Ostfalia
Hochschule für angewandte Wissenschaften

Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb

Masterarbeit

Martin Langer
Matrikelnummer: 70405974
Eingereicht am: 21.10.2016

Erstprüfer: Prof. Dr.-Ing. Rainer Bermbach
Zweitprüfer: Prof. Dr.-Ing. Diederich Wermser

Ostfalia, Wolfenbüttel
Fakultät Elektrotechnik
Studiengang: Intelligente Mobilität und Energiesystemtechnik

Dokumentversion: 1.0.1.3122
Letze Änderung: 10.11.2016 09:04:00

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

_____	_____
Ort, Datum	Unterschrift des Verfassers

Danksagung

Die Bearbeitung und Umsetzung der Abschlussarbeit ist für viele Studenten ein großer Meilenstein im Leben. Daher möchte an dieser Stelle die Gelegenheit nutzen, an all jene meinen Dank aussprechen die zum Gelingen dieser Arbeit beigetragen haben.

Mein größter Dank gilt Herrn Prof. Dr.-Ing. R. Bermbach für die umfassende Betreuung meiner Arbeit. Seine unermüdliche Geduld und seine vielen hilfreichen Ratschläge, haben mich sehr unterstützt und motiviert. Mein weiterer Dank gilt Herrn Prof. Dr.-Ing. D. Wermser, der sich als Zweitprüfer bereit erklärt hat und somit zu meinem Abschluss beiträgt. Herrn K. Heine danke ich für die Bereitstellung von Räumlichkeiten und Arbeitsmitteln in der Ostfalia. Des Weiteren danke ich auch den Kommilitonen S. Häußler, C. Jütte, S. König, T. Kompa und T. Tuschik, die im Rahmen ihrer Projektarbeit zur Umsetzung meiner Arbeit beitrugen.

Ebenfalls möchte ich mich herzlich bei Herrn Dr. D. Sibold und Herrn K. Teichel von der Physikalisch-Technischen Bundesanstalt bedanken. Die gute Zusammenarbeit und die konstruktiven Gespräche waren eine wichtige Grundlage zur erfolgreichen Umsetzung der Arbeit.

Auch meiner Familie danke ich vom ganzen Herzen, die mich die vielen Jahre meines Studiums unterstützt und motiviert hat. Ihr habe ich es zu verdanken, dass ich diesen Weg gehen konnte.

Abstract

Die technischen Entwicklungen in den letzten Jahrzehnten führten zu einem drastischen Anstieg der Vernetzungsdichte von Computersystemen. Je nach Anwendungsgebiet sind genaue und zuverlässige Zeitinformationen vonnöten, damit derartige Systeme ihre Tätigkeiten ausüben können. Die Bereitstellung dieser Daten erfolgt heutzutage durch weit verbreitete Zeitsynchronisierungsprotokolle, wie etwa das *Network Time Protocol* (NTP) oder das *Precision Time Protocol* (PTP), mit denen sich große Computernetze zeitlich synchronisieren lassen. Aufgrund der steigenden Nachfrage an Sicherheit, die auch die unverfälschte Übertragung von Zeitdaten beinhaltet, wurden im Jahr 2011 die bestehenden Sicherheitsmechanismen der Zeitprotokolle untersucht. Das Ergebnis zeigte gravierende Schwächen und setzte dadurch die Hebel zur Ausarbeitung einer neuen und zeitgemäßen Spezifikation in Gange. Fortan wurde an einem neuen Protokoll unter dem Namen *Network Time Security* (NTS) gearbeitet, welches sich derzeit in der Entwicklungsphase befindet und zukünftig die erforderliche Sicherheit garantieren soll.

Diese Arbeit beschäftigt sich mit Aufbau, Ablauf und Implementierung des NTS-Protokolls im Unicast-Betriebsmodus und dessen Einbettung in NTPv4. Dazu folgen eine detaillierte Beschreibung zum Softwaredesign und der konzeptionellen Vorgehensweise, die während der Umsetzung des NTS-Protokolls Anwendung fand. Die entwickelte Software und die anschließenden Tests sollen hierbei den Beweis zu korrekten Funktion der NTS-Spezifikation liefern und darüber hinaus eine Referenzimplementierung für zukünftige Entwicklungen bieten.

Inhaltsverzeichnis

Abkürzungsverzeichnis **vii**

Abbildungsverzeichnis **x**

Gleichungsverzeichnis **xii**

Listing-Verzeichnis **xiii**

Tabellenverzeichnis **xv**

1 Einführung **1**

2 Allgemeine Grundlagen **3**

2.1 Das OSI-Referenzmodell ... 3
2.2 Zeitsynchronisierungsdienste ... 6
2.2.1 Network Time Protocol (NTP) ... 6
2.2.2 Precision Time Protocol (PTP) ... 9
2.3 Abstract Syntax Notation One (ASN.1) ... 9
2.3.1 Kommunikation zwischen heterogenen Systemen ... 10
2.3.2 Die Syntax in ASN.1 ... 11
2.3.3 Kodierungsverfahren ... 12
2.3.3.1 Basic Encoding Rules (BER) ... 12
2.3.3.2 Distinguished Encoding Rules (DER) ... 12
2.3.3.3 XML Encoding Rules (XER) ... 13
2.4 Kryptografie ... 13
2.4.1 Verschlüsselungsverfahren ... 13
2.4.1.1 Symmetrische Verschlüsselung ... 14
2.4.1.2 Asymmetrische Verschlüsselung ... 15
2.4.1.3 Hybride Verschlüsselung ... 15
2.4.2 Digitale Fingerabdrücke und Signaturen ... 16
2.4.2.1 Kryptografische Hash-Funktionen ... 16
2.4.2.2 MAC-Funktionen ... 17
2.4.2.3 Digitale Signatur ... 17
2.4.3 Zertifikate und Zertifizierungsstruktur ... 19
2.5 Cryptographic Message Syntax (CMS) ... 20

3 Das Network-Time-Security-Protokoll **21**

3.1 Übersicht zu NTS ... 22
3.1.1 Entwicklungsverlauf und aktueller Stand der Spezifikation ... 22
3.1.2 NTP-bezogene Sicherheitsprobleme ... 24
3.1.3 Probleme bestehender Sicherheitsmechanismen in NTP ... 28

3.1.3.1 Das *Symmetric-Key*-Verfahren ... 28
3.1.3.2 Das *Autokey*-Verfahren ... 29

3.1.4 Sicherheitsziele von NTS ... 32
3.1.5 NTS im OSI-Referenzmodell ... 34

3.2 NTS im Unicast-Betrieb ... 35
3.2.1 Eigenschaften der Kommunikationspartner ... 35
3.2.1.1 Eigenschaften des NTS-Clients ... 35
3.2.1.2 Eigenschaften des NTS-Servers ... 36

3.2.2 Übersicht zur Funktionsweise des Unicast-Betriebs ... 36

3.3 Nachrichtentypen im Unicast-Betrieb ... 40
3.3.1 Zugriffsnachrichten ... 41
3.3.1.1 Die *client_access*-Nachricht ... 41
3.3.1.2 Die *server_access*-Nachricht ... 41

3.3.2 Assoziationsnachrichten ... 42
3.3.2.1 Die *client_assoc*-Nachricht ... 42
3.3.2.2 Die *server_assoc*-Nachricht ... 43

3.3.3 Cookie-Nachrichten ... 44
3.3.3.1 Die *client_cook*-Nachricht ... 45
3.3.3.2 Die *server_cook*-Nachricht ... 45

3.3.4 Zeitnachrichten ... 46
3.3.4.1 Die *time_request*-Nachricht ... 46
3.3.4.2 Die *time_response*-Nachricht ... 47

3.4 NTS-Nachrichtenstruktur ... 48
3.4.1 Die NTPv4-Erweiterungsfelder ... 49
3.4.1.1 Aufbau der Erweiterungsfelder ... 50
3.4.1.2 Verwendung der Erweiterungsfelder in NTS ... 50

3.4.2 Archetypen ... 53
3.4.2.1 NTS-Plain ... 53
3.4.2.2 NTS-Signed ... 54
3.4.2.3 NTS-Encrypted-and-Signed ... 55

3.4.3 CMS-Strukturen ... 56
3.4.3.1 ContentInfo ... 56
3.4.3.2 SignedData ... 57
3.4.3.3 EnvelopedData ... 61

3.4.4 NTS-Nachrichtenobjekte ... 64
3.4.4.1 ClientAccessData ... 64
3.4.4.2 ServerAccessData ... 65
3.4.4.3 ClientAssocData ... 65

3.4.4.4 ServerAssocData 67
3.4.4.5 ClientCookieData 69
3.4.4.6 ServerCookieData 71
3.4.4.7 TimeRequestSecurityData 72
3.4.4.8 TimeResponseSecurityData 73
3.4.4.9 NTSMessageAuthenticationCode 74

3.5 Kommunikationsablauf im Unicast-Betrieb 75
3.5.1 Zugriffsphase 76
3.5.2 Assoziationsphase 77
3.5.3 Cookie-Austauschphase 80
3.5.4 Zeitsynchronisationsphase 83

3.6 Gesonderte Themen 85
3.6.1 Die NTS-Version 85
3.6.2 Der Server Seed 89
3.6.3 Vergleich der MAC-Algorithmen 90
3.6.4 Zertifikatseigenschaften und -regelungen 92
3.6.4.1 Das Gültigkeitsdatum 93
3.6.4.2 Die Zertifikatserweiterungen 93
3.6.4.3 Zertifikatidentifizierung 96
3.6.4.4 Beispiele für gültige Zertifikatkonfigurationen 98

3.6.5 Empfehlungen für Schlüssel und Algorithmen 101
3.6.6 Definition der Fehlercodes 103
3.6.7 Zulässige und sinnvolle Paketgrößen bei NTS 104
3.6.8 Latenzzeiten und verursachte Ungenauigkeiten durch NTS 105
3.6.8.1 Latenzzeiten bei ungesichertem NTP 106
3.6.8.2 Latenzzeiten bei NTS-gesichertem NTP 106

4 Implementierung von NTS 109

4.1 Betriebssystem und Entwicklungsumgebung 109
4.1.1 Das eingesetzte Betriebssystem 109
4.1.2 Programmiersprache und Entwicklungswerkzeuge 110
4.1.3 Externe Bibliotheken 113
4.1.3.1 Die ausgewählte ASN.1-Bibliothek: asn1c 114
4.1.3.2 Die ausgewählte Kryptografie-Bibliothek: OpenSSL 115

4.2 Softwarestruktur und Programmmodule 115
4.2.1 Das ASN.1-Modul 116
4.2.1.1 ASN.1-Basisfunktionen (Wrapper) 118
4.2.1.2 Der Aufbau der NTS-Nachrichtenobjekte 119

4.2.2 Das Kryptografiemodul 123
4.2.3 Das Client-Unicast-Modul 129
4.2.4 Das Server-Unicast-Modul 132
4.2.5 Das NTS-Unicast-Service-Modul 134

4.3 Schnittstellen zu NTP 136

4.4 Programmablauf der Implementierung ... 142
4.4.1 Die Initialisierungsphase ... 143
4.4.2 Verarbeitungsprozess im NTS-Unicast-Service-Modul ... 143
4.4.3 Verarbeitungsprozess im Client-Unicast-Modul ... 144
4.4.3.1 Verarbeitung empfangener NTS-Nachrichten ... 144
4.4.3.2 Generierung neuer NTS-Nachrichten ... 146
4.4.4 Verarbeitungsprozess im Server-Unicast-Modul ... 147
4.5 Konfigurierung des NTS-Dienstes ... 149
4.5.1 Die globale NTS-Konfigurationsdatei ... 150
4.5.2 Die Konfigurationsdatei des Client-Unicast-Moduls ... 153
4.5.3 Die Konfigurationsdatei des Server-Unicast-Moduls ... 157
4.6 Projektaufbau und Entwicklungsstand ... 159
4.6.1 Projektorganisation der NTS-Implementierung ... 160
4.6.2 Fortschritt und Umfang der NTS-Implementierung ... 161
4.6.3 Implementierungsstatistiken ... 162

5 Tests und Ergebnisse 167

5.1 Messung der NTS-Nachrichtengrößen ... 167
5.1.1 Minimale Nachrichtengrößen in NTS ... 167
5.1.2 Maximale Nachrichtengrößen in NTS ... 169
5.2 Messung der benötigten Ressourcen ... 170
5.3 Messung der Verarbeitungsdauer ... 171
5.4 Test unter realen Bedingungen über NTPv4 ... 173

6 Weiterführende Entwicklungen 175

6.1 Herausforderungen der NTS-Spezifikation ... 175
6.1.1 Anpassung der Verhandlungsphase (Key Exchange) ... 175
6.1.2 Probleme mit Filterung und IP-Fragmentierung ... 177
6.2 Weitere Änderungen der NTS-Spezifikation ... 179
6.2.1 Probleme durch den Ersatz von Hash- durch MAC-Algorithmen ... 179
6.2.2 Die Wahl des *keyEncryption*-Algorithmus ... 181
6.2.3 Der verwendete Signaturalgorithmus ... 181
6.3 Weiterentwicklung der NTS-Implementierung ... 182
6.3.1 Ausgiebige Tests mit geeigneten Testfällen ... 182
6.3.2 Optimierung der NTS-Implementierung ... 182
6.3.3 Funktionale Erweiterungen ... 183

7 Zusammenfassung 187

Quellenverzeichnis I

Anhang XVI

Abkürzungsverzeichnis

AES	**A**dvanced **E**ncryption **S**tandard
API	**A**pplication **P**rogramming **I**nterface
ARP	**A**ddress **R**esolution **P**rotocol
ASN.1	**A**bstract **S**yntax **N**otation **O**ne
ATM	**A**synchronous **T**ransfer **M**ode
BER	**B**asic **E**ncoding **R**ules
BSI	**B**undesamt für **S**icherheit in der **I**nformationstechnik
CA	**C**ertificate **A**uthority
CBC	**C**ipher **B**lock **C**haining Mode
CDT	**C**/C++ **D**evelopment **T**ooling
CMS	**C**ryptographic **M**essage **S**yntax
CRC	**C**yclic **R**edundancy **C**heck
CRL	**C**ertificate **R**evocation **L**ist
DER	**D**istinguished **E**ncoding **R**ules
DES	**D**ata **E**ncryption **S**tandard
DH	**D**iffie-**H**ellman
DIN	**D**eutsches **I**nstitut für **N**ormung
DoS	**D**enial **o**f **S**ervice
DDoS	**D**istributed **D**enial **o**f **S**ervice
DNS	**D**omain **N**ame **S**ystem
DSA	**D**igital **S**ignature **A**lgorithm
DTLS	**D**atagram **T**ransport **L**ayer **S**ecurity
EC	**E**lliptic **C**urve
ECB	**E**lectronic **C**ode **B**ook Mode
ECC	**E**lliptic **C**urve **C**ryptography
ECDSA	**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
FDDI	**F**iber **D**istributed **D**ata **I**nterface
FTP	**F**ile **T**ransfer **P**rotocol
GCC	**G**NU **C**ompiler **C**ollection
HMAC	Keyed-**H**ash **M**essage **A**uthentication **C**ode
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment
IEC	**I**nternational **E**lectrotechnical **C**ommission

IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IGMP	**I**nternet **G**roup **M**anagement **P**rotocol
ITU-T	**I**nternational **T**elecommunication **U**nion - **T**elecommunications Sector
IP	**I**nternet **P**rotocol
IPv4	**I**nternet **P**rotocol **v**ersion **4**
IPv6	**I**nternet **P**rotocol **v**ersion **6**
IPX	**I**nternetwork **P**acket **E**xchange
ISO	**I**nternational **O**rganization for **S**tandardization
IT	**I**nformations**t**echnik
KIV	**K**ey **I**nput **V**alue
KoD	**K**iss-**o**f-**d**eath Packet
LAN	**L**ocal **A**rea **N**etwork
LLC	**L**ogical **L**ink **C**ontrol
LSB	**L**east **S**ignificant **B**it
LXDE	**L**ightweight **X**11 **D**esktop **E**nvironment
MAC	**M**essage **A**uthentication **C**ode / medium access control
MD5	**M**essage-**D**igest Algorithm **5**
MITM	**M**an-**I**n-**T**he-**M**iddle
MSB	**M**ost **S**ignificant **B**it
MTU	**M**aximum **T**ransmission **U**nit
NID	**N**umerical **Id**entifier
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NTF	**N**etwork **T**ime **F**oundation
NTP	**N**etwork **T**ime **P**rotocol
NTS	**N**etwork **T**ime **S**ecurity
OCSP	**O**nline **C**ertificate **S**tatus **P**rotocol
OID	**O**bject **Id**entifier
OSI	**O**pen **S**ystems **I**nterconnection
PDU	**P**rotocol **D**ata **U**nit
PEM	**P**rivacy-enhanced **E**lectronic **M**ail
PKCS	**P**ublic **K**ey **C**ryptography **S**tandards
PKI	**P**ublic-**K**ey **I**nfrastructure
PKIX	**P**ublic-**K**ey **I**nfrastructure **X**.509
PSK	**P**re-**S**hared **K**ey
PTB	**P**hysikalisch-**T**echnische **B**undesanstalt
PTP	**P**recision **T**ime **P**rotocol
RAII	**R**esource **A**cquisition **I**s **I**nitialization

RC4	**R**on's **C**ode 4
RFC	**R**equests **f**or **C**omments
RSA	**R**ivest, **S**hamir und **A**dleman
S/MIME	**S**ecure / **M**ultipurpose **I**nternet **M**ail **E**xtensions
SCTP	**S**tream **C**ontrol **T**ransmission **P**rotocol
SHA	**S**ecure **H**ash **A**lgorithm
SM	**S**mart **M**eter
SMGW	**S**mart **M**eter **G**ateway
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SPX	**S**equenced **P**acket **E**xchange
SSL	**S**ecure **S**ockets **L**ayer
TCP	**T**ransmission **C**ontrol **P**rotocol
TESLA	**T**imed **E**fficient **S**tream **L**oss-tolerant **A**uthentication
TBD	**T**o **B**e **D**efined
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TLV	**T**ype-**L**ength-**V**alue
TR	**T**echnische **R**ichtlinie
TU	**T**echnische **U**niversität
UDP	**U**ser **D**atagram **P**rotocol
WAN	**W**ide **A**rea **N**etwork
WWW	**W**orld **W**ide **W**eb
XER	**X**ML **E**ncoding **R**ules
XML	E**x**tensible **M**arkup **L**anguage
XOR	E**x**clusive **Or**

Abbildungsverzeichnis

Abbildung 1.1: Symbolische Integration von NTS in Zeitprotokolle 2
Abbildung 2.1: Kommunikationswege im OSI-Referenzmodell 4
Abbildung 2.2: Protokolle der jeweiligen OSI-Schichten 4
Abbildung 2.3: Zeitstempel im NTPv4-Protokoll 8
Abbildung 2.4: Kommunikation heterogener Systeme 10
Abbildung 2.5: Grundprinzip der Verschlüsselung 13
Abbildung 2.6: Prinzip der symmetrischen Ver- und Entschlüsselung 14
Abbildung 2.7: Prinzip der asymmetrischen Ver- und Entschlüsselung 15
Abbildung 2.8: Prinzip zur Erzeugung eines digitalen Fingerabdrucks (Hash-Wert) 17
Abbildung 2.9: Prinzip zur Erzeugung eines MAC 17
Abbildung 2.10: Prinzip zur Erzeugung einer digitalen Signatur 18
Abbildung 2.11: Zerlegung des Signaturalgorithmus in seine Bestandteile 18
Abbildung 3.1: Angriffsszenario: Man-In-The-Middle-Attacke 24
Abbildung 3.2: Angriffsszenario: der Package Injector 25
Abbildung 3.3: Asymmetrische NTP-Paketlaufzeiten durch einen Packet-Delay-Angriff 27
Abbildung 3.4: Zusammensetzung der Autokey-Protokollelemente 30
Abbildung 3.5: Zuordnung von NTS im OSI-Referenzmodell 34
Abbildung 3.6: Kernphasen des NTS-Protokolls 37
Abbildung 3.7: Zusammensetzung der NTS-Protokollelemente 39
Abbildung 3.8: Untergliederung der NTS-Verhandlungsphase 40
Abbildung 3.9: Struktureller Aufbau von NTPv4-Paketen 48
Abbildung 3.10: Struktureller Aufbau von NTS-Inhalten 49
Abbildung 3.11: Struktureller Aufbau der NTPv4-Erweiterungsfelder 50
Abbildung 3.12: Berücksichtigte NTPv4-Felder zur Generierung der NTS-MAC 52
Abbildung 3.13: Aufbau des NTS-Plain-Archetyps 54
Abbildung 3.14: Aufbau des NTS-Signed-Archetyps 55
Abbildung 3.15: Aufbau des NTS-Encrypted-ans-Signed-Archetyps 55
Abbildung 3.16: Aufbau der CMS-Struktur: ContentInfo 56
Abbildung 3.17: Aufbau der CMS-Struktur: SignedData 57
Abbildung 3.18: Aufbau der CMS-Struktur: EnvelopedData 62
Abbildung 3.19: Allgemeiner Kommunikationsablauf des NTS-Protokolls 75
Abbildung 3.20: Geschwindigkeitsvergleich unterschiedlicher MAC-Algorithmen 92
Abbildung 3.21: Herleitung der zulässigen Größe der NTP-Erweiterungsfelder 105
Abbildung 3.22: Zusammensetzung eines Ethernet-Datenframes mit NTPv4 105
Abbildung 3.23: Zeitstempel einer NTP-Nachricht ohne NTS 106

Abbildung 3.24: Zeitstempel einer NTP-Nachricht mit NTS 107
Abbildung 4.1: LXDE-Desktopumgebung unter Linux Debian v8.5.0 110
Abbildung 4.2: Programmoberfläche der Eclipse-CDT-Entwicklungsumgebung 111
Abbildung 4.3: Beispielhafte Ausgabe des Eclipse-Plugins "Log Viewer" 112
Abbildung 4.4: Struktureller Aufbau der NTS-Implementierung für den Unicast-Betrieb 117
Abbildung 4.5: Aufbau eines NTS-Nachrichtenobjekts als C++-Klasse 120
Abbildung 4.6: Analyse eines ASN.1-Datenstroms mit dem Programm "ASN.1 Editor" 125
Abbildung 4.7: Aufbau des Client-Unicast-Moduls 130
Abbildung 4.8: Aufbau des Server-Unicast-Moduls 133
Abbildung 4.9: Aufbau des NTS-Unicast-Service-Moduls 135
Abbildung 4.10: Schnittstellenfunktionen zwischen NTP und NTS 137
Abbildung 4.11: Allgemeiner Nachrichtenfluss in NTS 142
Abbildung 4.12: Ordnerstruktur im Projektverzeichnis der NTS-Implementierung 160
Abbildung 4.13: Prozentuale Verteilung der Dateigrößen in der NTS-Implementierung 164
Abbildung 4.14: Prozentuale Verteilung der Funktionen in der NTS-Implementierung 164
Abbildung 4.15: Quellcode-Zusammensetzung der NTS-Implementierung 166
Abbildung 5.1: Arbeitsspeicherbedarf der NTS-Client-Anwendung mit 30 Zeitserver 171

Gleichungsverzeichnis

Gleichung 2.1: Berechnung des Delays in NTP 8
Gleichung 2.2: Berechnung des Offsets in NTP 8
Gleichung 3.1: Cookie- Generierung im Autokey-Verfahren 29
Gleichung 3.2: Autokey-Generierung im Autokey-Verfahren 29
Gleichung 3.3: MAC-Generierung im Autokey-Verfahren 30
Gleichung 3.4: Cookie-Berechnung im NTS-Protokoll 38
Gleichung 3.5: KIV-Berechnung im NTS-Protokoll 38
Gleichung 3.6: MAC-Berechnung im NTS-Protokoll 38
Gleichung 3.7: Access-Key-Berechnung im NTS-Protokoll 41
Gleichung 3.8: Access-Key-Berechnung im NTS-Protokoll 76
Gleichung 3.9: KIV-Berechnung im NTS-Protokoll 81
Gleichung 3.10: Cookie-Berechnung im NTS-Protokoll 82
Gleichung 3.11: MAC-Berechnung im NTS-Protokoll 83
Gleichung 3.12: Erzeugung des HMAC 90
Gleichung 3.13: Durch NTS beeinflusster Delay im NTPv4-Protokoll 108
Gleichung 3.14: Durch NTS beeinflusster Offset im NTPv4-Protokoll 108
Gleichung 3.15: Zusätzlich erzeugter Offset durch das NTS-Protokoll 108
Gleichung 6.1: Berechnung des KIV entsprechend der aktuellen NTS-Spezifikation 180
Gleichung 6.2: Alternative Berechnung des KIV aufgrund Spezifikationskonflikte 180

Listing-Verzeichnis

Listing 2.1: Beispiel eines ASN.1-Moduls 11
Listing 3.1: ASN.1-Beschreibung der NTS-Struktur "NTSExtensionFieldContent" 52
Listing 3.2: ASN.1-Beschreibung der NTS-Struktur "ClientAccessData" 64
Listing 3.3: Kennung der NTS-Struktur "ClientAccessData" 64
Listing 3.4: ASN.1-Beschreibung der NTS-Struktur "ServerAccessData" 65
Listing 3.5: Kennung der NTS-Struktur "ServerAccessData" 65
Listing 3.6: ASN.1-Beschreibung des NTS-Elements "NTSAccessKey" 65
Listing 3.7: ASN.1-Beschreibung der NTS-Struktur "ClientAssocData" 66
Listing 3.8: Kennung der NTS-Struktur "ClientAssocData" 66
Listing 3.9: ASN.1-Beschreibung des NTS-Elements "NTSNonce" 66
Listing 3.10: ASN.1-Beschreibung des NTS-Elements "NTSVersion" 67
Listing 3.11: ASN.1-Beschreibung der NTS-Struktur "ServerAssocData" 68
Listing 3.12: Kennung der NTS-Struktur "ServerAssocData" 68
Listing 3.13: ASN.1-Beschreibung der NTS-Struktur "ClientCookieData" 70
Listing 3.14: Kennung der NTS-Struktur "ClientCookieData" 70
Listing 3.15: ASN.1-Beschreibung der NTS-Struktur "ServerCookieData" 71
Listing 3.16: Kennung der NTS-Struktur "ServerCookieData" 72
Listing 3.17: ASN.1-Beschreibung der NTS-Struktur "TimeRequestSecurityData" 72
Listing 3.18: Kennung der NTS-Struktur "TimeRequestSecurityData" 73
Listing 3.19: ASN.1-Beschreibung der NTS-Struktur "TimeResponseSecurityData" 73
Listing 3.20: Kennung der NTS-Struktur "TimeResponseSecurityData" 74
Listing 3.21: ASN.1-Beschreibung der NTS-Struktur "NTSMessageAuthenticationCode" ... 74
Listing 3.22: Kennung der NTS-Struktur "NTSMessageAuthenticationCode" 74
Listing 3.23: ASN.1-Beschreibung des NTS-Elements "NTSVersion" 86
Listing 3.24: Konstellation 1 der NTS-Versionsaushandlung 86
Listing 3.25: Konstellation 2 der NTS-Versionsaushandlung 87
Listing 3.26: Konstellation 3 der NTS-Versionsaushandlung 87
Listing 3.27: Konstellation 4 der NTS-Versionsaushandlung 88
Listing 3.28: Konstellation 5 der NTS-Versionsaushandlung 88
Listing 4.1: XER-kodiertes NTS-Nachrichtenobjekt (ClientAssocData) 119
Listing 4.2: ASN.1-Beschreibung der NTS-Struktur "ClientAssocData" 120
Listing 4.3: Konsolenbefehl zur Konvertierung der ASN.1-Struktur in C-Code 121
Listing 4.4: Von asn1c erzeugte Quellcode-Dateien nach der Konvertierung 121
Listing 4.5: Baumansicht des konvertierten NTS-Nachrichtenobjekts (ClientAssocData) 122
Listing 4.6: Zugriff auf Elemente einer generierten Kernstruktur 123

Listing 4.7: Beispielname für Zertifikatsinhaber und -aussteller 127
Listing 4.8: Aufruf der processNtpMessage()-Methode im NTS-Unicast-Service-Modul ... 137
Listing 4.9: Aufruf der processResponseMessage()-Methode im Client-Unicast-Modul 144
Listing 4.10: Aufruf der createRequestMessage()-Methode im Client-Unicast-Modul 146
Listing 4.11: Aufruf der processNtpMessage()-Methode im Server-Unicast-Modul 147
Listing 4.12: Exemplarischer Aufbau einer Konfigurationsdatei 149
Listing 4.13: Aufbau der globalen NTS-Konfiguration (NtsGlobal.ini) 151
Listing 4.14: Beispielhafte Einträge der NTS-Logdatei 152
Listing 4.15: Aufbau der Konfigurationsdatei für das Client-Unicast-Modul 154
Listing 4.16: Aufbau der Konfigurationsdatei für das Server-Unicast-Modul 157
Listing 4.17: Aktuelle Programmversion der NTS-Implementierung: 159

Tabellenverzeichnis

Tabelle 3.1: Zeitlicher Verlauf der NTS-Spezifikation 23
Tabelle 3.2: Sicherheitsanforderungen an Zeitprotokolle gemäß RFC 7384 33
Tabelle 3.3: Klassifizierung von Kennungen für NTS-Nachrichten 51
Tabelle 3.4: Geschwindigkeitsvergleich unterschiedlicher MAC-Algorithmen 91
Tabelle 3.5: Zertifikatsidentifizierung anhand der Zertifikatserweiterungen 97
Tabelle 3.6: Zertifikatsidentifizierung gemäß RFC 5280 (PKIX Certificate) 98
Tabelle 3.7: Erlaubte Zertifikatserweiterungen für selbstgenerierte Clientzertifikate 99
Tabelle 3.8: Gültige Zertifikat-Flags für selbstgenerierte Clientzertifikate 99
Tabelle 3.9: Erlaubte Zertifikatserweiterungen für ausgestellte Clientzertifikate 99
Tabelle 3.10: Gültige Zertifikat-Flags für ausgestellte Clientzertifikate 100
Tabelle 3.11: Erlaubte Zertifikatserweiterungen für ausgestellte Serverzertifikate 100
Tabelle 3.12: Gültige Zertifikat-Flags für ausgestellte Serverzertifikate 101
Tabelle 3.13: Empfohlene und nicht empfohlene ECC-Algorithmen 102
Tabelle 3.14: Empfohlene und nicht empfohlene Blockchiffren 102
Tabelle 3.15: Empfohlene und nicht empfohlene Betriebsarten für Blockchiffren 102
Tabelle 3.16: Empfohlene und nicht empfohlene Hash-Algorithmen 102
Tabelle 3.17: Empfohlene und nicht empfohlene MAC-Algorithmen 102
Tabelle 3.18: Vorläufige Definition der Fehlercodes im NTS-Protokoll 103
Tabelle 4.1: Testkonstellationen von X.509-Zertifikaten 128
Tabelle 4.2: Öffentliche Klassenmethoden der NtpNtsInterface-Klasse 138
Tabelle 4.3: Private Attribute der NtpNtsInterface-Klasse 138
Tabelle 4.4: Auflistung der Quellcodedateien im NTS-Projekt 163
Tabelle 4.5: Zusammenfassung der implementierten Funktionen im NTS-Projekt 164
Tabelle 4.6: Zusammensetzung des Quellcodes im NTS-Projekt 165
Tabelle 5.1: Minimale Nachrichtengrößen in NTS 168
Tabelle 5.2: Maximale Nachrichtengrößen in NTS (sinnvoller Anwendungsfall) 169
Tabelle 5.3: Durch NTS erzeugte Latenzzeiten 172
Tabelle 6.1: ASN.1-Beschreibung der fiktiven NTS-Struktur "CookieRequestData" 176
Tabelle 6.2: ASN.1-Beschreibung der fiktiven NTS-Struktur "CookieResponseData" 176

1 Einführung

Die Zeit spielt seit jeher eine wichtige Rolle in der Menschheit, mit deren Hilfe sich Abfolgen von Ereignissen beschreiben lassen. Sie ist eine fundamental wichtige Größe in der Physik und gerade heutzutage in vielen Lebensbereichen unersetzlich geworden. Egal ob Navigation, astronomische Längenbestimmung, Überwachung von Abläufen oder digitale Kommunikation, die Zeit stellt immer eine grundlegende Ressource dar. Insbesondere durch die technischen Entwicklungen in den letzten Jahrzehnten und der steigenden Vernetzungsdichte der elektronischen Geräte, ist es wichtig, eine zuverlässige Quelle für die Zeiterfassung zu haben. Atomuhren erfüllen derzeit diese Anforderungen und erreichen dabei sehr hohe Genauigkeiten[1], die es nun global zu verteilen gilt. Diese Aufgabe übernehmen in vielen Bereichen Zeitsynchronisierungsprotokolle, mit denen sich ganze Computernetze zeitlich synchronisieren lassen. Die bekanntesten Vertreter sind hierbei das NTP- und das PTP-Protokoll.

Für die Bereitstellung der gesetzlichen Zeit in Deutschland gemäß § 6 EinhZeitG [1] ist die Physikalisch-Technische Bundesanstalt (PTB) in Braunschweig beauftragt. Diese betreibt vier Atomuhren [2] deren Zeitinformationen über Funk mittels DCF77 [3] und über drei NTP-Zeitserver [4] verbreitet wird. Ferner ist die PTB auch an der Entwicklung von intelligenten Messsystemen (sog. *Smart-Meter-Gateways*[2]) beteiligt, deren Einführung auf Grundlage der europäischen Binnenmarktrichtlinie 2009/72/EG [5] und des nationalen Energiewirtschaftsgesetzes gemäß § 21 EnWG [6] erfolgt. Da diese Geräte eichpflichtige Vorgaben erfüllen müssen [7] [8] , stellt die sichere und unveränderbare Übertragung der Zeit eine wichtige Rolle dar. Ein erfolgreicher Angriff auf die verwendeten Zeitdienste würde somit der Reputation schaden und ist daher zu verhindern. Auch in anderen Bereichen wie z.B. Datenbanksystemen, Finanzmärkten, E-Government oder Militär wächst die Nachfrage nach Sicherheit, was auch authentifizierte Zeitinformationen miteinschließt.

Aus gegebenen Anlass folgten im Jahr 2011 Analysen der bestehenden Sicherheitsmechanismen für Zeitprotokolle, mit besonderem Fokus auf NTP. Das Ergebnis zeigte massive Schwächen der bisherigen Verfahren (s.a. Kapitel 3.1.3), wodurch eine gesicherte Zeitübertragung ausgeschlossen wird. Diese Erkenntnis bildete den Startschuss zur Entwicklung einer neuen Spezifikation, mit deren Hilfe die gesicherte Zeitübertragung zukünftig zu realisieren ist. Federführend hierbei ist die PTB, welche die neue Spezifikation unter dem Namen *Network Time Security* (NTS) [9] über die IETF veröffentlicht. Das Protokoll NTS befindet sich zum derzeitigen Stand noch in der Entwicklungsphase, ist allerdings schon weit fortgeschritten. Dessen Ziel ist die gesicherte Übertragung von Zeitinformationen über bestehende

[1] Die Standardabweichung einer Cäsium-Atomuhr beträgt etwa 1 Sekunde alle 1 Million Jahre [126]. Bei der Cäsium-Fontäne *NIST-F1* konnte diese noch weiter verringert werden (1 Sekunde alle 20 Millionen Jahre) [125].
[2] stellt die zentrale Kommunikationseinheit eines intelligenten Messsystems dar [128]

Protokolle wie NTP oder PTP unter Einhaltung aller wichtigen Kriterien entsprechend dem RFC 7384[3]. Der Aufbau und Ablauf der derzeitigen Zeitprotokolle wird dabei nicht verändert, sondern lediglich um die Sicherheitsfunktionen erweitert (vgl. Abbildung 1.1). Die Kompatibilität zu ungesicherten Verbindungen bleibt nach wie vor erhalten.

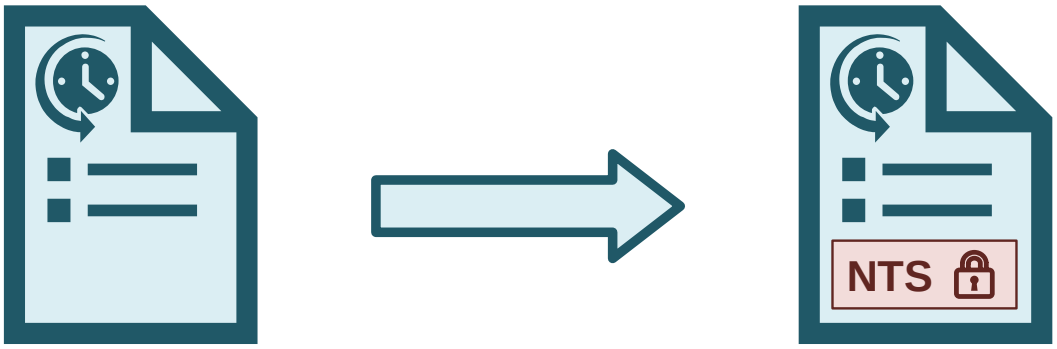


Abbildung 1.1: Symbolische Integration von NTS in Zeitprotokolle

Das Ziel dieser Arbeit ist die Implementierung von NTS für das NTPv4-Protokoll. Dies geschieht auf Basis der öffentlich zugänglichen Spezifikationen und den Informationen, die in Zusammenarbeit mit der PTB ausgetauscht worden sind. Darüber hinaus werden auch die gesetzlichen Richtlinien des BSI (BSI-TR-03116-3 [10]) bei der Umsetzung berücksichtigt. Die Implementierung soll die korrekte Funktionalität der Spezifikation nachweisen und mögliche Definitionslücken aufzeigten, um diese anschließend behandeln zu können. Des Weiteren sei ausdrücklich darauf hingewiesen, dass sich die NTS-Implementierung derzeit nur auf den Unicast-Betriebsmodus konzentriert. Da dieser Modus bereits eine hohe Komplexität und einen großen Umfang besitzt, ist der Broadcast-Betrieb ist kein Bestandteil dieser Arbeit.

Im weiteren Verlauf dieser Dokumentation werden zunächst die Grundlagen vorgestellt, die im engeren Zusammenhang zu der NTS-Spezifikation oder der Implementierung stehen. Das darauffolgende Kapitel 3 thematisiert detailliert das NTS-Protokoll auf Basis der Spezifikation. Der anschließende Abschnitt 4 befasst sich dann mit der Implementierung der Spezifikation und geht dabei auf Konzept und Umsetzungsstrategien ein. Die letzten Kapitel zeigen dazugehörige Testergebnisse auf und gehen abschließend auf weiterführende Entwicklungsschritte ein.

[3] RFC 7384 [71] bietet eine Zusammenfassung von Sicherheitsanforderungen bezogen auf Zeitsynchronisierungsprotokolle (s.a. Kapitel 3.1.4).

2 Allgemeine Grundlagen

Dieses Kapitel erläutert allgemeine Themen, die im direkten oder indirekten Zusammenhang zu NTS stehen und deren grundlegendes Verständnis für die folgenden Kapitel vorausgesetzt wird. Die Beschreibung der einzelnen Gebiete besitzt eher informativen Charakter, wodurch die Darlegung als Übersicht zu betrachten ist. Detaillierte Erläuterungen aus den jeweiligen Themengebieten sind daher aus den angeführten Referenzen zu entnehmen.

2.1 Das OSI-Referenzmodell

Bei dem OSI-Modell handelt es sich in der Netzwerktechnik um ein grundlegendes Referenzmodell, das die Konzipierung von Netzwerkprotokollen erleichtert und somit einen Informationsaustausch heterogener Kommunikationssysteme ermöglicht [11] [12]. Es ist von der ITU und der ISO standarisiert [13] und basiert auf einer Schichtenarchitektur, in der jede Schicht bestimmte Aufgaben verfolgt. Die Umsetzung jeder Schicht kann wiederum durch unterschiedliche Protokolle realisiert werden. Die Kommunikation zwischen den Schichten geschieht über definierte Schnittstellen, wodurch die Protokolle einer Schicht je nach Kommunikationssystem ausgetauscht werden können, ohne die Protokolle der anderen Schichten zu beeinflussen. Dem ist allerdings hinzuzufügen, dass die Protokolle beim Sender und Empfänger identisch sein müssen, um die Nachrichtenübermittlung zu ermöglichen. Das OSI-Referenzmodell wird hierzu in sieben Schichten unterteilt, wobei die untersten vier Schichten das Transportsystem und die oberen drei Schichten das Anwendungssystem bilden (vgl. Abbildung 2.1). Protokolle, die den jeweiligen Schichten zugeordnet werden können, sind beispielhaft in Abbildung 2.2 dargestellt [14, p. 31 ff.].

Die Kommunikation zwischen zwei Teilnehmern kann im OSI-Referenzmodell als *physikalische Verbindung* und als *logische Verbindung* dargestellt werden. Die physikalische Verbindung stellt den realen Datenfluss einer Nachricht dar, die beim Sender in der Schicht 7 beginnt und anschließend alle darunterliegenden Schichten durchläuft. Beim Empfänger angekommen, wandert diese Nachricht beginnend von Schicht 1 aufwärts durch alle Schichten. Damit eine Kommunikation möglich ist, müssen sowohl beim Sender, als auch beim Empfänger alle Schichten durchlaufen werden. Dabei fügt der Sender den Nutzdaten in jeder Schicht einen separaten *Header* und ggf. einen *Trailer* hinzu, wodurch die Verarbeitung in der jeweiligen Schicht des Empfängers ermöglicht wird. Die jeweiligen Inhalte sind hierbei von den verwendeten Protokollen abhängig. Die Kommunikation zwischen zwei Protokollen derselben Schicht, wird als *logische Verbindung* bezeichnet. Diese Protokolle tauschen über den Header der jeweiligen Schicht Informationen aus und agieren entsprechend deren Protokollbeschreibungen. Die darüber- und darunterliegenden Schichtprotokolle, haben hierauf keinen Einfluss.

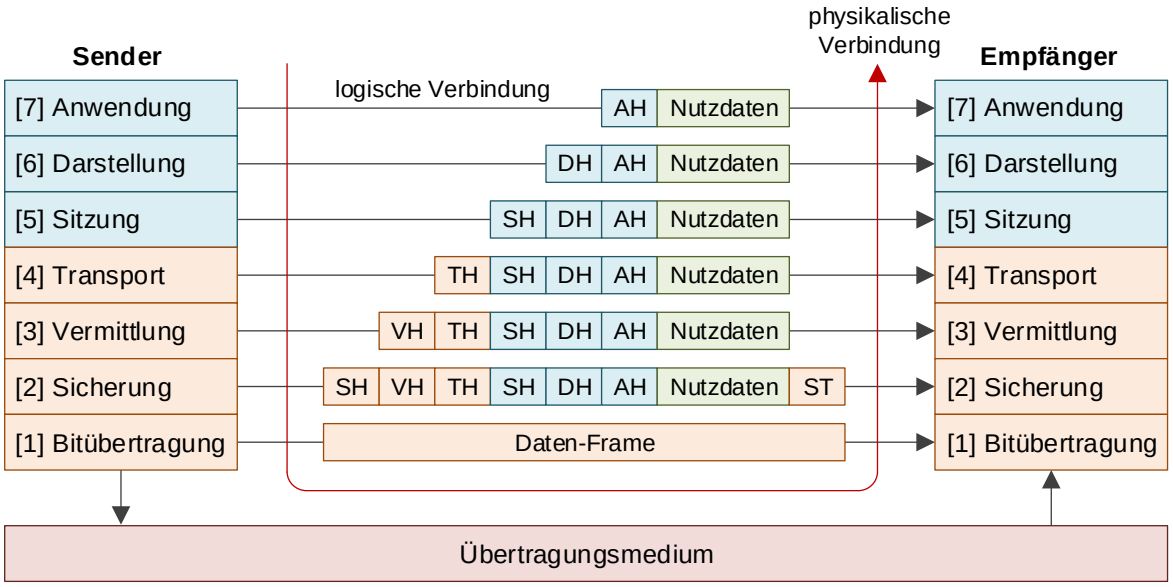


Abbildung 2.1: Kommunikationswege im OSI-Referenzmodell

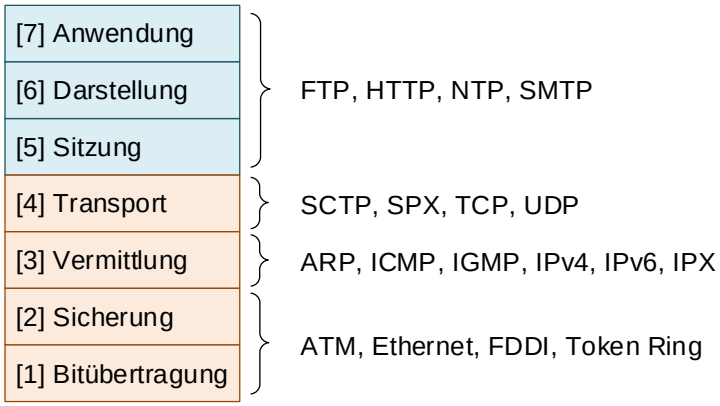


Abbildung 2.2: Protokolle der jeweiligen OSI-Schichten

Die Aufgaben der einzelnen Schichten im OSI-Modell werden im Folgenden kurz vorgestellt und erläutert [14, p. 40 ff.] [15, p. 68 ff.] [16]:

Schicht 1: Bitübertragungsschicht

Die Bitübertragungsschicht (engl. *Physical Layer*) definiert die Schnittstelle zum Übertragungsmedium und ist für den Transfer der Bitströme verantwortlich. Die physikalische Grundlage zur Übertragung der Signale, die Übertragungsrichtung- und Geschwindigkeit können hierbei variieren. Übliche Übertragungsmedien sind Kupferkabel (elektrisch), Glasfaser (Licht) oder Funk (elektromagnetische Wellen).

Schicht 2: Sicherungsschicht

Die Sicherungsschicht (engl. *Data Link Layer*) unterteilt sich MAC-Schicht (*Medium Access Control*) und LLC-Schicht (*Logical Link Control*). Die MAC-Schicht grenzt hierbei an die Bitübertragungsschicht und regelt deren Nutzung. Darüber hinaus verwaltet diese Schicht die physikalische Sende- und Empfangsadresse. Die LLC-Schicht grenzt an die Vermittlungsschicht und dient zur Datenflusssteuerung, Fehlererkennung und -korrektur. Zudem erfolgt hier die Teilung der Datenströme in sogenannte *Frames*.

Schicht 3: Vermittlungsschicht

Die Vermittlungsschicht (engl. *Network Layer*) regelt die Weiterleitung (Routing) der Datenpakete zwischen den Netzwerkknoten, da diese meist nicht direkt an das Ziel übermittelt werden können. Weitere Aufgaben dieser Schicht ist die Datenflusssteuerung und Fragmentierung der Datenpakete.

Schicht 4: Transportschicht

Die Transportschicht (engl. *Transport Layer*) ist die oberste Schicht im Transportsystem und stellt somit die Schnittstelle zum Anwendungssystem dar. In dieser erfolgt die Zuordnung der Daten an Programme bzw. Prozesse durch die Verwendung von Ports.

Schicht 5: Sitzungsschicht

Die Sitzungsschicht (engl. *Session Layer*) ist die unterste Schnittstelle im Anwendungssystem und dient dem Aufbau logischer Verbindungen zwischen dem Sender und dem Empfänger, wodurch die Steuerung von Sitzungen ermöglicht wird.

Schicht 6: Darstellungsschicht

Die Darstellungsschicht (engl. *Presentation Layer*) dient der Konvertierung der übertragenen Daten in ein Format, das neben dem Sender als auch der Empfänger korrekt interpretiert. Hier finden auch Kodierungen, Verschlüsselungen und Datenkompression statt.

Schicht 7: Anwendungsschicht

Die Anwendungsschicht (engl. *Application Layer*) stellt die High-Level-APIs bereit und fungiert damit als Schnittstelle zur eigentlichen Benutzeranwendung. Diese werden als typische Programmfunktionen implementiert und wandeln die vom Benutzer verwendbare Daten in entsprechende Netzwerkdaten um.

2.2 Zeitsynchronisierungsdienste

Die zeitliche Synchronisierung von Computersystemen innerhalb eines Netzwerks ist durch den Einsatz entsprechender Netzwerkprotokolle realisierbar. Dieser Abschnitt stellt hierzu zwei bekannte Vertreter dieser Protokolle vor, deren Integritätsschutz durch die Einbindung des NTS-Protokolls potentiell möglich ist.

2.2.1 Network Time Protocol (NTP)

Eines der ältesten Zeitprotokolle stellt das *Network Time Protocol* (NTP) dar, welches von David L. Mills entwickelt und im Jahr 1985 als RFC 958 [17] veröffentlicht wurde. Den Standard hält die IETF und wurde im Laufe der Zeit immer weiter verbessert, sodass es seit 2010 in der aktuellen Version 4 [18] vorliegt. NTP nutzt paketbasierte Kommunikationsnetze und verwendet das verbindungslose UDP-Transportprotokoll, um die entsprechenden Zeitinformationen zu übertragen. Derzeit kommen am häufigsten die NTP-Versionen 3 und 4 zur Anwendung, die in der Basisfunktionalität komplett miteinander kompatibel sind. Die Version 4 bietet gegenüber der dritten Version weitere Verbesserungen zum Integritätsschutz durch den Einsatz von sogenannten *Erweiterungsfeldern* und dem *Autokey*-Verfahren. Die nähere Beschreibung dieser Eigenschaften und Mechanismen erfolgt im Kapitel 3.1, welches zudem die bereitgestellten Sicherheitsmaßnahmen untersucht [19] [20].

Die Zeitstempel besitzen im NTP-Protokoll eine Länge von 64 Bit, wodurch sich ein Zeitraum von etwa 136 Jahre mit einer Auflösung von 232 Pikosekunden darstellen lässt [21, p. 7]. In Verbindung mit den Paketlaufzeiten im Netzwerk und den daraus resultierenden Fehlern können Genauigkeiten von $10\ ms$ im WAN und $< 1\ ms$ im LAN erreicht werden. Die Zeitserver sind bei NTP hierarchisch aufgebaut, wobei das jeweils verwendete Stratum die Qualität der bezogenen Uhrzeit widerspiegelt. Ein Stratum von 0 entspricht hierbei dem Zeitnormal und wird beispielsweise durch eine Atomuhr repräsentiert. Bei jeder weiteren Verteilungsinstanz (Zeitserver), die sich in der Kette zwischen dem Stratum-1-Zeitserver und dem NTP-Client befindet, erhöht sich dieses Stratum jeweils um eins.

NTP unterstützt verschiedene Betriebsmodi, welche die Art der verwendeten Kommunikationen zwischen den NTP-Teilnehmern (sogenannte *Peers*) bestimmt. Im symmetrischen Modus kann ein NTP-Server sowohl die Zeit verteilen, als auch empfangen. Dadurch garantiert der Zeitserver seine eigene Synchronität durch den Abgleich mit Zeitservern geringerer Strata. Des Weiteren existiert der Broadcast-Modus, in dem der Server periodisch Zeitinformationen in einem Netzwerk aussendet, die bei Bedarf von den angebundenen Clients zur Synchronisierung genutzt werden können. Darüber hinaus kommt auch der Client/Server-Modus zum Einsatz, der mit am häufigsten Verwendung findet und den Unicast-Modus darstellt. In diesem sendet der Client einmalig oder periodisch Zeitanfragen an einem Zeitserver, worauf dieser mit entsprechenden Antworten reagiert und somit die gewünschten Informationen übermittelt [22].

Die Zeitsynchronisation zwischen NTP-Client und Zeitserver funktioniert im Unicast-Betrieb wie folgt. Der Client sendet eine Anfrage in Form eines NTP-Pakets an den Zeitserver. Dieser verarbeitet das Paket, tauscht die IP-Adressen und einige Felder aus und sendet dieses zurück. In dieser Abfolge ergeben sich vier Zeitstempel (vgl. Abbildung 2.3), wovon die ersten drei Zeitstempel ($t_1$bis t_3) im NTP-Paket gespeichert werden. Folgende Zeitstempel werden hierbei erfasst:

- t_1: der Client sendet die Anfragenachricht an den Server
- t_2: der Server hat die Anfragenachricht empfangen
- t_3: der Server sendet die Antwortnachricht an den Client zurück
- t_4: der Client hat die Antwortnachricht empfangen.

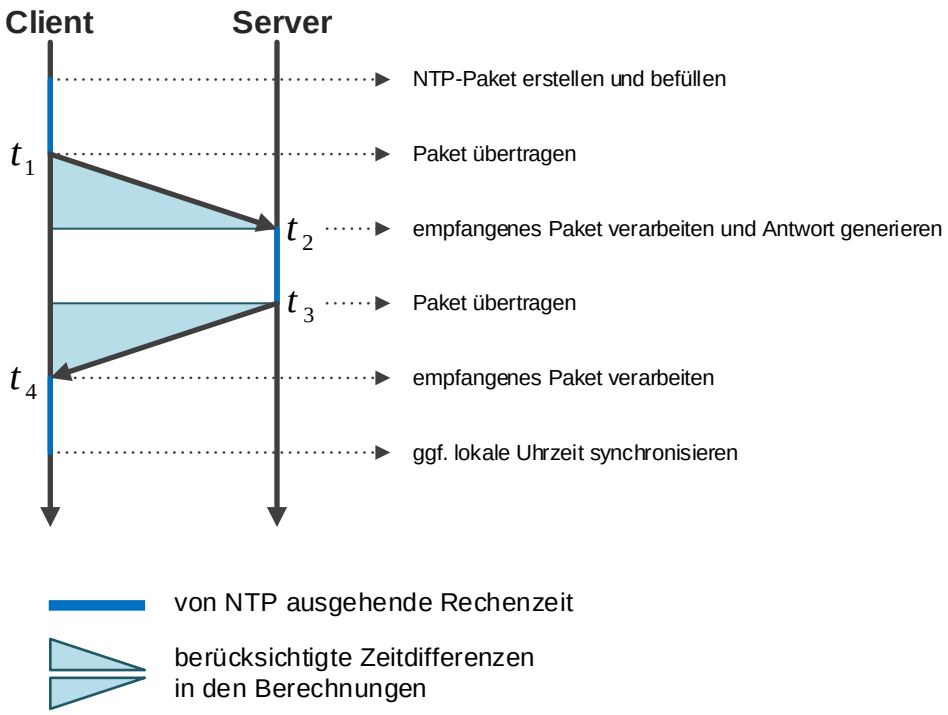


Abbildung 2.3: Zeitstempel im NTPv4-Protokoll

Aus diesen Zeiten kann der NTP-Client nun den Delay und den Offset berechnen. Der Delay stellt die Rundlaufzeit des NTP-Paketes ohne die Rechenzeit dar. Es ist also die reine Zeit, in der sich das Paket im Netzwerk befand. Der Delay berechnet sich gemäß der folgenden Gleichung 2.1 [18]:

$$\delta = (t_4 - t_1) - (t_3 - t_2) \tag{2.1}$$

Im NTP-Protokoll wird davon ausgegangen, dass die Übertragungsdauer der Pakete in beide Richtungen identisch sind. Abweichungen aufgrund von Asymmetrie fließen als Fehler in den Offset mit ein. Dieser berechnet sich entsprechend der Gleichung 2.2:

$$\theta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.2}$$

Der Offset ist die Zeitspanne, um die die Uhrzeiten von Client und Server differieren. Dieser ist die Mittelung der Laufzeitunterschiede der Anfrage- und Antwortnachricht im Netzwerk. Die Rechenzeiten fließen in dieser Berechnung nicht mit ein. Somit ist die Genauigkeit bei der Synchronisierung der lokalen Uhrzeit vom Offset abhängig, der daher möglichst gering ausfallen sollte.

2.2.2 Precision Time Protocol (PTP)

Eine mögliche Alternative zu NTP stellt das *Precision Time Protocol* (PTP) dar, welches in der IEEE 1588 [23] spezifiziert wird. PTP ist für kleinere Netzwerke konzipiert und verfolgt eine höhere Genauigkeit als NTP. Die Umsetzung hierzu erfolgt durch eine hardwarenahe Implementierung, wodurch die Verarbeitungslatenz der jeweiligen PTP-Nachrichten minimiert werden können. Durch dieses Vorgehen können Uhren miteinander synchronisiert werden, deren Zeitabweichung anschließend unter einer Mikrosekunde liegt. Derartige Genauigkeiten sind unter anderem bei Automatisierungsprozessen in der Industrie wichtig. PTP arbeitet mit dezentralen Uhren (sog. *Slaves*), die ihre Zeitinformationen von einem zentralen Zeitserver (dem sog. *Master*) beziehen. Beim Austausch von Nachrichten werden zusätzlich die Paketlaufzeit im Netzwerk (*Delay*) ermittelt, wodurch die Berechnung entsprechender Korrekturwerte möglich ist. Hierbei wird angenommen, dass Paketlaufeiten zwischen Master und Slave auf Hin- und Rückweg identisch sind und sich nur langsam ändern [24] [25]. Zum Schutz der Zeitinformationen vor Manipulation existiert ein entsprechender Mechanismus, der unter dem Namen *Annex K* zu finden ist. Hierbei handelt es sich um ein experimentelles Verfahren, das auf der symmetrischen Kryptografie basiert, aber derzeit noch keine Referenzimplementierung anbietet [26]. Eine kryptografisch gesicherte Zeitsynchronisierung ist im PTP daher noch nicht gegeben.

2.3 Abstract Syntax Notation One (ASN.1)

Für die Übertragung von Informationen zwischen Kommunikationsteilnehmern kommt im NTS-Protokoll die *Abstract Syntax Notation One* (ASN.1) zum Einsatz. Hierbei handelt es sich um eine Beschreibungssprache, in der Datenstrukturen auf einer hohen Abstraktionsebene definiert werden. ASN.1 bietet die Möglichkeit, Informationen nicht nur zwischen homogenen Systemen, sondern auch zwischen heterogenen Systemen auszutauschen. Somit ist sie sprach- und plattformunabhängig und eignet sich zur Spezifizierung von Kommunikationsprotokollen. Die Standardisierung hierfür erfolgte erstmals 1984 durch ISO[4], IEC[5] und der ITU-T[6] und wurde 1988 als eigenständiger Standard in X.680 [27] definiert.

[4] ISO: International Organization for Standardization
[5] IEC: International Electrotechnical Commission
[6] ITU-T: International Telecommunication Union-Telecommunication Standardization Sector

2.3.1 Kommunikation zwischen heterogenen Systemen

Das grundlegende Problem bei einem Datenaustausch zwischen zwei oder mehreren Kommunikationspartnern ist die Tatsache, dass diese heterogen sein können. Das bedeutet, dass die verwendete Maschinenarchitektur oder die verwendete Programmiersprache zwischen den Teilnehmern unterschiedlich sind (vgl. Abbildung 2.4) [28, p. 8 ff.].

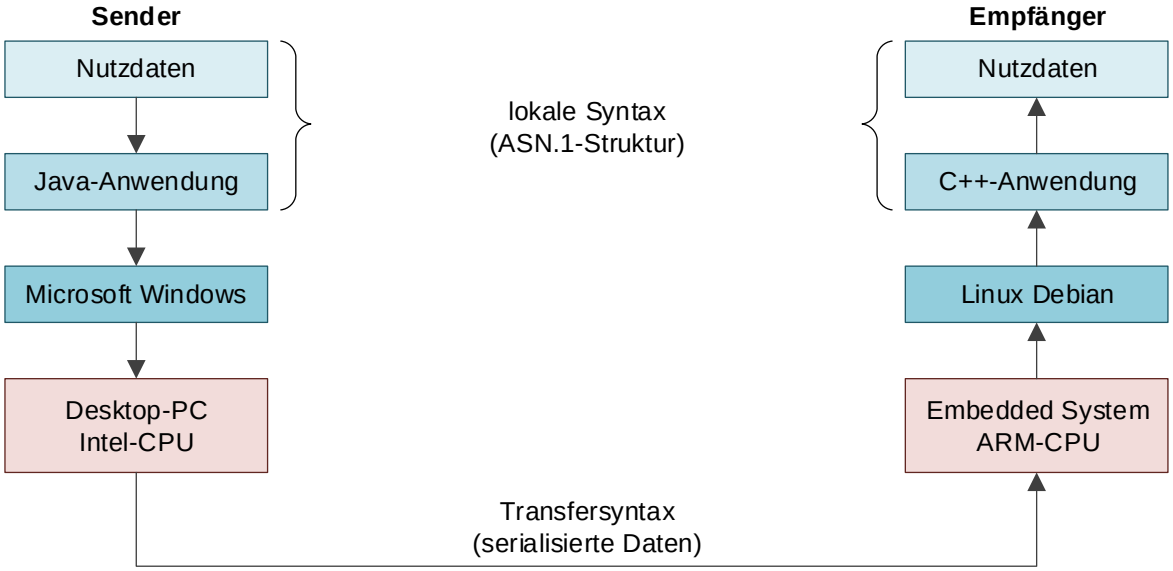


Abbildung 2.4: Kommunikation heterogener Systeme

In Folge dessen ist die Fehlinterpretation der übertragenen Daten sehr wahrscheinlich, da unterschiedliche Definitionen zur Verarbeitung von Daten auf Hard- und Software-Ebene angewendet werden. Ein Beispiel hierfür die Festlegung der Bit-Größe eines Integers (Ganzzahl). Die Standardgröße richtet sich häufig nach der verwendeten Hardware-Plattform und liegt meist im Bereich von 16 Bit (z.B. bei einfachen Mikrocontrollern) bis 64 Bit bei heutigen Desktop-Computern. Darüber hinaus kann auch die Byte-Reihenfolge (*BigEndian*/*LittleEndian* [28, p. 9] [29, p. 6]) unterschiedlich sein, wodurch zusätzliche Schwierigkeiten bei der korrekten Interpretation der Daten entstehen. ASN.1 setzt hier an und fungiert gewissermaßen als Dolmetscher zwischen unterschiedlichen Systemen. Es verwendet eine eigene Syntax, in der komplexe Datenstrukturen beschrieben werden und deren Serialisierung im Anschluss durch entsprechende Kodierungsverfahren möglich ist. Nach der Übertragung der serialisierten Daten können diese unter Verwendung des Dekodierers in ein Format überführt werden, die eine korrekte Interpretation der Daten vom Zielsystem garantiert. Die Portierung der ASN.1-Strukturen in das benötigte Format, die durch die verwendete Programmiersprache definiert wird, ermöglicht somit den Austausch und die korrekte Interpretation der übertragenen Informationen [28].

2.3.2 Die Syntax in ASN.1

Für die Beschreibung von Datenstrukturen liefert ASN.1 ein ausreichendes Arsenal an Schlüsselwörtern, mit denen sich Strukturen und Module wie im folgenden Beispiel definieren lassen:

Listing 2.1: Beispiel eines ASN.1-Moduls

```
ModulBeispiel {1 2 34 5 67 1} DEFINITIONS ::=
BEGIN
    Verbindung ::= SEQUENCE
    {
        kennung OBJECT_IDENTIFIER,
        version INTEGER,
        modus Modus,
        rohdaten OCTET_STRING
    }

    Modus ::= ENUMERATED
    {
        Unicast (0),
        Broadcast (1)
    }
END
```

Ein Modul stellt eine ASN.1-Einheit dar, die eine Zusammenstellung von Strukturen, Typdefinitionen und ASN.1- Basistypen beinhaltet. Die enthaltenen Elemente können aufeinander referenzieren, wodurch die Beschreibung komplexer Datenstrukturen in beliebiger Tiefe möglich ist. Die Basistypen werden hierbei von ASN.1 definiert und repräsentieren festgelegte Inhalte oder Datentypen. Ein *OCTET_STRING* enthält somit Binärdaten in beliebiger Länge, während ein *INTEGER* eine Ganzzahl (unbegrenzter Größe[7]) darstellt. Neben diesen beiden gibt es noch eine Vielzahl an weiterer Basistypen, mit denen sich unter anderem Arrays, Bitströme, darstellbare Zeichenketten, Uhrzeiten und Strukturen beschreiben lassen [30, p. 3] [31]. Durch die Kombination dieser Basistypen und durch den Einsatz spezieller *Flags* (z.B. Markierung optionaler Felder) sind auch eigene Typdefinitionen möglich.

Um die übertragenen Strukturen oder Module korrekt parsen zu können, muss der Empfänger natürlich die beschriebene Struktur kennen und identifizieren können. Hierfür kommen sogenannte *Object Identifier* (OID) zum Einsatz, die auf den Inhalt eines Moduls oder einer Struktur schließen lassen. Der Aufbau der Kennungen ist strikt festgelegt und verfolgt eine Struktur, die im RFC 3061 [32] näher betrachtet wird. So repräsentiert beispielsweise die OID „1.2.840.113549.1.1.5“ gemäß der angewandten Punktnotation den Signaturalgorithmus *sha1WithRSAEncryption* [33]. Die Identifizierung und Suche bestehender OIDs kann in den entsprechenden Registern bzw. Repositorien [34] erfolgen.

[7] Die benötigte Byte-Länge wird vom Kodierungsverfahren festgelegt und wächst je nach dargestellter Zahl automatisch.

2.3.3 Kodierungsverfahren

Um die ASN.1-Strukturen – die sogenannte lokale Syntax – auf ein Zielsystem übertragen zu können, müssen diese Daten in die Transfersyntax überführt werden. Diese repräsentiert die spezifischen Daten als serialisierten Datenstrom. Die Umwandlungen der Informationen zwischen lokaler Syntax und Transfersyntax erfolgt über eines von mehreren möglichen Kodierungsverfahren. In diesem Abschnitt werden hierzu die Kodierungsverfahren vorgestellt, die im Bereich NTS zur Anwendung kommen.

2.3.3.1 Basic Encoding Rules (BER)

Ein grundlegendes Kodierungsverfahren stellen die *Basic Encoding Rules* dar, mit denen die Umwandlung der ASN.1-Strukturen in ein *Type-Length-Value*-Format (TLV) möglich ist. Dieses Format bietet eine einfache Methode zum Parsen der Daten und erlaubt zudem die Verarbeitung von unvollständig empfangenen Datenströmen. Die entsprechende Standardisierung ist in X.690-0207 [35] festgesetzt und beschreibt die geltenden Verarbeitungsregeln der ASN.1-Datentypen. Die BER-Kodierung stellt das Standardverfahren dar und sollte von jeder Anwendung, die ASN.1 verwendet, unterstützt werden. Darüber hinaus ist ein BER-Dekodierer auch immer in der Lage DER-kodierte Datenströme zu verarbeiten.

2.3.3.2 Distinguished Encoding Rules (DER)

Die *Distinguished Encoding Rules* sind ebenfalls in der X.690-0207 [35] spezifiziert und stellen eine Unterart der BER-Kodierung dar. Im Gegensatz zur BER-Kodierung ist ein DER-kodierter ASN.1-Datenstrom immer gleich aufgebaut, welches durch strikte Kodierungsregeln realisiert wird. Ein Beispiel hierfür ist ein Boolescher Wert, der sich in einer geeigneten ASN.1-Struktur befindet und den Wert *true* zugewiesen bekommt. Bei einer BER- bzw. DER-Kodierung wird dieser Eintrag als ein Byte dargestellt, wodurch sich ein Wertebereich von 0 bis 255 ergibt. In der BER-Kodierung gilt der Eintrag als *true*, wenn der Wert größer null ist. Somit können 255 unterschiedliche Kodierungen den Wert *true* darstellen. Im Gegensatz dazu repräsentiert in der DER-Kodierung der exakte Wert 255 *true* und 0 *false*. Die identische Serialisierung von Daten, wie es bei der DER-Kodierung ist, spielt besonders in der Kryptografie eine wichtige Rolle, da die erfolgreiche Überprüfung einer MAC oder einer digitalen Signatur immer gleiche Eingabewerte erfordert. Typische Einsatzgebiete für dieses Kodierungsverfahren sind unter anderem Zertifikate, Schlüssel und CMS-Strukturen.

2.3.3.3 XML Encoding Rules (XER)

Die XER-Kodierung ist ein weiteres Verfahren das in X.693-0112 [36] definiert ist und keine Serialisierung der ASN.1-Strukturen anstrebt. Hierbei werden die ASN.1-Dateien lediglich in eine XML-Syntax umgeformt, wodurch die Kommunikation mit Anwendungen möglich ist, die kein ASN.1 nutzen. Darüber ermöglich diese Formatierung auch Diagnosemöglichkeiten, die unter anderem bei der Implementierung von NTS Anwendung finden (s.a. Kapitel 4.2.1.1).

2.4 Kryptografie

Die Kryptografie [37] ist ein Bestandteil der Kryptologie [38] und beschäftigt sich mit dem verschlüsselten Schreiben von Informationen, um deren Manipulation und unbefugtes Auslesen durch Dritte zu unterbinden. Dieser Abschnitt erläutert dazu die grundlegenden Verschlüsselungsverfahren, die als Bestandteil von NTS in die Implementierung einfließen. Darüber hinaus werden auch Hash- und MAC-Funktionen, digitale Signaturen und Zertifizierungsstrukturen thematisiert, die im näheren Zusammenhang zueinanderstehen.

2.4.1 Verschlüsselungsverfahren

Verschlüsselungen ermöglichen unter Verwendung eines geheimen Schlüssels die Umwandlung von Klartextinformationen in einem Geheimtext (vgl. Abbildung 2.5). Als Grundlage hierfür dienen üblicherweise Algorithmen, die ein schweres mathematisches Problem beinhalten, wodurch die Wiederherstellung der Klartextinformationen ohne den korrekten Schlüssel gar nicht oder nur unter sehr großen Aufwand möglich ist. Verschlüsselungsverfahren bieten ein breites Anwendungsspektrum, die nicht nur die Vertraulichkeit von gespeicherten oder übertragenen Informationen sichern. Sie dienen auch dem Integritätsschutz von Nachrichten bei einer MAC-Berechnung oder ermöglichen die Prüfung der Authentizität von übermittelten Nachrichten durch eine digitale Signatur. Die Unterteilung der Verschlüsselungsverfahren erfolgt derzeit in symmetrische und asymmetrische Verfahren, die im Folgenden vorgestellt und beschrieben werden.

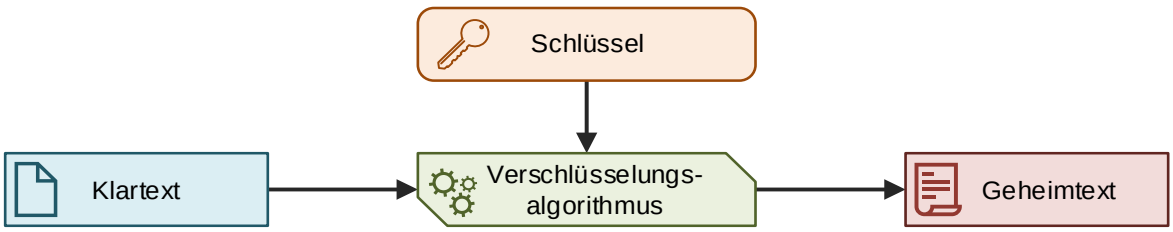


Abbildung 2.5: Grundprinzip der Verschlüsselung

2.4.1.1 Symmetrische Verschlüsselung

Beim symmetrischen Verschlüsselungsverfahren besitzen beide Kommunikationspartner denselben Schlüssel, mit dem die zu schützenden Informationen sowohl verschlüsselt, als auch entschlüsselt werden (vgl. Abbildung 2.6). Als Ansatz kommen hierfür Transpositions-, Substitutions- oder kombinierte Verfahren in Frage [39]. Beim Transpositionsverfahren findet eine Permutation der Nachrichten statt. Das bedeutet, es kommt nur zu einer Vertauschung von Zeichen, Zahlen oder Buchstaben innerhalb des Klartextes, ohne diese zu verändern. Bei der Substitution werden hingegen Nachrichtenteile ersetzt, was einzelne Zeichen oder auch ganze Wörter bzw. Phrasen einbezieht. Der Zusammenschluss beider Ansätze ergibt ein kombiniertes Verfahren, das in der heutigen Informationstechnik oft angewandt wird. Zwei bekannte Vertreter sind der *Data Encryption Standard* (DES) [40] und der *Advanced Encryption Standard* (AES) [41] [42]. Diese unterstützen zudem unterschiedliche Modi für Blockverschlüsselungen, wie *Electronic Code Book* (ECB) oder *Cipher Block Chaining* (CBC)[8]. Die Blockverschlüsselung mittels ECB erlaubt die einzelne Verarbeitung von Datensegmenten. Dies verhindert zwar aufgrund der einzelnen Verarbeitung der Datensegmente eine mögliche Fehlerfortpflanzung, bietet jedoch auch Angriffsmöglichkeiten, da identische Datensegmente immer denselben verschlüsselten Inhalt generieren. Im Gegensatz dazu sind die Datenblöcke im CBC voneinander abhängig. Der verschlüsselte Inhalt eines Datensegments wird hierbei als Eingabewert für die Verarbeitung des folgenden Datensegments verwendet. Die einzelne Verschlüsselung der Datensegmente ist zwar nicht mehr möglich, verhindert aber den genannten Angriffspunkt. Neben den beiden eingangs erwähnten Betriebsmodi für Blockverschlüsselungen, gibt es zudem noch viele weitere [14, p. 330].

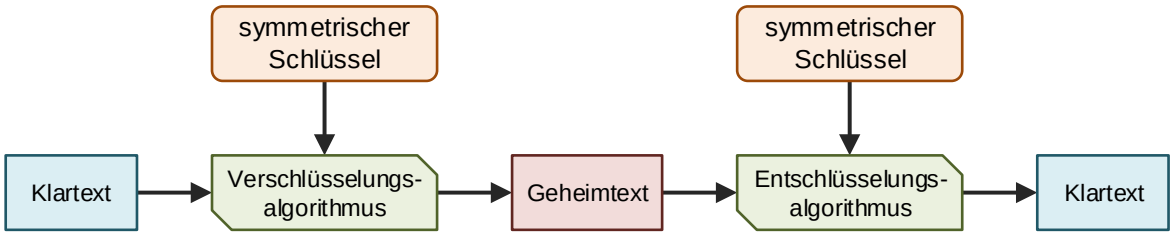


Abbildung 2.6: Prinzip der symmetrischen Ver- und Entschlüsselung

Die symmetrische Verschlüsselung bietet gegenüber der asymmetrischen Verschlüsselung einen großen Geschwindigkeitsvorteil, weshalb diese sich zur Verschlüsselung großer Datenmengen eignet. Der Nachteil ist jedoch, dass beide Kommunikationspartner denselben Schlüssel besitzen müssen. Dieser ist vorab über einen gesicherten Kanal auszutauschen, um ein Auslesen durch Dritte zu verhindern.

[8] Definiert in *ISO/IEC 10116;* (siehe auch [105])

2.4.1.2 Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung besitzen beide Kommunikationsteilnehmer jeweils einen öffentlichen und einen privaten Schlüssel. Dies unterscheidet sich von der symmetrischen Verschlüsselung, welche einen gemeinsamen Schlüssel verwendet. Der öffentliche Schlüssel kann über einen unsicheren Kanal getauscht werden und erlaubt nur die Verschlüsselung von Daten. Die Entschlüsselung der Informationen ist anschließend nur noch über den privaten Schlüssel möglich, der bei den jeweiligen Kommunikationsteilnehmern verbleibt und geheim gehalten wird (vgl. Abbildung 2.7). Eine Entschlüsselung der Daten über den öffentlichen Schlüssel ist nicht möglich. Dies bietet gegenüber der symmetrischen Verschlüsselung einen großen Vorteil. Ein Angreifer, der während einer Kommunikation den öffentlichen Schlüssel abfängt, ist nicht in der Lage die verschlüsselten Inhalte zu lesen [14, p. 334].

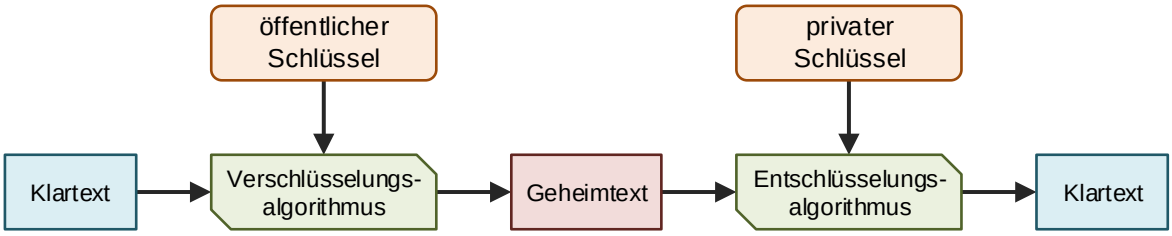


Abbildung 2.7: Prinzip der asymmetrischen Ver- und Entschlüsselung

Die asymmetrische Verschlüsselung – auch *Public-Key-Verschlüsselung* genannt – beruht auf Einwegfunktionen, bei denen die Berechnung in eine Richtung sehr einfach, eine Umkehrung jedoch aufwendig ist [43]. Dies kann beispielsweise mit Modulo-Rechenarten umgesetzt werden. Die aktuellen Verfahren nutzen hierbei schwere mathematische Probleme, wie die Zerlegung einer großen Zahl in ihre beiden Primfaktoren oder die Nutzung des diskreten Logarithmus. Bekannte Vertreter der asymmetrischen Verschlüsselung sind RSA (*Rivest, Shamir und Adleman*) und ECC (*Elliptic Curve Cryptography*). Der Nachteil dieser Verschlüsselungsvariante ist die vergleichsweise langsame Ver- und Entschlüsselung von Daten (bezogen auf die symmetrische Verschlüsselung) sowie die größeren Schlüssellängen.

2.4.1.3 Hybride Verschlüsselung

Die hybride Verschlüsselung kombiniert symmetrische und asymmetrische Verschlüsslung miteinander und fusioniert deren Vorteile. Hierbei generiert der Sender einen zufälligen symmetrischen Schlüssel, den er mit dem öffentlichen Schlüssel des Empfängers asymmetrisch verschlüsselt und an diesen überträgt. Der Empfänger kann nun durch Verwendung seines privaten Schlüssels und dem entsprechenden asymmetrischen Verschlüsselungsverfahren den symmetrischen Schlüssel entschlüsseln. Von nun an können Sender und Emp-

fänger durch Verwendung von symmetrischen Verschlüsselungsverfahren Daten austauschen. Die hybride Verschlüsselung bietet somit den Geschwindigkeitsvorteil der symmetrischen Verschlüsselung und erlaubt den sicheren Austausch des Schlüssels durch Verwendung der asymmetrischen Verschlüsselung [44].

2.4.2 Digitale Fingerabdrücke und Signaturen

Die Generierung von digitalen Fingerabdrücken oder Signaturen kommt in der Kryptografie immer dann zur Anwendung, wenn Datenmengen zu vergleichen sind. Somit können durch unterschiedliche Verfahren Integritätsprüfungen vollzogen werden. Dieser Abschnitt stellt dazu gängige Methoden dar und zeigt deren Unterschiede und Einsatzgebiete auf.

2.4.2.1 Kryptografische Hash-Funktionen

Hash-Funktionen sind mathematische Algorithmen, die eine Abbildung von Daten mit beliebiger Länge in eine Zeichenkette fester Länge ermöglichen (vgl. Abbildung 2.8). Diese erzeugen somit einen digitalen Fingerabdruck, der sich leicht vergleichen lässt und zudem eine Integritätsprüfung von Daten ermöglicht. Hash-Algorithmen sind so konzipiert, dass diese bei kleinsten Änderungen der Eingabewerte völlig andere Hash-Werte generieren. Jedoch erzeugen diese bei gleichen Eingabewerten immer dieselbe Ausgabe. Die Länge des erzeugten Hash-Wertes hängt hierbei ausschließlich vom verwendeten Algorithmus ab und liegt in der Regel bei 128 bis 512 Bit. Hash-Algorithmen sind sogenannte *Einwegfunktionen*, die eine Konstruktion der ursprünglichen Daten auf Basis des Hash-Wertes unmöglich machen[9]. Somit unterscheiden sie sich von Verschlüsselungsalgorithmen, bei denen diese Bedingung erfüllt sein muss. Eine wichtige Anforderung zur kryptografischen Sicherheit stellt bei Hash-Algorithmen die *Kollisionsresistenz* dar. Eine Kollision bezeichnet das Auftreten gleicher Hash-Werte für unterschiedliche Eingabewerte. Dies ist problematisch, da eine mögliche Verfälschung der Eingabewerte auf Basis des Hash-Wertes nicht erkennbar ist. In der Praxis ist eine Kollisionsfreiheit nicht realisierbar, da die Ausgabewerte begrenzt und die Eingabewerte in Länge und Inhalt beliebig sind. Dennoch ist die Wahrscheinlichkeit für das Auftreten von Kollisionen in modernen Hash-Algorithmen so gering, dass diese vernachlässigt werden können [42].

[9] Eine theoretische Möglichkeit bietet jedoch der *Urbild*-Angriff. Hierbei werden so lange unterschiedliche Eingabewerte generiert werden, bis der gewünschte Hash-Wert gefunden wurde [128].

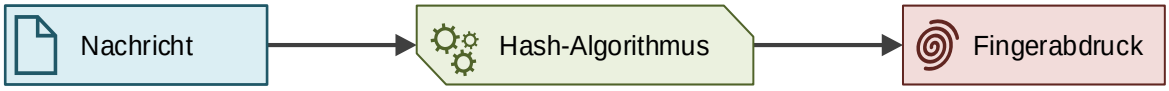


Abbildung 2.8: Prinzip zur Erzeugung eines digitalen Fingerabdrucks (Hash-Wert)

2.4.2.2 MAC-Funktionen

MAC-Funktionen sind vom Verwendungszweck her mit Hash-Algorithmen vergleichbar. Auch diese generieren anhand der Eingabedaten einen eindeutigen Fingerabdruck. Die Ausgabe besitzt eine feste Länge, die nur vom verwendeten MAC-Algorithmus abhängt und liegt meist bei 128 bis 512 Bit. Im Gegensatz zu einer einfachen Hash-Funktion fließt allerdings noch ein zweiter Eingabewert mit in die Berechnung ein (vgl. Abbildung 2.9). Hierbei handelt es sich um einen symmetrischen Schlüssel, den nur die entsprechenden Kommunikationspartner kennen und der geheim gehalten wird. Dadurch garantiert der MAC neben der Integrität auch die Authentizität der Daten. Darüber hinaus ist ein *Urbild*-Angriff nicht mehr möglich, da ein Angreifer dazu im Besitz des symmetrischen Schlüssels sein muss.

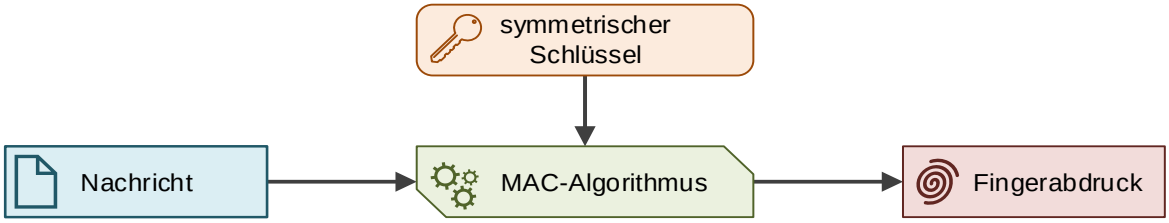


Abbildung 2.9: Prinzip zur Erzeugung eines MAC

Die Arbeitsweise von MAC-Algorithmen kann unterschiedlich realisiert werden. Ein häufig verwendeter Algorithmus ist der HMAC [45], der auf Basis einer beliebigen Hash-Funktion basiert. Daneben existiert auch der CMAC-Algorithmus [46], der sich die symmetrische Verschlüsselung (z.B. mit AES) zunutze macht und bei hardwareunterstützter AES-Beschleunigung einen großen Geschwindigkeitsvorteil gegenüber anderen Methoden erzielt (s.a. Kapitel 3.6.3). Der generierte MAC dient als Prüfteil einer Nachricht und wird normalerweise nach seiner Erzeugung an diese angehängt. Eine Geheimhaltung der Nachricht, wie es bei einer Verschlüsselung der Fall ist, erfolgt also nicht.

2.4.2.3 Digitale Signatur

Die digitale Signatur ist eine weitere Möglichkeit, die Integrität der übertragenen Daten zu gewährleisten. Der Verwendungszweck geht allerdings über den Nutzen eines einfachen MAC hinaus. Bedingt durch die Generierung der digitalen Signatur ist die Nachweisbarkeit

des Absenders möglich. Dies unterscheidet sich von einem MAC, bei dem jeder Teilnehmer, der im Besitz des symmetrischen Schlüssels ist, den identischen MAC erzeugen kann. Daher ist bei einer MAC-Generierung die Nachweisbarkeit des Absenders nicht möglich. Die Erzeugung einer digitalen Signatur geschieht über einen entsprechenden Signaturalgorithmus (vgl. Abbildung 2.10), der sich wiederum in einen Hash- und asymmetrischen Verschlüsselungsalgorithmus aufteilen lässt (vgl. Abbildung 2.11). Ein Beispiel für einen Signaturalgorithmus wäre *sha1WithRSAEncryption*, der sich aus den genannten Komponenten zusammensetzt.

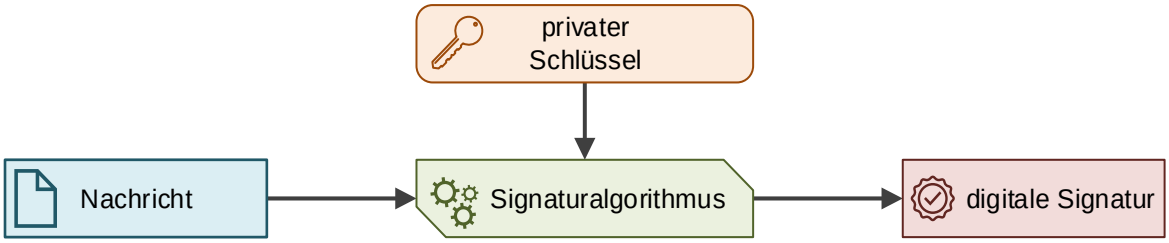


Abbildung 2.10: Prinzip zur Erzeugung einer digitalen Signatur

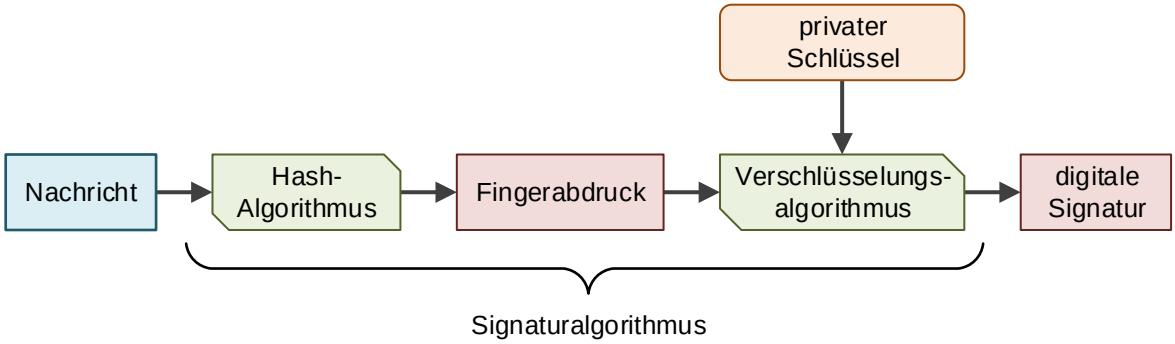


Abbildung 2.11: Zerlegung des Signaturalgorithmus in seine Bestandteile

Bei Generierung der Signatur wird anhand der Eingabedaten (die Nachricht) und unter Verwendung eines Hash-Algorithmus, der jeweilige Fingerabdruck erzeugt. Im Anschluss erfolgt zusammen mit dem privaten Schlüssel und dem dazugehörigen Algorithmus die asymmetrische Verschlüsselung des Fingerabdrucks. Das resultierende Ergebnis ist die digitale Signatur, die nun analog zur MAC an die Nachricht angehängt und mit dieser an den Empfänger versendet wird. Der Empfänger kann nun über den öffentlichen Schlüssel des Absenders die Signatur entschlüsseln und somit den enthaltenen Hash-Wert extrahieren. Anschlie-

ßend berechnet er ebenfalls den Hash-Wert der empfangenen Nachricht (ohne der angehängten Signatur) und vergleicht das Ergebnis mit dem entschlüsselten Hash-Wert. Sind diese identisch, so liefert es dem Empfänger den Beweis, dass die Nachricht unverfälscht ist und vom erwarteten Absender stammt [42].

2.4.3 Zertifikate und Zertifizierungsstruktur

Zertifikate sind in der Informationstechnik das elektronische Äquivalent zum Personalausweis. Diese bestätigen die Authentizität des Absenders und ermöglichen darüber hinaus den Austausch des öffentlichen Schlüssels. Die Integrität des Zertifikates wird zudem durch eine Signatur geschützt, die von einer externen Zertifizierungsinstanz erstellt wird. Ein derzeit weit verbreiteter Standard ist das X.509-Zertifikat [47], welches in der aktuellen Version 3 vorliegt. Dieses besitzt in der Regel folgende Komponenten:

- die verwendete Zertifikatsversion
- die Seriennummer des Zertifikats
- den vollständigen Namen des Zertifikatsinhabers
- den vollständigen Namen des Zertifikatsausstellers
- die Gültigkeitsperiode
- den öffentlichen Schlüssel des Zertifikatsinhabers
- die digitale Signatur
- ggf. diverse Zertifikatserweiterungen

Eine besondere Herausforderung der Zertifikate stellt deren Authentifikation dar. So ist es prinzipiell möglich, dass sich ein Angreifer als vermeintlicher Kommunikationsteilnehmer ausgibt und ein verfälschtes Zertifikat überträgt. Um diesem entgegenzuwirken, kommt die *Public-Key-Infrastruktur* (PKI) zum Einsatz [48]. Diese Zertifizierungsstruktur beinhaltet Zertifikatsaussteller (*certification authority* (CA)) und Registrierungsstellen (*registration authority* (RA)), bei denen Zertifikate angefragt und überprüft werden können. Die ausgestellten Zertifikate werden hierbei von den Zertifizierungsinstanzen digital signiert, um die Überprüfung zu ermöglichen. Dabei ist es möglich, dass diese ausgestellten Zertifikate wiederum andere Zertifikate signieren, wodurch sich eine Zertifizierungshierarchie bildet. Sogenannte Root-CAs stellen hierbei die höchsten Zertifizierungsinstanzen der Hierarchie dar. Diese bieten öffentlich zugängliche Wurzelzertifikate an, die einen vertrauenswürdigen Anker repräsentieren. Derartige Zertifikate sind selbstsigniert und ermöglichen die Überprüfung des ausgetauschten Zertifikats durch den Aufbau eines *Zertifizierungspfades*. Hierbei kann die Signatur des ausgestellten Zertifikats über mehrere Zertifizierungsinstanzen bis zur Root-CA zurückverfolgt und verifiziert werden.

2.5 Cryptographic Message Syntax (CMS)

Die *Cryptographic Message Syntax* (CMS) ist eine Zusammenstellung von Strukturen, die in ASN.1 beschrieben sind und unterschiedliche kryptografische Aufgaben erfüllen. Die aktuelle Spezifikation aus dem Jahr 2009 ist im RFC 5652 [49] beschrieben und leitet sich aus den älteren *Public-Key Cryptography Standards #7* (PKCS #7) [50] ab. Je nach Verwendungszweck stellt CMS unterschiedliche Strukturen bereit, die zudem ineinander verschachtelt werden können. An dieser Stelle werden dazu jene Strukturen vorgestellt, die auch in der NTS-Spezifikation Anwendung finden.

CMS-Struktur: *Data*

Die *Data*-Struktur repräsentiert die binären Nutzdaten in Form eines ASN.1 *OCTET STRINGS*, wodurch es genau betrachtet keine echte Struktur darstellt.

CMS-Struktur: *ContentInfo*

ContentInfo stellt die Grundstruktur in CMS dar und beinhaltet eine andere CMS-Struktur, sowie deren Kennung in Form einer Objekt ID (OID). Diese Struktur ermöglicht die Identifizierung und folglich das korrekte Parsen einer übertragenen CMS-Struktur.

CMS-Struktur: *SignedData*

SignedData ist eine komplexe CMS-Struktur, die eine Signierung der enthaltenen Daten ermöglicht. Diese nutzt X.509-Zertifikate und unterstützt mehrere unabhängige Unterzeichner mit unterschiedlichen Signaturalgorithmen.

CMS-Struktur: *EnvelopedData*

EnvelopedData wird zur Verschlüsselung von Daten verwendet und kann darüber hinaus mit der *SignedData*-Struktur kombiniert werden. Die Struktur unterstützt unterschiedliche Methoden für den Schlüsselaustausch und nutzt das hybride Verschlüsselungsverfahren, um Nutzdaten und Schlüssel zu verschlüsseln. Auch diese Struktur nutzt X.509-Zertifikate, um die entsprechenden Schlüssel zu beziehen und den Empfänger zu identifizieren.

3 Das Network-Time-Security-Protokoll

In diesem Kapitel werden die verschiedenen Aspekte des *Network-Time-Security*-Protokolls (NTS) dargelegt. Hierbei handelt es sich um ein Sicherheitsprotokoll, das in bestehende Zeitsynchronisierungsprotokolle wie NTP oder PTP eingebettet wird und diese um Sicherheitsfunktionen erweitert. Das NTS-Protokoll soll zukünftig bestehende Sicherheitsmechanismen ablösen, da diese nicht mehr den aktuellen Sicherheitsanforderungen entsprechen. Der Fokus dieses Kapitels bezieht sich ausschließlich auf den Unicast-Betrieb des NTS-Protokolls und dessen Anwendung in NTPv4. Dieser setzt auf dem entsprechenden Betriebsmodus des Zeitsynchronisierungsprotokolls auf und wird im Grundlagenkapitel 2.2.1 näher beschrieben.

Die Untergliederung dieses Kapitels erfolgt in sechs Abschnitten, die den Fokus auf verschiedene Aspekte des NTS-Protokolls lenken. Diese Untergliederung beginnt zunächst mit einer Übersicht zu NTS, welche die Motivationen und die Entwicklung der neuen Spezifikation näherbringt. Im Anschluss erfolgt eine kurze Funktionsbeschreibung des NTS-Protokolls in Bezug auf den Unicast-Betrieb (siehe Kapitel 3.2). Der nachfolgende Abschnitt beschreibt für diesen Betriebsmodus die jeweiligen Nachrichten, die während der Kommunikation zum Einsatz kommen. Die Zerlegung dieser Nachrichten in deren Bestandteilen und die Durchleuchtung der Kommunikationsstruktur thematisiert Abschnitt 3.4. Dies bildet die Grundlage für das Verständnis des folgenden Unterkapitels, in dem eine detaillierte Beschreibung des Kommunikationsablaufs erfolgt und alle zuvor behandelten Aspekte zusammengefügt werden. Themen, die sich auf NTS beziehen und in jeweiligen Abschnitten unzureichend untersucht wurden, behandelt das Unterkapitel 3.6 gesondert. Dies sind beispielsweise Empfehlungen, Sonderfälle oder zusätzliche Informationen zu einem Themengebiet.

Wichtige Informationen vorab

An dieser Stelle wird noch einmal nachdrücklich darauf hingewiesen, dass dieses Dokument auf einer NTS-Spezifikation basiert, die sich in der Entwurfsphase befindet und nicht abgeschlossen ist. Einige Informationen in den derzeitigen Dokumenten der Spezifikation sind daher nicht mehr aktuell oder gar widersprüchlich. Auf die einzelnen Herausforderungen, die derzeit zu lösen sind, wird in den jeweiligen Textstellen hingewiesen. Zusätzlich werden diese im Kapitel 6.1 aufgeführt und näher untersucht. Die Auseinandersetzung mit der in diesem Kapitel beschriebenen Thematik, basiert auf der direkten Kommunikation mit der *Physikalisch-Technische Bundesanstalt* (PTB) und auf den folgenden Hauptdokumenten:

- *draft-ietf-ntp-network-time-security-15* (gültig bis: 26.03.2017) [9]
- *draft-ietf-ntp-using-nts-for-ntp-06* (gültig bis: 26.03.2017) [51]
- *draft-ietf-ntp-cms-for-nts-message-07* (gültig bis: 22.09.2016) [52]

3.1 Übersicht zu NTS

In diesem Abschnitt werden die Aspekte zu NTS behandelt, die zu deren Spezifikation führten. Ein kurzer Exkurs in die Vergangenheit bringt hierzu die Motivation und den Anstoß der Entwicklung näher. Anschließend erfolgt die genauere Untersuchung des NTP-Protokolls und dessen generelle Sicherheitsprobleme. Um diesen entgegenzuwirken, wurden in der Vergangenheit Sicherheitsmechanismen geschaffen, die jedoch nicht mehr den aktuellen Sicherheitsanforderungen gerecht werden. Die Probleme und Lücken dazu betrachtet Abschnitt 3.1.3 weiter. Die daraus resultierenden Verbesserungen stellen das neue Sicherheitsprotokoll NTS dar, dessen Ziele und Eigenschaften Unterkapitel 3.1.4 erläutert. Abschließend erfolgt ein kurzer Blick auf den NTS-Protokollaufbau in Bezug auf das OSI-Referenzmodell.

3.1.1 Entwicklungsverlauf und aktueller Stand der Spezifikation

Der Ursprung des NTS-Protokolls begann im Jahr 2011 mit der Untersuchung des NTP-Autokey-Verfahrens, im Rahmen einer Projektarbeit der TU Braunschweig in Zusammenarbeit mit der PTB [53]. Grund für die Untersuchung war unter anderem das Energiewirtschaftsgesetz (§ 21 ff.) [6], welches die Installation der *intelligenten* Stromzähler („*Smart Meter*“) in Neubauten vorsieht. Da diese Geräte eichpflichtig sind und für die korrekte Arbeitsweise eine genaue Uhrzeit benötigen, muss die Zeitquelle die entsprechenden Zeitdaten gesichert an die *Smart Meter* übertragen, um deren Beeinflussung durch einen Angreifer zu verhindern. Da in der Regel die Übertragung der Zeitdaten auf dem NTP-Protokoll beruht, wurden die bestehenden Sicherheitsmechanismen für dieses Protokoll analysiert. Das Ergebnis zeigte gleich mehrere schwerwiegende Schwachstellen der bestehenden Schutzmaßnahmen, die nach derzeitigen Stand keine ausreichende Sicherheit mehr bieten oder nicht skalieren. Nach Vorstellung der Ergebnisse auf dem 83. IETF-Treffen begann zunächst die Entwicklung der *Autokey-Version-2*-Spezifikation [54]. In deren Verlauf wurde Mitte 2013 die Spezifikation – aufgrund der negativen Assoziation mit dem Namen *Autokey* – in *Network Time Security* umbenannt und bei der IETF als Entwurf (*draft-ietf-ntp-network-time-security-00*) veröffentlicht. Diese detaillierte sich im Laufe der Zeit, worauf Ende 2014 ein zweites Dokument mit dem Namen *draft-ietf-ntp-cms-for-nts-message-00* entstand. Dies beinhaltet Informationen über den strukturellen Aufbau der definierten NTS-Nachrichten durch Verwendung von ASN.1- und CMS-Strukturen. Anfang 2015 erschien dann das dritte und letzte Dokument (*draft-ietf-ntp-using-nts-for-ntp-00*), welches direkt mit NTP verbunden wird. Dieses definiert die Regelungen von NTS, bezogen auf das NTPv4-Protokoll. Die zeitliche Entwicklung der Dokumente und die Vorstellungen auf den IETF-Treffen können Tabelle 3.1 entnommen werden.

Tabelle 3.1: Zeitlicher Verlauf der NTS-Spezifikation

Datum	IETF-Treffen	NTS-bezogenes Thema	vorliegende Entwürfe (Spezifikationen)
30.03.2012	**83**	Analysis of NTP's Autokey Protocol [55]	---
03.08.2012	**84**	Plan for Autokey Update [56]	---
09.11.2012	**85**	Autokey Version 2 Specification [57]	draft-sibold-autokey-00
15.03.2013	**86**	Autokey Version 2 Specification [58]	draft-sibold-autokey-02
02.08.2013	**87**	Network Time Security (NTS) [59]	draft-ietf-ntp-network-time-security-00
08.11.2013	**88**	Network Time Security (NTS) [60]	draft-ietf-ntp-network-time-security-01
07.03.2014	**89**	Network Time Security (NTS) [61]	draft-ietf-ntp-network-time-security-02
25.07.2014	**90**	Network Time Security (NTS) [62]	draft-ietf-ntp-network-time-security-04
14.11.2014	**91**	Network Time Security (NTS) [63]	draft-ietf-ntp-network-time-security-05 draft-ietf-ntp-cms-for-nts-message-00
22.03.2015	**92**	Network Time Security (NTS) [64]	draft-ietf-ntp-network-time-security-08 draft-ietf-ntp-using-nts-for-ntp-00 draft-ietf-ntp-cms-for-nts-message-02
24.07.2015	**93**	Network Time Security (NTS) [65]	draft-ietf-ntp-network-time-security-09 draft-ietf-ntp-using-nts-for-ntp-01 draft-ietf-ntp-cms-for-nts-message-04
06.11.2015	**94**	Network Time Security (NTS) [66]	draft-ietf-ntp-network-time-security-11 draft-ietf-ntp-using-nts-for-ntp-02 draft-ietf-ntp-cms-for-nts-message-04
08.04.2016	**95**	Network Time Security (NTS) [67]	draft-ietf-ntp-network-time-security-14 draft-ietf-ntp-using-nts-for-ntp-05 draft-ietf-ntp-cms-for-nts-message-06
22.07.2016	**96**	Network Time Security (NTS) [68]	draft-ietf-ntp-network-time-security-14 draft-ietf-ntp-using-nts-for-ntp-05 draft-ietf-ntp-cms-for-nts-message-06

Derzeit ist die NTS-Spezifikation schon weit fortgeschritten, aber noch nicht fertiggestellt. Der Broadcast-Modus wurde bisher noch nicht implementiert und es ist zu erwarten, dass es dort noch zu Verbesserungen der Spezifikation kommt. Auch die NTS-Definitionen für andere Zeitprotokolle wie z.B. PTP sind noch nicht umgesetzt, da die Priorität zurzeit auf dem NTP-Protokoll liegt. Der aktuelle Entwicklungsstand kann jederzeit bei der IETF verfolgt werden [69] [70].

3.1.2 NTP-bezogene Sicherheitsprobleme

Das NTP-Protokoll gehört zu den ältesten Netzwerkprotokollen, die es gibt. Dieses wurde von David L. Mills entwickelt und 1985 als RFC 958 [17] veröffentlicht. Im Laufe der Zeit wurde das Protokoll immer weiter verbessert und liegt seit 2010 in der aktuellen Version 4 [18] vor. Aufgrund der rasanten Entwicklung in der Computertechnik und deren immer stärkerer Vernetzung steigen auch die Sicherheitsanforderungen. Da bisher derartige Anforderungen nicht bzw. nur geringfügig in die Entwicklung von NTP eingeflossen sind, ist dieses Protokoll anfällig für Angriffe. Zeitinformationen werden im Klartext übertragen und können von Dritten manipuliert werden. Die bisherigen Schutzmaßnahmen werden kaum angewendet und bieten aufgrund mehrerer Schwächen keinen ausreichenden Schutz (siehe Kapitel 3.1.3). Um geeignete Schutzmaßnehmen zu designen, ist es zunächst notwendig, die Angriffspunkte von NTP zu kennen, um anschließend Gegenmaßnahmen zu entwickeln. Im Folgenden werden einige davon genannt, um einen allgemeinen Überblick zu verschaffen. Die vollständige Liste der bekannten Angriffsmethoden auf Zeitsynchronisierungsdienste kann im RFC 7384 [71] eingesehen werden.

Man-In-The-Middle-Angriffe

Bei dem *Man-In-The-Middle*-Angriff (MITM) handelt es sich um ein klassisches Angriffsmodell und stellt in Bezug auf NTP keine Neuigkeit dar (vgl. Abbildung 3.1). In dieser Konstellation setzt sich der Angreifer zwischen die beiden Kommunikationsteilnehmer, die in diesem Beispiel den NTP-Client und den NTP-Server darstellen.

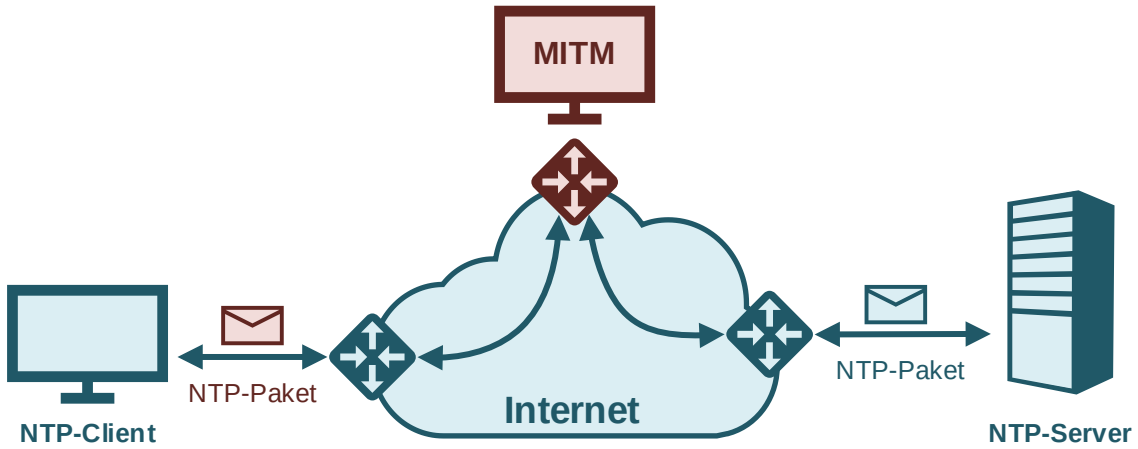


Abbildung 3.1: Angriffsszenario: Man-In-The-Middle-Attacke

Der Angreifer gibt sich durch IP-Spoofing als jeweils anderer Kommunikationsteilnehmer aus und hat nun die Möglichkeit, passiv die Kommunikation zu belauschen oder diese aktiv zu beeinträchtigen. Dazu kann er verschiedene Angriffe auf die Kommunikation und deren Teilnehmer durchführen, um verschiedene Effekte herbeizuführen. Da die NTP-Pakete im Klartext übertragen werden, können diese direkt manipuliert werden, um die Zeit des Clients zu verändern. Außerdem kann der Angreifer NTP-Pakete verwerfen oder einschleusen, um die Uhrzeit den Clients zusätzlich zu beeinflussen. Des Weiteren kann auch ein *Denial-of-Service*-Angriff (DoS) durchgeführt werden, um beispielsweise den Server zu überlasten und den NTP-Dienst zu unterbinden [72].

Der Package Injector

Ähnlich wie der MITM, setzt auch der *Package Injector* (PI) IP-Spoofing ein, um die Identität des NTP-Servers vorzugaukeln. Dieser greift im Gegensatz zum MITM nicht die Daten der Kommunikationsteilnehmer ab, sondern schleust neue NTP-Pakete ein (vgl. Abbildung 3.2). Somit kann eine Veränderung der Uhrzeit des Clients oder die Erhöhung von Ungenauigkeiten herbeigeführt werden. Durch Angriffe wie den *Kiss-of-Death*, kann die Verbindung zu einem oder mehreren Servern auch komplett deaktiviert werden. Im Folgenden werden einige mögliche Angriffe erläutert, die durch den PI oder den MITM durchgeführt werden können [71, p. 8].

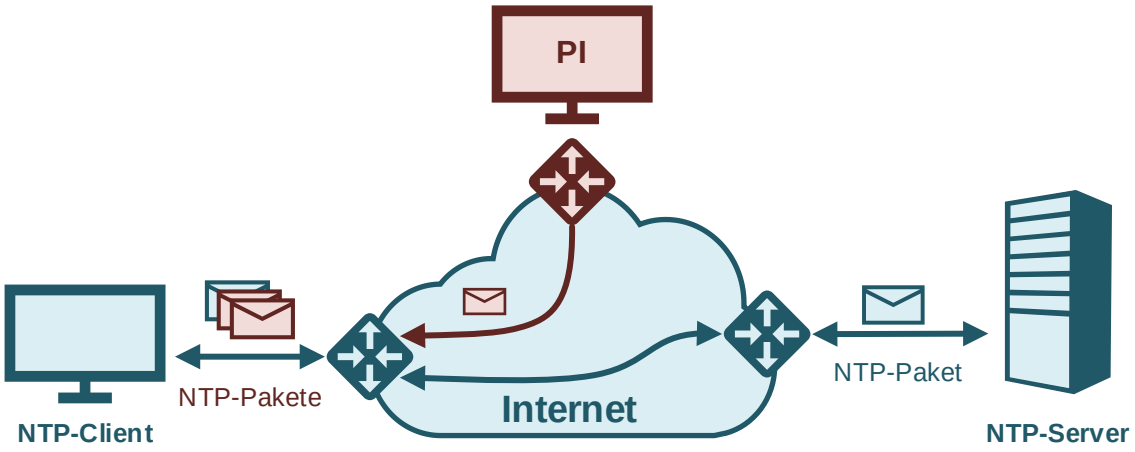


Abbildung 3.2: Angriffsszenario: der Package Injector

Denial-of-Service-Angriff

Unter einem *Denial-of-Service*-Angriff (DoS) versteht man eine Attacke auf einen Computer (z.B. einen Server) oder eine Komponente in einem Datennetz, mit dem Ziel, die Verfügbarkeit eines Dienstes durch Überlastung außer Kraft zu setzen. Wird dies dabei durch eine größere Anzahl anderer Systeme verursacht, so spricht man von einem *Distributed Denial of Service* (DDoS) [73] [74]. In Bezug auf NTP könnte ein Angreifer eine Anfrage an einem Zeitserver stellen, wobei er die IP-Adresse des Opfers als Quelle im NTP-Paket ersetzt. Der NTP-Server sendet daraufhin eine Antwortnachricht an den Zielcomputer, der jedoch nie eine Anfrage gesendet hat. Nutzt der Angreifer zudem Serverlisten, um viele Zeitserver auf diese Art und Weise anzusprechen, so wird der Zielrechner durch eine sehr hohe Anzahl von Antworten verschiedenster Zeitserver bombardiert. Da Antwortnachrichten durchaus um ein Vielfaches größer als Anfragenachrichten sein können, führt dies zu einer zusätzlichen Belastung[10]. Wird eine gewisse Intensität erreicht, führt dies zum Versagen des Zielrechners oder seiner Konnektivität zum Netzwerk. In der Vergangenheit konnten somit Netzlasten von 400 Gigabit pro Sekunde durch die Ausnutzung von NTP-Servern erreicht werden [75].

Kiss-of-Death DoS

Der *Kiss-of-Death*-Angriff (KoD) eine Unterart des DoS und verfolgt das Ziel, einen NTP-Client dauerhaft zu deaktivieren oder die Verwendung bestimmter NTP-Server zu unterbinden. Ein KoD-Paket kann ein NTP-Server beispielsweise bei Überlastung an einen Client senden, um ihm explizit die temporäre Dienstverweigerung mitzuteilen. Der entsprechende Client wird nun für eine gewisse Zeit keine Anfragen mehr an den Server stellen, wodurch dieser die Überlast ausgleichen kann. Würde ein NTP-Server stattdessen einfach die Anfragen des Clients verwerfen, so könnte dies eine höhere Anfragefrequenz des Clients zur Folge haben, wodurch die serverseitige Last weiter steigt. Die Verwendung von KoD-Paketen bieten allerdings einen Angriffspunkt auf NTP. In einem solchen Fall kann ein Angreifer zunächst die Anfragen des Clients belauschen und aus diesen Nachrichten die IP-Adresse des Zeitservers kopieren. Anschließend gibt sich der Angreifer mittels IP-Spoofing selbst als Zeitserver aus und sendet vermehrt KoD-Pakete an den Client. Durch dieses Vorgehen kann das Anfrageintervall des Clients extrem reduziert werden, wodurch eine Kommunikation mit dem jeweiligen Server praktisch deaktiviert wird. Führt der Angreifer dies für alle angefragten Zeitserver durch, kann der NTP-Client komplett deaktiviert werden. Durch diese Methode kann ein Client auch gezwungen werden, einen schlechten Zeitserver zu verwenden [76] [77].

[10] Diese Methode ist zudem als Verstärkungsangriff (*amplification DoS*) bekannt, bei dem kleine Clientanfragen große Serverantworten erzeugen. Durch sehr viele Anfragen kann somit eine massive Belastung der verfügbaren Netzwerkbandbreite erzeugt werden [75].

Replay-Angriff

Eine weitere Angriffsmöglichkeit bietet der sogenannte *Replay*-Angriff. Hierbei zeichnet der Angreifer die Kommunikation zwischen einem Client und dem Server auf. Anschließend kann der Angreifer auf die Anfragen des Clients mit den aufgezeichneten Serverpaketen antworten. Diese Gefährdung entzieht sich kryptografischer Sicherungsmethoden, da Zertifikate und digitale Signaturen bei verwendeten Schutzmechanismen weiterhin gültig bleiben. Ein Client kann somit nicht erkennen, ob es sich um eine echte oder aufgezeichnete Nachricht handelt. Eine Gegenmaßnahme bietet eine sogenannte *Nonce*, bei der es sich um eine Zufallszahl handelt, die bei jeder Anfrage neu generiert und der Nachricht hinzugefügt wird. Der Server sendet in der Antwortnachricht diese Nonce zurück, wodurch der Client anschließend die empfangene Nonce mit der gespeicherten vergleichen kann. Handelt es sich um eine echte Serverantwort, so müssen diese identisch sein [78].

Packet-Delay-Manipulation

Auch diese Gefährdungen entziehen sich kryptografischer Sicherungsmaßnahmen, bei der ein Angreifer die Serverantworten abfängt und verzögert an den Client weiterleitet. Bei NTP führt dies aufgrund der asymmetrischen Verzögerung des Paketaustausches zu einer möglichen Zeitverfälschung (vgl. Abbildung 3.3) [71]. Die Zusammensetzung der dargestellten Zeitstempel und die daraus resultierenden Berechnungen, betrachtet Kapitel 2.2.1 näher.

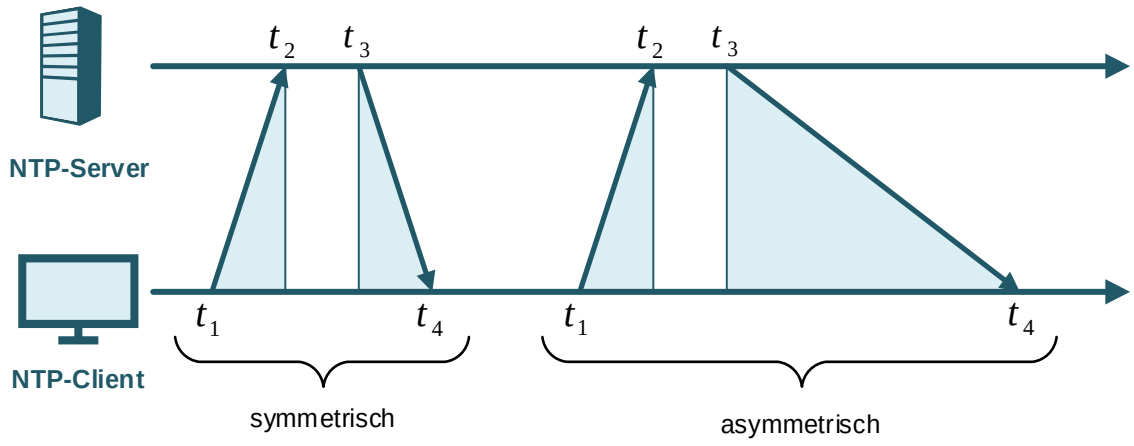


Abbildung 3.3: Asymmetrische NTP-Paketlaufzeiten durch einen Packet-Delay-Angriff

Die Folge eines *Packet-Delay*-Angriffs ist die Erhöhung des Offsets zwischen dem Client und dem jeweiligen Server. Durch den clientseitigen Vergleich der Zeiten mit anderen NTP-Servern, kann dies zur Ignorierung des jeweiligen Servers führen, da dieser augenscheinlich schlechtere Werte zur Verfügung stellt. Wird ein derartiger Angriff auf NTP-Nachrichten verschiedener Zeitserver angewendet, so steigt die clientseitige Ungenauigkeit.

3.1.3 Probleme bestehender Sicherheitsmechanismen in NTP

Zum Schutz vor Manipulationen der NTP-Pakete existieren bereits zwei Sicherheitsmechanismen, die auf kryptografischer Grundlage basieren. Dies ist zum einen das *Symmetric-Key*-Verfahren und zum anderen das *Autokey*-Verfahren. Beide Lösungen werden allerdings aufgrund Ihrer Nachteile oder Schwächen kaum verwendet, obwohl die Nachfrage nach gesicherten Zeitinformationen steigt.

3.1.3.1 Das *Symmetric-Key*-Verfahren

Das *Symmetric-Key*-Verfahren („*Symmetric Key Cryptography*“) wurde im Jahre 1992 als Bestandteil des NTPv3-Protokolls im RFC 1305 [79] veröffentlicht. Dieses bietet durch die Nutzung von kryptografischen Funktionen die Möglichkeit, eine Prüfung der Authentifikation der beteiligten NTP-Kommunikationspartner vorzunehmen. Durch die Verwendung eines MAC wird die Integrität der NTP-Nachrichten gewährleistet. Ursprünglich wurde dafür die DES-Verschlüsselung im CBC-Betriebsmodus verwendet. Diese wurde jedoch später durch einen HMAC-Algorithmus unter Verwendung des MD5- bzw. SHA1-Algorithmus und einem privaten Schlüssel ersetzt [80]. Da die kryptografische Sicherheit insbesondere von der Größe des privaten Schlüssels abhängig ist, gilt ein MD5-HMAC trotz der Kollisionsanfälligkeit[11] des MD5-Algorithmus als sicher [45] [81]. Der MAC, der an die jeweiligen NTP-Paketen angehängt wird, besteht aus einer 32 Bit großen Schlüsselkennung (*KeyID*), gefolgt von einem 128 Bit oder 160 Bit großen Nachrichten-Hash. Der Nachrichten-Hash berechnet sich über das komplette NTP-Paket, wobei die Größe des Hash-Wertes vom verwendeten Algorithmus abhängt (128 Bit bei MD5 und 160 Bit bei SHA1). Der *KeyID* identifiziert einen von 65.534 möglichen Schlüsseln auf eindeutige Weise. Dieser hat eine Größe von 32 Bit und wird zwischen Client und Server ausgehandelt. Der große Nachteil dieses Verfahrens ist die fehlende Skalierbarkeit. Die *KeyIDs* mit den jeweiligen Schlüsseln werden über einen Key-Generator („*ntp-keygen*“) erzeugt [82] und anschließend als Datei („*ntp.keys*“) gespeichert. Sowohl der Client, als auch der Server müssen diese Datei verwenden, um über den *KeyID* den richtigen Schlüssel zu nutzen. Dieser Austausch dieser Datei ist vor der Nutzung des NTP-Dienstes über einen geschützten Kanal zu tätigen (*pre-shared keys*).

[11] Eine Kollision bezeichnet den Fall, dass zwei unterschiedliche Datenblöcke den selben Hash-Wert erzeugen

3.1.3.2 Das *Autokey*-Verfahren

Eine Alternative zum *Symmetric-Key*-Verfahren bietet die *Autokey*-Methode, welche im NTPv4-Protokoll verfügbar ist und im RFC 5906 [83] zusätzlich zum *Symmetric-Key*-Verfahren definiert wurde. *Autokey* bietet gegenüber dem *Symmetric-Key*-Verfahren den Vorteil, dass die Schlüssel nicht vorab ausgetauscht werden müssen. Somit ist dieses Verfahren skalierbar und einfach für die entsprechenden Kommunikationsteilnehmer nutzbar. Die *Autokey*-Methode basiert ebenfalls auf der Verwendung von symmetrischen Schlüsseln (den sogenannten *Autokeys*), die nur für jeweils ein NTP-Paket gültig sind. Die Berechnungsgrundlage dieser Schlüssel erfolgt über ein gemeinsames Geheimnis (dem *Cookie*), das im Kommunikationsverlauf vom Server an den Client übertragen wird.

Kurze Ablaufbeschreibung

Der Protokollablauf erfolgt durch mehrere Nachrichten, in denen zunächst die Parameter zwischen Client und Server ausgetauscht werden. In dieser Phase überträgt der Server zusätzlich sein X.509-Zertifikat an den Client, wodurch dieser die Authentifikation des Servers bestätigen kann. Zusätzlich wird der darin enthaltende öffentliche Schlüssel zur Prüfung der Signaturen verwendet. Der Austausch dieser Informationen wird mit den Erweiterungsfeldern realisiert, die im NTPv4-Protokoll zur Verfügung stehen. Das Cookie, welches der Server an den Client sendet, wird entsprechend der folgenden Gleichung 3.1 berechnet:

$$Cookie = MSB_{32}\big(H(Client\ IP \parallel Server\ IP \parallel 0 \parallel Server\ Seed)\big) \tag{3.1}$$

Dieses besitzt eine Länge von 32 Bit, wobei nur die höherwertigsten Bit aus dem Berechnungsergebnis der Hash-Funktion verwendet werden. Als Hash-Funktion kommt der MD5-Algorithmus zum Einsatz, der den entsprechenden Wert über die Verkettung mehrerer Parameter errechnet. Dazu zählen die IP-Adresse[12] des Clients, die IP-Adresse des Servers, eine 32 Bit lange Nullfolge und der *Server Seed*, der eine geheime Zufallszahl des Servers darstellt. Nachdem der Client das Cookie vom Server erhalten hat, kann dieser nun den *Autokey* gemäß der folgenden Gleichung 3.2 berechnen:

$$Autokey = H(Sender\ IP \parallel Empfänger\ IP \parallel KeyID \parallel Cookie) \tag{3.2}$$

Der hierfür verwendete Hash-Algorithmus ist wieder MD5, der den Hash-Wert über die verketteten Parameter berechnet. Die verwendete *KeyID* ist 32 Bit lang und kann vom Client

[12] Dies kann entweder eine 32 Bit lange IPv4-Adresse oder eine 128 Bit lange IPv6-Adresse sein.

frei gewählt werden. Stellt der Client nun eine Anfrage an den Server, so fügt dieser die verwendete *KeyID* und einen MAC hinzu. Dieser berechnet sich gemäß der Gleichung 3.3:

$$MAC = H(Autokey \parallel NTP\text{-}Paket) \tag{3.3}$$

Der berechnete MAC besitzt eine Länge von 128 Bit, bei Verwendung eines MD5-Algorithmus oder 160 Bit bei Nutzung von SHA1. Die Berechnung erfolgt mit dem *Autokey* und dem kompletten Inhalt des NTP-Pakets. Anschließend wird der MAC an das NTP-Paket angehängt und an den Server übertragen. Da dieser zustandslos ist, berechnet er zunächst das Cookie und den jeweiligen *Autokey*, um damit nun den MAC der NTP-Nachricht zu generieren und es mit dem empfangenen MAC zu vergleichen. Sind diese identisch, so erzeugt der Server eine Antwortnachricht. Diese enthält die Zeitinformationen, die zuvor empfangene *KeyID* und einen neuen MAC. Da bei der erneuten Berechnung des *Autokey* die IP-Adressen vertauscht werden, ist der berechnete *Autokey* mit dem zuvor empfangenen *Autokey* nicht identisch. Nach Erhalt der Serverantwort prüft der Client seinerseits die Nachricht nach dem gleichen Schema und kann diese bei erfolgreicher Überprüfung zur Zeitsynchronisation nutzen. Der Zusammenhang der einzelnen Komponenten wird zur Übersicht in der folgenden Abbildung 3.4 dargestellt:

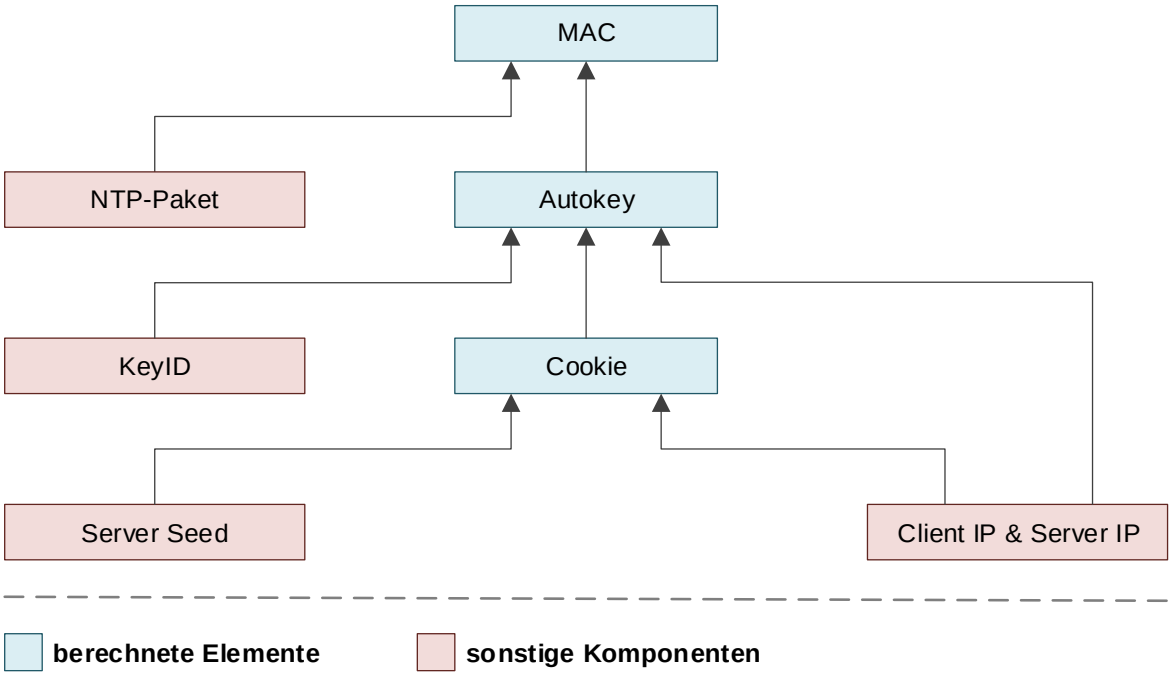


Abbildung 3.4: Zusammensetzung der Autokey-Protokollelemente

Schwachstellen im *Autokey*-Verfahren

Die Sicherheitsanalyse durch S. Röttger [53] zeigte viele Schwachstellen, die eine sichere Verwendung von NTP mit dem *Autokey*-Verfahren nicht ermöglichen. Folgende Schwachstellen wurden hierbei aufgedeckt:

- **Der *Server Seed*:** Dieser ist nur 32 Bit lang und bietet keinen ausreichenden Schutz. Ein Angreifer kann ein Cookie vom Server anfordern und anschließend dessen *Server Seed* per *Brute-Force*-Angriff [84] ermitteln. Hierbei berechnet der Angreifer selbst das Cookie und probiert dabei alle möglichen Werte des *Server Seed* aus. Ist das berechnete Cookie mit dem zuvor empfangenen identisch, so hat der Angreifer den richtigen *Server Seed* gefunden. Auf einem Standard-PC kann auf diese Weise der *Server Seed* in etwa 30 Minuten gefunden werden.

- **Das Cookie:** Auch dieses ist nur 32 Bit lang und kann nach dem gleichen Schema geknackt werden, nachdem es vom Server abgefangen wurden ist. Außerdem werden zur Generierung IP-Adressen verwendet, die leicht zu manipulieren sind und kleinen Schutz bieten.

- **Identitätsprüfung des Servers:** Die angewandten *Identity Schemes* [85] zur Authentifikation des Servers bieten keinen ausreichenden Schutz. Der Client vertraut automatisch jedem Serverzertifikat, das selbstsigniert ist und in dessen *Extended-Key-Usage*-Erweiterungsfeld das *trustRoot* Bit gesetzt wurde. Ein Angreifer kann mit Leichtigkeit ein solches Zertifikat generieren und die entsprechenden Werte setzen. Ein weiterer Mangel stellt das *Challenge-Response*-Verfahren dar. Dieses wird inkorrekt angewendet, wodurch ein Angreifer den Schutzmechanismus umgehen kann.

- **Identitätsprüfung des Clients:** Der Server ist zustandslos und nutzt zur Identitätsprüfung ausschließlich die IP-Adresse des Clients. Dadurch kann sich ein Angreifer mittels IP-Spoofing als Client tarnen. Zusätzlich können Nachrichten außerhalb des Protokollablaufs an den Server gesendet werden.

- **IP-Spoofing**: Durch Manipulation der IP-Adressen, kann ein Angreifer das Cookie für den entsprechenden Client lesbar an sich senden.

3.1.4 Sicherheitsziele von NTS

Basierend auf den Analysen der bestehenden Sicherheitsmechanismen sowie den bekannten Angriffen auf Zeitsynchronisierungsprotokolle und deren Gegenmaßnahmen, die im RFC 7384 „*Security Requirements of Time Protocols in Packet Switched Networks*“ [71] thematisiert werden, wurden die entsprechenden Ziele für das NTS-Protokoll gesetzt. Folgende Punkte werden hierbei berücksichtigt:

- **Authentizität:** Der Client muss die Möglichkeit haben, seinen Zeitserver zu authentifizieren.

- **Autorisierung:** Der Client muss die Möglichkeit haben, die Autorisierung des Zeitservers zu prüfen. Außerdem soll auch dem Zeitserver diese Option zur Verfügung stehen, um beispielsweise nur zugelassenen Clients die Zeitdaten zur Verfügung zu stellen.

- **Request-Response-Konsistenz:** NTS räumt dem Client die Möglichkeit ein, die empfangenen Servernachrichten dahingehend zu prüfen, ob diese auch zu der zuvor gesendeten Anfragenachricht passen.

- **Integrität:** Das NTS-Protokoll soll die Integrität von Zeitsynchronisationspaketen durch einen Message Authentication Code (MAC) sichern.

- **Anwendbarkeit in bestehenden Zeitprotokollen:** NTS kann dazu genutzt werden, um verschiedene Zeitsynchronisierungsprotokolle um Sicherheitsfunktionen zu erweitern (z.B. NTP oder PTP). Dabei sollen alle Betriebsmodi unterstützt werden.

- **Keine Vertraulichkeit:** Das NTS-Protokoll soll keine Vertraulichkeit von Zeitinformationen sicherstellen (die Verschlüsselung der Daten), um die Kompatibilität zu erhöhen.

- **Geringe zusätzliche Latenz:** Die durch NTS erzeugten Latzenzzeiten müssen so gering wie möglich gehalten werden, da dies die Genauigkeit des Zeitprotokolls beeinträchtigt (siehe dazu Kapitel 3.6.8).

- **Kompatibilität:** NTS darf Zeitsynchronisierungsprotokolle nicht negativ beeinflussen. Auch ungesicherte Verbindungen müssen nach wie vor in der Lage sein, die Zeitnachrichten wie gewohnt zu verarbeiten, die von einem Teilnehmer mit aktivierten NTS übertragen werden.

- **Leistung:** NTS darf die Leistungsfähigkeit und die Genauigkeit der Zeitsynchronisierungsprotokolle nicht signifikant verschlechtern und somit deren Zweck *ad absurdum* führen. Auch die zusätzlich benötigten Ressourcen (Speicher und Rechenleistung) sollten so gering wie möglich sein.

- **Hybride Funktionalität:** Es muss die Möglichkeit gegeben werden, das entsprechende Zeitsynchronisierungsprotokoll sowohl mit, als auch ohne NTS betreiben zu können.

- **Serverseitige Zustandslosigkeit:** Während der Nutzung aller Sicherheitsmechanismen bleibt der Server zustandslos und speichert keinerlei Informationen oder Verbindungsdaten der bedienten Clients.

- **Schutz vor Verstärkungsattacken:** Die komplette Kommunikation auf NTS-Ebene muss vor Verstärkungsattacken (*amplification DoS*) geschützt sein. Dies stellt eine besondere Form der DoS-Attacke dar, bei der kleine Clientanfragen große Serverantworten generieren (s.a. Kapitel 3.1.2).

- **Skalierbarkeit:** Der durch NTS erzeugte Schutzmechanismus soll unabhängig von der Anzahl der Kommunikationsteilnehmer sein. Neue Verbindungen zwischen Client und Server sollen dynamisch möglich sein, ohne vorher Parameter oder Schlüssel auszutauschen (*pre-shared key*).

Die im RFC 7384 [71] gestellten Anforderungen und deren Berücksichtigung in NTS werden in der folgenden Tabelle 3.2 noch mal zur Übersicht dargestellt:

Tabelle 3.2: Sicherheitsanforderungen an Zeitprotokolle gemäß RFC 7384

Kapitel im RFC	Anforderung aus dem RFC 7384-Dokument	Anforderungs-Level	NTS
5.1.1	Authentifizierung der Server	MUSS	OK
5.1.1	Autorisierung der Server	MUSS	OK
5.1.2	Rekursive Authentifizierung der Server (Stratum 1)	MUSS	OK
5.1.2	Rekursive Autorisierung der Server (Stratum 1)	MUSS	OK
5.1.3	Authentifizierung der Clients	KANN	Optional
5.1.3	Autorisierung der Clients	KANN	Optional
5.2	Integritätsschutz	MUSS	OK
5.3	Spoofing-Abwehr	MUSS	OK
5.4	Schutz vor DoS-Attacken gegen das Zeitprotokoll	SOLLTE	OK
5.5	Schutz vor Replay-Attacken	MUSS	OK
5.6.1	Erneuerung von kryptografischen Schlüsseln	MUSS	OK
5.6.2	Sicherheitsvereinbarung (*security association*)	SOLLTE	OK

5.6.3	Sicherheitsvereinbarung für Unicast- und Multicast-Verbindungen	SOLLTE	OK
5.7	Leistung: Keine signifikante Verschlechterung der Qualität in der Zeitübertragung	MUSS	OK
5.7	Leistung: Minimierung der Rechenlast	SOLLTE	OK
5.7	Leistung: minimale Speicheranforderungen	SOLLTE	OK
5.7	Leistung: minimale Erhöhung der benötigten Bandbreite	SOLLTE	OK
5.8	Vertraulichkeitsschutz	KANN	NEIN
5.9	Schutz vor *packet delay* und Abfangangriffen (*interception*)	MUSS	Einge-schränkt*
5.10.1	Sicherheitsmodus	MUSS	OK
5.10.2	Hybridmodus (Unterstützung ungesicherter Teilnehmer)	SOLLTE	---

3.1.5 NTS im OSI-Referenzmodell

In diesem Abschnitt wird kurz auf die Zuordnung des NTS-Protokolls im OSI-Referenzmodell eingegangen. Hierbei handelt es sich um ein Modell, das den logischen Kommunikationsfluss zwischen zwei Teilnehmern beschreibt, wobei die bei einer Verbindung anfallenden Aufgaben den verschiedenen Protokollschichten zugeteilt werden (s.a. Kapitel 2.1). Im dargestellten Beispiel (vgl. Abbildung 3.5) wird das NTPv4-Protokoll genutzt, welches über das verbindungslose Transportprotokoll UDP (Port 123) kommuniziert. Die Adressierung erfolgt mit IPv4 bzw. IPv6 über eine Ethernet-Verbindung.

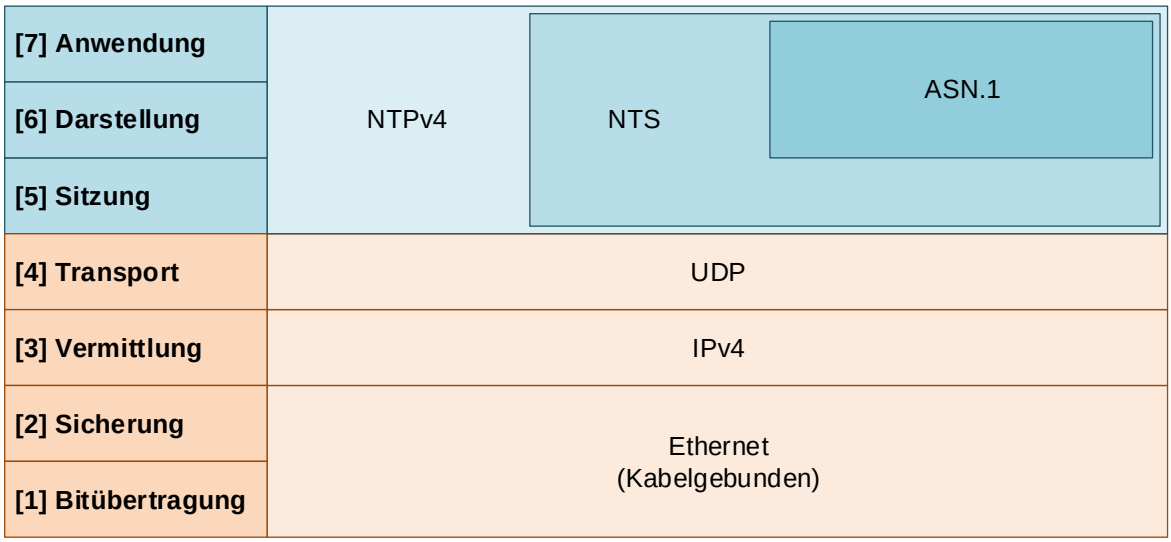


Abbildung 3.5: Zuordnung von NTS im OSI-Referenzmodell

NTS wird ebenso wie das NTP-Protokoll den obersten drei Schichten im OSI-Modell zugeteilt. Der Datenaustausch mit NTS erfolgt intern über das NTP-Protokoll, durch die Verwendung seiner Erweiterungsfelder und den definierten Schnittstellen. Die kommunizierten Daten müssen an dieser Stelle zunächst durch die ASN.1-Funktionalität kodiert bzw. dekodiert werden (s.a. Kapitel 2.3). Dadurch können die NTS-spezifischen Daten zwischen heterogenen Systemen ausgetauscht werden. Die Zuordnung von ASN.1 in das OSI-Modell erfolgt daher in der Darstellungsschicht.

3.2 NTS im Unicast-Betrieb

Der Unicast-Betrieb stellt im NTS-Protokoll die Kommunikation zwischen zwei Teilnehmern dar, die sich im Client/Server-Modus zueinander befinden. Hierbei nutzen beide Teilnehmer dasselbe Zeitprotokoll (z.B. NTP) mit dem eingebetteten NTS als Sicherheitsmechanismus. Der Integritätsschutz der ausgetauschten Zeitnachrichten wird dabei über einen symmetrischen Schlüssel (dem sogenannten Cookie) realisiert, welches im Verlauf der Kommunikation ausgetauscht wird. Damit können Zeitnachrichten mit einem MAC versehen und durch die Teilnehmer überprüft werden. Im Unicast-Betrieb wird die volle Kompatibilität gewährleistet, sodass auch Teilnehmer die Pakete verarbeiten können, die nicht NTS unterstützen.

3.2.1 Eigenschaften der Kommunikationspartner

Die Kommunikation zwischen Client und Server findet in den meisten Anwendungen über das Internet statt. Wird als Trägernachricht das NTPv4-Protokoll verwendet, so kommunizieren Client und Server über das Transportprotokoll UDP auf Port 123. Die Kommunikation zwischen den Teilnehmern erfolgt somit verbindungslos. Dies stellt zum jetzigen Zeitpunkt besondere Herausforderungen an das NTS-Design dar, die im Kapitel 6.1 gesondert behandelt werden.

3.2.1.1 Eigenschaften des NTS-Clients

Der Client ist im NTS-Protokoll zustandsbehaftet und baut im Verlaufe der Kommunikation eine „*lose Verbindung*“ zum Server auf. Das bedeutet der Client speichert ausgehandelte Parameter und ist in der Lage, eine Kommunikation nach einer Unterbrechung an gleicher Stelle fortzuführen, ohne eine erneute Aushandlung der Parameter starten zu müssen. Jeder NTS-Client muss die simultane Kommunikation zu mehreren Servern unterstützen, wobei

die Parameter und Zustände für jeden einzelnen Server individuell sind. Die vom Client unterstützen Parameter können manuell festgelegt werden, müssen allerdings die Mindestanforderung des Servers erfüllen. Die für die Verhandlungsphase benötigten Schlüsselpaare und Zertifikate darf der Client selber generieren, sofern der Server keine Autorisierung verlangt. Wird letzteres benötigt, so muss das Zertifikat des Clients von einer Zertifizierungsstelle ausgestellt werden, wodurch die Identität und die Rechte des Clients beglaubigt werden.

3.2.1.2 Eigenschaften des NTS-Servers

Der Server ist im Gegensatz zum Client zustandslos und speichert keinerlei Daten. Dies hat zur Folge, dass der Server ausschließlich auf die Anfragen des Clients reagieren kann, ohne vorher ausgetauschte Nachrichten zu berücksichtigen. Dies senkt die vom Server benötigten Ressourcen, wie etwa der benötigte Arbeitsspeicher und die benötigte Rechenkapazität. Der Server muss zudem in der Lage sein, viele Clients zu bedienen und die durch NTS zusätzlich erzeugten Latenzzeiten gering zu halten. Die vom NTS-Server unterstützten Verbindungsparameter können frei festgelegt werden, müssen aber den örtlichen bzw. gesetzlichen Sicherheitsstandards entsprechen [86] [87]. Die für die Verbindung benötigten Zertifikate müssen von einer Zertifizierungsstelle ausgestellt werden, sofern der NTS-Server öffentlich zugänglich ist. Ausnahmen stellen hier beispielsweise lokale Netzwerke von größeren Unternehmen dar, die intern eine Zertifizierungsstruktur für die NTS-Clients und -Server zur Verfügung stellen.

3.2.2 Übersicht zur Funktionsweise des Unicast-Betriebs

Die Unterteilung des Protokollablaufs erfolgt bei NTS in die Verhandlungs- und die Zeitsynchronisationsphase[13] (vgl. Abbildung 3.6). Die Ausführung der Verhandlungsphase erfolgt in separaten Nachrichten (s.a. Kapitel 3.3) und werden so wie die Zeitnachrichten über das Zeitprotokoll (NTP) übertragen. Auch wenn eine NTS-Implementierung die Verhandlungsphase beinhalten muss, so ist es dennoch erlaubt, diese über einen externen Mechanismus zu realisieren, sofern er die Anforderungen der NTS-Spezifikationen erfüllt [9, p. 7]. Somit ist es beispielsweise möglich, die Verhandlungsphase in eine separate TLS-Verbindung auszulagern.

[13] Hiermit ist nicht nur die reine Synchronisierung der lokalen Uhrzeit gemeint, sondern auch der Austausch der Zeitnachrichten.

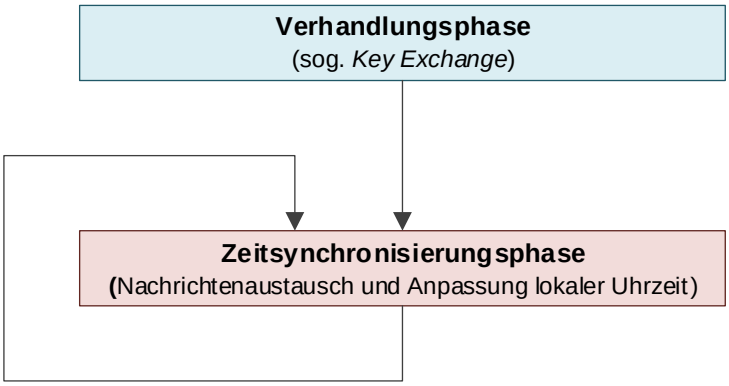


Abbildung 3.6: Kernphasen des NTS-Protokolls

In der Verhandlungsphase findet die Aushandlung von Parametern und die Überprüfung der Kommunikationsteilnehmer statt. Die Verwendung von X.509-Zertifkaten [47] sowie einer Zertifizierungsstruktur[14] ermöglicht dem Client die Prüfung der Authentifizierung und Autorisierung des Servers. Zudem ist auch der Server in der Lage, die Autorisierung des Clients zu verlangen und zu prüfen, sofern es dieser verlangt. Ein weiterer Verwendungszweck der Zertifikate ist der Schlüsselaustausch. Sowohl Client als auch Server besitzen ein Schlüsselpaar, das aus einem öffentlichen Schlüssel und einem privaten Schlüssel besteht. Das Schlüsselpaar des Clients wird zur Verschlüsselung und Entschlüsselung des Cookies verwendet. Das Schlüsselpaar des Servers dient der digitalen Signatur der NTS-Nachrichten und ermöglicht dem Client die Echtheitsprüfung der Nachrichten und somit auch des Cookies. Ist ein Teilnehmer sowohl als Client als auch als Server aktiv, muss dieser unterschiedliche Schlüsselpaare verwenden. Im Falle eines Angriffs auf einem Schlüsselpaar werden dadurch nicht beide Rollen gefährdet.

Das Cookie stellt in NTS einen symmetrischen Schlüssel dar, der während der Verhandlungsphase verschlüsselt vom Server an den Client übertragen wird und für die spätere Generierung des MAC von Bedeutung ist. Da der Server zustandslos ist, kann kein zufällig generiertes Cookie zum Einsatz kommen, sondern es muss bei jeder Zeitanfrage des Clients neu berechnet werden. In diesem Falle muss die Berechnung stark deterministisch sein, um das exakte Cookie reproduzieren zu können. Nur so ist wird die Prüfbarkeit des MAC einer NTS-Nachricht gewährleistet. Aus diesem Grund ist es wichtig, dass ein Angreifer nicht in der Lage ist, dieses Cookie zu errechnen, und somit die NTS-Nachrichten manipulieren kann. Die Berechnung des Cookies erfolgt immer serverseitig gemäß der folgenden Gleichung 3.4:

[14] Ein Zertifizierungspfad, mit dem sich Zertifikate bis zu einer *Trusted Authority* zurückverfolgen lassen und somit deren Echtheit beglaubigen.

$$Cookie = MSB_<B_c> \left(MAC(Server\ Seed,\ KIV)\right) \tag{3.4}$$

MSB_<B$_c$> () schneidet dabei die höchstwertigsten Bits des Ergebnisses der MAC-Funktion ***MAC()*** ab, wobei B_c die Anzahl der Bits repräsentiert. Zum jetzigen Stand der NTS-Spezifikation sind dies 128 Bit. Die MAC-Funktion bezeichnet einen ausgehandelten Algorithmus (s.a. Kapitel 3.6.3) zur Berechnung des Cookies und ist nicht mit dem angehängten MAC einer NTS-Nachricht gleichzusetzen. Als Eingabewerte nutzt die MAC-Funktion einen *Server Seed* und einen *Key Input Value* (KIV). Der *Server Seed* ist eine vom Server generierte und geheim gehaltene Zufallszahl (s.a. Kapitel 3.6.2). Diese ist an einem Hash-Algorithmus gebunden und verfällt periodisch, um die Lebensdauer der erzeugten Cookies zu begrenzen. Der KIV wird vom Client generiert und muss bei jeder Zeitanfrage an den Server übertragen werden[15]. Das daraus generierte Cookie ist somit an diesem Client gebunden. Der für das Cookie benötigte KIV berechnet sich entsprechend der Gleichung 3.5:

$$KIV = MSB_<B_{KIV}> \left(H(Zertifikat\ des\ Clients)\right) \tag{3.5}$$

Analog zur Berechnung des Cookies werden auch beim KIV nur die höchstwertigsten $\boldsymbol{B_{KIV}}$ Bits übernommen. Derzeit liegt dieser Wert bei 128 Bit. Als Funktion kommt ein Hash-Algorithmus zum Einsatz, auf den sich Client und Server während der Verhandlungsphase einigen. Der berechnete Hash-Wert wird dabei über das Zertifikat des Clients gebildet.

Nachdem das vom Server erzeugte Cookie an den Client übertragen wurde, beginnt die Zeitsynchronisationsphase. Diese Phase kann vom Client so oft wiederholt werden, wie das empfangene Cookie gültig ist. Dazu sendet der Client eine Zeitanfrage an den Server und sichert diese Nachricht mit einem MAC. Die Berechnung des MAC wird in der folgenden Gleichung 3.6 definiert:

$$MAC = MSB_<B_{MAC}> \left(MAC(Cookie,\ NTP\text{-}Paket)\right) \tag{3.6}$$

Auch bei dieser Berechnung werden nur die höchstwertigsten $\boldsymbol{B_{MAC}}$ Bits (zurzeit 128 Bit) berücksichtigt. Als Parameter der MAC-Funktion wird das ausgetauschte Cookie und das zu sendende NTP-Paket verwendet. Das NTP-Paket darf zu diesem Zeitpunkt nicht mehr verändert werden[16] und muss in einer serialisierten Form für die Berechnung vorliegen. Der generierte MAC wird anschließend in Form eines Erweiterungsfeldes an das NTP-Paket angehängt (s.a. Kapitel 3.4.1) und an den Server übertragen.

[15] . Die NTS-Spezifikation erlaubt es zusätzlich, dass der KIV auch über Dritte (z.B. eine externe Verteilungsinstanz) bezogen werden kann. Dies könnte bei Verhandlungsphasen genutzt werden, die nicht über das NTS-Protokoll laufen.

[16] Hiermit ist vor allem die Aktualisierung der Zeitstempel gemeint.

Nach Erhalt der Zeitanfrage beim Server wird zunächst der MAC[17] aus dem NTP-Paket extrahiert. Da der ausgehandelte MAC-Algorithmus und der KIV in der Zeitanfrage des Clients enthalten sind, kann der Server daraus erneut das Cookie errechnen. Mit diesem generiert der Server gemäß Gleichung 3.6 den MAC und nutzt als Parameter das empfangene NTP-Paket. Der mit dem NTP-Paket empfangene MAC wird dabei von der Berechnung ausgeschlossen. Im Anschluss wird der errechnete MAC mit dem empfangenen MAC verglichen. Stimmen diese überein, so liefert es dem Server die Bestätigung, dass dieses NTP-Paket nicht verändert wurde. Außerdem garantiert es dem Server, dass es sich um einen autorisierten Client handelt, sofern dies vom Server gefordert war. Bei erfolgreicher Prüfung des MAC erzeugt der Server eine Antwortnachricht, versieht diese ebenfalls mit einem MAC und überträgt die MAC-gesicherte Nachricht an den Client. Dieser überprüft die empfangene Nachricht nach dem gleichen Schema, nutzt dabei allerdings das bereits gespeicherte Cookie für die notwenigen Berechnungen. Ist die Überprüfung erfolgreich, so kann der Client die empfangenen Zeitdaten zu Synchronisierung der lokalen Uhrzeit verwenden, indem dieser die Freigabe der Zeitdaten auf NTP-Ebene erteilt. Schlägt die Gültigkeitsprüfung fehl, so ist entweder die empfangene Nachricht korrupt oder das Cookie abgelaufen. In diesem Falle kehrt der Client in die Verhandlungsphase zurück und fordert beim Server ein neues Cookie an. Der Zusammen zwischen den benötigten Elementen zur Generierung des MAC, wird in der folgenden Abbildung 3.7 nochmals verdeutlicht:

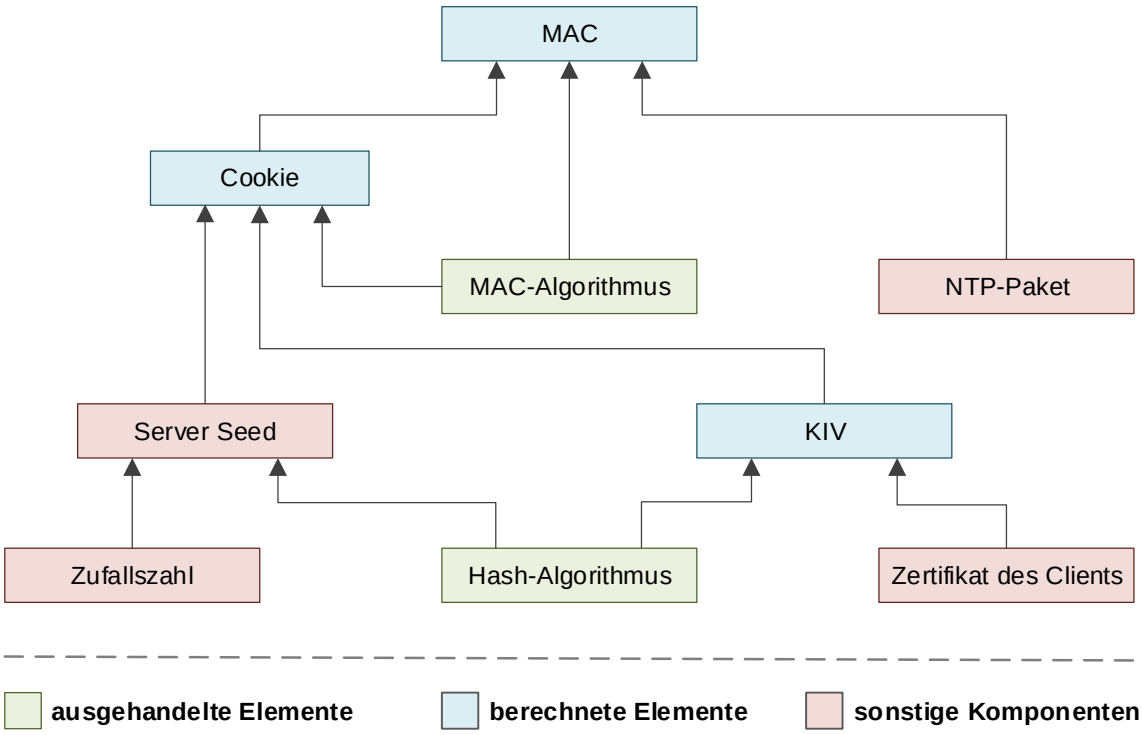


Abbildung 3.7: Zusammensetzung der NTS-Protokollelemente

[17] Dies bezieht sich, sofern es nicht explizit genannt wird, immer auf die im Erweiterungsfeld angehängte NTS-MAC und nicht auf die native NTP-MAC.

Die blauen Elemente stellen Berechnungen dar, die für den Integritätsschutz der NTS-Nachrichten durchgeführt werden müssen. Grüne Elemente sind Parameter, die im Verlaufe der Verhandlungsphase ausgetauscht werden. Gelbe Elemente sind Ressourcen oder Daten, die für die Berechnungen benötigt, aber nicht bzw. nicht direkt ausgehandelt werden. Der dargestellte *Server Seed* wird an dieser Stelle nicht errechnet, sondern es wird eine zuvor generierte Zufallszahl anhand des Hash-Algorithmus ausgewählt[18] (Stichwort: *Lookup table*).

3.3 Nachrichtentypen im Unicast-Betrieb

Die in der NTS-Spezifikation definierte Verhandlungsphase setzt sich aus drei unterschiedlichen Nachrichtenpaaren zusammen (vgl. Abbildung 3.8). Dies sind die Zugriffsnachrichten, die Assoziationsnachrichten und die Cookie-Nachrichten. Diese werden in fester Abfolge durchlaufen und werden zum Austausch der Parameter und Kommunikationsdaten benötigt. Die Abfolge dieser Nachrichten erfolgt einmalig bei der ersten Kommunikation mit dem NTS-Server und ggf. bei Verbindungsfehlern bzw. ungültigen Zeitnachrichten. Die im Anschluss übertragenen Zeitnachrichten können dann vom Client periodisch angestoßen werden.

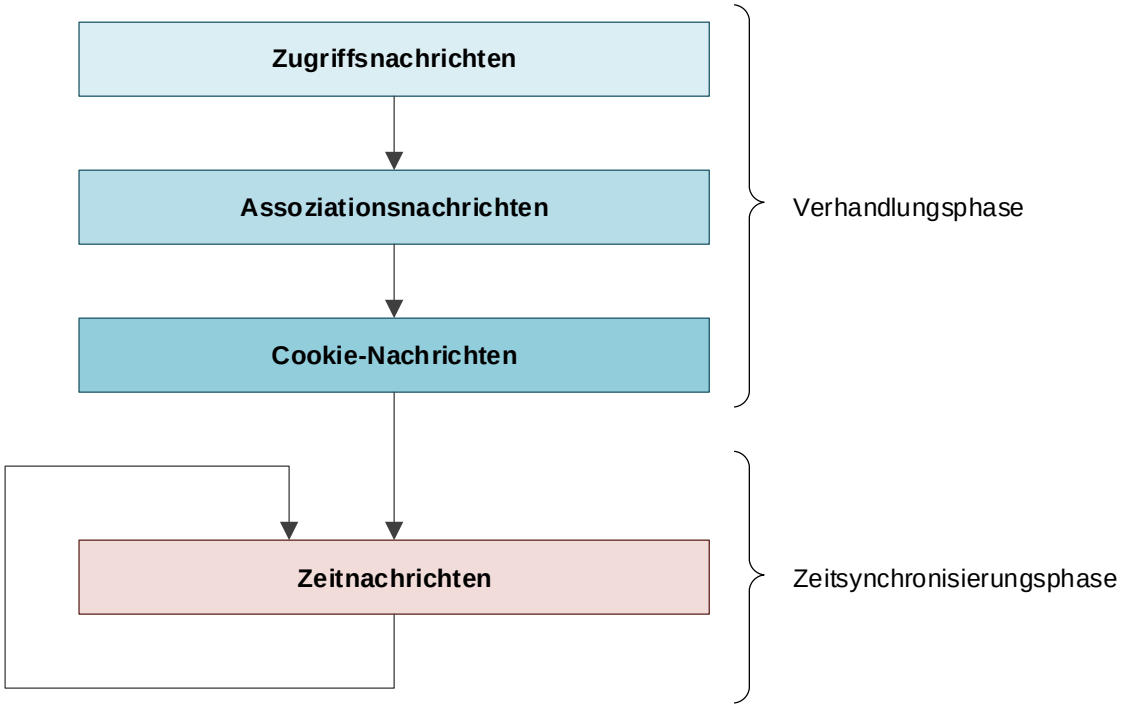


Abbildung 3.8: Untergliederung der NTS-Verhandlungsphase

[18] Nähere Details dazu werden im Kapitel 3.6.2 erläutert

3.3.1 Zugriffsnachrichten

Die Zugriffsnachrichten stellen in NTS die erste Verbindungsanfrage des Clients an den Server dar und initiieren den Verhandlungsprozess. Die Verwendung dieser Nachrichten bieten dem Server einen Schutz vor sogenannten Verstärkungsangriffen (engl. *Amplification Attack*), bei denen kleine Anfragen des Clients große Serverantworten provozieren (s.a. Kapitel 3.1.2). Dies trifft beispielsweise bei den Assoziationsnachrichten zu, bei denen die Serverantwort leicht um das Zehnfache (oder höher) im Verhältnis zur Clientanfrage anwächst. Ein Verstärkungsangriff zählt zu den DoS-Attacken, bei denen zeitgleich viele Anfragen an den Server gesendet werden, um eine Überlastung zu bewirken. Ein derartiger Angriff würde somit eine große Auslastung der verfügbaren Netzwerkbandbreite bedeuten und möglicherweise den Zeitdienst betriebsunfähig machen (s.a. Kapitel 3.1.2). Da Zugriffsnachrichten wesentlich kleiner sind und der Größenunterschied zwischen Anfrage- und Antwortnachrichten gering ist, können derartige Angriffe abgewehrt oder das Angriffspotential zumindest minimiert werden.

3.3.1.1 Die *client_access*-Nachricht

Die *client_access*-Nachricht wird vom Client generiert und anschließend an den Server übertragen. Die Struktur der Nachricht entspricht dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ClientAccessData* (s.a. Kapitel 3.4). Diese Nachricht enthält keine weiteren Informationen und teilt ausschließlich dem Server mit, dass der Client zukünftig einen Austausch der Assoziationsnachrichten wünscht.

3.3.1.2 Die *server_access*-Nachricht

Die *server_access*-Nachricht wird vom Server generiert und anschließend an den Client übertragen. Die Struktur dieser Nachricht entspricht auch dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ServerAccessData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet einen Zugriffsschlüssel, den sogenannten *Access Key*. Dieser berechtigt den anfragenden Client, eine Assoziationsnachricht zu senden. Der Zugriffsschlüssel ist an den Client gebunden und wird vom Server gemäß der folgenden Gleichung 3.7 berechnet:

$$Access\ Key\ =\ MSB_ < B_{AK} >\ \big(MAC(Server\ Seed,\ \ IP\ des\ Clients)\big) \qquad (3.7)$$

MSB_<B_AK> () schneidet die höchstwertigsten Bits des Ergebnisses der MAC-Funktion ***MAC()*** ab, wobei B_{AK} die Anzahl der Bits repräsentiert. Zum jetzigen Stand der NTS-Spe-

zifikation hat der Zugriffsschlüssel eine Länge von 128 Bit. Die für die Berechnung benötigte MAC-Funktion kann der Server selbst festlegen, da diese nicht mit dem Client ausgehandelt wird. Die „*IP des Clients*“ stellt die Netzwerkadresse des Clients dar und kann sowohl die IPv4, als auch die IPv6-Adresse sein. Die Festlegung geschieht über die Netzwerkverbindung, die von dem NTP-Protokoll genutzt wird. Die Gültigkeit des Zugriffsschlüssels wird somit auf die IP-Adresse des Clients und die Lebensdauer des *Server Seed* begrenzt.

3.3.2 Assoziationsnachrichten

Die Assoziationsnachrichten dienen der Aushandlung von Parametern zwischen dem Client und dem Server. Dazu zählen die NTS-Version, die MAC- und die Verschlüsselungsalgorithmen. Diese werden nach der Aushandlung durchgehend im Protokoll angewendet. Zusätzlich erhält der Client das Serverzertifikat, mit dem alle vom Server übertragenen Nachrichten auf deren Integrität hin überprüft werden können. Durch den Aufbau einer Zertifizierungskette ist es dem Client zudem möglich, die Authentizität des Servers zu überprüfen. Dadurch stellt der Client sicher, dass alle empfangenen Nachrichten von einem zugelassenen Zeitserver stammen und auf der unverfälschten Anfrage basieren.

3.3.2.1 Die *client_assoc*-Nachricht

Die *client_assoc*-Nachricht wird vom Client generiert und anschließend an den Server übertragen. Die Struktur der Nachricht entspricht dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ClientAssocData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- den Zugriffsschlüssel
- eine Nonce
- die minimal unterstützte NTS-Version[19]
- eine Liste mit unterstützten MAC-Algorithmen
- eine Liste mit unterstützten asymmetrischen Verschlüsselungsalgorithmen
- eine Liste mit unterstützten symmetrischen Verschlüsselungsalgorithmen

Der zuvor empfangene Zugriffsschlüssel aus der *server_access*-Nachricht wird wieder in der *client_assoc*-Nachricht eingebunden, wodurch der Server später den Zugriff durch den Client prüfen kann. Bei der Nonce handelt es sich um eine vom Client generierte Zufallszahl,

[19] Die Aushandlung der NTS-Version beschreibt das Kapitel 3.6.1 detailliert, da Client und Server jeweils nur eine Versionsgrenze kommunizieren.

die derzeit eine Länge von 128 Bit hat. Diese wird bei jeder Anfrage neu generiert und muss bis zum Eintreffen der Antwortnachricht des Servers gespeichert werden. Der Server sendet in der Antwortnachricht die Nonce zurück, wodurch der Client über Vergleichsoperatoren die gespeicherte Nonce mit der empfangenen Nonce prüfen kann. Sind diese identisch, so liefert es dem Client den Beweis, dass die Antwortnachricht des Servers auf der Anfrage des Clients beruht. Da die Nonce bei jeder Anfrage neu generiert wird, bietet sie zusätzlich Schutz vor sogenannten *replay*-Angriffen. Bei diesen versucht der Angreifer, durch aufgezeichnete Nachrichten die Identität des Zeitservers vorzutäuschen (s.a. Kapitel 3.1.2). Bei der NTS-Version handelt es sich um die minimal unterstützte NTS-Version, die vom Client akzeptiert wird. Der Server muss mindestens diese oder eine höhere NTS-Version unterstützen. Die Algorithmen, die der Client in die Nachricht einbettet, können anschließend vom Server frei ausgewählt werden, wobei aus jeder Liste genau ein Algorithmus gewählt werden muss. Der MAC-Algorithmus wird für die Berechnung des KIV, des Cookies und des MAC für die NTP-Nachricht benötigt. Der symmetrische Verschlüsselungsalgorithmus kommt im weiteren Kommunikationsablauf bei der Verschlüsselung des Cookies zum Einsatz. Der während dieser Phase erzeugte symmetrische Schlüssel wird anschließend asymmetrisch mit dem festgelegten asymmetrischen Verschlüsselungsalgorithmus verschlüsselt (s.a. Kapitel 3.5.3).

3.3.2.2 Die *server_assoc*-Nachricht

Die *server_assoc*-Nachricht wird vom Server generiert und anschließend an den Client übertragen. Die Struktur der Nachricht entspricht dem *NTS-Signed*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ServerAssocData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- die empfangene Nonce
- die vorgeschlagene NTS-Version
- die empfangene Liste mit den MAC-Algorithmen
- den ausgewählten MAC-Algorithmus
- die empfangene Liste mit den asymmetrischen Verschlüsselungsalgorithmen
- den ausgewählten asymmetrischen Verschlüsselungsalgorithmus
- die empfangene Liste mit den symmetrischen Verschlüsselungsalgorithmen
- den ausgewählten symmetrischen Verschlüsselungsalgorithmus
- die Zertifikatskette[20] des Servers
- die digitale Signatur der NTS-Nachricht

[20] Besteht aus dem primären Zertifikat des Servers und allen zwischengeschalteten CA-Zertifikaten, mit denen sich eine Zertifizierungskette zur Überprüfung aufbauen lässt (s.a. Kapitel 2.4.3).

Die in der Antwortnachricht hinzugefügte Nonce ist die empfangene und unveränderte Nonce aus der *client_assoc*-Nachricht. Die vom Server vorgeschlagene NTS-Version sollte die höchste Version sein, die der Server unterstützt. Dabei muss diese mindestens so hoch wie die vom Client übertragene NTS-Version sein. Die Listen mit den kryptografischen Algorithmen werden zunächst unverändert in die Antwortnachricht kopiert. Dies gibt dem Client die Möglichkeit, eventuelle Manipulationen an den Listen zu erkennen, da die *client_assoc*-Nachricht ungeschützt an den Server übertragen wird und durch einen Angreifer verändert werden könnte. Der Server wählt nun aus den Listen jeweils einen Algorithmus aus, der von Client und Server unterstützt wird. Hierbei ist der Server bestrebt, die jeweils sichersten Algorithmen auszuwählen (s.a. Kapitel 3.6.5). Damit der Client die Echtheit des Servers prüfen kann, wird das Serverzertifikat zur Nachricht hinzugefügt. Dies beinhaltet auch alle Zertifikate, die zum Aufbau einer Zertifikatskette hin zur Zertifizierungsstelle (*Root CA*) benötigt werden. Um die Integrität der Antwortnachricht zu gewährleisten, wird das eingebettete NTS-Nachrichtenobjekt (s.a. Kapitel 3.4) – in dem die Parameter gespeichert sind – mit dem privaten Schlüssel des Servers und dem ausgewählten MAC-Algorithmus[21] digital signiert. Kommt es bei der Auswahl der Parameter zu einem Fehler, weil diese nicht vom Server unterstützt werden, wird die Verarbeitung der Nachricht abgebrochen und ein Fehlercode an den Client übertragen.

3.3.3 Cookie-Nachrichten

Die Cookie-Nachrichten dienen im NTS-Protokoll dem Austausch des symmetrischen Schlüssels (dem Cookie), mit dessen Hilfe der Integritätsschutz der zukünftigen Zeitnachrichten gewährleistet wird. Außerdem ermöglichen diese Nachrichten dem Server, die Autorität des Clients zu überprüfen, sofern dieser es wünscht. Dadurch könnten beispielsweise kostenpflichtige NTS-Dienste realisiert werden, den nur zugelassene NTS-Clients nutzen dürfen. Des Weiteren wird mit dem im Clientzertifikat enthaltenen öffentlichen Schlüssel das serverseitig generierte Cookie verschlüsselt. Die Cookie-Nachrichten garantieren somit, dass das ausgetauschte Cookie auf die originale und unveränderte Anfrage des Clients zurückgeht und nicht von Dritten ausgelesen oder verändert wurde.

[21] Der verwendete MAC-Algorithmus muss momentan ein HMAC-Algorithmus sein, da für die digitale Signatur und die spätere Berechnung des KIV ein Hash-Algorithmus benötigt wird (s.a. Kapitel 3.6.3 und Kapitel 6.2).

3.3.3.1 Die *client_cook*-Nachricht

Die *client_cook*-Nachricht wird vom Client generiert und anschließend an den Server übertragen. Die Struktur der Nachricht entspricht dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ClientCookieData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- die verwendete NTS-Version
- eine Nonce
- den ausgehandelten Signaturalgorithmus[22]
- den ausgehandelten MAC-Algorithmus
- den ausgehandelten asymmetrischen Verschlüsselungsalgorithmus
- den ausgehandelten symmetrischen Verschlüsselungsalgorithmus
- die Zertifikatskette des Clients

Die ausgehandelte NTS-Version ist in der Nachricht implizit vorhanden und wird nicht als explizite Information übertragen (siehe Kapitel 3.4.1). Die Nonce ist eine neue, vom Client generierte Zufallszahl, die so wie in der *client_assoc*-Nachricht gespeichert und nach Erhalt der Serverantwort mit der empfangenen Nonce verglichen wird. Die zuvor ausgehandelten Algorithmen werden der Nachricht hinzugefügt und serverseitig zur Generierung und Verschlüsselung des Cookies sowie zur Erzeugung der digitalen Signatur verwendet. Das beigefügte Clientzertifikat (bzw. die Zertifikatskette) werden zur Berechnung des Cookies und zur optionalen Autoritätsüberprüfung beim Server benötigt.

3.3.3.2 Die *server_cook*-Nachricht

Die *server_cook*-Nachricht wird vom Server generiert und anschließend an den Client übertragen. Die Struktur der Nachricht entspricht dem *NTS-Encrypted-and-Signed*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *ServerCookieData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- die verwendete NTS-Version
- die empfangene Nonce
- das berechnete Cookie

[22] Ist derzeit immer der Signaturalgorithmus aus dem Serverzertifikat (s.a. Kapitel 6.2)

Die ausgehandelte NTS-Version ist in der Nachricht implizit vorhanden und wird nicht als explizite Information übertragen (siehe Kapitel 3.4.1). Die in der Antwortnachricht hinzugefügte Nonce ist die empfangene und unveränderte Nonce aus der *client_cook*-Nachricht. Das Cookie wird mit Hilfe der ausgehandelten Parameter und des Clientzertifikats berechnet und mit dem öffentlichen Schlüssel des Clients verschlüsselt. Die Entschlüsselung des Cookies ist nur noch für den Client möglich, der im Besitz des privaten Schlüssels ist. Um die Nachricht vor Veränderungen zu schützen, wird diese durch den Server digital signiert. Treten bei der Generierung des Cookies oder bei der Verarbeitung der Anfrage Fehler auf, so kann der Server einen Fehlercode an den Client senden (s.a. Kapitel 3.6.6).

3.3.4 Zeitnachrichten

Die Zeitnachrichten werden im NTS-Protokoll zur Synchronisierung der lokalen Uhrzeit des Clients benötigt. Diese Anfragen kann der Client so lange an den Server senden, wie das von ihm übertragene Cookie gültig ist. Dessen Lebensdauer wird hierbei durch den *Server Seed* begrenzt, welcher periodisch vom Server aktualisiert wird. Die Nachrichten, die zwischen dem Client und Server ausgetauscht werden, sind durch einen MAC geschützt. Dieser garantiert den Integritätsschutz der Nachrichten und beglaubigt die Echtheit der beiden Kommunikationsteilnehmer. Dabei ist es wichtig, dass die durch NTS getätigten Prozesse so schnell sind, dass die zeitliche Synchronisierung des Clients nicht signifikant beeinträchtigt wird und somit der Sinn von NTS-geschütztem NTP *ad absurdum* geführt wird. Die Übertragung dieser Nachrichten erfolgt ausschließlich über ein geeignetes Zeitsynchronisierungsprotokoll wie NTP oder PTP.

3.3.4.1 Die *time_request*-Nachricht

Die *time_request*-Nachricht wird vom Client generiert und anschließend an den Server übertragen. Die Struktur der Nachricht entspricht dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *TimeRequestSecurityData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- die verwendete NTS-Version
- eine Nonce
- den ausgehandelten MAC-Algorithmus
- den *Key Input Value* (KIV)
- den MAC

Die ausgehandelte NTS-Version ist in der Nachricht implizit vorhanden und wird nicht als explizite Information übertragen (siehe Kapitel 3.4.1). Die Nonce ist eine neue, vom Client generierte Zufallszahl, die so wie in der *client_assoc*- und *client_cook*-Nachricht gespeichert und nach Erhalt der Serverantwort mit der empfangenen Nonce verglichen wird. Der zuvor ausgehandelte MAC-Algorithmus wird serverseitig für die Generierung des Cookies und des MAC benötigt. Der *Key Input Value* ist der Hash-Wert des Clientzertifikats und wird vom Server für die Berechnung des Cookies benötigt. Der für den KIV angewendete Hash-Algorithmus leitet sich vom MAC-Algorithmus ab. Die für die Nachricht verwendete NTS-Version ist implizit enthalten und wird über das NTP-Erweiterungsfeld ermittelt. Nach der Generierung des NTP-Pakets mit der enthaltenen NTS-Nachricht, wird zum Schutz der Daten ein MAC generiert und an das NTP-Paket anhängt. Das Hinzufügen des MAC ist clientseitig gemäß der NTS-Spezifikation optional, jedoch bei der Verwendung des NTP-Zeitsynchronisierungsprotokoll (wie in diesem Falle) Pflicht. Die Berechnung des MAC erfolgt über das komplette NTP-Paket mit dem ausgehandelten MAC-Algorithmus und dem Cookie, welches zuvor mit der *server_cook*-Nachricht empfangen wurde.

3.3.4.2 Die *time_response*-Nachricht

Die *time_response*-Nachricht wird vom Server generiert und anschließend an den Client übertragen. Die Struktur der Nachricht entspricht dem *NTS-Plain*-Archetyp mit dem eingebetteten NTS-Nachrichtenobjekt *TimeResponseSecurityData* (s.a. Kapitel 3.4). Diese Nachricht beinhaltet folgende Elemente:

- die verwendete NTS-Version
- die empfangene Nonce
- den generierten MAC

Die ausgehandelte NTS-Version ist in der Nachricht implizit vorhanden und wird nicht als explizite Information übertragen (siehe Kapitel 3.4.1). Die in der Antwortnachricht hinzugefügte Nonce ist die empfangene und unveränderte Nonce aus der *time_request*-Nachricht. Nach erfolgreicher Überprüfung der *time_request*-Nachricht generiert der Server ein NTP-Paket und integriert die NTS-Nachricht. Das zur Berechnung des MAC benötigte Cookie errechnet der Server mit dem KIV des Clients, dem ausgehandelten MAC-Algorithmus und dem *Server Seed*, welchen nur der Server kennt und geheim bleibt. Mit diesem Cookie kann der Server nun den MAC über das ganze NTP-Paket generieren, der schließlich an diesen angehängt und an den Client übertragen wird. Dieser kann nach Empfang den MAC durch sein eigenes Cookie prüfen und bei Erfolg die Zeitdaten zur Synchronisierung der lokalen Uhrzeit auf NTP-Ebene freigeben.

3.4 NTS-Nachrichtenstruktur

Jede erzeugte NTS-Nachricht benötigt ein Zeitsynchronisierungsprotokoll als Träger. Dieses wird durch NTS geschützt und dient zusätzlich zur Übertragung der benötigten Daten während der Verhandlungsphase. Auch wenn die Übermittlung der Parameter gemäß der NTS-Spezifikation über einen externen Kanal realisiert werden dürfen, so muss das Zeitsynchronisierungsprotokoll dennoch in der Lage sein, diese Übertragung selber zu tätigen. Zusätzlich muss die volle Kompatibilität gewährleistet werden, sodass auch Teilnehmer, die kein NTS unterstützen, die entsprechenden Daten ohne Nutzung der Schutzfunktion verarbeiten können. Bei der Verwendung des NTPv4-Protokolls kommen die sogenannten Erweiterungsfelder zum Einsatz (vgl. Abbildung 3.9). Diese werden genutzt, um die NTS-Nachrichten und den durch NTS generierten MAC zu transportieren. Das von NTP bereitgestellte MAC-Feld wird nicht verwendet, da es nicht die Anforderungen der NTS-Spezifikation erfüllt[23].

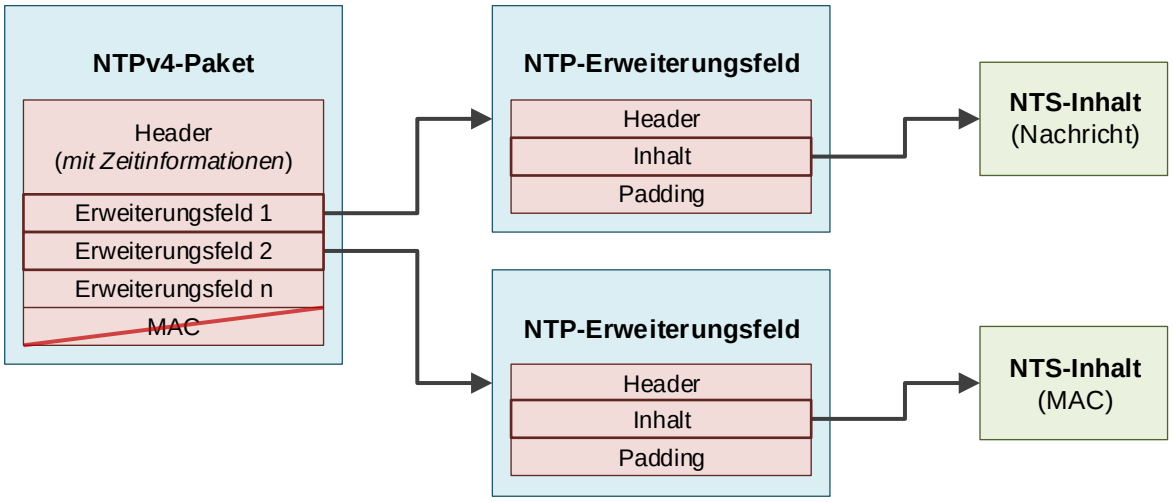


Abbildung 3.9: Struktureller Aufbau von NTPv4-Paketen

Die eingebetteten NTS-Inhalte und die Anzahl der benötigten Erweiterungsfelder sind von der aktuellen Phase der Kommunikation abhängig. Die NTS-Inhalte stellen in diesem Kontext entweder eine NTS-Nachricht oder einen von NTS generierten MAC dar. Je nach verwendeter Nachricht sind diese zudem unterschiedlich strukturiert, wobei bestimmte Grundstrukturen immer vorhanden sind (vgl. Abbildung 3.10). Dies ist beispielsweise das *NTSExtensionFieldContent*-Objekt, welches den sogenannten NTS-Archetyp (s.a. Kapitel 3.4.2) und mögliche Fehler kommuniziert. Der im Objekt enthaltene Archetyp beschreibt hierbei

[23] Das von NTP gelieferte MAC-Feld legt eine feste Größe anhand des verwendeten Algorithmus (MD5 oder SHA1) zugrunde [83].

den Aufbau der enthaltenen Struktur. Diese kann das NTS-Nachrichtenobjekt sein, welches die eigentlichen Daten zwischen den Kommunikationspartnern transportiert oder auch eine CMS-Struktur in der das NTS-Nachrichtenobjekt eingebettet wird. Die Beschreibung der einzelnen Elemente erfolgt in den folgenden Unterkapiteln 3.4.1 bis 3.4.4.

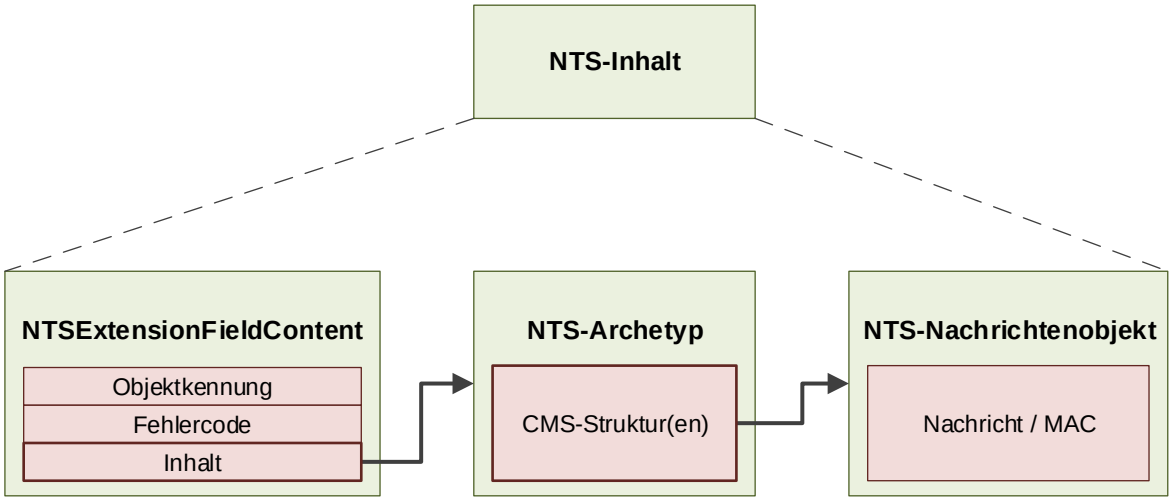


Abbildung 3.10: Struktureller Aufbau von NTS-Inhalten

3.4.1 Die NTPv4-Erweiterungsfelder

Die in NTPv4 vorhandenen Erweiterungsfelder (engl. *extension fields*) sind optionale Felder, in denen zusätzliche Informationen hinzugefügt und zwischen Client und Server ausgetauscht werden können. Dies ermöglicht die Erweiterung des NTP-Protokolls, ohne eine Änderung an der Spezifikation vornehmen zu müssen. Die dabei verwendete Anzahl von Erweiterungsfeldern ist beliebig und wird nur von der Größe des verwendeten Transportprotokolls begrenzt (s.a. Kapitel 6.1.2). Die Definition der Erweiterungsfelder erfolgt im RFC 5905 (*Network Time Protocol Version 4*) [18] und wurde durch das RFC 7822 (*NTP Extension Fields*) [88] aktualisiert. Zum jetzigen Zeitpunkt werden diese Erweiterungsfelder nur durch das *Autokey Protocol* [83] und das *UDP Checksum Complement* [89] verwendet. In Bezug auf NTS enthalten die Erweiterungsfelder jeweils eine Anfrage- oder Antwortnachricht. Wenn der Empfänger ein Erweiterungsfeld mit einer unbekannten Kennung empfängt, dann sollte er dieses Erweiterungsfeld ignorieren und kann auch das komplette NTP-Paket verwerfen, sofern es gefordert wird.

3.4.1.1 Aufbau der Erweiterungsfelder

Alle in NTPv4 verwendeten Erweiterungsfelder folgen dem gleichen Aufbau entsprechend der Abbildung 3.11. Die Erweiterungsfelder befinden sich bei NTPv4 immer zwischen den festen Feldern des Headers, in denen sich die Zeitinformationen befinden und dem NTP-MAC, wobei letzterer optional und somit nicht zwingend erforderlich ist.

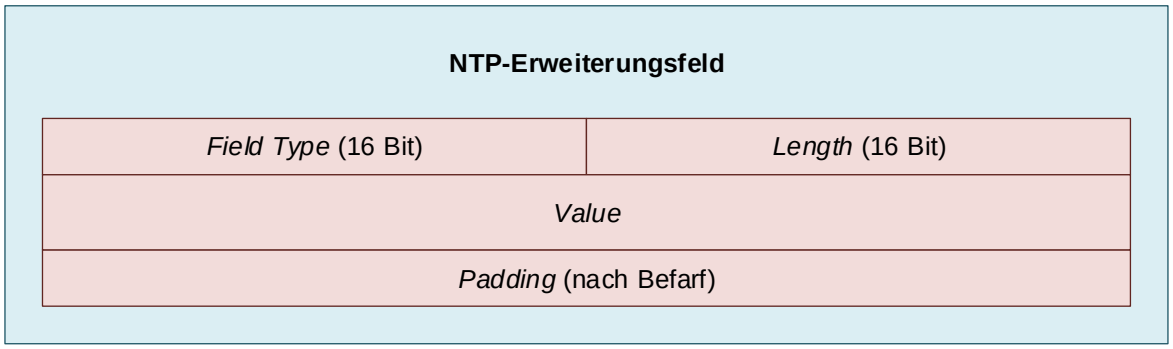


Abbildung 3.11: Struktureller Aufbau der NTPv4-Erweiterungsfelder

Das Erweiterungsfeld selbst besteht bei NTP aus einem Header, dem Inhalt und einem *Padding*. Der Header beinhaltet die beiden Elemente *Field Type* und *Length*. Der *Field Type* ist hierbei ein 16 Bit langer vorzeichenloser Wert (Integer), der den Inhalt des *Value*-Feldes beschreibt. Die Definition der möglichen Feldtypen wird durch die IANA geregelt [88, p. 4]. Das *Length*-Feld ist ebenfalls ein 16 Bit langer vorzeichenloser Wert (Integer), der die Größe des *Value*-Feldes angibt. Das *Value*-Feld stellt den Inhalt dar und unterliegt vom Aufbau her keiner weiteren Regeln, die durch NTP vorgegeben werden. Das abschließende *Padding* wird bei Erweiterungsfeldern nur benötigt, wenn das Erweiterungsfeld als Ganzes die Mindestgröße unterschreitet. In diesem Falle wird die Größendifferenz durch das *Padding* aufgefüllt. Die dafür erforderliche Mindestgröße ist von der Konstellation der Erweiterungsfelder und dem NTP-MAC abhängig. Wenn das NTP-Paket nur ein Erweiterungsfeld beinhaltet, so muss die Länge des Erweiterungsfeldes mindestens 28 Byte betragen. Sind mehrere Erweiterungsfelder vorhanden, so muss jedes Erweiterungsfeld mindestens 16 Byte und das letzte Erweiterungsfeld mindestens 28 Byte groß sein [88, p. 6].

3.4.1.2 Verwendung der Erweiterungsfelder in NTS

Um den Austausch der Parameter und den Integritätsschutz der NTP-Nachrichten zu realisieren, nutzt NTS diese Erweiterungsfelder. Dazu ist es nötig, bei der IANA neue Kennungen zu registrieren, die zukünftig auf einen NTS bezogenen Inhalt in den Erweiterungsfeldern schließen lassen. Die folgende Tabelle 3.3 listet die zu registrierenden Kennungen auf:

Tabelle 3.3: Klassifizierung von Kennungen für NTS-Nachrichten

***Field Type*-Wert**	**Enthaltene NTS-Nachricht im *Value*-Feld**
TBD1[24]	`ClientAccessData, ServerAccessData`
TBD1	`ClientAssocData, ServerAssocData`
TBD1	`ClientCookieData, ServerCookieData`
TBD2	`TimeRequestSecurityData, TimeResponseSecurityData`
TBD3	`NTSMessageAuthenticationCode`

Die Kennungen der NTS-Nachrichten sind zum jetzigen Zeitpunkt nicht existent und müssen noch festgelegt werden. Dafür sind drei neue Kennungen vorgesehen. TBD1 definiert Erweiterungsfelder, die NTS-Nachrichten enthalten und während der Verhandlungsphase zwischen dem Client und Server ausgetauscht werden. TBD2 definiert Erweiterungsfelder, die NTS-Nachrichten enthalten und während der Zeitsynchronisierungsphase ausgetauscht werden. TBD3 definiert die Erweiterungsfelder, die einen NTS-MAC enthalten und auch während der Zeitsynchronisierungsphase ausgetauscht werden. Diese werden dann als zusätzliches Erweiterungsfeld in das NTP-Paket eingebettet. Die verwendeten Kennungen geben außerdem Aufschluss über die verwendete NTS-Version und müssen beim Parsen berücksichtigt werden. Die Kennungen TBD1 bis TBD3 werden zunächst nur als Version 1 vorliegen, wodurch die Festlegung der NTS-Version in der Verhandlungsphase und die Verarbeitung aller NTS-Nachrichten auf Version 1 beruhen.

Die Einbettung der NTS-Inhalte in das NTP-Paket unterliegt einer festen Reihenfolge. Währen in der Verhandlungsphase nur ein Erweiterungsfeld mit NTS-Inhalten (die NTS-Nachricht) in das NTP-Paket eingebettet wird, so sind es in der Zeitsynchronisierungsphase zwei (NTS-Nachricht und MAC). Da der MAC erst nach dem Einfügen der NTS-Nachricht in das NTP-Paket berechnet wird, muss der NTS-MAC immer der NTS-Nachricht als Erweiterungsfeld folgen und sollte das letzte Erweiterungsfeld des NTP-Paketes sein. Jeder NTS-MAC muss über den kompletten NTP-Header und alle vorgeschalteten Erweiterungsfelder bis zum MAC-tragenden Erweiterungsfeld generiert werden. Konstellationen, in denen der MAC vor der NTS-Nachricht steht, sind prinzipiell ungültig, da der Schutzmechanismus nicht gegeben ist. Es ist allerdings möglich, weitere Erweiterungsfelder hinzuzufügen. Dabei würden alle Inhalte, die vor dem Erweiterungsfeld mit dem NTS-MAC liegen, durch den Integritätsschutz abgedeckt. Erweiterungsfelder hinter dem MAC blieben jedoch ungeschützt, da diese nicht in die Berechnung des MAC einfließen. Dies wird durch die folgende Abbildung 3.12 noch mal verdeutlicht:

[24] TBD: To Be Defined

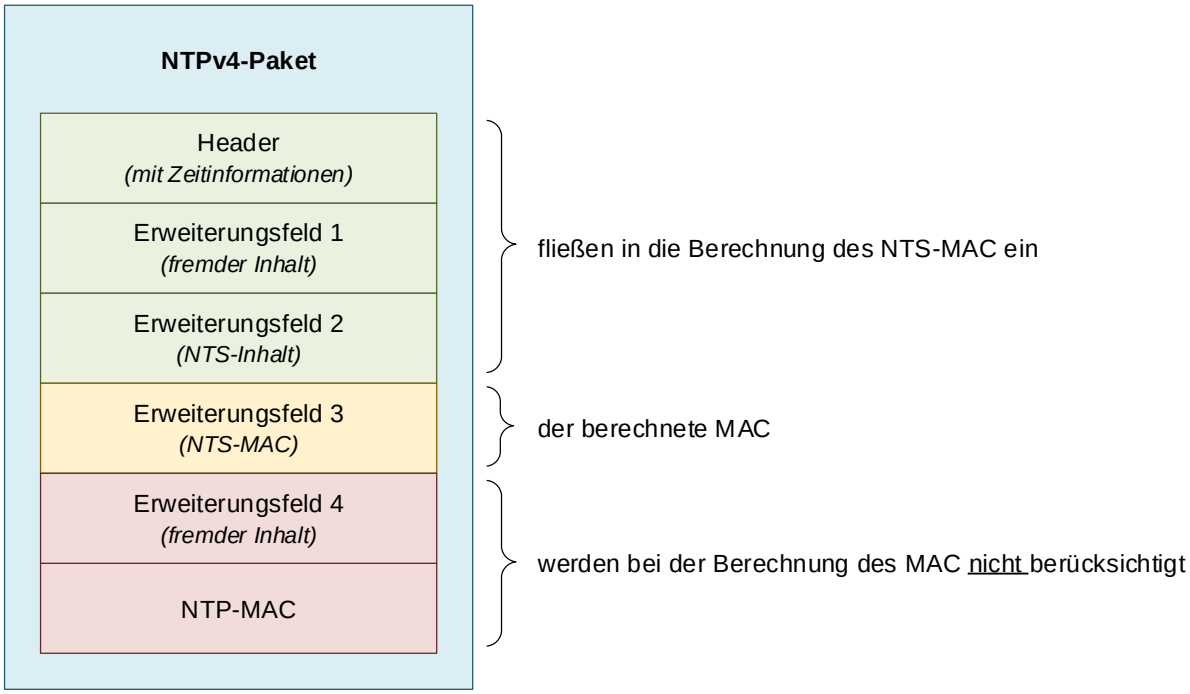


Abbildung 3.12: Berücksichtigte NTPv4-Felder zur Generierung der NTS-MAC

Wird ein NTP-Paket empfangen, welches ein Erweiterungsfeld mit einer NTS-Kennung (TBD1, TBD2, TBD3) im *Field Type* beinhaltet, so wird es auf immer gleiche Weise geparst. Dies ist unabhängig davon, ob es sich um eine NTS-Nachricht oder den MAC handelt. Das im *Value*-Feld befindliche Objekt ist somit bei allen NTS-Inhalten strukturmäßig identisch. Es handelt sich dabei um ein ASN.1-Objekt vom Typ *NTSExtensionFieldContent*, das wie folgt aufgebaut ist:

Listing 3.1: ASN.1-Beschreibung der NTS-Struktur "NTSExtensionFieldContent"

```
NTSExtensionFieldContent ::= SEQUENCE {
      oid       OBJECT IDENTIFIER,
      errnum    OCTET STRING (SIZE(2)),
      content   ANY DEFINED BY oid
}
```

Das *oid*-Element (*Object Identifier*) stellt die Kennung des Inhaltes dar und gibt Aufschluss über den verwendeten Archetyp. Die Kennung entspricht immer einer NTS-Nachricht (z.B. *server_assoc*) oder dem NTS-MAC, auch dann wenn der eingebettete Inhalt eine CMS-Struktur ist und somit eine andere Kennung (z.B. *id-contentInfo*) zu erwarten wäre. Die NTS-Kennungen (OIDs) sind zum aktuellen Zeitpunkt noch nicht vorhanden und müssen noch definiert werden.

Das Element *errnum* kann vom Server verwendet werden, um den Client über Fehler zu informieren. Dieser Wert muss null sein, wenn die Generierung der Serverantwort auf eine Clientanfrage erfolgreich war. Andernfalls muss dieser Wert ungleich null sein. Die aufgetretenen Fehler werden dabei bitweise gesetzt und im Kapitel 3.6.6 näher erläutert. Die Auswertung dieses Elements ist optional und dient clientseitig hauptsächlich zu Diagnosezwecken. Das Setzen eines Fehlers ist zudem nur serverseitig sinnvoll.

Der *content* stellt den eigentlichen Inhalt (NTS-Nachricht oder MAC) dar. Dabei handelt es sich um ein BER- oder DER-kodiertes ASN.1-Objekt. Die enthaltene Struktur, die sich hinter diesem Objekt befindet (der Archetyp), wird von der Kennung (*oid*) definiert.

3.4.2 Archetypen

Die Archetypen repräsentieren in NTS den strukturellen Aufbau der Inhalte, die in das *NTS-ExtensionFieldContent*-Objekt eingebettet werden. Diese bestehen aus einer Verschachtelung von CMS-Strukturen und dem NTS-Nachrichtenobjekt. Abhängig von der verwendeten Struktur ist die Verschlüsselung oder Signierung der eingebetteten Daten möglich. Es ist nicht notwendig, dass NTS-Nachrichtenobjekte, die sich im Kern jeder Struktur befinden, alle benötigten Daten für den einzelnen Nachrichtentyp beinhalten. Diese können auch in den CMS-Strukturen transportiert werden, sofern es die Anforderungen der NTS-Spezifikation erfüllt. Derzeit kommen die drei Archetypen *NTS-Plain*, *NTS-Signed* und *NTS-Encrypted-and-Signed* zum Einsatz, die nachfolgend näher erläutert werden.

3.4.2.1 NTS-Plain

Bei *NTS-Plain* handelt es sich um den einfachsten Archetyp, der keine CMS-Strukturen beinhaltet und somit weder eine Signatur trägt, noch eine Verschlüsselung der Daten erlaubt. Dieser Archetyp stellt somit das reine NTS-Nachrichtenobjekt dar, welches als BER- oder DER-kodierter ASN.1-Datenstrom in Klartext übertragen wird (vgl. Abbildung 3.13).

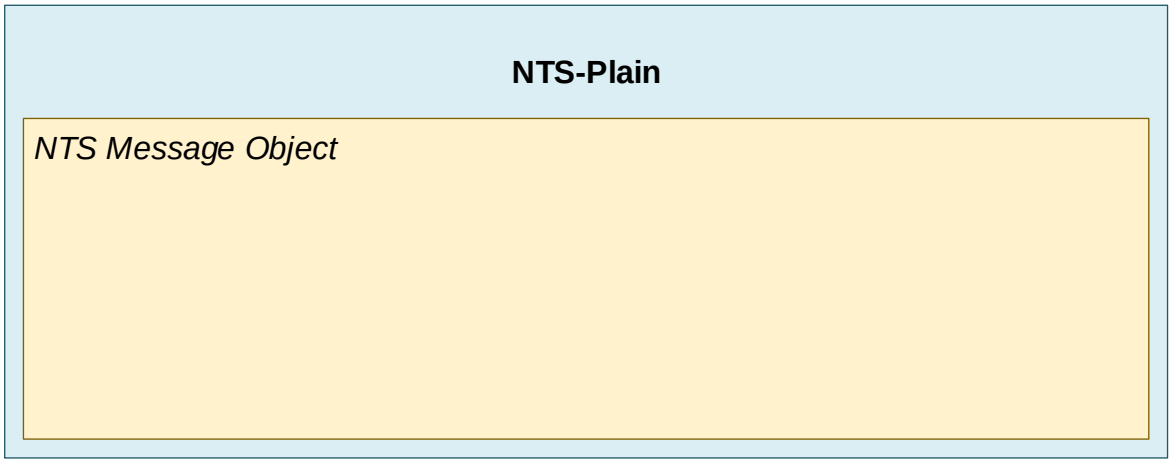


Abbildung 3.13: Aufbau des NTS-Plain-Archetyps

NTS-Plain wird von folgenden NTS-Inhalten verwendet:

- *client_access*-Nachricht
- *server_access*-Nachricht
- *client_assoc*-Nachricht
- *client_cook*-Nachricht
- *time_request*-Nachricht
- *time_response*-Nachricht
- NTS-MAC

3.4.2.2 NTS-Signed

Der *NTS-Signed*-Archetyp wird ausschließlich für die *server_assoc*-Nachricht verwendet und ermöglicht die digitale Signatur der enthaltenen Daten. Dieser Archetyp setzt sich aus der CMS-Struktur *Signed Data* (s.a. Kapitel 3.4.3) und dem darin enthaltenen NTS-Nachrichtenobjekt zusammen (vgl. Abbildung 3.14). Die verwendete CMS-Struktur *Signed Data* wird in die *Content Info*-Struktur eingebettet und ermöglicht beim Parsen die Identifizierung der enthaltenen CMS-Strukturen. Neben der digitalen Signatur verwendet NTS diese Struktur auch zur Übertragung von Zertifikaten und Zertifikatssperrlisten. Die Verwendung dieses Archetyps garantiert dem Client, dass die darin befindlichen Daten nicht durch Dritte manipuliert wurden.

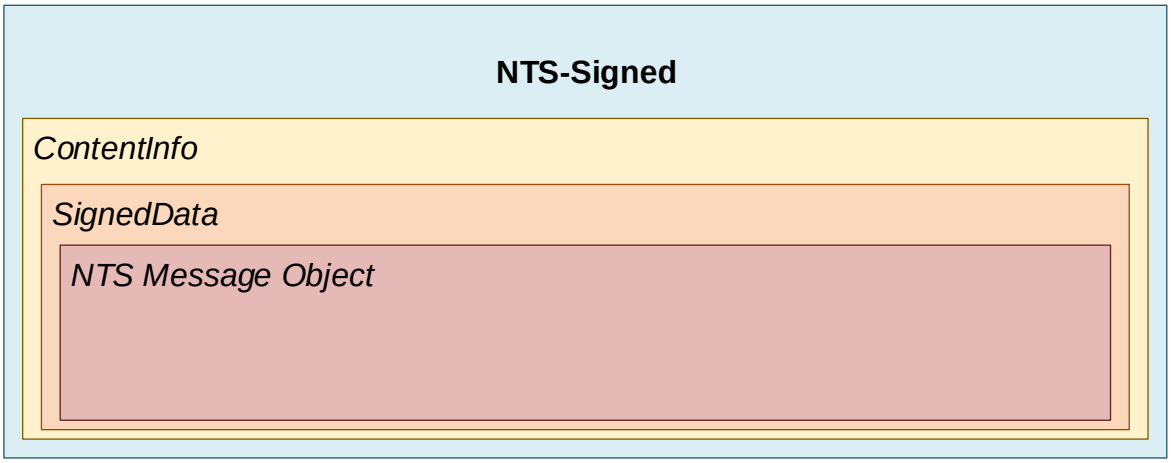


Abbildung 3.14: Aufbau des NTS-Signed-Archetyps

3.4.2.3 NTS-Encrypted-and-Signed

Der *NTS-Encrypted-and-Signed*-Archetyp erlaubt wie *NTS-Signed* die digitale Signierung der enthaltenen Daten und realisiert so den Integritätsschutz der Nachricht. Darüber hinaus ermöglicht dieser Archetyp die Verschlüsselung der eingebetteten Inhalte und verhindert somit das Mitlesen der Informationen durch Dritte. Dazu kommt sowohl die symmetrische, als auch die asymmetrische Verschlüsselung zum Einsatz, die im folgenden Unterkapitel 3.4.3 näher erläutert werden. Realisiert wird dies durch das Hinzufügen der CMS-Struktur *Enveloped Data*. Diese wird in die CMS-Struktur *Signed Data* eingebettet und beinhaltet das zu übertragene NTS-Nachrichtenobjekt (vgl. Abbildung 3.15). Dieser Archetyp wird serverseitig für die *server_cook*-Nachricht benötigt, um das geheime Cookie an den Client zu übermitteln.

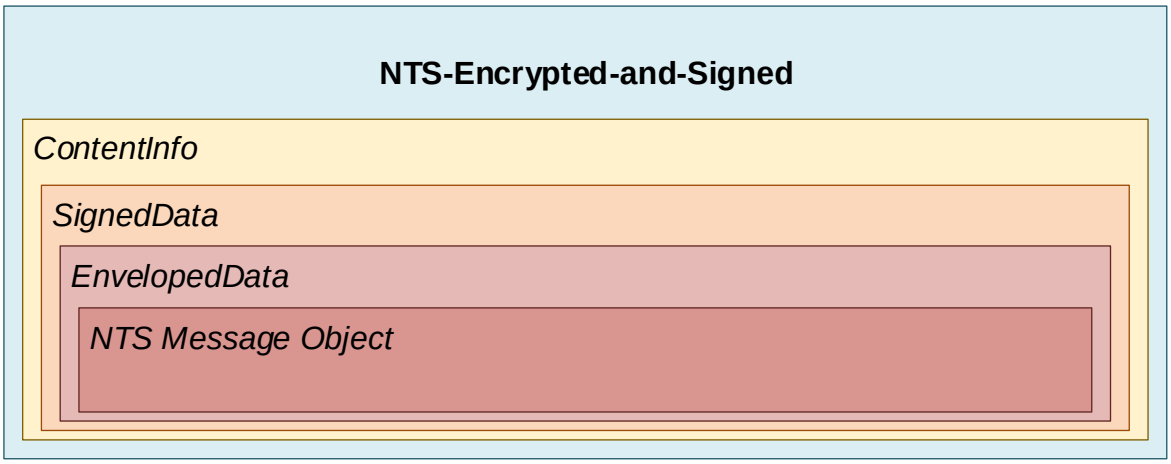


Abbildung 3.15: Aufbau des NTS-Encrypted-ans-Signed-Archetyps

3.4.3 CMS-Strukturen

Die in den Archetypen verwendeten CMS-Strukturen *ContentInfo*, *SignedData* und *EnvelopedData* müssen bestimmte Werte und Eigenschaften beinhalten, um mit der NTS-Spezifikation konform zu sein. Die wesentlichen CMS-Elemente, die für NTS von Bedeutung sind, werden in den folgenden Abschnitten näher erläutert. Die detaillierten Inhalte mit denen die CMS-Strukturen komplett beschrieben werden, können aus den Spezifikationen [47] [49] [52] entnommen werden.

3.4.3.1 ContentInfo

Bei *ContentInfo* handelt es sich um eine grundlegende CMS-Struktur, die als äußere Schicht einer CMS-Struktur betrachtet werden kann (vgl. Abbildung 3.16). Dabei spielt es keine Rolle, ob es sich dabei um eine einzelne oder eine verschachtelte CMS-Struktur handelt.

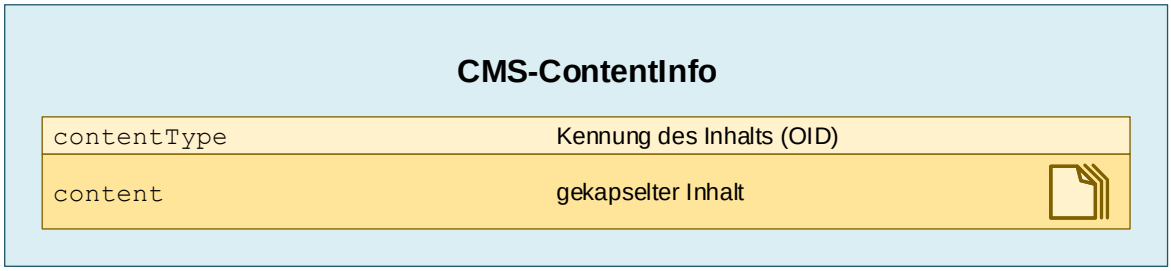


Abbildung 3.16: Aufbau der CMS-Struktur: ContentInfo

Die *ContentInfo*-Struktur beinhaltet nur zwei Felder. Der *contentType* beinhaltet eine Objektkennung in Form einer OID. Diese gibt Aufschluss über den im *content*-Feld befindlichen Inhalt. Der *content* stellt den Inhalt selbst dar, der in Form eines BER- oder DER-kodierten ASN.1-Datenstroms eingebettet wird. Einschränkungen bezüglich Inhalt und Größe der Daten gibt es nicht. Im Falle von NTS wird bei Verwendung dieser CMS-Struktur[25] immer eine *id-signedData*-OID als Kennung und eine DER-kodierte *SignedData*-Struktur als Inhalt erwartet.

[25] Nur in den Archetypen *NTS-Signed* und *NTS-Encrypted-and-Signed*

3.4.3.2 SignedData

Die CMS-Struktur *SignedData* dient der digitalen Signatur der enthaltenen Daten. Diese Struktur unterstützt fast jede Form der digitalen Signatur und ermöglicht mehreren Unterzeichnern[26] die enthaltenen Daten mit einem eigenen Signaturalgorithmus zu signieren. Im NTS-Protokoll muss an dieser Stelle genau ein Unterzeichner (der NTS-Server) vorhanden sein. Die Signatur beschränkt sich dabei nicht nur auf das eingebettete Objekt, sondern erstreckt sich über die gesamte *SignedData*-Struktur (mit Ausnahme weniger Parameter). Der Inhalt der Struktur wird in Abbildung 3.17 dargestellt und im Folgenden näher erläutert.

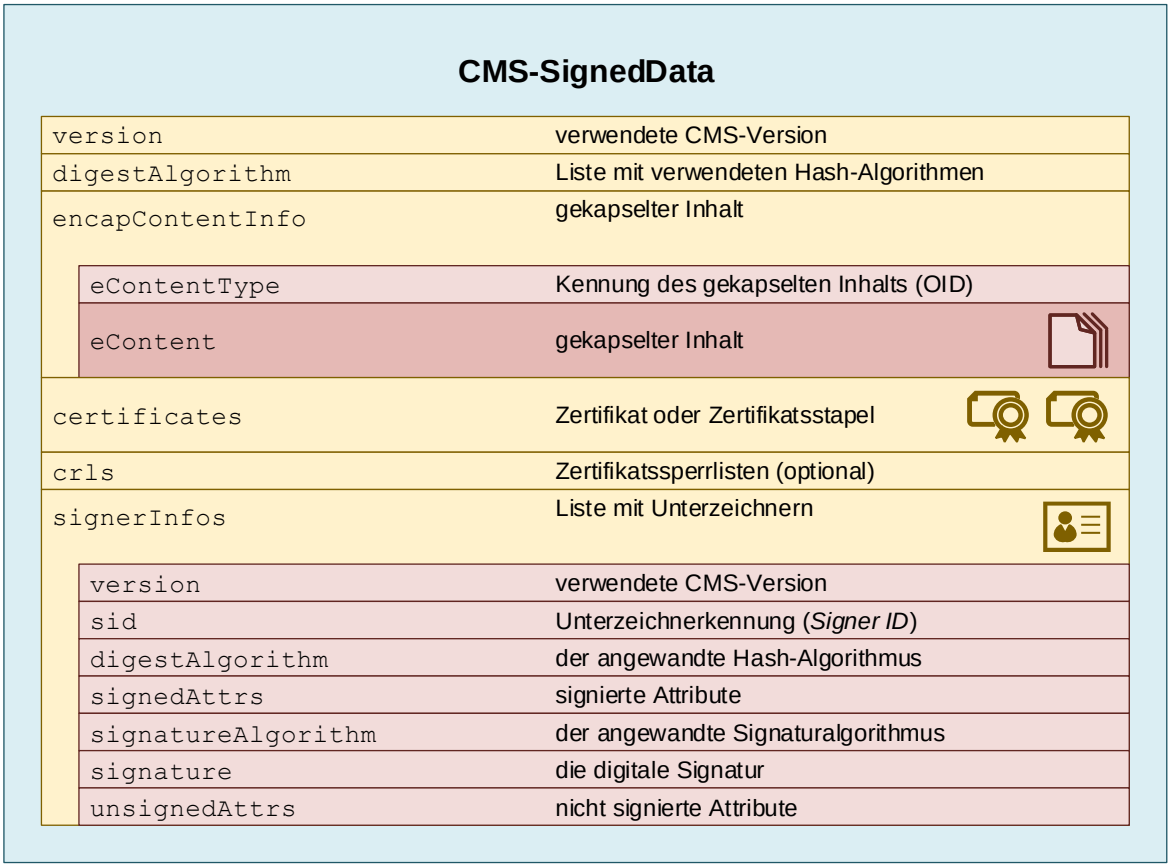


Abbildung 3.17: Aufbau der CMS-Struktur: SignedData

Element: version

Der Eintrag *version* definiert die verwendete CMS-Version in der SignedData-Struktur. Die Wahl der Version ist von dem Inhalt abhängig und wird in der CMS-Spezifikation [49, p. 10] genau definiert.

[26] Ein Unterzeichner meint in diesem Kontext eine Instanz, die mit Hilfe eines Zertifikats die enthaltenen Daten der *SignedData* digital signiert.

Element: digestAlgorithms

Der Eintrag *digestAlgorithms* ist eine Liste von Hash-Algorithmen, die in der jeweiligen *SignedData*-Struktur zur Anwendung kommen. Die Anzahl und Art der enthaltenen Algorithmen werden aus den Signaturalgorithmen der jeweiligen Unterzeichner abgeleitet. Da im NTS-Protokoll genau ein Unterzeichner vorhanden sein darf, beinhaltet die Liste auch genau einen Hash-Algorithmus. Dieser wird in der Assoziationsphase von NTS ausgehandelt und entspricht dem Element *choiceMacAlgo* aus der *server_assoc*-Nachricht (s.a. Kapitel 3.4.4).

Element: encapContentInfo

Das Element *encapContentInfo* (*encapsulated content information*) ist eine Unterstruktur und beinhaltet die gekapselten Nutzdaten, die durch die digitale Signatur geschützt werden. Im NTS-Protokoll muss diese Struktur immer vorhanden sein.

Element: encapContentInfo → eContentType

Der *eContentType* definiert den im *eContent* befindlichen Inhalt und wird in Form einer OID-Kennung gespeichert. Bei der Verwendung des *NTS-Signed*-Archetyps, muss er den Typ der gekapselten NTS-Nachricht identifizieren. Beim Einsatz des *NTS-Encrypted-and-Signed*-Archetyps muss es die Kennung der CMS-Struktur *EnvelopedData* beinhalten.

Element: encapContentInfo → eContent

Das Element *eContent* stellt die Nutzlast dar, die in *SignedData* eingebettet wird. Bei der Verwendung des *NTS-Signed*-Archetyps, muss es das DER-kodierte NTS-Nachrichtenobjekt enthalten. Beim Einsatz des *NTS-Encrypted-and-Signed*-Archetyps muss er die CMS-Struktur *EnvelopedData* ausweisen.

Element: certificates

Das Element *certificates* ist eine Sammlung von DER-kodierten Zertifikaten, die zur Signatur der eingebetteten Daten und zur Authentifizierung des Unterzeichners benötigt werden. Im NTS-Protokoll muss es bei Verwendung des *NTS-Signed*-Archetyps das primäre Zertifikat des Servers beinhalten. Zusätzlich muss der Server sämtliche Zertifikate hinzufügen, die

zum Aufbau der Zertifizierungskette bis zur Zertifizierungsstelle (der Root CA) benötigt werden. Dies ermöglicht dem Client die Überprüfung des Servers, ohne das OCSP-Protokoll explizit nutzen zu müssen. Zudem können auf diesem Wege die Zertifikate direkt transportiert und clientseitig gespeichert werden. Wird stattdessen der *NTS-Encrypted-and-Signed*-Archetyp verwendet (wie in der *server_cook*-Nachricht), so sollte dieses Feld keine Zertifikate beinhalten. In diesem Falle besitzt der Client bereits die Zertifikate des Servers und kann dadurch die Signatur der CMS-Struktur prüfen. Dem ist hinzuzufügen, dass sich dadurch die Größe der *SignedData*-Struktur enorm verringert, da die Zertifikate den wesentlichen Teil der NTS-Nachricht ausmachen.

Element: crls

Die *crls* (*certificate revocation list*) sind DER-kodierte Zertifikatssperrlisten, die der Unterzeichner in die *SignedData*-Struktur einbinden kann, um veraltete oder ungültige Serverzertifikate zu sperren. Dieser Eintrag ist optional und darf somit leer sein.

Element: signerInfos

Die Unterstruktur *signerInfos* ist eine Sammlung von Listen, die verschiedene Informationen über jeden Unterzeichner separat beinhalten. Die Anzahl dieser Listen entspricht der Anzahl der Unterzeichner. Im NTS-Protokoll muss genau ein Unterzeichner (der NTS-Server) vorhanden sein, wodurch auch nur eine *signerInfo*-Struktur vorhanden sein darf.

Element: signerInfos → version

Das Element *version* ist die Versionsnummer der verwendeten Syntax. Diese wird durch den Inhalt bestimmt und muss bei NTS die Version 3 beinhalten [49].

Element: signerInfos → sid

Der *sid* (*signer identifier*) identifiziert den Unterzeichner der jeweiligen *signerInfos*-Liste. Für die Identifizierung kann der *subject key identifier* (SKI) oder der Subjekt- und Ausstellername des zugehörigen Zertifikats verwendet werden. Der SKI wird meist über einen 160-

Bit-SHA-1-Hash[27] des öffentlichen Schlüssels gebildet, welcher sich im Zertifikat des Unterzeichners befindet. Im NTS-Protokoll wird stets der SKI anstelle des Subjekt- und Ausstellernamens verwendet, wodurch die Größe der *SignedData*-Struktur vermindert wird.

Element: signerInfos → digestAlgorithm

Der *digestAlgorithm* ist der verwendete Hash-Algorithmus mit allen damit verbunden Parametern, der für die Signatur angewandt wird. Im NTS-Protokoll muss der Inhalt dieses Feldes mit dem *digestAlgorithms*-Feld aus der *SignedData*-Struktur identisch sein.

Element: signerInfos → signedAttrs

Die *signedAttrs* ist eine Liste mit Attributen, die durch die digitale Signatur abgedeckt werden. Im NTS-Protokoll müssen die Attribute *Content Type* und *Message Digest* enthalten sein. Dabei muss der *Content Type* genau einen Wert enthalten und mit dem Inhalt des *eContentType*-Feldes in der *SignedData*-Unterstruktur *encapContentInfo* übereinstimmen. Der *Message Digest* ist der errechnete Hash-Wert, der im *eContent*-Feld der *encapContentInfo*-Unterstruktur befindlichen Daten. Der dafür verwendete Algorithmus befindet sich im *digestAlgorithm*-Feld der jeweiligen *signerInfo*-Struktur.

Zusätzlich dürfen im NTS-Protokoll die Attribute *Signing Time* und *Binary Signing Time* verwendet werden. Diese Werte geben den Zeitpunkt der Signatur an und können beispielsweise für Diagnosezwecke eingesetzt werden. Die Verwendung der Zeit zur lokalen Synchronisierung der Uhrzeit auf der Clientseite ist nicht erlaubt.

Element: signerInfos → signatureAlgorithm

Der *signatureAlgorithm* identifiziert den verwendeten Signaturalgorithmus und alle damit verbundenen Parameter, die vom Unterzeichner zur Generierung der digitalen Signatur verwendet werden. Der Algorithmus setzt sich dabei aus dem gewählten Hash-Algorithmus (im *digestAlgorithm*-Feld) und dem Schlüsseltyp des Unterzeichnerzertifikats zusammen (z.B. RSA oder ECC).

[27] Der genaue Aufbau und die alternativen Generierungsmöglichkeiten werden im RFC 5280 [97] beschrieben

Element: signerInfos → signature

Dieses Feld beinhaltet die digitale Signatur, die gemäß RFC 5652 [49] unter Verwendung des Signaturalgorithmus und dem privaten Schlüssel des Unterzeichners (dem NTS-Server) generiert wird. Mit Hilfe des öffentlichen Schlüssels, welcher sich im Serverzertifikat in der *SignedData*-Struktur befindet, kann der Client die Gültigkeit der Signatur überprüfen.

Element: signerInfos → unsignedAttrs

Die *unsignedAttrs* ist eine Liste mit optionalen Attributen, die nicht durch die Signatur abgedeckt werden. Im NTS-Protokoll dürfen solche Attribute nicht eingesetzt werden. Sind derartige Attribute in einer Nachricht enthalten, so müssen diese Attribute ignoriert werden. Alternativ ist die komplette Nachricht zu verwerfen.

3.4.3.3 EnvelopedData

Die *EnvelopedData*-Struktur unterstützt fast jede Form des Schlüsselmanagements und wird in NTS zur Übertragung von Schlüsseln, Schlüsselparametern und verschlüsselten Inhalten verwendet. Diese kommt ausschließlich in dem *NTS-Encrypted-and-Signed*-Archetypen zum Einsatz. Mit dieser Struktur kann der Server das generierte Cookie an den Client übertragen, ohne dass es Dritte mitlesen können. Die Verschlüsselung des Cookies erfolgt dabei in zwei Schritten. Zunächst wird beim Server eine kryptografisch sichere Zufallszahl gebildet, die einen symmetrischen Schlüssel repräsentiert. Das Cookie wird nun mit diesem Schlüssel und dem *contentEncryption*-Algorithmus symmetrisch verschlüsselt. Die zur Verschlüsselung benötigten Algorithmen werden in der Assoziationsphase zwischen Client und Server ausgehandelt und stehen dem Server im Nachrichtenobjekt der *client_cook*-Nachricht zur Verfügung. Im zweiten Schritt wird nun der zuvor generierte symmetrische Schlüssel asymmetrisch verschlüsselt. Dies erfolgt mit dem *keyEncryption*-Algorithmus und dem öffentlichen Schlüssel des Clients, welcher sich im Zertifikat der übertragenen *client_cook*-Nachricht befindet. Von nun an ist nur noch der Client in der Lage, den symmetrischen Schlüssel zur Entschlüsselung des Cookies durch Nutzung seines privaten Schlüssels zu entschlüsseln. Diese Methode nutzt somit den Geschwindigkeitsvorteil der symmetrischen Verschlüsselung, ohne dass der Schlüssel zuvor ausgetauscht werden muss. Der Inhalt der *EnvelopedData*-Struktur wird in Abbildung 3.18 dargestellt und im Folgenden näher erläutert.

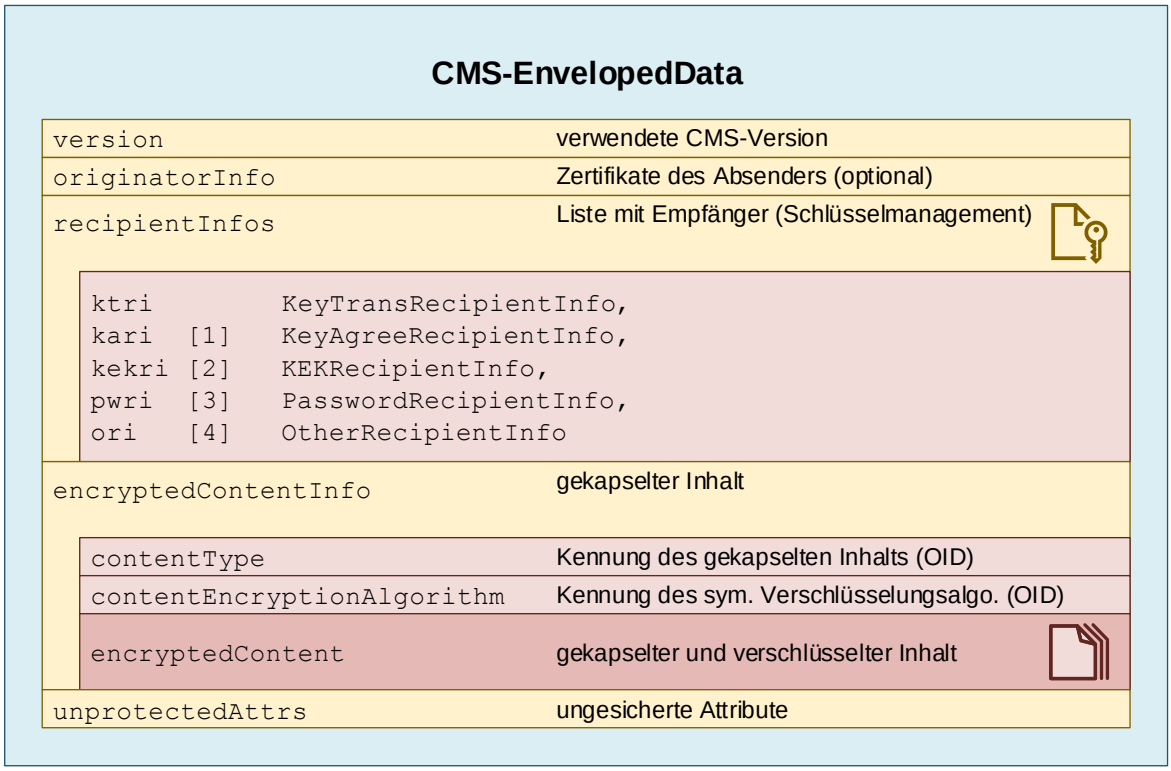


Abbildung 3.18: Aufbau der CMS-Struktur: EnvelopedData

Element: version

Der Eintrag *version* definiert die verwendete CMS-Version in der *EnvelopedData*-Struktur. Die verwendete Version ist von dem Inhalt abhängig und wird in der CMS-Spezifikation [49, p. 19] genau definiert.

Element: originatorInfo

Das *originatorInfo*-Feld beinhaltet eine Sammlung von Zertifikaten und Zertifikatssperrlisten des Absenders (dem NTS-Server). Diese Struktur ist nur vorhanden, wenn es vom Schlüsselmanagement-Algorithmus gefordert wird. Im NTS-Protokoll muss diese Struktur vorhanden sein, wenn der *key agreement algorithm* verwendet wird. Wird hingegen der *key transport algorithm*[28] verwendet, darf diese Struktur nicht verwendet werden. Unabhängig davon, ob das *originatorInfo*-Feld vorhanden sein muss, sollten keine Zertifikate oder Zertifikatssperrlisten eingebettet werden, da diese dem NTS-Client bereits vorliegen.

[28] Die Wahl des Schlüsselmanagement-Algorithmus erfolgt im *recipientInfos*-Feld

Element: recipientInfos

Das *recipientInfos*-Feld ist in *EnvelopedData* immer vorhanden. Es ist eine Liste mit Empfängern, die jeweils separate Informationen zum verschlüsselten Inhalt besitzen. Dazu zählt das verwendete Schlüsselmanagement, die Schlüssel und deren Parameter, die für jeden Empfänger individuell sind. Das verwendete Schlüsselmanagement wird hierbei durch den im Zertifikat befindlichen öffentlichen Schlüssel (z.B. RSA oder ECC) festgelegt. Das *recipientInfos*-Feld muss im NTS-Protokoll genau einen Eintrag enthalten und wird mit Hilfe des Clientzertifikats erzeugt.

Element: encryptedContentInfo

Das *encryptedContentInfo*-Feld ist eine Unterstruktur und beinhaltet Informationen zum verschlüsselten Inhalt. Dieses Feld muss immer vorhanden sein.

Element: encryptedContentInfo → contentType

Der *contentType* definiert den im *encryptedContent* befindlichen Inhalt in Form einer OID-Kennung. Bei der Verwendung des *NTS-Encrypted-and-Signed*-Archetyps, muss es den Typ der gekapselten NTS-Nachricht identifizieren.

Element: encryptedContentInfo → contentEncryptionAlgorithm

Dieses Element beinhaltet den *contentEncryption*-Algorithmus, mit dem der in *encryptedContent* gekapselten Inhalt symmetrisch verschlüsselt wurde. Im NTS-Protokoll muss dieser Algorithmus mit dem ausgehandelten Inhaltsverschlüsselungsalgorithmus (*contentEncryption*-Algorithmus) in der *client_cook*-Nachricht identisch sein.

Element: encryptedContentInfo → encryptedContent

Dieses Feld beinhaltet die symmetrisch verschlüsselte Nutzlast. Im NTS-Protokoll muss es das BER- oder DER-kodierte NTS-Nachrichtenobjekt beinhalten.

Element: encryptedContentInfo → unprotectedAttrs

Die *unprotectedAttrs* ist eine Liste mit optionalen Attributen, die durch keinen Mechanismus geschützt werden. Im NTS-Protokoll ist die Verwendung dieser Attribute nicht erlaubt. Sind derartige Attribute in einer Nachricht enthalten, so müssen diese Attribute ignoriert werden. Alternativ ist die komplette Nachricht zu verwerfen.

3.4.4 NTS-Nachrichtenobjekte

Die NTS-Nachrichtenobjekte stellen den Kern einer NTS-Nachricht dar und beinhalten die Parameter, die zwischen Client und Server ausgetauscht werden (s.a. Kapitel 3.3). Diese werden durch die Beschreibungssprache ASN.1 definiert und als BER- bzw. DER-kodierte Daten in den entsprechenden Archetyp der jeweiligen Nachricht eingebettet [52]. In diesem Abschnitt werden dazu die Inhalte der einzelnen Nachrichten erläutert. Die komplette Aufschlüsselung der ASN.1-Strukturen steht im Anlagendokument II[29] zur Verfügung. Die Kennungen der einzelnen NTS-Nachrichtenobjekte sind zum jetzigen Zeitpunkt noch nicht definiert und nutzen daher den Platzhalter TBD (*to be defined*).

3.4.4.1 ClientAccessData

Das *ClientAccessData*-Objekt stellt in NTS die einfachste ASN.1-Struktur dar und findet in der *client_access*-Nachricht ihren Einsatz. Diese Struktur überträgt keine Informationen und ist somit leer. Die Aufgabe der Struktur dient ausschließlich der Identifizierung der NTS-Nachricht. Der Server, der eine solche Nachricht erhält, generiert allein aus dieser Information die korrekte Antwort (die *server_access*-Nachricht). Das *ClientAccessData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.2: ASN.1-Beschreibung der NTS-Struktur "ClientAccessData"

```
ClientAccessData ::= NULL
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.3: Kennung der NTS-Struktur "ClientAccessData"

```
id-ct-nts-clientAccess OBJECT IDENTIFIER ::= TBD
```

[29] Anlage II: „ASN.1-Modulbeschreibungen in der NTS-Implementierung“

3.4.4.2 ServerAccessData

Das *ServerAccessData*-Objekt wird von der *server_access*-Nachricht verwendet, um den Zugriffsschlüssel an den Client zu übertragen. Das *ServerAccessData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.4: ASN.1-Beschreibung der NTS-Struktur "ServerAccessData"

```
ServerAccessData ::= SEQUENCE {
      accessKey         NTSAccessKey
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.5: Kennung der NTS-Struktur "ServerAccessData"

```
id-ct-nts-serverAccess OBJECT IDENTIFIER ::= TBD
```

Element: accessKey

Der *accessKey* spezifiziert den Zugriffsschlüssel, der zur Limitierung der Assoziationsanfragen des Clients benötigt wird und mögliche Verstärkungsangriffe abwehrt. Dieser wird serverseitig durch die IP-Adresse des Clients, den *Server Seed* und einem entsprechenden MAC-Algorithmus berechnet. Der *accessKey* ist vom Typ *NTSAccessKey*, der folgendermaßen definiert ist:

Listing 3.6: ASN.1-Beschreibung des NTS-Elements "NTSAccessKey"

```
NTSAccessKey ::= OCTET STRING (SIZE(16))
```

3.4.4.3 ClientAssocData

Das *ClientAssocData*-Objekt wird von der *client_assoc*-Nachricht verwendet und dient der Übertragung von Parametern und Algorithmen, die zwischen Client und Server ausgehandelt werden sollen. Das *ClientAssocData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.7: ASN.1-Beschreibung der NTS-Struktur "ClientAssocData"

```
ClientAssocData ::= SEQUENCE {
      accessKey          NTSAccessKey,
      nonce              NTSNonce,
      minVersion         NTSVersion,
      macAlgos           AlgorithmIdentifiers,
      keyEncAlgos        KeyEncryptionAlgorithmIdentifiers,
      contentEncAlgos    ContentEncryptionAlgorithmIdentifiers
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.8: Kennung der NTS-Struktur "ClientAssocData"

```
id-ct-nts-clientAssoc OBJECT IDENTIFIER ::= TBD
```

Element: accessKey

Der *accessKey* ist der Zugriffsschlüssel, der unverändert aus der zuvor empfangenen *server_access*-Nachricht eingetragen wird. Dieser wird serverseitig überprüft und erlaubt dem Client die Aushandlung der Parameter mit dem Server. Ist der Zugriffsschlüssel ungültig, so verwirft der Server die *client_assoc*-Nachricht.

Element: nonce

Die *nonce* ist eine vom Client erstellte 128 Bit lange Zufallszahl, die bei jeder Anfrage neu generiert wird. Diese wird vor der Übertragung der Nachricht gespeichert und nach Erhalt der entsprechenden Serverantwort mit der empfangenen Nonce verglichen. Sind die Werte identisch, so bestätigt es dem Client, dass die Antwortnachricht aus der zuvor übertragenen Anfrage erzeugt wurde. Ein weiter Zweck ist der Schutz vor *Replay*-Angriffen durch einen Angreifer. Die Nonce ist vom Typ *NTSNonce* und wird wie folgt definiert:

Listing 3.9: ASN.1-Beschreibung des NTS-Elements "NTSNonce"

```
NTSNonce ::= OCTET STRING (SIZE(16))
```

Element: minVersion

Die *minVersion* ist die minimale vom Client unterstützte NTS-Version, die zur Fortführung der weiteren Kommunikation benötigt wird. Dieser Wert wird durch eine Ganzzahl repräsentiert, deren Wert im Bereich 0 bis 255 liegt. Der eingetragene Wert entspricht hierbei der Versionsnummer (der Wert 1 entspricht also der Version 1)[30]. Die *NTSVersion* wird wie folgt definiert:

Listing 3.10: ASN.1-Beschreibung des NTS-Elements "NTSVersion"

```
NTSVersion ::= INTEGER (0..255)
```

Element: macAlgos

Das *macAlgos*-Element ist eine Liste mit MAC-Algorithmen, die der Client unterstützt. Diese werden später zur Berechnung des Cookies und des MAC benötigt und liegen als OID-Kennungen vor.

Element: keyEncAlgos

Das *keyEncAlgos*-Element ist eine Liste mit *keyEncryption*-Algorithmen, die vom Client unterstützt werden. Die Algorithmen werden im weiteren Protokollverlauf zur asymmetrischen Verschlüsselung eines symmetrischen Schlüssels in der CMS-Struktur benötigt.

Element: contentEncAlgos

Das *contentEncAlgos*-Element ist eine Liste mit *contentEncryption*-Algorithmen, die vom Client unterstützt werden. Die Algorithmen werden im weiteren Protokollverlauf zur symmetrischen Verschlüsselung des Cookies in der CMS-Struktur benötigt.

3.4.4.4 ServerAssocData

Das *ServerAssocData*-Objekt wird von der *server_assoc*-Nachricht verwendet. Diese enthält die Parameter, die aus der Anfragenachricht des Clients empfangen und vom Server ausgewählt wurden. Die ausgehandelten Parameter werden anschließend für die weitere Kommunikation genutzt. Das *ServerAssocData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

[30] Dies ist in der aktuellen Spezifikation jedoch nicht eindeutig festgelegt.

Listing 3.11: ASN.1-Beschreibung der NTS-Struktur "ServerAssocData"

```
ServerAssocData ::= SEQUENCE {
      nonce                   NTSNonce,
      proposedVersion         NTSVersion,
      macAlgos                AlgorithmIdentifiers,
      choiceMacAlgo           AlgorithmIdentifier,
      keyEncAlgos             KeyEncryptionAlgorithmIdentifiers,
      choiceKeyEncAlgo        KeyEncryptionAlgorithmIdentifier,
      contentEncAlgos         ContentEncryptionAlgorithmIdentifers,
      choiceContentEncAlgo    ContentEncryptionAlgorithmIdentifier
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.12: Kennung der NTS-Struktur "ServerAssocData"

```
id-ct-nts-serverAssoc OBJECT IDENTIFIER ::= TBD
```

Element: nonce

Die *nonce* ist der unveränderte Wert aus der zuvor empfangenen *client_assoc*-Nachricht.

Element: proposedVersion

Die *proposedVersion* ist die bevorzugte NTS-Version des Servers. Diese muss mindestens der *minVersion* aus der zuvor empfangenen *client_assoc*-Nachricht entsprechen und kann höher sein. Die vom Server eingetragene Version sollte die höchste Version sein, die serverseitig unterstützt wird (s.a. Kapitel 3.6.1).

Element: macAlgos

Das *macAlgos*-Element beinhaltet die unveränderte Liste mit den MAC-Algorithmen, die aus der *client_assoc*-Nachricht empfangen wurden.

Element: choiceMacAlgo

Das *choiceMacAlgo*-Element beinhaltet genau einen vom Server ausgewählten MAC-Algorithmus. Dieser Algorithmus muss vom Server unterstützt werden und sich auch in der

macAlgos-Liste befinden. Der Server sollte hierbei den sichersten Algorithmus auswählen, den sowohl Client als auch Server unterstützen.

Element: keyEncAlgos

Das *keyEncAlgos*-Element beinhaltet die unveränderte Liste mit den *keyEncryption*-Algorithmen, die aus der *client_assoc*-Nachricht empfangen wurden.

Element: choiceKeyEncAlgo

Das *choiceKeyEncAlgo*-Element beinhaltet genau einen vom Server ausgewählten *keyEncryption*-Algorithmus. Dieser Algorithmus muss vom Server unterstützt werden und sich auch in der *keyEncAlgos*-Liste befinden. Der Server sollte hierbei den sichersten Algorithmus auswählen, den sowohl Client als auch Server unterstützen.

Element: contentEncAlgos

Das *contentEncAlgos*-Element beinhaltet die unveränderte Liste mit den *contentEncryption*-Algorithmen, die aus der *client_assoc*-Nachricht empfangen wurden.

Element: choiceContentEncAlgo

Das *choiceContentEncAlgo*-Element beinhaltet genau einen vom Server ausgewählten *contentEncryption*-Algorithmus. Dieser Algorithmus muss vom Server unterstützt werden und sich auch in der *contentEncAlgos*-Liste befinden. Der Server sollte hierbei den sichersten Algorithmus auswählen, den sowohl Client als auch Server unterstützen.

3.4.4.5 ClientCookieData

Das *ClientCookieData*-Objekt wird in der *client_cook*-Nachricht verwendet und enthält die ausgehandelten Parameter, sowie die Zertifikate des Clients. Das *ClientCookieData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.13: ASN.1-Beschreibung der NTS-Struktur "ClientCookieData"

```
ClientCookieData ::= SEQUENCE {
      nonce             NTSNonce,
      signAlgo          SignatureAlgorithmIdentifier,
      macAlgo           AlgorithmIdentifier,
      encAlgo           ContentEncryptionAlgorithmIdentifier,
      keyEncAlgo        KeyEncryptionAlgorithmIdentifier,
      certificates      CertificateSet
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.14: Kennung der NTS-Struktur "ClientCookieData"

```
id-ct-nts-clientCookie OBJECT IDENTIFIER ::= TBD
```

Element: nonce

Die *nonce* ist eine neue vom Client generierte Zufallszahl, die gespeichert und nach Erhalt der entsprechenden Serverantwort mit der empfangenen Nonce verglichen wird. Sind die Werte identisch, so bestätigt es dem Client, dass die Antwortnachricht aus der zuvor übertragenen Anfrage erzeugt wurde.

Element: signAlgo

Der *signAlgo* ist der verwendete Signaturalgorithmus, der serverseitig zur digitalen Signatur der *server_cook*-Nachricht benötigt wird. Die eingetragene Kennung muss dem Signaturalgorithmus aus dem Serverzertifikat entsprechen, welcher zuvor in der *server_assoc*-Nachricht empfangen wurde.

Element: macAlgo

Der *macAlgo* ist der verwendete MAC-Algorithmus, der zwischen Server und Client in der Assoziationsphase ausgehandelt wurde. Der eingetragene Wert ist genau eine OID, die den Algorithmus mit allen damit verbundenen Parametern identifiziert und mit dem Inhalt der *choiceMacAlgo* aus dem bereits empfangenen *ServerAssocData*-Objekt identisch ist. Der Algorithmus wird serverseitig zur Generierung des Cookies benötigt.

Element: encAlgo

Der *encAlgo* ist der verwendete *contentEncryption*-Algorithmus, der zwischen Server und Client in der Assoziationsphase ausgehandelt wurde. Der eingetragene Wert ist genau eine OID, die den Algorithmus mit allen damit verbundenen Parametern identifiziert und mit dem Inhalt der *choiceContentEncAlgo* aus dem bereits empfangenen *ServerAssocData*-Objekt identisch ist. Der Algorithmus wird serverseitig zur symmetrischen Verschlüsselung des Cookies benötigt.

Element: keyEncAlgo

Der *keyEncAlgo* ist der verwendete *keyEncryption*-Algorithmus, der zwischen Server und Client in der Assoziationsphase ausgehandelt wurde. Der eingetragene Wert ist genau eine OID, die den Algorithmus mit allen damit verbundenen Parametern identifiziert und mit dem Inhalt der *choiceKeyEncAlgo* aus dem bereits empfangenen *ServerAssocData*-Objekt identisch ist. Der Algorithmus wird serverseitig zur asymmetrischen Verschlüsselung des symmetrischen Schlüssels benötigt.

Element: certificates

Das *certificates*-Element enthält eine Sammlung von Zertifikaten, die dem Server eine Autorisierungsprüfung des Clients durch Aufbau des Zertifizierungspfads ermöglichen. Verzichtet der Server auf eine Überprüfung, beinhaltet dieses Feld nur das primäre Clientzertifikat mit dem darin befindlichen öffentlichen Schlüssel.

3.4.4.6 ServerCookieData

Das *ServerCookieData*-Objekt wird in der *server_cook*-Nachricht verwendet und enthält eine Nonce sowie das generierte Cookie. Das *ServerCookieData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.15: ASN.1-Beschreibung der NTS-Struktur "ServerCookieData"

```
ServerCookieData ::= SEQUENCE {
      nonce       NTSNonce,
      cookie      OCTET STRING (SIZE(16))
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.16: Kennung der NTS-Struktur "ServerCookieData"

```
id-ct-nts-serverCookie OBJECT IDENTIFIER ::= TBD
```

Element: nonce

Die *nonce* ist der unveränderte Wer, aus der zuvor empfangenen *client_cook*-Nachricht.

Element: cookie

Das *cookie* ist der 128 Bit lange symmetrische Schlüssel[31], der für die clientseitige Überprüfung der zukünftigen Zeitnachrichten benötigt wird.

3.4.4.7 TimeRequestSecurityData

Das *TimeRequestSecurityData*-Objekt wird in der *time_request*-Nachricht verwendet. Dieses übermittelt dem Server die Parameter zur erneuten Berechnung des Cookies. Damit kann der MAC der empfangenen Nachricht überprüft und die Antwortnachricht durch Generierung einer neuen MAC gesichert werden. Das *TimeRequestSecurityData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.17: ASN.1-Beschreibung der NTS-Struktur "TimeRequestSecurityData"

```
TimeRequestSecurityData ::= SEQUENCE {
      nonce             NTSNonce,
      macAlgo           AlgorithmIdentifier,
      keyInputValue     OCTET STRING (SIZE(16))
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

[31] Das Cookie darf nicht mit dem symmetrischen Schlüssel in der CMS-Struktur *EnvelopedData* verwechselt werden, der zur symmetrischen Verschlüsselung des Cookies angewandt wird.

Listing 3.18: Kennung der NTS-Struktur "TimeRequestSecurityData"

```
id-ct-nts-securityDataReq OBJECT IDENTIFIER ::= TBD
```

Element: nonce

Die *nonce* ist eine neue, vom Client generierte Zufallszahl, die gespeichert und nach Erhalt der entsprechenden Serverantwort mit der empfangenen Nonce verglichen wird. Sind die Werte identisch, so bestätigt es dem Client, dass die Antwortnachricht aus der zuvor übertragenen Anfrage erzeugt wurde.

Element: macAlgo

Der *macAlgo* ist der verwendete MAC-Algorithmus, der zwischen Server und Client in der Assoziationsphase ausgehandelt wurde. Der Algorithmus wird serverseitig zur Überprüfung und Generierung der MAC und für die Berechnung des Cookies benötigt.

Element: keyInputValue

Der *keyInputValue* ist der 128 Bit lange Hash-Wert, der clientseitig mit dem ausgehandelten MAC-Algorithmus[32] und dem Clientzertifikat erzeugt wird.

3.4.4.8 TimeResponseSecurityData

Das *TimeResponseSecurityData*-Objekt wird in der *time_response*-Nachricht verwendet und enthält nur eine Nonce. Das *TimeResponseSecurityData*-Objekt wird in ASN.1-Syntax wie folgt definiert:

Listing 3.19: ASN.1-Beschreibung der NTS-Struktur "TimeResponseSecurityData"

```
TimeResponseSecurityData ::= SEQUENCE {
      nonce    NTSNonce
}
```

[32] Der für die Berechnung benötigte Hash-Algorithmus kann aus dem MAC-Algorithmus abgeleitet werden, sofern es sich um ein HMAC-Algorithmus handelt. Hier gibt es gesonderte Fälle (siehe Kapitel 6.2.1).

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.20: Kennung der NTS-Struktur "TimeResponseSecurityData"

```
id-ct-nts-securityDataResp OBJECT IDENTIFIER ::= TBD
```

Element: nonce

Die *nonce* ist der unveränderte Wert aus der zuvor empfangenen *time_request*-Nachricht.

3.4.4.9 NTSMessageAuthenticationCode

Das *NTSMessageAuthenticationCode*-Objekt wird in den Zeitnachrichten *time_request* und *time_response* verwendet und beinhaltet den NTS-MAC. Das *NTSMessageAuthentication-Code*-Objekt wird in das *NTSExtensionFieldContent*-Objekt eingebettet und als zweites Erweiterungsfeld in das NTP-Paket eingefügt. Die ASN.1-Syntax für das *NTSMessageAuthenticationCode*-Objekt wird wie folgt definiert:

Listing 3.21: ASN.1-Beschreibung der NTS-Struktur "NTSMessageAuthenticationCode"

```
NTSMessageAuthenticationCode ::= SEQUENCE {
      mac   OCTET STRING (SIZE(16))
}
```

Die Kennung des Nachrichtenobjekts lautet hierbei wie folgt:

Listing 3.22: Kennung der NTS-Struktur "NTSMessageAuthenticationCode"

```
id-ct-nts-ntsForNtpMessageAuthenticationCode OBJECT IDENTIFIER ::= TBD
```

Element: mac

Das *mac*-Element beinhaltet den 128 Bit langen MAC, der den Integritätsschutz der Zeitnachrichten gewährleistet. Der Inhalt berechnet sich durch den ausgehandelten MAC-Algorithmus aus dem Cookie und dem NTP-Paket, welches in serieller Form vorliegen muss.

3.5 Kommunikationsablauf im Unicast-Betrieb

Die Abfolge der Kommunikation zwischen NTS-Client und -Server erfolgt im Unicast-Betrieb in immer gleicher Weise. Diese beginnt clientseitig durch die Generierung der Zugriffsnachricht und deren Übertragung an den Server. Der dadurch angestoßene Informationsaustausch verläuft nun linear durch die Verarbeitung und Generierung der Nachrichten zwischen den Kommunikationspartnern ab. In der Zeitsynchronisierungsphase hängt die vom Client gesendete Nachricht von dessen aktuellem Zustand ab (vgl. Abbildung 3.19). Werden Zeitnachrichten erfolgreich vom Client empfangen und überprüft, so ist die Synchronisierung der lokalen Uhrzeit durch das NTP-Protokoll möglich. Bei Verbindungsfehlern oder ungültigen Parametern, werden vom Server übertragene Pakete verworfen und neue Anfragen generiert. Ist hingegen der MAC einer empfangenen Nachricht ungültig, so springt der Client in die Verhandlungsphase zurück und fordert ein neues Cookie an. Der genauere Ablauf der Unicast-Kommunikation wird den folgenden Abschnitten näher erläutert.

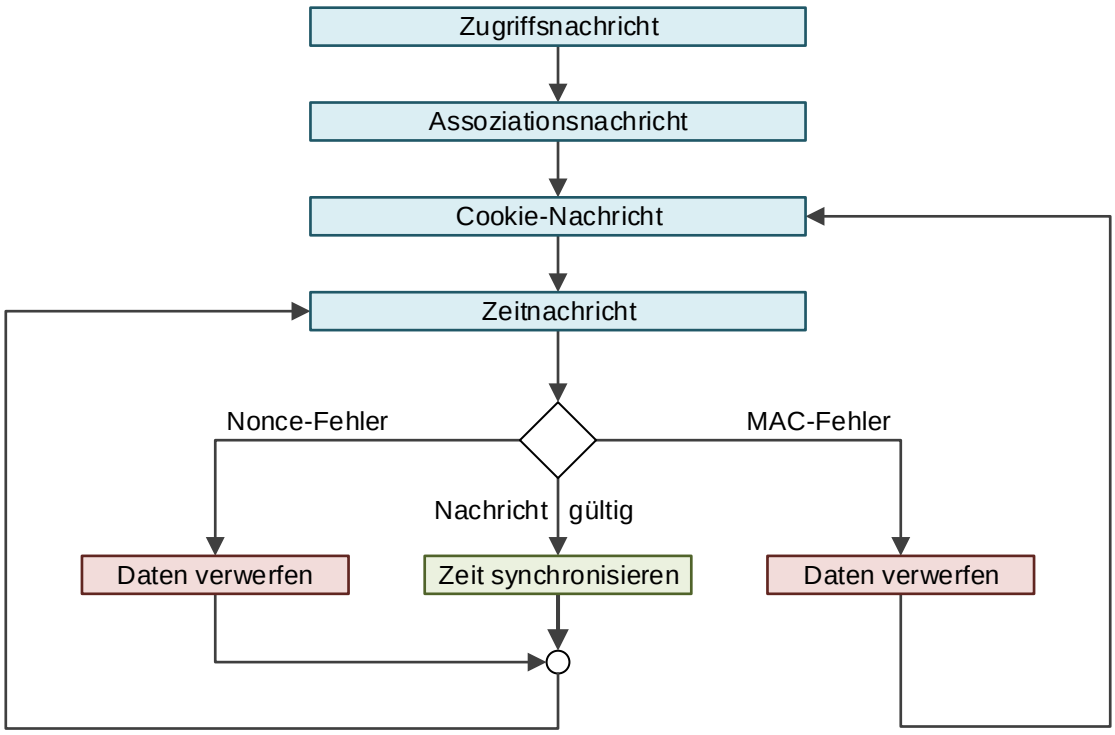


Abbildung 3.19: Allgemeiner Kommunikationsablauf des NTS-Protokolls

3.5.1 Zugriffsphase

Clientseitig

Der Anstoß der Kommunikation erfolgt clientseitig durch das Trägerprotokoll, welches bei NTP durch das sogenannte *Polling*[33] erfolgt. Dieses veranlasst bei NTS die Generierung einer Nachricht auf Basis des aktuellen Zustands. Bei der ersten Kontaktaufnahme zum NTS-Server wird dies durch die *client_access*-Nachricht realisiert. Diese wird entsprechend dem *NTS-Plain*-Archetyp strukturiert und beinhaltet das *ClientAccessData*-Nachrichtenobjekt als Kern. Die Nachricht enthält außer der Nachrichtenkennung keine weiteren Informationen. Diese wird nun in das NTP-Paket eingebettet und an den Server übertragen.

Serverseitig

Nach Erhalt der *client_access*-Nachricht konstruiert der Server einen Zugriffsschlüssel (den *Access Key*), der sich aus dem MAC-Algorithmus, dem *Server Seed* und der IP-Adresse des Clients entsprechend der Gleichung 3.8 zusammensetzt.

$$Access\ Key\ =\ MSB_{128}\ (MAC(Server\ Seed,\ \ IP\ des\ Clients)) \tag{3.8}$$

Der MAC-Algorithmus wird hierbei vom Server festgelegt und nicht mit dem Client ausgehandelt. Die Wahl des Algorithmus muss den Sicherheitsrichtlinien von NTS entsprechen, wird aber ansonsten nicht weiter eingeschränkt. Wichtig ist jedoch, dass für jeden Client immer der gleiche Algorithmus verwendet wird, um eine spätere Überprüfung möglich zu machen. Der *Server Seed* ist eine vom Server generierte und geheim gehaltene Zufallszahl. Diese ist an den jeweiligen MAC-Algorithmus gebunden und dient als Schlüssel für die Berechnung. Die benötigte IP-Adresse wird durch das NTP-Protokoll zur Verfügung gestellt und bindet den Zugriffsschlüssel an genau den anfragenden Client. Die zu verwendende Version der IP-Adresse (IPv4 oder IPv6) ist nicht festgelegt, muss aber mindestens während der Zugriffsphase gleichbleiben. Nach abgeschlossener Berechnung wird das Ergebnis auf 128 Bit gekürzt, wobei nur die höherwertigsten Bits erhalten bleiben.

Der Server generiert nun eine *server_access*-Nachricht, gemäß dem *NTS-Plain*-Archetyp mit dem *ServerAccessData*-Nachrichtenobjekt als Kern, in welchem der generierte Zugriffsschlüssel eingebettet wird. Anschließend wird die NTS-Nachricht über ein NTP-Paket an den anfragenden Client übertragen.

[33] Das *Polling* ist in NTP ein Abfragezyklus, der zwischen Client und Server in festen Intervallen geschieht.

Clientseitig

Nach Erhalt der Serverantwort extrahiert und speichert der Client den empfangen Zugriffsschlüssel, um die folgenden Anfragen generieren zu können. Das NTP-Paket, welches durch den Zeitserver schon Zeitdaten beinhaltet, wird nun verworfen. Dazu teilt NTS dem NTP-Protokoll über einen geeigneten Mechanismus mit, dass die empfangenen Zeitdaten nicht verwendet werden dürfen. Dieser Mechanismus ist von der jeweiligen Implementierung und den genutzten Schnittstellen zwischen NTP und NTS abhängig. Nach Abschluss dieser Prozedur wechselt der Client in die Assoziationsphase.

3.5.2 Assoziationsphase

Clientseitig

In dieser Phase findet zwischen dem Client und dem Server die Aushandlung der Parameter statt, die für alle zukünftigen Nachrichten verwendet werden sollen. Dazu generiert der Client eine *client_assoc*-Nachricht, die nach dem *NTS-Plain*-Archetyp aufgebaut ist und das *ClientAssocData*-Nachrichtenobjekt beinhaltet. In dieses fügt der Client zunächst den Zugriffsschlüssel ein, welchen er zuvor vom Server empfangen hatte. Anschließend generiert der Client eine 128 Bit lange Nonce und fügt diese in das Nachrichtenobjekt ein. Die Nonce ist eine Zufallszahl, die Schutz vor *replay*-Angriffen liefert und die spätere Zuordnung der Serverantwort zur passenden Clientanfrage ermöglicht. Daher muss die Nonce bis zum Eintreffen der Serverantwort gespeichert werden. Zusätzlich enthält die Nachricht die minimale vom Client unterstützte NTS-Version (zurzeit nur Version 1) und die Listen mit unterstützten Algorithmen. Dazu zählen zunächst einmal die MAC-Algorithmen, die für die spätere Berechnung des Cookies und des MAC von Bedeutung sind. Nachfolgend kommen die asymmetrischen Verschlüsslungsalgorithmen (*keyEncAlgos*), die zur Verschlüsselung eines symmetrischen Schlüssels[34] zur Anwendung kommen. Abschließend folgen die symmetrischen Verschlüsslungsalgorithmen (*contentEncAlgos*), die zur späteren Verschlüsselung des Cookies benötigt werden. Von jeder Liste wird in der weiteren Kommunikation jeweils ein Algorithmus eingesetzt. Nach der Befüllung der Nachricht sendet der Client diese über ein NTP-Paket an den Server.

[34] Hiermit ist nicht das Cookie gemeint, sondern ein separater Schlüssel, der im weiteren Kommunikationsverlauf für die CMS-Struktur erzeugt wird.

Serverseitig

Nach Erhalt der *client_assoc*-Nachricht überprüft der Server zunächst den Zugriffsschlüssel. Da der Server zustandslos ist und daher auf keine gespeicherten Daten zurückgreifen kann, generiert er diesen nach dem gleichen Schema wie in der *server_access*-Nachricht entsprechend der Gleichung 3.8. Diesen vergleicht der Server nun mit dem Zugriffsschlüssel aus der *client_assoc*-Nachricht. Sind diese identisch, so führt der Server die Verarbeitung der Nachricht fort. Andernfalls bricht er sofort die Verarbeitung der Nachricht und die Kommunikation zum Client ab. Zur clientseitigen Fehlerdiagnose sollte der Server jedoch in einem solchen Fall einen Fehlercode an den Client senden. Dies realisiert er durch die Generierung einer Antwortnachricht, die außer dem Fehlercode keine weiteren Daten beinhaltet. Bei erfolgreicher Überprüfung des Zugriffsschlüssels werden die empfangenen Parameter verarbeitet. Um die zukünftige Kommunikation zu gewährleisten, muss zunächst die NTS-Version überprüft werden. Der Server muss hierbei mindestens die vom Client übertragene NTS-Version beherrschen. Anschließend werden die Listen mit den Algorithmen untersucht. Aus jeder Liste muss der Server mindestens einen unterstützen, um die weitere Kommunikation fortzuführen. Werden diese Bedingungen nicht erfüllt, so wird die weitere Verarbeitung abgebrochen und der Server kann optional noch einen Fehlercode an den Client übertragen.

Ist die Verarbeitung der Clientnachricht erfolgreich, so muss der Server mit diesen Informationen eine *server_assoc*-Nachricht generieren, die nach dem *NTS-Signed*-Archetyp strukturiert wird und ein *ServerAssocData*-Nachrichtenobjekt beinhaltet. In diesem werden zunächst die Nonce und die empfangenen Listen mit den Algorithmen in unveränderter Form eingefügt. Anschließend fügt der Server die Algorithmen ein, die für die weitere Kommunikation verwendet werden sollen. Dazu wählt der Server aus jeder Liste einen Algorithmus aus, der die größte Sicherheit bietet und den auch beide Teilnehmer unterstützen. Danach fügt der Server die *proposedVersion* hinzu. Diese vom Server vorgeschlagene NTS-Version, ist die höchste Version, die der Server beherrscht und muss mindestens der vom Client empfangenen *minVersion* entsprechen.

Nach der Befüllung des Nachrichtenobjekts generiert der Server die CMS-*SignedData*-Struktur und bettet diese ein. Anschließend fügt der Server alle Zertifikate in die CMS-Struktur ein, die der Client für den Aufbau des Zertifizierungspfades benötigt. Zusätzlich kann der Server auch Zertifikatssperrlisten hinzufügen, um veraltete Serverzertifikate zu widerrufen. Im Anschluss wird die CMS-Struktur mit allen enthaltenen Daten digital signiert. Dies erfolgt mit dem privaten Schlüssel des Servers und dem Signaturalgorithmus, welche sich im Serverzertifikat befindet. Abschließend wird diese CMS-Struktur noch mit dem *ContentInfo*-Objekt gekapselt, welches ebenfalls eine CMS-Struktur darstellt. Die NTS-Nachricht wird dem jeweiligen NTP-Paket hinzugefügt und an den Client übertragen.

Clientseitig

Nach Erhalt der Serverantwort überprüft der Client zunächst standardmäßig, ob es sich um die richtige Nachricht handelt, auf die gewartet wurde. Nach der Identifizierung der *server_assoc*-Nachricht parst der Client die enthaltene CMS-Struktur und überprüft den Aufbau, dessen Inhalt der NTS-Spezifikationen gerecht werden muss (siehe dazu Kapitel 3.4.3). Anschließend werden die Zertifikate und Zertifikatssperrlisten extrahiert und überprüft. Die Überprüfung kann offline über den Aufbau der Zertifizierungskette oder online erfolgen. Für die Offline-Überprüfung benötigt der Client zusätzlich eine Liste mit Root-CA-Zertifikaten, die vertrauenswürdige Instanzen darstellen. Für eine erfolgreiche Überprüfung muss eines dieser Zertifikate an der Ausstellung des Serverzertifikats direkt oder indirekt (über zwischengeschaltete CA-Zertifikate) beteiligt sein. Ist letzteres der Fall, müssen die entsprechenden Zertifikate in der CMS-Struktur vorhanden sein, um die vollständige Zertifikatskette aufbauen zu können. Die Überprüfung kann mit den Zertifikatssperrlisten erweitert werden, um Zertifikate, die der Client möglicherweise schon besitzt, für ungültig zu erklären. Neben der Offline-Überprüfung kann auch die Online-Überprüfung über das OCSP-Protokoll genutzt werden, um die Gültigkeit der Zertifikate zu bestätigen. Dieses kann der Client zur Erhöhung der Sicherheit auch periodisch durchführen. Nach der Überprüfung der Zertifikate speichert der Client diese für spätere Zwecke ab und prüft nun die Signatur der CMS-Struktur. Dazu nutzt er den öffentlichen Schlüssel des Servers, welcher sich im Serverzertifikat befindet, und den zugehörigen Signaturalgorithmus. Letzterer befindet sich ebenfalls im Serverzertifikat und in der jeweiligen CMS-Struktur. Da der Signaturalgorithmus nicht ausgehandelt wird, muss der Client vorab prüfen, ob dieser Algorithmus unterstützt wird. Dies hängt von der Implementierung und den lokalen bzw. gesetzlichen Sicherheitsrichtlinien ab.

Ist die Überprüfung der CMS-Signatur erfolgreich verlaufen, so extrahiert der Client das enthaltene Nachrichtenobjekt und parst dieses. Die darin befindliche Nonce muss zunächst mit der zuvor gespeicherten Nonce identisch sein. Anschließend prüft der Client, ob die vom Server gewünschte NTS-Version unterstützt wird. Falls nicht, wählt der Client die höchste Version aus, die von beiden unterstützt werden. Liegt die vom Server vorgeschlagenen Version unterhalb der vom Client verlangten Version, so stellt dies einen Fehler dar, der den Abbruch der Nachrichtverarbeitung zur Folge hat. Nach der Festlegung der NTS-Version überprüft der Client die enthaltenen Listen mit den Algorithmen. Diese müssen mit den Listen identisch sein, die zuvor in der *client_assoc*-Nachricht an den Server übertragen wurden. Diese Überprüfung wird benötigt, da die zuvor gesendete Clientanfrage ungesichert an den Server übertragen wurde. Ein Angreifer wäre somit in der Lage, die Anfrage zu manipulieren, in dem dieser starke Algorithmen in den Listen durch schwache ersetzt oder löscht. Sind die empfangenen Listen mit den zuvor gesendeten Listen identisch, so liefert es dem Client den Beweis, dass diese nicht manipuliert wurden. Nach der Überprüfung werden die vom

Server ausgewählten Algorithmen untersucht. Diese müssen in den jeweiligen Listen enthalten sein. Trifft dies zu, so ist die Überprüfung der empfangenen Servernachricht abgeschlossen. Der Client speichert die empfangenen Parameter ab, passt seinen aktuellen Verbindungszustand an und wechselt in die Cookie-Phase. Die von NTP gelieferten Zeitdaten werden dabei nach wie vor verworfen. Treten bei den Überprüfungen Fehler auf oder wird ein Fehlercode vom Server empfangen, so wird die Verarbeitung abgebrochen und der Client wechselt je nach Fehler in den passenden Zustand. Dies kann beispielsweise eine erneute Zugriffs- oder Assoziationsnachricht bedeuten oder die Blockierung des Servers[35] hervorrufen, wenn wichtige Parameter nicht unterstützt werden.

3.5.3 Cookie-Austauschphase

Clientseitig

In dieser Phase findet zwischen den Kommunikationspartnern der Austausch des Cookies statt, der für die spätere Überprüfung der Zeitnachrichten benötigt wird. Dazu generiert der Client eine *client_cook*-Nachricht, die nach dem *NTS-Plain*-Archetyp strukturiert ist und das *ClientCookieData*-Nachrichtenobjekt beinhaltet. In dieses fügt er zunächst eine Nonce ein, die neu generiert und bis zum Erhalt der Antwortnachricht gespeichert wird. Anschließend bettet der Client die zuvor ausgehandelten Algorithmen ein, die für die Generierung der Serverantwort benötigt werden. Dazu zählen MAC-Algorithmus, *keyEncryption*-Algorithmus und *contentEncryption*-Algorithmus. Zusätzlich fügt der Client den Signaturalgorithmus hinzu, der für die digitale Signatur der CMS-Struktur verwendet werden soll. Dies ist aktuell der Signaturalgorithmus aus dem Serverzertifikat, welches der Client bereits besitzt. Abschließend fügt der Client sein eigenes Zertifikat oder seine Zertifikatskette in das NTS-Nachrichtenobjekt ein. Dies ist dabei von den Forderungen des Servers abhängig. Verlangt dieser eine Autorisierung, um somit nur bestimmten Clients den NTS-Dienst zugänglich zu machen, so muss das Clientzertifikat von einer Zertifizierungsstelle ausgestellt worden sein. Dieses kann der Server mit Hilfe der Zertifikatskette oder dem OCSP-Protokoll prüfen. Es ist möglich, dass der Client mehrere Schlüsselpaare und die dazugehörigen ausgestellten Zertifikate besitzt und dem Server das jeweils korrekte Zertifikat überträgt. Diese müssen bereits vor dem Start des NTS-Dienstes als Ressource (z.B. als Dateien) vorhanden und zugreifbar sein. Wird vom Server keine Autorisierung verlangt, so generiert der Client an der Stelle sein eigenes Schlüsselpaar und das entsprechende selbstsignierte Zertifikat auf Basis der ausgehandelten Algorithmen[36]. Einigen sich die Kommunikationspartner beispielsweise beim *keyEncryption*-Algorithmus auf RSA, so generiert der Client dementsprechend ein

[35] Dies wird jedoch nicht empfohlen, da Fehlercodes ungesichert übertragen werden und somit von einem Angreifer ausgenutzt werden können (ähnliche Methode wie beim KoD-DoS (s.a. Kapitel 3.1.3).
[36] Steht derzeitig im Konflikt mit der Spezifikation (siehe dazu Kapitel 6.2.2).

RSA-Schlüsselpaar. Die dafür benötigten Parameter (Exponent und Bit-Länge) die den Sicherheitsanforderungen gerecht werden müssen, legt der Client selber fest. Der für die Signatur des Clientzertifikats benötigte Hash-Algorithmus, kann entweder aus dem ausgehandelten MAC-Algorithmus oder dem Signaturalgorithmus aus dem Serverzertifikat abgeleitet werden[37]. Das befüllte NTS-Nachrichtenobjekt wird nun in das Erweiterungsfeld der NTP-Nachricht eingebettet. Die Information, um welche NTS-Version es sich handelt, wird von nun an über die Kennung im *Field Type* des Erweiterungsfeldes abgeleitet, die derzeit immer Version 1 ist. Nach erfolgreicher Befüllung des NTP-Pakets, wird dieses nun an den Server übertragen.

Serverseitig

Nach Erhalt der *client_cook*-Nachricht überprüft der Server zunächst die verwendete NTS-Version, die er aus dem *Field Type* des Erweiterungsfeldes ableitet. Anschließend parst er das NTS-Nachrichtenobjekt und überprüft, ob die darin enthaltenen Algorithmen unterstützt werden. Der Signaturalgorithmus muss hierbei mit dem Algorithmus aus dem Serverzertifikat identisch sein. Als nächstes überprüft der Server das Clientzertifikat. Wird eine Autorisierung benötigt, so muss der Server das Clientzertifikat durch Aufbau der Zertifizierungskette bis zu einer vertrauenswürdigen Zertifizierungsstelle überprüfen können. Wenn dies nicht erforderlich ist, wird das Clientzertifikat nur auf Konsistenz geprüft (Inhalt und Signatur). Treten bei der Überprüfung Fehler auf, so wird die Verarbeitung abgebrochen und der Server kann zur Diagnose Fehlercodes an den Client senden.

Ist die Verarbeitung erfolgreich, muss der Server aus den empfangenen Informationen eine *server_cook*-Nachricht generieren. Diese muss nach dem *NTS-Encrypted-and-Signed*-Archetyp strukturiert sein und das *ServerCookieData* als NTS-Nachrichtenobjekt beinhalten. Der Server berechnet zunächst den KIV entsprechend der folgenden Gleichung 3.9:

$$KIV = MSB_{128}\big(H(Zertifikat\ des\ Clients)\big) \tag{3.9}$$

Der KIV ist der Hash-Wert des Clientzertifikats, wobei nur die 128 höherwertigsten Bits verwendet werden. Das Clientzertifikat muss hierfür in serieller Form als DER-kodierter ASN.1-Datenstrom vorliegen[38]. Der zur Berechnung verwendete Hash-Algorithmus wird aus dem MAC-Algorithmus abgeleitet[39]. Mit den derzeit zur Verfügung stehenden und sicher geltenden Algorithmen (z.B.: *hmac-sha512*) wird die Länge des berechneten KIV von 128 Bit nie unterschritten. Sollte dies später durch möglicherweise neue Algorithmen der

[37] Dieses Vorgehen ist in der Spezifikation derzeit nicht genau festgelegt (s.a. Kapitel 3.6.3 und 6.2.3).
[38] Die Kodierungsform ist derzeit nicht definiert, allerdings gilt die DER-Kodierung als sinnvoll.
[39] Dies ist derzeit nicht eindeutig festgelegt (s.a. Kapitel 6.2.1).

Fall sein, so werden die fehlenden Bits durch ein *Zero-Padding* aufgefüllt[40]. Mit Hilfe des KIV kann der Server nun das Cookie gemäß folgenden der Gleichung 3.10 berechnen.

$$Cookie = MSB_{128}\left(MAC(Server\ Seed,\ KIV)\right) \tag{3.10}$$

Für diese Berechnung wird neben dem KIV auch der ausgehandelte MAC-Algorithmus und der *Server Seed* benötigt. Letzterer ist an den MAC-Algorithmus gekoppelt und liegt bereits als *lookup table*[41] vor. Das Ergebnis wird anschließend wieder auf die Länge von 128 Bits (beginnend vom MSB) angepasst. Nach der Berechnung wird das Cookie und die vom Client empfangene und unveränderte Nonce in das Nachrichtenobjekt integriert.

Im nächsten Schritt erfolgt die Verschlüsselung des Nachrichtenobjekts. Dazu generiert der Server eine kryptografisch sichere Zufallszahl, die als symmetrischer Schlüssel verwendet wird. Die Bitlänge des Schlüssels hängt vom ausgehandelten *contentEncryption*-Algorithmus ab. Mit diesem Algorithmus und dem generierten Schlüssel erfolgt nun die symmetrische Verschlüsselung des Nachrichtenobjekts, welches anschließend der Server in die CMS-*EnvelopedData*-Struktur kapselt. Der symmetrische Schlüssel wird nun mit dem ausgehandelten *keyEncryption*-Algorithmus und dem öffentlichen Schlüssel des Clients asymmetrisch verschlüsselt. Diesen entnimmt der Server aus dem empfangenen Clientzertifikat. Der verschlüsselte Schlüssel wird nun ebenfalls in diese CMS-Struktur eingebettet.

Als nächstes erzeugt der Server eine *SignedData*-Struktur, in der das *EnvelopedData*-Objekt gekapselt wird. Diese Struktur wird nach dem gleichen Muster wie in der *server_assoc*-Nachricht gefüllt und mit dem privaten Schlüssel des Servers digital signiert, wobei die Serverzertifikate nicht mehr hinzugefügt werden sollten. Diese besitzt der Client bereits und würden zudem die NTS-Nachricht unnötig vergrößern. Abschließend wird alles wieder in eine *ContentInfo*-Struktur eingebettet, in das NTP-Erweiterungsfeld eingefügt und an den Client übertragen.

Clientseitig

Bei Erhalt der Serverantwort überprüft der Client zunächst standardmäßig, ob es sich um die richtige Nachricht handelt, auf die gewartet wird. Nach Identifizierung der *server_cook*-Nachricht parst der Client die enthaltene CMS-Struktur und überprüft den Aufbau der *SignedData*-Struktur. Anschließend wird die digitale Signatur der CMS-Struktur geprüft. Hierfür verwendet der Client den entsprechenden Signaturalgorithmus und den öffentlichen

[40] Ist eine Annahme, da dieser Fall nicht definiert ist.
[41] Diese wird serverseitig beim Programmstart generiert und periodisch erzeugt (s.a. Kapitel 3.6.2)

Schlüssel des Servers, den er aus dem bereits gespeicherten Serverzertifikat entnimmt. Darauf wird das gekapselte *EnvelopedData*-Objekt aus der *SignedData*-Struktur extrahiert. Der Client entnimmt nun den verschlüsselten symmetrischen Schlüssel aus dieser Struktur und entschlüsselt diesen mit seinem privaten Schlüssel und dem ausgehandelten *keyEncryption*-Algorithmus. Anschließend wird mit dem symmetrischen Schlüssel und dem ausgehandelten *contentEncryption*-Algorithmus das in *EnvelopedData* befindliche Nachrichtenobjekt entschlüsselt. Der Client gleicht nun die darin enthaltene Nonce mit der zuvor gespeicherten Nonce ab. Stimmen diese überein, speichert der Client das Cookie ab, verwirft die Zeitdaten aus dem NTP-Paket und geht in die Zeitsynchronisationsphase über. Tritt hingegen ein Fehler auf oder wird ein Fehlercode vom Server empfangen, verwirft der Client die Nachricht und setzt seinen Zustand auf Basis des Fehlers.

3.5.4 Zeitsynchronisationsphase

Clientseitig

In dieser Phase werden die gesicherten Zeitinformationen übertragen, mit denen der Client die lokale Uhrzeit anpassen kann. Dazu generiert er eine *time_request*-Nachricht, die nach dem *NTS-Plain*-Archetyp aufgebaut ist und das *TimeRequestSecurityData*-Nachrichtenobjekt beinhaltet. In dieses fügt er zunächst eine neu generierte Nonce ein, die er wieder bis zum Erhalt der Antwortnachricht speichert. Anschließend berechnet er den KIV nach dem gleichen Schema wie der Server in der *server_cook*-Nachricht gemäß der Gleichung 3.9. Dieses speichert der Client zusammen mit dem ausgehandelten MAC-Algorithmus in das Nachrichtenobjekt. Dieses wird wieder in das erste Erweiterungsfeld des NTP-Pakets eingefügt, wobei dessen *Field Type* erneut die NTS-Version in Form der Kennung implizit beinhaltet.

Im nächsten Schritt wird der MAC[42] des NTP-Pakets generiert. Dazu veranlasst NTS zunächst die Aktualisierung der Zeitstempel durch das NTP-Protokoll. Dies eliminiert die Zeiten, die NTS für die bisherige Verarbeitung der Nachricht benötigte. Nachfolgende Veränderungen oder Aktualisierungen der Inhalte im NTP-Paket sind nun nicht mehr erlaubt. Die einzige Ausnahme ist nur das Hinzufügen des MAC, der zunächst berechnet werden muss. Bevor dies geschieht, speichert der Client den *transmit_timestamp* aus dem NTP-Paket ab, den er später zusammen mit der Nonce für die Überprüfung der Antwortnachricht benötigt. Als nächstes generiert der Client den MAC entsprechend der folgenden Gleichung 3.11:

$$MAC = MSB_{128}\left(MAC(Cookie,\ NTP\text{-}Paket)\right) \tag{3.11}$$

[42] Dies bezieht sich erneut auf den NTS-MAC und nicht den nativen NTP-MAC

Hierfür muss das NTP-Paket vollständig in serieller Form vorliegen. Für die Berechnung wird außerdem der ausgehandelte MAC-Algorithmus und das gespeicherte Cookie benötigt. Das Ergebnis wird wieder auf 128 Bit (beginnend vom höherwertigsten Bit) gekürzt. Anschließend wird der MAC als zweites Erweiterungsfeld in das NTP-Paket eingebettet und an den Server übertragen.

Serverseitig

Nach Erhalt der *time_request*-Nachricht überprüft der Server zunächst die verwendete NTS-Version, die er aus dem *Field Type* des Erweiterungsfeldes abgeleitet. Anschließend parst er das NTS-Nachrichtenobjekt und überprüft, ob der darin enthaltene MAC-Algorithmus unterstützt wird. Im nächsten Schritt berechnet der Server mit Hilfe des KIV und des MAC-Algorithmus das Cookie neu. Dies geschieht wieder auf Basis der Gleichung 3.10. Das generierte Cookie wird daher mit dem vom Client gespeicherten Cookie identisch sein. Im Folgenden Schritt extrahiert der Server den MAC aus dem NTP-Paket und berechnet diesen selbst. Dazu wird zunächst wieder das NTP-Paket (ohne den MAC) serialisiert. Nun wird der MAC gemäß der Gleichung 3.11 berechnet und anschließend mit dem extrahierten MAC verglichen. Sind diese identisch, liefert es dem Server den Beweis, dass es sich um einen NTS-Client handelt und die übertragenen Daten nicht verändert wurden. Dies ist beispielsweise für den Server wichtig, wenn er nur autorisierte Clients zulässt. Schlägt die Überprüfung fehl oder sind die übertragenen Parameter unzulässig, so bricht der Server die Verarbeitung ab und kann dem Client die Ursache durch einen Fehlercode mitteilen.

Bei erfolgreicher Überprüfung generiert der Server aus den empfangenen Informationen eine *time_response*-Nachricht. Diese ist nach dem *NTS-Plain*-Archetyp aufgebaut und beinhaltet das *TimeResponseSecurityData* als NTS-Nachrichtenobjekt. In dieses kopiert der Server die vom Client empfangene und unveränderte Nonce. Das Nachrichtenobjekt wird nun in das erste Erweiterungsfeld des neuen NTP-Pakets eingebettet und die Zeitstempel werden aktualisiert. Die folgende Verarbeitung ist nun zeitkritisch und wirkt sich negativ auf die Genauigkeit von NTP aus. Eine begrenzte und festgelegte Verarbeitungsdauer gibt es nicht, als Richtlinie sind aber Zeiten von 10µs bis 30µs anzustreben. Nach dem gleichen Muster wie beim Client, serialisiert der Server nun das NTP-Paket und erzeugt mit dem empfangenen MAC-Algorithmus und dem berechneten Cookie den MAC. Dieser wird als zweites Erweiterungsfeld an das NTP-Paket angehängt und an den Client übertragen.

Clientseitig

Nach Erhalt der Serverantwort überprüft der Client zunächst, ob es sich um die richtige Nachricht in der korrekten NTS-Version handelt. Nach der Identifizierung der *time_respone*-Nachricht vergleicht der Client zunächst die enthaltene Nonce mit der zuvor gespeicherten. Sind diese identisch, prüft der Client den gespeicherten *transmit_timestamp* mit dem *originator_timestamp* aus dem empfangenen NTP-Paket. Auch diese müssen identisch sein. Anschließend extrahiert der Client den MAC aus dem NTP-Paket, generiert nach dem bekannten Schema einen neuen MAC für das empfangene NTP-Paket und vergleicht diesen mit dem extrahierten MAC. Sind auch diese identisch, so liefert es den Beweis, dass die empfangenen Zeitdaten von einem NTS-Zeitserver stammen und die Zeitdaten nicht verändert wurden. Der Client teilt dies dem NTP-Protokoll mit, das die Zeitdaten nun für die Synchronisierung der lokalen Uhrzeit verwenden darf. Danach beginnt die Zeitsynchronisationsphase von vorne. Treten hingegen Fehler auf, so wird das Paket verworfen und je nach Art des Fehlers wird der entsprechende Zustand gesetzt.

3.6 Gesonderte Themen

Dieses Unterkapitel behandelt einige NTS-bezogene Themen separat, die zusätzliche Regelungen verfolgen oder informativen Charakter besitzen. Dazu zählen die Aushandlung der NTS-Version, die Generierung des *Server Seeds*, die unterstützten MAC-Algorithmen, der Aufbau der Zertifikate, die Definition der Fehlercodes, die zulässigen Paketgrößen und die Behandlung der durch NTS erzeugten Latenz.

3.6.1 Die NTS-Version

Die Aushandlung und Verwendung der NTS-Version erfolgt auf unterschiedlichen Wegen. Die Verhandlung wird in den Assoziationsnachrichten über das clientseitige Element *minVersion* und dem serverseitigen Element *proposedVersion* realisiert. Die *minVersion* stellt hierbei die minimale vom Client unterstützte NTS-Version dar. Die *proposedVersion* ist die vom Server vorgeschlagene Version. Dies ist zudem die höchste Version, die der Server unterstützt. Die *proposedVersion* muss für eine erfolgreiche Einigung mit dem Client mindestens der *minVersion* entsprechen oder höher sein. Der Client legt anschließend die Version fest, indem er die höchste Version auswählt, die von Client und Server unterstützt werden. Die Version wird hierbei durch eine Ganzzahl repräsentiert, deren Wert im Bereich 0 bis 255 liegt. Der eingetragene Wert entspricht hierbei die Versionsnummer (der Wert 1 entspricht also der Version 1). Die *NTSVersion* wird wie folgt definiert:

Listing 3.23: ASN.1-Beschreibung des NTS-Elements "NTSVersion"

```
NTSVersion ::= INTEGER (0..255)
```

Die Nutzung der NTS-Version nach der Aushandlung erfolgt allerdings nicht mehr über ein separates ASN.1-Element im NTS-Nachrichtenobjekt, sondern direkt im Erweiterungsfeld des NTP-Pakets. Die für NTS verwendete Kennung im *Field Type* erlaubt neben der Identifizierung des enthaltenen NTS-Inhalts auch die Zuordnung der verwendeten NTS-Version. Diese ist somit implizit vorhanden und ermöglicht das korrekte Parsen der enthaltenen Nachrichten. Zugriffs- und Assoziationsnachrichten werden zudem immer in Version 1 übertragen, da die NTS-Version zu diesem Zeitpunkt der Kommunikation noch nicht festgelegt ist. Derzeit sind für NTS-Inhalte im NTP-Erweiterungsfeld noch keine *Field Types* definiert, werden aber zunächst nur als Version 1 vorliegen.

Da Client und Server jeweils nur eine zulässige Versionsgrenze und nicht den kompletten Wertebereich kommunizieren, kann es bei der Aushandlung in bestimmten Konstellationen zu Fehlern kommen, die erst später erkannt werden. Die möglichen Konstellationen und deren Auswirkungen werden an dieser Stelle näher betrachtet. Hierfür werden dem Client und dem Server fiktive NTS-Versionen zugeteilt, die während der Assoziationsphase ausgehandelt werden sollen.

Fall 1: Die Bereiche der unterstützten Versionen sind unterschiedlich und überdecken sich nicht. Der Wertebereich des Clients ist kleiner als der des Servers.

Listing 3.24: Konstellation 1 der NTS-Versionsaushandlung

```
NTS-Versionen des Clients: 1, 2, 3, 4
NTS-Versionen des Servers:             5, 6, 7, 8
```

minVerison des Clients:	1
proposedVersion des Servers:	8
festgelegte Version durch den Client:	4

Ergebnis: Die vom Client festgelegte Version ist ungültig. Dieser wählt die höchste Version aus, die Client und Server unterstützen. Dass der Server das NTS-Protokoll erst ab Version 5 unterstützt, ist dem Client jedoch unbekannt. In einer solchen Konstellation wird der Fehler erst serverseitig in den Cookie-Nachrichten fest-

gestellt. Diese bricht dann die Kommunikation ab und kann einen entsprechenden Fehlercode an den Client übertragen. Dieser könnte nun über eine Blockierung des Servers entscheiden, da es scheinbar nie eine Einigung geben wird[43].

Fall 2: Die Bereiche der unterstützen Versionen überdecken sich. Der Wertebereich des Clients ist kleiner als der des Servers.

Listing 3.25: Konstellation 2 der NTS-Versionsaushandlung

```
NTS-Versionen des Clients:    2, 3, 4, 5, 6
NTS-Versionen des Servers:          4, 5, 6, 7, 8
```

minVerison des Clients:	2
proposedVersion des Servers:	8
festgelegte Version durch den Client:	6

Ergebnis: Die vom Client festgelegte Version ist gültig. Die weitere Kommunikation erfolgt in der ausgehandelten NTS-Version 6.

Fall 3: Die Bereiche der unterstützen Versionen überdecken sich. Der Wertebereich des Clients ist größer als der des Servers.

Listing 3.26: Konstellation 3 der NTS-Versionsaushandlung

```
NTS-Versionen des Clients:          4, 5, 6, 7, 8
NTS-Versionen des Servers:    2, 3, 4, 5
```

minVerison des Clients:	4
proposedVersion des Servers:	5
festgelegte Version durch den Client:	5

[43] Die Blockierung eines NTS-Servers durch den Client auf Basis der Fehlercodes ist gefährlich und angreifbar, da diese Informationen nicht geschützt sind. Ein Angreifer könnte sich in solchen Fall bewusst als NTS-Server ausgeben und Fehlercodes übertragen, um die Blockade des echten NTS-Servers durch den Client zu provozieren. Die Blockierung eines Servers ist KEIN Teil der NTS-Spezifikation und stellt nur das mögliche Handeln des Clients dar.

Ergebnis: Die vom Client festgelegte Version ist gültig. Die weitere Kommunikation erfolgt in der ausgehandelten NTS-Version 5.

Fall 4: Die Bereiche der unterstützen Versionen sind unterschiedlich und überdecken sich nicht. Der Wertebereich des Clients ist größer als der des Servers.

Listing 3.27: Konstellation 4 der NTS-Versionsaushandlung

```
NTS-Versionen des Clients:              5, 6, 7, 8
NTS-Versionen des Servers:  1, 2, 3
```

minVerison des Clients: 5
proposedVersion des Servers: -
festgelegte Version durch den Client: -

Ergebnis: Zwischen Client und Server kommt keine Einigung der NTS-Version zustande. Der Server stellt in den Assoziationsnachrichten fest, dass seine Version die Bedingung $\boldsymbol{proposedVersion \geq minVersion}$ nicht erfüllt und sendet daher nur einen entsprechenden Fehlercode an den Client zurück. Eine Versionseinigung ist hier nicht möglich.

Fall 5: Dies ist ein Sonderfall, in dem ein Teilnehmer keinen vollständigen Versionsbereich unterstützt. Derartige Fälle werden nicht speziell behandelt, da es zum jetzigen Zeitpunkt der Spezifikation auch nicht notwendig ist.

Listing 3.28: Konstellation 5 der NTS-Versionsaushandlung

```
NTS-Versionen des Clients: 1, 2, 3, 4, 5, 6
NTS-Versionen des Servers:       3, 4, 5,       8, 9
```

minVerison des Clients: 1
proposedVersion des Servers: 9
festgelegte Version durch den Client: 6

Ergebnis: Die vom Client festgelegte Version wird auch hier nicht unterstützt, da die vom Client festgelegte Version 6 nicht vom Server unterstützt wird (vergleichbar mit Fall 1).

Mögliche Lösungsansätze

Die einzige Lösung wäre in einem solchen Fall, die komplette Übertragung der unterstützten Versionsnummern vom Client zum Server und umgekehrt. Erst dann kann eine korrekte Festlegung der Versionsnummer erfolgen. Die aktuell spezifizierte Aushandlung der Version ist dennoch unproblematisch und würde höchstens zu einer Fehlererkennung im fortgeschrittenen Kommunikationsablauf führen. Die Sicherheit wird dadurch nicht gefährdet.

3.6.2 Der Server Seed

Der *Server Seed* ist ein symmetrischer Schlüssel, der nur serverseitig existiert und geheim bleibt. Dieser wird durch einen kryptografisch sicheren Zufallszahlengenerator (CSPRNG) erzeugt und kommt bei der Berechnung des Cookies zur Anwendung. Auch clientseitig wird ein solcher Zufallszahlengenerator für die Generierung der Schlüsselpaare benötigt. Da ein Zufallszahlengenerator in Software implementiert wird und somit stark deterministisch ist, generiert dieser ausschließlich Pseudozufallszahlen. Diese können leicht ermittelt werden und wodurch diese keinen Schutz bieten. Um dem entgegenzuwirken, kann der Zufallszahlengenerator mit einem Startwert (einem sogenannten *Seed*) versehen werden. Dieser wird einmalig bei der Initialisierung eingesetzt und erschwert das Erraten der Zufallszahl erheblich. Da in der Kryptografie hohe Anforderungen gestellt werden, muss auch dieser Startwert so zufällig wie möglich sein, um diesen nicht erraten zu können [90]. Aus diesem Grund ist die Wahl einer geeigneten Entropiequelle wichtig. Die Entropie stellt das Maß der Zufälligkeit dar, wobei eine hohe Entropie eine qualitativ bessere Zufallszahl erzeugen kann. Da dies softwarebasiert kaum realisierbar ist (diese arbeitet stark deterministisch), nutzt man bevorzugt die Hardware. Hierzu seien einige Beispiele von gut geeigneten und schlechten Entropiequellen genannt [91] [92]:

Gute Entropiequellen:

- Rauschsignale (aus der Hardware)
- Atomare Zerfallsprozesse

Schlechte Entropiequellen:

- Systemzeit
- Serien- oder Modellnummern (Hardware)
- Tastatureingaben
- Zeitintervalle zwischen Tastatureingaben
- Zählprozesse
- Netzwerkstatistiken

In Betriebssystemen werden dazu bestimmte Schnittstellen zur Verfügung gestellt, die eine geeignete Entropiequelle liefern. Dies ist beispielsweise bei Microsoft Windows die *CryptoAPI* und bei Linux-Systemen die „*/dev/urandom*“- oder „*/dev/random*“-Quelle [93] [94].

Da der vom NTS-Server generierte *Server Seed* bei einem erfolgreichen Angriff auf ein ausgestelltes Cookie herausgefunden werden kann, sind weitere Sicherheitsmaßnahmen notwendig. Aus diesem Grund generiert der Server für jeden unterstützten MAC- und Hash-Algorithmus einen separaten *Seed*, der an den jeweiligen Algorithmus gebunden wird. Auf diesen kann der Server anschließend über eine *lookup table* zugreifen. Durch diesen Mechanismus würde ein erfolgreicher Angriff auf ein Cookie lediglich nur einen Algorithmus betreffen. Eine weitere Schutzfunktion ist die periodische Aktualisierung des *Seeds*. Dies sorgt dafür, dass die ausgestellten Cookies ablaufen und somit der Angriff zusätzlich erschwert wird. Die Periode, in der dies geschehen muss, ist nicht weiter definiert und muss sinnvoll gewählt werden. Denkbar sind Perioden von wenigen Tagen bis hin zu einigen Monaten. Möglich ist zudem die Aktualisierung auf Basis der Zugriffe. Wird auf einen *Seed* besonders häufig zugegriffen oder wird ein Schwellwert von erlaubten Zugriffen überschritten, so kann die Erneuerung des *Seed* zusätzlich durchgeführt werden.

3.6.3 Vergleich der MAC-Algorithmen

Die von NTS unterstützten MAC-Algorithmen werden für die Berechnung des Cookies und des MAC benötigt. Diese stellen zum aktuellen Zeitpunkt noch Herausforderungen dar, die im Kapitel 6.2.1 näher erläutert werden. Aktuell wird hierfür ein geeigneter HMAC-Algorithmus empfohlen. Der Server darf hierbei keinen *hmac-sha1* oder schwächere MAC-Algorithmen einsetzen und muss mindestens *hmacWithSHA256* oder stärke Algorithmen verwenden. Der HMAC-Algorithmus ist im RFC 2104 [45] definiert und berechnet sich entsprechend Gleichung 3.12:

$$HMAC(K, m) = H((K' \oplus opad) || H((K' \oplus ipad) || m)) \qquad (3.12)$$

$\boldsymbol{K}$ repräsentiert den geheimen Schlüssel und kann beispielsweise der *Server Seed* für die Berechnung des Cookies oder das Cookie für die Berechnung des MAC sein. Die Variable $\boldsymbol{m}$ ist die Nachricht, die zu authentifizieren ist, und ist beispielsweise das NTP-Paket. $\boldsymbol{H}(\dots)$ ist die dafür verwendete Hash-Funktion (z.B. SHA512). $\boldsymbol{K'}$ ist der geheime Schlüssel, der allerdings mit einem *zero padding* so weit vergrößert wird, wie es von der Hash-Funktion benötigt wird. Dieser wird zusammen mit einem *Padding* (opad/ipad[44]) durch eine XOR-Funktion ($\oplus$) verarbeitet. Der dargestellte Doppelstrich ($||$) verkettet zwei Elemente miteinander.

Neben dem HMAC-Algorithmus gibt es noch weitere wie CMAC, DMAC, GMAC, VMAC und sonstige. Die Verwendung ist derzeit noch nicht empfohlen, da diese momentan noch zur Diskussion stehen. Der Vorteil anderer MAC-Algorithmen ist meist die Berechnungsbasis. Der HMAC-Algorithmus nutzt für die Berechnungen des MAC einen Hash-Algorithmus. CMAC- oder GMAC-Algorithmen funktionieren auf der Grundlage der symmetrischen Verschlüsselung und nutzten AES. Dies ist besonders auf modernen Computern mit Intel- oder AMD-Prozessoren vorteilhaft, die eine hardwarebasierte AES-Beschleunigung bieten. Berechnungen sind somit deutlich schneller und verringern möglicherweise die Latenzzeiten, die während der Generierung der *time_request*- und *time_response*-Nachricht durch die kryptografische Funktion hervorgerufen werden.

Ein erster Vergleich der MAC-Algorithmen wurde im Draft *MACs for NTP* [95] veröffentlicht. In diesem werden verschiedene MAC-Algorithmen auf ihrer Leistungsfähigkeit hin untersucht. Dies erfolgt jeweils mit eingeschalteter AES-Beschleunigung und ohne. Im Test wurde dazu ein 48 Byte großer Datenblock verwendet, der das NTP-Paket darstellen soll[45]. Anschließend wurde gemessen, wie oft ein Algorithmus diesen Datenblock pro Sekunde verarbeiten konnte. Die Messung erfolgte dazu auf einem *Intel(R) Xeon(R) E5-2676v3*-Prozessor mit 2,40 GHz unter Verwendung eines physischen Prozessorkerns. Die Ergebnisse können aus der Tabelle 3.4 und dem dazugehörigen Diagramm (Abbildung 3.20) entnommen werden.

Tabelle 3.4: Geschwindigkeitsvergleich unterschiedlicher MAC-Algorithmen

Algorithmus	Berechnungen pro Sekunde mit AES-Beschleunigung	Berechnungen pro Sekunde ohne AES-Beschleunigung
legacy MD5	3,188 M	3,165 M
HMAC-MD5	2,742 M	2,749 M
HMAC-SHA-224	1,265 M	1,267 M
CMAC(AES)	7,567 M	4,388 M
GMAC(AES)	16,612 M	4,627 M
Poly1305(ChaCha20)	2,598 M	2,398 M

[44] *opad* ist das äußere Padding (0x5c5c…) und *ipad* das innere Padding (0x36…) [45]
[45] 48 Byte entsprechen der Größe des NTP-Headers [18] ohne Erweiterungsfelder und ohne MAC

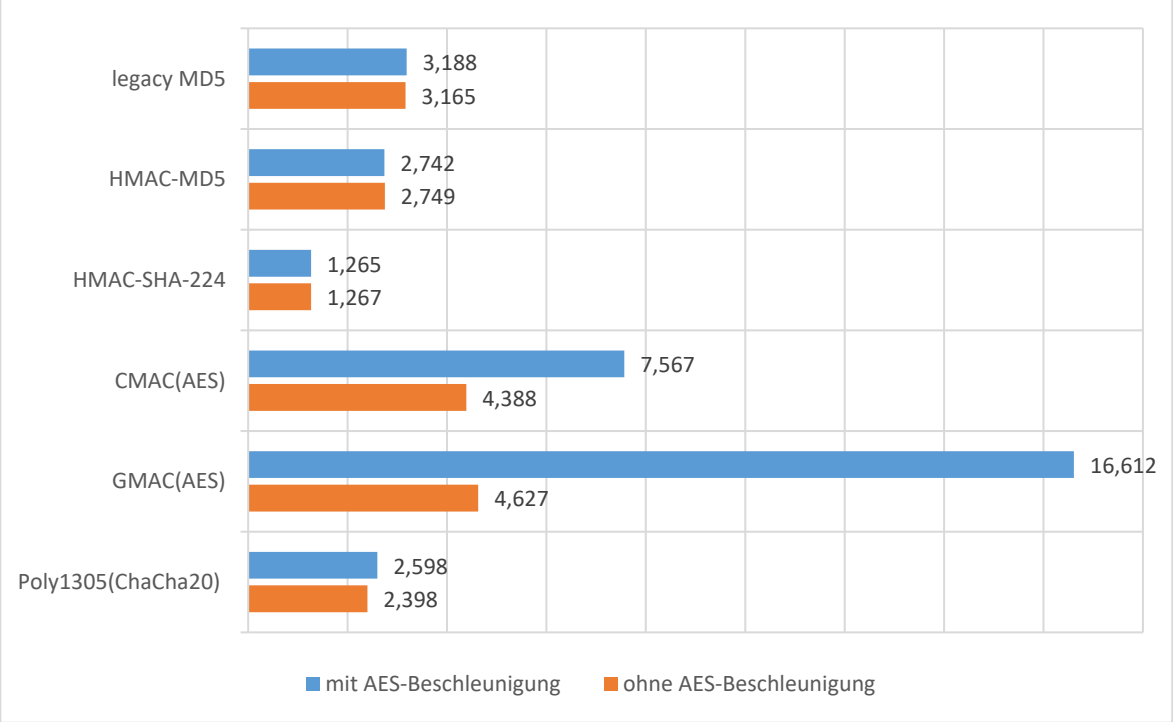


Abbildung 3.20: Geschwindigkeitsvergleich unterschiedlicher MAC-Algorithmen

Es ist ersichtlich, dass der verwendete GMAC-Algorithmus mit eingeschalteter AES-Beschleunigung etwa 13-mal schneller als die HMAC-Berechnung ist. Weitere unabhängige Vergleiche zeigten ähnliche Ergebnisse [96]. Dennoch müssen mehrere Tests gefahren werden, um eine genaue Aussage treffen zu können. Gerade schwächere Hardware wie Einplatinencomputer (z.B. der Raspberry Pi) verwenden andere Hardwarearchitekturen, bei denen auch die AES-Beschleunigung nicht zur Verfügung steht.

3.6.4 Zertifikatseigenschaften und -regelungen

Während der Verhandlungsphase kommen im NTS-Protokoll Zertifikate bzw. Zertifikatsketten zum Einsatz. Hierbei handelt es sich um gängige X.509-Zertifikate [47] [97], die bestimmte Inhalte benötigen, um von den jeweiligen NTS-Teilnehmern akzeptiert zu werden. Die folgenden Beschreibungen sind derzeit noch nicht in den Spezifikationen aufgeführt oder weichen davon ab. Die entsprechenden Informationen wurden mit der PTB kommuniziert und werden voraussichtlich in den zukünftigen Dokumenten der NTS-Spezifikation vorliegen.

3.6.4.1 Das Gültigkeitsdatum

Ein mögliches Problem von NTS stellt die Überprüfung der Zertifikate dar. Jedes ausgestellte oder generierte Zertifikat beinhaltet ein Gültigkeitsdatum, welches den Beginn und das Ende dieser Periode als Zeitstempel beinhaltet. Die Gültigkeitsperiode der Zertifikate ist individuell anpassbar. Der Regelfall sieht aber Perioden von ein bis fünf Jahren[46] vor. Erhält der Client während der Assoziationsphase das Serverzertifikat, so kann dieses für den Client während der Überprüfung als ungültig empfunden werden, obwohl dieses gültig ist. Dieses Problem taucht auf, wenn die lokale Uhrzeit des Clients signifikant von der Serverzeit abweicht. Das ist möglicherweise bei der ersten Inbetriebnahme einer Hardware der Fall, die über NTS gesicherte Zeitdaten empfangen soll. In diesem Szenario liegt die lokale Uhrzeit des Clients außerhalb des im Serverzertifikat angegebenen Zeitbereichs, wodurch dieses Zertifikat als ungültig betrachtet wird. Als Folge bricht der Client die Kommunikation zum NTS-Server ab. Das Ergebnis ist nun ein Henne-Ei-Problem. Der Client braucht gesicherte Zeitdaten vom NTS-Server, um die lokale Uhrzeit zu stellen und verweigert diese, da seine Uhrzeit signifikant von der Zeit des Servers abweicht. Die Lösung ist nicht ganz einfach und von den Anforderungen und der Implementierung abhängig. Somit müsste die Hardware nach der ersten Inbetriebnahme die Zeit zunächst einmalig über das ungesicherte NTP synchronisieren. Alternativ müsste von Hand das Datum angepasst werden, um Zeitspanne zu verringern.

3.6.4.2 Die Zertifikatserweiterungen

Auch die Zertifikatserweiterungen spielen eine wichtige Rolle zur korrekten Identifizierung und Übertragung der Authentizitäts- und Autorisierungsinformationen. Nach derzeitigem Stand kommen bis zu vier Zertifikatserweiterungen in den NTS-Zertifikaten zum Einsatz, die nachfolgend näher beschrieben werden. Die korrekte Verwendung dieser Erweiterungen wird im Abschnitt 3.6.4.4 dargestellt.

Basic Constraints

Diese Erweiterung ist in den NTS-Zertifikaten optional. Sie kann von dem NTS-Server verwendet werden, um den Typ des Zertifikats eindeutig zu machen, sofern es dem restlichen Zertifikat nicht widerspricht. Der Typ definiert in diesem Fall, ob es sich um ein primäres Zertifikat handelt – welches von einer CA ausgestellt wurde – oder ob sich das Zertifikat selber als eine CA ausgibt. Die Erweiterung unterstützt folgende zwei Parameter:

[46] Das sind durchschnittliche Werte von gängigen Zertifikaten, deren Gültigkeitsdauer vom Einsatzgebiet abhängt.

- CA=<TRUE, FALSE>
- pathlen=<länge>

Der erste Parameter gibt den Typ des Zertifikats an. Mit „*CA=true*“ liefert das Zertifikat die Information, dass es sich um eine CA handelt. Der zweite Parameter *pathlen* beschränkt die Zertifikatskette. Ein Wert von null bedeutet, dass die Zertifikate die durch dieses Zertifikat signiert wurden, nicht berechtigt sind, selber Zertifikate zu signieren.

Subject Key Identifier

Die *Subject-Key-Identifier*-Erweiterung (SKI) muss in den NTS-Zertifikaten enthalten sein. Diese wird für die Identifizierung des Unterzeichners (der NTS-Server) in den CMS-Strukturen benötigt. Der SKI wird über einen 160-Bit-SHA-1-Hash des öffentlichen Schlüssels gebildet, welcher sich im Zertifikat befindet. Alternative Generierungsmöglichkeiten werden zudem im RFC 5280 [97] beschrieben.

Key Usage

Die *Key-Usage*-Erweiterung legt den Verwendungszweck des im Zertifikat enthaltenen öffentlichen Schlüssels fest. Dazu können verschiedene *Flags* gesetzt werden, um die genauen Eigenschaften zu definieren. Die Kombination mehrerer *Flags* ist hierbei erlaubt, die nachfolgend kurz erläutert werden [47] [98]:

- **Digitale Signatur** (*digitalSignature*): Dieses Bit wird gesetzt, wenn der öffentliche Schlüssel für digitale Signaturen verwendet werden soll und nicht den Zweck der *Nichtabstreitbarkeit* zu erfüllen hat, d.h. eher einen kurzfristigen Charakter haben.

- **Nichtabstreitbarkeit** (*nonRepudiation*): Diese Option ist gesetzt, wenn der Schlüssel für digitale Signaturen verwendet werden soll, die sich vom Erzeuger nicht abstreiten lassen und dadurch eine rechtliche Relevanz besitzen können.

- **Schlüsselverschlüsselung** (*keyEncipherment*): Diese Option ist gesetzt, wenn der öffentliche Schlüssel für die Verschlüsselung von anderen Schlüsseln oder Sicherheitsinformation verwendet werden soll. Im Zusammenspiel mit *encipher only* und *decipher only* kann die Verwendung zusätzlich eingeschränkt werden.

- **Datenverschlüsselung** (*dataEncipherment*): Dieses *Flag* wird verwendet, wenn der Schlüssel zur Verschlüsselung von Daten verwendet werden soll. Die Verschlüsselung von anderen Schlüsseln wird allerdings ausgeschlossen.

- **Schlüsselvereinbarung** (*keyAgreement*): Dieses Bit wird gesetzt, wenn der *Diffie-Hellman*-Algorithmus für die Schlüsselvereinbarung verwendet werden soll.

- **Zertifikatsignatur** (*keyCertSign*): Der Schlüssel wird für die Verifizierung von Zertifikatsignaturen verwendet. Dies ist bei CA-Zertifikaten sinnvoll.

- **CRL-Signatur** (*cRLSign*): Der Schlüssel wird für die Verifizierung von Signaturen auf Zertifikatssperrlisten verwendet. Dies ist bei CA-Zertifikaten sinnvoll.

- **Nur Verschlüsselung** (*encipherOnly*): Kann nur zusammen mit dem *keyAgreement*-Bit verwendet werden und darf nur zur Verschlüsselung von Daten eingesetzt werden.

- **Nur Entschlüsselung** (*decipherOnly*): Kann nur zusammen mit dem *keyAgreement*-Bit verwendet werden und darf nur zur Entschlüsselung von Daten eingesetzt werden.

Extended Key Usage

Neben der *Key-Usage*-Erweiterung gibt es auch noch die *Extended-Key-Usage*-Erweiterung, die zusätzliche Informationen zur Verwendung des Zertifikats beinhaltet. Im NTS-Protokoll sind dazu drei neue Kennungen vorgesehen, die noch definiert werden müssen:

- **NTS-Server-Authentifikation** (*id-kp-ntsServerAuth*): Wenn ein Zertifikatsaussteller diese Kennung in die *Extended-Key-Usage*-Erweiterung einschließt, so liefert es die Beglaubigung, dass das Zertifikatsubjekt ein Zeitserver ist, der das NTS-Protokoll unterstützt.

- **NTS-Server-Autorisierung** (*id-id-kp-ntsServerAuthz*): Wenn ein Zertifikatsaussteller diese Kennung in die *Extended-Key-Usage*-Erweiterung einschließt, so liefert es die Beglaubigung, dass das Zertifikatsubjekt ein Zeitserver ist, der das NTS-Protokoll unterstützt und der Aussteller glaubt, dass dieser die korrekte Zeit verbreitet und dazu auch in der Lage ist.

- **NTS-Client-Autorisierung** (*id-id-kp-ntsClientAuthz*): Wenn ein Zertifikatsaussteller diese Kennung in die *Extended-Key-Usage*-Erweiterung einschließt, so liefert es die Beglaubigung, dass das Zertifikatsubjekt ein NTS-Client ist, der vom Aussteller die Autorisierung für den Zugriff zur sicheren Zeitsynchronisierung bekommen hat. Dies könnte beispielsweise bei kostenpflichtigen Diensten angewendet werden.

3.6.4.3 Zertifikatidentifizierung

Die gängigen X.509-Zertifikat können in drei Typen klassifiziert werden. Dies sind die primären Zertifikate, die zwischengeschalteten CA-Zertifikate und die Root-CA-Zertifikate. Die primären Zertifikate werden von einer Zertifizierungsstelle ausgestellt und stellen das Ende einer Zertifikatskette dar. Charakteristisch für diese Zertifikate ist die digitale Signatur, die immer von dem Zertifikat der ausstellenden Instanz signiert wird und die unterschiedlichen Namen im entsprechenden Subjekt- und Ausstellerfeld (*subject* und *issuer*). Erweiterungen können dabei enthalten sein, dürfen aber nicht auf ein CA-Zertifikat schließen lassen. Ist dies jedoch der Fall, z.B. in dem das *Key-Usage*-Flag „*keyCertSign*“ gesetzt ist, so gilt das Zertifikat als zwischengeschaltetes CA-Zertifikat. Root-CA-Zertifikate zeichnen sich hingegen dadurch aus, dass sie ihr eigenes Zertifikat signierten und dass die Namen im Subjekt- und Ausstellerfeld identisch sind. Außerdem beinhalten diese Zertifikate Erweiterungen, die dies zusätzlich bestätigen (z.B. die *Basic-Constraints*-Erweiterung mit dem Wert „*CA=TRUE*“). Die korrekte Verwendung unterschiedliche Erweiterungen zur Identifizierung eines Root-CA-Zertifikats ist nicht eindeutig und wird unterschiedlich gehandhabt. Die folgenden Beispiele zeigen hierzu unterschiedliche Definitionen von Root-CA-Zertifikaten:

OpenSSL (v1.1.0)

Die Identifizierung des Zertifikats basiert bei OpenSSL auf den allgemeinen Inhalten (z.B. Subjekt- und Ausstellername sowie Signatur), der verwendeten Zertifikatsversion und dem Zusammenspiel der enthaltenen Erweiterungen. Die in der Tabelle 3.5 dargestellten Konfigurationen zeigen das jeweilige Identifizierungsergebnis in Abhängigkeit der verwendeten Erweiterungen. Hierfür wurden sowohl ausgestellte, als auch selbstgenerierte Zertifikate untersucht. Bei selbstgenerierten Zertifikaten ist der Subjekt- und Ausstellername identisch und die Signatur erfolgt mit dem eigenen öffentlichen Schlüssel. Die Ergebnisse wurden hierbei auf Basis der OpenSSL-Funktion “*X509_check_ca* ()“ erzeugt [99]. Die entsprechenden Werte in der Tabelle sind binär dargestellt und geben im oberen Bereich an, welche Flags bei den jeweiligen Konfigurationen aktiviert wurden. Eine mögliche Konfiguration stellt dabei eine Spalte dar. Erweiterungen, in denen keine Flags aktiviert sind, sind nicht in den jeweiligen Zertifikaten enthalten (also kein Einsatz von leeren Erweiterungen im Zertifikat). Der untere Bereich der Tabelle zeigt den identifizierten Zertifikattyp an. Dieser wird in der jeweiligen Spalte mit einer 1 dargestellt, wenn die Typerkennung in der jeweiligen Zeile zutrifft. Dabei fällt auf, dass bei einigen Konfigurationen die Identifizierung des Zertifikats nicht möglich ist. Derartige Konfigurationen müssen bei der Generierung der NTS-Zertifikate vermieden werden.

Tabelle 3.5: Zertifikatsidentifizierung anhand der Zertifikatserweiterungen

aktivierte Zertifikatseigenschaft																									
Basic Constraints (CA:TRUE)	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**
Basic Constraints (CA:FALSE)	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
Key Usage (keyCertSign)	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**
Key Usage (cRLSign)	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
Key Usage (sonstige)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ausgestelltes Zertifikat mit Zertifizierungspfad (*Subjekt ≠ Aussteller; durch Aussteller signiert*)																									
ist primäres Zertifikat	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**
ist zwisch. CA Zertifikat	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
ist root-CA Zertifikat	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
selbstgeneriertes Zertifikat (*Subjekt = Aussteller; selbstsigniertes Zertifikat*)																									
ist primäres Zertifikat	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ist zwisch. CA Zertifikat	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ist root-CA Zertifikat	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**

Microsoft

Die selbstgenerierten Root-CA-Zertifikate für Windows-Server-Systeme verlangen den Einsatz der folgenden *Key-Usage*-Flags [100]:

- „Certificate Signing“ (entspricht *keyCertSign*)
- „Offline CRL Signing“ (entspricht *cRLSign*)
- „CRL Signing“ (entspricht *cRLSign*)

Federal PKI X.509 Certificate and CRL Extensions Profile

Dieses Dokument spezifiziert X.509v3-Zertifikate und Version-2-Zertifikatssperrlisten für *Federal Public Key Infrastructure* (FPKI)-Systeme im *E-Government*-Bereich [101]. Dieses schlägt den Einsatz der folgenden *Key-Usage*-Flags vor, die die CA-Zertifikate beinhalten müssen:

- cRLSign
- keyCertSign

RFC 5280: PKIX Certificate and CRL Profile

Im RFC 5280 [97] werden die möglichen Konfigurationen der CA-Zertifikate beschrieben, die je nach verwendeten Schlüsseltyp bestimmte *Key-Usage*-Flags verwenden *können*. Die Übersicht der verwendeten Flags in Anhängigkeit des Schlüsseltyps, wird in der folgenden Tabelle 3.6 dargestellt.

Tabelle 3.6: Zertifikatsidentifizierung gemäß RFC 5280 (PKIX Certificate)

	RSA	DSA	ECDSA
digitalSignature	Ja	Ja	Ja
nonRepudiation	Ja	Ja	Ja
keyEncipherment	Ja	Nein	Nein
dataEncipherment	Ja	Nein	Nein
keyAgreement	Nein	Nein	Ja
keyCertSign	Ja	Ja	Ja
cRLSign	Ja	Ja	Ja
encipherOnly	Nein	Nein	Nein
decipherOnly	Nein	Nein	Nein

3.6.4.4 Beispiele für gültige Zertifikatkonfigurationen

In NTS ist die korrekte Verwendung der Zertifikatserweiterungen wichtig, da sonst die Erkennung des Zertifikattyps und die darauffolgende Überprüfung fehlschlägt. Dem ist hinzuzufügen, dass derartige Erweiterungen clientseitig nur dann sinnvoll sind, wenn eine Überprüfung möglich ist. Dies ist beispielsweise bei selbstgenerierten Zertifikaten nicht der Fall, da hier keine Infrastruktur zur Verfügung steht, um die Echtheit der Zertifikate zu bestätigen. Für NTS sind die folgenden Zertifikatkonfigurationen sinnvoll.

Selbstgeneriertes Clientzertifikat

- Es wird ein X.509-Zertifikat der Version 3 verwendet
- Subjektname und Ausstellername sind identisch
- Die Gültigkeitsdauer des Zertifikats beträgt mindestens ein Jahr, dessen Beginn auf Basis der lokalen (nicht synchronisierten) Uhrzeit erfolgt
    - Optional liegt der Beginn der Gültigkeitsdauer ein halbes Jahr in der Vergangenheit. Dies ermöglicht dem Server die Prüfung des Zertifikats, wenn die Zeit des Clients vorgeht.

- Das Zertifikat wird durch den eigenen öffentlichen Schlüssel signiert
- Verwendung der Zertifikatserweiterung gemäß der folgenden Tabelle 3.7 und Tabelle 3.8.

Tabelle 3.7: Erlaubte Zertifikatserweiterungen für selbstgenerierte Clientzertifikate

Zertifikatserweiterung	in Zertifikat enthalten	Erweiterung ist kritisch[47]
Basic Constraints	Optional	Nein
Subject Key Identifier	Ja	Nein
Key Usage	Nein	Nein
Extended Key Usage	Nein	Nein

Tabelle 3.8: Gültige Zertifikat-Flags für selbstgenerierte Clientzertifikate

Zertifikatserweiterung	Flag	Verwendbarkeit
Basic Constraints	CA:FALSE	Optional
Basic Constraints	CA:TRUE	Nein
Basic Constraints	Pathlen=<länge>	Nein

Ausgestelltes Clientzertifikat

- Es wird ein X.509-Zertifikat der Version 3 verwendet
- Subjektname und Ausstellername sind unterschiedlich
- Die Gültigkeitsdauer des Zertifikats beträgt mindestens ein Jahr
- Das Zertifikat wird durch den öffentlichen Schlüssel des Ausstellers signiert
- Verwendung der Zertifikatserweiterung gemäß der folgenden Tabelle 3.9 und Tabelle 3.10.

Tabelle 3.9: Erlaubte Zertifikatserweiterungen für ausgestellte Clientzertifikate

Zertifikatserweiterung	in Zertifikat enthalten	Erweiterung ist kritisch
Basic Constraints	Optional	Nein
Subject Key Identifier	Ja	Nein
Key Usage	Optional	Nein
Extended Key Usage	Optional	Nein

[47] Wird ein Zertifikat mit einer unbekannten und als *kritisch* markierten Erweiterung verarbeitet, so muss dieses Zertifikat verworfen werden. Andernfalls kann das Zertifikat verworfen oder die Erweiterung ignoriert werden.

Tabelle 3.10: Gültige Zertifikat-Flags für ausgestellte Clientzertifikate

Zertifikatserweiterung	Flag	Verwendbarkeit
Basic Constraints	CA:FALSE	Optional
Basic Constraints	CA:TRUE	Nein
Basic Constraints	pathlen=<länge>	Nein
Key Usage	digitalSignature	Optional, nicht sinnvoll
Key Usage	nonRepudiation	Optional, nicht sinnvoll
Key Usage	keyEncipherment	Optional
Key Usage	dataEncipherment	Optional
Key Usage	keyAgreement	Optional, nicht sinnvoll
Key Usage	keyCertSign	Nein
Key Usage	cRLSign	Nein
Key Usage	encipherOnly	Optional, nicht sinnvoll
Key Usage	decipherOnly	Optional, nicht sinnvoll
Extended Key Usage	id-id-kp-ntsClientAuthz	je nach Serveranforderung

Ausgestelltes Serverzertifikat

- Es wird ein X.509-Zertifikat der Version 3 verwendet
- Subjektname und Ausstellername sind unterschiedlich
- Die Gültigkeitsdauer des Zertifikats wird entsprechend der lokalen oder gesetzlichen Richtlinien festgelegt
- Das Zertifikat wird durch den öffentlichen Schlüssel des Ausstellers signiert
- Verwendung der Zertifikatserweiterung gemäß der folgenden Tabelle 3.11und Tabelle 3.12.

Tabelle 3.11: Erlaubte Zertifikatserweiterungen für ausgestellte Serverzertifikate

Zertifikatserweiterung	in Zertifikat enthalten	Erweiterung ist kritisch
Basic Constraints	Optional	Nein
Subject Key Identifier	Ja	Nein
Key Usage	Optional	Nein
Extended Key Usage	Optional	Nein

Tabelle 3.12: Gültige Zertifikat-Flags für ausgestellte Serverzertifikate

Zertifikatserweiterung	Flag	Verwendbarkeit
Basic Constraints	CA:FALSE	Optional
Basic Constraints	CA:TRUE	Nein
Basic Constraints	pathlen=<länge>	Nein
Key Usage	digitalSignature	Optional, sinnvoll
Key Usage	nonRepudiation	Optional
Key Usage	keyEncipherment	Optional, nicht sinnvoll
Key Usage	dataEncipherment	Optional, nicht sinnvoll
Key Usage	keyAgreement	Optional, nicht sinnvoll
Key Usage	keyCertSign	Nein
Key Usage	cRLSign	Nein
Key Usage	encipherOnly	Optional, nicht sinnvoll
Key Usage	decipherOnly	Optional, nicht sinnvoll
Extended Key Usage	id-id-kp-ntsServerAuth	Optional, sinnvoll
Extended Key Usage	id-id-kp-ntsServerAuthz	Optional, sinnvoll

3.6.5 Empfehlungen für Schlüssel und Algorithmen

Die in NTS zu verwendenden Algorithmen und Schlüssellängen werden nicht fest definiert und richten sich nach den lokalen oder gesetzlichen Vorgaben, die in zeitlicher Abhängigkeit stehen. Die derzeit empfohlenen Schlüssellängen und Parameter der verschiedenen Algorithmen werden an dieser Stelle kurz aufgeführt und richten sich an den aktuellen gesetzlichen Vorgaben des BSI [86]. Die folgenden Werte aus Tabelle 3.13 bis Tabelle 3.17 stellen mögliche Algorithmen dar, die als sicher oder unsicher gelten.

Asymmetrische Verschlüsslung mit dem RSA-Verfahren

Die empfohlene Länge des Modulus n sollte mindestens 2000 Bit betragen und ab dem Jahre 2017 mindestens 3000 Bit [86, p. 15].

Asymmetrische Verschlüsslung mit Elliptischen Kurven

Empfohlen wird derzeit eine Bitlänge von 250 Bit. Dies ist vergleichbar mit einer Bitlänge von ca. 3000 Bit beim RSA-Verfahren [86, p. 15].

Tabelle 3.13: Empfohlene und nicht empfohlene ECC-Algorithmen

Empfohlene ECC-Algorithmen	Nicht empfohlene ECC-Algorithmen
brainpoolP256r1 [102]	SECP-Reihe [103] (z.B. secp521r1)
brainpoolP320r1 [102]	
brainpoolP384r1 [102]	
brainpoolP512r1 [102]	

Eine umfangreiche Liste mit aktuellen Algorithmen und deren Sicherheitseinstufungen kann zusätzlich auf *safecurves.cr.yp.to* [104] eingesehen werden.

Symmetrische Verschlüsselung

Tabelle 3.14: Empfohlene und nicht empfohlene Blockchiffren

Empfohlene Blockchiffren	Nicht empfohlene Blockchiffren
AES-128 [41]	Triple-DES (diversen Schwächen) [40]
AES-192 [41]	
AES-256 [41]	

Tabelle 3.15: Empfohlene und nicht empfohlene Betriebsarten für Blockchiffren

Empfohlene Betriebsarten	Nicht empfohlene Betriebsarten
CBC (Cipher-Block Chaining) [105]	ECB [105]
CTR (Counter Mode) [105]	
GCM (Galois-Counter-Mode) [105]	

Hash-Algorithmen

Tabelle 3.16: Empfohlene und nicht empfohlene Hash-Algorithmen

Empfohlene Hash-Algorithmen	Nicht empfohlene Hash-Algorithmen
SHA-256 [106]	MD4 [107]
SHA-384 [106]	MD5 [108]
SHA-512 [106]	SHA-1 [106]

MAC-Algorithmen

Tabelle 3.17: Empfohlene und nicht empfohlene MAC-Algorithmen

Empfohlene MAC-Algorithmen	Nicht empfohlene MAC-Algorithmen
CMAC (Schlüssellänge ≥ 128 Bit) [46]	keine bekannt
HMAC (Schlüssellänge ≥ 128 Bit) [45]	
GMAC (Schlüssellänge ≥ 128 Bit) [109]	

3.6.6 Definition der Fehlercodes

Die in NTS verwendeten Fehlercodes sind derzeit nicht definiert, wurden aber vorläufig anhand der Implementierungsergebnisse festgelegt. Diese kann der Server an den Client senden, wenn es bei der Verarbeitung der Clientnachrichten zu Fehlern kam. Eingebettet wird dieser Fehlercode in das *errnum*-Element des *NTSExtensionFieldContent*-Objekts. Dieses wird direkt in das NTP-Erweiterungsfeld eingebettet und transportiert neben der NTS-Nachricht (bzw. dem NTS-MAC), die Kennung des Inhalts und die Fehlernummer. Trat bei der serverseitigen Verarbeitung ein Fehler auf, so muss das *errnum*-Element einen Wert ungleich null haben. Andernfalls muss es auf null gesetzt werden. Das *errnum*-Element hat eine Größe von 2 Byte und stellt somit 16 Bit zur Verfügung, die jeweils einen Fehler repräsentieren können. Durch diese Methode können auch mehrere Fehler gleichzeitig dargestellt werden, in dem die entsprechenden Bits gesetzt werden. Die durch die Bits dargestellten Fehler, werden in der Tabelle 3.18 aufgeführt, wobei Bit 0 das höchstwertigste Bit (MSB) repräsentiert.

Tabelle 3.18: Vorläufige Definition der Fehlercodes im NTS-Protokoll

Fehler	Bit	Beschreibung
ServerNotOperational	0	Der NTS-Server ist nicht betriebsbereit (Wartungsmodus)
VersionNotSupported	1	Die NTS-Version des Clients wird nicht unterstützt
AuthenticationFailed	2	Die Authentifikation ist fehlgeschlagen
AuthorizationFailed	3	Die Autorisierung ist fehlgeschlagen
CertificateInvalid	4	Das Zertifikat ist ungültig oder nicht mit der NTS-Spezifikation konform
AccessKeyInvalid	5	Der Zugriffsschlüssel ist ungültig
MacAlgoNotSupported	6	Der MAC-Algorithmus wird nicht unterstützt oder die entsprechende Liste enthält keine unterstützten MAC-Algorithmen
KeyEncryptAlgoNotSupported	7	Der keyEncrypt-Algorithmus wird nicht unterstützt oder die entsprechende Liste enthält keine unterstützten asymmetrischen Algorithmen
ContentEncryptAlgoNotSupported	8	Der contentEncrypt-Algorithmus wird nicht unterstützt oder die entsprechende Liste enthält keine unterstützten symmetrischen Algorithmen
MacInvalid	9	Die empfangene NTS-MAC ist ungültig
UnknownError	10	Ein unbekannter und unbehandelter Fehler ist aufgetreten (z.B. Programmfehler)
InvalidNtpConstruction	11	Der Aufbau der NTP-Erweiterungsfelder ist ungültig (z.B. Zeitnachricht ohne NTS-MAC)
NtsMessageCorrupted	12	NTS-Nachricht ist korrupt (fehlerhafte Signatur oder Dekodierungsfehler)
NtsMessageInvalid	13	Die NTS-Nachricht ist ungültig (Nachricht ist korrupt oder nicht mit der NTS-Spezifikation konform)
Reserved1	14	Reserviert
Reserved2	15	Reserviert

Anmerkung: Rot markierte Einträge werden derzeit nicht verwendet und können sich noch ändern oder wegfallen.

Zusätzlich ist zu beachten, dass der Fehlercode während der NTS-Verhandlungsphase nicht gesichert übertragen wird, d.h. nicht von der Signatur abgedeckt ist. Ein Angreifer kann diesen Wert somit manipulieren. Aus diesem Grund dient dieser Wert ausschließlich zu Diagnosezwecken. Eine Ausnahme liefern die Zeitnachrichten, in denen das erste Erweiterungsfeld (mit der NTS-Nachricht) komplett durch den MAC geschützt wird. Des Weiteren ist anzumerken, dass zukünftig auch andere Definitionen der Fehlercodes existieren können. Hierbei könnte ein definierter Wert einen Fehler repräsentieren, wobei $2^{16} - 1$ Werte (1 bis 65537) dargestellt werden können.

3.6.7 Zulässige und sinnvolle Paketgrößen bei NTS

Die in NTS verwendeten Paketgrößen schwanken je nach Nachrichtentyp und dessen Inhalt. Einfache Zeitanfragen und -antworten erzeugen nur geringe Mengen an Nutzdaten auf NTS-Ebene und liegen im Bereich von etwa 100 Bytes. Zugriffsanfragen liegen noch weit darunter und sind im Schnitt um den Faktor fünf kleiner. Allerdings gibt es in den Assoziationsnachrichten auch große Datenmengen, die im Bereich von 1 bis 10 Kilobyte oder sogar darüber liegen können. Begründet wird dies durch lange Schlüssellängen, den daraus resultierenden digitalen Signaturen, den Zertifikatsketten und auch Zertifikatssperrlisten. Eine Begrenzung der Datenmengen ist in der NTS-Spezifikation allerdings nicht definiert.

In Anwendung mit NTPv4 wird die zulässige Größe zunächst einmal nur durch das NTP-Erweiterungsfeld begrenzt, welches Datenmengen von 65.535 Bytes ($2^{16} - 1$) erlaubt. Darüber hinaus wird die Größe des gesamten NTP-Pakets, inklusive des NTS-Inhalts, durch das UDP-Transportprotokoll begrenzt, welches auch eine theoretische Größe von 65.535 Bytes zulässt. Bei dieser Größe kommt es jedoch zu einer Fragmentierung der Nachricht auf den unteren Netzwerkprotokollen, was derzeit noch Probleme mit sich bringt (siehe dazu Kapitel 6.1.2). Will man diese Fragmentierung verhindern, so stehen NTS bei einer Ethernet-Verbindung (mit IPv4), abzüglich sämtlicher *Header* und *Trailer*, maximal 1.424 Bytes pro Nachricht zur Verfügung[48]. Werden zusätzlich die NTP-Felder für die *KeyID* und des nativen *MAC* verwendet, so ist deren Größe von der maximalen NTS-Nachrichtengröße abzuziehen. Abbildung 3.21 und Abbildung 3.22 zeigen den Aufbau eines Datenframes bei einer Ethernet-Verbindung und den Nutzdatenanteil bezogen auf die jeweiligen Schichten im OSI-Referenzmodell. Der Interpacket Gap des Datenframes bezeichnet hierbei den minimalen zeitlichen Abstand zwischen zwei gesendeten Paketen auf dem Übertragungsmedium.

[48] Hierbei wird von der Standardgröße von 20 Bytes bei IPv4 ausgegangen. Bei der Nutzung aller Felder im Header (unüblich) würde die Größe von IPv4 auf 60 Bytes wachsen. Bei IPv6 ist die Größe konstant und liegt bei 40 Bytes.

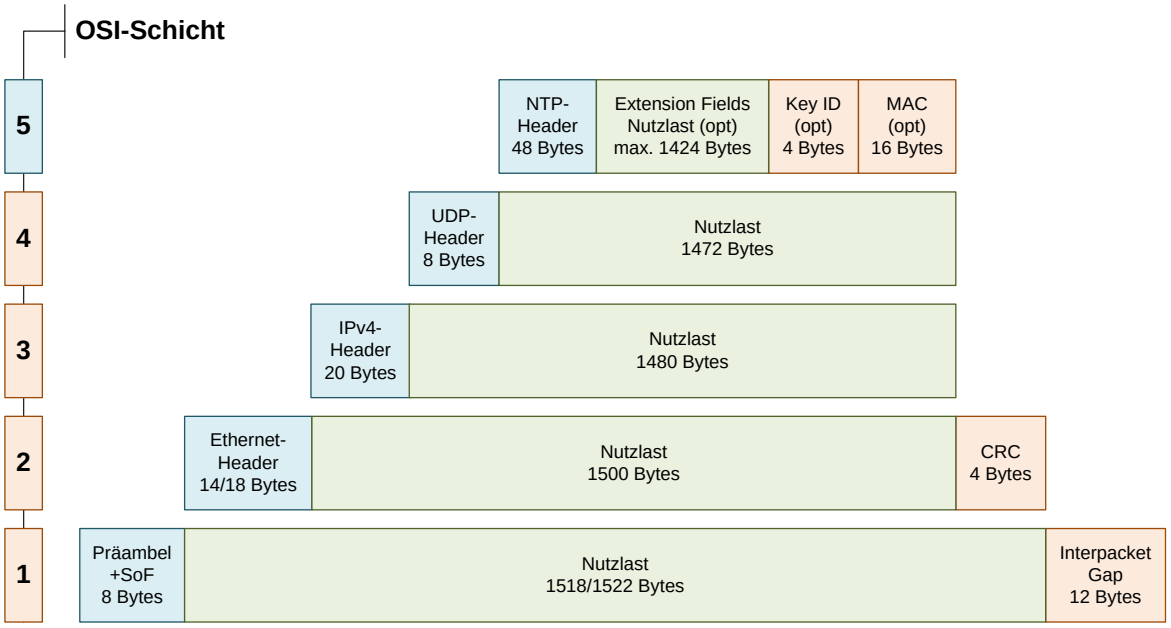


Abbildung 3.21: Herleitung der zulässigen Größe der NTP-Erweiterungsfelder

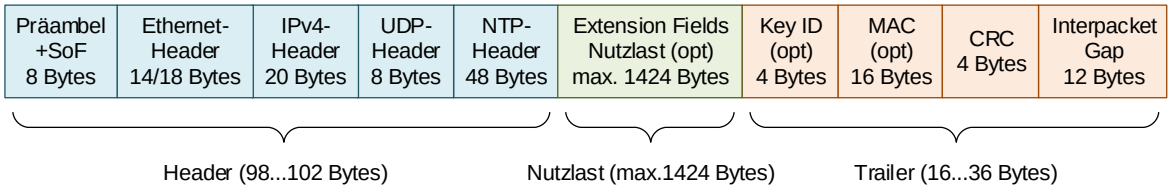


Abbildung 3.22: Zusammensetzung eines Ethernet-Datenframes mit NTPv4

3.6.8 Latenzzeiten und verursachte Ungenauigkeiten durch NTS

In diesem Abschnitt wird die durch NTS erzeugte Latenz und deren Auswirkungen auf die Genauigkeit untersucht. Hierzu erfolgt ein kurzer Vergleich mit ungesichertem NTP, um die Auswirkungen deutlich zu machen.

3.6.8.1 Latenzzeiten bei ungesichertem NTP

In NTP hängt die Genauigkeit der übertragenen Uhrzeit stark von den Paketlaufzeiten ab. Im Idealfall sind die Zeiten, die eine NTP-Nachricht im Netzwerk unterwegs ist, für Hin- und Rückweg identisch. Zur Erkennung von asymmetrischen Laufzeiten werden in NTP vier Zeitstempel verwendet (vgl. Abbildung 3.23), von denen t_1 bis t_3 mit dem NTP-Paket übertragen werden. Der Zeitstempel t_4 wird clientseitig beim Empfangen der NTP-Nachricht erzeugt. Die Berechnungen die aus diesen Werten resultieren, betrachtet Kapitel 2.2.1 näher. Die zeitliche Ungenauigkeit bei ungesichertem NTS erfolgt ausschließlich durch asymmetrische Paketlaufzeiten im Netzwerk, die als Fehler in den Offset einfließen. Die internen Rechenzeiten haben keinen Einfluss hierauf.

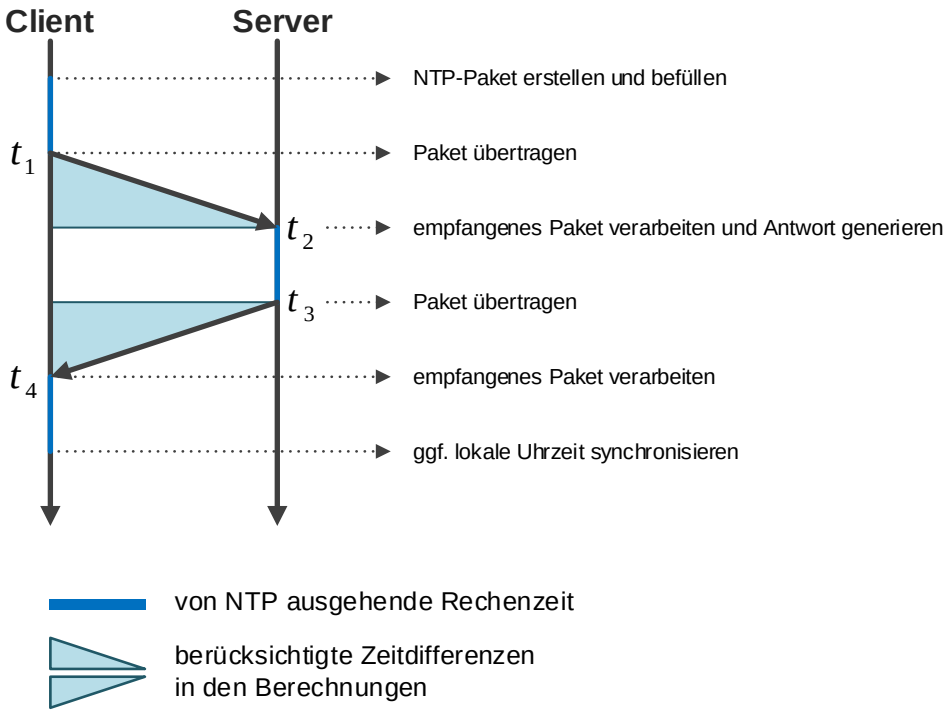


Abbildung 3.23: Zeitstempel einer NTP-Nachricht ohne NTS

3.6.8.2 Latenzzeiten bei NTS-gesichertem NTP

Die durch NTS zusätzlich verursachte Latenz wirkt sich negativ auf die Genauigkeit der Zeitdaten aus und muss somit so gering wie möglich gehalten werden. Die Ursache dafür ist bei der Generierung des MAC zu finden. Wird dieser berechnet, so darf sich ab diesem Zeitpunkt ein NTP-Paket nicht mehr verändern. Dies betrifft natürlich auch die Zeitstempel, die nach der Generierung des MAC nicht mehr aktuell sind. Dadurch fließt die Zeit, die für die Berechnung des MAC benötigt wurde, in den Delay und den Offset mit ein. Da zudem die

Verarbeitungsgeschwindigkeiten zwischen Client und Server differieren, erzeugt dies eine zusätzliche Asymmetrie und folglich ungenauere Zeitdaten. Der zeitliche Ablauf wird in der Abbildung 3.24 veranschaulicht. Die blau gekennzeichneten Pfade stellen die benötigte Rechenzeit dar, die vom NTP-Protokoll beansprucht werden. Die rot markierten Pfade stellen die Rechenzeit dar, die das NTS-Protokoll benötigt. Die Längen der jeweiligen Pfadabschnitte geben hier keinen Aufschluss über das Verhältnis der jeweiligen Rechenzeiten und sollen nur den allgemeinen Ablauf der Verarbeitungsfolge widerspiegeln. Die zusätzlichen Zeitstempel t_{1C} und t_{3C} stellen den Zeitpunkt dar, an dem die Berechnung der MAC erfolgt. Die im NTP-Paket übertragenen Zeitstempel sind nun: t_{1C} , t_2 und t_{3C}.

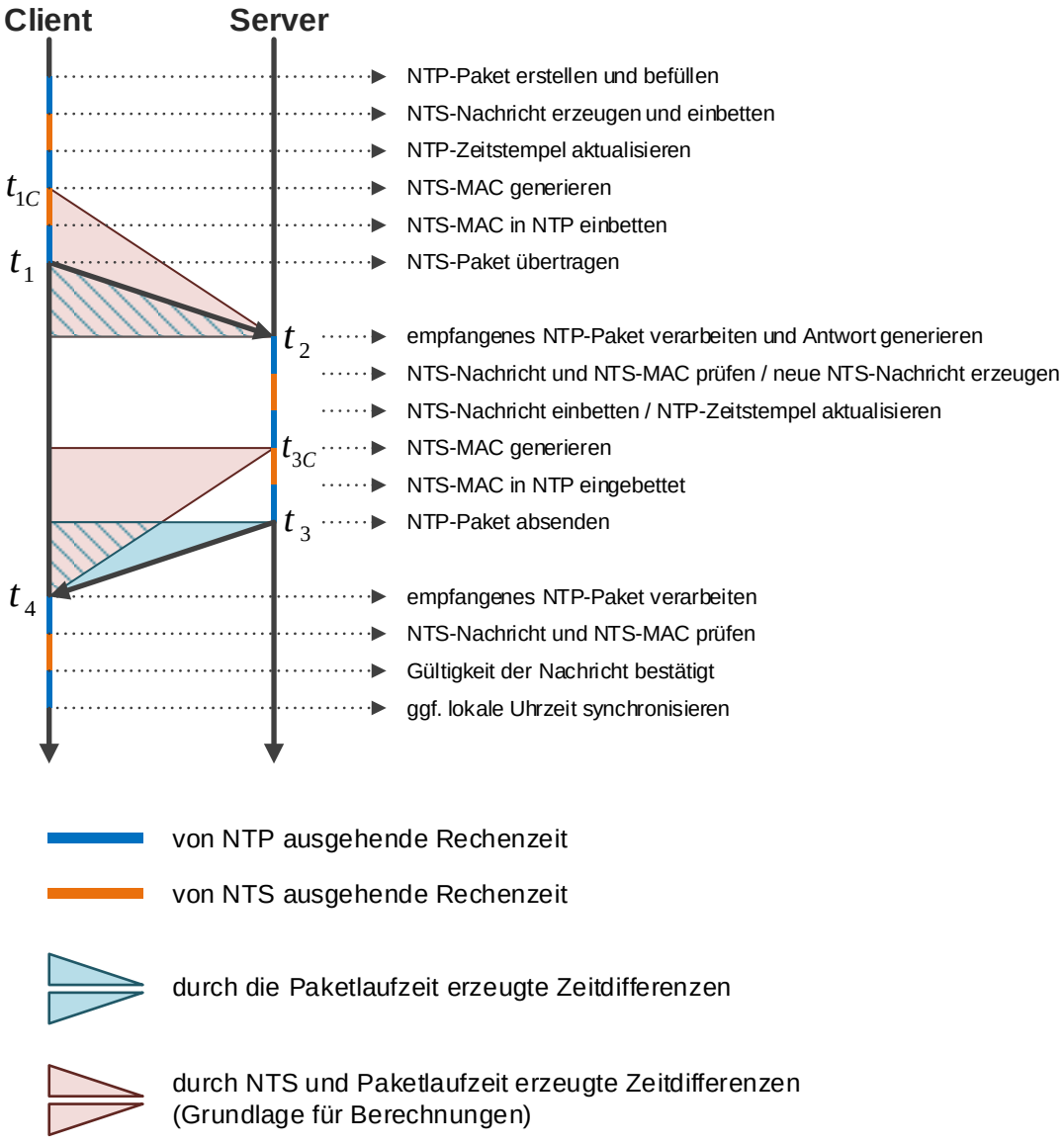


Abbildung 3.24: Zeitstempel einer NTP-Nachricht mit NTS

Der Delay und der entsprechende Offset berechnen sich nun entsprechend der beiden Gleichung 3.13 und Gleichung 3.14:

$$\delta_C = (t_4 - t_{1C}) - (t_{3C} - t_2) \tag{3.13}$$

$$\theta_C = \frac{(t_2 - t_{1C}) - (t_4 - t_{3C})}{2} \tag{3.14}$$

Durch die von NTS verursachten Beträge $t_1 - t_{1C}$ und $t_3 - t_{3C}$, entsteht ein systematischer Fehler, der sich als höhere Paketlaufzeit äußert und unabhängig vom verwendeten Netz ist. Bei Zeitsynchronisationen über das Internet, dominieren die Paketlaufzeiten im Netz, wodurch die zusätzlichen und wesentlich kleineren Beträge vernachlässigt werden können. Bei der Verwendung im lokalen Netzwerk sind die Paketlaufzeiten wesentlich geringer, wodurch die von NTS erzeugten Beträge dominieren und größere Ungenauigkeiten erzeugen, auch wenn diese sich „nur“ im Mikrosekundenbereich bewegen. Der reine von NTS erzeugte zusätzliche Offset ist von der Differenz der Berechnungszeit der MAC von Client und Server abhängig. Dieser berechnet sich gemäß der Gleichung 3.15:

$$\theta_{NTS} = \frac{(t_1 - t_{1C}) - (t_3 - t_{3C})}{2} \tag{3.15}$$

4 Implementierung von NTS

Dieses Kapitel dokumentiert die Entwicklung des NTS-Dienstes als ausführbares Computerprogramm auf Basis der im Kapitel 3 beschriebenen Spezifikation und unter Berücksichtigung der gesetzlichen Richtlinien des BSI [10, p. 19 ff.]. Die Verfügbarkeit des Dienstes ist zunächst nur Linux-Betriebssystemen vorbehalten[49] und kann zukünftig auch auf andere Betriebssysteme portiert werden. Der NTS-Dienst ist außerdem in eine geeignete NTPv4-Implementierung einzubinden, die eine externe Softwarekomponente darstellt und somit nicht durch die NTS-Implementierung abgedeckt wird. Die Beschreibung der benötigten Schritte erfolgt dabei in den folgenden Unterkapiteln. Hierfür wird zunächst auf die Entwicklungsumgebung und das Betriebssystem eingegangen, in denen die Programmierung durchgeführt wird. Anschließend erfolgen die detaillierten Beschreibungen der Softwarestruktur und den darin enthaltenen Modulen. Abschnitt 4.4 schildert anschließend den Programmfluss und die einnehmbaren Zustände, die das Verhalten des NTS-Dienstes als Ganzes darlegen. Das darauffolgende Unterkapitel geht auf die Konfigurations- und Diagnosemöglichkeiten des Dienstes ein, die das Softwareverhalten am realen System beeinflussen und lenken können. Abschließend erfolgt eine kurze Übersicht zum aktuellen Projektaufbau und zum derzeitigen Stand der Implementierung. Erreichte Ziele und diverse Statistiken sind an dieser Stelle auch aufgeführt.

4.1 Betriebssystem und Entwicklungsumgebung

In diesem Abschnitt wird auf das Betriebssystem, die integrierte Entwicklungsumgebung und die Programmierwerkzeuge eingegangen, die für die Implementierung notwendig sind oder eine sinnvolle Ergänzung darstellen. Dazu werden die einzelnen Komponenten kurz vorgestellt und deren Einsatzzweck während der Implementierung beschrieben. Die Auswahl dieser Komponenten ist nicht festgeschrieben und können durchaus variieren. Die hier aufgeführte Konstellation bezieht sich somit auf Komponenten, die als geeignet betrachtet wurden und mit denen auch die Umsetzung der Arbeit erfolgte.

4.1.1 Das eingesetzte Betriebssystem

Die Wahl eines geeigneten Linux-Betriebssystems fiel auf die *Debian*-Distribution [110]. Diese stellt hohe Anforderungen an die Robustheit und benötigt bei Verwendung der LXDE-Desktopumgebung sehr wenig Ressourcen. Daher eignet sich Debian auch hervorragend zur

[49] Dies ist durch die Programmentwicklung begründet, die auf einem Linux-System stattfindet.

Installation in einer virtuellen Maschine[50], wenn die Softwareentwicklung auf einen Computer mit *Microsoft Windows* erfolgen soll. Für die Implementierung wurde hierfür die 32-Bit-Version 8.5.0 (i386) mit der LXDE-Desktopumgebung verwendet (vgl. Abbildung 4.1).

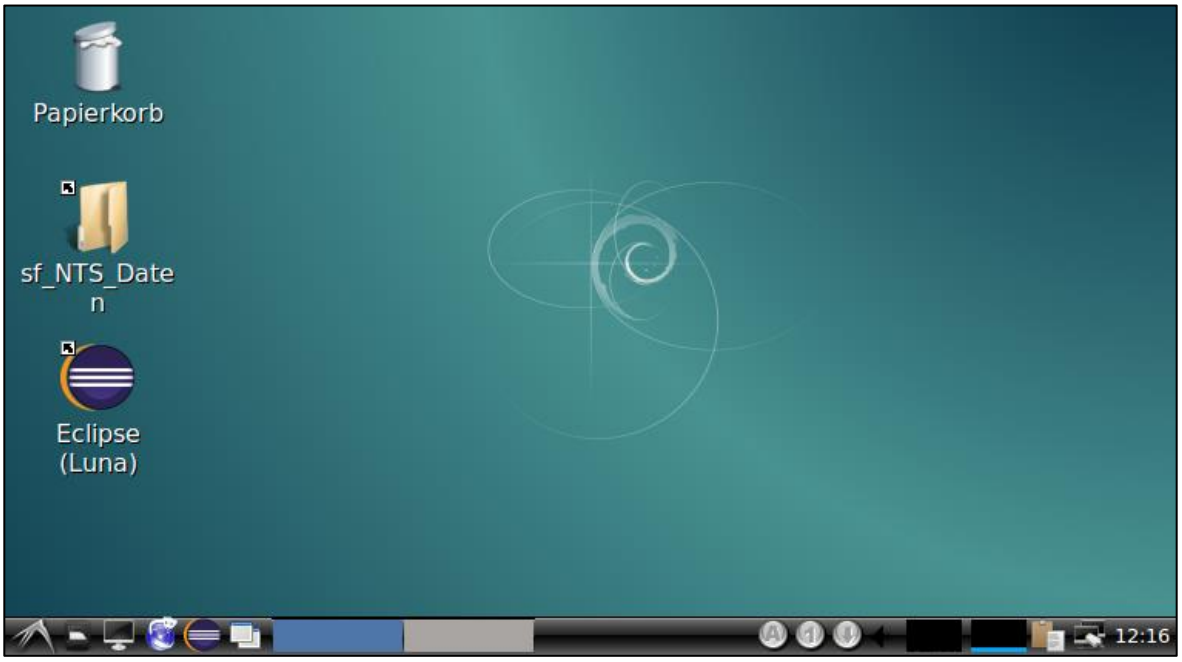


Abbildung 4.1: LXDE-Desktopumgebung unter Linux Debian v8.5.0

Debian ist im Vergleich zu anderen bekannten Distribution wie *Linux Mint* oder *Ubuntu* weniger benutzerfreundlich, stellt aber grundlegende Programmelemente bereit mit denen die Einrichtung erleichtert wird. Dies ist beispielsweise die Paketverwaltung *Synaptic*, mit der die erforderlichen Softwarekomponenten einfach und schnell installiert werden können. Eine detaillierte Anleitung zur Einrichtung des Betriebssystems, der Entwicklungsumgebung und weiterer Komponenten ist im Anlagendokument I[51] zu finden.

4.1.2 Programmiersprache und Entwicklungswerkzeuge

Die verwendete Programmiersprache: C++ 11

Die Implementierung des NTS-Dienstes erfolgt in der Programmiersprache C++ 11. Diese ist weit verbreitet und benötigt im Gegensatz zu Java keine Laufzeitumgebung zur Ausführung des Dienstes. Jenes wäre zudem schwer realisierbar, da eine Java-basierte Protokollentwicklung nicht die Geschwindigkeitsanforderungen[52] erfüllen würde und wohl kaum mit

[50] z.B. *Virtual PC*, *VMware* oder das hier verwendete *VirtualBox*
[51] Anlage I: „Installation und Einrichtung der Eclipse-Entwicklungsumgebung unter Linux in einer virtuellen Maschine“
[52] Java ist eine Interpretersprache und somit kategorisch ungeeignet, um performante und zeitkritische Anwendungen zu erzeugen

den Anforderungen einer NTP-Implementierung konform ist. Gegenüber der etablierten Programmiersprache C bietet C++ den Vorteil der objektorientierten Programmierung, wodurch die modulare Implementierung vereinfacht wird. Die Abwärtskompatibilität von C++ zu C erlaubt zudem eine bessere Integrierung des NTS-Dienstes in eine existierende NTP-Implementierung, da diese häufig in C geschrieben sind. Unter *Linux Debian* sind dafür die beiden Compiler *gcc* und *g++* empfehlenswert, die zur Umsetzung dieser Arbeit auch verwendet werden.

Die integrierte Entwicklungsumgebung: Eclipse CDT

Neben den Compilern wird natürlich auch eine geeignete Entwicklungsumgebung benötigt, um die Programmieraufgaben umzusetzen. Sowohl unter *Microsoft Windows*, als auch unter *Linux* oder *Apple Macintosh* gibt es eine Vielzahl an derartigen Editoren und Entwicklungswerkzeugen. Die Wahl unter Linux fiel hierbei auf *Eclipse CDT* (Luna) v4.4.2 [111] welches eine ausgereifte Entwicklungsumgebung darstellt und deren Funktionsumfang durch sogenannte *Plugins* erweitert werden kann (vgl. Abbildung 4.2). Dies CDT-Version (*C/C++ Development Tooling*) ist hierbei auf die Entwicklung mit C/C++ ausgelegt und beinhalt bereits entsprechende Komponenten zur Entwicklung und Kompilierung des Quellcodes.

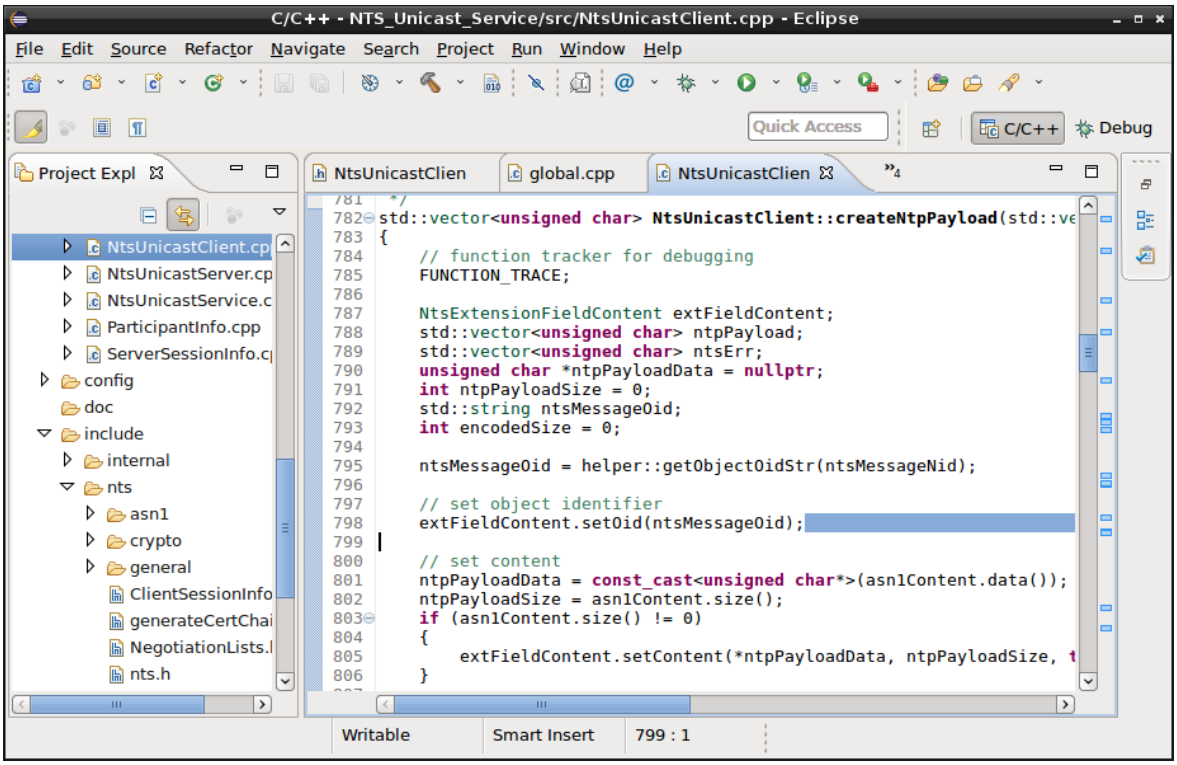


Abbildung 4.2: Programmoberfläche der Eclipse-CDT-Entwicklungsumgebung

Eclipse-Erweiterung: Valgrind

Eine sehr sinnvolle Erweiterung für die Eclipse-Entwicklungsumgebung stellt das Plugin *Valgrind* [112] dar, mit deren Unterstützung das entwickelte Programm auf bestimmte dynamische Fehler analysiert werden kann. Es wird als *Debugger* und *Profiler* klassifiziert und ermöglich somit auch eine Leistungsanalyse des Programms zur Laufzeit. Die wichtigste Eigenschaft stellt während der Implementierung die Entdeckung von Speicherlecks und Pufferüberläufen dar, die den Programmablauf und dessen Stabilität beeinträchtigen können. Pufferüberläufe stellen zudem erhebliche Sicherheitsrisiken dar, die durch Angreifer ausgenutzt werden können, um empfindliche Daten abzugreifen oder Fremdcode auszuführen.

Eclipse-Erweiterung: LogViewer

Auch der sogenannte *LogViewer* ist eine praktische Erweiterung und eine sinnvolle Ergänzung zur klassischen Konsolenausgabe. Diese Erweiterung überwacht den Inhalt benutzerdefinierter Dateien und gibt diese in einem separaten Fensterelement von Eclipse aus. Somit können Warn- und Fehlermeldungen zur Programmlaufzeit in separate Logdateien umgeleitet und von der Konsolenausgabe getrennt werden. Zusätzlich erlaubt diese Erweiterung die farbliche Hervorhebung von Zeilen, indem sie enthaltene Schlüsselwörter – die selbst definiert werden können – sucht und entsprechend der Konfiguration darstellt (vgl. Abbildung 4.3).

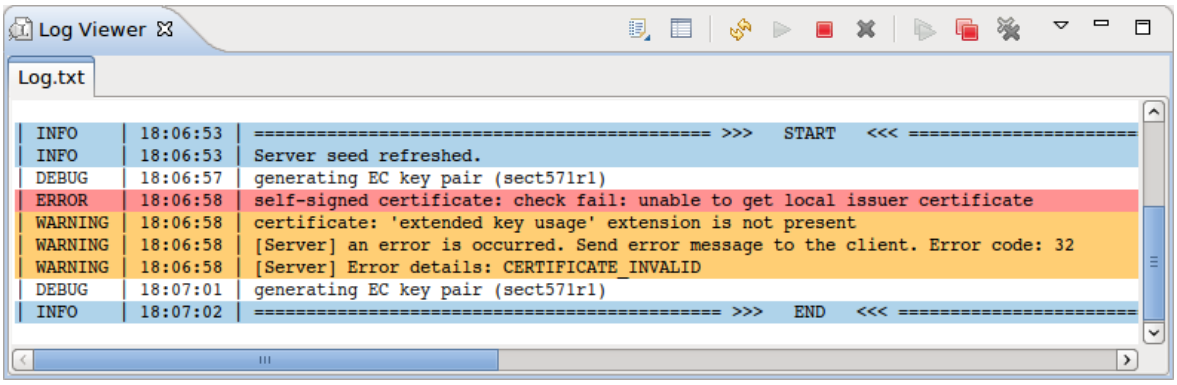


Abbildung 4.3: Beispielhafte Ausgabe des Eclipse-Plugins "Log Viewer"

Inline Code-Dokumentation mit **Doxygen**

Eine gute Dokumentation der Software ist eine wichtige Grundlage zur späteren Weiterentwicklung bzw. Code-Pflege durch den Urheber oder Dritte. Aus diesem Grund ist der Einsatz des Dokumentationsprogramms *Doxygen* [113] sinnvoll, welches die von C++ gegebene Syntax um spezielle Befehle erweitert. Diese werden als Kommentare in den Code integriert, ohne den eigentlichen Programmcode zu beeinflussen. Doxygen generiert anhand dieser Kommentare eine umfangreiche Dokumentation, deren Zielformat (z.B. PDF-Datei, *LaTeX*, Webseiten) frei wählbar ist. Dabei werden auch dynamisch verknüpfte Grafiken erzeugt, welche die Relationen der einzelnen Programmteile zueinander darstellen. Zur internationalen Verständlichkeit von Software wird die Verwendung einer englischsprachigen Dokumentation empfohlen. Diese wird für Implementierung des NTS-Dienstes auch angewendet.

4.1.3 Externe Bibliotheken

Neben der Entwicklungsumgebung und den Programmierwerkzeugen sind für die Implementierung von NTS auch externe Bibliotheken nötig, die den Programmieraufwand verringern und diverse Funktionserweiterungen liefern. Für die NTS-Entwicklung müssen geeignete Bibliotheken genutzt werden, die zusammen folgende Funktionen abdecken:

- grundlegende ASN.1-Funktionen
  - Austausch von Daten zwischen heterogenen Systemen
  - Bereitstellung entsprechender Kodierer (BER- und DER-Kodierer)

- Bereitstellung und Verarbeitung vom CMS-Strukturen
- Verarbeitung und Generierung von X.509-Zertifikaten
- kryptografische Funktionen
  - Generierung von Schlüsselpaaren (RSA und ECC)
  - Generierung sicherer Zufallszahlen
  - symmetrische Ver- und Entschlüsselung von Daten
  - asymmetrische Ver- und Entschlüsselung von Daten
  - Erstellung und Prüfung von digitalen Signaturen

Neben den geforderten Grundfunktionen sind durch die Bibliotheken auch weitere Eigenschaften zu erfüllen. Somit müssen diese mit der verwendeten Programmiersprache kompatibel sein, wodurch sie entweder entsprechende APIs[53] bereitzustellen haben oder das direkte

[53] APIs sind Schnittstellen (Funktionsaufrufe) zu externen Bibliotheken, die vor der Verwendung des kompilierten Programms im jeweiligen Betriebssystem installiert werden müssen.

Einbinden in das NTS-Projekt unterstützen müssen. Des Weiteren ist die Plattformunabhängigkeit zu gewährleisten, um den NTS-Dienst auf andere Betriebssysteme portieren zu können. Hierbei ist neben Linux insbesondere das Betriebssystem *Microsoft Windows* gemeint. Ferner sollte auch eine Portierung auf *Apple Macintosh* oder *Android* möglich sein, wobei dies zurzeit keine Priorität darstellt. Zusätzlich ist bei der Wahl der Bibliotheken auf den Umfang der unterstützten kryptografischen Verfahren zu achten. Diese müssen mindestens die in NTS-spezifizierten Sicherheitsmaßstäbe einhalten und sollten zudem die lokalen oder gesetzlichen Sicherheitsrichtlinien erfüllen. Dies umfasst beispielsweise verfügbare MAC- und Hash-Algorithmen oder eine Auswahl an diversen elliptische Kurven. Bei der Wahl der kryptografischen Bibliothek ist außerdem die Aktualität und die Softwarepflege von großer Bedeutung. Veraltete Bibliotheken, in denen bekannte und nicht korrigierte Sicherheitslücken enthalten sind, eigenen sich nicht zur Verwendung in der NTS-Implementierung.

4.1.3.1 Die ausgewählte ASN.1-Bibliothek: asn1c

Die Wahl einer geeigneten ASN.1-Bibliothek zur Implementierung der NTS-Nachrichtenobjekte erwies sich als schwierig. Folgende ASN.1-Bibliotheken stehen hierbei zur Auswahl, die für die Programmiersprache C/C++ geeignet sind:

- asn1c [114]
- ASN1C (Objective Systems) [115]
- FFASN1 [116]
- Libasn1 [117]
- OpenSSL [118]

Bei der Suche zeigen diese Bibliotheken allerdings verschiedene Schwächen, die mehr oder weniger ausgeprägt sind. Bei *OpenSSL* und *FFASN1* gibt es somit nahezu keine Dokumentation, die auf die korrekte Verwendung der Bibliothek schließen lässt. Die Einarbeitungszeit ist dementsprechend hoch und funktioniert teilweise nur über die *Trial-and-Error*-Methode. Mögliche Aktualisierungen der Bibliothek können zudem zu einem hohen Wartungsaufwand führen. *ASN1C* von *Objective Systems* scheint hingegen sehr ausgereift zu sein, wobei der angebotene ASN.1-Compiler kostenpflichtig ist und aus diesem Grund von der Wahl ausgeschlossen wird. *Libasn1* und *asn1c* bieten im Vergleich die besten Bedingungen, da beide Bibliotheken ausreichend dokumentiert sind und frei verwendet werden können. Allerdings existieren hierfür keine oder keine aktiven Foren bzw. Plattformen, um online Unterstützung zu erhalten. Die Wahl für die NTS-Implementierung benötigte ASN.1-Bibliothek fiel nach kurzer Einarbeitung auf *asn1c* in der Version 0.9.27 vom 17.09.2014. Diese Bibliothek beinhaltet noch einige Softwarefehler, die allerdings unkritisch sind. Der Grund

für diese Entscheidung war die gute Dokumentation und die recht einfache Installation. Außerdem stellt *asn1c* einen Compiler bereit, der die verwendete ASN.1-Syntax in C-Code umwandeln kann. Dieser kann somit direkt in die Implementierung eingebettet werden, ohne dass API-Aufrufe vonnöten sind. Dadurch muss *asn1c* nicht als statische oder dynamische Programmbibliothek auf dem Betriebssystem zur Verfügung stehen, um die Ausführung zu ermöglichen (vergleichbar mit DLL-Dateien unter Microsoft Windows). Die Anwendung des bereitgestellten Compilers und die Nutzung der Bibliothek beschreibt Kapitel 4.2.1 näher.

4.1.3.2 Die ausgewählte Kryptografie-Bibliothek: OpenSSL

Die Wahl der Bibliothek zur Verwendung von kryptografischen Funktionen in NTS fällt wesentlich einfacher aus. Hierfür wird die OpenSSL-Bibliothek in der derzeitigen Version 1.1.0 vom 25.08.2016 verwendet [118]. OpenSSL bietet ein großes Arsenal an kryptografischen Funktionen in den verschiedensten Bereichen an, wie etwa für CMS, X.509-Zertifikate oder auch TLS. Die Online-Community ist groß und die Software wird in regelmäßigen Abständen aktualisiert und verbessert. Nachteilhaft von OpenSSL ist die lückenhafte Dokumentation, die einige Funktionen gar nicht beschreibt. Abhilfe schafft hier nur die intensive Suche in Foren oder die Analyse des zusätzlich schlecht kommentierten OpenSSL-Quellcodes. Im Gegensatz zur *asn1c*-Bibliothek wird OpenSSL in der Regel über API-Funktionen angesprochen, wodurch OpenSSL auf dem jeweiligen Betriebssystem verfügbar sein muss. Neben dieser gibt es noch viele weitere Bibliotheken, die unterschiedlichen Merkmale aufweisen und potentiell geeignet wären. Eine gute Gegenüberstellung dieser Bibliotheken kann in Wikipedia unter dem Begriff *Comparison of TLS implementations* [119] nachgeschlagen werden. Da OpenSSL als externe Bibliothek in die Implementierung eingebunden wird, sind folgende Linker-Flags im Projekt der Entwicklungsumgebung zu setzen: „*-lcrypto -lssl*“, um auf die kryptografischen Funktionen zugreifen zu können.

4.2 Softwarestruktur und Programmmodule

Dieser Abschnitt beschreibt den strukturellen Aufbau der NTS-Implementierung und die logische Verknüpfung der jeweiligen Module miteinander. Dazu werden in den folgenden Unterkapiteln alle entwickelten Module einzeln betrachtet und deren Funktionsweise näher dargestellt. Der Aufbau der Implementierung verfolgt generell einen hierarchischen Ansatz, in dem die enthaltenen Module aufeinander aufbauen (vgl. Abbildung 4.4). An der Spitze steht hierbei das Hauptprogramm in Form einer ausführbaren Datei. Dieses kommuniziert mit einem enthaltenen NTP-Dienst, welcher entweder implementiert oder als vorhandene Bibli-

othek eingebettet wird. Die NTP-Implementierung wird an dieser Stelle als Blackbox betrachtet, da die interne Verarbeitungsroutine ohnehin für NTS nicht relevant ist. Der NTP-Dienst muss nun seinerseits die NTS-Implementierung inkludieren, die in Form von zusätzlichen Quellcodedateien in das jeweilige Projekt eingebunden wird. Die Kommunikation zwischen NTP und NTS erfolgt über eine separate Schnittstelle, die das Kapitel 4.3 weiter beschreibt. Den Einstiegspunkt in NTS stellt das *NTS-Unicast-Service*-Modul dar, welches die höchste Hierarchieebene des Dienstes darstellt. Dieses analysiert die Erweiterungsfelder der übergebenen NTP-Pakete und leitet enthaltene NTS-Unicast-Nachrichten entsprechend an das *Client-Unicast*-Modul oder das *Server-Unicast*-Modul weiter. Zusätzlich greift dieses Modul auf externe Ressourcen zu, um Konfigurationsdateien zu laden oder Protokolle zu speichern. Die genannten Client- und Servermodule stellen eine tiefere Hierarchieebene dar, welche die jeweiligen Nachrichten verarbeiteten und neue generieren. Die dazu benötigten Grundfunktionen befinden sich unter anderem im ASN.1- und Kryptografiemodul. Diese stellen im Gegensatz zu dem Client- und Servermodul keine in sich geschlossenen Module im engeren Sinne dar, sondern enthalten lose Gruppen von Verarbeitungsroutinen. Diese Routinen stellen die kleinsten Programmeinheiten dar und greifen (im Falle von kryptografischen Funktionen) über APIs auf externe Bibliotheken zu.

4.2.1 Das ASN.1-Modul

ASN.1-Funktionen sind ein wichtiger Bestandteil des NTS-Dienstes und werden zur Übertragung der Nutzdaten zwischen den kommunizierenden Systemen benötigt. Die im ASN.1-Modul enthaltenen Funktionen werden dabei ausschließlich für die NTS-Nachrichtenobjekte eingesetzt. Die CMS-Strukturen, welche auch über ASN.1 definiert sind, werden von der OpenSSL-Bibliothek verarbeitet. Dieses bietet native Funktionen zum Parsen und Befüllen jener Strukturen an, wodurch eine zusätzliche Implementierung überflüssig ist. Um die korrekte Funktionalität der asn1c-Bibliothek zu gewährleisten, muss diese im *Workspace* von Eclipse liegen. Hierbei handelt es sich um den Arbeitsordner, in welchem sich das Projekt zur NTS-Implementierung befindet. Wird hingegen außerhalb des Arbeitsverzeichnisses auf die asn1c-Bibliothek referenziert, so kommt es während der Kompilierung des NTS-Projekts zu diversen Fehlern. Die Einteilung der im ASN.1-Modul enthaltenen Funktionen erfolgt in zwei Gruppen, die im weiteren Verlauf des Kapitels beschrieben werden. Hierbei handelt es sich um allgemeine Grundfunktionen und um die NTS-Nachrichtenobjekte.

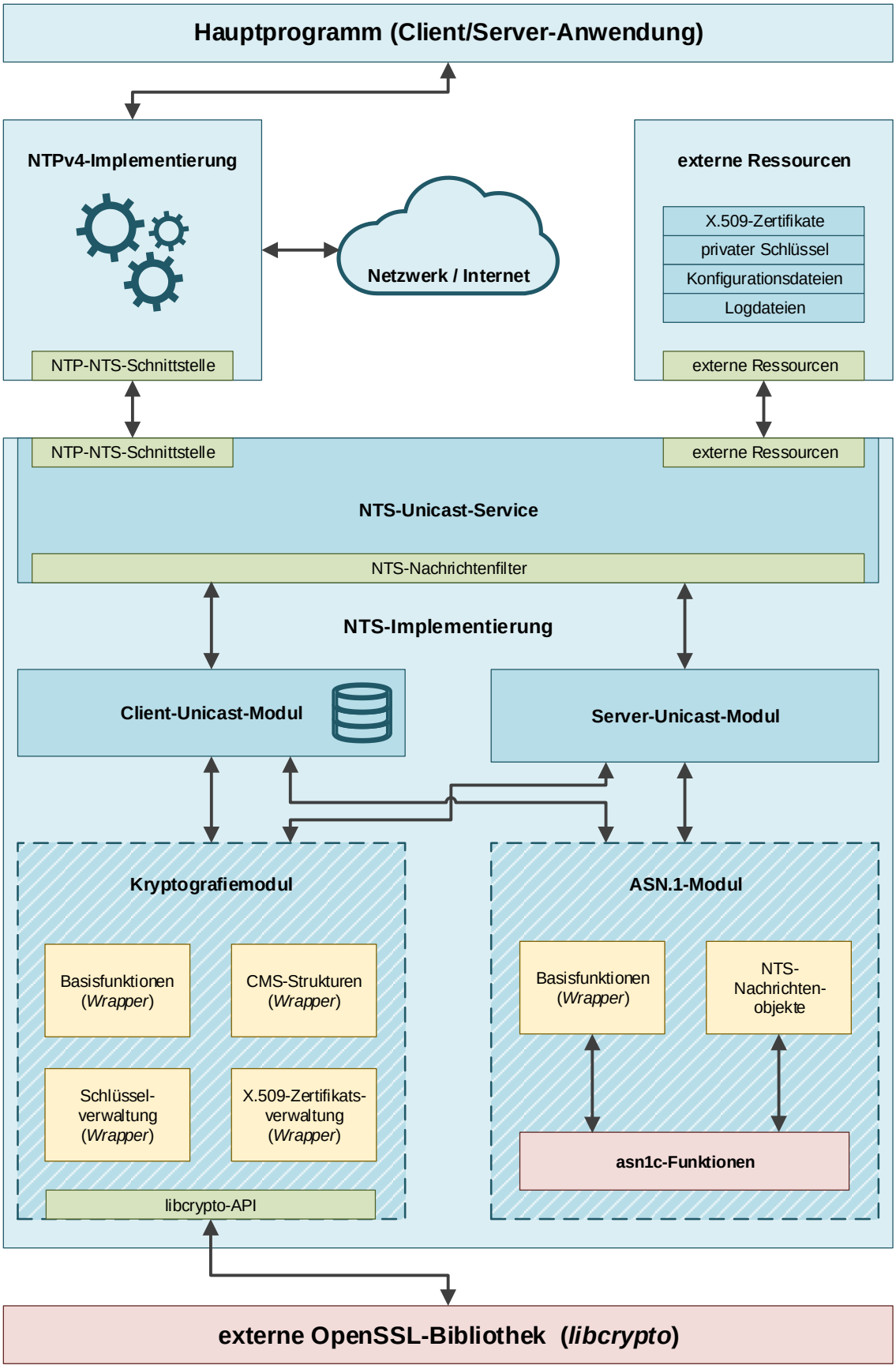


Abbildung 4.4: Struktureller Aufbau der NTS-Implementierung für den Unicast-Betrieb

4.2.1.1 ASN.1-Basisfunktionen (Wrapper)

Die implementierten ASN.1-Grundfunktionen sind sogenannte *Wrapper-Funktionen*. Diese umhüllen die von der asn1c-Bibliothek bereitgestellten Funktionen und können als eine Art Adapter betrachtet werden. Ein wesentlicher Grund für diesen Schritt ist Inhomogenität der Quellcodes, da die genutzte Bibliothek in C++-kompatiblem C-Code[54] verfasst wurde. Die dort eingesetzte Programmiersprache und der vorherrschende Programmierstil weichen signifikant von der eigenen Implementierung ab. Eine direkte Nutzung würde zu einem höheren Wartungsaufwand und zu schlecht lesbarem Code führen. Die Umhüllung der Bibliotheksfunktionen in separate Methoden bietet eine saubere Trennung mit definierten Schnittstellen. Somit würde eine Aktualisierung oder der Austausch der Bibliothek nur die Schnittstellen (den Wrapper) betreffen und nicht den kompletten Quellcode. Zusätzlich können durch die implementierten Wrapper die Bibliotheksfunktionen in gewisser Weise erweitert oder angepasst werden. Die dabei umgesetzten Funktionen richten sich hierbei nach den verwendeten ASN.1-Typen, die in den NTS-Nachrichtenobjekten zur Anwendung kommen. Für folgende ASN.1-Typen wurden derzeit Wrapper implementiert:

- ANY
- INTEGER
- OCTET STRING
- OBJECT IDENTIFIER
- AlgorithmIdentifier [120]

Neben den Basistypen sind auch folgende ASN.1-Kodierungsverfahren im ASN1-Modul enthalten:

- BER-Dekodierer
- DER-Kodierer
- XER-Kodierer

Die Implementierungen des BER-Kodierers und des DER-Dekodierers sind überflüssig, da ein BER-Dekodierer auch DER-kodierte Daten verarbeiten kann (s.a. Kapitel 2.3). Der XER-Kodierer dient in NTS ausschließlich zu Diagnosezwecken und ermöglicht die Darstellung vom Daten in Klartext, die hierbei im XML-Format strukturiert werden. Das folgende Beispiel zeigt eine Ausgabe, nachdem ein fiktiv befüllten NTS-Nachrichtenobjekt (*ClientAssocData*) mit einem XER-Kodierer verarbeitet wurde:

[54] Hiermit ist normaler C-Code gemeint, der über die Schlüsselwortkonstruktion *extern "C" {...}* in den C++-Quellcode integriert und separat mit einem C-Compiler übersetzt wird. Die ermöglicht die Verwendung von C-Code in einem C++-Projekt.

Listing 4.1: XER-kodiertes NTS-Nachrichtenobjekt (ClientAssocData)

```
<ClientAssocData>
    <accessKey>
        41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 00
</accessKey>
    <nonce>
        31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 00
    </nonce>
    <minVersion>1</minVersion>
    <macAlgos>
        <AlgorithmIdentifier>
            <algorithm>1.3.2.6.214.6</algorithm>
        </AlgorithmIdentifier>
    </macAlgos>
    <keyEncAlgos>
        <AlgorithmIdentifier>
            <algorithm>1.3.2.6.214.6</algorithm>
            <parameters>
                04 14 74 68 69 73 20 69 73 20 70 61 72 61 6D 65
                74 65 72 20 32 00
            </parameters>
        </AlgorithmIdentifier>
    </keyEncAlgos>
    <contentEncAlgos>
        <AlgorithmIdentifier>
            <algorithm>1.4.2.5.7.34.7.3</algorithm>
        </AlgorithmIdentifier>
        <AlgorithmIdentifier>
            <algorithm>1.6.34.7.9</algorithm>
        </AlgorithmIdentifier>
    </contentEncAlgos>
</ClientAssocData>
```

4.2.1.2 Der Aufbau der NTS-Nachrichtenobjekte

Die implementierten ASN.1-Basisfunktionen werden ausschließlich zur Verarbeitung der NTS-Nachrichtenobjekte eingesetzt. Diese Nachrichtenobjekte werden als C++-Klassen definiert und bestehen aus jeweils zwei Komponenten. Dies ist zum einen die Kernstruktur und zum anderen die entsprechende Schnittstelle, die in Form von Klassenmethoden implementiert wird. Der Zugriff auf die Kernstruktur erfolgt in der Regel über die ASN.1-Basisfunktionen, die als vererbte Klasse in das jeweilige NTS-Nachrichtenobjekt eingebunden wird. Dies kann allerding auch direkt erfolgen, wobei der Zugriff in solch einem Falle meist über private Klassenmethoden erfolgt (vgl. Abbildung 4.5). Der direkte Zugriff geschieht immer dann, wenn auf spezielle Elemente der Kernstruktur zuzugreifen ist (z.B. enthaltene Listen für Algorithmen). Die Aufgaben der Klassenmethoden erstrecken sich über das Lesen, Schreiben, Prüfen und Kodieren[55] der Kernstruktur. Die Überprüfung umfasst an dieser

[55] Die Kodierung ist die Serialisierung des NTS-Nachrichtenobjekts in einem ASN.1-Datenstrom, durch Verwendung eines geeigneten Verfahrens (BER- oder DER-Kodierung)

Stelle eine umfangreiche Konsistenzprüfung aller Strukturelemente. So werden beispielsweise enthaltene Objektkennungen und Datenblöcke auf Gültigkeit geprüft. Objektkennungen (OIDs) müssen entsprechend der Punktnotation nach einem definierten Schema aufgebaut sein, um als gültig betrachtet zu werden [121]. Datenblöcke werden hingegen auf Plausibilität geprüft, indem verwendete Datenblöcke zugreifbar und größer als null Bytes sein müssen. Je nach Element des NTS-Nachrichtenobjekts beinhaltet die Prüfung auch die Untersuchung des Inhalts, bei dem leere Datenblöcke als ungültig angesehen werden. Hierbei gilt ein Datenblock als leer, wenn dieser größer als null Bytes ist und alle enthaltene Bytes den Wert Null repräsentieren.

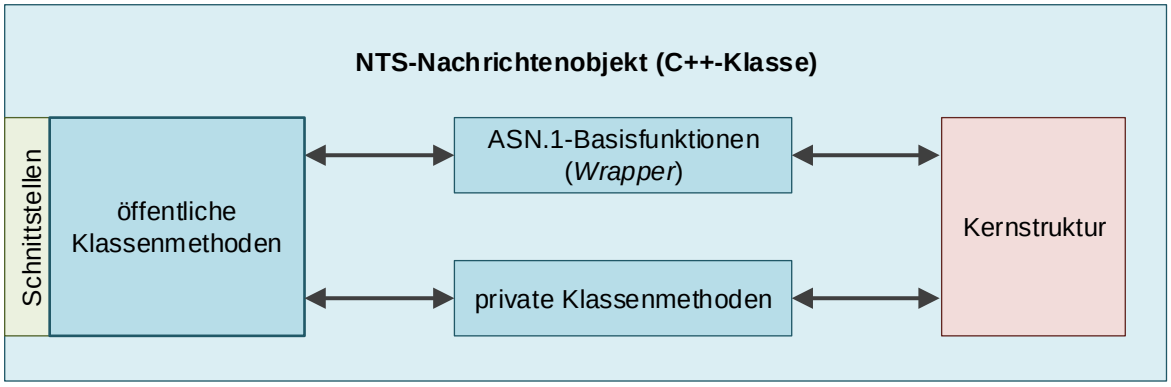


Abbildung 4.5: Aufbau eines NTS-Nachrichtenobjekts als C++-Klasse

Die genannte Kernstruktur besteht aus verschieden Elementen und Substrukturen, die von der verwendeten ASN.1-Stuktur abhängig ist. Deren Größe und Tiefe ist hierbei nicht begrenz und kann zu einer hohen Anzahl an benötigten Klassenmethoden führen. Um eine Kernstruktur zu generieren, wird ein ASN.1-Modul[56] oder eine ASN.1-Struktur benötigt. Als nachfolgendes Beispiel wird hierzu die ASN.1-Struktur des *ClientAssocData*-Nachrichtenobjekts verwendet. Diese ist gemäß der ASN.1-Syntax wie folgt aufgebaut:

Listing 4.2: ASN.1-Beschreibung der NTS-Struktur "ClientAssocData"

```
ClientAssocData ::= SEQUENCE {
      accessKey OCTET STRING (SIZE(16)),
      nonce OCTET STRING (SIZE(16)),
      minVersion INTEGER,
      macAlgos SET OF AlgorithmIdentifier,
      keyEncAlgos SET OF AlgorithmIdentifier,
      contentEncAlgos SET OF AlgorithmIdentifier
}
```

[56] Ein ASN.1-Modul ist in diesem Falle eine komplexe ASN.1-Struktur, die aus mehreren Substrukturen besteht.

In diesem Beispiel wurde die ASN.1-Struktur durch Substitution vereinfacht, indem die Elemente wie *NTSNonce* bereits durch den hinterlegten Wert (*OCTET STRING (SIZE(16))*) ersetzt wurden. Die komplette Struktur aller NTS-Nachrichtenobjekte sind dazu im Anlagendokument II[57] einsehbar. Zur Umwandlung der ASN.1-Syntax muss diese Struktur zunächst in eine Datei gespeichert werden. Hierbei handelt es sich um eine einfache Textdatei, die den sinnvollen Namen *ClientAssocData.asn1* erhält. Für das weitere Vorgehen muss nun der *asn1c*-Compiler auf dem jeweiligen System installiert sein. Ist dies der Fall, so ist die Generierung der Kernstruktur durch Eingabe des folgenden Befehls in Konsole zu tätigen:

Listing 4.3: Konsolenbefehl zur Konvertierung der ASN.1-Struktur in C-Code

```
asn1c ./ClientAssocData.asn1
```

Als Generierungsergebnis erhält man verschiedene Header- und Quellcodedaten in der Programmiersprache C. Diese Dateien beinhalten die ASN.1-Grundfunktionen (die Datentypen), sowie die benötigte Kernstruktur (vgl. Listing 4.4).

Listing 4.4: Von asn1c erzeugte Quellcode-Dateien nach der Konvertierung

```
./ANY.h
./ANY.c
./INTEGER.h
./INTEGER.c
./OBJECT_IDENTIFIER.h
./OBJECT_IDENTIFIER.c
./OCTET_STRING.h
./OCTET_STRING.c
./ber_decoder.h
./ber_decoder.c

[...]

./ClientAssocData.h
./ClientAssocData.c
```

Wird anstatt einer einzelnen ASN.1-Stuktur ein komplettes Modul in der Datei *ClientAssocData.asn1* gespeichert, so erzeugt der Compiler für jede enthaltene Struktur eine eigene Header- und Quellcodedatei. Die erzeugten Strukturen sind vom Datentyp *struct*, welche weitere Unterstrukturen beinhalten. Der Zugriff auf die enthaltenen Elemente durch entsprechende Funktionen wird durch die Generierung nicht abgedeckt und erfolgt ausschließlich über die eigene Implementierung. Die folgende Darstellung zeigt die aus der *ClientAssocData* generierte Kernstruktur in einer Baumansicht, um einen besseren Überblick der enthaltenen Daten zu erhalten:

[57] Anlage II: „ASN.1-Modulbeschreibungen in der NTS-Implementierung“

Listing 4.5: Baumansicht des konvertierten NTS-Nachrichtenobjekts (ClientAssocData)

```
ClientAssocData    (struct)
{
    accessKey      (OCTET STRING)
    .   buf        (uint8*)
    .   size       (int)
    .   _asn_ctx   (struct)
    .
    nonce          (OCTET STRING)
    .   buf        (uint8*)
    .   size       (int)
    .   _asn_ctx   (struct)
    .
    minVersion     (INTEGER)
    .   buf        (uint8*)
    .   size       (int)
    .
    macAlgos
    .   list       (struct)
    .   array      (AlgorithmIdentifier**)
    .   .   array
    .   .   .   algorithm      (OBJECT IDENTIFIER)
    .   .   .   .   buf        (uint8*)
    .   .   .   .   size       (int)
    .   .   .   .
    .   .   .   parameters     (ANY*)
    .   .   .   .   buf        (uint8*)
    .   .   .   .   size       (int)
    .   .   .   .   _asn_ctx   (struct)
    .   .   .   .
    .   .   .   _asn_ctx       (struct)
    .   .   .
    .   .   count  (int)
    .   .   size   (int)
    .   .   free   (void(*)(AlgorithmIdentifier*))
    .   .
    .   _asn_ctx   (struct)
    .
    keyEncAlgos
    .   list       (struct)
    .   .   <...>
    .   _asn_ctx   (struct)
    .
    contentEncAlgos
    .   list       (struct)
    .   .   <...>
    .   _asn_ctx   (struct)
    .
    _asn_ctx       (struct)
       <...>
}
```

Die fettgedruckten Einträge stellen hierbei die ASN.1-Elemente dar, die in der jeweiligen ASN.1-Struktur vorhanden sind. Die geklammerten Inhalte, welche sich rechts daneben befinden, zeigen den jeweilig repräsentierten Datentyp an. Dies kann ein nativer C/C++-Datentyp oder auch ein weiteres ASN.1-Element sein. Die dargestellten Sterne (z.B. bei uint8*) stellen in diesem Zusammenhang Zeiger dar und werden durch die generierte Struktur vorgegeben. Der Zugriff auf die einzelnen Einträge erfolgt durch die blau markierten Elemente. Diese beinhalten die entsprechenden Nutzdaten und müssen durch geeignete

Funktionen bedient werden. Dies erfolgt bei einfachen Einträgen (wie der Nonce) über die implementierten ASN.1-Basisfunktionen (Wrapper). Komplexe Einträge, wie die enthaltenen Listen, werden zusätzlich durch private Klassenmethoden des NTS-Nachrichtenobjekts verwaltet. Auf die rot markierten Elemente wird nicht direkt zugegriffen, da sich diese durch die generierten Strukturen selbst verwalten. Dies sind beispielsweise Informationen über die Listengröße (*count*) und der dafür reservierte Speicher (*size*). Die enthaltenen *_asn_ctx*-Elemente beinhalten verschiedene Kontextinformationen zur jeweiligen Struktur und kommen bei deren Kodierung bzw. Dekodierung zur Anwendung. Der Inhalt dieser Strukturen ist für die Implementierung der eigenen Funktionen jedoch nicht relevant.

Die Umsetzung der ASN.1-Basisfunktionen (Wrapper) stellt besondere Herausforderungen dar, die anhand des folgenden Beispiels veranschaulicht werden soll:

Listing 4.6: Zugriff auf Elemente einer generierten Kernstruktur

```
mp_clientAssocData->keyEncAlgos.list.array[0]->parameters->buf = ...
```

Die dargestellte Codezeile zeigt einen Schreibzugriff auf die Kernstruktur, die durch die Membervariable *mp_clientAssocData* repräsentiert wird. An dieser Stelle soll der Parameter eines *KeyEncryption*-Algorithmus hinzugefügt bzw. geändert werden, der sich im ersten Listenelement befindet. Die vier rot markierten Bereiche stellen Pfadelemente dar, die durch Zeiger erreichbar sind. Erfolgt im dargestellten Beispiel ein Lese- oder Schreibzugriff, so müssen jedes Mal die rot markierten Pfadelemente auf Gültigkeit geprüft werden. Ist eines dieser Pfadelemente ungültig (z.B. bei fehlender Initialisierung), so führt dies in der Regel zu einem Programmabsturz. Auch beim Datenpuffer (->*buf*) muss auf die strikte Prüfung der verwendeten Größe geachtet werden, um angreifbare Pufferüberläufe zu verhindern. Diese umfangreichen und häufigen Überprüfungen stellen einen erhöhten Aufwand dar und können die Effizienz des Quellcodes negativ beeinflussen (s.a. Kapitel 6.3).

4.2.2 Das Kryptografiemodul

Einen Großteil der Verarbeitungsroutinen der NTS-Implementierung beinhaltet das Kryptografiemodul. Hierbei handelt es sich im Gegensatz zu den Komponenten im ASN.1-Modul nicht um geschlossene C++-Klassen, sondern um eine große Sammlung von losen Funktionen ohne zugehörige Membervariablen. Diese Funktionen sind durch verschiedene Namensräume zusammengefasst und bilden somit Gruppen für verschiedene Aufgabenbereiche. In der derzeitigen Implementierung sind vier solcher Gruppen vorhanden, die nachfolgend kurz beschrieben werden.

Die Basisfunktionen

Grund- und Hilfsfunktionen, auf die alle Komponenten im Kryptografiemodul zugreifen dürfen, sind in den Basisfunktionen zu finden. Diese beinhalten unter anderem Definitionen, um das C++-RAII-Idiom zu realisieren. Hierbei wird das Ziel der automatisierten Freigabe aller verwendeten Ressourcen verfolgt (Speicherreservierungen), um Speicherlecks, Datenkorruption und ungültige Programmzustände zu unterbinden (s.a. Kapitel 6.3). Des Weiteren beinhalten die Basisfunktionen auch Implementierungen, die keiner anderen Gruppe zugeordnet werden können. Dies sind beispielsweise Initialisierungsfunktionen für den Zufallszahlengenerator und die unterstützten Algorithmen. Auch Hash- und MAC-Funktionen sind in dieser Funktionsgruppe zu finden. Zusätzlich sind auch Funktionen zur Konvertierung und Verarbeitung von Objektkennungen (OIDs) vorhanden, um Algorithmen unabhängig von Bibliotheken zu verarbeiten. Die Konvertierung ist notwendig, da die OpenSSL-Bibliothek primär mit NIDs (*numerical identifier*) arbeitet und somit ein anderes Format verfolgt.

CMS-Strukturen

Diese Gruppe bietet verschiedene Wrapper-Funktionen zur Verarbeitung und Generierung von CMS-Strukturen mit allen darin enthaltenen Eigenschaften. Der Zugriff auf die einzelnen ASN.1-Elemente innerhalb der CMS-Struktur erfolgt über die OpenSSL-Bibliothek, die grundlegende Funktionen für derartige Zugriffe bereitstellt. Die aktuelle Implementierung berücksichtigt hierbei explizit die *SignedData*- und die *EnvelopedData*-Struktur. Die *ContentInfo*-Struktur wird je nach Aufbau und Verschachtelung der CMS-Struktur automatisch hinzugefügt und ist somit implizit vorhanden. Neben den Funktionen zum Setzen und Auslesen der enthaltenen Elemente wurden auch Routinen zur Verschlüsselung, Prüfung und Kodierung der jeweiligen Strukturen implementiert. Ver- und Entschlüsselungsvorgänge erfolgen dabei in den Funktionen, die ausschließlich der *EnvelopedData*-Struktur zuzuordnen sind. Die Wahl der hierbei anzuwendenden Algorithmen kann über diverse Funktionsparameter festgelegt werden. Die Überprüfung der CMS-Struktur umfasst den Integritätscheck durch Verifizierung der Inhalte und Prüfung der digitalen Signatur. Die Prüfung der enthaltenen Zertifikate (sofern vorhanden) ist an dieser Stelle abgekoppelt und erfolgt ausschließlich über die Zertifikatsfunktionen, um eine separierte Überprüfung zu ermöglichen. Auch die Einhaltung der NTS-Spezifikation, die sich auf den Aufbau der CMS-Struktur bezieht, wird nicht in diesen Überprüfungsroutinen berücksichtigt, sondern erfolgen im *Client-Unicast*- bzw. *Server-Unicast*-Modul. Die Serialisierung der CMS-Struktur erfolgt über den DER-Kodierer, der ebenfalls über OpenSSL bereitgestellt wird. Ebenso existiert ein passender DER-Dekodierer, um ASN.1-Datenströme in eine entsprechende CMS-Struktur umzuwandeln.

An dieser Stelle sei noch erwähnt, dass das kostenlose Programm *ASN.1 Editor* eine hervorragende Möglichkeit bietet, gespeicherte ASN.1-Datenströme zu visualisieren (vgl. Abbildung 4.6). Dies geht mit allen ASN.1-Daten, wie kodierte NTS-Nachrichtenobjekte, X509-Zertifikate, Schlüssel oder die hier dargestellten CMS-Strukturen.

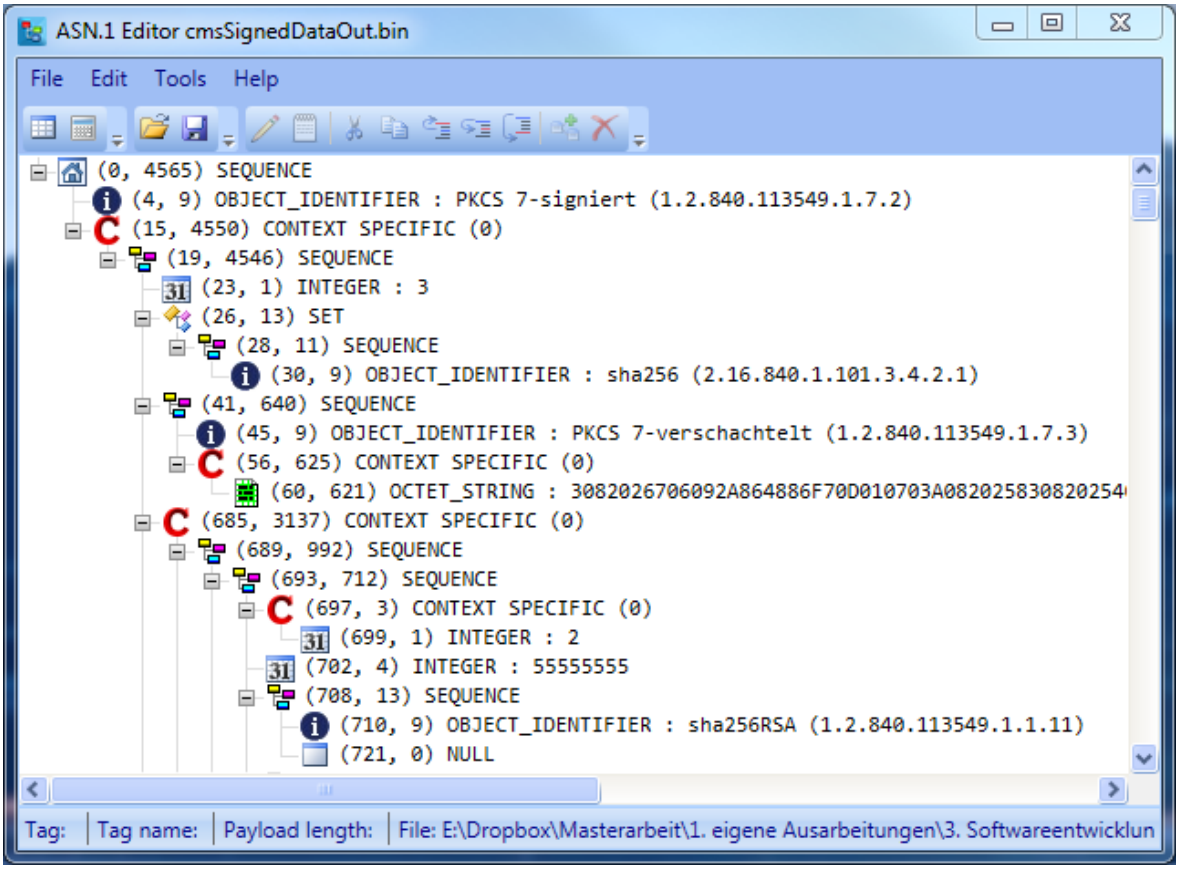


Abbildung 4.6: Analyse eines ASN.1-Datenstroms mit dem Programm "ASN.1 Editor"

Schlüsselverwaltung

Einen weiterern Bereich des Kryptografiemoduls stellen die Funktionen zur Schlüsselverwaltung dar. Diese Funktionsgruppe ermöglicht die Generierung, Überprüfung und Kodierung von Schlüsseln, die für Verschlüsselungen und den digitalen Signaturen verwendet werden können. In der Implementierung werden derzeit RSA- und verschiedene ECC-Algorithmen zur Schlüsselgenerierung akzeptiert. Die einzelnen Parameter wie Bitlänge und Exponent beim RSA-Algorithmus können frei gewählt werden. Die erwähnten Überprüfungsroutinen stellen hier einen Integritätscheck dar. Diese Prüfung ist sinnvoll, wenn die verwendeten Schlüssel als Dateien im PEM-Format vorliegen und eingelesen werden müssen. Ein zweiter Mechanismus erlaubt die Zugehörigkeitsprüfung zweier Schlüssel. Hierbei wird

festgestellt, ob ein privater Schlüssel zum angegebenen öffentlichen Schlüssel oder zum entsprechenden Zertifikat gehört. Die Kodierungsmöglichkeiten erlauben nach dem gleichen Schema wie bei den CMS-Strukturen die Schlüssel in einen ASN.1-Datenstrom umzuwandeln oder daraus einen entsprechenden Schlüssel zu generieren.

X.509-Zertifikatsverwaltung

Die letzte und umfangreichste Komponente im Kryptografiemodul stellt die Zertifikatsverwaltung dar. In dieser findet die Verarbeitung und Generierung einzelner X.509-Zertifikate oder ganzer Zertifikatgruppen statt. Die implementierten Funktionen bieten ein großes Arsenal zur Verarbeitung aller gängigen Zertifikatseigenschaften, die im Folgenden genannt werden:

- Zertifikatsversion
- Seriennummer
- Zertifikatsinhaber (Subjektname)
- Ausstellername
- Gültigkeitsperiode
    - Beginn der Gültigkeitsperiode
    - Ende der Gültigkeitsperiode

- der öffentliche Schlüssel (RSA oder ECC)
- Signatur
    - Signaturalgorithmus
    - digitale Signatur

- Zertifikatserweiterungen
    - *Basic Constraints*
    - *Key Usage*
    - *Extended Key Usage*
    - *Subject Key Identifier*

Der direkte Zugriff auf die Zertifikatstruktur erfolgt über die OpenSSL-Bibliothek, was beispielsweise bei der Generierung und Befüllung der Zertifikate der Fall ist. Die Wrapper-Funktionen erhöhen dabei den Funktionsumfang und sorgen für die korrekte Konvertierung der Daten in das benötigte Format. Die Gültigkeitsperioden der Zertifikate unterstützen somit neben den ASN.1-Zeitformaten (*GeneralizedTime* und *UTCTime* [31, p. 91 ff.]) auch andere Formatierungen und Zeitzonen. Auch die Namen des Zertifikatsinhabers und des Zertifikatsausstellers werden korrekt geparst, wobei derzeit die folgenden Standardattribute unterstützt werden:

- C (*country name*)
- ST (*state or province name*)
- L (*locality*)
- OU (*organizational unit*)
- O (*organization*)
- CN (*common name*)

Durch diese Attribute können die enthaltenen Informationen einzeln extrahiert oder geändert werden, wobei nicht alle Attribute genutzt werden müssen. Ein gültiger Zertifikateintrag könnte dementsprechend wie folgt aussehen:

Listing 4.7: Beispielname für Zertifikatsinhaber und -aussteller

```
CN = Entrust Root Certification Authority
OU = "(c) 2006 Entrust, Inc."
O  = "Entrust, Inc."
C  = US
```

Eine weitere wichtige Funktion der Zertifikatsverwaltung stellt die Überprüfung einzelner Zertifikate und Zertifikatsketten dar. Die Prüfung berücksichtigt nicht nur die digitale Signatur, sondern auch die Gültigkeitsperioden und die zulässigen Längen der Zertifizierungspfade. Letzteres wird überprüft, wenn ein Zertifikat in der Kette die *Key-Usage*-Zertifikatserweiterung beinhaltet, die das „*pathlen*“-Flag enthält. Das Ergebnis der Überprüfung ist von mehreren Faktoren abhängig, wodurch auch gültige Zertifikate als ungültig gewertet werden können. Dies hängt primär von den verwendeten Zertifikatserweiterungen und der Identifizierung des Zertifikattyps ab (s.a. Kapitel 3.6.4). In der aktuellen Implementierung werden derzeit drei Zertifikattypen identifiziert. Dies sind primäre Zertifikate, zwischengeschaltete CA-Zertifikate und Root-CA-Zertifikate. Für diese Typen existieren unterschiedliche Überprüfungsroutinen, wobei ungünstige Konstellationen zu einem Fehler bei der Überprüfung führen. Zur Veranschaulichung wird dies in der folgenden Tabelle 4.1 dargestellt.

Die Tabelle zeigt verschiedene Zertifikattypen, die mit unterschiedlichen Routinen geprüft werden. Der obere Tabellenbereich (*aktivierte Zertifikatseigenschaft*) zeigt hierzu die verwendeten Zertifikatserweiterungen und die eingesetzten Flags. Der mittlere Bereich zeigt die Testergebnisse für ausgestellte Zertifikate an und der untere Tabellenbereich die Testergebnisse für selbstsignierte Zertifikate. Die jeweiligen Testfälle stellen dabei immer eine vollständige Spalte aller drei Bereiche dar. Zudem ist erkennbar, dass die Tabellenbereiche z.B. für ausgestellte Zertifikate weiterer untergliedert sind. Hierbei stellen die oberen drei Einträge ([T1] bis [T3]) den Zertifikattyp dar. Wird durch die implementierte Erkennungsroutine ein bestimmter Zertifikattyp identifiziert, so wird dies als Eins in der jeweiligen Zeile dargestellt. Falls nicht, wird dies durch eine Null repräsentiert. Die folgenden Einträge ([P1] bis [P4]) zeigen die Testergebnisse der vier Prüfungsroutinen an.

Tabelle 4.1: Testkonstellationen von X.509-Zertifikaten

aktivierte Zertifikatseigenschaft																									
Basic Constraints (CA:TRUE)	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
Basic Constraints (CA:FALSE)	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
Key Usage (keyCertSign)	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
Key Usage (cRLSign)	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
Key Usage (sonstige)	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
ausgestelltes Zertifikat mit Zertifizierungspfad (*Subjekt ≠ Aussteller; durch Aussteller signiert*)																								
[T1] ist primäres Zertifikat	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	0	1
[T2] ist zwisch. CA Zertifikat	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	1	0
[T3] ist root-CA Zertifikat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
[P1] mit root CA bundle	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
[P2] ohne root CA bundle	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
[P3] Zert. ist root CA bundle	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
[P4] Zert. Add to root CA bundle	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
[E] geeignete Konstellation	0	-	-	0	+	+	0	-	-	+	+	+	0	-	-	0	+	+	0	-	-	+	-	+
selbstgeneriertes Zertifikat (*Subjekt = Aussteller; selbstsigniertes Zertifikat*)																								
[T1] ist primäres Zertifikat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
[T2] ist zwisch. CA Zertifikat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
[T3] ist root-CA Zertifikat	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	1	0
[P1] mit root CA bundle	S	S	S	U	U	U	S	S	S	U	U	U	S	S	S	U	U	U	S	S	S	S	S	S
[P2] ohne root CA bundle	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	O	O	O
[P3] Zert. ist root CA bundle	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	O	O	O
[P4] Zert. Add to root CA bundle	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	U	U	U	O	O	O	O	O	O
[E] geeignete Konstellation	0	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-	-	0	-	-	+	-	+

Legende

1	Auswahl trifft zu
0	Auswahl trifft nicht zu

O	**OK**
U	**Error: unable to get local issuer certificate**
S	**Error: self signed certificate**

+	Konstellation geeignet
0	Konstellation nicht geeignet (Flags sind widersprüchlich)
-	Konstellation ungültig

Bei der Prüfung P1 werden Root-CA-Zertifikate eingesetzt, wodurch die Routine nun bestrebt ist, eine Zertifikatskette aufzubauen. Bei selbstsignierten Zertifikaten ist dies natürlich nicht möglich, wodurch diese Prüfungen scheitern. Derartige Zertifikate können nur durch die Prüfungsroutine P2 verifiziert werden. In dieser werden keine Root-CA-Zertifikate eingesetzt, wodurch auch keine Zertifikatskette aufgebaut wird. Folglich muss sich ein Zertifikat selbst signieren, um als gültig betrachtet zu werden. Die Prüfungen P3 und P4 stellen Sonderfälle dar, in denen das zu prüfende Zertifikat auch als Root-CA-Zertifikat eingesetzt wird bzw. in der Liste der Root-CA-Zertifikate enthalten ist. Je nach Konstellation werden von OpenSSL drei mögliche Ergebnisse zurückgegeben. Ein „O“ repräsentiert in der Tabelle eine erfolgreiche Verifizierung. Das „S“ stellt einen Fehler dar, bei dem es sich um ein selbstsigniertes Zertifikat handelt und nicht mit der jeweiligen Routine geprüft werden kann. Auch der Buchstabe „U“ repräsentiert einen Fehler, in dem die Zertifikatskette nicht aufgebaut werden kann. In diesem Fall wird ein Zertifikatsaussteller in den Root-CA-Zertifikaten gesucht, der jedoch nicht enthalten ist. Die unterste Zeile in den jeweiligen Tabellenabschnitten zeigt geeignete Konstellationen ([E]). Diese geben Aufschluss darüber welche Zertifikatserweiterungen zu verwenden sind, um die korrekte Identifizierung und Prüfung der Zertifikate zu ermöglichen. Bei ausgestellten Zertifikaten ist dabei immer die Prüfung P1 und bei selbstsignierten Zertifikaten die Prüfung P2 zu verwenden. P3 und P4 haben eher informativen Charakter und kommen in der Implementierung nicht zum Einsatz. Ein Plus (+) zeigt geeignete Zertifikate an und ein Minus (-) ungültige. Die Null (0) stellt gültige Konstellationen dar, die aufgrund von widersprüchlichen Flags in den Zertifikatserweiterungen jedoch ungeeignet sind. Es ist außerdem erkennbar, dass die beiden geeigneten Konstellationen bei den selbstsignierten Zertifikaten keinem Zertifikattyp zugewiesen werden können. Dies stellt zum derzeitigen Punkt noch einen Mangel dar, der in der Implementierung durch weitere Identifizierungsmaßnahmen behoben werden muss. Eine mögliche Lösung stellt die Definition eines zusätzlichen Zertifikattyps dar. Die Zertifikatsverwaltung bietet neben den Überprüfungsroutinen auch wieder die Möglichkeit, die Zertifikate mittels DER-Kodierer in einem ASN.1-Datenstrom zu konvertieren oder auch die Dekodierung vorzunehmen. Des Weiteren ist es auch möglich, einzelne oder mehrere Zertifikate in das PEM-Format zu konvertieren und diese als Dateien zu speichern.

4.2.3 Das Client-Unicast-Modul

Die Generierung von Clientanfragen und die Verarbeitung von Serverantworten erfolgen im Client-Unicast-Modul. Das Design der Implementierung erlaubt dabei die unabhängige Kommunikation mit einer beliebigen Anzahl von Zeitservern. Außerdem besteht die Möglichkeit, mehrere Clientinstanzen simultan zu nutzen, auch wenn dies nicht den Regelfall darstellt. Dieser Abschnitt beschreibt mit Hilfe der Abbildung 4.7 den schematischen Aufbau des Client-Unicast-Moduls und die Eigenschaften der enthaltenen Komponenten. Der interne Programmablauf wird hingegen im Kapitel 4.4 näher erläutert.

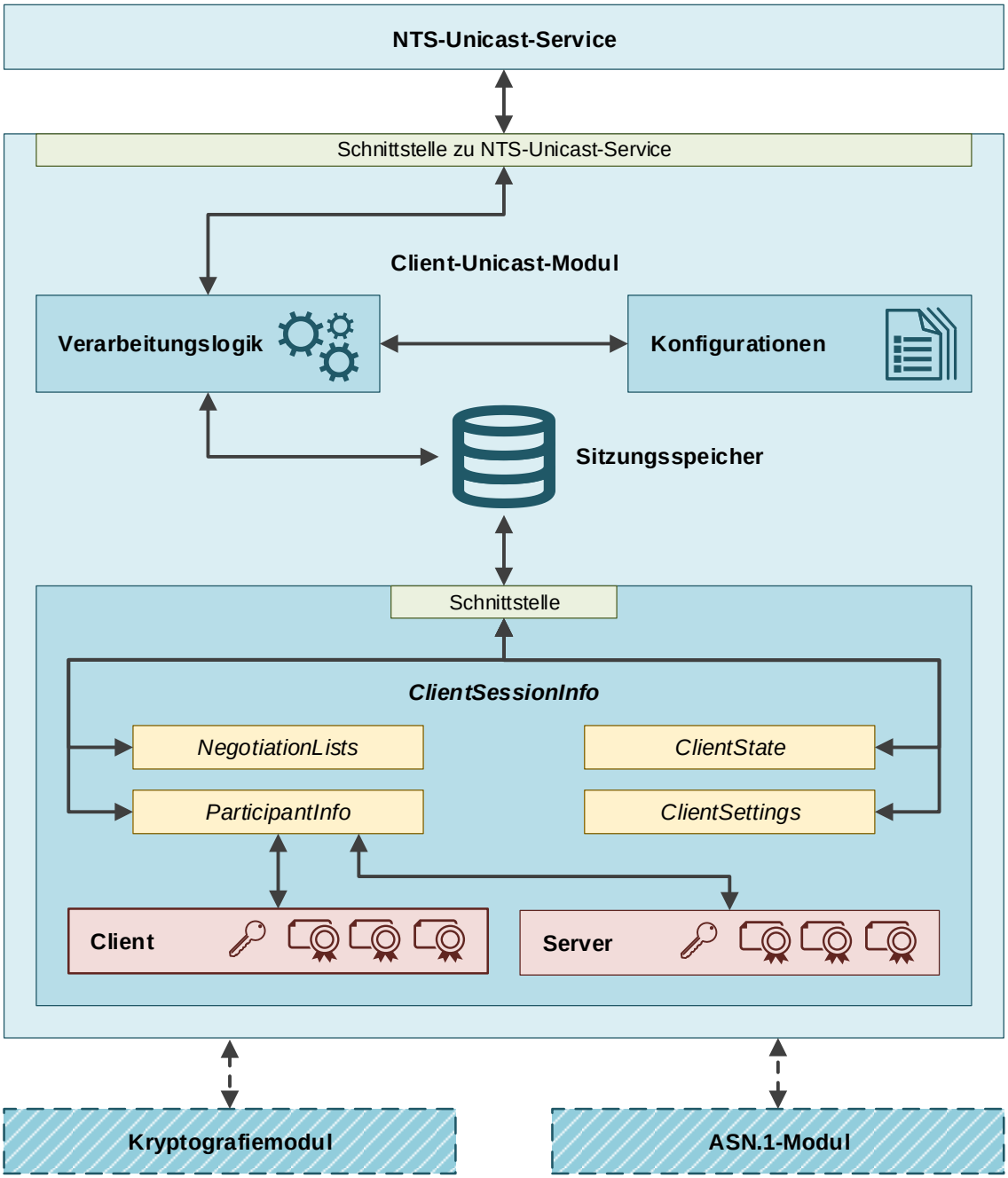


Abbildung 4.7: Aufbau des Client-Unicast-Moduls

Das Client-Unicast-Modul stellt in der Implementierung eine C++-Klasse dar, die über eine definierte öffentliche Schnittstelle mit dem NTS-Unicast-Service-Modul kommuniziert. Außerdem interagiert es mit dem ASN.1- und Kryptografiemodul, wobei dieses intern geschieht und es daher keine expliziten Schnittstellen für diese Module bereitstellt. Das Client-Unicast-Modul beinhaltet derzeit drei Kernkomponenten, die für die korrekte Funktionsweise benötigt werden. Dies sind die Verarbeitungslogik, die Konfiguration und der Sitzungsspeicher.

Die Verarbeitungslogik stellt im Prinzip die Menge an Funktionen dar, die für die Verarbeitung und Generierung der Nachrichten benötigt werden. Diese Funktionen greifen auf die jeweiligen ASN.1- und Kryptografiemodule zu und stellen bei der Verarbeitung der Nachrichten zudem die Einhaltung der NTS-Spezifikation sicher. Die Verarbeitungslogik wird durch die verwendete Konfiguration beeinflusst und steuert zudem den Sitzungsspeicher und diverse Ausgaben (z.B. Diagnoseinformationen). Wird eine Instanz des Client-Unicast-Moduls erzeugt, so muss diese zunächst konfiguriert werden, um eine Kommunikation mit einem NTS-Server zu ermöglichen. Die Konfiguration erfolgt über das NTS-Unicast-Service-Modul, indem dieses eine Initialisierungsmethode aufruft und einen Dateipfad als Parameter übergibt. Dieser Pfad muss auf eine gültige Konfigurationsdatei verweisen (z.B. NtsClient.ini), die bei Bedarf eingelesen werden kann. Zurzeit ist dies dann der Fall, wenn eine neue Verbindung zu einem unbekannten NTS-Server aufgebaut werden soll. Der Inhalt dieser Konfigurationsdatei wird im Kapitel 4.5 thematisiert und beinhaltet grundlegende Verbindungs- und Sicherheitseinstellungen. Der Sitzungsspeicher ist eine einfache Datenbank, in der Verbindungen zu verschiedenen NTS-Servern gehalten werden. Das bedeutet, dass sowohl Client- als auch Serveraushandlungen als eine Einheit betrachtet werden. Dies beinhaltet auch individuelle Clienteinstellungen oder Zustände. Sowohl die gekapselten Informationen, die der Client für den entsprechenden Server speichert, als auch der Zustand, den der Client gegenüber dem Server besitzt, werden zusammen als Sitzungen bezeichnet. Zu jedem Zeitserver kann genau eine Sitzung existieren, wobei mehrere Sitzungen zu verschiedenen Zeitservern problemlos möglich sind. Diese sind strikt voneinander getrennt und können sich nicht gegenseitig beeinflussen. Der Zugriff auf eine bestimmte Sitzung erfolgt durch die IP-Adresse des Zeitservers. Wird der Zugriff auf eine Sitzung mit einer noch unbekannten IP-Adresse getätigt, so wird automatisch eine neue Sitzung angelegt und vorkonfiguriert. Die Speicherung der Sitzungen erfolgt ausschließlich im Arbeitsspeicher. Wird das Programm beendet, so gehen auch alle Sitzungsdaten verloren. Prinzipiell wäre es auch möglich, die Sitzungen in Dateiform zu speichern, um nach einem Programmneustart die jeweiligen Verbindungen fortzusetzen. Eine Implementierung dazu gibt es aber derzeit nicht.

Die einzelnen Sitzungen werden in der Implementierung als eine C++-Klasse repräsentiert, die unter den Namen *ClientSessionInfo* verfügbar ist. Diese beinhaltet wiederum vier Komponenten, in denen die benötigten Dateien gespeichert werden. Die erste Komponente stellt hierbei die eingebettete *NegotiationLists*-Klasse dar. Diese beinhaltet die Listen mit den Algorithmen und den Versionsnummern, die vom Client unterstützt werden. Die Daten dazu werden nach der Erstellung einer neuen Sitzung durch das Einlesen der Konfigurationsdatei hinzugefügt. Neben den Listen beinhaltet diese Klasse auch die Algorithmen und die verwendete NTS-Versionsnummer, die mit dem jeweiligen Server ausgehandelt wurden. Die zweite Komponente stellt die eingebettete *Participant*-Klasse dar, die zweimal existiert und spezifische Daten vom Client und Server getrennt speichert. Die besagte Klasse beinhaltet die verwendete Netzwerkadresse der Kommunikationsteilnehmer, die Zertifikate bzw. Zertifikatsketten und die verwendeten kryptografischen Schlüssel. Dabei müssen allerding nicht

zwangsläufig alle Informationen verwendet werden. Die dritte Komponente stellt der *Client-State* dar, in dem der aktuelle Verbindungszustand des Clients zum jeweiligen Server gespeichert wird. Folgende Zustände können hierbei eingenommen werden:

- not_init → Client ist nicht initialisiert
- init → Client ist initialisiert (bereit für Access-Nachrichten)
- server_blocked → die Verbindung zum Server ist blockiert (experimentell)
- access_phase → Verhandlungsphase: Zugriffsphase
- assoc_phase → Verhandlungsphase: Assoziationsphase
- cookie_phase → Verhandlungsphase: Cookie-Austauschphase
- timeSync_phase → Zeitsynchronisationsphase

Die Wahl des Zustands hängt dabei von den Verarbeitungsergebnissen der Nachrichten und von den vorherrschenden Verbindungseigenschaften ab. Letzteres wird durch die Anzahl der Verbindungsversuche und der Verbindungs-Time-Outs bestimmt. Die Einstellungen dazu werden aus den *ClientSettings* bezogen, welche auch die letzte Komponente darstellt. Diese beinhaltet allgemeine Einstellungen zur jeweiligen Sitzung und den Inhalt der eingelesenen Konfigurationsdatei.

4.2.4 Das Server-Unicast-Modul

Das Server-Unicast-Modul stellt das Gegenstück zum Client-Unicast-Modul dar, welches Clientanfragen verarbeitet und Serverantworten generiert. Der schematische Aufbau ähnelt dem Client-Unicast-Modul, in dem allerdings keine Datenhaltung zum Speichen von Sitzungsdaten integriert ist (vgl. Abbildung 4.8). Des Weiteren beinhaltet dieses Modul keine aktive Zustandsregelung, wodurch die enthaltenen Verarbeitungsroutinen als zustandslos betrachtet werden können. Da die Wahl der korrekten Verarbeitungsroutine nur von der empfangenen Nachricht abhängt, kann das Modul eine beliebige Anzahl von Clients bedienen. In der Implementierung stellt dieses Modul eine C++-Klasse dar, die über eine öffentliche Schnittstelle mit dem NTS-Unicast-Service-Modul kommuniziert. Zusätzlich interagiert dieses auch mit dem ASN.1- und Kryptografiemodul, wobei dieses intern geschieht und es daher keine expliziten Schnittstellen für diese Module bereitstellt. Die Ausführung des Server-Unicast-Moduls ist zudem in mehreren Instanzen möglich, wodurch eine unbegrenzte Anzahl unabhängiger NTS-Server mit jeweils unterschiedlichen Konfigurationen bereitgestellt werden können. In der Praxis stellt dies allerdings keinen typischen Anwendungsfall dar.

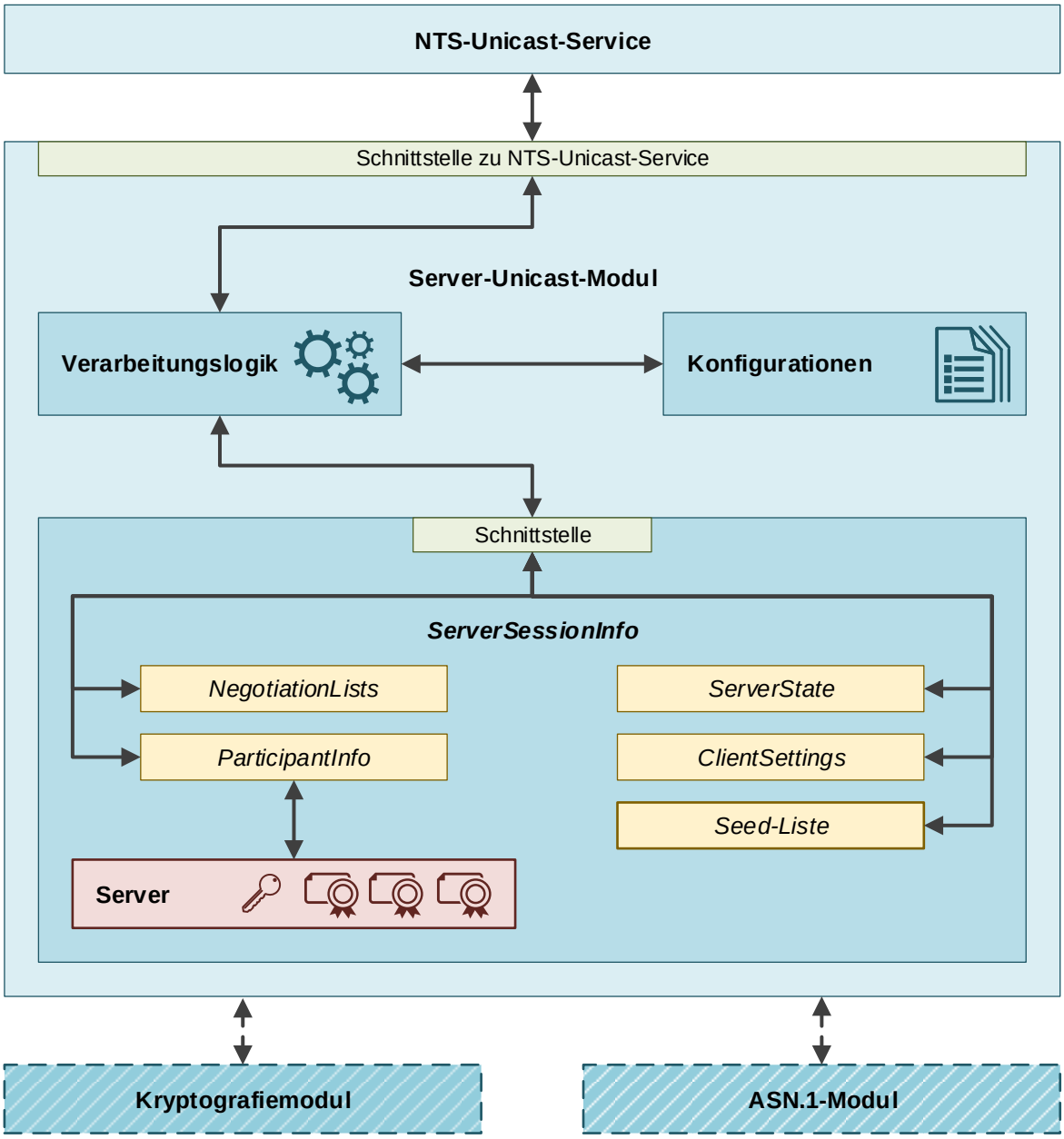


Abbildung 4.8: Aufbau des Server-Unicast-Moduls

Das Server-Unicast-Modul beinhaltet derzeit drei Kernkomponenten, die für die korrekte Funktionsweise benötigt werden. Dazu zählt die Verarbeitungslogik, die Konfiguration und das *ServerSessionInfo*-Objekt. Die Verwaltungslogik stellt erneut das Kernstück der Klasse dar, in dem sowohl die Verarbeitung der NTS-Nachrichten, als auch die Kommunikation zwischen den einzelnen Komponenten geschieht. Die Konfiguration beinhaltet an dieser Stelle einige Daten aus der Konfigurationsdatei, die bei der Modulinitialisierung als Parameter übergeben und anschließend eingelesen wird. Hierbei handelt es sind ausschließlich um Einstellungen, die zu Diagnosezwecken aktiviert werden können, um die genaue Beobachtung des Modulverhaltens zu ermöglichen. Neben der Konfiguration beinhaltet die *ServerSessionInfo*-Komponente weitere Informationen, die für die Kommunikation mit den

Clients benötigt werden. Dabei handelt es sich um eine eingebettete C++-Klasse, welche die grundlegende Konfiguration des Servers beinhaltet. Der Aufbau ähnelt der *ClientSession-Info*-Klasse aus dem Client-Unicast-Modul, wobei sich nur einige Segmente unterschieden. Die eingebettete *NegotiationLists*-Klasse beinhaltet Listen mit den Algorithmen und den Versionsnummern, die vom Server unterstützt werden. Einträge für die jeweils ausgehandelten Algorithmen und der NTS-Version kommen dabei jedoch nicht zur Anwendung. Die entsprechenden Inhalte der Listen werden nach dem Einlesen der Konfigurationsdatei während der Modulinitialisierung in den *ServerSettings* gespeichert und von dort aus herangezogen. Ein weiteres Segment stellt die eingebettete *Participant*-Klasse dar, welche die verwendete Netzwerkadresse des Servers, die Zertifikate bzw. Zertifikatsketten und die verwendeten kryptografischen Schlüssel enthält. Das vierte Segment ist der *ServerState*, in dem der aktuelle Betriebszustand des Servers gespeichert wird. Folgende Zustände können hierbei eingenommen werden:

- not_init → Server ist nicht initialisiert
- init → Server ist initialisiert (Nachrichtenverarbeitung möglich)

Die Wahl des Zustands hängt hierbei von der Modulinitialisierung ab. Ist diese erfolgreich abgeschlossen und beinhalten alle Listen mindestens einen Eintrag, so gilt das Modul als initialisiert. In diesem Zustand kann nun auf entsprechende Clientanfragen reagiert werden. Der für einige Serverantworten benötigten *Server Seed* wird hierbei aus der *Seed-Liste* bezogen. Diese stellt das letzte Segment dar und beinhaltet für jeden MAC- bzw. Hash-Algorithmus einen individuellen Seed. Der Zugriff darauf erfolgt über selbigen Algorithmus, der in der Liste als Kennung fungiert, um den korrekten Seed zu zurückzuliefern. Die *Seed-Liste* beinhaltet zudem auch eine Logik, mit der abgelaufene Seeds beim jeweiligen Zugriff automatisch aktualisiert werden. Die entsprechende Gültigkeitsperiode der Seeds wird aus den *ServerSettings* bezogen, die wiederum aus der Konfigurationsdatei stammen.

4.2.5 Das NTS-Unicast-Service-Modul

Der Zusammenschluss des Client- und Servermoduls repräsentiert die komplette Funktionalität, die den Unicast-Betrieb im NTS-Protokoll ausmacht. Dies wird durch das *NTS-Unicast-Service*-Modul realisiert (vgl. Abbildung 4.9), welches abermals als C++-Klasse implementiert wurde und die Ausführung von mehreren Instanzen unterstützt. Ein Objekt dieser Klasse kann direkt in eine bestehende NTPv4-Implementierung eingebettet werden, wobei die Einbindung von NTS auch als externe Bibliothek denkbar wäre. Die Kommunikation zwischen NTP und NTS würde zudem in beiden Varianten über dieselbe definierte Schnittstelle (s.a. Kapitel 4.3) erfolgen. Des Weiteren ist es denkbar, dass dieses Modul andere Kommunikationskanäle (z.B. TLS) verwendet, um den Austausch des Cookies vorzunehmen. Die Sicherheitsmechanismen bleiben in solchen Fällen nach wie vor erhalten.

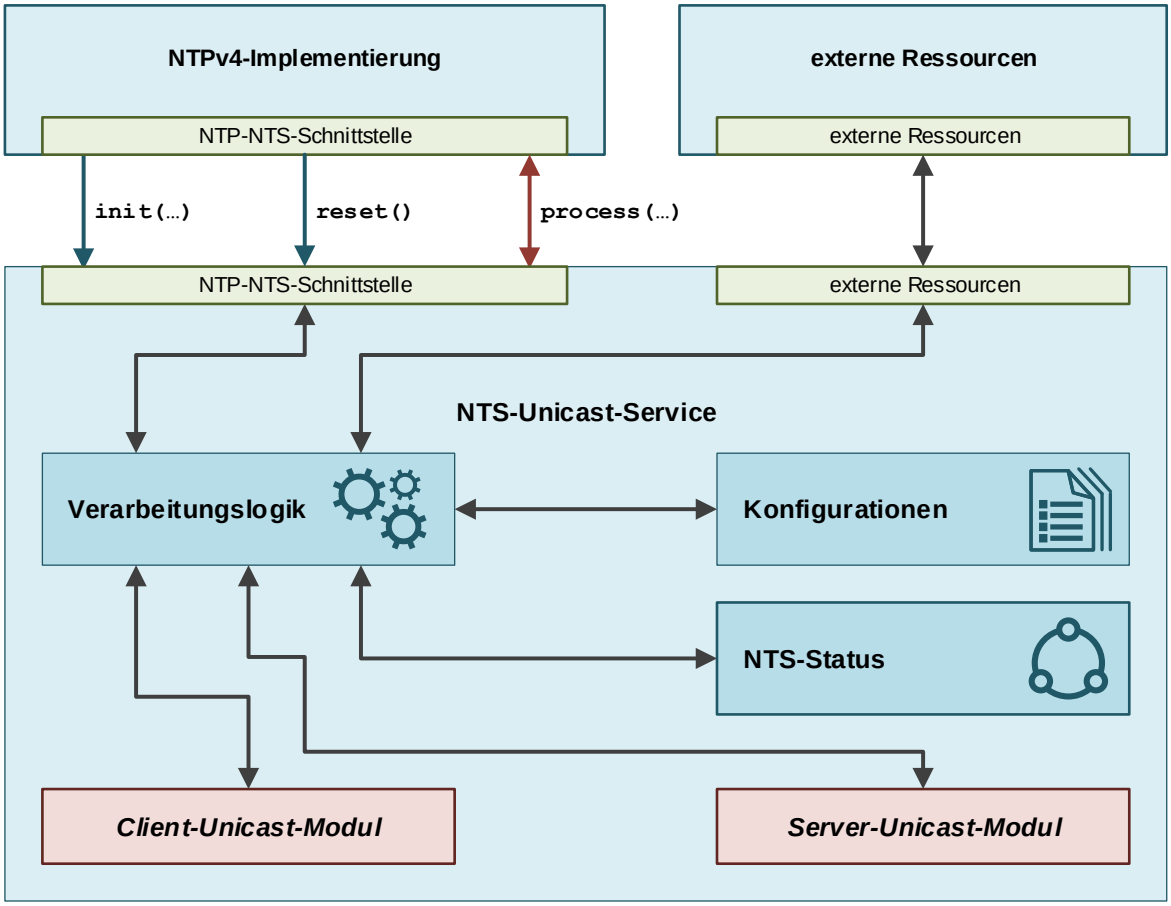


Abbildung 4.9: Aufbau des NTS-Unicast-Service-Moduls

Die Verarbeitungslogik stellt das Herz des Moduls dar, die mit allen enthaltenen Komponenten verbunden ist. Es hat die Aufgabe den Informationenfluss zu filtern und entsprechende NTS-Nachrichten an die jeweiligen Komponenten weiterzuleiten. Dabei kann es sowohl eine bestehende Kommunikation verarbeiten, als auch eine neue initiieren. Dieses hängt von der Nutzung der öffentlichen Funktionen ab, die NTS der NTP-Implementierung zur Verfügung stellt. Derzeit sind dies drei Funktionen, die in der Abbildung 4.9 vereinfacht als *init(…)*, *reset()*, und *process(…)* dargestellt werden. Die *init*-Funktion initialisiert das *NTS-Unicast-Service*-Modul, welches zuvor durch die NTP-Implementierung zu instanziieren ist. Als Parameter wird ein Dateipfad zu einer Konfigurationsdatei übergeben, aus dem die globalen NTS-Einstellungen zu beziehen sind. Diese wird während der Initialisierung geladen und anschließend als Konfiguration im Modul bereitgestellt. Die enthaltenen Einstellungen regeln dabei die Konsolen- und Logausgaben, sowie die Kennungen der NTS-Nachrichtenobjekte (s.a. Kapitel 4.5). Ferner werden während der *init*-Phase auch alle enthaltenen NTS-Module rekursiv initialisiert. Damit sind die beiden eingebundenen Client- und Servermodule, als auch das ASN.1- und das Kryptografiemodul gemeint. Letzteres lädt

unter anderem die ASN.1-Kennungen und initiiert den Zufallszahlengenerator. Ist der Initialisierungsvorgang des *NTS-Unicast-Service*-Moduls abgeschlossen, so wird der entsprechende Zustand gesetzt. Hierbei können folgende Zustände eingenommen werden:

- not_init → nicht initialisiert oder Initialisierung ist gescheitert
- init → das Unicast-Modul ist initialisiert und einsatzbereit

Wünscht die NTP-Implementierung ein Zurücksetzen des Moduls, so kann dieses die *reset*-Funktion aufrufen. Dadurch werden alle Einstellungen und Verbindungen rekursiv gelöscht und das Modul wird als *nicht initialisiert* markiert. Erst nach dem erneuten Aufruf der *init*-Funktion kann das Modul wieder in den betriebsbereiten Zustand versetzt werden, das fortan den Aufruf der *process*-Methode gestattet. Diese Funktion verarbeitet die übergebenen NTP-Pakete und leitet diese entsprechend weiter. Dazu prüft und analysiert es die im NTP-Paket enthaltenen Erweiterungsfelder und extrahiert die jeweilige Kennung der NTS-Nachricht. Identifizierte Clientnachrichten werden somit an das Servermodul weitergeleitet und Servernachrichten an das Clientmodul. Diese Module sind als Objekte in dem *NTS-Unicast-Service*-Modul eingebettet und verarbeiten die entsprechenden Nachrichten. Die zurückgelieferten Ergebnisse werden anschließend über die *process*-Funktion an die NTP-Implementierung weitergereicht. Client- und Servermodul sind dabei unabhängig voneinander und können durch die Konfiguration separat aktiviert oder deaktiviert werden. Wird eine Nachricht für ein deaktiviertes Modul empfangen oder während der Analyse als ungültig betrachtet, so wird die jeweilige Nachricht verworfen.

4.3 Schnittstellen zu NTP

Das Design der NTS-Implementierung verfolgt einen modularen Ansatz für den Aufbau der einzelnen Komponenten. Dies trifft somit auch auf das *NTS-Unicast-Service*-Modul zu, das als in sich geschlossene Einheit zu betrachten ist. Der große Vorteil, der sich daraus ergibt, ist die einfache Wiederverwendung der Implementierung in anderen Trägerprotokollen. Dem ist hinzuzufügen, dass auch die Wartung der Module durch diese Maßnahme vereinfacht wird, da modulinterne Änderungen die äußeren Schnittstellen in der Regel nicht beeinflussen. Die Modularisierung bringt jedoch auch Nachteile mit sich. So leidet die Effizienz des Quellcodes, da direkte Zugriffe auf interne Ressourcen nicht erlaubt sind und entsprechende Anfragen durch Funktionsrufe weitergereicht werden. Basierend auf der Beschreibung der Schnittstellenfunktionen aus dem vorherigen Abschnitt 4.2 werden an dieser Stelle die ausgetauschten Daten zwischen der NTP- und der NTS-Implementierung näher betrachtet. Das Augenmerk liegt hierbei auf der *process*-Funktion, in welcher der besagte Austausch stattfindet (vgl. Abbildung 4.10).

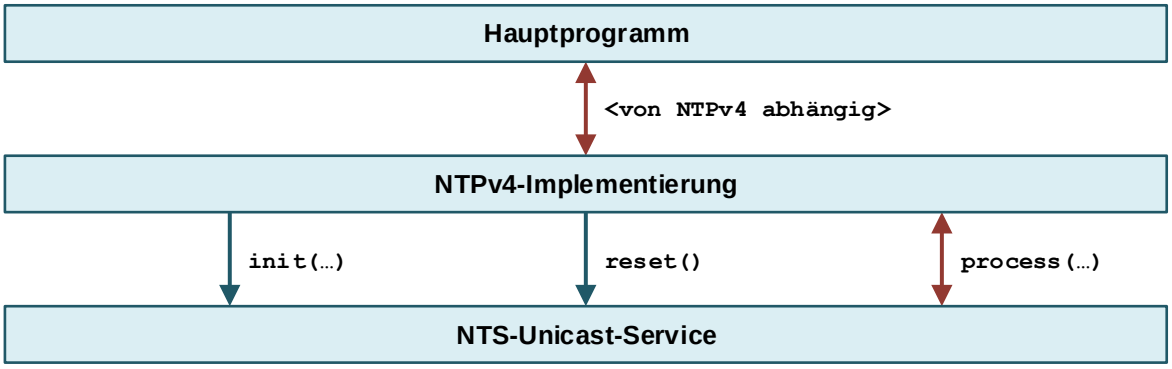


Abbildung 4.10: Schnittstellenfunktionen zwischen NTP und NTS

Zum aktuellen Stand der Implementierung steht hinter der *process*-Funktion folgende C++-Klassenmethode:

Listing 4.8: Aufruf der processNtpMessage()-Methode im NTS-Unicast-Service-Modul

```
void processNtpMessage(NtpNtsInterface &receivedPackage,
                       NtpNtsInterface &newResponsePackage);
```

Diese wird vom *NTS-Unicast-Service*-Modul als öffentliche Funktion der NTP-Implementierung bereitgestellt und beinhaltet die beiden Parameter *receivedPackage* und *newResponsePackage*. Der erste Parameter enthält hierbei das empfangene NTP-Paket, das vom jeweiligen NTP-Kommunikationspartner über das Netzwerk übertragen wurde. Jenes wird durch das *NTS-Unicast-Service*-Modul verarbeitet, wodurch eine entsprechende Antwortnachricht generiert wird. Diese bettet NTS in den zweiten Parameter ein, welches auch ein NTP-Paket für die Antwortnachricht enthält. Die NTS-Implementierung erlaubt hierbei auch die Übergabe „leerer“ Objekte. Wird der erste Parameter als leer betrachtet, weil das enthaltene NTP-Paket keine NTS-spezifischen Erweiterungsfelder enthält, so generiert die NTS-Implementierung automatisch eine entsprechende Clientanfrage, die sich auf den derzeitigen Zustand zum jeweiligen Kommunikationspartner bezieht. Bei der ersten Kontaktaufnahme vom Client zum Server wird somit immer eine *client_access*-Nachricht erzeugt und in den zweiten Parameter (*newResponsePackage)* eingebettet. Handelt es sich hierbei um einen NTS-Server, in dem das Clientmodul deaktiviert wurde, so wird natürlich keine Nachricht generiert und der *receivedPackage*-Parameter wird verworfen. Auch der zweite Parameter kann leer sein bzw. durch die NTS-Implementierung als unbrauchbar markiert werden. Dies ist immer dann der Fall, wenn eine empfangene NTS-Nachricht während der Verarbeitung als ungültig betrachtet wird und keine entsprechende Antwort generiert werden kann. Ist der zweite Parameter dementsprechend markiert, so ist das enthaltene NTP-Paket zu verwerfen.

Beide Funktionsparameter der *processNtpMessage*-Methode sind vom Datentyp *NtpNtsInterface*. Dieser Datentyp stellt eine C++-Klasse dar, die sowohl von der NTS-Implementierung, als auch von NTP verwendet wird. Die instanziierten Objekte dieser Klasse beinhalten jeweils ein NTP-Paket, verschiedene Attribute und Methoden (vgl. Tabelle 4.2 und Tabelle 4.3). Das NTP-Paket wird hierbei durch die NTP-Implementierung eingebettet, welches zudem die vollen Zugriffsrechte auf die instanziierten *NtpNtsInterface*-Objekte, als auch auf das jeweils enthaltene NTP-Paket besitzt. Der Zugriff durch NTS auf die enthaltenen Informationen geschieht durch die öffentlichen Schnittstellen der *NtpNtsInterface*-Klasse (vgl. Tabelle 4.2), die im Folgenden beschrieben werden.

Tabelle 4.2: Öffentliche Klassenmethoden der NtpNtsInterface-Klasse

Rückgabewert	**öffentliche Klassenmethode**
double	getTransmitTimestamp()
double	getOriginTimestamp()
std::string	getIpAddress()
void	updatePackageTimestamps()
int	getExtensionFieldCount()
unsigned short	getExtensionFieldType(index)
std::vector<unsigned char>	getExtensionFieldValue(index)
void	addExtensionField(fieldType, value)
void	delExtensionField(index)
std::vector<unsigned char>	getSerializedPackage()
void	setUseMessageForTimeSync(state)
void	setSendResponseMessage(state)

Tabelle 4.3: Private Attribute der NtpNtsInterface-Klasse

Datentyp	**private[58] Attribute im Objekt**
bool	m_useMessageForTimeSync
bool	m_sendResponseMessage
NTPPacket	m_ntpPacket

Klassenmethode: *getTransmitTimestamp()*

Diese Methode wird bei NTS clientseitig während der Generierung der *time_request*-Nachricht verwendet, um den *transmitTimestamp* der zu übertragenen NTP-Nachricht zu erhalten. Dieser wird vom Client gespeichert und später bei der Prüfung der zugehörigen Antwortnachricht benötigt. Der verwendete Datentyp ist *double*, könnte allerdings auch beliebig geändert werden, sofern die Daten über einfache Vergleichsoperatoren verglichen werden können.

[58] Der Zugriff auf private Attribute eines Objekts erfolgt ausschließlich über die öffentlichen Klassenmethoden

Klassenmethode: *getOriginTimestamp()*

Diese Funktion wird von NTS clientseitig nach Erhalt der *time_response*-Nachricht benötigt, um den *originTimestamp* des empfangenen NTP-Pakets zu bekommen. Dieser muss dann mit dem zuvor gespeicherten *transmitTimestamp* identisch sein. Um die Prüfung zu ermöglichen, müssen *getTransmitTimestamp()* und *getOriginTimestamp()* denselben Datentyp als Rückgabewert nutzen.

Klassenmethode: *getIpAddress()*

Die IP-Adresse wird sowohl vom Client, als auch vom Server benötigt und kann entweder als IPv4 oder IPv6 vorliegen. Die einzige Restriktion ist hierbei die konsequente Verwendung eines Typs während der kompletten Kommunikationsphase. Der Client nutzt diese IP-Adresse um den jeweiligen Zeitserver einer entsprechenden Sitzung zuordnen zu können. Der Server benötigt diese Adresse, um während der *server_access*-Nachricht den Zugriffsschlüssel zu berechnen oder um diesen nach Erhalt der *client_assoc*-Nachricht zu prüfen. Derzeit wird die IP-Adresse als Zeichenkette im Klartextformat übertragen (z.B. „64.124.74.165“), wobei der Inhalt theoretisch auch anders aussehen könnte. Denkbar wäre auch die Verwendung von DNS-Bezeichnungen wie „*ptbtime1.ptb.de*“, die einem Zeitserver eindeutig zugeordnet werden können.

Klassenmethode: *updatePakageTimestamps()*

Diese Funktion wird vom NTS-Client und -Server unmittelbar vor der Berechnung des MAC benötigt, der anschließend als zweites Erweiterungsfeld in das NTP-Paket eingefügt wird. Sie soll die NTP-Implementierung dazu veranlassen, alle im NTP-Paket enthaltenen Zeitstempel zu aktualisieren. Dadurch wird ein Teil der Berechnungszeit eliminiert, die NTS für die Erstellung der NTS-Nachricht benötigt hat.

Klassenmethode: *getExtensionFieldCount()*

Die Anzahl der im NTP-Paket enthaltenen Erweiterungsfelder wird durch die *getExtensionFieldCount*-Funktion bestimmt. Diese wird unter anderem im *NTS-Unicast-Service*-Modul verwendet, um die Erweiterungsfelder zu analysieren.

Klassenmethode: *getExtensionFieldType(index)*

Diese Methode wird von NTS benötigt, um den Inhalt eines NTP-Erweiterungsfeldes in Erfahrung zu bringen. Das *NTS-Unicast-Service*-Modul kann somit feststellen, ob es sich um eine NTS-Nachricht bzw. einen NTS-MAC handelt. Zusätzlich lässt sich über diesen Feldtyp auch die verwendete NTS-Version bestimmen, wodurch die Inhalte korrekt geparst werden. Der übergebene Funktionsparameter stellt einen Index dar, mit dem auf ein bestimmtes Erweiterungsfeld zugegriffen wird. Dies ist notwendig, da ein NTP-Paket mehrere Erweiterungsfelder beinhalten kann.

Klassenmethode: *getExtensionFieldValue(index)*

Diese Funktion liefert den Inhalt des angegebenen Erweiterungsfeldes als Rohdaten. Der Zugriff auf dieses erfolgt durch die Angabe des entsprechenden Index als Funktionsparameter.

Klassenmethode: *addExtensionField(fieldType, value)*

Die *addExtensionField*-Methode ermöglicht NTS das Hinzufügen neuer Erweiterungsfelder in das NTP-Paket. Diese werden dabei immer an das Ende bestehender Erweiterungsfelder eingefügt, wodurch eine Sortierung nicht direkt möglich ist. Beim Aufruf dieser Funktion werden zwei Parameter erwartet. Bei dem ersten handelt es sich dabei um die Kennung, die in den Feldtyp einzutragen ist. Der zweite Parameter ist der dazugehörige Inhalt, der bei NTS das BER- oder DER-kodierte *NTSExtensionFieldContent*-Objekt beinhaltet.

Klassenmethode: *delExtensionField(index)*

Mit der Funktion *delExtensionField(index)* wird NTS die Möglichkeit geboten, ein beliebiges NTP-Erweiterungsfeld aus einem NTP-Paket zu löschen, in dem es über den Funktionsparameter auf ein bestimmtes Erweiterungsfeld referenziert. NTS benötigt diese Funktion, um im NTP-Paket enthaltene MAC-Nachrichten nach deren Auslesung zu entfernen. Anschließend kann NTS erneut einen MAC über das NTP-Paket generieren und anschließend mit den zuvor ausgelesenen Daten vergleichen.

Klassenmethode: *getSerializedPackage()*

Diese Funktion serialisiert das NTP-Paket mit allen darin befindlichen Inhalten und liefert das Ergebnis als binären Datenstrom zurück. NTS nutzt diese Funktion für die Generierung der MAC, da das NTP-Paket als Objektstruktur nicht geeignet ist.

Klassenmethode: *setUseMessageForTimeSync(state)*

Das *Client-Unicast*-Modul setzt diese Methode am Ende jeder Nachrichtenverarbeitung ein, um der NTP-Implementierung die explizite Freigabe der im NTP-Paket enthaltenen Zeitinformationen zu bestätigen oder zu verweigern. Hierbei setzt diese Funktion den Booleschen Wert der *m_useMessageForTimeSync*-Membervariable. Wird dieser Wert von NTS auf *true* gesetzt, so wird die Verwendung der Zeitdaten gestattet. Dies ist allerdings nur clientseitig bei empfangenen Zeitnachrichten der Fall, bei denen der MAC erfolgreich verifiziert wurde. Aus diesem Grund wird der Aufruf dieser Funktion nur im *NtpNtsInterface*-Objekt ausgeführt, welches als erster Parameter der *processNtpMessage*-Funktion übergeben wird.

Klassenmethode: *setSendResponseMessage(state)*

Diese Funktion setzt das *Client-Unicast*- und das *Server-Unicast*-Modul am Ende jeder Nachrichtengenerierung ein, um der NTP-Implementierung die explizite Übertragung des enthaltenen NTP-Pakets zu gestatten oder zu verweigern. Die *setSendResponseMessage*-Methode setzt hierbei den Booleschen Wert der *m_sendResponseMessage*-Membervariable. Wird dieser Wert von NTS auf *true* gesetzt, so darf dieses NTP-Paket durch die NTP-Implementierung gesendet werden. Dieser Fall tritt immer dann ein, wenn die Verarbeitung einer empfangenen Nachricht gültig war. Aber auch bei Fehlschlägen wird dieser Wert auf *true* gesetzt, wenn der NTS-Client eine erneute Anfrage sendet oder der NTS-Server einen entsprechenden Fehlercode übertragen möchte. Der Aufruf dieser Funktion erfolgt nur im *NtpNtsInterface*-Objekt, das als zweiter Parameter der *processNtpMessage*-Funktion übergeben wird.

4.4 Programmablauf der Implementierung

In diesem Abschnitt erfolgt die Programmablaufbeschreibung der aktuellen NTS-Implementierung, die sich in die drei Bereiche *NTS-Unicast-Service*-Modul, *Client-Unicast*-Modul und *Server-Unicast*-Modul aufteilen lässt. Je nachdem, welche Nachricht dabei von der NTP-Schnittstelle übergeben wird, verfolgt die NTS-Implementierung einen bestimmten Verarbeitungspfad (vgl. Abbildung 4.11). Diese Pfade werden im Folgenden näher beschrieben um, eine allgemeine Übersicht des Nachrichtenflusses zu verschaffen. Eine zusätzliche Übersicht als Sequenz- und Aktivitätsdiagramm kann dazu aus den Anlagen[59] entnommen werden. Eine detaillierte Beschreibung zur Verarbeitung bestimmter Nachrichtentypen erfolgt an dieser Stelle jedoch nicht. Da sich die NTS-Implementierung allerdings stark an die Spezifikation richtet, können diese Informationen aus Kapitel 3.3 und 0 bezogen werden.

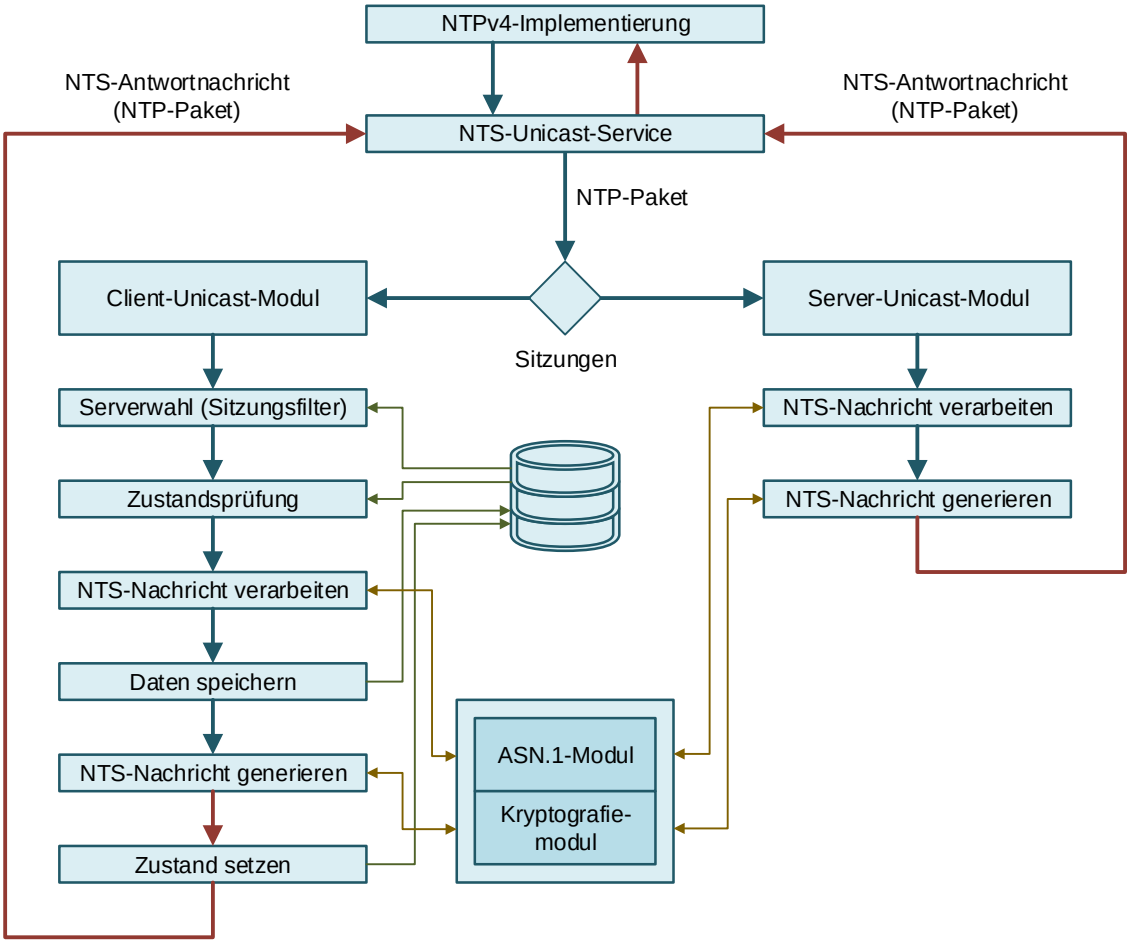


Abbildung 4.11: Allgemeiner Nachrichtenfluss in NTS

[59] Anlage III: Ablaufbeschreibung durch Sequenzdiagramme;
Anlage IV: Ablaufbeschreibung durch Aktivitätsdiagramme

4.4.1 Die Initialisierungsphase

Bevor die Verarbeitung und Generierung der NTS-Nachrichten erfolgen kann, muss zunächst das instanziierte *NTS-Unicast-Service*-Modul korrekt initialisiert werden. Dies erfolgt über NTP mit dem Aufruf der Initialisierungsmethode, die das Modul bereitstellt. Die interne Verarbeitungsroutine führt dazu die Initialisierung des Kryptografiemoduls und der damit verbundenen OpenSSL-Bibliothek durch. Diese stellt die unterstützten Algorithmen bereit und initiiert den Zufallszahlengenerator mit einem geeigneten Seed, der wiederum durch die bereitgestellte Entropie (s.a. Kapitel 3.6.2) der Hardware erzeugt wird. Anschließend erfolgt die Zurücksetzung der *NTS-Unicast-Service*-Instanz, wodurch alle Inhalte auf einen definierten Zustand gebracht werden. Ist dieser erreicht, lädt das *NTS-Unicast-Service*-Modul die Konfigurationsdatei, die beim Aufruf der Initialisierungsmethode der *NTS-Unicast-Service*-Instanz als Parameter übergeben wurde. Nach erfolgreichem Einlesen der Datei findet die Konfiguration und die Prüfung der Objektkennungen (OIDs und NIDs) im ASN.1- und im Kryptografiemodul statt. Dadurch können nun empfangene NTS-Nachrichten eindeutig identifiziert werden. Nach Abschluss des Vorgangs werden die beiden im *NTS-Unicast-Service*-Modul enthaltenen *Client-Unicast-* und *Server-Unicast*-Module initialisiert und entsprechend der Konfigurationsdatei eingestellt. Ist dieses erfolgreich, so gilt das *NTS-Unicast-Service*-Modul als initialisiert, worauf dieses den entsprechenden Zustand setzt.

4.4.2 Verarbeitungsprozess im NTS-Unicast-Service-Modul

Nach der Initialisierung ist das *NTS-Unicast-Service*-Modul bereit, Nachrichten zu verarbeiten oder zu generieren. Ruft NTP die *processNtpMessage*-Funktion des Moduls auf, so startet die Verarbeitungsroutine immer nach demselben Schema. Der erste Schritt ist hierbei die Überprüfung des Modulzustandes. Dieser muss als initialisiert gelten, damit die Verarbeitung fortgeführt werden kann. Ist dies nicht der Fall, so werden entsprechende Anfragen zur Verarbeitung oder Generierung von Nachrichten an dieser Stelle verworfen. Gilt der Zustand als initialisiert, beginnt das *NTS-Unicast-Service*-Modul mit Verarbeitung der entsprechenden Anfrage. Dazu werden die Erweiterungsfelder aus dem übergebenen NTP-Paket untersucht, die durch den Funktionsaufruf der *processNtpMessage*-Methode als erster Parameter (*receivedPackage*) übergeben wird. Beinhaltet dieses keine Erweiterungsfelder, so werden die übergebenen Parameter direkt das *Client-Unicast*-Modul übergeben. Dieses generiert – sofern es aktiviert ist – eine entsprechende Nachricht anhand des jeweiligen Verbindungszustandes. Eine Weiterleitung an das *Server-Unicast*-Modul ist ausgeschlossen, da dieses immer eine gültige NTS-Nachricht im Erweiterungsfeld der NTP-Nachricht voraussetzt. Befinden sich im *NtpNtsInterface*-Objekt des *receivedPackage*-Parameters hingegen ein oder mehrere Erweiterungsfelder, so findet deren Parsung und Überprüfung statt. Dabei wird unter anderem die Anzahl und Position der Erweiterungsfelder geprüft. NTS-Nachrichten müssen somit immer vor dem NTS-MAC stehen, wobei fremde Erweiterungsfelder ignoriert

werden. Diese dürfen sowohl vor, zwischen oder hinter den NTS-Erweiterungsfeldern enthalten sein. Des Weiteren muss immer ein MAC-Erweiterungsfeld enthalten sein, wenn es sich bei der NTS-Nachricht um eine *time_request*- oder *time_response*-Nachricht handelt. Ist eine andere NTS-Nachricht enthalten, so darf kein MAC enthalten sein. Ist diese Überprüfung abgeschlossen und gilt der Aufbau der Erweiterungsfelder als gültig, so werden die Parameter der *processNtpMessage*-Methode je nach enthaltener NTS-Nachricht korrekt weitergeleitet. Clientnachrichten werden somit an das *Server-Unicast*-Modul und Servernachrichten ans das *Client-Unicast*-Modul durchgereicht. Dies ist natürlich nur dann möglich, wenn die entsprechenden Module über die Konfigurationsdatei aktiviert wurden. Die Verarbeitungsergebnisse dieser Module werden anschließend über den zweiten Parameter (*newResponsePackage*) der *processNtpMessage*-Funktion an die NTP-Implementierung zurückgeliefert, welches das enthaltene NTP-Paket anschließend übertragen kann.

4.4.3 Verarbeitungsprozess im Client-Unicast-Modul

Der Verarbeitungsroutine im *Client-Unicast*-Modul ist durch zwei separate Funktionen getrennt, die unabhängig voneinander genutzt werden können. Eine Funktion dient der Überprüfung von empfangenen Nachrichten, in der auch die enthaltenen Verbindungsparameter extrahiert und gespeichert werden. Die andere Funktion ermöglicht die Generierung neuer NTS-Nachrichten auf Basis des aktuellen Zustands zum jeweiligen Zeitserver.

4.4.3.1 Verarbeitung empfangener NTS-Nachrichten

Die Verarbeitung von NTS-Nachrichten erfolgt durch den Aufruf der folgenden Methode, die das *Client-Unicast*-Modul bereitstellt:

Listing 4.9: Aufruf der processResponseMessage()-Methode im Client-Unicast-Modul

```
void processResponseMessage(NtpNtsInterface &receivedPackage);
```

Dieses erwartet als Parameter ein *NtpNtsInterface*-Objekt, in dem sich das empfangene NTP-Paket befindet. Daraus extrahiert das Clientmodul als erstes die IP-Adresse des Zeitservers und lädt die entsprechende Sitzung. Existiert diese für den jeweiligen Zeitserver noch nicht, so wird diese automatisch angelegt und entsprechend der Konfigurationsdaten initialisiert. Im zweiten Schritt werden die im NTP-Paket enthaltenen Erweiterungsfelder geparst und nochmalig auf Konsistenz und Inhalt überprüft. Dies wirkt redundant, da diese Überprüfung zuvor auch im *NTS-Unicast-Service*-Modul stattgefunden hat. Der Grund dafür liegt im modularen Aufbau der NTS-Implementierung, bei dem die einzelnen Module strikt voneinander getrennt sind. Während der Überprüfung findet allerdings auch die Identifizierung

der enthaltenen NTS-Nachricht statt, die das Clientmodul zunächst als separate Information speichert. Befindet sich neben der NTS-Nachricht auch ein MAC in den Erweiterungsfeldern, so wird dieser zunächst extrahiert und gespeichert. Anschließend löscht das Clientmodul dieses Erweiterungsfeld und serialisiert das NTP-Paket durch Funktionsaufrufe, die als öffentliche Schnittstelle im *NtpNtsInterface*-Objekt verfügbar sind. Das serialisierte NTP-Paket wird nun von den entsprechenden Funktionen zurückgegeben und für die spätere Überprüfung des MAC gespeichert. Im nächsten Schritt gleicht das Clientmodul den empfangenen Nachrichtentyp mit seinem Sitzungszustand ab. Hier wird demnach überprüft, ob die empfangene Nachricht auch erwartet wurde. Befindet sich der Client vom Zustand her in der Cookie-Austauschphase, so erwartet dieser folglich eine *server_cook*-Nachricht. Das Empfangen anderer NTS-Nachrichten führt an dieser zum Abbruch der weiteren Verarbeitung. Passt die Nachricht jedoch zum Zustand, ist die Verarbeitung fortzuführen und es erfolgt anschließend die Prüfung des Fehlercodes der empfangenen NTS-Nachricht. Ist dieser ungleich null, so führt dies zur weiteren Untersuchung des Fehlers. Je nach Art des Fehlers und verwendeter Konfiguration bricht der Client die aktuelle Verarbeitung ab und setzt einen entsprechenden Modulzustand, um den weiteren Kommunikationsverlauf zu lenken (z.B. zurücksetzen der bisherigen Kommunikation). Hat ein Fehlercode eine Zustandsänderung zur Folge, führt dies auch zur Zurücksetzung interner Zähler für Verbindungsversuche und Verbindungs-Time-Outs. Diese werden beim Absenden einer Nachricht gesetzt, um weitere Verbindungsprobleme zu erkennen. Die Überprüfung dieser Werte erfolgt allerdings nur bei der Generierung der Nachrichten (siehe dazu Kapitel 4.4.3.2). Enthält die empfangene NTS-Nachricht keinen Fehler, so wird mit der Überprüfung des MAC fortgefahren. Hierzu lädt das Clientmodul die ausgehandelten Algorithmen aus der aktiven Sitzung und berechnet unter Einbeziehung des serialisierten NTP-Pakets den MAC. Dieser wird anschließend mit dem zuvor extrahierten und gespeicherten MAC verglichen. Sind diese identisch, so gelten die NTS-Nachricht und das NTP-Paket als gültig. Dem ist hinzuzufügen, dass die Berechnung des MAC ausschließlich bei NTS-Nachrichten erfolgt, die Zeitnachrichten beinhalten.

Nach diesen Überprüfungsroutinen erfolgt nun die Verarbeitung der NTS-Nachricht. Diese wird je nach verwendetem Archetyp entsprechend dekodiert und geparst. Im Anschluss folgt die Verifikation der enthaltenen Daten, die den entsprechenden Rahmen der NTS-Spezifikation einzuhalten haben. Ist dieser Vorgang erfolgreich abgeschlossen, so werden die benötigten Parameter in der aktiven Sitzung gespeichert. Nachfolgend wird der neue Verbindungszustand gesetzt, wodurch nun eine entsprechende Antwortnachricht generiert werden kann. Interne Zähler für Verbindungsversuche und Verbindungs-Time-Outs werden an dieser Stelle auch zurückgesetzt. Abschließend erfolgt durch das Clientmodul der Aufruf der *setUseMessageForTimeSync*-Funktion im *NtpNtsInterface*-Objekt, in dem die explizite Freigabe der im NTP-Paket enthaltenen Zeitdaten bewilligt oder abgelehnt wird. Die Freigabe erfolgt ausschließlich bei NTP-Paketen, die eine *time_response*-Nachricht beinhalten und bei denen alle Überprüfungen erfolgreich waren.

4.4.3.2 Generierung neuer NTS-Nachrichten

Die Generierung neuer NTS-Nachrichten erfolgt durch den Aufruf der folgenden Funktion, die im *Client-Unicast*-Modul bereitsteht:

Listing 4.10: Aufruf der createRequestMessage()-Methode im Client-Unicast-Modul

```
void createRequestMessage(NtpNtsInterface &newResponsePackage,
                          bool force = false);
```

Diese erwartet als ersten Parameter ein *NtpNtsInterface*-Objekt, in dem sich das zu übertragene NTP-Paket befindet. Der zweite Parameter ist optional und erzwingt die Generierung einer Nachricht, wenn der Parameter den Wert *true* einnimmt. Ansonsten trifft das Clientmodul anhand der Konfiguration die alleinige Entscheidung, ob eine Nachricht generiert werden soll oder nicht. Im ersten Schritt extrahiert das Clientmodul die IP-Adresse des Zeitservers aus dem *newResponsePackage*-Parameter und lädt die entsprechende Sitzung. Anschließend überprüft das Clientmodul die internen Zähler für Verbindungsversuche und die Verbindungs-Time-Outs. Diese werden über die Konfigurationsdatei eingestellt und regulieren das Anfrageintervall des Clients an den entsprechenden Zeitserver. Der angewendete Machanimus funktioniert hierbei wie folgt: Nach der Übertragung einer NTS-Nachricht an den Server, speichert der Client einen entsprechenden Zeitstempel in der aktiven Sitzung. Empfängt der Client innerhalb eines definierten Time-Outs die entsprechende und fehlerfreie Serverantwort, so wird der Zeitstempel zurückgesetzt. Solange das Time-Out nicht überschritten wird, generiert die *createRequestMessage*-Funktion für den entsprechenden Zeitserver keine neue NTS-Anfrage mehr, es sei denn der *force*-Parameter ist auf *true* gesetzt. Durch diesen Mechanismus sendet der Client trotz *NTP-Polling* nicht immer eine NTS-Nachricht an den Zeitserver. Überschreitet das Time-Out hingegen den definierten Grenzwert oder empfängt der Client eine falsche bzw. fehlerhafte NTS-Nachricht, so generiert dieser eine neue Nachricht und sendet sie an den Server. Der Zeitstempel wird hierbei aktualisiert und der Zähler für die Verbindungsversuche erhöht sich um eins. Dieser Vorgang wiederholt sich erneut, wenn derartige Probleme weiterhin auftreten. Überschreiten die Verbindungsversuche einen definierten Schwellwert, erfolgt die Zurücksetzung der jeweiligen Kommunikation durch Änderung des Verbindungszustandes auf die *Zugriffsphase*. Die Zurücksetzung betrifft auch die internen Zähler für Verbindungsversuche und die Verbindungs-Time-Outs. Zudem ist bei entsprechenden Fehlschlägen auch die Blockierung des Servers möglich, dessen Sperrperiode in der Konfiguration anzupassen ist.

Entscheidet sich der Client für die Generierung einer Nachricht, so wird zunächst ein entsprechender Archetyp erzeugt und befüllt. Dies geschieht auf Grundlage der NTS-Spezifikation und nutzt die bereits gespeicherten Daten aus der aktiven Sitzung. Anschließend erfolgt die Einbettung der Nachricht als neues Erweiterungsfeld in das NTP-Paket, sowie die Aktualisierung der Zeitstempel, die sich im entsprechenden NTP-Paket befinden. Letzteres

geschieht durch den Aufruf der *updatePackageTimestamps*-Funktion aus dem *NtpNtsInterface*-Objekt. Dies eliminiert die Zeiten, die NTS für die bisherige Verarbeitung und Generierung der NTS-Nachricht benötigte. Handelt es sich bei der Nachricht um eine *time_request*-Nachricht, so muss an dieser Stelle noch der NTS-MAC berechnet werden. Dazu speichert das Clientmodul den im NTP-Paket enthaltenen *transmit_timestamp* in die aktuelle Sitzung, da dieser bei der späteren Überprüfung der Antwortnachricht zur Anwendung kommt. Die folgenden Schritte sind zeitkritisch und sollten so wenig Zeit wie möglich beanspruchen, da die NTP-Zeitstempel nicht mehr aktualisiert werden können und die zusätzlichen Berechnungszeiten als Fehler in den Offset der NTP-Implementierung einfließen. Zunächst wird das NTP-Paket serialisiert und zusammen mit den benötigten Algorithmen zur MAC-Berechnung herangezogen. Der MAC wird anschließend als neues Erweiterungsfeld hinzugefügt und liegt somit direkt hinter der NTS-Nachricht. Abschließend wird die Schnittstellenfunktion *setSendResponseMessage(true)* aufgerufen, wodurch die Übertragung des NTP-Pakets explizit freigegeben wird.

4.4.4 Verarbeitungsprozess im Server-Unicast-Modul

Die Verarbeitung von Clientnachrichten und Generierung von Servernachrichten geschieht durch den Aufruf folgender Funktion, die im *Server-Unicast*-Modul bereitgestellt wird:

Listing 4.11: Aufruf der processNtpMessage()-Methode im Server-Unicast-Modul

```
void processNtpMessage(NtpNtsInterface &receivedPackage,
                       NtpNtsInterface &newResponsePackage);
```

Das Servermodul nutzt im Gegensatz zum Clientmodul nur eine Funktion, die sowohl die erforderlichen Überprüfungen vornimmt, als auch die entsprechende Antwortnachricht generiert. Der Grund hierfür liegt an der serverseitigen Zustandslosigkeit, wodurch eine funktionale Trennung der Routinen in separate Funktionen einen höheren Implementierungsaufwand mit sich bringt. Die *processNtpMessage*-Funktion ist was die Parametrierung angeht mit der gleichnamigen Funktion im *NTS-Unicast-Service*-Modul identisch. Die Methode besitzt auch hier zwei Parameter, in denen jeweils ein *NtpNtsInterface*-Objekt übergeben wird. Der erste Parameter beinhaltet hierbei das empfangene NTP-Paket und der zweite kommt für die zugehörige Antwortnachricht zum Einsatz.

Erfolgt nun der Methodenaufruf durch das *NTS-Unicast-Service*-Modul, so prüft das Servermodul zunächst seinen aktuellen Zustand. Gilt dieses als initialisiert, werden im folgenden Schritt die Erweiterungsfelder des empfangenen NTP-Pakets untersucht. Nachdem das Servermodul eine NTS-Nachricht fand, extrahiert es diese und speichert sie temporär. Handelt

es sich bei dabei um eine *time_request*-Nachricht, so muss auch ein dazugehöriger NTS-MAC im NTP-Paket enthalten sein. Dieses wird ebenso extrahiert und für spätere Überprüfungsschritte gespeichert. Anschließend wird das MAC-tragende Erweiterungsfeld aus dem NTP-Paket gelöscht. Nach dessen Entfernung wird nun besagtes NTP-Paket serialisiert und ebenfalls gespeichert. Ist trotz *time_request*-Nachricht kein MAC vorhanden, so stellt dieses eine ungültige NTP-Nachricht dar, worauf das Servermodul einen entsprechenden Fehlercode erzeugt. Gleiches gilt auch für NTP-Pakete, die zwar einen NTS-MAC beinhalten, aber keine *time_request*-Nachricht. Ist die bisherige Verarbeitung jedoch soweit gültig, so wird als nächstes der Inhalt der NTS-Nachricht überprüft. Diese muss hierbei die Vorgaben der NTS-Spezifikation erfüllen. Wenn dies jedoch nicht der Fall oder die NTS-Nachricht fehlerhaft ist, generiert das Servermodul auch hier einen Fehlercode. Ist die Verifikation allerdings erfolgreich verlaufen, so wird die letzte Überprüfungsroutine gestartet. Diese überprüft den empfangenen MAC, sofern es durch die NTS-Nachricht (z.B. bei Zeitnachrichten) verlangt wird. Dazu wird mit Hilfe des serialisierten NTP-Pakets und den empfangenen Algorithmen ein MAC generiert, der anschließend mit den zuvor gespeicherten MAC verglichen wird. Sind die beiden identisch, dann liefert es dem Servermodul den Beweis, dass das jeweilige NTP-Paket nicht manipuliert wurde und vom korrekten NTS-Client stammt. Sind diese jedoch unterschiedlich, so stellt es einen Fehler dar und das Servermodul setzt einen entsprechenden Fehlercode.

Nach Abschluss aller Überprüfungsroutinen ist nun die Antwortnachricht zu generieren, die zur jeweiligen Clientanfrage passt. Dies ist allerdings nur dann der Fall, wenn bisher keine Fehler aufgetreten sind und somit der Fehlercode den Wert Null repräsentiert. Andernfalls wird dieser Schritt übersprungen und die zu übertragene NTS-Nachricht bleibt leer. In solchen Fällen wird nur die leere NTS-Nachricht und der entsprechende Fehlercode in das NTP-Paket (im *newResponsePackage*-Parameter) eingebettet. Bei einer fehlerfreien Verarbeitung wird stattdessen die generierte NTS-Nachricht mit dem Fehlercode Null in das NTP-Paket als Erweiterungsfeld hinzugefügt. Anschließend werden durch entsprechende Funktionsaufrufen im jeweiligen *NtpNtsInterface*-Objekt die Zeitstempel des NTP-Pakets aktualisiert, um die von NTS erzeugten Verarbeitungszeiten zu eliminieren. Handelt es sich bei der zu übertragenen NTS-Nachricht um eine *time_response*-Nachricht, so muss an dieser Stelle noch die Berechnung des MAC erfolgen. Dazu wird zunächst das NTP-Paket mit enthaltener NTS-Antwortnachricht serialisiert, welches unter Verwendung der ausgehandelten Algorithmen in die MAC-Berechnung einfließt. Der generierte MAC wird anschließend als zusätzliches Erweiterungsfeld in das NTP-Paket integriert, das sich positionsmäßig hinter der NTS-Nachricht befindet. Abschließend erteilt das Servermodul durch die *setSendResponseMessage*-Funktion die Sendefreigabe an die NTP-Implementierung, worauf das NTP-Paket zum jeweiligen Client übertragen wird.

4.5 Konfigurierung des NTS-Dienstes

Dieser Abschnitt thematisiert die Einstellungsmöglichkeiten des NTS-Dienstes durch die Verwendung entsprechender Konfigurationsdateien. In der aktuellen Version der NTS-Implementierung kommen dazu drei separate Konfigurationsdateien zum Einsatz, die den jeweiligen Modulen zuzuordnen sind. Deren Aufbau folgt der Syntax einer gängigen Initialisierungsdatei (.ini-Datei), wie das folgendes Beispiel zeigt:

Listing 4.12: Exemplarischer Aufbau einer Konfigurationsdatei

```
; exemplarischer Aufbau einer Konfigurationsdatei

[Sektion 1]
Schlüssel 1 = Wert 1
Schlüssel 2 = Wert 2

[Sektion 2]
; Diese Zeile stellt einen gültigen Kommentar dar
# Diese Zeile stellt einen alternativen Kommentar dar

Integer_1   = 123456789
Integer_2   = -1234
Gleitkomma  = 123.456789
Bool        = true
String      = Dies ist eine beliebige Zeichenkette
```

Derartige Konfigurationsdateien bieten eine einfache Struktur, die sowohl übersichtlich ist, als auch leicht über gewöhnliche Editoren angepasst werden kann. Für den aktuellen Stand der NTS-Implementierung stellt dies die beste Lösung dar, um diverse Einstellungen vorzunehmen. Das Parsen und Bearbeiten der Konfigurationsdateien erfolgt über die eigene Implementierung, die direkt in die bestehende NTS-Entwicklung eingebettet ist. Dies macht die Hinzunahme einer externen Bibliothek für diese Anwendung überflüssig. Der implementierte Parser unterstützt alle gängigen Basisdatentypen sowie Zeichenketten und stellt folgende Bedingungen an den Aufbau der Konfigurationsdateien:

- **Jede Sektion muss einmalig sein**. Eine zweite Sektion mit einem bereits verwendeten Namen wird ignoriert.
- **Jeder Schlüssel innerhalb einer Sektion muss einmalig sein**. Weitere Schlüssel mit demselben Namen werden ignoriert.
- **Keine Unterscheidung zwischen Groß- und Kleinschreibung** bei Sektionen und Schlüsselnamen.

- **Kommentarzeilen beginnen mit einem Semikolon** (;), wobei auch das Rautezeichen (#) akzeptiert wird.

- **Leerzeilen sind erlaubt**

- **Tabulatoren und Leerzeichen sind erlaubt**, wenn diese sich zwischen Schlüssel und Gleichheitszeichen (=) oder Gleichheitszeichen (=) und Wert befinden.

- **Zeichen für Sektions- und Schlüsselnamen**, die frei verwendet werden dürfen:
  - lateinische Kleinbuchstaben (a-z)
  - lateinische Großbuchstaben (A-Z)
  - Ziffern (0-9)
  - Unterstriche (_)
  - Punkte (.)
  - Leerzeichen

- **Boolesche Werte werden interpretiert**:
  - als „*wahr*“ gelten die Werte: „*1*“, „*true*“ und „*yes*“
  - als „*falsch*“ gelten die Werte: „*0*“, „*false*“ und „*no*“

- **Werte, die eine Zeichenkette repräsentieren, können alle druckbaren Zeichen enthalten**. Dies beinhaltet auch beliebige Sonderzeichen.

4.5.1 Die globale NTS-Konfigurationsdatei

Die erste Konfigurationsdatei (beispielsweise mit den Namen *NtsGlobal.ini*) wird dem *NTS-Unicast-Service*-Modul zugeordnet und beinhaltet grundlegende Einstellungen zu NTS. Diese muss in einem geeigneten Ordner liegen, wobei der Arbeitspfad der kompilierten Anwendung die beste Möglichkeit darstellt. Die Verwendung von absoluten und relativen Pfaden ist generell möglich. Die Bereitstellung der Daten erfolgt über die NTP-Implementierung, die nach der Instanziierung des *NTS-Unicast-Service*-Moduls die entsprechende Initialisierungsmethode aufruft und den Dateipfad zur Konfigurationsdatei übergibt. Das Laden der Inhalte erfolgt anschließend über das jeweilige NTS-Modul. Die *NtsGlobal.ini* beinhaltet derzeit folgenden Inhalt, der im Anschluss weiter erläutert wird:

Listing 4.13: Aufbau der globalen NTS-Konfiguration (NtsGlobal.ini)

```
[general]
clientMode = true
serverMode = true
clientConfigFile = ./config/nts/NtsClient.ini
serverConfigFile = ./config/nts/NtsServer.ini

[debug]
showConsoleOutput = true
debugLevel = 1
ignoreErrors = false

[logging]
logFile = ./out/log.txt
logFatal = true
logError = true
logWarn = true
logInfo = true
logDebug = true

[ntsMessageIdentifiers]
ClientAccessData = 1.2.840.113549.1.9.16.1.101
ServerAccessData = 1.2.840.113549.1.9.16.1.102
ClientAssocData = 1.2.840.113549.1.9.16.1.103
ServerAssocData = 1.2.840.113549.1.9.16.1.104
ClientCookieData = 1.2.840.113549.1.9.16.1.105
ServerCookieData = 1.2.840.113549.1.9.16.1.106
TimeRequestSecurityData = 1.2.840.113549.1.9.16.1.107
TimeResponseSecurityData = 1.2.840.113549.1.9.16.1.108
NTSMessageAuthenticationCode = 1.2.840.113549.1.9.16.1.109

[certificateFlagIdentifiers]
ntsServerAuth = 1.3.6.1.5.5.7.3.101
ntsServerAuthz = 1.3.6.1.5.5.7.3.102
ntsClientAuthz = 1.3.6.1.5.5.7.3.103

[ntpFieldTypeIdentifiers]
ntsMessageV1 = 0x400
```

Blau markierte Einträge stellen Einstellungen dar, die derzeit in Verwendung sind und angepasst werden können. Rote Einträge werden hingegen noch nicht verarbeitet, da diese in den implementierten Routinen noch nicht berücksichtigt wurden. In den zukünftigen Versionen stehen diese Optionen jedoch zur Verfügung.

Sektion: *general*

Diese Sektion beinhaltet Informationen zur Einstellung des Client- und Servermoduls. Die beiden Einträge *clientMode* und *serverMode* ermöglichen hierbei die Aktivierung bzw. Deaktivierung der jeweiligen Module. So kann ein Teilnehmer entweder als Client oder als Server fungieren, wobei auch beide Rollen gleichzeitig aktivierbar sind. Die Deaktivierung

beider Rollen ist zwar auch möglich, hätte aber eine komplette Deaktivierung von NTS zur Folge. Die Einträge *clientConfigFile* und *serverConfigFile* beinhalten einen Dateipfad, die das *NTS-Unicast-Service*-Modul an das jeweilige Client- bzw. Servermodul übergibt. Sind die entsprechenden Module aktiviert, so werden die im Dateipfad angegeben Konfigurationen von den jeweiligen Modulen geladen. Existieren diese Dateien nicht, so führt dies in NTS zu einem Fehler während der Initialisierungsphase.

Sektion: *debug*

In diesem Bereich können Debug-Einstellungen vorgenommen werden, die zu Diagnosezwecken einzusetzen sind. Der Schlüsseleintrag *showConsoleOutput* aktiviert bei *true* die Ausgabe von Debug-Informationen in der aktiven Konsole. Das *debugLevel* reguliert hierfür die Menge an ausgegebenen Diagnosedaten. Somit kann man kurze Informationen beim Empfang oder Senden von Daten anzeigen oder sämtliche Vorgänge und Parameter einer Verarbeitung plotten lassen. Der Eintrag *ignoreErrors* erlaubt die Verarbeitung von NTS-Nachrichten, die nicht komplett mit der NTS-Spezifikation konform sind. Somit ist die Verarbeitung von Nachrichten möglich, die sonst Aufgrund banaler Fehler abbrechen würde. Dies kann beim Testen unterschiedlicher NTS-Implementierungen nützlich sein, die miteinander kommunizieren. Abweichungen von Nachrichten im Bezug der NTS-Spezifikation werden somit kenntlich gemacht.

Sektion: *logging*

Eine weitere Diagnosemöglichkeit bietet das Loggen von Informationen. An dieser Stelle können unabhängig von den Debug-Einstellungen die Ausgaben gefiltert werden. Das Ausgabeformat der Logdatei sieht in etwa folgendermaßen aus:

Listing 4.14: Beispielhafte Einträge der NTS-Logdatei

```
| INFO    | 18:06:53 | [Server] Server seed refreshed
| DEBUG   | 18:06:57 | [Client] generating EC key pair (sect571r1)
| ERROR   | 18:06:58 | [Server] certificate: authentication fail
| WARNING | 18:06:58 | [Server] an error is occurred.
| WARNING | 18:06:58 | [Server] Error details: CERTIFICATE_INVALID
| SUCCESS | 18:18:51 | [Client] received message: TimeSyncMsg valid
```

Neben diesen Informationen sind noch weitere enthalten, wie beispielsweise der genaue Ursprung der ausgelösten Lognachricht im Quellcode.

Sektion: *ntsMessageIdentifiers*

Diese Sektion beinhaltet die OID-Kennungen der NTS-Nachrichtenobjekte. Diese richten sich derzeit nach dem „*SMI Security for S/MIME CMS Content Type (1.2.840.113549.1.9.16.1)*“ [52], in dem CMS-spezifische Inhalte registriert sind und von der NTS-Spezifikation in Betracht gezogen werden. Die genaue Spezifikation der NTS-bezogenen OIDs existiert noch nicht, wodurch die eingetragenen Werte nur einen temporären Charakter besitzen.

Sektion: *certificateFlagIdentifiers*

Auch diese Sektion beinhaltet die OID-Kennungen, die als zusätzliche *Flags* in die *Extended-Key-Usage*-Erweiterung eingebettet werden sollen. Diese richten sich hierbei nach der „*SMI Security for PKIX Extended Key Purpose Identifier (1.3.6.1.5.5.7.3)*”-Tabelle [52].

Sektion: *ntpFieldTypeIdentifiers*

Die letzte Sektion beinhaltet derzeit nur den Eintrag *ntsMessageV1*. Dieser repräsentiert die vorläufige NTS-Kennung im Feldtyp eines NTP-Erweiterungsfeldes. Wird diese Kennung in einem Erweiterungsfeld erkannt, so ist der zugehörige Inhalt einem NTS-Inhalt der Version 1 zuzuordnen.

4.5.2 Die Konfigurationsdatei des Client-Unicast-Moduls

Die zweite Konfigurationsdatei wird vom Clientmodul verwendet und beinhaltet Informationen über externe Ressourcen sowie über Verbindungs- und Sicherheitsparameter. Theoretisch kann dabei jede vom Client erzeugte Sitzung über eine eigene Konfigurationsdatei verfügen. Dies wird in der aktuellen Implementierung jedoch noch nicht berücksichtigt und stellt eine mögliche Erweiterung dar, die zukünftig bereitgestellt werden könnte. Die Konfigurationsdatei sieht dabei wie folgt aus:

Listing 4.15: Aufbau der Konfigurationsdatei für das Client-Unicast-Modul

```
[general]
useIssuedCertificate = true
issuedCertFile = ./out/clientCert.pem
privateKeyFile = ./out/clientKey.pem
rootCaBundleFile = ./out/rootCaBundle.pem
messageResponseTimeout = 10
maxConnectionAttempts = 3
allowServerBlocking = true
maxBlockingPeriod = 3600
acceptSelfSignedServerCertificates = false

[connectionRequirements]
needServerAuthentication = true
needServerAuthorization = true

[certificateGenerator]
serialNumber = 15224
periodOfValidityInDays = 365
periodOffsetInDays = -182
subjectC = DE
subjectST = Braunschweig
subjectL = Wolfenbuettel
subjectO = Ostfalia
subjectOU = Fakultaet Elektrotechnik
subjectCN = NTS-Anwender

[rsaEncryptionKey]
bitLength = 4096
exponent = 65537

[negotiationLists]
supportedNtsVersions = 3, 2, 1
macAlgorithms = sha512, sha384, sha256, ripemd160
contentEncryptAlgorithms = aes-256-cbc, aes-192-cbc, aes-128-cbc
keyEncryptAlgorithms = c2tnb431r1, sect571r1, rsa, rsaEncryption
```

Sektion: *general*

Dieser Bereich beinhaltet zum einen die Dateipfade zu den verwendeten Zertifikaten bzw. Schlüsseln und zum anderen die Verbindungsparameter. Der *useIssuedCertificate*-Eintrag legt an dieser Stelle die Art des Zertifikats fest. Ist der Wert auf *true* gesetzt, so verwendet der Client das Zertifikat bzw. die Zertifikatskette, welche(s) sich hinter dem *issuedCertFile*-Eintrag verbirgt. Des Weiteren wird dann auch der zugehörige private Schlüssel verwendet, der über den *privateKeyFile*-Eintrag bezogen wird. Während der Modulinitialisierung werden diese Ressourcen geladen und anschließens überprüft. Der private Schlüssel muss hierfür zu dem jeweiligen Zertifikat passen. Außerdem darf das Zertifikat kein selbstgeneriertes sein, sondern muss von einem Zertifikatsaussteller stammen. Wird der *useIssuedCertificate*-Eintrag hingegen auf *false* gesetzt, so werden die genannten Einträge ignoriert und der Client

generiert während der Verhandlungsphase mit dem Server sein eigenes Schlüsselpaar und das entsprechende Zertifikat. In der Regel ist es sinnvoll, diesen Eintrag auf *false* zu stellen, da hierfür kein ausgestelltes Zertifikat nötig ist. Dieses wird nur dann benötigt, wenn der Server eine Autorisierung vom NTS-Client verlangt. Neben den genannten Einträgen und deren Konfiguration wird der *rootCaBundleFile*-Eintrag immer benötigt. Dieser verweist auf einen Zertifikatsstapel, in dem sich Root-CA-Zertifikate befinden und wodurch die Überprüfung von Serverantworten ermöglicht werden.

Der zweite Bestandteil dieser Sektion sind die Verbindungsparameter, in dem die Zustandsregelung beeinflusst wird. Der *messageResponseTimeout*-Eintrag stellt dazu das Verbindungs-Time-Out in Sekunden ein. Wird nach dieser Zeit keine korrekte Serverantwort zur jeweiligen Clientanfrage empfangen, so gilt dies als gescheiterter Verbindungsversuch und eine neue Anfrage wird generiert. Die Anzahl der zulässigen Verbindungsversuche wird über den Eintrag *maxConnectionAttempts* festgelegt. Wird dieser Schwellwert überschritten, so wird die aktuelle Verbindung zum Server zurückgesetzt und der Client beginnt mit der erneuten Generierung der Zugriffsnachricht. Ist eine Einigung mit einem Server nicht möglich oder ist die Verbindung zu schlecht (häufige Fehler/Abbrüche), so kann ein Server über den *maxBlockingPeriod*-Eintrag für eine definierte Zeit in Sekunden geblockt werden, sofern der *allowServerBlocking*-Eintrag aktiviert wurde. Da eine Blockierung des Servers durch Fehlercodes angegriffen werden kann, ist diese experimentelle Einstellung derzeit deaktiviert. Auch der *acceptSelfSignedServerCertificates*-Eintrag ist durch den experimentellen Charakter derzeit abgeschaltet. Wird dieser Eintrag auf *true* gesetzt, so werden selbstsignierte Serverzertifikate akzeptiert. Dieses Szenario ist in lokalen Netzwerken denkbar (z.B. größere Unternehmen), in dem eine interne Zertifikatstruktur vorhanden ist und der NTS-Server gleichzeitig eine Root-CA darstellt. Dieses Szenario muss allerdings noch ausgiebig analysiert und getestet werden, bevor die Freigabe dieser Einstellung erfolgt.

Sektion: *connectionRequirements*

In dieser Sektion können zusätzliche Einstellungen zur Sicherheit getätigt werden, in dem der Client weitere Verbindungseigenschaften festlegt. Wird der Wert des *needServerAuthentication*-Eintrags auf *true* gesetzt, so verlangt der Client die entsprechende Authentifikation des Zeitservers durch seinen Zertifikatsaussteller. Das Serverzertifikat muss also die *id-kp-ntsServerAuth*-Kennung in der *Extended-Key-Usage*-Zertifikatserweiterung beinhalten, damit dieses vom Client akzeptiert wird. Wird hingegen der Wert des *needServerAuthorization*-Eintrags auf *true* gesetzt, so verlangt der Client die entsprechende Autorisierung des Servers. Dazu muss der Server die *id-kp-ntsServerAuthz*-Kennung in der *Extended-Key-Usage*-Zertifikatserweiterung beinhalten, damit dieses Zertifikat vom Client akzeptiert wird. Dem ist hinzuzufügen, dass beide Einstellungen optional sind und auch gleichzeitig aktiviert oder deaktiviert sein können.

Sektion: *certificateGenerator*

In diesem Bereich befinden sich die Einstellungen, die der Client zur Generierung des Zertifikats verwendet. Diese kommen allerdings nur dann zur Anwendung, wenn in der *general*-Sektion der *useIssuedCertificate*-Eintrag auf *true* gesetzt ist. Der *serialNumber*-Eintrag konfiguriert, wie der Name vermuten lässt, die Seriennummer des Zertifikats. Diese kann frei gewählt werden und liegt im Wertebereich von 1 bis $2^{32} - 1$. Der *periodOfValidityInDays*-Eintrag legt die Gültigkeitsdauer des Clientzertifikats fest. Diese beginnt zum Zeitpunkt der Zertifikaterstellung und endet nach einer festgelegten Anzahl von Tagen, die durch diesen Wert repräsentiert wird. Problematisch ist hierbei der Fall, wenn ein Clientzertifikat vom Server als ungültig betrachtet wird, weil diese Gültigkeitsperiode aufgrund der falschen Uhrzeit des Clients (wenn diese vorgeht) in der Zukunft liegt. Um diesem Problem entgegenzuwirken, kommt der der *periodOffsetInDays*-Eintrag ins Spiel. In diesem kann das Erstellungsdatum um eine angegebene Anzahl von Tagen verschoben werden. Ein negativer Wert bedeutet in diesem Fall, dass der Beginn der Gültigkeitsperiode um die angegebene Anzahl von Tagen in die Vergangenheit verschoben wird. Das Ende der Periode verschiebt sich dabei entsprechend mit. Diese Einstellung ist in der aktuellen Version der NTS-Implementierung allerdings noch nicht vorhanden und wird noch hinzugefügt. Neben der Gültigkeitsperiode kann in dieser Sektion noch der Subjektname (Name des Zertifikatsinhabers) eingestellt werden. Dieser ist frei wählbar, wobei längere Namen entsprechend größere Zertifikate und somit größere NTS-Nachrichten erzeugen.

Sektion: *rsaEncryptionKey*

In dieser Sektion befinden sich die Einstellungen, die der Client zur Generierung eines RSA-Schlüsselpaares benötigt. Dies ist zum einen der Exponent und zum anderen die Schlüssellänge. Beide Einstellungen müssen gültige Werte besitzen, die dem RSA-Kryptosystem gerecht werden (s.a. Kapitel 2.4.1.2) . Als Exponent werden somit die Werte 3, 5, 17, 257 oder 65537 erwartet. Typische Bitlänge sind hingegen 1024, 2048 oder 4096. Diese kommen allerdings nur dann zur Anwendung, wenn in der *general*-Sektion der *useIssuedCertificate*-Eintrag auf *true* gesetzt ist. Außerdem müssen Client und Server das RSA-Verfahren als asymmetrischen Algorithmus ausgehandelt haben, damit diese Einstellungen wirksam werden.

Sektion: *negotiationLists*

Diese Sektion beinhaltet alle Parameter, die der Client aktiv mit dem Server aushandelt. Dabei beinhaltet der *supportedNtsVersions*-Eintrag eine Liste mit allen vom Client unterstützten NTS-Versionen, die jeweils mit einem Komma getrennt sind. In der aktuellen Implementierungsversion wird diese Liste jedoch nicht ausgewertet, da derzeit nur eine NTS-

Version existiert. Die restlichen Einträge (*macAlgorithms*, *contentEncryptAlgorithms* und *keyEncryptAlgorithms*) beinhalten die kryptografischen Algorithmen, die ebenfalls vom Client unterstützt werden. Hierbei muss jede Liste mindestens ein Eintrag enthalten, damit das Clientmodul korrekt initialisiert werden kann. Die jeweiligen Algorithmen und deren Bezeichnung werden hierbei durch OpenSSL bestimmt. Die komplette Liste aller verfügbaren Algorithmen können durch die Eingabe entsprechender Befehle im Linux-Terminal eingesehen werden (z.B. ECC-Algorithmen: *openssl ecparam -list_curves*). An dieser Stelle sei noch gesagt, dass die Einträge dieser Sektion eine Priorisierung unterstützen. Ein Wert, der sich direkt rechts neben dem Gleichheitszeichen befindet, erhält die höchste Priorität. Die Priorität der folgenden Werte, die durch Kommata getrennt sind, nimmt anschließend laufend ab. Somit besitzt der letzte Wert einer Liste die geringste Priorität. Ein Client wird somit bei der Aushandlung der Algorithmen die höherpriorisierten bevorzugen.

4.5.3 Die Konfigurationsdatei des Server-Unicast-Moduls

Die dritte Konfigurationsdatei wird vom Servermodul verwendet und beinhaltet ebenfalls Informationen über externe Ressourcen, sowie über den Verbindungs- und Sicherheitsparametern. Die Konfigurationsdatei sieht dabei wie folgt aus:

Listing 4.16: Aufbau der Konfigurationsdatei für das Server-Unicast-Modul

```
[general]
useIssuedCertificate = true
issuedCertFile = ./out/serverCert.pem
privateKeyFile = ./out/serverKey.pem
rootCaBundleFile = ./out/rootCaBundle.pem
seedRefreshPeriod = 86400
accessKeyHashAlgo = sha512
sendErrorCode = true
useSubjectKeyIdentifier = true

[connectionRequirements]
needClientAuthorization = true

[negotiationLists]
supportedNtsVersions = 1, 2, 3
macAlgorithms = sha512, sha384, sha256, ripemd160
contentEncryptAlgorithms = aes-256-cbc, aes-192-cbc, des-ede3-cbc
keyEncryptAlgorithms = prime239v3, sect571r1, rsaEncryption
```

Sektion: *general*

In diesem Bereich befinden sich die Dateipfade zu den verwendeten Zertifikaten und Schlüsseln des Servers sowie weitere grundlegende Einstellungen. Der *useIssuedCertificate*-Eintrag legt hierbei die Art des verwendeten Zertifikats fest. Ist der Wert auf *true* gesetzt, so verwendet der Server das Zertifikat bzw. die Zertifikatskette, welche(s) sich hinter dem Eintrag *issuedCertFile* verbirgt. Da derzeit keine selbstgenerierten Serverzertifikate unterstützt werden, wird der jeweilige Wert nicht ausgelesen. Dieser ist derzeit immer auf *true* festgesetzt und kann nicht geändert werden. Neben dem Zertifikat benötigt der Server auch einen privaten Schlüssel zur Generierung der digitalen Signatur, den er über den *privateKeyFile*-Eintrag bezieht. Während der Modulinitialisierung werden die Zertifikate und der Schlüssel geladen und anschließend überprüft. Der private Schlüssel muss an dieser Stelle zu dem jeweiligen Zertifikat des Servers passen. Außerdem darf das Zertifikat kein selbstgeneriertes sein, sondern muss immer von einem Zertifikatsaussteller stammen. Der *rootCaBundleFile*-Eintrag verweist auf einen Zertifikatsstapel, in dem sich Root-CA-Zertifikate befinden. Diese werden zur Überprüfung der Clientzertifikate benötigt, sofern der Server eine Autorisierung verlangt.

Neben der Zertifikatstruktur befinden sich noch weitere Einstellungen in dieser Sektion. So ermöglicht der *seedRefreshPeriod*-Eintrag die Konfiguration des *Server Seed*. Der entsprechende Wert gibt hierbei das Aktualisierungsintervall in Sekunden an. Der *accessKeyHashAlgo* beinhaltet den vom Server verwendeten Hash-Algorithmus, der bei der Berechnung des Zugriffsschlüssels in den *server_access*- und nach Erhalt der *client_assoc*-Nachrichten zur Anwendung kommt. Der darauffolgende *sendErrorCode*-Eintrag wird in der aktuellen Implementierung noch nicht berücksichtigt und ist standardmäßig auf *true* gesetzt. Dieser Eintrag ermöglicht die Aktivierung bzw. Deaktivierung von Fehlernachrichten. Da diese derzeit aktiviert ist, überträgt der Server bei Verarbeitungsfehlern einer Clientnachricht den entsprechenden Fehlercode. Wäre dieser Eintrag deaktiviert, so würde der Server die Nachricht einfach verwerfen, ohne dem Client eine Rückmeldung zu geben. Der *useSubjectKeyIdentifier* ist der letzte Eintrag der Sektion und hat eher experimentellen Charakter. Diese Option aktiviert bzw. deaktiviert die Verwendung des *Subject Key Identifiers* in der *EnvelopedData*-CMS-Struktur. Dies beeinflusst die Nachrichtenarchitektur der *server_cook*-Nachricht und somit auch deren Größe. Eine Vorgabe seitens NTS-Spezifikation gibt es nicht, allerdings ist die Aktivierung des Eintrags – aufgrund der kleineren Nachrichtengröße – sinnvoll.

Sektion: *connectionRequirements*

In dieser Sektion können zusätzliche Einstellungen zur Sicherheit getätigt werden, indem der Server weitere Verbindungseigenschaften festlegt. Wird dazu der *needClientAuthorization*-Eintrag auf *true* gesetzt, so verlangt der Server die entsprechende Autorisierung des Clients. Dazu muss der Client die *id-kp-ntsClientAuthz*-Kennung in der *Extended-Key-*

Usage-Erweiterung seines Zertifikats beinhalten, damit dieses vom Server akzeptiert wird. Außerdem muss dieses Zertifikat von einer Zertifizierungsstelle ausgestellt worden sein, die dem Server eine Verifizierung ermöglicht.

Sektion: *negotiationLists*

Dieser Bereich beinhaltet alle Parameter, die der Server aktiv mit dem Client aushandelt. Die Einträge und deren Verarbeitung sind mit den entsprechenden Einträgen aus der Konfigurationsdatei des Clientmoduls identisch. Hierbei stellt der *supportedNtsVersions*-Eintrag eine Liste mit allen vom Server unterstützten NTS-Versionen, wobei derzeit nur Version 1 existiert und somit diese Liste nicht verarbeitet wird. Die restlichen Einträge (*macAlgorithms*, *contentEncryptAlgorithms* und *keyEncryptAlgorithms*) beinhalten abermals die kryptografischen Algorithmen, die der Server unterstützt. Jede Liste muss dabei mindestens einen Eintrag enthalten, damit die korrekte Initialisierung des Servermoduls möglich ist. Dem ist hinzuzufügen, dass die Einträge dieser Sektion eine Priorisierung unterstützen. Somit besitzt der erste Wert der Liste (direkt neben dem Gleichheitszeichen) die höchste Priorität und der letzte Wert die geringste. Somit präferiert der Server bestimmte Algorithmen, die während der Assoziationsphase mit dem Client festgelegt werden.

4.6 Projektaufbau und Entwicklungsstand

In diesem Unterkapitel werden abschließend noch einige Informationen zum derzeitigen Projektstand aufgeführt. Dazu wird zunächst auf die Projekt- und Ordnerstruktur eingegangen, die hierfür zum Einsatz kommt. Im Anschluss erfolgt eine Übersicht zum derzeitigen Funktionsumfang der NTS-Implementierung und eine Darstellung der Codezusammensetzung. Die hierbei genannten Informationen beziehen sich auf die folgende NTS-Implementierungsversion:

Listing 4.17: Aktuelle Programmversion der NTS-Implementierung:

NTS-Softwareversion: **0.8.0.0 Beta vom 27.08.2016**

verwendete Bibliotheken:

- OpenSSL **v.1.1.0** (vom 25.08.2016)
- asn1c **v0.9.27** (vom 17.09.2014)

4.6.1 Projektorganisation der NTS-Implementierung

Die Organisation des Projektordners lehnt sich in der NTS-Implementierung an gängige Verzeichnisstrukturen an, die in der Programmiersprache C/C++ häufig angewandt werden. Der aktuelle Aufbau wird hierzu in Abbildung 4.12 dargestellt und im Folgenden kurz erläutert.

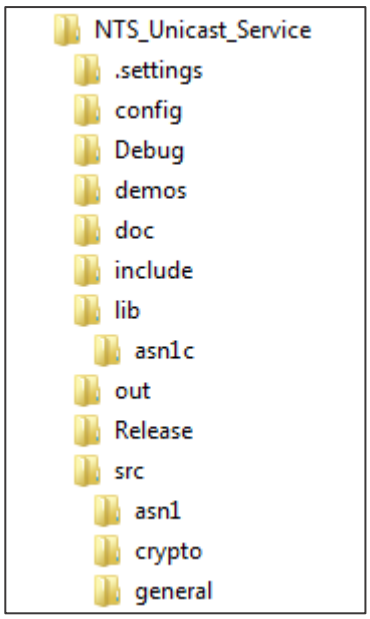


Abbildung 4.12: Ordnerstruktur im Projektverzeichnis der NTS-Implementierung

Ordner: **.settings**: Dieses Verzeichnis beinhaltet interne Projekteinstellungen, die von der verwendeten Entwicklungsumgebung (Eclipse CDT) verwaltet werden.

Ordner: **config**: In diesem befinden sich zusätzliche Konfigurationsdateien für die *Doxygen*-Dokumentation und den *Valgrind*-Profiler. Außerdem liegen hier auch die drei Konfigurationsdateien für die implementierten NTS-Module.

Ordner: **Debug**: Der Debug-Ordner beinhaltet das kompilierte NTS-Programm, welches für Test- und Diagnosezwecke verwendet wird und daher Debug-Informationen beinhaltet.

Ordner: **demos**: Zusätzliche Quellcodedateien zum Demonstrieren oder Testen von Modulen befinden sich in diesem Verzeichnis.

Ordner: **doc**: In diesem Ordner befindet sich die durch Doxygen erzeugte Codedokumentation, die beispielsweise in Form einer Web-Hilfe oder als PDF-Datei vorliegt.

Ordner: **include**: Dieses Verzeichnis beinhaltet die Header-Dateien der NTS-Implementierung, die von NTP zur Verwendung des NTS-Dienstes inkludiert werden müssen.

Ordner: **lib**: Dieser Ordner enthält die von NTS benötigten Bibliotheken. Derzeit befindet sich nur die *asn1c*-Bibliothek in diesem Verzeichnis, da OpenSSL extern über das Betriebssystem (als dynamische Bibliothek) bezogen wird.

Ordner: **out**: Hier finden sich alle Dateien, die durch NTS generiert oder anderweitig genutzt werden. Dies sind beispielsweise Logdateien, Zertifikate und Schlüssel.

Ordner: **Release**: Der Release-Ordner beinhaltet das kompilierte und optimierte NTS-Programm, welches als finale Programmversion zu verwenden ist.

Ordner: **src**: Der *source*-Ordner beherbergt die Codeimplementierungen als cpp-Dateien. Dieses Verzeichnis unterteilt sich in *asn1*, *crypto* und *general*. Die ersten beiden Ordner inklidieren das ASN.1- und Kryptografiemodul. Der letzte Ordner schließt diverse Zusatzfunktionen ein, die in NTS genutzt werden (z.B. der Logger).

4.6.2 Fortschritt und Umfang der NTS-Implementierung

Wie die aktuelle NTS-Programmversion *0.8.0.0 Beta* schon vermuten lässt, ist die Programmentwicklung noch nicht abgeschlossen. Dies ist zum Teil durch die NTS-Spezifikation begründet, die derzeit noch nicht abgeschlossen ist. Änderungen sind von dieser Seite aus definitiv zu erwarten. Des Weiteren sind in NTS noch Funktionen zu implementieren, die zwar optional sind, aber als Bestandteil einer abgeschlossenen Implementierung angesehen werden. Eine solche Funktion wäre die Online-Zertifikatprüfung via OCSP. Weitere Funktionen

werden dazu im Kapitel 6.3.3 thematisiert. Ein weiterer Punkt ist der Test der Software. Unit- und Modultests wurden im Verlaufe der Implementierung durchgeführt, allerdings sind noch ausgiebige Tests und die Erstellung derartiger Testfälle notwendig. Zusätzlich sind Optimierungen sinnvoll, um zeitkritische Programmpassagen zu verkürzen und somit die bei den Zeitnachrichten erzeugten Ungenauigkeiten zu minimieren. Da die momentane Programmversion bereits alle derzeitigen NTS-Spezifikationen einhält, funktionsfähig ist, stabil läuft und intensiv mit Analysewerkzeugen auf Speicherzugriffsfehler untersucht wurde, erhält sie den *Beta*-Status. Nachfolgend werden dazu die Eigenschaften, Komponenten und Funktionen aufgeführt, die in der derzeitigen NTS-Programmversion enthalten sind:

➔ vollständige Implementierung des NTS-Unicast-Betriebsmodus
- enthält Wrapper für die ASN.1- und OpenSSL-Bibliothek
- Umsetzung aller NTS-Nachrichtenobjekte (für Unicast)
- clientseitige Verarbeitung
- serverseitige Verarbeitung

➔ vollständig englischsprachige Codedokumentation mit Doxygen

➔ Einhaltung des RAII-Idioms[60]

➔ Exception-sichere Implementierung mit *Basic*-Garantie[61]

➔ Implementierte *function trace*-Funktion zur optionalen Rückverfolgung von Fehlern.

➔ zusätzlich implementierte Funktionen
- ein Logger, zur formatierten Ausgabe von Ereignissen
- ein Parser zur Verarbeitung und Editierung von Konfigurationsdateien

4.6.3 Implementierungsstatistiken

An dieser Stelle werden abschließend noch einige Statistiken aufgeführt, die sich auf den derzeitigen Entwicklungsstand beziehen. Die folgenden Tabellen und Diagramme haben daher nur einen informativen Charakter und werden sich im Verlaufe der weiteren Programmentwicklung weiter ändern. Die folgenden Tabellen 4.4 und 4.5 stellen hierzu die entwickelten Komponenten dar, die in separate Dateien gespeichert wurden. Außerdem zeigen

[60] Automatische Freigabe von Ressourcen, um Speicherlecks und undefinierte Zustände zu verhindern.
[61] Garantiert, dass bei Ausnahmefehlern (engl. exceptions) immer ein definierter Zustand erreicht wird.

diese Informationen die Anzahl der implementierten Programmfunktionen der jeweiligen Dateien und deren resultierende Dateigrößen.

Tabelle 4.4: Auflistung der Quellcodedateien im NTS-Projekt

NTS-Projektbereich	Dateinamen (.h + .cpp)		Anzahl der Funktionen	Dateigrößen (.h + .cpp)
NTS-Grundfunktionen	ClientSessionInfo		29	24,5 KB
(./…)	NegotiationLists		36	13,8 KB
	NtsExtensionFields		16	13,1 KB
	NtsUnicastClient		28	92,7 KB
	NtsUnicastServer		18	73,2 KB
	NtsUnicastService		9	19,0 KB
	ParticipantInfo		18	10,0 KB
	ServerSessionInfo		17	17,6 KB
			171	**263,8 KB**
ASN.1-Modul	*mit asn1c generierte ASN.1-Module*		20	58,4 KB
(./asn1/…)	Asn1CustomTypes		17	25,8 KB
	Asn1Wrapper		30	40,7 KB
	NtsClientAssocData		68	79,6 KB
	NtsClientCookieData		60	68,9 KB
	NtsExtensionFieldContent		25	24,4 KB
	NtsMessageAuthenticationCode		14	15,9 KB
	NtsServerAccessData		15	15,9 KB
	NtsServerAssocData		89	106,5 KB
	NtsServerCookieData		21	20,9 KB
	NtsTimeRequestSecurityData		30	32,2 KB
	NtsTimeResponseSecurityData		15	16,4 KB
			404	**505,4 KB**
Kryptografiemodul	cms		36	61,5 KB
(./crypto/…)	general	*(Grundfunktionen)*	25	16,6 KB
	helper	*(Hilfsfunktionen)*	17	35,3 KB
	pKey	*(Schlüsselverwaltung)*	23	30,6 KB
	x509	*(Zertifikatsverwatung)*	76	113,0 KB
			177	**256,9 KB**
Zusatzfunktionen	ConfigFile	*(ini-Parser)*	30	36,6 KB
(./general/…)	FunctionTrace		11	7,2 KB
	global	*(OID-Kennungen)*	28	17,4 KB
	globalFunctions		8	5,3 KB
	NtsErr	*(Fehlerbehandlung)*	11	6,7 KB
	NtsLogger	*(Logger)*	11	11,5 KB
			99	**84,7 KB**

Tabelle 4.5: Zusammenfassung der implementierten Funktionen im NTS-Projekt

Projektbereich	Anzahl der Funktionen	Dateigrößen (.h + .cpp)
NTS-Grundfunktionen	171	263,8 KB
ASN.1-Modul	404	505,4 KB
Kryptografiemodul	177	256,9 KB
Zusatzfunktionen	99	84,7 KB
Gesamt	**851**	**1110,9 KB**

Die Diagramme der folgenden Abbildungen 4.13 und 4.14 zeigen den prozentualen Anteil der einzelnen NTS-Module, bezogen auf die Dateigrößen bzw. auf die Funktionen:

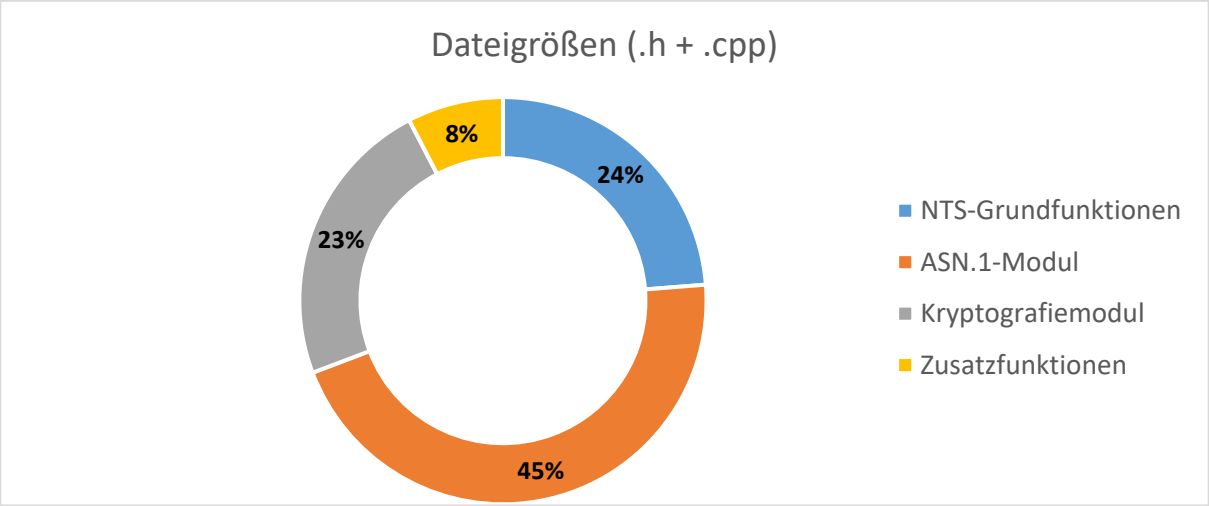


Abbildung 4.13: Prozentuale Verteilung der Dateigrößen in der NTS-Implementierung

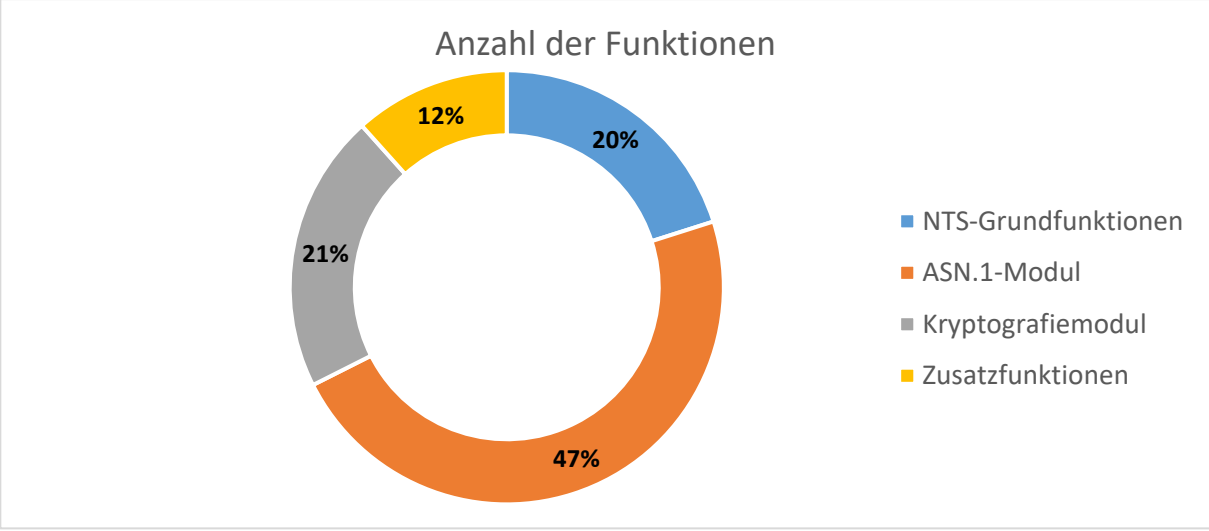


Abbildung 4.14: Prozentuale Verteilung der Funktionen in der NTS-Implementierung

Die nachfolgende Tabelle 4.6 und das dazugehörige Diagramm in Abbildung 4.15 geben eine Übersicht zur Zusammensetzung des NTS-Quellcodes ohne die implementierten Testumgebungen. Die in der Tabelle aufgeführten *gemischten Zeilen* stellen hierbei Zeilen dar, die sich aus Quellcode- und Kommentarzeilen zusammensetzen. Die hohe Anzahl an Kommentarzeilen ist hierbei durch die umfangreiche Codedokumentation mit Doxygen begründet. Natürlich sei hier auch erwähnt, dass die Anzahl der Codezeilen keine Rückschlüsse auf die Qualität des implementierten Programms zulässt. Jedoch zeigt es den Entwicklungsaufwand und den ungefähren Umfang der Implementierung.

Tabelle 4.6: Zusammensetzung des Quellcodes im NTS-Projekt

NTS-Implementierung	gesamte Zeilen	Quell-codezeilen	gemischte Zeilen	Kommentar-zeilen	Leer-zeilen
ASN.1-Modul Wrapper-Funktionen	2508	970	11	1134	393
ASN.1-Modul NTS-Nachrichtenbeschreibungen	2037	1302	338	181	216
ASN.1-Modul NTS-Nachrichtenobjekte	13532	4279	39	6924	2290
ASN.1-Modul Gesamt	18077	6551	388	8239	2899
Kryptografiemodul Allgemein	1933	1019	12	554	348
Kryptografiemodul CMS	2055	1047	14	617	377
Kryptografiemodul pKey (Schlüsselverwaltung)	1173	581	5	368	219
Kryptografiemodul X.509 (Zertifikatsverwaltung)	3933	1908	12	1329	684
Kryptografiemodul Gesamt	9094	4555	43	2868	1628
NTS-Grundfunktionen NTS-Client-Modul (komplett)	2905	1676	7	403	819
NTS-Grundfunktionen NTS-Server-Modul (komplett)	2257	1310	6	311	630
NTS-Grundfunktionen NTS-Unicast-Service-Modul	711	399	1	113	198
NTS-Grundfunktionen Grund- und Zusatzfunktionen	6934	3692	75	1303	1864
NTS-Grundfunktionen Gesamt	12807	7077	89	2130	3511
Gesamte Implementierung	**39978**	**18183**	**520**	**13237**	**8038**

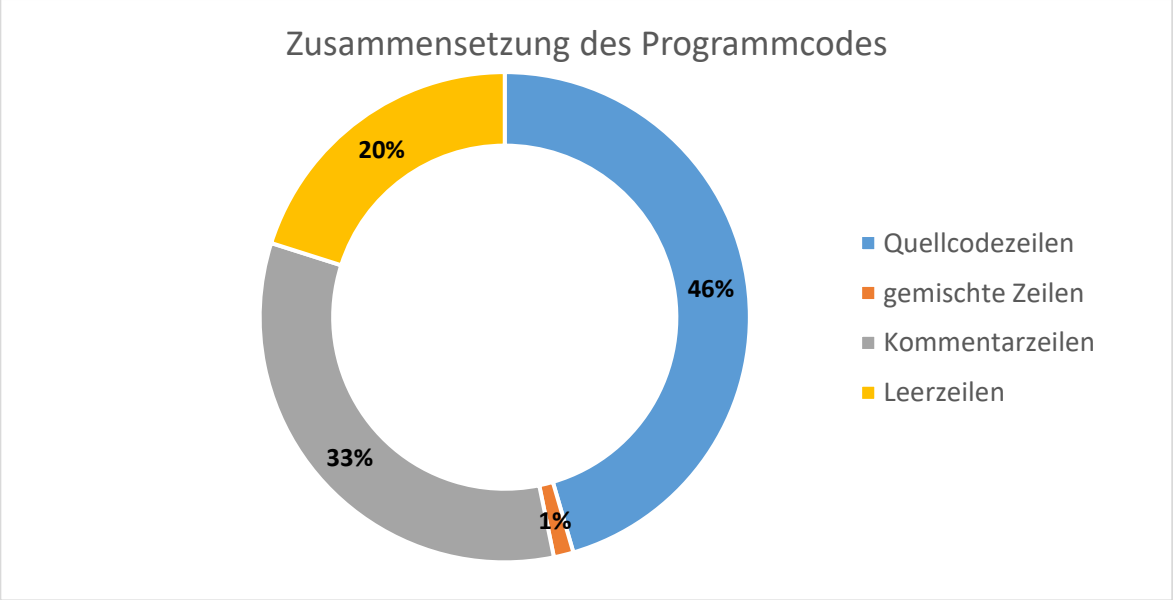


Abbildung 4.15: Quellcode-Zusammensetzung der NTS-Implementierung

5 Tests und Ergebnisse

Dieses Kapitel zeigt vorläufige Testkonfigurationen und -ergebnisse auf, die sich auf den derzeit beschriebenen Stand der NTS-Implementierung beziehen. An dieser Stelle sei ausdrücklich erwähnt, dass der Testumfang nur grundlegende Eigenschaften der Implementierung darstellen soll, um später einen Vergleich zu schaffen und kritische Verhaltensmerkmale aufzudecken. Für ausgiebige Tests sind noch ausstehende Änderungen der NTS-Spezifikationen und grundlegende Optimierungen der NTS-Implementierung einzupflegen, die im nachfolgenden Kapitel 6 näher erläutert werden. Darüber hinaus sind auch entsprechende Testfälle zu entwickeln, mit denen die korrekte Kommunikation zwischen NTS und NTP überprüft werden kann. Nachfolgend werden die bisherigen Messmethoden erläutert und deren Ergebnisse vorgestellt.

5.1 Messung der NTS-Nachrichtengrößen

Das Ziel dieser Messung ist die Ermittlung der NTS-Nachrichtengrößen, die zwischen Client und Server ausgetauscht werden. Das erzeugte Datenvolumen gibt Aufschluss über mögliche Probleme, die in der Praxis auftreten können (z.B. IP-Fragmentierung, Ressourcenbelebung oder Netzwerklast). Hierzu wurden unterschiedliche Client- und Serverkonfigurationen erzeugt, um die Nachrichtengröße zu beeinflussen. Anschließend erfolgte die Ausführung des NTS-Dienstes, in dem der vollständige Kommunikationsablauf stattfand. Die nachfolgenden Ergebnisse zeigen die Datengrößen, die NTS als Nutzlast im *Value*-Feld der entsprechenden NTP-Erweiterungsfelder (s.a. Kapitel 3.4.1) einbettet.

5.1.1 Minimale Nachrichtengrößen in NTS

Die erste Testkonfiguration verfolgt das Ziel, möglichst kleine Nachrichtengrößen zu generieren. Als weitere Bedingung gilt allerdings, dass die derzeitigen Sicherheitsstandards erfüllt sind. Kryptografisch unsichere Algorithmen sind somit vom Test ausgeschlossen. Die verwendeten Parameter sind in diesem Aufbau bei Client und Server identisch und sehen wie folgt aus:

- unterstützte NTS-Versionen: 1
- unterstützte Hash-/MAC-Algorithmen: sha256
- unterstützte asymmetrische Algorithmen: brainpoolP256r1
- unterstützte symmetrische Algorithmen: aes-256-cbc

- Zertifikatseigenschaften
  - öffentlicher Schlüssel: ECC, brainpoolP256r1
  - verwendeter Signaturalgorithmus: ecdsa-with-SHA256
  - zwischengeschaltete CA-Zertifikate: keine

Neben dem Server nutzt auch der Client ein ausgestelltes Zertifikat. Ein selbsterzeugtes Clientzertifikat würde jedoch keine oder nur eine minimale Änderung der Nachrichtengröße nach sich ziehen. Des Weiteren verwenden alle CMS-Strukturen[62] den *Subjekt Key Identifier*, um die Inhalte den entsprechenden Empfänger zuordnen zu können. Die Verwendung der SKIs verringert im Vergleich zum Zertifikatnamen und dessen Seriennummer die Nachrichtengröße. Der Größenunterschied ist insbesondere von der Anzahl der enthaltenen Zertifikate abhängig und liegt bei durchschnittlich 100-150 Bytes pro eingebettetem Zertifikat.

Tabelle 5.1: Minimale Nachrichtengrößen in NTS

generierte NTS-Nachricht	Payload-Größe NTP-Erweiterungsfeld 1 [NTS-Nachricht]	Payload-Größe NTP-Erweiterungsfeld 2 [NTS-MAC]
client_access	21 Bytes	0 Bytes
server_access	41 Bytes	0 Bytes
client_assoc	107 Bytes	0 Bytes
server_assoc	1056 Bytes	0 Bytes
client_cook	736 Bytes	0 Bytes
server_cook	635 Bytes	0 Bytes
time_request	72 Bytes	41 Bytes
time_response	41 Bytes	41 Bytes

Wie die aufgezeichneten Messergebnisse in Tabelle 5.1 zeigen, fallen besonders die Nachrichten in der Verhandlungsphase groß aus. Dies liegt hauptsächlich an den eingebetteten Zertifikaten, die in dieser Messung eine Binärgröße von etwa 630 Bytes aufweisen. Auch die enthaltenen Signaturen und verschlüsselten Inhalte tragen signifikant zur Nachrichtengröße bei. Da alle Nachrichten bei dieser Testkonstellation unter 1424 Bytes liegen (s.a. Kapitel 3.6.7), würde es außerdem zu keiner IP-Fragmentierung kommen. Hier sei allerdings noch erwähnt, dass die Einhaltung der Nachrichtengröße auch durch die Root-CA beeinflusst wird. Je nach angewandtem Signaturalgorithmus kann sich die Größe des ausgestellten Zertifikats mehr als verdoppeln und somit doch eine IP-Fragmentierung provozieren (s.a. Kapitel 6.1.2).

[62] An dieser Stelle sind die beiden Strukturen *SignedData* und *EnvelopedData* gemeint.

5.1.2 Maximale Nachrichtengrößen in NTS

Diese Testkonfiguration setzt sich das Ziel, überdurchschnittlich große NTS-Nachrichten zu erzeugen. Eine explizite Größenbeschränkung existiert auf Basis der NTS-Spezifikation zwar nicht, jedoch ist eine Limitierung durch darunterliegenden Netzwerkprotokolle (wie UDP) von ca. 64 Kilobyte gegeben. Das Ziel dieser Testkonfiguration ist nicht, diese Größe auszureizen, sondern praxistaugliche Nachrichten zu erzeugen, die aufgrund ihrer Konfiguration die zu erwartende Durchschnittsgröße übersteigen. Die verwendeten Parameter sind in diesem Aufbau bei Client und Server unterschiedlich, verwenden allerdings die gleiche Anzahl an unterstützten Algorithmen und nutzen ähnliche Konfigurationen für die Zertifikate. Der Aufbau sieht wie folgt aus:

- Anzahl unterstützter NTS-Versionen: 6
- Anzahl unterstützter Hash-/MAC-Algorithmen: 4
- Anzahl unterstützter asymmetrische Algorithmen: 6
- Anzahl unterstützter symmetrische Algorithmen: 4
- Zertifikatseigenschaften
  - öffentlicher Schlüssel: RSA, 4096 Bit
  - verwendeter Signaturalgorithmus: sha512WithRSAEncryption
  - zwischengeschaltete CA-Zertifikate: 3

Auch bei dieser Konstellation nutzen Client und Server ausgestellte Zertifikate und verwenden in den CMS-Strukturen die SKIs zur Identifikation der Inhalte. Die durch den Aufbau bedingten Änderungen sind in der folgenden Tabelle 5.2 einsehbar. Hierbei ist ersichtlich, dass die generierten NTS-Nachrichten im Vergleich zu Tabelle 5.1 um ein Vielfaches größer geworden sind. Die Zugriffs- und Zeitnachrichten sind von diesen Änderungen jedoch nicht betroffen und weisen nach wie vor die identische Größe auf.

Tabelle 5.2: Maximale Nachrichtengrößen in NTS (sinnvoller Anwendungsfall)

generierte NTS-Nachricht	Payload-Größe NTP-Erweiterungsfeld 1 [NTS-Nachricht]	Payload-Größe NTP-Erweiterungsfeld 2 [NTS-MAC]
client_access	21 Bytes	0 Bytes
server_access	41 Bytes	0 Bytes
client_assoc	221 Bytes	0 Bytes
server_assoc	7076 Bytes	0 Bytes
client_cook	6164 Bytes	0 Bytes
server_cook	1492 Bytes	0 Bytes
time_request	72 Bytes	41 Bytes
time_response	41 Bytes	41 Bytes

5.2 Messung der benötigten Ressourcen

Ein weiterer wichtiger Test ist die Messung der benötigen Ressourcen, die der NTS-Dienst zur Programmlaufzeit benötigt. Vertreter dieser Gruppe sind insbesondere die verfügbare Rechenleistung und der Arbeitsspeicher. Die benötigte Rechenleistung wurde im Rahmen dieser Tests noch nicht ermittelt. Ein aussagefähiges Ergebnis erfordert umfangreiche Tests auf verschiedenen Hardware-Plattformen, unter Verwendung unterschiedlicher kryptografischer Algorithmen und mit unterschiedlichen Intensitäten[63]. In der lokalen Testumgebung zeigte der NTS-Dienst allerding nur sehr geringe zusätzliche Rechenlasten.

Die durchgeführten Messungen beziehen sich auf den clientseitigen Speicherbedarf. Da dieser zustandsbehaftet ist und möglicherweise die Kommunikation zu mehreren Servern aufrechterhält, alloziert dieser im Verlauf zusätzlichen Arbeitsspeicher. Das Messergebnis soll zeigen, in welchen Bereich sich dieser Speicherbedarf bewegt und ob der prinzipielle Einsatz von NTS auch auf schwacher Hardware möglich ist. Für diesen Versuch sind dem Client 30 NTS-Server zugeordnet worden, die untereinander jedoch denselben Aufbau (Zertifikate, verwendete Algorithmen) nutzten. Die Zusammensetzung der Zertifikate entsprach in etwa der Konfiguration im Kapitel 5.1.1. Um den maximalen Speicherbedarf zu ermitteln, wurde eine vollständige Kommunikationskette[64] zu allen Servern nacheinander durchgeführt. Um den weiteren Speicherverlauf zu beurteilen, folgte eine Wiederholung der Kommunikation mit allen Servern. Das Ergebnis stellt dazu die Abbildung 5.1 dar.

Wie die Abbildung 5.1 zeigt, liegt der clientseitige Speicherbedarf bei ca. 1,4 MB (grüne Kennlinie). Dieser Wert ist insbesondere von der Anzahl der NTS-Server abhängig, mit denen der Client kommuniziert. Auch die Anzahl der Algorithmen und ausgetauschten Zertifikate haben großen Einfluss auf den tatsächlichen Speicherbedarf. Der Anstieg in der ersten Hälfte der Grafik zeigt die Speicherallokation durch den Client, die er zum Speichern der ausgetauschten Algorithmen und Zertifikate benötigt. In der zweiten Hälfte findet kein weiterer Anstieg statt, da der Client bereits im Besitz aller Parameter des jeweiligen Servers ist. Die temporären Spitzen sind durch die Verarbeitungen der Nachrichten begründet, die zum Zeitpunkt der Messung stattfanden und kurzfristig zu einem erhöhten Speicherbedarf führten. Dies war beispielsweise dann der Fall, wenn der Client durch Verfall des *Server Seed* ein neues *Cookie* anforderte. Die restlichen Kennlinien im Diagramm zeigen die Zusammensetzung des Speicherbedarfs und sind primär durch die Softwarearchitektur begründet. Die x-Achse stellt in diesem Zusammenhang den zeitlichen Verlauf der Messung dar. Die Darstellung ist jedoch nicht in Millisekunden, sondern in Anzahl der durchgeführten Befehle.

[63] Dies bezieht sich besonders auf die Anzahl der serverseitig eintreffenden NTS-Nachrichten, die pro Zeiteinheit zu verarbeiten sind.
[64] Die Verhandlungsphase wurde durchlaufen und es fand mindestens ein erfolgreicher Austausch der NTS gesicherten Zeitinformationen statt.

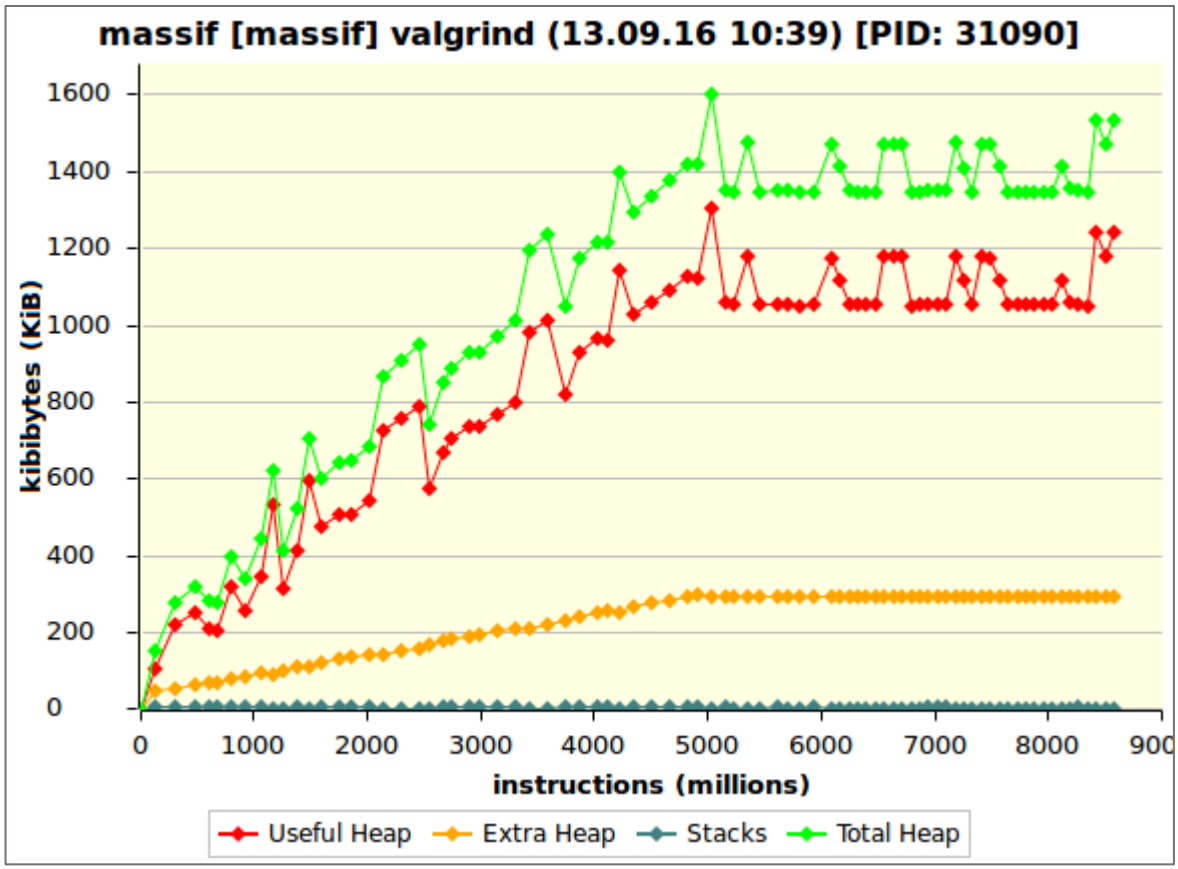


Abbildung 5.1: Arbeitsspeicherbedarf der NTS-Client-Anwendung mit 30 Zeitserver

An dieser Stelle sei noch angemerkt, dass eine serverseitige Messung keine Relevanz hat und daher grafisch nicht erfasst wurde. Im Gegensatz zum Client findet serverseitig kein Zuwachs des Arbeitsspeichers aufgrund der Zustandslosigkeit statt. Der Speicherbedarf ist konstant und liegt bei durchschnittlich 170 KB im Leerlauf bis 400 KB während der Verarbeitung einer empfangenen NTS-Nachricht.

5.3 Messung der Verarbeitungsdauer

Die Ermittlung der durch NTS erzeugten Latenz ist ein wesentliches Maß zur Beurteilung der Effizienz des implementierten NTS-Protokolls. Bei dieser Messung steht die reine Verarbeitungsdauer einzelner NTS-Nachrichten im Fokus der Untersuchung. Die Zeitmessung fand hierbei durch den Einsatz eines Hochleistungszählers[65] (engl. *High-Performance-*

[65] Ein Hochleistungszähler ist eine Hardware-Komponente, die sich unter anderem auf jedem Desktop-PC befindet und mit einer hohen definierten Frequenz eine Ganzzahl inkrementiert. Durch Software-Schnittstellen lassen sich Frequenz und Zählerstände auslesen, wodurch die Zeitdifferenz zweier Messpunkte (Zählerstände) errechnet werden kann.

Counter) statt, wodurch Zeitmessungen im oberen Nanosekundenbereich möglich sind. Die aufgenommenen Messergebnisse werden in der Tabelle 5.3 dargestellt und im Folgenden näher erläutert.

Tabelle 5.3: Durch NTS erzeugte Latenzzeiten

Teilnehmer	durchgeführte Aktion(en)	Verarbeitungszeit [Debug-Modus]	Verarbeitungszeit [Realbedingung]
Client	client_access generieren	3,540 ms	0,767 ms
Server	client_access verarbeiten server_access generieren	2,356 ms	0,112 ms
Client	server_access verarbeiten client_assoc generieren	4,135 ms	0,224 ms
Server	client_assoc verarbeiten server_assoc generieren	21,459 ms	7,334 ms
Client	server_assoc verarbeiten client_cook generieren	7,736 ms	2,285 ms
Server	client_cook verarbeiten server_cook generieren	10,590 ms	8,138 ms
Client	server_cook verarbeiten time_request generieren	4,638 ms	2,540 ms
Server	time_request verarbeiten	0,612 ms	0,058 ms
Server	time_response generieren	1,462 ms	0,082 ms
Client	time_response verarbeiten	1,395 ms	0,097 ms

Die Tabelle 5.3 zeigt die Verarbeitungszeiten von Nachrichten auf Client- und Serverseite und differenziert hierbei zwischen dem Debug-Modus und dem Test unter Realbedingungen. Dabei wurde die vollständige Kommunikationskette durchlaufen, wobei jede Nachricht im NTS-Protokoll mindestens einmal generiert und verarbeitet wurde. Der Debug-Modus beschreibt die Zeitmessung in der virtuellen Maschine, in dem zudem noch diverse Diagnosefunktionen aktiviert waren. Beim Test unter Realbedingungen sind diese Funktionen deaktiviert worden und es erfolgte die Kompilierung des Quellcodes im Release-Modus, was grundlegende Optimierungen des Codes zur Folge hat. Die Ausführung des NTS-Dienstes fand anschließend auf einem mit Linux vorinstallierten Laptop[66] statt.

Die Messungen zeigen erhöhte Werte während der Assoziations- und Cookie-Austauschphase. Dies ist durch die kryptografischen Funktionen (Signatur und Verschlüsselung) und dem größeren Datenaufkommen der Nachrichten zu begründen. Kritisch sind ausschließlich die *time_request*- und *time_response*-Nachrichten, die bei asymmetrischen Verarbeitungszeiten des NTS-MAC den Offset der Nachrichten im NTP-Protokoll negativ beeinflussen

[66] Laptop-Konfiguration: Linux Debian 8.5.0 i386, CPU: Intel i5-6200U (2,3 GHz), RAM: 8 GB

(s.a. Kapitel 3.6.8.2). Die Generierungs- und Verarbeitungszeiten liegen hier unter 100 Mikrosekunden, in denen neben der Erzeugung NTS-Nachricht auch die Berechnung des NTS-MAC einfließen. Angestrebt wird hierbei eine Zeit von unter 10 Mikrosekunden für die Generierung des NTS-MAC. Die aufgeführten Messwerte sind eher als grobe Ergebnisse anzusehen, da diese durch unterschiedliche Einflüsse zusätzlich verfälscht wurden. Insbesondere die Konsolenausgaben und die Zeitmessungen selbst erhöhen die gemessenen Werte deutlich. Eine genauere Messung kann möglicherweise durch externe Software oder Hardware erfolgen.

5.4 Test unter realen Bedingungen über NTPv4

Die bisherigen Tests wurden zumeist lokal in der virtuellen Maschine durchgeführt, wobei NTS direkt oder unter Zuhilfenahme eines NTP-Dummys[67] Gegenstand der Messung war. Der Test unter realen Bedingungen sieht eine Konfiguration vor, in der der NTS-Dienst in eine NTP-Implementierung eingebettet und verwendet wird. Die Prüflinge stellen Computer dar, die über das Internet miteinander verbunden sind und idealerweise unterschiedliche Hardwarearchitekturen verwenden. Jedem Computer wird außerdem eine Rolle als Client oder Server zugewiesen. Anschließend sollen alle Clients eine NTS-gesicherte Zeitsynchronisation durch einen oder mehrere NTS-Zeitservern durchführen.

Die durch eine externe Entwicklergruppe umgesetzte NTP-Implementierung befindet sich derzeit noch in Entwicklung, weshalb dieser Test noch nicht durchgeführt werden konnte.

[67] Das NTP-Dummy ist eine implementierte C++-Klasse, die NTPv4-Nachrichten bereitstellt und simuliert.

6 Weiterführende Entwicklungen

Die NTS-Spezifikation befindet sich derzeit noch in der Entwurfsphase. Somit sind diverse Änderungen zu erwarten bevor NTS den RFC-Status erlangt. Diese Änderungen sind natürlich auch in der entsprechenden Implementierung nachzuziehen, um mit der Spezifikation konform zu bleiben. Die folgenden Abschnitte beschreiben hierzu die derzeit zu lösenden Herausforderungen, die bevorstehenden Entwicklungen und auch die zusätzlichen Erweiterungen, die in der Implementierung zukünftig umsetzbar wären.

6.1 Herausforderungen der NTS-Spezifikation

Dieser Abschnitt beschreibt einige der Probleme, die derzeit zur Diskussion stehen und entsprechend zu behandeln sind. Hieran ist unter anderem das *Network Time Security WGLC*[68] *Design Team* beteiligt, in dem die Autoren der NTS-Spezifikation und einige NTP-Entwickler zusammen an einer Lösung arbeiten. Wann entsprechende Lösungen bereitstehen und in welchem Ausmaß dies an Änderungen der NTS-Spezifikation nach sich zieht, ist derzeit nicht vorhersagbar. An dieser Stelle werden hierzu die beiden Punkte thematisiert, die aktuell die höchste Priorität besitzen. Die vollständige Liste der Themen ist hierzu auf der IETF-Webseite [122] [123] zu finden.

6.1.1 Anpassung der Verhandlungsphase (Key Exchange)

Ein wichtiger, jedoch unkritischer Punkt ist die mögliche Änderung der Verhandlungsphase. Derzeit enthält diese Zugriffs-, Assoziations- und Cookie-Nachrichten, um den Austausch der benötigten Informationen zu realisieren. Die aktuelle Diskussion sieht eine mögliche Vereinfachung vor, in der die Assoziations- und Cookie-Nachrichten zusammengefasst werden. Dies bietet den Vorteil, dass der Verarbeitungsaufwand der Verhandlungsphase sinkt und dies zusätzlich die Netzwerkbelastung verringert. Der NTS-Protokollverlauf würde sich somit um ein Nachrichtenpaar verkürzen. Dem stehen jedoch der zusätzliche Änderungsaufwand, die resultierende Nachrichtengröße und zusätzliche Hürden entgegen. Die Schwierigkeit, die hierdurch entsteht, soll durch das folgende Beispiel verdeutlicht werden. Hierzu wurden die Assoziations- und Cookie-Nachrichten zu einem fiktiven Nachrichtenpaar zusammengefasst, die hier als *cookie_request*- und *cookie_response*-Nachricht definiert werden. Der Client würde somit nach der Zugriffsphase eine *cookie_request*-Nachricht erzeugen, die ein *CookieRequestData*-Nachrichtenobjekt beinhaltet und gemäß dem *NTS-Plain-*

[68] WGLC (Working Group Last Call)

Archetyp strukturiert ist. Der Server reagiert entsprechend mit einer *cookie_response*-Nachricht, die ein *CookieResponseData*-Nachrichtenobjekt enthält und nach dem *NTS-Encrypted-and-Signed*-Archetyp strukturiert ist. Die genannten Nachrichtenobjekte würden dabei folgenden Aufbau haben:

Tabelle 6.1: ASN.1-Beschreibung der fiktiven NTS-Struktur "CookieRequestData"

```
CookieRequestData ::= SEQUENCE {
      accessKey               NTSAccessKey,
      nonce                   NTSNonce,
      minVersion              NTSVersion,
      macAlgos                AlgorithmIdentifiers,
      keyEncAlgos             KeyEncryptionAlgorithmIdentifiers,
      contentEncAlgos         ContentEncryptionAlgorithmIdentifiers,
      certificates            CertificateSet
}
```

Tabelle 6.2: ASN.1-Beschreibung der fiktiven NTS-Struktur "CookieResponseData"

```
CookieResponseData ::= SEQUENCE {
      nonce                   NTSNonce,
      proposedVersion         NTSVersion,
      macAlgos                AlgorithmIdentifiers,
      choiceMacAlgo           AlgorithmIdentifier,
      keyEncAlgos             KeyEncryptionAlgorithmIdentifiers,
      contentEncAlgos         ContentEncryptionAlgorithmIdentifers,
      cookie                  OCTET STRING (SIZE(16))
}
```

Die Inhalte der Nachrichtenobjekte orientieren sich an den ausgetauschten Daten der bestehenden Assoziations- und Cookie-Nachrichten. Fügt man die jeweiligen Informationen wie dargestellt zusammen, so entstehen unter anderem folgende Konflikte:

Die Aushandlung der NTS-Version

Eine Aushandlung wie in den Assoziationsnachrichten ist bei dieser Konstellation nicht möglich bzw. nicht sinnvoll. Die ausgehandelte NTS-Version wäre somit nur in den Zeitsynchronisationsnachrichten anwendbar, aber nicht während der Verhandlungsphase. Eine Auslagerung in die Zugriffsnachrichten ist ebenso wenig möglich, da diese nicht durch eine digitale Signatur geschützt sind und somit angreifbar wären (sogenannter *Downgrade*-Angriff).

Die clientseitige Zertifikatgenerierung

Ein weiteres Problem stellt die clientseitige Zertifikatgenerierung bzw. Zertifikatauswahl dar. Dieser generiert normalerweise sein Zertifikat auf Basis des ausgehandelten *keyEncryption*-Algorithmus. Dies stellt den asymmetrischen Verschlüsselungsalgorithmus dar, der in der Serverantwort für die CMS-Struktur benötigt wird. Da allerdings zu diesem Zeitpunkt der Algorithmus schon im Clientzertifikat enthalten sein muss, ist eine Aushandlung des entsprechenden Algorithmus zwecklos.

Erhöhter Aufwand des Servers bei Seed-Aktualisierung

Ein weiterer Punkt ist die Erhöhung der Serverlast bei einer Server-Seed-Aktualisierung. Dieses führt clientseitig zu einer erneuten Anfrage an den Server, welcher nun mehr Daten zu prüfen und zu verarbeiten hat, bis eine entsprechende Antwort generiert und gesendet wurde. Da nach einer solchen Seed-Aktualisierung sehr viele Clients nahezu gleichzeitig neue Anfragen stellen, können dadurch kurzfristig ungewollt hohe Serverlasten entstehen.

Erhöhung der Nachrichtengröße

Kritisch ist zudem die Erhöhung der Nachrichtengröße, die durch den Zusammenschluss der Nachrichten begünstigt wird. Dies führt zu einem höheren Ressourcenbedarf bei der Bearbeitung der Nachrichten und fördert darüber hinaus die IP-Fragmentierung, welches ein schwerwiegendes Problem darstellt.

6.1.2 Probleme mit Filterung und IP-Fragmentierung

Die Filterung und IP-Fragmentierung [124] der NTP-Nachrichten stellen derzeit die größten Probleme dar, die es zu lösen gilt. Bei der Filterung handelt es sich um einen gezielten Verwurf von NTP-Nachrichten, die jedoch gültig sind und möglicherweise eine NTS-Nachricht transportieren. Dieses Problem ist allerdings nirgends offiziell dokumentiert und wurde vom *Network Time Security WGLC Design Team* festgestellt. Hierbei filtern einige Netzwerkkomponenten NTP-Nachrichten aus, die häufiger als einmal pro Sekunde oder häufiger als sechsmal pro Minute an einen Empfänger übertragen werden[69]. Darüber hinaus betrifft es auch NTP-Nachrichten, welche die durchschnittliche Größe der NTPv4-Nachricht von 48

[69] Diese Aussage nicht derzeit nicht offiziell bestätigt und könnte zukünftig widerrufen werden.

Bytes signifikant überschreiten. Dies bietet zwar einen möglichen Schutz vor DoS-Angriffen, unterbindet jedoch den NTS-Dienst komplett. Eine weitere Filterung scheint es auch bei der NTP-Referenzimplementierung NTPD (*Network Time Protocol daemon*) zu geben, die Nachrichtengrößen über 1 KB verwirft.

Im Falle von NTS kommt es bei zu großen Nachrichten außerdem zur IP-Fragmentierung der UDP-Pakete, die somit in mehrere Datenframes aufgeteilt und separat übertragen werden. Die eigentliche Fragmentierung erfolgt hierbei in der Vermittlungsschicht (Schicht 3) des OSI-Referenzmodells durch das Internet-Protokoll (in der Regel IPv4). Die einzelnen Fragmente werden anschließend beim Empfänger durch das Internet-Protokoll zusammengesetzt und als UDP-Paket an die Transportschicht übergeben. Bei diesem Verfahren können allerdings verschiedene Effekte auftreten, was die Kommunikation negativ beeinträchtigt. So können einzelne Fragmente verloren gehen, mehrmals empfangen werden oder aufgrund von Mehrwegeausbreitung in einer falschen Reihenfolge beim Empfänger eintreffen. In Folge dessen können übertragene UDP-Pakete ungültig sein, wodurch die Transportschicht derartige Nachrichten verwirft.

Um eine Fragmentierung zu verhindern, dürfen NTS-Nachrichten eine Größe von 1424 Bytes bei IPv4 nicht überschreiten. Die Herleitung dieser Größe und mögliche Abweichungen werden dazu im Kapitel 3.6.7 beschrieben. Die Unterschreitung der NTS-Nachrichtengröße von 1424 Bytes ist prinzipiell in allen Nachrichten möglich, wie die durchgeführten Tests im Kapitel 5 (siehe dazu Tabelle 5.1) zeigen. Dies ist allerdings von mehreren Faktoren abhängig, die nicht alle beeinflusst werden können. Insbesondere stellen die X.509-Zertifikate den Knackpunkt der NTS-Kommunikation dar, da diese ausgetauscht werden müssen und den Großteil der jeweiligen Nachrichten ausmachen. Die durchschnittliche Zertifikatgröße liegt hier im Bereich von 630 bis 2000 Bytes und ist insbesondere vom enthaltenen öffentlichen Schlüssel und dessen Signatur abhängig. Eine Lösung, in der NTS selber die Fragmentierung von Nachrichten regelt, ist ungünstig, da dies verschiedene Probleme mit sich bringt. So muss serverseitig ein Zustand gehalten werden und es müssen Regelungen für Time-Outs, Sendeintervalle und Verbindungsabbrüche ausgearbeitet und geprüft werden.

Eine mögliche Lösung stellt die Auslagerung der NTS-Verhandlungsphase über DTLS dar, welcher als separater Kommunikationskanal fungiert. DTLS basiert auf TLS, nutzt allerdings das UDP-Transportprotokoll zur Datenübertragung. Dieses bietet den Vorteil, dass es eine interne Segmentierungsverwaltung beinhaltet. Große Nachrichten werden somit automatisch fragmentiert gesendet und beim Empfänger korrekt zusammengesetzt. Dies garantiert die korrekte Übertragung großer Datagramme. Allerdings stellt sich die Frage, wie weit die aktuelle Verhandlungsphase bei dieser Lösung notwendig ist. DTLS baut seinerseits eine sichere Verbindung auf, worüber das vom NTS-Client benötigte Cookie übertragen werden kann. Der von NTS spezifizierte *Key Exchange* wäre somit überflüssig. Der zweite Punkt ist

die Tatsache, dass für eine NTS-gesicherte NTP-Nachricht immer ein zweiter Datenkanal geöffnet werden muss, um zunächst das benötigte Cookie zu übertragen. Welche Lösung sich zukünftig durchsetzt, bleibt abzuwarten.

6.2 Weitere Änderungen der NTS-Spezifikation

Neben den geplanten Änderungen, die auf den derzeitigen Herausforderungen basieren, gibt es noch weitere Baustellen, die aufgrund von Aktualisierungen der NTS-Spezifikation vorhanden sind. Einige wurden bereits kommuniziert und werden in den folgenden Änderungen berücksichtigt. In diesem Abschnitt werden diese kleineren Konflikte thematisiert, die teilweise noch nicht in der NTS-Implementierung übernommen werden konnten.

6.2.1 Probleme durch den Ersatz von Hash- durch MAC-Algorithmen

Hash-Algorithmen spielen in der NTS-Spezifikation und in der zugehörigen Implementierung eine besondere Rolle. In den vorherigen NTS-Dokumenten (*draft-ietf-ntp-network-time-security-13*, *draft-ietf-ntp-using-nts-for-ntp-04* und *draft-ietf-ntp-cms-for-nts-message-06*) wurden noch explizit Hash-Algorithmen zwischen dem Client und dem Server ausgetauscht. Dies erfolgte über die NTS-Nachrichtenobjekte der Assoziationsnachrichten durch dem *hmacHashAlgos*-Eintrag. Der ausgehandelte Hash-Algorithmus wurde anschließend für die Bereitstellung unterschiedlicher Server Seeds verwendet, ermöglichte die Berechnung des KIV, kam als HMAC-Parameter zur Anwendung und war Bestandteil digitaler Signaturen. In den aktuellen NTS-Dokumenten wurde der bisher genutzte HMAC-Algorithmus durch einen allgemeinen MAC-Algorithmus ersetzt. Der Grund hierfür ist die Erhöhung der Flexibilität von NTS, der nunmehr den MAC-Algorithmus frei aushandeln kann und somit nicht mehr an den HMAC-Algorithmus gebunden ist. Der entsprechende *hmacHashAlgos*-Eintrag aus den Assoziationsnachrichten wurde in diesem Schritt zu *macAlgos* umbenannt und beinhaltet jetzt MAC-Algorithmen statt Hash-Algorithmen. Durch die freie Aushandlung ist die Verwendung effizienterer MAC-Algorithmen möglich, deren Vergleich im Kapitel 3.6.3 einsehbar ist. Allerdings bringt die Umstellung auch Probleme mit sich, die derzeit in der NTS-Spezifikation noch nicht berücksichtigt wurden:

1. Die Bereitstellung der *Server Seeds*

Der erste Punkt ist die Bereitstellung verschiedener Server Seeds, die eigentlich auf den unterschiedlichen Hash-Algorithmen basieren. Dieser Umstand lässt sich jedoch leicht beheben, in dem Server Seeds auf Basis von MAC-Algorithmen gebildet werden. Der Grund für die Generierung verschiedener Server Seeds dokumentiert das Kapitel 3.6.2.

2. Die Berechnung des *Key Input Value*

Ein weiterer Punkt ist die Generierung des KIV, der sich gemäß der aktuellen NTS-Spezifikation entsprechend der Gleichung 6.1 berechnet:

$$KIV = MSB_{128}\big(H(Zertifikat\ des\ Clients)\big) \tag{6.1}$$

Da der Hash-Algorithmus allerdings nicht mehr zur Verfügung steht, ist eine alternative Berechnung notwendig. In diesem Falle ist der Einsatz eines MAC-Algorithmus denkbar, da sich der daraus berechnete KIV ebenfalls als Eingabewert zur Generierung des Cookies eignet. Die Gleichung 6.2 zeigt die dazu notwendige Änderung:

$$KIV = MSB_{128}\big(MAC(Zertifikat\ des\ Clients)\big) \tag{6.2}$$

3. Die Berechnung des *Access Key*

Der *Access Key* ist von den Änderungen indirekt betroffen, da auch hier der HMAC-Algorithmus durch den verallgemeinerten MAC-Algorithmus ersetzt wurde. Allerdings wird dieser Algorithmus nicht zwischen Client und Server ausgehandelt, wodurch der Server einen geeigneten Algorithmus festlegen kann. Die aktuelle Implementierung verwendet allerdings noch den HMAC-Algorithmus mit einem konfigurierbaren Hash-Algorithmus.

4. Die Generierung digitaler Signaturen

Der schwierigste Punkt ist die digitale Signatur, die der Client bei der Generierung seines Zertifikats und der Server bei der Erstellung der CMS-Strukturen benötigt. Hier lassen sich die Algorithmen nicht einfach durch einen MAC-Algorithmus ersetzen, wodurch die Aushandlung eines Hash-Algorithmus weiterhin nötig wäre. Eine mögliche Lösung ist die Verwendung des Signaturalgorithmus aus dem Serverzertifikat. Dieser wird somit vom Server zur Signierung von CMS-Strukturen verwendet und der Client kann den Algorithmus als

Basis zur Erstellung seines eigenen Zertifikats nutzen. Hierbei muss man allerdings voraussetzen, dass der Client den Signaturalgorithmus des Servers unterstützt. Die Übertragung des Signaturalgorithmus vom Server zum Client erfolgt somit über das Serverzertifikat und wird nicht ausgehandelt.

6.2.2 Die Wahl des *keyEncryption*-Algorithmus

Die Aushandlung des *keyEncryption*-Algorithmus ist derzeit nur indirekt möglich, wie die aktuelle NTS-Implementierung zeigt. Client und Server sollen hierzu einen asymmetrischen Algorithmus aushandeln, der zur Verschlüsselung eines symmetrischen Schlüssels in der CMS-Struktur zum Einsatz kommt. Den dafür benötigten öffentlichen Schlüssel bezieht der Server direkt aus dem Clientzertifikat. Auch der angewendete Verschlüsselungsalgorithmus wird aus dem Zertifikat des Clients entnommen und nicht aus dem NTS-Nachrichtenobjekt, in dem sich der ausgehandelte *keyEncryption*-Algorithmus befindet. Das hat folgende Bewandtnis: Befindet sich im Zertifikat ein öffentlicher Schlüssel, der beispielsweise durch einen RSA-Algorithmus erzeugt wurde, so muss bei der Verschlüsselung von Daten auch ein RSA-Algorithmus zum Einsatz kommen. Kombinationen wie ECC-Verschlüsselung mit einem RSA-Schlüssel sind nicht möglich. Aus diesem Grund muss der *keyEncryption*-Eintrag in den NTS-Nachrichtenobjekten immer den asymmetrischen Algorithmus des Clientzertifikats nutzen. Eine Aushandlung ist jedoch möglich, indem sich Client und Server auf einen Algorithmus einigen und der Client auf dieser Basis sein Zertifikat generiert. Wird jedoch vom Server eine Clientautorisierung verlangt, so müsste der Client verschiedene ausgestellte Zertifikate und Schlüssel mit unterschiedlichen Algorithmen verwalten und den korrekten auswählen. Eine derartige Funktion ist allerdings noch nicht implementiert.

6.2.3 Der verwendete Signaturalgorithmus

Der letzte Konflikt ist der Signaturalgorithmus, der sich in der *client_cook*-Nachricht befindet. Dieser wird während der Verhandlungsphase nicht ausgehandelt, wodurch der Inhalt dieses Eintrags nicht festgelegt ist und daher verworfen wird. Eine Definition zu diesem Eintrag ist in der NTS-Spezifikation derzeit nicht vorhanden. Aus diesem Grund sollte der entsprechende Eintrag im NTS-Nachrichtenobjekt ignoriert werden oder den Signaturalgorithmus des Serverzertifikats beinhalten.

6.3 Weiterentwicklung der NTS-Implementierung

Dieser Abschnitt befasst sich mit der NTS-Implementierung in Bezug auf deren Test, Optimierung und Erweiterung. Die Implementierung ist schon weit fortgeschritten, aber noch nicht für den Einsatz in der Praxis geeignet. Dafür muss zunächst einmal die NTS-Spezifikation abgeschlossen sein, um eine solide Grundlage zu haben. Nur so kann gewährleistet werden, dass später unterschiedliche NTS-Implementierungen miteinander konform sind. Im Folgenden werden die einzelnen Punkte beschrieben, die für den Abschluss der Implementierung notwendig sind oder eine sinnvolle Ergänzung darstellen.

6.3.1 Ausgiebige Tests mit geeigneten Testfällen

Da es sich bei NTS um einen sicherheitskritischen Dienst handelt, ist eine sorgfältige Prüfung unerlässlich. Dies ist über geeignete Testfälle zu realisieren, die für diesen Zweck noch zu entwerfen sind. Die grundlegende Stabilität der aktuellen Implementierung wurde durch diverse Modul- und Komponententests überprüft, sie sind allerdings bei weitem nicht ausreichend, um die korrekte Funktion der Software in der Praxis zu garantieren. Neben der Stabilität sind auch weitere Faktoren zu prüfen. So sollte im Zuge einer Codeoptimierung auch ein Codeüberdeckungstest erfolgen, um nicht erreichbare Codeabschnitte zu entfernen oder mögliche Verarbeitungsfehler ausfindig zu machen. Darüber hinaus ist auch die Effizienz der Implementierung zu analysieren, da diese Ungenauigkeiten während der Zeitsynchronisation hervorrufen kann. Diese Analyse kann unter Zuhilfenahme verschiedener *Profiler* erfolgen, um den programminternen Ablauf besser verfolgen zu können. Dies sollte zudem mit unterschiedlichen Algorithmen und auf unterschiedlichen Hardwareplattformen getestet werden, um aussagekräftige Ergebnisse zu erzielen. Ein weiterer sinnvoller Test ist die Kommunikation mit einer anderen NTS-Implementierung, sofern diese vorliegt. Die anschließend entdeckten Fehler würden dann mögliche Interpretationsfreiheiten oder Lücken der NTS-Spezifikation aufzeigen.

6.3.2 Optimierung der NTS-Implementierung

Die Codeoptimierung ist ein weiterer wichtiger Punkt, der früher oder später erfolgen muss. Diese hat das Ziel, die Lesbarkeit und Wartungsfreundlichkeit des Quellcodes zu verbessern und soll zudem die Kompatibilität erhöhen. Es ist wünschenswert, dass die Implementierung auf verschiedenen Hard- und Softwareplattformen läuft und somit ihren Anwendungsbereich vergrößert. Im Folgenden werden hierzu sinnvolle Optimierungen geschildert, die derzeit noch offen sind.

Multi-Thread-Unterstützung

Eine sinnvolle Optimierung stellt die Multi-Thread-Unterstützung dar. Diese ermöglicht die simultane Ausführung von mehreren NTS-Instanzen, ohne die Gefährdung von Speicherzugriffsverletzungen. Dies ist für Anwendungsgebiete interessant, in denen ein Teilnehmer sowohl als Client, als auch als Server fungieren möchte. Die korrekte Verwaltung der Ressourcen von simultan arbeitenden NTS-Instanzen, ist derzeit noch nicht getestet worden. Wie weit dies auch mit NTP-Implementierungen vereinbar ist, ist zudem noch zu prüfen.

Verbesserung der Zertifikatklassifizierung

Wie Tabelle 4.1 in Kapitel 4.2.2 bereits zeigte, ist die korrekte Erkennung der Zertifikattypen nicht in allen Fällen gewährleistet. Je nach Zertifikatserweiterung und enthaltenen Zertifikatseigenschaften muss jedoch eine korrekte Klassifizierung erfolgen. Eine mögliche Korrektur stellt die Erweiterung der aktuellen Klassen dar. Derzeit sind hierzu drei Klassen definiert (*primäre Zertifikate*, *zwischengeschaltete CA-Zertifikate*, *Root-CA-Zertifikate*), die durch die neue Klasse *selbstgeneriertes Zertifikat* vervollständigt werden kann. Eine genaue Definition zur Erkennung dieser Klasse existiert jedoch noch nicht.

RAII und Codeoptimierung

Ein weiterer Schritt ist die Anwendung des RAII-Idioms (*Resource Acquisition Is Initialization)* auf alle Module der bisherigen NTS-Implementierung. Hierbei geht es um die automatische Freigabe verwendeter Ressourcen und die Sicherstellung des Programmzustandes bei einem unerwarteten Fehler. Aufgrund der strukturellen Entwicklung der Implementierung und der begrenzten Entwicklungszeit konnte dieses Idiom noch nicht auf das ASN.1-Modul angewendet werden. Die restliche Implementierung verfolgt jedoch schon diesen Ansatz. Neben RAII ist zudem eine mögliche Optimierung der Überprüfungsroutinen vorzunehmen, sofern redundante Bereiche existieren und gefahrlos entfernt werden können.

6.3.3 Funktionale Erweiterungen

Abschließend werden an dieser Stelle noch funktionale Erweiterungen oder sinnvolle Anpassungen vorgestellt, die in der zukünftigen NTS-Implementierung zur Anwendung kommen sollten.

Alternativimplementierung des ASN.1-Moduls

Die derzeitige Implementierung verwendet die asn1c-Bibliothek zur Generierung und Verarbeitung von ASN.1-Nachrichten. Möglicherweise bietet eine Neuimplementierung auf Basis von OpenSSL eine bessere Performance, wodurch die NTS-internen Rechenzeiten verringert werden können. Wie weit dieses jedoch zutrifft, ist durch nähere Analysen und Tests in Erfahrung zu bringen.

Automatische Wahl von Schlüsseln und ausgestellten Zertifikaten

Eine sinnvolle Erweiterung ist die Verwaltung von mehreren Zertifikaten und den dazugehörigen Schlüsseln. Dies kommt immer dann zum Einsatz, wenn die Aushandlung der Algorithmen zwischen Client und Server vollzogen ist und der Server eine Autorisierung vom Client verlangt. In diesem Falle muss der Client auf ausgestellte Zertifikate zurückgreifen und das passende Zertifikat wählen, welches den ausgehandelten asymmetrischen Verschlüsselungsalgorithmus beinhaltet. Derzeit kann ein Client immer nur ein ausgestelltes Zertifikat nutzen und ist somit an genau einen Verschlüsselungsalgorithmus gebunden. Verlangt der Server hingegen keine Autorisierung, so können die entsprechenden Algorithmen frei gewählt werden.

Automatische Filterung von Zertifikaten zur Generierung von Zertifikatsketten

Im Zuge der Zertifikatsverwaltung ist auch eine automatisierte Generierung der Zertifikatsketten sinnvoll. Hierbei beinhaltet der Client einen Zertifikatsstapel mit zwischengeschalteten CA-Zertifikaten. Wird vom Client während der Verhandlungsphase ein ausgestelltes Zertifikat ausgewählt, so werden aus dem Stapel die entsprechenden Zertifikate zum Aufbau der Zertifikatskette herangezogen. Derzeit muss ein ausgestelltes Zertifikat – das in Form einer Datei vorliegt – auch die zwischengeschalteten CA-Zertifikate beinhalten, damit der Aufbau einer Zertifikatskette möglich ist.

Unterstützung von Zertifikatssperrlisten (CRLs)

In der NTS-Spezifikation als optionale Eigenschaft aufgeführt, jedoch derzeit nicht implementiert, ist die Verwendung von Zertifikatssperrlisten zur Widerrufung von verteilten Zertifikaten. Wie weit diese Funktion sinnvoll zur Anwendung kommen sollte, ist jedoch fraglich, da die Nachrichtengrößen in NTS entsprechend wachsen würden. Ein Nachrüsten der aktuellen Implementierung mit dieser Eigenschaft sollte allerdings problemlos möglich sein, da diverse Grundfunktionen von OpenSSL bereitstehen.

Onlineprüfung von Zertifikaten via OCSP

Neben der Offlineprüfung von Zertifikaten über den Aufbau von Zertifikatsketten, ist auch die Onlineprüfung via OCSP möglich. Dieses Protokoll würde eine Prüfung der ausgetauschten Zertifikate ermöglichen, auch wenn die Offlineprüfung aufgrund fehlender CA-Zertifikate nicht möglich wäre. Die Nachrüstung dieser Funktion ist mit geringem Implementierungsaufwand möglich und basiert auf Funktionen der OpenSSL-Bibliothek.

Speichern und Wiederherstellen von Sitzungen

Eine weitere sinnvolle Funktionserweiterung wäre das Speichern und Wiederherstellen von Sitzungen, die der Client zur Programmlaufzeit erstellt und verwaltet. Hierbei wird periodisch oder beim Programmende der Sitzungsspeicher als Datei gespeichert und beim Neustart des NTS-Dienstes geladen. Dies würde einen *Warmstart*[70] ermöglichen, durch den der Client ohne Verhandlungsphase direkt Zeitnachrichten anfordern kann. Im Falle einer Realisierung dieser Funktion sollten auch Mechanismen zum Entfernen veralteter Sitzungen hinzugefügt werden. Andernfalls könnten Sitzungsleichen entstehen, die somit den Ressourcenbedarf des NTS-Dienstes erhöhen.

Unterstützung von DNS-Namen

Die derzeitige Identifizierung der Teilnehmer erfolgt im ersten Schritt über deren IP-Adresse. Im Falle von gespeicherten Sitzungen könnten allerdings Probleme auftreten, bei denen ein Zeitserver seine IP-Adresse ändert und somit der Client nicht mehr auf die existierende Sitzung referenzieren kann. Abhilfe schafft hierbei die Verwendung von DNS-Namen (z.B. *ptbtime1.ptb.de*), bei denen unabhängig von der IP-Adresse immer die korrekte Identifizierung möglich ist.

Implementierung des Broadcast-Betriebsmodus

Ein weiterer großer Schritt ist die Implementierung des Broadcast-Modus. Dieser steht in der aktuellen NTS-Implementierung noch nicht zur Verfügung, ist allerdings in der jeweiligen Spezifikation schon definiert.

[70] Bei einem Warmstart springt der Client nach dem Laden und Prüfen des Sitzungsspeichers direkt zum gespeicherten Zustand, ohne eine Initialisierung oder einen Verbindungsaufbau zum Server vornehmen zu müssen.

7 Zusammenfassung

Aufgrund der technischen Entwicklungen der letzten Jahrzehnte steigt auch die Vernetzungsdichte von Computersystemen. Je nach Anwendungsgebiet benötigen diese genaue und zuverlässige Zeitinformationen, um ihre Tätigkeiten wahrnehmen zu können. Die Realisierung erfolgt heutzutage durch weit verbreitete Zeitsynchronisierungsprotokolle, wie etwa NTP oder PTP, die große Computernetze zeitlich synchronisieren können. Aufgrund des steigenden Bedarfs an Sicherheit, die auch die unverfälschte Übertragung von Zeitdaten beinhaltet, wurden im Jahr 2011 die bestehenden Sicherheitsmechanismen der Zeitprotokolle untersucht. Das Ergebnis zeigte gravierende Schwächen und setzte die Hebel zur Ausarbeitung einer neuen und zeitgemäßen Spezifikation in Gange. Fortan wurde an dem neuen *Network-Time-Security*-Protokoll gearbeitet, welches sich derzeit immer noch in der Entwicklungsphase befindet und zukünftig die erforderliche Sicherheit garantieren soll.

NTS stellt eine Erweiterung bestehender Zeitprotokolle dar und garantiert die volle Kompatibilität zu ungesicherten Verbindungen. Im NTPv4-Protokoll wird NTS in den Erweiterungsfeldern eingebettet und ermöglicht somit den Austausch von Nachrichten zwischen dem Client und Zeitserver. Hierbei unterteilt sich die NTS-Kommunikation in die Verbindungs- und Zeitsynchronisationsphase. In der Verbindungsphase – auch *Key Exchange* genannt – kommt es zum Austausch von Verbindungsparametern und Zertifikaten. Diese Phase untergliedert sich wiederum in drei Nachrichtenpaare. Die Kommunikation beginnt zunächst mit einer Zugriffsanfrage des Clients, in welcher der Server die entsprechende Berechtigung durch Übertragung eines Zugriffsschlüssels bewilligt. Anschließend folgen die Assoziationsnachrichten, in denen die kryptografischen Algorithmen und die Protokollversion ausgehandelt werden. Zudem erfolgt der Austausch des Serverzertifikats, wodurch der Client die Authentizität des Zeitservers prüfen kann. Im Anschluss generiert der Client eine Cookie-Nachricht, in dem er einen geheimen Schlüssel anfordert den nur Client und Server kennen. Ist der Client nach diesem Nachrichtenaustausch im Besitz des Cookies, so gilt die Verhandlungsphase als abgeschlossen. In der nun folgenden Zeitsynchronisationsphase fordert der Client MAC-gesicherte Zeitnachrichten an. Diese kann er nach Erhalt durch das gespeicherte Cookie überprüfen und somit die Integrität der Nachricht bestätigen. Die anschließende Freigabe an das übergeordnete NTP-Protokoll ermöglicht die Synchronisierung der lokalen Uhrzeit. Der Client kann fortan erneut Zeitnachrichten anfordern, so lange das Cookie gültig ist. An dieser Stelle sei noch erwähnt, dass der Zeitserver während der kompletten Kommunikation im Gegensatz zum Client zustandslos ist und keine Informationen speichert.

Der strukturelle Aufbau von NTS wird durch sogenannte Archetypen definiert. Diese bestehen aus CMS-Strukturen, die in der Beschreibungssprache ASN.1 definiert sind. Sie beinhalteten die verschlüsselten Informationen (das Cookie), digitale Signaturen oder X.509-

Zertifikate und transportieren das NTS-Nachrichtenobjekt, in dem die NTS-bezogenen Verbindungsparameter gespeichert sind. Die entsprechenden Strukturen werden als NTS-Nachricht oder NTS-MAC in die NTP-Erweiterungsfelder integriert und sind durch eindeutige Kennungen identifizierbar.

Die dazugehörige Implementierung ist modular aufgebaut und in der Programmiersprache C++ 11 umgesetzt. Die Implementierung kommuniziert über eine definierte Schnittstelle mit NTP und erlaubt somit die einfache Anpassung an andere Protokolle. Die interne Struktur setzt sich aus Sub-Modulen zusammen und ermöglicht zum einen die Kommunikation zu externen Bibliotheken (ASN.1 und OpenSSL) und zum anderen einen strukturierten Datenfluss zwischen den Komponenten. Der NTS-Dienst beinhaltet sowohl den Client, als auch den Server. Beide arbeiten unabhängig voneinander und können über entsprechende Konfigurationsdateien frei angepasst werden. Darüber hinaus können beide mit einer unbegrenzten Anzahl an Teilnehmer kommunizieren. Zudem ist es möglich, mehrere Instanzen des NTS-Diensts simultan zu betreiben.

Wie die anschließenden Tests zeigten, funktioniert die Implementierung entsprechend der Spezifikation korrekt. Die benötigten Ressourcen (Arbeitsspeicher) sind gering und die von NTS zusätzlich erzeugte Latenz ist ausreichend niedrig, sodass diese den derzeitigen Erwartungen entspricht. Nichtsdestotrotz sind noch diverse Optimierungen und intensive Tests der Implementierung nötig, um den praxistauglichen Einsatz des NTS-Dienstes zu gewährleisten. Zudem ist die weitere Entwicklung der NTS-Spezifikation abzuwarten, da sich diese noch in der Entwicklungsphase befindet und diverse Änderungen zu erwarten sind.

Quellenverzeichnis

Fachbücher

[14] A. Sikora, *Technische Grundlagen der Rechnerkommunikation - Internet-Protokolle und Anwendungen*, München, Wien: Fachbuchverl. Leipzig im Carl-Hanser-Verl., 2003, ISBN: 3-446-22455-6.

[15] J. Kurose; K. Ross, *Computernetzwerke - Der Top-Down-Ansatz*, 6. aktualisierte Auflage, Hallbergmoos: Pearson Studium, 2014, ISBN: 978-3-86894-237-8.

[28] O. Dubuisson, *ASN.1 - Communication between Heterogeneous Systems*, San Diego: Morgan Kaufmann, 2000, ISBN: 0-12-6333361-0.

[31] J. Larmouth, *ASN.1 Complete*, San Diego, CA: Academic Press, 1999, ISBN-13: 978-0122334351.

[42] J. Buchmann, *Einführung in die Kryptographie*, Berlin, Heidelberg: 4. Auflage: Springer Verlag, 2008, ISBN: 978-3-540-74451-1.

Berichte, Fachpräsentationen, Papers und Veröffentlichungen

[3] D. Piester; P. Hetzel; A. Bauch, *Zeit- und Normalfrequenzverbreitung mit DCF77*, PTB-Mitteilungen 114, Heft 4. Themenschwerpunkt 345: Physikalisch-Technische Bundesanstalt, 2004, [Online] URL: https://www.ptb.de/cms/fileadmin/internet/fachabteilungen/abteilung_4/4.4_zeit_und_frequenz/4.42/dcf77.pdf – Datum des Abrufs: 16.10.2016.

[12] H. Zimmermann, *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, IEEE transactions on communications, Vol. COM-28: S. 425 ff., 04. April 1980, [Online] URL: http://www.cs.cmu.edu/~srini/15-744/F02/-readings/Zim80.pdf – Datum des Abrufs: 16.10.2016.

[21] D. Mills, *Network Time Protocol Version 4 - Reference and Implementation Guide*, Technical Report 06-6-1 (TR 06-6-1), University of Delaware: NTP Working Group, Juni 2006, [Online] URL: https://www.eecis.udel.edu/~mills/database/reports/ntp4/-ntp4.pdf – Datum des Abrufs: 16.10.2016.

[23] A. Dreher; D. Mohl, *Präzise Uhrzeitsynchronisation - Der Standard IEEE 1588*, White Paper, Rev. 1.2, Neckartenzlingen (Germany): Hirschmann Automation and Control GmbH, 2005, [Online] URL: http://www.pdv.reutlingen-university.de/rte/-White_paper_ieee1588_de_v1-2.pdf – Datum des Abrufs: 16.10.2016.

[26] E. Itkin; A. Wool, *A Security Analysis and Revised Security Extension for the Precision Time Protocol*, arXiv:1603.00707v3 [cs.CR], Tel Aviv University, Israel: Tel Aviv University, 28. Mai 2016, [Online] URL: https://arxiv.org/pdf/1603.00707-v3.pdf – Datum des Abrufs: 16.10.2016.

[29] B. Blanc; B. Maaraoui, *Endianness or Where is Byte 0?*, White Paper: 3B Consultancy, Dez. 2005, [Online] URL: http://3bc.bertrand-blanc.com/endianness05.pdf – Datum des Abrufs: 16.10.2016.

[30] B. S. Kaliski Jr., *A Layman's Guide to a Subset of ASN.1, BER, and DER*, RSA Laboratories Technical Note, Redwood City (USA): RSA Laboratories, 01. Nov. 1993, [Online] URL: http://www.zytrax.com/books/ldap/apb/asn1.pdf – Datum des Abrufs: 16.10.2016.

[53] S. Röttger, *Analyse des NTP-Autokey-Verfahrens (Projektarbeit)*, Technische Universität Braunschweig, Physikalisch-Technische Bundesanstalt, 09. Sep. 2011, [Online] URL: http://docplayer.org/4552446-Analyse-des-ntp-autokey-verfahrens.html – Datum des Abrufs: 16.10.2016.

[55] D. Sibold; S. Röttger, *Analysis of NTP's Autokey Protocol,* Fachpräsentation, 83. IETF-Treffen, Paris (France): Physikalisch-Technische Bundesanstalt, 30. März 2012, [Online] URL: https://www.ietf.org/proceedings/83/slides/slides-83-tictoc-1.pdf – Datum des Abrufs: 16.10.2016.

[56] D. Sibold, *Plan for Autokey Update*, Fachpräsentation, 84. IETF-Treffen, Vancouver (Canada): Physikalisch-Technische Bundesanstalt, 03. Aug. 2012, [Online] URL: https://www.ietf.org/proceedings/84/slides/slides-84-tictoc-0.pdf – Datum des Abrufs: 16.10.2016.

[57] D. Sibold; S. Röttger, *Autokey Version 2 Specification*, Fachpräsentation, 85. IETF-Treffen, Atlanta (USA): Physikalisch-Technische Bundesanstalt, 09. Nov. 2012, [Online] URL: https://www.ietf.org/proceedings/85/slides/slides-85-tictoc-5.pdf – Datum des Abrufs: 16.10.2016.

[58] D. Sibold; S. Röttger, *Autokey Version 2 Specification*, Fachpräsentation, 86. IETF-Treffen, Orlando (USA): Physikalisch-Technische Bundesanstalt, 15. März 2013, [Online] URL: https://www.ietf.org/proceedings/86/slides/slides-86-tictoc-7.pdf – Datum des Abrufs: 16.10.2016.

[59] D. Sibold; S. Röttger, *Network Time Security*, Fachpräsentation, 87. IETF-Treffen, Berlin (Germany): Physikalisch-Technische Bundesanstalt, 02. Aug. 2013, [Online] URL: https://www.ietf.org/proceedings/87/slides/slides-87-tictoc-0.pdf – Datum des Abrufs: 16.10.2016.

[60] D. Sibold; K. Teichel, *Network Time Security*, Fachpräsentation, 88. IETF-Treffen, Vancouver, BC (Canada): Physikalisch-Technische Bundesanstalt, 08. Nov. 2013, [Online] URL: https://www.ietf.org/proceedings/88/slides/slides-88-ntp-0.pdf – Datum des Abrufs: 16.10.2016.

[61] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 89. IETF-Treffen, GB (London): Physikalisch-Technische Bundesanstalt, 07. März 2014, [Online] URL: https://www.ietf.org/proceedings/89/slides/slides-89-tictoc-2.pdf – Datum des Abrufs: 16.10.2016.

[62] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 90. IETF-Treffen, Toronto: Physikalisch-Technische Bundesanstalt, 25. Juli 2014, [Online] URL: https://www.ietf.org/proceedings/90/slides/slides-90-ntp-0.pdf – Datum des Abrufs: 16.10.2016.

[63] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 91. IETF-Treffen, Honolulu: Physikalisch-Technische Bundesanstalt, 14. Nov. 2014, [Online] URL: https://www.ietf.org/proceedings/91/slides/slides-91-ntp-3.pdf – Datum des Abrufs: 16.10.2016.

[64] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 92. IETF-Treffen, Dallas: Physikalisch-Technische Bundesanstalt, 22. März 2015, [Online] URL: https://www.ietf.org/proceedings/92/slides/slides-92-ntp-2.pdf – Datum des Abrufs: 16.10.2016.

[65] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 93. IETF-Treffen, Prague (Czech Republic): Physikalisch-Technische Bundesanstalt, 24. Juli 2015, [Online] URL: https://www.ietf.org/proceedings/93/slides/slides-93-ntp-1.pdf – Datum des Abrufs: 16.10.2016.

[66] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 94. IETF-Treffen, Yokohama (Japan): Physikalisch-Technische Bundesanstalt, 06. Nov. 2015, [Online] URL: https://www.ietf.org/proceedings/94/slides/slides-94-ntp-1.pdf – Datum des Abrufs: 16.10.2016.

[67] D. Sibold; K. Teichel; S. Röttger, *Network Time Security*, Fachpräsentation, 95. IETF-Treffen, Buenos Aires (Argentina): Physikalisch-Technische Bundesanstalt, 08. Apr. 2016, [Online] URL: https://www.ietf.org/proceedings/95/slides/slides-95-ntp-3.pdf – Datum des Abrufs: 16.10.2016.

[68] D. Sibold; K. Teichel, *Network Time Security*, Fachpräsentation, 96. IETF-Treffen, Berlin (Germany): Physikalisch-Technische Bundesanstalt, 22. Juli 2016, [Online] URL: https://www.ietf.org/proceedings/96/slides/slides-96-ntp-1.pdf – Datum des Abrufs: 16.10.2016.

[76] A. Malhotra; I. E. Cohen; E. Brakke; S. Goldberg, *Attacking the Network Time Protocol*, Boston University, Boston, MA., 21. Okt. 2015, [Online] URL: https://www.cs.bu.edu/~goldbe/papers/NTPattack.pdf – Datum des Abrufs: 16.10.2016.

[81] M. Bellare, *New Proofs for NMAC and HMAC: Security without Collision-Resistance*, CRYPTO '06, Lecture Notes in Computer Science Vol. 4117, University of California San Diego: Springer-Verlag, Juni 2006, [Online] URL: https://cse-web.ucsd.edu/~mihir/papers/hmac-new.pdf – Datum des Abrufs: 16.10.2016.

[90] H. Brandenburg, *Pseudozufallszahlengeneratoren in der Kryptographie*, Diplomarbeit, Humboldt-Universität zu Berlin: Institut für Informatik, 21. Dez. 2006, [Online] URL: https://www.informatik.hu-berlin.de/de/forschung/gebiete/algorithmenII/Lehre/abschlussarbeiten/pzzg.pdf – Datum des Abrufs: 16.10.2016.

[91] C. Schiestl, *Pseudozufallszahlen in der Kryptographie*, Universität Klagenfurt: Diplomarbeit, Mai 1999, [Online] URL: https://books.google.de/books?id=DYF2-AQAAQBAJ&lpg=PA15&ots=Od6Cfcucj5&dq=Pseudozufallszahlen%20in%20der%20Kryptographie&hl=de&pg=PA132#v=onepage&q&f=false – Datum des Abrufs: 16.10.2016, p. 132-139.

[103] Certicom Research, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Certicom Research, Version 1.0, 20. Sep. 2000, [Online] URL: http://www.secg.org-/SEC2-Ver-1.0.pdf – Datum des Abrufs: 16.10.2016.

[124] C. Shannon; D. Moore; K. Claffy, *Beyond Folklore: Observations on Fragmented Traffic*, Journal, IEEE/ACM transactions on networking, Volume 10 Issue 6, Dez. 2002, [Online] URL: https://www.caida.org/publications/papers/2002/Frag/frag.pdf – Datum des Abrufs: 16.10.2016, p. 709-720.

[128] BSI, *Das Smart Meter Gateway, Sicherheit für intelligente Netze*, BSI-Bro15/332, Bonn (Germany): Bundesamt für Sicherheit in der Informationstechnik, Okt. 2015, [Online] URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Smart-Meter-Gateway.pdf?__blob=publicationFile&v=2 – Datum des Abrufs: 16.10.2016.

Standards, Gesetze und Richtlinien

[1] BMJV, *Gesetz über die Einheiten im Messwesen und die Zeitbestimmung (Einheiten- und Zeitgesetz - EinhZeitG)*, Bundesministeriums der Justiz und für Verbraucherschutz, 02.07.1969, [Online] URL: https://www.gesetze-im-internet.de/bundesrecht/me_einhg/gesamt.pdf – Datum des Abrufs: 16.10.2016.

[5] Rat der Europäischen Union, *Richtlinie 2009/72/EG des europäischen Parlaments und des Rates*, Amtsblatt der Europäischen Union, Vorschriften für den Elektrizitätsbinnenmarkt: Europäisches Parlament, 13. Juli 2009, [Online] URL: http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32009L0072&from=DE – Datum des Abrufs: 16.10.2016.

[7] BSI, *Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems*, TR-03109-1, Bonn (Germany): Bundesamt für Sicherheit in der Informationstechnik, 18. März 2013, [Online] URL: https://-www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR03109-1.pdf?__blob=publicationFile – Datum des Abrufs: 16.10.2016.

[8] PTB, *PTB-Anforderungen: Smart Meter Gateway,* PTB-A 50.8: Physikalisch-Technische Bundesanstalt, Dez. 2014, [Online] URL: https://www.ptb.de/cms-/fileadmin/internet/fachabteilungen/abteilung_q/q.3_gesetzliches_messwesen/q.31/ptb-a/PTB-A_50-08.pdf – Datum des Abrufs: 16.10.2016.

[9] D. Sibold; S. Roettger; K. Teichel, *Network Time Security (NTS),* Internet-Draft: draft-ietf-ntp-network-time-security-15: Internet Engineering Task Force (IETF), 22. Sep. 2016, [Online] URL: https://tools.ietf.org/pdf/draft-ietf-ntp-network-time-security-15.pdf – Datum des Abrufs: 16.10.2016.

[10] BSI, *Kryptographische Vorgaben für Projekte der Bundesregierung, Teil 3 - Intelligente Messsysteme,* BSI TR-03116-3, Bonn (Germany): Bundesamt für Sicherheit in der Informationstechnik, 26. März 2015, [Online] URL: https://-www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-3.pdf?__blob=publicationFile&v=3 – Datum des Abrufs: 16.10.2016.

[13] ISO/IEC, *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model,* ISO/IEC 7498-1:1994(E), Switzerland, 15. Nov. 1994, [Online] URL: http://www.ecma-international.org/activities/Communications/TG-11/s020269e.pdf – Datum des Abrufs: 16.10.2016.

[17] D. Mills, *Network Time Protocol (NTP),* Request for Comments: 958: Internet Engineering Task Force (IETF), Sep. 1985, [Online] URL: https://tools.ietf-.org/pdf/rfc958.pdf – Datum des Abrufs: 16.10.2016.

[18] D. Mills; et al., *Network Time Protocol Version 4: Protocol and Algorithms Specification,* Request for Comments: 5905: Internet Engineering Task Force (IETF), Juni 2010, [Online] URL: https://tools.ietf.org/pdf/rfc5905.pdf – Datum des Abrufs: 16.10.2016.

[27] ITU-T Study Group, *Abstract Syntax Notation One (ASN.1) - Specification of basic notation,* ITU-T Rec. X.680-0207, ISO/IEC 8824-1: ITU Telecommunication Standardization Sector, 14. Juli 2002, [Online] URL: https://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf – Datum des Abrufs: 16.10.2016.

[32] M. Mealling, *A URN Namespace of Object Identifiers,* Request for Comments: 3061: Internet Engineering Task Force (IETF), Feb. 2001, [Online] URL: https://tools.ietf.org/pdf/rfc3061.pdf – Datum des Abrufs: 16.10.2016.

[35] ITU-T Study Group, *ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, ITU-T Rec. X.690-0207, ISO/IEC 8825-1: ITU Telecommunication Standardization Sector, 14. Juli 2002, , [Online] URL: https://www.itu.int/ITU-T/studygroups-/com17/languages/X.690-0207.pdf – Datum des Abrufs: 16.10.2016.

[36] ITU-T Study Group 17, *ASN.1 encoding rules: XML Encoding Rules (XER)*, ITU-T Rec. X.693-0112, ISO/IEC 8825-4: ITU Telecommunication Standardization Sector, 22. Dez. 2001, [Online] URL: https://www.itu.int/ITU-T/studygroups-/com17/languages/X.693-0112.pdf – Datum des Abrufs: 16.10.2016.

[40] FIPS, *Data Encryption Standard (DES)*, FIPS PUB 46-3: Federal Information Processing Standard, 25. Okt. 1999, [Online] URL: http://csrc.nist.gov-/publications/fips/fips46-3/fips46-3.pdf – Datum des Abrufs: 16.10.2016.

[41] FIPS, *Advanced Encryption Standard (AES)*, FIPS PUB 197: Federal Information Processing Standard, 26 Nov. 2001, [Online] URL: http://csrc.nist.gov-/publications/fips/fips197/fips-197.pdf – Datum des Abrufs: 16.10.2016.

[45] Krawczyk; et. al., *HMAC: Keyed-Hashing for Message Authentication*, Request for Comments: 2104: Internet Engineering Task Force (IETF), Feb. 1997, [Online] URL: https://tools.ietf.org/pdf/rfc2104.pdf – Datum des Abrufs: 16.10.2016.

[46] Song; et al., *The AES-CMAC Algorithm*, Request for Comments: 4493: Internet Engineering Task Force (IETF), Juni 2006, [Online] URL: https://tools.ietf.org-/pdf/rfc4493.pdf – Datum des Abrufs: 16.10.2016.

[47] ITU-T, *Open systems interconnection – The Directory: Public-key and attribute certificate frameworks*, ITU-T Rec. X.509 (2000 E), ISO/IEC 9594-8: ITU Telecommunication Standardization Sector, 31. März 2000, [Online] URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-200003-S!!PDF-E&type=items – Datum des Abrufs: 16.10.2016.

[48] Housley; et. al., *Internet X.509 Public Key Infrastructure - Certificate and CRL Profile*, Request for Comments: 2459: Internet Engineering Task Force (IETF), Jan. 1999, [Online] URL: https://tools.ietf.org/pdf/rfc2459.pdf – Datum des Abrufs: 16.10.2016.

[49] R. Housley, *Cryptographic Message Syntax (CMS)*, Request for Comments: 5652: Internet Engineering Task Force (IETF), Sep. 2009, [Online] URL: https://tools.ietf.org/pdf/rfc5652.pdf – Datum des Abrufs: 16.10.2016.

[50] B. Kaliski, *PKCS #7: Cryptographic Message Syntax - Version 1.5*, Request for Comments: 2315: Internet Engineering Task Force (IETF), März 1998, [Online] URL: https://tools.ietf.org/pdf/rfc2315.pdf – Datum des Abrufs: 16.10.2016.

[51] D. Sibold; S. Roettger; K. Teichel, *Using the Network Time Security Specification to Secure the Network Time Protocol (NTS4NTP)*, Internet-Draft: draft-ietf-ntp-using-nts-for-ntp-06: Internet Engineering Task Force (IETF), 22. Sep. 2016, [Online] URL: https://tools.ietf.org/pdf/draft-ietf-ntp-using-nts-for-ntp-06.pdf – Datum des Abrufs: 16.10.2016.

[52] Sibold; et al., *Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS), CMS4NTS*, Internet-Draft: draft-ietf-ntp-cms-for-nts-message-07: Internet Engineering Task Force (IETF), 21. März 2016, [Online] URL: https://tools.ietf.org/pdf/draft-ietf-ntp-cms-for-nts-message-06.pdf – Datum des Abrufs: 16.10.2016.

[54] D. Sibold; S. Roettger, *Network Time Protocol: autokey Version 2 Specification*, Internet-Draft: draft-sibold-autokey-00: Internet Engineering Task Force (IETF), 30. Juli 2012, [Online] URL: https://tools.ietf.org/pdf/draft-sibold-autokey-00.pdf – Datum des Abrufs: 16.10.2016.

[71] T. Mizrahi, *Security Requirements of Time Protocols in Packet Switched Networks*, Request for Comments: 7384: Internet Engineering Task Force (IETF), Okt. 2014, [Online] URL: https://tools.ietf.org/pdf/rfc7384.pdf – Datum des Abrufs: 16.10.2016.

[79] D. Mills, *Network Time Protocol (Version 3) - Specification, Implementation and Analysis*, Request for Comments: 1305: Internet Engineering Task Force (IETF), März 1992, [Online] URL: https://tools.ietf.org/pdf/rfc1305.pdf – Datum des Abrufs: 16.10.2016.

[83] B. Haberman; D. Mills, *Network Time Protocol Version 4: Autokey Specification*, Request for Comments: 5906: Internet Engineering Task Force (IETF), Juni 2010, [Online] URL: https://tools.ietf.org/pdf/rfc5906.pdf – Datum des Abrufs: 16.10.2016.

[86] BSI, *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*, BSI TR-02102-1, v2016-01, Bonn (Germany): Bundesamt für Sicherheit in der Informationstechnik, 15. Feb. 2016, [Online] URL: https://www.bsi.bund.de-/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile – Datum des Abrufs: 16.10.2016.

[87] BSI, *Anlage I: CMS-Datenformat für die Inhaltsdatenverschlüsselung und -signatur*, BSI TR-03109-1, Version 1.0, Bonn (Germany): Bundesamt für Sicherheit in der Informationstechnik, 2013, [Online] URL: https://www.bsi.bund.de-/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_CMS.pdf;jsessionid=8A20507BDECE5C82078D9435BA8-2EB81.2_cid286?__blob=publicationFile&v=1 – Datum des Abrufs: 16.10.201.

[88] T. Mizrahi; D. Mayer, *Network Time Protocol Version 4 (NTPv4) Extension Fields*, Request for Comments: 7822: Internet Engineering Task Force (IETF), März 2016, [Online] URL: https://tools.ietf.org/pdf/rfc7822.pdf – Datum des Abrufs: 16.10.2016.

[89] T. Mizrahi, *UDP Checksum Complement in the Network Time Protocol (NTP)*, Request for Comments: 7821: Internet Engineering Task Force (IETF), März 2016, [Online] URL: https://tools.ietf.org/pdf/rfc7821.pdf – Datum des Abrufs: 16.10.2016.

[92] Eastlake; et al., *Randomness Requirements for Security*, Request for Comments: 4086: Internet Engineering Task Force (IETF), Juni 2005, [Online] URL: https://tools.ietf.org/pdf/rfc4086.pdf – Datum des Abrufs: 16.10.2016.

[95] A. Malhotra; S. Goldberg, *Message Authentication Codes for the Network Time Protocol*, Internet-Draft: draft-aanchal4-ntp-mac-00: Internet Engineering Task Force (IETF), 08. Juli 2016, [Online] URL: https://tools.ietf.org/pdf/draft-aanchal4-ntp-mac-00.pdf – Datum des Abrufs: 16.10.2016.

[97] Cooper; et al., *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Request for Comments: 5280: Internet Engineering Task Force (IETF), Mai 2008, [Online] URL: https://tools.ietf.org/pdf/rfc5280.pdf – Datum des Abrufs: 16.10.2016.

[101] Booz; et al., *Federal Public Key Infrastructure (PKI) X.509 Certificate and CRL Extensions Profile*, Draft, Linthicum, Maryland: National Institute of Standards and

Technology, 30. Apr. 2002, [Online] URL: http://csrc.nist.gov/archive/pki-twg/y2002/papers/twg-02-10.pdf – Datum des Abrufs: 16.10.2016.

[102] M. Lochter; J. Merkle, *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*, Request for Comments: 5639: Internet Engineering Task Force (IETF), März 2010, [Online] URL: https://tools.ietf.org/pdf/rfc5639.pdf – Datum des Abrufs: 16.10.2016.

[106] FIPS, *Secure Hash Standard (SHS)*, FIPS PUB 180-4: Federal Information Processing Standard, 05. Aug. 2015, [Online] URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf – Datum des Abrufs: 16.10.2016.

[107] R. Rivest, *The MD4 Message-Digest Algorithm*, Request for Comments: 1320: Internet Engineering Task Force (IETF), Apr. 1992, [Online] URL: https://tools.ietf-.org/pdf/rfc1320.pdf – Datum des Abrufs: 16.10.2016.

[108] R. Rivest, *The MD5 Message-Digest Algorithm*, Request for Comments: 1321: Internet Engineering Task Force (IETF), Apr. 1992, [Online] URL: https://tools.ietf-.org/pdf/rfc1321.pdf – Datum des Abrufs: 16.10.2016.

[109] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, NIST Special Publication 800-38D, Gaithersburg: NISTNational Institute of Standards and Technology, Nov. 2007, [Online] URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication-800-38d.pdf – Datum des Abrufs: 16.10.2016.

[120] ITU-CCITT, *Authentication Framework*, Fascicle VIII.8 – Rec. X.509, ISO 9594-8: ITU Telecommunication Standardization Sector, Nov. 1988, [Online] URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-198811-S!!PDF-E&type=items – Datum des Abrufs: 16.10.2016.

[121] ITU-T, *Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree*, Rec. ITU-T X.660 (07/2011), ISO/IEC 9834-1, Nov. 2011.

Internetquellen

[2] ptb.de, *Zeit und Frequenz,* [Online] URL: http://www.ptb.de/cms/ptb/fachabteilungen/abt4/fb-44.html – Datum des Abrufs: 16.10.2016.

[4] ptb.de, *Serversysteme, Datenhaltung und Zeitverteilung mit IP,* [Online] URL: https://www.ptb.de/cms/ptb/fachabteilungen/abtq/fb-q4/ag-q42/zeitsynchronisation-von-rechnern-mit-hilfe-des-network-time-protocol-ntp.html – Datum des Abrufs: 16.10.2016.

[6] buzer.de, *Gesetz über die Elektrizitäts- und Gasversorgung (Energiewirtschaftsgesetz - EnWG),* [Online] URL: https://www.buzer.de/gesetz/2151/index.htm – Datum des Abrufs: 16.10.2016.

[11] elektronik-kompendium.de, *ISO/OSI-7-Schichtenmodell,* [Online] URL: http://-www.elektronik-kompendium.de/sites/kom/0301201.htm – Datum des Abrufs: 16.10.2016.

[16] netzwerke.com, *OSI-Schichten-Modell,* [Online] URL: http://www.netzwerke.com/OSI-Schichten-Modell.htm – Datum des Abrufs: 16.10.2016.

[19] ntp.org, *ntp.org,* [Online] URL: http://www.ntp.org/ – Datum des Abrufs: 16.10.2016.

[20] D. Mills, *The Network Time Protocol (NTP) Distribution,* 2014, [Online] URL: https://www.eecis.udel.edu/~mills/ntp/html/index.html – Datum des Abrufs: 16.10.2016.

[22] O. Hager, *Allgemeines zum Network Time Protocol,* 2002, [Online] URL: http://doc-tcpip.org/Ntp/basics.html – Datum des Abrufs: 16.10.2016.

[24] itwissen.info, *PTP (precision time protocol),* [Online] URL: http://www.itwissen.info/definition/lexikon/precision-time-protocol-PTP.html – Datum des Abrufs: 16.10.2016.

[25] Ingenieurbüro obonic, *PTP - IEEE 1588,* [Online] URL: http://www.obonic.de/ptp-ieee-1588-precision-time-protocol/ – Datum des Abrufs: 16.10.2016.

[33] oid-info.com, *sha1-with-rsa-signature(5),* [Online] URL: http://www.oid-info.com/get/1.2.840.113549.1.1.5 – Datum des Abrufs: 16.10.2016.

[34] oid-info.com, *www.oid-info.com*, [Online] URL: http://www.oid-info.com – Datum des Abrufs: 16.10.2016.

[37] kryptowissen.de, *Kryptographie*, [Online] URL: http://www.kryptowissen.de/kryptographie.html – Datum des Abrufs: 16.10.2016.

[38] kryptowissen.de, *Kryptologie*, [Online] URL: http://www.kryptowissen.de/kryptologie.html – Datum des Abrufs: 16.10.2016.

[39] A Romeyke; et al., *Kryptologie - eine Einführung*, [Online] URL: http://andreas-romeyke.de/Projekt1/ffbr/web.html#vorwort – Datum des Abrufs: 16.10.2016.

[43] elektronik-kompendium.de, *Asymmetrische Kryptografie (Verschlüsselung)*, [Online] URL: http://www.elektronik-kompendium.de/sites/net/1910111.htm – Datum des Abrufs: 16.10.2016.

[44] itwissen.info, *Hybrid-Verschlüsselung*, [Online] URL: http://www.itwissen-.info/definition/lexikon/Hybrid-Verschluesselung-hybrid-encryption.html – Datum des Abrufs: 16.10.2016.

[69] ietf.org, *Network Time Security*, [Online] URL: https://datatracker.ietf-.org/doc/draft-ietf-ntp-network-time-security – Datum des Abrufs: 16.10.2016.

[70] ietf.org, *Network Time Protocol (ntp)*, [Online] URL: https://datatracker.ietf-.org/wg/ntp/documents/ – Datum des Abrufs: 16.10.2016.

[72] elektronik-kompendium.de, *Man-in-the-Middle*, [Online] URL: http://www.elektronik-kompendium.de/sites/net/1710251.htm – Datum des Abrufs: 16.10.2016.

[73] itwissen.info, *DoS (denial of service) - DoS-Attacke*, [Online] URL: http://www.itwissen.info/definition/lexikon/denial-of-service-DoS-DoS-Attacke.html – Datum des Abrufs: 16.10.2016.

[74] itwissen.info, *DDoS (distributed denial of service) - DDoS-Attacke*, [Online] URL: http://www.itwissen.info/definition/lexikon/distributed-denial-of-service-DDoS-DDoS-Attacke.html – Datum des Abrufs: 16.10.2016.

[75] J. Graham-Cumming, *Understanding and mitigating NTP-based DDoS attacks*, 2014, [Online] URL: https://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks/ – Datum des Abrufs: 16.10.2016.

[77] H. Böck, *Zeitserver: Neue Angriffe auf das Network Time Protocol*, 2015, [Online] URL: http://www.golem.de/news/zeitserver-neue-angriffe-auf-das-network-time-protocol-1510-117076.html – Datum des Abrufs: 16.10.2016.

[78] itwissen.info, *Replay-Angriff*, [Online] URL: http://www.itwissen.info/definition/lexikon/Replay-Angriff-replay-attack.html – Datum des Abrufs: 16.10.2016.

[80] ntp.org, *Authentication Options*, 2006, [Online] URL: http://doc.ntp.org-/4.2.4/authopt.html – Datum des Abrufs: 16.10.2016.

[82] ntp.org, *ntp-keygen - generate public and private keys*, 2005, [Online] URL: http://doc.ntp.org/4.2.4/keygen.html – Datum des Abrufs: 16.10.2016.

[84] itwissen.info, *Brute-Force-Angriff*, [Online] URL: http://www.itwissen.info/-definition/lexikon/Brute-Force-Angriff-brute-force-attack.html – Datum des Abrufs: 16.10.2016.

[85] D. Mills, *Autokey Identity Schemes*, [Online] URL: https://www.eecis.udel.edu-/~mills/ident.html – Datum des Abrufs: 16.10.2016.

[93] microsoft.com, *The Cryptography API, or How to Keep a Secret*, 1996, [Online] URL: https://msdn.microsoft.com/en-us/library/ms867086.aspx – Datum des Abrufs: 16.10.2016.

[94] man7.org, *Linux Programmer's Manual - RANDOM(4)*, [Online] URL: http://man7.org/linux/man-pages/man4/random.4.html – Datum des Abrufs: 16.10.2016.

[96] D. Wei, *Crypto++ 5.6.0 Benchmarks (Speed Comparison of Popular Crypto Algorithms)*, 2009, [Online] URL: https://www.cryptopp.com/benchmarks.html – Datum des Abrufs: 16.10.2016.

[98] cryptas.com, *Grundlagen einer Public Key Infrastructure*, 2011, [Online] URL: http://www.cryptas.com/cryptas/wissen-unterstuetzung/pki-grundlagen/x509-zertifikate.html – Datum des Abrufs: 16.10.2016.

[99] openssl.org, *X509_check_ca*, [Online] URL: https://www.openssl.org/docs/man-master/crypto/X509_check_ca.html – Datum des Abrufs: 16.10.2016.

[100] microsoft.com, *How to make a stand-alone certification authority that is running Windows Server 2003*, [Online] URL: https://support.microsoft.com/en-us/kb/888180 – Datum des Abrufs: 16.10.2016.

[104] safecurves.cr.yp.to, *SafeCurves: choosing safe curves for elliptic-curve cryptography*, 2014, [Online] URL: https://safecurves.cr.yp.to/ – Datum des Abrufs: 16.10.2016.

[105] itwissen.info, *Blockchiffre*, [Online] URL: http://www.itwissen.info/definition-/lexikon/Blockchiffre-block-cipher.html – Datum des Abrufs: 16.10.2016.

[110] debian.org, *Über Debian*, [Online] URL: https://www.debian.org/intro/about – Datum des Abrufs: 16.10.2016.

[111] eclipse.org, *Eclipse CDT (C/C++ Development Tooling)*, [Online] URL: http://www.eclipse.org/cdt/ – Datum des Abrufs: 16.10.2016.

[112] valgrind.org, *About Valgrind*, 2005, [Online] URL: http://valgrind.org-/info/about.html – Datum des Abrufs: 16.10.2016.

[113] stack.nl, *Doxygen*, [Online] URL: http://www.stack.nl/~dimitri/doxygen/ – Datum des Abrufs: 16.10.2016.

[114] lionet.info, *asn1c Documentation*, [Online] URL: http://lionet.info/asn1c/-documentation.html – Datum des Abrufs: 16.10.2016.

[115] objective Systems inc., *ASN1C ASN.1 Compiler*, [Online] URL: https://www.obj-sys.com/products/asn1c/ – Datum des Abrufs: 16.10.2016.

[116] bellard.org, *FFASN1 Compiler*, [Online] URL: http://bellard.org/ffasn1/ – Datum des Abrufs: 16.10.2016.

[117] gnu.org, *GNU Libtasn1*, [Online] URL: https://www.gnu.org/software/libtasn1/ – Datum des Abrufs: 16.10.2016.

[118] openssl.org, *Welcome to OpenSSL!*, [Online] URL: https://www.openssl.org/ – Datum des Abrufs: 16.10.2016.

[119] wikipedia.org, *Comparison of TLS implementations*, 2016, [Online] URL: https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations – Datum des Abrufs: 16.10.2016.

[122] ietf.org, *Network Time Security WGLC Design Team Discussions*, [Online] URL: https://trac.tools.ietf.org/wg/ntp/trac/wiki/NtsWglcDesignTeam – Datum des Abrufs: 16.10.2016.

[123] Working Group Last Call Discussion, *Agenda Items (and Priorities) for the Design Team Assigned to Resolving Working Group Last Call Discussion on NTS Drafts*, [Online] URL: https://docs.google.com/document/d/1CR5mFOP_WZ_FZDTf-0IWsXLFOvhO4AXokZNvOb3PWI7E – Datum des Abrufs: 16.10.2016.

[125] NIST, *A Brief History of Atomic Clocks at NIST*, [Online] URL: http://tf.nist-.gov/cesium/atomichistory.htm – Datum des Abrufs: 16.10.2016.

[126] Dellwall, *Die exakte Uhrzeit der Atomuhr für Deutschland*, [Online] URL: http://www.atomuhr-infos.de/ – Datum des Abrufs: 16.10.2016.

[127] inf-schule.de, *Fachkonzept - Kryptologische Hash-Funktion*, [Online] URL: http://www.inf-schule.de/kommunikation/kryptologie/digitalesignatur/konzept_-hashfunktion – Datum des Abrufs: 16.10.2016.

Anhang

➔ beigefügten DVD:

```
01 - Dokumentation und Anlagen
02 - Präsentation
03 - Quellen
      01 - Fachbücher
      02 - Berichte, Fachpräsentationen, Papers und Veröffentlichungen
      03 - Standards, Gesetze und Richtlinien
      04 - Internetquellen

04 - NTS Implementierung
      01 - C++ Quellcode
      02 - Doxygen-Dokumentation
      03 - Entwicklungsumgebung
            01 - VirtualBox-Image
            02 - Softwarepakete

05 - Sonstiges
```

➔ vier Anlagendokumente:

```
Anlage I:    Installation und Einrichtung der Eclipse-Entwicklungsum-
             gebung unter Linux in einer virtuellen Maschine

Anlage II:   ASN.1-Modulbeschreibungen in der NTS-Implementierung

Anlage III:  Ablaufbeschreibung durch Sequenzdiagramme

Anlage IV:   Ablaufbeschreibung durch Aktivitätsdiagramme
```

Anlagen

Ostfalia
Hochschule für angewandte Wissenschaften

Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb

Anlage I:

Installation und Einrichtung der Eclipse-Entwicklungs-umgebung unter Linux in einer virtuellen Maschine

Martin Langer
Matrikelnummer: 70405974
Eingereicht am: 21.10.2016

Ostfalia, Wolfenbüttel
Fakultät Elektrotechnik
Studiengang: Intelligente Mobilität und Energiesystemtechnik

Dokumentversion: 1.0.0.232
Letze Änderung: 18.10.2016 23:11:00

Inhaltsverzeichnis

Inhaltsverzeichnis i

Abbildungsverzeichnis ii

Listing-Verzeichnis iii

1 Installation der Entwicklungsumgebung 1

1.1 Voraussetzungen zur Installation ... 1
1.2 Einrichtung eines Profils in der virtuellen Maschine ... 2
1.3 Installation von Linux Debian 8.6.0 (x86) ... 6
1.4 Grundlegende Konfigurierung von Debian ... 11
1.4.1 Paketquellen anpassen ... 12
1.4.2 Die VM-Gasterweiterung installieren ... 13
1.4.3 Konfiguration des GRUB-Bootloaders ... 14
1.4.4 Benutzerrechte anpassen ... 15
1.4.5 Automatische Benutzeranmeldung ... 16
1.5 Installation der benötigten Software ... 16
1.5.1 Deinstallation nicht benötigter Softwarekomponenten ... 16
1.5.2 Installation benötigter Softwarekomponenten ... 17
1.5.2.1 Installation grundlegender Werkzeuge ... 17
1.5.2.2 ASN.1-Bibliothek (asn1c v0.9.27) ... 17
1.5.2.3 Doxygen (v1.8.13) ... 18
1.5.2.4 OpenSSL (v1.1.0b) ... 19
1.5.2.5 Eclipse-CDT (Luna v4.4) ... 20
1.6 Einrichtung der Software ... 20
1.6.1 Einrichten von Eclipse-CDT ... 21
1.6.1.1 Rechte zum Ausführen erteilen ... 21
1.6.1.2 Eclipse-CDT als Konsolenbefehl hinzufügen ... 21
1.6.1.3 Eclipse-CDT im Startmenü eintragen ... 21
1.6.1.4 Workspace-Pfad setzen ... 22
1.6.1.5 Eclipse-Plugin installieren: eclox ... 22
1.6.1.6 Eclipse-Plugin installieren: LogViewer ... 23
1.6.2 Anpassen der Debian-Oberfläche ... 23
1.6.2.1 Deaktivierung des Bildschirmschoners ... 23
1.6.2.2 Erstellen von zusätzlichen Desktop-Verknüpfungen ... 24
1.6.2.3 Konfiguration der Taskleiste (*LXPanel*) ... 25

2 Einrichten von Eclipse-Projekten 29

2.1.1 C++ 11 aktivieren ... 29
2.1.2 OpenSSL in Eclipse-Projekte nutzen ... 29

Abbildungsverzeichnis

Abbildung 1.1: Grundlegende VM-Einstellungen (Allgemein).....3
Abbildung 1.2: Grundlegende VM-Einstellungen (System 1).....3
Abbildung 1.3: Grundlegende VM-Einstellungen (System2).....4
Abbildung 1.4: Grundlegende VM-Einstellungen (System 3).....4
Abbildung 1.5: Grundlegende VM-Einstellungen (Anzeige) 5
Abbildung 1.6: Grundlegende VM-Einstellungen (Massenspeicher).....5
Abbildung 1.7: Grundlegende VM-Einstellungen (Gemeinsame Ordner) 6
Abbildung 1.8: Linux-Installation: Installationsbildschirm von Linux Debian 8.6.0.....7
Abbildung 1.9: Linux-Installation: Festplatte partitionieren 8
Abbildung 1.10: Linux-Installation: Wahl der Desktopumgebung.....10
Abbildung 1.11: Erster Zugriff auf das Terminal 11
Abbildung 1.12: Bearbeitung der Paketquellen 12
Abbildung 1.13: Gasterweiterung einlegen.....13
Abbildung 1.14: Start von Debian nach Installation der Software 20
Abbildung 1.15: Abgeschlossene Einrichtung von Debian 28

Listing-Verzeichnis

Listing 1.1: Eigenschaften der virtuellen Maschine 2
Listing 1.2: Debian-Installation: Grundlegende Einstellungen 7
Listing 1.3: Debian-Installation: Festplatte partitionieren 8
Listing 1.4: Debian-Installation: Paketmanager konfigurieren 9
Listing 1.5: Debian-Installation: Desktopumgebung und Standardsoftware 9
Listing 1.6: Debian-Installation: GRUB-Bootloader konfigurieren 10
Listing 1.7: Als Root anmelden 11
Listing 1.8: Root-Rechte erteilen 11
Listing 1.9: Paketquellen öffnen 12
Listing 1.10: Angepasster Inhalt der Paketquellen 12
Listing 1.11: Aktualisierung des Betriebssystems 13
Listing 1.12: Installation der Gasterweiterung 14
Listing 1.13: Betriebssystem neustarten 14
Listing 1.14: Als Root anmelden 14
Listing 1.15: GRUB-Konfiguration öffnen 14
Listing 1.16: Neue Inhalte der GRUB-Konfiguration 15
Listing 1.17: GRUB-Bootloader aktualisieren 15
Listing 1.18: Ausführung von visudo 15
Listing 1.19: „user" volle Rechte erteilen 15
Listing 1.20: Die Konfiguration des Light Display Managers öffnen 16
Listing 1.21: Die Konfiguration des Light Display Managers anpassen 16
Listing 1.22: Zu löschende Software 17
Listing 1.23: Installation grundlegender Werkzeuge 17
Listing 1.24: Installation der ASN.1-Bibliothek 18
Listing 1.25: Installation der Werkzeuge für Doxygen 18
Listing 1.26: Installation von Doxygen 18
Listing 1.27: Installation von "zlib" für OpenSSL 19
Listing 1.28: Installation von OpenSSL 19
Listing 1.29: Installation von Eclipse-CDT 20
Listing 1.30: Eclipse die Rechte zum Ausführen erteilen 21
Listing 1.31: Eclipse als Konsolenbefehl hinzufügen 21

Listing 1.32: Eclipse zum Startmenü hinzufügen 21
Listing 1.33: Inhalt der "eclipse.desktop"-Datei 22
Listing 1.34: Eclipse-Workspace-Pfad definieren 22
Listing 1.35: Installation des Eclipse-Plugins: eclox 22
Listing 1.36: Installation des Eclipse-Plugins: LogViewer 23
Listing 1.37: Keyboard-Konfiguration öffnen 23
Listing 1.38: Angepasste kbd-Konfiguration 23
Listing 1.39: Den kbd-Dienst neustarten 24
Listing 1.40: Die Konfiguration des Light Display Managers öffnen 24
Listing 1.41: Die Konfiguration des Light Display Managers anpassen 24
Listing 1.42: Eclipse-Desktop-Verknüpfung erstellen 24
Listing 1.43: Öffnen des Dateimanagers im angegebenen Pfad 25
Listing 1.44: Desktop-Verknüpfung für den Worksapce-Ordner erstellen 25
Listing 1.45: Versteckte Dateien und Ordner anzeigen 25
Listing 1.46: Öffnen der LXPanel-Einstellungen 25
Listing 1.47: Neuer Inhalt der LXPanel-Konfiguration - Part 1 26
Listing 1.48: Neuer Inhalt der LXPanel-Konfiguration - Part 2 27
Listing 1.49: Neuer Inhalt der LXPanel-Konfiguration - Part 3 28
Listing 1.50: Betriebssystem neustarten 28
Listing 2.1: Aktivierung der C++ 11-Syntax 29
Listing 2.2: Nutzung der OpenSSL-Bibliothek in Projekten 29
Listing 2.3: Hinzufügen benutzerdefinierter Pfade in Eclipse-Projekten 30

1 Installation der Entwicklungsumgebung

Dieses Kapitel beschreibt die Installation und Einrichtung der Entwicklungsumgebung in Form einer Schritt-für-Schritt-Anleitung. Die Anleitung beinhaltet die Einrichtung eines Profils in der virtuellen Maschine (*VirtualBox*), die Installation und Einrichtung des Linux-Betriebssystems (*Debian x86*), und die Installation und Einrichtung von Eclipse-CDT (*Luna*) mit allen nötigen Komponenten. Diese Anleitung ist für Implementierung und Weiterentwicklung des NTS-Protokolls ausgerichtet und diskutiert keine Problembehandlungen im Kontext zum Linux-Betriebssystem oder dessen Bedienung.

1.1 Voraussetzungen zur Installation

Der Verlauf der folgenden Installation und Einrichtung hängt zunächst vom verwendeten Betriebssystem ab. Das implementierte NTS-Protokoll soll zunächst auf Linux-Betriebssystemen arbeiten, da Linux im Gegensatz zu *Microsoft Windows* oder *Apple Macintosh* eine wesentlich größere Flexibilität in Bezug auf die Hardware anbietet. So unterstützt Linux nicht nur leistungsstarke Computersysteme wie Server, sondern auch leistungsschwache Einplatinencomputer unterschiedlicher Hardware- und Prozessorarchitekturen. Steht zum Zeitpunkt der Einrichtung bereits ein Computersystem mit einem Linux-Betriebssystem zur Verfügung, so kann direkt mit der Systemkonfiguration im Kapitel 1.4 begonnen werden. Befindet sich auf dem Computer hingegen ein *Microsoft Windows*-Betriebssystem – was wahrscheinlich den Regelfall darstellt – so ist die Nutzung einer virtuellen Maschine (VM) ratsam. In diesem Falle beginnt die Anleitung im folgenden Kapitel 1.2.

Je nach Startpunkt der Anleitung werden folgende Ressourcen benötigt:

- eine Virtualisierungssoftware (z.B. ***VirtualBox 5.1.6***)
- ein Installation-Image oder -Datenträger von Linux (z.B. ***Debian 8.6.0 x86***)
- eine stabile Internetverbindung (**6 Mbit/s** oder höher empfohlen)
- ausreichend Festplattenspeicher (**min. 10 GB**)
- ausreichend Arbeitsspeicher (**min. 500 MB** für die VM)
- optionaler Weise einen zweiten Bildschirm

Der Installations- und Einrichtungsaufwand der kompletten Anleitung beträgt etwa 1 bis 3 Stunden und ist hauptsächlich von der Internet- und Computergeschwindigkeit abhängig. Der Nettoaufwand für die Durchführung der einzelnen Schritte, beträgt hingegen nur 10 bis 15 Minuten.

1.2 Einrichtung eines Profils in der virtuellen Maschine

Kommt als Host-System ein *Microsoft Windows*- oder *Apple Macintosh*-Betriebssystem zum Einsatz, so empfiehlt sich die Verwendung einer virtuellen Maschine. In dieser Anleitung wird dazu auf die kostenlose Virtualisierungssoftware *VirtualBox*[1] in der Version 5.1.6 zugegriffen, da diese sehr ausgereift ist und häufig Aktualisierungen bereitstellt. Der große Vorteil ist die simultane Ausführung von Windows und Linux mit geringen Leistungseinbußen und sauber getrennten Systemen. Darüber hinaus lassen sich virtuelle Maschinen exportieren und somit in wenigen Minuten auf andere Computersysteme übertragen.

Ist die Installation von VirtualBox vollzogen, so ist diese im Anschluss auszuführen[2]. Nach dem Start der Anwendung wird zunächst ein neues Profil (eine virtuelle Maschine) angelegt, der folgende Eigenschaften übertragen werden:

Listing 1.1: Eigenschaften der virtuellen Maschine

```
Name der virtuellen Maschine:          NTS (Debian x86)

Arbeitsspeicher (RAM):                 min. 500 MB
Festplattenspeicher:                   min.  10 GB
Festplattentyp:                        dynamisch wachsend

Grafikspeicher:                        128 MB
zugewiesene Prozessorkerne:            1

3D-Beschleunigung:                     aktiviert
Zwischenablage:                        bidirektional
gemeinsamer Ordner:                    E:\NTS (beispielhaft)
```

Die Abbildungen 1.1 bis 1.7 zeigen hierzu angefertigte Bildschirmfotos, in denen die entsprechenden Einstellungen zu tätigen sind. An dieser Stelle sei noch erwähnt, dass die Zuordnung der Prozessorkerne auf 1 festgelegt sein sollte. Die Zuweisung weiterer Kerne (*multicore*) reduziert entgegen den Erwartungen die Geschwindigkeit der VM drastisch[3]. Bei Problemen mit der Prozessorauslastung (durchgehende Volllast der VM) sollte zunächst die Einstellung „*Nested Paging*“ (vgl. Abbildung 1.4) deaktiviert werden. Bestehen weiterhin Probleme so ist die individuelle Fehlersuche mit Hilfe von Internet-Foren ratsam. Die Festlegung des *gemeinsamen Ordners* ermöglicht den Austausch von Dateien zwischen der VM und dem Host-System. Das angegebene Verzeichnis kann beliebig gewählt werden, sollte jedoch keine zu langen Pfade beinhalten, da diese auf Windows-Betriebssystem in der Regel auf 255- Zeichen begrenzt sind.

[1] Die aktuelle Version von *VirtualBox* kann jederzeit auf der offiziellen Webseite (https://www.virtualbox.org) bezogen werden.
[2] Schlägt die Ausführung fehl, so muss unter Umständen die Virtualisierung im BIOS (*Basic Input Output System*) des Computers aktiviert werden. Dieses Vorgehen ist individuell und wird an dieser Stelle nicht weiter erläutert.
[3] Dieses Problem kann individuell auftreten. Der Betrieb der VM im Singlecore-Modus ist daher oft schneller.

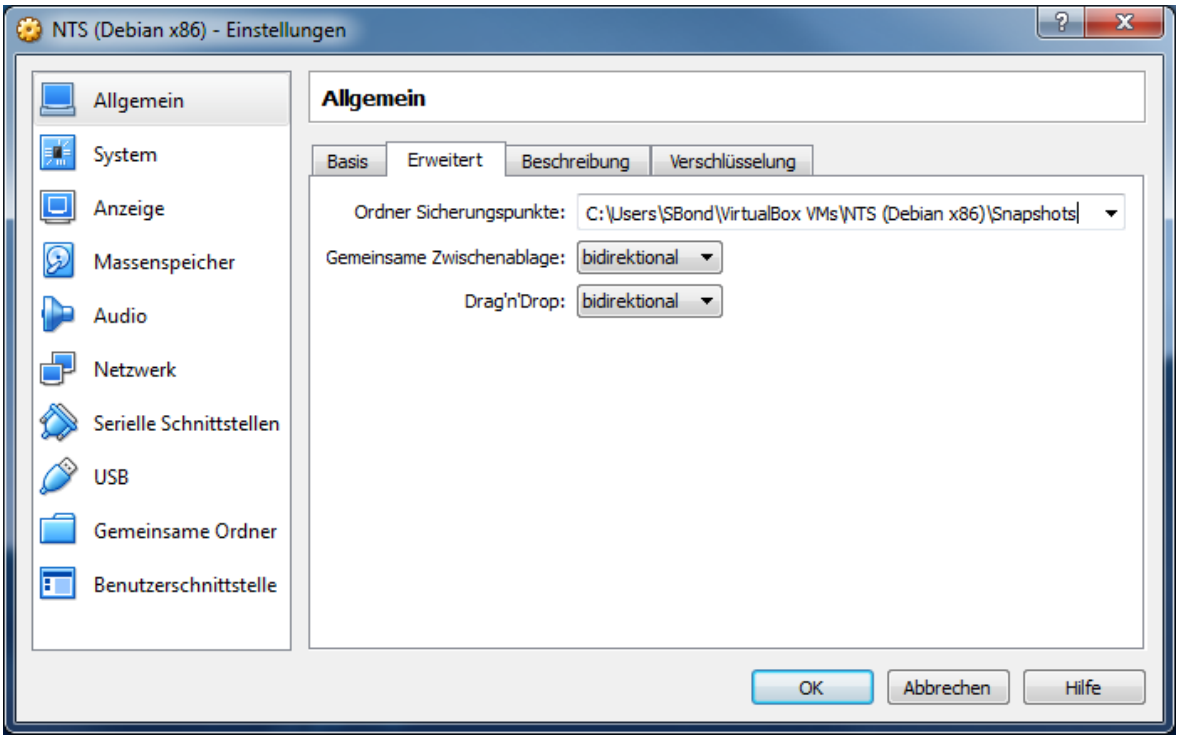


Abbildung 1.1: Grundlegende VM-Einstellungen (Allgemein)

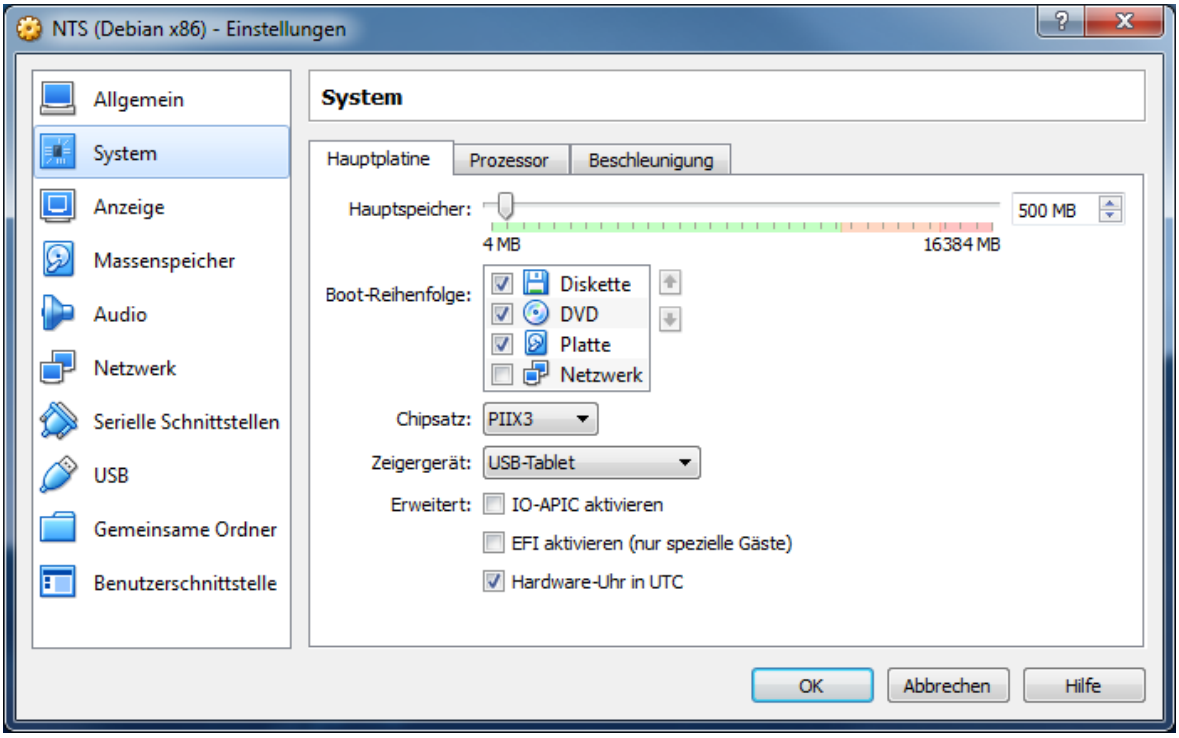


Abbildung 1.2: Grundlegende VM-Einstellungen (System 1)

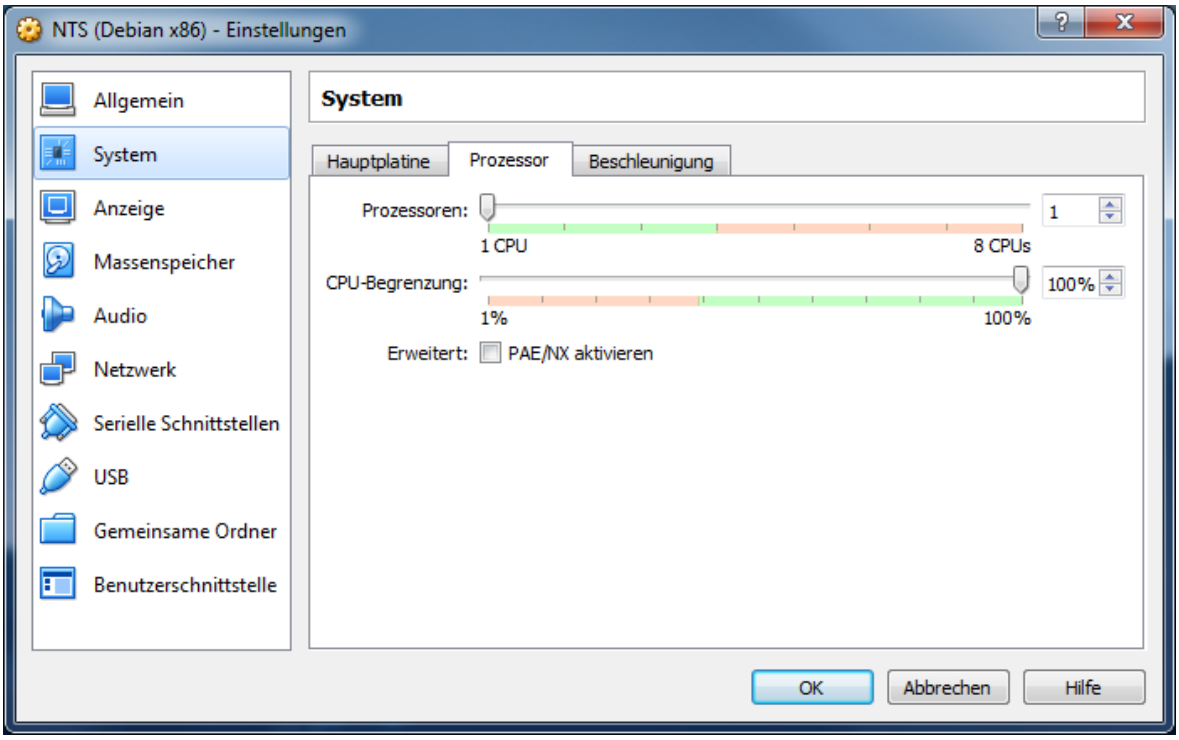


Abbildung 1.3: Grundlegende VM-Einstellungen (System2)

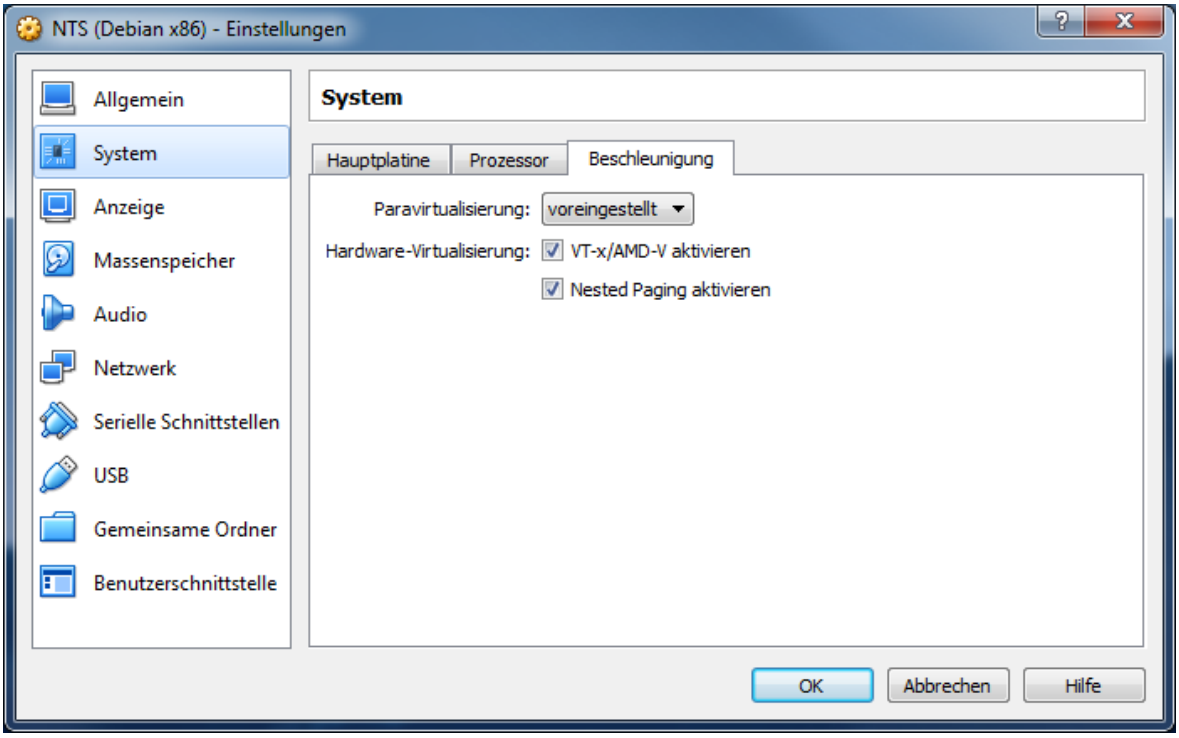


Abbildung 1.4: Grundlegende VM-Einstellungen (System 3)

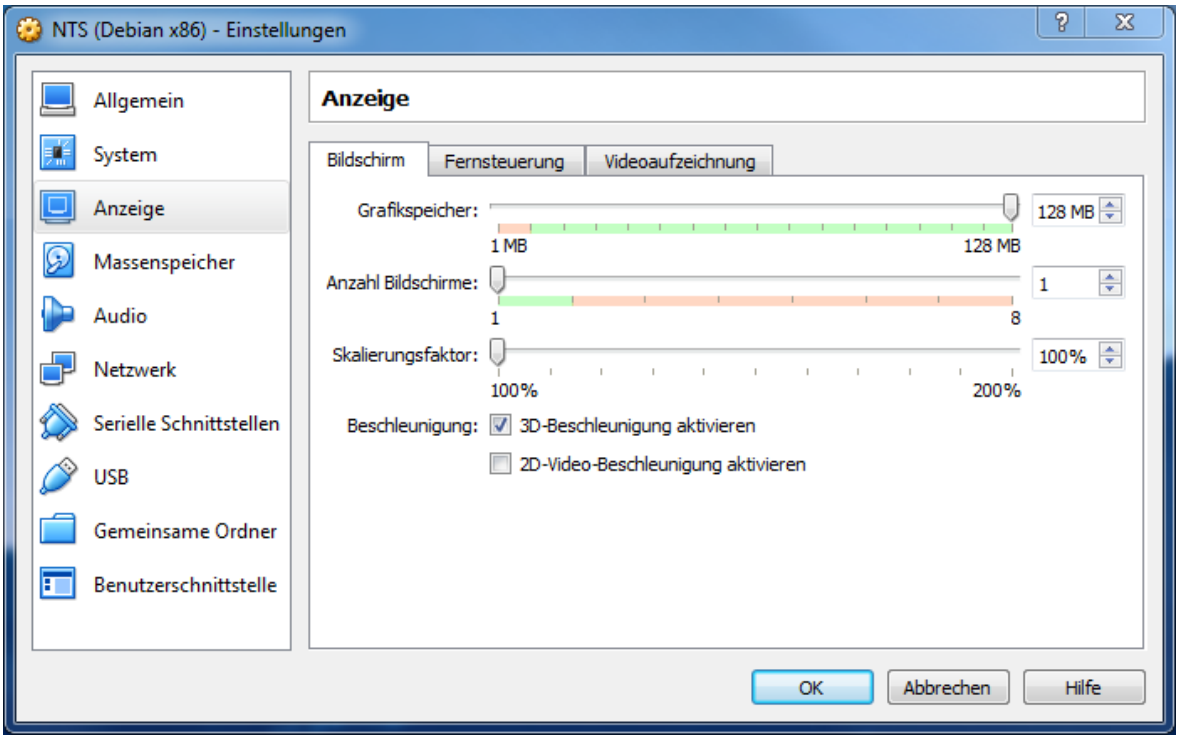


Abbildung 1.5: Grundlegende VM-Einstellungen (Anzeige)

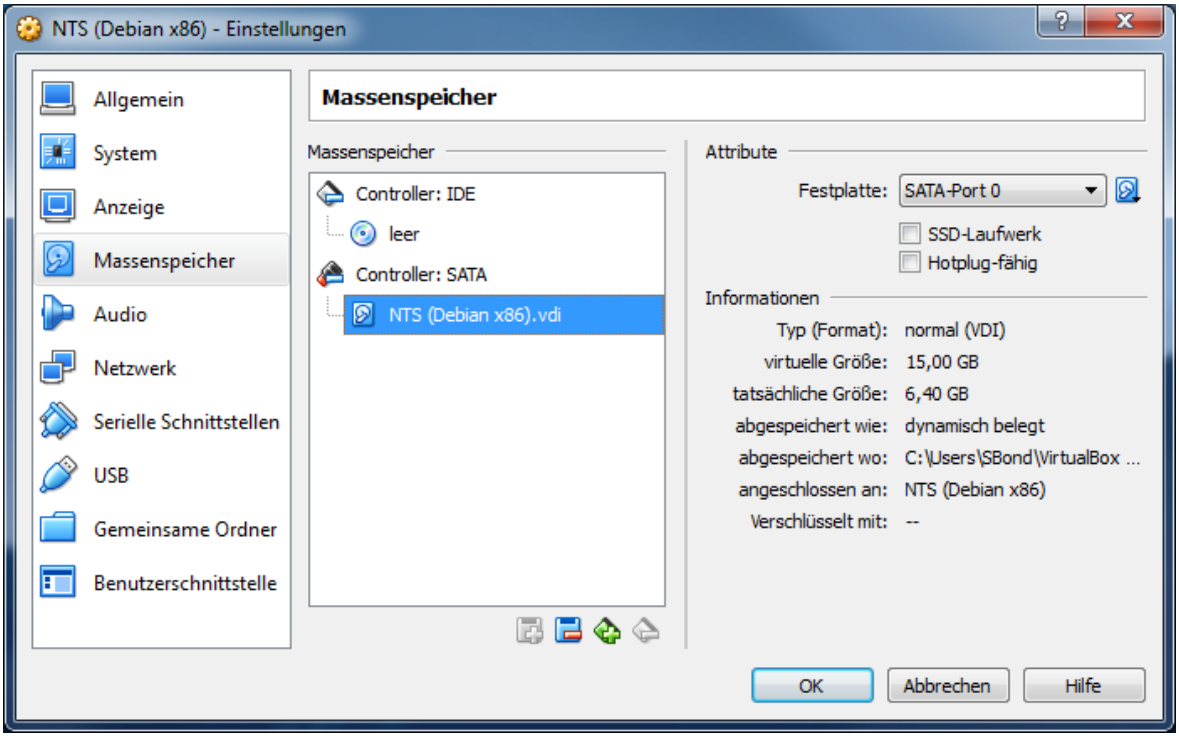


Abbildung 1.6: Grundlegende VM-Einstellungen (Massenspeicher)

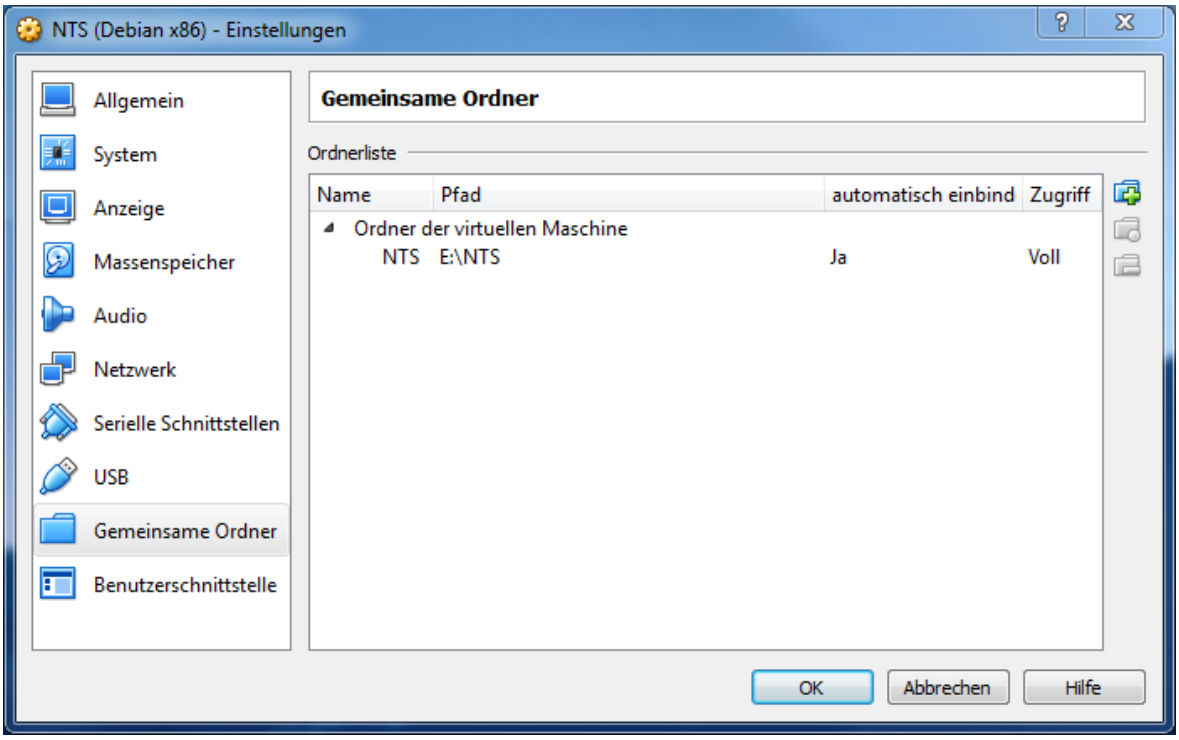


Abbildung 1.7: Grundlegende VM-Einstellungen (Gemeinsame Ordner)

1.3 Installation von Linux Debian 8.6.0 (x86)

Nach der erfolgreichen Einrichtung der virtuellen Maschine ist diese nun auszuführen. Beim ersten Start wird zunächst nach einem Datenträger verlangt, von dem gebootet werden soll. An dieser Stelle kann entweder ein physisches CD/DVD-Laufwerk oder eine Image-Datei (z.B. im .ISO-Format) übergeben werden. Liegt keine CD mit dem Linux-Betriebssystem vor, so ist die Verwendung einer Image-Datei empfehlenswert. Dazu muss die Datei *debian-8.6.0-i386-netinst.iso*[4] angegeben und mit der Schaltfläche „OK“ bestätigt werden. Die anschließende Fortführung des Startvorgangs veranlasst die VM, vom angegebenen Datenträger zu booten, worauf nun der Installationsbildschirm von Debian erscheint. Die Initialisierung der Installation wird nun über den Menüeintrag „*Graphical install*“ (vgl. Abbildung 1.8) veranlasst.

[4] Die aktuelle Version kann im Internet unter „*https://www.debian.org/distrib/netinst*“ bezogen werden. Für gängige Computersysteme ist hierbei die „i386“-Architektur zu wählen.

Abbildung 1.8: Linux-Installation: Installationsbildschirm von Linux Debian 8.6.0

Zu Beginn fordert der Installationsassistent den Benutzer zur Eingabe verschiedener Daten auf, bevor die eigentliche Übertragung des Betriebssystems auf den Zielcomputer (bzw. auf die virtuelle Maschine) erfolgt. Bei der Installation in einer VM sollten die Eingaben wie folgt aussehen:

Listing 1.2: Debian-Installation: Grundlegende Einstellungen

```
Sprache:                                        German - Deutsch
Standort:                                       Deutschland
Tastaturlayout:                                 Deutsch
Rechnername:                                    debian
Domain-Name:                                    <leer lassen>
Root-Passwort:                                  root
Vollständiger Name des Benutzers:               user
Benutername für Ihr Konto:                      user
Benutzer-Passwort:                              user
Zeitzone:                                       Berlin
```

Alternative Angaben sind problemlos möglich, müssen jedoch im Verlaufe der Installation berücksichtigt werden, da Pfade und Passwörter anschließend von der hier beschriebenen Anleitung abweichen.

Im weiteren Verlauf sammelt der Installationsassistent zusätzliche Informationen, die für die Partitionierung der Festplatte vonnöten sind. Da bei einer neu angelegten VM eine leere virtuelle Festplatte erzeugt wird, kann diese vollständig formatiert werden. Hierfür eignet sich der *geführte Modus* am besten. Das Listing 1.3 zeigt die dafür nötigen Schritte auf, die der Reihe nach durchgeführt werden müssen. Die Abbildung 1.9 zeigt die zugehörige Oberfläche bei diesem Vorgang. Die Speichergröße der angezeigten Festplatte kann variieren und ist von den in Kapitel 1.2 festgelegten Einstellungen der VM abhängig.

Listing 1.3: Debian-Installation: Festplatte partitionieren

```
➔  Geführt - vollständige Festplatte verwenden
➔  SCSI1 (0,0,0) (sda) - 16.1 GB ATA VBOX HARDDISK wählen
➔  Alle Dateien auf eine Partition, für Anfänger empfohlen
➔  Partitionierung beenden und Änderungen übernehmen
➔  Änderungen auf Festplatte schreiben?: Ja
```

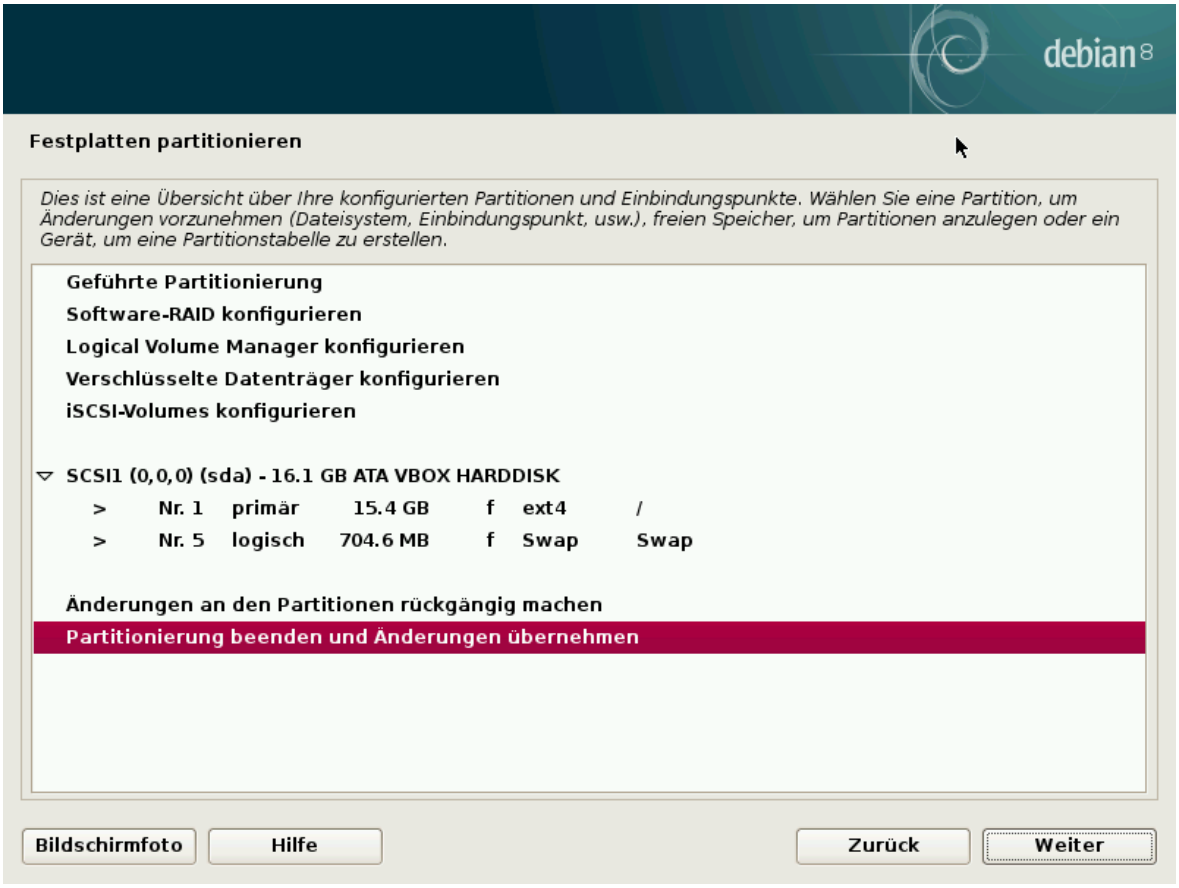


Abbildung 1.9: Linux-Installation: Festplatte partitionieren

Nach Bestätigung der Einstellungen wird nun das angegebene Laufwerk formatiert und es folgt die automatische Installation des Grundsystems. Nach diesem Vorgang folgt die Konfiguration des Paketmanagers. Aus dem angegebenen Repository bezieht das Betriebssystem zukünftig seine Programme und Systemupdates. Die Einstellungen sind gemäß Listing 1.4 zu entnehmen und in der Installationsoberfläche einzutragen. Ab dieser Stelle ist zudem die Verbindung zum Internet nötig, um die nachfolgenden Softwarekomponenten online beziehen und installieren zu können.

Listing 1.4: Debian-Installation: Paketmanager konfigurieren

```
Paketmanager konfigurieren:                          Deutschland
Debian-Archiv-Spiegelserver:                         ftp.de.debian.org
HTTP-Proxy-Daten:                                    <leer lassen>

An der Paketverwendungserfassung teilnehmen:         Nein
```

Nach Bestätigung der Daten erfolgt kurz die Konfiguration von dem in Debian enthaltenen Paketverwaltungssystem APT (*Advanced Packaging Tool*). Im Anschluss erscheint eine Auswahl an Software, die zusätzlich installiert werden kann. Das Listing 1.5 und die Abbildung 1.10 zeigen hierzu die entsprechenden Komponenten, die es auszuwählen gilt. Für die Entwicklung mit Eclipse-CDT ist die Installation der Desktopumgebung zwingend erforderlich. Die verwendete Oberfläche kann allerdings frei gewählt werden. Erfolgt die Installation in einer VM, so ist die Verwendung von *LXDE* (*Lightweight X11 Desktop Environment*) ratsam, da es schnell und energiesparend arbeitet. Auch die Installation des *Druckservers* und der *Standard-Systemwerkzeuge* ist ratsam, da sie den weiteren Verlauf der Einrichtung erleichtert. Nach Bestätigung der Auswahl durch das Klicken der „Weiter“-Schaltfläche, beginnt die Installation. Da jetzt einige hundert MegaByte heruntergeladen werden, kann die Installation je nach Internetgeschwindigkeit bis zu einer Stunde (oder auch länger) andauern. Während dieser Zeit ist die Anwesenheit am Computer nicht notwendig, da währenddessen keine weiteren Eingaben erfolgen.

Listing 1.5: Debian-Installation: Desktopumgebung und Standardsoftware

```
Softwareauswahl:                    Debian desktop environment
                                    LXDE
                                    Druckserver
                                    Standard-Systemwerkzeuge
```

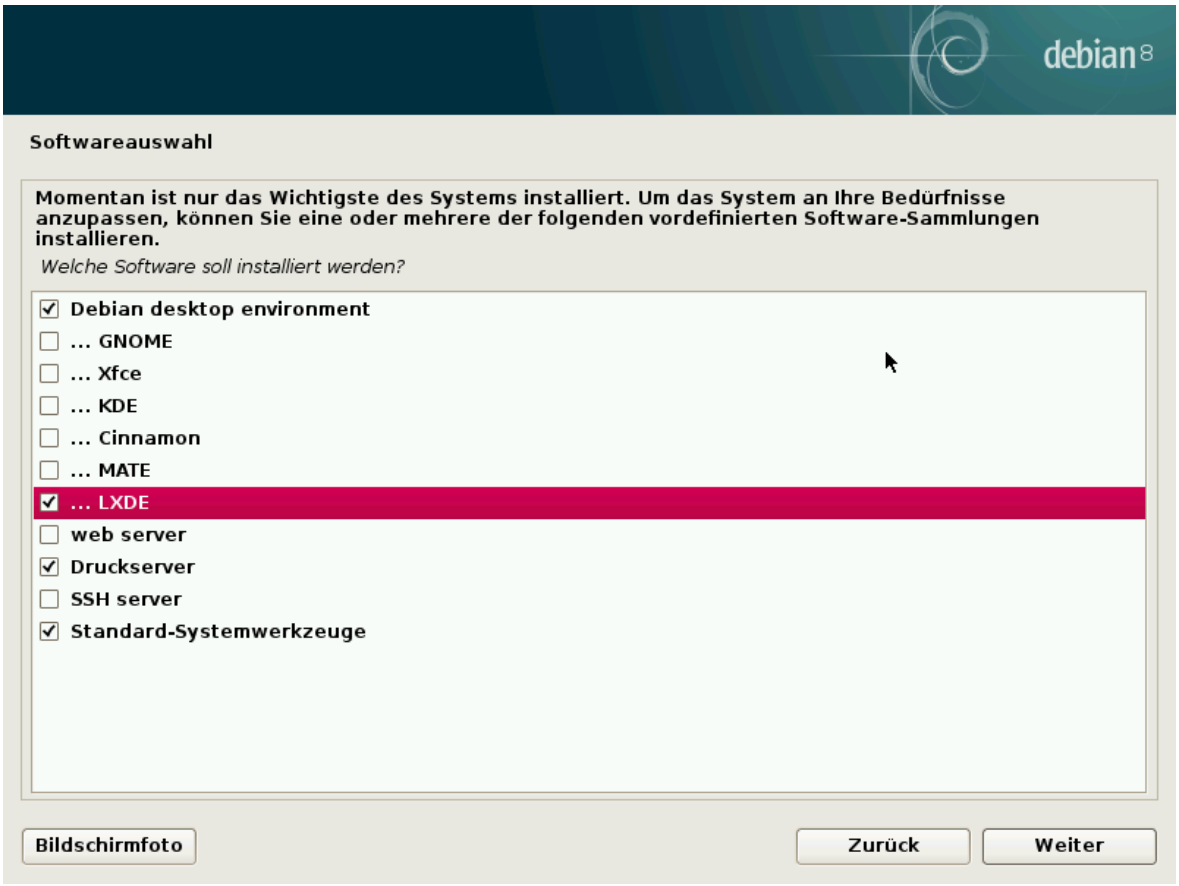


Abbildung 1.10: Linux-Installation: Wahl der Desktopumgebung

Nach erfolgreicher Installation ist abschließend der Bootloader zu konfigurieren. Die benötigten Einstellungen stellt das Listing 1.6 bereit:

Listing 1.6: Debian-Installation: GRUB-Bootloader konfigurieren

```
GRUB-Bootloader installieren:      Ja
Gerät für Bootloader-Installation: /dev/sda (ata-VBOX_HARDDISK_VBxxxxxxxxxxxx)
```

Mit der Übertragung der Konfiguration ist die Installation abgeschlossen und das System kann neu gestartet werden. Das Betriebssystem wird geladen und bootet bis zum Anmeldebildschirm. Die nun nötige Konfiguration beschreibt Abschnitt 1.4 näher.

1.4 Grundlegende Konfigurierung von Debian

Nachdem das Betriebssystem neu gestartet wurde und nun im Anmeldebildschirm verweilt, ist es Zeit, die grundlegenden Einstellungen am System vorzunehmen. Hierzu ist zunächst die Anmeldung als *Root*[5] notwendig. Das benötigte Passwort wurde zu Beginn der Installation definiert und muss nun eingegeben werden. In dieser Anleitung kommen die Daten gemäß Listing 1.7 zur Anwendung.

Listing 1.7: Als Root anmelden

```
Benutzer:   root
Paswort:    root
```

Nachdem das System hochgefahren und der Desktop sichtbar ist, ist zunächst die Konsole bzw. das Terminal zu öffnen. Hierzu sollte das *Root Terminal* verwendet werden, wie es Abbildung 1.11 dargestellt.

Abbildung 1.11: Erster Zugriff auf das Terminal

Hierdurch öffnet sich das Terminal mit Root-Rechten. Ist dies der Fall, so ist der Schriftzug „*root@debian:~#*“ im Terminal erkennbar. Falls nicht, können diese Rechte durch die Eingabe des folgenden Befehls angefordert werden:

Listing 1.8: Root-Rechte erteilen

```
sudo su
```

[5] Der *Root* hat die vollen Systemrechte, weshalb der Einsatz immer mit Bedacht erfolgen sollte.

1.4.1 Paketquellen anpassen

Nachdem das Terminal mit Root-Rechten geöffnet ist, werden die Paketquellen editiert. Dies ermöglicht den Bezug von Softwarekomponenten, die bislang noch nicht freigegeben sind. Dazu wird zunächst folgender Befehl eingegeben:

Listing 1.9: Paketquellen öffnen

```
leafpad /etc/apt/sources.list
```

Dies öffnet den Texteditor mit der aktuellen Konfiguration der Paketquellen. Der Inhalt wird nun durch den Inhalt aus Listing 1.11 ersetzt, wodurch das Ergebnis der Abbildung 1.12 entspricht. Anschließend ist die Datei unter gleichem Namen zu speichern und zu schließen.

Listing 1.10: Angepasster Inhalt der Paketquellen

```
anpassen:
--------------------------
deb http://ftp.de.debian.org/debian/ jessie main contrib non-free
deb-src http://ftp.de.debian.org/debian/ jessie main contrib non-free

deb http://security.debian.org/ jessie/updates main contrib non-free
deb-src http://security.debian.org/ jessie/updates main contrib non-free

# jessie-updates, previously known as 'volatile'
deb http://ftp.de.debian.org/debian/ jessie-updates main contrib non-free
deb-src http://ftp.de.debian.org/debian/ jessie-updates main contrib non-free
```

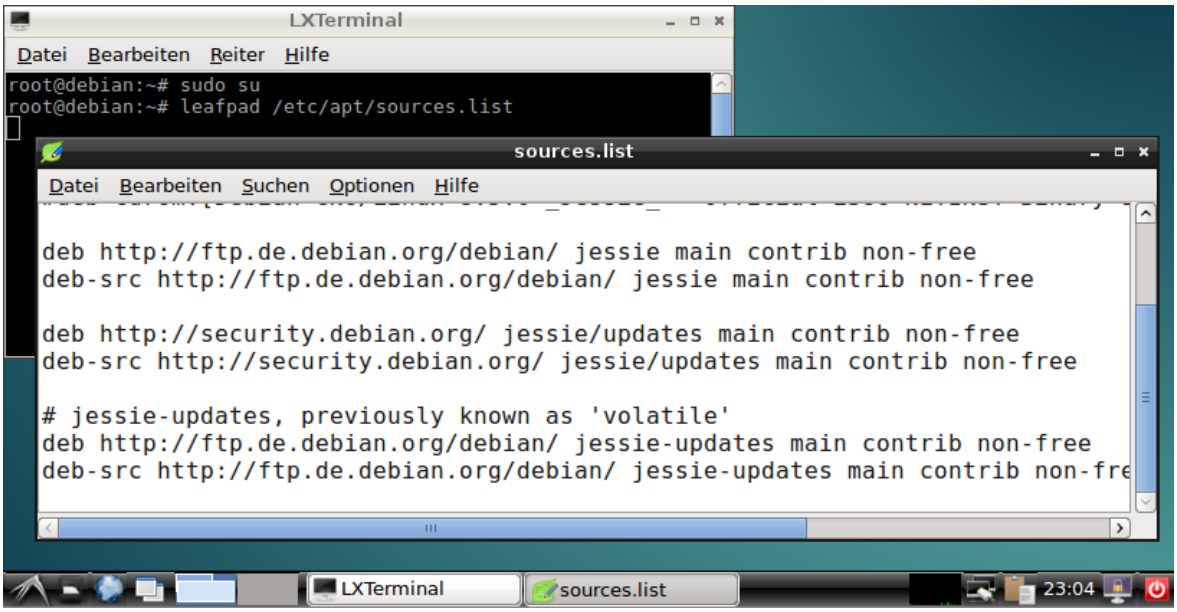


Abbildung 1.12: Bearbeitung der Paketquellen

Als nächstes ist das Betriebssystem zu aktualisieren. Hierzu sind die folgenden Befehle nacheinander im Terminal einzugeben und auszuführen. Geschieht das direkt nach der Installation von Debian, so sind möglicherweise keine Aktualisierungen vorhanden.

Listing 1.11: Aktualisierung des Betriebssystems

```
apt-get update
apt-get upgrade -y
```

Der Befehl *apt-get* veranlasst eine Operation des Paketverwaltungsprogramms. Der Parameter *update* aktualisiert hierbei nur die lokalen Paketlisten ohne Änderungen an Software vorzunehmen. Der Parameter *upgrade* gleicht anschließend die installierten Softwarekomponenten mit den Paketlisten ab. Neuere Paketversionen in den Listen werden heruntergeladen und ersetzen die veralteten Versionen. Im Anschluss ist das Betriebssystem mit allen enthaltenen Softwarekomponenten aktualisiert. Der zusätzliche Parameter „-y“ erlaubt hierbei die Installation der Softwarepakete und unterdrückt derartige Nachfragen für die Berechtigung von Softwareinstallationen durch *apt-get*. Hierbei sei noch angemerkt, dass diese Befehle eine bestehende Internetverbindung voraussetzen.

1.4.2 Die VM-Gasterweiterung installieren

In diesem Schritt steht die Optimierung des Gastsystems im Vordergrund und ist zu überspringen, wenn das Betriebssystem nicht in einer VM installiert ist. Wurde das System hingegen in einer virtuellen Maschine installiert, so sind anfänglich noch nicht alle Einstellungen der VM wirksam. Dies betrifft beispielsweise den *gemeinsamen Ordner*, die dynamische Skalierung der Gastoberfläche oder auch das Fangen und Freigeben des Mauszeigers. Um diese Defizite zu entfernen, muss zunächst die Gasterweiterung entsprechend der Abbildung 1.13 eingelegt werden.

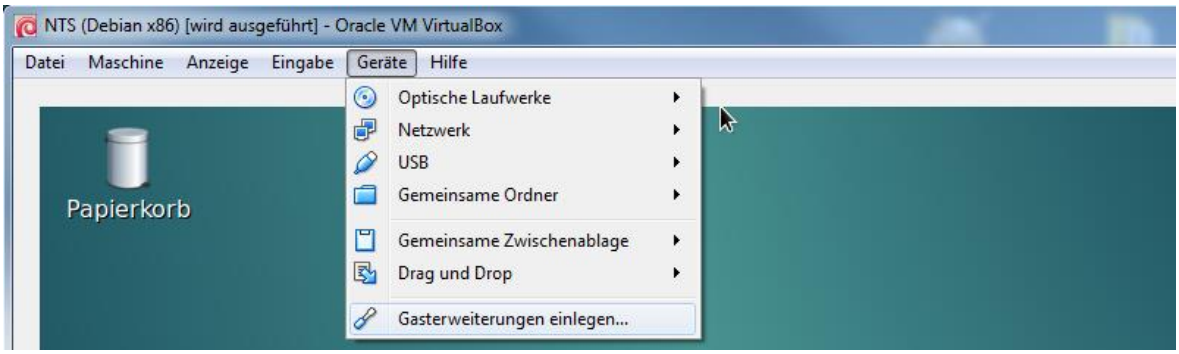


Abbildung 1.13: Gasterweiterung einlegen

Anschließend sind die folgenden Befehle im Terminal auszuführen, um die Gasterweiterung zu installieren:

Listing 1.12: Installation der Gasterweiterung

```
apt-get install -y linux-headers-$(uname -r) build-essential
sh /media/cdrom0/VBoxLinuxAdditions.run
```

Abschließend muss Debian neu gestartet werden, wodurch nun die volle Funktionalität der VM gegeben ist. Der entsprechende Befehl ist im folgenden Listing 1.13 aufgeführt:

Listing 1.13: Betriebssystem neustarten

```
reboot
```

Nach dem Neustart des Systems ist erneut die Anmeldung als Root mit den entsprechenden Benutzerdaten erforderlich:

Listing 1.14: Als Root anmelden

```
Benutzer: root
Paswort:  root
```

1.4.3 Konfiguration des GRUB-Bootloaders

Nach dem Neustart des Systems und der Anmeldung als Root ist zunächst wieder das Terminal zu öffnen. Nun ist folgender Befehl einzugeben, um die aktuellen Einstellungen des Bootloaders im Testeditor zu öffnen:

Listing 1.15: GRUB-Konfiguration öffnen

```
leafpad /etc/default/grub
```

Der Editor zeigt eine Vielzahl von Einstellungen an, die nun geändert werden können. Das Ziel dieser Anpassung ist zum einen der sofortige Start der virtuellen Maschine, ohne das Time-Out von 30 Sekunden abwarten zu müssen, und zum anderen die Erhöhung der Bildschirmauflösung zur besseren Lesbarkeit der Konsolenausgaben während des Hochfahrens. Hierzu müssen Einstellungen angepasst und hinzugefügt werden. Die jeweiligen Einträge sind aus dem Listing 1.16 zu entnehmen.

Listing 1.16: Neue Inhalte der GRUB-Konfiguration

```
# anpassen:
# ------------------
GRUB_TIMEOUT=0

# hinzufügen:
# ------------------
GRUB_CMDLINE_LINUX_DEFAULT="noplymouth"
GRUB_GFXMODE=1024x768x32
GRUB_GFXPAYLOAD_LINUX=keep
```

Nach dem Abspeichern der Änderungen ist der Editor wieder zu schließen. Danach ist im Terminal der folgende Befehl auszuführen, damit die Einstellungen übernommen werden und beim nächsten Betriebssystemneustart wirksam sind:

Listing 1.17: GRUB-Bootloader aktualisieren

```
update-grub
```

1.4.4 Benutzerrechte anpassen

Die Erhöhung der Rechte des „user“-Accounts ist optional und räumt den Benutzer mehr Freiheiten ein. Dieser Schritt kann sinnvoll sein, wenn der Root nicht der Hauptbenutzer sein soll. Hierzu ist folgender Befehl im Terminal auszuführen:

Listing 1.18: Ausführung von visudo

```
visudo
```

Unter Umständen kann es nun passieren, dass Debian eine Autorisierung verlangt und daher zur Eingabe des Root-Passworts auffordert. Ist dieses eingegeben, so wird der Vorgang fortgeführt und der entsprechende Editor öffnet sich nun innerhalb des Terminals. Die Navigation erfolgt hierbei mit den Pfeiltasten und diversen Tastenkombinationen. Als nächstes ist eine neue Zeile im Editor hinzuzufügen, deren Inhalt Listing 1.19 dargestellt. Anschließend werden mit den Tastenkombinationen *Strg+O* und *Strg+X* die Einstellungen übernommen und der Editor geschlossen.

Listing 1.19: „user" volle Rechte erteilen

```
# hinzufügen  (unter "root   ALL=(ALL:ALL) ALL"):
# ------------------------------------------------
user   ALL=(ALL)ALL

# mit Strg+O speichern und bestätigen
# mit Strg+X visudo beenden
```

1.4.5 Automatische Benutzeranmeldung

Damit nach einem Systemneustart nicht immer die Benutzerdaten eingegeben werden müssen, kann eine automatische Benutzeranmeldung eingerichtet werden. Hierzu ist die Ausführung des folgenden Befehls notwendig:

Listing 1.20: Die Konfiguration des Light Display Managers öffnen

```
leafpad /etc/lightdm/lightdm.conf
```

Dadurch öffnet sich die *LightDM*-Konfiguration im Texteditor, die es nun anzupassen gilt. Listing 1.21 zeigt die entsprechenden Einträge, die geändert werden müssen. Hierbei ist außerdem das Doppelkreuz (#) am Anfang der entsprechenden Zeile zu entfernen, sofern dieses vorhanden ist. Das Doppelkreuz stellt in der Konfiguration eine Kommentarzeile dar, wodurch einzelne Einträge deaktiviert werden können.

Listing 1.21: Die Konfiguration des Light Display Managers anpassen

```
# anpassen:
# -------------------------
autologin-user=root              → und # am Zeilenanfang entfernen
autologin-user-timeout=0         → und # am Zeilenanfang entfernen
```

1.5 Installation der benötigten Software

Nach der grundlegenden Anpassung des Betriebssystems ist es Zeit, die restliche Software zu installieren und darüber hinaus unnötige Komponenten zu entfernen. Die entsprechenden Schritte beschreiben die folgenden Unterkapitel näher.

1.5.1 Deinstallation nicht benötigter Softwarekomponenten

Ist die Linux-Installation in einer virtuellen Maschine mit einer kleinen virtuellen Festplatte (<12 GB) erfolgt, so ist es sinnvoll, unnötige Softwarekomponenten zu entfernen, um dadurch belegten Speicherplatz freizugeben. Listing 1.22 listet hierzu die Befehle auf, die der Reihe nach einzugeben und auszuführen sind. Das Doppelkreuz markiert den nachfolgenden Kommentar und kann in einer Befehlszeile enthalten sein.

Listing 1.22: Zu löschende Software

```
apt-get remove -y --purge libreoffice*  # Office-Paket
apt-get remove -y gimp                  # Grafikbearbeitungsprogramm
apt-get remove -y xsane                 # Hilfsprogramm für Scanner
apt-get remove -y deluge                # BitTorrent-Client
apt-get remove -y deluge-webui          # Web-Oberfläche für deluge
apt-get clean
apt-get -y autoremove
```

1.5.2 Installation benötigter Softwarekomponenten

Als nächstes ist die Software zu installieren, die für die Programmierung wichtig ist und bei der Entwicklung von NTS zum Einsatz kommt.

1.5.2.1 Installation grundlegender Werkzeuge

Der erste Schritt ist die Installation grundlegender Programme und Werkzeuge. Diese werden zwingend benötigt, um die Installation der nachfolgenden Software zu ermöglichen. Dazu sind die im Listing 1.23 aufgeführten Befehle der Reihe nach auszuführen.

Listing 1.23: Installation grundlegender Werkzeuge

```
apt-get install -y openjdk-7-jre          # Java Runtime Environment
apt-get install -y gdb                    # Quelltext-Debugger
apt-get install -y g++                    # C++ Compiler
apt-get install -y make                   # Build-Management-Tool
apt-get install -y valgrind               # Profiler / Analysetool
apt-get install -y libwebkitgtk-1.0-0     # interner Webbrowser für Eclipse
apt-get install -y linux-tools-3.16       # Profiler Tools
```

1.5.2.2 ASN.1-Bibliothek (asn1c v0.9.27)

Nachdem die grundlegenden Werkzeuge installiert sind, ist die Installation der ASN.1-Bibliothek möglich. Dazu können alle Einträge im Listing 1.24 markiert und kopiert werden. Anschließend ist das Einfügen der Befehle über das Kontextmenü des Terminals möglich. Dieses öffnet sich durch einen einfachen Rechtsklick mit der Maus innerhalb des Terminals. Durch diese Maßnahme werden alle Befehle automatisch eingefügt und nacheinander ausgeführt. Die roten Zeilen stellen hierbei Umbrüche dar, die aufgrund der begrenzten Zeilenlänge erfolgen. Beim Kopieren werden diese Zeilen automatisch korrekt zusammengeführt.

Listing 1.24: Installation der ASN.1-Bibliothek

```
cd /root/Dokumente
wget https://github.com/vlm/asn1c/releases/download/v0.9.27/asn1c-
0.9.27.tar.gz
tar xfz asn1c-0.9.27.tar.gz
cd asn1c-0.9.27
./configure
make
make install
cd /root/Dokumente
rm /root/Dokumente/asn1c-0.9.27.tar.gz
rm -r /root/Dokumente/asn1c-0.9.27
asn1c -version
```

1.5.2.3 Doxygen (v1.8.13)

In diesem Schritt erfolgt die Installation von *Doxygen*. Hierbei handelt es sich um ein Programm, mit dem die Dokumentation des Quellcodes verbessert wird. Für die Installation wird die aktuelle Doxygen-Version direkt aus dem Git-Repository bezogen. Daher sind vorab noch einige Werkzeuge zu installieren, um den anschließend heruntergeladenen Quellcode kompilieren zu können. Folgende Befehle sind der Reihe nach auszuführen:

Listing 1.25: Installation der Werkzeuge für Doxygen

```
apt-get install -y git       # Versionsverwaltungsprogramm
apt-get install -y cmake     # Build-System für Softwareentwickler
apt-get install -y flex      # Parsergenerator; wird zur Installation benötigt
apt-get install -y bison     # Parsergenerator; wird zur Installation benötigt
apt-get install -y graphviz  # Erstellung von Beziehungsdiagrammen in Doxygen
apt-get install -y dot2tex   # Graphviz-Konverter für LaTeX
```

An dieser Stelle sei noch angemerkt, dass das gleichzeitige Kopieren aller *apt-get*-Befehle und das Einfügen im Terminal nicht korrekt funktionieren und häufig zu Fehlern führt. Daher sollte jeder Befehl einzeln kopiert werden, um mögliche Probleme vorzubeugen. Nach Abschluss des Vorgangs ist nun Doxygen zu installieren. Die folgenden Befehle können wieder als Ganzes kopiert und im Terminal eingefügt werden:

Listing 1.26: Installation von Doxygen

```
cd /root/Dokumente
git clone https://github.com/doxygen/doxygen.git
cd doxygen
mkdir build
cd build
cmake -G "Unix Makefiles" ..
make
make install
cd /root/Dokumente
rm -r /root/Dokumente/doxygen
doxygen -v
```

1.5.2.4 OpenSSL (v1.1.0b)

Die Installation von OpenSSL in der benötigten Version ist über das Paketverwaltungsprogramm nicht ohne weiteres möglich, da hier generell ältere – und somit als stabil geltende Versionen – bevorzugt werden. Die Einrichtung erfolgt somit manuell. Dazu muss zunächst *zlib* installiert werden, da es während der Kompilierung von OpenSSL benötigt wird. Bei *zlib* handelt es sich um eine freie Bibliothek zur Komprimierung und Dekomprimierung von Daten. Die Installation erfolgt erneut durch das Kopieren-Einfügen-Schema der folgenden Befehle:

Listing 1.27: Installation von "zlib" für OpenSSL

```
cd /root/Dokumente
wget http://zlib.net/zlib-1.2.8.tar.gz
tar xzf zlib-1.2.8.tar.gz
cd zlib-1.2.8
./configure --prefix=/usr --shared
make
make install
ldconfig
cd /root/Dokumente
rm /root/Dokumente/zlib-1.2.8.tar.gz
rm -r /root/Dokumente/zlib-1.2.8
```

War dieser Vorgang erfolgreich, so wird nach der gleichen Methode OpenSSL installiert. Dazu ist erneut das Kopieren-Einfügen-Schema der folgenden Befehle anzuwenden:

Listing 1.28: Installation von OpenSSL

```
cd /root/Dokumente
wget https://www.openssl.org/source/openssl-1.1.0b.tar.gz
tar xfz openssl-1.1.0b.tar.gz
cd openssl-1.1.0b
./config --prefix=/usr zlib-dynamic --openssldir=/etc/ssl shared
make
make install
cd /root/Dokumente
rm /root/Dokumente/openssl-1.1.0b.tar.gz
rm -r /root/Dokumente/openssl-1.1.0b
openssl version
```

Je nach Geschwindigkeit des Computers kann dieser Vorgang mehrere Minuten in Anspruch nehmen. Bei erfolgreicher Installation der Software ist in der letzten Zeile des Terminals der Eintrag „*OpenSSL 1.1.0b 26 Sep 2016*“ sichtbar.

1.5.2.5 Eclipse-CDT (Luna v4.4)

Als nächstes folgt die Installation der eigentlichen Entwicklungsumgebung, in welcher die Programme in C++ geschrieben werden. Hierzu kommt das Programm Eclipse-CDT in der älteren Version 4.4 zum Einsatz. Die ältere Version arbeitet in einer VM deutlich schneller, weshalb sich diese Version am besten eignet. In einer nativen Linux-Umgebung kann allerdings auch die aktuelle Version genutzt werden. Die Installation des Programms geschieht wieder nach dem Kopieren-Einfügen-Schema der folgenden Befehle:

Listing 1.29: Installation von Eclipse-CDT

```
cd /root/Dokumente
wget http://ftp.halifax.rwth-
aachen.de/eclipse//technology/epp/downloads/release/luna/SR2/eclipse-cpp-luna-
SR2-linux-gtk.tar.gz
tar xfz eclipse-cpp-luna-SR2-linux-gtk.tar.gz
rm eclipse-cpp-luna-SR2-linux-gtk.tar.gz
mv /root/Dokumente/eclipse /opt/eclipse
```

Die rot markierten Zeilen stellen erneut Zeilenumbrüche dar, die jedoch keinen Einfluss auf das Kopieren mittels *Strg+C* haben.

1.6 Einrichtung der Software

Nachdem alle Softwarekomponenten installiert sind, erscheint der Desktop nach wie vor unverändert (vgl. Abbildung 1.14). In den folgenden Abschnitten werden diverse Veränderungen vorgenommen, mit denen die weitere Bedienung vereinfacht wird.

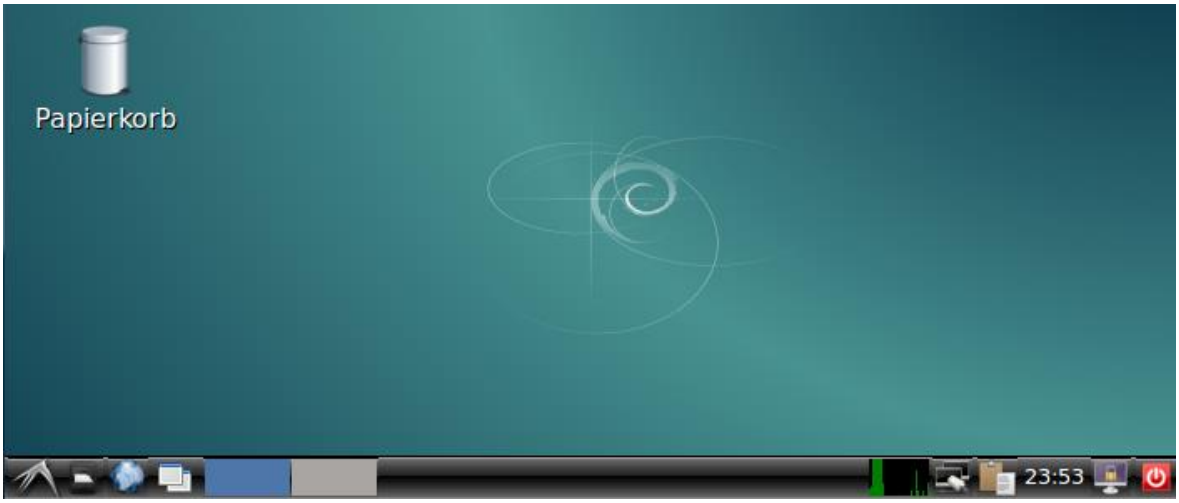


Abbildung 1.14: Start von Debian nach Installation der Software

1.6.1 Einrichten von Eclipse-CDT

Zunächst ist die Eclipse-Programmierumgebung einzurichten, da diese noch nicht funktionsfähig ist. Hierzu sind einige Schritte notwendig, um die Nutzung des Programms zu ermöglichen.

1.6.1.1 Rechte zum Ausführen erteilen

Der erste Schritt ist das Setzen von Rechten bzw. Eigenschaften. Der folgende Befehl ermöglicht dazu die zukünftige Ausführung von Eclipse-CDT.

Listing 1.30: Eclipse die Rechte zum Ausführen erteilen

```
sudo chmod +x /opt/eclipse/eclipse
```

1.6.1.2 Eclipse-CDT als Konsolenbefehl hinzufügen

Im nächsten Schritt wird Eclipse als Konsolenbefehl hinzugefügt. Dies ermöglicht fortan die Ausführung von Eclipse-CDT durch den „eclipse“-Befehl im Terminal. Um dies zu erreichen ist folgendes auszuführen:

Listing 1.31: Eclipse als Konsolenbefehl hinzufügen

```
sudo ln -s /opt/eclipse/eclipse /usr/local/bin
```

1.6.1.3 Eclipse-CDT im Startmenü eintragen

Auch das Hinzufügen einer Eclipse-Verknüpfung im Startmenü ist sinnvoll, um die Ausführung ohne Terminal zu realisieren. Hierzu ist folgender Befehl einzugeben:

Listing 1.32: Eclipse zum Startmenü hinzufügen

```
leafpad /usr/share/applications/eclipse.desktop
```

Nach der Ausführung öffnet sich ein leerer Texteditor, da die angegebene Datei noch nicht existent ist. Im Editor ist nun der Inhalt aus dem Listing 1.33 einzufügen. Anschließend ist zu speichern und der Editor zu schließen.

Listing 1.33: Inhalt der "eclipse.desktop"-Datei

```
[Desktop Entry]
Version=1.0
Name=Eclipse (Luna)
Comment=Eclipse Integrated Development Environment
Exec=/opt/eclipse/eclipse
Icon=/opt/eclipse/icon.xpm
Terminal=false
Type=Application
Categories=Development;IDE;
```

Von nun an steht im Startmenü in der Kategorie *Entwicklung* der Eintrag *Eclipse (Luna)* zur Verfügung.

1.6.1.4 Workspace-Pfad setzen

Der Workspace-Ordner ist der allgemeine Arbeitspfad von Eclipse, in dem sich Projekte und Einstellungen befinden. Dieser sollte im *gemeinsamen Ordner* der VM angelegt werden, damit die Projekte von der VM ausgekoppelt sind. Das entsprechende Verzeichnis befindet sich im Systemverzeichnis „/media/“ und setzt sich aus der Bezeichnung „sf_“ und dem Namen des gemeinsamen Ordners zusammen. Im Falle dieser Anleitung ergibt sich der Pfad „/media/sf_NTS/“. Nach dem Start von Eclipse erfolgt eine Abfrage zur Festlegung des Workspace-Ordners. Dieser wird nun wie folgt definiert:

Listing 1.34: Eclipse-Workspace-Pfad definieren

```
/media/sf_NTS/Workspace
```

1.6.1.5 Eclipse-Plugin installieren: eclox

Nachdem Eclipse gestartet ist, sollte außerdem das *eclox*-Plugin installiert werden. Dieses ermöglicht die Verwendung von Doxygen innerhalb von Eclipse, in dem es entsprechende Schaltflächen bereitstellt und die Syntaxhervorhebung ermöglicht. Die Installation von *eclox* erfolgt gemäß Listing 1.35 innerhalb der Eclipse-Anwendung:

Listing 1.35: Installation des Eclipse-Plugins: eclox

```
➔ Eclipse-Menüeintrag: Help ➔ Install New Software...
➔ "Add..." klicken
➔ Eingabemaske ausfüllen:
        Name:       http://download.gna.org/eclox/update
        Location:   http://download.gna.org/eclox/update

➔ "eclox" in der Liste auswählen und "Next" klicken
➔ Lizenz akzeptieren und installieren
```

1.6.1.6 Eclipse-Plugin installieren: LogViewer

Das zweite Plugin ist der *LogViewer*, womit sich Inhalte von Dateien einlesen und ausgeben lassen. Somit können diverse Fehlermeldungen von der internen Eclipse-Konsole getrennt und in Log-Dateien ausgelagert werden. Hierzu sind folgende Schritte nötig, um das Plugin zu installieren:

Listing 1.36: Installation des Eclipse-Plugins: LogViewer

```
➔ Eclipse-Menüeintrag: Help → Eclipse Marketplace...

➔ " LogViewer " suchen und installieren
```

1.6.2 Anpassen der Debian-Oberfläche

In diesem Abschnitt geht es um optionale Anpassungen der Debian-Oberfläche. Das Ziel ist die bessere Bedienung und die Abschaltung störender Einflüsse.

1.6.2.1 Deaktivierung des Bildschirmschoners

Gerade beim Arbeiten in der VM kann der Debian-interne Bildschirmschoner störend sein. Dieser färbt die VM schwarz ein, sobald diese mehrere Minuten inaktiv ist. Das Verhalten ist unabhängig von den Einstellungen des Host-Systems und sollte daher deaktiviert werden. Dazu muss zunächst die Keyboard-Konfiguration wie folgt geöffnet werden:

Listing 1.37: Keyboard-Konfiguration öffnen

```
leafpad /etc/kbd/config
```

Im Texteditor sind nun die folgenden Zeilen zu suchen und entsprechend anzupassen:

Listing 1.38: Angepasste kbd-Konfiguration

```
# anpassen:
# -------------------------
BLANK_TIME=0
POWERDOWN_TIME=0
```

Anschließend wird die Datei gespeichert und der Editor ist zu schließen. Als nächstes wird der dazugehörige *kbd*-Dienst neu gestartet, damit die Änderungen wirksam werden. Dazu ist der folgende Befehl im Terminal auszuführen:

Listing 1.39: Den kbd-Dienst neustarten

```
/etc/init.d/kbd restart
```

Nach dem Neustart des Dienstes ist das Schwarzfärben des Bildschirminhalts auf Debian-Ebene deaktiviert. Allerdings liefert die Desktopumgebung auch einen Bildschirmschoner, der ebenfalls deaktiviert werden muss. Dazu ist zunächst die *LightDM*-Konfiguration wie folgt zu öffnen:

Listing 1.40: Die Konfiguration des Light Display Managers öffnen

```
leafpad /etc/lightdm/lightdm.conf
```

Danach öffnet sich erneut der Texteditor mit dem entsprechenden Inhalt. Nun muss der folgende Eintrag gesucht und entsprechend angepasst werden:

Listing 1.41: Die Konfiguration des Light Display Managers anpassen

```
# anpassen:
# -------------------------
[SeatDefault]
xserver-command=X -s 0 dpms            → # am Zeilenanfang entfernen
```

Ist vor dem Eintrag ein Doppelkreuz vorhanden, so ist dieses zusätzlich zu entfernen. Anschließend ist der Inhalt zu speichern und der Editor zu schließen.

1.6.2.2 Erstellen von zusätzlichen Desktop-Verknüpfungen

Das Hinzufügen weiterer Desktop-Verknüpfungen erleichtert die Bedienung und ermöglicht den schnelleren Zugriff auf unterschiedliche Ressourcen. Ist eine Eclipse-Verknüpfung erwünscht, so ist dies direkt über das Startmenü realisierbar. Folgende Schritte sind durchzuführen:

Listing 1.42: Eclipse-Desktop-Verknüpfung erstellen

```
    → Start --> Entwicklung → Eclipse (Luna)
    → Rechtsklick → "Dem Desktop hinzufügen"
```

Auch eine Verknüpfung zum Workspace-Verzeichnis bzw. zum *gemeinsamen Ordn*er der VM ist sinnvoll, um einen schnellen Zugriff zu erhalten. Dies muss über den Dateimanager erfolgen der daher mit folgendem Befehl auszuführen ist:

Listing 1.43: Öffnen des Dateimanagers im angegebenen Pfad

```
pcmanfm /media
```

Dadurch öffnet sich der Dateimanager direkt im angegebenen Verzeichnis (*/media/*). Nun sind die folgenden Schritte durchzuführen, um eine Desktop-Verknüpfung zu erhalten:

Listing 1.44: Desktop-Verknüpfung für den Worksapce-Ordner erstellen

```
➔ 'sf_NTS' markieren
➔ Menüeintrag: Bearbeiten → "Neue Verknüpfung"
➔ Desktop auswählen → "OK"
```

Zusätzlich können durch folgende Einstellung versteckte Dateien und Ordner angezeigt werden, sofern dies gewünscht ist:

Listing 1.45: Versteckte Dateien und Ordner anzeigen

```
➔ Menüeintrag: Ansicht → "Versteckte anzeigen"
➔ Eintrag aktivieren
```

1.6.2.3 Konfiguration der Taskleiste (*LXPanel*)

Das *LXPanel* ist das Äquivalent zur Taskleiste in *Microsoft Windows*, lässt sich jedoch weitaus flexibler anpassen. Die folgenden Einstellungen fügen hierzu diverse Menüpunkte und Sensorwerte hinzu, um beispielsweise die Ressourcenauslastung (CPU und RAM) der VM beobachten zu können. Dazu wird zunächst der folgende Befehl im Terminal eingegeben, worauf sich der Texteditor mit dem entsprechenden Inhalt öffnet.

Listing 1.46: Öffnen der LXPanel-Einstellungen

```
leafpad /root/.config/lxpanel/LXDE/panels/panel
```

Der komplette Inhalt (sofern vorhanden) wird nun ersetzt. Dazu werden die folgenden Inhalte (Listing 1.47 bis Listing 1.49) nacheinander in den Editor eingefügt. Anschließend wird dieser gespeichert und geschlossen.

Listing 1.47: Neuer Inhalt der LXPanel-Konfiguration - Part 1

```
# lxpanel <profile> config file. Manually editing is not recommended.
# Use preference dialog in lxpanel to adjust config when you can.

Global {
  edge=bottom
  allign=left
  margin=0
  widthtype=percent
  width=100
  height=25
  transparent=0
  tintcolor=#000000
  alpha=0
  setdocktype=1
  setpartialstrut=1
  usefontcolor=1
  fontcolor=#ffffff
  background=1
  backgroundfile=/usr/share/lxpanel/images/background.png
}
Plugin {
  type=space
  Config {
       Size=2
  }
}
Plugin {
  type=menu
  Config {
       image=/usr/share/lxde/images/lxde-icon.png
       system {
       }
       separator {
       }
       item {
         command=run
       }
       separator {
       }
       item {
         image=gnome-logout
         command=logout
       }
  }
}
Plugin {
  type=launchbar
  Config {
       Button {
         id=pcmanfm.desktop
       }
       Button {
         id=menu://applications/System/lxterminal.desktop
       }
       Button {
         id=menu://applications/Internet/iceweasel.desktop
       }
       Button {
         id=menu://applications/Development/eclipse.desktop
       }
  }
}
```

Listing 1.48: Neuer Inhalt der LXPanel-Konfiguration - Part 2

```
Plugin {
  type=space
  Config {
       Size=4
  }
}
Plugin {
  type=wincmd
  Config {
       Button1=iconify
       Button2=shade
  }
}
Plugin {
  type=space
  Config {
       Size=4
  }
}
Plugin {
  type=pager
  Config {
  }
}
Plugin {
  type=space
  Config {
       Size=4
  }
}
Plugin {
  type=taskbar
  expand=1
  Config {
       tooltips=1
       IconsOnly=0
       AcceptSkipPager=1
       ShowIconified=1
       ShowMapped=1
       ShowAllDesks=0
       UseMouseWheel=1
       UseUrgencyHint=1
       FlatButton=0
       MaxTaskWidth=150
       spacing=1
  }
}
Plugin {
  type=kbled
  Config {
       ShowCapsLock=1
       ShowNumLock=1
       ShowScrollLock=1
  }
}
Plugin {
  type=volumealsa
  Config {
  }
}
```

Listing 1.49: Neuer Inhalt der LXPanel-Konfiguration - Part 3

```
Plugin {
  type=monitors
  Config {
       DisplayCPU=1
       DisplayRAM=1
       CPUColor=#FFAA00
       RAMColor=#00AAFF
  }
}
Plugin {
  type=tray
  Config {
  }
}
Plugin {
  type=dclock
  Config {
       ClockFmt=%R
       TooltipFmt=%A %x
       BoldFont=0
       IconOnly=0
       CenterText=0
  }
}
```

Nun muss das Betriebssystem durch folgenden Befehl gestartet werden, damit alle Änderungen wirksam werden:

Listing 1.50: Betriebssystem neustarten

```
reboot
```

Nach dem Neustart des Systems ist die Einrichtung abgeschlossen und der Desktop präsentiert das neue Aussehen des *LXPanels* (vgl. Abbildung 1.15).

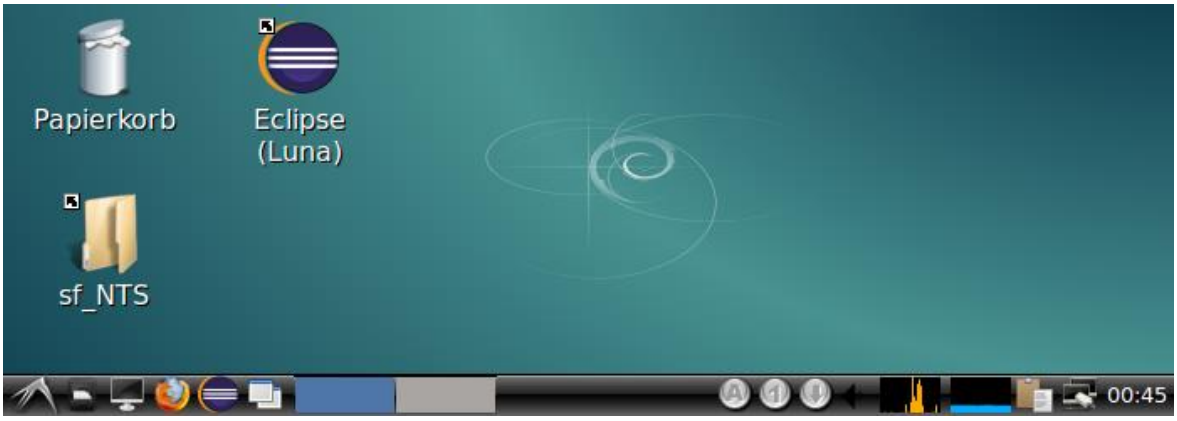


Abbildung 1.15: Abgeschlossene Einrichtung von Debian

2 Einrichten von Eclipse-Projekten

Dieser Abschnitt gibt einige Tipps oder Hilfestellungen bezüglich neu erstellter C++-Projekte in Eclipse.

2.1.1 C++ 11 aktivieren

Wird ein neues C++-Projekt erstellt, so basiert dieses aufgrund des verwendeten Compilers (g++) zunächst auf dem C++ 03-Standard. Da der neuere Standard C++ 11 allerdings einen größeren Funktionsumfang und Verbesserungen liefert, ist es sinnvoll, auch diesen Standard zu nutzen. Dafür muss lediglich ein Eintrag in den Einstellungen wie folgt vorgenommen werden:

Listing 2.1: Aktivierung der C++ 11-Syntax

```
➔ Eclipse-Menüeintrag: Windows → Preferences
➔ Preferences: C/C++ → Build → Settings
➔ Reiter: Discovery
➔ Eintrag: "CDT GCC Built-in Compiler Settings [ Shared ]"
➔ Eintrag wie folgt ändern:
${COMMAND} ${FLAGS} -E -P -v -dD -std=c++11 "${INPUTS}"
```

2.1.2 OpenSSL in Eclipse-Projekte nutzen

Auch um die OpenSSL-Bibliothek nutzen zu können, sind einige Anpassungen an das jeweilige Projekt vorzunehmen. Hierzu sind folgende Schritte durchzuführen:

Listing 2.2: Nutzung der OpenSSL-Bibliothek in Projekten

```
➔ Eclipse-Menüeintrag: Project → Properties
➔ Properties : C/C++ Build → Settings
➔ Eintrag: GCC C++ Linker → Miscellaneous settings
➔ Feld: „All options:"
➔ foglendes hinzufügen:
 "-lcrypto -lssl"
```

Darüber hinaus kann es erforderlich sein, benutzerdefinierte Pfade in das Projekt zu integrieren. Beispiele für sinnvolle Pfade stellt Listing 2.3 dar:

Listing 2.3: Hinzufügen benutzerdefinierter Pfade in Eclipse-Projekten

```
➔ Eclipse-Menüeintrag: Project → Properties

➔ Properties: C/C++ General → Paths and Symbols

➔ mögliche Pfade, die in den entsprechenden Bereichen hinzuzufügen sind:

Symbole:  /usr/include
Libs:     /usr/lib
```

Ostfalia
Hochschule für angewandte Wissenschaften

Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb

Anlage II:

ASN.1-Modulbeschreibungen in der NTS-Implementierung

Martin Langer
Matrikelnummer: 70405974
Eingereicht am: 21.10.2016

Ostfalia, Wolfenbüttel
Fakultät Elektrotechnik
Studiengang: Intelligente Mobilität und Energiesystemtechnik

Dokumentversion: 1.0.0.43
Letze Änderung: 18.10.2016 23:22:00

Inhaltsverzeichnis

1 Informationen zu den ASN.1-Strukturen 1

2 Übersicht zu den verwendeten Strukturen 2

2.1 Network Time Security (alle Draft-Dokumente) 2
2.2 Cryptographic Message Syntax (RFC 5652) 2
2.3 PKIX Certificate and CRL Profile (RFC 5280) 3
2.4 AC Profile for Authorization (RFC 5755) 4
2.5 Authentication Framework (X.509-1988) 4
2.6 Authentication Framework (X.509-2000) 4

3 Die NTS-Nachrichtenobjekte 5

3.1 Vereinfachte Modulbeschreibung 5
3.2 Vollständige Modulbeschreibung 6

4 Komplette ASN.1-Beschreibung für NTS 13

4.1 Network Time Security (alle Draft-Dokumente) 13
4.2 Cryptographic Message Syntax (RFC 5652) 14
4.3 PKIX Certificate and CRL Profile (RFC 5280) 20
4.4 AC Profile for Authorization (RFC 5755) 23
4.5 Authentication Framework (X.509-1988) 24
4.6 Authentication Framework (X.509-2000) 24

1 Informationen zu den ASN.1-Strukturen

Dieses Dokument beinhaltet sämtliche ASN.1-Strukturen, die in NTS zur Anwendung kommen. Dabei handelt es sich um die kompletten Beschreibungen der NTS-Nachrichtenobjekte, der verwendeten CMS-Strukturen und den X.509-Zertifikaten. Zum aktuellen Stand kommen 135 verschiedene ASN.1-Strukturen bzw. Elemente zum Einsatz und erreichen Verschachtelungstiefen von bis zu 22 Ebenen.

Kapitel 2 listet hierzu die verwendeten Strukturnamen in alphabetisch sortierter Reihenfolge auf. Die Überschriften der enthaltenen Unterkapitel stellen hierbei die Spezifikationen dar, in denen die entsprechenden ASN.1-Module definiert sind.

Kapitel 3 beinhaltet die Modulbeschreibung der NTS-Nachrichtenobjekte in zwei verschiedenen Formen. Die *vereinfachte Modulbeschreibung* enthält alle Strukturen in der NTS-Unicast-Kommunikation, die auch für Implementierung verwendet wird. Hierbei wurde die Komplexität durch das Einsetzen von Strukturen in den entsprechenden Platzhaltern verringert. Darüber hinaus wurde das Element „certificates“ in der *ClientCookieData* durch das Schlüsselwort *ANY* substituiert. Dieses Schlüsselwort ermöglicht die Integration von Zertifikaten als bereits kodierten ASN.1-Datenstrom. Dadurch entfällt die Implementierung umfangreicher Strukturen und erleichtert die Programmwartung. Im Gegensatz dazu beinhaltet die *vollständige Modulbeschreibung* alle ASN.1-Strukturen ohne Vereinfachungen oder Substitution.

Kapitel 4 beinhaltet abschließend alle ASN.1-Modulbeschreibungen als Ganzes, die in den NTS-Nachrichtenobjekten, CMS-Strukturen und X.509-Zertifikaten verwendet werden. Die manuelle Implementierung der CMS-Strukturen und Zertifikate ist in NTS nicht notwendig, da diese durch OpenSSL bereitgestellt werden.

Derzeitig verwendete Quellen für die ASN.1-Module:

Spezifikation	Name
NTS Draft 15	Network Time Security (*draft-ietf-ntp-network-time-security-15*)
NTS4NTP Draft 06	Using the Network Time Security Specification to Secure the Network Time Protocol (*draft-ietf-ntp-using-nts-for-ntp-06*)
CMS4NTS Draft 07	Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS) (*draft-ietf-ntp-cms-for-nts-message-07*)
RFC 5652	Cryptographic Message Syntax
RFC 5280	PKIX Certificate and CRL Profile
RFC 5755	AC Profile for Authorization
X.509-1988	Authentication Framework
X.509-2000	Authentication Framework

2 Übersicht zu den verwendeten Strukturen

2.1 Network Time Security (alle Draft-Dokumente)

```
ClientAccessData
ClientAssocData
ClientCookieData
ContentEncryptionAlgorithmIdentifier
ContentEncryptionAlgorithmIdentifiers
KeyEncryptionAlgorithmIdentifier
KeyEncryptionAlgorithmIdentifiers
NTSAccessKey
NTSExtensionFieldContent
NTSMessageAuthenticationCode
NTSNonce
NTSVersion
ServerAccessData
ServerAssocData
ServerCookieData
TimeRequestSecurityData
TimeResponseSecurityData

id-ct-nts-clientAccess
id-ct-nts-clientAssoc
id-ct-nts-clientCookie
id-ct-nts-ntsForNtpMessageAuthenticationCode
id-ct-nts-serverAccess
id-ct-nts-serverAssoc
id-ct-nts-serverCookie
id-ct-nts-securityDataReq
id-ct-nts-securityDataResp
id-kp-ntsClientAuthz
id-kp-ntsServerAuth
id-kp-ntsServerAuthz
```

2.2 Cryptographic Message Syntax (RFC 5652)

```
AttCertVersionV1
Attribute
AttributeCertificateInfoV1
AttributeCertificateV1
AttributeCertificateV2
AttributeValue
CertificateChoices
CertificateSet
CMSVersion
ContentEncryptionAlgorithmIdentifier
ContentInfo
ContentType
DigestAlgorithmIdentifier
DigestAlgorithmIdentifiers
EncapsulatedContentInfo
EncryptedContent
EncryptedContentInfo
```

```
EncryptedKey
EnvelopedData
ExtendedCertificate
ExtendedCertificateInfo
IssuerAndSerialNumber
KEKIdentifier
KEKRecipientInfo
KeyAgreeRecipientIdentifier
KeyAgreeRecipientInfo
KeyDerivationAlgorithmIdentifier
KeyEncryptionAlgorithmIdentifier
KeyTransRecipientInfo
OriginatorIdentifierOrKey
OriginatorInfo
OriginatorPublicKey
OtherCertificateFormat
OtherKeyAttribute
OtherRecipientInfo
OtherRevocationInfoFormat
PasswordRecipientInfo
RecipientEncryptedKey
RecipientEncryptedKeys
RecipientIdentifier
RecipientInfo
RecipientInfos
RecipientKeyIdentifier
RevocationInfoChoice
RevocationInfoChoices
Signature
SignatureAlgorithmIdentifier
SignatureValue
SignedAttributes
SignedData
SignerIdentifier
SignerInfo
SignerInfos
SubjectKeyIdentifier
Time
UnauthAttributes
UnprotectedAttributes
UnsignedAttributes
UserKeyingMaterial
```

2.3 PKIX Certificate and CRL Profile (RFC 5280)

```
AdministrationDomainName
AnotherName
AttributeType
AttributeTypeAndValue
BuiltInDomainDefinedAttribute
BuiltInDomainDefinedAttributes
BuiltInStandardAttributes
Certificate
CertificateList
CertificateSerialNumber
CountryName
```

```
DirectoryString
EDIPartyName
Extension
ExtensionAttribute
ExtensionAttributes
Extensions
GeneralName
GeneralNames
Name
NetworkAddress
NumericUserIdentifier
ORAddress
OrganizationalUnitName
OrganizationalUnitNames
OrganizationName
PersonalName
PrivateDomainName
RDNSequence
RelativeDistinguishedName
SubjectPublicKeyInfo
TBSCertificate
TBSCertList
TerminalIdentifier
UniqueIdentifier
Validity
Version
X121Address
```

2.4 AC Profile for Authorization (RFC 5755)

```
AttCertValidityPeriod
IssuerSerial
```

2.5 Authentication Framework (X.509-1988)

```
AlgorithmIdentifier
```

2.6 Authentication Framework (X.509-2000)

```
AttCertIssuer
AttCertVersion
AttributeCertificate
AttributeCertificateInfo
Holder
ObjectDigestInfo
```

3 Die NTS-Nachrichtenobjekte

3.1 Vereinfachte Modulbeschreibung

```
NtsUnicastModule

DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

	ClientAccessData ::= NULL

	ServerAccessData ::= SEQUENCE {
		accessKey OCTET STRING (SIZE(16))
	}

	ClientAssocData ::= SEQUENCE {
		accessKey OCTET STRING (SIZE(16)),
		nonce OCTET STRING (SIZE(16)),
		minVersion INTEGER,
		macAlgos SET OF AlgorithmIdentifier,
		keyEncAlgos SET OF AlgorithmIdentifier,
		contentEncAlgos SET OF AlgorithmIdentifier
	}

	ServerAssocData ::= SEQUENCE {
		nonce OCTET STRING (SIZE(16)),
		proposedVersion INTEGER,
		macAlgos SET OF AlgorithmIdentifier,
		choiceMacAlgo AlgorithmIdentifier,
		keyEncAlgos SET OF AlgorithmIdentifier,
		choiceKeyEncAlgo AlgorithmIdentifier,
		contentEncAlgos SET OF AlgorithmIdentifier,
		choiceContentEncAlgo AlgorithmIdentifier
	}

	ClientCookieData ::= SEQUENCE {
		nonce OCTET STRING (SIZE(16)),
		signAlgo AlgorithmIdentifier,
		macAlgo AlgorithmIdentifier,
		encAlgo AlgorithmIdentifier,
		keyEncAlgo AlgorithmIdentifier,
		certificates SET OF ANY
		-- contains the 'Certificate' structure (X.509)
	}

	ServerCookieData ::= SEQUENCE {
		nonce OCTET STRING (SIZE(16)),
		cookie OCTET STRING (SIZE(16))
	}

	TimeRequestSecurityData ::= SEQUENCE {
		nonce OCTET STRING (SIZE(16)),
		macAlgo AlgorithmIdentifier,
		keyInputValue OCTET STRING (SIZE(16))
	}

	TimeResponseSecurityData ::= SEQUENCE {
		nonce OCTET STRING (SIZE(16))
	}
```

```
	NTSExtensionFieldContent ::= SEQUENCE {
		oid OBJECT IDENTIFIER,
		errnum OCTET STRING (SIZE(2)),
		content ANY DEFINED BY oid
	}

	NTSMessageAuthenticationCode ::= SEQUENCE {
		mac OCTET STRING (SIZE(16))
	}

	AlgorithmIdentifier ::= SEQUENCE {
		algorithm OBJECT IDENTIFIER,
		parameters ANY DEFINED BY algorithm OPTIONAL
	}

END
```

3.2 Vollständige Modulbeschreibung

```
NtsUnicastModule

DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

	ClientAccessData ::= NULL

	ServerAccessData ::= SEQUENCE {
		accessKey NTSAccessKey
	}

	ClientAssocData ::= SEQUENCE {
		accessKey  NTSAccessKey,
		nonce NTSNonce,
		minVersion NTSVersion,
		macAlgos DigestAlgorithmIdentifiers,
		keyEncAlgos KeyEncryptionAlgorithmIdentifiers,
		contentEncAlgos ContentEncryptionAlgorithmIdentifiers
	}

	ServerAssocData ::= SEQUENCE {
		nonce NTSNonce,
		proposedVersion NTSVersion,
		macAlgos DigestAlgorithmIdentifiers,
		choiceMacAlgo DigestAlgorithmIdentifier,
		keyEncAlgos KeyEncryptionAlgorithmIdentifiers,
		choiceKeyEncAlgo KeyEncryptionAlgorithmIdentifier,
		contentEncAlgos ContentEncryptionAlgorithmIdentifiers,
		choiceContentEncAlgo ContentEncryptionAlgorithmIdentifier
	}

	ClientCookieData ::= SEQUENCE {
		nonce NTSNonce,
		signAlgo SignatureAlgorithmIdentifier,
		macAlgo DigestAlgorithmIdentifier,
		encAlgo ContentEncryptionAlgorithmIdentifier,
		keyEncAlgo KeyEncryptionAlgorithmIdentifier,
		certificates CertificateSet
	}
```

```
ServerCookieData ::= SEQUENCE {
       nonce NTSNonce,
       cookie OCTET STRING (SIZE(16))
}

TimeRequestSecurityData ::= SEQUENCE {
       nonce NTSNonce,
       macAlgo DigestAlgorithmIdentifier,
       keyInputValue OCTET STRING (SIZE(16))
}

TimeResponseSecurityData ::= SEQUENCE {
       nonce NTSNonce
}

NTSExtensionFieldContent ::= SEQUENCE {
       oid OBJECT IDENTIFIER,
       errnum OCTET STRING (SIZE(2)),
       content ANY DEFINED BY oid
}


NTSMessageAuthenticationCode ::= SEQUENCE {
       mac OCTET STRING (SIZE(16))
}


NTSAccessKey ::= OCTET STRING (SIZE(16))
NTSNonce ::= OCTET STRING (SIZE(16))
NTSVersion ::= INTEGER (0..255)

KeyEncryptionAlgorithmIdentifiers ::=
SET OF KeyEncryptionAlgorithmIdentifier

ContentEncryptionAlgorithmIdentifiers ::=
SET OF ContentEncryptionAlgorithmIdentifier

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

DigestAlgorithmIdentifier ::= AlgorithmIdentifier
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

AlgorithmIdentifier ::= SEQUENCE {
       algorithm OBJECT IDENTIFIER,
       parameters ANY DEFINED BY algorithm OPTIONAL
}

CertificateSet ::= SET OF CertificateChoices

CertificateChoices ::= CHOICE {
       certificate Certificate,
       extendedCertificate [0] IMPLICIT ExtendedCertificate, -- Obsolete
       v1AttrCert [1] IMPLICIT AttributeCertificateV1, -- Obsolete
       v2AttrCert [2] IMPLICIT AttributeCertificateV2,
       other [3] IMPLICIT OtherCertificateFormat
}

Certificate ::= SEQUENCE {
       tbsCertificate TBSCertificate,
       signatureAlgorithm AlgorithmIdentifier,
       signatureValue BIT STRING
}
```

```
TBSCertificate ::= SEQUENCE {
       version [0] EXPLICIT Version DEFAULT v1,
       serialNumber CertificateSerialNumber,
       signature AlgorithmIdentifier,
       issuer Name,
       validity Validity,
       subject Name,
       subjectPublicKeyInfo SubjectPublicKeyInfo,
       issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
       -- If present, version MUST be v2 or v3
       subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
       -- If present, version MUST be v2 or v3

       extensions [3] EXPLICIT Extensions OPTIONAL
       -- If present, version MUST be v3
}

Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER

Name ::= CHOICE {
       rdnSequence RDNSequence -- only one possibility for now --
}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
       type AttributeType,
       value AttributeValue
}

Validity ::= SEQUENCE {
       notBefore Time,
       notAfter Time
}

Time ::= CHOICE {
       utcTime UTCTime,
       generalTime GeneralizedTime
}

SubjectPublicKeyInfo ::= SEQUENCE {
       algorithm AlgorithmIdentifier,
       subjectPublicKey BIT STRING
}

UniqueIdentifier ::= BIT STRING
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

Extension ::= SEQUENCE {
       extnID OBJECT IDENTIFIER,
       critical BOOLEAN DEFAULT FALSE,
       extnValue OCTET STRING
       -- contains the DER encoding of an ASN.1 value
       -- corresponding to the extension type identified
       -- by extnID
}

AttributeValue ::= ANY

ExtendedCertificate ::= SEQUENCE {
       extendedCertificateInfo ExtendedCertificateInfo,
       signatureAlgorithm SignatureAlgorithmIdentifier,
       signature Signature
}
```

```
ExtendedCertificateInfo ::= SEQUENCE {
       version CMSVersion,
       certificate Certificate,
       attributes UnauthAttributes
}

Signature ::= BIT STRING
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
CMSVersion ::= INTEGER { v0(0), v1(1), v2(2), v3(3), v4(4), v5(5) }
UnauthAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
       attrType OBJECT IDENTIFIER,
       attrValues SET OF AttributeValue
}

AttributeValue ::= ANY

AttributeCertificateV1 ::= SEQUENCE {
       acInfo AttributeCertificateInfoV1,
       signatureAlgorithm AlgorithmIdentifier,
       signature BIT STRING
}

AttributeCertificateInfoV1 ::= SEQUENCE {
       version AttCertVersionV1 DEFAULT v1,
       subject CHOICE {
              baseCertificateID [0] IssuerSerial,
              -- associated with a Public Key Certificate
              subjectName [1] GeneralNames  -- associated with a name
       },
       issuer GeneralNames,
       signature AlgorithmIdentifier,
       serialNumber CertificateSerialNumber,
       attCertValidityPeriod AttCertValidityPeriod,
       attributes SEQUENCE OF Attribute,
       issuerUniqueID UniqueIdentifier OPTIONAL,
       extensions Extensions OPTIONAL
}

AttCertVersionV1 ::= INTEGER { v1(0) }

IssuerSerial ::=  SEQUENCE {
       issuer         GeneralNames,
       serial         CertificateSerialNumber,
       issuerUID      UniqueIdentifier OPTIONAL
}

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
       otherName [0] AnotherName,
       rfc822Name [1] IA5String,
       dNSName [2] IA5String,
       x400Address [3] ORAddress,
       directoryName [4] Name,
       ediPartyName [5] EDIPartyName,
       uniformResourceIdentifier [6] IA5String,
       iPAddress [7] OCTET STRING,
       registeredID [8] OBJECT IDENTIFIER
}
-- AnotherName replaces OTHER-NAME ::= TYPE-IDENTIFIER, as
-- TYPE-IDENTIFIER is not supported in the '88 ASN.1 syntax
```

```
AnotherName ::= SEQUENCE {
       type-id OBJECT IDENTIFIER,
       value [0] EXPLICIT ANY DEFINED BY type-id
}

ORAddress ::= SEQUENCE {
       built-in-standard-attributes BuiltInStandardAttributes,

       built-in-domain-defined-attributes BuiltInDomainDefinedAttributes
       OPTIONAL,  -- see also teletex-domain-defined-attributes

       extension-attributes ExtensionAttributes OPTIONAL
}

BuiltInStandardAttributes ::= SEQUENCE {
       country-name CountryName OPTIONAL,
       administration-domain-name AdministrationDomainName OPTIONAL,

       network-address [0] IMPLICIT NetworkAddress OPTIONAL,
       -- see also extended-network-address

       terminal-identifier [1] IMPLICIT TerminalIdentifier OPTIONAL,
       private-domain-name [2] PrivateDomainName OPTIONAL,

       organization-name [3] IMPLICIT OrganizationName OPTIONAL,
       -- see also teletex-organization-name

       numeric-user-identifier [4] IMPLICIT NumericUserIdentifier OPTIONAL,

       personal-name [5] IMPLICIT PersonalName OPTIONAL,
       -- see also teletex-personal-name

       organizational-unit-names [6] IMPLICIT OrganizationalUnitNames
       OPTIONAL
}
-- see also teletex-organizational-unit-names

CountryName ::= [APPLICATION 1] CHOICE {
       x121-dcc-code NumericString (SIZE (ub-country-name-numeric-length)),
       iso-3166-alpha2-code PrintableString
       (SIZE (ub-country-name-alpha-length))
}

AdministrationDomainName ::= [APPLICATION 2] CHOICE {
       numeric NumericString (SIZE (0..ub-domain-name-length)),
       printable PrintableString (SIZE (0..ub-domain-name-length))
}

NetworkAddress ::= X121Address -- see also extended-network-address
X121Address ::= NumericString (SIZE (1..ub-x121-address-length))
TerminalIdentifier ::= PrintableString (SIZE (1..ub-terminal-id-length))

PrivateDomainName ::= CHOICE {
       numeric NumericString (SIZE (1..ub-domain-name-length)),
       printable PrintableString (SIZE (1..ub-domain-name-length))
}

OrganizationName ::= PrintableString
(SIZE (1..ub-organization-name-length))
-- see also teletex-organization-name

NumericUserIdentifier ::= NumericString
(SIZE (1..ub-numeric-user-id-length))
```

```
PersonalName ::= SET {
       surname [0] IMPLICIT PrintableString (SIZE (1..ub-surname-length)),

       given-name [1] IMPLICIT PrintableString
       (SIZE (1..ub-given-name-length)) OPTIONAL,

       initials [2] IMPLICIT PrintableString
       (SIZE (1..ub-initials-length)) OPTIONAL,

       generation-qualifier [3] IMPLICIT PrintableString
       (SIZE (1..ub-generation-qualifier-length)) OPTIONAL
}
-- see also teletex-personal-name

OrganizationalUnitNames ::= SEQUENCE SIZE
(1..ub-organizational-units) OF OrganizationalUnitName
-- see also teletex-organizational-unit-names

OrganizationalUnitName ::= PrintableString
(SIZE (1..ub-organizational-unit-name-length))

BuiltInDomainDefinedAttributes ::= SEQUENCE SIZE
(1..ub-domain-defined-attributes) OF BuiltInDomainDefinedAttribute

BuiltInDomainDefinedAttribute ::= SEQUENCE {
       type PrintableString
       (SIZE (1..ub-domain-defined-attribute-type-length)),

       value PrintableString
       (SIZE (1..ub-domain-defined-attribute-value-length))
}

ExtensionAttributes ::= SET SIZE
(1..ub-extension-attributes) OF ExtensionAttribute

ExtensionAttribute ::= SEQUENCE {
       extension-attribute-type [0] IMPLICIT INTEGER
       (0..ub-extension-attributes),

       extension-attribute-value [1] ANY DEFINED BY
       extension-attribute-type
}

EDIPartyName ::= SEQUENCE {
       nameAssigner [0] DirectoryString OPTIONAL,
       partyName [1] DirectoryString
}

DirectoryString ::= CHOICE {
       teletexString TeletexString (SIZE (1..MAX)),
       printableString PrintableString (SIZE (1..MAX)),
       universalString UniversalString (SIZE (1..MAX)),
       utf8String UTF8String (SIZE (1..MAX)),
       bmpString BMPString (SIZE (1..MAX))
}

CertificateSerialNumber ::= INTEGER

AttCertValidityPeriod ::= SEQUENCE {
       notBeforeTime  GeneralizedTime,
       notAfterTime   GeneralizedTime
}

AttributeCertificateV2 ::= AttributeCertificate
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}
```

```
	AttributeCertificateInfo ::= SEQUENCE{
		version AttCertVersion, --version is v2
		holder Holder,
		issuer AttCertIssuer,
		signature AlgorithmIdentifier,
		serialNumber CertificateSerialNumber,
		attrCertValidityPeriod AttCertValidityPeriod,
		attributes SEQUENCE OF Attribute,
		issuerUniqueID UniqueIdentifier OPTIONAL,
		extensions Extensions OPTIONAL
	}

	AttCertVersion ::= INTEGER { v2(1) }

	Holder ::= SEQUENCE {
		baseCertificateID [0] IssuerSerial OPTIONAL,
		entityName [1] GeneralNames OPTIONAL,
		objectDigestInfo [2] ObjectDigestInfo OPTIONAL
	}

	ObjectDigestInfo ::= SEQUENCE {
		digestedObjectType ENUMERATED {
			publicKey (0),
			publicKeyCert (1),
			otherObjectTypes (2)
		},
		otherObjectTypeID OBJECT IDENTIFIER OPTIONAL,
		digestAlgorithm AlgorithmIdentifier,
		objectDigest BIT STRING
	}

	AttCertIssuer ::= [0] SEQUENCE {
		issuerName GeneralNames OPTIONAL,
		baseCertificateID [0] IssuerSerial OPTIONAL,
		objectDigestInfo [1] ObjectDigestInfo OPTIONAL
	}

	-- At least one component shall be present
	( WITH COMPONENTS { ..., issuerName PRESENT } |
	WITH COMPONENTS { ..., baseCertificateID PRESENT } |
	WITH COMPONENTS { ..., objectDigestInfo PRESENT } )

	OtherCertificateFormat ::= SEQUENCE {
		otherCertFormat OBJECT IDENTIFIER,
		otherCert ANY DEFINED BY otherCertFormat
	}
END
```

4 Komplette ASN.1-Beschreibung für NTS

4.1 Network Time Security (alle Draft-Dokumente)

```
ClientAccessData ::= NULL


ServerAccessData ::= SEQUENCE {
       accessKey NTSAccessKey
}


ClientAssocData ::= SEQUENCE {
       accessKey  NTSAccessKey,
       nonce NTSNonce,
       minVersion NTSVersion,
       macAlgos AlgorithmIdentifiers,
       keyEncAlgos KeyEncryptionAlgorithmIdentifiers,
       contentEncAlgos ContentEncryptionAlgorithmIdentifiers
}


ServerAssocData ::= SEQUENCE {
       nonce NTSNonce,
       proposedVersion NTSVersion,
       macAlgos AlgorithmIdentifiers,
       choiceMacAlgo AlgorithmIdentifier,
       keyEncAlgos KeyEncryptionAlgorithmIdentifiers,
       choiceKeyEncAlgo KeyEncryptionAlgorithmIdentifier,
       contentEncAlgos ContentEncryptionAlgorithmIdentifiers,
       choiceContentEncAlgo ContentEncryptionAlgorithmIdentifier
}


ClientCookieData ::= SEQUENCE {
       nonce NTSNonce,
       signAlgo SignatureAlgorithmIdentifier,
       macAlgo AlgorithmIdentifier,
       encAlgo ContentEncryptionAlgorithmIdentifier,
       keyEncAlgo KeyEncryptionAlgorithmIdentifier,
       certificates CertificateSet
}


ServerCookieData ::= SEQUENCE {
       nonce NTSNonce,
       cookie OCTET STRING (SIZE(16))
}


TimeRequestSecurityData ::= SEQUENCE {
       nonce NTSNonce,
       macAlgo AlgorithmIdentifier,
       keyInputValue OCTET STRING (SIZE(16))
}


TimeResponseSecurityData ::= SEQUENCE {
       nonce NTSNonce,
}
```

```
NTSExtensionFieldContent := SEQUENCE {
       oid OBJECT IDENTIFIER,
       errnum OCTET STRING (SIZE(2)),
       content ANY DEFINED BY oid
}


NTSMessageAuthenticationCode := SEQUENCE {
       mac OCTET STRING (SIZE(16))
}


NTSAccessKey ::= OCTET STRING (SIZE(16))
NTSNonce ::= OCTET STRING (SIZE(16))
NTSVersion ::= INTEGER (0..255)


KeyEncryptionAlgorithmIdentifiers ::=
    SET OF KeyEncryptionAlgorithmIdentifier


ContentEncryptionAlgorithmIdentifiers ::=
    SET OF ContentEncryptionAlgorithmIdentifier


id-ct-nts-clientAccess OBJECT IDENTIFIER ::= TBD1
id-ct-nts-serverAccess OBJECT IDENTIFIER ::= TBD2
id-ct-nts-clientAssoc OBJECT IDENTIFIER ::= TBD3
id-ct-nts-serverAssoc OBJECT IDENTIFIER ::= TBD4
id-ct-nts-clientCookie OBJECT IDENTIFIER ::= TBD5
id-ct-nts-serverCookie OBJECT IDENTIFIER ::= TBD6
id-ct-nts-securityDataReq OBJECT IDENTIFIER ::= TBD7
id-ct-nts-securityDataResp OBJECT IDENTIFIER ::= TBD8
id-kp-ntsServerAuth OBJECT IDENTIFIER ::= TBD14
id-kp-ntsServerAuthz OBJECT IDENTIFIER ::= TBD15
id-kp-ntsClientAuthz OBJECT IDENTIFIER ::= TBD16
id-ct-nts-ntsForNtpMessageAuthenticationCode OBJECT IDENTIFIER ::= TBD2
```

4.2 Cryptographic Message Syntax (RFC 5652)

```
ContentInfo ::= SEQUENCE {
       contentType ContentType,
       content [0] EXPLICIT ANY DEFINED BY contentType
}


ContentType ::= OBJECT IDENTIFIER


SignedData ::= SEQUENCE {
       version CMSVersion,
       digestAlgorithms DigestAlgorithmIdentifiers,
       encapContentInfo EncapsulatedContentInfo,
       certificates [0] IMPLICIT CertificateSet OPTIONAL,
       crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
       signerInfos SignerInfos
}
```

```
DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier
SignerInfos ::= SET OF SignerInfo


EncapsulatedContentInfo ::= SEQUENCE {
       eContentType ContentType,
       eContent [0] EXPLICIT OCTET STRING OPTIONAL
}


SignerInfo ::= SEQUENCE {
       version CMSVersion,
       sid SignerIdentifier,
       digestAlgorithm DigestAlgorithmIdentifier,
       signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
       signatureAlgorithm SignatureAlgorithmIdentifier,
       signature SignatureValue,
       unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL
}


SignerIdentifier ::= CHOICE {
       issuerAndSerialNumber IssuerAndSerialNumber,
       subjectKeyIdentifier [0] SubjectKeyIdentifier
}


SignedAttributes ::= SET SIZE (1..MAX) OF Attribute
UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute


Attribute ::= SEQUENCE {
       attrType OBJECT IDENTIFIER,
       attrValues SET OF AttributeValue
}


AttributeValue ::= ANY
SignatureValue ::= OCTET STRING


EnvelopedData ::= SEQUENCE {
       version CMSVersion,
       originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
       recipientInfos RecipientInfos,
       encryptedContentInfo EncryptedContentInfo,
       unprotectedAttrs [1] IMPLICIT UnprotectedAttributes OPTIONAL
}


OriginatorInfo ::= SEQUENCE {
       certs [0] IMPLICIT CertificateSet OPTIONAL,
       crls [1] IMPLICIT RevocationInfoChoices OPTIONAL
}


RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo


EncryptedContentInfo ::= SEQUENCE {
       contentType ContentType,
       contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
       encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL
}
```

```
EncryptedContent ::= OCTET STRING
UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute


RecipientInfo ::= CHOICE {
       ktri KeyTransRecipientInfo,
       kari [1] KeyAgreeRecipientInfo,
       kekri [2] KEKRecipientInfo,
       pwri [3] PasswordRecipientInfo,
       ori [4] OtherRecipientInfo
}


EncryptedKey ::= OCTET STRING


KeyTransRecipientInfo ::= SEQUENCE {
       version CMSVersion, -- always set to 0 or 2
       rid RecipientIdentifier,
       keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
       encryptedKey EncryptedKey
}


RecipientIdentifier ::= CHOICE {
       issuerAndSerialNumber IssuerAndSerialNumber,
       subjectKeyIdentifier [0] SubjectKeyIdentifier
}


KeyAgreeRecipientInfo ::= SEQUENCE {
       version CMSVersion, -- always set to 3
       originator [0] EXPLICIT OriginatorIdentifierOrKey,
       ukm [1] EXPLICIT UserKeyingMaterial OPTIONAL,
       keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
       recipientEncryptedKeys RecipientEncryptedKeys
}


OriginatorIdentifierOrKey ::= CHOICE {
       issuerAndSerialNumber IssuerAndSerialNumber,
       subjectKeyIdentifier [0] SubjectKeyIdentifier,
       originatorKey [1] OriginatorPublicKey
}


OriginatorPublicKey ::= SEQUENCE {
       algorithm AlgorithmIdentifier,
       publicKey BIT STRING
}


RecipientEncryptedKeys ::= SEQUENCE OF RecipientEncryptedKey


RecipientEncryptedKey ::= SEQUENCE {
       rid KeyAgreeRecipientIdentifier,
       encryptedKey EncryptedKey
}


KeyAgreeRecipientIdentifier ::= CHOICE {
       issuerAndSerialNumber IssuerAndSerialNumber,
       rKeyId [0] IMPLICIT RecipientKeyIdentifier
}
```

```
RecipientKeyIdentifier ::= SEQUENCE {
       subjectKeyIdentifier SubjectKeyIdentifier,
       date GeneralizedTime OPTIONAL,
       other OtherKeyAttribute OPTIONAL
}


SubjectKeyIdentifier ::= OCTET STRING


KEKRecipientInfo ::= SEQUENCE {
       version CMSVersion, -- always set to 4
       kekid KEKIdentifier,
       keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
       encryptedKey EncryptedKey
}


KEKIdentifier ::= SEQUENCE {
       keyIdentifier OCTET STRING,
       date GeneralizedTime OPTIONAL,
       other OtherKeyAttribute OPTIONAL
}


PasswordRecipientInfo ::= SEQUENCE {
       version CMSVersion, -- always set to 0
       keyDerivationAlgorithm [0] KeyDerivationAlgorithmIdentifier OPTIONAL,
       keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
       encryptedKey EncryptedKey
}


OtherRecipientInfo ::= SEQUENCE {
       oriType OBJECT IDENTIFIER,
       oriValue ANY DEFINED BY oriType
}



UnauthAttributes ::= SET SIZE (1..MAX) OF Attribute
DigestAlgorithmIdentifier ::= AlgorithmIdentifier
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier
RevocationInfoChoices ::= SET OF RevocationInfoChoice


RevocationInfoChoice ::= CHOICE {
       crl CertificateList,
       other [1] IMPLICIT OtherRevocationInfoFormat
}


OtherRevocationInfoFormat ::= SEQUENCE {
       otherRevInfoFormat OBJECT IDENTIFIER,
       otherRevInfo ANY DEFINED BY otherRevInfoFormat
}
```

```
CertificateChoices ::= CHOICE {
       certificate Certificate,
       extendedCertificate [0] IMPLICIT ExtendedCertificate, -- Obsolete
       v1AttrCert [1] IMPLICIT AttributeCertificateV1, -- Obsolete
       v2AttrCert [2] IMPLICIT AttributeCertificateV2,
       other [3] IMPLICIT OtherCertificateFormat
}


AttributeCertificateV2 ::= AttributeCertificate


OtherCertificateFormat ::= SEQUENCE {
       otherCertFormat OBJECT IDENTIFIER,
       otherCert ANY DEFINED BY otherCertFormat
}


CertificateSet ::= SET OF CertificateChoices


IssuerAndSerialNumber ::= SEQUENCE {
       issuer Name,
       serialNumber CertificateSerialNumber
}


CMSVersion ::= INTEGER { v0(0), v1(1), v2(2), v3(3), v4(4), v5(5) }
UserKeyingMaterial ::= OCTET STRING


OtherKeyAttribute ::= SEQUENCE {
       keyAttrId OBJECT IDENTIFIER,
       keyAttr ANY DEFINED BY keyAttrId OPTIONAL
}


Time ::= CHOICE {
       utcTime UTCTime,
       generalTime GeneralizedTime
}


-- Obsolete Extended Certificate syntax from PKCS #6
-----------------------------------------------------
ExtendedCertificate ::= SEQUENCE {
       extendedCertificateInfo ExtendedCertificateInfo,
       signatureAlgorithm SignatureAlgorithmIdentifier,
       signature Signature
}


ExtendedCertificateInfo ::= SEQUENCE {
       version CMSVersion,
       certificate Certificate,
       attributes UnauthAttributes
}


Signature ::= BIT STRING


-- Content Type Object Identifiers
-----------------------------------------------------
id-ct-contentInfo OBJECT IDENTIFIER ::={
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) ct(1) 6}
```

```
id-data OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1}

id-signedData OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2}

id-envelopedData OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3}

id-digestedData OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 5}

id-encryptedData OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 6}

id-ct-authData OBJECT IDENTIFIER ::= {
iso(1) member-body(2)us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 2}


-- Attribute Object Identifiers
---------------------------------------------------------
id-contentType OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3}

id-messageDigest OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4}

id-signingTime OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5}


AttributeCertificateV1 ::= SEQUENCE {
       acInfo AttributeCertificateInfoV1,
       signatureAlgorithm AlgorithmIdentifier,
       signature BIT STRING
}


AttributeCertificateInfoV1 ::= SEQUENCE {
       version AttCertVersionV1 DEFAULT v1,
       subject CHOICE {
              baseCertificateID [0] IssuerSerial,
         -- associated with a Public Key Certificate
              subjectName [1] GeneralNames
              -- associated with a name
       },
       issuer GeneralNames,
       signature AlgorithmIdentifier,
       serialNumber CertificateSerialNumber,
       attCertValidityPeriod AttCertValidityPeriod,
       attributes SEQUENCE OF Attribute,
       issuerUniqueID UniqueIdentifier OPTIONAL,
       extensions Extensions OPTIONAL
}


AttCertVersionV1 ::= INTEGER { v1(0) }
```

4.3 PKIX Certificate and CRL Profile (RFC 5280)

```
-- rfc5280 - PKIX Certificate and CRL Profile
PKIX1Explicit88 {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) mod(0) pkix1-explicit(18)
}


Certificate ::= SEQUENCE {
       tbsCertificate TBSCertificate,
       signatureAlgorithm AlgorithmIdentifier,
       signatureValue BIT STRING
}


TBSCertificate ::= SEQUENCE {
       version [0] EXPLICIT Version DEFAULT v1,
       serialNumber CertificateSerialNumber,
       signature AlgorithmIdentifier,
       issuer Name,
       validity Validity,
       subject Name,
       subjectPublicKeyInfo SubjectPublicKeyInfo,
       issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
       -- If present, version MUST be v2 or v3
       subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
       -- If present, version MUST be v2 or v3
       extensions [3] EXPLICIT Extensions OPTIONAL
       -- If present, version MUST be v3
}


Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER


Name ::= CHOICE {
       rdnSequence RDNSequence -- only one possibility for now --
}


RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue


AttributeTypeAndValue ::= SEQUENCE {
       type AttributeType,
       value AttributeValue
}


AttributeType ::= OBJECT IDENTIFIER


Validity ::= SEQUENCE {
       notBefore Time,
       notAfter Time
}


SubjectPublicKeyInfo ::= SEQUENCE {
       algorithm AlgorithmIdentifier,
       subjectPublicKey BIT STRING
}
```

```
UniqueIdentifier ::= BIT STRING
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension


Extension ::= SEQUENCE {
       extnID OBJECT IDENTIFIER,
       critical BOOLEAN DEFAULT FALSE,
       extnValue OCTET STRING
       -- contains the DER encoding of an ASN.1 value
       -- corresponding to the extension type identified
       -- by extnID
}


CertificateList ::= SEQUENCE {
       tbsCertList TBSCertList,
       signatureAlgorithm AlgorithmIdentifier,
       signature BIT STRING
}


TBSCertList ::= SEQUENCE {
       version Version OPTIONAL, -- if present, MUST be v2
       signature AlgorithmIdentifier,
       issuer Name,
       thisUpdate Time,
       nextUpdate Time OPTIONAL,
       revokedCertificates SEQUENCE OF SEQUENCE {
              userCertificate CertificateSerialNumber,
              revocationDate Time,
              crlEntryExtensions Extensions OPTIONAL
              -- if present, version MUST be v2
       } OPTIONAL,
       crlExtensions [0] Extensions OPTIONAL
}
-- if present, version MUST be v2
-- Version, Time, CertificateSerialNumber, and Extensions were
   defined earlier for use in the certificate structure


GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName


GeneralName ::= CHOICE {
       otherName [0] AnotherName,
       rfc822Name [1] IA5String,
       dNSName [2] IA5String,
       x400Address [3] ORAddress,
       directoryName [4] Name,
       ediPartyName [5] EDIPartyName,
       uniformResourceIdentifier [6] IA5String,
       iPAddress [7] OCTET STRING,
       registeredID [8] OBJECT IDENTIFIER
}
-- AnotherName replaces OTHER-NAME ::= TYPE-IDENTIFIER, as
-- TYPE-IDENTIFIER is not supported in the '88 ASN.1 syntax


AnotherName ::= SEQUENCE {
       type-id OBJECT IDENTIFIER,
       value [0] EXPLICIT ANY DEFINED BY type-id
}
```

```
ORAddress ::= SEQUENCE {
       built-in-standard-attributes BuiltInStandardAttributes,
       built-in-domain-defined-attributes
       BuiltInDomainDefinedAttributes OPTIONAL,
       -- see also teletex-domain-defined-attributes
       extension-attributes ExtensionAttributes OPTIONAL
}


BuiltInStandardAttributes ::= SEQUENCE {
       country-name CountryName OPTIONAL,
       administration-domain-name AdministrationDomainName OPTIONAL,
       network-address [0] IMPLICIT NetworkAddress OPTIONAL,
       -- see also extended-network-address
       terminal-identifier [1] IMPLICIT TerminalIdentifier OPTIONAL,
       private-domain-name [2] PrivateDomainName OPTIONAL,
       organization-name [3] IMPLICIT OrganizationName OPTIONAL,
       -- see also teletex-organization-name
       numeric-user-identifier [4] IMPLICIT NumericUserIdentifier OPTIONAL,
       personal-name [5] IMPLICIT PersonalName OPTIONAL,
       -- see also teletex-personal-name
       organizational-unit-names [6] IMPLICIT OrganizationalUnitNames OPTIONAL
}
-- see also teletex-organizational-unit-names


CountryName ::= [APPLICATION 1] CHOICE {
       x121-dcc-code NumericString (SIZE (ub-country-name-numeric-length)),
       iso-3166-alpha2-code PrintableString (SIZE (ub-country-name-alpha-length))
}


AdministrationDomainName ::= [APPLICATION 2] CHOICE {
       numeric NumericString (SIZE (0..ub-domain-name-length)),
       printable PrintableString (SIZE (0..ub-domain-name-length))
}


NetworkAddress ::= X121Address -- see also extended-network-address
X121Address ::= NumericString (SIZE (1..ub-x121-address-length))
TerminalIdentifier ::= PrintableString (SIZE (1..ub-terminal-id-length))


PrivateDomainName ::= CHOICE {
       numeric NumericString (SIZE (1..ub-domain-name-length)),
       printable PrintableString (SIZE (1..ub-domain-name-length))
}


OrganizationName ::= PrintableString (SIZE (1..ub-organization-name-length))
-- see also teletex-organization-name

NumericUserIdentifier ::= NumericString (SIZE (1..ub-numeric-user-id-length))


PersonalName ::= SET {
       surname [0] IMPLICIT PrintableString
       (SIZE (1..ub-surname-length)),
       given-name [1] IMPLICIT PrintableString
       (SIZE (1..ub-given-name-length)) OPTIONAL,
       initials [2] IMPLICIT PrintableString
       (SIZE (1..ub-initials-length)) OPTIONAL,
       generation-qualifier [3] IMPLICIT PrintableString
       (SIZE (1..ub-generation-qualifier-length)) OPTIONAL
}
-- see also teletex-personal-name
```

```
OrganizationalUnitNames ::= SEQUENCE SIZE (1..ub-organizational-units)
OF OrganizationalUnitName    -- see also teletex-organizational-unit-names

OrganizationalUnitName ::= PrintableString
(SIZE (1..ub-organizational-unit-name-length))


BuiltInDomainDefinedAttributes ::=
SEQUENCE SIZE (1..ub-domain-defined-attributes) OF BuiltInDomainDefinedAttribute


BuiltInDomainDefinedAttribute ::= SEQUENCE {
       type PrintableString (SIZE (1..ub-domain-defined-attribute-type-length)),
       value PrintableString (SIZE (1..ub-domain-defined-attribute-value-length))
}


ExtensionAttributes ::=
SET SIZE (1..ub-extension-attributes) OF ExtensionAttribute


ExtensionAttribute ::= SEQUENCE {
       extension-attribute-type [0] IMPLICIT INTEGER (0..ub-extension-
attributes),
       extension-attribute-value [1] ANY DEFINED BY extension-attribute-type
}


EDIPartyName ::= SEQUENCE {
       nameAssigner [0] DirectoryString OPTIONAL,
       partyName [1] DirectoryString
}


DirectoryString ::= CHOICE {
       teletexString TeletexString (SIZE (1..MAX)),
       printableString PrintableString (SIZE (1..MAX)),
       universalString UniversalString (SIZE (1..MAX)),
       utf8String UTF8String (SIZE (1..MAX)),
       bmpString BMPString (SIZE (1..MAX))
}
```

4.4 AC Profile for Authorization (RFC 5755)

```
PKIXAttributeCertificate-2008 {
       iso(1) identified-organization(3)
       dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
       id-mod-attribute-cert-v2(61)
}


AttCertValidityPeriod ::= SEQUENCE {
       notBeforeTime  GeneralizedTime,
       notAfterTime   GeneralizedTime
}


IssuerSerial ::=  SEQUENCE {
       issuer         GeneralNames,
       serial         CertificateSerialNumber,
       issuerUID      UniqueIdentifier OPTIONAL
}
```

4.5 Authentication Framework (X.509-1988)

```
AlgorithmIdentifier ::= SEQUENCE {
       algorithm OBJECT IDENTIFIER,
       parameters ANY DEFINED BY algorithm OPTIONAL
}
```

4.6 Authentication Framework (X.509-2000)

```
AttributeCertificate ::= SIGNED {AttributeCertificateInfo}


AttributeCertificateInfo ::= SEQUENCE{
       version AttCertVersion, --version is v2
       holder Holder,
       issuer AttCertIssuer,
       signature AlgorithmIdentifier,
       serialNumber CertificateSerialNumber,
       attrCertValidityPeriod AttCertValidityPeriod,
       attributes SEQUENCE OF Attribute,
       issuerUniqueID UniqueIdentifier OPTIONAL,
       extensions Extensions OPTIONAL
}


AttCertVersion ::= INTEGER { v2(1) }


Holder ::= SEQUENCE {
       baseCertificateID [0] IssuerSerial OPTIONAL,
       entityName [1] GeneralNames OPTIONAL,
       objectDigestInfo [2] ObjectDigestInfo OPTIONAL
}


ObjectDigestInfo ::= SEQUENCE {
       digestedObjectType ENUMERATED {
              publicKey (0),
              publicKeyCert (1),
              otherObjectTypes (2)
       },
       otherObjectTypeID OBJECT IDENTIFIER OPTIONAL,
       digestAlgorithm AlgorithmIdentifier,
       objectDigest BIT STRING
}


AttCertIssuer ::= [0] SEQUENCE {
       issuerName GeneralNames OPTIONAL,
       baseCertificateID [0] IssuerSerial OPTIONAL,
       objectDigestInfo [1] ObjectDigestInfo OPTIONAL
}

-- At least one component shall be present
( WITH COMPONENTS { ..., issuerName PRESENT } |
WITH COMPONENTS { ..., baseCertificateID PRESENT } |
WITH COMPONENTS { ..., objectDigestInfo PRESENT } )
```

Ostfalia
Hochschule für angewandte Wissenschaften

Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb

Anlage III:

Ablaufbeschreibung durch Sequenzdiagramme

Martin Langer
Matrikelnummer: 70405974
Eingereicht am: 21.10.2016

Ostfalia, Wolfenbüttel
Fakultät Elektrotechnik
Studiengang: Intelligente Mobilität und Energiesystemtechnik

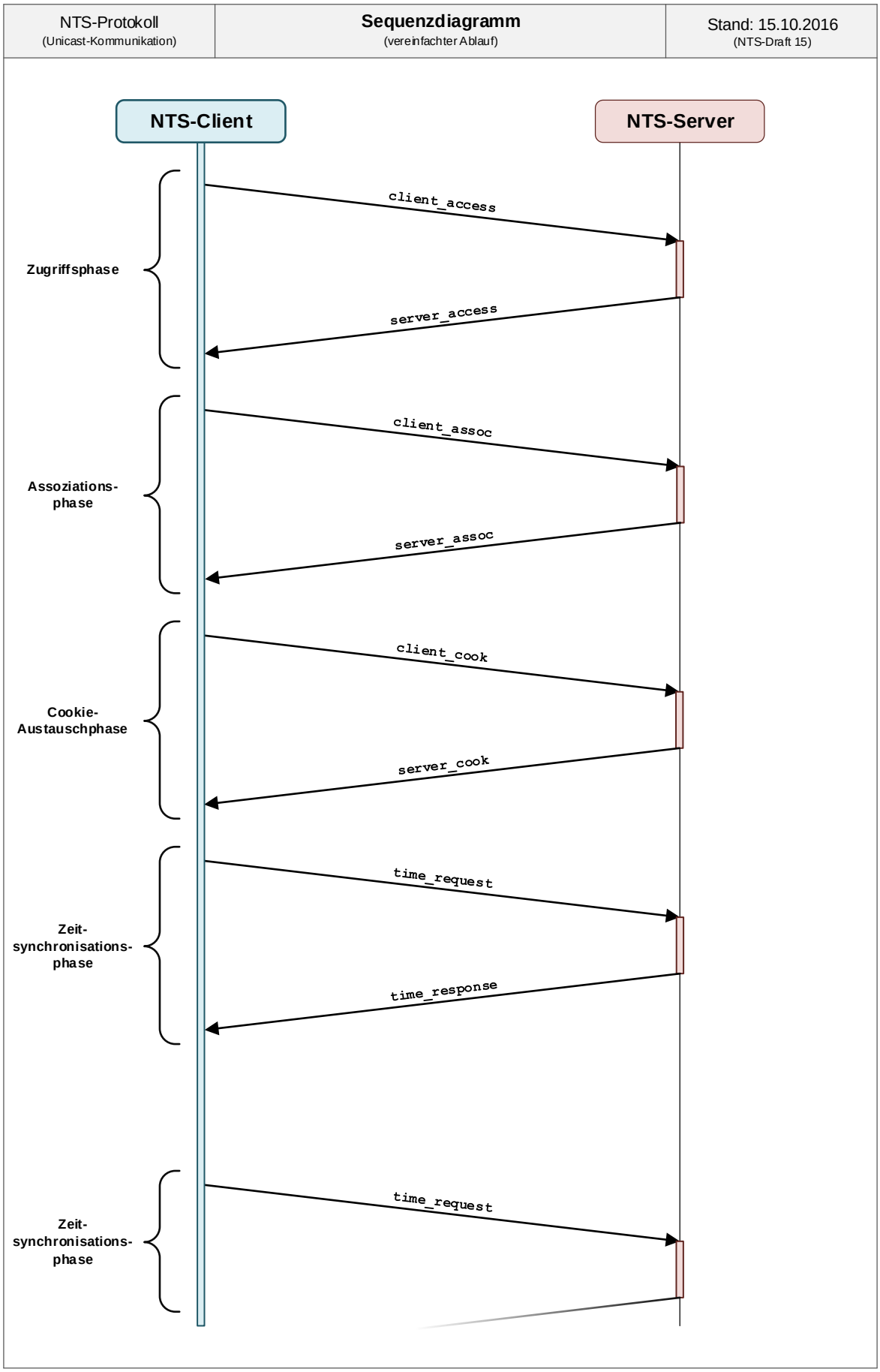
Dokumentversion: 1.0.0.17
Letze Änderung: 18.10.2016 23:29:00

Die NTS-Ablaufbeschreibung mit Sequenzdiagrammen

Die folgenden Sequenzdiagramme stellen den vereinfachten und detaillierten Ablauf des NTS-Protokolls dar. Der Fokus liegt hierbei auf den einzelnen Elementen, die zwischen Client und Server ausgetauscht werden. Somit ist leicht ersichtlich, an welchen Stellen bestimmte Algorithmen oder Schlüssel zum Einsatz kommen und an welchen Stellen diese wieder verworfen werden. An dieser Stelle sei auch erwähnt, dass der Server im Gegensatz zum Client zustandslos ist und keinerlei Informationen nach dem Absenden einer Antwortnachricht speichert.

Die Sequenzdiagramme basieren hierbei auf den folgenden NTS-Dokumente:

- *draft-ietf-ntp-network-time-security-15* (gültig bis: 26.03.2017)
- *draft-ietf-ntp-using-nts-for-ntp-06* (gültig bis: 26.03.2017)
- *draft-ietf-ntp-cms-for-nts-message-07* (gültig bis: 22.09.2016)


NTS-Protokoll
(Unicast-Kommunikation)
Sequenzdiagramm
(vereinfachter Ablauf)
Stand: 15.10.2016
(NTS-Draft 15)
NTS-Client
NTS-Server
client_access
server_access
Zugriffsphase
client_assoc
server_assoc
Assoziations-
phase
client_cook
server_cook
Cookie-
Austauschphase
time_request
time_response
Zeit-
synchronisations-
phase
time_request
Zeit-
synchronisations-
phase

NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Legende - Seite 1/3)	Stand: 15.10.2016 (NTS-Draft 15)

Bereiche einer NTS-Nachricht

Hellgrau: Informationen in diesem Teil einer NTS-Nachricht sind **implizit** enthalten (z.B. die angewendete NTS-Version oder Daten in der CMS-Struktur). Diese Informationen können ggf. nicht sofort extrahiert werden.

Grau: repräsentiert das unverschlüsselte NTS-Nachrichtenobjekt einer NTS-Nachricht, auf dessen Informationen direkt zugegriffen werden kann

Weiß-Rot gestreift: repräsentiert das verschlüsselte NTS-Nachrichtenobjekt einer NTS-Nachricht, auf dessen Inhalte erst nach der Entschlüsselung zugegriffen werden kann

Farben der Elemente

Element — **Blau**: sind Parameter bzw. Informationen, die dem Client zu Beginn der Kommunikation zur Verfügung stehen oder im Verarbeitungsprozess generiert werden

Element — **Lila**: sind Informationen, die vom Client an den Server übertragen und unverändert vom Server zurückgesendet werden

Element — **Rot**: sind Parameter bzw. Informationen, die dem Server zu Beginn der Kommunikation zur Verfügung stehen oder im Verarbeitungsprozess generiert werden

Element — **Orange**: sind Informationen, die vom Server an den Client übertragen und unverändert vom Client zurückgesendet werden

Element — **Grün**: sind Informationen, die vom Client und Server ausgehandelt wurden und im weiteren Verlauf der Kommunikation Anwendung finden

Element — **Durchgestrichen**: sind Informationen, die nicht mehr benötigt und gelöscht werden

Schlüssel im NTS-Protokoll

öffentlicher Schlüssel des Clients: wird zur Verschlüsselung des symmetrischen Schlüssels verwendet

privater Schlüssel des Clients: wird zur Entschlüsselung des symmetrischen Schlüssels verwendet

öffentlicher Schlüssel des Servers: ermöglicht dem Client die Prüfung der Server-Signaturen

privater Schlüssel des Servers: wird vom Server zur Signierung der NTS-Nachrichten verwendet

symmetrischer Schlüssel: wird zur Ver- und Entschlüsselung des NTS-Nachrichtenobjekts benötigt

NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Legende - Seite 2/3)	Stand: 15.10.2016 (NTS-Draft 15)

Beschreibung der Elemente

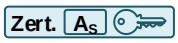
A_{CE}	***ContentEncryption*-Algorithmus**: kommt bei der Ver- und Entschlüsselung des NTS-Nachrichtenobjekts zur Anwendung (symmetrischer Verschlüsselungsalgorithmus)
A_{KE}	***KeyEncryption*-Algorithmus**: kommt bei der Ver- und Entschlüsselung des symmetrischen Schlüssels ( ) zur Anwendung (asymmetrischer Verschlüsselungsalgorithmus)
A_M	**MAC-Algorithmus**: wird zur der Berechnung unterschiedlicher Elemente benötigt (*Access Key*, *Cookie*, *KIV*, *MAC* für NTP-Paket)
A_S	**Signaturalgorithmus**: kommt bei der Signierung der NTS-Nachricht (CMS-SignedData-Struktur) und bei X.509-Zertifikaten zum Einsatz
AL_{CE}	**Liste mit *ContentEncryption*-Algorithmen**: enthält eine Sammlung von symmetrischen Verschlüsselungsalgorithmen, die unterstützt werden
AL_{KE}	**Liste mit *KeyEncryption*-Algorithmen**: enthält eine Sammlung von asymmetrischen Verschlüsselungsalgorithmen, die unterstützt werden
AL_M	**Liste mit MAC-Algorithmen**: enthält eine Sammlung von MAC-Algorithmen (z.B. HMAC-SHA-512, GMAC, CMAC), die unterstützt werden
AL_S	**Signatur-Algorithmusliste**: beinhaltet eine Sammlung von Signaturalgorithmen, die unterstützt werden
AK	**Access Key:** der Zugriffsschlüssel erlaubt dem Client die Anfrage einer Assoziationsnachricht an den Server
Cookie	**Cookie**: ist ein (symmetrischer) Schlüssel, der vom Server generiert wird und die spätere Überprüfung der NTP-Nachrichten (anhand des *MAC*) ermöglicht
IP	**IP-Adresse:** ist die Netzwerkadresse (IPv4 oder IPv6) und wird zur Berechnung des *Access Keys* benötigt.
KIV	**Key Input Value**: ist der berechnete MAC des Clientzertifikats und wird bei der Erzeugung des *Cookies* benötigt
MAC	**Message Authentication Code**: ist der „Fingerabdruck" der NTP-Nachricht und ermöglicht deren spätere Überprüfung auf Integrität und Authentizität.
N	**Nonce**: ist eine kryptografische Zufallszahl, die vom Client generiert und an eine NTS-Nachricht angehängt wird (zum Schutz vor Replay-Angriffen)
NTP-Paket	**NTP-Paket:** stellt das komplette NTPv4-Paket dar
Seed	**Server Seed**: ist eine kryptografische Zufallszahl, die vom Server generiert und geheim gehalten wird. Diese ändert sich periodisch und wird zur Berechnung des *Cookies* benötigt.

NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Legende - Seite 3/3)	Stand: 15.10.2016 (NTS-Draft 15)

Beschreibung der Elemente

Zeitstempel (*transmit_timestamp*): ist der Zeitstempel aus dem NTP-Paket direkt vor der Übertragung der NTP-Nachricht durch den Client

Zeitstempel (*originate_timestamp*): ist der Zeitstempel aus dem NTP-Paket, den der Server in der Antwortnachricht angelegt und mit dem *transmit_timestamp* der Anfrage identisch sein muss

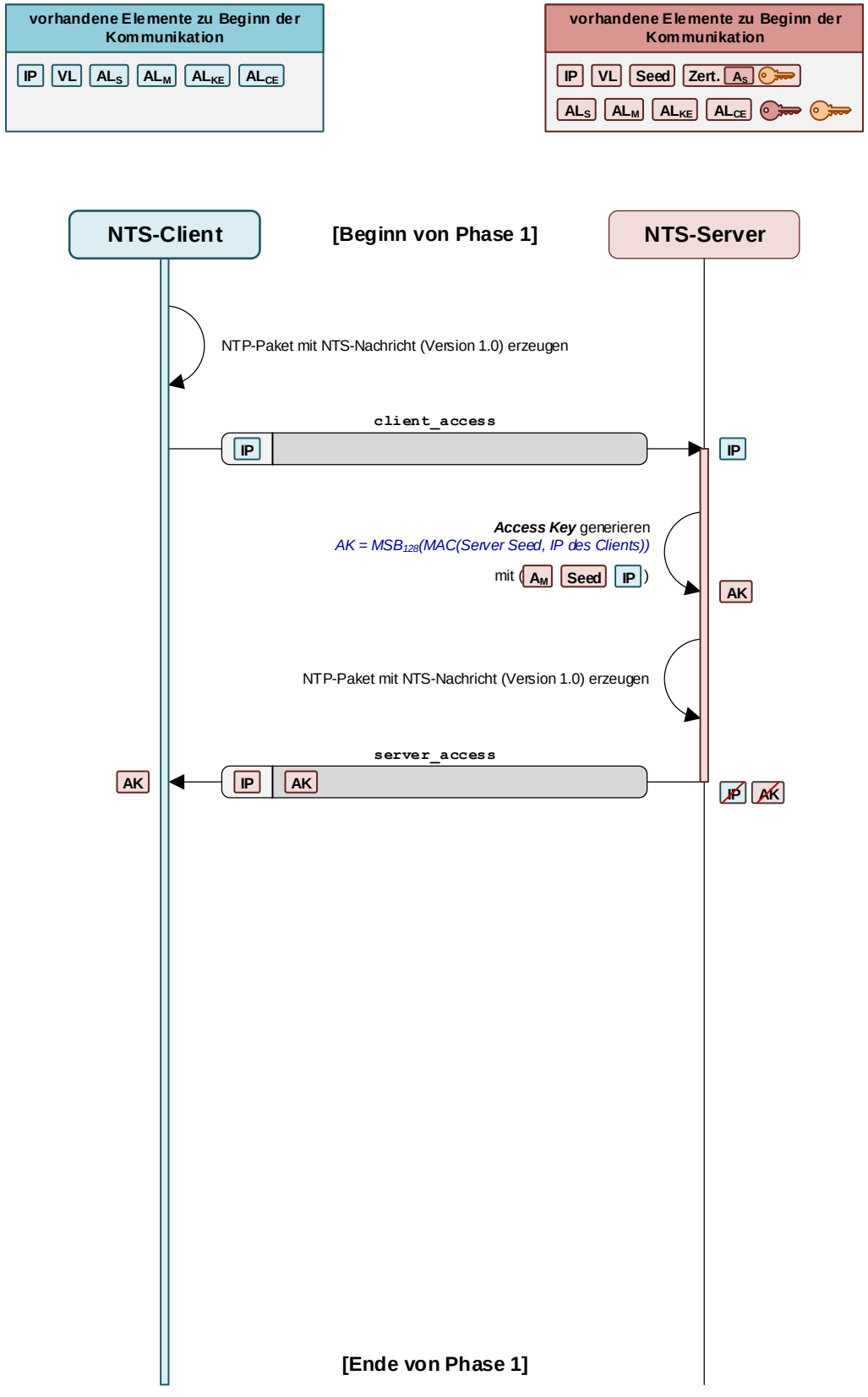


Zertifikat: ist ein X.509v3-Zertifikat, des einen öffentlichen Schlüssel und einen Signaturalgorithmus beinhaltet

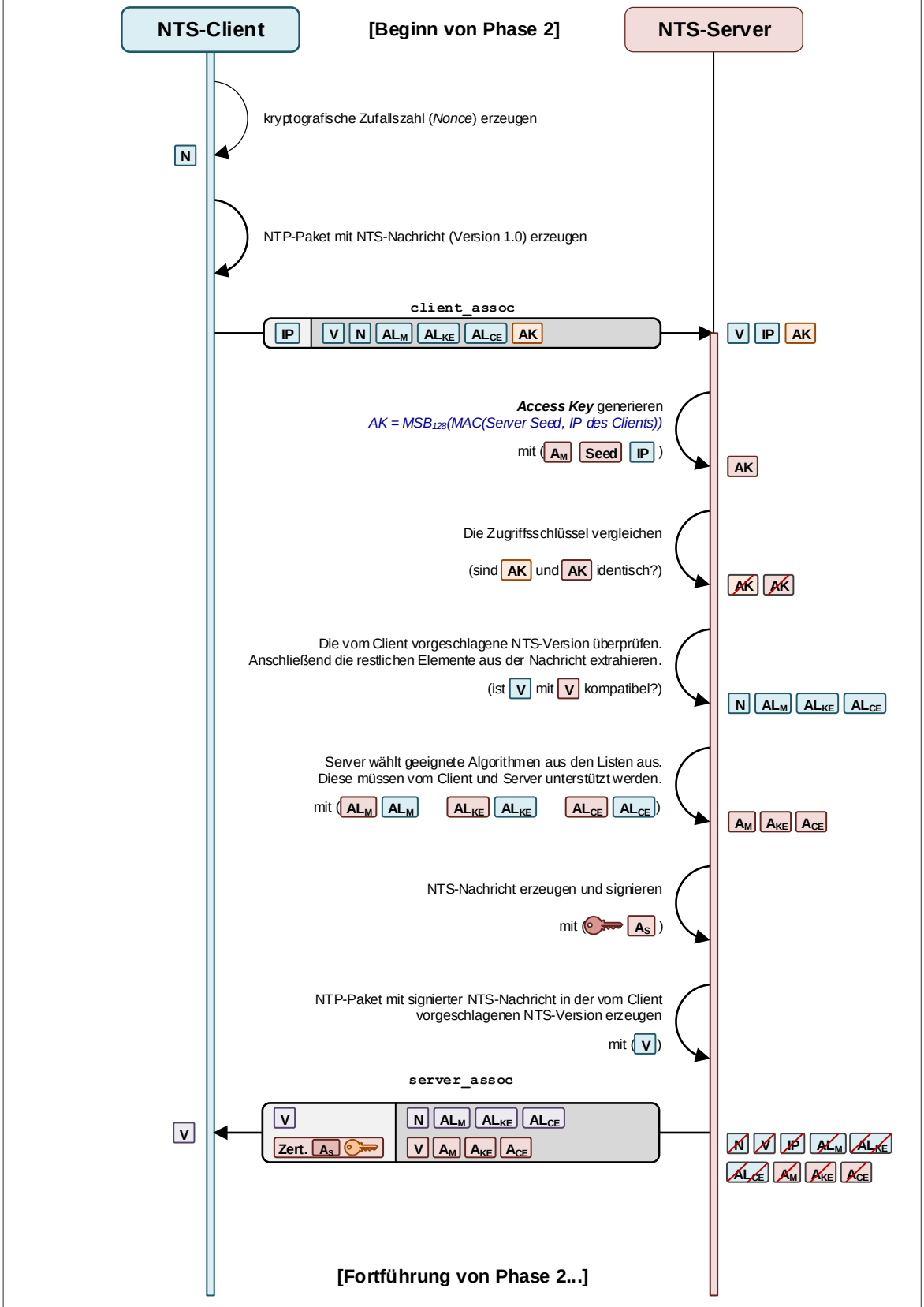
NTS-Version: die Version, die zwischen Client und Server ausgehandelt wird (bestimmt den strukturellen Aufbau einer NTS-Nachricht)

Liste mit unterstützten NTS-Versionen: enthält alle unterstützen NTS-Versionen, die frei ausgehandelt werden können

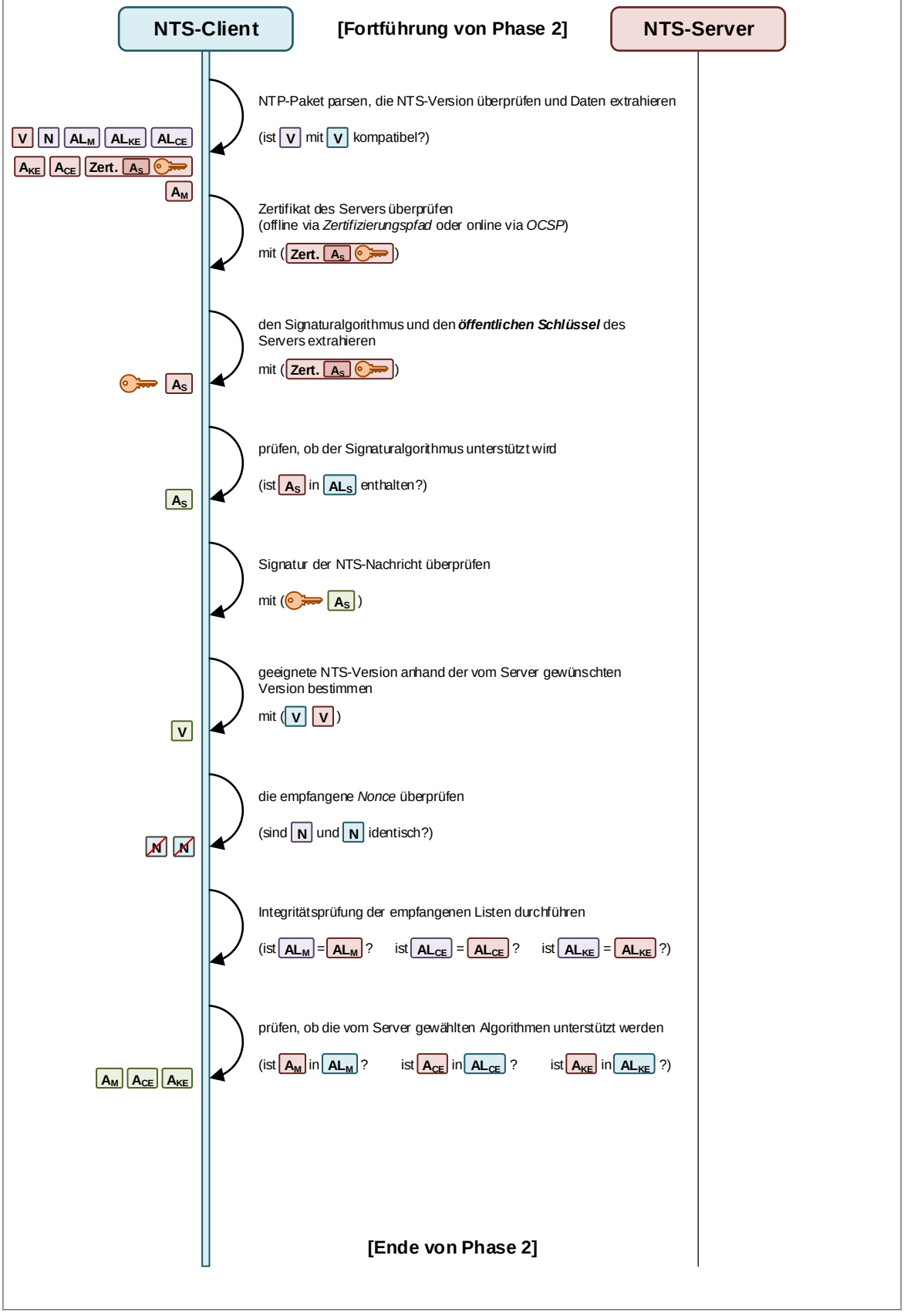
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 1/4 - Zugriffsnachrichten - Seite 1/1)	Stand: 15.10.2016 (NTS-Draft 15)



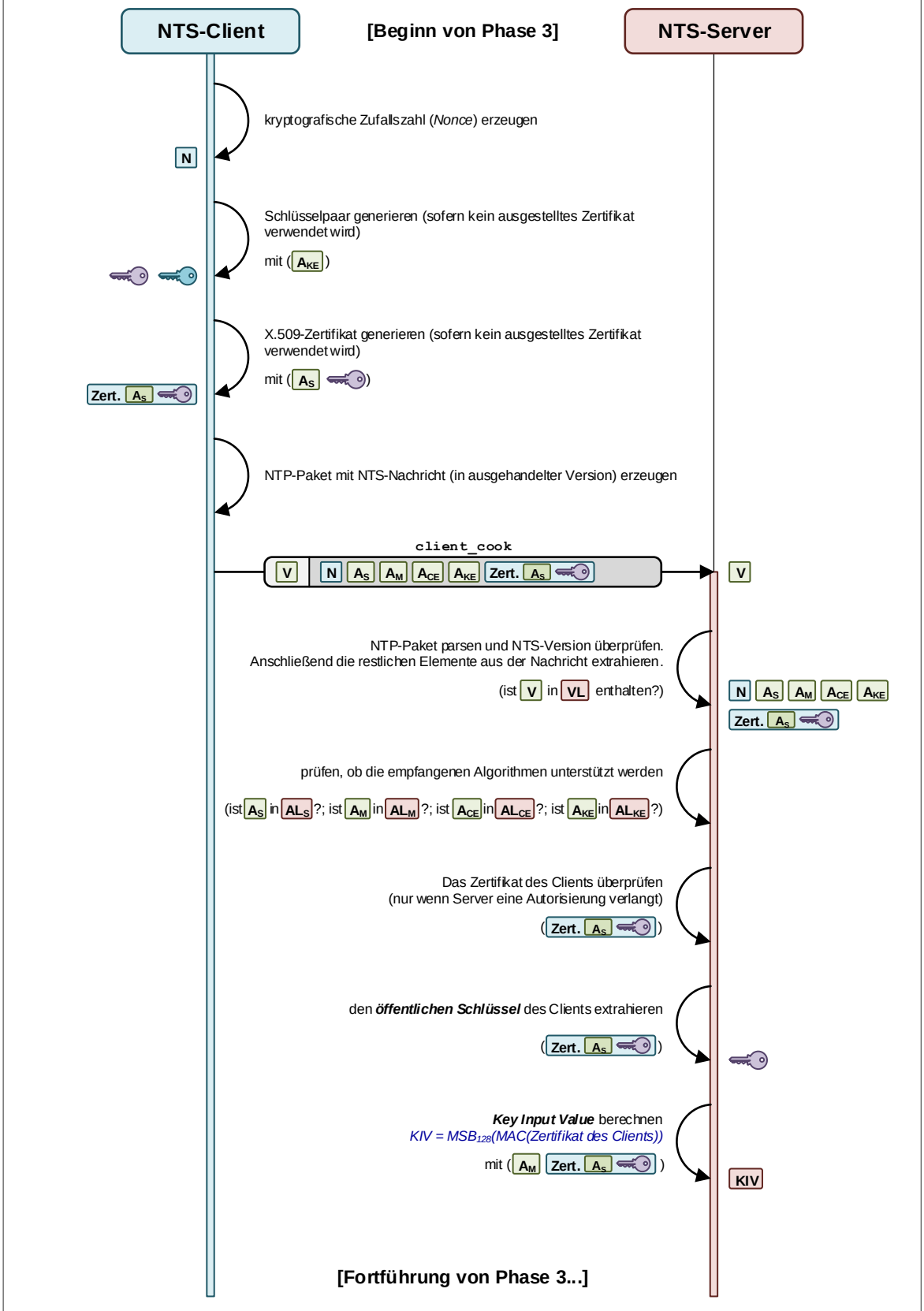
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 2/4 - Zugriffsnachrichten - Seite 1/2)	Stand: 15.10.2016 (NTS-Draft 15)



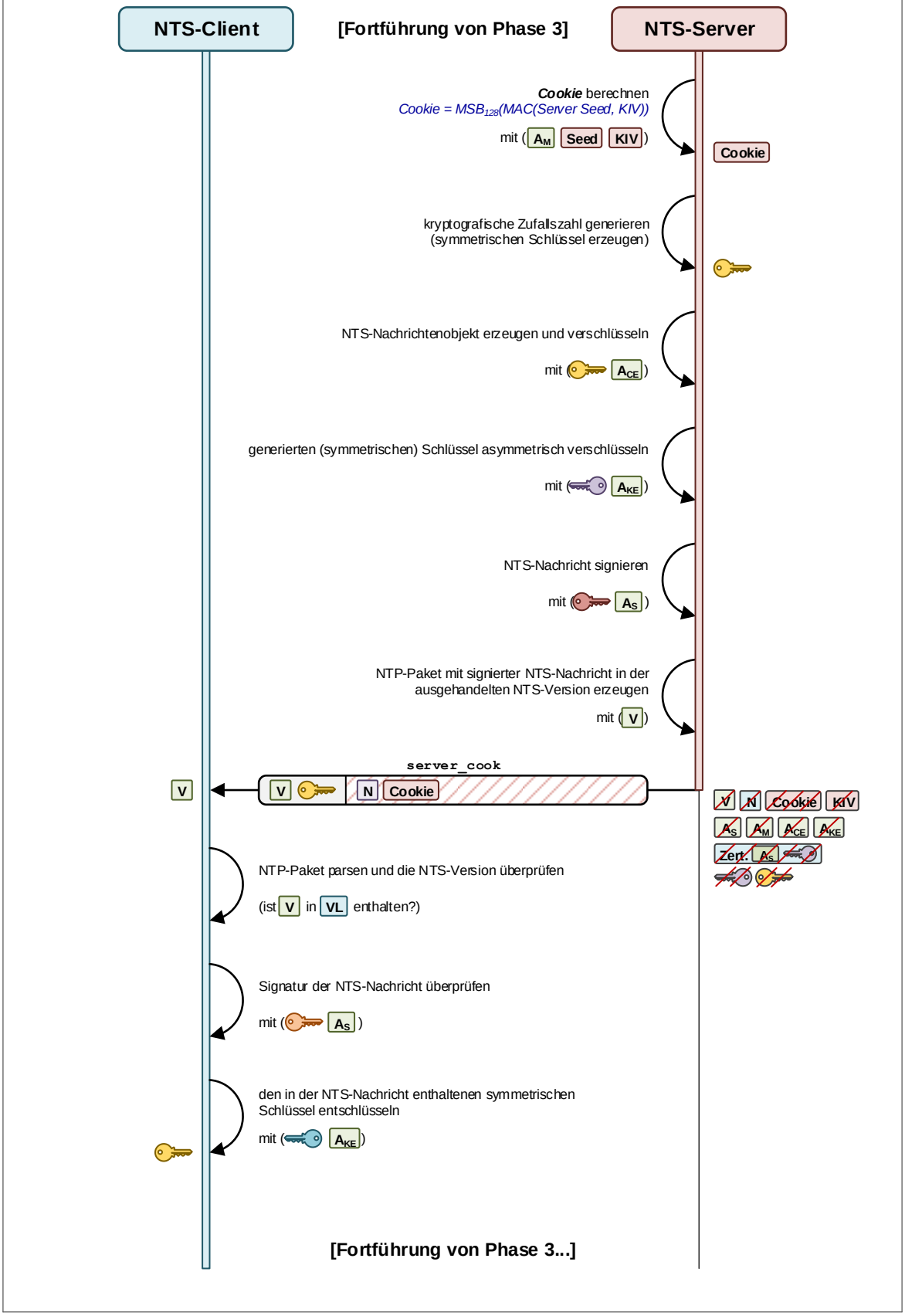
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 2/4 - Zugriffsnachrichten - Seite 2/2)	Stand: 15.10.2016 (NTS-Draft 15)



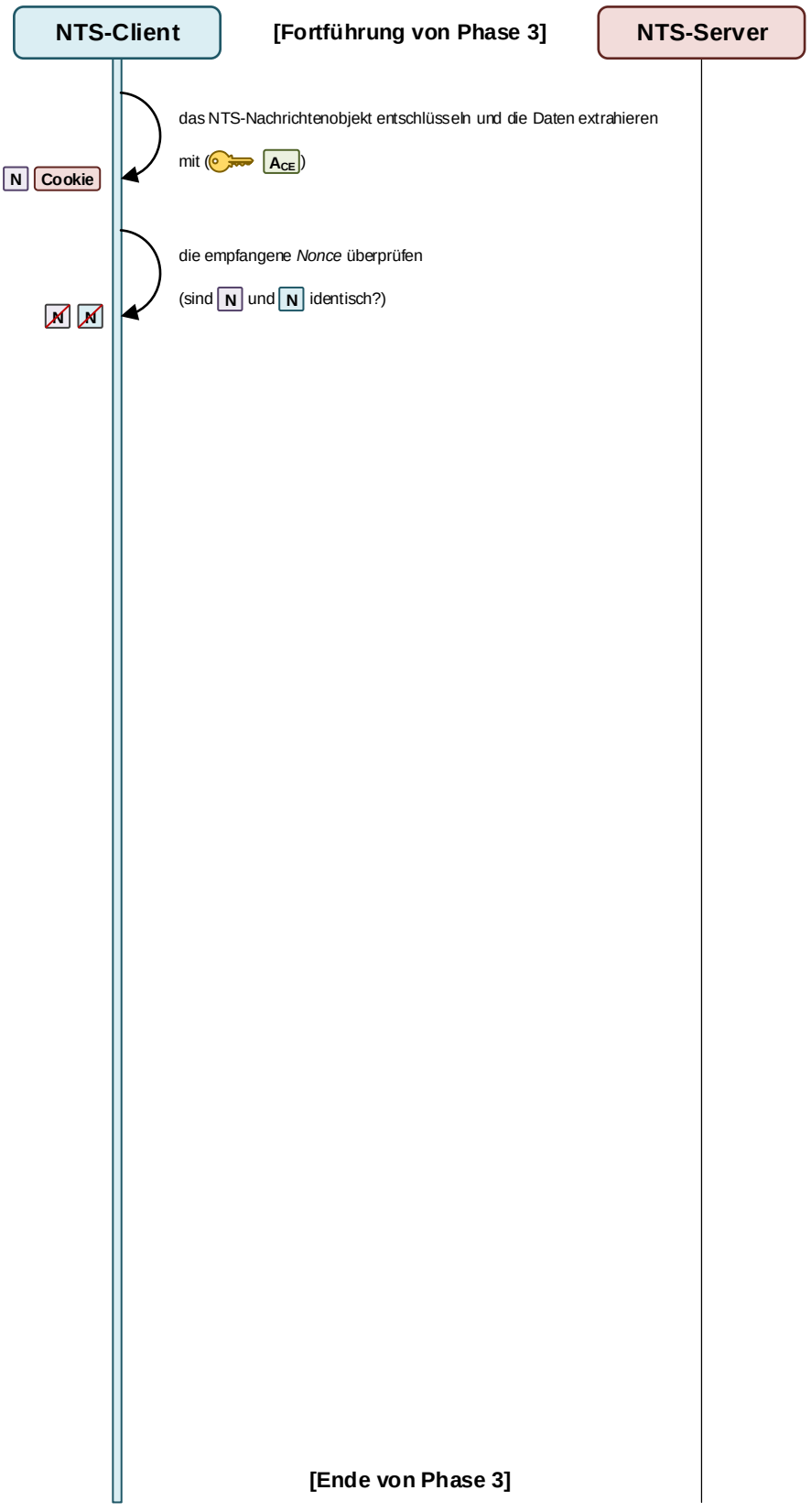
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 3/4 - Zugriffsnachrichten - Seite 1/3)	Stand: 15.10.2016 (NTS-Draft 15)



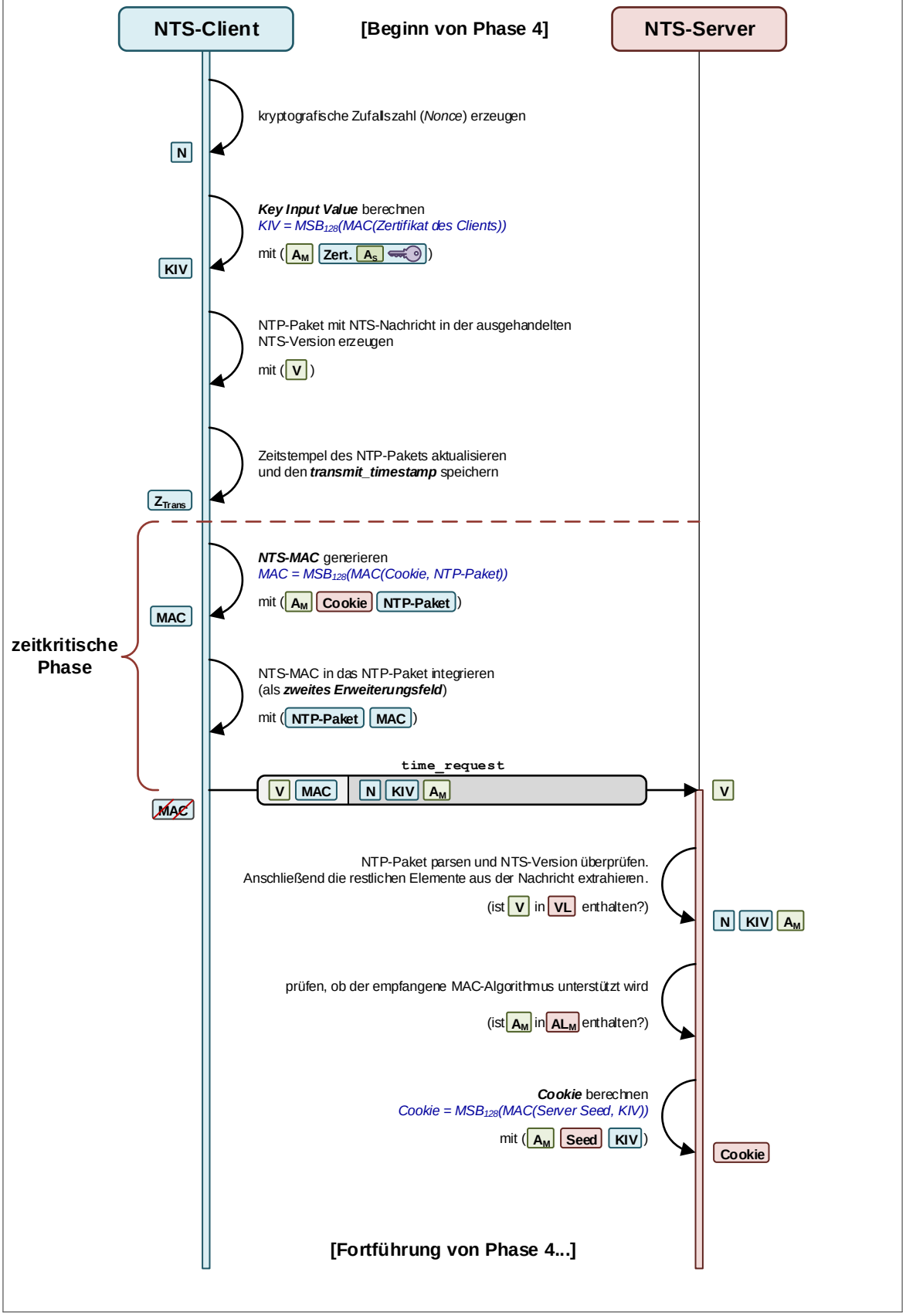
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 3/4 - Zugriffsnachrichten - Seite 2/3)	Stand: 15.10.2016 (NTS-Draft 15)



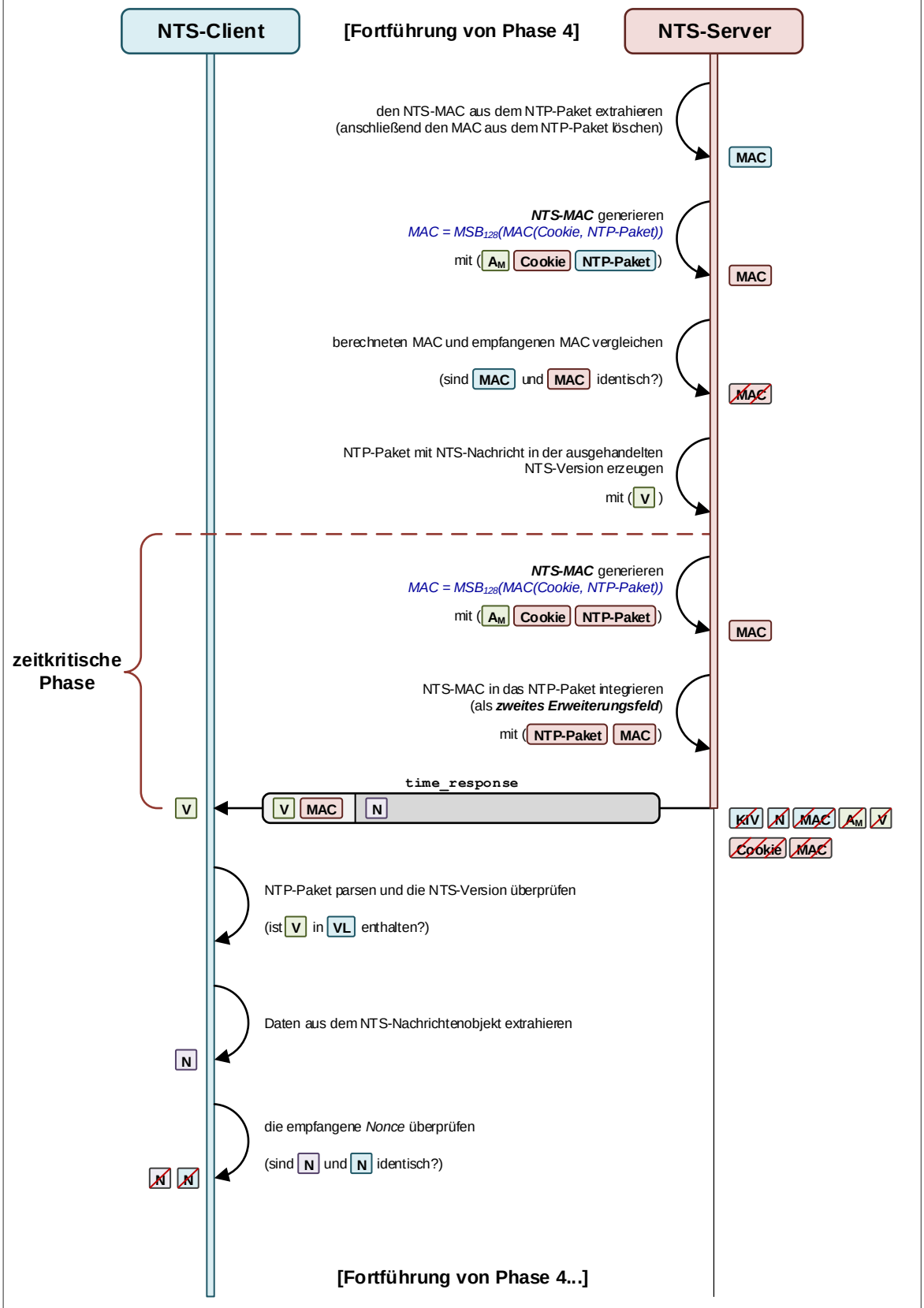
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 3/4 - Zugriffsnachrichten - Seite 3/3)	Stand: 15.10.2016 (NTS-Draft 15)



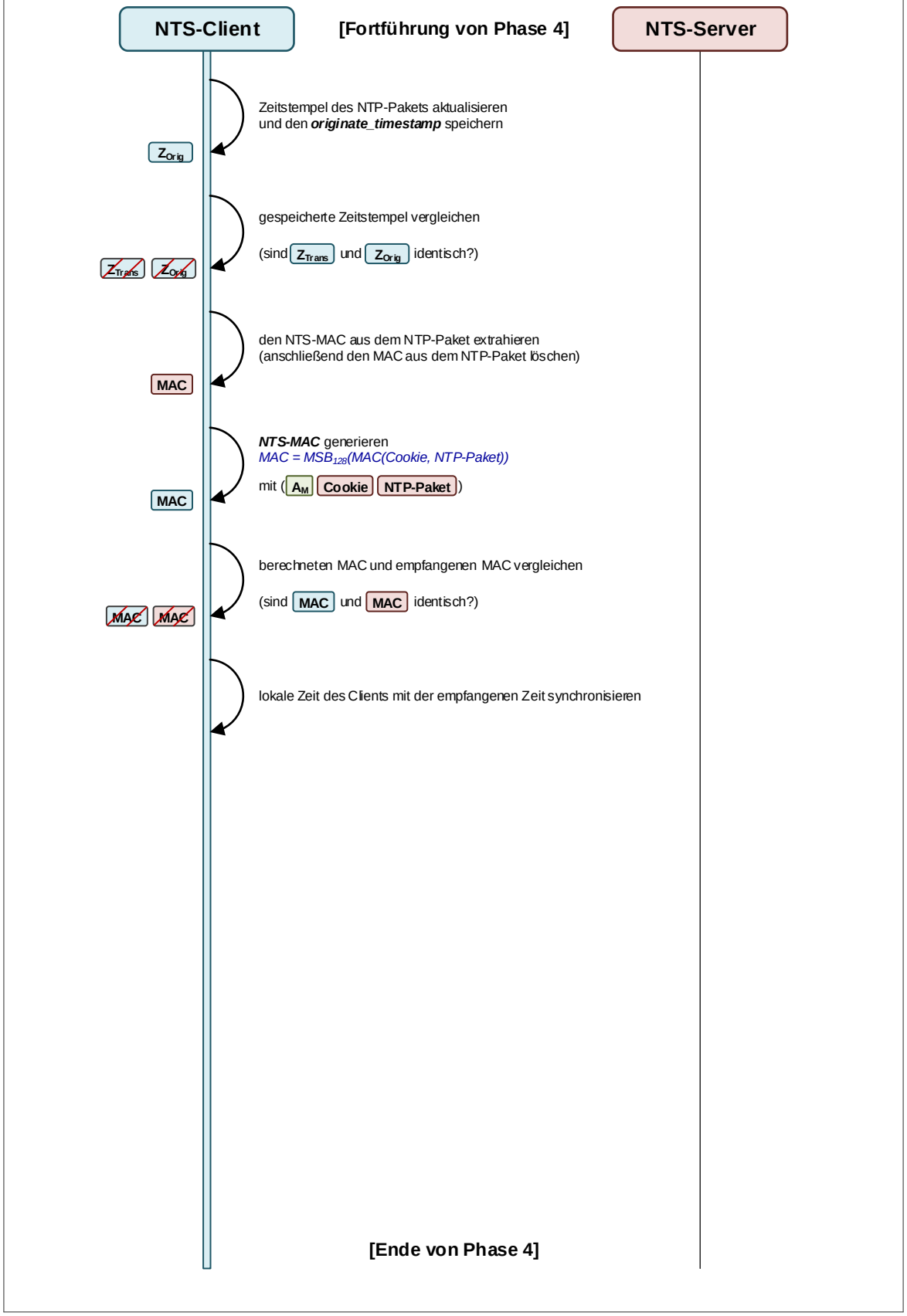
NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 4/4 - Zugriffsnachrichten - Seite 1/3)	Stand: 15.10.2016 (NTS-Draft 15)



NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 4/4 - Zugriffsnachrichten - Seite 2/3)	Stand: 15.10.2016 (NTS-Draft 15)



NTS-Protokoll (Unicast-Kommunikation)	**Sequenzdiagramm** (Phase 4/4 - Zugriffsnachrichten - Seite 3/3)	Stand: 15.10.2016 (NTS-Draft 15)



Ostfalia
Hochschule für angewandte Wissenschaften

Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb

Anlage IV:

Ablaufbeschreibung durch Aktivitätsdiagramme

Martin Langer
Matrikelnummer: 70405974
Eingereicht am: 21.10.2016

Ostfalia, Wolfenbüttel
Fakultät Elektrotechnik
Studiengang: Intelligente Mobilität und Energiesystemtechnik

Dokumentversion: 1.0.0.7
Letze Änderung: 18.10.2016 23:32:00

Die NTS-Ablaufbeschreibung mit Aktivitätsdiagrammen

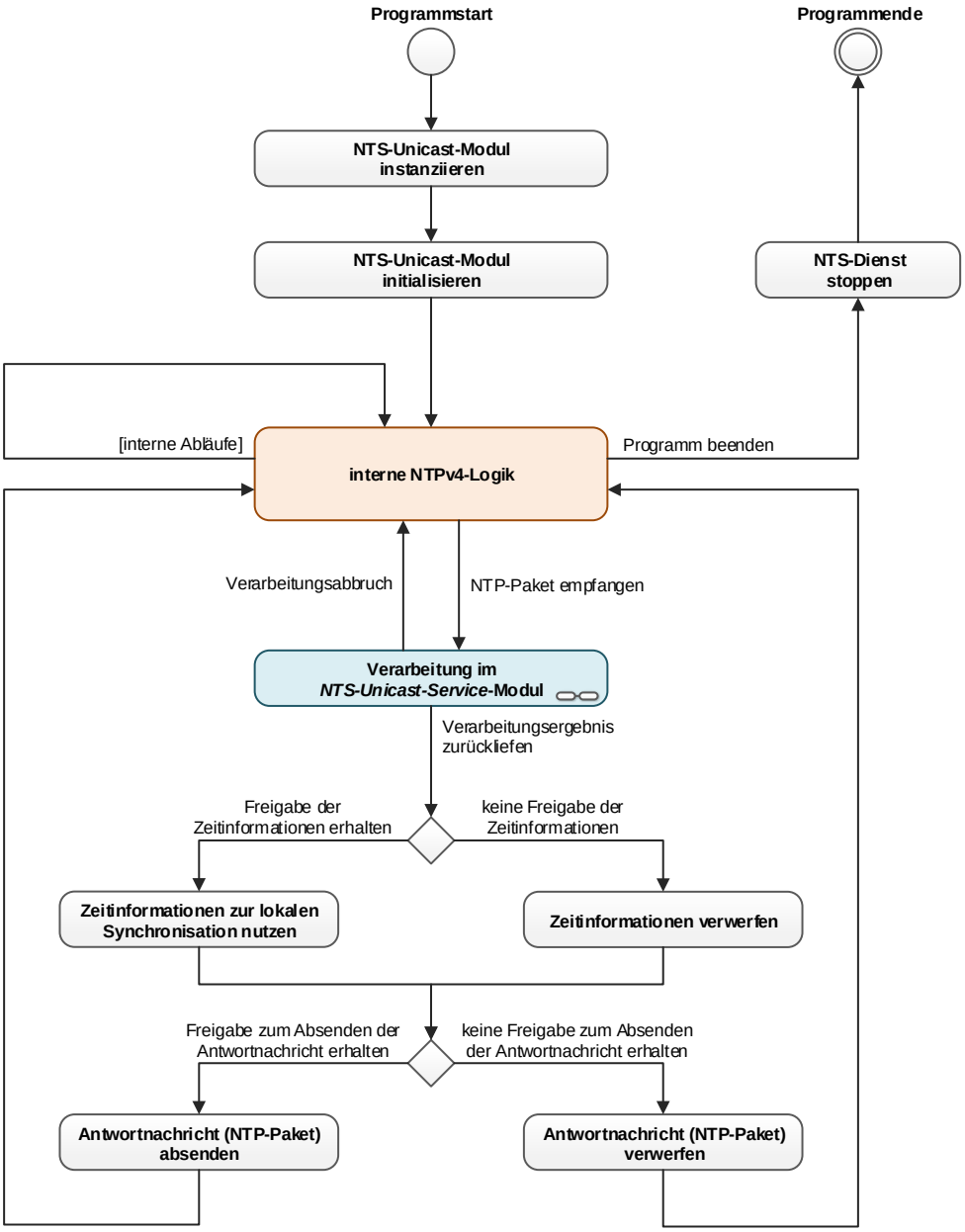
Die folgenden Aktivitätsdiagramme stellen den vereinfachten Ablauf des NTS-Protokolls dar. Der Fokus liegt hierbei auf die einzelnen Verarbeitungsschritte, die während einer Verarbeitungsroutine durchlaufen werden.

Die Grundlage der Aktivitätsdiagramme basiert auf den folgenden NTS-Dokumenten und auf die NTS-Implementierung:

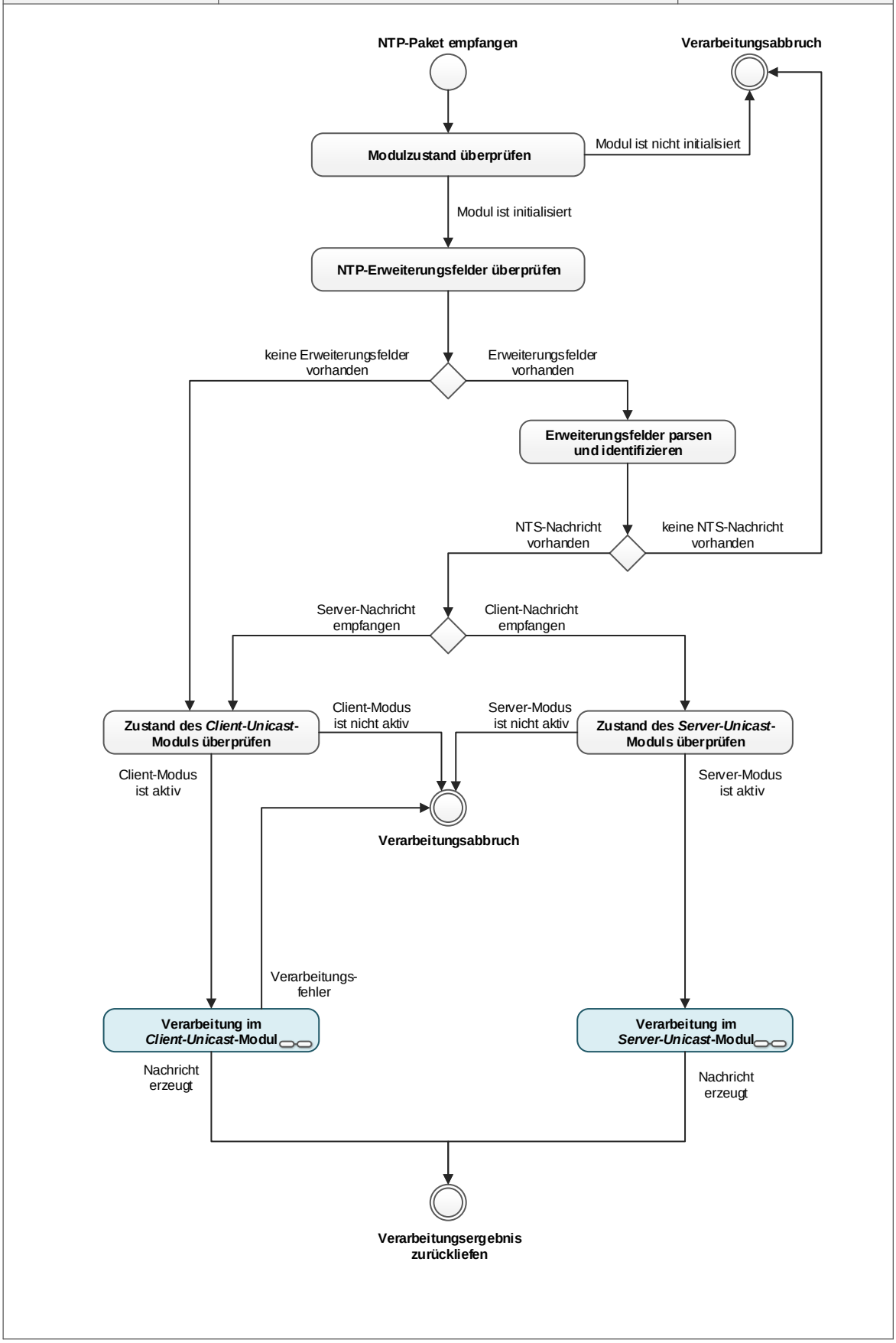
- *draft-ietf-ntp-network-time-security-15* (gültig bis: 26.03.2017)
- *draft-ietf-ntp-using-nts-for-ntp-06* (gültig bis: 26.03.2017)
- *draft-ietf-ntp-cms-for-nts-message-07* (gültig bis: 22.09.2016)

- NTS-Implementierungsstand: v0.8.0.0 Beta (vom: 27.08.2016)

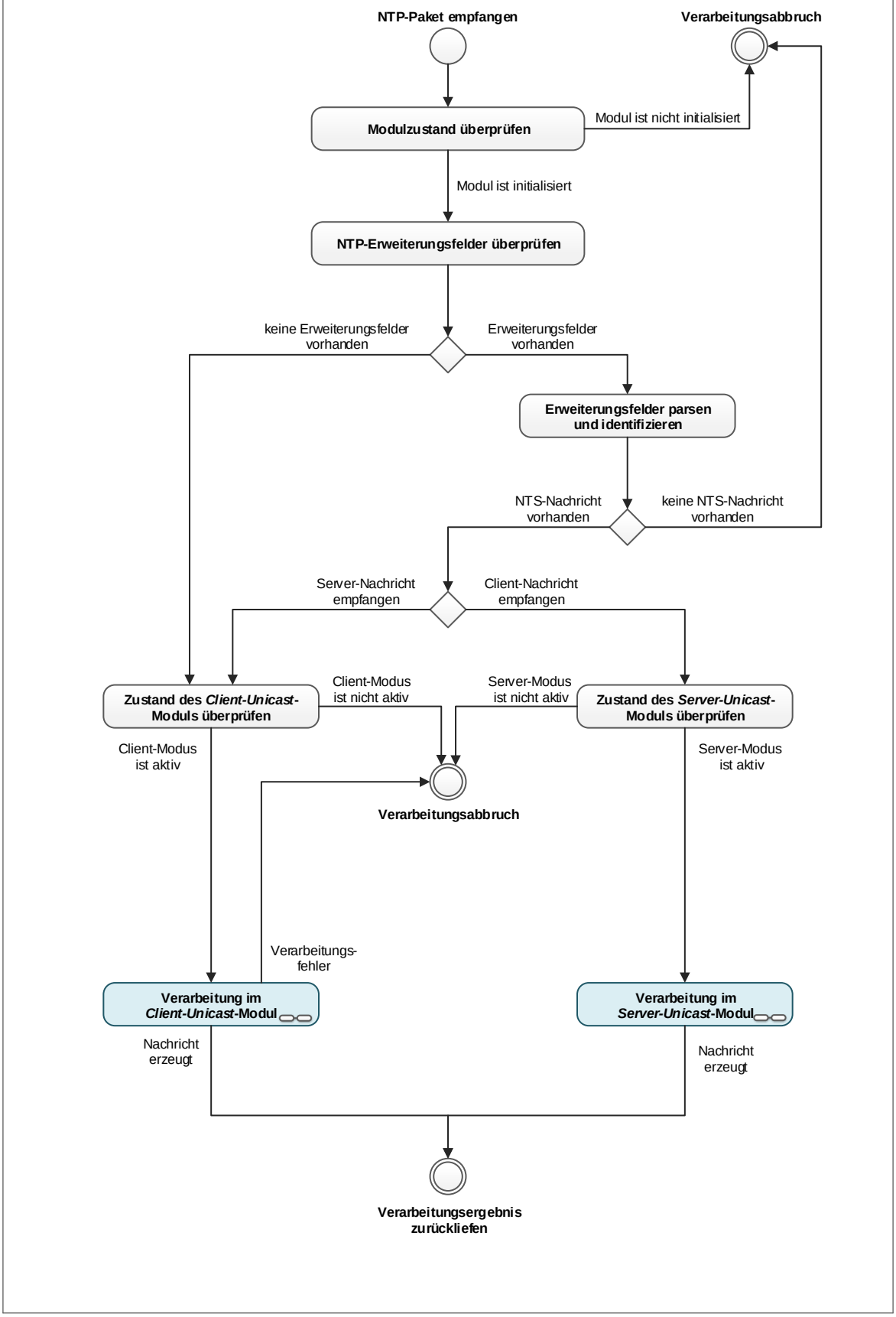
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (Hauptstruktur: NTP-Ebene)	Stand: 18.10.2016 (NTS-Draft 15)



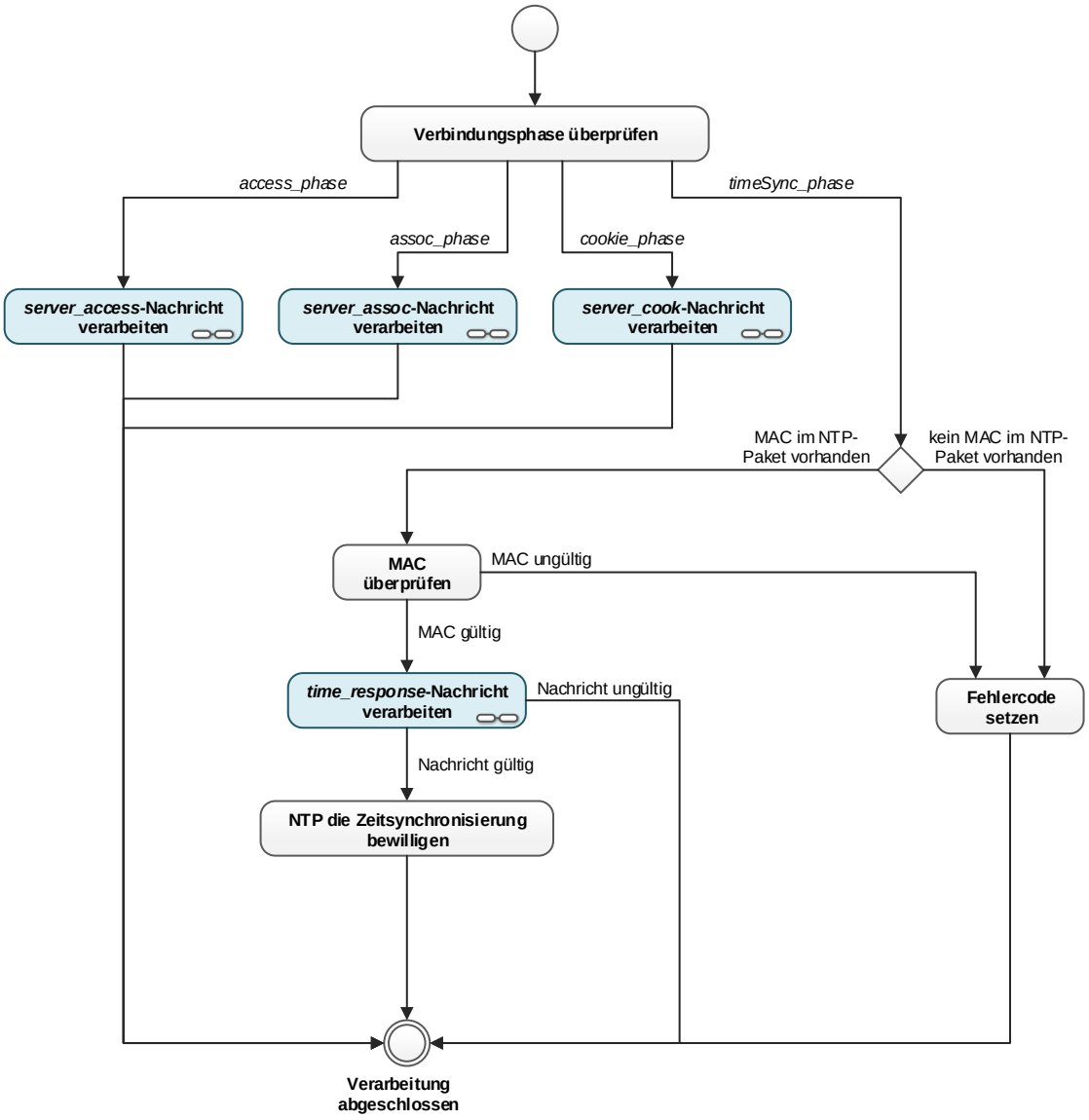
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*NTS-Unicast-Service*-Modul)	Stand: 18.10.2016 (NTS-Draft 15)



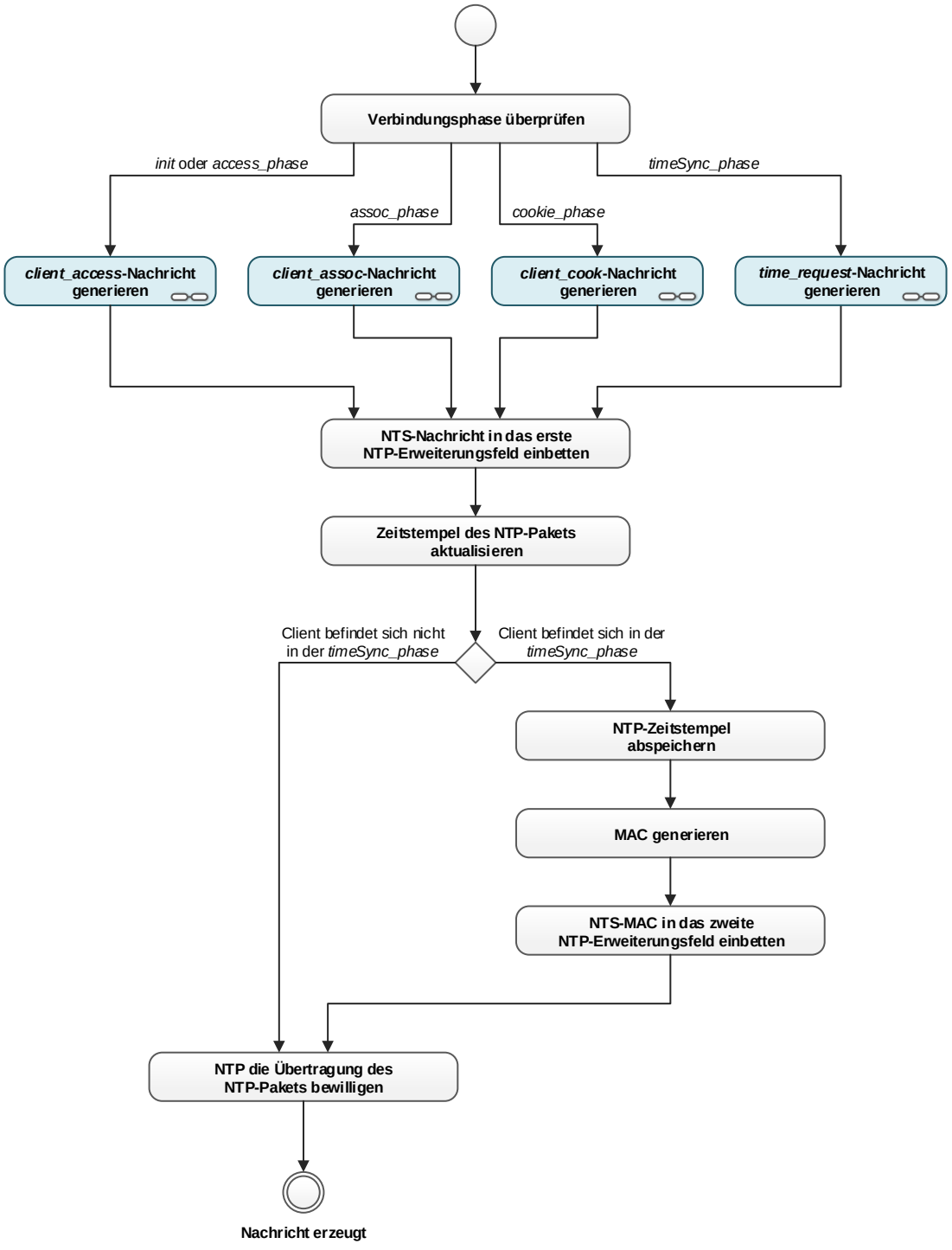
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*NTS-Unicast-Service*-Modul)	Stand: 18.10.2016 (NTS-Draft 15)



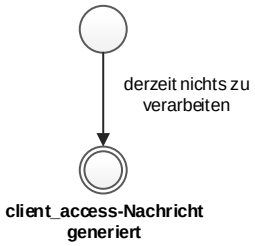
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: Nachrichten verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



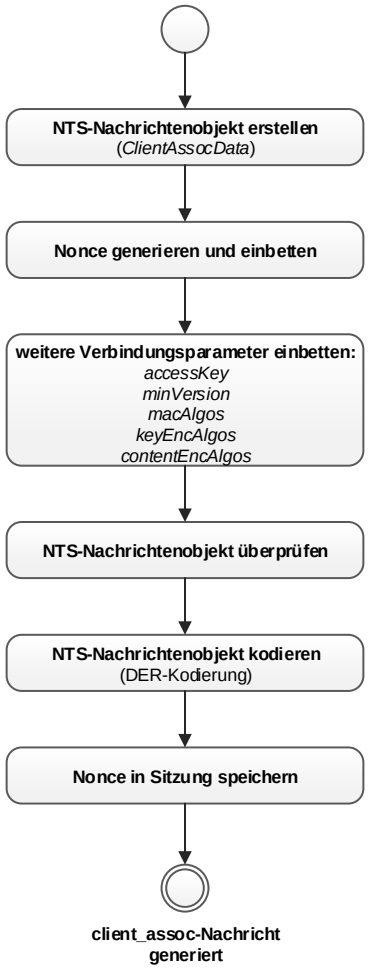
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: Nachrichten generieren)	Stand: 18.10.2016 (NTS-Draft 15)



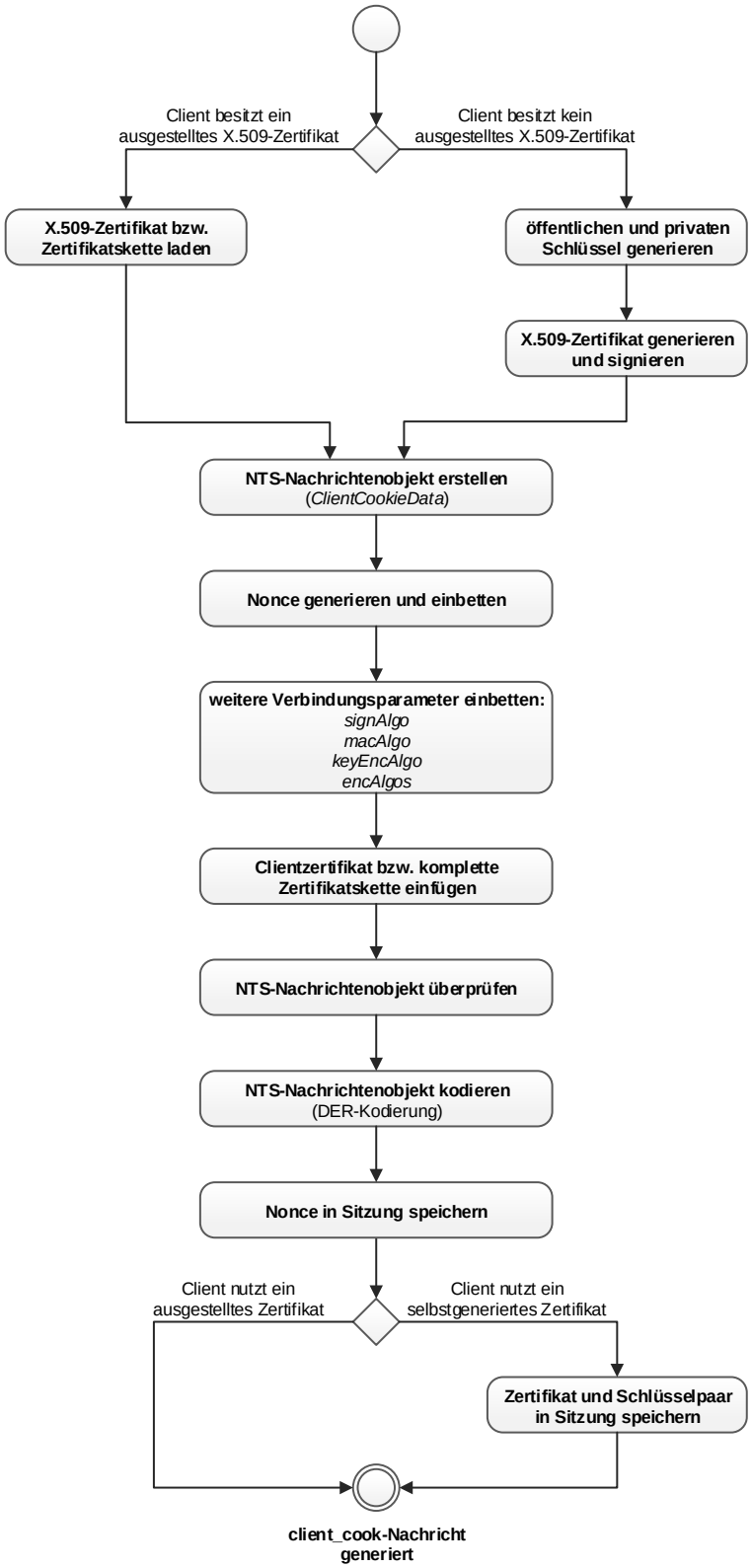
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: client_access generieren)	Stand: 18.10.2016 (NTS-Draft 15)



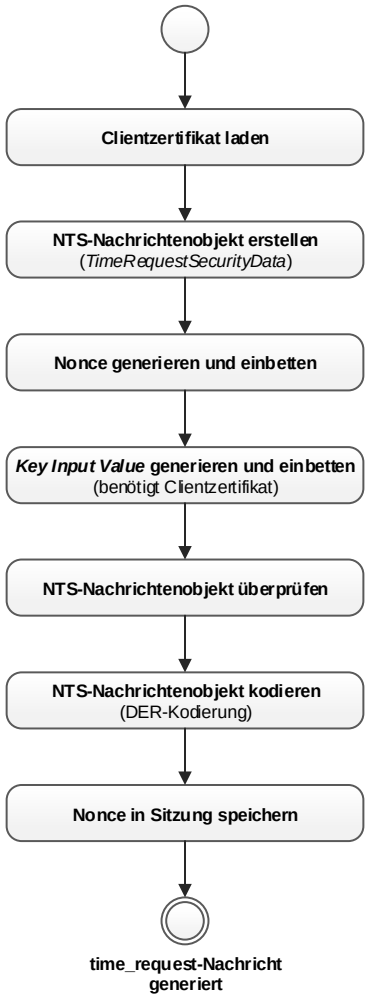
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: client_assoc generieren)	Stand: 18.10.2016 (NTS-Draft 15)



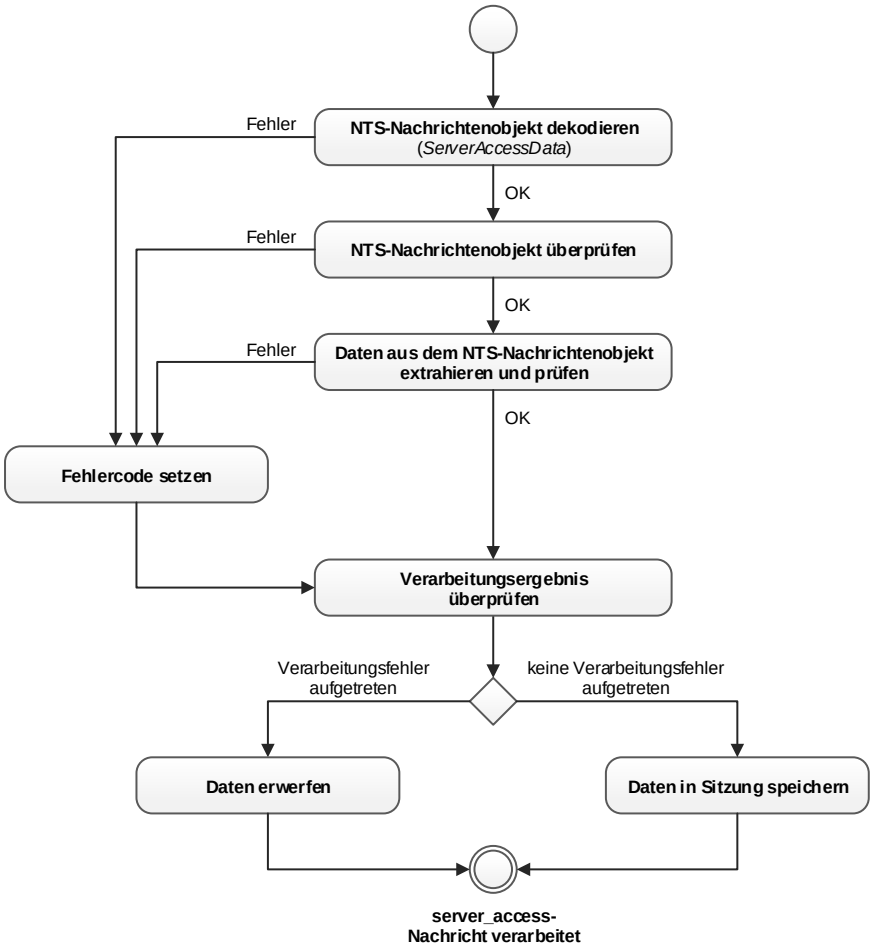
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: client_cook generieren)	Stand: 18.10.2016 (NTS-Draft 15)



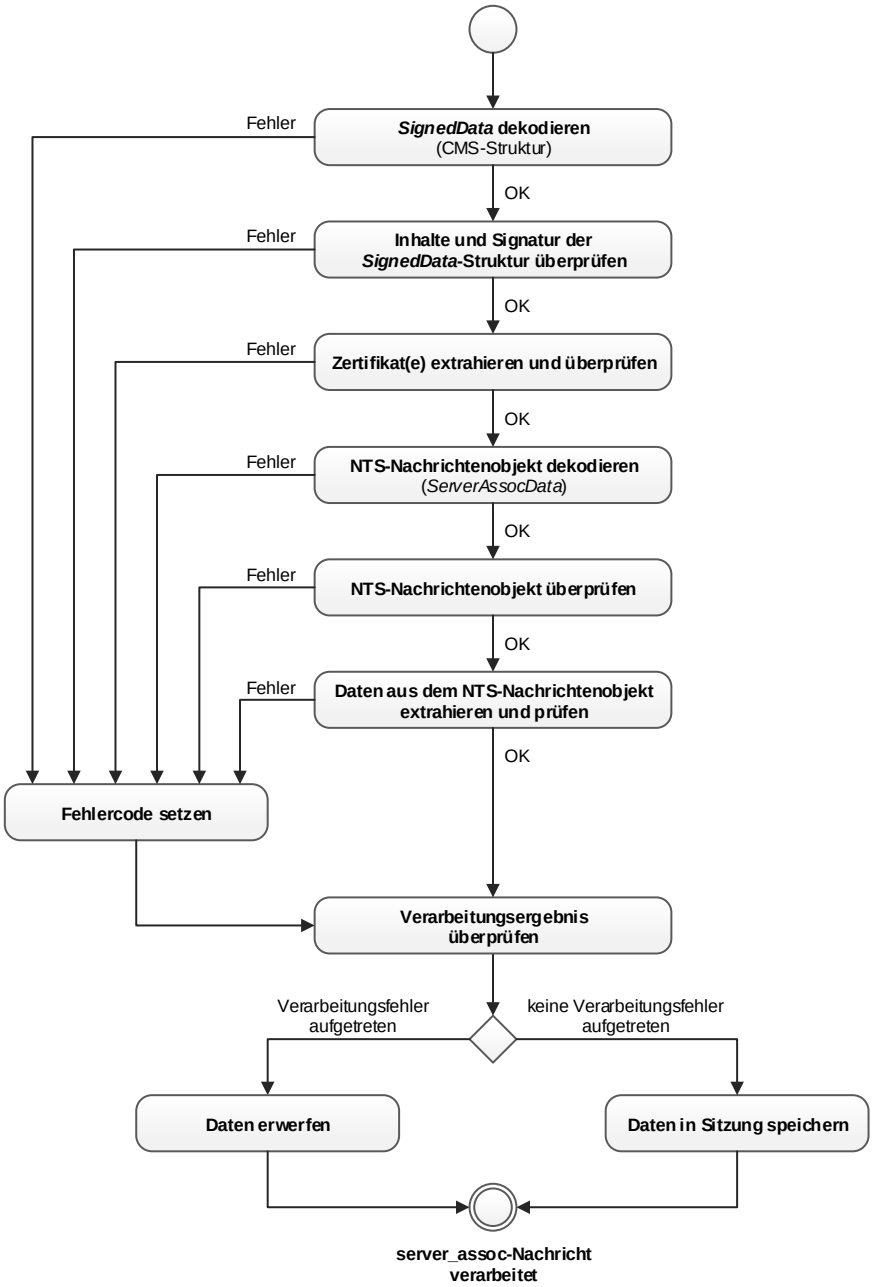
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: time_request generieren)	Stand: 18.10.2016 (NTS-Draft 15)



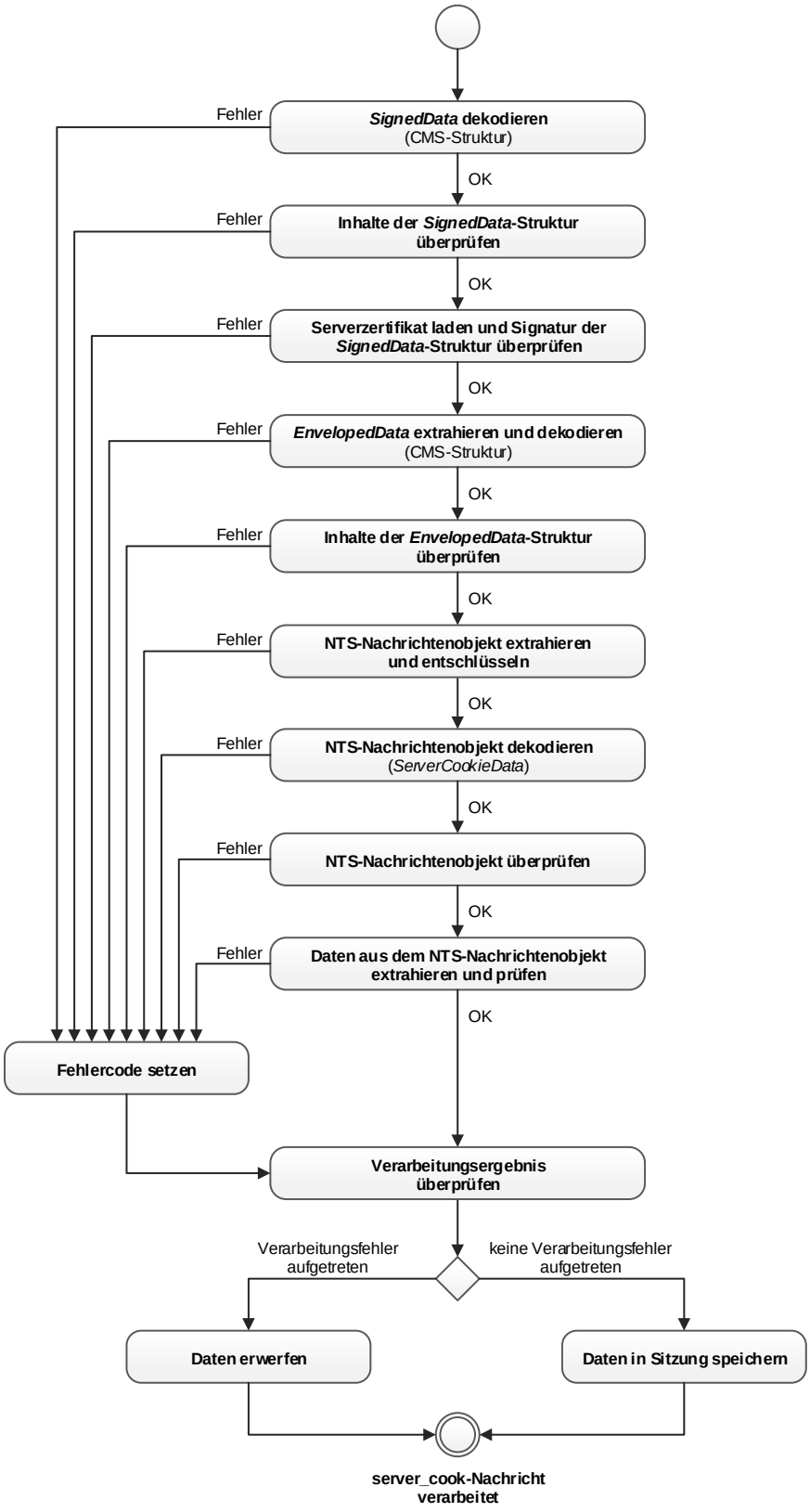
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: server_access verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



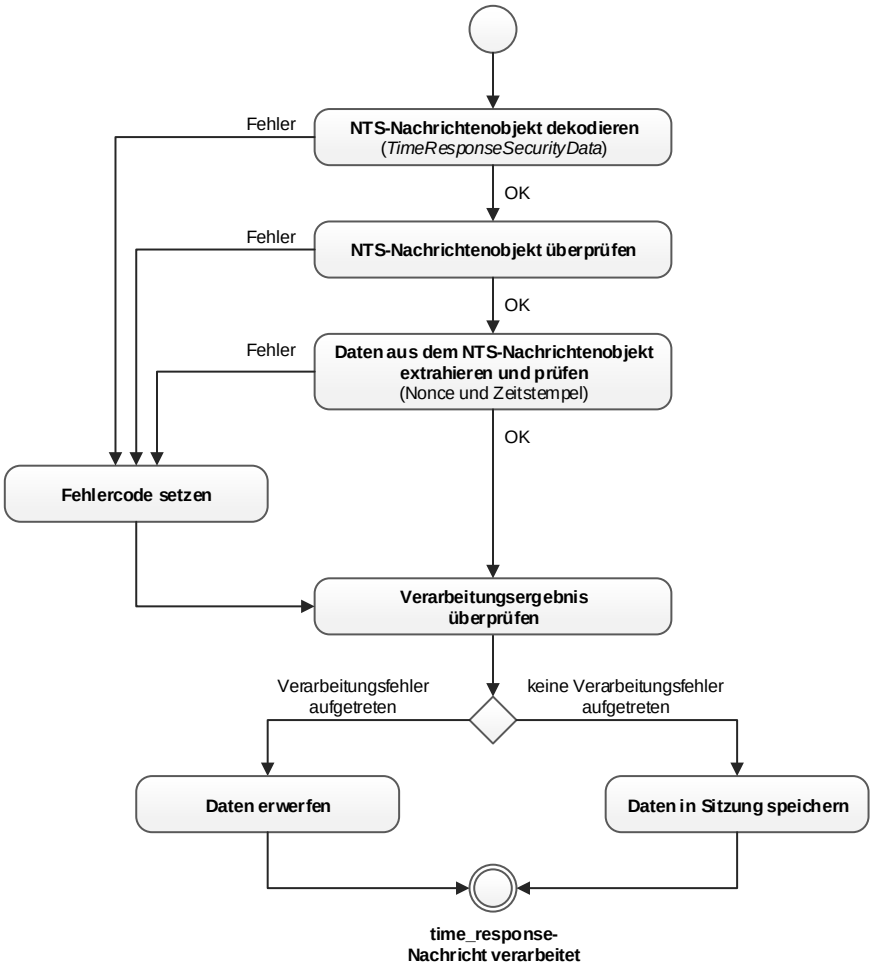
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: server_assoc verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



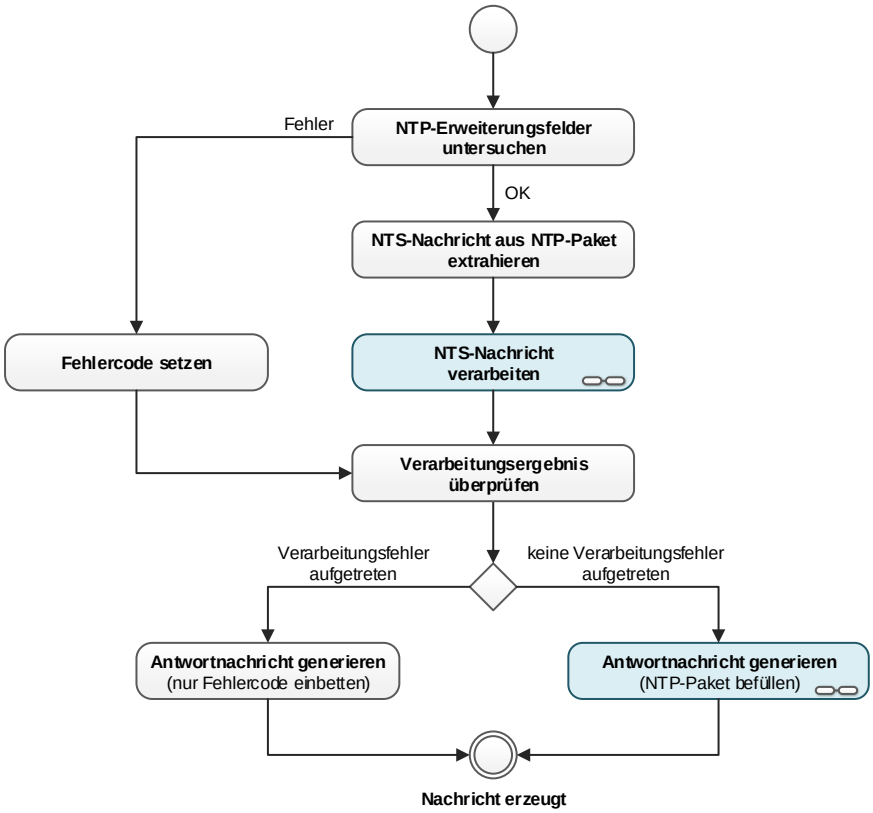
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: server_cook verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



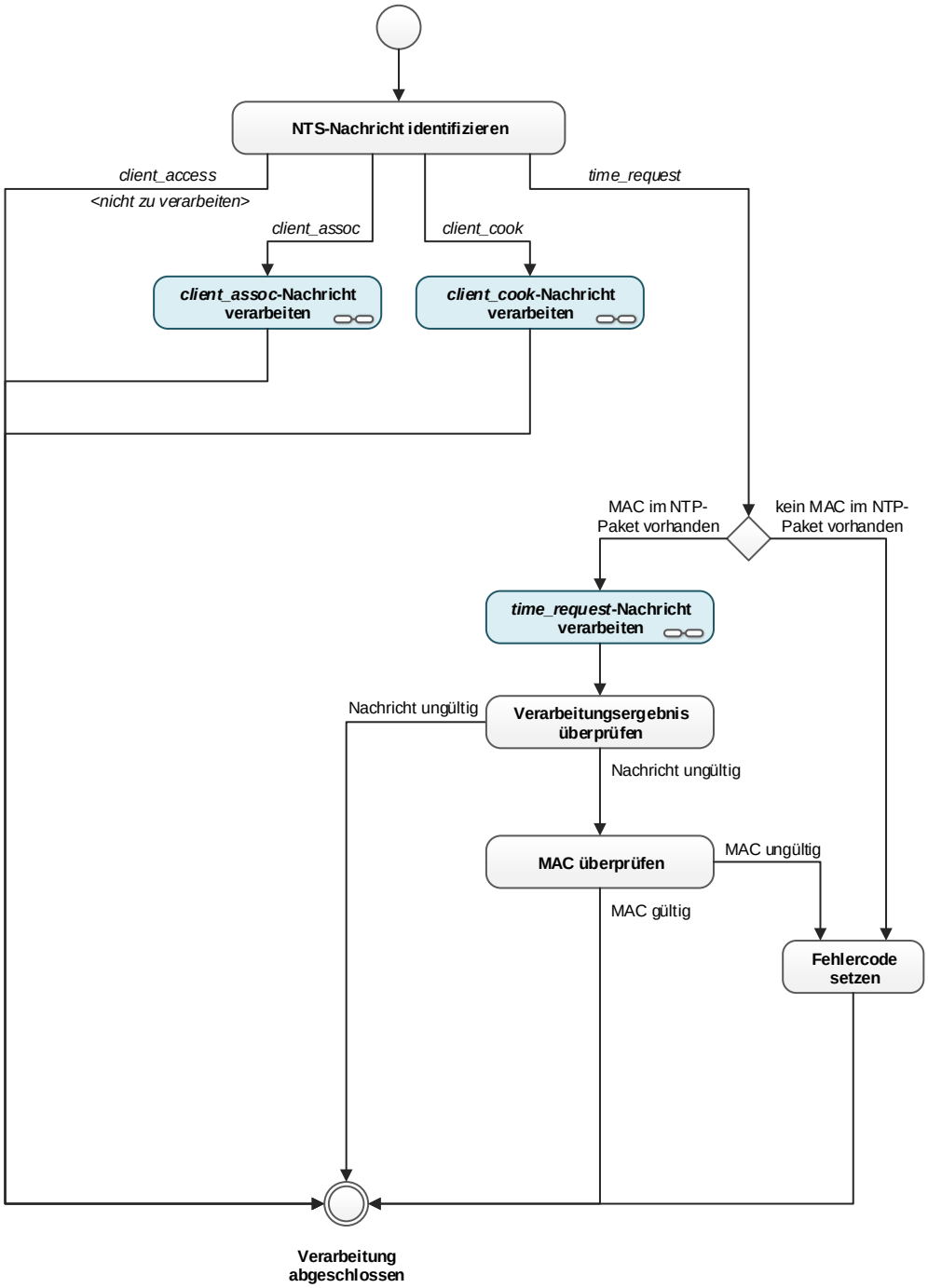
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Client-Unicast*-Modul: time_response verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



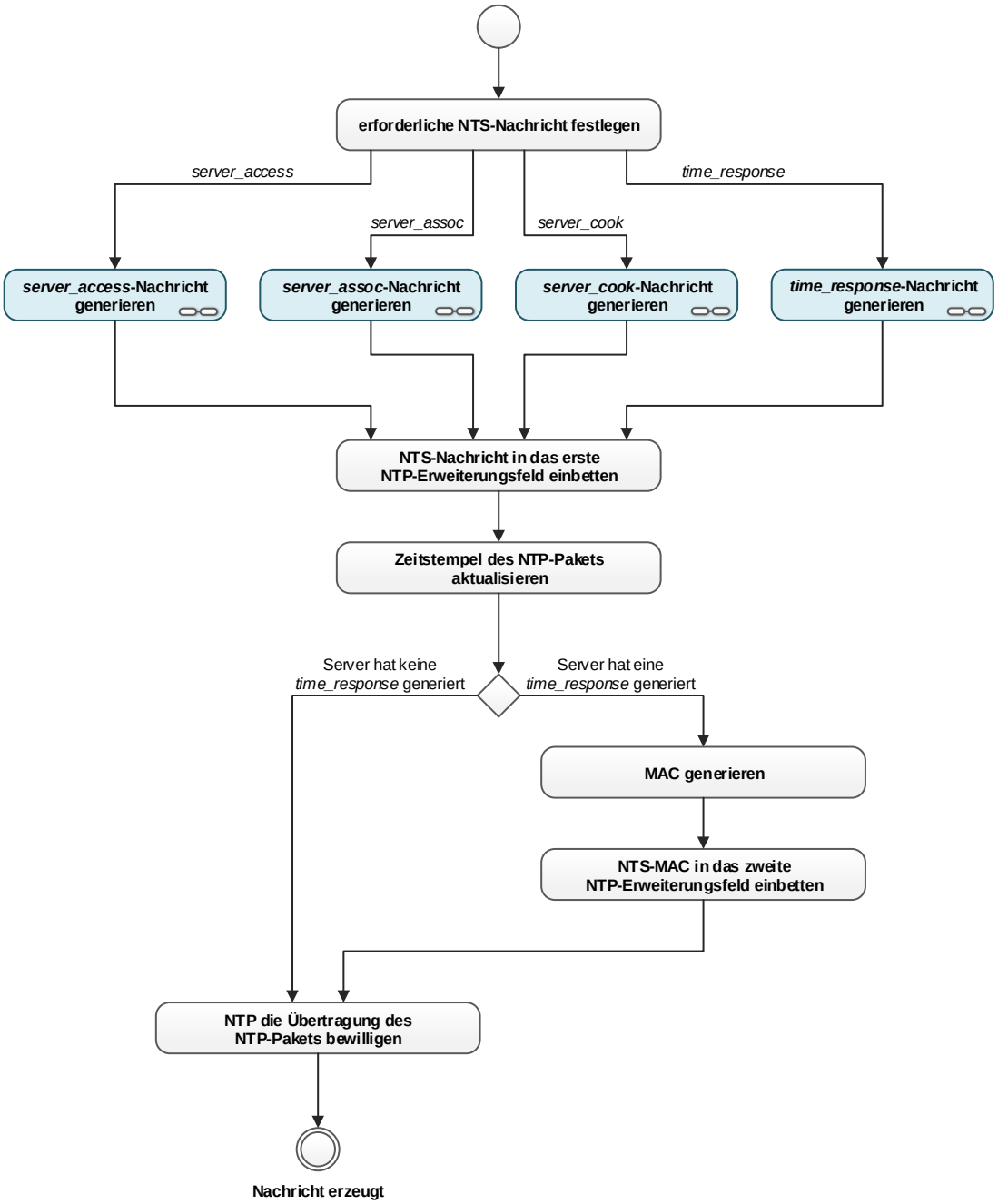
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: Hauptansicht)	Stand: 18.10.2016 (NTS-Draft 15)



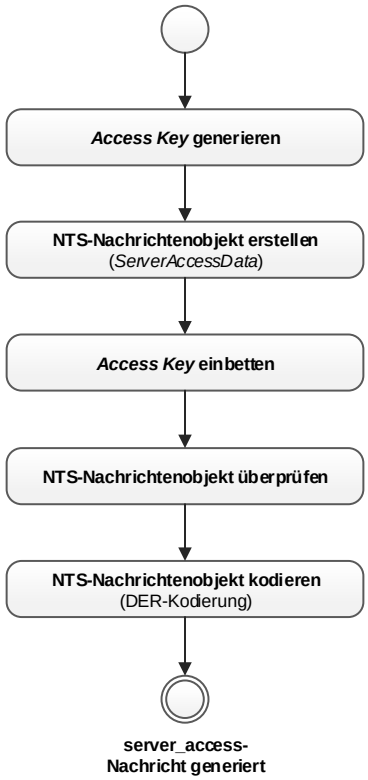
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: Nachrichten verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



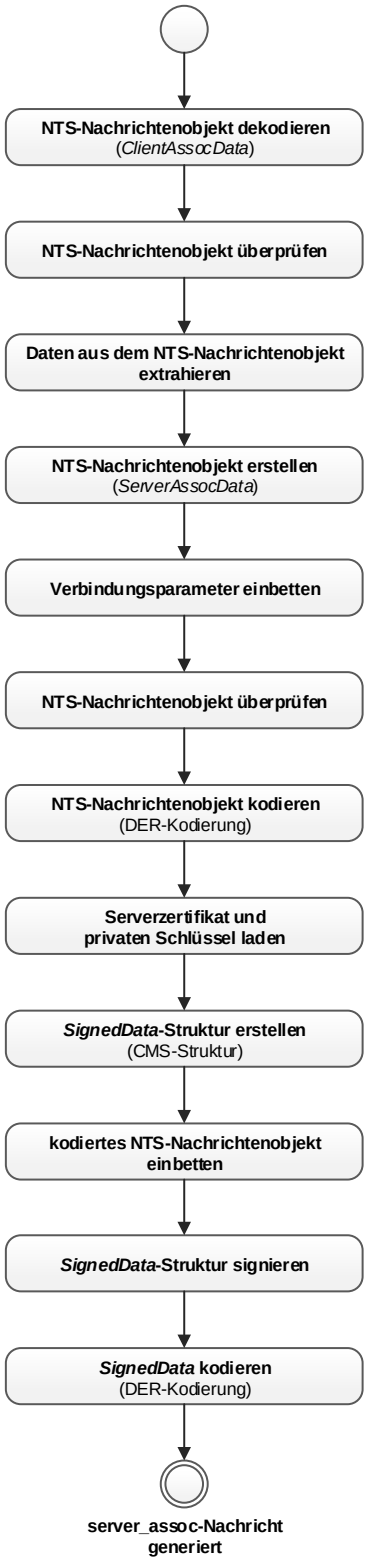
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: Nachrichten generieren)	Stand: 18.10.2016 (NTS-Draft 15)



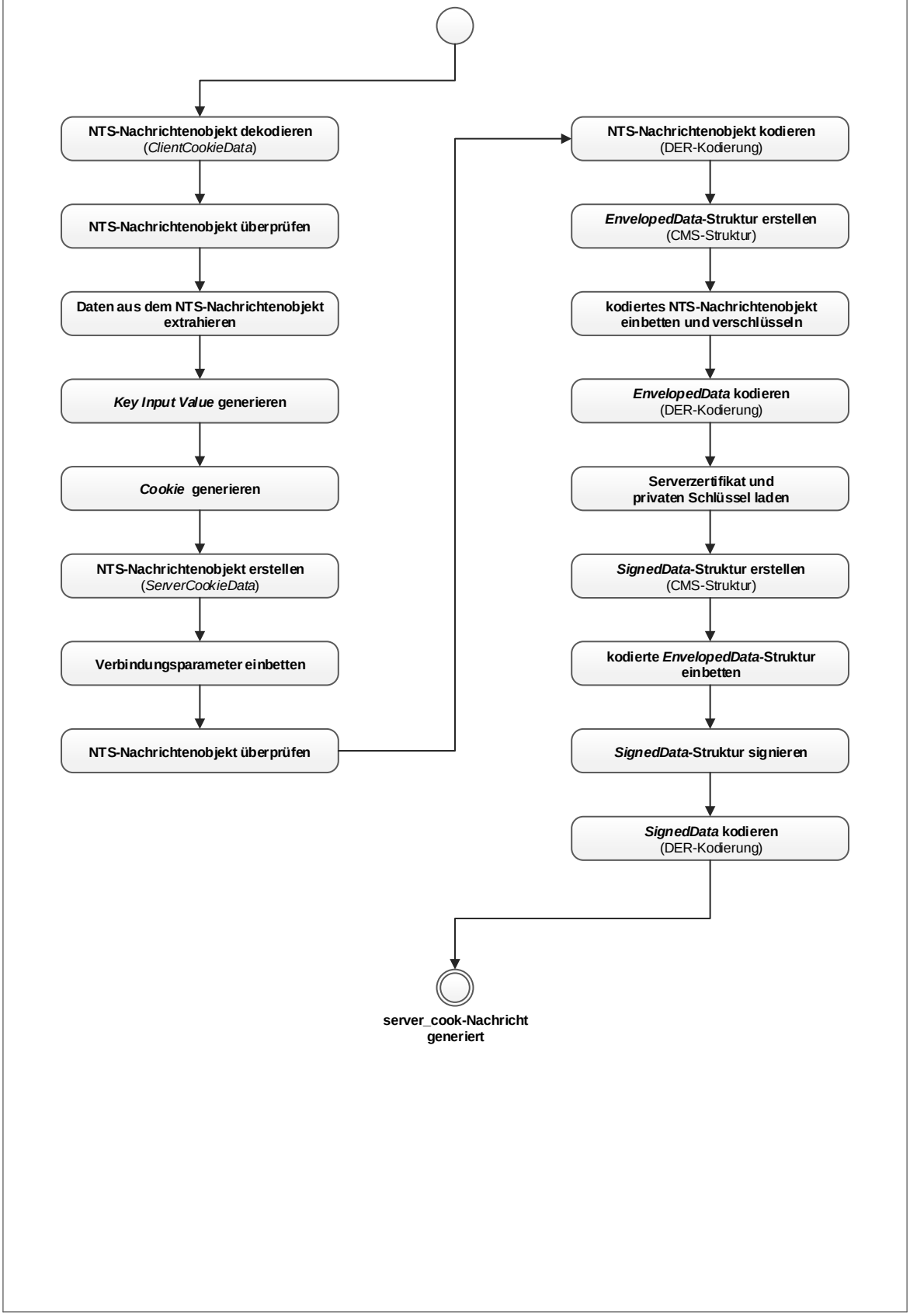
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: server_access generieren)	Stand: 18.10.2016 (NTS-Draft 15)



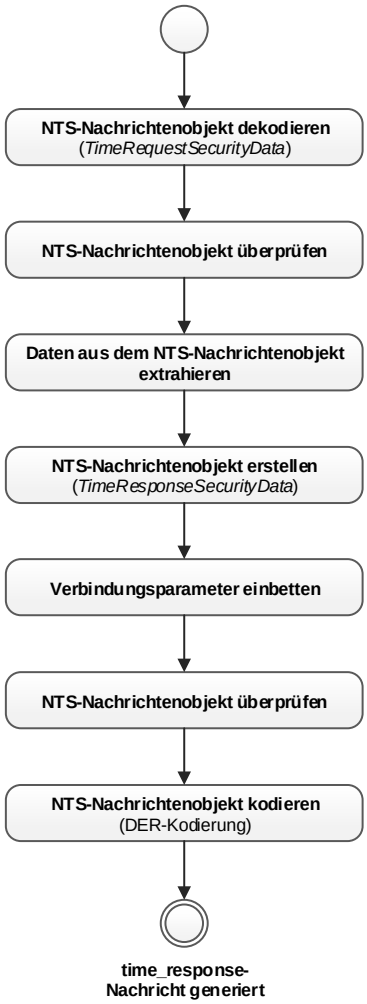
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: server_assoc generieren)	Stand: 18.10.2016 (NTS-Draft 15)



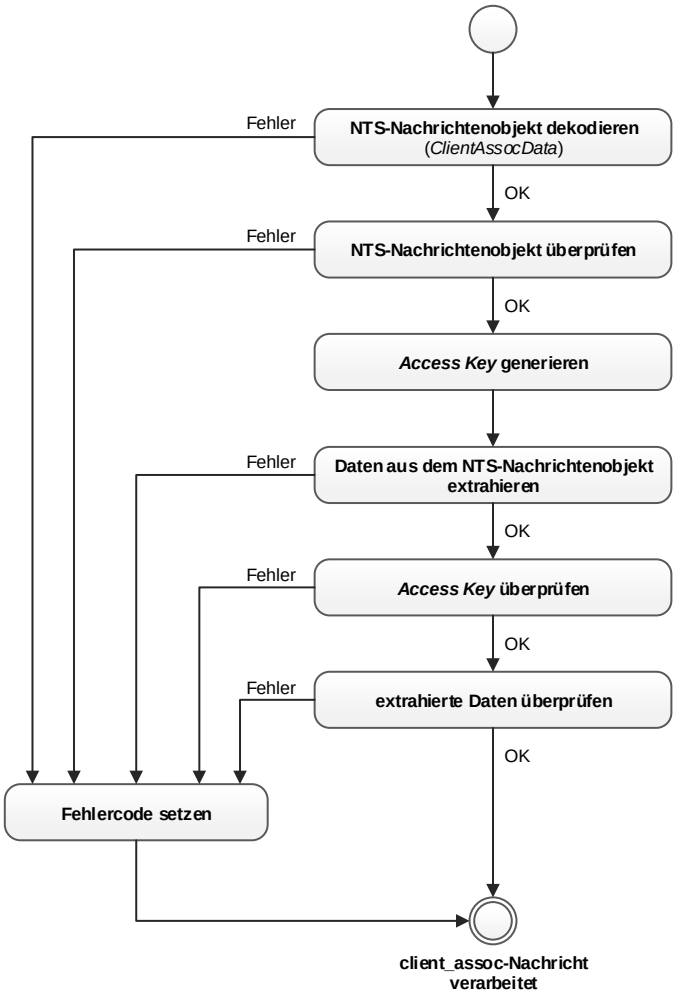
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: server_cook generieren)	Stand: 18.10.2016 (NTS-Draft 15)



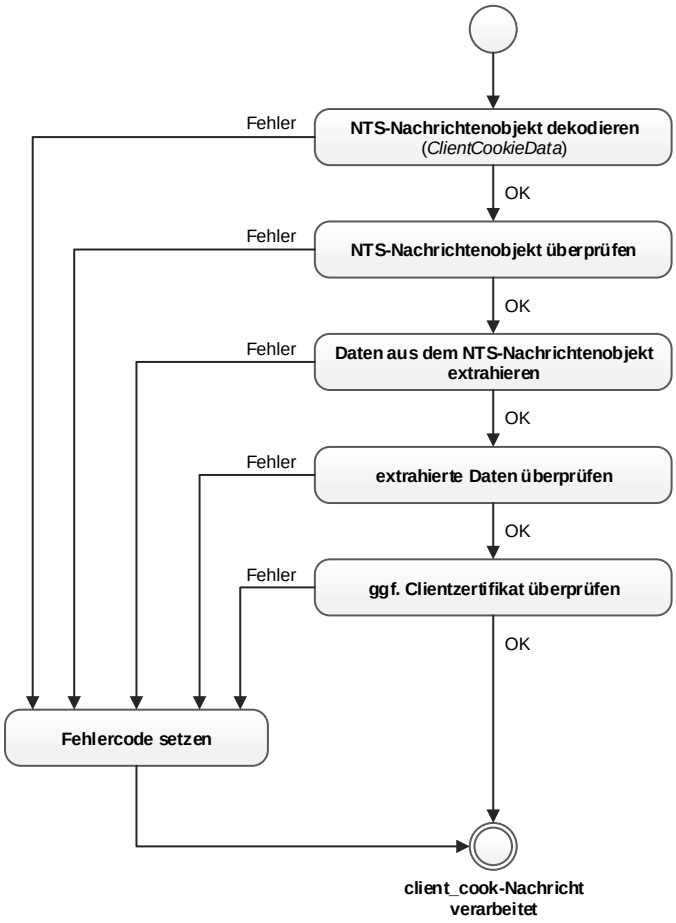
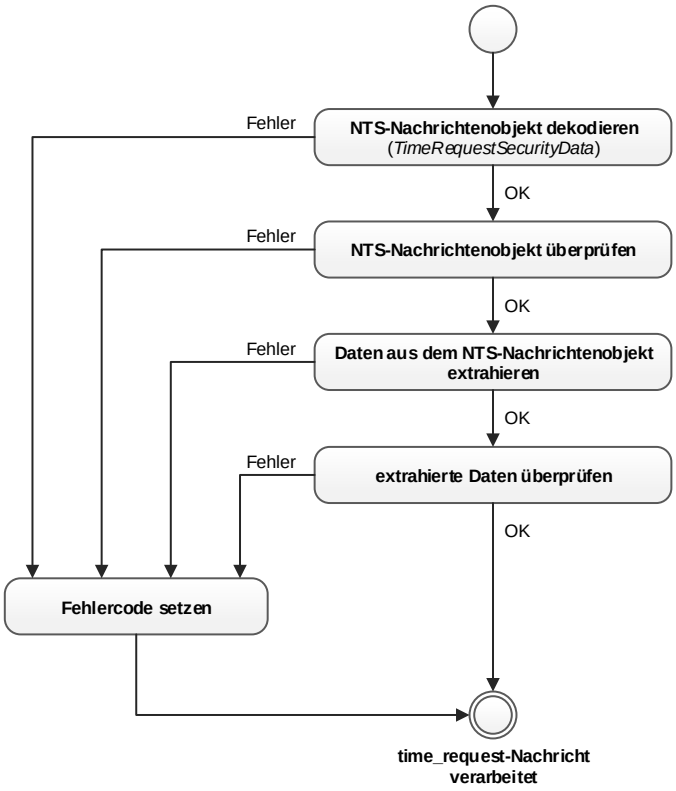
NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: time_response generieren)	Stand: 18.10.2016 (NTS-Draft 15)



NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: client_assoc verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: client_cook verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)



NTS-Protokoll (Unicast-Kommunikation)	**Aktivitätsdiagramm** (*Server-Unicast*-Modul: time_request verarbeiten)	Stand: 18.10.2016 (NTS-Draft 15)