

**Elektronische Gesundheitskarte und Telematikinfrastruktur**

# Übergreifende Spezifikation Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur

|                  |                                  |
|------------------|----------------------------------|
| Version:         | 2. <del>28</del> 29.0            |
| Revision:        | <del>645058832330</del>          |
| Stand:           | <del>09.06.2023</del> 30.01.2024 |
| Status:          | freigegeben                      |
| Klassifizierung: | öffentlich                       |
| Referenzierung:  | gemSpec_Krypt                    |

---

## Dokumentinformationen

---

### Änderungen zur Vorversion

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

### Dokumentenhistorie

| Version | Datum    | Grund der Änderung, besondere Hinweise   | Bearbeitung |
|---------|----------|--|-------------|
| 2.21.0  | 31.01.22 | Einarbeitung ePA_Maintenance_21.5 und<br>Konn_Maintenance_21.6   | gematik     |
| 2.22.0  | 01.03.22 | Einarbeitung E-Rezept_Maintenance_21.3   | gematik     |
| 2.23.0  | 20.09.22 | Einarbeitung gemSpec_Krypt_Maintenance_22.1<br>(neu: Kap. 2.5)<br>Einarbeitung Änderungsliste E-Rezept_Maintenance<br>22.2   | gematik     |
| 2.24.0  | 08.12.22 | Einarbeitung CI_Maintenance_22.5,<br>Konn_Maintenance_22.5 und 22.6,<br>Kap. 5.4 Typo raus, redaktionelle Anpassungen  | gematik     |
| 2.25.0  | 14.02.23 | Einarbeitung VSDM++, Maintenance_23.1  | gematik     |
| 2.26.0  | 09.03.23 | Einarbeitung Konn_Maintenance_23.0   | gematik     |
| 2.27.0  | 14.04.23 | Einarbeitung Konn_Maintenance_23.1   | gematik     |
| 2.28.0  | 09.06.23 | Einarbeitung VSDM_Maintenance_23.2<br>und CI_Maintenance_23.1  | gematik     |
| 2.29.0  | 30.01.24 | Einarbeitung ePA für alle: Kap. "VAU-Protokoll für E-<br>Rezept" überarbeitet, neues Kap. "VAU-Protokoll für<br>ePA für alle" eingefügt<br>redaktionelle Anpassungen (z. B. -* bei Afo-<br>Referenzen ergänzt) | gematik     |

## Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>1 Einführung</b>   | <b>10</b> |
| 1.1 Zielsetzung und Einordnung des Dokuments                              | 10        |
| 1.2 Zielgruppe  | 10        |
| 1.3 Geltungsbereich   | 11        |
| 1.4 Abgrenzung des Dokuments  | 11        |
| 1.5 Methodik  | 11        |
| <b>2 Einsatzszenarioübergreifende Algorithmen</b>                         | <b>12</b> |
| 2.1 Identitäten   | 12        |
| 2.1.1 X.509-Identitäten   | 12        |
| 2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen             | 14        |
| 2.1.1.2 Qualifizierte elektronische Signaturen                            | 17        |
| 2.1.1.3 TLS-Authentifizierung   | 19        |
| 2.1.1.4 IPsec-Authentifizierung   | 20        |
| 2.1.1.5 Digitale Signaturen durch TI-Komponenten                          | 20        |
| 2.1.1.6 Verschlüsselung   | 20        |
| 2.1.2 CV-Identitäten  | 20        |
| 2.1.2.1 CV-Zertifikate G2   | 20        |
| 2.1.2.2 CV-Certification-Authority (CV-CA) Zertifikat G2                  | 21        |
| 2.2 Zufallszahlengeneratoren  | 21        |
| 2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren)                  | 22        |
| 2.4 Schlüsselerzeugung und Schlüsselbestätigung                           | 22        |
| 2.4.1 Prüfung auf angreifbare (schwache) Schlüssel                        | 24        |
| 2.4.2 ECC-Schlüssel in X.509-Zertifikaten                                 | 25        |
| 2.4.3 RSA-Schlüssel in X.509-Zertifikaten                                 | 25        |
| 2.5 Einmalpasswörter  | 26        |
| <b>3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien</b> | <b>27</b> |
| 3.1 Kryptographische Algorithmen für XML-Dokumente                        | 27        |
| 3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen                   | 28        |
| 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  | 36        |

|  |           |
|--|-----------|
| 3.3.3 DNSSEC-Kontext.....  | 43        |
| <b>3.4 Masterkey-Verfahren (informativ).....</b>                             | <b>44</b> |
| <b>3.5 Hybride Verschlüsselung binärer Daten.....</b>                        | <b>46</b> |
| 3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten .....  | 46        |
| 3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten ..... | 47        |
| <b>3.6 Symmetrische Verschlüsselung binärer Daten.....</b>                   | <b>47</b> |
| <b>3.7 Signatur binärer Inhaltsdaten (Dokumente) .....</b>                   | <b>48</b> |
| <b>3.8 Signaturen innerhalb von PDF/A-Dokumenten.....</b>                    | <b>49</b> |
| <b>3.9 Kartenpersonalisierung.....</b>                                       | <b>50</b> |
| <b>3.10 Bildung der pseudonymisierten Versichertenidentität .....</b>        | <b>50</b> |
| <b>3.11 Spezielle Anwendungen von Hashfunktionen .....</b>                   | <b>50</b> |
| 3.11.1 Hashfunktionen und OCSP (informativ) .....                            | 51        |
| <b>3.12 kryptographische Vorgaben für die SAK des Konnektors.....</b>        | <b>52</b> |
| <b>3.13 Migration im PKI-Bereich .....</b>                                   | <b>52</b> |
| <b>3.14 Spezielle Anwendungen von kryptographischen Signaturen.....</b>      | <b>53</b> |
| <b>3.15 ePA-spezifische Vorgaben .....</b>                                   | <b>54</b> |
| 3.15.1 Verbindung zur VAU .....  | 54        |
| 3.15.2 Vorgaben für ePA-Schlüssel und ePA-Chiffre .....                      | 55        |
| 3.15.3 ePA-Aktensysteminterne Schlüssel .....                                | 55        |
| 3.15.4 ePA-spezifische TLS-Vorgaben .....                                    | 58        |
| 3.15.5 Schlüsselableitungsfunktionalität ePA.....                            | 59        |
| <b>3.16 E-Rezept-spezifische Vorgaben .....</b>                              | <b>61</b> |
| <b>3.17 KOM-LE-spezifische Vorgaben .....</b>                                | <b>63</b> |
| <b>3.18 HMAC-Sicherung der Prüfziffer VSDM.....</b>                          | <b>63</b> |
| <b>3.19 spezifische TLS-Vorgaben für VSDM.....</b>                           | <b>65</b> |
| <b>4 Umsetzungsprobleme mit der TR-03116-1.....</b>                          | <b>66</b> |
| 4.1 XMLDSig und PKCS1-v2.1 .....   | 66        |
| 4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM .....                     | 66        |
| 4.3 XML Signature Wrapping und XML Encryption Wrapping .....                 | 67        |
| 4.4 Güte von Zufallszahlen .....   | 67        |
| <b>5 Migration 120-Bit-Sicherheitsniveau.....</b>                            | <b>68</b> |
| 5.1 PKI-Begriff Schlüsselgeneration.....                                     | 68        |
| 5.2 X.509-Root der TI.....   | 69        |
| 5.3 TLS-Dienst und ECDSA-basierte TLS allgemein.....                         | 71        |
| 5.4 ECC-Unterstützung bei TLS.....   | 71        |
| 5.5 ECC-Unterstützung bei IPsec.....   | 73        |
| 5.6 ECDSA-Signaturen.....  | 75        |

|   |            |
|---|------------|
| 5.6.1 ECDSA-Signaturen im XML-Format .....  | 75         |
| 5.6.2 ECDSA-Signaturen im CMS-Format .....  | 75         |
| <b>5.7 ECIES .....</b>  | <b>76</b>  |
| 5.7.1 ECIES und authentifizierte Broadcast-Encryption .....                       | 80         |
| 5.7.2 ECIES und mobKT .....   | 81         |
| <b>5.8 ECC-Migration eHealth-KT .....</b>   | <b>81</b>  |
| 5.8.1 Interoperabilität zwischen eHealth-KT und Konnektor .....                   | 83         |
| <b>5.9 ECC-Migration Konnektor .....</b>  | <b>85</b>  |
| <b>5.10 Verschiedene Produkttypen und ECC-Migration (informativ) .....</b>        | <b>86</b>  |
| <b>6 Kommunikationsprotokoll zwischen ePA-VAU und ePA-Clients .....</b>           | <b>88</b>  |
| 6.1 Motivation .....  | 88         |
| 6.2 Übersicht .....   | 88         |
| 6.3 VAUClientHello-Nachricht .....  | 90         |
| 6.4 VAUServerHello-Nachricht .....  | 92         |
| 6.5 Schlüsselableitung .....  | 94         |
| 6.6 VAUClientSigFin-Nachricht .....   | 95         |
| 6.7 VAUServerFin-Nachricht .....  | 96         |
| 6.8 Nutzerdatentransport .....  | 98         |
| 6.9 VAUServerError-Nachricht .....  | 100        |
| 6.10 Abbrechen des Protokollablaufs .....   | 101        |
| 6.11 VAU-Kanal und MTOM/XOP .....   | 102        |
| 6.12 Zusätzliche HTTP-Header-Informationen .....                                  | 103        |
| 6.13 Tracing in Nichtproduktivumgebungen .....                                    | 104        |
| 6.14 Optionale PQC-Erweiterung (informativ) .....                                 | 105        |
| <b>7 Kommunikationsprotokoll zwischen E-Rezept-VAU und E-Rezept-Clients .....</b> | <b>107</b> |
| 7.1 Übersicht (informativ) .....  | 107        |
| 7.2 Definition .....  | 110        |
| 7.2.1 E-Rezept-VAU-Identität .....  | 110        |
| 7.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität .....                     | 111        |
| 7.2.3 E-Rezept-VAU-Request und -Response .....                                    | 114        |
| 7.2.4 Zufallsquelle für Clients .....   | 120        |
| <b>8 Post-Quanten-Kryptographie (informativ) .....</b>                            | <b>148</b> |
| <b>9 Erläuterungen (informativ) .....</b>   | <b>149</b> |
| 9.1 Prüfung auf angreifbare (schwache) Schlüssel .....                            | 149        |
| 9.2 RSA-Schlüssel in X.509-Zertifikaten .....                                     | 149        |

|   |            |
|---|------------|
| <b>10 Anhang — Verzeichnisse .....</b>  | <b>153</b> |
| 10.1 Abkürzungen .....  | 153        |
| 10.2 Glossar .....  | 155        |
| 10.3 Abbildungsverzeichnis .....  | 155        |
| 10.4 Tabellenverzeichnis .....  | 155        |
| 10.5 Referenzierte Dokumente .....  | 157        |
| 10.5.1 Dokumente der gematik .....  | 157        |
| 10.5.2 Weitere Dokumente .....  | 158        |
| <b>1 Einführung .....</b>   | <b>10</b>  |
| 1.1 Zielsetzung und Einordnung des Dokuments .....                              | 10         |
| 1.2 Zielgruppe .....  | 10         |
| 1.3 Geltungsbereich .....   | 11         |
| 1.4 Abgrenzung des Dokuments .....  | 11         |
| 1.5 Methodik .....  | 11         |
| <b>2 Einsatzszenarioübergreifende Algorithmen .....</b>                         | <b>12</b>  |
| 2.1 Identitäten .....   | 12         |
| 2.1.1 X.509-Identitäten .....   | 12         |
| 2.1.1.1 Digitale nicht-qualifizierte elektronische Signaturen .....             | 14         |
| 2.1.1.2 Qualifizierte elektronische Signaturen .....                            | 17         |
| 2.1.1.3 TLS-Authentifizierung .....   | 19         |
| 2.1.1.4 IPsec-Authentifizierung .....   | 20         |
| 2.1.1.5 Digitale Signaturen durch TI-Komponenten .....                          | 20         |
| 2.1.1.6 Verschlüsselung .....   | 20         |
| 2.1.2 CV-Identitäten .....  | 20         |
| 2.1.2.1 CV-Zertifikate G2 .....   | 20         |
| 2.1.2.2 CV-Certification-Authority (CV-CA) Zertifikat G2 .....                  | 21         |
| 2.2 Zufallszahlengeneratoren .....  | 21         |
| 2.3 Hilfestellung bei der Umsetzung (Zufallsgeneratoren) .....                  | 22         |
| 2.4 Schlüsselerzeugung und Schlüsselbestätigung .....                           | 22         |
| 2.4.1 Prüfung auf angreifbare (schwache) Schlüssel .....                        | 24         |
| 2.4.2 ECC-Schlüssel in X.509-Zertifikaten .....                                 | 25         |
| 2.4.3 RSA-Schlüssel in X.509-Zertifikaten .....                                 | 25         |
| 2.5 Einmalpasswörter .....  | 26         |
| <b>3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien .....</b> | <b>27</b>  |
| 3.1 Kryptographische Algorithmen für XML-Dokumente .....                        | 27         |
| 3.1.1 XML-Signaturen für nicht-qualifizierte Signaturen .....                   | 28         |
| 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         |

|  |           |
|--|-----------|
| <b>3.2 Karten-verifizierbare Authentifizierung und Verschlüsselung</b> | <b>32</b> |
| 3.2.1 Card-to-Card-Authentisierung G2                                  | 32        |
| 3.2.2 Card-to-Server (C2S) Authentisierung und Trusted Channel G2      | 33        |
| <b>3.3 Netzwerkprotokolle</b>  | <b>33</b> |
| 3.3.1 IPsec-Kontext  | 33        |
| 3.3.2 TLS-Verbindungen   | 36        |
| 3.3.3 DNSSEC-Kontext   | 43        |
| <b>3.4 Masterkey-Verfahren (informativ)</b>                            | <b>44</b> |
| <b>3.5 Hybride Verschlüsselung binärer Daten</b>                       | <b>46</b> |
| 3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten  | 46        |
| 3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten | 47        |
| <b>3.6 Symmetrische Verschlüsselung binärer Daten</b>                  | <b>47</b> |
| <b>3.7 Signatur binärer Inhaltsdaten (Dokumente)</b>                   | <b>48</b> |
| <b>3.8 Signaturen innerhalb von PDF/A-Dokumenten</b>                   | <b>49</b> |
| <b>3.9 Kartenpersonalisierung</b>                                      | <b>50</b> |
| <b>3.10 Bildung der pseudonymisierten Versichertenidentität</b>        | <b>50</b> |
| <b>3.11 Spezielle Anwendungen von Hashfunktionen</b>                   | <b>50</b> |
| 3.11.1 Hashfunktionen und OCSP (informativ)                            | 51        |
| <b>3.12 kryptographische Vorgaben für die SAK des Konnektors</b>       | <b>52</b> |
| <b>3.13 Migration im PKI-Bereich</b>                                   | <b>52</b> |
| <b>3.14 Spezielle Anwendungen von kryptographischen Signaturen</b>     | <b>53</b> |
| <b>3.15 ePA-spezifische Vorgaben</b>                                   | <b>54</b> |
| 3.15.1 Verbindung zur VAU  | 54        |
| 3.15.2 ePA-Aktensysteminterne Schlüssel                                | 55        |
| 3.15.3 ePA-spezifische TLS-Vorgaben                                    | 58        |
| <b>3.16 E-Rezept-spezifische Vorgaben</b>                              | <b>61</b> |
| <b>3.17 KOM-LE-spezifische Vorgaben</b>                                | <b>63</b> |
| <b>3.18 HMAC-Sicherung der Prüfziffer VSDM</b>                         | <b>63</b> |
| <b>3.19 spezifische TLS-Vorgaben für VSDM</b>                          | <b>65</b> |
| <b>4 Umsetzungsprobleme mit der TR-03116-1</b>                         | <b>66</b> |
| 4.1 XMLDSig und PKCS1-v2.1   | 66        |
| 4.2 XMLEnc: Die Nutzung von RSAES-OAEP und AES-GCM                     | 66        |
| 4.3 XML Signature Wrapping und XML Encryption Wrapping                 | 67        |
| 4.4 Güte von Zufallszahlen   | 67        |
| <b>5 Migration 120-Bit-Sicherheitsniveau</b>                           | <b>68</b> |
| 5.1 PKI-Begriff Schlüsselgeneration                                    | 68        |
| 5.2 X.509-Root der TI  | 69        |
| 5.3 TSL-Dienst und ECDSA-basierte TSL allgemein                        | 71        |

|  |            |
|--|------------|
| <b>5.4 ECC-Unterstützung bei TLS .....</b>                                 | <b>71</b>  |
| <b>5.5 ECC-Unterstützung bei IPsec .....</b>                               | <b>73</b>  |
| <b>5.6 ECDSA-Signaturen .....</b>  | <b>75</b>  |
| 5.6.1 ECDSA-Signaturen im XML-Format .....                                 | 75         |
| 5.6.2 ECDSA-Signaturen im CMS-Format .....                                 | 75         |
| <b>5.7 ECIES .....</b>   | <b>76</b>  |
| 5.7.1 ECIES und authentifizierte Broadcast-Encryption .....                | 80         |
| 5.7.2 ECIES und mobKT .....  | 81         |
| <b>5.8 ECC-Migration eHealth-KT .....</b>                                  | <b>81</b>  |
| 5.8.1 Interoperabilität zwischen eHealth-KT und Konnektor .....            | 83         |
| <b>5.9 ECC-Migration Konnektor .....</b>                                   | <b>85</b>  |
| <b>5.10 Verschiedene Produkttypen und ECC-Migration (informativ) .....</b> | <b>86</b>  |
| <b>6 VAU-Protokoll für E-Rezept .....</b>                                  | <b>87</b>  |
| <b>6.1 Übersicht (informativ) .....</b>                                    | <b>89</b>  |
| <b>6.2 Definition .....</b>  | <b>110</b> |
| 6.2.1 E-Rezept-VAU-Identität .....   | 110        |
| 6.2.2 Client-seitige Prüfung der E-Rezept-VAU-Identität .....              | 111        |
| 6.2.3 E-Rezept-VAU-Request und -Response .....                             | 114        |
| 6.2.4 Zufallsquelle für Clients .....                                      | 120        |
| <b>7 VAU-Protokoll für ePA für alle .....</b>                              | <b>121</b> |
| <b>7.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung .....</b>          | <b>124</b> |
| <b>7.2 Transport und Sicherung der Nutzdaten .....</b>                     | <b>135</b> |
| <b>7.3 OIDC-Authentisierung eines Clients (Nutzer) .....</b>               | <b>139</b> |
| <b>7.4 Authentisierung des E-Rezept-FD als ePA-Client .....</b>            | <b>141</b> |
| <b>7.5 Routing auf VAU-Instanzen .....</b>                                 | <b>143</b> |
| <b>7.6 Fehlersignalisierung .....</b>                                      | <b>145</b> |
| <b>7.7 Tracing in Nichtproduktivumgebungen .....</b>                       | <b>146</b> |
| 7.7.1 Zufallsquelle für Clients .....                                      | 147        |
| <b>8 Post-Quanten-Kryptographie (informativ) .....</b>                     | <b>148</b> |
| <b>9 Erläuterungen (informativ) .....</b>                                  | <b>149</b> |
| <b>9.1 Prüfung auf angreifbare (schwache) Schlüssel .....</b>              | <b>149</b> |
| <b>9.2 RSA-Schlüssel in X.509-Zertifikaten .....</b>                       | <b>149</b> |
| <b>10 Anhang – Verzeichnisse .....</b>                                     | <b>153</b> |
| <b>10.1 Abkürzungen .....</b>  | <b>153</b> |
| <b>10.2 Glossar .....</b>  | <b>155</b> |
| <b>10.3 Abbildungsverzeichnis .....</b>                                    | <b>155</b> |



|   |            |
|---|------------|
| <b>10.4 Tabellenverzeichnis .....</b>     | <b>155</b> |
| <b>10.5 Referenzierte Dokumente .....</b> | <b>157</b> |
| 10.5.1 Dokumente der gematik .....        | 157        |
| 10.5.2 Weitere Dokumente .....            | 158        |

---

## **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.

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

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

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

### **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. Dokumentenlandkarte, Produkttypsteckbrief, Leistungsbeschreibung) festgelegt und bekannt gegeben.

### **Schutzrechts-/Patentrechtshinweis**

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

## **1.4 Abgrenzung des Dokuments**

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

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

## **1.5 Methodik**

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

Sie werden im Dokument wie folgt dargestellt:

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

Text / Beschreibung

[<=]

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

---

## **2 Einsatzszenarioübergreifende Algorithmen**

---

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

### **GS-A\_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.

#### **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<br>Signatur durch zentrale Komponente der TI-Plattform<br>Code-Signatur  |
| 2.1.1.3  | Identitäten für die Client-Server-Authentifizierung | Identitäten für den Aufbau von TLS-Verbindungen  | Fachdienst TLS – Server<br>Fachdienst TLS – Client<br>zentrale TI-Plattform TLS – Server<br>zentrale TI-Plattform TLS – Client<br>AUT-Identität der SMC-B<br>AUT-Identität des Kartenterminals<br>AUT-Identität des Anwendungskonnektors<br>AUT-Identität der SAK<br>AUT-Identität der eGK<br>AUTN-Identität der eGK<br>AUT-Identität des HBA |
| 2.1.1.4  |   | Identitäten für den Aufbau von IPsec-Verbindungen  | ID.NK.VPN<br>ID.VPNK.VPN  |
| 2.1.1.6  | Verschlüsselungs-zertifikate                        | Identitäten, für die medizinische Daten verschlüsselt werden                                       | ENC-Identität des Versicherten<br>ENCV-Identität der eGK des Versicherten<br>ENC-Identität des HBA<br>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“**

| <b>Anwendungsfall</b>   | <b>Vorgaben</b>  |
|---|--|
| Art und Kodierung des öffentlichen Schlüssels   | RSA (OID 1.2.840.113549.1.1.1)<br>zu verwendende Schlüssellänge: 2048 Bit, zulässig bis Ende 2025, vgl. auch A_15590                         |
| Signatur eines Zertifikats<br>Signatur einer OCSP-Response<br>Signatur eines OCSP-Responder-Zertifikats<br>Signatur einer CRL<br>Signatur des Zertifikats das Basis der Signaturprüfung einer CRL ist | sha256withRSAEncryption<br>(OID 1.2.840.113549.1.1.11)<br>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  | <p>ecPublicKey<br/>{OID 1.2.840.10045.2.1}<br/><b>Entweder</b><br/>auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1]<br/>zulässig bis Ende 2029+<br/><b>oder</b><br/>auf der Kurve P-256 [FIPS-186-45]<br/>zulässig bis Ende 2029+</p> <p>Verständnishinweis: vgl. auch A_23139 bezüglich der Entweder-Oder-Beziehung</p> <p>Die Kodierung des öffentlichen Punkt erfolgt nach [RFC5480, Abschnitt 2], vgl. Beispiel in Abschnitt 5.2)</p> <p>Der privater Schlüssel muss zufällig und gleichverteilt aus <math>\{1, \dots, q-1\}</math> gewählt werden. (<math>q</math> ist die Ordnung des Basispunkts und <math>\text{ceil}(\log_2 q)=256</math> ).</p> |
| Signatur eines Zertifikats<br>Signatur einer OCSP-Response<br>Signatur eines OCSP-Responder-Zertifikates<br>Signatur einer CRL<br>Signatur des Zertifikats das Basis der Signaturprüfung einer CRL ist | <p>ecdsa-with-SHA256 [RFC-3279] {OID 1.2.840.10045.4.3.2}<br/><b>Entweder</b><br/>auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1]<br/>zulässig bis Ende 2029+<br/><b>oder</b><br/>auf der Kurve P-256 [FIPS-186-45]<br/>zulässig bis Ende 2029+</p> <p>vgl. Beispiel in Abschnitt 5.2</p> <p>Der privater Schlüssel muss zufällig und gleichverteilt aus <math>\{1, \dots, q-1\}</math> gewählt werden. (<math>q</math> ist die Ordnung des Basispunkts und <math>\text{ceil}(\log_2 q)=256</math> ).</p>  |

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

Für die maximale Gültigkeitsdauer der Zertifikate gilt die Anforderung [GS-A\_3080].

**A\_22220 - Konnektor: zulässige Algorithmen und Domainparameter bei Zertifikatsprüfungen**

Ein Konnektor KANN bei einer Zertifikatsprüfung alle im SOGIS-Katalog [SOGIS-2020] 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. [ $\leq$ ]

**A\_19073 - Feste Laufzeit CV-Zertifikate einer Karte (eGK/HBA/SMC-B)**

Die Anbieter CVC-TSP eGK, Anbieter HBA und Anbieter SMC-B MÜSSEN CV-Zertifikate tagesgenau in der Laufzeit auf die am kürzest gültigen X.509-Zertifikate der "Schlüsselgeneration ECDSA" der Karte beschränken.

Sind keine X.509-Zertifikate der "Schlüsselgeneration ECDSA" auf der Karte vorgesehen, dann MUSS die Laufzeit auf die am kürzest gültigen X.509-Zertifikate der "Schlüsselgeneration RSA" der Karte beschränkt werden. [ $\leq$ ]

**A\_19173 - Feste Laufzeit X.509-Zertifikate einer Karte (eGK/HBA/SMC-B)**

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

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

**A\_23139 - TSP-X.509-nonQES: ECC-Kurvenparameter, Komplexitätsreduktion**

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

1. ein öffentlicher ECC-Schlüssel im CA-Zertifikat,
2. die öffentlichen ECC-Schlüssel der zum CA-Zertifikat aus (1) zugehörigen OCSP-Zertifikate (vgl. [RFC-6960#4.2.2.2] bzw. A\_23142), und
3. die öffentlichen ECC-EE-Schlüssel in den EE-Zertifikate, die durch die CA mit dem Schlüssel aus (1) prüfbar sind,

die gleichen Kurvenparameter (brainpoolP256r1, P-256 etc. vgl. [gemSpec\_Krypt#Tab\_KRYPT\_002a]) besitzen.

[ $\leq$ ]

Verständnishinweis:

Die Chipkarten der TI verwenden für ihre ECC-EE-Schlüssel alle die Kurvenparameter brainpoolP256r1. Dies ist in den Objektsystem-Spezifikationen (und damit auch den Objektsystemen) der Chipkarten fixiert. Die CA-en, die EE-Zertifikate für diese Chipkarten bestätigen, müssen nach A\_23139-\* ebenfalls ein ECC-Schlüsselpaar auf Basis von brainpoolP256r1 verwenden.

Die Komponenten-PKI der TI besitzt mehrere CA-Zertifikate. Es gibt mindestens ein CA-Zertifikat, das für die Prüfung der ECC-EE-Zertifikate von SMC-K, SMC-KT und der



meisten Fachdienste verwendet wird. Dieses CA-Zertifikat verwendet ebenfalls als öffentlichen Prüfschlüssel ein Schlüssel auf brainpoolP256r1-Basis (A\_23139-\*).

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

| <b>Anwendungsfälle</b>       | <b>Vorgaben</b>  |
|------------------------------|--|
| 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.<br>Eine auswertende Komponente muss mit beliebigen (also auch nicht-RSA basierten) Signaturen eines VDA-Zertifikats umgehen können (bspw. Signatur des VDA-Zertifikats nicht auswerten, Authentizität und Integrität des Zertifikats wird über die Vertrauensliste sichergestellt). |

| Anwendungsfälle  | Vorgaben   |
|--|--|
| <p>Art und Kodierung des öffentlichen EE-Schlüssels</p>  | <p><u>RSA-Signaturvariante:</u><br/> <b>Entweder</b><br/> OID 1.2.840.113549.1.1.1 (rsaEncryption)<br/> (zulässig bis gemäß [SOG-IS-2020])<br/> <b>oder</b><br/> OID 1.2.840.113549.1.1.10 (id-RSASSA-PSS) [RFC-5756].<br/> (zulässig bis gemäß [SOG-IS-2020])</p> <p>Die Auswahl obliegt dem EE-Zertifikatsausgebenden VDA.</p> <p><u>RSA-Schlüssellänge:</u><br/> zu verwendende Schlüssellänge: 2048 Bit, zulässig bis vgl. Angabe in [SOG-IS-2020]</p>   |
| <p>Signatur eines Zertifikats,<br/> Signatur einer OCSP-Response<br/> oder<br/> Signatur eines OCSP-Responder-Zertifikates</p> | <p><b>Entweder</b><br/> sha256withRSAEncryption (OID 1.2.840.113549.1.1.11)<br/> (zulässig bis gemäß [SOG-IS-2020])<br/> <b>oder</b><br/> id-RSASSA-PSS (1.2.840.113549.1.1.10) [RFC-5756]<br/> (zulässig bis gemäß [SOG-IS-2020])</p> <p>zu verwendende Schlüssellänge: 2048 Bit, zulässig bis vgl. Angabe in [SOG-IS-2020]</p> <p>Die Hashfunktion für die Hashwertberechnung der TBSCertificate-Datenstruktur MUSS eine nach [SOG-IS-2020] zulässige Hashfunktion („Agreed Hash Function“) sein. Als Hashfunktion SOLL SHA-256 [FIPS-180-4] verwendet werden.<br/> 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.)<br/> Die Saltlänge MUSS mindestens 256 Bit betragen.(Die Maximallänge des Salts ergibt sich nach [PKCS#1] in Abhängigkeit von der Länge des Moduls.)</p> |

**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.<br>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}<br>auf der Kurve brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] (zulässig bis gemäß [SOG-IS-2020])<br><br>Die Kodierung des öffentlichen Punkt erfolgt nach [RFC5480, Abschnitt 2], vgl. Beispiel in Abschnitt 5.2). Der private Schlüssel muss zufällig und gleichverteilt aus $\{1, \dots, q-1\}$ gewählt werden. ( $q$ ist die Ordnung des Basispunkts und $\text{ceil}(\log_2 q)=256$ ).  |
| Signatur eines Zertifikats,<br>Signatur einer OCSP-Response oder<br>Signatur eines OCSP-Responder-Zertifikates | ecdsa-with-SHA256 [RFC-3279] {OID 1.2.840.10045.4.3.2}<br>auf Kurve der brainpoolP256r1 [RFC-5639#3.4, brainpoolP256r1] (zulässig bis gemäß [SOG-IS-2020])<br><br>vgl. Beispiel in Abschnitt 5.2   |

### 2.1.1.3 TLS-Authentifizierung

#### **GS-A\_4359-02 - X.509-Identitäten für die Durchführung einer TLS-Authentifizierung**

Alle Produkttypen, die X.509-Identitäten für eine TLS-Authentifizierung verwenden, MÜSSEN alle in Tab\_KRYPT\_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.

Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab\_KRYPT\_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [ $\leq$ ]

#### **A\_22457 - TLS-Clients, Ciphersuiten bei TLS-Verbindung mit eHealth-KT**

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

#### **2.1.1.4 IPsec-Authentifizierung**

##### **GS-A\_4360-01 - X.509-Identitäten für die Durchführung der IPsec-Authentifizierung**

Alle Produkttypen, die X.509-Identitäten für eine IPsec-Authentifizierung verwenden, MÜSSEN alle in Tab\_KRYPT\_002 aufgeführten Algorithmen unterstützen und die Tabellenanforderungen erfüllen.

Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab\_KRYPT\_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [≤]

#### **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 unterstützen und die Tabellenanforderungen erfüllen.

Produkttypen die Zertifikate (X.509-Identitäten) auf Basis der Schlüsselgeneration „ECDSA“ ausstellen (vgl. Abschnitt 5.1) oder verwenden, MÜSSEN die in Tab\_KRYPT\_002a aufgeführten Algorithmen und die Tabellenvorgaben erfüllen. [≤]

#### **2.1.2 CV-Identitäten**

CV-Identitäten werden für die Authentifizierung zwischen Karten verwendet.

##### **2.1.2.1 CV-Zertifikate G2**

###### **GS-A\_4365-02 - 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<br>bestätigtes<br>Schlüsselpaar | <b>Authentisierung ohne<br/>Sessionkey-Aushandlung</b><br>[RFC-5639#3.4, brainpoolP256r1]<br>ecdsa-with-SHA256<br>{OID 1.2.840.10045.4.3.2}<br><br><b>Authentisierung mit<br/>Sessionkey-Aushandlung</b><br>[RFC-5639#3.4, brainpoolP256r1]<br>authS_gemSpec-COS-G2_ecc-with-<br>sha256<br>{OID 1.3.36.3.5.3.1} | 256 Bit bis Ende 2029+ |
| Signatur des<br>Endnutzerzertifikats                | [RFC-5639#3.4, brainpoolP256r1]<br>ecdsa-with-SHA256<br>{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 die Tab\_KRYPT\_007 aufgeführten Algorithmen verwenden und die Tabellenanforderungen erfüllen.

[<=]

**Tabelle 7: Tab\_KRYPT\_007 Algorithmen für CV-CA-Zertifikate**

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

### **GS-A\_4368 - Schlüsselerzeugung**

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

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

**GS-A\_5021 - Schlüsselerzeugung bei einer Schlüsselspeicherpersonalisierung**

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

[<=]

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

**GS-A\_5338 - HBA/SMC-B – Erzeugung asymmetrischer Schlüsselpaare auf der jeweiligen Karte selbst**

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

[<=]

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

**GS-A\_5386 - kartenindividuelle geheime und private Schlüssel G2-Karten**

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

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

1. das verwendete Schlüsselableitungsverfahren (KDF) unumkehrbar und nicht-vorhersagbar ist (Hilfestellung: Beispiele in [gemSpec\_Krypt, 2.4 und 3.4]).
2. der Masterkey (Key Derivation Key (KDK)) GS-A\_4368 erfüllt (insbesondere Entropie-Vorgaben). Der KDK MUSS eine Mindestentropie von 120 Bit besitzen.

[<=]

Für private Schlüssel bei HBA und SMC-B wird die kartenindividuelle Erzeugung und Personalisierung durch GS-A\_5338 technisch sichergestellt. Je nach verwendetem COS, insbesondere dessen spezifischen Personalisierungsverfahrens, kann es sein, dass ein Kartenherausgeber symmetrische Schlüssel aus technischen Gründen personalisieren muss, obwohl er später nicht plant mit diesen Schlüsseln bspw. im Rahmen eines CMS zu arbeiten. Es ist sicherheitskritisch, dass auch diese symmetrischen Schlüssel ebenfalls die Anforderungen GS-A\_5021 bzw. GS-A\_4368 erfüllen.

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

- alle Verfahren aus [NIST-SP-800-108] mittels CMAC [NIST-SP-800-38B],



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

#### **A\_23900 - TSP-X.509: Einzigartigkeit der bestätigten öffentlichen Schlüssel**

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

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

Erläuterung zu A\_23900:

Die Einzigartigkeit der öffentlichen EE-Schlüssel ist nach A\_23900 nur TSP-intern sicherzustellen, es wird also nicht verlangt, dass die Schlüsselspeicher i. S. v. A\_23900 über alle TSP ausgetauscht oder synchronisiert werden. A\_23900 stellt eine Detaillierung von [gemRL\_TSL\_SP\_CP#GS-A\_4906] dar.

### **2.4.1 Prüfung auf angreifbare (schwache) Schlüssel**

#### **A\_17294 - TSP-X.509: Prüfung auf angreifbare (schwache) Schlüssel**

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

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

Mindestumfang der Prüfung MÜSSEN

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

Der TSP MUSS den Mindestumfang der Prüfung bei Bekanntwerden neuer Angriffsmöglichkeiten gemäß [gemSpec\_DS\_Anbieter#GS-A\_5560] erweitern. [ $\leq$ ]

TSPs, die im Internet TLS-Zertifikate ausgeben (bspw. für die Verwendung von HTTPS), müssen aufgrund der Baseline Requirement des CA/Browser Forums ( <https://cabforum.org/baseline-requirements-documents/> ) vor der Zertifikatserzeugung kryptographische Prüfungen des zu bestätigenden öffentlichen Schlüssels durchführen. Analog gilt dies mit A\_17294 auch für TI-TSPs. Die gematik stellt auf Anfrage eine Beispielimplementierung für die Tests des Mindestumfangs bereit.

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



## **2.4.2 ECC-Schlüssel in X.509-Zertifikaten**

### **GS-A\_5518 - Prüfung Kurvenpunkte bei einer Zertifikatserstellung**

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.

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

### **A\_17092 - 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-03116-1#3.2 RSA])?  
Falls nein, ist das Ergebnis FAIL.
2. Ist der Modulus des untersuchten RSA-Schlüssels kleiner als  $2^{2048}$ ?  
Falls nein, ist das Ergebnis FAIL.
3. Ist der Modulus des untersuchten RSA-Schlüssels relativ prim zu allen Primzahlen kleiner als 100?  
Falls nein, ist das Ergebnis FAIL.

[<=]

Erläuterungen zu A\_17092 befinden sich in Abschnitt 9.2.

### **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 (im Sinne von [gemSpec\_Krypt#9.2 ], entropy()-Funktion) kleiner als 6.72? Falls ja, so ist das Ergebnis FAIL.

[<=]

Erläuterungen zu A\_17093 befinden sich in Abschnitt 9.2.

## **2.5 Einmalpasswörter**

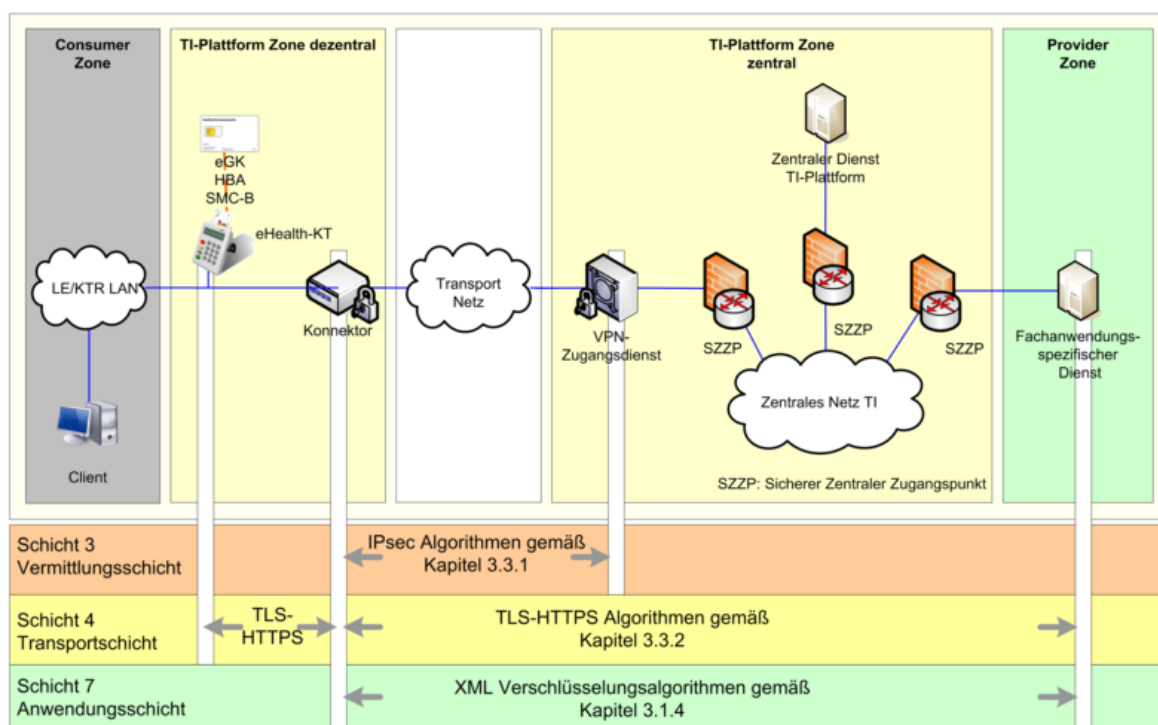
Bei verschiedenen Registrierungs- oder Anmeldeprozessen werden Einmalpasswörter (insbesondere bei KIM) verwendet. Für diese Einmalpasswörter gilt folgende Vorgabe:

### **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. [ $\leq$ ]

### 3 Konkretisierung der Algorithmen für spezifische Einsatzszenarien

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



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

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

#### 3.1 Kryptographische Algorithmen für XML-Dokumente

##### GS-A\_4370 - Kryptographische Algorithmen für XML-Dokumente

Alle Produkttypen, die XML-Dokumente

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

[<=]

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

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

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

**Tabelle 8: Tab\_KRYPT\_008 Beispiele für solche Algorithmen-URIs**

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

### **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 9: Tab\_KRYPT\_009 Algorithmen für die Erzeugung von nicht-qualifizierten elektronischen XML-Signaturen**

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

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

### 3.1.2 XML-Signaturen für qualifizierte elektronische Signaturen

#### GS-A\_4372-02 - XML-Signaturen für qualifizierte elektronische Signaturen

Alle Produkttypen, die XML-Signaturen für qualifizierte elektronische Signaturen erzeugen oder prüfen, MÜSSEN die Vorgaben der Tabelle Tab\_KRYPT\_010 erfüllen. [ <= ]

**Tabelle 10: Tab\_KRYPT\_010 Algorithmen für qualifizierte XML-Signaturen**

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

| <b>Signaturbestandteil</b>                 | <b>Beschreibung</b>  | <b>Algorithmus</b>   | <b>Anmerkung</b>  |
|--|--|--|---|
| kryptographische<br>s<br>Signaturverfahren | Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel | RSASSA-PSS mit SHA256 bis gemäß [SOGIS-2020]<br><br>ECDSA mit SHA-256 bis gemäß [SOGIS-2020]<br><br>(Hinweis: siehe Abschnitt 4.1)                                 | Der Algorithmus muss für alle qualifizierten Signaturen verwendet werden.<br><br>Alle hier aufgeführten Signaturverfahren müssen von einer Signaturprüfenden Komponente überprüfbar sein. |
| DigestMethod                               | Methode zur Berechnung eines Digest der zu signierenden Bereiche   | SHA-256<br>Die [XMLDSig] konforme Bezeichnung lautet:<br><a href="http://www.w3.org/2001/04/xmldsig-core#sha256">http://www.w3.org/2001/04/xmldsig-core#sha256</a> | Der Algorithmus muss für alle qualifizierten Signaturen verwendet werden.   |
| Kryptographische 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.

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

#### GS-A\_4381-01 - Schlüssellängen Algorithmus AES

Alle Produkttypen, die den Algorithmus AES nutzen, MÜSSEN die Schlüssellängen gemäß Tabelle Tab\_KRYPT\_012 nutzen. [<=]

**Tabelle 11: Tab\_KRYPT\_012 Algorithmen für Card-to-Server-Authentifizierung**

| Algorithmen Typ   | Algorithmus                                 | Schlüssellänge                        |
|---|---|---------------------------------------|
| Authentifizierung und Verschlüsselung der Authentisierungsdaten | AES im CBC-Modus (OID 2.16.840.1.101.3.4.1) | 128 Bit<br>zulässig bis<br>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 Sicherheitgateway Bestandsnetze und somit die in einem Zertifikat angegebene Gültigkeitsdauer DARF NICHT 5 Jahre überschreiten.

[<=]

### **3.3.2 TLS-Verbindungen**

#### **GS-A\_4385 - TLS-Verbindungen, Version 1.2**

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 Konfigurationen (Handshake-Modi, (EC)DH-Gruppen, Signaturverfahren, Ciphersuiten etc.) verwenden, und
2. mindestens die Ciphersuite "~~TSL~~**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 Einwegeneigenschaft 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 unterstützten Hashfunktionen kodiert. Dort muss also nach A\_21275-\* mindestens SHA-256 enthalten sein. Bei TLS 1.2 wird von fast allen TLS-Bibliotheken ebenfalls SHA-1 angegeben, dieses Verhalten lässt sich im Normalfall nicht ohne Code-Änderungen in den Bibliotheken verändern -- dieses Verhalten widerspricht zunächst A\_21275-\*. Das bloße Aufführen von SHA-1 als grundsätzlich unterstützte Hashfunktion soll nicht als fehlerhaftes Verhalten gelten. Wichtig für die Umsetzung von A\_21275-\* sind die tatsächlich erstellten Signaturen und die Prüfung dieser Signaturen.

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

#### **GS-A\_4384-03GS-A\_4384-01 - 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-186-45] 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   | <a href="http://oid-info.com/get/1.2.840.10045.3.1.7">http://oid-info.com/get/1.2.840.10045.3.1.7</a>                 |
| prime256v1   | [RFC-3279],<br>openssl ecparam -list_curves   |
| secp256r1    | [RFC-5480], <a href="http://www.secg.org/collateral/sec2_final.pdf">http://www.secg.org/collateral/sec2_final.pdf</a> |
| P-256        | [FIPS186-45]  |

Analog P-384 [FIPS186-45]:

|              |   |
|--------------|---|
| ansix9p384r1 | [ANSI-X9.62#L.6.5.2]  |
| ansip384r1   | <a href="http://oid-info.com/get/1.3.132.0.34">http://oid-info.com/get/1.3.132.0.34</a>                               |
| prime384v1   | [RFC-3279],<br>openssl ecparam -list_curves   |
| secp384r1    | [RFC-5480], <a href="http://www.secg.org/collateral/sec2_final.pdf">http://www.secg.org/collateral/sec2_final.pdf</a> |
| P-384        | [FIPS186-45]  |

### **GS-A\_5541 - TLS-Verbindungen als TLS-Klient zur Störungssampel oder SM**

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

(1) auf die explizite Prüfung, dass der TLS-Server die (EC)DH-Gruppe für den ephemeren (EC)DH-Schlüsselaustausch spezifikationskonform gewählt hat (vgl. GS-A\_4384-\* und A\_17124-\* Punkt 4), verzichten,

und

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

[<=]

### **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-Clients 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.stempel.telematik:8443/I_Monitoring_Message
```

II.

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

### **GS-A\_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 Schlüsselmateriale über einen authentisierten Diffie-Hellman-Schlüsselaustausch gebildet wird, welches das alte Material ersetzt, wobei das alte Material sowohl im Klienten als auch im Server sicher gelöscht wird. Insbesondere bei der Nutzung von TLS-Resumption (vgl. [RFC-5246, S. 36] oder [RFC-5077]) kann die Dauer einer TLS-Session deutlich länger sein als die Lebensdauer der TCP-Verbindung innerhalb welcher der initiale Schlüsselaustausch stattgefunden hat. Aus diesem Grunde werden analog zu den IPsec-Vorgaben (vgl. [gemSpec\_Krypt#GS-A\_4383]) Vorgaben für die maximale Gültigkeitsdauer dieses Schlüsselmateriale gemacht (vgl. auch [SDH-2016]).

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



Schlüsselmateriale, dass bei der Erzeugung des SessionTickets (für die Sicherung von Vertraulichkeit und Authentizität der SessionTickets) verwendet wird, MUSS spätestens alle 48 Stunden gewechselt werden und das alte Materiale 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 Schlüsselmateriale 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](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](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-2020] und [BSI-TR-03116-1]) 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\_5345-04GS-A\_5345-01 - 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-186-45] 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.

#### **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 "3.15.43- ePA-spezifische TLS-Vorgaben " gibt es weitere TLS-Vorgaben.

### **3.3.3 DNSSEC-Kontext**

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

### **GS-A\_4388 - DNSSEC-Kontext**

Alle Produkttypen, die DNSSEC verwenden, MÜSSEN die Algorithmen und Vorgaben gemäß Tabelle Tab\_KRYPT\_017 erfüllen.

[<=]

**Tabelle 12: Tab\_KRYPT\_017 Algorithmen für DNSSEC**

| <b>Algorithmen Typ</b>   | <b>Algorithmus</b>     | <b>Schlüssellänge</b> |
|--|------------------------|-----------------------|
| 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<br>Asymmetrische Schlüssel zur Wahrung der Authentizität und Integrität von Zonendatenobjekten.   | RSA-SHA-256 [RFC-5702] | 2048 Bit              |
| DNSSEC KSK<br>Asymmetrische Schlüssel zur Wahrung der Authentizität und Integrität von Zonendatenobjekten.   | RSA-SHA-256 [RFC-5702] | 2048 Bit              |

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

## **3.4 Masterkey-Verfahren (informativ)**

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

- Der Ableitungsprozess muss unumkehrbar und nicht-vorhersehbar sein, um sicherzustellen, dass die Kompromittierung eines abgeleiteten Schlüssels nicht den Ableitungsschlüssel oder andere abgeleitete Schlüssel kompromittiert.
- Bei einer Schlüsselableitung (im Sinne von [ISO-11770]) basiert die kryptographische Stärke der abgeleiteten Schlüssel auf der Ableitungsfunktion und der kryptographischen Stärke des geheimen Ableitungsschlüssels (insbesondere hier dessen Entropie). Die Entropie der abgeleiteten Schlüssel ist kleiner gleich der Entropie des geheimen Ableitungsschlüssels. Um die Entropie der abgeleiteten Schlüssel sicherzustellen, muss die Entropie des geheimen Ableitungsschlüssels (deutlich) größer sein als die zu erreichende Entropie der abgeleiteten Schlüssel.

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

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

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

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

**Tabelle 13: Tab\_KRYPT\_018 Ablauf zur Berechnung eines versichertenindividuellen Schlüssels**

| Reihenfolge | Beschreibung  | Formale Darstellung   |
|-------------|---|---|
| 1           | Bildung eines Hashwertes über dem versichertenindividuellen Merkmal unter Verwendung eines statischen Padding-Verfahrens für den Fall, dass das versichertenindividuelle Merkmal in seiner Länge nicht der Blocklänge des Hash-Algorithmus entspricht.<br>Im Ergebnis wird ein versichertenindividuelles Merkmal geeigneter Länge für den nächsten Schritt erzeugt. | $\text{HASH\#1} = \text{SHA-256}(\text{versichertenindividuelles Merkmal})$ |
| 2           | AES-Verschlüsselung des Resultats mit dem Masterkey.<br>Durch die Verschlüsselung an dieser Stelle ist sichergestellt, dass der versichertenindividuelle Schlüssel nur durch den Besitzer des geheimen Masterkeys erzeugt werden kann.  | $\text{ENC\#1} = \text{AES-256}(\text{HASH\#1})$                            |

| Reihenfolge | Beschreibung  | Formale Darstellung                                     |
|-------------|---|---|
| 3           | Bildung eines Hashwertes über dem Ergebnis des vorherigen Verarbeitungsschritts.<br>Dies stellt sicher, dass ein Schlüssel geeigneter Länge erzeugt wird. | Versichertenindividueller Schlüssel =<br>SHA-256(ENC#1) |

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

**Tabelle 14: Tab\_KRYPT\_019 eingesetzte Algorithmen für die Ableitung eines versichertenindividuellen Schlüssels**

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

## 3.5 Hybride Verschlüsselung binärer Daten

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

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

### 3.5.1 Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten

#### **GS-A\_4389 - Symmetrischer Anteil der hybriden Verschlüsselung binärer Daten**

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

- 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].*

### **3.5.2 Asymmetrischer Anteil der hybriden Verschlüsselung binärer Daten**

#### **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.

[<=]

### **3.6 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.[<=]

### 3.7 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 15: Tab\_KRYPT\_020 Algorithmen für die Erzeugung und Prüfung von binären Daten im Kontext von Dokumentensignaturen**

| Signaturbestandteil                 | Beschreibung   | Algorithmus   | Anmerkung   |
|-------------------------------------|--|---|---|
| Signaturstandard                    | Signaturstandard   | ETSI TS 101 733 V1.7.4 (2008-07)<br>Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAvES)<br>[ETSI-CAvES] | Die Verwendung des Standards ist für die Signatur von Dokumenten verpflichtend die mittels CMS [RFC-5652] erzeugt werden.   |
| kryptographisches Signaturverfahren | Algorithmus für die Berechnung des Nachrichten-Digest (Hashwert des Dokuments) und die Signatur mit dem privaten Schlüssel | RSASSA-PSS mit SHA-256<br>bis Ende 2025<br><br>ECDSA mit SHA-256<br>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<br>2.1.1.1<br>2.1.1.2  | Die Auswahl des kryptographischen Tokens ist von dem jeweiligen Einsatzzweck abhängig.  |



### 3.8 Signaturen innerhalb von PDF/A-Dokumenten

#### GS-A\_5081-01 - 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 16: Tab\_KRYPT\_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-Dokumentensignaturen**

| Signaturbestandteil                        | Beschreibung   | Algorithmus  | Anmerkung   |
|--|--|--|---|
| <b>Signaturstandard</b>                    | 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.               |
| <b>kryptographisches Signaturverfahren</b> | 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<br><br>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. |
| <b>DigestMethod</b>                        | Methode zur Berechnung eines Digest der zu signierenden Bereiche   | SHA-256  | Die Verwendung des Algorithmus ist verpflichtend.   |

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

### 3.9 Kartenpersonalisierung

Vgl. auch Abschnitt 2.4 [\(– Schlüsselerzeugung\) 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.

[<=]

### 3.10 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.[<=]

### 3.11 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.  
[<=]

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

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

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

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

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

Bei der Wahl der Hashfunktion kann er sich nur darauf verlassen, dass der OCSP-Responder als Hashalgorithmus (vgl. „hashAlgorithm“-Datenfeld) SHA-1 [FIPS-180-4] unterstützt. Für den Anfragenden und den OCSP-Responder gilt dementsprechend GS-A\_5131. Er muss SHA-1 für die CertID-Struktur verwenden. Ein OCSP-Responder, der zusätzlich weitere Hashfunktionen unterstützt, muss nichts zurückbauen – er darf auch so in der TI arbeiten.

Warum ist der Einsatz von SHA-1 an dieser Stelle kryptographisch gesehen ausreichend? Da (1) ein OCSP-Responder der TI nicht für beliebige CAs arbeitet (Wahl von DN und öffentlichen Schlüssel ist damit beschränkt) und (2) i. d. R. die CertHash-Extension Teil der OCSP-Antwort ist und innerhalb der CertHash-Extension in der TI eine kryptographisch hochwertige Hashfunktion verwendet wird, ist die Verwendung von SHA-1 hier aus 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 OSCP-Anfrage erzeugt der OSCP-Responder u.  
a. eine CertHash-Datenstruktur:

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

Hierfür muss eine kryptographisch hochwertige (nach [BSI-TR-03116-1] zulässige) Hashfunktion verwendet werden. Normativ ist an dieser Stelle: „GS-A\_4393 Algorithmus bei der Erstellung von Hashwerten von Zertifikaten oder öffentlichen Schlüsseln“. Spätestens an dieser Stelle können OSCP-Client und OSCP-Server sich sicher sein, ob sie über das gleiche Zertifikat sprechen.

**Dritte Stelle:** Die OSCP-Response muss am Ende vom OSCP-Responder signiert werden. Dafür ist die Vorgabe aus Tab\_KRYPT\_002 „Signatur der OSCP-Response“ normativ, welche über die für die jeweiligen Zertifikate geltenden Anforderungen (bspw. GS-A\_4357-0102) angezogen werden.

### 3.12 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<sub>p</sub>) (vgl. Technische Richtlinie 03111, Version 2.0) auf der Kurve P256r1 [RFC-5639].

[<=]

### 3.13 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. [ $\leq$ ]

### **3.14 Spezielle Anwendungen von kryptographischen Signaturen**

#### **GS-A\_5207 - Signaturverfahren beim initialen Pairing zwischen Konnektor und eHealth-Kartenterminal**

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

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

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

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

#### **GS-A\_5208 - Signaturverfahren für externe Authentisierung**

Der Konnektor MUSS an der Schnittstelle für die externe Authentisierung die Signaturverfahren RSASSA-PKCS1-v1\_5 [PKCS#1] und RSASSA-PSS [PKCS#1] anbieten. [ $\leq$ ]

Erläuterung: Der Konnektor erlaubt (bei entsprechender Berechtigung) die direkte Nutzung der privaten Schlüssel MF/ DF.ESIGN/ PrK.HP.AUT.\* auf einem HBA oder MF/ DF.ESIGN/ PrK.HCI.AUT.\* auf einer SMC-B durch ein Primärsystem. Dies wird fast immer für eine klientenseitige TLS-Authentisierung gegenüber einem TLS-Server (außerhalb der TI) verwendet. Dafür werden über die Schnittstelle RSASSA-PKCS1-v1\_5-Signaturen von den entsprechenden Karten erzeugt und über den Konnektor an ein Primärsystem übergeben. Für unbenannte Anwendungen müssen auch RSASSA-PSS-Signaturen erzeugbar sein. Diese Signaturen sind nicht als Dokumentensignaturen verwendbar, der Verwendungszweck ist in den zu den privaten Schlüsseln gehörigen Zertifikaten kodiert (ExtendedKeyUsage: keyPurposeId = id-kp-clientAuth).

Hinweis: GS-A\_5208 ist nicht dem PTV4-Konnektor zugewiesen, sondern die erweiterten Anforderungen A\_17209.

#### **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-Identifizier „http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1“ nach [RFC-6931, Abschnitt „2.3.10 RSASSA-PSS Without Parameters“]. [ $\leq$ ]

## 3.15 ePA-spezifische Vorgaben

### 3.15.1 Verbindung zur VAU

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

#### **A\_15549-01 - ePA-VAU-Client: Kommunikation zwischen ePA-Client und ePA-VAU**

~~A\_15546 - ePA-Frontend des Versicherten: Kommunikation zwischen ePA-FdV und VAU~~ Ein ePA-Client ~~Das ePA-Frontend des Versicherten~~ MUSS bei der Kommunikation mit der VAU das Kommunikationsprotokoll aus [gemSpec\_Krypt#Abschnitt "Kommunikationsprotokoll zwischen VAU und Protokoll für ePA-Clients für alle"] verwenden und dabei die Rolle Client einnehmen. Dabei MUSS es die CipherConfiguration "AES-256-GCM-BrainpoolP256r1-SHA-256" (vgl. Abschnitt 6) verwenden. ~~Das ePA-Frontend des Versicherten.~~

Der Client einer VAU MUSS nach spätestens 24 Stunden das Aushandeln eines neuen AES-Sitzungsschlüssels erzwingen. ~~Es MUSS den abgelaufenen Sitzungsschlüssel~~ 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\_15549 - VAU-Client: Kommunikation zwischen VAU-Client und VAU~~**

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

#### **~~A\_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.~~ [<=]

#### **A\_15547-01 - ePA-VAU: Kommunikation zwischen ePA-VAU und ePA-Client**

~~A\_15547 - VAU: Kommunikation zwischen VAU und ePA-FdV bez. FM-ePA~~ Das ePA-Aktensystem MUSS sicherstellen, dass dessen VAU bei der Kommunikation mit dem ePA-Frontend des Versicherten oder dem FM-Client das Kommunikationsprotokoll aus [gemSpec\_Krypt#Abschnitt "VAU-Protokoll für ePA"] ~~das Kommunikationsprotokoll aus [gemSpec\_Krypt#Abschnitt "Kommunikationsprotokoll zwischen VAU und ePA-Clients für alle"] verwendet und dabei die Rolle Server einnimmt. Dabei MUSS es die CipherConfiguration "AES-256-GCM-BrainpoolP256r1-SHA-256" (vgl. Abschnitt 6)~~

verwenden.

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

Hinweis: Das AUT "-", das einen ECC-EE-Schlüssel der VAU bestätigt. Die VAU MUSS für die Erstellung der VAUHelloServerHello-Nachricht mit dem zugehörigen privaten EE-Schlüssel signieren (Signatur der VAUHelloServerData). In der VAUHelloServer-Nachricht MUSS die VAU das Zertifikat aufführen und die dazugehörige OCSP-Response.

[ $\leq$ ]

hat die VAU auch schon bei ePA 1.x und 2.x verwendet.

### **~~3.15.2 Vorgaben für ePA-Schlüssel und ePA-Chifftrate~~**

#### **~~A\_15705 — Vorgaben Aktenschlüssel (RecordKey) und Kontextschlüssel (ContextKey)~~**

Ein Client eines ePA-Aktensystems MUSS sicherstellen, dass

- ~~1. die von ihnen erzeugten Aktenschlüssel (RecordKey) und Kontextschlüssel (ContextKey) AES-Schlüssel [FIPS-197] mit 256-Bit-Schlüssellänge sind,~~
- ~~2. diese Schlüssel von ihnen ausschließlich mittels AES/GCM analog [gemSpec\_Krypt#GS-A\_4373] bzw. [gemSpec\_Krypt#GS-A\_4389] verwendet werden und~~
- ~~3. sie die Arbeit mit Aktenschlüssel (RecordKey) und Kontextschlüssel (ContextKey), die nicht Spiegelstrich 1. erfüllen, ablehnen.~~

### **~~3.15.3 [ $\leq$ ]~~**

#### **~~A\_18004 — Vorgaben für die Kodierung von Chiffraten (innerhalb von ePA)~~**

Ein Client eines ePA-Aktensystems MUSS Folgendes sicherstellen:

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

### **~~3.15.43.15.2 [ $\leq$ ]~~**

### **~~3.15.53.15.3 ePA-Aktensysteminterne Schlüssel~~**

#### **A\_15745-01 - Betreiberschlüssel Aktensystem**

**~~A\_15745 — Verschlüsselte Speicherung der verschlüsselten ePA-Daten~~** Ein ePA-Aktensystem MUSS sicherstellen, dass



1. es ~~einen betreiberspezifischen~~ zwei betreiberspezifische Schlüssel (BS / Masterkey für Daten und Masterkey für Befugnisse) gibt,  
~~1. dieser~~ diese Schlüssel ~~ein~~ AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge ~~ist~~,
2. ~~dieser~~ (vgl. A\_24645-\* bezüglich Speicherung der Schlüssel ~~in einem mindestens nach FIPS 140-2 Level 3 zertifizierten HSM liegt und nur dort verwendet wird~~) sind,
3. ~~dieser~~ diese Schlüssel im Betrieb ausschließlich der VAU des entsprechenden ePA-Aktensystem zugänglich ~~ist~~ sind,
4. ~~dieser~~ diese Schlüssel nur zur Schlüsselableitung nach einem in [gemSpec\_Krypt#Abschnitt 2.4] zulässigen Verfahren verwendet ~~wird~~ werden,
5. es eine Schlüsselableitung mit ~~diesem~~ diesen betreiberspezifischen ~~Schlüssel~~ Schlüsseln und einem aktenspezifischen Merkmal (bspw. der KVNR) gibt und daraus ~~ein aktenspezifischer Schlüssel (ABS)~~ versichertenindividuelle Persistierungsschlüssel für die Daten und für die Befugnisse abgeleitet ~~wird~~ werden,
6. ~~dieser aktenspezifische Schlüssel~~ ~~ein~~ diese versichertenindividuelle Persistierungsschlüssel AES-Schlüssel [FIPS-197] mit 256 Bit Schlüssellänge ~~ist~~ sind,  
~~1. die verschlüsselten ePA Daten einer Akte mit diesem aktenspezifischen Schlüssel verschlüsselt werden,~~  
~~1. die verschlüsselten ePA Daten außerhalb der VAU niemals im Klartext (also ohne mittels des ABS verschlüsselt zu sein) liegen,~~
7. ~~dieser Schlüssel (ABS)~~ diese Schlüssel ausschließlich mittels AES/GCM analog [gemSpec\_Krypt#GS-A\_4389] verwendet ~~wird~~ (~~der ABS wird durch Anfrage der VAU im HSM berechnet (Schlüsselableitung) und dann von dort an die VAU übermittelt, die AES/GCM Operationen mit dem ABS finden in der VAU statt~~); werden,
8. ~~dieser~~ diese Schlüssel ~~(ABS)~~ im Betrieb ausschließlich der VAU des entsprechenden ePA-Aktensystem zugänglich ~~ist~~ sind.

[<=]

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

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



#### A\_15746-01 - Sicherstellung der Verfügbarkeit der betreiberspezifischen Schlüssel

##### ~~A\_15746—Sicherstellung der Verfügbarkeit des betreiberspezifischen Schlüssels~~

Ein ePA-Aktensystem MUSS sicherstellen, dass für die Sicherstellung der Verfügbarkeit ~~des~~ betreiberspezifischen ~~Schlüssels~~ Schlüssel (vgl. A\_15745-\*) eine sicherheitstechnisch geeignete Sicherung (Backup) des Schlüsselmaterials erzeugt und sicher verwahrt wird.

[<=]

#### ~~A\_16176-01~~A\_16176 - Mindestvorgaben für ePA-Aktensystem-interne Schlüssel

Ein ePA-Aktensystem MUSS bei innerhalb des Aktensystems eingesetzten Schlüsselmaterial, das nicht aus der TI-PKI kommt (Signatur 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-186-45],
  - 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\_20519-01 - Wechsel der betreiberspezifischen Schlüssel

~~A\_20519—Wechsel des betreiberspezifischen Schlüssels~~Ein ePA-Aktensystem MUSS sicherstellen, dass

1. ~~der betreiberspezifische~~die betreiberspezifischen Schlüssel (BS) (vgl. A\_15745-\*) mindestens ~~alle 5 Jahre~~jährlich gewechselt ~~wird~~werden,
2. nach Erzeugung eines neuen BS alle auf Grundlage des alten BS erzeugten Chiffre umgeschlüsselt werden (~~neue Schlüsselableitung der aktenspezifischen Schlüssel (ABS) A\_15745#6, Umschlüsselung der chiffrierten Aktendaten~~),
3. anschließend ~~die alten~~alte Chiffre und der alte BS sicher gelöscht werden, und
4. die Schritte 2. und 3. spätestens 4 Wochen nach Schlüsselwechsel ~~des~~eines BS abgeschlossen sind.

[<=]

Hinweis: ~~Der betreiberspezifische~~Die betreiberspezifischen Schlüssel (BS) ~~darf~~dürfen gemäß A\_15745-\* ausschließlich der VAU des ePA-Aktensystems zugänglich sein. Daher muss die Umschlüsselung in der VAU stattfinden. ~~Dabei ist \*keine\* Mitwirkung des Versicherten (oder eines von ihm berechtigten Nutzers) erforderlich.~~

### 3-15-63.15.4 ePA-spezifische TLS-Vorgaben

#### A\_15751-02 - TLS-Verbindung zwischen ePA-Aktensystem und ePA-Client

~~A\_15751 – TLS-Verbindung zwischen ePA-Aktensystem und ePA-FdV~~ Ein ePA-Aktensystem und ein ePA-~~Frontend des Versicherten~~Client 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-186-45] 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).

[<=]

#### 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 Versuchs 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\_15833-01A\_15833 - TLS-Verbindungen ePA-FdV

Ein ePA-Frontend des Versicherten MUSS die TLS-Vorgaben in A\_15751-\* bei allen seinen TLS-Verbindungen einhalten.

[<=]

Es gibt ePA-Aktensysteme deren "~~Zugangsgateway des Versicherten~~"Access Gateway [gemSpec\_Zugangsgateway\_VersAktensystem\_ePAfuerAlle] die Verwendung von TLS-Session-Resumption per Session-ID von einem ePA-FdV verlangen. Ebenfalls gibt es ePA-

Aktensysteme deren ~~Zugangsgateway des Versicherten~~ **Access Gateway** die Verwendung von TLS-Session-Resumption per Session-Tickets von einem ePA-FdV verlangen. Nach einem TLS-Verbindungsende muss ein ePA-FdV eine TLS-Session-Resumption verwenden, ansonsten muss der Versicherte sich neu authentisieren (erneuter Login).

Ein ePA-FdV muss, um die beiden Anwendungsfälle (1) Aktensystemwechsel und (2) Versicherten-Vertreter über ein ePA-FdV eines anderen Aktensystemherstellers zu ermöglichen, beide Varianten der TLS-Session-Resumption unterstützen. Die meisten TLS-Bibliotheken unterstützen beide Varianten per Default.

#### **~~A\_21269—ePA-FdV: TLS-Session-Resumption per Session-ID und per Session-Ticket~~**

~~Ein ePA-Frontend des Versicherten MUSS sowohl TLS-Session-Resumption per Session-ID als auch TLS-Session-Resumption per Session-Tickets unterstützen. Falls im ePA-FdV (Rolle TLS-Client) Session-Daten (Session-ID oder oder Session-Ticket—jeweils inkl. kryptographischer Daten (Premaster-Secret etc.)) vorliegen, so MUSS das ePA-FdV bei einem erneuten TLS-Verbindungsaufbau zum Zugangsgateway des Versicherten (Rolle TLS-Server) diese Session-Daten im Rahmen des TLS-Verbindungsaufbaus mit Session-Resumption verwenden.~~

~~Ein ePA-FdV (Rolle TLS-Client) MUSS falls ein Zugangsgateway des Versicherten (Rolle TLS-Server) im Rahmen eines TLS-Handshakes ein TLS-Session-Tickets sendet und der TLS-Handshake erfolgreich war, dieses Session-Ticket inkl. der für eine spätere TLS-Session-Resumption notwendigen kryptographischen Parameter (Premaster-Secret etc.) lokal speichern.~~

~~Ein ePA-FdV (Rolle TLS-Client) MUSS falls ein Zugangsgateway des Versicherten (Rolle TLS-Server) im Rahmen eines TLS-Handshakes kein TLS-Session-Tickets sendet und der TLS-Handshake erfolgreich war, alle für eine spätere TLS-Session-Resumption per Session-ID notwendigen Daten lokal speichern.~~

~~Ein ePA-FdV MUSS nach Ende der Aktensitzung (bspw. der Versicherte beendet die ePA-FdV-App) alle für eine TLS-Session-Resumption notwendigen Daten lokal löschen.~~

~~[<=>]~~

### **3.15.7 Schlüsselableitungsfunktionalität ePA**

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

#### **~~A\_17876—SGD: Schlüsselableitung der spezifischen Schlüssel~~**

~~Ein SGD-ePA MUSS folgende Vorgaben durchsetzen:~~

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

~~[<=>]~~

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

### **~~A\_17872—Ver- und Entschlüsselung der Akten- und Kontextschlüssel (Schlüsselableitungsfunktionalität ePA)~~**

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

1. 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.
2. Der IV MUSS dabei zufällig erzeugt werden (vgl. [NIST-SP-800-38D#S.25] und [BSI-TR-02102-1#S.24]).
3. Der IV MUSS eine Bitlänge von 96-Bit (12-Byte) besitzen.

#### **~~{<=>}~~**

Für die Ende-zu-Ende-verschlüsselte Datenübertragung zwischen Client und SGD-HSM wird ECIES (vgl. [SEC1-2009#5.1 Elliptic Curve Integrated Encryption Scheme], [TR-02102-1#3.3. ECIES-Verschlüsselungsverfahren] und Abschnitt 5.7...ECIES...) verwendet. Dabei besitzt der Empfänger einen elliptischen Kurvenpunkt (öffentlicher Schlüssel), dessen Authentizität der Sender prüfen kann. Dies wird erreicht, indem der Kurvenpunkt des Empfängers (entweder Client oder SGD-HSM) mittels der Langzeitidentität des Empfängers signiert ist. Der Sender erzeugt ein ephemeres ECDH-Schlüsselpaar. Mit diesem und dem Kurvenpunkt des Empfängers führt der Sender einen ECDH-Schlüsselaustausch durch. Aus dem berechneten ECDH-Geheimnis berechnet der Sender mittels einer HKDF auf Basis von SHA-256 einen AES-256-Bit-Schlüssel, der im Galois/Counter-Mode (GCM) verwendet wird (Authenticated Encryption). Damit verschlüsselt der Sender den Klartext und erhält ein AES-GCM-Chifftrat. Der Sender sendet seinen erzeugten ephemeren Kurvenpunkt und das AES-GCM-Chifftrat an den Empfänger. Damit ist der Nachrichtentransport Ende-zu-Ende-verschlüsselt zwischen Sender und Empfänger, jedoch nur einseitig authentisiert. Die beidseitige Authentisierung wird über einen Authentisierungstoken, den das SGD-HSM für einen Client erzeugt, erreicht (vgl. [gemSpec\_SGD\_ePA#Datenkanal zwischen Client und SGD (informativ)]). Für das ECIES-Verfahren gilt der kryptographische Sicherheitsbeweis aus [ABR-1999].

### **~~A\_17873—SGD, SGD-HSM-authentisiertes ECIES-Schlüsselpaar~~**

Ein SGD-ePA MUSS sicherstellen, dass die zwei Schlüsselpaare (vgl. [gemSpec\_SGD\_ePA#A\_17910 (S4)]) für den ECIES-Nachrichtenempfang durch das SGD-HSM auf Basis der Kurvenparameter brainpoolP256r1 [RFC-5639] gewählt werden. Für die Authentisierung der öffentlichen ECIES-Schlüssels (Signatur mit [gemSpec\_SGD\_ePA#A\_17910 (S1)]) und Kodierung nach [gemSpec\_SGD\_ePA#A\_17894]) MUSS ECDSA [BSI-TR-03111] verwendet werden.

#### **~~{<=>}~~**

### **~~A\_17874—SGD-Client, Client-authentisiertes ECIES-Schlüsselpaar~~**

Ein Client eines SGD-ePA MUSS für den Nachrichtenempfang mittels des ECIES-Verfahrens im Kontext der Schlüsselableitungsfunktionalität ePA bei den verwendeten ECC-Schlüsseln die Kurvenparameter brainpoolP256r1 [RFC-5639] verwenden. Für die Authentisierung des öffentlichen ECIES-Schlüssels des Clients (Signatur mit AUT-Identität des Nutzers des Clients gemäß [gemSpec\_SGD\_ePA#A\_17901]) MUSS ECDSA [BSI-TR-03111] verwendet werden.

#### **~~{<=>}~~**

In A\_17873 und in A\_17874 wird nicht aufgeführt, welche Hashfunktion im Rahmen der Signaturstellung und -prüfung zu verwenden ist. Dies wird mit folgender Anforderung nachgeholt:

#### ~~A\_19971—SGD und SGD-Client, Hashfunktion für Signaturerstellung und-~~ ~~prüfung~~

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

- ~~1. Signaturerstellung mit [gemSpec\_SGD\_ePA#A\_17910 (S1)] und Kodierung nach  
[gemSpec\_SGD\_ePA#A\_17894] (vgl. A\_17873), bzw. Prüfung dieser Signatur,  
und~~
- ~~2. Signaturerstellung für die Authentisierung der öffentlichen ECIES-Schlüssels des  
Clients (vgl. A\_17874), bzw. Prüfung dieser Signatur~~

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

#### ~~A\_17875—ECIES-verschlüsselter Nachrichtenversand zwischen SGD-Client und SGD-HSM~~

Ein SGD-ePA und ein Client eines SGD-ePA MÜSSEN folgende Vorgaben umsetzen:

- ~~1. Für den Ende-zu-Ende-verschlüsselten Datenaustausch zwischen SGD-Client und  
SGD-HSM MUSS das ECIES-Verfahren [SEC1-2009] verwendet werden.~~
- ~~2. Der ECDH-Schlüsselaustausch innerhalb von ECIES zwischen SGD-Client und  
SGD-HSM MUSS nach [NIST-800-56-A#5.7.1.2] (Hinweis: ist fachlich identisch zu  
[SEC1-2009#3.3.1]) durchgeführt werden.~~
- ~~3. Aus dem gemeinsamen ECDH-Geheimnis MUSS mit der HKDF nach [RFC-  
5869] auf Basis von SHA-256 ein AES-256-Bit-Schlüssel abgeleitet werden.~~
- ~~4. Dieser Schlüssel (siehe Punkt 3) MUSS mittels AES-GCM und den fachlichen  
Vorgaben für AES-GCM aus A\_17872 verwendet werden, um den symmetrisch  
Teil der ECIES-Verschlüsselung authentisiert zu ver- bzw. zu entschlüsseln.~~

~~Hinweis: die Kodierung der Chiffre wird in [gemSpec\_SGD\_ePA] festgelegt.~~

~~[<=]~~

#### ~~A\_21269-01 - ePA-Client: TLS-Session-Resumption~~

#### ~~A\_18023—SGD, Ableitungsschlüssel Authentisierungstoken~~

Ein SGD-ePA MUSS folgende Vorgaben umsetzen:

- ~~1. Die Ableitungsschlüssel für die Erstellung der Authentisierungstoken  
[gemSpec\_SGD\_ePA#A\_17910 (S5)] MÜSSEN eine Mindestentropie von 256-Bit  
besitzen.~~
- ~~1. Diese Ableitungsschlüssel MÜSSEN mit der HKDF nach [RFC-5869] auf Basis von  
SHA-256 verwendet werden.~~

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

### 3.16 E-Rezept-spezifische Vorgaben

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

#### ~~A\_21332-02A\_21332 - 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-186-45] 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 Abschnitt [7 – Kommunikationsprotokoll zwischen VAU-Protokoll für E-Rezept-VAU und E-Rezept-Clients](#) 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.

### **3.17 KOM-LE-spezifische Vorgaben**

Bei KOM-LE werden E-Mail-Anhänge , deren Gesamtgröße 15 MiB überschreitet, separat symmetrisch verschlüsselt und das Chiffre auf dem "Fachdienst Download-Server (KAS)" abgelegt. Das Chiffre erhält eine ID, die aus dem Hashwert des Chiffres 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 Chiffre auf dem Download-Server (KAS) werden automatisch nach einer bestimmten im FD festgelegten Zeit gelöscht.

#### **A\_19644 - Hashfunktion für Hashwert-Referenzen beim Fachdienst Download-Server (KAS)**

Ein KOM-LE-Client und der Fachdienst Download-Server (KAS) MÜSSEN 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 HMAC-Sicherung der Prüfziffer VSDM**

#### **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-Chiffre> || <16 Byte AuthenticationTag> (vgl. auch Tab\_KRYPT\_ERP und folgende die Beispielvechlüsselung).  
Die Koordinaten sind (wie üblich) vorne mit chr(0) zu paden solange bis sie eine Kodierungslänge von 32 Byte erreichen.

[<=]

Beispiel:

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

```
-----BEGIN CERTIFICATE-----
MIICWzCCAgKgAwIBAgIUUYVvKtweOLq4G7U28Gwwb3tAghcW8wCgYIKoZIzj0EAwIw
gYIx CzA JBgNVBAYTAkRFRMQ8wDQYDVQQIDAZCZXJsaW4x DzANBgNVBAcMBk Jlc m x p
b j e Q M A 4 G A 1 U E C g w H Z 2 V t Y X R p a z E Q M A 4 G A 1 U E C w w H Z 2 V t Y X R p a z E t M C s G A 1 U E A w w k
R S 1 S Z X p l c H Q t V k F V I E J l a X N w a W V s a W l w b G V t Z W 5 0 a W V y d W 5 n M B 4 X D T I y M T A w N D I x
N T g l O F o X D T I z M T A w N D I x N T g l O F o w g Y I x C z A J B g N V B A Y T A k R F M Q 8 w D Q Y D V Q Q I D A Z C
Z X J s a W 4 x D z A N B g N V B A c M B k J l c m x p b j e Q M A 4 G A 1 U E C g w H Z 2 V t Y X R p a z E Q M A 4 G A 1 U E
C w w H Z 2 V t Y X R p a z E t M C s G A 1 U E A w w k R S 1 S Z X p l c H Q t V k F V I E J l a X N w a W V s a W l w b G V t
Z W 5 0 a W V y d W 5 n M F o w F A Y H K o Z I z j 0 C A Q Y J K y Q D A w I I A Q E H A 0 I A B B 1 w l e U 9 e g l D 8 Q 7 j
7 H u T V y G 7 K J I R o J 6 R t 1 x 8 + G H v b 3 x e h t 3 f 5 R P e E T i + p y 3 t L K 5 R W h F e G O I K m n y d o B x J
m K 6 V c m i j U z B R M B 0 G A 1 U d D g Q W B B T y O v u l E 6 x m E M u / 6 K I h I / k Q J Y w 9 C T A f B g N V H S M E
G D A w G B T y O v u l E 6 x m E M u / 6 K I h I / k Q J Y w 9 C T A P B g N V H R M B a f 8 E B T A D A Q H / M A o G C C q G
S M 4 9 B A M C A 0 c A M E Q C I G w Y u A D P / A u l q 0 / Q I z J D k C g q A X l V v m r 9 2 U P b f c R C l v N A i A c
w 8 V E 8 x B t M x 9 g 5 s L K / H i D C s r Q 9 T N Z j w p b 5 r b 1 t 4 o z g g = =
```



-----END CERTIFICATE-----

und folgendes der dazugehörige private Schlüssel:

```
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQc=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHgCAQEEII1+MNpxZfd+cPjE1Z5DCHCnuGCQt5MA6U55yuaxoB8CoAsGCSskAwMC
CAEBB6FEA0IABB1wleU9eg1D8Q7j7HuTVyG7KJIROJ6Rt1x8+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

019dec56554203624c214ac5321c798b78281d3ffff4a0a4e187319ebbebba2ace40fe71913c  
6e02998cd3cd6f8268a0daf9f40eb1561541de9868d46eb913e3f3157d5ed6bd9d4aef792d4  
73be5fa2868dbcd228e5a603afdae2c69fc459e656513cff835934de5e2f159b772ebbd21d6  
dfef2a4960ad829b968fc974b

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

### **3.19 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 verzichtet 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".

---

## **4 Umsetzungsprobleme mit der TR-03116-1**

---

Das u. a. durch die TR-03116-1 [BSI-TR-03116-1] angestrebte Sicherheitsniveau soll persönliche medizinische Daten effektiv schützen. Dazu lehnt sie sich an die sehr starken kryptographischen Vorgaben für die qualifizierte elektronische Signatur [SOG-IS-2020] an. Einige Formate (bspw. XMLDSig) oder Implementierungen (bspw. Standard-Java-Bibliotheken) können einige Vorgaben von Hause aus nicht erfüllen.

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

### **4.1 XMLDSig und PKCS1-v2.1**

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

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

#### **GS-A\_5091 - Verwendung von RSASSA-PSS bei XMLDSig-Signaturen**

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

[<=]

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

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

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

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

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

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

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

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

### **4.3 XML Signature Wrapping und XML Encryption Wrapping**

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

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

### **4.4 Güte von Zufallszahlen**

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

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

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

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

---

## **5 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 Schlüsselmateriale 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.

### **5.1 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.

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\_4357-0102, Schlüsselgeneration „ECDSA“].

```
-----BEGIN CERTIFICATE-----
MIICajCCAg+gAwIBAgIBATAKBggqhkJOPQQDAjBtMQswCQYDVQQGEwJERTEVMBMG
A1UECgwMZ2VtYXRpayBHbWJIMTQwMgYDVQQLDCTaZW50cmFsZSBSb290LUNBIGNR
ciBUZWxlbWw0aWtpbmZyYXN0cnVrdHVyMREwDwYDVQQDDAhHRU0uUkNBMAEwOx
NjEyMDkwODQxNTZaFw0yNjEyMDcwODQxNTZaMG0xCzAJBgNVBAYTAkRFRMRUwEwYD
VQQKDAxnZW1hdGlrIEdtYkgxNDAYBgNVBASMK1plbnRyYWx1IFJvb3QtQ0EgZGVy
IFRlbGVtYXRpa2luZnJhc3RydWt0dXlxeTAPBgNVBAMMCEdFTS5SQ0EzMFowFAYH
KoZIZj0CAQYJKyQDAwIIAQEHA0IABDd0NFCa3LuCf3Ss163wznKqKN3FO+PxXqgC
OpsHIScdU2SplobAIJK7+e/emHiEe5DwnZC3rEGTVTggJYpY39WjgZ4wgZswHQYD
VR0OBBYEFBERSneTkJZDKt3uLzjddI870TMmMEIGCCsGAQUFBwEBBDYwNDAYBggr
BgEFBQcwAYYmaHR0cDovL29jc3Aucm9vdC1jYS50aS1kaWVuc3RlLmRlL29jc3Aw
DwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwFQYDVR0gBA4wDDAKBggq
ghQATASBIzAKBggqhkJOPQQDAgNJADBGAiEApQ6qGHTx97IsdzoWH9/W32yt4rk
udUis0xxGZ48YOUCIQCTQ4puol5YyIAZYk74mfid3JBOvMBV/XgPV2WpS/99yg==
-----END CERTIFICATE-----
```

Relativ am Anfang des Zertifikats befindet sich die OID gemäß GS-A\_4357-~~0102~~

```
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 der
Telematikinfrastruktur, CN = GEM.RCA3
```



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

### 5.3 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-Identifizier „<http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256>“ [XMLDSig]. Als Hashfunktion (Messagedigest) MUSS SHA-256 [FIPS-180-4] verwendet werden.  
[<=]

### 5.4 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\_17124-03A-17124-01 - 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-~~1~~-\*] 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-186-45] 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.

## **5.5 ECC-Unterstützung bei IPsec**

Das IKE-Protokoll [RFC-7296] wird verwendet um Schlüsselmaterial auszuhandeln für die folgende Verschlüsselung und Integritätssicherung der über IPsec geschützten IP-Pakete. 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\_KEY\_PAYLOAD-Nachricht, in der eine vom Responder unterstützte und präferierte kryptographische Gruppe angegeben ist [RFC-7296#Abschnitt 1.2]. Somit kommt es bei einem initialen Verbindungsaufbau zwischen 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\_KEY\_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\_22343-01 - 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 den nächsten regulären Verbindungsaufbau zum VPN-Konzentrator gemäß der Vorgaben aus A\_17125 durchführen.[<=]

### **A\_17210 - Konnektor, IKE-Schlüsselaushandlung Fallback (ECC-Migration)**

Ein Konnektor MUSS, falls beim IKE-Verbindungsaufbau klar wird, dass der IKE-Responder (VPN-Konzentrator, VPN-Zugangsdienst) (noch) keine ECC-Verfahren unterstützt (INVALID\_KEY\_PAYLOAD-Nachricht), auf die Vorgaben aus GS-A\_4382-\* "zurückfallen".

[<=]

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 ephemeres 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.  
[<=]

## **5.6 ECDSA-Signaturen**

### **5.6.1 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-Identifizier „ <http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256>“ [XMLDSig] verwenden. Als Hashfunktion (Messagedigest) MÜSSEN sie SHA-256 [FIPS-180-4] verwenden.

[<=]

Die Anforderung A\_17206 gilt für allgemeine XML-Datensignaturen, also auch für Tokensignaturen etc. A\_17360 fordert für die Interoperabilität bei der Prüfbarkeit von Dokumentensignaturen die Verwendung des interoperablen Containerformats nach [ETSI-XAdES].

#### **A\_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.

[<=]

### **5.6.2 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-CAAdES].

#### **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-CAAdES] (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.

[<=]

## **5.7 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 Chifftrat nicht entschlüsseln, auch wenn die Chipkarte den prinzipiell richtigen privaten Schlüssel in sich trägt.

Da man das gesamte Chifftrat 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-Chifftrat-Kodierung unüblich ist (weil ohne TI-Chipkarteneinsatz unnötig), ist die Kodierung der Chifftrate 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-Chifftrat 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<sub>A6</sub>-(oidDO || keyDO || cipherDO || macDO)'.  
iii. *oidDO* = '06-L<sub>06</sub>-oid'.
- iv. *keyDO* = '7F49-L<sub>7F49</sub>-(86-L<sub>86</sub>-PO<sub>A</sub>)'.
- v. *cipherDO* = '86-L<sub>86</sub>-(02 || C)'.
- vi. *macDO* = '8E-L<sub>8E</sub>-T'.

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

Beispiel:

```
poGOBgkrJAMDAggBAQd/SUOGQQRouC6tM2TQQ+RP3pptgdAaDF8Te7IVCkUBe2H+PJSLK4W/BXI
X
kndiBwEfftd5wk4pjzCdC2j1q14/CIWcW89nhjEC7G47UAu2ZqmbIhxstkXV3UI2UUek/qwBwtb
2
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-Chifftrat 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-Chiffreten 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#5.7-ECIES]).
2. Mit diesem Transportschlüssel MÜSSEN die Nutzerdaten mittels AES-GCM und den Vorgaben aus GS-A\_4389 verschlüsselt werden.
3. Der Transportschlüssel MUSS unkodiert mit den in [gemSpec\_COS#6.8.1.4 ELC Verschlüsselung] aufgeführten Vorgaben mittels ECIES verschlüsselt werden (siehe Erklärung in [gemSpec\_Krypt#ECIES]). Das damit entstehende Chifftrat wird im Folgenden als Transport-Chifftrat bezeichnet.



4. Das Transport-Chifftrat MUSS wie in [gemSpec\_Krypt#5.7-ECIES] beschrieben in eine ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler ASN.1-Binärverpackung kodiert werden.
5. Diese Kodierung MUSS in eine keyEncryptionAlgorithm-Datenstruktur mit der OID oid\_ti\_ecies\_transport\_encryption [gemSpec\_OID] eingebracht werden.
6. Die restliche Kodierung des mittels AES-GCM erzeugten Chiffrats der Nutzerdaten MUSS wie in CMS üblich [RFC-5626] [RFC-5084] erfolgen (vgl. Darstellung in [gemSpec\_Krypt#5.7-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-Chifftrat MUSS wie in [gemSpec\_Krypt#5.7-ECIES] beschrieben in eine ASN.1-Struktur (Tupel (PO, C, T) in kartenkompatibler ASN.1-Binärverpackung kodiert werden.
2. Diese ASN.1-Binärverpackung MUSS Base64-kodiert werden und so kodiert in eine XML-Datenstruktur, so wie sie in [gemSpec\_Krypt#5.7-ECIES] beschrieben ist, eingebracht werden (Hinweis: man beachte ohne "RecipientKeyInfo" Tags).
3. Die mittels AES-GCM verschlüsselten Nutzerdaten MÜSSEN ebenfalls Base64-kodiert in die eben erzeugte XML-Datenstruktur gemäß [gemSpec\_Krypt#5.7-ECIES] eingebracht werden.

[<=]

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

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

In [SEC1-2009#5.2] und in [RFC-5753#4] wird auf ein potentiell 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.

### **5.7.2 ECIES und mobKT**

Ein "Mobiles Kartenterminal" [gemSpec\_MobKT] ist so etwas wie ein Tablet mit zwei Chipkartenslots. Es ermöglicht einem LE außerhalb seiner Praxisräumlichkeiten (also insbesondere ohne Konnektor und stationäres eHealth-Kartenterminal), auf Daten eines Versicherten auf dessen eGK zuzugreifen. Das Mobile Kartenterminal muss die dabei ausgelesenen versichertenspezifischen Daten verschlüsselt lokal im Gerät ablegen. Wenn das Mobile Kartenterminal verloren geht, sind damit diese Daten geschützt. Sie können erst mit Hilfe des gesteckten und freigeschalteten HBA des LE wieder entschlüsselt (genutzt) werden, bspw. zur Übertragung ins Primärsystem. Für die Verschlüsselung muss nach Ende 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 Chiffre 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#5.7. 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.

### **5.8 ECC-Migration eHealth-KT**

Mit A\_17089-01 und A\_17090-01 (vgl. Abschnitt 3.3.2) wird vom eHealth-Kartenterminal die Unterstützung der mit der Kartengeneration 2.1 (vgl. [gemSpec\_gSMC-KT\_ObjSys\_G2.1]) hinzugekommenen ECDSA-basierten Identität bereitgestellt.

### **A\_17089-03A\_17089-01 - 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 MÜSSEN TLS\_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-186-45] 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-186-45] 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 Komponenten-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.

### **5.8.1 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-Protokolls verwenden.
2. TLS\_ECDHE\_ECDSA... (also ECDSA) auswählt, dann MUSS es eine ECDSA-basierte Identität zur Authentisierung im Kontext des TLS-Protokolls 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).

[<=]

## 5.9 ECC-Migration Konnektor

### **A\_17094-02A\_17094-01 - 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-186-45] 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:



```
-----BEGIN EC PARAMETERS-----
BgkrJAMDAggBAQc=
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHgCAQEEIC5PFJ/4fOdPLZ1BebMqpHN8ydae9kSPGS5qICS/WGpnoAsGCSskAwMC
CAEBB6FEA0IABCpcFsaT3f+O9Cg676mRWR4WazrrbFMcMZ5dwJhrhVsjszA3z85q
u8eCSvAkeE3jt2+nOTNh1s3ExkZVSOJD1aLc=
-----END EC PRIVATE KEY-----
```

(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.

## 5.10 Verschiedene Produkttypen und ECC-Migration (informativ)

Dem VPN-Zugangsdienst sind die IPsec-Anforderungen [A\\_17125](#) und [A\\_17126](#) zugewiesen, ebenso [A\\_17205](#). Im Rahmen der Registrierung bei einem VPN-Zugangsdienst wird vom Konnektor eine Signatur mittels einer SMC-B erzeugt. Diese muss der VPN-Zugangsdienst (Registrierungsserver) prüfen können, dafür gilt für diesen [A\\_17206](#).

Für die Produkttypen, die bei der Anwendung VSDM verwendet werden, ist die ECC-RSA-TSL-Auswertung [A\\_17205](#) und die ECC-Unterstützung bei TLS [A\\_17124](#)-\* relevant.

Fachanwendungsspezifische Anpassungen aufgrund der ECC-Migration befinden sich in den jeweiligen Spezifikationen (bspw. [\[gemSpec\\_CM\\_KOMLE#A\\_17464\]](#)).



---

## **6 VAU-Protokoll für E-Rezept**

---

---

## **6-Kommunikationsprotokoll zwischen ePA-VAU und ePA-Clients**

---

### **6.1 Motivation**

Die Architekturentscheidung, dass auf der Verbindungsstrecke zwischen einem ePA-Frontend eines Versicherten (Client) bzw. eines FM ePA (Client) und einer VAU (Server) immer HTTP über ein Gateway als Kommunikationsprotokoll verwendet wird, hat Vorteile. Ein Nachteil ist, dass damit keine direkte TLS-Verbindung zwischen Client und Server möglich ist. Der TLS-Kanal terminiert am Gateway. Um die "letzten Meile" zwischen Gateway und VAU innerhalb des ePA-Aktensystems nicht ungeschützt zu lassen, wird das "VAU-Protokoll" eingeführt und verwendet. Es wird nicht der Weg gewählt HTTP über TLS über HTTP über TLS zu tunneln. Stattdessen wird das folgende Protokoll eingeführt als eine leichtgewichtige Sicherungsschicht zwischen einer VAU (Server) und einem ePA-Frontend eines Versicherten (Client) bzw. eines FM ePA (Client). Der Server und der Client besitzen jeweils eine kryptographische Langzeitidentität. Diese sind Grundlage für einen beidseitig authentisierten ephemeren ECDH. Aus dem gemeinsamen ECDH-Geheimnis werden mittels einer HKDF zwei AES-Schlüssel abgeleitet (vgl. A-16943-\*). Per AES-GCM werden nach dem Schlüsselaustausch alle ausgetauschten Nutzerdaten kryptographisch gesichert. Es gibt einen Nachrichtenzähler der vor Replay-Attacken schützt.

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

### **6.2 Übersicht**

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

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

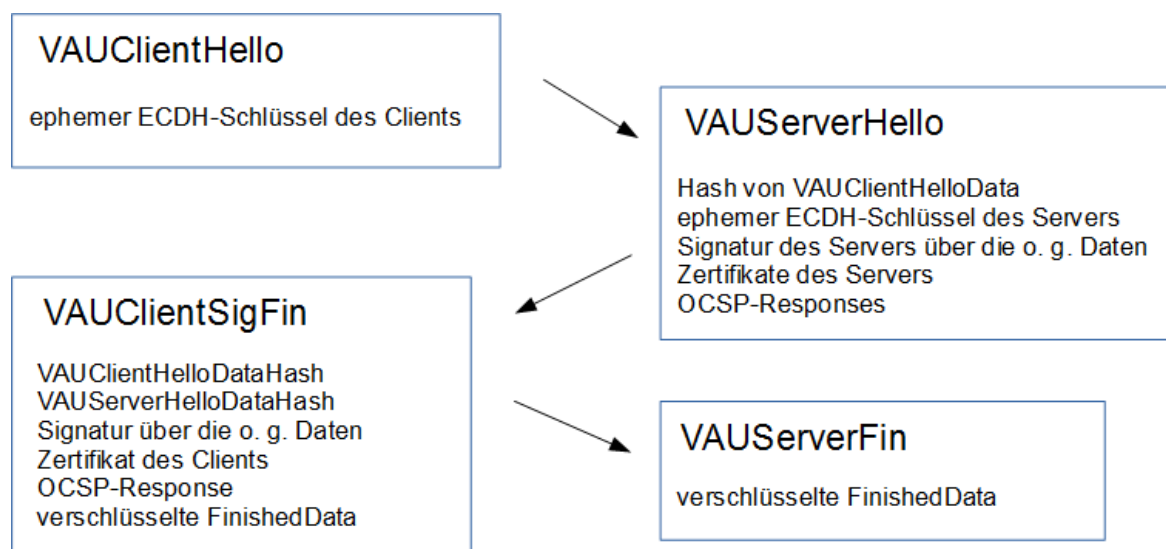
Ein Designziel ist es, den Verbindungsaufbau möglichst "menschenslesbar" zu gestalten und so die Implementierung und die Fehlersuche (u. a. auch im Betrieb) zu erleichtern. So sind die meisten Nachrichten einfache, menschenlesbare JSON-Objekte. Die im Protokoll definierten Fehlermeldungen (VAUServerError-Nachricht)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

einfach, so dass ein Client oder ein Server bei Auftreten eines Fehlers selten etwas anderes tun kann, als die Protokollabarbeitung abubrechen (und als Client einen neuen Protokolldurchlauf zu initiieren). Dies bedeutet: die Art der aufgetretenen Fehlernachricht ist für den Client i. d. R. irrelevant, weil die Handlungsoptionen sehr begrenzt sind. Erst bei der manuellen Fehleranalyse werden die differenzierten Fehlermeldungen hilfreich.

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

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

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



**Abbildung 3: Übersicht über das VAU-Protokoll**

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

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

#### **A\_16884 VAU-Protokoll: Nachrichtentypen und HTTP-Content-Type**

**6.3 Es gibt genau zwei Nachrichtenarten: (1) Nachrichten zur Schlüsselaushandlung (VAUClientHello, VAUServerHello, VAUClientSigFin und VAUServerFin) und Fehlermeldungsübermittlung (VAUServerError) und (2) Nachrichten, die kryptographisch geschützte Nutzerdaten transportieren.**  
**Typ (1) Nachrichten MÜSSEN vom Client und vom Server jeweils**

~~per HTTP mit dem Content-Type 'application/json' übermittelt werden und Typ (2) Nachrichten mit dem Content-Type 'application/octet-stream'. [≤=]~~

#### ~~A\_17074 – VAU-Protokoll: Ignorieren von zusätzlichen Datenfeldern in Protokoll-Nachrichten~~

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

~~Die in den Anforderungen angegebenen Reihenfolgen von Attribute-Value-Pairs innerhalb der JSON-Objekte sind als Beispiele zu verstehen und nicht in ihrer Reihenfolge normativ. Es gilt der JSON-Standard [ecma-262].~~

#### ~~A\_17081 – VAU-Protokoll: zu verwendende Signaturschlüssel~~

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

### 6.4 VAU-ClientHello-Nachricht

#### ~~A\_16883-01 – VAU-Protokoll: Aufbau VAU-ClientHello-Nachricht~~

~~6.5 Der Client MUSS die Kommunikation mittels einer VAU-ClientHello-Nachricht initiieren. Dafür erzeugt er zunächst eine VAU-ClientHelloData-Datenstruktur der Form~~

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

~~Der Client MUSS im Rahmen der Schlüsselaushandlung ein ECDH-Schlüsselpaar basierend auf der Kurve BrainpoolP256r1 [RFC-5639] erzeugen. Er MUSS im "PublicKey"-Feld den öffentlichen Punkt des ephemeren ECDH-Schlüsselpaares Base64-kodiert gemäß [TR-03111#5.1.1 X9.62 Format] eintragen. Im "AuthorizationAssertion"-Feld MUSS der Client die Base64-kodierte Authorization-Assertion gemäß A\_15592-03 eintragen. Der Client MUSS im "CertificateHash"-Feld den Base64-kodierten~~

~~Hashwert seines Client-Zertifikats (AUT oder AUT\_alt-Zertifikat) eintragen (Der Hashwert wird vom kompletten DER-kodierten X.509-Zertifikat inkl. äußerer Zertifikatssignatur erzeugt. Der SHA-256-Hashwert (d. h. 256-Bit = 32-Byte) wird anschließend Base64-kodiert. Diese Kodierung wird als Wert bei "CertificateHash" eingetragen.).~~

#### **Beispiel:**

```
{
  "DataType": "VAUClientHelloData",
  "CipherConfiguration": [ "AES-256-GCM-BrainpoolP256r1-SHA-256" ],
  "PublicKey": "MFowFAYHKOziZj0CAQYJKyQDAwIIAQEHA0IABDY8OMZlrJpLUdgnm8gHbevPFjemkL8IxMXohQlw3VHePf+T1lW+P0nW9VpnU1SxwCkjY1PU6HGTT+3wawKvRIE=",
  "AuthorizationAssertion": ".....",
  "CertificateHash": "wu72yzp4KdteWV/vJUeKWl4UL+FIJyWewgETbxwDK+4="
}
```

~~Hinweis: Der öffentliche Schlüssel im Beispiel hat nach der Base64-Dekodierung folgende ASN.1-Datenstruktur:~~

```
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 36 3C 38 C6 65 AC 9A 4B 51 D8 27 9B C8 07 6D
: EB E9 16 37 A6 90 BF 08 C4 C5 E8 85 09 70 DD 51
: DE 3D FF 93 D6 55 BE 3F 49 D6 F5 5A 67 53 54 B1
: C0 29 23 63 53 D4 E8 71 93 4F ED F0 6B 02 AF 44
: 81
: }
```

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

~~Die VAUClientHello-Nachricht hat folgenden Aufbau:~~

```
{
```

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

~~[<=]~~

#### ~~A\_16897 VAU Protokoll: Versand der VAUClientHello Nachricht~~

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

### ~~6.6 VAU ServerHello Nachricht~~

~~A\_16898 VAU Protokoll: Erzeugung des Hashwert vom Data-Feld aus der VAUClientHello Nachricht~~

~~6.7 Der Server MUSS beim Empfang der VAUClientHello Nachricht einen SHA-256 Hashwert der Daten im Data-Feld (zunächst keine Base64-Dekodierung durchführen) erzeugen. [<=]~~

~~A\_16901-02 VAU Protokoll: Aufbau der VAU ServerHello Nachricht~~

~~6.8 Der Server MUSS auf die VAUClientHello Nachricht mit einer VAU ServerHello Nachricht, folgender Form, antworten~~

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

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

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

```
{  
  "DataType": "VAUServerHelloData",  
  "CipherConfiguration": [ "AES-256-GCM-BrainpoolP256r1-SHA-  
256" ],  
  "VAUClientHelloDataHash": "...SHA-256-Hashwert-des-  
erhaltenen-Data-Felds-in-VAUClientHello...",  
  "PublicKey": "...Base64-kodierter-ECC-Schlüssel(DER)...",  
  "CertificateHash": "...Base64-kodierter-SHA-256-Hashwert-des-  
Server-X.509-Zertifikats"  
}
```

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

~~Der Server MUSS im Feld "VAUClientHelloDataHash" den Base64-kodierten SHA-256-Hashwert der empfangenen VAUClientHelloData (ohne Base64-Dekodierung) eintragen (vgl. A\_16898).~~

~~Der Server MUSS die in der Datenstruktur (VAUServerHello) angegebene Signatur erzeugen (über den Base64-kodierten Wert im "Data"-Feld). Im "Certificate"-Feld MUSS er das für eine Signaturprüfung notwendige EE-Zertifikat eintragen und im "OCSPResponse"-Feld die OCSP-Response, die nicht älter als 24 Stunden sein darf, für dieses EE-Zertifikat. [≤]~~

~~A\_16902—VAU-Protokoll: Versand der VAUServerHello-Nachricht~~

~~6.9 Der Server MUSS auf eine VAUClientHello-Nachricht mit einer VAUServerHello-Nachricht antworten. Der Server MUSS die VAUServerHello-Nachricht per HTTP mit dem Content-Type 'application/json' an den Client senden. [≤]~~

~~A\_16903—VAU-Protokoll: Client, Prüfung des VAUClientHelloDataHash-Werts (aus VAUServerHelloData)~~

~~6.10 Der Client MUSS beim Empfang der VAUServerHello-Nachricht den Hashwert "VAUClientHelloDataHash" (vgl. A\_16901\*) mit dem vom ihm (Client) vor dem Versand der VAUClientHello-Nachricht (vgl. A\_16883-01) errechneten Wert vergleichen. Sind~~



~~die beiden Werte verschieden, so MUSS der Client den  
Protokollablauf abbrechen. [ <= ]~~

~~A\_16941-01—VAU-Protokoll: Client, Prüfung der Signatur der  
VAUServerHelloData~~

~~6.11 Der Client MUSS die Signatur der Daten von "Data" prüfen  
(bitgenau den Datenwert von "Data" nehmen, ohne eine Base64-  
Dekodierung) der "Data"-Daten vorzunehmen. Der Client MUSS  
dafür den Signaturschlüssel des Servers auf Authentizität und  
Integrität prüfen. (Hinweis: in einem Client wird die TSL der TI als  
Prüfgrundlage für die Prüfung von TI-Zertifikaten verwendet.)  
Falls die Signaturprüfung kein positives Ergebnis erbringt, so  
MUSS der Client den Protokollablauf abbrechen (vgl. A\_16849).  
Der Client MUSS prüfen, ob der im VAUServerHelloData-  
>CertificateHash aufgeführte Hashwert mit dem Hashwert des  
Server-Zertifikats im VAUServerHello->Certificate-Feld  
übereinstimmt (vgl. Erzeugung des Hashwerts in A\_16901-\*).  
Falls nein, so MUSS der Client den Protokollablauf abbrechen  
(vgl. A\_16849). [ <= ]~~

~~A\_23273—VAU-Protokoll: Client, Prüfung der Signatur der VAUServerHelloData,  
Gültigkeit von OCSP-Antworten~~

~~Der Client MUSS bei der Signaturprüfung der VAUServerHelloData die im  
"OCSPResponse"-Feld enthaltene OCSP-Response verwenden, wenn diese nicht älter als  
24 Stunden ist.~~

~~Dabei MUSS geprüft werden: Systemzeit - 24h <= producedAt. (Eine zusätzliche  
Auswertung von nextUpdate SOLL NICHT erfolgen.)~~

~~Andernfalls MUSS der Client eine frische OCSP-Response einholen. [ <= ]~~

## 6.12 Schlüsselableitung

~~A\_16852-01—VAU-Protokoll: ECDH durchführen~~

~~6.13 Der Client und auch der Server MÜSSEN jeweils für sich  
prüfen, ob der empfangene ephemere öffentliche elliptische  
Kurvenpunkt der Gegenseite auch auf der von ihnen verwendeten  
Kurve (BrainpoolP256r1) liegt.  
Falls nein, MÜSSEN sie jeweils den Protokollablauf abbrechen.  
Falls der Server derjenige ist, der in diesem Fall abbricht, MUSS  
der zuvor an den Client eine VAUServerError-Nachricht mit der  
Fehlermeldung "invalid curve (ECDH)" senden.  
Falls ja, MÜSSEN beide einen ECDH nach [NIST-800-56-A]~~

~~durchführen. Das dabei erzeugte gemeinsame Geheimnis ist folgend Grundlage von drei Schlüsselableitungen (vgl. A\_16943-01).[<=]~~

#### ~~A\_16943-01 – VAU-Protokoll: Schlüsselableitung (HKDF)~~

~~Für die Schlüsselableitung MÜSSEN Client und Server die HKDF nach [RFC-5869] auf Basis von SHA-256 verwenden.~~

~~Das "Input Keying Material" (IKM) [RFC-5869] ist das in A\_16852-01 erzeugte gemeinsame ECDH-Geheimnis zwischen Server und Client.~~

~~Die erste Schlüsselableitung hat den Ableitungsvektor "KeyID" ("info" Parameter aus [RFC-5869] ist dann also "KeyID") und erzeugt einen 256-Bit langen Schlüsselidentifizier.~~

~~Die zweite Schlüsselableitung mit dem Ableitungsvektor "AES-256-GCM-Key-Client-to-Server" erzeugt den 256-Bit AES-Schlüssel für die Verwendung innerhalb von AES-256-GCM für Nachrichten, die der Client für den Server verschlüsselt.~~

~~Die dritte Schlüsselableitung mit dem Ableitungsvektor "AES-256-GCM-Key-Server-to-Client" erzeugt den 256-Bit AES-Schlüssel für die Verwendung innerhalb von AES-256-GCM für Nachrichten, die der Server für den Client verschlüsselt.[<=]~~

## 6.14 VAUClientSigFin-Nachricht

#### ~~A\_17070-02 – VAU-Protokoll: Aufbau der VAUClientSigFin-Nachricht~~

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

~~Die VAUClientSigFin-Nachricht hat folgenden Aufbau:~~

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

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

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

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

~~Eine ECDSA-Signatur im "Signature"-Feld MUSS nach [TR-03111#5.2.2: X9.62-Format] ("ecdsa-with-sha256") kodiert sein.~~

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

~~Eine RSASSA-PSS-Signatur MUSS nach [RFC-8017] (PKCS#1) kodiert werden. Diese so kodierte Signatur wird Base64-kodiert und als Wert im "Signature"-Feld eingetragen.~~

(Verständnishinweis: Eine G2-Karte kann in Bezug auf AUT-Schlüssel nur RSA-Signaturen erzeugen. Eine G2.x-Karte kann und MUSS im Kontext VAU-Protokoll ECDSA-Signaturen erzeugen.)

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

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

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

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

~~"VAUClientSigFin" ||~~

~~unkodierter Hashwert aus "VAUClientHelloDataHash" ||~~

~~unkodierter Hashwert aus "VAUServerHelloDataHash"~~

Diese Zeichenkette MUSS 15+32+32=79 Bytes lang sein. Der Client MUSS diese Zeichenkette mittels AES-GCM (vgl. A\_16943\_01) verschlüsseln und dabei folgende Zeichenkette bilden

~~256-Bit KeyID || 96-Bit Nonce (IV) mit Ciphertext und 128 Bit Authentication-Tag~~

Diese Zeichenkette MUSS er Base64-kodieren und das Ergebnis als Wert des "FinishedData"-Feld eintragen.

~~[<=]~~

Hinweis: Obwohl innerhalb einer der VAUServerHelloData der Wert VAUClientHelloDataHash enthalten ist und damit auch in die Berechnung von VAUServerHelloDataHash mit einfließt, wird der Wert VAUClientHelloDataHash explizit in VAUClientSigFin aufgeführt. Ziel ist es möglichst direkt Transparenz über die bestätigten Daten zu schaffen.

#### **~~A\_17071 VAU-Protokoll: Versand der VAUClientSigFin-Nachricht~~**

Der Client MUSS auf eine VAUServerHello-Nachricht mit einer VAUClientSigFin-Nachricht antworten. Der Client MUSS die VAUClientSigFin-Nachricht per HTTP mit dem Content-Type 'application/json' an den Server senden. ~~[<=]~~

### **~~6.15 VAUServerFin-Nachricht~~**

#### **~~A\_17072-01 VAU-Protokoll: Empfang der VAUClientSigFin-Nachricht~~**

Der Server MUSS beim Empfang der VAUClientSigFin-Nachricht prüfen,

- ~~1. ob die darin enthaltene Signatur gültig ist,~~
- ~~2. ob der Hashwert des Client-Zertifikats aus dem "Certificate"-Feld gleich dem Hashwert aus dem ClientHelloData->CertificateHash-Feld ist (vgl. Erzeugung des Hashwerts in A\_16883-01), und~~
- ~~3. ob der Wert im "FinishedData"-Feld der nach A\_17070-02 zu erwartenden Wert entspricht.~~

Falls eine der Prüfungen 1 bis 3 ein nicht-positives Prüfergebnis liefert, so MUSS der Server mit einer VAUServerError-Nachricht antworten und die weitere

Protokolldurchführung abbrechen. Wobei er folgende Fehlermeldung pro Prüfung verwenden MUSS:

1. → "Signature from VAUClientSigFin invalid"
2. → "Client Certificate inconsistent", und
3. → "VAUClientSigFin invalid".

## ~~6.16 [ $\Leftarrow$ ]~~

### ~~A\_16899 VAU-Protokoll: Aufbau der VAUServerFin-Nachricht~~

#### ~~6.17 Der Server MUSS eine wie folgt aufgebaute VAUServerFin-Nachricht erzeugen:~~

```
{  
  "MessageType" : "VAUServerFin",  
  "FinishedData" : "...Base64-kodierte-verschlüsselte-Finished-Daten..."  
}
```

#### ~~Der Server MUSS für die Berechnung des "FinishedData"-Feldes zunächst folgende Zeichenkette bilden~~

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

#### ~~Diese Zeichenkette MUSS 12+32+32=76 Bytes lang sein. Der Server MUSS diese Zeichenkette mittels AES-GCM (vgl. A\_16943-01) verschlüsseln und dabei folgende Zeichenkette bilden~~

```
256-Bit KeyID || 96-Bit Nonce (IV) mit Ciphertext und 128-Bit Authentication-Tag
```

#### ~~Diese Zeichenkette MUSS er-Base64-kodieren und das Ergebnis als Wert des "FinishedData"-Feld eintragen. [ $\Leftarrow$ ]~~

### ~~A\_17073 VAU-Protokoll: Versand der VAUServerFin-Nachricht~~

#### ~~6.18 Der Server MUSS nach dem Erhalt einer VAUClientSigFin-Nachricht mit einer VAUServerFin-Nachricht antworten. Der Server MUSS die VAUServerFin-Nachricht per HTTP mit dem Content-Type 'application/json' an den Client senden. [ $\Leftarrow$ ]~~

### ~~A\_17084 VAU-Protokoll: Empfang der VAUServerFin-Nachricht~~

~~Der Client MUSS beim Empfang der VAUServerFin-Nachricht prüfen, ob der Wert im "FinishedData"-Feld der nach A\_16899 zu erwartenden Wert entspricht. Falls nein, so MUSS der Client den weiteren Protokollablauf abbrechen (vgl. A\_16849). [ $\Leftarrow$ ]~~

## 6.19 Nutzerdatentransport

### ~~A\_16945-02—VAU-Protokoll: Client, verschlüsselte Kommunikation (1)~~

~~Wie bei der Schlüsselaushandlung MUSS der Client mittels HTTP POST Request die nun verschlüsselte Kommunikation initiieren.~~

~~Der Client MUSS einen unsigned 64-Bit Nachrichtenzähler führen, die er bei jeder abgeschickten Nachricht um zwei erhöhen MUSS.~~

~~Er bildet die Datenstruktur "P1" mit~~

~~P1-Version (ein Byte mit dem Wert 0x01) ||  
Nachrichtenzähler (unsigned 64-Bit im Big-Endian-Format) ||  
Anzahl der Bytes der folgenden zusätzlichen HTTP-  
Header-Informationen (unsigned 32-Bit im Big-Endian-Format) ||  
zusätzliche HTTP-Header-Informationen ||  
Plaintext  
wobei „Plaintext“ die zu übertragende Nutzlast (bspw. SOAP Request) bezeichnet.~~

~~Wenn die Anzahl der Bytes der folgenden zusätzlichen HTTP-Header-Informationen mit 0 (also 0x00000000) angegeben wird, so gibt es keine folgenden zusätzlichen HTTP-Header-Informationen, d. h. es folgen direkt die Plaintext-Bytes.~~

~~Der Nachrichtenzähler MUSS initial mit 1 starten.~~

~~Der Client MUSS zunächst einen IV wie folgt erzeugen:~~

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

~~6.20 Damit ist der IV 96-Bit lang. Unter Verwendung dieses IV und des zweiten aus A\_16943-01 abgeleiteten Schlüssel (Client-to-Server-Schlüssel) wird P1 verschlüsselt. Der Client berechnet so den "Ciphertext".~~

~~Die vom Client nun an den Server zu übermittelnde Datenstruktur MUSS folgende Form besitzen:~~

~~256-Bit KeyID || 96-Bit IV mit Ciphertext und 128-Bit Authentication-Tag~~

~~Diese Nachricht MUSS der Client per HTTP POST Request mit Content-Type 'application/octet-stream' ohne weitere Kodierungen versenden.[<=]~~

### ~~A\_16952-02—VAU-Protokoll: Server, verschlüsselte Kommunikation~~

~~Der Server erkennt aus der KeyID, welchen AES-Schlüssel er für die Entschlüsselung verwenden muss (vgl. A\_16943-\* zweiter abgeleiteter Schlüssel (Client-to-Server-Schlüssel)).~~

~~Falls ihm die KeyID unbekannt ist, so MUSS er mit einer VAUServerError-Nachricht mit der Fehlermeldung "KeyID-XXX not found" antworten, wobei er XXX durch die empfangene KeyID in Hexadezimalform ersetzen MUSS.~~

~~Falls bei der Entschlüsselung ein Fehler auftritt (bspw. Authentication-Tag passt nicht zur Nachricht), MUSS der Server mit einer VAUServerError-Nachricht mit der Fehlermeldung~~

~~"AES-GCM-decryption-error." antworten.~~

~~Falls die Entschlüsselung erfolgreich war, MUSS der Server den Klartext gemäß der Struktur von P1 aus A\_16945\* interpretieren.~~

~~Falls die Version in P1 ungleich 0x01 ist, so MUSS der Server (1) eine VAUServerError-Nachricht gemäß A\_16851\* mit der Fehlermeldung "invalid-protocol-version" senden und gemäß A\_16849 die Protokollausführung abbrechen.~~

~~Sei mit "Server-Zählerwert" der letzte vom Server für den Nachrichtenversand verwendete Zählerwert bezeichnet. Initial (d. h. es wurde innerhalb eines Protokollablaufs noch nie eine Nachricht vom Server versendet) MUSS dieser Server-Zählerwert gleich 0 sein.  
Der Server MUSS prüfen, ob der Zählerwert im Klartext größer als der "Server-Zählerwert" ist. Falls nein, so MUSS der Server (1) eine VAUServerError-Nachricht gemäß A\_16851\* mit der Fehlermeldung "invalid-counter-value" senden und (2) gemäß A\_16849 die Protokollausführung abbrechen.  
Der Server MUSS den "Server-Zählerwert" auf Zählerwert + 1 setzen.  
Falls es dabei zu einem Zählerüberlauf kommt, so MUSS der Server (1) eine VAUServerError-Nachricht gemäß A\_16851\* mit der Fehlermeldung "message-counter-overflow" senden und (2) gemäß A\_16849 die Protokollausführung abbrechen.  
Der Server MUSS zusätzliche HTTP-Header-Informationen analog zu A\_16945\* interpretieren und verwenden. Falls dies nicht möglich ist (bspw. Längenwert ist größer als eigentliche Nachrichtengröße), so MUSS der Server (1) eine VAUServerError-Nachricht gemäß A\_16851\* mit der Fehlermeldung "HTTP-additional-header-length-error" senden und (2) gemäß A\_16849 die Protokollausführung abbrechen.~~

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

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

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

**~~6.21 Damit ist der IV 96-Bit lang. Unter Verwendung dieses IV und des dritten aus A\_16943\* abgeleiteten Schlüssel (Server-to-Client-Schlüssel) MUSS der Server P2 verschlüsseln.~~**

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

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

~~A\_16957-01—VAU-Protokoll: Client, verschlüsselte Kommunikation (2)~~

~~Beim Empfang der Antwort (vgl. A\_16952-\*) MUSS der Client folgende Vorgaben durchsetzen:~~

~~Falls~~

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

~~6.22 so MUSS der Client die Nachricht verwerfen und die weitere Protokollausführung mittels des empfangenen KeyID abbrechen.~~

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

~~A\_16958—VAU-Protokoll: Client, Neuinitiiieren einer Schlüsselaushandlung~~

~~6.23 Der Client KANN jeder Zeit eine neue Schlüsselaushandlung (VAUClientHello etc.) initiieren. [≤]~~

~~A\_17069—VAU-Protokoll: Client Zählerüberlauf~~

~~Der Client MUSS, falls so viele Nachrichten ausgetauscht werden, dass für den unsigned 64-Bit-Nachrichtenzähler ein arithmetischer Überlauf droht, eine neue Schlüsselaushandlung initiieren (VAUClientHello etc.). [≤]~~

~~6.24 VAUServerError-Nachricht~~

~~A\_16851-01—VAU-Protokoll: VAUServerError-Nachrichten~~

~~6.25 Der Server MUSS folgende Vorgaben umsetzen:  
In verschiedenen im Protokoll beschriebenen Fehlerfällen sendet der Server eine VAUServerError-Nachricht an den Client.  
Für die eigentliche Fehlerübermittlung MUSS folgende Datenstruktur erzeugt:~~



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

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

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

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

~~Die ECDSA-Signatur im "Signature"-Feld MUSS nach [TR-03111#5.2.2. X9.62-Format] ("ecdsa-with-Sha256") kodiert sein. Im "Certificate"-Feld MUSS der Server, das verwendete Signaturzertifikat aufführen, und im "OCSPResponse"-Feld eine OCSP-Response für dieses Zertifikat, welche nicht älter als 24 Stunden ist. [<=]~~

~~A\_16900—VAU-Protokoll: Client, Behandlung von Fehlernachrichten~~

~~6.26 Erhält der Client eine VAUServerError-Nachricht (vgl. A\_16851-01), MUSS er die Signatur prüfen. Falls die Prüfung positiv ist, so MUSS er die Protokolldurchführung abbrechen (vgl. A\_16849). [<=]~~

~~A\_23282—VAU-Protokoll: Signaturen im VAU-Protokoll~~

~~Clients und Server im VAU-Protokoll-Sinne MÜSSEN bei der Signaturerzeugung und-Prüfung beachten, dass die Signaturen der Nachrichten des VAU-Protokoll (wie in A\_16851-\* definiert) über die Base64-kodierten Daten erfolgt. Dies gilt für alle Protokoll-Nachrichten im VAU-Protokoll—stets werden die Base64-kodierten Daten des "Data"-Felds signiert. [<=]~~

## 6.27 Abbrechen des Protokollablaufs

~~A\_16849—VAU-Protokoll: Aktionen bei Protokollabbruch~~

~~Wenn ein Client oder ein Server den Protokollablauf nach Protokollbeschreibung abbrechen muss, dann MUSS dieser die eventuell aktuell vorhandene KeyID aus seiner~~

~~Datenbasis löschen und die damit verbundenen Schlüssel sicher löschen:  
{<=}~~

## 6.28 ~~VAU-Kanal und MTOM/XOP~~

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

Beispiel:

~~Ohne MTOM/XOP:~~

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

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

~~Mit MTOM/XOP:~~

```
POST /URL1 HTTP/1.1
Content-Type: Multipart/Related; boundary="====_Part_1_1234_====",
type="application/xop+xml", start="Eintrag-1", start-info="application/soap+xml",
charset=UTF-8
Content-Transfer-Encoding: binary
Content-Length: ...

====_Part_1_1234_====
Content-Type: application/xop+xml; type="application/soap+xml"; charset="UTF-8"
Content-Transfer-Encoding: 8bit
Content-ID: Eintrag-1

<s11:Envelope xmlns:s11='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:xmime='http://www.w3.org/2005/05/xmlmime'>
  <s11:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo xmime:contentType='image/jpeg'>
```

```
-----<xop:Include xmlns:xop='http://www.w3.org/2004/08/xop/include'>
  href='cid:Bild-1'/>
</m:photo>
</m:data>
</s11:Body>
</s11:Envelope>
-
-----=_Part_1_1234-----
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: Bild-1
-
... hier kommen die binären Daten ...
-----=_Part_1_1234-----
-

```

Für die Dekodierung einer solchen MIME-Multipart-Kodierung ist es sehr hilfreich (und ggf. notwendig) als Empfänger den vom Sender intendierten (vollständigen) „Content-Type“ aus dem HTTP-Request-Header bzw. dem HTTP-Response-Header zu kennen. Dieser wird jedoch durch das VAU-Protokoll überschrieben (vgl. A\_16945\* und A\_16952\*). Um die u. a. nach [gemSpec\_FM\_ePA#A\_16220\*] und [gemSpec\_ePA\_FdV#A\_16222\*] geforderte Verwendung von MTOM/XOP zu unterstützen, wird der originär intendierte Content-Type, der im „start“-Feld ggf. vertrauliche Daten enthalten kann, innerhalb der „zusätzlichen HTTP-Header-Informationen“ (vgl. A\_16945\* und A\_16952\*) mitgeliefert.

#### **~~A\_18465-01-VAU-Protokoll: MTOM/XOP-HTTP-Header-Informationen~~**

Wenn ein VAU-Client oder ein VAU-Server Übertragungen mittels MTOM/XOP durchführen, so MÜSSEN sie die durch MTOM/XOP erzeugten „Content-Type“-Informationen (vgl. [gemSpec\_Krypt#6.11 VAU-Kanal und MTOM/XOP]) innerhalb ihrer Nachricht als „zusätzliche HTTP-Header-Informationen“ gemäß A\_16945\* und A\_16952\* aufführen. [~~=~~]

## **~~6.29-Zusätzliche HTTP-Header-Informationen~~**

### **~~A\_20549-VAU-Protokoll: Einbringen der ursprünglich intendierten Content-Type-Variable~~**

**~~6.30-Ein VAU-Client oder ein VAU-Server MÜSSEN in den „zusätzliche HTTP-Header-Informationen“ gemäß A\_16945\* und A\_16952\*, die ursprünglich intendierten Content-Type-HTTP-Header-Variable einbringen. D. h. Ein Kommunikationspartner erstellt einen HTTP-Request oder eine HTTP-Response. Diese hat eine "ursprünglich intendierten Content-Type", bspw. "application/soap+xml; charset=utf-8". Diese MUSS dann bei der Verschlüsselung der Nachricht mittels des VAU-Protokolls in die~~**

~~"zusätzlichen HTTP-Header-Informationen" eingetragen werden. [<=]~~

~~A\_18466-01—VAU-Protokoll: zusätzliche HTTP-Header-Informationen~~

~~6.316.1 Wenn ein VAU-Client oder ein VAU-Server Übertragungen durchführen und in den empfangenen Nachrichten sind „zusätzliche HTTP-Header-Informationen“ gemäß A\_16945-\* und A\_16952-\*, so MÜSSEN sie diese auswerten (Hinweis: insbesondere „Content-Type“-Informationen (vgl. [gemSpec\_Krypt#6.11 VAU-Kanal und MTOM/XOP])).~~

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

### ~~6.321.1 Tracing in Nichtproduktivumgebungen~~

Für die Fehlersuche — 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#5.8. Tracing in Nichtproduktivumgebungen].

#### ~~A\_21888—VAU-Client, Nichtproduktivumgebung, vorgegebene ECDH-Schlüssel~~

Ein Client im VAU-Protokoll MUSS in einer Nichtproduktivumgebung von A\_16883-\* abweichen in dem er sein ephemeres ECDH-Schlüsselpaar nicht zufällig erzeugt, sondern folgendes ECDH-Schlüsselpaar (auf der Kurve BrainpoolP256r1) für jeden VAU-Protokollverbindungsaufbau verwendet:

- privater Schlüssel:  
d=0x78ece94904fc6bf9900cccd4c5af4dcd4fbeb585b2054b636cc3f76333bdee
- öffentlicher  
Schlüssel: x=0x21fafedc8ba1ef5477995a28a9794f86355df305f1f58afc88f87e91c664353a  
y=0x27803e3dcf670bd305f3b923f915bfe119389d869c3565828c89bd422231f9f4

[<=]

Hinweis-1: SHA-256("gemSpec\_Krypt: privater VAUClient-ECDH-Schlüssel fuer Nichtproduktivumgebungen nach A\_21888-\*) ist als hexdump-kodiert gleich 078ece94904fc6bf9900cccd4c5af4dcd4fbeb585b2054b636cc3f76333bdee

Hinweis-2:

Folgend ist der private Schlüssel aus A\_21888-\* als PKCS8-Container PEM-kodiert aufgeführt:

```
-----BEGIN PRIVATE KEY-----
MIGIAgEAMBOCBYqGSM49AgEGCSokAwMCCAEBBwRtMCSCAQEEIACoZpSQT8a/mQDM
7Uxa9NzU++vFhbIFS2Nsw/djM73ueUQDQgAEIfr+3Iuh71R3mVooqXlPhjVd8wXx
9Yr8iPh+keZkNTongD49z2cL0wXzuSP5Fb/hCTidhpw1ZYKMib1CIjH59A==
-----END PRIVATE KEY-----
```

~~A\_21977—VAU-Client, Nichtproduktivumgebung, vorgegebene ECDH-Schlüssel, optionale Konfigurierbarkeit~~

~~6.33 Ein Client im VAU-Protokoll KANN in einer Nichtproduktivumgebung die Umsetzung von A\_21888\* durch einen Nutzer (FdV) oder Administrator (Konnektor etc.) konfigurierbar machen, d. h. das Verhalten des Clients zwischen zufälliger Schlüsselerzeugung und festen Schlüssel aus A\_21888\* per Konfiguration veränderlich machen. [ <= ]~~

~~A\_21889—VAU-Server, Produktivumgebung, Ablehnung der ECDH-Client-Schlüssel nach A\_21888\*~~

~~Ein Server im VAU-Protokoll MUSS genau in der Produktivumgebung (PU) bei einem VAU-Handshake den Client ECDH-Schlüssel aus A\_21888\* ablehnen. D. h. er muss falls ein Client im VAUClientHello den öffentlichen Schlüssel aus A\_21888\* präsentiert, den Handshake mit einem VAUServerError gemäß A\_16851\* abbrechen. [ <= ]~~

## 6.34 Optionale PQC-Erweiterung (informativ)

In diesem Abschnitt wird eine optionale Protokollerweiterung des VAU-Protokolls für ePA dargestellt, die sich aktuell in der gematik in der Evaluierung befindet. Kryptographisch gesehen entspricht sie dem PQC Vorgehen bei TLS aus [TLS-CC-2021]. Mehr Information über PQC findet man in Abschnitt "8 Post-Quanten-Kryptographie (informativ)".

Ein Client initiiert mit einer VAUClientHello-Nachricht den VAU-Protokoll-Handshake (Schlüsselvereinbarung). Möchte er zusätzlich zum klassischen ephemeren ECDH-Schlüsselaustausch ein PQC-Verfahren anwenden und die beiden gemeinsam mit dem VAU-Server dabei erzeugten Geheimnisse in die Schlüsselableitung für die symmetrischen Sitzungsschlüssel einfließen lassen, so führt er im Handshake einen Verfahrensidentifikator inkl. dazugehörigen öffentlichen Schlüssel eines der unten aufgeführten PQC-Verfahren mit an:

In [liboqs] sind die Referenzimplementierungen der Round-3-Kandidaten der NIST-PQC-Standardisierung [NIST-PQC] enthalten. Ebenfalls sind dort folgende Identifier definiert, die den Definitionen aus den Round-3-Kandidaten entsprechen:

Key encapsulation mechanisms (KEM):

- ◆ ~~BIKE: BIKE-L1, BIKE-L3~~
- ◆ ~~Classic McEliece: Classic-McEliece-348864, Classic-McEliece-348864f, Classic-McEliece-460896, Classic-McEliece-460896f, Classic-McEliece-6688128, Classic-McEliece-6688128f, Classic-McEliece-6960119, Classic-McEliece-6960119f, Classic-McEliece-8192128, Classic-McEliece-8192128f~~

- ~~FrodoKEM: FrodoKEM-640-AES, FrodoKEM-640-SHAKE, FrodoKEM-976-AES, FrodoKEM-976-SHAKE, FrodoKEM-1344-AES, FrodoKEM-1344-SHAKE~~
- ~~HQC: HQC-128, HQC-192, HQC-256~~
- ~~Kyber: Kyber512, Kyber512-90s, Kyber768, Kyber768-90s, Kyber1024, Kyber1024-90s~~
- ~~NTRU: NTRU-HPS-2048-509, NTRU-HPS-2048-677, NTRU-HPS-4096-821, NTRU-HPS-4096-1229, NTRU-HRSS-701, NTRU-HRSS-1373~~
- ~~NTRU-Prime: ntrulpr653, ntrulpr761, ntrulpr857, ntrulpr1277, sntrup653, sntrup761, sntrup857, sntrup1277~~
- ~~SABER: LightSaber-KEM, Saber-KEM, FireSaber-KEM~~

Im ~~VAUClientHelloData~~ führt der Client zusätzlich auf:

~~"PQC.PublicKey" : [ "...Verfahrensidentifikator...", "...base64-kodierter Public-Key des PQC-KEM-Verfahrens..." ]~~

~~"Verfahrensidentifikator"~~ steht für einen der o. g. KEM-Identifizier. In [liboqs] ist jeweils für jedes Verfahren eine "KeyGen"-Funktion definiert. Diese erzeugt zufällig ein Schlüsselpaar und gibt als Ergebnis u. a. den öffentlichen Schlüssel aus. Dieser wird so wie er aus der Referenzimplementierung kommt, per base64-Kodierung kodiert im Feld ~~"base64-kodierter Public-Key des PQC-KEM-Verfahrens"~~ eingetragen.

Der Server akzeptiert optionale Attribute in den JSON-Nachrichten des VAU-Handshakes auf Grund von A\_17074-\*. Wenn er das Verfahren ebenfalls unterstützt, dann fügt er den öffentlichen Teil des Ergebnisses der Encaps(öffentlicher Schlüssel des Clients)-Funktion im ~~VAUServerHelloData~~ wie folgt base64-kodiert auf:

~~"PQC.Encaps" : "... base64-kodierter öffentlicher Teil von Encaps(öffentlicher Schlüssel des Clients)..."~~

Beim Aufruf von Encaps(öffentlicher Schlüssel des Clients) wird als Ergebnis der Funktion ein öffentlicher Teil und ein geheimer Schlüssel K als Ergebnis berechnet. Der VAU-Server fügt bei der Schlüsselableitung der symmetrischen Sitzungsschlüssel (vgl. A\_16943-\*) neben dem gemeinsamen ECDH-Geheimnis noch "Verfahrensidentifikator" || K als Eingabe für die HKDF hinzu (IKM= gemeinsames ECDH-Geheimnis || Verfahrensidentifikator || K). Damit hängen die abgeleiteten Sitzungsschlüssel sowohl vom ECDH-Geheimnis als auch vom geheimen Schlüssel K, also vom PQC-KEM, ab:

Bei Erhalt der VAUServerHello-Nachricht im VAU-Client, führt der VAU-Client die Funktion Decaps(s, decodierter öffentlicher Encaps-Wert) aus, wobei s der private Client-PQC-Schlüssel aus der Schlüsselpaarerzeugung für PublicKeyPQC ist (vgl. KeyGen-Funktion). Als Ergebnis erhält er K. Damit kann er genau wie der VAU-Server die Schlüsselableitung der symmetrischen Sitzungsschlüssel (vgl. A\_16943-\*) durchführen (IKM= gemeinsames ECDH-Geheimnis || Verfahren-ID || K).

Sollte der Client bei Erhalt der VAUServerHello-Nachricht kein "Encaps"-Datenfeld im VAUServerHelloData-Teil vorfinden, so unterstützt der VAU-Server das Verfahren nicht. Dem VAU-Client steht es dann frei, den Verbindungsaufbau ohne PQC-Verfahren weiter durchzuführen, den Verbindungsaufbau abubrechen oder den Verbindungsaufbau neu zu beginnen.

Wenn die Schlüsselableitung bei beiden Kommunikationspartnern erfolgreich durchgeführt werden konnte – d. h., der VAU-Protokoll-Handshake konnte erfolgreich abgeschlossen werden, dann verläuft der Protokollablauf wie in Abschnitt 6.8ff definiert wie üblich weiter.

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

### ~~7.16.2~~ Übersicht (informativ)

Ähnlich wie bei der Anwendung ePA besitzt der Fachdienst E-Rezept **mindestens** zwei ~~HTTP~~**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 ~~7.6.2~~ definierte Weise. Diese so erzeugten HTTP-Requests (äußere Requests) sendet der Client per HTTPS an eine der Webschnittstellen des E-Rezept-Dienstes. Dabei ist für die Schnittstelle ein regelmäßig wechselndes Nutzerpseudonym im äußeren HTTP-Request sichtbar. Dieses unterstützt den Fachdienst E-Rezept bei der Lastverteilung und beim DoS-Schutz auf Applikationsebene. Von der Webschnittstelle erhält die E-Rezept-VAU die verschlüsselten Requests, entschlüsselt und verarbeitet diese. Als Antwort erzeugt die E-Rezept-VAU einen für den Client verschlüsselte HTTP-Response. Den Schlüssel dafür hat der Client für die VAU ephemeral erzeugt und dieser Schlüssel ist Teil des verschlüsselten Requests. Die Webschnittstelle empfängt das Chiffre von der VAU und gibt dieses als Antwort auf den HTTP-Request an den Client zurück.

Ablauf:

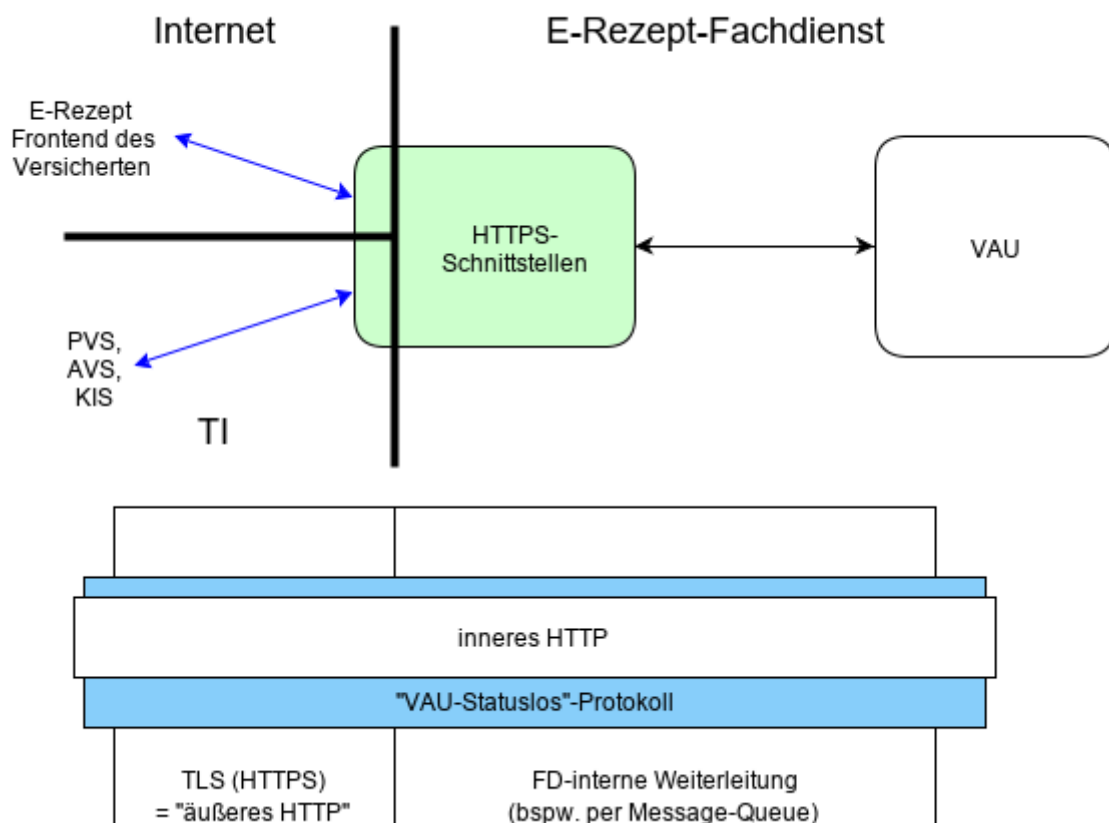
1. Der Client erfragt initial das X.509-Zertifikat der VAU (A\_20160-\*). Je nachdem, ob die X.509-Root der TI oder die TSL als Prüfbasis verwendet wird, erfolgt die Abfrage des VAU-X.509-Zertifikats spezifisch (entweder A\_21216 oder A\_21218). Im Zertifikat ist der öffentliche Verschlüsselungsschlüssel der E-Rezept-VAU enthalten. Dieses Zertifikat kommt aus der Komponenten-PKI der TI und ist langlebig, d. h. der Client kann dieses Zertifikat (inkl. Schlüssel) für folgende Verschlüsselungen lokal vorhalten.
2. Der Client erzeugt einen HTTP-Request mit Request-Body und Request-Header als Datenstrukturen (= innerer HTTP-Request) (A\_20161-\*).
3. Der Client verschlüsselt diesen Request mit dem Verschlüsselungsschlüssel der E-Rezept-VAU. Im Chiffre ist ein Authentisierungstoken des Nutzers enthalten, eine zufällig gewählte Request-ID, ein vom Client ephemeral erzeugter symmetrischer Antwortschlüssel und der am Anfang erzeugte HTTP-Request als Datenstrom (A\_20161-\*).
4. Der Client erzeugt einen neuen (äußeren) HTTP-Request und überträgt darin das Chiffre an eine HTTPS-Schnittstelle des Fachdienstes E-Rezept. Dabei gibt der Client ggf. das schon aus einer vorherigen Response des Fachdienstes E-Rezept mitgeteilte Nutzerpseudonym (A\_20161-\*) mit.
5. Die Webschnittstelle nimmt den Request entgegen und trifft eine Routing-Entscheidung oder eine Priorisierungsentscheidung im Lastfall. Anschließend leitet sie den Request an die VAU weiter (A\_20162).

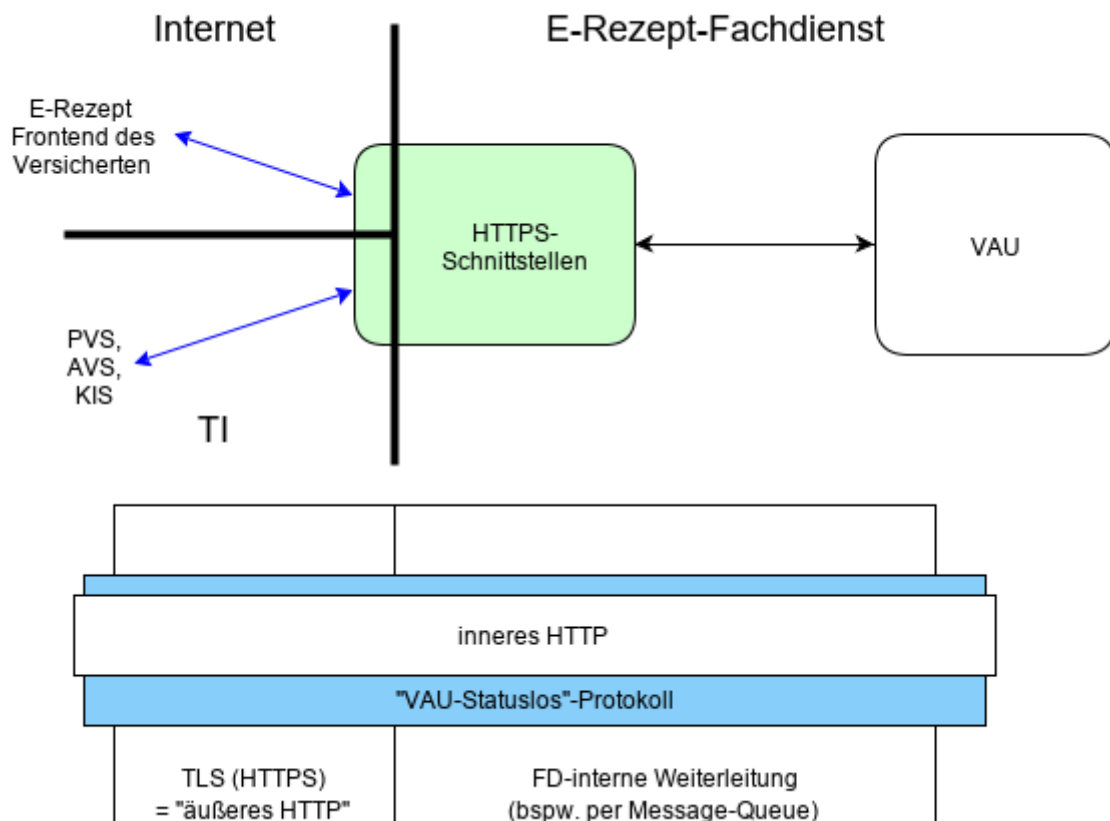


6. Die VAU entschlüsselt den Request und verarbeitet ihn. Die Antwort wird inkl. der Request-ID mit dem vom Nutzer erzeugten Antwortschlüssel verschlüsselt und an die Webschnittstelle gesendet.
7. Die Webschnittstelle antwortet gegenüber dem Client mit diesem Chifftrat auf den HTTP-Request.
8. Der Client entschlüsselt das Chifftrat und prüft die Request-ID in der entschlüsselten Antwort.

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

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





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

Die gematik stellt eine Beispielimplementierung bereit.

## 7-26.3 Definition

### 7-2-16.3.1 E-Rezept-VAU-Identität

#### A\_20160-01 - 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.
5. Der Fachdienst E-Rezept MUSS eine maximal 12 Stunden alte OCSP-Response für das VAU-Zertifikat in seinen Webschnittstellen unter dem Pfad `/VaucertificateOCSPResponse` für Clients abrufbar machen.

**[<=]**

Hinweis: Unter `"/VaucertificateOCSPResponse"` erhält ein Client (einfacher GET-Request) eine korrekte OCSP-Response für das VAU-Zertifikat (A\_20160-\*, Punkt 1). Dies ist analog wie OCSP-Stapling bei TLS zu sehen, nur auf einer höheren OSI-Schicht. Der FD stellt korrekte OCSP-Responses zur Verfügung damit nicht jeder Client selbst den OCSP-Responder fragen muss. Diese Funktionalität hat nichts mit der OCSP-Proxy-Funktionalität zu tun wie sie bspw. beim Zugangsgateway des Versicherten bei ePA angeboten wird. Der FD fragt bspw. stündlich selbst den OCSP-Status für sein VAU-Zertifikat ab, prüft die Antwort und stellt sie unter `"/VaucertificateOCSPResponse"` Clients zur Verfügung.

**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: A\_20967 beschreibt ein analoges Vorgehen wie es bei den SGD(1,2) üblich ist. Der Betreiber des FD besitzt für bestimmte Rollenauthentisierungen für die HSM Authentisierungschipkarten inkl. PIN, die gematik ebenfalls. Nur bei gleichzeitiger Authentisierung beider Beteiligten erlauben die HSM bestimmte Aktionen/Funktionalitäten.

**7.2.26.3.2 Client-seitige Prüfung der E-Rezept-VAU-Identität**

Bevor ein E-Rezept-Client die E-Rezept-VAU-Identität für die Verschlüsselung seiner Request (vgl. [7.2.3- 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 zwei TI-Zertifikate prüfen muss, wird im FdV die technisch deutlich einfachere Zertifikatsprüfung auf Basis der TI-X.509-Root verwendet [gemSpec\_eRp\_FdV#A\_20032-\*], und nicht die Prüfung über die TSL wie im Client eines PVS oder AVS [gemILF\_PS\_eRP#A\_20764]. Beide Prüfwege sind sicherheitstechnisch äquivalent.

### **A\_21216 - E-Rezept-Client, Zertifikatsprüfung auf TLS-Basis**

Ein E-Rezept-Client, der nicht das E-Rezept-FdV ist, MUSS das VAU-Zertifikat vom E-Rezept-FD beziehen (vgl. A\_20160-\*, URL /VAUCertificate) und ebenfalls für dieses Zertifikat die OCSP-[ResponseResponse](#) für dieses Zertifikat beziehen (vgl. A\_20160-\*, URL /VAUCertificateOCSPResponse). Er MUSS das Zertifikat mittels TUC\_PKI\_018 (OCSP-Graceperiod=12h, PolicyList={oid\_erp-vau}) prüfen und dabei die vom FD bezogene OCSP-Response verwenden. [ $\leq$ ]

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 und

Tab\_KRYPT\_ERP\_Algorithmus\_FD\_Zertifikatsliste\_erstellen zwei Hilfsdefinitionen.

**Tabelle 17: Tab\_KRYPT\_ERP\_Zertifikatsliste Definition Datenstruktur Zertifikatsliste**

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

**Tabelle 18: Tab\_KRYPT\_ERP\_Algorithmus\_FD\_Zertifikatsliste\_erstellen**

- (1)  
Als Startpunkt muss eine Datenstruktur nach "Tab\_KRYPT\_ERP\_Zertifikatsliste" (ein Associative-Array) erzeugt werden, bei dem die Array-Elemente jeweils leer sind:  
{ "add\_roots" : [], "ca\_certs" : [], "ee\_certs" : [] }
- (2)  
Im Array "ee\_certs" wird das E-Rezept-VAU-Zertifikat (vgl. A\_20160-\*) base64-kodiert hinzugefügt werden.
- (3)  
Im Array "ee\_certs" werden alle IDP-Signatur-Zertifikate (oid\_idpd [gemSpec\_OID]) die dem FD bekannt sind base64-kodiert am Ende angefügt.  
Die gematik teilt dem E-Rezept-FD regelmäßig die aktuelle vollständige Liste mit.
- (4)  
Im Array "ca\_certs" werden alle Komponenten-CA-Zertifikat base64-kodiert eingetragen, die die Zertifikate aus "ee\_certs" bestätigt haben. Hinweis: Diese Komponenten-CA-Zertifikate findet man in der TSL und ebenfalls auf <https://download.tsl.ti-dienste.de/SUB-CA/>
- (5)  
Falls es in "ca\_certs" CA-Zertifikate gibt die nicht per Signaturprüfung auf den Root-Schlüssel vom RCA3 rückführbar sind, werden nacheinander die chronologisch auf einander folgenden base64-kodierten Root-Cross-Zertifikate (RCA3->RCA4, ... RCA(x)->RCA(x+1)) am Ende des Arrays "add\_roots" angefügt, solange bis alle CA-Zertifikate über einen der Root-Schlüssel prüfbar sind, d. h. per Signaturprüfung auf einen Root-Schlüssel im Array rückführbar sind. Hinweis: dieses Cross-Zertifikate findet man auf <https://download.tsl.ti-dienste.de/ROOT-CA/>

Hinweis: zum produktiven Start des E-Rezept-FD enthält das Array "ee\_certs" drei Elemente (Zertifikat des VAU-Protokolls, Signaturzertifikat des Discovery-Dokumentes, Signaturzertifikat des ID\_Token), das Array "ca\_certs" ein Element und das Array "add\_roots" null Elemente.

## **A\_21217 - E-Rezept-FD, Zertifikatslisten und OCSP-Response für Clients**

Ein E-Rezept-FD MUSS

1. nachdem die E-Rezept-VAU-Identität (vgl. A\_20160-\*) erzeugt wurde mittels des Algorithmus aus Tab\_KRYPT\_ERP\_FD\_Zertifikatsliste\_erstellen die Datenstruktur analog zu Tab\_KRYPT\_ERP\_Zertifikatsliste erstellen.
2. diese erzeugte Zertifikatsliste auf seiner Webschnittstelle (HTTPS) über die URL /CertList (GET-Methode, Reponse-Content-Type 'application/json') verfügbar machen.
3. die Zertifikatsliste mittels des Algorithmus aus Tab\_KRYPT\_ERP\_FD\_Zertifikatsliste\_erstellen aktualisieren wenn (1) die E-Rezept-VAU-Identität aktualisiert wurde (Schlüsselwechsel der VAU etc.) oder (2) die gematik neue IDP-Zertifikate über die Liste E-Rezept-Clients bekannt machen möchte. Im Fall (2) meldet die gematik dies an den FD.
4. an seiner Webschnittstelle (HTTPS) über die URL /OCSPList (GET-Methode, Reponse-Content-Type 'application/json') ein Array mit den base64-kodierten OCSP-Responses für die alle Zertifikate aus dem Array "ee\_certs". Diese OCSP-Response-Liste muss der FD mindestens alle 11 Stunden mit "frischen" OCSP-Responses für die Zertifikate aus dem Array "ee\_certs" aktualisieren.

[<=]

**Tabelle 19: Tab\_KRYPT\_ERP\_FdV\_Truststore\_aktualisieren**

- (1)  
Alle Zertifikate im Truststore müssen gelöscht werden mit Ausnahme von RCA3.  
Der Truststore besitzt, wie in A\_21218 definiert, Prüfschlüssel/Zertifikate in vier Kategorien (A) Root-Schlüssel, (B) CA-Zertifikate, (C) E-Rezept-VAU-Zertifikat, (D) IDP-Zertifikat(e).
- (2)  
Falls die Zertifikatsliste im Array "add\_roots" nichtleer ist, so wird das erste Cross-Zertifikat im Array auf Basis von RCA3 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" 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 jedes Zertifikat im Array "ee\_certs" 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.

**Tabelle 20: Tab\_KRYPT\_ERP\_OCSP\_Liste Definition Datenstruktur OCSPList**

```
{  
  "OCSP Responses": [ "base64-kodiertes-OCSP-Response-EE-Zertifikat-1",  
    ... ]  
}
```

#### **A\_21218 - E-Rezept-Client, Zertifikatsprüfung auf Basis der X.509-Root**

Das E-Rezept-FdV MUSS die RCA3 (das Zertifikat der Version 3 der X.509-Root der TI) als Vertrauensanker im Programm-Code bzw. mit dem Programm-Code fest assoziiert enthalten und als Basis für die Prüfung von TI-Zertifikat verwenden.

Das FdV MUSS einen TI-Zertifikate-Truststore enthalten und pflegen, wie folgend definiert.

Der Truststore MUSS Prüfschlüssel/Zertifikat aufgeteilt in folgende vier Kategorien enthalten: (A) Root-Schlüssel, (B) CA-Zertifikate, (C) E-Rezept-VAU-Zertifikat, (D) IDP-Zertifikat(e). Initial kann dieser Truststore nur RCA3 enthalten oder die Zertifikate die mittels Tab\_KRYPT\_ERP\_FD\_Zertifikatsliste\_erstellen ermittelt werden.

Falls im Truststore keine Zertifikate für Kategorie (C) und (D) vorliegen, so MUSS das FdV den Truststore aktualisieren indem es über den FD (URL /CertList) die Zertifikatsliste lädt und diese mittels des Algorithmus

Tab\_KRYPT\_ERP\_FdV\_Truststore\_aktualisieren prüft und ggf. in den Truststore lädt.

Das E-Rezept-FdV MUSS über den FD (URL /OCSPList) OCSP-Responses für die Zertifikate (C) und (D) beziehen, wenn aktuell keine OCSP-Responses für diese Zertifikate im FdV vorliegen, die jünger als 12 Stunden sind.

Falls in der OCSP-Liste OCSP-Responses enthalten sind die zu keinem der Zertifikate (C) und (D) passen, so MUSS das FdV den Truststore aktualisieren (s. o.).

Zertifikate aus (C) und (D) MÜSSEN OCSP-Responses, die jünger als 12 Stunden sind besitzen, damit diese Zertifikate in fachliche Use-Cases im FdV verwendet werden können.

Die OCSP-Responder-Zertifikate MÜSSEN per Signaturprüfung auf ein Zertifikat der Kategorie (B) rückführbar sein, ansonsten MÜSSEN die entsprechenden OCSP-Responses verworfen werden.

Das FdV MUSS bei der Prüfung der TI-Zertifikate in fachlichen Use-Cases im FdV, prüfen ob das Zertifikat im oben beschriebenen Truststore enthalten ist und eine gültige OCSP-Response enthält die jünger als 12 Stunden ist. Falls dies nicht so ist, so ist das Ergebnis der Prüfung des TI-Zertifikats FAIL.

[<=]

#### **A\_21222 - 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 (vgl. A\_21216 und A\_21218). [<=]

### **7.2.36.3.3 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] durchführen. Das somit erzeugte gemeinsame Geheimnis ist Grundlage für die folgende Schlüsselableitung.
  - b. Als Schlüsselableitungsfunktion MUSS er die HKDF nach [RFC-5869] auf Basis von SHA-256 verwenden.
  - c. Dabei MUSS er den Ableitungsvektor "ecies-vau-transport" verwenden, d. h. in der Formulierung von [RFC-5869] info="ecies-vau-transport" .
  - d. Er MUSS mit dieser Schlüsselableitung einen AES-128-Bit Content-Encryption-Key für die Verwendung von AES/GCM ableiten.
  - e. Er MUSS für Verschlüsselung mittels AES/GCM einen 96 Bit langen IV zufällig erzeugen.
  - f. Er MUSS mit dem CEK und dem IV mittels AES/GCM p verschlüsseln, wobei dabei ein 128 Bit langer Authentication-Tag zu verwenden ist.
  - g. Er MUSS das Ergebnis wie folgt kodieren: chr(0x01) || <32 Byte X-Koordinate von öffentlichen Schlüssel aus (a) > || <32 Byte Y-Koordinate> || <12 Byte IV> || <AES-GCM-Chiffre> || <16 Byte AuthenticationTag> (vgl. auch Tab\_KRYPT\_ERP und folgende die Beispielverschlüsselung).  
Die Koordinaten sind (wie üblich) vorne mit chr(0) zu padden solange bis sie eine Kodierungslänge von 32 Byte erreichen.
7. Er erzeugt einen HTTPS-Request an den FD mit der POST-Methode und dem Pfad /VAU/<Nutzerpseudonym>[/optional-beliebiger-weiterer-URL-Pfadteil] mit dem Content-Type 'application/octet-stream' und sendet diesen an die Webschnittstelle des FD.  
"Nutzerpseudonym" MUSS eine ggf. aus der vorherigen (zeitlich letzten) Antwort des FD dem Nutzer übergebene URL-sichere Zeichenkette sein (bspw. ein 128 Byte langer Hexadezimal-Kode).

Falls dem Client kein Nutzerpseudonym vorliegt so MUSS er "0" als Nutzerpseudonym verwenden.

[<=]

**Tabelle 21: Tab\_KRYPT\_ERP Kodierung des Chiffrats aus A\_20161-\***

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

Hinweise zu A\_20161-\*:

1. Beispiel für eine hexadezimalkodierte Request-ID nach A\_20161-\* (2):  
"32b6594cc29bb54a14cb2a8e09558817"
2. Beispiel für einen 128-Bit-AES-Schlüssel nach A\_20161-\* (3):  
"a576daad8096fc52987250a8e7eb9bd3"
3. Aus kryptographischer Sicht könnte ein Client den Antwort-AES-Schlüssel (A\_20161-\* Punkt (3)) auch für weitere Requests verwenden. Dies würde jedoch erzwingen, dass es im Client eine komplexere Überwachung des Lebenslaufs des Schlüssels gibt (Wann wurde er erzeugt, d. h. wann muss er gewechselt werden etc.). Um dies zu verhindern und die Implementierung im Client einfacher zu halten, gibt es A\_20161-\* Punkt (4).

Beispielverschlüsselung (Testvektor):

Der Langzeit-VAU-Schlüssel nach A\_20160-\* sei folgender:

x=0x8634212830dad457ca05305e6687134166b9c21a65ffebf555f4e75dfb048888  
y=0x66e4b6843624cbda43c97ea89968bc41fd53576f82c03efa7d601b9facac2b29

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

Der Client erzeugt zufällig den privaten ECC-Schlüssel

d=5bbba34d47502bd588ed680dfa2309ca375eb7a35ddb6d67cc7f8b6b687a1c1d

Damit ist dessen öffentlicher ephemerer ECDH-Schlüssel:

x=0x754e548941e5cd073fed6d734578a484be9f0bbfa1b6fa3168ed7fffb22878f0f

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 Chifftrat:

86c2b491c7a8309e750b und folgenden 16 Byte großen Authentication

Tag 4e6e307219863938c204dfe85502ee0a

Das Gesamt-Chifftrat 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 erhaltenen 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 kodierten 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 überschreiben.)
3. Er MUSS das Antwort-Chifftrat mit den Vorgaben aus A\_20163 (9) und dem AES-Antwort entschlüsseln und prüfen ob die Entschlüsselung erfolgreich möglich war.
4. Er MUSS prüfen, ob die Struktur des erhaltenen Klartextes p der Struktur aus A\_20163 (8) entspricht.
5. Er MUSS prüfen, ob die Request-ID in p der Request-ID aus dem Client-Request entspricht (Gleichheit prüfen).
6. Er MUSS das dritte Feld-Element in p ("Response-Header-und-Body") als HTTP-Antwort der E-Rezept-VAU in fachlich weiter verarbeiten.

[<=]

#### **A\_20175 - E-Rezept-Client, Speicherung Nutzerpseudonym**

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

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

**Tabelle 22: Tab\_KRYPT\_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept)**

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

### **7.2.46.3.4 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"
```

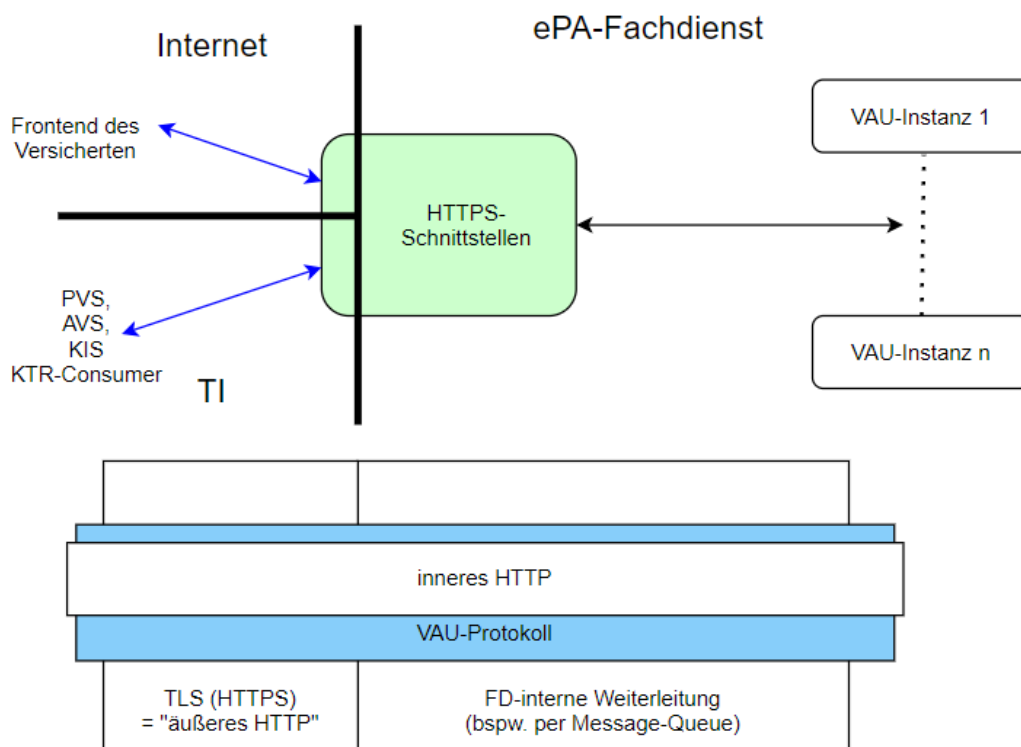
**[<=]**

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

## 7 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.

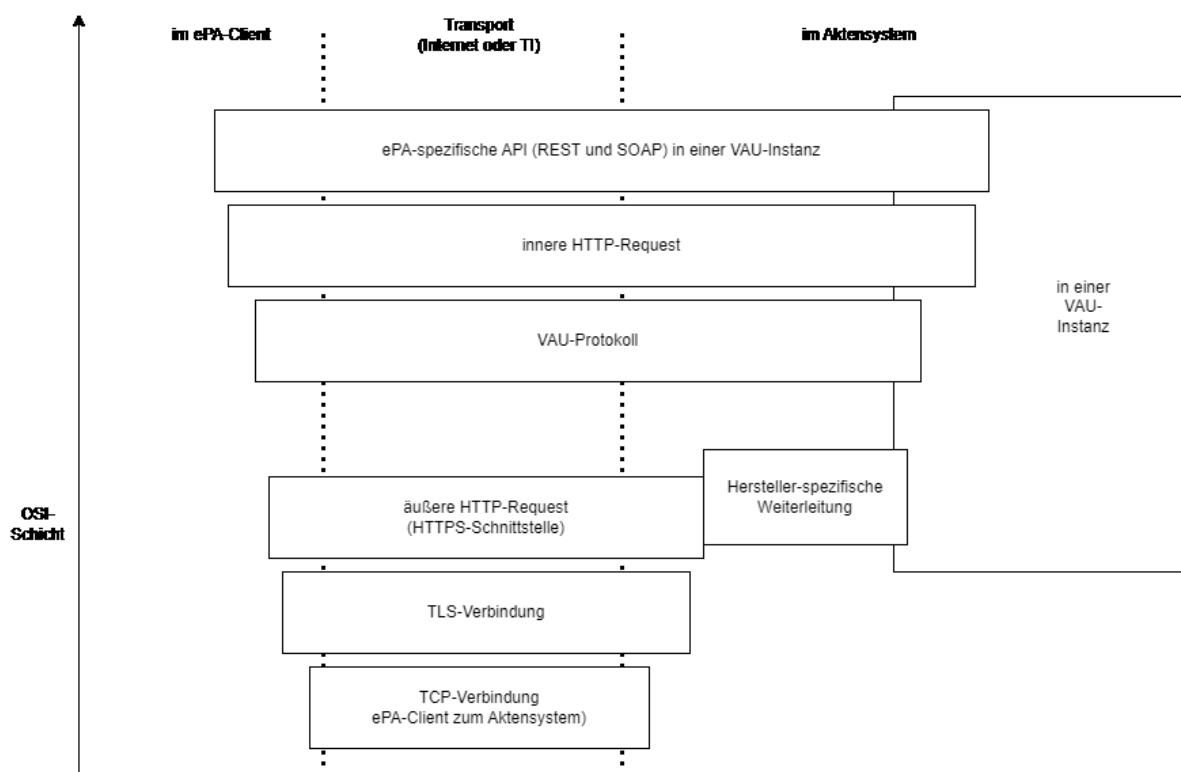
Wenn über eine HTTPS-Schnittstelle ein Request eines ePA-Clients eintrifft, steht es einem Aktensystem frei, über welches Protokoll es diesen Request an die VAU-Instanzen verteilt (bspw. per Message-Queue), d. h. die Aktensysteminterne Verbindungsstrecke zwischen HTTPS-Schnittstellen und VAU-Instanzen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

Hinweis: Im Kontext ePA bietet HTTP/2 nur geringe Vorteile gegenüber HTTP/1.1. HTTP/2 besitzt jedoch eine deutlich höhere Komplexität. Ein Hersteller ist mit A\_24654-\* nicht verpflichtet, eine HTTP-Version größer als 1.1 zu implementieren. A\_24654 ist nicht dahingehend zu verstehen, dass von einem Aktensystem perfekte Konformität zu HTTP/1.1 umzusetzen ist. Im Kontext der ePA-Funktionalität wird nur der Umfang von HTTP verwendet, den man normaler Weise bei Webservices verwendet.

Wenn im folgenden innerhalb dieses Kapitels 7. vom VAU-Protokoll gesprochen wird, ist stets die Variante für ePA für alle gemeint.



**Abbildung 5: OSI-Schichten in einem Aktensystem**

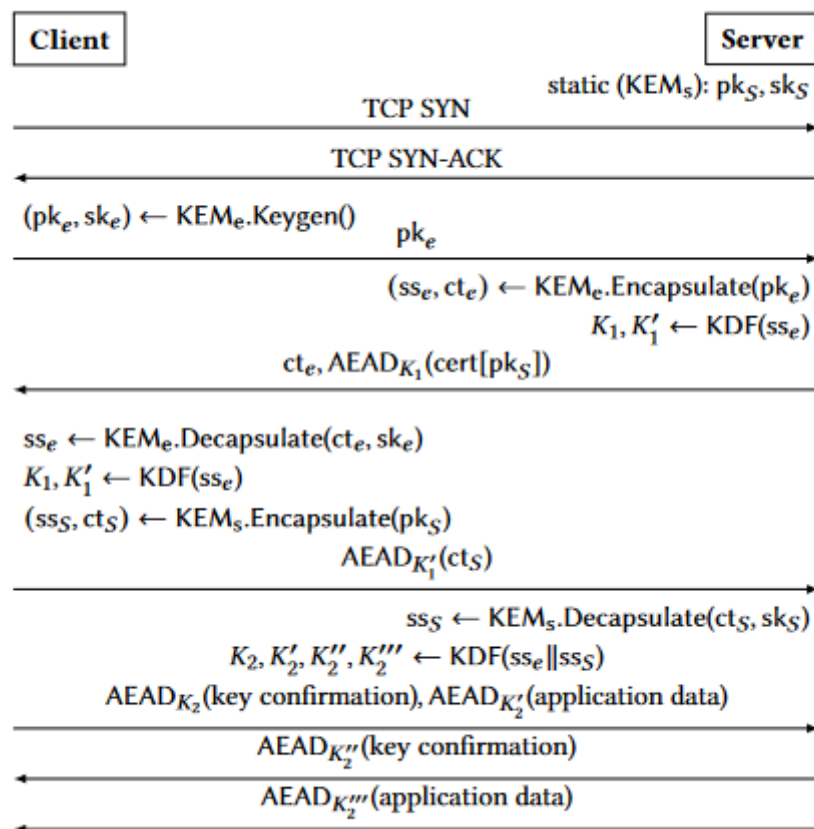
Wie im OSI-Referenzmodell üblich, ist das VAU-Protokoll agnostisch gegenüber dem über ihm transportierten Daten oder Protokollen. Das VAU-Protokoll ermöglicht einen Ende-zu-Ende gesicherten Transport von Daten, bei dem es irrelevant ist, welche Protokolle unter oder über ihm in Bezug auf das OSI-Modell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

#### **A\_24656 - ePA, Unabhängigkeit von TLS-Ebene und VAU-Protoll-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-Protoll-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 authentifizierte 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 authentifizierte 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 Abschnitt 6 (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 6 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.

*Hinweis, der nach der Kommentierung in der finalisierten gemSpec\_Krypt entfernt wird:  
Aktuell befindet sich die Beispielimplementierung  
bei [https://bitbucket.org/andreas\\_hallof/vau-protokoll/src/master/](https://bitbucket.org/andreas_hallof/vau-protokoll/src/master/)  
Sie wird später wie üblich nach github überführt.  
Der reine kryptographische Teil eines Verbindungsaufbaus (Schlüsselaushandlung)  
benötigt ca. 100ms (vgl. Performanz-Test im Repository) und entspricht den üblichen  
Zeiten bei PQC-Hybrid-TLS.  
Die Minimal-Variante in der Beispielimplementierung von Client- und VAU-Code hat  
zusammen unter 250 Zeilen Code (mit Verwendung einer Standard-Krypto-Bibliothek  
für die kryptographischen Verfahren). Es handelt sich also um ein relativ einfaches und  
damit relativ leicht zu implementierendes Protokoll.*

## 7.1 Übersicht Verbindungsaufbau/Schlüsselaushandlung

Bei im Prinzip allen in der Praxis verwendeten kryptographischen Protokollen gibt es mehrere Phasen im Protokoll: Zunächst erfolgt eine Schlüsselaushandlung bei der

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

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 | <p>Client -&gt; VAU</p> <p>Ein Client erzeugt zwei Schlüsselpaare (ECDH, Kyber768) (analog zu [IETF-Hybrid-TLS]) und sendet die beiden öffentlichen Schlüssel an die VAU (= Nachricht 1).</p> <p>Bei diesen Schlüsselpaaren handelt es sich also um ephemere Schlüssel des Clients, d. h. pro Verbindungsaufbau/Handshake werden diese vom Client neu erzeugt.</p>  |
| Nachricht 2 | <p>VAU -&gt; Client</p> <p>Die VAU verwendet die öffentlichen Schlüssel jeweils mittels eines KEM (Encapsulate), um zwei gemeinsame Geheimnisse (ss_e_ecdh, ss_e_kyber768) und zwei KEM-Chiffre zu erzeugen. Diese Geheimnisse fließen in eine KDF (RFC-5859/SHA-256) ein (Basis: ss_e = ss_e_ecdh    ss_e_kyber768) und damit werden die symmetrischen Schlüssel K1_c2s und K1_s2c erzeugt. Wie bei TLS 1.3 üblich, werden diese Schlüssel im weiteren für die Sicherung von Daten in der Aushandlungsphase des Protokolls für die Kommunikation zum Server (K1_c2s) und zum Client (K1_s2c) verwendet.</p> <p>Die KEM-Chiffre und über AES/CGM verschlüsselte semi-statische VAU-Schlüssel (A_24425-*) werden an den Client gesendet (= Nachricht 2).</p> <p>Verständnishinweis: Die authentifizierte Verschlüsselung der öffentlichen VAU-Schlüssel (A_24425-*) erfolgt aufgrund des Grundprinzips bei TLS 1.3 möglichst früh, um Daten auch in der Handshake-Phase zu verschlüsseln. Die öffentlichen VAU-Schlüssel (A_24425-*) sind nicht vertraulich.</p> |

| Nachricht   | Aktion  |
|-------------|---|
| Nachricht 3 | <p>Client -&gt; VAU</p> <p>Mittels der KEM-Chifftrate aus Nachricht 2 und der privaten Schlüssel des Clients (vgl. Nachricht 1: ECDH, Kyber768) berechnet der Client K1_c2s und K1_s2c und kann die öffentlichen VAU-Schlüssel erfolgreich entschlüsseln (AEAD-Chiffre). Er verwendet die öffentlichen VAU-Schlüssel (ECDH, Kyber768) und erhält mittels zwei KEM-Encapsulate-Berechnungen zwei Geheimnisse und zwei KEM-Chifftrate. Er berechnet aus den zwei Geheimnissen: <math>ss_s = ss_s\_ecdH \parallel ss\_s\_kyber768</math>. Diese KEM-Chifftrate verschlüsselt er über AES/GCM mittels des Schlüssels K1_c2s und erhält ein Ergebnis-Chifftrat.</p> <p>Mittels einer KDF auf Basis von <math>ss = ss\_e \parallel ss\_s</math> werden abgeleitet: vier vertrauliche AES/GCM-Schlüssel:</p> <ul style="list-style-type: none"> <li>• K2_c2s_key_confirmation,</li> <li>• K2_c2s_app_data,</li> <li>• K2_s2c_key_confirmation, und</li> <li>• K2_s2c_app_data</li> </ul> <p>und eine (nicht vertrauliche) KeyID.</p> <p>Der Client erzeugt den Hashwert des Transskript der zuvor ausgetauschten Daten (analog TLS). Diesen Hashwert verschlüsselt er mittels des Schlüssels K2_c2s_key_confirmation (Ziel: Key-Confirmation).</p> <p>Der Client sendet das Ergebnis-Chifftrat (K1_c2s) und das Key-Confirmation-Chifftrat (K2_c2s_key_confirmation) an die VAU (= Nachricht 3).</p> |

| Nachricht   | Aktion  |
|-------------|---|
| Nachricht 4 | <p>VAU -&gt; Client</p> <p>Mittels der zwei KEM-Chifftrate aus Nachricht 3 und der KEM-Decapsulation-Funktionen werden die beiden Geheimnisse <code>ss_s_ecdh</code> und <code>ss_s_kyber768</code> berechnet, und anschließend <code>ss_s = ss_s_ecdh    ss_s_kyber768</code> berechnet. Analog wie im Client bei Nachricht 3 werden die dort aufgeführten vier vertraulichen AES/GCM-Schlüssel und die nicht-vertrauliche KeyID abgeleitet auf Basis von <code>ss = ss_e    ss_s</code>.</p> <p>Mit Hilfe des Schlüssels <code>K2_c2s_key_confirmation</code> und der im Server gespeicherten vorherigen Nachrichten prüft der Server das vom Client gesendete Transskript-Chifftrat bzw. den Transskript-Hash darin.</p> <p>Der Server erzeugt selbst einen Transskript-Hash (Hashwert der Konkatenation Nachricht 1    Nachricht 2    Nachricht 3) und sendet diesen mittels <code>K2_s2c_key_confirmation</code> verschlüsselt an den Client zurück.</p> |

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 - 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-Bit-big-endian,
          "y" : Binärwert-x-Koordinate-32-Bit-big-endian,
        },
    "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 [CBOR-RFC] serialisiert/kodiert werden und per Mime-Type "application/cbor" [RFC-CBOR] (HTTP Content-Type) an den VAU-Client als Response auf den GET-Request gesendet werden.

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

[<=]

Erläuterung:

In A\_24425-\* werden die "signed\_pub\_keys" mit dem Schlüsselmaterial der AUT-VAU-Identität des Aktensystems authentisiert, analog zum VAU-Protokoll ePA 1.x und 2.x. Dieses Zertifikat ist über die Signaturkettenprüfung der TI-PKI prüfbar. Dafür benötigt der VAU-Client die Zertifikate, die die Prüfkette ausmachen. Als Vertrauensanker besitzt der Client entweder X.509-Root-Zertifikate, und zwar mindestens RCA5 (PU), oder die TSL. Die Konstruktion nach A\_24957-\* stellt die Zertifikate für die Prüfkette und das AUT-VAU-Zertifikat selbst ("cert"-Feld) zur Verfügung. Aufgrund der Konstruktion kann man davon ausgehen, dass die Daten in solch einer Datei sich nie ändern werden, i. S. v. einmal vom Client per GET bezogen werden und dann ewig gespeichert werden kann. Sollte sich nach einem Deployment der CertData-Datei (i. S. v. einige Clients haben diese Datei schon bezogen) bspw. CertData-e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855-1 herausstellen, dass dort Fehler enthalten sind, dann kann man über die Versionsnummer im Handshake (vgl. "cdv"-Feld, A\_24425-\*) und der Erstellung einer neuen CertData-e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855-2 diese Fehler im laufenden Betrieb korrigieren.

#### **A\_24958 - VAU-Protokoll: VAU-Client Prüfbasis Zertifikatsprüfung**

Ein VAU-Client (PVS, ePA-FdV etc.) MUSS als Prüfbasis eine Root-Version (X.509-Root-TI-Zertifikat), die mindestens 2 Jahre alt ist, verwenden oder die TSL. Der VAU-Client MUSS die Operation nach A\_24957-\* verwenden, um die für die Zertifikatsprüfung von A\_24425-\* notwendigen Zertifikate zu beziehen. Die bezogenen Zertifikatsdaten MUSS der VAU-Client lokal (zeitlich unbegrenzt) vorhalten (Caching). Bei Erhalt einer Nachricht 2 (A\_24608-\*) MUSS er zunächst prüfen, ob er die für die Zertifikatsprüfung notwendigen Zertifikate im lokalen Cache vorrätig hat, und falls ja MUSS er diese verwenden.

[<=]

#### **A\_24427 - VAU-Protokoll: VAU-Schlüssel für die VAU-Protokoll-Schlüsselaushandlung**

Ein Aktensystem KANN in Bezug auf die Erzeugung der signierten öffentlichen VAU-Schlüssel (vgl. A\_24425-\*) und der Erzeugung und der Vorhaltung der dazu passenden zwei privaten Schlüssel (ECDH, Kyber768) zwischen zwei Modellen wählen:

(1)

Im Normalfall wird eine VAU-Instanz "auf Vorrat" im Aktensystem gestartet. Beim Starten der Instanz erzeugt die Instanz die beiden Schlüsselpaare und signierte diese mittels der im VAU-HSM befindlichen Signaturidentität (gemäß A\_24425-\*). Die privaten Schlüssel bleiben genau nur in dieser VAU-Instanz.

(2)

Eine VAU-Instanz erzeugt regelmäßig die Schlüsselpaare und anschließend die

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 - VAU-Protokoll: Nutzerpseudonym**

Ein VAU-Client MUSS die HTTP-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]) den inneren HTTP-Response-Header prüfen, ob die Variable "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 7.5- 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-Chiffre || 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-5859] 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\_2416) heißt aed\_ciphertext\_msg\_2.

Sie MUSS folgende Datenstruktur erzeugen:

```
{
  "MessageType" : "M2",
  "ECDH_ct"      : ... analog ECDH_PK aus A_24488-* ...,
  "Kyber768_ct"  : Kyber768-Ciphertext analog [IETF-Kyber768], 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. [ $\leq$ ]

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. [ $\leq$ ]

#### **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\_24488 (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-5859] 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 [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 Chifftrat sei als aead\_ciphertext\_msg\_3\_key\_confirmation hier bezeichnet.

Dann MUSS er folgende Datenstruktur erzeugen:

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

Diese Datenstruktur MUSS er per CBOR serialisieren, das Ergebnis ist Nachricht 3. Er MUSS die Nachricht 3 per äußeren HTTP-Request an das Aktensystem senden und dabei den Wert der VAU-CID (vgl. A\_24622) als URL-Pfadnamen verwenden, unter Verwendung der HTTP-POST-Methode.

[<=]

#### **A\_24624 - 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. Hat er bei einem der folgenden Prüfpunkte kein positives Prüfergebnis, MUSS er die Verarbeitung (Handshake) abbrechen.

1. Prüfung des TI-Zertifikats aus dem "cert"-Feld 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 Zertifikat MUSS aus der aus der Komponenten-PKI der TI stammen und die Rollen-OID "oid\_epa\_vau" besitzen.

Das VAU-Zertifikat MUSS zeitlich gültig sein. Es MUSS kryptographisch in einer Zertifikats-/Signaturprüfungskette rückführbar auf eine X.509-Root-Version der TI-PKI sein. [ <= ]

#### **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 Chifftrat 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\_24635-\* das Transskript des Clients und dessen SHA-256-Wert berechnen. Sie MUSS mittels K2\_c2\_key\_confirmation das Chifftrat "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 Konkretion 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 Chifftrat genannt "AEAD\_ct\_key\_confirmation".

Sie MUSS die folgende Datenstruktur erzeugen:

```
{  
  "MessageType" : "M4",  
  "AEAD_ct_key_confirmation" : Chifftrat-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. [ $\leq$ ]

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 Chifftrat "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. [ $\leq$ ]

## **7.2 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 (Chifftrat) ggf. nicht übermittelt werden konnte, der Zähler MUSS bei jeder neuen Nutzung um eins erhöht werden. [ $\leq$ ]

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

#### **A\_24628 - 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  $IV=a \parallel \text{Zähler}$  erzeugt. 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 Chiffre 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.<br>In der Kodierung des Response-Chiffre hat dieses Byte den Wert 2, was markiert, dass es sich um eine Response handelt. |
| Request-Counter  | 8 Byte  | Eindeutige Zählernummer für diesen Request<br>Für jeden neuen Request wird vom Client dieser Wert um eins erhöht.   |
| 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 erweiterte Chiffre (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 Chiffre 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.<br>In der Kodierung des Response-Chiffre 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.   |
| KeyID            | 32 Byte | KeyID aus dem Handshake (vgl. A_24623-*)  |

| Name      | Länge                     | Beschreibung bzw. Vorgabe des Werts   |
|-----------|---------------------------|---|
| IV        | 12 Byte<br>(= 96<br>Bit)  | IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A_24628-*)                                      |
| CT        | variabel                  | eigentliche AES/GCM-Chiffre, dessen Länge gleich der Länge des Klartextes ist   |
| GMAC-Wert | 16 Byte<br>(= 128<br>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 Chiffrats 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 (Chifftrat) 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 Chifftrat (vgl. A\_24628-\*, also Chifftrat inkl. Header) MUSS an den Client im HTTP-Response-Body versendet werden, wobei das Aktensystem den Media-Type 'application/octet-stream' (HTTP Content-Type) verwenden MUSS.

[<=]

#### **A\_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 Chifftrat) 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.

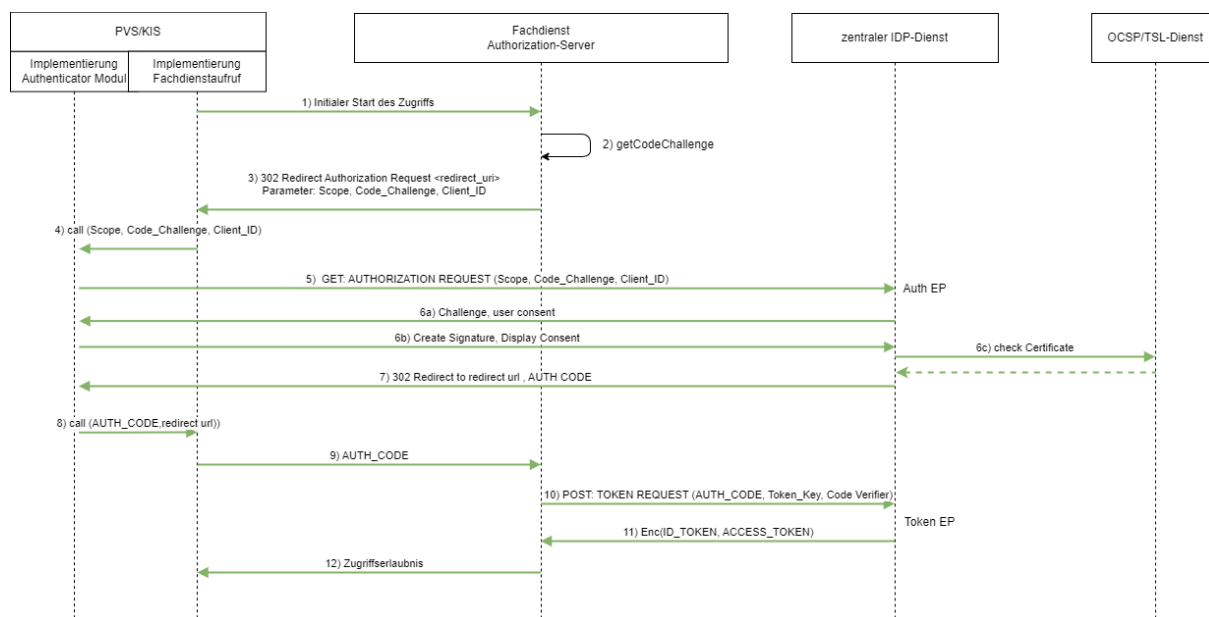
Ergibt eine der Prüfungen ein nicht-positives Ergebnis, MUSS der Client die Verarbeitung der Response abbrechen.

Er MUSS das Chifftrat mit dem mit der KeyID verbundenen Schlüssel K2\_s2c\_app\_data mittels AES/GCM entschlüsseln und dabei den Header als "Additional Associated Data" in die Entschlüsselung mit einfließen lassen. Ergibt die Entschlüsselung das Symbol "FAIL", so MUSS er die Verarbeitung des Chiffrats/Response abbrechen.

[<=]

## 7.3 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 authentifizierte und vertrauliche Datenverbindung besteht zwischen ePA-Client und VAU-Instanz. Über diese Datenverbindung findet auf einer höheren OSI-Schicht eine Client-Authentisierung statt. Dafür werden vom Client innere HTTP-Requests über das VAU-Protokoll an die VAU-Instanz gesendet. Im OIDC Fall zwei Stück:

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

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

#### **A\_24634 - VAU-Protokoll: VAU-Instanz: notwendige Authentifizierung des VAU-Clients (Nutzers)**

Eine VAU-Instanz MUSS sicherstellen, dass, solange der ePA-Nutzer noch nicht authentisiert ist, keine ePA-Funktionalität (ePA-spezifische APIs, die nicht zur weiteren Client-Authentisierung dienen) vom ePA-Client verwendbar sind (siehe Erläuterung vor A\_24634-\*).[<=]

#### **A\_25055 - VAU-Protokoll: OIDC Authentisierung oberhalb der VAU-Protokoll-Schicht**

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

Nach erfolgreicher Authentisierung über OIDC MUSS die VAU-Instanz in den Metadaten der Verbindungsschlüssel (KeyID, K2\_\*\_app\_data) den neuen Authentisierungszustand vermerken (Rückwirkung auf A\_24634-\*).

Nach erfolgreicher Authentisierung über OIDC, genauer innerhalb des Response-Headers (innerer Request) nach dem POST-Request für den Authentication-Codes durch den Client, MUSS sie ein Nutzerpseudonym dem Client in der HTTP-Variable "VAU-NP" verfügbar machen, 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 (Response-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)",
  "Connnection-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#A\_25006-\* gelten. [≤]

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

## 7.4 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 - VAU-Protokoll: PKI-basierte Client-Authentisierung**

Eine VAU-Instanz MUSS über einen inneren HTTP-Request zwei Operation wie folgt anbieten.

(1) Über den Pfadnamen /FrischeParameter MUSS die VAU-Instanz Daten per HTTP-GET bereitstellen.

Diese Daten sind eine base64-kodierte Zeichenkette. Der Media-Type (Context-Type) MUSS 'text/plain;charset=UTF-8' 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#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 den Pfadnamen /PKI-Aut MUSS sie die Möglichkeit anbieten, per HTTP-POST ein Authentisierungstoken an die VAU-Instanz zu senden.

Die Authentisierungstoken sind JWT [RFC-7519], deren Body wie folgt aufgebaut ist:

```
{
  "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 - 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, prüft, ob das "authentisierende" EE-AUT-Zertifikat des Nutzers

1. professionItem und professionOID gemäß [gemSpec\_OID#GS-A\_4446-\*] oid\_erp-vau und 1.2.276.0.76.4.258 als Attribute besitzt, und
2. aus einer Komponenten-PKI-CA der TI-PKI stammt (vgl. Implementierungshinweis).

[<=]

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 Nutzerauthentifizierung 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. [ $\leq$ ]

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. Abschnitt 7.5- 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).

## **7.5 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 - VAU-Protokoll: VAU-Instanz, Nutzerpseudonym erzeugen**

Nach erfolgreicher Nutzer-Authentisierung des Clients MUSS die VAU-Instanz ein Nutzerpseudonym (NP) erzeugen. Dieses NP MUSS, im HTTP-Response-Header des

inneren HTTP-Request unter dem Variablennamen "VAU-NP" eingetragen werden und so an den Client übertragen werden. [ $\leq$ ]

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). [ $\leq$ ]

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 [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. [ $\leq$ ]

#### **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-Protokoll-Verbindungsaufbau durchlaufen (inkl. Nutzer-Authentisierung gegenüber der VAU-Instanz). [ $\leq$ ]

## **7.6 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 [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"<br>Fehler in einer Kodierung bspw. der CBOR-Kodierung   |
| 400        | 2         | "Missing Parameters"<br>notwendige Datenfelder bspw. in der Handshake fehlen   |
| 403        | 3         | "GCM returns FAIL"<br>Eine AES/GCM-Entschlüsselung ergibt das Symbol "FAIL".   |
| 403        | 4         | "PU/nonPU Failure"   |
| 403        | 5         | "Transscript Error"<br>Im Handshake wird Ungleichheit zwischen den beiden<br>Transskript-Hashwerten festgestellt.          |
| 400        | 6         | "bad format: extended ciphertext"<br>Die erste Sanity-Prüfung des erweiterten Chifftrat bspw. bei<br>A_24630 schlägt fehl. |
| 403        | 7         | "is not a request"<br>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.[<=]

## **7.7 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#5.8. 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 Chiffprat kann eine VAU-Instanz sicher feststellen, für welche Umgebung ein Client ein Chiffprat erzeugt hat. Ein VAU-Instanz lehnt in der Produktivumgebung dann solche Nichtproduktivumgebungs-Chifftrate ab.

#### **A\_24477 - VAU-Client, Nichtproduktivumgebung, Offenlegung von symmetrischen Verbindungsschlüsseln**

Ein VAU-Client in einer Nichtproduktivumgebung MUSS nach einer erfolgreichen Schlüsselaushandlung mit dem VAU-Protokoll die ausgehandelten symmetrischen Schlüssel K2\_c2s\_app\_data und K2\_s2c\_app\_data base64-kodiert innerhalb einer HTTP-Request-Header-Variable "VAU-nonPU-Tracing" bei jedem äußeren HTTP-Request aufführen. Dabei sind die beiden Base64-kodierten Schlüsselwerte durch Leerzeichen zu trennen. Beispiel:

```
VAU-nonPU-Tracing: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=
```

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 Chiffprat 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 (Chiffprat im Request-Body des äußeren Requests) keine VAU-Instanz erreicht.

Eine VAU-Instanz in der Produktivumgebung, die am Chiffprat am nonPU/PU-Byte erkennt (vgl. A\_24628-\*), dass es sich um ein nonPU-Chiffprat handelt, MUSS den Request mit einer Fehlermeldung an die äußere Schicht des Aktensystems (Webschnittstelle) beantworten. Die VAU-Instanz MUSS sicherstellen, dass das Chiffprat nicht entschlüsselt und das Chiffprat in der VAU-Instanz sicher gelöscht wird. [ <= ]

### **7.7.1 Zufallsquelle für Clients**

#### **A\_24921 - ePA-Aktensystem, Random-Operation**

Ein Aktensystem MUSS an seiner äußeren HTTPS-Schnittstellen eine Random-Operation, wie in A\_21215-\* definiert, zur Verfügung stellen. [ <= ]

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

---

## **8 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-2020] und [BSI-TR-03116-1]), nicht in Frage stehen.

Schwieriger wird es mit der Verschlüsselung mit klassischen asymmetrischen Verfahren bei denen Chiffre 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").

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 ~~2023/24 einen geeigneten Stand erreichen werden. Bei TI-spezifischen Protokollen (VAU-Protokoll, SGD-Protokoll) evaluiert die gematik aktuell PQC-Protokollerweiterungen die kryptographisch denen bei TLS [TLS-CC-2021] entsprechen (vgl. Abschnitt "6.14 Optionale PQC-Erweiterung (informativ)").~~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-PQC] 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 Erläuterungen (informativ)**

---

### **9.1 Prüfung auf angreifbare (schwache) Schlüssel**

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

### **9.2 RSA-Schlüssel in X.509-Zertifikaten**

In anderen, nicht-TI Public-Key-Infrastrukturen werden öffentliche Schlüssel bei einer Zertifikatsantragsstellung immer mittels ihrem korrespondierenden privaten Schlüssel signiert (vgl. Certificate Signing Request [RFC-2986], proof of possession). Dort kann der TSP sich nach einer erfolgreichen Signaturprüfung sicher sein, dass er aus Kodierungssicht den "richtigen" Schlüssel in den Händen hält, weil ansonsten die Signaturprüfung mit praktischer Sicherheit fehlschlägt. Missverständnisse aufgrund von "falscher" Byte-Order oder verschiedener Kodierung sind somit praktisch (Falsch-Positiv-Rate  $< 2^{-100}$ ) ausgeschlossen. In der PKI der TI werden mehr als 95 % aller Zertifikatserstellungen ohne eine Signatur mittels der jeweiligen privaten Schlüssel durchgeführt. Ein TSP der TI kann damit bei RSA-Schlüsseln – aus den Schlüsselwerten an sich – im Regelfall nicht sicher erkennen, ob eine Fehlkodierung (Missverständnis zwischen Zertifikatsantragssteller und TSP) aufgetreten ist. Es gibt effiziente Möglichkeiten solche Fehlkodierungen zu erkennen. Den Einsatz solcher Möglichkeiten möchte die gematik befördern und gibt mit A\_17092 und A\_17093 zwei Verfahren als KANN-Anforderungen an.

Die Untersuchungen aus [MK-2016] und [ROCA-2017] zeigen, dass es hilfreich ist sich mit den konkreten Werten der RSA-Schlüssel zu beschäftigen. Die folgenden Verfahren nutzen Struktureigenschaften von RSA-Schlüsseln, die nicht-RSA-Schlüssel im Normalfall nicht vorweisen.

#### keine kleinen Primteiler:

Der Modulus eines RSA-Schlüssels muss aus genau zwei (oder wenigen [RFC-8017]) "großen" Primfaktoren bestehen. Falls der vom TSP angetroffene Wert durch eine vom TSP vorgegebene Primzahl teilbar ist, so ist der RSA-Schlüssel ungeeignet. Falls dieser Primteiler deutlich kleiner als  $2^{1023}$  ist, so kann es sich nicht um einen korrekten RSA-Schlüssel handeln.

Wird ein Modulus unabsichtlich von einem Sender falsch kodiert, so ist der dadurch entstehende Wert statistisch über alle möglichen Fehlkodierungen betrachtet im Normalfall mit der Wahrscheinlichkeit von 1/2 durch zwei teilbar, mit der Wahrscheinlichkeit von 1/3 durch 3 teilbar, mit einer Wahrscheinlichkeit von 1/5 durch 5 teilbar usw. Falls man nun den Modulus durch die ersten Primzahlen kleiner als 100 (25 Primzahlen) versucht zu teilen (was effizient möglich ist) und eine Teilbarkeit ausschließen kann, so kann man eine unabsichtliche Fehlkodierung mit einer hohen Wahrscheinlichkeit erkennen.



In der gematik wurden mehr als eine Billion ( $10^{12}$ ) RSA-Schlüssel zufällig erzeugt und verschiedene Fehlkodierungen dieser Schlüssel auf Primteiler kleiner 100 untersucht. Es wurden dabei folgende Erkennungsraten festgestellt.

| <b>Art der Fehlkodierung</b>   | <b>Erkennungsrate in Prozent<br/>(auf zwei Nachkommastellen gerundet)</b> |
|--------------------------------|---|
| Byte-Order falsch (vertauscht) | 76,03 %   |
| Off-by-One (left shift)        | 100 %   |
| Off-by-One (right shift)       | 88,07 %   |
| Base64-kodiert anstatt Binär   | 87,60 %   |
| Base58-kodiert anstatt Binär   | 88,01 %   |
| Hex-kodiert anstatt Binär      | 87,91 %   |

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

#### öffentlicher Exponent ist prim:

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

Diese Überlegungen führen zu den Tests in A\_17092. Diese Tests haben eine Falsch-Positiv-Rate von 0 und benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

#### Entropie der Schlüsselkodierung:

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

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

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

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

Die gematik verwendet folgende C-Implementierung:

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

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

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

Ein Histogramm eines Beispiel-Testlaufs mit mehr als eine Billion ( $10^{12}$ ) RSA-Schlüsseln:

| 8 | $\geq H(X) > 7,455$         | 3396         |
|---|-----------------------------|--------------|
|   | $7,455 \geq H(X) > 7,2135$  | 234195871140 |
|   | $7,2135 \geq H(X) > 6,9715$ | 765693789112 |
|   | $6,9715 \geq H(X) > 6,7295$ | 112336352    |

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

Ein Histogramm eines Beispiel-Testlaufs mit mehr 10 Milliarden ( $1 \cdot 10^{10}$ ) Schlüsseln (diese wurden jeweils in unterschiedlichen Kodierungsvarianten kodiert und danach wurde die entropy()-Funktion auf die Kodierung angewendet):

| korrekt kodiert (s. o.) | Base64-kodiert  | Base58-kodiert  | als Hexadezimal-Zahl kodiert |
|-------------------------|-----------------|-----------------|------------------------------|
| 7.40 49808              | 5.90 9981660    | 5.80 11220      |                              |
| 7.30 97418682           | 5.80 6902282798 | 5.70 4102181822 | 3.90 10758804076             |
| 7.20 3351624284         | 5.70 3861576302 | 5.60 6615817484 | 3.80 40835572                |
| 7.10 6095663118         | 5.60 25792616   | 5.50 81606244   | 3.70 352                     |
| 7.00 1210930726         | 5.50 6624       | 5.40 23230      |                              |
| 6.90 43696816           |                 |                 |                              |
| 6.80 256390             |                 |                 |                              |
| 6.70 176                |                 |                 |                              |

Diese Überlegungen bezüglich der Entropie der RSA-Schlüsselkodierung führen zu dem Test in A\_17093. Dieser Test hat eine Falsch-Positiv-Rate von weniger als  $2^{-40}$  und benötigen weniger als 7 Mikrosekunden pro RSA-Schlüssel.

## 10 Anhang – Verzeichnisse

### 10.1 Abkürzungen

| Kürzel | Erläuterung  |
|--------|--|
| BS     | betreiberspezifischer Schlüssel                                  |
| C2C    | Card to Card   |
| C2S    | Card to Server   |
| CEK    | Content Encryption Key   |
| CA     | Certificate Authority  |
| CBC    | Cipher Block Chaining  |
| DNS    | Domain Name System   |
| DNSSEC | Domain Name System Security Extensions                           |
| DRNG   | Deterministic Random Number Generator                            |
| eGK    | elektronische Gesundheitskarte                                   |
| HKDF   | HMAC-based Extract-and-Expand Key Derivation Function            |
| HMAC   | Hash-based Message Authentication Code                           |
| IV     | Initialization Vector (Initialisierungsvektor) bspw. bei AES-GCM |
| ICV    | Integrity Check Value, Authentisierungswert (MAC) bei AES-GCM    |

| Kürzel      | Erläuterung   |
|-------------|---|
| KDF         | Key Derivation Function (Schlüsselableitungsfunktion)   |
| MAC         | Message Authentication Code   |
| OCSP        | Online Certificate Status Protocol  |
| OID         | Object Identifier   |
| OSI         | Open Systems Interconnection  |
| SAK         | Signaturanwendungskomponente  |
| SGD         | Schlüsselgenerierungsdienst   |
| SE          | Secure Element  |
| SM          | Service Monitoring  |
| TCB         | Trusted Computing Base  |
| TI          | Telematikinfrastuktur   |
| TLS         | Transport Layer Security  |
| TPM         | Trusted Plattform Module  |
| TSIG        | Transaction Signature   |
| URI         | Uniform Resource Identifier   |
| VAU         | vertrauenswürdige Ausführungsumgebung, vgl.<br>[gemSpec_DokumentenverwaltungAktensystem_ePAfuerAlle]      |
| WANDA Basic | Weitere Anwendungen für den Datenaustausch ohne Nutzung der TI oder<br>derer kryptografischen Identitäten |

## 10.2 Glossar

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

## 10.3 Abbildungsverzeichnis

|   |                |
|---|----------------|
| <del>Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht .....</del>                             | <del>27</del>  |
| <del>Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält .....</del>                  | <del>77</del>  |
| <del>Abbildung 3: Übersicht über das VAU-Protokoll .....</del>  | <del>89</del>  |
| <del>Abbildung 4: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU .....</del> | <del>110</del> |
| Abbildung 1: Verwendung von Algorithmen nach Zonen und OSI-Schicht .....  | 27             |
| Abbildung 2: ASN.1-Kodierung des Chiffrats was den Transportschlüssel enthält .....                             | 77             |
| Abbildung 3: Sicherungsschichten beim Datentransport zwischen E-Rezept-Client und E-Rezept-VAU .....            | 110            |
| Abbildung 4: Sicherungsschichten beim Datentransport zwischen ePA-Client und ePA-VAU-Instanz .....              | 121            |
| Abbildung 5: OSI-Schichten in einem Aktensystem .....   | 122            |
| Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS] .....  | 124            |
| Abbildung 7 : OAuth2/OIDC/PKCE-Authentisierung einer LEI am zentralen IDP der TI.                               | 139            |

## 10.4 Tabellenverzeichnis

|   |               |
|---|---------------|
| <del>Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten .....</del>  | <del>13</del> |
| <del>Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „RSA“ .....</del>          | <del>14</del> |
| <del>Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“ .....</del>       | <del>15</del> |
| <del>Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter elektronischer Signaturen für die Schlüsselgeneration „RSA“ .....</del> | <del>17</del> |
| <del>Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“ .....</del>             | <del>19</del> |
| Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate .....   | 21            |
| Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate .....  | 21            |
| Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs .....  | 28            |
| Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten elektronischen XML-Signaturen .....   | 29            |
| Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen .....  | 30            |

|   |     |
|---|-----|
| Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung .....  | 33  |
| Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC.....   | 44  |
| Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen<br>Schlüssels .....   | 45  |
| Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines<br>versichertenindividuellen Schlüssels.....  | 46  |
| Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären<br>Daten im Kontext von Dokumentensignaturen .....                            | 48  |
| Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-<br>Dokumentensignaturen .....  | 49  |
| Tabelle 17: Tab_KRYPT_ERP_Zertifikatsliste-Definition-Datenstruktur-Zertifikatsliste...   | 112 |
| Tabelle 18: Tab_KRYPT_ERP_Algorithmus_FD-Zertifikatsliste-erstellen.....  | 112 |
| Tabelle 19: Tab_KRYPT_ERP_FdV_Truststore-aktualisieren.....   | 113 |
| Tabelle 20: Tab_KRYPT_ERP_OCSP_Liste-Definition-Datenstruktur-OCSPList.....   | 114 |
| Tabelle 21: Tab_KRYPT_ERP_Kodierung des Chiffrats aus A_20161-*   | 116 |
| Tabelle 22: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht<br>kryptographisch gesicherten VAU-Kommunikation (E-Rezept) .....                   | 119 |
| Tabelle 1: Tab_KRYPT_001 Übersicht über Arten von X.509-Identitäten .....   | 13  |
| Tabelle 2: Tab_KRYPT_002 Algorithmen für X.509-Identitäten zur Erstellung nicht-<br>qualifizierter Signaturen für die Schlüsselgeneration „RSA“ .....         | 14  |
| Tabelle 3: Tab_KRYPT_002a Algorithmen für X.509-Identitäten zur Erstellung nicht-<br>qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“ .....      | 15  |
| Tabelle 4: Tab_KRYPT_003 Algorithmen für X.509-Identitäten zur Erstellung qualifizierter<br>elektronischer Signaturen für die Schlüsselgeneration „RSA“ ..... | 17  |
| Tabelle 5: Tab_KRYPT_003a Algorithmen für X.509-Identitäten zur Erstellung<br>qualifizierter Signaturen für die Schlüsselgeneration „ECDSA“ .....             | 19  |
| Tabelle 6: Tab_KRYPT_006 Algorithmen für CV-Zertifikate.....  | 21  |
| Tabelle 7: Tab_KRYPT_007 Algorithmen für CV-CA-Zertifikate.....   | 21  |
| Tabelle 8: Tab_KRYPT_008 Beispiele für solche Algorithmen-URIs .....  | 28  |
| Tabelle 9: Tab_KRYPT_009 Algorithmen für die Erzeugung von nicht-qualifizierten<br>elektronischen XML-Signaturen .....  | 29  |
| Tabelle 10: Tab_KRYPT_010 Algorithmen für qualifizierte XML-Signaturen .....  | 30  |
| Tabelle 11: Tab_KRYPT_012 Algorithmen für Card-to-Server-Authentifizierung .....  | 33  |
| Tabelle 12: Tab_KRYPT_017 Algorithmen für DNSSEC.....   | 44  |
| Tabelle 13: Tab_KRYPT_018 Ablauf zur Berechnung eines versichertenindividuellen<br>Schlüssels .....   | 45  |
| Tabelle 14: Tab_KRYPT_019 eingesetzte Algorithmen für die Ableitung eines<br>versichertenindividuellen Schlüssels.....  | 46  |
| Tabelle 15: Tab_KRYPT_020 Algorithmen für die Erzeugung und Prüfung von binären<br>Daten im Kontext von Dokumentensignaturen .....                            | 48  |



|  |     |
|--|-----|
| Tabelle 16: Tab_KRYPT_021 Algorithmen für die Erzeugung und Prüfung von PDF/A-Dokumentensignaturen .....                                 | 49  |
| Tabelle 17: Tab_KRYPT_ERP_Zertifikatsliste Definition Datenstruktur Zertifikatsliste ...   | 112 |
| Tabelle 18: Tab_KRYPT_ERP_Algorithmus_FD_Zertifikatsliste_erstellen.....   | 112 |
| Tabelle 19: Tab_KRYPT_ERP_FdV_Truststore_aktualisieren.....  | 113 |
| Tabelle 20: Tab_KRYPT_ERP_OCSP_Liste Definition Datenstruktur OCSPList .....   | 114 |
| Tabelle 21: Tab_KRYPT_ERP Kodierung des Chiffrats aus A_20161-* .....  | 116 |
| Tabelle 22: Tab_KRYPT_VAUERR Auftretende Fehler bei auf Anwendungsschicht kryptographisch gesicherten VAU-Kommunikation (E-Rezept) ..... | 119 |

## 10.5 Referenzierte Dokumente

### 10.5.1 Dokumente der gematik

Die nachfolgende Tabelle enthält die Bezeichnung der in dem vorliegenden Dokument referenzierten Dokumente der gematik zur Telematikinfrastruktur. Der mit der vorliegenden Version korrelierende Entwicklungsstand dieser Konzepte und Spezifikationen wird pro Release in einer Dokumentenlandkarte definiert; Version und Stand der referenzierten Dokumente sind daher in der nachfolgenden Tabelle nicht aufgeführt. Deren zu diesem Dokument jeweils gültige Versionsnummer entnehmen Sie der aktuellen, von der gematik veröffentlichten Dokumentenlandkarte, in der die vorliegende Version aufgeführt wird.

| [Quelle]                                  | Herausgeber: Titel  |
|---|---|
| [gemGlossar]                              | gematik: Glossar der Telematikinfrastruktur   |
| [gemSpec_COS]                             | gematik: Spezifikation des Card Operating System (COS)                              |
| <del>[gemSpec_Dokumentenverwaltung]</del> | <del>gematik: Spezifikation ePA-Dokumentenverwaltung</del>                          |
| [gemSpec_DS_Anbieter]                     | Spezifikation Datenschutz- und Sicherheitsanforderungen der TI an Anbieter          |
| [gemSpec_eGK_ObjSys]                      | gematik: Die Spezifikation der elektronischen Gesundheitskarte (eGK) – Objektsystem |
| [gemSpec_KT]                              | gematik: Spezifikation eHealth-Kartenterminal                                       |
| [gemSpec_MobKT]                           | gematik: Spezifikation Mobiles Kartenterminal                                       |

|                               |   |
|-------------------------------|---|
| [gemSpec_SGD_ePA]             | gematik: Spezifikation Schlüsselkommentierungsdienst ePA        |
| [gemSpec_SST_FD_VSDM]         | gematik: Schnittstellenspezifikation Fachdienste (UFS/VSDM/CMS) |
| [gemSpec_Zugangsgateway_Vers] | gematik: Spezifikation Zugangsgateway des Versicherten ePA      |

## 10.5.2 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<br><a href="http://web.cs.ucdavis.edu/~rogaway/papers/dhies.pdf">http://web.cs.ucdavis.edu/~rogaway/papers/dhies.pdf</a>   |
| [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,<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifierung/Interpretation/AIS20_Functionality_Classes_Evaluation_Methodology_DRNG.pdf?__blob=publicationFile">https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifierung/Interpretation/AIS20_Functionality_Classes_Evaluation_Methodology_DRNG.pdf?__blob=publicationFile</a> |
| [AIS-20]      | AIS 20: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren, Version 3, 15.05.2013,<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifierung/Interpretation/AIS_20_pdf.pdf?__blob=publicationFile">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifierung/Interpretation/AIS_20_pdf.pdf?__blob=publicationFile</a>  |
| [AIS-31]      | AIS 31: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren, Version 3, 15.05.2013,<br><a href="http://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifierung/Interpretationen/AIS_31_pdf.pdf?__blob=publicationFile">http://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifierung/Interpretationen/AIS_31_pdf.pdf?__blob=publicationFile</a>   |
| [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: <a href="https://www.bundesanzeiger.de">https://www.bundesanzeiger.de</a> mit dem Suchbegriff „BAnz AT 30.12.2016 B5“)  |

| [Quelle]             | Herausgeber (Erscheinungsdatum): Titel  |
|----------------------|---|
| [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.<br><a href="http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf">http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf</a>                             |
| [ANSI-X9.62]         | ANSI X9.62:2005<br>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<br>Public Key Cryptography for the Financial Services Industry<br>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<br>19.10.2005<br><a href="http://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf">http://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf</a>  |
| [Breaking-TLS]       | Lucky Thirteen: Breaking the TLS and DTLS Record Protocols<br>Nadhem J. AlFardan and Kenneth G. Paterson<br>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<br><a href="http://www.nds.rub.de/media/nds/veroeffentlichungen/2011/10/22/HowToBreakXMLenc.pdf">http://www.nds.rub.de/media/nds/veroeffentlichungen/2011/10/22/HowToBreakXMLenc.pdf</a>   |
| [BSI-PQC-2020]       | Migration zu Post-Quanten-Kryptografie, Handlungsempfehlungen des BSI,<br>Stand: August 2020, <a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?__blob=publicationFile&amp;v=2">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf?__blob=publicationFile&amp;v=2</a>                 |
| [BSI-TR-02102-1]     | BSI TR-02102-1 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“<br>Version 2023-01<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.html">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.html</a> |

| [Quelle]         | Herausgeber (Erscheinungsdatum): Titel  |
|------------------|---|
| [BSI-TR-02102-2] | BSI TR-02102-2 Technische Richtlinie „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Teil 2 – Verwendung von Transport Layer Security (TLS), Version 2023-01<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.html">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.html</a>  |
| [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 2023-01<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-3.html">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-3.html</a>  |
| [BSI-TR-03111]   | Technical Guideline BSI TR-03111<br>Elliptic Curve Cryptography, Version 2.10, Date: 2018-06-01<br><a href="https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03111/TR-03111_node.html">https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03111/TR-03111_node.html</a>  |
| [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 <a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116.html">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116.html</a><br><br>Die Technische Richtlinie befindet sich aktuell in der Überarbeitung. Sobald die Überarbeitung der TR-03116-1 erfolgt ist, wird die Referenz auf die neue Version der TR-03116-1 hier aktualisiert. |
| [CM-2014]        | 20 Years of SSL/TLS Research, An Analysis of the Internet’s Security Foundation, Christopher Meyer, 9. February 2014<br><a href="http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf">http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf</a>   |
| [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 <a href="https://www.ecma-international.org/publications/standards/Ecma-262.htm">https://www.ecma-international.org/publications/standards/Ecma-262.htm</a>   |
| [EN-14890-1]     | DIN EN 14890-1:2008<br>Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic services   |

| [Quelle]                 | Herausgeber (Erscheinungsdatum): Titel   |
|--------------------------|--|
| [ENISA-PQC-2021]         | Post-Quantum Cryptography: Current state and quantum mitigation, European Union Agency for Cybersecurity (ENISA), May 2021, <a href="https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation">https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation</a>  |
| [ETSI-CAAdES]            | ETSI TS 101 733 V1.7.4 (2008-07), Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAAdES)   |
| [ETSI_TS_102_231_v3.1.2] | ETSI (Dezember 2009): ETSI Technical Specification TS 102 231 ('Provision of harmonized Trust Service Provider (TSP) status information') – Version 3.1.2  |
| [ETSI-XAdES]             | ETSI TS 101 903 V1.4.2 (2010-12), Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)  |
| [FIPS-180-4]             | Federal Information, Processing Standards Publication 180-4, Secure Hash Standard (SHS), March 2012<br><a href="http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf">http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf</a>   |
| [FIPS-186-2+CN1]         | FIPS 186-2 - National Institute of Standards and Technology, <u>Digital Signature Standard (DSS)</u> , Federal Information Processing Standards Publication 186-2, January 27, 2000 – Appendix 3.1 unter der Beachtung des Change Notice 1, vom 5. Oktober 2001<br><a href="http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2-change1.pdf">http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2-change1.pdf</a> |
| [FIPS-186-5]             | FIPS 186-5 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (Supersedes FIPS 186-4) Digital Signature Standard (DSS), 03.02.2023<br><a href="https://csrc.nist.gov/pubs/fips/186-5/final">https://csrc.nist.gov/pubs/fips/186-5/final</a><br><a href="https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf">https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf</a>   |
| [FIPS-197]               | Federal Information Processing Standards Publication 197, (FIPS-197), November 26, 2001, Announcing the ADVANCED ENCRYPTION STANDARD (AES)<br><a href="http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf">http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf</a>  |
| [FIPS-202]               | NIST, FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015,<br><a href="http://nvlpubs.nist.gov/nistpubs/fips/NIST.FIPS.202.pdf">http://nvlpubs.nist.gov/nistpubs/fips/NIST.FIPS.202.pdf</a>   |
| [IETF-KEM-TLS]           | KEM-based Authentication for TLS 1.3<br>T. Wiggers, S. Celi, P. Schwabe, D. Stebila, N. Sullivan<br><a href="https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/">https://datatracker.ietf.org/doc/draft-celi-wiggers-tls-authkem/</a>  |

| [Quelle]         | Herausgeber (Erscheinungsdatum): Titel  |
|------------------|---|
| [IETF-Kyber]     | Kyber Post-Quantum KEM, P. Schwabe, B. E. Westerbaan<br>Cloudflare, 2023, <a href="https://www.ietf.org/archive/id/draft-cfrg-schwabe-kyber-03.html">https://www.ietf.org/archive/id/draft-cfrg-schwabe-kyber-03.html</a>   |
| [IR-2014]        | Bulletproof SSL and TLS: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Ivan Ristić, 2014<br><a href="https://www.feistyduck.com/books/bulletproof-ssl-and-tls/">https://www.feistyduck.com/books/bulletproof-ssl-and-tls/</a>   |
| [ISO-11770]      | ISO/IEC 11770: 1996, Information technology – Security techniques –<br>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.<br>siehe auch <a href="http://www.petitcolas.net/fabien/kerckhoffs/">http://www.petitcolas.net/fabien/kerckhoffs/</a>  |
| [KEM-TLS]        | Post-quantum TLS without handshake signatures<br>Peter Schwabe, Douglas Stebila, and Thom Wiggers<br><a href="https://eprint.iacr.org/2020/534">https://eprint.iacr.org/2020/534</a>  |
| [KS-2011]        | W. Killmann, W. Schindler, „A proposal for: Functionality classes for random number generators“, Version 2.0, September 2011<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifizierung/Interpretation/AIS31_Functionality_classes_for_random_number_generators.pdf?__blpb=publicationFile">https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifizierung/Interpretation/AIS31_Functionality_classes_for_random_number_generators.pdf?__blpb=publicationFile</a> |
| [MK-2016]        | The Million-Key Question – Investigating the Origins of RSA Public Keys,<br>Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, Vashek Matyas,<br>The 25th USENIX Security Symposium (UsenixSec'2016)<br><a href="https://crocs.fi.muni.cz/public/papers/usenix2016">https://crocs.fi.muni.cz/public/papers/usenix2016</a>   |
| [NIST-PQC]       | Post-Quantum Cryptography, National Institute of Standards and Technology (NIST),<br><a href="https://csrc.nist.gov/projects/post-quantum-cryptography">https://csrc.nist.gov/projects/post-quantum-cryptography</a>  |
| [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<br><a href="https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final">https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final</a>  |

| [Quelle]               | Herausgeber (Erscheinungsdatum): Titel   |
|------------------------|--|
| [NIST-SP-800-38A]      | NIST Special Publication 800-38A,<br>Recommendation for Block, Cipher Modes of Operation, Methods and<br>Techniques, Morris Dworkin, December 2001 Edition,<br><a href="http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf">http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf</a>       |
| [NIST-SP-800-38B]      | NIST Special Publication 800-38B,<br>Recommendation for Block Cipher Modes of Operation: The CMAC<br>Mode for Authentication, Morris Dworkin, May 2005 Edition,<br><a href="http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf">http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf</a> |
| [NIST-SP-800-38D]      | NIST Special Publication 800-38D,<br>Recommendation for Block Cipher Modes of Operation:<br>Galois/Counter<br>Mode (GCM) and GMAC, Morris Dworkin, November, 2007  |
| [NIST-SP-800-56-A]     | NIST Special Publication 800-56A<br>Recommendation for Pair-Wise Key Establishment Schemes Using<br>Discrete Logarithm Cryptography,<br>April 2018<br><a href="https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final">https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final</a>                |
| [NIST-SP-800-56-B]     | NIST Special Publication 800-56B<br>Recommendation for Pair-Wise Key Establishment Schemes Using<br>Integer Factorization Cryptography, August 2009  |
| [NIST-SP-800-56C]      | NIST Special Publication 800-56C<br>Recommendation for Key Derivation through Extraction-then-<br>Expansion,<br>November 2011  |
| [NIST-SP-800-108]      | NIST Special Publication 800-108<br>Recommendation for Key Derivation Using Pseudorandom Functions,<br>October 2009  |
| [NK-PP]                | Common Criteria Schutzprofil (Protection Profile)<br>Schutzprofil 1: Anforderungen an den Netzkonnektor, BSI-CC-PP-<br>0097  |
| [Oorschot-Wiener-1996] | On Diffie-Hellman Key Agreement with Short Exponents, Paul C. van<br>Oorschot, Michael J Weiner, Eurocrypt' 96   |
| [Padding-Oracle-2005]  | Padding Oracle Attacks on CBC-mode Encryption with Secret and<br>Random IVs<br>Arnold K. L. Yau, Kenneth G. Paterson and Chris J. Mitchell, FSE 2005<br><a href="http://www.isg.rhul.ac.uk/~kp/secretIV.pdf">http://www.isg.rhul.ac.uk/~kp/secretIV.pdf</a>  |



| [Quelle]            | Herausgeber (Erscheinungsdatum): Titel  |
|---------------------|---|
| [PADES-3]           | ETSI TS 102 778-3 V1.2.1,<br>PDF Advanced Electronic Signature Profiles; Part 3: PAdES Enhanced<br>–<br>PAdES-BES and PAdES-EPES Profiles<br>Technical Specification, 2010  |
| [PDF/A-2]           | ISO 19005-2:2011 – Document management – Electronic document<br>file<br>format for long-term preservation – Part 2: Use of ISO 32000-1<br>(PDF/A-2)   |
| [PKCS#1]            | vgl. [RFC-8017]   |
| [PP-0082]           | Common Criteria Protection Profile, Card Operating System<br>Generation<br>2 (PP COS G2), BSI-CC-PP-0082-V2, Version 1.9, 18th November<br>2014   |
| [PQC-Hybrid-Chrome] | Protecting Chrome Traffic with Hybrid Kyber KEM, Thursday, August<br>10, 2023<br><a href="https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html">https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html</a>              |
| [PQC-Hybrid-Signal] | The PQXDH Key Agreement Protocol, Revision 2, 2023-05-24<br><a href="https://signal.org/docs/specifications/pqxdh/">https://signal.org/docs/specifications/pqxdh/</a><br><br>(vgl. auch <a href="https://signal.org/blog/pqxdh/">https://signal.org/blog/pqxdh/</a> ) |
| [RFC-2119]          | RFC 2119 (März 1997): Key words for use in RFCs to Indicate<br>Requirement<br>Levels, S. Bradner,<br><a href="http://tools.ietf.org/html/rfc2119">http://tools.ietf.org/html/rfc2119</a>  |
| [RFC-2590]          | RFC 2590 (June 1999): X.509 Internet Public Key Infrastructure<br>Online<br>Certificate Status Protocol - OCSP<br><a href="https://tools.ietf.org/html/rfc2560">https://tools.ietf.org/html/rfc2560</a><br>(Obsoleted by [RFC-6960])                                  |
| [RFC-2986]          | RFC 2986 (November 2000): PKCS #10: Certification Request Syntax<br>Specification, Version 1.7<br><a href="https://tools.ietf.org/html/rfc2986">https://tools.ietf.org/html/rfc2986</a>   |

| [Quelle]   | Herausgeber (Erscheinungsdatum): Titel   |
|------------|--|
| [RFC-3279] | RFC 3279 (April 2002): Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile<br><a href="https://tools.ietf.org/html/rfc3279">https://tools.ietf.org/html/rfc3279</a>      |
| [RFC-3526] | RFC 3526 (Mai 2003: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)<br><a href="http://tools.ietf.org/html/rfc3526">http://tools.ietf.org/html/rfc3526</a>   |
| [RFC-4051] | Additional XML Security Uniform Resource Identifiers (URIs), April 2005<br><a href="https://tools.ietf.org/html/rfc4051">https://tools.ietf.org/html/rfc4051</a>   |
| [RFC-4635] | RFC 4635 (August 2006): HMAC SHA TSIG Algorithm Identifiers<br><a href="http://tools.ietf.org/html/rfc4635">http://tools.ietf.org/html/rfc4635</a>   |
| [RFC-5077] | Transport Layer Security (TLS) Session Resumption without Server-Side State, January 2008, <a href="https://tools.ietf.org/html/rfc5077">https://tools.ietf.org/html/rfc5077</a>   |
| [RFC-5084] | RFC 5084: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS), November 2007<br><a href="https://tools.ietf.org/html/rfc5084">https://tools.ietf.org/html/rfc5084</a>   |
| [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<br><a href="https://tools.ietf.org/html/rfc5091">https://tools.ietf.org/html/rfc5091</a> |
| [RFC-5246] | The Transport Layer Security (TLS) Protocol Version 1.2, August 2008,<br><a href="https://tools.ietf.org/html/rfc5246">https://tools.ietf.org/html/rfc5246</a>   |
| [RFC-5280] | RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, Mai 2008<br><a href="https://tools.ietf.org/html/rfc5280">https://tools.ietf.org/html/rfc5280</a>  |
| [RFC-5480] | RFC 5480 (March 2009): Elliptic Curve Cryptography Subject Public Key Information, <a href="https://tools.ietf.org/html/rfc5480">https://tools.ietf.org/html/rfc5480</a>   |
| [RFC-5639] | RFC 5639 (March 2010): Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation,<br><a href="http://www.ietf.org/rfc/rfc5639.txt">http://www.ietf.org/rfc/rfc5639.txt</a>  |

| [Quelle]   | Herausgeber (Erscheinungsdatum): Titel  |
|------------|---|
| [RFC-5652] | RFC 5652 (September 2009): Cryptographic Message Syntax (CMS), R. Housley, <a href="http://tools.ietf.org/html/rfc5652">http://tools.ietf.org/html/rfc5652</a>  |
| [RFC-5702] | RFC 5702 (October 2009): Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC, <a href="http://tools.ietf.org/html/rfc5702">http://tools.ietf.org/html/rfc5702</a>                          |
| [RFC-5746] | RFC 5746: Transport Layer Security (TLS) Renegotiation Indication Extension, February 2010, <a href="https://tools.ietf.org/html/rfc5746">https://tools.ietf.org/html/rfc5746</a>                                       |
| [RFC-5753] | RFC 5753: Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), January 2010, <a href="https://tools.ietf.org/html/rfc5753">https://tools.ietf.org/html/rfc5753</a>                |
| [RFC-5869] | HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010, <a href="https://tools.ietf.org/html/rfc5869">https://tools.ietf.org/html/rfc5869</a>   |
| [RFC-5903] | Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2, June 2010, <a href="https://tools.ietf.org/html/rfc5903">https://tools.ietf.org/html/rfc5903</a>   |
| [RFC-6090] | RFC 6090: Fundamental Elliptic Curve Cryptography Algorithms, February 2011, <a href="https://tools.ietf.org/html/rfc6090">https://tools.ietf.org/html/rfc6090</a>  |
| [RFC-6954] | Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2), July 2013, <a href="https://tools.ietf.org/html/rfc6954">https://tools.ietf.org/html/rfc6954</a> |
| [RFC-6960] | RFC 6960 (June 2013): X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, <a href="https://tools.ietf.org/html/rfc6960">https://tools.ietf.org/html/rfc6960</a>                         |
| [RFC-7027] | RFC 7027: (October 2013) Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS), <a href="https://tools.ietf.org/html/rfc7027">https://tools.ietf.org/html/rfc7027</a>                   |
| [RFC-7296] | RFC 7296 (October 2014): Internet Key Exchange Protocol Version 2 (IKEv2), <a href="https://tools.ietf.org/html/rfc7296">https://tools.ietf.org/html/rfc7296</a>  |
| [RFC-7427] | RFC 7427 (January 2015): Signature Authentication in the Internet Key Exchange Version 2 (IKEv2), <a href="https://tools.ietf.org/html/rfc7427">https://tools.ietf.org/html/rfc7427</a>                                 |

| [Quelle]                | Herausgeber (Erscheinungsdatum): Titel  |
|-------------------------|---|
| [RFC-8017],<br>[PKCS#1] | "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2", November 2016<br><a href="https://tools.ietf.org/html/rfc8017">https://tools.ietf.org/html/rfc8017</a>  |
| [RFC-931]               | RFC 6931: Additional XML Security Uniform Resource Identifiers (URIs),<br>Donald Eastlake, April 2013, <a href="https://tools.ietf.org/html/rfc6931">https://tools.ietf.org/html/rfc6931</a>  |
| [RFC-9155]              | RFC 9155: Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2, December 2021, <a href="https://datatracker.ietf.org/doc/rfc9155/">https://datatracker.ietf.org/doc/rfc9155/</a>  |
| [ROCA-2017]             | The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli,<br>Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, Vashek Matyas<br>24th ACM Conference on Computer and Communications Security (CCS'2017)<br><a href="https://crocs.fi.muni.cz/public/papers/rsa_ccs17">https://crocs.fi.muni.cz/public/papers/rsa_ccs17</a> |
| [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<br><a href="https://www.secg.org/sec1-v2.pdf">https://www.secg.org/sec1-v2.pdf</a>  |
| [SDH-2016]              | Measuring the Security Harm of TLS Crypto Shortcuts, Drew Springall, Zakir Durumeic, J. Alex Halderman, November 2016,<br><a href="https://jhalderm.com/pub/papers/forward-secrecy-imc16.pdf">https://jhalderm.com/pub/papers/forward-secrecy-imc16.pdf</a>   |
| [SOG-IS-2020]           | SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms, Version 1.2, January 2020<br><a href="https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf">https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf</a>   |
| [TLS-Attacks]           | Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses, Christopher Meyer und Jörg Schwenk, 31. Januar 2013, <a href="http://eprint.iacr.org/2013/049">http://eprint.iacr.org/2013/049</a>  |
| [TLS-CC-2021]           | Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS), September 2021, <a href="https://datatracker.ietf.org/doc/draft-campagna-tls-bike-sike-hybrid/">https://datatracker.ietf.org/doc/draft-campagna-tls-bike-sike-hybrid/</a>  |
| [XMLCan_V1.0]           | Exclusive XML Canonicalization, Version 1.0, W3C Recommendation 18 July 2002, <a href="http://www.w3.org/TR/xml-exc-c14n/">http://www.w3.org/TR/xml-exc-c14n/</a>   |

| [Quelle]          | Herausgeber (Erscheinungsdatum): Titel  |
|-------------------|---|
| [XMLDSig]         | XML Signature Syntax and Processing Version 1.1, W3C Recommendation 11 April 2013<br><a href="https://www.w3.org/TR/xmlsig-core1/">https://www.w3.org/TR/xmlsig-core1/</a>  |
| [XMLDSig-Draft]   | XML Signature Syntax and Processing Version 2.0, W3C Editor's Draft 04 February 2014, <a href="http://www.w3.org/2008/xmlsec/Drafts/xmlsig-core-20/">http://www.w3.org/2008/xmlsec/Drafts/xmlsig-core-20/</a>   |
| [XMLDSig-RSA-PSS] | RSA-PSS in XMLDSig, 25/26 September 2007, Konrad Lanz, Dieter Bratko, Peter Lipp,<br><a href="http://www.w3.org/2007/xmlsec/ws/papers/08-lanz-iaik/">http://www.w3.org/2007/xmlsec/ws/papers/08-lanz-iaik/</a>  |
| [XMLEnc]          | XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002, <a href="http://www.w3.org/TR/xmlenc-core/">http://www.w3.org/TR/xmlenc-core/</a>  |
| [XMLEnc-CM]       | Technical Analysis of Countermeasures against Attack on XML Encryption - or - Just Another Motivation for Authenticated Encryption.<br>Juraj Somorovsky, Jörg Schwenk. 2011<br><a href="http://www.w3.org/2008/xmlsec/papers/xmlEncCountermeasuresW3C.pdf">http://www.w3.org/2008/xmlsec/papers/xmlEncCountermeasuresW3C.pdf</a>                                      |
| [XMLEnc-1.1]      | XML Encryption Syntax and Processing, W3C Recommendation 11 April 2013, <a href="http://www.w3.org/TR/xmlenc-core1/">http://www.w3.org/TR/xmlenc-core1/</a>   |
| [XSpRES]          | XML Spoofing Resistant Electronic Signature (XSpRES) -- Sichere Implementierung für XML-Signaturen<br>Bundesamt für Sicherheit in der Informationstechnik 2012<br><a href="https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SOA/XSpRESS.pfd?__blob=publicationFile">https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SOA/XSpRESS.pfd?__blob=publicationFile</a> |
| [XSW-Attack]      | On Breaking SAML: Be Whoever You Want to Be<br>Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, Meiko Jensen, Usenix 2012<br><a href="http://www.nds.rub.de/media/nds/veroeffentlichungen/2012/08/03/BreakingSAML.pdf">http://www.nds.rub.de/media/nds/veroeffentlichungen/2012/08/03/BreakingSAML.pdf</a>  |
| [Vaudenay-2002]   | Security Flaws Induced by CBC Padding: Applications to SSL, IPsec, WTLS ... , Serge Vaudenay, Eurocrypt 2002, LNCS 2332/2002, 535-545<br><a href="https://www.iacr.org/cryptodb/data/paper.php?pubkey=2850">https://www.iacr.org/cryptodb/data/paper.php?pubkey=2850</a>  |