HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Studienarbeit

Titel der Studienarbeit

„VoIP Security für Siemens Brandmeldeanlagen“

Herbstsemester 2011

Verfasser

Khalid Abdul und Patrik Naef

Betreuer: Prof. Dr. Andreas Steffen

Abstract

Brandmeldeanlagen der Siemens, kommunizieren über Voice over IP. Um das System vor unbefugten Zugriffen zu schützen, müssen Sicherheitsmechanismen, wie Verschlüsselung und Authentisierung angewendet werden. Dies setzt voraus, dass für die Übertragung ein Protokoll wie Real-Time Transport Protocol eingesetzt wird. Dieses Protokoll gewährt Sicherheit, benötigt im Gegenzug aber mehr Ressourcen. Bei den Brandmeldeanlagen der Siemens handelt es sich um ein Embedded System, welche über begrenzte Ressourcen verfügt. Das Embedded System wird mit einem linuxbasierten Betriebssystem namens Ångström betrieben. Dabei weist das System eine Prozessortaktrate von 400 MHz auf.

Diese Arbeit hat sich das Ziel gesetzt, die Auslastung eines solchen Systems unter die Lupe zu nehmen. Zum einen, wie sich die VoIP-Kommunikation in den unterschiedlichen Modi verhält und zum Anderen, wie stark diese die Systemressourcen auslasten. Zu diesen Modi gehört unverschlüsselt, authentisiert, verschlüsselt und die Kombination der letzten beiden Varianten. Andererseits wurde auch der Aspekt der Auslastung bei unterschiedlichen Datenraten, mit Verwendung der unterschiedlichen Modi untersucht. Dies aus dem Grund, dass bei Notfällen, Durchsagen an bis zu 100, mit Lautsprechern ausgerüstete, vernetzte Brandmeldeknoten getätigt werden müssen. Die erwarteten Werte sollen in einem Bereich liegen, welcher es erlaubt zwei parallele, bidirektionale Gespräche verschlüsselt und authentisiert zu verarbeiten.

Die Messdaten für die Verzögerung der Kommunikation, die durch das Einsetzen der Verschlüsselung und Authentisierung, bei einer Datenpaketgrösse von 20 bis 1000 Byte auftreten, bewegen sich im Bereich weniger Millisekunden. Im Detail betrachtet, wird beispielsweise für eine Übertragung von 20 Byte an unverschlüsselten Daten 2.5 ms benötigt. Für die Übertragung derselben Datenmenge mit zweimaligem ver- und entschlüsseln, sowie authentisieren, ist lediglich eine Verzögerung von 1.6 ms gemessen worden. Bei grossen Datenmengen, wie zum Beispiel bei 1000 Byte wird bei der Übertragung des unverschlüsselten Inhalts 9 ms gemessen. Wenn ein Paket mit derselben Datenmenge zusätzlich verschlüsselt und authentisiert wird, ist dabei lediglich ein Plus von 3 ms zu messen. Diese Werte zeigen auf, dass die Verzögerung durch die Verschlüsselung und Authentisierung der Datenpakete, im realen System kaum einen hörbaren Unterschied ausmachen.

Die Daten, welche für die CPU-Auslastung des Systems erhoben wurden sprechen dafür, dass zwei bidirektionale VoIP-Gespräche, das heisst zwei Mal 192 kbit/s in Senderichtung und zwei Mal in Empfangsrichtung, das System nicht wesentlich belasten. Auch die Wahl der unterschiedlichen Betriebsmodi hat keinen Einfluss auf die Auslastung.

Die erhaltenen und ausgewerteten Daten haben ergeben, dass ein Emedded System, basierend auf Ångström, durchaus in der Lage ist die verlangte Datenmenge zu verarbeiten, ohne das System total an seine Grenzen zu treiben.

Inhaltsverzeichnis

Teil I.

Grundlagen

1. Protokolle

1.1. RTP (Real-Time Transport Protocol)

Das Real-Time Transport Protocol (RTP[1]) ist ein Protokoll zur Übertragung von Multimedia Streams über das Netzwerk. RTP-Pakete werden mittels User Datagram Protocol (UDP[2]) verschickt.

1.1.1. Funktionsweise

Mit RTP werden Echtzeitmedien (Audio, Sprache, Video) als eine zusammenhängende Folge von RTP-Paketen über einen Media-Kanal, wie eine virtuelle Verbindung, übermittelt. Die übermittelten RTP-Pakete werden nummeriert, sodass die richtige Reihenfolge am Ziel wiederhergestellt werden kann. Zudem garantiert RTP die Isochronität. Dabei werden die ausgesandten Pakete mit Zeitstempel markiert, damit Zeitabstände am Ziel wiederhergestellt werden können.

1.1.2. Aufbau eines RTP-Pakets

<table>
<tr><th colspan="8">Byte 0</th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td colspan="2">V=2</td><td>P</td><td>X</td><td colspan="4">CC</td><td>M</td><td colspan="7">PT</td><td colspan="16">sequence number</td></tr>
<tr><td colspan="32">timestamp (in sample rate units)</td></tr>
<tr><td colspan="32">synchronization source (SSRC) identifier</td></tr>
<tr><td colspan="32">contributing source (CSRC) identifiers (optional)</td></tr>
<tr><td colspan="32">Header Extension (optional)</td></tr>
</table>

Tabelle 1.1.: Aufbau eines RTP-Pakets

Version (V): 2 Bit Versionsstand des RTP-Protokolls. Die Version wird durch die Spezifikation definiert.

Padding (P): 1 Bit Das Padding-Bit ist gesetzt, wenn am Ende des Pakets zusätzliche Bits angehängt sind, welche nicht zum Payload gehören. Das letzte Oktett des Paddings beinhaltet die Anzahl der Padding Oktetts, die angehängt wurden inkl. sich selbst.

Extension (X): 1 Bit Wenn das Extension Bit gesetzt ist, muss der RTP Header um genau einen Header erweitert werden.

CSRC Count: 4 Bits Der CSRC Zähler beinhaltet die Nummer von CSRC identifiers, welche dem Header folgen.

Marker (M): 1 Bit Je nach Profil wird das Marker-Bit anders interpretiert. Es wird zum Beispiel genutzt um das Ende eines Datenstreams zu markieren.

Payload Type (PT): 7 Bits Kennzeichnet das Format des Payloadtyps[3] des Pakets.

Sequence Number: 16 Bits Laufzähler für die einzelnen Pakete, wobei die Startnummer zufällig ausgewählt wird.

Timestamp: 32 Bits Der Zeitstempel gibt den Zeitpunkt des ersten Oktetts des RTP-Datenpakets an.

SSRC: 32 Bits Das SSRC Feld kennzeichnet die Synchronisationsquelle. Dieses sollte nach dem Zufallsprinzip gewählt werden, so dass während der ganzen RTP Session keine zwei Pakete dieselbe SSRC haben.

1.2. SRTP (Secure Real-Time Transport Protocol)

Das Secure Real-Time Transport Protocol (SRTP[11]) ist eine Erweiterung des RTP Standards und zielt auf die Sicherheit des Datenpakets ab. Dabei wird der Payload des RTP-Pakets verschlüsselt. Es kann sein, dass der verschlüsselte Inhalt die gleiche Länge wie der unverschlüsselte aufweist, muss aber nicht. SRTP bietet auch zusätzliche Features um die Sicherheit der Übertragung zu erhöhen.

- Ein einzelner Master Key bietet Integrität und Vertraulichkeit für den SRTP Stream. Alle für die Session nötigen Schlüssel werden schlussendlich von dem einzigen Master Key abgeleitet.
- Zusätzlich ist es möglich die Schlüsselableitung periodisch zu erneuern, so dass die Sicherheit erhöht wird.
- Die sogenannten Salting Keys werden benutzt um das Paket vor Vorberechnungen und Time-Memory Trade-Off Attacken zu schützen.

1.2.1. Aufbau eines SRTP-Pakets

<table>
<tr><th colspan="8">Byte 0</th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td colspan="2">V=2</td><td>P</td><td>X</td><td colspan="4">CC</td><td>M</td><td colspan="7">PT</td><td colspan="16">sequence number</td></tr>
<tr><td colspan="32">timestamp (in sample rate units)</td></tr>
<tr><td colspan="32">synchronization source (SSRC) identifier</td></tr>
<tr><td colspan="32">contributing source (CSRC) identifiers (optional)</td></tr>
<tr><td colspan="32">RTP Extension (optional)</td></tr>
<tr><td colspan="32">RTP Payload</td></tr>
<tr><td colspan="16"></td><td colspan="24">RTP padding</td><td colspan="8">RTP pad count</td></tr>
<tr><td colspan="32">SRTP mater key identifier (MKI, optional)</td></tr>
<tr><td colspan="32">authentication tag (recommended)</td></tr>
</table>

Tabelle 1.2.: Aufbau eines SRTP-Pakets

Wobei sich das SRTP vom Aufbau her nur beim optionalen Feld ”MKI” und dem empfohlenen Feld ”authentication tag” unterscheiden.

MKI (Master Key Identifier): konfigurierbare Länge Das MKI-Feld identifiziert den Master Key von dem aus die Session Keys abgeleitet werden, welche das Paket schlussendlich authentifizieren oder entschlüsseln. SRTP benutzt zwei Arten von Keys. Zum einen sind dies sogenannte Session Keys und zum anderen Master Keys. Der Master Key ist dabei ein zufälliger Bit-String, der durch das Key Management Protocol erstellt wird. Alle anderen Keys, darunter auch der Session Key werden durch eine Schlüsselableitungsfunktion generiert.

Authentication tag: konfigurierbare Länge Dieses Feld beinhaltet die Daten zur Nachrichtenauthentifizierung. Die Authentifizierungsdaten eines SRTP-Pakets, bestehen aus dem RTP-Header, gefolgt von den verschlüsselten Daten des SRTP-Pakets. Somit sollte, wenn Verschlüsselung und Authentifizierung angewandt wird, das Paket zuerst verschlüsselt und erst anschliessend authentifiziert werden.

1.3. UDP (User Datagram Protocol)

UDP[2] ist ein verbindungsloses Netzwerkprotokoll und arbeitet somit auf der 4. Schicht des OSI-Schichtenmodells[5]. In der Vorgehensweise, ist es vergleichbar mit dem Transmission Control Protocol (TCP[**?**]), wobei bei UDP die Ankunft der Pakete nicht garantiert wird. Dies hat jedoch einen Geschwindigkeitsvorteil, da kein Overhead generiert wird. UDP wird vor allem bei Verbindungen eingesetzt, bei denen die Geschwindigkeit eine wichtige Rolle spielt, aber die Reihenfolge oder der Verlust von Paketen nicht wesentlich ist. Bei der Voice over IP Übertragung wäre TCP zu wenig effizient, da bei Paketverlusten das Paket neu angefordert wird. Dies würde die ganze Kommunikation verlangsamen.

1.3.1. Funktionsweise

Bei UDP werden die Datenpakete von Programmen auf die jeweils konfigurierten Ports gesendet bzw. empfangen. Dabei sind die Ports von 0 bis 1023 für bekannte Dienste reserviert. Der Datenblock eines UDP-Pakets kann bis zu 65.535 Bytes betragen, da das Längen-Feld des UDP-Headers auf 16 Bits beschränkt ist.

1.3.2. Aufbau eines UDP-Pakets

Source Port (16 Bits)	Destination Port (16 Bits)
Length (16 Bits)	Checksum (16 Bits)
UDP Data (UDP SDU)	

Tabelle 1.3.: Aufbau eines UDP-Pakets

2. Kryptographie

2.1. Verschlüsselung

Verschlüsselung bezeichnet den Vorgang, einen unverschlüsselten Text (Plaintext) mit Hilfe eines Verschlüsselungsverfahrens in einen verschlüsselten Text (Ciphertext) zu wandeln, der ohne Entschlüsselung nicht interpretiert werden kann. Dabei gibt es grundsätzlich zwei verschiedene Verfahren, das symmetrische und das asymmetrische. Ein symmetrisches Kryptosystem ist ein Kryptosystem, bei welchem beide Teilnehmer den gleichen Schlüssel für die Verschlüsselung verwenden. Dieser Schlüssel muss zuerst über einen sicheren Kanal ausgetauscht werden. Er sollte periodisch gewechselt werden, ansonsten wird der Schlüssel zum Sicherheitsrisiko. Die symmetrische Verschlüsselung ist dabei in zwei Verfahren aufgeteilt, in das Stromchiffren- und Blockchiffren-basierte Verfahren. Beim Stromchiffren-basierten Verfahren wird der Plaintext Zeichen für Zeichen verschlüsselt, um den Geheimtext zu erhalten, bzw. entschlüsselt um wieder den Plaintext zu erhalten. Beim Blockchiffren-basierten Verfahren wird der Plaintext blockweise ver- bzw. entschlüsselt.

2.2. Authentisierung

Die Authentisierung[7] eines Pakets dient dazu die Integrität der übermittelten Daten zu überprüfen. Eine Methode, die Integrität[8] von Daten zu überprüfen, ist die Prüfsumme, welche aber die Daten nur auf Korrektheit überprüft. Diese Massnahme schützt dabei nicht vor absichtlicher Veränderung. Mit einem Message Authentication Code können hingegen absichtliche Veränderungen, wie auch Übertragungsfehler ermittelt werden.

2.3. Advanced Encryption Standard

Der Advanced Encryption Standard (AES[9]) ist ein sogenanntes symmetrisches Kryptographieverfahren. Bei AES wird das Blockchiffren-Verfahren eingesetzt. Dabei kann die Schlüssellänge wahlweise auf 128, 192, 256 Bit festgelegt werden. In einer Runde der AES Verschlüsselung wird der Plaintext verschiedenen Arbeitsschritten unterzogen. Die Anzahl der Runden hängt von der Schlüssellänge ab. Diese Abhängigkeit wird in Tabelle 2.1 aufgezeigt.

Schlüssellänge	Rundenzahl
128 Bit	10
192 Bit	12
256 Bit	14

Tabelle 2.1.: Abhängigkeit zwischen Rundenzahl und der Schlüssellänge

2.3.1. Advanced Encryption Standard in Counter Mode

Bei AES gibt es verschiedene Verschlüsselungsmethoden. Eine davon ist der Counter Mode. Der Counter Mode braucht entgegen anderer Modi kein Padding. Dies bedeutet, dass ein Datenpaket nicht mit einem beliebigen Inhalt auf eine bestimmte Grösse bzw. auf ein Vielfaches eines Werts aufgefüllt werden muss. AES im Counter Mode benutzt den AES Blockcipher um einen Streamcipher zu generieren. Er unterstützt die Schlüsselstromvorberechnung, da die Berechnung des Schlüsselstroms nicht von den Daten im Paket abhängig ist. Der Schlüsselstrom kann vorberechnet werden, sobald die Nonce (number used once) und der IV (initialization vector) bekannt sind. Diese Vorberechnung kann die Paketverzögerung positiv beeinflussen. Der Empfänger hingegen kann eine solche Vorberechnung nicht vollziehen, da der Initialisierungsvektor erst bekannt ist, wenn das Paket ankommt.

Funktionsweise

Um einen Plaintext mit AES im Counter Mode zu verschlüsseln wird dieser in 128 Bit grosse Blöcke unterteilt, wobei der letzte Block kleiner als 128 Bit sein kann. Jeder dieser Blöcke wird mit dem Schlüsselstrom XOR verknüpft. Aus dieser Operation entsteht der Ciphertext.

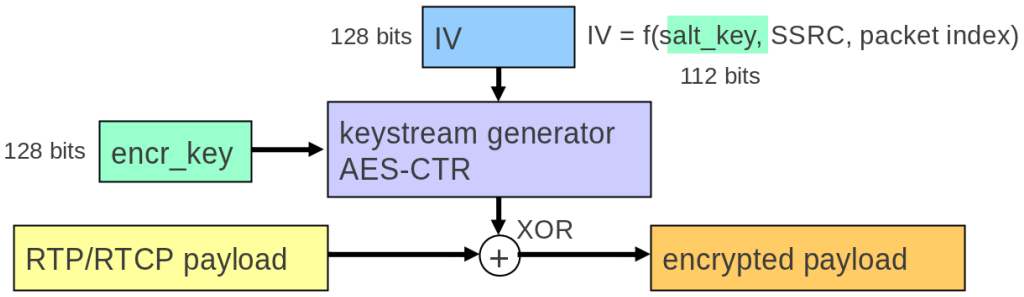

Abbildung 2.1.: Funktionsweise von AES im Counter Mode[10]

2.3.2. Key Derivation

Die Schlüsselableitungsfunktion, englisch key derivation[11], leitet geheime Sessionschlüssel von einem Master Key und einem Master Salt ab. Der Master Key ist ein zufälliger Bit-String, welcher durch das Key Management Protokoll erstellt wird. Der Master Salt, ebenfalls zufällig, kann im Gegensatz zum Master Key öffentlich zugängig sein. Die Schlüsselableitungsfunktion kann so konfiguriert werden, dass die Schlüssel zu einem gewissen Zeitpunkt erneuert werden. Dies ist von Vorteil, um sich gegen Kryptoanalyse zu schützen, da dabei nur eine gewisse Anzahl von Plaintexten mit demselben Schlüssel verschlüsselt werden.

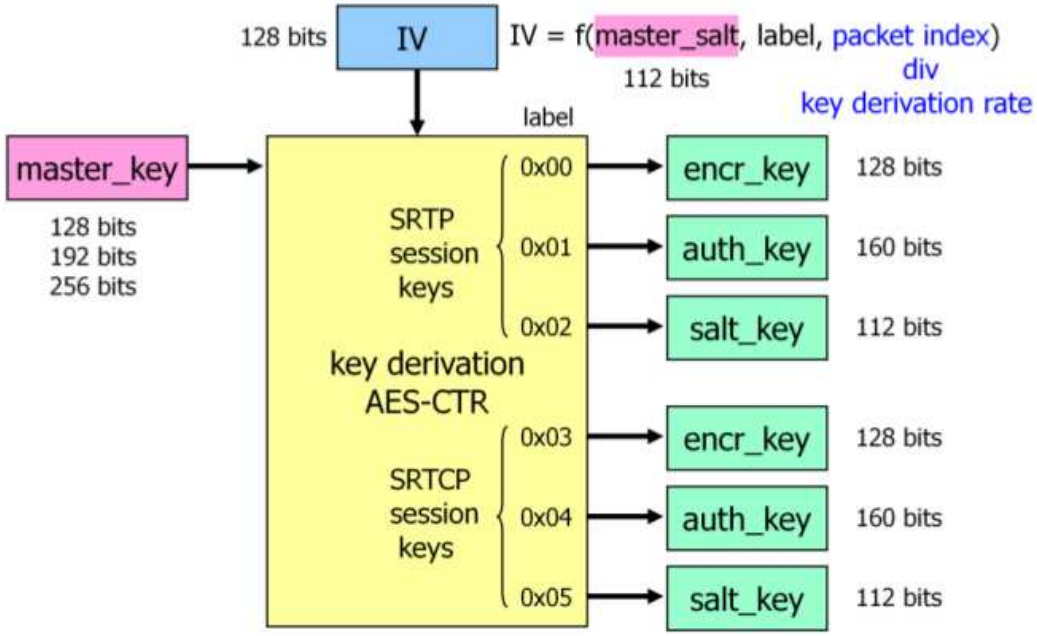

Abbildung 2.2.: Funktionsweise der Key Derivation[10]

2.4. Message Authentication Code

Ein Message Authentication Code[12] dient dazu eine Nachricht zu authentisieren. Der Benutzer erlangt durch die Information Gewissheit über den Ursprung der Nachricht, sowie dessen korrekte Übertragung. Dabei vereinbaren Sender und Empfänger einer Nachricht einen geheimen Schlüssel, wie bei der symmetrischen Verschlüsselung. Der Sender berechnet für den Schlüssel und die Nachricht, welche er übertragen will, den MAC. Dieser wird schlussendlich zusammen mit der Nachricht an den Empfänger überliefert. Der Empfänger wiederum berechnet ebenfalls den MAC und vergleicht diesen mit dem erhaltenen. So kann der Empfänger sicherstellen, dass die Daten unterwegs nicht manipuliert wurden, sofern der Schlüssel keinem Anderen bekannt ist. Der HMAC (Hash-based Message Authentication Code)[13] wie der SHA-1 oder die Erweiterung SHA-256 ist eine spezifische Implementation um Message Authentication Codes zu berechnen und basieren auf Hashfunktionen.

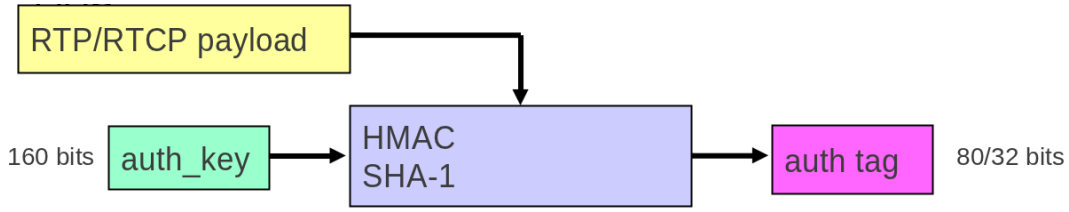

Abbildung 2.3.: Funktionsweise des SHA1-HMAC[10]

2.5. Libraries

2.5.1. GNU ccRTP

Die GNU ccRTP Library[14] ist eine Open-Source Implementierung von RTP der Internet Engineering Task Force (IETF). Ab der Version 1.5.0 ist ebenfalls auch Secure RTP eingebettet.

Die Library ist in C++ realisiert und auf Standard-Konformität, wie auch auf Flexibilität und Erweiterbarkeit ausgelegt. Die Library ist sehr performant, daher eignet sie sich sowohl für Server-Applikationen mit hohem Durchsatz, als auch für Client-Applikationen.

3. Hardware

3.1. Ångström

Ångström ist eine auf Linux basierende Distribution für Embedded Systeme. Diese Distribution ist das Resultat der Zusammenarbeit von unterschiedlichen Entwicklern von den Projekten OpenZaurus, OpenEmbedded und OpenSIMpad. Bei der Benutzeroberfläche handelt es sich dabei um das von OpenEmbedded beigesteuerte Build-Tool BitBake. Ziel der Zusammenarbeit ist das Bündeln ihrer Arbeit, wie auch eine stabile und benutzerfreundliche Distribution für Embedded Systeme. Die Brandmeldegeräte der Siemens laufen mit diesem System.

Wie viele moderne Distributionen, basiert Ångström vollständig auf Paketen. Dabei bietet Ångström gewisse "Images" an, die nichts anderes als eine Ansammlung von Core Paketen sind, welche zusammengeführt ein Archiv oder ein Filesystem-Image bilden. Diese Images lassen sich via BitBake installieren und bilden die grundlegenden Funktionalitäten des Ångström Betriebsystems. Dadurch das Ångström lediglich aus Core Paketen besteht, lässt es sich vom User durch das Hinzufügen von einzelnen Zusatzpaketen[15] individualisieren.

3.2. OpenEmbedded

OpenEmbedded steht für eine Ansammlung von Meta-Informationen und Konfigurationsdateien die von BitBake verwendet werden. Mit Hilfe der Meta-Informationen und Konfigurationsdateien werden Linux-Images für bestimmte Geräte oder Softwarepakete erzeugt.

Bei den vorhin erwähnten BitBake handelt es sich um ein Build-Tool. Mit einem BitBake wird beschrieben wie eine Applikation kompiliert werden soll und wie ihre Abhängigkeiten sind. Dabei werden die Elemente angegeben, welche in maschinenlesbare Schrift übersetzt werden sollen, wie die nötigen Bibliotheken. Zusätzlich kann noch die Abhängigkeit der Bibliothek mit angegeben werden. So kann garantiert werden, dass bei der Kompilierung alle nötigen Elemente vorhanden sind. Mit diesem System sind die Applikationen nicht vom Trägersystem abhängig, da sie jeweils für die benötigte Plattform cross-kompiliert werden. Der Cross-Kompilierer beschreibt einen Kompilierer, der auf einem bestimmten System läuft, aber Objekte für ein anderes System erzeugt. Somit stellen die BitBake, die Ansammlung von Meta-Daten dar. Eine offene Menge von weitgehend unabhängigen Paketen für unterschiedliche Systeme und Plattformen.

3.3. BitBake

BitBake ist ein Build-Tool, welches vor allem auf embedded Linux und Cross-Kompilation spezialisiert ist, wobei es nicht nur auf den Embedded Systemen eingesetzt werden kann.

3.3.1. BitBake Recipes

Ein BitBake Recipe ist eine Anleitung, welche dem BitBake mitteilt, wie einzelne Packages zu erstellen sind. Es enthält alle Abhängigkeiten, Links zum Quellcode, sowie Konfigurations-,

Kompilierungs-, Build-, Install- und Remove-Instruktionen.

Listing 3.1: HelloWorld Beispiel (hello.cpp)

```
#include "stdio.h"

void main() {
    cout << "hello world" << endl;
}
```

Listing 3.2: BitBake Recipe Beispiel (hello.bb)

```
DESCRIPTION = "hello world sample program"
PR = "r0"
SRC_URI = "file://hello.cpp"

S = "${WORKDIR}"

do_compile () {
  ${CXX} ${CXXFLAGS} ${LDFLAGS} hello.cpp -o hello
}

do_install () {
  install -d ${D}${bindir}/
  install -m 0755 ${S}/hello ${D}${bindir}/
}
FILES_${PN} = "${bindir}/hello"
```

Recipes-Methoden Für die BitBake Recipes gibt es vordefinierte Methoden, die alle ziemlich selbsterklärend sind und an ein Kochrezept erinnern, darum wohl auch der Name BitBake Recipe.

- do_fetch() : Fetch the source files from the URI given in the SRC_URI.
- do_unpack() : Unpack the source files into WORKDIR.
- do_patch() : Patch the source code, if necessary, before configuration.
- do_configure() : For doing the configuration
- do_compile() : To compile the program(s) using a compiler specified by CC variable.
- do_build() : To build the package including its build-time dependencies.
- do_install() : To install the package on the filesystem i.e copy the binaries where required.

3.3.2. BitBake Befehle

Um die beschriebenen Anweisungen in den *.bb Dateien auszuführen, ist der Task Executor notwendig. Im Allgemeinen passiert dies mit dem Aufruf ”BitBake <paketname>”, wobei man noch zusätzliche Optionen angeben kann. Nachfolgend werden lediglich die Befehle erläutert, die während der Studienarbeit verwendet wurden.

–version	gibt die aktuelle Version von BitBake aus
-h	gibt die Hilfe zu BitBake aus
-c clean	bereinigt bereits kompilierte BitBake Pakete
-s	gibt eine Liste aller zur Verfügung stehenden Pakete aus

Tabelle 3.1.: BitBake Befehle

Eine wichtige Bemerkung am Rande. Bereits kompilierte BitBake Pakete sollten immer mit -c clean bereinigt werden. Ansonsten kann es passieren, dass das BitBake Framework nicht realisiert, dass es Anpassungen am Paket gab und somit das Paket nicht neu kompiliert. Wird vor dem Kompilieren "BitBake -c clean <paketname>" ausgeführt, kann dieses Problem umgangen werden.

3.4. Hardware Spezifikation

Taurus ist ein SO-DIMM grosses Computermodul basierend auf dem Atmel® AT91SAM9G20® Prozessor. Die Taktrate des Prozessors beträgt 400 MHz, welcher einen Stromverbrauch von wenigen Watt verursacht. Der Mikrocontroller beinhaltet einen ARM926EJ-S™ ARM® Thumb® Prozessor mit 32 kByte Datencache und 32 kByte Instruktionscache. Er bietet alle gängigen Schnittstellen, welche ein heutiges System ebenfalls beinhaltet.

3.4.1. Features des ARM926EJ-S™

Manche der AT91SAM9G20® Pins sind auf mehrere Schnittstellen gemapped. Daher können nicht alle Funktionen simultan genutzt werden.

- 1 USB 2.0 full speed (12 MBit per second) device port
- 2 USB 2.0 full speed (12 MBit per second) host port
- Ethernet MAC 10/100 Base T
- Image sensor interface (ISI)
- Periodic interval timer (PIT)
- Watchdog timer
- Real-time timer (RTT)
- Reset controller
- Advanced interrupt controller (AIC)
- Debug unit (DBGU)
- 4-channel analog to digital converter (ADC)
- 3 32-Bit parallel input / output controllers (PIOA, PIOB, PIOC)
- Multi media card interface (MCI)
- Synchronous serial controller (SSC)

- 4 universal synchronous / asynchronous receiver transmitter (USART)
- 2 2-wire universal asynchronous receiver transmitter (UART)
- 2 master / slave serial peripheral interfaces (SPI)
- 2 3-channel 16-Bit timer / counter (TC)
- 2-wire interface (TWI)

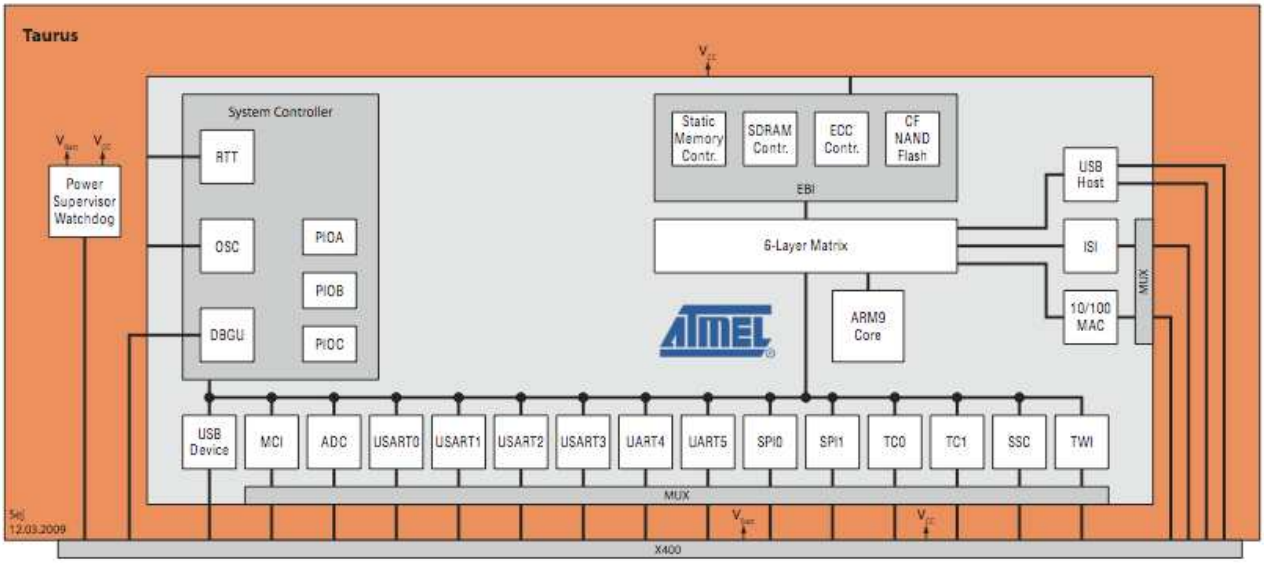

Abbildung 3.1.: Blockdiagramm des Taurus Boards

3.4.2. Speicher

Der SDR SDRAM (Single data rate SDRAM) Speicher des Boards wird durch 32 bit organisiert. Unterstützt wird 16 MByte bis zu 128 MByte, wobei die Busgeschwindigkeit 133 MHz beträgt. Ausserdem enthält das Board einen NAND Flash, welcher durch 8 bit organisiert wird. Dabei unterstützt das Board 256 MByte bis zu 8 GByte. Die Busgeschwindigkeit beträgt ebenfalls 133 MHz. Das SRAM wird durch 16 bit organisiert. Unterstützt wird dabei bis zu 1 MByte. Falls eine permanente Speicherung nötig ist, muss zusätzlich eine externe Stromversorgung angeschlossen werden.

3.4.3. Ethernet

Das Board ist mit zwei Ethernet Schnittstellen bestückt, welche eine Datenrate von 10/100 Mbit/s erlauben, die über die Linux Distribution Ångström verwaltet werden.

3.4.4. USB

Das Board verfügt über eine USB Schnittstelle. Mit dieser Schnittstelle wird die Software, die sogenanten Recipes, auf das Board geflasht.

3.4.5. RS-232

Das Board verfügt über eine RS-232 Schnittstelle. Anhand dieser Schnittstelle kann beispielsweise via Putty auf das Board zugegriffen werden.

Teil II.

Softwaredokumentation

4. Design

4.1. Architektur

Da die Software lediglich zur Evaluation der Verzögerung und der Auslastung des Prozessors benötigt wurde, lag das Hauptaugenmerk auf dem Erzielen von plausiblen Resultaten und nicht auf der Schönheit des Quellcodes. Deshalb wird die Struktur der Applikation nur soweit dokumentiert, wie sie zum Verständnis der Arbeit beiträgt. Für die Realisierung der Studienarbeit wurden zwei unabhängige Applikationen erstellt, VoIPSecTime und VoIPSecCPU. Der Grund weshalb wir zwei voneinander unabhängige aber generell sehr ähnliche Applikationen erstellt haben lag darin, dass wir einerseits keine Applikation erstellen wollten, welche unzählige Parameter entgegen nimmt und andererseits, weil wir zwei verschiedene Konfigurationen benötigten, da VoIPSecCPU auf die Geschwindigkeit und VoIPSecTime auf die Zeit abzielt.

4.1.1. Klassendiagramm

VoIPSecTime

VoIPSecTime ist die Applikation, welche die Verzögerung der Kommunikation durch die verschiedenen Modi evaluiert. Die Applikation besteht aus den Klassen **Packet, Receiver, Sender, TimeMeasure und Logger**. Das main.cpp wird in dem Klassendiagramm nicht dargestellt, da es sich dabei um keine logische Klasse handelt. Die Logger-, wie auch die TimeMeasure-Klasse wurden als Singletons implementiert, da sie vom Sender und vom Receiver benötigt werden, um Log-Einträge beim senden und beim empfangen zu generieren, oder um die Zeit zu starten und zu stoppen.

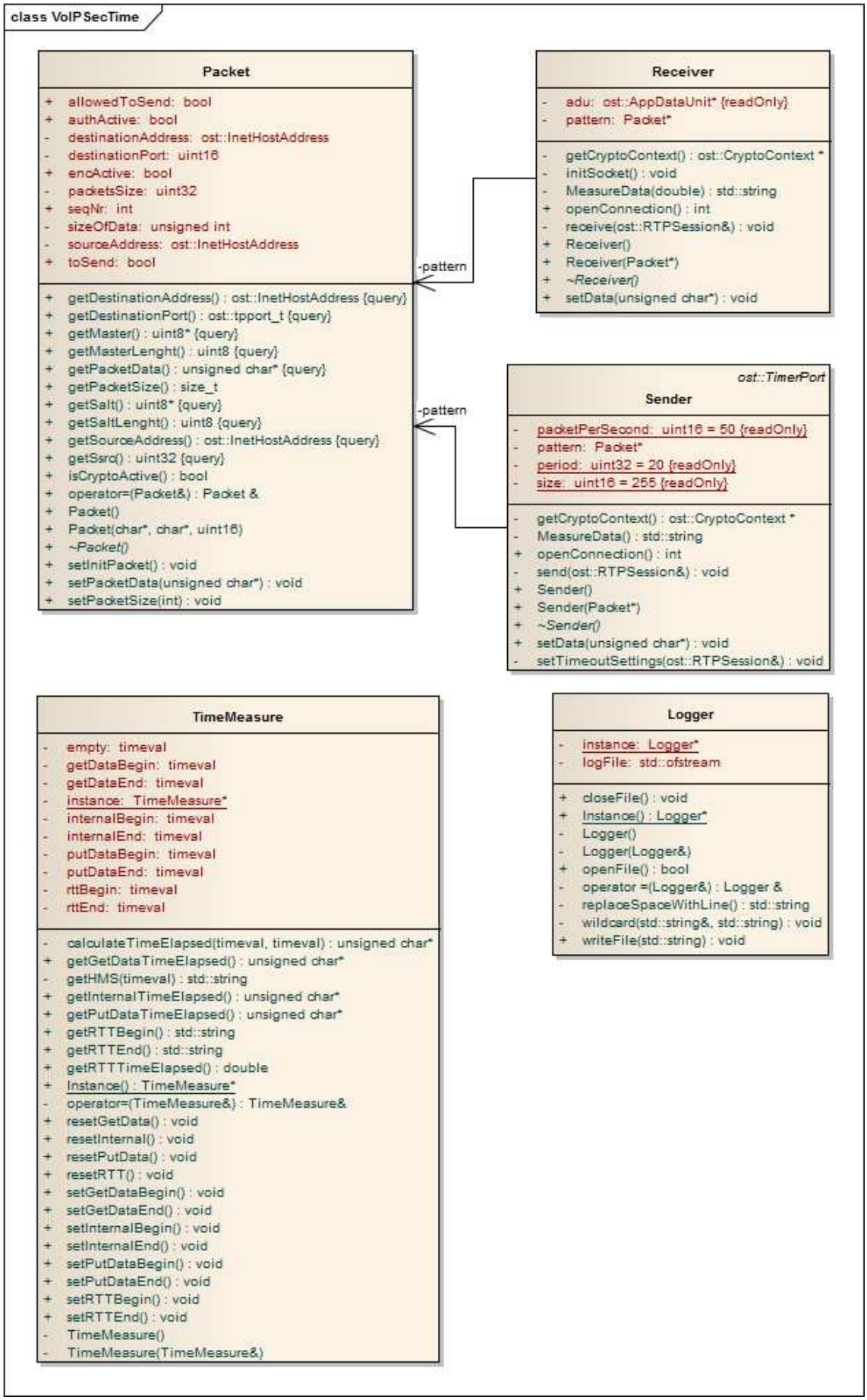

Abbildung 4.1.: Klassendiagramm der VoIPSecTime Applikation

Packet Die Klasse Packet ist das zentrale Element in der Applikation, welche zugleich auch die Datenhaltung übernimmt. Aufgabe dieser Klasse ist es, die Daten jederzeit und überall zur Verfügung zu stellen. In der Klasse werden die Daten zur Kommunikation, zur Betriebsart und die eigentlich zu übermittelnden Daten gehalten.

Sender Die Sender Klasse ist eine Komponente der Kommunikation. Hier wird die Kommunikation eröffnet, bzw. die Daten versendet. In dieser Klasse wird je nach Modi das Paket modifiziert, also verschlüsselt und/oder authentisiert.

Receiver Diese Klasse stellt das Gegenstück der Sender Klasse dar. Hier wird das Paket entgegengenommen. Die empfangenen Daten werden je nach Modi unterschiedlich aufbereitet, also entschlüsselt und/oder die Authentisierung geprüft.

Logger Der Logger erstellt die Protokolle für die Aufzeichnungen. Die Protokolle werden jeweils anhand des aktuellen Datums und Uhrzeit benannt. Somit wird vermieden, dass zwei unterschiedliche Protokolle denselben Namen erhalten.

TimeMeasure Die Zeitmessungen werden mit der Klasse TimeMeasure gemacht. Die Klasse enthält die Messungen für den RTT, die interne Zeit der Verarbeitung und für die Methode putData() des Receivers.

VoIPSecCPU

Die eigenständige Applikation VoIPSecCPU, dient dazu die Auslastung des, auf dem Board verbauten, ARM9 Prozessors während dem verschlüsseln und authentisieren zu dokumentieren. Grob gesagt ist der Aufbau der Applikation samt ihren Klassen derselbe wie bei VoIPSecTime. Lediglich wurde sie durch das entfernen des Receivers und des Loggers geschmälert. Gleichzeitig wurde die Klasse TimeMeasure durch die Klasse CPUUsage ersetzt. Die Ergebnisse werden nicht mehr wie bei VoIPSecTime in ein File geschrieben, sondern direkt im Terminal ausgegeben.

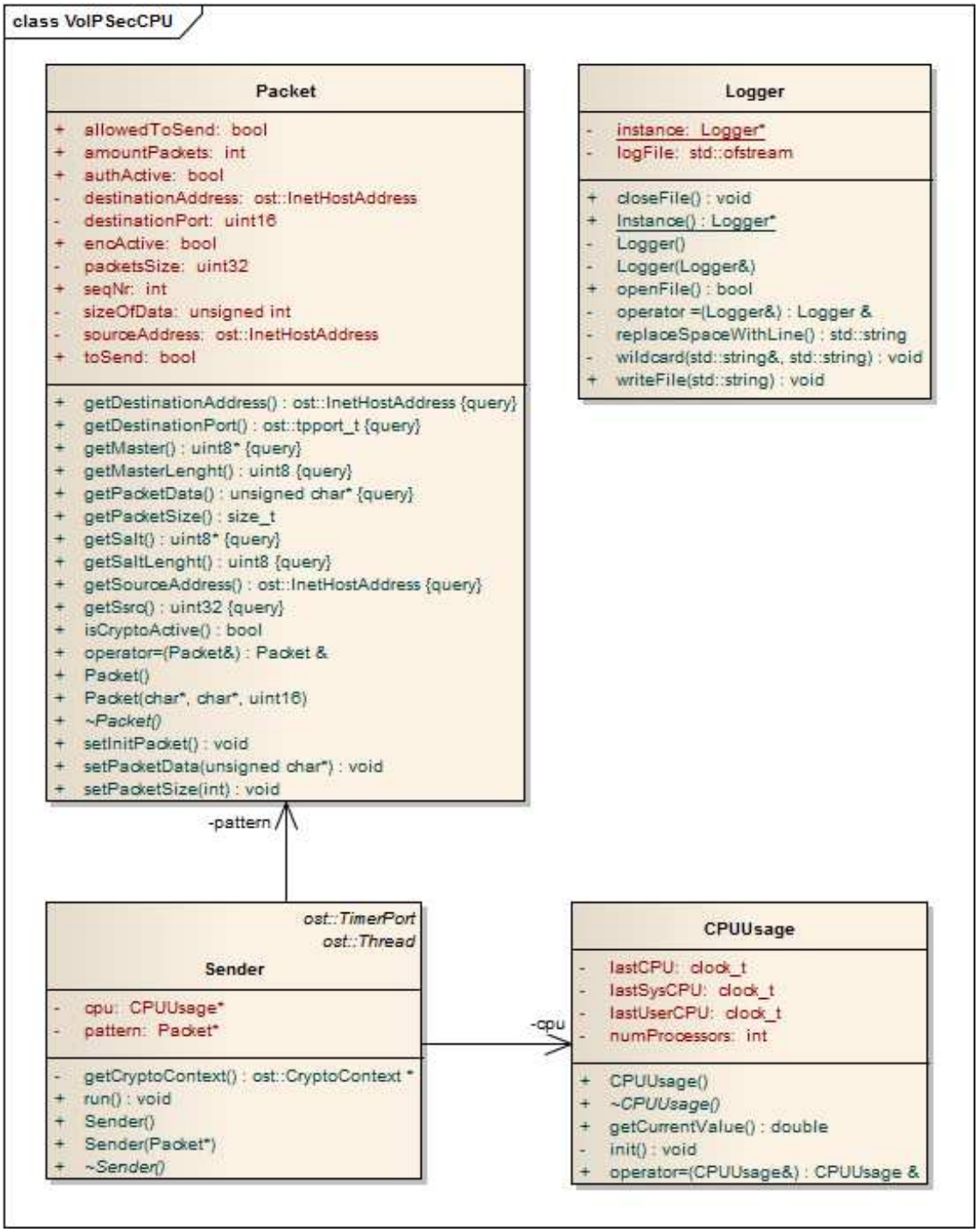

Abbildung 4.2.: Klassendiagramm der VoIPSecCPU Applikation

CPUUsage Die Auslastung des Prozessors wird mit Hilfe der Klasse CPUUsage gemessen. Bei jedem Messpunkt werden die Systemzeit, User-Mode-Zeit und Kern-Mode-Zeit genommen. Mit Hilfe dieser Angaben kann die Auslastung in Prozent berechnet werden.

$$Auslastung = \frac{(kern_{neu} - kern_{alt}) + (user_{neu} - user_{alt})}{system_{neu} - system_{alt}} * 100$$

$system$ = Systemzeit
$kern$ = Kern-Mode-Zeit
$user$ = User-Mode-Zeit

4.1.2. Sequenzdiagramm

Um die Aufrufe und das Zusammenspiel der einzelnen Klassen untereinander ein wenig klarer zu machen wurde ein Sequenzdiagramm erstellt. Dieses zeigt zum Einen auf, welche Klassen zu welchem Zeitpunkt und von wem initialisiert werden und zum Anderen die Methodenaufrufe die klassenübergreifend während der Laufzeit der Applikation entstehen.

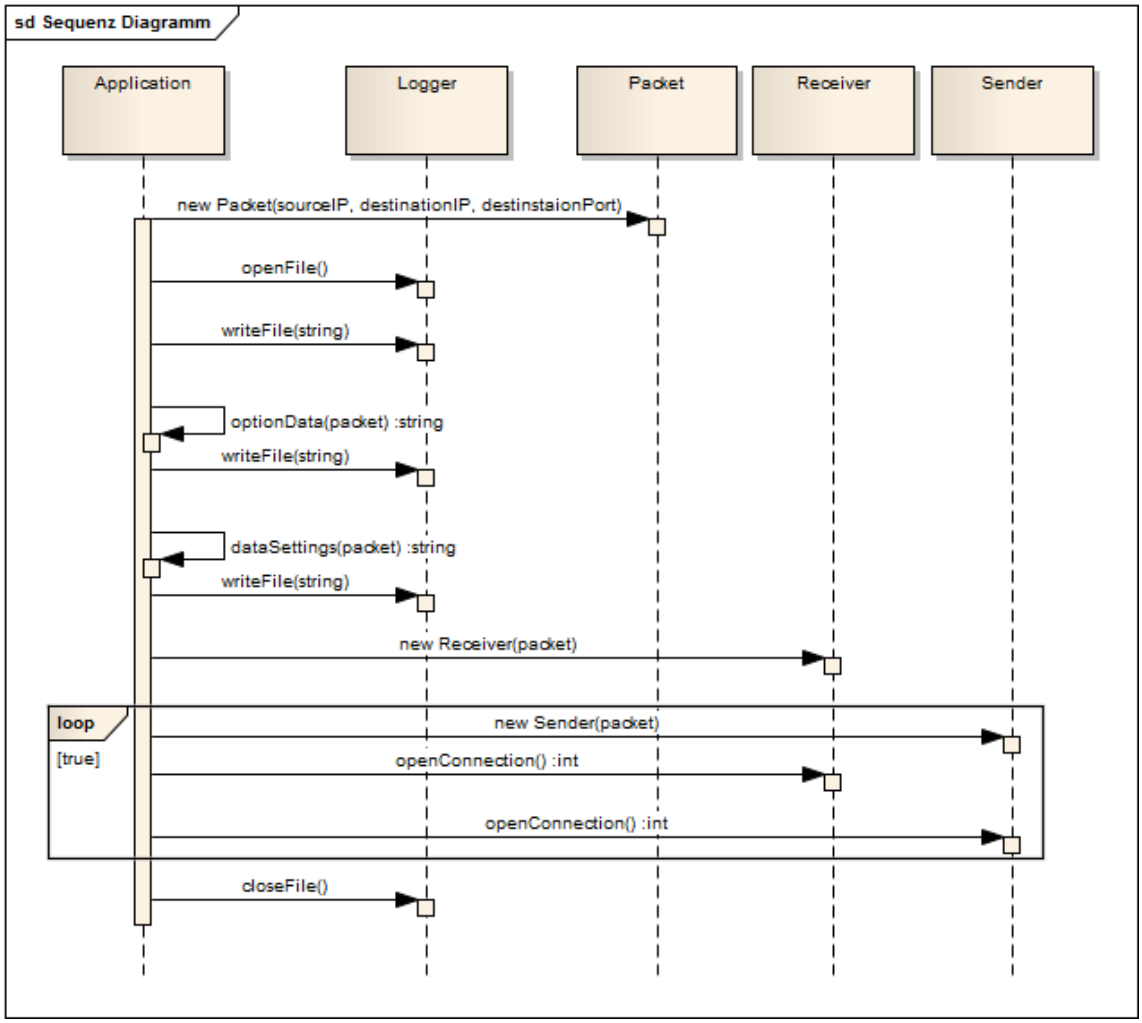

Abbildung 4.3.: Sequenzdiagramm der VoIPSecTime Applikation

5. Software Engineering

5.1. Installation

5.1.1. Vorbereitung

Als erstes muss man im Verzeichnis /home/one/ccp den Befehl ". envvars taurus" absetzen. Damit werden auf dem Debian die Umgebungsvariablen gesetzt und die nachfolgenden Befehle sollten dem Debian bekannt sein.

5.1.2. Libraries installieren (direkt)

Um Libraries auf den Boards zu installieren gibt es zweierlei Wege. Die Library direkt zu installieren oder manuell. Wenn die Library schon in der Datenbank von OpenEmbedded eingetragen sind, kann man mit dem Befehl

Listing 5.1: BitBake einer Library

```
1 bitbake <libraryname>
```

die Library installieren. Um eine Auflistung der vorhandenen Pakete zu sehen kann der Befehl

Listing 5.2: Auflistung aller Packages

```
1 bitbake -s
```

angewendet werden. Für unsere Applikation wurden zwei Libraries installiert. CommonCPP und ccRTP. Beide waren schon in der Datenbank vorhanden und konnten via oben genanntem Befehl installiert werden.

Listing 5.3: BitBake Aufruf unserer Library

```
1 bitbake commoncpp2
2 bitbake ccrtp
```

5.1.3. Libraries installieren (manuell)

Falls die Libraries nicht in der Datenbank von OpenEmbedded eingetragen sind, müssen Libraries manuell installiert werden. In unserem Fall war die Library von ccRTP veraltet. Bei OpenEmbedded war die Version 1.7.0 eingetragen. Die neuste Version ist Version 2.0.1. Von Herrn Schmidt bekamen wir den Tipp eine neue Version auf Foren zu suchen, die sich mit OpenEmbedded beschäftigen. So wurden wir auf einen Eintrag aufmerksam, bei dem ein Entwickler die Version 1.8.0 geposted hat. Der Download bestand aus einem *.bb-File und einem *.ac-File. Diese Files wurden von uns heruntergeladen und in das untenstehende Verzeichnis kopiert.

```
Recipes Pfad: /home/one/ccp/ccp/recipes/
BitBake Pfad: /home/one/ccp/ccp/recipes/PACKAGENAME/
Files Pfad: /home/one/ccp/ccp/recipes/PACKAGENAME/files/
```

Dabei wird das *.ac-File in den Files Ordner kopiert und das *.bb-File in das übergeordnete Verzeichnis. Unter Umständen, wie auch bei unserer Library, können *.bbclass-Files fehlen. Diese sind zum Beispiel nötig, wenn autotools verwendet wird. Autotools ist eine Sammlung von Tools als Hilfestellung für die Programmierung, welche von GNU entwickelt wurde. Die Tools sind vor allem für das Portieren von Source-Paketen auf Unix-Systeme gedacht. Dieses autotools_stage.bbclass-File haben wir einzeln heruntergeladen und in den Ordner /home/one/ccp/ccp/classes kopiert. Um das Package zu installieren, kann man schlussendlich den Filenamen des *.bb-Files beim BitBake Befehl angeben.

5.2. Eigenes Programm entwickeln (mit Link zu einer Library)

Um ein eigenes Programm auf die Hardware zu portieren, bedarf es wie bei den Libraries, selbstverständlich auch eines *.bb-Files und den Source-Files, seien es C-Files oder C++-Files. Die Files werden dabei in den Ordner für die Files kopiert und das BitBake-File wird im übergeordneten Pfad hinterlegt. Auf dem Debian wäre es nun möglich mit dem GCC oder G++ das Programm zu kompilieren und dabei einen Pfad für die Library anzugeben. Im Beispiel wird auf die ccRTP Library referenziert.

Listing 5.4: G++ Command zum kompilieren eines C++-Projekts

```
g++ main.cpp send.cpp receive.cpp packet.cpp packet.h
-o ExampleApp -L/usr/local/lib/ -lccrtp
```

-o (Object File): Steht für den Namen des Object Files, welches nach der Kompilierung entsteht.

-L (Library Path): Steht für den Pfad in dem sich die Library befindet.

-l (Library Name): Steht für den Namen der Library. Dabei muss beachtet werden, dass die Library in unserem Fall libccrtp.so heisst. Dennoch muss man beim kompilieren nur den Namen "ccrtp" ohne "lib" und ohne Endung angeben.

Da wir nun aber eine Cross-Kompilierung vornehmen, und keine statischen oder Debian spezifischen Pfade angeben können, müssen wir den Library Pfad dynamisch angeben können. Da das BitBake vordefinierte Variablen beinhaltet, ist dies mit einem simplen Aufruf möglich.

Listing 5.5: BitBake Command zum kompilieren eines C++-Projekts

```
${CXX} ${CXXFLAGS} ${LDFLAGS} main.cpp send.cpp receive.cpp packet.
    cpp packet.h -o ExampleApp -L${LIBRARYDIR} -lccrtp1
```

Um dieses BitBake nun zu kompilieren, geht man gleich vor, wie bei den Libraries. Im Terminal kann nach ausführen des ". envvars taurus"-Befehls, im Ordner /home/one/ccp, durch eingeben des Befehls "bitbake <bbfilename> (ohne version und .bb)" das Programm kompiliert werden. Falls man die Applikation in das base-image einflechten will, kann man im Ordner /home/one/ccp/taurus/recipes/images/ das base-image.bb-File anpassen, indem man in die vorgegebene Liste der Applikationen die eigene hineinschreibt.

5.2.1. Auf das Taurus Eval installieren

Als allererstes muss das Board per USB und RS232 Kabel mit dem PC verbunden werden. Dabei werden die Treiber automatisch installiert. Das angeschlossene Gerät muss dann für die VM verfügbar gemacht werden. In der VM, auf der ein Debian läuft, kann nun per Terminal in das Verzeichnis /dev gewechselt werden. Dort kann man mit Hilfe des Befehls "ls -al tty*" die verbundenen Geräte anzeigen lassen. In der Liste sollte neben ttyUSB0 auch ttyACM0 erscheinen, welches die Serielle Schnittstelle des Boards repräsentiert. Um die Arbeit mit dem Gerät zu erleichtern kann man mittels dem Befehl "sudo ln -s ttyACM0 ttyUSB3" die serielle Schnittstelle symbolisch auf den USB3 verlinken.

Damit man auf das Ångström auf dem Board zugreifen kann, öffnet man Putty mit der Konfiguration, welche auf Abbildung 5.1 dargestellt ist.

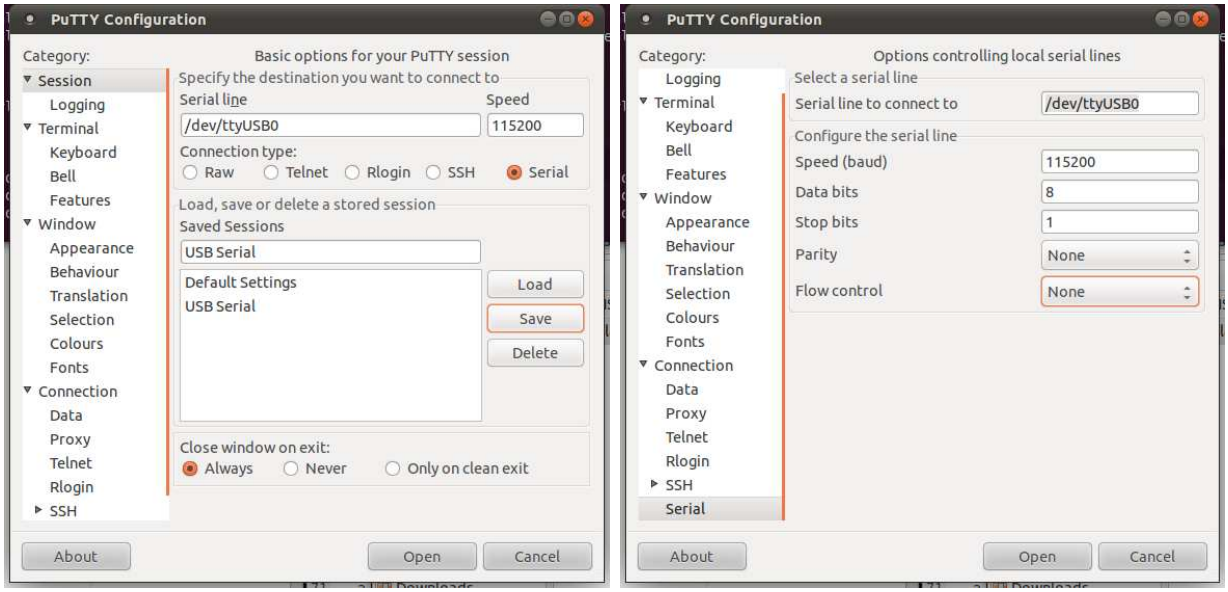

Abbildung 5.1.: Konfiguration von Putty

Nach dem verbinden, das Board neu starten bzw. einschalten und auf der Konsole erscheint die Ausgabe "RomBOOT", was so viel bedeutet wie, es ist noch kein Betriebssystem installiert. Falls schon ein Ångström auf dem Board vorhanden ist, erscheint auf der Konsole die übliche Ångström Startsequenz. Falls das Board schon ein Betriebssystem enthält, muss dieses zuerst deinstalliert werden. Um dies zu erreichen, muss man während dem booten bei der Meldung "Hit any key to stop autoboot" den Startvorgang mit einer beliebigen Taste unterbrechen (Abbildung 5.2). An dieser Stelle kann man den Befehl "nand erase 0" eingeben, welche den Speicher des Boards formatiert. Wenn die Formatierung beendet ist, wird mit dem Befehl "re" das Board zurückgesetzt und auf der Konsole erscheint die Meldung ">RomBOOT". Zurück im Terminal der virtuellen Debian Maschine kann das base-image gleich wie die Applikation mit dem Befehl "bitbake base-image" kompiliert werden. Dabei werden alle im File enthaltenen Programme mitkompiliert.

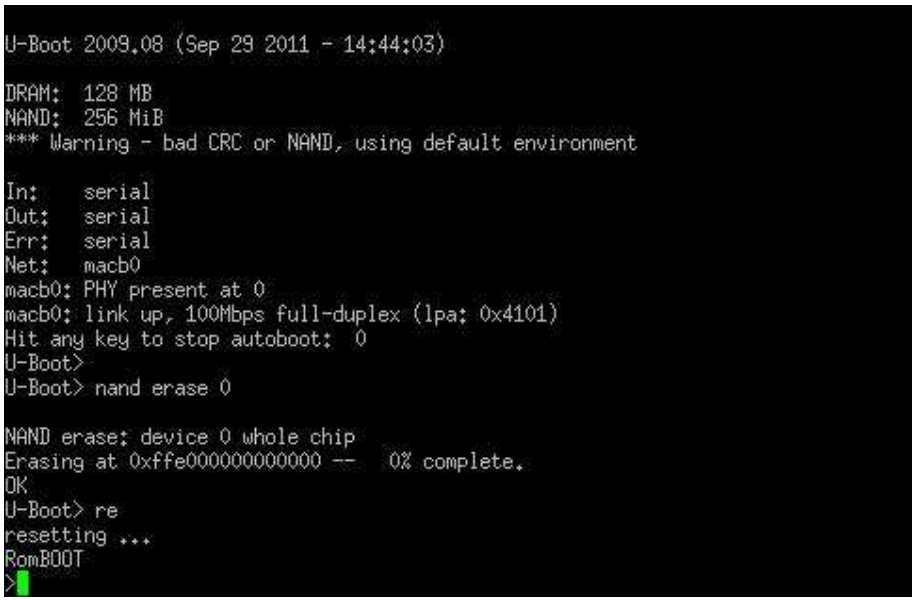

Abbildung 5.2.: Unterbrechung des Startvorgangs

Damit man jetzt das fertige Ångström Image auf das Board spielen kann, muss im Ordner /home/one/ccp/taurus-ref/deploy/eglibc/images/taurus das Shell-Script taurus-eval-nand.sh, mit ". taurus-eval-nand.sh", ausgeführt werden. Auf der Konsole des Boards sollte dabei ein Arbeitsvorgang sichtbar sein. Wenn dieser beendet ist, und sich im Terminal des Debian das Log öffnet, muss das Board von Hand mit dem Reboot-Knopf neu gestartet werden. Der Login für das Ångström ist "root" und das Passwort ist leer.

5.3. Benutzerhandbuch

Um das Ziel der Arbeit zu erfüllen wurden zwei Applikationen erstellt. Dabei testet eine Applikation mit dem Namen, VoIPSecTime, die Verzögerung und VoIPSecCPU, die Auslastung durch die Benutzung der einzelnen Modi. Dadurch dass die Applikationen lediglich zur Evaluation benötigt werden, lassen sie sich mit wenigen Parametern bedienen.

-s	Starten als Sender
-a	Schaltet Authentifizierung ein
-e	Schaltet Verschlüsselung ein
-d (wert)	Datengrösse in Byte [max. 4096]
-l	Schaltet den Logger ein

Tabelle 5.1.: Parameter der VoIPSecTime Applikation

-a	Schaltet Authentifizierung ein
-e	Schaltet Verschlüsselung ein
-d (wert)	Datengrösse in Byte [max. 4096]

Tabelle 5.2.: Parameter der VoIPSecCPU Applikation

Beispiel für eine Kommunikation zwischen einem Sender und Empfänger.

Aufruf auf Sender Seite	Aufruf auf Empfänger Seite
VoIPSecTime -lsaed 100	VoIPSecTime -aed 100

Tabelle 5.3.: Aufruf der Applikation mit eingeschaltetem Logger, Authentifizierung, Verschlüsselung und 100 Byte Daten

Teil III.

Tests und Auswertungen

6. Tests und Auswertungen

6.1. Vorgehen

6.1.1. Topologie

Der Testaufbau für die Aufzeichnungen ist ein abgeschottetes lokales Netzwerk mit statischen IP-Adressen. Der Aufbau wurde so gewählt, damit optimale Bedingungen für die Aufzeichnungen gegeben sind. So wird verhindert, dass die Kommunikation von anderen Geräten im Netzwerk beeinflusst wird. Der Testaufbau besteht aus folgenden Elementen:

Anzahl	Elemente
2	Taurus Eval Board (Atmel AT91SAM9G20 CPU)
1	Ethernet Switch Allied Telesyn AT-FS708 (10/100 Mbit)
1	DC Speisegerät Agilent E3630A

Tabelle 6.1.: Verwendete Elemente für den Testaufbau

In der Topologie Abbildung 6.1 ist ersichtlich wie die Elemente angeordnet sind.

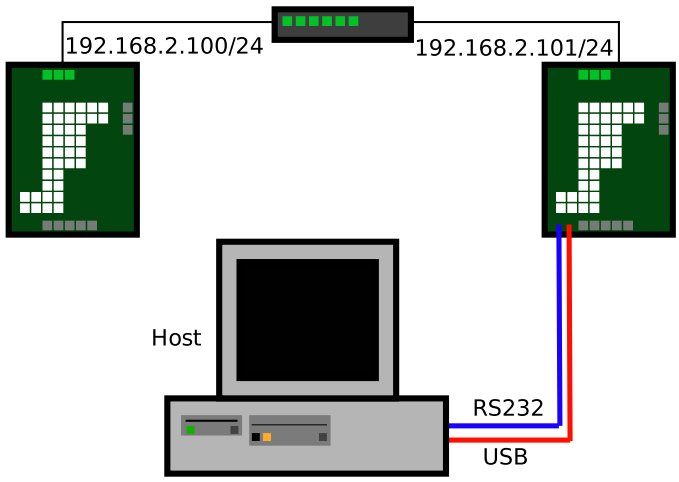

Abbildung 6.1.: Topologie des Testaufbaus

6.1.2. Zeitmessung

Ein Bestandteil der Arbeit ist die Ermittlung der **Round Trip Time (RTT)** der SRTP-Pakete. Um diesen Wert zu ermitteln, wird an unterschiedlichen Stellen im Programm die Zeit gestoppt. Die daruaus resultierende Messzeit setzt sich aus zweimaligem ver- bzw. entschlüsseln und der Zeit, die benötigt wird um die Daten vom Board A zum Board B zu senden, zusammen.

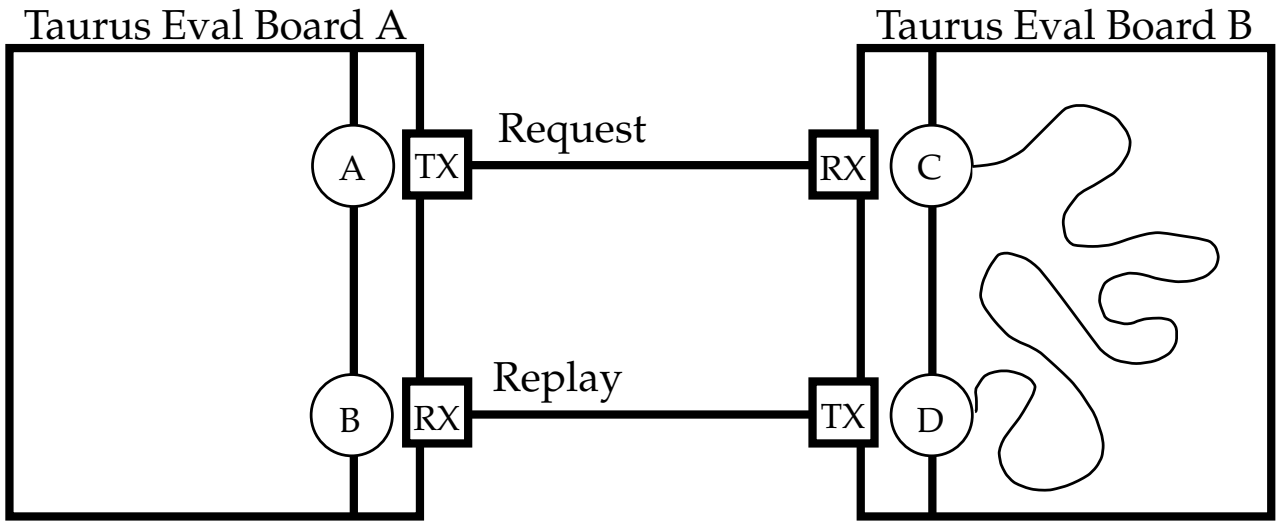

Abbildung 6.2.: Grafische Darstellung der Zeitmessung

Wie die Zeitmessung funktioniert wird anhand eines Beispiels instruiert. Die Messung wird beim Taurus Eval Board A gestartet. Somit sendet das Board A ein SRTP-Paket an das Taurus Eval Board B. Das Board B empfängt das Paket und entschlüsselt dieses. Nach der Entschlüsselung wird im Board B ebenfalls eine Zeitmessung gestartet. Diese Zeitmessung befasst sich mit der Zeit die intern verstreicht. Diese interne Zeitmessung wird vor der Verschlüsselung gestoppt und mit dem Antwort-Paket an das Board A mitgesendet. Das Board A empfängt das Paket und entschlüsselt dieses und stoppt zugleich die Zeitmessung. Daraufhin wird die RTT berechnet, dazu wird die Zeit vom Board A genommen und die interne Zeit vom Board B abgezogen.

- An Punkt A wird die Startzeit für die RTT genommen
- An Punkt B wird die Endzeit für die RTT gestoppt
- An Punkt C wird die Startzeit für die interne Zeitmessung genommen
- An Punkt D wird die Endzeit für die interne Zeitmessung gestoppt

Rechnerisch sieht dies wie folgt aus:

$$\boxed{RTT = t_b - t_a - (t_d - t_c)}$$

6.1.3. Auslastungstest

Der Auslastungstest beinhaltet die Abhängigkeit zwischen der Datenrate und der Auslastung des Boards. Für diesen Zweck haben wir eine eigenständige Applikation entwickelt, welche zwei bidirektionale Verbindungen simuliert. Anders ausgedrückt sind dies zwei Mal 192 kbit/s in Senderichtung und zwei Mal 192 kbit/s in Empfangsrichtung. Um dies zu simulieren werden zwei Threads à 384 kbit/s gestartet. Ein jeweiliger Thread versendet 48 Pakete in der Sekunde, mit einem Abstand von 20 ms. Die Anzahl der Pakete resultiert daraus, da 384 kbit/s 48 KByte/s entsprechen und die Paketgrösse 1000 Byte beträgt. Der Abstand von 20 ms ist das Resultat von 1000 ms durch die Anzahl zu versendenden Pakete.

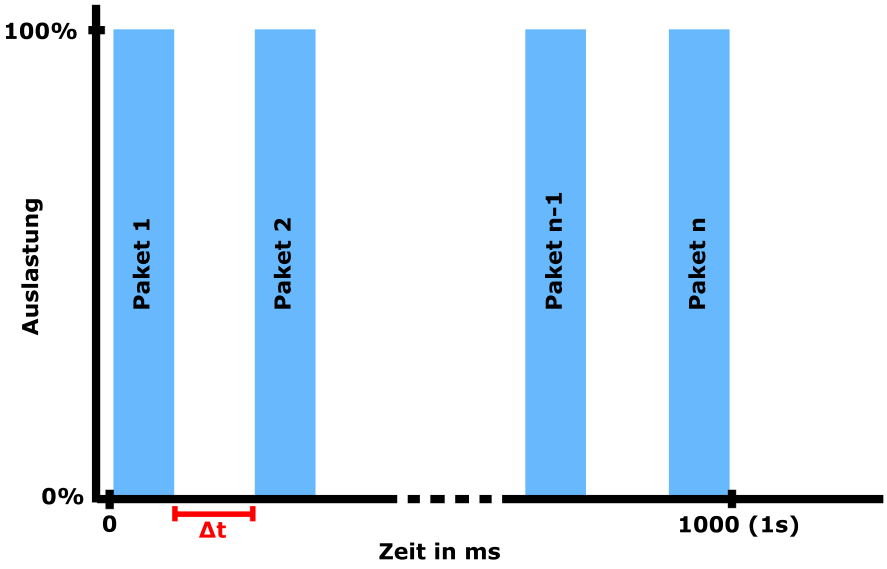

Abbildung 6.3.: Schematische Darstellung der CPU Messung

$$\Delta t = \frac{1000ms \times \text{Paketgrösse}}{\text{Bytes pro Sekunde}}$$

6.2. Ergebnisse

6.2.1. Zeitmessung

Die Daten der Aufzeichnungen werden im Comma-Separated-Values Dateiformat gespeichert. Dabei wird als Trennzeichen das Semikolon ";" verwendet. Dieses Dateiformat wurde gewählt, weil es von Excel und anderen Tabellenverarbeitungsprogrammen interpretiert werden kann. Die Aufzeichnungen wurden für die Datengrössen 20, 100, 200, 500 und 1000 Byte gemacht. Die einzelnen Messserien bestehen aus vier Messungen: unverschlüsselt, authentifiziert, verschlüsselt und die Kombination der letzten beiden. Die Ergebnisse werden auf den nächsten Seiten präsentiert und zum Schluss gegenübergestellt.

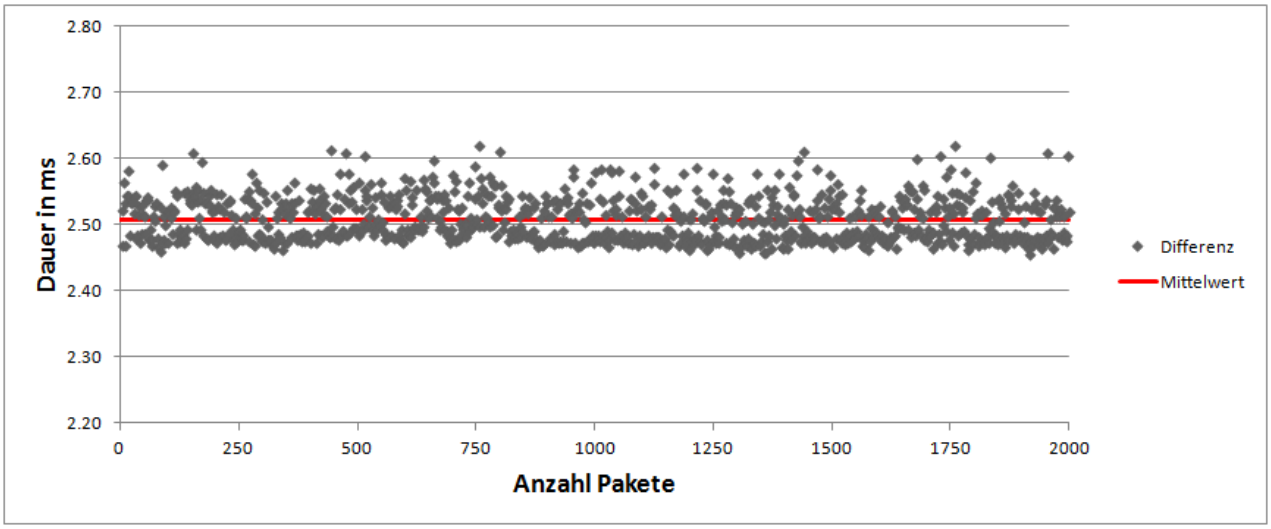

Abbildung 6.4.: Round Trip Time bei 20 Byte Plaintext

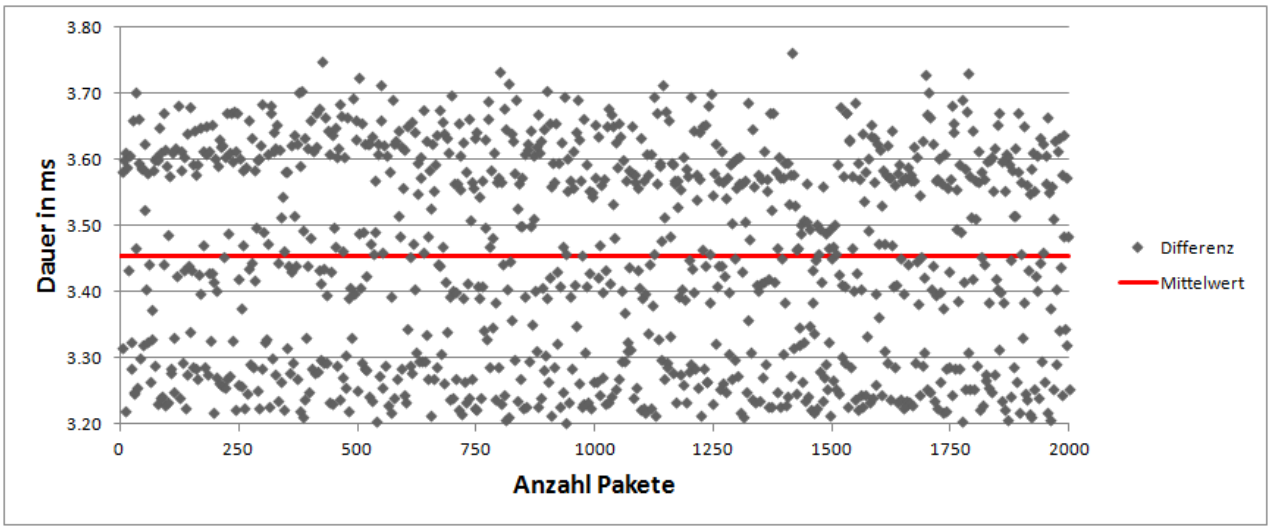

Abbildung 6.5.: Round Trip Time bei 20 Byte mit Authentication

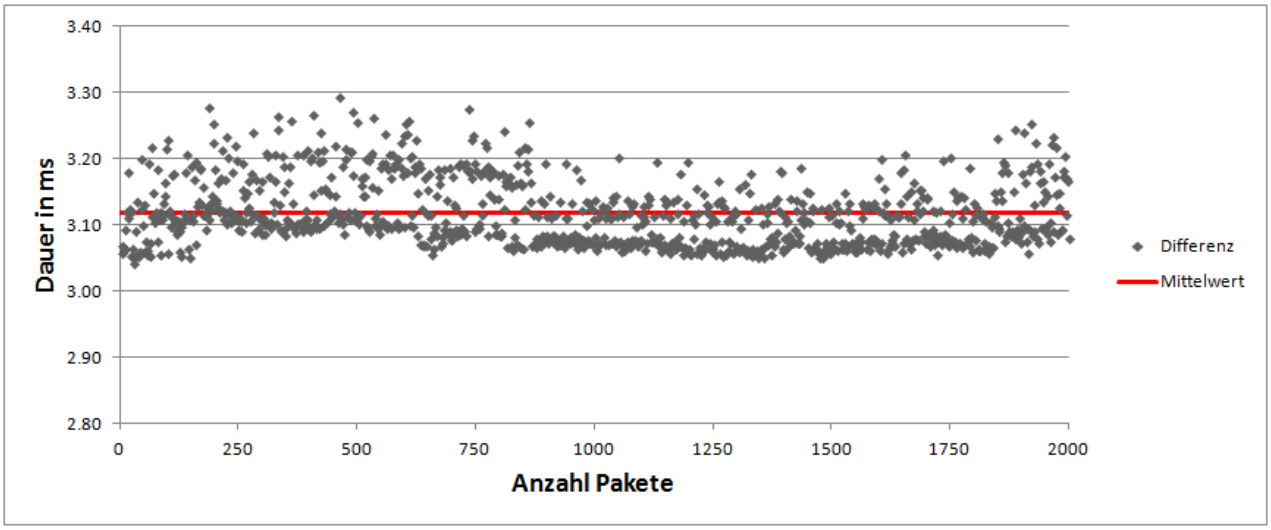

Abbildung 6.6.: Round Trip Time bei 20 Byte mit Encryption

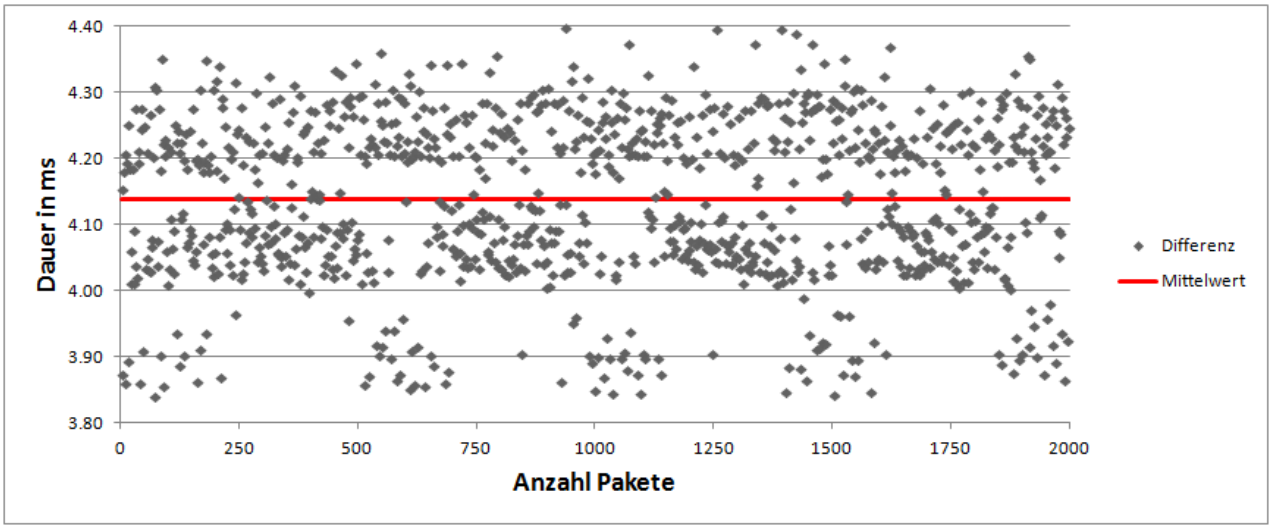

Abbildung 6.7.: Round Trip Time bei 20 Byte mit Authentication und Encryption

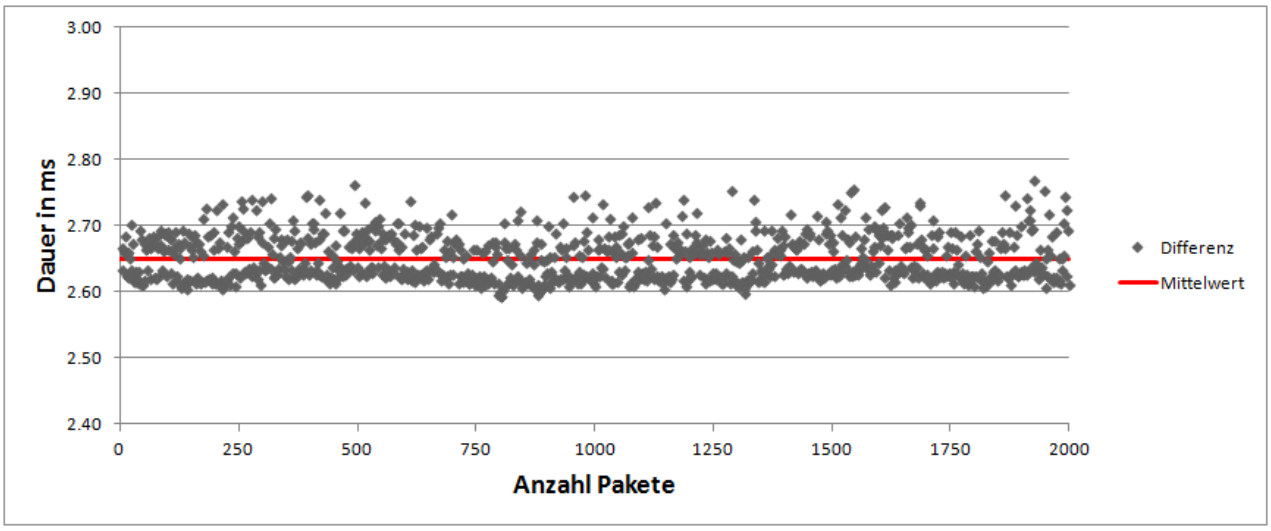

Abbildung 6.8.: Round Trip Time bei 100 Byte Plaintext

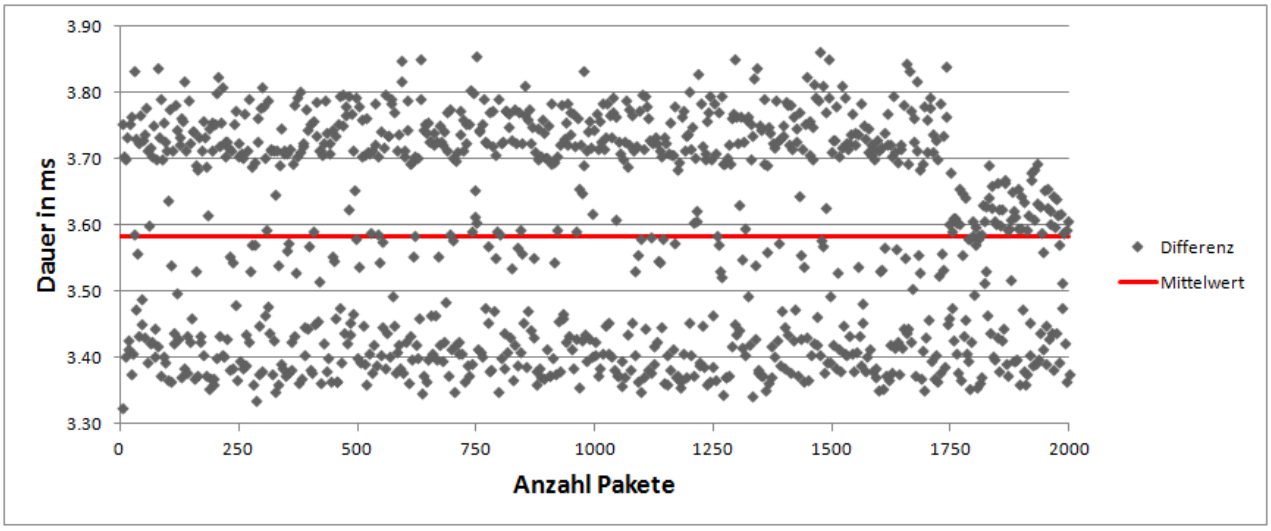

Abbildung 6.9.: Round Trip Time bei 100 Byte mit Authentication

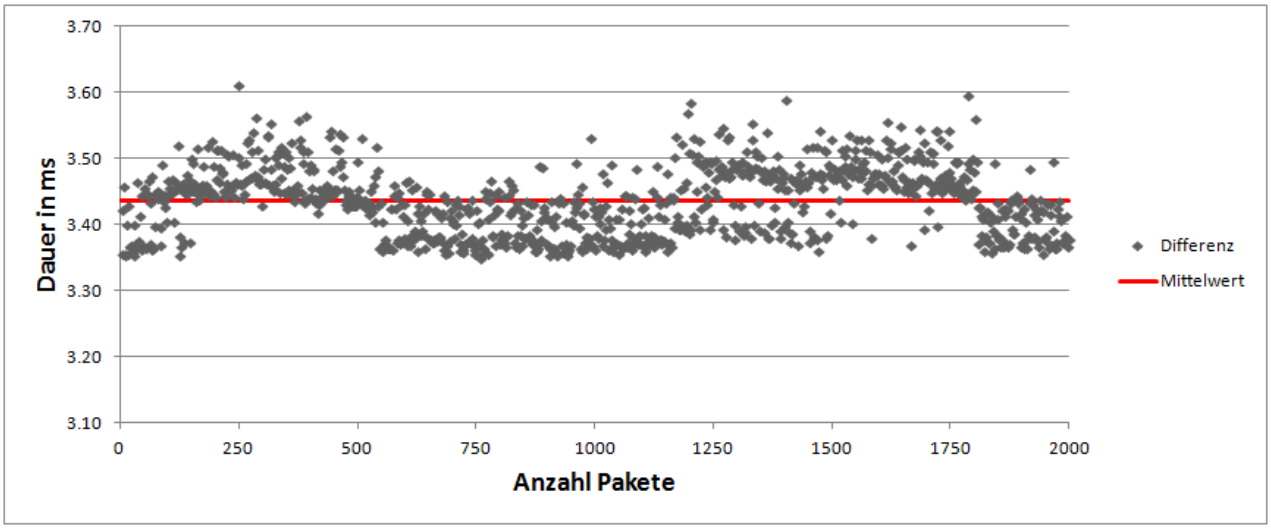

Abbildung 6.10.: Round Trip Time bei 100 Byte mit Encryption

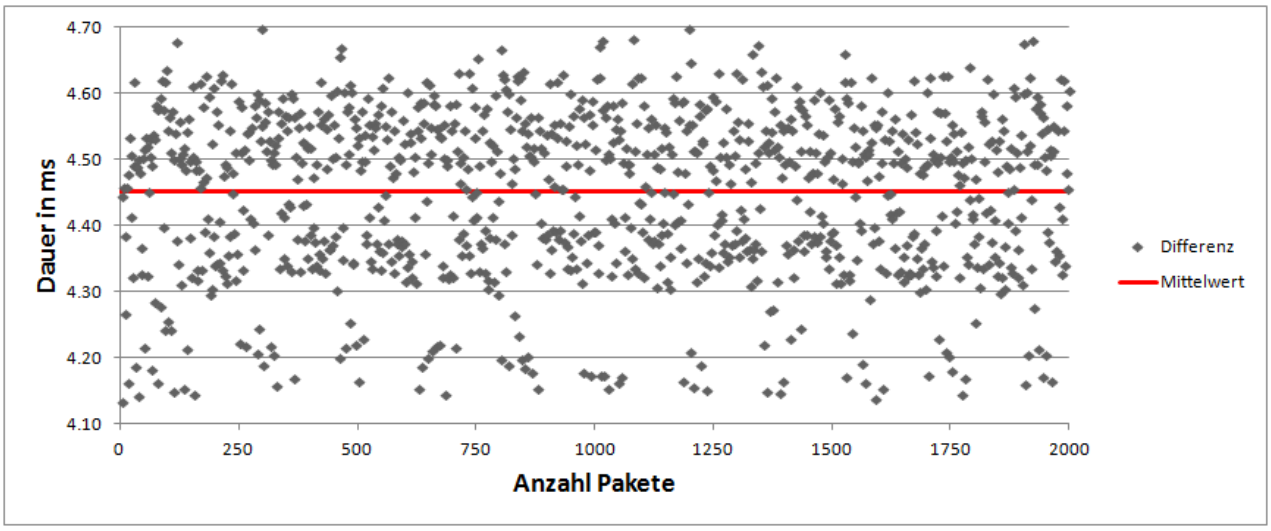

Abbildung 6.11.: Round Trip Time bei 100 Byte mit Authentication und Encryption

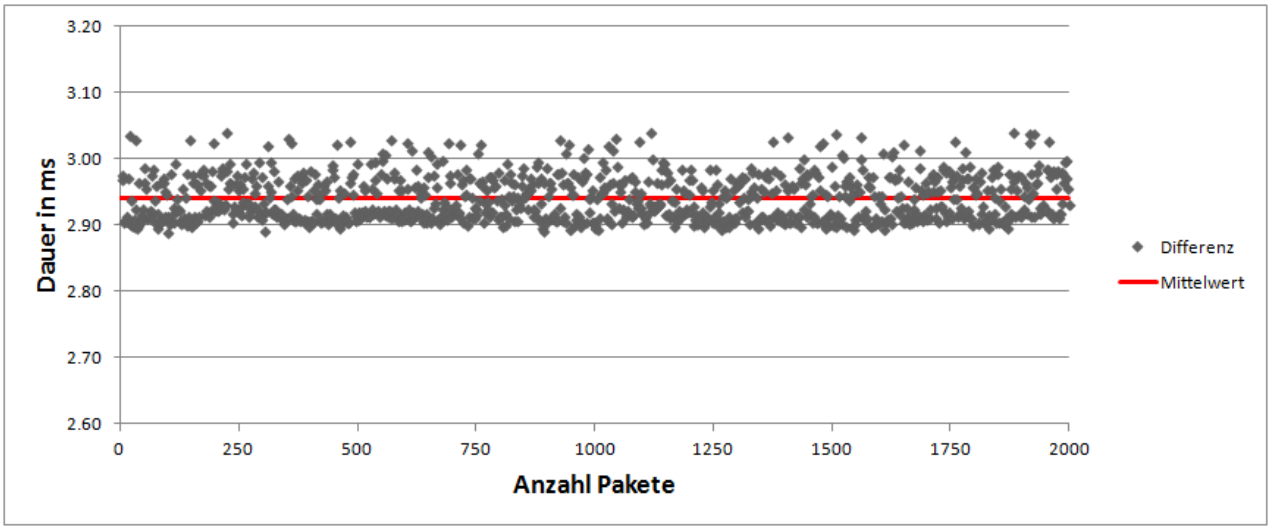

Abbildung 6.12.: Round Trip Time bei 200 Byte Plaintext

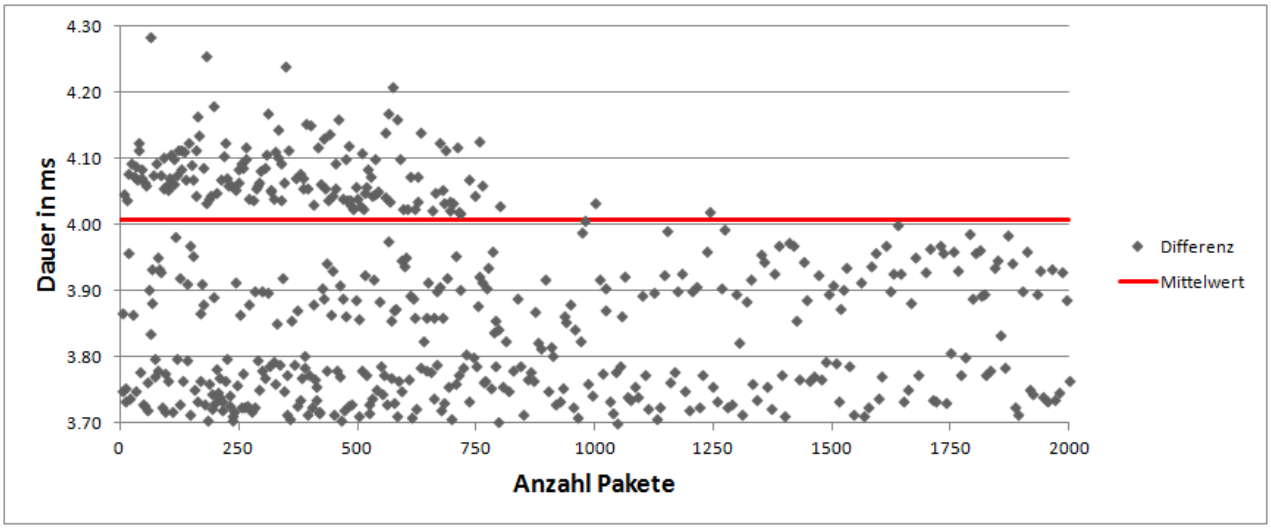

Abbildung 6.13.: Round Trip Time bei 200 Byte mit Authentication

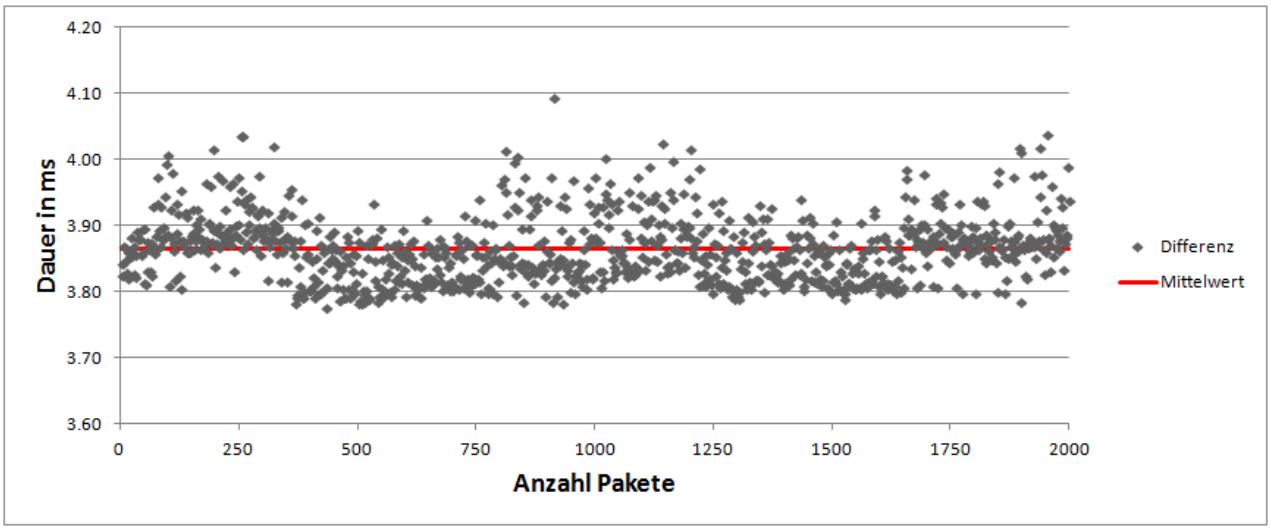

Abbildung 6.14.: Round Trip Time bei 200 Byte mit Encryption

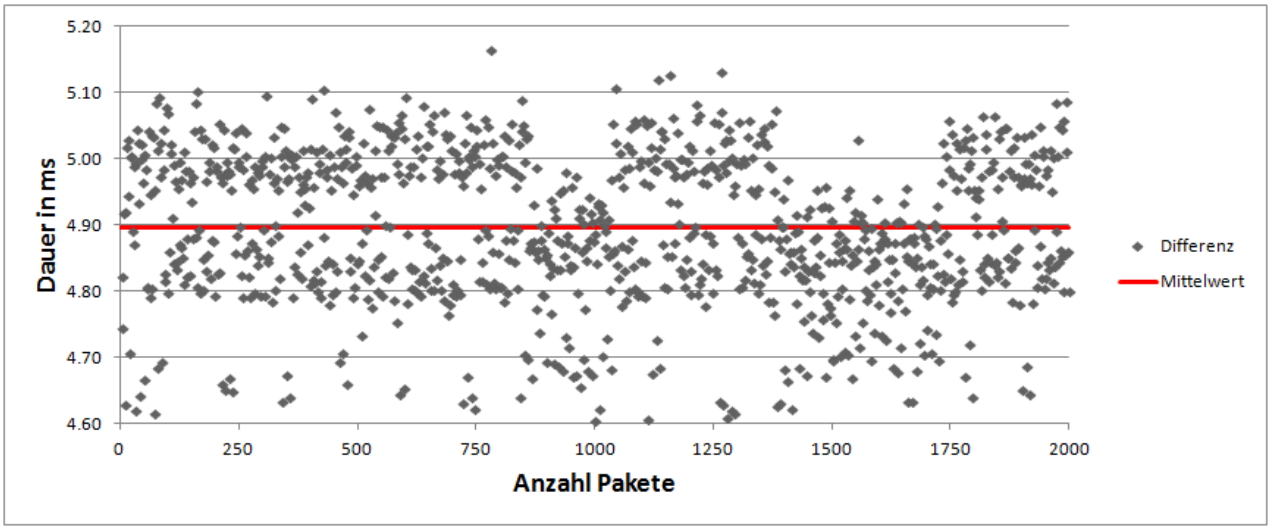

Abbildung 6.15.: Round Trip Time bei 200 Byte mit Authentication und Encryption

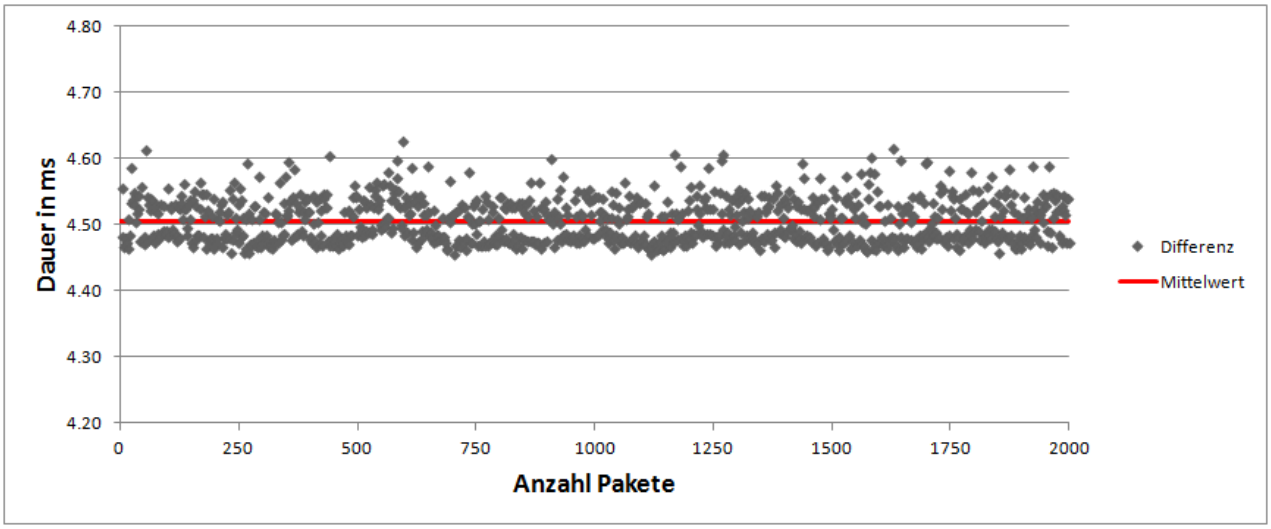

Abbildung 6.16.: Round Trip Time bei 500 Byte Plaintext

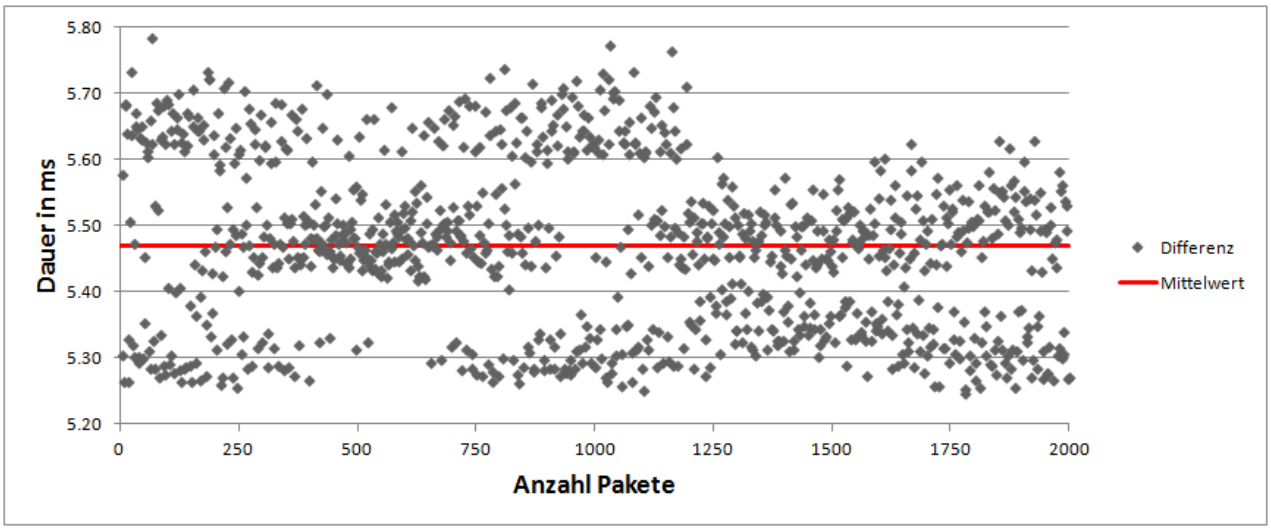

Abbildung 6.17.: Round Trip Time bei 500 Byte mit Authentication

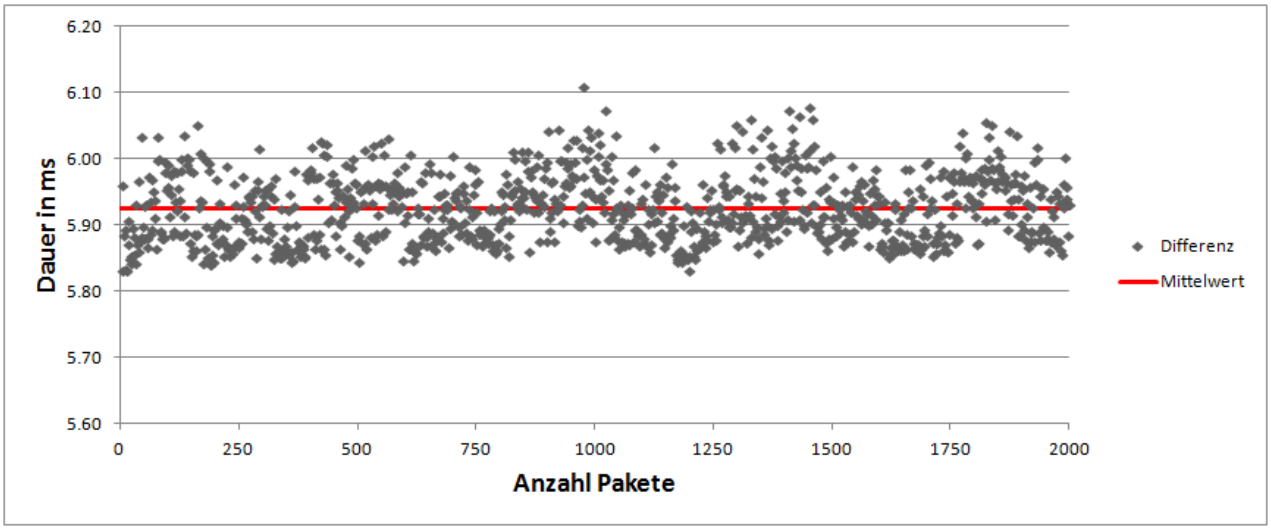

Abbildung 6.18.: Round Trip Time bei 500 Byte mit Encryption

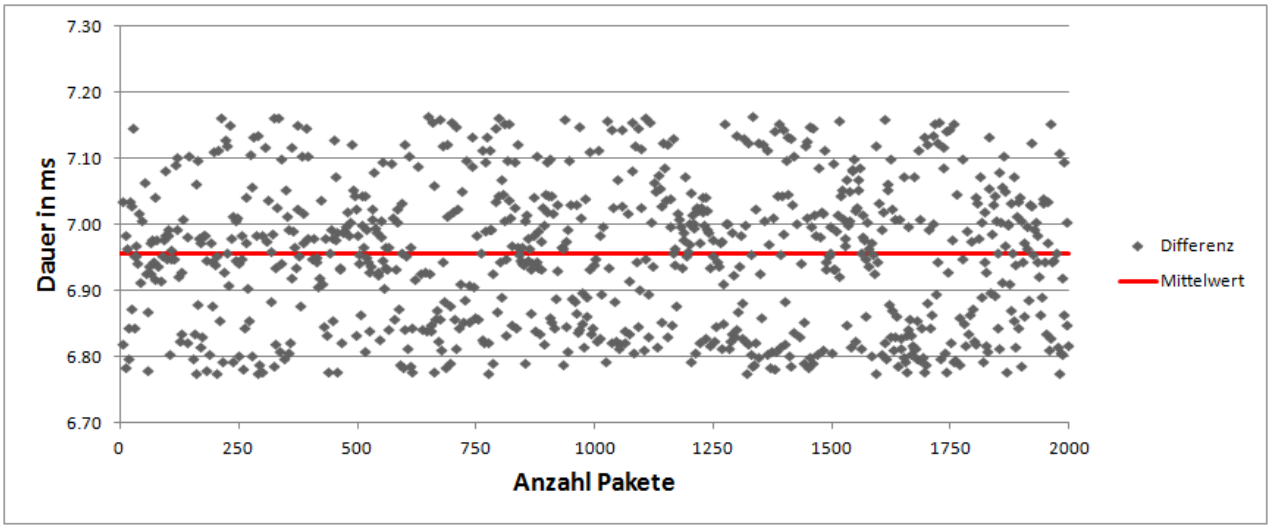

Abbildung 6.19.: Round Trip Time bei 500 Byte mit Authentication und Encryption

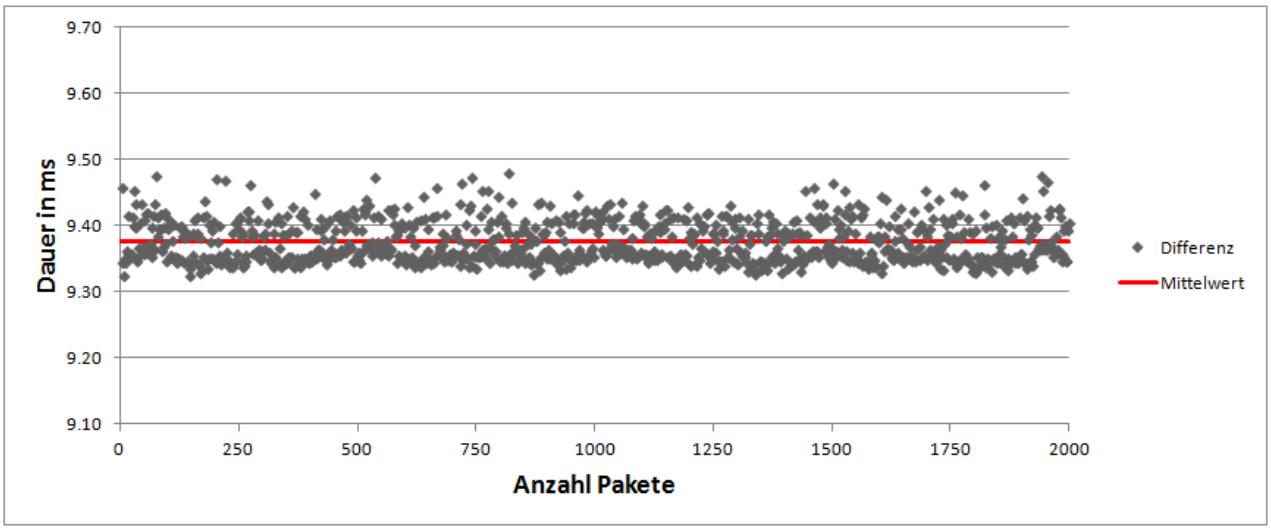

Abbildung 6.20.: Round Trip Time bei 1000 Byte Plaintext

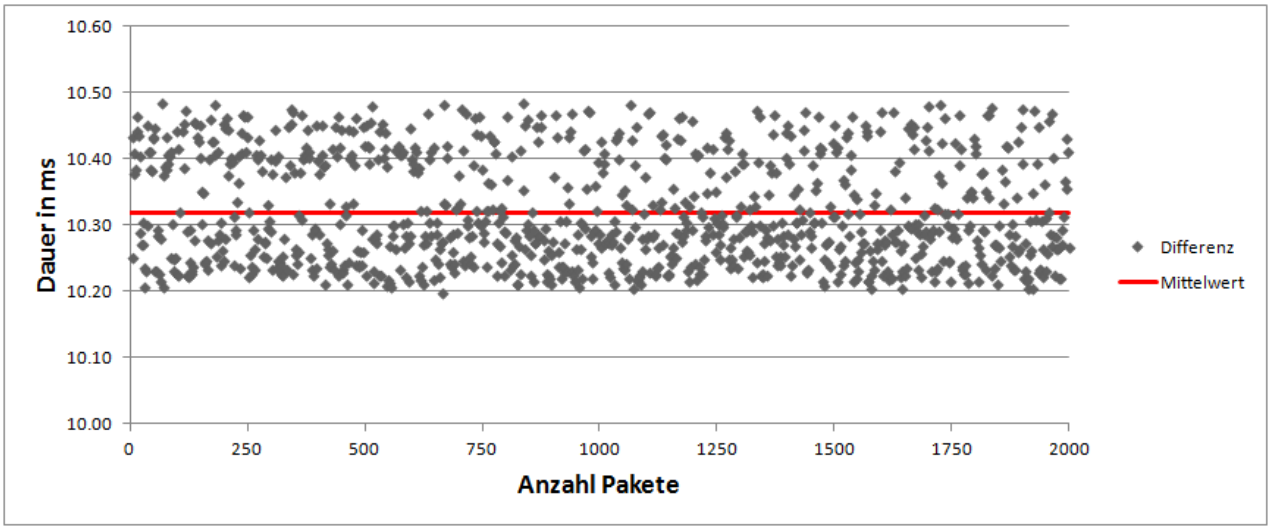

Abbildung 6.21.: Round Trip Time bei 1000 Byte mit Authentication

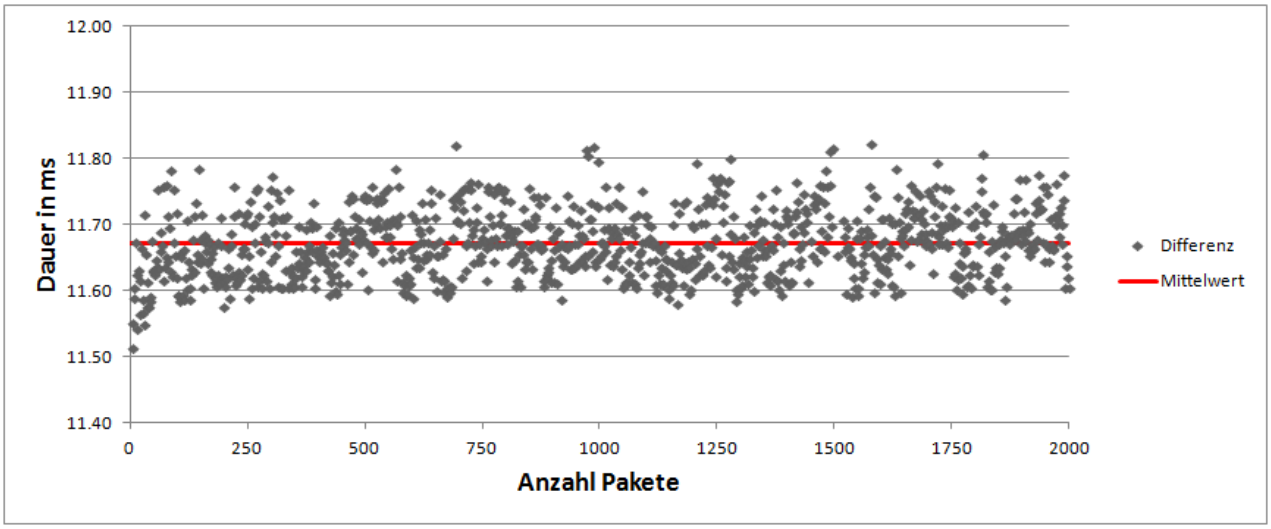

Abbildung 6.22.: Round Trip Time bei 1000 Byte mit Encryption

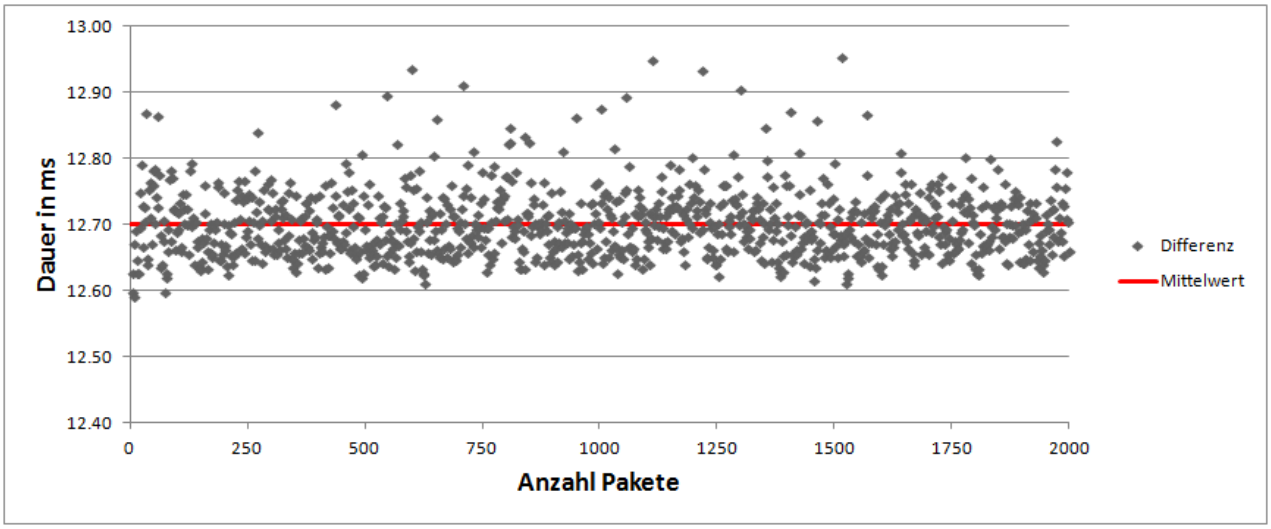

Abbildung 6.23.: Round Trip Time bei 1000 Byte mit Authentication und Encryption

Gegenüberstellung

	20 B	100 B	200 B	500 B	1000 B
Plaintext	2.508	2.650	2.940	4.505	9.375
Authentication	3.454	3.583	4.007	5.469	10.318
Encryption	3.117	3.436	3.864	5.925	11.672
Authentication und Encryption	4.137	4.452	4.895	6.955	12.700

Tabelle 6.2.: Gegenüberstellung der Mittelwerte [ms]

	20 B	100 B	200 B	500 B	1000 B
Plaintext	0	0	0	0	0
Authentication	0.946	0.934	1.067	0.964	0.942
Encryption	0.609	0.786	0.924	1.420	2.297
Authentication und Encryption	1.629	1.802	1.955	2.449	3.325

Tabelle 6.3.: Gegenüberstellung der Mittelwerte [ms] mit Abzug der Grundzeiten

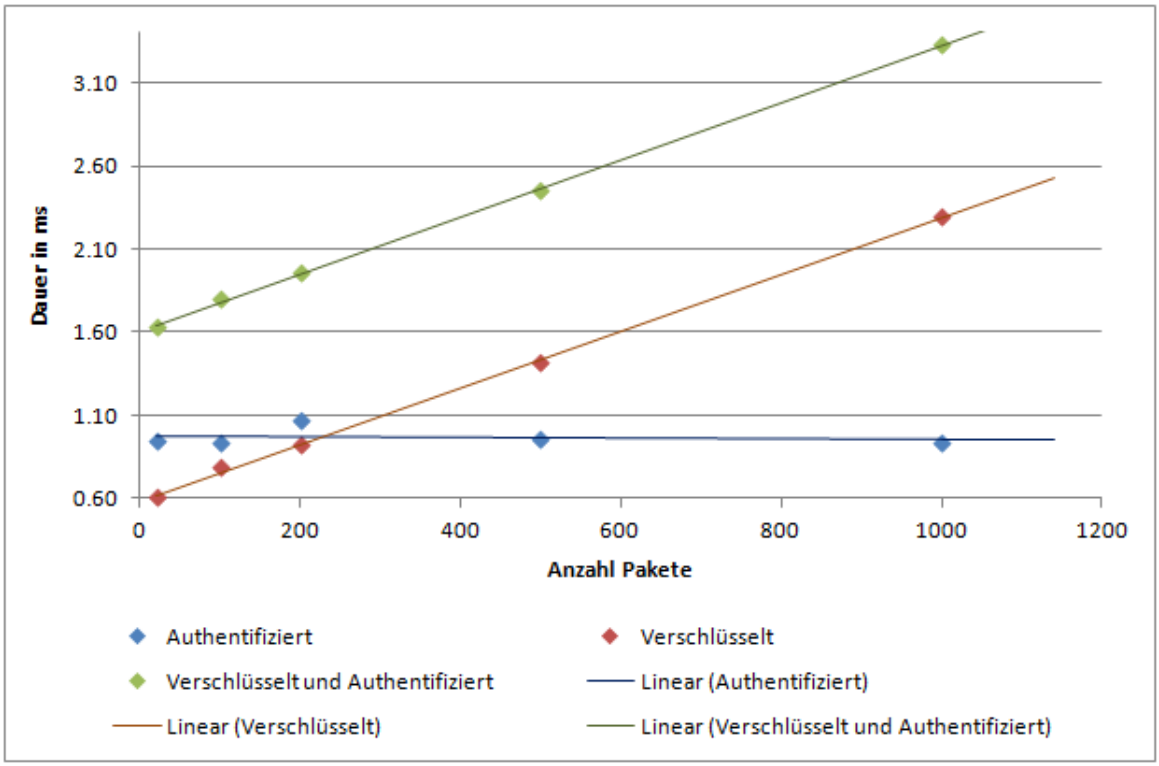

Abbildung 6.24.: Gesamtuebersicht

Interpretation

Aus der Abbildung 6.24 ist ersichtlich, dass die Authentisierung bei allen Datenpaketgrössen ungefähr gleich lange braucht. Dies macht Sinn, da bei einer Authentisierung lediglich Daten an das Paket angehängt werden. Bei der Verschlüsselung der Daten, lässt sich erkennen, dass bei zunehmender Grösse die aufgewendete Zeit linear ansteigt. Dass die Verschlüsselungszeit bei zunehmender Paketgrösse zunimmt, macht ebenfalls Sinn, da mehr Daten verschlüsselt werden müssen. Wenn man den Graph für die Kombination von Verschlüsselung und Authentisierung betrachtet sieht man, dass er sich ziemlich exakt aus der Addition der einzelnen Graphen für die Verschlüsselung bzw. Authentisierung zusammensetzt. Die Ergebnisse der Zeitmessung zeigen, dass es technisch möglich ist mit diesem Embedded System und dem GNU ccRTP Stack eine VoIP-Lösung mit Secure RTP zu implementieren, ohne Gefahr zu laufen, dass die Kommunikation gross verzögert wird. Die Zeiten aus der Tabelle 6.3 zeigen, dass die Verschlüsselung und Authentisierung, selbst bei Datenpaketgrössen von 1000 Byte, maximal 3.325 ms verzögert.

6.2.2. Auslastungstest

Die Auswertung hat ergeben, dass ein Durchsatz von 768 kbit/s keine Belastung für das System darstellt, in Zahlen ausgedrückt 0%. Auch die Auswahl der unterschiedlichen Modi hat keinen Einfluss auf die Auslastung. Ergänzend zu den Resultaten muss jedoch erwähnt werden, dass maximal alle 10 ms ein Paket versendet werden kann. Dabei spielt die Grösse der Pakete keine Rolle, es ist auf die Anzahl der Pakete limitiert. Dies ist jedoch vertretbar, da fast alle Sprach-Codecs mit Frames von 10-20 ms arbeiten. Die Anforderung von zwei Mal 192 kbit/s in Senderichtung und zwei Mal 192 kbit/s in Empfangsrichtung ist damit erfüllt. Sollte ein grösserer Durchsatz gefragt sein, können mehrere Threads gestartet werden.

Die Resultate wurden mit dem top-Befehl von Linux verglichen. Dieser gab ebenfalls den Wert von 0% aus.

Um die Daten zu verifizieren haben wir eine einfache Applikation entwickelt, welche über einen simplen UDP-Socket Pakete versendet. Diese Applikation besteht nur aus einem einfachen Socket, welcher die Daten auf das Netzwerk bringt. Dabei werden die Pakete in einer Endlosschleife auf das Netzwerk gebracht. Zwischen den einzelnen Paketen wird die Wartezeit abgewartet. Bei einer Datenrate von 192 kbit/s ergab sich eine Auslastung von 0%. Diese Auslastung hat sich bis zur Datenrate von 768 kbit/s nicht verändert.

Interpretation

Die Werte lassen darauf schliessen, dass ein Taurus Eval Board zwei bidirektional Verbindungen à 192 kbit/s verarbeiten kann.

Teil IV.

Anhang

7. Quellenverzeichnis

[1] RTP: A Transport Protocol for Real-Time Applications, http://www.ietf.org/rfc/rfc3550.txt, letzter Zugriff am 01.12.2011

[2] User Datagram Protocol, http://www.ietf.org/rfc/rfc768.txt, letzter Zugriff am 12.12.2011

[3] RTP Profile for Audio and Video Conferences with Minimal Control - Payload Type Definitions, http://tools.ietf.org/html/rfc3551#section-6, letzter Zugriff am 25.11.2011

[4] The Secure Real-time Transport Protocol (SRTP), http://www.ietf.org/rfc/rfc3711.txt, letzter Zugriff am 14.12.2011

[5] Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip, letzter Zugriff am 19.12.2011

[6] Transmission Control Prtocol, http://www.ietf.org/rfc/rfc793.txt, letzter Zugriff am 25.11.2011

[7] Bruce Schneier, Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C Datenverschlüsselung, Addison-Wesley, 1997, ISBN 3-89319-854-7, S. 63-67

[8] Bruce Schneier, Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C Datenverschlüsselung, Addison-Wesley, 1997, ISBN 3-89319-854-7, S. 2

[9] Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP), http://www.ietf.org/rfc/rfc3686.txt, letzter Zugriff am 14.12.2011

[10] Aus InternetSicherheit 2, Foliensatz VoIP-Security.

[11] The Secure Real-time Transport Protocol (SRTP), http://www.ietf.org/rfc/rfc3711.txt, letzter Zugriff am 28.11.2011

[12] Bruce Schneier, Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C Datenverschlüsselung, Addison-Wesley, 1997, ISBN 3-89319-854-7, S. 520-524

[13] HMAC: Keyed-Hashing for Message Authentication, http://www.ietf.org/rfc/rfc2104.txt, letzter Zugriff am 16.12.2011

[14] GNU ccRTP, http://www.gnu.org/s/ccrtp, letzter Zugriff am 22.12.2011

[15] The Ångström Distribution - Feed Browser, http://www.angstrom-distribution.org/repo/, letzter Zugriff am 22.12.2011

8. Tabellenverzeichnis

9. Bilderverzeichnis

10. Abkürzungsverzeichnis

VoIP	Voice over Internet Protocol
OSI	Open Systems Interconnection
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
RTP	Real-time Transport Protocol
SRTP	Secure RTP
AES	Advanced Encryption Standard
MAC	Message Authentication Code
HMAC	Hash-based MAC
SHA	Secure Hash Algorithm
CSRC	Contributing source
SSRC	Synchronization source
MKI	Master Key Identifier

DES	Data Encryption Standard
IDEA	International Data Encryption Algorithm
CCP	Common Controller Platform
SO-DIMM	Small Outline Dual Inline Memory Module. Arbeitsspeicher der wegen seiner kompakten Bauform vorallem in Notebooks eingesetzt wird.
ARM	Advanced RISC Machines. Britischer Hersteller von Chipsätzen.
SDRAM	Synchronous Dynamic Random Access Memory. Art eines Arbeitsspeichers bei Computern.
SRAM	Static Random-Access Memory. Flüchtiger Speicher.