Bachelorarbeit

Low Latency-Kryptographie für transparente Layer 2 - Verschlüsselung mit MACsec

Gregor Garten

1. Oktober 2018

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Datenschutz und Datensicherheit

Betreuender Hochschullehrer: Prof. Dr. rer. nat. Thorsten Strufe
Betreuende Mitarbeiter: Dr.-Ing. Stefan Köpsell
Dipl.-Inf. Tim Lackorzynski

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Datenschutz und Datensicherheit

AUFGABENSTELLUNG FÜR DIE BACHELORARBEIT

Name, Vorname des Studenten: Garten, Gregor

Immatrikulationsnummer: 4055357

Studiengang: Informatik (2009)

Thema (deutsch): **Low-Latency-Kryptographie für transparente Layer-2 Verschlüsselung mit MACsec**

Thema (englisch): **Low-Latency Cryptographic Algorithms for MACsec providing transparent Layer-2 encryption**

Zielstellung:

MACsec (IEEE 802.1AE) ist ein Protokoll, daß eine Integritätsprüfung sowie Verschlüsselung des Datenverkehrs auf Schicht 2 ermöglicht. Dabei werden die zu übertragenden Ethernet-Pakete mittels MAC geschützt und gegebenenfalls verschlüsselt. Aktuell ist in MACsec jedoch nur ein Kryptoverfahren (AES-GCM) spezifiziert, welches dazu nur mit einer Schlüssellänge (128 bit) implementiert ist. Das Kryptoverfahren stellt die wesentliche und rechenaufwändigste (Gruppe von) Operation(en) dar und charakterisiert damit das gesamte Verfahren bezüglich seiner Effizienz und Anwendbarkeit auf verschiedene Anwendungsgebiete. Ziel ist es daher, die vorhandene Implementation von MACsec (Linux-Kernelmodul plus entsprechende Systemwerkzeuge) so zu erweitern, daß es möglich wird, weitere Kryptoverfahren in MACsec verwenden zu können. Dazu soll zuerst eine Auswahl verschiedener in Frage kommender Verfahren getroffen werden. Dies soll in Abhängigkeit des Anwendungsfalles (Kommunikation im industriellen Umfeld) und bereits vorhandener Implementationen geschehen. Zwei Abschlussarbeiten am Lehrstuhl beschäftigten sich bereits mit unterschiedlichen Low-Latency-Kryptoverfahren und deren Effizienz. Die Ergebnisse dieser Arbeiten sollen herangezogen werden. Nachdem MACsec mit den identifizierten Verfahren erweitert wurde, muss gezeigt werden, daß es immer noch aus funktionaler Sicht genau das leistet, was es soll. Ferner ist zu belegen, daß die Sicherheit nur gemäß des veränderten Kryptoverfahrens verändert wird. Darüber hinaus sollen die neuen Verfahren mit AES-GCM und gegeneinander bezüglich bestimmter Performanceparameter wie Latenz und Bandbreite verglichen werden. Dazu ist eine entsprechende Messinfrastruktur aufzusetzen. Für eine erfolgreiche Bearbeitung sind folgende Teilaufgaben zu erfüllen:

- Festlegung der zu verwendenden Kryptoverfahren
- Erweiterung der MACsec-Implementation sowie benötigter Systemwerkzeuge um die ausgewählten Kryptoverfahren
- Aufbau einer Messinfrastruktur für die Evaluation
- Bewertung der verwendeten Verfahren nach Funktionalität, Sicherheit und bestimmter Performanceparameter, wie Latenz und Bandbreite

Betreuer: Dr.-Ing. Stefan Köpsell

Verantwortlicher Hochschullehrer: Prof. Thorsten Strufe

Institut: Systemarchitektur

Beginn am: 4. Juni 2018 Einzureichen am: 20. August 2018

4.06.2018 .....

Datum, Unterschrift der/des Studierenden

.....

Unterschrift des betreuenden Hochschullehrers

Verteiler: Prüfungsamt (Original); Student (Original – Deckblatt der Bachelorarbeit); HSL + Sekr. (Kopie); elektr. Datei → Sekretariat

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Dresden, den 1. Oktober 2018

Gregor Garten

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Digitalisierung zählt zu den großen Herausforderungen der gegenwärtigen Zeit und auch für die Zukunft. Es erfordert umfangreiche Anpassungen unter anderem in der Industrie. In der Bachelorarbeit sollen für diese zentrale Aufgabe zunächst einige Grundlagen dargestellt, um dann Lösungswege zur Verbesserung untersucht und herausgearbeitet werden.
Es ist neben der Vernetzung von Systemen eines der Hauptziele der Industrie 4.0, die entstehenden Datenaufkommen zu sichern. Jedoch entstehen mit der zunehmenden Vernetzung von Komponenten innerhalb eines Unternehmens neue Möglichkeiten, um in ein Netzwerk einzudringen und Daten unerlaubt abzuschöpfen. Das fastvpn Projekt hat hierfür ein Sicherheitskonzept entwickelt, das sich diesem Problem widmet [Blu+18]. Hierfür kommunizieren sicherheitsrelevante Systeme mit FastVPN-Boxen, die eine sichere Kommunikationsübertragung bereitstellen können und als Vermittler zwischen den sicherheitsrelevanten Systemen fungieren. Mit der Anforderung der Echtzeitkommunikation in der Industrie 4.0 werden enorme Ansprüche an Netzwerksicherheitsstandards gesetzt. Diese müssen nicht nur die größtmöglichste Sicherheit gewährleisten, sondern auch geringe Auswirkungen auf die Übertragungszeit von Nachrichten haben. Daher wird die Performance von Netzwerkprotokollen zu einem entscheidenden Kriterium.
Eines der Netzwerkprotokolle, welches für die Sicherheit in den FastVPN-Boxen zuständig ist, ist der Institute of Electrical and Electronics Engineers (IEEE) Sicherheitsstandard Medium Access Control Security (MACsec). MACsec arbeitet auf der zweiten Schicht des Open Systems Interconnection Model (OSI-Modell) und nutzt den Betriebsmodus Authenticated Encryption with Associated Data (AEAD). Dadurch können gleich mehrere Sicherheitsziele, darunter Vertraulichkeit und Integrität durch nur einen Verschlüsselungsalgorithmus erreicht werden. Der verwendete Verschlüsselungsalgorithmus ist der Advanced Encryption Standard (AES), der im Galois Counter Mode (AES-GCM) betrieben wird. AES ist einer der meistgenutzten Verschlüsselungsalgorithmen, der in diversen Sicherheitsprotokollen benutzt wird. Durch die weite Verbreitung von AES findet sich in fast jedem Computer die Hardware Unterstützung Advanced Encryption Standard New Instructions (AES-NI) wieder, von der der Algorithmus zusätzlich profitiert. Die Verbreitung von AES bringt allerdings auch diverse Probleme mit sich. AES-GCM nutzt eine der rechenaufwändigsten Operationen und kann ohne Hardware Unterstützung nicht mit anderen neueren Verschlüsselungsalgorithmen mithalten. Mit Blick auf den Anwendungsfall der industriellen Kommunikation ist es daher sinnvoll, nach anderen Verschlüsselungsalgorithmen zu suchen, die möglicherweise eine bessere Performance erreichen können, da die Systeme im industriellen Bereich oft keine AES-NI Hardware Unterstützung beherbergen. Um eine effiziente Alternative zu besitzen, sollte der Verschlüsselungsalgorithmus nicht von der Hardware Unterstützung abhängig sein, um eine gute Performance zu erreichen. Diese Arbeit widmet sich dem Problemfall

und setzt sich das Ziel nach einem vielversprechenden Verschlüsselungsalgorithmus zu suchen, der eine Alternative zu AES-GCM bieten kann.
Dafür muss eine Auswahl aus potentiellen Verschlüsselungsalgorithmen getroffen werden, die daraufhin in das MACsec Modul implementiert werden. Des Weiteren muss überprüft werden, ob sich durch einen Austausch der Verschlüsselungskomponente eine Performancesteigerung des Sicherheitsstandards erreichen lässt. Dafür wird eine Testumgebung kreiert und Performance Tests durchgeführt. Um einen Einblick in die Thematik zu geben, werden im nächsten Kapitel die Grundlagen erklärt. Daraufhin werden Verschlüsselungsalgorithmen vorgestellt, die mögliche Kandidaten für eine Erweiterung in das MACsec Modul sind. Danach werden die Änderungen beschrieben, die im MACsec Modul getätigt werden müssen, um zusätzliche Verschlüsselungsalgorithmen nutzen zu können. Diese werden nach den Kriterien der Sicherheit, Funktionalität und Geschwindigkeit bewertet. Zum Schluss wird ein Fazit aus den Ergebnissen der Analyse gezogen.

2 Grundlagen

Um einen Überblick in die Grundlagen dieser Bachelorarbeit zu geben, erfolgt in diesem Kapitel zunächst die Erläuterung zentraler Begriffe der Netzwerktheorie. Das ist eine wichtige Voraussetzung, um die Funktionsweise vom Sicherheitsstandard MACsec verstehen zu können. Danach werden grundlegende im Standard benutzte Sicherheitsbegriffe beschrieben, um anschließend auf den Standard als solches, dessen Funktion und Sicherheitsziele einzugehen. Falls es nicht anders erwähnt wird, werden die Kenntnisse für den Abschnitt 2.1 aus [TW12] entnommen.

2.1 Netzwerktheorie

OSI Referenzmodell

Das OSI Referenzmodell wurde von der International Standards Organization (ISO) entwickelt. Es war der erste Versuch Kommunikationsprotokolle zu standardisieren, um zwischen verschiedenen technischen Systemen zu kommunizieren. Das OSI-Modell besteht aus sieben Schichten, wobei jede einzelne Schicht eine genau definierte Funktion erfüllt. Wenn eine Nachricht von einem zum anderen Computer geschickt wird, durchläuft die Nachricht zuerst alle 7 Schichten auf dem Computer des Senders und ein zweites Mal alle 7 Schichten auf dem Computer des Empfängers. Für den Gegenstand der Bachelorarbeit ist die Sicherungsschicht von besonderer Bedeutung, da MACsec selbst auf der 2. Schicht des OSI-Modells arbeitet und Kenntnisse aus dieser Schicht relevant zum Verständnis dieser Arbeit sind.

Sicherungsschicht

Die Sicherungsschicht ist die zweite Ebene des OSI-Modells. Ihre Aufgabe ist es den Bitstrom aus der Bitübertragungsschicht (Schicht 1) in Blöcke, die Rahmen genannt werden, aufzuteilen und an die Vermittlungsschicht (Schicht 3) weiterzuleiten. So muss durch die Sicherungsschicht eine zuverlässige Übertragung der Daten gewährleistet werden. Hierfür werden zusätzliche Redundanzen zum Rahmen hinzugefügt, um gesendete Informationen auf Fehler zu prüfen und Maßnahmen der Fehlerkorrektur durchführen zu können. Deshalb muss in der Vermittlungsschicht nicht mehr auf grobe Fehler in dem Datenpaket geprüft werden. Ein weiteres wichtiges Kriterium ist die Regulierung des Datenflusses durch den Empfänger. Dies dient dazu, dass es nicht zu einer Überlastung des Empfängers kommt, falls der Sender schneller Daten sendet als der Empfänger annehmen kann. Die Sicherungsschicht wird in zwei Teilschichten untergliedert, die Medium Access Control (MAC) und die Logical Link Control (LLC).

Präambel 8 Bytes	Zieladresse 6 Bytes	Quelladresse 6Bytes	Ether Type 2 Bytes	Daten 46 -1500 Bytes	Checksumme 4 Bytes

Abbildung 2.1: In der Abbildung werden die 6 Bestandteile eines Ethernet Rahmen gezeigt. Dazu wird die Größe eines Feldes in Bytes angezeigt.

Medium Access Control

In der Medium Access Control Teilschicht werden die vorhandenen Kommunikationskanäle zwischen den Mitgliedern innerhalb eines Netzwerks organisiert. Hierbei wird während der Übertragung überprüft, ob es zu einer Kollision auf dem Kanal zwischen den Kommunikationspartnern gekommen ist. Im Falle einer Kollision werden je nach Protokoll unterschiedliche Maßnahmen ergriffen, um die Kollisionsgefahr auf dem Kanal zu verringern. Allerdings müssen bei einer Kollision die Nachrichten generell erneut gesendet werden. In der Literatur findet man auch kollisionsfreie Übertragungsprotokolle, diese finden in der Praxis jedoch kaum Anwendung.

Ethernet

Ethernet wurde unter dem Begriff IEEE Norm 802.3 standardisiert. Daher können IEEE Norm 802.3 und Ethernet als Synonym verwendet werden. In dieser Bachelorarbeit wird stets der Begriff Ethernet verwendet, da dies die gängigste Verbreitung des Ausdrucks ist. Ethernet beschreibt im Allgemeinen die physische Verbindung von 2 Systemen mittels eines Kabels, wodurch die Kommunikation der beiden Systeme erfolgt.

Ethernet auf der MAC-Teilschicht

Die Daten, die über die MAC-Teilschicht des OSI-Modells laufen, werden in der Fachliteratur als Rahmen bezeichnet. Deshalb wird im Verlauf der Arbeit stets der Begriff Rahmen für diese Daten übernommen. Ein Rahmen beim Ethernet in der MAC-Teilschicht weist folgende Struktur auf. Als erstes beginnt ein Rahmen mit einer Präambel, die eine Länge von 8 Bytes aufweist, wobei jedes Byte das Bitmuster 10101010 besitzt, ausgenommen sind die letzten beiden Bits. Diese werden auf 11 gesetzt. Dies hat den Zweck, Empfänger und Sender zu synchronisieren. Als nächstes folgen zwei 6 Bytes lange Stücke. Der erste Teil beinhaltet die MAC-Addresse des Empfängers und der zweite die des Senders. Eine Besonderheit der MAC-Adresse ist dadurch gekennzeichnet, dass die Vergabe nur von der IEEE erfolgt und jede MAC-Adresse global einzigartig ist. Anschließend wird die Länge des Rahmens in einem 2 Byte großem Feld angegeben. In neueren Ethernet Versionen wird das 2 Byte große Feld zum Typ Feld, sodass Protokolle mittels des Typ Feldes unterschieden werden können. Danach folgt nun das Daten Feld, in denen sich die zu sendenden Daten befinden. Das Daten Feld ist maximal 1500 Bytes groß. Also kann ein Rahmen in der MAC-Teilschicht eine

maximale Größe von 1518 Bytes haben, wenn man die Größe des Headers und Trailers hinzurechnet. Außerdem gibt es auch eine Mindestgröße des Daten Feldes von 46 Bytes. Sollte eine Nachricht trotzdem kleiner sein, so wird die Nachricht mit Nullen auf 46 Bytes aufgefüllt. Daraus folgt, dass ein kompletter Rahmen mindestens eine Größe von 64 Bytes hat. Dies ist die minimale Größe, die benötigt wird, um die Verfahren zur Kollisionserkennung gewährleisten zu können. Ansonsten ist die ganze Nachricht bereits gesendet, bevor der Empfänger realisieren kann, dass eine Kollision vorlag. Zuletzt kommt ein 4 Byte großes Feld, in dem sich die Checksumme befindet. Durch die Checksumme kann überprüft werden, ob die Nachricht korrekt übertragen wurde. Falls dies nicht der Fall sein sollte, kann der Empfänger ein Signal senden, damit der Sender den Rahmen erneut überträgt. Nach einer übermittelten Nachricht wird eine 9,6 µs lange Sendepause eingelegt. Die Struktur eines Ethernet Rahmens mit seinen einzelnen Feldern ist in der Abbildung 2.1 zu sehen.

ICMP

Das Internet Control Message Protocol (ICMP) ist ein Steuerprotokoll, welches auf der Vermittlungsschicht arbeitet. Sollte im Verlauf einer Paketübertragung ein Router ausfallen oder ein anderes Problem auftreten, so wird dem Sender durch das ICMP ein Paket übermittelt, in dem der Sender über das Ereignis informiert wird. Eine Liste der unterstützten Nachrichten findet man unter [Pos+18].

TCP

Das Transmission Control Protocol (TCP) ist ein Internetübertragungsprotokoll, das eine zuverlässige Übertragung garantiert. Außerdem wird bei TCP auf die Reihenfolge der Pakete wert gelegt. Es wird in der der Transportschicht des OSI-Modells, der 4. Schicht, eingesetzt. Auch wenn die vorhandenen Netzwerke unzuverlässig sind, kann sich das TCP robust auf alle Arten von Fehlern einstellen. Beim TCP-Protokoll werden übertragende Daten als Segmente bezeichnet. Ein TCP-Segment fängt mit einem 20-Byte langem Header an, gefolgt von einem optionalen Teil und anschließend von einem Datenbytes-Feld, das auch, anders als bei Ethernet, keine Daten enthalten kann.

2.2 Sicherheit

In diesem Abschnitt werden einige Grundlagen der Sicherheit in Rechnernetzen erläutert, die für das Verständnis der Funktionsweise des Sicherheitsstandards MACsec von Bedeutung sind. Die Informationen für diesen Abschnitt sind aus [Pfi00] entnommen.

Sicherheitsziele

Es wird im Allgemeinen in 3 Sicherheitsziele unterteilt:

- Vertraulichkeit: Es ist nicht möglich, dass eine unautorisierte Partei einen Informationsgewinn über den Inhalt der gesendeten Daten erhält.

- Integrität: Es ist nicht möglich, dass eine unautorisierte Partei den Inhalt von Daten modifizieren kann, ohne das dies bemerkt wird.
- Verfügbarkeit: Es ist nicht möglich, dass eine unautorisierte Partei die Funktionalität eines Dienstes beeinträchtigen kann.

In dieser Arbeit ist noch ein 4. Sicherheitsziel von Bedeutung:

- Authentizität: Eine gesendete Nachricht kann einem Sender zugeordnet werden.

Symmetrisches Konzelationssystem

Bei einem symmetrischen Konzelationssystem wird ein zufälliger Schlüssel generiert, der zwischen Sender und Empfänger ausgetauscht wird. Dieser Schlüssel wird sowohl bei der Verschlüsselung als auch Entschlüsselung von Nachrichten angewendet. Mit einem symmetrischen Konzelationssystem kann das Schutzziel der Vertraulichkeit erfüllt werden.

Symmetrisches Authentikationssystem

Bei einem symmetrischen Authentikationssystem wird ebenfalls ein zufälliger Schlüssel generiert, der zwischen Sender und Empfänger ausgetauscht wird. Anders als bei einem Konzelationssystem werden die ausgetauschten Nachrichten nicht verschlüsselt und entschlüsselt, sondern es wird ein Prüfteil an die zu sendende Nachricht angefügt. Der Empfänger einer Nachricht kann mit dem richtigen Schlüssel den Prüfteil selbständig berechnen und vergleichen. Sollte sich der berechnete Prüfteil von der erhaltenen Nachricht unterscheiden, dann wurde die Nachricht verändert. Mit einem Authentikationssystem kann das Schutzziel der Integrität erfüllt werden. In der restlichen Arbeit wird der gängigere englische Begriff des Message Authentikation Codes MAC anstelle des Authentikationssystems verwendet.

Replay-Angriff

Bei einem Replay-Angriff werden bereits gesendete Nachrichten von einem Angreifer gesammelt und erneut in den Nachrichtenfluss gestreut, meistens mit der Intention das Sicherheitsziel der Authentizität zu schwächen.

bounded receive delay

Unter bounded receive delay versteht man, dass ein Man-in-the-Middle Angreifer keine Pakete in einem Netzwerk verzögern oder abfangen kann, ohne das dies von den Beteiligten in einem Netzwerk erkannt wird.

Denial of Service

Denial of Service (DoS) ist ein Angriff mit der Absicht, das Sicherheitsziel der Verfügbarkeit zu schwächen. Dazu werden an einen Server eine Vielzahl von Anfragen geschickt, bis der Server überlastet ist und diese nicht mehr beantworten kann. Der Server kann zu diesem Zeitpunkt keine sinnvollen Anfragen von Nutzern mehr beantworten, die auf den Service des Servers zugreifen wollen. Somit ist der Service für die Nutzer nicht mehr verfügbar.

2.3 Authenticated Encryption with Associated Data

Authenticated Encryption with Associated Data (AEAD) ist ein Betriebsmodus, in dem ein Konzelationssystem und ein Authentikationssystem in einem Verschlüsselungsprotokoll vereint werden. Dabei ist es nicht unüblich, dass zwei unterschiedliche Algorithmen, eine Stromchiffre und ein Message Authentikation Code zu einen Schema verbunden werden. Um mittels AEAD verschlüsseln zu können, braucht man folgende Inputs:

- den symmetrischen Schlüssel (K)
- den Initialisierungsvektor (IV)
- die associated Data (AD)
- den Klartext (M)

Die AD ist ein zusätzlicher Input, der zu dem Verschlüsselungsprotokoll hinzugefügt wird, aber intern nicht verschlüsselt wird. Vielmehr wird die AD in den Zustand des Algorithmus geladen, um das Schutzziel der Authentizität zu erreichen. Im MACsec Protokoll werden Teile des MACsec Headers wie Sende- und Empfängeradresse als Associated Data benutzt. Das wird später in 2.4 erläutert. Die Länge des symmetrischen Schlüssels und die Länge des Initialisierungsvektors muss nicht gleich sein. So ist zum Beispiel die Länge des Schlüssels bei AES-GCM 128 Bit lang, wobei der selbe Algorithmus nur einen 96 Bit langen Initialisierungsvektor verwendet. Allerdings verlieren die meisten AEAD Algorithmen ihre Sicherheit, wenn der gleiche Schlüssel mit dem gleichen Initialisierungsvektor benutzt wird. Daher muss AES-GCM mit einem kürzeren IV öfter den Schlüssel wechseln als andere Algorithmen, die einen längeren Initialisierungsvektor benutzen.
Als Output bekommt man einmal das Chiffrat (C) und den MAC (T), der unter anderem auch als Tag bezeichnet wird. Der MAC wird an die verschlüsselte Nachricht konkateniert, sodass am Ende des Verschlüsselungsprotokolls die Nachricht um die Länge des MACs vergrößert wird. Wie bereits erwähnt, kann über den MAC das Schutzziel der Integrität erreicht werden. Dadurch, dass am Anfang die AD in den Algorithmus geladen wird, kann auch über den MAC die Authentizität des Senders bestätigt werden.

Zum Entschlüsseln benötigt man 4 Inputs:

- Den Schlüssel (K)

- den Initialisierungsvektor (IV)
- die associated Data (AD)
- das Chiffrat (C)

Während der Entschlüsselung wird aus dem erhaltenden Chiffrat ein MAC errechnet. In diesem Vorgang wird der errechnete MAC mit dem gesendeten verglichen und bei erfolgreicher Validierung wird die Nachricht ausgegeben. Sollte sich dabei herausstellen, dass der MAC nicht korrekt ist, wird bei diversen AEAD Algorithmen empfohlen, den produzierten MAC nicht auszugeben, da sonst die Algorithmen anfällig gegenüber known-plaintext oder chosen-ciphertext Angriffen sind [McG08].

2.3.1 Generische AEAD Algorithmen

Ein beliebter Grundgedanke, um einen AEAD Algorithmus zu erstellen, ist die Kombination aus einer effizienten Stromchiffre mit einem schnellen Message Authentikation Code. Diesen Ansatz bezeichnet man als generische AEAD Algorithmen. Um die geforderten Sicherheitsziele der Vertraulichkeit, Authentizität und Integrität zu garantieren, müssen 2 Bedingungen erfüllt werden:

1. Der Verschlüsselungsalgorithmus muss gegen einen chosen-plaintext Angriff sicher sein.
2. Der Message Authentication Code muss fälschungssicher bei chosen-message Angriffen sein.

Es gibt 3 Methoden, um einen generischen AEAD Algorithmus zu bilden.
Die 1. Variante nennt sich Encrypt-and-MAC. Bei Encrypt-and-MAC wird der Klartext verschlüsselt und es wird ein Tag vom Klartext gebildet. Anschließend wird der Tag an den Ciphertext konkateniert. Bei der Verschlüsselung wird als erstes der Ciphertext entschlüsselt. Anschließend wird aus dem Klartext ein Tag berechnet und mit dem erhaltenen Tag verglichen.
Eine weiterere Möglichkeit nennt sich MAC-then-encrypt. Wenn MAC-then-encrypt verwendet wird, dann wird als erstes ein MAC an den Klartext konkateniert. Daraufhin wird der komplette Klartext mit dem MAC verschlüsselt. Um den MAC zu verifizieren, muss als erstes die komplette Nachricht entschlüsselt werden. Dann wird aus dem Klartext ein MAC berechnet und verglichen.
Die letzte Variante wird als Encrypt-then-MAC bezeichnet. Es ist das sicherste Verfahren, bei der als erstes die Nachricht verschlüsselt und danach aus dem Ciphertext der MAC gebildet wird. Bei der Authentifizierung der Nachricht erfolgt die Berechnung des MAC aus dem Ciphertext. Im Anschluss findet der Vergleich mit den beiden MACs statt und bei erfolgreicher Validierung folgt die Entschlüsselung des Ciphertextes [BN00].

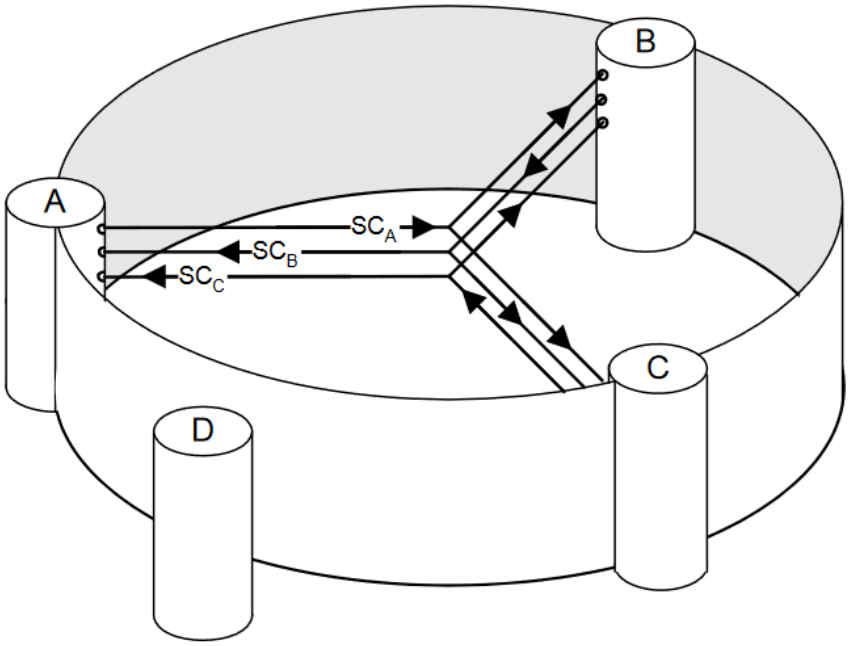

Abbildung 2.2: In dieser Abbildung werden 3 Stationen gezeigt, die sich in eine Connectivity Association befinden und mittels Secure Channel kommunizieren können.

2.4 Media Access Control Security - MACsec

Media Access Control Security MACsec ist ein Sicherheitsprotokoll, welches von dem Institute of Electrical and Electronics Engineers IEEE veröffentlicht worden ist. Es ist auch unter dem Begriff IEEE 802.1AE bekannt. Das Sicherheitsprotokoll MACsec bietet Schutz auf der 2. Ebene des OSI-Modells und ist in der Lage, Integrität, Vertraulichkeit und Authentizität zu gewährleisten, weil es das Verschlüsselungsschema AEAD verwendet. Soweit es nicht anders erwähnt wird, werden die Informationen für dieses Kapitel aus [06] entnommen.

MACsec kann eine sichere Kommunikation in einem LAN-Netzwerk bereitstellen. Um das MACsec Protokoll verwenden zu können, muss jeder Nutzer eine MAC Security Entity (SecY) und eine Security Key Agreement Entity (KaY) benutzen. Eine SecY ist eine Instanz des MACsec Protokolls, die alle Operationen ausführt, die im MACsec Standard beschrieben werden. Allerdings kann die SecY nicht zwischen Nutzern, die das MACsec Protokoll benutzen und normalen Nutzern unterscheiden. Um Nutzer authentifizieren zu können, wird die KaY benötigt. Die KaY ist für den Schlüsselaustausch, die Bereitstellung der Schlüssel, für die Autorisierung und Authentifizierung von anderen Nutzern mit einer SecY verantwortlich. Hierfür ist ein eigener Standard, das MACsec Key Agreement Protokoll (MKA) entstanden[10].
Ein Nutzer mit einer SecY kann in genau eine Connectivity Association (CA) eintreten.

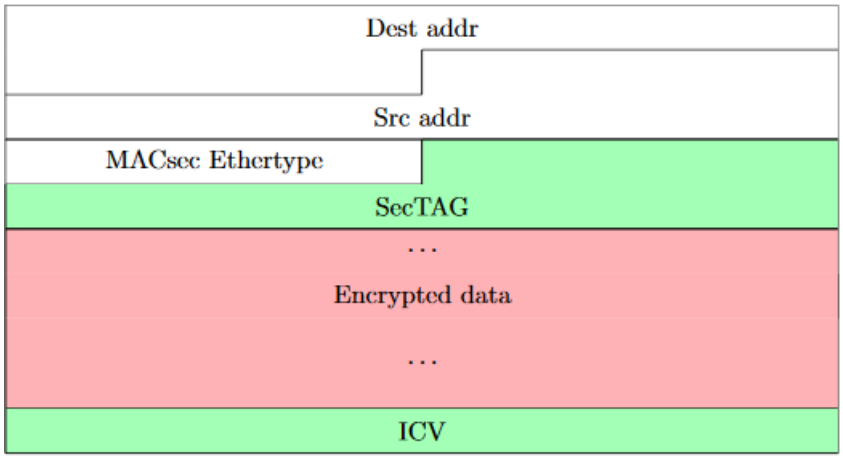

Abbildung 2.3: In der Abbildung werden die Änderungen veranschaulicht, die der MACsec Standard an einem Rahmen vornimmt. Die grün und rot markierten Felder werden vom ICV geschützt. Das rote Feld beschreibt zudem das Nutzdaten Feld, welches verschlüsselt werden kann.

CA bilden sichere Umgebungen, in denen die Mitglieder miteinander kommunizieren können. Nur die KaY kann Nutzer innerhalb einer CA erkennen. Eine MACsec SecY muss sich indes gar nicht bewusst sein, dass sie sich in einer CA befindet. Wenn sich zwei SecY innerhalb einer CA befinden, so können sie über einen Secure Channels (SC) miteinander kommunizieren. Ein SC wird auf 4 Secure Associations (SA) aufgeteilt. Jede Secure Association besitzt ihren eigenen Secure Association Key (SAK), ein Schlüssel, der durch die KaY bereitgestellt wird. Jede SA wird durch einen Secure Channel Identifier (SCI) identifiziert. Der Empfänger einer Nachricht kann durch den SCI eine SA erkennen, was wichtig ist, um den richtigen Schlüssel für die Entschlüsselung zu wählen. 2.2 zeigt die SC zwischen mehreren SecY innerhalb eines CA.

2.4.1 MACsec auf Layer 2

Wie bereits erwähnt, arbeitet MACsec auf der zweiten Ebene des OSI-Modells. Wird ein Paket über MACsec verschickt, dann verändert MACsec den Rahmen. Es wird ein zusätzliches Feld der SecTag angefügt und der Ethertype des Rahmens wird auf 0x88e5 gesetzt, den Ethertype von MACsec. Zusätzlich wird ein Integrity Check Value (ICV) nach dem Daten Feld konkateniert. Der ICV wird durch den Message Authentication Code, der in MACsec genutzt wird, gebildet. Als Input wird der SecTag, der Ethertype und das Daten Feld genommen, sodass der ICV alle wichtigen Bestandteile des Rahmens schützt. Es gibt in MACsec mehrere Möglichkeiten, um einen Rahmen zu schützen. Man kann die Vertraulichkeit und Integrität oder nur Integrität schützen. Bei Vertraulichkeit und Integrität wird als erstes das Daten Feld verschlüsselt und dann der ICV über den kompletten Rahmen gebildet. Wenn man nur Integrität auswählt, dann wird nur ein ICV gebildet. Die Änderungen am Rahmen ermöglichen MACsec Angriffsmöglichkeiten wie Replay Angriffe und Denial of Service zu unterbinden. Außerdem

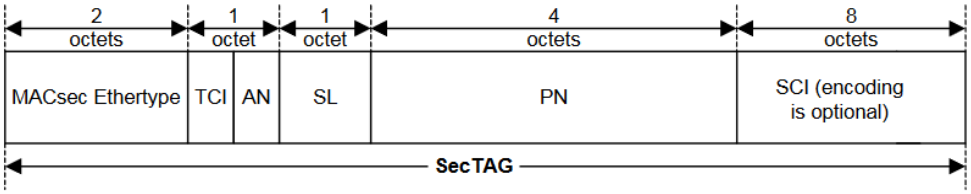

Abbildung 2.4: Die Abbildung zeigt den SecTag Header des MACsec Standards. Darin sind die einzelnen Felder und deren Größe in Bytes zu erkennen.

lehnt MACsec Rahmen ab, die eine zu lange Übertragungszeit haben. Damit wird ein möglicher bounded receive delay unterbunden. In der Abbildung 2.3 wird ein MACsec Rahmen dargestellt. Die grün und rot markierten Felder werden vom ICV geschützt. Das rote Feld zeigt die verschlüsselten Daten.

MACsec SecTag

Der SecTag ist ein zusätzlicher Header, der zwischen dem Ethertype Feld und dem Daten Feld eines MACsec Rahmens eingefügt wird. Je nach Auswahl besitzt der SecTag eine Größe von 8 oder 16 Bytes und besteht aus folgenden Feldern:

- dem Ethertype mit einer Größe von 2 Bytes.
- der Tag Control Information mit einer Größe von 6 Bits.
- der Association Number mit einer Größe von 2 Bits.
- dem Short Field mit einer Größe von 1 Byte.
- der Paket Nummer mit einer Größe von 4 Bytes.
- dem SCI ein Optionales Feld mit einer Größe von 8 Bytes.

Ob ein Rahmen durch ein Protokoll genutzt oder verändert worden ist, kann man an dem Ethertype Feld erkennen. Bei MACsec hat das Ethertype Feld den statischen Wert 0x88E5. Darauf folgt das TAG Control Information Feld (TCI), das im nächsten Absatz näher erläutert wird. Die Association Number (AN) wird mit dem Secure Channel Identifier konkateniert, um eine Secure Association zu erkennen. Danach kommt das Feld Short Field. Dieses Feld ist in der Regel auf 0 gesetzt. Sollte allerdings ein Paket kürzer, als die von MACsec vorausgesetzten 48 Bytes sein, wird in diesem Feld die Länge des Pakets als Integer angegeben. Die Paketnummer ist Teil des Initialisierungsvektors. Jedes Paket, das gesendet wird, bekommt seine eigene Paketnummer, die inkrementiert wird und somit immer unterschiedlich ist. Zum Schluss kommt das optionale Feld mit dem Secure Channel Identifier. Der SCI wird nur aktiviert, wenn das SC Bit im TCI gesetzt ist. Mit dem SCI und der AN kann man zusammen die SA bestimmen. Das ist besonders wichtig, da jede SA ihren eigenen SAK besitzt. Somit muss erkannt werden, in welcher SA die Nachricht übermittelt worden ist, um den richtigen Schlüssel zur

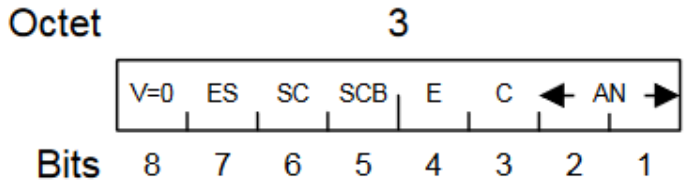

Abbildung 2.5: In der Abbildung werden die einzelnen Bits des Tag Control Information Feldes aufgezeigt.

Entschlüsselung zu benutzen. Des Weiteren wird der SCI genutzt, um die SecY zu identifizieren, die einen Rahmen zur Station gesendet hat. Wird kein SCI benutzt, so übermittelt das MKA einen Wert und eine Verbindung mittels Point-to-Point Protocol (PPP) wird erstellt. In der Abbildung 2.4 werden die einzelnen Bestandteile eines MACsec SecTag dargstellt. Zusätzlich wird die Größe jedes Feldes in Bytes angezeigt.

MACsec Tag Control Information

Das Tag Control Information Feld ist Bestandteil des MACsec SecTag Headers. Das TCI ist nur 6 Bits lang, allerdings hat jedes Bit eine feste Funktion. Das erste Bit steht für die Versionsnummer. Im Standard wird angemerkt, dass in der Zukunft mehr als ein Bit benötigt werden, um die Versionsnummer anzuzeigen. Das zweite Bit steht für das End Station Bit (ES). Das ES Bit wird nur gesetzt, wenn die ersten 6 Bytes des TCI identisch zu den ersten 6 Bytes der MAC Adresse sind. Ist das ES Bit gesetzt, so wird vom Standard [06] empfohlen, das SCI nicht direkt im SecTag zu kodieren und das SC Bit nicht zu setzen. Wenn ein SCI benutzt wird, ist das SC Bit aktiv. Nur wenn ein MACsec Rahmen mit einem Secure Channel assoziiert wird und EPON Single Copy Broadcast capability unterstützt, dann wird das SCB Bit gesetzt. Das E Bit ist gesetzt, wenn das Secure Payload Feld verschlüsselt werden soll. Das letzte Bit von der TCI ist das Changed Text Bit. Ist das E Bit gesetzt und das Changed Text Bit, so wurden erfolgreich die Nutzdaten durch die SecY verarbeitet. Ist das E Bit gesetzt, aber das Changed Text Bit noch nicht, dann wurden die Nutzdaten noch nicht durch die SecY verändert. In der Abbildung 2.5 ist das 6-Bit lange TCI Feld mit seinen einzelnen Komponenten und der AN dargestellt.

MACsec Secure Data

Das Secure Data Feld beschreibt die Nutzdaten zwischen dem SecTag und dem ICV. Das Feld kann nicht leer sein und muss eine Länge von mindestens 48 Bytes haben, da dies die kleinste Größe eines Rahmens im Ethernet Standard ist. Ist es kürzer, dann erfolgt die Nutzung von Padding.

Integrity Check Value

Der Integrity Check Value überprüft, ob eine Nachricht manipuliert worden ist. Im

MACsec Standard ist der ICV zwischen 8 und 16 Bytes lang, je nachdem welche Cipher Suite ausgewählt worden ist. Wenn der vom Standard empfohlene Algorithmus AES-GCM ausgewählt ist, beträgt die Länge 16 Bytes.

2.4.2 MACsec Cipher Suite

In der Cipher Suite werden alle kryptografischen Algorithmen spezifiziert, die in MACsec angewendet werden können. Zu diesem Zeitpunkt befindet sich nur AES-GCM mit einer Schlüssellänge von 128 Bit in der Cipher Suite von MACsec. Es ist bereits eine Erweiterung der Cipher Suite geplant, in der AES-GCM mit einer Schlüssellänge von 256 Bit hinzugefügt werden soll[11]. Will man einen weiteren Algorithmus zur Cipher Suite hinzufügen, so müssen folgende Bedingungen erfüllt werden:

- Durch den Algorithmus müssen die Parameter SCI, Paketnummer, Source Address, Destination Address, SecTag und die Nutzdaten geschützt werden.
- Der Algorithmus muss sowohl Integrität als auch Vertraulichkeit schützen können.
- Ein Verschlüsselungsalgorithmus muss bis zu 2^{32} -1 mal aufrufbar sein, ohne das der SAK gewechselt werden muss.

Default Cipher Suite AES-GCM

Die empfohlene Cipher von MACsec ist AES-GCM auch bekannt unter AES im Galois Counter Mode. 2007 wurde AES-GCM von der amerikanischen Bundesbehörde National Institute of Standards and Technology (NIST), die auch schon den Data Encryption Standard (DES) und AES standardisiert hat, ein Paper veröffentlicht, in dem AES-GCM empfohlen wurde. Es ist eine generische AEAD Cipher, die aus der Stromchiffre AES-CTR mit einer Blocklänge von 128 Bits besteht und aus dem Message Authentikation Code GMAC, der die ICV von MACsec mit einer Länge von 128 Bits berechnet. Wie bereits bei 2.3 erwähnt, benötigt ein AEAD Schema einen fest definierten Input, der wie folgt aussieht:

$$(\mathrm{C},\mathrm{T}) = enc(\mathrm{IV},\mathrm{M},\mathrm{A},\mathrm{K})$$

Die Schlüssellänge von AES-GCM ist 128 Bits lang und als Schlüssel wird der SAK einer SA benutzt. Die ersten 64 Bits von dem 96 Bits langen IV bestehen aus dem SCI. Die restlichen 32 Bits bilden die iterierende Paket Number. Der Input der AD ist die Konkatenation der Destination Mac Adresse, der Source Mac Adresse, des SecTags und zum Schluss der Nutzdaten. Die Argumente werden auch in der genannten Reihenfolge in den Algorithmus geladen. Unter Anderem bietet MACsec 3 Möglichkeiten an, um eine Nachricht zu schützen:
1. Wenn MACsec nur die Integrität einer Nachricht sichern soll, dann ist der Input der Nachricht(M) leer. Die restlichen Parameter werden wie oben beschrieben als Input genommen. Die Secure Data(C) ist in diesem Fall unverändert und es folgt ein 128 Bit langer ICV, der die Nachricht schützt.
2. Wenn Vertraulichkeit und auch Integrität einer Nachricht geschützt werden sollen, dann muss man einmal unterscheiden, ob man die Nachricht mit einem confidentiality offset oder ohne confidentiality offset verschlüsseln möchte.

2.1 Wird mit einem confidentiality offset verschlüsselt, dann wird ein fest auswählbarer Wert z.B. 30 Bytes von der Nachricht (M) nicht verschlüsselt, sondern nur mittels Integrität geschützt. Also die ersten Bytes von der Nachricht direkt nach dem SecTag sind in Klartext verfügbar, werden aber trotzdem durch den ICV geschützt. Der Teil der Nachricht, der nicht zum confidentiality offset gehört, wird normal verschlüsselt, ist aber nicht durch den ICV geschützt. Die Secure Data ist der confidentiality offset konkateniert mit den chiffrierten Nutzdaten.
2.2 Wird ohne confidentiality offset verschlüsselt, dann wird das gesamte Nutzdaten Feld chiffriert. Die AD wird wie oben beschrieben behandelt.

2.5 Softwareoptimierung

Softwareoptimierungen sind angepasste Implementationen, sodass die bestehenden Hardware Unterstützungen von Systemen besser genutzt werden können, um dadurch eine Performance Steigerung zu erreichen. Es gibt Hardware Unterstützungen, die besonders nützlich für Verschlüsselungsalgorithmen sind. Das liegt daran, dass sich die CPU Instruktionen besonders gut dafür eignen, um die verwendeten mathematische Operationen in Verschlüsselungsalgorithmen zu berechnen. Viele Chiffren machen von Software Optimierungen Gebrauch und der AES Verschlüsselungsalgorithmus besitzt sogar einen eigenen Hardware Befehlssatz namens AES-NI. In diesem Abschnitt wird daher ein Übersicht über die gängigsten Hardware Unterstützungen geschaffen, die von Verschlüsselungsalgorithmen benutzt werden.

2.5.1 AES-NI

Wie bereits erwähnt, verfügt der Verschlüsselungsalgorithmus AES über einen eigenen Befehlssatz AES New Instructions kurz AES-NI. Die Befehlssatzerweiterung ist seit 2010 in x86-Prozessoren zu finden. Damit werden 7 zusätzliche Assemblerbefehle unterstützt, die unter anderem Befehle für eine Verschlüsselungsrunde, eine Entschlüsselungsrunde und der Schlüsselgenerierung von AES unterstützt. In [Gue10] wird außerdem gezeigt, dass durch AES-NI die Performance um das 2-3 Fache gesteigert werden kann. Es wird sogar geschildert, dass AES-NI die Sicherheit von AES erhöhen kann, da durch die Instruktionen side-channel Schwächen in der Software von AES verringert werden.

2.5.2 SSE

Eine weitere wichtige Befehlssatzerweiterung ist die Streaming Simd Extensions (SSE). SSE bringt 70 Befehlsinstruktionen mit sich und wurde bereits 1999 in x86-Prozessoren unterstützt. Aufgrund der Verwendbarkeit der Instruktionen ist der Befehlssatz in fast jedem x86-Prozessor verfügbar und seitdem sind sogar mehrere Nachfolger von SSE wie z.B SSE2 oder AVX erschienen. In SSE gibt es bis zu 16 128-Bit große Register, die über Vektor Operationen manipuliert werden können. Das hat zur Folge, dass mehrere Befehle an einem Register gleichzeitig durchgeführt werden können. Aus diesem Grund hängt es von der Parallelisierbarkeit des Verschlüsselungsalgorithmus ab, wie sehr die Chiffre vom Befehlssatz profitieren kann[AA16].

2.5.3 AVX

Die Advanced Vector Extensions Befehlssatzerweiterung AVX ist wie bereits erwähnt ein Nachfolger von SSE. Bei AVX wurden die 16 128-Bit Register auf 16 256-Bit Register mit neuen Instruktionen erweitert. Das Prinzip bleibt das gleiche. Es können mit Vektor Operationen parallel mehrere Befehle an einem Register ausgeführt werden. Durch die 256-Bit großen Register können doppelt so viele Daten simultan verarbeitet werden[Mos17].

3 AEAD Algorithmen

In diesem Abschnitt werden verschiedene AEAD Algorithmen vorgestellt. Daraufhin werden aus den Algorithmen Kandidaten ausgewählt, die vielversprechende Ergebnisse aufweisen. Die ausgewählten Kandidaten werden detaillierter analysiert und erläutert.

3.1 CAESAR Competition

Am 5. Juli 2012 wurde von Daniel J. Bernstein ein rundenbasierter kryptografischer Wettbewerb namens Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) gestartet, um eine alternative AEAD Chiffre zum weit verbreiteten AES-GCM zu finden [Ber14]. Im Ausschuss, der über die Verschlüsselungsalgorithmen berät, sitzen neben Bernstein auch andere Spezialisten wie z.B. Joan Daemen oder Vincent Rijmen. Beide haben den Rijndael-Algorithmus konstruiert, der später vom Nist weltweit als Standard namens AES gewählt worden ist und in fast jedem netzwerkfähigen Gerät benutzt wird. Auch Bernstein hat mit Chacha20-Poly1305 eine weit verbreitete Chiffre, die bereits oft eine Alternative zu AES-GCM bietet. Das besondere an der CAESAR Competition ist, dass es erstmals eine standardisierte Hardware API gibt, auf dem jeder Algorithmus getestet werden kann. So können leichter Vergleiche zwischen den Chiffren auf der gleichen Plattform gezogen werden und die Ergebnisse erlangen eine größere Relevanz. Die eingereichten Algorithmen sollten den Galois Counter Mode in mindestens einem von 3 Anwendungsfällen übertreffen. Die Anwendungsfälle sind wie folgt definiert:

1. Lightweight Applications
2. High Performance Applications
3. Defense in Depth

Chiffren, die für Anwendungfall 1 eingereicht werden, sollten sehr effizient auf 8-Bit Central Processing Units (CPU) arbeiten. Trotzdem sollten die Algorithmen nicht gegenüber Side-Channel Attacks anfällig sein.
Anwendungsfall 2 beschreibt Chiffren die eine sehr gute Leistung auf 64-Bit CPUs erbringen können. Außerdem sollten diese Algorithmen ebenfalls effizient auf 32-Bit CPUs sein, die hauptsächlich in Handys verbaut werden.
In dem letzten Anwendungsfall soll das Sicherheitsziel der Authentizität dennoch gewährleistet werden, obwohl der Initialisierungsvektor mehrfach mit dem gleichen Schlüssel benutzt worden ist. In diesem Fall spricht man auch von einem Missbrauch des Initialisierungsvektors. Eine genaue Beschreibung der Anwendungsfälle ist in einer Email von einem der Ausschussmitglieder des CAESAR Wettbewerbs Daniel J. Bernstein zu

finden[1]. Die CAESAR Competition durchläuft seit 2012 mehrere Runden und hat im März 2018 7 Finalisten bekanntgegeben. Aus diesen 7 Finalisten wird ein Portfolio an Algorithmen erstellt, die von dem Gremium, in den verschiedenen Anwendungsfällen empfohlen werden.

3.1.1 Finalisten des CAESAR Wettbewerbs

Bei dem CAESAR Wettbewerb wurden über 50 unterschiedliche AEAD Algorithmen in der ersten Runde eingereicht. Da Experten der Kryptografie im Gremium sitzen, sind alle Finalisten mögliche Kandidaten, die in MACsec implementiert werden können. Deshalb werden in diesem Abschnitt die Finalisten in alphabetischer Reihenfolge kurz zusammengefasst.

ACORN: ACORN wurde von Hongjun Wu eingereicht und ist für den Anwendungsfall 1 konfiguriert worden. Es ist eine Stromchiffre, welche auf einem linear rückgekoppelten Schieberegister mit einer Größe von 293 Bits basiert. Der Verschlüsselungsalgorithmus wurde so entworfen, dass die Chiffre besonders effizient auf Hardware läuft. Wegen der Parallelisierbarkeit von ACORN ist der Algorithmus auch verhältnismäßig schnell in Software[Wu16].

AEGIS: AEGIS basiert auf der Rundenfunktion von AES[2]. Die Stromchiffre wurde von Bart Preneel[3] und Hongjun Wu veröffentlicht und kann einen sehr hohen Durchsatz erzielen. Sie wird deshalb für Anwendungsfall 2 empfohlen. Die Rundenfunktion von AEGIS wird je nach Auswahl 5 bis 8 mal wiederholt und liegt damit unter den 10 Wiederholungen, die bei AES mit einer Schlüssellänge von 128-Bit üblich sind. Aufgrund der Ähnlichkeit zu AES kann AEGIS auf die Hardware Unterstützung AES-NI zurückgreifen[WP].

Ascon: Ascon wurde von Christoph Dobraunig, Maria Eichlseder, Florian Mendel und Martin Schläffer konstruiert. Durch die Struktur von Ascon und einer Ressourcen schonenden Rundenfunktion ist Ascon auf Anwendungsfall 1 spezialisiert und gehört zu den hardware optimierten Stromchiffren[Dob+16].

COLM: COLM[4] ist einer von 2 Finalisten, die für Anwendungsfall 3 ausgewählt wurden. Es ist eine Blockchiffre, die ebenfalls wie AEGIS auf der Rundenfunktion von AES aufgebaut ist[And+16].

Deoxys-II: Ein weiterer Algorithmus von den Finalisten, der auf den Missbrauch des Initialisierungsvektors spezialisiert ist. Veröffentlicht wurde Deoxys-II von: Jérémy Jean, Ivica Nikolić, Thomas Peyrin und Yannick Seurin. Ähnlich wie bei COLM basiert

[1] https://groups.google.com/forum/!topic/crypto-competitions/DLv193SPSDc
[2] Es ist nicht die letzte Rundenfunktion gemeint.
[3] Bart Preneel als auch Hongjun Wu sind beide Auschussmitglieder von der CAESAR Competition.
[4] Die Autoren von Colm sind: Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, Kan Yasuda

auch Deoxys-II auf der AES Rundenfunktion und kann somit auch von der AES-NI Hardware Unterstützung profitieren[Jea+16].

MORUS: MORUS ist eine Stromchiffre, die von Hongjun Wu und Tao Huang eingereicht worden ist. Morus ist schnell auf der Hardware, da nur hardwarefreundliche Operationen wie AND und XOR in der Rundenfunktion vom Algorithmus verwendet werden. Der Algorithmus erreicht ebenfalls gute Resultate in der Software, weil Morus effizient mit den SSE Instruktionen implementiert werden kann. Daher wird Morus für den Anwendungsfall 2 empfohlen[WH].

OCB: Ein weiterer schneller Verschlüsselungsalgorithmus ist von Ted Krovetz und Phillip Rogaway für den Anwendungsfall 2 vorgeschlagen worden. OCB orientiert sich an dem AES Verschlüsselungsverfahren und nutzt infolgedessen die Rundenfunktion von AES. Die Chiffre ist in [KR14] standardisiert worden. Der Standardisierungsprozess von OCB hatte sich verzögert, da Verfahren, die im Algorithmus benutzt wurden, patentiert sind. Mittlerweile ist es erlaubt, OCB zu benutzen unter der Bedingung, dass entweder die GNU General Public License verwendet oder die Chiffre nicht kommerziell benutzt wird[KR16].

3.1.2 Weitere vielversprechende AEAD Algorithmen

NORX: NORX wurde für die Hardware als auch für die Software optimiert. Die Cipher wurde von Jean-Philippe Aumasson, Philipp Jovanovic und Samuel Neves konstruiert. Der Grund, weshalb NORX hardwarefreundlich ist, kann man auf die genutzten Operationen im Algorithmus zurückführen. Es werden ausschließlich XORs, ANDs ,SHIFTs und Rotationen benutzt. NORX kann durch die SSE Instruktionen sehr effizient in Software implementiert werden. Als nicht generische Chiffre wurde NORX auch in die CAESAR Competition eingereicht. Allerdings ist NORX nach der zweiten Runde ausgeschieden[AJN15].

ChaCha20-POLY1305: Die von Daniel J. Bernstein erstellte Chiffre wurde bereits in RFC7539 standardisiert. CHACHA20-POLY1305 ist ein generisches AEAD Verschlüsselungsverfahren. Es besteht aus der Stromchiffre ChaCha20 und dem Poly1305 MAC. ChaCha20 ist eine verbesserte Variante von der ebenfalls von Bernstein konstruierten Salsa20 Chiffre. Im Gegensatz zu AES ist ChaCha20 nicht anfällig gegenüber timing attacks. Zudem ist ChaCha20 bis zu dreimal schneller, wenn das unterliegende System keine AES-NI Hardware Unterstützung besitzt[Ber08]. Der Message Authentikation Code Poly1305 wurde als erstes in Kombination mit AES als AEAD Algorithmus benutzt. Poly1305-AES bietet eine höhere Sicherheit und ist wesentlich schneller als andere bekannte generische AEAD Algorithmen wie z.B AES in Kombination mit CBC-MAC[Ber05].

3.2 CAESAR Performance Analyse

Alle Algorithmen des CAESAR Wettbewerbs werden veröffentlicht, damit andere Spezialisten die Algorithmen auf Schwächen und Performance analysieren können. Daher wurden diverse wissenschaftliche Artikel zu dem Wettbewerb publiziert, in denen die Sicherheit oder Performance der einzelnen Algorithmen untersucht worden ist.

3.2.1 Software Benchmarking

In [AA16] wurden alle Kandidaten aus der 2. Runde, darunter auch die Finalisten, der CAESAR Competition, analysiert. Bei der genutzten Benchmarksoftware wurde nur die Software Performance getestet. Demnach werden die Clockzyklen von einer CPU gezählt, die benötigt werden, um eine Operation eines Verschlüsselungsalgorithmus auszuführen. Zudem werden unterschiedlich große Nachrichten und AD verschlüsselt, um möglichst unterschiedliche und reale Bedingungen zu simulieren. In der Publikation wird insbesondere darauf eingegangen, welche Auswirkungen Befehlssätze auf unterschiedliche Verschlüsselungsalgorithmen haben. So wird unter anderem erwähnt, dass aufgrund der vielen AES-basierten Algorithmen die Performance durch den AES-NI Befehlssatz im Schnitt um das 2-3 Fache auf der Softwareebene gesteigert werden kann. Von den Autoren gibt es kein Fazit, welche Chiffre am besten abgeschnitten haben. Vielmehr stellen sie eine Übersicht bereit, lassen jedoch die Ergebnisse weitestgehend unkommentiert. Sie weisen lediglich darauf hin, dass die Chiffren, die software-optimierte Implementationen besitzen, wesentlich bessere Ergebnisse erzielen. Gleichwohl fällt es auf, dass die CAESAR Finalisten AEGIS und MORUS durchschnittlich die besten Resultate erreichen. Das MORUS so hervorragende Ergebnisse erzielt ist überraschend, da MORUS eine Chiffre ist, die das Hauptaugenmerk auf die Hardwareoptimierung gelegt hat.

SUPERCOP

System for Unified Performance Evaluation related to Cryptographic Operations and Primitives (SUPERCOP) ist ein Benchmarking Toolkit für kryptographische Chiffren und wurde bereits in mehreren Projekten wie eBASH[5], eBASC[6], eBAEAD[7]verwendet. Da auch eingereichte CAESAR Chiffren von den Autoren in SUPERCOP hinzugefügt werden müssen, besitzt SUPERCOP eine außerordentlich große Bibliothek an Verschlüsselungsalgorithmen zum Testen[Ber09]. Um genaue Ergebnisse zu erreichen nutzt SUPERCOP den Time Stamp Counter, ein 64-Bit Register, das die Anzahl der CPU Takte zählt. Wenn ein Test gestartet wird, so wird das Register auf 0 gesetzt und zählt die CPU Takte bis der Algorithmus beendet wird. Die Tests wurden auf unterschiedlichen Plattformen untersucht und sind in [Ber18] aufzufinden. Hier spiegeln sich die Resultate aus[AA16] wieder. Die softwareoptimierten Implementationen der Algorithmen erreichen weitaus bessere Ergebnisse als ohne Optimierung.

[5] ECRYPT Benchmarking of All Submitted Hashes
[6] ECRYPT Benchmarking of Stream Ciphers
[7] ECRYPT Benchmarking of Authenticated Ciphers

Algorithmus	Schlüssellänge	IV Länge	Blockgröße	Patentfrei	Anwendungsfall	Softwareoptimierung
ACORN-128 v3	128	128	128	Ja	1	x
AEGIS128L v1.1	128	128	128	Ja	2	AES-NI
Ascon-128 v1.2	128	128	64	Ja	1	x
COLM v1	128	64	128	Nein	3	AES-NI
DEOXYS-II-128-128 v1.14	128	128	120	Ja	3	AES-NI
MORUS1280-128 v2	128	128	x	Ja	2	SSE, AVX
OCB-128-128	128	128	x	Nein	2	AES-NI, SSE
NORX-32-4-1	128	128	x	Ja	1	AVX,AVX2
ChaCha20-Poly1305	256	96	x	Ja	x	AVX2
AES-GCM	128	96	128	Ja	x	AES-NI

Tabelle 3.1: In der Tabelle findet man eine Übersicht der empfohlenen Algorithmen aus der CAESAR Competition mit den von den Autoren vorgeschlagenen Parametern.

3.2.2 Hardware Benchmarking

Ein großes Problem bei Benchmarking von Hardware ist die Unterschiedlichkeit der Plattformen. Es gibt keine einheitliche Übereinstimmung auf welcher Hardware eine Chiffre getestet wird. Somit sind zwei kryptographische Algorithmen nicht miteinander vergleichbar, falls sie auf unterschiedlichen Systemen mit unterschiedlicher Hardware und anderen Hardware Unterstützungen getestet wurden. Die CAESAR Competition bietet eine Hardware Application Programming Interface (API) an, wodurch der Aufwand, um die verschiedenen Algorithmen auf der gleichen Plattform zu vergleichen, wesentlich vereinfacht wird. In [Tem+18] wurden alle CAESAR Kandidaten aus der dritten Runde auf der gleichen Hardware mittels der von CAESAR zur Verfügung gestellten Hardware API getestet. Dabei wurde ein Zynq-7000 System on Chip (SoC) mit zwei ARM Cortex-A9, ein programmierbaren Xilinx FPGA und ein AXI Interface verwendet. Um die Resultate besser nachvollziehen zu können, wurde neben den 11 CAESAR Kandidaten aus der 3. Runde eine zusätzliche ”dummy1”Chiffre implementiert, die hauptsächlich aus einer simplen XOR Funktion besteht. Dadurch wird der Hardware Overhead vom Präprozessor, Postprozessor und weiteren Hardwarekomponenten gemessen. In den Ergebnissen wird eine Übersicht über die Geschwindigkeit, der benötigte Speicher während der Berechnung und Stromverbrauch auf der Hardware bereitgestellt. Beim Vergleichen fällt auf, dass AEGIS128L durchaus viel Speicher im Vergleich zu anderen Chiffren benötigt, aber AEGIS128L dafür auch der schnellste Verschlüsselungsalgorithmus ist. Dazu kommt, dass die Algorithmen, die ungefähr die gleiche Performance in Geschwindigkeit erreichen,einen höheren Stromverbrauch und mindestens genauso viel Speicher benötigen.

3.3 Ausgewählte Verschlüsselungsalgorithmen

Das Ziel der Arbeit ist, mögliche Algorithmen für MACsec zu finden, die in den Kategorien Durchsatz, Latenz oder CPU Auslastung bessere Resultate als AES-GCM erzielen. Daher sind die Algorithmen vom Anwendungsfall 2 in der CAESAR Competition besonders vielversprechend. Da diese für 64-Bit-Architekturen optimiert worden sind und heutzutage 64-Bit-Architekturen am meisten verbreitet sind. Es wurden AEGIS128L

und MORUS640 von den CAESAR Finalisten ausgewählt. Laut 3.2.1 gehören MORUS640 und AEGIS128L zu den schnellensten Chiffren von den Finalisten. Außerdem können beide Chiffren sehr effizient in Software implementiert werden, da AEGIS128L von der AES-NI Hardware Unterstützung profitieren kann und MORUS640 ist gut parallelisierbar, weil MORUS640 umso mehr von der SSE Hardware Unterstützung profitiert. Ebenso wurde ChaCha20Poly1305 ausgesucht. Dieser generische AEAD Algorithmus genießt bereits große Beliebtheit und wird bereits als effiziente Alternative benutzt. In der Tabelle 3.1 findet man eine Übersicht über die vorgestellten Algorithmen und deren Eigenschaften. Des Weiteren werden in der Tabelle auch die Hardware Unterstützungen aufgelistet, die der jeweilige Algorithmus benutzt, um seine Performance zu verbessern. Im nächsten Abschnitt werden die 3 ausgesuchten Algorithmen detaillierter erläutert.

3.3.1 AEGIS

AEGIS ist eine Stromchiffre, die von Hongjun Wu und Bart Preneel veröffentlicht worden ist. Für AEGIS wurden 3 verschiedene Varianten vorgestellt AEGIS128L, AEGIS128 und AEGIS256. Alle 3 Implementationen sind für den Anwendungsfall 2 vorgeschlagen worden, wobei laut den Autoren AEGIS128L die schnellste von den 3 Implementationen ist. Die AEGIS Familie basiert auf der AES-Rundenfunktion und profitiert sowohl von den AES-NI Instruktionen, als auch von den Möglichkeiten der Parallelisierung, die AES besitzt. Durch die weite Verbreitung von AES werden über die Zeit neue Instruktionen wie z.B. PCLMULQDQ[8] zu AES-NI oder die AES-Rundenfunktion in alle x86 Prozessoren von Intel hinzugefügt. Das sind alles Vorteile, von denen auch die AEGIS Algorithmen profitieren. Zudem nutzt AEGIS neben der AES Rundenfunktion auch XORs und ANDs, welche hardwarefreundliche Operationen sind. Die Rundenfunktion[9] zum Aktualisieren des Zustands wird pro Variante von AEGIS unterschiedlich oft wiederholt.
Im folgenden wird ein kompletter Verschlüsselungsvorgang von AEGIS128L beschrieben. AEGIS256 und AEGIS128 unterscheiden sich nur in einigen strukturellen Details und in den Wiederholungen der Rundenfunktionen voneinander. AEGIS128L kann in 4 verschiedene Abschnitte geteilt werden:

1. Initialisierung: Der Initialisierungsvektor und der geheime Schlüssel werden mit fest definierten Konstanten in den internen Zustand des Algorithmus geladen. Danach wird 10 mal die Rundenfunktion durchlaufen, um den Zustand zu aktualisieren. Am Ende jeder Rundenfunktion wird der Schlüssel als auch der Initialisierungsvektor zum Zustand XORed.

2. Associated Data: In diesem Schritt wird die AD zum internen Zustand des Algorithmus hinzugefügt. Dabei wird die AD in diesem Vorgang nicht verschlüsselt. Wenn die AD kein Vielfaches von 256 Bits ist, dann wird die AD mit 0 aufgefüllt.

3. Verschlüsselung: In diesem Abschnitt findet die eigentliche Verschlüsselung statt. Es werden zwei[10] 16-Bytes große Nachrichtenblöcke mit dem Bitstrom, der aus

[8] PCLMULQDQ ist ein übertragsfreier Multiplikationsbefehl
[9] Die Wiederholung findet 5 mal bei AEGIS128, 6 mal bei AEGIS256 und 8 mal bei AEGIS128L statt.
[10] In AEGIS128 und AEGIS256 wird nur ein 16-Byte großer Nachrichtenblock verschlüsselt.

dem internen Zustand von AEGIS generiert wird, XORed. Daraus entsteht das Chiffrat. Im Anschluss wird der Nachrichtenblock ein weiteres Mal mit dem internen Zustand XORed, um den Zustand bis zum nächsten Verschlüsselungsschritt zu aktualisieren. Auch hier werden die Nachrichten mit 0 aufgefüllt, sollte ein Nachrichtenblock kürzer als 16-Byte sein.

4. Finalisierung: Als letztes wird der Message Authentication Code an die Nachricht konkateniert. Dies wird bewerkstelligt, indem die Rundenfunktion von AEGIS durchlaufen wird, um den Zustand zu aktualisieren. Anschließend wird die Länge von der AD und der Nachricht zum Zustand XORed. Dann wird ein Block aus dem internen Zustand entnommen und als Tag an die Nachricht angeheftet.

Die Entschlüsselung von AEGIS funktioniert genau wie die Verschlüsselung, nur das anstelle des Nachrichtenblocks, das zu entschlüsselnde Chiffrat als Input genommen wird.
Laut den Autoren benötigt AEGIS nur die Hälfte der Berechnungszeit von AES-GCM. Außerdem kann durch den Designansatz der Tag ohne großen Mehraufwand berechnet werden. Obwohl AEGIS auf der AES Rundenfunktion aufbaut, kann trotzdem eine höhere TAG Sicherheit erreicht werden, als die von AES-GCM. Trotz einer gründlichen Analyse von möglichen Angriffen auf AEGIS gibt es zum jetzigen Zeitpunkt noch keinen Sicherheitsbeweis von AEGIS. Nichtsdestotrotz ist AEGIS als Finalist der CAESAR Challenge ausgewählt worden. Deshalb kann man davon ausgehen, dass die Chiffre noch nicht gebrochen worden ist und die Experten aus der Kommission noch keinen Angriff gefunden haben, um die Sicherheit von AEGIS signifikant zu verringern.[WP14][Mos17]

3.3.2 MORUS

Die Stromchiffre MORUS wurde von Hongjun Wu und Tao Huang entwickelt. Von dem MORUS Algorithmus gibt es 3 verschiedene Varianten, deren Parametersets in Größe des Zustandsraums und Schlüsselgröße unterscheiden. In diesem Algorithmus gibt es eine Aktualisierungsfunktion, die mit den SSE Instruktionen optimiert werden können. Dadurch erreicht MORUS eine geringere cpb[11] als AES-GCM. In MORUS werden nur logische Gatter wie XOR, AND und bitweise Verschiebungen genutzt. Diese Designentscheidung macht MORUS zu einer Hardware schonenden Chiffre. MORUS besitzt den gleichen strukturellen Aufbau wie AEGIS. Abgesehen von der Aktualisierungsfunktion für den internen Zustand unterscheiden sich MORUS und AEGIS nur in Details in der Initialisierung und Finalisierung. Aus diesem Grund wird die Beschreibung eines Verschlüsselungsvorgangs nicht wiederholt. Es werden 5 Register[12]benutzt. Das Prinzip ist hier das gleiche wie bei AEGIS. Es gibt einen geheimen internen Zustand, der mittels der Rundenfunktion aktualisiert wird. Während des Verschlüsselungsvorgangs werden „Schlüssel“-Blöcke aus dem geheimen internen Zustand generiert und mit den Nachrichtenblöcken bitweise addiert. Das Ergebnis ist der verschlüsselte Nachrichtenblock.

[11] Clock cycles per byte. Das ist eine Einheit, um die Performance eines Algorithmus zu darzustellen
[12] Die Registergröße unterscheidet sich je nach Variante. MORUS640 nutzt 128-Bit große Register und MORUS1280 256 große Register.

Folgende Vorteile werden von den Autoren der MORUS Chiffren hervorgehoben: MORUS ist einer der schnellsten AEAD Algorithmen ohne auf das Design der AES Rundenfunktion zu setzen. Dies hat zur Folge, dass MORUS plattformübergreifend eine konstante Performance aufweist. Außerdem wird mit MORUS640-128 ein Algorithmus angeboten, der durch seine geringen Zustandsgröße auch für niedrige Bit-Architekturen geeignet ist. Zudem können MORUS und AEGIS eine stärkere Sicherheit des Message Authentikation Codes als AES-GCM garantieren. Ein weiterer Vorteil ist, dass mit dem Design des Algorithmus der Message Authentikation Code ohne großen Aufwand berechnet werden kann. Auch für MORUS existiert noch kein formaler Beweis, der die Sicherheit des Algorithmus bestätigt[WH][Mos17].

3.3.3 ChaCha20-Poly1305

Chacha20-Poly1305 ist ein generischer AEAD Verschlüsselungsalgorithmus. Der Algorithmus besteht aus den beiden Bausteinen ChaCha20, eine Stromchiffre und den Message Authentikation Code Poly1305. ChaCha20-Poly1305 gewinnt an zunehmender Beliebtheit, da die Chiffre unabhängig von der Plattform eine konstant gute Performance aufweisen kann. Der Entwickler der unterliegenden generischen Primitive ist Daniel J. Bernstein, der 2008 zum ersten Mal eine verbesserte Variante von Salsa20, genannt ChaCha20, vorstellte[Ber08]. Kurz vorher hat er ein wissenschaftliches Dokument über Poly1305 in der Kombination mit AES, genannt Poly1305-AES [Ber05], veröffentlicht, in der Bernstein seinen Message Authentikation Code vorstellt. 2015 wurde ChaCha20-Poly1305 von Google in [NL15] standardisiert mit dem Gedanken, eine effiziente Alternative zu AES bieten zu können [Lan13] [13]. Einen formalen Sicherheitsbeweis kann man unter [Pro14] finden. ChaCha20-Poly1305 verfolgt den Encrypt-then-MAC Ansatz und benutzt einen 256-Bit langen Schüssel mit einem 96-Bit langen Initialisierungsvektor. Im Anschluss wird die Nachricht von ChaCha20 mit dem gleichen Schlüssel, einem IV und einen Counter verschlüsselt. Sobald die Nachricht verschlüsselt ist, bildet Poly1305 den Tag wie folgt: Aus dem 256 langem Bit Schlüssel wird ein Schlüssel für den Poly1305 Authentikator gebildet. Poly1305 nutzt zwar einen 256 Bit langen Schlüssel, der Tag von Poly1305 ist allerdings nur 128-Bit lang. Poly1305 verlangt einen fest definierten Input, der diese Reihenfolge besitzen muss:

1. Associated Data
2. Mittels Padding wird die Nachricht mit 0 wird auf ein Vielfaches von 16 aufgefüllt.
3. Ciphertext
4. Die Länge der Associated Data
5. Die Länge der Nachricht

Wenn dieser Vorgang abgeschlossen ist, dann wird der Tag berechnet. Hierbei wird eine Nachricht in Vielfache von 16-Bytes geteilt. Die Blöcke werden nun als Zahl interpretiert

[13] In [Lan13] war noch von Salsa20 die Rede. In Laufe der Zeit scheint man sich für ChaCha20 entschieden zu haben.

und mittels little-endian gelesen. Jeder einzelne Block wird nun um ein Bit vergrößert. Mathematisch gesehen ist das eine Addition von 2^{128}. Ist bis zu diesem Zeitpunkt der Nachrichtenblock nicht 17 Bytes lang, so wird Padding benutzt[14]. Die Blöcke werden iterativ mit dem Zwischenergebnis addiert[15]. Wenn dieser Vorgang abgeschlossen ist, wird das Ergebnis mit r mod $2^{130} - 5$ multipliziert. Als letzter Schritt wird das Ergebnis mit dem Wert s[16] bitweise addiert [NL15].

[14] Es mit wird mit 0 aufgefüllt. Dadurch das die Bytes mittels little-endian gelesen werden, ändert das Padding nichts an dem eigentlichen Wert der Nachricht.
[15] Der erste Block wird mit 0 addiert
[16] r und s sind Variablen, die als Resultat der Schlüsselgenerierung, gebildet werden

4 Implementation

In Kapitel 3 wurde begründet, welche Algorithmen für die Implementation in den Sicherheitsstandard ausgewählt wurden. Im Folgenden wird erklärt, auf welchem aktuellen Stand sich die MACsec Implementierung auf dem Linux Kernel befindet und welche Änderungen an dem bestehenden Quellcode gemacht werden müssen, um weitere Algorithmen hinzuzufügen.

4.1 Linux

Linux ist ein Betriebssystem, welches maßgeblich von Linus Torvalds geprägt worden ist. Eine Besonderheit an Linux ist, dass es von mehreren Unternehmen weiterentwickelt wird, die regelmäßig das bestehende Betriebssystem erweitern, indem neue Versionen für das Betriebssystem und austauschbare Linux-Kernel veröffentlicht werden.
Ein Linux-Kernel ist das Herzstück des Betriebssystems und stellt Schnittstellen für die Software bereit, um auf die Hardware eines Systems zugreifen zu können[17].

Linux Krypto API

Die Linux Kernel Application Programming Interface ist eine Programmierschnittstelle, die durch den Linux-Kernel bereitgestellt wird. Mit der Linux Krypto API haben Kernel Treiber wie MACsec Zugriff auf alle Verschlüsselungsalgorithmen, die durch den Linux-Kernel unterstützt werden. Außerdem überprüft die Linux Krypto API das System und kann somit automatisch die softwareoptimierten Implementationen der Verschlüsselungsalgorithmen auswählen, falls das unterliegende System Hardware Unterstützungen anbietet. Es werden unterschiedliche "Cipher APIs" angeboten für verschiedene Algorithmentypen:

- Blockchiffren
- Stromchiffren
- AEAD Algorithmen
- Hash Funktionen
- Zufallszahlengeneratoren

[17] Die unterschiedlichen Kernel-Versionen, die von Linux unterstützt werden, sind unter kernel.org aufzufinden.

Damit ein Verschlüsselungsalgorithmus erfolgreich aufgerufen werden kann, muss ein transformation object (tfm) initialisiert werden. Das tfm ist eine Instanz der Implementation eines Verschlüsselungsalgorithmus. Oft wird so ein Objekt auch "Cipher Handle"genannt. Ein Aufruf eines Verschlüsselungsalgorithmus muss somit folgende Schritte absolvieren.

1. Initialisierung eines tfm
2. Anwendung der Verschlüsselungsoperation
3. Zerstörung des tfm

Jede Cipher API stellt eigene Funktionen zur Verfügung, um ein tfm zu initialisieren und auszuführen. Im Folgenden werden die wichtigsten Funktionen aus der AEAD Cipher API vorgestellt, die in MACsec Verwendung finden:

- crypto_alloc_aead initialisiert die Cipher Handle.
- crypto_free_aead zerstört die Cipher Handle.
- crypto_aead_ivsize gibt den Initialisierungsvektor zurück.
- crypto_aead_authsize konfiguriert die Größe des Tags.
- crypto_aead_setkey konfiguriert den Schlüssel und Schlüssellänge.
- crypto_aead_encrypt startet die Verschlüsselung.
- crypto_aead_decrypt startet die Entschlüsselung.
- aead_request_set_crypt nimmt die zu verschlüsselnden/entschlüsselnden Daten entgegen und konkateniert die Daten intern hinter die AD.
- aead_request_set_tfm allokiert Speicher für das tfm.
- aead_request_set_ad konfiguriert die Größe der AD.

Wie bereits erwähnt, muss als erstes ein tfm initialisiert werden. Dafür wird die Funktion crypto_alloc_aead aufgerufen, die mit einem String den richtigen Algorithmus auswählt und initialisiert. Doch bevor es zu einer korrekten Anwendung der Verschlüsselungsoperation kommen kann, müssen einige Vorbereitungen gemacht werden. Es muss die crypto_aead_setkey Funktion aufgerufen werden. Damit wird der Cipher Handle die Schlüssellänge und der zu nutzende Schlüssel mitgeteilt. Ebenfalls muss die Tag Größe mit der crypto_aead_setauthsize Funktion definiert werden. In dem MACsec Modul werden diese Vorbereitungen direkt nach der Initialisierung in der Funktion macsec_alloc_tfm umgesetzt.
Nachdem Schlüssel und Tag konfiguriert worden sind, muss der Speicher für die Cipher Handle allokiert werden. Normalerweise wird hierfür die aead_request_alloc Funktion aufgerufen, die ein tfm entgegennimmt und automatisch den Speicher allokiert. Da das

MACsec Modul mit unterschiedlich großen Nachrichten arbeitet und die Nachrichtengröße ebenfalls in die Allokation mit einbezogen werden muss, wird in der Funktion macsec_alloc_req der benötigte Speicher errechnet und anschließend mit aead_request_set_tfm allokiert.
Erst jetzt ist es möglich, die zu verschlüsselnden/entschlüsselnden Daten für den Algorithmus bereitzustellen. Die MACsec AEAD Cipher API nutzt für die zu bearbeitenden Daten eine Speicherstruktur namens scatter-gather Liste. Die scatter-gather Liste setzt Pointer auf die Daten. Das erleichtert den Umgang mit größeren Datenmengen, da nur eine scatter-gather Liste mit den Pointern übergeben werden muss. In aead_request_set_crypt werden die scatter-gather Liste, der Initialisierungsvektor und die Länge der Nachricht an den Verschlüsselungsalgorithmus übergeben. Außerdem wird noch die Länge der AD benötigt. Das wird mit der Funktion aead_request_set_ad umgesetzt. In aead_request_set_callback wird eine Funktion entgegengenommen, die aufgerufen wird, wenn die der Verschlüsselungsvorgang/ Entschlüsselungsvorgang beendet wird. Erst wenn diese Vorbereitungen abgeschlossen sind und alle Funktionen aufgerufen worden sind, ist es möglich, die Verschlüsselungsoperation mit crypto_aead_encrypt oder crypto_aead_decrypt zu starten. Das MACsec Modul hat seine eigene Verschlüsselungsfunktion namens macsec_encrypt in der viele Maßnahmen präpariert werden, um eine erfolgreiche Verschlüsselung durchführen zu können. Neben der Ausführung der AEAD Cipher API Funktionen muss MACsec in der macsec_encrypt und macsec_decrypt den Initialisierungsvektor bilden, den Header des Rahmens gemäß der MACsec Definitionen aus 2.4.1 verändern und die scatter-gather Liste initialisieren [S M18] [Mos17].

4.2 MACsec Kernel Modul

Obwohl die Publikation des MACsec Sicherheitsstandards bereits 2006 erschienen ist, wurde die Implementation des Standards erst 2016 in den Linux-Kernel aufgenommen. Einen Überblick über den strukturellen Aufbau des MACsec Treibers wird von der Autorin des Standards Sabrina Dubroca in [Dub16b] gegeben. Wird ein Rahmen über MACsec gesendet, so wird die Funktion macsec_start_xmit aufgerufen. In dieser Funktion wird sowohl der komplette Prozess der Verschlüsselung als auch die Modifizierung des Rahmens durch MACsec an kleinere Teilfunktionen dirigiert. Die Funktion übernimmt 2 Parameter, einen Socket Buffer sk_buff mit dem bis zu 64.000 Bytes große Nachrichten verschickt werden können und ein net_device mit dem die SecY initialisiert wird. Im Verlauf von macsec_start_xmit wird die macsec_encrypt Funktion aufgerufen. Dort wird die entscheidende Verschlüsselung vorbereitet und ausgeführt. Erhält MACsec ein Rahmen, so wird die Funktion macsec_handle_frame aufgerufen. In dieser Funktion wird der Rahmen auf mögliche Änderungen überprüft. Je nach Sicherheitseinstellungen wird ein Paket abgewiesen oder weiter bearbeitet. Aber auch wenn eine zu große Zeitspanne zwischen dem Senden und dem Erhalten des Rahmens vergangen ist, wird die Nachricht abgewiesen. Erst nach einem erfolgreichen Abschluss aller Überprüfungen erfolgt die Entschlüsselung der Methode macsec_decrypt.

4.3 iproute2

iproute2 ist ein Linux Werkzeug, dass zurzeit von Stephen Hemminger bearbeitet wird. Intern nutzt iproute2 einen Linux Service namens netlink, um mit den Netzwerkstack des Kernels zu kommunizieren. Für die Konfiguration von MACsec bietet iproute2 eine Reihe von Befehlen an, die unter [Dub16a] zu finden sind. Mit der aktuellen Implementierung kann man bei iproute2 den Verschlüsselungsalgorithmus von MACsec auswählen. Mittels dem Optional Cipher besteht die Möglichkeit, „default[18]“ und „aes-gcm-128“ auszuwählen. Um mehrere Algorithmen nutzen zu können, muss das iproute2 erweitert werden, damit die hinzugefügten Algorithmen auswählbar und von MACsec erkannt werden.
Allerdings treten dabei drei essentielle Probleme in der aktuellen Implementierung auf:

1. Die MACsec Cipher Suite unterstützt nur einen Algorithmus.
2. Die derzeitige SecY im MACsec Modul kann nicht erkennen, welcher Algorithmus benutzt wird. Um zwischen verschiedenen Algorithmen auswählen zu können, muss daher die SecY von MACsec erweitert werden.
3. Das aktuelle MACsec Modul im Linux Kernel ist statisch programmiert und genau auf die Parameter von AES-GCM-128 angepasst. Daher muss die Implementierung dynamischer gestaltet werden, da nicht alle Verschlüsselungsalgorithmen genau die gleichen Parameter von AES-GCM verwenden.

Die MACsec Cipher Suite ist eine Liste von statischen Variablen und Structs, in denen Parameter, die MACsec vom Verschlüsselungsalgorithmus verlangt, definiert werden. Die Erweiterung der Cipher Suite ist daher simpel, da nur eine statische Variable mit einer einzigartigen Identifikationsnummer für den hinzugefügten Algorithmus erstellt werden muss.

iproute2 Schnittstelle

Mit dem Cipher Optional wird ein String entgegen genommen. Daraufhin überprüft das iproute2, ob es zu dem String eine Identifikationsnummer aus der Cipher Suite von MACsec gibt. Ist das der Fall, so wird die Identifikationsnummer an das MACsec übermittelt.

4.4 MACsec Erweiterung

Als ein großes Problem in der Programmierung stellte sich heraus, dass MACsec kategorisch unterschiedliche Algorithmen mit anderen Parametersätzen wie z.B. einer anderen Schlüssellänge als 128-Bit ablehnt. Zum einen liegt das daran, dass MACsec eine Reihe von Sicherheitsvorkehrungen hat, die vor der Initialisierung eines Algorithmus genau überprüfen, ob die Schlüssellänge exakt mit den Parametern von AES-GCM-128

[18] Wenn default ausgewählt wird, dann wird automatisch AES-GCM mit einer Schlüssellänge von 128 Bits verwendet.

```
1  struct macsec_secy {
2  struct net_device *netdev;
3  unsigned int n_rx_sc;
4  sci_t sci;
5  u16 key_len;
6  u16 icv_len;
7  enum macsec_validation_type validate_frames;
8  bool operational;
9  bool protect_frames;
10 bool replay_protect;
11 u32 replay_window;
12 struct macsec_tx_sc tx_sc;
13 struct macsec_rx_sc __rcu *rx_sc;
14 u64 csid;
15 };
16
```

Abbildung 4.1: MACsec Die Felder der Security Entity. Die Änderung an der Security Entity wurde mit rot markiert.

übereinstimmen. Die Sicherheitsüberprüfungen sind so komplex, dass noch keine Möglichkeit gefunden wurde, Algorithmen mit größeren Schlüsseln dynamisch in MACsec zu implementieren, ohne dabei die Sicherheit von MACsec zu gefährden. Zum anderen wurden in der aktuellen Implementation z.B. die Länge des Initialisierungsvektors mit einer statischen Variable der Länge von 12-Bytes definiert. Deshalb wurden alle Vorkommen der statischen Variable mit einer Funktion namens crypto_aead_ivsize() aus der Linux Kryptographie Bibliothek ausgetauscht. Die Funktion erkennt die initialisierte Chiffre und gibt die Länge des Initialisierungsvektors aus. In einigen Methoden muss Speicherkapazität für die Parameter allokiert werden. So wird garantiert, dass egal welcher Algorithmus benutzt wird, die richtige Größe des Initialisierungsvektors allokiert wird. Um Algorithmen mit einem variablen Initialisierungsvektor in MACsec unterstützen zu können, muss zudem der SecTag vergrößert werden, da der Initialisierungsvektor ein Bestandteil vom SecTag ist. Somit wird der SecTag um 4 Byte vergrößert, damit MACsec auch Algorithmen unterstützen kann, die einen Initialisierungsvektor mit einer Länge von 16-Bytes benötigen.

4.4.1 SecY Erweiterungen

Um mehrere Algorithmen auswählen zu können, muss die Security Entity von Macsec erweitert werden. Die SecY von MACsec wird mit der Funktion macsec_changelink_common initialisiert. In dieser Funktion werden die Parameter von der iproute2 Konfiguration an die SecY von MACsec übergeben. Im Anschluss validiert MACsec die überreichte

Identifikationsnummer des Verschlüsselungsalgorithmus aus der Cipher Suite. Wird kein passender Algorithmus gefunden, gibt MACsec eine Fehlermeldung an den Kernel zurück. Die Identifikationsnummer wird mit dem Feld u64 csid beschrieben. In dieser Arbeit wurde die SecY mit der u64 csid erweitert. So ist es möglich, die Identifikationsnummer der Chiffre zu speichern. Da die SecY maßgeblich am Verschlüsselungsprozess teilnimmt, kann man in den entscheidenden Schritten über die Identifikationsnummer prüfen und den richtigen Parameter auswählen. Die Struktur von der SecY ist in der Abbildung 4.1 zu sehen.

4.5 Herausforderungen

Die größte Herausforderung war die Fehlersuche im Kernel Modul. Tritt im Programm ein Fehler auf, kann trotzdem ein Paket verschickt werden. Das hat zur Folge, dass fehlerhafte Pakete erst beim Validieren von dem Programm bemerkt werden, obwohl die Ursache des Problems woanders liegt. Aufgrund dieser Tatsache hat es sich als äußerst schwer herausgestellt, den Ursprung eines Fehlers zu finden. Es hat einen hohen Zeitaufwand benötigt, um einzelne Probleme zu ermitteln, da nicht direkt der richtige Ansatz zum Lösen des Problems gefunden wurde.

4.5.1 MORUS640, AEGIS128L

Während der Tests mit den Programmen ping und iperf3 traten in 1 Prozent der Fälle Authentifizierungsfehler bei AEGIS128L und MORUS640 auf. Die Authentifizierungsfehler kamen zustande, weil in der Entschlüsselungsfunktion ein Buffer Stream einzelne ankommende Pakete in zwei geteilt hat. Daher wurde ein Buffer Overflow in einer Streaming Methode, die beim Entschlüsseln aufgerufen wird, vermutet. Aufgrund der Tatsache, dass die beiden Algorithmen erst kürzlich zum Linux Kernel hinzugefügt worden sind[19],schien es als wahrscheinlich, dass noch mögliche Fehler in der Implementierung vorhanden sind. Daher wurde der Autor der Implementation um Rat gefragt und es wurde versucht, den Fehler in der Linux Bibliothek libkcapi zu reproduzieren. Im Nachhinein hat sich herausgestellt, dass durch die statische Programmierung von MACsec der Initialisierungsvektor unbemerkt auf 12-Bit verkleinert wurde. Das hat zu kritischen Fehlern geführt, da MORUS640 und AEGIS128L einen Initialisierungsvektor mit der Länge von 16-Bit benötigen.

4.5.2 ChaCha20-Poly1305

ChaCha20-Poly1305 war der einzige Algorithmus, der sich nicht korrekt mittels crypto_alloc_aead initialisieren ließ. Es wurde erwähnt, dass ChaCha20-Poly1305 einen besonderen Input benötigt, um den Message Authentication Code bilden zu können[NL15]. Daher lag die Vermutung nahe, dass die interne Datenverarbeitung von MACsec diese Struktur nicht bereitstellen kann oder stattdessen eine andere Struktur benutzt. Im Endeffekt konnte ChaCha20-Poly1305 nicht initialisiert werden, da die MACsec Cipher

[19] `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=3e1a29b3bf66c2850ea8eba78c59c234921c0b69`

Suite keine Algorithmen mit einer 256-Bit Schlüssellänge unterstützt. Aus diesem Grund konnte kein gültiger SAK generiert werden, der für die korrekte Initialisierung des Algorithmus unverzichtbar ist. Das Problem wurde mit einem statischen Schlüssel behoben. Das hat zwar Auswirkungen auf die Sicherheit von MACsec, aber für die Performancemessungen ist dies unerheblich.

5 Evaluation

Es wurden nun unterschiedliche Verschlüsselungsalgorithmen vorgestellt und in das aktuelle implementiert. Im Folgenden werden Performance Tests durchgeführt, um zu sehen, ob dadurch eine höhere Datenübertragungsrate und niedrigere Latenz erreicht werden kann.
Leider war es nicht möglich, MORUS640 und AEGIS128L mit dem software-optimierten Implementationen zu testen. Die Implementationen werden erst ab dem Linux Kernel 4.18 unterstützt, der zum Zeitpunkt der Test noch nicht funktionsfähig zur Verfügung stand. Deshalb mussten die Implementationen zu einer älteren Kernel-Version hinzugefügt werden, wodurch noch keine Softwareoptimierungen unterstützt werden können. In 3.2.1 wurde gezeigt, wie sehr sich die Optimierungen auf die Performance auswirken. Daher ist zu erwarten, dass die fehlenden Optimierungen negative Auswirkungen auf die Latenz und Datenübertragungsrate von MORUS640 und AEGIS128L haben.

5.1 Testaufbau

Die Testergebnisse werden auf zwei Rechnern evaluiert, die mittels eines LAN-Kabels miteinander verbunden sind. Auf den Rechnern läuft folgende Hardware:

Prozessor:	Intel i5-4590 3.3GHz
RAM:	16GB
Betriebssystem:	Ubuntu 18.04 mit dem Linux-Kernel 4.15.0-32

Tabelle 5.1: In der Tabelle wird die Hardware des Systems aufgelistet.

Jeder Algorithmus wird mit verschiedenen Paketgrößen getestet, um reale Bedingungen zu simulieren. Dabei wird geschaut, ob die unterschiedlichen Paketgrößen Auswirkungen auf die Latenz oder die Datenübertragungsrate haben. Die Datenübertragungsrate kann mit dem Werkzeug iperf3 gemessen werden. Neben der Bandbreite sind die Latenz und die CPU Auslastung entscheidende Messkriterien. Dafür bietet sich das Werkzeug ping an. Damit lässt sich die Round-trip time feststellen. Das ist die zeitliche Übertragung eines festen Pakets zum Zielrechner und die Beantwortung des Pakets vom Zielrechner. Folgende Tests werden durchgeführt:

1. MACsec wird mit den Verschlüsselungsalgorithmen AES-GCM, ChaCha20-Poly1305 , AEGIS128L und MORUS640 mit ping getestet. Die verwendeten Paketgrößen sind 128, 256, 512, 1024, 1400 und 1514 Bytes. Die Tests werden sowohl mit Verschlüsselung als auch ohne Verschlüsselung ausgeführt.

2. MACsec wird mit den Verschlüsselungsalgorithmen AES-GCM, ChaCha20-Poly1305 , AEGIS128L und MORUS640 mit iperf3 getestet. Die verwendeten

Paketgrößen sind 128, 256, 512, 1024, 1400 und 1514 Bytes. Die Tests werden sowohl mit Verschlüsselung als auch ohne Verschlüsselung ausgeführt.

3. Ethernet wird mit den Paketgrößen 128, 256, 512, 1024, 1400 und 1514 Bytes mit iperf3 und ping getestet.

Die verwendeten Paketgrößen sind gängige Größen, um Verschlüsselungsalgorithmen zu testen. Daher werden diese Werte übernommen, um leichter Vergleiche zu anderen Arbeiten ziehen zu können. Die Auswertungen werden mit einem Bash-Skript ausgeführt, welches auf beiden Testrechnern die Konfigurationen mit den unterschiedlichen Paketgrößen und Verschlüsselungsalgorithmen via Secure Shell (SSH) aktualisiert und die Analyse mit ping oder iperf3 startet. Iperf3 sendet in einem Zeitraum von 10 Sekunden Pakete an einen Server, der auf dem Empfängerrechner konfiguriert wurde und misst jede Sekunde den durchschnittlichen Datendurchsatz. Die Resultate der Testsoftware werden in eine Datei geschrieben und gespeichert. In eine Datei wird jeweils nur ein Test mit einem Verschlüsselungsalgorithmus und nur einer Paketgröße geschrieben. Im Zuge der Datenanalyse hat sich herausgestellt, dass die Daten leichter zu untersuchen sind, wenn sie sich in einer Datei befinden. Daher wurden im Nachhinein alle Dateien eines Verschlüsselungsalgorithmus mit unterschiedlichen Paketgrößen konkateniert und zu einer Datei gebündelt, was zu einer Erleichterung der Datenanalyse führte.
Die iperf3 Tests wurden mit AES-GCM und ChaCha20-Poly1305 für jede Paketgröße 250 mal wiederholt. Die Evaluation von MORUS640 und AEGIS128L konnten aufgrund von Implementationsproblemen erst später durchgeführt werden und wurden aus Zeitgründen 100 mal für jede Paketgröße wiederholt.
Die ping Tests wurden 50000 mal für jede Paketgröße wiederholt. Normalerweise wird eine ping Anfrage in einem festen Intervall gesendet, der eine Sekunde beträgt. Mit der adaptiven ping Option kann der Testvorgang beschleunigt werden, indem die ping Anfragen sofort gesendet werden, sobald die vorherige Anfrage beantwortet wurde.

5.1.1 Datendurchsatz

In Abbildung 5.1 und 5.2 sind die Ergebnisse der Evaluation in Diagrammen dargestellt. Aus den Diagrammen ist ersichtlich, dass es keinen Unterschied macht, ob MACsec die Daten verschlüsselt und dann den TAG bildet oder die Verschlüsselung auslässt und nur den TAG generiert. Das ist damit zu erklären, dass bei AEAD Algorithmen die Verschlüsselung und Tag Bildung in der gleichen Verschlüsselungsrunde durchgeführt werden. Deshalb benötigt ein Algorithmus trotzdem die gleiche Anzahl an Zyklen, um einen TAG zu einer Nachricht zu bilden, egal ob Verschlüsselung angewendet wird oder nicht. Des Weiteren sieht man, dass noch Verbesserungspotenzial im Vergleich zur Ethernet möglich ist, wenn ein Verschlüsselungsalgorithmus mit einer niedrigen Paketgröße ausgewählt wurde. Allerdings kann jeder Algorithmus die Leitung komplett auslasten, sobald eine Paketlänge von 1024 Bytes erreicht wird. AES-GCM und ChaCha20-Poly1305 haben sehr ähnliche Testergebnisse. ChaCha20-Poly1305 kann also von der Performance durchaus eine Alternative zu AES-GCM sein. MORUS640 und AEGIS128L schließen ab einer Paketlänge von 1024 Bytes mit ChaCha20-Poly1305 und AES-GCM auf. Bei

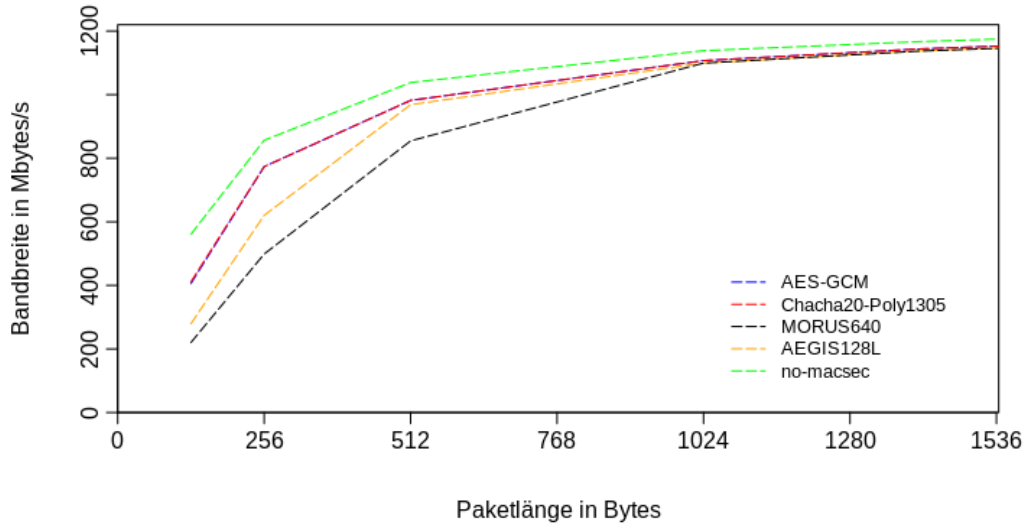

Abbildung 5.1: In dem Diagramm wird die Bandbreite der Verschlüsselungsalgorithmen in mbytes/s dargestellt. Die Test wurden mit aktivierter Verschlüsselung durchgeführt.

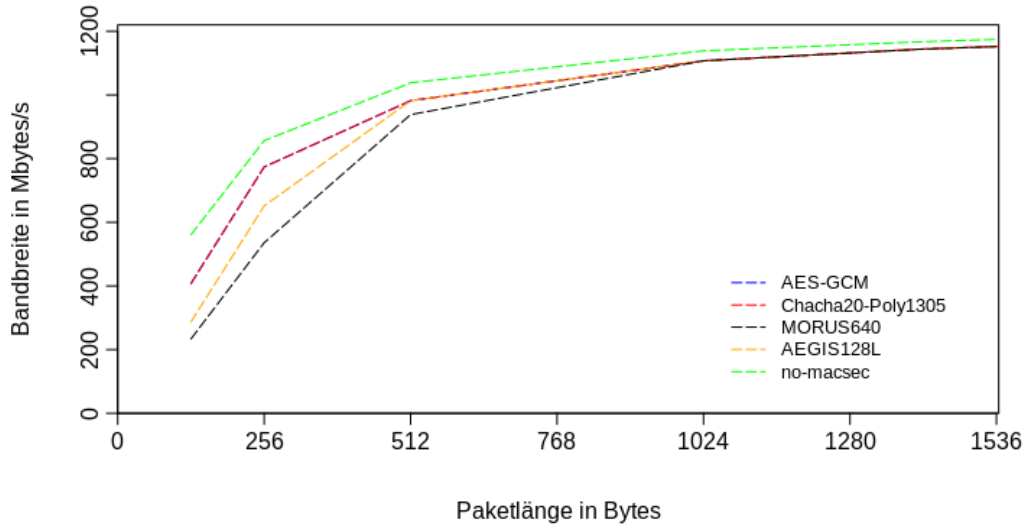

Abbildung 5.2: In dem Diagramm wird die Bandbreite der Verschlüsselungsalgorithmen in mbytes/s dargestellt. Die Test wurden ohne Verschlüsselung durchgeführt.

niedrigen Paketlängen erreicht MORUS640 knapp die Hälfte des Durchsatzes von AES-GCM. AEGIS128L ist im Vergleich zu MORUS640 ein wenig schneller und kann bereits ab einer Paketlänge von 512 Bytes zu AES-GCM und ChaCha20-Poly1305 aufholen. Die genauen Testresultate findet man in der Appendix in den Tabellen 7.1 und 7.2 .

5.1.2 CPU Auslastung

Während der iperf3 Testvorgänge wurde gleichzeitig die CPU Auslastung des Computers ermittelt. Die Testergebnisse sind in den Diagrammen 5.3 und 5.4 aufzufinden. Zusätzlich gibt es in der Appendix in den Tabellen 7.3 und 7.4 die genauen Resultate. Die Daten zeigen, dass alle hinzugefügten Algorithmen eine hohe CPU Auslastung bei niedrigen Paketgrößen aufweisen. Bei MORUS640 ist ein sogenannter Bottleneck bei den Paketgrößen erkennbar. Das heißt, dass MORUS640 die CPU komplett auslastet und somit die Performance von MORUS640 durch die CPU gedrosselt wird. Die CPU Auslastung bei MORUS640 liegt bei den Paketgrößen 128, 256 und 512 Bytes im Schnitt bei 96,8 Prozent. Die enorme Belastung der CPU in den niedrigen Paketgrößen lässt sich dadurch erklären, dass bei geringen Paketgrößen die Verschlüsselungsalgorithmen wesentlich häufiger aufgerufen werden müssen als bei großen Paketgrößen. Der häufige Funktionsaufruf führt auch bei ChaCha20-Poly1305 zu einer hohen CPU Auslastung. Diese liegt im Durchschnitt bei 96,6 Prozent, wenn eine Paketlänge von 128 benutzt wird. Einzig und allein AES-GCM hat durchgängig eine durchschnittliche CPU Auslastung von knapp einem Prozent und liegt damit fortwährend sehr nah an der CPU Auslastung von Ethernet und erzielt in einigen Fällen sogar bessere Ergebnisse als die Ethernet. Dies lässt sich nur durch Messungenauigkeiten in den Resultaten erklären.

5.1.3 Latenz

Die Paketumlaufzeit ist in der Appendix in den Tabellen 7.5 und 7.6 aufzufinden. Auch bei der Latenz weist AES-GCM die besten Resultate auf mit einer durchschnittlichen Zeit von 0,143 ms bei einer Paketgröße von 128 Bytes und 0,147 ms bei 1514 Bytes. Die Latenz ist nur minimal größer als die von Ethernet mit 0,136 ms bei 128 Bytes und ist in einigen Fällen sogar etwas besser mit 0,15 bei 1514 Bytes. Das ist erneut durch eine Messungenauigkeit erklärbar. MORUS640 und AEGIS128L haben mit Abstand die höchste Latenz, die im Schnitt bei 0,162 ms bei AEGIS128L und 0,16 ms bei MORUS640 liegt. AEGIS128L ist somit 10 Prozent langsamer als AES-GCM mit einer durchschnittlichen Zeit von 0,145 ms. Ein Grund dafür könnte die fehlende Software Optimierung sein. Daher wurden diese Ergebnisse, wie bereits in der Einleitung des Kapitels erwähnt, erwartet. Auffallend ist die geringe Varianz von ChaCha20-Poly1305 und AES-GCM. Die durchschnittliche Paketumlaufzeit zwischen den Paketlängen hat bei ChaCha20-Poly1305 nur eine Differenz von 0,005 ms und ist nur noch bei AES-GCM minimal geringer mit 0,004 ms. Daraus lässt sich eine konstante Performance von AES-GCM und ChaCha20-Poly1305 erkennen. Die Diagramme sind in der Appendix 7.1 aufzufinden.

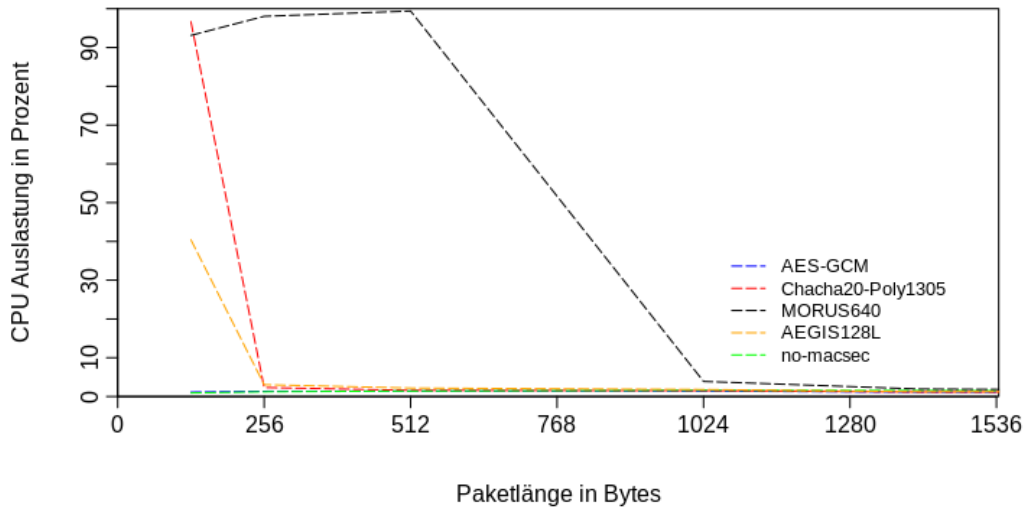

Abbildung 5.3: Das Diagramm zeigt die CPU Auslastung in Prozent der einzelnen Verschlüsselungsalgorithmen, Während der Tests wurde die Verschlüsselung aktiviert.

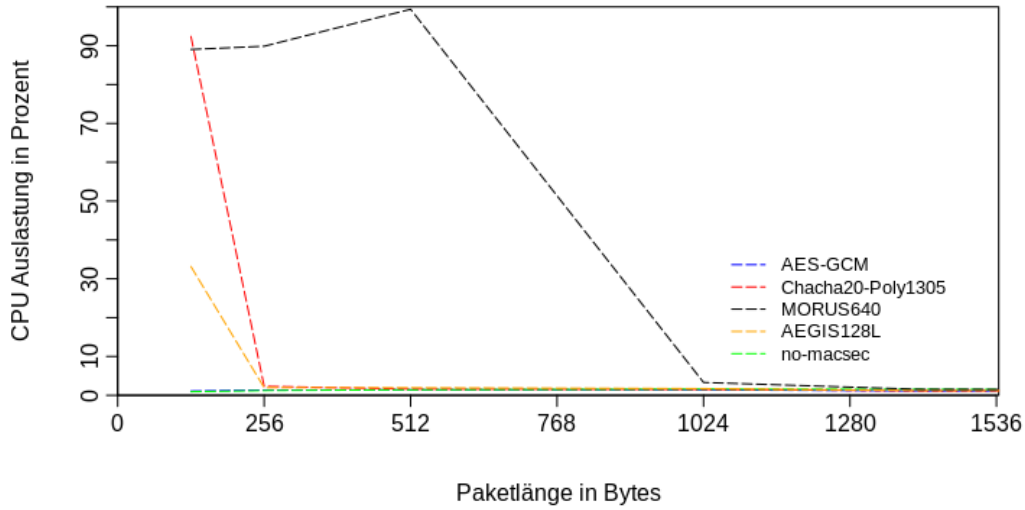

Abbildung 5.4: Das Diagramm zeigt die CPU Auslastung in Prozent der einzelnen Verschlüsselungsalgorithmen, Während der Tests wurde keine Verschlüsselung verwendet.

5.2 Sicherheitsevaluation

Der Sicherheitsstandard MACsec garantiert folgende Sicherheitsziele: Vertraulichkeit, Integrität, Authentizität, bounded receive delays und Schutz vor Replay Angriffen. Im folgenden Abschnitt wird auf die Änderungen eingegangen, die in Kapitel 4 beschrieben worden sind und ob diese Auswirkungen auf die genannten Sicherheitsmerkmale haben. Im Rahmen der Bachelorarbeit wird es nicht möglich sein, komplette Sicherheitsanalysen der einzelnen Algorithmen vorzustellen, daher werden nur die wichtigsten Kriterien behandelt. Im kompletten Abschnitt wird davon ausgegangen, dass die Algorithmen korrekt verwendet werden.

5.2.1 Vertraulichkeit

Da ein Verschlüsselungsalgorithmus maßgeblich für das Sicherheitsziel der Vertraulichkeit verantwortlich ist, muss man sehr sorgfältig überprüfen, ob die gemachten Änderungen einen Einfluss auf die Sicherheit haben. Es wurden insgesamt 3 weitere Verschlüsselungsalgorithmen implementiert.

ChaCha20-Poly1305

ChaCha20-Poly1305 besitzt eine Schlüssellänge von 256-Bits und ist somit doppelt so lang wie die Schlüssellänge von AES-GCM. Desweiteren bietet ChaCha20-Poly1305 wenig Angriffsfläche gegenüber ”timing-attacks”, die bei AES-GCM gefunden wurden[BM06]. Somit kann sogar eine höhere Sicherheit der Vertraulichkeit gewährleistet werden als bei AES-GCM. An dieser Stelle muss erwähnt werden, dass die Implementation von ChaCha20-Poly1305 nur mit einem statischen Schlüssel erfolgen konnte. In diesem Fall wirkt sich der geringe Initialisierungsvektor negativ aus, da mit einem Initialisierungsvektor der Länge von 96-Bit nur ca 256 Gigabyte verschlüsselt werden können, ohne das sich der Initialisierungsvektor wiederholt[NL15]. In Anbetracht der Tatsache, dass die Implementierung des Autors von ChaCha20-Poly1305 nur einen Prototyp darstellt, um Performance-Analysen ausführen zu können, ist die Schwächung der Sicherheit von MACsec durch einen statischen Schlüssel vernachlässigbar, da eine korrekte Implementierung von ChaCha20-Poly1305 die Sicherheit des MACsec Moduls nicht beeinträchtigen würde.

AEGIS128L, MORUS640

AEGIS128L und MORUS640 benutzen jeweils eine Schlüssellänge von 128-Bits. In der wissenschaftlichen Publikation von AEGIS128L[WP14] und MORUS640[WH] ist eine genaue Sicherheitsanalyse zu den Algorithmen zu finden. Allerdings existiert noch kein formaler Sicherheitsbeweis für die beiden Algorithmen. Dies könnte daran liegen, dass die Algorithmen erst 2014 publiziert worden sind. Aus diesem Grund können AEGIS128L und MORUS640 noch nicht zu den wohluntersuchten Algorithmen gezählt werden. Das hat keine negativen Auswirkungen auf das Sicherheitsziel der Vertraulichkeit, aber bei der Anwendung von AEGIS128L und MORUS640. Dementsprechend

sollte bei der Anwendung von AEGIS128L und MORUS640 weniger Vertrauen entgegengebracht werden.

5.2.2 Integrität

Da der Betriebsmodus AEAD verwendet wird, wird auch Message Authentication Code ausgetauscht, sobald eine andere Chiffre verwendet wird. Deshalb muss der verwendete Message Authentication Code der einzelnen Chiffren betrachtet werden. Der Aufbau der Message Authentication Codes wurde bereits in 3.3.1 und 3.3.3 genauer erläutert.

Poly1305 Die Nutzung von Poly1305 hat keinen negativen Effekt auf die Integrität des Sicherheitsstandards.

AEGIS128L, MORUS640

Trotz eines fehlenden Sicherheitsbeweises wird in [WP14] und [WH] explizit hervorgehoben, dass AEGIS128L und MORUS640 eine 128-Bit Sicherheit für Authentikation bereitstellen können, die höher ist als die von AES-GCM. Demzufolge wird die Sicherheit der Integrität bei der Anwendung von MORUS640 und AEGIS128L verstärkt.

Authentizität

Da die Authentizität durch den ICV des MACsec Standards gesichert und der ICV durch den Message Authentication Code gebildet wird, schwächt dies nicht die Authentizität, weil in diesem Abschnitt gezeigt wurde, dass ein Austausch des Verschlüsselungsalgorithmus keine negativen Auswirkungen auf die Integrität ausübt.

5.2.3 Verfügbarkeit

Im Sicherheitsstandard wurden nur zusätzliche Verschlüsselungsalgorithmen hinzugefügt. Diese Änderungen können Auswirkungen auf die Vertraulichkeit oder Integrität haben. Allerdings hat ein anderer Verschlüsselungsalgorithmus keinen Effekt auf die Verfügbarkeit, solange das MACsec Modul korrekt verwendet wird und kein aktiver Angriff auf das System vorliegt. Darüber hinaus hat ein aktiver Angriff keine Auswirkungen auf die Funktionsweise des Verschlüsselungsalgorithmus.

5.2.4 Replay Angriffe

MACsec schützt vor Replay Angriffen, in dem jedes versendete Paket eine Paketnummer zugeteilt bekommt, die nach jedem Paket inkrementiert wird. Wiederholt sich eine Paketnummer, so wird das Paket abgelehnt. Zwar wurden aufgrund der Paketnummer Änderungen am SecTag vorgenommen, das beschränkt sich allerdings nur auf die Vergrößerung des Paketnummer Feldes, um größere Initialisierungsvektoren zu unterstützen. Trotzdem wird der vergrößerte SecTag weiterhin komplett vom ICV geschützt, sodass keine neuen Angriffsmöglichkeiten auf den SecTag bestehen.

5.3 Resultate

Die Ergebnisse fallen durchmischt aus. ChaCha20-Poly1305 kann größtenteils bei der Bandbreite und CPU Auslastung mit AES-GCM mithalten. Auch bei der Latenz ist ChaCha20-Poly1305 im Schnitt nur 4,1 Prozent langsamer als AES-GCM, was ein tolerierbarer Wert ist. Anders sehen die Ergebnisse bei AEGIS128L und MORUS640 aus. Die Daten zeigen, dass die Latenz von MORUS640 und AEGIS128L weit hinter denen von ChaCha20-Poly1305 und AES-GCM liegen. Wenn die Bandbreite betrachtet wird, können AEGIS128L und MORUS640 bei geringen Paketgrößen auch nicht mithalten. Trotzdem muss erwähnt werden, dass Software Optimierungen große Auswirkungen auf einen Algorithmus haben können[AA16] und sowohl AEGIS128L als auch MORUS640 zum Zeitpunkt der Experimente keinen Zugriff auf deren Optimierungen hatten. In Anbetracht dieser Tatsache fallen die Ergebnisse von MORUS640 und AEGIS128L durchaus akzeptabel aus. Insgesamt zeigt sich, dass auf dieser Testumgebung die aktuelle ChaCha20-Poly1305 Implementation durchaus eine ernsthafte Alternative zu AES-GCM bieten kann.

6 Fazit

Das Ziel dieser Arbeit war es, das MACsec Modul mit zusätzlichen Verschlüsselungsalgorithmen zu erweitern. Dabei wurden unterschiedliche Algorithmen vorgestellt und analysiert. Anschließend wurde eine Auswahl aus den vielversprechendsten Algorithmen getroffen, die daraufhin in ein erweitertes MACsec implementiert wurden. Damit unterschiedliche Chiffren unterstützt werden, musste das Kernel Modul dynamischer gestaltet und unter anderem der SecTag vergrößert werden. Mögliche Schwächen und Sicherheitsmängel der Änderungen sind angesprochen worden.
Im Anschluss erfolgten die Performance Tests der einzelnen Chiffren, in dem die Latenz, Datenübertragungsrate und die Auslastung der CPU gemessen wurden. Um einen Vergleichswert zu haben, wurden ebenfalls Messungen mit normalen Ethernet Paketen gemacht.
Die Ergebnisse zeigen, dass ChaCha20-Poly1305 durchaus als Alternative zu AES-GCM genommen werden kann. Die Ergebnisse in der Bandbreite sind fast identisch und die CPU Auslastung befindet sich ebenfalls im gleichen Bereich. Nur bei der Paketumlaufzeit ist ChaCha20-Poly1305 im Durchschnitt 4 Prozent langsamer. Für einen Algorithmus, der nicht auf die AES-NI Hardware Unterstützung setzt, ist das ein gutes Ergebnis.
AEGIS128L und MORUS640 haben als Algorithmen viel Potential. Die Ergebnisse in dieser Arbeit spiegeln das nicht wider. Im Gegensatz dazu erreichen AEGIS128L und MORUS640 in anderen Publikationen[AA16] und Benchmarking Tools [Ber09] sogar bessere Ergebnisse in der Datenübertragung als AES-GCM. Ein Grund für das schlechte Abschneiden dieser Algorithmen könnte sein, dass es nicht möglich war, die Algorithmen mit ihren Softwareoptimierungen zu testen. Daher müssen in dieser Hinsicht noch mehr Nachforschungen betrieben werden.

7 Appendix

7.1 Iperf3 Testergebnisse

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes			in mbytes/s		
128	560.89	405.67	279.42	411.31	220.18
256	856.55	773.62	620.32	773.34	498.61
512	1038.22	981.99	968.6	981.89	854.6
1024	1138.45	1107.16	1099.26	1107.10	1099.25
1400	1136.97	1143.71	1137.6	1143.61	1136.93
1514	1145.89	1152.72	1146.48	1152.51	1145.89

Tabelle 7.1: Die Bandbreite in mbyte/s mit Verschlüsselung.

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes			in mbytes/s		
128	560.89	408.04	286.94	406.62	233.43
256	856.55	774.06	651.54	773.45	535.74
512	1038.22	982.04	1107.23	981.93	938.27
1024	1138.45	1107.12	1143.69	1107.16	1107.38
1400	1136.97	1143.67	1137.6	1143.65	1143.36
1514	1145.89	1152.69	1152.1	1152.52	1152.21

Tabelle 7.2: Die Bandbreite in mbyte/s ohne Verschlüsselung.

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes			in Prozent		
128	0.91	1.13	40.39	96.63	93.13
256	1.21	1.34	3.04	2.32	98.04
512	1.41	1.4	2.19	1.58	99.41
1024	1.60	1.39	1.81	1.51	3.88
1400	1.55	1.11	1.22	1.16	1.93
1514	1.5	1.11	1.22	1.15	1.84

Tabelle 7.3: Die CPU Auslastung in Prozent mit Verschlüsselung.

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes	in Prozent				
128	0.91	1.08	33.15	92.39	89.04
256	1.21	1.35	1.96	2.32	89.87
512	1.41	1.4	1.95	1.57	99.39
1024	1.60	1.41	1.72	1.5	3.3
1400	1.55	1.12	1.21	1.14	1.54
1514	1.5	1.12	1.2	1.15	1.51

Tabelle 7.4: Die CPU Auslastung in Prozent ohne Verschlüsselung.

7.2 ping Testergebnisse

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes			in ms		
128	0.136	0.143	0.147	0.149	0.152
256	0.14	0.144	0.148	0.15	0.154
512	0.142	0.144	0.161	0.152	0.16
1024	0.146	0.146	0.174	0.153	0.17
1514	0.15	0.147	0.181	0.154	0.177

Tabelle 7.5: Die durchschnittlichen Paketumlaufzeiten mit Verschlüsselung in ms.

Paketgröße	Baseline	AES-GCM	AEGIS128L	ChaCha20-Poly1305	MORUS640
in bytes			in ms		
128	0.136	0.141	0.148	0.148	0.15
256	0.14	0.145	0.147	0.148	0.153
512	0.142	0.149	0.15	0.149	0.157
1024	0.146	0.155	0.155	0.15	0.166
1514	0.15	0.161	0.159	0.151	0.17

Tabelle 7.6: Die durchschnittlichen Paketumlaufzeiten ohne Verschlüsselung in ms.

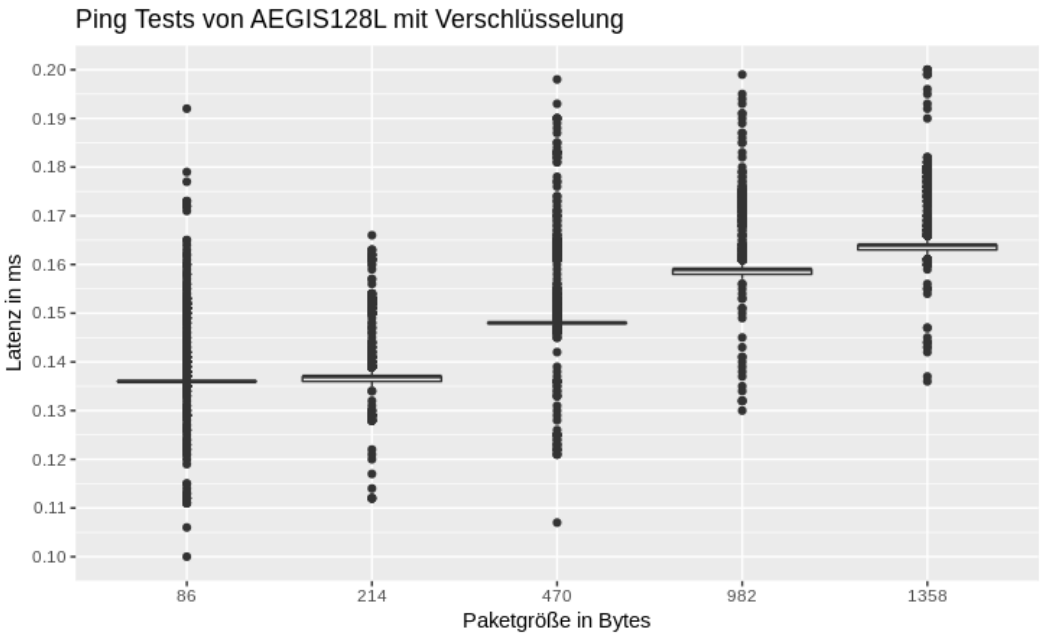

Abbildung 7.1: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus AEGIS128L. Es wurden 50.000 Tests durchgeführt. Es wurde mit Verschlüsselung getestet.

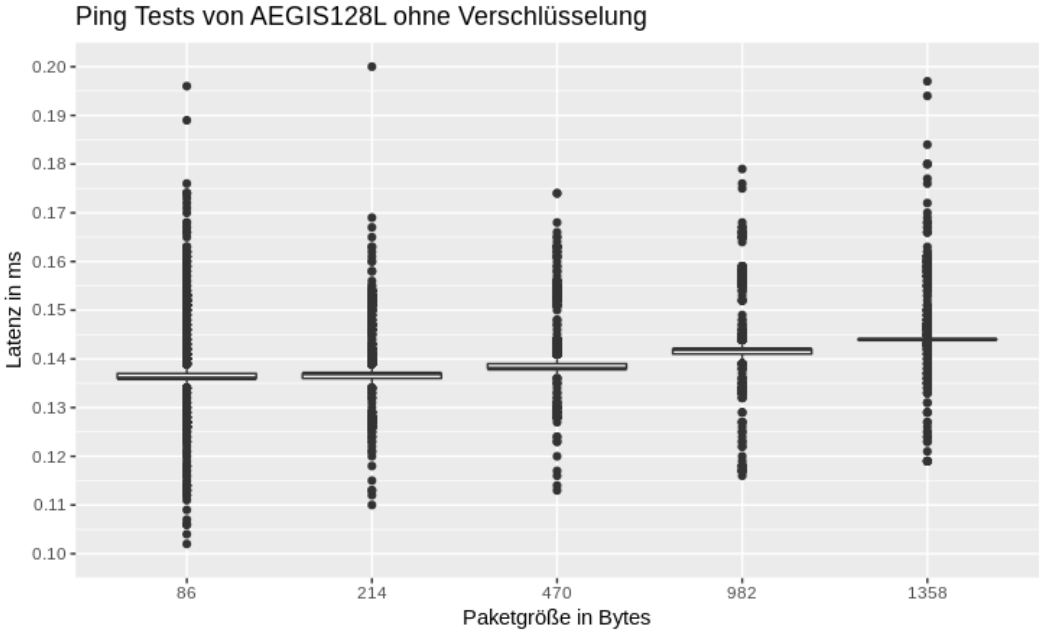

Abbildung 7.2: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus AEGIS128L. Es wurden 50.000 Tests durchgeführt. Es wurde ohne Verschlüsselung getestet.

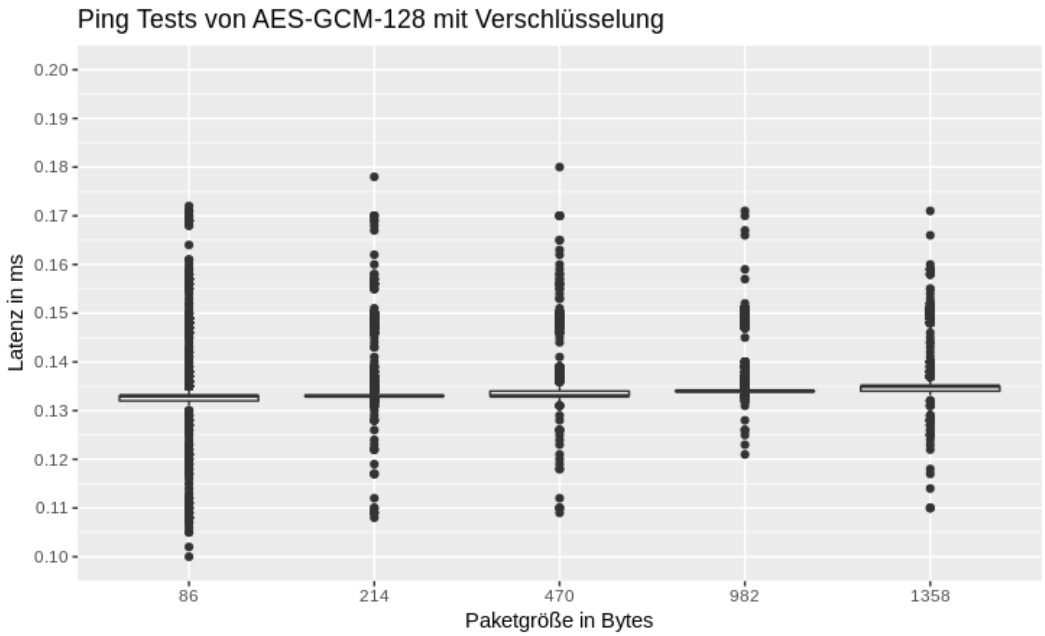

Abbildung 7.3: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus AES-GCM. Es wurden 50.000 Tests durchgeführt. Es wurde mit Verschlüsselung getestet.

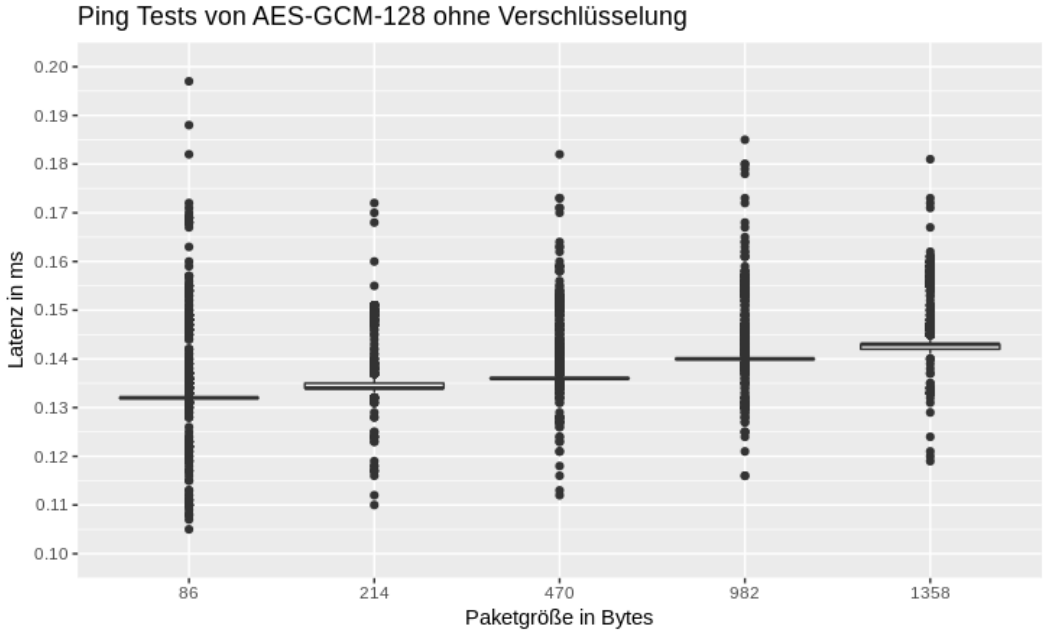

Abbildung 7.4: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus AES-GCM. Es wurden 50.000 Tests durchgeführt. Es wurde ohne Verschlüsselung getestet.

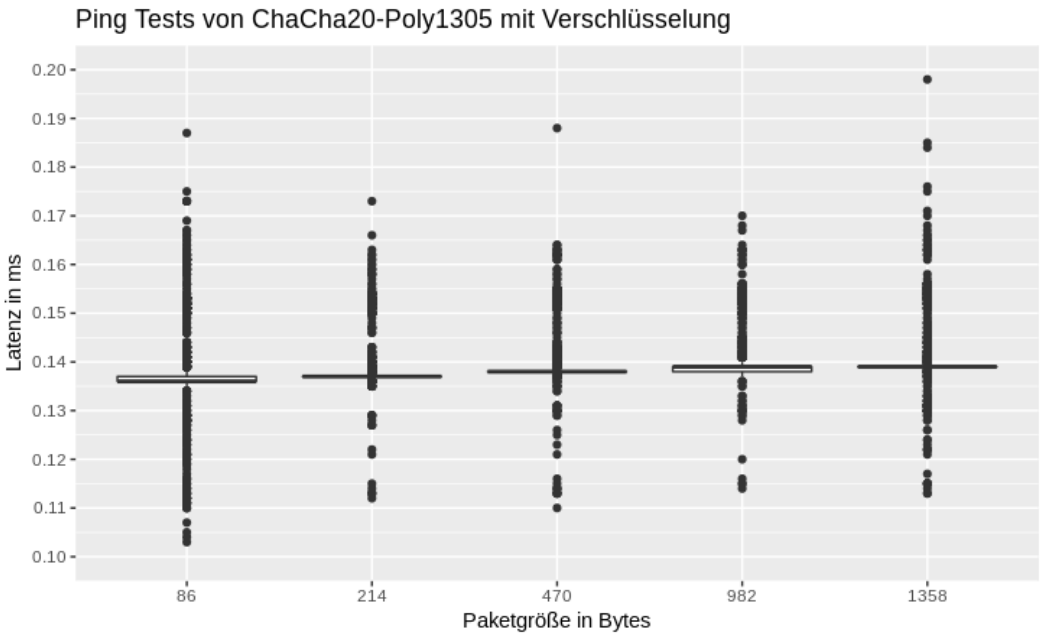

Abbildung 7.5: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus ChaCha20-Poly1305. Es wurden 50.000 Tests durchgeführt. Es wurde mit Verschlüsselung getestet.

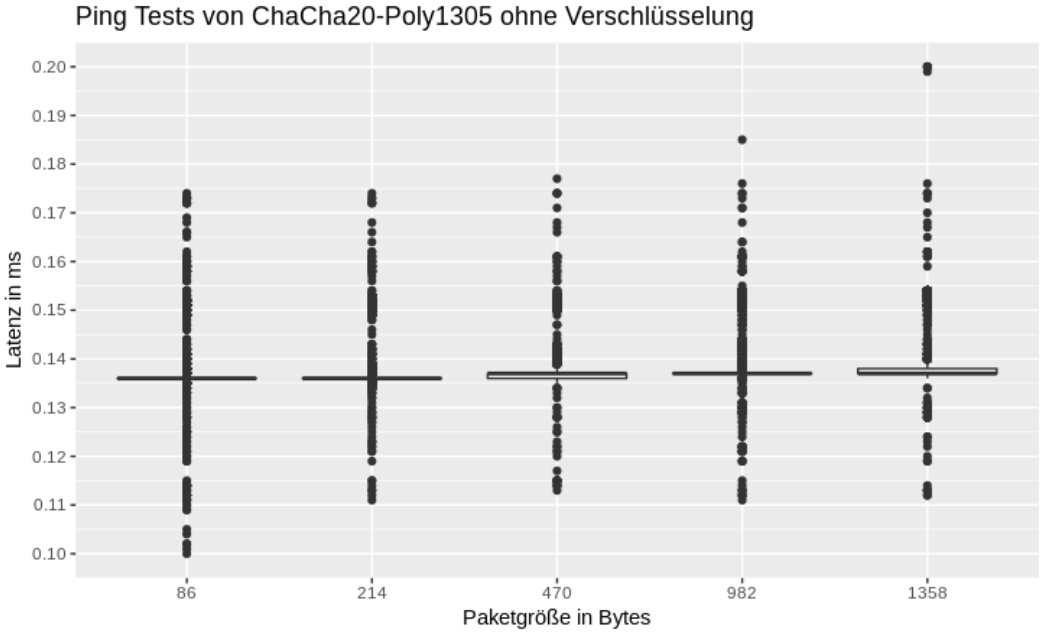

Abbildung 7.6: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus ChaCha20-Poly1305. Es wurden 50.000 Tests durchgeführt. Es wurde ohne Verschlüsselung getestet.

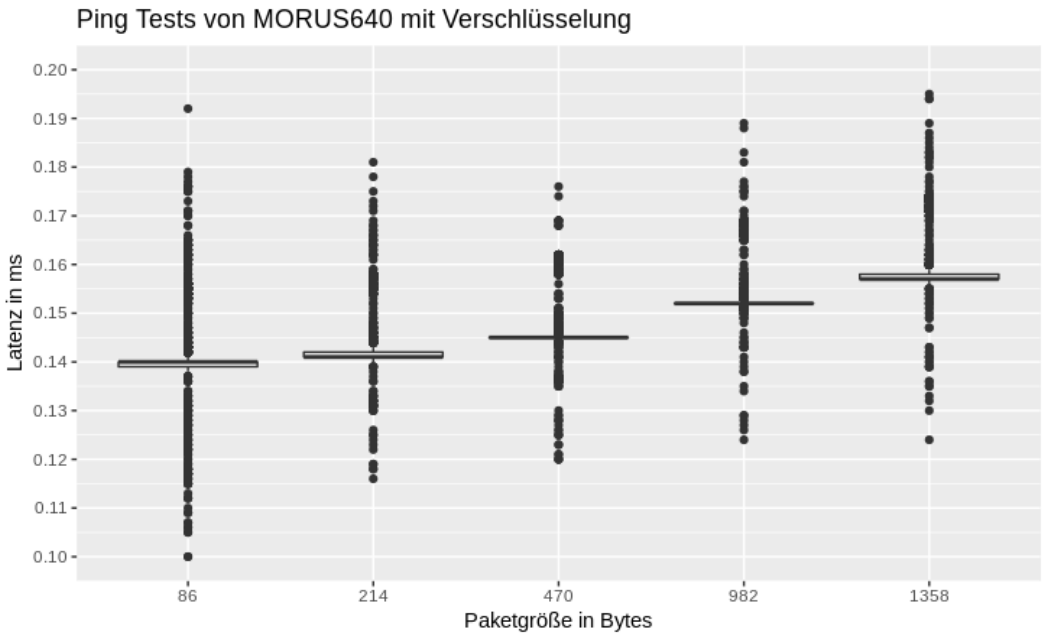

Abbildung 7.7: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus MORUS640. Es wurden 50.000 Tests durchgeführt. Es wurde mit Verschlüsselung getestet.

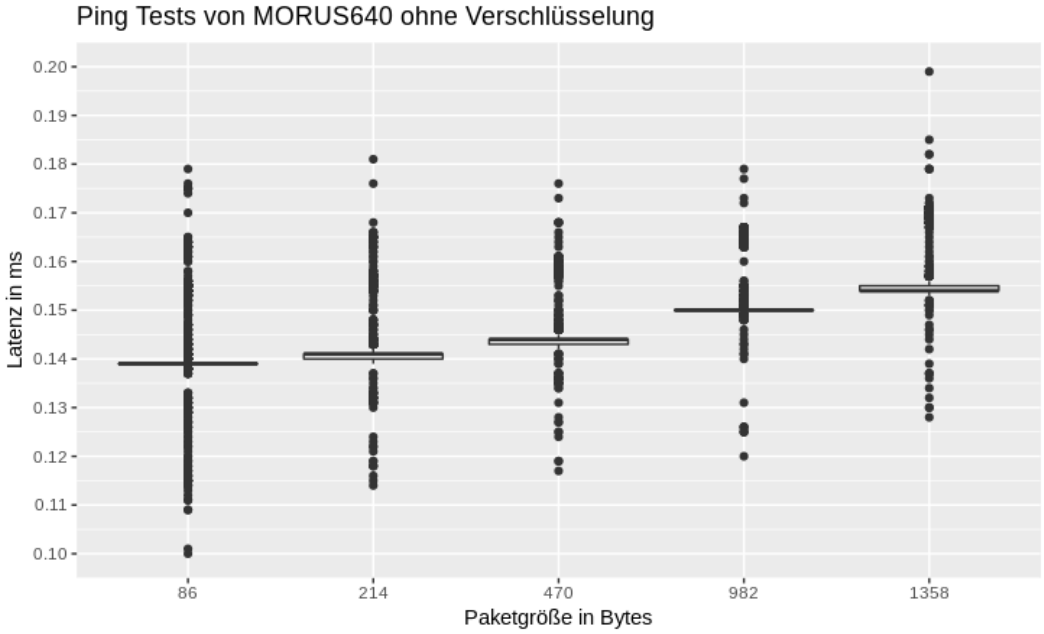

Abbildung 7.8: Das Boxplot Diagramm zeigt die Ping Tests mit dem Verschlüsselungsalgorithmus MORUS640. Es wurden 50.000 Tests durchgeführt. Es wurde ohne Verschlüsselung getestet.

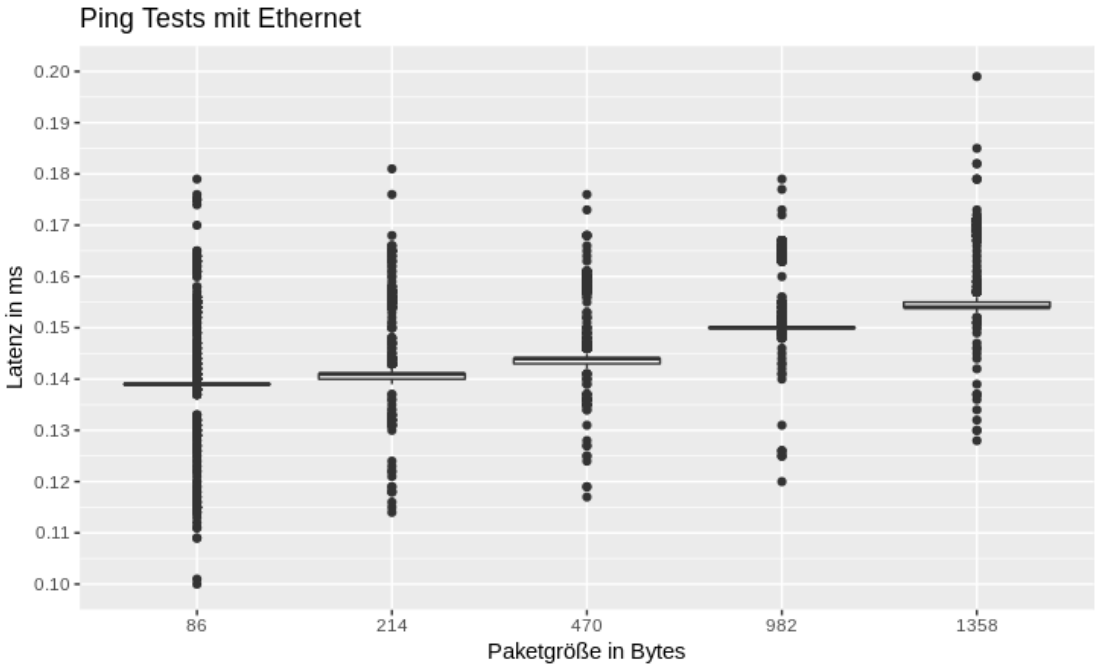

Abbildung 7.9: Das Boxplot Diagramm zeigt die Ping Tests mit normalen Ethernet Paketen. Es wurden 50.000 Tests durchgeführt.

Akronyme

AD	associated Data. 7, 8, 13, 14, 19, 21, 22
AEAD	Authenticated Encryption with Associated Data. 1, 7–9, 13, 17, 18, 21, 23, 33, 38
AES	Advanced Encryption Standard. 1, 13, 14, 16–19, 21, 23
AES-GCM	Advanced Encryption Standard with Galois Counter Mode. 1, 2, 7, 13, 16, 20, 22, 23, 28, 32, 33, 35, 37–40
AES-NI	Advanced Encryption Standard New Instructions. 1, 14, 17–19, 21, 40
AN	Association Number. 11, 12
API	Application Programming Interface. 20
C	Chiffrat. 7, 8
CA	Connectivity Association. 9, 10
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness. 16–22
CPU	Central Processing Unit. 16, 19, 20, 35, 39, 40
DES	Data Encryption Standard. 13
DoS	Denial of Service. 7
ES	End Station Bit. 12
ICMP	Internet Control Message Protocol. 5
ICV	Integrity Check Value. 10, 13, 14, 38
IEEE	Institute of Electrical and Electronics Engineers. 1, 4, 9
ISO	International Standards Organization. 3
IV	Initialisierungsvektor. 7, 8, 13, 23
K	symmetrischen Schlüssel. 7, 8
KaY	Security Key Agreement Entity. 9, 10
LLC	Logical Link Control. 3
M	Klartext. 7
MAC	Medium Access Control. 3, 4, 6

MACsec	Medium Access Control Security. 1–3, 7, 9–13, 17, 25, 27–29, 31–33, 37, 38, 40
MKA	MACsec Key Agreement Protokoll. 9, 12
NIST	National Institute of Standards and Technology. 13
OSI-Modell	Open Systems Interconnection Model. 1, 3–5, 9, 10
PPP	Point-to-Point Protocol. 12
SA	Secure Association. 10, 11, 13
SAK	Secure Association Key. 10, 11, 13, 31
SC	Secure Channel. 10, 11
SCI	Secure Channel Identifier. 10–13
SecY	MAC Security Entity. 9, 10, 12, 28, 29
SoC	System on Chip. 20
SSE	Streaming Simd Extensions. 14, 15, 18, 21, 22
SSH	Secure Shell. 33
SUPERCOP	System for Unified Performance Evaluation related to Cryptographic Operations and Primitives. 19
TCI	TAG Control Information Feld. 11, 12
TCP	Transmission Control Protocol. 5

Literatur

[06] „IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security“. In: *IEEE Std 802.1AE-2006* (Aug. 2006), S. 1–150. DOI: 10.1109/IEEESTD.2006.245590.

[10] „IEEE Standard for Local and metropolitan area networks–Port-Based Network Access Control“. In: *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)* (Feb. 2010), S. 1–205. DOI: 10.1109/IEEESTD.2010.5409813.

[11] „IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Security Amendment 1: Galois Counter Mode–Advanced Encryption Standard– 256 (GCM-AES-256) Cipher Suite“. In: *IEEE Std 802.1AEbn-2011 (Amendment to IEEE Std 802.1AE-2006)* (Okt. 2011), S. 1–52. DOI: 10.1109/IEEESTD.2011.6047536. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6047536.

[AA16] Ralph Ankele und Robin Ankele. „Software Benchmarking of the 2nd round CAESAR Candidates“. In: *IACR Cryptology ePrint Archive* 2016 (2016), S. 740. URL: https://eprint.iacr.org/2016/740.pdf.

[AJN15] Jean-Philippe Aumasson, Philipp Jovanovic und Samuel Neves. *NORX v3. 0. Submission to CAESAR (2016)*. 2015. URL: https://competitions.cr.yp.to/round3/norxv30.pdf.

[And+16] Elena Andreeva u. a. „COLM v1“. In: *CAESAR competition proposal* (2016). URL: https://competitions.cr.yp.to/round3/colmv1.pdf.

[Ber05] Daniel J. Bernstein. „The Poly1305-AES Message-Authentication Code“. In: *Fast Software Encryption*. Hrsg. von Henri Gilbert und Helena Handschuh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 32–49. ISBN: 978-3-540-31669-5. URL: https://link.springer.com/chapter/10.1007%2F11502760_3.

[Ber08] Daniel J Bernstein. „ChaCha, a variant of Salsa20“. In: *Workshop Record of SASC*. Bd. 8. 2008, S. 3–5. URL: http://cr.yp.to/chacha/chacha-20080120.pdf.

[Ber09] Daniel J Bernstein. *Supercop: System for unified performance evaluation related to cryptographic operations and primitives*. 2009. URL: https://bench.cr.yp.to/supercop.html.

[Ber14] Daniel J Bernstein. *Caesar: Competition for authenticated encryption: Security, applicability, and robustness*. 2014. URL: https://competitions.cr.yp.to/caesar.html.

[Ber18] Daniel J. Bernstein. *SuperCop AEAD List*. 2018. URL: https://bench.cr.yp.to/primitives-aead.html.

[Blu+18] *fastvpn – Secure and Flexible Networking for Industry 4.0*. Berlin, Germany.: VDE Verlag GmbH - Berlin - Offenbach, Apr. 2018, S. 28–35. ISBN: 978-3-8007-4637-8.

[BM06] Joseph Bonneau und Ilya Mironov. „Cache-Collision Timing Attacks Against AES“. In: *Cryptographic Hardware and Embedded Systems—CHES 2006*. Bd. 4249. Springer, Okt. 2006, S. 201–215. URL: https://www.microsoft.com/en-us/research/publication/cache-collision-timing-attacks-against-aes/.

[BN00] Mihir Bellare und Chanathip Namprempre. „Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm“. In: *Advances in Cryptology — ASIACRYPT 2000*. Hrsg. von Tatsuaki Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, S. 531–545. ISBN: 978-3-540-44448-0.

[Dob+16] Christoph Dobraunig u. a. „Ascon v1. 2“. In: *Submission to the CAESAR Competition* (2016). URL: https://competitions.cr.yp.to/round3/asconv12.pdf.

[Dub16a] Sabrina Dubroca. *ip-macsec - MACsec device configuration*. Hrsg. von Sabrina Dubroca. 7. Mai 2016. URL: https://manpages.debian.org/stretch/iproute2/ip-macsec.8.en.html.

[Dub16b] Sabrina Dubroca. *MACsec: Encryption for the wired LAN*. Hrsg. von Sabrina Dubroca. 12. Feb. 2016. URL: file:///C:/Users/Gregor/Desktop/Uni/Bachelorarbeit/Quellen/MACsec-Encryption-for-the-wired-LAN.pdf.

[Gue10] Shay Gueron. „Intel® advanced encryption standard (AES) new instructions set“. In: *Intel Corporation* (2010). URL: https://www.intel.com.bo/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf.

[Jea+16] Jérémy Jean u. a. „Deoxys v1. 41“. In: *Submitted to CAESAR* (2016). URL: http://www1.spms.ntu.edu.sg/~syllab/m/images/3/3b/Deoxysv141.pdf.

[KR14] Ted Krovetz und Phillip Rogaway. *The OCB Authenticated-Encryption Algorithm*. RFC 7253. Mai 2014. DOI: 10.17487/RFC7253. URL: https://rfc-editor.org/rfc/rfc7253.txt.

[KR16] Ted Krovetz und Phillip Rogaway. *OCB (v1. 1). Submission to CAESAR (2016)*. 2016. URL: https://competitions.cr.yp.to/round3/ocbv11.pdf.

[Lan13] Adam Langley. *[TLS] Salsa20 and Poly1305 in TLS*. 29. Juli 2013. URL: https://www.ietf.org/mail-archive/web/tls/current/msg09707.html.

[McG08] Dr. David A. McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. Jan. 2008. DOI: 10.17487/RFC5116. URL: https://rfc-editor.org/rfc/rfc5116.txt.

[Mos17] Ondrej Mosnáček. „Optimizing authenticated encryption algorithms“. In: (2017). URL: https://is.muni.cz/th/qvh3t/masters-thesis.pdf?so=nx.

[NL15] Yoav Nir und Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. Mai 2015. DOI: 10.17487/RFC7539. URL: https://rfc-editor.org/rfc/rfc7539.txt.

[Pfi00] Andreas Pfitzmann. „Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme“. In: *Lecture script* (2000). URL: https://dud.inf.tu-dresden.de/%7esk13/TUD-Web-CMS/LV/SaC-II/Skript.pdf.

[Pos+18] Jon Postel u. a. *Internet Control Message Protocol (ICMP) Parameters*. 15. Juni 2018. URL: https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml.

[Pro14] Gordon Procter. „A Security Analysis of the Composition of ChaCha20 and Poly1305.“ In: *IACR Cryptology ePrint Archive* 2014 (2014), S. 613. URL: https://eprint.iacr.org/2014/613.pdf.

[S M18] M. Vasut S. Mueller. *The Linux Kernel documentation. Kernel Crypto API Interface Specification*. 30. Sep. 2018. URL: https://www.kernel.org/doc/html/latest/crypto/intro.html.

[Tem+18] M. Tempelmeier u. a. „The CAESAR-API in the real world — Towards a fair evaluation of hardware CAESAR candidates“. In: *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Apr. 2018, S. 73–80. DOI: 10.1109/HST.2018.8383893. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8383893.

[TW12] Andrew S. Tanenbaum und David Wetherall. *Computernetzwerke /*. 5., aktualisierte Aufl. München [u.a.] : Pearson, 2012. ISBN: 9783868941371. URL: http://slubdd.de/katalog?TN_libero_mab23259530.

[WH] Hongjun Wu und Tao Huang. *The authenticated cipher MORUS (v2)(2016)*. URL: https://competitions.cr.yp.to/round3/morusv2.pdf.

[WP] Hongjun Wu und Bart Preneel. *AEGIS: A Fast Authenticated Encryption Algorithm (v1. 1). Submission to CAESAR (2016)*. URL: https://competitions.cr.yp.to/round3/aegisv11.pdf.

[WP14] Hongjun Wu und Bart Preneel. „AEGIS: A Fast Authenticated Encryption Algorithm“. In: *Selected Areas in Cryptography – SAC 2013*. Hrsg. von Tanja Lange, Kristin Lauter und Petr Lisoněk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 185–201. ISBN: 978-3-662-43414-7. URL: https://competitions.cr.yp.to/round3/aegisv11.pdf.

[Wu16] Hongjun Wu. „ACORN: a lightweight authenticated cipher (v3)“. In: *Candidate for the CAESAR Competition. See also https://competitions. cr. yp. to/round3/acornv3. pdf* (2016). URL: `https://competitions.cr.yp.to/round3/acornv3.pdf`.