Elektronische Gesundheitskarte und Telematikinfrastruktur

Übergreifende Spezifikation

Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur

Version:	2.49.1
Revision:	1760370
Stand:	24.06.2026
Status:	freigegeben
Klassifizierung:	öffentlich
Referenzierung:	gemSpec_Krypt

Dokumentinformationen

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird in diesem Dokument überwiegend die männliche Form verwendet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Änderungen zur Vorversion

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

Dokumentenhistorie

Version	Datum	Grund der Änderung, besondere Hinweise	Bearbeitung
...			
2.40.0	28.03.2025	Einarbeitung Konn_25.1	gematik
2.40.1	22.05.2025	Ausbau Schlüsselgenerierungsdienst	gematik
2.40.2	25.06.2025	editorisch: Übernahme Anforderungen aus gemSpec_DS_Anbieter (TI Security Standard und zu Wanda_25_1)	gematik
2.41.0	01.10.2025	Einarbeitung gemäß Änderungsliste E-Rezept_25.1, Übernahme Afos aus gemF_eRp_T-Rezept	gematik
2.42.0	12.12.2025	Release NCPeH_25_1 (C_12532), Zuordnungen zu Anb_TI-M (statt Anb_TI_Gateway)	gematik
2.43.0	17.12.2025	Release NCPeH_25_1 / E-Rezept_25_3 (C_12440) / E-Rezept C_12190	gematik
2.44.0	20.01.2026	Release E-Rezept_25_3 (C_12445, C_12410)	gematik
2.45.0	03.02.2026	Einarbeitung KIM_1_5_5	gematik
2.46.0	03.03.2026	Anpassungen aufgrund PoPP1_25_2	gematik

2.47.0	20.03.2026	Anpassungen aufgrund ePA_3.1.3-2	gematik
2.48.0	27.04.2026	Anpassung aufgrund PoPP1_26_1 (C_12662)	gematik
2.49.0	11.05.2026	Anpassung aufgrund CI_26_3	gematik
2.49.1	24.06.2026	A_27857-02 entfernt (noch nicht freigegeben), A_27857-01 weiter enthalten	gematik

Inhaltsverzeichnis

1 Einführung 7

1.1 Zielsetzung und Einordnung des Dokuments 7

1.2 Zielgruppe 7

1.3 Geltungsbereich 7

1.4 Methodik 8

2 Kryptographische Verfahren und deren Schlüssel 9

2.1 Identitäten 9

2.1.1 X.509-Identitäten 9

2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen 11

2.1.1.2 Qualifizierte elektronische Signaturen 14

2.1.1.3 TLS-Authentifizierung 16

2.1.1.4 IPsec-Authentifizierung 16

2.1.1.5 Digitale Signaturen durch TI-Komponenten 16

2.1.1.6 Verschlüsselung 17

2.1.2 CV-Identitäten 17

2.1.2.1 CV-Zertifikate G2 17

2.1.2.2 CV-Certification-Authority (CV-CA) Zertifikat G2 17

2.2 Zufallszahlengeneratoren 18

2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren) 18

2.4 Masterkey-Verfahren (informativ) 19

2.5 Schlüsselerzeugung und Schlüsselbestätigung 21

2.5.1 Prüfung auf angreifbare (schwache) Schlüssel 23

2.5.2 ECC-Schlüssel in X.509-Zertifikaten 23

2.5.3 RSA-Schlüssel in X.509-Zertifikaten 24

2.6 Einmalpasswörter 24

2.7 Schlüsselverwaltung 25

2.8 Import von Root-CA-Schlüsseln über Cross-Zertifikate 26

3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien 28

3.1 Kryptographische Algorithmen für XML-Dokumente 28

3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen 29

3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen 30

3.1.3 Webservice Security Standard (WSS) 31

3.1.4 XML-Verschlüsselung – Symmetrisch 32

3.1.5 XML-Verschlüsselung – Hybrid 32

3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung 32

3.2.1 Card-to-Card-Authentisierung G2 32

3.2.2 Card-to-Server (C2S) Authentisierung und Trusted Channel G2 33

3.3 Netzwerkprotokolle....33
3.3.1 IPsec-Kontext....33
3.3.2 TLS-Verbindungen....35
3.3.3 DNSSEC-Kontext....44
3.4 Hybride Verschlüsselung binärer Daten....45
3.5 Symmetrische Verschlüsselung binärer Daten....46
3.6 Signatur binärer Inhaltsdaten (Dokumente)....46
3.7 Signaturen innerhalb von PDF/A-Dokumenten....47
3.8 Kartenpersonalisierung....48
3.9 Bildung der pseudonymisierten Versichertenidentität....49
3.10 Spezielle Anwendungen von Hashfunktionen....49
3.10.1 Hashfunktionen und OCSP (informativ)....50
3.11 kryptographische Vorgaben für die SAK des Konnektors....51
3.12 Migration im PKI-Bereich....51
3.13 Spezielle Anwendungen von kryptographischen Signaturen....51
3.14 ePA-spezifische Vorgaben....52
3.14.1 Verbindung zur VAU....52
3.14.2 ePA-Aktensysteminterne Schlüssel....52
3.14.3 ePA-spezifische TLS-Vorgaben....54
3.14.4 Zugriffscode-Erzeugung....56
3.15 Anomalie-Erkennung....57
3.16 E-Rezept-spezifische Vorgaben....60
3.16.1 E-Rezept-spezifische TLS-Vorgaben (Internet)....61
3.17 KOM-LE-spezifische Vorgaben....62
3.18 Kryptographisch gesicherte VSDM-Prüfziffer Version 1....62
3.19 Kryptographisch gesicherte VSDM-Prüfziffer Version 2....64
3.20 spezifische TLS-Vorgaben für VSDM....73
3.21 NCPeH....74
4 Migration 120-Bit-Sicherheitsniveau....75
4.1 PKI-Begriff Schlüsselgeneration....75
4.2 X.509-Root der TI....76
4.3 TSL-Dienst und ECDSA-basierte TSL allgemein....78
4.4 ECC-Unterstützung bei TLS....78
4.5 ECC-Unterstützung bei IPsec....80
4.6 ECDSA-Signaturen....82
4.6.1 ECDSA-Signaturen im XML-Format....82
4.6.2 ECDSA-Signaturen im CMS-Format....82
4.7 ECIES....83
4.7.1 ECIES und authentisierte Broadcast-Encryption....87
4.7.2 ECIES und mobKT....87

4.8 ECC-Migration eHealth-KT....88
4.8.1 Interoperabilität zwischen eHealth-KT und Konnektor....90
4.9 ECC-Migration Konnektor....92
4.10 Verschiedene Produkttypen und ECC-Migration (informativ)....93
5 VAU-Protokoll für E-Rezept....94
5.1 Übersicht (informativ)....94
5.2 Definition....96
5.2.1 E-Rezept-VAU-Identität....96
5.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität....96
5.2.3 E-Rezept-VAU-Request und -Response....101
5.2.4 Zufallsquelle für Clients....106
6 VAU-Protokoll für ePA für alle....108
6.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung....111
6.2 Transport und Sicherung der Nutzdaten....121
6.3 OIDC-Authentisierung eines Clients (Nutzer)....125
6.4 Authentisierung des E-Rezept-FD als ePA-Client....127
6.5 Routing auf VAU-Instanzen....130
6.6 Fehlersignalisierung....132
6.7 Tracing in Nichtproduktivumgebungen....133
6.8 Zufallsquelle für Clients....134
7 ZETA/ASL (VAU-Protokoll)....135
7.1 Verbindungsaufbau/Schlüsselaushandlung....135
7.2 Transport und Sicherung der Nutzdaten....142
7.3 Fehlersignalisierung....146
7.4 Tracing in Nichtproduktivumgebungen....147
8 Post-Quanten-Kryptographie (informativ)....149
9 Anhang – Verzeichnisse....150
9.1 Abkürzungen....150
9.2 Glossar....152
9.3 Abbildungsverzeichnis....152
9.4 Tabellenverzeichnis....152
9.5 Referenzierte Dokumente....153
9.5.1 Dokumente der gematik....153
9.5.2 Weitere Dokumente....153

1 Einführung

1.1 Zielsetzung und Einordnung des Dokuments

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

Nur die in [gemSpec_Krypt] aufgeführten kryptographischen Verfahren dürfen in den Produkten in der TI verwendet werden. Die Vorgaben im Dokument werden in Zusammenarbeit mit dem BSI festgelegt. Neben den eigentlichen kryptographischen Verfahren werden ebenfalls erlaubte Domainparameter (Schlüssellänge, ECC-Kurvenparameter etc.) und Vorgaben zu sicherheitstechnisch geeigneten Verwendung der kryptographischen Verfahren in [gemSpec_Krypt] festgelegt.

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

Bei neuen Erkenntnissen über die verwendeten kryptographischen Algorithmen 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.

1.2 Zielgruppe

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. gemPTV_ATV_Festlegungen, 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 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 Kryptographische Verfahren und deren Schlüssel

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

GS-A_3080 -asymmetrischen Schlüssel maximale Gültigkeitsdauer
Die Lebensdauer von asymmetrischen Schlüsseln und somit die in einem Zertifikat angegebene Gültigkeitsdauer SOLL maximal 5 Jahre betragen.**[<=]**

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.

GS-A_2158-01 -Trennung von kryptographischen Identitäten und Schlüsseln in Produktiv- und Testumgebungen
Der Anbieter MUSS sicherstellen, dass kryptographische Identitäten bzw. Schlüssel der Produktivumgebung der TI des Anbieters (Umgebungen mit Echtdaten) nicht in Testumgebungen und dass keine kryptographischen Identitäten bzw. Schlüssel aus Testumgebungen in der Produktivumgebung der TI genutzt werden.**[<=]**

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.

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-Identitätder eGK des Versicherten ENC-Identität des HBA ENC-Identität der SMC-B

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

2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen

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

Alle Produkttypen, die X.509-Identitäten bei der Erstellung oder Prüfung digitaler nicht-qualifizierter elektronischer Signaturen verwenden, MÜSSEN die in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenvorgaben erfüllen.
Produkttypen, die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen.
[<=]

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

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

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

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

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

A_23458 -Konnektor, Zulässigkeitszeiträume kryptographische Algorithmen

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

Erläuterung: Analog zu A_15590 für die TSP der TI gilt, dass die Unterbindung der Verwendung von RSA mit Schlüssellängen unter 3000 Bit durch die gematik erfolgt durch die Herausnahme der entsprechenden RSA-basierten Sub-CA-Zertifikate aus der TSL zum Zeitpunkt des Ablaufens der Zulässigkeit (gemäß TIP1-A_2062).

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

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_22220-01 -Konnektor: zulässige Algorithmen und Domainparameter bei Zertifikatsprüfungen
Ein Konnektor KANN bei einer Zertifikatsprüfung alle im SOGIS-Katalog [SOG-IS] als zulässig aufgeführten kryptographischen Signaturverfahren inkl. der dem jeweiligen Verfahren zugehörigen Domainparametern (Mindestschlüssellängen, Kurvenparameter etc.) für eine Zertifikatsprüfung verwenden, sofern die Angaben aus [gemSpec_Krypt#Tab_KRYPT_002 und _002a (und auch _003 und _003a)] als Mindestvorgaben (Mindestschlüssellängen, Mindestgrößen der Kurvenparameter etc.) eingehalten werden. **[<=]**

A_19073 -Feste Laufzeit CV-Zertifikate einer Karte (eGK/HBA/SMC-B)
Die Anbieter CVC-TSP eGK, Anbieter HBA und Anbieter SMC-B MÜSSEN CV-Zertifikate tagesgenau in der Laufzeit auf die am kürzest gültigen X.509-Zertifikate der "Schlüsselgeneration ECDSA" der Karte beschränken.
Sind keine X.509-Zertifikate der "Schlüsselgeneration ECDSA" auf der Karte vorgesehen, dann MUSS die Laufzeit auf die am kürzest gültigen X.509-Zertifikate der "Schlüsselgeneration RSA" der Karte beschränkt werden.**[<=]**

A_19173 -Feste Laufzeit X.509-Zertifikate einer Karte (eGK/HBA/SMC-B)
Der Anbieter HBA, Anbieter SMC-B und der Anbieter X.509 TSP eGK MÜSSEN alle X.509-Zertifikate der "Schlüsselgeneration ECDSA" der Karte tagesgenau in der Laufzeit auf die der am längsten gültigen CV-Zertifikate der Karte beschränken. Sind keine X.509-Zertifikate der "Schlüsselgeneration ECDSA" auf der Karte vorgesehen, dann MUSS die Laufzeit aller X.509-Zertifikate der "Schlüsselgeneration RSA" der Karte tagesgenau in der Laufzeit auf die der am längsten gültigen CV-Zertifikate der Karte beschränkt werden.
[<=]

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

A_23139 -TSP-X.509-nonQES: ECC-Kurvenparameter, Komplexitätsreduktion
Ein TSP-X.509-nonQES, der nicht die X.509-Root-CA der TI ist, MUSS sicherstellen, dass

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

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

Verständnishinweis:

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

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

Für bestimmte Fachdienste, die zukünftig direkt von einen Primärsystem per TLS erreichbar sein sollen, sollen TLS-Zertifikate in der Komponenten-PKI der TI erzeugt werden können, die anstatt brainpool-Kurvenpunkte (brainpoolP256r1) NIST-Kurvenpunkte (P-256) als öffentliche Schlüssel enthalten. Grund dafür ist die deutlich bessere Unterstützung der NIST-Kurvenparameter durch verschiedene Standard-Kryptographie-Softwarebibliotheken.

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

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

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

2.1.1.2 Qualifizierte elektronische Signaturen

GS-A_4358-01 -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.**[<=]**

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	RSA-Signaturvariante: **Entweder** OID 1.2.840.113549.1.1.1 (rsaEncryption) (zulässig bis gemäß [SOG-IS]) **oder** OID 1.2.840.113549.1.1.10 (id-RSASSA-PSS) [RFC-5756]. (zulässig bis gemäß [SOG-IS]) Die Auswahl obliegt dem EE-Zertifikatsausgebenden VDA. RSA-Schlüssellänge: zu verwendende Schlüssellänge: 2048 Bit, zulässigbis vgl. Angabe in [SOG-IS]
Signatur eines Zertifikats, Signatur einer OCSP-Response	**Entweder** sha256withRSAEncryption (OID 1.2.840.113549.1.1.11) (zulässig bis gemäß [SOG-IS]) **oder**

oder Signatur eines OCSP-Responder-Zertifikates	id-RSASSA-PSS (1.2.840.113549.1.1.10) [RFC-5756] (zulässig bis gemäß [SOG-IS]) zu verwendende Schlüssellänge: 2048 Bit, zulässig bis vgl. Angabe in [SOG-IS] Die Hashfunktion für die Hashwertberechnung der TBSCertificate-Datenstruktur MUSS eine nach [SOG-IS] zulässige Hashfunktion („Agreed Hash Function“) sein. Als Hashfunktion SOLL SHA-256 [FIPS-180-4] verwendet werden. Als MGF MUSS MGF1 [PKCS#1] verwendet werden. Die innerhalb der MGF1 verwendete Hashfunktion MUSS die gleiche Hashfunktion sein, wie die Hashfunktion der Hashwertberechnung der TBSCertificate-Datenstruktur. (Dies entspricht der Empfehlung aus [RFC-5756] bzw. [RFC-4055, 3.1] und dient der Komplexitätsreduktion.) Die Saltlänge MUSS mindestens 256 Bit betragen.(Die Maximallänge des Salts ergibt sich nach [PKCS#1] in Abhängigkeit von der Länge des Moduls.)

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

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

OCSP-Responder-Zertifikates	

2.1.1.3 TLS-Authentifizierung

GS-A_4359-02 -X.509-Identitäten für die Durchführung einer TLS-Authentifizierung
Alle Produkttypen, die X.509-Identitäten für eine TLS-Authentifizierung verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen.**[<=]**

A_22457 -TLS-Clients, Ciphersuiten bei TLS-Verbindung mit eHealth-KT
Alle Produkttypen, die als TLS-Client gegenüber dem eHealth-Kartenterminal agieren, DÜRFEN bei beidseitig authentisierten TLS-Verbindungen NICHT Ciphersuiten mit Authentisierungsalgorithmen (RSA bzw. ECDSA) anbieten, wenn sie nicht auch für die Clientauthentisierung Schlüsselmaterial und Zertifikat für diese Authentisierungsalgorithmen besitzen.**[<=]**

2.1.1.4 IPsec-Authentifizierung

GS-A_4360-01 -X.509-Identitäten für die Durchführung der IPsec-Authentifizierung
Alle Produkttypen, die X.509-Identitäten für eine IPsec-Authentifizierung verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen.**[<=]**

2.1.1.5 Digitale Signaturen durch TI-Komponenten

GS-A_4361-02 -X.509-Identitäten für die Erstellung und Prüfung digitaler Signaturen
Alle Produkttypen, die X.509-Identitäten verwenden, die zur Erstellung und Prüfung digitaler Signaturen in Bezug auf TI-Komponenten (technische X.509-Zertifikate) genutzt werden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.
Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab_KRYPT_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen.**[<=]**

2.1.1.6 Verschlüsselung

GS-A_4362-02 -X.509-Identitäten für Verschlüsselungszertifikate
Alle Produkttypen, die X.509-Identitäten für die Verschlüsselung (Verschlüsselungszertifikate) verwenden, MÜSSEN alle in Tab_KRYPT_002 aufgeführten Algorithmen unterstützten 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_4365-02 -CV-Zertifikate G2
Alle Produkttypen, die CV-Zertifikate der Kartengeneration G2 erstellen oder prüfen, MÜSSEN die in Tab_KRYPT_006 aufgeführten Algorithmen verwenden und die Tabellenanforderungen erfüllen.
[<=]

Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate

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

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_4366-02 -CV-CA-Zertifikate G2
Alle Produkttypen, die CV-CA-Zertifikate der Kartengeneration G2 erstellen oder prüfen, MÜSSEN dien 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 2029+
Signatur des CA-Zertifikates	[RFC-5639#3.4, brainpoolP256r1] ecdsa-with-SHA256 {OID 1.2.840.10045.4.3.2}	256 Bit bis Ende 2029+

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

2.2 Zufallszahlengeneratoren

GS-A_4367 -Zufallszahlengenerator
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 Masterkey-Verfahren (informativ)

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

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

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

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

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

Tabelle 8: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen Schlüssels

Reihenfolge	Beschreibung	Formale Darstellung
1	Bildung eines Hashwertes über dem	HASH#1 =

	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.	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)

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

Tabelle 9: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines versichertenindividuellen Schlüssels

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

2.5 Schlüsselerzeugung und Schlüsselbestätigung

GS-A_4368 -Schlüsselerzeugung
Alle Produkttypen, die Schlüssel erzeugen, MÜSSEN die Anforderungen aus [BSI-TR-03116-1#3.9 Schlüsselerzeugung] erfüllen.**[<=]**

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

GS-A_5021 -Schlüsselerzeugung bei einer Schlüsselspeicherpersonalisierung
Ein Herausgeber von Sicherheitsmodulen für kryptographisches Schlüsselmaterial, welche in der TI genutzt werden (also bspw. eGK, SMC-B, HSM-B, SMC-KT und HBA), MUSS sicherstellen, dass auf dem Sicherheitsmodul gespeicherten Schlüssel die Anforderungen

aus [BSI-TR-03116-1#3.5 Schlüsselerzeugung] erfüllen.
[<=]

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

GS-A_5338 -HBA/SMC-B – Erzeugung asymmetrischer Schlüsselpaare auf der jeweiligen Karte selbst
Ein Kartenherausgeber oder, falls der Kartenherausgeber einen Dritten mit der Kartenpersonalisierung beauftragt, der Kartenpersonalisierer für HBA oder SMC-B MUSS sicherstellen, dass bei der Personalisierung der Karten HBA und SMC-B alle asymmetrischen Schlüsselpaare, bei denen die privaten Schlüssel auf der Karte gespeichert werden, auf der Karte erzeugt werden.
[<=]

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

GS-A_5386 -kartenindividuelle geheime und private Schlüssel G2-Karten
Ein Kartenherausgeber, der G2-Karten herausgibt, MUSS sicherstellen, dass bei der Personalisierung der Karten alle für eine Karte zu personalisierenden privaten und geheimen Schlüssel kartenindividuell sind. Bei Beauftragung eines Dritten mit der Schlüsselerzeugung ist dies durch den Dritten sicherzustellen.
Falls symmetrische Schlüssel (bspw. SK.CMS.AES128) nicht pro Karte zufällig erzeugt werden, sondern mit einem Schlüsselableitungsverfahren erzeugt werden, so MUSS der Kartenherausgeber sicherstellen, dass

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

[<=]

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

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

- alle Verfahren aus [NIST-SP-800-108] mittels CMAC [NIST-SP-800-38B],
- alle Verfahren aus [NIST-SP-800-56-A] bzw. [NIST-SP-800-56-B] mittels jeder nach [BSI-TR-02102-1] zulässigen Hashfunktion,
- alle Verfahren aus [NIST-SP-800-56C] mittels CMAC [NIST-SP-800-38B] oder eines HMAC, der auf einer nach [BSI-TR-02102-1] zulässigen Hashfunktion basiert,
- das Verfahren nach [ANSI-X9.63, Abschnitt 5.6.3] mittels jeder nach [BSI-TR-02102-1] zulässigen Hashfunktion.

A_23900 -TSP-X.509: Einzigartigkeit der bestätigten öffentlichen Schlüssel
Ein TSP-X.509 nonQES SMC-B MUSS TSP-intern prüfen, ob EE-Schlüssel in den vom TSP auszustellenden Zertifikaten pro Identität einzigartig sind. Der TSP MUSS TSP-intern sicherstellen, dass zwei unterschiedliche End-Entitäten nicht das selbe Schlüsselpaar verwenden.
Der TSP MUSS von ihm in Zertifikaten bestätigte öffentliche EE-Schlüssel in einem Schlüsselspeicher ablegen. Dieser Schlüsselspeicher ist initial leer. Vor Ausstellen eines neuen Zertifikats MUSS der TSP prüfen, ob der öffentliche EE-Schlüssel schon im Schlüsselspeicher enthalten ist. Falls ja, so MUSS der TSP die Zertifikatserstellung ablehnen. Falls nein, MUSS der TSP nach erfolgreicher Zertifikatserstellung den nun im Zertifikat bestätigten öffentliche EE-Schlüssel im Schlüsselspeicher einfügen.**[<=]**

Erläuterung zu A_23900:

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

GS-A_5572 -Sichere Speicherung privater Schlüssel für TI-Identitäten
Der Anbieter weiterer Anwendungen mit Beeinträchtigung der TI MUSS die ihm zugeordneten privaten asymmetrischen Schlüssel von Identitäten der Telematikinfrastruktur sicher speichern, so dass der Schlüssel während der Speicherung insbesondere nicht unautorisiert ausgelesen, unautorisiert verändert, unautorisiert ersetzt oder in anderer Weise unautorisiert benutzt werden kann.**[<=]**

GS-A_5578 -Kryptographische Verfahren bei Anbietern ohne Beeinträchtigung der TI
Falls der Anbieter weiterer Anwendungen ohne Beeinträchtigung der TI kryptographische Verfahren zum Schutz personenbezogener medizinischer Daten verwendet, MÜSSEN die Vorgaben aus [gemSpec_Krypt] eingehalten werden**[<=]**

GS-A_5574 -Kryptographische Verfahren bei Anbietern mit Beeinträchtigung der TI
Falls der Anbieter weiterer Anwendungen mit Beeinträchtigung der TI kryptographische Verfahren zum Schutz personenbezogener medizinischer Daten verwendet, MÜSSEN die Vorgaben aus [gemSpec_Krypt] eingehalten werden.**[<=]**

2.5.1 Prüfung auf angreifbare (schwache) Schlüssel

A_17294 -TSP-X.509: Prüfung auf angreifbare (schwache) Schlüssel
Ein TSP-X.509-nonQES MUSS vor einer Zertifikatserzeugung den durch das Zertifikat zu bestätigenden öffentlichen Schlüssel auf dessen kryptographische Angreifbarkeit hin prüfen.
Falls die Prüfung des öffentlichen Schlüssels das Ergebnis „angreifbar“ liefert, so MUSS der TSP die Zertifikatserstellung für diesen Schlüssel ablehnen.
Mindestumfang der Prüfung MÜSSEN

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

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

TSPs, die im Internet TLS-Zertifikate ausgeben (bspw. für die Verwendung von HTTPS), müssen aufgrund der Baseline Requirement des CA/Browser Forums (https://cabforum.org/baseline-requirements-documents/) vor der

Zertifikatserzeugung kryptographische Prüfungen des zu bestätigenden öffentlichen Schlüssels durchführen. Analog gilt dies mit A_17294 auch für TI-TSPs. Die gematik stellt auf Anfrage eine Beispielimplementierung für die Tests des Mindestumfangs bereit.

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

2.5.2 ECC-Schlüssel in X.509-Zertifikaten

GS-A_5518 -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.
[<=]

A_17091 -ECC-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.

2.5.3 RSA-Schlüssel in X.509-Zertifikaten

A_17092-01 -RSA-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-02102-1#2.3.2 RSA-Verschlüsselung])?
   Falls nein, ist das Ergebnis FAIL.
2. Ist der Modulus des untersuchten RSA-Schlüssels kleiner als 2^2048?
   Falls ja, 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.

[<=]

A_17093 -RSA-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 (vgl. Erläuterung nach A_17093-*) kleiner als 6.72? Falls ja, so ist das Ergebnis FAIL.

[<=]

Erläuterung zu A_17093-*:

entropy()-Funktion in C:

```
double entropy(char *S, int len) {
      int table[256]={0};
      int i;
      double H=0;

      for(i=0; i<len; i++) {
          table[(unsigned char)S[i]]++;
      }
      for(i=0; i<256; i++) {
          if (table[i]>0) {
              H-= (double) table[i]/len*log2( (double) table[i]/len);
          }
      }
      return H;
}
```

2.6 Einmalpasswörter

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

A_22686 -Einmalpasswörter (One-Time-Passwords, OTP), Mindestentropie
Alle Produkttypen, die Einmalpasswörter (One-Time-Passwords, OTP) erzeugen, MÜSSEN sicherstellen, dass diese Einmalpasswörter eine Mindestentropie von 120 Bit besitzen. D. h. sie werden zufällig erzeugt und sind mit praktischer Sicherheit - Wahrscheinlichkeit gleich 1 – 2^(–120)) - nicht erratbar.**[<=]**

2.7 Schlüsselverwaltung

Dieses Modul ist nur von Anbietern umzusetzen, die eine Schlüsselverwaltung anbieten.

Hinweis: Die betroffenen zentralen Produkttypen sind CVC-Root, gematik Root-CA, Intermediär VSDM, Namensdienst, TSP CVC, TSP X.509 nonQES (HBA, SMC-B, gSMC, FD, ZD), TSL-Dienst, Fachdienst KOM-LE.

GS-A_3078 -Anbieter einer Schlüsselverwaltung: verpflichtende Migrationsstrategie bei Schwächung kryptographischer Primitive
Der Anbieter einer Schlüsselverwaltung innerhalb der TI MUSS in seinem Sicherheitskonzept notwendige Umstellungsprozesse bei Schwächung von kryptographischen Primitiven beschreiben und die Wirksamkeit der getroffenen Maßnahmen ist nachzuweisen (z.B. durch dokumentierte Notfallübungen, Ablaufprotokolle von abgewickelten Vorfällen).**[<=]**

GS-A_3125 -Schlüsselinstallation und Verteilung: Dokumentation gemäß Minimalitätsprinzip

Der Anbieter einer Schlüsselverwaltung MUSS in seinem Sicherheitskonzept dokumentieren, welcher Schlüssel in welcher Phase seines Lebenszyklus in welcher Systemkomponente transportiert wird. Es MUSS dabei sichergestellt werden, dass die Schlüssel nur an diejenigen Systemkomponenten verteilt werden, in denen ihr Aufenthalt vorgesehen ist und wo sie hinreichend geschützt sind.**[<=]**

GS-A_3130 -Krypto_Schlüssel_Installation: Dokumentation der Schlüsselinstallation gemäß Minimalitätsprinzip
Der Anbieter einer Schlüsselverwaltung MUSS in seinem Sicherheitskonzept dokumentieren, welcher Schlüssel in welcher Systemkomponente installiert wird. Es MUSS dabei sichergestellt werden, dass die Schlüssel in Systemkomponenten installiert werden, in denen ihr Aufenthalt vorgesehen ist und wo sie hinreichend geschützt sind. **[<=]**

GS-A_3139 -Krypto_Schlüssel: Dienst Schlüsselableitung
Der Anbieter einer Schlüsselverwaltung MUSS sicherstellen, dass der Ableitungsprozess unumkehrbar und nicht vorhersehbar ist (die Kompromittierung eines abgeleiteten Schlüssels darf nicht den Ableitungsschlüssel oder andere abgeleitete Schlüssel kompromittieren).**[<=]**

GS-A_3141 -Krypto_Schlüssel_Ableitung: Maßnahmen bei Bekanntwerden von Schwächen in der Schlüsselableitungsfunktion
Der Anbieter einer Schlüsselverwaltung MUSS im Falle des Einsatzes einer Schlüsselableitung (nach [ISO11770]) in seinem Sicherheitskonzept Maßnahmen für das Bekanntwerden von Schwächen des kryptographischen Verfahrens, welche die Grundlage der Schlüsselableitung ist, darlegen.**[<=]**

GS-A_3149 -Krypto_Schlüssel_Archivierung: Dokumentation der Schlüsselarchivierung gemäß Minimalitätsprinzip
Der Anbieter einer Schlüsselverwaltung MUSS, falls er kryptographische Schlüssel archiviert, in seinem spezifischen Sicherheitskonzept beschreiben, welcher Schlüssel in welcher Phase in welcher Systemkomponente archiviert wird. Er MUSS sicherstellen, dass die Schlüssel nur an diejenigen Systemkomponenten verteilt werden, in denen ihr Aufenthalt vorgesehen ist und wo sie hinreichend geschützt sind.**[<=]**

2.8 Import von Root-CA-Schlüsseln über Cross-Zertifikate

Bei verschiedenen Client-Systemen wird initial eine Root-CA-Version in Form eines Root-CA-Zertifikats als Vertrauensanker eingebracht (vgl. A_24958-*, A_26923-*, A_24470-* etc.). Root-CA-Versionen werden regelmäßig (i. d. R. alle 2 Jahre) von der X.509-Root-CA der TI neu erzeugt und diese müssen im Client sicherheitstechnisch geeignet automatisiert importiert werden können. Dafür werden Cross-Zertifikate von der X.509-Root-CA erzeugt und veröffentlicht (vgl. bspw. https://download.tsl.ti-dienste.de/ECC/ROOT-CA/roots.json). Aufgrund des Schalenmodells können die Cross-Zertifikate nicht die gleiche Gültigkeitsdauer wie die selbstsignierten Root-Zertifikate besitzen, die die Cross-Zertifikate bestätigen. Dies führt dazu, dass je nachdem mit welcher initiale Root-CA-Version der Client "begonnen" hat, die Root-Schlüssel im Vergleich zu anderen Clients mit unterschiedlichen Gültigkeitsdauern vorliegen. Dies ist eine technische Eigenschaft, die im vorliegenden Szenario eine ungewünschte Komplexitätserhöhung erzeugt, ohne wirklichen fachlichen Gewinn. Aus diesem Grunde wird folgende vereinfachenden Festlegung getroffen.

A_28419 -FdV: Import von Root-CA-Schlüssel über Cross-Zertifikate
Ein Client, der Root-CA-Schlüssel über Cross-Zertifikate importiert, MUSS folgendes Vorgehen umsetzen.

Im Client liegt ein Root-CA-Zertifikat als Start eines Import-Vorgangs vor, und der Client erhält ein Cross-Zertifikat für eine neue Root-CA-Version und das dazugehörige selbstsignierte Root-CA-Zertifikat der zu importierenden Root-CA-Version. Das Cross-Zertifikat wird im folgenden als C bezeichnet, das dazugehörige selbstsignierte Root-CA-Zertifikat als S.
Der Client hat das Ziel S als weiteren Vertrauensanker bei sich lokal hinzuzufügen. Dafür MUSS er

1. prüfen, ob die Signatur von C per Signaturprüfung valide rückführbar ist auf ein schon im System als ein Vertrauensanker vorhandenes Root-CA-Zertifikat.
2. prüfen, ob C zum Prüfzeitpunkt/Import-Zeitpunkt zeitlich noch gültig ist.
3. prüfen, ob der SubjectCommonName der im Cross-Zertifikat bestätigten Identität dem Namensschema "GEM.RCA<natürliche Zahl>" entspricht.
4. prüfen, ob der SubjectKeyIdentifier in C gleich dem in S ist.
5. prüfen, ob der SubjectCommonName in C und in S gleich ist.
6. prüfen, ob bestätigte öffentliche Schlüssel in C gleich dem in S ist.
7. prüfen, ob die Signatur von S valide per Signaturprüfung mit dem öffentlichen Schlüssel aus C prüfbar ist.

Ergibt eine der o. g. Prüfungen kein positives Prüfergebnis MÜSSEN C und S verworfen werden (der Import kann also nicht erfolgen).
Anderenfalls MUSS vom Client S als weiterer Vertrauensanker im Client hinzugefügt werden.
[<=]

3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien

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

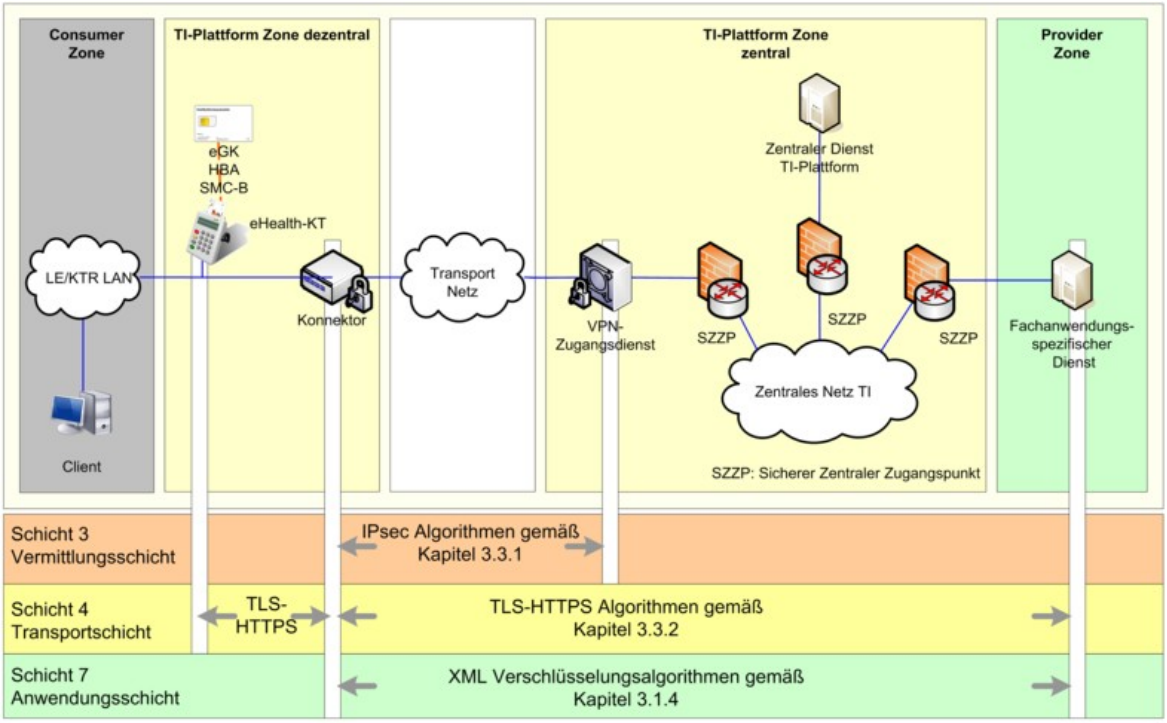

Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht

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

3.1 Kryptographische Algorithmen für XML-Dokumente

GS-A_4370 -Kryptographische Algorithmen für XML-Dokumente
Alle Produkttypen, die XML-Dokumente

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

[<=]

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

GS-A_5091 -Verwendung von RSASSA-PSS bei XMLDSig-Signaturen
Produkttypen, die RSASSA-PSS-Signaturen [PKCS#1] innerhalb von XMLDSig erstellen oder prüfen, MÜSSEN die Identifier aus [RFC-6931] Abschnitt „2.3.9 RSASSA-PSS With Parameters“ und „2.3.10 RSASSA-PSS Without Parameters“ für die Kodierung dieser Signaturen verwenden.
[<=]

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

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

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

3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen

GS-A_4371-02 -XML-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 10: 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.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten	RSASSA-PSS mit SHA-256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+ (Hinweis: siehe Abschnitt 4.1)	Die Verwendung des Algorithmus ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen

	Schlüssel		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/xmlenc#sha256	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	Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.

3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen

GS-A_4372-02 -XML-Signaturen für qualifizierte elektronische Signaturen
Alle Produkttypen, die XML-Signaturen für qualifizierte elektronische Signaturen erzeugen oder prüfen, MÜSSEN die Vorgaben der Tabelle Tab_KRYPT_010 erfüllen.**[<=]**

Tabelle 11: 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 Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA256 bis gemäß [SOG-IS] ECDSA mit SHA-256 bis gemäß [SOG-IS] (Hinweis: siehe Abschnitt 4.1)	Der Algorithmus muss für alle qualifizierten Signaturen verwendet werden. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256 Die [XMLDSig] konforme Bezeichnung lautet: http://www.w3.org/2001/04/xmlenc#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.

3.1.3 Webservice Security Standard (WSS)

Nicht relevant für den Wirkbetrieb der TI.

3.1.4 XML-Verschlüsselung – Symmetrisch

GS-A_4373 -XML-Verschlüsselung - symmetrisch

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

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

[<=]

3.1.5 XML-Verschlüsselung – Hybrid

GS-A_4374 -XML-Verschlüsselung - Hybrid
Alle Produkttypen, die Dokumente mittels [XMLEnc-1.1] hybrid verschlüsseln, MÜSSEN das Dokument gemäß [gemSpec_Krypt#GS-A_4373] symmetrisch verschlüsseln, wobei der eingesetzte symmetrischer Schlüssel (jeweils) für eine spezifische Person oder Komponente asymmetrisch verschlüsselt wird.
(Hinweis: Analog zum Hinweis in [gemSpec_Krypt#GS-A_4373] gilt auch hier, dass im Normalfall für die Sicherung der Integrität und Authentizität der übertragenen Daten zudem noch eine Signatur dieser Daten notwendig ist.)
[<=]

GS-A_4376-02 -XML-Verschlüsselung - Hybrid, Schlüsseltransport RSAES-OAEP
Alle Produkttypen, die Dokumente mittels [XMLEnc-1.1] RSA-basiert hybrid ver- und entschlüsseln, MÜSSEN für den Schlüsseltransport den Algorithmus RSAES-OAEP gemäß [PKCS#1] verwenden.**[<=]**

3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung

3.2.1 Card-to-Card-Authentisierung G2

GS-A_4379 -Card-to-Card-Authentisierung G2
Alle Produkttypen, die die Card-to-Card-Authentisierung für Karten der Generation G2 durchführen, MÜSSEN dabei eine CV-Identität gemäß [gemSpec_Krypt#GS-A_4365-*] verwenden.
[<=]

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

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

GS-A_4380 -Card-to-Server (C2S) Authentisierung und Trusted Channel G2
Alle Produkttypen, die eine Card-to-Server-Authentisierung für Karten der Generation G2 durchführen, MÜSSEN die folgenden Vorgaben berücksichtigen:

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

[<=]

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

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

GS-A_4381-01 -Schlüssellängen Algorithmus AES
Alle Produkttypen, die den Algorithmus AES nutzen, MÜSSEN die Schlüssellängen gemäß Tabelle Tab_KRYPT_012 nutzen.**[<=]**

Tabelle 12: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung

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

3.3 Netzwerkprotokolle

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

3.3.1 IPsec-Kontext

GS-A_4382-04 -IPsec-Kontext - Schlüsselvereinbarung
Alle Produkttypen, die die Authentifizierung, den Schlüsselaustausch und die verschlüsselte Kommunikation im IPsec-Kontext durchführen, MÜSSEN die 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 2025) 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-256 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, andere Verfahren dürfen nicht verwendet werden.
- Als PRF MUSS PRF_HMAC_SHA2_256 unterstützt werden. Es KÖNNEN weitere Verfahren nach [BSI-TR-02102-3, Abschnitt 3.2.2, Tabelle 3] verwendet werden, andere Verfahren dürfen nicht 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.

[<=]

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

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

GS-A_5548 -Mindestgültigkeitszeiten IKE- und IPsec-SAs (Konnektor)
Der Konnektor MUSS die Konfiguration der Gültigkeitsdauern der IKE- bzw. IPsec-SAs auf (1) mindestens 90% und (2) kleiner als 100% der in GS-A_4382-* Spiegelstrich

„Schlüsselaktualisierung“ aufgeführten Maximalwerte setzen.
[<=]

Auszug Beispielkonfiguration /etc/ipsec.conf

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

GS-A_5549 -Mindestgültigkeitszeiten IKE- und IPsec-SAs (VPN-Zugangsdienst)
Der VPN-Zugangsdienst MUSS die Konfiguration der Gültigkeitsdauern der IKE- bzw. IPsec-SAs auf die in GS-A_4382-* Spiegelstrich „Schlüsselaktualisierung“ aufgeführten Maximalwerte setzen.
[<=]

GS-A_5508 -IPsec make_before_break
Alle Produkttypen, die mittels IPsec Daten schützen, MÜSSEN die Reauthentication (vgl. [RFC-7296#2.8.3 „Reauthentication is done by […]“]) durchführen, indem die neue IKE-SA aufgebaut wird bevor die bestehende IKE-SA gelöscht wird.
[<=]

GS-A_4383 -IPsec-Kontext – Verschlüsselte Kommunikation
Alle Produkttypen, die mittels IPsec-Daten schützen, MÜSSEN dies ausschließlich auf Grundlage der in GS-A_4382-* als zulässig aufgeführten Verfahren und Vorgaben tun.
[<=]

A_14652 -SZZP-light, asymmetrischen Schlüssel maximale Gültigkeitsdauer
Die Lebensdauer von asymmetrischen Schlüsseln für die IPsec-Verbindungen im SZZP-light sowie Sicherheitsgateway Bestandsnetze und somit die in einem Zertifikat angegebene Gültigkeitsdauer DARF NICHT 5 Jahre überschreiten.
[<=]

3.3.2 TLS-Verbindungen

GS-A_4385 -TLS-Verbindungen, Version 1.2
Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die TLS-Version 1.2 [RFC-5246] unterstützen.
[<=]

A_18467 -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 Verbindungskonfigurationen (Handshake-Modi, (EC)DH-Gruppen, Signaturverfahren, Ciphersuiten etc.) verwenden, und
2. mindestens die Ciphersuite "TLS_AES_128_GCM_SHA256" dabei unterstützen.

[<=]

A_18464 -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 -TLS-Verbindungen, nicht Version 1.0

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

GS-A_5035 -Nichtverwendung des SSL-Protokolls
Alle Produkttypen, die Daten über Datenleitungen übertragen wollen, DÜRFEN NICHT das SSL-Protokoll unterstützen.**[<=]**

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

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

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

A_21275-01 -TLS-Verbindungen, zulässige Hashfunktionen bei Signaturen im TLS-Handshake
Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN sicherstellen, dass

1. sie im Rahmen der Erstellung und Prüfung von digitalen Signaturen im Rahmen des TLS-Handshakes ausschließlich folgende kryptographisch geeignete Hashfunktionen verwenden:
   a. SHA-256, SHA-384, SHA-512 [FIPS-180-4]
   b. SHA3-256, SHA3-384, SHA3-512 [FIPS-202]
2. sie dabei mindestens SHA-256 unterstützen,

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

Umsetzungshinweise zu A_21275-*:

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

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

GS-A_4384-03 -TLS-Verbindungen
Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die folgenden Vorgaben erfüllen:

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

[<=]

Erläuterung zu GS-A_4384-*:

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

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

ansix9p256r1	[ANSI-X9.62#L.6.4.3]
ansip256r1	http://oid-info.com/get/1.2.840.10045.3.1.7
prime256v1	[RFC-3279], `openssl ecparam -list_curves`
secp256r1	[RFC-5480],http://www.secg.org/collateral/sec2_final.pdf
P-256	[FIPS186-5]

Analog P-384 [FIPS186-5]:

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

GS-A_5541 -TLS-Verbindungen als TLS-Klient zur Störungsampel oder SM

Alle Produkttypen, die das TLS-Protokoll als TLS-Klient zur Störungsampel 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.

[<=]

GS-A_5580-01 -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.**[<=]**

A_22430 -TLS-Klient für betriebsunterstützende Dienste im Internet
Alle Produkttypen, welche die Betriebsdatenerfassung im Internet nutzen, MÜSSEN prüfen, ob das von der Betriebsdatenerfassung an der Internetschnittstelle während des TLS-Verbindungsaufbaus präsentierte TLS-Serverzertifikat gültig ist (d. h. u. a. per Zertifikatsprüfung rückführbar auf ein CA-Zertifikat einer CA, die die "CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates" (https://cabforum.org/baseline-requirements-documents/) erfüllt) und für den erwarteten FQDN ausgestellt wurde. Bei negativen Prüfergebnis MUSS der TLS-Verbindungsaufbau zur Betriebsdatenerfassung abgelehnt werden.**[<=]**

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

GS-A_5581 -"TUC vereinfachte Zertifikatsprüfung“ (Komponenten-PKI)
Alle Produkttypen, die eine Zertifikatsprüfung konform zu in dieser Anforderung definierten „TUC vereinfachte Zertifikatsprüfung“ durchführen wollen, erreichen dies indem sie folgende Vorgaben erfüllen.
(1) Es MUSS einen Prozess geben der authentisch und integer die Komponenten-CA-Zertifikate der TI regelmäßig (mindestens einmal pro Monat) ermittelt. Diese sind Basis für die folgenden Prüfschritte.
(2) Es MUSS geprüft werden, ob im vom TLS-Server präsentierten Zertifikat der korrekte (i. S. v. vom TLS-Client erwartete) FQDN enthalten ist (bspw. monitoring-update.stampel.telematik).
(3) Es MUSS geprüft werden, ob das präsentierte Zertifikat per Signaturprüfung rückführbar ist zu einem der CA-Zertifikate aus (1).
(4) Es MUSS geprüft werden, ob das präsentierte Zertifikat zeitlich gültig ist.

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

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

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

(a) Übergabe bei einem Vororttermin in der gematik,

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

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

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

Ziel ist es, dass für die Verbindung zur Störungsampel oder zum Service Monitoring auch einfach verfügbare und einfach verwendbare HTTPS-Clienten wie `wget` oder `curl` verwendet werden können.
Unter der Annahme, dass

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

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

```
wget --ca-directory=/etc/TI-Komponenten-CAs --post-
file=SOAP_Daten https://monitoring-update.stampel.telematik:8443/
I_Monitoring_Message
```

II.

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

GS-A_5542 -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 Cipher-Suites (vgl. GS-A_4384-* und A_17124-*) widerspiegelt. Um dieses Ziel zu erreichen, muss sichergestellt werden, dass in regelmäßigen Abständen frisches Schlüsselmaterial 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 Schlüsselmaterials gemacht (vgl. auch [SDH-2016]).

GS-A_5322 -Weitere 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 Schlüsselmaterial und alles davon abgeleitete Schlüsselmaterial (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 Schlüsselmaterial und alles davon abgeleitete Schlüsselmaterial (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 Schlüsselmaterial und alles davon abgeleitete Schlüsselmaterial (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 Schlüsselmaterial, dass bei der Erzeugung des SessionTickets (für die Sicherung von Vertraulichkeit und Authentizität der SessionTickets) verwendet wird, MUSS spätestens alle 48 Stunden gewechselt werden und das alte Material MUSS sicher gelöscht werden. Als kryptographische Verfahren zur Erzeugung/Sicherung der SessionTickets MÜSSEN ausschließlich nach [BSI-TR-03116-1] zulässige Verfahren verwendet werden und das Schlüsselmaterial muss die Entropieanforderungen gemäß [gemSpec_Krypt#GS-A_4368] erfüllen.
- Falls ein Produkttyp als *Klient* oder *Server* im Rahmen von TLS die Renegotiation unterstützt, so MUSS er dies ausschließlich nach [RFC-5746] tun. Ansonsten MUSS er die Renegotiation-Anfrage des Kommunikationspartners ablehnen.

[<=]

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

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

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

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

[<=]

GS-A_5525 -TLS-Renegotiation Konnektor

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

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

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

GS-A_5526 -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 -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 Cipher-Suiten aus GS-A_4384-* weitere Cipher-Suiten unterstützt werden. Mit der mittelfristigen Anhebung des zu erreichenden Sicherheitsniveaus auf 120 Bit (vgl. [SOG-IS] und [BSI-TR-02102-1]) werden die folgenden Cipher-Suiten 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_5345-04 -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_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 und P-384 [FIPS-186-5] 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.

[<=]

A_23226-01 -TLS-Verbindung, Konnektor: Legacy-KT-Unterstützung
Der Konnektor MUSS für die Unterstützung von alten eHealth-KT folgende TLS-Vorgaben ebenfalls unterstützen:

- Als Cipher Suite MUSS TLS_DHE_RSA_WITH_AES_128_CBC_SHA oder TLS_DHE_RSA_WITH_AES_256_CBC_SHA unterstützt werden.
- Dabei MUSS für die Schlüsselaushandlung Gruppe 14 (definiert in [RFC-3526], verwendbar bis Ende 2025) verwendet werden.
- Der private DH-Exponent für den Schlüsselaustausch MUSS eine Länge von mindestens 256 Bit haben.

[<=]

A_18183 -TLS-Protokoll-Verwendung in WANDA Basic
Falls ein Anbieter einer anderen Anwendung des Gesundheitswesens ohne Zugriff auf Dienste der TI in angeschlossenen Netzen des Gesundheitswesens (WANDA Basic) 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 (WANDA Basic) muss beim TLS-basierten Nachrichtentransport durch die TI nach [TR-02102-2] sichere Cipher-Suiten und Domainparameter verwenden. Für solch eine Anwendung ist eine die Interoperabilität mit TI-Diensten sicherstellende Einschränkung der Cipher-Suiten und Domainparameter nach GS-A_4384-* und A_17124-* nicht notwendig, d. h. beide Anforderungen gelten nicht für solche Anwendungen, sondern A_18183 gilt.

Gleiches gilt für die Verwendung von TLS für die Anbindung von Leistungserbringernetzen an das TI-Gateway, da bei dieser VPN-Anbindung Client und Server Teil des selben Produkttyps sind (TI-Gateway Zugangsmodul),sowie für die TLS-Verbindungen zwischen PoPP-Client bzw. -Modul zum PoPP-Service.

A_24779-02 -PoPP, TI-Gateway-Zugangsmodul und eHealth-CardLink - TLS-Cipher-Suiten
Die im Folgenden genannten Komponenten MÜSSEN ausschließlich TLS-Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] mit den dort vorgegebenen Domainparametern (zulässige Schlüssellängen, zulässige ECC-Kurven-Parameter etc.) verwenden bzw. bei Verwendung von TLS 1.3 die Vorgaben aus [TR-02102-2, Abschnitt 3.4] befolgen:

1. TI-Gateway Zugangsmodul, falls das TLS-Protokoll für die Sicherung der Datenübertragung aus dem Nutzernetzwerk zum Zugangsmodul des TI-Gateway verwendet wird,
2. PoPP-Service,
3. PoPP-Client,
4. PoPP-Modul,
5. eHealth-Cardlink, für die Verbindung der Clients eines Nutzers zum eHealth-Cardlink

[<=]

A_18986 -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.

Hinweis: In Abschnitt "3.14.3- ePA-spezifische TLS-Vorgaben " gibt es weitere TLS-Vorgaben.

A_28866 -Auswahl der Domain für TLS-terminierte Vertrauenswürdige Ausführungsumgebungen (VAU)
Der Anbieter, der eine TLS-Terminierte VAU betreibt, MUSS diese Endpunkte unter einer eigenständigen Domain betreiben, die für nichts anderes als Endpunkte dieser VAU verwendet wird.

Hinweis:
Um Überschneidungen der Namensräume von Zertifikaten zu vermeiden, also dass zum Beispiel ein *.firma.tld Zertifikat auch den vep1.firma.tld (oder ein *.sub1.fima.tld auch vep1.sub1.firma.tld) VAU Endpunkt umfasst, wird die Verwendung einer eigenständigen Second-Level Domain vorgeschrieben, die für nichts anderes als VAU Endpunkte des selben Betreibers verwendet werden darf.
Ist eine VAU über mehrere Endpunkten zu erreichen, so sind diese jeweils als alternative Namen (Subject Alternative Name) im Zertifikat aufzuführen.
Die Nutzung von Wildcard-Zertifikaten ist nicht zulässig.**[<=]**

A_28886 -Verbot von wildcard-Namenseinträgen für TLS-terminierte VAU Endpunkte
Zertifikate für TLS-Terminierte VAU Endpunkte DÜRFEN NICHT Wildcard-Namenseinträge beinhalten.**[<=]**

A_28976 -TLS-Verbindungen, TLS-Schnittstellen Internet-PKI
Ein Anbieter von Produkttypen der TI, desses Produkttypen TLS-Schnittstellen besitzen mit kryptographischen Identitäten aus einer Internet-PKI, MUSS sicherstellen, dass die TLS-Identitäten, die ein Anbieter (bzw. dessen beauftragter Betreiber) von einem Internet-TSP bezieht, mindestens ein Sicherheitsniveau (vgl. [gemSpec_Krypt#"Migration 120-Bit-Sicherheitsniveau"]) von 120 Bit besitzen. D. h. die im Zertifikat (TLS-Idenität) bestätigten Schlüssel MÜSSEN den Vorgaben aus [TR-02102-2#"Domainparameter und Schlüssellängen"] genügen.**[<=]**

3.3.3 DNSSEC-Kontext

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

GS-A_4388 -DNSSEC-Kontext

Alle Produkttypen, die DNSSEC verwenden, MÜSSEN die Algorithmen und Vorgaben gemäß Tabelle Tab_KRYPT_017 erfüllen.
[<=]

Tabelle 13: Tab_KRYPT_017 Algorithmen für DNSSEC

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

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

3.4 Hybride Verschlüsselung binärer Daten

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

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

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

- 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 zu verschlüsselnden Daten zudem noch eine Signatur dieser Daten notwendig ist.

[<=]

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

GS-A_4390 -Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten
Produkttypen, die die hybride Verschlüsselung binärer Daten durchführen, MÜSSEN für den asymmetrischen Anteil der Verschlüsselung die folgenden Vorgaben berücksichtigen:

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

[<=]

3.5 Symmetrische Verschlüsselung binärer Daten

GS-A_5016 -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].

A_15561 -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.**[<=]**

3.6 Signatur binärer Inhaltsdaten (Dokumente)

GS-A_5080-01 -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 14: 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 (CAdES) [ETSI-CAdES]	Die Verwendung des Standards ist für die Signatur von Dokumenten verpflichtend die mittels CMS [RFC-5652] erzeugt werden.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA-256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+	Die Verwendung einer dieser Algorithmen ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
DigestMethod	Methode zur Berechnung eines Digest der zu signierenden Bereiche	SHA-256	Die Verwendung des Algorithmus ist verpflichtend.
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.

3.7 Signaturen innerhalb von PDF/A-Dokumenten

GS-A_5081-01 -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 15: 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 Profiles; Part 3: PAdES Enhanced – PAdES-BES and PAdES-EPES Profiles Technical Specification, 2010 [PAdES-3]	Die Verwendung des Standards ist für die Signatur von PDF/A [PDF/A-2] Dokumenten verpflichtend, die mittels eingebetteter Signaturen signiert werden.
kryptographisches Signaturverfahren	Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel	RSASSA-PSS mit SHA-256 bis Ende 2025 ECDSA mit SHA-256 bis Ende 2029+	Die Verwendung einer dieser Algorithmen ist verpflichtend. Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein.
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 Kartenpersonalisierung

Vgl. auch Abschnitt 2.5- Schlüsselerzeugung und Schlüsselbestätigung .

GS-A_4391 -MAC im Rahmen der Personalisierung der eGK

Der Herausgeber der eGK MUSS sicherstellen, dass bei der Personalisierung der eGK die Daten bei der Übermittlung integritätsgeschützt werden. Für die Absicherung der Integrität ist in diesem Kontext der AES-256 CMAC nach [NIST-SP-800-38B] (vgl. [BSI-TR-03116-1#3.2.2, 4.5.2]) zu verwenden.
Die Länge des CMAC muss 128 Bit betragen.
Nach [NIST-SP-800-38B#S.13] sollen nicht mehr als 2^{48} Nachrichtenblöcke (2^{22} GByte) mit demselben Schlüssel verarbeitet werden. Nach [NIST-SP-800-38B#S.14] ist ein CMAC anfällig für Replay-Attacken, was bei der Anwendung des CMACs zu berücksichtigen ist.
[<=]

3.9 Bildung der pseudonymisierten Versichertenidentität

GS-A_4392 -Algorithmus im Rahmen der Bildung der pseudonymisierten Versichertenidentität
Alle Produkttypen, die pseudonymisierte Versichertenidentitäten berechnen, MÜSSEN den Hash-Algorithmus SHA-256 [FIPS-180-4] verwenden.**[<=]**

3.10 Spezielle Anwendungen von Hashfunktionen

GS-A_4393 -Algorithmus bei der Erstellung von Hashwerten von Zertifikaten oder öffentlichen Schlüsseln
Alle Produkttypen, die Fingerprints eines öffentlichen Schlüssels oder eines Zertifikates erstellen, MÜSSEN den Hash-Algorithmus SHA-256 [FIPS-180-4] dafür verwenden.**[<=]**

Erläuterung:

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

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

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

GS-A_5131 -Hash-Algorithmus bei OCSP/CertID
Alle Produkttypen, die OCSP-Anfragen stellen oder beantworten, MÜSSEN bei der Erstellung und Verwendung der CertID-Struktur (vgl. [RFC-6960, Abschnitt 4.1.1] oder [RFC-2560, Abschnitt 4.1.1]) den Hash-Algorithmus SHA-1 [FIPS-180-4] verwenden.
Ein OCSP-Server KANN auch zusätzlich andere Hashfunktionen im Rahmen der CertID, die nach [BSI-TR-03116-1] zulässig sind, unterstützen.
[<=]

A_27608 -Zu unterstützende Hashfunktionen und RSA-Schlüssellänge für Auswertung der BNetzA-VL
Alle Produkttypen, die die eIDAS-Vertrauensliste der BNetzA ("BNetzA-VL") auswerten, MÜSSEN im Rahmen der Signaturprüfung der Liste die Hashfunktionen SHA-256, SHA-384 und SHA-512 und RSA-Schlüssel bis 4096 Bit unterstützen.**[<=]**

Hinweis:
Der Signer der BNetzA verwendet einen RSA-Schlüssel mit 4096 Bit Schlüssellänge.

3.10.1 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 Sicherheitssicht 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-02102-1] 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_4357-02) angezogen werden.

3.11 kryptographische Vorgaben für die SAK des Konnektors

GS-A_5071-01 -kryptographische 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:

- RSA
  - SHA-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
  - SHA-256 nach FIPS-180-4 (März 2012) [FIPS-180-4] (Abschnitt 6.2),
  - ECDSA basierend auf E(F_p) (vgl. Technische Richtlinie 03111, Version 2.0) auf der Kurve P256r1 [RFC-5639].

[<=]

3.12 Migration im PKI-Bereich

GS-A_5079 -Migration 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.**[<=]**

3.13 Spezielle Anwendungen von kryptographischen Signaturen

GS-A_5207-01 -Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal
Alle Produkttypen MÜSSEN 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.

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

[<=]

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_5340 -Signatur der TSL
Der TSL-Dienst MUSS für die Signatur der TSL das Signaturverfahren RSASSA-PSS [PKCS#1] verwenden mit dem XMLDSig-Identifier „http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1“ nach [RFC-6931, Abschnitt „2.3.10 RSASSA-PSS Without Parameters“].**[<=]**

3.14 ePA-spezifische Vorgaben

3.14.1 Verbindung zur VAU

Der Begriff "vertrauenswürdige Ausführungsumgebung" (VAU) wird in [gemSpec_Aktensystem_ePAfueralle] eingeführt. Jeder ePA-Client 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_15549-01 -ePA-VAU-Client: Kommunikation zwischen ePA-Client und ePA-VAU
Ein ePA-Client MUSS bei der Kommunikation mit der VAU das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "VAU-Protokoll für ePA für alle"] verwenden.
Der Client einer VAU MUSS nach spätestens 24 Stunden das Aushandeln neuer AES-Verbindungsschlüssel erzwingen. Er MUSS zeitlich abgelaufene Verbindungsschlüssel bei sich sicher löschen.**[<=]**

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

A_15547-01 -ePA-VAU: Kommunikation zwischen ePA-VAU und ePA-Client
Das ePA-Aktensystem MUSS sicherstellen, dass dessen VAU bei der Kommunikation mit dem ePA-Client das Kommunikationsprotokoll aus [gemSpec_Krypt#Abschnitt "VAU-Protokoll für ePA für alle"] verwendet.
Die VAU MUSS sicherstellen, dann jeder ausgehandelte AES-Verbindungsschlüssel nach spätestens 24 Stunden sicher gelöscht wird.
Die VAU MUSS ein AUT-Zertifikat aus der Komponenten-PKI der TI besitzen (mit Rollenkennung-OID "oid_epa_vau") das Verbindungsparameter (vgl. A_24425-*) authentisiert.**[<=]**

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

3.14.2 ePA-Aktensysteminterne Schlüssel

A_15745-01 -Betreiberschlüssel Aktensystem
Ein ePA-Aktensystem MUSS sicherstellen, dass

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

[<=]

A_15746-01 -Sicherstellung der Verfügbarkeit der betreiberspezifischen Schlüssel
Ein ePA-Aktensystem MUSS sicherstellen, dass für die Sicherstellung der Verfügbarkeit der betreiberspezifischen Schlüssel (vgl. A_15745-*) eine sicherheitstechnisch geeignete Sicherung (Backup) des Schlüsselmaterials erzeugt und sicher verwahrt wird.
[<=]

A_16176-01 -Mindestvorgaben für ePA-Aktensystem-interne Schlüssel
Ein ePA-Aktensystem MUSS bei innerhalb des Aktensystems eingesetzten Schlüsselmaterial, das nicht aus der TI-PKI kommt (Signatur Authorisierungstoken etc.), folgende Vorgaben umsetzen:

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

[<=]

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

A_20519-03 -Wechsel der betreiberspezifischen Schlüssel
Ein ePA-Aktensystem MUSS sicherstellen, dass

1. die betreiberspezifischen Schlüssel (BS) (vgl. A_15745-*) mindestens jährlich gewechselt werden,
2. alle auf Grundlage eines alten BS erzeugten Chiffrate dieser Akte für den aktuellsten BS umgeschlüsselt werden, und
3. anschließend die mit einem alten BS erzeugten Chiffrate dieser Akte sicher gelöscht werden.

[<=]

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

A_26250 -CMAC und Befugnisverifikation
Ein ePA-Aktensystem MUSS sicherstellen, dass

1. die symmetrischer Schlüssel für CMAC-Sicherung der Befugnisse 128 Bit AES-Schlüssel sind (vgl. [gemSpec_Aktensystem_ePAfueralle#3.3. Sichere Speicherung sensibler Schlüssel und Informationen im VAU-HSM]),
2. als CMAC-Verfahren das CMAC-Verfahren nach [NIST-SP-800-38B] mittels AES zum Einsatz kommt,
3. die Ausgabelänge des CMAC von 128 Bit (= 16 Byte) ungekürzt im ePA-Aktensystem bei der Erstellung und der Prüfung eines CMAC-Wertes verwendet wird,
4. die CMAC-Schlüssel regelmäßig (mindestens jährlich) gewechselt werden (vgl. Hinweise zu A_26250-*).

[<=]

Hinweis zu A_26250-*:

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

3.14.3 ePA-spezifische TLS-Vorgaben

A_15751-03 -TLS-Verbindung zwischen ePA-Aktensystem und ePA-Client
Ein ePA-Aktensystem und ein ePA-Client MÜSSEN in Bezug auf die TLS-Verbindung zwischen ihnen

1. folgende Ciphersuiten unterstützen
   - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C),
   - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B).
2. Sie KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.
3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch und bei der Signaturprüfung mittels ECDSA MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-5] unterstützt werden. Daneben 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).

[<=]

A_26025 -ePA: TLS-Identitäten, IOP, P-256 Basierte ECC-Schlüssel
Ein ePA-Aktensystem MUSS sicherstellen, dass seine TLS-Identitäten (vgl. A_15751-*) auf der Kurve P-256 [FIPS-186-5] basieren. D. h., der EE-Schlüssel im TLS-Server-Zertifikat als auch das bestätigende CA-Zertifikat (Komponenten-PKI-CA-Zertifikat) MÜSSEN als öffentliche Schlüssel Kurvenpunkte auf der ECC-Kurve P-256 besitzen.**[<=]**

Erläuterung:

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

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

A_24913 -ePA: TLS-Verbindungen, OCSP-Stapling
Ein ePA-Aktensystem MUSS bei allen seinen ePA-spezifischen HTTPS-Schnittstellen (Rolle TLS-Server) TLS-OCSP-Stapling [RFC-6066] verwenden (aktivieren). Es MUSS sicherstellen, dass die im TLS-Handshake mit gesendeten OCSP-Responses max. 50 Minuten alt sind. Sollte vom entsprechenden OCSP-Responder für den Bezug der OCSP-Responses trotz regelmäßigen Versuchens keine OCSP-Response bezogen werden können, so MUSS das Aktensystem die jüngste ihm zur Verfügung stehende OCSP-Response verwenden, und es regelmäßig weiter probieren (bspw. im 5'-Takt).

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

Erläuterung:

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

A_15833-01 -TLS-Verbindungen ePA-FdV
Ein ePA-Frontend des Versicherten MUSS die TLS-Vorgaben in A_15751-* bei allen seinen TLS-Verbindungen einhalten.
[<=]

A_21269-01 -ePA-Client: TLS-Session-Resumption
Ein ePA-Client SOLL TLS-Session-Resumption (per Session-ID oder per TLS-Session-Resumption per Session-Tickets) unterstützen.**[<=]**

3.14.4 Zugriffscode-Erzeugung

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

A_26301 -ePA-Client: Zugriffscode Erzeugung
Ein ePA-Client, der Zugriffscodes erzeugt (vgl. [gemSpec_ePA_FdV#Befugnisverwaltung EU-Zugriff]), MUSS folgendes sicherstellen:

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

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

Implementierungshinweis zu A_26301-*

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

Beispiel:

```
$ python
Python 3.9.16 (main, Mar  8 2023, 22:47:22)
>>> 26*2+10
62
>>> 62**6
56800235584
>>> import math
>>> math.ceil(math.log2(62))
6
```

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

3.15 Anomalie-Erkennung

Für die betreiberübergereifende Anomalie-Erkennung werden in der VAU bestimmte Attribute (vgl. [gemSpec_Perf#A_22469-*]) pseudonymisiert aus der VAU exportiert. Das Cyber-Defense-Center der TI kann bei der Feststellung von Anomalien, die einen Angriff als Ursache dringend vermuten lassen, die Pseudonyme depseudonymisieren, um weitere Maßnahmen zum Schutz der medizinischen Daten gezielt durchführen zu können.

A_27332-01 -ePA-Aktensystem und E-Rezept-FD: Pseudonymisieren der gematik-Logdaten
Ein ePA-Aktensystem und der E-Rezept-FD, die im Rahmen der Anomalie-Erkennung pseudonymisierte Daten erzeugen, MÜSSEN die zu pseudonymisierenden Daten (Teilmenge der Logdaten) mittels AES/CBC mit dem Schlüssel key_pn_log und dem festen Initialisierungsvektor IV=0...0 (16 Null-Bytes) verschlüsseln. Dabei MUSS ein von der gematik (CDC) bereitgestellter 128 Bit AES-Schlüssel verwendet werden (vgl. A_27392-*). Bei einer Verschlüsselung im CBC-Modus MUSS der zu verschlüsselnde Klartext gleich einem Vielfachen der Blocklänge der Blockchiffre sein. Dies ist bei den zu pseudonymiserenden Daten (DTBP) selten der Fall. Deshalb MUSS eine Kodierung der DTBP vor der Verschlüsselung wie folgt stattfinden:

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

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

Erläuterung zu A_27332-*: Durch die Verwendung eines konstanten Initialisierungsvektors (IV) ist die Verschlüsselung eine deterministische Verschlüsselung.

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

Der Leer-Block am Ende dient als (in seiner Leistungsfähigkeit sicher beschränkter) Integritäts- und Authentitätsschutz. Damit kann ein Depseudonymisierer ermitteln, ob ein

Pseudonym gültig ist. Ein umfassenderer Integritäts- und Authentitätsschutz macht fachlich genau an dieser Stelle wenig Sinn -- der Schutz muss über die Log-Daten in Ihrer Gesamtheit erfolgen.

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

Die gematik stellt Beispiel-Code für die Erstellung von Pseudonymen nach A_27332-* bereit.

A_27392-01 -ePA-Aktensystem und E-Rezept-FD: Import eines Pseudonymisierungsschlüssels (Anomalie-Erkennung)
Ein ePA-Aktensystem und der E-Rezept-FD, die im Rahmen der Anomalie-Erkennung pseudonymisierte Daten erzeugen, MÜSSEN von der gematik verschlüsselte Pseudonymisierungschlüssel importieren können. Dabei wird ein Export-Paket wie folgt (Hinweis: ähnlich zu A_27276-*) verwendet mit den gleichen kryptographischen Vorgaben aus A_27275-*. Die JSON-Struktur für den Export/Import hat folgende Struktur:

```
{
    "Schlüsseltyp": "Pseudonymisierungsschlüssel CDC",
    "key_version": "<eine natürliche Zahl als String hier aufgeführt>",
    "enc_cert_hash": "<sha-256-Hashwert-des-verwendeten-
Verschlüsselungszertifikats-in-Hexkodierung-Kleinbuchstaben(0-9a-f)>",
    "iat": "<YYYY-MM-DD>",
    "encrypted_key":
"016d604c0222680f2e5be7c15e173dcb9c59bf7e4aab4fa74b65e9ebe7b3ab9bc26702e117
a11aeacc109cb391d67b04fbf86e10d1a58402ef5765e670489426bd47077a0e683f397a169
7c1d91d34a4227c3cf19ffcc1d5be2d12509322c2097c40c62271a1a5ffe23a1f9e72"
}
```

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

Erläuterungen zu A_27392-*:

In "enc_cert_hash" wird der SHA-256-Hashwert des Verschlüsselungszertifikats aufgeführt auf dessen Grundlage die Verschlüsselung des Export-Pakets statt gefunden hat.

Das Chiffrat in "encrypted_key" hat einen 128-Bit AES-Schlüssel als Klartext. Aufgrund der kryptographischen Vorgaben aus (A_27275-*) bedeutet dies die Länge des Wertes (Zeichenkette) bei "encrypted_key" hat eine Länge von 218 Bytes. Ein Versionsbyte (01), X-Koordinate ephemerer ECC-Punkt (32 Bytes), analog Y-Koordinate (32 Bytes), Initialisierungsvektor (12 Bytes), eigentliche AES/GCM-Chiffrat (16 Bytes), Authentication-Tag (16 Bytes) macht in Summe 109 Bytes. Diese 109 Bytes werden hexadezimal kodiert, damit verdoppelt sich die Länge und man erhält 218 Bytes als Ergebnis.

Bei ePA und E-Rezept wird die Pseudonymisierung von Daten (Telemetriedaten / Anomalie-Erkennung) nach den zuvor in diesem Abschnitt aufgeführten Vorgaben durchgeführt. Für alle anderen Anwendungen wird für die Pseudonymisierung ein hybrides Verschlüsselungsverfahren (ECIES mit AES/GCM) eingesetzt. Dieses ist zwar weniger performant als das Verfahren aus A_27332-*, aber dafür kann auf die Verteilung der geheimen Pseudonymisierungsschlüssel (vgl. A_27392-*) verzichtet werden.

A_28578 -PoPP-Service - Pseudonymisierung bei den Telemetriedaten (Anomalie-Erkennung)

Der PoPP-Service MUSS bei der Pseudonymisierung von Daten im Kontext von A_26532-* folgende Vorgaben umsetzen:

1. Der PoPP-Service MUSS ein von der gematik/CDC bereitgestelltes Verschlüsselungszertifikat, integritäts- und authentitätsgeschützt in der VAU einpflegen (Initialisierung des Fachdienstes). (Hinweis: Das Verschlüsselungszertifikat ist ECC-basiert, d. h. auch der EE-Schlüssel ist ein ECC-Schlüssel.)
2. Der PoPP-Service MUSS auf Anweisung der gematik/CDC dieses Verschlüsselungszertifikat innerhalb von 5 Werktagen wechseln können. (Kontext: Regelmäßiger Wechsel des Verschlüsselungszertifikats)
3. Sei P die zu pseudonymisierende Zeichenkette. Der PoPP-Service MUSS am Ende von P solange Leerzeichen anfügen bis die Länge der somit erzeugten Zeichenkette ein Vielfaches von 32 ist. Sei Padding-Länge die Anzahl der hinzugefügten Leerzeichen. Diese Padding-Länge wird als Byte kodiert (Beispiel: Padding-Länge 0 würde als \x00 Byte kodiert) und der erzeugten Zeichenkette vorangestellt. Das Ergebnis wird als Plaintext bezeichnet.
4. Er MUSS diesen Plaintext mittels ECIES (AES/GCM) unter Verwendung des öffentlichen Verschlüsselungsschlüssels aus dem Verschlüsselungszertifikat aus Punkt 1 (bzw. 2) verschlüsseln.
   a. Dabei MUSS der PoPP-Service ein ephemeres ECDH-Schlüsselpaar zufällig erzeugen und mit diesem und dem öffentlichen Schlüssel aus dem Verschlüsselungszertifikat 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-cdc-p17" verwenden, d. h. in der Formulierung von [RFC-5869] info="ecies-cdc-p17".
   d. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key (CEK) für die Verwendung von AES/GCM ableiten.
   e. Er MUSS für die Verschlüsselung mittels AES/GCM einen 96 Bit langen Null-Vektor (0…0) als Initialisierungsvektor (IV) verwenden.
   f. Er MUSS mit dem CEK und dem IV mittels AES/GCM den Plaintext verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
   g. Aus dem Verschlüsselungszertifikat MUSS er den SubjectKeyIdentifier (SKI) entnehmen und die ersten 16 Bit als „gekürzter SKI“ im folgenden Schritt verwenden.
   h. Er MUSS das Ergebnis wie folgt kodieren: chr(0x02) || <gekürzter SKI> || <32 Byte X-Koordinate des öffentlichen Schlüssels aus (a) > || <AES-GCM-Chiffrat> || <16 Byte AuthenticationTag>. (Hinweis: es fehlt absichtlich die Y-Koordinate und der IV).
   i. Die X-Koordinate ist (wie üblich) vorne mit chr(0) zu padden solange bis sie eine Kodierungslänge von 32 Byte erreicht. Die Byte-Order MUSS Network-Byte-Order (= Big) sein.
   j. Das so kodierte Ergebnis wird als erweitertes Chiffrat bezeichnet.
5. Der PoPP-Service MUSS das erweiterte Chiffrat mittels Base64 kodieren. Das Ergebnis ist ein Pseudonym von P (vgl. Punkt 3)

[<=]

Hinweise zu A_28578-:*

1. *Die gematik stellt Beispiel-Code für die Pseudonymisierung von Daten nach A_28578-* bereit.*
2. *Aufgrund der zufälligen Erzeugung der ephemeren Schlüssel in A_28578-* Punkt 4 a) erzeugt jede Pseudonymisierung gemäß A_28578-* ein neues (anderes) Pseudonym.*

3.16 E-Rezept-spezifische Vorgaben

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

A_21332-02 -E-Rezept: TLS-Vorgaben
Ein E-Rezept-FD, ein Apothekenverzeichnis, ein E-Rezept-Client und ein IDP MÜSSEN in Bezug auf die TLS-Verbindung zwischen ihnen

1. folgende Ciphersuiten unterstützen
   - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30),
   - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F),
   - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C),
   - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B).
2. Sie KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.
3. Bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch und bei der Signaturprüfung mittels ECDSA MÜSSEN die Kurven P-256 oder P-384 [FIPS-186-5] unterstützt werden. Daneben SOLLEN die Kurven brainpoolP256r1, brainpoolP384r1 oder brainpoolP512r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven SOLLEN NICHT verwendet werden (Hinweis: die Intention des letzten Satzes ist insbesondere, dass die Ordnung des Basispunktes in E(F_p) nicht zu klein werden darf).

[<=]

Ähnlich wie bei der Anwendung ePA endet die TLS-Verbindung am E-Rezept-FD an der Webschnittstelle (Eingangspunkt). Ziel ist es die Code-Komplexität innerhalb der VAU so gering wie möglich zu halten (Trusted Computing Base), um eine ausreichende Sicherheitsanalyse des VAU-Programmcodes überhaupt erst möglich zu machen. Dafür werden die Probleme des TLS-Handlings, der Lastverteilung und des DoS-Schutzes auf Applikationsebene außerhalb der VAU an den Webschnittstellen des Fachdienstes E-Rezept bearbeitet. So kann sich der Programmcode in der VAU auf seine zentrale Aufgabe des Zugriffsschutzes der über die VAU einstellbaren und abholbaren E-Rezepte fokussieren.

Um die Verbindungsstrecke zwischen Webschnittstelle und E-Rezept-VAU in Bezug auf Vertraulichkeit zu schützen, wird eine Verschlüsselung auf Anwendungsebene eingeführt. Bei ePA ist dies das VAU-Protokoll. Beim E-Rezept kann aufgrund der andersartigen Anwendungslogik in der E-Rezept-VAU ein einfacheres Sicherungsverfahren verwendet werden. Dieses ist in Abschnitt5- VAU-Protokoll für E-Rezept normativ definiert.

A_22698 -E-Rezept, Erzeugung des Nutzerpseudonyms LEI
Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

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

[<=]

Erläuterung zu A_22698-*:
Der Pseudonymisierungsschlüssel kann auch nur in Software vorliegen – muss also nicht zwangsweise in einem HSM vorliegen.
Für die Unterstützung von betrieblichen Prozessen soll dem E-Rezept-Projekt ein Überblick über die Anzahl der aktuell im Feld befindlichen Primärsystem-Versionen zur Verfügung gestellt werden. Der E-Rezept-FD übermittelt die Pseudonyme als Teil der Rohdatenlieferung an die gematik.

3.16.1 E-Rezept-spezifische TLS-Vorgaben (Internet)

A_27855-01 -E-Rezept: Zugriff auf Webdienste - Webzertifikat aus Internet PKI
Der E-Rezept-Fachdienst MUSS bei Aufbau einer HTTPS-Verbindung zu Diensten im Internet sicherstellen, dass das zu prüfende Zertifikat auf ein CA-Zertifikat der [CA/B Forum] Baseline Requirements per Signaturkette kryptographisch rückführbar ist.**[<=]**

Hinweis: Die Anforderung drückt aus, dass das zu prüfende Zertifikat Teil der Internet PKI ist.

A_27856-01 -E-Rezept: Zugriff auf Webdienste - Hostname-Prüfung für TLS-Server-Zertifikat durchführen
Der E-Rezept-Fachdienst MUSS bei Aufbau einer HTTPS-Verbindung zu Diensten im Internet sicherstellen, dass der angesprochene Hostname mit dem im X.509-Zertifikat des Ziel-Webdienstes angegebenen Common Name (CN) oder den Subject Alternative Names (SANs) gemäß RFC 6125 übereinstimmt und bei Nichtübereinstimmung den Verbindungsaufbau abbrechen. **[<=]**

A_27857-01 -E-Rezept: Zugriff auf Webdienste - Sperrprüfung für TLS-Server-Zertifikat durchführen
Der E-Rezept-Fachdienst MUSS vor Nutzung des Ziel-Webdienstes im Internet gemäß RFC 5280 prüfen, ob das präsentierte TLS-Server-Zertifikat gesperrt wurde (z. B. mittels OCSP oder CRL) und im Fall einer Sperrung oder bei nicht durchführbarer Sperrprüfung die Verbindung abbrechen.**[<=]**

A_27858-01 -E-Rezept: Zugriff auf Webdienste - TLS-Protokollversion und Cipher Suites
Der E-Rezept-Fachdienst MUSS für die Verbindung zu Webdiensten im Internet mindestens TLS-Versionen 1.2 oder höher verwenden und ausschließlich Cipher Suites mit entsprechenden Domain Parametern nach [BSI TR-02102-2] verwenden. **[<=]**

3.17 KOM-LE-spezifische Vorgaben

Bei KOM-LE werden E-Mail-Anhänge, deren Gesamtgröße 15 MiB überschreitet, separat symmetrisch verschlüsselt und das Chiffrat auf dem "Fachdienst Download-Server (KAS)" abgelegt. Das Chiffrat erhält eine ID, die aus dem Hashwert des Chiffrats gebildet wird. Dabei ist die in A_19644-* festgelegte Hashfunktion zu verwenden. Der verwendete symmetrische Schlüssel und die Hashwert-Referenz sind dann Teil des Klartextes der verschlüsselten E-Mail-Nachricht (KOM-LE). Die Chiffrate auf dem Download-Server (KAS) werden automatisch nach einer bestimmten im FD festgelegten Zeit gelöscht.

A_19644-01 -Hashfunktion für Hashwert-Referenzen beim Fachdienst Download-Server (KAS)
Ein KOM-LE-Client MUSS bei der Erzeugung und Verwendung von Hashwert-Referenzen für Anhänge - die auf dem Fachdienst Download-Server (KAS) abgelegt werden - die Hashfunktion SHA-256 [FIPS-180-4] verwenden.**[<=]**

3.18 Kryptographisch gesicherte VSDM-Prüfziffer Version 1

A_23460 -VSDM-Betreiber: HMAC-Schlüsselerzeugung
Ein Betreiber eines VSDM-Dienstes MUSS den HMAC-Sicherungsschlüssel für die kryptographische Sicherung der VSDM-Prüfziffern zufällig mit einer Länge von 256 Bit (= 32 Byte) und einer Mindestentropie von 120 Bit erzeugen.**[<=]**

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

A_23461 -VSDM-Betreiber: HMAC-Verfahren
Ein Betreiber eines VSDM-Dienstes MUSS die HMAC-Sicherung der VSDM-Prüfziffern das HMAC-Verfahren aus [RFC-2104] mit der Hashfunktion SHA-256 (also nicht wie im RFC beschrieben mittels SHA-1) verwenden. Für das dabei zu verwendende geheime Schlüsselmaterial gilt A_23460-*.**[<=]**

Beispiel:

Wenn der geheime HMAC-Schlüssel (hexdump) 3a8e0064436bf2dbe7ca41ec6f1ed60beec083bc4100633281eb397cb294391c ist, so ist der HMAC-SHA-256-Wert gemäß A_23461-* auf die leere Bytefolge folgende Bitfolge (hexdump)4c0a04f65d498113a5df2ab388d99d2c0bc6224a662b1ce529342745e7af414a

A_23463 -VSDM-Betreiber: verschlüsselter Export des HMAC-Schlüssels für die E-Rezept-VAU
Ein Betreiber eines VSDM-Dienstes MUSS den HMAC-Sicherungsschlüssel mittels des ECIES-Verfahrens [SEC1-2009] für den Export an den E-Rezept-FD verschlüsseln und dabei folgende Vorgaben umsetzen

1. 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.
2. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von SHA-256 verwenden.
3. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in der Formulierung von [RFC-5869] info="ecies-vau-transport" .
4. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key für die Verwendung von AES/GCM ableiten.
5. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig erzeugen.
6. Er MUSS mit dem CEK und dem IV mittels AES/GCM p verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
7. Er MUSS das Ergebnis wie folgt kodieren: chr(0x01) || <32 Byte X-Koordinate von öffentlichen Schlüssel aus (a) > || <32 Byte Y-Koordinate> || <12 Byte IV> || <AES-GCM-Chiffrat> || <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.

[<=]

Beispiel:

Sei folgendes Zertifikat ein Beispiel-E-Rezept-VAU-Verschlüsselungszertifikat:

```
-----BEGIN CERTIFICATE-----
MIICWzCCAgKgAwIBAgIUYVkTweOLq4G7U28Gwwb3tAghcW8wCgYIKoZIzj0EAwIw
gYIxCzAJBgNVBAYTAkRFMQ8wDQYDVQQIDAZCZXJsaW4xDzANBgNVBAcMBkJlcmxp
bjEQMA4GA1UECgwHZ2VtYXRpazEQMA4GA1UECwwHZ2VtYXRpazEtMCsGA1UEAwwk
RS1SZXplcHQtVkFVIEJlaXNwaWVsaW1wbGVtZW50aWVydW5nMB4XDTIyMTAwNDIx
NTg1OFoXDTIzMTAwNDIxNTg1OFowgYIxCzAJBgNVBAYTAkRFMQ8wDQYDVQQIDAZC
ZXJsaW4xDzANBgNVBAcMBkJlcmxpbjEQMA4GA1UECgwHZ2VtYXRpazEQMA4GA1UE
CwwHZ2VtYXRpazEtMCsGA1UEAwwkRS1SZXplcHQtVkFVIEJlaXNwaWVsaW1wbGVt
ZW50aWVydW5nMFowFAYHKoZIzj0CAQYJKyQDAwIIAQEHA0IABB1wleU9eglD8Q7j
7HuTVyG7KJIRoJ6Rt1x8+GHvb3xeht3f5RPeETi+py3tLK5RWhFeGOIKmnydoBxJ
mK6VcmijUzBRMB0GA1UdDgQWBBTyOvu1E6xmEMu/6KIhI/kQJYw9CTAfBgNVHSME
GDAWgBTyOvu1E6xmEMu/6KIhI/kQJYw9CTAPBgNVHRMBAf8EBTADAQH/MAoGCCqG
SM49BAMCA0cAMEQCIGgWYuADP/Aulq0/QIzJDkCgqAXlVvmr92UPbfcRClvNAiAc
w8VE8xBtMx9g5sLK/HiDCsrQ9TNZjwpb5rb1t4ozgg==
-----END CERTIFICATE-----
```

und folgendes der dazugehörige private Schlüssel:

```
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQc=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHgCAQEEII1+MNpxZfd+cPjE1Z5DCHCnuGCQt5MA6U55yuaxoB8CoAsGCSskAwMC
CAEBB6FEA0IABB1wleU9eglD8Q7j7HuTVyG7KJIRoJ6Rt1x8+GHvb3xeht3f5RPe
ETi+py3tLK5RWhFeGOIKmnydoBxJmK6Vcmg=
-----END EC PRIVATE KEY-----
```

Weiterhin sei folgendes der zu verschlüsselnde HMAC-Schlüssel (hexdump)

```
3a8e0064436bf2dbe7ca41ec6f1ed60beec083bc4100633281eb397cb294391c
```

Dann ist folgendes der Hexdump eines Chiffrats nach A_23463

```
019dec56554203624c214ac5321c798b78281d3fff4a0a4e187319ebebeba2ace40fe71913c
6e02998cd3cd6f8268a0daf9f40eb1561541de9868d46eb913e3f3157d5ed6bd9d4aef792d4
73be5fa2868dbcd228e5a603afdae2c69fc459e656513cff835934de5e2f159b772ebbd21d6
dfef2a4960ad829b968fc974b
```

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

3.19 Kryptographisch gesicherte VSDM-Prüfziffer Version 2

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

A_27274 -VSDM-Anbieter: jährliche Erzeugung des gemeinsamen Geheimnisses
Ein Anbieter eines VSDM-Dienstes MUSS (min.) jährlich ein Geheimnis für die kryptographische Sicherung der VSDM-Prüfziffern zufällig mit einer Länge von 256 Bit (= 32 Byte) und einer Mindestentropie von 120 Bit erzeugen.**[<=]**
[<=]

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

Die erzeugten Geheimnisse müssen für die prüfenden Systeme E-Rezept-VAU und ePA-Aktensystem-VAU (Plural) verschlüsselt (A_27275-*) überführt werden. Dafür gibt es im Kontext Prüfziffer Version 1 einen etablierten Prozess, bei dem die Authentizität und Integrität der verschlüsselten Geheimnisse sichergestellt wird. Dieser Prozess wird unverändert weitergeführt auch für den sicheren Transport der gemeinsamen Geheimnisse im Kontext Prüfziffer 2.

A_27275 -VSDM-Anbieter: verschlüsselter Export des gemeinsamen Geheimnisses für Prüfziffer prüfende Fachdienste-VAUs
Ein Anbieter eines VSDM-Dienstes MUSS das gemeinsame Geheimnis (vgl. A_27274-*) mittels des ECIES-Verfahrens [SEC1-2009] für den Export an die VAUs der prüfenden Fachdienste (E-Rezept-FD, ePA-Aktensysteme) verschlüsseln und dabei folgende Vorgaben umsetzen

1. Er MUSS ein ephemeres ECDH-Schlüsselpaar erzeugen, auf der Kurve des EE-Schlüssels aus dem Verschlüsselungszertifikat des Empfängers (VAUENC) -- also P-256 oder brainpoolP256r1, 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.

2. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von SHA-256 verwenden.
3. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in der Formulierung von [RFC-5869] info="ecies-vau-transport" .
4. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key für die Verwendung von AES/GCM ableiten.
5. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig erzeugen.
6. Er MUSS mit dem CEK und dem IV mittels AES/GCM den Klartext verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
7. Er MUSS das Ergebnis wie folgt kodieren: chr(0x01) || <32 Byte X-Koordinate vom öffentlichen Schlüssel aus "1." > || <32 Byte Y-Koordinate> || <12 Byte IV> || <AES-GCM-Chiffrat> || <16 Byte AuthenticationTag> (vgl. auch Tab_KRYPT_ERP und folgende die Beispielverschlüsselung).
   Die Koordinaten sind (wie üblich) vorne mit chr(0) zu padden solange bis sie eine Kodierungslänge von 32 Byte erreichen.

[<=]

Hinweis: Die gematik stellt Beispiel-Code bereit.

A_27276 -VSDM: Export-Paket
Ein VSDM-FD MUSS bei der Erstellung der Export-Pakete für den Export der gemeinsamen Geheimnisse (vgl. A_27274-*) an die prüfenden Systeme (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) die gleiche Export-Paket-Struktur verwenden wie im Kontext Prüfziffer Version 1.**[<=]**

Beispiel für ein Export-Paket

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

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

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

Die gematik stellt Beispiel-Code für die Erzeugung eine Export-Pakets gemäß A_27276-* bereit.

A_27356 -VSDM: explizite Angabe der Geheimnis-Version bei Erzeugung
Ein VSDM-FD MUSS es einem VSDM-FD-Anbieter ermöglichen:

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

[<=]

Erläuterung zu A_27356-*: Als Beispiel verwendet ein VSDM-FD die Schlüsselversion 3, keine anderen Schlüsselversionen liegen im VSDM-FD vor. Dann kommt es zur (im Regelfall jährlichen) Erneuerung. Dabei muss ein Anbieter angeben können, dass das neu erzeugte Schlüsselmaterial die Version "1" haben soll. Dafür wird ein Export-Paket erzeugt und die Geheimnisse wurden in die ePA-Aktensysteme und den E-Rezept-FD erfolgreich importiert. Dann muss der Anbieter des VSDM-FD konfigurieren können: ab jetzt soll Version "1" aktiv sein (Version "3" ist dann inaktiv). Nach erfolgreicher Funktionsüberprüfung in den ePA-Aktensystemen und im E-Rezept-FD wird im VSDM-FD Version "3" gelöscht.

A_27277 -VSDM-Anbieter: Schlüsselwechsel
Ein Anbieter eines VSDM-Dienstes MUSS folgendes sicherstellen.

1. Nach der jährlicher Erneuerung des gemeinsamen Geheimnisses (vgl. A_27274-*) MUSS dieses mittels A_27275-* und A_27276-* und des schon für die Prüfziffer Version 1 etablierten Prozess jeweils für die prüfenden Systeme (E-Rezept-VAU und ePA-Aktensystem-VAUs) verschlüsselt und an diese übermittelt werden.
2. Das gemeinsame Geheimnis MUSS zunächst "inaktiv" sein.
3. Erst nach erfolgreichen Import durch alle prüfenden Systeme MUSS das gemeinsame Geheimnis aktiviert werden und mittels A_27286-* der entsprechende AES/GCM-Schlüssel abgeleitet werden. Dieser ist dann aktiv und der jüngste AES/GCM-Schlüssel und MUSS nach A_27278-* für die Erzeugung der Prüfziffern Version 2 verwendet werden.

[<=]

A_27286 -VSDM-FD: Ableitung des AES/GCM-Schlüssel für die Sicherung der Prüfziffern Version 2 aus dem gemeinsamen Geheimnis
Ein VSDM-FD MUSS nach der Erzeugung eines gemeinsamen Geheimnisses (vgl. A_27274-*), mit dem Geheimnis eine Schlüsselableitung durchführen. Mittels der HKDF nach [RFC-5869] auf Basis von SHA-256 und dem Ableitungsvektor "VSDM+ Version 2 AES/GCM" ein 128 Bit (=16 Byte) Wert (Bitfolge) abgeleitet werden. Diese 128 Bit Bitfolge MUSS als AES/GCM-Schlüssel für die Erzeugung der Prüfziffern Version 2 gemäß A_27278-* verwendet werden.

D. h. Es wird nach A_27274-* ein neuer Schlüssel mindestens jährlich erzeugt, der dann die Versionsnummer x habe. Anschließend erfolgt einmalig eine Schlüsselableitung wie in A_27286-* definiert. Damit wird der AES/GCM-Schlüssel-Version x erzeugt. Nach "Aktivierung" (vgl. A_27277-*) wird dieser gemäß A_27278-* verwendet um Prüfziffern zu erzeugen.**[<=]**

Hinweis: Die gematik stellt Beispiel-Code zur Verfügung.

Beispiel:

Sei das gemeinsame 256-Bit lange Geheimnis (hexdump):

000000000000000000000000000000000000000000000000000000000000001

dann ist der nach A_27286-* abgeleitete 128-Bit lange AES/GCM-Schlüssel (hexdump):

b453cd39ea09dbc3a4ff47ebc8bbbfb2

A_27323 -VSDM-FD: relative Zeit (Zeit-offset) Prüfziffer Version 2
Ein VSDM-FD MUSS als Offset für die Zeitkodierung (iat) bei der Erstellung der Prüfziffer Version 2 (A_27278-*) folgende Vorgabe verwenden:

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

[<=]

Erläuterung zu A_27323-*: Für die Kodierung von Daten in der Prüfziffer Version 2 stehen nur 18 Byte zur Verfügung. Deshalb wird die iat-Zeit nicht wie bei der Unix-Zeit üblich von 1.1.1970 00:00:00 (UTC) startend kodiert, sondern vor der Kodierung in der Prüfziffer die für die Kodierung notwendige Bitlänge durch Subtraktion eines entsprechenden Offsets reduziert.

Die Kodierungslänge von iat wird wie in A_27278-* definiert sogar nochmal um 3 Bit reduziert (r_iat_8).

A_27352 -VSDM-Prüfziffer Version 2: Erzeugung von hcv
Ein den hcv-Wert (Hash Check Value) erzeugendes System (VSDM-FD oder Primärsystem) MUSS bei der Erzeugung des hcv-Wertes, wie folgt vorgehen:

1. Es sei VB gleich der Versicherungsbeginn (UC_AllgemeineVersicherungsdatenXML.Versicherter.Versicherungsschutz.Beginn, https://github.com/gematik/api-telematik/blob/OPB5/fa/vsds/Schema_VSD.xsd). VB MUSS keine Leerzeichen enthalten. (siehe Erläuterungen nach A_27352-*)
2. Falls der Versicherte eine "StrassenAdresse" (vgl. XML-Schema) und darin ein nichtleeres Element "Strasse" besitzt, dann sei SAS gleich der Wert in diesem Element. Andernfalls sei SAS="" (leere Zeichenfolge). Ggf. Führende oder endende Leerzeichen MÜSSEN entfernt werden. Die Kodierung der Inhalte von "Strasse" MUSS ISO-8859-15 (Latin-9) sein. (siehe Erläuterungen nach A_27352-*)
3. Sei SHA-256 wie in [FIPS-180-4] definiert.
4. Sei H = SHA-256(VB || SAS).
5. Sei H_40 die ersten 5 Bytes (40 Bit) von H.
6. Von H_40 setzt man das MSBit im ersten Byte auf 0, dass Ergebnis sei H_40_0.

Der hcv-Wert ist gleich H_40_0.

[<=]

Erläuterungen zu A_27352-*: Die Versicherten-Daten müssen nach Spezifikation VSDM [gemSpec_eGK_Fach_VSDM], in ISO-8859-15 (Latin-9) vom eGK-Personalisierer und vom VSDM-FD kodiert eingebracht werden ("Die persönlichen Versichertendaten PD [...]. Der zu verwendende Zeichensatz für die fachlichen Inhalte ist ISO8859-15."). Der Konnektor

(genauer das VSDM-Fachmodul im Konnektor) verändert diese Kodierung nicht. D. h. die Versicherten-Daten, die ein Primärsystem über ReadVSD erhält, sind schon in der (im Sinne von A_27352-*) "korrekten" Zeichenkodierung und müssen ohne Umkodierung für die Hashwert-Erzeugung verwendet werden.

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

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

Beispiele:

VB	SAS	Data-to-be-hashed (hexdump)	H_40_0-Wert (hexdump)
20190212	(leere Zeichenkette)	3230313930323132	4885ee8394
19981123	Berliner Straße	3139393831313233426526c696e65722053747261df65	6545491d14
19841003	Angermünder Straße	313938343130303341 6e6765726dfc6e6465722053747261df65	7cc49e7af4
20010119	Björnsonstraße	32303031303131394 26af6726e736f6e73747261df65	186269e4f7
20040718	Schönhauser Allee	32303034303731385 36368f66e6861757365722041 6c6c6565	353646b5c8

Ein Primärsystem muss bei der Befugniserstellung ein hcv-Wert (H_40_0) in ein JWT einbetten (vgl. [gemSpec_Aktensystem_ePAfueralle#A_24590-*]) dafür muss der 5 Byte lange hcv-Wert per Base64 kodiert werden. Beispiel:

VB	SAS	hcv-Wert base64-kodiert für das JWT
20190212	(leere Zeichenkette)	SIXug5Q=
19981123	Berliner Straße	ZUVJHRQ=
19841003	Angermünder Straße	fMSeevQ=
20010119	Björnsonstraße	GGJp5Pc=
20040718	Schönhauser Allee	NTZGtcg=

Fünf Bytes Binärdaten (hcv-Wert) per Base64 zu kodieren, ergibt eine 8 Zeichen lange Bytefolge (vgl. Tabelle). Diese Bytefolge ist dann im JWT als hcv-Wert einzutragen.

A_27278 -VSDM-FD: Struktur einer Prüfziffer der Version 2
Ein VSDM-FD MUSS zunächst folgende innere Datenstruktur (Klartext) erstellen:

Name	Länge	Festlegungen und Erläuterung
I_Feld_1	5	In diesem Feld sind Sperrinformationen und ein gekürzter Hashwert kodiert. Sperrinformation: Falls die eGK ungültig/gesperrt ist, sei S=128, anderenfalls sei S=0. Hashwert: Der hcv-Wert MUSS wie in A_27352-* definiert berechnet werden. Und wird im Folgenden als H_40_0 bezeichnet. Sei H_40_0[0] das erste Byte von H_40_0 und H_40_0[1:] alle restlichen Bytes von H_40_0. Dann ist I_Feld_1 = (H_40_0[0] | S) || H_40_0[1:]. (Erläuterung zum besseren Verständnis: das MSBit im ersten Byte von H_40 wird auf 0 gesetzt (A_27352-*, Schritt 6) und man erhält H_40_0. Anschließend wird die Sperrinformation auf das erste Byte aufaddiert. D. h. wenn von I_Feld_1 das MSBit des ersten Bytes 1 ist, dann ist die eGK ungültig/gesperrt.)
r_iat_8	3	Sei iat die aktuelle Unix-Zeit (UTC) in Sekunden (also keine Nachkommastellen) im VSDM-FD zum Erzeugungszeitpunkt der Prüfziffer. Dann MUSS r_iat_8 = (iat - iat_offset) >> 3 Alle Zahlenwerte MÜSSEN in Network-Byte-Order (= Byte-Order Big) kodiert werden. Alle Zeiten sind wie bei der Unix-Zeit üblich UTC. Hinweis: für iat_offset vgl. A_27323-*. Beispiel: Wird die Prüfziffer um "2025-01-02T00:00:00+00:00" = 1735776000 erzeugt, dann ist r_iat = (1735776000 - 1735689600) >> 3 = 10800
KVNR	10	10 Stellige KVNR, ASCII-kodiert (Beispiel: A123456789)

Der Klartext hat damit eine Länge von 18 Byte.

Name	Länge	
Feld_1	1	In diesem Feld sind drei Informationen kodiert:

		(1) Kennzeichnung für Version 2 der Prüfziffer Sei V = 128. (2) Betreiberkennung Die Betreiberkennung (wie bspw. im Export-Paket (A_27276-*) übertragen) ist zunächst ein Buchstabe von 'A' bis 'Z'. Sei BK diese Betreiberkennung als ASCII-Zeichen. Sei BK_D = BK - 65 und sei BK_D_4 = BK_D << 2. (3) Geheimnis/Schlüssel-Version Sei SV gleich die Geheimnis/Schlüssel-Version, die für die Verschlüsselung des Klartextes (siehe Tabelle oben) verwendet wird. SV wird binär kodiert, bspw. wenn SV = 2 ist, so ist SV kodiert \x02. SV MUSS kleiner 4 sein (vgl. auch A_27356-*). Sei Feld_1 = V + BK_D_4 + SV Beispiel: Sei die Betreiberkennung gleich 'B' und die Schlüssel-Version gleich 2, dann ist BK_D_4 gleich 4. Und damit Feld_1 = 128 + 4 + 2 = 134.
Intitialisierungsvektor für AES/GCM	12	wie bei AES/GCM üblich MUSS der 96 Bit lange IV (= 12 Bytes) pro Verschlüsselung zufällig über eine kryptographisch hochwertige Zufallsquelle erzeugt werden
eigentliches Chiffrat	18	mittels AES/GCM verschlüsselter Klartext (innere Datenstruktur, s. o.) mit dem jüngsten aktivierten AES/GCM-Schlüssel (vgl. A_27286-*)
AES/GCM Authentication-Tag (GMAC)	16	128-Bit Authentication-Tag, der bei der AES/GCM entsteht (berechnet wird)

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

[<=]

Hinweise zu A_27278-*: Am ersten Byte (Feld_1) kann man eine Prüfziffer Version 1 (beginnt immer mit Zeichen aus dem Intervall [0x4a, 0x5a]) und eine Prüfziffer Version 2 eindeutig unterscheiden. Wenn das erste Byte von Feld_1 kleiner als 128 ist, so muss es eine Prüfziffer der Version 1 sein, anderenfalls ist es eine Prüfziffer der Version 2.

Aktuell (Januar 2024) besitzen alle gemeinsamen Geheimnisse in den VSDM-Fachdiensten die Versionsnummer 2. Mit der nächsten regulären Erneuerung (Q2/Q3 2025, A_27274-*) wird die Versionsnummer 3 verwendet, usw..

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

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

A_27299 -VSDM-Prüfziffer Version 2: prüfenden Systeme, Import der gemeinsamen Geheimnisse und AES/GCM-Schlüsselableitung
Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-FD-VAU, ePA-Aktensystem-VAU/VAU-HSM etc.) MUSS Export-Pakete gemäß A_27276-* importieren. Es werden dabei gemeinsame Geheimnisse gemäß A_27274-* importiert. Diese MUSS die im Export-Paket aufgeführte Betreiberkennung und Version zugeordnet werden. Mit dem gemeinsamen Geheimnis MUSS das prüfende System eine Schlüsselableitung gemäß A_27286-* durchführend und dem erhaltenen AES/GCM-Schlüssel die Betreiberkennung und Version des gemeinsamen Geheimnisses (also aus dem entsprechenden Import) zuordnen.

Anschließend MÜSSEN die AES/GCM über Betreiberkennung und Version (vgl. Kodierung der beiden Werte in einer Prüfziffer Version 2 in A_27278-*) im prüfenden System verfügbar/adressierbar sein.

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

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

A_27342 -Konfigurationsvariable enforce_hcv_check
Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) MUSS eine Konfigurationsvariable `enforce_hcv_check` besitzen, die standarmäßig auf `false` gesetzt ist.**[<=]**

A_27279 -VSDM-Prüfziffer Version 2: Prüfung und Entschlüsselung
Ein die Prüfziffer Version 2 prüfendes System (E-Rezept-VAU, ePA-Aktensystem-VAU-HSM etc.) MUSS folgende Prüfungen der Prüfziffer Version 2 vornehmen. Ergibt eine der Prüfungen ein nicht-positives Prüfergebnis, so MUSS die Prüfziffer als ungültig abgelehnt werden.

1. Prüfung: besitzt die Prüfziffer eine Länge von 64 Bytes.
2. Prüfung: kann die Prüfziffer erfolgreich base64-dekodiert werden.
   Die nun erhaltene erfolgreich base64-dekodierte 47 Byte lange Bytefolge wird als dtbc (data to be checked) im Folgenden bezeichnet.
3. Prüfung: ist das erste Byte von dtbc größer als 128 (das MSBit ist also auf 1 gesetzt). Hinweis: ansonsten handelt es sich um eine Prüfziffer Version 1 (für die Prüfung in A_27279-* also nicht geeignet).
4. Prüfung: gibt es im prüfenden System einen AES/GCM-Schlüssel mit der im ersten Byte von dtbc aufgeführter Betreiberkennung und aufgeführter Version (vgl. A_27299-* und A_27278-*).
5. Die folgenden 12 Byte in dtbc werden als IV bezeichnet. Die darauf folgenden 18 Bytes werden als ciphertext bezeichnet. Die darauf folgenden 16 Bytes werden als Authentication-Tag bezeichnet.

Prüfung: ist die AES/GCM-Entschlüsselung erfolgreich mittels des in Schritt 4 identifizierten AES/GCM-Schlüssels und mit IV, ciphertext, Authentication-Tag
D. h. gibt es gerade kein Symbol FAIL als Ergebnis der AES/GCM-Entschlüsselung -- die Authentizität und Integrität des Chiphertexts/Klartexts ist damit festgestellt.
Im folgenden wird der erfolgreich entschlüsselte Klartext betrachtet.

6. Prüfung: ist das MSBit im ersten Byte des Klartextes gleich 0 (ansonsten ist die eGK gesperrt).
7. Prüfung zeitliche Gültigkeit der Prüfziffer:
   Sei, wie in A_27278-*, definiert r_iat_8 die Bytefolge von Byte-Offset 5 bis inkl. Byte-Offset 7 (also 3 Byte groß) aus dem Klartext. r_iat_8 MUSS im Network-Byte-Order (= Byte-Order Big) kodierte unsignierte Zahl interpretiert werden.
   Zunächst MUSS der Wert `iat` mit `iat = (r_iat << 3) + iat_offset` (vgl. A_27323-*) berechnet werden.
   Anschließend MUSS anwendungsspezifisch `iat` mit der aktuellen Zeit überprüft werden:
   ePA:A_24573-* (20 Minuten Fenster)
   E-Rezept: A_23451-* (30 Minuten Fenster)
8. Prüfung hcv:
   Im folgenden werden die ersten 5 Byte des Klartextes als H_40_0 bezeichnet.
   Wenn vom Primärsystem ein hcv-Wert übergeben wurde, prüfe ob H_40_0 gleich dem vom Primärsystem übergebenen hcv-Wert ist (vgl. A_24590-*).
   Wenn vom Primärsystem kein hcv-Wert übergeben wurde: Wenn `enforce_hcv_check` (vgl. A_27342-*) auf true gesetzt ist, dann FAIL, anderen falls OK.
   (Der hcv-Wert aus A_24590-* MUSS vor dem Vergleich base64-dekodiert werden.)
9. Die letzten 10 Byte des Klartextes werden als KVNR bezeichnet.
   Prüfung: ist die KVNR aus dem Klartext gleich der KVNR, die im Anwendungskontext erwartet wird.

[<=]

Hinweis zu A_27279-*: Je nach Anwendung und Implementierung (ePA oder E-Rezept) werden Teile der Anforderung im VAU-HSM und in der VAU erbracht.

Nur als Verständnishinweis: bei Prüfschritt 8 (hcv-Wertprüfung) ist die Prüfreihenfolge motiviert durch Abhängigkeiten im Umsetzungsplan. Erst wenn enforce_hcv_check auf True gesetzt wird, ist die Prüfung aus Sicherheitssicht effektiv.

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

3.20 spezifische TLS-Vorgaben für VSDM

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

A_23912 -VSDM: Unterstützung von NIST-Kurven beim ephemeren ECDH (TLS-Handshake)
Ein VSDM KANN auf die Unterstützung von NIST-Kurven beim ephemeren ECDH (TLS-Handshake) bei der Umsetzung von GS-A_4384-* und A_17124-* verzichten. Er MUSS in diesem Fall die in GS-A_4384-* und A_17124-* anderen aufgeführten ECC-Gruppen

(brainpoolP256r1 und brainpoolP384r1) unterstützen.
[<=]

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

A_23913 -Intermediär: TLS, Kurven beim ephemeren ECDH (TLS-Handshake)
Ein Intermediär MUSS bei der Umsetzung von GS-A_4384-* und A_17124-* die Kurven brainpoolP256r1 und brainpoolP384r1 beim ephemeren ECDH (TLS-Handshake) unterstützen.**[<=]**

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

3.21 NCPeH

Im NCPeH-FD werden medizinische Daten ausschließlich innerhalb von VAU-Instanzen verarbeitet. Dabei verarbeitete Daten dürfen nur kryptographisch gesichert außerhalb einer VAU-Instanz persistiert werden -- insbesondere sind hierbei Audit-Trail-Daten zu betrachten (A_28365-*). Die kryptographischen Vorgaben dafür sind die gleichen wie bei ePA.

A_28522 -NCPeH-FD, Persistierung von Daten außerhalb einer VAU, authenticated encryption
Ein NCPeH-FD MUSS für die Persistierung von Daten außerhalb der VAU (bspw. Audit-Trail-Daten) die zu persistierenden Daten mittels des Authenticated-Encryption-Verfahrens AES/GCM [FIPS-197], [NIST-SP-800-38D] verschlüsseln. Dabei MUSS eine Schlüssellänge von 256 Bit und eine Authentication-Tag Länge von 128 Bit verwendet werden. Die Initialisierungsvektoren (IV) MÜSSEN bei jeder Verschlüsselung zufällig mit einer Länge von 96-Bit erzeugt werden und bei der Verschlüsselung verwendet werden.**[<=]**

A_28523 -NCPeH-FD, Schlüsselableitung für die Persistierung von Daten außerhalb einer VAU
Ein NCPeH-FD MUSS für die Persistierung von Daten außerhalb der VAU (bspw. Audit-Trail-Daten) eine versichertenindividuelle Schlüsselableitung durchführen auf Basis der dafür vorgesehenen Masterkeys (vgl. A_28372-*). Dabei MUSS zur Schlüsselableitung die HKDF nach [RFC-5869] auf Basis von SHA-256 [FIPS-180-4] verwendet werden. Als Ableitungsvektor (info-Parameter in [RFC-5869]) MUSS die KVNR der versicherten Person verwendet werden, dessen Daten persistiert werden. Die abgeleiteten Schlüssel MÜSSEN nach A_28522-* verwendet werden für die Verschlüsselung der zu persistierenden Daten.
[<=]

4 Migration 120-Bit-Sicherheitsniveau

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

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

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

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

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

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

4.1 PKI-Begriff Schlüsselgeneration

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

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

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

4.2 X.509-Root der TI

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

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

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

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

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

```
(0x377434509adcbb827f74acd7adf0ce72aa28ddc53be3f15ea8023a9b0722c09d,
0x5364a99686c02092bbf9efde9878847b90f09d90b7ac4193553820258a58dfd5)
```

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

```
MFowFAYHKoZIzj0CAQYJKyQDAwIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgCOpsH
IsCdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39U=


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

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

```
-----BEGIN CERTIFICATE-----
MIICajCCAg+gAwIBAgIBATAKBggqhkjOPQQDAjBtMQswCQYDVQQGEwJERTEVMBMG
A1UECgwMZ2VtYXRpayBHbWJIMTQwMgYDVQQLDCtaZW50cmFsZSBSb290LUNBIGRl
ciBUZWxlbWF0aWtpbmZyYXN0cnVrdHVyMREwDwYDVQQDDAhHRU0uUkNBMzAeFw0x
NjEyMDkwODQxNTZaFw0yNjEyMDcwODQxNTZaMG0xCzAJBgNVBAYTAkRFMRUwEwYD
VQQKDAxnZW1hdGlrIEdtYkgxNDAyBgNVBAsMK1plbnRyYWxlIFJvb3QtQ0EgZGVy
IFRlbGVtYXRpa2luZnJhc3RydWt0dXIxETAPBgNVBAMMCEdFTS5SQ0EzMFowFAYH
KoZIzj0CAQYJKyQDAwIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgC
OpsHIsCdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39WjgZ4wgZswHQYD
VR0OBBYEFBERSneTkJZDKt3uLzjddI870TMmMEIGCCsGAQUFBwEBBDYwNDAyBggr
BgEFBQcwAYYmaHR0cDovL29jc3Aucm9vdC1jYS50aS1kaWVuc3RlLmRlL29jc3Aw
DwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwFQYDVR0gBA4wDDAKBggq
ghQATASBIzAKBggqhkjOPQQDAgNJADBGAiEApQ6qGHTx97IsdzgoWH9/W32yt4rk
udUis0xxGZ48YOUCIQCTQ4puol5YYIAZYk74mfid3JBOvMBV/XgPV2WpS/99yg==
-----END CERTIFICATE-----
```

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

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

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

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

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

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

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

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

die Signatur überprüfen und erhält als Ausgabe:

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

```
error 18 at 0 depth lookup:self signed certificate
OK
```

4.3 TSL-Dienst und ECDSA-basierte TSL allgemein

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

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

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

4.4 ECC-Unterstützung bei TLS

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

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

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

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

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

A_17124-03 -TLS-Verbindungen (ECC-Migration)
Alle Produkttypen, die Übertragungen mittels TLS durchführen, MÜSSEN die folgenden Vorgaben erfüllen:

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

[<=]

A_17775 -TLS-Verbindungen Reihenfolge Ciphersuiten (ECC-Migration)
Alle Produkttypen, die Übertragungen mittels TLS durchführen und in der Rolle TLS-Server agieren, SOLLEN die Reihenfolge der Ciphersuiten in der Liste "cipher_suites" aus dem TLS-ClientHello bei der Auswahl der Ciphersuite befolgen.
[<=]

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

A_17322 -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.

4.5 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. Auszuhandeln 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, SAi1, KEi, Ni  -->
```

[RFC-7296]. In KEi 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_KE_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 einen "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_KE_PAYLOAD-Nachricht erhält, wird dann auf die Vorgaben GS-A_4382-* "zurückfallen".

A_22342 -Konnektor, IKE-Schlüsselaushandlung – Erleichterung Migrationsphase 1 (ECC-Migration)
Solange ein Konnektor nur mit einem RSA-Zertifikat am VPN-Zugangsdienst registriert ist, KANN der Konnektor den IKE-Verbindungsaufbau gemäß der Vorgaben aus GS-A_4382-* durchführen.**[<=]**

A_22343-02 -Verwendung von ECC beim Verbindungsaufbau nach RE-Registrierung mit ECC-NK-Zertifikat (ECC-Migration)
Sobald der Konnektor mit einem ECC-Zertifikat am VPN-Zugangsdienst registriert ist, MUSS er vom nächsten regulären Verbindungsaufbau zum VPN-Konzentrator an, immer den Verbindungsaufbau zuerst gemäß der Vorgaben aus A_17125-* durchführen. Nur bei Scheitern eines Verbindungsaufbaus basierend auf dem ECC-NK-Zertifikat MUSS der Konnektor im Falle einer bestehenden Registrierung mit einem RSA-NK-Zertifikat den Verbindungsaufbau gemäß der Vorgaben aus GS-A_4382-* erneut versuchen.**[<=]**

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

A_17125 -IKE-Schlüsselaushandlung für IPsec (ECC-Migration)
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:

1. Zur Authentisierung MUSS eine Identität mit einem X.509-Zertifikat gemäß [gemSpec_Krypt#GS-A_4360-*] Schlüsselgeneration "ECDSA" verwendet werden.
2. Für „Hash und URL“ MUSS SHA-1(vgl. [RFC-7296#3.6]) verwendet werden.

3. Für den Schlüsselaustausch MUSS ein ephemer ECDH verwendet werden. Dabei MUSS die Kurve brainpoolP256r1 [RFC-6954] unterstützt werden. Es KÖNNEN die Kurven brainpoolP384r1, brainpoolP512r1 [RFC-6954] und ECP Gruppen 19, 20 und 21 [RFC-5903] unterstützt werden.
4. Als Verschlüsselungsverfahren im Rahmen von IKE MUSS AEAD_AES_128_GCM und AEAD_AES_256_GCM [RFC-5282] unterstützt werden (IANA.-Nr. 20) (Hinweis verpflichtend Unterstützung nach [RFC-5282#3.2]). Es MÜSSEN zudem AEAD_AES_128_GCM_12 und AEAD_AES_256_GCM_12 (IANA-Nr. 19) unterstützt werden. Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.2.1 Tabelle 2] unterstützt werden.
5. Als PRF für die Schlüsselerzeugung MUSS PRF_HMAC_SHA2_256 (IANA-Nr. 5) [RFC-4868] unterstützt werden. Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.2.2 Tabelle 3] unterstützt werden.
6. Als Authentisierungsverfahren MUSS ECDSA-256 als Basis von brainpoolP256r1 (IANA-Nr. 14) [RFC-7427] unterstützt werden. Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.2.5 Tabelle 6] unterstützt werden.
7. Für die Verschlüsselung der ESP-Pakete MUSS AES-GCM mit 16 Byte großem ICV (IANA-Nr. 20) und AES-GCM mit 12 Byte großem ICV (IANA-Nr. 19) [RFC-4106] jeweils mit 128 und 256 Bit Schlüssellänge unterstützt werden.
   Es KÖNNEN weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt werden.
8. Falls weitere Verfahren nach [TR-02021-3#3.3.1 Tabelle 7] unterstützt werden, so MUSS mindestens ein Verfahren zum Integritätsschutz der ESP-Pakete aus [TR-02021-3#3.3.2 Tabelle 8] unterstützt werden. (Hinweis: bei den verpflichtend zu unterstützenden AEAD-Verfahren aus Spiegelstrich 7 ist ein zusätzlicher Integritätsschutz by-design nicht notwendig.)

[<=]

Hinweis:

"strongSwan" unterstützt diese Algorithmen, vgl.
https://wiki.strongswan.org/projects/strongswan/wiki/IKEv2CipherSuites

A_17126 -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.
[<=]

4.6 ECDSA-Signaturen

4.6.1 ECDSA-Signaturen im XML-Format

A_17206 -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-Identifier „ 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_17360 -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.
[<=]

4.6.2 ECDSA-Signaturen im CMS-Format

A_17207 -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 Dokumentensignaturen die Verwendung des interoperablen Containerformats nach [ETSI-CAdES].

A_17359 -Signaturen binärer Daten (Dokumente) (ECC-Migration)
Alle Produkttypen, die (nicht-XML-)Signaturen von Dokumenten auf Basis eines ECC-Schlüssels erzeugen oder prüfen, MÜSSEN dabei die Vorgaben aus A_17207 umsetzen und die Signatur nach [ETSI-CAdES] (interoperables Container-Format) bei der Erzeugung kodieren bzw. bei der Prüfung auswerten.
[<=]

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

A_17208 -Signaturen von PDF/A-Dokumenten (ECC-Migration)
Alle Produkttypen, die (nicht-XML-)Signaturen auf Basis eines ECC-Schlüssels erzeugen oder prüfen, MÜSSEN dafür das Signaturverfahren ECDSA [BSI-TR-03111] auf Basis der Domainparameter brainpoolP256r1 nach [PAdES-3] und [PDF/A-2] verwenden. Als Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4] verwenden.
[<=]

4.7 ECIES

In der TI wird für die ECC-basierte Ver- und Entschlüsselung das "Elliptic Curve Integrated Encryption Scheme (ECIES)" verwendet. Es ist das einzige ECC-basierte, von den Chipkarten der TI unterstützte, Verschlüsselungsverfahren. Das ECIES ist ein hybrides Verfahren basierend auf [ABR-1999]. Es besteht aus einem asymmetrischen Teil (elliptic curve diffie hellman) und einen symmetrischen Teil (Verschlüsselungsverfahren und MAC-Verfahren). Weiterhin ist eine Schlüsselableitungsfunktion für das Verfahren notwendig. In [gemSpec_COS#6.8.2.3 Asymmetrische Entschlüsselung mittels ELC] wird definiert, welche Varianten dieser drei notwendigen Verfahren eine Chipkarte der TI unterstützt (ECDH [BSI-TR-03111#4.3.1 Key Agreement Algorithm], HKDF mittels SHA-256 und einem Zähler nach X9.63 [BSI-TR-03110-3#A.2.3.2], AES-256-CBC und CMAC).

Da im Normalfall immer für eine Identität, die Chipkarten-basiert ist, verschlüsselt wird, muss ein Sender genau diese Verfahren einsetzen. Ansonsten kann die Chipkarte das Chiffrat nicht entschlüsseln, auch wenn die Chipkarte den prinzipiell richtigen privaten Schlüssel in sich trägt.

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

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

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

Das erhaltene ECIES-Chiffrat muss dann als eine ASN.1-Struktur kodiert werden, die genau dem Aufbau entspricht, den man benötigt um eine Entschlüsselung mittels einer 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. $oidDO = ´06–L_{06}–oid\ ´.$
   iv. $keyDO = ´7F49–L_{7F49}–(\ 86 – L_{86} – PO_A\ )´.$
   v. $cipherDO = ´86–L_{86}–(\ 02 \ || \ C\ )´.$
   vi. $macDO = ´8E–L_{8E}–T´.$

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

Beispiel:

```
poGOBgkrJAMDAggBAQd/SUOGQQRouC6tM2TQQ+RP3pptgdAaDF8Te7IVCkUBe2H+PJSLK4W/
BXIX
kndiBwEfftd5wk4pjzCdC2j1q14/CIWcW89nhjEC7G47UAu2ZqmbIhxstkXV3UI2UUek/
qwBwtb2
6aUild+5kkTZXf5674OKHSdj6IFwjggvhYt9b/CTsA==

$ dumpasn1 a.bin
0 142: [6] {
3   9:   OBJECT IDENTIFIER brainpoolP256r1 (1 3 36 3 3 2 8 1 1 7)
14  67:   [APPLICATION 73] {
17  65:     [6]
     :       04 68 B8 2E AD 33 64 D0 43 E4 4F DE 9A 6D 81 D0
     :       1A 0C 5F 13 7B B2 15 0A 45 01 7B 61 FE 3C 94 8B
     :       2B 85 BF 05 72 17 92 77 62 07 01 1F 7E D7 79 C2
     :       4E 29 8F 30 9D 0B 68 F5 AB 5E 3F 08 85 9C 5B CF
     :       67
     :     }
84  49:   [6]
     :     02 EC 6E 3B 50 0B B6 66 A9 9B 22 1C 6C B6 45 D5
     :     DD 42 36 51 47 A4 FE AC 01 C2 D6 F6 E9 A5 22 95
     :     DF B9 92 44 D9 5D FE 7A EF 83 8A 1D 27 63 E8 81
     :     70
135   8:   [14] 2F 85 8B 7D 6F F0 93 B0
     :   }
```

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

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

```
ContentInfo
.contentType:1.2.840.113549.1.9.16.1.23 (id-ct-authEnvelopedData)
..AuthEnvelopedData
...version: 0
...recipientInfos
....RecipientInfo
.....ktri
......version: 0
......rid
.......issuerAndSerialNumber
........issuer
........[... weitere Kindelemente ...]
........SerialNumber: 123456789
.......keyEncryptionAlgorithm
........oid_ti_ecies_transport_encryption
.......encryptedKey: [ ... ASN.1-Struktur (Tupel (PO, C, T) in
kartenkompatibler ASN.1-Binärverpackung) ... ]
....RecipientInfo
.....ktriversion: 0
......rid
.......issuerAndSerialNumber
........issuer
........[... weitere Kindelemente ...]
```

```
........SerialNumber: 314159265
.......keyEncryptionAlgorithm
........OID TI-ECIES-TransportEncryption
.......encryptedKey: [ ... ASN.1-Struktur (Tupel (PO, C, T) in
kartenkompatibler ASN.1-Binärverpackung) ... ]
...authEncryptedContentInfo
....contentType:1.2.840.113549.1.7.1(id-data)
....contentEncryptionAlgorithm:
.....algorithm: 2.16.840.1.101.3.4.1.46 (id-aes256-gcm)
......parameters:
.......aes-nonce: [... IV ...]
.......aes-ICVlen: [... ICVLen ... ]
....encryptedContent: [...]
...mac: [...]
...unauthAttrs
....Attribute (id-recipientEmails)
.....attrType: komle-recipient-emails
```

und so weiter ...

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xenc:EncryptedData
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:dsig11="http://www.w3.org/2009/xmldsig11#"
  xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
  Type="http://www.w3.org/2001/04/xmlenc#">
 <xenc:EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#aes256-
gcm" />
<!-- Damit ist len(IV)=12 Byte und TagLen=16 Byte, vgl. [XMLEnc#5.2.4 AES-
GCM] -->
<ds:KeyInfo>
  <xenc:EncryptedKey>
  <!-- Hybridschlüssel für einen Empfänger, für weitere Empfänger gäbe es
jeweils ein weiteres EncryptedKey-Element -->
    <xenc:EncryptionMethod Algorithm="http://gematik.de/ecies/2019" />
    <!-- Die Version des speziellen ECIES -->
    <ds:KeyInfo>
        <ds:X509Data>
      <ds:X509Certificate>
       <!-- Base64-kodiertes X.509-Zertifikat (DER) des Empfängers -->
      </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>
<!-- Base64-kodierte ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler
ASN.1-Binärverpackung) -->
      </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedKey>
</ds:KeyInfo>
```

```
<xenc:CipherData>
  <xenc:CipherValue>
<!--
Base64-kodiertes symmetrisch mittels AES-256-GCM verschlüsseltes Dokument
(IV || ciphertext || Tag) mit len(IV)=12 Byte und len(Tag)=16 Byte,
vgl. [XMLEnc#5.2.4 AES-GCM]
-->
  </xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
```

A_17220 -Verschlüsselung binärer Daten (ECIES) (ECC-Migration)
Alle Produkttypen, die binäre Daten (also nicht XML-Daten) ECC-basiert verschlüsseln (im Folgenden als Nutzerdaten bezeichnet) und diese mittels CMS [RFC-5626] kodieren, MÜSSEN folgende Vorgaben umsetzen.

1. Zunächst MUSS ein 256-Bit AES-Schlüssel zufällig erzeugt werden (vgl. GS-A_4368), Dieser Schlüssel wird im Folgenden als Transportschlüssel bezeichnet (vgl. Erklärung in Abschnitt [gemSpec_Krypt#4.7- 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 Chiffrat wird im Folgenden als Transport-Chiffrat bezeichnet.
4. Das Transport-Chiffrat MUSS wie in [gemSpec_Krypt#4.7- 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 Chiffrats der Nutzerdaten MUSS wie in CMS üblich [RFC-5626] [RFC-5084] erfolgen (vgl. Darstellung in [gemSpec_Krypt#4.7- ECIES]).

[<=]

A_17221-01 -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-Chiffrat MUSS wie in [gemSpec_Krypt#4.7- 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#4.7- 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#4.7- ECIES] eingebracht werden.

[<=]

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

4.7.1 ECIES und authentisierte 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-Schlüsselmaterial). Es ist davon auszugehen, dass andere Anwendungen, die an mehrere Empfänger Nachrichten/Daten "gleichzeitig" versenden, ebenfalls zusätzliche Maßnahmen ergreifen müssen.

4.7.2 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-Karterterminal), 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 freigeschaltenen HBA des LE wieder entschlüsselt (genutzt) werden, bspw. zur Übertragung ins Primärsystem. Für die Verschlüsselung muss nach Ende 2025 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 Chiffrate auswerten können". Deshalb gibt es nachfolgend eine Spezialisierung von A_17220 für das Mobile Kartenterminal.

A_17575 -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.

3. Der Transportschlüssel MUSS mit den in [gemSpec_COS#6.8.1.4 ELC Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden (siehe Erklärung in [gemSpec_Krypt#4.7- ECIES] und [gemSpec_Krypt#Hinweis zu A_17575]).
4. Falls auf dem gesteckten HBA ein ECC-basiertes ENC-Zertifikat vorhanden ist, so MUSS ECIES für die Ver- und Entschlüsselung des Transportschlüssels verwendet werden, anstatt von RSA-OAEP. Falls noch kein ECIES-verschlüsselter Transportschlüssel im Mobilen Kartenterminal vorliegt, sondern ein RSA-OAEP-verschlüsselter Transportschlüssel, so MUSS dieser Transportschlüssel zusätzlich mittels ECIES und dem ECC-ENC-Schlüssel des HBAs des LE verschlüsselt werden.

[<=]

Hinweis zu A_17575:
In einem HBA steht die Schnittstelle [gemSpec_COS#6.8.1.4 ELC Verschlüsselung] für die Verschlüsselung des Transportschlüssels zur Verfügung. Dass der Transportschlüssel verschlüsselt werden muss, wird nur sehr selten als Anwendungsfall vorkommen. Die Entschlüsselung - notwendiger Weise mit und durch den HBA - jedoch häufig.

4.8 ECC-Migration eHealth-KT

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

A_17089-03 -eHealth-Kartenterminals: TLS-Verbindungen (ECC-Migration)
Ein eHealth-Kartenterminal MUSS prüfen, ob die in ihm gesteckte SMC-KT für die TLS-Verbindung zum Konnektor eine RSA-basierte Identität (AUT) und/oder eine ECDSA-basierte Identität besitzt (vgl. [gemSpec_gSMC-KT_ObjSys_G2.1], bspw. jeweils EFs mit ShortFileIdentifier 1 und 4 prüfen).
Falls eine RSA-basierte Identität dort vorhanden ist, so MUSS das eHealth-Kartenterminal folgende TLS-folgende Vorgaben erfüllen:

1. Als Cipher-Suite MÜSSENTLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 und TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 unterstützt werden.
2. Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE" im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5] unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in A_17089-* aufgeführt DÜRFEN NICHT verwendet werden.
3. Es KÖNNEN weitere Cipher-Suiten aus [TR-02102-2, Abschnitt 3.3.1 Tabelle 1] unterstützen.

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

1. Als Ciphersuiten MÜSSEN TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0,0x2B) und TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0,0x2C) unterstützt werden.
2. Beim ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (vgl. "ECDHE" im Namen der Cipher-Suites) MÜSSEN die Kurven P-256 und P-384 [FIPS-186-5] unterstützt werden. Es SOLLEN die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven als in GS-A_17089-* aufgeführt DÜRFEN NICHT verwendet 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_17090-01 -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). Für die Kodierung der ECC-Parameter in der Signatur MUSS die plain-Kodierung nach [TR-03111#5.2.1] verwendet werden.

[<=]

Hinweis: Das in A_17090-01 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_17183 -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 Komponten-PKI speichern.
[<=]

A_22458 -TLS-Algorithmus passend zum Pairing
Der Konnektor MUSS beim TLS-Verbindungsaufbau (TLS-Handshake) zu einem eHealth-Kartenterminal ausschließlich Ciphersuiten mit solchen Authentisierungsalgorithmen (entweder RSA oder ECDSA) anbieten, die zum Algorithmus des gespeicherten KT-Zertifikats passen, wenn zu diesem Kartenterminal bereits Pairinginformationen (Shared Secret in Kombination mit dem Zertifikat des Kartenterminals) gespeichert sind.**[<=]**

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

Cipher-Suiten anbieten, die auch zum für das betroffene Kartenterminal gespeicherte Zertifikat passen.

4.8.1 Interoperabilität zwischen eHealth-KT und Konnektor

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

A_22451 -ClientHello ohne akzeptable Cipher-Suite
Falls die ClientHello-Nachricht keine der gemäß [gemSpec_Krypt#A_17089-01] zu unterstützenden Cipher-Suiten enthält, dann MUSS das eHealth-Kartenterminal die ClientHello-Nachricht mit einem "handshake_failure" beantworten, siehe [RFC-5426#7.4.1.2].**[<=]**

Hinweise zum Thema Renegotiation:

Hinweis 1: Gemäß GS-A_5525 unterstützt der Konnektor "renegotiation". Daraus folgt, dass er diese Fähigkeit in der ClientHello- Nachricht entweder durch eine "renegotiation_info" extension anzeigt, oder durch ein TLS_EMPTY_RENEGOTIATION_INFO_SCSV (siehe [RFC-5746#3.4, §1, erster Punkt]).

Hinweis 2: Gemäß [RFC-5246#7.4.1.3 letzter §] ist es einem Server nur dann erlaubt in der ServerHello-Nachricht eine "renegotiation_info" extension zu schicken, wenn die ClientHello-Nachricht Informationen zu "renegotiation" enthält. Gemäß dem vorstehenden Hinweis ist das beim TLS-Handshake zwischen Konnektor und eHealth-KT stets der Fall.

Hinweis 3: Die gematik legt nicht fest, wie sich ein eHealth-KT zu verhalten hat, wenn die ClientHello-Nachricht weder eine "renegotiation_info" extension, noch ein TLS_EMPTY_RENEGOTIATION_INFO_SCSV enthält.

Hinweis 4: Es gilt für das eHealth-KT GS-A_5524-*, womit sichergestellt ist, dass ein eHealth-KT keine unsichere TLS-Renegotiation durchführt.

A_22453 -ServerHello, Cipher-Suite
Von den gemäß [gemSpec_Krypt#A_17089-01] zu unterstützenden Cipher-Suiten wählt das eHealth-Kartenterminal eine aus, vergleiche [RFC-5426#7.4.1.3]. Falls das eHealth-Kartenterminal eine Cipher-Suite aus [gemSpec_Krypt#A_17089-01] in der Form

1. TLS_DHE_RSA... (also RSA) auswählt, dann MUSS es eine RSA-basierte Identität zur Authentisierung im Kontext des TLS-Protokols verwenden.
2. TLS_ECDHE_ECDSA... (also ECDSA) auswählt, dann MUSS es eine ECDSA-basierte Identität zur Authentisierung im Kontext des TLS-Protokols verwenden.

[<=]

A_22454 -CertificateRequest
Das eHealth-Kartenterminal MUSS eine CertificateRequest-Nachricht schicken, für die folgendes gilt:

1. Die Struktur CertificateRequest gemäß [RFC-5246#7.4.4] MUSS im Abschnitt certificate_types genau ein Element vom Typ ClientCertificateType enthalten.
2. Falls das eHealth-Kartenterminal in der ServerHello-Nachricht eine Cipher-Suite aus [gemSpec_Krypt#A_17089-01] in der Form
   a. TLS_DHE_RSA... (also RSA) anzeigt, dann MUSS als ClientCertificateType "rsa_sign" gewählt werden, siehe [RFC-5246#7.4.4].

b. TLS_ECDHE_ECDSA... (also ECDSA) anzeigt, dann MUSS als ClientCertificateType "ecdsa_sign" gewählt werden, siehe [RFC-8422#5.5].

[<=]

A_22455 -ClientCertificate
Bezüglich der ClientCertificate-Nachricht (siehe [RFC-5246#7.4.6]) gilt für das eHealth-Kartenterminal folgendes Verhalten:

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

[<=]

4.9 ECC-Migration Konnektor

A_17094-02 -TLS-Verbindungen Konnektor (ECC-Migration)
Der Konnektor MUSS zusätzlich zu den RSA-basierten TLS-Ciphersuiten (vgl. GS-A_4385 und GS-A_5345-01) die TLS-Ciphersuiten

1. TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 und
2. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

unterstützen. Dabei MÜSSEN bei dem ephemeren Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch die Kurven P-256 und P-384 [FIPS-186-5] und die Kurven brainpoolP256r1 und brainpoolP384r1 (vgl. [RFC-5639] und [RFC-7027]) unterstützt werden. Andere Kurven SOLLEN NICHT verwendet werden.
Falls der Konnektor in der Rolle TLS-Client agiert, so MUSS er die eben genannten Ciphersuiten gegenüber RSA-basierten Ciphersuiten (vgl. GS-A_4384-*) bevorzugen (in der Liste "cipher_suites" beim ClientHello vorne an stellen, vgl. [RFC-5256#7.4.1.2 Client Hello]).
[<=]

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

A_17209 -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.**[<=]**

A_23511 -Konnektor, IOP, Kodierung ECC-Schlüssel, Primärsystem-Verbindungssicherung
Der Konnektor MUSS sicherstellen, dass bei der Umsetzung von TIP1-4517-* ECC-Schlüssel stets in der "named-curve"-Kodierung exportiert werden (d. h., eben gerade nicht die "explizite" Kurvenparameter-Kodierung verwenden).
D. h., sowohl

1. für ein Primärsystem von einem Konnektor erzeugte und exportierte private ECC-Schlüssel MÜSSEN mittels named-curve-Darstellung kodiert sein (siehe Hinweise) und

2. bei den vom Konnektor erzeugten TLS-Authentisierungszertifikaten MÜSSEN die im Zertifikat bestätigten öffentlichen ECC-Schlüssel in der named-curve-Darstellung kodiert sein (siehe Hinweise).

[<=]

Hinweise:

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

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

```
-----BEGINECPARAMETERS-----
BgkrJAMDAggBAQc=
-----ENDECPARAMETERS-----
-----BEGINECPRIVATEKEY-----
MHgCAQEEIC5PFJ/4fOdPLZ1BebMqpHN8ydae9kSPGS5qICS/WGpnoAsGCSskAwMC
CAEBB6FEA0IABCpcFsaT3f+O9Cg676mRWR4WazrrbFMcMZ5dwJhrhVsjSzA3z85q
u8eCSvAkE3jt2+nOTNh1s3ExkZVSOJD1aLc=
-----ENDECPRIVATEKEY-----
```

(über

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

erzeugt)

Die DER-Kodierung sieht dann wie folgt aus

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

0 warnings, 0 errors.
```

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

4.10 Verschiedene Produkttypen und ECC-Migration (informativ)

Dem VPN-Zugangsdienst sind die IPsec-AnforderungenA_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 (Registierungsserver) prüfen können, dafür gilt für diesen A_17206.

Für die Produkttypen, die bei der Anwendung VSDM verwendet werden, ist die ECC-RSA-TSL-AuswertungA_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]).

5 VAU-Protokoll für E-Rezept

5.1 Übersicht (informativ)

Ähnlich wie bei der Anwendung ePA besitzt der Fachdienst E-Rezept mindestens zwei HTTPS-Schnittstellen. Die dabei vom Nutzer (Client) aufgebaute TLS-Verbindung endet an einer dieser HTTPS-Schnittstellen. Die E-Rezept-VAU liegt hinter den Webschnittstellen. Um die Verbindungsstrecke zwischen HTTPS-Schnittstellen und E-Rezept-VAU zu schützen, verschlüsselt und kodiert der Client seine HTTP-Requests (als innere Requests bezeichnet) auf die im Abschnitt 5.2 definierte Weise. Diese so erzeugten HTTP-Requests (äußere Requests) sendet der Client per HTTPS an eine der Webschnittstellen des E-Rezept-Dienstes. Dabei ist für die Schnittstelle ein regelmäßig wechselndes Nutzerpseudonym im äußeren HTTP-Request sichtbar. Dieses unterstützt den Fachdienst E-Rezept bei der Lastverteilung und beim DoS-Schutz auf Applikationsebene. Von der Webschnittstelle erhält die E-Rezept-VAU die verschlüsselten Requests, entschlüsselt und verarbeitet diese. Als Antwort erzeugt die E-Rezept-VAU einen für den Client verschlüsselte HTTP-Response. Den Schlüssel dafür hat der Client für die VAU ephemer erzeugt und dieser Schlüssel ist Teil des verschlüsselten Requests. Die Webschnittstelle empfängt das Chiffrat von der VAU und gibt dieses als Antwort auf den HTTP-Request an den Client zurück.

Ablauf:

1. Der Client erfragt initial das X.509-Zertifikat der VAU (A_20160-*). Je nachdem, ob die X.509-Root der TI oder die TSL als Prüfbasis verwendet wird, erfolgt die Abfrage des VAU-X.509-Zertifikats spezifisch (entweder A_21216 oder A_24465 ff.). 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 Chiffrat ist ein Authentisierungstoken des Nutzers enthalten, eine zufällig gewählte Request-ID, ein vom Client ephemer 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 Chiffrat an eine HTTPS-Schnittstelle des Fachdienstes E-Rezept. Dabei gibt der Client ggf. das schon aus einer vorherigen Response des Fachdienstes E-Rezept mitgeteilte Nutzerpseudonym (A_20161-*) mit.
5. Die Webschnittstelle nimmt den Request entgegen und trifft eine Routing-Entscheidung oder eine Priorisierungsentscheidung im Lastfall. Anschließend leitet sie den Request an die VAU weiter (A_20162).
6. Die VAU entschlüsselt den Request und verarbeitet ihn. Die Antwort wird inkl. der Request-ID mit dem vom Nutzer erzeugten Antwortschlüssel verschlüsselt und an die Webschnittstelle gesendet.

7. Die Webschnittstelle antwortet gegenüber dem Client mit diesem Chiffrat auf den HTTP-Request.
8. Der Client entschlüsselt das Chiffrat und prüft die Request-ID in der entschlüsselten Antwort.

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

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

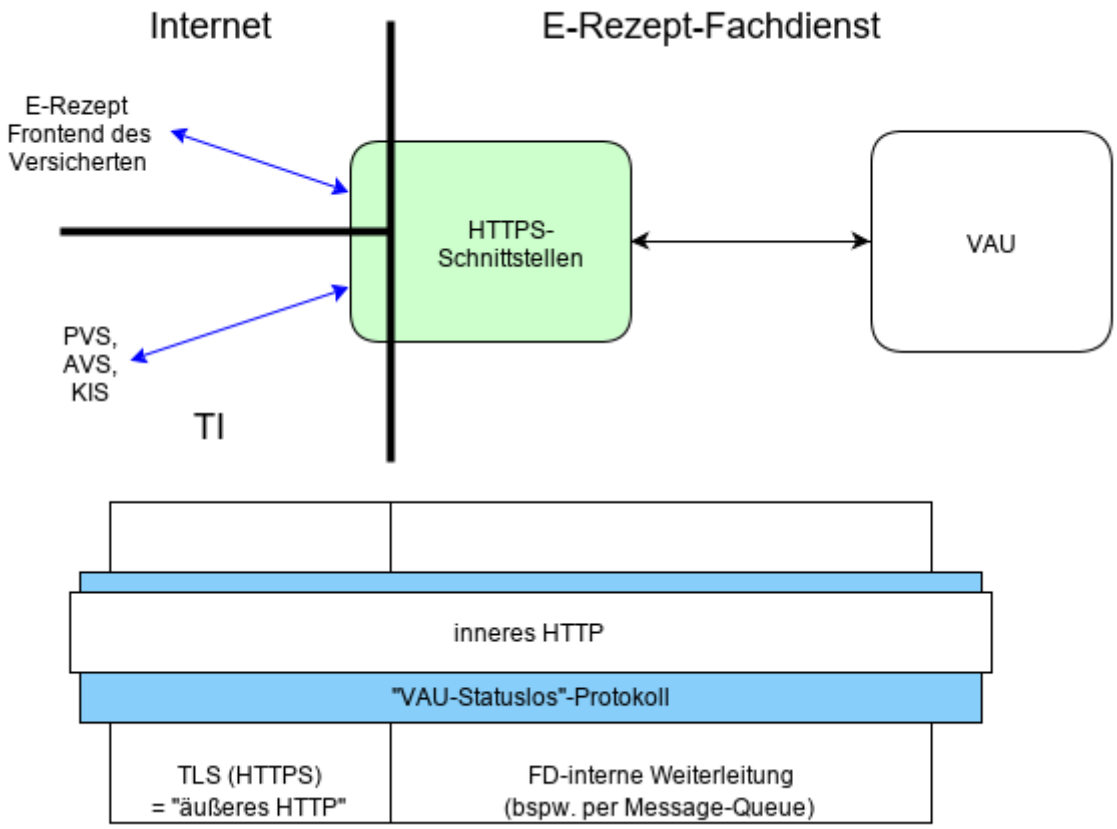

Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU

Die gematik stellt eine Beispielimplementierung bereit.

5.2 Definition

5.2.1 E-Rezept-VAU-Identität

A_20160-03 -E-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.

[<=]

Hinweis: Die Validierung und Verifizierung des VAUCertificate für Primärsysteme erfolgt mittels Konnektoroperation `VerifyCertificate`. DasE-Rezept-FdV validiert das Zertifikat mit den vom E-Rezept-Fachdienst bereitgestellten Endpunkten GET /PKICertificate und GET /OCSPResonse (siehe 6.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität).

A_20967 -E-Rezept-VAU, Erstellung und Pflege der Schlüssel im Mehr-Augen-Prinzip
Der Fachdienst E-Rezept MUSS folgende Punkte sicherstellen.

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

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

Hinweis: Der Betreiber des FD besitzt für bestimmte Rollenauthentisierungen für die HSM Authentisierungschipkarten inkl. PIN, die gematik ebenfalls. Nur bei gleichzeitiger Authentisierung beider Beteiligter erlauben die HSM bestimmte Aktionen/Funktionalitäten.

5.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität

Bevor ein E-Rezept-Client die E-Rezept-VAU-Identität für die Verschlüsselung seiner Request (vgl.5.2.3- E-Rezept-VAU-Request und -Response) verwendet, muss der Client deren X.509-Zertifikat prüfen ([gemSpec_eRp_FdV#A_19739], [gemILF_PS_eRP#A_20769]).

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

Da ein E-Rezept-FdV nur wenige TI-Zertifikate prüfen muss, wird im FdV die im Folgenden beschriebene technisch deutlich einfachere Zertifikatsprüfung auf Basis der TI-X.509-Root

verwendet, und nicht die Prüfung über die TSL wie im Client eines PVS oder AVS [gemILF_PS_eRP#A_20764]. Beide Prüfwege sind sicherheitstechnisch äquivalent.

In den folgenden Anforderungen wird beschrieben, wie ein E-Rezept-Clientdie Zertifikatsprüfung durchführen muss.

A_21222-01 -E-Rezept-Client, allgemein Zertifikatsprüfung
Ein E-Rezept-Client MUSS bevor er TI-X.509-Zertifikate in fachlichen Abläufen (bspw. VAU-Kanal) verwendet, diese Zertifikate prüfen.**[<=]**

A_21216-02 -E-Rezept-Client, Prüfung VAU-Zertifikat auf TSL-Basis
Ein E-Rezept-Client, der nicht das E-Rezept-FdV ist, MUSS das VAU-Zertifikat vom E-Rezept-FD beziehen und mittels TUC_PKI_018 (OCSP-Graceperiod=12h, PolicyList={oid_erp-vau}) prüfen.**[<=]**

Hinweis: E-Rezept-Clients, die Konnektoren nutzen, verwenden zur Prüfung des VAU-Zertifikats die Operation `VerifyCertificate`. Die in der Response enthaltene Rolle muss der oid`1.2.276.0.76.4.258` (oid_erp-vau) entsprechen. Damit wird die Anforderung A_21216-* erfüllt.

Folgendes kann umgesetzt werden:

(1) Beziehen des VAU-Zertifikat von /VAUCertificate

(2) Lokales Speichern der aktuellen Zeit mit dem VAU-Zertifikat als Tupel

(3) Prüfen des VAU-Zertifikates mittels der Konnektor-Operation VerifyCertificate

(4) Abbrechen falls INVALID

(5) if (get_current_time() < gespeicherte Zeit + 12h) { VAU-Zertifikat wird als gültig angesehen, Nutzen des VAU-Zertifikat }

if (get_current_time() >= gespeicherte Zeit + 12h) { VAU-Zertifikat neu beziehen, siehe (1)}

(6) if (VerificationStatus.RoleList.Role == 1.2.276.0.76.4.258) {VAU-Zertifikat entspricht der Identität oid_erp-vau}

Hinweis zum Fehlerhandling: Nur wenn der äußere Response der E-Rezept-Fachdienstes den Response-Code 200 liefert, enthält der payload eine mittels VAU-Protokoll verschlüsselte Response. Liefert der äußere Response eine Code >= 400, ist im VAU-Protokoll ein Fehler aufgetreten. Das PS muss nicht versuchen, den payload zu entschlüsseln.

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

Tabelle 16: Tab_KRYPT_ERP Definition Datenstruktur PKI-Zertifikatsliste

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

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

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

A_24465 -E-Rezept-Fachdienst - Bereitstellung von CA-Zertifikaten
Der E-Rezept-Fachdienst MUSS die CA-Zertifikate (`ca_certs`) über den Endpunkt `GET /PKICertificates?currentRoot` dem E-Rezept-FdV bereitstellen. Der E-Rezept-Fachdienst MUSS, um die CA-Zertifikate zu ermitteln, die TSPServices der TSL.xml nach folgenden Kriterien filtern:

- ServiceInformation.ServiceTypeIdentifier == "http://uri.etsi.org/TrstSvc/Svctype/CA/PKC"
- ServiceInformation.ServiceInformationExtensions.Extension.ExtensionOID == "1.2.276.0.76.4.202" OR "1.2.276.0.76.4.203"

Der E-Rezept-Fachdienst MUSS alle Zertifikate aus`ServiceDigitalIdentity.DigitalId.X509Certificate` der in der Ergebnismenge vorhandenen TSPServices extrahieren und unter "`ca_certs`" base64-kodiert dem Array hinzufügen und bereitstellen.**[<=]**

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

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

A_24466 -E-Rezept-Fachdienst - Bereitstellung von Cross Zertifikaten
Der E-Rezept-Fachdienst MUSS die Cross Zertifikate (`add_roots`) über den Endpunkt `GET /PKICertificates?currentRoot` dem E-Rezept-FdV bereitstellen. Der E-Rezept-Fachdienst MUSS, um die benötigten Cross Zertifikate zu ermitteln, den SubjectCN aus dem URL-Parameter "currentRoot" extrahieren und dann wie folgt vorgehen:
Der E-Rezept-Fachdienst MUSS, falls es in der Ergebnismenge der `ca_certs` Zertifikate gibt, die nicht per Signaturprüfung auf den angegebenen Root-Schlüssel vom E-Rezept-FdV rückführbar sind, nacheinander die chronologisch auf einander folgenden und vorausgehenden base64-kodierten Root-Cross-Zertifikate, bspw.:

- GEM.RCA4->GEM.RCA5, ... GEM.RCA(x)->GEM.RCA(x+1)
- GEM.RCA4 -> GEM.RCA3, ... GEM.RCA(x)->GEM.RCA(x-1)

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

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

Tabelle 17: API von GET /OCSPResponse

Beispiel URL	GET /OCSPResponse?issuer-cn=GEM.KOMP-CA3&serial-

	nr=123124
URL-Parameter "serial-nr"	Seriennummer (Serial Number) des zu prüfenden Zertifikats
URL-Parameter "issuer-cn"	Name des Ausstellenden Zertifikats (Issuer Common Name)
Response	OCSP Response nach RFC-6960

A_24467 -E-Rezept-Fachdienst - Bereitstellung von OCSP Responses
Der E-Rezept-Fachdienst MUSS über den Endpunkt `GET /OCSPResonse?issuer-cn&serial-nr` einen OCSP Request nach [RFC-6960] mit dem Issuer Zertifikat aus der TSL erstellen und an den OCSP Responder zustellen. Die OCSP Response wird an den anfragenden Client zurückgeben.
Falls eine entsprechende OCSP Response bereits im Cache vorhanden ist, kann diese zurückgegeben werden, wenn diese nach geltenden Caching-Regeln noch aktuell ist.**[<=]**

A_24468 -E-Rezept-Fachdienst - Caching von OCSP Responses
Der E-Rezept-Fachdienst MUSS angefragte OCSP Responses von`GET /OCSPResonse?issuer-cn&serial-nr` bis zu 4 Stunden cachen und bei einer entsprechend passenden OCSP-Anfrage, anstatt neu den OCSP-Responder anzufragen, die im Cache befindliche OCSP-Antwort ausliefern.**[<=]**

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

A_24469 -E-Rezept-FdV - Installierter Vertrauensanker
Das E-Rezept-FdV MUSS eine noch mindestens fünf Jahre gültige ECC ROOT-CA aus https://download.tsl.ti-dienste.de/ECC/ROOT-CA/ als Vertrauensanker im Programm-Code bzw. mit dem Programm-Code fest assoziiert enthalten und als Basis für die Prüfung von TI-Zertifikat verwenden.**[<=]**

A_24470 -E-Rezept-FdV - Truststore mit TI-Zertifikaten
Das E-Rezept-FdV MUSS einen TI-Zertifikate-Truststore enthalten und pflegen, der folgende fünf Kategorien beinhaltet:

- (A) Root-Schlüssel und Cross-Zertifikate,
- (B) CA-Zertifikate,
- (C) E-Rezept-VAU-Zertifikat,
- (D) IDP-Zertifikat(e).
- (E) E-Rezept-Fachdienst Signaturzertifikat

Initial enthält dieser Truststore nur das vorinstallierte Root-Zertifikat.
[<=]

A_25058 -E-Rezept-FdV- Befüllen des Truststores
Das E-Rezept-FdV MUSS für den Fall, dass keine Zertifikate der Kategorie (C) und (D) vorhanden sindden Truststore mittels des Algorithmus `Tab_KRYPT_ERP_FdV_Truststore_aktualisieren` initial befüllen.**[<=]**

A_25059 -E-Rezept-FdV - OCSP Responses für TI-Zertifikate
Das E-Rezept-FdV MUSS bei der Prüfung von TI-Zertifikaten OCSP Responses für die Zertifikate aus (C), (D) und (E) holen, wenn kein weniger als 12h alter OCSP Response vorliegt.**[<=]**

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

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

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

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

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

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

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

Tabelle 18: Tab_KRYPT_ERP_FdV_Truststore_aktualisieren

(1) Alle Zertifikate im Truststore müssen gelöscht werden mit Ausnahme des vorinstallierten Root-Zertifikats. Der Truststore besitzt, wie in (A_24470 - E-Rezept-FdV - Truststore mit TI-Zertifikaten) definiert, Prüfschlüssel/Zertifikate in folgenden Kategorien (A) Root-Schlüssel und Cross-Zertifikate, (B) CA-Zertifikate, (C) E-Rezept-VAU-Zertifikat, (D) IDP-Zertifikat(e), (E) E-Rezept-Fachdienst Signaturzertifikat. (2) Falls die Zertifikatsliste im Array "add_roots" aus`GET /PKICertificates`nicht leer ist, so wird das erste Cross-Zertifikat im Array auf Basis des vorinstallierten Root Zertifikats per Signaturprüfung geprüft. Ebenfalls wird geprüft, ob der bestätigte CommonName dem Muster "GEM.RCA" + Zahl entspricht. Falls beide Prüfungen mit positiven Prüfergebnis erfolgen konnten, so wird das Zertifikat in Kategorie (A) des Truststores aufgenommen. Weitere Zertifikate im Array werden analog mit dem jeweils vorherigen Cross-Zertifikat per Signaturprüfung und CommonName-Prüfung geprüft und bei positiven Prüfergebnissen in Kategorie (A) des Truststores aufgenommen.

(3)
Für jedes Zertifikat im Array "ca_certs" aus`GET /PKICertificates`wird überprüft, ob es zeitlich gültig ist, ob es per Signaturprüfung auf einen Root-Schlüssel aus (A) rückführbar ist und ob der bestätigte CommonName dem Muster "GEM.KOMP-CA" + Zahl entspricht. Wenn alle drei Prüfungen ein positives Prüfergebnis liefern, so wird das Zertifikat in Kategorie (B) des Truststores aufgenommen.
(4)
Für das VAU-Zertifikat aus`GET /VAUCertificate`und die IDP-Zertifikate vom IDP-Dienst wird überprüft ob es zeitlich gültig ist und ob es per Signaturprüfung auf einen CA-Schlüssel aus (B) rückführbar ist. Wenn nicht beide Prüfungen ein positives Prüfergebnis liefern, so wird das Zertifikat verworfen.
Anderen falls wird geprüft, ob das Zertifikat eine OID mit dem Wert oid_erp-vau [gemSpec_OID] enthält. Dann wird es im Truststore bei Kategorie (C) eingefügt. Falls das Zertifikat eine OID mit dem Wert oid_idpd [gemSpec_OID] enthält. Dann wird es im Truststore bei Kategorie (D) eingefügt.

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

5.2.3 E-Rezept-VAU-Request und -Response

A_20161-01 -E-Rezept-Client, Request-Erstellung
Ein E-Rezept-Client MUSS, falls ihm noch kein gültiges E-Rezept-VAU-Zertifikat vorliegt, ein solches nach den fachlichen Vorgaben von A_20160-* beziehen (`/VAUCertificate`).
Ein E-Rezept-Client MUSS sicherstellen, dass gültige Sperrinformation (OCSP-Response mit Sperrstatus "good") für das Zertifikat vorliegen, 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 erzeugen.
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 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] durchgeführen. 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 Fomulierung 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-Chiffrat> || <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 19: Tab_KRYPT_ERP Kodierung des Chiffrats aus A_20161-*

Offset	Länge in Bytes	Erläuterung
0	1	Versionsfeld
1	32	X-Koordinate öffentlicher ephemer Sender-ECDH-Schlüssel
33	32	Y-Koordinate öffentlicher ephemer Sender-ECDH-Schlüssel
65	12	IV zufällig pro Nachricht zu erzeugen (A_20161-* Punkt e)
77	gleich Länge des Plaintextes (= LP)	"eigentliche" AES-GCM-Chiffrat
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=5bbba34d47502bd588ed680dfa2309ca375eb7a35ddbbd67cc7f8b6b687a1c1d

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 Chiffrat:

86c2b491c7a8309e750b und folgenden 16 Byte großen Authentication Tag4e6e307219863938c204dfe85502ee0a

Das Gesamt-Chiffrat 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_20162 -E-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.

[<=]

Beispiele für mögliche Pfade die nach A_20162 von einem Client erzeugt werden könnten:

/VAU/0
/VAU/0/Task/create
/VAU/270810c79748768a9b0aefbf52c8d72be7ad5e0d2d328d9bb70dbf58623fc7ae

A_20163 -E-Rezept-VAU, Nutzeranfrage, Ent- und Verschlüsselung
Die E-Rezept-VAU MUSS das Folgende sicherstellen und im Falle eines Fehlschlagens die Abarbeitung des Requests mit einer entsprechenden Fehlermeldung an die sie aufrufende Webschnittstelle abbrechen.

1. Die E-Rezept-VAU MUSS einen verschlüsselten Nutzer-Request von der Webschnittstelle entgegennehmen.
2. Die E-Rezept-VAU MUSS einen verschlüsselten Nutzer-Request nach den kryptographischen Vorgaben aus A_20161-* und mit dem privaten Schlüssel aus A_20160-* versuchen zu entschlüsseln.
3. Die E-Rezept-VAU MUSS den erhaltenden Klartext p auf den Strukturaufbau aus A_20160-* prüfen.
4. Die E-Rezept-VAU MUSS das JWT-Authentisierungstoken auf Gültigkeit prüfen.

5. Die E-Rezept-VAU MUSS den in p kodieren HTTP-Request abarbeiten.
6. Die E-Rezept-VAU MUSS einen 128-Bit-AES-CMAC-Schlüssel zufällig erzeugen und mindestens alle 10 Tage wechseln.
7. Die E-Rezept-VAU MUSS aus dem "sub"-Feld-Wert mittels des CMAC-Schlüssels den 128 Bit langen CMAC-Wert berechnen und hexadezimal kodieren (32 Byte lang). Dies sei das Prenutzerpseudonym (PNP).
8. Die Antwort "a" auf den HTTP-Request aus p MUSS wie folgt kodiert werden: a="1" + " " + Request-ID-aus-p + " " + Response-Header-und-Body.
9. Die E-Rezept-VAU MUSS a mittels des 128-Bit langen AES-Schlüssels aus p und AES/GCM (96 Bit zufällig erzeugter IV, 128 Authentication Tag) verschlüsseln und erhält c'.
10. Die E-Rezept-VAU MUSS c' und das PNP an die Webschnittstelle als Antwort übergeben.

[<=]

A_20174 -E-Rezept-Client, Response-Auswertung
Ein E-Rezept-Client MUSS bei der Response-Auswertung (vgl. vorgehenden Client-Request aus A_20161-*) folgende Schritte durchführen. Dabei MUSS der Client bei Fehlschlagens im Folgenden aufgeführten Prüfungen die Analyse der Response abbrechen, und er MUSS die Request-ID und den AES-Antwortschlüssel sicher löschen.

1. Er MUSS prüfen, ob der Content-Type der Response 'application/octet-stream' ist.
2. Wenn im Response-Header die Variable "Userpseudonym" vorhanden ist, so MUSS er den Wert von "Userpseudonym" als NP für den nächsten Request an die VAU verwenden. (Der Client MUSS einen ggf. vorhandenen alten Wert des NP im Client überschrieben.)
3. Er MUSS das Antwort-Chiffrat mit den Vorgaben aus A_20163 (9) und dem AES-Antwort entschlüsseln und prüfen ob die Entschlüsselung erfolgreich möglich war.
4. Er MUSS prüfen, ob die Struktur des erhaltenen Klartextes p der Struktur aus A_20163 (8) entspricht.
5. Er MUSS prüfen, ob die Request-ID in p der Request-ID aus dem Client-Request entspricht (Gleichheit prüfen).
6. Er MUSS das dritte Feld-Element in p ("Response-Header-und-Body") als HTTP-Antwort der E-Rezept-VAU in fachlich weiter verarbeiten.

[<=]

A_20175 -E-Rezept-Client, Speicherung Nutzerpseudonym
Ein E-Rezept-Client MUSS das im Request verwendete Nutzerpseudonym (NP) in Software speichern (kein HSM/TPM/SE) und das NP ausschließlich für seinen Einsatzzweck der E-Rezept-VAU-Kommunikation verwenden. Insbesondere MUSS der Client die Vertraulichkeit des NP wahren (bspw. nicht unnötig in Protokolleinträgen und Fehlermeldungen aufführen).**[<=]**

Der Fachdienst E-Rezept besitzt eine REST-Schnittstelle, d. h. Fehler werden mittels HTTP-Status/Fehler-Codes signalisiert. Die in der folgenden Tabelle (Tab_KRYPT_VAUERR) aufgeführten Fehler kann ein E-Rezept-Client in Bezug auf die in diesem Abschnitt definierte kryptographische Sicherung zwischen Client und VAU treffen.

Tabelle 20: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept)

Fehlerfall	HTTP-Response-Code der E-Rezept
JWT-Client-Token ungültig	403 Forbidden
Entschlüsselung des Chiffrats des äußeren Requests schlug fehl	400 Bad Request
Strukturaufbau des Klartextes aus A_20160-* ist falsch	400 Bad Request

5.2.4 Zufallsquelle für Clients

Zur Auffrischung des Entropie-Pools des Clients kann der Client von verschiedenen Diensten der TI Zufall ausreichender Güte (vgl. GS-A_4367) beziehen. Der FD-E-Rezept ist dafür eine Quelle von mehreren. Ein Client muss verschiedene unabhängige Quellen abfragen und die Zufallsdaten kryptographisch geeignet (bspw. über den Fortuna-PRNG-Algorithmus) zusammenführen.

A_21215 -E-Rezept-FD, Random-Operation
Der Fachdienst E-Rezept MUSS an seiner Webschnittstelle (HTTPS) unter der URL /Random (GET-Methode) 128 Byte Zufallsdaten hexadezimalkodiert (Lower-Case -- [0-9a-f]) einen Client zur Verfügung stellen. Bei jedem Request MÜSSEN die Zufallsdaten neu erzeugt werden. Die Response MUSS den Content-Type application/json besitzen.
Teil einer Beispiel-Response:

```
Content-length: 258
Content-type: application/json
Date: Tue,01 Dec 2020 12:46:18 GMT
Server: nginx/1.14.0 (Ubuntu)

"a5dc9d13ee2e76ddd9b75e9c28421fc4b5a9a131751a3dad1203f8d1b149366ef938163d43
718f31fe5464e05f236ba62588cea48ff8cdb9f77abe52a03a389f8a2573127c70629742387
14e457399cfc9fcd7eeb656c2cfd3bf50fb1d74b4cd5c73607283533f423760c2e38a3fd646
602ef244d4dbdb332c8f696b5e07ef51"
```
[<=]

Um die Anforderung umzusetzen ist es im Normalfall ausreichend /dev/urandom als Zufallsquelle auf einen Linux-Server zu verwenden.

6 VAU-Protokoll für ePA für alle

Der grundsätzliche architektonische Aufbau des ePA-Aktensystems als Fachdienst bei ePA für alle entspricht dem bei ePA 1.x, 2.x und beim E-Rezept: Das Aktensystem besteht aus mehreren Webschnittstellen, bei denen über das HTTPS-Protokoll Requests eintreffen. Diese Requests werden im Aktensystem an verschiedene VAU-Instanzen zur Verarbeitung weitergeleitet. Die Antwort von einer VAU-Instanz erzeugt und an die Webschnittstelle übergeben. Von dieser wird sie per HTTPS an die Clients innerhalb einer HTTP-Response übermittelt. Zwischen ePA-Client und VAU gibt es also keine durchgehende TLS-Verbindung -- eine TLS-Verbindung des Clients endet an einer Webschnittstelle des Aktensystems.
Das VAU-Protokoll erzeugt einen durchgehenden (also ununterbrochenen) Kanal zwischen ePA-Client und VAU.

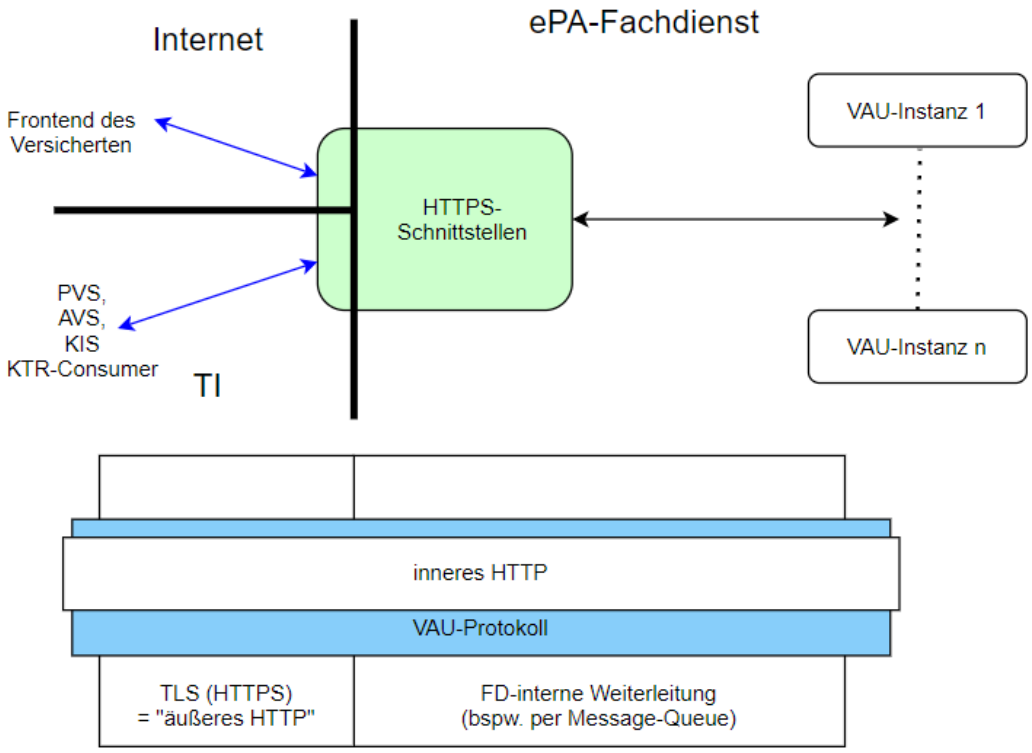

Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-VAU-Instanz

Bei ePA 1.x (und ePA 2.x), beim E-Rezept und bei ePA für alle werden jeweils unterschiedliche Authentisierungsarten für die Client-Authentisierung verwendet. Die Client-Authentisierung durchläuft quasi eine Evolution, der das VAU-Protokoll durch Anpassung folgen muss, damit die Sicherheitsziele weiterhin erreicht werden. Bei ePA 1.x und 2.x war es möglich, dass Nutzer des Aktensystems direkt über private AUT-Schlüssel Verbindungsparameter signieren konnten. Beim E-Rezept wird aktuell eine Variante des OIDC-Protokolls verwendet, bei der ein E-Rezept-Client direkt das Auth-Token lokal

vorliegen hat. Solche Auth-Token verwendet er bei der Authentisierung im VAU-Protokoll beim E-Rezept-FD. Bei ePA für alle gibt es einen Authentication-Flow nach [RFC-7636] (OAuth 2.0 PKCE). Dort hat ein Client keinen Zugriff mehr auf das Auth-Token, d. h. das VAU-Protokoll aus dem E-Rezept in der aktuellen Ausbaustufe kann nicht verwendet werden. Deshalb wird für ePA für alle eine neue Version des VAU-Protokolls eingeführt. In einer späteren Ausbaustufe wechselt auch beim E-Rezept die Authentisierung auf diese OIDC-Variante -- ab dann wird bei ePA für alle und bei E-Rezept dasselbe VAU-Protokoll verwendet.

Wenn über eine HTTPS-Schnittstelle ein Request eines ePA-Clients eintrifft, steht es einem Aktensystem frei, über welches Protokoll es diesen Request an die VAU-Instanzen verteilt (bspw. per Message-Queue), d. h. die Aktensysteminterne Verbindungsstrecke zwischen HTTPS-Schnittstellen und VAU-Instanzen ist frei durch den Aktensystemhersteller wählbar.

A_24654 -ePA: HTTP-Version
Ein ePA-Aktensystem MUSS an seinen HTTPS-Schnittstellen, i. S. v. Schnittstellen, die ein ePA-Client anspricht, mindestens HTTP Version 1.1 unterstützen.**[<=]**

Hinweis: Im Kontext ePA bietet HTTP/2 nur geringe Vorteile gegenüber HTTP/1.1. HTTP/2 besitzt jedoch eine deutlich höhere Komplexität. Ein Hersteller ist mit A_24654-* nicht verpflichtet, eine HTTP-Version größer als 1.1 zu implementieren. A_24654 ist nicht dahingehend zu verstehen, dass von einem Aktensystem perfekte Konformität zu HTTP/1.1 umzusetzen ist. Im Kontext der ePA-Funktionalität wird nur der Umfang von HTTP verwendet, den man normaler Weise bei Webservices verwendet.

Wenn im folgenden innerhalb dieses Kapitels6 vom VAU-Protokoll gesprochen wird, ist stets die Variante für ePA für alle gemeint.

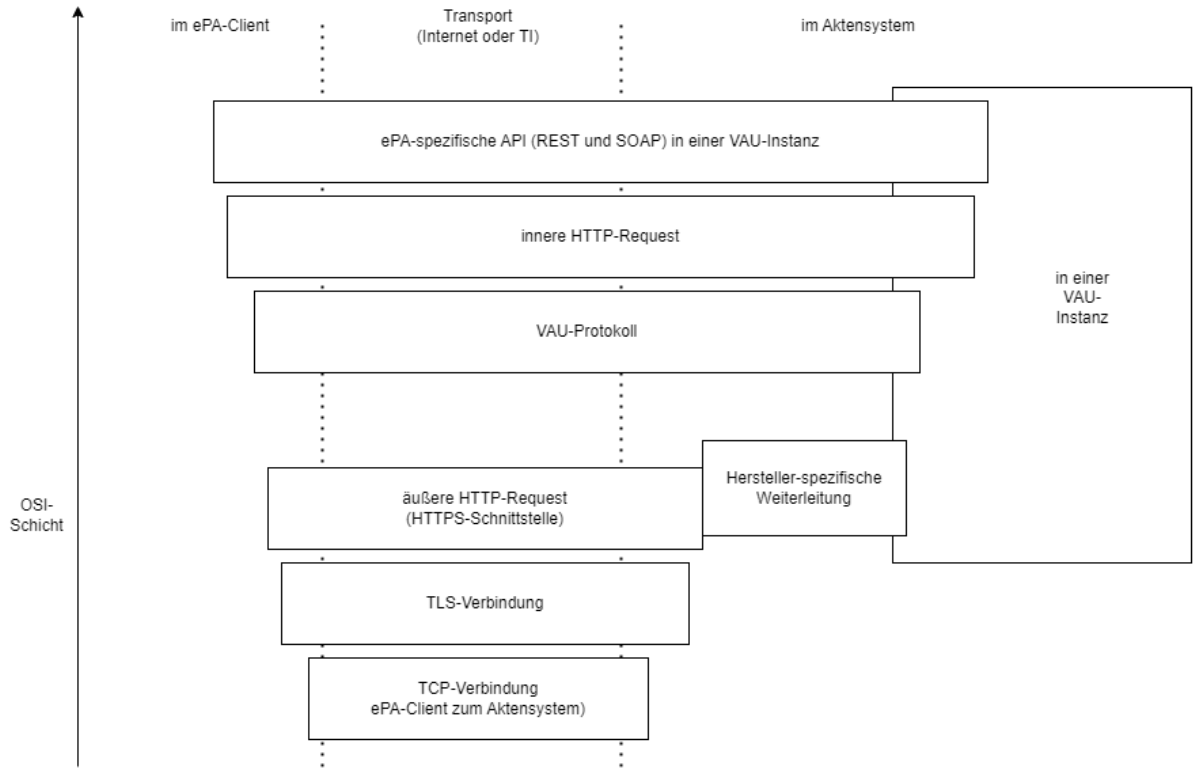

Abbildung 5: OSI-Schichten in einem Aktensystem

Wie im OSI-Referenzmodell üblich, ist das VAU-Protokoll agnostisch gegenüber dem über ihm transportierten Daten oder Protokollen. Das VAU-Protokoll ermöglich einen Ende-zu-Ende gesicherten Transport von Daten, bei dem es irrelevant ist, welche Protokolle unter oder über ihm in Bezug auf das OSI-Modell verwendet werden.

A_24656 -ePA, Unabhängigkeit von TLS-Ebene und VAU-Protokoll-Ebene
Ein ePA-Aktensystem und ein ePA-Client MÜSSEN sicherstellen, dass die TLS-Ebene unabhängig zur VAU-Protokoll-Ebene ist, i. S. v. es gibt keine Bindung zwischen TLS-Session und VAU-Protokoll-Verbindung.
Es MUSS möglich sein, die TLS-Verbindung, über die erfolgreich ein VAU-Protokoll-Handshake stattgefunden hat, zu beenden und auf einer neuen unabhängigen TLS-Verbindung weiter das VAU-Protokoll mit schon ausgehandelten Verbindungsschlüsseln zu verwenden.**[<=]**

Das VAU-Protokoll ermöglicht (zunächst) eine einseitig authentisierte Schlüsselaushandlung zwischen Client (FdV, LEI-PVS/AVS/KIS, KTR-Consumer) und Server (VAU-Instanz). Ähnlich wie bei einer TLS-Verbindung werden nach einer Schlüsselaushandlungsphase symmetrische Schlüssel gemeinsam berechnet und diese symmetrischen Schlüssel werden anschließend für die Sicherung der nun folgenden Nutzdaten auf Anwendungsebene verwendet. Direkt nach erfolgreich abgeschlossener Schlüsselaushandlung verwendet ein Client innerhalb der etablierten VAU-Protokoll-Verbindung auf einer höheren OSI-Schicht das OIDC-Protokoll für die Nutzer-Authentisierung gegenüber der VAU-Instanz. Ist die Authentifizierung erfolgreich gewesen, vermerkt die VAU-Instanz bei den ausgehandelten symmetrischen Schlüsseln diesen Authentisierungsstatus in den Metadaten für diese Verbindungsschlüssel. Ab dann besteht eine beidseitig authentisierte über das VAU-Protokoll gesicherte Verbindung insbesondere mit den gemeinsam berechneten symmetrischen Verbindungsschlüsseln.
Das VAU-Protokoll verwendet kryptographisch für die Schlüsselaushandlung das KEM-TLS-Protokoll ([KEM-TLS], [IETF-KEM-TLS]). Die Kodierung der KEM-TLS-Nachrichten muss spezifisch für die gewählte Architektur bei ePA für alle angepasst werden. Für die Schlüsselaushandlung wird ein PQC-sicheres Hybridverfahren auf Basis von Kyber768 und ECDH analog zu [IETF-Hybrid-TLS] verwendet. Die Verwendung von PQC-sicheren Hybridverfahren findet man ebenfalls bei Messengern [PQC-Hybrid-Signal] oder bei Web-Browsern [PQC-Hybrid-Chrome]. Das VAU-Protokoll erreicht Forward-Secrecy bei der Schlüsselaushandlung -- eine etwaige Kompromittierung von Langzeit-Schlüsseln einer VAU beeinflusst nicht die Sicherheitseigenschaften von VAU-Verbindungen der Vergangenheit. Nach der Schlüsselaushandlung wählt ein Client zufällig für jeden Request eine Request-ID, die der Server in der Response aufführt. Somit kann ein Client Request-Response-Paare sicher zuordnen und kann ebenfalls Replay-Angriffe abwehren. Im Vergleich zur VAU-Protokoll-Variante aus Abschnitt5 (E-Rezept aktuelle Ausbaustufe) wird ca. ab der vierten Nutzdaten-Nachricht ein Geschwindigkeitsvorteil erreicht, weil für die direkte Sicherung der Nutzdaten keine asymmetrischen Verfahren mehr verwendet werden (vgl. im Gegensatz dazu A_20161-* Punkt 6). Die Forward-Secrecy und die Post-Quantum-Resistenz wird bei der VAU-Protokoll-Variante aus Abschnitt 5 nicht erreicht -- es wird also mit der Anpassung des VAU-Protokolls, die aufgrund der Authentisierungsveränderung unabdingbar ist, sowohl eine Verbesserung der Sicherheitseigenschaften als auch der Performanz erreicht.

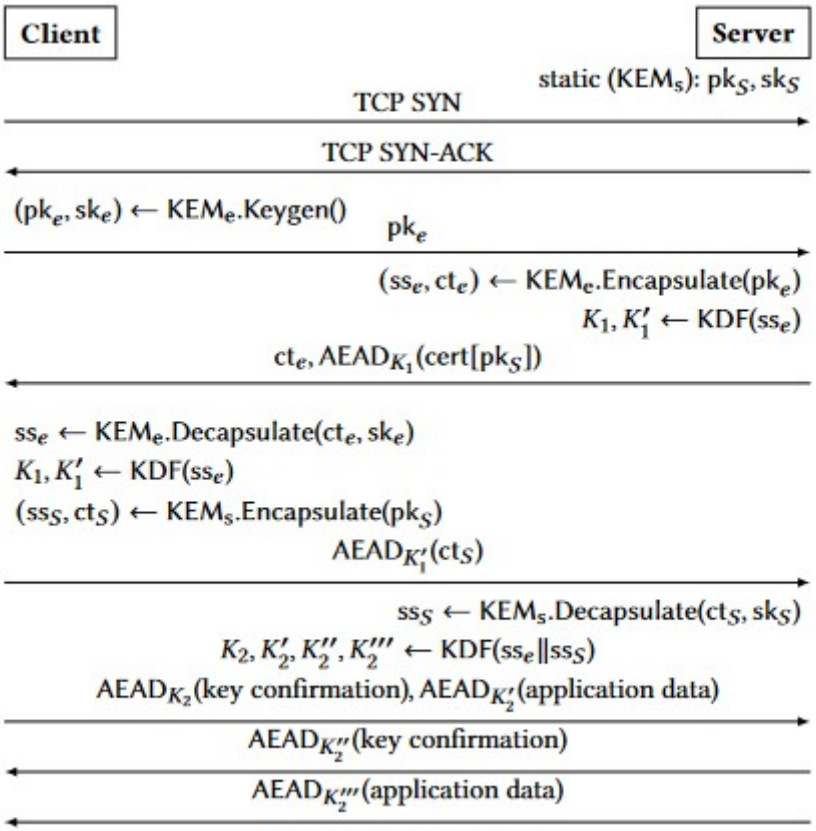

Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS]

Die gematik stellt Beispielimplementierungen des VAU-Protokolls in verschiedenen Programmiersprachen zur Verfügung.

6.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung

Bei im Prinzip allen in der Praxis verwendeten kryptographischen Protokollen gibt es mehrere Phasen im Protokoll: Zunächst erfolgt eine Schlüsselaushandlung bei der hauptsächlich asymmetrische kryptographische Verfahren eingesetzt werden. Als Ergebnis der Schlüsselaushandlung werden symmetrische Schlüssel (AES-Schlüssel) von Client und Server gemeinsam berechnet. Diese symmetrischen Schlüssel werden anschließend für die kryptographische Sicherung der Nutzdaten verwendet. Es werden wie üblich je nach Kommunikationsrichtung (Client->Server, Server->Client) unterschiedliche Schlüssel verwendet (analog zu TLS, IPsec/IKE, SSH etc.). Die symmetrischen Schlüssel sind dann nutzerspezifisch (und je nach gewählter Implementierungs-Architektur im Aktensystem auch VAU-Instanz-spezifisch). Sie sind nicht aktenspezifisch. D. h., dieselben schon ausgehandelten symmetrischen Schlüssel können für Zugriffe auf mehrere Akten bspw. durch ein PVS verwendet werden.

Die Schlüsselaushandlung beim VAU-Protokoll erfolgt nach [KEM-TLS] und benötigt vier Nachrichten (zwei Round-Trips), die zwischen Client und VAU ausgetauscht werden. Der kryptographische Verbindungsaufbau ist in [KEM-TLS] genauer beschrieben und ebenfalls

die kryptographische Funktion der einzelnen Schlüsselableitungen. Im Folgenden wird eine vereinfachte Übersicht als Verständnishilfe aufgeführt.

Nachricht	**Aktion**
Nachricht 1	Client -> VAU Ein Client erzeugt zwei Schlüsselpaare (ECDH, Kyber768) (analog zu [IETF-Hybrid-TLS]) und sendet die beiden öffentlichen Schlüssel an die VAU (= Nachricht 1). Bei diesen Schlüsselpaaren handelt es sich also um ephemere Schlüssel des Clients, d. h. pro Verbindungsaufbau/Handshake werden diese vom Client neu erzeugt.
Nachricht 2	VAU -> Client Die VAU verwendet die öffentlichen Schlüssel jeweils mittels eines KEM (Encapsulate), um zwei gemeinsame Geheimnisse (ss_e_ecdh, ss_e_kyber768) und zwei KEM-Chiffrate zu erzeugen. Diese Geheimnisse fließen in eine KDF (RFC-5869/SHA-256) ein (Basis: ss_e = ss_e_ecdh || ss_e_kyber768) und damit werden die symmetrischen Schlüssel K1_c2s und K1_s2c erzeugt. Wie bei TLS 1.3 üblich, werden diese Schlüssel im weiteren für die Sicherung von Daten in der Aushandlungsphase des Protokolls für die Kommunikation zum Server (K1_c2s) und zum Client (K1_s2c) verwendet. Die KEM-Chiffrate und über AES/CGM verschlüsselte semi-statische VAU-Schlüssel (A_24425-*) werden an den Client gesendet (= Nachricht 2). Verständnishinweis: Die authentisierte Verschlüsselung der öffentlichen VAU-Schlüssel (A_24425-*) erfolgt aufgrund des Grundprinzips bei TLS 1.3 möglichst früh, um Daten auch in der Handshake-Phase zu verschlüsseln. Die öffentlichen VAU-Schlüssel (A_24425-*) sind nicht vertraulich.
Nachricht 3	Client -> VAU Mittels der KEM-Chiffrate aus Nachricht 2 und der privaten Schlüssel des Clients (vgl. Nachricht 1: ECDH, Kyber768) berechnet der Client K1_c2s und K1_s2c und kann die öffentlichen VAU-Schlüssel erfolgreich entschlüsseln (AEAD-Chiffre). Er verwendet die öffentlichen VAU-Schlüssel (ECDH, Kyber768) und erhält mittels zwei KEM-Encapsulate-Berechnungen zwei Geheimnisse und zwei KEM-Chiffrate. Er berechnet aus den zwei Geheimnissen: ss_s = ss_s_ecdh || ss_s_kyber768. Diese KEM-Chiffrate verschlüsselt er über AES/GCM mittels des Schlüssels K1_c2s und erhält ein Ergebnis-Chiffrat. Mittels einer KDF auf Basis von ss = ss_e || ss_s werden abgeleitet: vier vertrauliche AES/GCM-Schlüssel:

	• K2_c2s_key_confirmation, • K2_c2s_app_data, • K2_s2c_key_confirmation, und • K2_s2c_app_data und eine (nicht vertrauliche) KeyID. Der Client erzeugt den Hashwert des Transskript der zuvor ausgetauschten Daten (analog TLS). Diesen Hashwert verschlüsselt er mittels des Schlüssels K2_c2s_key_confirmation (Ziel: Key-Confirmation). Der Client sendet das Ergebnis-Chiffrat (K1_c2s) und das Key-Confirmation-Chiffrat (K2_c2s_key_confirmation) an die VAU (= Nachricht 3).
Nachricht 4	VAU -> Client Mittels der zwei KEM-Chiffrate aus Nachricht 3 und der KEM-Decapsulation-Funktionen werden die beiden Geheimnisse ss_s_ecdh und ss_s_kyber768 berechnet, und anschließend ss_s= ss_s_ecdh || ss_s_kyber768 berechnet. Analog wie im Client bei Nachricht 3 werden die dort aufgeführten vier vertraulichen AES/GCM-Schlüssel und die nicht-vertrauliche KeyID abgeleitet auf Basis von ss = ss_e || ss_s. Mit Hilfe des Schlüssels K2_c2s_key_confirmation und der im Server gespeicherten vorherigen Nachrichten prüft der Server das vom Client gesendete Transskript-Chiffrat bzw. den Transskript-Hash darin. Der Server erzeugt selbst einen Transskript-Hash (Hashwert der Konkatenation Nachricht 1 || Nachricht 2 || Nachricht 3) und sendet diesen mittels K2_s2c_key_confirmation verschlüsselt an den Client zurück.

Client und VAU prüfen die jeweiligen Key-Confirmation (jeweils mittels der Schlüssel K2_c2s_key_confirmation und K2_s2c_key_confirmation), bevor sie die Schlüssel K2_c2s_app_data und K2_s2c_app_data in der nächsten Phase des Protokolls -- Sicherung der Nutzdaten -- verwenden.

A_24425-01 -VAU-Protokoll: VAU-Schlüssel für die VAU-Protokoll-Schlüsselaushandlung
Ein Aktensystem MUSS sicherstellen, dass

1. es eine Signatur-Identität aus der Komponenten-PKI der TI gibt, die technisch sichergestellt ausschließlich nur von VAU-Instanzen verwendbar ist (AUT-Zertifikat und Schlüsselmaterial (VAU-HSM) wie bei ePA 1x. und 2.x).
2. es semi-statische Schlüsselpaare für ECDH (auf Basis Kurve P-256 [FIPS-186-5]) und Kyber768 [IEFT-Kyber] gibt, deren private Schlüssel, technisch sichergestellt, ausschließlich von VAU-Instanzen verwendbar sind.
3. die privaten Schlüssel in einer VAU-Instanz erzeugt und verarbeitet werden (also nicht im VAU-HSM),
4. die semi-statischen Schlüssel eine maximale Lebensdauer von einem Monat besitzen (Hinweis: die Forward-Secrecy hängt nicht vom Wechselintervall ab, innerhalb eines

Verbindungsaufbaus und der Schlüsselaushandlung dabei fließen ephemere Schlüsselwerte von Client und Sever ein).

5. die semi-statischen Schlüssel in einer über die Signatur-Identität authentisierten folgenden Datenstruktur aufgeführt werden.

Struktur der signierten semi-statischen öffentlichen VAU-Schlüssel

```
VAU_Keys = {
          "ECDH_PK" :
            { "crv" : "P-256",
              "x" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),
              "y" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),
            },
          "Kyber768_PK" : Binärwert-öffentlicher-Schlüssel-nach-keygen-
Spec-Kyber768,
          "iat" : Erzeugungszeits-Sekunden-Since-Epoch (integer),
          "exp" : Nicht-mehr-Verwendbar-nach (integer),
          "comment" : "Erzeugt bei VAU-Instanz xyz, Meta-Info abcd"
}
```

In "comment" KÖNNEN beliebige Text-Daten aufgeführt werden. Es können weitere Attribute hinzugeführt werden. Ein Client MUSS ihm unbekannte Attribute ignorieren. Diese Struktur wird mittels CBOR [RFC-CBOR] binär kodiert und im Folgenden VAU_Keys_encoded genannt.

Diese binäre Byte-Folge wird in folgende Datenstruktur eingebracht

```
{
 "signed_pub_keys" : VAU_Keys_encoded,
 "signature-ES256" : ECDSA-Signatur-SHA-256-analog-RFC-7515 (R||S => 64
Byte) binär,
 "cert_hash"       : SHA-256-Wert des "signierenden" AUT-VAU-Zertifikats,
 "cdv"             : Cert-Data-Version (natürliche Zahl, beginnend mit 1,
vgl. A_24957-*),
 "ocsp_response"   : OCSP-Response-für-das-VAU-Signaturzertifikat-nicht-
älter-als-24-Stunden-DER-Kodierung
}
```

Diese Datenstruktur wird mittels CBOR binär kodiert (serialisiert). Das Ergebnis der Kodierung wird "signierte öffentliche VAU-Schlüssel" (Plural) genannt.

[<=]

A_24957 -VAU-Protokoll: Verfügbarmachung des AUT-VAU-Zertifikat plus Prüfkette
Ein Aktensystem MUSS über an seinen Webschnittstellen mittels eines äußeren HTTPS-Requests per HTTP-GET unter dem Pfadnamen /CertData.<SHA-256-Hashwert-Hex-[0-9a-f]>-Versionszahl (a-f kleingeschrieben) folgende Datenstruktur zur Verfügung stellen:

```
{
   "cert": DER-kodiertes-AUT-VAU-Zertifikat,
   "ca"  : DER-kodiertes-Komponenten-PKI-CA-aus-dem-"cert"-kommt,
   "rca_chain" : [Cross-Zertifikat-1, ..., Cross-Zertifikat-n],
}
```

Die Versionszahl MUSS eine natürliche Zahl sein, beginnend mit 1, die es trotz A_24958-* erlaubt, Fehler in den Daten zu korrigieren (siehe Erläuterung nach A_24957-*).

Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert/kodiert werden und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) an den VAU-Client als Response auf den GET-Request gesendet werden.

In "rca_chain" MÜSSEN alle Cross-Zertifikate in chronologischer Ordnung von RCA5 ausgehend aufgeführt werden, bis die Root-Schlüssel (Cross-Zertifikat) erreicht werden, mit denen das "ca"-Zertifikat bestätigt (signiert) wurde; d. h., sozusagen eine einfach verkettete Liste von Cross-Zertifikaten chronologisch aufsteigend.
[<=]

Erläuterung:

In A_24425-* werden die "signed_pub_keys" mit dem Schlüsselmaterial der AUT-VAU-Identität des Aktensystems authentisiert, analog zum VAU-Protokoll ePA 1.x und 2.x. Dieses Zertifikat ist über die Signaturkettenprüfung der TI-PKI prüfbar. Dafür benötigt der VAU-Client die Zertifikate, die die Prüfkette ausmachen. Als Vertrauensanker besitzt der Client entweder X.509-Root-Zertifikate, und zwar mindestens RCA5 (PU), oder die TSL. Die Konstruktion nach A_24957-* stellt die Zertifikate für die Prüfkette und das AUT-VAU-Zertifikat selbst ("cert"-Feld) zur Verfügung. Aufgrund der Konstruktion kann man davon ausgehen, dass die Daten in solch einer Datei sich nie ändern werden, i. S. v. einmal vom Client per GET bezogen werden und dann ewig gespeichert werden kann. Sollte sich nach einem Deployment der CertData-Datei (i. S. v. einige Clients haben diese Datei schon bezogen) bspw. CertData-e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855-1 herausstellen, dass dort Fehler enthalten sind, dann kann man über die Versionsnummer im Handshake (vgl. "cdv"-Feld, A_24425-*) und der Erstellung einer neuen CertData-e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855-2 diese Fehler im laufenden Betrieb korrigieren.

A_24958 -VAU-Protokoll: VAU-Client Prüfbasis Zertifikatsprüfung
Ein VAU-Client (PVS, ePA-FdV etc.) MUSS als Prüfbasis eine Root-Version (X.509-Root-TI-Zeritifikat), die mindestens 2 Jahre alt ist, verwenden oder die TSL. Der VAU-Client MUSS die Operation nach A_24957-* verwenden, um die für die Zertifikatsprüfung von A_24425-* notwendigen Zertifikate zu beziehen. Die bezogenen Zertifikatsdaten MUSS der VAU-Client lokal (zeitlich unbegrenzt) vorhalten (Caching). Bei Erhalt einer Nachricht 2 (A_24608-*) MUSS er zunächst prüfen, ob er die für die Zertifikatsprüfung notwendigen Zertifikate im lokalen Cache vorrätig hat, und falls ja MUSS er diese verwenden.
[<=]

A_24427 -VAU-Protokoll: VAU-Schlüssel für die VAU-Protokoll-Schlüsselaushandlung
Ein Aktensystem KANN in Bezug auf die Erzeugung der signierten öffentlichen VAU-Schlüssel (vgl. A_24425-*) und der Erzeugung und der Vorhaltung der dazu passenden zwei privaten Schlüssel (ECDH, Kyber768) zwischen zwei Modellen wählen:
(1)
Im Normalfall wird eine VAU-Instanz "auf Vorrat" im Aktensystem gestartet. Beim Starten der Instanz erzeugt die Instanz die beiden Schlüsselpaare und signierte diese mittels der im VAU-HSM befindlichen Signaturidentität (gemäß A_24425-*). Die privaten Schlüssel bleiben genau nur in dieser VAU-Instanz.
(2)
Eine VAU-Instanz erzeugt regelmäßig die Schlüsselpaare und anschließend die

Datenstruktur der signierten öffentlichen VAU-Schlüssel. Die privaten Schlüssel werden auf sichere Weise bei den VAU-Instanzen verfügbar gemacht.**[<=]**

Erläuterung: Je nach verwendeten Implementierungs-Architektur und den zur Implementierung verwendeten HSM (die recht unterschiedliche Schlüsselverteilungsmethoden haben), kann für einen Hersteller ein Modell einfacher umzusetzen sein als das andere. Die Motivation der Anforderung ist festzustellen, dass beide Modelle explizit erlaubt sind.

A_24757-01 -VAU-Protokoll: Nutzerpseudonym
Ein VAU-Client MUSS die Variable "vau-np" aus vorherigen VAU-Protokoll-Verbindungen speichern und bei den folgenden VAU-Protokoll-Handshakes bei der Nachricht 1 (vgl. A_24428-*) im HTTP-Request-Header aufführen (Hinweis: zu dem Zeitpunkt kann es noch keine inneren HTTP-Request geben / Verbindungsaufbau). Initial hat ein Client diesen Wert noch nicht, dann führt er die Variable nicht im Request-Header auf.
Ein VAU-Client MUSS nach einer erfolgreichen Client-Authentisierung (vgl. [gemSpec_Krypt#7.3 und 7.4]) die innere HTTP-Response prüfen, ob die Variable "VAU-NP" (bzw. vau-np) aufgeführt ist, falls ja MUSS er einen lokal gespeicherten Wert ggf. aktualisieren (i. S. v. ein Aktensystem (VAU-Instanz) wird dieses Nutzerpseudonym auch wechseln).
[<=]

Erläuterung: Das VAU-NP erleichtert die Implementierung im Aktensystem. Es ist für die Implementierung günstig, wenn ein Nutzer mit möglichst vielen seiner Requests (bzw. neuer VAU-Verbindungen) in derselben VAU-Instanz landet. Die Variable dient dem Routing nach der HTTPS-Schnittstellen des Aktensystems (Verteilung der Requests auf die verschiedenen VAU-Instanzen, vgl. Abschnitt 6.5- Routing auf VAU-Instanzen).

Bestimmte Clients (bspw. der E-Rezept-FD in der Rolle als ePA-Client) werden ggf. auf mehrere VAU-Instanz verteilt, die konkrete Ausgestaltung liegt in der Entscheidungshoheit des Aktensystemherstellers.

Vgl. auch A_24773-* (Neustart der Verbindung bei unterschiedlichen Nutzergeräten im Erstnutzungsfall).

A_24428 -VAU-Protokoll: VAU-Client: Nachricht 1
Ein VAU-Client MUSS für die Erzeugung folgende Schritte durchlaufen.
Ein VAU-Client MUSS

1. ein ECC-Schlüsselpaar auf der Kurve P-256 [FIPS-186-5] erzeugen.
2. ein Kyber768-Schlüsselpaar [IETF-Kyber] erzeugen.

Anschließend MUSS er die öffentlichen Schlüssel in folgende Datenstruktur überführen

```
{
    "MessageType" : "M1",
    "ECDH_PK"    :  { "crv" : "P-256",
                      "x" : analog A_24425-*,
                      "y" : analog A_24425-* },
    "Kyber768_PK" : analog zu A_24425-*
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] in eine Bytefolge kodieren (serialisieren).
Diese Bytefolge ist die Nachricht 1.

Diese Nachricht 1 MUSS der VAU-Client an die HTTPS-Schnittstelle des Aktensystem per POST auf den Pfad /VAU senden, wobei er den Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) verwendet und die Nachricht 1 im Request-Body aufführt.

Vgl. auch A_24757-*
[<=]

Hinweis: Für ein ePA-FdV ist das Access Gateway im Internet und dessen HTTPS-Schnittstelle der Zugang zum Aktensystem, an den das ePA-FdV seinen Request (bspw. A_24428-*) sendet. Ein ePA-Client aus der TI verwendet direkt die äußere HTTPS-Schnittstelle eines Aktensystems innerhalb der TI (analog zu ePA 1.x und 2x.).

A_24429 -VAU-Protokoll: VAU-Instanz: Erhalt von Nachricht 1
Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die für Nachricht 1 erzeugten Datenobjekte nach A_24428-* verarbeiten können.**[<=]**

A_24619 -VAU-Protokoll: AES/GCM-Verschlüsselung im Handshake
Eine VAU-Instanz und ein ePA-Client verschlüsseln im Rahmen des Handshakes des VAU-Protokolls verschiedene Nachrichten-Teile mittels AES/GCM. Dabei MÜSSEN sie pro Verschlüsselung den IV jeweils zufällig als 96-Bit-Wert erzeugen. Der Authentication-Tag MUSS 128 Bit lang sein. Das Ergebnis der Verschlüsselung MUSS dann in der folgenden Kodierung aufgeführt werden:
IV || eigentliche AES-GCM-Chiffrat || 128-Bit langer Authentication-Tag.**[<=]**

A_24608 -VAU-Protokoll: VAU-Instanz: Nachricht 2
Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die beiden öffentlichen Schlüssel aus Nachricht 1 (vgl. A_24428-*) prüfen (korrekte ECC-Kurve ("crv":"P-256"), öffentlicher Punkt liegt auf der Kurve P-256 [FIPS-186-5], der öffentliche Kyber768-Schlüssel ist valide [IETF-Kyber]).
Sie MUSS für ECDH und Kyber768 jeweils die KEM-Encapsulate-Funktion verwenden und erhält dabei zwei Geheimnisse (ss_e_ecdh, ss_e_kyber768) und zwei Ciphertexte (ECDH_ct, Kyber768_ct).
Sie MUSS für die beiden Geheimnisse zusammenfügen: ss_e = ss_e_ecdh || ss_e_kyber768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = '' (leere Zeichenkette)), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1_c2s und die letzten 32 Byte heißen K1_s2c.

Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_24619-*) MÜSSEN die "signierten öffentlichen VAU-Schlüssel" (vgl. A_24425-*) verschlüsselt werden. Das Ergebnis (vgl. A_24619-*) heißt aed_ciphertext_msg_2.

Sie MUSS folgende Datenstruktur erzeugen:

```
{
    "MessageType" : "M2",
    "ECDH_ct"     :  ... analog ECDH_PK aus A_24428-* ...,
    "Kyber768_ct" : Kyber768-Ciphertext analog [IETF-Kyber], vgl. Referenz-
Implementierung Kyber768,
    "AEAD_ct"     : aead_ciphertext_msg_2
}
```

Diese Datenstruktur MUSS sie mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 2.

Sie MUSS einen ID erzeugen, kodiert aus der Zeichenmenge
A-Za-z0-9-/
die maximal 200 Zeichen lang ist. Die ID MUSS mit "/" (Slash) beginnen und MUSS ein gültiger URL-Pfadname sein. Diese ID MUSS es dem Aktensystem ermöglichen, die VAU-Instanz und den aktuellen Handshake bei Eintreffen der Nachricht-3 des ePA-Clients, der diese ID mitsendet, zu identifizieren. Das Aktensystem kann die Struktur der ID selbst definieren. Ein ePA-Client MUSS die ID als opake Zeichenkette behandeln.

Sie MUSS die Nachricht 2 inkl. ID an die HTTPS-Schnittstelle des Aktensystem übergeben (die Art der Verbindung zwischen VAU-Instanz und HTTPS-Schnittstelle des Aktensystems kann ein Hersteller frei wählen). Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 1 eingetroffen ist, die Nachricht 2 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "VAU-CID" die ID aufgeführt werden.**[<=]**

Hinweis: Bei der KEM-Encapsulate-Funktion fließt Zufall aus dem System ein. Für die VAU-Instanzen (und auch für die ePA-Clients) gelten die Vorgaben aus Abschnitt 2.4 (Güte der Zufallserzeugung, Zuweisung über den Produkttypsteckbrief).

Beispiele für die ID aus A_24608:

- /VAU/ 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08/1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
- /VAU/A-XYZ/ 1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

A_24622 -VAU-Protokoll: VAU-Client: Erhalt von Nachricht 2
Ein VAU-Client MUSS die für Nachricht 2 erzeugten Datenobjekte nach A_24608-* verarbeiten können.
Er MUSS prüfen, ob

1. im HTTP-Response-Header die Variable "VAU-CID" enthalten ist, falls nicht Abbruch.
2. der Wert der Variable eine Bytefolge ist, dessen Länge maximal 200 Byte lang ist und die nur die Zeichen
   A-Za-z0-9-/
   enthält und mit "/" (Slash) beginnt. Falls nicht Abbruch.

Er MUSS den Wert als Pfad für das Versenden der Nachricht 3 (vgl. A_24623) und aller weiteren Nachrichten -- also auch nach dem erfolgreichen Handshake -- im Kontext dieser Verbindung verwenden.**[<=]**

A_24623 -VAU-Protokoll: VAU-Client: Nachricht 3
Ein VAU-Client MUSS aus der Nachricht 2 (vgl. A_24622-*) der VAU-Instanz mittels der Ciphertexte ECDH_ct und Kyber768_ct und den privaten ephemeren Client-Schlüsseln aus A_24428-* (Nachricht 1) und der jeweiligen KEM-Decapsulation-Funktionen zwei Geheimnisse berechnen: ss_e_ecdh und ss_e_kyber768. Er MUSS die beiden Geheimnisse zusammenfügen: ss_e = ss_e_ecdh || ss_e_kyber768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = '' (leere Zeichenkette), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1_c2s und die letzten 32 Byte heißen K1_s2c.

Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_24619-*) MUSS der Ciphertext "AEAD_ct" entschlüsselt werden: die "signierten öffentlichen VAU-Schlüssel" (vgl. A_24425-*) werden so als Klartext erhalten. Diese VAU-Schlüssel MUSS er nach A_24624-* prüfen. Mittels der öffentlichen Schlüssel (ECDH_PK, Kyber768_PK) MUSS er jeweils die KEM-Encapsulation-Funktion ausführen und er erhält zwei Geheimnisse: ss_s_ecdh und ss_s_kyber768. Diese führt er zusammen: ss_s = ss_s_ecdh || ss_s_kyber768. Weiter berechnet er ss = ss_e || ss_s.
Dieses Geheimnis verwendet die HKDF [RFC-5859] auf Basis von SHA-256 (info = '' (leere Zeichenkette)), um 160 Byte (=5 * 32 Byte) abzuleiten. Er MUSS diese 160 Byte in 32 Byte-Blöcke (von offset 0 bis zum Ende) auf folgende fünf Variablen (vier Schlüssel + eine KeyID) verteilen:

- K2_c2s_key_confirmation,
- K2_c2s_app_data,
- K2_s2c_key_confirmation,
- K2_s2c_app_data

und

- KeyID (nicht vertraulich).

Diese ersten vier sind vertrauliche Schlüsselwerte für AES/GCM. Die KeyID wird nach dem Handshake als eindeutige ID für die K2*_app_data Schlüssel dienen.

Er MUSS eine Datenstruktur wie folgt erzeugen:

```
{
    "ECDH_ct"    : client_kem_result_2["ECDH_ct"],
    "Kyber768_ct" : client_kem_result_2["Kyber768_ct"],
    "ERP" : False,
    "ESO" : False
}
```

ERP steht für "Enforce Replay Protection" und ESO steht für "Enforce Sequence Order". Innerhalb der Spezifikation heißt diese Datenstruktur Nachricht_3_inner_Layer. Diese Datenstruktur MUSS er per CBOR [RFC-CBOR] serialisieren/kodieren. Diese Serialisierung MUSS er mittels K1_c2s verschlüsseln (vgl. A_24619) (= "ciphertext_msg_3).
Er MUSS die komplette Nachricht-1 (CBOR-Kodierung), die Nachricht-2 und ciphertext_msg_3 konkatenieren (= Transskript des Client) und davon den SHA-256-Hashwert berechnen. Diesen Hashwert MUSS er mittels K2_c2s_key_confirmation verschlüsseln (vgl. A_24619), das Chiffrat sei
als aead_ciphertext_msg_3_key_confirmation hier bezeichnet.

Dann MUSS er folgende Datenstruktur erzeugen:

```
{
     "MessageType" : "M3",
     "AEAD_ct"     : ciphertext_msg_3,
     "AEAD_ct_key_confirmation" : aead_ciphertext_msg_3_key_confirmation
}
```

Diese Datenstruktur MUSS er per CBOR serialisieren, das Ergebnis ist Nachricht 3.
Er MUSS die Nachricht 3 per äußeren HTTP-Request an das Aktensystem senden und dabei den Wert der VAU-CID (vgl. A_24622) als URL-Pfadnamen verwenden, unter Verwendung der HTTP-POST-Methode.
[<=]

A_24624-01 -VAU-Protokoll: VAU-Client: Prüfung der "signierten öffentlichen VAU-Schlüssel"
Ein VAU-Client MUSS die "signierten öffentlichen VAU-Schlüssel" (vgl. A_24425-*) bei der Verarbeitung von Nachricht 3 (vgl. A_24623-*) wie folgt prüfen. Erhält er bei einem der folgenden Prüfpunkte kein positives Prüfergebnis, so MUSS er die Verarbeitung (Handshake) abbrechen.

1. Prüfung des TI-Zertifikats, das den Hashwert aus dem "cert_hash"-Datenfeld besitzt (Bezug des Zertifikats vgl. A_24957-*), u. a. unter der Verwendung der OCSP-Response aus "ocsp_response" für die Prüfung des Sperrstatus (Prüfung ob "good"). Die OCSP-Response darf dabei nicht älter als 24 Stunden sein.

2. Das VAU/TI-Zertifikat MUSS zeitlich gültig sein. Es MUSS kryptographisch in einer Zertifikats-/Signaturprüfungskette rückführbar auf eine X.509-Root-Version der TI-PKI sein.
3. Das TI-Zertifikat MUSS aus der Komponenten-PKI der TI stammt (vgl. Implementierungshinweis A_25192-*#Punkt-2) und die Rollen-OID "oid_epa_vau" besitzt.
4. Die Signatur im "signature-ES256"-Datenfeld MUSS eine valide Signatur für die Daten im Datenfeld "signed_pub_keys" (Signaturprüfung ergibt "valid/accept") sein, unter Verwendung des öffentlichen Signatur-Schlüssels aus dem VAU/TI-Zertifikats bei der Signaturprüfung.
5. Der ECC-Schlüssel in signed_pub_keys (vgl. Erzeugung bei A_24425) MUSS ein gültiger Punkt der Kurve P-256 [FIPS-186-5] sein. Der öffentliche Schlüssel in "Kyber768_PK"-Datenfeld MUSS ein gültiger Kyber-768-Schlüssel [IEFT-Kyber] sein.
6. Die Zeit in "signed_pub_keys.exp" MUSS größer als die aktuelle Systemzeit (Seconds since epoch) sein, d. h. die beiden Schlüssel (ECDH_PK und Kyber768_PK) sind noch zeitlich gültig.

[<=]

A_24625 -VAU-Protokoll: VAU-Instanz: Erhalt von Nachricht 3
Eine VAU-Instanz (Server im VAU-Protokoll) MUSS die für Nachricht 3 erzeugten Datenobjekte nach A_24623-* verarbeiten können.**[<=]**

A_24626 -VAU-Protokoll: VAU-Instanz: Nachricht 4
Eine VAU-Instanz MUSS bei Erhalt der Nachricht 3 das Chiffrat AEAD_ct mittels des Schlüssels K1_c2s entschlüsseln. Schlägt dies fehl, MUSS sie den Verbindungsaufbau mit einem Fehler (vgl. A_24635-*) abbrechen. Sie MUSS analog wie der Client (vgl. A_24623-*) die Schlüsselerzeugung der folgenden Schlüssel durchführen, nur dass sie dafür die KEM-Decapsulation-Methode verwendet:

- K2_c2s_key_confirmation,
- K2_c2s_app_data,
- K2_s2c_key_confirmation,
- K2_s2c_app_data

und

- KeyID (nicht vertraulich).

Sie MUSS analog zu A_24623-* das Transskript des Clients und dessen SHA-256-Wert berechnen. Sie MUSS mittels K2_c2s_key_confirmation das Chiffrat "AEAD_ct_key_confirmation" aus Nachricht 3 entschlüsseln und prüfen, ob der von ihr (= VAU-Instanz) berechnete Transskript-Client-Hashwert mit dem entschlüsselten Klartext übereinstimmt. Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen berechnete Transskript-Client-Hashwert und Klartext festgestellt, MUSS sie den Handshake mit einem Fehler abbrechen (vgl. A_24635-*).
Sie MUSS den Transskript der VAU-Instanz als die Konketation von Nachricht1 || Nachricht 2 || Nachricht 3 berechnen, davon den SHA-256-Hashwert berechnen und diesen mittels K2_s2c_key_confirmation verschlüsseln (vgl. A_24619-*), und erhält ein Chiffrat genannt "AEAD_ct_key_confirmation".
Sie MUSS die folgende Datenstruktur erzeugen:

```
{
     "MessageType" : "M4",
     "AEAD_ct_key_confirmation" : Chiffrat-AEAD_ct_key_confirmation
```

}
Diese Datenstruktur MUSS sie mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 4.

Sie MUSS die Nachricht 4 inkl. ID (vgl. A_24608-*) an die HTTPS-Schnittstelle des Aktensystem übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 3 eingetroffen ist, die Nachricht 4 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "VAU-CID" die ID aufgeführt werden.**[<=]**

Erläuterung:

Die Aufführung von "VAU-CID" im Response-Header ist eigentlich nicht mehr absolut notwendig, da der Client diese schon bei Nachricht 2 erhalten und als URL-Pfadname ab jetzt im Laufe der weiteren Verbindung verwenden wird (A_24622-*). Die Absicht der Aufführung ist, eine etwaige Fehlersuche bspw. bei IOP-Tests zu unterstützen.

A_24627 -VAU-Protokoll: VAU-Client: Erhalt von Nachricht 4
Ein VAU-Client MUSS die für Nachricht 4 erzeugten Datenobjekte nach A_24626-* verarbeiten können.

Er MUSS die Nachrichten 1, 2 und 3 konkatenieren und den SHA-256-Hashwert erzeugen. Dies ist der Transskript-Hashwert. Er MUSS mittels K2_s2c_key_confirmation das Chiffrat "AEAD_ct_key_confirmation" entschlüsseln (vgl. A_24619-*).
Der erhaltene Klartext ist der gesendete Transskript-Server-Hashwert.
Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen eben berechneten Transskript-Hashwert und gesendeten Transskript-Server-Hashwert (Klartext des Chiffrats) festgestellt, MUSS er den Handshake mit einem Fehler abbrechen.**[<=]**

6.2 Transport und Sicherung der Nutzdaten

A_24629 -VAU-Protokoll: VAU-Client: Verschlüsselungszähler
Ein VAU-Client MUSS sicherstellen, dass der Schlüssel K2_c2s_app_data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Der VAU-Client MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.**[<=]**

Hinweis: Ein Zählerüberlauf kann praktisch nie erreicht werden. Würde ein Client im Nano-Sekunden-Takt den Zähler erhöhen, würde erst nach mehr als 583 Jahren der Überlauf eintreten. Deshalb wird auf einen Überlauftest verzichtet.

A_24628-01 -VAU-Protokoll: VAU-Client: Request erzeugen/verschlüsseln
Ein VAU-Client MUSS einen inneren HTTP-Request als Klartext erzeugen.

Er MUSS den Klartext mittels AES/GCM und dem Schlüssel K2_c2s_app_data verschlüsseln (siehe AAD weiter unten).
Dafür wird ein 32-Bit Zufallswert a erzeugt. Nach A_24629-* wird der mit K2_c2s_app_data verbundene Zähler um eins erhöht. Es wird der IV mit IV=a || Zähler erzeugt (dessen Länge ist damit 96-Bit). Dieser IV wird für die AES/GCM-Verschlüsselung verwendet.

Er MUSS einen Request-Counter pflegen, der verbunden ist mit der KeyID. Für jeden

Request MUSS der VAU-Client diesen Request-Counter erhöhen, bei Empfang einer Antwort ist es dem Client möglich, die Response zum zuvor gestellten Request sicher zuzuordnen (vgl. A_24633). (vgl. Erläuterungen zu A_24628-*).

Es wird folgender Header erzeugt:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/ Request	1 Byte	Für eine Nachricht des ePA-Clients an eine VAU-Instanz wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrats hat dieses Byte den Wert 2, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_24623-*)

Dieser Header stellt die "Additional Associated Data" dar, die in die Berechnung des Authentication-Tag bei der AES/GCM-Verschlüsselung einfließen MÜSSEN. Der Authentication-Tag MUSS 128 Bit lang sein.

Das erweitere Chiffrat (also inkl. Header) MUSS folgende Struktur haben:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/ Request	1 Byte	Für eine Nachricht des ePA-Clients an eine VAU-Instanz wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrats wird es auf den Wert 2 gesetzt, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.

KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_24623-*)
IV	12 Byte (= 96 Bit)	IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A_24628-*)
CT	variabel	eigentliche AES/GCM-Chiffrat, dessen Länge gleich der Länge des Klartextes ist
GMAC-Wert	16 Byte (= 128 Bit)	Authentication-Tag, der während der AES/GCM-Verschlüsselung inkl. der Associated Data (Daten aus der Header-Tabelle, s. o.) berechnet wird.

Der VAU-Client MUSS diese Datenstruktur per HTTP-POST an die VAU-Instanz senden und als URL-Pfadnamen dabei den Wert der VAU-CID (vgl. A_24622-*) verwenden. Dabei MUSS er den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden. **[<=]**

Erläuterung:

Der Verschlüsselungszähler, der im Client mit dem Schlüssel K2_c2s_app_data verbunden ist, hat die Funktion für den Galois Counter Mode (GCM) sicherzustellen, dass der jeweils pro Verschlüsselungsvorgang erzeugte Initialisierungsvektor (IV) (bei gleichen Wert von K2_c2s_app_data) einzigartig ist.

Der Request-ID-Zähler hat drei Funktionen. Einmal soll im Client eine Response des Servers sicher einem vorher gesendeten Request zuordenbar sein. Sollte im Verbindungsaufbau "Enforce Replay Protection" aktiviert worden sein, prüft der Server, ob Requests mit gleicher Request-ID mehrfach eingetroffen sind (ähnlich der "Anti Replay Window"-Technik bei IPsec). Wenn analog "Enforce Sequence Order" aktiviert worden ist, dann prüft der Server die Folge der Request-ID der einkommenden Requests auf strenge Monotonie.

In der aktuellen Ausbaustufe von ePA für alle werden die letzten beiden Funktionen nicht benötigt (wie auch aktuell beim E-Rezept nicht).

Da Verschlüsselungszähler und Request-ID unterschiedliche Funktionen/Motivationen besitzen, sind sie als separate (theoretisch auch im Wert unterschiedliche) Datenobjekte aufgeführt.

A_24630 -VAU-Protokoll: VAU-Instanz: Request entschlüsseln/auswerten
Ein Request erreicht das Aktensystem über eine HTTPS-Schnittstelle. Im äußeren Request nimmt das Aktensystem das Routing mittels der Verbindungs-ID (URL-Pfadname, A_24622-* und A_24628-*) und/oder der KeyID im Header des Chiffrats vor.

Eine VAU-Instanz MUSS bei Erhalt eines Chiffrat nach A_24628-* folgendes prüfen.

1. Ist die Länge der Datenstruktur mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. S. v. korrekte Umgebung, in der auch die VAU-Instanz arbeitet.
3. Ist das Request/Response-Byte gleich 1.
4. Ist die KeyID bekannt.

Sollte eine dieser Prüfungen oder eine weitere der folgenden Prüfungen ein nicht-positives Prüfergebnis ergeben, so MUSS die VAU-Instanz die Verarbeitung des Requests

abbrechen und mit einer Fehlermeldung (vgl. A_24635-*) dem Client antworten.

Die VAU-Instanz MUSS den mit der KeyID verbundenen Schlüssel K2_c2s_app_data für die Entschlüsselung des Chiffrats verwenden und dabei analog A_24628-* den Header als "Additional Associated Data" mit in die Entschlüsselung einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS Abbruch und Fehlermeldung (vgl. A_24635-*) erfolgen. Die VAU-Instanz MUSS den Request-Counter-Wert speichern, der für die Antwort (A_24632-*) benötigt wird.
[<=]

Hinweis:

"Enforce Replay Protection" und "Enforce Sequence Order" werden in der aktuellen Ausbaustufe von ePA für alle nicht umgesetzt, weil fachlich noch nicht benötigt.

A_24631 -VAU-Protokoll: VAU-Instanz: Verschlüsselungszähler
Eine VAU-Instanz MUSS sicherstellen, dass der Schlüssel K2_s2c_app_data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Eine VAU-Instanz MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.**[<=]**

Verständnishinweis: analog A_24629-*.

A_24632 -VAU-Protokoll: VAU-Instanz: Response erstellen/verschlüsseln
In der VAU-Instanz wird der innere HTTP-Request (Ergebnis aus A_24630-*) fachlich verarbeitet und als Antwort eine innere HTTP-Response erzeugt. Diese ist der Klartext, der jetzt behandelt wird.

Eine VAU-Instanz MUSS analog zu A_24628-* einen Header erzeugen, wobei sie

1. beim Response/Request-Byte den Wert 2 verwenden MUSS,
2. im Request-Counter den gespeicherten Wert aus A_24630-* (Eingang des Requests) verwenden MUSS.

Die anderen Header-Variablen MÜSSEN analog zu A_24628-* festgelegt werden.
Die Konstruktion des IV MUSS analog zu A_24628-* erfolgen, nur dass für den Verschlüsselungszähler der Wert nach A_24631-* (K2_s2_app_data) verwendet werden MUSS.
Mit dem IV und dem Schlüssel K2_s2c_app_data MUSS der Klartext (s. o.) verschlüsselt werden mit AES/GCM, wobei der Header als "Additional Associated Data" bei der Verschlüsselung mit einfließt.

Das so erzeugte erweiterte Chiffrat (vgl. A_24628-*, also Chiffrat inkl. Header) MUSS an den Client im HTTP-Response-Body versendet werden, wobei das Aktensystem den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden MUSS.
[<=]

A_24633 -VAU-Protokoll: VAU-Client: Response entschlüsseln/auswerten
Ein VAU-Client MUSS bei Erhalt einer Antwort gemäß A_24632-* auf einen Request gemäß A_24628-* folgendes prüfen:

1. Ist die Länge der Datenstruktur (erweiterte Chiffrat) mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. sS v. korrekte Umgebung, in der auch der VAU-Client arbeitet.
3. Ist das Request/Response-Byte gleich 2.

4. Hat der Response-Counter den von Client erwarteten Wert.
5. Ist die KeyID bekannt.

Ergibt eine der Prüfungen ein nicht-positives Ergebnis, MUSS der Client die Verarbeitung der Response abbrechen.
Er MUSS das Chiffrat mit dem mit der KeyID verbundenen Schlüssel K2_s2c_app_data mittels AES/GCM entschlüsseln und dabei den Header als "Additional Associated Data" in die Entschlüsselung mit einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS er die Verarbeitung des Chiffrats/Response abbrechen.
[<=]

6.3 OIDC-Authentisierung eines Clients (Nutzer)

Bei den meisten ePA-Clients wird für die Authentisierung des Nutzers der OAuth2/OIDC/PKCE-Authentication-Workflow nach Etablierung der kryptographischen Phase des VAU-Protokolls verwendet.

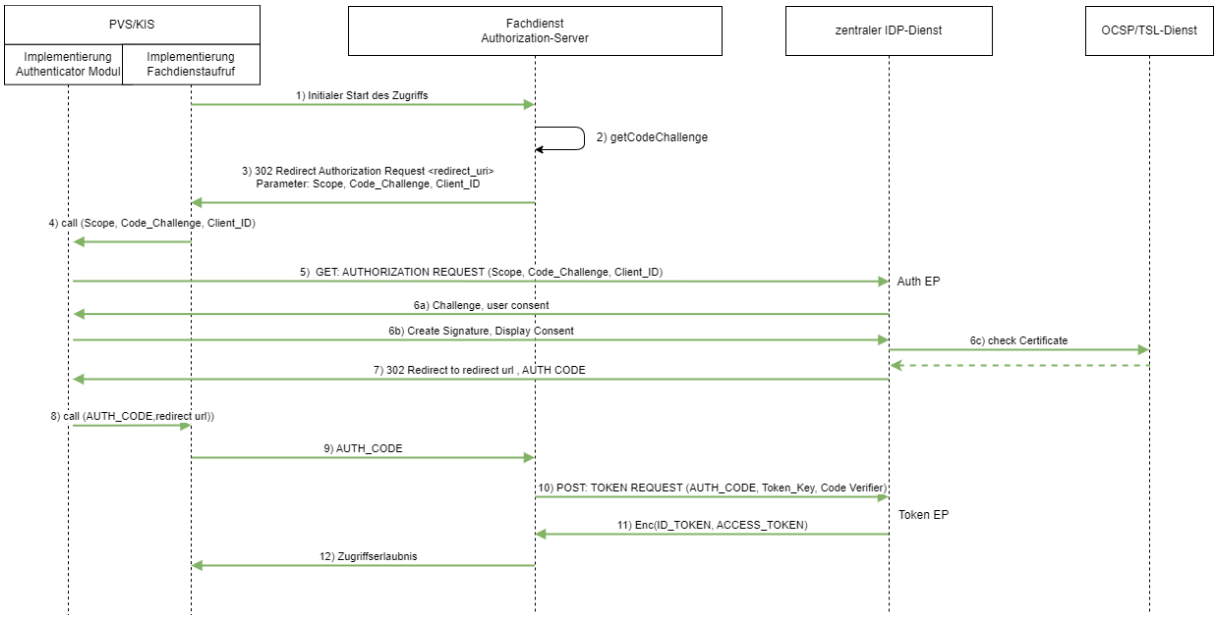

Abbildung 7: OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI

D. h., der rein kryptographische Teil des VAU-Protokolls hat erfolgreich stattgefunden und zwischen ePA-Client und VAU-Instanz konnten zwei symmetrische Schlüssel in einer Form der Multi-Party-Computation zusammen berechnet werden. Eine einseitig authentisierte und vertrauliche Datenverbindung besteht zwischen ePA-Client und VAU-Instanz. Über diese Datenverbindung findet auf einer höheren OSI-Schicht eine Client-Authentisierung statt. Dafür werden vom Client innere HTTP-Requests über das VAU-Protokoll an die VAU-Instanz gesendet. Im OIDC Fall zwei Stück:

1. einmal wird ein Hashwert (die "Codechallenge") per GET bezogen und
2. als zweites wird ein "Authentication Code" per POST vom Client an die VAU-Instanz gesendet.

Mittels des Urspungsbilds der Codechallenge (Zufallswert erzeugt von der VAU-Instanz/ Input-Daten für die Hashwertberechnung) und dem AUTH-Code bezieht die VAU-Instanz ein Identity-Token vom IDP, in dem Identitätsangaben des Clients (KVNR oder Telematik-ID) stehen. Nach Prüfung und Analyse dieses Identity-Tokens kennt die VAU-Instanz die Client-Identität -- eine beidseitig authentisierte Datenverbindung steht ab dann bereit. Bevor dies erfolgt ist, darf eine VAU-Instanz keine zusätzliche Funktionalität (i. S. v. ePA-spezifische APIs) per innerer HTTP-Requests einem noch unauthentisierten Client verfügbar machen.

A_24634 -VAU-Protokoll: VAU-Instanz: notwendige Authentifizierung des VAU-Clients (Nutzers)
Eine VAU-Instanz MUSS sicherstellen, dass, solange der ePA-Nutzer noch nicht authentisiert ist, keine ePA-Funktionalität (ePA-spezifische APIs, die nicht zur weiteren Client-Authentisierung dienen) vom ePA-Client verwendbar sind (siehe Erläuterung vor A_24634-*).**[<=]**

A_25055-02 -VAU-Protokoll: OIDC Authentisierung oberhalb der VAU-Protokoll-Schicht
Eine VAU-Instanz MUSS für eine OIDC/OAuth2/PCKE notwendige HTTP Endpunkte per inneren HTTP-Request/Responses einem Client verfügbar machen (Codechallenge per GET zur Verfügung stellen, Authentication Code per POST vom Client empfangen).

Nach erfolgreichen Authentisierung über OIDC MUSS die VAU-Instanz in den Metadaten der Verbindungsschlüssel (KeyID, K2_*_app_data) den neuen Authentisierungszustand vermerken (Rückwirkung auf A_24634-*).

Nach erfolgreicher Authentisierung über OIDC MUSS die VAU-Instanz ein Nutzerpseudonym (NP) in der Response des inneren POST-Requests für den Authentication -Code als "vau-np" eintragen und so an den Client übertragen, vgl. A_24770-*.
[<=]

A_25143 -VAU-Protokoll: Abfrage aktueller Status der Nutzerauthentisierung
Eine VAU-Instanz MUSS über innere HTTP-Requests über den Pfadnamen /VAU-Status per HTTP-GET wie folgend aufgeführt Informationen über den Status der aktuellen VAU-Verbindung bereitstellen. Als Antwort MUSS eine VAU-Instanz eine JSON-Datenstruktur der folgenden Struktur senden (Reponse-Content-Type 'application/json'):

```
{  "VAU-Type": "epa",
   "VAU-Version" : "Version bspw. in der Form <Hersteller>-1.2.3.4p8",
   "User-Authentication" : "None | TID:<Telematik-ID> | KVNR:<KVNR> |
GID:<Gesundheits-ID>",
   "KeyID": " ... in Hexadezimalform [0-9a-f] (kleinbuchstaben)",
   "Connection-Start": "2024-01-16T10:08:42.123Z"
}
```

Bei "User-Authentication" steht entweder "None" falls noch keine erfolgreiche Nutzerauthentisierung stattgefunden hat, die Telematik-ID falls eine erfolgreiche Nutzerauthentisierung eines Nutzers, der eine Telematik-ID besitzt (bspw. LEI), die KVNR eines erfolgreich authentisierten Versicherten oder (in einer späteren Ausbaustufe) die Gesundheits-ID des erfolgreich authentisierten Nutzers.

In der Datenstruktur KÖNNEN weitere Daten (Key-Value-Paare) von der VAU-Instanz aufgeführt werden. Die maximale Größe der von der VAU-Instanz gesendeten Datenstruktur MUSS kleiner gleich 2 MiB sein.

Die Abfrage von /VAU-Status MUSS als "Aktivität" im Sinne von [gemSpec_Aktensystem_ePAfueralle#A_25006-*] gelten.**[<=]**

Erläuterung: Die Maximalgrößenangabe dient dazu, damit eine Primärsystem-Implementierung (ePA-Client) weiß welche Datengrößen es maximal akzeptieren können muss.

6.4 Authentisierung des E-Rezept-FD als ePA-Client

Bei der Authentisierung des E-Rezept-FD als ePA-Client wird nicht OIDC zur Client-Authentisierung verwendet, sondern folgendes einfaches Challenge/Response-Verfahren.

A_24658-01 -VAU-Protokoll: PKI-basierte Client-Authentisierung
Eine VAU-Instanz MUSS über einen inneren HTTP-Request zwei Operation wie folgt anbieten.

(1) Über operationid = getFreshnessParameter MUSS die VAU-Instanz Daten per HTTP-GET bereitstellen.
Diese Daten sind eine base64-kodierte Zeichenkette. Der Media-Type (Content-Type) MUSS 'application/json'sein.
Die Zeichenkette ist für einen Client opak. Diese Zeichenkette MUSS es einer VAU-Instanz ermöglichen, zu ermitteln, ob und wann ein VAU-HSM oder die Befugnisverifikations-VAU (BV-VAU [gemSpec_Aktensystem_ePAfueralle#3.4 Befugnisverifikations-Modul], das/die mit der VAU-Instanz verbunden ist, diese Zeichenkette erzeugt hat (bspw. Unix-Zeit + ECDSA-Signatur HSM). Es MUSS sichergestellt sein, dass nur das VAU-HSM oder die BV-VAU, das/die mit der VAU-Instanz verbunden ist, den Frischeparameter erzeugt haben kann.

(2) Über operationid = sendAuthorizationRequestBearerTokenMUSS sie die Möglichkeit anbieten, per HTTP-POST ein Authentisierungstoken an die VAU-Instanz zu senden.

Die Authentisierungstoken sind JWT [RFC-7519], deren Body mindestens folgende Elemente enthalten MUSS:

```
{
    "type" : "ePA-Authentisierung über PKI",
    "iat" : ...zeit...,
    "challenge" : Frischeparameter,
    "sub": ...Telematik-ID...
}
```

Im Header des JWT MUSS innerhalb eines x5c-Array das "signierende" Zertifikat enthalten sein. Bei "sub" trägt der Client/Nutzer seine Telematik-ID ein.

Die VAU-Instanz MUSS das vom Client gesendete Authentisierungstoken prüfen:

1. Ist das im Header aufgeführte "signierende" Zertifikat gültig, inkl. OCSP-Prüfung (vgl. OCSP-Response-Caching nach [gemSpec_PKI#A_23225]).
2. Ist die Signatur valide.
3. Stimmt der "sub"-Wert im Body mit der Telematik-ID im "signierenden" Zertifikat überein.
4. Ist die "iat"-Zeit nicht älter als 10 Minuten.
5. Ist die "challenge" vom VAU-HSM oder einer BV-VAU erzeugt worden.
6. Ist die challenge nicht älter als 10 Minuten.

7. Ist der "type" gleich "ePA-Authentisierung über PKI"

Bei Prüfung mit positivem Prüfergebnis MUSS die VAU-Instanz den neuen Authentisierungsstatus (bspw. E-Rezept-FD wurde authentifiziert) in den Metadaten den Verbindungsschlüssel (K2_c2s_app_data, K2_s2c_app_data) vermerken.

Im inneren HTTP-Response-Header MUSS die VAU-Instanz das Nutzerpseudonym (vgl. A_24770-*) übertragen.

Die VAU MUSS die Authentisierung beliebiger Nutzer mit gültigen AUT-Zertifikat und Telematik-ID darin erlauben (also nicht nur eingeschränkt auf den E-Rezept-FD).**[<=]**

Beim Frischeparameter soll sichergestellt werden, dass der Authentisierungsvorgang des Clients nicht zu alt ist. Dies muss auch das VAU-HSM bzw. die BV-VAU prüfen. Das Signaturmaterial, das die Challenge/Frischeparameter für die Prüfung im Aktensystem authentisiert, muss keinen Zusammenhang zu TI-PKI haben, da die Challenge vom Client als opakes Objekt behandelt wird (A_24771-*). D. h., nur das Aktensystem selbst (VAU, VAU-HSM, BV-VAU) selbst prüft die Challenge.

A_24959 -VAU-Protokoll: PKI-basierte Client-Authentisierung (VAU-HSM, BV-VAU)
Ein Aktensystem MUSS sicherstellen, dass bei der Prüfung der PKI-basierten Authentisierung (bspw. durch den E-Rezept-FD) die Prüfung des vom Client signierten JWT mindestens die Prüfschritte aus A_24658-* bei der Prüfung im VAU-HSM oder in der BV-VAU durchgeführt werden. Bei nicht-positiven Prüfergebnis MUSS das VAU-HSM oder die BV-VAU das JWT ablehnen -- die Client-Authentisierung beim VAU-HSM bzw. BV-HSM kann nicht stattfinden.**[<=]**

A_25192-02 -VAU-Protokoll: VAU-Instanz, zusätzliche TID-Prüfung E-Rezept-FD
Ein Aktensystem MUSS sicherstellen, dass in einer VAU-Instanz bei jeder Nutzer-Authentisierung (i. S. v. egal ob über A_24568-* oder A_25055-*) eines ePA-Nutzers, bei der der Nutzer die Telematik-ID "9-E-Rezept-Fachdienst" besitzt, die VAU-Instanz prüft, ob das "authentisierende" EE-AUT-Zertifikat des Nutzers

1. professionOID gleich 1.2.276.0.76.4.258 gemäß [gemSpec_OID#GS-A_4446-*, oid_erp-vau] als Attribut besitzt und
2. aus einer Komponenten-PKI-CA der TI-PKI stammt (vgl. Implementierungshinweis).

Liefern diese Prüfungen kein positives Prüfergebnis, so MUSS die VAU-Instanz die Authentifizierung ablehnen.**[<=]**

Eine E-Rezept-VAU als ePA-Nutzer hat innerhalb eines Aktensystems besonders umfassende Zugriffsrechte. Als Risikomitigation wird bei einer Nutzerauthentisierung in einer ePA-VAU-Instanz überprüft, ob das AUT-Zertifikat der E-Rezept-VAU (als ePA-Nutzer) besondere Eigenschaft besitzt.

Implementierungshinweis:

Sowohl bei A_24568-* als auch bei A_25055-* wird eine auf dem EE-AUT-Zertifikat basierende Signatur vom Nutzer erzeugt. Solch eine Signatur muss eine ePA-VAU-Instanz im Rahmen der Nutzerauthentitfizierung prüfen, was auch die Prüfung des EE-AUT-Zertifikats inkludiert. Die TI-PKI besitzt genau drei Hierarchie-Ebenen: Root-Zertifikat, CA-Zertifikat, EE-Zertifikat. Es ist für die Prüfung A_25192-*#Punkt-2 ausreichend zu prüfen, ob das bestätigende CA-Zertifikat im CommonName mit "GEM.KOMP-CA" (anschließend kommt eine natürliche Zahl) beginnt und ob das CA-Zertifikat im Signatur-Graph Kind einer TI-PKI-Root ist. Beide Bedingungen sind technisch leicht zu prüfen, dennoch empfiehlt es sich innerhalb einer VAU-Instanz das Prüfergebnis zu cachen (Hashwert des AUT-Zertifikats + besondere-E-Rezept-Prüfung-OK).

A_24771 -VAU-Protokoll: E-Rezept als Client
Der E-Rezept-FD MUSS für die Authentisierung bei einer VAU-Instanz nach dem erfolgreichen Durchlaufen der VAU-Protokoll-Handshake-Phase den Mechanismus nach A_24658-* verwenden.

Dabei bezieht er zunächst eine Challenge/Frischeparameter nach A_24658-*. Diese MUSS er als opakes Objekt behandeln (also nicht versuchen, den Inhalt auszuwerten).

Anschließend MUSS er einen signierten JWT nach A_24658-* erzeugen und diese per HTTP-POST im inneren Request eine VAU-Protokoll-Verbindung an die VAU-Instanz senden.**[<=]**

Der E-Rezept-FD kann für die Datenübertragung zum Aktensystem folgendes Vorgehen wählen. Er verbindet sich initial einmal per VAU-Protokoll mit einer VAU-Instanz und verwendet den Mechanismus nach A_24658-* zur Authentisierung des E-Rezept-FD bei der VAU-Instanz. Hat dies erfolgreich stattgefunden, erhält der E-Rezept-FD als "normaler" ePA-Client ein Nutzerpseudonym (vgl. A_24770-*). Nun kann er mit diesem Nutzerpseudonym (A_24757-*) bspw. 50 neue TLS-Verbindungen zum Aktensystem eröffnen. Bezüglich des DoS-Schutzes am Aktensystem: an der IP-Adresse kann ein Aktensystem schon erahnen, dass es sich um den E-Rezept-FD handelt und so diese 50 Verbindungen auf IP-Ebene dem Client erlaubt (Perimeter/Firewall), was es bei anderen IP-Adressen evtl. nicht tut (Rate-Limiting). Innerhalb der neuen TLS-Verbindungen führt der E-Rezept-FD jeweils VAU-Protokoll-Verbindungsaufbauten plus Nutzerauthentisierung (A_24770-*) durch und ein Aktensystem kann "geeignet" (vgl. Abschnitt6.5- Routing auf VAU-Instanzen) die Verbindungsaufbauten auf für das Aktensystem günstige VAU-Instanzen verteilen. Im Beispiel kann der E-Rezept-FD dann über 51 Datenkanäle parallel Rezeptinformationen für verschiedene Versicherte in deren Akten einbringen (Parallelverarbeitung).

6.5 Routing auf VAU-Instanzen

Für ein Aktensystem erleichtert es in bestimmten Konstellationen die Implementierung, wenn schon beim VAU-Protokoll-Verbindungsaufbau es Hinweise darüber gibt, um welche Client-Identität (i. S. v. Nutzer-Identität) es sich handelt. Dann kann eine Routing-Komponente des Aktensystems, die nach den HTTPS-Schnittstellen und vor den VAU-Instanzen liegt, die Requests "geeignet" auf die verschiedenen VAU-Instanzen verteilen. Was "geeignet" hier bedeutet, liegt in der Ausgestaltungshoheit des Aktensystemherstellers.

A_24770-01 -VAU-Protokoll: VAU-Instanz, Nutzerpseudonym erzeugen
Nach erfolgreicher Nutzer-Authentisierung des Clients MUSS die VAU-Instanz ein Nutzerpseudonym (NP) erzeugen. Dieses NP MUSS in der Response des inneren HTTP-Requests als "vau-np" eingetragen und so an den Client übertragen werden.**[<=]**

Beispiel

VAU-NP: 1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

In A_24757-* ist definiert, dass ein VAU-Client solch ein Nutzerpseudonym bei seinen Requests für die Nachricht 1 im Handshake (VAU-Protokoll Verbindungsaufbau) im Request-Header im (äußeren) HTTP-Request aufführen muss und dass er auch eine Aktualisierung des Nutzerpseudonym nach erfolgreicher Nutzer-Authentisierung erkennen und sein lokal gespeichertes Nutzerpseudonym aktualisieren muss.

A_25150 -VAU-Protokoll: VAU-Instanz, Nutzerpseudonym regelmäßig wechseln

Ein Aktensystem MUSS sicherstellen, dass ein Nutzerpseudonym nach maximal 2 Monaten gewechselt wird, i. S. v. nach erfolgreicher Authentisierung von Entität A wird NP_1 an den Client zurückgegeben (A_24770-*) nach etwas mehr als zwei Monaten authentisiert sich A erneuert und erhält NP_2, dann MUSS NP_1 ungleich NP_2 sein. Die bloße Kenntnis von NP_1 und NP_2 DARF NICHT erkennen lassen, dass diese beide Pseudonyme zur gleichen Entität gehören oder gar zu A gehören.
(vgl. Implementierungshinweis).**[<=]**

Implementierungshinweis zu A_25150:

Eine mögliche Herangehensweise für A_25150-* ist, dass alle VAU-Instanzen Zugriff auf einen geheimen Pseudonymisierungsschlüssel für die Pseudonymerstellung haben. Dieser Schlüssel liegt nicht notwendiger Weise ausschließlich im HSM, sondern kann beim Start der VAU-Instanz in diese sicher geladen werden und im sicheren Speicher der VAU-Instanz verbleiben. Der Schlüssel wird alle zwei Monate Aktensystemweit gewechselt. Nach erfolgreicher Authentisierung wird mit dem Entitäts-ID (bspw. KVNR) ein HMAC(K=<Schlüssel>, text=<KVNR>) erzeugt. Dieser berechnete HMAC-Wert ist dann das NP.

Routingentscheidung

Es gibt wenige Fälle, in denen aufgrund von fehlenden Informationen eine Routing-Entscheidung im Aktensystem nicht optimal getroffen werden konnte. In diesen Fällen darf ein Aktensystem das Neuverbinden des Client einfordern. Ein solcher Fall ist bspw., wenn eine Arztpraxis auf einem PVS-Rechner eine Akte aktuell in Verwendung hat. Und auf einem "neuen" PVS-Rechner/Tablet etc., der noch nie mit dem Aktensystem in Kontakt stand -- also lokal noch kein Nutzerpseudonym gespeichert hat (vgl. A_24757-*), eine Akte verwenden möchte. Auf dem neuen PVS-Rechner würde dann ein VAU-Protokoll-Verbindungsaufbau und Nutzer-Authentisierung stattfinden. Mit hoher Wahrscheinlichkeit landet diese Verbindung in einer anderen VAU-Instanz. Dies würde dann erkannt und A_24772-* kann vom Aktensystem ausgelöst werden. Der "neue" PVS-Rechner würde sich neu verbinden, diesmal mit ihm bekanntem Nutzerpseudonym. Das Aktensystem kann dann eine für sich günstigere Routing-Entscheidung treffen. Das Aktensystem ist frei in der Entscheidung, ob und bei welchen Konstellationen es von dieser Möglichkeit Gebrauch macht.

A_24772 -VAU-Protokoll: Restart für Änderung der Routingentscheidung
Ein ePA-Aktensystem KANN nach erfolgreicher Authentisierung des ePA-Clients (i. S. v. Nutzer-Authentisierung) folgende Retry-Nachricht senden:

```
{
    "MessageType" : "Restart",
    "KeyID" : ... KeyID ...
}
```

Genauer: die Nutzer-Authentisierung hat erfolgreich stattgefunden, und bei dem nächsten Request des Clients KANN ein Aktensystem die Restart-Nachricht als Antwort (Response) auf den Request des Clients senden.

Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert und die erzeugte Kodierung unter Verwendung des Media-Type 'application/cbor' (HTTP Content-Type) an den ePA-Client im äußeren HTTP zurückgesendet werden. **[<=]**

A_24773 -VAU-Protokoll: Clients: Neustart/Wiederholung des Verbindungsaufbaus
Ein VAU-Client MUSS Restart-Nachrichten nach A_24772-* verarbeiten können, und bei Erhalt einer solchen Nachricht die aktuelle VAU-Protokoll-Verbindung, die mit der aufgeführten KeyID verbunden ist, beenden, indem es die mit der KeyID verbundenen symmetrischen Schlüssel sicher löscht. Anschließend MUSS es einen neuen VAU-Protoll-

Verbindungsaufbau durchlaufen (inkl. Nutzer-Authentisierung gegenüber der VAU-Instanz).**[<=]**

6.6 Fehlersignalisierung

A_24635 -VAU-Protokoll: VAU-Instanz und Aktensystem: Erzeugung von Fehlermeldungen
Eine VAU-Instanz MUSS das Auftreten von Fehlern bei der Abarbeitung des VAU-Protokolls an das "äußere" Aktensystem melden. Daraufhin MUSS das Aktensystem an der HTTPS-Schnittstelle den Client-Request in folgender Weise mit einer Fehlermeldung beantworten:

Als HTTP-Response-Code MUSS das Aktensystem einen HTTP-Fehlercode (i. S. v. eben nicht 200) in der äußeren HTTP-Response verwenden (siehe folgende Tabelle). Es MUSS weiterhin eine Datenstruktur der folgenden Art erzeugen:

```
{
            "MessageType" : "Error",
            "ErrorCode" : ...natürliche-Zahl...,
            "ErrorMessage" : "... Menschenlesbarer Text bzw.
Fehlerursache ..."
}
```

Diese Datenstruktur MUSS es per CBOR [RFC-CBOR] serialisieren und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) im äußeren Response-Body aufführen.

Ein Aktensystem KANN weitere selbst definierte Variablen-Werte-Paare in der o. g. Datenstruktur aufführen.

Folgende Fehler-Meldungen MUSS es mindestens geben:

HTTP-Error	ErrorCode	ErrorMessage / Beschreibung
400	1	"Decoding Error" Fehler in einer Kodierung bspw. der CBOR-Kodierung
400	2	"Missing Parameters" notwendige Datenfelder bspw. in der Handshake fehlen
403	3	"GCM returns FAIL" Eine AES/GCM-Entschlüsselung ergibt das Symbol "FAIL".
403	4	"PU/nonPU Failure"
403	5	"Transscript Error" Im Handshake wird Ungleichheit zwischen den beiden Transskript-Hashwerten festgestellt.

400	6	"bad format: extended ciphertext" Die erste Sanity-Prüfung des erweiterten Chiffrat bspw. bei A_24630 schlägt fehl.
403	7	"is not a request" A_24630 (Prüfschritt 3)
403	8	"unknow KeyID"
403	9	"unknown CID"

[<=]

A_24767 -VAU-Protokoll, Fehlerverarbeitung im VAU-Client
Ein VAU-Client MUSS die Fehlermeldungen nach A_24635-* verarbeiten können.**[<=]**

6.7 Tracing in Nichtproduktivumgebungen

Für die Fehlersuche in Nichtproduktivumgebungen -- insbesondere bei IOP-Problemen zwischen Produkten verschiedener Hersteller in einer fortgeschrittenen Entwicklungsphase -- hat es sich als notwendig erwiesen, dass ein Fehlersuchender den Klartext der Kommunikation zwischen ePA-Client und VAU-Instanz mitlesen kann, vgl. [gemSpec_Aktensystem_ePAfueralle#3.18.4 Tracing in Nichtproduktivumgebungen].

Bei ePA für alle unterscheidet sich der Mechanismus hier im Vergleich zu ePA 2.x ein wenig: Es werden nicht für den Client feste ECDH-Schlüssel für die Schlüsselaushandlung im VAU-Protokoll in Nichtproduktivumgebungen vorgegeben, sondern der Client wird verpflichtet, die ausgehandelten symmetrischen Sitzungsschlüssel (K2_c2s_app_data, K2_s2c_app_data) im äußeren HTTP-Request im Request-Header aufzuführen. Die Motivation dafür ist, dass es bei einigen Krypto-Bibliotheken ggf. schwieriger sein kann, den Zufall, der bei der KEM-Encapsulation-Operation einfließt, fest vorzugeben. Es erleichtert also die Implementierbarkeit. Am Chiffrat kann eine VAU-Instanz sicher feststellen, für welche Umgebung ein Client ein Chiffrat erzeugt hat. Ein VAU-Instanz lehnt in der Produktivumgebung dann solche Nichtproduktivumgebungs-Chiffrate ab.

A_24477 -VAU-Client, Nichtproduktivumgebung, Offenlegung von symmetrischen Verbindungsschlüsseln
Ein VAU-Client in einer Nichtproduktivumgebung MUSS nach einer erfolgreichen Schlüsselaushandlung mit dem VAU-Protokoll die ausgehandelten symmetrischen Schlüssel K2_c2s_app_data und K2_s2c_app_data base64-kodiert innerhalb einer HTTP-Request-Header-Variable "VAU-nonPU-Tracing" bei jedem äußeren HTTP-Request aufführen. Dabei sind die beiden Base64-kodierten Schlüsselwerte durch Leerzeichen zu trennen. Beispiel:

```
 VAU-nonPU-Tracing: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=
```

Als erstes ist der Wert von Schlüssel K2_c2s_app_data aufzuführen, als zweiter Wert ist der Wert von Schlüssel K2_s2c_app_data aufzuführen.**[<=]**

Hinweis: Ebenfalls muss ein VAU-Client nach A_24628-* im Chiffrat kennzeichnen, ob es eine Nachricht in einer Nichtproduktivumgebung ist oder nicht.

A_24478 -VAU-Instanz, Nichtproduktivumgebung, Ablehnung von nonPU-Chiffraten in der PU
Ein Aktensystem in der Produktivumgebung MUSS äußere HTTPS-Request, die im Request-Header eine Variable "VAU-nonPU-Tracing" enthalten, mit einer HTTP-Fehlermeldung 403-Forbidden beantworten. Das Aktensystem MUSS sicherstellen, dass der innere Request (Chiffrat im Request-Body des äußeren Requests) keine VAU-Instanz erreicht.
Eine VAU-Instanz in der Produktivumgebung, die am Chiffrat am nonPU/PU-Byte erkennt (vgl. A_24628-*), dass es sich um ein nonPU-Chiffrat handelt, MUSS den Request mit einer Fehlermeldung an die äußere Schicht des Aktensystems (Webschnittstelle) beantworten. Die VAU-Instanz MUSS sicherstellen, dass das Chiffrat nicht entschlüsselt und das Chiffrat in der VAU-Instanz sicher gelöscht wird.**[<=]**

6.8 Zufallsquelle für Clients

A_24921 -ePA-Aktensystem, Random-Operation
Ein Aktensystem MUSS an seiner äußeren HTTPS-Schnittstellen eine Random-Operation, wie in A_21215-* definiert, zur Verfügung stellen.**[<=]**

Erläuterung: Auf einem Linux-Server ist es ausreichend, Daten aus /dev/urandom zu verwenden.

7 ZETA/ASL (VAU-Protokoll)

Bei allen Anwendungen der TI, die personenbezogenen medizinische Daten verarbeiten, gibt es das Grundprinzip der Mehrschichtigen Sicherheit (bspw. E-Rezept, ePA, KIM). Es gibt eine Sicherungsschicht (meistens TLS, vgl. Abschnitt 3.3.2 TLS-Verbindungen) und oberhalb dieser Sicherungsschicht gibt es eine weitere Sicherungsschicht, die eine Unabhängigkeit von möglichen Schwachstellen auf der unteren Schicht erzeugt.

Bei TLS gab es in der Vergangenheit:

1. Schwachstellen im Protokoll selbst (BEAST (Browser Exploit Against SSL/TLS), CRIME (Compression Ratio Info-leak Made Easy), DROWN (Decrypting RSA with Obsolete and Weakened eNcryption), POODLE (Padding Oracle On Downgraded Legacy Encryption), ROBOT (Return Of Bleichenbacher`s Oracle Threat), Ticketbleed etc.)
2. schwerwiegende Implementierungsfehler (Heartbleed) in oft verwendeten TLS-Implementierungen
3. erfolgreiche Angriffe im Internet-CA-Vertrauensraum (DigiNotar) (Hinweis: die Grundlage für die TLS-Verwendung bei Zero-Trust ist der Internet-CA-Vertrauensraum).

Unter anderen der Einsatz der hier definierten zweiten Sicherungsschicht (Additional Security Layer (ASL)) hilft derartige auch zukünftig zu erwartenden Arten von Schwachstellen aus (1) bis (3) abzufangen. Dies ermöglicht die unmittelbare Weiterführung des Betriebs (Betriebssicherheit) der TI-Anwendung auch bei Bekanntwerden von Schwachstellen auf einer Sicherungsschicht - man kann die Schwachstellen schließen, ohne zwischenzeitlich den Betrieb einstellen zu müssen.

Das in diesem Abschnitt definierte kryptographische Protokoll ist bis auf eine Aktualisierung das VAU-Protokoll von ePA für alle. Aktualisierung: nach Abschluss der Standardisierung von FIPS 203 [FIPS-203] wird für das zu verwendetes PQC-KEM-Verfahren nicht mehr auf auf Kyber Version 3.02 [IETF-Kyber] verwiesen sondern eben auf den nun finalisierten FIPS 203 (ML-KEM-768).

Die Verwendung des VAU-NP (vgl. A_24757-*) macht im Kontext Zero-Trust keinen Sinn mehr und wird deshalb bei ZETA/ASL nicht mehr verwendet.

A_26920 -ZETA/ASL mindestens HTTP-Version 1.1
Ein ZETA/ASL-Server MUSS an seinen HTTPS-Schnittstellen, i. S. v. Schnittstellen, die ein ZETA/ASL-Client anspricht, mindestens HTTP Version 1.1 unterstützen.**[<=]**

7.1 Verbindungsaufbau/Schlüsselaushandlung

A_26921 -ZETA/ASL: ZETA/ASL-Schlüssel für die ASL-Protokoll-Schlüsselaushandlung
Ein ZETA/ASL-Server MUSS sicherstellen, dass

1. es eine Signatur-Identität (AUT) aus der Komponenten-PKI der TI gibt, die technisch sichergestellt ausschließlich nur von ZETA/ASL-Instanzen verwendbar ist.
2. es semi-statische Schlüsselpaare für ECDH (auf Basis Kurve P-256 [FIPS-186-5]) und ML-KEM-768 [FIPS-203] gibt, deren private Schlüssel, technisch sichergestellt, ausschließlich von ZETA/ASL-Server verwendbar sind.

3. die privaten Schlüssel (aus 2.) in einer ZETA/ASL-Instanz erzeugt und verarbeitet werden,
4. die semi-statischen Schlüssel eine maximale Lebensdauer von einem Monat besitzen (Hinweis: die Forward-Secrecy hängt nicht vom Wechselintervall ab, innerhalb eines Verbindungsaufbaus und der Schlüsselaushandlung dabei fließen ephemere Schlüsselwerte von Client und Server ein).
5. die semi-statischen Schlüssel in einer über die Signatur-Identität authentisierten folgenden Datenstruktur aufgeführt werden.

Struktur der signierten semi-statischen öffentlichen ASL-Schlüssel

```
ASL_Keys = {
          "ECDH_PK" :
            { "crv" : "P-256",
              "x" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),
              "y" : Binärwert-x-Koordinate-32-Byte-big-endian (256 Bit),
            },
          "ML-KEM-768_PK" : Binärwert-öffentlicher-Schlüssel-nach-keygen-
FIPS-203-ML-KEM-768,
          "iat" : Erzeugungszeits-Sekunden-Since-Epoch (integer),
          "exp" : Nicht-mehr-Verwendbar-nach (integer),
          "comment" : "Erzeugt bei ASL-Instanz xyz, Meta-Info abcd"
}
```

In "comment" KÖNNEN beliebige Text-Daten aufgeführt werden. Es können weitere Attribute hinzugeführt werden. Ein Client MUSS ihm unbekannte Attribute ignorieren. Diese Struktur wird mittels CBOR [RFC-CBOR] binär kodiert und im Folgenden ASL_Keys_encoded genannt.

Diese binäre Byte-Folge wird in folgende Datenstruktur eingebracht

```
{
 "signed_pub_keys" : ASL_Keys_encoded,
 "signature-ES256" : ECDSA-Signatur-SHA-256-analog-RFC-7515 (R||S => 64
Byte) binär,
 "cert_hash"       : SHA-256-Wert des "signierenden" AUT-ASL-Zertifikats,
 "cdv"             : Cert-Data-Version (natürliche Zahl, beginnend mit 1,
vgl. A_26922-*),
 "ocsp_response"   : OCSP-Response-für-das-ASL-Signaturzertifikat-nicht-
älter-als-24-Stunden-DER-Kodierung
}
```

Diese Datenstruktur wird mittels CBOR binär kodiert (serialisiert). Das Ergebnis der Kodierung wird "signierte öffentliche ZETA/ASL-Schlüssel" (Plural) genannt.

[<=]

A_26922 -ZETA/ASL: Verfügbarmachung des AUT-ZETA/ASL-Zertifikats plus Prüfkette

Ein ZETA/ASL-Server MUSS über an seinen Webschnittstellen mittels eines äußeren HTTPS-Requests per HTTP-GET unter dem Pfadnamen /CertData.<SHA-256-Hashwert-Hex-[0-9a-f]>-Versionszahl (a-f kleingeschrieben) folgende Datenstruktur zur Verfügung stellen:

```
{
```

```
   "cert": DER-kodiertes-AUT-ZETA/ASL-Zertifikat,
   "ca"  : DER-kodiertes-Komponenten-PKI-CA-aus-dem-"cert"-kommt,
   "rca_chain" : [Cross-Zertifikat-1, ..., Cross-Zertifikat-n],
}
```

Die Versionszahl MUSS eine natürliche Zahl sein, beginnend mit 1, die es trotz A_26923-* erlaubt, Fehler in den Daten zu korrigieren (siehe Erläuterung nach A_26922-*).

Diese Datenstruktur MUSS per CBOR [RFC-CBOR] serialisiert/kodiert werden und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) an den ZETA/ASL-Client als Response auf den GET-Request gesendet werden.

In "rca_chain" MÜSSEN alle Cross-Zertifikate in chronologischer Ordnung von RCA7 ausgehend aufgeführt werden, bis die Root-Schlüssel (Cross-Zertifikat) erreicht werden, mit denen das "ca"-Zertifikat bestätigt (signiert) wurde; d. h., sozusagen eine einfach verkettete Liste von Cross-Zertifikaten chronologisch aufsteigend.**[<=]**

A_26923 -ZETA/ASL: ZETA/ASL-Client Prüfbasis Zertifikatsprüfung
Ein ZETA/ASL-Client MUSS als Prüfbasis eine Root-Version (X.509-Root-TI-Zeritifikat), die mindestens 2 Jahre alt ist, verwenden oder die TSL. Der ZETA/ASL-Client MUSS die Operation nach A_26922-* verwenden, um die für die Zertifikatsprüfung von A_26921-* notwendigen Zertifikate zu beziehen. Die bezogenen Zertifikatsdaten MUSS der ZETA/ASL-Client lokal (zeitlich unbegrenzt) vorhalten (Caching). Bei Erhalt einer Nachricht 2 (A_26935-*) MUSS er zunächst prüfen, ob er die für die Zertifikatsprüfung notwendigen Zertifikate im lokalen Cache vorrätig hat, und falls ja MUSS er diese verwenden.**[<=]**

A_26932 -ZETA/ASL: ZETA/ASL-Client, Nachricht 1
Ein ZETA/ASL-Client MUSS für die Erzeugung folgende Schritte durchlaufen.
Ein ZETA/ASL-Client MUSS

1. ein ECC-Schlüsselpaar auf der Kurve P-256 [FIPS-186-5] erzeugen.
2. ein ML-KEM-768-Schlüsselpaar [FIPS-203] erzeugen.

Anschließend MUSS er die öffentlichen Schlüssel in folgende Datenstruktur überführen

```
{
    "MessageType" : "M1",
    "ECDH_PK"    :  { "crv" : "P-256",
                      "x" : analog A_26921-*,
                      "y" : analog A_26921-* },
    "ML-KEM-768_PK" : analog zu A_26921-*
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] in eine Bytefolge kodieren (serialisieren).
Diese Bytefolge ist die Nachricht 1.

Diese Nachricht 1 MUSS der ZETA/ASL-Client an die HTTPS-Schnittstelle des Fachdienstes per POST auf den Pfad /ASL senden, wobei er den Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) verwendet und die Nachricht 1 im Request-Body aufführt.
[<=]

A_26933 -ZETA/ASL-Protokoll: ZETA/ASL-Server: Erhalt von Nachricht 1
Eine ZETA/ASL-Server MUSS die für Nachricht 1 erzeugten Datenobjekte nach A_26932-* verarbeiten können.**[<=]**

A_26934 -ZETA/ASL-Protokoll: AES/GCM-Verschlüsselung im Handshake

Ein ZETA/ASL-Server und -Client verschlüsseln im Rahmen des Handshakes des ZETA/ASL-Protokolls verschiedene Nachrichten-Teile mittels AES/GCM. Dabei MÜSSEN sie pro Verschlüsselung den IV jeweils zufällig als 96-Bit-Wert erzeugen. Der Authentication-Tag MUSS 128 Bit lang sein. Das Ergebnis der Verschlüsselung MUSS dann in der folgenden Kodierung aufgeführt werden:
IV || eigentliche AES-GCM-Chiffrat || 128-Bit langer Authentication-Tag.hb**[<=]**

A_26935 -ZETA/ASL-Protokoll: ZETA/ASL-Server: Nachricht 2
Ein ZETA/ASL-Server MUSS die beiden öffentlichen Schlüssel aus Nachricht 1 (vgl. A_26932-*) prüfen (korrekte ECC-Kurve ("crv":"P-256"), öffentlicher Punkt liegt auf der Kurve P-256 [FIPS-186-5], der öffentliche ML-KEM-768-Schlüssel ist valide [FIPS-203]). Er MUSS für ECDH und ML-KEM-768 jeweils die KEM-Encapsulate-Funktion verwenden und erhält dabei zwei Geheimnisse (ss_e_ecdh, ss_e_mlkem768) und zwei Ciphertexte (ECDH_ct, ML-KEM-768_ct).
Er MUSS für die beiden Geheimnisse zusammenfügen: ss_e = ss_e_ecdh || ss_e_mlkem768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = '' (leere Zeichenkette)), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1_c2s und die letzten 32 Byte heißen K1_s2c.

Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_26934-*) MÜSSEN die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A_26921-*) verschlüsselt werden. Das Ergebnis (vgl. A_26934-*) heißt aed_ciphertext_msg_2.

Er MUSS folgende Datenstruktur erzeugen:

```
{
    "MessageType" : "M2",
    "ECDH_ct"     :  ... analog ECDH_PK aus A_26932-* ...,
    "ML-KEM-768_ct" : ML-KEM-768-Ciphertext analog [FIPS-203],
    "AEAD_ct"     : aead_ciphertext_msg_2
}
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 2.

Er MUSS eine ID erzeugen, kodiert aus der Zeichenmenge
A-Za-z0-9-/
die maximal 200 Zeichen lang ist. Die ID MUSS mit "/" (Slash) beginnen und MUSS ein gültiger URL-Pfadname sein. Diese ID MUSS es ZETA/ASL-Server ermöglichen, den aktuellen Handshake entsprechend zuzuordnen bei Eintreffen der Nachricht-3 des ZETA/ASL-Clients, der diese ID mitsendet. Der ZETA/ASL-Server kann die Struktur der ID selbst definieren. Ein ZETA/ASL-Client MUSS die ID als opake Zeichenkette behandeln.
Er MUSS die Nachricht 2 inkl. ID an die HTTPS-Schnittstelle des Fachdienstes übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 1 eingetroffen ist, die Nachricht 2 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "ZETA-ASL-CID" die ID aufgeführt werden.**[<=]**

Hinweis: Bei der KEM-Encapsulate-Funktion fließt Zufall aus dem System ein. Für den ZETA/ASL-Kommunikationspartner gelten die Vorgaben aus Abschnitt 2.4 (Güte der Zufallserzeugung, Zuweisung über den Produkttypsteckbrief).

Beispiele für die ID aus A_26935-*:

- /ZT-ASL/
  9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08/1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

- /ASL/1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014

A_26936 -ZETA/ASL-Protokoll: ZETA/ASL-Client: Erhalt von Nachricht 2
Ein ZETA/ASL-Client MUSS die für Nachricht 2 erzeugten Datenobjekte nach A_26935-* verarbeiten können.
Er MUSS prüfen, ob

1. im HTTP-Response-Header die Variable "ZETA-ASL-CID" enthalten ist, falls nicht Abbruch.
2. der Wert der Variable eine Bytefolge ist, dessen Länge maximal 200 Byte lang ist und die nur die Zeichen
   A-Za-z0-9-/
   enthält und mit "/" (Slash) beginnt. Falls nicht Abbruch.

Er MUSS den Wert als Pfad für das Versenden der Nachricht 3 (vgl. A_26937) und aller weiteren Nachrichten - also auch nach dem erfolgreichen Handshake - im Kontext dieser Verbindung verwenden.**[<=]**

A_26937 -ZETA/ASL-Protokoll: ZETA/ASL-Client: Nachricht 3
Ein ZETA/ASL-Client MUSS aus der Nachricht 2 (vgl. A_26936-*) des ZETA/ASL-Servers mittels der Ciphertexte ECDH_ct und ML-KEM-768_ct und den privaten ephemeren Client-Schlüsseln aus A_26932-* (Nachricht 1) und der jeweiligen KEM-Decapsulation-Funktionen zwei Geheimnisse berechnen: ss_e_ecdh und ss_e_mlkem768. Er MUSS die beiden Geheimnisse zusammenfügen: ss_e = ss_e_ecdh || ss_e_ml_kem_768 und das Ergebnis mittels der HKDF [RFC-5869] auf Basis von SHA-256 verwenden (info = '' (leere Zeichenkette), um 64 Byte abzuleiten. Die ersten 32 Byte (=256 Bit) heißen K1_c2s und die letzten 32 Byte heißen K1_s2c.

Mit dem Schlüssel K1_s2c mittels AES/CGM (vgl. A_26934-*) MUSS der Ciphertext "AEAD_ct" entschlüsselt werden: die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A_26921-*) werden so als Klartext erhalten. Diese ZETA/ASL-Schlüssel MUSS er nach A_26938-* prüfen. Mittels der öffentlichen Schlüssel (ECDH_PK, ML-KEM-768_PK) MUSS er jeweils die KEM-Encapsulation-Funktion ausführen und er erhält zwei Geheimnisse: ss_s_ecdh und ss_s_mlkem768. Diese führt er zusammen: ss_s = ss_s_ecdh || ss_s_mlkem768. Weiter berechnet er ss = ss_e || ss_s.
Dieses Geheimnis verwendet die HKDF [RFC-5859] auf Basis von SHA-256 (info = '' (leere Zeichenkette)), um 160 Byte (=5 * 32 Byte) abzuleiten. Er MUSS diese 160 Byte in 32 Byte-Blöcke (von offset 0 bis zum Ende) auf folgende fünf Variablen (vier Schlüssel + eine KeyID) verteilen:

- K2_c2s_key_confirmation,
- K2_c2s_app_data,
- K2_s2c_key_confirmation,
- K2_s2c_app_data

und

- KeyID (nicht vertraulich).

Diese ersten vier sind vertrauliche Schlüsselwerte für AES/GCM. Die KeyID wird nach dem Handshake als eindeutige ID für die K2*_app_data Schlüssel dienen.

Er MUSS eine Datenstruktur wie folgt erzeugen:

```
{
    "ECDH_ct"    : client_kem_result_2["ECDH_ct"],
```

```
    "ML-KEM-768_ct" : client_kem_result_2["ML-KEM-768_ct"],
    "ERP" : False,
    "ESO" : False
}
```

ERP steht für "Enforce Replay Protection" und ESO steht für "Enforce Sequence Order". Innerhalb der Spezifikation heißt diese Datenstruktur Nachricht_3_inner_Layer. Diese Datenstruktur MUSS er per CBOR [RFC-CBOR] serialisieren/kodieren. Diese Serialisierung MUSS er mittels K1_c2s verschlüsseln (vgl. A_26934-*) (= "ciphertext_msg_3).
Er MUSS die komplette Nachricht-1 (CBOR-Kodierung), die Nachricht-2 und ciphertext_msg_3 konkatenieren (= Transskript des Client) und davon den SHA-256-Hashwert berechnen. Diesen Hashwert MUSS er mittels K2_c2s_key_confirmation verschlüsseln (vgl. A_26934-*), das Chiffrat sei
als aead_ciphertext_msg_3_key_confirmation hier bezeichnet.

Dann MUSS er folgende Datenstruktur erzeugen:

```
{
     "MessageType" : "M3",
     "AEAD_ct"     : ciphertext_msg_3,
     "AEAD_ct_key_confirmation" : aead_ciphertext_msg_3_key_confirmation
}
```

Diese Datenstruktur MUSS er per CBOR serialisieren, das Ergebnis ist Nachricht 3.
Er MUSS die Nachricht 3 per äußeren HTTP-Request an das Aktensystem senden und dabei den Wert der ZETA/ASL-CID (vgl. A_26936) als URL-Pfadnamen verwenden, unter Verwendung der HTTP-POST-Methode.
[<=]

A_26938 -ZETA/ASL-Protokoll: ZETA/ASL-Client: Prüfung der "signierten öffentlichen ZETA/ASL-Schlüssel"
Ein ZETA/ASL-Client MUSS die "signierten öffentlichen ZETA/ASL-Schlüssel" (vgl. A_26921-*) bei der Verarbeitung von Nachricht 3 (vgl. A_26937-*) wie folgt prüfen. Erhält er bei einem der folgenden Prüfpunkte kein positives Prüfergebnis, so MUSS er die Verarbeitung (Handshake) abbrechen.

1. Prüfung des TI-Zertifikats, das den Hashwert aus dem "cert_hash"-Datenfeld besitzt (Bezug des Zertifikats vgl. A_26922-*), u. a. unter der Verwendung der OCSP-Response aus "ocsp_response" für die Prüfung des Sperrstatus (Prüfung ob "good"). Die OCSP-Response darf dabei nicht älter als 24 Stunden sein.
2. Das ZT-Server/TI-Zertifikat MUSS zeitlich gültig sein. Es MUSS kryptographisch in einer Zertifikats-/Signaturprüfungskette rückführbar auf eine X.509-Root-Version der TI-PKI sein.
3. Das TI-Zertifikat MUSS aus der Komponenten-PKI der TI stammt (vgl. Implementierungshinweis A_25192-*#Punkt-2) und die vom Client gewünschte Rollen-OID (bspw. "oid_epa_vau") besitzt.
4. Die Signatur im "signature-ES256"-Datenfeld MUSS eine valide Signatur für die Daten im Datenfeld "signed_pub_keys" (Signaturprüfung ergibt "valid/accept") sein, unter Verwendung des öffentlichen Signatur-Schlüssels aus dem ZETA/ASL-TI-Zertifikats bei der Signaturprüfung.
5. Der ECC-Schlüssel in signed_pub_keys (vgl. Erzeugung bei A_26921-*) MUSS ein gültiger Punkt der Kurve P-256 [FIPS-186-5] sein. Der öffentliche Schlüssel in "ML-KEM-768_PK"-Datenfeld MUSS ein gültiger ML-KEM-768-Schlüssel [FIPS-203] sein.

6. Die Zeit in "signed_pub_keys.exp" MUSS größer als die aktuelle Systemzeit (Seconds since epoch) sein, d. h. die beiden Schlüssel (ECDH_PK und ML-KEM-768_PK) sind noch zeitlich gültig.

[<=]

A_26939 -ZETA/ASL-Protokoll: ZETA/ASL-Server: Erhalt von Nachricht 3
Ein ZETA/ASL-Server MUSS die für Nachricht 3 erzeugten Datenobjekte nach A_26937-* verarbeiten können.**[<=]**

A_26940 -ZETA/ASL-Protokoll: ZETA/ASL-Server: Nachricht 4
Ein ZETA/ASL-Server MUSS bei Erhalt der Nachricht 3 das Chiffrat AEAD_ct mittels des Schlüssels K1_c2s entschlüsseln. Schlägt dies fehl, MUSS er den Verbindungsaufbau mit einem Fehler (vgl. A_26924-*) abbrechen. Er MUSS analog wie der Client (vgl. A_26937-*) die Schlüsselerzeugung der folgenden Schlüssel durchführen, nur dass er dafür die KEM-Decapsulation-Methode verwendet:

- K2_c2s_key_confirmation,
- K2_c2s_app_data,
- K2_s2c_key_confirmation,
- K2_s2c_app_data

und

- KeyID (nicht vertraulich).

Er MUSS analog zu A_26937-* das Transskript des Clients und dessen SHA-256-Wert berechnen. Er MUSS mittels K2_c2s_key_confirmation das Chiffrat "AEAD_ct_key_confirmation" aus Nachricht 3 entschlüsseln und prüfen, ob der von ihm (= ZETA/ASL-Server) berechnete Transskript-Client-Hashwert mit dem entschlüsselten Klartext übereinstimmt. Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen berechnete Transskript-Client-Hashwert und Klartext festgestellt, MUSS er den Handshake mit einem Fehler abbrechen (vgl. A_26924-*).
Er MUSS den Transskript des ZETA/ASL-Servers als die Konketation von Nachricht1 || Nachricht 2 || Nachricht 3 berechnen, davon den SHA-256-Hashwert berechnen und diesen mittels K2_s2c_key_confirmation verschlüsseln (vgl. A_26934-*), und erhält ein Chiffrat genannt "AEAD_ct_key_confirmation".
Er MUSS die folgende Datenstruktur erzeugen:

```
{
     "MessageType" : "M4",
     "AEAD_ct_key_confirmation" : Chiffrat-AEAD_ct_key_confirmation
 }
```

Diese Datenstruktur MUSS er mittels CBOR [RFC-CBOR] serialisieren. Diese Bytefolge ist Nachricht 4.

Er MUSS die Nachricht 4 inkl. ID (vgl. A_26935-*) an die HTTPS-Schnittstelle des Aktensystem übergeben. Das Aktensystem (bzw. die äußere HTTPS-Schnittstelle) MUSS als Antwort auf den HTTPS-Request, über den die Nachricht 3 eingetroffen ist, die Nachricht 4 als Antwort im Response-Body senden. Der zu verwendende Mime-Type MUSS "application/cbor" (HTTP Content-Type) für die Response sein. Im Response-Header MUSS mit der HTTP-Header-Variable "ZETA/ASL-CID" die ID aufgeführt werden.**[<=]**

Erläuterung:

Die Aufführung von "ZETA/ASL-CID" im Response-Header ist eigentlich nicht mehr absolut notwendig, da der Client diese schon bei Nachricht 2 erhalten und als URL-Pfadname ab jetzt im Laufe der weiteren Verbindung verwenden wird (A_26936-*). Die Absicht der Aufführung ist, eine etwaige Fehlersuche bspw. bei IOP-Tests zu unterstützen.

A_26941 -ZETA/ASL-Protokoll: ASL/ZT-Client: Erhalt von Nachricht 4
Ein ZETA/ASL-Client MUSS die für Nachricht 4 erzeugten Datenobjekte nach A_26940-* verarbeiten können.

Er MUSS die Nachrichten 1, 2 und 3 konkatenieren und den SHA-256-Hashwert erzeugen. Dies ist der Transskript-Hashwert. Er MUSS mittels K2_s2c_key_confirmation das Chiffrat "AEAD_ct_key_confirmation" entschlüsseln (vgl. A_26934-*).
Der erhaltene Klartext ist der gesendete Transskript-Server-Hashwert.
Schlägt die Entschlüsselung fehl oder wird eine Ungleichheit zwischen eben berechneten Transskript-Hashwert und gesendeten Transskript-Server-Hashwert (Klartext des Chiffrats) festgestellt, MUSS er den Handshake mit einem Fehler abbrechen.**[<=]**

7.2 Transport und Sicherung der Nutzdaten

A_26926 -ZETA/ASL: ZETA/ASL-Client, Verschlüsselungszähler
Ein ZETA/ASL-Client MUSS sicherstellen, dass der Schlüssel K2_c2s_app_data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Der ZETA/ASL-Client MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.**[<=]**

Hinweis: Ein Zählerüberlauf kann praktisch nie erreicht werden. Würde ein Client im Nano-Sekunden-Takt den Zähler erhöhen, würde erst nach mehr als 583 Jahren der Überlauf eintreten. Deshalb wird auf einen Überlauftest verzichtet.

A_26927 -ZETA/ASL: ZETA/ASL-Client: Request erzeugen/verschlüsseln
Ein ZETA/ASL-Client MUSS einen inneren HTTP-Request als Klartext erzeugen.

Er MUSS den Klartext mittels AES/GCM und dem Schlüssel K2_c2s_app_data verschlüsseln (siehe AAD weiter unten).
Dafür wird ein 32-Bit Zufallswert a erzeugt. Nach A_26926-* wird der mit K2_c2s_app_data verbundene Zähler um eins erhöht. Es wird der IV mit IV=a || Zähler erzeugt (dessen Länge ist damit 96-Bit). Dieser IV wird für die AES/GCM-Verschlüsselung verwendet.

Er MUSS einen Request-Counter pflegen, der verbunden ist mit der KeyID. Für jeden Request MUSS der ZETA/ASL-Client diesen Request-Counter erhöhen, bei Empfang einer Antwort ist es dem Client möglich, die Response zum zuvor gestellten Request sicher zuzuordnen (vgl. A_26931-*, vgl. Erläuterungen zu A_26927-*).

Es wird folgender Header erzeugt:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/ Request	1 Byte	Für eine Nachricht des ZETA/ASL-Clients an einen ZTA/ASL-Server wird der Wert auf 1 gesetzt.

		In der Kodierung des Response-Chiffrats hat dieses Byte den Wert 2, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_26937-*)

Dieser Header stellt die "Additional Associated Data" dar, die in die Berechnung des Authentication-Tag bei der AES/GCM-Verschlüsselung einfließen MÜSSEN. Der Authentication-Tag MUSS 128 Bit lang sein.

Das erweitere Chiffrat (also inkl. Header) MUSS folgende Struktur haben:

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/ Request	1 Byte	Für eine Nachricht des ZETA/ASL-Clients an einen ZETA/ASL-Server wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrats wird es auf den Wert 2 gesetzt, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request Für jeden neuen Request wird vom Client dieser Wert um eins erhöht. Die Zählernummer wird vom Client in Network-Byte-Order (= Big-Endian) kodiert.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_26937-*)
IV	12 Byte (= 96 Bit)	IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A_26927-*)
CT	variabel	eigentliche AES/GCM-Chiffrat, dessen Länge gleich der Länge des Klartextes ist
GMAC-Wert	16 Byte (= 128 Bit)	Authentication-Tag, der während der AES/GCM-Verschlüsselung inkl. der Associated Data (Daten aus der Header-Tabelle, s. o.) berechnet wird.

Der ZETA/ASL-Client MUSS diese Datenstruktur per HTTP-POST an den ZETA/ASL-Server senden und als URL-Pfadnamen dabei den Wert der ZETA/ASL-CID (vgl. A_26936-*) verwenden. Dabei MUSS er den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden.
[<=]

Erläuterung:

Der Verschlüsselungszähler, der im Client mit dem Schlüssel K2_c2s_app_data verbunden ist, hat die Funktion für den Galois Counter Mode (GCM) sicherzustellen, dass der jeweils pro Verschlüsselungsvorgang erzeugte Initialisierungsvektor (IV) (bei gleichen Wert von K2_c2s_app_data) einzigartig ist.

Der Request-ID-Zähler hat drei Funktionen. Einmal soll im Client eine Response des Servers sicher einem vorher gesendeten Request zuordenbar sein. Sollte im Verbindungsaufbau "Enforce Replay Protection" aktiviert worden sein, prüft der Server, ob Requests mit gleicher Request-ID mehrfach eingetroffen sind (ähnlich der "Anti Replay Window"-Technik bei IPsec). Wenn analog "Enforce Sequence Order" aktiviert worden ist, dann prüft der Server die Folge der Request-ID der einkommenden Requests auf strenge Monotonie.

In der aktuellen Ausbaustufe von ePA für alle werden die letzten beiden Funktionen nicht benötigt (wie auch aktuell beim E-Rezept nicht).

Da Verschlüsselungszähler und Request-ID unterschiedliche Funktionen/Motivationen besitzen, sind sie als separate (theoretisch auch im Wert unterschiedliche) Datenobjekte aufgeführt.

A_26928 -ZETA/ASL: ZETA/ASL-Server: Request entschlüsseln/auswerten
Ein Request erreicht den ZETA/ASL-Server über eine HTTPS-Schnittstelle. Im äußeren Request nimmt der ZETA/ASL-Server das Routing mittels der Verbindungs-ID (URL-Pfadname, A_26936-* und A_26927-*) und/oder der KeyID im Header des Chiffrats vor.

Ein ZETA/ASL-Server MUSS bei Erhalt eines Chiffrat nach A_26927-* folgendes prüfen.

1. Ist die Länge der Datenstruktur mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. S. v. korrekte Umgebung, in der auch der ZETA/ASL-Server arbeitet.
3. Ist das Request/Response-Byte gleich 1.
4. Ist die KeyID bekannt.

Sollte eine dieser Prüfungen oder eine weitere der folgenden Prüfungen ein nicht-positives Prüfergebnis ergeben, so MUSS der ZETA/ASL-Server die Verarbeitung des Requests abbrechen und mit einer Fehlermeldung (vgl. A_26924-*) dem Client antworten.

Der ZETA/ASL-Server MUSS den mit der KeyID verbundenen Schlüssel K2_c2s_app_data für die Entschlüsselung des Chiffrats verwenden und dabei analog A_26927-* den Header als "Additional Associated Data" mit in die Entschlüsselung einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS Abbruch und Fehlermeldung (vgl. A_26924-*) erfolgen. Der ZETA/ASL-Server MUSS den Request-Counter-Wert speichern, der für die Antwort (A_26930-*) benötigt wird.**[<=]**

Hinweis:

"Enforce Replay Protection" und "Enforce Sequence Order" werden in der aktuellen Ausbaustufe nicht umgesetzt, weil fachlich noch nicht benötigt.

A_26929 -ZETA/ASL: ZETA/ASL-Server, Verschlüsselungszähler
Ein ZETA/ASL-Server MUSS sicherstellen, dass der Schlüssel K2_s2c_app_data als Attribut einen 64-Bit-Zähler besitzt. Dieser Zähler MUSS initial 0 sein. Ein ZETA/ASL-Server MUSS sicherstellen, dass bei jeder Verwendung des Schlüssels bei der Verschlüsselung einer Nachricht der Zähler um eins erhöht wird. Der erste verwendete Zählerwert ist also 1. Es spielt dabei keine Rolle, ob die Nachricht (Chiffrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden.**[<=]**

Verständnishinweis: analog A_26926-*.

A_26930 -ZETA/ASL: ZETA/ASL-Server, Response erstellen/verschlüsseln
In der ZETA/ASL-Server wird der innere HTTP-Request (Ergebnis aus A_26928-*) fachlich verarbeitet und als Antwort eine innere HTTP-Response erzeugt. Diese ist der Klartext, der jetzt behandelt wird.

Ein ZETA/ASL-Server MUSS analog zu A_26927-* einen Header erzeugen, wobei er

1. beim Response/Request-Byte den Wert 2 verwenden MUSS,
2. im Request-Counter den gespeicherten Wert aus A_26928-* (Eingang des Requests) verwenden MUSS.

Die anderen Header-Variablen MÜSSEN analog zu A_26927-* festgelegt werden.
Die Konstruktion des IV MUSS analog zu A_26927-* erfolgen, nur dass für den Verschlüsselungszähler der Wert nach A_26929-* (K2_s2_app_data) verwendet werden MUSS.
Mit dem IV und dem Schlüssel K2_s2c_app_data MUSS der Klartext (s. o.) verschlüsselt werden mit AES/GCM, wobei der Header als "Additional Associated Data" bei der Verschlüsselung mit einfließt.

Das so erzeugte erweiterte Chiffrat (vgl. A_26927-*, also Chiffrat inkl. Header) MUSS an den Client im HTTP-Response-Body versendet werden, wobei das Aktensystem den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden MUSS.**[<=]**

A_26931 -ZETA/ASL: ZETA/ASL-Client, Response entschlüsseln/auswerten
Ein ZETA/ASL-Client MUSS bei Erhalt einer Antwort gemäß A_26930-* auf einen Request gemäß A_26927-* folgendes prüfen:

1. Ist die Länge der Datenstruktur (erweiterte Chiffrat) mindestens 72 Byte lang.
2. Ist die "PU/nonPU" Byte korrekt, i. sS v. korrekte Umgebung, in der auch der ZETA/ASL-Client arbeitet.
3. Ist das Request/Response-Byte gleich 2.
4. Hat der Response-Counter den von Client erwarteten Wert.
5. Ist die KeyID bekannt.

Ergibt eine der Prüfungen ein nicht-positives Ergebnis, MUSS der Client die Verarbeitung der Response abbrechen.
Er MUSS das Chiffrat mit dem mit der KeyID verbundenen Schlüssel K2_s2c_app_data mittels AES/GCM entschlüsseln und dabei den Header als "Additional Associated Data" in die Entschlüsselung mit einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS er die Verarbeitung des Chiffrats/Response abbrechen.**[<=]**

7.3 Fehlersignalisierung

A_26924 -ZETA/ASL: ZETA/ASL-Server, Erzeugung von Fehlermeldungen

Eine ZETA/ASL-Server MUSS das Auftreten von Fehlern bei der Abarbeitung des ZETA/ASL-Protokolls wie folgt an den ZETA/ASL-Client melden.

Als HTTP-Response-Code MUSS der ZETA/ASL-Server einen HTTP-Fehlercode (i. S. v. eben nicht 200) in der äußeren HTTP-Response verwenden (siehe folgende Tabelle). Es MUSS weiterhin eine Datenstruktur der folgenden Art erzeugen:

```
{
            "MessageType" : "Error",
            "ErrorCode" : ...natürliche-Zahl...,
            "ErrorMessage" : "... Menschenlesbarer Text bzw.
Fehlerursache ..."
}
```

Diese Datenstruktur MUSS es per CBOR [RFC-CBOR] serialisieren und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) im äußeren Response-Body aufführen.

Ein ZETA/ASL-Server KANN weitere selbst definierte Variablen-Werte-Paare in der o. g. Datenstruktur aufführen.

Folgende Fehler-Meldungen MUSS es mindestens geben:

HTTP-Error	ErrorCode	ErrorMessage / Beschreibung
400	1	"Decoding Error" Fehler in einer Kodierung bspw. der CBOR-Kodierung
400	2	"Missing Parameters" notwendige Datenfelder bspw. in der Handshake fehlen
403	3	"GCM returns FAIL" Eine AES/GCM-Entschlüsselung ergibt das Symbol "FAIL".
403	4	"PU/nonPU Failure"
403	5	"Transscript Error" Im Handshake wird Ungleichheit zwischen den beiden Transskript-Hashwerten festgestellt.
400	6	"bad format: extended ciphertext" Die erste Sanity-Prüfung des erweiterten Chiffrat bspw. bei A_26928-* schlägt fehl.
403	7	"is not a request"

		A_26928-* (Prüfschritt 3)
403	8	"unknow KeyID"
403	9	"unknown CID"

[<=]

A_26925 -ZETA/ASL: ZETA/ASL-Client, Verarbeitung von Fehlermeldungen
Ein ZETA/ASL-Client MUSS die Fehlermeldungen nach A_26924-* verarbeiten können.
[<=]

7.4 Tracing in Nichtproduktivumgebungen

A_26942 -ZETA/ASL-Client, Nichtproduktivumgebung, Offenlegung von symmetrischen Verbindungsschlüsseln
Ein ZETA/ASL-Client in einer Nichtproduktivumgebung MUSS nach einer erfolgreichen Schlüsselaushandlung mit dem ZETA/ASL-Protokoll die ausgehandelten symmetrischen Schlüssel K2_c2s_app_data und K2_s2c_app_data base64-kodiert innerhalb einer HTTP-Request-Header-Variable "ZETA/ASL-nonPU-Tracing" bei jedem äußeren HTTP-Request aufführen. Dabei sind die beiden Base64-kodierten Schlüsselwerte durch Leerzeichen zu trennen. Beispiel:

```
ZETA-ASL-nonPU-Tracing: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=
```

Als erstes ist der Wert von Schlüssel K2_c2s_app_data aufzuführen, als zweiter Wert ist der Wert von Schlüssel K2_s2c_app_data aufzuführen.**[<=]**

Hinweis: Ebenfalls muss ein ZETA/ASL-Client nach A_26927-* im Chiffrat kennzeichnen, ob es eine Nachricht in einer Nichtproduktivumgebung ist oder nicht.

A_26943 -ZETA/ASL-Server, Nichtproduktivumgebung, Ablehnung von nonPU-Chiffraten in der PU
Ein ZETA/ASL-Server in der Produktivumgebung MUSS äußere HTTPS-Request, die im Request-Header eine Variable "ZETA/ASL-nonPU-Tracing" enthalten, mit einer HTTP-Fehlermeldung 403-Forbidden beantworten. Der ZETA/ASL-Server MUSS sicherstellen, dass der innere Request (Chiffrat im Request-Body des äußeren Requests) nicht an einen Resource-Server weitergereicht wird.
Ein ZETA/ASL-Server in der Produktivumgebung, die am Chiffrat am nonPU/PU-Byte erkennt (vgl. A_26927-*), dass es sich um ein nonPU-Chiffrat handelt, MUSS den Request mit einer Fehlermeldung an die äußere Schicht des Aktensystems (Webschnittstelle) beantworten. Der ZETA/ASL-Server MUSS sicherstellen, dass das Chiffrat nicht entschlüsselt und das Chiffrat sicher gelöscht wird.**[<=]**

8 Post-Quanten-Kryptographie (informativ)

Wie in [BSI-PQC-2020] dargestellt, erscheint es mit Blick auf dem aktuellen Stand von Wissenschaft und Technik als sehr unwahrscheinlich, dass aktuell Quanten-Computer in ausreichender Leistungsstärke verfügbar sind, die der asymmetrischen Kryptographie -- so wie sie aktuell in der TI eingesetzt wird -- gefährlich werden könnten.

Dies bedeutet, dass insbesondere Authentisierungsvorgänge, die im "Jetzt" mit klassischen asymmetrischen kryptographischen Verfahren mit ausreichend großen Schlüssellängen durchgeführt werden (vgl. [SOG-IS] und [BSI-TR-02102-1]), nicht in Frage stehen.

Schwieriger wird es mit der Verschlüsselung mit klassischen asymmetrischen Verfahren bei denen Chiffrate erzeugt werden, die ein Angreifer sammeln kann, und durch deren Entschlüsselung zu einer Zeit in der Zukunft, bei der solche Quanten-Computer verfügbar sind, dann Schaden entstehen kann ("harvest now, decrypt later"), vgl. [EU-BSI-JointStatement].

Dort erscheint es ratsam auf hybride Lösungen zu migrieren. D. h., etablierte kryptographische asymmetrische Verfahren und post-quantum-sichere asymmetrische kryptographische Verfahren werden in einer Weise kombiniert, dass wenn genau nur eines der beiden Verfahren gebrochen wird, die Sicherheit (Vertraulichkeit und Integrität) der verschlüsselten Daten nicht gefährdet ist.

Aktuell gibt es diesbezüglich verschiedene Forschungsaktivitäten und Vorschläge für eine entsprechende hybride Protokoll-Erweiterungen von kryptographischen Protokollen wie TLS, IKE oder SSH [TLS-CC-2021], [ENISA-PQC-2021#6.1]. Die gematik beobachtet intensiv die laufenden Forschungs- und Evaluierungsaktivitäten [NIST-PQC] die voraussichtlich 2024 einen geeigneten Stand erreichen werden. Beim VAU-Protokoll für ePA für alle wird bereits eine PQC-sichere Schlüsselaushandlung auf Basis von [KEM-TLS] und [IETF-Hybrid-TLS] eingesetzt. D. h., Anwendungsdaten zwischen ePA-Clients und ePA-VAU sind beim Transport bereits PQC-sicher. In einer nächsten Ausbaustufe verwendet der E-Rezept-Projekt ebenfalls diese VAU-Protokollvariante für die Verbindung zwischen E-Rezept-Client und E-Rezept-VAU.

Bei Messengern [PQC-Hybrid-Signal] oder bei Web-Browsern [PQC-Hybrid-Chrome] nimmt die Verwendung von PQC-sicheren Hybrid Verfahren immer mehr zu. Es ist absehbar, dass dieses Vorgehen bald generell der Stand der Technik ist.

9 Anhang – Verzeichnisse

9.1 Abkürzungen

Kürzel	Erläuterung
ASL	Additional Security Layer
BS	betreiberspezifischer Schlüssel
C2C	Card to Card
C2S	Card to Server
CEK	Content Encryption Key
CA	Certificate Authority
CBC	Cipher Block Chaining
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DRNG	Deterministic Random Number Generator
eGK	elektronische Gesundheitskarte
HCV	Hash Check Value
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
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OSI	Open Systems Interconnection
SAK	Signaturanwendungskomponente
SE	Secure Element
SM	Service Monitoring
TCB	Trusted Computing Base
TI	Telematikinfrastruktur
TLS	Transport Layer Security
TPM	Trusted Plattform Module
TSIG	Transaction Signature
URI	Uniform Resource Identifier
VAU	vertrauenswürdige Ausführungsumgebung, vgl. [gemSpec_Aktensystem_ePAfueralle]
WANDA Basic	Weitere Anwendungen für den Datenaustausch ohne Nutzung der TI oder derer kryptografischen Identitäten

9.2 Glossar

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

9.3 Abbildungsverzeichnis

Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht.....28

Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält.....84

Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU 95

Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-VAU-Instanz 108

Abbildung 5: OSI-Schichten in einem Aktensystem.....110

Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS] 111

Abbildung 7: OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI.....126

9.4 Tabellenverzeichnis

Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten.....10

Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „RSA“.....11

Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“.....12

Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter elektronischer Signaturen für die Schlüsselgeneration „RSA“.....14

Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“.....15

Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate.....17

Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate.....17

Tabelle 8: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen Schlüssels.....20

Tabelle 9: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines versichertenindividuellen Schlüssels.....20

Tabelle 10: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten elektronischen XML-Signaturen.....29

Tabelle 11: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen.....30

Tabelle 12: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung.....33

Tabelle 13: Tab_KRYPT_017 Algorithmen für DNSSEC.....44

Tabelle 14: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären Daten im Kontext von Dokumentensignaturen.....46

Tabelle 15: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-Dokumentensignaturen.....47

Tabelle 16: Tab_KRYPT_ERP Definition Datenstruktur PKI-Zertifikatsliste.....98

Tabelle 17: API von GET /OCSPResponse.....99
Tabelle 18: Tab_KRYPT_ERP_FdV_Truststore_aktualisieren.....100
Tabelle 19: Tab_KRYPT_ERP Kodierung des Chiffrats aus A_20161-*.....102
Tabelle 20: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept).....106

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.

[Quelle]	Herausgeber: Titel
[gemGlossar]	gematik: Glossar der Telematikinfrastruktur
[gemSpec_COS]	gematik: Spezifikation des Card Operating System (COS)
[gemSpec_DS_Anbieter]	Spezifikation Datenschutz- und Sicherheitsanforderungen der TI an Anbieter
[gemSpec_eGK_ObjSys]	gematik: Die Spezifikation der elektronischen Gesundheitskarte (eGK) – Objektsystem
[gemSpec_KT]	gematik: Spezifikation eHealth-Kartenterminal
[gemSpec_MobKT]	gematik: Spezifikation Mobiles Kartenterminal
[gemSpec_SST_FD_VSDM]	gematik: Schnittstellenspezifikation Fachdienste (UFS/VSDD/CMS)

9.5.2 Weitere Dokumente

[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.demit 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-PQC-2020]	Migration zu Post-Quanten-Kryptografie, Handlungsempfehlungen des BSI, Stand: August 2020, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?__blob=publicationFile&v=2
[BSI-TR-02102-1]	BSI TR-02102-1 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“ Version 2025-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.html
[BSI-TR-02102-2]	BSI TR-02102-2 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 2 – Verwendung von Transport Layer Security (TLS), Version 2025-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.html
[BSI-TR-02102-3]	BSI TR-02102-3 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 3 – Verwendung von Internet Protocol Security (IPsec) und Internet Key Exchange (IKEv2)“ Version 2025-01 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-3.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/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03111/TR-03111_node.html
[BSI-TR-03116-1]	Technische Richtlinie BSI TR-03116-1 Kryptographische Vorgaben für Projekte der Bundesregierung, Version: 3.20, Fassung September 2018, 21.09.2018 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116.html
[CM-2014]	20 Years of SSL/TLS Research, An Analysis of the Internet’s Security Foundation, Christopher Meyer, 9. February 2014 http://www-brs.ub.ruhr-uni-bochum.de/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

[ecma-262]	JSON Standard https://www.ecma-international.org/publications/standards/Ecma-262.htm
[EN-14890-1]	DIN EN 14890-1:2008 Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic services
[ENISA-PQC-2021]	Post-Quantum Cryptography: Current state and quantum mitigation, European Union Agency for Cybersecurity (ENISA), May 2021, https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation
[ETSI-CAdES]	ETSI TS 101 733 V1.7.4 (2008-07), Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)
[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)
[EU-BSI-JointStatement]	Securing Tomorrow, Today: Transitioning to Post-Quantum Cryptography date, 01.07.2025, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/PQC-joint-statement-2025.html
[FIPS-180-4]	Federal Information, Processing Standards Publication 180-4, Secure Hash Standard (SHS), March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf
[FIPS-186-2+CN1]	FIPS 186-2 - National Institute of Standards and Technology, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2, January 27, 2000 – Appendix 3.1 unter der Beachtung des Change Notice 1, vom 5. Oktober 2001 http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2-change1.pdf
[FIPS-186-5]	FIPS 186-5 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (Supersedes FIPS 186-4) Digital Signature Standard (DSS), 03.02.2023 https://csrc.nist.gov/pubs/fips/186-5/final https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf
[FIPS-197]	Federal Information Processing Standards Publication 197, (FIPS–197), November 26, 2001, Announcing the ADVANCED ENCRYPTION STANDARD (AES) http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[FIPS-202]	NIST, FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015, http://nvlpubs.nist.gov/nistpubs/fips/NIST.FIPS.202.pdf
[IETF-Hybrid-TLS]	Hybrid key exchange in TLS 1.3 Douglas Stebila, Scott Fluhrer, Shay Gueron https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/
[IETF-KEM-TLS]	KEM-based Authentication for TLS 1.3 T. Wiggers, S. Celi, P. Schwabe, D. Stebila, N. Sullivan https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/
[IETF-Kyber]	Kyber Post-Quantum KEM, P. Schwabe, B. E. Westerbaan Cloudflare, 2024, https://datatracker.ietf.org/doc/draft-cfrg-schwabe-kyber/
[IR-2014]	Bulletproof SSL and TLS: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Ivan Ristić, 2014 https://www.feistyduck.com/books/bulletproof-ssl-and-tls/
[ISO-11770]	ISO/IEC 11770: 1996, Information technology – Security techniques – Key management, Part 3: Mechanisms using asymmetric techniques
[Ker-1883]	Auguste Kerckhoffs, "La cryptographie militaire", Journal des sciences militaires, vol. IX, Seite 5–83, Jan. 1883, Seite 161–191, Feb. 1883. siehe auchhttp://www.petitcolas.net/fabien/kerckhoffs/
[KEM-TLS]	Post-quantum TLS without handshake signatures Peter Schwabe, Douglas Stebila, and Thom Wiggers https://eprint.iacr.org/2020/534
[KS-2011]	W. Killmann, W. Schindler, „A proposal for: Functionality classes for random number generators“, Version 2.0, September 2011 https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifierung/Interpretation /AIS31_Functionality_classes_for_random_numder_generators.pdf?__blpb=publicationFile
[MK-2016]	The Million-Key Question – Investigating the Origins of RSA Public Keys, Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, Vashek Matyas, The 25th USENIX Security Symposium (UsenixSec'2016) https://crocs.fi.muni.cz/public/papers/usenix2016
[NIST-PQC]	Post-Quantum Cryptography, National Institute of Standards and Technology (NIST), https://csrc.nist.gov/projects/post-quantum-cryptography

[NIST-SP-800-22]	A. Ruskin et al., A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, SP 800-22 Rev. 1a , 2010 https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
[NIST-SP-800-38A]	NIST Special Publication 800-38A, Recommendation for Block, Cipher Modes of Operation, Methods and Techniques, Morris Dworkin, December 2001 Edition, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
[NIST-SP-800-38B]	NIST Special Publication 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Morris Dworkin, May 2005 Edition, http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
[NIST-SP-800-38D]	NIST Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, Morris Dworkin, November, 2007
[NIST-SP-800-56-A]	NIST Special Publication 800-56A Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, April 2018 https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final
[NIST-SP-800-56-B]	NIST Special Publication 800-56B Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography, August 2009
[NIST-SP-800-56C]	NIST Special Publication 800-56C Recommendation for Key Derivation through Extraction-then-Expansion, November 2011
[NIST-SP-800-108]	NIST Special Publication 800-108 Recommendation for Key Derivation Using Pseudorandom Functions, October 2009
[NK-PP]	Common Criteria Schutzprofil (Protection Profile) Schutzprofil 1: Anforderungen an den Netzkonnektor, BSI-CC-PP-0097
[Oorschot-Wiener-1996]	On Diffie-Hellman Key Agreement with Short Exponents, Paul C. van Oorschot, Michael J Weiner, Eurocrypt’ 96
[Padding-Oracle-2005]	Padding Oracle Attacks on CBC-mode Encryption with Secret and Random IVs Arnold K. L. Yau, Kenneth G. Paterson and Chris J. Mitchell, FSE 2005

	http://www.isg.rhul.ac.uk/~kp/secretIV.pdf
[PAdES-3]	ETSI TS 102 778-3 V1.2.1, PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced – PAdES-BES and PAdES-EPES Profiles Technical Specification, 2010
[PDF/A-2]	ISO 19005-2:2011 – Document management – Electronic document file format for long-term preservation – Part 2: Use of ISO 32000-1 (PDF/A-2)
[PKCS#1]	vgl. [RFC-8017]
[PP-0082]	Common Criteria Protection Profile, Card Operating System Generation 2 (PP COS G2), BSI-CC-PP-0082-V2, Version 1.9, 18th November 2014
[PQC-Hybrid-Chrome]	Protecting Chrome Traffic with Hybrid Kyber KEM, Thursday, August 10, 2023 https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html
[PQC-Hybrid-Signal]	The PQXDH Key Agreement Protocol, Revision 2, 2023-05-24 https://signal.org/docs/specifications/pqxdh/ (vgl. auch https://signal.org/blog/pqxdh/)
[RFC-2119]	RFC 2119 (März 1997): Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, http://tools.ietf.org/html/rfc2119
[RFC-2590]	RFC 2590 (June 1999): X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP https://tools.ietf.org/html/rfc2560 (Obsoleted by [RFC-6960])
[RFC-2986]	RFC 2986 (November 2000): PKCS #10: Certification Request Syntax Specification, Version 1.7 https://tools.ietf.org/html/rfc2986
[RFC-3279]	RFC 3279 (April 2002): Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

	https://tools.ietf.org/html/rfc3279
[RFC-3526]	RFC 3526 (Mai 2003: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) http://tools.ietf.org/html/rfc3526
[RFC-4051]	Additional XML Security Uniform Resource Identifiers (URIs), April 2005 https://tools.ietf.org/html/rfc4051
[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-7519]	RFC 7519 JSON Web Token (JWT), M. Jones, J. Bradley, N. Sakimura, Mai 2015 https://datatracker.ietf.org/doc/html/rfc7519
[RFC-8017], [PKCS#1]	"Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2", November 2016 https://tools.ietf.org/html/rfc8017
[RFC-931]	RFC 6931: Additional XML Security Uniform Resource Identifiers (URIs), Donald Eastlake, April 2013,https://tools.ietf.org/html/rfc6931
[RFC-9155]	RFC 9155:Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2, December 2021,

	https://datatracker.ietf.org/doc/rfc9155/
[RFC-CBOR]	RFC-8949: Concise Binary Object Representation (CBOR), C. Bormann, P. Hoffman, December 2020, https://datatracker.ietf.org/doc/html/rfc8949
[ROCA-2017]	The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli, Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, Vashek Matyas 24th ACM Conference on Computer and Communications Security (CCS'2017) https://crocs.fi.muni.cz/public/papers/rsa_ccs17
[SEC1-2009]	Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography, Certicom Research, Contact: Daniel R. L. Brown (dbrown@certicom.com), May 21, 2009, Version 2.0 https://www.secg.org/sec1-v2.pdf
[SDH-2016]	Measuring the Security Harm of TLS Crypto Shortcuts, Drew Springall, Zakir Durumeic, J. Alex Halderman, November 2016,https://jhalderm.com/pub/papers/forward-secrecy-imc16.pdf
[SOG-IS]	SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms, Version 1.3, January 2023 https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf
[TLS-Attacks]	Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses, Christopher Meyer und Jörg Schwenk, 31. Januar 2013,http://eprint.iacr.org/2013/049
[TLS-CC-2021]	Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS), September 2021, https://datatracker.ietf.org/doc/draft-campagna-tls-bike-sike-hybrid/
[XMLCan_V1.0]	Exclusive XML Canonicalization, Version 1.0, W3C Recommendation 18 July 2002,http://www.w3.org/TR/xml-exc-c14n/
[XMLDSig]	XML Signature Syntax and Processing Version 1.1, W3C Recommendation 11 April 2013 https://www.w3.org/TR/xmldsig-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/xmldsig-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