitcloud" (zur Anbindung von Zigbee) und „AVR2025 MAC“ in Bezug auf bereits integrierte Krypto-Softwaremodule, die auf den verfügbaren Hardware-AES-Block zugreifen, vorzunehmen. Die Untersuchung soll nach den Kriterien erfolgen:

- Energieeffizienz
- Ressourcennutzung
- Sicherheit

Atmel stellt für den Zugriff auf die Verschlüsselungshardware die Software „Atmel-Security Toolbox“ bereit. Diese Schnittstelle ist im Weiteren zu evaluieren. Mit den gewonnenen Erkenntnissen sollen Szenarien/Prozeduren zur Sicherheit (z.B. Schlüsselaustausch, Schlüssellänge) entwickelt werden, die sich im Grad der Sicherheit und des Ressourcenverbrauchs unterscheiden. Die erstellten Prozeduren sollen im nächsten Schritt als eigenständige Softwaremodule für den Mikrocontroller umgesetzt werden. Dabei soll folgender Ablauf beachtet werden:

- Programmablaufplan (PAB) erstellen
- Szenarienablaufplan verifizieren
- Softwaremodul erstellen
- Testapplikation erstellen
- Testmessung durchführen (Datendurchsatz, Latenzzeiten mit Hilfe von IEEE 802.15.4-Sniffer)

Als Hardwarebasis soll der ATmega128RFA1 mit vorhandenem Quellcode für MAC-Stack und Bitcloud-Stack dienen. Als Programmierumgebung sind "AVR-Studio" bzw. „Eclipse“ vorgesehen.

**Betreuer:** Dipl.-Ing. Axel Schmidt IfN TUD  
Dipl.-Ing. (FH) Thomas Bartzsch HTW

**Ausgehändigt am:** 02.4.2012  
**Einzureichen bis:** 31.8.2012

  
Prof. Dr.-Ing. habil. A. Finger

**Verantwortlicher Hochschullehrer**

# 2. GRUNDLAGEN - KRYPTOGRAPHIE UND WPANS

## 2.1. Ziele der Kryptographie

Die folgende Liste zeigt die Ziele auf, welche die Kryptographie zu erreichen versucht:

**Vertraulichkeit** Es soll sichergestellt sein, dass wirklich nur derjenige die Nachricht empfangen und lesen kann, für den sie auch bestimmt war.

**Integrität** Der Empfänger soll feststellen können, ob die Daten oder die Nachricht nach ihrer Erzeugung verändert wurden.

**Authentizität** Absender oder Urheber von Daten bzw. Nachrichten sollen identifizierbar sein, bzw. der Empfänger soll nachprüfen können, wer der Urheber ist.

**Verbindlichkeit** Der Urheber soll nicht abstreiten können, dass er auch der Urheber der Daten/Nachricht ist.

Versucht man diese vier Punkte zu erfüllen, ist es möglich den meisten Anforderungen einer Sicherheitsarchitektur gerecht zu werden und den größten Teil der Angriffsszenarien zu verhindern. Nachfolgend wird in diesem Kapitel auf mehrere mathematische Verfahren eingegangen, mit denen alle vier Ziele in akzeptablem Maße erreicht werden können.

## 2.2. Integrität und Authentizität mittels MAC/MIC

Der Message Authentication Code (MAC)<sup>1</sup> und der Message Integrity Check (MIC)<sup>2</sup> sind sich in ihrer Funktion zum größten Teil gleich und unterscheiden sich nur in einigen wenigen Punkten. Beide stellen eine Art Prüfsumme dar, mit der die Integrität und Authentizität einer Nachricht nachgewiesen werden kann. Das bedeutet, es kann überprüft werden, ob die Nachricht nachträglich auf ihrem Weg manipuliert worden ist und ob die Nachricht wirklich vom angegebenen Sender stammt.

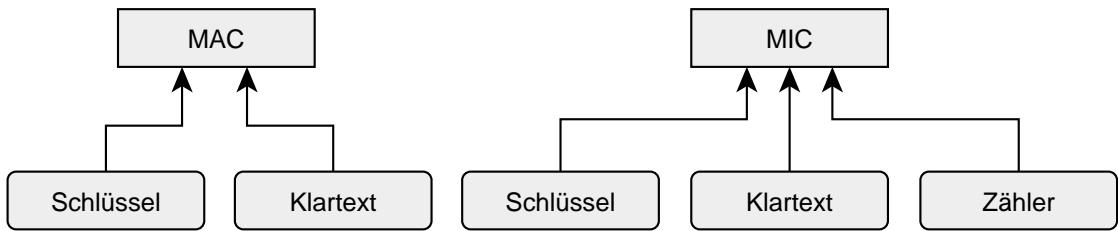


Abbildung 2.1.: Eingabe zur Berechnung der MAC/MIC

Dazu wird zur Berechnung der MAC/MIC der zu sichernde Klartext und ein geheimer Schlüssel, welcher nur den beiden Kommunikationsteilnehmern bekannt sein darf, verwendet. Wie in Abbildung 2.1 zu erkennen ist, dienen beide als Eingabe der Berechnung und mensch erhält einen eindeutigen Wert, der vom Klartext und vom Schlüssel abhängig ist. Der Unterschied zwischen dem MAC und dem MIC besteht in einem zusätzlichen Parameter: Einem Zähler. Dieser wird beim Senden von mehreren Datenpaketen kontinuierlich um eins erhöht und geht in die Berechnung des MIC mit ein. Dies verhindert zahlreiche Arten von Angriffen, denen ein Netz ausgeliefert ist.

Es gibt verschiedene Methoden, eine MAC/MIC zu erstellen. Eine Möglichkeit besteht darin mittels einer *Kryptographischen Hashfunktion*<sup>3</sup> einen *Hashwert* des Klartextes zu berechnen und diesen Hashwert mit einem Schlüssel, den nur die Kommunikationsteilnehmer kennen, zu verschlüsseln. Der Empfänger kann die MAC/MIC dann mit dem selben Schlüssel entschlüsseln und den erhaltenen Haswert mit dem selbst berechneten Hashwert des Klartextes vergleichen. Stimmen beide Werte überein, kann die Nachricht nicht nachträglich verändert worden sein, da ein Angreifer zum Fälschen des MAC/MIC den Schlüssel benötigt. Die Nachricht muss vom angegebenen Sender kommen, da nur dieser den Schlüssel hat, um den Hashwert zu verschlüsseln. Dieses Verfahren wird Keyed-Hash Message Authentication Code (HMAC) genannt.<sup>4</sup>

<sup>1</sup>[http://de.wikipedia.org/wiki/Message\\_Authentication\\_Code](http://de.wikipedia.org/wiki/Message_Authentication_Code), sowie Menezes u. a.: Handbook of Applied Cryptography. S. 364.

<sup>2</sup>[http://de.wikipedia.org/wiki/Message\\_Integrity\\_Check](http://de.wikipedia.org/wiki/Message_Integrity_Check)

<sup>3</sup>Siehe Kryptographische Hash-Funktion im Glossar

<sup>4</sup><http://de.wikipedia.org/wiki/HMAC>, sowie Stallings: Cryptography and Network Security: Principles and Practice. S. 399–400.

## 2.3. Kryptosysteme

Eine andere Möglichkeit besteht darin mittels eines Blockchiffre<sup>5</sup> den MAC/MIC zu berechnen. Hier werden verschiedene Modi genutzt, mit denen ein Klartext<sup>6</sup> durch einen Blockchiffre verschlüsselt werden kann, wobei das Ergebnis einer Blockverschlüsselung in die Verschlüsselung des nächsten Blocks eingeht. Damit hat auf die Chiffrierung eines Blocks jeder vorhergehende Block Einfluss. Das Ergebnis der letzten Blockverschlüsselung ist damit von allen vorhergehenden Blöcken und damit von allen Daten im Klartext abhängig. Eine Änderung dieser Daten erzwingt ein neues Ergebnis am Ende der Verschlüsselung. Auch hier gibt es wieder einen Klartext und einen Schlüssel (der nur den Kommunikationsteilnehmern bekannt ist), welche als Eingabe der Berechnung dienen.

Der Unterschied zwischen eines MAC/MIC und einer Hashfunktion besteht darin, dass eine Hashfunktion nur den Klartext als Eingabe hat, aber keinen geheimen Schlüssel. Damit kann jeder, der die Hashfunktion kennt, den Hashwert eines Klartextes berechnen und somit ist diese nicht mehr zur Authentifizierung und Integritätsprüfung einer Nachricht geeignet.

Im Gegensatz zu den digitalen Signaturen kann jeder der beiden Kommunizierenden den MAC berechnen. Damit ist von außen nicht nachweisbar, wer von beiden die Nachricht geschickt hat. Als Teilnehmer der Kommunikation weiß man allerdings, dass nur noch der andere als Absender in Frage kommen kann.

## 2.3. Kryptosysteme

### 2.3.1. Ziele und Eigenschaften

Ein Kryptosystem, auch Verschlüsselungsalgorithmus oder Chiffre genannt, ist ein Verfahren, welches einen Klartext mit Hilfe eines Schlüssels in einen Geheimtext überführen kann. Ziel ist es, dass nur derjenige den Klartext aus dem Geheimtext erstellen kann, der den nötigen Schlüssel zur Entschlüsselung besitzt. Eine gute und im kryptographischen Sinne sichere Verschlüsselung muss dabei mehrere Bedingungen erfüllen:

**Konfusion** Diese Eigenschaft besagt, dass mensch zwischen dem Geheimtext und dem Schlüssel, mit dem er verschlüsselt wurde, keinen Zusammenhang herstellen kann.

Da der Schlüssel als Eingabe eines Verschlüsselungsalgorithmus auch Teil der Berechnung ist, somit auch ein mathematischer Zusammenhang zwischen Schlüssel und Algorithmus besteht und der Algorithmus und der Geheimtext für den Angreifer ohne weiteres einsehbar sind, darf es für den Angreifer nicht möglich sein aus dem Geheimtext in irgendeiner Art und Weise den Schlüssel zu berechnen oder herauszufinden. (In Blockchiffren werden dazu sogenannte S-Boxen verwendet, durch die eine Substitution ausgeführt wird.)

---

<sup>5</sup>Siehe 2.3.2

<sup>6</sup>Siehe Klartext im Glossar

## 2. Grundlagen - Kryptographie und WPANs

**Diffusion** Eine gute Diffusion eines Chiffre bewirkt, dass im Chiffrat selbst keinerlei statistische Informationen zu finden sind.

Wird z.B. in einem Text einer bestimmten Sprache jeder Buchstaben durch einen anderen ersetzt (monoalphabetische Substitution), dann wirkt der Text zwar unleserlich, allerdings bleibt die typische statistische Verteilung von bestimmten Buchstaben, wie sie für jede Sprache charakteristisch ist, erhalten.

Ein Geheimtext mit hoher Diffusion dagegen wirkt eher wie ein Rauschen. In ihm sind alle Zeichen und Werte gleich oft verteilt, was dazu führt, dass man Geheimtexte nicht mehr komprimieren kann, da Kompressionsalgorithmen gerade auf statistisch unterschiedlich verteilte Werte setzen. Da aber alle Werte gleich oft vorkommen, haben diese keine Angriffspunkt mehr.

Mensch kann sich solch einen Geheimtext als ein leeres weißes Blatt Papier vorstellen. Ohne Schlüssel ist keinerlei Information zu finden. Erst mit Schlüssel erscheinen darauf plötzlich Buchstaben und Zahlen, als wären sie mit unsichtbarer Tinte geschrieben, die unerwartet und plötzlich zum Vorschein kommt.

**Lawineneffekt** Er besagt, dass jedes Bit im Geheimtext mit jedem Bit im Schlüssel und im Klartext in Verbindung steht. Sobald ein Bit im Klartext geändert wird, muss dies Auswirkungen auf jedes Bit im Geheimtext haben.

### 2.3.2. Chiffrearten

Kryptosysteme lassen sich je nach ihren Eigenschaften und mathematischen Verfahren, auf denen sie aufbauen, in folgende Kategorien einordnen. Dabei ist es auch möglich, dass ein Chiffre gleichzeitig mehreren Kategorien angehören kann.

#### Blockchiffre

Ein Kryptosystem wird Blockchiffre genannt, wenn es einen Block von einer bestimmten Größe mit einem Schritt verschlüsselt. Als Eingabe dient somit ein ganzer Block von Daten.

Hier ergibt sich das Problem, dass Nachrichten, die größer als ein solcher Block sind, nicht ohne weiteres verschlüsselt werden können. Sie müssen in Blöcke gleicher Größe aufgeteilt werden und die Verschlüsselung mit Hilfe eines solchen Blockchiffre muss in einem bestimmten Modus durchgeführt werden. Die Wahl eines solchen Modus hat große Auswirkungen auf die Sicherheit einer Verschlüsselung.

### 2.3. Kryptosysteme

Wenn nach der Aufteilung der Nachricht in Blöcke fester Größe ein Block mit geringerer Größe übrig bleibt, muss dieser mit Daten aufgefüllt werden, bis die feste Größe erreicht ist (dieses Verfahren wird *Padding*<sup>7</sup> genannt).

Blockchiffren gelten als gut untersucht und weisen daher eine beständige Resistenz gegenüber Angriffen auf. Die Wahrscheinlichkeit, dass plötzlich eine Sicherheitslücke im Algorithmus gefunden wird, ist dementsprechend gering.

### Stromchiffre

Dies ist das Pendant zum *Blockchiffre*. Im Gegensatz zu diesem hat der *Stromchiffre* einen Bitstrom, also einzelne Bits als Eingabe, und keine ganzen Datenblock. Hier wird also jedes Bit separat für sich verschlüsselt. Damit erledigen sich die oben genannten Probleme eines Blockchiffre.

Sie lassen sich besonders gut in Hardware implementieren. Bis vor einigen Jahren gab es jedoch noch keine Stromchiffren, die den nötigen Sicherheitsanforderungen genügten. Ihre Theorie ist, verglichen mit den Blockchiffren, noch nicht so weit entwickelt. Allerdings endete 2008 das *eSTREAM-Projekt*<sup>8</sup>, ein Wettbewerb, in dem mehrere Stromchiffren auf ihre Schwächen untersucht worden sind. Das Ziel war es, einen Algorithmus zu finden, der auch in naher Zukunft keine Sicherheitsschwächen aufweisen wird und somit auf lange Zeit in sicherheitskritischen Umgebungen sowohl in Hardware als auch in Software implementiert werden kann.

### Symmetrische Verschlüsselung

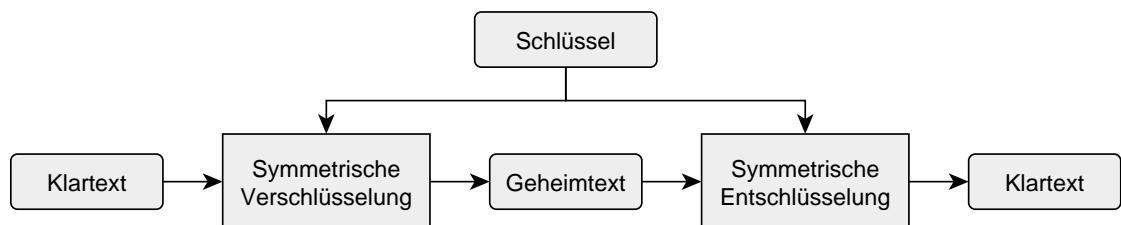


Abbildung 2.2.: Struktur eines symmetrischen Chiffres

Charakteristisch für eine *symmetrische Verschlüsselung* ist, dass der selbe Schlüssel sowohl für die Ver- als auch für die Entschlüsselung genutzt wird, wie in Abbildung 2.2 zu erkennen ist.

<sup>7</sup>Siehe Padding im Glossar

<sup>8</sup><http://www.ecrypt.eu.org/stream/>

## 2. Grundlagen - Kryptographie und WPANs

### Asymmetrische Verschlüsselung

Im Gegensatz zur Symmetrischen Verschlüsselung werden hier zwei verschiedene Schlüssel genutzt. In Abbildung 2.3 ist dieser Zusammenhang schematisch dargestellt. Der Schlüssel A wird zur Verschlüsselung und Schlüssel B zur Entschlüsselung verwendet. Dieses Verfahren wird so angewandt, dass einer der Schlüssel geheim gehalten und der andere veröffentlicht wird.

Dieses Prinzip mutet erst einmal etwas unnatürlich an. Lässt sich die *symmetrische Verschlüsselung* noch mit Hilfe des Bildes eines normalen Schlosses und eines dazugehörigen Schlüssels erklären, so scheitert dieses Bild bei den *asymmetrischen Kryptosystemen*. Dieses System bietet allerdings einige wichtige Vorteile:

Wenn eine Nachricht mit dem geheimen Schlüssel verschlüsselt worden ist, so kann der Geheimtext nur noch mit dem öffentlichen Schlüssel entschlüsselt werden. Auch wenn jeder an diesen Schlüssel herankommen und somit jeder den Klartext lesen kann, so liegt das Augenmerk darauf, dass die Nachricht vom Besitzer des geheimen Schlüssels stammt, da nur jemand mit diesem Schlüssel den Geheimtext erstellt haben kann.

Anders herum kann nur derjenige den (mit dem öffentlichen Schlüssel verschlüsselten) Geheimtext entschlüsseln, der den geheimen Schlüssel besitzt.

Dieses System eignet sich also sowohl zur Verschlüsselung als auch zur Authentifizierung. Leider haben asymmetrische Systeme den Nachteil, dass sie deutlich langsamer sind als ihre symmetrischen Pendants. Aus diesem Grund gibt es die sogenannten *hybriden Verfahren*.

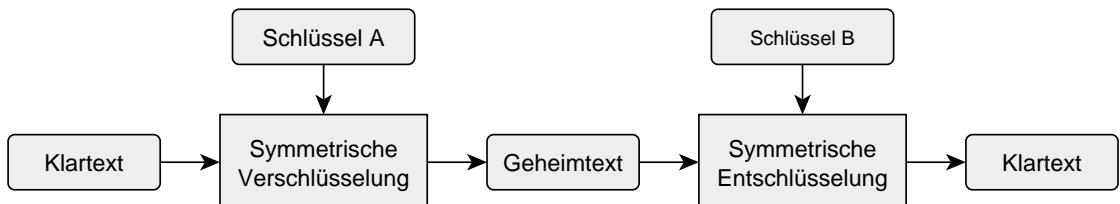


Abbildung 2.3.: Struktur eines asymmetrischen Chiffres

### Hybride Verschlüsselung

In einem *hybriden* Verfahren werden *symmetrische* und *asymmetrische* Systeme miteinander kombiniert. Mit dem *asymmetrischen* Systemen wird ein temporär erzeugter Schlüssel zur *symmetrischen Verschlüsselung* zwischen beiden Parteien ausgetauscht. Der erste Teilnehmer verschlüsselt diesen nun mit dem öffentlichen Schlüssel des anderen Teilnehmers und schickt ihm diesen chiffrierten Schlüssel zu. Nur der andere Teilnehmer kann diesen Schlüssel wieder entschlüsseln, da nur er im Besitz des passenden geheimen Schlüssel ist und sonst niemand. Nun haben beide Parteien einen Schlüssel, um die

## 2.4. Verschlüsselungsmodi

Kommunikation *symmetrisch* zu verschlüsseln, ohne dass dieser ungesichert ausgetauscht werden musste.

### 2.3.3. AES - Ein symmetrischer Blockchiffre

So wie das *eSTREAM*-Projekt sich zum Ziel gesetzt hatte einen sicheren Stromchiffre zu finden, so hat das US-amerikanische *National Institute of Standards and Technology* einen Wettbewerb ausgeschrieben, in welchem der beste symmetrische Chiffre ausgewählt werden sollte, der daraufhin die Bezeichnung *Advanced Encryption Standard (AES)* tragen sollte. Das Rennen hat ein Algorithmus namens *Rijndael* gemacht. Bis auf die variablen Blocklängen von 128, 192 und 256 Bit im *Rijndael*-Algorithmus wurden alle Eigenschaften des Chiffre in den AES übernommen. Dabei wurde die Blocklänge auf feste 128 Bit festgelegt.

AES gilt als besonders gut untersucht und seit seiner Ernennung vor 10 Jahren wurde kein praktisch relevanter Angriff gegen ihn gefunden. Er kann sowohl in Software als auch in Hardware effektiv implementiert werden. Das Besondere an ihm ist, dass seine S-Boxen berechenbar sind und so deren einzelne Elemente zur Laufzeit ermittelt werden können, ohne dass die gesamte Box im Speicher gehalten werden muss. Besonders bei Embedded Devices kommt dieser Vorteil zum Tragen. In dieser Studienarbeit kommt der Chip ATMega128RFA1 auf einem Radio Controller Board (RCB) zur Anwendung. Dieser besitzt durch seinen Strom sparenden und kostengünstigen Charakter wenig Rechengeschwindigkeit und Speicher.

AES spielt im Zusammenhang mit *Counter with CBC-MAC (CCM)* und *Enhanced Counter with CBC-MAC (CCM\*)*<sup>9</sup> in den Standards *IEEE 802.15.4* und *ZigBee* eine große Rolle und ist maßgeblich für die Sicherheits-Infrastruktur beider Standards verantwortlich.

## 2.4. Verschlüsselungsmodi

### 2.4.1. Sinn und Zweck von Verschlüsselungsmodi

Bei Blockchiffren kann nur ein Block fester Größe (oftmals 128 Bit) mit einem Mal verschlüsselt werden. Wie bereits im Kapitel 2.3.2 erwähnt, werden die Daten, sobald sie größer sind als die Blockgröße, in Teile dieser Größe gesplittet und der letzte Teil, wenn er kleiner als die Blockgröße ist, mit Zahlen aufgefüllt (Padding).

---

<sup>9</sup>Siehe Kapitel 2.4.6

## 2. Grundlagen - Kryptographie und WPANs

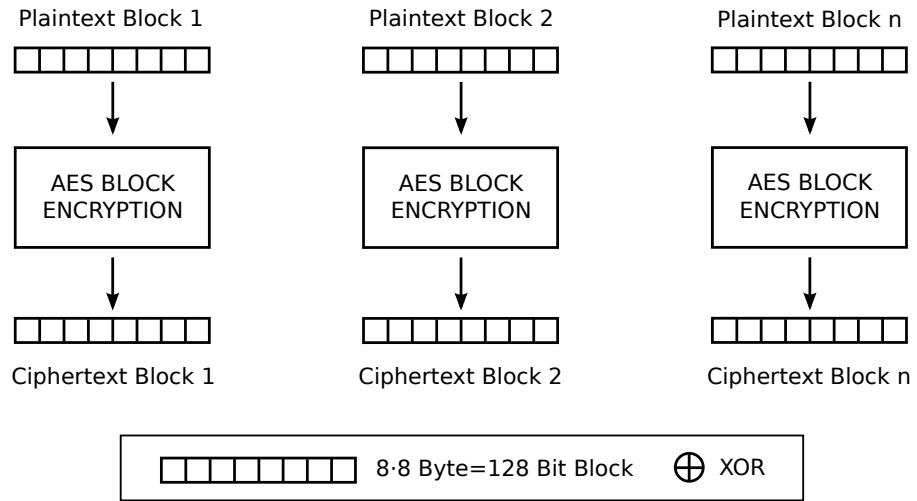


Abbildung 2.4.: Ablauf der Verschlüsselung mit Hilfe eines Blockchiffre im ECB-Mode

Wie diese Blöcke nun mit dem Blockchiffre verschlüsselt werden, legt ein sogenannter Verschlüsselungsmodus fest. Seine Arbeitsweise hat grundlegende Auswirkungen auf die Sicherheit der Verschlüsselung.

### 2.4.2. ECB - Electronic Code Book Mode

Im *Electronic Code Book (ECB)*-Modus<sup>10</sup> wird jeder der Blöcke einzeln und unabhängig mit dem selben Schlüssel verschlüsselt. Dieser Modus ist veraltet und unsicher und von seiner Verwendung wird abgeraten.

Der Ablauf dieses Modus wird in Abbildung 2.4 für die Verschlüsselung in Abbildung 2.5 für die Entschlüsselung dargestellt.

Da identische Blöcke durch Verwendung des selben Schlüssels immer zu selben Geheimtextblöcken führen, ist es möglich statistische Muster im Chiffraum zu erkennen und die Verschlüsselung so zu brechen.

### 2.4.3. CBC - Cipher Block Chaining Mode

Im *Cipher Block Chaining (CBC)*-Modus<sup>11</sup> werden die Blöcke nicht wie im ECB-Modus unabhängig voneinander verschlüsselt, sondern beeinflussen sich gegenseitig. Die Chiffrierung eines Blocks geht in die des nachfolgenden Blocks mit ein, indem das Ergebnis der

<sup>10</sup>Siehe ECB im Glossar und [http://de.wikipedia.org/wiki/Electronic\\_Code\\_Book\\_Mode](http://de.wikipedia.org/wiki/Electronic_Code_Book_Mode)

<sup>11</sup>Siehe CBC im Glossar

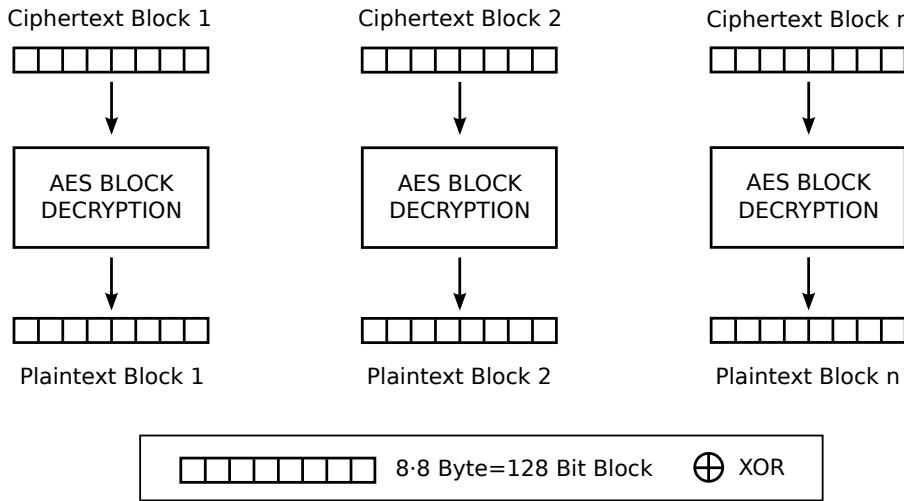


Abbildung 2.5.: Ablauf der Entschlüsselung mit Hilfe eines Blockchiffre im ECB-Mode

Chiffrierung mit dem folgendem Klartextblock XOR-verknüpft und danach mittels Blockchiffre wieder verschlüsselt wird. Dieses Ergebnis wird wiederum mit dem nachfolgenden Klartextblock XOR-verknüpft und wieder verschlüsselt. Dabei wird in jedem Schritt der Chiffreblock gespeichert und alle zusammen ergeben die verschlüsselten Daten.

Interessant ist dabei der Anfang des Prozesses. Da der erste Block keinen Vorgänger hat, muss an dessen Stelle ein anderer Datenblock herhalten. Dieser zur Initialisierung genutzte Datenblock wird *Initialisierungsvektor (IV)* genannt. Er muss folglich genauso groß sein wie ein zu verschlüsselnder Block.

Der Ablauf dieses Modus wird für die Verschlüsselung in Abbildung 2.6 und für die Entschlüsselung in Abbildung 2.7 dargestellt.

Dabei muss der *IV* bestimmten Bedingungen genügen um die Sicherheit der Verschlüsselung nicht zu gefährden:

- Der *IV* darf nur einmal verwendet werden. Sobald er öfters zur Verschlüsselung genutzt wird, ist es für einen Angreifer möglich statistische Informationen aus dem Chiffrat zu gewinnen, da mit dem selben Schlüssel und dem selben *IV* unterschiedliche Klartexte verschlüsselt werden.
- Er muss nicht geheimgehalten werden und darf im Klartext übertragen werden. Eine Geheimhaltung würde nicht zur Sicherheit beitragen. Da der Empfänger sowieso den *IV* zur Entschlüsselung braucht, muss dieser ihm bekannt sein.
- Oftmals wird eine zufällig erzeugte Zahl als *IV* verwendet. Dies soll sicherstellen, dass keine gleichen *IVs* mehrmals verwendet werden. Allerdings kann der *IV* auch deterministisch erstellt werden, je nachdem in welchen Kontext er verwendet wird. Wichtig ist nur, dass jeder *IV* einmal verwendet wird.

## 2. Grundlagen - Kryptographie und WPANs

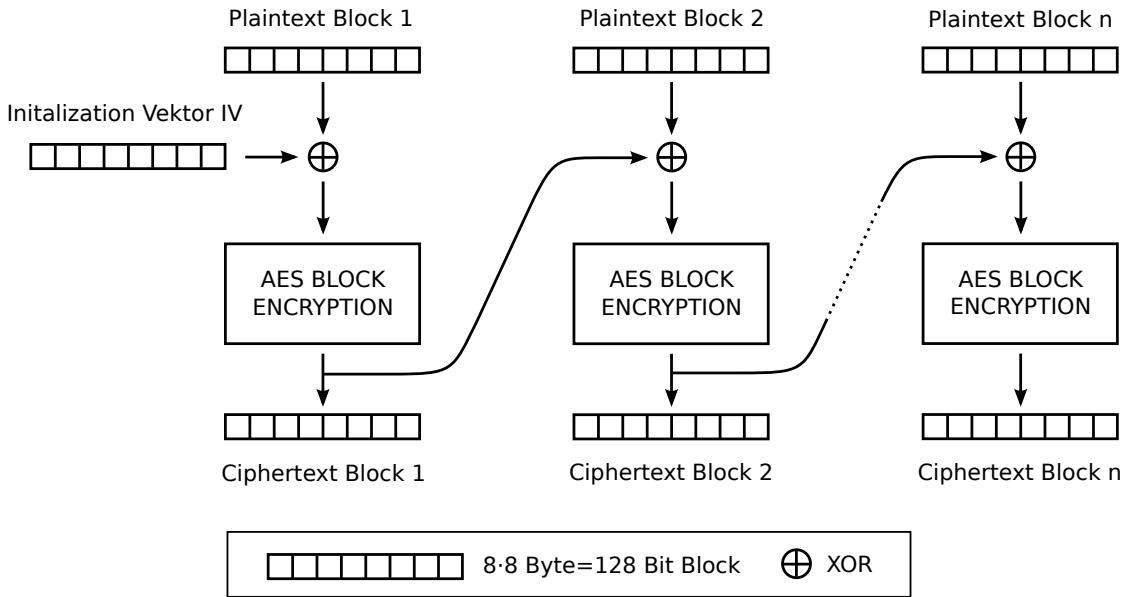


Abbildung 2.6.: Ablauf der Verschlüsselung mit Hilfe eines Blockchiffre im CBC-Mode

### 2.4.4. CBC-MAC

Der Modus *CBC-Message Authentication Code (CBC-MAC)* dient dazu, einen *MAC* eines Datenstroms zu berechnen. Genutzt wird dabei der *CBC-Mode*. Mit ihm ist jede Verschlüsselung eines Blocks von der Verschlüsselung der vorhergehenden Blöcke abhängig. Somit ist der letzte verschlüsselte Block abhängig von allen vorhergehenden Blöcken und ändert sich, sobald auch nur ein Bit des zu verschlüsselnden Datenstroms sich ändert, was durch den anfangs erwähnten *Lawineneffekt* des Blockchiffre verursacht wird. Dieser letzte Chiffreblock wird als *MAC* genutzt.

### 2.4.5. CTR - Counter Mode

Im Gegensatz zum *CBC-Mode* wird im *CTR-Mode*<sup>12</sup> jeder Block einzeln und unabhängig von den anderen Blöcken verschlüsselt. Obwohl gerade dieser Umstand im *ECB-Mode* ein Sicherheitsproblem darstellt, führt dies im *CTR-Mode* zu keinerlei Problemen. Dies liegt daran, dass hier für jeden Block ein einzigartiger *IV* verwendet wird. Dieser wird mit einem Blockchiffre verschlüsselt und anschließend mit dem dazugehörigen Block XOR-verknüpft. Jeder *IV* besteht aus einer sogenannten *Nonce*<sup>13</sup>, die für jeden zu verschlüsselnden Datenstrom einzigartig ist, jedoch bei jedem Block in diesem Datenstrom konstant bleibt, und einem Zähler, der für jeden Block um eins erhöht wird. Diese *IVs* werden

<sup>12</sup>Siehe Counter (Zähler) (CTR) im Glossar

<sup>13</sup>Siehe Nonce im Glossar

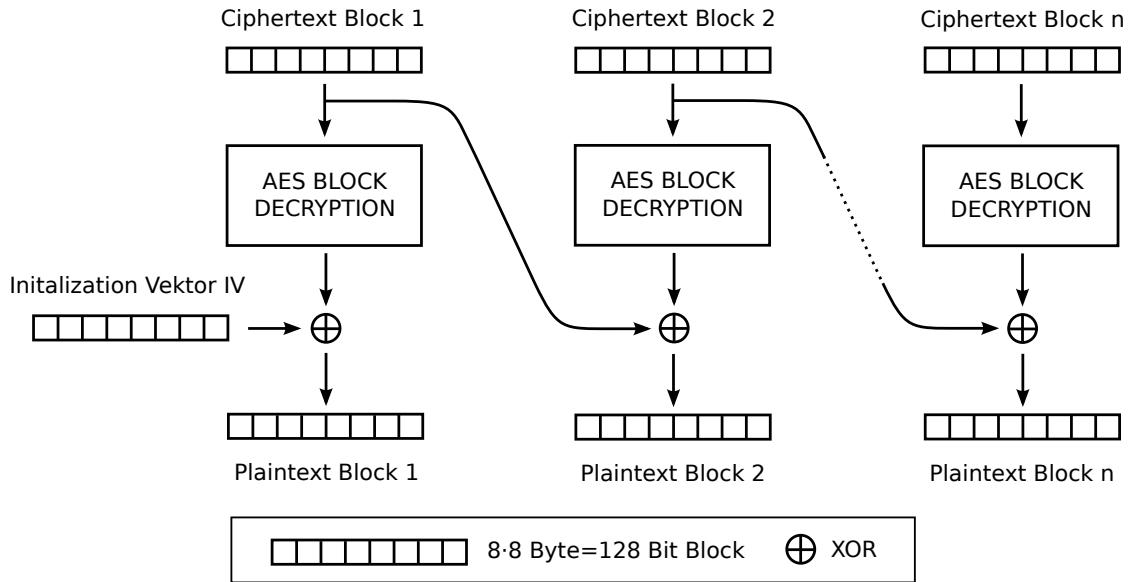


Abbildung 2.7.: Ablauf der Entschlüsselung mit Hilfe eines Blockchiffre im CBC-Mode

nun einzeln verschlüsselt und anschließend mit dem dazugehörigen Datenblock XOR-verknüpft. Die einzelnen verschlüsselten IVs ergeben hintereinander angeordnet einen sogenannten Key Stream<sup>14</sup>. Dies bringt einige wesentliche Vorteile mit sich:

- durch den Aufbau jedes IVs können sie alle unabhängig voneinander im Voraus berechnet werden. Da die Verschlüsselung eines Klartextblocks unabhängig von den anderen Blöcken ist, kann die Verschlüsselung parallelisiert werden
- um einen Block zu entschlüsseln, muss nicht mehr der gesamte Datenstrom entschlüsselt werden, wie dies im *CBC-Mode* der Fall ist, sondern es kann dieser Block direkt entschlüsselt werden, da der dazugehörige IV direkt berechnet werden kann
- durch die Unabhängigkeit der Chiffreblöcke untereinander können sich Fehler in der Übertragung nicht fortpflanzen (wie z.B. im *CBC-Mode*)
- die Ver- und Entschlüsselung sind komplett identisch, da der Key Stream in beiden Fällen der gleiche ist und die XOR-Verknüpfung durch nochmalige Anwendung wieder den Ausgangswert ergibt, also die vorhergehende XOR-Verknüpfung rückgängig macht

## 2. Grundlagen - Kryptographie und WPANs

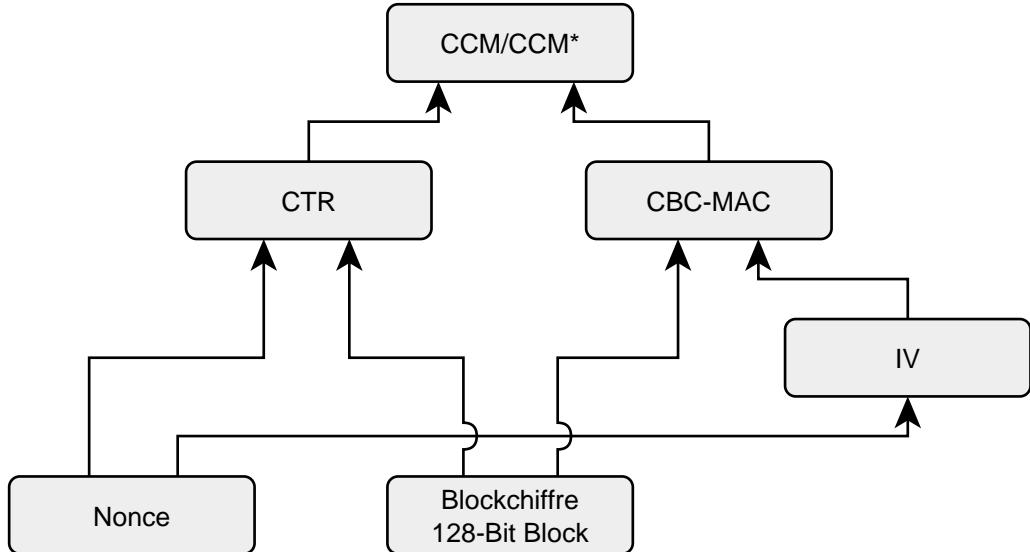


Abbildung 2.8.: der Aufbau des CCM-Mode durch anderer Verschlüsselungsmodi und Komponenten

### 2.4.6. CCM/CCM\* - Counter with CBC-MAC

Dieser Modus spielt in Verbindung mit *IEEE 802.15.4* und *ZigBee* eine große Rolle. In diesen Standards dient er nämlich zur Verschlüsselung und Sicherung der Daten, die über Funk übertragen werden sollen. Er wird definiert im RFC 3610<sup>15</sup>.

*CCM/CCM\** setzt sich dabei aus den oben beschriebenen Modi zusammen und kombiniert diese in geschickter Art und Weise um Daten zu verschlüsseln und eine Authentizitäts- und Integritätsprüfung selbiger zu ermöglichen. Seinen Aufbau kann man sich als einen hierarchischen Baum Vorstellen: Als Grundlage oder unterste Ebene dient ein beliebiger 128 Bit Block Blockchiffre und eine *Nonce*, wie dies in Abbildung 2.8 zu sehen ist.

Die *Nonce* ist wiederum Teil eines IV, welcher vom *CBC-MAC-Mode* zur Authentifizierung mittels MAC-Berechnung genutzt wird. Beide Modi nutzen den selben Blockchiffre und den selben Schlüssel namens K. Im *CTR-Mode* geht die *Nonce* ebenfalls zusammen mit einem Zähler ein, der für jeden Block um eins erhöht wird. Beide sind Teil eines Strings, der durch Verschlüsselung mit dem Blockchiffre den Keystream ergibt. Beide Modi bilden den Kern des *CCM/CCM\*-Mode*.

*CCM/CCM\** besitzt insgesamt zwei Parameter und vier verschiedene Inputs:

<sup>14</sup>Siehe Key Stream im Glossar

<sup>15</sup><http://tools.ietf.org/html/rfc3610>

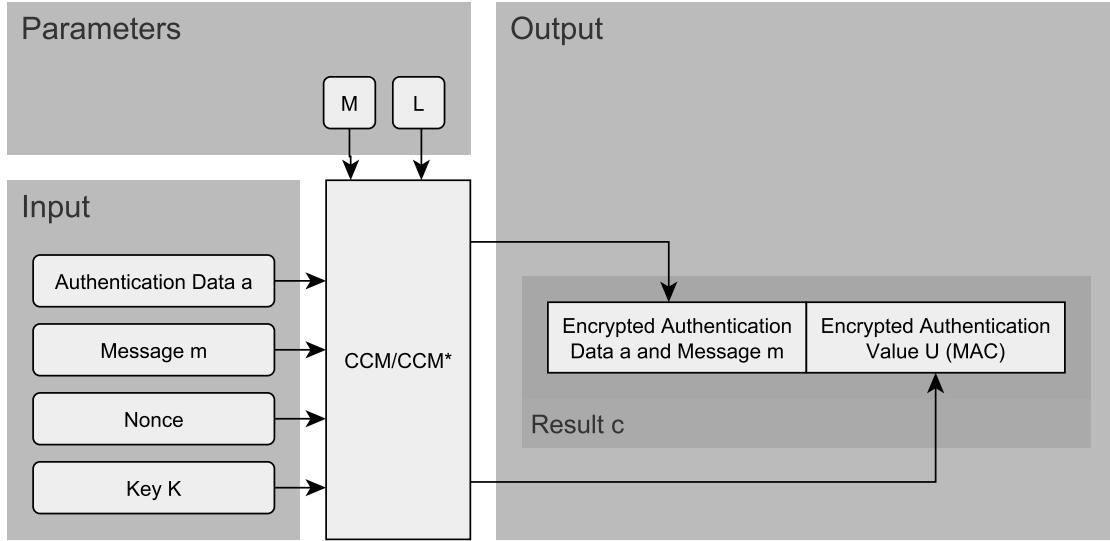


Abbildung 2.9.: Die Eingaben, Parameter und Ausgaben des CCM-Mode

**M** beschreibt die Größe des *authentication field*, sprich den MAC, und kann die Werte 4, 6, 8, 10, 12, 14, und 16 Bytes annehmen. Je größer der Wert ist, desto sicherer ist die Authentizitätsprüfung, allerdings wird damit die mögliche Größe der zu verschlüsselnden Nachricht kleiner.

**L** Dieser Parameter regelt die Größe des *Lengthfield* und damit auch die maximale Größe der *Message m*. Der Wert von L und die Größe der Nonce stehen dabei im umgekehrten Verhältnis: Wird ein großer Wert L gewählt um eine lange Message m zu verschlüsseln, wird die Größe der Nonce kleiner. Wird L klein gewählt auf Grund einer kleinen Message m, wird die Nonce N wiederum größer. Der Wert von L kann von 2 bis 8 variieren. In den Spezifikationen des IEEE 802.15.4 Standard hat L einen festen Wert von 2.

**Authentication Data a** ist der Teil der Daten, der nicht verschlüsselt sondern nur in die Berechnung des MAC eingeht, sprich nur authentifiziert werden soll. Dabei kann es sich z.B. um Routinginformationen handeln oder andere Informationen, die den verschlüsselten Daten einen Kontext geben.

**Nachricht m** ist der Teil der Daten, der sowohl authentifiziert als auch verschlüsselt werden soll.

**Nonce** Sie wird von *CCM/CCM\** nicht vorgegeben, sondern kann je nach Implementierung frei gewählt und damit den besonderen Umständen angepasst werden.

**Schlüssel K** wird vom Blockchiffre sowohl zur Verschlüsselung der Daten als auch zur Berechnung der MAC genutzt. Seine Größe ist vom Standard selbst nicht festgelegt und hängt vom gewählten Blockchiffre sowie von der Implementierung ab.

## 2. Grundlagen - Kryptographie und WPANs

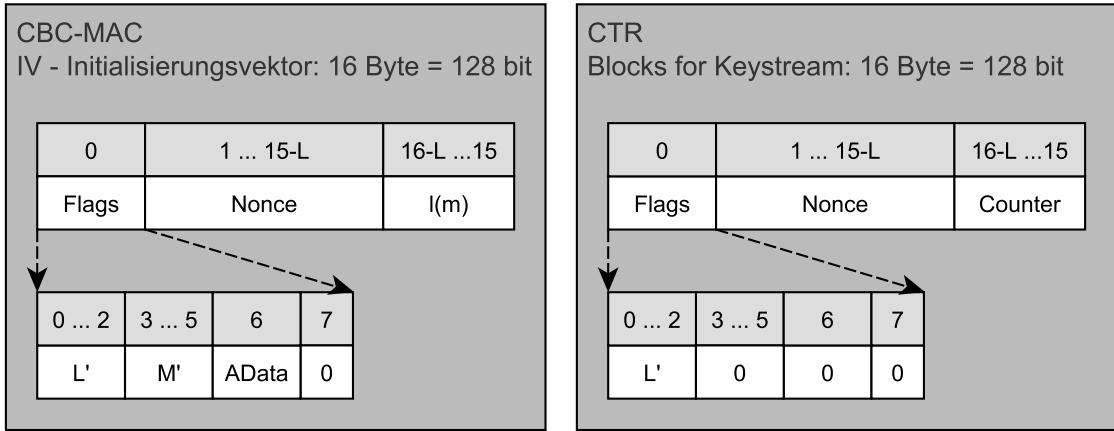


Abbildung 2.10.: Links: Aufbau des IV im CBC-MAC-Mode, Rechts: Aufbau der Blöcke für den Keystream im CTR-Mode

Das Zusammenspiel der Parameter und der Eingaben ist in Abbildung 2.9 dargestellt.

*CCM/CCM\** gibt nach der Sicherung die verschlüsselte Authentication Data a und die verschlüsselte Nachricht m aus gefolgt von dem verschlüsselten Authentication Value U.

In Abbildung 2.10 wird der Aufbau des IV für den *CBC-MAC-Mode* und für die Blöcke zur Keystream-Generierung für den *CTR-Mode* schematisch dargestellt.

Der IV besteht einmal aus einem 1 Byte großen Bitfeld - jedes einzelne Bit wird auch Flag genannt - aus der Nonce selbst und am Ende aus dem *Lengthfield*, welches die Länge der Message beinhaltet. Dabei ist die Grenze zwischen der Nonce und dem *Lengthfield* am Ende variabel und wird vom Parameter L gesteuert. Im Fall des IEEE 802.15.4 Standards hat L den festen Wert 2. Damit hat der IV stehts den in Abbildung 2.10 gezeigten Aufbau. Das Flagfield besteht aus den Parametern M' und L', die aus den Formeln

$$M' = \frac{M - 2}{2} \quad (2.1)$$

bzw.

$$L' = L - 1 \quad (2.2)$$

berechnet werden. Das Bit AData ist nur gesetzt, falls die Größe der Daten, die nur mit einer MIC geschützt werden sollen (authentication data a), größer null ist. Ansonsten, falls also keine Daten ausschließlich mit einer MIC geschützt werden sollen, ist dieses Flag gelöscht.

Im *CTR-Mode* dient die Nonce zur Generierung der Blöcke, mit denen der Keystream erstellt wird. Hier besteht das Flagfield nur aus den drei Bits für den Wert L', ansonsten

wird alles auf 0 gesetzt. Die Nonce ist genau dieselbe wie im *CBC-MAC-Mode* und hat auch deren Länge. Das Lengthfield allerdings übernimmt hier die Funktion eines Zählers und erhöht sich für jeden Block von 128 Bit Größe um eins. Das bedeutet, dieser Block wird mit dem Zähler 0 verschlüsselt und mit dem ersten Klartextblock XOR-verknüpft. Der nächste Block hat den Zähler 1, wird verschlüsselt und wiederum mit dem nächsten Klartextblock XOR-verknüpft. Dies geht so lange weiter bis der Block  $n$  erreicht ist. Die Verschlüsselung der Blöcke mit dem Zähler kann auch in einem Rutsch geschehen und der Keystream damit im Voraus berechnet werden.

In einem Bericht über die Sicherheit dieses Modus hat JAKOB JONSSON festgestellt, dass CCM gleich auf liegt mit anderen etablierten Sicherheitsstandards und gegen Angriffe weitgehendst gefeilt ist. Beachtung sollte allerdings dem Fall geschenkt werden, dass eine Schlüssellänge von 128 Bit gewählt wird. Dann nämlich wird eine *Precomputation Attack* deutlich erleichtert. In dieser Art Angriff werden durch die bekannte Nonce und zufällig generierte Schlüssel die Blöcke vorausberechnet und in einer Tabelle abgelegt (wie bei einer *Rainbow Table* um Kryptografische Hashfunktionen zu brechen). Durch Vergleich der ermittelten Werte kann der Key ermittelt werden. Abhilfe schafft die Wahl eines längeren Schlüssels, was die Berechnung mittels Tabellen deutlich erschwert, oder die Verwendung von unterschiedlichen Zufallswerten in der Nonce. Möglich ist auch im Falle eines Funknetzwerks die *Extended Address* eines Netzwerknotens in der Nonce zu speichern, wie es beispielsweise das Protokoll *ZigBee* umsetzt. Trotzdem bleibt die Frage offen, wie sich dieser Umstand in der Zukunft entwickelt und ob er dann durch leistungsfähigere Hardware ausgenutzt werden kann.

Security

## 2.5. WPAN-Standards

### 2.5.1. IEEE 802.15.4

IEEE 802.15.4 übernimmt die Aufgabe der zwei untersten Schichten des OSI-Modells<sup>16</sup>: Die Bit-Übertragungsschicht (Physical Layer) und die Sicherungsschicht (Data Link Layer). Ziel ist es, dass weitere Standards auf diesen beiden Schichten aufbauen und somit die restlichen Schichten implementieren können (wie z.B. ZigBee).

Innerhalb der Spezifikationen gibt es zwei verschiedene Arten von Netzwerkgeräten:

**Full Function Device (Gerät mit vollem Funktionsumfang) (FFD)** Sie zeichnen sich dadurch aus, dass das gesamte Spektrum an Funktionen implementiert ist. Meistens handelt es sich um Geräte mit fester Stromversorgung.

---

<sup>16</sup>Das OSI-Modell beschreibt ein Ebenenmodell, in dem jede Ebene eine spezielle Aufgabe übernimmt und so ein Gesamtsystem entsteht, welches die Übertragung von Informationen unabhängig vom Medium garantiert. Siehe <http://de.wikipedia.org/wiki/OSI-Modell>

## 2. Grundlagen - Kryptographie und WPANs

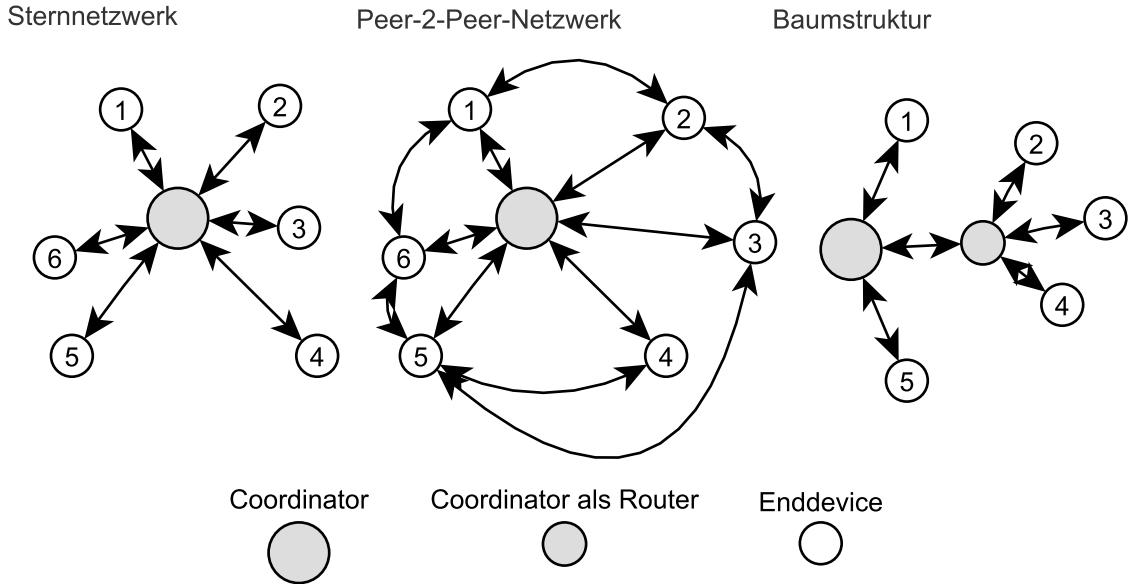


Abbildung 2.11.: Die verschiedenen Topologien, in denen ein Netzwerk aufgebaut werden kann.

**Reduced Function Device (Gerät mit reduziertem Funktionsumfang) (RFD)** Da die Hardware möglichst kostengünstig sein muss und dadurch Speicher und andere Ressourcen knapp sind, werden in dieser Gerätekasse nur die wichtigsten Funktionen realisiert.

Aus diesen Geräteklassen setzen sich nun die Netzwerke zusammen. Es werden standardmäßig drei verschiedene Netzwerk-Topologien unterstützt, die in Abbildung 2.11 zu sehen sind:

**Stern-Topologie** Zentrum dieses Netzwerkaufbaus ist der Koordinator, ein FFD. Alle anderen Knoten kommunizieren mit dem Koordinator direkt. Damit ein Knoten mit einem weiteren Knoten Verbindung aufnehmen kann, muss der Koordinator zwischen beiden Knoten vermitteln.

**Peer-to-Peer** Wie in der Sterntopologie existiert ein Koordinator. In dieser Konstellation aber können einzelne Knoten direkt miteinander in Verbindung treten, ohne dass der Koordinator vermitteln muss.

**Baumstruktur** Wie bei einem Baum, von der Wurzel bis zur Spitze und den Blättern, gibt es mehrere Abzweigungen, auch Knoten genannt. Die Wurzel ist ein Coordinator und dieser ist mit weiteren Koordinatoren (durch FFDs realisiert), die die Knoten oder Abzweigungen darstellen, verbunden bis hin zum Personal Area Network (Persönliches Umgebungsnetzwerk) (PAN)-Coordinator, welcher ein Sternnetzwerk mit

mehreren RFDs, den Blättern, aufbaut. Diese Blätter werden oftmals Endgeräte genannt.

IEEE 802.15.4 kann selber keine komplexeren Netzwerke bilden und verwalten, wie es z.B. bei einer Baumstruktur der Fall ist. Dazu sind beispielsweise Funktionen zum Routen von Nöten, wie sie dieses Protokoll nicht anbietet.

Ein komplexeres Netzwerk kann aus mehreren kleinen Netzwerken (genannt Subnetwork) zusammengesetzt werden. Um nicht jeden Knoten mit einer 64 Bit langen *Extended Address* ansprechen zu müssen, ist es im IEEE 802.15.4 Standard möglich, einem solchen *Subnetwork* eine sogenannte PAN-ID zuzuweisen und die Knoten in diesem *Subnetwork* mit einer 16 Bit langen *Short Address* zu identifizieren. Damit werden pro *Frame* 4 Byte weniger Daten übertragen.

Der Funkverkehr selber findet im Frequenzbereich von 2400 - 2483,5 Mhz statt. Die Bandbreite von 2 MHz, unterteilt in 16 Kanäle, stellt eine Datenübertragungsgeschwindigkeit von 250 KBit/s zur Verfügung. Da bei diesen Frequenzen auch Bluetooth und WLAN funkeln, kann es oftmals zu Störungen kommen. Dazu wurde eine Ausweichmöglichkeit geschaffen, indem auch auf 868 - 868,6 MHz gefunkt werden kann, was allerdings die Geschwindigkeit deutlich begrenzt.

Um die Kompatibilität zwischen den verschiedenen Implementierungen zu gewährleisten, spezifiziert der Standard sogenannte PAN Information Base (PAN Informationsdatenbank) (PIB)-Attribute, mit denen das Verhalten des Netzwerkes und der einzelnen Devices gesteuert werden kann. Mit ihnen werden beispielsweise die Sicherheitsfunktionen des Netzwerkes und der Nodes konfiguriert. Sie werden im Knoten selber in einer Art Datenbank gespeichert und können mit speziellen Funktionen, den *Service Access Point* (*Zugriffspunkt für Dienste*) (*SAP*)s, geschrieben und wieder abgefragt werden. Diese SAPs dienen ebenfalls als Schnittstelle sowohl für eine Applikation, die auf dem MAC Stack aufbaut, als auch für ein die höheren OSI-Schichten implementierendes Protokoll, welches ebenfalls auf dem MAC Stack aufbaut.

### 2.5.2. ZigBee

ZigBee wiederum ist ein Protokoll, welches auf IEEE 802.15.4 aufbaut und dessen Services nutzt. Es wurde für Sensornetzwerke entworfen und verwaltet eine groß Anzahl an Sensoren, die möglichst wartungsarm und mit Hilfe von Batterien energieautark über einen langen Zeitraum (bis zu 15 Jahren) arbeiten. Da diese Devices oftmals schwer zugänglich sind, ist es von besonderem Interesse, dass der Datenaustausch möglichst energiearm von statten läuft um die Lebensdauer zu erhöhen. Dies ist eines der wichtigsten Designziele von ZigBee.

Im Gegensatz zu IEEE 802.15.4 bietet ZigBee die Möglichkeit den Datenverkehr zu routen und damit komplexe Netzwerke aufzubauen.

## *2. Grundlagen - Kryptographie und WPANs*

Zu den Standardklassen FFD und RFD, und damit zu den Rollen Koordinator und Endgerät, kommt eine weitere Gerätekasse hinzu: Der Router. Er stellt ein FFD dar mit dem eine Baumstruktur bestehend aus Subnetzwerken, die durch Router gesteuert werden, gebildet werden kann. Sie übernehmen die Weiterleitung von Daten in andere Subnetzwerke.

# 3. KONZEPT UND SZENARIEN

## 3.1. Technische Randbedingungen und Angriffsszenarien

Die Datenübertragung findet mittels Funkübertragung durch den freien Raum statt. Der freie Raum stellt ein *Shared Medium* dar, welches somit auch von anderen Teilnehmern, die sich im näheren Umfeld bewegen, genutzt wird. Dies ist ein großer Unterschied zur kabelgebundenen Datenübertragung, bei der sich ein Angreifer noch direkt physischen Zugang zum Kabel verschaffen musste, was meist durch eine Änderung des Innenwiderstandes der Leitung entdeckt werden konnte. Auch physische Barrieren konnten nicht autorisierte Teilnehmer vom Kabel und dadurch von der Netzwerkkommunikation fernhalten. Autorisiert waren automatisch diejenigen, welche einen Kabelanschluss und damit physischen Zugang erhalten haben.

In Funknetzwerken ist die Situation anders: Hier kann jeder den Raum als Medium nutzen und damit auch die Kommunikation anderer empfangen und an diese Daten senden. Das Netzwerk braucht hier andere Regeln und Mechanismen um zu entscheiden, welcher Kommunikationspartner vertrauensvoll ist und welcher nicht.

Dieses Problem ist nicht zu unterschätzen, da auch von einem auf *IEEE 802.15.4* oder *ZigBee* aufbauendem Funknetzwerk gesendete Informationen mit entsprechender Hardware (z.B. mit Hilfe bessere Antennen und einem dementsprechend größeren Antennengewinn) gesendete Daten über einen großen Radius hinaus aufgefangen werden können, obwohl die autorisierten Nodes nur für kleinere Räden ausgelegt sind und darüber hinaus keine Daten mehr empfangen können. (*ZigBee* bietet hier für einige Fälle, in denen Passwörter im Klartext gesendet werden, die Möglichkeit die Signalstärke zu senken um

### 3. Konzept und Szenarien

nur den gewünschten Node zu erreichen. Mit Richtfunkantennen kann dieser Datenfunk aber möglicherweise noch in größerer Entfernung abgehört werden.)

Hinzu kommt, dass es durchaus möglich ist, auf die Endgeräte physikalisch zuzugreifen. Da einige Netzwerke aus mehreren hundert oder gar tausend Knoten bestehen, kann nicht jeder einzelne vor Zugriffen Fremder effektiv geschützt werden.

Die Arten eines Angriffs lassen sich nun grob in zwei Kategorien einordnen:

- Passive Angriffe
- Aktive Angriffe

Folgende Szenarien, die durch einen Angreifer von außen realisiert werden können, sollen verhindert werden:

- das Mitlesen der Netzwerkkommunikation (Passiver Angriff)
- die Manipulation des Netzwerkes (Aktiver Angriff)
  - das Vortäuschen einer falschen Identität, um so Zugang zum Netzwerk zu erlangen
  - das nachträgliche Ändern einer von einer bereits autorisierten Knoten abgesendeten Nachricht
  - das Wiedereinspielen einer von einer bereits autorisierten Knoten abgesendeten Nachricht -> Replay-Attack
  - der physikalische Zugriff auf installierte Netzwerkknoten, um geheime Informationen durch direkten Zugriff auf die Hardware zu erlangen

Mit Hilfe von *Passiven Angriffen* ist es möglich an Informationen zu kommen, ohne selbst an der Kommunikation teilzunehmen. Diese Informationen können einmal die zu übertragenen Nutzdaten<sup>1</sup> sein, wie beispielsweise Messdaten oder Informationen über den Zustand des Netzwerkes und dessen Knoten (ob z.B. eine Lampe in einem Raum an oder aus ist). Aber auch andere Informationen können aus weniger wichtig erscheinenden Übertragungen gewonnen werden. So ist es möglich aus Beacon<sup>2</sup>- oder Broadcast<sup>3</sup>-Pakete an Informationen über die Netzwerkstruktur und deren Funktionsweise oder allgemein an stochastisch aussagekräftige Werte zu gelangen (z.B. das Hochzählen eines Zählers in einem Paket, der sonst verschlüsselt und damit nicht sichtbar wäre, um so auf ein voraussagbares Verhalten zu schließen). Der Angreifer muss dazu nicht einmal Einfluss auf das Netzwerk nehmen. Da hier über ein *Shared Medium* gesendet wird, können alle Geräte, solange sie genügend nahe am Sender sind, den Funkverkehr mithören.

---

<sup>1</sup>engl. *Payload* siehe Glossar

<sup>2</sup>Siehe Beacon-Packet im Glossar

<sup>3</sup>Siehe Broadcast-Paket im Glossar

### 3.1. Technische Randbedingungen und Angriffsszenarien

*Aktive Angriffe* hingegen setzen einen Eingriff von außen durch Dritte voraus. Hier wird direkt Einfluss auf das Netzwerk und dessen Verhalten genommen. Ziel ist es dieses so zu manipulieren, dass das Netzwerk seine Funktion nicht mehr erfüllen kann, eine andere Funktion, die nicht geplant ist, erfüllt oder der Angreifer an wichtige und aussagekräftige Daten gelangen kann. Beispielsweise wurde *WEP* gebrochen, indem durch das Wiedereinspielen von bereits empfangenen Paketen das Netzwerk zu einem massiv erhöhten Datenversand bewogen wurde. Diese Art Angriff nennt man *Replay-Attack*. Da hier der Schwachpunkt in dem zu kurz gewählten IV lag, wiederholte dieser sich nach einer bestimmten Anzahl von bereits versendeten Paketen (durchschnittliche alle 5000 Pakete). Das bedeutet, dass unterschiedliche Daten entstanden, die mit demselben Schlüssel und demselben IV chiffriert wurden. Dieses Wissen schränkte den Schlüsselraum so weit ein, dass dieser durch eine *Bruteforce Attack* gebrochen werden konnte.<sup>4</sup>

Um nicht autorisierte Zuhörer (Passiver Angriff) vom Mithören der Kommunikation abzuhalten, wird auf ein Mathematisches Verfahren namens *Verschlüsselung* zurückgegriffen. Auf diesem Verfahren baut der größte Teil der Sicherheitsarchitektur auf. Mit der Verschlüsselung ist es nicht nur möglich Informationen zu chiffrieren, sondern auch Prüfsummen zu erstellen (*MAC* oder auch *MIC* genannt), mit deren Hilfe Teilnehmer authentifiziert oder nachträgliche Manipulationen an den übertragenen Daten aufgedeckt werden können. Dieser Mechanismus wird durch weitere Verfahren ergänzt um Schutz vor weiteren Angriffsszenarien, gegen die die „reine“ Verschlüsselung nicht hilft, zu erhalten.

Weiterhin ist es nötig, das Netzwerk vor fremden und nicht vertrauensvollen Teilnehmern zu schützen. So muss der Ursprung eines Paketes (*Authentifizierung*) und seine *Integrität* (ob eine Manipulation stattfand oder nicht) überprüfbar sein. Dies kann durch die Berechnung von einer *MAC* oder *MIC* realisiert werden. Das Verhalten des Netzwerks sollte durch Pakete, die einem fremden Ursprung entstammen, nach Möglichkeit nicht verändert werden, da der Angreifer sonst Einfluss nehmen kann. Der Umgang sämtlicher Nodes mit solchen Paketen muss genau definiert werden: Auf keinen Fall darf es möglich sein, dass das Herausgeben von Informationen provoziert werden kann, da diese Hinweise auf Schwachstellen geben, welche der Angreifer ausnutzen kann.

Ebenso stellt sich die Frage, wie die Schlüssel auf die Geräte verteilt werden können. Da dieser Typ von Hardware sehr beschränkte Ressourcen und Rechenkapazität besitzt, fällt die *asymmetrische Verschlüsselung* als Möglichkeit heraus, da diese zu rechenintensiv ist. So muss auf deren Vorteile ganz und gar verzichtet werden. Ein Konzept, um die Schlüssel auf die Netzwerknoten zu bringen, muss sich alleine auf die *symmetrischen Verschlüsselung* stützen. Und da liegt das Problem: Egal welches Schlüsselaustausch-Verfahren gewählt wird, jedes Verfahren basiert auf einen zentralen Schlüssel, der vorher auf die Geräte vorinstalliert oder anders gesichert übertragen werden muss.

Das bedeutet: wird ein solches Schlüsselaustausch-Protokoll gewählt, ist die gesamte Sicherheit gefährdet, sobald der zentrale Schlüssel, auf dem dieses Protokoll basiert, dem Angreifer in die Hände fällt. Damit lassen sich weitere Szenarien ableiten. So könnte ein

---

<sup>4</sup>Siehe [7, S. 53, 54]

### *3. Konzept und Szenarien*

Angreifer einen beliebigen Knoten in seine Hände bekommen und daraus den Schlüssel auslesen. Damit kann er den Netzwerkverkehr abhören, die Schlüssel zum chiffrierten Datenaustausch zwischen zwei Knoten abhören und eigene Knoten in das Netzwerk einspeisen, da auch die Authentifizierung bei der Anmeldung auf solch einem Schlüssel basiert.

Selbst wenn der Schlüssel sich in einem flüchtigen Speicher befindet und dieser beim Ablöten und damit beim Abtrennen von der Energiequelle verschwindet, gibt es Techniken, mit denen der Speicherinhalt auch bei flüchtigen Speicher erhalten und somit ausgelesen werden kann.<sup>5</sup>

Aber nicht nur durch den direkten Datenverkehr können Informationen gesammelt werden, auch die Abstrahlung des Mikroprozessors kann Aufschluss über intime Daten geben, wenn er nicht gut genug abgeschirmt ist. So ist es möglich durch Analyse dieser Elektromagnetischen Wellen Rückschlüsse auf den Schlüssel zu ziehen, auch wenn er im Gerät selbst erzeugt und noch nicht übertragen worden ist.

Vor allem sollte darauf verzichtet werden, Sicherheit durch komplizierte Programmierung zu schaffen oder Verschleierung des Codes. Auch wenn der Quellcode der Software nicht verfügbar ist, ist es trotzdem möglich die Software zu analysieren und nach zu programmieren.

So wurde im Verschlüsselungsstandart für Blue-Rays eine in Java programmierte Virtual Machine eingebettet, sodass das Medium nur auf Playern mit einer solchen Virtual Machine lauffähig war. Einige Monate später haben Unbekannte diese Virtual Machine durch Reverse-Engineering analysiert und das gesamte Sicherheitskonzept damit ausgehebelt.<sup>6</sup> Sämtliche Sicherheitsmechanismen, die auf Verschleierung und Geheimhaltung des Codes basieren, wurden innerhalb kurzer Zeit gebrochen.

## **3.2. Energieverbrauch**

Es stellt sich die Frage, welchen Einfluss die Verschlüsselung auf den Energieverbrauch der Geräte hat. Da diese zumeist über Batterien mit Energie versorgt werden, kann eine falsche Einschätzung dieses Einflusses die Lebensdauer aller Geräte enorm herabsetzen. Damit können Sensornetzwerke nicht in dem für sie spezifizierten Zeitraum ohne Störungen und Wartung funktionieren. Um diesen Einfluss nun abschätzen zu können, soll eine Energiemessung durchgeführt werden (Siehe Kapitel 5).

Bei den durch die Messung erbrachten Ergebnissen ist zu erwarten, dass sich ein deutlicher Anstieg des Energieverbrauchs zeigt, da zusätzliche Hardware für die Verschlüsselung mit Energie gespeist werden muss. Da die Verschlüsselungshardware unabhängig

---

<sup>5</sup><http://www.heise.de/security/meldung/Passwortklau-durch-gekuehlten-Speicher-182603.html>

<sup>6</sup><http://www.heise.de/newsticker/meldung/Foren-Mitglied-knackt-erweiterten-Blu-ray-Kopierschutz-BD-214819.html>

### 3.3. Versuchsaufbau

vom Prozessor ist, stellt sich die Frage, ob die Verarbeitungsdauer der Daten ansteigt oder nicht, da diese Hardware parallel zum Prozessor agieren kann, falls dies von der Software unterstützt wird.

## 3.3. Versuchsaufbau

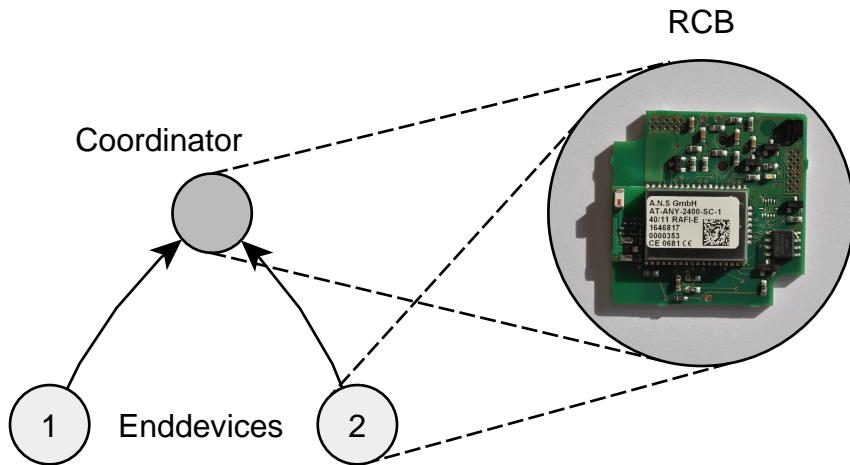


Abbildung 3.1.: Aufbau des Netzwerks, welches getestet und untersucht werden soll

Um die Bedingungen in einem Wireless Personal Area Networks (Drahtloses Persönliches Umgebungsnetzwerk) (WPAN) nachzustellen und analysieren zu können, wird ein Netzwerk aufgespannt, bestehend aus einem Koordinator und zwei verschiedenen Endgeräten. Der Coordinator soll ein Netzwerk erstellen und auf die Anmeldung zweier Endgeräte warten. Danach sollen sich beide Endgeräte unabhängig voneinander mit dem Koordinator verbinden und am Netzwerk anmelden. Beiden Endgeräten soll in diesem Prozess vom Koordinator eine Netzwerkadresse in Form einer *Short Address* zugewiesen werden. Im weiteren Verlauf senden dabei beide Endgeräte kontinuierlich und in periodischen Abständen Daten an den Coordinator.

Damit wird die typische Arbeit eines Sensornetzwerkes nachgestellt. In der Realität sind Sensornetzwerke jedoch um einiges größer, allerdings lassen sich die gewonnenen Erfahrungen auf größere Netzwerke verallgemeinern, da der Versuchsaufbau nur einen kleinen Teil größerer Netzwerke darstellt und damit die Grundstruktur dieser widerspiegelt. Der gerade beschriebene Ablauf lässt sich so in jedem größeren Netzwerk wiederfinden.

Dieser Ablauf soll nun durch die von der Implementierung beider Softwarestacks der Firma ATMEL (ATMEL AVR2025 MAC Stack und ATMEL BitCloud) angebotenen Sicherheitsmechanismen abgesichert und wo nötig und möglich erweitert werden.

### 3. Konzept und Szenarien

Als erstes gilt es zu überprüfen, ob das *End Device* auch am Netzwerk teilnehmen darf und vertrauenswürdig ist. Wie bereits erwähnt steht die *Asymmetrische Verschlüsselung* nicht zur Verfügung. Beide Implementierungen werden also höchstwahrscheinlich auf Verfahren zurückgreifen, die auf *Symmetrischer Verschlüsselung* aufbauen oder ganz auf Verschlüsselung verzichten. Auch muss das *End Device* entscheiden, ob der gefundene Coordinator legitimiert und autorisiert ist und sich nicht fälschlicherweise in ein nicht vertrauenswürdiges Netzwerk einwählt. Es wäre auch möglich, dass sich das *End Device* an einem falschen Coordinator anmeldet und dieser damit eine *Man-in-the-middle-Attack* ausführt.

Im Laufe dieser Prozedur wäre es wünschenswert beim Anmelden Schlüssel auszutauschen und das auf möglichst sichere Art und Weise. Da, wie gesagt, nur die *Symmetrische Verschlüsselung* zur Verfügung steht, ist es eine spannende Frage, wie der Schlüsselaustausch, wenn überhaupt, von den verschiedenen Implementierungen umgesetzt wurde. Geplant ist einmal die Verwendung eines Network Keys, der für alle Geräte gleichermaßen gültig ist sowie verschiedene Link Keys, die für jedes Paar Geräte, die miteinander kommunizieren wollen, einzigartig sind.

Sind die Schlüssel nun bei den einzelnen Parteien vorhanden, kann bereits die gesamte Kommunikation chiffriert werden. Es fehlt jedoch noch die Anmeldeprozedur, in der die Enddevices vom Coordinator registriert und ihnen eine Adresse zugewiesen wird. Es wäre natürlich von Vorteil diesen Schritt bereits zu verschlüsseln.

Nach erfolgreicher Anmeldung folgt der Datenaustausch. Dieser muss soweit abgesichert werden, dass zu übertragene Daten sowohl geheimgehalten, als auch deren Authentizität und Integrität überprüft werden können. Weiterhin dürfen nicht legitimierte Teilnehmer an keine Informationen kommen, die etwas über die Sicherheit des Netzwerkes aussagt, sowohl durch reines Abhören des Funkverkehrs, als auch durch Manipulation. Nur zugelassene Teilnehmer dürfen Einfluss auf das Netzwerk haben.

Es stellen sich nun folgende Fragen nach der Umsetzung:

- Wie wird die Authentifizierung beim Anmeldevorgang und Datenaustausch sicher gestellt?
- Wie werden, wenn überhaupt, die Schlüssel beim Anmelden ausgetauscht?
- Wie werden die Schlüssel für das Netzwerk und die verschiedenen Nodes verwaltet?
- Wie wird eine Integritätsprüfung der Nachrichten ermöglicht?
- Wie wird Rücksicht genommen auf spezielle Angriffe, wie z.B. eine Replay-Attack?
- Gibt es allgemeine Design-Fehler? (wie z.B. bei den Protokollen Wired Equivalent Privacy, HDCP oder CSS)

# 4. IMPLEMENTIERUNG

Im Laufe der Arbeit hat sich herausgestellt, dass wichtige Informationen über die Funktionsweise der Software (ATMEL MAC Stack und ATMEL BitCloud) nur schwierig oder überhaupt nicht zu beschaffen waren. Deswegen war ein integraler Bestandteil dieser Arbeit die Recherche nach solchen Informationen. ATMEL hat zu den Themen, die untersucht werden sollten, nur wenig Informationsmaterial zur Verfügung gestellt, welches teilweise veraltet oder sogar falsch gewesen ist (z.B. der Gebrauch von Konstanten, die nicht mehr existieren, oder die Verwendung von Header-Dateien, die nirgends aufzufinden waren). Besonders über die Funktionsweise der Sicherheitsarchitektur haben sich die Dokumentationen ausgeschwiegen, was dazu führte, dass nur durch die Analyse von Beispielprogrammen und deren Quelltext genauere Informationen in Erfahrung gebracht werden konnten. Teilweise musste sogar der Quelltext des ATMEL MAC Stacks selbst analysiert werden, um zu verstehen, was welche Konstanten bedeuten, und welche Parameter einer Funktion diese oder jene Auswirkung haben (z.B. der PIB-Index in der Funktion `wpan_mcsp_data_req()`). Dieser Prozess nahm eine Menge Zeit in Anspruch und führte in einigen Fällen zur Entdeckung weiterer Probleme, die vorher noch nicht abzuschätzen oder überhaupt sichtbar waren (so beispielsweise der Bug in der Datei `mac_security.c`). Aus diesen Gründen nimmt die Dokumentation des ATMEL MAC Stacks in diesem Kapitel einen großen Platz ein.

## 4.1. ATMEL MAC Stack 802.15.4

### 4.1.1. Verwendete Hardware

labelsub:hardware Bei der verwendeten Hardware handelt es sich um ein Radio Controller Board - RCB. Auf diesem befindet sich - neben einiger Peripherie wie eine LED und ein

#### 4. Implementierung

Sensor - eine IEEE 802.15.4/ZigBee RF Modul<sup>1</sup>. In diesem befindet sich der Transceiver, der Receiver und der Chip ATMega128RFA1<sup>2</sup>, sowie eine Hardwareinheit zum Verschlüsseln der Daten per AES. Um den Chip nun programmieren zu können, wurde auf das Gerät ATMEL AVR Dragon<sup>3</sup>.

##### 4.1.2. Konfiguration der Software und der Entwicklungsumgebung

**Software** Die Firma ATMEL stellt zur Entwicklung von Applikationen, die auf dem MAC Stack aufbauen, eine Entwicklungsumgebung zur Verfügung genannt AVR Studio (in diesem Fall die Version 5.1), mit deren Hilfe der gesamte Entwicklungsprozess bewältigt werden kann. Die Bibliothek MAC IEEE 802.15.4 (Version 2.7.1) liegt als Ordner vor, in dem sich in entsprechenden Unterverzeichnissen die benötigten Header- und Sourcecode-Dateien befinden. Über eine baumartige Struktur von Makefiles werden diese Dateien dann kompiliert und am Ende mit dem zu erstellenden Programm gelinkt.

**Makefiles** Die Bibliothek wird von AVR Studio durch Makefiles eingebunden. Für jedes Projekt gibt es zwei zentrale Makefiles mit den Namen `Makefile_Debug` und `Makefile`. Beide steuern das Kompilierverhalten des avr-gcc, wobei Erstere Debug-Informationen generiert, die ins Image geschrieben werden. Letztere verzichtet auf diesen Schritt. Des Weiteren werden durch sie Flags und Konstanten zur Steuerung und Konfiguration des MAC Stacks gesetzt oder gelöscht (z.B. `MAC_SECURITY_ZIP` um die Verschlüsselung einzuschalten). Dadurch sind diese in jeder Datei global verfügbar. Für den ATMega128RFA1 fanden sich im Verzeichnis `Applications` einige Beispielprogramme, deren Makefiles als Grundlage des hier vorgestellten Programms dienen. Für das Testnetzwerk wurden das Makefiles aus dem Verzeichnis `Applications/STB_Examples/Secure_Star_Network/Coordinator/ATMEGA128RFA1_RCB_6_3_SENS_TERM_BOARD` als Grundlage übernommen und für die spezielle Hardware angepasst. Der Quellcode des Programms wurde hingegen von Grund auf neu geschrieben.

**Anpassung** Folgende Konstanten mussten in den Makefiles angepasst werden:

- `_BOARD_TYPE = RCB_6_3_PLAIN`

Da auf dem hier verwendeten Board die Belegung und Verbindung der Peripherie und anderer Bauelemente mit dem Chip eine andere ist, muss diese Variable dementsprechend angepasst werden.

- `PATH_ROOT`

Diese Konstante zeigt nun auf einen anderen Ordner, damit der Quellcode nicht auf die Ordner des MAC Stack beschränkt ist.

<sup>1</sup>[http://www.an-solutions.de/webshop/modules-ghz-any2400sc-zigbee-module-p-35.html?cPath=1\\_7](http://www.an-solutions.de/webshop/modules-ghz-any2400sc-zigbee-module-p-35.html?cPath=1_7)

<sup>2</sup><http://www.atmel.com/devices/atmega128rfa1.aspx>

<sup>3</sup><http://www.atmel.com/tools/avrdragon.aspx>

#### 4.1. ATMEL MAC Stack 802.15.4

- In **CFLAGS** wurde der Parameter **-DUSBO** in **-DUART0** umgewandelt.

Dies stellt sicher, dass der bereits erwähnte SIO-Layer über USB genutzt werden kann. Mit diesem Hintergrund scheint es etwas paradox, statt **-DUSBO** die konstante **-DUART0** zu verwenden, allerdings benutzt die angegeschlossene Hardware eben die auf Universal Asynchronous Receiver Transmitter (UART)<sup>4</sup> basierende Technik, obwohl sie über USB kommuniziert. Dies hat schlichtweg nichts mit dem Anschluss, sondern eher etwas mit der Übertragung an sich zu tun. Zusätzlich muss im Quelltext noch die Funktion **pal\_sio\_init(SIO\_CHANNEL)** aufgerufen werden, um die Schnittstelle zu initialisieren.

- **CFLAGS += -DENABLE\_TSTAMP**

Schlussendlich wurde diese Zeile noch hinzugefügt, um alle verfügbaren Möglichkeiten des MAC Stacks zu prüfen.

Mit diesen Änderungen im Makefile ist es möglich ein Programm zu erstellen, welches auf der in dieser Studienarbeit untersuchten Hardware lauffähig ist.

Der Mac Stack wird mit einer Grundkonfiguration ausgestattet. Über verschiedene Konstanten, die im gesamten Quelltext verteilt sind und so je nach Bedingung das Einfügen oder Auslassen von Quelltext steuern, werden beispielsweise die Größe bestimmter Strukturen, die im Speicher liegen, definiert. So z.B. die Konstante **MAC\_ZIP\_MAX\_KEY\_TABLE\_ENTRIES**: Sie definiert die Anzahl der Felder und damit den Speicherplatz, mit denen Schlüssel zur chiffrierten Kommunikation gespeichert werden können. Da das Testnetzwerk aus zwei Enddevices und einem Coordinator besteht und ein globalen Network Key und zwei lokale Link Keys - je einer für ein *End Device* - verwendet werden sollen, muss diese Konstante, die den Standardwert 2 hat, auf 3 erhöht werden.

**Konstanten**

- `#define MAC_ZIP_MAX_KEY_TABLE_ENTRIES = 3` in der Datei **mac\_api.h**

Probleme gab es hingegen mit der Datei **mac\_security.c** im Verzeichnis **MAC/Src**. In ihr befindet sich ein Bug, der das Auffinden einer *Extended Address* zu einem Device verhindert und damit eine falsche Nonce beim Empfang von verschlüsselten Daten erzeugt. Folglich können die Daten weder entschlüsselt, noch kann die richtige MIC errechnet werden, was intern den Rückgabewert **STB\_CCM\_MICERR** in der Bibliothek verursacht und damit die Meldung einer Indikation **status\_ind** an die Applikation zurück gibt statt der entschlüsselten Nachricht. Eine korrigierte Version dieser Datei liegt der Arbeit bei (Siehe beigelegte CD). Auf die interne Arbeitsweise dieser Sicherungsschicht wird später noch genauer eingegangen.

**Bug**

Nach der Version des IEEE 802.15.4 Standards aus dem Jahre 2003 wurde eine modifizierte Version aus dem Jahre 2006 veröffentlicht. Eine der Änderungen betrifft die

**Versionen**

---

<sup>4</sup>Siehe UART im Glossar

#### 4. Implementierung

Verwaltung der gespeicherten Schlüssel. Um diese neu hinzugekommene Eigenschaft einzuschalten, muss die Option `MAC_SECURITY_2006` im genutzten Makefile gesetzt werden. Leider kompiliert in dieser Konfiguration der Compiler nicht bis zum Schluss und bricht mit einer Fehlermeldung ab. Damit können die neu hinzugekommenen Möglichkeiten im Rahmen dieser Studienarbeit nicht untersucht werden.

#### 4.1.3. Programmierung des MAC Stack

##### **Finite State Machine**

Eine Applikation und den MAC Stack selbst kann man sich am besten vorstellen als eine *Finite State Machine*. Sie durchläuft verschiedene Zustände und reagiert je nach Rück- oder Eingabewerte, indem sie abhängig von diesen in die verschiedenen Zustände wechselt.

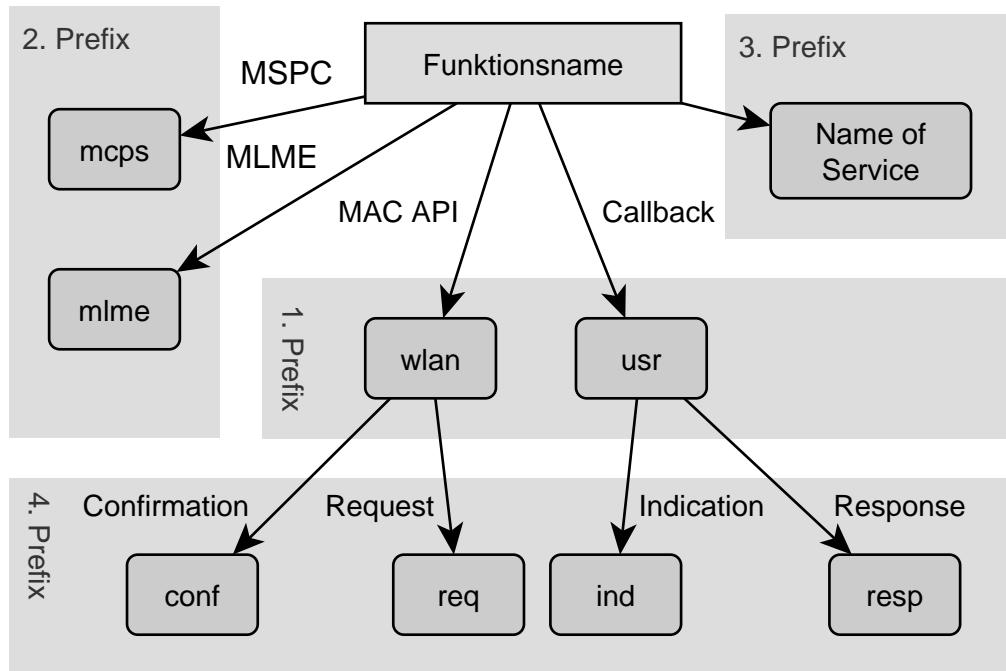


Abbildung 4.1.: Aufbau der Bezeichnungen von den Funktionen, mit denen der ATTEL MAC Stack programmiert werden kann.

##### **Queue**

Der MAC Stack selbst besitzt eine Queue, die von ihm selbst oder der Applikation mit Nachrichten gefüllt wird. Eine von der Applikation aufgerufene Funktion, die vom MAC-Stack bereitgestellt wird, oder Ereignisse von außen verursachen eine Nachricht, die dann in die Queue geschoben wird. Der MAC Stack arbeitet nun alle Nachrichten in der Queue nach und nach ab. Somit kann es passieren, dass nach Aufruf eines Services durch die Applikation die von ihr generierte Nachricht zeitverzögert bearbeitet wird, da noch andere Nachrichten in der Queue vorhanden sein können und damit Vorrang haben.

## 4.1. ATMEL MAC Stack 802.15.4

Damit ist der Ablauf des Programms *asynchron*, die Beantwortung der Anfrage kann also je nach Zustand früher oder später erfolgen.

IEEE 802.15.4 soll die zwei untersten Schichten des *OSI Schichtenmodells* übernehmen. Die restlichen Schichten sollen von anderen Protokollen implementiert werden, die dann auf IEEE 802.15.4 aufbauen und auf dessen Funktionen zurückgreifen können. Damit das funktioniert, muss IEEE 802.15.4 standardisierte Schnittstellen anbieten, die unabhängig von der Implementierung den selben Funktions- und Dienstumfang besitzen. Diese Schnittstellen werden *SAP* genannt. Der ATMEL MAC Stack implementiert diese Service Access Points durch Funktionen, die einerseits vom MAC Stack angeboten und von der Applikation aufgerufen werden können oder andererseits - sogenannte Callback-Funktionen - von der Applikation implementiert und wiederum vom MAC Stack aufgerufen werden können.

**SAP**

Das Abstrahierungskonzept bietet nach außen hin zwei Elemente an, die den Zugriff mit Hilfe der SAPs auf das Netzwerk und das Device strukturieren und ermöglichen. Das eine Element wird *MAC Common Part Layer (MCPS)*, das andere *MAC Layer Management Entity (MLME)* genannt. MCPS kümmert sich allein um den Datenaustausch zwischen verschiedenen Devices. MLME hingegen bietet Zugang zu den verschiedenen Layern des MAC Stacks. Dazu gehört auch die Verwaltung einer Datenbank, die wichtige Informationen über das Netzwerk, die Devices und deren Zustände beinhaltet. Die Datenbank wird *MAC Layer PAN Information Base* genannt und kann über die MLME-Services gelesen und geschrieben werden.

**MCPS, MLME,  
PIB**

Die Namensgebung der Funktionen, über die die Applikation Zugang zu den SAPs bekommt, müssen gewissen Regeln folgen. Ihre Namen setzen sich aus vier verschiedenen Präfixen zusammen, die alle durch den Unterstrich `_` voneinander getrennt werden. Jeder Präfix spiegelt die Zuordnung der Funktion zu einer bestimmten Klasse wider. Dieses Schema wird in Abbildung 4.1 dargestellt.

### • 1. Prefix

- handelt es sich um eine Funktion, die vom MAC Stack aufgerufen wird, lautet der Präfix `usr`
- im Falle dass es eine Funktion der MAC API ist, lautet der Präfix `wpan`

### • 2. Prefix

- bietet die Funktion Zugriff auf Services des MLME, lautet der Präfix `mlme`
- handelt es sich um den Zugriff auf Services zum Datenaustausch, ist das Präfix `mcps`

### • 3. Prefix

- an dieser Stelle kommt der Name oder ein kürzerer Bezeichner des aufgerufenen Services selbst

## 4. Implementierung

### • 4. Prefix

- bei einer Anfrage an den MAC Stack durch die Applikation (*Request*) lautet der Präfix **req**
- soll die Funktion eine Antwort des MAC Stacks auf ein Request darstellen (*Confirmation*), lautet der Präfix **conf**
- versucht der MAC Stack die Applikation mit Hilfe der Funktion über ein äußeres Ereignis zu informieren (*Indication*), lautet der Präfix **ind**
- soll die Funktion eine Antwort der Applikation auf eine Indication vom MAC Stack sein (*Response*), lautet der Präfix **resp**

Folgende Beispiele sollen das Namensschema veranschaulichen:

- Um den Zustand eines Devices im Netzwerk zurückzusetzen, also um sämtliche Optionen und Informationen, ob beispielsweise das Device einem Netzwerk angehört, zu löschen, wird die Funktion `wpan_mlme_reset_req()` vom Programm aufgerufen.
- Der MAC Stack reagiert auf eine solche Anfrage mit dem Aufruf der Callback-Funktion `usr_mlme_reset_conf()`.
- Um anzuzeigen, dass ein Device ein Datenpaket empfangen hat, wird vom MAC Stack die Callback-Funktion `usr_mcps_data_ind()` aufgerufen.
- Tritt ein Device dem Netzwerk bei, wird dem Coordinator dies mit dem Aufruf seiner Funktion `usr_mlme_associate_ind()` bekannt gemacht. Dieser reagiert wiederum mit der Funktion `wpan_mlme_associate_resp()`

### Aufbau

Es gibt verschiedene Möglichkeiten auf den durch den IEEE 802.15.4 MAC Stack angebotenen Funktionsumfang zuzugreifen. Dieser besteht aus verschiedenen Komponenten, die alle aufeinander aufbauen und sich in ihrer Funktion gegenseitig ergänzen. Möchte die Applikation nur auf die SAPs zugreifen, die laut Standard ausreichen um den vollen Funktionsumfang des IEEE 802.15.4 Stacks nutzen zu können, kann diese den inneren Aufbau getrost ignorieren. Sie stellen eine Schnittstelle zwischen dem MAC Stack und der Applikation dar und verbergen unnötige Informationen und Aspekte ganz nach dem Ansatz der objektorientierten Programmierung.

#### 4.1. ATMEL MAC Stack 802.15.4

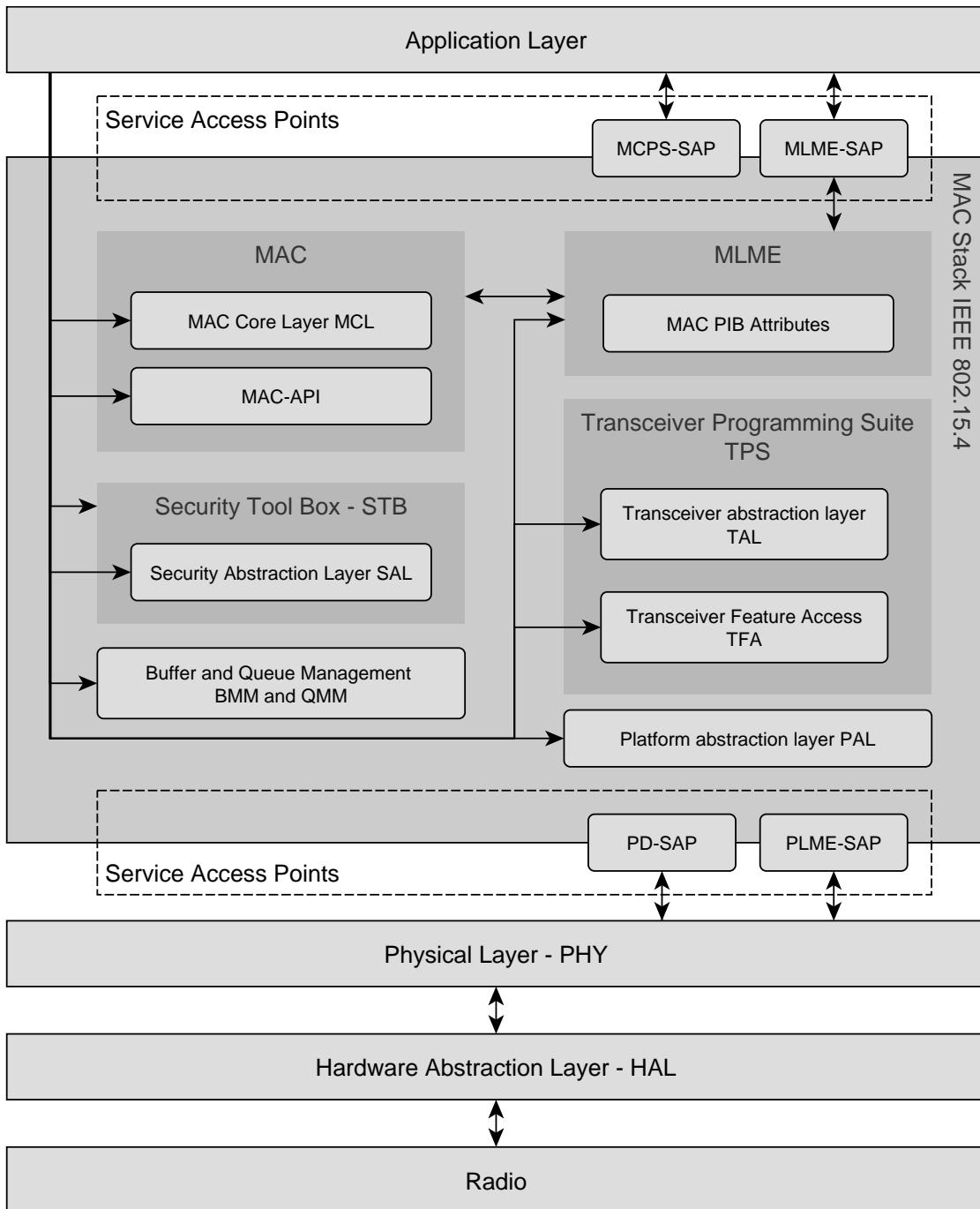


Abbildung 4.2.: Grundstruktur des MAC Stack sowie das Zusammenspiel zwischen den äußeren Schnittstellen (SAP) und den inneren Komponenten.

## 4. Implementierung

**Tiefere Schichten** In einigen Fällen kann es jedoch von Vorteil sein, die SAPs zu umgehen und auf Routinen tiefer liegender Schichten zurückzugreifen. Beispielsweise könnte in einigen Szenarien auf Standards verzichtet werden um das Protokoll um eigene spezifische Funktionen zu erweitern. Dies geht zwar auf Kosten der Kompatibilität, ist manchmal jedoch unumgänglich. Dem Programmierer stehen in diesem Falle Funktionen zur Verfügung, mit deren Hilfe er aus der Applikation heraus Verbindung zu Layern herstellen kann, die der Hardware näher sind. So können Effizienz und Flexibilität gesteigert werden, während der Rest des Programms auf den Code des MAC Stacks zurückgreift. Auch kann so Code des MAC Stacks durch eigenen Code ersetzt werden. Ist dies der Fall, muss jedoch darauf geachtet werden, dass der eigene Code nicht mit dem MAC Stack kollidiert, da sich ansonsten beide bei der Konfiguration des Systems in die Quere kommen. Dies erfordert Wissen über den genauen Ablauf des Codes innerhalb des MAC Stacks und dessen Routinen.

### 4.1.4. Zugriff auf die Security-Schicht

**Hierarchie** Es gibt nun drei verschiedene Möglichkeiten um Zugang zu Routinen der Sicherheitsmechanismen sowie der Verschlüsselungshardware selbst zu erlangen. Jede dieser Möglichkeiten wird durch eine Schicht innerhalb des Stacks symbolisiert. Sie lassen sich in einer Hierarchie anordnen, da jede Schicht die Möglichkeiten und Funktionalität der unter ihr liegenden Schicht ergänzt und erweitert und gleichzeitig auf deren Routinen zurückgreift. Alle drei Layer bauen also - wie es im MAC Stack üblich ist - aufeinander auf. Die Schnittstelle der obersten Schicht mit ihrer Umgebung wiederum wurde mit den SAPs realisiert und beinhaltet damit alle Spezifikationen des IEEE 802.15.4 Standards. Grafik 4.3 veranschaulicht diesen Aufbau.

#### MAC-Security

**MAC-Security** Die oberste Ebene, wie sie in Abbildung 4.3 zu sehen ist, wird durch die Routinen des MAC Stacks dargestellt. Auf sie kann mittels der SAPs zugegriffen werden. Sie bestehen zum einen aus der Verwaltung und automatischen Zuordnung der Schlüssel zu den entsprechenden Devices und zum anderen aus der Implementierung des *CCM/CCM\*-Mode*. So müssen über die SAPs die nötigen PIB-Attribute, acht an der Zahl, gesetzt und konfiguriert werden. Im wesentlichen handelt es sich dabei um folgende Attribute:

- macDefaultKeySource
- macSecurityLevelTableEntries
- macSecurityLevelTable
- macKeyTableEntries
- macKeyTable

#### 4.1. ATMEL MAC Stack 802.15.4

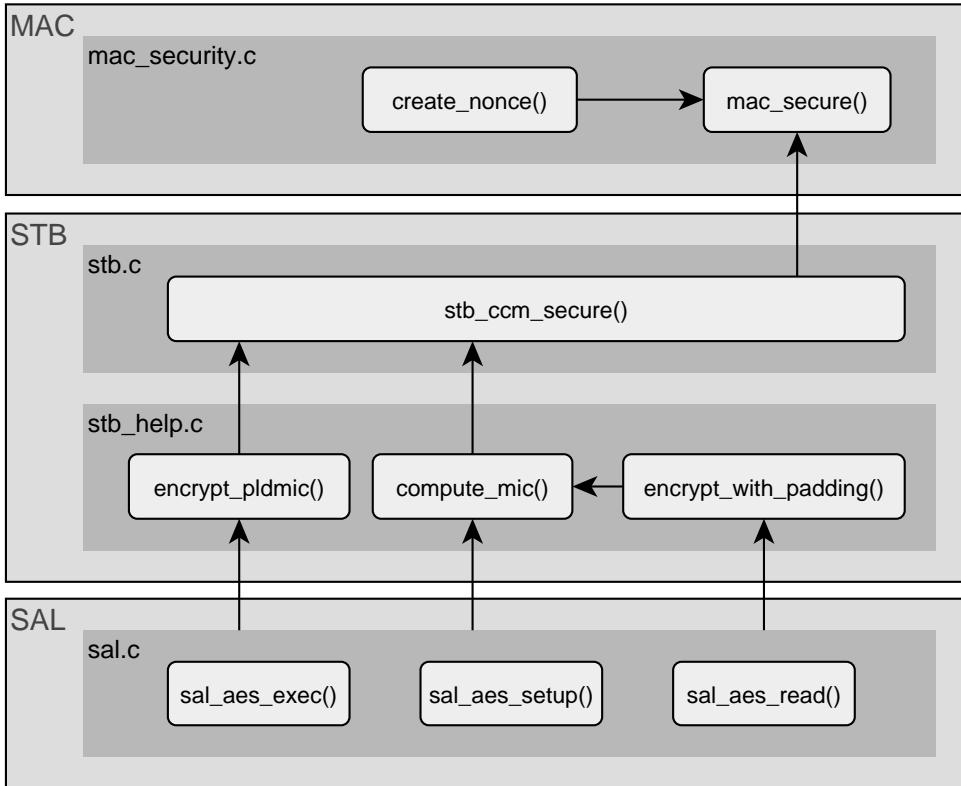


Abbildung 4.3.: Aufbau des ATMEL MAC Stack und Einordnung in die Layer-Struktur

- `macDeviceTableEntries`
- `macDeviceTable`
- `macFrameCounter`

Wie man unschwer erkennen kann, gibt es drei der Attribute noch einmal mit dem Zusatz **Entries**. Der MAC Stack hat keine automatische Speicherallokierung, was auf einem Embedded Device mit knappen Speicherressourcen aufgrund von Defragmentierung kaum möglich ist. So muss dieser eine Speicherstruktur mit fester Größe vorhalten, um die Daten organisieren zu können. Ihre Größe kann im Programmablauf nicht verändert werden und muss somit gut gewählt werden. Jedes zu speichernde Attribut wird durch eine Struktur repräsentiert (**struct**), deren Datenfelder die nötigen Werte enthalten. Sie liegen im Speicher direkt hintereinander. Ihre Anzahl bestimmt die Höhe des Speicherverbrauchs. Dazu verwendet der MAC Stack Konstanten mit vordefinierten Werten. Der dazugehörige Code befindet sich in der Datei `/MAC/Inc/mac_api.h`. Damit der MAC Stack nun unter den verschiedenen Instanzen eines Attributs unterscheiden kann, benötigt er einen Index, wie bei einem Array. Die Verwaltung dieser als Array angeordneten Instanzen muss also der Programmierer übernehmen. Z.B. benötigt jedes Device für die

#### 4. Implementierung

verschlüsselte Kommunikation mit einem anderen Device dessen *Extended Address*, die der MAC Stack im PIB Attribute `macDeviceTable` speichert. Bei mehreren Devices muss es also auch mehrere Instanzen des Attributes geben, eine Instanz für jedes Device. Nur Wenn die Verschlüsselung mittels `MAC_SECURITY_ZIP` eingeschaltet ist, wird dieser Index beim setzen der Attribute bei der Funktion `wpan_mlme_set_req()` als Parameter mit übergeben. Ansonsten findet sich der Index als Variable nicht in der Parameterliste der Funktion wieder.

**macDefaultKeySource** Hierbei handelt es sich um eine Art ID, mit deren Hilfe ein Schlüssel identifiziert werden kann. Um beispielsweise einen Schlüssel für das gesamte Netzwerk zu nutzen, einen *Network Key*, kann bei der Verschlüsselung angegeben werden, dass über diese ID der Schlüssel ausgewählt werden soll. Mithilfe eines weiteren Key-Index kann so zwischen verschiedenen Schlüsseln unterschieden werden. Damit ist es möglich für eine ID einen Satz an Schlüsseln zu verwalten. Diese Möglichkeit entspricht dem Mode 0x01 (explicit key identification). Beim Versenden der Daten mittels der Funktion `wpan_mcps_data_req()` wird der Mode als Argument mit angegeben. Im PIB Attribut `macKeyTable` kann einem Schlüssel diese ID und der dazugehörige Key-Index zugeordnet werden.

**macSecurityLevelTable** In dieser Struktur wird festgelegt, welche Frame-Typen verschlüsselt werden sollen, nach welcher Frame-ID bei der Verschlüsselung unterschieden werden soll, wie hoch der Default-Security-Level ist und ob dieser überschrieben werden darf. Der ATMEL MAC Stack bietet hierfür jedoch keinerlei Konstanten an, um die richtigen Werte in die Felder zu schreiben. Sie wurden deswegen in der Applikation definiert. Der ATMEL MAC Stack ist ebenfalls auch nur in der Lage Datenframes zu verschlüsseln. Im IEEE 802.15.4-Standard ist jedoch spezifiziert, dass auch Beacon-Frames und Command-Frames verschlüsselt werden können.<sup>5</sup> <sup>6</sup>. Auch nach der Frame-ID unterscheidet der ATMEL MAC Stack nicht, obwohl dies so spezifiziert ist. Die Konstante `MAC_ZIP_MAX_SEC_LVL_TABLE_ENTRIES` legt die maximale Anzahl an Array-Elementen bei der Kompilierung fest.

**macSecurityLevelTableEntries** Dieses Attribut legt fest, wie viele Instanzen vom Attribut `macSecurityLevelTable` maximal verwaltet werden sollen.

**macKeyTable** Die Verwaltung der Schlüssel übernimmt dieses Attribut. Jede Instanz beschreibt dabei einen Key und speichert dessen Informationen, wie z.B. zu welchem Device er gehört. die Konstante `MAC_ZIP_MAX_KEY_TABLE_ENTRIES` legt deren Standartanzahl an Instanzen fest. Der Defaultwert beträgt 2 und musste für diese Studienarbeit auf 3 erhöht werden, um drei Schlüssel verwalten zu können.

---

<sup>5</sup>Siehe IEEE S.139 Table 79

<sup>6</sup>Siehe IEEE S.209 Table 92

#### 4.1. ATMEL MAC Stack 802.15.4

**macKeyTableEntries** Dieses Attribut legt fest, wie viele Instanzen vom Attribut macKeyTable maximal verwaltet werden sollen.

**macDeviceTable** Jedes Device muss für eine verschlüsselte Kommunikation die Informationen anderer Devices kennen. Diese Informationen, wie beispielsweise die *Extended Address* oder der Framecounter, werden in diesem Attribut gespeichert. Diese Attribute können jedoch erst für ein Device gesetzt werden, wenn es dem Netzwerk bereits beigetreten ist und sich beim Coordinator angemeldet hat. Ansonsten wird eine Fehlermeldung zurückgegeben.

**macDeviceTableEntries** Dieses Attribut legt fest, wie viele Instanzen vom Attribut macDeviceTable maximal verwaltet werden sollen.

**macFrameCounter** Mithilfe dieses Zählers (engl. Counter) - je ein einzigartiger Zähler für ein Device - kann jeder Frame durchnummeriert werden, um z.B. *Replay-Attacks* zu verhindern. Dieser Framecounter geht auch in die Berechnung in die Nonce für die Verschlüsselung laut *IEEE 802.15.4*-Standard. Sobald der Framecounter seinen maximalen Wert erreicht hat (Da er ein 32-Bit Wert ist, ist dies die Zahl 0xFFFFFFFF), werden alle zur Verschlüsselung genutzten Informationen als ungültig erklärt und müssen neu registriert werden. Das gesamte Schlüsselmaterial muss ausgetauscht werden. Dies hat den Hintergrund, dass bei der Generierung der Nonce der Framezähler mit einfließt. Die Nonce muss jedoch stets einzigartig sein, dieselbe Nonce darf nicht wiederverwendet werden. Sollte der Framecounter sich wiederholen dürfen, würde irgendwann eine Nonce generiert werden, die bereits verwendet worden ist. Dies würde die Sicherheit des *CCM/CCM\*-Mode* stark gefährden.

Um die Verschlüsselung des Datenverkehrs nutzen zu können, müssen die Schlüssel von der Applikation im MAC-Stack über diese PIB Attribute registriert werden, sowie die *Short Address* und *Extended Address* der Devices, mit denen kommuniziert werden soll. Der MAC Stack erhält durch die in den empfangenen Datenpaketen befindlichen Header die notwendigen Informationen, um den Sender identifizieren zu können. So kann der MAC Stack über die *Short Address* des Senders in seiner Datenbank dessen *Extended Address* ermitteln. Diese wird benötigt, um die Nonce generieren zu können, die zur Ver- und Entschlüsselung im CCM/CMM\*-Mode notwendig ist. Kann der MAC-Stack keine *Extended Address* finden, gilt der Kommunikationsversuch als gescheitert und wird mit Hilfe einer entsprechenden Statusmeldung über den Aufruf der Callback-Funktion `usr_mlme_comm_status_ind` der Applikation bekannt gemacht. Auch wenn die Nachricht z.B. eine falsche MIC enthält, wird dies mit dem Aufruf selbiger Funktion kenntlich gemacht.<sup>7</sup> Im anderen Fall wird die Callback-Funktion `usr_mcps_data_ind` aufgerufen und signalisiert damit der Applikation, dass alles funktioniert hat.

---

<sup>7</sup>Erst registrieren in der PIB Attribute, wenn Device bereits beigetreten ist

## 4. Implementierung

**Schlüsselverwaltung** Der MAC Stack nimmt dem Programmierer die Arbeit ab, sämtliche verwendeten Schlüssel zu verwalten. Da es in einem Netzwerk wünschenswert ist, mehrere Schlüssel zu verwenden und nicht nur einen einzigen, um den Datenverkehr zu sichern, kommt diesem Thema große Bedeutung zu. So gibt es drei verschiedene Mechanismen, mit denen der MAC Stack automatisch den passenden Schlüssel für die Ver- und Entschlüsselung auswählt. Zwei der Mechanismen, die seit den Anfängen des IEEE 802.15.4 Standards spezifiziert worden sind, hat der ATMEL Stack umgesetzt. Der dritte Mechanismus, der 2006 hinzugekommen ist, kann im ATMEL MAC Stack nicht genutzt werden, da hierzu der Schalter `MAC_SECURITY_2006` aktiviert werden muss, was beim Kompilieren jedoch zu einem Fehler führt. Dieser Modus wird auch nur im Falle eines Command-Frame angewendet, da dieser Type eine Frame-ID besitzt, nach der der Schlüssel ausgewählt wird. Der ATMEL Stack kann aber nur Data-Frames verschlüsseln.

**Mode 0x01 - Explicit Key Identification** Eine der beiden Möglichkeiten funktioniert nach dem bereits beim PIB Attribut `macDefaultKeySource` erklärten Modell. Über dieses Attribut wird also ein globaler Schlüsselbund identifiziert, aus dem mit Hilfe eines Key-Index ein Schlüssel ausgewählt werden kann.

**Mode 0x00 - Implicit Key Identification** Schlussendlich bietet der MAC Stack einen Mechanismus an, mit dessen Hilfe der Schlüssel automatisch für jedes Device einzeln ausgewählt werden kann. Dieses Verfahren gliedert sich in zwei unterschiedliche Abläufe auf. Zum einen kann die ID, die den Schlüssel identifiziert, aus der *Extended Address* des Empfängers bestehen, zum anderen aus der *Short Address* des Empfängers und der PAN-ID des Sub-Netzwerks, in dem er sich befindet. Die ID im ersten Fall ist 72 Bit groß, da sie sich aus der *Extended Address* (8 Byte) und aus der Key-ID (1 Byte) zusammensetzt. Hierzu muss noch die Variable `LookupDataSize` im PIB Attribute `macKeyTable` mit einer 0x00 besetzt werden, da dieser Wert eine Größe von 9 Byte entspricht. Im zweiten Fall muss diese Variable mit dem Wert 0x01 belegt werden. Dieser Wert zeigt an, dass die ID aus nur 5 Byte besteht, der PAN-ID (2 Byte), der *Short Address* (2 Byte) und der Konstante 0x00 (1 Byte).

Alle so eben beschriebenen Mechanismen sind in der Datei `/MAC/Src/mac_security.c` zu finden.

**Security Level** Laut IEEE 802.15.4 Standard werden der Applikation verschiedene Security Level angeboten, die sich untereinander in der Länge der MIC unterscheiden und darin, ob der Payload verschlüsselt werden soll oder nicht.

Im MIC Attribut `macSecurityLevelTable` kann der minimale Security-Level gesetzt werden, der im gesamten Programm als untere Schranke gilt. Ansonsten muss der Security-Level beim Aufruf der Funktion `wpan_mcps_data_req()` als Parameter mit übergeben werden. MIC-32, MIC-64 und MIC-128 in den Leveln 1-3 geben an, wie groß die MIC,

#### 4.1. ATMEL MAC Stack 802.15.4

Security level identifier	Security control field b2 b1 b0	Security attributes
0x00	000	None
0x01	001	MIC-32 Bit
0x02	010	MIC-64 Bit
0x03	011	MIC-128 Bit
0x04	100	ENC
0x05	101	ENC-MIC-32 Bit
0x06	110	ENC-MIC-64 Bit
0x07	111	ENC-MIC-128 Bit

Tabelle 4.1.: Die Security-Levels

die dem Payload angehangen wird, sein darf. Dabei macht es in der Berechnung der MIC keinen Unterschied, welche Größe gewählt wird. Die Berechnung selbiger basiert, bedingt durch die Verwendung von *CCM/CCM\** in IEEE 802.15.4, auf dem *CBC-MAC-Mode* mit einer Blockgröße von 128 Bit. Das Ergebnis, die MIC also, hat damit ebenfalls immer eine Größe von 128 Bit. MIC-32 und MIC-64 bedeuten nichts anderes, als das die MIC-128 auf 32 Bit bzw. 64 Bit gekürzt werden, indem der Rest einfach abgeschnitten wird. Eine kleinere MIC bedeutet somit keinen Geschwindigkeitsvorteil bei ihrer Berechnung. Allein die geringere Größe kann hier als Vorteil gezählt werden, die bei einer Beschränkung eines Datenpakets auf etwa 100 Byte einiges an Gewicht hat.

Ab dem Level 4 kommt die Verschlüsselung hinzu und wird in allen darüber liegenden Leveln mit der MIC kombiniert. Level 0 bedeutet ganz einfach, dass der Payload weder mit einer MIC geschützt, noch verschlüsselt wird.

**Nonce** Wie bereits in Kapitel 2.4.6 CCM/CCM\* - Counter with CBC-MAC beschrieben, lässt der Standard IEEE 802.15.4 den Aufbau der Nonce unspezifiziert, legt jedoch fest, wie sich der restliche IV zusammensetzt. Das Layer mac\_security kümmert sich also um die Generierung der Nonce, während das darunterliegende Layer CCM/CCM\*, welches durch die *Security Toolbox - STB* implementiert wird, den Rest des IVs generiert. Da keine Spezifikation für die Nonce vorliegt, übernimmt der MAC Stack die Spezifikationen aus ZigBee<sup>8</sup>.

Aus der Abbildung 4.4 geht hervor, dass die Nonce aus der *Extended Address*, dem Framecounter und dem Security-Level besteht. An diesem Punkt erklärt sich auch, warum es sich bei der Berechnung der Prüfsumme mittels *CBC-MAC* nicht mehr um eine MAC, sondern um eine MIC handelt. Da durch die Nonce auch der Framecounter in den IV mit einfließt, um das Netzwerk beispielsweise vor einer *Replay-Attack* zu schützen, erweitert sich die Funktion der MAC, da die Berechnung dieser nun vom Framecounter abhängig

---

<sup>8</sup>Siehe [14]

#### 4. Implementierung

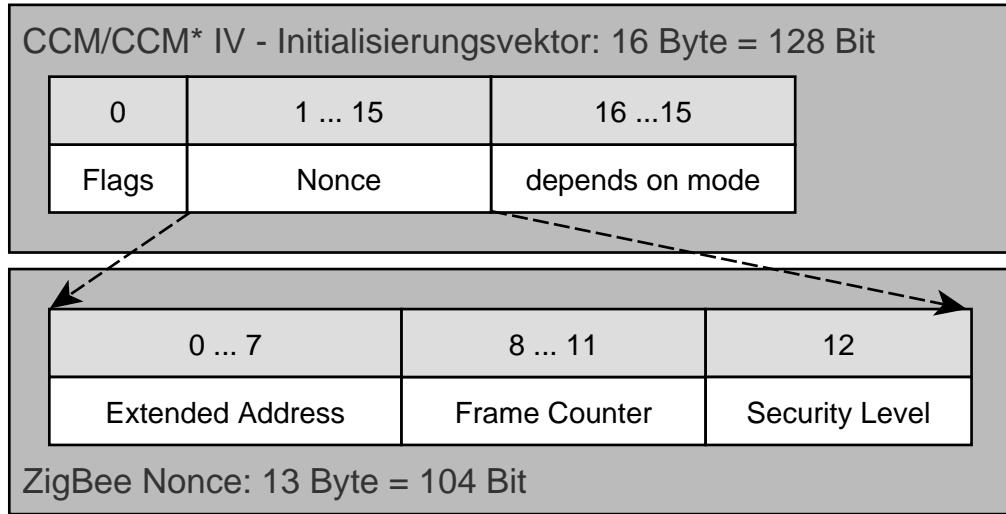


Abbildung 4.4.: Aufbau der Nonce, wie er im ZigBee-Standard vorgesehen ist.

ist. Ein falscher Framecounter erzeugt somit eine falsche MAC/MIC. Um dies kenntlich zu machen, wurde die Bezeichnung *Message Integrity Check - MIC* eingeführt.

**Security Tool Box - STB** Dieses Layer implementiert die Funktionen und Spezifikationen des *CCM/CCM\*-Mode*. Das bedeutet auch, dass sie den von *CCM/CCM\** spezifizierten Bereich des IVs generiert. Damit teilt sich die Erstellung des IVs zwischen der STB-Schicht und der mac\_security-Schicht auf.<sup>9</sup>

Allein die Funktion `stb_ccm_secure()` ist notwendig, um einen Datenblock mittels *CCM/CCM\** zu schützen. Sie kümmert sich auch um die Umsetzung der verschiedenen Security-Level. (An diesem Punkt wird es schwierig die Grenze zu ziehen zwischen den verschiedenen Standards und den Implementierungen. Die Security-Levels sind in IEEE 802.15.4 spezifiziert, und nicht in *CCM/CCM\**.) Beim Entschlüsseln eines Datenblocks kann automatisch auf die Richtigkeit der MIC geprüft werden. Der Speicherbereich, der der Funktion als Pointer übergeben werden muss, muss mindestens noch 128 Bit zusätzlich am Ende zur freien Verfügung haben, da hier die MIC, mit maximal 128 Bit Größe, von der Funktion abgelegt wird. Ansonsten schreibt die Funktion in einen falschen Speicherbereich, was zu Systemabstürzen führen kann.

Außerdem muss darauf geachtet werden, dass der *radio controller* und damit der Verschlüsselungsschip eingeschaltet ist, da ansonsten das Flag zur Signalisierung des Endes der Verschlüsselung nicht gesetzt wird und die Funktion, welche darauf wartet, in einer Endlosschleife hängen bleibt. Dies kann beispielsweise mit dem Setzen des PIB Attributes `macRxOnWhenIdle` auf `true` bewirkt werden.

<sup>9</sup>Es kann kein Bereich im Payload angegeben werden, der nicht verschlüsselt werden soll: keine Routing-Informationen

Die *Security Abstraction Layer - SAL* ist die Unterste von allen drei Schichten. Sie kommuniziert auf direktem Wege mit der Verschlüsselungshardware und abstrahiert diese nach oben hin. Mit ihr können der Modus der Verschlüsselung festgelegt, die Verschlüsselung selbst durchgeführt, das Resultat gelesen und nach dem Aufwachen des Chips aus dem Sleep-Mode der Zustand der Verschlüsselungshardware wieder hergestellt werden. Dazu bietet dieser Layer folgende Funktionen an:

- `sal_init()` Initialisiert das SAL-Layer. Für den ATMega128rfa1 ist diese Funktion leer, in Fällen anderer Hardware kann diese jedoch notwendigen und wichtigen Code enthalten.
- `sal_aes_setup(uint8_t *key, uint8_t enc_mode, uint8_t dir)` Setzt den genutzten Schlüssel zur Ver- oder Entschlüsselung (`key*`), den Mode (`enc_mode` mit den möglichen Konstanten `AES_MODE_ECB` und `AES_MODE_CBC`), sowie ob ver- oder entschlüsselt werden soll (`dir` mit den Möglichen Werten `AES_DIR_ENCRYPT` und `AES_DIR_DECRYPT`)
- `sal_aes_restart()` Stellt den Systemzustand nach dem Sleep-Mode wieder her.
- `sal_aes_exec(uint8_t *data)` ver- oder entschlüsselt einen 128-Bit Block Daten
- `sal_aes_read(uint8_t *data)` liest das Resultat der Ver- oder Entschlüsselung

Im Verzeichnis **SAL** finden sich alle wichtigen Code-Module, aus denen sich die SAL-Schicht zusammen setzt. Unterverzeichnisse werden den verschiedenen Hardware-Implementierungen zugeordnet. Für diese Studienarbeit ist dabei nur das Verzeichnis **/SAL/ATMEGARF\_SAL/Src** von Interesse, da hier der Hardwarespezifische Code für den ATMega128rfa1 zu finden ist. Für eine andere Hardwarebasis kann es sein, dass sich die SAL-Schicht aus anderen Funktionen zusammen setzt. Wie bereits im Kapitel 2.4.6 erläutert wurde, basiert *CCM/CCM\** auf verschiedenen anderen Verschlüsselungsmodi. Diese Modi werden durch das SAL-Modul implementiert. Es übernimmt die Verschlüsselung im ECB- und *CBC-Mode*. Alles andere, was vom IEEE 802.15.4-Standard gefordert wird, wie zum Beispiel die Berechnung einer MIC oder die Erstellung eines IV und einer Nonce, wird von den darüber liegenden Schichten übernommen oder muss vom Programmierer selbst umgesetzt werden.

#### 4.1.5. Programmablauf

Das Erarbeiten der Informationen, wie der MAC Stack zu programmieren ist, war nicht trivial. Die Dokumentation war recht undetailliert, besonders was das Zusammenspiel der einzelnen Funktionen anging.

Die Applikation hat den Aufbau einer *Finite State Machine* (Siehe Abbildung 4.7 und 4.5). Je nach den Eingaben, die sie erhält, wechselt sie in einen anderen Zustand, um auf Anfragen reagieren zu können. In dieser Applikation, die zu dieser Studienarbeit ge-

#### 4. Implementierung

hört, wurde auf eine spezielle Schleife, in der in verschiedene Zustände gewechselt werden konnte, verzichtet. Es sollte sich so nah wie möglich an den mitgelieferten Beispielprogrammen orientieren, in denen der Zustand allein durch den Aufruf von Request- und Confirmation-Funktionen sowie Indication- und Responsefunktionen gewechselt wird. Es sollten möglichst viele Fehler vermieden und ausgeschlossen werden, die durch eine zentrale Schleife entstanden wären, da das Zusammenspiel zwischen den einzelnen Funktionen des MAC Stack noch unklar war und dadurch entstandene Fehler mit Hilfe der Beispielprogramme als Orientierung leichter aufzuspüren waren.

#### Coordinator

Das Programm beginnt damit den Zustand des MAC Stacks zurückzusetzen und die einzelnen Optionen für das Device und das Netzwerk mit neuen Werten zu belegen, um eine solide Grundkonfiguration zu erhalten. Als erstes werden die notwendigen PIB Attribute des Coordinators gesetzt (ShortAddress, die Erlaubnis, dass andere Devices dem Netzwerk beitreten dürfen, dass der Receiver im Idle-Mode schläft).

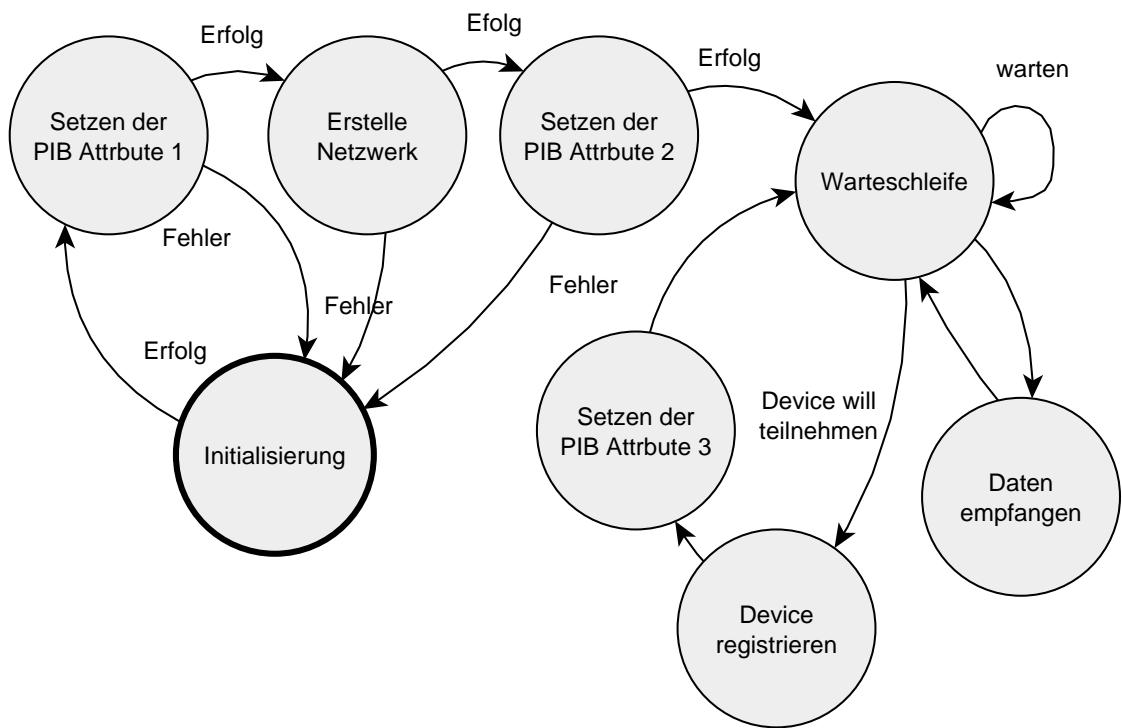


Abbildung 4.5.: Die einzelnen Zustände, die der Coordinator in seinem Betrieb durchläuft

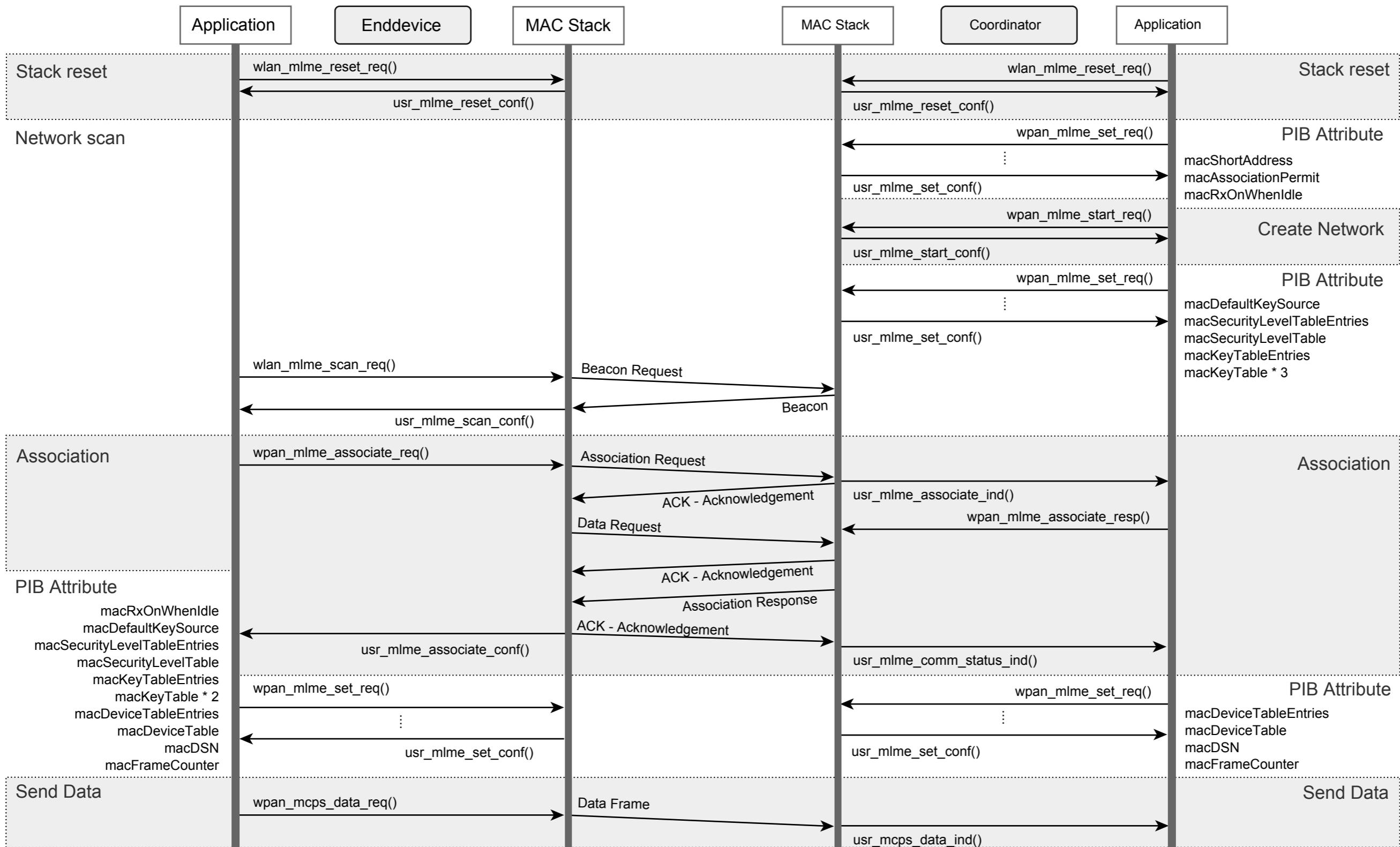


Abbildung 4.6.: Die zeitliche Abfolge der Kommunikation zwischen MAC Stack und der Applikation sowie zwischen End Device und Coordinator

#### *4. Implementierung*

Dann wird das Netzwerk gestartet. Danach werden die nötigen PIB Attribute zum Speichern und Verwalten der Schlüssel und der Konfiguration der Verschlüsselung gesetzt. Als nächstes wartet der Coordinator darauf, dass ein *End Device* dem Netzwerk beitreten möchte.

Ist dies der Fall, wird er mit einer Indication-Nachricht vom MAC Stack darüber informiert. Er vergleicht die *Extended Address* des *End Device* mit denen in der gespeicherten Liste (mit zwei Einträgen) und teilt dem *End Device* je nach Wert eine *Short Address* zu. Danach wird das *End Device* mittels der PIB Attribute in der Datenbank gespeichert, damit der MAC Stack die Schlüssel für selbiges auswählen kann. Wichtig ist an dieser Stelle, dass diese PIB Attribute nicht gesetzt werden können, wenn das *End Device* dem Netzwerk noch nicht beigetreten ist. Der MAC Stack würde diesen Versuch mit einer Fehlermeldung quittieren.

Wenn alle PIB Attribute gespeichert sind, wartet der Coordinator darauf Datenpakete von anderen Enddevices zu empfangen. Dabei kann er anhand der *Short Address* zwischen zwei Enddevices unterscheiden. Das Entschlüsseln dieser Nachricht übernimmt wie gesagt der MAC Stack selbst.

#### End Device

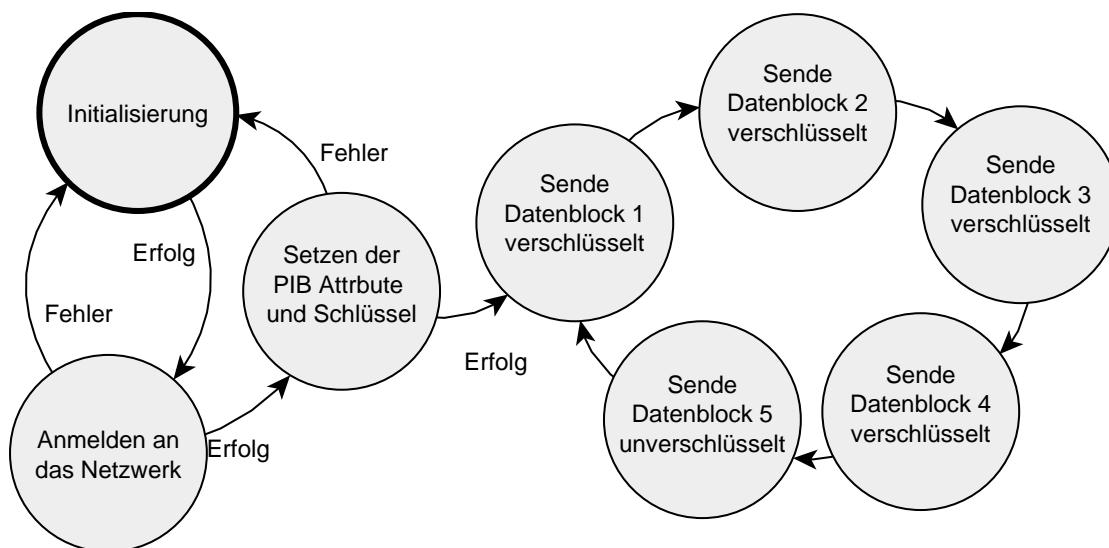


Abbildung 4.7.: Die einzelnen Zustände, die das *End Device* in seinem Betrieb durchläuft

Die Applikation setzt, wie beim Coordinator, alle PIB Attribute und damit den Zustand des Netzwerks zurück, wie in Abbildung 4.7 zu sehen ist. Danach wird nach allen Netzwerken, die das *End Device* erreichen, gescannt und dasjenige ausgewählt, welches den richtigen Coordinator enthält. Daraufhin versucht es diesem Netzwerk beizutreten und erhält eine *Short Address* vom Coordinator zugeteilt. Von da an kann das *End Device*

#### 4. Implementierung

sämtliche für den Betrieb des Netzwerks erforderlichen PIB Attribute in einem Rutsch setzen. Dazu gehören vor allem die Konfiguration der Schlüsselverwaltung. Nach Abschluss dieser Konfiguration beginnt das *End Device* periodisch Datenpakete an den Coordinator zu senden. Die ersten vier von ihnen sind gesichert, jeweils zwei nur mit einer MIC verschiedener Größe und ohne Verschlüsselung und zwei mit einer MIC verschiedener Größe und mit Verschlüsselung. Das fünfte Datenpaket hingegen ist komplett unverschlüsselt und ohne MIC gesichert.

Um das ineinander greifen der einzelnen Funktionen besser zu verstehen, wurde der Ablauf in der Abbildung 4.6 schematisch dargestellt. Im Anhang A sind die gesamten Programm-Ablauf-Pläne abgebildet. Sie zeigen, aus welchem Code die einzelnen Callback-Funktionen bestehen, die in der Applikation implementiert worden sind.

##### 4.1.6. Aufbau der versendeten Daten

Durch das Mitsniffen des Datenverkehrs ließ sich herausfinden, wie die Daten nun im einzelnen verschlüsselt werden, welche Datenblöcke unverschlüsselt bleiben und nur durch die MIC geschützt werden und welche sowohl verschlüsselt als auch mit einer MIC geschützt werden.

In Abbildung 4.8 ist der Aufbau eines Datenframes schematisch dargestellt.

Durch die Verschlüsselung werden zusätzliche Header in das Daten-Frame geschrieben, um die Informationen über die Verschlüsselung verwalten zu können. Dabei handelt es sich um den *Auxillary Security Header*. Dieser beinhaltet alle wichtigen Informationen über die verwendete Verschlüsselung. Dabei definiert das *Security Control Field* mit seinen verschiedenen Bitfeldern, welcher *Security-Level* und welcher *Keymode* gewählt worden ist. Im Falle des *Explicit Key Identification*-Mode kommt noch ein weiteres Feld hinzu, der *Key Index*. Er enthält ganz einfach den Key-Index. Sollte für die zu versendenden Daten eine MIC generiert werden, wird diese an den Payload hinten angehängt und im Falle einer Verschlüsselung mit chiffriert. Schlimmstenfalls kommen so 39Byte hinzu, mindestens (ohne MIC) jedoch 9Byte wegen den zusätzlichen Headern.

#### 4.1. ATTEL MAC Stack 802.15.4

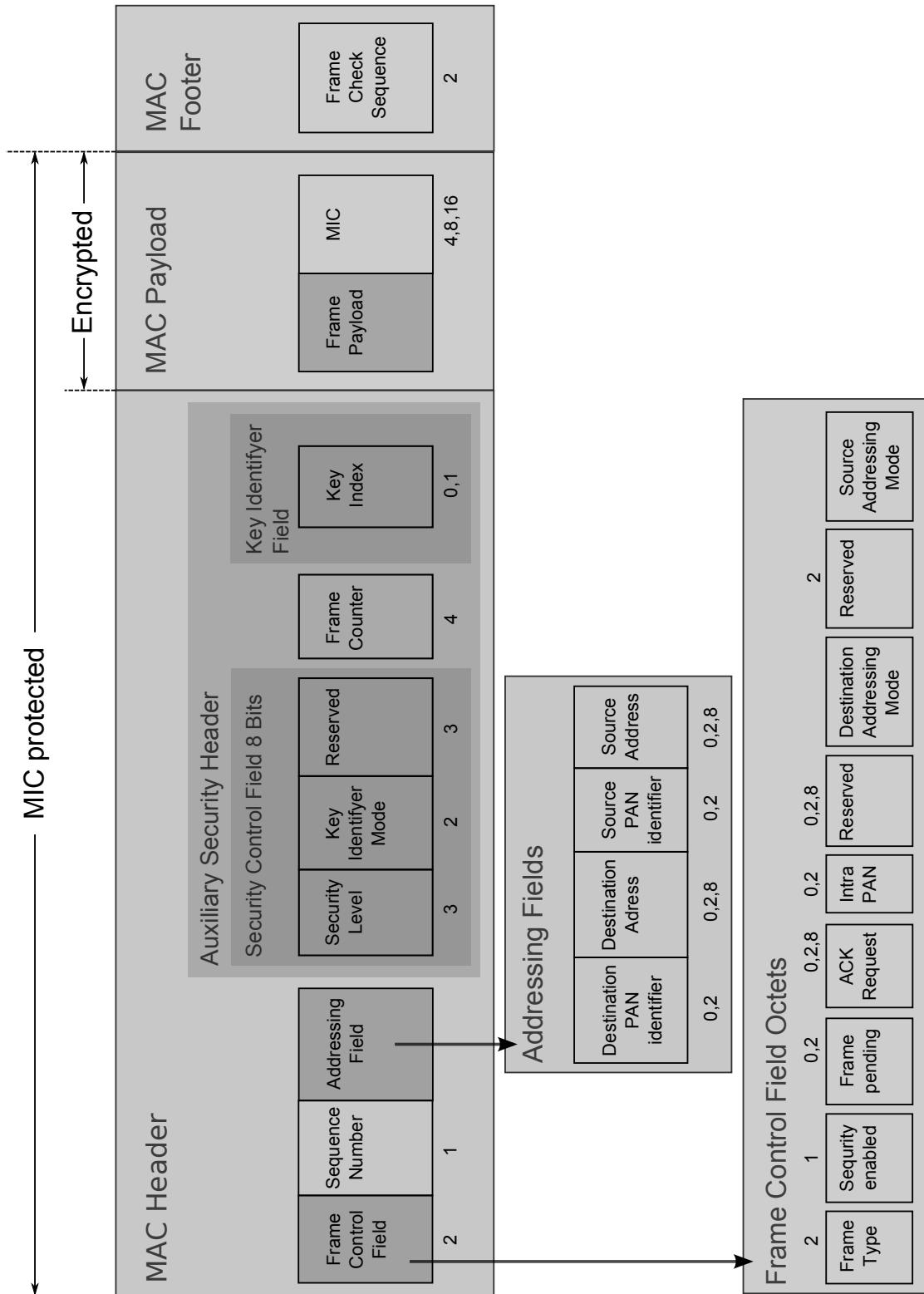


Abbildung 4.8.: Aufbau eines Datenframes, welches verschlüsselt übertragen wird.

## 4. Implementierung

Übrigens ist zu erkennen, dass die *Frame Check Sequence* zur Überprüfung der Richtigkeit der Daten, in den *MAC Footer* gepackt wird, also ganz zum Schluss kommt. Sie ist in jedem Fall sowohl von der Verschlüsselung, als auch von der *MIC-Protection* ausgenommen. Sie kann also manipuliert werden, ohne, dass dies durch die Sicherheitsmechanismen überprüft werden kann. Dies sollte beim Design der Software unbedingt mit berücksichtigt werden.

### 4.1.7. Erstellung des Quellcodes

Die in diesem Kapitel aufgezählten Grundlagen sollen nun der Implementierung als Basis dienen. Während des gesamten Entwicklungsprozesses konnte keine einheitliche Linie verfolgt werden. Eher wurde durch das Schreiben kleiner Programme herausgefunden, wie die Grenzen einzelner Befehle sind und wie bestimmte Konstrukte (wie die Queue) zu verändern sind. Diese Erkenntnisse flossen dann in den endgültigen Code mit ein, so dass es schwierig ist, einen Roten Faden anzugeben.

Auf diese Weise konnte auch der Bug im MAC Stack gefunden werden. Der Code, der ihn verursachte, wurde in einem einfachen Programm ausgeführt, so dass andere Fehlerquellen ausgeschlossen werden konnten.

Als Orientierungshilfe während des Programmierens boten sich die Zustandsdiagramme des Koordinators (Abbildung 4.5) und des Endgerätes (Abbildung 4.7) an. Nach ihrem Vorbild sollte der Code strukturiert werden, so dass beim Beschreiben des Quelltextes auf dieses Schema immer wieder zurückgekommen werden kann.

Um die in Kapitel 4.1.4 aufgezählten Möglichkeiten der Sicherung umzusetzen, fiel die Wahl auf die Sicherheitsfunktionen des MAC Stack. Diese bieten die größte Kompatibilität. Die Möglichkeit auf die Schichten *Security Tool Box* und *Security Abstraction Layer* direkt zurückzugreifen, fiel heraus, da in diesem Fall stets der Zustand der Hardware hätte überprüft und gesteuert werden und die Generierung der Nonce (im Falle des STB) und des IVs (im Falle der SAL) von der Applikation selbst übernommen werden müssen. Der MAC Stack selber hat Routinen zur Konfiguration der Hardware und kümmert sich um diese (das Aufwachen aus dem Sleep-Mode, sichern und neu setzen der Register usw.). Außerdem hätte die Applikation sich mit dem MAC Stack synchronisieren und auf dessen Aktionen Rücksicht nehmen müssen, um diesem nicht in die Quere zu kommen.

## Schlüssel- Infrastruktur

Nachdem, was in Kapitel 4.1.4 bereits beschrieben wurden ist, unterteilt der MAC Stack die verwendeten Schlüssel nicht in *Network Key* und *Link Key*. Bis auf die grundlegenden Funktionen, die Schlüssel zu verwalten und automatisch zur Ver- und Entschlüsselung auszuwählen, wird es dem Entwickler überlassen, eine solche Klassifizierung vorzunehmen (falls dies gewünscht ist). Ebenso bietet der MAC Stack keine Funktionen um die Schlüssel während des Betriebes zu erzeugen und diese sicher auszutauschen. Auch dies muss vom Programmierer selber implementiert werden.

Während der Entwicklung des Quellcodes war es nicht möglich den Koordinator und das Device unabhängig zu programmieren. Für beide wurde der Quellcode parallel entwickelt, da darauf geachtet werden musste, dass die Programme richtig aufeinander reagieren und miteinander interagieren. Als Basis wurden die Dateien der Beispieldapplikation aus dem Verzeichnis `/Atmel/MAC_v_2_7_1/Applications/STB_Examples/Secure_Star_Network` genommen. Die Unterverzeichnisse Coordinator und Device wurden mit sämtlichen darin befindlichen Ordner an eine andere Stelle kopiert um mit ihnen arbeiten und alle darin befindlichen Dateien verändern zu können. Als Basisplattform wurde `ATMEGA128RFA1_RCB_6_3_SENS_TERM_BOARD` gewählt, welches in beiden Ordnern vorhanden ist. In ihm befinden sich die benötigten Projektdateien für ATMEL Studio und in einem weiteren Unterverzeichnis namens `GCC` die nötigen Makefiles zum Kompilieren. Der Quelltext dagegen ist unabhängig von der Plattform im Ordner `Src`. Die benötigten Header-Dateien befinden sich im Ordner `Inc`.

Sowohl für den Koordinator als auch für das Device existiert also ein Satz von Dateien bestehend aus dem Makefile `Makefile`, dem Quelltext `main.c` und der Header-Datei `app_config.h`

Als erster Schritt wurde das Makefile `Makefile_Debug` dahingehend modifiziert, dass die benutzte Hardware unterstützt wird. Die einzelnen Schritte dazu wurden bereits im Kapitel 4.1.2 beschrieben. Das Makefile namens `Makefile_Debug` für den Koordinator und das Endgerät unterscheiden sich in einem Parameter: `-DFFD` für den Koordinator und `-DRFD` für das Device. Sie legen den benötigten Funktionsumfang fest, der vom ATMEL MAC Stack unterstützt und einkompliert werden soll. Danach wurde der Quelltext erstellt. Er befindet sich in einer einzelnen Datei `main.c`, die es einmal für das Device und einmal für den Koordinator gibt.

## Koordinator

In Abbildung 4.5 sind die einzelnen Zustände aufgeführt, die das Programm durchläuft. Wie diese programmiert worden sind, soll nun erklärt werden.

Die `main()`-Funktion, wie sie für jeden C-Quelltext obligatorisch ist, beinhaltet sämtliche Initialisierungsroutinen. Vor allem wird über den Befehl `tal_pib_ieeeAddress = GCoordExtAddr` die Extended Address festgelegt.

Als nächstes wird der MAC Stack mittels der Funktion `wpan_mlme_reset_req()` zurückgesetzt um den Koordinator konfigurieren zu können. Danach wird über ein Schleife, die sich endlos wiederholt, immer wieder die Funktion `wpan_task()` ausgeführt. Sie arbeitet die in Kapitel 4.1.3 Programmierung des MAC Stack beschrieben Queue ab.

Die vorher aufgerufene Funktion `wpan_mlme_reset_req()` gehört zu den Request-Funktionen und hat damit als Antwort eine Confirm-Funktion. Dabei handelt es sich um eine Callback-Funktion, wie es in Abbildung 4.1 dargestellt ist. Diese muss also von der Applikation implementiert werden. In

## Initialisierung

## Setzen der PIB-Attribute

1

## 4. Implementierung

diesem Fall ist es die Funktion `usr_mlme_reset_conf()`. Ihr wird die Variable `Status` übergeben um überprüfen zu können, ob die Anfrage erfolgreich war oder nicht. Die Callback-Funktion ruft nun wiederum die Funktion `wpan_mlme_set_req(macShortAddress, ...)` auf. Mit ihr wird die *Short Address* des Koordinators festgelegt. Dabei handelt es sich um das erste *PIB-Attribut*, das in diesem Quelltext gesetzt wird, wie es bereits in Kapitel 4.1.3 erklärt wurde. Bei den erwähnten Funktionen handelt es sich um die vom MAC Stack angebotenen *SAPs*. Die PIB-Attribute werden generell so gesetzt, dass der Funktion `wpan_mlme_set_req()` die Adresse zu einer Variablen übergeben wird, die die zu setzenden Attribute enthält. Dabei kann es sich um elementare Variablen handeln, als auch um komplexere Strukturen. Dies ist abhängig vom Attribut.

Im gesamten Quelltext handelt sich das Programm nun von einer Request-Funktion über die daraufhin zurückgegebene Confirm-Funktion zur nächsten Request-Funktion.

Um das Setzen der einzelnen PIB-Attribute kümmert sich die Callback-Funktion `usr_mlme_set_conf()`. Sie wird bei jedem erneuten setzen eines PIB-Attributs vom MAC Stack aufgerufen und das vorher gesetzte PIB-Attribut als Argument übergeben, so dass sie über eine `switch`-Anweisung einzeln entscheiden kann, wie auf welches PIB-Attribute zu reagieren ist. Mit ihr werden sämtliche PIB-Werte gesetzt.

Damit dem Netzwerk auch andere Geräte beitreten dürfen, muss das Attribut `macAssociationPermit` gesetzt werden. Damit die Energiemessung unter möglichst realistischen Bedingungen abläuft, soll der Koordinator und auch das Device in den Stromspar-Modus schalten und den *Receiver* abschalten können, sobald keine Daten gesendet werden. Dazu muss noch das Attribut `macRxOnWhenIdle` gesetzt werden.

### Erstelle Netzwerk

Als nächstes muss das Netzwerk vom Koordinator aufgebaut werden. Dafür wird vom ATMEL MAC Stack die Funktion `wpan_mlme_start_req()` angeboten. Sie wird als Reaktion auf das Setzen des PIB-Attributes `macRxOnWhenIdle` in der Funktion `usr_mlme_set_conf()` aufgerufen. Dieses Verhalten wird im Programm-Ablauf-Plan in den Abbildungen C.14, C.15, C.16 und C.17 abgebildet. Daraufhin wird wieder die passende Confirm-Funktion `usr_mlme_start_conf()` aufgerufen.

### Setzen der PIB Attribute 2

Nach diesem Schritt werden weitere PIB-Attribute gesetzt. Damit wechselt der Koordinator in den Zustand „Setze PIB Attribute 2“ und dementsprechend wieder in die Funktion `usr_mlme_set_conf()`. An dieser Stelle nun werden dem MAC Stack die ersten wichtigen Informationen übergeben, um die Schlüsselverwaltung zu konfigurieren. Dabei geht es um die PIB-Attribute

- `macDefaultKeySource`
- `macSecurityLevelTableEntries`
- `macSecurityLevelTable`
- `macKeyTableEntries`

- macKeyTable

Ihre Rolle wurde bereits in Kapitel 4.1.4 besprochen.

Hier beginnt der Ablauf des Programms sich von dem anderer Programme ohne Verschlüsselung zu unterscheiden. Dabei geht es nur um das Setzen der oben aufgelisteten PIB-Attribute.

Wenn danach ein Gerät dem Netzwerk beitreten möchte, läuft die Anmeldung genauso ab, wie bei einem Netzwerk ohne Verschlüsselung. Durch die Sicherungsmechanismen entsteht beim Anmeldeprozess kein zusätzliche Datenverkehr. Dies ist in Abbildung 4.6 schematisch dargestellt.

Da der hier konfigurierte Schlüssel die Aufgabe des Network Key übernehmen soll, kann er unabhängig von einem dem Netzwerk beitretenden Knoten konfiguriert werden. Da dieser Schlüssel für sämtliche Geräte im Netzwerk gelten soll, fiel die Wahl auf den Mode 0x01 - Explicit Key Identification. Dieser Mechanismus ist durch den *Key Index* bei der Schlüsselauswahl recht flexibel und kann somit für andere Szenarien (wie die Unterteilung in Unternetzwerke und darauf angewandte spezialisierte Schlüssel) angepasst werden.

Die beiden Link Keys werden ebenfalls in diesem Zustand im MAC Stack installiert. Da Link Keys für jedes Kommunikationspaar einzigartig sind, fiel die Wahl für die Schlüssel auf den Mode 0x00 - Implicit Key Identification, der die Schlüssel je nach Netzwerknoten automatisch auswählt. Um jedem Netzwerknoten einen Schlüssel zuzuweisen ist dieser Modus ideal. Falls allerdings jedes Kommunikationspaar (wie bei einem Link Key üblich) einen eigenen Schlüssel haben soll, dieser also nicht vom Device, sondern von dessen Paarung mit einem anderen Knoten abhängig sein soll, dann ist der Mode 0x01 besser geeignet, da der Mode 0x00 den Schlüssel nur nach der Short Address und der PAN-ID eines Knoten auswählt und den zweiten Knoten nicht berücksichtigt.

An dieser Stelle schaltet der Koordinator in den Zustand „Warteschleife“. Von nun an wartet er darauf, dass ein Gerät dem Netzwerk beitreten möchte.

### Warteschleife

Sollte dies geschehen, wird ihm das über den Aufruf der Callback-Funktion `usr_mlme_associate_ind()` kenntlich gemacht. Daraufhin überprüft er die erhaltene Extended Address mit den gespeicherten Adressen und weist dem Gerät in Abhängigkeit dieser Adresse eine *Short Address* mit Hilfe der Funktion `wpan_mlme_associate_resp()` zu. Die Callback-Funktion `usr_mlme_comm_status_ind()` informiert nun über Erfolg oder Misserfolg des Beitritts. Daraufhin setzt der Koordinator in Abhängig vom beigetretenen Gerät die PIB-Attribute

- macDeviceTableEntries
- macDeviceTable
- macDSN
- macFrameCounter

#### 4. Implementierung

##### Setzen der PIB

###### Attribute 3

Ihre Aufgabe wurde ebenfalls im Kapitel 4.1.4 besprochen. Leider musste das Konfigurieren dieser Attribute an diese Stelle des Quellcodes verschoben werden, da sie nur gesetzt werden können, wenn ein Gerät dem Netzwerk bereits beigetreten ist, welches durch diese Attribute beschrieben wird.

Dieser Zustand ist für die Sicherheitsschicht von großer Bedeutung und in einem Netzwerk ohne Sicherheitsmechanismen überflüssig.

##### Warteschleife

Das Empfangen der Daten, was daraufhin folgt, läuft wiederum genauso ab, wie in einem ungesicherten Netzwerk. Nur die Parameterliste der Funktion `usr_mcps_data_ind()` wird in diesem Fall erweitert. Der Ablauf ist jedoch der selbe.

Damit sind die Differenzen des Quelltextes eines Koordinators mit und ohne Verschlüsselung verhältnismäßig klein. Bis auf die Zustände „Setzen der PIB-Attribute 2“ und „Setzen der PIB-Attribute 3“, sowie die Parameterliste der Funktion `usr_mcps_data_ind()` zum Empfangen der Daten, ist der Ablauf der Kommunikation und das Senden der Daten der gleiche. Dabei handelt es sich um die Parameter `SecurityLevel`, `KeyIdMode` und `KeyIndex`. Ihre Bedeutung ist ebenfalls im Kapitel 4.1.4 beschrieben. Besonders nach außen hin scheint es im Netzwerk keinen Unterschied zu geben.

Größtenteils bestand das Erstellen des Quelltextes aus vielen Try-and-Error-Versuchen. Dabei ist z.B. erst herausgekommen, dass der Queue nicht mehrere unterschiedliche Nachrichten übergeben werden können, da dieser Versuch mit einer Fehlermeldung vom MAC Stack quittiert wird. Nach jedem Aufruf einer Request-Funktion muss zuerst auf dessen Confirm-Funktion gewartet werden. Auch das Zusammenspiel, wann welche Attribute gesetzt werden können, musste durch viele Versuche erst herausgefunden werden.

Zwischenzeitlich kam es zu dem Problem, dass der MAC Stack während der Kommunikation zwischen Koordinator und End-Gerät einen Fehler ausgab, sobald mehrere Schlüssel verwaltet werden mussten. Sobald jedoch wieder nur auf einen einzigen Schlüssel zurückgegriffen wurde, verschwand das Problem. Während des Debuggens kam heraus, dass die Nachricht nicht entschlüsselt werden konnte. Der Fehler musste also irgendwo in der Funktion `stb_ccm_secure()`. Diese gab beim Versuch, die Nachricht zu entschlüsseln, den Fehler `STB_CCM_MICERR` zurück. Damit lag das Problem also beim Berechnen der MIC. Sobald jedoch die richtige MIC in den zu vergleichenden Speicherbereich geschrieben wurde, verschwand der Fehler und die Nachricht konnte Problemlos entschlüsselt werden. Die falsche Wahl des Schlüssels zur Entschlüsselung konnte ausgeschlossen werden, da sich der richtige Schlüssel im Speicher befand und auf diesen auch vom MAC Stack zurückgegriffen wurde. Die Falsche MIC unterschied sich von der richtigen MIC nur in den Bytes, an denen die Nonce zu stehen hatte. An deren Stelle jedoch befanden sich in allen Bytes nur der Wert 0x00. Somit war klar, dass der Fehler in der Berechnung der Nonce lag. Laut Abbildung 4.1.3 wird die Nonce vom MAC Stack selbst berechnet, da ihr Aufbau nicht Teil der IEEE 802.15.4 Spezifikation ist. Damit konnte das Problem auf den Code in der Datei `mac_security.c` eingegrenzt werden. Hier stellte sich heraus, dass eine Codezeile einen falschen Index benutzte, in der eigentlich die *Extended Address* des Senders aus der PIB-Datenbank gelesen werden sollte. Diese stand an der richtigen Spei-

cherstelle, auf sie wurde jedoch nicht zugegriffen. Nachdem der Code abgeändert wurde, verschwanden die Fehlermeldungen und die verschlüsselte Kommunikation mit mehreren Schlüsseln lief tadellos.

Zusammenfassend lässt sich sagen, dass sich der Ablauf zwischen einem Koordinator mit und ohne Sicherheitsmechanismen in den folgenden Punkten unterscheidet:

- als erstes müssen die PIB-Attribute `macDefaultKeySource`, `macSecurityLevelTableEntries`, `macSecurityLevelTable`, `macKeyTableEntries` und `macKeyTable` gesetzt werden, bevor sich ein Endgerät anmeldet
- danach müssen nach dem Anmelden eines Endgerätes die PIB-Attribute `macDeviceTableEntries`, `macDeviceTable`, `macDSN` und `macFrameCounter` gesetzt werden, da sie dieses Endgerät beschreiben
- der letzte Unterschied betrifft die Funktion `usr_mcps_data_ind()`: sie erhält die neuen Parameter `SecurityLevel`, `KeyIdMode` und `KeyIndex`, um die Konfiguration der Verschlüsselung zu erfahren

## Endgerät

Abbildung 4.7 zeigt die einzelnen Zustände, die das Endgerät durchläuft.

Wie beim Koordinator beginnt das Programm mit der Funktion `main()`. Ebenso werden die nötigen Initialisierungen vorgenommen um den MAC Stack betriebsbereit zu machen. Auch hier bildet die Funktion `wpan_mlme_reset_req()` den Einsprungpunkt in die Kette aus *Request-* und *Confirm-Funktionen*, über die die einzelnen Zustände abgearbeitet werden.

## Initialisierung

Als nächstes sucht das Gerät in der Umgebung nach Netzwerken mit Hilfe der Funktion `wlan_mlme_scan_req()`, an die es sich anmelden kann. Über ein *Beacon-Request-Frame* fordert dieses alle Koordinatoren auf, mit einem *Beacon-Frame* ihre Anwesenheit zu signalisieren. Sobald ein Netzwerk gefunden ist, wird dies mit dem Aufruf der Callback-Funktion `usr_mlme_scan_conf()` gemeldet. Endet der Versuch mit einem Erfolg, tritt das Endgerät mit dem Aufruf der Funktion `wpan_mlme_associate_req()` dem Netzwerk bei und wechselt zum Zustand „Setzen der PIB Attribute und Schlüssel“. Dieser Ablauf ist in der Abbildung 4.6 zu sehen.

## Anmelden an das Netzwerk

Als nächstes werden über die Funktion `wpan_mlme_set_req()` die benötigten PIB-Attribute gesetzt. Dabei handelt es sich um folgende Attribute:

## Setzen der PIB Attribute und Schlüssel

- `macRxOnWhenIdle`
- `macDefaultKeySource`
- `macSecurityLevelTableEntries`
- `macSecurityLevelTable`

#### 4. Implementierung

- macKeyTableEntries
- macKeyTable
- macDeviceTableEntries
- macDeviceTable
- macDSN
- macFrameCounter

Die Attribute, die hier für die Sicherheitsarchitektur ausschlaggebend sind, heißen:

- macDefaultKeySource
- macSecurityLevelTableEntries
- macSecurityLevelTable
- macKeyTableEntries
- macKeyTable
- macDeviceTableEntries
- macDeviceTable

Ihre Funktion wurden in Kapitel 4.1.4 detailliert diskutiert.

Das Setzen dieser für die Sicherheit wichtige Attribute unterscheidet ein Endgerät mit Verschlüsselung von einem ohne Verschlüsselung.

Nachdem nun die Schlüssel eingerichtet und die Sicherheitsschicht konfiguriert worden ist, kann der Datenaustausch beginnen. Aus der Funktion `wpan_mlme_set_req()` heraus wird ein Zeitgeber eingerichtet, der nach einer gewissen Zeit eine von mehreren Funktionen zum Versenden der Daten aufruft. Von diesen Funktionen gibt es fünf an der Zahl. Diese werden zyklisch aufgerufen, wie es in Abbildung 4.7 zu sehen ist. Jede Funktion beinhaltet verschiedene *Securitylevels*, die umgesetzt werden, und ruft wiederum einen Zeitgeber auf, der die nächste Funktion in der Schlange ausführt. Dabei handelt es sich um die Funktionen

- `app_send_data_crypt_lvl1_km0()` mit dem Securitylevel=1 und dem KeyMode=0
- `app_send_data_crypt_lvl7_km0()` mit dem Securitylevel=7 und dem KeyMode=0
- `app_send_data_crypt_lvl6_km1()` mit dem Securitylevel=6 und dem KeyMode=1
- `app_send_data_crypt_lvl3_km1()` mit dem Securitylevel=3 und dem KeyMode=1
- `app_send_data()` ohne Verschlüsselung

Die Liste auf Seite 66 in Kapitel 4.1.8 schlüsselt die angewandten Sicherungstechniken je nach *Securitylevel* und *Keymode* auf. Dies soll veranschaulichen, welche Möglichkeiten der ATTEL MAC Stack bietet und wie diese umgesetzt werden können. Die Einzelheiten, wie die Techniken funktionieren, wurden bereits in Kapitel 4.1.4 besprochen.

Der große Unterschied in diesem Codebereich betrifft die Parameterliste der Funktion `wpan_mlme_data_req()`. Im Falle der eingeschalteten Sicherungsmechanismen kommen noch drei Parameter hinzu: Der *Securitylevel*, der *KeyIdMode* und der *KeyIndex*. Damit kann während des Betriebs die verschlüsselte Datenübertragung variabel gestaltet werden.

**Securitylevel** Der Securitylevel legt die Einzelheiten zur Sicherung des Payloads fest (Siehe dazu Kapitel 4.1.4).

**KeyIdMode** Dieser Parameter bestimmte in welcher Art und Weise der Schlüssel zur Verschlüsselung ausgewählt werden soll (Siehe dazu Kapitel 4.1.4)

**KeyIndex** Sollte der Schlüssel mittels Mode 0x01 ausgewählt werden, muss dazu noch mithilfe eines Keyindex der Schlüssel aus einem Bund an Schlüsseln ausgewählt werden (Siehe dazu Kapitel 4.1.4). Sollte der Schlüssel mittels Mode 0x01 ausgewählt werden, muss dazu noch mithilfe eines Keyindex ein Schlüssel aus einem Bund an Schlüsseln ausgewählt werden (Siehe dazu Kapitel 4.1.4).

Ziel der periodischen Datenübertragung ist es, sämtliche Modi zu präsentieren und beim Sniffen miteinander vergleichen zu können. Die Anwendung mehrerer Modi und Securitylevels hat ebenfalls dabei geholfen, den bereits besprochenen Bug zu finden.

Zusammenfassend lässt sich sagen, dass die folgenden Schritte ein Endgerät mit unterstützter Sicherheitsfunktionen von einem ohne Sicherheitsfunktionen unterscheidet:

- als erstes müssen die PIB-Attribute `macDefaultKeySource`, `macSecurityLevelTableEntries`, `macSecurityLevelTable`, `macKeyTableEntries`, `macKeyTable`, `macDeviceTableEntries` und `macDeviceTable` gesetzt werden
- ansonsten liegt der letzte Unterschied in der Funktion `wpan_mcps_data_req()`: sie enthält die neuen Parameter `SecurityLevel`, `KeyIdMode` und `KeyIndex`, um die Konfiguration der Verschlüsselung vorzugeben

#### 4.1.8. Anpassen des MAC Quelltext für die Energiemessung

Da von großem Interesse ist, wie sich der Energieverbrauch bei angeschalteter Verschlüsselung ändert, ist ein Messaufbau von Nöten, mit dem es möglich ist, mit Hilfe eines Oszilloskops die Spannung über einen Messwiderstand darzustellen und daraus den Strom zu berechnen, welcher schlussendlich auf den Energieverbrauch schließen lässt. Dieser Aufbau ist in Abbildung 4.9 schematisch dargestellt. Der Messwiderstand muss daher

#### Energieverbrauch

#### 4. Implementierung

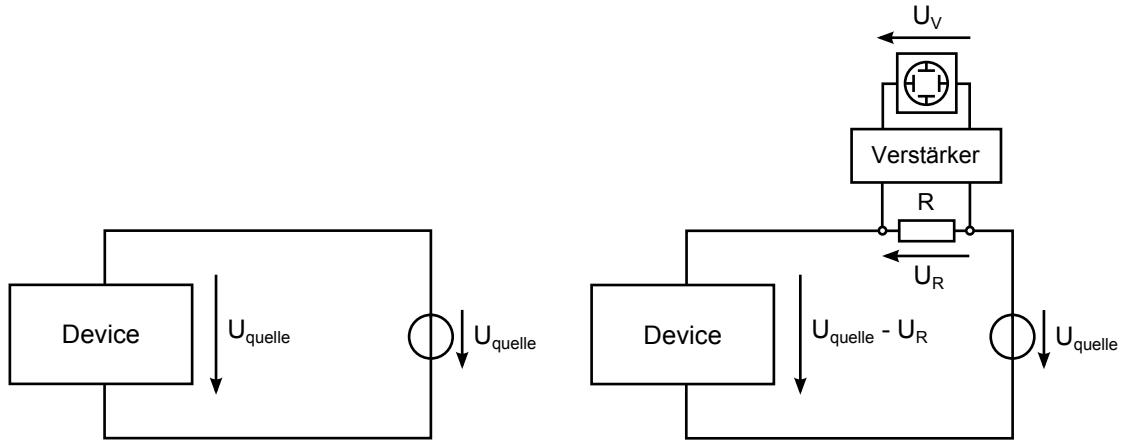


Abbildung 4.9.: Aufbau der Schaltung zur Messung des Energieverbrauchs

möglichst klein gewählt werden, damit der Einfluss auf das Gerät oder die Schaltung vernachlässigbar bleibt und zu keinen verfälschten Ergebnissen führt. Dies bedeutet, dass folgendes gelten muss:  $U_R \ll U_q$ . Das Device zieht eine bestimmte Menge Strom aus der Spannungsquelle. Dieser Strom passiert den Messwiderstand  $R$  und verursacht über ihn einen Spannungsabfall  $U_R$ . Dieser Spannungsabfall muss so klein sein, dass das Device noch annähernd mit der Spannung  $U_q = 3,3V$  versorgt wird. Wird  $R$  zu groß gewählt, steht dem Device zu wenig Spannung zur Verfügung, da der Widerstand einen zu großen Spannungsabfall verursacht. Ist der Wert jedoch zu gering gewählt, ist der vom Oszilloskop gemessene Spannungsabfall zu gering und verschwindet im Rauschen. Selbst wenn die Auflösung des Oszilloskops groß genug ist, die Messung findet in einem normalen Raum ohne besondere Vorkehrungen der Elektromagnetischen Abschirmung statt und damit haben in diesem Messwertebereich selbst Berührungen mit der Hand oder die elektrostatische Aufladung der Luft Einfluss auf den zu messenden Spannungswert. Die Ergebnisse können also nur ein grober Richtwert zur Orientierung sein, dürfen aber auf keinen Fall als unverfälschter Messwert interpretiert werden.

Um also einen Widerstand wählen zu können, der klein genug ist, wird ein Verstärker an den Messwiderstand geschaltet, um die geringe Spannung zu verstärken. Im Versuch wurde eine Verstärkung von 1000 gewählt. Die Größe des Widerstands wurde durch Erfahrung und Probieren letztendlich auf 0,1 Ohm festgesetzt. Damit fällt über ihn bei einer maximalen Stromstärke  $I = 60mA$  eine Spannung von  $U_R = 6mV = 0.006V$  ab. Das maximale Verhältnis aus der Messspannung zur Spannungsversorgung beträgt  $\frac{U_R}{U_q} = 1,8\%$  und damit ist der Wert klein genug um ihn zu vernachlässigen.

Es ist davon auszugehen, dass das Device minimal ca.  $I = 5mA$  während der Messung verbraucht, da der Prozessor zumindest in diesem Bereich stets am Rechnen ist. Damit ergibt sich eine minimale Spannung über den Messwiderstand von  $U_R = 0,5mV$ . Diese wird durch den Verstärker um den Faktor 1000 verstärkt und das ergibt die Spannung  $U_V = 1000 \cdot U_R = 500mV = 0,5V$ . Die kleinste zu messende Spannung über dem Mess-

widerstand von  $0,5mV$  ist für das Oszilloskop zu gering und würde - wie bereits gesagt - im Rauschen untergehen. Dieses Rauschen wird nicht bedingt durch die Bauteile auf dem Netzwerk Device, sondern durch die Bauteile im Oszilloskop. Durch den Verstärker wird diese untere Spannungswert in einen Bereich von  $0,5V < U_V < 6V$  verschoben, in dem das Rauschen der Baugruppen im Oszilloskop nicht mehr ins Gewicht fällt.

Innerhalb der Verstärkerschaltung musste noch ein regelbarer Widerstand an die Lötjumper gesetzt werden. Da bei der Verstärkerschaltung ein Offset von  $700mV$  gemessen wurde, musste dieser erst auf  $0mV$  geregelt werden. Um die Schaltung zu eichen, dient der regelbare Widerstand, der die Masse soweit verschiebt, bis die Ausgangsspannung ohne Eingangsspannung  $0mV$  beträgt.

Damit wurde eine Schaltung konstruiert, die nach „unten hin“ nicht zu ungenau beim Messen wird, aber minimalen Einfluss auf das Netzwerkgerät hat. Diese ist in Abbildung 4.10 dargestellt.

Es muss jedoch beachtet werden, dass Einflüsse und Störungen von der Umgebung mit verstärkt werden und das Ergebnis verfälschen können.

Um nun die einzelnen Bereiche beim Messen kennzeichnen zu können, wird für das Oszilloskop ein Trigger-Signal benötigt. Da in dieser Messung das Modell HP infinium mit vier Messspitzen verwendet wurde, können drei von ihnen genutzt werden, um Trigger-Signale abzugreifen. Dazu wurde in der Software an bestimmten Stellen Befehle eingefügt, die spezielle Pins am Gehäuses, welches den Transceiver und den ATMega128rfa1 beinhaltet, auf einen High- oder Lowpegel setzen können. Es wurden drei Pins ausgewählt, die mit keiner anderen Peripherie verbunden sind und damit frei zur Verfügung stehen, ohne das die Funktionen des Devices beeinträchtigt werden. Dabei handelt es sich um die Pins 16, 23 und 36. Diese sind direkt mit dem Chip verbunden und können über die AVR-LIBC mit den Bezeichnungen PORTD, PORTB und PORTF angesprochen werden. Dabei handelt es sich um ganze Register, die gleich mehrere Pins beinhalten. Um nun die einzelnen Pins in den Registern auszuwählen, können die einzelnen Bit mit den Konstan-

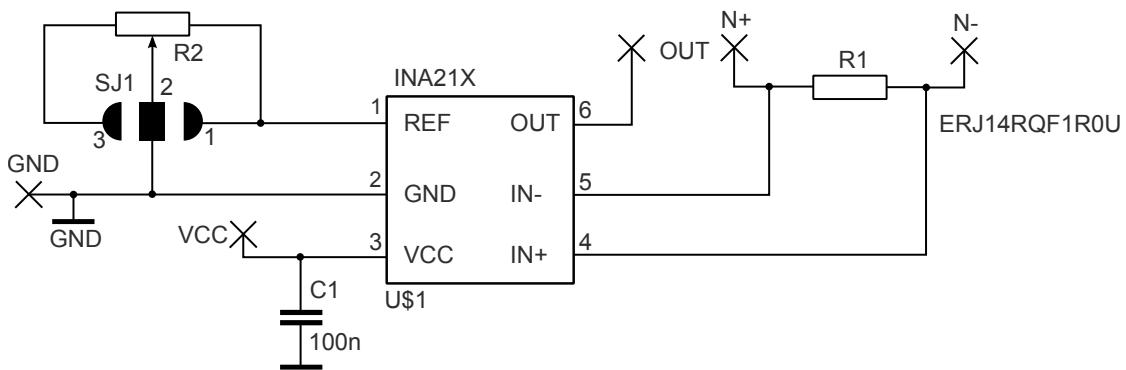


Abbildung 4.10.: Die Schaltung des Verstärkers, mit dem die Spannung über dem Messwiderstand für das Oszilloskop verstärkt wird.

#### 4. Implementierung

ten PD6, PB6 und PF3 manipuliert werden. Dazu muss eine Bitmaske erstellt werden, die dann in die Register geschrieben wird. Damit die Pins jedoch als Datenausgang genutzt werden können, müssen sie in speziellen Registern als solche gesetzt werden.

Die Stellen im Code, die mit Hilfe der Trigger-Signale kenntlich gemacht werden sollten, sind einmal:

- der Anfang der Funktion `build_data_frame()`, in der das Daten-Frame erstellt wird und folglich - abhängig von der Konfiguration - die Verschlüsselungs Routinen nur hier aufgerufen werden,
- der Anfang der Funktion `tx_done_handling()`, die aufgerufen wird, sobald das Datenpaket den Transceiver verlassen hat und dieser somit fertig mit Senden ist
- der Anfang und das Ende der Funktion `stb_ccm_secure()`, die dem Layer STB angehört und damit den Standard *CCM/CCM\** implementiert
- innerhalb der Funktion `stb_ccm_secure()` vor dem Beginn der Berechnung der MIC und nach dem Ende der Verschlüsselung im Layer SAL

Die Abbildung 4.11 zeigt, an welchen Stellen die Signale abgegriffen werden.

Damit alle Möglichkeiten des MAC Stacks ausgenutzt werden, sollen fünf Datenpakete zyklisch versendet werden.

- Das erste Datenpaket wird nicht verschlüsselt, sondern nur mit einer 32-Bit MIC geschützt. Der Schlüssel wird über eine globale ID ausgewählt. Das bedeutet, der Securitylevel beträgt 1 und der Keymode hat den Wert 0. Dieser spielt allerdings ohne Verschlüsselung keine Rolle.
- Das zweite Datenpaket wird verschlüsselt und mit einer 128-Bit MIC geschützt. Der Schlüssel wird über eine globale ID ausgewählt. Das bedeutet, der Securitylevel beträgt 7 und der Keymode hat den Wert 0.
- Das dritte Datenpaket wird verschlüsselt und mit einer 64-Bit MIC geschützt. Der Schlüssel wird über die *Short Address* ausgewählt. Das bedeutet, der Securitylevel beträgt 6 und der Keymode hat den Wert 1.
- Das vierte Datenpaket wird nicht verschlüsselt und mit einer 128-Bit MIC geschützt. Der Schlüssel wird über die *Short Address* ausgewählt. Das bedeutet, der Securitylevel beträgt 3 und der Keymode hat den Wert 0. Dieser spielt allerdings ohne Verschlüsselung keine Rolle.
- Das 5. Datenpaket wird weder verschlüsselt, noch mit einer MIC geschützt.

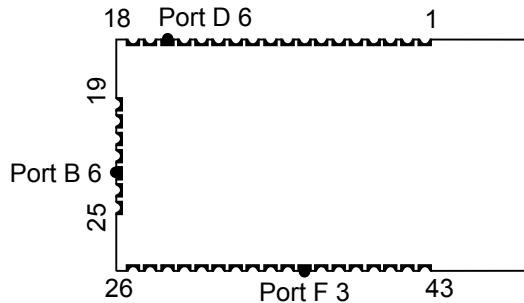


Abbildung 4.11.: Die Belegung der einzelnen Pins, ihre Bezeichnung im Quellcode und ihr Zuordnung zu den jeweiligen Triggersignalen.

## 4.2. ZigBee

Um die Entwicklungszeit während der Studienarbeit möglichst kurz zu halten und die Arbeit auf die zentralen Aspekte zu konzentrieren, wurde als Grundlage die Software *WSNDemo* gewählt und dahingehend modifiziert, dass die verschiedenen Möglichkeiten der Sicherung des Netzwerkes angewandt und überprüft werden können. Dazu wurden dieselben Geräte verwendet, wie bei der Überprüfung des MAC Stack IEEE 802.15.4.

Der Quellcode bot die Möglichkeiten über Makros des Präprozessors das gesamte Programm zu modifizieren und an die gegebenen Bedingungen anzupassen. Die große Menge an Quellcode jedoch gestaltete die Analyse der Software recht schwierig und damit zu zeitaufwendig. Deswegen wurden nur die Bereiche des Codes analysiert, die für die Implementierung der Sicherheit des Netzwerkes wichtig waren. So wurde ein Netzwerk bestehend aus einem *Coordinator*, der gleichzeitig auch die Rolle des *Trustcenters* übernahm, und zwei *Enddevices* geschaffen. Nach kurzer Einarbeitungszeit war das Netzwerk funktionsfähig. Der Quellcode von BitCloud erlaubte eine recht schnelle Modifizierung und Anpassung des Codes auf die geforderten Bedingungen, ohne dabei tief greifend in die Software-Architektur einzutragen und sich dabei mit zeitaufwendigen Details auseinander zu setzen.

Die verschiedenen Möglichkeiten die Sicherheit im Netzwerk zu gestalten, gliedern sich dabei in folgende drei Kategorien:

**kleiner Test**

**Standard security** Mit Hilfe eines einzigen symmetrischen Schlüssels, dem *Network Key*, wird der gesamte Datenverkehr auf der *Network Layer - NWK* genannten Ebene verschlüsselt. Dieser kann entweder auf einem *End Device* vorinstalliert sein oder aber durch einen Teilnehmer vom Coordinator angefordert werden. Die Verschlüsselung findet dabei auf dem sogenannten *Network Layer - NWK* statt. Im ersten Fall sind auf beiden Stellen die Schlüssel bekannt und der Datenverkehr kann nach der Anmeldung direkt gestartet werden. Im zweiten Fall wird der Schlüssel vom Coordinator verschlüsselt zum Knoten gesandt, der dem entsprechenden *End Device* übergeordnet ist (das kann ein Router sein oder im einfachsten Fall der Coordi-

#### 4. Implementierung

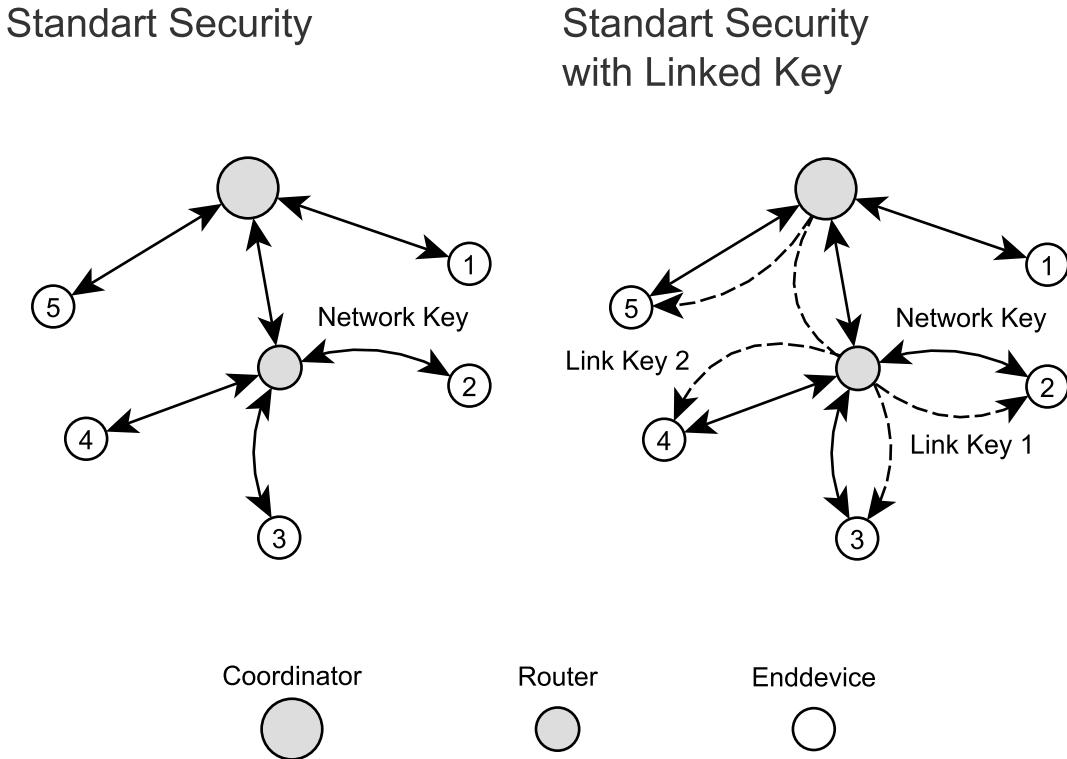


Abbildung 4.12.: Darstellung, wie die verschiedenen Schlüsselarten genutzt werden und zusammen wirken.

nator selbst), und wird von dort an unverschlüsselt zum *End Device* übertragen. Bei dieser unverschlüsselten Transaktion des Schlüssels wird die Sendeleistung des Senders soweit heruntergefahren, dass der Schlüssel nur zum *End Device* gelangen, darüber hinaus jedoch keinen weiteren entfernten Knoten erreichen kann. Damit soll das Abfangen des Schlüssels erschwert werden.

Die Authentifizierung läuft in beiden Fällen nahezu gleich ab: Jedes mal dient die *Extended Address* des Trustcenters als Basis. Diese wird vom *End Device* verschlüsselt empfangen und mit der in dessen Speicher abgelegten *Extended Address* verglichen. Im Falle einer Übereinstimmung gilt die Authentifizierung als erfolgreich. Ansonsten gilt die Teilnahme als gescheitert.

**Standard security with link Keys** Zum *Networkkey* kommt noch ein weiterer Schlüssel hinzu: der *Link key*. Mit ihm wird eine weitere zu verschlüsselnde Ebene hinzugefügt, genannt *Application Support Sublayer - APS*. Sie ist dem *Network Layer - NWK* übergeordnet und damit in dessen Payload integriert. Als erstes wird der *APS-Frame* mit dem *Link Key* verschlüsselt. Es kommt der Network Header hinzu und das Resultat wird wiederum mit dem *Network Key* verschlüsselt. Damit kann

jeder Netzwerknoten das Paket auf NWK-Ebene entschlüsseln und, wenn nötig bearbeiten, aber nicht auf den mit dem Link Key verschlüsselten Payload zugreifen. Diesen kann nur das *End Device* lesen, für welches die Nachricht bestimmt ist.

Die Authentifikation funktioniert hier über eine Liste von Paaren, bestehend aus *Extended Address* des Devices und dem dazugehörigen Link Key. Wenn keine passende *Extended Address* zu einem Device, welches dem Netzwerk beitreten möchte, gefunden wird, gilt die Authentifizierung als nicht erfolgreich und dem Device wird der Zutritt verwehrt.

**High security** um die Verteilung und Verwaltung der einzelnen Link keys zu erleichtern wurde der Modus *High Security* definiert. in diesem Fall übernimmt ein Protokoll mit dem Namen *Symmetric-Key Key Establishment - SKKE* die Schlüsselgenerierung und deren Verteilung. Auf Grundlage eines einzelnen Master keys, der entweder auf allen Devices vorinstalliert ist oder während des Betriebs angefordert werden kann, findet sowohl die Generierung der Link Keys statt, als auch die Authentifizierung. Die Richtigkeit des Coordinators wird über dessen *Extended Address* geprüft. Ein Device, welches dem Netzwerk beitreten möchte, authentifiziert sich über seinen Master Key. Mit ihm wird die Kommunikation während des Authentifizierungsprozess verschlüsselt. Eine Challenge Response stellt sicher, dass es sich bei den Geräten um jene handelt, die sie vorgeben zu sein. Nach erfolgreicher Authentifizierung werden die Link Keys auf Grundlage des Master Keys generiert und mit ihm verschlüsselt ausgetauscht.

In Abbildung 4.12 ist dargestellt, wie die einzelnen Keys ineinander greifen.

Um die einzelnen Modi einzuschalten, war es völlig ausreichend, die passende Bibliothek zu verlinken. Für *Standard security* musste die Bibliothek

`All_Sec_Rcb_Atmega128rf1_8Mhz_Gcc` genutzt werden. Im Falle des *Standard security with link Keys* Mode muss die Bibliothek `All_StdlinkSec_Rcb_Atmega128rf1_8Mhz_Gcc` gelinkt werden. Für den Mode *High Security* wurde keine Bibliothek mitgeliefert. ATMEL stellt diese nur auf Anfrage bereit. Damit konnte dieser Mode nicht überprüft werden.

Für die Verschlüsselung greift der BitCloud Stack immer auf den Securitylevel 5 zurück, sprich: Die Daten werden verschlüsselt und mit einer 32- Bit langen MIC geschützt.

Als Testnetzwerk wurde wiederum der selbe Aufbau genutzt, wie bereits in Kapitel 3.3 beschrieben. Auch die selben Devices kamen zum Einsatz, nur das eben ihre Firmware ausgetauscht worden ist.



# 5. FUNKTIONSTEST UND ENERGIEMESSUNG

## 5.1. ATMEL MAC Stack

### 5.1.1. Aufbau des Sensornetzwerks für den Funktionstest

Das Netzwerk hat die gleiche Struktur, wie der bereits in Abbildung 3.1 gezeigte Aufbau. Hinzu kommt noch ein Netzwerksniffer, um den Funkverkehr mitschneiden zu können, der über USB mit einem Laptop verbunden ist, und den jeweiligen Abläufen in der Software zuordnen zu können. Dabei handelt es sich einmal um ein Sensor Terminal Board<sup>1</sup> und ein darauf aufgesetztes Radio Controller Board RCB230<sup>2</sup>. Die von ihnen gelieferten Daten können mit der Software Wireshark<sup>3</sup> in Kombination mit der Software uracoli<sup>4</sup> ausgelesen und dargestellt werden. Zudem können die Daten, die vom Koordinator mittels UART<sup>5</sup> ausgegeben werden, über die USB-Schnittstelle mit Hilfe eines Terminals (hier wird HTerm<sup>6</sup> verwendet) gelesen werden. Diese Versuchsanordnung ist in Abbildung 5.1 abgebildet. Der Ablauf der Messung orientiert sich nach dem beschriebenen Aufbau aus Kapitel 3.3.

**Sniffer**

Um die Ergebnisse des Funktionstests interpretieren zu können, ist es sinnvoll auf die in Kapitel 3.3 gestellten Fragen zurückzukommen und diese zu beantworten.

---

<sup>1</sup><http://www.dresden-elektronik.de/funktechnik/products/boards-and-kits/development-boards/sensor-terminal-board/description/>

<sup>2</sup><http://www.dresden-elektronik.de/shop/prod68.html>

<sup>3</sup><http://www.wireshark.org/>

<sup>4</sup><http://www.nongnu.org/uracoli/>

<sup>5</sup>Siehe UART im Glossar

<sup>6</sup><http://www.der-hammer.info/terminal/>

## 5. Funktionstest und Energiemessung

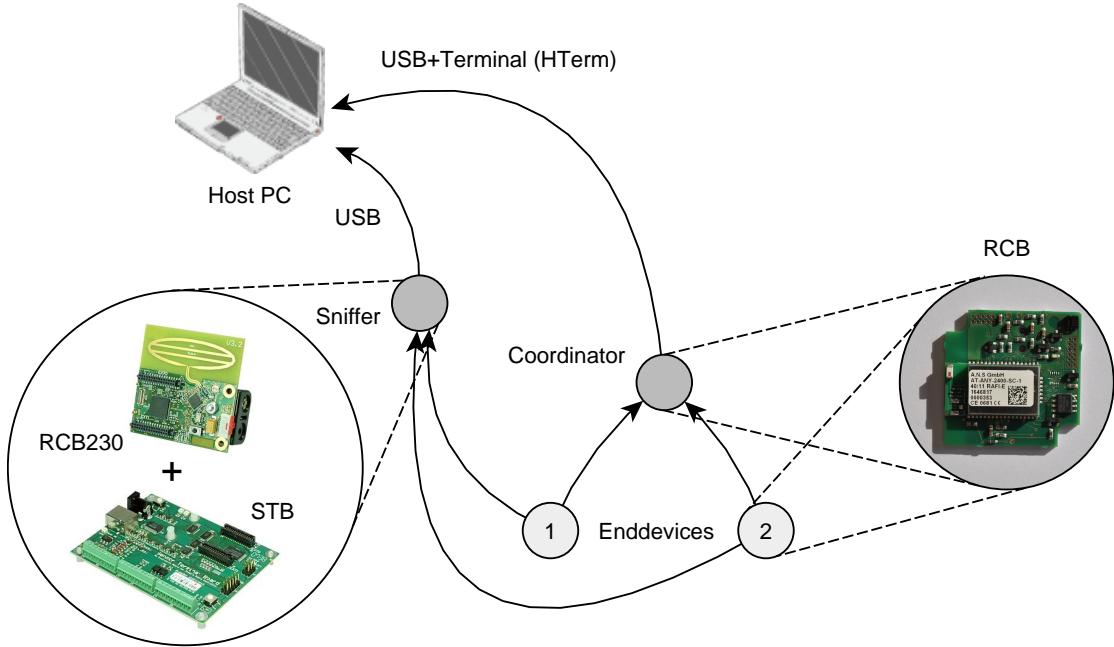


Abbildung 5.1.: Aufbau des Sensorsnetzwerks mit Sniffer

### Wie wird die Authentifizierung beim Anmeldevorgang und Datenaustausch sichergestellt?

Der MAC Stack bietet für den Anmeldevorgang eines Knoten an ein Netzwerk keinerlei solcher Funktionen an. Der Programmierer muss dies selbst implementieren. Für den Datenaustausch gilt dasselbe, obwohl über die Verwendung einer MIC sichergestellt ist, dass die Nachricht nicht nachträglich verändert werden kann. Um jedoch über die MIC eine Authentifizierung durchzuführen, muss ein System eine Schlüssel-Infrastruktur anbieten, was beim MAC Stack nicht der Fall ist. Auch dies muss vom Entwickler selbst implementiert werden. Durch die limitierten Hardwareressourcen jedoch fällt ein asymmetrisches Verschlüsselungssystem als Möglichkeit heraus. Ein System basierend auf Symmetrischen Schlüsseln ist jedoch anfälliger für Angriffe, da dies meistens auf einen einzigen *Master Key* basiert, dessen Bekanntwerden die Sicherheit des Systems zerstören würde.

### Wie werden, wenn überhaupt, die Schlüssel beim Anmelden ausgetauscht?

Da der MAC Stack kein Schlüsselaustauschsystem anbietet, müssen die Schlüssel vorher im Quelltext des Programms oder direkt in der Hardware implementiert werden. Sollen Schlüssel für Kommunikationspaare während des Betriebs erstellt werden (wie dies bei ZigBee über *Link Keys* und SKKE realisiert wird), muss dies vom Entwickler implementiert werden.

### Wie werden die Schlüssel für das Netzwerk und die verschiedenen Nodes verwaltet?

Wie bereits in Kapitel 4.1.4 geschildert, werden zwei Möglichkeiten angeboten. So kann einmal über eine globale ID und einem zusätzlichen Index aus einem Bund aus Schlüsseln einer ausgewählt werden, oder der MAC Stack wählt automatisch über die Adresse des Empfängers einen Schlüssel aus. Die erste Möglichkeit ist dabei die flexibelste von beiden.

### Wie wird die Integritätsprüfung der Nachricht ermöglicht?

Nachrichten können mit Hilfe einer MIC geschützt werden. Durch den Standard CCM/CCM\* kann davon ausgegangen werden, dass dieses System keine ausnutzbaren Schwächen besitzt. Durch verschiedene *Securitylevel*, die CCM/CCM\* anbietet, kann die Berechnung der MIC an die Gegebenheiten angepasst werden. Dies betrifft jedoch nur die Größe der MIC, auf die Geschwindigkeit ihrer Berechnung kann kein Einfluss genommen werden.

### Wie wird Rücksicht genommen auf spezielle Angriffe, wie z.B. eine Replay-Attack?

Da in die Berechnung der MIC auch der *Framecounter* mit eingeht, ist das System vor *Replay-Attacks* gut gefeilt. Der in [8] erwähnten *Precomputation-Attack* wird durch die Verwendung der *Extended Address* in der Nonce effektiv entgegengewirkt, der *Framecounter* tut dabei sein Übriges. Da die Nonce nicht auf Zufallszahlen basiert<sup>7</sup>, kann auch nicht das Problem beim WEP-Standard auftreten, dass sich der IV nach mehreren Paketen wiederholt.

### Gibt es allgemeine Designfehler? (wie z.B. bei den Protokollen Wired Equivalent Privacy, HDCP oder CSS)

Der Standard IEEE 802.15.4 ist ein Kompromiss zwischen Sicherheit, Leistungsfähigkeit und Patentfreiheit. Auch wenn die verwendeten Modi nicht die Neuesten und Effektivsten sind, sie sind allemal robust und widerstandsfähig gegen heute bekannte Angriffe, was sich die nächsten Jahre wohl kaum ändern wird.

Es hat sich jedoch herausgestellt, dass die Verschlüsselung in der Implementierung nur für Daten-Frames unterstützt wird, was dazu führt, dass beispielsweise Beacon-Frames unverschlüsselt übertragen werden und dadurch Informationen, ob kontextabhängig interpretierbar oder nur statistischer Natur, nach außen getragen werden können.

Die Verwaltung mehrerer Schlüssel wird durch einen Bug in der Software unmöglich. Zwar können mehrere Schlüssel gespeichert werden, jedoch führt der Bug dazu, dass für

---

<sup>7</sup><http://www.heise.de/security/meldung/Fehler-in-Software-fuer-ZigBee-Funkmodule-erleichtert-Lauschangriff-906983.html>

## 5. Funktionstest und Energiemessung

ein Device keine *Extended Address* gefunden werden kann, zumindest wenn mehrere Devices verwaltet werden sollen. Die *Extended Address* ist nötig um die richtige Nonce zu generieren. Der MAC Stack quittiert den Versuch eine Nachricht zu entschlüsseln mit einem Fehler, weil die falsche MIC berechnet worden ist. Mit der mitgegebenen und gepatchten Version der Datei `mac_secuirty.c` Siehe Anhang CD-Inhalt funktioniert die Schlüsselverwaltung auch für mehrere Devices wieder. Leider führt ein weiterer Bug dazu, dass der Quellcode für die Spezifikationen aus dem Jahre 2006 des IEEE 802.15.4 Standards nicht kompiliert werden kann, da hier die falschen Variablenannahmen zur Ermittlung der Frame-ID genutzt werden. Auch dieser Bug befindet sich in der selben Datei. Damit kann die Schlüsselauswahl in Abhängigkeit der Frame ID nicht genutzt werden.

Die Spezifikationen des *CCM/CCM\*-Mode* sichern das Netzwerk vor den typischen Angriffen ab. Es wurde besonders darauf geachtet, dass die Designfehler aus dem *WEP*-Standard nicht wiederholt werden. Anders als jedoch erwartet, lässt einem die Spezifikation nicht die freie Auswahl zwischen dem *CBC-MAC-Mode* und dem *CTR-Mode* zur Verschlüsselung. In jedem Fall kommen Beide Modi zum Einsatz, was eine doppelte Ver- und Entschlüsselung bedeutet, da CBC-MAC zur MIC-Generierung und der *CTR-Mode* zur Verschlüsselung genutzt wird. Bei großen Datenmengen macht der *CTR-Mode* Sinn, da auf jeden verschlüsselten Block unabhängig zugegriffen werden kann ohne dabei die anderen Blöcke zu entschlüsseln. Bei 100 Byte Daten (Ein Daten-Frame darf nicht weit über diese Größe hinaus gehen) scheint dies jedoch kaum ins Gewicht zu fallen, die doppelte Verschlüsselung jedoch in Anbetracht der Energieeffizienz schon. Der CBC-MAC könnte hier gleich zur Verschlüsselung genutzt werden, was den zweiten Durchlauf im *CTR-Mode* unnötig macht und damit kostbare Energie sparen würde.

Des weiteren hat sich herausgestellt, dass der *CCM/CCM\*-Mode* zwar die Unterscheidung zwischen Daten, die verschlüsselt und durch eine MIC geschützt werden sollen, und Daten, welche nicht verschlüsselt und nur durch eine MIC geschützt werden sollen, macht. Jedoch ist dies im IEEE 802.15.4-Standard so umgesetzt, dass die über ihm liegende Schicht nicht selbst entscheiden kann, welche Daten in welche Kategorie fallen. So werden nur die durch diesen Standard spezifizierten Header nicht verschlüsselt. Der gesamte Payload hingegen wird immer chiffriert, was auch die Header der darüber liegenden Schicht mit einschließt. So müssen beispielsweise im Falle von Routing-Informationen alle Daten komplett ent- und wieder verschlüsselt werden, wenn der Empfänger wissen möchte, ob das Paket für ihn bestimmt ist oder er es weiter schicken soll. Die Zeit, die die Verschlüsselung der Daten einnimmt, ist zwar sehr gering, jedoch kann sich dies durchaus beim Versenden über viele Knoten negativ auswirken und eine durchaus zu messende Zeitverzögerung verursachen. Dieses Manko hat weitreichendere Konsequenzen, als auf den ersten Blick ersichtlich ist. Die Schicht, die auf dem MAC Stack aufsetzt, muss sich nun entscheiden, ob sie die Sicherheitsmechanismen des MAC Stack nutzt, oder auf diese gänzlich verzichtet, um ihre eigenen Sicherheitsmechanismen integriert. Im ersten Fall muss der gesamte Payload verschlüsselt werden und bringt damit die oben erwähnten Nachteile mit sich. Im zweiten Fall jedoch kümmert sich die Schicht über dem MAC Stack selber um die Verschlüsselung und die Generierung der MIC. Und hier lauert die

Falle: Diese Schicht hat keinen Zugriff auf den Header des MAC Stack, der im jedem Datenpaket zu finden sein wird. Das bedeutet, diese Schicht kann nur für den Payload eine MIC generieren und muss den MAC Header davon ausnehmen. Dieser muss also ungesichert vom MAC Stack akzeptiert werden und da liegt das Problem.

Alles in allem jedoch funktioniert die grundlegende Verschlüsselung auf einem mehr als akzeptablen Niveau. Die Schlüsselverwaltung bietet die Möglichkeit, komfortabel aus mehreren Schlüsseln einen für jedes Kommunikationspaar auszuwählen. Die Umsetzung aller Spezifikationen des *CCM/CCM\*-Mode* werden dem Programmierer abgenommen, und alle übrigen von *CCM/CCM\** nicht festgelegten Details, wie die Generierung der Nonce, werden vom MAC Stack implementiert. Die Daten, die verschlüsselt gesendet werden, konnten mit einem Sniffer zwar abgefangen, jedoch ohne den Schlüssel nicht gelesen werden.

### 5.1.2. Ergebnisse der Energiemessung

Die Abbildungen 5.2, 5.3 und 5.4 zeigen den Verlauf des Stroms durch den Messwiderstand. Die Auswirkungen des Verstärkers wurden bereits herausgerechnet.

Es wurden drei Trigger-Signale in der Software gesetzt und mit dem Oszilloskop abgegriffen. Das Setzen selbiger ist im Quellcode des ATTEL MAC Stack an verschiedenen Stellen verteilt und signalisiert den Eintritt oder Austritt in oder aus einer Funktion. Diese sind in den grau hinterlegten Bereichen aufgeführt.

Trigger-Signal 1 kennzeichnet bei steigender Flanke den Eintritt in die Funktion `build_data_frame()`, welche das Daten-Frame erstellt, und mit fallender Flanke den Eintritt in die Funktion `tx_done_handling()`.

Trigger-Signal 2 kennzeichnet bei steigender Flanke den Eintritt in und bei fallender Flanke den Austritt aus der Funktion `stb_ccm_secure()`.

Trigger-Signal 3 zeigt bei steigender Flanke den Beginn der Berechnung der MIC mittels der Funktion `compute_mic()` und bei fallender Flanke das Ende der Verschlüsselung des Payload und der MIC durch die Funktion `encrypt_pldmic()`.

## 5. Funktionstest und Energiemessung

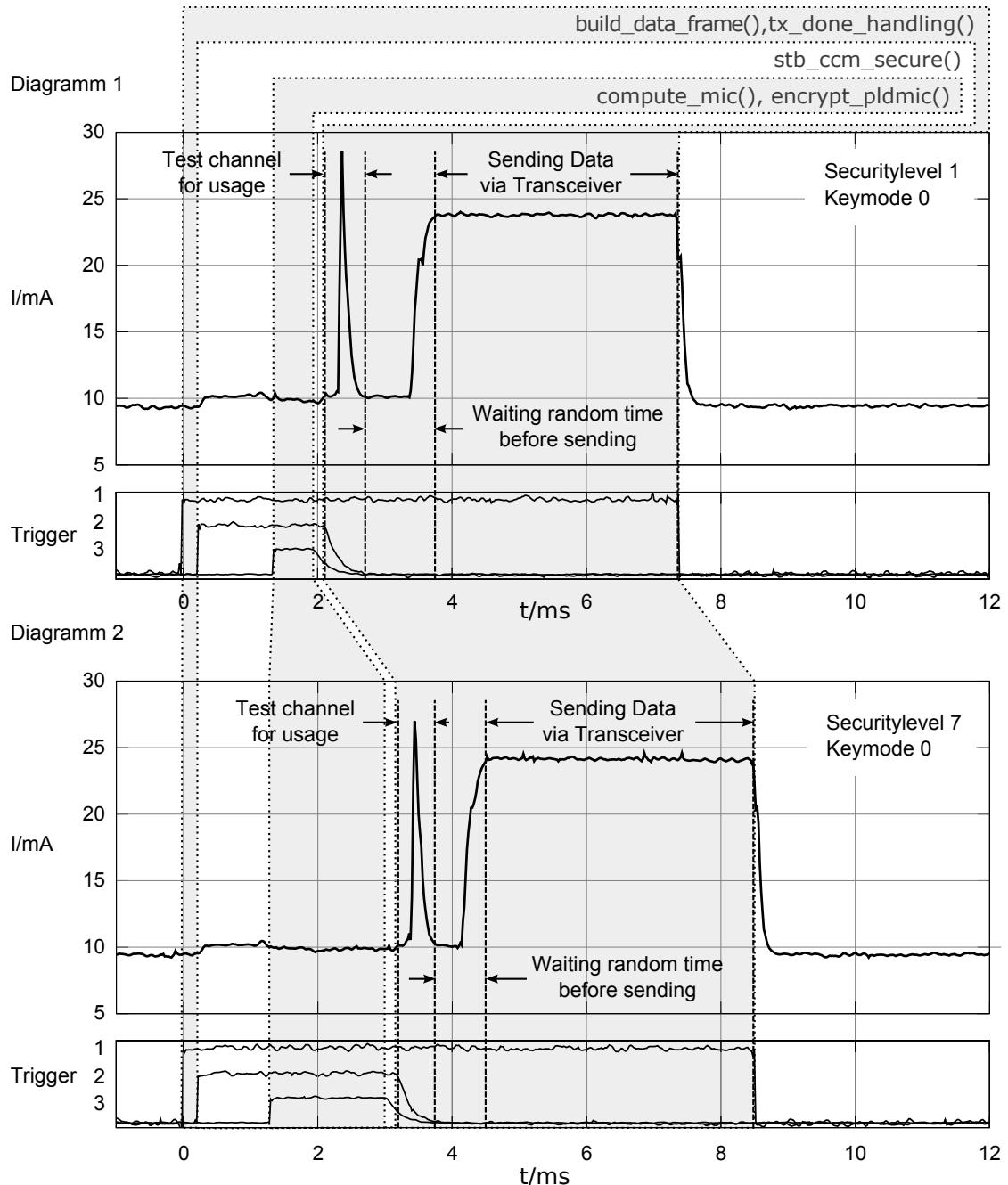


Abbildung 5.2.: Messung des Stromverbrauchs.

Oben (1): Nur MIC-32: 1,45ms zusätzlich

Unten (2): Verschlüsselung und MIC-128: 2,98 ms zusätzlich

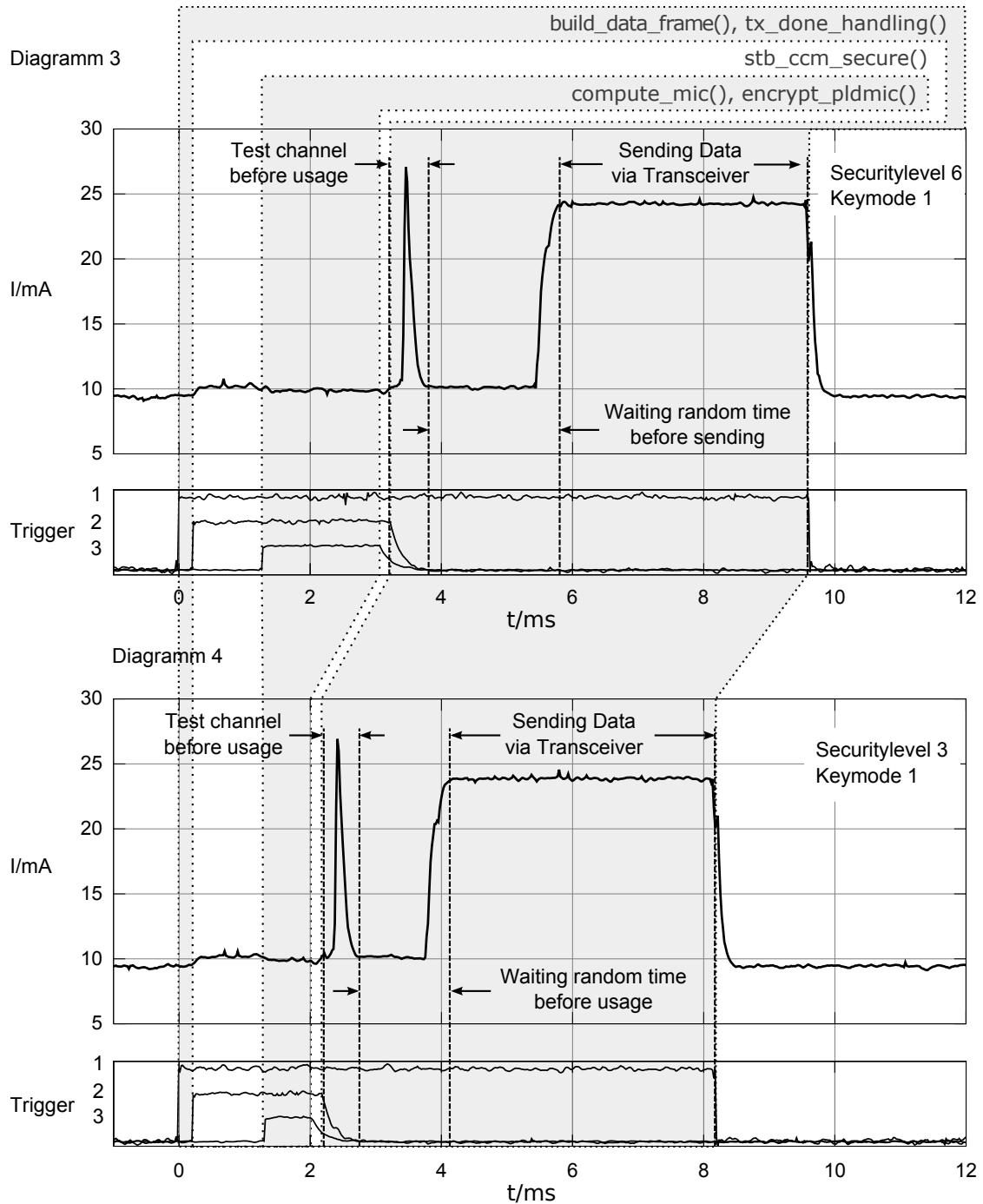


Abbildung 5.3.: Messung des Stromverbrauchs.

Oben (3): Verschlüsselung und MIC-64: 2,76 ms zusätzlich  
Unten (4): Nur MIC-128: 1,88 ms zusätzlich

## 5. Funktionstest und Energiemessung

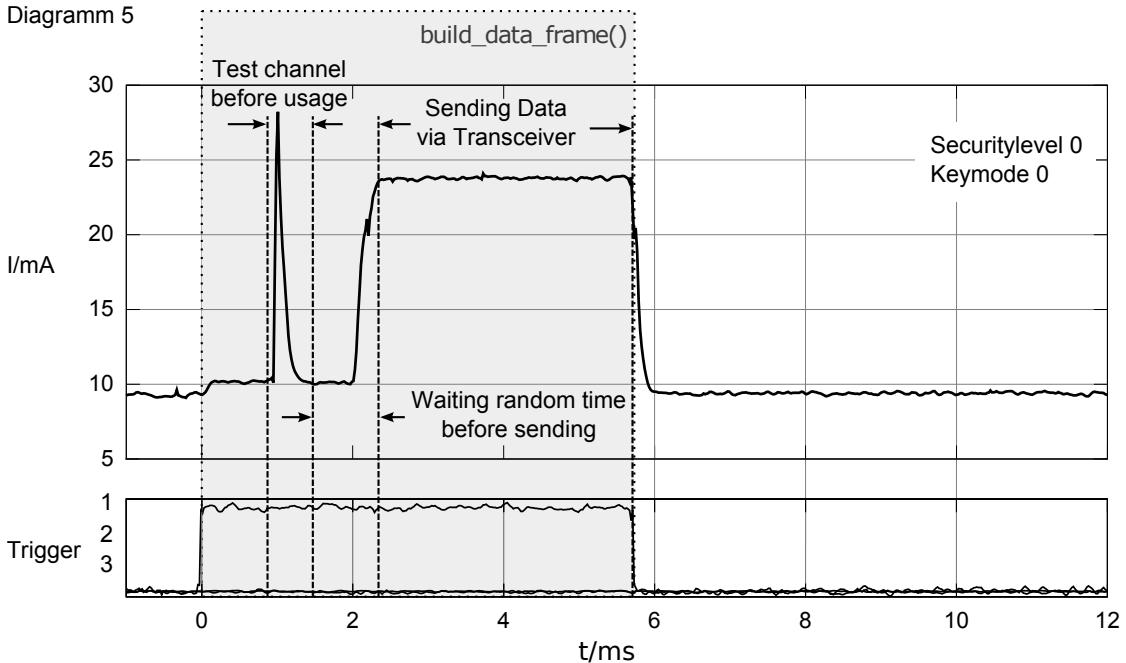


Abbildung 5.4.: Messung des Stromverbrauchs.

(5) Weder Verschlüsselung noch mit einer MIC geschützt.

Man kann anhand der Messkurve des Stroms grob drei Zustände beobachten: Die Kurve hält sich im ersten Bereich auf einem Niveau von etwa  $9mA$ . Hier rechnet der Prozessor. Der Receiver ist ebenfalls angeschaltet, da sich in ihm die Hardwareeinheit zur Verschlüsselung befindet, der Transceiver hingegen ist deaktiviert. Als nächstes kommt eine starke Messspitze von etwa  $28mA$ . In diesem kurzen Zeitbereich von etwa  $500s$  wird der Receiver hochgefahren und mit voller Leistung der Kanal gescannt, ob er frei ist und der Transceiver somit senden kann. Die Messspitze liegt mit ihrem Wert über dem Strombedarf des Transceivers, da er beim Empfangen mehr Energie benötigt als der Transceiver beim Senden. Es folgt ein weiterer Bereich mit konstanten  $9mA$  wie zu Beginn. Hier wartet das Device eine zufälligen langen Zeitabschnitt ab, bevor es mit dem Senden beginnt. Zum Schluss kommt ein etwas längerer Bereich mit einer Stromstärke von  $23,5mA$  über einen Zeitraum von  $3,3ms$  bis  $4ms$ . Hier werden die Daten mit Hilfe des Transceivers über die Luft übertragen. Am Ende der Übertragung fällt der Pegel wieder auf  $9mA$  ab.

Im ersten Bereich findet das Zusammenstellen des Daten-Frames statt, sowie dessen Verschlüsselung und Sicherung mittels einer MIC, falls dies so konfiguriert worden ist. Die für diesen Prozess benötigte Zeit liegt zwischen  $0,84ms$  und  $3,18ms$ . Am kürzesten fällt die Zeit mit  $0,84ms$  im Diagramm 5 aus Bild 5.4 aus, da hier nicht verschlüsselt werden soll. In Diagramm 1 Bild 5.2 und Diagramm 4 Bild 5.3 beträgt die Zeit  $2,08ms$  und  $2,2ms$ , da hier nur die MIC berechnet, jedoch nichts verschlüsselt werden soll. In

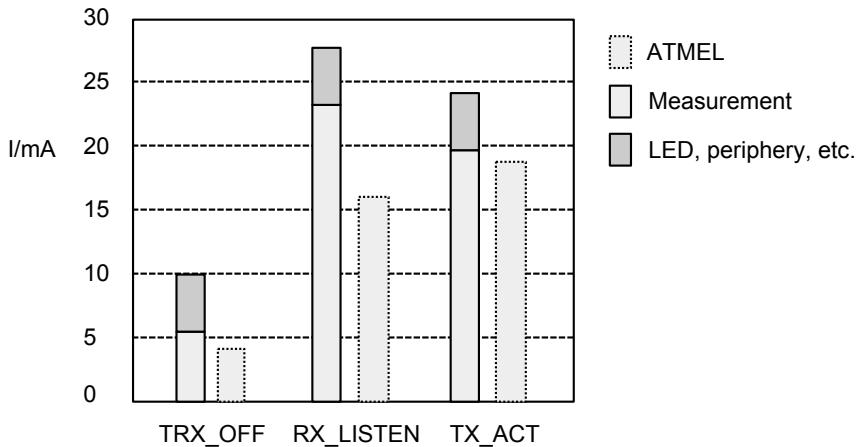


Abbildung 5.5.: Messung des Stromverbrauchs. Zu sehen sind die Stromwerte in Abhängigkeit des Zustandes, verglichen mit den offiziellen Werten von ATMEL

Diagramm 2 Bild 5.2 und Diagramm 3 Bild 5.3 schließlich soll sowohl eine MIC berechnet, als auch die Daten verschlüsselt werden. Aus der Funktionsweise von *CCM/CCM\** ergibt sich, dass nochmals neben der MIC Berechnung in *CTR-Mode* verschlüsselt werden soll. Dies erklärt eine Zeitspanne von  $3,18ms$  in beiden Fällen.

Der zweite Bereich, in welchen der Kanal gescannt wird, hat in allen Fällen die selbe Dauer.

Der dritte Bereich, das Senden der Daten, ist mit einer Dauer von  $3,34ms$  in Diagramm 5 Bild 5.4 am kürzesten, da das Daten-Frame eine Länge von 103 Byte hat. Diagramm 1 Bild 5.2 hat eine Länge von  $3,55ms$  beim Senden von 112 Byte. Diagramm 2 Bild 5.2 zeigt eine Länge von  $3,97ms$  bei einem 124 Byte Frame. Diagramm 3 Bild 5.3 hat eine Länge von  $3,76ms$  und eine Framelänge von 117 Byte. Diagramm 4 Bild 5.3 zeigt eine Länge von  $3,86ms$  und eine Framelänge von 125 Byte. Die unterschiedlichen Längen ergeben sich daraus, dass unterschiedliche Header wegen der Verschlüsselung hinzukommen, um diese auch verwälten zu können. Auch die Länge der MIC, die an das Ende des *Payload* angehängt wird, hat einen großen Einfluss auf die Framelänge. Neben dem *Auxiliary Security Header* kommt in den Diagrammen 3 und 4 noch der *Key Index* mit einer Länge von 1 Byte hinzu. Dies erklärt die unterschiedlichen Längen dieses Bereichs.

Sollen der erste Bereich (Sicherung durch die Funktion `stb_ccm_secure()`) und der zweite Bereich (Senden der Daten) als Basis dienen, um die Zeitspanne in den verschiedenen Modi zu untersuchen und die gemessenen Werte mit denen aus Diagramm 5 verglichen werden, ergibt sich nun folgendes Bild:

Diagramm 1 hat eine zusätzliche Dauer von  $1,45ms$ , Diagramm 2 hat eine zusätzliche Dauer von  $2,98ms$ , Diagramm 3 hat eine zusätzliche Dauer von  $2,76ms$  und Diagramm 4 eine zusätzliche Dauer von  $1,88ms$ .

## 5. Funktionstest und Energiemessung

Die starken Zeitunterschiede lassen sich zum einen damit erklären, dass in Diagramm 1 und 4 keine Verschlüsselung statt findet, was die Dauer der Sicherung verkürzt, während in Diagramm 2 und 3 die Daten zusätzlich verschlüsselt werden, was etwa  $0,7ms$  zusätzlich dauert. Zum anderen hängt die Zeit auch von der Menge der Daten ab, die versendet werden soll. Wenn nur eine 32 Bit lange MIC gewünscht ist und der *Key Index* fehlt, da ein *Keymode* von 0 gewählt wurde (was zusätzlichen 4 Byte Daten entspricht wie in Diagramm 1), werden weniger Daten versendet, als wenn eine 128 Bit lange MIC Verwendung findet und der *Keyindex* 1 ist (da hier 17 Byte Daten zusätzlich versendet werden, wie in Diagramm 4). Die Kombination aus der Sicherungsart und der Menge an Daten, bedingt durch die zusätzliche MIC und die eingefügten Header-Daten, ist entscheidend für die gemessenen Zeitunterschiede.

In Abbildung 5.5 werden die gemessenen Werte in den verschiedenen Zuständen noch einmal grafisch dargestellt. Als Vergleich sind ihnen die Messwerte von ATMEL<sup>8</sup> zur Seite gestellt. Da sich auf dem Board noch weitere Peripherie befindet, die zusätzlich Energie benötigt (wie beispielsweise eine LED), sind die gemessenen Werte ein wenig zu groß. Ausgehend davon, dass die LED etwa  $5mA$  benötigt, schrumpfen die Messwerte jedoch wieder nahe an die offiziellen Werte von ATMEL heran. Zu beachten ist außerdem, dass der Messwiderstand die Werte ebenfalls im geringen Maße verfälscht. Damit bleibt zu sagen, dass die gemessenen Stromwerte mit den angegebenen Werten von ATMEL gut übereinstimmen.

Allerdings ist es bedauerlich, dass der *CCM/CCM\*-Mode* nicht die Möglichkeit bietet selbst zu entscheiden, ob die Daten zusätzlich noch einmal mit dem *CTR-Mode* verschlüsselt werden sollen oder auf den zweiten Durchlauf mit diesem Mode verzichtet werden sollte, da bei der MIC-Generierung mittels CBC-MAC die Daten ebenfalls verschlüsselt werden. Bei dieser geringen Menge an Daten bietet der CTR-Mode kaum Vorteile, jedoch nur den Nachteil, dass die zusätzliche Verschlüsselung  $0,7ms$  benötigt.

## 5.2. BitCloud

Die Verschlüsselung funktionierte in allen Fällen ohne Probleme. Die Konfiguration dieser war äußerst komfortabel und mit wenig Aufwand verbunden. Der ZigBee Stack nahm dem Programmierer dabei viel Arbeit ab.

### Wie wird die Authentifizierung beim Anmeldevorgang und Datenaustausch sichergestellt?

Die Details hängen von der gewählten Sicherheitsstufe ab (siehe Kapitel 4.2). Allerdings ist für jede Stufe gleich, dass die *Extended Address* des *Trust Centers* zur Authentifizierung dient. Das teilnehmende Gerät kann sich hingegen nur über den bereits gespeicherten

---

<sup>8</sup>ATmega128RFA1 datasheet S.4

Schlüssel authentifizieren, oder darüber, dass es bereits im Trustcenter in einer Liste eingetragen ist und so auch ohne Schlüssel teilnehmen kann, der ihm dann unverschlüsselt zugeschickt wird. In jedem Fall stellt sich die Frage, ob ein Angreifer an die *Extended Address* des *Trust Centers* gelangen und damit dessen Identität vortäuschen kann. Dies ist zumindest nicht unmöglich.

Je nach der gewählten Sicherheitsstufe kann sich ein Netzwerknoten mithilfe seines Schlüssels beim Datenverkehr authentifizieren. Dies kann entweder der *Network Key* sein, womit eine individuelle Authentifizierung nicht möglich ist, da diesen Schlüssel jeder Knoten besitzt, oder ein *Link Key*.

#### **Wie werden, wenn überhaupt, die Schlüssel beim Anmelden ausgetauscht?**

Die sicherste Methode ist es, die Schlüssel vor Inbetriebnahme auf dem Knoten zu installieren, sie ist allerdings auch die aufwendigste. Des weiteren können die Schlüssel auch unverschlüsselt übertragen werden, damit ein Knoten dem Netzwerk ohne Schlüssel beitreten kann. Davon wird jedoch in dieser Arbeit abgeraten. Zum Schluss besteht noch die Möglichkeit über das Protokoll SKKE *Link Keys* während des Betriebs zu erzeugen und zu verteilen. Zwar ist dieser Weg sehr komfortabel, allerdings hängt die Sicherheit hier von einem einzigen *Master Key* ab, der ebenfalls irgendwie auf den Geräten vorhanden sein muss.

#### **Wie werden die Schlüssel für das Netzwerk und die verschiedenen Nodes verwaltet?**

BitCloud übernimmt die Verwaltung zum größten Teil eigenständig. Zumindest die *Link Keys* - sollten diese genutzt werden - müssen vom Entwickler mit der passenden *Extended Address* in den Geräten gespeichert werden. Ansonsten ist nur das Speichern des *Network Keys* notwendig. Sollte SKKE zur Anwendung kommen, übernimmt dieses Protokoll die Verwaltung der *Link Keys* eigenständig.

#### **Wie wird die Integritätsprüfung der Nachricht ermöglicht?**

Mit Hilfe einer MIC werden alle Nachrichten vor unbefugten Veränderungen geschützt. Dieser Mechanismus unterscheidet sich kaum von den IEEE 802.15.4 Standards.

#### **Wie wird Rücksicht genommen auf spezielle Angriffe, wie z.B. eine Replay-Attack?**

Die Techniken in BitCloud sind denen im ATMEL MAC Stack gleich, da dieser die Generierung der Nonce von BitCloud übernommen hat. Damit ist auch das Sicherheitsniveau dem des IEEE 802.15.4 Standards gleichwertig.

## 5. Funktionstest und Energiemessung

### Gibt es allgemeine Designfehler? (wie z.B. bei den Protokollen Wired Equivalent Privacy, HDCP oder CSS)

Die Tatsache, dass ZigBee auf dem MAC Stack aufbaut, gestaltet den Zugriff für Zigbee auf diesen recht schwierig. Vor allem auf die Datenbereiche eines Frames des MAC Stack kann ZigBee nicht zugreifen, was dazu führt, dass ZigBee seine eigene Sicherheitsarchitektur implementiert und die des MAC Stack außen vorlässt. Mit dieser eigenen Lösung jedoch können Bereiche des Frames, auf die ZigBee keinen Zugriff hat, nicht abgesichert werden. Dies betrifft vor allem den *MAC Header*, der bei ZigBee nicht durch die MIC geschützt werden kann.

Die Energiemessung jedoch konnte nicht durchgeführt werden, da die benötigten Triggersignale nicht gesetzt werden konnten. Der Quelltext von BitCloud steht zu einem großen Teil nicht zur Verfügung und konnte damit auch nicht verändert werden.

# 6. ZUSAMMENFASSUNG UND AUSBLICK

Zusammenfassend lässt sich sagen, dass die Verschlüsselung sowohl beim ATMEL MAC Stack als auch beim ATMEL BitCloud Stack funktioniert. Das bedeutet, es ist möglich die Kommunikation und das Netzwerk vor Manipulationen gegen die meisten Angriffe zu schützen. Dabei stehen vor allem die Probleme des WEP-Standards im Vordergrund (Replay-Attack, sich wiederholende IVs). Sämtliche Probleme oder Nachteile der Implementierungen oder Spezifikationen halten sich in Grenzen, können ausgeglichen oder auch verkraftet werden (wie z.B. die doppelte Verschlüsselung im *CCM/CCM\*-Mode*, der Bug im ATMEL MAC Stack, der ohne viel Aufwand gepatcht werden kann), da der Gewinn an Sicherheit dies um ein Vielfaches wieder aufwiegt.

Die Achillesferse des gesamten Systems besteht jedoch darin, dass keine asymmetrische Verschlüsselung zur Verfügung steht. Entweder müssen alle Schlüssel auf jedem Gerät vorinstalliert werden, was sehr aufwändig und bei großen Netzwerken kaum zu bewerkstelligen ist, oder sie werden mittels eines Protokolls verteilt. Im letzten Fall hängt die Schlüsselweitergabe jedoch immer von nur einem einzigen Schlüssel, dem Master Key, ab. Sollte dieser in fremde Hände gelange, ist die gesamte Sicherheit des Netzwerkes in Gefahr und alle Mechanismen bieten keinerlei Sicherheit mehr. Es wäre also wünschenswert, ein asymmetrisches System zu implementieren, welches mit geringeren Schlüssellängen auskommt. Die Lösung liegt in der *Elliptic Curve Cryptography - ECC*. Die Schlüssellänge befinden sich hier im Bereich der symmetrischen Kryptosysteme. Dadurch lässt sich dies deutlich effizienter in Hardware im Embedded Bereich umsetzen. Dieses Kryptosystem ist zwar generell frei von Patenten, die effiziente Implementierung in Hardware jedoch ist es nicht. Diese Patentbelastung verhindert eine Implementierung. Es wäre daher wünschenswert, die Verfahren soweit zu verfeinern, dass sie in diesem hier beschriebenen Szenario funktionstüchtig sind, keinerlei Patente verletzen und trotzdem effizient funktionieren.

## 6. Zusammenfassung und Ausblick

Des weiteren gibt es durchaus symmetrische Schlüsselaustauschverfahren, die auf einem annehmbaren Niveau arbeiten. Eines davon ist das *Blom-Verfahren*.<sup>1</sup> Allein mit der Hilfe der Algebra lassen sich mittels einer Matrix Schlüssel generieren, die dann zur Kommunikation genutzt werden können. Zwar existiert auch hier ein Master Key und markiert damit wieder einen Schwachpunkt, jedoch lässt sich diese Matrix zerteilen und damit auf mehrere Devices verteilen. Es müssen erst mehrere Devices vom Angreifer in Besitz gebracht werden, um das Verfahren zu knacken. Die Anzahl der Geräte, die kompromittiert werden müssen, hängt von den Dimensionen dieser Matrix ab. Je größer sie ist, desto mehr mehr Devices sind es, desto höher ist also die Sicherheit. Allerdings beschränkt auch hier die niedrige Speicherausstattung der Embedded Devices diese Größe. Allerdings gibt es Verfahren, die einzelne Elemente dieser Matrix in Echtzeit berechnen können, ohne dass diese komplett im Speicher vorrätig sein muss. Auch hier ist die Art der Implementierung wieder der imitierende Faktor. So wurde dieses Verfahren im Kopierschutzsystem *HDCP* eingesetzt, jedoch auch nach einiger Zeit gebrochen.<sup>2</sup> <sup>3</sup>. So wurde sogar eine Man-In-The-Middle-Attack<sup>4</sup> erfolgreich gegen dieses System eingesetzt.

Das Hauptaugenmerk liegt in dieser Arbeit besonders darin, die Software zu analysieren und die Spezifikationen und deren Umsetzung auf Schwachstellen zu überprüfen. Was dabei außen vorgelassen wurde, ist die Hardware, auf der die Software läuft. So sind die Schlüssel das Fundament eines jeden Kryptographischen Sicherheitssystems. Sobald sie bekannt werden, greift kein einziges Sicherheitsmerkmal mehr. Es stellt sich also die Frage, wo diese Schlüssel gespeichert werden. An die Devices zu gelangen ist unter diesen Randbedingungen kein großes Problem. Diese auseinander zu nehmen und deren Inhalt zu analysieren ist ein Szenario, das recht wahrscheinlich ist, da die Technik dazu heute schon mit verhältnismäßig wenig Geld erworben werden kann. Sollten die Schlüssel sich z.B. im Flash befinden, kann dieser nach Abschalten des Geräts und Ablöten der Bausteine einfach ausgelesen werden. Auch im flüchtigen Speicher kann dieser mit spezieller Technik noch ausgelesen werden, wie beispielsweise mittels einer *Cold-boot-Attack*<sup>5</sup>. Es gibt bereits Projekte, in denen versucht wird das Schlüsselmaterial in Registern des Prozessors zu speichern, da es unwahrscheinlich ist, dass ein Angreifer an diese gelangen kann ohne den Chip vorher zu zerstören.<sup>6</sup>

Dies sind alles Szenarien, die durch die rasante Entwicklung der Technik immer wahrscheinlicher werden. Sie müssen in die Entwicklung von Spezifikationen und Standards mit einfließen, sonst verhält sich jedes Kryptografische Sicherheitssystem wie eine Kette mit nur einem schwachen Glied: Sie bricht auseinander.

---

<sup>1</sup>Siehe [6]

<sup>2</sup><http://www.heise.de/newsticker/meldung/Intels-HDCP-Chiffrierung-geknackt-46312.html>

<sup>3</sup><http://www.heise.de/newsticker/meldung/Intels-HDCP-Videoverschlüsselung-angeblich-geknackt-1078386.html>

<sup>4</sup><http://www.heise.de/security/meldung/Forschern-gelingt-Man-in-the-Middle-Angriff-auf-HDCP-Kopierschutz-1384461.html>

<sup>5</sup><http://de.wikipedia.org/wiki/Kaltstartattacke>

<sup>6</sup><http://events.ccc.de/congress/2011/Fahrplan/events/4869.en.html>

# A. CD-INHALT

**latex** enthält den Latex-Quellcode für die PDF-Datei

**img** enthält die Bilddateien teilweise im Format SVG, PNG, JPEG und PDF

**graph** enthält die Graphen im graphml-, SVG- und PDF-Format

**license** enthält die CC-Lizenz im Format XMP

**Messung** enthält die CSV-Dateien von den Messungen des Oszilloskops und das dazugehörige Skript für Gnuplot zum Erstellen der Diagramme

**patch** enthält die gepatchte Version der Datei `mac_security.c`

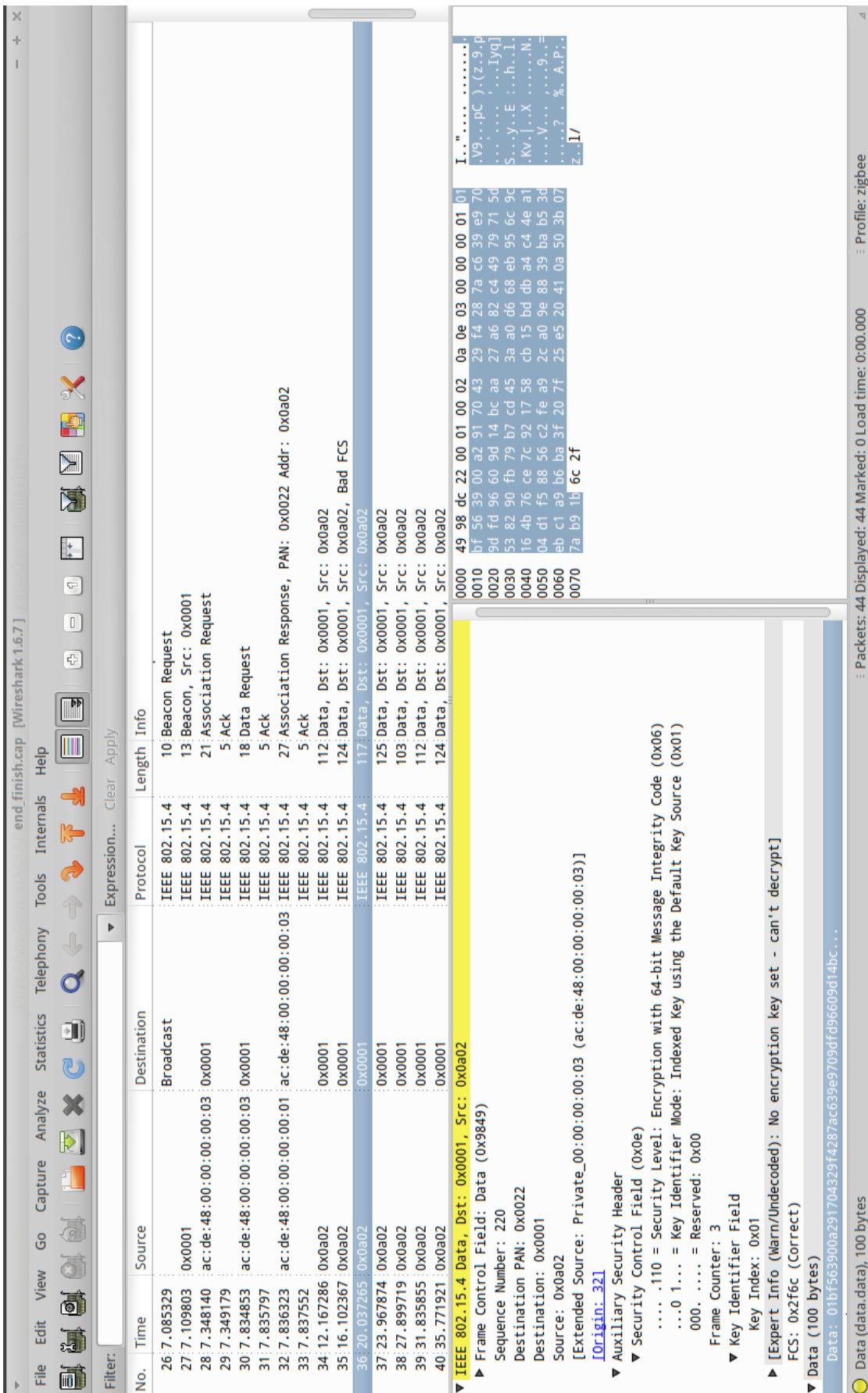
**Präsentation** enthält die Folien für den Zwischenvortrag

**wireshark**



## B. SCREENSHOTS VON WIRESHARK

## B. Screenshots von Wireshark



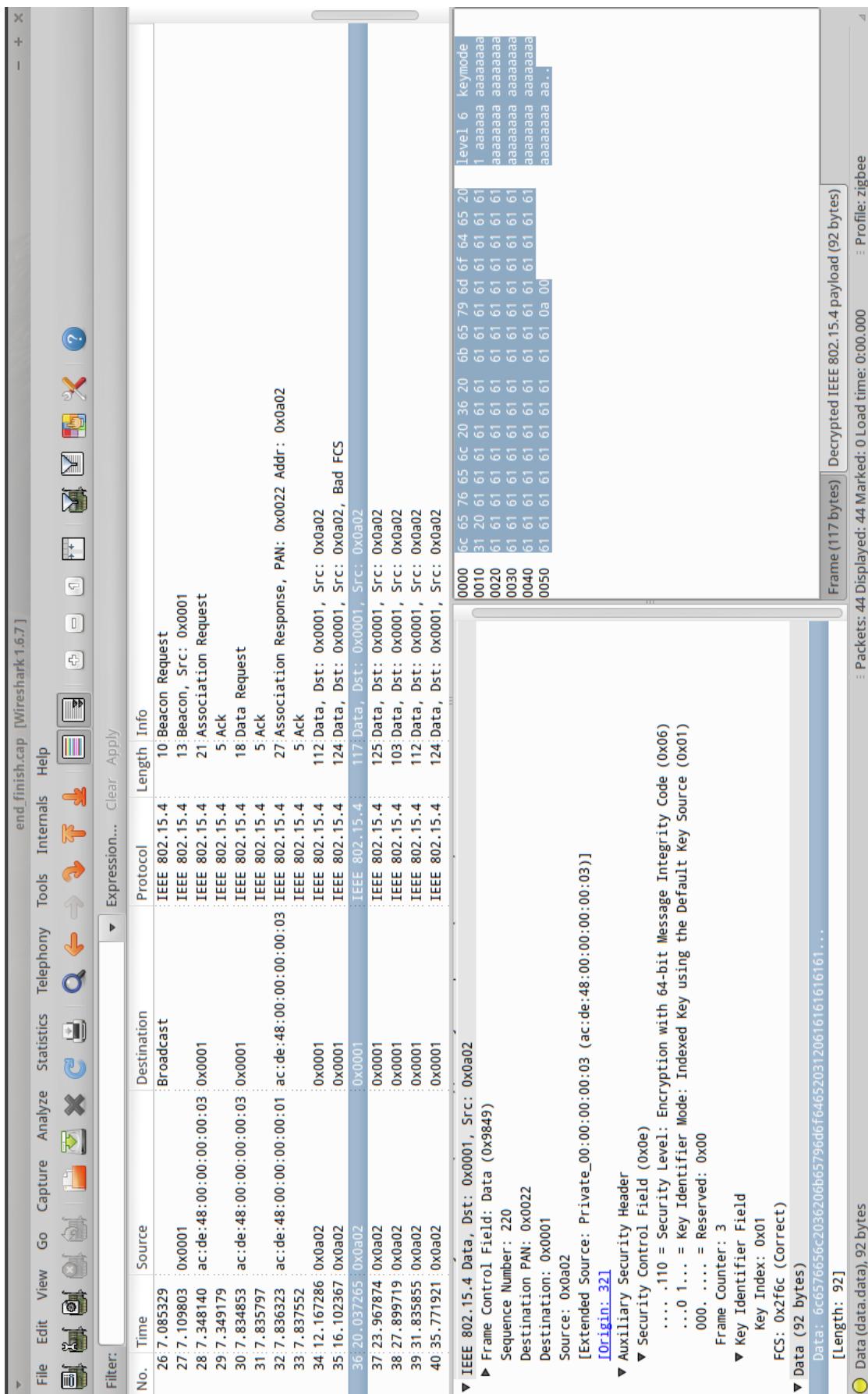


Abbildung B.2.: Wireshark: Entschlüsselter Daten-Frame



# C. PROGRAMM-ABLAUF-PLAN

## C.1. Device

### C. Programm-Ablauf-Plan

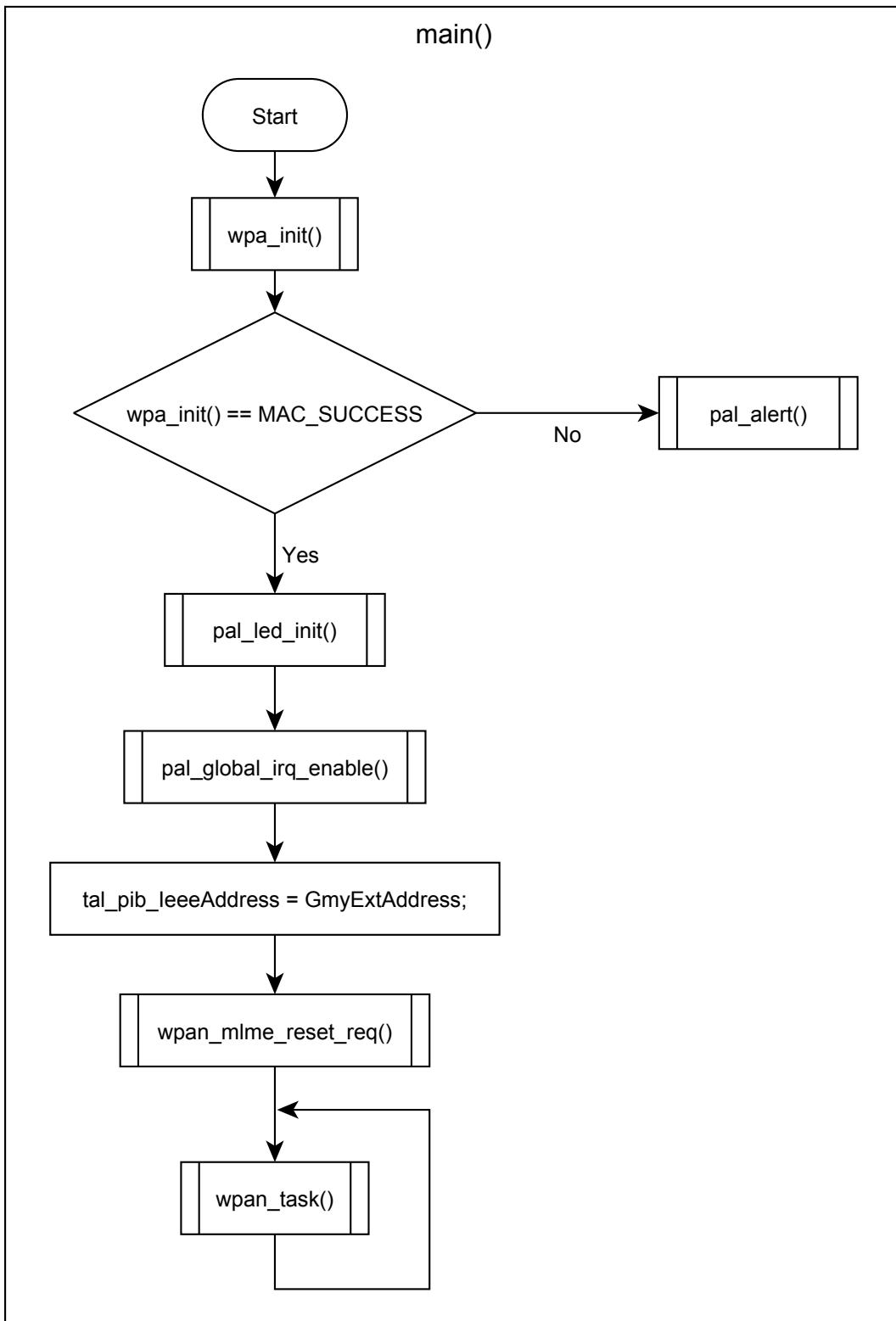


Abbildung C.1.: Device: `main()`

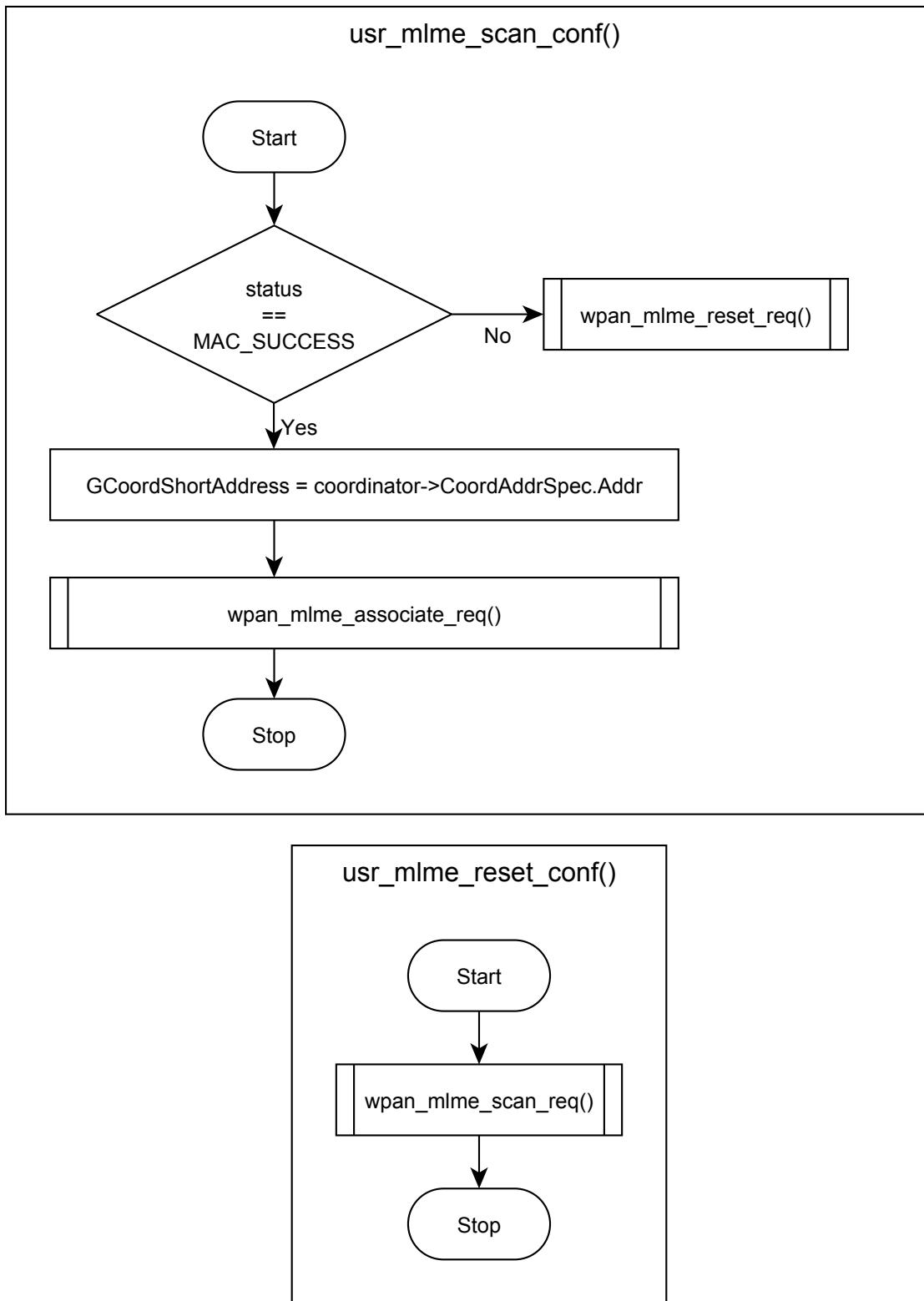


Abbildung C.2.: Device: `usr_mlme_scanf_conf()`, `usr_mlme_reset_conf()`

### C. Programm-Ablauf-Plan

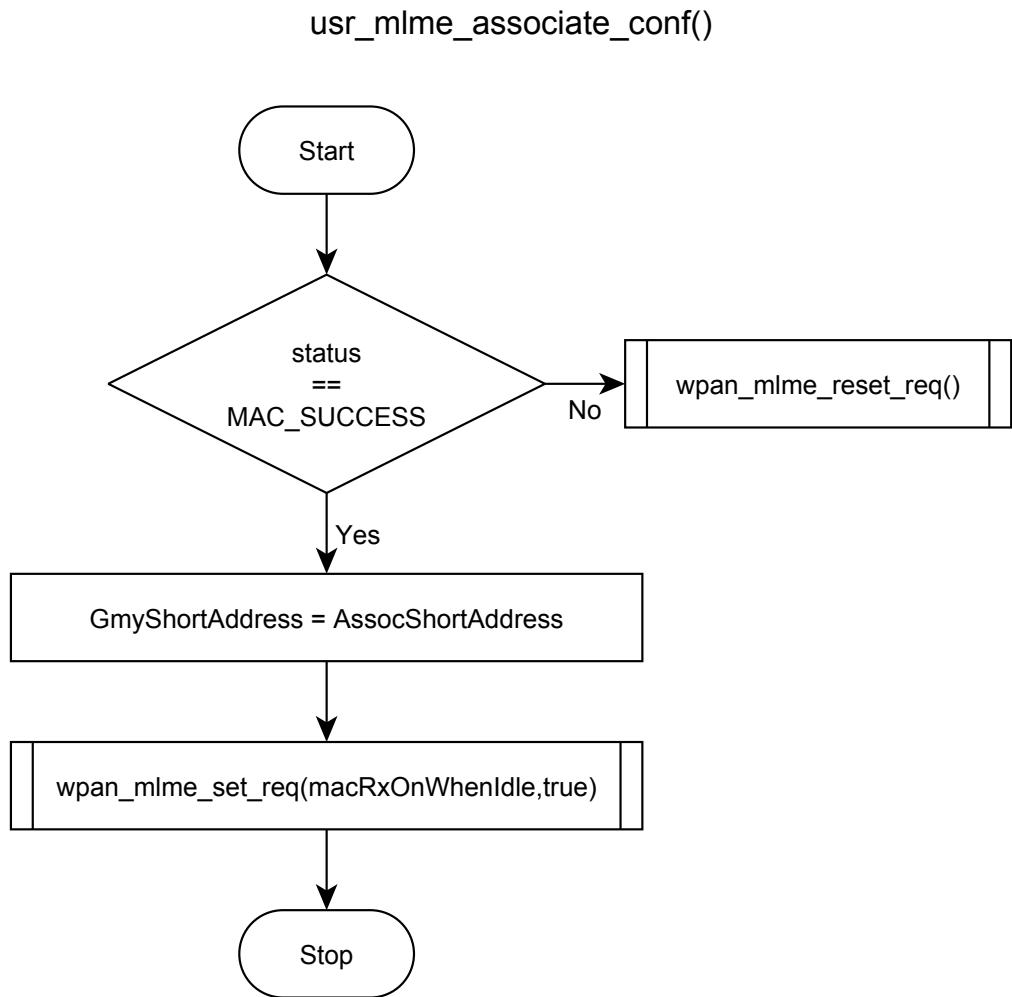


Abbildung C.3.: Device: usr\_mlme\_associate\_conf()

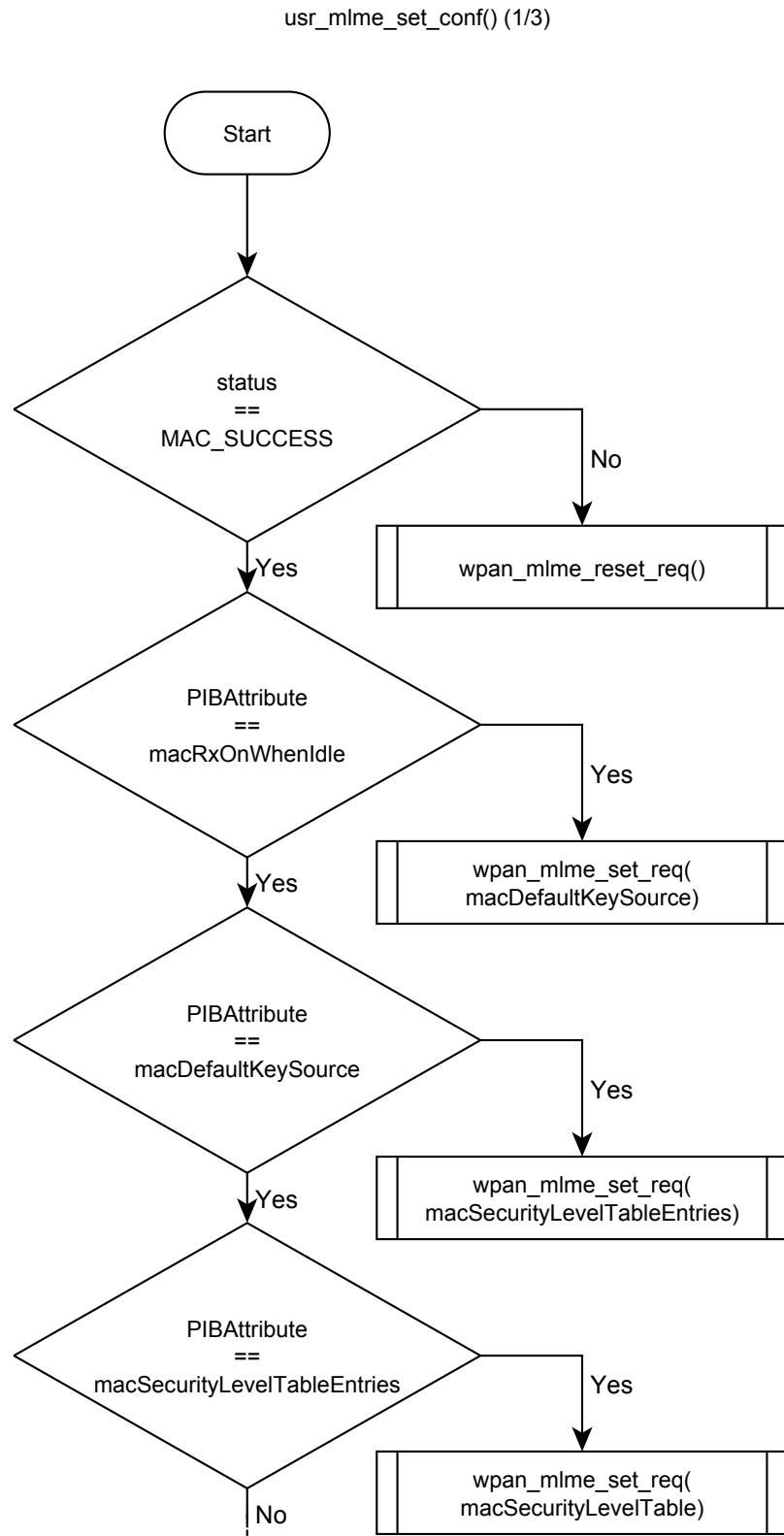


Abbildung C.4.: Device: `usr_mlme_set_conf()`

### C. Programm-Ablauf-Plan

`usr_mlme_set_conf() (2/3)`

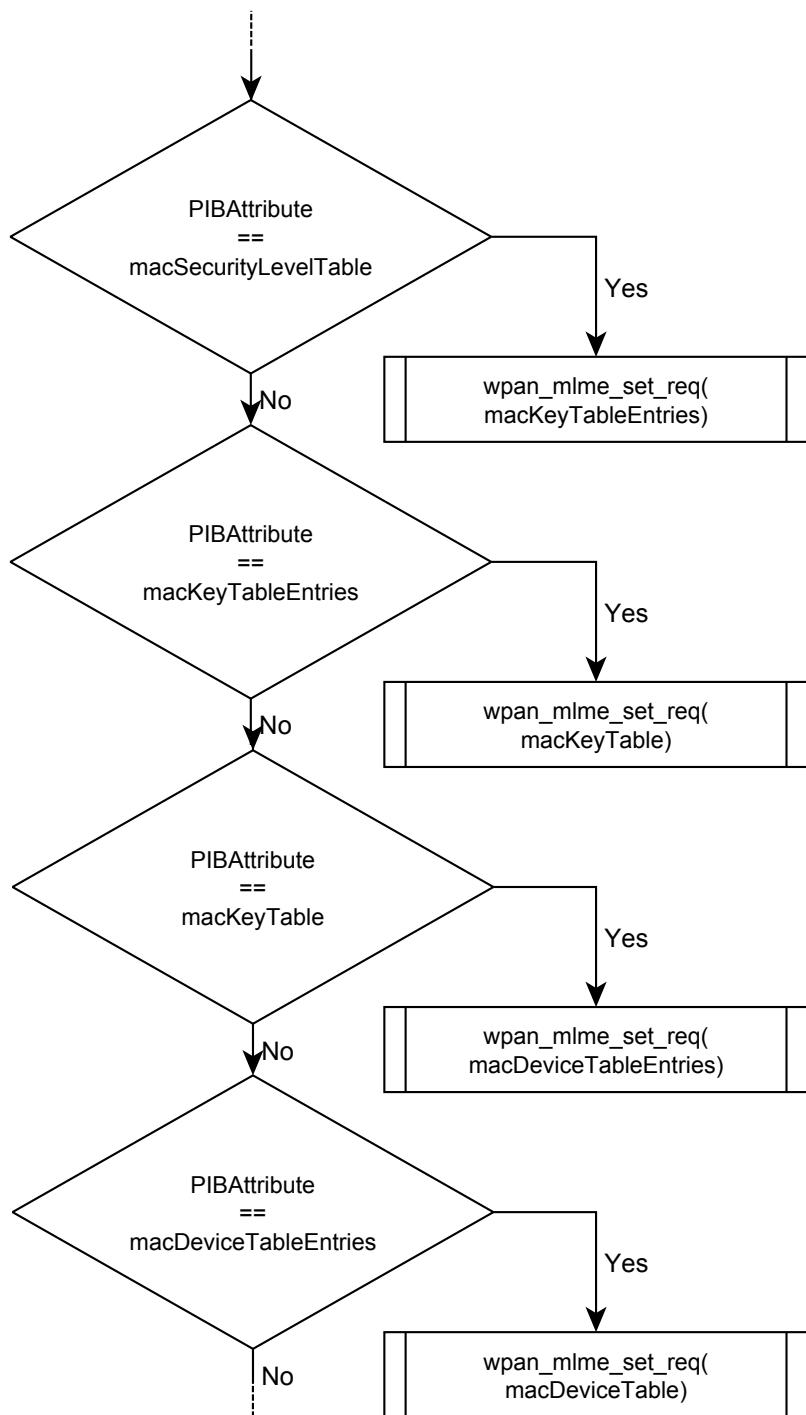


Abbildung C.5.: Device: `usr_mlme_set_conf()`

`usr_mlme_set_conf() (3/3)`

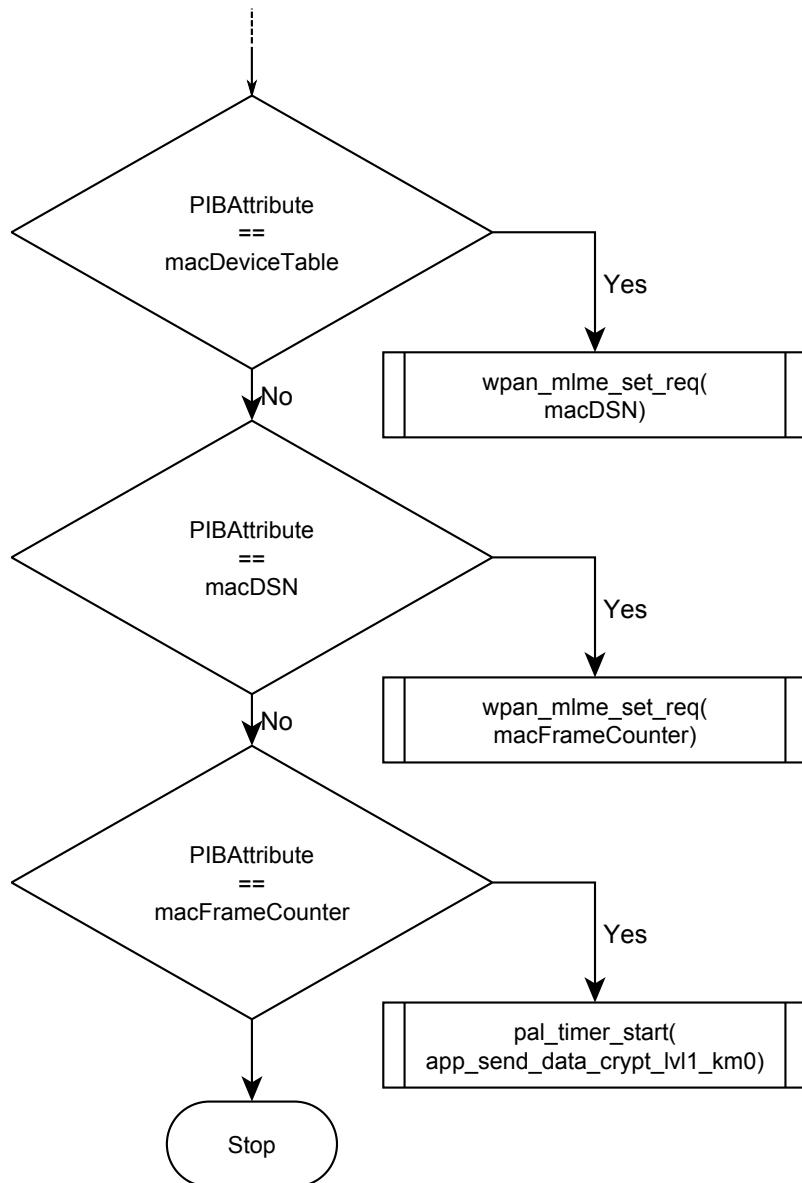


Abbildung C.6.: Device: `usr_mlme_set_conf()`

### C. Programm-Ablauf-Plan

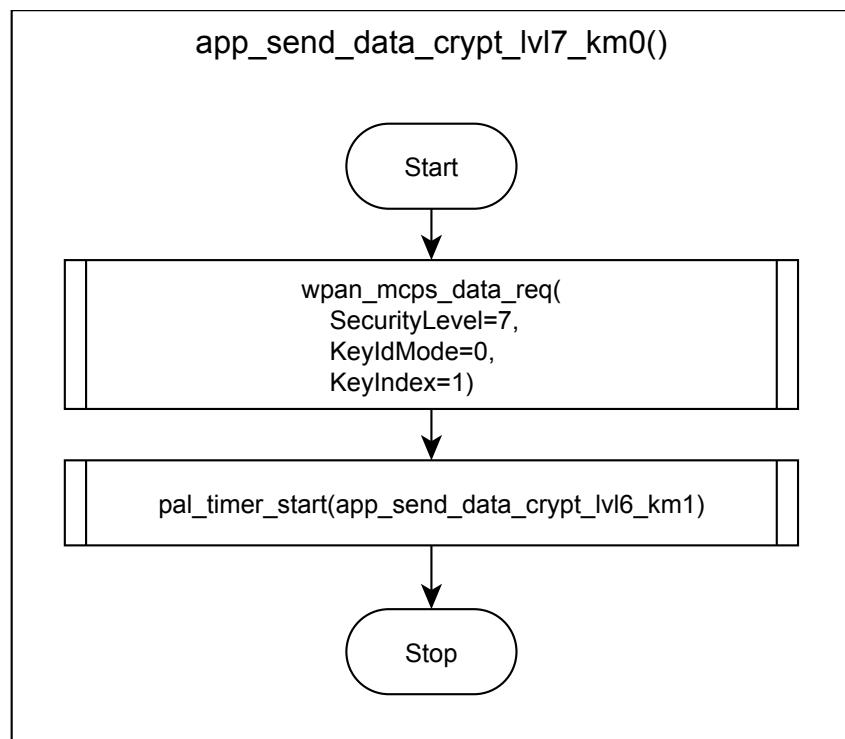
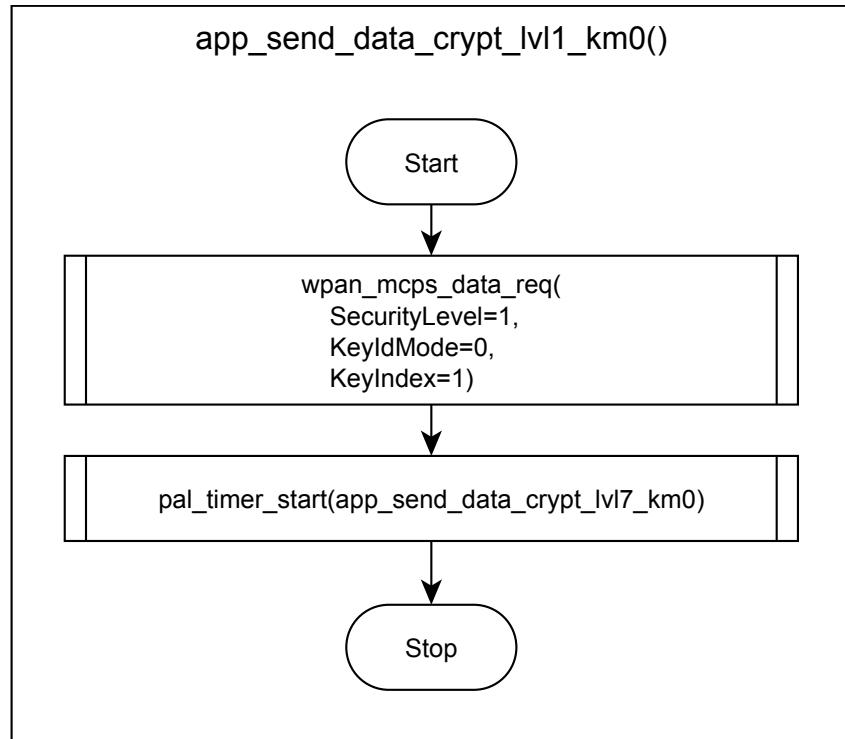


Abbildung C.7.: Device:  
app\_send\_data\_crypt\_lvl1\_km0(),  
app\_send\_data\_crypt\_lvl7\_km0()  
98

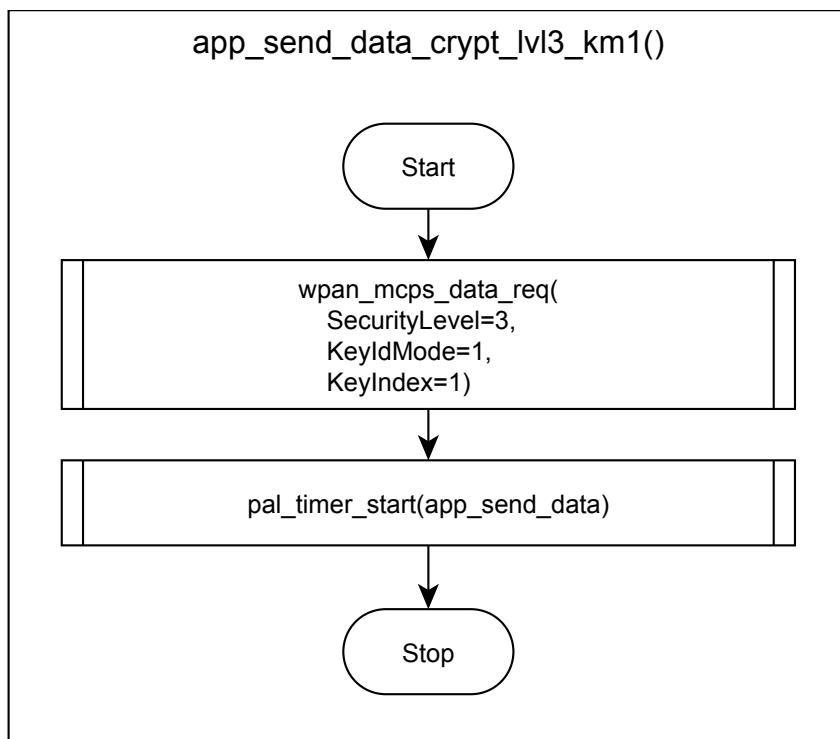
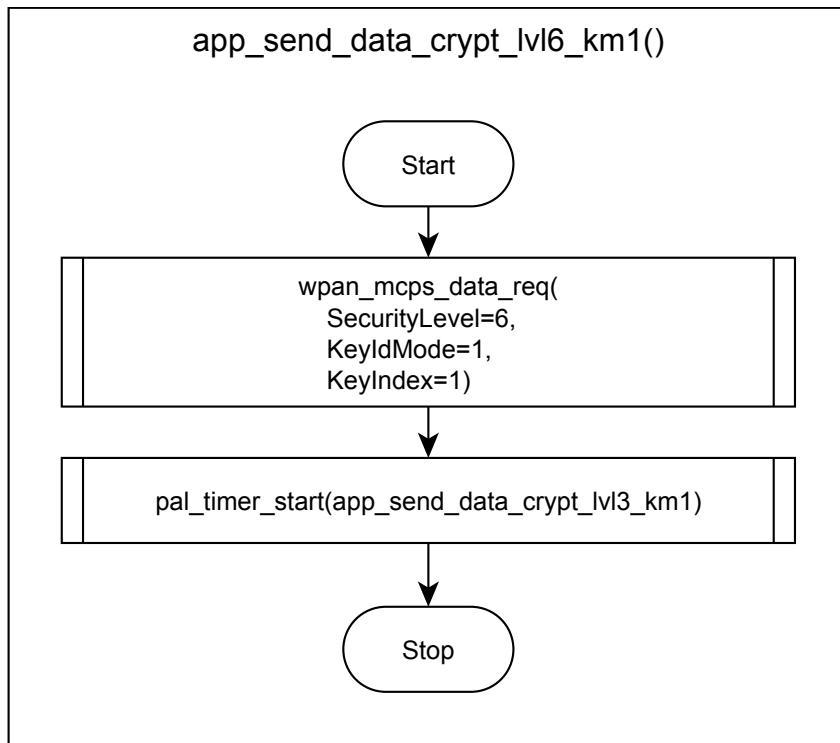


Abbildung C.8.: Device: `app_send_data_crypt_lvl6_km1()`,  
`app_send_data_crypt_lvl3_km1()`

### C. Programm-Ablauf-Plan

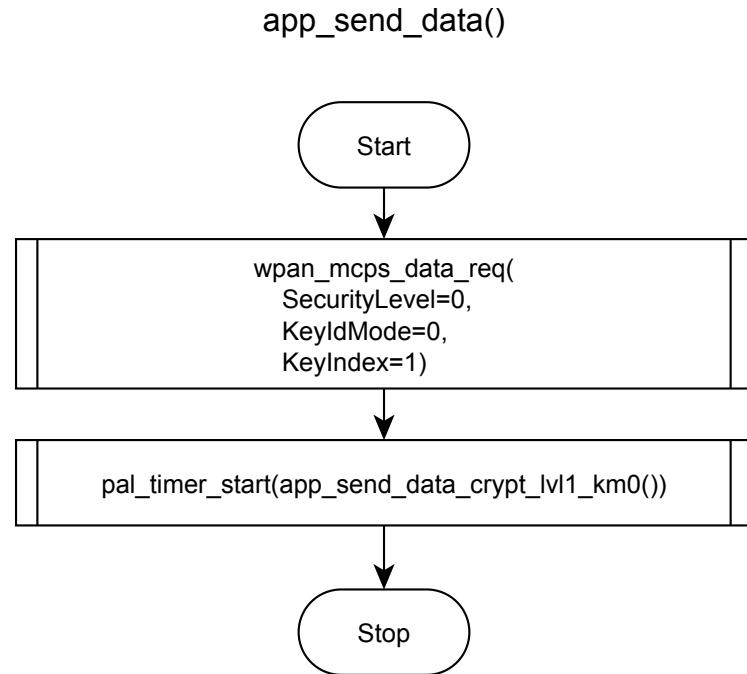


Abbildung C.9.: Device: app\_send\_data()

## C.2. Coordinator

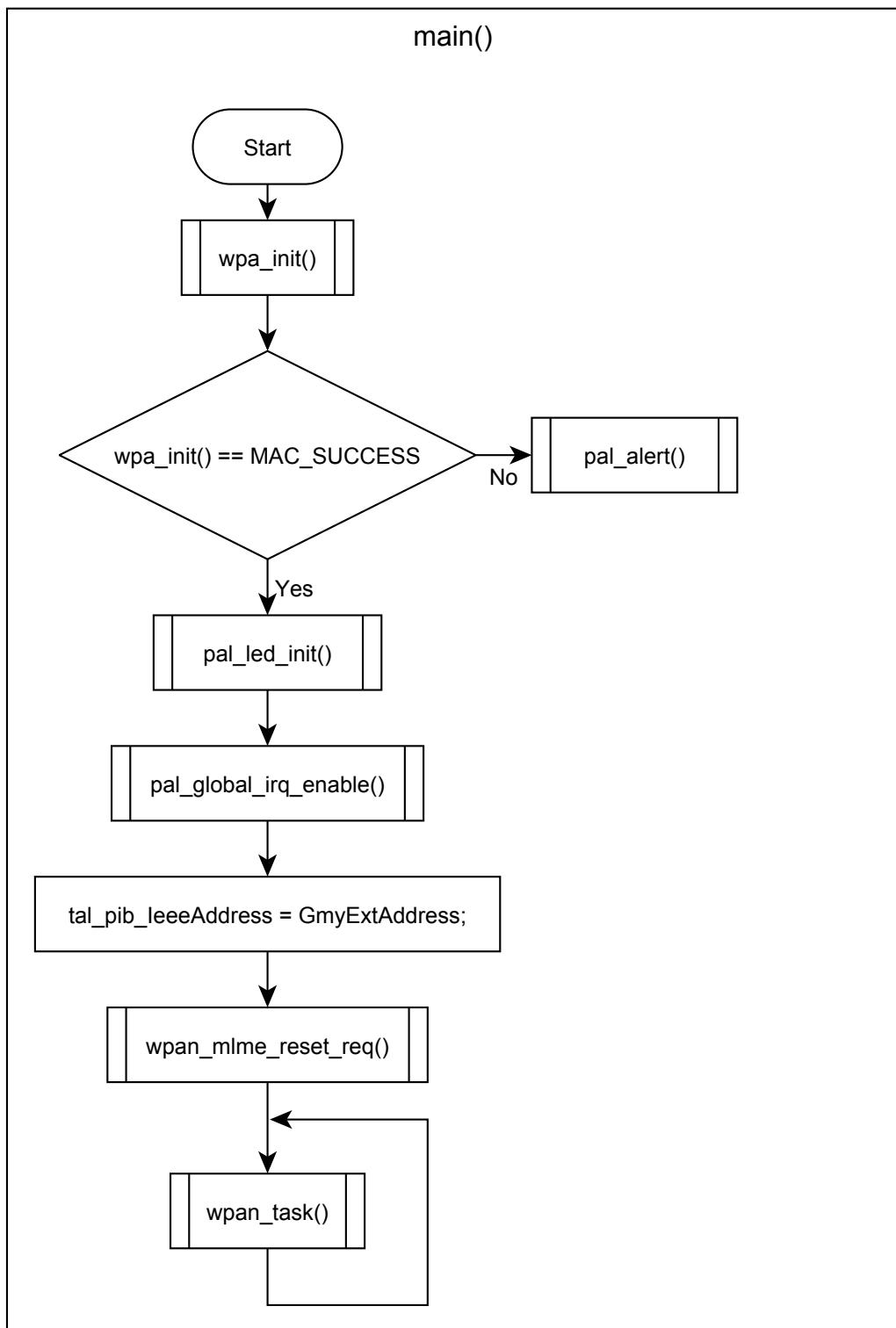


Abbildung C.10.: Device: main()

### C. Programm-Ablauf-Plan

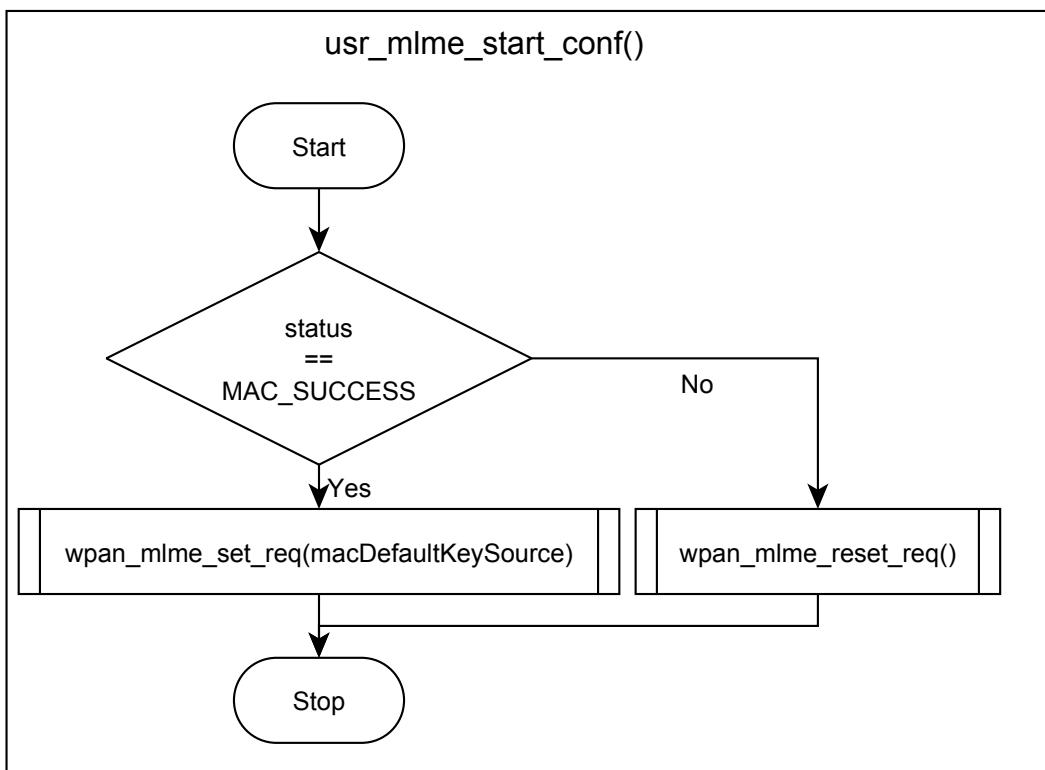
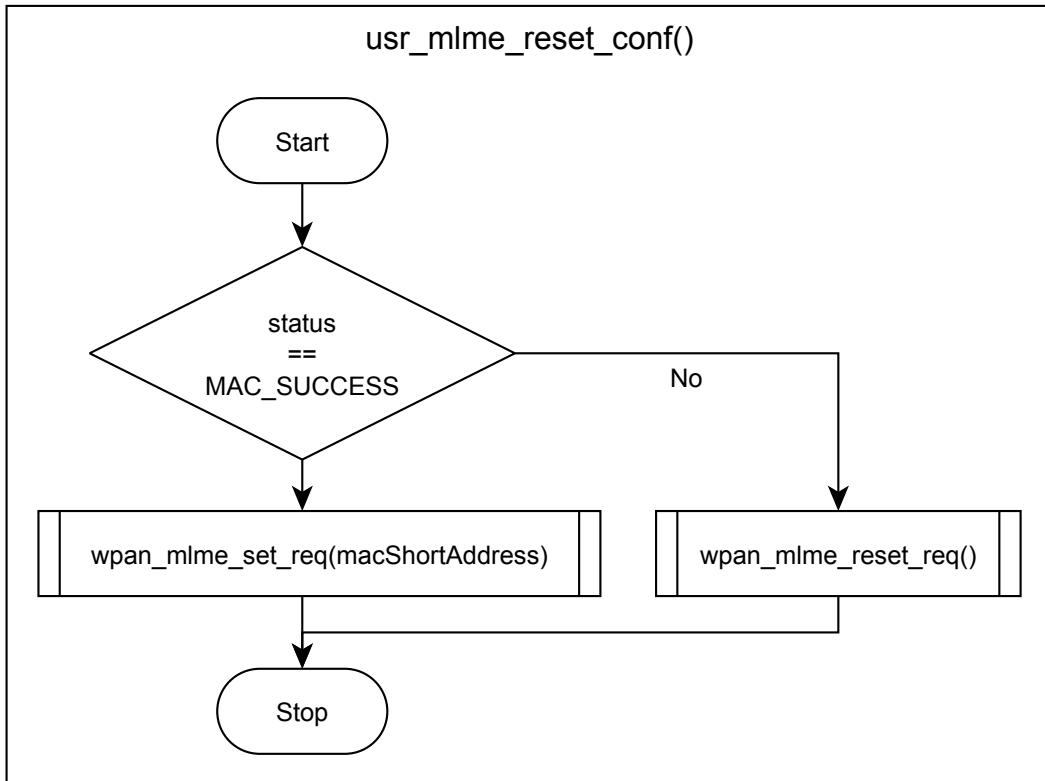


Abbildung C.11.: Coordinator: `usr_mlme_reset_conf()`, `usr_mlme_start_conf()`

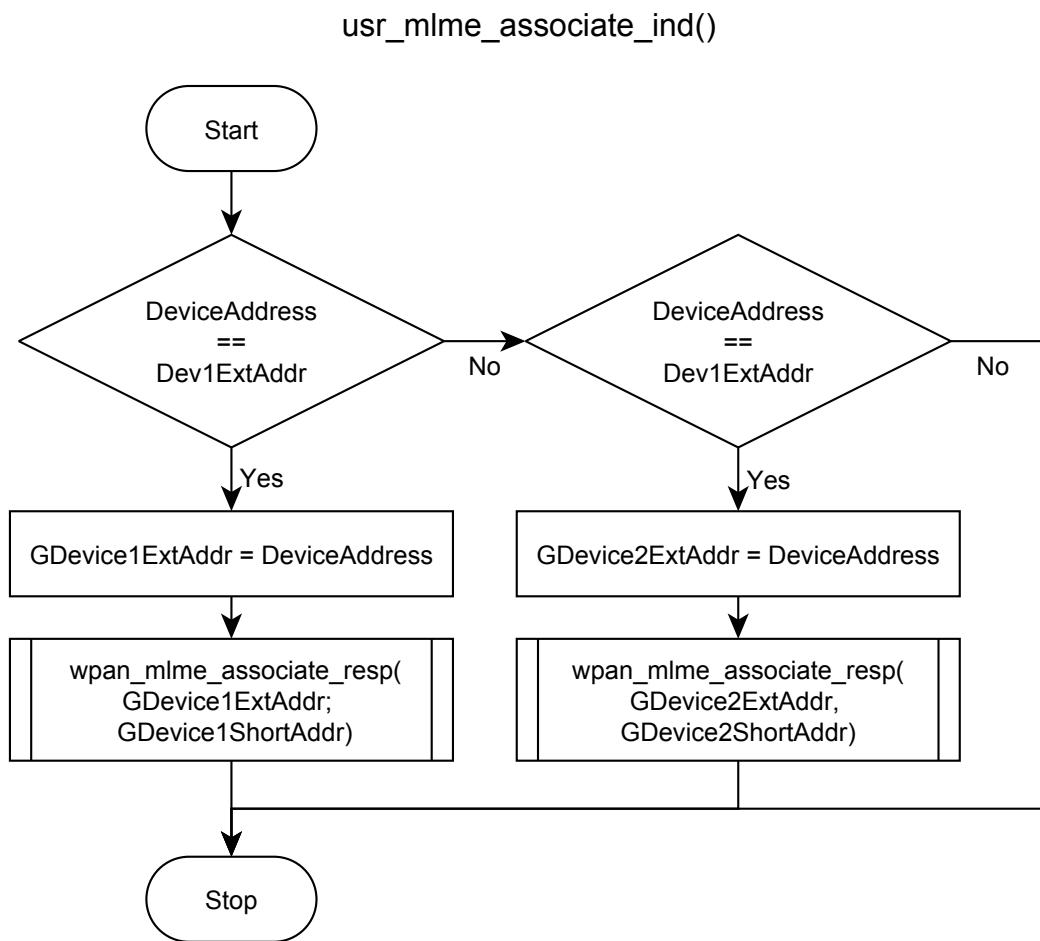


Abbildung C.12.: Coordinator: usr\_mlme\_associate\_ind()

### C. Programm-Ablauf-Plan

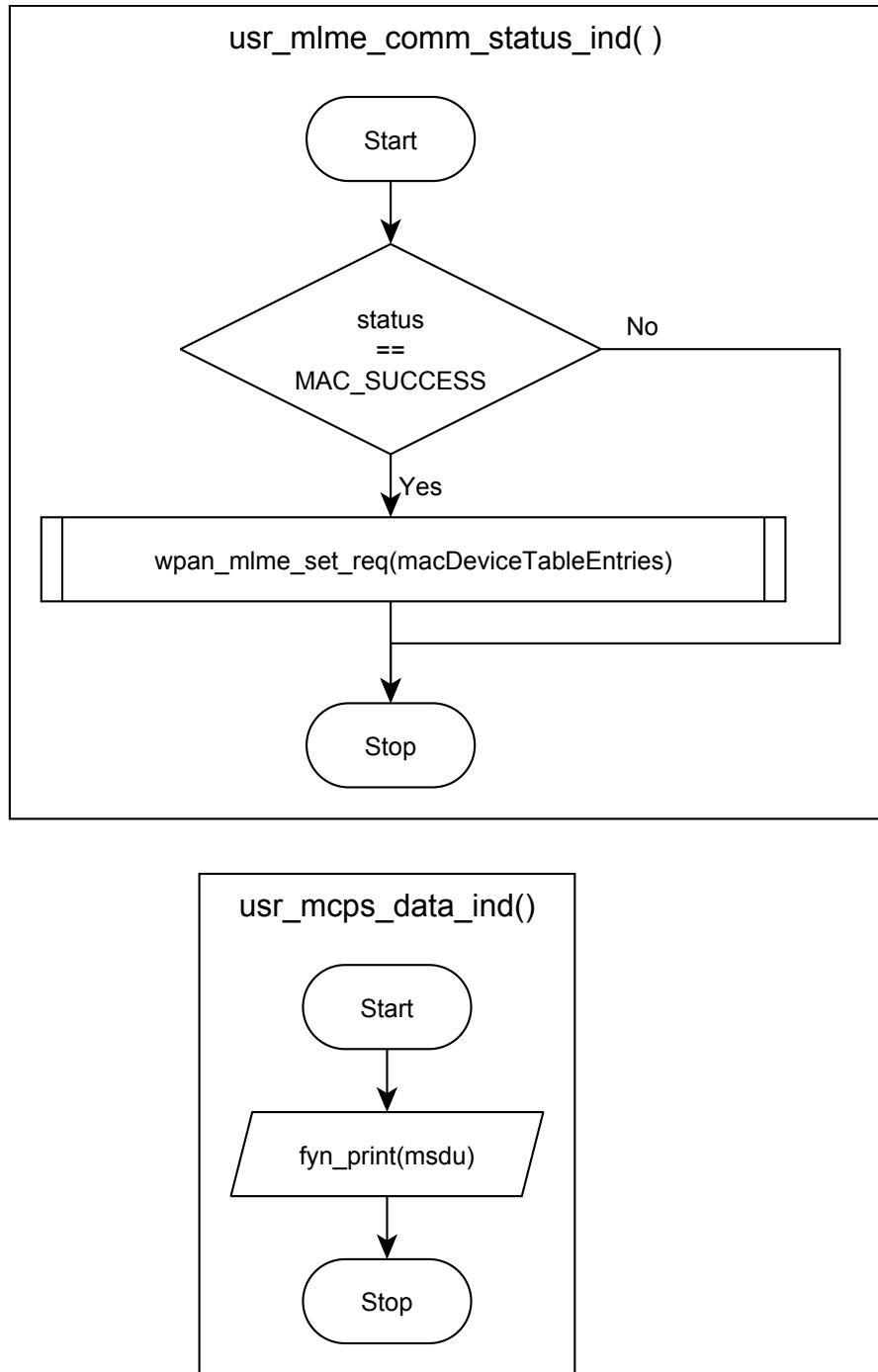
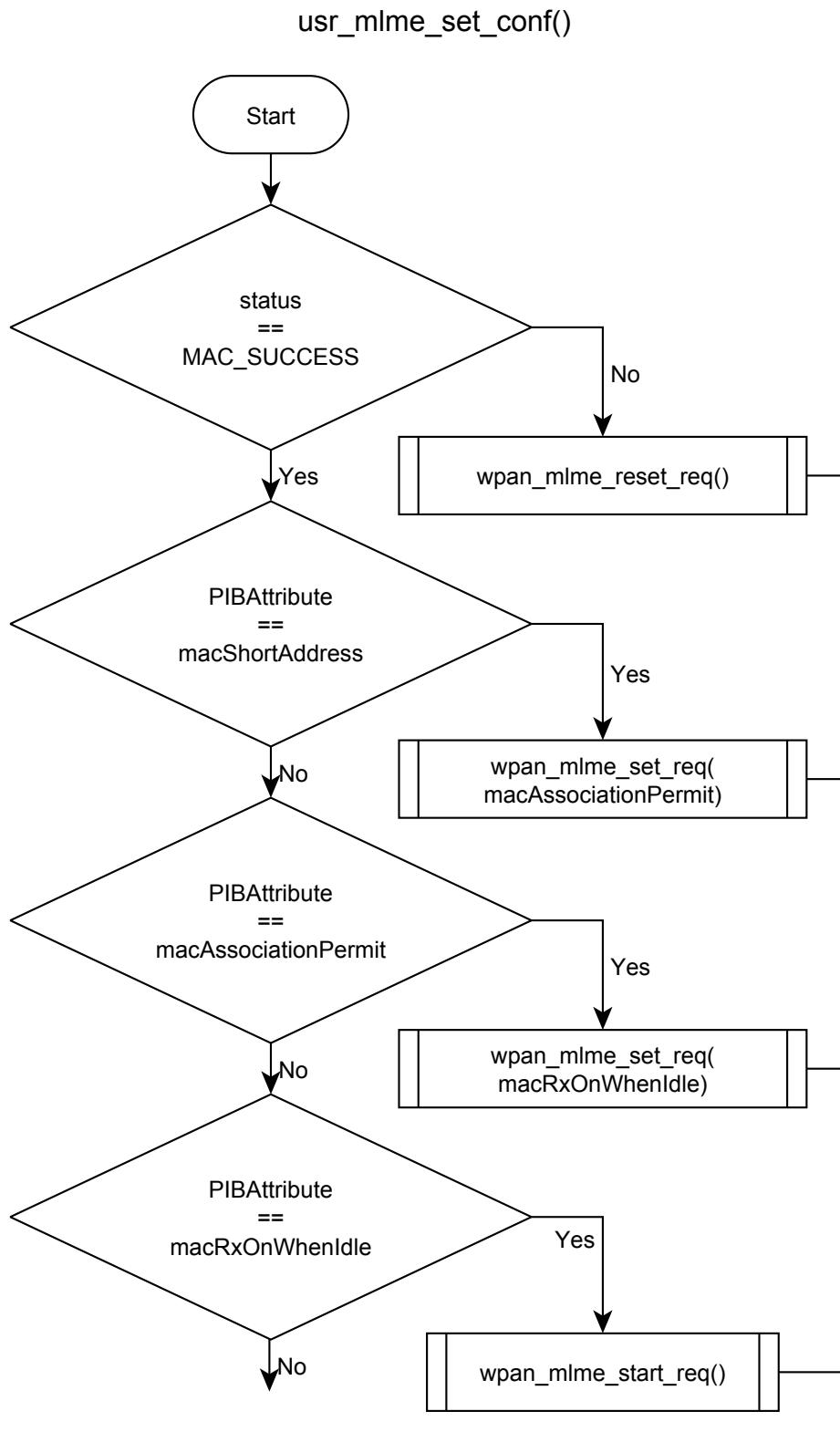


Abbildung C.13.: Coordinator: `usr_mlme_comm_status_ind()`, `usr_mcps_data_ind()`

Abbildung C.14.: Coordinator: `usr_mlme_set_conf()`

### C. Programm-Ablauf-Plan

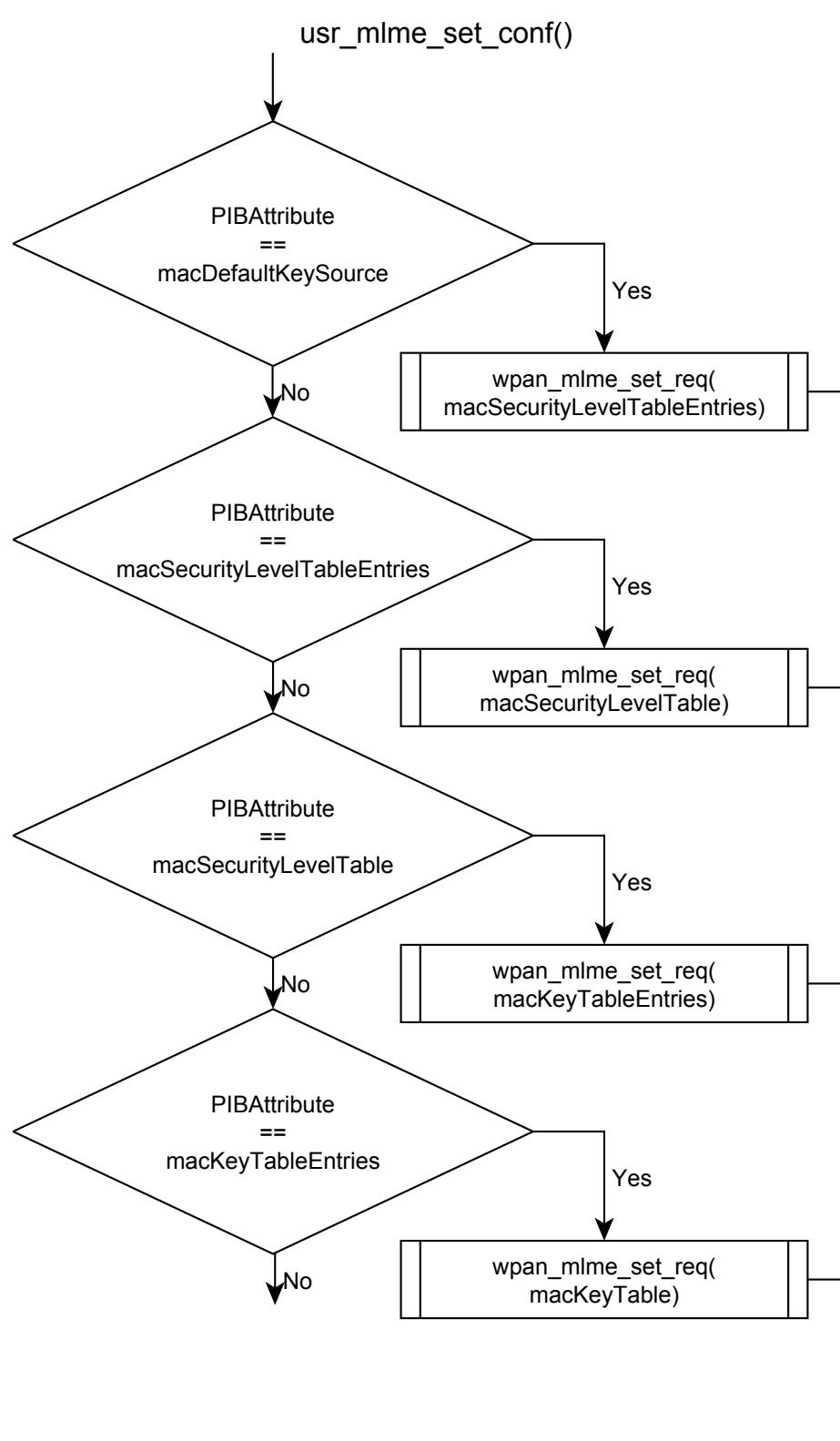
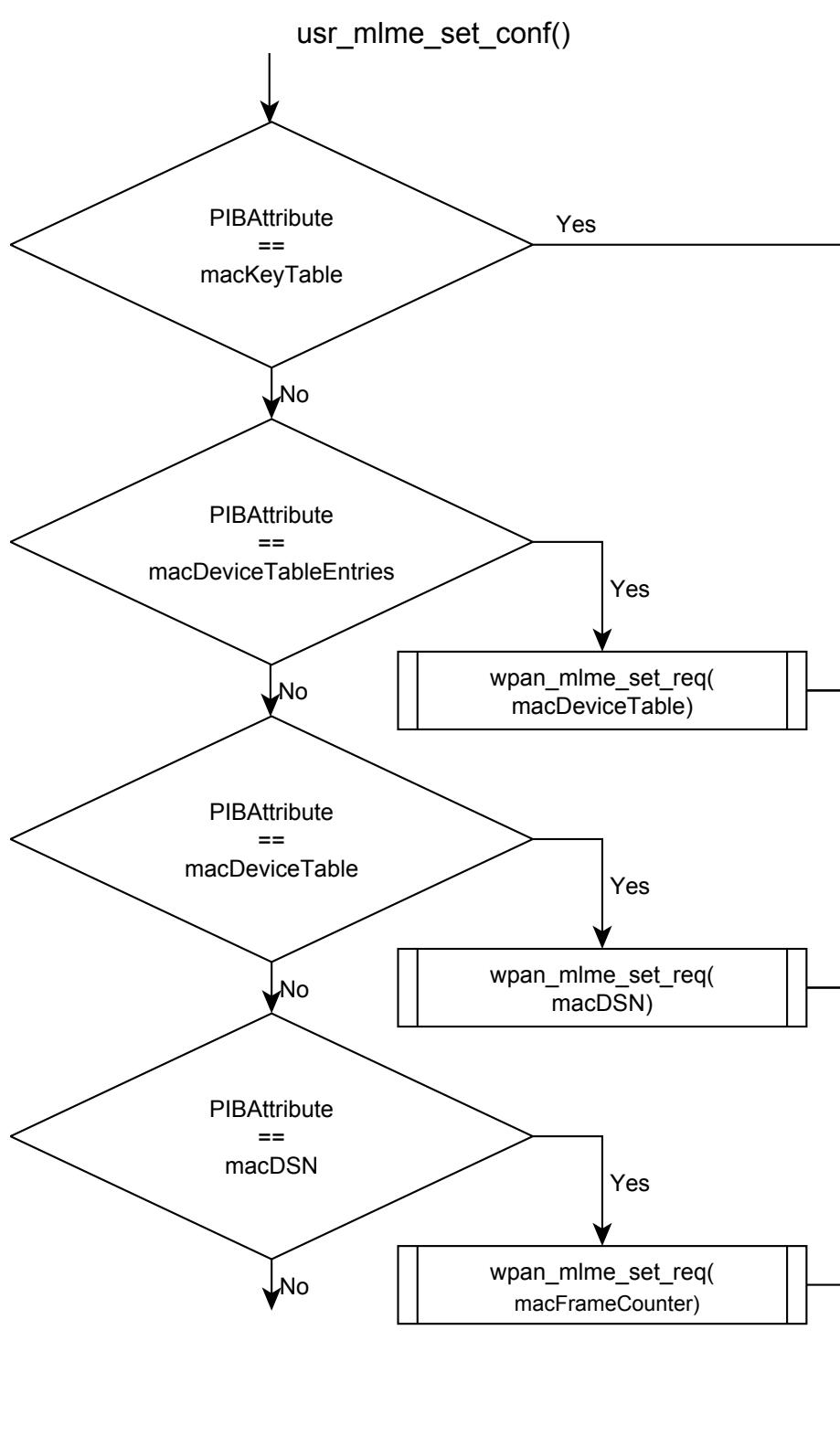


Abbildung C.15.: Coordinator: `usr_mlme_set_conf()`

Abbildung C.16.: Coordinator: `usr_mlme_set_conf()`

C. Programm-Ablauf-Plan

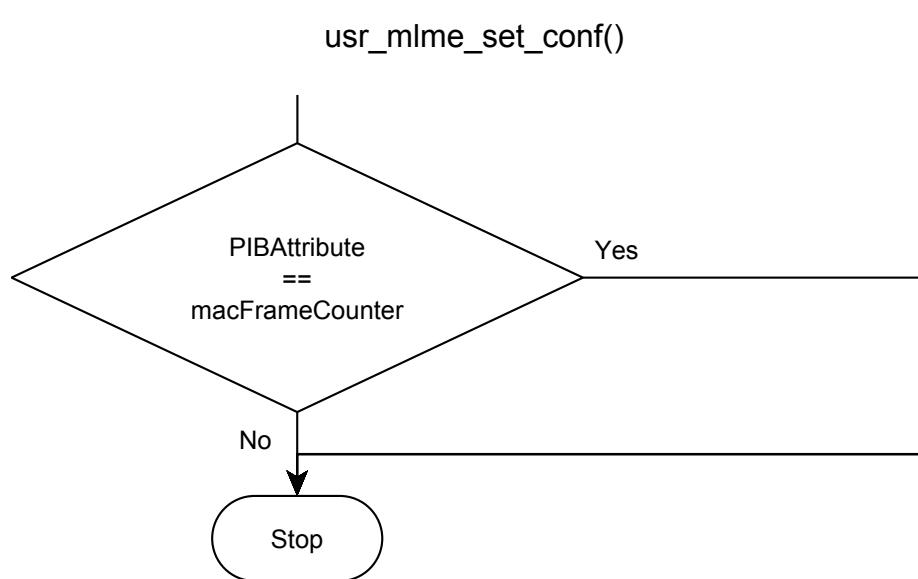


Abbildung C.17.: Coordinator: `usr_mlme_set_conf()`

# FORMELVERZEICHNIS

$U_q$	in [V]	Spannung, die die Spannungsquelle abgibt
$R$	in [ $\Omega$ ]	Messwiderstand
$U_R$	in [mV]	Spannung, die über dem Messwiderstand $R$ abfällt
$U_V$	in [mV]	Spannung am Ausgang des Verstärkers und damit die verstärkte Spannung $U_R$ , die vom Oszilloskop gemessen wird
$I$	in [mA]	Strom, der durch den Messwiderstand und das Netzwerkgerät fließt
$M$	in [Byte]	Länge des MIC
$L$	in [Byte]	Länge des <i>Length Field</i>
$M'$	in [Byte]	Transformierte Länge des MIC, sodass es im IV Platz findet
$L'$	in [Byte]	Transformierte Länge des <i>Length Field</i> , sodass es im IV Platz findet
$k$		Der zur Verschlüsselung genutzte Schlüssel
$n$		Anzahl der zu verschlüsselnden Blöcke
$m$		<i>Message</i> - Teil des Payloads, der durch eine MAC und durch eine Verschlüsselung geschützt wird
$a$		<i>Authentication Data</i> - Teil des Payloads, der nur durch eine MAC geschützt, jedoch nicht verschlüsselt wird
$N$		Nonce - Teil des IVs, der pro Verschlüsselung nur einmal benutzt wird
$U$		Verschlüsselter <i>Authentication Value</i> (MAC)



# ABBILDUNGSVERZEICHNIS

2.1.	Eingabe zur Berechnung der MAC/MIC . . . . .	12
2.2.	Struktur eines symmetrischen Chiffres . . . . .	15
2.3.	Struktur eines asymmetrischen Chiffres . . . . .	16
2.4.	Ablauf der Verschlüsselung mit Hilfe eines Blockchiffre im ECB-Mode . .	18
2.5.	Ablauf der Entschlüsselung mit Hilfe eines Blockchiffre im ECB-Mode . .	19
2.6.	Ablauf der Verschlüsselung mit Hilfe eines Blockchiffre im CBC-Mode . .	20
2.7.	Ablauf der Entschlüsselung mit Hilfe eines Blockchiffre im CBC-Mode . .	21
2.8.	der Aufbau des CCM-Mode durch anderer Verschlüsselungsmodi und Komponenten . . . . .	22
2.9.	Die Eingaben, Parameter und Ausgaben des CCM-Mode . . . . .	23
2.10.	Links: Aufbau des IV im CBC-MAC-Mode, Rechts: Aufbau der Blöcke für den Keystream im CTR-Mode . . . . .	24
2.11.	Die verschiedenen Topologien, in denen ein Netzwerk aufgebaut werden kann. . . . .	26
3.1.	Aufbau des Netzwerks, welches getestet und untersucht werden soll . . . .	33
4.1.	Aufbau der Bezeichnungen von den Funktionen, mit denen der ATMEL MAC Stack programmiert werden kann. . . . .	38
4.2.	Grundstruktur des MAC Stack sowie das Zusammenspiel zwischen den äußereren Schnittstellen (SAP) und den inneren Komponenten. . . . .	41
4.3.	Aufbau des ATMEL MAC Stack und Einordnung in die Layer-Struktur .	43
4.4.	Aufbau derNonce, wie er im ZigBee-Standard vorgesehen ist. . . . .	48
4.5.	Die einzelnen Zustände, die der Coordinator in seinem Betrieb durchläuft	50
4.6.	Die zeitliche Abfolge der Kommunikation zwischen MAC Stack und der Applikation sowie zwischen End Device und Coordinator . . . . .	51
4.7.	Die einzelnen Zustände, die das <i>End Device</i> in seinem Betrieb durchläuft .	53
4.8.	Aufbau eines Datenframes, welches verschlüsselt übertragen wird. . . . .	55
4.9.	Aufbau der Schaltung zur Messung des Energieverbrauchs . . . . .	64

## Abbildungsverzeichnis

# TABELLENVERZEICHNIS

4.1. Die Security-Levels . . . . .	47
------------------------------------	----



# LITERATURVERZEICHNIS

- [1] ATMEL. *ATmega128RFA1 datasheet*. ATmega128RFA1datasheet. Zugriff am:: 29.07.2012.
- [2] ATMEL. *Atmel AVR2025: IEEE 802.15.4 MAC Software Package - User Guide*. <http://www.atmel.com/Images/doc8412.pdf>. Zugriff am:: 05.06.2012.
- [3] ATMEL. *Atmel AVR2050: Atmel BitCloud Developer Guide*. <http://www.atmel.com/images/doc8199.pdf>. Zugriff am:: 07.06.2012.
- [4] ATMEL. *Atmel AVR2052: Atmel BitCloud Quick Start Guide*. <http://www.atmel.com/images/doc8200.pdf>. Zugriff am:: 07.06.2012.
- [5] ATMEL. *IEEE 802.15.4 MAC User Guide*. <http://www.atmel.com/Images/doc5182.pdf>. Zugriff am:: 02.06.2012.
- [6] Rolf Blom. *LNCS 0209 - An Optimal Class of Symmetric Key Generation Systems*. <http://www.csl.mtu.edu/cs6461/www/Reading/blom-eurocrypt84.pdf>. Zugriff am:: 20.07.2012.
- [7] Matthias Hofherr. *WLAN-Sicherheit - professionelle Absicherung von 802.11-Netzen*. Heise, 2005.
- [8] D. Whiting; Hifn; R. Housley; Vigil Security; N. Ferguson; MacFergus. *Counter with CBC-MAC (CCM)*. <http://tools.ietf.org/html/rfc3610>. Zugriff am:: 02.08.2012.
- [9] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Standard for Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements-Part 15.4: Wireless MAC and PHY Specifications for Low-Rate WPANs*. <http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>. Zugriff am:: 18.06.2012.
- [10] Bruce Schneier. *Angewandte Kryptographie - Protokolle, Algorithmen und Sourcecode in C: der Klassiker*. Pearson Education, 2006.

## Literaturverzeichnis

- [11] Prof. Dr.-Ing. Axel Sikora. *ZigBee: Grundlagen und Applikation.* [http://www.stzedn.de/standards.html?file=tl\\_files/files/stz\\_zigbee\\_elektronik\\_wireless\\_0401.pdf](http://www.stzedn.de/standards.html?file=tl_files/files/stz_zigbee_elektronik_wireless_0401.pdf). Zugriff am:: 07.05.2012.
- [12] Claudia Eckert Thomas Stibor. *Kryptographie - Kapitel 5: Designziele in Blockchifren.* <http://www.sec.in.tum.de/assets/lehre/ss09/kryptographie/Kapitel.5.pdf>. Zugriff am:: 20.06.2012.
- [13] Reinhard Wobst. *Abenteuer Kryptologie . Methoden, Risiken und Nutzen der Datenverschlüsselung.* Addison-Wesley, 1. aufl. edition, 2001.
- [14] ZIGBEE. *ZigBee Specification.* <http://www.zigbee.org/Specifications/ZigBee/download.aspx>. Zugriff am:: 20.06.2012.

# INDEX

- Bruteforce Attack, 31  
AES, 17  
CBC-MAC, 20  
CBC, 18  
CCM\*, 17  
CCM, 17  
ECB, 18  
IV, 19  
MAC, 20, 31  
MCPS, 39  
MLME, 39  
Payload, 30  
PIB, 57  
SAP, 39, 57  
SAPs, 27  
  
Aktive Angriffe, 30  
Application Support Sublayer - APS, 67  
APS-Frame, 67  
asymmetrische, 16  
Asymmetrische Verschlüsselung, 34  
asymmetrische Verschlüsselung, 31  
asymmetrischen, 16  
asymmetrischen Kryptosystemen, 16  
asynchron, 39  
Authentication Data, 107  
Authentication Value, 107  
authentification field, 23  
  
Authentifizierung, 31  
Auxiliary Security Header, 77  
Auxillary Security Header, 53  
  
Beacon-Frame, 60  
Beacon-Request-Frame, 60  
Blockchiffre, 15  
Blom-Verfahren, 82  
  
CBC-MAC, 47  
CBC-MAC-Mode, 22, 24, 25, 47, 72  
CBC-Mode, 20, 21, 49  
CCM/CCM\*, 22–24, 47–49, 65, 73, 77  
CCM/CCM\*-Mode, 22, 42, 45, 48, 72, 73, 78, 81  
Cold-boot-Attack, 82  
Confirm, 60  
Confirmation, 40  
Coordinator, 66  
CTR-Mode, 20, 22, 24, 72, 77, 78  
  
ECB-Mode, 20  
Elliptic Curve Cryptography - ECC, 81  
End Device, 34, 37, 52, 53, 66–68  
Enddevices, 66  
eSTREAM, 15, 17  
Explicit Key Identification, 53  
Extended Address, 25, 27, 37, 44–47, 52, 59, 67, 68, 71, 72, 78, 79

## *Index*

- Finite State Machine, 38, 49
- Frame, 27
- Frame Check Sequence, 55
- Framecounter, 71
- Hashwert, 12
- HDCP, 82
- High Security, 68
- hybriden, 16
- hybriden Verfahren, 16
- IEEE 802.15.4, 17, 22, 29, 45
- Indication, 40
- Integrität, 31
- IV, 20
- Key Index, 53, 58, 77, 78
- KeyIdMode, 62
- KeyIndex, 62
- Keyindex, 78
- Keymode, 53, 62, 78
- Kryptographischen Hashfunktion, 12
- Lawineneffekt, 20
- Length Field, 107
- Lengthfield, 23, 24
- Link Key, 55, 67, 79
- Link key, 67
- Link Keys, 70, 79
- MAC Footer, 55
- MAC Header, 80
- MAC Layer PAN Information Base, 39
- Man-in-the-middle-Attack, 34
- Master Key, 70, 79
- Message, 23, 107
- Message Integrity Check - MIC, 48
- MIC, 31
- MIC-Protection, 55
- National Institute of Standards and Technology, 17
- Network Key, 44, 55, 66, 67, 79
- Network Keys, 79
- Network Layer - NWK, 66, 67
- Networkkey, 67
- Nonce, 20, 22
- OSI Schichtenmodells, 39
- Padding, 15
- Passiven Angriffen, 30
- Payload, 77
- Precomputation Attack, 25
- Precomputation-Attack, 71
- radio controller, 48
- Rainbow Table, 25
- Receiver, 57
- Replay-Attack, 31, 47
- Replay-Attacks, 45, 71
- Request, 40, 60
- Response, 40
- Rijndael, 17
- Security Abstraction Layer, 55
- Security Abstraction Layer - SAL, 49
- Security Control Field, 53
- Security Tool Box, 55
- Security Toolbox - STB, 47
- Security-Level, 53
- Securitylevel, 62, 71
- Securitylevels, 61
- Shared Medium, 8, 29, 30
- Short Address, 27, 33, 45, 46, 52, 57, 58, 65
- Standard security, 68
- Standard security with link Keys, 68
- Stromchiffre, 15
- Subnetwork, 27
- Symmetric-Key Key Establishment - SKKE, 68
- symmetrisch, 17
- symmetrische, 16
- Symmetrische Verschlüsselung, 34
- symmetrische Verschlüsselung, 15, 16
- symmetrischen Verschlüsselung, 16, 31
- Symmetrischer Verschlüsselung, 34
- Trust Centers, 78, 79

Trustcenters, 66  
Verschlüsselung, 31  
WEP, 31, 72  
WSNDemo, 66  
ZigBee, 17, 22, 25, 29



# GLOSSAR

**avr-gcc** Der C-Compiler des Projekts GCC, der C-Quellcode in maschinenlesbaren Code übersetzen kann, welcher auf einem ATMEL AVR Chip lauffähig ist..

**Beacon-Packet** Hierbei handelt es sich um ein Paket im Netzwerk, welches mit geringer Größe schnell und periodisch gesendet wird..

**Broadcast-Paket** Hierbei handelt es sich um ein Paket im Netzwerk, welches bestimmte Informationen enthält und an alle Netzwerkknoten gesendet wird..

**Bruteforce Attack** Ist ein Angriff auf eine Verschlüsselung, indem ganz einfach jedes mögliche Passwort ausprobiert wird. Dabei kommen alle möglichen Kombinationen in Frage, die im Schlüsselraum liegen, sprich: Der Schlüssel wird geraten. Je größer der Schlüsselraum ist, desto mehr Passwörter müssen ausprobiert werden, was dementsprechend länger dauert. Die Sicherheit einer Verschlüsselung beruht nun darauf, dass mit der verfügbaren Hardware in keinem annehmbar kurzen Zeitraum alle Schlüssel durchprobieren können. Damit stellt dieser Angriff die obere Schranke an Aufwand zum Brechen eines Chiffre dar. Alle anderen Angriffe versuchen diesen Aufwand zu minimieren..

**Challenge Response** Um herauszufinden, ob ein Teilnehmer wirklich jener ist, der er vorgibt zu sein, stellt man ihm eine Aufgabe, die er nur lösen kann, wenn er bestimmte geheime Informationen kennt, die sonst keiner kennt. Verschlüsselt man beispielsweise eine Information und der Gegenüber ums auf sie Reagieren oder sie in bestimmter Weise verändern, dann kann er dies nur tun, wenn er ebenfalls den Schlüssel kennt. Dieses Verfahren wird Challenge Response genannt..

**Cipher Block Chaining Mode (CBC)** Ein Modus um einen Blockchiffre auf Datenblöcke anzuwenden, die größer sind als die von ihm unterstützte Blockgröße mit dem Vorteil, dass vorhergehende Blöcke in die Chiffrierung mit einbezogen werden..

**Cold-Boot-Attack** Der Speicherinhalt von flüchtigem Speicher kann sich durch dessen Kühlung länger halten als gewohnt und verschwindet damit nicht nach dem Ausschalten. Diesen Speicherbaustein kann man nun an ein anderes System anschließen und auslesen. So kann man z.B. an geheimes Schlüsselmaterial gelangen, welches im Speicher zum Arbeiten gehalten wird..

**Electronic Code Book Mode (ECB)** Ein Modus um einen Blockchiffre auf Datenblöcke anzuwenden, die größer sind als die von ihm unterstützte Blockgröße.

**Image** Das Image ist ein Datenformat, welches aus dem vom avr-gcc generierten Code erstellt wird, und direkt in eine spezielle Speicherstelle eines Netzwerkgerätes geschrieben wird. Von dort aus kann es von diesem ausgeführt werden..

**Key Stream** Eine Möglichkeit, einen Text zu verschlüsseln, besteht darin den Text mit einem Schlüssel XOR zu verknüpfen. Dieser Schlüssel muss genauso lang sein wie der Text. Mit Hilfe eines Blockchiffre in einem speziellen Modus oder Stromchiffre wird der Schlüssel erstellt und Keystream genannt. Genügt der Keystream bestimmten Anforderungen, wie z.B. starke Zufälligkeit, wird dieses Verfahren als One Time Pad bezeichnet..

**Klartext** Ein Klartext ist ein Datensatz in unverschlüsselter Form..

**Kryptographische Hash-Funktion** Mithilfe dieser Klasse von Funktionen können Prüfsummen generiert werden, die charakteristisch für einen Datensatz sind. Diese Prüfsumme wird Hash-Wert genannt. *Kryptographisch* bedeutet in diesem Zusammenhang, dass nach Möglichkeit keine zwei Datensätze gefunden werden können, die den selben Hash-Wert ergeben. Sollte dies dennoch passieren, spricht mensch von einer Kollision..

**Man-In-The-Middle-Attack** Bauen zwei Parteien eine Verbindung auf, kann sich eine dritte Partei der ersten als die zweite und der zweiten als die erste Partei ausgeben und erhält in beiden Fällen die Kommunikationsschlüssel. Jede der zwei Parteien glaubt mit der jeweils anderen Partei zu kommunizieren, spricht aber mit der dritten Partei. So kann der gesamte Datenaustausch mitgehört werden..

**Nonce** Eine Nonce ist eine Art Zahlenvektor, der beliebige Werte annehmen kann und nach seiner Verwendung wieder verworfen wird. Im Kontext von bestimmten Sicherheitsmechanismen muss diese jedoch einigen Ansprüchen genügen, wie z.B. dass sie nicht wiederverwendet werden darf. Man kann sie mit einer Email-Wegwerf-Adresse vergleichen, die man benötigt, um sich irgendwo zu registrieren. Ist dieser Registrierungsprozess vorbei, braucht man die Email-Adresse nicht mehr..

**Padding** Bei der Verschlüsselung eines Klartexts mithilfe eines Blockchiffres muss der Datensatz in Blöcke fester Größe aufgeteilt werden, die vom Blockchiffre unter-

stützt wird. Bleibt dabei ein Block geringerer Größe übrig, muss dieser mit Daten aufgefüllt werden, bis die feste Größe erreicht wird..

**Payload** Der Payload ist der Teil der übertragenen Daten eines Frames, der aus den Nutzdaten besteht..

**Precomputation-Attack** Ist an einer Verschlüsselung ein IV beteiligt, wie z.B. bei CBC-MAC, kann mittels einer Tabelle, in der mit verschiedenen Keys und der bekannten Nonce oder dem bekannten IV das Chiffraut vorausberechnet wird, durch Abgleich dieser mit dem zu brechenden Chiffraut der Key ermittelt. Dies funktioniert ähnlich wie mit Rainbowtables, um z.B. Hashfunktionen zu brechen..

**Replay-Attack** Auch wenn sämtliche Nachrichten mit einer MIC gesichert und durch Verschlüsselung vor dem Abhören geschützt ist, hindert dies einen Angreifer nicht daran, bereits abgefangene Pakete wieder einzuspielen. Der Empfänger wird durch eine MIC und der Verschlüsselung allein nicht unterscheiden können, ob er das Paket bereits erhalten hat und ob es richtig authentifiziert ist. Mit Hilfe eines Counters, der die Pakete durchnummeriert, kann er erkennen, ob das Paket bereits empfangen worden ist, da dessen Counter kleiner als der aktuelle des Empfängers sein wird. Im Standard WEP wurde diese Lücke ausgenutzt um zusätzlichen Datenverkehr zu provozieren und aus den gewonnenen Daten statistische Informationen zu gewinnen..

**Serial I/O support** Dies ist eine Komponente des MAC Stack, um mit Hilfe von UART nach außen hin Daten austauschen zu können..

**Universal Asynchronous Receiver Transmitter** Hierbei handelt es sich um eine digitale Schnittstelle, mit deren Hilfe über eine Signalleitung seriell Daten übertragen werden können. Eine Form ist die bekannte Serielle Schnittstelle an einem PC (RS-232). In eingebetteten Systemen wird durch einen Emulator diese Schnittstelle über USB ermöglicht, sodass diese Form der Hardware mit einem PC kommunizieren kann..

**Wired Equivalent Privacy** Dieses Verschlüsselungsprotokoll fand bei der Einführung des Standards IEEE 802.11 zur Bildung von WLANs Verwendung. Es beinhaltet eklatante Sicherheitsschwächen und gilt als das Standardbeispiel für Designfehler bei der Implementation von Sicherheitsmechanismen..