

Beim vorliegenden Dokument handelt es sich um einen Entwurf der gematik in Vorbereitung auf zukünftige normative Festlegungen als Grundlage entsprechender Zulassungs- und Bestätigungsverfahren. Die gematik veröffentlicht diesen Entwurf mit dem Ziel, dass sich Interessierte bereits frühzeitig einen Überblick über die mögliche Weiterentwicklung der Telematikinfrastuktur verschaffen können. Die gematik übernimmt keine Gewähr für die Aktualität, Richtigkeit und Vollständigkeit dieses Entwurfes und behält sich das Recht vor, ohne vorherige Ankündigung Änderungen oder Ergänzungen vorzunehmen oder von den Regelungen insgesamt bzw. teilweise Abstand zu nehmen.

Elektronische Gesundheitskarte und Telematikinfrastuktur

Übergreifende Spezifikation Verwendung kryptographischer Algorithmen in der Telematikinfrastuktur

Version: ~~2.17~~18.0 [CC](#)
Revision: ~~242991~~269877
Stand: ~~30-06~~17.08.2020
Status: [zur Abstimmung freigegeben](#)
Klassifizierung: öffentlich [Entwurf](#)
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
1.4.0	03.07.08	freigegeben (für Release 2.3.4)	gematik
1.9.0	26.06.12	Kommentierung	gematik
1.10.0	13.09.12	Einarbeitung der Gesellschafterkommentare	gematik
2.0.0	15.10.12	bQS Kommentare eingearbeitet	gematik
2.1.0	06.06.13	Erweiterung im Rahmen der PP-Erstellung Konnektor (kryptographische Vorgaben für die SAK); Anpassung an das fortgeschriebene PP Konnektor ORS1 (BSI-CC-PP-0046), Konsistenz zur veränderten gemSpec_Kon herstellen	gematik
2.2.0	21.02.14	Losübergreifende Synchronisation	gematik
2.3.0	17.06.14	Entfernung des CBC-Modus bei der Dokumentenver- und -entschlüsselung gemäß P11-Änderungsliste	gematik
2.4.0	17.07.15	Einarbeitung Änderungen aus Errata 1.4.6	gematik
2.5.0	03.05.16	Anpassungen zum Online-Produktivbetrieb (Stufe 1)	gematik
2.6.0	24.08.16	Einarbeitung weiterer Kommentare	gematik
2.7.0	28.10.16	Anpassungen gemäß Änderungsliste	gematik
2.8.0	20.04.17	Start der Migration 120-Bit-Sicherheitsniveau kryptographische Verfahren in der TI (PKI der Kartengeneration 2.1), Anpassungen gemäß Änderungsliste	gematik
2.9.0	19.12.17	C_6260 (IPsec), C_6289 (qeS)	gematik

2.10.0	14.05.18	C_6195 (IPsec), C_6374 (ed. IPsec), C_6375 (TLS für SM), C_6399 (IPsec)	gematik
2.11.0	26.10.18	Veröffentlichung im Rahmen von Release 2.1.3	gematik
2.11.0	29.10.18	C_6639 (RSA 2048 Bit Zertifikate) etc., redaktionelle Aktualisierung der Anforderungen A_14653, A_15699 im Rahmen von Release 2.1.3	gematik
2.12.0	18.12.18	ePA-Inhalte ergänzt	gematik
2.13.0	15.05.19	Anpassungen gemäß Änderungsliste P18.1	gematik
2.14.0	28.06.19	Anpassungen gemäß Änderungsliste P19.1	gematik
2.15.0	02.10.19	Anpassungen gemäß Änderungsliste P20.1/2	gematik
2.16.0	02.03.20	Anpassungen gemäß Änderungsliste P20.4/21.1	gematik
2.17.0	30.06.20	Anpassungen gemäß Änderungsliste P22.1 und Scope-Themen aus Systemdesign R4.0.0	gematik
2.18.0 CC	17.08.20	Einarbeitung Scope-Themen zu R4.0.1 zur Abstimmung freigegeben	gematik

Inhaltsverzeichnis

1	Einführung	1010
1.1	Zielsetzung und Einordnung des Dokuments	1010
1.2	Zielgruppe	1010
1.3	Geltungsbereich	1111
1.4	Abgrenzung des Dokuments	1111
1.5	Methodik	1111
2	Einsatzszenarioübergreifende Algorithmen	1212
2.1	Identitäten	1212
2.1.1	X.509-Identitäten	1212
2.1.1.1	Digitale nicht-qualifizierte elektronische Signaturen	1414
2.1.1.2	Qualifizierte elektronische Signaturen	1616
2.1.1.3	TLS-Authentifizierung	1818
2.1.1.4	IPsec-Authentifizierung	1818
2.1.1.5	Digitale Signaturen durch TI-Komponenten	1918
2.1.1.6	Verschlüsselung	1918
2.1.2	CV-Identitäten	1919
2.1.2.1	CV-Zertifikate G2	1919
2.1.2.2	CV-Certification Authority (CV-CA) Zertifikat G2	2019
2.2	Zufallszahlengeneratoren	2020
2.3	Hilfestellung bei der Umsetzung (Zufallsgeneratoren)	2020
2.4	Schlüsselerzeugung und Schlüsselbestätigung	2121
2.4.1	Prüfung auf angreifbare (schwache) Schlüssel	2322
2.4.2	ECC-Schlüssel in X.509-Zertifikaten	2323
2.4.3	RSA-Schlüssel in X.509-Zertifikaten	2423
3	Konkretisierung der Algorithmen für spezifische Einsatzszenarien	2525
3.1	Kryptographische Algorithmen für XML-Dokumente	2525
3.1.1	XML-Signaturen für nicht-qualifizierte Signaturen	2726
3.1.2	XML-Signaturen für qualifizierte elektronische Signaturen	2828
3.1.3	Webservice Security Standard (WSS)	2929
3.1.4	XML-Verschlüsselung – Symmetrisch	2929
3.1.5	XML-Verschlüsselung – Hybrid	3030
3.2	Karten-verifizierbare Authentifizierung und Verschlüsselung	3030
3.2.1	Card-to-Card-Authentisierung G2	3030
3.2.2	Card-to-Server (C2S) Authentisierung und Trusted Channel G2	3130
3.3	Netzwerkprotokolle	3131
3.3.1	IPsec-Kontext	3231
3.3.2	TLS-Verbindungen	3433
3.3.3	DNSSEC-Kontext	4241

73	3.4 Masterkey-Verfahren (informativ).....	4341
74	3.5 Hybride Verschlüsselung binärer Daten.....	4443
75	3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten	4543
76	3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten	4544
77	3.6 Symmetrische Verschlüsselung binärer Daten.....	4644
78	3.7 Signatur binärer Inhaltsdaten (Dokumente)	4645
79	3.8 Signaturen innerhalb von PDF/A-Dokumenten.....	4746
80	3.9 Kartenpersonalisierung.....	4847
81	3.10 Bildung der pseudonymisierten Versichertenidentität.....	4947
82	3.11 Spezielle Anwendungen von Hashfunktionen	4947
83	3.11.1 Hashfunktionen und OCSP (informativ)	5048
84	3.12 kryptographische Vorgaben für die SAK des Konnektors.....	5149
85	3.13 Migration im PKI-Bereich	5149
86	3.14 Spezielle Anwendungen von kryptographischen Signaturen.....	5150
87	3.15 ePA-spezifische Vorgaben	5251
88	3.15.1 Verbindung zur VAU	5251
89	3.15.2 Vorgaben für ePA-Schlüssel und ePA-Chiffre	5452
90	3.15.3 ePA-Aktensysteminterne Schlüssel	5452
91	3.15.4 ePA-spezifische TLS-Vorgaben	5654
92	3.15.5 Schlüsselableitungsfunktionalität ePA.....	5755
93	3.16 E-Rezept-spezifische Vorgaben (informativ).....	5957
94	3.17 KOM-LE-spezifische Vorgaben	6057
95	4 Umsetzungsprobleme mit der TR-03116-1.....	6159
96	4.1 XMLDSig und PKCS1 v2.1	6159
97	4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM.....	6159
98	4.3 XML Signature Wrapping und XML Encryption Wrapping	6260
99	4.4 Güte von Zufallszahlen	6260
100	5 Migration 120-Bit-Sicherheitsniveau.....	6361
101	5.1 PKI-Begriff-Schlüsselgeneration.....	6361
102	5.2 X.509-Root der TI.....	6462
103	5.3 TSL-Dienst und ECDSA-basierte TSL allgemein.....	6664
104	5.4 ECC-Unterstützung bei TLS.....	6664
105	5.5 ECC-Unterstützung bei IPsec.....	6866
106	5.6 ECDSA-Signaturen.....	7068
107	5.6.1 ECDSA-Signaturen im XML-Format.....	7068
108	5.6.2 ECDSA-Signaturen im CMS-Format	7068
109	5.7 ECIES.....	7169
110	5.7.1 ECIES und authentifizierte Broadcast-Encryption	7673

5.7.2 ECIES und mobKT	7674
5.8 ECC Migration eHealth-KT	7774
5.9 ECC Migration Konnektor	7876
5.10 Verschiedene Produkttypen und ECC Migration (informativ)	8077
6 Kommunikationsprotokoll zwischen ePA-VAU und ePA-Clients	8178
6.1 Motivation	8178
6.2 Übersicht	8178
6.3 VAUClientHello-Nachricht	8380
6.4 VAUServerHello-Nachricht	8481
6.5 Schlüsselableitung	8683
6.6 VAUClientSigFin-Nachricht	8783
6.7 VAUServerFin-Nachricht	8884
6.8 Nutzerdatentransport	8985
6.9 VAUServerError-Nachricht	9288
6.10 Abbrechen des Protokollablaufs	9389
6.11 VAU-Kanal und MTOM/XOP	9389
7 Kommunikationsprotokoll zwischen E-Rezept-VAU und E-Rezept-Clients	9692
7.1 Übersicht (informativ)	9692
7.2 Definition	9893
8 Erläuterungen (informativ)	104100
8.1 Prüfung auf angreifbare (schwache) Schlüssel	104100
8.2 RSA-Schlüssel in X.509-Zertifikaten	104100
9 Anhang – Verzeichnisse	108104
9.1 Abkürzungen	108104
9.2 Glossar	109105
9.3 Abbildungsverzeichnis	110106
9.4 Tabellenverzeichnis	110106
9.5 Referenzierte Dokumente	111107
9.5.1 Dokumente der gematik	111107
9.5.2 Weitere Dokumente	112108
1 Einführung	1010
1.1 Zielsetzung und Einordnung des Dokuments	1010

145	1.2 Zielgruppe	1010
146	1.3 Geltungsbereich	1111
147	1.4 Abgrenzung des Dokuments	1111
148	1.5 Methodik	1111
149	2 Einsatzszenarioübergreifende Algorithmen	1212
150	2.1 Identitäten	1212
151	2.1.1 X.509-Identitäten	1212
152	2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen	1414
153	2.1.1.2 Qualifizierte elektronische Signaturen	1616
154	2.1.1.3 TLS-Authentifizierung	1818
155	2.1.1.4 IPsec-Authentifizierung	1818
156	2.1.1.5 Digitale Signaturen durch TI-Komponenten	1918
157	2.1.1.6 Verschlüsselung	1918
158	2.1.2 CV-Identitäten	1919
159	2.1.2.1 CV-Zertifikate G2	1919
160	2.1.2.2 CV-Certification-Authority (CV-CA) Zertifikat G2	2019
161	2.2 Zufallszahlengeneratoren	2020
162	2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren)	2020
163	2.4 Schlüsselerzeugung und Schlüsselbestätigung	2121
164	2.4.1 Prüfung auf angreifbare (schwache) Schlüssel	2322
165	2.4.2 ECC-Schlüssel in X.509-Zertifikaten	2323
166	2.4.3 RSA-Schlüssel in X.509-Zertifikaten	2423
167	3 Konkretisierung der Algorithmen für spezifische	
168	Einsatzszenarien	2525
169	3.1 Kryptographische Algorithmen für XML-Dokumente	2525
170	3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen	2726
171	3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen	2828
172	3.1.3 Webservice Security Standard (WSS)	2929
173	3.1.4 XML-Verschlüsselung – Symmetrisch	2929
174	3.1.5 XML-Verschlüsselung – Hybrid	3030
175	3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung	3030
176	3.2.1 Card-to-Card-Authentisierung G2	3030
177	3.2.2 Card-to-Server (C2S) Authentisierung und Trusted Channel G2	3130
178	3.3 Netzwerkprotokolle	3131
179	3.3.1 IPsec-Kontext	3231
180	3.3.2 TLS-Verbindungen	3433
181	3.3.3 DNSSEC-Kontext	4241
182	3.4 Masterkey-Verfahren (informativ)	4341
183	3.5 Hybride Verschlüsselung binärer Daten	4443
184	3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten	4543
185	3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten	4544
186	3.6 Symmetrische Verschlüsselung binärer Daten	4644
187	3.7 Signatur binärer Inhaltsdaten (Dokumente)	4645

188	3.8 Signaturen innerhalb von PDF/A-Dokumenten	4746
189	3.9 Kartenpersonalisierung	4847
190	3.10 Bildung der pseudonymisierten Versichertenidentität	4947
191	3.11 Spezielle Anwendungen von Hashfunktionen	4947
192	3.11.1 Hashfunktionen und OCSP (informativ)	5048
193	3.12 kryptographische Vorgaben für die SAK des Konnektors	5149
194	3.13 Migration im PKI-Bereich	5149
195	3.14 Spezielle Anwendungen von kryptographischen Signaturen	5150
196	3.15 ePA-spezifische Vorgaben	5251
197	3.15.1 Verbindung zur VAU	5251
198	3.15.2 Vorgaben für ePA-Schlüssel und ePA-Chiffre	5452
199	3.15.3 ePA-Aktensysteminterne Schlüssel	5452
200	3.15.4 ePA-spezifische TLS-Vorgaben	5654
201	3.15.5 Schlüsselableitungsfunktionalität ePA	5755
202	3.16 E-Rezept-spezifische Vorgaben (informativ)	5957
203	3.17 KOM-LE-spezifische Vorgaben	6057
204	4 Umsetzungsprobleme mit der TR-03116-1	6159
205	4.1 XMLDSig und PKCS1-v2.1	6159
206	4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM	6159
207	4.3 XML Signature Wrapping und XML Encryption Wrapping	6260
208	4.4 Güte von Zufallszahlen	6260
209	5 Migration 120-Bit-Sicherheitsniveau	6361
210	5.1 PKI-Begriff Schlüsselgeneration	6361
211	5.2 X.509-Root der TI	6462
212	5.3 TSL-Dienst und ECDSA-basierte TSL allgemein	6664
213	5.4 ECC-Unterstützung bei TLS	6664
214	5.5 ECC-Unterstützung bei IPsec	6866
215	5.6 ECDSA-Signaturen	7068
216	5.6.1 ECDSA-Signaturen im XML-Format	7068
217	5.6.2 ECDSA-Signaturen im CMS-Format	7068
218	5.7 ECIES	7169
219	5.7.1 ECIES und authentifizierte Broadcast-Encryption	7673
220	5.7.2 ECIES und mobKT	7674
221	5.8 ECC-Migration eHealth-KT	7774
222	5.9 ECC-Migration Konnektor	7876
223	5.10 Verschiedene Produkttypen und ECC-Migration (informativ)	8077

6	Kommunikationsprotokoll zwischen ePA-VAU und ePA-Clients	8178
6.1	Motivation	8178
6.2	Übersicht	8178
6.3	VAUClientHello-Nachricht	8380
6.4	VAUServerHello-Nachricht	8481
6.5	Schlüsselableitung	8683
6.6	VAUClientSigFin-Nachricht	8783
6.7	VAUServerFin-Nachricht	8884
6.8	Nutzerdatentransport	8985
6.9	VAUServerError-Nachricht	9288
6.10	Abbrechen des Protokollablaufs	9389
6.11	VAU-Kanal und MTOM/XOP	9389
6.12	Zusätzliche HTTP-Header-Informationen	9590
7	Kommunikationsprotokoll zwischen E-Rezept-VAU und E-Rezept-Clients	9692
7.1	Übersicht (informativ)	9692
7.2	Definition	9893
8	Erläuterungen (informativ)	104100
8.1	Prüfung auf angreifbare (schwache) Schlüssel	104100
8.2	RSA-Schlüssel in X.509-Zertifikaten	104100
9	Anhang – Verzeichnisse	108104
9.1	Abkürzungen	108104
9.2	Glossar	109105
9.3	Abbildungsverzeichnis	110106
9.4	Tabellenverzeichnis	110106
9.5	Referenzierte Dokumente	111107
9.5.1	Dokumente der gematik	111107
9.5.2	Weitere Dokumente	112108

254

1 Einführung

255

1.1 Zielsetzung und Einordnung des Dokuments

256
257
258

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

259
260
261
262
263
264
265
266
267
268
269

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.

270
271
272
273
274
275
276

Dieses Dokument folgt den Konventionen der TR. Diese hat einen Betrachtungszeitraum von sechs bzw. sieben Jahren. Analog zu Kapitel 1 [BSI-TR-03116-1] bedeutet eine Aussage „Algorithmus X ist geeignet bis Ende 2023+“ generell nicht, dass Algorithmus X nach Ende 2023 nicht mehr geeignet ist, sondern lediglich, dass über die Eignung nach Ende 2023 in der TR 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“.

277
278
279
280
281
282
283
284
285
286

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.

287

1.2 Zielgruppe

288
289

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

1.3 Geltungsbereich

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

Schutzrechts-/Patentrechtshinweis

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

1.4 Abgrenzung des Dokuments

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

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

1.5 Methodik

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

Sie werden im Dokument wie folgt dargestellt:

<AFO-ID> - <Titel der Afo>

Text / Beschreibung

[<=]

Dabei umfasst die Anforderung sämtliche zwischen Afo-ID und der Textmarke [<=] 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_3080GS-A 3080GS-A 3080 - asymmetrischen Schlüssel maximale Gültigkeitsdauer

asymmetrischen Schlüssel maximale Gültigkeitsdauer

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

[<=<=<=]

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.

364 **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

365 Für den Aufbau der X.509-Zertifikate gelten die Vorgaben aus den jeweiligen
366 Spezifikationen der X.509-Zertifikate.

2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen

GS-A_4357[GS-A_4357](#)[GS-A_4357](#) - X.509-Identitäten für die Erstellung und Prüfung digitaler nicht-qualifizierter elektronischer SignaturenX.509-Identitäten für die Erstellung und Prüfung digitaler nicht-qualifizierter elektronischer SignaturenX.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 2023 [BSI-TR-03116-1], 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 2023 [BSI-TR-03116-1], vgl. auch A_15590

A_15590[A_15590](#)[A_15590](#) - Zertifikatslaufzeit bei Erstellung von X.509-Zertifikaten mit RSA 2048 BitZertifikatslaufzeit bei Erstellung von X.509-Zertifikaten mit RSA 2048 BitZertifikatslaufzeit 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.

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

Anwendungsfall	Vorgabe
Art und Kodierung des öffentlichen Schlüssels	<p>ecPublicKey {OID 1.2.840.10045.2.1} auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] zulässig bis Ende 2023+</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>
<p>Signatur eines Zertifikats</p> <p>Signatur einer OCSP-Response</p> <p>Signatur eines OCSP-Responder-Zertifikats</p> <p>Signatur einer CRL</p> <p>Signatur des Zertifikats das Basis der Signaturprüfung einer CRL ist</p>	<p>ecdsa-with-SHA256 [RFC-3279] {OID 1.2.840.10045.4.3.2} auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] zulässig bis Ende 2023+</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>

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

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

A_19073A 19073A 19073 - Feste Laufzeit CV-Zertifikate einer Karte (eGK/HBA/SMC-B)
Feste Laufzeit CV-Zertifikate einer Karte (eGK/HBA/SMC-B)
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. [$\leq \leq \leq$]

A_19173A 19173A 19173 - Feste Laufzeit X.509-Zertifikate einer Karte (eGK/HBA/SMC-B)
Feste Laufzeit X.509-Zertifikate einer Karte (eGK/HBA/SMC-B)
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. [$\leq \leq \leq$]

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

2.1.1.2 Qualifizierte elektronische Signaturen

GS-A_4358GS-A 4358GS-A 4358 - X.509-Identitäten für die Erstellung und Prüfung qualifizierter elektronischer Signaturen
X.509-Identitäten für die Erstellung und Prüfung qualifizierter elektronischer Signaturen
X.509-Identitäten für die Erstellung und Prüfung qualifizierter elektronischer Signaturen

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

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

[$\leq \leq \leq$]

Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung 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).
Art und Kodierung des öffentlichen EE-Schlüssels	<u>RSA-Signaturvariante:</u> Entweder OID 1.2.840.113549.1.1.1 (rsaEncryption) (zulässig bis Ende 2022 [SOG-IS-2018]) oder OID 1.2.840.113549.1.1.10 (id-RSASSA-PSS) [RFC-5756].

	<p>(ohne zeitliche Beschränkung der Zulässigkeit [SOG-IS-2018]) Die Auswahl obliegt dem EE-Zertifikatsausgebenden VDA.</p> <p><u>RSA-Schlüssellänge:</u> zu verwendende Schlüssellänge: 2048 Bit, zulässig bis Ende 2024 [SOG-IS-2018]</p>
<p>Signatur eines Zertifikats, Signatur einer OCSP-Response oder Signatur eines OCSP-Responder-Zertifikates</p>	<p>Entweder sha256withRSAEncryption (OID 1.2.840.113549.1.1.11) (zulässig bis Ende 2022 [SOG-IS-2018]) oder id-RSASSA-PSS (1.2.840.113549.1.1.10) [RFC-5756] (ohne zeitliche Beschränkung der Zulässigkeit [SOG-IS-2018])</p> <p>zu verwendende Schlüssellänge: 2048 Bit, zulässig bis Ende 2024 [SOG-IS-2018]</p> <p>Die Hashfunktion für die Hashwertberechnung der TBSCertificate-Datenstruktur MUSS eine nach [SOG-IS-2018] 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>

449
450

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

Anwendungsfall	Vorgabe
<p>Signatur des VDA-Zertifikats</p>	<p>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).</p>

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 Ende 2023+ 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 Ende 2023+ vgl. Beispiel in Abschnitt 5.2

2.1.1.3 TLS-Authentifizierung

GS-A_4359[GS-A_4359](#)[GS-A_4359](#) - X.509-Identitäten für die Durchführung einer TLS-Authentifizierung
[X.509-Identitäten für die Durchführung einer TLS-Authentifizierung](#)
[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.

[<=<=<=]

2.1.1.4 IPsec-Authentifizierung

GS-A_4360[GS-A_4360](#)[GS-A_4360](#) - X.509-Identitäten für die Durchführung der IPsec-Authentifizierung
[X.509-Identitäten für die Durchführung der IPsec-Authentifizierung](#)
[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.

[<=<=<=]

2.1.1.5 Digitale Signaturen durch TI-Komponenten

GS-A_4361[GS-A_4361](#)[GS-A_4361](#) - X.509-Identitäten für die Erstellung und Prüfung digitaler Signaturen
[X.509-Identitäten für die Erstellung und Prüfung digitaler Signaturen](#)
[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.

[<=<=<=]

2.1.1.6 Verschlüsselung

GS-A_4362[GS-A_4362](#)[GS-A_4362](#) - X.509-Identitäten für Verschlüsselungszertifikate
[X.509-Identitäten für Verschlüsselungszertifikate](#)
[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.

[<=<=<=]

2.1.2 CV-Identitäten

CV-Identitäten werden für die Authentifizierung zwischen Karten verwendet.

2.1.2.1 CV-Zertifikate G2

GS-A_4365[GS-A_4365](#)[GS-A_4365](#) - CV-Zertifikate G2
[CV-Zertifikate G2](#)
[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.

[<=<=<=]

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}	256 Bit bis Ende 2023+

	Authentisierung mit Sessionkey-Aushandlung [RFC-5639#3.4, brainpoolP256r1] authS_gemSpec-COS-G2_ecc-with-sha256 {OID 1.3.36.3.5.3.1}	
Signatur des Endnutzerzertifikats	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2023+

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

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

GS-A_4366GS-A 4366GS-A 4366 - CV-CA-Zertifikate G2CV-CA-Zertifikate G2

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

[<=<=<=]

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 2023+
Signatur des CA-Zertifikates	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2023+

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

2.2 Zufallszahlengeneratoren

GS-A_4367GS-A 4367GS-A 4367

- ZufallszahlengeneratorZufallszahlengeneratorZufallszahlengenerator

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

[<=<=<=]

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_4368GS-A 4368GS-A 4368

- SchlüsselerzeugungSchlüsselerzeugungSchlüsselerzeugung

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_5021GS-A 5021GS-A 5021 - Schlüsselerzeugung bei einer
SchlüsselspeicherpersonalisierungSchlüsselerzeugung bei einer
SchlüsselspeicherpersonalisierungSchlüsselerzeugung 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_5338GS-A 5338GS-A 5338 - HBA/SMC-B – Erzeugung asymmetrischer
Schlüsselpaare auf der jeweiligen Karte selbstHBA/SMC-B – Erzeugung
asymmetrischer Schlüsselpaare auf der jeweiligen Karte selbstHBA/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_5386GS-A 5386GS-A 5386 - kartenindividuelle geheime und private
Schlüssel G2-Kartenkartenindividuelle geheime und private Schlüssel G2-
Kartenkartenindividuelle 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.

2.4.1 Prüfung auf angreifbare (schwache) Schlüssel

A_17294A_17294A_17294 - TSP-X.509: Prüfung auf angreifbare (schwache) Schlüssel **TSP-X.509: Prüfung auf angreifbare (schwache) Schlüssel** **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. [$\leq \leq \leq$]

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.

2.4.2 ECC-Schlüssel in X.509-Zertifikaten

GS-A_5518GS-A_5518GS-A_5518 - Prüfung Kurvenpunkte bei einer Zertifikatserstellung **Prüfung Kurvenpunkte bei einer Zertifikatserstellung** **Prüfung Kurvenpunkte bei einer Zertifikatserstellung**

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

[$\leq \leq \leq$]

A_17091A 17091A 17091 - ECC-SchlüsselkodierungECC-SchlüsselkodierungECC-Schlüsselkodierung

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

[<=<=<=]

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

2.4.3 RSA-Schlüssel in X.509-Zertifikaten

A_17092A 17092A 17092 - RSA-Schlüssel Zertifikatserstellung, keine kleinen Primteiler und e ist primRSA-Schlüssel Zertifikatserstellung, keine kleinen Primteiler und e ist primRSA-Schlüssel Zertifikatserstellung, keine kleinen Primteiler und e ist prim

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

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

[<=<=<=]

Erläuterungen zu A_17092 befinden sich in Abschnitt 8.2.

A_17093A 17093A 17093 - RSA-Schlüssel Zertifikatserstellung, Entropie der SchlüsselkodierungRSA-Schlüssel Zertifikatserstellung, Entropie der SchlüsselkodierungRSA-Schlüssel Zertifikatserstellung, Entropie der Schlüsselkodierung

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

1. Ist die Entropie des kodierten RSA-Schlüssels (im Sinne von [gemSpec_Krypt#8.2], entropy()-Funktion) kleiner als 6.72? Falls ja, so ist das Ergebnis FAIL.

[<=<=<=]

Erläuterungen zu A_17093 befinden sich in Abschnitt 8.2.

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.

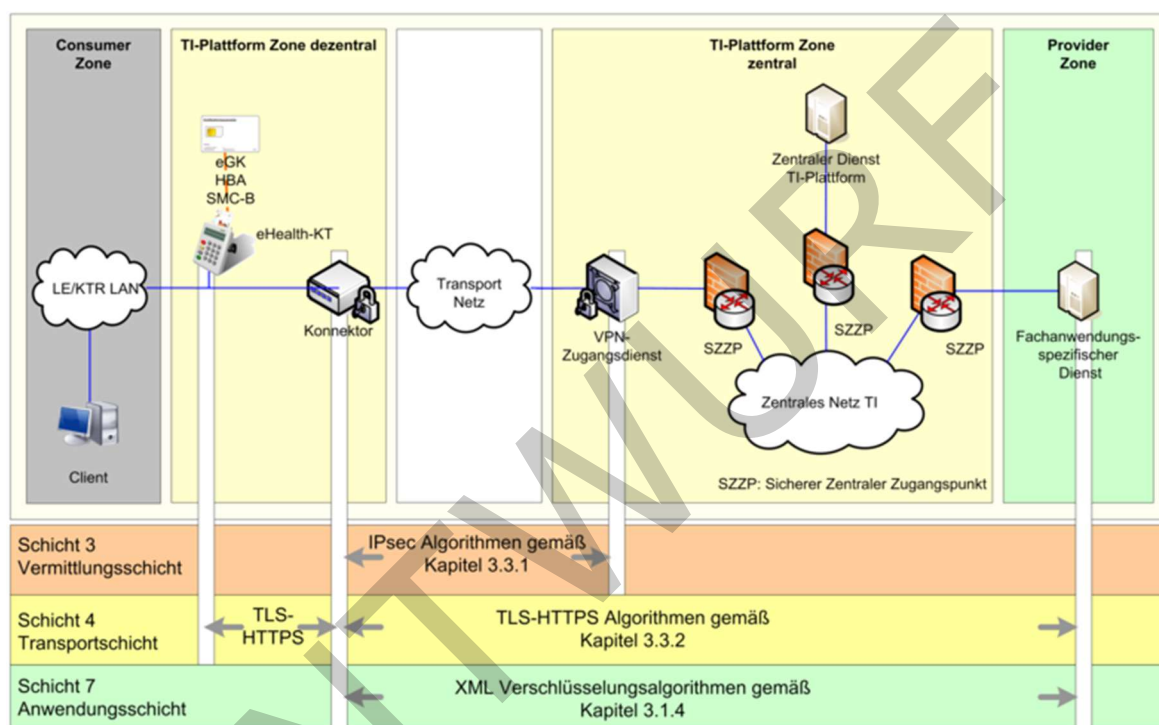


Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht

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 **GS-A_4370** **GS-A_4370** - Kryptographische Algorithmen für XML-Dokumente
Kryptographische Algorithmen für XML-Dokumente
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.

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/xmlsig#enveloped-signature	[XMLDSig]
http://www.w3.org/2001/04/xmlsig-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/xmlsig-more#sha256-rsa-MGF1	[RFC-6931]

3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen

GS-A_4371GS-A_4371GS-A_4371 - XML-Signaturen für nicht-qualifizierte SignaturenXML-Signaturen für nicht-qualifizierte SignaturenXML-Signaturen für nicht-qualifizierte Signaturen

Alle Produkttypen, die XML-Signaturen für nicht-qualifizierte Signaturen erzeugen oder prüfen, MÜSSEN die Algorithmen und Vorgaben der Tabelle Tab_KRYPT_009 erfüllen.

[<=<=<=<=]

Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten 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 und die Verschlüsselung mit dem privaten Schlüssel	RSASSA-PSS mit SHA256 bis nach Ende 2024+ verwendbar (Ende des Betrachtungshorizonts) (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	SHA-256 Die [XMLDSig] konforme Bezeichnung lautet:	Die Verwendung des Algorithmus

	signierenden Bereiche	http://www.w3.org/2001/04/xmenc#sha256	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	Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.

3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen

GS-A_4372 **GS-A_4372** **GS-A_4372** - XML-Signaturen für qualifizierte elektronische Signaturen
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.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des	RSASSA-PSS mit SHA256 bis nach Ende 2023+ verwendbar (Ende des Betrachtungshorizonts)	Der Algorithmus muss für alle qualifizierten

n	Nachrichten-Digest und die Verschlüsselung mit dem privaten Schlüssel	(Hinweis: siehe Abschnitt 4.1)	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.

754 **3.1.3 Webservice Security Standard (WSS)**

755 Nicht relevant für den Wirkbetrieb der TI.

756 **3.1.4 XML-Verschlüsselung – Symmetrisch**

757 **GS-A_4373GS-A_4373GS-A_4373 - XML-Verschlüsselung - symmetrischXML-**
758 **Verschlüsselung - symmetrischXML-Verschlüsselung - symmetrisch**

759 Alle Produkttypen, die XML-Dokumente mittels [XMLEnc-1.1] verschlüsseln, MÜSSEN die
760 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[GS-A_4374](#)[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[GS-A_4376-02](#)[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[GS-A_4379](#)[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_4380GS-A 4380GS-A 4380 - Card-to-Server (C2S) Authentisierung und Trusted Channel G2 **Card-to-Server (C2S) Authentisierung und Trusted Channel G2** **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 2024+ (meint bis Ende des Betrachtungsraums der TR) zulässig.

GS-A_4381GS-A 4381GS-A 4381 - Schlüssellängen Algorithmus **AESSchlüssellängen Algorithmus AESSchlüssellängen 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 2023+

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_4382GS-A_4382GS-A_4382 - IPsec-Kontext - SchlüsselvereinbarungIPsec-Kontext - SchlüsselvereinbarungIPsec-Kontext - Schlüsselvereinbarung

Alle Produkttypen, die die Authentifizierung, den Schlüsselaustausch und die verschlüsselte Kommunikation im IPsec-Kontext durchführen, MÜSSEN die Schlüsselvereinbarung mittels IKEv2 [RFC-7296] gemäß den folgenden Vorgaben durchführen:

- Zur Authentisierung MUSS eine Identität mit einem X.509-Zertifikat gemäß [gemSpec_Krypt#GS-A_4360] verwendet werden.
- Für „Hash und URL“ MUSS SHA-1 verwendet werden.
- Die Diffie-Hellman-Gruppe Gruppe 14 (definiert in [RFC-3526], verwendbar bis Ende 2023) MUSS für den Schlüsselaustausch unterstützt werden. Zusätzlich KÖNNEN Gruppen aus [BSI-TR-02102-3, Abschnitt 3.2.4, Tabelle 5], bei denen der Verwendungszeitraum ein „+“ enthält, verwendet werden.
- Der private DH-Exponent für den Schlüsselaustausch MUSS eine Länge von mindestens 256 Bit haben.
- Die Authentisierung der ephemeren (EC)DH-Parameter erfolgt durch eine Signatur der Parameter durch den jeweiligen Protokollteilnehmer. Bei dieser Signatur MUSS SHA-256 als Hashfunktion verwendet werden. Es SOLL die Authentisierungsmethode „Digital Signature“ nach [RFC-7427] dabei verwendet werden.
- Bei den symmetrische Verschlüsselungsalgorithmen MUSS AES mit 256 Bit Schlüssellänge im CBC-Modus unterstützt werden (sowohl für IKE-Nachrichten als auch später für die Verschlüsselung von ESP-Paketen). Es KÖNNEN weitere Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.1, Tabelle 2] bzw. [BSI-TR-02102-3, Abschnitt 3.3.1, Tabelle 7] verwendet werden.
- Für den Integritätsschutz (sowohl innerhalb von IKEv2 als auch anschließend für ESP-Pakete) MUSS HMAC mittels SHA-1 und SHA-256 (vgl. [gemSpec_Krypt#Hinweis-4382-1]) unterstützt werden. Es KÖNNEN weitere Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.3, Tabelle 4] bzw. [BSI-TR-02102-3, Abschnitt 3.3.1, Tabelle 8] verwendet werden.
- Als PRF MÜSSEN PRF_HMAC_SHA1 und PRF_HMAC_SHA2_256 (vgl. [gemSpec_Krypt#Hinweis-4382-1]) unterstützt werden. Es KÖNNEN weitere Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.2, Tabelle 3] verwendet werden.
- Schlüsselaktualisierung: die IKE-Lifetime darf maximal 24*7 Stunden betragen (Reauthentication). Die IPsec-SA-Lifetime darf maximal 24 Stunden betragen (Rekeying). Der Initiator soll nach Möglichkeit vor Ablauf der Lifetime das Rekeying anstoßen. Ansonsten muss der Responder bei Ablauf der Lifetime das Rekeying von sich aus sicherstellen, bzw. falls dies nicht möglich ist, die Verbindung beenden.
- Für die Schlüsselberechnung muss Forward Secrecy [BSI-TR-02102-1, S.ix] (in [RFC-7296] „Perfect Forward Secrecy“ genannt) gewährleistet werden. Meint die Wiederverwendung von zuvor schon verwendeten (EC-)Diffie-Hellman-Schlüsseln ([RFC-7296, Abschnitt 2.12]) ist nicht erlaubt.

[<=<=<=]

Hinweis-4382-1: In [NK-PP] wird mit FCS_COP.1/NK.HMAC und FCS_COP.1/NK.Hash die Unterstützung von SHA-1 und SHA-256 gefordert. Da für den Einsatz innerhalb einer HMAC-Funktion und innerhalb einer PRF die Einwegeneigenschaft der Hashfunktion im Vordergrund steht und nicht die allgemeine Kollisionsresistenz, ist dort der Einsatz von SHA-1 noch zulässig (vgl. auch [BSI-TR-02102-3, Abschnitt 3.2.2, Tabelle 3, 4 und 8]). Es ist davon auszugehen, dass die Zulässigkeit von SHA-1 bei diesen beiden Einsatzzwecken zukünftig nicht mehr gegeben sein kann, und sowohl im NK als auch im VPN-Zugangsdienst, bspw. per Konfiguration, deaktiviert werden muss.

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

GS-A_5547[GS-A_5547](#)[GS-A_5547](#) - gleiche VPN-IP-Adresse nach Reauthentication
[gleiche VPN-IP-Adresse nach Reauthentication](#)
[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[GS-A_5548](#)[GS-A_5548](#) - Mindestgültigkeitszeiten IKE- und IPsec-SAs
(Konnektor)[Mindestgültigkeitszeiten IKE- und IPsec-SAs](#)
(Konnektor)[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 Beispielfunktion /etc/ipsec.conf

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

GS-A_5549[GS-A_5549](#)[GS-A_5549](#) - Mindestgültigkeitszeiten IKE- und IPsec-SAs
(VPN-Zugangsdienst)[Mindestgültigkeitszeiten IKE- und IPsec-SAs](#) (VPN-
Zugangsdienst)[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[GS-A_5508](#)[GS-A_5508](#) - IPsec make_before_breakIPsec
[make_before_breakIPsec 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[GS-A_4383](#)[GS-A_4383](#) - IPsec-Kontext – Verschlüsselte
[KommunikationIPsec-Kontext – Verschlüsselte KommunikationIPsec-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[A_14652](#)[A_14652](#) - SZZP-light, asymmetrischen Schlüssel maximale
[GültigkeitsdauerSZZP-light, asymmetrischen Schlüssel maximale](#)
[GültigkeitsdauerSZZP-light, asymmetrischen Schlüssel maximale](#)
[Gültigkeitsdauer](#)

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

[<=<=<=]

3.3.2 TLS-Verbindungen

GS-A_4385[GS-A_4385](#)[GS-A_4385](#) - TLS-Verbindungen, Version 1.2[TLS-](#)
[Verbindungen, Version 1.2](#)[TLS-Verbindungen, Version 1.2](#)

Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die TLS-Version 1.2 [RFC-5246] unterstützen.

[<=<=<=]

A_18467[A_18467](#)[A_18467](#) - TLS-Verbindungen, Version 1.3[TLS-Verbindungen,](#)
[Version 1.3](#)[TLS-Verbindungen, Version 1.3](#)

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

1. dabei nur nach [BSI-TR-02102-2] empfohlene Konfigurationen (Handshake-Modi, (EC)DH-Gruppen, Signaturverfahren, Ciphersuiten etc.) verwenden, und
2. mindestens die Ciphersuite "TLS_AES_128_GCM_SHA256" dabei unterstützen.

[<=<=<=]

A_18464[A_18464](#)[A_18464](#) - TLS-Verbindungen, nicht Version 1.1[TLS-](#)
[Verbindungen, nicht Version 1.1](#)[TLS-Verbindungen, nicht Version 1.1](#)

Alle Produkttypen, die Übertragungen mittels TLS durchführen, DÜRFEN NICHT die TLS-Version 1.1 [RFC-4346] unterstützen.[<=<=<=]

GS-A_4387[GS-A_4387](#)[GS-A_4387](#) - TLS-Verbindungen, nicht Version 1.0[TLS-](#)
[Verbindungen, nicht Version 1.0](#)[TLS-Verbindungen, nicht Version 1.0](#)

Alle Produkttypen, die Übertragungen mittels TLS durchführen, DÜRFEN NICHT die TLS-Version 1.0 unterstützen.[<=<=<=]

GS-A_5035[GS-A_5035](#)[GS-A_5035](#) - Nichtverwendung des SSL-
Protokolls[Nichtverwendung des SSL-Protokolls](#)[Nichtverwendung des SSL-
Protokolls](#)

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

GS-A_4384[GS-A_4384](#)[GS-A_4384](#) - TLS-Verbindungen[TLS-Verbindungen](#)[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 MUSS TLS_DHE_RSA_WITH_AES_128_CBC_SHA oder TLS_DHE_RSA_WITH_AES_256_CBC_SHA verwendet werden.
- Es MUSS für die Schlüsselaushandlung Gruppe 14 (definiert in [RFC-3526], verwendbar bis Ende 2023) verwendet werden.
- Der private DH-Exponent für den Schlüsselaustausch MUSS eine Länge von mindestens 256 Bit haben.

[$\leq \leq \leq$]

Für Embedded-Systeme (Konnektor, eHealth-KT) ist in diesem Zusammenhang lesenswert: [Oorschot-Wiener-1996].

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

GS-A_5541[GS-A_5541](#)[GS-A_5541](#) - TLS-Verbindungen als TLS-Klient zur
Störungssampel oder SM[TLS-Verbindungen als TLS-Klient zur Störungssampel](#)
[oder SMTLS-Verbindungen als TLS-Klient zur Störungssampel oder SM](#)

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

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

und

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

[$\leq \leq \leq$]

GS-A_5580-01[GS-A_5580-01](#)[GS-A_5580-01](#) - TLS-Klient für
betriebsunterstützende Dienste[TLS-Klient für betriebsunterstützende](#)
[Dienste](#)[TLS-Klient für betriebsunterstützende Dienste](#)

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

1022 **GS-A_5581GS-A 5581GS-A 5581 - "TUC vereinfachte Zertifikatsprüfung"**
1023 **(Komponenten-PKI)"TUC vereinfachte Zertifikatsprüfung" (Komponenten-**
1024 **PKI)"TUC vereinfachte Zertifikatsprüfung" (Komponenten-PKI)**

1038 Wenn einer der Prüfschritte aus (2) bis (4) fehlschlägt, MUSS der Verbindungsaufbau
1039 abgebrochen werden.

1042 Als Hilfestellung: für die Umsetzung von GS-A_5581 Spiegelstrich (1) kann man bspw.
1043 folgende Maßnahmen wählen.

1051 Ziel ist es, dass für die Verbindung zur Störungsampel oder zum Service Monitoring auch
1052 einfach verfügbare und einfach verwendbare HTTPS-Clients wie `wget` oder `curl`
1053 verwendet werden können.

```
1061 I.  
1062 wget --ca-directory=/etc/TI-Komponenten-CAs --post-  
1063 file=SOAP_Daten https://monitoring-  
1064 update.stempel.telematik:8443/I Monitoring Message
```


II.

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

GS-A_5542GS-A 5542GS-A 5542 - TLS-Verbindungen (fatal Alert bei Abbrüchen)TLS-Verbindungen (fatal Alert bei Abbrüchen)TLS-Verbindungen (fatal Alert bei Abbrüchen)

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

[<=<=<=]

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

GS-A_5322GS-A 5322GS-A 5322 - Weitere Vorgaben für TLS-VerbindungenWeitere Vorgaben für TLS-VerbindungenWeitere Vorgaben für TLS-Verbindungen

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

- Falls der Produkttyp als *Klient* oder als *Server* im Rahmen von TLS an einer Session-Resumption mittels SessionID (vgl. [RFC-5246, Abschnitt 7.4.1.2]) teilnimmt, MUSS er sicherstellen, dass nach spätestens 24 Stunden das über den Diffie-Hellman-Schlüsselaustausch ausgehandelte Schlüsselmateriale und alles davon abgeleitete Schlüsselmateriale (vgl. [RFC-5246, Abschnitt 8.1 und 6.3]) bei ihm sicher gelöscht wird.
- Falls der Produkttyp als *Klient* im Rahmen von TLS an einer Session-Resumption nach [RFC-5077] teilnimmt, MUSS er sicherstellen, dass nach spätestens 24 Stunden das über den Diffie-Hellman-Schlüsselaustausch ausgehandelte Schlüsselmateriale und alles davon abgeleitete Schlüsselmateriale (vgl. [RFC-5246, Abschnitt 8.1 und 6.3]) bei ihm sicher gelöscht wird. Damit verbundene SessionTickets MUSS er ebenfalls sicher löschen.
- Falls der Produkttyp als *Server* im Rahmen von TLS an einer Session-Resumption nach [RFC-5077] teilnimmt, MUSS er sicherstellen, dass nach spätestens 24 Stunden das über den Diffie-Hellman-Schlüsselaustausch ausgehandelte Schlüsselmateriale und alles davon abgeleitete Schlüsselmateriale (vgl. [RFC-5246, Abschnitt 8.1 und 6.3]) bei ihm sicher gelöscht wird. Damit verbundene SessionTickets MUSS er, falls bei ihm vorhanden, sicher löschen. Das Schlüsselmateriale, dass bei der Erzeugung des SessionTickets (für die Sicherung

GS-A_5526**GS-A_5526****GS-A_5526 - TLS-Renegotiation-Indication-Extension****TLS-Renegotiation-Indication-Extension****TLS-Renegotiation-Indication-Extension**

Alle Produkttypen, die das TLS-Protokoll verwenden, SOLLEN den RFC 5746 (TLS-Renegotiation-Indication-Extension [RFC-5746]) unterstützen.

[<=<=<=]

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

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

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

[<=<=<=]

Für eine verbesserte Interoperabilität zu bestimmten TLS-Implementierungen (bspw. SChannel, vgl. auch (

https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations bzw.

<https://www.ssllabs.com/ssltest/clients.html>) sollen im Konnektor zusätzlich zu den

Ciphersuiten aus GS-A_4384 weitere Ciphersuiten unterstützt werden. Mit der mittelfristigen Anhebung des zu erreichenden Sicherheitsniveaus auf 120 Bit (vgl. [SOG-IS-2018] und [BSI-TR-03116-1]) werden die folgenden Ciphersuiten mittelfristig verpflichtend. In diesem Kontext spielt die Performanz (3000 Bit Diffie-Hellman vs. 256 Bit Elliptic Curve Diffie-Hellman) bei Embedded-Geräten wie dem Konnektor eine wichtige Rolle.

GS-A_5345**GS-A_5345****GS-A_5345 - TLS-Verbindungen Konnektor****TLS-Verbindungen Konnektor****TLS-Verbindungen Konnektor**

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

1. Der Konnektor MUSS zusätzlich folgende Ciphersuiten unterstützen:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x13),
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x14),
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x27),
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28),
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2f) und
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30).

2. Der Konnektor KANN weitere Ciphersuiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.

3. Falls Ciphersuiten aus Spiegelstrich (1) oder (2) unterstützt werden,

- a. MÜSSEN bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch die Kurven P-256 oder P-384 [FIPS-186-4] unterstützt werden,
- b. MÜSSEN die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden.

Andere Kurven SOLLEN NICHT verwendet werden.

4. Falls Ciphersuiten aus (1) oder (2) unterstützt werden, so MÜSSEN diese im CC-Zertifizierungsverfahren berücksichtigt werden.

[<=<=<=]

Von einem TLS-Server, dessen Kommunikationspartner Standard-Webbrowser sind (bspw. einem Webserver), wird wie folgt eine Webbrowser-Interoperabilität bez. der unterstützten TLS-Ciphersuiten gefordert.

GS-A_5339GS-A 5339GS-A 5339 - TLS-Verbindungen, erweiterte Webbrowser-Interoperabilität
TLS-Verbindungen, erweiterte Webbrowser-Interoperabilität

Alle Produkttypen, die TLS verwenden und bei denen insbesondere Webbrowser-Interoperabilität (Webportale, Download-Punkte o. Ä.) wichtig ist, MÜSSEN zur Absicherung der TLS-Übertragung neben der in [gemSpec_Krypt#GS-A_4384] aufgeführten Vorgaben zusätzlich Folgendes sicherstellen:

1. Der Produkttyp MUSS zusätzlich folgende Ciphersuiten unterstützen:
 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x14),
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x13),
 - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30) und
 - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F).
2. Der TLS-Server KANN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.
3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-4] unterstützt werden. Daneben KÖNNEN 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).

[<=<=<=]

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
secp256r1	[RFC-5480], http://www.secg.org/collateral/sec2_final.pdf
P-256	[FIPS186-4]

Analog P-384 [FIPS186-4]:

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	[FIPS186-4]

Der VZD wird u. Um. direkt von einem Webbrowser angesprochen, daher wird für eine größere Interoperabilität zu verschiedenen Webbrowsern von ihm die Unterstützung zusätzlicher TLS-Ciphersuiten gefordert.

GS-A_5482GS-A 5482GS-A 5482 - zusätzliche TLS-Ciphersuiten für VZD

Der VZD MUSS in Bezug auf TLS neben den in [gemSpec_Krypt#GS-A_4384] aufgeführten Ciphersuiten folgende Vorgaben umsetzen:

1. Der VZD MUSS zusätzlich folgende Ciphersuiten unterstützen:
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x13),
 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x14),
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x27),
 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28),
 - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2f)und
 - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30).
2. Der VZD KANN weitere Ciphersuiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.
3. Der VZD MUSS bei den TLS-Ciphersuiten aus Spiegelstrich (1) oder (2) bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch die Kurven P-256 oder P-384 [FIPS-186-4] unterstützen. DanebenKÖNNEN die Kurven brainpoolP256r1 und brainpoolP384r1 (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).

[<=<=<=]

A_18183A 18183A 18183 - TLS-Protokoll-Verwendung in aAdG-NetGTLS-Protokoll-Verwendung in aAdG-NetGTLS-Protokoll-Verwendung in aAdG-NetG

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

Erläuterung: Eine andere Anwendung des Gesundheitswesens ohne Zugriff auf Dienste der TI in angeschlossenen Netzen des Gesundheitswesens (aAdG-NetG) muss beim TLS-basierten Nachrichtentransport durch die TI nach [TR-02102-2] sichere Ciphersuiten und Domainparameter verwenden. Für solch eine Anwendung ist eine die Interoperabilität mit

TI-Diensten sicherstellende Einschränkung der Ciphersuiten und Domainparameter nach GS-A_4384 und A_17124 nicht notwendig, d. h. beide Anforderungen gelten nicht für solche Anwendungen, sondern A_18183 gilt.

A_18986 **A_18986A_18986A_18986 - Fachdienst-interne TLS-Verbindungen** **Fachdienst-interne TLS-Verbindungen** **Fachdienst-interne TLS-Verbindungen**

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

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

3.3.3 DNSSEC-Kontext

GS-A_4388 **GS-A_4388GS-A_4388 - DNSSEC-Kontext** **DNSSEC-Kontext** **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.

1299 3.4 Masterkey-Verfahren (informativ)

1300 Die gematik wurde aufgefordert, beispielhaft ein mögliches Ableitungsverfahren für einen
1301 versichertenindividuellen symmetrischen Schlüssel auf Grundlage eines
1302 Ableitungsschlüssels (Masterkey) aufzuführen. Ein Kartenherausgeber ist frei in der Wahl
1303 seines Ableitungsverfahrens. Jedoch müssen beim Einsatz eines Ableitungsverfahrens,
1304 um die Qualität der Ableitung zu garantieren, insbesondere folgende Punkte beachtet
1305 werden:

- 1306 • Der Ableitungsprozess muss unumkehrbar und nicht-vorhersehbar sein, um
1307 sicherzustellen, dass die Kompromittierung eines abgeleiteten Schlüssel nicht
1308 den Ableitungsschlüssel oder andere abgeleitete Schlüssel kompromittiert.
- 1309 • Bei einer Schlüsselableitung (im Sinne von [ISO-11770]) basiert die
1310 kryptographische Stärke der abgeleiteten Schlüssel auf der Ableitungsfunktion
1311 und der kryptographischen Stärke des geheimen Ableitungsschlüssels
1312 (insbesondere hier dessen Entropie). Die Entropie der abgeleiteten Schlüssel ist
1313 kleiner gleich der Entropie des geheimen Ableitungsschlüssels. Um die Entropie
1314 der abgeleiteten Schlüssel sicherzustellen, muss die Entropie des geheimen
1315 Ableitungsschlüssels (deutlich) größer sein als die zu erreichende Entropie der
1316 abgeleiteten Schlüssel.
- 1317 • Der Betreiber eines Schlüsseldienstes muss im Falle des Einsatzes einer
1318 Schlüsselableitung (nach [ISO-11770]) in seinem Sicherheitskonzept Maßnahmen
1319 für das Bekanntwerden von Schwächen des kryptographischen Verfahrens, welche
1320 die Grundlage der Schlüsselableitung ist, darlegen.

1321 Ein Kartenherausgeber hat auch die Freiheit, gar kein Ableitungsverfahren zu verwenden,
1322 sondern alle symmetrischen SK.CMS aller seiner Karten sicher in seinem RZ vorzuhalten.

1323 Ziel des Masterkey-Verfahrens zur Ableitung eines versichertenindividuellen Schlüssels ist
1324 es, aus einem geheimen Masterkey und einem öffentlichen versichertenindividuellen
1325 Merkmal einen geheimen symmetrischen Schlüssel abzuleiten, der zur Absicherung der
1326 Verbindung zwischen CMS und Smartcard verwendet wird. Öffentlich bedeutet an dieser
1327 Stelle nicht, dass die Merkmale selbst nicht schützenswert sind, es soll jedoch
1328 ausdrücken, dass die Vertraulichkeit des versichertenindividuellen Schlüssels nicht von
1329 der Geheimhaltung dieser Merkmale abhängt. Die Vertraulichkeit der Daten muss durch
1330 die Geheimhaltung des Masterkeys gewährleistet sein. Das bedeutet, die Geheimhaltung
1331 anderer Daten als des Masterkeys darf für die Vertraulichkeit der Daten nicht notwendig
1332 sein. Die Durchführung dieses Verfahrens muss bei gleichen Eingangsparametern immer
1333 das gleiche Ergebnis generieren.

1334 Für die Durchführung des Algorithmus wird neben dem Masterkey auch noch mindestens
1335 ein versichertenindividuelles Merkmal verwendet. Die Auswahl des Merkmals ist fachlich
1336 motiviert und wird daher in diesem Dokument nicht spezifiziert. Das in Tabelle 20
1337 beispielhafte Verfahren besteht aus einer Kombination von AES-Verschlüsselung [FIPS-
1338 197] und Hashwert-Bildung. Die Schlüssel- bzw. Hashwert-Länge ergibt sich gemäß
1339 Tabelle 21.

1340

1341 **Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen**
1342 **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.	HASH#1 = SHA-256(versichertenindividuelles Merkmal)
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)

1343 In der nachfolgenden Tabelle werden Kürzel entsprechend der Definition aus Abschnitt
1344 3.2.3 verwendet.

1345

1346 **Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines**
1347 **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 2023+

1348 3.5 Hybride Verschlüsselung binärer Daten

1349 Für die hybride Verschlüsselung werden die Daten zunächst symmetrisch mittels eines
1350 zufällig gewählten geheimen symmetrischen Schlüssels verschlüsselt. Der geheime
1351 Schlüssel wird im Anschluss asymmetrisch für jeden Empfänger separat verschlüsselt.

1352 *Hinweis: unter binären Daten sind im gesamten Dokument beliebige Daten insbesondere*
1353 *beliebigen Typs (Text, HTML, PDF, JPG etc.) zu verstehen. Es gilt das Prinzip: das*
1354 *Spezielle vor dem Allgemeinen: gibt es weitere spezielle Vorgaben für bestimmte*

1357 **3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer**
1358 **Daten**

3.6 Symmetrische Verschlüsselung binärer Daten

GS-A_5016GS-A_5016GS-A_5016 - Symmetrische Verschlüsselung binärer Daten Symmetrische Verschlüsselung binärer Daten Symmetrische Verschlüsselung binärer Daten

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

- 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.

[<=<=<=]

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

3.7 Signatur binärer Inhaltsdaten (Dokumente)

GS-A_5080GS-A_5080GS-A_5080 - Signaturen binärer Daten (Dokumente)Signaturen binärer Daten (Dokumente) Signaturen binärer Daten (Dokumente)

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

[<=<=<=]

Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären 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 (CAvES) [ETSI-CAvES]	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 und die Verschlüsselung mit dem privaten Schlüssel	RSASSA-PSS mit SHA256 bis nach Ende 2023+ verwendbar (Ende des Betrachtungshorizonts)	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.
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.

3.8 Signaturen innerhalb von PDF/A-Dokumenten

GS-A_5081 [GS-A_5081](#) [GS-A_5081](#) - Signaturen von PDF/A- Dokumenten [Signaturen von PDF/A-Dokumenten](#) [Signaturen von PDF/A- Dokumenten](#)

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

[<=<=<=]

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

Signaturbestandteil	Beschreibung	Algorithmus	Anmerkung
Signaturstandard	Signaturstandard	ETSI TS 102 778-3 V1.2.1, PDF Advanced Electronic Signature	Die Verwendung des Standards ist für die Signatur von PDF/A

		Profiles; Part 3: PAdES Enhanced – PAdES-BES and PAdES-EPES Profiles Technical Specification, 2010 [PAdES-3]	[PDF/A-2] Dokumenten verpflichtend, die mittels eingebetteter Signaturen signiert werden.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten Digest und die Verschlüsselung mit dem privaten Schlüssel	RSASSA-PSS mit SHA256 bis nach Ende 2023+ verwendbar (Ende des Betrachtungshorizonts)	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.
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.

1434 3.9 Kartenpersonalisierung

1435 Vgl. auch Abschnitt 2.4 (Schlüsselerzeugung).

1436 **GS-A_4391GS-A_4391GS-A_4391 - MAC im Rahmen der Personalisierung der**
1437 **eGKMAC im Rahmen der Personalisierung der eGKMAC im Rahmen der**
1438 **Personalisierung der eGK**

1439 Der Herausgeber der eGK MUSS sicherstellen, dass bei der Personalisierung der eGK die
1440 Daten bei der Übermittlung integritätsgeschützt werden. Für die Absicherung der
1441 Integrität ist in diesem Kontext der AES-256 CMAC nach [NIST-SP-800-38B] (vgl. [BSI-
1442 TR-03116-1#3.2.2, 4.5.2]) zu verwenden.

1443 Die Länge des CMAC muss 128 Bit betragen.

1444 Nach [NIST-SP-800-38B#S.13] sollen nicht mehr als 2^{48} Nachrichtenblöcke (2^{22} GByte)
1445 mit demselben Schlüssel verarbeitet werden. Nach [NIST-SP-800-38B#S.14] ist ein
1446 CMAC anfällig für Replay-Attacken, was bei der Anwendung des CMACs zu
1447 berücksichtigen ist.
1448 [$\leq \leq \leq$]

1449 3.10 Bildung der pseudonymisierten Versichertenidentität

1450 **GS-A_4392**[GS-A_4392](#)[GS-A_4392](#) - Algorithmus im Rahmen der Bildung der
1451 pseudonymisierten Versichertenidentität [Algorithmus im Rahmen der Bildung](#)
1452 [der pseudonymisierten Versichertenidentität](#) [Algorithmus im Rahmen der](#)
1453 [Bildung der pseudonymisierten Versichertenidentität](#)
1454 Alle Produkttypen, die pseudonymisierte Versichertenidentitäten berechnen, MÜSSEN den
1455 Hash-Algorithmus SHA-256 [FIPS-180-4] verwenden. [$\leq \leq \leq$]

1456 3.11 Spezielle Anwendungen von Hashfunktionen

1457 **GS-A_4393**[GS-A_4393](#)[GS-A_4393](#) - Algorithmus bei der Erstellung von
1458 Hashwerten von Zertifikaten oder öffentlichen Schlüsseln [Algorithmus bei der](#)
1459 [Erstellung von Hashwerten von Zertifikaten oder öffentlichen](#)
1460 [Schlüsseln](#) [Algorithmus bei der Erstellung von Hashwerten von Zertifikaten oder](#)
1461 [öffentlichen Schlüsseln](#)
1462 Alle Produkttypen, die Fingerprints eines öffentlichen Schlüssels oder eines Zertifikates
1463 erstellen, MÜSSEN den Hash-Algorithmus SHA-256 [FIPS-180-4] dafür
1464 verwenden. [$\leq \leq \leq$]

1465 Erläuterung:

1466 Alle CAs und der TSL-Dienst müssen im Rahmen ihrer Prozesse öffentliche Schlüssel oder
1467 Zertifikate (bspw. auf Webseiten) veröffentlichen. Dabei wird auch jeweils der SHA-256
1468 Hashwert mit veröffentlicht.

1469 Hersteller einer gSMC-KT müssen den Hashwert des auf der Karte befindlichen Zertifikats
1470 in MF/DF.KT/EF.C.SMKT.AUT.R2048 entweder auf dem ID-1-Kartenkörper drucken (das
1471 ID-000-Modul ist dann herausbrechbar) oder ausgedruckt mitliefern. Der Konnektor muss
1472 den Hashwert des Zertifikats bei initialen Pairing mit dem KT berechnen und dem
1473 Administrator präsentieren.

1474 Innerhalb der CertHash-Extension als Teil einer OCSP-Response wird vom TSP ein SHA-
1475 256 Hashwert des Zertifikats, über das eine Sperrinformation gegeben wird, mitgeliefert.

1476 **GS-A_5131**[GS-A_5131](#)[GS-A_5131](#) - Hash-Algorithmus bei OCSP/CertID [Hash-](#)
1477 [Algorithmus bei OCSP/CertID](#) [Hash-Algorithmus bei OCSP/CertID](#)

1478 Alle Produkttypen, die OCSP-Anfragen stellen oder beantworten, MÜSSEN bei der
1479 Erstellung und Verwendung der CertID-Struktur (vgl. [RFC-6960, Abschnitt 4.1.1] oder
1480 [RFC-2560, Abschnitt 4.1.1]) den Hash-Algorithmus SHA-1 [FIPS-180-4] verwenden.
1481 Ein OCSP-Server KANN auch zusätzlich andere Hashfunktionen im Rahmen der CertID,
1482 die nach [BSI-TR-03116-1] zulässig sind, unterstützen.

1483 [$\leq \leq \leq$]

3.11.1 Hashfunktionen und OCSP (informativ)

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

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

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

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

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

Warum ist der Einsatz von SHA-1 an dieser Stelle kryptographisch gesehen ausreichend? Da (1) ein OCSP-Responder der TI nicht für beliebige CAs arbeitet (Wahl von DN und öffentlichen Schlüssel ist damit beschränkt) und (2) i. d. R. die CertHash-Extension Teil der OCSP-Antwort ist und innerhalb der CertHash-Extension in der TI eine kryptographisch hochwertige Hashfunktion verwendet wird, ist die Verwendung von SHA-1 hier aus Sicherheitsicht betrachtet unbedenklich. (Vgl. analoges Vorgehen BNetzA-OCSP-Responder für den qualifizierten Vertrauensraum.) Es ist also sichergestellt, dass zwischen OCSP-Client und -Responder keine (evtl. von einem Angreifer böswillig herbeigeführten) Unklarheiten darüber entstehen können über welches Zertifikat gerade gesprochen wird. Es geht bei GS-A_5131 vornehmlich um die Interoperabilität von OCSP-Client und OCSP-Responder.

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

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

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

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

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

3.12 kryptographische Vorgaben für die SAK des Konnektors

GS-A_5071[GS-A_5071](#)[GS-A_5071](#) - kryptographische Vorgaben für eine Signaturprüfung in der SAK-Konnektorkryptographische Vorgaben für eine Signaturprüfung in der SAK-Konnektorkryptographische Vorgaben für eine Signaturprüfung in der SAK-Konnektor

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

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

[<=<=<=]

3.13 Migration im PKI-Bereich

GS-A_5079[GS-A_5079](#)[GS-A_5079](#) - Migration von Algorithmen und Schlüssellängen bei PKI-BetreibernMigration von Algorithmen und Schlüssellängen bei PKI-BetreibernMigration von Algorithmen und Schlüssellängen bei PKI-Betreibern

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

3.14 Spezielle Anwendungen von kryptographischen Signaturen

GS-A_5207[GS-A_5207](#)[GS-A_5207](#) - Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-KartenterminalSignaturverfahren beim initialen Pairing zwischen Konnektor und eHealth- KartenterminalSignaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal

Alle Produkttypen, die beim initialen Pairing zwischen Konnektor und eHealth-
Kartenterminal

1. die Signatur des Shared-Secret (ShS.AUT.KT vgl. [gemSpec_KT#2.5.2.1, 3.7.2.1]) erzeugen oder prüfen, und
2. auf Basis von RSA die TLS-Verbindung betreiben, die für das aktuell durchzuführende Pairing notwendig ist,

MÜSSEN für die Signatur des Shared-Secret und dessen Signaturprüfung RSASSA-PSS [PKCS#1] verwenden.

[<=<=<=]

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

GS-A_5208[GS-A_5208](#)**GS-A_5208 - Signaturverfahren für externe Authentisierung**[Signaturverfahren für externe Authentisierung](#)

Der Konnektor MUSS an der Schnittstelle für die externe Authentisierung die Signaturverfahren RSASSA-PKCS1-v1_5 [PKCS#1] und RSASSA-PSS [PKCS#1] anbieten.[<=<=<=]

Erläuterung: Der Konnektor erlaubt (bei entsprechender Berechtigung) die direkte Nutzung der privaten Schlüssel MF/ DF.ESIGN/ PrK.HP.AUT.* auf einem HBA oder MF/ DF.ESIGN/ PrK.HCI.AUT.* auf einer SMC-B durch ein Primärsystem. Dies wird fast immer für eine klientenseitige TLS-Authentisierung gegenüber einem TLS-Server (außerhalb der TI) verwendet. Dafür werden über die Schnittstelle RSASSA-PKCS1-v1_5-Signaturen von den entsprechenden Karten erzeugt und über den Konnektor an ein Primärsystem übergeben. Für unbenannte Anwendungen müssen auch RSASSA-PSS-Signaturen erzeugbar sein. Diese Signaturen sind nicht als Dokumentensignaturen verwendbar, der Verwendungszweck ist in den zu den privaten Schlüsseln gehörigen Zertifikaten kodiert (ExtendedKeyUsage: keyPurposeId = id-kp-clientAuth).

Hinweis: GS-A_5208 ist nicht dem PTV4-Konnektor zugewiesen, sondern die erweiterte Anforderungen A_17209.

GS-A_5340[GS-A_5340](#)**GS-A_5340 - Signatur der TSL**[Signatur der TSL](#)

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

3.15 ePA-spezifische Vorgaben

3.15.1 Verbindung zur VAU

Die "vertrauenswürdige Ausführungsumgebung" (VAU) wird in [gemSpec_Dokumentenverwaltung] eingeführt. Jedes ePA-Frontend des Versicherten (FdV) muss mit jeder beliebigen VAU (egal von welchem Anbieter ePA-Aktensystem) kommunizieren können. Deshalb ist es für die Interoperabilität notwendig, das Kommunikationsprotokoll zwischen beiden Kommunikationspartnern zu definieren und dessen Verwendung zu fordern.

A_15546A_15546A_15546 - ePA-Frontend des Versicherten: Kommunikation zwischen ePA-FdV und VAU
ePA-Frontend des Versicherten: Kommunikation zwischen ePA-FdV und VAU
ePA-Frontend des Versicherten: Kommunikation zwischen ePA-FdV und VAU

Das ePA-Frontend des Versicherten MUSS bei der Kommunikation mit der VAU das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "Kommunikationsprotokoll zwischen VAU und ePA-Clients"] verwenden und dabei die Rolle Client einnehmen. Dabei MUSS es die CipherConfiguration "AES-256-GCM-BrainpoolP256r1-SHA-256" (vgl. Abschnitt 6) verwenden. Das ePA-Frontend des Versicherten MUSS nach spätestens 24 Stunden das Aushandeln eines neuen AES-Sitzungsschlüssels erzwingen. Es MUSS den abgelaufenen Sitzungsschlüssel bei sich sicher löschen.

[<=<=<=]

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_15784) [gemSpec_ePA_FdV] muss dabei die TSL der TI Prüfungsgrundlage sein [gemSpec_ePA_FdV].

A_15549A_15549A_15549 - VAU-Client: Kommunikation zwischen VAU-Client und VAU
VAU-Client: Kommunikation zwischen VAU-Client und VAU
VAU-Client: Kommunikation zwischen VAU-Client und VAU

Ein Client einer VAU MUSS bei der Kommunikation mit der VAU das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "Kommunikationsprotokoll zwischen VAU und ePA-Clients"] verwenden. Dabei MUSS es die CipherConfiguration "AES-256-GCM-BrainpoolP256r1-SHA-256" (vgl. Abschnitt 6) verwenden. Der Client einer VAU MUSS nach spätestens 24 Stunden das Aushandeln eines neuen AES-Sitzungsschlüssels erzwingen. Er MUSS den abgelaufenen Sitzungsschlüssel bei sich sicher löschen.[<=<=<=]

A_15561A_15561A_15561 - AES-NI
AES-NI
AES-NI

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

A_15547A_15547A_15547 - VAU: Kommunikation zwischen VAU und ePA-FdV bez. FM ePA
VAU: Kommunikation zwischen VAU und ePA-FdV bez. FM ePA
VAU: Kommunikation zwischen VAU und ePA-FdV bez. FM ePA

Das ePA-Aktensystem MUSS sicherstellen, dass dessen VAU bei der Kommunikation mit dem ePA-Frontend des Versicherten oder dem FM ePA das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "Kommunikationsprotokoll zwischen VAU und ePA-Clients"] verwendet und dabei die Rolle Server einnimmt. Dabei MUSS es die CipherConfiguration "AES-256-GCM-BrainpoolP256r1-SHA-256" (vgl. Abschnitt 6) verwenden.

Die VAU MUSS nach spätestens 24 Stunden das Aushandeln eines neuen AES-Sitzungsschlüssels erzwingen. Die VAU MUSS den abgelaufenen Sitzungsschlüssel und das ephemere EC-Schlüsselpaar, das im ECDH Grundlage der Schlüsselableitung für diesen Schlüssel war, sicher löschen.

Die VAU MUSS ein Zertifikat aus der Komponenten-PKI der TI besitzen (mit Rollenkennung-OID "oid_epa_vau"), das einen ECC-EE-Schlüssel der VAU bestätigt. Die VAU MUSS für die Erstellung der VAUHelloServer-Nachricht mit dem zugehörigen privaten EE-Schlüssel signieren (Signatur der VAUHelloServerData). In der VAUHelloServer-Nachricht MUSS die VAU das Zertifikat aufführen und die dazugehörige

```
1661 OSCP-Response.  
1662 [<=<=<=]
```

1663 3.15.2 Vorgaben für ePA-Schlüssel und ePA-Chiffrate

1664 **A_15705A_15705A_15705 - Vorgaben Aktenschlüssel (RecordKey) und**
 1665 **Kontextschlüssel (ContextKey)**Vorgaben Aktenschlüssel (RecordKey) und
 1666 Kontextschlüssel (ContextKey)Vorgaben Aktenschlüssel (RecordKey) und
 1667 Kontextschlüssel (ContextKey)

1668 Ein Client eines ePA-Aktensystems MUSS sicherstellen, dass

- 1669 1. die von ihnen erzeugten Aktenschlüssel (RecordKey) und Kontextschlüssel
1670 (ContextKey) AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge sind,
- 1671 2. diese Schlüssel von ihnen ausschließlich mittels AES/GCM analog
1672 [gemSpec_Krypt#GS-A_4373] bzw. [gemSpec_Krypt#GS-A_4389] verwendet
1673 werden und
- 1674 3. sie die Arbeit mit Aktenschlüssel (RecordKey) und Kontextschlüssel (ContextKey),
1675 die nicht Spiegelstrich 1. erfüllen, ablehnen.

1676
1677 [<=<=<=]

1678 **A_18004A_18004A_18004 - Vorgaben für die Kodierung von Chiffraten**
1679 **(innerhalb von ePA)**Vorgaben für die Kodierung von Chiffraten (innerhalb von
1680 ePA)Vorgaben für die Kodierung von Chiffraten (innerhalb von ePA)

1681 Ein Client eines ePA-Aktensystems MUSS Folgendes sicherstellen.

1. Der bei der Verschlüsselung mittels AES/GCM verwendete IV MUSS immer zufällig erzeugt werden und dessen Länge MUSS stets 96 Bits (12 Byte) betragen.
2. Ein Chifftrat (base64-dekodiert) MUSS immer die Struktur:
12 Byte IV + AES-GCM-Ciphertext + 16 Byte AuthTag (ICV)
aufweisen.

1687 [<=<=<=]

1688 3.15.3 ePA-Aktensysteminterne Schlüssel

1689 **A_15745A_15745A_15745 - Verschlüsselte Speicherung der verschlüsselten**
1690 **ePA-Daten**
1691 **Verschlüsselte Speicherung der verschlüsselten ePA-**
Daten

1692 Ein ePA-Aktensystem MUSS sicherstellen, dass

- | | |
|------|--|
| 1693 | 1. es einen betreiberspezifischen Schlüssel (BS) gibt, |
| 1694 | 2. dieser Schlüssel ein AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge ist, |
| 1695 | 3. dieser Schlüssel in einem mindestens nach FIPS-140-2 Level 3 zertifizierten HSM |
| 1696 | liegt und nur dort verwendet wird, |
| 1697 | 4. dieser Schlüssel im Betrieb ausschließlich der VAU des entsprechenden ePA- |
| 1698 | Aktensystem zugänglich ist, |
| 1699 | 5. dieser Schlüssel nur zur Schlüsselableitung nach einem in |
| 1700 | [gemSpec_Krypt#Abschnitt 2.4] zulässigen Verfahren verwendet wird, |

6. es eine Schlüsselableitung mit diesem betreiberspezifischen Schlüssel und einem aktenspezifischen Merkmal (bspw. der KVNR) gibt und daraus ein aktenspezifischer Schlüssel (ABS) abgeleitet wird,
7. dieser aktenspezifische Schlüssel ein AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge ist,
8. die verschlüsselten ePA-Daten einer Akte mit diesem aktenspezifischen Schlüssel verschlüsselt werden,
9. die verschlüsselten ePA-Daten außerhalb der VAU niemals im Klartext (also ohne mittels des ABS verschlüsselt zu sein) liegen,
10. dieser Schlüssel (ABS) ausschließlich mittels AES/GCM analog [gemSpec_Krypt#GS-A_4389] verwendet wird (der ABS wird durch Anfrage der VAU im HSM berechnet (Schlüsselableitung) und dann von dort an die VAU übermittelt, die AES/GCM-Operationen mit dem ABS finden in der VAU statt),
11. dieser Schlüssel (ABS) im Betrieb ausschließlich der VAU des entsprechenden ePA-Aktensystem zugänglich ist.

[<=<=<=]

Erläuterung: Das zu erreichende Ziel ist, dass, wenn ein Angreifer (mit hohem Angriffspotential, im Sinne von CC) selbst unter (1) der (hypothetischen) Annahme, die ePA an sich wäre überhaupt nicht verschlüsselt (es würde gar kein Aktenschlüssel existieren etc.) und (2), er alles im ePA-Aktensystem außer der VAU (inkl. HSM mit dem betreiberspezifischen Schlüssel) sicherheitstechnisch kompromittiert hätte, der Angreifer immer noch nicht auf die Klartextdaten zugreifen könnte.

Hintergrund ist, dass es unter dem Hybrid-Modell der ePA-Architektur aus Zugriffskontrolle und Verschlüsselung bei Entzug einer Berechtigung einem nun nicht mehr berechtigten Nutzer kryptographisch theoretisch immer noch möglich ist, die Daten der Akten, auf die er vormals berechtigten Zugriff hatte, zu entschlüsseln (er bricht bspw. in das Rechenzentrum des Anbieters ePA-Aktensystem ein und stiehlt dort alle Festplatten). Durch den in einem HSM beschützten betreiberspezifischen Schlüssel ist dieser beschriebene Angriff nun unterbunden.

A_15746A 15746A 15746 - Sicherstellung der Verfügbarkeit des betreiberspezifischen Schlüssels

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

[<=<=<=]

A_16176A 16176A 16176 - Mindestvorgaben für ePA-Aktensystem-interne Schlüssel

Ein ePA-Aktensystem MUSS bei innerhalb des Aktensystems eingesetzten Schlüsselmaterial, das nicht aus der TI-PKI kommt (Signatur Autorisierungstoken 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-186-4],
 - b. brainpoolP256r1, brainpoolP384r1 oder brainpoolP512r1 [RFC-5639].

[<=<=<=]

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

A_20519A_20519A_20519 - Wechsel des betreiberspezifischen Schlüssels

Ein ePA-Aktensystem MUSS sicherstellen, dass

1. der betreiberspezifische Schlüssel (BS) (vgl. A_15745) mindestens alle 5 Jahre gewechselt wird,
2. nach Erzeugung eines neuen BS alle auf Grundlage des alten BS erzeugten Chiffre umgeschlüsselt werden (neue Schlüsselableitung der aktenspezifischen Schlüssel (ABS) A_15745#6, Umschlüsselung der chiffrierten Aktendaten),
3. anschließend die alten Chiffre und der alte BS sicher gelöscht werden, und
4. die Schritte 2. und 3. spätestens 4 Wochen nach Schlüsselwechsel des BS abgeschlossen sind.

[<=<=<=]

Hinweis: Der betreiberspezifische Schlüssel (BS) darf gemäß A_15745 ausschließlich der VAU des ePA-Aktensystems zugänglich sein. Daher muss die Umschlüsselung in der VAU stattfinden. Dabei ist *keine* Mitwirkung des Versicherten (oder eines von ihm berechtigten Nutzers) erforderlich.

3.15.4 ePA-spezifische TLS-Vorgaben

A_15751A_15751A_15751 - TLS-Verbindung zwischen ePA-Aktensystem und ePA-FdV

Ein ePA-Aktensystem und ein ePA-Frontend des Versicherten 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).

- 1786 2. Sie KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1]
1787 unterstützen.
- 1788 3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch und bei der
1789 Signaturprüfung mittels ECDSA MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-
1790 4] unterstützt werden. Daneben SOLLEN die Kurven brainpoolP256r1,
1791 brainpoolP384r1 oder brainpoolP512r1 (vgl. [RFC-5639] und [RFC-7027])
1792 unterstützt werden. Andere Kurven SOLLEN NICHT verwendet werden (Hinweis:
1793 die Intention des letzten Satzes ist insbesondere, dass die Ordnung des
1794 Basispunktes in $E(F_p)$ nicht zu klein werden darf).

1795 [$\leq \leq \leq$]

1796 **A_15833A_15833A_15833 - TLS-Verbindungen ePA-FdV**
1797 **TLS-Verbindungen ePA-FdV**

1798 Ein ePA-Frontend des Versicherten MUSS die TLS-Vorgaben in A_15751 bei allen seinen
1799 TLS-Verbindungen einhalten.

1800 [$\leq \leq \leq$]

1801 3.15.5 Schlüsselableitungsfunktionalität ePA

1802 Zur Schlüsselableitung bei der Schlüsselableitungsfunktionalität ePA wird die HKDF nach
1803 [RFC-5869] auf Basis von SHA-256 verwendet. Diese Funktion wird auch als Grundlage
1804 der Schlüsselableitungen bei TLS Version 1.3 verwendet.

1805 **A_17876A_17876A_17876 - SGD: Schlüsselableitung der spezifischen**
1806 **Schlüssel**
1807 **SGD: Schlüsselableitung der spezifischen Schlüssel**
1808 **Schlüsselableitung der spezifischen Schlüssel**

1808 Ein SGD ePA MUSS folgende Vorgaben durchsetzen:

- 1809 1. Als Ableitungsverfahren für die Schlüsselableitung der versichertenindividuellen
1810 Schlüssel MUSS das HKDF nach [RFC-5869] auf Basis von SHA-256 verwendet
1811 werden.
- 1812 2. Die Ableitungsschlüssel MÜSSEN eine Mindestentropie von 512 Bit besitzen.

1813 [$\leq \leq \leq$]

1814 Ein Client eines SGD (bspw. ein ePA-FdV) erhält über einen beidseitig authentisierten
1815 Ende-zu-Ende-verschlüsselten Kanal von jeweils zwei unabhängigen SGD AES-256-Bit-
1816 Schlüssel. Diese beiden Schlüssel nutzt der Client, um den Akten- und Kontextschlüssel
1817 des Versicherten im "Zwiebelschalenprinzip" zu ver- oder zu entschlüsseln.

1818 **A_17872A_17872A_17872 - Ver- und Entschlüsselung der Akten und**
1819 **Kontextschlüssel (Schlüsselableitungsfunktionalität ePA)**
1820 **Ver- und Entschlüsselung der Akten und Kontextschlüssel**
1821 **(Schlüsselableitungsfunktionalität ePA)**
1822 **Ver- und Entschlüsselung der Akten und Kontextschlüssel (Schlüsselableitungsfunktionalität ePA)**

1823 Ein Client eines SGD ePA MUSS bei der Ver- und Entschlüsselung der Akten- und
1824 Kontextschlüssel im Kontext Schlüsselableitungsfunktionalität ePA folgende Vorgaben
1825 umsetzen.

- 1826 1. Als symmetrische Block-Chiffre MUSS AES [FIPS-197] mit einer Schlüssellänge
1827 von 256 Bit im Galois/Counter Mode (GCM) gemäß [NIST-SP-800-38D] mit der
1828 Tag-Länge von 128 Bit verwendet werden.

1831 3. Der IV MUSS eine Bitlänge von 96 Bit (12 Byte) besitzen.

1833 Für die Ende-zu-Ende-verschlüsselte Datenübertragung zwischen Client und SGD-HSM
1834 wird ECIES (vgl. [SEC1-2009#5.1 Elliptic Curve Integrated Encryption Scheme], [TR-
1835 02102.1#3.3. ECIES-Verschlüsselungsverfahren] und Abschnitt 5.7- ECIES) verwendet.
1836 Dabei besitzt der Empfänger einen elliptischen Kurvenpunkt (öffentlicher Schlüssel),
1837 dessen Authentizität der Sender prüfen kann. Dies wird erreicht, indem der Kurvenpunkt
1838 des Empfängers (entweder Client oder SGD-HSM) mittels der Langzeitidentität des
1839 Empfängers signiert ist. Der Sender erzeugt ein ephemeres ECDH-Schlüsselpaar. Mit
1840 diesem und dem Kurvenpunkt des Empfängers führt der Sender einen ECDH-
1841 Schlüsselaustausch durch. Aus dem berechneten ECDH-Geheimnis berechnet der Sender
1842 mittels einer HKDF auf Basis von SHA-256 einen AES-256-Bit-Schlüssel der im
1843 Galois/Counter-Mode (GCM) verwendet wird (Authenticated Encryption). Damit
1844 verschlüsselt der Sender den Klartext und erhält ein AES-GCM-Chifftrat. Der Sender
1845 sendet seinen erzeugten ephemeren Kurvenpunkt und das AES-GCM-Chifftrat an den
1846 Empfänger. Damit ist der Nachrichtentransport Ende-zu-Ende-verschlüsselt zwischen
1847 Sender und Empfänger, jedoch nur einseitig authentisiert. Die beidseitige Authentisierung
1848 wird über einen Authentisierungstoken, den das SGD-HSM für einen Client erzeugt,
1849 erreicht (vgl. [gemSpec_SGD_ePA#[Datenkanal zwischen Client und SGD \(informativ\)](#)]).
1850 Für das ECIES-Verfahren gilt der kryptographische Sicherheitsbeweis aus [ABR-1999].

Ein SGD ePA MUSS sicherstellen, dass die zwei Schlüsselpaare (vgl.
[gemSpec_SGD_ePA#A 17910 (S4)]) für den ECIES-Nachrichtempfang durch das
SGD-HSM auf Basis der Kurvenparameter brainpoolP256r1 [RFC-5639] gewählt werden.
Für die Authentisierung der öffentlichen ECIES-Schlüssels (Signatur mit
[gemSpec_SGD_ePA#A 17910 (S1)]) und Kodierung nach
[gemSpec_SGD_ePA#A 17894]) MUSS ECDSA [BSI-TR-03111] verwendet werden.

Ein Client eines SGD ePA MUSS für den Nachrichtenempfang mittels des ECIES-Verfahrens im Kontext der Schlüsselableitungsfunktionalität ePA bei den verwendeten ECC-Schlüsseln die Kurvenparameter brainpoolP256r1 [RFC-5639] verwenden. Für die Authentisierung des öffentlichen ECIES-Schlüssels des Clients (Signatur mit AUT-Identität des Nutzers des Clients gemäß [gemSpec_SGD_ePA#A 17901]) MUSS ECDSA [BSI-TR-03111] verwendet werden.

1871 In A_17873 und in A_17874 wird nicht aufgeführt, welche Hashfunktion im Rahmen der
1872 Signaturstellung und -prüfung zu verwenden ist. Dies wird mit folgender Anforderung
1873 nachgeholt.

**A_19971A_19971A_19971 - SGD und SGD-Client, Hashfunktion für
Signaturerstellung und -prüfungSGD und SGD-Client, Hashfunktion für
Signaturerstellung und -prüfungSGD und SGD-Client, Hashfunktion für
Signaturerstellung und -prüfung**

Ein SGD ePA und ein Client eines SGD ePA MÜSSEN bei der Signaturerstellung und -
prüfung im Kontext:

1. Signaturerstellung mit [gemSpec_SGD_ePA#A_17910 (S1)] und Kodierung nach
[gemSpec_SGD_ePA#A_17894] (vgl. A_17873), bzw. Prüfung dieser Signatur,
und
2. Signaturerstellung für die Authentisierung der öffentlichen ECIES-Schlüssels des
Clients (vgl. A_17874), bzw. Prüfung dieser Signatur

die Hashfunktion SHA-256 [FIPS-180-4] verwenden. [<= <= <=]

**A_17875A_17875A_17875 - ECIES-verschlüsselter Nachrichtenversand
zwischen SGD-Client und SGD-HSMECIES-verschlüsselter Nachrichtenversand
zwischen SGD-Client und SGD-HSM**

Ein SGD ePA und ein Client eines SGD ePA MÜSSEN folgende Vorgaben umsetzen.

1. Für den Ende-zu-Ende-verschlüsselten Datenaustausch zwischen SGD-Client und
SGD-HSM MUSS das ECIES-Verfahren [SEC1-2009] verwendet werden.
2. Der ECDH-Schlüsselaustausch innerhalb von ECIES zwischen SGD-Client und
SGD-HSM MUSS nach [NIST-800-56-A#5.7.1.2] (Hinweis: ist fachlich identisch zu
[SEC1-2009#3.3.1]) durchgeführt werden.
3. Aus dem gemeinsamen ECDH-Geheimnis MUSS mit der HKDF nach [RFC-
5869] auf Basis von SHA-256 ein AES-256-Bit-Schlüssel abgeleitet werden.
4. Dieser Schlüssel (siehe Punkt 3) MUSS mittels AES-GCM und den fachlichen
Vorgaben für AES-GCM aus A_17872 verwendet werden, um den symmetrisch
Teil der ECIES-Verschlüsselung authentisiert zu ver- bzw. zu entschlüsseln.

Hinweis: die Kodierung der Chiffre wird in [gemSpec_SGD_ePA] festgelegt.

[<= <= <=]

**A_18023A_18023A_18023 - SGD, Ableitungsschlüssel
AuthentisierungstokenSGD, Ableitungsschlüssel AuthentisierungstokenSGD,
Ableitungsschlüssel Authentisierungstoken**

Ein SGD ePA MUSS folgende Vorgaben umsetzen.

1. Die Ableitungsschlüssel für die Erstellung der Authentisierungstoken
[gemSpec_SGD_ePA#A_17910 (S5)] MÜSSEN eine Mindestentropie von 256 Bit
besitzen.
2. Diese Ableitungsschlüssel MÜSSEN mit der HKDF nach [RFC-5869] auf Basis von
SHA-256 verwendet werden.

[<= <= <=]

3.16 E-Rezept-spezifische Vorgaben (informativ)

Der Fachdienst E-Rezept besitzt zwei HTTPS-Schnittstellen, eine in der TI und eine im
Internet. Die Vorgaben für das dabei verwendete TLS-Protokoll befinden sich in

- 1987 • Für die symmetrische Verschlüsselung mittels AES-GCM ([FIPS-197], [NIST-SP-
1988 800-38D]) gibt es keine Algorithmen-Identifizierung innerhalb von [XMLEnc]. Solche
1989 gibt es in [XMLEnc-1.1, Abschnitt 5.2.4].
- 1990 • Für die Kodierung von RSA-OAEP-Chiffren innerhalb von [XMLEnc] fehlt in
1991 [XMLEnc] ein Identifier für RSAES-OAEP mit der MGF1 basierend auf SHA-256
1992 (vgl. auch Kapitel 5.10 „MGF Mask Generation Function“ in [gemSpec_COS]).
1993 Einen solchen Identifier („<http://www.w3.org/2009/xmlenc11#mgf1sha256>“) gibt
1994 es in XMLEnc Version 1.1 [XMLEnc-1.1, Abschnitt 5.5.2].
- 1995 Aus diesem Grund hat die gematik entschieden für die XML-Verschlüsselung die
1996 Vorgaben aus [XMLEnc-1.1] zu verwenden.

1997 **4.3 XML Signature Wrapping und XML Encryption Wrapping**

- 1998 Komplexität ist der natürliche Feind von Sicherheit. Die unter dem Sammelbegriff XML
1999 betitelten Formate und Protokolle sind sehr flexibel und leistungsfähig, aber auch sehr
2000 komplex. Noch dazu sind Sicherheitsmechanismen in diesem Bereich zum Teil
2001 nachträglich beigelegt worden und sind damit oft weniger leistungsfähig als im CMS-
2002 Bereich. XML-Daten effektiv zu schützen ist aktives Forschungsthema [XMLEnc-CM],
2003 [XSpRES]. Öfter als in anderen Bereichen werden neue Schwachstellen bekannt
2004 [BreakingXMLEnc], [XSW-Attack].
- 2005 Aus diesem Grunde wird bei einer Sicherheitsevaluierung gesondert auf derartige Angriffe
2006 geachtet. Die gematik beobachtet neue Entwicklungen im Bereich der XML-Sicherheit und
2007 leitet falls notwendig Maßnahmen ein.

2008 **4.4 Güte von Zufallszahlen**

- 2009 Nach dem Kerckhoffs'schen Prinzip von 1883 [Ker-1883] darf die Sicherungsleistung von
2010 kryptographischen Verfahren allein auf der Geheimhaltung der geheimen oder privaten
2011 Schlüssel beruhen. Geheimhaltung inkludiert insbesondere, dass sie nicht erraten werden
2012 können. Wenn bei einer Schlüsselerzeugung zu wenig Entropie vorhanden ist, kann die
2013 Geheimhaltung nicht gewährleistet werden. Die kryptographischen Verfahren, welche mit
2014 diesen Schlüsseln dann arbeiten, können die von ihnen verlangten Sicherheitsleistungen
2015 nicht mehr erbringen. Aus diesem Grunde verlangt [BSI-TR-03116-1] eine Mindestgüte
2016 der Zufallszahlenerzeugung u. a. bei einer Schlüsselerzeugung. Die Basis für die
2017 Beurteilung der Güte stellt [AIS-20] und [AIS-31] dar.
- 2018 Aktuell sind nicht alle Produkte in der TI bez. dieser Mindestgüte bewertet worden. Davon
2019 sind Smartcards nicht betroffen, da diese eine Sicherheitsevaluierung/-zertifizierung
2020 durchlaufen haben, bei der die Güte der Zufallszahlenerzeugung positiv beurteilt wurde.
2021 Probleme bereiten insbesondere HSMs.
- 2022 Neben einer möglichen Common-Criteria-Zertifizierung dieser Produkte, bei der analog zu
2023 den Smartcards die Güte geprüft wird, gibt es weitere mögliche Lösungen:
- 2024 1. gesonderte Prüfung der Güte nach [AIS-20] und [AIS-31] ohne komplette
2025 Common-Criteria-Zertifizierung,
- 2026 2. Herstellererklärung über die Güte (wie sie bspw. aktuell bei der Kartenproduktion
2027 üblich ist).

2028

5 Migration 120-Bit-Sicherheitsniveau

2029 Das „Sicherheitsniveau eines kryptographischen Verfahrens“ ist definiert als der
2030 Logarithmus zur Basis 2 der Anzahl der „Rechenschritte“ die notwendig sind um ein
2031 kryptographisches Verfahren mit hoher Wahrscheinlichkeit zu brechen. Was als
2032 „Rechenschritt“ definiert ist, ist vom Verfahren abhängig. Das Sicherheitsniveau wird in
2033 Bit angegeben. Beispielsweise nimmt man aktuell an, dass für das Brechen einer AES-
2034 Chiffre mit 128 Bit Schlüssellänge rund $2^{126,4}$ Rechenschritte, die der Durchführung einer
2035 AES-Verschlüsselung (eines 128-Bit Eingabeblocks) entsprechen, im Mittel notwendig
2036 sind. Somit erreicht eine AES-128-Bit-Verschlüsselung maximal ein Sicherheitsniveau von
2037 ca. 126,4 Bit. Eine RSA-2048-Bit-Verschlüsselung erreicht ein Sicherheitsniveau von ca.
2038 100 Bit.

2039 Für den qualifizierten Vertrauensraum ist ab Ende 2024 [SOG-IS-2018] und für die TI ab
2040 Ende 2023 ein Sicherheitsniveau von mindestens 120 Bit für alle kryptographischen
2041 Verfahren vorgeschrieben [BSI-TR-03116-1]. Daher ist bis dahin eine Migration aller
2042 Komponenten und Dienste notwendig, die kryptographische Verfahren mit
2043 Schlüssellängen bez. Domainparametern verwenden die nur ein Sicherheitsniveau von
2044 unter 120 Bit erreichen können.

2045 Aufgrund der höheren Performanz, insbesondere in Chipkarten und Embedded-Geräten,
2046 wird nicht auf RSA-3072-Bit sondern auf ECDSA mit 256-Bit-Schlüsseln migriert.

2047 Es gibt Produkttypen, die kryptographische Verfahren so einsetzen, dass diese keine
2048 direkten Wechselwirkungen bei anderen Produkttypen besitzen. Beispielsweise werden
2049 von einem ePA-Aktensystem Autorisierungstoken (inkl. Signatur) erzeugt und diese
2050 werden von einem ePA-FdV oder als FM ePA als opakes Objekt behandelt. Dabei kann
2051 weiterhin RSA verwendet werden, solange die dabei verwendeten Schlüsselgrößen
2052 mindestens 3000 Bit betragen (Sicherheitsniveau 120-Bit erzielen) (A_16176). Ggf. ist
2053 es empfehlenswert dennoch auf ECC-basierte Verfahren zu migrieren (schnellere
2054 Ausführungsgeschwindigkeit, geringere Signaturgröße).

2055 Die Migration erfolgt schrittweise und Komponenten und Dienste werden zusätzlich mit
2056 Schlüsselmateriale und Zertifikaten auf Basis von ECDSA auf der Kurve brainpoolP256r1
2057 ausgestattet werden. Es gibt bis maximal Ende 2023 (vgl. Abschnitt 2.1.1.1) einen
2058 Parallelbetrieb in der TI.

2059 Nachdem die X.509-Root der TI (Produkttyp „gematik Root-CA“), die TSPs der TI und die
2060 Objektsysteme der Chipkarten um ECC-Unterstützung für X.509-Identitäten erweitert
2061 wurden, erfolgt die schrittweise und parallele Unterstützung dieser Identitäten nun in
2062 weiteren Produkttypen bzw. Fachanwendungen.

2063 5.1 PKI-Begriff Schlüsselgeneration

2064 In [gemKPT_PKI_TIP#3.2] wird der Begriff der Schlüsselgeneration eingeführt. Eine CA
2065 signiert Zertifikate im abstrahierten Sinne mit „ihrem Signaturschlüssel“. Dieser Schlüssel
2066 wird regelmäßig neu erzeugt und solange Verfahren und Schlüssellänge bzw.
2067 Domainparameter gleichbleiben, handelt es sich um eine neue Schlüsselversion.
2068 Kryptographisch betrachtet wurde der neue Signaturschlüssel zufällig (vgl. GS-A_4368)
2069 erzeugt, ist also kryptographisch unabhängig vom alten Signaturschlüssel, und die CA
2070 arbeitet mit mehreren kryptographischen Schlüsseln.

2071 Beispiel: im Fall der X.509-Root der TI (vgl. Abschnitt 5.2) wird ihr Signaturschlüssel im
2072 Regelfall alle zwei Jahre neu erzeugt (vgl. GEM.RCA1 und GEM.RCA2,
2073 <https://download.tsl.ti-dienste.de/>). Der Signaturschlüssel liegt hier in zwei Versionen
2074 vor. Beide Schlüssel kommen aus der Schlüsselgeneration „RSA“.
2075 Für die Migration muss ein Signaturschlüssel in der X.509-Root der TI erzeugt werden,
2076 der aus der Schlüsselgeneration „ECDSA“ stammt. Für ihn gelten die Vorgaben aus
2077 [gemSpec_Krypt#GS-A_4357, Schlüsselgeneration „ECDSA“].

2078 5.2 X.509-Root der TI

2079 Die X.509-Root der TI (Produkttyp: gematik Root-CA) ermöglicht es über eine klassische
2080 PKI-Baumstruktur die meisten Zertifikate der TI zu prüfen. Für zukünftige Anwendungen,
2081 die nur mit erhöhten Kosten das leistungsstarke, aber auch deutlich komplexere TSL-
2082 Modell auswerten können, ist sie eine Infrastrukturleistung der TI, so wie auch die CVC-
2083 Root.

2084 Die X.509-Root muss für die Migration ECDSA-basierte Zertifikate für TSPs ausstellen
2085 können. Aufgrund von [gemSpec_PKI#GS-A_5511] muss die X.509-Root der TI neben
2086 dem Signaturschlüssel für die Schlüsselgeneration „RSA“ auch einen Signaturschlüssel für
2087 die Schlüsselgeneration „ECDSA“ gemäß GS-A_4357 (brainpoolP256r1) erzeugen, und
2088 diesen verwenden können.

2089 Als Hilfestellung wird im Folgenden ein X.509-Root-TI-Zertifikat betrachtet. Gemäß GS-
2090 A_4357 muss der öffentliche ECDSA-Schlüssel der Schlüsselgeneration „ECDSA“ auf der
2091 Kurve brainpoolP256r1 liegen. Sei

2092 `d=SHA-256 („gemSpec_Krypt-Beispiel X.509-Root-TI ECDSA-Schlüssel“)`
2093 `=0x62e50dca4da29b0b10ead635a20b51fb1ec281d11f90cde8b5a9d92371ae8052`

2094 Dieses `d` wird als Ganzzahl (Little-Endian) interpretiert und dies sei der für das Beispiel
2095 maßgebliche private Schlüssel. Damit ergibt sich folgender öffentlicher Punkt auf der
2096 Kurve brainpoolP256r1:

2097 `(0x377434509adcb827f74acd7adf0ce72aa28ddc53be3f15ea8023a9b0722c09d,`
2098 `0x5364a99686c02092bbf9efde9878847b90f09d90b7ac4193553820258a58dfd5)`

2099 Folgend ist die ASN.1-DER-Kodierung des Schlüssels, so wie sie sich später auch im
2100 Zertifikat befindet, aufgeführt:

2101 `MFowFAYHKoZIzj0CAQYJKyQDAwIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgCOPsH`
2102 `IsCdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39U=`

```
2103
2104     0  90: SEQUENCE {
2105     2  20: SEQUENCE {
2106     4   7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
2107    13   9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2108     :   }
2109    24  66: BIT STRING
2110     :    04 37 74 34 50 9A DC BB 82 7F 74 AC D7 AD F0 CE
2111     :    72 AA 28 DD C5 3B E3 F1 5E A8 02 3A 9B 07 22 C0
2112     :    9D 53 64 A9 96 86 C0 20 92 BB F9 EF DE 98 78 84
2113     :    7B 90 F0 9D 90 B7 AC 41 93 55 38 20 25 8A 58 DF
2114     :    D5
2115     :   }
```

2116 Das selbstsignierte Beispiel-Root-Zertifikat im PEM-Format:

```
2117 -----BEGIN CERTIFICATE-----
2118 MIICajCCAg+gAwIBAgIBATAKBggqhkJOPQDDAjBtMQswCQYDVQQGEwJERTEVMBMG
2119 A1UECgwMZ2VtYXRpayBHbWJIMTQwMgYDVQQGLDctaZW50cmFsZSBSb290LUNBIGNl
2120 ciBUZWNxbWw0aWtpbmZyYXN0cnVrdHVyMREwDwYDVQQDDAhHRU0uUkNBMAEwOx
2121 NjEyMDkwODQxNTZaFw0yNjEyMDcwODQxNTZaMG0xCzAJBgNVBAYTAkRFRUwEwYD
2122 VQKDAxNzW1hdGlrIEdtYkgxNDAYBgNVBAsMK1plbnRyYWx1IFJvb3QtQ0EgZGVy
2123 IFRlbGVtYXRpa2luZnJhc3RydWt0dXlxEtAPBgNVBAMCEdFTS5SQ0EzMfowFAYH
2124 KoZIZj0CAQYJKyQDAWIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgC
2125 OpsHIScdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39WjgZ4wgZswHQYD
2126 VR0OBBYEFBERSneTkjZDKt3uLzjddI870TMMMEIGCCSGAQUBwEBBDYwNDAYBggr
2127 BgEFBQcwAYYmaHR0cDovL29jc3Aucm9vdC1jYS50aS1kaWVuc3RlLmRlL29jc3Aw
2128 DwYDR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwFQYDVR0gBA4wDDAKBgqg
2129 ghQATASBIzAKBgqhkJOPQDDAgNjADBGAiEApQ6qGHTx97IsdzoWH9/W32yt4rk
2130 udUis0xxGZ48YOUCIQCTQ4puo15YyIAZYk74mfid3JBovMBV/XgPV2WpS/99yg==
2131 -----END CERTIFICATE-----
```

2132 Relativ am Anfang des Zertifikats befindet sich die OID gemäß GS-A_4357

```
2133 16 10: SEQUENCE {
2134 18 8: OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
2135 : }
```

2136 Ab Offset 280 befindet sich der schon o. g. öffentlicher Schlüssel:

```
2137 282 90: SEQUENCE {
2138 284 20: SEQUENCE {
2139 286 7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
2140 295 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2141 : }
2142 306 66: BIT STRING
2143 : 04 37 74 34 50 9A DC BB 82 7F 74 AC D7 AD F0 CE
2144 : 72 AA 28 DD C5 3B E3 F1 5E A8 02 3A 9B 07 22 C0
2145 : 9D 53 64 A9 96 86 C0 20 92 BB F9 EF DE 98 78 84
2146 : 7B 90 F0 9D 90 B7 AC 41 93 55 38 20 25 8A 58 DF
2147 : D5
2148 : }
```

2149 Und am Ende des Zertifikats befindet sich die ECDSA-Signatur:

```
2150 535 10: SEQUENCE {
2151 537 8: OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
2152 : }
2153 547 73: BIT STRING, encapsulates {
2154 550 70: SEQUENCE {
2155 552 33: INTEGER
2156 : 00 A5 0E AA 18 74 F1 F7 B2 2C 77 38 28 58 7F 7F
2157 : 5B 7D B2 B7 8A E4 B9 D5 22 B3 4C 71 19 9E 3C 60
2158 : E5
2159 587 33: INTEGER
2160 : 00 93 43 8A 6E A2 5E 58 60 80 19 62 4E F8 99 F8
2161 : 9D DC 90 4E BC C0 55 FD 78 0F 57 65 A9 4B FF 7D
2162 : CA
2163 : }
2164 : }
2165 : }
```

2166 Wenn das oben aufgeführte Zertifikat sich in der Datei "root.pem" befindet, so kann man
2167 bspw. mittels

```
2168 openssl verify -check_ss_sig root.pem
```

2169 die Signatur überprüfen und erhält als Ausgabe:

```
2170 root.pem: C = DE, O = gematik GmbH, OU = Zentrale Root-CA der
2171 Telematikinfrastruktur, CN = GEM.RCA3
```

2174 5.3 TSL-Dienst und ECDSA-basierte TSL allgemein

A_17322A_17322A_17322 - TLS-Verbindungen nur zulässige Ciphersuiten und TLS-Versionen (ECC-Migration)
TLS-Verbindungen nur zulässige Ciphersuiten und TLS-Versionen (ECC-Migration)
TLS-Verbindungen nur zulässige Ciphersuiten und TLS-Versionen (ECC-Migration)

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

[<=<=<=]

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

5.5 ECC-Unterstützung bei IPsec

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

Initiator	Responder

HDR, SA _i l, KE _i , Ni -->	

[RFC-7296]. In KE_i ist der ephemere ECDH-Schlüssel auf Grundlage der Domainparameter brainpoolP256r1 enthalten. Falls der Responder diese Domainparameter (ECC-Kurve) nicht unterstützt, antwortet der Responder mit einer INVALID_KEY_PAYLOAD-Nachricht, in der eine vom Responder unterstützte und präferierte kryptographische Gruppe angegeben ist [RFC-7296#Abschnitt 1.2]. Somit kommt es bei einem initialen Verbindungsaufbau zwischen einem "neuen" Konnektor und einem "alten" VPN-Zugangsdienst zu einem zusätzlichen "roundtrip", was akzeptiert wird, weil dies die Schlüsselaushandlung und damit den folgenden Verbindungsfall im Normalfall nur unwesentlich verzögert. Ein "neuer" Konnektor, der ggf. solch eine INVALID_KEY_PAYLOAD-Nachricht erhält, wird dann auf die Vorgaben GS-A_4382 "zurückfallen". Da davon auszugehen ist, dass alle VPN-Zugangsdienste rein technisch die mit A_17125 geforderten Algorithmen schon beherrschen, wird es wahrscheinlich in der Praxis selten ein "Zurückfallen" geben.

A_17210A_17210A_17210 - Konnektor, IKE-Schlüsselaushandlung Fallback (ECC-Migration)
Konnektor, IKE-Schlüsselaushandlung Fallback (ECC-Migration)
Konnektor, IKE-Schlüsselaushandlung Fallback (ECC-Migration)

Ein Konnektor MUSS, falls beim IKE-Verbindungsaufbau klar wird, dass der IKE-Responder (VPN-Konzentrator, VPN-Zugangsdienst) (noch) keine ECC-Verfahren unterstützt (INVALID_KEY_PAYLOAD-Nachricht), auf die Vorgaben aus GS-A_4382

2306 "zurückfallen".

2307 [$\leq \leq \leq$]

2308 Analog zum TLS-Protokoll wählt der Responder die CipherSuite und ein "alter" Konnektor
2309 kann nicht erkennen, dass es sich evtl. um einen "neuen" VPN-Zugangsdienst handelt.

2310 **A_17125A 17125A 17125 - IKE-Schlüsselaushandlung für IPsec (ECC-**
2311 **Migration)****IKE-Schlüsselaushandlung für IPsec (ECC-Migration)****IKE-**
2312 **Schlüsselaushandlung für IPsec (ECC-Migration)**

2313 Alle Produkttypen, die die Authentifizierung, den Schlüsselaustausch und die
2314 verschlüsselte Kommunikation im IPsec-Kontext durchführen, MÜSSEN die
2315 Schlüsselvereinbarung mittels IKEv2 [RFC-7296] gemäß den folgenden Vorgaben
2316 durchführen:

- 2317 1. Zur Authentisierung MUSS eine Identität mit einem X.509-Zertifikat gemäß
2318 [gemSpec_Krypt#GS-A_4360] Schlüsselgeneration "ECDSA" verwendet werden.
- 2319 2. Für „Hash und URL“ MUSS SHA-1(vgl. [RFC-7296#3.6]) verwendet werden.
- 2320 3. Für den Schlüsselaustausch MUSS ein ephemeres ECDH verwendet werden. Dabei
2321 MUSS die Kurve brainpoolP256r1 [RFC-6954] unterstützt werden. Es KÖNNEN die
2322 Kurven brainpoolP384r1, brainpoolP512r1 [RFC-6954] und ECP Gruppen 19, 20
2323 und 21 [RFC-5903] unterstützt werden.
- 2324 4. Als Verschlüsselungsverfahren im Rahmen von IKE MUSS AEAD_AES_128_GCM
2325 und AEAD_AES_256_GCM [RFC-5282] unterstützt werden (IANA.-Nr. 20)
2326 (Hinweis verpflichtend Unterstützung nach [RFC-5282#3.2]). Es MÜSSEN zudem
2327 AEAD_AES_128_GCM_12 und AEAD_AES_256_GCM_12 (IANA-Nr. 19) unterstützt
2328 werden. Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.2.1 Tabelle 2]
2329 unterstützt werden.
- 2330 5. Als PRF für die Schlüsselerzeugung MUSS PRF_HMAC_SHA2_256 (IANA-Nr. 5)
2331 [RFC-4868] unterstützt werden. Es KÖNNEN weitere Verfahren nach [TR-02021-
2332 3#3.2.2 Tabelle 3] unterstützt werden.
- 2333 6. Als Authentisierungsverfahren MUSS ECDSA-256 als Basis von brainpoolP256r1
2334 (IANA-Nr. 14) [RFC-7427] unterstützt werden. Es KÖNNEN weitere Verfahren
2335 nach [TR-02021-3#3.2.5 Tabelle 6] unterstützt werden.
- 2336 7. Für die Verschlüsselung der ESP-Pakete MUSS AES-GCM mit 16 Byte großem ICV
2337 (IANA-Nr. 20) und AES-GCM mit 12 Byte großem ICV (IANA-Nr. 19) [RFC-4106]
2338 jeweils mit 128 und 256 Bit Schlüssellänge unterstützt werden.
2339 Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt
2340 werden.
- 2341 8. Falls weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt werden,
2342 so MUSS mindestens ein Verfahren zum Integritätsschutz der ESP-Pakete
2343 aus [TR-02021-3#3.3.2 Tabelle 8] unterstützt werden. (Hinweis: bei den
2344 verpflichtend zu unterstützenden AEAD-Verfahren aus Spiegelstrich 7 ist ein
2345 zusätzlicher Integritätsschutz by-design nicht notwendig.)

2346 [$\leq \leq \leq$]

2347 Hinweis:

2348 "strongSwan" unterstützt diese Algorithmen,

2349 vgl. <https://wiki.strongswan.org/projects/strongswan/wiki/IKEv2CipherSuites>

A_17126A_17126A_17126 - IPsec-Kontext -- Verschlüsselte Kommunikation (ECC-Migration)IPsec-Kontext -- Verschlüsselte Kommunikation (ECC-Migration)IPsec-Kontext -- Verschlüsselte Kommunikation (ECC-Migration)

Alle Produkttypen, die mittels IPsec-Daten schützen, MÜSSEN dies ausschließlich auf Grundlage der in A_17125 (und ggf. GS-A_4382 vgl. diesbezüglich A_17210) als zulässig aufgeführten Verfahren und Vorgaben tun.
[<=<=<=]

5.6 ECDSA-Signaturen

5.6.1 ECDSA-Signaturen im XML-Format

A_17206A_17206A_17206 - XML-Signaturen (ECC-Migration)XML-Signaturen (ECC-Migration)XML-Signaturen (ECC-Migration)

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

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

A_17360A_17360A_17360 - XML-Signaturen (Dokumente) (ECC-Migration)XML-Signaturen (Dokumente) (ECC-Migration)XML-Signaturen (Dokumente) (ECC-Migration)

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

5.6.2 ECDSA-Signaturen im CMS-Format

A_17207A_17207A_17207 - Signaturen binärer Daten (ECC-Migration)Signaturen binärer Daten (ECC-Migration)Signaturen binärer Daten (ECC-Migration)

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

Die Anforderung A_17207 gilt für allgemeine (nicht-XML-)Datensignaturen, also auch für Tokensignaturen etc. A_17359 fordert für die Interoperabilität bei der Prüfbarkeit von

2392 Dokumentensignaturen die Verwendung des interoperablen Containerformats nach
2393 [ETSI-CAvES].

2394 **A_17359A 17359A 17359 - Signaturen binärer Daten (Dokumente) (ECC-**
2395 **Migration)Signaturen binärer Daten (Dokumente) (ECC-Migration)Signaturen**
2396 **binärer Daten (Dokumente) (ECC-Migration)**

2397 Alle Produkttypen, die (nicht-XML-)Signaturen von Dokumenten auf Basis eines ECC-
2398 Schlüssels erzeugen oder prüfen, MÜSSEN dabei die Vorgaben aus A_17207 umsetzen
2399 und die Signatur nach [ETSI-CAvES] (interoperables Container-Format) bei der
2400 Erzeugung kodieren bzw. bei der Prüfung auswerten.

2401 [$\leq \leq \leq$]

2402 Hinweis: Signaturen in PDF/A-Dokumenten werden mittels CMS kodiert.

2403 **A_17208A 17208A 17208 - Signaturen von PDF/A-Dokumenten (ECC-**
2404 **Migration)Signaturen von PDF/A-Dokumenten (ECC-Migration)Signaturen von**
2405 **PDF/A-Dokumenten (ECC-Migration)**

2406 Alle Produkttypen, die (nicht-XML-)Signaturen auf Basis eines ECC-Schlüssels erzeugen
2407 oder prüfen, MÜSSEN dafür das Signaturverfahren ECDSA [BSI-TR-03111] auf Basis der
2408 Domainparameter brainpoolP256r1 nach [PAvES-3] und [PDF/A-2] verwenden. Als
2409 Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4] verwenden.

2410 [$\leq \leq \leq$]

2411 5.7 ECIES

2412 In der TI wird für die ECC-basierte Ver- und Entschlüsselung das "Elliptic Curve
2413 Integrated Encryption Scheme (ECIES)" verwendet. Es ist das einzige ECC-basierte, von
2414 den Chipkarten der TI unterstützte, Verschlüsselungsverfahren. Das ECIES ist ein
2415 hybrides Verfahren basierend auf [ABR-1999]. Es besteht aus einem asymmetrischen Teil
2416 (elliptic curve diffie hellman) und einen symmetrischen Teil (Verschlüsselungsverfahren
2417 und MAC-Verfahren). Weiterhin ist eine Schlüsselableitungsfunktion für das Verfahren
2418 notwendig. In [gemSpec_COS#6.8.2.3 Asymmetrische Entschlüsselung mittels ELC] wird
2419 definiert, welche Varianten dieser drei notwendigen Verfahren eine Chipkarte der TI
2420 unterstützt (ECDH [BSI-TR-03111#4.3.1 Key Agreement Algorithm], HKDF mittels SHA-
2421 256 und einem Zähler nach X9.63 [BSI-TR-03110-3#A.2.3.2], AES-256-CBC und CMAC).
2422 Da im Normalfall immer für eine Identität, die Chipkarten-basiert ist, verschlüsselt wird,
2423 muss ein Sender genau diese Verfahren einsetzen. Ansonsten kann die Chipkarte das
2424 Chiffre nicht entschlüsseln, auch wenn die Chipkarte den prinzipiell richtigen privaten
2425 Schlüssel in sich trägt.

2426 Da man das gesamte Chiffre für eine Entschlüsselung auf einmal an die Karte (innerhalb
2427 einer APDU) senden muss, kann man nur etwas weniger als 8KiB entschlüsseln (bzw.
2428 64KiB bei extended APDUs, die jedoch nicht alle Kartenterminal unterstützen), obwohl
2429 das ECIES-Verfahren an für sich die Ver- und Entschlüsselung praktisch beliebig großer
2430 Datenmengen unterstützt. Auch wäre es aus Nutzersicht in Bezug auf die Performanz
2431 nicht akzeptabel, schon allein moderat große verschlüsselte Dokumente komplett zu
2432 einer Karte für eine Entschlüsselung zu transportieren (mehr als 18 Minuten würde dies
2433 für ein 10 MiB großes Dokument benötigen). Deswegen wird für die TI hier eine
2434 zusätzliche Metaebene eingeführt und normativ gefordert. Analog zu einer
2435 Verschlüsselung mittels RSAES-OAEP muss ein Sender einen Transportschlüssel zufällig
2436 erzeugen. Ein solcher Transportschlüssel, ist dann aus Sicht der Chipkarte der zu ver-
2437 oder entschlüsselnde Klartext. Der aus Nutzersicht eigentliche Klartext (Dokumente) wird
2438 nie an die Karte gesendet. Die Karte entschlüsselt den verschlüsselten Transportschlüssel

2439 und übergibt ihn anschließend an den Kartennutzer. Dieser kann mit dem
2440 Transportschlüssel nun das Dokument unabhängig von der Chipkarte entschlüsseln. Bei
2441 RSAES-OAEP wird der Transportschlüssel als Content-Encryption-Key (CEK) bezeichnet.
2442 Im hier vorliegenden Fall bei ECIES kann diese Bezeichnung zu Missverständnissen
2443 führen. Mittels ECIES wird über ein ECDH und folgender Schlüsselableitung ein
2444 Verschlüsselungsschlüssel erzeugt. Diesen kann man auch als CEK bezeichnen,
2445 weswegen im Folgenden immer nur vom Transportschlüssel gesprochen wird.

2446 Da eine solche Metaebene für eine ECIES-Chiffre-Kodierung unüblich ist (weil ohne TI-
2447 Chipkarteneinsatz unnötig), ist die Kodierung der Chiffre mittels CMS oder XMLEnc
2448 nichttrivial und wird daher im Interesse der zu erzielenden Interoperabilität in diesem
2449 Abschnitt ausführlicher dargestellt.

2450 Für die symmetrische Verschlüsselung der Nutzerdaten (Dokumente etc.) wird zufällig ein
2451 Transportschlüssel erzeugt. Für diesen gelten die Vorgaben aus GS-A_4389
2452 (symmetrische Verschlüsselung) und GS-A_4368 (Schlüsselerzeugung). Der
2453 Transportschlüssel wird dann unkodiert ("is just the \"value\" of the content-encryption
2454 key" [RFC-5652#6.4]) zur Verschlüsselung an das ECIES-Verfahren übergeben. Bei der
2455 ECIES-Verschlüsselung müssen dann die Parameter so gewählt werden, dass eine
2456 Chipkarte der TI diese unterstützt, d. h. nach [gemSpec_COS#6.8.2.3 Asymmetrische
2457 Entschlüsselung mittels ELC].

2458 Das erhaltene ECIES-Chiffre muss dann als eine ASN.1-Struktur kodiert werden, die
2459 genau dem Aufbau entspricht, den man benötigt um eine Entschlüsselung mittels einer
2460 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'.

2461

2462 **Abbildung 2: ASN.1-Kodierung des Chiffres was den Transportschlüssel enthält**

2463 Beispiel:

```
2464 poGOBgkrJAMDAggBAQd/SUOGQQRouC6tM2TQQ+RP3pptgdAaDF8Te7IVCkUBe2H+PJSLK4W/BXI
2465 X
2466 kndiBwEfftd5wk4pjzCdC2j1q14/CIWcW89nhjEC7G47UAu2ZqmbIhxstkXV3UI2UUek/qwBwtb
2467 2
2468 6aUild+5kkTZxf5674OKHSdj6IFwjggvYt9b/CTsA==
2469
2470 $ dumpasn1 a.bin
2471 0 142: [6] {
2472 3 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2473 14 67: [APPLICATION 73] {
2474 17 65: [6]
2475 : 04 68 B8 2E AD 33 64 D0 43 E4 4F DE 9A 6D 81 D0
2476 : 1A 0C 5F 13 7B B2 15 0A 45 01 7B 61 FE 3C 94 8B
2477 : 2B 85 BF 05 72 17 92 77 62 07 01 1F 7E D7 79 C2
2478 : 4E 29 8F 30 9D 0B 68 F5 AB 5E 3F 08 85 9C 5B CF
2479 : 67
```

```
2480      :      }
2481  84  49:  [6]
2482      :      02 EC 6E 3B 50 0B B6 66 A9 9B 22 1C 6C B6 45 D5
2483      :      DD 42 36 51 47 A4 FE AC 01 C2 D6 F6 E9 A5 22 95
2484      :      DF B9 92 44 D9 5D FE 7A EF 83 8A 1D 27 63 E8 81
2485      :      70
2486 135  8:  [14] 2F 85 8B 7D 6F F0 93 B0
2487      :      }
```

2488 Diese Datenstruktur soll konzeptionell so behandelt werden, wie ein RSA-OAEP-Chifftrat
2489 eines CEK. Es ist jedoch die hiermit neu definierte OID `oid_ti_ecies_transport_encryption`
2490 [`gemSpec_OID`] zu verwenden.

2491

2492 Ein Beispiel aus [`gemSpec_SMIME-KOMLE`] dient als Vorlage für die Darstellung der zu
2493 verwendenden Kodierung mittels CMS (S/MIME):

```
2494 ContentInfo
2495 .contentType: 1.2.840.113549.1.9.16.1.23 (id-ct-authEnvelopedData)
2496 ..AuthEnvelopedData
2497 ...version: 0
2498 ...recipientInfos
2499 ....RecipientInfo
2500 .....ktri
2501 .....version: 0
2502 .....rid
2503 .....issuerAndSerialNumber
2504 .....issuer
2505 .....[... weitere Kindelemente ...]
2506 .....SerialNumber: 123456789
2507 .....keyEncryptionAlgorithm
2508 .....oid_ti_ecies_transport_encryption
2509 .....encryptedKey: [ ... ASN.1-Struktur (Tupel (PO, C, T) in
2510 kartenkompatibler ASN.1-Binärverpackung) ... ]
2511 ....RecipientInfo
2512 .....ktriversion: 0
2513 .....rid
2514 .....issuerAndSerialNumber
2515 .....issuer
2516 .....[... weitere Kindelemente ...]
2517 .....SerialNumber: 314159265
2518 .....keyEncryptionAlgorithm
2519 .....OID TI-ECIES-TransportEncryption
2520 .....encryptedKey: [ ... ASN.1-Struktur (Tupel (PO, C, T) in
2521 kartenkompatibler ASN.1-Binärverpackung) ... ]
2522 ...authEncryptedContentInfo
2523 ...contentType: 1.2.840.113549.1.7.1 (id-data)
2524 ...contentEncryptionAlgorithm:
2525 .....algorithm: 2.16.840.1.101.3.4.1.46 (id-aes256-gcm)
2526 .....parameters:
2527 .....aes-nonce: [... IV ...]
2528 .....aes-ICVlen: [... ICVLen ... ]
2529 ....encryptedContent: [...]
2530 ...mac: [...]
2531 ...unauthAttrs
2532 ....Attribute (id-recipientEmails)
2533 .....attrType: komle-recipient-emails
```


2534 und so weiter ...

2535 Für die Kodierung von TI-ECIES-Chiffreten innerhalb von XML wird hiermit der neue
2536 Identifier "http://gematik.de/ecies/2019" definiert. Für die XML-Kodierung ist eine
2537 Kodierung analog der folgenden Struktur zu verwenden.

```
2538 <?xml version="1.0" encoding="UTF-8"?>
2539 <xenc:EncryptedData
2540     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2541     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
2542     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2543     xmlns:dsig11="http://www.w3.org/2009/xmldsig11#"
2544     xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
2545     Type="http://www.w3.org/2001/04/xmlenc#"
2546     <xenc:EncryptionMethod
2547         Algorithm="http://www.w3.org/2009/xmlenc11#aes256-gcm" />
2548 <!-- Damit ist len(IV)=12 Byte und TagLen=16 Byte, vgl. [XMLEnc#5.2.4 AES-
2549 GCM] -->
2550 <ds:KeyInfo>
2551     <xenc:EncryptedKey>
2552         <!-- Hybridschlüssel für einen Empfänger, für weitere Empfänger gäbe es
2553 jeweils ein weiteres EncryptedKey-Element -->
2554         <xenc:EncryptionMethod Algorithm="http://gematik.de/ecies/2019" />
2555         <!-- Die Version des speziellen ECIES -->
2556         <ds:KeyInfo>
2557             <xenc:RecipientKeyInfo>
2558                 <ds:X509Data>
2559                     <ds:X509Certificate>
2560                         <!-- Base64-kodiertes X.509-Zertifikat (DER) des Empfängers -->
2561                         </ds:X509Certificate>
2562                     </ds:X509Data>
2563                 </xenc:RecipientKeyInfo>
2564             </ds:KeyInfo>
2565             <xenc:CipherData>
2566                 <xenc:CipherValue>
2567                     <!-- Base64-kodierte ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler
2568 ASN.1-Binärverpackung) -->
2569                     </xenc:CipherValue>
2570                 </xenc:CipherData>
2571             </xenc:EncryptedKey>
2572         </ds:KeyInfo>
2573         <xenc:CipherData>
2574             <xenc:CipherValue>
2575                 <!--
2576 Base64-kodiertes symmetrisch mittels AES-256-GCM verschlüsseltes Dokument
2577 (IV || ciphertext || Tag) mit len(IV)=12 Byte und len(Tag)=16 Byte,
2578 vgl. [XMLEnc#5.2.4 AES-GCM]
2579 -->
2580                 </xenc:CipherValue>
2581             </xenc:CipherData>
2582         </xenc:EncryptedData>
```

2584

A_17220A_17220A_17220 - Verschlüsselung binärer Daten (ECIES) (ECC-Migration)
Verschlüsselung binärer Daten (ECIES) (ECC-Migration)
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]).
2. Mit diesem Transportschlüssel MÜSSEN die Nutzerdaten mittels AES-GCM und den Vorgaben aus GS-A_4389 verschlüsselt werden.
3. Der Transportschlüssel MUSS unkodiert mit den in [gemSpec_COS#6.8.1.4 ELC Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden (siehe Erklärung in [gemSpec_Krypt#ECIES]). Das damit entstehende Chifftrat wird im Folgenden als Transport-Chifftrat bezeichnet.
4. Das Transport-Chifftrat MUSS wie in [gemSpec_Krypt#5.7-ECIES] beschrieben in eine ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler ASN.1-Binärverpackung kodiert werden.
5. Diese Kodierung MUSS in eine keyEncryptionAlgorithm-Datenstruktur mit der OID oid_ti_ecies_transport_encryption [gemSpec_OID] eingebracht werden.
6. Die restliche Kodierung des mittels AES-GCM erzeugten Chifftrats der Nutzerdaten MUSS wie in CMS üblich [RFC-5626] [RFC-5084] erfolgen (vgl. Darstellung in [gemSpec_Krypt#5.7-ECIES]).

[<=<=<=]

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

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

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

[<=<=<=]

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

5.7.1 ECIES und authentifizierte Broadcast-Encryption

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

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

5.7.2 ECIES und mobKT

Ein "Mobiles Kartenterminal" [gemSpec_MobKT] ist so etwas wie ein Tablet mit zwei Chipkartenslots. Es ermöglicht einem LE außerhalb seiner Praxisräumlichkeiten (also insbesondere ohne Konnektor und stationäres eHealth-Kartenterminal), auf Daten eines Versicherten auf dessen eGK zuzugreifen. Das Mobile Kartenterminal muss die dabei ausgelesenen versichertenspezifischen Daten verschlüsselt lokal im Gerät ablegen. Wenn das Mobile Kartenterminal verloren geht, sind damit diese Daten geschützt. Sie können erst mit Hilfe des gesteckten und freigeschalteten HBA des LE wieder entschlüsselt (genutzt) werden, bspw. zur Übertragung ins Primärsystem. Für die Verschlüsselung muss nach Ende 2023 ECIES und nicht mehr RSA-OAEP verwendet werden. Es gelten dafür fachlich quasi die gleichen Vorgaben wie in A_17220, nur gibt es keine Notwendigkeit in Bezug auf die Kodierung des Chiffrats interoperabel sein zu müssen. Denn nur das spezifische Mobile Kartenterminal selbst muss die Daten ver- und entschlüsseln können im Sinne von "die Kodierung der Chifftrate auswerten können". Deshalb gibt es nachfolgend eine Spezialisierung von A_17220 für das Mobile Kartenterminal.

A_17575A 17575A 17575 - MobKT: Verschlüsselung binärer Daten (ECIES) (ECC-Migration) MobKT: Verschlüsselung binärer Daten (ECIES) (ECC-Migration) MobKT: Verschlüsselung binärer Daten (ECIES) (ECC-Migration)

Ein Mobiles Kartenterminal MUSS folgende Vorgaben umsetzen.

1. Für die Verschlüsselung der Versichertendaten MUSS ein 256-Bit AES-Schlüssel zufällig erzeugt werden (vgl. GS-A_4368). Dieser Schlüssel wird im Folgenden als Transportschlüssel bezeichnet.
2. Mit diesem Transportschlüssel MÜSSEN die Versichertendaten mit AES-GCM und den Vorgaben aus GS-A_4389 verschlüsselt werden.

- 2673 3. Der Transportschlüssel MUSS mit den in [gemSpec_COS#6.8.1.4 ELC
2674 Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden
2675 (siehe Erklärung in [gemSpec_Krypt#5.7- ECIES] und [gemSpec_Krypt#Hinweis
2676 zu A_17575]).
- 2677 4. Falls auf dem gesteckten HBA ein ECC-basiertes ENC-Zertifikat vorhanden ist, so
2678 MUSS ECIES für die Ver- und Entschlüsselung des Transportschlüssels verwendet
2679 werden, anstatt von RSA-OAEP. Falls noch kein ECIES-verschlüsselter
2680 Transportschlüssel im Mobilien Kartenterminal vorliegt, sondern ein RSA-OAEP-
2681 verschlüsselter Transportschlüssel, so MUSS dieser Transportschlüssel zusätzlich
2682 mittels ECIES und dem ECC-ENC-Schlüssel des HBAs des LE verschlüsselt werden.
- 2683 [\leq \leq \leq]
- 2684 Hinweis zu A_17575:
2685 In einem HBA steht die Schnittstelle [gemSpec_COS#6.8.1.4 ELC Verschlüsselung] für
2686 die Verschlüsselung des Transportschlüssels zur Verfügung. Dass der Transportschlüssel
2687 verschlüsselt werden muss, wird nur sehr selten als Anwendungsfall vorkommen. Die
2688 Entschlüsselung - notwendiger Weise mit und durch den HBA - jedoch häufig.

2689 5.8 ECC-Migration eHealth-KT

- 2690 Mit A_17089 und A_17090 (vgl. Abschnitt 3.3.2) wird vom eHealth-Kartenterminal die
2691 Unterstützung der mit der Kartengeneration 2.1 (vgl. [gemSpec_gSMC-
2692 KT_ObjSys_G2.1]) hinzugekommenen ECDSA-basierten Identität bereitgestellt.
- 2693 **A_17089A_17089A_17089 - eHealth-Kartenterminals: TLS-Verbindungen (ECC-
2694 Migration)eHealth-Kartenterminals: TLS-Verbindungen (ECC-Migration)eHealth-
2695 Kartenterminals: TLS-Verbindungen (ECC-Migration)**
- 2696 Ein eHealth-Kartenterminal MUSS prüfen, ob die in ihm gesteckte SMC-KT für die TLS-
2697 Verbindung zum Konnektor eine RSA-basierte Identität (AUT) und/oder eine ECDSA-
2698 basierte Identität besitzt (vgl. [gemSpec_gSMC-KT_ObjSys_G2.1], bspw. jeweils EFs mit
2699 ShortFileIdentifier 1 und 4 prüfen).
- 2700 Falls eine RSA-basierte Identität dort vorhanden ist, so MUSS das eHealth-Kartenterminal
2701 folgende TLS-folgende Vorgaben erfüllen:
- 2702 1. Als Ciphersuiten MÜSSEN TLS_DHE_RSA_WITH_AES_128_CBC_SHA und
2703 TLS_DHE_RSA_WITH_AES_256_CBC_SHA unterstützt werden.
- 2704 2. Es MUSS dabei für die Schlüsselaushandlung Gruppe 14 (definiert in [RFC-3526],
2705 verwendbar bis Ende 2023) unterstützt und verwendet werden.
- 2706 3. Der private ephemere DH-Exponent für den Schlüsselaustausch MUSS eine Länge
2707 von mindestens 256 Bit haben.
- 2708 Falls eine ECDSA-basierte Identität vorhanden ist, so MUSS das eHealth-Kartenterminal
2709 zusätzlich folgende Vorgaben erfüllen:
- 2710 4. Als Ciphersuiten MÜSSEN TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
2711 (0xC0,0x2B) und TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
2712 (0xC0,0x2C) unterstützt werden.
- 2713 5. Als Basis für den ephemeren ECDH MUSS die Kurve brainpoolP256r1 und
2714 brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt und verwendet
2715 werden.

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

[<=<=<=]

A_17090A 17090A 17090 - eHealth-Kartenterminals: Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal (ECC-Migration)
eHealth-Kartenterminals: Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal (ECC-Migration)
eHealth-Kartenterminals: Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal (ECC-Migration)

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

1. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine RSA-basierte Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret (ShS.AUT.KT vgl. [gemSpec_KT#2.5.2.1, 3.7.2.1]) RSASSA-PSS [PKCS#1] und SHA-256 verwendet werden. (Hinweis: die Parameter für RSASSA-PSS wie MGF oder Salt-Länge sind durch die SMC-KT eindeutig und fest vorgegeben und werden deshalb hier nicht aufgeführt.)
2. Falls für die aktuelle TLS-Verbindung mit dem Konnektor eine ECDSA-basierte Ciphersuite verwendet wird, so MUSS für die Signatur des Shared-Secret ECDSA [BSI-TR-03111] und SHA-256 verwendet werden (Hinweis: die zu verwendenden Domainparameter (Kurve etc.) sind durch die SMC-KT eindeutig und fest vorgegeben).

[<=<=<=]

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

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

A_17183A 17183A 17183 - CA-Zertifikate der relevanten TSP speichern (ECC-Migration)
CA-Zertifikate der relevanten TSP speichern (ECC-Migration)
CA-Zertifikate der relevanten TSP speichern (ECC-Migration)

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

[<=<=<=]

5.9 ECC-Migration Konnektor

Die Verpflichtung der Implementierung bestimmter (s. u.) für die ECC-Migration notwendiger Funktionalitäten wird für den PTV4-Konnektor mit Hinblick auf die fristgerechte Umsetzung der ePA zunächst ausgesetzt:

- Für die TLS-Schnittstellen, die bereits mit dem PTV3-Konnektor zertifiziert wurden (Schnittstellen zu Kartenterminals, Clientsystemen, Intermediär, zentralen

2760 Diensten), wird in PTV4 auf eine verpflichtende ECC-Unterstützung zunächst
2761 verzichtet.

2762 • Ebenso wird auf die verpflichtende Umsetzung der ECC-Unterstützung an der
2763 IPsec/IKE-Schnittstelle zum VPN-Konzentrator zunächst verzichtet.

2764 Unverändert verpflichtend gefordert wird die ECC-Unterstützung an der Schnittstelle zum
2765 ePA-Aktensystem, bei den Karten und für KOM-LE. Insbesondere bedeutet dies, dass ein
2766 PTV4-Konnektor auch weiterhin
2767

2768 • die TSL(ECC-RSA) prüfen und verwenden muss (vgl. [A_17205](#)),
2769 • die Signaturerstellung und -prüfungen auf ECDSA-Basis beherrschen muss (vgl.
2770 [A_17206](#), [A_17360](#), [A_17207](#), [A_17359](#), [A_17208](#) und die VAU-Protokoll und die
2771 SGD-Protokoll relevanten Operationen) und

2772 • die Ver- und Entschlüsselung über das ECIES-Verfahren (vgl. [A_17220](#) , [A_17221-](#)
2773 [01](#) und die Transport-Verschlüsselung innerhalb des SGD-Protokolls mit
2774 [A_17875](#))

2775 unterstützen muss.
2776

2777 Werden die SOLL-Anforderungen [A_17904](#) und [A_18624](#) nicht umgesetzt, so ist dies mit
2778 einem Firmwareupdate im Jahre 2021 nachzuholen.

2779 **[A_17094](#)[A_17094](#) [17094](#) - TLS-Verbindungen Konnektor (ECC-Migration)**[TLS-](#)**
2780 **[Verbindungen Konnektor \(ECC-Migration\)](#)[TLS-Verbindungen Konnektor \(ECC-](#)**
2781 **[Migration\)](#)****

2782 Der Konnektor SOLL zusätzlich zu den RSA-basierten TLS-Ciphersuiten (vgl. GS-A_4385
2783 und GS-A_5345) die TLS-Ciphersuiten

2784 1. TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 und
2785 2. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

2786 unterstützen. Dabei MÜSSEN bei dem ephemeren Elliptic-Curve-Diffie-Hellman-
2787 Schlüsselaustausch die Kurven P-256 und P-384 [FIPS-186-4] und die Kurven
2788 brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt
2789 werden. Andere Kurven SOLLEN NICHT verwendet werden.

2790 Falls der Konnektor in der Rolle TLS-Client agiert, so MUSS er die eben genannten
2791 Ciphersuiten gegenüber RSA-basierten Ciphersuiten (vgl. GS-A_4384) bevorzugen (in der
2792 Liste "cipher_suites" beim ClientHello vorne an stellen, vgl. [RFC-5256#7.4.1.2 Client
2793 Hello]).
2794 [[<=<=<=](#)]

2795

2796 **[A_18624](#)[A_18624](#) [18624](#) - Konnektor, IPsec/IKE: optionale ECC-**
2797 **[Unterstützung](#)[Konnektor, IPsec/IKE: optionale ECC-Unterstützung](#)[Konnektor,](#)**
2798 **[IPsec/IKE: optionale ECC-Unterstützung](#)**

2799 Ein Konnektor SOLL bei der Verwendung von IPsec/IKE neben den Verfahren aus GS-
2800 A_4382 bzw. GS-A_4383 auch Verfahren auf Basis von ECC verwenden. Falls der
2801 Konnektor dies tut, so MUSS er die Vorgaben aus [A_17125](#) und [A_17126](#) umsetzen.
2802 [[<=<=<=](#)]

A_17209 [A_17209A](#) [17209](#) - Signaturverfahren für externe Authentisierung (ECC-Migration)
[Signaturverfahren für externe Authentisierung \(ECC-Migration\)](#)
[Signaturverfahren für externe Authentisierung \(ECC-Migration\)](#)

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

5.10 Verschiedene Produkttypen und ECC-Migration (informativ)

Dem VPN-Zugangsdienst sind die IPsec-Anforderungen [A_17125](#) und [A_17126](#) zugewiesen, ebenso [A_17205](#). Im Rahmen der Registrierung bei einem VPN-Zugangsdienst wird vom Konnektor eine Signatur mittels einer SMC-B erzeugt. Diese muss der VPN-Zugangsdienst (Registrierungsserver) prüfen können, dafür gilt für diesen [A_17206](#).

Für die Produkttypen, die bei der Anwendung VSDM verwendet werden, ist die ECC-RSA-TSL-Auswertung [A_17205](#) und die ECC-Unterstützung bei TLS [A_17124](#) relevant.

Fachanwendungsspezifische Anpassungen aufgrund der ECC-Migration befinden sich in den jeweiligen Spezifikationen (bspw. [\[gemSpec CM KOMLE#A_17464\]](#)).

6 Kommunikationsprotokoll zwischen ePA-VAU und ePA-Clients

6.1 Motivation

Die Architekturentscheidung, dass auf der Verbindungsstrecke zwischen einem ePA-Frontend eines Versicherten (Client) bzw. eines FM ePA (Client) und einer VAU (Server) immer HTTP über ein Gateway als Kommunikationsprotokoll verwendet wird, hat Vorteile. Ein Nachteil ist, dass damit keine direkte TLS-Verbindung zwischen Client und Server möglich ist. Der TLS-Kanal terminiert am Gateway. Um die "letzten Meile" zwischen Gateway und VAU innerhalb des ePA-Aktensystems nicht ungeschützt zu lassen, wird das "VAU-Protokoll" eingeführt und verwendet. Es wird nicht der Weg gewählt HTTP über TLS über HTTP über TLS zu tunneln. Stattdessen wird das folgende Protokoll eingeführt als eine leichtgewichtige Sicherungsschicht zwischen einer VAU (Server) und einem ePA-Frontend eines Versicherten (Client) bzw. eines FM ePA (Client). Der Server und der Client besitzen jeweils eine kryptographische Langzeitidentität. Diese sind Grundlage für einen beidseitig authentisierten ephemeren ECDH. Aus dem gemeinsamen ECDH-Geheimnis wird mittels einer HKDF ein AES-Schlüssel abgeleitet. Per AES-GCM werden nach dem Schlüsselaustausch alle ausgetauschten Nutzerdaten kryptographisch gesichert. Es gibt einen Nachrichtenzähler der vor Replay-Attacken schützt.

Die gematik stellt eine Beispielimplementierung bereit (<https://fachportal.gematik.de/service/entwicklung/>).

6.2 Übersicht

Es gibt bei der Kommunikation, die das Protokoll steuert, zwei aktive und einen passiven Teilnehmer. Der Client initiiert die Kommunikation per HTTP-POST-Request. Der Server antwortet als HTTP-Server auf diesen Request mit einer HTTP-Response. Das Gateway leitet die Daten weiter und erscheint unsichtbar. Das Gateway erzwingt die Verwendung des HTTP-Protokolls.

Das Kommunikationsprotokoll hat als Hauptaufgabe die zwischen Client und Server (VAU) auszutauschenden Nutzerdaten vor dem Gateway in Bezug auf Vertraulichkeit, Authentizität, Integrität (inkl. Verhinderung von Replay-Attacken) zu schützen. Das Protokoll bietet (absichtlich) keinen Schutz der HTTP-Header-Informationen. Es ist bei der ePA-Architektur sichergestellt, dass eine Änderung der HTTP-Header-Informationen keine negativen Auswirkungen auf die Vertraulichkeit, Integrität und Authentizität der Nutzerdaten hat. Hinweis zum besseren Verständnis: Zwischen Gateway und Client besteht immer eine TLS-Verbindung, so dass nur das Gateway bzw. nur ein Angreifer im Aktensystem selbst die Header-Informationen ändern könnte.

Ein Designziel ist es, den Verbindungsaufbau möglichst "menschenslesbar" zu gestalten und so die Implementierung und die Fehlersuche (u. a. auch im Betrieb) zu erleichtern. So sind die meisten Nachrichten einfache, menschenlesbare JSON-Objekte. Die im Protokoll definierten Fehlermeldungen (VAUServerError-Nachricht) haben den Hauptzweck, die manuelle Fehleranalyse zu erleichtern. Das VAU-Protokoll ist relativ einfach, so dass ein Client oder ein Server bei Auftreten eines Fehlers selten etwas

2860 anderes tun kann, als die Protokollabarbeitung abubrechen (und als Client einen neuen
2861 Protokolldurchlauf zu initiieren). Dies bedeutet: die Art der aufgetretenen Fehlernachricht
2862 ist für den Client i. d. R. irrelevant, weil die Handlungsoptionen sehr begrenzt sind. Erst
2863 bei der manuellen Fehleranalyse werden die differenzierten Fehlermeldungen hilfreich.

2864 Viele Nachrichten und Datenfelder ähneln fachlich Bestandteilen aus dem TLS-Protokoll
2865 und werden absichtlich anders benannt, um Verwechslungen zu vermeiden.

2866 Das Protokoll bietet die Möglichkeit, zukünftig verschiedene CipherConfiguration (im TLS-
2867 Protokoll entspricht dies den TLS-Cipher-Suiten) zu unterstützen; aktuell gibt es genau
2868 eine ("AES-256-GCM-BrainpoolP256r1-SHA-256").

2869 Der Ablauf der Schlüsselaushandlung des VAU-Protokolls ist im folgenden
2870 Ablaufdiagramm dargestellt.

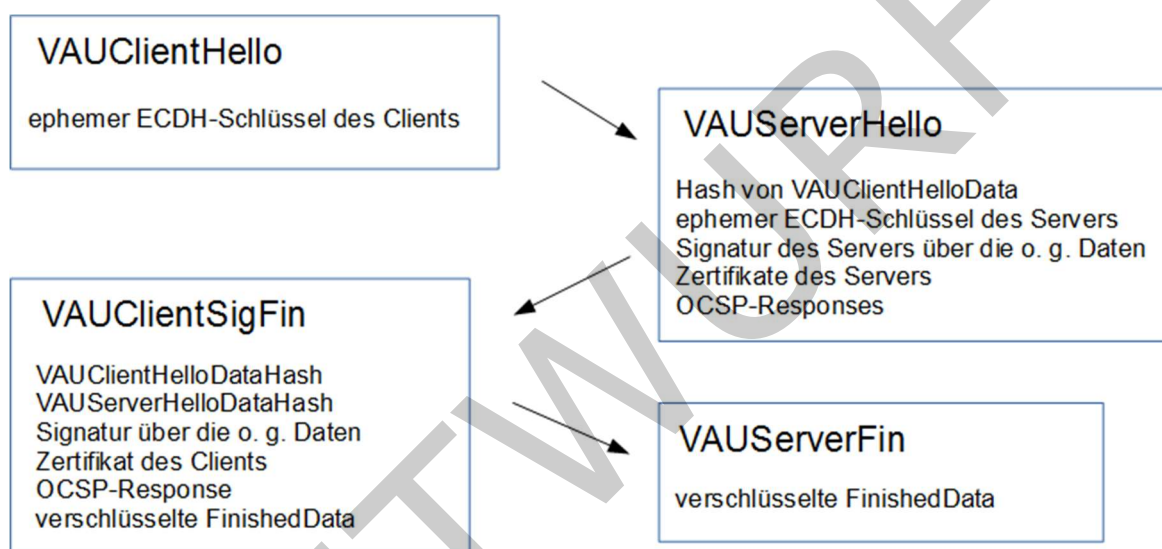


Abbildung 3: Übersicht über das VAU-Protokoll

2871
2872 Da die Werte des ephemeren öffentlichen ECDH-Schlüssels des jeweiligen anderen
2873 Kommunikationspartners in die eigene Signatur eingehen, können sowohl Client und
2874 Server bei der Signaturprüfung sicherstellen, dass die Schlüsselaushandlung frisch ist
2875 (vgl. [Boyd-Mathuria-2003#Abschnitt "1.5 Freshness"]).

2877 Aus dem gemeinsamen ECDH-Geheimnis wird ein AES-Schlüssel abgeleitet und damit die
2878 weitere Kommunikation mittels AES-GCM in Bezug auf Vertraulichkeit und Integrität
2879 geschützt. Der dabei explizit mitgelieferte Nachrichtenzähler schützt vor Replay-Attacken.

2880 **A_16884A 16884A 16884 - VAU-Protokoll: Nachrichtentypen und HTTP-**
2881 **Content-TypeVAU-Protokoll: Nachrichtentypen und HTTP-Content-TypeVAU-**
2882 **Protokoll: Nachrichtentypen und HTTP-Content-Type**

2883 Es gibt genau zwei Nachrichtenarten: (1) Nachrichten zur Schlüsselaushandlung
2884 (VAUClientHello, VAUServerHello, VAUClientSigFin und VAUServerFin) und
2885 Fehlermeldungsübermittlung (VAUServerError) und (2) Nachrichten, die kryptographisch
2886 geschützte Nutzerdaten transportieren.

2887 Typ-(1)-Nachrichten MÜSSEN vom Client und vom Server jeweils per HTTP mit dem
2888 Content-Type 'application/json' übermittelt werden und Typ-(2)-Nachrichten mit dem
2889 Content-Type 'application/octet-stream'. [$\leq \leq \leq$]

A_17074A 17074A 17074 - VAU-Protokoll: Ignorieren von zusätzlichen Datenfeldern in Protokoll-Nachrichten
VAU-Protokoll: Ignorieren von zusätzlichen Datenfeldern in Protokoll-Nachrichten
VAU-Protokoll: Ignorieren von zusätzlichen Datenfeldern in Protokoll-Nachrichten

Ein Client oder ein Server MUSS zusätzliche (i. S. v. ihm unbekannte) Datenfelder (Key-Value-Paare) in JSON-Objekten (Typ-(1)-Nachrichten und "Data"-Feldern darin) im Rahmen des VAU-Protokolls ignorieren. [$\leq \leq \leq$]

A_17081A 17081A 17081 - VAU-Protokoll: zu verwendende Signaturschlüssel
VAU-Protokoll: zu verwendende Signaturschlüssel
VAU-Protokoll: zu verwendende Signaturschlüssel

Ein Client und ein Server MUSS für die Signatur im Rahmen des VAU-Protokolls (VAU-ServerHello- und VAU-ClientSigFin-Nachrichten) Schlüsselmateriale verwenden, dass dediziert für die Entity-Authentication vorgesehen ist (AUT-Schlüsselmateriale (einer eGK, einer SMC-B etc.) oder privates Schlüsselmateriale der VAU-Server-Identität (Rollenprofil "oid_epa_vau")). [$\leq \leq \leq$]

6.3 VAU-ClientHello-Nachricht

A_16883-01A 16883-01A 16883-01 - VAU-Protokoll: Aufbau VAU-ClientHello-Nachricht
VAU-Protokoll: Aufbau VAU-ClientHello-Nachricht
VAU-Protokoll: Aufbau VAU-ClientHello-Nachricht

Der Client MUSS die Kommunikation mittels einer VAU-ClientHello-Nachricht initiieren. Dafür erzeugt er zunächst eine VAU-ClientHelloData-Datenstruktur der Form

```
{  
  "DataType" : "VAUClientHelloData",  
  "CipherConfiguration" : [ "AES-256-GCM-BrainpoolP256r1-SHA-256" ],  
  "PublicKey" : "...Base64-kodierter-ECC-Schlüssel (DER) ...",  
  "AuthorizationAssertion" : "Authorization Assertion (Base64-kodiert)",  
  "CertificateHash" : "...Base64-kodierter SHA-256 Hashwert des Client-X.509-Zertifikats"  
}
```

Der Client MUSS im Rahmen der Schlüsselaushandlung ein ECDH-Schlüsselpaar basierend auf der Kurve BrainpoolP256r1 [RFC-5639] erzeugen. Er MUSS im "PublicKey"-Feld den öffentlichen Punkt des ephemeren ECDH-Schlüsselpaars Base64-kodiert gemäß [TR-03111#5.1.1 X9.62 Format] eintragen. Im "AuthorizationAssertion"-Feld MUSS der Client die Base64-kodierte Authorization-Assertion gemäß A_15592 eintragen. Der Client MUSS im "CertificateHash"-Feld den Base64-kodierten Hashwert seines Client-Zertifikats (AUT- oder AUT_alt-Zertifikat) eintragen (Der Hashwert wird vom kompletten DER-kodierten X.509-Zertifikat inkl. äußerer Zertifikatssignatur erzeugt. Der SHA-256 Hashwert (d. h. 256-Bit = 32 Byte) wird anschließend Base64-kodiert. Diese Kodierung wird als Wert bei "CertificateHash" eingetragen.).

Beispiel:

```
{  
  "DataType" : "VAUClientHelloData",  
  "CipherConfiguration": [ "AES-256-GCM-BrainpoolP256r1-SHA-256" ],  
  "PublicKey" :  
    "MFowFAYHKoZIzj0CAQYJKyQDAwIIAQEHA0IABDY8OMZlrJpLUdgnm8gHbevPFjemkL8IxMXohQ  
    lw3VHePf+T1lW+P0nW9VpnU1SxwCkjY1PU6HGT+3wawKvRIE=",  
  "AuthorizationAssertion" : ".....",  
  "CertificateHash" : "wu72yzp4KdteWV/vJUcKW14UL+FIJyWcwgETbxwDK+4="
```

```
2939 }
2940
2941 Hinweis: Der öffentliche Schlüssel im Beispiel hat nach der Base64-Dekodierung folgende
2942 ASN.1-Datenstruktur:
2943 0 90: SEQUENCE {
2944 2 20: SEQUENCE {
2945 4 7: OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
2946 13 9: OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
2947 : }
2948 24 66: BIT STRING
2949 : 04 36 3C 38 C6 65 AC 9A 4B 51 D8 27 9B C8 07 6D
2950 : EB E9 16 37 A6 90 BF 08 C4 C5 E8 85 09 70 DD 51
2951 : DE 3D FF 93 D6 55 BE 3F 49 D6 F5 5A 67 53 54 B1
2952 : C0 29 23 63 53 D4 E8 71 93 4F ED F0 6B 02 AF 44
2953 : 81
2954 : }
```

Der Client MUSS diese Datenstruktur Base64-kodieren und vom Ergebnis einen SHA-256-Hashwert bilden, den er später mit dem im VAUClientHello aufgeführten Wert vergleichen muss. In das Datenfeld "Data" in der folgenden VAUClientHello-Nachricht MUSS er die Base64-kodierte VAUClientHelloData-Daten eintragen.

Die VAUClientHello-Nachricht hat folgenden Aufbau:

```
{
"MessageType" : "VAUClientHello",
"Data" : "...Base64-kodierte-VAUClientHelloData..."
}
```

[<=<=<=<=]

A_16897A 16897A 16897 - VAU-Protokoll: Versand der VAUClientHello-Nachricht **VAU-Protokoll: Versand der VAUClientHello-Nachricht** **VAU-Protokoll: Versand der VAUClientHello-Nachricht**

Der Client MUSS die VAUClientHello-Nachricht per HTTP mit dem Content-Type 'application/json' an den Server senden. [<=<=<=<=]

6.4 VAUClientHello-Nachricht

A_16898A 16898A 16898 - VAU-Protokoll: Erzeugung des Hashwert vom Data-Feld aus der VAUClientHello-Nachricht **VAU-Protokoll: Erzeugung des Hashwert vom Data-Feld aus der VAUClientHello-Nachricht** **VAU-Protokoll: Erzeugung des Hashwert vom Data-Feld aus der VAUClientHello-Nachricht**

Der Server MUSS beim Empfang der VAUClientHello-Nachricht einen SHA-256-Hashwert der Daten im Data-Feld (zunächst keine Base64-Dekodierung durchführen) erzeugen. [<=<=<=<=]

A_16901-02A 16901-02A 16901-02 - VAU-Protokoll: Aufbau der VAUClientHello-Nachricht **VAU-Protokoll: Aufbau der VAUClientHello-Nachricht** **VAU-Protokoll: Aufbau der VAUClientHello-Nachricht**

Der Server MUSS auf die VAUClientHello-Nachricht mit einer VAUClientHello-Nachricht, folgender Form, antworten

```
{
"MessageType" : "VAUClientHello",
```

```
2988 "Data" : "...Base64-kodierte-Daten...",
2989 "Signature" : "...Base64-kodierte-ECDSA-Signatur...",
2990 "Certificate" : "...Base64-kodiertes-Signaturzertifikat...",
2991 "OCSPResponse" : "...Base64-kodierte-OCSP-Response-für-dieses-
2992 Zertifikat..."
2993 }
```

2994
2995 Die ECDSA-Signatur MUSS nach [TR-03111#5.2.2. X9.62 Format] ("ecdsa-with-
2996 Sha256") kodiert sein.

2997 In den Daten von "Data" MUSS der Server in die Base64-kodierte VAUHelloData-
2998 Datenstruktur der folgenden Form eintragen. Der Server MUSS im "CertificateHash"-Feld
2999 den Base64-kodierten SHA-256 Hashwert des Server-X.509-Zertifikats eintragen. (Der
3000 Hashwert wird vom kompletten DER-kodierten X.509-Zertifikat inkl. äußerer
3001 Zertifikatssignatur erzeugt. Der SHA-256 Hashwert (d. h. 256-Bit = 32 Byte) wird
3002 anschließend Base64-kodiert. Diese Kodierung wird als Wert bei "CertificateHash"
3003 eingetragen.)

```
3004 {
3005   "DataType" : "VAUHelloData",
3006   "CipherConfiguration" : [ "AES-256-GCM-BrainpoolP256r1-SHA-256" ],
3007   "VAUClientHelloDataHash" : "...SHA-256-Hashwert-des-erhaltenen-Data-Felds-
3008 in-VAUClientHello...",
3009   "PublicKey" : "...Base64-kodierter-ECC-Schlüssel(DER)...",
3010   "CertificateHash" : "...Base64-kodierter SHA-256 Hashwert des Server-X.509-
3011 Zertifikats"
3012 }
```

3013 Der Server MUSS im "PublicKey"-Feld den öffentlichen Punkt seines ephemeren ECDH-
3014 Schlüsselpaars Base64-kodiert gemäß [TR-03111#5.1.1 X9.62 Format] eintragen.

3015
3016 Der Server MUSS im Feld "VAUClientHelloDataHash" den Base64-kodierten SHA-256-
3017 Hashwert der empfangenen VAUClientHelloData (ohne Base64-Dekodierung) eintragen
3018 (vgl. A_16898).

3019
3020 Der Server MUSS die in der Datenstruktur (VAUHelloData) angegebene Signatur
3021 erzeugen (über den Base64-kodierten Wert im "Data"-Feld). Im "Certificate"-Feld MUSS
3022 er das für eine Signaturprüfung notwendige EE-Zertifikat eintragen und im
3023 "OCSPResponse"-Feld die OCSP-Response, die nicht älter als 24 Stunden sein darf, für
3024 dieses EE-Zertifikat.

3025 [$\leq \leq \leq$]

3026 **A_16902A 16902A 16902 - VAU-Protokoll: Versand der VAUHelloData-**
3027 **Nachricht**VAU-Protokoll: Versand der VAUHelloData-NachrichtVAU-Protokoll:
3028 Versand der VAUHelloData-Nachricht

3029 Der Server MUSS auf eine VAUClientHello-Nachricht mit einer VAUHelloData-Nachricht
3030 antworten. Der Server MUSS die VAUHelloData-Nachricht per HTTP mit dem Content-
3031 Type 'application/json' an den Client senden. [$\leq \leq \leq$]

3032 **A_16903A 16903A 16903 - VAU-Protokoll: Client, Prüfung des**
3033 **VAUClientHelloDataHash-Werts (aus VAUHelloData)**VAU-Protokoll: Client,
3034 Prüfung des VAUClientHelloDataHash-Werts (aus VAUHelloData)VAU-
3035 Protokoll: Client, Prüfung des VAUClientHelloDataHash-Werts (aus
3036 VAUHelloData)

3037 Der Client MUSS beim Empfang der VAUHelloData-Nachricht den Hashwert
3038 "VAUClientHelloDataHash" (vgl. A_16901) mit dem vom ihm (Client) vor dem Versand
3039 der VAUClientHello-Nachricht (vgl. A_16883) errechneten Wert vergleichen. Sind die

beiden Werte verschieden, so MUSS der Client den Protokollablauf
abbrechen. [$\leq \leq \leq$]

**A_16941-01A_16941-01A_16941-01 - VAU-Protokoll: Client, Prüfung der
Signatur der VAUHelloDataVAU-Protokoll: Client, Prüfung der Signatur
der VAUHelloDataVAU-Protokoll: Client, Prüfung der Signatur der
VAUHelloData**

Der Client MUSS die Signatur der Daten von "Data" prüfen (bitgenau den Datenwert von
"Data" nehmen, ohne eine Base64-Dekodierung) der "Data"-Daten vorzunehmen. Der
Client MUSS dafür den Signaturschlüssel des Servers auf Authentizität und Integrität
prüfen. (Hinweis: in einem Client wird die TSL der TI als Prüfgrundlage für die Prüfung
von TI-Zertifikaten verwendet.) Falls die Signaturprüfung kein positives Ergebnis
erbringt, so MUSS der Client den Protokollablauf abbrechen (vgl. A_16849).
Der Client MUSS prüfen, ob der im VAUHelloData->CertificateHash aufgeführter
Hashwert mit dem Hashwert des Server-Zertifikats im VAUHello->Certificate-Feld
übereinstimmt (vgl. Erzeugung des Hashwerts in ~~ML-102746 - Missing cross reference~~).
A_16901-*). Falls nein, so MUSS der Client den Protokollablauf abbrechen
(vgl. A_16849). [$\leq \leq \leq$]

6.5 Schlüsselableitung

**A_16852-01A_16852-01A_16852-01 - VAU-Protokoll: ECDH durchführenVAU-
Protokoll: ECDH durchführenVAU-Protokoll: ECDH durchführen**

Der Client und auch der Server MÜSSEN jeweils für sich prüfen, ob der empfangene
ephemere öffentliche elliptische Kurvenpunkt der Gegenseite auch auf der von ihnen
verwendeten Kurve (BrainpoolP256r1) liegt.
Falls nein, MÜSSEN sie jeweils den Protokollablauf abbrechen. Falls der Server derjenige
ist, der in diesem Fall abbricht, MUSS der zuvor an den Client eine VAUServerError-
Nachricht mit der Fehlermeldung "invalid curve (ECDH)" senden.
Falls ja, MÜSSEN beide einen ECDH nach [NIST-800-56-A] durchführen. Das dabei
erzeugte gemeinsame Geheimnis ist folgend Grundlage von drei Schlüsselableitungen
(vgl. A_16943-01). [$\leq \leq \leq$]

**A_16943-01A_16943-01A_16943-01 - VAU-Protokoll: Schlüsselableitung
(HKDF)VAU-Protokoll: Schlüsselableitung (HKDF)VAU-Protokoll:
Schlüsselableitung (HKDF)**

Für die Schlüsselableitung MÜSSEN Client und Server die HKDF nach [RFC-5869] auf
Basis von SHA-256 verwenden.
Das "Input Keying Material" (IKM) [RFC-5869] ist das in A_16852-01 erzeugte
gemeinsame ECDH-Geheimnis zwischen Server und Client.
Die erste Schlüsselableitung hat den Ableitungsvektor "KeyID" ("info" Parameter aus
[RFC-5869] ist dann also "KeyID") und erzeugt einen 256 Bit langen Schlüsselidentifizier.
Die zweite Schlüsselableitung mit dem Ableitungsvektor "AES-256-GCM-Key-Client-to-
Server" erzeugt den 256-Bit AES-Schlüssel für die Verwendung innerhalb von AES-256-
GCM für Nachrichten, die der Client für den Server verschlüsselt.
Die dritte Schlüsselableitung mit dem Ableitungsvektor "AES-256-GCM-Key-Server-to-
Client" erzeugt den 256-Bit AES-Schlüssel für die Verwendung innerhalb von AES-256-
GCM für Nachrichten, die der Server für den Client verschlüsselt. [$\leq \leq \leq$]

6.6 VAUClientSigFin-Nachricht

A_17070-02A_17070-02A_17070-02 - VAU-Protokoll: Aufbau der VAUClientSigFin-Nachricht

Der Client MUSS auf eine VAUServerHello-Nachricht mit einer wie folgt definierten VAUClientSigFin-Nachricht antworten.

Die VAUClientSigFin-Nachricht hat folgenden Aufbau:

```
{
  "MessageType"      : "VAUClientSigFin",
  "VAUClientHelloDataHash" : "...SHA-256-Hashwert-der-Base64-kodierten-VAUClientHelloData...",
  "VAUServerHelloDataHash" : "...SHA-256-Hashwert-der-erhaltenen-Base64-kodierten-VAUServerHelloData...",
  "Signature"        : "...Base64-kodierte-Signatur...",
  "Certificate"       : "...Base64-kodiertes-Signaturzertifikat...",
  "OCSPResponse"      : "...Base64-kodierte-OCSP-Response-für-dieses-Zertifikat...",
  "FinishedData"      : "...Base64-kodierte-verschlüsselte-Finished-Daten ..."
```

Im "VAUClientHelloDataHash"-Feld MUSS der Client den Base64-kodierten Hashwert seiner Base64-kodierten VAUClientHelloData eintragen.

Im "VAUServerHelloDataHash"-Feld MUSS der Client den Base64-kodierten Hashwert der empfangenen Base64-kodierten VAUServerHelloData eintragen.

Die folgende Signatur MUSS der Client über die beiden konkatenierten Base64-kodierten Zeichenketten (Inhalt vom "VAUClientHelloDataHash"-Feld || Inhalt vom "VAUClientServerDataHash"-Feld) bilden.

Eine ECDSA-Signatur im "Signature"-Feld MUSS nach [TR-03111#5.2.2. X9.62 Format] ("ecdsa-with-Sha256") kodiert sein.

Diese so kodierte Signatur wird Base64-kodiert und als Wert im "Signature"-Feld eingetragen.

Eine RSASSA-PSS-Signatur MUSS nach [RFC-8017] (PKCS#1) kodiert werden. Diese so kodierte Signatur wird Base64-kodiert und als Wert im "Signature"-Feld eingetragen. (Verständnishinweis: Eine G2-Karte kann in Bezug auf AUT-Schlüssel nur RSA-Signaturen erzeugen. Eine G2.x-Karte kann und MUSS im Kontext VAU-Protokoll ECDSA-Signaturen erzeugen.)

Der Client MUSS im "Certificate"-Feld das für die Prüfung der Signatur notwendige X.509-EE-Zertifikat Base64-kodiert eintragen.

Er SOLL für dieses Zertifikat im "OCSPResponse"-Feld die OCSP-Response, die nicht älter als 24 Stunden sein darf, eintragen.

Falls ihm keine OCSP-Response zur Verfügung steht, so MUSS er im OCSPResponse-Feld den Leerstring als Wert eintragen ("OCSPResponse" : "").

Der Client MUSS für die Berechnung des "FinishedData"-Feldes zunächst folgende Zeichenkette bilden

```
"VAUClientSigFin" ||
unkodierter Hashwert aus "VAUClientHelloDataHash" ||
unkodierter Hashwert aus "VAUServerHelloDataHash"
```

Diese Zeichenkette MUSS 15+32+32=79 Bytes lang sein. Der Client MUSS diese

3154 6.7 VAUServerFin-Nachricht

3158 Der Server MUSS beim Empfang der VAUClientSigFin-Nachricht prüfen,

- Falls eine der Prüfungen 1 bis 3 ein nicht-positives Prüfergebnis liefert, so MUSS der Server mit einer VAUServerError-Nachricht antworten und die weitere Protokolldurchführung abbrechen. Wobei er folgende Fehlermeldung pro Prüfung verwenden MUSS:

- 3172 [<=<=<=]

3176 Der Server MUSS eine wie folgt aufgebaute VAUServerFin-Nachricht erzeugen.

Seite 88 von 122
Stand: 17.08.2020

3180 }
3181 Der Server MUSS für die Berechnung des "FinishedData"-Feldes zunächst folgende
3182 Zeichenkette bilden
3183 "VAUServerFin" ||
3184 unkodierter Hashwert aus "VAUClientHelloDataHash" ||
3185 unkodierter Hashwert aus "VAUServerHelloDataHash"
3186 Diese Zeichenkette MUSS 12+32+32=76 Bytes lang sein. Der Server MUSS diese
3187 Zeichenkette mittels AES-GCM (vgl. A_16943-01) verschlüsseln und dabei folgende
3188 Zeichenkette bilden
3189 256-Bit KeyID || 96-Bit Nonce (IV) mit Ciphertext und 128 Bit
3190 Authentication-Tag
3191 Diese Zeichenkette MUSS er Base64-kodieren und das Ergebnis als Wert
3192 des "FinishedData"-Feld eintragen. [$\leq \leq \leq$]
3193 **A_17073A_17073A_17073 - VAU-Protokoll: Versand der VAUServerFin-**
3194 **NachrichtVAU-Protokoll: Versand der VAUServerFin-NachrichtVAU-Protokoll:**
3195 **Versand der VAUServerFin-Nachricht**
3196 Der Server MUSS nach dem Erhalt einer VAUClientSigFin-Nachricht mit einer
3197 VAUServerFin-Nachricht antworten. Der Server MUSS die VAUServerFin-Nachricht per
3198 HTTP mit dem Content-Type 'application/json' an den Client senden. [$\leq \leq \leq$]
3199 **A_17084A_17084A_17084 - VAU-Protokoll: Empfang der VAUServerFin-**
3200 **NachrichtVAU-Protokoll: Empfang der VAUServerFin-NachrichtVAU-Protokoll:**
3201 **Empfang der VAUServerFin-Nachricht**
3202 Der Client MUSS beim Empfang der VAUServerFin-Nachricht prüfen, ob der Wert im
3203 "FinishedData"-Feld der nach A_16899 zu erwartenden Wert entspricht. Falls nein, so
3204 MUSS der Client den weiteren Protokollablauf abbrechen (vgl. A_16849). [$\leq \leq \leq$]

3205 6.8 Nutzerdatentransport

3206 **A_16945-02A_16945-02A_16945-02A_16945-01 - VAU-Protokoll: Client,**
3207 **verschlüsselte Kommunikation (1)VAU-Protokoll: Client, verschlüsselte**
3208 **Kommunikation (1)VAU-Protokoll: Client, verschlüsselte Kommunikation (1)**
3209 Wie bei der Schlüsselaushandlung MUSS der Client mittels HTTP-POST-Request die nun
3210 verschlüsselte Kommunikation initiieren.
3211 Der Client MUSS einen unsigned 64-Bit-Nachrichtenzähler führen, die er bei jeder
3212 abgeschickten Nachricht um zwei erhöhen MUSS.
3213
3214 Er bildet die Datenstruktur "P1" mit
3215 P1=Version (ein Byte mit dem Wert 0x01) ||
3216 Nachrichtenzähler (unsigned 64-Bit im Big-Endian-Format) ||
3217 Anzahl der Bytes der folgenden ~~optionalen~~ zusätzlichen HTTP-
3218 Header-Informationen (unsigned 32-Bit im Big-Endian-Format) ||
3219 ~~optionale~~ zusätzliche HTTP-Header-Informationen ||
3220 Plaintext
3221 wobei „Plaintext“ die zu übertragende Nutzlast (bspw. SOAP-Request) bezeichnet.
3222
3223 Wenn die Anzahl der Bytes der folgenden ~~optionalen~~ zusätzlichen HTTP-Header-
3224 Informationen mit 0 (also 0x00000000) angegeben wird, so gibt es keine folgenden
3225 ~~optionalen~~ zusätzlichen HTTP-Header-Informationen, d. h. es folgen direkt die Plaintext-
3226 Bytes.
3227

3228 Der Nachrichtenzähler MUSS initial mit 1 starten.
3229 Der Client MUSS zunächst einen IV wie folgt erzeugen:
3230 1. Sei a ein zufällig erzeugtes 32-Bitfeld.
3231 2. Sei IV=a (32 Bit) || Nachrichtenzähler (unsigned 64-Bit im Big-Endian-Format).
3232 Damit ist der IV 96-Bit lang. Unter Verwendung dieses IV und des zweiten aus A_16943-
3233 01 abgeleiteten Schlüssel (Client-to-Server-Schlüssel) wird P1 verschlüsselt. Der Client
3234 berechnet so den "Ciphertext".
3235 Die vom Client nun an den Server zu übermittelnde Datenstruktur MUSS folgende Form
3236 besitzen.
3237 256-Bit KeyID || 96-Bit IV mit Ciphertext und 128 Bit Authentication-Tag
3238
3239 Diese Nachricht MUSS der Client per HTTP-POST-Request mit Content-Type
3240 'application/octet-stream' ohne weitere Kodierungen versenden. [~~==~~ <= <= <=]
3241 **A_16952-02A_16952-02A_16952-02A_16952-01 - VAU-Protokoll: Server,**
3242 **verschlüsselte KommunikationVAU-Protokoll: Server, verschlüsselte**
3243 **KommunikationVAU-Protokoll: Server, verschlüsselte Kommunikation**
3244 Der Server erkennt aus der KeyID, welchen AES-Schlüssel er für die Entschlüsselung
3245 verwenden muss (vgl. A_16943-01 zweiter abgeleiteter Schlüssel (Client-to-Server-
3246 Schlüssel)).
3247 Falls ihm die KeyID unbekannt ist, so MUSS er mit einer VAUServerError-Nachricht mit
3248 der Fehlermeldung "KeyID XXX not found" antworten, wobei er XXX durch die
3249 empfangene KeyID in Hexadezimalform ersetzen MUSS.
3250 Falls bei der Entschlüsselung ein Fehler auftritt (bspw. Authentication-Tag passt nicht zur
3251 Nachricht), MUSS der Server mit einer VAUServerError-Nachricht mit der Fehlermeldung
3252 "AES-GCM decryption error." antworten.
3253
3254 Falls die Entschlüsselung erfolgreich war, MUSS den Klartext gemäß der Struktur von P1
3255 aus A_16945-~~01~~02 interpretieren.
3256
3257 Falls die Version in P1 ungleich 0x01 ist, so MUSS der Server (1) eine VAUServerError-
3258 Nachricht gemäß A_16851-01 mit der Fehlermeldung "invalid protocol version" senden
3259 und gemäß A_16849 die Protokollausführung abbrechen.
3260
3261 Sei mit "Server-Zählerwert" der letzte vom Server für den Nachrichtenversand
3262 verwendete Zählerwert bezeichnet. Initial (d. h. es wurde innerhalb eines
3263 Protokollablaufs noch nie eine Nachricht vom Server versendet) MUSS dieser Server-
3264 Zählerwert gleich 0 sein.
3265 Der Server MUSS prüfen, ob der Zählerwert im Klartext größer als der "Server-
3266 Zählerwert" ist. Falls nein, so MUSS der Server (1) eine VAUServerError-Nachricht gemäß
3267 A_16851-01 mit der Fehlermeldung "invalid counter value" senden und (2) gemäß
3268 A_16849 die Protokollausführung abbrechen.
3269 Der Server MUSS den "Server-Zählerwert" auf Zählerwert + 1 setzen.
3270 Falls es dabei zu einem Zählerüberlauf kommt, so MUSS der Server (1) eine
3271 VAUServerError-Nachricht gemäß A_16851-01 mit der Fehlermeldung "message counter
3272 overflow" senden und (2) gemäß A_16849 die Protokollausführung abbrechen.
3273 Der Server MUSS ~~optionale~~ zusätzliche HTTP-Header-Informationen analog zu A_16945-
3274 ~~01~~02 interpretieren und verwenden. Falls dies nicht möglich ist (bspw. Längenwert ist
3275 größer als eigentliche Nachrichtengröße), so MUSS der Server (1) eine VAUServerError-
3276 Nachricht gemäß A_16851-01 mit der Fehlermeldung "HTTP additional header length
3277 error" senden und (2) gemäß A_16849 die Protokollausführung abbrechen.
3278

Der Server erzeugt zunächst die Datenstruktur "P2" mit
P2 = Version (ein Byte mit dem Wert 0x01) ||
Server-Zählerwert (unsigned 64-Bit, big-endian-format) ||
Anzahl der Bytes der folgenden ~~optionalen~~ zusätzlichen HTTP-Header-
Informationen (unsigned 32-Bit im Big-Endian-Format) ||
~~optionale zusätzliche~~ HTTP-Header-Informationen ||
Klartext-Antwort des Servers

Der Server MUSS P2 mit AES-256-GCM verschlüsseln. Dafür MUSS der Server zufällig eine 96-Bit-großen IV wie folgt erzeugen:

1. Sei a ein zufällig erzeugtes 32-Bitfeld.
2. Sei IV=a (32 Bit) || Server-Nachrichtenzähler (unsigned 64-Bit im Big-~~End~~
~~ian~~Endian-Format).

Damit ist der IV 96-Bit lang. Unter Verwendung dieses IV und des dritten aus A_16943-01 abgeleiteten Schlüssel (Server-to-Client-Schlüssel) MUSS der Server P2 verschlüsseln.

Die zu übermittelnde Datenstruktur MUSS folgende Form besitzen
256-Bit KeyID || 96-Bit IV mit Ciphertext und 128 Bit Authentication-Tag

Diese Datenstruktur MUSS der Server per HTTP-Response mit Content-Type 'application/octet-stream' ohne weitere Kodierungen versenden. [~~=~~ ~~=~~ ~~=~~]

A_16957-01A_16957-01A_16957-01A_16957 - VAU-Protokoll: Client, verschlüsselte Kommunikation (2)
VAU-Protokoll: Client, verschlüsselte Kommunikation (2)
VAU-Protokoll: Client, verschlüsselte Kommunikation (2)

Beim Empfang der Antwort (vgl. A_16952-01-*) MUSS der Client folgende Vorgaben durchsetzen.
Falls

1. er die KeyID nicht kennt,
2. die Entschlüsselung fehlschlägt (bspw. Authentication-Tag passt nicht zur Nachricht), oder
3. der 64-Bit Zählerwert ungleich 1 plus dem Zählerwert ist, den der Client für den Request verwendet hat,

so MUSS der Client die Nachricht verwerfen und die weitere Protokollausführung mittels des empfangenen KeyID abbrechen.

Anderen falls (alles ok, kein Abbruch) MUSS der Client den mit der KeyID verbundenen Zählerwert um eins erhöhen. D. h., Nachrichten vom Client an den Server haben immer einen ungeraden Zählerwert. [~~=~~ ~~=~~ ~~=~~]

A_16958A_16958A_16958 - VAU-Protokoll: Client, Neuintiieren einer Schlüsselaushandlung
VAU-Protokoll: Client, Neuintiieren einer Schlüsselaushandlung
VAU-Protokoll: Client, Neuintiieren einer Schlüsselaushandlung

Der Client KANN jeder Zeit eine neue Schlüsselaushandlung (VAUClientHello etc.) initiieren. [~~=~~ ~~=~~ ~~=~~]

A_17069A 17069A 17069 - VAU-Protokoll: Client ZählerüberlaufVAU-Protokoll: Client ZählerüberlaufVAU-Protokoll: Client Zählerüberlauf
Der Client MUSS, falls so viele Nachrichten ausgetauscht werden, dass für den unsigned 64-Bit-Nachrichtenzähler ein arithmetischer Überlauf droht, eine neue Schlüsselaushandlung initiieren (VAUClientHello etc.).[<=<=<=]

6.9 VAUServerError-Nachricht

A_16851-01A 16851-01A 16851-01 - VAU-Protokoll: VAUServerError-NachrichtenVAU-Protokoll: VAUServerError-NachrichtenVAU-Protokoll: VAUServerError-Nachrichten

Der Server MUSS folgende Vorgaben umsetzen:
In verschiedenen im Protokoll beschriebenen Fehlerfällen sendet der Server eine VAUServerError-Nachricht an den Client.

Für die eigentliche Fehlerübermittlung MUSS folgende Datenstruktur erzeugt:

```
{  
  "DataType" : "VAUServerErrorData",  
  "Data"      : "...Fehlermeldung...",  
  "Time"      : "...aktuelle-Zeit-in-der-VAU..."  
}
```

Die Zeit im "Time"-Feld MUSS im Format nach ISO-8601 kodiert werden (Beispiel: "2018-11-22T10:00:00.123456").

Diese Datenstruktur MUSS der Server Base64-kodieren und in der folgenden Nachricht im Datenfeld "Data" einbetten.

```
{  
  "MessageType" : "VAUServerError",  
  "Data"         : "...Base64-kodierte-VAUServerErrorData..",  
  "Signature"    : "...Base64-kodierte-ECDSA-Signatur...",  
  "Certificate"  : "...Base64-kodiertes-Signaturzertifikat...",  
  "OCSPResponse" : "...Base64-kodierte-OCSP-Response-für-dieses-Zertifikat..."  
}
```

Die ECDSA-Signatur im "Signature"-Feld MUSS nach [TR-03111#5.2.2. X9.62 Format] ("ecdsa-with-Sha256") kodiert sein.

Im "Certificate"-Feld MUSS der Server, das verwendete Signaturzertifikat aufführen, und im "OCSPResponse"-Feld eine OCSP-Response für diese Zertifikat, welche nicht älter als 24 Stunden ist.[<=<=<=]

A_16900A 16900A 16900 - VAU-Protokoll: Client, Behandlung von FehlernachrichtenVAU-Protokoll: Client, Behandlung von FehlernachrichtenVAU-Protokoll: Client, Behandlung von Fehlernachrichten

Erhält der Client eine VAUServerError-Nachricht (vgl. A_16851-01), MUSS er die Signatur prüfen. Falls die Prüfung positiv ist, so MUSS er die Protolldurchführung abbrechen (vgl. A_16849).[<=<=<=]

6.10 Abbrechen des Protokollablaufs

A_16849A_16849A_16849 - VAU-Protokoll: Aktionen bei ProtokollabbruchVAU-Protokoll: Aktionen bei Protokollabbruch

Wenn ein Client oder ein Server den Protokollablauf nach Protokollbeschreibung abbrechen muss, dann MUSS dieser die eventuell aktuell vorhandene KeyID aus seiner Datenbasis löschen und die damit verbundenen Schlüssel sicher löschen.

[<=<=<=<=]

6.11 VAU-Kanal und MTOM/XOP

Nachdem die Etablierung des VAU-Kanals abgeschlossen ist, kommuniziert ein VAU-Client (bspw. ein ePA-FdV) mit der VAU bspw. um einen großen verschlüsselten Arztbrief der Akte hinzuzufügen. Dabei ist es für die Performanz (also auch für die Nutzerakzeptanz) günstig, größere Daten mittels der „SOAP Message Transmission Optimization Mechanism (MTOM)“/ „XML-binary Optimized Packaging (XOP)“-Kodierung zu transportieren. MTOM/XOP ist die nach W3C empfohlene Methode, um über HTTP/SOAP größere binäre Daten zu transportieren. Dabei werden die Binärdaten nicht Base64- innerhalb einer XML-Datenstruktur kodiert, sondern beim HTTP/SOAP-Request (bzw. bei der -Response) über eine MIME-Multipart-Kodierung innerhalb von HTTP. Dabei werden innerhalb der SOAP-Nachricht die Binärdaten über eine ID (<xop:Include href="cid:Beispiel-ID1"/>) referenziert und innerhalb des HTTP-Request bzw. der HTTP-Response über die „Content-ID“ (vgl. das folgende Beispiel) und eine MIME-Kodierung binär kodiert. Dies führt zur Reduktion Datengröße der zu übertragenden Daten von rund 27,5%.

Beispiel:

Ohne MTOM/XOP:

```
POST /URL1 HTTP/1.1
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: binary
Content-Length: ...

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header />
  <SOAP-ENV:Body>
    <ns1:uploadFileRequest xmlns:ns1="urn:example:Upload">
      <ns1:name>Test-Bild.png</ns1:name>
      <ns1:content> ... Hier kommen in Base64 kodierte Daten ... </ns1:content>
    </ns1:uploadFileRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Mit MTOM/XOP:

```
POST /URL1 HTTP/1.1
Content-Type: Multipart/Related; boundary="-----_Part_1_1234-----";
type="application/xop+xml"; start="Eintrag-1"; start-info="application/soap+xml";
charset=UTF-8
Content-Transfer-Encoding: binary
Content-Length: ...

-----_Part_1_1234-----
```

Für die Dekodierung einer solchen MIME-Multipart-Kodierung ist es sehr hilfreich (und ggf. notwendig) als Empfänger den vom Sender intendierten (vollständigen) „Content-Type“ aus dem HTTP-Request-Header bzw. dem HTTP-Response-Header zu kennen. Dieser wird jedoch durch das VAU-Protokoll überschrieben (vgl. A_16945-01-* und A_16952-01-*). Um die u. a. nach [gemSpec_FM_ePA#A_16220] und [gemSpec_ePA_FdV#A_16222] geforderte Verwendung von MTOM/XOP zu unterstützen, wird der originär intendierte Content-Type, der im „start“-Feld ggf. vertrauliche Daten enthalten kann, innerhalb der „optionalen“ zusätzlichen HTTP-Header-Informationen“ (vgl. A_16945-01-* und A_16952-01-*) mitgeliefert.

3451 Wenn ein VAU-Client oder ein VAU-Server Übertragungen mittels MTOM/XOP
3452 durchführen, so MÜSSEN sie die durch MTOM/XOP erzeugten „Content-Type“-
3453 Informationen (vgl. [gemSpec_Krypt#6.11 VAU-Kanal und MTOM/XOP]) innerhalb ihrer
3454 Nachricht als „zusätzliche HTTP-Header-Informationen“ gemäß A_16945-01-* und
3455 A_16952-01-* aufführen.[<=<=<=<=]

6.136.12 A_18466—VAU-Protokoll: zusätzliche optionale HTTP-Header-Informationen

Ein VAU-Client oder ein VAU-Server **MÜSSEN** in den „zusätzliche HTTP-Header-Informationen“ gemäß A 16945-* und A 16952-*, die ursprünglich intendierten Content-Type HTTP-Header-Variable einbringen. D. h. Ein Kommunikationspartner erstellt einen HTTP-Request oder eine HTTP-Response. Diese hat eine "ursprünglich intendierten Content-Type", bspw. "application/soap+xml; charset=utf-8". Diese **MUSS** dann beim der Verschlüsselung der Nachricht mittels des VAU-Protokolls in die "zusätzlichen HTTP-Header-Informationen" eingetragen werden. [$\leq \leq \leq$]

Wenn ein VAU-Client oder ein VAU-Server Übertragungen durchführen und in den empfangenen Nachrichten sind „zusätzliche **optionale** HTTP-Header-Informationen“ gemäß A_16945-01-* und A_16952-01-*, so MÜSSEN sie diese auswerten (Hinweis: insbesondere „Content-Type“-Informationen (vgl. [gemSpec_Krypt#6.11 VAU-Kanal und MTOM/XOP])).

Dabei "überschreiben" diese zusätzlichen HTTP-Header-Information HTTP-Header-Variablen im "äußeren" HTTP-Request (insbesondere "Content-Type") bei der Interpretation der Informationen aus dem Request.

7 Kommunikationsprotokoll zwischen E-Rezept-VAU und E-Rezept-Clients

7.1 Übersicht (informativ)

Ähnlich wie bei der Anwendung ePA besitzt der Fachdienst E-Rezept zwei HTTP-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 7.2 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 unterschützt 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 Chifftrat 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). 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 Chifftrat 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 Chifftrat 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.

3524 7. Die Webschnittstelle antwortet gegenüber dem Client mit diesem Chiffprat auf den
3525 HTTP-Request.

3526 8. Der Client entschlüsselt das Chiffprat und prüft die Request-ID in der
3527 entschlüsselten Antwort.

3528 Das Protokoll ist statuslos. Zwischen verschiedenen VAU-Instanzen (bspw. VAU-
3529 Instanzen in unterschiedlichen Brandabschnitten eines Rechenzentrum) müssen keine
3530 Session-Schlüssel oder ähnliches synchronisiert werden.

3531 Die Struktur der inneren HTTP-Request ist so einfach, dass davon auszugehen ist, dass in
3532 der VAU-Instanz keine umfangreichen Webserver oder HTTP-Bibliotheken notwendig
3533 sind. Ziel der E-Rezept-VAU ist es die Code-Komplexität (Trusted Computing Base, TCB)
3534 so weit es nur irgendwie geht minimal zu halten.

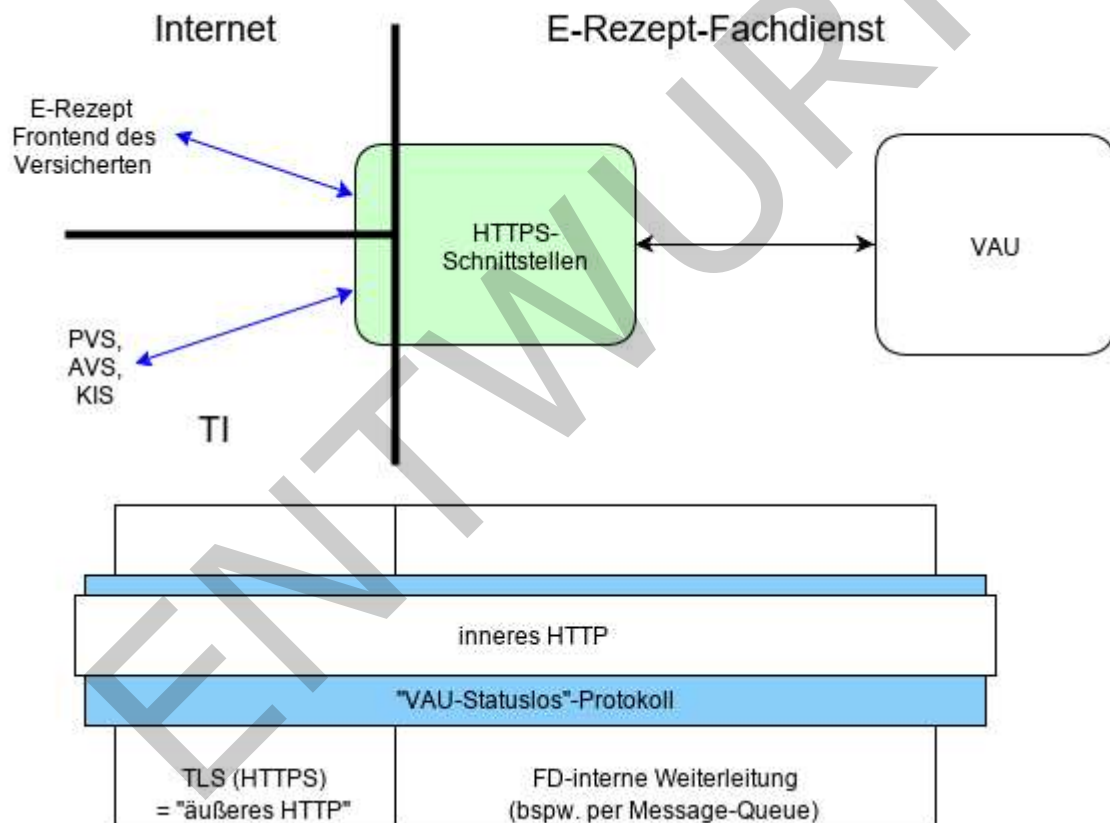


Abbildung 4: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU

Die gematik stellt eine Beispielimplementierung bereit.

7.2 Definition

A_20160A_20160A_20160 - E-Rezept-VAU, Schlüsselpaar und ZertifikatE-Rezept-VAU, Schlüsselpaar und Zertifikat

Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

1. Die VAU MUSS ein EE-X.509-Zertifikat aus der Komponenten-PKI der TI besitzen (mit Rollenkennung-OID "oid_erp_vau"), das einen ECC-EE-Schlüssel der VAU bestätigt.
2. Die VAU MUSS die Vertraulichkeit des privaten Schlüssels für diese Zertifikat sicherstellen.
3. Die notwendige Sicherung (Backup) und Verteilung dieses privaten Schlüssels MUSS ausschließlich im Mehr-Augen-Prinzip und mit geeigneten Maßnahmen zur Wahrung der Vertraulichkeit des Schlüssels geschehen.
4. Der Fachdienst E-Rezept MUSS das VAU-Zertifikat in seinen Webschnittstellen unter dem Pfad /VAUCertificate (einer URL) durch Clients abrufbar machen. Dieses Zertifikat MUSS DER-kodiert sein.
5. Der Fachdienst E-Rezept MUSS eine maximal 12 Stunden alte OCSP-Response für das VAU-Zertifikat in seinen Webschnittstellen unter dem Pfad /VAUCertificateOCSPResponse für Clients abrufbar machen.

[<=<=<=]

A_20161-01A_20161-01A_20161-01A_20161 - E-Rezept-Client, Request-ErstellungE-Rezept-Client, Request-ErstellungE-Rezept-Client, Request-Erstellung

~~Ein~~Es E-Rezept-Client MUSS ~~bei der Request-Erstellung folgende Schritte durchführen.~~

~~Falls im E-Rezept-Client, falls ihm~~ noch kein gültiges E-Rezept-VAU-Zertifikat vorliegt, ~~so MUSS der E-Rezept-Client ein solches nach den fachlichen Vorgaben von A_20160 beziehen (/VAUCertificate) und ggf. prüfen ("gute").~~
~~Ein E-Rezept-Client MUSS sicherstellen, dass gültige Sperrinformation (OCSP-Response mit Sperrstatus "good") für das Zertifikat istvorliegen, die maximal 12 Stunden alt) sind. Liegen diese nicht vor so MUSS der Client ein Verbindungsaufbau auf VAU-Protokoll-Ebene ablehnen/unterbinden.~~

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

1. Er erzeugt einen HTTP-Request, den er an die VAU senden möchte, als Datenstruktur (vgl. Beispiele nach dieser Anforderung).
2. Er erzeugt zufällig eine 128-Bit lange hexadezimalkodierte Request-ID (also 32 Zeichen, Buchstaben a-f kleingeschrieben).
3. Er erzeugt zufällig einen 128-Bit AES-Schlüssel (im Weiteren auch Antwortschlüssel genannt), den er hexadezimal kodiert (also 32 Zeichen, Buchstaben a-f kleingeschrieben).
4. Er MUSS die Request-ID und den AES-Schlüssel für jeden HTTP-Request an die VAU zufällig neu wählenerzeugen.
5. Er erzeugt die folgende Zeichenkette p mit
p="1" + " " + JWT-Authentisierungstoken + " " + Request-ID + " " + AES-Schlüssel + " " + Datenstruktur aus Schritt 1.

6. Die Zeichenkette p MUSS mittels des ECIES-Verfahrens [SEC1-2009] und eines AES-128-Bit Content-Encryption-Key und AES/GCM verschlüsselt werden. Dabei ist ein zufällig erzeugte 96 Bit langer IV und ein 128 Bit langer Authentication-Tag zu verwenden. Das Ergebnis sei mit Chiffre c bezeichnet (IV || Ciphertext || Authentication-Tag), mit folgenden Vorgaben verschlüsselt werden:
- a. Er MUSS ein ephemeres ECDH-Schlüsselpaar erzeugen und mit diesem und dem VAU-Schlüssel aus A 20160-* ein ECDH gemäß [NIST-800-56-A] durchführen. Das somit erzeugte gemeinsame Geheimnis ist Grundlage für die folgende Schlüsselableitung.
 - b. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von SHA-256 verwenden.
 - c. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in der Formulierung von [RFC-5869] info="ecies-vau-transport".
 - d. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key für die Verwendung von AES/GCM ableiten.
 - e. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig erzeugen.
 - f. Er MUSS mit dem CEK und dem IV mittels AES/GCM p verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
 - g. Er MUSS das Ergebnis wie folgt kodieren: chr(0x01) || <32 Byte X-Koordinate von öffentlichen Schlüssel aus (a)> || <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.
7. Er erzeugt einen HTTPS-Request an den FD mit der POST-Methode und dem Pfad /VAU/<Nutzerpseudonym>[/optional-beliebiger-weiterer-URL-Pfadteil] mit dem Content-Type 'application/octet-stream' und sendet diesen an die Webschnittstelle des FD.
"Nutzerpseudonym" MUSS eine ggf. aus der vorherigen (zeitlich letzten) Antwort des FD dem Nutzer übergebene URL-sichere Zeichenkette sein (bspw. ein 128 Byte langer Hexadezimal-Kode).
Falls dem Client kein Nutzerpseudonym vorliegt so MUSS er "0" als Nutzerpseudonym verwenden.

[<=<=<=]

Tabelle 17: Tab KRYPT ERP Kodierung des Chiffre aus A 20161-*

Offset	Länge in Bytes	Erläuterung
<u>0</u>	<u>1</u>	<u>Versionsfeld</u>
<u>1</u>	<u>32</u>	<u>X-Koordinate öffentlicher ephemere Sender-ECDH-Schlüssel</u>

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-Chiffre
77 + LP	16	128 Bit langer Authentication-Tag

[Hinweise zu A_20161-*:](#)

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

[Beispielverschlüsselung \(Testvektor\):](#)

[Der Langzeit-VAU-Schlüssel nach A_20160-* sei folgender:](#)

[x=0x8634212830dad457ca05305e6687134166b9c21a65ffebf555f4e75dfb048888](#)
[y=0x66e4b6843624cbda43c97ea89968bc41fd53576f82c03efa7d601b9facac2b29](#)

[Die zu Verschlüsselnde Nachricht sei "Hallo Test" \(10 Zeichen\).](#)

[Der Client erzeugt zufällig den privaten ECC-Schlüssel](#)

[d=5bbba34d47502bd588ed680dfa2309ca375eb7a35ddb6d67cc7f8b6b687a1c1d](#)

[Damit ist dessen öffentlicher ephemerer ECDH-Schlüssel:](#)

[x=0x754e548941e5cd073fed6d734578a484be9f0bbfa1b6fa3168ed7ffb22878f0f](#)
[y=0x9aef9bbd932a020d8828367bd080a3e72b36c41ee40c87253f9b1b0beb8371bf](#)

[Nach ECDH ergibt sich damit folgendes gemeinsames Geheimnis:](#)

[9656c2b4b3da81d0385f6a1ee60e93b91828fd90231c923d53ce7bbbcd58ceaa](#)

[Nach Schlüsselableitung \(info="ecies-vau-transport"\) erhält der Client folgende CEK:](#)

[23977ba552a21363916af9b5147c83d4](#)

[Der Client erzeugt zufällig einen 12 Byte großen IV:](#)

[257db4604af8ae0dfced37ce](#)

[Die AES/GCM-Chiffre berechnet aus der Nachricht, dem CEK und dem IV folgendes Chiffre:](#)

[86c2b491c7a8309e750b](#) und folgenden 16 Byte großen Authentication
Tag [4e6e307219863938c204dfe85502ee0a](#)

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

01 754e548941e5cd073fed6d734578a484be9f0bbfa1b6fa3168ed7ffb22878f0f
9aef9bbd932a020d8828367bd080a3e72b36c41ee40c87253f9b1b0beb8371bf
257db4604af8ae0dfced37ce 86c2b491c7a8309e750b
4e6e307219863938c204dfe85502ee0a

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

A_20162A_20162A_20162 - E-Rezept-FD, Webschnittstellen, VAU-RequestsE-Rezept-FD, Webschnittstellen, VAU-Requests

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

1. Er MUSS unter dem Pfad /VAU/<Nutzerpseudonym>[/optional-beliebiger-weiterer-URL-Pfadteil] (der URL) mit dem Content-Type 'application/octet-stream' HTTPS-Requests entgegennehmen (nach dem Nutzerpseudonym kann u. Um. ein "/" und anschließend weitere Pfadangaben vom Client angegeben werden, vgl. Beispiele unten).
2. Er MUSS in der Webschnittstelle über einen AES-CMAC 128 Bit Schlüssel verfügen, der mindestens alle 10 Tage zufällig neu erzeugt wird. Dieser Schlüssel MUSS als reiner Software-Schlüssel (nicht in einem HSM) in der Webschnittstelle vorliegen.
3. Er MUSS das Nutzerpseudonym (NP) auf Integrität (CMAC) prüfen (vgl. Schritt 8). Ist die Integrität nicht gegeben, so MUSS er anstatt des übergebenen NP "0" als Wert verwenden.
4. Er MUSS anhand des NP eine Lastverteilung innerhalb des FD und eine NP-zentrierte Lastbeschränkung durchführen.
Im Lastszenario MÜSSEN Requests mit NP "0" mindestens 10 mal geringer priorisiert werden, als Requests mit gültigem NP.
5. Er muss den Request zur Abarbeitung an einen geeigneten Verarbeitungskontext der VAU übergeben.
6. Die VAU-Instanz muss eine verschlüsselte Antwort (vgl. A_20163) erzeugen und an die Schnittstelle senden.
7. In der Antwort der VAU-Instanz MUSS die VAU das Prenutzerpseudonym (PNP, vgl. A_20163) als Teil der Antwort der VAU auf den Nutzer-Request an die Webschnittstelle übergeben.
8. Er MUSS mittels des CMAC-Schlüssels (vgl. Schritt 2) den 128-Bit-lange CMAC-Wert des PNP erzeugen und diesen hexadezimal kodieren (= CMAC). Die Zeichenkette "<PNP>" + "-" + "<CMAC>" sei das NP.
9. Als Antwort MUSS die Schnittstelle eine HTTP-Response senden mit dem Content-Type 'application/octet-stream', der Antwort der VAU-Instanz als Bytestrom (Octet-Stream) und im HTTP-Response-Header MUSS 'Userpseudonym: <NP>' enthalten sein.

3740 1. Er MUSS prüfen, ob der Content-Type der Response 'application/octet-stream' ist.

3741 2. Wenn im Response-Header die Variable "Userpseudonym" vorhanden ist, so MUSS

3742 er den Wert von "Userpseudonym" als NP für den nächsten Request an die VAU

3768

8 Erläuterungen (informativ)

3769

8.1 Prüfung auf angreifbare (schwache) Schlüssel

3770

Im Folgerelease wird es in diesem Abschnitt Hinweise für die Anforderungen aus Abschnitt 2.4.1 geben.

3771

3772

8.2 RSA-Schlüssel in X.509-Zertifikaten

3773

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.

3774

3775

3776

3777

3778

3779

3780

3781

3782

3783

3784

3785

3786

3787

3788

3789

3790

3791

3792

keine kleinen Primteiler:

3793

3794

3795

3796

3797

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.

3798

3799

3800

3801

3802

3803

3804

3805

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.

3806 In der gematik wurden mehr als eine Billion (10^{12}) RSA-Schlüssel zufällig erzeugt und
3807 verschiedene Fehlkodierungen dieser Schlüssel auf Primteiler kleiner 100 untersucht. Es
3808 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 %

3809 Man muss davon ausgehen, dass die entsprechende Fehlkodierung nicht nur bei einem
3810 einzigen RSA-Schlüssel, sondern bei allen RSA-Schlüsseln eines
3811 Personalisierungsauftrags auftritt. Somit nähert sich die Erkennungsrate exponentiell
3812 100 % an. Beispiel: bei einer Erkennungsrate von 75 % für eine bestimmte Art der
3813 Fehlkodierung (vgl. Tabelle) erhält man bei 1000 RSA-Schlüsseln in Gesamtheit eine
3814 Erkennungsrate von mehr als $1 - 1,15 \cdot 10^{-125}$, also nahe 1.

3815 öffentlicher Exponent ist prim:

3816 Sei e der öffentliche Exponent und n der Modulus eines RSA-Schlüssels. Bei der Wahl von
3817 e ist es notwendig, dass dieser relativ prim zu $\phi(n)$ ist. Um die Schlüsselerzeugung zu
3818 vereinfachen (und zu beschleunigen), wählen jedoch faktisch alle kryptographischen
3819 Softwarebibliotheken, Chipkarten und HSMs e prim. Wenn also dem TSP ein e vorliegt,
3820 das nicht prim ist, so kann er davon ausgehen, dass ein Fehler vorliegt.

3821 Diese Überlegungen führen zu den Tests in A_17092. Diese Tests haben eine Falsch-
3822 Positiv-Rate von 0 und benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

3823 Entropie der Schlüsselkodierung:

3824 Der Modulus eines RSA-Schlüssels muss aus genau zwei (oder wenigen [RFC-8017])
3825 "großen" zufällig gewählten Primfaktoren bestehen. Diese zufällige Wahl hat zur Folge,
3826 dass die Entropie der kodierten Schlüsselwerte eine hohe Entropie im Sinne von
3827 notwendigen Bits pro Byte besitzt. Eine Fehlkodierung wird evtl. weniger Entropie (Bits
3828 pro Byte) besitzen, weil sie, wie die base64-Kodierung, bestimmte Bits immer auf 0
3829 setzt. In [NIST-SP-800-22] werden verschiedene Tests spezifiziert, um die "Zufälligkeit"
3830 einer Zeichenfolge zu bestimmen. Diese Tests zielen auf größere Datengrößen (Längen
3831 der Zeichenfolge) ab und sind nur teilweise dafür geeignet die Kodierung von 256 Byte
3832 langen RSA-Moduli zu bewerten. Anstatt diese Tests als Basis zu verwenden, wird im
3833 Folgenden die klassische Berechnung der Shannon-Entropie vieler RSA-Moduli in
3834 unterschiedlichen Kodierungsformen betrachtet. Von einer Zeichenkette X wird mittels

$$H(X) = - \sum_{i=0}^{255} p_i \log p_i$$

3835

3836 die Entropie im Sinne von notwendigen Bits pro Byte des kodierten Schlüsselwertes
3837 berechnet. Dabei ist X die Kodierung (Bytefolge) des Schlüssels und p_i die relative
3838 Häufigkeit von Byte i (wobei nach Konvention in dem Kontext gilt: $\log(0)=0$).

3839 Unter <https://rosettacode.org/wiki/Entropy> findet man Implementierung dieser Entropie-
3840 Berechnungsfunktion in 78 Programmiersprachen.

3841 Die gematik verwendet folgende C-Implementierung:

```
3842 double entropy(char *S, int len) {
3843     int table[256]={0};
3844     int i;
3845     double H=0;
3846
3847     for(i=0; i<len; i++) {
3848         table[(unsigned char)S[i]]++;
3849     }
3850     for(i=0; i<256; i++) {
3851         if (table[i]>0) {
3852             H-= (double) table[i]/len*log2((double) table[i]/len);
3853         }
3854     }
3855     return H;
3856 }
```

3857 In der gematik wurden mehr als eine Billion (10^{12}) RSA-Schlüssel zufällig erzeugt und
3858 verschiedene Fehlkodierungen auf ihre Entropie (notwendigen Bits pro Byte in der
3859 Kodierung) untersucht.

3860 Ein Histogramm eines Beispiel-Testlaufs mit mehr als eine Billion (10^{12}) RSA-
3861 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

3862 Es wurde kein Schlüssel gefunden, dessen korrekte Kodierung eine Entropie kleiner als
3863 6,7295 besitzt.

3864 Ein Histogramm eines Beispiel-Testlaufs mit mehr 10 Milliarden ($1 \cdot 10^{10}$) Schlüsseln
3865 (diese wurden jeweils in unterschiedlichen Kodierungsvarianten kodiert und danach
3866 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			

3867 Diese Überlegungen bezüglich der Entropie der RSA-Schlüsselkodierung führen zu dem
3868 Test in A_17093. Dieser Test hat eine Falsch-Positiv-Rate von weniger als 2^{-40} und
3869 benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

3870

9 Anhang – Verzeichnisse

3871

9.1 Abkürzungen

Kürzel	Erläuterung
aAdG-NetG	Andere Anwendungen des Gesundheitswesens ohne Zugriff auf Dienste der TI in angeschlossenen Netzen des Gesundheitswesens
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

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_Dokumentenverwaltung]

3872 **9.2 Glossar**

3873 Das Glossar wird als eigenständiges Dokument, vgl. [gemGlossar] zur Verfügung gestellt.

9.3 Abbildungsverzeichnis

Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht	2525
Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält	7270
Abbildung 3: Übersicht über das VAU-Protokoll	8279
Abbildung 4: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU	9793
Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht	2525
Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält	7270
Abbildung 3: Übersicht über das VAU-Protokoll	8279
Abbildung 4: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU	9793

9.4 Tabellenverzeichnis

Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten	1313
Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „RSA“	1414
Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	1515
Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter elektronischer Signaturen für die Schlüsselgeneration „RSA“	1616
Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	1717
Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate	1919
Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate	2019
Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs	2626
Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten elektronischen XML-Signaturen	2727
Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen	2828
Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung	3131
Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC	4241
Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versicherten individuellen Schlüssels	4342
Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines versicherten individuellen Schlüssels	4443
Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären Daten im Kontext von Dokumentensignaturen	4645

Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-Dokumentensignaturen	4746
Tabelle 17: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept)	10399
Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten	1313
Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „RSA“	1414
Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	1515
Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter elektronischer Signaturen für die Schlüsselgeneration „RSA“	1616
Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“	1717
Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate.....	1919
Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate.....	2019
Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs	2626
Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten elektronischen XML-Signaturen	2727
Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen	2828
Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung	3131
Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC.....	4241
Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen Schlüssels	4342
Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines versichertenindividuellen Schlüssels.....	4443
Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären Daten im Kontext von Dokumentensignaturen	4645
Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-Dokumentensignaturen	4746
Tabelle 17: Tab_KRYPT_ERP Kodierung des Chiffrats aus A 20161-*	9995
Tabelle 18: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept)	10399

9.5 Referenzierte Dokumente

9.5.1 Dokumente der gematik

Die nachfolgende Tabelle enthält die Bezeichnung der in dem vorliegenden Dokument referenzierten Dokumente der gematik zur Telematikinfrastruktur. Der mit der vorliegenden Version korrelierende Entwicklungsstand dieser Konzepte und

3948 Spezifikationen wird pro Release in einer Dokumentenlandkarte definiert; Version und
3949 Stand der referenzierten Dokumente sind daher in der nachfolgenden Tabelle nicht
3950 aufgeführt. Deren zu diesem Dokument jeweils gültige Versionsnummer entnehmen Sie
3951 der aktuellen, von der gematik veröffentlichten Dokumentenlandkarte, in der die
3952 vorliegende Version aufgeführt wird.
3953

[Quelle]	Herausgeber: Titel
[gemGlossar]	gematik: Glossar der Telematikinfrastruktur
[gemSpec_COS]	gematik: Spezifikation des Card Operating System (COS)
[gemSpec_Dokumentenverwaltung]	gematik: Spezifikation ePA-Dokumentenverwaltung
[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)

3954 9.5.2 Weitere Dokumente

3955

[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)
[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-TR-02102-1]	BSI TR-02102-1 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“ Version 2018-02, Stand 29.05.2018 https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_hm.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 2018-01 https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_hm.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 2018-01 https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_hm.html
[BSI-TR-03111]	Technical Guideline BSI TR-03111 Elliptic Curve Cryptography, Version 2.10, Date: 2018-06-01 https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03111/index_hm.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/DE/Publikationen/TechnischeRichtlinien/tr03116/index_hm.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/netahtml/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
[EN-14890-1]	DIN EN 14890-1:2008 Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic services
[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)
[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-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
[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
[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/
[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-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
[NIST-SP-800-56A]	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]
[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

[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
[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
[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-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
[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-2018]	SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms, Version 1.1, June 2018 https://www.sogis.org/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.1.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
[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/
[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
--	--

3956

ENTWURF