

Elektronische Gesundheitskarte und Telematikinfrastruktur

Übergreifende Spezifikation Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur

Version:	2. 37 39.0 <u>CC</u>
Revision:	1129989 1131903
Stand:	24.10.2024 14.02.2025
Status:	<u>zur Abstimmung</u> freigegeben
Klassifizierung:	öffentlich <u>Entwurf</u>
Referenzierung:	gemSpec_Krypt

Dokumentinformationen

Änderungen zur Vorversion

Anpassungen des vorliegenden Dokumentes im Vergleich zur Vorversion können Sie der nachfolgenden Tabelle entnehmen.

Dokumentenhistorie

Version	Datum	Grund der Änderung, besondere Hinweise	Bearbeitung
2.21.0	31.01.2023	Einarbeitung ePA_Maintenance_21.5 und Konn_Maintenance_21.6	gematik
2.22.0	01.03.2023	Einarbeitung E-Rezept_Maintenance_21.3	gematik
2.23.0	20.09.2023	Einarbeitung gemSpec_Krypt_Maintenance_22.1 (neu: Kap. 2.5) Einarbeitung Änderungsliste E- Rezept_Maintenance 22.2	gematik
2.24.0	08.12.2023	Einarbeitung CI_Maintenance_22.5, Konn_Maintenance_22.5 und 22.6, Kap. 5.4 Typo raus, redaktionelle Anpassungen	gematik
2.25.0	14.02.2023	Einarbeitung VSDM++, Maintenance_23.1	gematik
2.26.0	09.03.2023	Einarbeitung Konn_Maintenance_23.0	gematik
2.27.0	14.04.2023	Einarbeitung Konn_Maintenance_23.1	gematik
2.28.0	09.06.2023	Einarbeitung VSDM_Maintenance_23.2 und CI_Maintenance_23.1	gematik
2.29.0	30.01.2024	Einarbeitung ePA für alle: Kap. "VAU-Protokoll für E-Rezept" überarbeitet, neues Kap. "VAU-Protokoll für ePA für alle" eingefügt redaktionelle Anpassungen (z. B. -* bei Afo- Referenzen ergänzt)	gematik
2.30.0	23.02.2024	Einarbeitung HSK_Maintenance_23.6	gematik
2.31.0	19.02.2024	Einarbeitung Änderungsliste Smartcards_23.3	gematik
2.32.0	28.03.2024	Einarbeitung ePA für alle Release 3.0.1	gematik

Version	Datum	Grund der Änderung, besondere Hinweise	Bearbeitung
2.33.0	21.05.2024	Einarbeitung C_11598 (Änderungsliste E-Rezept_FdV_Kassen-App)	gematik
	30.05.2024	Draft - ePA für alle - Release 3.0.2	gematik
2.34.0	02.07.2024	Anpassung Zuordnungen für E-Rezept_1_6_5 (aus gemF_eRp_ePA, E-Rezept_Maintenance_24_1), Anpassung Zuordnungen für gemF_eRp_DiGA (neuer Steckbrief gemSST_CS_eRp_KTR)	gematik
2.35.0	05.07.2024	Einarbeitung Konn_24.1	gematik
2.36.0	12.07.2024	ePA für alle - Release 3.0.2	gematik
2.37.0	24.10.2024	ePA für alle - Release 3.0.3	gematik
2.38.0	14.02.2025	initiale ZETA-Spezifikation	gematik
2.39.0 CC	14.02.2025	ePA für alle - Release 3.0.5	gematik

Inhaltsverzeichnis

1 Einführung	11
1.1 Zielsetzung und Einordnung des Dokuments	11
1.2 Zielgruppe	11
1.3 Geltungsbereich	12
1.4 Abgrenzung des Dokuments	12
1.5 Methodik	12
2 Einsatzszenarioübergreifende Algorithmen	13
2.1 Identitäten	13
2.1.1 X.509-Identitäten	13
2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen	15
2.1.1.2 Qualifizierte elektronische Signaturen	18
2.1.1.3 TLS-Authentifizierung	20
2.1.1.4 IPsec-Authentifizierung	21
2.1.1.5 Digitale Signaturen durch TI-Komponenten	21
2.1.1.6 Verschlüsselung	21
2.1.2 CV-Identitäten	21
2.1.2.1 CV-Zertifikate G2	21
2.1.2.2 CV-Certification Authority (CV-CA) Zertifikat G2	22
2.2 Zufallszahlengeneratoren	22
2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren)	23
2.4 Schlüsselerzeugung und Schlüsselbestätigung	23
2.4.1 Prüfung auf angreifbare (schwache) Schlüssel	25
2.4.2 ECC-Schlüssel in X.509-Zertifikaten	25
2.4.3 RSA-Schlüssel in X.509-Zertifikaten	26
2.5 Einmalpasswörter	27
3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien	28
3.1 Kryptographische Algorithmen für XML-Dokumente	29
3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen	30
3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen	32
3.1.3 Webservice Security Standard (WSS)	33
3.1.4 XML-Verschlüsselung – Symmetrisch	34
3.1.5 XML-Verschlüsselung – Hybrid	34
3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung	34
3.2.1 Card-to-Card-Authentisierung G2	34
3.2.2 Card-to-Server (C2S) Authentisierung und Trusted Channel G2	35
3.3 Netzwerkprotokolle	35
3.3.1 IPsec-Kontext	35
3.3.2 TLS-Verbindungen	37

73	3.3.3 DNSSEC-Kontext.....	46
74	3.4 Masterkey-Verfahren (informativ).....	46
75	3.5 Hybride Verschlüsselung binärer Daten.....	48
76	3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten	48
77	3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten	49
78	3.6 Symmetrische Verschlüsselung binärer Daten.....	49
79	3.7 Signatur binärer Inhaltsdaten (Dokumente)	50
80	3.8 Signaturen innerhalb von PDF/A-Dokumenten.....	51
81	3.9 Kartenpersonalisierung	52
82	3.10 Bildung der pseudonymisierten Versichertenidentität	52
83	3.11 Spezielle Anwendungen von Hashfunktionen	52
84	3.11.1 Hashfunktionen und OCSP (informativ)	53
85	3.12 kryptographische Vorgaben für die SAK des Konnektors.....	54
86	3.13 Migration im PKI-Bereich	55
87	3.14 Spezielle Anwendungen von kryptographischen Signaturen.....	55
88	3.15 ePA-spezifische Vorgaben	55
89	3.15.1 Verbindung zur VAU	55
90	3.15.2 ePA-Aktensysteminterne Schlüssel	56
91	3.15.3 ePA-spezifische TLS-Vorgaben	58
92	3.15.4 Zugriffscode-Erzeugung	59
93	3.16 E-Rezept-spezifische Vorgaben	62
94	3.17 KOM-LE-spezifische Vorgaben	63
95	3.18 HMAC-Sicherung der Prüfziffer VSDM.....	64
96	3.19 spezifische TLS-Vorgaben für VSDM.....	75
97	4 Umsetzungsprobleme mit der TR-03116-1.....	76
98	4.1 XMLDSig und PKCS1-v2.1	76
99	4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM	76
100	4.3 XML Signature Wrapping und XML Encryption Wrapping	77
101	4.4 Güte von Zufallszahlen	77
102	5 Migration 120-Bit-Sicherheitsniveau.....	78
103	5.1 PKI-Begriff Schlüsselgeneration.....	78
104	5.2 X.509-Root der TI.....	79
105	5.3 TSL-Dienst und ECDSA-basierte TSL allgemein.....	81
106	5.4 ECC-Unterstützung bei TLS.....	81
107	5.5 ECC-Unterstützung bei IPsec.....	83
108	5.6 ECDSA-Signaturen.....	85
109	5.6.1 ECDSA-Signaturen im XML-Format.....	85

110	5.6.2 ECDSA-Signaturen im CMS-Format	85
111	5.7 ECIES	86
112	5.7.1 ECIES und authentifizierte Broadcast-Encryption	90
113	5.7.2 ECIES und mobKT	91
114	5.8 ECC-Migration eHealth-KT	92
115	5.8.1 Interoperabilität zwischen eHealth-KT und Konnektor	93
116	5.9 ECC-Migration Konnektor	95
117	5.10 Verschiedene Produkttypen und ECC-Migration (informativ)	97
118	6 VAU-Protokoll für E-Rezept	98
119	6.1 Übersicht (informativ)	98
120	6.2 Definition	100
121	6.2.1 E-Rezept VAU-Identität	100
122	6.2.2 Client-seitige Prüfung der E-Rezept VAU-Identität	101
123	6.2.3 E-Rezept VAU-Request und Response	105
124	6.2.4 Zufallsquelle für Clients	110
125	7 VAU-Protokoll für ePA für alle	112
126	7.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung	117
127	7.2 Transport und Sicherung der Nutzdaten	128
128	7.3 OIDC-Authentisierung eines Clients (Nutzer)	132
129	7.4 Authentisierung des E-Rezept-FD als ePA-Client	135
130	7.5 Routing auf VAU-Instanzen	137
131	7.6 Fehlersignalisierung	139
132	7.7 Tracing in Nichtproduktivumgebungen	140
133	7.7.1 Zufallsquelle für Clients	141
134	8 Post-Quanten-Kryptographie (informativ)	156
135	9 Erläuterungen (informativ)	157
136	9.1 Prüfung auf angreifbare (schwache) Schlüssel	157
137	9.2 RSA-Schlüssel in X.509-Zertifikaten	157
138	10 Anhang — Verzeichnisse	161
139	10.1 Abkürzungen	161
140	10.2 Glossar	163
141	10.3 Abbildungsverzeichnis	163
142	10.4 Tabellenverzeichnis	163
143	10.5 Referenzierte Dokumente	165
144	10.5.1 Dokumente der gematik	165
145	10.5.2 Weitere Dokumente	166

1	Einführung	11
1.1	Zielsetzung und Einordnung des Dokuments	11
1.2	Zielgruppe	11
1.3	Geltungsbereich	12
1.4	Abgrenzung des Dokuments	12
1.5	Methodik	12
2	Einsatzszenarioübergreifende Algorithmen	13
2.1	Identitäten	13
2.1.1	X.509-Identitäten	13
2.1.1.1	Digitale nicht-qualifizierte elektronische Signaturen	15
2.1.1.2	Qualifizierte elektronische Signaturen	18
2.1.1.3	TLS-Authentifizierung	20
2.1.1.4	IPsec-Authentifizierung	21
2.1.1.5	Digitale Signaturen durch TI-Komponenten	21
2.1.1.6	Verschlüsselung	21
2.1.2	CV-Identitäten	21
2.1.2.1	CV-Zertifikate G2	21
2.1.2.2	CV-Certification-Authority (CV-CA) Zertifikat G2	22
2.2	Zufallszahlengeneratoren	22
2.3	Hilfestellung bei der Umsetzung (Zufallsgeneratoren)	23
2.4	Schlüsselerzeugung und Schlüsselbestätigung	23
2.4.1	Prüfung auf angreifbare (schwache) Schlüssel	25
2.4.2	ECC-Schlüssel in X.509-Zertifikaten	25
2.4.3	RSA-Schlüssel in X.509-Zertifikaten	26
2.5	Einmalpasswörter	27
3	Konkretisierung der Algorithmen für spezifische Einsatzszenarien	28
3.1	Kryptographische Algorithmen für XML-Dokumente	29
3.1.1	XML-Signaturen für nicht-qualifizierte Signaturen	30
3.1.2	XML-Signaturen für qualifizierte elektronische Signaturen	32
3.1.3	Webservice Security Standard (WSS)	33
3.1.4	XML-Verschlüsselung – Symmetrisch	34
3.1.5	XML-Verschlüsselung – Hybrid	34
3.2	Karten-verifizierbare Authentifizierung und Verschlüsselung	34
3.2.1	Card-to-Card-Authentisierung G2	34
3.2.2	Card-to-Server (C2S) Authentisierung und Trusted Channel G2	35
3.3	Netzwerkprotokolle	35
3.3.1	IPsec-Kontext	35
3.3.2	TLS-Verbindungen	37
3.3.3	DNSSEC-Kontext	46
3.4	Masterkey-Verfahren (informativ)	46
3.5	Hybride Verschlüsselung binärer Daten	48

188	3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten	48
189	3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten	49
190	3.6 Symmetrische Verschlüsselung binärer Daten.....	49
191	3.7 Signatur binärer Inhaltsdaten (Dokumente)	50
192	3.8 Signaturen innerhalb von PDF/A-Dokumenten.....	51
193	3.9 Kartenpersonalisierung	52
194	3.10 Bildung der pseudonymisierten Versichertenidentität	52
195	3.11 Spezielle Anwendungen von Hashfunktionen	52
196	3.11.1 Hashfunktionen und OCSP (informativ)	53
197	3.12 kryptographische Vorgaben für die SAK des Konnektors	54
198	3.13 Migration im PKI-Bereich	55
199	3.14 Spezielle Anwendungen von kryptographischen Signaturen.....	55
200	3.15 ePA-spezifische Vorgaben	55
201	3.15.1 Verbindung zur VAU	55
202	3.15.2 ePA-Aktensysteminterne Schlüssel	56
203	3.15.3 ePA-spezifische TLS-Vorgaben	58
204	3.15.4 Zugriffscode-Erzeugung	59
205	3.16 Anomalie-Erkennung	60
206	3.17 E-Rezept-spezifische Vorgaben	62
207	3.18 KOM-LE-spezifische Vorgaben	63
208	3.19 Kryptographisch gesicherte VSDM-Prüfziffer Version 1	64
209	3.20 Kryptographisch gesicherte VSDM-Prüfziffer Version 2	66
210	3.21 spezifische TLS-Vorgaben für VSDM	75
211	4 Umsetzungsprobleme mit der TR-03116-1.....	76
212	4.1 XMLDSig und PKCS1-v2.1	76
213	4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM	76
214	4.3 XML Signature Wrapping und XML Encryption Wrapping	77
215	4.4 Güte von Zufallszahlen	77
216	5 Migration 120-Bit-Sicherheitsniveau.....	78
217	5.1 PKI-Begriff Schlüsselgeneration	78
218	5.2 X.509-Root der TI.....	79
219	5.3 TSL-Dienst und ECDSA-basierte TSL allgemein	81
220	5.4 ECC-Unterstützung bei TLS	81
221	5.5 ECC-Unterstützung bei IPsec	83
222	5.6 ECDSA-Signaturen	85
223	5.6.1 ECDSA-Signaturen im XML-Format	85
224	5.6.2 ECDSA-Signaturen im CMS-Format	85

225	5.7 ECIES.....	86
226	5.7.1 ECIES und authentifizierte Broadcast-Encryption	90
227	5.7.2 ECIES und mobKT	91
228	5.8 ECC-Migration eHealth-KT	92
229	5.8.1 Interoperabilität zwischen eHealth-KT und Konnektor	93
230	5.9 ECC-Migration Konnektor.....	95
231	5.10 Verschiedene Produkttypen und ECC-Migration (informativ).....	97
232	6 VAU-Protokoll für E-Rezept.....	98
233	6.1 Übersicht (informativ)	98
234	6.2 Definition.....	100
235	6.2.1 E-Rezept-VAU-Identität	100
236	6.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität.....	101
237	6.2.3 E-Rezept-VAU-Request und -Response	105
238	6.2.4 Zufallsquelle für Clients.....	110
239	7 VAU-Protokoll für ePA für alle.....	112
240	7.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung	117
241	7.2 Transport und Sicherung der Nutzdaten	128
242	7.3 OIDC-Authentisierung eines Clients (Nutzer)	132
243	7.4 Authentisierung des E-Rezept-FD als ePA-Client	135
244	7.5 Routing auf VAU-Instanzen	137
245	7.6 Fehlersignalisierung	139
246	7.7 Tracing in Nichtproduktivumgebungen	140
247	7.7.1 Zufallsquelle für Clients.....	141
248	8 ZETA/ASL (VAU-Protokoll)	142
249	8.1 Verbindungsaufbau/Schlüsselaushandlung.....	142
250	8.2 Transport und Sicherung der Nutzdaten	149
251	8.3 Fehlersignalisierung	153
252	8.4 Tracing in Nichtproduktivumgebungen	155
253	9 Post-Quanten-Kryptographie (informativ)	156
254	10 Erläuterungen (informativ)	157
255	10.1 Prüfung auf angreifbare (schwache) Schlüssel.....	157
256	10.2 RSA-Schlüssel in X.509-Zertifikaten	157
257	11 Anhang – Verzeichnisse	161
258	11.1 Abkürzungen	161
259	11.2 Glossar	163

260	<u>11.3 Abbildungsverzeichnis.....</u>	163
261	<u>11.4 Tabellenverzeichnis.....</u>	163
262	<u>11.5 Referenzierte Dokumente.....</u>	165
263	<u>11.5.1 Dokumente der gematik.....</u>	165
264	<u>11.5.2 Weitere Dokumente.....</u>	166
265		

266

1 Einführung

267

1.1 Zielsetzung und Einordnung des Dokuments

268
269
270

Die vorliegende übergreifende Spezifikation definiert Anforderungen an Produkte der TI bezüglich kryptographischer Verfahren. Diese Anforderungen sind als übergreifende Regelungen relevant für Interoperabilität und Verfahrenssicherheit.

271
272
273
274
275
276
277
278
279
280
281

Für die TI ist die Technische Richtlinie 03116 Teil 1 [BSI-TR-03116-1] normativ, d. h. nur dort aufgeführte kryptographische Verfahren dürfen von Produkten in der TI verwendet werden. Wenn mehrere unterschiedliche Produkttypen der TI zusammenarbeiten ist es bez. der Interoperabilität nicht sinnvoll wenn jeder beteiligte Produkttyp alle dort aufgeführten Verfahren umsetzen muss, da er vermuten muss, die Gegenstelle beherrscht nur eine Teilmenge der dort aufgeführten Verfahren. Um einen gemeinsamen Nenner zu definieren, legt dieses Dokument für bestimmte Einsatzzwecke ein Mindestmaß an verpflichtend zu implementierenden Verfahren aus [BSI-TR-03116-1] fest, oftmals mit spezifischen Parametern. Ein Produkttyp ist frei, weitere Verfahren aus der [BSI-TR-03116-1] optional zu implementieren, kann sich jedoch nicht ohne Weiteres darauf verlassen, dass sein potentieller Kommunikationspartner diese auch beherrscht.

282
283
284
285
286
287
288
289
290
291

In Bezug auf die Formulierung der Ende-Daten der Zulässigkeit eines kryptographischen Verfahrens wird die Konvention aus der TR-02102- und der TR-03116-Familie verwendet, d. h., eine Aussage „Algorithmus X ist geeignet bis Ende 2029+“ bedeutet generell nicht, dass Algorithmus X nach Ende 2029 nicht mehr geeignet ist, sondern lediglich, dass über die Eignung nach Ende 2029 keine explizite Aussage gemacht wird und dass aus heutiger Sicht die weitere Eignung nicht ausgeschlossen ist. Aussagen über den Betrachtungszeitraum hinaus sind mit einem höheren Maß an Spekulation verbunden. Sollte bei den Angaben zum Ende der zeitlichen Zulässigkeit kein "+" aufgeführt sein (bspw. "Ende 2025") , so bedeutet dies, dass eine Verlängerung der Zulässigkeit über den aufgeführten Zeitpunkt hinaus nicht geplant ist.

292
293
294
295
296
297
298
299
300
301

Bei neuen Erkenntnissen über die verwendeten kryptographischen Algorithmen, die zu einer Änderung der TR-03116-1 führen, wird eine Anpassung dieses Dokumentes erfolgen. Für Verwendungszwecke, bei denen bereits eine Migration zu stärkeren Algorithmen in Planung ist oder die Verwendung von Algorithmen unterschiedlicher Stärke zulässig ist, wird ein Ausblick gegeben, bis wann welche Algorithmen ausgetauscht sein müssen. Bei den Migrationsstrategien für kryptographische Algorithmen ist darauf zu achten, dass hinterlegte Objekte umzuschlüsseln sind bzw. die älteren Algorithmen (unter der Bedingung, dass sie sicherheitstechnisch noch geeignet sind) für eine gewisse Übergangsphase weiter unterstützt werden müssen und danach zuverlässig in den Komponenten deaktiviert werden müssen.

302

1.2 Zielgruppe

303
304

Das Dokument richtet sich an Hersteller und Anbieter von Produkten der TI, die kryptographische Objekte verwalten.

305 **1.3 Geltungsbereich**

306 Dieses Dokument enthält normative Festlegungen zur Telematikinfrastruktur des
307 deutschen Gesundheitswesens. Der Gültigkeitszeitraum der vorliegenden Version und
308 deren Anwendung in Zulassungsverfahren wird durch die gematik GmbH in gesonderten
309 Dokumenten (z. B. gemPTV_ATV_Festlegungen, Produkttypsteckbrief,
310 Leistungsbeschreibung) festgelegt und bekannt gegeben.

311 **Schutzrechts-/Patentrechtshinweis**

312 *Die nachfolgende Spezifikation ist von der gematik allein unter technischen*
313 *Gesichtspunkten erstellt worden. Im Einzelfall kann nicht ausgeschlossen werden, dass*
314 *die Implementierung der Spezifikation in technische Schutzrechte Dritter eingreift. Es ist*
315 *allein Sache des Anbieters oder Herstellers, durch geeignete Maßnahmen dafür Sorge zu*
316 *tragen, dass von ihm aufgrund der Spezifikation angebotene Produkte und/oder*
317 *Leistungen nicht gegen Schutzrechte Dritter verstoßen und sich ggf. die erforderlichen*
318 *Erlaubnisse/Lizenzen von den betroffenen Schutzrechtsinhabern einzuholen. Die gematik*
319 *GmbH übernimmt insofern keinerlei Gewährleistungen.*

320 **1.4 Abgrenzung des Dokuments**

321 Aufgabe des Dokumentes ist es nicht, eine Sicherheitsbewertung von kryptographischen
322 Algorithmen vorzunehmen. Dieser Gesichtspunkt wird in [BSI-TR-03116-1] behandelt. Es
323 werden lediglich die dort vorgegebenen Algorithmen weiter eingeschränkt, um die
324 Herstellung der Interoperabilität zu unterstützen.

325 Es ist nicht Ziel dieses Dokumentes, den Prozess zum Austauschen von Algorithmen zu
326 definieren, sondern lediglich den zeitlichen Rahmen für die Verwendbarkeit von
327 Algorithmen festzulegen und somit auf den Bedarf für die Migration hinzuweisen.

328 **1.5 Methodik**

329 Anforderungen als Ausdruck normativer Festlegungen werden durch eine eindeutige ID
330 sowie die dem RFC 2119 [RFC-2119] entsprechenden, in Großbuchstaben geschriebenen
331 deutschen Schlüsselworte MUSS, DARF NICHT, SOLL, SOLL NICHT, KANN
332 gekennzeichnet.

333 Sie werden im Dokument wie folgt dargestellt:

334 **<AFO-ID> - <Titel der Afo>**

335 Text / Beschreibung

336 [**<=**]

337 Dabei umfasst die Anforderung sämtliche zwischen Afo-ID und der Textmarke [**<=**]
338 angeführten Inhalte.

2 Einsatzszenarioübergreifende Algorithmen

Nachfolgend werden grundlegende Festlegungen zur Verwendung von Algorithmen innerhalb der Telematikinfrastuktur getroffen. Diese Anforderungen sind unabhängig von den im nachfolgenden Kapitel definierten Einsatzszenarien und werden durch diese verwendet.

GS-A_3080 - asymmetrischen Schlüssel maximale Gültigkeitsdauer

Die Lebensdauer von asymmetrischen Schlüsseln und somit die in einem Zertifikat angegebene Gültigkeitsdauer SOLL maximal 5 Jahre betragen. [\leq]

2.1 Identitäten

Der Begriff „kryptographische Identität“ (nachfolgend nur noch als Identität bezeichnet) bezeichnet einen Verbund aus Identitätsdaten und einem kryptographischen Objekt, das bspw. im Rahmen einer Authentisierung und Authentifizierung verwendet werden kann. Im Allgemeinen handelt es sich um Schlüsselpaare, bestehend aus öffentlichem und privatem Schlüssel, sowie einem Zertifikat, das die Kombination aus Attributen und öffentlichem Schlüssel durch eine übergeordnete Instanz (CA – Certification Authority) bestätigt.

Bei den Algorithmenvorgaben für Identitäten muss u. a. spezifiziert werden:

- für welche Algorithmen und für welchen Verwendungszweck die Schlüssel verwendet werden (Bestimmte Verwendungszwecke schließen einander aus, bspw. dürfen nicht Signaturschlüssel für die Sicherung von Authentizität und Integrität von Dokumenten als Signaturschlüssel für beliebige Challenges im Rahmen einer Authentisierung verwendet werden.),
- welche Algorithmen für die Signatur des Zertifikates verwendet werden,
- mit welchen Algorithmen die OCSP-Responses signiert werden und
- wie die Zertifikate des OCSP-Responders signiert sind.

2.1.1 X.509-Identitäten

Eine X.509-Identität ist eine Identität gemäß Abschnitt 2.1, bei der ein X.509-Zertifikat [RFC-5280] verwendet wird.

Bei der Aufteilung von X.509-Identitäten wurden die Identitäten zunächst nach Gruppen für verschiedene Einsatzzwecke des Schlüssels unterteilt und diese bei Bedarf um einen notwendigen Einsatzkontext erweitert. Aus dieser Aufteilung ergibt sich die nachfolgend tabellarisch dargestellte Übersicht der Arten von X.509-Identitäten. Der exemplarische Einsatzort der Identitäten ist hierbei rein informativ, die Ausprägung wird in den Spezifikationen festgelegt, die eine kryptographische Identität benötigen.

373 **Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten**

Referenz	Gruppe	Kontext	Exemplarische Identitäten zur Verwendung (nicht vollständig)
2.1.1.1	Identitäten für die Erstellung von Signaturen	Identitäten für die Erstellung nicht-qualifizierter digitaler Signaturen	OSIG-Identität der SMC-B bzw. HSM-B
2.1.1.2		Identitäten für die Erstellung qualifizierter Signaturen	QES-Identität des HBA
2.1.1.5		Signaturidentitäten, die in den Diensten der TI-Plattform und den Fachdiensten zum Einsatz kommen.	Fachdienstsignatur Signatur durch zentrale Komponente der TI-Plattform Code-Signatur
2.1.1.3	Identitäten für die Client-Server-Authentifizierung	Identitäten für den Aufbau von TLS-Verbindungen	Fachdienst TLS – Server Fachdienst TLS – Client zentrale TI-Plattform TLS – Server zentrale TI-Plattform TLS – Client AUT-Identität der SMC-B AUT-Identität des Kartenterminals AUT-Identität des Anwendungskonnektors AUT-Identität der SAK AUT-Identität der eGK AUTN-Identität der eGK AUT-Identität des HBA
2.1.1.4		Identitäten für den Aufbau von IPsec-Verbindungen	ID.NK.VPN ID.VPNK.VPN
2.1.1.6	Verschlüsselungs-zertifikate	Identitäten, für die medizinische Daten verschlüsselt werden	ENC-Identität des Versicherten ENCV-Identität der eGK des Versicherten ENC-Identität des HBA ENC-Identität der SMC-B

374 Für den Aufbau der X.509-Zertifikate gelten die Vorgaben aus den jeweiligen
375 Spezifikationen der X.509-Zertifikate.

2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen

GS-A_4357-02 - X.509-Identitäten für die Erstellung und Prüfung digitaler nicht-qualifizierter elektronischer Signaturen

Alle Produkttypen, die X.509-Identitäten bei der Erstellung oder Prüfung digitaler nicht-qualifizierter elektronischer Signaturen verwenden, MÜSSEN die in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenvorgaben erfüllen.

Produkttypen, die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen.

[<=]

Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „RSA“

Anwendungsfall	Vorgaben
Art und Kodierung des öffentlichen Schlüssels	RSA (OID 1.2.840.113549.1.1.1) zu verwendende Schlüssellänge: 2048 Bit, zulässig bis Ende 2025, vgl. auch A_15590
Signatur eines Zertifikats Signatur einer OCSP-Response Signatur eines OCSP-Responder-Zertifikats Signatur einer CRL Signatur des Zertifikats das Basis der Signaturprüfung einer CRL ist	sha256withRSAEncryption (OID 1.2.840.113549.1.1.11) zu verwendende Schlüssellänge: 2048 Bit, zulässig bis Ende 2025, vgl. auch A_15590

A_15590 - Zertifikatslaufzeit bei Erstellung von X.509-Zertifikaten mit RSA 2048 Bit

Ein TSP-X.509-nonQES, der X.509-Zertifikate erstellt auf Basis der Schlüsselgeneration „RSA“ (d. h., für den die Vorgaben aus Tab_KRYPT_002 gelten), MUSS das Ende der Zertifikatsgültigkeitsdauer für das auszustellende Zertifikat unabhängig von der in Tab_KRYPT_002 festgelegten Endedaten der Zulässigkeit der verwendeten RSA-Schlüssellängen festlegen.[<=]

Erläuterung: Die technische Durchsetzung des Endes der Zulässigkeit von RSA mit weniger als 3000 Bit Schlüssellänge in X.509-Zertifikaten erfolgt durch die Herausnahme der entsprechenden RSA-basierten Sub-CA-Zertifikate aus der TSL zum Zeitpunkt des Ablaufens der Zulässigkeit (gemäß TIP1-A_2062). Ein TSP muss bez. der Zertifikatsgültigkeitsdauer der von ihm ausgegebenen Zertifikate das nach Spezifikationslage definierte Verhalten zeigen (i. A. Zertifikatsgültigkeitsdauer der ausgegebenen Zertifikate von 5 Jahren). Ein TSP kann auch mit dem Kartenherausgeber beliebige Gültigkeitsdauern unter 5 Jahren für die Laufzeit der vom TSP ausgegebenen Zertifikate vereinbaren.

A_23458 - Konnektor, Zulässigkeitszeiträume kryptographische Algorithmen

Der Konnektor SOLL NICHT die Zulässigkeitszeiträume kryptographischer Algorithmen technisch durchsetzen.[<=]

Erläuterung: Analog zu A_15590 für die TSP der TI gilt, dass die Unterbindung der Verwendung von RSA mit Schlüssellängen unter 3000 Bit durch die gematik erfolgt durch

410 die Herausnahme der entsprechenden RSA-basierten Sub-CA-Zertifikate aus der TSL zum
411 Zeitpunkt des Ablaufens der Zulässigkeit (gemäß TIP1-A_2062).

412 **Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-**
413 **qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“**

Anwendungsfall	Vorgabe
Art und Kodierung des öffentlichen Schlüssels	<p>ecPublicKey {OID 1.2.840.10045.2.1}</p> <p>Entweder auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] zulässig bis Ende 2029+</p> <p>oder auf der Kurve P-256 [FIPS-186-5] zulässig bis Ende 2029+</p> <p>Verständnishinweis: vgl. auch A_23139 bezüglich der Entweder-Oder-Beziehung</p> <p>Die Kodierung des öffentlichen Punkt erfolgt nach [RFC5480, Abschnitt 2], vgl. Beispiel in Abschnitt 5.2)</p> <p>Der privater Schlüssel muss zufällig und gleichverteilt aus $\{1, \dots, q-1\}$ gewählt werden. (q ist die Ordnung des Basispunkts und $\text{ceil}(\log_2 q)=256$).</p>
Signatur eines Zertifikats Signatur einer OCSP- Response Signatur eines OCSP- Responder-Zertifikates Signatur einer CRL Signatur des Zertifikats das Basis der Signaturprüfung einer CRL ist	<p>ecdsa-with-SHA256 [RFC-3279] {OID 1.2.840.10045.4.3.2}</p> <p>Entweder auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] zulässig bis Ende 2029+</p> <p>oder auf der Kurve P-256 [FIPS-186-5] zulässig bis Ende 2029+</p> <p>vgl. Beispiel in Abschnitt 5.2</p> <p>Der privater Schlüssel muss zufällig und gleichverteilt aus $\{1, \dots, q-1\}$ gewählt werden. (q ist die Ordnung des Basispunkts und $\text{ceil}(\log_2 q)=256$).</p>

414 Aktuell werden in der TI CRLs ausschließlich im Rahmen des IPsec-Verbindungsaufbaus
415 (Verbindung der Konnektoren in die TI) verwendet.

416 Für die maximale Gültigkeitsdauer der Zertifikate gilt die Anforderung [GS-A_3080].

417 **A_22220-01 - Konnektor: zulässige Algorithmen und Domainparameter bei**
418 **Zertifikatsprüfungen**

419 Ein Konnektor KANN bei einer Zertifikatsprüfung alle im SOGIS-Katalog [SOG-IS] als
420 zulässig aufgeführten kryptographischen Signaturverfahren inkl. der dem jeweiligen
421 Verfahren zugehörigen Domainparametern (Mindestschlüssellängen, Kurvenparameter

etc.) für eine Zertifikatsprüfung verwenden, sofern die Angaben aus
[gemSpec_Krypt#Tab_KRYPT_002 und _002a (und auch _003 und _003a)] als
Mindestvorgaben (Mindestschlüssellängen, Mindestgrößen der Kurvenparameter etc.)
eingehalten werden. [\leq]

A_19073 - Feste Laufzeit CV-Zertifikate einer Karte (eGK/HBA/SMC-B)

Die Anbieter CVC-TSP eGK, Anbieter HBA und Anbieter SMC-B MÜSSEN CV-Zertifikate
tagesgenau in der Laufzeit auf die am kürzest gültigen X.509-Zertifikate der
"Schlüsselgeneration ECDSA" der Karte beschränken.
Sind keine X.509-Zertifikate der "Schlüsselgeneration ECDSA" auf der Karte vorgesehen,
dann MUSS die Laufzeit auf die am kürzest gültigen X.509-Zertifikate der
"Schlüsselgeneration RSA" der Karte beschränkt werden. [\leq]

A_19173 - Feste Laufzeit X.509-Zertifikate einer Karte (eGK/HBA/SMC-B)

Der Anbieter HBA, Anbieter SMC-B und der Anbieter X.509 TSP eGK MÜSSEN alle X.509-
Zertifikate der "Schlüsselgeneration ECDSA" der Karte tagesgenau in der Laufzeit auf die
der am längsten gültigen CV-Zertifikate der Karte beschränken. Sind keine X.509-
Zertifikate der "Schlüsselgeneration ECDSA" auf der Karte vorgesehen, dann MUSS die
Laufzeit aller X.509-Zertifikate der "Schlüsselgeneration RSA" der Karte tagesgenau in
der Laufzeit auf die der am längsten gültigen CV-Zertifikate der Karte beschränkt
werden. [\leq]

Hinweis: "Tagesgenau" bedeutet, dass der Zeitpunkt sich nicht im Kalenderdatum,
jedoch in der Uhrzeit unterscheiden darf.

A_23139 - TSP-X.509-nonQES: ECC-Kurvenparameter, Komplexitätsreduktion

Ein TSP-X.509-nonQES, der nicht die X.509-Root-CA der TI ist, MUSS sicherstellen, dass

1. ein öffentlicher ECC-Schlüssel im CA-Zertifikat,
2. die öffentlichen ECC-Schlüssel der zum CA-Zertifikat aus (1) zugehörigen OCSP-
Zertifikate (vgl. [RFC-6960#4.2.2.2] bzw. A_23142), und
3. die öffentlichen ECC-EE-Schlüssel in den EE-Zertifikate, die durch die CA mit dem
Schlüssel aus (1) prüfbar sind,

die gleichen Kurvenparameter (brainpoolP256r1, P-256 etc. vgl.
[gemSpec_Krypt#Tab_KRYPT_002a]) besitzen.
[\leq]

Verständnishinweis:

Die Chipkarten der TI verwenden für ihre ECC-EE-Schlüssel alle die Kurvenparameter
brainpoolP256r1. Dies ist in den Objektsystem-Spezifikationen (und damit auch den
Objektsystemen) der Chipkarten fixiert. Die CA-en, die EE-Zertifikate für diese
Chipkarten bestätigen, müssen nach A_23139-* ebenfalls ein ECC-Schlüsselpaar auf
Basis von brainpoolP256r1 verwenden.

Die Komponenten-PKI der TI besitzt mehrere CA-Zertifikate. Es gibt mindestens ein CA-
Zertifikat, das für die Prüfung der ECC-EE-Zertifikate von SMC-K, SMC-KT und der
meisten Fachdienste verwendet wird. Dieses CA-Zertifikat verwendet ebenfalls als
öffentlichen Prüfschlüssel ein Schlüssel auf brainpoolP256r1-Basis (A_23139-*).

Für bestimmte Fachdienste, die zukünftig direkt von einem Primärsystem per TLS
erreichbar sein sollen, sollen TLS-Zertifikate in der Komponenten-PKI der TI erzeugt
werden können, die anstatt brainpool-Kurvenpunkte (brainpoolP256r1) NIST-

468 Kurvenpunkte (P-256) als öffentliche Schlüssel enthalten. Grund dafür ist die deutlich
469 bessere Unterstützung der NIST-Kurvenparameter durch verschiedene Standard-
470 Kryptographie-Softwarebibliotheken.

471 Es gibt in der Komponenten-PKI mindestens ein CA-Zertifikat, dessen öffentlicher ECC-
472 Schlüssel NIST-kurvenbasiert ist. Falls ein Fachdienst einen CSR mit einen NIST-
473 Kurvenpunkt als öffentlichen Schlüssel einreicht bei der Komponenten-PKI, dann wird
474 dieser unter der CA bestätigt, die NIST-kurvenbasiert ist.

475 In der Regel werden X.509-Root-CA-Zertifikate (RCA7 etc.) NIST-kurvenbasiert sein.
476 Damit kann ein PVS mit einer Kryptographie-Softwarebibliothek ohne brainpool-
477 Kurvenunterstützung mit solch einem Root-CA-Zertifikat die komplette Zertifikatskette
478 bis zum Fachdienst prüfen.

479 Für die X.509-Root-CA gilt A_23139-* absichtlich nicht.

480 **2.1.1.2 Qualifizierte elektronische Signaturen**

481 **GS-A_4358-01 - X.509-Identitäten für die Erstellung und Prüfung qualifizierter** 482 **elektronischer Signaturen**

483 Alle Produkttypen, die X.509-Identitäten für die Erstellung oder Prüfung von qualifizierten
484 elektronischen Signaturen verwenden, MÜSSEN mindestens alle in Tabelle
485 Tab_KRYPT_003 aufgeführten Algorithmen unterstützen und die Tabellenvorgaben
486 erfüllen.

487 TSP-X.509-QES, die qualifizierte Zertifikate (X.509-Identitäten) auf Basis der
488 Schlüsselgeneration „ECDSA“ (vgl. Abschnitt 5.1) erstellen oder verwenden MÜSSEN die
489 in Tab_KRYPT_003a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [<=]

490

491 **Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung** 492 **qualifizierter elektronischer Signaturen für die Schlüsselgeneration „RSA“**

Anwendungsfälle	Vorgaben
Signatur des VDA-Zertifikats	Nachdem die eIDAS-Verordnung das Signaturgesetz vollständig abgelöst hat, steht es einem VDA frei zu entscheiden welche Signatur (bspw. signiert von einer beliebigen VDA-internen CA) sein VDA-Zertifikat haben soll. Insbesondere kann die Signatur mit einem Nicht-RSA-Verfahren erstellt werden. Eine auswertende Komponente muss mit beliebigen (also auch nicht-RSA basierten) Signaturen eines VDA-Zertifikats umgehen können (bspw. Signatur des VDA-Zertifikats nicht auswerten, Authentizität und Integrität des Zertifikats wird über die Vertrauensliste sichergestellt).

Anwendungsfälle	Vorgaben
Art und Kodierung des öffentlichen EE-Schlüssels	<p><u>RSA-Signaturvariante:</u> Entweder OID 1.2.840.113549.1.1.1 (rsaEncryption) (zulässig bis gemäß [SOG-IS]) oder OID 1.2.840.113549.1.1.10 (id-RSASSA-PSS) [RFC-5756]. (zulässig bis gemäß [SOG-IS])</p> <p>Die Auswahl obliegt dem EE-Zertifikatsausgebenden VDA.</p> <p><u>RSA-Schlüssellänge:</u> zu verwendende Schlüssellänge: 2048 Bit, zulässig bis vgl. Angabe in [SOG-IS]</p>
Signatur eines Zertifikats, Signatur einer OCSP-Response oder Signatur eines OCSP-Responder-Zertifikates	<p>Entweder sha256withRSAEncryption (OID 1.2.840.113549.1.1.11) (zulässig bis gemäß [SOG-IS]) oder id-RSASSA-PSS (1.2.840.113549.1.1.10) [RFC-5756] (zulässig bis gemäß [SOG-IS])</p> <p>zu verwendende Schlüssellänge: 2048 Bit, zulässig bis vgl. Angabe in [SOG-IS]</p> <p>Die Hashfunktion für die Hashwertberechnung der TBSCertificate-Datenstruktur MUSS eine nach [SOG-IS] zulässige Hashfunktion („Agreed Hash Function“) sein. Als Hashfunktion SOLL SHA-256 [FIPS-180-4] verwendet werden. Als MGF MUSS MGF1 [PKCS#1] verwendet werden. Die innerhalb der MGF1 verwendete Hashfunktion MUSS die gleiche Hashfunktion sein, wie die Hashfunktion der Hashwertberechnung der TBSCertificate-Datenstruktur. (Dies entspricht der Empfehlung aus [RFC-5756] bzw. [RFC-4055, 3.1] und dient der Komplexitätsreduktion.) Die Saltlänge MUSS mindestens 256 Bit betragen. (Die Maximallänge des Salts ergibt sich nach [PKCS#1] in Abhängigkeit von der Länge des Moduls.)</p>

494 **Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung**
495 **qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“**

Anwendungsfall	Vorgabe
Signatur des VDA-Zertifikats	Nachdem die eIDAS-Verordnung das Signaturgesetz vollständig abgelöst hat, steht es einem VDA frei zu entscheiden welche Signatur (bspw. signiert von einer beliebigen VDA-internen CA) sein VDA-Zertifikat haben soll. Insbesondere kann die Signatur mit einem Nicht-ECDSA-Verfahren erstellt werden. Eine auswertende Komponente muss mit beliebigen (also auch nicht-ECDSA basierten) Signaturen eines VDA-Zertifikats umgehen können (bspw. Signatur des VDA-Zertifikats nicht auswerten, Authentizität und Integrität des Zertifikats wird über die Vertrauensliste sichergestellt).
Art und Kodierung des öffentlichen EE-Schlüssels	ecPublicKey {OID 1.2.840.10045.2.1} auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] (zulässig bis gemäß [SOG-IS]) Die Kodierung des öffentlichen Punkt erfolgt nach [RFC5480, Abschnitt 2], vgl. Beispiel in Abschnitt 5.2). Der private Schlüssel muss zufällig und gleichverteilt aus $\{1, \dots, q-1\}$ gewählt werden. (q ist die Ordnung des Basispunkts und $\text{ceil}(\log_2 q)=256$).
Signatur eines Zertifikats, Signatur einer OCSP-Response oder Signatur eines OCSP-Responder-Zertifikates	ecdsa-with-SHA256 [RFC-3279] {OID 1.2.840.10045.4.3.2} auf Kurve der brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] (zulässig bis gemäß [SOG-IS]) vgl. Beispiel in Abschnitt 5.2

2.1.1.3 TLS-Authentifizierung

GS-A_4359-02 - X.509-Identitäten für die Durchführung einer TLS-Authentifizierung

Alle Produkttypen, die X.509-Identitäten für eine TLS-Authentifizierung verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.

Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [\leq]

A_22457 - TLS-Clients, Ciphersuiten bei TLS-Verbindung mit eHealth-KT

Alle Produkttypen, die als TLS-Client gegenüber dem eHealth-Kartenterminal agieren, DÜRFEN bei beidseitig authentisierten TLS-Verbindungen NICHT Ciphersuiten mit Authentisierungsalgorithmen (RSA bzw. ECDSA) anbieten, wenn sie nicht auch für die Clientauthentisierung Schlüsselmaterial und Zertifikat für diese Authentisierungsalgorithmen besitzen. [\leq]

2.1.1.4 IPsec-Authentifizierung

GS-A_4360-01 - X.509-Identitäten für die Durchführung der IPsec-Authentifizierung

Alle Produkttypen, die X.509-Identitäten für eine IPsec-Authentifizierung verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [\leq]

2.1.1.5 Digitale Signaturen durch TI-Komponenten

GS-A_4361-02 - X.509-Identitäten für die Erstellung und Prüfung digitaler Signaturen

Alle Produkttypen, die X.509-Identitäten verwenden, die zur Erstellung und Prüfung digitaler Signaturen in Bezug auf TI-Komponenten (technische X.509-Zertifikate) genutzt werden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [\leq]

2.1.1.6 Verschlüsselung

GS-A_4362-02 - X.509-Identitäten für Verschlüsselungszertifikate

Alle Produkttypen, die X.509-Identitäten für die Verschlüsselung (Verschlüsselungszertifikate) verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [\leq]

2.1.2 CV-Identitäten

CV-Identitäten werden für die Authentifizierung zwischen Karten verwendet.

2.1.2.1 CV-Zertifikate G2

GS-A_4365-02 - CV-Zertifikate G2

Alle Produkttypen, die CV-Zertifikate der Kartengeneration G2 erstellen oder prüfen, MÜSSEN die in Tab_KRYPT_006 aufgeführten Algorithmen verwenden und die Tabellenanforderungen erfüllen.
[\leq]

546 **Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate**

Algorithmen Typ	Algorithmus	Schlüssellänge
über das Zertifikat bestätigtes Schlüsselpaar	Authentisierung ohne Sessionkey-Aushandlung [RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2} Authentisierung mit Sessionkey-Aushandlung [RFC-5639#3.4, brainpoolP256r1] authS_gemSpec-COS-G2_ecc-with- sha256 {OID 1.3.36.3.5.3.1}	256 Bit bis Ende 2029+
Signatur des Endnutzerzertifikats	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2029+

547 Für die maximale Gültigkeitsdauer der Zertifikate gilt die Anforderung [GS-A_3080].

548 **2.1.2.2 CV-Certification-Authority (CV-CA) Zertifikat G2**

549 **GS-A_4366-02 - CV-CA-Zertifikate G2**

550 Alle Produkttypen, die CV-CA-Zertifikate der Kartengeneration G2 erstellen oder prüfen,
551 MÜSSEN die Tab_KRYPT_007 aufgeführten Algorithmen verwenden und die
552 Tabellenanforderungen erfüllen.

553 [\leq]

554 **Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate**

Algorithmen Typ	Algorithmus	Schlüssellänge
über das Zertifikat bestätigtes Schlüsselpaar	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2029+
Signatur des CA- Zertifikates	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2029+

555 Für die maximale Gültigkeitsdauer der Zertifikate gilt die Anforderung [GS-A_3080].

556 **2.2 Zufallszahlengeneratoren**

557 **GS-A_4367 - Zufallszahlengenerator**

558 Alle Produkttypen, die Zufallszahlen generieren, MÜSSEN die Anforderungen aus [BSI-
559 TR-03116-1#3.8 Erzeugung von Zufallszahlen] erfüllen.

560 [\leq]

2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren)

(Hinweis: dies ist das ehemalige „Kapitel 5.2.4 Hilfestellung bei der Umsetzung der Anforderungen“. Der Text in diesem Abschnitt entstand in enger Abstimmung mit dem BSI auf Gesellschafterwunsch.)

Die Sicherheit eines deterministischen Zufallszahlengenerators (DRNGs) hängt maßgeblich von drei Faktoren ab:

- von der Entropie des Seeds,
- vom algorithmischen Anteil (generelles Design) und
- dem Schutz des inneren Zustands (und der zur Ausgabe vorgesehenen Zufallszahlen).

Der Nachweis, dass der algorithmische Anteil eines DRNGs den Anforderungen einer bestimmten Funktionalitätsklasse genügt, kann schwierig und aufwändig sein. Deshalb wurde das BSI gebeten, die DRNGs in [FIPS-186-2+CN1] und [ANSI-X9.31] in Bezug auf die kryptographische Güte ihres algorithmischen Anteils zu bewerten.

Das Ergebnis ist:

A) [FIPS-186-2+CN1]: Lässt man in dem DRNG aus Appendix 3.1 (S. 16f.) in Schritt 3c bzw. in dem DRNG aus Algorithmus 1 (Change Notice 1, S. 72f.) in Schritt 3.3 den Term "mod q" weg, so werden gleich verteilt 160-Bit Zufallszahlen bzw. 320-Bit Zufallszahlen erzeugt (vgl. Abschnitt „General Purpose Random Number Generation“ (Change Notice 1, S. 74)).

Beide DRNGs sind dann

1. algorithmisch geeignet für die Klasse K4 [AIS-20-1999] und
2. erfüllen die algorithmischen Anforderungen aus DRG.3 [AIS-20].

Ob eine konkrete Implementierung eines dieser DRNG bspw. Teil der Klasse DRG.3 ist, bleibt im Einzelfall zu prüfen, da dazu u. a. auch Fragen über die Initialisierung zu beantworten sind (vgl. (DRG.3.1) [KS-2011]).

Das BSI empfiehlt bei den Zufallsgeneratoren aus [FIPS-186-2+CN1] nach Möglichkeit SHA-256 [FIPS-180-4] anstatt SHA-1 zu verwenden. Folgt man der Empfehlung, so ist der Algorithmus dementsprechend zu adaptieren.

B) [ANSI-X9.31]: Der Zufallsgenerator aus Appendix A.2.4 ist

- (1) algorithmisch geeignet für die Klasse K3 [AIS-20-1999] und
- (2) erfüllt die algorithmischen Anforderungen aus DRG.2 [AIS-20].

2.4 Schlüsselerzeugung und Schlüsselbestätigung

GS-A_4368 - Schlüsselerzeugung

Alle Produkttypen, die Schlüssel erzeugen, MÜSSEN die Anforderungen aus [BSI-TR-03116-1#3.9 Schlüsselerzeugung] erfüllen. [≤=]

Hinweis: im Rahmen der Sicherheitszertifizierung von Komponenten, wie bspw. des Konnektors, wird dies überprüft.

GS-A_5021 - Schlüsselerzeugung bei einer Schlüsselspeicherpersonalisierung

Ein Herausgeber von Sicherheitsmodulen für kryptographisches Schlüsselmaterial, welche in der TI genutzt werden (also bspw. eGK, SMC-B, HSM-B, SMC-KT und HBA), MUSS sicherstellen, dass auf dem Sicherheitsmodul gespeicherten Schlüssel die Anforderungen aus [BSI-TR-03116-1#3.5 Schlüsselerzeugung] erfüllen.
[<=]

Hinweis: Dies ist eine Anforderung an Kartenherausgeber, die so sicherstellen müssen, dass das in den Sicherheitsmodulen (also auch HSM-B) zur Verfügung stehende kryptographische Schlüsselmaterial geeignet ist Daten mit sehr hohem Schutzbedarf schützen zu können. (siehe auch Kapitel 4.4)

GS-A_5338 - HBA/SMC-B – Erzeugung asymmetrischer Schlüsselpaare auf der jeweiligen Karte selbst

Ein Kartenherausgeber oder, falls der Kartenherausgeber einen Dritten mit der Kartenpersonalisierung beauftragt, der Kartenpersonalisierer für HBA oder SMC-B MUSS sicherstellen, dass bei der Personalisierung der Karten HBA und SMC-B alle asymmetrischen Schlüsselpaare, bei denen die privaten Schlüssel auf der Karte gespeichert werden, auf der Karte erzeugt werden.

[<=]

Aufgrund des geringeren Mengengerüsts bei HBA und SMC-B ist dort die On-Card-Generierung der entsprechenden Schlüsselpaare möglich. Somit (vgl. auch [PP-0082, FPT_EMS.1]) ist technisch sichergestellt, dass keine Kopie der privaten Schlüssel außerhalb der Chipkarte existiert (Kontext: Ende-zu-Ende-Verschlüsselung von medizinischen Daten).

GS-A_5386 - kartenindividuelle geheime und private Schlüssel G2-Karten

Ein Kartenherausgeber, der G2-Karten herausgibt, MUSS sicherstellen, dass bei der Personalisierung der Karten alle für eine Karte zu personalisierenden privaten und geheimen Schlüssel kartenindividuell sind. Bei Beauftragung eines Dritten mit der Schlüsselerzeugung ist dies durch den Dritten sicherzustellen.

Falls symmetrische Schlüssel (bspw. SK.CMS.AES128) nicht pro Karte zufällig erzeugt werden, sondern mit einem Schlüsselableitungsverfahren erzeugt werden, so MUSS der Kartenherausgeber sicherstellen, dass

1. das verwendete Schlüsselableitungsverfahren (KDF) unumkehrbar und nicht-vorhersagbar ist (Hilfestellung: Beispiele in [gemSpec_Krypt, 2.4 und 3.4]).
2. der Masterkey (Key Derivation Key (KDK)) GS-A_4368 erfüllt (insbesondere Entropie-Vorgaben). Der KDK MUSS eine Mindestentropie von 120 Bit besitzen.

[<=]

Für private Schlüssel bei HBA und SMC-B wird die kartenindividuelle Erzeugung und Personalisierung durch GS-A_5338 technisch sichergestellt. Je nach verwendetem COS, insbesondere dessen spezifischen Personalisierungsverfahrens, kann es sein, dass ein Kartenherausgeber symmetrische Schlüssel aus technischen Gründen personalisieren muss, obwohl er später nicht plant mit diesen Schlüsseln bspw. im Rahmen eines CMS zu arbeiten. Es ist sicherheitskritisch, dass auch diese symmetrischen Schlüssel ebenfalls die Anforderungen GS-A_5021 bzw. GS-A_4368 erfüllen.

Als geeignete Schlüsselableitungsverfahren (KDF) für die Erzeugung von kartenindividuellen Schlüssel sind bspw. folgende Verfahren geeignet:

- alle Verfahren aus [NIST-SP-800-108] mittels CMAC [NIST-SP-800-38B],
- alle Verfahren aus [NIST-SP-800-56-A] bzw. [NIST-SP-800-56-B] mittels jeder nach [BSI-TR-03116-1] zulässigen Hashfunktion,

- alle Verfahren aus [NIST-SP-800-56C] mittels CMAC [NIST-SP-800-38B] oder eines HMAC, der auf einer nach [BSI-TR-03116-1] zulässigen Hashfunktion basiert,
- das Verfahren nach [ANSI-X9.63, Abschnitt 5.6.3] mittels jeder nach [BSI-TR-03116-1] zulässigen Hashfunktion.

A_23900 - TSP-X.509: Einzigartigkeit der bestätigten öffentlichen Schlüssel

Ein TSP-X.509 nonQES SMC-B MUSS TSP-intern prüfen, ob EE-Schlüssel in den vom TSP auszustellenden Zertifikaten pro Identität einzigartig sind. Der TSP MUSS TSP-intern sicherstellen, dass zwei unterschiedliche End-Entitäten nicht das selbe Schlüsselpaar verwenden.

Der TSP MUSS von ihm in Zertifikaten bestätigte öffentliche EE-Schlüssel in einem Schlüsselspeicher ablegen. Dieser Schlüsselspeicher ist initial leer. Vor Ausstellen eines neuen Zertifikats MUSS der TSP prüfen, ob der öffentliche EE-Schlüssel schon im Schlüsselspeicher enthalten ist. Falls ja, so MUSS der TSP die Zertifikatserstellung ablehnen. Falls nein, MUSS der TSP nach erfolgreicher Zertifikatserstellung den nun im Zertifikat bestätigten öffentlichen EE-Schlüssel im Schlüsselspeicher einfügen. [<=]

Erläuterung zu A_23900:

Die Einzigartigkeit der öffentlichen EE-Schlüssel ist nach A_23900 nur TSP-intern sicherzustellen, es wird also nicht verlangt, dass die Schlüsselspeicher i. S. v. A_23900 über alle TSP ausgetauscht oder synchronisiert werden. A_23900 stellt eine Detaillierung von [gemRL_TSL_SP_CP#GS-A_4906] dar.

2.4.1 Prüfung auf angreifbare (schwache) Schlüssel

A_17294 - TSP-X.509: Prüfung auf angreifbare (schwache) Schlüssel

Ein TSP-X.509-nonQES MUSS vor einer Zertifikatserzeugung den durch das Zertifikat zu bestätigenden öffentlichen Schlüssel auf dessen kryptographische Angreifbarkeit hin prüfen.

Falls die Prüfung des öffentlichen Schlüssels das Ergebnis „angreifbar“ liefert, so MUSS der TSP die Zertifikatserstellung für diesen Schlüssel ablehnen.

Mindestumfang der Prüfung MÜSSEN

1. der Test auf die "Debian-OpenSSL-PRNG-Schwachstelle" und
2. der Test auf die Anfälligkeit gegen den ROCA-Angriff sein.

Der TSP MUSS den Mindestumfang der Prüfung bei Bekanntwerden neuer Angriffsmöglichkeiten gemäß [gemSpec_DS_Anbieter#GS-A_5560] erweitern. [<=]

TSPs, die im Internet TLS-Zertifikate ausgeben (bspw. für die Verwendung von HTTPS), müssen aufgrund der Baseline Requirement des CA/Browser Forums (<https://cabforum.org/baseline-requirements-documents/>) vor der Zertifikatserzeugung kryptographische Prüfungen des zu bestätigenden öffentlichen Schlüssels durchführen. Analog gilt dies mit A_17294 auch für TI-TSPs. Die gematik stellt auf Anfrage eine Beispielimplementierung für die Tests des Mindestumfangs bereit.

Unter <https://security.googleblog.com/2022/08/announcing-open-sourcing-of-paranoids.html> ist eine Vielzahl von Schlüsseltests als OpenSource verfügbar.

2.4.2 ECC-Schlüssel in X.509-Zertifikaten

GS-A_5518 - Prüfung Kurvenpunkte bei einer Zertifikatserstellung

690 Alle Produkttypen, die X.509-Zertifikate erstellen und dabei öffentliche Punkte auf einer
691 elliptischen Kurve in diesen Zertifikaten bestätigen, MÜSSEN überprüfen, ob die zu
692 bestätigenden Punkte auch auf der zugehörigen Kurve (im Regelfall brainpoolP256r1
693 [RFC-5639#3.4]) liegen. Falls nein, MUSS der Produkttyp eine Zertifikatsausstellung
694 verweigern.
695 [\leq]

696 **A_17091 - ECC-Schlüsselkodierung**

697 Ein TSP-X.509-nonQES MUSS sicherstellen, dass wenn er ECC-Schlüssel für eine
698 Zertifikatserstellung erhält, diese in unkomprimierter Form (d. h. explizite Aufführung der
699 vollständigen x- und y-Koordinaten [BSI-TR-03111#Abschnitt 3.2.1 "Uncompressed
700 Encoding"]) vom Antragsteller übergeben werden.
701 [\leq]

702 Hinweis: Diese Kodierungsform (uncompressed encoding) ist auch die Form, wie sie
703 letztendlich in den X.509-Zertifikaten verwendet wird. Weiterhin kann ein TSP in dieser
704 Form mit der Prüfung aus GS-A_5518 sicherstellen, dass keine Fehlkodierung des zu
705 bestätigenden ECC-Schlüssels aufgetreten ist.

706 **2.4.3 RSA-Schlüssel in X.509-Zertifikaten**

707 **A_17092 - RSA-Schlüssel Zertifikatserstellung, keine kleinen Primteiler und e** 708 **ist prim**

709 Ein TSP KANN im Rahmen der Zertifikatsbeantragung, bei denen öffentliche RSA-
710 Schlüssel bestätigt werden, folgende Tests auf die RSA-Schlüssel anwenden. Wenn ein u.
711 g. Test das Ergebnis FAIL als Ergebnis liefert, so ist der Schlüssel fehlerhaft und der TSP
712 muss die Zertifikatserstellung für diesen Schlüssel ablehnen.

- 713 1. Ist der öffentliche Exponent e (des untersuchten RSA-Schlüssels) prim und gilt
714 $2^{16} < e < 2^{256}$ (vgl. [BSI-TR-03116-1#3.2 RSA])?
715 Falls nein, ist das Ergebnis FAIL.
- 716 2. Ist der Modulus des untersuchten RSA-Schlüssels kleiner als 2^{2048} ?
717 Falls nein, ist das Ergebnis FAIL.
- 718 3. Ist der Modulus des untersuchten RSA-Schlüssels relativ prim zu allen Primzahlen
719 kleiner als 100?
720 Falls nein, ist das Ergebnis FAIL.

721 [\leq]

722 Erläuterungen zu A_17092 befinden sich in Abschnitt 910.2.

723 **A_17093 - RSA-Schlüssel Zertifikatserstellung, Entropie der Schlüsselkodierung**

724 Ein TSP KANN im Rahmen der Zertifikatsbeantragung, bei denen öffentliche RSA-
725 Schlüssel bestätigt werden, folgenden Test auf die RSA-Schlüssel anwenden. Wenn ein
726 Test das Ergebnis FAIL liefert, so ist der Schlüssel fehlerhaft und der TSP muss die
727 Zertifikatserstellung für diesen Schlüssel ablehnen.

- 728 1. Ist die Entropie des kodierten RSA-Schlüssels (im Sinne von
729 [gemSpec_Krypt#910.2], entropy()-Funktion) kleiner als 6.72? Falls ja, so ist das
730 Ergebnis FAIL.

731 [\leq]

732 Erläuterungen zu A_17093 befinden sich in Abschnitt 910.2.

733 2.5 Einmalpasswörter

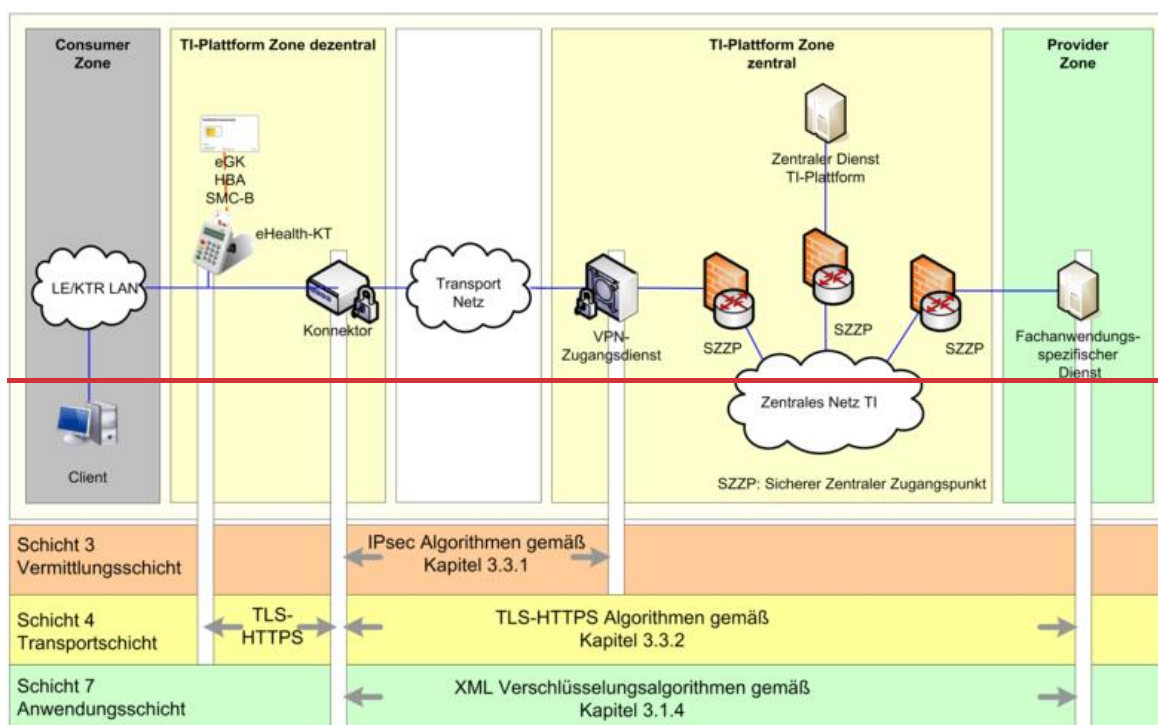
734 Bei verschiedenen Registrierungs- oder Anmeldeprozessen werden Einmalpasswörter
735 (insbesondere bei KIM) verwendet. Für diese Einmalpasswörter gilt folgende Vorgabe:

736 **A_22686 - Einmalpasswörter (One-Time-Passwords, OTP), Mindestentropie**

737 Alle Produkttypen, die Einmalpasswörter (One-Time-Passwords, OTP) erzeugen, MÜSSEN
738 sicherstellen, dass diese Einmalpasswörter eine Mindestentropie von 120 Bit besitzen. D.
739 h. sie werden zufällig erzeugt und sind mit praktischer Sicherheit - Wahrscheinlichkeit
740 gleich $1 - 2^{(-120)}$ - nicht erratbar. [\leq]

3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien

In den nachfolgenden Abschnitten werden die kryptographischen Algorithmen für verschiedene Einsatzszenarien spezifiziert. In diesem Zusammenhang sind ausschließlich die kryptographischen Aspekte der Einsatzszenarien relevant.



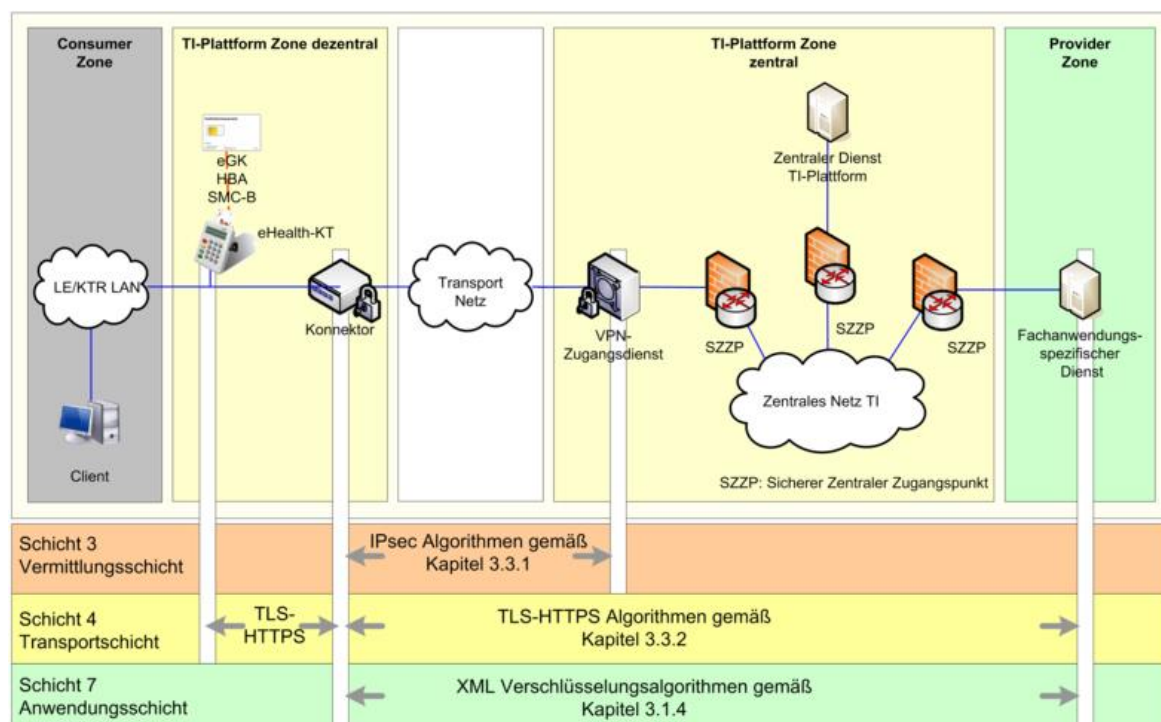


Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht

Abbildung 1 stellt beispielhaft die für die Vertraulichkeit von medizinischen Daten relevanten Algorithmen auf den verschiedenen OSI-Schichten in einer Übersicht dar. Es besteht in dieser Abbildung kein Anspruch auf Vollständigkeit.

3.1 Kryptographische Algorithmen für XML-Dokumente

GS-A_4370 - Kryptographische Algorithmen für XML-Dokumente

Alle Produkttypen, die XML-Dokumente

- verschlüsseln, MÜSSEN dies mittels CMS [RFC-5652] oder XMLEnc durchführen,
- signieren, MÜSSEN dies mittels CMS [RFC-5652] oder XMLDSig durchführen.

[<=]

XML-Signaturen sind bezüglich der verwendeten Algorithmen selbst beschreibend, die für die Erstellung einer Signatur verwendeten Algorithmen sind in der Signatur aufgeführt.

Zur vollständigen Spezifikation der Algorithmen für XML-Signaturen müssen für alle Signaturbestandteile Algorithmen spezifiziert werden. Die nachfolgenden Abschnitte wählen aus der Menge der zulässigen Algorithmen die jeweiligen Algorithmen für die einzelnen Einsatzszenarien aus.

Die Referenzierung von Algorithmen in XML-Signaturen und XML-Verschlüsselungen erfolgt nicht wie in Zertifikaten oder Signaturen binärer Daten über OIDs sondern über URIs. Die URIs der Algorithmen dienen als eindeutige Identifier und nicht dazu, dass unter der jeweils angegebenen URI die Beschreibung zu finden ist.

771 **Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs**

Algorithmen Identifier	Erläutert in
http://www.w3.org/2001/04/xmlenc#aes256-cbc	[XMLEnc]
http://www.w3.org/2001/04/xmlenc#rsa-1_5	[XMLEnc]
http://www.w3.org/2001/04/xmlenc#sha256	[XMLDSig]
http://www.w3.org/2000/09/xmldsig#enveloped-signature	[XMLDSig]
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256	[RFC-4051] bzw. [RFC-6931]
http://www.w3.org/2001/10/xml-exc-c14n#	[XMLCan_V1.0]
http://www.w3.org/2009/xmlenc11#aes256-gcm	[XMLEnc-1.1]
http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1	[RFC-6931]

772 **3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen**

773 **GS-A_4371-02 - XML-Signaturen für nicht-qualifizierte Signaturen**

774 Alle Produkttypen, die XML-Signaturen für nicht-qualifizierte Signaturen erzeugen oder
775 prüfen, MÜSSEN die Algorithmen und Vorgaben der Tabelle Tab_KRYPT_009
776 erfüllen. [<=]

777

778 **Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten**
779 **elektronischen XML-Signaturen**

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Signaturstandard	Signaturstandard	ETSI TS 101 903 V1.4.2 (2010-12) Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES) [ETSI-XAdES]	Die Verwendung des Standards ist für die Signatur von XML-Dokumenten verpflichtend, die nicht über CMS [RFC-5652] signiert werden.
kryptographische Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA-256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+ (Hinweis: siehe Abschnitt 4.1)	Die Verwendung des Algorithmus ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256 Die [XMLDSig] konforme Bezeichnung lautet: http://www.w3.org/2001/04/xmldsig-core#sha256	Die Verwendung des Algorithmus ist verpflichtend.

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Kryptographisches Token	Kryptographisches Token für die Signatur, bestehend aus einem privaten Schlüssel und einem zugehörigen X.509-Zertifikat	Identitäten gemäß einem der folgenden Abschnitte 2.1.1.1	Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.

3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen

GS-A_4372-02 - XML-Signaturen für qualifizierte elektronische Signaturen

Alle Produkttypen, die XML-Signaturen für qualifizierte elektronische Signaturen erzeugen oder prüfen, MÜSSEN die Vorgaben der Tabelle Tab_KRYPT_010 erfüllen. [<=]

Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Signaturstandard	Signaturstandard	ETSI TS 101 903 V1.4.2 (2010-12) Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES) [ETSI-XAdES]	Die Verwendung des Standards ist für die Signatur von XML-Dokumenten verpflichtend, die nicht über CMS [RFC-5652] signiert werden.

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA256 bis gemäß [SOG-IS] ECDSA mit SHA-256 bis gemäß [SOG-IS] (Hinweis: siehe Abschnitt 4.1)	Der Algorithmus muss für alle qualifizierten Signaturen verwendet werden. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256 Die [XMLDSig] konforme Bezeichnung lautet: http://www.w3.org/2001/04/xmldsig-core#sha256	Der Algorithmus muss für alle qualifizierten Signaturen verwendet werden.
Kryptographisches Token	Kryptographisches Token für die Signatur, bestehend aus einem privaten Schlüssel und einem zugehörigen X.509-Zertifikat	Identitäten gemäß dem folgenden Abschnitt 2.1.1.2	Es darf nur eine Identität, die den Ansprüchen qualifizierter Signaturen entspricht, verwendet werden.

786 **3.1.3 Webservice Security Standard (WSS)**

787 Nicht relevant für den Wirkbetrieb der TI.

3.1.4 XML-Verschlüsselung – Symmetrisch

GS-A_4373 - XML-Verschlüsselung - symmetrisch

Alle Produkttypen, die XML-Dokumente mittels [XMLEnc-1.1] verschlüsseln, MÜSSEN die folgenden Vorgaben umsetzen:

- Als symmetrische Block-Chiffre muss AES [FIPS-197] mit einer Schlüssellänge von 256 Bit im Galois/Counter Mode (GCM) gemäß [NIST-SP-800-38D] mit der Tag-Länge von 128 Bit verwendet werden.
- Die IVs dürfen sich bei gleichem Schlüssel nicht wiederholen (vgl. [NIST-SP-800-38D#S.25] und [BSI-TR-02102-1#S. 24]). Der IV soll eine Bitlänge von 96 Bit besitzen, seine Länge muss mindestens 96 Bit sein. Es wird empfohlen den IV zufällig zu wählen (vgl. [gemSpec_Krypt#GS-A_4367]).
- Hinweis: Im Normalfall ist davon auszugehen, dass für die Sicherung der Integrität und Authentizität der übertragenen Daten zudem noch eine Signatur der zu verschlüsselnden Daten notwendig ist.

[<=]

3.1.5 XML-Verschlüsselung – Hybrid

GS-A_4374 - XML-Verschlüsselung - Hybrid

Alle Produkttypen, die Dokumente mittels [XMLEnc-1.1] hybrid verschlüsseln, MÜSSEN das Dokument gemäß [gemSpec_Krypt#GS-A_4373] symmetrisch verschlüsseln, wobei der eingesetzte symmetrischer Schlüssel (jeweils) für eine spezifische Person oder Komponente asymmetrisch verschlüsselt wird.

(Hinweis: Analog zum Hinweis in [gemSpec_Krypt#GS-A_4373] gilt auch hier, dass im Normalfall für die Sicherung der Integrität und Authentizität der übertragenen Daten zudem noch eine Signatur dieser Daten notwendig ist.)

[<=]

GS-A_4376-02 - XML-Verschlüsselung - Hybrid, Schlüsseltransport RSAES-OAEP

Alle Produkttypen, die Dokumente mittels [XMLEnc-1.1] RSA-basiert hybrid ver- und entschlüsseln, MÜSSEN für den Schlüsseltransport den Algorithmus RSAES-OAEP gemäß [PKCS#1] verwenden.[<=]

3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung

3.2.1 Card-to-Card-Authentisierung G2

GS-A_4379 - Card-to-Card-Authentisierung G2

Alle Produkttypen, die die Card-to-Card-Authentisierung für Karten der Generation G2 durchführen, MÜSSEN dabei eine CV-Identität gemäß [gemSpec_Krypt#GS-A_4365-*] verwenden.

[<=]

Das Verfahren zur Durchführung der Card-to-Card-Authentisierung wird in [gemSpec_COS] spezifiziert.

3.2.2 Card-to-Server (C2S) Authentisierung und Trusted Channel G2

GS-A_4380 - Card-to-Server (C2S) Authentisierung und Trusted Channel G2

Alle Produkttypen, die eine Card-to-Server-Authentisierung für Karten der Generation G2 durchführen, MÜSSEN die folgenden Vorgaben berücksichtigen:

- Die Authentisierung muss mit AES analog [EN-14890-1#8.8] erfolgen.
- Die Schlüsselvereinbarung muss analog zu [EN-14890-1#8.8.2] erfolgen.

[<=]

Das Verfahren zur Durchführung der Card-to-Server-Authentisierung wird in [gemSpec_COS] spezifiziert.

C2S-Authentisierung bzw. der Trusted-Channel wird zwischen der Karte und dem zugeordneten Management-System verwendet.

Der Algorithmus AES ist nach [BSI-TR-03116-1] in der TI bis Ende 2029+ (meint bis Ende des Betrachtungsraums der TR) zulässig.

GS-A_4381-01 - Schlüssellängen Algorithmus AES

Alle Produkttypen, die den Algorithmus AES nutzen, MÜSSEN die Schlüssellängen gemäß Tabelle Tab_KRYPT_012 nutzen. [<=]

Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung

Algorithmen Typ	Algorithmus	Schlüssellänge
Authentifizierung und Verschlüsselung der Authentisierungsdaten	AES im CBC-Modus (OID 2.16.840.1.101.3.4.1)	128 Bit zulässig bis Ende 2029+

3.3 Netzwerkprotokolle

Im Gegensatz zu kryptographischen Verfahren für den Integritätsschutz oder die Vertraulichkeit von Daten, bei denen keine direkte Kommunikation zwischen dem Sender bzw. dem Erzeuger und dem Empfänger stattfindet, kann bei Netzwerkprotokollen eine Aushandlung des kryptographischen Algorithmus erfolgen. Das Ziel der nachfolgenden Festlegungen ist es daher, jeweils genau einen verpflichtend zu unterstützenden Algorithmus festzulegen, so dass eine Einigung zumindest auf diesen Algorithmus immer möglich ist. Zusätzlich können aber auch optionale Algorithmen festgelegt werden, auf die sich Sender und Empfänger ebenfalls im Zuge der Aushandlung einigen können. Es darf jedoch durch keine der Komponenten vorausgesetzt werden, dass der Gegenpart diese optionalen Algorithmen unterstützt.

3.3.1 IPsec-Kontext

GS-A_4382-04 - IPsec-Kontext - Schlüsselvereinbarung

Alle Produkttypen, die die Authentifizierung, den Schlüsselaustausch und die verschlüsselte Kommunikation im IPsec-Kontext durchführen, MÜSSEN die

- 859 Schlüsselvereinbarung mittels IKEv2 [RFC-7296] gemäß den folgenden Vorgaben
860 durchführen:
- 861 • Zur Authentisierung MUSS eine Identität mit einem X.509-Zertifikat gemäß
862 [gemSpec_Krypt#GS-A_4360-*) verwendet werden.
 - 863 • Für „Hash und URL“ MUSS SHA-1 verwendet werden.
 - 864 • Die Diffie-Hellman-Gruppe Gruppe 14 (definiert in [RFC-3526], verwendbar bis
865 Ende 2025) MUSS für den Schlüsselaustausch unterstützt werden. Zusätzlich
866 KÖNNEN Gruppen aus [BSI-TR-02102-3, Abschnitt 3.2.4, Tabelle 5], bei denen
867 der Verwendungszeitraum ein „+“ enthält, verwendet werden.
 - 868 • Der private DH-Exponent für den Schlüsselaustausch MUSS eine Länge von
869 mindestens 256 Bit haben.
 - 870 • Die Authentisierung der ephemeren (EC)DH-Parameter erfolgt durch eine Signatur
871 der Parameter durch den jeweiligen Protokollteilnehmer. Bei dieser Signatur MUSS
872 SHA-256 als Hashfunktion verwendet werden. Es SOLL die
873 Authentisierungsmethode „Digital Signature“ nach [RFC-7427] dabei verwendet
874 werden.
 - 875 • Bei den symmetrische Verschlüsselungsalgorithmen MUSS AES mit 256 Bit
876 Schlüssellänge im CBC-Modus unterstützt werden (sowohl für IKE-Nachrichten als
877 auch später für die Verschlüsselung von ESP-Paketen). Es KÖNNEN weitere
878 Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.1, Tabelle 2] bzw. [BSI-TR-
879 02102-3, Abschnitt 3.3.1, Tabelle 7] verwendet werden.
 - 880 • Für den Integritätsschutz (sowohl innerhalb von IKEv2 als auch anschließend für
881 ESP-Pakete) MUSS HMAC mittels SHA-256 unterstützt werden. Es KÖNNEN
882 weitere Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.3, Tabelle 4] bzw. [BSI-
883 TR-02102-3, Abschnitt 3.3.1, Tabelle 8] verwendet werden, andere Verfahren
884 dürfen nicht verwendet werden.
 - 885 • Als PRF MUSS PRF_HMAC_SHA2_256 unterstützt werden. Es KÖNNEN weitere
886 Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.2, Tabelle 3] verwendet werden,
887 andere Verfahren dürfen nicht verwendet werden.
 - 888 • Schlüsselaktualisierung: die IKE-Lifetime darf maximal 24*7 Stunden betragen
889 (Reauthentication). Die IPsec-SA-Lifetime darf maximal 24 Stunden betragen
890 (Rekeying). Der Initiator soll nach Möglichkeit vor Ablauf der Lifetime das
891 Rekeying anstoßen. Ansonsten muss der Responder bei Ablauf der Lifetime das
892 Rekeying von sich aus sicherstellen, bzw. falls dies nicht möglich ist, die
893 Verbindung beenden.
 - 894 • Für die Schlüsselberechnung muss Forward Secrecy [BSI-TR-02102-1, S.ix] (in
895 [RFC-7296] „Perfect Forward Secrecy“ genannt) gewährleistet werden. Meint die
896 Wiederverwendung von zuvor schon verwendeten (EC-)Diffie-Hellman-Schlüsseln
897 ([RFC-7296, Abschnitt 2.12]) ist nicht erlaubt.

898 [**<=**]

899 Ziel ist es zum Zeitpunkt der IKE-SA-Reauthentication ausgeführte Anwendungsfälle
900 nicht zu unterbrechen. Aktuell wird aufgrund von TIP1-A_4492 im Rahmen der
901 Reauthentication dem Konnektor eine neue (i.d.R. andere) VPN-TI-IP-Adresse
902 zugewiesen, was dazu führt, dass bestehende TCP-Verbindungen in die TI effektiv
903 zerstört und laufende Anwendungsfälle unterbrochen werden. Perspektivisch wird die
904 folgende Anforderung als MUSS-Anforderung in TIP1-A_4492 integriert.

GS-A_5547 - gleiche VPN-IP-Adresse nach Reauthentication

Der VPN-Zugangsdienst KANN nach einer Reauthentication (vgl. GS-A_4382-* Spiegelstrich „Schlüsselaktualisierung“) die gleiche VPN-IP-Adresse wie vor der Reauthentication vergeben. Die Reauthentication ist in Bezug auf TIP1-A_4492 nicht als „neue Verbindung/Neuaufbau des Tunnels“ zu betrachten.
[<=]

Da noch nicht alle VPN-Zugangsdienste technisch in der Lage sind GS-A_5547 umzusetzen werden als Symptomlinderung die Gültigkeitsdauern der ausgehandelten Schlüssel erhöht, auch in Anbetracht, dass weitere Sicherheitsmaßnahmen (bspw. TIP1-A_5389) umgesetzt werden neben den klassischen Prüfungen, die im Rahmen einer Reauthentication durchgeführt werden.

GS-A_5548 - Mindestgültigkeitszeiten IKE- und IPsec-SAs (Konnektor)

Der Konnektor MUSS die Konfiguration der Gültigkeitsdauern der IKE- bzw. IPsec-SAs auf (1) mindestens 90% und (2) kleiner als 100% der in GS-A_4382-* Spiegelstrich „Schlüsselaktualisierung“ aufgeführten Maximalwerte setzen.
[<=]

Auszug Beispielkonfiguration /etc/ipsec.conf

```
ikelifetime=161h
lifetime=23h
margintime = 20m
rekeyfuzz = 40%
keyexchange=ikev2
```

GS-A_5549 - Mindestgültigkeitszeiten IKE- und IPsec-SAs (VPN-Zugangsdienst)

Der VPN-Zugangsdienst MUSS die Konfiguration der Gültigkeitsdauern der IKE- bzw. IPsec-SAs auf die in GS-A_4382-* Spiegelstrich „Schlüsselaktualisierung“ aufgeführten Maximalwerte setzen.
[<=]

GS-A_5508 - IPsec make_before_break

Alle Produkttypen, die mittels IPsec Daten schützen, MÜSSEN die Reauthentication (vgl. [RFC-7296#2.8.3 „Reauthentication is done by [...]“]) durchführen, indem die neue IKE-SA aufgebaut wird bevor die bestehende IKE-SA gelöscht wird.
[<=]

GS-A_4383 - IPsec-Kontext – Verschlüsselte Kommunikation

Alle Produkttypen, die mittels IPsec-Daten schützen, MÜSSEN dies ausschließlich auf Grundlage der in GS-A_4382-* als zulässig aufgeführten Verfahren und Vorgaben tun.
[<=]

A_14652 - SZZP-light, asymmetrischen Schlüssel maximale Gültigkeitsdauer

Die Lebensdauer von asymmetrischen Schlüsseln für die IPsec-Verbindungen im SZZP-light sowie Sicherheitgateway Bestandsnetze und somit die in einem Zertifikat angegebene Gültigkeitsdauer DARF NICHT 5 Jahre überschreiten.
[<=]

3.3.2 TLS-Verbindungen

GS-A_4385 - TLS-Verbindungen, Version 1.2

949 Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die TLS-Version
950 1.2 [RFC-5246] unterstützen.
951 [\leq]

952 **A_18467 - TLS-Verbindungen, Version 1.3**

953 Alle Produkttypen, die Übertragungen mittels TLS durchführen, KÖNNEN die TLS-Version
954 1.3 [RFC-8446] unterstützen, falls sie

955 1. dabei nur nach [BSI-TR-02102-2] empfohlene Konfigurationen
956 (Handshake-Modi, (EC)DH-Gruppen, Signaturverfahren, Ciphersuiten etc.)
957 verwenden, und

958 2. mindestens die Ciphersuite "TLS_AES_128_GCM_SHA256" dabei unterstützen.

959 [\leq]

960 **A_18464 - TLS-Verbindungen, nicht Version 1.1**

961 Alle Produkttypen, die Übertragungen mittels TLS durchführen, DÜRFEN NICHT die TLS-
962 Version 1.1 [RFC-4346] unterstützen. [\leq]

963 **GS-A_4387 - TLS-Verbindungen, nicht Version 1.0**

964 Alle Produkttypen, die Übertragungen mittels TLS durchführen, DÜRFEN NICHT die TLS-
965 Version 1.0 unterstützen. [\leq]

966 **GS-A_5035 - Nichtverwendung des SSL-Protokolls**

967 Alle Produkttypen, die Daten über Datenleitungen übertragen wollen, DÜRFEN NICHT das
968 SSL-Protokoll unterstützen. [\leq]

969 Bei TLS 1.1 oder älter ist im Rahmen des TLS-Verbindungsaufbaus die Verwendung von
970 SHA-1 bei der Erstellung und Prüfung von Signaturen (TLS-Handshake) notwendig. Es
971 gibt keine Möglichkeit der Aushandlung/Vereinbarung der Verwendung von
972 kryptographisch höherwertigeren Hashfunktionen dafür. Dies wurde erst mit TLS 1.2
973 möglich. SHA-1 erbringt in Konstruktionen, die nur die kryptographische
974 Einwegeneigenschaft der Hashfunktion benötigen (bspw. bei der HMAC-Berechnung auf
975 SHA-1-Basis) noch ein -- zwar nicht hochwertiges, aber immer noch -- vertretbares
976 Sicherheitsniveau. Die allgemeine Kollisionsresistenz, so wie sie bei Signaturen
977 notwendig ist, kann SHA-1 nicht mehr leisten. SHA-1 wurde in diesem Aspekt praktisch
978 (i. S. v. nicht nur theoretisch) -- und öffentlichkeitswirksam demonstriert -- gebrochen.
979 Aus diesem Grunde hat die IETF alle TLS Version unter 1.2 abgekündigt und bspw. alle
980 Webbrowser-Hersteller haben die Unterstützung von TLS-Versionen unter 1.2 deaktiviert.

981 Einen lesenswerten Abriss bekannter Angriffe auf TLS findet man in [TLS-Attacks], vgl.
982 auch [Breaking-TLS].

983 Analog zu IKE/IPsec und GS-A_4382-* (Spiegelstrich 5) wird deshalb folgend mit
984 A_21275-* eine Verwendung von SHA-1 bei der Signaturerstellung und -prüfung im
985 Kontext des TLS-Handshakes untersagt.

986 **A_21275-01 - TLS-Verbindungen, zulässige Hashfunktionen bei Signaturen im
987 TLS-Handshake**

988 Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN sicherstellen,
989 dass

990 1. sie im Rahmen der Erstellung und Prüfung von digitalen Signaturen im Rahmen
991 des TLS-Handshakes ausschließlich folgende kryptographisch geeignete
992 Hashfunktionen verwenden:

993 a. SHA-256, SHA-384, SHA-512 [FIPS-180-4]

994 b. SHA3-256, SHA3-384, SHA3-512 [FIPS-202]

2. sie dabei mindestens SHA-256 unterstützen,

(Bitte die Umsetzungshinweise in Bezug auf die "signature_algorithms"-Extension in gemSpec_Krypt#A_21275-* beachten.)[<=]

Umsetzungshinweise zu A_21275-*:

Bei den Anwendungsfällen der TI-Anwendungen sind die Mehrzahl der TLS-Verbindungen einseitig authentisiert. D. h. beim TLS-Handshake signiert nur der TLS-Server dessen (EC)DH-Schlüssel. Bei der Initiierung der TLS-Verbindung sendet der TLS-Client in der "signature_algorithms"-Extension beim ClientHello. In der Extension werden alle vom Client unterstützten Hashfunktionen kodiert. Dort muss also nach A_21275-* mindestens SHA-256 enthalten sein. Bei TLS 1.2 wird von fast allen TLS-Bibliotheken ebenfalls SHA-1 angegeben, dieses Verhalten lässt sich im Normalfall nicht ohne Code-Änderungen in den Bibliotheken verändern -- dieses Verhalten widerspricht zunächst A_21275-*. Das bloße Aufführen von SHA-1 als grundsätzlich unterstützte Hashfunktion soll nicht als fehlerhaftes Verhalten gelten. Wichtig für die Umsetzung von A_21275-* sind die tatsächlich erstellten Signaturen und die Prüfung dieser Signaturen.

Informationen zu Algorithmen in der "signature_algorithms"-Extension findet man in [RFC-5246#7.4.1.4.1.] und [RFC-8446-4.2.3.], vgl. auch [RFC-9155].

GS-A_4384-03 - TLS-Verbindungen

Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die folgenden Vorgaben erfüllen:

- Zur Authentifizierung MUSS eine X.509-Identität gemäß [gemSpec_Krypt#GS-A_4359-*] verwendet werden.
- Als Cipher-Suite MÜSSEN TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 und TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 unterstützt werden.
- Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE" im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5] unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in GS-A_4384-* aufgeführt DÜRFEN NICHT verwendet werden.
- Es KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützt werden.

[<=]

Erläuterung zu GS-A_4384-*:

In einigen Konstellationen (ePA-FdV auf iOS-Geräten) ist die Verwendung von brainpool-Kurven nur schwer möglich. Dort bedeutet die SOLL-Bestimmung aus GS-A_4384-*, dass es zulässig ist auf die brainpool-Kurven-Unterstützung dort zu verzichten.

Hinweis: hinter den folgenden Identifier-n verbirgt sich kryptographisch gesehen jeweils die gleiche Kurve:

ansix9p256r1	[ANSI-X9.62#L.6.4.3]
ansip256r1	http://oid-info.com/get/1.2.840.10045.3.1.7
prime256v1	[RFC-3279], openssl ecparam -list_curves

ansix9p256r1	[ANSI-X9.62#L.6.4.3]
secp256r1	[RFC-5480], http://www.secg.org/collateral/sec2_final.pdf
P-256	[FIPS186-5]

1033 Analog P-384 [FIPS186-5]:

ansix9p384r1	[ANSI-X9.62#L.6.5.2]
ansip384r1	http://oid-info.com/get/1.3.132.0.34
prime384v1	[RFC-3279], openssl ecparam -list_curves
secp384r1	[RFC-5480], http://www.secg.org/collateral/sec2_final.pdf
P-384	[FIPS186-5]

1034

1035 **GS-A_5541 - TLS-Verbindungen als TLS-Klient zur Störungssampel oder SM**

1036 Alle Produkttypen, die das TLS-Protokoll als TLS-Klient zur Störungssampel oder zum
1037 Service-Monitoring verwenden, KÖNNEN

1038 (1) auf die explizite Prüfung, dass der TLS-Server die (EC)DH-Gruppe für den
1039 ephemeren (EC)DH-Schlüsselaustausch spezifikationskonform gewählt hat (vgl.
1040 GS-A_4384-* und A_17124-* Punkt 4), verzichten,

1041 und

1042 (2) davon ausgehen, dass der TLS-Server die Auswahl der TLS-
1043 Verbindungsparameter (TLS-Version, TLS-Ciphersuite etc.) korrekt, i.S.v.
1044 spezifikationskonform, durchführt.

1045 [**<=**]

1046 **GS-A_5580-01 - TLS-Klient für betriebsunterstützende Dienste**

1047 Alle Produkttypen, die das TLS-Protokoll als TLS-Klient für Betriebsunterstützende
1048 Dienste (Service-Monitoring, Betriebsdaten-Erfassung etc.) verwenden, MÜSSEN das vom
1049 Betriebsunterstützenden Dienst präsentierte Zertifikat prüfen. Für diese Prüfung MUSS
1050 entweder TUC_PKI_018 oder die vereinfachte Zertifikatsprüfung (GS-A_5581
1051 „TUC vereinfachte Zertifikatsprüfung“ (Komponenten-PKI)) verwendet werden. [**<=**]

1052 **A_22430 - TLS-Klient für betriebsunterstützende Dienste im Internet**

1053 Alle Produkttypen, welche die Betriebsdatenerfassung im Internet nutzen, MÜSSEN
1054 prüfen, ob das von der Betriebsdatenerfassung an der Internetschnittstelle während des
1055 TLS-Verbindungsaufbaus präsentierte TLS-Serverzertifikat gültig ist (d. h. u. a. per
1056 Zertifikatsprüfung rückführbar auf ein CA-Zertifikat einer CA, die die "CA/Browser Forum
1057 Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates"
1058 (<https://cabforum.org/baseline-requirements-documents/>) erfüllt) und für den
1059 erwarteten FQDN ausgestellt wurde. Bei negativen Prüfergebnis MUSS der TLS-
1060 Verbindungsaufbau zur Betriebsdatenerfassung abgelehnt werden. [**<=**]

1061 Bei bestimmten Produkttypen, bspw. TSPs, beschränkt sich die Prüfung von Zertifi
1062 katen beim TLS-Verbindungsaufbau in Bezug auf die TI ausschließlich auf die Prüfung des
1063 Zertifikats des Service Monitorings oder anderer betriebsunterstützender Dienste. Dafür
1064 ist der TUC_PKI_018 unangemessen leistungsstark und komplex. Deshalb wird folgend
1065 mit GS-A_5581 eine passgenauere Zertifikatsprüfung als Alternative definiert.

1066 **GS-A_5581 - "TUC vereinfachte Zertifikatsprüfung" (Komponenten-PKI)**

1067 Alle Produkttypen, die eine Zertifikatsprüfung
1068 konform zu in dieser Anforderung definierten „TUC vereinfachte Zertifikatsprüfung“
1069 durchführen wollen, erreichen dies indem sie folgende Vorgaben erfüllen.

1070 (1) Es MUSS einen Prozess geben der authentisch und integer die Komponenten-CA-
1071 Zertifikate der TI regelmäßig (mindestens einmal pro Monat) ermittelt. Diese sind Basis
1072 für die folgenden Prüfschritte.

1073 (2) Es MUSS geprüft werden, ob im vom TLS-Server präsentierten Zertifikat der
1074 korrekte (i. S. v. vom TLS-

1075 Client erwartete) FQDN enthalten ist (bspw. monitoring-update.stempel.telematik).

1076 (3) Es MUSS geprüft werden, ob das präsentierte Zertifikat per Signaturprüfung
1077 rückführbar ist zu einem der CA-Zertifikate aus (1).

1078 (4) Es MUSS geprüft werden, ob das präsentierte Zertifikat zeitlich gültig ist.

1079
1080 Wenn einer der Prüfschritte aus (2) bis (4) fehlschlägt, MUSS der Verbindungsaufbau
1081 abgebrochen werden.

1082
1083 Es gibt GS-A_5581 folgend in gemSpec_Krypt Anwendungshinweise. [<=]

1084 Als Hilfestellung: für die Umsetzung von GS-A_5581 Spiegelstrich (1) kann man bspw.
1085 folgende Maßnahmen wählen.

1086 (a) Übergabe bei einem Vororttermin in der gematik,

1087 (b) Regelmäßiger Download über <https://download.tsl.ti-dienste.de/>

1088 (c) Verwendung einer dedizierten Software zum Download, Signaturprüfung und
1089 Auswertung der TI-TSL (es existiert dafür jeweils mindestens eine Open-Source-
1090 Lösung und eine kommerzielle Lösung)

1091 (d) oder andere Lösung, die die Integrität und Authentizität der Zertifikate sicherstellt.

1092

1093 Ziel ist es, dass für die Verbindung zur Störungsampel oder zum Service Monitoring auch
1094 einfach verfügbare und einfach verwendbare HTTPS-Clients wie `wget` oder `curl`
1095 verwendet werden können.

1096

1097 Unter der Annahme, dass

1098 (a) im Verzeichnis `/etc/TI-Komponenten-CAs` die in GS-A_5581 Punkt (1) aufgeführten
1099 Zertifikate liegen und

1100 (b) die an die Störungsampel zu sendende Information (i. d. R. unsignierte XML-Daten)
1101 in der Datei `SOAP_Daten` liegen,
1102 erfüllen folgende Aufrufe die Punkt (2)-(4) aus GS-A_5581.

1103 I.

1104 `wget --ca-directory=/etc/TI-Komponenten-CAs --post-`
1105 `file=SOAP_Daten https://monitoring-`

1106 `update.stempel.telematik:8443/I_Monitoring_Message`

1107 II.

1108 curl --capath /etc/TI-Komponenten-CAs -d SOAP_Daten https://monitoring-
1109 update.stempel.telematik:8443/I_Monitoring_Message

1110 **GS-A_5542 - TLS-Verbindungen (fatal Alert bei Abbrüchen)**

1111 Alle Produkttypen, die das TLS-Protokoll verwenden, MÜSSEN sicherstellen, dass alle von
1112 ihnen durchgeführten Verbindungsabbrüche (egal ob im noch laufenden TLS-Handshake
1113 oder in einer schon etablierten TLS-Verbindung) mit einer im TLS-Protokoll aufgeführten
1114 Fehlermeldung (fataler Alert) angekündigt werden, außer das TLS-Protokoll untersagt
1115 dies explizit.

1116 [\leq]

1117 Sicherheitsziel bei der Verwendung von TLS in der TI ist die Forward Secrecy [BSI-TR-
1118 02102-1, S. ix], was sich u. a. in den vorgegebenen Cipher-Suites (vgl. GS-A_4384-
1119 * und A_17124-*) widerspiegelt. Um dieses Ziel zu erreichen, muss sichergestellt
1120 werden, dass in regelmäßigen Abständen frisches Schlüsselmateriel über einen
1121 authentisierten Diffie-Hellman-Schlüsselaustausch gebildet wird, welches das alte
1122 Material ersetzt, wobei das alte Material sowohl im Klienten als auch im Server sicher
1123 gelöscht wird. Insbesondere bei der Nutzung von TLS-Resumption (vgl. [RFC-5246, S.
1124 36] oder [RFC-5077]) kann die Dauer einer TLS-Session deutlich länger sein als die
1125 Lebensdauer der TCP-Verbindung innerhalb welcher der initiale Schlüsselaustausch
1126 stattgefunden hat. Aus diesem Grunde werden analog zu den IPsec-Vorgaben (vgl.
1127 [gemSpec_Krypt#GS-A_4383]) Vorgaben für die maximale Gültigkeitsdauer dieses
1128 Schlüsselmateriels gemacht (vgl. auch [SDH-2016]).

1129

1130 **GS-A_5322 - Weitere Vorgaben für TLS-Verbindungen**

1131 Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN u. a. folgende
1132 Vorgaben erfüllen:

- 1133
- 1134 • Falls der Produkttyp als *Klient* oder als *Server* im Rahmen von TLS an einer
1135 Session-Resumption mittels SessionID (vgl. [RFC-5246, Abschnitt 7.4.1.2])
1136 teilnimmt, MUSS er sicherstellen, dass nach spätestens 24 Stunden das über den
1137 Diffie-Hellman-Schlüsselaustausch ausgehandelte Schlüsselmateriel und alles
1138 davon abgeleitete Schlüsselmateriel (vgl. [RFC-5246, Abschnitt 8.1 und 6.3]) bei
ihm sicher gelöscht wird.
 - 1139 • Falls der Produkttyp als *Klient* im Rahmen von TLS an einer Session-Resumption
1140 nach [RFC-5077] teilnimmt, MUSS er sicherstellen, dass nach spätestens 24
1141 Stunden das über den Diffie-Hellman-Schlüsselaustausch ausgehandelte
1142 Schlüsselmateriel und alles davon abgeleitete Schlüsselmateriel (vgl. [RFC-5246,
1143 Abschnitt 8.1 und 6.3]) bei ihm sicher gelöscht wird. Damit verbundene
1144 SessionTickets MUSS er ebenfalls sicher löschen.
 - 1145 • Falls der Produkttyp als *Server* im Rahmen von TLS an einer Session-Resumption
1146 nach [RFC-5077] teilnimmt, MUSS er sicherstellen, dass nach spätestens 24
1147 Stunden das über den Diffie-Hellman-Schlüsselaustausch ausgehandelte
1148 Schlüsselmateriel und alles davon abgeleitete Schlüsselmateriel (vgl. [RFC-5246,
1149 Abschnitt 8.1 und 6.3]) bei ihm sicher gelöscht wird. Damit verbundene
1150 SessionTickets MUSS er, falls bei ihm vorhanden, sicher löschen. Das
1151 Schlüsselmateriel, dass bei der Erzeugung des SessionTickets (für die Sicherung
1152 von Vertraulichkeit und Authentizität der SessionTickets) verwendet wird, MUSS
1153 spätestens alle 48 Stunden gewechselt werden und das alte Material MUSS sicher
1154 gelöscht werden. Als kryptographische Verfahren zur Erzeugung/Sicherung der
1155 SessionTickets MÜSSEN ausschließlich nach [BSI-TR-03116-1] zulässige

1156 Verfahren verwendet werden und das Schlüsselmaterial muss die
1157 Entropieanforderungen gemäß [gemSpec_Krypt#GS-A_4368] erfüllen.

1158 • Falls ein Produkttyp als *Klient* oder *Server* im Rahmen von TLS die Renegotiation
1159 unterstützt, so MUSS er dies ausschließlich nach [RFC-5746] tun. Ansonsten
1160 MUSS er die Renegotiation-Anfrage des Kommunikationspartners ablehnen.

1161 [**<=**]

1162 Aktuell gibt es in der TI keine Anwendungsfälle (Wechsel der kryptographischen Identität
1163 innerhalb einer TLS-Verbindung oder erzwungene Schlüssel-„Auffrischung“ der
1164 Sitzungsschlüssel), die eine Session-Renegotiation im Rahmen von TLS unmittelbar
1165 erforderlich machen. Lesenswert bez. des Themas Sicherheitsprobleme mit TLS-Session-
1166 Renegotiation ist [IR-2014, S.181ff] und allgemein [CM-2014].

1167 Es hat sich gezeigt, dass es notwendig ist weitere Vorgaben zur TLS-Renegotiation für die
1168 Sicherstellung der Interoperabilität zwischen Komponenten und Diensten zu machen.

1169 **GS-A_5524 - TLS-Renegotiation eHealth-KT**
1170 Das eHealth-KT MUSS beim einen TLS-Verbindungsaufbau die TLS-Extension
1171 „renegotiation_info“ gemäß [RFC-5746] senden, unabhängig davon ob das eHealth-KT
1172 TLS-Renegotiation unterstützt oder nicht unterstützt. Im weiteren TLS-Protokollverlauf
1173 MUSS das eHealth-KT eines der beiden folgenden Verhalten aufweisen:

1174 1. Entweder das eHealth-KT lehnt jede Renegotiation mit einem „no_renegotiation“-
1175 Alert ab, oder

1176 2. das eHealth-KT unterstützt die Renegotiation gemäß [RFC-5746], wobei
1177 ausschließlich „Secure Renegotiation“ durch das eHealth-KT akzeptiert werden
1178 (d.h., falls das „secure_renegotiation“-flag [RFC-5746#3.7] gleich FALSE ist,
1179 muss das KT die Renegotiation mit einem „no_renegotiation“-Alert ablehnen).

1180 [**<=**]

1181 **GS-A_5525 - TLS-Renegotiation Konnektor**
1182 Der Konnektor MUSS den RFC 5746 (TLS-Renegotiation-Indication-Extension [RFC-
1183 5746]) unterstützen und nur „Secure Renegotiation“ erlauben und durchführen.
1184 [**<=**]

1185 Für eine Java-Implementierung bedeutet dies, dass allowLegacyHelloMessages und
1186 allowUnsafeRenegotiation jeweils auf false gesetzt sind ("Modus Strict",
1187 <http://www.oracle.com/technetwork/java/javase/overview/tlsreadme2-176330.html>).

1188 Da der Angriff [Ray-2009], der zur Erstellung des [RFC-5746] führte, praktisch
1189 durchführbar war, wurde die Mehrzahl der existierenden TLS-Bibliotheken relativ zügig
1190 angepasst (Timeline in [IR-2014, S. 190, Abbildung 7.2]). (Vgl. die erste Spalte „Secure
1191 Renegotiation“ bei
1192 https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations#Extensions) Um für
1193 den unwahrscheinlichen Fall, dass aktuell ein schon bestehender Fachdienst Probleme bei
1194 der Umsetzung der folgenden Anforderung hat, wurde diese als SOLL-Anforderung
1195 formuliert. Es ist geplant diese Anforderung zukünftig in eine MUSS-Anforderung zu
1196 ändern.

1197 **GS-A_5526 - TLS-Renegotiation-Indication-Extension**
1198 Alle Produkttypen, die das TLS-Protokoll verwenden, SOLLEN den RFC 5746 (TLS-
1199 Renegotiation-Indication-Extension [RFC-5746]) unterstützen.
1200 [**<=**]

1201 Die folgende Anforderung hat den Zweck die Interoperabilität zwischen Konnektor und
1202 Intermediär sicherzustellen.

1203 **GS-A_5527 - TLS-Renegotiation-Indication-Extension Intermediär**

1204 Der Intermediär MUSS den RFC 5746 (TLS-Renegotiation-Indication-Extension [RFC-
1205 5746]) unterstützen und nur „Secure Renegotiation“ erlauben und durchführen.
1206 [\leq]

1207 Für eine verbesserte Interoperabilität zu bestimmten TLS-Implementierungen (bspw.
1208 SChannel, vgl. auch (
1209 https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations bzw.
1210 <https://www.ssllabs.com/ssltest/clients.html>) sollen im Konnektor zusätzlich zu den
1211 Cipher-Suiten aus GS-A_4384-* weitere Cipher-Suiten unterstützt werden. Mit der
1212 mittelfristigen Anhebung des zu erreichenden Sicherheitsniveaus auf 120 Bit (vgl. [SOG-
1213 IS] und [BSI-TR-03116-1]) werden die folgenden Cipher-Suiten mittelfristig
1214 verpflichtend. In diesem Kontext spielt die Performanz (3000 Bit Diffie-Hellman vs. 256
1215 Bit Elliptic Curve Diffie-Hellman) bei Embedded-Geräten wie dem Konnektor eine wichtige
1216 Rolle.

1217 **GS-A_5345-04 - TLS-Verbindungen Konnektor**

1218 Der Konnektor MUSS für die TLS gesicherten Verbindungen neben den in
1219 [gemSpec_Krypt#GS-A_4384-*] aufgeführten Ciphersuiten folgende Vorgaben
1220 umsetzen:
1221

- 1222 1. Der Konnektor MUSS zusätzlich folgende Ciphersuiten unterstützen:
- 1223 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x27),
 - 1224 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28),
 - 1225 • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2f) und
 - 1226 • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30).
- 1227 2. Der Konnektor KANN weitere Ciphersuiten aus [TR-02102-2, Abschnitt 3.3.1
1228 Tabelle 1] unterstützen.
- 1229 3. Falls Ciphersuiten aus Spiegelstrich (1) oder (2) unterstützt werden,
- 1230 a. MÜSSEN bei dem ephemeren Elliptic-Curve-Diffie-Hellman-
1231 Schlüsselaustausch die Kurven P-256 und P-384 [FIPS-186-5] unterstützt
1232 werden,
 - 1233 b. MÜSSEN die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639]
1234 und [RFC-7027]) unterstützt werden.
- 1235 Andere Kurven SOLLEN NICHT verwendet werden.
- 1236 4. Falls Ciphersuiten aus (1) oder (2) unterstützt werden, so MÜSSEN diese im CC-
1237 Zertifizierungsverfahren berücksichtigt werden.

1238 [\leq]

1239 **A_23226-01 - TLS-Verbindung, Konnektor: Legacy-KT-Unterstützung**

1240 Der Konnektor MUSS für die Unterstützung von alten eHealth-KT folgende TLS-Vorgaben
1241 ebenfalls unterstützen:

- 1242 • Als Cipher Suite MUSS TLS_DHE_RSA_WITH_AES_128_CBC_SHA oder
1243 TLS_DHE_RSA_WITH_AES_256_CBC_SHA unterstützt werden.

1244 • Dabei MUSS für die Schlüsselaushandlung Gruppe 14 (definiert in [RFC-3526],
1245 verwendbar bis Ende 2025) verwendet werden.

1246 • Der private DH-Exponent für den Schlüsselaustausch MUSS eine Länge von
1247 mindestens 256 Bit haben.

1248 [**<=**]

1249 **A_18183 - TLS-Protokoll-Verwendung in WANDA Basic**

1250 Falls ein Anbieter einer anderen Anwendung des Gesundheitswesens ohne Zugriff auf
1251 Dienste der TI in angeschlossenen Netzen des Gesundheitswesens (WANDA Basic) das
1252 TLS-Protokoll verwendet, so MUSS er dabei ausschließlich Ciphersuiten und
1253 Domainparameter (Schlüssellängen, Kurvenparameter etc.), die nach [TR-02102-2]
1254 empfohlen sind, verwenden. [**<=**]

1255 Erläuterung: Eine andere Anwendung des Gesundheitswesens ohne Zugriff auf Dienste
1256 der TI in angeschlossenen Netzen des Gesundheitswesens (WANDA Basic) muss beim
1257 TLS-basierten Nachrichtentransport durch die TI nach [TR-02102-2] sichere Cipher-
1258 Suiten und Domainparameter verwenden. Für solch eine Anwendung ist eine die
1259 Interoperabilität mit TI-Diensten sicherstellende Einschränkung der Cipher-Suiten und
1260 Domainparameter nach GS-A_4384-* und A_17124-* nicht notwendig, d. h. beide
1261 Anforderungen gelten nicht für solche Anwendungen, sondern A_18183 gilt.

1262 Gleiches gilt für die Verwendung von TLS für die Anbindung von Leistungserbringernetzen
1263 an das TI-Gateway, da bei dieser VPN-Anbindung Client und Server Teil des selben
1264 Produkttyps sind (TI-Gateway Zugangsmodul).

1265 **A_24779-01 - TI-Gateway-Zugangsmodul und eHealth-CardLink - TLS-Cipher- 1266 Suiten**

1267 Das TI-Gateway Zugangsmodul und der eHealth-CardLink MÜSSEN, falls sie das TLS-
1268 Protokoll für die Sicherung der Datenübertragung aus dem Nutzernetzwerk zum
1269 Zugangsmodul des TI-Gateway bzw. vom Client des Nutzers zum eHealth-CardLink
1270 verwenden, ausschließlich TLS-Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle
1271 1] mit den dort vorgegebenen Domainparametern (Schlüssellänge, ECC-Kurven-
1272 Parameter etc.) verwenden bzw. bei Verwendung von TLS 1.3 die Vorgaben aus [TR-
1273 02102-2, Abschnitt 3.4] befolgen.

1274 [**<=**]

1275

1276 **A_18986 - Fachdienst-interne TLS-Verbindungen**

1277 Alle Produkttypen, die Übertragungen mittels TLS durchführen, die nur innerhalb ihres
1278 Produkttypen verlaufen (bspw. ePA-Aktensystem interne TLS-Verbindungen zwischen
1279 dem Zugangsgateway und der Komponente Authentisierung), KÖNNEN für diese TLS-
1280 Verbindungen neben den in GS-A_4384-* und ggf. A_17124-* festgelegten TLS-
1281 Vorgaben ebenfalls alle weiteren in [TR-02102-2] empfohlenen TLS-Versionen und TLS-
1282 Ciphersuiten mit den jeweiligen in [TR-02102-2] dafür aufgeführten Domainparametern
1283 (Kurven, Schlüssellängen etc.) verwenden. [**<=**]

1284 Erläuterung: A_18986 "befreit" Produkttypen-interne TLS-Verbindungen von der
1285 Beschränkung auf die Vorgaben von GS-A_4384-* und ggf. A_17124-* und erweitert
1286 diese Vorgaben auf die Gesamtheit der in [TR-02102-2] empfohlenen TLS-
1287 Konfigurationen.

1288 Hinweis: In Abschnitt "3.15.3- ePA-spezifische TLS-Vorgaben " gibt es weitere TLS-
1289 Vorgaben.

3.3.3 DNSSEC-Kontext

Hinweis: Die Verwendung von DNSSEC innerhalb der TI wird seit 2018 nicht mehr in den TI-Spezifikationen vorgeschrieben. DNSSEC wird in der TI von den meisten Komponenten und Fachdiensten nicht mehr verwendet. Die Unterstützung von DNSSEC in den DNS-Servern der TI wird nur noch für Legacy-Anwendungen weiterbetrieben. In der TI sind fast alle Kommunikationen über TLS und damit über die PKI der TI abgesichert.

GS-A_4388 - DNSSEC-Kontext

Alle Produkttypen, die DNSSEC verwenden, MÜSSEN die Algorithmen und Vorgaben gemäß Tabelle Tab_KRYPT_017 erfüllen.

[<=]

Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC

Algorithmen Typ	Algorithmus	Schlüssellänge
TSIG – symmetrischer Schlüssel zur Absicherung der Transaktionskanäle zwischen zwei Name-Server-Instanzen bei Zonentransfers, Änderungsbenachrichtigungen, dynamischen Updates und rekursiven Queries.	HMAC-SHA-256	256 Bit
DNSSEC ZSK Asymmetrische Schlüssel zur Wahrung der Authentizität und Integrität von Zonendatenobjekten.	RSA-SHA-256 [RFC-5702]	2048 Bit
DNSSEC KSK Asymmetrische Schlüssel zur Wahrung der Authentizität und Integrität von Zonendatenobjekten.	RSA-SHA-256 [RFC-5702]	2048 Bit

Hinweis: Nach [RFC-5702] ist die Verwendung von SHA-256 [FIPS-180-4] möglich. Schlüssellängen von RSA zwischen 512 bis 4096 Bit sind seit den Anfängen von DNSSEC möglich. Bei TSIG ist nach [RFC-4635] auch SHA-256 verwendbar und bspw. von bind seit der Version 9.5 unterstützt.

3.4 Masterkey-Verfahren (informativ)

Die gematik wurde aufgefordert, beispielhaft ein mögliches Ableitungsverfahren für einen versichertenindividuellen symmetrischen Schlüssel auf Grundlage eines Ableitungsschlüssels (Masterkey) aufzuführen. Ein Kartenherausgeber ist frei in der Wahl seines Ableitungsverfahrens. Jedoch müssen beim Einsatz eines Ableitungsverfahrens, um die Qualität der Ableitung zu garantieren, insbesondere folgende Punkte beachtet werden:

- Der Ableitungsprozess muss unumkehrbar und nicht-vorhersehbar sein, um sicherzustellen, dass die Kompromittierung eines abgeleiteten Schlüssels nicht den Ableitungsschlüssel oder andere abgeleitete Schlüssel kompromittiert.
- Bei einer Schlüsselableitung (im Sinne von [ISO-11770]) basiert die kryptographische Stärke der abgeleiteten Schlüssel auf der Ableitungsfunktion

und der kryptographischen Stärke des geheimen Ableitungsschlüssels (insbesondere hier dessen Entropie). Die Entropie der abgeleiteten Schlüssel ist kleiner gleich der Entropie des geheimen Ableitungsschlüssels. Um die Entropie der abgeleiteten Schlüssel sicherzustellen, muss die Entropie des geheimen Ableitungsschlüssels (deutlich) größer sein als die zu erreichende Entropie der abgeleiteten Schlüssel.

- Der Betreiber eines Schlüsseldienstes muss im Falle des Einsatzes einer Schlüsselableitung (nach [ISO-11770]) in seinem Sicherheitskonzept Maßnahmen für das Bekanntwerden von Schwächen des kryptographischen Verfahrens, welche die Grundlage der Schlüsselableitung ist, darlegen.

Ein Kartenherausgeber hat auch die Freiheit, gar kein Ableitungsverfahren zu verwenden, sondern alle symmetrischen SK.CMS aller seiner Karten sicher in seinem RZ vorzuhalten.

Ziel des Masterkey-Verfahrens zur Ableitung eines versichertenindividuellen Schlüssels ist es, aus einem geheimen Masterkey und einem öffentlichen versichertenindividuellen Merkmal einen geheimen symmetrischen Schlüssel abzuleiten, der zur Absicherung der Verbindung zwischen CMS und Smartcard verwendet wird. Öffentlich bedeutet an dieser Stelle nicht, dass die Merkmale selbst nicht schützenswert sind, es soll jedoch ausdrücken, dass die Vertraulichkeit des versichertenindividuellen Schlüssels nicht von der Geheimhaltung dieser Merkmale abhängt. Die Vertraulichkeit der Daten muss durch die Geheimhaltung des Masterkeys gewährleistet sein. Das bedeutet, die Geheimhaltung anderer Daten als des Masterkeys darf für die Vertraulichkeit der Daten nicht notwendig sein. Die Durchführung dieses Verfahrens muss bei gleichen Eingangsparametern immer das gleiche Ergebnis generieren.

Für die Durchführung des Algorithmus wird neben dem Masterkey auch noch mindestens ein versichertenindividuelles Merkmal verwendet. Die Auswahl des Merkmals ist fachlich motiviert und wird daher in diesem Dokument nicht spezifiziert. Das in Tabelle 20 beispielhafte Verfahren besteht aus einer Kombination von AES-Verschlüsselung [FIPS-197] und Hashwert-Bildung. Die Schlüssel- bzw. Hashwert-Länge ergibt sich gemäß Tabelle 21 .

Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen Schlüssels

Reihenfolge	Beschreibung	Formale Darstellung
1	Bildung eines Hashwertes über dem versichertenindividuellen Merkmal unter Verwendung eines statischen Padding-Verfahrens für den Fall, dass das versichertenindividuelle Merkmal in seiner Länge nicht der Blocklänge des Hash-Algorithmus entspricht. Im Ergebnis wird ein versichertenindividuelles Merkmal geeigneter Länge für den nächsten Schritt erzeugt.	$\text{HASH\#1} = \text{SHA-256}(\text{versichertenindividuelles Merkmal})$

Reihenfolge	Beschreibung	Formale Darstellung
2	AES-Verschlüsselung des Resultats mit dem Masterkey. Durch die Verschlüsselung an dieser Stelle ist sichergestellt, dass der versichertenindividuelle Schlüssel nur durch den Besitzer des geheimen Masterkeys erzeugt werden kann.	ENC#1 = AES-256(HASH#1)
3	Bildung eines Hashwertes über dem Ergebnis des vorherigen Verarbeitungsschritts. Dies stellt sicher, dass ein Schlüssel geeigneter Länge erzeugt wird.	Versichertenindividueller Schlüssel = SHA-256(ENC#1)

In der nachfolgenden Tabelle werden Kürzel entsprechend der Definition aus Abschnitt 3.2.3 verwendet.

Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines versichertenindividuellen Schlüssels

Algorithmen Typ	Algorithmus	Unterverfahren
Masterkey-Verfahren für die Generierung des versichertenindividuellen Schlüssel innerhalb eines CMS	AES basiertes Verfahren gemäß vorheriger Definition	AES-256 SHA-256 anwendbar bis Ende 2029+

3.5 Hybride Verschlüsselung binärer Daten

Für die hybride Verschlüsselung werden die Daten zunächst symmetrisch mittels eines zufällig gewählten geheimen symmetrischen Schlüssels verschlüsselt. Der geheime Schlüssel wird im Anschluss asymmetrisch für jeden Empfänger separat verschlüsselt.

Hinweis: unter binären Daten sind im gesamten Dokument beliebige Daten insbesondere beliebigen Typs (Text, HTML, PDF, JPG etc.) zu verstehen. Es gilt das Prinzip: das Spezielle vor dem Allgemeinen: gibt es weitere spezielle Vorgaben für bestimmte Datenformate, sind diese für die entsprechenden Daten verpflichtend (überschreiben oder ergänzen die allgemeinen Vorgaben).

3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten

GS-A_4389 - Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten
Produkttypen, die die hybride Verschlüsselung binärer Daten durchführen, MÜSSEN für den symmetrischen Anteil der Verschlüsselung die folgenden Vorgaben berücksichtigen:

- 1369 • Als symmetrische Block-Chiffre muss AES [FIPS-197] mit einer Schlüssellänge von
1370 256 Bit im Galois/Counter Mode (GCM) gemäß [NIST-SP-800-38D] mit der Tag-
1371 Länge von 128 Bit verwendet werden.
- 1372 • Die IVs dürfen sich bei gleichem Schlüssel nicht wiederholen (vgl. [NIST-SP-800-
1373 38D#S.25] und [BSI-TR-02102-1#S.24]). Der IV soll eine Bitlänge von 96 Bit
1374 besitzen, seine Länge muss mindestens 96 Bit sein. Es wird empfohlen den IV
1375 zufällig zu wählen (vgl. [gemSpec_Krypt#GS-A_4367]).
- 1376 • Hinweis: Im Normalfall ist davon auszugehen, dass für die Sicherung der
1377 Integrität und Authentizität der zu verschlüsselnden Daten zudem noch eine
1378 Signatur dieser Daten notwendig ist.

1379 [<=]

1380 *Hinweis: In [RFC-5084] findet man Informationen über die Verwendung von AES-GCM*
1381 *innerhalb von CMS [RFC-5652].*

1382 **3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer** 1383 **Daten**

1384 **GS-A_4390 - Asymmetrischer Anteil der hybriden Verschlüsselung binärer** 1385 **Daten**

1386 Produkttypen, die die hybride Verschlüsselung binärer Daten durchführen, MÜSSEN für
1387 den asymmetrischen Anteil der Verschlüsselung die folgenden Vorgaben berücksichtigen:

- 1388 • Als asymmetrisches Verschlüsselungsverfahren MUSS RSAES-OAEP gemäß
1389 [PKCS#1, Kapitel 7.1] verwendet werden.
- 1390 • Als Mask-Generation-Function für die Verwendung in RSAES-OAEP MUSS MGF 1
1391 mit SHA-256 als Hash-Funktion gemäß [PKCS#1, Anhang B.2.1] verwendet
1392 werden.

1393 [<=]

1394 **3.6 Symmetrische Verschlüsselung binärer Daten**

1395 **GS-A_5016 - Symmetrische Verschlüsselung binärer Daten**

1396 Produkttypen, die die symmetrische Verschlüsselung binärer Daten durchführen,
1397 MÜSSEN die folgenden Vorgaben berücksichtigen:

- 1398 • Als symmetrische Block-Chiffre muss AES [FIPS-197] mit einer Schlüssellänge von
1399 256 Bit im Galois/Counter Mode (GCM) gemäß [NIST-SP-800-38D] mit der Tag-
1400 Länge von 128 Bit verwendet werden.
- 1401 • Die IVs dürfen sich bei gleichem Schlüssel nicht wiederholen (vgl. [NIST-SP-800-
1402 38D#S.25] und [BSI-TR-02102-1#S.24]). Der IV soll eine Bitlänge von 96 Bit
1403 besitzen, seine Länge muss mindestens 96 Bit sein. Es wird empfohlen den IV
1404 zufällig zu wählen (vgl. [gemSpec_Krypt#GS-A_4367]).
- 1405 • Hinweis: Im Normalfall ist davon auszugehen, dass für die Sicherung der
1406 Integrität und Authentizität der übertragenen Daten zudem noch eine Signatur
1407 der zu verschlüsselnden Daten notwendig ist.

1408 [<=]

1409 *Hinweis: In [RFC-5084] findet man Informationen über die Verwendung von AES-GCM*
1410 *innerhalb von CMS [RFC-5652].*

1411 **A_15561 - AES-NI**

1412 Wenn der eingesetzte Konnektor AES-NI unterstützt und AES-NI dort aktiviert ist (vgl.
1413 [BSI-TR-03116-1#Abschnitt "4.7 Hardware-Unterstützung AES (AES-NI)"]), MUSS der
1414 Konnektor für alle AES-Ausführungen die AES-NI verwenden. [\leq]

1415 **3.7 Signatur binärer Inhaltsdaten (Dokumente)**

1416 **GS-A_5080-01 - Signaturen binärer Daten (Dokumente)**

1417 Alle Produkttypen, die CMS-Signaturen [RFC-5652] von Inhaltsdaten (wie bspw.
1418 Textdokumenten ungleich PDF/A) erzeugen oder prüfen, MÜSSEN die Algorithmen und
1419 Vorgaben der Tabelle Tab_KRYPT_020 erfüllen. [\leq]

1420

1421 **Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären**
1422 **Daten im Kontext von Dokumentensignaturen**

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Signaturstandard	Signaturstandard	ETSI TS 101 733 V1.7.4 (2008-07) Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAAdES) [ETSI-CAAdES]	Die Verwendung des Standards ist für die Signatur von Dokumenten verpflichtend die mittels CMS [RFC- 5652] erzeugt werden.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA- 256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+	Die Verwendung einer dieser Algorithmen ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256	Die Verwendung des Algorithmus ist verpflichtend.

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Kryptographisches Token	Kryptographisches Token für die Signatur, bestehend aus einem privaten Schlüssel und einem zugehörigen X.509-Zertifikat	Identitäten gemäß einem der folgenden Abschnitte 2.1.1.1 2.1.1.2	Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.

1423 3.8 Signaturen innerhalb von PDF/A-Dokumenten

1424 **GS-A_5081-01 - Signaturen von PDF/A-Dokumenten**

1425 Alle Produkttypen, die in PDF/A-Dokumenten [PDF/A-2] Signaturen einbetten/erzeugen
1426 oder diese Signaturen prüfen, MÜSSEN die Algorithmen und Vorgaben der Tabelle
1427 Tab_KRYPT_021 erfüllen.[<=]

1428

1429 **Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-**
1430 **Dokumentensignaturen**

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Signaturstandard	Signaturstandard	ETSI TS 102 778-3 V1.2.1, PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced – PAdES-BES and PAdES-EPES Profiles Technical Specification, 2010 [PAdES-3]	Die Verwendung des Standards ist für die Signatur von PDF/A [PDF/A-2] Dokumenten verpflichtend, die mittels eingebetteter Signaturen signiert werden.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA-256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+	Die Verwendung einer dieser Algorithmen ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256	Die Verwendung des Algorithmus ist verpflichtend.
Kryptographisches Token	Kryptographisches Token für die Signatur, bestehend aus einem privaten Schlüssel und einem zugehörigen X.509-Zertifikat	Identitäten gemäß einem der folgenden Abschnitte 2.1.1.1 2.1.1.2	Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.

1431 **3.9 Kartenpersonalisierung**

1432 Vgl. auch Abschnitt 2.4: Schlüsselerzeugung und Schlüsselbestätigung .

1433 **GS-A_4391 - MAC im Rahmen der Personalisierung der eGK**

1434 Der Herausgeber der eGK MUSS sicherstellen, dass bei der Personalisierung der eGK die
1435 Daten bei der Übermittlung integritätsgeschützt werden. Für die Absicherung der
1436 Integrität ist in diesem Kontext der AES-256 CMAC nach [NIST-SP-800-38B] (vgl. [BSI-
1437 TR-03116-1#3.2.2, 4.5.2]) zu verwenden.

1438 Die Länge des CMAC muss 128 Bit betragen.

1439 Nach [NIST-SP-800-38B#S.13] sollen nicht mehr als 2^{48} Nachrichtenblöcke (2^{22} GByte)
1440 mit demselben Schlüssel verarbeitet werden. Nach [NIST-SP-800-38B#S.14] ist ein
1441 CMAC anfällig für Replay-Attacken, was bei der Anwendung des CMACs zu
1442 berücksichtigen ist.

1443 [\leq]

1444 **3.10 Bildung der pseudonymisierten Versichertenidentität**

1445 **GS-A_4392 - Algorithmus im Rahmen der Bildung der pseudonymisierten 1446 Versichertenidentität**

1447 Alle Produkttypen, die pseudonymisierte Versichertenidentitäten berechnen, MÜSSEN den
1448 Hash-Algorithmus SHA-256 [FIPS-180-4] verwenden. [\leq]

1449 **3.11 Spezielle Anwendungen von Hashfunktionen**

1450 **GS-A_4393 - Algorithmus bei der Erstellung von Hashwerten von Zertifikaten 1451 oder öffentlichen Schlüsseln**

1452 Alle Produkttypen, die Fingerprints eines öffentlichen Schlüssels oder eines Zertifikates
1453 erstellen, MÜSSEN den Hash-Algorithmus SHA-256 [FIPS-180-4] dafür verwenden. [≤]

1454 Erläuterung:

1455 Alle CAs und der TSL-Dienst müssen im Rahmen ihrer Prozesse öffentliche Schlüssel oder
1456 Zertifikate (bspw. auf Webseiten) veröffentlichen. Dabei wird auch jeweils der SHA-256
1457 Hashwert mit veröffentlicht.

1458 Hersteller einer gSMC-KT müssen den Hashwert des auf der Karte befindlichen Zertifikats
1459 in MF/DF.KT/EF.C.SMKT.AUT.R2048 entweder auf dem ID-1-Kartenkörper drucken (das
1460 ID-000-Modul ist dann herausbrechbar) oder ausgedruckt mitliefern. Der Konnektor muss
1461 den Hashwert des Zertifikats bei initialen Pairing mit dem KT berechnen und dem
1462 Administrator präsentieren.

1463 Innerhalb der CertHash-Extension als Teil einer OCSP-Response wird vom TSP ein SHA-
1464 256 Hashwert des Zertifikats, über das eine Sperrinformation gegeben wird, mitgeliefert.

1465 **GS-A_5131 - Hash-Algorithmus bei OCSP/CertID**

1466 Alle Produkttypen, die OCSP-Anfragen stellen oder beantworten, MÜSSEN bei der
1467 Erstellung und Verwendung der CertID-Struktur (vgl. [RFC-6960, Abschnitt 4.1.1] oder
1468 [RFC-2560, Abschnitt 4.1.1]) den Hash-Algorithmus SHA-1 [FIPS-180-4] verwenden.
1469 Ein OCSP-Server KANN auch zusätzlich andere Hashfunktionen im Rahmen der CertID,
1470 die nach [BSI-TR-03116-1] zulässig sind, unterstützen.
1471 [≤]

1472 **3.11.1 Hashfunktionen und OCSP (informativ)**

1473 Es hat sich gezeigt, dass zum folgenden Themenkomplex eine Erläuterung hilfreich ist.

1474 Im Zusammenspiel OCSP-Anfrage und OCSP-Antwort werden an drei Stellen
1475 Hashfunktionen verwendet, die theoretisch alle paarweise verschieden sein können.

1476 **Erste Stelle:** Zunächst erzeugt ein OCSP-Client eine OCSP-Anfrage (vgl. [RFC-6960,
1477 Abschnitt 4.1.1] oder [RFC-2560, Abschnitt 4.1.1]). Dafür muss dieser u. a. eine CertID-
1478 Datenstruktur erzeugen:

```
1479     CertID          ::=      SEQUENCE {  
1480         hashAlgorithm      AlgorithmIdentifier,  
1481         issuerNameHash     OCTET STRING, -- Hash of issuer's DN  
1482         issuerKeyHash      OCTET STRING, -- Hash of issuer's public key  
1483         serialNumber       CertificateSerialNumber }
```

1484 Bei der Wahl der Hashfunktion kann er sich nur darauf verlassen, dass der OCSP-
1485 Responder als Hashalgorithmus (vgl. „hashAlgorithm“-Datenfeld) SHA-1 [FIPS-180-4]
1486 unterstützt. Für den Anfragenden und den OCSP-Responder gilt dementsprechend GS-
1487 A_5131. Er muss SHA-1 für die CertID-Struktur verwenden. Ein OCSP-Responder, der
1488 zusätzlich weitere Hashfunktionen unterstützt, muss nichts zurückbauen – er darf auch
1489 so in der TI arbeiten.

1490 Warum ist der Einsatz von SHA-1 an dieser Stelle kryptographisch gesehen ausreichend?
1491 Da (1) ein OCSP-Responder der TI nicht für beliebige CAs arbeitet (Wahl von DN und
1492 öffentlichen Schlüssel ist damit beschränkt) und (2) i. d. R. die CertHash-Extension Teil
1493 der OCSP-Antwort ist und innerhalb der CertHash-Extension in der TI eine
1494 kryptographisch hochwertige Hashfunktion verwendet wird, ist die Verwendung von SHA-
1495 1 hier aus Sicherheitsicht betrachtet unbedenklich. (Vgl. analoges Vorgehen BNetzA-
1496 OCSP-Responder für den qualifizierten Vertrauensraum.) Es ist also sichergestellt, dass

1497 zwischen OCSP-Client und -Responder keine (evtl. von einem Angreifer böswillig
1498 herbeigeführten) Unklarheiten darüber entstehen können über welches Zertifikat gerade
1499 gesprochen wird. Es geht bei GS-A_5131 vornehmlich um die Interoperabilität von OCSP-
1500 Client und OCSP-Responder.

1501 Die optionale Signatur einer OCSP-Anfrage wird in der TI nicht verwendet, damit ist die
1502 dort verwendete Hashfunktion für die aktuelle Betrachtung irrelevant.

1503 **Zweite Stelle:** Für die Beantwortung der OCSP-Anfrage erzeugt der OCSP-Responder u.
1504 a. eine CertHash-Datenstruktur:

```
1505         id-commonpki-at-certHash OBJECT IDENTIFIER ::= {1 3 36 8 313}
1506         CertHash ::= SEQUENCE {
1507             hashAlgorithm AlgorithmIdentifier, -- The identifier
1508             -- of the algorithm that has been used the hash value below.
1509             certificateHash OCTET STRING }
```

1510 Hierfür muss eine kryptographisch hochwertige (nach [BSI-TR-03116-1] zulässige)
1511 Hashfunktion verwendet werden. Normativ ist an dieser Stelle: „GS-A_4393 Algorithmus
1512 bei der Erstellung von Hashwerten von Zertifikaten oder öffentlichen Schlüsseln“.
1513 Spätestens an dieser Stelle können OCSP-Client und OCSP-Server sich sicher sein, ob sie
1514 über das gleiche Zertifikat sprechen.

1515 **Dritte Stelle:** Die OCSP-Response muss am Ende vom OCSP-Responder signiert werden.
1516 Dafür ist die Vorgabe aus Tab_KRYPT_002 „Signatur der OCSP-Response“ normativ,
1517 welche über die für die jeweiligen Zertifikate geltenden Anforderungen (bspw. GS-
1518 A_4357-02) angezogen werden.

1519 **3.12 kryptographische Vorgaben für die SAK des Konnektors**

1520 **GS-A_5071-01 - kryptographische Vorgaben für eine Signaturprüfung in der** 1521 **SAK-Konnektor**

1522 Die SAK des Konnektors MUSS bei der Prüfung von qualifizierten elektronischen
1523 Signaturen mindestens folgende Verfahren wie im Algorithmenkatalog [ALGCAT]
1524 benannt, unterstützen:

- 1525 • RSA
 - 1526 • SHA-256, SHA-384, SHA-512 nach FIPS-180-4 (März 2012) [FIPS-180-4]
1527 (jeweils Abschnitt 6.2, 6.7, 6.5 und 6.4 ebenda),
 - 1528 • RSASSA-PSS nach PKCS#1 (PKCS#1 v2.1: RSA Cryptographic Standard,
1529 14.06.2002) Abschnitt 8.1 und 9.1, - 1530 • RSASSA-PKCS1-v1_5 nach PKCS#1 (PKCS#1 v2.1: RSA Cryptographic
1531 Standard, 14.06.2002) Abschnitt 8.2 und 9.2, - 1532 • bei RSA muss ein Modulus zwischen 1976 bis 4096 Bit verwendbar sein,
- 1533 • ECDSA
 - 1534 • SHA-256 nach FIPS-180-4 (März 2012) [FIPS-180-4] (Abschnitt 6.2),
 - 1535 • ECDSA basierend auf E(F_p) (vgl. Technische Richtlinie 03111, Version 2.0)
1536 auf der Kurve P256r1 [RFC-5639].

1537 [**<=**]

1538 **3.13 Migration im PKI-Bereich**

1539 **GS-A_5079 - Migration von Algorithmen und Schlüssellängen bei PKI-Betreibern**

1540 Der Anbieter einer Schlüsselverwaltung MUSS neue Vorgaben zu Algorithmen und/oder
1541 Schlüssellängen der gematik nach einer vorgegebenen Übergangsfrist umsetzen. Nach
1542 Ablauf der Übergangsfrist MÜSSEN ausschließlich diese geänderten Parameter bei der
1543 Erzeugung von Zertifikaten verwendet werden. [\leq]

1544 **3.14 Spezielle Anwendungen von kryptographischen Signaturen**

1545 **GS-A_5207-01 - Signaturverfahren beim initialen Pairing zwischen Konnektor 1546 und eHealth-Kartenterminal**

1547 Alle Produkttypen MÜSSEN in Bezug auf das verwendete Signaturverfahren beim initialen
1548 Pairing zwischen Konnektor und eHealth-Kartenterminal folgende Vorgaben umsetzen:

- 1549 1. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine RSA-basierte
1550 Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret
1551 (ShS.AUT.KT vgl. [gemSpec_KT#2.5.2.1, 3.7.2.1]) RSASSA-PSS [PKCS#1] und
1552 SHA-256 verwendet werden.
- 1553 2. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine ECDSA-basierte
1554 Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret ECDSA
1555 [BSI-TR-03111] und SHA-256 verwendet werden

1556 [\leq]

1557 Erläuterung: Beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal wird
1558 vom Konnektor ein 16 Byte langes Geheimnis erzeugt, das bei späteren
1559 Verbindungsaufbauten zwischen Konnektor und KT im Rahmen eines Challenge-
1560 Response-Verfahrens ([gemSpec_KT#3.7.2]) verwendet wird. Dieses Geheimnis wird von
1561 der gSMC-KT des KT beim initialen Pairing signiert. Die Signatur wird vom KT zum
1562 Konnektor transportiert und dort vom Konnektor geprüft.

1563

1564 **GS-A_5340 - Signatur der TSL**

1565 Der TSL-Dienst MUSS für die Signatur der TSL das Signaturverfahren RSASSA-PSS
1566 [PKCS#1] verwenden mit dem XMLDSig-Identifizier „http://www.w3.org/2007/05/xmldsig-
1567 more#sha256-rsa-MGF1“ nach [RFC-6931, Abschnitt „2.3.10 RSASSA-PSS Without
1568 Parameters“]. [\leq]

1569 **3.15 ePA-spezifische Vorgaben**

1570 **3.15.1 Verbindung zur VAU**

1571 Der Begriff "vertrauenswürdige Ausführungsumgebung" (VAU) wird in
1572 [gemSpec_Aktensystem_ePAfueralle] eingeführt. Jeder ePA-Client muss mit jeder
1573 beliebigen VAU (egal von welchem Anbieter ePA-Aktensystem) kommunizieren können.
1574 Deshalb ist es für die Interoperabilität notwendig, das Kommunikationsprotokoll zwischen
1575 beiden Kommunikationspartnern zu definieren und dessen Verwendung zu fordern.

A_15549-01 - ePA-VAU-Client: Kommunikation zwischen ePA-Client und ePA-VAU

Ein ePA-Client MUSS bei der Kommunikation mit der VAU das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "VAU-Protokoll für ePA für alle"] verwenden.
Der Client einer VAU MUSS nach spätestens 24 Stunden das Aushandeln neuer AES-Verbindungsschlüssel erzwingen. Er MUSS zeitlich abgelaufene Verbindungsschlüssel bei sich sicher löschen. [\leq]

Hinweis: ein ePA-Frontend des Versicherten ist nach A_15872 (bzw. A_15873-*) [gemSpec_ePA_FdV] verpflichtet, das Zertifikat des Kommunikationspartners (VAU) zu prüfen (Kontext: Prüfung Authentizität des empfangene ECDH-Schlüssels). Nach A_15873-* (vgl. auch A_15874-*) in [gemSpec_ePA_FdV] muss dabei die TSL der TI Prüfungsgrundlage sein [gemSpec_ePA_FdV].

A_15547-01 - ePA-VAU: Kommunikation zwischen ePA-VAU und ePA-Client

Das ePA-Aktensystem MUSS sicherstellen, dass dessen VAU bei der Kommunikation mit dem ePA-Client das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "VAU-Protokoll für ePA für alle"] verwendet.

Die VAU MUSS sicherstellen, dass jeder ausgehandelte AES-Verbindungsschlüssel nach spätestens 24 Stunden sicher gelöscht wird.

Die VAU MUSS ein AUT-Zertifikat aus der Komponenten-PKI der TI besitzen (mit Rollenkennung-OID "oid_epa_vau") das Verbindungsparameter (vgl. A_24425-*) authentisiert. [\leq]

Hinweis: Das AUT-Zertifikat hat die VAU auch schon bei ePA 1.x und 2.x verwendet.

3.15.2 ePA-Aktensysteminterne Schlüssel

A_15745-01 - Betreiberschlüssel Aktensystem

Ein ePA-Aktensystem MUSS sicherstellen, dass

1. es zwei betreiberspezifische Schlüssel (BS / Masterkey für Daten und Masterkey für Befugnisse) gibt,
2. diese Schlüssel AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge (vgl. A_24645-* bezüglich Speicherung der Schlüssel) sind,
3. diese Schlüssel im Betrieb ausschließlich der VAU des entsprechenden ePA-Aktensystem zugänglich sind,
4. diese Schlüssel nur zur Schlüsselableitung nach einem in [gemSpec_Krypt#Abschnitt 2.4] zulässigen Verfahren verwendet werden,
5. es eine Schlüsselableitung mit diesen betreiberspezifischen Schlüsseln und einem aktenspezifischen Merkmal (bspw. der KVNR) gibt und daraus versichertenindividuelle Persistierungsschlüssel für die Daten und für die Befugnisse abgeleitet werden,
6. diese versichertenindividuelle Persistierungsschlüssel AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge sind,
7. diese Schlüssel ausschließlich mittels AES/GCM analog [gemSpec_Krypt#GS-A_4389] verwendet werden,
8. diese Schlüssel im Betrieb ausschließlich der VAU des entsprechenden ePA-Aktensystem zugänglich sind.

[\leq]

A_15746-01 - Sicherstellung der Verfügbarkeit der betreiberspezifischen Schlüssel

Ein ePA-Aktensystem MUSS sicherstellen, dass für die Sicherstellung der Verfügbarkeit der betreiberspezifischen Schlüssel (vgl. A_15745-*) eine sicherheitstechnisch geeignete Sicherung (Backup) des Schlüsselmaterials erzeugt und sicher verwahrt wird.

[<=]

A_16176-01 - Mindestvorgaben für ePA-Aktensystem-interne Schlüssel

Ein ePA-Aktensystem MUSS bei innerhalb des Aktensystems eingesetzten Schlüsselmaterial, das nicht aus der TI-PKI kommt (Signatur Authorisierungstoken etc.), folgende Vorgaben umsetzen:

1. Alle verwendeten nicht-TI-Schlüssel MÜSSEN ein Sicherheitsniveau von 120 Bit ermöglichen (vgl. [gemSpec_Krypt#5 "Migration 120-Bit Sicherheitsniveau"]).
2. Alle nicht-TI-RSA-Schlüssel MÜSSEN eine Mindestschlüssellänge von 3000 Bit besitzen.
3. Alle nicht-TI-ECC-Schlüssel MÜSSEN auf einem folgenden der Domainparametern (Kurven) basieren:
 - a. P-256 oder P-384 [FIPS-186-5],
 - b. brainpoolP256r1, brainpoolP384r1 oder brainpoolP512r1 [RFC-5639].

[<=]

Erläuterung: Ziel von A_15751 und A_16176-01 ist es, den Umstellungsbedarf im Rahmen der ECC-Migration der TI und ihrer Anwendungen in der Phase 2 zu minimieren.

A_20519-02 - Wechsel der betreiberspezifischen Schlüssel

Ein ePA-Aktensystem MUSS sicherstellen, dass

1. die betreiberspezifischen Schlüssel (BS) (vgl. A_15745-*) mindestens jährlich gewechselt werden,
2. wenn der Health Record Context einer Akte aufgrund eines Nutzerzugriffs aktiviert ist, alle auf Grundlage eines alten BS erzeugten Chiffre dieser Akte für den aktuellsten BS umgeschlüsselt werden, und
3. anschließend die mit einem alten BS erzeugten Chiffre dieser Akte sicher gelöscht werden.

[<=]

Hinweis: Die betreiberspezifischen Schlüssel (BS) dürfen gemäß A_15745-* ausschließlich der VAU des ePA-Aktensystems zugänglich sein. Daher muss die Umschlüsselung in der VAU stattfinden. Mehr Information zum Thema Umschlüsselung und Überschlüsselung findet man in [gemSpec_Aktensystem_ePAfuerAlle#3.6. Umschlüsselung und Überschlüsselung].

A_26250 - CMAC und Befugnisverifikation

Ein ePA-Aktensystem MUSS sicherstellen, dass

1. die symmetrischer Schlüssel für CMAC-Sicherung der Befugnisse 128 Bit AES-Schlüssel sind (vgl. [gemSpec_Aktensystem_ePAfuerAlle#3.3. Sichere Speicherung sensibler Schlüssel und Informationen im VAU-HSM]),
2. als CMAC-Verfahren das CMAC-Verfahren nach [NIST-SP-800-38B] mittels AES zum Einsatz kommt,

- 1663 3. die Ausgabelänge des CMAC von 128 Bit (= 16 Byte) ungekürzt im ePA-
1664 Aktensystem bei der Erstellung und der Prüfung eines CMAC-Wertes verwendet
1665 wird,
1666 4. die CMAC-Schlüssel regelmäßig (mindestens jährlich) gewechselt werden (vgl.
1667 Hinweise zu A_26250-*).

1668 [**<=**]

1669 Hinweis zu A_26250-*:

1670 Die CMAC-Schlüssel in VAU-Token-Modul, Befugnisverifikations-VAU o. Ä. werden
1671 verwendet um das Ergebnis von bestimmten rechenintensiven Prüfungen (VAU-
1672 Attestierung und HSM-ID-Token, Signatur-/Zertifikatsprüfungen, Prüfung von ID-Token
1673 etc.) für eine beschränkte Zeit im Aktensystem sicherheitstechnisch geeignet zu
1674 "cachen". Um einen Schlüsselwechsel (A_26250-* Punkt 4) zu erleichtern, ist es
1675 empfehlenswert den erstellten CMAC-Werten entsprechende Metadaten (Label des
1676 verwendeten CMAC-Schlüssels) beizufügen.

1677 **3.15.3 ePA-spezifische TLS-Vorgaben**

1678

1679 **A_15751-03 - TLS-Verbindung zwischen ePA-Aktensystem und ePA-Client**

1680 Ein ePA-Aktensystem und ein ePA-Client MÜSSEN in Bezug auf die TLS-Verbindung
1681 zwischen ihnen

- 1682 1. folgende Ciphersuiten unterstützen
- 1683 • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C),
 - 1684 • TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B).
- 1685 2. Sie KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1]
1686 unterstützen.
- 1687 3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch und bei der
1688 Signaturprüfung mittels ECDSA MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-
1689 5] unterstützt werden. Daneben KÖNNEN die Kurven brainpoolP256r1,
1690 brainpoolP384r1 oder brainpoolP512r1 (vgl. [RFC-5639] und [RFC-7027])
1691 unterstützt werden. Andere Kurven SOLLEN NICHT verwendet werden (Hinweis:
1692 die Intention des letzten Satzes ist insbesondere, dass die Ordnung des
1693 Basispunktes in $E(F_p)$ nicht zu klein werden darf).

1694 [**<=**]

1695 **A_26025 - ePA: TLS-Identitäten, IOP, P-256 Basierte ECC-Schlüssel**

1696 Ein ePA-Aktensystem MUSS sicherstellen, dass seine TLS-Identitäten (vgl. A_15751-*)
1697 auf der Kurve P-256 [FIPS-186-5] basieren. D. h., der EE-Schlüssel im TLS-Server-
1698 Zertifikat als auch das bestätigende CA-Zertifikat (Komponenten-PKI-CA-Zertifikat)
1699 MÜSSEN als öffentliche Schlüssel Kurvenpunkte auf der ECC-Kurve P-256 besitzen. [**<=**]

1700 Erläuterung:

1701 Ziel ist es die Interoperabilität zwischen Standard-TLS-Bibliotheken/Security-Providern
1702 auf den Primärsystemen und der TLS-Implementierung/Konfiguration ePA-Aktensystem
1703 im Kontext des TLS-Verbindungsaufbaus mit dem ePA-Aktensystem sicherzustellen. Es
1704 wird mit A_26025-* gefordert, dass die Kurvenparameter P-256 für die TLS-Server-
1705 Schlüssel der ePA-Aktensysteme verwendet werden.

1706 Ein Aktensystem-Betreiber erzeugt wie üblich die CSR für die TLS-Zertifikat des vom ihm
1707 betriebenen Aktensystems -- nun mit den ECC-Schlüsseln auf Basis von P-256 -- und
1708 übergibt diese per TMS an die Komponenten-PKI zur Zertifikatserstellung. Die
1709 Komponenten-PKI prüft den CSR und wählt automatisch eine Komponenten-CA, die auf
1710 P-256 basiert, als bestätigende Instanz (vgl. auch A_23139-*).

1711 **A_24913 - ePA: TLS-Verbindungen, OCSP-Stapling**

1712 Ein ePA-Aktensystem MUSS bei allen seinen ePA-spezifischen HTTPS-Schnittstellen (Rolle
1713 TLS-Server) TLS-OCSP-Stapling [RFC-6066] verwenden (aktivieren). Es MUSS
1714 sicherstellen, dass die im TLS-Handshake mit gesendeten OCSP-Responses max. 50
1715 Minuten alt sind. Sollte vom entsprechenden OCSP-Responder für den Bezug der OCSP-
1716 Responses trotz regelmäßigen Versuchs keine OCSP-Response bezogen werden
1717 können, so MUSS das Aktensystem die jüngste ihm zur Verfügung stehende OCSP-
1718 Response verwenden, und es regelmäßig weiter probieren (bspw. im 5'-Takt).

1719
1720 Ein ePA-Client MUSS in seiner TLS-Implementierung OCSP-Stapling unterstützen und die
1721 dort aufgeführten OCSP-Responses verwenden. Sollte diese zu alt sein (vgl.
1722 [gemILF_PS_ePA#A_24900]), so MUSS der ePA-Client versuchen, selbst OCSP-
1723 Responses einzuholen, wobei er die Client-seitiges OCSP-Response-Caching nach
1724 [gemSpec_PKI#A_23225] umsetzen MUSS. [\leq]

1725 Erläuterung:

1726 Die Erfahrungen in der PU haben in den letzten Jahren gezeigt, dass ohne dedizierte
1727 Maßnahmen wie OCSP-Stapling und Client-seitiges OCSP-Response-Caching eine zu hohe
1728 Last an den OCSP-Respondern der Komponenten-PKI erzeugt wird.

1729 **A_15833-01 - TLS-Verbindungen ePA-FdV**

1730 Ein ePA-Frontend des Versicherten MUSS die TLS-Vorgaben in A_15751-* bei allen
1731 seinen TLS-Verbindungen einhalten.
1732 [\leq]

1733 **A_21269-01 - ePA-Client: TLS-Session-Resumption**

1734 Ein ePA-Client SOLL TLS-Session-Resumption (per Session-ID oder per TLS-Session-
1735 Resumption per Session-Tickets) unterstützen. [\leq]

1736 **3.15.4 Zugriffscode-Erzeugung**

1737 Im Rahmen des Anwendungsfalls "Befugnis für einen EU-Zugriff erstellen"
1738 [gemSpec_ePA_FdV#Befugnisverwaltung EU-Zugriff] werden Zugriffscode (6 stellige
1739 maximal eine Stunde gültige Einmalpasswörter) erzeugt. Die Symbolmenge ist {a..z,
1740 A..Z, 0..9} also 62 mögliche Zeichen. Damit ist der Kardinalität der Menge aller
1741 möglichen Einmalpasswörter $62^6 = 56.800.235.584$.

1742 **A_26301 - ePA-Client: Zugriffscode Erzeugung**

1743 Ein ePA-Client, der Zugriffscode erzeugt (vgl. [gemSpec_ePA_FdV#Befugnisverwaltung
1744 EU-Zugriff]), MUSS folgendes sicherstellen:

- 1745 1. Er MUSS als Basis für die zufällige Zugriffscode-Erzeugung einen
1746 Zufallszahlengenerator gemäß GS-A_4367 verwenden.
- 1747 2. Er MUSS 6 Zeichen jeweils zufällig aus der Menge {a..z, A..Z, 0..9} auswählen.
- 1748 3. Er MUSS bei der zufälligen Auswahl die Sample&Reject-Strategie verwenden (vgl.
1749 Implementierungshinweise zu A_26301-*), um statistische Schiefen bei der
1750 Auswahl zu vermeiden.

1751 Die Aneinanderreihung der 6 zufällig ausgewählten Zeichen MUSS der "Zugriffscode"
1752 sein. [<=]

1753 Implementierungshinweis zu A_26301-*

1754 Ein häufig auftretendes Problem bei der zufälligen Auswahl von Elementen aus einer
1755 Menge ist, dass die Kardinalität der Auswahlmenge meist keine Zweierpotenz ist und die
1756 Zufallsgeneratoren immer nur Bitströme als Ausgabe erzeugen (also anders formuliert
1757 Zahlen zwischen 0 und einer Zweierpotenz). Meist geht man mit diesem Problem um
1758 indem man ein sogenanntes Sample&Reject-Verfahren verwendet. Man berechnet zu der
1759 Kardinalität der Auswahlmenge die nächst größere Zweierpotenz. Davon berechnet man
1760 den Logarithmus zur Basis 2.

1761 Beispiel:

```
1762 $ python  
1763 Python 3.9.16 (main, Mar 8 2023, 22:47:22)  
1764 >>> 26*2+10  
1765 62  
1766 >>> 62**6  
1767 56800235584  
1768 >>> import math  
1769 >>> math.ceil(math.log2(62))  
1770 6  
1771
```

1772 Für eine zufällige Auswahl eines Symbols entnimmt man der Zufallsquelle die
1773 entsprechende Anzahl der Bit aus der Zufallsquelle -- hier also 6 Bits. Damit erhält man
1774 eine Zufallszahl zwischen 0 und 63. Man tut dies (sampling) solange bis man eine Zahl
1775 zwischen 0 und 61 erhält, falls man 62 oder 63 erhält verwirft man diese (reject). Die
1776 Laufzeit der Auswahl wird damit nichtdeterministisch, was im Kontext Zugriffscode-
1777 Erzeugung unproblematisch ist, und man stellt mit diesem Vorgehen sicher, dass es eine
1778 keine statistischen Schiefen bei der zufälligen Auswahl gibt.

1779 **3.16 Anomalie-Erkennung**

1780 Für die betreiberübergreifende Anomalie-Erkennung werden in der VAU bestimmte
1781 Attribute (vgl. A 22496-*) pseudonymisiert aus der VAU exportiert (vgl.
1782 [gemSpec Aktensystem PAFueralle#"Logging und Monitoring]). Das Cyber-Defense-
1783 Center der TI kann bei der Feststellung von Anomalien, die einen Angriff als Ursache
1784 dringend vermuten lassen, die Pseudonyme depseudonymisieren, um weitere
1785 Maßnahmen zum Schutz der medizinischen Daten gezielt durchführen zu können.

1786 **A 27332 - ePA-Aktensystem - Pseudonymisieren der gematik-Logdaten**
1787 Das ePA-Aktensystem MUSS die in A 27331-* für die Pseudonymisierung
1788 gekennzeichneten Informationen der Logdaten mittels AES/CBC mit dem Schlüssel
1789 key_pn_log und dem festen Initialisierungsvektor IV=0...0 (16 Null-Bytes)
1790 verschlüsseln.
1791 Bei einer Verschlüsselung im CBC-Modus muss der zu verschlüsselnde Klartext gleich
1792 einem Vielfachen der Blocklänge der Blockchiffre sein. Dies ist bei den zu
1793 pseudonymisierenden Daten (DTBP) selten der Fall. Deshalb MUSS eine Kodierung der

DTBP vor der Verschlüsselung wie folgt stattfinden:

<u>Name</u>	<u>Länge</u>	<u>Erläuterung / Vorgaben</u>
<u>Längenfeld</u>	<u>8 Bytes</u>	<u>In diesem 64-Bit großen Längenfeld wird die Länge der DTBP in Network-Byte-Order (Byte-Order Big) kodiert.</u>
<u>DTBP</u>	<u>variabel</u>	<u>Die zu pseudonymisierenden Daten (DTBP) werden hier aufgeführt (Byte-Strom) der Länge "Längenfeld"</u>
<u>Padding</u>	<u>variabel</u>	<u>Das Padding besteht als Leerzeichen (chr(32)) und zwar so vielen, dass die Länge der Konkatination <Längenfeld> <DTBP> <Padding> ein Vielfaches der AES-Blocklänge (128 Bit = 16 Byte) ist.</u> <u>Damit kann es zwischen 0 und 15 Padding-Leerzeichen abhängig von der Länge der DTBP geben.</u>
<u>Leer-Block</u>	<u>16 Bytes</u>	<u>16 Leerzeichen (chr(32))</u>

Diese Kodierung wird Klartext (DTBE) genannt. Der Klartext wird per AES/CBC mit dem key pn_log und dem festen Initialisierungsvektor IV=0...0 (16 Null-Bytes) verschlüsselt und man erhält damit ein Chifftrat. Dieses Chifftrat MUSS base64 kodiert werden. Das Ergebnis (diese Kodierung) ist das Pseudonym von DTBP.

[<=]

Erläuterung zu A 27332-*: Durch die Verwendung eines konstanten Initialisierungsvektors (IV) ist die Verschlüsselung eine deterministische Verschlüsselung.

Im Normalfall haben die zu pseudonymisierenden Daten (DTBP) eine Länge von kleiner 256 Bytes. Um an dieser Stelle aber auf der sicheren Seite zu sein (weil bspw. in einer späteren Ausbaustufe einige DTBP länger sein könnten), wurde ein 64-Bit-Wert als Längenfeld definiert, so dass man physikalisch dieses Limit nicht erreichen kann.

Der Leer-Block am Ende dient als (in seiner Leistungsfähigkeit sicher beschränkter) Integritäts- und Authentitätsschutz. Damit kann ein Depseudonymisierer ermitteln ob ein Pseudonym gültig ist. Ein umfassenderer Integritäts- und Authentitätsschutz macht fachlich genau an dieser Stelle wenig Sinn, der Schutz muss über die Log-Daten in Ihrer Gesamtheit erfolgen.

Auf die explizierte Anführung einer Versionskennung des verwendeten Pseudonymisierungsschlüssel wurde verzichtet.

Die gematik stellt Beispiel-Code für die Erstellung von Pseudonymen nach A 27332-* bereit.

A 27392 - ePA-Aktensystem - Import eines Pseudonymisierungsschlüssels

Ein ePA-Aktensystem MUSS von der gematik für die entsprechende ePA-VAU verschlüsselte Pseudonymisierungsschlüssel importieren können. Dabei wird ein Export-Paket analog zu A 27276-* verwendet mit den gleichen kryptographischen Vorgaben

(A_27275-*). Die JSON-Struktur für den Import hat folgende Struktur:

```
{  
  "Schlüsseltyp": "Pseudonymisierungsschlüssel CDC",  
  "version": "<eine natürliche Zahl als String hier aufgeführt>",  
  "iat": "<YYYY-MM-DD>",  
  "encrypted key":  
    "01b07b90f7379b06fd4e3b406fafce16c73c8e95abfdae60219e779bb7d76cf802a75f9f7d  
    fac99e8f998b27f876dc5af910b685261f2ce38004fe638f25c5451f1b028c2b7c3d707cle5  
    dad3ed8b34bd4406284a43a15654f677b6ed7c7041cf7eaec59fa2093fc51de71dd4ad461dd  
    4a2d8e1f5d611a25bd99ac1129"  
}
```

Der Import eines neuen Pseudonymisierungsschlüssels überschreibt/ersetzt den älteren/vorhergehenden Pseudonymisierungsschlüssel in der VAU. D. h., nach dem Import wird also der neue Pseudonymisierungsschlüssel sofort für eine dem erfolgreichen Import folgende Pseudonymisierung verwendet.
Die Zeit in iat MUSS keine Konsequenz im VAU-HSM haben -- das Attribut dient nur zu organisatorischen Zwecken beim Import/Export-Prozess.

[<=]

3-163.17 E-Rezept-spezifische Vorgaben

Der Fachdienst E-Rezept besitzt zwei HTTPS-Schnittstellen, eine in der TI und eine im Internet.

A_21332-02 - E-Rezept: TLS-Vorgaben

Ein E-Rezept-FD, ein Apothekenverzeichnis, ein E-Rezept-Client und ein IDP MÜSSEN in Bezug auf die TLS-Verbindung zwischen ihnen

1. folgende Ciphersuiten unterstützen

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30),
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F),
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C),
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B).

2. Sie KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.

3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch und bei der Signaturprüfung mittels ECDSA MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-5] unterstützt werden. Daneben SOLLEN die Kurven brainpoolP256r1, brainpoolP384r1 oder brainpoolP512r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven SOLLEN NICHT verwendet werden (Hinweis: die Intention des letzten Satzes ist insbesondere, dass die Ordnung des Basispunktes in $E(F_p)$ nicht zu klein werden darf).

[<=]

Ähnlich wie bei der Anwendung ePA endet die TLS-Verbindung am E-Rezept-FD an der Webschnittstelle (Eingangspunkt). Ziel ist es die Code-Komplexität innerhalb der VAU so gering wie möglich zu halten (Trusted Computing Base), um eine ausreichende

1868 Sicherheitsanalyse des VAU-Programmcodes überhaupt erst möglich zu machen. Dafür
1869 werden die Probleme des TLS-Handlings, der Lastverteilung und des DoS-Schutzes auf
1870 Applikationsebene außerhalb der VAU an den Webschnittstellen des Fachdienstes E-
1871 Rezept bearbeitet. So kann sich der Programmcode in der VAU auf seine zentrale
1872 Aufgabe des Zugriffsschutzes der über die VAU einstellbaren und abholbaren E-Rezepte
1873 fokussieren.

1874 Um die Verbindungsstrecke zwischen Webschnittstelle und E-Rezept-VAU in Bezug auf
1875 Vertraulichkeit zu schützen, wird eine Verschlüsselung auf Anwendungsebene eingeführt.
1876 Bei ePA ist dies das VAU-Protokoll. Beim E-Rezept kann aufgrund der andersartigen
1877 Anwendungslogik in der E-Rezept-VAU ein einfacheres Sicherungsverfahren verwendet
1878 werden. Dieses ist in Abschnitt 6. - VAU-Protokoll für E-Rezept normativ definiert.

1879 **A_22698 - E-Rezept, Erzeugung des Nutzerpseudonyms LEI**

1880 Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

- 1881 1. Die VAU MUSS einen mindestens 120-Bit-Entropie-haltigen
1882 Pseudonymisierungsschlüssel erzeugen und zur Verwendung durch die VAU
1883 vorhalten.
- 1884 2. Dieser Pseudonymisierungsschlüssel MUSS ausschließlich durch die VAU
1885 verwendbar sein (Backups durch den Betreiber, welche durch ein Mehr-Augen-
1886 Prinzip geschützt werden, sind zulässig).
- 1887 3. Dieser Pseudonymisierungsschlüssel MUSS halbjährlich automatisch durch die
1888 VAU neu erzeugt (gewechselt) werden.
- 1889 4. Die VAU MUSS im Falle, dass der Nutzer eine LEI ist, die Telematik-ID des Nutzers
1890 ermitteln und dann mittels der HKDF nach [RFC-5869] auf Basis von SHA-256,
1891 dem geheimen Pseudonymisierungsschlüssel und der Telematik-ID ein 256 Bit
1892 langes LEI-Pseudonym erzeugen (d. h., Ausgabelänge der HKDF ist also 256 Bit
1893 (32 Byte), IKM (vgl. [RFC-5869] = PS, info (vgl. [RFC-5869]) = Telematik-ID, salt
1894 (vgl. [RFC-5869] = „" (leere Zeichenkette)).
- 1895 5. Die VAU MUSS das in (4) erzeugte LEI-Pseudonym zusammen mit den weiteren,
1896 für die Rohdatenlieferung definierten, Informationen an den äußeren E-Rezept-FD
1897 (!= VAU) weiter geben.

1898 [**<=**]

1899 Erläuterung zu A_22698-*:

1900 Der Pseudonymisierungsschlüssel kann auch nur in Software vorliegen – muss also nicht
1901 zwangsweise in einem HSM vorliegen.

1902 Für die Unterstützung von betrieblichen Prozessen soll dem E-Rezept-Projekt ein
1903 Überblick über die Anzahl der aktuell im Feld befindlichen Primärsystem-Versionen zur
1904 Verfügung gestellt werden. Der E-Rezept-FD übermittelt die Pseudonyme als Teil der
1905 Rohdatenlieferung an die gematik.

1906 **3.173.18 KOM-LE-spezifische Vorgaben**

1907 Bei KOM-LE werden E-Mail-Anhänge , deren Gesamtgröße 15 MiB überschreitet, separat
1908 symmetrisch verschlüsselt und das Chifftrat auf dem "Fachdienst Download-Server (KAS)"
1909 abgelegt. Das Chifftrat erhält eine ID, die aus dem Hashwert des Chifftrats gebildet wird.
1910 Dabei ist die in A_19644 festgelegte Hashfunktion zu verwenden. Der verwendete
1911 symmetrische Schlüssel und die Hashwert-Referenz sind dann Teil des Klartextes der

1912 verschlüsselten E-Mail-Nachricht (KOM-LE). Die Chiffre auf dem Download-Server
1913 (KAS) werden automatisch nach einer bestimmten im FD festgelegten Zeit gelöscht.

1914 **A_19644 - Hashfunktion für Hashwert-Referenzen beim Fachdienst Download-**
1915 **Server (KAS)**

1916 Ein KOM-LE-Client und der Fachdienst Download-Server (KAS) MÜSSEN bei der
1917 Erzeugung und Verwendung von Hashwert-Referenzen für Anhänge - die auf dem
1918 Fachdienst Download-Server (KAS) abgelegt werden - die Hashfunktion SHA-256 [FIPS-
1919 180-4] verwenden. [\leq]

1920 ~~3.183.19~~ **HMAC-Sicherung der Kryptographisch gesicherte VSDM-**
1921 **Prüfziffer ~~VSDM~~Version 1**

1922 **A_23460 - VSDM-Betreiber: HMAC-Schlüsselerzeugung**

1923 Ein Betreiber eines VSDM-Dienstes MUSS den HMAC-Sicherungsschlüssel für die
1924 kryptographische Sicherung der VSDM-Prüfziffern zufällig mit einer Länge von 256 Bit (= 32 Byte) und einer Mindestentropie von 120 Bit erzeugen. [\leq]

1926 Hinweis: es gelten die Anforderungen aus Abschnitt "2.2 Zufallszahlengeneratoren" (Güte der Zufallsquellen) auch für die VSDM-Betreiber (Anbietersteckbrief).

1928 **A_23461 - VSDM-Betreiber: HMAC-Verfahren**

1929 Ein Betreiber eines VSDM-Dienstes MUSS die HMAC-Sicherung der VSDM-Prüfziffern das
1930 HMAC-Verfahren aus [RFC-2104] mit der Hashfunktion SHA-256 (also nicht wie im RFC
1931 beschrieben mittels SHA-1) verwenden. Für das dabei zu verwendende geheime
1932 Schlüsselmaterial gilt A_23460-*. [\leq]

1933 Beispiel:

1934 Wenn der geheime HMAC-Schlüssel (hexdump)
1935 3a8e0064436bf2dbe7ca41ec6f1ed60beec083bc4100633281eb397cb294391c ist, so ist
1936 der HMAC-SHA-256-Wert gemäß A_23461-* auf die leere Bytefolge folgende Bitfolge
1937 (hexdump) 4c0a04f65d498113a5df2ab388d99d2c0bc6224a662b1ce529342745e7
1938 af414a

1939

1940 **A_23463 - VSDM-Betreiber: verschlüsselter Export des HMAC-Schlüssels für die**
1941 **E-Rezept-VAU**

1942 Ein Betreiber eines VSDM-Dienstes MUSS den HMAC-Sicherungsschlüssel mittels des
1943 ECIES-Verfahrens [SEC1-2009] für den Export an den E-Rezept-FD verschlüsseln und
1944 dabei folgende Vorgaben umsetzen

- 1945 1. Er MUSS ein ephemeres ECDH-Schlüsselpaar erzeugen und mit diesem und dem
1946 VAU-Schlüssel aus A_20160-* ein ECDH gemäß [NIST-800-56-A] durchführen.
1947 Das somit erzeugte gemeinsame Geheimnis ist Grundlage für die folgende
1948 Schlüsselableitung.
- 1949 2. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von
1950 SHA-256 verwenden.
- 1951 3. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in
1952 der Formulierung von [RFC-5869] info="ecies-vau-transport" .
- 1953 4. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key
1954 für die Verwendung von AES/GCM ableiten.

2001

2002 Die gematik stellt Beispiel-Code für die Erzeugung eines Export-Pakets bereit.

2003 **3.20 Kryptographisch gesicherte VSDM-Prüfziffer Version 2**

2004 Bei der kryptographischen Sicherung der VSDM-Prüfziffer Version 1 wird im VSDM-FD im
2005 Jahresrhythmus ein Geheimnis erzeugt, das als Grundlage für die kryptographische
2006 Sicherung der Integrität einer VSDM-Prüfziffer über einen HMAC dient. Für Version 2 der
2007 kryptographischen Sicherung der VSDM-Prüfziffer werden die wesentlichen Teile der
2008 Prüfziffer verteilungsgeschützt (verschlüsselt) und authentizitäts-/integritätsgeschützt.
2009 Dabei wird wie in der TI üblich AES/GCM als "Authenticated Encryption with Associated
2010 Data (AEAD)"-Verfahren verwendet. Damit werden die Klartext-Daten bei der AES/GCM-
2011 Verschlüsselung ebenfalls authentizitäts- und integritätsgeschützt (GMAC-
2012 Wert/Authentication-Tag). Deshalb kommt bei der Version 2 kein HMAC mehr bei der
2013 eigentlichen Sicherung der Prüfziffern Version 2 zur Anwendung sondern AES/GCM (inkl.
2014 GMAC).

2015 **A 27274 - VSDM-Anbieter: jährliche Erzeugung des gemeinsamen Geheimnisses**
2016 Ein Anbieter eines VSDM-Dienstes MUSS (min.) jährlich ein Geheimnis für die
2017 kryptographische Sicherung der VSDM-Prüfziffern zufällig mit einer Länge von 256 Bit (=
2018 32 Byte) und einer Mindestentropie von 120 Bit erzeugen. [\leq]
2019 [\leq]

2020 Hinweis: es gelten die Anforderungen aus Abschnitt "2.2 Zufallszahlengeneratoren" (Güte
2021 der Zufallsquellen) auch für die VSDM-Anbieter (Anbietersteckbrief).

2022 Die erzeugten Geheimnisse müssen für die prüfenden Systeme E-Rezept-VAU und ePA-
2023 Aktensystem-VAU (Plural) verschlüsselt (A 27275-*) überführt werden. Dafür gibt es im
2024 Kontext Prüfziffer Version 1 einen etablierten Prozess, bei dem die Authentizität und
2025 Integrität der verschlüsselten Geheimnisse sichergestellt wird. Dieser Prozess wird
2026 unverändert weitergeführt auch für den sicheren Transport der gemeinsamen
2027 Geheimnisse im Kontext Prüfziffer 2.

2028 **A 27275 - VSDM-Anbieter: verschlüsselter Export des gemeinsamen**
2029 **Geheimnisses für Prüfziffer prüfende Fachdienste-VAUs**

2030 Ein Anbieter eines VSDM-Dienstes MUSS das gemeinsame Geheimnis (vgl. A 27274-*)
2031 mittels des ECIES-Verfahrens [SEC1-2009] für den Export an die VAUs der prüfenden
2032 Fachdienste (E-Rezept-FD, ePA-Aktensysteme) verschlüsseln und dabei folgende
2033 Vorgaben umsetzen

- 2034 1. Er MUSS ein ephemeres ECDH-Schlüsselpaar erzeugen, auf der Kurve des EE-
- 2035 Schlüssels aus dem Verschlüsselungszertifikat des Empfängers (VAUENC) -- also
- 2036 P-256 oder brainpoolP256r1, und mit diesem und dem VAU-Schlüssel aus
- 2037 A 20160-* ein ECDH gemäß [NIST-800-56-A] durchführen. Das somit erzeugte
- 2038 gemeinsame Geheimnis ist Grundlage für die folgende Schlüsselableitung.
- 2039 2. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von
- 2040 SHA-256 verwenden.
- 2041 3. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in
- 2042 der Formulierung von [RFC-5869] info="ecies-vau-transport".
- 2043 4. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key
- 2044 für die Verwendung von AES/GCM ableiten.

5. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig erzeugen.
6. Er MUSS mit dem CEK und dem IV mittels AES/GCM den Klartext verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
7. Er MUSS das Ergebnis wie folgt kodieren: chr(0x01) || <32 Byte X-Koordinate vom öffentlichen Schlüssel aus "1." > || <32 Byte Y-Koordinate> || <12 Byte IV> || <AES-GCM-Chiffre> || <16 Byte AuthenticationTag> (vgl. auch Tab KRYPT ERP und folgende die Beispielverschlüsselung). Die Koordinaten sind (wie üblich) vorne mit chr(0) zu padden solange bis sie eine Kodierungslänge von 32 Byte erreichen.

[<=]

Hinweis: Die gematik stellt Beispiel-Code bereit.

A 27276 - VSDM: Export-Paket

Ein VSDM-FD MUSS bei der Erstellung der Export-Pakete für den Export der gemeinsamen Geheimnisse (vgl. A 27274-*) an die prüfenden Systeme (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) die gleiche Export-Paket-Struktur verwenden wie im Kontext Prüzziffer Version 1. [<=]

Beispiel für ein Export-Paket

```
{  
  "betreiberkennung": "X",  
  "version": "2",  
  "exp": "2025-07-31",  
  "encrypted_key":  
    "019cd8fd69893e0cca78284b73281cb5e6978f9ec69f8475e30da8709d582d1241188e5f11  
    ae14b68defcb28f55b279a61e2b0a03314a9d105c67089602c0904f76f0f90b93547f02078f  
    2c6c3d0469b6e43fe2e5a512c3594537184b15c9aaf4b77b7da792e2e1eaae92d812ddf7633  
    c8b6e9bfe3fa7ff67daedb1185",  
  "hmac_empty_string":  
    "b9cda130455534eca5c767d8e1a6e62ff896c2e4a3a02fd7515466f4de2eb0e6"  
}
```

Hinweis: Auch bei dem Export-Paket für Version 2 wird als Integritätssicherung des Exports ein HMAC -- wie bei Version 1 -- verwendet. Für die Sicherung der Prüzziffern selbst im Betrieb wird dann AES/GCM verwendet.

Die Betreiberkennungen werden durch die gematik zugewiesen, es handelt sich um Großbuchstaben A bis Z.

Die gematik stellt Beispiel-Code für die Erzeugung eines Export-Pakets gemäß A 27276-* bereit.

A 27356 - VSDM: explizite Angabe der Geheimnis-Version bei Erzeugung

Ein VSDM-FD MUSS es einem VSDM-FD-Anbieter ermöglichen:

1. bei der (im Regelfall jährlichen) Erzeugung des Geheimnisses die Schlüsselversion, die bei der Erzeugung zu verwenden ist, explizit und beliebig (außer die aktuell aktive Schlüsselversion) anzugeben, (Es ist also keine strenge Monotonie bei der Folge der Schlüsselversionen durchzusetzen.)
2. explizit und beliebig auszuwählen welche Schlüsselversion aktiv (zur Erzeugung von Prüzziffern) verwendet werden soll, und

Ein VSDM-FD MUSS als Offset für die Zeitkodierung (iat) bei der Erstellung der Prüfziffer Version 2 (A 27278-*) folgende Vorgabe verwenden:

offset- Name	Wert	Erläuterung
iat_offset	1735689600	offset für die Erzeugungszeit (iat) der Prüfziffer Version 2 Hinweis: Die Zeit "2025-01-01T00:00:00+00:00" (ISO-Format 8601) nach Unix-Zeit (UTC) konvertiert ist 1735689600.

[<=]

Erläuterung zu A 27323-*: Für die Kodierung von Daten in der Prüfziffer Version 2 stehen nur 18 Byte zur Verfügung. Deshalb wird die iat-Zeit nicht wie bei der Unix-Zeit üblich von 1.1.1970 00:00:00 (UTC) startend kodiert, sondern vor der Kodierung in der Prüfziffer die für die Kodierung notwendige Bitlänge durch Subtraktion eines entsprechenden Offsets reduziert.

Die Kodierungslänge von iat wird wie in A 27278-* definiert sogar nochmal um 3 Bit reduziert (r iat 8).

A 27352 - VSDM-Prüfziffer Version 2: Erzeugung von hcv

Ein den hcv-Wert (Hash Check Value) erzeugendes System (VSDM-FD oder Primärsystem) MUSS bei der Erzeugung des hcv-Wertes, wie folgt vorgehen:

1. Es sei VB gleich der Versicherungsbeginn
(UC AllgemeineVersicherungsdatenXML.Versicherter.Versicherungsschutz.Beginn, https://github.com/gematik/api-telematik/blob/OPB5/fa/vsds/Schema_VSD.xsd).
VB MUSS keine Leerzeichen enthalten. (siehe Erläuterungen nach A 27352-*)
2. Falls der Versicherte eine "StrassenAdresse" (vgl. XML-Schema) und darin ein nichtleeres Element "Strasse" besitzt, dann sei SAS gleich der Wert in diesem Element. Andernfalls sei SAS="" (leere Zeichenfolge). Ggf. Führende oder endende Leerzeichen MÜSSEN entfernt werden. Die Kodierung der Inhalte von "Strasse" MUSS ISO-8859-15 (Latin-9) sein. (siehe Erläuterungen nach A 27352-*)
3. Sei SHA-256 wie in [FIPS-180-4] definiert.
4. Sei H = SHA-256(VB || SAS).
5. Sei H 40 die ersten 5 Bytes (40 Bit) von H.
6. Von H 40 setzt man das MSBit im ersten Byte auf 0, dass Ergebnis sei H 40 0.

Der hcv-Wert ist gleich H 40 0.

[<=]

Erläuterungen zu A 27352-*: Die Versicherten-Daten müssen nach Spezifikation VSDM [gemSpec eGK Fach VSDM], in ISO-8859-15 (Latin-9) vom eGK-Personalisierer und vom VSDM-FD kodiert eingebracht werden ("Die persönlichen Versichertendaten PD [...]. Der zu verwendende Zeichensatz für die fachlichen Inhalte ist ISO8859-15."). Der Konnektor (genauer das VSDM-Fachmodul im Konnektor) verändert diese Kodierung

nicht. D. h. die Versicherten-Daten, die ein Primärsystem über ReadVSD erhält, sind schon in der (im Sinne von A 27352-*) "korrekten" Zeichenkodierung und müssen ohne Umkodierung für die Hashwert-Erzeugung verwendet werden.

Das VB-Datum ist nach https://github.com/gematik/api-telematik/blob/OPB5/fa/vsds/Schema_VSD.xsd in einer dort definierten Datentyps "VSD:ISO8601Date". Auszug aus der Definition

```
<xs:pattern value="\d{4}(0[0-9]|1[012])(0[0-9]|1[12][0-9]|3[01])"/>
```

Beispiele:

<u>VB</u>	<u>SAS</u>	<u>Data-to-be-hashed (hexdump)</u>	<u>H 40 0-Wert (hexdump)</u>
<u>20190212</u>	(leere Zeichenkette)	<u>3230313930323132</u>	<u>4885ee8394</u>
<u>19981123</u>	<u>Berliner Straße</u>	<u>31393938313132334265726c696e65722053747261df65</u>	<u>6545491d14</u>
<u>19841003</u>	<u>Angermünder Straße</u>	<u>3139383431303033416e6765726dfc6e6465722053747261df65</u>	<u>7cc49e7af4</u>
<u>20010119</u>	<u>Björnsonstraße</u>	<u>3230303130313139426af6726e736f6e73747261df65</u>	<u>186269e4f7</u>
<u>20040718</u>	<u>Schönhauser Allee</u>	<u>3230303430373138536368f66e68617573657220416c6c6565</u>	<u>353646b5c8</u>

A 27278 - VSDM-FD: Struktur einer Prüfziffer der Version 2

Ein VSDM-FD MUSS zunächst folgende innere Datenstruktur (Klartext) erstellen:

Name	Länge	Festlegungen und Erläuterung
<u>I Feld 1</u>	<u>5</u>	<p>In diesem Feld sind Sperrinformationen und ein gekürzter Hashwert kodiert.</p> <p>Sperrinformation: Falls die eGK ungültig/gesperrt ist, sei $S=128$, anderenfalls sei $S=0$.</p> <p>Hashwert: Der hcv-Wert MUSS wie in A 27352-* definiert berechnet werden. Und wird im Folgenden als H_{40_0} bezeichnet. Sei $H_{40_0}[0]$ das erste Byte von H_{40_0} und $H_{40_0}[1:]$ alle restlichen Bytes von H_{40_0}.</p> <p>Dann ist $I_Feld_1 = (H_{40_0}[0] \mid S) \parallel H_{40_0}[1:]$.</p> <p>(Erläuterung zum besseren Verständnis: das MSBit im ersten Byte von H_{40} wird auf 0 gesetzt (A 27352-*, Schritt 6) und man erhält H_{40_0}. Anschließend wird die Sperrinformation auf das erste Byte aufaddiert. D. h. wenn von I_Feld_1 das MSBit des ersten Bytes 1 ist, dann ist die eGK ungültig/gesperrt.)</p>
<u>r_iat_8</u>	<u>3</u>	<p>Sei iat die aktuelle Unix-Zeit (UTC) in Sekunden (also keine Nachkommastellen) im VSDM-FD zum Erzeugungszeitpunkt der Prüfziffer. Dann MUSS $r_iat_8 = (iat - iat_offset) \gg 3$</p> <p>Alle Zahlenwerte MÜSSEN in Network-Byte-Order (= Byte-Order Big) kodiert werden. Alle Zeiten sind wie bei der Unix-Zeit üblich UTC.</p> <p>Hinweis: für iat_offset vgl. A 27323-*.</p> <p>Beispiel: Wird die Prüfziffer um "2025-01-02T00:00:00+00:00" = 1735776000 erzeugt, dann ist $r_iat = (1735776000 - 1735689600) \gg 3 = 10800$</p>
<u>KVNR</u>	<u>10</u>	<u>10</u> Stellige KVNR, ASCII-kodiert (Beispiel: A123456789)

Der Klartext hat damit eine Länge von 18 Byte.

Name	Länge	
<u>Feld 1</u>	<u>1</u>	<p>In diesem Feld sind drei Informationen kodiert:</p> <p>(1) Kennzeichnung für Version 2 der Prüfziffer Sei $V = 128$.</p> <p>(2) Betreiberkennung Die Betreiberkennung (wie bspw. im Export-Paket (A 27276-*) übertragen) ist zunächst ein Buchstabe von 'A' bis 'Z'. Sei BK diese Betreiberkennung als ASCII-Zeichen. Sei $BK_D = BK - 65$ und sei $BK_D_4 = BK_D \ll 2$.</p> <p>(3) Geheimnis/Schlüssel-Version Sei SV gleich die Geheimnis/Schlüssel-Version, die für die Verschlüsselung des Klartextes (siehe Tabelle oben) verwendet wird. SV wird binär kodiert, bspw. wenn $SV = 2$ ist, so ist SV kodiert <code>\x02</code>. SV MUSS kleiner 4 sein (vgl. auch A 27356-*).</p> <p>Sei $Feld_1 = V + BK_D_4 + SV$</p> <p>Beispiel: Sei die Betreiberkennung gleich 'B' und die Schlüssel-Version gleich 2, dann ist BK_D_4 gleich 4. Und damit $Feld_1 = 128 + 4 + 2 = 134$.</p>
<u>Initialisierungsvektor für AES/GCM</u>	<u>12</u>	wie bei AES/GCM üblich MUSS der 96 Bit lange IV (= 12 Bytes) pro Verschlüsselung zufällig über eine kryptographisch hochwertige Zufallsquelle erzeugt werden
<u>eigentliches Chifftrat</u>	<u>18</u>	mittels AES/GCM verschlüsselter Klartext (innere Datenstruktur, s. o.) mit dem jüngsten aktivierten AES/GCM-Schlüssel (vgl. A 27286-*)
<u>AES/GCM Authentication-Tag (GMAC)</u>	<u>16</u>	<u>128-Bit Authentication-Tag, der bei der AES/GCM entsteht (berechnet wird)</u>

Die oben aufgeführte Datenstruktur hat die Gesamtgröße von 47 Byte. Diese Datenstruktur MUSS base64-kodiert werden. Das Ergebnis der Kodierung ist "die Prüfziffer Version 2". Deren Länge ist 64 Byte (Hinweis: gleiche Länge wie eine Prüfziffer Version 1).

[<=]

Hinweise zu A 27278-*: Am ersten Byte (Feld 1) kann man eine Prüfziffer Version 1 (beginnt immer mit Zeichen aus dem Intervall [0x4a, 0x5a]) und eine Prüfziffer Version 2 eindeutig unterscheiden. Wenn das erste Byte von Feld 1 kleiner als 128 ist, so muss es eine Prüfziffer der Version 1 sein, anderenfalls ist es eine Prüfziffer der Version 2.

Aktuell (Januar 2024) besitzen alle gemeinsamen Geheimnisse in den VSDM-Fachdiensten die Versionsnummer 2. Mit der nächsten regulären Erneuerung (Q2/Q3 2025, A 27274-*) wird die Versionsnummer 3 verwendet, usw..

Mit der Erzeugungsvorschrift und Kodierung von r_iat_8 kommt es mit dem 03.04.2029 um 10:42:07 (UTC) zum Zählerüberlauf bei r_iat_8. Dies ist also eine obere Schranke für die Verwendbarkeit der Prüfziffer Version 2.

Die gematik stellt Beispiel-Code für die Erstellung und Prüfung einer Prüfziffer bereit.

A 27299 - VSDM-Prüfziffer Version 2: prüfenden Systeme, Import der gemeinsamen Geheimnisse und AES/GCM-Schlüsselableitung

Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-FD-VAU, ePA-Aktensystem-VAU/VAU-HSM etc.) MUSS Export-Pakete gemäß A 27276-* importieren. Es werden dabei gemeinsame Geheimnisse gemäß A 27274-* importiert. Diese MUSS die im Export-Paket aufgeführte Betreiberkennung und Version zugeordnet werden. Mit dem gemeinsamen Geheimnis MUSS das prüfende System eine Schlüsselableitung gemäß A 27286-* durchführend und dem erhaltenen AES/GCM-Schlüssel die Betreiberkennung und Version des gemeinsamen Geheimnisses (also aus dem entsprechenden Import) zuordnen.

Anschließend MÜSSEN die AES/GCM über Betreiberkennung und Version (vgl. Kodierung der beiden Werte in einer Prüfziffer Version 2 in A 27278-*) im prüfenden System verfügbar/adressierbar sein.

Ein prüfendes System MUSS die Möglichkeit besitzen alte gemeinsame Geheimnisse und abgeleitete AES/GCM-Schlüssel (Kontext Prüfung Prüfziffer Version 2) im System per Administration zu löschen.

[<=]

Erläuterung: Bei ePA erfolgt die Administration (also auch die konfigurative Änderungen) der VAU-HSM im technisch durchgesetzten 4-Augen-Prinzip mit ePA-Aktensystem-Betreiber und gematik zusammen.

A 27342 - Konfigurationsvariable enforce_hcv_check

Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) MUSS eine Konfigurationsvariable `enforce_hcv_check` besitzen, die standarmäßig auf `false` gesetzt ist. **[<=]**

A 27279 - VSDM-Prüfziffer Version 2: Prüfung und Entschlüsselung

Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) MUSS folgende Prüfungen der Prüfziffer Version 2 vornehmen. Ergibt eine der Prüfungen ein nicht-positives Prüfergebnis, so MUSS die Prüfziffer als ungültig abgelehnt werden.

1. Prüfung: besitzt die Prüfziffer eine Länge von 64 Bytes.

2. Prüfung: kann die Prüfziffer erfolgreich base64-dekodiert werden.

Die nun erhaltene erfolgreich base64-dekodierte 47 Byte lange Bytefolge wird als `dtbc` (data to be checked) im Folgenden bezeichnet.

3. Prüfung: ist das erste Byte von dtbc größer als 128 (das MSBit ist also auf 1 gesetzt).
Hinweis: ansonsten handelt es sich um eine Prüfziffer Version 1 (für die Prüfung in A 27279-* also nicht geeignet).
4. Prüfung: gibt es im prüfenden System einen AES/GCM-Schlüssel mit der im ersten Byte von dtbc aufgeführter Betreiberkennung und aufgeführter Version (vgl. A 27299-* und A 27278-*).
5. Die folgenden 12 Byte in dtbc werden als IV bezeichnet. Die darauf folgenden 18 Bytes werden als ciphertext bezeichnet. Die darauf folgenden 16 Bytes werden als Authentication-Tag bezeichnet.
Prüfung: ist die AES/GCM-Entschlüsselung erfolgreich mittels des in Schritt 4 identifizierten AES/GCM-Schlüssels und mit IV, ciphertext, Authentication-Tag
D. h. gibt es gerade kein Symbol FAIL als Ergebnis der AES/GCM-Entschlüsselung -- die Authentizität und Integrität des Chiphertexts/Klartexts ist damit festgestellt.
Im folgenden wird der erfolgreich entschlüsselte Klartext betrachtet.
6. Prüfung: ist das MSBit im ersten Byte des Klartextes gleich 0 (ansonsten ist die eGK gesperrt).
7. Prüfung zeitliche Gültigkeit der Prüfziffer:
Sei, wie in A 27278-*, definiert r_iat 8 die Bytefolge von Byte-Offset 5 bis inkl. Byte-Offset 7 (also 3 Byte groß) aus dem Klartext. r_iat 8 MUSS im Network-Byte-Order (= Byte-Order Big) kodierte unsigned Zahl interpretiert werden.
Zunächst MUSS der Wert iat mit $iat = (r_iat \ll 3) + iat_offset$ (vgl. A 27323-*) berechnet werden.
Anschließend MUSS anwendungsspezifisch iat mit der aktuellen Zeit überprüft werden:
ePA: A 24573-* (20 Minuten Fenster)
E-Rezept: A 23451-* (30 Minuten Fenster)
8. Prüfung hcv:
Im folgenden werden die ersten 5 Byte des Klartextes als H 40 0 bezeichnet.
Wenn vom Primärsystem ein hcv-Wert übergeben wurde, prüfe ob H 40 0 gleich dem vom Primärsystem übergebenen hcv-Wert ist (vgl. A 24590-*).
Wenn vom Primärsystem kein hcv-Wert übergeben wurde: Wenn `enforce hcv check` (vgl. A 27342-*) auf true gesetzt ist, dann FAIL, anderen falls OK.
(Der hcv-Wert aus A 24590-* MUSS vor dem Vergleich base64-dekodiert werden.)
9. Die letzten 10 Byte des Klartextes werden als KVNR bezeichnet.
Prüfung: ist die KVNR aus dem Klartext gleich der KVNR, die im Anwendungskontext erwartet wird.

[<=]

Hinweis zu A 27279-*: Je nach Anwendung und Implementierung (ePA oder E-Rezept) werden Teile der Anforderung im VAU-HSM und in der VAU erbracht.

Nur als Verständnishinweis: bei Prüfschritt 8 (hcv-Wertprüfung) ist die Prüfreihefolge motiviert durch Abhängigkeiten im Umsetzungsplan. Erst wenn `enforce hcv check` auf True gesetzt wird, ist die Prüfung aus Sicherheitsicht effektiv.

Die gematik stellt Beispiel-Code für die Erstellung und Prüfung einer Prüfziffer bereit

2289 **3-193.21 spezifische TLS-Vorgaben für VSDM**

2290 Es gelten zunächst auch für VSDM u. a. die allgemeinen TLS-Anforderungen GS-A_4384-
2291 * und A_17124-*. Um die Umsetzung in den VSDM zu erleichtern wird auf die zwingende
2292 Unterstützung von NIST-Kurven beim ephemeren ECDH (TLS-Handshake) verzichtet --
2293 es können ausschließlich brainpool-Kurven verwendet werden.

2294 **A_23912 - VSDM: Unterstützung von NIST-Kurven beim ephemeren ECDH (TLS-
2295 Handshake)**

2296 Ein VSDM KANN auf die Unterstützung von NIST-Kurven beim ephemeren ECDH (TLS-
2297 Handshake) bei der Umsetzung von GS-A_4384-* und A_17124-* verzichten. Er MUSS in
2298 diesem Fall die in GS-A_4384-* und A_17124-* anderen aufgeführten ECC-Gruppen
2299 (brainpoolP256r1 und brainpoolP384r1) unterstützen.

2300 [**<=**]

2301 Erläuterung zu A_23912-*: Das "SOLL" in GS-A_4384-* und A_17124-* für die
2302 brainpool-Kurven wird mit A_23912 zu einem "MUSS", falls auf die Unterstützung von
2303 NIST-Kurven beim ephemeren ECDH (TLS-Handshake) im VSDM verzichtet wird.

2304 **A_23913 - Intermediär: TLS, Kurven beim ephemeren ECDH (TLS-Handshake)**

2305 Ein Intermediär MUSS bei der Umsetzung von GS-A_4384-* und A_17124-* die Kurven
2306 brainpoolP256r1 und brainpoolP384r1 beim ephemeren ECDH (TLS-Handshake)
2307 unterstützen. [**<=**]

2308 Erläuterung: Durch A_23913 wird beim Intermediär dass "SOLL" in GS-A_4384-* und
2309 A_17124-* zu einem "MUSS".

4 Umsetzungsprobleme mit der TR-03116-1

Das u. a. durch die TR-03116-1 [BSI-TR-03116-1] angestrebte Sicherheitsniveau soll persönliche medizinische Daten effektiv schützen. Dazu lehnt sie sich an die sehr starken kryptographischen Vorgaben für die qualifizierte elektronische Signatur [SOG-IS] an. Einige Formate (bspw. XMLDSig) oder Implementierungen (bspw. Standard-Java-Bibliotheken) können einige Vorgaben von Hause aus nicht erfüllen.

Dieses Kapitel weist auf Umsetzungsprobleme hin (ehemals Kapitel 3.3 aus dem Kryptographiekonzept des Basis-Rollouts).

4.1 XMLDSig und PKCS1-v2.1

Mit [XMLDSig] allein ist aktuell keine Nutzung von RSASSA-PSS [PKCS#1] möglich.

Aus diesem Grund hat die gematik entschieden für die Signatur nach [XMLDSig] zusätzliche Identifier für RSASSA-PSS aus [RFC-6931] innerhalb der TI zu verwenden, welche auf der Lösung aus [XMLDSig-RSA-PSS] basieren. Der RFC-6931 [RFC-6931] ist die Aktualisierung von [RFC-4051]. Die in Abschnitt „2.3.9 RSASSA-PSS With Parameters“ und „2.3.10 RSASSA-PSS Without Parameters“ aufgeführten Identifier für RSASSA-PSS-Signaturen müssen innerhalb von XMLDSig für solche Signaturen verwendet werden.

GS-A_5091 - Verwendung von RSASSA-PSS bei XMLDSig-Signaturen

Produkttypen, die RSASSA-PSS-Signaturen [PKCS#1] innerhalb von XMLDSig erstellen oder prüfen, MÜSSEN die Identifier aus [RFC-6931] Abschnitt „2.3.9 RSASSA-PSS With Parameters“ und „2.3.10 RSASSA-PSS Without Parameters“ für die Kodierung dieser Signaturen verwenden.

[<=]

Ein Beispiel aus [RFC-6931] Abschnitt „2.3.10 RSASSA-PSS Without Parameters“:

```
<SignatureMethod
  Algorithm=
    "http://www.w3.org/2007/05/xmlencsig-more#sha256-rsa-MGF1"
/>
```

Vgl. [gemSpec_COS, (N003.000)]: Die Hashfunktion, auf der die Mask-generation-function basiert, ist SHA-256 [FIPS-180-4]. Die Länge des salt ist gleich der Ausgabelänge eben jener Hashfunktion (= 256 Bit).

4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM

Bei der Verschlüsselung mittels XMLEnc [XMLEnc] gibt es zwei Probleme in Bezug auf fehlende Identifier für kryptographische Verfahren, die in Abstimmung mit dem BSI für den Einsatz in der TI notwendig sind.

- Für die symmetrische Verschlüsselung mittels AES-GCM ([FIPS-197], [NIST-SP-800-38D]) gibt es keine Algorithmen-Identifier innerhalb von [XMLEnc]. Solche gibt es in [XMLEnc-1.1, Abschnitt 5.2.4].

- 2348 • Für die Kodierung von RSA-OAEP-Chiffreten innerhalb von [XMLEnc] fehlt in
2349 [XMLEnc] ein Identifier für RSAES-OAEP mit der MGF1 basierend auf SHA-256
2350 (vgl. auch Kapitel 5.10 „MGF Mask Generation Function“ in [gemSpec_COS]).
2351 Einen solchen Identifier („<http://www.w3.org/2009/xmlenc11#mgf1sha256>“) gibt
2352 es in XMLEnc Version 1.1 [XMLEnc-1.1, Abschnitt 5.5.2].

2353 Aus diesem Grund hat die gematik entschieden für die XML-Verschlüsselung die
2354 Vorgaben aus [XMLEnc-1.1] zu verwenden.

2355 **4.3 XML Signature Wrapping und XML Encryption Wrapping**

2356 Komplexität ist der natürliche Feind von Sicherheit. Die unter dem Sammelbegriff XML
2357 betitelten Formate und Protokolle sind sehr flexibel und leistungsfähig, aber auch sehr
2358 komplex. Noch dazu sind Sicherheitsmechanismen in diesem Bereich zum Teil
2359 nachträglich beigelegt worden und sind damit oft weniger leistungsfähig als im CMS-
2360 Bereich. XML-Daten effektiv zu schützen ist aktives Forschungsthema [XMLEnc-CM],
2361 [XSpRES]. Öfter als in anderen Bereichen werden neue Schwachstellen bekannt
2362 [BreakingXMLEnc], [XSW-Attack].

2363 Aus diesem Grunde wird bei einer Sicherheitsevaluierung gesondert auf derartige Angriffe
2364 geachtet. Die gematik beobachtet neue Entwicklungen im Bereich der XML-Sicherheit und
2365 leitet falls notwendig Maßnahmen ein.

2366 **4.4 Güte von Zufallszahlen**

2367 Nach dem Kerckhoffs'schen Prinzip von 1883 [Ker-1883] darf die Sicherungsleistung von
2368 kryptographischen Verfahren allein auf der Geheimhaltung der geheimen oder privaten
2369 Schlüssel beruhen. Geheimhaltung inkludiert insbesondere, dass sie nicht erraten werden
2370 können. Wenn bei einer Schlüsselerzeugung zu wenig Entropie vorhanden ist, kann die
2371 Geheimhaltung nicht gewährleistet werden. Die kryptographischen Verfahren, welche mit
2372 diesen Schlüsseln dann arbeiten, können die von ihnen verlangten Sicherheitsleistungen
2373 nicht mehr erbringen. Aus diesem Grunde verlangt [BSI-TR-03116-1] eine Mindestgüte
2374 der Zufallszahlerzeugung u. a. bei einer Schlüsselerzeugung. Die Basis für die
2375 Beurteilung der Güte stellt [AIS-20] und [AIS-31] dar.

2376 Aktuell sind nicht alle Produkte in der TI bez. dieser Mindestgüte bewertet worden. Davon
2377 sind Smartcards nicht betroffen, da diese eine Sicherheitsevaluierung/-zertifizierung
2378 durchlaufen haben, bei der die Güte der Zufallszahlenerzeugung positiv beurteilt wurde.
2379 Probleme bereiten insbesondere HSMs.

2380 Neben einer möglichen Common-Criteria-Zertifizierung dieser Produkte, bei der analog zu
2381 den Smartcards die Güte geprüfte wird, gibt es weitere mögliche Lösungen:

- 2382 1. gesonderte Prüfung der Güte nach [AIS-20] und [AIS-31] ohne komplette
2383 Common-Criteria-Zertifizierung,
2384 2. Herstellererklärung über die Güte (wie sie bspw. aktuell bei der Kartenproduktion
2385 üblich ist).

2386

5 Migration 120-Bit-Sicherheitsniveau

2387 Das „Sicherheitsniveau eines kryptographischen Verfahrens“ ist definiert als der
2388 Logarithmus zur Basis 2 der Anzahl der „Rechenschritte“ die notwendig sind um ein
2389 kryptographisches Verfahren mit hoher Wahrscheinlichkeit zu brechen. Was als
2390 „Rechenschritt“ definiert ist, ist vom Verfahren abhängig. Das Sicherheitsniveau wird in
2391 Bit angegeben. Beispielsweise nimmt man aktuell an, dass für das Brechen einer AES-
2392 Chiffre mit 128 Bit Schlüssellänge rund $2^{126,4}$ Rechenschritte, die der Durchführung einer
2393 AES-Verschlüsselung (eines 128-Bit Eingabeblocks) entsprechen, im Mittel notwendig
2394 sind. Somit erreicht eine AES-128-Bit-Verschlüsselung maximal ein Sicherheitsniveau von
2395 ca. 126,4 Bit. Eine RSA-2048-Bit-Verschlüsselung erreicht ein Sicherheitsniveau von ca.
2396 100 Bit.

2397 Für die TI ist ab Ende 2025 ein Sicherheitsniveau von mindestens 120 Bit für alle
2398 kryptographischen Verfahren zu erreichen. Daher ist bis dahin eine Migration aller
2399 Komponenten und Dienste notwendig, die kryptographische Verfahren mit
2400 Schlüssellängen bez. Domainparametern verwenden, die nur ein Sicherheitsniveau von
2401 unter 120 Bit erreichen können.

2402 Aufgrund der höheren Performanz, insbesondere in Chipkarten und Embedded-Geräten,
2403 wird nicht auf RSA-3072-Bit sondern auf ECDSA mit 256-Bit-Schlüsseln migriert.

2404 Es gibt Produkttypen, die kryptographische Verfahren so einsetzen, dass diese keine
2405 direkten Wechselwirkungen bei anderen Produkttypen besitzen. Beispielsweise werden
2406 von einem ePA-Aktensystem Autorisierungstoken (inkl. Signatur) erzeugt und diese
2407 werden von einem ePA-FdV oder als FM ePA als opakes Objekt behandelt. Dabei kann
2408 weiterhin RSA verwendet werden, solange die dabei verwendeten Schlüsselgrößen
2409 mindestens 3000 Bit betragen (Sicherheitsniveau 120-Bit erzielen) (A_16176-01). Ggf.
2410 ist es empfehlenswert dennoch auf ECC-basierte Verfahren zu migrieren (schnellere
2411 Ausführungsgeschwindigkeit, geringere Signaturgröße).

2412 Die Migration erfolgt schrittweise und Komponenten und Dienste werden zusätzlich mit
2413 Schlüsselmateriale und Zertifikaten auf Basis von ECDSA auf der Kurve brainpoolP256r1
2414 ausgestattet werden. Es gibt bis maximal Ende 2025 (vgl. Abschnitt 2.1.1.1) einen
2415 Parallelbetrieb in der TI.

2416 Nachdem die X.509-Root der TI (Produkttyp „gematik Root-CA“), die TSPs der TI und die
2417 Objektsysteme der Chipkarten um ECC-Unterstützung für X.509-Identitäten erweitert
2418 wurden, erfolgt die schrittweise und parallele Unterstützung dieser Identitäten nun in
2419 weiteren Produkttypen bzw. Fachanwendungen.

2420 5.1 PKI-Begriff Schlüsselgeneration

2421 In [gemKPT_PKI_TIP#3.2] wird der Begriff der Schlüsselgeneration eingeführt. Eine CA
2422 signiert Zertifikate im abstrahierten Sinne mit „ihrem Signaturschlüssel“. Dieser Schlüssel
2423 wird regelmäßig neu erzeugt und solange Verfahren und Schlüssellänge bzw.
2424 Domainparameter gleichbleiben, handelt es sich um eine neue Schlüsselversion.
2425 Kryptographisch betrachtet wurde der neue Signaturschlüssel zufällig (vgl. GS-A_4368)
2426 erzeugt, ist also kryptographisch unabhängig vom alten Signaturschlüssel, und die CA
2427 arbeitet mit mehreren kryptographischen Schlüsseln.

2428 Beispiel: im Fall der X.509-Root der TI (vgl. Abschnitt 5.2) wird ihr Signaturschlüssel im
2429 Regelfall alle zwei Jahre neu erzeugt (vgl. GEM.RCA1 und GEM.RCA2,
2430 <https://download.tsl.ti-dienste.de/>). Der Signaturschlüssel liegt hier in zwei Versionen
2431 vor. Beide Schlüssel kommen aus der Schlüsselgeneration „RSA“.

2432 Für die Migration muss ein Signaturschlüssel in der X.509-Root der TI erzeugt werden,
2433 der aus der Schlüsselgeneration „ECDSA“ stammt. Für ihn gelten die Vorgaben aus
2434 [gemSpec_Krypt#GS-A_4357-02, Schlüsselgeneration „ECDSA“].

2435 5.2 X.509-Root der TI

2436 Die X.509-Root der TI (Produkttyp: gematik Root-CA) ermöglicht es über eine klassische
2437 PKI-Baumstruktur die meisten Zertifikate der TI zu prüfen. Für zukünftige Anwendungen,
2438 die nur mit erhöhten Kosten das leistungsstarke, aber auch deutlich komplexere TSL-
2439 Modell auswerten können, ist sie eine Infrastrukturleistung der TI, so wie auch die CVC-
2440 Root.

2441 Die X.509-Root muss für die Migration ECDSA-basierte Zertifikate für TSPs ausstellen
2442 können. Aufgrund von [gemSpec_PKI#GS-A_5511] muss die X.509-Root der TI neben
2443 dem Signaturschlüssel für die Schlüsselgeneration „RSA“ auch einen Signaturschlüssel für
2444 die Schlüsselgeneration „ECDSA“ gemäß GS-A_4357-02 (brainpoolP256r1) erzeugen, und
2445 diesen verwenden können.

2446 Als Hilfestellung wird im Folgenden ein X.509-Root-TI-Zertifikat betrachtet. Gemäß GS-
2447 A_4357-02 muss der öffentliche ECDSA-Schlüssel der Schlüsselgeneration „ECDSA“ auf
2448 der Kurve brainpoolP256r1 liegen. Sei

2449 `d=SHA-256 („gemSpec_Krypt-Beispiel X.509-Root-TI ECDSA-Schlüssel“)`
2450 `=0x62e50dca4da29b0b10ead635a20b51fb1ec281d11f90cde8b5a9d92371ae8052`

2451 Dieses `d` wird als Ganzzahl (Little-Endian) interpretiert und dies sei der für das Beispiel
2452 maßgebliche private Schlüssel. Damit ergibt sich folgender öffentlicher Punkt auf der
2453 Kurve brainpoolP256r1:

2454 `(0x377434509adcb827f74acd7adf0ce72aa28ddc53be3f15ea8023a9b0722c09d,`
2455 `0x5364a99686c02092bbf9efde9878847b90f09d90b7ac4193553820258a58dfd5)`

2456 Folgend ist die ASN.1-DER-Kodierung des Schlüssels, so wie sie sich später auch im
2457 Zertifikat befindet, aufgeführt:

2458 `MFowFAYHKoZIzj0CAQYJKyQDAWIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgCOPsH`
2459 `IsCdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39U=`

2460

```
2461     0  90: SEQUENCE {
2462     2  20: SEQUENCE {
2463     4   7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
2464    13   9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2465     :   }
2466    24  66: BIT STRING
2467     :    04 37 74 34 50 9A DC BB 82 7F 74 AC D7 AD F0 CE
2468     :    72 AA 28 DD C5 3B E3 F1 5E A8 02 3A 9B 07 22 C0
2469     :    9D 53 64 A9 96 86 C0 20 92 BB F9 EF DE 98 78 84
2470     :    7B 90 F0 9D 90 B7 AC 41 93 55 38 20 25 8A 58 DF
2471     :    D5
2472     :   }
```

2473 Das selbstsignierte Beispiel-Root-Zertifikat im PEM-Format:

```
2474 -----BEGIN CERTIFICATE-----
2475 MIICajCCAq+gAwIBAgIBATAKBggqhkJOPQQDAjBtMQswCQYDVQQGEwJERTEVMBMG
2476 A1UECgwMZ2VtYXRpayBHbWJIMTQwMgYDVQLDCTaZW50cmFsZSBSb290LUNBIGN1
2477 ciBUZWxlbWFOaWtpbmZyYXN0cnVrdHVyMREwDwYDVQDDAhHRU0uUkNBMAAeFw0x
2478 NjEyMDkwODQxNTZaFw0yNjEyMDcwODQxNTZaMG0xCzAJBgNVBAYTAkRFRMRUwEwYD
2479 VQKDAxNzW1hdGlrIEdtYkgxNDAYBgNVBAsMK1plbnRyYWx1IFJvb3QtQ0EgZGVy
2480 IFRlbGVtYXRpa2luZnJhc3RydWt0dXlxEtAPBgNVBAMCEdFTS5SQ0EzMFowFAYH
2481 KoZIZj0CAQYJKyQDAwIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgC
2482 OpsHIScdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39WjgZ4wgZswHQYD
2483 VR0OBByEFBERSneTkJZDKt3uLzjddI870TmMEIGCCsGAQUFBwEBBDYwNDAYBggr
2484 BgEFBQcwAYYmaHR0cDovL29jc3Aucm9vdC1jYS50aS1kaWVuc3RlLmRlL29jc3Aw
2485 DwYDR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwFQYDVR0gBA4wDDAKBgqq
2486 ghQATASBIzAKBggqhkJOPQQDAgNjADBGAiEApQ6qGHTx97IsdzgoWH9/W32yt4rk
2487 udUis0xxGZ48YOUCIQCTQ4puo15YyIAZYk74mfid3JBOvMBV/XgPV2WpS/99yg==
2488 -----END CERTIFICATE-----
```

2489 Relativ am Anfang des Zertifikats befindet sich die OID gemäß GS-A_4357-02

```
2490 16 10: SEQUENCE {
2491 18 8: OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
2492 : }
```

2493 Ab Offset 280 befindet sich der schon o. g. öffentlicher Schlüssel:

```
2494 282 90: SEQUENCE {
2495 284 20: SEQUENCE {
2496 286 7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
2497 295 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2498 : }
2499 306 66: BIT STRING
2500 : 04 37 74 34 50 9A DC BB 82 7F 74 AC D7 AD F0 CE
2501 : 72 AA 28 DD C5 3B E3 F1 5E A8 02 3A 9B 07 22 C0
2502 : 9D 53 64 A9 96 86 C0 20 92 BB F9 EF DE 98 78 84
2503 : 7B 90 F0 9D 90 B7 AC 41 93 55 38 20 25 8A 58 DF
2504 : D5
2505 : }
```

2506 Und am Ende des Zertifikats befindet sich die ECDSA-Signatur:

```
2507 535 10: SEQUENCE {
2508 537 8: OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
2509 : }
2510 547 73: BIT STRING, encapsulates {
2511 550 70: SEQUENCE {
2512 552 33: INTEGER
2513 : 00 A5 0E AA 18 74 F1 F7 B2 2C 77 38 28 58 7F 7F
2514 : 5B 7D B2 B7 8AE4 B9 D5 22 B3 4C 71 19 9E 3C 60
2515 : E5
2516 587 33: INTEGER
2517 : 00 93 43 8A 6E A2 5E 58 60 80 19 62 4E F8 99 F8
2518 : 9D DC 90 4E BC C0 55 FD 78 0F 57 65 A9 4B FF 7D
2519 : CA
2520 : }
2521 : }
2522 : }
```

2523 Wenn das oben aufgeführte Zertifikat sich in der Datei "root.pem" befindet, so kann man
2524 bspw. mittels

```
2525 openssl verify -check_ss_sig root.pem
```

2526 die Signatur überprüfen und erhält als Ausgabe:

```
2527 root.pem: C = DE, O = gematik GmbH, OU = Zentrale Root-CA der
2528 Telematikinfrastruktur, CN = GEM.RCA3
```

2529 error 18 at 0 depth lookup:self signed certificate
2530 OK

2531 **5.3 TSL-Dienst und ECDSA-basierte TSL allgemein**

2532 Durch die ECC-Migration dürfen bereits produktiv betriebene Komponenten und Dienste
2533 in ihrer Verfügbarkeit nicht gefährdet werden. Aus diesem Grunde wird es eine zweite
2534 TSL "TSL(ECC-RSA)" geben. Diese wird mittels ECDSA (brainpoolP256r1) signiert sein
2535 und RSA- und ECDSA-basierte CA-Zertifikate enthalten. Bis zum Abschluss der ECC-
2536 Migration wird es zwei TSL in der TI geben: die seit Beginn des Online-Betriebs der TI
2537 bestehende RSA-basierte "TSL(RSA)" und die ECDSA-basierte "TSL(ECC-RSA)". Die
2538 beiden TSL sind technisch unabhängig voneinander (Kontext Sequenznummern etc.).
2539 Dementsprechend wird es in Bezug auf die ECC-Migration keinen
2540 Vertrauensankerwechsel im Sinne von [ETSI_TS_102_231_v3.1.2] geben. Die
2541 Vertrauensbeziehung zwischen den beiden durch die zwei TSL beschriebenen
2542 Vertrauensräumen wird über den klassischen Mechanismus der Cross-Zertifizierung
2543 realisiert. Die RSA-basierte X.509-TSL-Signer-CA wird ein X.509-Cross-Zertifikat "für" die
2544 ECDSA-basierte X.509-TSL-Signer-CA der TI ausstellen (vgl. [\[gemSpec_PKI#A1_7689\]](#))
2545 und vice versa.

2546 Analog zur VL der BNetzA wird es die Möglichkeit geben vom Downloadpunkt des TSL-
2547 Dienstes "TSL(ECC-RSA)" einen Hashwert der aktuellen TSL zu erhalten und damit für
2548 das Prüfen der Aktualität der lokal gespeicherten TSL nicht immer die gesamte TSL vom
2549 Downloadpunkt neu laden zu müssen (vgl. [\[gemSpec_TSL#A_17682\]](#)).

2550 **A_17205 - Signatur der TSL: Signieren und Prüfen (ECC-Migration)**
2551 Alle Produkttypen, die die TSL(ECC-RSA) signieren oder prüfen, MÜSSEN dafür das
2552 Signaturverfahren ECDSA [BSI-TR-03111] auf Basis der Domainparameter
2553 brainpoolP256r1 verwenden mit dem XMLDSig-Identifizier „
2554 <http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256>“ [XMLDSig]. Als Hashfunktion
2555 (Messagedigest) MUSS SHA-256 [FIPS-180-4] verwendet werden.
2556 [\leq]

2557 **5.4 ECC-Unterstützung bei TLS**

2558 Das TLS-Protokoll unterstützt die Verwendung von RSA- und ECC-basierten Cipher-
2559 Suiten.

2560 Als Beispiel soll sich ein Konnektor mit ECC-Unterstützung mit einem "alten" eHealth-
2561 Kartenterminal (das also nur GS-A_4359-* und nicht A_17124-* kennt) verbinden. Beim
2562 Verbindungsaufbau (TLS-ClientHello) gibt der TLS-Client (Konnektor) eine geordnete
2563 Liste von unterstützenden Cipher-Suiten an. Der TLS-Server (eHealth-KT) untersucht
2564 diese Liste von vorn nach hinten und wählt die erste auch von ihm unterstützte Cipher-
2565 Suite. Somit gilt:

- 2566 1. Ein TLS-Client kann durch die von ihm gewählte Reihenfolge in der Liste der
2567 Cipher-Suiten angeben, welche Cipher-Suite der Client präferiert (bspw. ECC-
2568 basierte Cipher-Suiten).
- 2569 2. Ein TLS-Client und ein TLS-Server können unterschiedliche Fähigkeiten besitzen
2570 (ECC-Unterstützung Ja/Nein). Solange sie eine gemeinsame Schnittmenge

2571 besitzen (in unserem Fall RSA-basierte Cipher-Suiten), können sie miteinander
2572 eine TLS-Verbindung aufbauen.

2573 Ein TLS-Verbindungsaufbau eines Konnektors mit ECC-Unterstützung unterscheidet sich
2574 inhaltlich nur durch 4 zusätzliche Bytes (c02bc02c, vgl. GS-A_5354-* und A_23226-*)
2575 von einem TLS-Verbindungsaufbau ohne ECC-Unterstützung.

2576 Verbindet sich beispielsweise ein "alter" Konnektor im Rahmen von VSDM mit einem
2577 Intermediär mit Option "ECC-Migration", wählt der Intermediär nach GS-A_4384-* eine
2578 RSA-basierte Cipher-Suite. Der Verbindungsaufbau kommt zu Stande und der Konnektor
2579 wird quasi nie erfahren, dass der Intermediär ebenfalls ECC-basierte Cipher-Suiten
2580 unterstützt. Erst wenn der Konnektor per Firmware-Upgrade sozusagen mit der Option
2581 "ECC-Migration" ausgestattet wird, muss er u. a. A_17124-* Spiegelstrich 3 umsetzen.
2582 Danach wird der Intermediär nach A_17124-* Spiegelstrich 4 die erste ECC-basierte
2583 Cipher-Suite bei einem TLS-Verbindungsaufbau auswählen.

2584 **A_17124-03 - TLS-Verbindungen (ECC-Migration)**

2585 Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die folgenden
2586 Vorgaben erfüllen:

- 2587 1. Zur Authentifizierung MUSS eine X.509-Identität gemäß [gemSpec_Krypt#GS-
2588 A_4359-*] verwendet werden.
- 2589 2. Als Ciphersuiten MÜSSEN TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
2590 (0xC0,0x2B) und TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
2591 (0xC0,0x2C) unterstützt werden.
- 2592 3. Falls der Produkttyp in der Rolle als TLS-Client agiert, so MUSS er die eben
2593 genannten Ciphersuiten gegenüber evtl. ebenfalls von ihm unterstützen RSA-
2594 basierte Ciphersuiten (vgl. GS-A_4384-*) bevorzugen (in der Liste "cipher_suites"
2595 beim ClientHello vorne an stellen, vgl. [RFC-5246#7.4.1.2 Client Hello]).
- 2596 4. Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE"
2597 im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5]
2598 unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1
2599 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in
2600 A_17124-* aufgeführt DÜRFEN NICHT verwendet werden.

2601 [**<=**]

2602 **A_17775 - TLS-Verbindungen Reihenfolge Ciphersuiten (ECC-Migration)**

2603 Alle Produkttypen, die Übertragungen mittels TLS durchführen und in der Rolle TLS-
2604 Server agieren, SOLLEN die Reihenfolge der Ciphersuiten in der Liste "cipher_suites" aus
2605 dem TLS-ClientHello bei der Auswahl der Ciphersuite befolgen.

2606 [**<=**]

2607 Die meisten Software-Pakete oder TLS-zentrierten Hardware-Lösungen (TLS-
2608 Terminatoren etc.) unterstützen die (wie oft formuliert) "Honorierung" der Reihenfolge
2609 aus der Liste "cipher_suites", aber nicht alle. Deshalb und weil die Honorierung wichtig
2610 aber nicht absolut notwendig ist, wurde A_17775 als SOLL-Anforderung formuliert.

2611 **A_17322 - TLS-Verbindungen nur zulässige Ciphersuiten und TLS-Versionen 2612 (ECC-Migration)**

2613 Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN sicherstellen,
2614 dass sie nur (durch andere Anforderungen) zugelassene TLS-Ciphersuiten bzw. TLS-
2615 Versionen anbieten bzw. verwenden.

2616 [**<=**]

2617 Hinweis: Im Rahmen der Zulassungstests und der CC-Evaluierung wurde dies (A_17322)
2618 stets so umgesetzt. Mit A_17322 soll dieses Vorgehen explizit auch auf
2619 Spezifikationsebene ausgesprochen und transparent gemacht werden.

2620 **5.5 ECC-Unterstützung bei IPsec**

2621 Das IKE-Protokoll [RFC-7296] wird verwendet um Schlüsselmateriale auszuhandeln für die
2622 folgende Verschlüsselung und Integritätssicherung der über IPsec geschützten IP-Pakete.
2623 Auszuhandeln bedeutet, dass ein (elliptische Kurven) Diffie-Hellman -Schlüsselaustausch
2624 durchgeführt wird. Im Gegensatz zum TLS-Protokoll Version 1.2 trägt schon die erste
2625 Protokollnachricht des Initiators (IKE_SA_INIT) einen (EC)DH-Schlüssel, evtl. aus einer
2626 kryptographischen Gruppe, die der Responder nicht unterstützt. Im Gegensatz zu TLS
2627 Version 1.3 kann dabei genau nur ein (EC)DH-Schlüssel übertragen werden, nicht eine
2628 Auswahl von Schlüsseln aus verschiedenen Gruppen. Der Initiator (Konnektor) kann im
2629 Normalfall nicht wissen, ob der Responder (VPN-Konzentrator) einen ECC-basierten DH-
2630 Schlüsselaustausch unterstützt. Der Initiator versucht es einfach und beginnt die IKE-
2631 Schlüsselaushandlung mit folgender Nachricht

```
2632 Initiator                               Responder
2633 -----
2634 HDR, SAI1, KEi, Ni -->
```

2636 [RFC-7296]. In KEi ist der ephemere ECDH-Schlüssel auf Grundlage der
2637 Domainparameter brainpoolP256r1 enthalten. Falls der Responder diese
2638 Domainparameter (ECC-Kurve) nicht unterstützt, antwortet der Responder mit
2639 einer INVALID_KEY_PAYLOAD-Nachricht, in der eine vom Responder unterstützte und
2640 präferierte kryptographische Gruppe angegeben ist [RFC-7296#Abschnitt 1.2]. Somit
2641 kommt es bei einem initialen Verbindungsaufbau zwischen einen "neuen" Konnektor und
2642 einem "alten" VPN-Zugangsdienst zu einem zusätzlichen "roundtrip", was akzeptiert wird,
2643 weil dies die Schlüsselaushandlung und damit den folgenden Verbindungsfall im
2644 Normalfall nur unwesentlich verzögert. Ein "neuer" Konnektor, der ggf. solch eine
2645 INVALID_KEY_PAYLOAD-Nachricht erhält, wird dann auf die Vorgaben GS-A_4382-*
2646 "zurückfallen".

2647 **A_22342 - Konnektor, IKE-Schlüsselaushandlung – Erleichterung** 2648 **Migrationsphase 1 (ECC-Migration)**

2649 Solange ein Konnektor nur mit einem RSA-Zertifikat am VPN-Zugangsdienst registriert
2650 ist, KANN der Konnektor den IKE-Verbindungsaufbau gemäß der Vorgaben aus GS-
2651 A_4382-* durchführen.[<=]

2652 **A_22343-01 - Verwendung von ECC beim Verbindungsaufbau nach RE-** 2653 **Registrierung mit ECC-NK-Zertifikat (ECC-Migration)**

2654 Sobald der Konnektor mit einem ECC-Zertifikat am VPN-Zugangsdienst registriert ist,
2655 MUSS er den nächsten regulären Verbindungsaufbau zum VPN-Konzentrator gemäß der
2656 Vorgaben aus A_17125 durchführen.[<=]

2657 Analog zum TLS-Protokoll wählt der Responder die Cipher-Suite und ein "alter"
2658 Konnektor kann nicht erkennen, dass es sich evtl. um einen "neuen" VPN-Zugangsdienst
2659 handelt.

2660 **A_17125 - IKE-Schlüsselaushandlung für IPsec (ECC-Migration)**

2661 Alle Produkttypen, die die Authentifizierung, den Schlüsselaustausch und die
2662 verschlüsselte Kommunikation im IPsec-Kontext durchführen, MÜSSEN die

- 2663 Schlüsselvereinbarung mittels IKEv2 [RFC-7296] gemäß den folgenden Vorgaben
2664 durchführen:
- 2665 1. Zur Authentisierung MUSS eine Identität mit einem X.509-Zertifikat gemäß
2666 [gemSpec_Krypt#GS-A_4360-*] Schlüsselgeneration "ECDSA" verwendet werden.
 - 2667 2. Für „Hash und URL“ MUSS SHA-1(vgl. [RFC-7296#3.6]) verwendet werden.
 - 2668 3. Für den Schlüsselaustausch MUSS ein ephemeres ECDH verwendet werden. Dabei
2669 MUSS die Kurve brainpoolP256r1 [RFC-6954] unterstützt werden. Es KÖNNEN die
2670 Kurven brainpoolP384r1, brainpoolP512r1 [RFC-6954] und ECP Gruppen 19, 20
2671 und 21 [RFC-5903] unterstützt werden.
 - 2672 4. Als Verschlüsselungsverfahren im Rahmen von IKE MUSS AEAD_AES_128_GCM
2673 und AEAD_AES_256_GCM [RFC-5282] unterstützt werden (IANA.-Nr. 20)
2674 (Hinweis verpflichtend Unterstützung nach [RFC-5282#3.2]). Es MÜSSEN zudem
2675 AEAD_AES_128_GCM_12 und AEAD_AES_256_GCM_12 (IANA-Nr. 19) unterstützt
2676 werden. Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.2.1 Tabelle 2]
2677 unterstützt werden.
 - 2678 5. Als PRF für die Schlüsselerzeugung MUSS PRF_HMAC_SHA2_256 (IANA-Nr. 5)
2679 [RFC-4868] unterstützt werden. Es KÖNNEN weitere Verfahren nach [TR-02021-
2680 3#3.2.2 Tabelle 3] unterstützt werden.
 - 2681 6. Als Authentisierungsverfahren MUSS ECDSA-256 als Basis von brainpoolP256r1
2682 (IANA-Nr. 14) [RFC-7427] unterstützt werden. Es KÖNNEN weitere Verfahren
2683 nach [TR-02021-3#3.2.5 Tabelle 6] unterstützt werden.
 - 2684 7. Für die Verschlüsselung der ESP-Pakete MUSS AES-GCM mit 16 Byte großem ICV
2685 (IANA-Nr. 20) und AES-GCM mit 12 Byte großem ICV (IANA-Nr. 19) [RFC-4106]
2686 jeweils mit 128 und 256 Bit Schlüssellänge unterstützt werden.
2687 Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt
2688 werden.
 - 2689 8. Falls weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt werden,
2690 so MUSS mindestens ein Verfahren zum Integritätsschutz der ESP-Pakete
2691 aus [TR-02021-3#3.3.2 Tabelle 8] unterstützt werden. (Hinweis: bei den
2692 verpflichtend zu unterstützenden AEAD-Verfahren aus Spiegelstrich 7 ist ein
2693 zusätzlicher Integritätsschutz by-design nicht notwendig.)
- 2694 [**<=**]
- 2695 Hinweis:
- 2696 "strongSwan" unterstützt diese Algorithmen,
2697 vgl. <https://wiki.strongswan.org/projects/strongswan/wiki/IKEv2CipherSuites>
- 2698 **A_17126 - IPsec-Kontext -- Verschlüsselte Kommunikation (ECC-Migration)**
2699 Alle Produkttypen, die mittels IPsec-Daten schützen, MÜSSEN dies ausschließlich auf
2700 Grundlage der in A_17125 (und ggf. GS-A_4382-* vgl. diesbezüglich A_17210) als
2701 zulässig aufgeführten Verfahren und Vorgaben tun.
2702 [**<=**]

2703 5.6 ECDSA-Signaturen

2704 5.6.1 ECDSA-Signaturen im XML-Format

2705 **A_17206 - XML-Signaturen (ECC-Migration)**

2706 Alle Produkttypen, die XML-Signaturen auf Basis eines ECC-Schlüssels erzeugen oder
2707 prüfen, MÜSSEN dafür das Signaturverfahren ECDSA [BSI-TR-03111] auf Basis der
2708 Domainparameter brainpoolP256r1 verwenden. Sie MÜSSEN dabei den XMLDSig-
2709 Identifier „ <http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256>“ [XMLDSig
2710 verwenden. Als Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4]
2711 verwenden.

2712 [\leq]

2713 Die Anforderung A_17206 gilt für allgemeine XML-Datensignaturen, also auch für
2714 Tokensignaturen etc. A_17360 fordert für die Interoperabilität bei der Prüfbarkeit von
2715 Dokumentensignaturen die Verwendung des interoperablen Containerformats nach
2716 [ETSI-XAdES].

2717 **A_17360 - XML-Signaturen (Dokumente) (ECC-Migration)**

2718 Alle Produkttypen, die XML-Signaturen von Dokumenten auf Basis eines ECC-Schlüssels
2719 erzeugen oder prüfen, MÜSSEN dabei die Vorgaben aus A_17206 umsetzen und die
2720 Signatur nach [ETSI-XAdES] (interoperables Container-Format) bei der Erzeugung
2721 kodieren bzw. bei der Prüfung auswerten.

2722 [\leq]

2723 5.6.2 ECDSA-Signaturen im CMS-Format

2724 **A_17207 - Signaturen binärer Daten (ECC-Migration)**

2725 Alle Produkttypen, die (nicht-XML-)Signaturen von Daten auf Basis eines ECC-Schlüssels
2726 erzeugen oder prüfen, MÜSSEN dafür das Signaturverfahren ECDSA [BSI-TR-03111] auf
2727 Basis der Domainparameter brainpoolP256r1 verwenden (vgl. [RFC-5753] und [RFC-
2728 6090]). Als Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4]
2729 verwenden.

2730 [\leq]

2731 Die Anforderung A_17207 gilt für allgemeine (nicht-XML-)Datensignaturen, also auch für
2732 Tokensignaturen etc. A_17359 fordert für die Interoperabilität bei der Prüfbarkeit von
2733 Dokumentensignaturen die Verwendung des interoperablen Containerformats nach
2734 [ETSI-CAAdES].

2735 **A_17359 - Signaturen binärer Daten (Dokumente) (ECC-Migration)**

2736 Alle Produkttypen, die (nicht-XML-)Signaturen von Dokumenten auf Basis eines ECC-
2737 Schlüssels erzeugen oder prüfen, MÜSSEN dabei die Vorgaben aus A_17207 umsetzen
2738 und die Signatur nach [ETSI-CAAdES] (interoperables Container-Format) bei der
2739 Erzeugung kodieren bzw. bei der Prüfung auswerten.

2740 [\leq]

2741 Hinweis: Signaturen in PDF/A-Dokumenten werden mittels CMS kodiert.

2742 **A_17208 - Signaturen von PDF/A-Dokumenten (ECC-Migration)**

2743 Alle Produkttypen, die (nicht-XML-)Signaturen auf Basis eines ECC-Schlüssels erzeugen
2744 oder prüfen, MÜSSEN dafür das Signaturverfahren ECDSA [BSI-TR-03111] auf Basis der
2745 Domainparameter brainpoolP256r1 nach [PAAdES-3] und [PDF/A-2] verwenden. Als

2746 Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4] verwenden.
2747 [\leq]

2748 **5.7 ECIES**

2749 In der TI wird für die ECC-basierte Ver- und Entschlüsselung das "Elliptic Curve
2750 Integrated Encryption Scheme (ECIES)" verwendet. Es ist das einzige ECC-basierte, von
2751 den Chipkarten der TI unterstützte, Verschlüsselungsverfahren. Das ECIES ist ein
2752 hybrides Verfahren basierend auf [ABR-1999]. Es besteht aus einem asymmetrischen Teil
2753 (elliptic curve diffie hellman) und einen symmetrischen Teil (Verschlüsselungsverfahren
2754 und MAC-Verfahren). Weiterhin ist eine Schlüsselableitungsfunktion für das Verfahren
2755 notwendig. In [gemSpec_COS#6.8.2.3 Asymmetrische Entschlüsselung mittels ELC] wird
2756 definiert, welche Varianten dieser drei notwendigen Verfahren eine Chipkarte der TI
2757 unterstützt (ECDH [BSI-TR-03111#4.3.1 Key Agreement Algorithm], HKDF mittels SHA-
2758 256 und einem Zähler nach X9.63 [BSI-TR-03110-3#A.2.3.2], AES-256-CBC und CMAC).
2759 Da im Normalfall immer für eine Identität, die Chipkarten-basiert ist, verschlüsselt wird,
2760 muss ein Sender genau diese Verfahren einsetzen. Ansonsten kann die Chipkarte das
2761 Chifftrat nicht entschlüsseln, auch wenn die Chipkarte den prinzipiell richtigen privaten
2762 Schlüssel in sich trägt.

2763 Da man das gesamte Chifftrat für eine Entschlüsselung auf einmal an die Karte (innerhalb
2764 einer APDU) senden muss, kann man nur etwas weniger als 8KiB entschlüsseln (bzw.
2765 64KiB bei extended APDUs, die jedoch nicht alle Kartenterminal unterstützen), obwohl
2766 das ECIES-Verfahren an für sich die Ver- und Entschlüsselung praktisch beliebig großer
2767 Datenmengen unterstützt. Auch wäre es aus Nutzersicht in Bezug auf die Performanz
2768 nicht akzeptabel, schon allein moderat große verschlüsselte Dokumente komplett zu
2769 einer Karte für eine Entschlüsselung zu transportieren (mehr als 18 Minuten würde dies
2770 für ein 10 MiB großes Dokument benötigen). Deswegen wird für die TI hier eine
2771 zusätzliche Metaebene eingeführt und normativ gefordert. Analog zu einer
2772 Verschlüsselung mittels RSAES-OAEP muss ein Sender einen Transportschlüssel zufällig
2773 erzeugen. Ein solcher Transportschlüssel, ist dann aus Sicht der Chipkarte der zu ver-
2774 oder entschlüsselnde Klartext. Der aus Nutzersicht eigentliche Klartext (Dokumente) wird
2775 nie an die Karte gesendet. Die Karte entschlüsselt den verschlüsselten Transportschlüssel
2776 und übergibt ihn anschließend an den Kartennutzer. Dieser kann mit dem
2777 Transportschlüssel nun das Dokument unabhängig von der Chipkarte entschlüsseln. Bei
2778 RSAES-OAEP wird der Transportschlüssel als Content-Encryption-Key (CEK) bezeichnet.
2779 Im hier vorliegenden Fall bei ECIES kann diese Bezeichnung zu Missverständnissen
2780 führen. Mittels ECIES wird über ein ECDH und folgender Schlüsselableitung ein
2781 Verschlüsselungsschlüssel erzeugt. Diesen kann man auch als CEK bezeichnen,
2782 weswegen im Folgenden immer nur vom Transportschlüssel gesprochen wird.

2783 Da eine solche Metaebene für eine ECIES-Chifftrat-Kodierung unüblich ist (weil ohne TI-
2784 Chipkarteneinsatz unnötig), ist die Kodierung der Chifftrate mittels CMS oder XMLEnc
2785 nichttrivial und wird daher im Interesse der zu erzielenden Interoperabilität in diesem
2786 Abschnitt ausführlicher dargestellt.

2787 Für die symmetrische Verschlüsselung der Nutzerdaten (Dokumente etc.) wird zufällig ein
2788 Transportschlüssel erzeugt. Für diesen gelten die Vorgaben aus GS-A_4389
2789 (symmetrische Verschlüsselung) und GS-A_4368 (Schlüsselerzeugung). Der
2790 Transportschlüssel wird dann unkodiert ("is just the \"value\" of the content-encryption
2791 key" [RFC-5652#6.4]) zur Verschlüsselung an das ECIES-Verfahren übergeben. Bei der
2792 ECIES-Verschlüsselung müssen dann die Parameter so gewählt werden, dass eine

2793 Chipkarte der TI diese unterstützt, d. h. nach [gemSpec_COS#6.8.2.3 Asymmetrische
2794 Entschlüsselung mittels ELC].

2795 Das erhaltene ECIES-Chifftrat muss dann als eine ASN.1-Struktur kodiert werden, die
2796 genau dem Aufbau entspricht, den man benötigt um eine Entschlüsselung mittels einer
2797 Chipkarte der TI durchzuführen (vgl. [gemSpec_COS#(N085.068) Spiegelstrich 7).

7. Es gilt (*Hinweis: cipher ist hier identisch zu (N090.300)c, (N091.700)d und
(N094.400)c definiert*):

i. *cipher* MUSS ein DER codiertes DOA6 sein.

ii. *cipher* = 'A6-L_{A6}-(*oidDO* || *keyDO* || *cipherDO* || *macDO*).

iii. *oidDO* = '06-L₀₆-oid '.

iv. *keyDO* = '7F49-L_{7F49}-(86 - L₈₆ - PO_A)'.

v. *cipherDO* = '86-L₈₆-(02 || C)'.

vi. *macDO* = '8E-L_{8E}-T'.

7. Es gilt (*Hinweis: cipher ist hier identisch zu (N090.300)c, (N091.700)d und
(N094.400)c definiert*):

i. *cipher* MUSS ein DER codiertes DOA6 sein.

ii. *cipher* = 'A6-L_{A6}-(*oidDO* || *keyDO* || *cipherDO* || *macDO*).

iii. *oidDO* = '06-L₀₆-oid '.

iv. *keyDO* = '7F49-L_{7F49}-(86 - L₈₆ - PO_A)'.

v. *cipherDO* = '86-L₈₆-(02 || C)'.

vi. *macDO* = '8E-L_{8E}-T'.

Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält

Beispiel:

```
poGOBgkrJAMDAggBAQd/SUOGQQRouC6tM2TQQ+RP3pptgdAaDF8Te7IVCkUBe2H+PJSLK4W/BXIX
kndiBwEfftd5wk4pjjzCdC2j1q14/CIWcW89nhjEC7G47UAu2ZqmbIhxstkXV3UI2UUek/qwBwtb2
6aUild+5kkTZxf5674OKHsdj6IFwjggvhYt9b/CTsA==
```

```
$ dumpasn1 a.bin
```

```
0 142: [6] {
3 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
14 67: [APPLICATION 73] {
17 65: [6]
: 04 68 B8 2E AD 33 64 D0 43 E4 4F DE 9A 6D 81 D0
: 1A 0C 5F 13 7B B2 15 0A 45 01 7B 61 FE 3C 94 8B
: 2B 85 BF 05 72 17 92 77 62 07 01 1F 7E D7 79 C2
: 4E 29 8F 30 9D 0B 68 F5 AB 5E 3F 08 85 9C 5B CF
: 67
: }
84 49: [6]
: 02 EC 6E 3B 50 0B B6 66 A9 9B 22 1C 6C B6 45 D5
: DD 42 36 51 47 A4 FE AC 01 C2 D6 F6 E9 A5 22 95
: DF B9 92 44 D9 5D FE 7A EF 83 8A 1D 27 63 E8 81
: 70
135 8: [14] 2F 85 8B 7D 6F F0 93 B0
```

2825 : }

2826 Diese Datenstruktur soll konzeptionell so behandelt werden, wie ein RSA-OAEP-Chifftrat

2827 eines CEK. Es ist jedoch die hiermit neu definierte OID oid_ti_ecies_transport_encryption

2828 [gemSpec_OID] zu verwenden.

2829

2830 Ein Beispiel aus [gemSpec_SMIME-KOMLE] dient als Vorlage für die Darstellung der zu

2831 verwendenden Kodierung mittels CMS (S/MIME):

2832 ContentInfo

2833 .contentType: 1.2.840.113549.1.9.16.1.23 (id-ct-authEnvelopedData)

2834 ..AuthEnvelopedData

2835 ...version: 0

2836 ...recipientInfos

2837RecipientInfo

2838ktri

2839version: 0

2840rid

2841issuerAndSerialNumber

2842issuer

2843[... weitere Kindelemente ...]

2844SerialNumber: 123456789

2845keyEncryptionAlgorithm

2846oid_ti_ecies_transport_encryption

2847encryptedKey: [... ASN.1-Struktur (Tupel (PO, C, T) in

2848 kartenkompatibler ASN.1-Binärverpackung) ...]

2849RecipientInfo

2850ktriversion: 0

2851rid

2852issuerAndSerialNumber

2853issuer

2854[... weitere Kindelemente ...]

2855SerialNumber: 314159265

2856keyEncryptionAlgorithm

2857OID TI-ECIES-TransportEncryption

2858encryptedKey: [... ASN.1-Struktur (Tupel (PO, C, T) in

2859 kartenkompatibler ASN.1-Binärverpackung) ...]

2860 ...authEncryptedContentInfo

2861contentType: 1.2.840.113549.1.7.1 (id-data)

2862contentEncryptionAlgorithm:

2863algorithm: 2.16.840.1.101.3.4.1.46 (id-aes256-gcm)

2864parameters:

2865aes-nonce: [... IV ...]

2866aes-ICVlen: [... ICVLen ...]

2867encryptedContent: [...]

2868 ...mac: [...]

2869 ...unauthAttrs

2870Attribute (id-recipientEmails)

2871attrType: komle-recipient-emails

2872 und so weiter ...

2873 Für die Kodierung von TI-ECIES-Chiffraten innerhalb von XML wird hiermit der neue

2874 Identifier "http://gematik.de/ecies/2019" definiert. Für die XML-Kodierung ist eine

2875 Kodierung analog der folgenden Struktur zu verwenden.

```
2876 <?xml version="1.0" encoding="UTF-8"?>
2877 <xenc:EncryptedData
2878     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2879     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
2880     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2881     xmlns:dsig11="http://www.w3.org/2009/xmldsig11#"
2882     xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
2883     Type="http://www.w3.org/2001/04/xmlenc#">
2884     <xenc:EncryptionMethod
2885 Algorithm="http://www.w3.org/2009/xmlenc11#aes256-gcm" />
2886 <!-- Damit ist len(IV)=12 Byte und TagLen=16 Byte, vgl. [XMLEnc#5.2.4 AES-
2887 GCM] -->
2888     <ds:KeyInfo>
2889         <xenc:EncryptedKey>
2890             <!-- Hybridschlüssel für einen Empfänger, für weitere Empfänger gäbe es
2891 jeweils ein weiteres EncryptedKey-Element -->
2892             <xenc:EncryptionMethod Algorithm="http://gematik.de/ecies/2019" />
2893             <!-- Die Version des speziellen ECIES -->
2894             <ds:KeyInfo>
2895                 <ds:X509Data>
2896                     <ds:X509Certificate>
2897                         <!-- Base64-kodiertes X.509-Zertifikat (DER) des Empfängers -->
2898                         </ds:X509Certificate>
2899                     </ds:X509Data>
2900                 </ds:KeyInfo>
2901                 <xenc:CipherData>
2902                     <xenc:CipherValue>
2903 <!-- Base64-kodierte ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler
2904 ASN.1-Binärverpackung) -->
2905                     </xenc:CipherValue>
2906                 </xenc:CipherData>
2907             </xenc:EncryptedKey>
2908         </ds:KeyInfo>
2909         <xenc:CipherData>
2910             <xenc:CipherValue>
2911 <!--
2912 Base64-kodiertes symmetrisch mittels AES-256-GCM verschlüsseltes Dokument
2913 (IV || ciphertext || Tag) mit len(IV)=12 Byte und len(Tag)=16 Byte,
2914 vgl. [XMLEnc#5.2.4 AES-GCM]
2915 -->
2916             </xenc:CipherValue>
2917         </xenc:CipherData>
2918     </xenc:EncryptedData>
2919
```

A_17220 - Verschlüsselung binärer Daten (ECIES) (ECC-Migration)

Alle Produkttypen, die binäre Daten (also nicht XML-Daten) ECC-basiert verschlüsseln (im Folgenden als Nutzerdaten bezeichnet) und diese mittels CMS [RFC-5626] kodieren, MÜSSEN folgende Vorgaben umsetzen.

1. Zunächst MUSS ein 256-Bit AES-Schlüssel zufällig erzeugt werden (vgl. GS-A_4368), Dieser Schlüssel wird im Folgenden als Transportschlüssel bezeichnet (vgl. Erklärung in Abschnitt [gemSpec_Krypt#5.7-ECIES]).

- 2928 2. Mit diesem Transportschlüssel MÜSSEN die Nutzerdaten mittels AES-GCM und den
2929 Vorgaben aus GS-A_4389 verschlüsselt werden.
- 2930 3. Der Transportschlüssel MUSS unkodiert mit den in [gemSpec_COS#6.8.1.4 ELC
2931 Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden
2932 (siehe Erklärung in [gemSpec_Krypt#ECIES]). Das damit entstehende Chifftrat
2933 wird im Folgenden als Transport-Chifftrat bezeichnet.
- 2934 4. Das Transport-Chifftrat MUSS wie in [gemSpec_Krypt#5.7..ECIES] beschrieben in
2935 eine ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler ASN.1-
2936 Binärverpackung kodiert werden.
- 2937 5. Diese Kodierung MUSS in eine keyEncryptionAlgorithm-Datenstruktur mit der OID
2938 oid_ti_ecies_transport_encryption [gemSpec_OID] eingebracht werden.
- 2939 6. Die restliche Kodierung des mittels AES-GCM erzeugten Chifftrats der Nutzerdaten
2940 MUSS wie in CMS üblich [RFC-5626] [RFC-5084] erfolgen (vgl. Darstellung
2941 in [gemSpec_Krypt#5.7..ECIES]).

2942 [**<=**]

2943 **A_17221-01 - XML-Verschlüsselung (ECIES) (ECC-Migration)**

2944 Alle Produkttypen, die XML-Dokumente mittels [XMLEnc-1.1] und ECC-basierte
2945 verschlüsseln, MÜSSEN die Vorgaben aus A_17220 Spiegelstrich 1 bis 3 umsetzen.
2946 Weiter MÜSSEN sie folgende Vorgaben umsetzen:

- 2947 1. Das Transport-Chifftrat MUSS wie in [gemSpec_Krypt#5.7..ECIES] beschrieben in
2948 eine ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler ASN.1-
2949 Binärverpackung kodiert werden.
- 2950 2. Diese ASN.1-Binärverpackung MUSS Base64-kodiert werden und so kodiert in
2951 eine XML-Datenstruktur, so wie sie in [gemSpec_Krypt#5.7..ECIES] beschrieben
2952 ist, eingebracht werden (Hinweis: man beachte ohne "RecipientKeyInfo" Tags).
- 2953 3. Die mittels AES-GCM verschlüsselten Nutzerdaten MÜSSEN ebenfalls Base64-
2954 kodiert in die eben erzeugte XML-Datenstruktur gemäß [gemSpec_Krypt#5.7..
2955 ECIES] eingebracht werden.

2956 [**<=**]

2957 Im Interesse der Interoperabilität stellt die gematik auf Anfrage Testvektoren (Beispiel-
2958 Chifftrate mit Klartext) zur Verfügung.

2959 **5.7.1 ECIES und authentifizierte Broadcast-Encryption**

2960 In [SEC1-2009#5.2] und in [RFC-5753#4] wird auf ein potentielles Problem
2961 hingewiesen, dass insbesondere bei der Verwendung eines static-static-ECDH innerhalb
2962 von ECIES auftreten kann (also Sender und Empfänger verwenden ihre Langzeit-Identität
2963 "static-static"). Verallgemeinert formuliert, handelt es sich um das Problem der
2964 authentisierten Broadcast-Verschlüsselung. Ein Sender möchte an mehrere Empfänger
2965 "gleichzeitig" den gleichen Klartext verschlüsselt senden (vgl. auch [RFC-4082#1]). Bei
2966 TI-ECIES-TransportEncryption [gemSpec_Krypt#ECIES] wird der Transportschlüssel
2967 jeweils für jeden Empfänger mittels ECIES verschlüsselt. Jeder Empfänger kennt nun
2968 diesen zwischen dem Sender und allen Empfängern geteilten Schlüssel
2969 (Transportschlüssel) und kann jetzt (aus kryptographischer Sicht) den Klartext beliebig
2970 verändern, ohne dass dies bei einer Entschlüsselung auffällt. Die "authenticated

2971 Encryption" via AES-GCM (Transportschlüssel) kann hier also nicht die von ihr evtl.
2972 angenommene Sicherheitsleistung erbringen.

2973 Wenn also bspw. bei KOM-LE eine Nachricht an mehrere Empfänger (eingetragen im CC-
2974 Feld) versendet werden soll, so muss an dieser Stelle zusätzlich die Authentizität der
2975 versendeten Nachricht gesichert werden. Bei KOM-LE erfolgt dies über die verpflichtende
2976 Signatur der Nachricht mit Hilfe der SMC-B (OSIG-Schlüsselmateriale). Es ist davon
2977 auszugehen, dass andere Anwendungen, die an mehrere Empfänger Nachrichten/Daten
2978 "gleichzeitig" versenden, ebenfalls zusätzliche Maßnahmen ergreifen müssen.

2979 **5.7.2 ECIES und mobKT**

2980 Ein "Mobiles Kartenterminal" [gemSpec_MobKT] ist so etwas wie ein Tablet mit zwei
2981 Chipkartenslots. Es ermöglicht einem LE außerhalb seiner Praxisräumlichkeiten (also
2982 insbesondere ohne Konnektor und stationäres eHealth-Kartenterminal), auf Daten eines
2983 Versicherten auf dessen eGK zuzugreifen. Das Mobile Kartenterminal muss die dabei
2984 ausgelesenen versichertenspezifischen Daten verschlüsselt lokal im Gerät ablegen. Wenn
2985 das Mobile Kartenterminal verloren geht, sind damit diese Daten geschützt. Sie können
2986 erst mit Hilfe des gesteckten und freigeschalteten HBA des LE wieder entschlüsselt
2987 (genutzt) werden, bspw. zur Übertragung ins Primärsystem. Für die Verschlüsselung
2988 muss nach Ende 2025 ECIES und nicht mehr RSA-OAEP verwendet werden. Es gelten
2989 dafür fachlich quasi die gleichen Vorgaben wie in A_17220, nur gibt es keine
2990 Notwendigkeit in Bezug auf die Kodierung des Chiffrats interoperabel sein zu müssen.
2991 Denn nur das spezifische Mobile Kartenterminal selbst muss die Daten ver- und
2992 entschlüsseln können im Sinne von "die Kodierung der Chiffre auswerten können".
2993 Deshalb gibt es nachfolgend eine Spezialisierung von A_17220 für das Mobile
2994 Kartenterminal.

2995 **A_17575 - MobKT: Verschlüsselung binärer Daten (ECIES) (ECC-Migration)**

2996 Ein Mobiles Kartenterminal MUSS folgende Vorgaben umsetzen.

- 2997 1. Für die Verschlüsselung der Versichertendaten MUSS ein 256-Bit AES-Schlüssel
2998 zufällig erzeugt werden (vgl. GS-A_4368). Dieser Schlüssel wird im Folgenden als
2999 Transportschlüssel bezeichnet.
- 3000 2. Mit diesem Transportschlüssel MÜSSEN die Versichertendaten mit AES-GCM und
3001 den Vorgaben aus GS-A_4389 verschlüsselt werden.
- 3002 3. Der Transportschlüssel MUSS mit den in [gemSpec_COS#6.8.1.4 ELC
3003 Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden
3004 (siehe Erklärung in [gemSpec_Krypt#5.7.-ECIES] und [gemSpec_Krypt#Hinweis
3005 zu A_17575]).
- 3006 4. Falls auf dem gesteckten HBA ein ECC-basiertes ENC-Zertifikat vorhanden ist, so
3007 MUSS ECIES für die Ver- und Entschlüsselung des Transportschlüssels verwendet
3008 werden, anstatt von RSA-OAEP. Falls noch kein ECIES-verschlüsselter
3009 Transportschlüssel im Mobilen Kartenterminal vorliegt, sondern ein RSA-OAEP-
3010 verschlüsselter Transportschlüssel, so MUSS dieser Transportschlüssel zusätzlich
3011 mittels ECIES und dem ECC-ENC-Schlüssel des HBAs des LE verschlüsselt werden.

3012 **[<=]**

3013 Hinweis zu A_17575:

3014 In einem HBA steht die Schnittstelle [gemSpec_COS#6.8.1.4 ELC Verschlüsselung] für
3015 die Verschlüsselung des Transportschlüssels zur Verfügung. Dass der Transportschlüssel

3016 verschlüsselt werden muss, wird nur sehr selten als Anwendungsfall vorkommen. Die
3017 Entschlüsselung - notwendiger Weise mit und durch den HBA - jedoch häufig.

3018 **5.8 ECC-Migration eHealth-KT**

3019 Mit A_17089-* und A_17090-* wird vom eHealth-Kartenterminal die Unterstützung der
3020 mit der Kartengeneration 2.1 (vgl. [gemSpec_gSMC-KT_ObjSys_G2.1])
3021 hinzugekommenen ECDSA-basierten Identität bereitgestellt für die Nutzung von TLS
3022 (vgl. auch Abschnitt 3.3.2- TLS-Verbindungen).

3023 **A_17089-03 - eHealth-Kartenterminals: TLS-Verbindungen (ECC-Migration)**

3024 Ein eHealth-Kartenterminal MUSS prüfen, ob die in ihm gesteckte SMC-KT für die TLS-
3025 Verbindung zum Konnektor eine RSA-basierte Identität (AUT) und/oder eine ECDSA-
3026 basierte Identität besitzt (vgl. [gemSpec_gSMC-KT_ObjSys_G2.1], bspw. jeweils EFs mit
3027 ShortFileIdentifier 1 und 4 prüfen).
3028 Falls eine RSA-basierte Identität dort vorhanden ist, so MUSS das eHealth-Kartenterminal
3029 folgende TLS-folgende Vorgaben erfüllen:

- 3030 1. Als Cipher-Suite MÜSSEN TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
3031 und TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 unterstützt werden.
- 3032 2. Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE"
3033 im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5]
3034 unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1
3035 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in
3036 A_17089-* aufgeführt DÜRFEN NICHT verwendet werden.
- 3037 3. Es KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1]
3038 unterstützen.

3039 Falls eine ECDSA-basierte Identität vorhanden ist, so MUSS das eHealth-Kartenterminal
3040 zusätzlich folgende Vorgaben erfüllen:

- 3041 1. Als Ciphersuiten MÜSSEN TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
3042 (0xC0,0x2B) und TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
3043 (0xC0,0x2C) unterstützt werden.
- 3044 2. Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE"
3045 im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5]
3046 unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1
3047 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in GS-
3048 A_17089-* aufgeführt DÜRFEN NICHT verwendet werden.

3049 Dies bedeutet, falls beide Identitäten auf der SMC-KT vorhanden sind (wie bei
3050 [gemSpec_gSMC-KT_ObjSys_G2.1]), so MÜSSEN alle vier oben genannten Ciphersuiten
3051 unterstützt werden.
3052 [\leq]

3053 **A_17090-01 - eHealth-Kartenterminals: Signaturverfahren beim initialen 3054 Pairing zwischen Konnektor und eHealth-Kartenterminal (ECC-Migration)**

3055 Ein eHealth-Kartenterminal MUSS in Bezug auf das verwendete Signaturverfahren beim
3056 initialen Pairing zwischen Konnektor und eHealth-Kartenterminal folgende Vorgaben
3057 umsetzen:

- 3058 1. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine RSA-basierte
3059 Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret

3060 (ShS.AUT.KT vgl. [gemSpec_KT#2.5.2.1, 3.7.2.1]) RSASSA-PSS [PKCS#1] und
3061 SHA-256 verwendet werden. (Hinweis: die Parameter für RSASSA-PSS wie MGF
3062 oder Salt-Länge sind durch die SMC-KT eindeutig und fest vorgegeben und
3063 werden deshalb hier nicht aufgeführt.)

3064 2. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine ECDSA-basierte
3065 Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret ECDSA
3066 [BSI-TR-03111] und SHA-256 verwendet werden (Hinweis: die zu verwendenden
3067 Domainparameter (Kurve etc.) sind durch die SMC-KT eindeutig und fest
3068 vorgegeben). Für die Kodierung der ECC-Parameter in der Signatur MUSS die
3069 plain-Kodierung nach [TR-03111#5.2.1] verwendet werden.

3070 [**<=**]

3071 Hinweis: Das in A_17090-01 geforderte Verhalten lässt sich bei OpenSSL leicht mit
3072 SSL_get_current_cipher() umsetzen.

3073 Ein eHealth-Kartenterminal muss beim TLS-Verbindungsaufbau, der vom Konnektor
3074 initiiert wird, dessen TLS-Client-Zertifikat prüfen. Dafür muss ein eHealth-Kartenterminal
3075 alle "CA-Zertifikate der relevanten TSP speichern" [gemSpec_KT#TIP1-A_3255]. Um
3076 diesen kritischen Punkt, jedoch noch einmal zu unterstreichen:

3077 **A_17183 - CA-Zertifikate der relevanten TSP speichern (ECC-Migration)**

3078 Das eHealth-Kartenterminal MUSS bei der Umsetzung von [gemSpec_KT#TIP1-A_3255]
3079 sowohl RSA-basierte CA-Zertifikate der Komponenten-PKI als auch ECC-basierte CA-
3080 Zertifikate (TSL(ECC-RSA)) der Komponenten-PKI speichern.

3081 [**<=**]

3082 **A_22458 - TLS-Algorithmus passend zum Pairing**

3083 Der Konnektor MUSS beim TLS-Verbindungsaufbau (TLS-Handshake) zu einem eHealth-
3084 Kartenterminal ausschließlich Ciphersuiten mit solchen Authentisierungsalgorithmen
3085 (entweder RSA oder ECDSA) anbieten, die zum Algorithmus des gespeicherten KT-
3086 Zertifikats passen, wenn zu diesem Kartenterminal bereits Pairinginformationen (Shared
3087 Secret in Kombination mit dem Zertifikat des Kartenterminals) gespeichert sind. [**<=**]

3088 Konnektoren speichern die Pairinginformationen zu den mit ihnen gepairten
3089 Kartenterminals als Tupel {Kartenterminal-Zertifikat; Shared Secret}, wobei das beim
3090 Pairing vom Kartenterminal genutzte Zertifikat gespeichert wird. Wird später (bspw.
3091 durch ein Software-Update des Kartenterminals, welches die Nutzung von ECDSA
3092 Identitäten ermöglicht) ein anderes Zertifikat vom Kartenterminal beim TLS-Handshake
3093 präsentiert (also bspw. eine ECDSA Identität statt der zuvor genutzten RSA-Identität),
3094 passt dies für den Konnektor nicht zur vorhandenen Pairinginformation. Der
3095 Verbindungsaufbau scheitert somit, bis ein neues Pairing vorgenommen wird (bei dem
3096 dann das ECDSA-Zertifikat vom Kartenterminal verwendet wird). Dieses Verhalten muss
3097 vermieden werden, da dies einen massenhaften Ausfall von Kartenterminals durch ein
3098 Software-Update der Terminals hervorrufen kann. Daher darf der Konnektor nur solche
3099 Cipher-Suiten anbieten, die auch zum für das betroffene Kartenterminal gespeicherte
3100 Zertifikat passen.

3101 **5.8.1 Interoperabilität zwischen eHealth-KT und Konnektor**

3102 Zur Sicherstellung der Interoperabilität zwischen Health-KT und Konnektor werden
3103 folgende Festlegungen getroffen. Bei den folgend aufgeführten Themen hatte es
3104 bezüglich der Interoperabilität Probleme gegeben.

3105 **A_22451 - ClientHello ohne akzeptable Cipher-Suite**

- 3106 Falls die ClientHello-Nachricht keine der gemäß [gemSpec_Krypt#A_17089-01] zu
3107 unterstützenden Cipher-Suiten enthält, dann MUSS das eHealth-Kartenterminal die
3108 ClientHello-Nachricht mit einem "handshake_failure" beantworten, siehe [RFC-
3109 5426#7.4.1.2]. [**<=**]
- 3110 Hinweise zum Thema Renegotiation:
- 3111 Hinweis 1: Gemäß GS-A_5525 unterstützt der Konnektor "renegotiation". Daraus folgt,
3112 dass er diese Fähigkeit in der ClientHello- Nachricht entweder durch eine
3113 "renegotiation_info" extension anzeigt, oder durch ein
3114 TLS_EMPTY_RENEGOTIATION_INFO_SCSV (siehe [RFC-5746#3.4, §1, erster Punkt]).
- 3115 Hinweis 2: Gemäß [RFC-5246#7.4.1.3 letzter §] ist es einem Server nur dann erlaubt in
3116 der ServerHello-Nachricht eine "renegotiation_info" extension zu schicken, wenn die
3117 ClientHello-Nachricht Informationen zu "renegotiation" enthält. Gemäß dem
3118 vorstehenden Hinweis ist das beim TLS-Handshake zwischen Konnektor und eHealth-KT
3119 stets der Fall.
- 3120 Hinweis 3: Die gematik legt nicht fest, wie sich ein eHealth-KT zu verhalten hat, wenn die
3121 ClientHello-Nachricht weder eine "renegotiation_info" extension, noch ein
3122 TLS_EMPTY_RENEGOTIATION_INFO_SCSV enthält.
- 3123 Hinweis 4: Es gilt für das eHealth-KT GS-A_5524-*, womit sichergestellt ist, dass ein
3124 eHealth-KT keine unsichere TLS-Renegotiation durchführt.
- 3125 **A_22453 - ServerHello, Cipher-Suite**
- 3126 Von den gemäß [gemSpec_Krypt#A_17089-01] zu unterstützenden Cipher-Suiten wählt
3127 das eHealth-Kartenterminal eine aus, vergleiche [RFC-5426#7.4.1.3]. Falls das eHealth-
3128 Kartenterminal eine Cipher-Suite aus [gemSpec_Krypt#A_17089-01] in der Form
- 3129 1. TLS_DHE_RSA... (also RSA) auswählt, dann MUSS es eine RSA-basierte Identität
3130 zur Authentisierung im Kontext des TLS-Protokolls verwenden.
- 3131 2. TLS_ECDHE_ECDSA... (also ECDSA) auswählt, dann MUSS es eine ECDSA-
3132 basierte Identität zur Authentisierung im Kontext des TLS-Protokolls verwenden.
- 3133 [**<=**]
- 3134 **A_22454 - CertificateRequest**
- 3135 Das eHealth-Kartenterminal MUSS eine CertificateRequest-Nachricht schicken, für die
3136 folgendes gilt:
- 3137 1. Die Struktur CertificateRequest gemäß [RFC-5246#7.4.4] MUSS im Abschnitt
3138 certificate_types genau ein Element vom Typ ClientCertificateType enthalten.
- 3139 2. Falls das eHealth-Kartenterminal in der ServerHello-Nachricht eine Cipher-Suite
3140 aus [gemSpec_Krypt#A_17089-01] in der Form
- 3141 a. TLS_DHE_RSA... (also RSA) anzeigt, dann MUSS als ClientCertificateType
3142 "rsa_sign" gewählt werden, siehe [RFC-5246#7.4.4].
- 3143 b. TLS_ECDHE_ECDSA... (also ECDSA) anzeigt, dann MUSS als
3144 ClientCertificateType "ecdsa_sign" gewählt werden, siehe [RFC-8422#5.5].
- 3145 [**<=**]
- 3146 **A_22455 - ClientCertificate**
- 3147 Bezüglich der ClientCertificate-Nachricht (siehe [RFC-5246#7.4.6]) gilt für das eHealth-
3148 Kartenterminal folgendes Verhalten:

- 3149 1. Falls der Client keine ClientCertificate-Nachricht schickt, dann MUSS der TLS-
3150 Handshake mit einem "failure alert" abgebrochen werden.
- 3151 2. Der TLS-Handshake MUSS fortgesetzt werden, selbst wenn die "certificate_list"
3152 a. leer ist (also kein Element enthält), oder
3153 b. kein End-Entity-Zertifikat daraus erfolgreich extrahiert und erfolgreich gegen
3154 eine im eHealth-Kartenterminal gespeicherte CA geprüft werden konnte, oder
3155 c. ein End-Entity-Zertifikat unpassenden Typs enthält (beispielsweise RSA-
3156 PublicKey statt ECDSA-PublicKey).

3157 [\leq]

3158 **5.9 ECC-Migration Konnektor**

3159 **A_17094-02 - TLS-Verbindungen Konnektor (ECC-Migration)**

3160 Der Konnektor MUSS zusätzlich zu den RSA-basierten TLS-Ciphersuiten (vgl. GS-A_4385
3161 und GS-A_5345-01) die TLS-Ciphersuiten

- 3162 1. TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 und
3163 2. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

3164 unterstützen. Dabei MÜSSEN bei dem ephemeren Elliptic-Curve-Diffie-Hellman-
3165 Schlüsselaustausch die Kurven P-256 und P-384 [FIPS-186-5] und die Kurven
3166 brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt
3167 werden. Andere Kurven SOLLEN NICHT verwendet werden.

3168 Falls der Konnektor in der Rolle TLS-Client agiert, so MUSS er die eben genannten
3169 Ciphersuiten gegenüber RSA-basierten Ciphersuiten (vgl. GS-A_4384-*) bevorzugen (in
3170 der Liste "cipher_suites" beim ClientHello vorne an stellen, vgl. [RFC-5256#7.4.1.2
3171 Client Hello]).

3172 [\leq]

3173 Hinweis: für den Konnektor gelten die IPsec-Anforderungen A_17125 und A_17126.

3174 **A_17209 - Signaturverfahren für externe Authentisierung (ECC-Migration)**

3175 Der Konnektor MUSS an der Schnittstelle für die externe Authentisierung
3176 die Signaturverfahren RSASSA-PKCS1-v1_5 [PKCS#1], RSASSA-PSS [PKCS#1] und
3177 ECDSA [BSI-TR-03111] anbieten. [\leq]

3178 **A_23511 - Konnektor, IOP, Kodierung ECC-Schlüssel, Primärsystem- 3179 Verbindungssicherung**

3180 Der Konnektor MUSS sicherstellen, dass bei der Umsetzung von TIP1-4517-* ECC-
3181 Schlüssel stets in der "named-curve"-Kodierung exportiert werden (d. h., eben gerade
3182 nicht die "explizite" Kurvenparameter-Kodierung verwenden).

3183 D. h., sowohl

- 3184 1. für ein Primärsystem von einem Konnektor erzeugte und exportierte private ECC-
3185 Schlüssel MÜSSEN mittels named-curve-Darstellung kodiert sein (siehe Hinweise)
3186 und
- 3187 2. bei den vom Konnektor erzeugten TLS-Authentisierungszertifikaten MÜSSEN die
3188 im Zertifikat bestätigten öffentlichen ECC-Schlüssel in der named-curve-
3189 Darstellung kodiert sein (siehe Hinweise).

3190 [\leq]

3191 Hinweise:

3192 (1) Die meisten in Primärsystemen verwendeten Kryptographie-Bibliotheken erlauben
3193 mittlerweile keine explizite Angabe von ECC-Kurvenparametern (i. S. v. explizite
3194 Aufführung von A, B, p und q-Werten). Damit sind bspw. vom Konnektor erzeugte
3195 Zertifikate, die nicht A_23511-konform sind, mit diesen Bibliotheken nicht verwendbar.
3196 Fachlicher Hintergrund ist CVE-2020-0601 (<https://nvd.nist.gov/vuln/detail/CVE-2020-0601>) und "Digital Signature Schemes with Domain Parameters", Serge Vaudenay, 2004
3197 (<https://lasec.epfl.ch/pub/lasec/doc/Vau04b.pdf>).
3198
3199

3200 (2) Ein Beispiel für einen privaten ECC-Schlüssel mit named-curve-Parameter-Kodierung
3201 im PEM-Format:

```
3202 -----BEGIN EC PARAMETERS-----  
3203 BgkrJAMDaggBAQc=  
3204 -----END EC PARAMETERS-----  
3205 -----BEGIN EC PRIVATE KEY-----  
3206 MHgCAQEEIC5PFJ/4fOdPLZlBebMqpHN8ydae9kSPGS5qICS/WGpnoAsGCSskAwMC  
3207 CAEBB6FEA0IABCpcFsaT3f+O9Cg676mRWR4WazrrbFMcMZ5dwJhrhVsjszA3z85q  
3208 u8eCSvAke3jt2+nOTNh1s3ExkZVSOJD1aLc=  
3209 -----END EC PRIVATE KEY-----
```

3210 (über

3211 openssl ecparam -name brainpoolP256r1 -genkey -param_enc named_curve -
3212 out<pre>example-named-curve-private-key.pem

3213 erzeugt)

3214 Die DER-Kodierung sieht dann wie folgt aus

```
3215 $ openssl ec -in example-named-curve-private-key.pem -pubout -outform der  
3216 -out mykey.pub  
3217 $ dumpasn1 mykey.pub  
3218 0 90: SEQUENCE {  
3219 2 20: SEQUENCE {  
3220 4 7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)  
3221 13 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)  
3222 : }  
3223 24 66: BIT STRING  
3224 : 04 2A 5C 16 C6 93 DD FF 8E F4 28 3A EF A9 91 59  
3225 : 1E 16 6B 3A EB 6C 53 1C 31 9E 5D C0 98 6B 85 5B  
3226 : 23 4B 30 37 CF CE 6A BB C7 82 4A F0 24 13 78 ED  
3227 : DB E9 CE 4C D8 75 B3 71 31 91 95 52 38 90 F5 68  
3228 : B7  
3229 : }
```

```
3230  
3231 0 warnings, 0 errors.
```

3232

3233 (3) In Abschnitt "5.2 X.509-Root der TI" gibt es ein ausführliches Beispiel für ein X.509-
3234 Zertifikat mit einem öffentlichen ECC-Schlüssel mit named-curve-Parameter Kodierung.

3235 **5.10 Verschiedene Produkttypen und ECC-Migration (informativ)**

- 3236 Dem VPN-Zugangsdienst sind die IPsec-Anforderungen A_17125 und
3237 A_17126 zugewiesen, ebenso A_17205. Im Rahmen der Registrierung bei einem VPN-
3238 Zugangsdienst wird vom Konnektor eine Signatur mittels einer SMC-B erzeugt. Diese
3239 muss der VPN-Zugangsdienst (Registrierungsserver) prüfen können, dafür gilt für
3240 diesen A_17206.
- 3241 Für die Produkttypen, die bei der Anwendung VSDM verwendet werden, ist die ECC-RSA-
3242 TSL-Auswertung A_17205 und die ECC-Unterstützung bei TLS A_17124-* relevant.
- 3243 Fachanwendungsspezifische Anpassungen aufgrund der ECC-Migration befinden sich in
3244 den jeweiligen Spezifikationen (bspw. [\[gemSpec CM KOMLE#A 17464\]](#)).

6 VAU-Protokoll für E-Rezept

6.1 Übersicht (informativ)

Ähnlich wie bei der Anwendung ePA besitzt der Fachdienst E-Rezept mindestens zwei HTTPS-Schnittstellen. Die dabei vom Nutzer (Client) aufgebaute TLS-Verbindung endet an einer dieser HTTPS-Schnittstellen. Die E-Rezept-VAU liegt hinter den Webschnittstellen. Um die Verbindungsstrecke zwischen HTTPS-Schnittstellen und E-Rezept-VAU zu schützen, verschlüsselt und kodiert der Client seine HTTP-Requests (als innere Requests bezeichnet) auf die im Abschnitt 6.2 definierte Weise. Diese so erzeugten HTTP-Requests (äußere Requests) sendet der Client per HTTPS an eine der Webschnittstellen des E-Rezept-Dienstes. Dabei ist für die Schnittstelle ein regelmäßig wechselndes Nutzerpseudonym im äußeren HTTP-Request sichtbar. Dieses unterstützt den Fachdienst E-Rezept bei der Lastverteilung und beim DoS-Schutz auf Applikationsebene. Von der Webschnittstelle erhält die E-Rezept-VAU die verschlüsselten Requests, entschlüsselt und verarbeitet diese. Als Antwort erzeugt die E-Rezept-VAU einen für den Client verschlüsselte HTTP-Response. Den Schlüssel dafür hat der Client für die VAU ephemeral erzeugt und dieser Schlüssel ist Teil des verschlüsselten Requests. Die Webschnittstelle empfängt das Chiffre von der VAU und gibt dieses als Antwort auf den HTTP-Request an den Client zurück.

Ablauf:

1. Der Client erfragt initial das X.509-Zertifikat der VAU (A_20160-*). Je nachdem, ob die X.509-Root der TI oder die TSL als Prüfbasis verwendet wird, erfolgt die Abfrage des VAU-X.509-Zertifikats spezifisch (entweder A_21216 oder A_21218). Im Zertifikat ist der öffentliche Verschlüsselungsschlüssel der E-Rezept-VAU enthalten. Dieses Zertifikat kommt aus der Komponenten-PKI der TI und ist langlebig, d. h. der Client kann dieses Zertifikat (inkl. Schlüssel) für folgende Verschlüsselungen lokal vorhalten.
2. Der Client erzeugt einen HTTP-Request mit Request-Body und Request-Header als Datenstrukturen (= innerer HTTP-Request) (A_20161-*).
3. Der Client verschlüsselt diesen Request mit dem Verschlüsselungsschlüssel der E-Rezept-VAU. Im Chiffre ist ein Authentisierungstoken des Nutzers enthalten, eine zufällig gewählte Request-ID, ein vom Client ephemeral erzeugter symmetrischer Antwortschlüssel und der am Anfang erzeugte HTTP-Request als Datenstrom (A_20161-*).
4. Der Client erzeugt einen neuen (äußeren) HTTP-Request und überträgt darin das Chiffre an eine HTTPS-Schnittstelle des Fachdienstes E-Rezept. Dabei gibt der Client ggf. das schon aus einer vorherigen Response des Fachdienstes E-Rezept mitgeteilte Nutzerpseudonym (A_20161-*) mit.
5. Die Webschnittstelle nimmt den Request entgegen und trifft eine Routing-Entscheidung oder eine Priorisierungsentscheidung im Lastfall. Anschließend leitet sie den Request an die VAU weiter (A_20162).
6. Die VAU entschlüsselt den Request und verarbeitet ihn. Die Antwort wird inkl. der Request-ID mit dem vom Nutzer erzeugten Antwortschlüssel verschlüsselt und an die Webschnittstelle gesendet.

7. Die Webschnittstelle antwortet gegenüber dem Client mit diesem Chifftrat auf den HTTP-Request.

8. Der Client entschlüsselt das Chifftrat und prüft die Request-ID in der entschlüsselten Antwort.

Das Protokoll ist statuslos. Zwischen verschiedenen VAU-Instanzen (bspw. VAU-Instanzen in unterschiedlichen Brandabschnitten eines Rechenzentrum) müssen keine Session-Schlüssel oder ähnliches synchronisiert werden.

Die Struktur der inneren HTTP-Request ist so einfach, dass davon auszugehen ist, dass in der VAU-Instanz keine umfangreichen Webserver oder HTTP-Bibliotheken notwendig sind. Ziel der E-Rezept-VAU ist es die Code-Komplexität (Trusted Computing Base, TCB) so weit es nur irgendwie geht minimal zu halten.

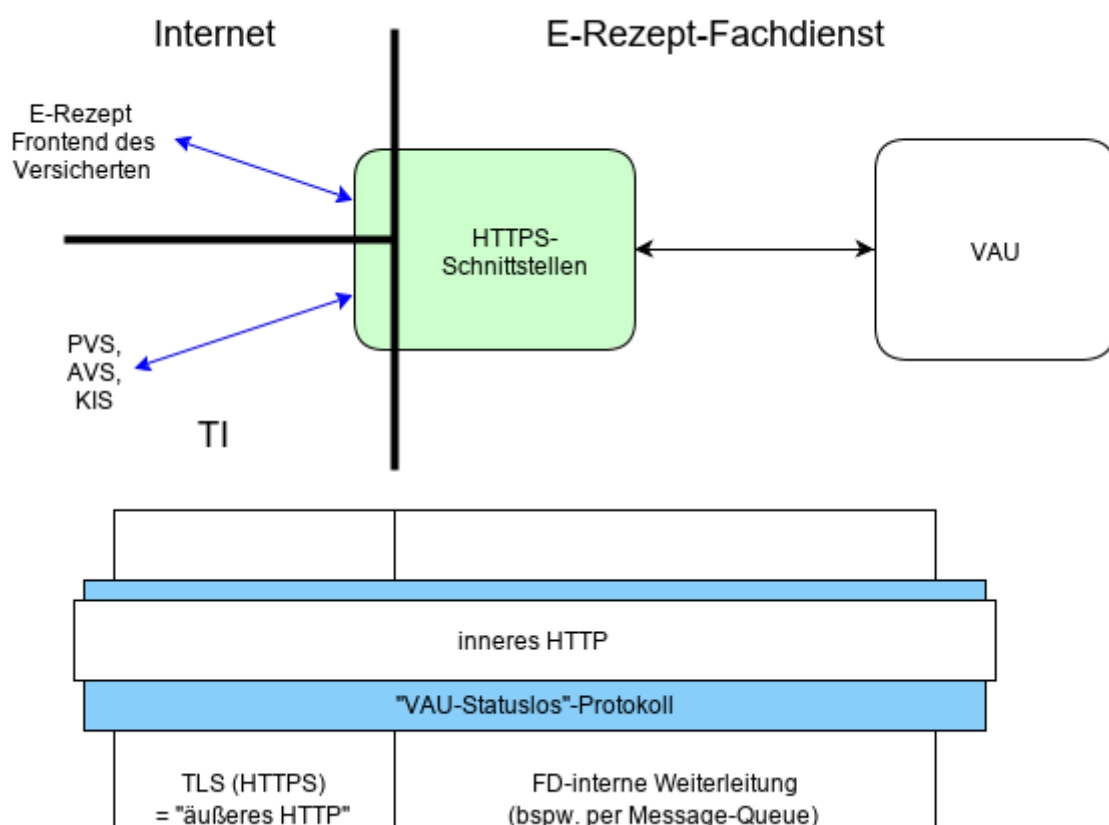


Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU

Die gematik stellt eine Beispielimplementierung bereit.

3305 **6.2 Definition**

3306 **6.2.1 E-Rezept-VAU-Identität**

3307 **A_20160-01 - E-Rezept-VAU, Schlüsselpaar und Zertifikat**

3308 Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

- 3309 1. Die VAU MUSS ein EE-X.509-Zertifikat aus der Komponenten-PKI der TI besitzen
3310 (mit Rollenkennung-OID "oid_erp-vau"), das einen ECC-EE-Schlüssel der VAU
3311 bestätigt.
- 3312 2. Die VAU MUSS die Vertraulichkeit des privaten Schlüssels für diese Zertifikat
3313 sicherstellen.
- 3314 3. Die notwendige Sicherung (Backup) und Verteilung dieses privaten Schlüssels
3315 MUSS ausschließlich im Mehr-Augen-Prinzip und mit geeigneten Maßnahmen zur
3316 Wahrung der Vertraulichkeit des Schlüssels geschehen.
- 3317 4. Der Fachdienst E-Rezept MUSS das VAU-Zertifikat in seinen Webschnittstellen
3318 unter dem Pfad `/Vaucertificate` (einer URL) durch Clients abrufbar machen.
3319 Dieses Zertifikat MUSS DER-kodiert sein.
- 3320 5. Der Fachdienst E-Rezept MUSS eine maximal 12 Stunden alte OCSP-Response für
3321 das VAU-Zertifikat in seinen Webschnittstellen unter dem Pfad
3322 `/VaucertificateOCSPResponse` für Clients abrufbar machen.

3323 **[<=]**

3324 Hinweis: Unter `/VaucertificateOCSPResponse` erhält ein Client (einfacher GET-Request)
3325 eine korrekte OCSP-Response für das VAU-Zertifikat (A_20160-*, Punkt 1). Dies ist
3326 analog wie OCSP-Stapling bei TLS zu sehen, nur auf einer höheren OSI-Schicht. Der FD
3327 stellt korrekte OCSP-Responses zur Verfügung damit nicht jeder Client selbst den OCSP-
3328 Responder fragen muss. Diese Funktionalität hat nichts mit der OCSP-Proxy-
3329 Funktionalität zu tun wie sie bspw. beim Zugangsgateway des Versicherten bei ePA
3330 angeboten wird. Der FD fragt bspw. stündlich selbst den OCSP-Status für sein VAU-
3331 Zertifikat ab, prüft die Antwort und stellt sie unter
3332 `/VaucertificateOCSPResponse` Clients zur Verfügung.

3333 **A_20967 - E-Rezept-VAU, Erstellung und Pflege der Schlüssel im Mehr-Augen-Prinzip**

3334 Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

- 3336 1. Die Erstellung, Sicherung und Wiederherstellung von nicht-flüchtigen (nicht
3337 kurzzeitig gültigen) Schlüsseln (A_20160-* Punkt 2-3, Masterkey für die
3338 Verschlüsselung der E-Rezept in der VAU etc.) MUSS im Mehr-Augen-Prinzip
3339 erfolgen bei dem mindestens ein gematik-Mitarbeiter beteiligt ist.
- 3340 2. Die Administration des/der HSM der VAU MUSS ebenfalls im Mehr-Augen-Prinzip
3341 erfolgen bei dem mindestens ein gematik-Mitarbeiter beteiligt ist.

3342 Die oben genannten Punkten MÜSSEN durch das/die HSM der VAU technisch
3343 durchgesetzt werden (Rechtekonzept). **[<=]**

3344 Hinweis: A_20967 beschreibt ein analoges Vorgehen wie es bei den SGD(1,2) üblich ist.
3345 Der Betreiber des FD besitzt für bestimmte Rollenauthentisierungen für die HSM
3346 Authentisierungschipkarten inkl. PIN, die gematik ebenfalls. Nur bei gleichzeitiger

3347 Authentisierung beider Beteiligten erlauben die HSM bestimmte
3348 Aktionen/Funktionalitäten.

3349 6.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität

3350 Bevor ein E-Rezept-Client die E-Rezept-VAU-Identität für die Verschlüsselung seiner
3351 Request (vgl. 6.2.3- E-Rezept-VAU-Request und -Response) verwendet, muss der Client
3352 deren X.509-Zertifikat prüfen ([gemSpec_eRp_FdV#A_19739],
3353 [gemILF_PS_eRP#A_20769]).

3354 Ein E-Rezept-Client auf einen Praxisverwaltungssystem (PVS) oder einen
3355 Apothekenverwaltungssystem (AVS) muss die TSL als Prüfbasis für solch eine Prüfung
3356 verwenden [gemILF_PS_eRP#A_20764].

3357
3358 Da ein E-Rezept-FdV nur wenige TI-Zertifikate prüfen muss, wird im FdV die im
3359 Folgenden beschriebene technisch deutlich einfachere Zertifikatsprüfung auf Basis der TI-
3360 X.509-Root verwendet, und nicht die Prüfung über die TSL wie im Client eines PVS oder
3361 AVS [gemILF_PS_eRP#A_20764]. Beide Prüfwege sind sicherheitstechnisch äquivalent.

3362 A_21216 - E-Rezept-Client, Zertifikatsprüfung auf TSL-Basis

3363 Ein E-Rezept-Client, der nicht das E-Rezept-FdV ist, MUSS das VAU-Zertifikat vom E-
3364 Rezept-FD beziehen (vgl. A_20160-*, URL /VAUCertificate) und ebenfalls für dieses
3365 Zertifikat die OCSP-Response für dieses Zertifikat beziehen (vgl. A_20160-*, URL
3366 /VAUCertificateOCSPResponse). Er MUSS das Zertifikat mittels TUC_PKI_018 (OCSP-
3367 Graceperiod=12h, PolicyList={oid_erp-vau}) prüfen und dabei die vom FD bezogene
3368 OCSP-Response verwenden.[<=]

3369 Der E-Rezept-FD muss für die Zertifikatsprüfung im E-Rezept-FdV eine (außer für die
3370 Verfügbarkeit) sicherheitsunkritische Unterstützungsleistung erbringen und dafür folgend
3371 benannte Zertifikate zum Download anbieten.
3372 Zunächst erfolgen mit Tab_KRYPT_ERP_Zertifikatsliste eine Hilfsdefinition.
3373

3374 Tabelle 17: Tab_KRYPT_ERP Definition Datenstruktur PKI-Zertifikatsliste

```
{  
  "add_roots" : [ "base64-kodiertes-Root-Cross-Zertifikat-1", ... ],  
  "ca_certs" : [ "base64-kodiertes-Komponenten-CA-Zertifikat-1", ... ]  
}
```

3375
3376 Das E-Rezept-FdV muss in die Lage versetzt werden, alle Zertifikate auf seinen
3377 vorinstallierten Vertrauensanker hin zu validieren. Dafür werden die CA-Zertifikate der
3378 Komponenten PKI (CA-Certs) benötigt.

3379 Der E-Rezept-Fachdienst stellt diese dem E-Rezept-FdV zur Verfügung.

3380 A_24465 - E-Rezept-Fachdienst - Bereitstellung von CA-Zertifikaten

3381 Der E-Rezept-Fachdienst MUSS die CA-Zertifikate (ca_certs) über den Endpunkt GET
3382 /PKICertificates?currentRoot dem E-Rezept-FdV bereitstellen. Der E-Rezept-
3383 Fachdienst MUSS, um die CA-Zertifikate zu ermitteln, die TSPServices der TSL.xml nach
3384 folgenden Kriterien filtern:

3385 • ServiceInformation.ServiceTypeIdentifier ==
3386 "http://uri.etsi.org/TrstSvc/Svctype/CA/PKC"

3387 • ServiceInformation.ServiceInformationExtensions.Extension.ExtensionOID ==
3388 "1.2.276.0.76.4.202" OR "1.2.276.0.76.4.203"

3389 Der E-Rezept-Fachdienst MUSS alle Zertifikate
3390 aus `ServiceDigitalIdentity.DigitalId.X509Certificate` der in der Ergebnismenge
3391 vorhandenen TSPServices extrahieren und unter "ca_certs" base64-kodiert dem Array
3392 hinzufügen und bereitstellen. [\leq]

3393 Das E-Rezept-FdV hat genau ein Root Zertifikat der TI installiert. Die in der
3394 Ergebnismenge der CA-Zertifikate ausgelieferten Zertifikate basieren nicht zwangsläufig
3395 auf den im FdV installierten Vertrauensanker. Daher gibt das E-Rezept-FdV dem E-
3396 Rezept-Fachdienst die SubjectCN des installierten Vertrauensanker mit.

3397 Der E-Rezept-Fachdienst fügt der Response dann alle Cross Zertifikate an, die das E-
3398 Rezept-FdV benötigt, um alle CA-Zertifikate validieren zu können.

3399 **A_24466 - E-Rezept-Fachdienst - Bereitstellung von Cross Zertifikaten**
3400 Der E-Rezept-Fachdienst MUSS die Cross Zertifikate (`add_roots`) über den Endpunkt `GET`
3401 /`PKICertificates?currentRoot` dem E-Rezept-FdV bereitstellen. Der E-Rezept-
3402 Fachdienst MUSS, um die benötigten Cross Zertifikate zu ermitteln, den SubjectCN aus
3403 dem URL-Parameter "currentRoot" extrahieren und dann wie folgt vorgehen:
3404 Der E-Rezept-Fachdienst MUSS, falls es in der Ergebnismenge der `ca_certs` Zertifikate
3405 gibt, die nicht per Signaturprüfung auf den angegebenen Root-Schlüssel vom E-Rezept-
3406 FdV rückführbar sind, nacheinander die chronologisch auf einander folgenden und
3407 vorausgehenden base64-kodierten Root-Cross-Zertifikate, bspw.:
3408 • GEM.RCA4->GEM.RCA5, ... GEM.RCA(x)->GEM.RCA(x+1)
3409 • GEM.RCA4 -> GEM.RCA3, ... GEM.RCA(x)->GEM.RCA(x-1)

3410 am Ende des Arrays "add_roots" anfügen, solange bis alle CA-Zertifikate über einen der
3411 Root-Schlüssel prüfbar, d.h. per Signaturprüfung auf den vom E-Rezept-FdV
3412 angegebenen Root-Schlüssel rückführbar sind. [\leq]

3413

3414 Damit das E-Rezept-FdV die Zertifikatsprüfung durchführen kann werden in regelmäßigen
3415 Zeitintervallen OCSP Responses für die im E-Rezept-FdV Truststore befindlichen
3416 Zertifikate benötigt. Der E-Rezept-Fachdienst stellt diese über einen Endpunkt `GET`
3417 /`OCSPResponse?issuer-cn&serial-nr` bereit. Über diese Angaben erstellt der E-Rezept-
3418 Fachdienst einen OCSP Request in der TI oder gibt die nach geltenden Caching-Regeln im
3419 Cache enthaltene OCSP Response zurück.

3420

3421 **Tabelle 18 : API von GET /OCSPResponse**

Beispiel URL	<code>GET /OCSPResponse?issuer-cn=GEM.KOMP-CA3&serial-nr=123124</code>
URL-Parameter "serial-nr"	Seriennummer (Serial Number) des zu prüfenden Zertifikats

Beispiel URL	GET /OCSPResponse?issuer-cn=GEM.KOMP-CA3&serial-nr=123124
URL-Parameter "issuer-cn"	Name des Ausstellenden Zertifikats (Issuer Common Name)
Response	OCSP Response nach RFC-6960

3422

3423 **A_24467 - E-Rezept-Fachdienst - Bereitstellung von OCSP Responses**

3424 Der E-Rezept-Fachdienst MUSS über den Endpunkt GET /OCSPResonse?issuer-
3425 cn&serial-nr einen OCSP Request nach [RFC-6960] mit dem Issuer Zertifikat aus der
3426 TSL erstellen und an den OCSP Responder zustellen. Die OCSP Response wird an den
3427 anfragenden Client zurückgeben.

3428 Falls eine entsprechende OCSP Response bereits im Cache vorhanden ist, kann diese
3429 zurückgegeben werden, wenn diese nach geltenden Caching-Regeln noch aktuell
3430 ist.[<=]

3431 **A_24468 - E-Rezept-Fachdienst - Caching von OCSP Responses**

3432 Der E-Rezept-Fachdienst MUSS angefragte OCSP Responses von GET
3433 /OCSPResonse?issuer-cn&serial-nr bis zu 4 Stunden cachen und bei einer
3434 entsprechend passenden OCSP-Anfrage, anstatt neu den OCSP-Responder anzufragen,
3435 die im Cache befindliche OCSP-Antwort ausliefern.[<=]

3436 Das E-Rezept-FdV ist nun in der Lage Zertifikate der PKI zu validieren.

3437 **A_24469 - E-Rezept-FdV - Installierter Vertrauensanker**

3438 Das E-Rezept-FdV MUSS eine noch mindestens fünf Jahre gültige ECC ROOT-CA
3439 aus <https://download.tsl.ti-dienste.de/ECC/ROOT-CA/> als Vertrauensanker im Programm-
3440 Code bzw. mit dem Programm-Code fest assoziiert enthalten und als Basis für die
3441 Prüfung von TI-Zertifikat verwenden.[<=]

3442 **A_24470 - E-Rezept-FdV - Truststore mit TI-Zertifikaten**

3443 Das E-Rezept-FdV MUSS einen TI-Zertifikate-Truststore enthalten und pflegen, der
3444 folgende fünf Kategorien beinhaltet:

- 3445
- (A) Root-Schlüssel und Cross-Zertifikate,
 - 3446 • (B) CA-Zertifikate,
 - 3447 • (C) E-Rezept-VAU-Zertifikat,
 - 3448 • (D) IDP-Zertifikat(e).
 - 3449 • (E) E-Rezept-Fachdienst Signaturzertifikat

3450 Initial enthält dieser Truststore nur das vorinstallierte Root-Zertifikat.

3451 [<=]

3452 **A_25058 - E-Rezept-FdV- Befüllen des Truststores**

3453 Das E-Rezept-FdV MUSS für den Fall, dass keine Zertifikate der Kategorie (C) und (D)
3454 vorhanden sind den Truststore mittels des Algorithmus
3455 Tab_KRYPT_ERP_FdV_Truststore_aktualisieren initial befüllen.[<=]

3456 **A_25059 - E-Rezept-FdV - OCSP Responses für TI-Zertifikate**

3457 Das E-Rezept-FdV MUSS bei der Prüfung von TI-Zertifikaten OSCP Responses für die
3458 Zertifikate aus (C), (D) und (E) holen, wenn kein weniger als 12h alter OSCP Response
3459 vorliegt.[<=]

3460 **A_25060 - E-Rezept-FdV - Prüfung OSCP Responder Zertifikate**
3461 Das E-Rezept-FdV MUSS die OSCP-Responder-Zertifikate prüfen und sicherstellen, dass
3462 diese Zertifikate per Signaturprüfung auf ein Zertifikat der Kategorie (B) rückführbar
3463 sind.[<=]

3464 Hinweis: Eine gültige OSCP-Antwort bedeutet, dass die Überprüfung des OSCP-Zertifikats
3465 erfolgreich abgeschlossen wurde. Dies impliziert, dass der Status des zu
3466 überprüfenden(TI-Zertifikats als "good" eingestuft wurde, was bestätigt, dass es nicht
3467 widerrufen ist und weiterhin als vertrauenswürdig gilt.

3468 **A_25061 - E-Rezept-FdV - Prüfung von TI-Zertifikaten der Kategorie (C) oder**
3469 **(D)**
3470 Das E-Rezept-FdV MUSS bei der Prüfung von TI-Zertifikaten der Kategorie (C) oder (D) in
3471 fachlichen Use-Cases prüfen, ob das Zertifikat in den Kategorien (C) oder (D) des
3472 Truststores enthalten ist und eine gültige, weniger als 12 Stunden alte OSCP-Response
3473 vorliegt.
3474 Ist dies nicht der Fall, gilt das Ergebnis der Prüfung des TI-Zertifikats als "FAIL".[<=]

3475 **A_25062 - E-Rezept-FdV - Prüfung von TI-Zertifikaten der Kategorie (E)**
3476 Das E-Rezept-FdV MUSS bei der Prüfung von TI-Zertifikaten der Kategorie (E) in
3477 fachlichen Use-Cases prüfen, ob das Zertifikat in der Kategorie (E) des Truststores
3478 enthalten ist und eine gültige, weniger als 12 Stunden alte OSCP-Response vorliegt.
3479 Falls das Zertifikat nicht im Truststore enthalten ist, muss das E-Rezept-FdV prüfen, ob
3480 das Zertifikat

- 3481 • zeitlich gültig ist,
- 3482 • per Signaturprüfung auf einen CA-Schlüssel aus (B) rückführbar ist,
- 3483 • das Zertifikat eine OID mit dem Wert oid_erezept [gemSpec_OID] enthält
- 3484 • und eine gültige, weniger als 12 Stunden alte OSCP-Response vorliegt.

3485 Ist dies der Fall, wird das Zertifikat in die Kategorie (E) des Truststores aufgenommen.
3486 Ist dies nicht der Fall, gilt das Ergebnis der Prüfung des TI-Zertifikats als "FAIL".[<=]

3487 **A_25063 - E-Rezept-FdV - Truststore aktualisieren**
3488 Das E-Rezept-FdV MUSS im Falle der fehlgeschlagenen Zertifikatsprüfung einmalig den
3489 Truststore mittels des Algorithmus `Tab_KRYPT_ERP_FdV_Truststore_aktualisieren`
3490 aktualisieren und die Prüfung des Zertifikates wiederholen.[<=]

3491

3492 **Tabelle 19: Tab_KRYPT_ERP_FdV_Truststore_aktualisieren**

(1)
Alle Zertifikate im Truststore müssen gelöscht werden mit Ausnahme des vorinstallierten Root-Zertifikats.
Der Truststore besitzt, wie in (A_24470 - E-Rezept-FdV - Truststore mit TI-Zertifikaten) definiert, Prüfschlüssel/Zertifikate in folgenden Kategorien (A) Root-Schlüssel und Cross-Zertifikate, (B) CA-Zertifikate, (C) E-Rezept-VAU-Zertifikat, (D) IDP-Zertifikat(e), (E) E-Rezept-Fachdienst Signaturzertifikat.

(2)
Falls die Zertifikatsliste im Array "add_roots" aus `GET /PKICertificates` nicht leer ist, so wird das erste Cross-Zertifikat im Array auf Basis des vorinstallierten Root Zertifikats per Signaturprüfung geprüft. Ebenfalls wird geprüft, ob der bestätigte CommonName dem Muster "GEM.RCA" + Zahl entspricht. Falls beide Prüfungen mit positiven Prüfergebnis erfolgen konnten, so wird das Zertifikat in Kategorie (A) des Truststores aufgenommen. Weitere Zertifikate im Array werden analog mit dem jeweils vorherigen Cross-Zertifikat per Signaturprüfung und CommonName-Prüfung geprüft und bei positiven Prüfergebnissen in Kategorie (A) des Truststores aufgenommen.

(3)
Für jedes Zertifikat im Array "ca_certs" aus `GET /PKICertificates` wird überprüft, ob es zeitlich gültig ist, ob es per Signaturprüfung auf einen Root-Schlüssel aus (A) rückführbar ist und ob der bestätigte CommonName dem Muster "GEM.KOMP-CA" + Zahl entspricht. Wenn alle drei Prüfungen ein positives Prüfergebnis liefern, so wird das Zertifikat in Kategorie (B) des Truststores aufgenommen.

(4)
Für das VAU-Zertifikat aus `GET /VAUCertificate` und die IDP-Zertifikate vom IDP-Dienst wird überprüft ob es zeitlich gültig ist und ob es per Signaturprüfung auf einen CA-Schlüssel aus (B) rückführbar ist. Wenn nicht beide Prüfungen ein positives Prüfergebnis liefern, so wird das Zertifikat verworfen.
Anderen falls wird geprüft, ob das Zertifikat eine OID mit dem Wert `oid_erp-vau [gemSpec_OID]` enthält. Dann wird es im Truststore bei Kategorie (C) eingefügt. Falls das Zertifikat eine OID mit dem Wert `oid_idpd [gemSpec_OID]` enthält. Dann wird es im Truststore bei Kategorie (D) eingefügt.

3493

3494 Die Spezifikation für die Bereitstellung von Zertifikaten des IDP-Dienstes sind in siehe
3495 [gemSpec_IDP_Dienst#4] beschrieben.

3496 **A_21222 - E-Rezept-Client, allgemein Zertifikatsprüfung**

3497 Ein E-Rezept-Client MUSS bevor er TI-X.509-Zertifikate in fachlichen Abläufen (bspw.
3498 VAU-Kanal) verwendet, diese Zertifikate prüfen (vgl. A_21216 und A_21218). [`<=`]

3499 **6.2.3 E-Rezept-VAU-Request und -Response**

3500 **A_20161-01 - E-Rezept-Client, Request-Erstellung**

3501 Ein E-Rezept-Client MUSS, falls ihm noch kein gültiges E-Rezept-VAU-Zertifikat vorliegt,
3502 ein solches nach den fachlichen Vorgaben von A_20160-* beziehen (`/VAUCertificate`).

3503 Ein E-Rezept-Client MUSS sicherstellen, dass gültige Sperrinformation (OCSP-Response mit
3504 Sperrstatus "good") für das Zertifikat vorliegen, die maximal 12 Stunden alt sind. Liegen diese nicht
3505 vor so MUSS der Client ein Verbindungsaufbau auf VAU-Protokoll-Ebene ablehnen/unterbinden.

3506
3507 Ein E-Rezept-Client MUSS bei der Request-Erstellung folgende Schritte durchführen.

- 3508 1. Er erzeugt einen HTTP-Request, den er an die VAU senden möchte, als
3509 Datenstruktur (vgl. Beispiele nach dieser Anforderung).

- 3510 2. Er erzeugt zufällig eine 128-Bit lange hexadezimalkodierte Request-ID (also 32
3511 Zeichen, Buchstaben a-f kleingeschrieben).
- 3512 3. Er erzeugt zufällig einen 128-Bit AES-Schlüssel (im Weiteren auch
3513 Antwortschlüssel genannt), den er hexadezimal kodiert (also 32 Zeichen,
3514 Buchstaben a-f kleingeschrieben).
- 3515 4. Er MUSS die Request-ID und den AES-Schlüssel für jeden HTTP-Request an die
3516 VAU zufällig neu erzeugen.
- 3517 5. Er erzeugt die folgende Zeichenkette p mit
3518 $p = "1" + " " + \text{JWT-Authentisierungstoken} + " " + \text{Request-ID} + " " + \text{AES-}$
3519 $\text{Schlüssel} + " " + \text{Datenstruktur aus Schritt 1.}$
- 3520 6. Die Zeichenkette p MUSS mittels des ECIES-Verfahrens [SEC1-2009] und mit
3521 folgenden Vorgaben verschlüsselt werden:
- 3522 a. Er MUSS ein ephemeres ECDH-Schlüsselpaar erzeugen und mit diesem und
3523 dem VAU-Schlüssel aus A_20160-* ein ECDH gemäß [NIST-800-56-A]
3524 durchführen. Das somit erzeugte gemeinsame Geheimnis ist Grundlage für
3525 die folgende Schlüsselableitung.
- 3526 b. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis
3527 von SHA-256 verwenden.
- 3528 c. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in
3529 der Formulierung von [RFC-5869] `info="ecies-vau-transport"`.
- 3530 d. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-
3531 Key für die Verwendung von AES/GCM ableiten.
- 3532 e. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig
3533 erzeugen.
- 3534 f. Er MUSS mit dem CEK und dem IV mittels AES/GCM p verschlüsseln, wobei
3535 dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
- 3536 g. Er MUSS das Ergebnis wie folgt kodieren: `chr(0x01) || <32 Byte X-Koordinate`
3537 `von öffentlichen Schlüssel aus (a) > || <32 Byte Y-Koordinate> || <12 Byte`
3538 `IV> || <AES-GCM-Chiffre> || <16 Byte AuthenticationTag>` (vgl. auch
3539 `Tab_KRYPT_ERP` und folgende die Beispielverschlüsselung).
3540 Die Koordinaten sind (wie üblich) vorne mit `chr(0)` zu padden solange bis sie
3541 eine Kodierungslänge von 32 Byte erreichen.
- 3542 7. Er erzeugt einen HTTPS-Request an den FD mit der POST-Methode und dem Pfad
3543 `/VAU/<Nutzerpseudonym>[/optional-beliebiger-weiterer-URL-Pfadteil]` mit
3544 dem Content-Type 'application/octet-stream' und sendet diesen an die
3545 Webschnittstelle des FD.
3546 "Nutzerpseudonym" MUSS eine ggf. aus der vorherigen (zeitlich letzten) Antwort
3547 des FD dem Nutzer übergebene URL-sichere Zeichenkette sein (bspw. ein 128
3548 Byte langer Hexadezimal-Kode).
3549 Falls dem Client kein Nutzerpseudonym vorliegt so MUSS er "0" als
3550 Nutzerpseudonym verwenden.
- 3551 [**<=**]
- 3552

3553 **Tabelle 20: Tab_KRYPT_ERP Kodierung des Chiffrats aus A_20161-***

Offset	Länge in Bytes	Erläuterung
0	1	Versionsfeld
1	32	X-Koordinate öffentlicher ephemer Sender-ECDH-Schlüssel
33	32	Y-Koordinate öffentlicher ephemer Sender-ECDH-Schlüssel
65	12	IV zufällig pro Nachricht zu erzeugen (A_20161-* Punkt e)
77	gleich Länge des Plaintextes (= LP)	"eigentliche" AES-GCM-Chifftrat
77 + LP	16	128 Bit langer Authentication-Tag

3554

3555 Hinweise zu A_20161-*:

- 3556 1. Beispiel für eine hexadezimalkodierte Request-ID nach A_20161-* (2):
3557 "32b6594cc29bb54a14cb2a8e09558817"
- 3558 2. Beispiel für einen 128-Bit-AES-Schlüssel nach A_20161-* (3):
3559 "a576daad8096fc52987250a8e7eb9bd3"
- 3560 3. Aus kryptographischer Sicht könnte ein Client den Antwort-AES-Schlüssel
3561 (A_20161-* Punkt (3)) auch für weitere Requests verwenden. Dies würde jedoch
3562 erzwingen, dass es im Client eine komplexere Überwachung des Lebenslaufs des
3563 Schlüssels gibt (Wann wurde er erzeugt, d. h. wann muss er gewechselt werden
3564 etc.). Um dies zu verhindern und die Implementierung im Client einfacher zu
3565 halten, gibt es A_20161-* Punkt (4).

3566 Beispielverschlüsselung (Testvektor):

3567 Der Langzeit-VAU-Schlüssel nach A_20160-* sei folgender:

3568 $x=0x8634212830dad457ca05305e6687134166b9c21a65ffebf555f4e75dfb048888$

3569 $y=0x66e4b6843624cbda43c97ea89968bc41fd53576f82c03efa7d601b9facac2b29$

3570 Die zu Verschlüsselnde Nachricht sei "Hallo Test" (10 Zeichen).

3571 Der Client erzeugt zufällig den privaten ECC-Schlüssel

3572 $d=5bbba34d47502bd588ed680dfa2309ca375eb7a35ddb67cc7f8b6b687a1c1d$

3573 Damit ist dessen öffentlicher ephemerer ECDH-Schlüssel:

3574 $x=0x754e548941e5cd073fed6d734578a484be9f0bbfa1b6fa3168ed7fffb22878f0f$

3575 $y=0x9aef9bbd932a020d8828367bd080a3e72b36c41ee40c87253f9b1b0beb8371bf$

3576

3577 Nach ECDH ergibt sich damit folgendes gemeinsames Geheimnis:

3578 $9656c2b4b3da81d0385f6a1ee60e93b91828fd90231c923d53ce7bbbcd58ceaa$

3579 Nach Schlüsselableitung (info="ecies-vau-transport") erhält der Client folgende CEK:
3580 23977ba552a21363916af9b5147c83d4

3581 Der Client erzeugt zufällig einen 12 Byte großen IV:
3582 257db4604af8ae0dfced37ce

3583 Die AES/GCM-Chiffre berechnet aus der Nachricht, dem CEK und dem IV folgendes
3584 Chifftrat:

3585 86c2b491c7a8309e750b und folgenden 16 Byte großen Authentication
3586 Tag 4e6e307219863938c204dfe85502ee0a

3587

3588 Das Gesamt-Chifftrat vollständig kodiert ist damit (ohne Leerzeichen, als Hexdump)

3589 01 754e548941e5cd073fed6d734578a484be9f0bbfa1b6fa3168ed7fffb22878f0f
3590 9aef9bbd932a020d8828367bd080a3e72b36c41ee40c87253f9b1b0beb8371bf
3591 257db4604af8ae0dfced37ce 86c2b491c7a8309e750b
3592 4e6e307219863938c204dfe85502ee0a

3593

3594 In der Webschnittstelle muss ein CMAC-Schlüssel für die Bildung der Nutzerpseudonyme
3595 vorliegen (A_20162-*). Dieser Schlüssel wird regelmäßig gewechselt (A_20162-*) und er
3596 kann in der Webschnittstelle als normales Datenobjekt (also nicht innerhalb des HSMs)
3597 vorliegen (Schutzbedarf bezüglich Vertraulichkeit: mittel).

3598 **A_20162 - E-Rezept-FD, Webschnittstellen, VAU-Requests**

3599 Der Fachdienst E-Rezept MUSS an seinen Webschnittstellen folgendes sicherstellen:

- 3600 1. Er MUSS unter dem Pfad /VAU/<Nutzerpseudonym>[/optional-beliebiger-
3601 weiterer-URL-Pfadteil] (der URL) mit dem Content-Type 'application/octet-
3602 stream' HTTPS-Requests entgegennehmen (nach dem Nutzerpseudonym kann u.
3603 Um. ein "/" und anschließend weitere Pfadangaben vom Client angegeben werden,
3604 vgl. Beispiele unten).
- 3605 2. Er MUSS in der Webschnittstelle über einen AES-CMAC 128 Bit Schlüssel
3606 verfügen, der mindestens alle 10 Tage zufällig neu erzeugt wird. Dieser Schlüssel
3607 MUSS als reiner Software-Schlüssel (nicht in einem HSM) in der Webschnittstelle
3608 vorliegen.
- 3609 3. Er MUSS das Nutzerpseudonym (NP) auf Integrität (CMAC) prüfen (vgl. Schritt 8).
3610 Ist die Integrität nicht gegeben, so MUSS er anstatt des übergebenen NP "0" als
3611 Wert verwenden.
- 3612 4. Er MUSS anhand des NP eine Lastverteilung innerhalb des FD und eine NP-
3613 zentrierte Lastbeschränkung durchführen.
3614 Im Lastszenario MÜSSEN Requests mit NP "0" mindestens 10 mal geringer
3615 priorisiert werden, als Requests mit gültigem NP.
- 3616 5. Er muss den Request zur Abarbeitung an einen geeigneten Verarbeitungskontext
3617 der VAU übergeben.
- 3618 6. Die VAU-Instanz muss eine verschlüsselte Antwort (vgl. A_20163) erzeugen und
3619 an die Schnittstelle senden.
- 3620 7. In der Antwort der VAU-Instanz MUSS die VAU das Prenutzerpseudonym (PNP,
3621 vgl. A_20163) als Teil der Antwort der VAU auf den Nutzer-Request an die
3622 Webschnittstelle übergeben.

- 3623 8. Er MUSS mittels des CMAC-Schlüssels (vgl. Schritt 2) den 128-Bit-lange CMAC-
3624 Wert des PNP erzeugen und diesen hexadezimal kodieren (= CMAC). Die
3625 Zeichenkette "<PNP>" + "-" + "<CMAC>" sei das NP.
- 3626 9. Als Antwort MUSS die Schnittstelle eine HTTP-Response senden mit dem Content-
3627 Type 'application/octet-stream', der Antwort der VAU-Instanz als Bytestrom
3628 (Octet-Stream) und im HTTP-Response-Header MUSS 'Userpseudonym: <NP>'
3629 enthalten sein.

3630 [**<=**]

3631 Beispiele für mögliche Pfade die nach A_20162 von einem Client erzeugt werden
3632 könnten:

3633 /VAU/0
3634 /VAU/0/Task/create
3635 /VAU/270810c79748768a9b0aefbf52c8d72be7ad5e0d2d328d9bb70dbf58623fc7ae

3636 **A_20163 - E-Rezept-VAU, Nutzeranfrage, Ent- und Verschlüsselung**

3637 Die E-Rezept-VAU MUSS das Folgende sicherstellen und im Falle eines Fehlschlagens die
3638 Abarbeitung des Requests mit einer entsprechenden Fehlermeldung an die sie aufrufende
3639 Webschnittstelle abbrechen.

- 3640 1. Die E-Rezept-VAU MUSS einen verschlüsselten Nutzer-Request von der
3641 Webschnittstelle entgegennehmen.
- 3642 2. Die E-Rezept-VAU MUSS einen verschlüsselten Nutzer-Request nach den
3643 kryptographischen Vorgaben aus A_20161-* und mit dem privaten Schlüssel aus
3644 A_20160-* versuchen zu entschlüsseln.
- 3645 3. Die E-Rezept-VAU MUSS den erhaltenden Klartext p auf den Strukturaufbau aus
3646 A_20160-* prüfen.
- 3647 4. Die E-Rezept-VAU MUSS das JWT-Authentisierungstoken auf Gültigkeit prüfen.
- 3648 5. Die E-Rezept-VAU MUSS den in p kodieren HTTP-Request abarbeiten.
- 3649 6. Die E-Rezept-VAU MUSS einen 128-Bit-AES-CMAC-Schlüssel zufällig erzeugen und
3650 mindestens alle 10 Tage wechseln.
- 3651 7. Die E-Rezept-VAU MUSS aus dem "sub"-Feld-Wert mittels des CMAC-Schlüssels
3652 den 128 Bit langen CMAC-Wert berechnen und hexadezimal kodieren (32 Byte
3653 lang). Dies sei das Prenutzerpseudonym (PNP).
- 3654 8. Die Antwort "a" auf den HTTP-Request aus p MUSS wie folgt kodiert werden:
3655 a="1" + " " + Request-ID-aus-p + " " + Response-Header-und-Body.
- 3656 9. Die E-Rezept-VAU MUSS a mittels des 128-Bit langen AES-Schlüssels aus p und
3657 AES/GCM (96 Bit zufällig erzeugter IV, 128 Authentication Tag) verschlüsseln und
3658 erhält c'.
- 3659 10. Die E-Rezept-VAU MUSS c' und das PNP an die Webschnittstelle als Antwort
3660 übergeben.

3661 [**<=**]

3662 **A_20174 - E-Rezept-Client, Response-Auswertung**

3663 Ein E-Rezept-Client MUSS bei der Response-Auswertung (vgl. vorgehenden Client-
3664 Request aus A_20161-*) folgende Schritte durchführen. Dabei MUSS der Client bei
3665 Fehlschlagens im Folgenden aufgeführten Prüfungen die Analyse der Response
3666 abbrechen, und er MUSS die Request-ID und den AES-Antwortschlüssel sicher löschen.

1. Er MUSS prüfen, ob der Content-Type der Response 'application/octet-stream' ist.
2. Wenn im Response-Header die Variable "Userpseudonym" vorhanden ist, so MUSS er den Wert von "Userpseudonym" als NP für den nächsten Request an die VAU verwenden. (Der Client MUSS einen ggf. vorhandenen alten Wert des NP im Client überschreiben.)
3. Er MUSS das Antwort-Chifftrat mit den Vorgaben aus A_20163 (9) und dem AES-Antwort entschlüsseln und prüfen ob die Entschlüsselung erfolgreich möglich war.
4. Er MUSS prüfen, ob die Struktur des erhaltenen Klartextes p der Struktur aus A_20163 (8) entspricht.
5. Er MUSS prüfen, ob die Request-ID in p der Request-ID aus dem Client-Request entspricht (Gleichheit prüfen).
6. Er MUSS das dritte Feld-Element in p ("Response-Header-und-Body") als HTTP-Antwort der E-Rezept-VAU in fachlich weiter verarbeiten.

[<=]

A_20175 - E-Rezept-Client, Speicherung Nutzerpseudonym

Ein E-Rezept-Client MUSS das im Request verwendete Nutzerpseudonym (NP) in Software speichern (kein HSM/TPM/SE) und das NP ausschließlich für seinen Einsatzzweck der E-Rezept-VAU-Kommunikation verwenden. Insbesondere MUSS der Client die Vertraulichkeit des NP wahren (bspw. nicht unnötig in Protokolleinträgen und Fehlermeldungen aufführen).[<=]

Der Fachdienst E-Rezept besitzt eine REST-Schnittstelle, d. h. Fehler werden mittels HTTP-Status/Fehler-Codes signalisiert. Die in der folgenden Tabelle (Tab_KRYPT_VAUERR) aufgeführten Fehler kann ein E-Rezept-Client in Bezug auf die in diesem Abschnitt definierte kryptographische Sicherung zwischen Client und VAU treffen.

Tabelle 21: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept)

Fehlerfall	HTTP-Response-Code der E-Rezept
JWT-Client-Token ungültig	403 Forbidden
Entschlüsselung des Chiffrats des äußeren Requests schlug fehl	400 Bad Request
Strukturaufbau des Klartextes aus A_20160-* ist falsch	400 Bad Request

6.2.4 Zufallsquelle für Clients

Zur Auffrischung des Entropie-Pools des Clients kann der Client von verschiedenen Diensten der TI Zufall ausreichender Güte (vgl. GS-A_4367) beziehen. Der FD-E-Rezept ist dafür eine Quelle von mehreren. Ein Client muss verschiedene unabhängige Quellen abfragen und die Zufallsdaten kryptographisch geeignet (bspw. über den Fortuna-PRNG-Algorithmus) zusammenführen.

A_21215 - E-Rezept-FD, Random-Operation

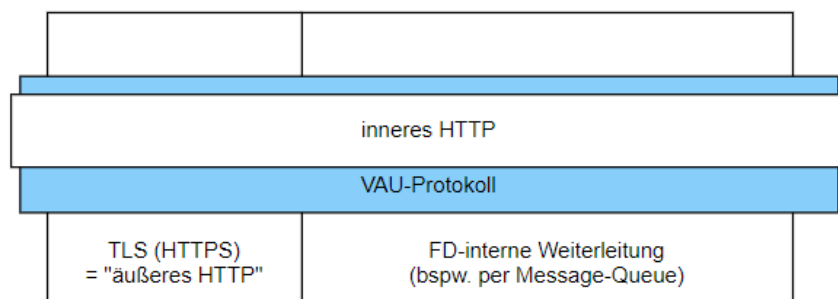
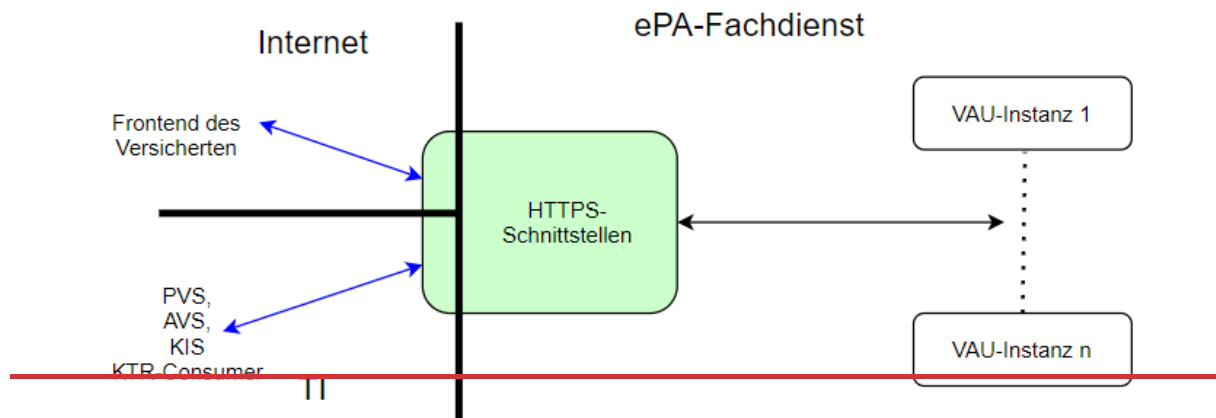
3700 Der Fachdienst E-Rezept MUSS an seiner Webschnittstelle (HTTPS) unter der URL
3701 /Random (GET-Methode) 128 Byte Zufallsdaten hexadezimalkodiert (Lower-Case -- [0-
3702 9a-f]) einen Client zur Verfügung stellen. Bei jedem Request MÜSSEN die Zufallsdaten
3703 neu erzeugt werden. Die Response MUSS den Content-Type application/json besitzen.
3704 Teil einer Beispiel-Response:
3705
3706 Content-length: 258
3707 Content-type: application/json
3708 Date: Tue, 01 Dec 2020 12:46:18 GMT
3709 Server: nginx/1.14.0 (Ubuntu)
3710
3711 "a5dc9d13ee2e76ddd9b75e9c28421fc4b5a9a131751a3dad1203f8d1b149366ef938163d43
3712 718f31fe5464e05f236ba62588cea48ff8cdb9f77abe52a03a389f8a2573127c70629742387
3713 14e457399cfc9fcd7eeb656c2cfd3bf50fb1d74b4cd5c73607283533f423760c2e38a3fd646
3714 602ef244d4dbdb332c8f696b5e07ef51"
3715 [**<=**]

3716 Um die Anforderung umzusetzen ist es im Normalfall ausreichend /dev/urandom als
3717 Zufallsquelle auf einen Linux-Server zu verwenden.

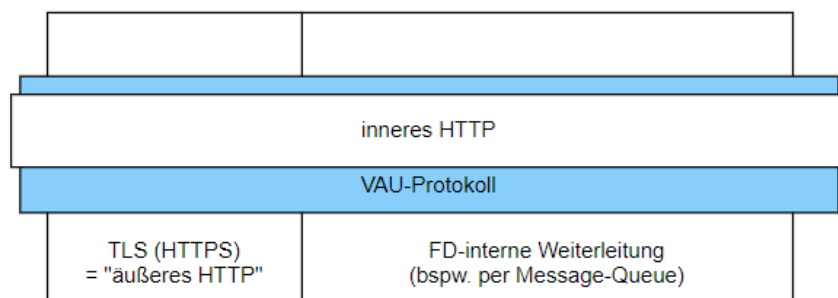
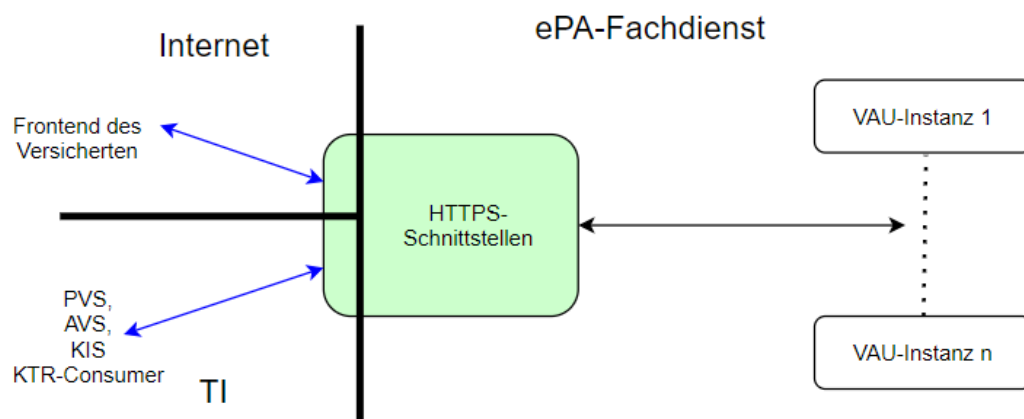
3718

7 VAU-Protokoll für ePA für alle

3719 Der grundsätzliche architektonische Aufbau des ePA-Aktensystems als Fachdienst bei ePA
3720 für alle entspricht dem bei ePA 1.x, 2.x und beim E-Rezept: Das Aktensystem besteht
3721 aus mehreren Webschnittstellen, bei denen über das HTTPS-Protokoll Requests
3722 eintreffen. Diese Requests werden im Aktensystem an verschiedene VAU-Instanzen zur
3723 Verarbeitung weitergeleitet. Die Antwort von einer VAU-Instanz erzeugt und an die
3724 Webschnittstelle übergeben. Von dieser wird sie per HTTPS an die Clients innerhalb einer
3725 HTTP-Response übermittelt. Zwischen ePA-Client und VAU gibt es also keine
3726 durchgehende TLS-Verbindung -- eine TLS-Verbindung des Clients endet an einer
3727 Webschnittstelle des Aktensystems.
3728 Das VAU-Protokoll erzeugt einen durchgehenden (also ununterbrochenen) Kanal
3729 zwischen ePA-Client und VAU.
3730



3731



3732

Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-VAU-Instanz

Bei ePA 1.x (und ePA 2.x), beim E-Rezept und bei ePA für alle werden jeweils unterschiedliche Authentisierungsarten für die Client-Authentisierung verwendet. Die Client-Authentisierung durchläuft quasi eine Evolution, der das VAU-Protokoll durch Anpassung folgen muss, damit die Sicherheitsziele weiterhin erreicht werden. Bei ePA 1.x und 2.x war es möglich, dass Nutzer des Aktensystems direkt über private AUT-Schlüssel Verbindungsparameter signieren konnten. Beim E-Rezept wird aktuell eine Variante des OIDC-Protokolls verwendet, bei der ein E-Rezept-Client direkt das Auth-Token lokal vorliegen hat. Solche Auth-Token verwendet er bei der Authentisierung im VAU-Protokoll beim E-Rezept-FD. Bei ePA für alle gibt es einen Authentication-Flow nach [RFC-7636] (OAuth 2.0 PKCE). Dort hat ein Client keinen Zugriff mehr auf das Auth-Token, d. h. das VAU-Protokoll aus dem E-Rezept in der aktuellen Ausbaustufe kann nicht verwendet werden. Deshalb wird für ePA für alle eine neue Version des VAU-Protokolls eingeführt. In einer späteren Ausbaustufe wechselt auch beim E-Rezept die Authentisierung auf diese OIDC-Variante -- ab dann wird bei ePA für alle und bei E-Rezept dasselbe VAU-Protokoll verwendet.

Wenn über eine HTTPS-Schnittstelle ein Request eines ePA-Clients eintrifft, steht es einem Aktensystem frei, über welches Protokoll es diesen Request an die VAU-Instanzen verteilt (bspw. per Message-Queue), d. h. die Aktensysteminterne Verbindungsstrecke zwischen HTTPS-Schnittstellen und VAU-Instanzen ist frei durch den Aktensystemhersteller wählbar.

A_24654 - ePA: HTTP-Version

Ein ePA-Aktensystem MUSS an seinen HTTPS-Schnittstellen, i. S. v. Schnittstellen, die ein ePA-Client anspricht, mindestens HTTP Version 1.1 unterstützen. [**<=**]

Hinweis: Im Kontext ePA bietet HTTP/2 nur geringe Vorteile gegenüber HTTP/1.1. HTTP/2 besitzt jedoch eine deutlich höhere Komplexität. Ein Hersteller ist mit A_24654-* nicht verpflichtet, eine HTTP-Version größer als 1.1 zu implementieren. A_24654 ist nicht dahingehend zu verstehen, dass von einem Aktensystem perfekte Konformität zu HTTP/1.1 umzusetzen ist. Im Kontext der ePA-Funktionalität wird nur der Umfang von HTTP verwendet, den man normaler Weise bei Webservices verwendet.

Wenn im folgenden innerhalb dieses Kapitels 7 vom VAU-Protokoll gesprochen wird, ist stets die Variante für ePA für alle gemeint.

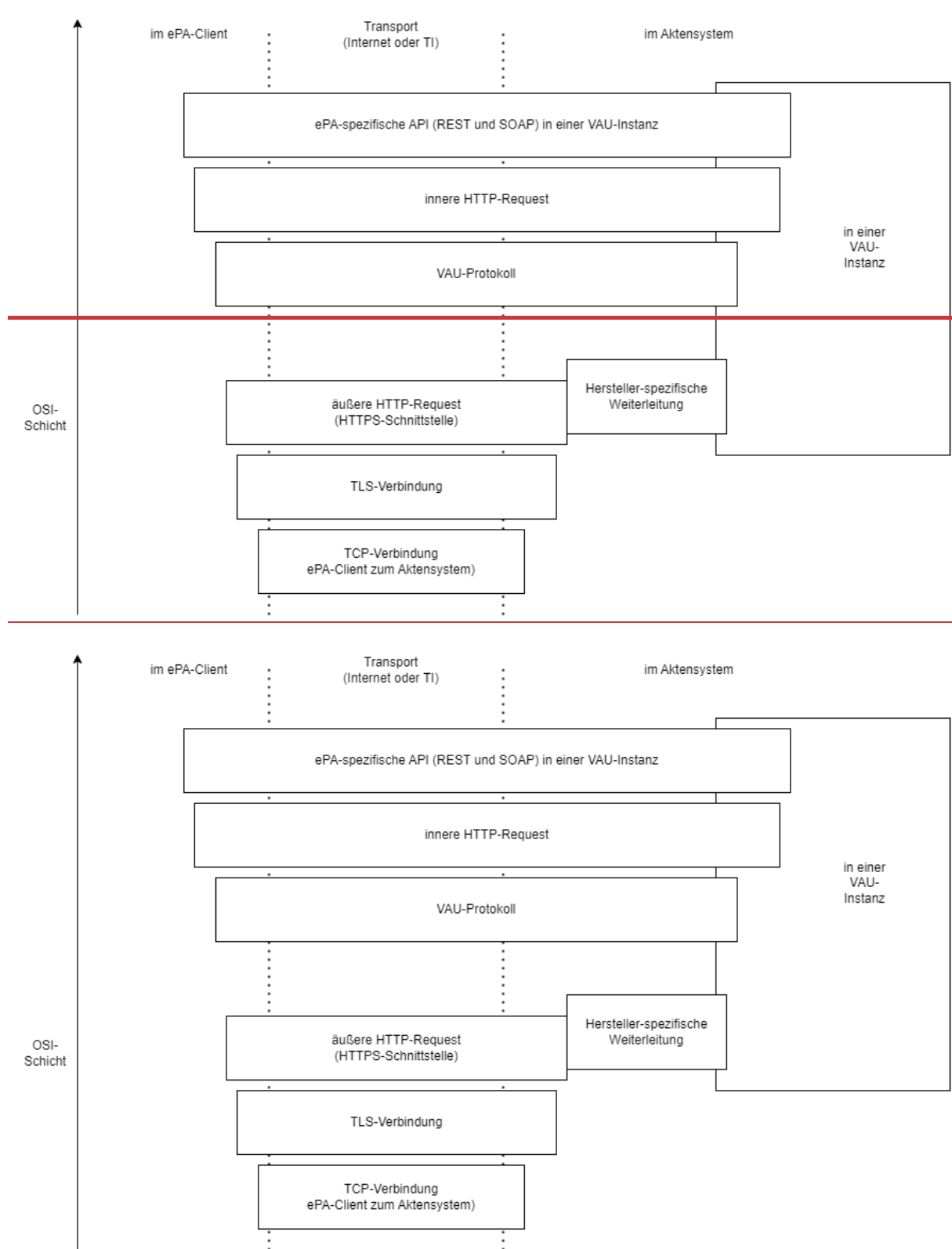


Abbildung 5: OSI-Schichten in einem Aktensystem

3770 Wie im OSI-Referenzmodell üblich, ist das VAU-Protokoll agnostisch gegenüber dem über
3771 ihm transportierten Daten oder Protokollen. Das VAU-Protokoll ermöglicht einen Ende-zu-
3772 Ende gesicherten Transport von Daten, bei dem es irrelevant ist, welche Protokolle unter
3773 oder über ihm in Bezug auf das OSI-Modell verwendet werden.

3774 **A_24656 - ePA, Unabhängigkeit von TLS-Ebene und VAU-Protokoll-Ebene**

3775 Ein ePA-Aktensystem und ein ePA-Client MÜSSEN sicherstellen, dass die TLS-Ebene
3776 unabhängig zur VAU-Protokoll-Ebene ist, i. S. v. es gibt keine Bindung zwischen TLS-
3777 Session und VAU-Protokoll-Verbindung.

3778 Es MUSS möglich sein, die TLS-Verbindung, über die erfolgreich ein VAU-Protokoll-
3779 Handshake stattgefunden hat, zu beenden und auf einer neuen unabhängigen TLS-
3780 Verbindung weiter das VAU-Protokoll mit schon ausgehandelten Verbindungsschlüsseln
3781 zu verwenden. [<=]

3782 Das VAU-Protokoll ermöglicht (zunächst) eine einseitig authentifizierte
3783 Schlüsselaushandlung zwischen Client (FdV, LEI-PVS/AVS/KIS, KTR-Consumer) und
3784 Server (VAU-Instanz). Ähnlich wie bei einer TLS-Verbindung werden nach einer
3785 Schlüsselaushandlungsphase symmetrische Schlüssel gemeinsam berechnet und diese
3786 symmetrischen Schlüssel werden anschließend für die Sicherung der nun folgenden
3787 Nutzdaten auf Anwendungsebene verwendet. Direkt nach erfolgreich abgeschlossener
3788 Schlüsselaushandlung verwendet ein Client innerhalb der etablierten VAU-Protokoll-
3789 Verbindung auf einer höheren OSI-Schicht das OIDC-Protokoll für die Nutzer-
3790 Authentisierung gegenüber der VAU-Instanz. Ist die Authentifizierung erfolgreich
3791 gewesen, vermerkt die VAU-Instanz bei den ausgehandelten symmetrischen Schlüsseln
3792 diesen Authentisierungsstatus in den Metadaten für diese Verbindungsschlüssel. Ab dann
3793 besteht eine beidseitig authentifizierte über das VAU-Protokoll gesicherte Verbindung
3794 insbesondere mit den gemeinsam berechneten symmetrischen Verbindungsschlüsseln.
3795 Das VAU-Protokoll verwendet kryptographisch für die Schlüsselaushandlung das KEM-
3796 TLS-Protokoll ([KEM-TLS], [IETF-KEM-TLS]). Die Kodierung der KEM-TLS-Nachrichten
3797 muss spezifisch für die gewählte Architektur bei ePA für alle angepasst werden. Für die
3798 Schlüsselaushandlung wird ein PQC-sicheres Hybridverfahren auf Basis von Kyber768
3799 und ECDH analog zu [IETF-Hybrid-TLS] verwendet. Die Verwendung von PQC-sicheren
3800 Hybridverfahren findet man ebenfalls bei Messengern [PQC-Hybrid-Signal] oder bei Web-
3801 Browsern [PQC-Hybrid-Chrome]. Das VAU-Protokoll erreicht Forward-Secrecy bei der
3802 Schlüsselaushandlung -- eine etwaige Kompromittierung von Langzeit-Schlüsseln einer
3803 VAU beeinflusst nicht die Sicherheitseigenschaften von VAU-Verbindungen der
3804 Vergangenheit. Nach der Schlüsselaushandlung wählt ein Client zufällig für jeden Request
3805 eine Request-ID, die der Server in der Response aufführt. Somit kann ein Client Request-
3806 Response-Paare sicher zuordnen und kann ebenfalls Replay-Angriffe abwehren. Im
3807 Vergleich zur VAU-Protokoll-Variante aus Abschnitt 6 (E-Rezept aktuelle Ausbaustufe)
3808 wird ca. ab der vierten Nutzdaten-Nachricht ein Geschwindigkeitsvorteil erreicht, weil für
3809 die direkte Sicherung der Nutzdaten keine asymmetrischen Verfahren mehr verwendet
3810 werden (vgl. im Gegensatz dazu A_20161-* Punkt 6). Die Forward-Secrecy und die Post-
3811 Quantum-Resistenz wird bei der VAU-Protokoll-Variante aus Abschnitt 6 nicht erreicht --
3812 es wird also mit der Anpassung des VAU-Protokolls, die aufgrund der
3813 Authentisierungsveränderung unabdingbar ist, sowohl eine Verbesserung der
3814 Sicherheitseigenschaften als auch der Performanz erreicht.

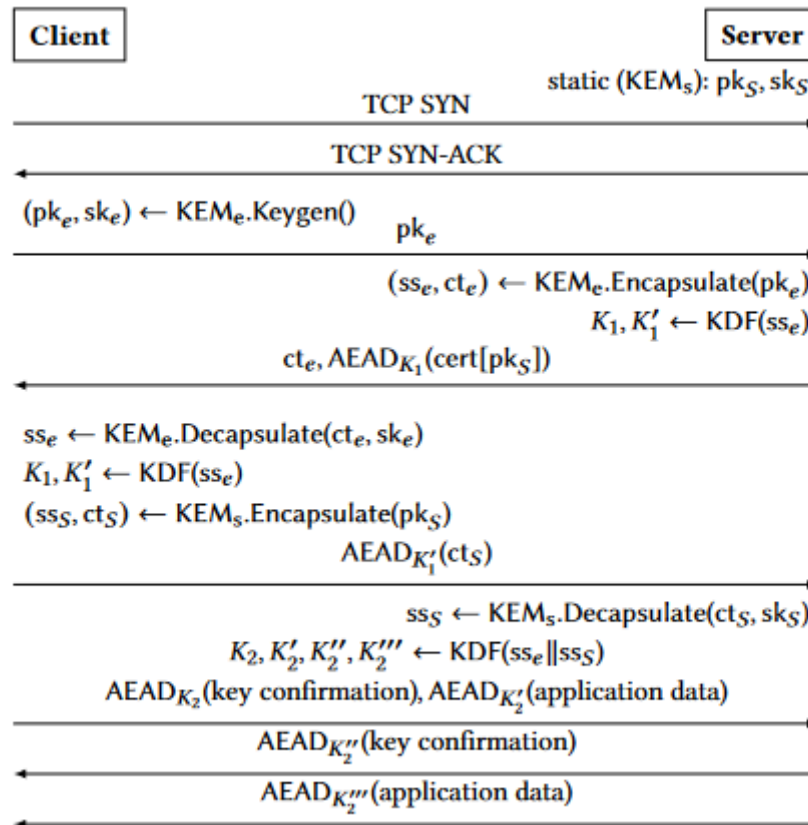


Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS]

Die gematik stellt Beispielimplementierungen des VAU-Protokolls in verschiedenen Programmiersprachen zur Verfügung.

7.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung

Bei im Prinzip allen in der Praxis verwendeten kryptographischen Protokollen gibt es mehrere Phasen im Protokoll: Zunächst erfolgt eine Schlüsselaushandlung bei der hauptsächlich asymmetrische kryptographische Verfahren eingesetzt werden. Als Ergebnis der Schlüsselaushandlung werden symmetrische Schlüssel (AES-Schlüssel) von Client und Server gemeinsam berechnet. Diese symmetrischen Schlüssel werden anschließend für die kryptographische Sicherung der Nutzdaten verwendet. Es werden wie üblich je nach Kommunikationsrichtung (Client->Server, Server->Client) unterschiedliche Schlüssel verwendet (analog zu TLS, IPsec/IKE, SSH etc.). Die symmetrischen Schlüssel sind dann nutzerspezifisch (und je nach gewählter Implementierungs-Architektur im Aktensystem auch VAU-Instanz-spezifisch). Sie sind nicht aktenspezifisch. D. h., dieselben schon ausgehandelten symmetrischen Schlüssel können für Zugriffe auf mehrere Akten bspw. durch ein PVS verwendet werden.

3834 Die Schlüsselaushandlung beim VAU-Protokoll erfolgt nach [KEM-TLS] und benötigt vier
3835 Nachrichten (zwei Round-Trips), die zwischen Client und VAU ausgetauscht werden. Der
3836 kryptographische Verbindungsaufbau ist in [KEM-TLS] genauer beschrieben und ebenfalls
3837 die kryptographische Funktion der einzelnen Schlüsselableitungen. Im Folgenden wird
3838 eine vereinfachte Übersicht als Verständnishilfe aufgeführt.
3839

Nachricht	Aktion
Nachricht 1	<p>Client -> VAU</p> <p>Ein Client erzeugt zwei Schlüsselpaare (ECDH, Kyber768) (analog zu [IETF-Hybrid-TLS]) und sendet die beiden öffentlichen Schlüssel an die VAU (= Nachricht 1).</p> <p>Bei diesen Schlüsselpaaren handelt es sich also um ephemere Schlüssel des Clients, d. h. pro Verbindungsaufbau/Handshake werden diese vom Client neu erzeugt.</p>
Nachricht 2	<p>VAU -> Client</p> <p>Die VAU verwendet die öffentlichen Schlüssel jeweils mittels eines KEM (Encapsulate), um zwei gemeinsame Geheimnisse (ss_e_ecdh, ss_e_kyber768) und zwei KEM-Chiffre zu erzeugen. Diese Geheimnisse fließen in eine KDF (RFC-5869/SHA-256) ein (Basis: ss_e = ss_e_ecdh ss_e_kyber768) und damit werden die symmetrischen Schlüssel K1_c2s und K1_s2c erzeugt. Wie bei TLS 1.3 üblich, werden diese Schlüssel im weiteren für die Sicherung von Daten in der Aushandlungsphase des Protokolls für die Kommunikation zum Server (K1_c2s) und zum Client (K1_s2c) verwendet.</p> <p>Die KEM-Chiffre und über AES/CGM verschlüsselte semi-statische VAU-Schlüssel (A_24425-*) werden an den Client gesendet (= Nachricht 2).</p> <p>Verständnishinweis: Die authentifizierte Verschlüsselung der öffentlichen VAU-Schlüssel (A_24425-*) erfolgt aufgrund des Grundprinzips bei TLS 1.3 möglichst früh, um Daten auch in der Handshake-Phase zu verschlüsseln. Die öffentlichen VAU-Schlüssel (A_24425-*) sind nicht vertraulich.</p>

Nachricht	Aktion
Nachricht 3	<p>Client -> VAU</p> <p>Mittels der KEM-Chifftrate aus Nachricht 2 und der privaten Schlüssel des Clients (vgl. Nachricht 1: ECDH, Kyber768) berechnet der Client $K1_c2s$ und $K1_s2c$ und kann die öffentlichen VAU-Schlüssel erfolgreich entschlüsseln (AEAD-Chiffre). Er verwendet die öffentlichen VAU-Schlüssel (ECDH, Kyber768) und erhält mittels zwei KEM-Encapsulate-Berechnungen zwei Geheimnisse und zwei KEM-Chifftrate. Er berechnet aus den zwei Geheimnissen: $ss_s = ss_s_ecdh \parallel ss_s_kyber768$. Diese KEM-Chifftrate verschlüsselt er über AES/GCM mittels des Schlüssels $K1_c2s$ und erhält ein Ergebnis-Chifftrat.</p> <p>Mittels einer KDF auf Basis von $ss = ss_e \parallel ss_s$ werden abgeleitet: vier vertrauliche AES/GCM-Schlüssel:</p> <ul style="list-style-type: none"> • $K2_c2s_key_confirmation$, • $K2_c2s_app_data$, • $K2_s2c_key_confirmation$, und • $K2_s2c_app_data$ <p>und eine (nicht vertrauliche) KeyID.</p> <p>Der Client erzeugt den Hashwert des Transskript der zuvor ausgetauschten Daten (analog TLS). Diesen Hashwert verschlüsselt er mittels des Schlüssels $K2_c2s_key_confirmation$ (Ziel: Key-Confirmation).</p> <p>Der Client sendet das Ergebnis-Chifftrat ($K1_c2s$) und das Key-Confirmation-Chifftrat ($K2_c2s_key_confirmation$) an die VAU (= Nachricht 3).</p>

Nachricht	Aktion
Nachricht 4	<p>VAU -> Client</p> <p>Mittels der zwei KEM-Chifftrate aus Nachricht 3 und der KEM-Decapsulation-Funktionen werden die beiden Geheimnisse <code>ss_s_ecdh</code> und <code>ss_s_kyber768</code> berechnet, und anschließend <code>ss_s = ss_s_ecdh ss_s_kyber768</code> berechnet. Analog wie im Client bei Nachricht 3 werden die dort aufgeführten vier vertraulichen AES/GCM-Schlüssel und die nicht-vertrauliche KeyID abgeleitet auf Basis von <code>ss = ss_e ss_s</code>.</p> <p>Mit Hilfe des Schlüssels <code>K2_c2s_key_confirmation</code> und der im Server gespeicherten vorherigen Nachrichten prüft der Server das vom Client gesendete Transskript-Chifftrat bzw. den Transskript-Hash darin.</p> <p>Der Server erzeugt selbst einen Transskript-Hash (Hashwert der Konkatenation Nachricht 1 Nachricht 2 Nachricht 3) und sendet diesen mittels <code>K2_s2c_key_confirmation</code> verschlüsselt an den Client zurück.</p>

3840

3841 Client und VAU prüfen die jeweiligen Key-Confirmation (jeweils mittels der Schlüssel
3842 `K2_c2s_key_confirmation` und `K2_s2c_key_confirmation`), bevor sie die Schlüssel
3843 `K2_c2s_app_data` und `K2_s2c_app_data` in der nächsten Phase des Protokolls --
3844 Sicherung der Nutzdaten -- verwenden.

3845

3846 **A_24425-01 - VAU-Protokoll: VAU-Schlüssel für die VAU-Protokoll- 3847 Schlüsselaushandlung**

3848 Ein Aktensystem MUSS sicherstellen, dass

- 3849 1. es eine Signatur-Identität aus der Komponenten-PKI der TI gibt, die technisch
3850 sichergestellt ausschließlich nur von VAU-Instanzen verwendbar ist (AUT-Zertifikat
3851 und Schlüsselmaterial (VAU-HSM) wie bei ePA 1x. und 2.x).
- 3852 2. es semi-statische Schlüsselpaare für ECDH (auf Basis Kurve P-256 [FIPS-186-5])
3853 und Kyber768 [IEFT-Kyber] gibt, deren private Schlüssel, technisch sichergestellt,
3854 ausschließlich von VAU-Instanzen verwendbar sind.
- 3855 3. die privaten Schlüssel in einer VAU-Instanz erzeugt und verarbeitet werden (also
3856 nicht im VAU-HSM),
- 3857 4. die semi-statischen Schlüssel eine maximale Lebensdauer von einem Monat
3858 besitzen (Hinweis: die Forward-Secrecy hängt nicht vom Wechselintervall ab,
3859 innerhalb eines Verbindungsaufbaus und der Schlüsselaushandlung dabei fließen
3860 ephemere Schlüsselwerte von Client und Sever ein).

- 3861 5. die semi-statischen Schlüssel in einer über die Signatur-Identität authentisierten
3862 folgenden Datenstruktur aufgeführt werden.

Struktur der signierten semi-statischen öffentlichen VAU-Schlüssel

```
VAU_Keys = {  
    "ECDH_PK" :  
        { "crv" : "P-256",  
          "x" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),  
          "y" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),  
        },  
    "Kyber768_PK" : Binärwert-öffentlicher-Schlüssel-nach-keygen-Spec-Kyber768,  
    "iat" : Erzeugungszeits-Sekunden-Since-Epoch (integer),  
    "exp" : Nicht-mehr-Verwendbar-nach (integer),  
    "comment" : "Erzeugt bei VAU-Instanz xyz, Meta-Info abcd"  
}
```

In "comment" KÖNNEN beliebige Text-Daten aufgeführt werden. Es können weitere Attribute hinzugeführt werden. Ein Client MUSS ihm unbekannte Attribute ignorieren. Diese Struktur wird mittels CBOR [RFC-CBOR] binär kodiert und im Folgenden VAU_Keys_encoded genannt.

Diese binäre Byte-Folge wird in folgende Datenstruktur eingebracht

```
{  
    "signed_pub_keys" : VAU_Keys_encoded,  
    "signature-ES256" : ECDSA-Signatur-SHA-256-analog-RFC-7515 (R||S => 64  
    Byte) binär,  
    "cert_hash" : SHA-256-Wert des "signierenden" AUT-VAU-Zertifikats,  
    "cdv" : Cert-Data-Version (natürliche Zahl, beginnend mit 1,  
    vgl. A_24957-*),  
    "ocsp_response" : OCSP-Response-für-das-VAU-Signaturzertifikat-nicht-  
    älter-als-24-Stunden-DER-Kodierung  
}
```

Diese Datenstruktur wird mittels CBOR binär kodiert (serialisiert). Das Ergebnis der Kodierung wird "signierte öffentliche VAU-Schlüssel" (Plural) genannt.

3863
3864 [**<=**]

3865 **A_24957 - VAU-Protokoll: Verfügbarmachung des AUT-VAU-Zertifikat plus** 3866 **Prüfkette**

3867 Ein Aktensystem MUSS über an seinen Webschnittstellen mittels eines äußeren HTTPS-
3868 Requests per HTTP-GET unter dem Pfadnamen /CertData.<SHA-256-Hashwert-Hex-[0-
3869 9a-f]>-Versionszahl (a-f kleingeschrieben) folgende Datenstruktur zur Verfügung stellen:

```
3870 {  
3871     "cert": DER-kodiertes-AUT-VAU-Zertifikat,  
3872     "ca" : DER-kodiertes-Komponenten-PKI-CA-aus-dem-"cert"-kommt,  
3873     "rca_chain" : [Cross-Zertifikat-1, ..., Cross-Zertifikat-n],  
3874 }  
3875
```

3876
3877 Die Versionszahl MUSS eine natürliche Zahl sein, beginnend mit 1, die es trotz A_24958-

* erlaubt, Fehler in den Daten zu korrigieren (siehe Erläuterung nach A_24957-*).

Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert/kodiert werden und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) an den VAU-Client als Response auf den GET-Request gesendet werden.

In "rca_chain" MÜSSEN alle Cross-Zertifikate in chronologischer Ordnung von RCA5 ausgehend aufgeführt werden, bis die Root-Schlüssel (Cross-Zertifikat) erreicht werden, mit denen das "ca"-Zertifikat bestätigt (signiert) wurde; d. h., sozusagen eine einfach verkettete Liste von Cross-Zertifikaten chronologisch aufsteigend.

[<=]

Erläuterung:

In A_24425-* werden die "signed_pub_keys" mit dem Schlüsselmaterial der AUT-VAU-Identität des Aktensystems authentisiert, analog zum VAU-Protokoll ePA 1.x und 2.x. Dieses Zertifikat ist über die Signaturkettenprüfung der TI-PKI prüfbar. Dafür benötigt der VAU-Client die Zertifikate, die die Prüfkette ausmachen. Als Vertrauensanker besitzt der Client entweder X.509-Root-Zertifikate, und zwar mindestens RCA5 (PU), oder die TSL. Die Konstruktion nach A_24957-* stellt die Zertifikate für die Prüfkette und das AUT-VAU-Zertifikat selbst ("cert"-Feld) zur Verfügung. Aufgrund der Konstruktion kann man davon ausgehen, dass die Daten in solch einer Datei sich nie ändern werden, i. S. v. einmal vom Client per GET bezogen werden und dann ewig gespeichert werden kann. Sollte sich nach einem Deployment der CertData-Datei (i. S. v. einige Clients haben diese Datei schon bezogen) bspw. CertData-e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855-1 herausstellen, dass dort Fehler enthalten sind, dann kann man über die Versionsnummer im Handshake (vgl. "cdv"-Feld, A_24425-*) und der Erstellung einer neuen CertData-e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855-2 diese Fehler im laufenden Betrieb korrigieren.

A_24958 - VAU-Protokoll: VAU-Client Prüfbasis Zertifikatsprüfung

Ein VAU-Client (PVS, ePA-FdV etc.) MUSS als Prüfbasis eine Root-Version (X.509-Root-TI-Zertifikat), die mindestens 2 Jahre alt ist, verwenden oder die TSL. Der VAU-Client MUSS die Operation nach A_24957-* verwenden, um die für die Zertifikatsprüfung von A_24425-* notwendigen Zertifikate zu beziehen. Die bezogenen Zertifikatsdaten MUSS der VAU-Client lokal (zeitlich unbegrenzt) vorhalten (Caching). Bei Erhalt einer Nachricht 2 (A_24608-*) MUSS er zunächst prüfen, ob er die für die Zertifikatsprüfung notwendigen Zertifikate im lokalen Cache vorrätig hat, und falls ja MUSS er diese verwenden.

[<=]

A_24427 - VAU-Protokoll: VAU-Schlüssel für die VAU-Protokoll-Schlüsselaushandlung

Ein Aktensystem KANN in Bezug auf die Erzeugung der signierten öffentlichen VAU-Schlüssel (vgl. A_24425-*) und der Erzeugung und der Vorhaltung der dazu passenden zwei privaten Schlüssel (ECDH, Kyber768) zwischen zwei Modellen wählen:

(1)

Im Normalfall wird eine VAU-Instanz "auf Vorrat" im Aktensystem gestartet. Beim Starten der Instanz erzeugt die Instanz die beiden Schlüsselpaare und signierte diese mittels der im VAU-HSM befindlichen Signaturidentität (gemäß A_24425-*). Die privaten Schlüssel bleiben genau nur in dieser VAU-Instanz.

(2)

Eine VAU-Instanz erzeugt regelmäßig die Schlüsselpaare und anschließend die

3928 Datenstruktur der signierten öffentlichen VAU-Schlüssel. Die privaten Schlüssel werden
3929 auf sichere Weise bei den VAU-Instanzen verfügbar gemacht. [<=]

3930 Erläuterung: Je nach verwendeten Implementierungs-Architektur und den zur
3931 Implementierung verwendeten HSM (die recht unterschiedliche
3932 Schlüsselverteilungsmethoden haben), kann für einen Hersteller ein Modell einfacher
3933 umzusetzen sein als das andere. Die Motivation der Anforderung ist festzustellen, dass
3934 beide Modelle explizit erlaubt sind.

3935 **AA_24757-01 - VAU-Protokoll: Nutzerpseudonym**

3936 Ein VAU-Client MUSS die ~~HTTP~~-Variable "vau-np" aus vorherigen VAU-Protokoll-
3937 Verbindungen speichern und bei den folgenden VAU-Protokoll-Handshakes bei der
3938 Nachricht 1 (vgl. A_24428-*) im HTTP-Request-Header aufführen (Hinweis: zu dem
3939 Zeitpunkt kann es noch keine inneren HTTP-Request geben / Verbindungsaufbau). Initial
3940 hat ein Client diesen Wert noch nicht, dann führt er die Variable nicht im Request-Header
3941 auf.

3942 Ein VAU-Client MUSS nach einer erfolgreichen Client-Authentisierung (vgl.
3943 [gemSpec_Krypt#7.3 und 7.4]) ~~den inneren~~ die innere HTTP-Response-~~Header~~ prüfen, ob
3944 die Variable "VAU-NP" (bzw. vau-np) aufgeführt ist, falls ja MUSS er einen lokal
3945 gespeicherten Wert ggf. aktualisieren (i. S. v. ein Aktensystem (VAU-Instanz) wird dieses
3946 Nutzerpseudonym auch wechseln).

3947 [<=]

3948 Erläuterung: Das VAU-NP erleichtert die Implementierung im Aktensystem. Es ist für die
3949 Implementierung günstig, wenn ein Nutzer mit möglichst vielen seiner Requests (bzw.
3950 neuer VAU-Verbindungen) in derselben VAU-Instanz landet. Die Variable dient dem
3951 Routing nach der HTTPS-Schnittstellen des Aktensystems (Verteilung der Requests auf
3952 die verschiedenen VAU-Instanzen, vgl. Abschnitt 7.5- Routing auf VAU-Instanzen).

3953 Bestimmte Clients (bspw. der E-Rezept-FD in der Rolle als ePA-Client) werden ggf. auf
3954 mehrere VAU-Instanz verteilt, die konkrete Ausgestaltung liegt in der
3955 Entscheidungshoheit des Aktensystemherstellers.

3956 Vgl. auch A_24773-* (Neustart der Verbindung bei unterschiedlichen Nutzergeräten im
3957 Erstnutzungsfall).

3958 **A_24428 - VAU-Protokoll: VAU-Client: Nachricht 1**

3959 Ein VAU-Client MUSS für die Erzeugung folgende Schritte durchlaufen.

3960 Ein VAU-Client MUSS

3961 1. ein ECC-Schlüsselpaar auf der Kurve P-256 [FIPS-186-5] erzeugen.

3962 2. ein Kyber768-Schlüsselpaar [IETF-Kyber] erzeugen.

3963 Anschließend MUSS er die öffentlichen Schlüssel in folgende Datenstruktur überführen

3964 {

3965 "MessageType" : "M1",

3966 "ECDH_PK" : { "crv" : "P-256",

3967 "x" : analog A_24425-*,

3968 "y" : analog A_24425-* },

3969 "Kyber768_PK" : analog zu A_24425-*

3970 }

3971

3972 Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] in eine Bytefolge kodieren
3973 (serialisieren).

3974 Diese Bytefolge ist die Nachricht 1.

3975

3976 Diese Nachricht 1 MUSS der VAU-Client an die HTTPS-Schnittstelle des Aktensystem per
3977 POST auf den Pfad /VAU senden, wobei er den Mime-Type "application/cbor" [RFC-CBOR]
3978 (HTTP Content-Type) verwendet und die Nachricht 1 im Request-Body aufführt.

3979
3980 Vgl. auch A_24757-*
3981 [\leq]

3982 Hinweis: Für ein ePA-FdV ist das Access Gateway im Internet und dessen HTTPS-
3983 Schnittstelle der Zugang zum Aktensystem, an den das ePA-FdV seinen Request (bspw.
3984 A_24428-*) sendet. Ein ePA-Client aus der TI verwendet direkt die äußere HTTPS-
3985 Schnittstelle eines Aktensystems innerhalb der TI (analog zu ePA 1.x und 2x.).

3986 **A_24429 - VAU-Protokoll: VAU-Instanz: Erhalt von Nachricht 1**

3987 Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die für Nachricht 1 erzeugten
3988 Datenobjekte nach A_24428-* verarbeiten können. [\leq]

3989 **A_24619 - VAU-Protokoll: AES/GCM-Verschlüsselung im Handshake**

3990 Eine VAU-Instanz und ein ePA-Client verschlüsseln im Rahmen des Handshakes des VAU-
3991 Protokolls verschiedene Nachrichten-Teile mittels AES/GCM. Dabei MÜSSEN sie pro
3992 Verschlüsselung den IV jeweils zufällig als 96-Bit-Wert erzeugen. Der Authentication-Tag
3993 MUSS 128 Bit lang sein. Das Ergebnis der Verschlüsselung MUSS dann in der folgenden
3994 Kodierung aufgeführt werden:
3995 IV || eigentliche AES-GCM-Chiffre || 128-Bit langer Authentication-Tag. [\leq]

3996 **A_24608 - VAU-Protokoll: VAU-Instanz: Nachricht 2**

3997 Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die beiden öffentlichen Schlüssel aus
3998 Nachricht 1 (vgl. A_24428-*) prüfen (korrekte ECC-Kurve ("crv":"P-256"), öffentlicher
3999 Punkt liegt auf der Kurve P-256 [FIPS-186-5], der öffentliche Kyber768-Schlüssel ist
4000 valide [IETF-Kyber]).

4001 Sie MUSS für ECDH und Kyber768 jeweils die KEM-Encapsulate-Funktion verwenden und
4002 erhält dabei zwei Geheimnisse (ss_e_ecdh, ss_e_kyber768) und zwei Ciphertexte
4003 (ECDH_ct, Kyber768_ct).

4004 Sie MUSS für die beiden Geheimnisse zusammenfügen: ss_e = ss_e_ecdh ||
4005 ss_e_kyber768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256
4006 verwenden (info = " (leere Zeichenkette)), um 64 Byte abzuleiten. Die ersten 32 Byte
4007 (=256 Bit) heißen K1_c2s und die letzten 32 Byte heißen K1_s2c.

4008
4009 Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_24169-*) MÜSSEN die "signierten
4010 öffentlichen VAU-Schlüssel" (vgl. A_24425-*) verschlüsselt werden. Das Ergebnis (vgl.
4011 A_24169-*) heißt aead_ciphertext_msg_2.

4012
4013 Sie MUSS folgende Datenstruktur erzeugen:

```
4014 {  
4015     "MessageType" : "M2",  
4016     "ECDH_ct"      : ... analog ECDH_PK aus A_24428-* ...,  
4017     "Kyber768_ct"  : Kyber768-Ciphertext analog [IETF-Kyber], vgl. Referenz-  
4018 Implementierung Kyber768,  
4019     "AEAD_ct"      : aead_ciphertext_msg_2  
4020 }
```

4021
4022 Diese Datenstruktur MUSS sie mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist
4023 Nachricht 2.

4024
4025 Sie MUSS einen ID erzeugen, kodiert aus der Zeichenmenge
4026 A-Za-z0-9-/

4027 die maximal 200 Zeichen lang ist. Die ID MUSS mit "/" (Slash) beginnen und MUSS ein
4028 gültiger URL-Pfadname sein. Diese ID MUSS es dem Aktensystem ermöglichen, die VAU-
4029 Instanz und den aktuellen Handshake bei Eintreffen der Nachricht-3 des ePA-Clients, der
4030 diese ID mitsendet, zu identifizieren. Das Aktensystem kann die Struktur der ID selbst
4031 definieren. Ein ePA-Client MUSS die ID als opake Zeichenkette behandeln.
4032 Sie MUSS die Nachricht 2 inkl. ID an die HTTPS-Schnittstelle des Aktensystem übergeben
4033 (die Art der Verbindung zwischen VAU-Instanz und HTTPS-Schnittstelle des
4034 Aktensystems kann ein Hersteller frei wählen). Das Aktensystem (bzw. die äußere
4035 HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 1
4036 eingetroffen ist, die Nachricht 2 als Antwort im Response-Body senden. Der zu
4037 verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response
4038 sein. Im Response-Header MUSS mit der HTTP-Header-Variable "VAU-CID" die ID
4039 aufgeführt werden. [<=]

4040 Hinweis: Bei der KEM-Encapsulate-Funktion fließt Zufall aus dem System ein. Für die
4041 VAU-Instanzen (und auch für die ePA-Clients) gelten die Vorgaben aus Abschnitt 2.4
4042 (Güte der Zufallserzeugung, Zuweisung über den Produkttypsteckbrief).

4043 Beispiele für die ID aus A_24608:

- 4044 • /VAU/9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
- 4045 • /1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
- 4046 • /VAU/A-
- 4047 • XYZ/1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
- 4048

4049

4050 **A_24622 - VAU-Protokoll: VAU-Client: Erhalt von Nachricht 2**

4051 Ein VAU-Client MUSS die für Nachricht 2 erzeugten Datenobjekte nach A_24608-*
4052 verarbeiten können.
4053 Er MUSS prüfen, ob

- 4054 1. im HTTP-Response-Header die Variable "VAU-CID" enthalten ist, falls nicht
4055 Abbruch.
- 4056 2. der Wert der Variable eine Bytefolge ist, dessen Länge maximal 200 Byte lang ist
4057 und die nur die Zeichen
4058 A-Za-z0-9-/
4059 enthält und mit "/" (Slash) beginnt. Falls nicht Abbruch.

4060 Er MUSS den Wert als Pfad für das Versenden der Nachricht 3 (vgl. A_24623) und aller
4061 weiteren Nachrichten -- also auch nach dem erfolgreichen Handshake -- im Kontext
4062 dieser Verbindung verwenden. [<=]

4063 **A_24623 - VAU-Protokoll: VAU-Client: Nachricht 3**

4064 Ein VAU-Client MUSS aus der Nachricht 2 (vgl. A_24622-*) der VAU-Instanz mittels der
4065 Ciphertexte ECDH_ct und Kyber768_ct und den privaten ephemeren Client-Schlüsseln
4066 aus A_24428-* (Nachricht 1) und der jeweiligen KEM-Decapsulation-Funktionen zwei
4067 Geheimnisse berechnen: ss_e_ecdh und ss_e_kyber768. Er MUSS die beiden
4068 Geheimnisse zusammenfügen: ss_e = ss_e_ecdh || ss_e_kyber768 und das Ergebnis
4069 mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = " (leere
4070 Zeichenkette), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1_c2s und
4071 die letzten 32 Byte heißen K1_s2c.

4072

4073 Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_24619-*) MUSS der Ciphertext
4074 "AEAD_ct" entschlüsselt werden: die "signierten öffentlichen VAU-Schlüssel" (vgl.

4075 A_24425-*) werden so als Klartext erhalten. Diese VAU-Schlüssel MUSS er nach
4076 A_24624-* prüfen. Mittels der öffentlichen Schlüssel (ECDH_PK, Kyber768_PK) MUSS er
4077 jeweils die KEM-Encapsulation-Funktion ausführen und er erhält zwei Geheimnisse:
4078 ss_s_ecdh und ss_s_kyber768. Diese führt er zusammen: ss_s = ss_s_ecdh ||
4079 ss_s_kyber768. Weiter berechnet er ss = ss_e || ss_s.
4080 Dieses Geheimnis verwendet die HKDF [RFC-5859] auf Basis von SHA-256 (info = "
4081 (leere Zeichenkette)), um 160 Byte (= 5 * 32 Byte) abzuleiten. Er MUSS diese 160 Byte
4082 in 32 Byte-Blöcke (von offset 0 bis zum Ende) auf folgende fünf Variablen (vier Schlüssel
4083 + eine KeyID) verteilen:

- 4084 • K2_c2s_key_confirmation,
- 4085 • K2_c2s_app_data,
- 4086 • K2_s2c_key_confirmation,
- 4087 • K2_s2c_app_data

4088 und

- 4089 • KeyID (nicht vertraulich).

4090 Diese ersten vier sind vertrauliche Schlüsselwerte für AES/GCM. Die KeyID wird nach
4091 dem Handshake als eindeutige ID für die K2*_app_data Schlüssel dienen.

4092
4093 Er MUSS eine Datenstruktur wie folgt erzeugen:

```
4094 {  
4095     "ECDH_ct"      : client_kem_result_2["ECDH_ct"],  
4096     "Kyber768_ct"  : client_kem_result_2["Kyber768_ct"],  
4097     "ERP"          : False,  
4098     "ESO"          : False  
4099 }  
4100
```

4101
4102 ERP steht für "Enforce Replay Protection" und ESO steht für "Enforce Sequence Order".
4103 Innerhalb der Spezifikation heißt diese Datenstruktur Nachricht_3_inner_Layer. Diese
4104 Datenstruktur MUSS er per CBOR [RFC-CBOR] serialisieren/kodieren. Diese
4105 Serialisierung MUSS er mittels K1_c2s verschlüsseln (vgl. A_24619) (= "ciphertext_msg_3").

4106
4107 Er MUSS die komplette Nachricht-1 (CBOR-Kodierung), die Nachricht-2 und
4108 ciphertext_msg_3 konkatenieren (= Transskript des Client) und davon den SHA-256-
4109 Hashwert berechnen. Diesen Hashwert MUSS er mittels K2_c2s_key_confirmation
4110 verschlüsseln (vgl. A_24619), das Chifftrat sei
4111 als aead_ciphertext_msg_3_key_confirmation hier bezeichnet.

4112
4113 Dann MUSS er folgende Datenstruktur erzeugen:

```
4114 {  
4115     "MessageType" : "M3",  
4116     "AEAD_ct"      : ciphertext_msg_3,  
4117     "AEAD_ct_key_confirmation" : aead_ciphertext_msg_3_key_confirmation  
4118 }  
4119
```

4120
4121 Diese Datenstruktur MUSS er per CBOR serialisieren, das Ergebnis ist Nachricht 3.
4122 Er MUSS die Nachricht 3 per äußeren HTTP-Request an das Aktensystem senden und
4123 dabei den Wert der VAU-CID (vgl. A_24622) als URL-Pfadnamen verwenden, unter

4124 Verwendung der HTTP-POST-Methode.

4125 [\leq]

4126 **A_24624-01 - VAU-Protokoll: VAU-Client: Prüfung der "signierten öffentlichen**
4127 **VAU-Schlüssel"**

4128 Ein VAU-Client MUSS die "signierten öffentlichen VAU-Schlüssel" (vgl. A_24425-*) bei
4129 der Verarbeitung von Nachricht 3 (vgl. A_24623-*) wie folgt prüfen. Erhält er bei einem
4130 der folgenden Prüfpunkte kein positives Prüfergebnis, so MUSS er die Verarbeitung
4131 (Handshake) abbrechen.

- 4132 1. Prüfung des TI-Zertifikats, das den Hashwert aus dem "cert_hash"-Datenfeld
4133 besitzt (Bezug des Zertifikats vgl. A_24957-*), u. a. unter der Verwendung der
4134 OSCP-Response aus "ocsp_response" für die Prüfung des Sperrstatus (Prüfung ob
4135 "good"). Die OSCP-Response darf dabei nicht älter als 24 Stunden sein.
- 4136 2. Das VAU/TI-Zertifikat MUSS zeitlich gültig sein. Es MUSS kryptographisch in einer
4137 Zertifikats-/Signaturprüfungskette rückführbar auf eine X.509-Root-Version der
4138 TI-PKI sein.
- 4139 3. Das TI-Zertifikat MUSS aus der Komponenten-PKI der TI stammen (vgl.
4140 Implementierungshinweis A_25192-*#Punkt-2) und die Rollen-OID "oid_epa_vau"
4141 besitzt.
- 4142 4. Die Signatur im "signature-ES256"-Datenfeld MUSS eine valide Signatur für die
4143 Daten im Datenfeld "signed_pub_keys" (Signaturprüfung ergibt "valid/accept")
4144 sein, unter Verwendung des öffentlichen Signatur-Schlüssels aus dem VAU/TI-
4145 Zertifikats bei der Signaturprüfung.
- 4146 5. Der ECC-Schlüssel in signed_pub_keys (vgl. Erzeugung bei A_24425) MUSS ein
4147 gültiger Punkt der Kurve P-256 [FIPS-186-5] sein. Der öffentliche Schlüssel in
4148 "Kyber768_PK"-Datenfeld MUSS ein gültiger Kyber-768-Schlüssel [IEFT-Kyber]
4149 sein.
- 4150 6. Die Zeit in "signed_pub_keys.exp" MUSS größer als die aktuelle Systemzeit
4151 (Seconds since epoch) sein, d. h. die beiden Schlüssel (ECDH_PK und
4152 Kyber768_PK) sind noch zeitlich gültig.

4153 [\leq]

4154 **A_24625 - VAU-Protokoll: VAU-Instanz: Erhalt von Nachricht 3**

4155 Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die für Nachricht 3 erzeugten
4156 Datenobjekte nach A_24623-* verarbeiten können.[\leq]

4157 **A_24626 - VAU-Protokoll: VAU-Instanz: Nachricht 4**

4158 Eine VAU-Instanz MUSS bei Erhalt der Nachricht 3 das Chiffre AEAD_ct mittels des
4159 Schlüssels K1_c2s entschlüsseln. Schlägt dies fehl, MUSS sie den Verbindungsaufbau mit
4160 einem Fehler (vgl. A_24635-*) abbrechen. Sie MUSS analog wie der Client (vgl.
4161 A_24623-*) die Schlüsselerzeugung der folgenden Schlüssel durchführen, nur dass sie
4162 dafür die KEM-Decapsulation-Methode verwendet:

- 4163 • K2_c2s_key_confirmation,
 - 4164 • K2_c2s_app_data,
 - 4165 • K2_s2c_key_confirmation,
 - 4166 • K2_s2c_app_data
- 4167 und
- 4168 • KeyID (nicht vertraulich).

4169 Sie MUSS analog zu A_24623-* das Transskript des Clients und dessen SHA-256-Wert
4170 berechnen. Sie MUSS mittels K2_c2s_key_confirmation das Chifftrat
4171 "AEAD_ct_key_confirmation" aus Nachricht 3 entschlüsseln und prüfen, ob der von ihr (=
4172 VAU-Instanz) berechnete Transskript-Client-Hashwert mit dem entschlüsselten Klartext
4173 übereinstimmt. Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit
4174 zwischen berechnete Transskript-Client-Hashwert und Klartext festgestellt, MUSS sie den
4175 Handshake mit einem Fehler abbrechen (vgl. A_24635-*).
4176 Sie MUSS den Transskript der VAU-Instanz als die Konkettation von Nachricht1 ||
4177 Nachricht 2 || Nachricht 3 berechnen, davon den SHA-256-Hashwert berechnen und
4178 diesen mittels K2_s2c_key_confirmation verschlüsseln (vgl. A_24619-*), und erhält ein
4179 Chifftrat genannt "AEAD_ct_key_confirmation".
4180 Sie MUSS die folgende Datenstruktur erzeugen:

```
4181 {  
4182     "MessageType" : "M4",  
4183     "AEAD_ct_key_confirmation" : Chifftrat-AEAD_ct_key_confirmation  
4184 }
```

4185 Diese Datenstruktur MUSS sie mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist
4186 Nachricht 4.

4187
4188 Sie MUSS die Nachricht 4 inkl. ID (vgl. A_24608-*) an die HTTPS-Schnittstelle des
4189 Aktensystem übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS
4190 als Antwort auf den HTTPS-Request, über den die Nachricht 3 eingetroffen ist, die
4191 Nachricht 4 als Antwort im Response-Body senden. Der zu verwendende Mime-Type
4192 MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-
4193 Header MUSS mit der HTTP-Header-Variable "VAU-CID" die ID aufgeführt werden. [<=]

4194 Erläuterung:

4195 Die Aufführung von "VAU-CID" im Response-Header ist eigentlich nicht mehr absolut
4196 notwendig, da der Client diese schon bei Nachricht 2 erhalten und als URL-Pfadname ab
4197 jetzt im Laufe der weiteren Verbindung verwenden wird (A_24622-*). Die Absicht der
4198 Aufführung ist, eine etwaige Fehlersuche bspw. bei IOP-Tests zu unterstützen.

4199 **A_24627 - VAU-Protokoll: VAU-Client: Erhalt von Nachricht 4**

4200 Ein VAU-Client MUSS die für Nachricht 4 erzeugten Datenobjekte nach A_24626-*
4201 verarbeiten können.

4202
4203 Er MUSS die Nachrichten 1, 2 und 3 konkatenieren und den SHA-256-Hashwert
4204 erzeugen. Dies ist der Transskript-Hashwert. Er MUSS mittels K2_s2c_key_confirmation
4205 das Chifftrat "AEAD_ct_key_confirmation" entschlüsseln (vgl. A_24619-*).
4206 Der erhaltene Klartext ist der gesendete Transskript-Server-Hashwert.
4207 Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen eben berechneten
4208 Transskript-Hashwert und gesendeten Transskript-Server-Hashwert (Klartext des
4209 Chifftrats) festgestellt, MUSS er den Handshake mit einem Fehler abbrechen. [<=]

4210 **7.2 Transport und Sicherung der Nutzdaten**

4211 **A_24629 - VAU-Protokoll: VAU-Client: Verschlüsselungszähler**

4212 Ein VAU-Client MUSS sicherstellen, dass der Schlüssel K2_c2s_app_data als Attribut
4213 einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Der VAU-Client MUSS
4214 sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer
4215 Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es

4216 spielt dabei keine Rolle, ob die Nachricht (Chiffirat) ggf. nicht übermittelt werden konnte,
4217 der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden. [≤]

4218 Hinweis: Ein Zählerüberlauf kann praktisch nie erreicht werden. Würde ein Client im
4219 Nano-Sekunden-Takt den Zähler erhöhen, würde erst nach mehr als 583 Jahren der
4220 Überlauf eintreten. Deshalb wird auf einen Überlauf test verzichtet.

4221 **A_24628-01 - VAU-Protokoll: VAU-Client: Request erzeugen/verschlüsseln**

4222 Ein VAU-Client MUSS einen inneren HTTP-Request als Klartext erzeugen.

4223
4224 Er MUSS den Klartext mittels AES/GCM und dem Schlüssel K2_c2s_app_data
4225 verschlüsseln (siehe AAD weiter unten).

4226 Dafür wird ein 32-Bit Zufallswert a erzeugt. Nach A_24629-* wird der mit
4227 K2_c2s_app_data verbundene Zähler um eins erhöht. Es wird der IV mit IV=a || Zähler
4228 erzeugt (dessen Länge ist damit 96-Bit). Dieser IV wird für die AES/GCM-Verschlüsselung
4229 verwendet.

4230
4231 Er MUSS einen Request-Counter pflegen, der verbunden ist mit der KeyID. Für jeden
4232 Request MUSS der VAU-Client diesen Request-Counter erhöhen, bei Empfang einer
4233 Antwort ist es dem Client möglich, die Response zum zuvor gestellten Request sicher
4234 zuzuordnen (vgl. A_24633). (vgl. Erläuterungen zu A_24628-*).

4235
4236 Es wird folgender Header erzeugt:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffirat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/Request	1 Byte	Für eine Nachricht des ePA-Clients an eine VAU-Instanz wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrats hat dieses Byte den Wert 2, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_24623-*)

4237
4238 Dieser Header stellt die "Additional Associated Data" dar, die in die Berechnung
4239 des Authentication-Tag bei der AES/GCM-Verschlüsselung einfließen MÜSSEN. Der
4240 Authentication-Tag MUSS 128 Bit lang sein.

4241
4242

Das erweiterte Chifftrat (also inkl. Header) MUSS folgende Struktur haben:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chifftrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/Request	1 Byte	Für eine Nachricht des ePA-Clients an eine VAU-Instanz wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chifftrats wird es auf den Wert 2 gesetzt, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_24623-*)
IV	12 Byte (= 96 Bit)	IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A_24628-*)
CT	variabel	eigentliche AES/GCM-Chifftrat, dessen Länge gleich der Länge des Klartextes ist
GMAC-Wert	16 Byte (= 128 Bit)	Authentication-Tag, der während der AES/GCM-Verschlüsselung inkl. der Associated Data (Daten aus der Header-Tabelle, s. o.) berechnet wird.

4243
4244
4245
4246
4247
4248

Der VAU-Client MUSS diese Datenstruktur per HTTP-POST an die VAU-Instanz senden und als URL-Pfadnamen dabei den Wert der VAU-CID (vgl. A_24622-*) verwenden. Dabei MUSS er den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden.

[<=]

4249

Erläuterung:

4250
4251
4252
4253

Der Verschlüsselungszähler, der im Client mit dem Schlüssel K2_c2s_app_data verbunden ist, hat die Funktion für den Galois Counter Mode (GCM) sicherzustellen, dass der jeweils pro Verschlüsselungsvorgang erzeugte Initialisierungsvektor (IV) (bei gleichen Wert von K2_c2s_app_data) einzigartig ist.

4254
4255
4256
4257
4258

Der Request-ID-Zähler hat drei Funktionen. Einmal soll im Client eine Response des Servers sicher einem vorher gesendeten Request zuordenbar sein. Sollte im Verbindungsaufbau "Enforce Replay Protection" aktiviert worden sein, prüft der Server, ob Requests mit gleicher Request-ID mehrfach eingetroffen sind (ähnlich der "Anti Replay Window"-Technik bei IPsec). Wenn analog "Enforce Sequence Order" aktiviert worden ist,

4259 dann prüft der Server die Folge der Request-ID der einkommenden Requests auf strenge
4260 Monotonie.

4261 In der aktuellen Ausbaustufe von ePA für alle werden die letzten beiden Funktionen nicht
4262 benötigt (wie auch aktuell beim E-Rezept nicht).

4263 Da Verschlüsselungszähler und Request-ID unterschiedliche Funktionen/Motivationen
4264 besitzen, sind sie als separate (theoretisch auch im Wert unterschiedliche) Datenobjekte
4265 aufgeführt.

4266 **A_24630 - VAU-Protokoll: VAU-Instanz: Request entschlüsseln/auswerten**
4267 Ein Request erreicht das Aktensystem über eine HTTPS-Schnittstelle. Im äußeren
4268 Request nimmt das Aktensystem das Routing mittels der Verbindungs-ID (URL-
4269 Pfadname, A_24622-* und A_24628-*) und/oder der KeyID im Header des Chiffrats vor.
4270

4271 Eine VAU-Instanz MUSS bei Erhalt eines Chiffrats nach A_24628-* folgendes prüfen.

4272 1. Ist die Länge der Datenstruktur mindestens 72 Byte lang.

4273 2. Ist die "PU/nonPU" Byte korrekt, i. S. v. korrekte Umgebung, in der auch die VAU-
4274 Instanz arbeitet.

4275 3. Ist das Request/Response-Byte gleich 1.

4276 4. Ist die KeyID bekannt.

4277 Sollte eine dieser Prüfungen oder eine weitere der folgenden Prüfungen ein nicht-
4278 positives Prüfergebnis ergeben, so MUSS die VAU-Instanz die Verarbeitung des Requests
4279 abbrechen und mit einer Fehlermeldung (vgl. A_24635-*) dem Client antworten.
4280

4281 Die VAU-Instanz MUSS den mit der KeyID verbundenen Schlüssel K2_c2s_app_data für
4282 die Entschlüsselung des Chiffrats verwenden und dabei analog A_24628-* den Header als
4283 "Additional Associated Data" mit in die Entschlüsselung einfließen lassen. Ergibt die
4284 Entschlüsselung das Symbol "FAIL", so MUSS Abbruch und Fehlermeldung (vgl.
4285 A_24635-*) erfolgen. Die VAU-Instanz MUSS den Request-Counter-Wert speichern, der
4286 für die Antwort (A_24632-*) benötigt wird.
4287 [**<=**]

4288 Hinweis:

4289 "Enforce Replay Protection" und "Enforce Sequence Order" werden in der aktuellen
4290 Ausbaustufe von ePA für alle nicht umgesetzt, weil fachlich noch nicht benötigt.

4291 **A_24631 - VAU-Protokoll: VAU-Instanz: Verschlüsselungszähler**
4292 Eine VAU-Instanz MUSS sicherstellen, dass der Schlüssel K2_s2c_app_data als Attribut
4293 einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Eine VAU-Instanz MUSS
4294 sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer
4295 Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es
4296 spielt dabei keine Rolle, ob die Nachricht (Chifftrat) ggf. nicht übermittelt werden konnte,
4297 der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden. [**<=**]

4298 Verständnishinweis: analog A_24629-*.

4299 **A_24632 - VAU-Protokoll: VAU-Instanz: Response erstellen/verschlüsseln**
4300 In der VAU-Instanz wird der innere HTTP-Request (Ergebnis aus A_24630-*) fachlich
4301 verarbeitet und als Antwort eine innere HTTP-Response erzeugt. Diese ist der Klartext,
4302 der jetzt behandelt wird.
4303

4304 Eine VAU-Instanz MUSS analog zu A_24628-* einen Header erzeugen, wobei sie

- 4305 1. beim Response/Request-Byte den Wert 2 verwenden MUSS,
- 4306 2. im Request-Counter den gespeicherten Wert aus A_24630-* (Eingang des
- 4307 Requests) verwenden MUSS.
- 4308 Die anderen Header-Variablen MÜSSEN analog zu A_24628-* festgelegt werden.
- 4309 Die Konstruktion des IV MUSS analog zu A_24628-* erfolgen, nur dass für den
- 4310 Verschlüsselungszähler der Wert nach A_24631-* (K2_s2_app_data) verwendet werden
- 4311 MUSS.
- 4312 Mit dem IV und dem Schlüssel K2_s2c_app_data MUSS der Klartext (s. o.) verschlüsselt
- 4313 werden mit AES/GCM, wobei der Header als "Additional Associated Data" bei der
- 4314 Verschlüsselung mit einfließt.
- 4315
- 4316 Das so erzeugte erweiterte Chifftrat (vgl. A_24628-*, also Chifftrat inkl. Header) MUSS an
- 4317 den Client im HTTP-Response-Body versendet werden, wobei das Aktensystem den
- 4318 Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden MUSS.
- 4319 [**<=**]
- 4320 **A_24633 - VAU-Protokoll: VAU-Client: Response entschlüsseln/auswerten**
- 4321 Ein VAU-Client MUSS bei Erhalt einer Antwort gemäß A_24632-* auf einen Request
- 4322 gemäß A_24628-* folgendes prüfen:
- 4323 1. Ist die Länge der Datenstruktur (erweiterte Chifftrat) mindestens 72 Byte lang.
- 4324 2. Ist die "PU/nonPU" Byte korrekt, i. sS v. korrekte Umgebung, in der auch der
- 4325 VAU-Client arbeitet.
- 4326 3. Ist das Request/Response-Byte gleich 2.
- 4327 4. Hat der Response-Counter den von Client erwarteten Wert.
- 4328 5. Ist die KeyID bekannt.
- 4329 Ergibt eine der Prüfungen ein nicht-positives Ergebnis, MUSS der Client die Verarbeitung
- 4330 der Response abbrechen.
- 4331 Er MUSS das Chifftrat mit dem mit der KeyID verbundenen Schlüssel K2_s2c_app_data
- 4332 mittels AES/GCM entschlüsseln und dabei den Header als "Additional Associated Data" in
- 4333 die Entschlüsselung mit einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL",
- 4334 so MUSS er die Verarbeitung des Chifftrats/Response abbrechen.
- 4335 [**<=**]

4336 **7.3 OIDC-Authentisierung eines Clients (Nutzer)**

- 4337 Bei den meisten ePA-Clients wird für die Authentisierung des Nutzers der
- 4338 OAuth2/OIDC/PKCE-Authentication-Workflow nach Etablierung der kryptographischen
- 4339 Phase des VAU-Protokolls verwendet.
- 4340

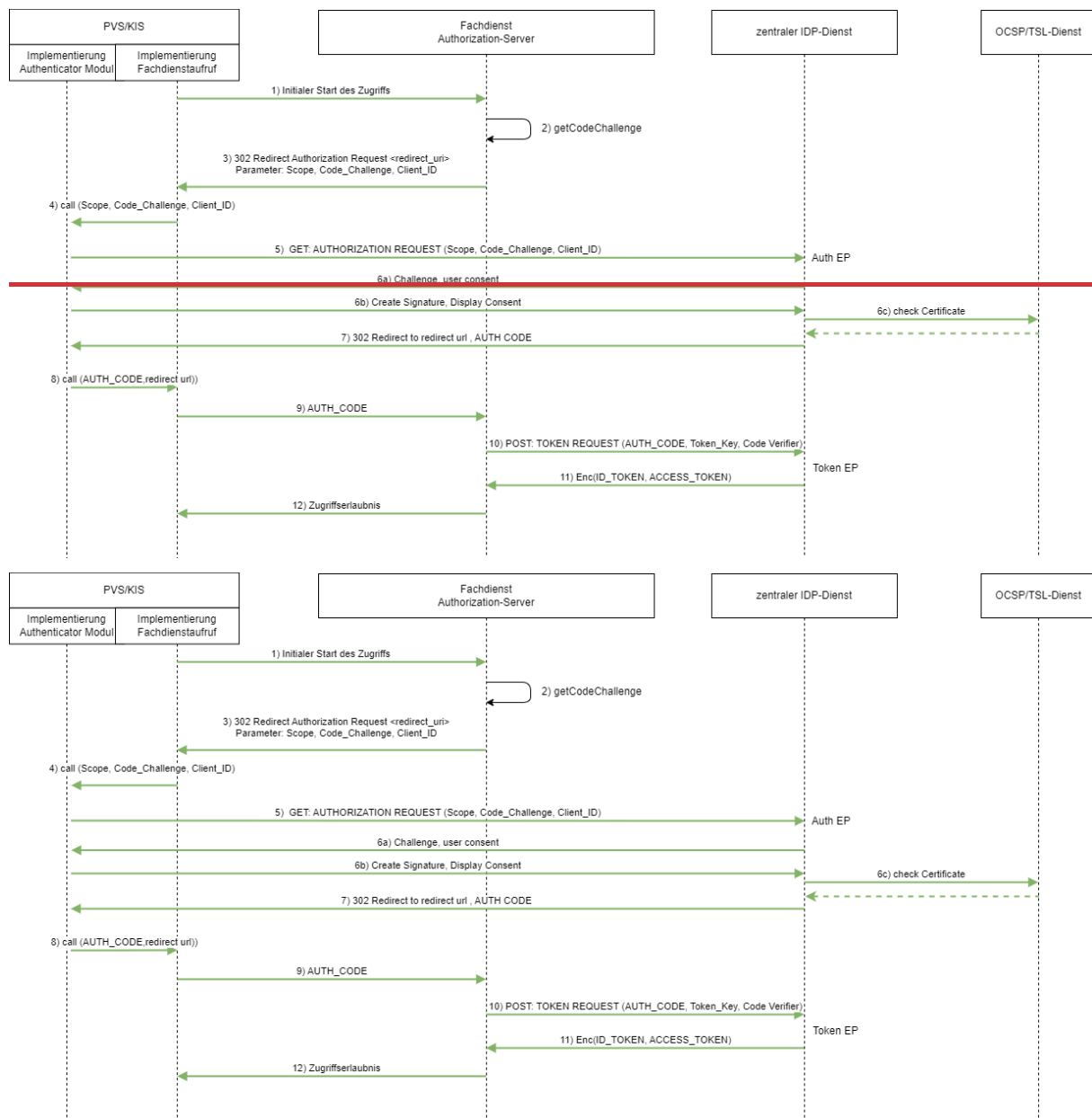


Abbildung 7 : OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI

D. h., der rein kryptographische Teil des VAU-Protokolls hat erfolgreich stattgefunden und zwischen ePA-Client und VAU-Instanz konnten zwei symmetrische Schlüssel in einer Form der Multi-Party-Computation zusammen berechnet werden. Eine einseitig authentifizierte und vertrauliche Datenverbindung besteht zwischen ePA-Client und VAU-Instanz. Über diese Datenverbindung findet auf einer höheren OSI-Schicht eine Client-Authentisierung statt. Dafür werden vom Client innere HTTP-Requests über das VAU-Protokoll an die VAU-Instanz gesendet. Im OIDC Fall zwei Stück:

1. einmal wird ein Hashwert (die "Codechallenge") per GET bezogen und
2. als zweites wird ein "Authentication Code" per POST vom Client an die VAU-Instanz gesendet.

Mittels des Ursprungsbilds der Codechallenge (Zufallswert erzeugt von der VAU-Instanz/ Input-Daten für die Hashwertberechnung) und dem AUTH-Code bezieht die VAU-Instanz ein Identity-Token vom IDP, in dem Identitätsangaben des Clients (KVNR oder Telematik-ID) stehen. Nach Prüfung und Analyse dieses Identity-Tokens kennt die VAU-Instanz die Client-Identität -- eine beidseitig authentifizierte Datenverbindung steht ab dann bereit. Bevor dies erfolgt ist, darf eine VAU-Instanz keine zusätzliche Funktionalität (i. S. v. ePA-spezifische APIs) per innerer HTTP-Requests einem noch unauthentisierten Client verfügbar machen.

A_24634 - VAU-Protokoll: VAU-Instanz: notwendige Authentifizierung des VAU-Clients (Nutzers)

Eine VAU-Instanz MUSS sicherstellen, dass, solange der ePA-Nutzer noch nicht authentisiert ist, keine ePA-Funktionalität (ePA-spezifische APIs, die nicht zur weiteren Client-Authentisierung dienen) vom ePA-Client verwendbar sind (siehe Erläuterung vor A_24634-*).[<=]

~~AA_25055-0102~~ - VAU-Protokoll: OIDC Authentisierung oberhalb der VAU-Protokoll-Schicht

Eine VAU-Instanz MUSS für eine OIDC/OAuth2/PCKE notwendige HTTP Endpunkte per inneren HTTP-Request/Responses einem Client verfügbar machen (Codechallenge per GET zur Verfügung stellen, Authentication Code per POST vom Client empfangen ~~machen~~).

Nach erfolgreicher Authentisierung über OIDC MUSS die VAU-Instanz in den Metadaten der Verbindungsschlüssel (KeyID, K2_*_app_data) den neuen Authentisierungszustand vermerken (Rückwirkung auf A_24634-*).

Nach erfolgreicher Authentisierung über OIDC, ~~genauer innerhalb der~~ MUSS die VAU-Instanz ein Nutzerpseudonym (NP) in der Response ~~(innerer Request) nach dem des inneren POST-Request/Requests~~ für den Authentication ~~Codes durch~~ -Code als "vau-np" eintragen und so an den Client, ~~MUSS sie ein Nutzerpseudonym dem Client in der HTTP-Variable "VAU-NP" verfügbar machen übertragen~~, vgl. A_24770-*.

[<=]

A_25143 - VAU-Protokoll: Abfrage aktueller Status der Nutzerauthentisierung

Eine VAU-Instanz MUSS über innere HTTP-Requests über den Pfadnamen /VAU-Status per HTTP-GET wie folgend aufgeführt Informationen über den Status der aktuellen VAU-Verbindung bereitstellen. Als Antwort MUSS eine VAU-Instanz eine JSON-Datenstruktur der folgenden Struktur senden (Reponse-Content-Type 'application/json'):

```
{  "VAU-Type": "epa",
  "VAU-Version" : "Version bspw. in der Form <Hersteller>-1.2.3.4p8",
  "User-Authentication" : "None | TID:<Telematik-ID> | KVNR:<KVNR> |
  "KeyID": " ... in Hexadezimalform [0-9a-f] (kleinbuchstaben)",
  "Connection-Start": "2024-01-16T10:08:42.123Z"
}
```

Bei "User-Authentication" steht entweder "None" falls noch keine erfolgreiche Nutzerauthentisierung stattgefunden hat, die Telematik-ID falls eine erfolgreiche Nutzerauthentisierung eines Nutzers, der eine Telematik-ID besitzt (bspw. LEI), die KVNR eines erfolgreich authentifizierte Versicherten oder (in einer späteren Ausbaustufe) die Gesundheits-ID des erfolgreich authentifizierte Nutzers.

4406 In der Datenstruktur KÖNNEN weitere Daten (Key-Value-Paare) von der VAU-Instanz
4407 aufgeführt werden. Die maximale Größe der von der VAU-Instanz gesendeten
4408 Datenstruktur MUSS kleiner gleich 2 MiB sein.

4409
4410 Die Abfrage von /VAU-Status MUSS als "Aktivität" im Sinne von
4411 [gemSpec_Aktensystem_ePAfueralle#A_25006-*] gelten. [≤]

4412 Erläuterung: Die Maximalgrößenangabe dient dazu, damit eine Primärsystem-
4413 Implementierung (ePA-Client) weiß welche Datengrößen es maximal akzeptieren können
4414 muss.

4415 **7.4 Authentisierung des E-Rezept-FD als ePA-Client**

4416 Bei der Authentisierung des E-Rezept-FD als ePA-Client wird nicht OIDC zur Client-
4417 Authentisierung verwendet, sondern folgendes einfaches Challenge/Response-Verfahren.

4418 **A_24658-01 - VAU-Protokoll: PKI-basierte Client-Authentisierung**

4419 Eine VAU-Instanz MUSS über einen inneren HTTP-Request zwei Operation wie folgt
4420 anbieten.

4421
4422 (1) Über operationid = getFreshnessParameter MUSS die VAU-Instanz Daten per HTTP-
4423 GET bereitstellen.

4424 Diese Daten sind eine base64-kodierte Zeichenkette. Der Media-Type (Content-Type)
4425 MUSS 'application/json' sein.

4426 Die Zeichenkette ist für einen Client opak. Diese Zeichenkette MUSS es einer VAU-
4427 Instanz ermöglichen, zu ermitteln, ob und wann ein VAU-HSM oder die
4428 Befugnisverifikations-VAU (BV-VAU [gemSpec_Aktensystem_ePAfueralle#3.4
4429 Befugnisverifikations-Modul], das/die mit der VAU-Instanz verbunden ist, diese
4430 Zeichenkette erzeugt hat (bspw. Unix-Zeit + ECDSA-Signatur HSM). Es MUSS
4431 sichergestellt sein, dass nur das VAU-HSM oder die BV-VAU, das/die mit der VAU-Instanz
4432 verbunden ist, den Frischeparameter erzeugt haben kann.

4433
4434 (2) Über operationid = sendAuthorizationRequestBearerToken MUSS sie die Möglichkeit
4435 anbieten, per HTTP-POST ein Authentisierungstoken an die VAU-Instanz zu senden.

4436
4437 Die Authentisierungstoken sind JWT [RFC-7519], deren Body mindestens folgende
4438 Elemente enthalten MUSS:

4439 {
4440 "type" : "ePA-Authentisierung über PKI",
4441 "iat" : ...zeit...,
4442 "challenge" : Frischeparameter,
4443 "sub": ...Telematik-ID...
4444 }

4445 Im Header des JWT MUSS innerhalb eines x5c-Array das "signierende" Zertifikat
4446 enthalten sein. Bei "sub" trägt der Client/Nutzer seine Telematik-ID ein.

4447
4448 Die VAU-Instanz MUSS das vom Client gesendete Authentisierungstoken prüfen:

4449 1. Ist das im Header aufgeführte "signierende" Zertifikat gültig, inkl. OCSP-Prüfung
4450 (vgl. OCSP-Response-Caching nach [gemSpec_PKI#A_23225]).

4451 2. Ist die Signatur valide.

- 4452 3. Stimmt der "sub"-Wert im Body mit der Telematik-ID im "signierenden" Zertifikat
4453 überein.
- 4454 4. Ist die "iat"-Zeit nicht älter als 10 Minuten.
- 4455 5. Ist die "challenge" vom VAU-HSM oder einer BV-VAU erzeugt worden.
- 4456 6. Ist die challenge nicht älter als 10 Minuten.
- 4457 7. Ist der "type" gleich "ePA-Authentisierung über PKI"

4458 Bei Prüfung mit positivem Prüfergebnis MUSS die VAU-Instanz den neuen
4459 Authentisierungsstatus (bspw. E-Rezept-FD wurde authentifiziert) in den Metadaten den
4460 Verbindungsschlüssel (K2_c2s_app_data, K2_s2c_app_data) vermerken.

4461 Im inneren HTTP-Response-Header MUSS die VAU-Instanz das Nutzerpseudonym (vgl.
4462 A_24770-*) übertragen.

4463 Die VAU MUSS die Authentisierung beliebiger Nutzer mit gültigen AUT-Zertifikat und
4464 Telematik-ID darin erlauben (also nicht nur eingeschränkt auf den E-Rezept-FD).[<=]

4467 Beim Frischeparameter soll sichergestellt werden, dass der Authentisierungsvorgang des
4468 Clients nicht zu alt ist. Dies muss auch das VAU-HSM bzw. die BV-VAU prüfen. Das
4469 Signaturmaterial, das die Challenge/Frischeparameter für die Prüfung im Aktensystem
4470 authentisiert, muss keinen Zusammenhang zu TI-PKI haben, da die Challenge vom Client
4471 als opakes Objekt behandelt wird (A_24771-*). D. h., nur das Aktensystem selbst (VAU,
4472 VAU-HSM, BV-VAU) selbst prüft die Challenge.

4473 **A_24959 - VAU-Protokoll: PKI-basierte Client-Authentisierung (VAU-HSM, BV- 4474 VAU)**

4475 Ein Aktensystem MUSS sicherstellen, dass bei der Prüfung der PKI-basierten
4476 Authentisierung (bspw. durch den E-Rezept-FD) die Prüfung des vom Client signierten
4477 JWT mindestens die Prüfschritte aus A_24658-* bei der Prüfung im VAU-HSM oder in der
4478 BV-VAU durchgeführt werden. Bei nicht-positiven Prüfergebnis MUSS das VAU-HSM oder
4479 die BV-VAU das JWT ablehnen -- die Client-Authentisierung beim VAU-HSM bzw. BV-HSM
4480 kann nicht stattfinden.[<=]

4481 **~~AA_25192-0102~~ - VAU-Protokoll: VAU-Instanz, zusätzliche TID-Prüfung E- 4482 Rezept-FD**

4483 Ein Aktensystem MUSS sicherstellen, dass in einer VAU-Instanz bei jeder Nutzer-
4484 Authentisierung (i. S. v. egal ob über A_24568-* oder A_25055-*) eines ePA-Nutzers,
4485 bei der der Nutzer die Telematik-ID "9-E-Rezept-Fachdienst" besitzt, die VAU-Instanz
4486 prüft, ob das "authentisierende" EE-AUT-Zertifikat des Nutzers

4487 1. ~~professionItem~~ und professionOID gleich 1.2.276.0.76.4.258
4488 gemäß [gemSpec_OID#GS-A_4446-*]-*, oid_erp-vau ~~und 1.2.276.0.76.4.258-]~~
4489 als ~~Attribute~~Attribut besitzt, und

4490 2. aus einer Komponenten-PKI-CA der TI-PKI stammt (vgl.
4491 Implementierungshinweis).

4492 Liefern diese Prüfungen kein positives Prüfergebnis, so MUSS die VAU-Instanz die
4493 Authentifizierung ablehnen.~~[<=]~~. [~~<=~~]

4494 Eine E-Rezept-VAU als ePA-Nutzer hat innerhalb eines Aktensystems besonders
4495 umfassende Zugriffsrechte. Als Risikomitigation wird bei einer Nutzerauthentisierung in
4496 einer ePA-VAU-Instanz überprüft, ob das AUT-Zertifikat der E-Rezept-VAU (als ePA-
4497 Nutzer) besondere Eigenschaft besitzt.

4498 Implementierungshinweis:

4499 Sowohl bei A_24568-* als auch bei A_25055-* wird eine auf dem EE-AUT-Zertifikat
4500 basierende Signatur vom Nutzer erzeugt. Solch eine Signatur muss eine ePA-VAU-
4501 Instanz im Rahmen der Nutzerauthentifizierung prüfen, was auch die Prüfung des EE-
4502 AUT-Zertifikats inkludiert. Die TI-PKI besitzt genau drei Hierarchie-Ebenen: Root-
4503 Zertifikat, CA-Zertifikat, EE-Zertifikat. Es ist für die Prüfung A_25192-*#Punkt-2
4504 ausreichend zu prüfen, ob das bestätigende CA-Zertifikat im CommonName mit
4505 "GEM.KOMP-CA" (anschließend kommt eine natürliche Zahl) beginnt und ob das CA-
4506 Zertifikat im Signatur-Graph Kind einer TI-PKI-Root ist. Beide Bedingungen sind
4507 technisch leicht zu prüfen, dennoch empfiehlt es sich innerhalb einer VAU-Instanz das
4508 Prüfergebnis zu cachen (Hashwert des AUT-Zertifikats + besondere-E-Rezept-Prüfung-
4509 OK).

4510 **A_24771 - VAU-Protokoll: E-Rezept als Client**

4511 Der E-Rezept-FD MUSS für die Authentisierung bei einer VAU-Instanz nach dem
4512 erfolgreichen Durchlaufen der VAU-Protokoll-Handshake-Phase den Mechanismus nach
4513 A_24658-* verwenden.

4514
4515 Dabei bezieht er zunächst eine Challenge/Frischeparameter nach A_24658-*. Diese
4516 MUSS er als opakes Objekt behandeln (also nicht versuchen, den Inhalt auszuwerten).

4517
4518 Anschließend MUSS er einen signierten JWT nach A_24658-* erzeugen und diese per
4519 HTTP-POST im inneren Request eine VAU-Protokoll-Verbindung an die VAU-Instanz
4520 senden.[<=]

4521 Der E-Rezept-FD kann für die Datenübertragung zum Aktensystem folgendes Vorgehen
4522 wählen. Er verbindet sich initial einmal per VAU-Protokoll mit einer VAU-Instanz und
4523 verwendet den Mechanismus nach A_24658-* zur Authentisierung des E-Rezept-FD bei
4524 der VAU-Instanz. Hat dies erfolgreich stattgefunden, erhält der E-Rezept-FD als
4525 "normaler" ePA-Client ein Nutzerpseudonym (vgl. A_24770-*). Nun kann er mit diesem
4526 Nutzerpseudonym (A_24757-*) bspw. 50 neue TLS-Verbindungen zum Aktensystem
4527 eröffnen. Bezüglich des DoS-Schutzes am Aktensystem: an der IP-Adresse kann ein
4528 Aktensystem schon erahnen, dass es sich um den E-Rezept-FD handelt und so diese 50
4529 Verbindungen auf IP-Ebene dem Client erlaubt (Perimeter/Firewall), was es bei anderen
4530 IP-Adressen evtl. nicht tut (Rate-Limiting). Innerhalb der neuen TLS-Verbindungen führt
4531 der E-Rezept-FD jeweils VAU-Protokoll-Verbindungsaufbauten plus Nutzerauthentisierung
4532 (A_24770-*) durch und ein Aktensystem kann "geeignet" (vgl. Abschnitt 7.5- Routing auf
4533 VAU-Instanzen) die Verbindungsaufbauten auf für das Aktensystem günstige VAU-
4534 Instanzen verteilen. Im Beispiel kann der E-Rezept-FD dann über 51 Datenkanäle parallel
4535 Rezeptinformationen für verschiedene Versicherte in deren Akten einbringen
4536 (Parallelverarbeitung).

4537 **7.5 Routing auf VAU-Instanzen**

4538 Für ein Aktensystem erleichtert es in bestimmten Konstellationen die Implementierung,
4539 wenn schon beim VAU-Protokoll-Verbindungsaufbau es Hinweise darüber gibt, um welche
4540 Client-Identität (i. S. v. Nutzer-Identität) es sich handelt. Dann kann eine Routing-
4541 Komponente des Aktensystems, die nach den HTTPS-Schnittstellen und vor den VAU-
4542 Instanzen liegt, die Requests "geeignet" auf die verschiedenen VAU-Instanzen verteilen.
4543 Was "geeignet" hier bedeutet, liegt in der Ausgestaltungshoheit des
4544 Aktensystemherstellers.

AA_24770-01 - VAU-Protokoll: VAU-Instanz, Nutzerpseudonym erzeugen

Nach erfolgreicher Nutzer-Authentisierung des Clients MUSS die VAU-Instanz ein Nutzerpseudonym (NP) erzeugen. Dieses NP MUSS, ~~im HTTP- in der~~ Response-Header des inneren HTTP-~~Request unter dem Variablennamen~~Requests als "vau-np" eingetragen werden und so an den Client übertragen werden. ~~[<=]~~. [~~<=~~]

Beispiel

VAU-NP: 1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

In A_24757-* ist definiert, dass ein VAU-Client solch ein Nutzerpseudonym bei seinen Requests für die Nachricht 1 im Handshake (VAU-Protokoll Verbindungsaufbau) im Request-Header im (äußeren) HTTP-Request aufführen muss und dass er auch eine Aktualisierung des Nutzerpseudonym nach erfolgreicher Nutzer-Authentisierung erkennen und sein lokal gespeichertes Nutzerpseudonym aktualisieren muss.

A_25150 - VAU-Protokoll: VAU-Instanz, Nutzerpseudonym regelmäßig wechseln

Ein Aktensystem MUSS sicherstellen, dass ein Nutzerpseudonym nach maximal 2 Monaten gewechselt wird, i. S. v. nach erfolgreicher Authentisierung von Entität A wird NP_1 an den Client zurückgegeben (A_24770-*) nach etwas mehr als zwei Monaten authentisiert sich A erneuert und erhält NP_2, dann MUSS NP_1 ungleich NP_2 sein. Die bloße Kenntnis von NP_1 und NP_2 DARF NICHT erkennen lassen, dass diese beide Pseudonyme zur gleichen Entität gehören oder gar zu A gehören. (vgl. Implementierungshinweis). [~~<=~~]

Implementierungshinweis zu A_25150:

Eine mögliche Herangehensweise für A_25150-* ist, dass alle VAU-Instanzen Zugriff auf einen geheimen Pseudonymisierungsschlüssel für die Pseudonymerstellung haben. Dieser Schlüssel liegt nicht notwendiger Weise ausschließlich im HSM, sondern kann beim Start der VAU-Instanz in diese sicher geladen werden und im sicheren Speicher der VAU-Instanz verbleiben. Der Schlüssel wird alle zwei Monate Aktensystemweit gewechselt. Nach erfolgreicher Authentisierung wird mit dem Entitäts-ID (bspw. KVNDR) ein HMAC(K=<Schlüssel>, text=<KVNDR>) erzeugt. Dieser berechnete HMAC-Wert ist dann das NP.

Routingentscheidung

Es gibt wenige Fälle, in denen aufgrund von fehlenden Informationen eine Routing-Entscheidung im Aktensystem nicht optimal getroffen werden konnte. In diesen Fällen darf ein Aktensystem das Neuverbinden des Client einfordern. Ein solcher Fall ist bspw., wenn eine Arztpraxis auf einem PVS-Rechner eine Akte aktuell in Verwendung hat. Und auf einem "neuen" PVS-Rechner/Tablet etc., der noch nie mit dem Aktensystem in Kontakt stand -- also lokal noch kein Nutzerpseudonym gespeichert hat (vgl. A_24757-*), eine Akte verwenden möchte. Auf dem neuen PVS-Rechner würde dann ein VAU-Protokoll-Verbindungsaufbau und Nutzer-Authentisierung stattfinden. Mit hoher Wahrscheinlichkeit landet diese Verbindung in einer anderen VAU-Instanz. Dies würde dann erkannt und A_24772-* kann vom Aktensystem ausgelöst werden. Der "neue" PVS-Rechner würde sich neu verbinden, diesmal mit ihm bekanntem Nutzerpseudonym. Das Aktensystem kann dann eine für sich günstigere Routing-Entscheidung treffen. Das Aktensystem ist frei in der Entscheidung, ob und bei welchen Konstellationen es von dieser Möglichkeit Gebrauch macht.

A_24772 - VAU-Protokoll: Restart für Änderung der Routingentscheidung

Ein ePA-Aktensystem KANN nach erfolgreicher Authentisierung des ePA-Clients (i. S. v. Nutzer-Authentisierung) folgende Retry-Nachricht senden:


```
4593 {  
4594     "MessageType" : "Restart",  
4595     "KeyID" : ... KeyID ...  
4596 }
```

4597
4598 Genauer: die Nutzer-Authentisierung hat erfolgreich stattgefunden, und bei dem
4599 nächsten Request des Clients KANN ein Aktensystem die Restart-Nachricht als Antwort
4600 (Response) auf den Request des Clients senden.
4601

4602 Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert und die erzeugte
4603 Kodierung unter Verwendung des Media-Type 'application/cbor' (HTTP Content-Type) an
4604 den ePA-Client im äußeren HTTP zurückgesendet werden. [\leq]

4605 **A_24773 - VAU-Protokoll: Clients: Neustart/Wiederholung des** 4606 **Verbindungsaufbaus**

4607 Ein VAU-Client MUSS Restart-Nachrichten nach A_24772-* verarbeiten können, und bei
4608 Erhalt einer solchen Nachricht die aktuelle VAU-Protokoll-Verbindung, die mit der
4609 aufgeführten KeyID verbunden ist, beenden, indem es die mit der KeyID verbundenen
4610 symmetrischen Schlüssel sicher löscht. Anschließend MUSS es einen neuen VAU-Protoll-
4611 Verbindungsaufbau durchlaufen (inkl. Nutzer-Authentisierung gegenüber der VAU-
4612 Instanz). [\leq]

4613 **7.6 Fehlersignalisierung**

4614 **A_24635 - VAU-Protokoll: VAU-Instanz und Aktensystem: Erzeugung von** 4615 **Fehlermeldungen**

4616 Eine VAU-Instanz MUSS das Auftreten von Fehlern bei der Abarbeitung des VAU-
4617 Protokolls an das "äußere" Aktensystem melden. Daraufhin MUSS das Aktensystem an
4618 der HTTPS-Schnittstelle den Client-Request in folgender Weise mit einer Fehlermeldung
4619 beantworten:
4620

4621 Als HTTP-Response-Code MUSS das Aktensystem einen HTTP-Fehlercode (i. S. v. eben
4622 nicht 200) in der äußeren HTTP-Response verwenden (siehe folgende Tabelle). Es MUSS
4623 weiterhin eine Datenstruktur der folgenden Art erzeugen:

```
4624 {  
4625     "MessageType" : "Error",  
4626     "ErrorCode" : ...natürliche-Zahl...,  
4627     "ErrorMessage" : "... Menschenlesbarer Text bzw. Fehlerursache  
4628     ..."  
4629 }
```

4630 Diese Datenstruktur MUSS es per CBOR [RFC-CBOR] serialisieren und per Mime-Type
4631 "application/cbor" [RFC-CBOR] (HTTP Content-Type) im äußeren Response-Body
4632 aufführen.
4633

4634 Ein Aktensystem KANN weitere selbst definierte Variablen-Werte-Paare in der o. g.
4635 Datenstruktur aufführen.

4636

4637 Folgende Fehler-Meldungen MUSS es mindestens geben:

HTTP-Error	ErrorCode	ErrorMessage / Beschreibung
400	1	"Decoding Error" Fehler in einer Kodierung bspw. der CBOR-Kodierung
400	2	"Missing Parameters" notwendige Datenfelder bspw. in der Handshake fehlen
403	3	"GCM returns FAIL" Eine AES/GCM-Entschlüsselung ergibt das Symbol "FAIL".
403	4	"PU/nonPU Failure"
403	5	"Transscript Error" Im Handshake wird Ungleichheit zwischen den beiden Transskript-Hashwerten festgestellt.
400	6	"bad format: extended ciphertext" Die erste Sanity-Prüfung des erweiterten Chifftrat bspw. bei A_24630 schlägt fehl.
403	7	"is not a request" A_24630 (Prüfschritt 3)
403	8	"unknow KeyID"
403	9	"unknown CID"

4638 [**<=**]

4639 **A_24767 - VAU-Protokoll, Fehlerverarbeitung im VAU-Client**

4640 Ein VAU-Client MUSS die Fehlermeldungen nach A_24635-* verarbeiten können.**[<=]**

4641 **7.7 Tracing in Nichtproduktivumgebungen**

4642 Für die Fehlersuche in Nichtproduktivumgebungen -- insbesondere bei IOP-Problemen
4643 zwischen Produkten verschiedener Hersteller in einer fortgeschrittenen
4644 Entwicklungsphase -- hat es sich als notwendig erwiesen, dass ein Fehlersuchender den

Klartext der Kommunikation zwischen ePA-Client und VAU-Instanz mitlesen kann, vgl. [gemSpec_Aktensystem_ePAfueralle#3.18.4 Tracing in Nichtproduktivumgebungen].

Bei ePA für alle unterscheidet sich der Mechanismus hier im Vergleich zu ePA 2.x ein wenig: Es werden nicht für den Client feste ECDH-Schlüssel für die Schlüsselaushandlung im VAU-Protokoll in Nichtproduktivumgebungen vorgegeben, sondern der Client wird verpflichtet, die ausgehandelten symmetrischen Sitzungsschlüssel (K2_c2s_app_data, K2_s2c_app_data) im äußeren HTTP-Request im Request-Header aufzuführen. Die Motivation dafür ist, dass es bei einigen Krypto-Bibliotheken ggf. schwieriger sein kann, den Zufall, der bei der KEM-Encapsulation-Operation einfließt, fest vorzugeben. Es erleichtert also die Implementierbarkeit. Am Chifftrat kann eine VAU-Instanz sicher feststellen, für welche Umgebung ein Client ein Chifftrat erzeugt hat. Ein VAU-Instanz lehnt in der Produktivumgebung dann solche Nichtproduktivumgebungs-Chifftrate ab.

A_24477 - VAU-Client, Nichtproduktivumgebung, Offenlegung von symmetrischen Verbindungsschlüsseln

Ein VAU-Client in einer Nichtproduktivumgebung MUSS nach einer erfolgreichen Schlüsselaushandlung mit dem VAU-Protokoll die ausgehandelten symmetrischen Schlüssel K2_c2s_app_data und K2_s2c_app_data base64-kodiert innerhalb einer HTTP-Request-Header-Variable "VAU-nonPU-Tracing" bei jedem äußeren HTTP-Request aufführen. Dabei sind die beiden Base64-kodierten Schlüsselwerte durch Leerzeichen zu trennen. Beispiel:

```
VAU-nonPU-Tracing: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=
```

Als erstes ist der Wert von Schlüssel K2_c2s_app_data aufzuführen, als zweiter Wert ist der Wert von Schlüssel K2_s2c_app_data aufzuführen. [≤]

Hinweis: Ebenfalls muss ein VAU-Client nach A_24628-* im Chifftrat kennzeichnen, ob es eine Nachricht in einer Nichtproduktivumgebung ist oder nicht.

A_24478 - VAU-Instanz, Nichtproduktivumgebung, Ablehnung von nonPU-Chiffraten in der PU

Ein Aktensystem in der Produktivumgebung MUSS äußere HTTPS-Request, die im Request-Header eine Variable "VAU-nonPU-Tracing" enthalten, mit einer HTTP-Fehlermeldung 403-Forbidden beantworten. Das Aktensystem MUSS sicherstellen, dass der innere Request (Chifftrat im Request-Body des äußeren Requests) keine VAU-Instanz erreicht.

Eine VAU-Instanz in der Produktivumgebung, die am Chifftrat am nonPU/PU-Byte erkennt (vgl. A_24628-*), dass es sich um ein nonPU-Chifftrat handelt, MUSS den Request mit einer Fehlermeldung an die äußere Schicht des Aktensystems (Webschnittstelle) beantworten. Die VAU-Instanz MUSS sicherstellen, dass das Chifftrat nicht entschlüsselt und das Chifftrat in der VAU-Instanz sicher gelöscht wird. [≤]

7.7.1 Zufallsquelle für Clients

A_24921 - ePA-Aktensystem, Random-Operation

Ein Aktensystem MUSS an seiner äußeren HTTPS-Schnittstellen eine Random-Operation, wie in A_21215-* definiert, zur Verfügung stellen. [≤]

Erläuterung: Auf einem Linux-Server ist es ausreichend, Daten aus /dev/urandom zu verwenden.

8 ZETA/ASL (VAU-Protokoll)

Bei allen Anwendungen der TI, die personenbezogenen medizinische Daten verarbeiten, gibt es das Grundprinzip der Mehrschichtigen Sicherheit (bspw. E-Rezept, ePA, KIM). Es gibt eine Sicherungsschicht (meistens TLS, vgl. Abschnitt 3.3.2 TLS-Verbindungen) und oberhalb dieser Sicherungsschicht gibt es eine weitere Sicherungsschicht, die eine Unabhängigkeit von möglichen Schwachstellen auf der unteren Schicht erzeugt.

Bei TLS gab es in der Vergangenheit:

1. Schwachstellen im Protokoll selbst (BEAST (Browser Exploit Against SSL/TLS), CRIME (Compression Ratio Info-leak Made Easy), DROWN (Decrypting RSA with Obsolete and Weakened eNcryption), POODLE (Padding Oracle On Downgraded Legacy Encryption), ROBOT (Return Of Bleichenbacher's Oracle Threat), Ticketbleed etc.)
2. schwerwiegende Implementierungsfehler (Heartbleed) in oft verwendeten TLS-Implementierungen
3. erfolgreiche Angriffe im Internet-CA-Vertrauensraum (DigiNotar) (Hinweis: die Grundlage für die TLS-Verwendung bei Zero-Trust ist der Internet-CA-Vertrauensraum).

Unter anderen der Einsatz der hier definierten zweiten Sicherungsschicht (Additional Security Layer (ASL)) hilft derartige auch zukünftig zu erwartenden Arten von Schwachstellen aus (1) bis (3) abzufangen. Dies ermöglicht die unmittelbare Weiterführung des Betriebs (Betriebssicherheit) der TI-Anwendung auch bei Bekanntwerden von Schwachstellen auf einer Sicherungsschicht - man kann die Schwachstellen schließen, ohne zwischenzeitlich den Betrieb einstellen zu müssen.

Das in diesem Abschnitt definierte kryptographische Protokoll ist bis auf eine Aktualisierung des VAU-Protokoll von ePA für alle. Aktualisierung: nach Abschluss der Standardisierung von FIPS 203 [FIPS-203] wird für das zu verwendete PQC-KEM-Verfahren nicht mehr auf auf Kyber Version 3.02 [IETF-Kyber] verwiesen sondern eben auf den nun finalisierten FIPS 203 (ML-KEM-768).

Die Verwendung des VAU-NP (vgl. A 24757-*) macht im Kontext Zero-Trust keinen Sinn mehr und wird deshalb bei ZETA/ASL nicht mehr verwendet.

A 26920 - ZETA/ASL mindestens HTTP-Version 1.1

Ein ZETA/ASL-Server MUSS an seinen HTTPS-Schnittstellen, i. S. v. Schnittstellen, die ein ZETA/ASL-Client anspricht, mindestens HTTP Version 1.1 unterstützen. [<=]

8.1 Verbindungsaufbau/Schlüsselaushandlung

A 26921 - ZETA/ASL: ZETA/ASL-Schlüssel für die ASL-Protokoll-Schlüsselaushandlung

Ein ZETA/ASL-Server MUSS sicherstellen, dass

1. es eine Signatur-Identität (AUT) aus der Komponenten-PKI der TI gibt, die technisch sichergestellt ausschließlich nur von ZETA/ASL-Instanzen verwendbar ist.

2. es semi-statische Schlüsselpaare für ECDH (auf Basis Kurve P-256 [FIPS-186-5]) und ML-KEM-768 [FIPS-203] gibt, deren private Schlüssel, technisch sichergestellt, ausschließlich von ZETA/ASL-Server verwendbar sind.
3. die privaten Schlüssel (aus 2.) in einer ZETA/ASL-Instanz erzeugt und verarbeitet werden,
4. die semi-statischen Schlüssel eine maximale Lebensdauer von einem Monat besitzen (Hinweis: die Forward-Secrecy hängt nicht vom Wechselintervall ab, innerhalb eines Verbindungsaufbaus und der Schlüsselaushandlung dabei fließen ephemere Schlüsselwerte von Client und Server ein).
5. die semi-statischen Schlüssel in einer über die Signatur-Identität authentisierten folgenden Datenstruktur aufgeführt werden.

Struktur der signierten semi-statischen öffentlichen ASL-Schlüssel

```
ASL Keys = {  
  "ECDH PK" :  
    { "crv" : "P-256",  
      "x" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),  
      "y" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),  
    },  
  "ML-KEM-768 PK" : Binärwert-öffentlicher-Schlüssel-nach-keygen-  
FIPS-203-ML-KEM-768,  
  "iat" : Erzeugungszeits-Sekunden-Since-Epoch (integer),  
  "exp" : Nicht-mehr-Verwendbar-nach (integer),  
  "comment" : "Erzeugt bei ASL-Instanz xyz, Meta-Info abcd"  
}
```

In "comment" KÖNNEN beliebige Text-Daten aufgeführt werden. Es können weitere Attribute hinzugeführt werden. Ein Client MUSS ihm unbekannte Attribute ignorieren. Diese Struktur wird mittels CBOR [RFC-CBOR] binär kodiert und im Folgenden ASL Keys encoded genannt.

Diese binäre Byte-Folge wird in folgende Datenstruktur eingebracht

```
{  
  "signed pub keys" : ASL Keys encoded,  
  "signature-ES256" : ECDSA-Signatur-SHA-256-analog-RFC-7515 (R||S => 64  
Byte) binär,  
  "cert hash" : SHA-256-Wert des "signierenden" AUT-ASL-Zertifikats,  
  "cdv" : Cert-Data-Version (natürliche Zahl, beginnend mit 1,  
vgl. A 26922-*),  
  "ocsp response" : OCSP-Response-für-das-ASL-Signaturzertifikat-nicht-  
älter-als-24-Stunden-DER-Kodierung  
}
```

Diese Datenstruktur wird mittels CBOR binär kodiert (serialisiert). Das Ergebnis der Kodierung wird "signierte öffentliche ZETA/ASL-Schlüssel" (Plural) genannt.

[<=]

A 26922 - ZETA/ASL: Verfügbarmachung des AUT-ZETA/ASL-Zertifikats plus Prüfkette

Ein ZETA/ASL-Server MUSS über an seinen Webschnittstellen mittels eines äußeren HTTPS-Requests per HTTP-GET unter dem Pfadnamen /CertData.<SHA-256-Hashwert-Hex-[0-9a-f]>-Versionszahl (a-f kleingeschrieben) folgende Datenstruktur zur Verfügung stellen:

```
{  
  "cert": DER-kodiertes-AUT-ZETA/ASL-Zertifikat,  
  "ca" : DER-kodiertes-Komponenten-PKI-CA-aus-dem-"cert"-kommt,  
  "rca chain" : [Cross-Zertifikat-1, ..., Cross-Zertifikat-n],  
}
```

Die Versionszahl MUSS eine natürliche Zahl sein, beginnend mit 1, die es trotz A 26923-* erlaubt, Fehler in den Daten zu korrigieren (siehe Erläuterung nach A 26922-*).

Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert/kodiert werden und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) an den ZETA/ASL-Client als Response auf den GET-Request gesendet werden.

In "rca chain" MÜSSEN alle Cross-Zertifikate in chronologischer Ordnung von RCA7 ausgehend aufgeführt werden, bis die Root-Schlüssel (Cross-Zertifikat) erreicht werden, mit denen das "ca"-Zertifikat bestätigt (signiert) wurde; d. h., sozusagen eine einfach verkettete Liste von Cross-Zertifikaten chronologisch aufsteigend. [<=]

A 26923 - ZETA/ASL: ZETA/ASL-Client Prüfbasis Zertifikatsprüfung

Ein ZETA/ASL-Client MUSS als Prüfbasis eine Root-Version (X.509-Root-TI-Zertifikat), die mindestens 2 Jahre alt ist, verwenden oder die TSL. Der ZETA/ASL-Client MUSS die Operation nach A 26922-* verwenden, um die für die Zertifikatsprüfung von A 26921-* notwendigen Zertifikate zu beziehen. Die bezogenen Zertifikatsdaten MUSS der ZETA/ASL-Client lokal (zeitlich unbegrenzt) vorhalten (Caching). Bei Erhalt einer Nachricht 2 (A 26935-*) MUSS er zunächst prüfen, ob er die für die Zertifikatsprüfung notwendigen Zertifikate im lokalen Cache vorrätig hat, und falls ja MUSS er diese verwenden. [<=]

A 26932 - ZETA/ASL: ZETA/ASL-Client, Nachricht 1

Ein ZETA/ASL-Client MUSS für die Erzeugung folgende Schritte durchlaufen.
Ein ZETA/ASL-Client MUSS

1. ein ECC-Schlüsselpaar auf der Kurve P-256 [FIPS-186-5] erzeugen.
2. ein ML-KEM-768-Schlüsselpaar [FIPS-203] erzeugen.

Anschließend MUSS er die öffentlichen Schlüssel in folgende Datenstruktur überführen

```
{  
  "MessageType" : "M1",  
  "ECDH PK" : { "crv" : "P-256",  
                "x" : analog A 26921-*,  
                "y" : analog A 26921-* },  
  "ML-KEM-768 PK" : analog zu A 26921-*,  
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] in eine Bytefolge kodieren (serialisieren).

Diese Bytefolge ist die Nachricht 1.

Diese Nachricht 1 MUSS der ZETA/ASL-Client an die HTTPS-Schnittstelle des Fachdienstes per POST auf den Pfad /ASL senden, wobei er den Mime-Type

"application/cbor" [RFC-CBOR] (HTTP Content-Type) verwendet und die Nachricht 1 im Request-Body aufführt.

[<=]

A 26933 - ZETA/ASL-Protokoll: ZETA/ASL-Server: Erhalt von Nachricht 1

Eine ZETA/ASL-Server MUSS die für Nachricht 1 erzeugten Datenobjekte nach A 26932-* verarbeiten können.[<=]

A 26934 - ZETA/ASL-Protokoll: AES/GCM-Verschlüsselung im Handshake

Ein ZETA/ASL-Server und -Client verschlüsseln im Rahmen des Handshakes des ZETA/ASL-Protokolls verschiedene Nachrichten-Teile mittels AES/GCM. Dabei MÜSSEN sie pro Verschlüsselung den IV jeweils zufällig als 96-Bit-Wert erzeugen. Der Authentication-Tag MUSS 128 Bit lang sein. Das Ergebnis der Verschlüsselung MUSS dann in der folgenden Kodierung aufgeführt werden:

IV || eigentliche AES-GCM-Chiffre || 128-Bit langer Authentication-Tag.hb[<=]

A 26935 - ZETA/ASL-Protokoll: ZETA/ASL-Server: Nachricht 2

Ein ZETA/ASL-Server MUSS die beiden öffentlichen Schlüssel aus Nachricht 1 (vgl. A 26932-*) prüfen (korrekte ECC-Kurve ("crv":"P-256"), öffentlicher Punkt liegt auf der Kurve P-256 [FIPS-186-5], der öffentliche ML-KEM-768-Schlüssel ist valide [FIPS-203]). Er MUSS für ECDH und ML-KEM-768 jeweils die KEM-Encapsulate-Funktion verwenden und erhält dabei zwei Geheimnisse (ss e ecdh, ss e mlkem768) und zwei Ciphertexte (ECDH ct, ML-KEM-768 ct). Er MUSS für die beiden Geheimnisse zusammenfügen: ss e = ss e ecdh || ss e mlkem768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = " (leere Zeichenkette)), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1 c2s und die letzten 32 Byte heißen K1 s2c.

Mit dem Schlüssel K1 s2c mittels AES/CGM (vgl. A 26934-*) MÜSSEN die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A 26921-*) verschlüsselt werden. Das Ergebnis (vgl. A 26934-*) heißt aed ciphertext msg 2.

Er MUSS folgende Datenstruktur erzeugen:

```
{  
  "MessageType" : "M2",  
  "ECDH ct" : ... analog ECDH PK aus A 26932-* ...,  
  "ML-KEM-768 ct" : ML-KEM-768-Ciphertext analog [FIPS-203],  
  "AEAD ct" : aead ciphertext msg 2  
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 2.

Er MUSS eine ID erzeugen, kodiert aus der Zeichenmenge

A-Za-z0-9-/

die maximal 200 Zeichen lang ist. Die ID MUSS mit "/" (Slash) beginnen und MUSS ein gültiger URL-Pfadname sein. Diese ID MUSS es ZETA/ASL-Server ermöglichen, den aktuellen Handshake entsprechend zuzuordnen bei Eintreffen der Nachricht-3 des ZETA/ASL-Clients, der diese ID mitsendet. Der ZETA/ASL-Server kann die Struktur der ID selbst definieren. Ein ZETA/ASL-Client MUSS die ID als opake Zeichenkette behandeln. Er MUSS die Nachricht 2 inkl. ID an die HTTPS-Schnittstelle des Fachdienstes übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 1 eingetroffen ist, die Nachricht 2 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP

Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "ZETA-ASL-CID" die ID aufgeführt werden. [\leq]

Hinweis: Bei der KEM-Encapsulate-Funktion fließt Zufall aus dem System ein. Für den ZETA/ASL-Kommunikationspartner gelten die Vorgaben aus Abschnitt 2.4 (Güte der Zufallserzeugung, Zuweisung über den Produkttypsteckbrief).

Beispiele für die ID aus A 26935-*:

- /ZT-
ASL/9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08/
1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
- /ASL/1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

A 26936 - ZETA/ASL-Protokoll: ZETA/ASL-Client: Erhalt von Nachricht 2

Ein ZETA/ASL-Client MUSS die für Nachricht 2 erzeugten Datenobjekte nach A 26935-* verarbeiten können.

Er MUSS prüfen, ob

1. im HTTP-Response-Header die Variable "ZETA-ASL-CID" enthalten ist, falls nicht Abbruch.
2. der Wert der Variable eine Bytefolge ist, dessen Länge maximal 200 Byte lang ist und die nur die Zeichen
A-Za-z0-9-/
enthält und mit "/" (Slash) beginnt. Falls nicht Abbruch.

Er MUSS den Wert als Pfad für das Versenden der Nachricht 3 (vgl. A 26937) und aller weiteren Nachrichten - also auch nach dem erfolgreichen Handshake - im Kontext dieser Verbindung verwenden. [\leq]

A 26937 - ZETA/ASL-Protokoll: ZETA/ASL-Client: Nachricht 3

Ein ZETA/ASL-Client MUSS aus der Nachricht 2 (vgl. A 26936-*) des ZETA/ASL-Servers mittels der Ciphertexte ECDH ct und ML-KEM-768 ct und den privaten ephemeren Client-Schlüsseln aus A 26932-* (Nachricht 1) und der jeweiligen KEM-Decapsulation-Funktionen zwei Geheimnisse berechnen: ss_e ecdh und ss_e mlkem768. Er MUSS die beiden Geheimnisse zusammenfügen: $ss_e = ss_e$ ecdh || ss_e mlkem768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = " (leere Zeichenkette), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1 c2s und die letzten 32 Byte heißen K1 s2c.

Mit dem Schlüssel K1 s2c mittels AES/CGM (vgl. A 26934-*) MUSS der Ciphertext "AEAD ct" entschlüsselt werden: die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A 26921-*) werden so als Klartext erhalten. Diese ZETA/ASL-Schlüssel MUSS er nach A 26938-* prüfen. Mittels der öffentlichen Schlüssel (ECDH PK, ML-KEM-768 PK) MUSS er jeweils die KEM-Encapsulation-Funktion ausführen und er erhält zwei Geheimnisse: ss_s ecdh und ss_s mlkem768. Diese führt er zusammen: $ss_s = ss_s$ ecdh || ss_s mlkem768. Weiter berechnet er $ss = ss_e$ || ss_s . Dieses Geheimnis verwendet die HKDF [RFC-5859] auf Basis von SHA-256 (info = " (leere Zeichenkette)), um 160 Byte (=5 * 32 Byte) abzuleiten. Er MUSS diese 160 Byte in 32 Byte-Blöcke (von offset 0 bis zum Ende) auf folgende fünf Variablen (vier Schlüssel + eine KeyID) verteilen:

- K2 c2s key confirmation,
- K2 c2s app data,

• K2 s2c key confirmation,

• K2 s2c app data

und

• KeyID (nicht vertraulich).

Diese ersten vier sind vertrauliche Schlüsselwerte für AES/GCM. Die KeyID wird nach dem Handshake als eindeutige ID für die K2* app data Schlüssel dienen.

Er MUSS eine Datenstruktur wie folgt erzeugen:

```
{  
  "ECDH ct" : client kem result 2["ECDH ct"],  
  "ML-KEM-768 ct" : client kem result 2["ML-KEM-768 ct"],  
  "ERP" : False,  
  "ESO" : False  
}
```

ERP steht für "Enforce Replay Protection" und ESO steht für "Enforce Sequence Order". Innerhalb der Spezifikation heißt diese Datenstruktur Nachricht 3 inner Layer. Diese Datenstruktur MUSS er per CBOR [RFC-CBOR] serialisieren/kodieren. Diese Serialisierung MUSS er mittels K1 c2s verschlüsseln (vgl. A 26934-*) (= "ciphertext msg 3"). Er MUSS die komplette Nachricht-1 (CBOR-Kodierung), die Nachricht-2 und ciphertext msg 3 konkatenieren (= Transskript des Client) und davon den SHA-256-Hashwert berechnen. Diesen Hashwert MUSS er mittels K2 c2s key confirmation verschlüsseln (vgl. A 26934-*), das Chifftrat sei als aead ciphertext msg 3 key confirmation hier bezeichnet.

Dann MUSS er folgende Datenstruktur erzeugen:

```
{  
  "MessageType" : "M3",  
  "AEAD ct" : ciphertext msg 3,  
  "AEAD ct key confirmation" : aead ciphertext msg 3 key confirmation  
}
```

Diese Datenstruktur MUSS er per CBOR serialisieren, das Ergebnis ist Nachricht 3. Er MUSS die Nachricht 3 per äußeren HTTP-Request an das Aktensystem senden und dabei den Wert der ZETA/ASL-CID (vgl. A 26936) als URL-Pfadnamen verwenden, unter Verwendung der HTTP-POST-Methode.

[<=]

A 26938 - ZETA/ASL-Protokoll: ZETA/ASL-Client: Prüfung der "signierten öffentlichen ZETA/ASL-Schlüssel"

Ein ZETA/ASL-Client MUSS die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A 26921-*) bei der Verarbeitung von Nachricht 3 (vgl. A 26937-*) wie folgt prüfen. Erhält er bei einem der folgenden Prüfpunkte kein positives Prüfergebnis, so MUSS er die Verarbeitung (Handshake) abbrechen.

1. Prüfung des TI-Zertifikats, das den Hashwert aus dem "cert hash"-Datenfeld besitzt (Bezug des Zertifikats vgl. A 26922-*), u. a. unter der Verwendung der OCSP-Response aus "ocsp_response" für die Prüfung des Sperrstatus (Prüfung ob "good"). Die OCSP-Response darf dabei nicht älter als 24 Stunden sein.

2. Das ZT-Server/TI-Zertifikat MUSS zeitlich gültig sein. Es MUSS kryptographisch in einer Zertifikats-/Signaturprüfungskette rückführbar auf eine X.509-Root-Version der TI-PKI sein.
3. Das TI-Zertifikat MUSS aus der Komponenten-PKI der TI stammen (vgl. Implementierungshinweis A 25192-*#Punkt-2) und die vom Client gewünschte Rollen-OID (bspw. "oid_eпа_vau") besitzt.
4. Die Signatur im "signature-ES256"-Datenfeld MUSS eine valide Signatur für die Daten im Datenfeld "signed_pub_keys" (Signaturprüfung ergibt "valid/accept") sein, unter Verwendung des öffentlichen Signatur-Schlüssels aus dem ZETA/ASL-TI-Zertifikats bei der Signaturprüfung.
5. Der ECC-Schlüssel in signed_pub_keys (vgl. Erzeugung bei A 26921-*) MUSS ein gültiger Punkt der Kurve P-256 [FIPS-186-5] sein. Der öffentliche Schlüssel in "ML-KEM-768_PK"-Datenfeld MUSS ein gültiger ML-KEM-768-Schlüssel [FIPS-203] sein.
6. Die Zeit in "signed_pub_keys.exp" MUSS größer als die aktuelle Systemzeit (Seconds since epoch) sein, d. h. die beiden Schlüssel (ECDH_PK und ML-KEM-768_PK) sind noch zeitlich gültig.

[<=]

A 26939 - ZETA/ASL-Protokoll: ZETA/ASL-Server: Erhalt von Nachricht 3

Ein ZETA/ASL-Server MUSS die für Nachricht 3 erzeugten Datenobjekte nach A 26937-* verarbeiten können.[<=]

A 26940 - ZETA/ASL-Protokoll: ZETA/ASL-Server: Nachricht 4

Ein ZETA/ASL-Server MUSS bei Erhalt der Nachricht 3 das Chifftrat AEAD ct mittels des Schlüssels K1_c2s entschlüsseln. Schlägt dies fehl, MUSS er den Verbindungsaufbau mit einem Fehler (vgl. A 26924-*) abbrechen. Er MUSS analog wie der Client (vgl. A 26937-*) die Schlüsselerzeugung der folgenden Schlüssel durchführen, nur dass er dafür die KEM-Decapsulation-Methode verwendet:

- K2_c2s key confirmation,
- K2_c2s app data,
- K2_s2c key confirmation,
- K2_s2c app data

und

- KeyID (nicht vertraulich).

Er MUSS analog zu A 26937-* das Transskript des Clients und dessen SHA-256-Wert berechnen. Er MUSS mittels K2_c2s key confirmation das Chifftrat "AEAD ct key confirmation" aus Nachricht 3 entschlüsseln und prüfen, ob der von ihm (= ZETA/ASL-Server) berechnete Transskript-Client-Hashwert mit dem entschlüsselten Klartext übereinstimmt. Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen berechnete Transskript-Client-Hashwert und Klartext festgestellt, MUSS er den Handshake mit einem Fehler abbrechen (vgl. A 26924-*).

Er MUSS den Transskript des ZETA/ASL-Servers als die Konkretation von Nachricht1 || Nachricht 2 || Nachricht 3 berechnen, davon den SHA-256-Hashwert berechnen und diesen mittels K2_s2c key confirmation verschlüsseln (vgl. A 26934-*), und erhält ein Chifftrat genannt "AEAD ct key confirmation".

Er MUSS die folgende Datenstruktur erzeugen:

{

```
"MessageType" : "M4",  
"AEAD ct key confirmation" : Chiffprat-AEAD ct key confirmation  
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 4.

Er MUSS die Nachricht 4 inkl. ID (vgl. A 26935-*) an die HTTPS-Schnittstelle des Aktensystem übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 3 eingetroffen ist, die Nachricht 4 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "ZETA/ASL-CID" die ID aufgeführt werden.[<=]

Erläuterung:

Die Aufführung von "ZETA/ASL-CID" im Response-Header ist eigentlich nicht mehr absolut notwendig, da der Client diese schon bei Nachricht 2 erhalten und als URL-Pfadname ab jetzt im Laufe der weiteren Verbindung verwenden wird (A 26936-*). Die Absicht der Aufführung ist, eine etwaige Fehlersuche bspw. bei IOP-Tests zu unterstützen.

A 26941 - ZETA/ASL-Protokoll: ASL/ZT-Client: Erhalt von Nachricht 4

Ein ZETA/ASL-Client MUSS die für Nachricht 4 erzeugten Datenobjekte nach A 26940-* verarbeiten können.

Er MUSS die Nachrichten 1, 2 und 3 konkatenieren und den SHA-256-Hashwert erzeugen. Dies ist der Transskript-Hashwert. Er MUSS mittels K2_s2c_key_confirmation das Chiffprat "AEAD ct key confirmation" entschlüsseln (vgl. A 26934-*). Der erhaltene Klartext ist der gesendete Transskript-Server-Hashwert. Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen eben berechneten Transskript-Hashwert und gesendeten Transskript-Server-Hashwert (Klartext des Chiffrats) festgestellt, MUSS er den Handshake mit einem Fehler abbrechen.[<=]

8.2 Transport und Sicherung der Nutzdaten

A 26926 - ZETA/ASL: ZETA/ASL-Client, Verschlüsselungszähler

Ein ZETA/ASL-Client MUSS sicherstellen, dass der Schlüssel K2_c2s_app_data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Der ZETA/ASL-Client MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffprat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.[<=]

Hinweis: Ein Zählerüberlauf kann praktisch nie erreicht werden. Würde ein Client im Nano-Sekunden-Takt den Zähler erhöhen, würde erst nach mehr als 583 Jahren der Überlauf eintreten. Deshalb wird auf einen Überlauf test verzichtet.

A 26927 - ZETA/ASL: ZETA/ASL-Client: Request erzeugen/verschlüsseln

Ein ZETA/ASL-Client MUSS einen inneren HTTP-Request als Klartext erzeugen.

Er MUSS den Klartext mittels AES/GCM und dem Schlüssel K2_c2s_app_data verschlüsseln (siehe AAD weiter unten).

Dafür wird ein 32-Bit Zufallswert a erzeugt. Nach A 26926-* wird der mit $K2_c2s_app_data$ verbundene Zähler um eins erhöht. Es wird der IV mit $IV=a || \text{Zähler}$ erzeugt (dessen Länge ist damit 96-Bit). Dieser IV wird für die AES/GCM-Verschlüsselung verwendet.

Er MUSS einen Request-Counter pflegen, der verbunden ist mit der KeyID. Für jeden Request MUSS der ZETA/ASL-Client diesen Request-Counter erhöhen, bei Empfang einer Antwort ist es dem Client möglich, die Response zum zuvor gestellten Request sicher zuzuordnen (vgl. A 26931-*, vgl. Erläuterungen zu A 26927-*).

Es wird folgender Header erzeugt:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chifftrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/Request	1 Byte	Für eine Nachricht des ZETA/ASL-Clients an einen ZTA/ASL-Server wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chifftrats hat dieses Byte den Wert 2, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A 26937-*)

Dieser Header stellt die "Additional Associated Data" dar, die in die Berechnung des Authentication-Tag bei der AES/GCM-Verschlüsselung einfließen MÜSSEN. Der Authentication-Tag MUSS 128 Bit lang sein.

Das erweiterte Chifftrat (also inkl. Header) MUSS folgende Struktur haben:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chifftrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.

<u>Name</u>	<u>Länge</u>	<u>Beschreibung bzw. Vorgabe des Werts</u>
<u>Response/Request</u>	<u>1 Byte</u>	<u>Für eine Nachricht des ZETA/ASL-Clients an einen ZETA/ASL-Server wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrats wird es auf den Wert 2 gesetzt, was markiert, dass es sich um eine Response handelt.</u>
<u>Request-Counter</u>	<u>8 Byte</u>	<u>Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.</u>
<u>KeyID</u>	<u>32 Byte</u>	<u>KeyID aus dem Handshake (vgl. A 26937-*)</u>
<u>IV</u>	<u>12 Byte (= 96 Bit)</u>	<u>IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A 26927-*)</u>
<u>CT</u>	<u>variabel</u>	<u>eigentliche AES/GCM-Chifftrat, dessen Länge gleich der Länge des Klartextes ist</u>
<u>GMAC-Wert</u>	<u>16 Byte (= 128 Bit)</u>	<u>Authentication-Tag, der während der AES/GCM-Verschlüsselung inkl. der Associated Data (Daten aus der Header-Tabelle, s. o.) berechnet wird.</u>

Der ZETA/ASL-Client MUSS diese Datenstruktur per HTTP-POST an den ZETA/ASL-Server senden und als URL-Pfadnamen dabei den Wert der ZETA/ASL-CID (vgl. A 26936-*) verwenden. Dabei MUSS er den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden.

[<=]

Erläuterung:

Der Verschlüsselungszähler, der im Client mit dem Schlüssel K2 c2s app data verbunden ist, hat die Funktion für den Galois Counter Mode (GCM) sicherzustellen, dass der jeweils pro Verschlüsselungsvorgang erzeugte Initialisierungsvektor (IV) (bei gleichen Wert von K2 c2s app data) einzigartig ist.

Der Request-ID-Zähler hat drei Funktionen. Einmal soll im Client eine Response des Servers sicher einem vorher gesendeten Request zuordenbar sein. Sollte im Verbindungsaufbau "Enforce Replay Protection" aktiviert worden sein, prüft der Server, ob Requests mit gleicher Request-ID mehrfach eingetroffen sind (ähnlich der "Anti Replay Window"-Technik bei IPsec). Wenn analog "Enforce Sequence Order" aktiviert worden ist, dann prüft der Server die Folge der Request-ID der einkommenden Requests auf strenge Monotonie.

In der aktuellen Ausbaustufe von ePA für alle werden die letzten beiden Funktionen nicht benötigt (wie auch aktuell beim E-Rezept nicht).

Da Verschlüsselungszähler und Request-ID unterschiedliche Funktionen/Motivationen besitzen, sind sie als separate (theoretisch auch im Wert unterschiedliche) Datenobjekte aufgeführt.

A 26928 - ZETA/ASL: ZETA/ASL-Server: Request entschlüsseln/auswerten

Ein Request erreicht den ZETA/ASL-Server über eine HTTPS-Schnittstelle. Im äußeren Request nimmt der ZETA/ASL-Server das Routing mittels der Verbindungs-ID (URL-Pfadname, A 26936-* und A 26927-*) und/oder der KeyID im Header des Chiffrats vor.

Ein ZETA/ASL-Server MUSS bei Erhalt eines Chiffrats nach A 26927-* folgendes prüfen.

1. Ist die Länge der Datenstruktur mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. S. v. korrekte Umgebung, in der auch der ZETA/ASL-Server arbeitet.
3. Ist das Request/Response-Byte gleich 1.
4. Ist die KeyID bekannt.

Sollte eine dieser Prüfungen oder eine weitere der folgenden Prüfungen ein nicht-positives Prüfergebnis ergeben, so MUSS der ZETA/ASL-Server die Verarbeitung des Requests abbrechen und mit einer Fehlermeldung (vgl. A 26924-*) dem Client antworten.

Der ZETA/ASL-Server MUSS den mit der KeyID verbundenen Schlüssel K2 c2s app data für die Entschlüsselung des Chiffrats verwenden und dabei analog A 26927-* den Header als "Additional Associated Data" mit in die Entschlüsselung einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS Abbruch und Fehlermeldung (vgl. A 26924-*) erfolgen. Der ZETA/ASL-Server MUSS den Request-Counter-Wert speichern, der für die Antwort (A 26930-*) benötigt wird.[<=]

Hinweis:

"Enforce Replay Protection" und "Enforce Sequence Order" werden in der aktuellen Ausbaustufe nicht umgesetzt, weil fachlich noch nicht benötigt.

A 26929 - ZETA/ASL: ZETA/ASL-Server, Verschlüsselungszähler

Ein ZETA/ASL-Server MUSS sicherstellen, dass der Schlüssel K2 s2c app data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Ein ZETA/ASL-Server MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.[<=]

Verständnishinweis: analog A 26926-*.

A 26930 - ZETA/ASL: ZETA/ASL-Server, Response erstellen/verschlüsseln

In der ZETA/ASL-Server wird der innere HTTP-Request (Ergebnis aus A 26928-*) fachlich verarbeitet und als Antwort eine innere HTTP-Response erzeugt. Diese ist der Klartext, der jetzt behandelt wird.

Ein ZETA/ASL-Server MUSS analog zu A 26927-* einen Header erzeugen, wobei er

1. beim Response/Request-Byte den Wert 2 verwenden MUSS,
2. im Request-Counter den gespeicherten Wert aus A 26928-* (Eingang des Requests) verwenden MUSS.

Die anderen Header-Variablen MÜSSEN analog zu A 26927-* festgelegt werden. Die Konstruktion des IV MUSS analog zu A 26927-* erfolgen, nur dass für den Verschlüsselungszähler der Wert nach A 26929-* (K2 s2 app data) verwendet werden

MUSS.

Mit dem IV und dem Schlüssel K2_s2c_app_data MUSS der Klartext (s. o.) verschlüsselt werden mit AES/GCM, wobei der Header als "Additional Associated Data" bei der Verschlüsselung mit einfließt.

Das so erzeugte erweiterte Chifftrat (vgl. A_26927-*, also Chifftrat inkl. Header) MUSS an den Client im HTTP-Response-Body versendet werden, wobei das Aktensystem den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden MUSS. [≤]

A_26931 - ZETA/ASL: ZETA/ASL-Client, Response entschlüsseln/auswerten

Ein ZETA/ASL-Client MUSS bei Erhalt einer Antwort gemäß A_26930-* auf einen Request gemäß A_26927-* folgendes prüfen:

1. Ist die Länge der Datenstruktur (erweiterte Chifftrat) mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. S. v. korrekte Umgebung, in der auch der ZETA/ASL-Client arbeitet.
3. Ist das Request/Response-Byte gleich 2.
4. Hat der Response-Counter den von Client erwarteten Wert.
5. Ist die KeyID bekannt.

Ergibt eine der Prüfungen ein nicht-positives Ergebnis, MUSS der Client die Verarbeitung der Response abbrechen.

Er MUSS das Chifftrat mit dem mit der KeyID verbundenen Schlüssel K2_s2c_app_data mittels AES/GCM entschlüsseln und dabei den Header als "Additional Associated Data" in die Entschlüsselung mit einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS er die Verarbeitung des Chifftrats/Response abbrechen. [≤]

8.3 Fehlersignalisierung

A_26924 - ZETA/ASL: ZETA/ASL-Server, Erzeugung von Fehlermeldungen

Eine ZETA/ASL-Server MUSS das Auftreten von Fehlern bei der Abarbeitung des ZETA/ASL-Protokolls wie folgt an den ZETA/ASL-Client melden.

Als HTTP-Response-Code MUSS der ZETA/ASL-Server einen HTTP-Fehlercode (i. S. v. eben nicht 200) in der äußeren HTTP-Response verwenden (siehe folgende Tabelle). Es MUSS weiterhin eine Datenstruktur der folgenden Art erzeugen:

```
{  
    "MessageType" : "Error",  
    "ErrorCode" : ...natürliche-Zahl...,  
    "ErrorMessage" : "... Menschenlesbarer Text bzw. Fehlerursache  
    ..."  
}
```

Diese Datenstruktur MUSS es per CBOR [RFC-CBOR] serialisieren und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) im äußeren Response-Body aufführen.

Ein ZETA/ASL-Server KANN weitere selbst definierte Variablen-Werte-Paare in der o. g. Datenstruktur aufführen.

Folgende Fehler-Meldungen MUSS es mindestens geben:

<u>HTTP-Error</u>	<u>ErrorCode</u>	<u>ErrorMessage / Beschreibung</u>
<u>400</u>	<u>1</u>	<u>"Decoding Error"</u> <u>Fehler in einer Kodierung bspw. der CBOR-Kodierung</u>
<u>400</u>	<u>2</u>	<u>"Missing Parameters"</u> <u>notwendige Datenfelder bspw. in der Handshake fehlen</u>
<u>403</u>	<u>3</u>	<u>"GCM returns FAIL"</u> <u>Eine AES/GCM-Entschlüsselung ergibt das Symbol "FAIL".</u>
<u>403</u>	<u>4</u>	<u>"PU/nonPU Failure"</u>
<u>403</u>	<u>5</u>	<u>"Transscript Error"</u> <u>Im Handshake wird Ungleichheit zwischen den beiden</u> <u>Transskript-Hashwerten festgestellt.</u>
<u>400</u>	<u>6</u>	<u>"bad format: extended ciphertext"</u> <u>Die erste Sanity-Prüfung des erweiterten Chifftrat bspw. bei</u> <u>A 26928-* schlägt fehl.</u>
<u>403</u>	<u>7</u>	<u>"is not a request"</u> <u>A 26928-* (Prüfschritt 3)</u>
<u>403</u>	<u>8</u>	<u>"unknow KeyID"</u>
<u>403</u>	<u>9</u>	<u>"unknown CID"</u>

[<=]

A 26925 - ZETA/ASL: ZETA/ASL-Client, Verarbeitung von Fehlermeldungen

Ein ZETA/ASL-Client MUSS die Fehlermeldungen nach A 26924-* verarbeiten können.

[<=]

8.4 Tracing in Nichtproduktivumgebungen

A 26942 - ZETA/ASL-Client, Nichtproduktivumgebung, Offenlegung von symmetrischen Verbindungsschlüsseln

Ein ZETA/ASL-Client in einer Nichtproduktivumgebung MUSS nach einer erfolgreichen Schlüsselaushandlung mit dem ZETA/ASL-Protokoll die ausgehandelten symmetrischen Schlüssel K2_c2s_app_data und K2_s2c_app_data base64-kodiert innerhalb einer HTTP-Request-Header-Variable "ZETA/ASL-nonPU-Tracing" bei jedem äußeren HTTP-Request aufführen. Dabei sind die beiden Base64-kodierten Schlüsselwerte durch Leerzeichen zu trennen. Beispiel:

ZETA-ASL-nonPU-Tracing: AA=
AAAE=

Als erstes ist der Wert von Schlüssel K2_c2s_app_data aufzuführen, als zweiter Wert ist der Wert von Schlüssel K2_s2c_app_data aufzuführen. [<=]

Hinweis: Ebenfalls muss ein ZETA/ASL-Client nach A 26927-* im Chifftrat kennzeichnen, ob es eine Nachricht in einer Nichtproduktivumgebung ist oder nicht.

A 26943 - ZETA/ASL-Server, Nichtproduktivumgebung, Ablehnung von nonPU-Chiffraten in der PU

Ein ZETA/ASL-Server in der Produktivumgebung MUSS äußere HTTPS-Request, die im Request-Header eine Variable "ZETA/ASL-nonPU-Tracing" enthalten, mit einer HTTP-Fehlermeldung 403-Forbidden beantworten. Der ZETA/ASL-Server MUSS sicherstellen, dass der innere Request (Chifftrat im Request-Body des äußeren Requests) nicht an einen Resource-Server weitergereicht wird.

Ein ZETA/ASL-Server in der Produktivumgebung, die am Chifftrat am nonPU/PU-Byte erkennt (vgl. A 26927-*), dass es sich um ein nonPU-Chifftrat handelt, MUSS den Request mit einer Fehlermeldung an die äußere Schicht des Aktensystems (Webschnittstelle) beantworten. Der ZETA/ASL-Server MUSS sicherstellen, dass das Chifftrat nicht entschlüsselt und das Chifftrat sicher gelöscht wird. [<=]

5203

89 Post-Quanten-Kryptographie (informativ)

- 5204 Wie in [BSI-PQC-2020] dargestellt, erscheint es mit Blick auf dem aktuellen Stand von
5205 Wissenschaft und Technik als sehr unwahrscheinlich, dass aktuell Quanten-Computer in
5206 ausreichender Leistungsstärke verfügbar sind, die der asymmetrischen Kryptographie --
5207 so wie sie aktuell in der TI eingesetzt wird -- gefährlich werden könnten.
- 5208 Dies bedeutet, dass insbesondere Authentisierungsvorgänge, die im "Jetzt" mit
5209 klassischen asymmetrischen kryptographischen Verfahren mit ausreichend großen
5210 Schlüssellängen durchgeführt werden (vgl. [SOG-IS] und [BSI-TR-03116-1]), nicht in
5211 Frage stehen.
- 5212 Schwieriger wird es mit der Verschlüsselung mit klassischen asymmetrischen Verfahren
5213 bei denen Chiffre erzeugt werden, die ein Angreifer sammeln kann, und durch deren
5214 Entschlüsselung zu einer Zeit in der Zukunft, bei der solche Quanten-Computer verfügbar
5215 sind, dann Schaden entstehen kann ("harvest now, decrypt later").
- 5216 Dort erscheint es ratsam auf hybride Lösungen zu migrieren. D. h., etablierte
5217 kryptographische asymmetrische Verfahren und post-quantum-sichere asymmetrische
5218 kryptographische Verfahren werden in einer Weise kombiniert, dass wenn genau nur
5219 eines der beiden Verfahren gebrochen wird, die Sicherheit (Vertraulichkeit und Integrität)
5220 der verschlüsselten Daten nicht gefährdet ist.
- 5221 Aktuell gibt es diesbezüglich verschiedene Forschungsaktivitäten und Vorschläge für eine
5222 entsprechende hybride Protokoll-Erweiterungen von kryptographischen Protokollen wie
5223 TLS, IKE oder SSH [TLS-CC-2021], [ENISA-PQC-2021#6.1]. Die gematik beobachtet
5224 intensiv die laufenden Forschungs- und Evaluierungsaktivitäten [NIST-PQC] die
5225 voraussichtlich 2024 einen geeigneten Stand erreichen werden. Beim VAU-Protokoll für
5226 ePA für alle wird bereits eine PQC-sichere Schlüsselaushandlung auf Basis von [KEM-TLS]
5227 und [IETF-Hybrid-TLS] eingesetzt. D. h., Anwendungsdaten zwischen ePA-Clients und
5228 ePA-VAU sind beim Transport bereits PQC-sicher. In einer nächsten Ausbaustufe
5229 verwendet der E-Rezept-Projekt ebenfalls diese VAU-Protokollvariante für die Verbindung
5230 zwischen E-Rezept-Client und E-Rezept-VAU.
- 5231 Bei Messengern [PQC-Hybrid-Signal] oder bei Web-Browsern [PQC-Hybrid-Chrome]
5232 nimmt die Verwendung von PQC-sicheren Hybrid Verfahren immer mehr zu. Es ist
5233 absehbar, dass dieses Vorgehen bald generell der Stand der Technik ist.

5234

910 Erläuterungen (informativ)

5235

9-110.1 Prüfung auf angreifbare (schwache) Schlüssel

5236

Im Folgerelease wird es in diesem Abschnitt Hinweise für die Anforderungen aus

5237

Abschnitt 2.4.1 geben.

5238

9-210.2 RSA-Schlüssel in X.509-Zertifikaten

5239

In anderen, nicht-TI Public-Key-Infrastrukturen werden öffentliche Schlüssel bei einer Zertifikatsantragsstellung immer mittels ihrem korrespondierenden privaten Schlüssel signiert (vgl. Certificate Signing Request [RFC-2986], proof of possession). Dort kann der TSP sich nach einer erfolgreichen Signaturprüfung sicher sein, dass er aus Kodierungssicht den "richtigen" Schlüssel in den Händen hält, weil ansonsten die Signaturprüfung mit praktischer Sicherheit fehlschlägt. Missverständnisse aufgrund von "falscher" Byte-Order oder verschiedener Kodierung sind somit praktisch (Falsch-Positiv-Rate $< 2^{-100}$) ausgeschlossen. In der PKI der TI werden mehr als 95 % aller Zertifikatserstellungen ohne eine Signatur mittels der jeweiligen privaten Schlüssel durchgeführt. Ein TSP der TI kann damit bei RSA-Schlüsseln – aus den Schlüsselwerten an sich – im Regelfall nicht sicher erkennen, ob eine Fehlkodierung (Missverständnis zwischen Zertifikatsantragssteller und TSP) aufgetreten ist. Es gibt effiziente Möglichkeiten solche Fehlkodierungen zu erkennen. Den Einsatz solcher Möglichkeiten möchte die gematik befördern und gibt mit A_17092 und A_17093 zwei Verfahren als KANN-Anforderungen an.

5254

Die Untersuchungen aus [MK-2016] und [ROCA-2017] zeigen, dass es hilfreich ist sich mit den konkreten Werten der RSA-Schlüssel zu beschäftigen. Die folgenden Verfahren nutzen Struktureigenschaften von RSA-Schlüsseln, die nicht-RSA-Schlüssel im Normalfall nicht vorweisen.

5258

keine kleinen Primteiler:

5259

Der Modulus eines RSA-Schlüssels muss aus genau zwei (oder wenigen [RFC-8017]) "großen" Primfaktoren bestehen. Falls der vom TSP angetroffene Wert durch eine vom TSP vorgegebene Primzahl teilbar ist, so ist der RSA-Schlüssel ungeeignet. Falls dieser Primteiler deutlich kleiner als 2^{1023} ist, so kann es sich nicht um einen korrekten RSA-Schlüssel handeln.

5264

Wird ein Modulus unabsichtlich von einem Sender falsch kodiert, so ist der dadurch entstehende Wert statistisch über alle möglichen Fehlkodierungen betrachtet im Normalfall mit der Wahrscheinlichkeit von 1/2 durch zwei teilbar, mit der Wahrscheinlichkeit von 1/3 durch 3 teilbar, mit einer Wahrscheinlichkeit von 1/5 durch 5 teilbar usw. Falls man nun den Modulus durch die ersten Primzahlen kleiner als 100 (25 Primzahlen) versucht zu teilen (was effizient möglich ist) und eine Teilbarkeit ausschließen kann, so kann man eine unabsichtliche Fehlkodierung mit einer hohen Wahrscheinlichkeit erkennen.

5272 In der gematik wurden mehr als eine Billion (10^{12}) RSA-Schlüssel zufällig erzeugt und
5273 verschiedene Fehlkodierungen dieser Schlüssel auf Primteiler kleiner 100 untersucht. Es
5274 wurden dabei folgende Erkennungsraten festgestellt.

Art der Fehlkodierung	Erkennungsrate in Prozent (auf zwei Nachkommastellen gerundet)
Byte-Order falsch (vertauscht)	76,03 %
Off-by-One (left shift)	100 %
Off-by-One (right shift)	88,07 %
Base64-kodiert anstatt Binär	87,60 %
Base58-kodiert anstatt Binär	88,01 %
Hex-kodiert anstatt Binär	87,91 %

5275 Man muss davon ausgehen, dass die entsprechende Fehlkodierung nicht nur bei einem
5276 einzigen RSA-Schlüssel, sondern bei allen RSA-Schlüsseln eines
5277 Personalisierungsauftrags auftritt. Somit nähert sich die Erkennungsrate exponentiell
5278 100 % an. Beispiel: bei einer Erkennungsrate von 75 % für eine bestimmte Art der
5279 Fehlkodierung (vgl. Tabelle) erhält man bei 1000 RSA-Schlüsseln in Gesamtheit eine
5280 Erkennungsrate von mehr als $1 - 1,15 \cdot 10^{-125}$, also nahe 1.

5281 öffentlicher Exponent ist prim:

5282 Sei e der öffentliche Exponent und n der Modulus eines RSA-Schlüssels. Bei der Wahl von
5283 e ist es notwendig, dass dieser relativ prim zu $\phi(n)$ ist. Um die Schlüsselerzeugung zu
5284 vereinfachen (und zu beschleunigen), wählen jedoch faktisch alle kryptographischen
5285 Softwarebibliotheken, Chipkarten und HSMs e prim. Wenn also dem TSP ein e vorliegt,
5286 das nicht prim ist, so kann er davon ausgehen, dass ein Fehler vorliegt.

5287 Diese Überlegungen führen zu den Tests in A_17092. Diese Tests haben eine Falsch-
5288 Positiv-Rate von 0 und benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

5289 Entropie der Schlüsselkodierung:

5290 Der Modulus eines RSA-Schlüssels muss aus genau zwei (oder wenigen [RFC-8017])
5291 "großen" zufällig gewählten Primfaktoren bestehen. Diese zufällige Wahl hat zur Folge,
5292 dass die Entropie der kodierten Schlüsselwerte eine hohe Entropie im Sinne von
5293 notwendigen Bits pro Byte besitzt. Eine Fehlkodierung wird evtl. weniger Entropie (Bits
5294 pro Byte) besitzen, weil sie, wie die base64-Kodierung, bestimmte Bits immer auf 0
5295 setzt. In [NIST-SP-800-22] werden verschiedene Tests spezifiziert, um die "Zufälligkeit"
5296 einer Zeichenfolge zu bestimmen. Diese Tests zielen auf größere Datengrößen (Längen
5297 der Zeichenfolge) ab und sind nur teilweise dafür geeignet die Kodierung von 256 Byte
5298 langen RSA-Moduli zu bewerten. Anstatt diese Tests als Basis zu verwenden, wird im
5299 Folgenden die klassische Berechnung der Shannon-Entropie vieler RSA-Moduli in
5300 unterschiedlichen Kodierungsformen betrachtet. Von einer Zeichenkette X wird mittels

$$\cancel{H(X) = - \sum_{i=0}^{255} p_i \log p_i}$$

$$H(X) = - \sum_{i=0}^{255} p_i \log p_i$$

die Entropie im Sinne von notwendigen Bits pro Byte des kodierten Schlüsselwertes berechnet. Dabei ist X die Kodierung (Bytefolge) des Schlüssels und p_i die relative Häufigkeit von Byte i (wobei nach Konvention in dem Kontext gilt: $\log(0)=0$).

Unter <https://rosettacode.org/wiki/Entropy> findet man Implementierung dieser Entropie-Berechnungsfunktion in 78 Programmiersprachen.

Die gematik verwendet folgende C-Implementierung:

```
double entropy(char *S, int len) {
    int table[256]={0};
    int i;
    double H=0;

    for(i=0; i<len; i++) {
        table[(unsigned char)S[i]]++;
    }
    for(i=0; i<256; i++) {
        if (table[i]>0) {
            H-= (double) table[i]/len*log2( (double) table[i]/len);
        }
    }
    return H;
}
```

In der gematik wurden mehr als eine Billion (10^{12}) RSA-Schlüssel zufällig erzeugt und verschiedene Fehlkodierungen auf ihre Entropie (notwendigen Bits pro Byte in der Kodierung) untersucht.

Ein Histogramm eines Beispiel-Testlaufs mit mehr als eine Billion (10^{12}) RSA-Schlüsseln:

8	$\geq H(X) > 7,455$	3396
	$7,455 \geq H(X) > 7,2135$	234195871140
	$7,2135 \geq H(X) > 6,9715$	765693789112
	$6,9715 \geq H(X) > 6,7295$	112336352

Es wurde kein Schlüssel gefunden, dessen korrekte Kodierung eine Entropie kleiner als 6,7295 besitzt.

5331 Ein Histogramm eines Beispiel-Testlaufs mit mehr 10 Milliarden ($1 \cdot 10^{10}$) Schlüsseln
5332 (diese wurden jeweils in unterschiedlichen Kodierungsvarianten kodiert und danach
5333 wurde die entropy()-Funktion auf die Kodierung angewendet):

korrekt kodiert (s. o.)	Base64-kodiert	Base58-kodiert	als Hexadezimal-Zahl kodiert
7.40 49808	5.90 9981660	5.80 11220	3.90 10758804076
7.30 97418682	5.80 6902282798	5.70 4102181822	3.80 40835572
7.20 3351624284	5.70 3861576302	5.60 6615817484	3.70 352
7.10 6095663118	5.60 25792616	5.50 81606244	
7.00 1210930726	5.50 6624	5.40 23230	
6.90 43696816			
6.80 256390			
6.70 176			

5334 Diese Überlegungen bezüglich der Entropie der RSA-Schlüsselkodierung führen zu dem
5335 Test in A_17093. Dieser Test hat eine Falsch-Positiv-Rate von weniger als 2^{-40} und
5336 benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

5337

10.11 Anhang – Verzeichnisse

5338

10.11.1 Abkürzungen

Kürzel	Erläuterung
<u>ASL</u>	<u>Additional Security Layer</u>
BS	betreiberspezifischer Schlüssel
C2C	Card to Card
C2S	Card to Server
CEK	Content Encryption Key
CA	Certificate Authority
CBC	Cipher Block Chaining
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DRNG	Deterministic Random Number Generator
eGK	elektronische Gesundheitskarte
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Hash-based Message Authentication Code
IV	Initialization Vector (Initialisierungsvektor) bspw. bei AES-GCM
ICV	Integrity Check Value, Authentisierungswert (MAC) bei AES-GCM

Kürzel	Erläuterung
KDF	Key Derivation Function (Schlüsselableitungsfunktion)
MAC	Message Authentication Code
OCSF	Online Certificate Status Protocol
OID	Object Identifier
OSI	Open Systems Interconnection
SAK	Signaturanwendungskomponente
SGD	Schlüsselgenerierungsdienst
SE	Secure Element
SM	Service Monitoring
TCB	Trusted Computing Base
TI	Telematikinfrastruktur
TLS	Transport Layer Security
TPM	Trusted Plattform Module
TSIG	Transaction Signature
URI	Uniform Resource Identifier
VAU	vertrauenswürdige Ausführungsumgebung, vgl. [gemSpec_Aktensystem_ePAfueralle]
WANDA Basic	Weitere Anwendungen für den Datenaustausch ohne Nutzung der TI oder derer kryptografischen Identitäten

5339 **10-211.2 Glossar**

5340 Das Glossar wird als eigenständiges Dokument, vgl. [gemGlossar] zur Verfügung gestellt.

5341 **10-311.3 Abbildungsverzeichnis**

5342	Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht	29
5343	Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält	87
5344	Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-	
5345	Rezept-VAU	99
5346	Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-	
5347	VAU-Instanz	114
5348	Abbildung 5: OSI-Schichten in einem Aktensystem	115
5349	Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS]	117
5350	Abbildung 7 : OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI.	133
5351	Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht	29
5352	Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält	87
5353	Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-	
5354	Rezept-VAU	99
5355	Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-	
5356	VAU-Instanz	114
5357	Abbildung 5: OSI-Schichten in einem Aktensystem	115
5358	Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS]	117
5359	Abbildung 7 : OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI.	133
5360		

5361 **10-411.4 Tabellenverzeichnis**

5362	Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten	14
5363	Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-	
5364	qualifizierter Signaturen für die Schlüsselgeneration „RSA“	15
5365	Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-	
5366	qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	16
5367	Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter	
5368	elektronischer Signaturen für die Schlüsselgeneration „RSA“	18
5369	Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung	
5370	qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	20
5371	Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate	22
5372	Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate	22

5373	<u>Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs</u>	<u>30</u>
5374	<u>Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten</u>	
5375	<u>elektronischen XML-Signaturen</u>	<u>31</u>
5376	<u>Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen</u>	<u>32</u>
5377	<u>Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung</u>	<u>35</u>
5378	<u>Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC.....</u>	<u>46</u>
5379	<u>Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen</u>	
5380	<u>Schlüssels</u>	<u>47</u>
5381	<u>Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines</u>	
5382	<u>versichertenindividuellen Schlüssels.....</u>	<u>48</u>
5383	<u>Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären</u>	
5384	<u>Daten im Kontext von Dokumentensignaturen</u>	<u>50</u>
5385	<u>Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-</u>	
5386	<u>Dokumentensignaturen</u>	<u>51</u>
5387	<u>Tabelle 17: Tab_KRYPT_ERP Definition Datenstruktur PKI-Zertifikatsliste</u>	<u>101</u>
5388	<u>Tabelle 18 : API von GET /OCSPResponse</u>	<u>102</u>
5389	<u>Tabelle 19: Tab_KRYPT_ERP_FdV_Truststore_aktualisieren.....</u>	<u>105</u>
5390	<u>Tabelle 20: Tab_KRYPT_ERP Kodierung des Chiffrats aus A_20161 *</u>	<u>107</u>
5391	<u>Tabelle 21: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht</u>	
5392	<u>kryptographisch gesicherten VAU-Kommunikation (E-Rezept)</u>	<u>110</u>
5393	<u>Tabelle 1: Tab KRYPT 001 Übersicht über Arten von X.509-Identitäten</u>	<u>14</u>
5394	<u>Tabelle 2: Tab KRYPT 002 Algorithmen für X.509-Identitäten zur Erstellung nicht-</u>	
5395	<u>qualifizierter Signaturen für die Schlüsselgeneration „RSA“</u>	<u>15</u>
5396	<u>Tabelle 3: Tab KRYPT 002a Algorithmen für X.509-Identitäten zur Erstellung nicht-</u>	
5397	<u>qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“</u>	<u>16</u>
5398	<u>Tabelle 4: Tab KRYPT 003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter</u>	
5399	<u>elektronischer Signaturen für die Schlüsselgeneration „RSA“</u>	<u>18</u>
5400	<u>Tabelle 5: Tab KRYPT 003a Algorithmen für X.509-Identitäten zur Erstellung</u>	
5401	<u>qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“</u>	<u>20</u>
5402	<u>Tabelle 6: Tab KRYPT 006 Algorithmen für CV-Zertifikate.....</u>	<u>22</u>
5403	<u>Tabelle 7: Tab KRYPT 007 Algorithmen für CV-CA-Zertifikate.....</u>	<u>22</u>
5404	<u>Tabelle 8: Tab KRYPT 008 Beispiele für solche Algorithmen-URIs</u>	<u>30</u>
5405	<u>Tabelle 9: Tab KRYPT 009 Algorithmen für die Erzeugung von nicht-qualifizierten</u>	
5406	<u>elektronischen XML-Signaturen</u>	<u>31</u>
5407	<u>Tabelle 10: Tab KRYPT 010 Algorithmen für qualifizierte XML-Signaturen</u>	<u>32</u>
5408	<u>Tabelle 11: Tab KRYPT 012 Algorithmen für Card-to-Server-Authentifizierung</u>	<u>35</u>
5409	<u>Tabelle 12: Tab KRYPT 017 Algorithmen für DNSSEC.....</u>	<u>46</u>
5410	<u>Tabelle 13: Tab KRYPT 018 Ablauf zur Berechnung eines versichertenindividuellen</u>	
5411	<u>Schlüssels</u>	<u>47</u>

5412	<u>Tabelle 14: Tab KRYPT 019 eingesetzte Algorithmen für die Ableitung eines</u>	
5413	<u>versichertenindividuellen Schlüssels.....</u>	48
5414	<u>Tabelle 15: Tab KRYPT 020 Algorithmen für die Erzeugung und Prüfung von binären</u>	
5415	<u>Daten im Kontext von Dokumentensignaturen</u>	50
5416	<u>Tabelle 16: Tab KRYPT 021 Algorithmen für die Erzeugung und Prüfung von PDF/A-</u>	
5417	<u>Dokumentensignaturen</u>	51
5418	<u>Tabelle 17: Tab KRYPT ERP Definition Datenstruktur PKI-Zertifikatsliste</u>	101
5419	<u>Tabelle 18 : API von GET /OCSPResponse</u>	102
5420	<u>Tabelle 19: Tab KRYPT ERP FdV Truststore aktualisieren.....</u>	105
5421	<u>Tabelle 20: Tab KRYPT ERP Kodierung des Chiffrats aus A 20161-*</u>	107
5422	<u>Tabelle 21: Tab KRYPT VAUERR Auftretende Fehler bei auf Anwendungsschicht</u>	
5423	<u>kryptographisch gesicherten VAU-Kommunikation (E-Rezept)</u>	110
5424		

5425 **10.5.11.5 Referenzierte Dokumente**

5426 **10.5.11.5.1 Dokumente der gematik**

5427 Die nachfolgende Tabelle enthält die Bezeichnung der in dem vorliegenden Dokument
5428 referenzierten Dokumente der gematik zur Telematikinfrastruktur.

[Quelle]	Herausgeber: Titel
[gemGlossar]	gematik: Glossar der Telematikinfrastruktur
[gemSpec_COS]	gematik: Spezifikation des Card Operating System (COS)
[gemSpec_DS_Anbieter]	Spezifikation Datenschutz- und Sicherheitsanforderungen der TI an Anbieter
[gemSpec_eGK_ObjSys]	gematik: Die Spezifikation der elektronischen Gesundheitskarte (eGK) – Objektsystem
[gemSpec_KT]	gematik: Spezifikation eHealth-Kartenterminal
[gemSpec_MobKT]	gematik: Spezifikation Mobiles Kartenterminal
[gemSpec_SGD_ePA]	gematik: Spezifikation Schlüsselkommentierungsdienst ePA
[gemSpec_SST_FD_VSDM]	gematik: Schnittstellenspezifikation Fachdienste (UFS/VSDD/CMS)

5429 **10.5-211.5.2 Weitere Dokumente**

5430

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[ABR-1999]	DHIES: An Encryption Scheme Based on the Diffie–Hellman Problem Abdalla, Michel and Bellare, Mihir and Rogaway, Phillip, 1999 http://web.cs.ucdavis.edu/~rogaway/papers/dhies.pdf
[AIS-20-1999]	W. Schindler: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators. Version 1.0, 02.12.1999, ehemalige mathematisch technische Anlage zur AIS20, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifierung/Interpretation/AIS20_Functionality_Classes_Evaluation_Methodology_DRNG.pdf?blob=publicationFile
[AIS-20]	AIS 20: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren, Version 3, 15.05.2013, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifierung/Interpretation/AIS_20_pdf.pdf?blob=publicationFile
[AIS-31]	AIS 31: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren, Version 3, 15.05.2013, http://www.bsi.bund.de/SharedDocs/Downloads/DE/BS/Zertifierung/Interpretationen/AIS_31_pdf.pdf?blob=publicationFile
[ALGCAT]	Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen), Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, vom 30.12.2016 (auch online verfügbar: https://www.bundesanzeiger.de mit dem Suchbegriff „BAnz AT 30.12.2016 B5“)
[ANSI-X9.31]	National Institute of Standards and Technology, NIST- Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms, January 31, 2005. http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf
[ANSI-X9.62]	ANSI X9.62:2005 Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[ANSI-X9.63]	American National Standard for Financial Services X9.63–2001 Public Key Cryptography for the Financial Services Industry Key Agreement and Key Transport Using Elliptic Curve Cryptography
[Boyd-Mathuria-2003]	Protocols for Authentication and Key Establishment, Colin Boyd and Anish Mathuria, 2003
[BrainPool]	ECC Brainpool Standard Curves and Curve Generation v. 1.0 19.10.2005 http://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf
[Breaking-TLS]	Lucky Thirteen: Breaking the TLS and DTLS Record Protocols Nadhem J. AlFardan and Kenneth G. Paterson Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, 6th February 2013
[BreakingXMLEnc]	How to Break XML Encryption, Tibor Jager, Juraj Somorovsky, 2011 http://www.nds.rub.de/media/nds/veroeffentlichungen/2011/10/22/HowToBreakXMLEnc.pdf
[BSI-PQC-2020]	Migration zu Post-Quanten-Kryptografie, Handlungsempfehlungen des BSI, Stand: August 2020, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?__blob=publicationFile&v=2
[BSI-TR-02102-1]	BSI TR-02102-1 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“ Version 2024-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.html
[BSI-TR-02102-2]	BSI TR-02102-2 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 2 – Verwendung von Transport Layer Security (TLS), Version 2024-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.html
[BSI-TR-02102-3]	BSI TR-02102-3 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 3 – Verwendung von Internet Protocol Security (IPsec) und Internet Key Exchange (IKEv2)“ Version 2024-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-3.html

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[BSI-TR-03111]	Technical Guideline BSI TR-03111 Elliptic Curve Cryptography, Version 2.10, Date: 2018-06-01 https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03111/TR-03111_node.html
[BSI-TR-03116-1]	Technische Richtlinie BSI TR-03116-1 Kryptographische Vorgaben für Projekte der Bundesregierung, Version: 3.20, Fassung September 2018, 21.09.2018 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116.html
[CM-2014]	20 Years of SSL/TLS Research, An Analysis of the Internet's Security Foundation, Christopher Meyer, 9. February 2014 http://www-brs.ub.ruhr-uni-bochum.de/net/html/HSS/Diss/MeyerChristopher/diss.pdf
[eIDAS]	Verordnung (EU) Nr. 910/2014 des europäischen Parlaments und des Rates vom 23. Juli 2014 über elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt und zur Aufhebung der Richtlinie 1999/93/EG
[ecma-262]	JSON Standard https://www.ecma-international.org/publications/standards/Ecma-262.htm
[EN-14890-1]	DIN EN 14890-1:2008 Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic services
[ENISA-PQC-2021]	Post-Quantum Cryptography: Current state and quantum mitigation, European Union Agency for Cybersecurity (ENISA), May 2021, https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation
[ETSI-CAeS]	ETSI TS 101 733 V1.7.4 (2008-07), Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAeS)
[ETSI_TS_102_231_v3.1.2]	ETSI (Dezember 2009): ETSI Technical Specification TS 102 231 ('Provision of harmonized Trust Service Provider (TSP) status information') – Version 3.1.2
[ETSI-XAdES]	ETSI TS 101 903 V1.4.2 (2010-12), Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[FIPS-180-4]	Federal Information, Processing Standards Publication 180-4, Secure Hash Standard (SHS), March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf
[FIPS-186-2+CN1]	FIPS 186-2 - National Institute of Standards and Technology, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2, January 27, 2000 – Appendix 3.1 unter der Beachtung des Change Notice 1, vom 5. Oktober 2001 http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2-change1.pdf
[FIPS-186-5]	FIPS 186-5 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (Supersedes FIPS 186-4) Digital Signature Standard (DSS), 03.02.2023 https://csrc.nist.gov/pubs/fips/186-5/final https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf
[FIPS-197]	Federal Information Processing Standards Publication 197, (FIPS-197), November 26, 2001, Announcing the ADVANCED ENCRYPTION STANDARD (AES) http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[FIPS-202]	NIST, FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015, http://nvlpubs.nist.gov/nistpubs/fips/NIST.FIPS.202.pdf
[IETF-Hybrid-TLS]	Hybrid key exchange in TLS 1.3 Douglas Stebila, Scott Fluhrer, Shay Gueron https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/
[IETF-KEM-TLS]	KEM-based Authentication for TLS 1.3 T. Wiggers, S. Celi, P. Schwabe, D. Stebila, N. Sullivan https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/
[IETF-Kyber]	Kyber Post-Quantum KEM, P. Schwabe, B. E. Westerbaan Cloudflare, 2024, https://datatracker.ietf.org/doc/draft-cfrg-schwabe-kyber/
[IR-2014]	Bulletproof SSL and TLS: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Ivan Ristić, 2014 https://www.feistyduck.com/books/bulletproof-ssl-and-tls/
[ISO-11770]	ISO/IEC 11770: 1996, Information technology – Security techniques – Key management, Part 3: Mechanisms using asymmetric techniques

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[Ker-1883]	Auguste Kerckhoffs, "La cryptographie militaire", Journal des sciences militaires, vol. IX, Seite 5–83, Jan. 1883, Seite 161–191, Feb. 1883. siehe auch http://www.petitcolas.net/fabien/kerckhoffs/
[KEM-TLS]	Post-quantum TLS without handshake signatures Peter Schwabe, Douglas Stebila, and Thom Wiggers https://eprint.iacr.org/2020/534
[KS-2011]	W. Killmann, W. Schindler, „A proposal for: Functionality classes for random number generators“, Version 2.0, September 2011 https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifizierung/Interpretation/AIS31_Functionality_classes_for_random_number_generators.pdf?__blpb=publicationFile
[MK-2016]	The Million-Key Question – Investigating the Origins of RSA Public Keys, Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, Vashek Matyas, The 25th USENIX Security Symposium (UsenixSec'2016) https://crocs.fi.muni.cz/public/papers/usenix2016
[NIST-PQC]	Post-Quantum Cryptography, National Institute of Standards and Technology (NIST), https://csrc.nist.gov/projects/post-quantum-cryptography
[NIST-SP-800-22]	A. Ruskin et al., A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, SP 800-22 Rev. 1a , 2010 https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
[NIST-SP-800-38A]	NIST Special Publication 800-38A, Recommendation for Block, Cipher Modes of Operation, Methods and Techniques, Morris Dworkin, December 2001 Edition, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
[NIST-SP-800-38B]	NIST Special Publication 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Morris Dworkin, May 2005 Edition, http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
[NIST-SP-800-38D]	NIST Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, Morris Dworkin, November, 2007

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[NIST-SP-800-56-A]	NIST Special Publication 800-56A Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, April 2018 https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final
[NIST-SP-800-56-B]	NIST Special Publication 800-56B Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography, August 2009
[NIST-SP-800-56C]	NIST Special Publication 800-56C Recommendation for Key Derivation through Extraction-then-Expansion, November 2011
[NIST-SP-800-108]	NIST Special Publication 800-108 Recommendation for Key Derivation Using Pseudorandom Functions, October 2009
[NK-PP]	Common Criteria Schutzprofil (Protection Profile) Schutzprofil 1: Anforderungen an den Netzkonnektor, BSI-CC-PP-0097
[Oorschot-Wiener-1996]	On Diffie-Hellman Key Agreement with Short Exponents, Paul C. van Oorschot, Michael J Weiner, Eurocrypt' 96
[Padding-Oracle-2005]	Padding Oracle Attacks on CBC-mode Encryption with Secret and Random IVs Arnold K. L. Yau, Kenneth G. Paterson and Chris J. Mitchell, FSE 2005 http://www.isg.rhul.ac.uk/~kp/secretIV.pdf
[PAdES-3]	ETSI TS 102 778-3 V1.2.1, PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced – PAdES-BES and PAdES-EPES Profiles Technical Specification, 2010
[PDF/A-2]	ISO 19005-2:2011 – Document management – Electronic document file format for long-term preservation – Part 2: Use of ISO 32000-1 (PDF/A-2)
[PKCS#1]	vgl. [RFC-8017]

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[PP-0082]	Common Criteria Protection Profile, Card Operating System Generation 2 (PP COS G2), BSI-CC-PP-0082-V2, Version 1.9, 18th November 2014
[PQC-Hybrid-Chrome]	Protecting Chrome Traffic with Hybrid Kyber KEM, Thursday, August 10, 2023 https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html
[PQC-Hybrid-Signal]	The PQXDH Key Agreement Protocol, Revision 2, 2023-05-24 https://signal.org/docs/specifications/pqxdh/ (vgl. auch https://signal.org/blog/pqxdh/)
[RFC-2119]	RFC 2119 (März 1997): Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, http://tools.ietf.org/html/rfc2119
[RFC-2590]	RFC 2590 (June 1999): X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP https://tools.ietf.org/html/rfc2560 (Obsoleted by [RFC-6960])
[RFC-2986]	RFC 2986 (November 2000): PKCS #10: Certification Request Syntax Specification, Version 1.7 https://tools.ietf.org/html/rfc2986
[RFC-3279]	RFC 3279 (April 2002): Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile https://tools.ietf.org/html/rfc3279
[RFC-3526]	RFC 3526 (Mai 2003: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) http://tools.ietf.org/html/rfc3526
[RFC-4051]	Additional XML Security Uniform Resource Identifiers (URIs), April 2005 https://tools.ietf.org/html/rfc4051

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[RFC-4635]	RFC 4635 (August 2006): HMAC SHA TSIG Algorithm Identifiers http://tools.ietf.org/html/rfc4635
[RFC-5077]	Transport Layer Security (TLS) Session Resumption without Server-Side State, January 2008, https://tools.ietf.org/html/rfc5077
[RFC-5084]	RFC 5084: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS), November 2007 https://tools.ietf.org/html/rfc5084
[RFC-5091]	RFC 5091: Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems, X. Boyen, L. Martin, December 2007 https://tools.ietf.org/html/rfc5091
[RFC-5246]	The Transport Layer Security (TLS) Protocol Version 1.2, August 2008, https://tools.ietf.org/html/rfc5246
[RFC-5280]	RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, Mai 2008 https://tools.ietf.org/html/rfc5280
[RFC-5480]	RFC 5480 (March 2009): Elliptic Curve Cryptography Subject Public Key Information, https://tools.ietf.org/html/rfc5480
[RFC-5639]	RFC 5639 (March 2010): Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, http://www.ietf.org/rfc/rfc5639.txt
[RFC-5652]	RFC 5652 (September 2009): Cryptographic Message Syntax (CMS), R. Housley, http://tools.ietf.org/html/rfc5652
[RFC-5702]	RFC 5702 (October 2009): Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC, http://tools.ietf.org/html/rfc5702
[RFC-5746]	RFC 5746: Transport Layer Security (TLS) Renegotiation Indication Extension, February 2010, https://tools.ietf.org/html/rfc5746

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[RFC-5753]	RFC 5753: Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), January 2010, https://tools.ietf.org/html/rfc5753
[RFC-5869]	HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010, https://tools.ietf.org/html/rfc5869
[RFC-5903]	Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2, June 2010, https://tools.ietf.org/html/rfc5903
[RFC-6090]	RFC 6090: Fundamental Elliptic Curve Cryptography Algorithms, February 2011, https://tools.ietf.org/html/rfc6090
[RFC-6954]	Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2), July 2013, https://tools.ietf.org/html/rfc6954
[RFC-6960]	RFC 6960 (June 2013): X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, https://tools.ietf.org/html/rfc6960
[RFC-7027]	RFC 7027: (October 2013) Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS), https://tools.ietf.org/html/rfc7027
[RFC-7296]	RFC 7296 (October 2014): Internet Key Exchange Protocol Version 2 (IKEv2), https://tools.ietf.org/html/rfc7296
[RFC-7427]	RFC 7427 (January 2015): Signature Authentication in the Internet Key Exchange Version 2 (IKEv2), https://tools.ietf.org/html/rfc7427
[RFC-7519]	RFC 7519 JSON Web Token (JWT), M. Jones, J. Bradley, N. Sakimura, Mai 2015 https://datatracker.ietf.org/doc/html/rfc7519
[RFC-8017], [PKCS#1]	"Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2", November 2016 https://tools.ietf.org/html/rfc8017
[RFC-931]	RFC 6931: Additional XML Security Uniform Resource Identifiers (URIs), Donald Eastlake, April 2013, https://tools.ietf.org/html/rfc6931

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[RFC-9155]	RFC 9155: Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2, December 2021, https://datatracker.ietf.org/doc/rfc9155/
[RFC-CBOR]	RFC-8949: Concise Binary Object Representation (CBOR), C. Bormann, P. Hoffman, December 2020, https://datatracker.ietf.org/doc/html/rfc8949
[ROCA-2017]	The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli, Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, Vashek Matyas 24th ACM Conference on Computer and Communications Security (CCS'2017) https://crocs.fi.muni.cz/public/papers/rsa_ccs17
[SEC1-2009]	Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography, Certicom Research, Contact: Daniel R. L. Brown (dbrown@certicom.com), May 21, 2009, Version 2.0 https://www.secg.org/sec1-v2.pdf
[SDH-2016]	Measuring the Security Harm of TLS Crypto Shortcuts, Drew Springall, Zakir Durumeic, J. Alex Halderman, November 2016, https://jhalderm.com/pub/papers/forward-secrecy-imc16.pdf
[SOG-IS]	SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms, Version 1.3, January 2023 https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf
[TLS-Attacks]	Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses, Christopher Meyer und Jörg Schwenk, 31. Januar 2013, http://eprint.iacr.org/2013/049
[TLS-CC-2021]	Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS), September 2021, https://datatracker.ietf.org/doc/draft-campagna-tls-bike-sike-hybrid/
[XMLCan_V1.0]	Exclusive XML Canonicalization, Version 1.0, W3C Recommendation 18 July 2002, http://www.w3.org/TR/xml-exc-c14n/
[XMLDSig]	XML Signature Syntax and Processing Version 1.1, W3C Recommendation 11 April 2013 https://www.w3.org/TR/xmlsig-core1/

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[XMLDSig-Draft]	XML Signature Syntax and Processing Version 2.0, W3C Editor's Draft 04 February 2014, http://www.w3.org/2008/xmlsec/Drafts/xmlsig-core-20/
[XMLDSig-RSA-PSS]	RSA-PSS in XMLDSig, 25/26 September 2007, Konrad Lanz, Dieter Bratko, Peter Lipp, http://www.w3.org/2007/xmlsec/ws/papers/08-lanz-iaik/
[XMLEnc]	XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002, http://www.w3.org/TR/xmlenc-core/
[XMLEnc-CM]	Technical Analysis of Countermeasures against Attack on XML Encryption - or - Just Another Motivation for Authenticated Encryption. Juraj Somorovsky, Jörg Schwenk. 2011 http://www.w3.org/2008/xmlsec/papers/xmlEncCountermeasuresW3C.pdf
[XMLEnc-1.1]	XML Encryption Syntax and Processing, W3C Recommendation 11 April 2013, http://www.w3.org/TR/xmlenc-core1/
[XSpRES]	XML Spoofing Resistant Electronic Signature (XSpRES) -- Sichere Implementierung für XML-Signaturen Bundesamt für Sicherheit in der Informationstechnik 2012 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SOA/XSpRESS.pfd?__blob=publicationFile
[XSW-Attack]	On Breaking SAML: Be Whoever You Want to Be Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, Meiko Jensen, Usenix 2012 http://www.nds.rub.de/media/nds/veroeffentlichungen/2012/08/03/BreakingSAML.pdf
[Vaudenay-2002]	Security Flaws Induced by CBC Padding: Applications to SSL, IPsec, WTLS ... , Serge Vaudenay, Eurocrypt 2002, LNCS 2332/2002, 535-545 https://www.iacr.org/cryptodb/data/paper.php?pubkey=2850

5431
5432