Forschungsbericht SS 2016

Network Time Security (NTS) – Unicast-Modus über NTP (Teil 1)

Prof. Dr.-Ing. Rainer Bermbach

Einleitung

Moderne vernetzte Systeme benötigen in den meisten Fällen eine möglichst genaue Uhrzeit, um ihre Aufgaben korrekt erfüllen zu können. Standards sind hierfür heute das Network Time Protocol NTP und für genauere Anwendungen das Precision Time Protocol PTP. Problematisch dabei ist, dass diese Protokolle im Allgemeinen nicht ausreichend gegen Angriffe und Manipulationen geschützt sind, so dass sicherheitskritische Aufgaben durch zusätzliche Mechanismen abgesichert werden müssen, beispielsweise die Kommunikation im Smart Grid, dem intelligenten Stromnetz. Dort benötigen z.B. die sog. Smart Meter Gateways eine verlässliche Uhrzeit, um den Verbrauch von Energie etc. exakt erfassen zu können (Zeitstempelung, insbesondere bei zeitabhängigen Tarifen). Daher erfolgt nach den bisherigen Vorgaben des Bundesamtes für Sicherheit in der Informationstechnik (BSI) [1] und der Physikalisch-Technischen Bundesanstalt (PTB) [2] die Zeitübertragung zwischen den Smart Meter Gateways und den zugehörigen Gateway-Administratoren über TLS-verschlüsselte Kanäle. Die Administratoren beziehen ihrerseits im derzeitigen Versuchsbetrieb ihre Informationen über eine symmetrisch gesicherte Verbindung von den Zeitservern der PTB. Dazu ist es nötig, jedem Administrator einen Schlüssel für die gesicherte Kommunikation zukommen zu lassen. Dies ist im tatsächlichen Betrieb auf Dauer nicht praktikabel.

Das zur Sicherung von NTP bislang empfohlene Autokey-Verfahren weist gravierende Sicherheitsmängel auf [3], weshalb Sibold (PTB), Röttger (Google) und Teichel (PTB) das Protokoll Network Time Security (NTS) vorgeschlagen haben, das momentan als IETF Draft [4] (bzw. in Kürze als RFC) in der Diskussion und Weiterentwicklung ist. Ziel ist es, eine sichere Authentifizierung des Zeitdienstes sowie eine Überprüfung der Integrität der Zeitinformation zu erreichen.

Grundsätzlich sollten Sicherungsmechanismen die zur Übertragung benötigte Bandbreite und vor allem die Latenz der Zeitinformation nur geringfügig erhöhen. Sollen die bisherigen Protokolle NTP [5] und PTP [6] weiterhin Verwendung finden, ist darauf zu achten, dass diese per UDP kommunizieren und daher ein zustandsloses Protokoll mit geringem Overhead auf der Zeitserverseite implementieren. Klassische Verfahren wie TLS benötigen TCP. Dort müsste der Server für jede Verbindung den Kommunikationszustand speichern, was bei Massen von Nutzern nicht praktikabel ist. Daher verwendet NTS eine zustandslose Implementierung beim Server und achtet darauf, nur geringen Overhead für die Datenübertragung hinzuzufügen und den benötigten Rechenaufwand gering zu halten. Damit sind auch kleinere Embedded Systeme in der Lage, das Protokoll zu nutzen.

NTS erlaubt dem Client, den Zeitserver zu authentifizieren sowie die Integrität der Zeitprotokollpakete mittels MAC (Message Authenticati-

on Code) zu überprüfen. Die Zeitpakete sind jedoch nicht verschlüsselt. Der Client ist über spezielle Maßnahmen in der Lage, zu überprüfen, ob eine Antwort zu der von ihm gestellten Anfrage gehört (Request-Response-Konsistenz) und ob sie unverändert bei ihm angekommen ist. NTS wird über Daten in sog. Extension Fields der Standardprotokolle (NTP, PTP) implementiert [7], so dass die Sicherungsmaßnahmen die Zeitsynchronisation von Teilnehmern in ungesichertem Betrieb nicht negativ beeinflussen.

NTS kennt zwei Betriebsarten: Unicast und Broadcast. Im Unicast Mode (1:1-Betrieb) sendet der Zeitserver gesicherte Zeitinformationen zu dem anfragenden Klienten. Hier besteht also eine – wenn auch serverseitig zustandslose – Verbindung zwischen den Kommunikationsteilnehmern. Im Broadcast-Betrieb (1 Sender – viele Empfänger, vgl. Rundfunk) empfängt der Client nach initialem Austausch regelmäßige, MAC-gesicherte Zeitinformationspakete, deren Authentizität er über das sog. TESLA-Protokoll (eine Art Einweg-Schlüsselkette) prüfen kann [8]. Der Broadcast-Betrieb wird aus Aufwandsgründen in dem Projekt nicht betrachtet. Ebenso konzentriert sich das Projekt auf das Zeitübertragungsprotokoll NTP.

Entwicklungsziele von NTS

Aufgrund der Sicherheitsmängel des für NTP entwickelten Sicherungsverfahrens *Autokey* und des davor eingesetzten Verfahrens mit symmetrischen Schlüsseln orientierte sich die Entwicklung des NTS-Protokolls an den Empfehlungen des RFC 7384 „*Security Requirements of Time Protocols in Packet Switched Networks*“ [9]. Folgende Ziele verfolgt NTS dabei:

- Authentifizierung des Zeitservers durch den Client
- Autorisierung des Zeitservers und optional der Clients
- Request-Response-Konsistenz (Überprüfbarkeit, dass die Servernachricht auf der Anfrage des Clients basiert)
- Integrität der Zeitsynchronisationspakete (Verwendung eines MAC)
- Anwendbarkeit in bestehenden Zeitprotokollen
- Keine Vertraulichkeit (Verschlüsselung) der Zeitinformationen (Kompatibilität)
- Möglichst geringer Latenzzuwachs (Genauigkeit des Zeitprotokolls)
- Kompatibilität (keine Veränderungen für ungesicherte Zeitprotokollverbindungen)
- Keine signifikante Leistungsverschlechterung, geringer Ressourcenbedarf
- Hybride Funktionalität (Zeitsynchronisierungsprotokoll mit und ohne NTS betreibbar
- Zustandsloser Server
- Schutz vor Verstärkungsattacken (amplification DoS)
- Skalierbarkeit (Unabhängigkeit von der Anzahl der Kommunikationsteilnehmer, keine pre-shared Keys o.ä.).

Grundsätzlicher Aufbau der Kommunikation in NTS

NTS-over-NTP nutzt das verbindungslose UDP als Transportprotokoll (Port 123), das über IPv4 oder IPv6 kommuniziert. Damit belegt NTP die oberen Ebenen des OSI-Modells, wobei NTS sog. Erweiterungsfelder von NTP verwendet, um seine Informationen einzubetten. Um eine universelle Einsetzbarkeit zu erreichen, werden die Informationen durch ASN.1 (s.u.) [10] kodiert. Im Unicast-Betrieb arbeiten die beiden Teilnehmer im Client/Server-Modus. Die Integrität der Zeitinformation wird über einen MAC sichergestellt. Der dazu benötigt symmetrische Schlüssel, das sog. Cookie, wird in der Anbahnung der Kommunikation auf sichere Weise den Teilnehmern zugestellt. Durch die Verwendung der Erweiterungsfelder wird gewährleistet, dass normale

Nutzer von NTP die Zeitpakete wie gewohnt ohne Einschränkungen verarbeiten können.

Das NTS-Client-Protokoll ist zustandsbehaftet und speichert die ausgehandelten Verbindungsparameter mit dem gewählten Server. Weiterhin unterstützt der Client die gleichzeitige Kommunikation mit verschiedenen Servern, wobei die jeweiligen Parameter unterschiedlich sein können. Die Kommunikationsparameter des Clients müssen die Mindestanforderungen des jeweiligen Servers erfüllen. Solange keine Autorisierung des Clients seitens des Servers gefordert ist, kann der Client seine Schlüsselpaare und Zertifikate selbst generieren bzw. signieren.

Der Server selbst ist – wie schon erwähnt – zustandslos, d.h. alle die spezifische Kommunikation betreffenden Informationen müssen jeweils vom Client zum Server in der entsprechenden Nachricht übertragen werden, auf die der Server dann reagiert. Die Zertifikate eines öffentlich zugänglichen Servers müssen von einer entsprechenden Zertifizierungsstelle stammen.

Die Kommunikation im NTS-Protokoll besteht aus zwei wesentlichen Phasen. Zu Anfang werden in mehreren Schritten die Kommunikation aufgebaut und dabei die speziellen Parameter ausgehandelt. Daran schließt sich die Übertragungsphase der Zeitinformationspakete an, die solange läuft, bis der Client seine Anfragen einstellt (s. Abb. 1). Ändert der Server aufgrund von Sicherheitsstrategien seine Kommunikationsparameter (z.B. den sog. Server Seed), so startet das Protokoll erneut mit der Verhandlungsphase.

Die verschiedenen Nachrichten der Verhandlungs- und der Zeitübertragungsphase zeigt Abb. 1. In der Zugriffsphase fragt der Client mit dem *ClientAccessData-O*bjekt, strukturiert als sog. *NTS-Plain*-Archetyp, beim Server an. Der Server antwortet ebenfalls mit einem *NTS-Plain*-Archetyp, der das *ServerAccessData-O*bjekt trägt. Dieses enthält den *Access Key*, den der Server als MAC über Server Seed und IP-Adresse des Clients berechnet hat. Dieser Key berechtigt den Client die folgende Assoziationsnachricht zu senden. (Der Server Seed ist eine Zufallszahl, die der Server erzeugt hat und in regelmäßigen Abständen erneuert.)

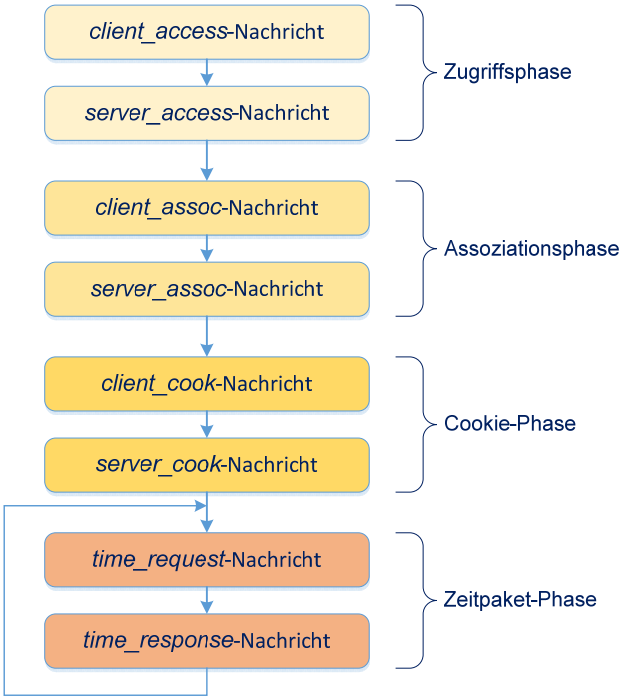

Abb. 1 Detaillierte Struktur des NTS-Nachrichtenablaufs

In der Assoziationsphase sendet der Client die *client_assoc*-Nachricht mit dem *ClientAssoc-Data*-Objekt (auch in *NTS-Plain*), das neben dem Access Key eine Zufallszahl des Clients (Nonce) und Listen mit unterstützten MAC- sowie symmetrischen und asymmetrischen Verschlüsselungsalgorithmen enthält. Der Server antwortet mit dem *ServerAssocData*-Objekt als *NTS-Signed*-Archetyp strukturiert, der u.a. in einer CMS-Struktur per Signatur gesichert die zurückgesendeten Daten enthält. Diese schließen die Nonce und die empfangenen Listen sowie die vom Server daraus ausgewählten Algorithmen ein. Zusätzlich beinhalten sie auch die Zertifikatskette des Servers, um dessen Echtheit prüfen zu können. Die Nonce dient dem Client zum Test der Request-Response-Konsistenz. Mit den zurückgesendeten Listen untersucht der

Client, ob daran Verfälschungen vorgenommen wurden, da seine *client_assoc*-Nachricht ja ungeschützt war.

Der Client überträgt in der folgenden Cookie-Phase die *client_cook*-Nachricht mit dem *ClientCookieData*-Objekt (in *NTS-Plain*), das neben einer neuen Nonce und Listen mit den ausgehandelten Algorithmen u.a. das Zertifikat bzw. die Zertifikatskette des Clients enthält. Die Listen sind nötig, da der Server ja zustandslos arbeitet. Dieser antwortet mit der *server_cook*-Nachricht, deren Struktur dem *NTS-Encrypted-and-Signed*-Archetyp entspricht. Dieser Typ erlaubt über entsprechend gestaltete CMS-Strukturen sowohl die Signierung als auch die Verschlüsselung der übertragenen Daten. Das *ServerCookieData*-Objekt überträgt die empfangene Nonce des Clients sowie das vom Server berechnete Cookie, das in der folgenden Kommunikation als symmetrischer Schlüssel fungiert. Dieses Cookie erzeugt er mittels des ausgehandelten MAC-Algorithmus aus Server Seed und dem sog. Key Input Value (KIV). Dieser ist ein Hash (mittels ausgehandeltem Algorithmus) über das Clientzertifikat. Mit Hilfe des öffentlichen Schlüssels des Clients erfolgt die Verschlüsselung des Cookies, das daher nur noch vom Client mit dessen privatem Schlüssel entschlüsselt werden kann. Da der Server zustandslos arbeitet, generiert er sich das Cookie jedes Mal neu, weshalb der Client bei jeder Zeitabfrage den Key Input Value überträgt. Mit der *client_cook*-Nachricht ist die Verhandlungsphase abgeschlossen und die Zeitübertragung kann beginnen.

Der Client startet jede Zeitanfrage mit der *time_request*-Nachricht. Sie beinhaltet das *TimeRequestSecurityData*-Nachrichtenobjekt in *NTS-Plain*-Struktur. Es enthält eine neue Nonce, KIV und MAC-Algorithmus sowie den MAC (mit Hilfe des Cookies berechnet) über das gesamte NTP-Paket, in das die *time_request*-Nachricht eingebettet ist. Mit der *time_response*-Nachricht in *NTS-Plain*-Struktur sendet der Server das *TimeResponseSecurityData*-Nachrichtenobjekt, das die empfangene Nonce und den MAC mit Hilfe des (erneut berechneten) Cookies über das ganze NTP-Paket enthält. Der Client kann nach Empfang mit seinem Cookie den MAC des Zeitinformationspakets überprüfen und die Gültigkeit der Zeitinformation der NTP-Auswertung garantieren. In den meisten Nachrichten wird implizit die NTS-Version mit übertragen und entsprechend ausgewertet. Für den gesamten Nachrichtenfluss wurden detaillierte Sequenzdiagramme erstellt.

Datenübertragung in NTP-Erweiterungsfeldern

Wie schon erwähnt, erfolgt die Übertragung der NTS-spezifischen Informationen in einem oder mehreren sog. Erweiterungsfeldern [5, 11] der regulären NTP-Nachricht. NTS nicht anwendende Empfänger ignorieren diese von NTS genutzten Felder. Die Erweiterungsfelder haben alle den gleichen Aufbau: einen Header (bestehend aus *Field Type* und *Length*), den NTS-Inhalt sowie bei Bedarf ein Padding. Der NTS-Inhalt (s. Abb. 2) ist als ASN.1-Objekt vom Typ *NTSExtensionFieldContent* aufgebaut. Neben der Objektkennung und einem Bereich für Fehlermitteilungen des Servers beinhaltet das *Content*-Feld den entsprechenden Archetyp mit dem in CMS-Strukturen verpackten Nachrichtenobjekt.

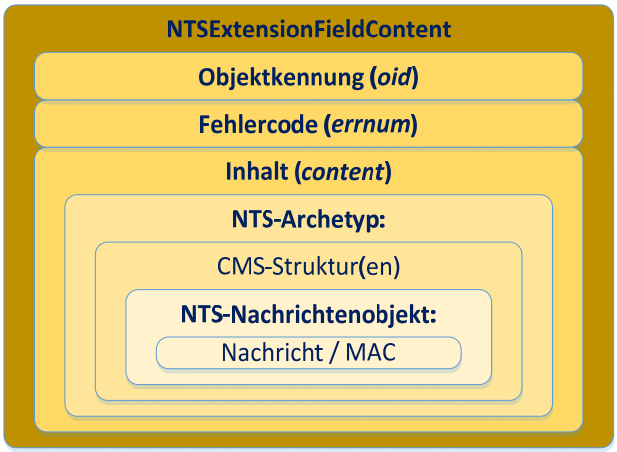

Abb. 2 Aufbau des NTS-Inhaltes eines NTP-Erweiterungsfelds

Für sämtliche oben aufgeführten Nachrichtenobjekte existieren detaillierte Beschreibungen in ASN.1-Notation sowie entsprechende Objektkennungen. U.a. wegen vieler Optionen und durch Variationen speziell bei den Zertifikaten können derzeit über 130 ASN.1-Beschreibungen teilweise mit Verschachtelungstiefen größer 20 auftreten.

Aufbau der CMS-Strukturen

Die NTS-Nachrichten verwenden drei verschiedene Archetypen (s. Abb. 3), um die jeweiligen Daten zu verpacken. Dabei kommen sog. CMS-Strukturen (Cryptographic Message Syntax) zum Einsatz, die in ASN.1 beschrieben sind und verschiedene kryptografische Aufgaben erledigen [12, 13].

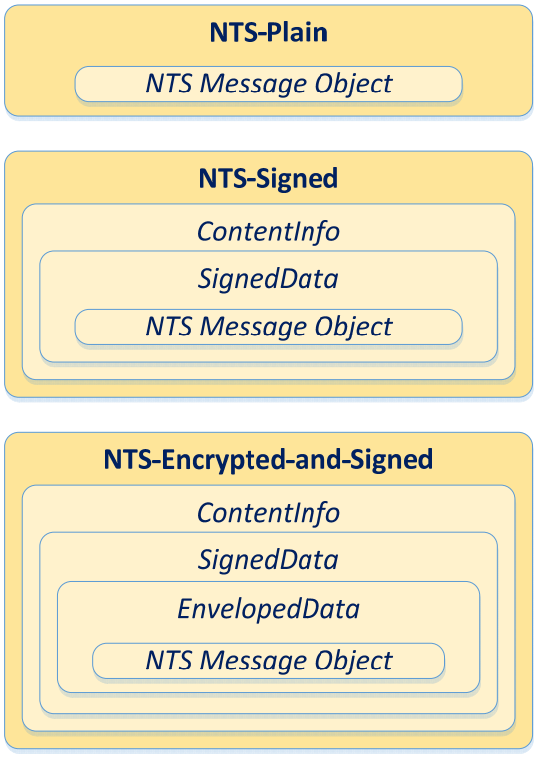

Abb. 3 Aufbau der eingesetzten CMS-Strukturen

Wie oben erwähnt, verwenden etliche Nachrichten den *NTS-Plain*-Archetyp. Er ist streng genommen keine CMS-Struktur und vergleichsweise einfach aufgebaut. Den *NTS-Signed*-Archetyp benutzt nur die *server_assoc*-Nachricht, die mit der dadurch möglichen Signierung dem Client die Echtheitsprüfung der übermittelten Daten ermöglicht. Die Struktur *ContentInfo* erlaubt bei der Verarbeitung die Identifizierung der enthaltenen CMS-Strukturen. Der Archetyp *NTS-Encrypted-and-Signed* gewährt zusätzlich eine Verschlüsselung des Inhalts in der Struktur *EnvelopedData*. Dieser Typ wird von der *server_cook*-Nachricht genutzt, um die Vertraulichkeit der Übertragung des Cookies zu gewährleisten. Den detaillierten Aufbau und die dazugehörigen Inhalte können u.a. [14] entnommen werden. Ihre Beschreibung würde den Rahmen dieses Berichts deutlich sprengen.

Benötigte Funktionalitäten

Für eine Implementierung des NTS-Protokolls ist eine Vielzahl von speziellen Funktionen nötig, die möglichst von fertigen Bibliotheken bereitgestellt werden sollten. Zur Beschreibung der Datenstrukturen verwendet NTS die *Abstract Syntax Notation One* (ASN.1) [10]. Sie ist sprach- und plattformunabhängig, wodurch Informationen auch zwischen völlig heterogenen Systemen ausgetauscht werden können. So spielen beispielsweise Bitgrößen der verwendeten Variablen und Strukturen sowie Endianess (Bytereihenfolge im Speicher) und andere Unterschiede keine Rolle. Die spezifische Syntax der teilweise sehr komplexen ASN-.1-Bescheibungen wird in eine Transfersyntax „serialisiert“ (Coder) und dann übertragen. Auf dem Zielsystem sorgt ein Decoder für die korrekte Umsetzung in Strukturen des Zielsystems. Als Codierungsverfahren stehen die *Basic Encoding Rules* (BER), die *Distinguished Encoding Rules* (DER) und die *XML Encoding Rules* (XER) zur Verfügung, die alle ihre speziellen Anwendungsgebiete haben. Eine Bibliothek zur Unterstützung der ASN.1-Verarbeitung muss also u.a. diese Codierungsverfahren bereitstellen und möglichst auch eine Umwandlung der ASN.1-Beschreibung in C- bzw. C++-Code erlauben. Nach ausgiebiger Untersu-

chung fiel die Wahl auf die Bibliothek asn1c [15], eine freie und brauchbar dokumentierte Bibliothek, wobei dennoch vermutlich eine Reihe von Wrapper-Funktionen zu erstellen sein wird.

NTS benötigt weiterhin die folgenden Funktionalitäten:

- Bereitstellung und Verarbeitung von CMS-Strukturen
- 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 Signaturen
- Verarbeitung und Generierung von X.509-Zertifikaten.

Sämtliche aufgeführten Funktionen stellt neben anderen die weit verbreitete, freie Library OpenSSL zur Verfügung [16]. OpenSSL ist für viele Betriebssysteme erhältlich, weshalb u.a. die Wahl auf sie fiel. Die Eignung von OpenSSL für die ASN.1-Verarbeitung konnte bislang nicht ausreichend geprüft werden. Ob OpenSSL aufgrund seiner Größe auch auf kleineren Embedded Systemen Einsatz finden kann, muss ebenfalls noch verifiziert werden. Allerdings existieren auch Erfahrungen mit kleineren Bibliotheken wie wolfSSL, die zumindest einen Teil der benötigten Funktionalitäten auch für schwächere Systeme zur Verfügung stellt.

Stand der Entwicklung

Insgesamt war der Aufwand für die Analyse des NTS-Protokolls und der zugehörigen IETF Drafts für eine Zeitübertragung über NTP im Unicast-Modus sowie kleinere bisherige Testimplementierungen erheblich. Für die anstehenden Implementierungen entstanden detaillierte Sequenz- und Aktivitätsdiagramme sowie Übersichten zu sämtlichen ASN.1-Strukturen, die in NTS zur Anwendung kommen. Letztere umfassen die kompletten Beschreibungen der NTS-Nachrichtenobjekte, der verwendeten CMS-Strukturen und der X.509-Zertifikate. Aufwändig waren auch die Nachverfolgung und das Einpflegen der vielen Änderungen in den Drafts. (Beispielsweise änderte sich die Main-Spec für NTS, der *draft-ietf-ntp-network-time-security*, von Version 10 bis heute auf Version 15.) Ebenfalls fanden auch viele Diskussionen mit den an der Spezifikationsarbeit Beteiligten der PTB statt. Da die Spezifikation langsam konvergiert und demnächst in einen RFC münden soll, besteht die Hoffnung, dass sich die Änderungen und damit der Anpassaufwand in Zukunft verringern werden. Mit Auswahl der Bibliotheken konnte auch mit der Planung der Implementation begonnen werden. Gleichzeitig begannen auch die Arbeiten an einer spezifischen NTP-Implementierung mit einer Schnittstelle zu NTS, damit der tatsächliche Betrieb bei Vorliegen einer NTS-Implementierung in vollem Umfang getestet werden kann. Aufgrund des unvorhergesehen großen Aufwands waren im Rahmen des ersten Teils des Projekts keine Implementierung und entsprechend kein Test einer realen Client/Server-Kommunikation möglich. Das ist für den zweiten Teil des Projektes geplant. Der vorgesehene Test mit einer Realisierung der Network Time Foundation (NTF) fand ebenfalls nicht statt, da die NTF mindestens ebenso weit hinter dem ursprünglichen Zeitplan liegt.

Literatur

[1] BSI: *Technische Richtlinie BSI TR-03109-1: Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems*. Bundesamt für Sicherheit in der Informationstechnik, Bonn, 2013.

[2] PTB: *PTB-A 50.8 – Smart Meter-Gateway*. Physikalisch-Technische Bundesanstalt, Braunschweig, 2013.

[3] Röttger, S.: *Analyse des NTP-Autokey-Verfahrens.* Projektarbeit. Braunschweig: Technische Universität Braunschweig, Physikalisch-Technische Bundesanstalt, 2011.

[4] Sibold, D., Röttger, S., Teichel, K.,: *Network Time Security*. draft-ietf-ntp-network-time-security-15, IETF, September 2016.

[5] Mills, D., Martin, J., Ed., Burbank, J., Kasch, W.: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905, DOI 10.17487/RFC5905, 2010.

[6] IEEE: *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std. 1588–2008. New York, 2008.

[7] Sibold, D., Röttger, S., Teichel, K.,: *Using the Network Time Security Specification to Secure the Network Time Protocol.* draft-ietf-ntp-using-nts-for-ntp-06, IETF, September 2016.

[8] Perrig, A., Song, D., Canetti, R., Tygar, J., Briscoe, B.: *Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction*. RFC 4082, DOI 10.17487/RFC4082, 2005.

[9] Mizrahi, T.: *Security Requirements of Time Protocols in Packet Switched Networks.* RFC 7384, DOI 10.17487/RFC7384, Okt. 2014.

[10] International Organization for Standardization: *Abstract Syntax Notation One (ASN.1) - Specification of basic notation*. Genf: ISO/IEC 8824-1:2015.

[11] Mizrahi, T., Mayer, D.: *Network Time Protocol Version 4 (NTPv4) Extension Fields*. RFC 7822, März 2016.

[12] Housley, R: *Cryptographic Message Syntax (CMS)*. RFC 5652, September 2009.

[13] Kaliski, B.: *PKCS #7: Cryptographic Message Syntax - Version 1.5*. RFC 2315, März 1998.

[14] Sibold, D., Teichel, K., Röttger, S., Housley, R.: *Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)*. draft-ietf-ntp-cms-for-nts-message-06, IETF, Februar 2016.

[15] Walkin, L.: *asn1c Documentation*. URL: http://lionet.info/asn1c/documentation.html

[16] OpenSSL Software Foundation: *Welcome to OpenSSL!* URL: https://www.openssl.org

Kontaktdaten

Ostfalia Hochschule für angewandte Wissenschaften
Fakultät Elektrotechnik
Prof. Dr.-Ing. Rainer Bermbach
Salzdahlumer Straße 46/48
38302 Wolfenbüttel
Telefon: +49 (0)5331 939 42620
E-Mail: r.bermbach@ostfalia.de
Internet: www.ostfalia.de/pws/bermbach