Bundesamt
für Sicherheit in der
Informationstechnik

Deutschland
Digital•Sicher•BSI•

Detailspezifikationen zur TR-03109-5

Anforderungen an die Interoperabilität eines CLS-Kommunikationsadapters

TR-03109-5 Version 1.0
Datum:2023-11-24, Commit:aeb06737

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
E-Mail: smartmeter@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhalt

1 Einleitung 1

2 HAN-Zertifikatsprofil 2
2.1 Einleitung 2
2.2 Laufzeit 3
2.3 Zertifikatsstruktur 3
2.4 HAN-TLS-Zertifikate des SMGW 6
2.5 TLS-Zertifikate der HAN-Teilnehmer 6

3 RESTful Webservice Protokoll 8
3.1 Einleitung 8
3.2 Anforderungen an HTTP-Clients 8
3.3 Content-Types 9
3.4 Status Codes 9
3.5 Header-Felder 10
3.6 URI-Pfade 11
3.7 Kanonisierung für JSON Signaturen 11

4 SOCKS-TLS Protokoll 12
4.1 Einleitung 12
4.2 Technische Anwendungsfälle des SOCKS-TLS Clients 12
4.3 Technische Anwendungsfälle für SOCKS-TLS Client und Server 13

5 mDNS - Device/Service Discovery Protokoll 15
5.1 Einleitung 15
5.2 Technische Anwendungsfälle für das mDNS Querying 15

6 TLS - Transport Layer Security Protokoll 17
6.1 Einleitung 17
6.2 Datenstrukturen 17
6.3 Technische Anwendungsfälle 18

7 NTP - Network Time Protocol 29
7.1 Einleitung 29
7.2 Datenstrukturen 29
7.3 Technische Anwendungsfälle des NTP-Clients 30

Literaturverzeichnis 34

Glossar 36

A Abkürzungsverzeichnis 37

1 Einleitung

Die folgenden Kapitel enthalten die Detailspezifikationen zur Interoperabilität für die Technische Richtlinie BSI TR-03109-5.

Die Detailspezifikationen konkretisieren die aus Kommunikationsszenarien der TR-03109-5 referenzierten Standards.

Zielgruppe dieses Dokumentes sind daher Hersteller von CLS-Komponenten und Prüfstellen für den CLS-Kommunikationsadapter.

Die in diesem Dokument enthaltenen Spezifikationen referenzieren in der Regel Universalspezifikationen oder Teile davon und können weitere anwendungsspezifische Anforderungen enthalten. Universalspezifikationen sind beispielsweise Normen, Standards, Richtlinien des BSI, der IETF und weiterer nationaler, europäischer oder internationaler Normungsorganisationen.

Die in diesem Dokument enthaltenen, einzelnen Kapitel widmen sich unterschiedlichsten Bereichen. Sie sind daher nicht zur zusammenhängenden Lektüre gedacht.

2 HAN-Zertifikatsprofil

2.1 Einleitung

Die HAN-Zertifikate, die die Implementierung für sich erstellt, verwendet und akzeptiert, verwenden die Syntax und Semantik von X.509-Zertifikaten in der Version 3 nach [RFC5280] Kapitel 4 und **SOLLEN** je nach Zertifikatstyp in ▸Tabelle 2.1 konform zu einem der in diesem Kapitel beschriebenen Zertifikatsprofiltypen sein: [REQ.ZertifikateHAN.Allgemein.10]

- CT_Selfsigned (Typ A): Selbstsigniert, nicht aus einer PKI
- CT_CA_Signed (Typ B): PKI-Zertifikat durch eine CA des GWA oder GWH ausgestellt
- CT_SMPKI_Signed (Typ C): C_{TLS}(SMGW)-Zertifikat der SM-PKI gemäß [TR-03109-4]

ICS.ZertifikateHAN.Allgemein.10

Der Hersteller **MUSS** im ICS deklarieren, ob die Implementierung die Anforderungen an die Verarbeitung von CT_Selfsigned HAN-Zertifikatsprofile für die GW_HAN_TLS_CRT-Zertifikate umsetzt.

ICS.ZertifikateHAN.Allgemein.11

Der Hersteller **MUSS** im ICS beschreiben, welche Abweichungen in der Implementierung bei Verarbeitung von selbstsignierten GW_HAN_TLS_CRT-Zertifikaten zu den HAN-Zertifikatsprofilen vom Typ CT_Selfsigned bestehen.

ICS.ZertifikateHAN.Allgemein.12

Der Hersteller **MUSS** im ICS deklarieren, ob die Implementierung die Anforderungen an die CT_Selfsigned HAN-Zertifikatsprofile für die CLS_HAN_TLS_CRT-Zertifikate umsetzt.

ICS.ZertifikateHAN.Allgemein.13

Der Hersteller **MUSS** im ICS beschreiben, welche Abweichungen in der Implementierung von selbstsignierten CLS_HAN_TLS_CRT-Zertifikaten zu den HAN-Zertifikatsprofilen vom Typ CT_Selfsigned bestehen.

Zertifikatstyp	Zertifikats-Profiltyp	Inhaber (Subject)	Aussteller (Issuer)
GW_HAN_TLS_CRT	CT_Selfsigned	SMGW	SMGW
GW_HAN_TLS_CRT (entspricht GW_WAN_TLS_CRT)	CT_SMPKI_Signed	SMGW	SUBCA SM-PKI
CLS_HAN_TLS_CRT	CT_Selfsigned	CLS	CLS[1]

Tabelle 2.1 Zertifikatsprofiltypen der HAN-Zertifikate

Die folgenden Abschnitte enthalten Tabellen in denen Anforderungen an das Vorhandensein von Attributen und Datenfeldern verschiedener Zertifikatsprofile beschrieben sind. Darin werden die folgenden Abkürzungen verwendet:

Zeichen	Langform	Bedeutung
m	Mandatory	Das Element muss vorhanden sein.
x	Not existent	Das Element darf nicht vorhanden sein.
o	Optional	Das Element kann vorhanden sein

[1] Erzeugt das SMGW das Schlüsselpaar und selbstsignierte Zertifikat im Auftrag des CLS, so enthalten Issuer und Subject die Funktionsrolle "cls"

Zeichen	Langform	Bedeutung
c	Conditional	Das Element ist abhängig von einem anderen Element vorhanden
r	Recommended	Das Element soll vorhanden sein.

Tabelle 2.2 Beschreibung der Abkürzung zum Vorhandensein von Zertifikats-Elementen

2.2 Laufzeit

Eine Prüfung der zeitlichen Gültigkeit von CT_Selfsigned Zertifikaten durch die Implementierung wird nicht vorgegeben, da es Aufgabe des Administrators ist, dass die Implementierung nur gültigen, selbst-signierten Zertifikaten vertraut und dies im Falle des SMGW über die HAN- und Proxy-Kommunikationsprofile parametriert.[2]HAN-Zertifikate vom Typ CT_SMPKI_Signed besitzen gemäß [TR-03109-4] eine Laufzeit von höchstens 2 Jahren.

Dennoch sollen die Verwender des GW_HAN_TLS_CRT im HAN Kenntnis über den geplanten Verwendungszeitraum der SMGW-Zertifikate erhalten können. Die Laufzeit richtet sich nach der erwarteten Sicherheit der verwendeten kryptografischen Verfahren unter Berücksichtigung einer möglichen kryptografischen Migration und entspricht der Laufzeit von Zertifikaten am LMN gemäß [TR-03109-3].

2.3 Zertifikatsstruktur

Zertifikatsfeld	Referenz in RFC5280	Element vorhanden	Wert
TBSCertificate	4.1.1.1	m	Siehe ▸Tabelle 2.4
SignatureAlgorithm	4.1.1.2	m	Siehe ▸Abschnitt 2.3.1
SignatureValue	4.1.1.3	m	Abhängig vom gewählten Signatur-Algorithmus nach [TR-03109-3]: • CT_Selfsigned: Mit dem privaten Schlüssel zum "subjectPublicKeyInfo" selbstsigniert • CT_CA_Signed, CT_SMPKI_Signed: Mit dem privaten Schlüssel des Ausstellers (Zertifizierungsstelle) signiert.

Tabelle 2.3 Struktur des Elementes "Certificate"

Die folgende Tabelle gibt die Struktur des Feldes "TBSCertificate" verbindlich vor.

Zertifikatsfeld	Referenz in RFC5280	Element vorhanden	Wert
Version	4.1.2.1	m	'v3'
SerialNumber	4.1.2.2	m	Zufällig gewählte, positive Ganzzahl, bestimmt vom Aussteller (8-20 Octets).
Signature	4.1.2.3	m	Gleicher Wert wie im Feld "SignatureAlgorithm" (s. ▸Abschnitt 2.3.1).
Issuer	4.1.2.4	m	Eindeutiger Name (Distinguished Name, DN) des Zertifikatsinhabers siehe ▸Abschnitt 2.3.3: • Für CT_Selfsigned HAN-Zertifikate identisch zu "Subject". • Für CT_CA_Signed, CT_SMPKI_Signed: *SubjectCN* der ausstellenden CA
Validity	4.1.2.5	m	Zeitpunkte für Beginn und Ende der Gültigkeit des Zertifikates (s. ▸Abschnitt 2.3.4).

[2] Geräte im HAN haben nach der Inbetriebnahme zunächst oft keine zuverlässige Kenntnis über die Zeit. Eine Prüfung der zeitlichen Gültigkeit ist hier nur eingeschränkt möglich.

Zertifikatsfeld	Referenz in RFC5280	Element vorhanden	Wert
Subject	4.1.2.6	m	Eindeutiger Name (Distinguished Name, DN) des Zertifikatsinhabers siehe ▸Abschnitt 2.3.3.
SubjectPublicKeyInfo	4.1.2.7	m	Siehe ▸Abschnitt 2.3.2.
IssuerUniqueId	4.1.2.8	x	Entfällt
SubjectUniqueId	4.1.2.8	x	Entfällt
Extensions	4.1.2.9	m	Siehe ▸Abschnitt 2.3.5.

Tabelle 2.4 Struktur des Elementes "TBSCertificate"

2.3.1 SignatureAlgorithm

Durch die Datenstruktur SignatureAlgorithm wird nach [RFC5280] der Signaturalgorithmus des Zertifikats angegeben. Dieser besteht aus der folgenden Datenstruktur:

```
AlgorithmIdentifier ::= SEQUENCE {algorithm OBJECT IDENTIFIER, parameters ANY DEFINED BY algorithm OPTIONAL}
```

Der Wert von "algorithm" muss nach den Vorgaben zur TLS-Kommunikation im HAN nach [TR-03109-3] gewählt werden. Das Datenfeld "parameters" muss leer bleiben.

2.3.2 SubjectPublicKeyInfo

Das Feld "SubjectPublicKeyInfo" besitzt folgende Struktur (s. [RFC5480]):

```
SubjectPublicKeyInfo ::= SEQUENCE {algorithm AlgorithmIdentifier, subjectPublicKey BIT STRING}
```

Die OID im Datenfeld "algorithm" enthält den Wert 1.2.840.10045.2.1 (id-ecPublicKey). Im Feld "ECParameters" ist gemäß [RFC5480] die Variante namedCurve zu verwenden. Für die aktuell zu verwendenden Werte siehe [TR-03109-3] TLS-Kommunikation im HAN.

Der PublicKey wird mit unkomprimierten Punktkoordinaten angegeben.

2.3.3 Issuer und Subject

- Das Namensschema für den "commonName" der HAN-Teilnehmer lehnt sich an [SM-PKI-CP] Anhang A an.
- Für CT_Selfsigned HAN-Zertifikate müssen "subject" und "issuer" gemäß [RFC5280] identisch sein.
- Einzelne Attribute von Issuer und Subject dürfen nicht länger als 64 Zeichen sein.
- Vorgaben an das Namensschema von Issuer und Subject sind in ▸Abschnitt 2.4 und ▸Abschnitt 2.5 beschrieben.

2.3.4 Validity

- Das Feld "notBefore" enthält den Erzeugungszeitpunkt des Schlüsselpaares und des Zertifikates.
- Das Feld "notAfter" enthält den Zeitpunkt, nach dem das Zertifikat und das dazugehörige Schlüsselpaar nicht mehr verwendet werden darf. Der Zeitpunkt muss nach dem Erzeugungszeitpunkt liegen.

2.3.5 Extensions

Die nun folgende Tabelle enthält eine Übersicht der Extensions und den Anforderungen an deren Vorhandensein.

Bezeichnung / Typ	CT_Selfsigned	CT_CA_Signed	CT_SMPKI_Signed: $C_{T\text{-}LS}$(SMGW)
AuthorityKeyIdentifier	o	m	m

Bezeichnung / Typ	CT_Selfsigned	CT_CA_Signed	CT_SMPKI_Signed: $C_{T\text{-}LS}$(SMGW)
SubjectKeyIdentifier	o	m	m
KeyUsage	r	r	m
PrivateKeyUsagePeriod	x	x	x
CertificatePolicies	x	o	o
SubjectAltNames	c (Bisher m)	o	x
BasicConstraints	r cA=FALSE	m cA=FALSE	m
ExtendedKeyUsage	r	r	r
CRLDistributionPoints	x	o	o
Weitere Extensions	o	x	o

Tabelle 2.5 Extensions

2.3.5.1 KeyUsage

- Extension-ID (OID): 2.5.29.15
- Kritisch: Nein für Typ CT_Selfsigned, Ja für Typ CT_CA_Signed, CT_SMPKI_Signed
- Beschreibung: Die Extension KeyUsage (spezifiziert in [RFC5280], 4.2.1.1) definiert, für welche Zwecke der zertifizierte öffentliche Schlüssel verwendet werden darf.
- Gesetzte Bits: digitalSignature(0)[3].

2.3.5.2 BasicConstraints

- Extension-ID (OID): 2.5.29.19
- Kritisch: Ja für CT_CA_Signed und CT_SMPKI_Signed Zertifikate, Nein für Typ CT_Selfsigned Zertifikate
- Beschreibung: Diese Extension (spezifiziert in [RFC5280], 4.2.1.9) gibt an, ob es sich bei dem gegebenen Zertifikat um eine CA handelt und wie viele CAs ihr folgen können.

2.3.5.3 SubjectAltName

- Extension-ID (OID): 2.5.29.27
- Kritisch: Nein
- Beschreibung: Diese Extension (spezifiziert in [RFC5280], 4.2.1.6) ermöglicht die Bindung von Identitäten an das Subject des Zertifikates. Diese Angabe ist alternativ oder zusätzlich zur Identifikation im Subject möglich.
- "dNSName": Es wird empfohlen, die eindeutige Kennung von SMGW oder HAN-Teilnehmer basierend auf [DIN43849] mit angehängtem ".sm" oder eine auf der Komponente des HAN-Teilnehmers lesbare, praktisch eindeutige, nicht-wechselnde, herstellerübergreifende Netzwerkschnittstellenadresse (MAC/EUI) mit angehängtem ".eui" als Domain Name Label zu verwenden.

2.3.5.4 ExtendedKeyUsage

- Extension-ID (OID): 2.5.29.37
- Kritisch: Nein
- Beschreibung: Die Extension ExtendedKeyUsage (spezifiziert in [RFC5280], 4.2.1.12) gibt an, ob das Zertifikat als TLS-Client und/oder als TLS-Server-Zertifikat genutzt werden kann.
- Die HAN-Teilnehmer Zertifikate sollen sowohl für TLS-Web-ServerAuthentifikation (1.3.6.1.5.5.7.3.1) als auch für TLS-Web-ClientAuthentifikation (1.3.6.1.5.5.7.3.2) verwendet werden.

[3] Für CLS-TLS-Zertifikate sind weitere Key Usages zulässig

- "TLS-Web-ClientAuthentication": Sofern GW_HAN_TLS_CRT Zertifikate des SMGW aus der SM-PKI stammen und die ExtendedKeyUsage "TLS-Web-ClientAuthentication" enthalten, dürfen diese vom Kommunikationsadapter auch zur Authentifizierung des SMGW als TLS-Server verwendet werden, sofern die im TLS Handshake für die TLS-Session abgeleiteten PSK- und Ephemeralen-Schlüssel für Client und Server Rolle des Kommunikationsadapters nur für ihre jeweilige Verbindung verwendet werden.

2.4 HAN-TLS-Zertifikate des SMGW

2.4.1 Einleitung

Zusätzlich zu den Anforderungen an CT_Selfsigned HAN-Zertifikate nach ▸Abschnitt 2.3 gelten für *GW_HAN_TLS_CRT*-Zertifikate die Anforderungen in ▸Abschnitt 2.4.

Die Anforderungen an die Zertifikatsstruktur der CT_SMPKI_Signed HAN-Zertifikate des SMGW sind in [TR-03109-4] C_{TLS}(SMGW) beschrieben.

2.4.2 Namensschema der SMGW-HAN-Zertifikate CT_Selfsigned

Attribut Typ	Kürzel	Vorgabe	Wert	Erläuterung
common name	CN	r	"<id>.smgw"	<id> Herstellerübergreifende Identifikationsnummer nach [DIN43849] mit der Sparte "E".
organisation	O	o	"<O>"	Falls verwendet darf das Feld "SM-*PKI" nicht enthalten.
organisational unit	OU	o	"<GWA-ID>"	Name des für das SMGW zuständigen GWA (aus "common name" des *GWA_WAN_SIG_CRT*).
country	C	r	"<LC>"	Zwei Zeichen Ländercode gemäß [ISO 3116 ALPHA-2]
serial number	SERIAL NUMBER	r	"<SN>"	Sequenznummer der Zertifikats im Bereich von 1 bis 2^31-1 und startet bei 1. Diese wird bei jedem neuen Zertifikat um den Wert 1 hochgezählt.

Tabelle 2.6 Namensschema der SMGW-HAN-Zertifikate

2.4.3 Namensschema der SMGW-HAN-Zertifikate CT_SMPKI_Signed

Die Anforderungen an das Namensschema sind in [SM-PKI-CP] Anhang A beschrieben.

2.5 TLS-Zertifikate der HAN-Teilnehmer

2.5.1 Einleitung

Zusätzlich zu den Anforderungen an Typ CT_Selfsigned HAN-Zertifikate nach ▸Abschnitt 2.3 gelten für *CLS_HAN_TLS_CRT*-Zertifikate die Anforderungen in ▸Abschnitt 2.5.

2.5.2 Namensschema der HAN-Teilnehmer-Zertifikate

HAN-Teilnehmerzertifikate (außer *GW_HAN_TLS_CRT*) enthalten folgende Distinguished Name Attribute:

Attribut Typ	Kürzel	Vorgabe	Wert	Erläuterung
common name	CN	r	"<id>.<func>"	• Sofern <func> gleich "cls" ist, enthält <id> die auf der HAN-Komponente ablesbare, kanonisierte Geräteidentifikation gemäß [DIN43849]. • Die <id> und das <idschema> für Service-Techniker und Letztverbraucher-Zertifikate wird gemäß organisatorischer Anforderungen des Ausstellers vergeben und vom GWA im SMGW konfiguriert. • Aus Gründen der Lesbarkeit des CommonName wird eine Schreibweise einheitlich mit Groß- oder Kleinbuchstaben empfohlen. Eine semantische Interpretation des CN findet nicht statt.
country	C	o	"<LC>"	Zwei Zeichen Ländercode gemäß [ISO 3116 ALPHA-2]
serial number	SERIAL NUMBER	o	"<SN>"	Falls vorhanden, enthält das Feld die Sequenznummer des Zertifikats im Bereich von 1 bis 2^31-1. und startet bei 1. Diese wird bei jedem neuen Zertifikat um den Wert 1 hochgezählt.

Tabelle 2.7 Namensschema der HAN-Teilnehmer-Zertifikate

Der Distinguished Name darf weitere Attribute enthalten (nicht empfohlen). Eine Prüfung durch das SMGW erfolgt aktuell nicht.

3 RESTful Webservice Protokoll

3.1 Einleitung

Das Hypertext Transfer Protokoll (HTTP) ist ein OSI Layer 7 Protokoll, das zwischen Transaktionen zustandslos ist. Es ermöglicht die Zusammenarbeit von Informationssystemen über ausgetauschte Textnachrichten (HTTP-Request und HTTP-Response genannt). Das Protokoll definiert dabei das Interface, um mit den Ressourcen zu interagieren.

Die RFCs [RFC9110] und [RFC9112] bilden den Rahmen der Anforderungen an die Implementierung von HTTP/1.1. Sie beschreiben aber auch viele Protokoll-Optionen. Für den interoperablen Betrieb und da die Implementierung über begrenzte Ressourcen verfügt, sind die Optionen allerdings nicht vollumfänglich zu implementieren. Insbesondere Aspekte wie Datenvolumen und Migration zwischen Versionen begründen eine Einschränkung auf die notwendigen Protokoll-Optionen für einen leistungsfähigen und interoperablen Betrieb.

Die umgesetzten Optionen der HTTP-Implementierung sollen den Vorgaben der [RFC9110] und [RFC9112] entsprechen

3.2 Anforderungen an HTTP-Clients

Dieses Kapitel beschreibt die Anforderungen an das Verhalten der Implementierung eines HTTP-Clients.

REQ.Webservice.HttpClient.10

Die Implementierung **MUSS** das Format, d.h. die Syntax, der HTTP-Nachrichten gemäß der Spezifikation in [RFC9112] verarbeiten können.

REQ.Webservice.HttpClient.50

Die Implementierung **MUSS** HTTP-Statuscodes gemäß [RFC9110] Kapitel 15 mit der Semantik gemäß ▸Tabelle 3.2 verarbeiten können.

REQ.Webservice.HttpClient.60

Die Implementierung **MUSS** die Methode POST für die adressierte Entitäten-Ressource (URI ohne trailing "/") mit der Semantik "Post" gemäß [RFC9110] Kapitel 9 verwenden, um den Inhalt des Requests als Nachricht an den Empfänger zu übermitteln.

REQ.Webservice.HttpClient.70

Die Implementierung **MUSS** einen zum Content passenden Content-Type Header gemäß ▸Tabelle 3.1 im Request senden, sofern ein Content-Body vorhanden ist.

REQ.Webservice.HttpClient.80

Die Implementierung **MUSS** sicherstellen, dass die HTTP-Request-Response-Transaktion in der Implementierung beendet wird, wenn die zugeordnete TLS-Verbindung getrennt wird.

REQ.Webservice.HttpClient.90

Die Implementierung **SOLL** zur Minimierung der Datenmenge nur die für die Interoperabilität notwendigen Request-Header senden (siehe ▸Tabelle 3.3).

REQ.Webservice.HttpClient.100

Die Implementierung **MUSS** Response-Header-Fields mit einer Länge bis zu 250 Zeichen (inkl. CR, LF) verarbeiten können.

REQ.Webservice.HttpClient.110

Die Implementierung **MUSS** Response-Header mit einer Länge bis zu 1000 Zeichen (inkl. CR, LF) verarbeiten können.

REQ.Webservice.HttpClient.120

Die Implementierung **MUSS** die Transfersyntax JSON gemäß [RFC8259] implementieren.

3.3 Content-Types

Tabelle 3.1 listet die in dieser Detailspezifikation referenzierten Content-Types auf. Kommunikationsszenarien beschreiben, welche Content-Types für die Mindestanforderungen an die Interoperabilität von der Implementierung unterstützt werden. Die Implementierung kann weitere Content-Types verwenden.

Content-Type	Content-Body Schachtelung	Referenzen
application/json	json-Datenstruktur	Transfersyntax gemäß [RFC8259]

Tabelle 3.1 Bedeutung der Content-Types

3.4 Status Codes

HTTP-Statuscodes werden vom HTTP-Server in der Response an den HTTP-Client übermittelt, um das Ergebnis der Verarbeitung des Requests im HTTP-Server zusammenzufassen. Tabelle 3.2 konkretisiert die Semantik der Statuscodes aus [RFC9110]. Client-Error Statuscodes (4xx), die nicht in [RFC9110] definiert sind und von der Implementierung nicht unterstützt werden, können auf "Bad request (400)" abgebildet werden.

Status	Beschreibung
Anfrage erfolgreich ausgeführt	
Ok (200)	Die GET, POST Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
Created (201)	Die POST-Anfrage wurde erfolgreich bearbeitet. Die angeforderte Ressource wurde vor dem Senden der Antwort erstellt.
Accepted (202)	Die POST-Anfrage wurde akzeptiert, wird aber zu einem späteren Zeitpunkt (asynchron) ausgeführt. Das Gelingen der Anfrage kann nicht garantiert werden.
No Content (204)	Die DELETE, PUT, POST Anfrage wurde erfolgreich durchgeführt, die Antwort enthält keine Ressource-Inhaltsdaten.
Partial Content (206)	GET File: Der angeforderte Teil wurde erfolgreich übertragen (wird im Zusammenhang mit einem "Content-Range"-Header-Field oder dem Content-Type multipart/byteranges verwendet). Kann einen HTTP-Client über Teil-Downloads informieren (wird zum Beispiel von Wget genutzt, um den Downloadfortschritt zu überwachen oder einen Download in mehrere Streams aufzuteilen).
Client-seitig fehlerhafte Anfrage führt zum Abbruch	
Bad Request (400)	Die Anfrage-Nachricht war fehlerhaft aufgebaut. Der HTTP-Client soll die Anfrage nicht unverändert noch einmal stellen.
Unauthorized (401)	Die Anfrage kann nicht ohne gültige Authentifizierung durchgeführt werden. Wie die Authentifizierung durchgeführt werden soll, wird im "WWW-Authenticate“-Header-Field der Antwort übermittelt.
Forbidden (403)	Die Anfrage wurde mangels Berechtigung des HTTP-Clients nicht durchgeführt, bspw. weil der authentifizierte Benutzer nicht berechtigt ist.
Not Found (404)	Die angeforderte Ressource wurde nicht gefunden.
Method Not Allowed (405)	Die Anfrage darf nur mit anderen HTTP-Methoden (zum Beispiel GET statt POST) gestellt werden.

Status	Beschreibung
Not Acceptable (406)	Die angeforderte Ressource steht nicht in der gewünschten Form (gemäß Accept-Header) zur Verfügung.
Request Timed Out (408)	Innerhalb der vom HTTP-Server erlaubten Zeitspanne wurde keine vollständige Anfrage des HTTP-Clients empfangen.
Conflict (409)	Die Anfrage wurde unter falschen Annahmen gestellt. Eine existierende Ressource kann nicht erneut angelegt werden.
Length Required (411)	Die Anfrage kann ohne ein "Content-Length"-Header-Field nicht bearbeitet werden.
Content Too Large (413)	Die Content-Body der gestellten Anfrage war zu groß, um vom HTTP-Server bearbeitet werden zu können.
Request URL Too Large (414)	Die URL der Anfrage war zu lang.
Unsupported Media Type (415)	Der Inhalt der Anfrage wurde mit ungültigem oder nicht unterstützen Content-Type übermittelt.
Requested Range Not Satisfiable (416)	Der angeforderte Teil einer Ressource war ungültig oder steht auf dem HTTP-Server nicht zur Verfügung.
Unprocessable Entity (422)	Syntaktischer Fehler in den Inhaltsdaten der Anfrage. CMS- oder Transfersyntax bzw. XML/json Struktur des Content-Body kann nicht ausgewertet werden (RFC4918).
Request Header Fields Too Large(431)	Die Maximallänge eines HTTP-Header-Fields oder des Gesamtheaders wurde überschritten (RFC6585).
Server-seitiger Fehler führt zum Abbruch	
Internal Server Error (500)	Allgemeiner Statuscode für Fehler des Servers auf FA-Ebene, der im Content-Body konkretisiert werden kann.
Bad Gateway (502)	Der HTTP-Server konnte seine Funktion als Gateway oder Proxy nicht erfüllen, weil er seinerseits keine oder eine ungültige Antwort erhalten hat.
Service Unavailable (503)	Der Anwendungsdienst steht temporär nicht zur Verfügung.
HTTP Version Not Supported (505)	Die im Request verwendete HTTP-Version wird vom HTTP-Server nicht unterstützt oder abgelehnt.

Tabelle 3.2 HTTP-Statuscodes

3.5 Header-Felder

HTTP-Header werden in Request und Response des HTTP-Servers und des HTTP-Clients zur Vereinbarung der Verarbeitung der Inhaltsdaten (Content-Body) übermittelt. ▸Tabelle 3.3 konkretisiert die in [RFC9110] beschriebenen Header für die Verwendung mit den Kommunikationsszenarien.

Name	Verwendung	Kontext
Allgemeine Header-Felder		
Content-Length	Gemäß [RFC9110].	• GET-Response • POST-Request • POST-Response • PUT-Request
Content-Type	Gemäß ▸Tabelle 3.1.	• GET-Response • POST-Request • POST-Response • PUT-Request
Accept	Gemäß ▸Tabelle 3.1.	• GET-Request • POST-Request

Name	Verwendung	Kontext
Accept-Encoding	Gemäß [RFC9110]	• GET-Request • POST-Request
Content-Encoding	Gemäß [RFC9110]	• PUT-Request • GET-Response • POST-Request • POST-Response

Tabelle 3.3 Header

3.6 URI-Pfade

URI-Pfade werden gemäß [RFC3986] in der Form "path-absolute" gebildet. ▸Tabelle 3.4 listet die Umsetzung des FA-Bezeichners auf die URI-Pfade.

Dabei ist {role} ein Platzhalter für die Rolle des Zugriffsberechtigten (lower-case) und [version} die Version des APIs (z.B. "v1").

{ResourceType} identifiziert den Typ der Datenstruktur der Ressource (Beispielsweise Konfigurations- oder Statusprofile, die in den Fachanwendungsfälle verwendet werden).

{Action} identifiziert die Funktion des auslösenden FA.

{ResourceId} identifiziert eine individuelle Instanz (Ressource) eines {ResourceType}s. Für Konfigurationsprofile besteht eine eindeutige Beziehung zum im SMGW eindeutigen Bezeichner des Profiles.

{ResourceContent} enthält den Inhalt einer identifizierten Ressource (eines {ResourceType}s).

[{ResultContent}] Optional den Response-Status ergänzende Informationen zum Ergebnis

FA-Bezeichner	Aufrufparameter	URI	Rückgabeparameter
FA.Do{Action}	(Abhängig vom FA)	POST /smgw/{role}/{version}/fas/{Action}	[{ResultContent}] FA-spezifisch

Tabelle 3.4 URI-Pfade

3.7 Kanonisierung für JSON Signaturen

JSON ermöglicht zur Verbesserung der Lesbarkeit das Einfügen von CR/LF und sog. Whitespaces in der Transfersyntax. Ohne Veränderung des Inhaltes oder der Semantik würden so verschiedene Nachrichten erzeugt. Zur Verbesserung der Interoperabilität werden die möglichen Freiheitsgrade durch eine Kanonisierungsregel eingeschränkt. Die kanonisierten Daten sind nicht an den Schnittstellen sichtbar, da sie nur als Grundlage für die Hash-Berechnung dienen, der der Signaturberechnung zugrundeliegt.

Zur Erzeugung eines Kanonisierten Strings werden alle CR (0x0D), LF (0x0A), TAB (0x09), SPACE (0x20) zwischen den serialisierten JSON-Komponenten entfernt.

4 SOCKS-TLS Protokoll

4.1 Einleitung

Diese Detailspezifikation beschreibt die Verwendung von SOCKSv5 und TLS gemäß [RFC1928] und [DRAFT-IETF-AFT-SOCKS-SSL-00] als Steuerungsprotokoll zum Verbindungsaufbau eines TLS-Proxy-Kanals vom SOCKS-Client zum TLS-Proxy als SOCKS-Server, um zu signalisieren, zu welchem Kommunikationspartner im WAN der TLS-Proxy eine TLS-Verbindung aufbauen soll.

Beim Aufbau der TLS-Verbindung zwischen CLS-Kommunikationsadapter und SMGW wird im SOCKS-TLS-Handshake mittels der TLS-Client- und TLS-Server-Zertifikate und der zugehörigen Schlüssel eine Client-Server Authentifizierung durchgeführt. Im SOCKS-Protokoll wird als Zieladresse ein eindeutiger Bezeichner für die TLS-Proxy-Funktion im SMGW übermittelt, der die Zieladresse und Identität des Kommunikationspartners im WAN über die Konfiguration der TLS-Proxy-Funktion im SMGW eindeutig identifiziert. Die TLS-Proxy-Funktion im SMGW initiiert bei berechtigter Proxy-Verbindung eine TLS-Verbindung zur konfigurierten Zieladresse.

4.2 Technische Anwendungsfälle des SOCKS-TLS Clients

4.2.1 TA.SOCKSTLS.InitiateAuthNegotiation

4.2.1.1 Beschreibung

Dieser TA beschreibt die Anforderungen für die Initiierung eines SOCKS-Verbindungsaufbaus zu einem SOCKS-Server gemäß [RFC1928] für TLS-Authentifizierung nach [DRAFT-IETF-AFT-SOCKS-SSL-00].

4.2.1.2 Anforderungen

- Die Implementierung **MUSS** den Verbindungsaufbau zum SOCKS-Server für TCP-basierte Clients gemäß [RFC1928] Kap. 3 für SOCKSv5 (Ver = 0x05) durchführen. [REQ.TA.SOCKSTLS.InitiateAuthNegotiation.10]
- Die Implementierung **MUSS** dem SOCKS-Server die Authentifizierungsmethode "TLS" (0x06) anbieten und akzeptieren und den Ablauf mit Fehler abbrechen, falls der SOCKS-Server mit einer anderen Methode antwortet. [REQ.TA.SOCKSTLS.InitiateAuthNegotiation.20]

  Method-Id=0x06 gemäß IANA Assignments Socks-Methods ([DRAFT-IETF-AFT-SOCKS-SSL-00] verwendet Id=0x86).
- Die Implementierung **DARF KEINE** Subauthentication für die Authentifizierungsmethode "TLS" durchführen. [REQ.TA.SOCKSTLS.InitiateAuthNegotiation.30]

4.2.2 TA.SOCKSTLS.ConnectRequest

4.2.2.1 Beschreibung

Dieser TA beschreibt die Anforderungen an die Initiierung eines Verbindungsaufbaues mit einem Kommunikationspartner im WAN über eine bestehende Verbindung zu einem SOCKS-Server.

Die Identifikation des Proxy-Kommunikationsprofiles im SMGW geschieht über eine IP-Adresse oder einen Server-Namen in der CONNECT-Nachricht.

4.2.2.2 Anforderungen

- Die Implementierung **MUSS** für jede weitere Proxy-Verbindung zu einem Kommunikationspartner im WAN eine neue TLS-Session zum SOCKS-Server aufbauen. [REQ.TA.SOCKSTLS.ConnectRequest.10]
- Die Implementierung **MUSS** SOCKSv5 und das Kommando CONNECT (0x01) gemäß [RFC1928] Kap. 4 verwenden. [REQ.TA.SOCKSTLS.ConnectRequest.20]
- Die Implementierung **MUSS** das CONNECT Kommando mit den Adresstyp "DomainName" (0x03) verwenden, wobei der Parameter DomainName nur die Zeichen 0-9, a-z, - und Punkt enthalten darf. [REQ.TA.SOCKSTLS.ConnectRequest.30]

- Die Implementierung **MUSS** das CONNECT Kommando innerhalb der bestehenden SOCKS-TLS-Verbindung und mit dem SOCKS-Framing State "Data Flow" (0x03) übermitteln. [REQ.TA.SOCKSTLS.ConnectRequest.40]
- Die Implementierung **MUSS** die Replies gemäß [RFC1928] Kap. 6 auswerten, Werte des Datenfelds "REP" ungleich 0 als Fehler interpretieren und den Ablauf mit Fehler abbrechen. [REQ.TA.SOCKSTLS.ConnectRequest.50]
- Die Implementierung **MUSS** die Datenfelder BND.ADDR und BND.PORT der CONNECT-Reply-Nachricht ignorieren, da diese undefiniert sind. [REQ.TA.SOCKSTLS.ConnectRequest.60]
- Die Implementierung **MUSS** nach einem Verbindungsfehler (REP > 0) innerhalb von 10s die TLS-Verbindung trennen. [REQ.TA.SOCKSTLS.ConnectRequest.70]
- Die Implementierung **MUSS** einen neuen Verbindungsaufbau nach einem Verbindungsfehler (REP > 0) mindestens 5 Sekunden bis maximal 60 Sekunden randomisiert verzögern, um eine Überlastung der TLS-Proxy-Funktion zu vermeiden. [REQ.TA.SOCKSTLS.ConnectRequest.80]

4.3 Technische Anwendungsfälle für SOCKS-TLS Client und Server

4.3.1 TA.SOCKSTLS.SendApplicationData

4.3.1.1 Beschreibung

Dieser TA enthält die Anforderungen an den Versand von Anwendungsdaten über eine bestehende SOCKS-TLS-Verbindung.

4.3.1.2 Anforderungen

REQ.TA.SOCKSTLS.SendApplicationData.10

Die Implementierung **MUSS** Anwendungsdaten innerhalb eines TLS-Application-Records senden , die in einem Rahmenformat gemäß [DRAFT-IETF-AFT-SOCKS-SSL-00] gekapselt sind und mit State="0x03" (Data Flow) übermittelt werden.

4.3.2 TA.SOCKSTLS.ReceiveApplicationData

4.3.2.1 Beschreibung

Dieser TA enthält die Anforderungen an den Empfang von Anwendungsdaten über eine bestehende SOCKS-TLS-Verbindung.

4.3.2.2 Anforderungen

REQ.TA.SOCKSTLS.ReceiveApplicationData.10

Die Implementierung **MUSS** die TLS-Application-Records, die in einem Rahmenformat gemäß [DRAFT-IETF-AFT-SOCKS-SSL-00] gekapselt sind und mit State="0x03" (Data Flow) übermittelt werden, verarbeiten.

4.3.3 TA.SOCKSTLS.SendSocksTlsRecord

4.3.3.1 Beschreibung

Dieser TA beschreibt die Anforderungen an die Implementierung zum Versand eines SOCKS-TLS-Records über ein oder mehrere TCP-Segmente. Die Implementierung bestimmt zuvor die Maximum Transmission Unit (MTU) zur Zerlegung von SOCKS-TLS-Records in TCP-Segmente.

4.3.3.2 Anforderungen

REQ.TA.SOCKSTLS.SendSocksTlsRecord.10

Die Implementierung **MUSS** den SOCKS-TLS-Frame gemäß [DRAFT-IETF-AFT-SOCKS-SSL-00] mit "VER" = 0x01 und "STATE" = (0x01-0x04) senden.

4.3.4 TA.SOCKSTLS.ReceiveSocksTlsRecord

4.3.4.1 Beschreibung

Dieser TA beschreibt die Anforderungen an die Implementierung zum Empfang und Assemblieren eines SOCKS-TLS-Records aus ein oder mehreren TCP-Segmenten. Der TA wartet bis ein vollständiger SOCKS-TLS-Record eingetroffen ist.

4.3.4.2 Anforderungen

- Die Implementierung **MUSS** den SOCKS-TLS-Frame gemäß [DRAFT-IETF-AFT-SOCKS-SSL-00] mit "VER" = 0x01 und "STATE" = (0x01-0x04) verarbeiten können. [REQ.TA.SOCKSTLS.ReceiveSocksTlsRecord.10]

4.3.5 TA.SOCKSTLS.ClosingHandshake

4.3.5.1 Beschreibung

Dieser TA beschreibt die Anforderungen an das geordnete Beenden einer SOCKS-TLS-Verbindung durch den SOCKS-TLS-Client oder SOCKS-TLS-Server, nachdem die SOCKS-TLS-Verbindung mit einem CONNECT erfolgreich hergestellt wurde. Sofern die Verbindung durch einen Fehler der TLS-Protokollschicht beendet wird, wird dieser im TLS-Alert übermittelt. Sofern der SOCKS-CONNECT durch einen Fehler beendet wird, wird dieser im REP Datenfeld der CONNECT-Response übermittelt und die TLS-Verbindung beendet. Die Übermittlung des Fehlergrunds nach dem Verbindungsaufbau ist im SOCKS-Protokoll nicht vorgesehen. Der Verbindungsabbau wird durch einen TLS-Alert im SOCKS-TLS-Framing durchgeführt. Anschließend beenden Client und Server die TCP-Verbindung.

Der Ablauf wird initiiert durch den SOCKS-TLS-Client oder SOCKS-TLS-Server, der die Verbindung beenden möchte. Sofern die Verbindung durch einen Fehler der TLS-Protokollschicht beendet wird, wird dieser im TLS-Alert übermittelt. Die Übermittlung des Fehlergrunds im SOCKS-Protokoll ist nicht vorgesehen.

4.3.5.2 Anforderungen

- Die Implementierung **MUSS** die TLS-Alert Records mit SOCKS-TLS-Framing gemäß ▸TA.SOCKSTLS.ReceiveSocksTlsRecord mit "expectedState" = "Closing Handshake" (0x04) empfangen und ▸TA.SOCKSTLS.ReceiveSocksTlsRecord mit "STATE" = "Closing Handshake" (0x04) senden. [REQ.TA.SOCKSTLS.ClosingHandshake.10]
- Die Implementierung **MUSS** auf einen empfangenen SOCKS-TLS-"Closing Handshake" mit einem SOCKS-TLS-"Closing Handshake" antworten und die zugehörige Transportverbindung innerhalb von 5 Sekunden beenden. [REQ.TA.SOCKSTLS.ClosingHandshake.20]

5 mDNS - Device/Service Discovery Protokoll

5.1 Einleitung

Dieses Kapitel beschreibt Anforderungen an die automatische, dynamische Adresskonfiguration, Device- und Service Discovery mit dem Multicast Domain Name Service Protokoll (mDNS) und Domain Name Service Service Discovery (SD) Protokoll. Multicast DNS und DNS-SD werden verwendet, um eine automatische Konfiguration von Geräten und Diensten in einem Netzwerk zu ermöglichen. Zusammen mit einer automatischen IP-Adressvergabe wird dieses Konzept "Zeroconf" genannt. Ein mDNS/DNS-SD Responder (siehe [RFC6762] Kapitel 6 und [RFC6763]) beantwortet Anfragen eines mDNS/DNS-SD Queriers.

Der mDNS Responder des SMGW beantwortet DNS-Adressanfragen nach <SMGW-ID>.local." (unique resource) und "smgw.local." (shared resource) mit der link-lokalen Netzwerkadresse des SMGW.

mDNS verwendet die Datenstrukturen des DNS-Protokolls und kommuniziert über Port 5353 mittels UDP-Protokoll und Multicast-IPv4 und/oder IPv6.

mDNS verwendet DNS-A-Records zum Mitteilen von IPv4-Adressen und DNS-AAAA-Records zum Mitteilen von IPv6-Adressen der Geräteschnittstelle im (lokalen) Netzwerk und kann IP-Adressen über DNS-PTR-Records zu Hostnamen auflösen.

DNS-SD verwendet DNS-SRV-Records zur Beschreibung der Dienstzugriffsadressen (z.B. Port-Nummer) und DNS-TXT-Records zur Übermittlung von detaillierteren, kontextabhängigen Dienstzugangsinformationen eines Gerätes im Netzwerk.

mDNS/DNS-SD bietet keinen Schutz vor manipulierten, verzögerten oder ausgespähten mDNS/DNS-SD-Nachrichten. Um einen gewissen Schutz zu erreichen, darf das Protokoll deshalb nur link-lokal und in physisch oder kryptografisch geschützten Netzwerken innerhalb von Liegenschaften verwendet werden.[1] Eine Betrachtung von Angreifer-Modellen zur Privacy von DNS-SD findet sich in [RFC8882].

5.2 Technische Anwendungsfälle für das mDNS Querying

Die folgenden Abschnitte enthalten die TA für die Implementierung des mDNS Querying zur Ermittlung von IP-Adressen und Service-Namen eines IP-Hosts im lokalen Netzwerk.

5.2.1 Beschreibung

Aussendung einer Multicast Abfrage an einen Host, der über einen eindeutigen ("Unique") oder generischen ("Shared") Hostnamen in einem lokalen Netzwerk identifiziert wird und Verarbeitung der Antwort.

5.2.2 Anforderungen

REQ.TA.mDNS.Querying.10

Die Implementierung **MUSS** die Anforderungen (MUST, SHOULD, MUST NOT, SHOULD NOT) an das mDNS Querying gemäß [RFC6762] erfüllen. Dazu gehören insbesondere die Anforderungen in Kapitel 6 (Responding) und Kapitel 8 (Probe and Announce).

REQ.TA.mDNS.Querying.20

Die Implementierung **MUSS** die Anforderungen an das Erzeugen von mDNS-Responses nach ▸Abschnitt 5.2.3 erfüllen.

REQ.TA.mDNS.Querying.30

Die Implementierung **MUSS** die Anforderungen an das Verarbeiten von mDNS-Queries nach ▸Abschnitt 5.2.4 erfüllen.

[1] Die Authentizität und Vertraulichkeit der Anwendungsdaten der mittels mDNS/DNS-SD bekanntgegebenen Dienste wird durch das TLS-Protokoll gewährleistet.

REQ.TA.mDNS.Querying.40

Die Implementierung **MUSS** die Queries nach *uniqueNames*[], *sharedNames*[] mit den link-lokalen IP-Adressen der Schnittstelle beantworten, über die die Anfrage empfangen wurde.

REQ.TA.mDNS.Querying.50

Die Implementierung **SOLL** die Anforderungen (MUST, SHOULD, MUST NOT, SHOULD NOT) an DND-SD nach [RFC6763] erfüllen.

5.2.3 TA.mDNS.BuildQuery

5.2.3.1 Beschreibung

Dieser TA beschreibt die Anforderungen an das Erzeugen einer mDNS Query.

5.2.3.2 Anforderungen

REQ.TA.mDNS.BuildQuery.20

Die Implementierung **SOLL** gewährleisten, dass sich die Antworten auf über IPv4 bzw. IPv6 empfangene Anfragen nicht unterscheiden.

REQ.TA.mDNS.BuildQuery.30

Die Implementierung **MUSS** Multicast mDNS-Nachrichtenaussendungen auf 30 pro Minute begrenzen.

5.2.4 TA.mDNS.ProcessResponse

5.2.4.1 Beschreibung

Dieser TA beschreibt die Anforderungen an das Verarbeiten von mDNS-Antwort-Nachrichten zur Ermittlung von IP-Adressen von selektierten mDNS-Hosts und Port-Nummern zur Service-Namen.

5.2.4.2 Anforderungen

- Die Implementierung **MUSS** die Nachricht *queryRequest* nach den Anforderungen von [RFC6762] und [RFC6763] verarbeiten und im Fehlerfall den Ablauf mit Fehler beenden. [REQ.TA.mDNS.ProcessResponse.10]
- Die Implementierung **SOLL** die Implementierung über IPv6 empfangene mDNS-Nachrichten verarbeiten können. [REQ.TA.mDNS.ProcessResponse.20]

6 TLS - Transport Layer Security Protokoll

6.1 Einleitung

TLS ermöglicht einen vertraulichen, integritätsgeschützten und gegenseitig authentifizierten, bidirektionalen Transportkanal zwischen zwei Anwendungen. TLS ist ein hybrides Verschlüsselungsprotokoll, dessen Aufgabe die Sicherung auf der Ebene der Transportschicht (Schicht 4 im OSI-Referenzmodell [X.200]) ist und dessen Funktionen bis in die Anwendungsschicht (Schicht 7) hineinreichen.

[RFC8446] präzisiert Interoperabilitäts-Anforderungen an Implementierungen, die sowohl das Protokoll TLS 1.3 als auch ältere Protokollversionen unterstützen. Da die Kommunikationspartner des iMSys keine TLS-Versionen vor 1.2 unterstützen dürfen, enthält diese Detailspezifikation Anforderungen an Implementierungen die ausschließlich TLS 1.2 gemäß [RFC5246] unterstützen und solche Implementierungen die sowohl TLS 1.2 gemäß [RFC5246] als auch TLS 1.3 gemäß [RFC8446] unterstützen. [RFC8446] stellt Anforderungen an TLS 1.2 Implementierungen, die der Verbesserung der Sicherheit und Interoperabilität dienen, und umgesetzt werden sollen. Die Interoperabilitätsanforderungen dieser Detailspezifikation folgen den Sicherheitsvorgaben der [TR-03109-3].

Die Anforderungen an die TLS-Implementierungen werden dabei in unterteilt in

- Allgemeine Anforderungen an die Interoperabilität beim Handshake, der Anwendungsdatenübertragung und dem Verbindungsende
- Ergänzende Anforderungen an den TLS-Handshake im HAN (Client, Server)

Übersicht über die in dieser Detailspezifikation referenzierten Spezifikationen:

- TLS 1.3 gemäß [RFC8446] und TLS 1.2-Handshake gemäß [RFC5246] mit den TLS 1.2 spezifischen Anforderungen aus [RFC8446]. [RFC8446] beschreibt u.a. das Handshake-Protokoll mit TLS 1.3-Hello-Extensions und Handshake-Nachrichten, [RFC5246] das Handshake-Protokoll mit TLS 1.2-Hello-Extentions- und Handshake-Nachrichten.
- Cipher Suiten, kryptografische Algorithmen und Parameter und weitere Anforderungen gemäß der in [TR-03109-3] referenzierten Spezifikationen.
  - Elliptische Kurven für TLS 1.2 gemäß [RFC8422], [RFC7027], [RFC5289]
  - encrypt_then_mac Extension gemäß [RFC7366] für AES-CBC Cipher Suiten in TLS 1.2
  - extended_master_secret gemäß [RFC7627] für den TLS 1.2-Handshake
  - Keine gmt_unix_time im client_hello des TLS 1.2-Handshake
- server_name Extension gemäß [RFC6066]
- application_layer_protocol_negotiation Extension gemäß [RFC7301]
- Die Anforderungen an TLS-Zertifikate basierend auf [RFC5280] sind in Zertifikatsprofilen in [TR-03109-4] und [DS] beschrieben.

6.2 Datenstrukturen

Folgende Daten werden von der Implementierung für das TLS-Protokoll gemäß dieser Detailspezifikation verwendet:

- Zur TLS-Authentifizierung der Implementierung gegenüber dem Kommunikationspartner: Privater ECC-Schlüssel, X.509 TLS-Zertifikat mit zugehörigem öffentlichem ECC-Schlüssel, die die kryptografische Identität der Implementierung darstellen.
- Zur TLS-Authentifizierung des Kommunikationspartners über PKI-Vertrauensanker: Ein oder zwei Vertrauensanker des Ausstellers (CA) zur Validierung der Zertifikatskette (*trustedCAs*). Für Zertifikats-Wechselprozesse werden zur Verbesserung der Verfügbarkeit zwei Vertrauensanker empfohlen. Die Implementierung und der Kommunikationspartner müssen sich im Überlappungszeitraum der Gültigkeit über das zu verwendende Zertifikat abstimmen.

- Zur TLS-Authentifizierung des Kommunikationspartners über "Direct-Trust": Ein TLS-X.509-Zertifikat zur Berechtigungsprüfung auf direkte Übereinstimmung, das für die Authentifizierung und Verschlüsselung der TLS- und Anwendungsdaten verwendet wird.
- Anwendungsparameter für den Sitzungsaufbau (Handshake-Mode (Client/Server/Full/Resumed) und Parameter für Hello-Extensions wie ALPN, SNI), die sich u.a. aus dem zugrundeliegenden Kommunikationsszenario ergeben.
- Flüchtige (interne) Zustandsdaten während der Lebensdauer einer TLS-Session gemäß [RFC8446] an die keine Interoperabilitätsanforderungen gestellt werden.
- Anwendungsdaten, die über TLS transportiert werden.

Die verwendeten TLS-Zertifikate sind im TLS 1.2-Handshake an den Schnittstellen beobachtbar, im TLS 1.3 Handshake werden sie verschlüsselt übertragen.

6.3 Technische Anwendungsfälle

Dieser Abschnitt umfasst die technischen Anwendungsfälle für TLS. Zu Beginn werden Anforderungen an alle Implementierungen definiert. Die folgenden Abschnitte ergänzen die allgemeinen Anforderungen um rollenspezifischen Anforderungen für TLS-Server und TLS-Client.

6.3.1 TA.TLS.Handshake

6.3.1.1 Beschreibung

Das TLS-Handshake-Protokoll hat die Aufgabe, die sichere Authentifizierung der Kommunikationspartner durchzuführen und die Rahmenbedingungen für einen vertraulichen Kommunikationskanal zu etablieren. Es gibt unterschiedliche Varianten des TLS 1.2-Handshakes (Mit oder ohne Serverside/Clientside State Resumption) und des TLS 1.3-Handshakes (bspw. mit oder ohne Kenntnis eines vorher vereinbartem Pre-Shared-Key) um den TLS-Verbindungsaufbau zu beschleunigen. Im Folgenden werden die Anforderungen definiert, die für alle Schnittstellen ungeachtet der Rolle gelten.

6.3.1.2 Ablaufbeschreibung

▸Abbildung 6.1 und ▸Abbildung 6.2 zeigt den Ablauf eines TLS 1.2-Handshakes zwischen TLS-Client und TLS-Server.

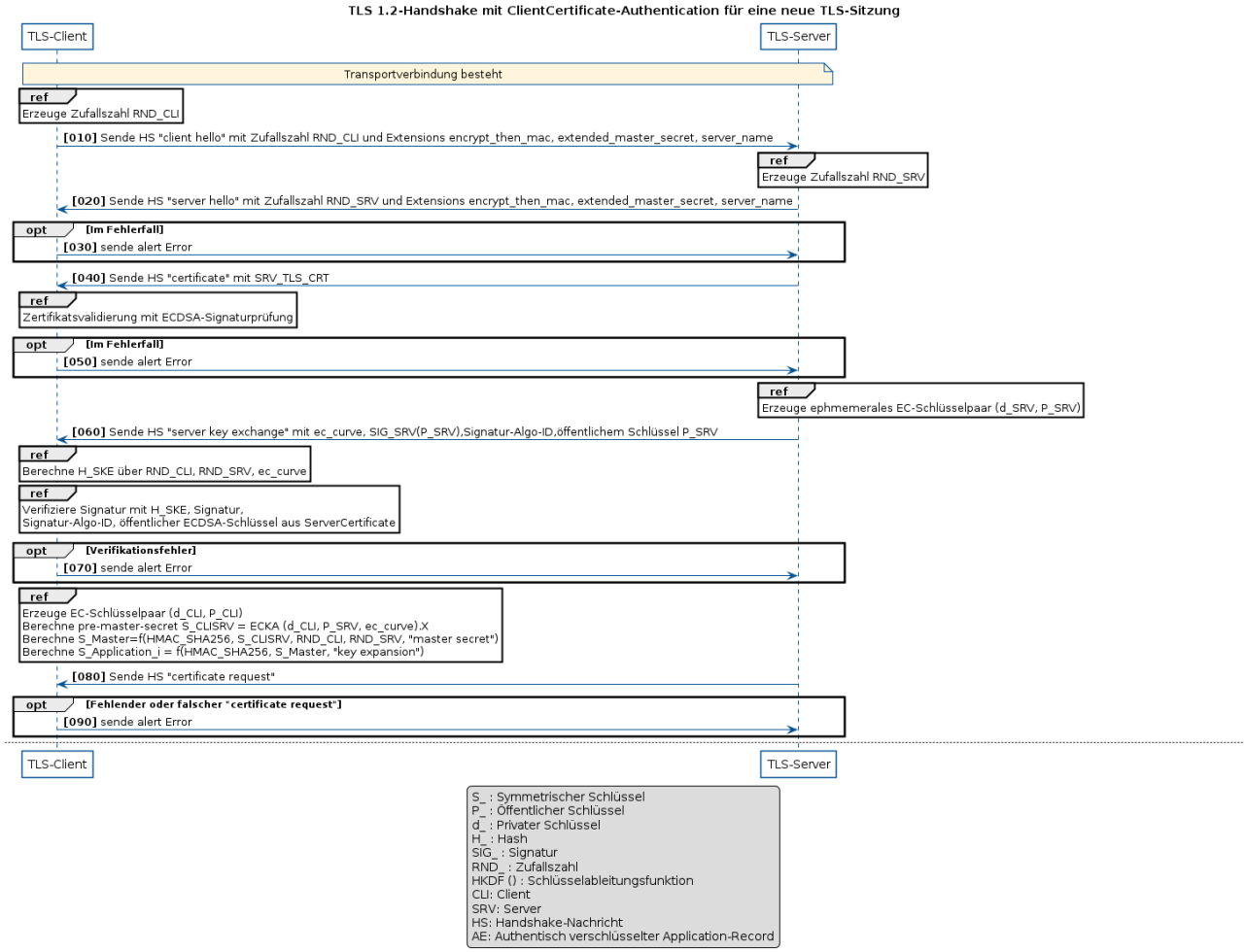

Abbildung 6.1. Ablauf eines TLS 1.2 Handshakes für eine neue TLS-Sitzung (1/2)

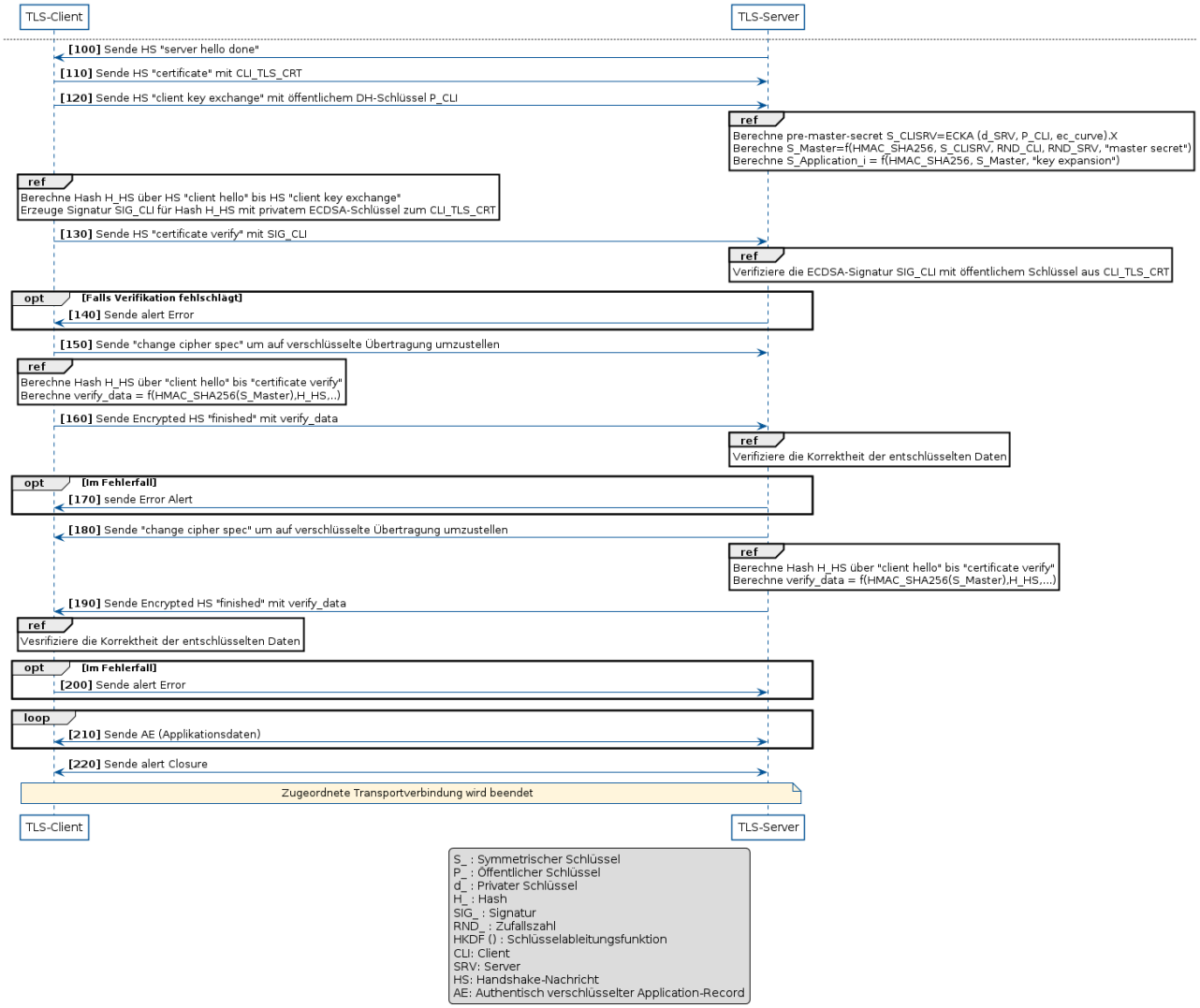

Abbildung 6.2. Ablauf eines TLS 1.2 Handshakes für eine neue TLS-Sitzung (2/2)

▸Abbildung 6.3 und ▸Abbildung 6.4 zeigt den Ablauf eines TLS 1.3-Handshakes zwischen TLS-Client und TLS-Server.

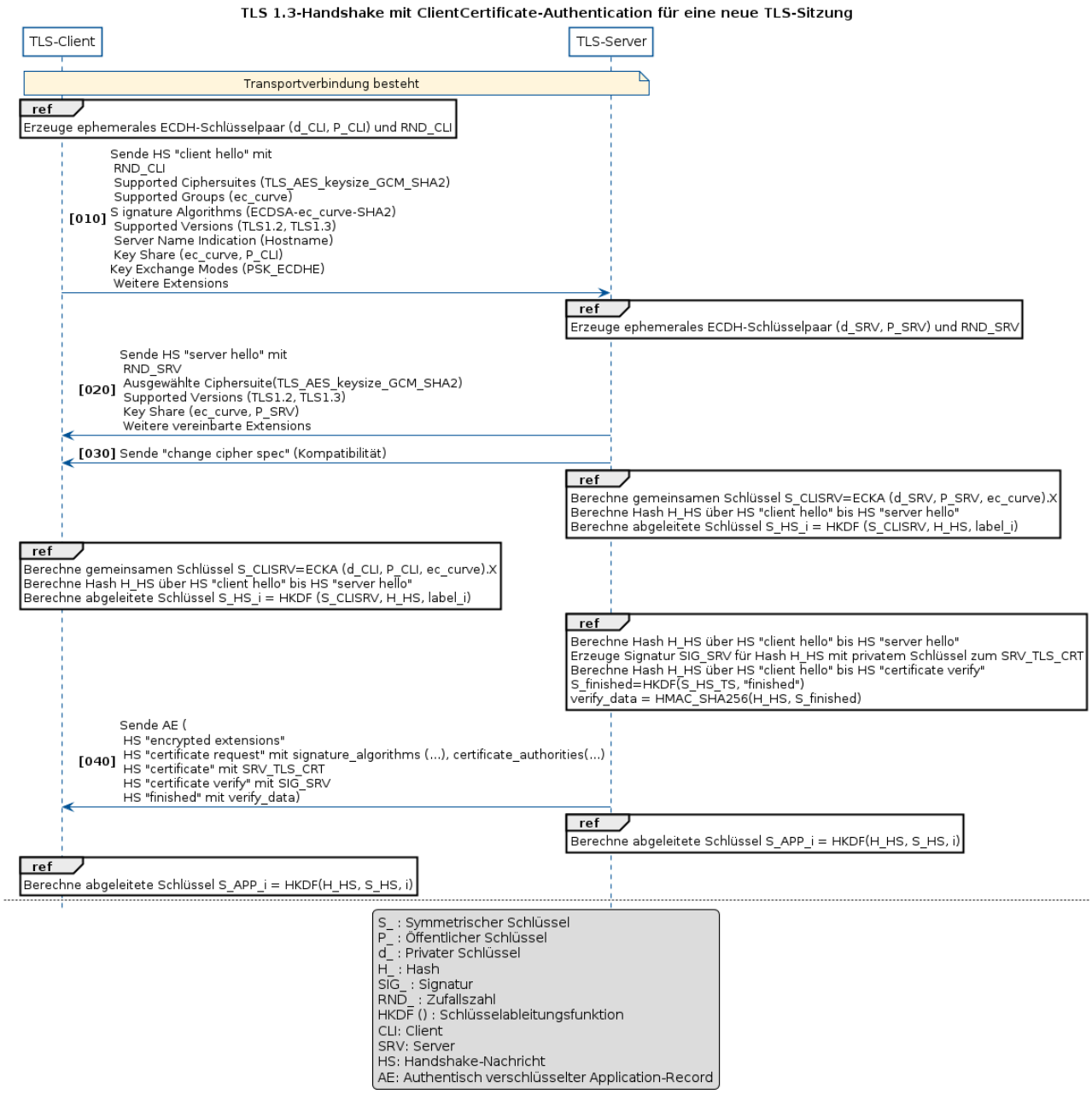

Abbildung 6.3. Ablauf eines TLS 1.3-Handshakes für eine neue TLS-Sitzung (1/2)

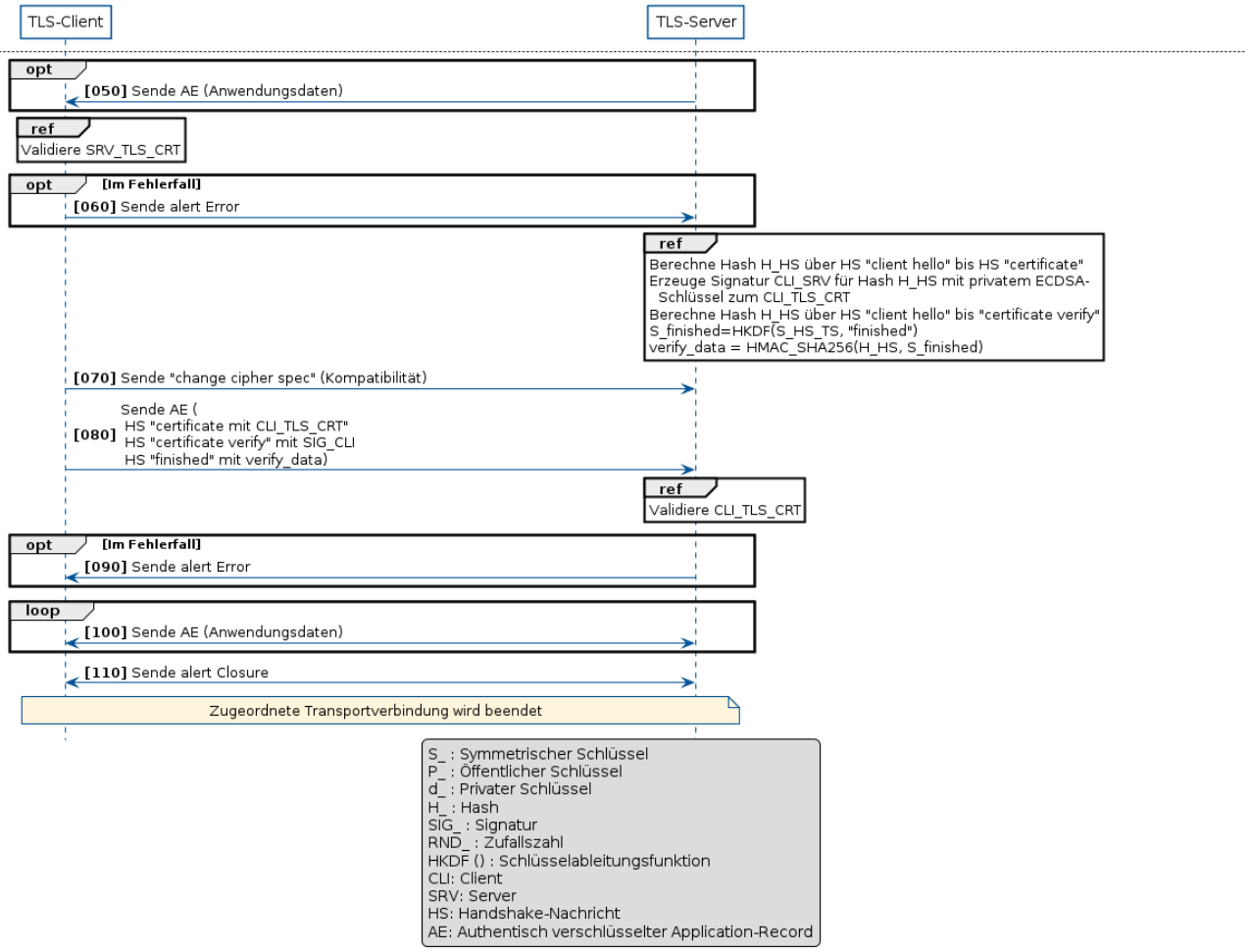

Abbildung 6.4. Ablauf eines TLS 1.3-Handshakes für eine neue TLS-Sitzung (2/2)

6.3.1.3 Anforderungen

REQ.TA.TLS.Handshake.10

Die Implementierung **MUSS** den TLS-Handshake der Version 1.2 nach [RFC5246] Kapitel 7.3 und 7.4 unterstützen.

REQ.TA.TLS.Handshake.20

Die Implementierung **MUSS** die verwendeten Zertifikate gemäß ihres Zertifikatsprofiles auf Integrität, zeitliche Gültigkeit und Schlüsseleignung im Kontext des Zugriffs prüfen.

REQ.TA.TLS.Handshake.30

Die Implementierung **MUSS** die nach [TR-03109-3] für die HAN-Schnittstelle genannten, mindestens zu unterstützenden Cipher Suiten und elliptischen Kurven akzeptieren.

REQ.TA.TLS.Handshake.50

Die Implementierung **MUSS** im TLS 1.2-Handshake die von der Implementierung unterstützten NamedCurveIDs (alias SupportedGroupsID) gemäß [RFC8422] Kapitel 5.1 senden bzw. akzeptieren.
▸ICS.TA.TLS.HanHandshake.30

REQ.TA.TLS.Handshake.60

Die Implementierung **SOLL** NamedCurveIds für höhere Ordnungen bzw. Schlüssellängen an den Anfang der Liste der supported_groups-Extension stellen.

REQ.TA.TLS.Handshake.70

Falls die Implementierung den TLS 1.3-Handshake unterstützt, **MUSS** die Implementierung die unterstützten NamedCurveIDs (alias SupportedGroupsID) der in [TR-03109-3] erlaubten Kurvenparameter gemäß [RFC8446] Kap 4.2.7 und [RFC8734] senden bzw. akzeptieren.

REQ.TA.TLS.Handshake.80

Die Implementierung **SOLL** für den TLS 1.2-Handshake die encrypt_then_mac Extension gemäß [RFC7366] unterstützen.

REQ.TA.TLS.Handshake.90

Die Implementierung **SOLL** für den TLS 1.2-Handshake die extended_master_secret Extension gemäß [RFC7627] unterstützen.

REQ.TA.TLS.Handshake.110

Die TLS-Client Implementierung **MUSS** den TLS-Handshake mit Alert "fatal" beenden, falls der Server Hello-Extensions aus [RFC6066] in einem "resumed" Handshake sendet.

REQ.TA.TLS.Handshake.120

Die Implementierung **SOLL** im ClientHello des TLS 1.2-Handshake anstelle der eigenen Zeit eine Zufallszahl als gmt_unix_time verwenden.

REQ.TA.TLS.Handshake.130

Die Implementierung **MUSS** den TLS-Handshake nach Zertifikatsvalidierungs- oder Authentifizierungsfehlern mit Alert "fatal" beenden.

REQ.TA.TLS.Handshake.140

Die Implementierung als TLS-Client **SOLL** im TLS1.2-Handshake in der Nachricht "ClientHello" die Extension "trusted_ca_keys" gemäß [RFC6066] mit einer Liste von X.501-DistinguishedNames der in der Implementierung administrativ konfigurierten (CA-)Zertifikate zur Authentifizierung des TLS-Server übermitteln, sofern mehr als ein CA-Zertifikat zur Authentifizierung konfiguriert ist (▸ICS.TA.TLS.Handshake.20).

REQ.TA.TLS.Handshake.145

Die Implementierung als TLS-Server **SOLL** im TLS1.2-Handshake in der Nachricht "CertificateRequest" in "certificate_authorities" eine Liste von X.501-DistinguishedNames der in der Implementierung administrativ konfigurierten (CA-)Zertifikate zur Authentifizierung des TLS-Clients übermitteln, sofern mehr als ein CA-Zertifikat zur Authentifizierung konfiguriert ist (▸ICS.TA.TLS.Handshake.30).

REQ.TA.TLS.Handshake.150

Die Implementierung als TLS-Client **MUSS** im TLS1.3-Handshake in der Nachricht "ClientHello" die Extension "certificate_authorities" gemäß [RFC8446] mit einer Liste von X.501-DistinguishedNames der in der Implementierung administrativ konfigurierten (CA-)Zertifikate zur Authentifi-

zierung des TLS-Server übermitteln, sofern mehr als ein CA-Zertifikat zur Authentifizierung konfiguriert ist.

REQ.TA.TLS.Handshake.155

Die Implementierung als TLS-Server **MUSS** im TLS1.3-Handshake in der Nachricht "CertificateRequest" die Extension "certificate_authorities" gemäß [RFC8446] mit einer Liste von X.501-DistinguishedNames der in der Implementierung administrativ konfigurierten (CA-)Zertifikate zur Authentifizierung des TLS-Clients übermitteln, sofern mehr als ein CA-Zertifikat zur Authentifizierung konfiguriert ist.

REQ.TA.TLS.Handshake.160

Die Implementierung des TLS-Clients **MUSS** ein empfangenes TLS-Server-Zertifikat vor der Verwendung mit Certificate Path Validation gemäß [RFC5280] bis zu einem administrativ konfigurierten CA-Zertifikat valideren. Für den TLS-Verbindungsaufbau zur Zeitsynchronisation kann auf die zeitliche Gültigkeitsprüfung verzichtet werden, solange keine vertrauenswürdige Zeit in der Implementierung verfügbar ist.

REQ.TA.TLS.Handshake.170

Sofern gemäß Kommunikationsszenario die Implementierung des TLS-Clients selbstsignierte TLS-Server Zertifikate akzeptieren kann, **MUSS** die Implementierung ein empfangenes, selbstsigniertes Server-Zertifikat vor der Verwendung auf Übereinstimmung mit einem für die Berechtigungsprüfung administrativ konfigurierten TLS-Zertifikat prüfen. Auf die zeitliche Gültigkeitsprüfung kann verzichtet werden.

REQ.TA.TLS.Handshake.180

Die Implementierung eines TLS-Servers **MUSS** im TLS 1.3-Handshake die Rückwärtskompatibilität mit älteren Clients gemäß [RFC8446] Annex D.2 unterstützen (supported_versions Extension)

REQ.TA.TLS.Handshake.190

Die Implementierung eines TLS-Clients **MUSS** im TLS 1.3-Handshake die Rückwärtskompatibilität mit älteren Servern gemäß [RFC8446] Annex D.1 unterstützen (supported_versions Extension)

6.3.1.4 Implementation Conformance Statements

ICS.TA.TLS.Handshake.10

Der Hersteller **MUSS** erklären, ob die Implementierung in der supported_groups extension die NamedCurveIds für höhere Schlüssellängen an den Anfang stellt.

ICS.TA.TLS.Handshake.20

Der Hersteller **MUSS** erklären, ob die Implementierung als TLS 1.2-Client die CA-Zertifikate in der trusted_ca_keys Extension der ClientHello-Nachricht sendet.

ICS.TA.TLS.Handshake.30

Der Hersteller **MUSS** erklären, ob die Implementierung als TLS 1.2-Server die CA-Zertifikate in certificate_authorities der CertificateRequest-Nachricht sendet.

6.3.1.5 Implementierungshinweise

1. Zur Vereinbarung eines gemeinsamen Vertrauensankers und zur Vereinfachung von Wechselprozessen des Vertrauensankers für die Zertifikatsvalidierung wird für alle Implementierungen empfohlen, die von der Implementierung verwendeten Vertrauensanker (Direct-Trust oder RootCA-Zertifikat) dem Kommunikationspartner im TLS-Handshake zu präsentieren. Für TLS 1.3 wird die Unterstützung der Certificate-Authorities Extension, für TLS 1.2 die Unterstützung der certificate_authorities im Certificate Request empfohlen.
2. Da in den folgenden Jahren noch Implementierungen genutzt werden, die ausschließlich TLS 1.2 unterstützen, ist eine Unterstützung sowohl des TLS 1.3-Handshake gemäß [RFC8446] als auch die Rückwärtskompatibilität mit einer reinen TLS1.2 Implementierung gemäß [RFC8446] Kap. D.1 (als TLS-Client) bzw. D.2 (als TLS-Server) notwendig.
3. Aufgrund der erforderlichen Rückwärtskompatibilität ist es erforderlich gemäß [TR-03109-3] sowohl AES-GCM- als auch AES-CBC-Ciphersuites im TLS 1.2-Handshake zu unterstützen. Dies erfordert die Unterstützung der TLS-Hello Extension encrypt_then_mac ([RFC7366]).
4. Kryptografische Agilität unterstützt die längerfristige Verwendung einer Implementierung. Deshalb wird empfohlen, die Implementierung (Hardware und Software) bereits für die künftig empfohlenen Verfahren gemäß [TR-03109-3] auszulegen. Eine Implementierung sollte inbesondere im TLS-Zertifikate mit NIST-P256/brainpoolP256r1/brainpoolP384r1/brainpoolP512r1 Kurvenparametern verarbeiten und speichern können (sowohl eigene als auch die des Kommunikationspartners).
5. Zur Verringerung der Datenmenge im Kommunikationsnetz, zur Beschleunigung des Verbindungsaufbaus und der Reduktion der Berechnungszeit für asymmetrische Kryptografie wird empfohlen innerhalb der maximalen TLS-Session-Lebensdauer (siehe [TR-03109-3] die PSK-basierte TLS-Session-Resumption unterstützen.

6.3.2 TA.TLS.Close

Die folgenden Anforderungen an das Verhalten beim Beenden der TLS-Verbindung ergänzen oder präzisieren die Anforderungen der TLS Spezifikationen.

REQ.TA.TLS.Close.10

Die Implementierung **MUSS** nach empfangenen TLS-Alerts [RFC8446] Kapitel 6 mit Level "fatal" die TLS-Verbindung schließen, so dass keine weiteren Anwendungsdaten über die TLS-Verbindung ausgetauscht werden.

REQ.TA.TLS.Close.20

Die Implementierung **SOLL** nach empfangenen TLS-Alerts mit Level "fatal" für den Betreiber/Administrator ein Ereignis protokollieren.

REQ.TA.TLS.Close.30

Die Implementierung **MUSS** Alerts gemäß [RFC8446] Kapitel 6 erzeugen können.

REQ.TA.TLS.Close.40

Die Implementierung **MUSS** Alerts gemäß [RFC8446] Kapitel 6 verarbeiten können.

REQ.TA.TLS.Close.60

Die TLS 1.2-Implementierung **MUSS** die Verbindung schließen, wenn application_data TLS-Records vor Abschluss des Handshakes eintreffen

REQ.TA.TLS.Close.70

Die TLS 1.2-Implementierung **MUSS** im AES-CBC- Mode TLS-Fragmente verarbeiten können, die ein AES-CBC-TLS-Block-Padding mit einer Länge von insgesamt 16-256 Bytes enthalten. Wird ein längerer Record empfangen, wird ein einen TLS-Alert "record_overflow" gesendet und die TLS-Verbindung beendet.

REQ.TA.TLS.Close.80

Die Implementierung **MUSS** innerhalb von 20s nach dem Senden eines TLS-close_notify-Alert nach [RFC8446] Kapitel 6.1 die damit verknüpfte Transportverbindung beenden.

REQ.TA.TLS.Close.90

Wird die Transportverbindung nicht innerhalb von 20 Sekunden nach Empfang eines close_notify geschlossen, **MUSS** die Implementierung die Verbindung einseitig beenden.

6.3.3 TA.TLS.OpenHanSessionAsClient

6.3.3.1 Beschreibung

Dieser Abschnitt beschreibt die Anforderungen für den TLS-Handshake, der durch den TLS-Client im HAN initiiert wird.

6.3.3.2 Anforderungen

REQ.TA.TLS.OpenHanSessionAsClient.10

Die Implementierung **SOLL** den TLS-Handshake der Version 1.3 nach [RFC8446] Kapitel 4 unterstützen.

REQ.TA.TLS.OpenHanSessionAsClient.20

Die Implementierung **SOLL** die nach [TR-03109-3] für die HAN-Schnittstelle genannten, **empfohlenen** Cipher Suiten und elliptischen Kurven unterstützen.

REQ.TA.TLS.OpenHanSessionAsClient.30

Der Implementierung des TLS-Clients **MUSS** in der ClientHello-Nachricht eine ClientHello-Extension "server_name" vom Typ host_name(0) nach [RFC6066] Kapitel 3 übermitteln. Der server_name MUSS einen Bezeichner im DNS-Format nach [RFC1035] Kapitel 2.3.1 enthalten. Der Bezeichner ist gemäß Kommunikationsszenario zu wählen.

REQ.TA.TLS.OpenHanSessionAsClient.40

Die Implementierung **KANN** im ClientHello weitere Extensions anbieten, sofern diese nicht durch [TR-03109-3] unzulässig sind und für die Interoperabilität die Antwort vom Server unerheblich ist.

6.3.3.3 ICS

ICS.TA.TLS.HanHandshake.10

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die Implementierung den TLS 1.3-Handshake gemäß [RFC8446] Kapitel 4 unterstützt.

ICS.TA.TLS.HanHandshake.20

Der Hersteller MUSS für die HAN-Schnittstelle deklarieren, welche Cipher Suiten die Implementierung anbietet bzw. akzeptiert.

ICS.TA.TLS.HanHandshake.30

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, welche namedCurveIDs die Implementierung für das Signaturverfahren ECDSA und das Schlüsselaustauschprotokoll ECDHE unterstützt (d.h. in der supported_groups-Extension bzw. ellipitic_curves-Extension anbietet).

ICS.TA.TLS.HanHandshake.40

Der Hersteller **MUSS** für die HAN-Schnittstelle die von der Implementierung unterstützten Zertifikats-Signatur-Algorithmen (signature_algorithms) deklarieren.

ICS.TA.TLS.HanHandshake.50

Der Hersteller **MUSS** für die HAN-Schnittstelle die zusätzlich zu den in [TR-03109-3] und [RFC8446] Kapitel 9.2 verpflichtend zu unterstützenden Client/Server-Hello-Extensions implementierten, Extensions deklarieren.

ICS.TA.TLS.HanHandshake.60

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die Implementierung die TLS 1.2-Serverside-Stateful Session-Resumption mit Abbreviated Handshake basierend auf Pre-Shared Key nach [RFC5246] Kapitel 7.3 unterstützt.

ICS.TA.TLS.HanHandshake.70

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die Implementierung den verkürzten TLS 1.3-Handshake basierend auf einem in einer vorherigen TLS-Session erzeugten PSK gemäß [RFC8446] Kapitel 2.2 unterstützt.

ICS.TA.TLS.HanHandshake.80

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die Implementierung für den TLS 1.2-Handshake die Extension encrypt_then_mac gemäß [RFC7366] unterstützt.

ICS.TA.TLS.HanHandshake.90

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die Implementierung für den TLS 1.2-Handshake die Extension extended_master_secret gemäß [RFC7627] unterstützt.

ICS.TA.TLS.HanHandshake.100

Der Hersteller **MUSS** für die HAN-Schnittstelle deklarieren, ob die gmt_unix_time in der von der Implementierung gesendeten ClientHello-Nachricht durch eine Zufallszahl ersetzt wird.

6.3.4 TA.TLS.OpenHanSessionAsServer

6.3.4.1 Beschreibung

Dieser Abschnitt beschreibt die ergänzenden Anforderungen für den TLS-Handshake, eines TLS-Servers im HAN.

6.3.4.2 Anforderungen

REQ.TA.TLS.OpenHanSessionAsServer.10

Die Implementierung **SOLL** an der HAN-Schnittstelle als Server den TLS-Handshake der Version 1.3 nach [RFC8446] Kapitel 4 unterstützen.

REQ.TA.TLS.OpenHanSessionAsServer.20

Sofern die Implementierung ein SM-PKI-TLS-Zertifikat mit ExtendedKeyUsage "TLS-Client" verwendet, **SOLL** dies sowohl für TLS-Client- als auch für TLS-Server-Authentifizierung verwenden werden können.

6.3.4.3 ICS

Siehe ▸Abschnitt 6.3.3.3

7 NTP - Network Time Protocol

7.1 Einleitung

Dieses Kapitel beschreibt Anforderungen an die Nachrichten zur Zeitsynchronisation mit dem Network-Time-Protokoll (NTP) bzw. Simple Network Time Protocol (SNTP). Es konkretisiert die Anforderungen der Spezifikation [RFC5905] zur Implementierung von NTP. Die Anforderungen werden dabei zur Referenzierung aus Kommunikationsszenarien in die Kommunikationsrollen (Client, Server) und die Dienstprimitive (Request, Response) unterschieden.

Die Beschreibung der Anforderungen an die Sicherung und den Aufbau der Kommunikationsverbindung für NTP sind nicht Bestandteil dieses Kapitels.

7.1.1 Abweichungen von RFC5905

- Die NTP-Authentication wird nicht verwendet, da die Authentifizierung zertifikatsbasiert durch TLS realisiert wird.
- Die NTP-Extension-Felder werden nicht verwendet.
- Der UDP-Header ist nicht Bestandteil der NTP-Nachricht.
- Die Port-Nummer des Transport-Layers weicht aufgrund des TLS-Transports von [RFC5905] ab.
- Der Transport findet über ein verbindungsorientiertes und nicht über verbindungsloses Transportprotokoll statt.
- Es wird nur die Association-Betriebsart Client/Server Mode (ephemeral) verwendet.
- Eine Anforderung an die Implementierung zur Verarbeitung von sogenannten Kiss-Codes[1] besteht nicht.
- Falls die Systemzeit in der Implementierung nicht valide oder unbekannt ist, wird der erste vom Zeitserver authentisch empfangene Zeitstempel zum Setzen der Systemzeit verwendet, um keinen manuellen Benutzereingriff zu erfordern.
- Zur Verbesserung des Datenschutzes empfiehlt diese Spezifikation gemäß [RFC8633] für NTP-Clients die im Request ausgesandten Informationen zu minimieren.

7.2 Datenstrukturen

7.2.1 Einleitung

Diese Detailspezifikation verwendet folgende Datenstrukturen:

- Die Anfrage an den Zeitserver und die Antwort vom Zeitserver im NTP Packet-Header-Format (ohne UDP Header) siehe ▸NTPMessage.

7.2.2 NTPMessage

Die Datenstruktur NTPMessage entspricht dem NTP-Packet-Header Format nach [RFC5905] Figure 8 und wird hier zur Förderung des Verständnisses mit den in dieser Detailspezifikation referenzieren Bezeichnern wiederholt. Bytefolgen, die Decimal- oder Integer-Datentypen darstellen, werden in Network Byte-Order (MSB first) übertragen.

Datenfeld	Datentyp (Wertebereich)	Beschreibung	Datenminimierung
leap	Bitstring 2 Bit (0-3)	Information des Servers zu Schaltsekunden nach [RFC5905] Figure 9.	0

[1] Kiss-of-Death Packets enthalten vier Zeichen Steuercodes im Datenfeld "refId", die vom NTP-Server zum Steuern des Clients gesendet werden können. Eine nach RFC5905 vorgesehene Verwendung ist das Verlangsamen der Abfragerate des NTP-Clients.

Datenfeld	Datentyp (Wertebereich)	Beschreibung	Datenminimierung
version	Bitstring 3 Bit (4)	Version des NTP-Protokolls.	4
mode	Bitstring 3 Bit (3-4)	Assoziations-Modus nach [RFC5905] Figure 10.	3
stratum	Integer 1 Byte (0-16)	Stratum des Servers nach [RFC5905] Figure 11.	0
poll	Integer 1 Byte (4-17)	log2 (Poll-Intervall in Sekunden)	0 oder Poll-Intervall
precision	Integer 1 Byte (-18..0)	log2 (Präzision der Systemuhr in Sekundenbruchteilen)	32
rootDelay	Decimal (2 Byte Seconds, 2 Byte Fraction)	Gesamte Round-Trip-Zeit vom Client zum Root-Zeitserver und zurück zum Client in Sekunden/Sekundenbruchteilen (vom Server).	0
rootDispersion	Decimal (2 Byte Seconds, 2 Byte Fraction)	Streuung der Zeit bezüglich der Zeit des Root-Zeitservers in Sekunden/Sekundenbruchteilen (vom Server).	0
refId	Octet-String 4 Byte	• Identifikation des Zeitservers zur Vermeidung von Synchronisationsschleifen falls "stratum" ungleich 0 oder "leap" ungleich 3. • Kiss-Code nach [RFC5905] Figure 13 vom Server, falls "stratum" gleich 0 und "leap" gleich 3.	0
referenceTimestamp	Decimal (4 Byte Seconds, 4 Byte Fraction)	Zeitstempel des Clients in Sekunden/Sekundenbruchteilen, der angibt, wann die Systemzeit zuletzt gesetzt oder synchronisiert wurde.	0
originTimestamp	Decimal (4 Byte Seconds, 4 Byte Fraction)	Zeitstempel des Clients in Sekunden/Sekundenbruchteilen, der angibt, wann der NTP-Request zum Server erzeugt wurde.	0
receiveTimestamp	Decimal (4 Byte Seconds, 4 Byte Fraction)	Zeitstempel in Sekunden/Sekundenbruchteilen, der angibt, wann der NTP-Request beim Server empfangen wurde.	0
transmitTimestamp	Decimal (4 Byte Seconds, 4 Byte Fraction)	Zeitstempel in Sekunden/Sekundenbruchteilen, der angibt, wann die NTP-Response vom Server versandt wurde.	0 oder random

Tabelle 7.1 NTPMessage

Hinweise:

- Der "destinationTimestamp" wird als Empfangszeitstempel der NTP-Response beim Client gebildet.
- Extension-Felder sind nicht vorhanden.
- Da NTP durch TLS geschützt ist, werden die Datenfelder "dgst" und "Key identifier" nicht verwendet.
- Die in der Spalte "Datenminimierung" genannten Werte werden für die Aussendung von NTP-Client Requests empfohlen (siehe [RFC8633]). RFC5905-konforme NTP-Server akzeptieren diese NTP-Requests.

7.3 Technische Anwendungsfälle des NTP-Clients

Die folgenden Abschnitte enthalten die TA für die Implementierung eines NTP-Clients.

7.3.1 TA.NTP.BuildRequest

7.3.1.1 Beschreibung

In diesem TA werden die Anforderungen an die Implementierung des NTP-Clients zur Erzeugung eines NTP-Requests beschrieben, um eine Zeitsynchronisation anzufordern.

7.3.1.2 Anforderungen

REQ.TA.NTP.BuildRequest.10

Die Implementierung **MUSS** einen NTP-Request als NTPMessage der Version 4 für einen Client gemäß [RFC5905] bilden.

REQ.TA.NTP.BuildRequest.20

Die Implementierung **MUSS** den NTP-Request ohne UDP-Header erzeugen und senden. [2]

REQ.TA.NTP.BuildRequest.30

Das Implementierung **MUSS** den NTP-Request ohne NTP-Extensions senden. Der NTP-Request hat eine Länge von 48 Bytes. [3]

REQ.TA.NTP.BuildRequest.40

Die Implementierung **SOLL** die Werte der Datenfelder des NTP-Request aus Gründen der Datensparsamkeit gemäß Spalte "Datenminimierung" aus ▸Tabelle 7.1 setzen. (▸ICS.TA.NTP.BuildRequest.10).

ICS

ICS.TA.NTP.BuildRequest.10

Der Hersteller MUSS im ICS deklarieren, ob die Datenminimierung von ausgesandten NTP-Client-Requests umgesetzt wird.

7.3.2 TA.NTP.ProcessResponse

7.3.2.1 Beschreibung

In diesem TA werden die Anforderungen an die Implementierung zur Verarbeitung einer NTP-Response vom Zeitserver beschrieben, um mit der NTP-Response eine Zeitsynchronisation durchzuführen. Die Systemzeit (UTC), die Information über die letzte erfolgreiche Synchronisation, der maximale Fehler (Abweichung der Systemzeit von der Root-Zeit = rootdelay/2+rootdisp) und die geschätzte Genauigkeit (dispersion/jitter) werden der Fachanwendung bereitgestellt.

7.3.2.2 Implementierungshinweise

1. Da die höchste Poll-Frequenz der Zeitsynchronisation durch andere Vorgaben dieser Spezifikation auf einmal pro Minute limitiert wird, ist eine Verarbeitung des Kiss-Codes "RATE" nicht vorgesehen.
2. Am 8. Februar 2036 beginnt die NTP Ära "1". Hier ist Aufmerksamkeit bei der Konvertierung in die Systemzeit und umgekehrt notwendig.
3. Am 19. Januar 2038 um 3:14:08 Uhr UTC sind Implementierungen, die vorzeichenbehaftete 32bit UTC-Counter verwenden, von einem Überlauf betroffen.

[2] Hinweis: Diese Anforderung stellt eine Änderung zu [RFC5905] Kapitel 7.3 dar.

[3] Hinweis: Diese Anforderung schränkt [RFC5905] ein.

4. Falls Schaltsekunden-Ankündigungen bei der Konvertierung in die Systemzeit ausgewertet werden, sollte der Synchronisation mit der Systemzeit besondere Aufmerksamkeit geschenkt werden.
5. [RFC5905] Annex A.5.1 und A.5.1.1 geben Hinweise zur Plausibilisierung empfangener Nachrichten, bevor diese zur Zeitsynchronisation verwendet werden.

7.3.2.3 Anforderungen

REQ.TA.NTP.ProcessResponse.10

Die Implementierung **MUSS** Nachrichten ignorieren, die auf dem Transportweg unzulässig verzögert wurden, d.h. die Differenz aus Versandzeitpunkt (*ntpRequests* "originTimestamp") und Empfangszeitpunkt (*ntpResponse* "destinationTimestamp") sind größer als 9 Sekunden.

REQ.TA.NTP.ProcessResponse.30

Die Implementierung **MUSS** Nachrichten ignorieren, deren Zeitstempel "originTimestamp" nicht identisch mit dem zuletzt gesendeten Zeitstempel "transmitTimestamp" des NTP-Requests ist.

REQ.TA.NTP.ProcessResponse.40

Die Implementierung **MUSS** Nachrichten ignorieren, deren Zeitstempel des Datenfeldes "transmitTimestamp" nicht in dem vom Hersteller im ▸ICS.TA.NTP.ProcessResponse.10 deklarierten Zeitraum zur korrekten Konvertierung in die UTC-Systemzeit liegt.

REQ.TA.NTP.ProcessResponse.50

Die Implementierung **MUSS** Nachrichten mit Datenfeld "leap"=3 (Nicht synchronisierter NTP-Server) ignorieren.

REQ.TA.NTP.ProcessResponse.51

Die Implementierung **MUSS** Nachrichten mit Datenfeld "version" < "4" (Nicht RFC5905-konform) ignorieren .

REQ.TA.NTP.ProcessResponse.52

Die Implementierung **MUSS** Nachrichten mit Datenfeld "mode" ungleich "4" (Kein NTP-Server) ignorieren.

REQ.TA.NTP.ProcessResponse.110

Falls die Systemzeit im synchronisierten Zustand ist, **SOLL** die Implementierung Nachrichten ignorieren, bei denen die Differenz zwischen "SystemTime" und empfangenem "transmitTimestamp" größer als 1000s ist. [4]

ICS

ICS.TA.NTP.ProcessResponse.10

Der Hersteller deklariert im ICS, bis zu welchem Datum die Implementierung empfangene Zeitstempel korrekt in die Zeitskala UTC konvertieren kann, um damit die Systemuhr zu synchronisieren.

[4] Hinweis: Dadurch sollten im synchronisierten Zustand Angriffe durch starke Änderungen an der NTP-Zeit des Zeitservers keine Auswirkungen auf den Betrieb der Implementierung haben. Nachdem die Implementierung in einen unsynchronisierten Zustand übergeht, ist eine Neusynchronisierung mit dem Zeitserver möglich.

Literaturverzeichnis

[DIN43849] *DIN43849 (Bisher DIN 43863-5:2012-04) Messeinrichtungen und systeme, sowie Zusatzeinrichtungen und Steuergeräte – Herstellerübergreifende Identifikationsnummer.* 2023. VDE|DKE K461.

[DRAFT-IETF-AFT-SOCKS-SSL-00] *Secure Sockets Layer for SOCKS Version 5.* IETF.

[DS] *TR-03109-1 Detailspezifikationen.* Bundesamt für Sicherheit in der Informationstechnik. Aktuelle Fassung.

[RFC1035] *Domain names - implementation and specification.* IETF und P. Mockapetris. 1987.

[RFC1928] *SOCKS Protocol Version 5.* IETF.

[RFC3986] *Uniform Resource Identifier (URI): Generic Syntax.* IETF, Tim Berners-Lee, Roy T. Fielding und Larry Masinter. 2005.

[RFC5246] *The Transport Layer Security (TLS) Protocol Version 1.2.* IETF, T. Dierks und E. Rescorla. 2008.

[RFC5280] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* IETF, D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley und W. Polk. 2008.

[RFC5289] *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM).* IETF und E. Rescorla. 2008.

[RFC5480] *Elliptic Curve Cryptography Subject Public Key Information.* IETF, Turner, Brown, Yiu, Housley und Polk. 2009.

[RFC5905] *Network Time Protocol Version 4: Protocol and Algorithms Specification.* IETF, D. Mills, J. Martin, J. Burbank und W. Kasch. 2010.

[RFC6066] *Transport Layer Security (TLS) Extensions: Extension Definitions.* IETF und D. Eastlake. 2011.

[RFC6762] *Multicast DNS.* IETF. Februar 2013.

[RFC6763] *DNS-Based Service Discovery.* IETF. Februar 2013.

[RFC7027] *Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS).* IETF, J. Merkle, und M. Lochter. 2013.

[RFC7301] *Transport Layer Security (TLS) - Application-Layer Protocol Negotiation Extension.* IETF, S. Friedl, A. Langley und E. Stephan. 2014.

[RFC7366] *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).* IETF und P. Gutmann. 2014.

[RFC7627] *Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension.* IETF und K. Bhargavan. 2015.

[RFC8259] *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF und T. Bray. 2017.

[RFC8422] *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier.* IETF, Y. Nir, S. Josefsson und M. Pegourie-Gonnard. 2018.

[RFC8446] *The Transport Layer Security (TLS) Protocol Version 1.3.* IETF. August 2018.

[RFC8633] *Network Time Protocol Best Current Practices.* IETF. July 2019.

[RFC8734] *Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS) Version 1.3.* IETF. February 2020.

[RFC8882] *DNS-Based Service Discovery (DNS-SD) Privacy and Security Requirements.* IETF. September 2020.

[RFC9110] *HTTP Semantics.* IETF, R. Fielding, M. Nottingham und J. Reschke. 2022.

[RFC9112] *HTTP/1.1.* IETF, R. Fielding, M. Nottingham und J. Reschke. 2022.

[SM-PKI-CP] *SM-PKI-CP - Certificate Policy für die SM-PKI v1.1.1.* 2017. Bundesamt für Sicherheit in der Informationstechnik.

[TR-03109-3] *Technische Richtlinie BSI-TR-03109-3: Kryptographische Vorgaben für die Infrastruktur von intelligenten Messsystemen.* 2014. Bundesamt für Sicherheit in der Informationstechnik.

[TR-03109-4] *Technische Richtlinie BSI-TR-03109-4: Smart Metering PKI - Public Key Infrastruktur für Smart Meter Gateways.* 2014. Bundesamt für Sicherheit in der Informationstechnik.

[X.200] *Information technology – Open Systems Interconnection – Basic Reference Model: The basic model.* 07/1994. ITU.

Glossar

GWA_WAN_SIG_CRT Das Inhaltsdaten-Signatur-Zertifikat des GWA mit dem Zertifikatsprofil eines C_{Sign}(GWA) nach [TR-03109-4] Anhang A.

SubjectCN Das "commonName" Attribut des X.520 "distinguishedName" Attributes des "Subject"-Datenfeldes eines X.509-Zertifikates. Enthält den Namen des Zertifikatsinhabers.

GW_HAN_TLS_CRT Das TLS-Authentifizierungszertifikat des SMGW an der HAN-Schnittstelle basierend auf [RFC5280] Kapitel 4.

CLS_HAN_TLS_CRT Das TLS-Authentifizierungszertifikat eines CLS-Gerätes als HAN-Teilnehmer basierend auf [RFC5280] Kapitel 4

Anhang A Abkürzungsverzeichnis

Abkürzung	Beschreibung
aEMT	Aktiver Externer Marktteilnehmer
AFL	Authentication and Fragmentation Layer, Wireless MBUS
API	Application Programming Interface
APL	Application Protocol Layer, Wireless MBUS
ARP	Address Resolution Protocol
ASN	Abstract Syntax Notation
BER	Basic Encoding Rules (ASN.1)
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certification Authority
CLS	Controllable Local System
CMS	Cryptographic Message Syntax, Inhaltsdatensicherung nach ASN.1
CON	Consumer bzw. Anschlussnutzer
COSEM	COmpanion Specification for Energy Metering
DER	Distinguished Encoding Rules (ASN.1)
DS	Detailspezifikation
EMT	Externer Marktteilnehmer
EnWG	Energiewirtschaftsgesetz
GDEW	Gesetz zur Digitalisierung der Energiewende
GWA	Smart-Meter-Gateway-Administrator
GWH	Smart-Meter-Gateway-Hersteller
HAN	Home Area Network
HDLC	High Level Data Link Control
HKS	HAN-Kommunikationsszenario
HTTP	HyperText Transfer Protocol
IC	Interface Class (für COSEM)
ICS	Implementation Conformance Statement
IETF	Internet Engineering Task Force
IP	Internet Protocol
KS	Kommunikationsszenario
LKS	LMN-Kommunikationsszenario
LMN	Local Meter Network
wM-Bus	Wireless Meter Bus
MessEG	Mess- und Eichgesetz
MessEV	Mess- und Eichverordnung
MK	Master Key
MSB	Messstellenbetreiber
MsbG	Messstellenbetriebsgesetz
MTR	Messeinrichtung
N/A	Nicht anwendbar

Abkürzung	Beschreibung
NTP	Network Time Protocol
OBIS	OBject Identification System (für COSEM)
PSK	Pre-Shared Key, zuvor vereinbarter symmetrischer Schlüssel
PTB	Physikalisch-Technische Bundesanstalt
RFC	Request For Comments
RTT	Round Trip Time
SM	Sicherheitsmodul
SM-PKI	Smart-Meter - Public Key Infrastructure
SMGW	Smart-Meter-Gateway
SML	Smart Message Language
SNI	Server Name Indication
SRV	Servicetechniker des SMGW
TCP	Transmission Control Procotol
TLS	Transport Layer Security, Transportsicherungsprotokoll
TPL	Transport Protocol Layer, Wireless MBUS
TR	Technische Richtlinie
UDP	User Datagram Protocol
UTC	Coordinated Universal Time, Zeitskala
WAN	Wide Area Network
WKS	WAN-Kommunikationsszenario
XML	Extendable Markup Language

Tabelle A.1 In der TR verwendete Abkürzungen