

Telematikinfrastuktur 2.0

Spezifikation Zero Trust Access (ZETA)

Version: [1.3-22.0.0_CC](#)
Revision: [165830056](#)
Stand: [26.0609.07.2026](#)
Status: [zur Abstimmung](#)
freigegeben
Klassifizierung: öffentlich [Entwurf](#)
Referenzierung: gemSpec_ZETA

Dokumentinformationen

ÄnderGender-Hinweis

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

Änderungen zur Vorversion

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

Dokumentenhistorie

Version	Stand	Kap./ Seite	Grund der Änderung, besondere Hinweise	Bearbeitung
1.0.0	28.02.202 5		initiale Erstellung	gematik
1.1.0	06.08.202 5		Einarbeitung ZETA_25_2	gematik
1.2.0	05.12.202 5		Einarbeitung ZETA_25_3	gematik
1.3.0	17.04.202 6		Einarbeitung ZETA_26_1	gematik
<u>1.3.12.0</u> <u>.0_CC</u>	<u>11.0509.0</u> <u>7.2026</u>		<u>Redaktionelle Korrektur in Kapitel-3.3</u> <u>Einarbeitung ZETA_26_2</u> <u>Einarbeitung ZETA Stufe 2 u.a. mit</u> <u>detaillierten Abläufen zu mobilen</u> <u>Clients, Ergänzung Notification</u> <u>Management und Client Management,</u> <u>Konkretisierung von HW Attestierung bei</u> <u>stationären Clients</u>	gematik
1.3.2	26.06.202 6		fehlende Tabellen in html-Darstellung bei A_26489-01, A_26486-01 ergänzt	gematik

Inhaltsverzeichnis

1 Einordnung des Dokuments.....	7
1.1 Zielsetzung.....	7
1.2 Zielgruppe.....	7
1.3 Abgrenzungen.....	7
1.4 Methodik.....	8
1.4.1 Anforderungen.....	8
2 Features und Epics.....	9
2.1 Client-Registrierung.....	9
2.1.1 Wiedererkennung bekannter Clients.....	9
2.1.2 Client-Attestation.....	9
2.1.3 Device Security Rating.....	10
2.2 Policy Enforcement.....	10
2.2.1 Zugriffsschutz.....	10
2.2.2 HTTP Proxy.....	10
2.3 Decide from Policies.....	10
2.3.1 Maschinenlesbare Zugriffsregeln.....	10
2.3.2 Ein reproduzierbares Ja/Nein.....	10
2.3.3 Policies nach Betroffenheit.....	11
2.4 Policy-Information und -Administration.....	11
2.4.1 Policy-Verwaltung.....	11
2.4.2 Monitoring.....	11
2.5 Authorization.....	11
2.5.1 Autorisierung auf Basis von Policy-Entscheidungen.....	11
2.5.2 Client-Authentication.....	12
3 Einordnung in die TI 2.0.....	13
3.1 ZETA als Umsetzung der Zero Trust Architektur.....	14
3.2 Kurzbeschreibung der Komponenten.....	15
3.3 Kurzbeschreibung der Schnittstellen.....	16
4 Technisches Konzept.....	18
4.1 ZETA Guard.....	18
4.2 Policy Enforcement Point (PEP).....	18
4.3 Policy Decision Point (PDP).....	19
4.4 ZETA Client.....	20
4.5 Policy-Information und -Administration.....	20
4.5.1 Policy Information Point (PIP).....	20
4.5.2 Policy Administration Point (PAP).....	20
4.5.3 PIP und PAP Ausprägung in ZETA.....	21

4.6 Client-Registrierung.....	21
4.7 Monitoring.....	21
4.7.1 Security Information and Event Management (SIEM).....	21
4.7.2 Shared Signals.....	22
4.7.3 Telemetrie, Monitoring und Logging.....	22
4.8 Zusammenspiel mit Identity Provider.....	22
4.9 TI 2.0-Dienst-Backend.....	23
5 Spezifikation.....	24
5.1 Übergreifende Anforderungen für Datenschutz und Sicherheit.....	24
5.2 Schlüssel Management und Verwaltung in ZETA.....	27
5.2.1 Nomenklatur für Schlüssel-IDs.....	27
5.2.2 Übersicht der ZETA Guard Schlüssel (Anbieter-Seite).....	28
5.2.3 ZETA Client Schlüssel (Nutzer-Seite).....	31
5.2.4 gematik verwaltete Schlüssel (TI).....	32
5.3 Sicherheits- und Datenschutzanforderungen an Logging und Monitoring.....	33
5.4 Sicherheits- und Datenschutz-Anforderungen an das Security Monitoring.....	34
5.5 Sicherheits- und Datenschutz-Anforderungen an die Protokollierung von Administrationsaktivitäten.....	42
5.6 Sicherheits- und Datenschutz-Anforderungen an die Verarbeitung von Daten mit dem Schutzbedarf "sehr hoch".....	43
5.7 Sicherheits- und Datenschutz Anforderungen an dem ZETA Client in FdVs.....	45
5.8 Clientsystem und ZETA Client.....	45
5.8.1 Hersteller.....	46
5.8.2 Verbindungsaufbau.....	46
5.8.3 Client-Registrierung.....	48
5.8.4 Nutzerauthentifizierung.....	50
5.8.5 Session Management.....	50
5.8.6 Liste der HTTP-Statuscodes.....	51
5.8.7 ZETA Attestation Service.....	54
5.9 Client-Registrierungsdaten.....	55
5.9.1 Client Assertion JWT.....	55
5.9.2 Client Statement.....	56
5.9.3 Posture Informationen.....	56
5.9.3.1 TPM Posture.....	56
5.9.3.2 Software Posture.....	57
5.9.3.3 Apple Posture.....	57
5.9.3.4 Android Posture.....	58
5.10 ZETA Guard.....	59
5.10.1 Klassifizierung der ZETA Guard Komponenten.....	62
5.10.1.1 Unveränderliche Kernkomponenten.....	63
5.10.1.2 Austauschbare Kernkomponenten.....	63
5.10.1.3 Hilfskomponenten und Funktionen.....	64
5.10.2 Kommunikation mit Diensten der gematik.....	64
5.10.3 Deployment Szenarien.....	65
5.10.3.1 Geo-Redundanz.....	65

5.10.4 Laufzeitüberwachung.....	65
5.10.4.1 Aktuell eingesetzte Verfahren.....	65
5.10.4.1.1 Pod Security Standards (PSS).....	65
5.10.4.1.2 Admission Controller.....	66
5.10.4.1.3 Network Policies.....	66
5.10.4.1.4 Service Mesh.....	66
5.10.4.1.5 Ingress und Egress / Gateway.....	67
5.10.4.2 Geplante Verfahren.....	67
5.10.4.2.1 Pod Überwachung gemäß Cilium Tetragon.....	67
5.10.4.2.2 RBAC-Härtung (Role-Based Access Control).....	68
5.11 Policy Enforcement Points.....	68
5.11.1 PEP HTTP Proxy.....	68
5.11.2 Sicherheits- und Datenschutz Anforderungen an den PEP.....	72
5.12 Policy Decision Point.....	73
5.12.1 Policy Engine.....	73
5.12.2 PDP Authorization Server.....	73
5.12.2.1 PDP Relying Party.....	78
5.12.2.2 Ablauf für den Zugriff auf einen Resource Server.....	78
5.12.2.3 Service Discovery.....	79
5.12.2.4 Client-Registrierung für stationäre Clients.....	81
5.12.2.5 Authentifizierung und Autorisierung für stationäre Clients.....	83
5.12.2.5.1 Pfad A: Token Austausch mit Attestierung.....	84
5.12.2.5.2 Pfad B: Token Erneuerung via Refresh Token.....	88
5.12.2.5.3 Gemeinsame nachfolgende Schritte.....	88
5.12.2.6 Ablauf der Authentifizierung bei Dienst-zu-Dienst Kommunikation.....	88
5.12.3 PDP Datenbank.....	90
5.12.4 Sicherheits- und Datenschutzanforderungen an den PDP.....	91
5.13 Policies und Daten.....	92
5.14 Telemetriedaten-Service.....	93
5.15 HSM Proxy.....	95
5.15.1 Sicherheits- und Datenschutzanforderungen an den HSM Proxy.....	96
5.16 Betrieb.....	97
5.16.1 Anforderungen an Hersteller einer ZETA Komponente.....	97
5.16.2 Anforderungen an Hersteller eines TI 2.0 Dienstes.....	97
5.16.3 Anforderungen an Anbieter eines TI 2.0 Dienstes.....	97
5.16.4 Anforderungen für nahtlose Aktualisierungen.....	98
5.16.5 Überwachung des Betriebsstatus.....	99
5.16.6 Leistungs Anforderungen.....	100
5.16.7 Betriebliche Schnittstellendefinition.....	101
5.16.8 Prozesse zur Inbetriebnahme eines ZETA Guard.....	103
5.17 Anforderungen an Dienste der TI.....	104
5.18 Sicherheitsleistungen des ZETA Guard für Resource Server.....	104
5.19 Weitere Leistungen des ZETA Guard für Resource Server.....	105
6 Beispiele und Referenzimplementierungen.....	106
7 Anhang A – Verzeichnisse.....	107

7.1	Abkürzungen	107
7.2	Glossar	108
7.3	Abbildungsverzeichnis	110
7.4	Tabellenverzeichnis	111
7.5	Referenzierte Dokumente	112
7.5.1	Dokumente der gematik	112
7.5.2	Weitere Referenzen	114
1	Einordnung des Dokuments	13
1.1	Zielsetzung	13
1.2	Zielgruppe	13
1.3	Abgrenzungen	13
1.4	Methodik	14
1.4.1	Anforderungen	14
2	Features und Epics	15
2.1	Client-Registrierung	15
2.1.1	Wiedererkennung bekannter Clients	15
2.1.2	Client-Attestation	15
2.1.3	Device Security Rating	16
2.2	Policy Enforcement	16
2.2.1	Zugriffsschutz	16
2.2.2	HTTP Proxy	16
2.3	Decide from Policies	16
2.3.1	Maschinenlesbare Zugriffsregeln	16
2.3.2	Ein reproduzierbares Ja/Nein	16
2.3.3	Policies nach Betroffenheit	17
2.4	Policy-Information und -Administration	17
2.4.1	Policy-Verwaltung	17
2.4.2	Monitoring	17
2.5	Authorization	17
2.5.1	Autorisierung auf Basis von Policy-Entscheidungen	17
2.5.2	Client Authentication	18
3	Einordnung in die TI 2.0	19
3.1	ZETA als Umsetzung der Zero Trust Architektur	21
3.2	Kurzbeschreibung der Komponenten	22
3.3	Kurzbeschreibung der Schnittstellen	23
4	Technisches Konzept	26
4.1	ZETA Guard	26
4.2	Policy Enforcement Point (PEP)	26
4.3	Policy Decision Point (PDP)	27

- 4.4 ZETA Client..... 28**
- 4.5 Policy-Information und -Administration..... 28**
 - 4.5.1 Policy Information Point (PIP)..... 28
 - 4.5.2 Policy Administration Point (PAP)..... 28
 - 4.5.3 PIP und PAP Ausprägung in ZETA..... 29
- 4.6 Client-Registrierung..... 29**
- 4.7 Monitoring..... 29**
 - 4.7.1 Security Information and Event Management (SIEM)..... 29
 - 4.7.2 Shared Signals..... 30
 - 4.7.3 Telemetrie, Monitoring und Logging..... 30
- 4.8 Zusammenspiel mit Identity Provider..... 30**
- 4.9 TI 2.0-Dienst-Backend..... 31**
- 5 Spezifikation..... 32**
 - 5.1 Anforderungen an die Sicherheit und den Datenschutz..... 32**
 - 5.2 Übergreifende Anforderungen für Datenschutz und Sicherheit..... 32**
 - berücksichtigen. Der Anbieter MUSS ein..... 35**
 - der Anbieter..... 36
 - iv X-Forwa..... 40
 - 5.3 Sicherheits- und Datenschutzerfordernungen an Logging und Monitoring 41**
 - 5.4 Sicherheits- und Datenschutz-Anforderungen an das Security Monitoring 42**
 - 5.5 Sicherheits- und Datenschutz-Anforderungen an die Protokollierung von Administrationsaktivitäten..... 50**
 - 5.6 Sicherheits- und Datenschutz-Anforderungen an die Verarbeitung von Daten mit dem Schutzbedarf "sehr hoch"..... 51**
 - 5.7 Sicherheits- und Datenschutz Anforderungen an dem ZETA Client in FdVs 53**
 - 5.8 Schlüssel Management und Verwaltung in ZETA..... 54**
 - 5.8.1 Nomenklatur für Schlüssel-IDs..... 54
 - 5.8.2 Übersicht der ZETA Guard Schlüssel (Anbieter-Seite)..... 55
 - 5.8.3 ZETA Client Schlüssel (Nutzer-Seite)..... 57
 - 5.8.4 gematik verwaltete Schlüssel (TI)..... 59
 - 5.9 ZETA Abläufe..... 60**
 - 5.9.1 Abläufe für stationäre Clients..... 60
 - 5.9.1.1 *Client Installation und Schlüsselgenerierung*..... 63
 - 5.9.1.1.1 Schlüsselgenerierung auf Windows und Linux Systemen..... 64
 - 5.9.1.1.2 Schlüsselgenerierung auf macOS Systemen..... 65
 - 5.9.1.2 *Client Start mit TPM und ZAS*..... 66
 - 5.9.1.3 *Service Discovery*..... 66
 - 5.9.1.4 *Vorbereitung der Client-Registrierung beim ZETA Guard*..... 68
 - 5.9.1.4.1 ZAS und TPM..... 68
 - 5.9.1.4.2 Secure Enclave..... 69
 - 5.9.1.5 *Client-Registrierung*..... 69
 - 5.9.1.6 *Authentifizierung*..... 72

- 5.9.1.6.1 Client Statement mit ZAS und TPM Attestation.....72
- 5.9.1.6.2 Client Statement mit Apple AppAttest.....73
- 5.9.1.6.3 Token Exchange mit Attestation.....75
- 5.9.1.6.4 Token Request mit grant_type=refresh_token.....78
- 5.9.1.7 Zugriff auf den Resource Server.....80
 - 5.9.1.7.1 Zugriff auf den Resource Server mit ZETA/ASL.....81
 - 5.9.1.7.2 Zugriff auf den Resource Server ohne ZETA/ASL.....83
- 5.9.2 Abläufe für mobile Clients.....85
 - 5.9.2.1 Initialisierung und Schlüsselgenerierung.....86
 - 5.9.2.1.1 Android TEE oder Strongbox.....86
 - 5.9.2.1.2 Apple Secure Enclave.....88
 - 5.9.2.2 Client Registrierung und Authentifizierung.....91
 - 5.9.2.3 Plattformabhängige Attestierung.....93
 - 5.9.2.3.1 Android.....93
 - 5.9.2.3.2 Apple.....95
 - 5.9.2.3.3 Software Attestierung (Fallback).....97
 - 5.9.2.4 Service Discovery.....97
 - 5.9.2.5 Authentifizierung.....97
 - 5.9.2.5.1 Voraussetzungen.....98
 - 5.9.2.5.2 Übersicht.....98
 - 5.9.2.5.3 Teilablauf (A): Authorization Request mit äußerem und innerem PAR. 99
 - 5.9.2.5.4 Teilablauf (B): Nutzerauthentisierung am sektoralen IDP.....102
 - 5.9.2.5.5 Teilablauf (C): Token-Bezug und Ausstellung der ZETA Token.....103
 - 5.9.2.5.6 Authentifizierung für eine andere Resource am gleichen ZETA Guard
.....105
- 5.9.3 Authentifizierung ohne Nutzer-Identität.....106
- 5.9.4 Dienst-zu-Dienst Kommunikation.....108
 - 5.9.4.1 Ausgehende Verbindungen der Dienst-zu-Dienst Kommunikation mit ZG
Client.....110
 - 5.9.4.1.1 Motivation und Abgrenzung.....110
 - 5.9.4.1.2 Architektur.....110
 - 5.9.4.1.3 Konfiguration über die Policy Engine.....111
 - 5.9.4.1.4 Ablauf.....111
 - 5.9.4.1.5 Anforderungen.....113
- 5.9.5 Token Revocation.....114
- 3. Erfolgreiche Client.....116**
- 5.10 Clientsystem und ZETA Client.....117**
 - 5.10.1 Hersteller.....117
 - 5.10.2 Verbindungsaufbau.....118
 - 5.10.2.1 Definition der Endpunkte.....118
 - 5.10.2.2 Adressierung und Gültigkeitsbereich (Audience und Scope).....118
 - 5.10.2.3 Zusammenhang der Token Exchange Komponenten.....119
 - 5.10.2.4 Resultierende Token für den Aufruf von Fachdiensten.....119
 - 5.10.2.5 Versionierung des Token-Ausstellungsverfahrens.....120

- 5.10.2.6 Anforderungen..... 120
- bei Unterstützung..... 123
- 5.10.3 Client-Registrierung..... 123
- 5.10.4 Nutzerauthentifizierung..... 125
- 5.10.5 Session Management..... 125
- 5.10.6 Fehlerbehandlung..... 126
 - 5.10.6.1 Liste der HTTP-Statuscodes..... 126
 - 5.10.6.2 Behandlung von PEP-Fehlern mittels HTTP-Header..... 130
- Für die korrekte Fehlerbehandlung im ZETA..... 130
- 5.10.7 ZETA Attestation Service..... 131
- 5.11 Client Management..... 132**
- 5.12 Client Registrierungsdaten..... 132**
 - 5.12.1 Client Assertion JWT..... 133
 - 5.12.2 Client Statement..... 133
 - 5.12.3 Posture Informationen..... 134
 - 5.12.3.1 TPM Posture..... 134
 - 5.12.3.2 Software Posture..... 134
 - 5.12.3.3 Apple Posture..... 135
 - 5.12.3.4 Android Posture..... 136
 - 5.12.4 Vertrauensmodell, Faktoren und Geltungsbereich..... 137
 - 5.12.5 Erstregistrierung (First Use)..... 139
 - 5.12.6 Folgeregistrierung und E-Mail-Bindung..... 139
 - 5.12.7 Schlüssel-Rollover (Proof-of-Possession)..... 142
 - 5.12.8 Verwaltung von Clients/Geräten..... 143
 - 5.12.9 Außerordentliche Löschung (Out-of-Band) und Protokollierung..... 144
- 5.13 ZETA Guard..... 145**
 - 5.13.1 Klassifizierung der ZETA Guard Komponenten..... 149
 - 5.13.1.1 Unveränderliche Kernkomponenten..... 149
 - 5.13.1.2 Austauschbare Kernkomponenten..... 150
 - 5.13.1.3 Hilfskomponenten und Funktionen..... 150
 - 5.13.2 Kommunikation mit Diensten der gematik..... 151
 - 5.13.3 Deployment Szenarien..... 152
 - 5.13.3.1 Geo-Redundanz..... 152
 - 5.13.4 Provisioning Image für den ZETA Guard..... 152
 - Das Provisioning Image dient der sicheren Bereitstellung und Verteilung von kryptografischen Trust Anchors (Vertrauensankern) und Konfigurationsdaten, die der ZETA Guard zur Verifikation von Signaturen, Attestierungen und Zertifikaten benötigt. Es handelt sich um ein rein datenführendes OCI Image, welches von der gematik signiert ber..... 152
 - 5.13.4.1 Struktur und Inhalt..... 152
 - 5.13.4.2 Integritätsschutz und Verifikation..... 154
 - 5.13.4.3 Verarbeitung durch ZETA Guard Komponenten..... 154
 - 5.13.4.4 Lifecycle und Update-Regeln..... 155
 - 5.13.5 Laufzeitüberwachung..... 155
 - 5.13.5.1 Aktuell eingesetzte Verfahren..... 156
 - 5.13.5.1.1 Pod Security Standards (PSS)..... 156
 - 5.13.5.1.2 Admission Controller..... 156
 - 5.13.5.1.3 Network Policies..... 156
 - 5.13.5.1.4 Service Mesh..... 157
 - 5.13.5.1.5 Ingress und Egress / Gateway..... 157
 - 5.13.5.2 Geplante Verfahren..... 157
 - 5.13.5.2.1 Pod-Überwachung gemäß Cilium Tetragon..... 157

5.13.5.2.2 RBAC-Härtung (Role-Based Access Control).....158

5.14 Policy Enforcement Points.....158

5.14.1 PEP HTTP Proxy..... 158

5.14.2 Sicherheits- und Datenschutz-Anforderungen an den PEP.....165

5.15 Policy Decision Point.....165

5.15.1 Policy Engine..... 165

5.15.2 PDP Authorization Server..... 166

5.15.2.1 PDP Relying Party..... 171

Claims amr und acr bei Token Re..... 172

5.15.2.2 quest..... 172

5.15.2.3 Token-Ausstellung..... 175

5.15.2.4 (RFC 7009)..... 181

r Client-Authentifizierung..... 184

5.15.3 PDP Datenbank..... 186

5.15.4 Sicherheits- und Datenschutzanforderungen an den PDP.....187

5.16 Policies und Daten.....188

5.17 Telemetriedaten-Service.....189

5.18 HSM Proxy.....191

5.18.1 Sicherheits- und Datenschutzanforderungen an den HSM Proxy.....192

5.19 Notification Service.....193

5.19.1 Ablauf des Push-Versands..... 194

5.19.2 Authentisierung..... 196

5.19.3 Rollen der Komponenten..... 197

5.19.4 Anforderungen..... 197

5.19.5 Der Notification Service MUSS sicherstellen, dass die HSM-Proxy-Operationen zur Envelope-Verschlüsselung (Encrypt/Decrypt) sowie das HSM-Wrapping/Entwrapping des Pseudonymisierungsschlüssels (Encrypt/Decrypt) ausschließlich durch eine attestierte, integre VAU-Instanz ausgelöst werden können (attestierter bzw. gepairter HSM Proxy gemäß gemSpec_ZETA A_28746-A_28748; Zugriffsbeschränkung nach Komponente und key_id per mTLS/ABAC gemäß A_28830). Weder die DEK noch entschlüsselte Inhalte oder Klartext-Identifikatoren DÜRFEN die VAU-Vertrauensgrenze verlassen. [<=].....199

5.20 Betrieb.....206

5.20.1 Anforderungen an Hersteller einer ZETA Komponente.....206

5.20.2 Anforderungen an Hersteller eines TI 2.0-Dienstes.....207

5.20.3 Anforderungen an Anbieter eines TI 2.0-Dienstes.....207

5.20.4 Anforderungen für nahtlose Aktualisierungen.....207

5.20.5 Überwachung des Betriebsstatus.....208

5.20.6 Leistungs-Anforderungen.....209

5.20.7 Betriebliche Schnittstellendefinition.....211

5.20.8 Prozesse zur Inbetriebnahme eines ZETA Guard.....213

5.21 Anforderungen an Dienste der TI.....213

5.22 Sicherheitsleistungen des ZETA Guard für Resource Server.....213

5.23 Weitere Leistungen des ZETA Guard für Resource Server.....215

6 Beispiele und Referenzimplementierungen.....216

7 Anhang A - Verzeichnisse.....217

7.1 Abkürzungen..... 217

7.2 Glossar..... 219

7.3 Abbildungsverzeichnis..... 221

7.4 Tabellenverzeichnis..... 222

7.5 Referenzierte Dokumente..... 224

 7.5.1 Dokumente der gematik..... 224

 7.5.2 Weitere Referenzen..... 227

|

1 Einordnung des Dokuments

Dieses Dokument stellt eine übergreifende Spezifikation dar, ohne einen konkreten Bezug zu einem Produkttypen herzustellen. Anforderungen dieses Dokuments werden Produkttypen, Schnittstellen, Komponenten oder Diensten von konkreten Use Cases bzw. von Fachanwendungen zugewiesen.

Die in diesem Dokument beschriebenen Konzepte, Abläufe und Informationsmodelle dienen der Umsetzung der Paradigmen des Zero Trust in der "Telematikinfrastruktur 2.0".

Das Zero Trust-Modell ist ein Sicherheitskonzept, das auf dem Prinzip strenger Zugriffskontrollen und dem grundsätzlichen Misstrauen (kein implizites Vertrauen) gegenüber jedem Kommunikationsteilnehmer beruht, selbst denen, die sich bereits innerhalb eines Netzwerkperimeters befinden. Es handelt sich um ein Sicherheitsrahmenwerk, das erfordert, dass alle Nutzer und deren Clients (Gerät und App), sowohl innerhalb als auch außerhalb der Netzwerkperimeter, authentifiziert, autorisiert und kontinuierlich auf ihre Sicherheitskonfiguration und Sicherheitsnachweise überprüft werden, bevor ihnen Zugriff auf Anwendungen und Daten gewährt oder dieser aufrechterhalten wird. Motiviert durch den „Assume Breach“-Ansatz basiert dieses Architekturdesign-Paradigma im Kern auf dem Prinzip der minimalen Rechte aller Entitäten in der Gesamtinfrastruktur.

1.1 Zielsetzung

Ziel des Dokuments ist die Sammlung der technischen, betrieblichen und testrelevanten Anforderungen an Clients, Komponenten und Dienste, die Zero Trust-Aspekte beinhalten oder nutzen.

Das Ziel des Zero Trust-Ansatzes besteht darin, die IT-Sicherheitslandschaft grundlegend zu transformieren, um den Schutz von Daten, Anwendungen und Systemen vor modernen Bedrohungen und Angreifern zu gewährleisten. Im Gegensatz zu traditionellen Sicherheitsmodellen, die auf dem Konzept eines sicheren Perimeters basieren, setzt Zero Trust auf die Annahme, dass keine Nutzer oder Systeme, unabhängig von ihrem Standort innerhalb oder außerhalb des Netzwerks, von Natur aus vertrauenswürdig sind.

1.2 Zielgruppe

Dieses Dokument richtet sich an Architekten und Entwickler von Komponenten, Diensten, Produkttypen, Schnittstellen und Clients für den Datenaustausch im deutschen Gesundheitswesen.

1.3 Abgrenzungen

Diesem Dokument ist kein Produkt- oder Anbietertyp zuzuordnen. Anforderungen in diesem Dokument finden Anwendung in Produkt- und Anbietertypen von konkreten Fachanwendungen bzw. Use Cases.

1.4 Methodik

1.4.1 Anforderungen

Anforderungen als Ausdruck normativer Festlegungen werden durch eine eindeutige ID sowie die dem [RFC2119] entsprechenden, in Großbuchstaben geschriebenen deutschen Schlüsselworten MUSS, DARF NICHT, SOLL, SOLL NICHT, KANN gekennzeichnet.

Da in dem Beispielsatz „Eine leere Liste DARF NICHT ein Element besitzen.“ die Phrase „DARF NICHT“ semantisch irreführend wäre (wenn nicht ein, dann vielleicht zwei?), wird in diesem Dokument stattdessen „Eine leere Liste DARF KEIN Element besitzen.“ verwendet. Die Schlüsselworte werden außerdem um Pronomen in Großbuchstaben ergänzt, wenn dies den Sprachfluss verbessert oder die Semantik verdeutlicht.

Anforderungen werden im Dokument wie folgt dargestellt:

<AFO-ID> - <Titel der Afo>

Text / Beschreibung

[<=]

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

2 Features und Epics

Der folgende Abschnitt gibt einen groben Überblick über die Features und Epics, die sich in Anwendungen wiederfinden, wenn sie nach dem Paradigma des Zero Trust umgesetzt werden. Diese Epics sind als Enabler zu verstehen, um Fachanwendungen einen sicheren Verbindungsaufbau zwischen Clients und Backenddiensten zu ermöglichen. Es werden keine User Stories formuliert, da für den Verbindungsaufbau keine Nutzerinteraktion angedacht ist.

Im Rahmen der Nutzeridentifikation (Authentifizierung) findet eine Verifikation ausgegebener Authentisierungsmerkmale statt, deren Nutzerinteraktion als Teil der Spezifikation des Identity Managements beschrieben sind.

2.1 Client-Registrierung

Gemäß des Zero Trust-Ansatzes ist jeder Schnittstellenaufruf potentiell gefährlich, soweit nicht anders festgestellt. Dazu zählt auch das Vertrauen in bekannte bzw. Misstrauen in unbekannte Clients. Clients sind hier als Kombination aus Gerät (z. B. PC oder mobile Device) und App (Client-Software wie FdV oder Primärsystem) definiert. Um Clients wiedererkennbar zu machen, muss eine Registrierung dieser erfolgen. Sind in der Registrierung zusätzliche Sicherheitsmerkmale über den Client und den Aufrufkontext feststellbar, stärken diese das Vertrauen in nachfolgenden Aufrufen fachlicher Schnittstellen.

2.1.1 Wiedererkennung bekannter Clients

Die Wiedererkennung bekannter Clients und deren Bindung an identifizierbare Nutzer des Gesundheitswesens muss über eine Registrierung erfolgen. Die Identifikation des Nutzers erfolgt dabei über ein unterstütztes Identifikationsmerkmal (SmartCard oder digitale Identität) und einen selbstgewählten, vom System unterstützten zweiten Faktor (E-Mail, SMS, etc.).

2.1.2 Client-Attestation

Die Client-Attestation dient dem Nachweis der Integrität einer Client-Instanz (z. B. einer Frontend-Applikation oder eines Primärsystems) gegenüber einem ZETA Guard. Im Rahmen der Client-Registrierung reicht der Client einen kryptografisch signierten Nachweis ein, der belegt, dass die Anwendung nicht durch unbefugte Dritte modifiziert wurde (Integrität).

Dieser Mechanismus ist ein wesentlicher Bestandteil des Zero-Trust-Modells der TI 2.0, um den Missbrauch von Client-Identitäten durch manipulierte Software oder automatisierte Skripte zu verhindern.

Die ZETA Client-Attestation erfolgt nach [RFC9334].

2.1.3 Device Security Rating

Zum Einschluss bzw. Ausschluss bestimmter Eigenschaften von Clients, sollen selbige einer automatischen Sicherheitsprüfung unterzogen werden können (Device Security Rating - DSR), soweit es die gegebenen Plattformmechanismen erlauben.

2.2 Policy Enforcement

Für den Zugriff auf personenbezogene und medizinische Daten und zur Sicherstellung der Integrität, Verfügbarkeit, Vertraulichkeit und Authentizität transportierter Daten gelten Regeln. Diese fachlichen, technischen und organisatorischen Regeln gelten bei jedem Zugriff auf Daten, die über eine Schnittstelle zugreifbar gemacht werden.

2.2.1 Zugriffsschutz

Das Policy Enforcement soll als eine Art Gatekeeper bzw. Türsteher den Zugriff auf Schnittstellen von Backendservices durch beliebige Clients durchsetzen. Grundlage ist das Vertrauen in eine Policy-Entscheidung durch eine Komponente zur Auswertung eines Regelwerks.

2.2.2 HTTP Proxy

Der HTTP Proxy stellt sicher, dass nur Requests mit gültigem Access Token sowie bestandenen zusätzlichen Prüfungen an den Resource Server weitergeleitet werden. Welche Prüfungen zusätzlich erfolgen, wird über Attribute im Access Token gesteuert.

2.3 Decide from Policies

Die Menge an Regeln für die Gewähr eines Zugriffs auf Daten oder Schnittstellen speist sich aus gesetzlichen Forderungen bzw. Verboten, Vertragskonstrukten, Sicherheitsmechanismen, Architekturentscheidungen und Informationen aus der "Umgebung" des Betriebs von Clients und Backendservices.

2.3.1 Maschinenlesbare Zugriffsregeln

Die Menge (potentiell) geltender Regeln zur Absicherung des Zugriffs auf Daten und Dienste formt ein Set von Policies. Um im Fall eines Zugriffsversuchs schnell entscheiden zu können, sollen diese Regeln maschinenlesbar definiert sein. Die Regeln sollen zusätzlich menschenlesbar sein, um die Entwicklung und Wartung der Regeln zu vereinfachen.

2.3.2 Ein reproduzierbares Ja/Nein

Die Auswertung eines komplexen Regelwerks liefert bei identischen Eingangsparametern reproduzierbar das identische Ergebnis.

2.3.3 Policies nach Betroffenheit

Regeln beziehen sich auf verschiedene Aspekte einer Zugriffsentscheidung. Es gelten fachliche Regeln, Regeln zur Benutzung von Clients und ebenso technische Regeln sowie solche, die Betriebsumgebung von Backenddiensten betreffend.

2.4 Policy-Information und -Administration

Die Aufgabe des Policy Information Point (PIP) ist es, relevante Attribute und Informationen zur Entscheidungsfindung (Daten) zu liefern, während der Policy Administration Point (PAP) für die Verwaltung und Bereitstellung der Richtlinien (Policies) verantwortlich ist.

2.4.1 Policy-Verwaltung

Eine Policy-Entscheidung kann Eingangsinformation für andere Policies sein, ebenso kann das Ändern von Rahmenbedingungen oder eine Anomalie-Erkennung zur Beeinflussung von Policies führen. Aus diesem Grund führen Beobachtungen über Policy-Entscheidungen zu Informationen über das Gesamtsystem, die als Eingangsdaten für nachfolgende Policy-Entscheidungen herangezogen werden. Daneben ist es erforderlich, Anpassungen am Regelwerk dem System über authentizitäts- und integritätsgeschützte Wege bekannt zu machen.

2.4.2 Monitoring

Durch ein Monitoring von Betriebsparametern und Telemetriedaten wird die Durchsetzung von Policies sowie die Auswirkung möglicher Policy-Änderungen transparent.

2.5 Authorization

Die Autorisierung (Authorization) beschreibt innerhalb von ZETA den Prozess der Prüfung und Erteilung von Zugriffsrechten auf geschützte Ressourcen. Während die *Authentisierung* die Identität eines Nutzers oder Systems feststellt, definiert die *Autorisierung*, welche Operationen dieser Teilnehmer auf welchen Daten oder Diensten ausführen darf.

In ZETA Guard bildet die Autorisierung den Kern der Zugriffskontrolle. Ziel ist es, das Prinzip der minimalen Rechtevergabe (Principle of Least Privilege) konsequent umzusetzen.

2.5.1 Autorisierung auf Basis von Policy-Entscheidungen

Die Autorisierung von Zugriffen auf Daten oder Schnittstellen wird bei positiver Entscheidung durch ein Set von Policies gewährt. Die Zugriffsentscheidung und -gewähr bettet sich in eine Verkettung von Informationen und von Aufrufen verschiedener Schnittstellen ein, die dem fachlichen Aufruf einer Schnittstelle bzw. Abruf von Daten voranstehen. Stand der Technik dieses Flows mehrerer Aufrufe und der dabei transportierten Informationen ist der OAuth2-Standard, vgl. [RFC6749 et al.].

2.5.2 Client Authentication

Menschen und Clients werden anhand sicherer Merkmale authentifiziert, die Identifikation ist nachrangig bzw. in nachgelagerten fachlichen Anwendungsfällen bzw. in fachlichen Zugriffsregeln relevant.

Kann ein Mensch oder Client nicht sicher authentifiziert werden oder wird der Authentifizierung zeitlich oder anderweitig nicht vertraut oder passen die Umgebungs- bzw. die den Aufruf begleitenden Parameter nicht zum Vertrauen in die Authentifizierung, wird eine erneute Authentifizierung als erforderlich angesehen ("Step-Up-Authentication").

3 Einordnung in die TI 2.0

Die TI 1.0 bildet eine Infrastruktur, deren Sicherheit auf der sicheren Zugangskontrolle zu einem geschlossenen zentralen Netzwerk mit Diensten beruht. In der TI 2.0 werden die Dienste dezentral im Internet angeboten und bedürfen daher eines Schutzes vor unberechtigtem Zugriff pro Dienst. Dieser Schutz wird nach dem Zero Trust-Paradigma durch den Policy Enforcement Point und den Policy Decision Point durchgesetzt.

Diese übergreifende Spezifikation richtet Anforderungen an Akteure, die sich über das Internet miteinander vernetzen. Diese Akteure seien im Folgenden einerseits Clients (Software: Aufrufende einer Schnittstelle, Anfragende an einen Datenabruf oder -zugriff, wird auf einem bestimmten Client ausgeführt), häufig bedient durch einen Menschen, und Backendservices (Software: bereitstellende Schnittstelle, Datenbereitstellung etc.) auf der anderen Seite.

Zur Absicherung der Clients und Backendservices werden Anforderungen erhoben, die in konkreten Softwarekomponenten innerhalb dieser Akteure umzusetzen sind. Die Separierung der Zero Trust-Mechanismen in unterschiedliche Komponenten folgt der Zero Trust-NIST-Referenzarchitektur.

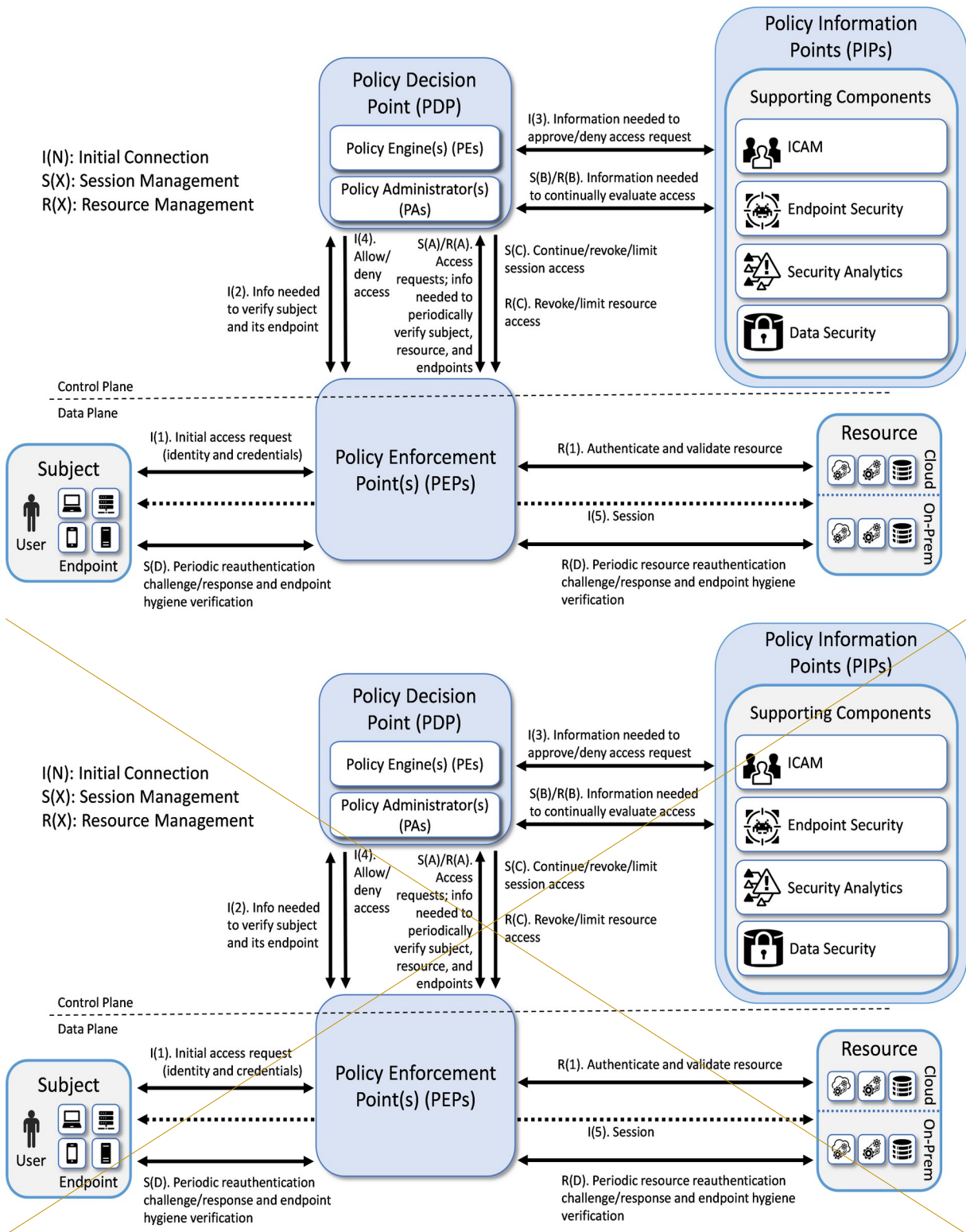


Abbildung 1: NIST Zero Trust-Referenzarchitektur, Quelle [NIST_SP1800-35_FIG1]

Im Architekturkonzept der TI 1.0 werden konkrete Umgebungsannahmen zu Consumer Zonen, Secure Consumer Zonen, Plattformzonen, Personal Zonen usw. getroffen, in denen kein (Personal Zone) bzw. ein gewisses Sicherheitsniveau (überall sonst)

axiomatisch angenommen wird. Das Zero Trust-Konzept löst sich von der Aufteilung in verschiedene Zonen, insbesondere, da keine TI-Plattform-Produkttypen in der Kommunikation zwischen Clients mit Diensten involviert werden.

3.1 ZETA als Umsetzung der Zero Trust Architektur

Im Folgenden ist eine generische Produkttypzerlegung für die Umsetzung der Zero Trust-Referenzarchitektur einer TI 2.0- Fachanwendung dargestellt. Die Zero Trust Ausprägung der Telematikinfrastruktur wird Zero Trust Access (ZETA) genannt.

In diesem Pattern greift ein Nutzer über ein Clientsystem (inkl. ZETA Client) auf Daten eines TI 2.0-Dienstes zu. Ein Nutzer wird hier als der Akteur definiert, der einen TI 2.0-Dienst nutzt. Dabei kann es sich um einen Versicherten oder um einen Mitarbeiter in einer Organisation des Gesundheitswesens handeln (z. B. LEI, LEO, Krankenkasse). Ein TI 2.0-Dienst setzt sich zusammen aus einem ZETA Guard und mindestens einem Resource Server. Der ZETA_Guard wird als eingebettetes Modul engmaschig mit dem Resource Server verbunden und vom Anbieter des TI 2.0-Dienstes zwingend mit betrieben. Der Anbieter implementiert zusätzliche Komponenten, um seine Infrastruktur vor Angriffen aus dem Internet zu schützen und um die Zugriffe zu optimieren (z. B. Content Delivery Network, eigener Ingress und Load Balancer, Web Application Firewall). Das folgende Bild zeigt eine Übersicht der beteiligten Komponenten in der Vernetzung zwischen einem Clientsystem (links grün) und einem Backendservice (rechts grün: Resource Server).

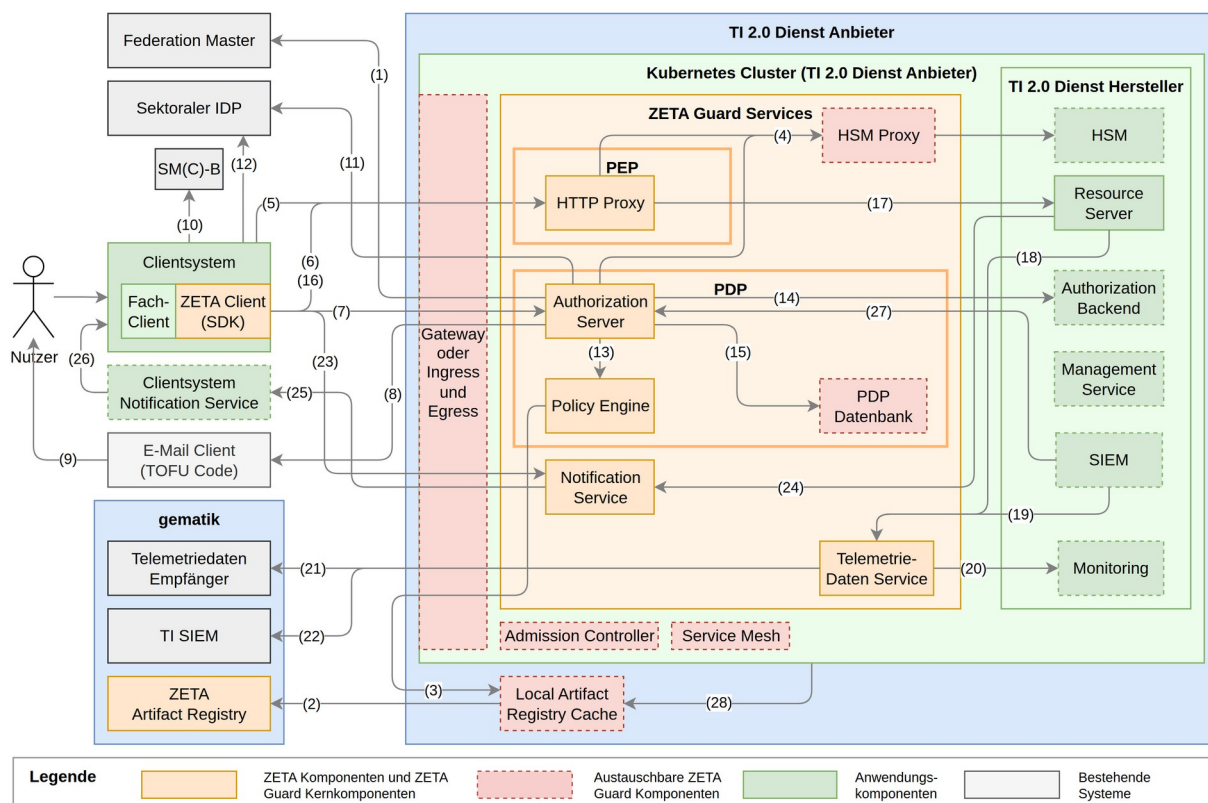


Abbildung 2: Abbildung Zero Trust-ZETA-Architektur der TI 2.0

Die obige Abbildung zeigt die Einbettung von ZETA bezogenen, logischen Komponenten

(orange) in die Aufrufkette zwischen einem Clientsystem und einem Resource Server (grün). Darüber hinaus soll durch die roten und gestrichelt umrandeten Komponenten hervorgehoben werden, dass je nach Einsatzumgebung und Anforderungen an die Umgebung verschiedene Deployment-Szenarien unterstützt werden. Dargestellt sind zusätzlich heute bereits vorhandene und genutzte Komponenten und Dienste, die für die Nutzerauthentifizierung (z. B. eGK und IDP) bzw. die Betriebsüberwachung (z. B. mittels gematik Telemetriedaten Empfänger) in Anwendungsfällen der TI 2.0 weitergenutzt werden können (grau). In diese Abbildung sind diverse Architekturentscheidungen eingearbeitet, die im Kapitel 4 und 5 erläutert bzw. spezifiziert werden.

3.2 Kurzbeschreibung der Komponenten

Zu ZETA gehören ZETA Guard, ZETA Client und die ZETA Artifact Registry. Der ZETA Client implementiert clientseitig die Schnittstellen des ZETA Guard und die Attestation-Funktionen des Clientsystems. Der ZETA Client wird von einem Fach-Client initialisiert und mit fachlichen HTTP Requests aufgerufen. ZETA Client führt alle Schritte aus, um ein Access Token vom ZETA Guard zu erhalten, um dann mit diesem Access Token den Request vom Fach-Client an den ZETA Guard zu senden. Die Response wird an den Fach-Client weitergeleitet. Die ZETA Artifact Registry stellt signierte Images für die ZETA Guard Komponenten sowie signierte Policies und Daten für die ZETA Guard Policy Engine bereit. Zusätzlich werden Provisioning Daten wie aktuelle TSL und TPM Hersteller-Stammzertifikate bereitgestellt.

ZETA Guard besteht aus folgenden Komponenten:

- PEP HTTP Proxy: Die zentrale Durchsetzungsinstanz für den Datenverkehr.
- PDP Authorization Server: Die Instanz zur Ausstellung von Access Token und Session-Management.
- PDP Policy Engine (OPA): Die Komponente zur Auswertung der Zugriffsregeln.
- Telemetriedaten-Service: Der OpenTelemetry Collector für sicherheitsrelevante Ereignisse, Traces, Log-Daten und Metriken.
- Notification Service: Optionale ZETA Guard Komponente zur einheitlichen Bereitstellung von Notification-Funktionen für mobile Clients.
- PDP Datenbank: Persistenzschicht für Sessions sowie Nutzer- und Client-Daten.
- HSM Proxy: Schnittstelle zu den ZETA Guard Komponenten und das Hardware-Sicherheitsmodul des Anbieters.
- Local Artifact Registry Cache: Lokales Repository zur Bereitstellung der Images innerhalb der Anbieter-Umgebung.
- Gateway oder Ingress und Egress: Netzwerk-Einstiegspunkt für den Kubernetes Cluster und kontrollierter Grenzübergang für den ausgehenden Verkehr.
- Admission Controller: Hilfskomponente, die Anfragen an den Kubernetes API Server überwacht um Security Policies (wie z. B. die Prüfung der Image-Signatur, bevor ein Image ausgeführt werden kann) durchzusetzen.
- Service Mesh: Infrastrukturschicht, die die Kommunikation zwischen den einzelnen Microservices innerhalb eines Clusters steuert, sichert und überwacht.

3.3 Kurzbeschreibung der Schnittstellen

(1) Die ZETA Guard Instanz wird beim TI Federation Master registriert. Dadurch wird die Zugehörigkeit des ZETA Guard zur TI (in einer bestimmten Umgebung wie Prod, Ref oder Test) durch Signatur des Federation Masters verifizierbar. In einer Folgeversion wird ZETA Guard im Authorization Server Well-known ein Trustmark vom Federation Master enthalten, das die Zugehörigkeit zur TI nachweist.

(2) Die gematik stellt eine ZETA Artifact Registry bereit, die Helm Charts, Images für die ZETA Guard Komponenten und Provisioning Images als OCI Container Images bereitstellt. Alle Images haben eine Signatur mit einer gematik Identität. Anbieter von TI 2.0-Diensten betreiben einen lokalen Artifact Registry Cache, der die ZETA OCI Images direkt in der Umgebung des Anbieter verfügbar macht. In einer Folgeversion wird der Local Artifact Registry Cache Teil von ZETA Guard, um die Anbieter von TI 2.0 Diensten zu entlasten.

(3) Die Policy Engine erhält die Policies und Daten direkt aus der ZETA Artifact Registry.

(4) Die Schnittstelle des HSM Proxy Richtung ZETA Guard Komponenten ist spezifiziert und ermöglicht die einheitliche Anbindung von HSMs verschiedener Hersteller. Der HSM Proxy enthält aktuell nur Funktionen, die von ZETA Guard Komponenten benötigt werden. Grundsätzlich soll der HSM Proxy auch Komponenten des Resource Servers den einheitlichen Zugriff auf HSMs ermöglichen. Fehlende Funktionen werden nach Bedarf ergänzt.

(5) Falls eine Mandantentrennung verwendet wird, dann kann der Fachclient durch Abfrage eines Well-known JSON-Dokuments die zu verwendende URL des Resource Servers ermitteln.

(6) Der ZETA Client bietet dem Fach-Client Operationen zur Initialisierung und zum Versenden von HTTP Requests an. Das heißt, der Fach-Client erstellt einen vollständigen HTTP Request inkl. fachlich benötigter Header und ruft den ZETA Client zur Ausführung des Requests auf. Der ZETA Client ermittelt aus der URL des HTTP Requests im ersten Schritt die Well-known Metadaten des Resource Servers und des Authorization Servers und konfiguriert sich für die Nutzung der ZETA Guard Instanz.

(7) Aus dem Authorization Server Well-known hat der ZETA Client die Endpunkte für die Client-Registrierung und Autorisierung ermittelt und beginnt den OAuth Flow.

(8) Bei mobilen Clients erfolgt eine Trust On First Use Bindung des Clients an eine E-Mail-Adresse des Nutzers.

(9) Der Nutzer muss einen Code aus der Mail vom ZETA Guard im mobilen Clients eingeben, um den Registrierungsprozess des mobilen Clients fortzusetzen.

(10) Bei Primärsystemen erfolgt die Authentifizierung des Nutzers (hier der Organisation) durch ein SMC-B-signiertes Token per OAuth Token Exchange. Die Signatur kann durch Nutzung eines Konnektors oder TI Gateways eraber auch durch Nutzung eines direkt am Primärsystem angeschlossenen Kartenterminals erfolgen.

(11), (12) Mobile Nutzer werden mit Hilfe des sektoralen IDPs und des OIDC Flows authentifiziert.

(13) Nachdem die Session-, Client- und Authentifizierungsdaten am Authorization Server validiert wurden, werden diese Daten an die Policy Engine übergeben. Die Policy Engine wendet die Daten auf die Policies und vorgegebenen Wertebereiche an, ermittelt eine Entscheidung und sendet die Entscheidung an den Authorization Server.

(14) Wenn eine Anwendung eine zusätzliche Authorization benötigt, dann wird der Authorization Server so konfiguriert, dass das Authorization Backend für weitere Autorisierungs-Schritte angefragt wird.

(15) Der Authorization Server speichert seine Session-, Client- und Userdaten in der PDP Datenbank. Wenn die Policy Engine eine "Allow"-Entscheidung gefällt hat und auch kein Verbot vom Authorization Backend vorliegt.

(16) Der ZETA Client sendet einen Request Richtung Resource Server. Der HTTP Proxy erlaubt den Zugriff auf Daten des Resource Servers, wenn ein gültiges Access Token im Authorization Header enthalten ist.

(17) Nach erfolgreicher Prüfung des Access Token und des DPoP Token (wenn vorhanden auch des PoPP Token) erfolgt die Weiterleitung des Requests zum Resource Server. Falls ZETA/ASL verwendet wird, erfolgt zuerst der ASL-Verbindungsaufbau.

(18) Vom Resource Server werden Traces und die Selbstauskunft an den Telemetriedaten-Service gesendet. Der Telemetriedaten-Service empfängt von allen ZETA Guard Komponenten Traces, Logs und Metriken. Die Verbindungspfeile dafür sind nicht dargestellt. Zum Einsatz kommt OpenTelemetry. Alle von OpenTelemetry unterstützten Protokolle können vom Anbieter durch Konfiguration des Telemetriedaten-Service verwendet werden, um Daten an den Telemetriedaten-Service zu senden und vom Telemetriedaten-Service zu empfangen. Empfohlen wird das native OTLP Protokoll über gRPC.

(19) Der Telemetriedaten-Service empfängt vom SIEM des TI 2.0-Dienst-Anbieters Security Events.

(20) Der Telemetriedaten-Service kann Monitoring-Daten der ZETA Guard-Komponenten an das Monitoring des TI 2.0-Dienst-Anbieters senden.

(21) Traces, Metriken und Log Events werden an den Telemetriedaten-Empfänger der gematik weitergeleitet.

(22) Security Events werden an das TI SIEM der gematik weitergeleitet.

(23) Das Clientsystem kann über den ZETA Client eine Push-Konfiguration im ZETA Guard Notification Service einrichten.

(24) (25) Wenn Ereignisse eintreten, für die der Resource Server eine Push Nachricht für den Nutzer generiert, dann werden entsprechend der vorhandenen Push-Konfigurationen des Nutzers im Notification Service, Benachrichtigungen an den Client gesendet.

(26) Über den Clientsystem Notification Service (und hier nicht dargestellte Notification Services des mobile OS Herstellers) werden Notifications an das Clientsystem gesendet.

(27) Mittels Shared Signals kann das Monitoring (oder eine andere Komponente des TI 2.0-Dienst-Anbieters) die Session eines Clients beenden und somit eine neue Authentifizierung erzwingen. Aktuell ist die Shared Signals Unterstützung durch SIEM Systeme und Infrastrukturkomponenten noch nicht hinreichend gegeben. Für die Session-Beendigung wird daher werden Shared Signals bis auf weieine anwendungsspezifische, serverseitige Auslösung der Session-teres noch nicht vmination über die Plug-In-Schnittstelle des Authorization ZETA-unterstützServers umgesetzt.

(28) Der Kubernetes Cluster bezieht seine Deployment-Images aus dem lokalen Artifact Registry Cache sowie die Provisioning Daten für ZETA Guard (TSL, roots.json, TPM-Hersteller Stammzertifikate, Federation Master URL).

Der Admission Controller setzt durch, dass nur von der gematik signierte OCI Container Images der ZETA Guard Kernkomponenten ausgeführt werden können.

Über ein Service Mesh kann sichergestellt werden, dass nur die Komponenten, die miteinander kommunizieren dürfen, eine Verbindung zueinander aufbauen können.

Der optionale Management Service überwacht die Konfiguration der ZETA Guard-Komponenten und setzt durch, dass die im Git Repository der ZETA Guard Instanz gespeicherte Konfiguration ausgeführt wird. Dadurch wird ein Configuration-Drift unterbunden.

4 Technisches Konzept

Im Kapitel zuvor wurden zwei Abbildungen vorgestellt, welche technischen ZETA Guard-Komponenten (orange) an der Umsetzung fachlicher Anwendungsfälle von Clients, Komponenten und Backendservices von Fachanwendungen (grün) beteiligt sind. Im Folgenden werden diese technischen Komponenten genauer beschrieben und eingeordnet, welche Rolle sie in einer Architektur nach dem Zero Trust-Paradigma einnehmen.

Zero Trust in der TI zeichnet sich über folgende Eigenschaften aus:

- Registrierung des Clients (Gerät und App) zu einer Identität
- Attestation der Client-Eigenschaften
- Bereitstellung einer von Maschinen interpretierbaren Policy durch die gematik
- Einheitliches Durchsetzen der Policy durch den ZETA Guard
- Sicherstellung des Sicherheitszustands der gesamten TI, Anbieter übergreifend
- Telemetrie und Monitoring

4.1 ZETA Guard

Der ZETA Guard besteht aus Policy Enforcement Point (PEP), Policy Decision Point (PDP) sowie betriebsunterstützenden Komponenten (Management Service und Telemetriedaten Service). Der PEP besteht aus einem HTTP Proxy. Der PDP enthält die Policy Engine, den Authorization Server und eine Datenbank. Jeder TI 2.0 Dienst hat einen ZETA Guard zum Schutz des Dienstes vor unberechtigtem Zugriff. Der ZETA Guard wird in der Verantwortung des TI 2.0-Dienst-Anbieters betrieben.

4.2 Policy Enforcement Point (PEP)

Ein Policy Enforcement Point (PEP) ist eine Schlüsselkomponente im Zero Trust-Paradigma, der darauf abzielt, dass Sicherheitsmodell von einem vertrauensbasierten auf ein verifizierungsbasiertes umzustellen. Der PEP dient dazu, den Zugriff auf Ressourcen, basierend auf vordefinierten Richtlinien, zu kontrollieren und durchzusetzen. Im Kontext der TI 2.0 übernimmt der PEP folgende Funktionen:

- Der PEP agiert als HTTP Proxy, der den Datenverkehr zwischen Clientanwendungen und den zu schützenden Ressourcen kontrolliert. Dadurch kann der PEP den gesamten Datenverkehr überwachen und filtern, um sicherzustellen, dass er den festgelegten Sicherheitsrichtlinien entspricht.
- Der PEP stellt die Außenschnittstelle des Dienstes dar, in den er integriert ist.

Insgesamt agiert der PEP als Kontrollpunkt in der Zero Trust-Architektur, der sicherstellt, dass nur autorisierte Nutzer und Clients Zugriff auf die Ressourcen eines Dienstes erhalten und dass dabei die definierten Sicherheitspolicies eingehalten werden. Die Entscheidung zwischen verschiedenen Policies auf Basis der vom Client übergebenen Signale, Sicherheitsnachweise und Token trifft der Policy Decision Point. Am PEP werden Betriebsdaten erhoben, verarbeitet und dem Telemetriedaten Service im ZETA Guard zur Verfügung gestellt.

4.3 Policy Decision Point (PDP)

Ein Policy Decision Point (PDP) ist die wesentliche Komponente im Zero Trust-Paradigma, die Zugriffsentscheidungen trifft, indem sie Richtlinien (Policies) interpretiert und anhand dieser Richtlinien Zugriffsanfragen bewertet. Folgende Funktionen eines PDP sind von besonderer Bedeutung:

- Der PDP fungiert als OAuth2 Authorization Server und verwaltet die Autorisierung von Nutzeranfragen auf geschützte Ressourcen. Zudem überwacht der Authorization Server die Nutzersessions, um sicherzustellen, dass sie gültig sind und den Sicherheitsrichtlinien entsprechen. Der Authorization Server ist als vertrauenswürdige Relying Party im föderierten Identitätsmanagement registriert. Dadurch kann der Authorization Server Identitätsinformationen von Nutzern sicher und vertrauenswürdig beziehen und bei Bedarf eine (erneute) Nutzerauthentifizierung an die IDPs delegieren. Der Authorization Server stellt sicher, dass nur authentifizierte Nutzer mit registrierten Clients Zugriff auf die geschützten Ressourcen erhalten.
- Der PDP ermöglicht am Authorization Server die dynamische Registrierung von Clients, die auf geschützte Ressourcen zugreifen möchten. Dies umfasst auch die Offband-Bestätigung, bei der zusätzliche Sicherheitsmechanismen (Verifikation via E-Mail) verwendet werden, um die Identität und Integrität (plattformabhängig) der registrierten Clients zu überprüfen.
- Der PDP analysiert und interpretiert in der Policy Engine die Sicherheitsrichtlinien, die im Rahmen des Zero Trust-Modells definiert sind. Diese Policies können Kriterien wie Nutzeridentität, Gerätetyp, Standort, Zeitpunkt der Anfrage und andere Kontextinformationen ("Signale") enthalten, die relevant für die Zugriffsentscheidung sind. Basierend auf der Interpretation der Policies trifft die Policy Engine Entscheidungen darüber, ob eine Zugriffsanfrage auf eine bestimmte Ressource genehmigt oder abgelehnt wird. Diese Entscheidungen erfolgen auf Plattformebene, was bedeutet, dass die Policy Engine die Zugriffsanfragen im Kontext der gesamten Plattform oder des Netzwerks bewertet, und nicht isoliert betrachtet. Die Zugriffsentscheidung resultiert dann in der Ausstellung eines Access Token, das für den konkret angefragten Zugriff verwendet wird (siehe Policy Enforcement).
- Die Policy Engine verwendet dabei die Informationen, die sie von der ZETA Artifact Registry bezieht und die Daten der Zugriffsanfrage vom Authorization Server, um die Zugriffsentscheidung zu treffen. Dazu gehören nicht nur die Policies selbst, sondern auch Echtzeitinformationen über den Zustand von Nutzeridentitäten, Clients und andere Kontextinformationen, die für die Bewertung der Zugriffsanfrage relevant sind.

Am PDP werden Betriebsdaten erhoben, verarbeitet und dem Telemetriedaten Service im ZETA Guard zur Verfügung gestellt.

4.4 ZETA Client

Im Kontext von Zero Trust der TI stellt der "ZETA Client" eine logische Komponente innerhalb einer Clientanwendung (Primärsystem (PS), Frontend des Versicherten (FdV) etc.) dar.

Ein ZETA Client im Zero Trust-Modell wird nicht als vertrauenswürdig angesehen, sondern muss - genauso wie alle Komponenten im Netzwerk - kontinuierlich authentifiziert und autorisiert werden. Die Zugriffsentscheidungen werden basierend auf aktuellen Richtlinien, Kontextinformationen, Bedrohungsinformationen und insbesondere in Kenntnis des diesen Client benutzenden Nutzers getroffen.

Die Aufgaben des ZETA Clients sind:

- Erzeugung, sichere Speicherung und Prüfung der kryptographischen App/Geräte Identität
- Erzeugung der App/Gerät-Attestierung und Ermittlung und Übertragung der Eigenschaften der Laufzeitumgebung (Betriebssystem, Betriebssystem Version, etc.)
- Implementierung des OAuth Flows
- Management der Sessions inkl. Verwaltung der Access und Refresh Token.
- Management der Client-Registrierungen

4.5 Policy-Information und -Administration

Im Zero Trust-Paradigma spielen der Policy Information Point (PIP) und der Policy Administration Point (PAP) wichtige Rollen bei der Verwaltung und Durchsetzung von Sicherheitsrichtlinien bzw. Policies. Zusammen ermöglichen der PIP und der PAP eine zentrale Verwaltung und Bereitstellung von Policies im Zero Trust-Netzwerk.

Der PAP stellt Policies bereit und der PIP stellt die Daten für die Policies bereit, sodass sich aus beiden ein Regelwerk ergibt, das die Policy Engine des PDP anwendet, um zu entscheiden, ob eine Kommunikationsanfrage zulässig ist.

4.5.1 Policy Information Point (PIP)

Der PIP ist für die Bereitstellung von Informationen über Sicherheitsrichtlinien zuständig. Er dient als zentraler Informationsdienst, der Policy Engines im Zero Trust-Netzwerk Zugriff auf aktuelle Sicherheitsrichtlinien ermöglicht. Der PIP kann Attribute wie Nutzerrollen, Zugriffsrechte, Clientzustände und andere Kontextinformationen bereitstellen, die von den Policy Engines für die Zugriffsentscheidung benötigt werden. Der PIP kann Daten aus verschiedenen Quellen beziehen, einschließlich einer zentralen Richtliniendatenbank, externen Identitätsanbietern, Sicherheitsinformationen von Clients und anderen Quellen.

4.5.2 Policy Administration Point (PAP)

Der PAP ist für die Verwaltung und Konfiguration von Sicherheitsrichtlinien verantwortlich. Er bietet eine Schnittstelle oder eine Konsole, über die Richtlinien in hoheitlicher Verantwortung definiert, geändert und gelöscht werden können. Policy-Administratoren können im PAP Zugriffsregeln, Autorisierungsniveaus, Bedrohungsabwehrmaßnahmen und andere Sicherheitsrichtlinien festlegen. Der PAP ermöglicht es Policy-Administratoren, Richtlinien - basierend auf verschiedenen Kriterien wie Nutzerrollen, Gruppenzugehörigkeit, Standorten und Clientattributen - zu differenzieren. Änderungen an den Sicherheitsrichtlinien, die im PAP vorgenommen werden, werden von den Policy Engines im Zero Trust Netzwerk übernommen. Das Vertrauen in bereitgestellte und angepasste Policies wird über Signaturen für die Sicherstellung von Integrität und Authentizität jeder Policy sichergestellt.

4.5.3 PIP und PAP Ausprägung in ZETA

In ZETA Guard wird als Policy Engine der Open Policy Agent (OPA) verwendet. OPA bezieht Policies (PAP) und Daten (Wertebereiche für die Policies; PIP) zusammengefasst

(OPA Bundle) über ein OCI Container Image aus der ZETA Artifact Registry der gematik. Die einzelnen Dateien im OPA Bundle sind mit einer Identität der gematik signiert.

Dadurch reduziert sich die Bereitstellung der PIP und PAP Daten zu einem Download in einer OCI Registry. Die Anforderungen an den PIP und PAP Service (siehe [5.139- Policies und Daten](#)) beziehen sich daher auf den Policies und Daten CI/CD-Prozess zur Entwicklung und Bereitstellung der Policies und Daten.

4.6 Client-Registrierung

Die Client-Registrierung dient dazu, eine konkrete Installation eines Clientsystems eindeutig zu identifizieren und, wenn möglich, mit einem Nutzer zu verknüpfen. Dabei werden sowohl statische Eigenschaften des Clientsystems als auch dynamische Eigenschaften der Client-Instanz übermittelt.

Die Clientattribute werden vom Client und von den Plattformen der Endgeräte geliefert. Ihre Erhebung erfolgt im ZETA Client des Endgeräts mittels plattformspezifischer Attestierungs- und Erhebungsmechanismen. Die Attribute sind daher für die jeweilige Plattform und ihr Sicherheitsmodell spezifisch. [Siehe Kapitel 25.9- Client-Registrierungsdaten-](#)

4.7 Monitoring

Das Monitoring im Kontext von Zero Trust ist ein entscheidender Aspekt, um die Sicherheit des Netzwerks und der Ressourcen kontinuierlich zu überwachen und potenzielle Bedrohungen oder Anomalien zu identifizieren.

4.7.1 Security Information and Event Management (SIEM)

SIEM-Systeme spielen eine zentrale Rolle im Monitoring im Zero Trust-Paradigma. Sie sammeln Daten aus verschiedenen Quellen wie Protokollen, Ereignissen und Alarmen von Sicherheitskomponenten im Netzwerk. Durch die Analyse dieser Daten in Echtzeit können SIEM-Systeme potenzielle Sicherheitsvorfälle erkennen und Anomalien identifizieren.

4.7.2 Shared Signals

Shared Signals [Shared Signals] sind Hinweise oder Indikatoren für Sicherheitsvorfälle, die von verschiedenen Systemen und Quellen im Netzwerk gemeinsam genutzt werden. Diese Signale können von verschiedenen Sicherheitskomponenten wie Firewalls, Endpunktschutzsystemen, Intrusion Detection Systems (IDS) und anderen generiert werden.

SIEM-Systeme aggregieren und korrelieren diese Signale, um umfassende Einblicke in die Sicherheitslage des Netzwerks zu erhalten und potenzielle Bedrohungen zu identifizieren. Durch die Integration von Shared Signals in das Monitoring kann eine umfassende und ganzheitliche Sicherheitsüberwachung gewährleistet werden, die potenzielle Angriffe frühzeitig erkennt und darauf reagiert.

Aktuell ist die Shared Signals Unterstützung durch SIEM Systeme und Infrastrukturkomponenten noch nicht hinreichend gegeben. [Für die Session-Beendigung wird daher werden Shared-Signals eine anwendungsspezifische, serverseitige Auslösung der Session-Termination über die Plug-In-Schnittstelle des Authorization Servers umgesetzt.](#)

A 29847 -ZETA Guard, Session-Beendigung in Mindestvariante

Die Komponenten des ZETA Guard MÜSSEN eine Session-Beendigung durch den TI 2.0-Dienstleister unterstützen.

Die Session-Beendigung MUSS den Entzug der zur Session gehörenden Refresh Token bewirken. Bereits ausgestellte Access Token KÖNNEN bis auf zum Ablauf ihrer Gültigkeit weiteres noch nicht verwendet werden.

Hinweis: In dieser Umsetzung wirkt die Session-Beendigung unmittelbar auf die Nutzung von ZETA-unterstützt-Refresh Token. Die maximale Restwirkdauer entspricht der Gültigkeitsdauer bereits ausgestellter Access Token.

[<=]

4.7.3 Telemetrie, Monitoring und Logging

Betriebliche Daten zum Zwecke des Monitorings (Telemetrie) werden von den ZETA Guard-Komponenten erhoben und für die eingesetzten ZETA Guard-Komponenten übergreifend mittels dem Telemetriedaten-Service erfasst. Bei der Nachbereitung der Telemetriedaten werden personenbezogene oder -beziehbare Daten anonymisiert, um diese bereinigten Daten dem Anbieter regelhaft zugänglich zu machen. Das bereinigte Monitoring-Log kann von dem Anbieter für sein eigenes betriebliches Monitoring und als Quelle für sein SIEM-System verwendet werden. Das bereinigte Monitoring-Log wird unter anderem zur Generierung von Traces und Metriken für die gematik benutzt.

4.8 Zusammenspiel mit Identity Provider

Das Stichwort "Step-up-Authentifizierung" bezieht sich auf eine Sicherheitsmaßnahme, bei der der Nutzer zusätzliche Authentifizierungsschritte durchlaufen muss, um auf sensible Ressourcen zuzugreifen. Diese Maßnahme wird wie folgt realisiert:

1. **Nutzer stellt über den ZETA Client eine Anfrage:** Der Nutzer versucht auf eine Ressource zuzugreifen, für die das aktuelle Access Token nicht ausreicht.
2. **PEP fängt Anfrage ab:** Der PEP empfängt die Anfrage.
3. **PEP validiert Access Token:** Der PEP prüft:
 - Ist das Access Token gültig (Signatur, Ablaufdatum etc.)?
 - Hat das Access Token den **erforderlichen Scope und die passende Audience-Resource-Identifier (aud)** für die angeforderte Ressource?
4. **Zugriff verweigert:** Wenn der erforderliche Scope oder die **Audienceer passende Resource-Identifier nicht** im Access Token enthalten ist, verweigert der PEP den Zugriff.
5. **PEP antwortet mit Step-up-Anforderung:** Wenn die angefragte Ressource auf dem Resource Server existiert, informiert der PEP den Nutzer, dass für den Zugriff ein Access Token mit einem bestimmten Scope und eine **fm bestimmten AudienceResource-Identifier** benötigt wird.
6. **Nutzer initiiert die Step-up-Authentifizierung:** Der ZETA Client kommuniziert mit dem Authorization Server, um die Authentifizierung für den benötigten Scope und die **Audienceen benötigten Resource-Identifier** zu beginnen.
7. **Authorization Server authentifiziert und gewährt neues Access Token:** Der Authorization Server:
 - Steuert die Durchführung der Step-up-Authentifizierung.

- Erstellt ein **neues Access Token** mit passendem **Scope** und passendem **Audience-Resource-Identifizier (aud)**.
 - Gibt das neue Access Token an den ZETA Client zurück.
8. **Nutzer stellt eine erneute Anfrage:** Der Nutzer sendet die Anfrage mit dem **neuen** Access Token an den PEP.
9. **PEP prüft das neue Token:** Der PEP prüft wieder:
- Ist das **neue** Access Token gültig?
 - Hat das **neue** Access Token den **Scope** und die **Audience-den Resource-Identifizier (aud)** für die angeforderte Ressource?
10. **Zugriff gewährt:** Wenn der erforderliche Scope und die **Audienceer passende Resource-Identifizier** im **neuen** Access Token enthalten ist, gewährt der PEP den Zugriff auf die Ressource.

Die Step-up-Authentifizierung stellt sicher, dass zusätzliche Sicherheitsmaßnahmen - wenn erforderlich - ergriffen werden, um die Integrität und Vertraulichkeit der geschützten Daten zu gewährleisten.

4.9 TI 2.0-Dienst-Backend

Das TI 2.0-Dienst-Backend (im folgenden Resource Server genannt) stellt das Ziel jedes Zugriffswunschs eines Nutzers über sein Clientsystem dar. Es stellt fachliche Schnittstellen zur Nutzung durch Clientsysteme dar, die über die Mechanismen des Zero Trust abgesichert werden.

5 Spezifikation

Dieses Kapitel beschreibt die technische Umsetzung der beschriebenen Konzepte an die oben eingeführten Komponenten des Zero Trust (ZETA Guard-Komponenten) als generische Produkt- und Anbietertypen. Diese Anforderungen finden Anwendung in den Steckbriefen von konkreten Produkt- und Anbietertypen der jeweiligen Fachanwendung und erhalten erst in der dortigen Zuordnung ein konkretes Prüfverfahren.

Die Festlegungen zu Schemas, OpenAPI Schnittstellen und Abbildungen sind in [gemAPI_ZETA] enthalten.

5.1 Anforderungen an die Sicherheit und den Datenschutz

5.2 Übergreifende Anforderungen für Datenschutz und Sicherheit

A_25400 -ZETA Guard - Umsetzung Sicherer Softwareentwicklungsprozess

Der Hersteller des ZETA Guards MUSS einen sicheren Softwareentwicklungsprozess umsetzen (siehe [gemSpec_DS_Hersteller#Kapitel 2.2 Sicherer Softwareentwicklungsprozess]).[<=]

A_25401 -ZETA Guard - Darstellung der Voraussetzungen für sicheren Betrieb des Produkts im Produkthandbuch

Der Hersteller des ZETA Guards MUSS für sein Produkt im dazugehörigen Produkthandbuch leicht ersichtlich darstellen, welche Voraussetzungen vom Anbieter und der Betriebsumgebung erfüllt werden müssen, damit ein sicherer Betrieb des Produktes gewährleistet werden kann.[<=]

A_28459 -ZETA Guard - Informationsobjekte im Produkthandbuch

Der Hersteller des ZETA Guards MUSS alle vom ZETA Guard verarbeiteten Informationsobjekte in seinem Produkthandbuch vollständig auflisten.[<=]

A_28463 -ZETA Guard - Informationsobjekte des ZETA Clients im Produkthandbuch

Der Hersteller des ZETA Guards MUSS alle vom ZETA Client verarbeiteten Informationsobjekte im Produkthandbuch des ZETA Guard vollständig auflisten.[<=]

A_28460 -ZETA Guard - Datenschutzrechtliche Bewertung durch den Dienstanbieter

Der Anbieter eines TI 2.0 Dienstes MUSS sich über die Informationsobjekte, die im ZETA Guard verarbeitet werden, aus dem Produkthandbuch informieren und als Datenschutzverantwortlicher für sein Dienst bewerten.[<=]

A_28461-01 -Informationspflicht des Client-Herstellers gegenüber Nutzern

Der Hersteller eines TI 2.0-Clients MUSS

- sich über die Informationsobjekte aus dem Produkthandbuch des ZETA Guard informieren
- und seine Nutzer über Verarbeitung der Informationsobjekte datenschutzkonform informieren.

[<=]

Hinweis: Es gibt ein Kapitel in dem ZETA Guard Handbuch, das die Integration von Clientsystemen mit Zeta Guard beschreibt. Das Kapitel beschreibt die Client-Informationenobjekte, die der Client verarbeiten muss.

A_25402 -ZETA Guard - Schutz der transportierten Daten

ZETA Guard MUSS sicherstellen, dass die Vertraulichkeit und Integrität der transportierten Daten gewährleistet ist.

Alle Endpunkte des ZETA Guards MÜSSEN TLS gesichert sein. [≤]

A_28781 -ZETA Guard - Schutz der internen Kommunikation

ZETA Guard MUSS sicherstellen, dass die interne Kommunikation zwischen ZETA Guard Komponenten durch TLS abgesichert ist. [≤]

Hinweis: Es wird empfohlen ein Service Mesh (z. B. Cilium, Istio oder linkerd) einzusetzen.

A_26517-01 -ZETA Guard - Unterstützung von mTLS

ZETA Guard MUSS die Konfiguration und Nutzung von mTLS für die interne Kommunikation und für die Kommunikation zum Resource Server unterstützen. [≤]

A_25403 -ZETA Guard - Schutzmaßnahmen gegen die OWASP Top 10 Risiken

ZETA Guard MUSS geeignete technische Maßnahmen zum Schutz vor den Risiken in der aktuellen Version der [OWASP-Top-10-Risiken] umsetzen. [≤]

Hinweis: Die Anforderungen gelten für die gesamte ZETA-Guard-Lösung. Entscheidet sich ein Anbieter dafür, eine eigene Komponente anstelle einer von ZETA bereitgestellten Komponente (z. B. Ingress) zu verwenden, legt der Sicherheitsgutachter (Anbieter) fest, welche Risiken aus der Top-10-Liste für diese Komponente relevant sind, da deren konkrete Umsetzung anbieterabhängig ist.

A_28961 -Maßnahmen gegen die OWASP Top 10 Kubernetes Risiken

Der Anbieter eines TI 2.0-Dienstes MUSS geeignete Maßnahmen zum Schutz vor den Risiken in der aktuellen Version der [OWASP-Top-Ten-Kubernetes] umsetzen.

[≤]

A_25404 -ZETA Guard - Angriffe erkennen

ZETA Guard MUSS Maßnahmen zur Erkennung, Kategorisierung und Protokollierung bzw. Meldung von Angriffen umsetzen. Die Kategorisierung von Angriffen MUSS nach "CAPEC: OWASP Related Patterns" [CAPEC OWASP] erfolgen. [≤]

A_25405 -ZETA Guard - Angriffen entgegenwirken

ZETA Guard MUSS Maßnahmen zur Schadensreduzierung und -verhinderung von Angriffen umsetzen. [≤]

A_25406 -ZETA Guard - Eingabe Validierung von Operationen

ZETA Guard MUSS sicherstellen, dass alle Daten und Parameter, die über eine API kommuniziert werden, sicherheitstechnisch validiert werden. [≤]

Hinweis: Eine Eingabe-Validierung von Resource Server APIs erfolgt im Resource Server und nicht in den Zero Trust-Komponenten.

A_25407 -ZETA Guard - Sicherheitstechnische Validierung von Policy und Konfigurationen

ZETA Guard MUSS sicherstellen, dass alle Daten und Parameter, die von einer Konfigurationsdatei oder Policy gelesen werden, sicherheitstechnisch validiert werden.

[≤]

A_25408-01 -ZETA Guard - Verbot Profilbildung

Der Anbieter eines TI 2.0-Dienstes DARF Profile - außer zum Zweck des Security Monitorings - NICHT bilden.

[≤]

A_25409 -ZETA Guard - Privacy by Design

ZETA Guard MUSS sicherstellen, dass bei Konfigurationsmöglichkeiten die datenschutzfreundlichere Option vorausgewählt ist. [≤]

A_25410 -ZETA Guard - Verbot von Werbe- und Usability-Tracking

ZETA Guard DARF im Produktivbetrieb ein Werbe- und Usability-Tracking NICHT verwenden. [≤]

A_25411 -ZETA Guard - Verbot vom dynamischen Inhalt

ZETA Guard DARF dynamischen Inhalt von Drittanbietern NICHT herunterladen und verwenden. [≤]

A_25412 -ZETA Guard - Zusätzliche Verschlüsselung bei der Persistierung

Unabhängig davon, ob die Daten schon verschlüsselt vorliegen, MUSS ZETA Guard seine Daten bei der Persistierung verschlüsseln. [≤]

A_25413-01 -ZETA Guard - Ordnungsgemäße IT-Administration

Der Anbieter eines TI 2.0-Dienstes MUSS die Maßnahmen für erhöhten Schutzbedarf aus dem BSI-Bausteins „OPS.1.1.2 Ordnungsgemäße IT-Administration“ [BSI-Grundschutz] während des gesamten Betriebs des ZETA Guards umsetzen. [≤]

A_26479-02 -ZETA Guard - Ordnungsgemäße Änderung von Konfigurationen

Der Anbieter eines TI 2.0-Dienstes MUSS durch technische und organisatorische Mittel sicherstellen, dass eine Änderung der Konfiguration des ZETA Guards nur unter 4-Augen erfolgen kann. [≤]

A_25718 -ZETA Guard - Bereitstellung Security-KPIs

ZETA Guard MUSS sicherstellen, dass die Security-KPIs in A_25484-* automatisch bereitgestellt werden.

[≤]

Hinweis: Die Anforderung ist besonders wichtig, falls die Zero Trust-Komponente in einer VAU betrieben wird.

A_28406-01 -ZETA Guard - Verification der ZETA Guard Images

Der Hersteller des TI 2.0-Dienstes MUSS vor der Aktualisierung von ZETA Guard die Authentizität und Aktualität der zu aktualisierenden Komponenten auf der Grundlage einer von der gematik vorgegebenen Signaturprüfung verifizieren und bei Fehlschlägen der Verifikation die Aktualisierung abbrechen und gematik umgehend informieren. [≤]

Hinweis: Die Images für ZETA Guard werden mit einem Zertifikat der Komponenten PKI signiert. Das Zertifikat und die zugehörige Kette müssen für die Signaturprüfung verwendet werden.

A_28407 -ZETA Guard - Nachweisbarkeit verwendete Version des ZETA-Images

Der Hersteller eines TI 2.0-Dienstes MUSS ein SBOM für sein Produkt erstellen, aus dem eindeutig hervorgeht, welches ursprüngliche ZETA Guard-Image verwendet wurde. [≤]

Hinweis: Das SBOM für das gesamte Produkt (inkl. ZETA Guard) kann durch den Hersteller über mehrere Dateien abgebildet werden. Für ZETA Guard wird immer ein eigenständiges SBOM geliefert.

ZETA-Guard implementiert für Stufe 1 eine vereinfachte Prüfung des Clients auf gesperrte IP-Adressen und Impossible Travel. Diese Vereinfachung ist möglich, da in Stufe 1 nur eine Rolle (LEI) mit einem stationären Endgerät vorgesehen ist.

A_28827 -ZETA-Guard - IP-Adresse Binding des Access-Token

ZETA Guard MUSS für Stufe 1 die anfragende IP-Adresse des Clients im Access-Token bei der Ausstellung des Access-Token hinterlegen. [≤]

A_28803 -ZETA Guard - Prüfung von Impossible Travel

ZETA Guard MUSS für Stufe 1 die im Access-Token hinterlegte IP-Adresse bei jedem Client-Request gegen die tatsächliche IP-Adresse des anfragenden Clients validieren. Bei

einem negativen Ergebnis MUSS ZETA Guard das Access-Token und Refresh-Token des Clients sperren und die aktuelle fachliche Operation abbrechen. [≤]

~~88121A_288289872 -ZETA Guard - WeiterleituIP-Binding Client-IP~~

~~Der Anbieter des TI 2.0-Dienstes MUSS alle in seiner Hoheit betriebenen-Netzwerkkomponenten (wie z. B. Load Balancer, Reverse Proxy, DDoS-Schutz, WAF, CDN- oder API-Gateway) zwischen Client und~~**bei IPv6**

~~Bei der Validierung gemäß A_28827 und A_28803 MUSS ZETA Guard so konfigurieren, dass die originale IP IPv6-Adresse des Clients über die gesamte Verarbeitungskette hinweg mittels standardisierter HTTP-Header (vorzugsweise Forwarded gemäß RFC 7239, alternativ X-Forwarded-For oder X-Real-IP) an ZETA Guard weitergeleitet wird.~~ [≤]

~~5.3 Schlüssel-Management und Verwaltung berücksichtigen. Der Anbieter MUSS ein ZETA~~

~~Teil der Sicherheitsarchitektur von ZETA ist ein robustes Schlüsselmanagement. Um Entwicklern, Betreibern und Auditoren ein klares Verständnis der kryptografischen Architektur zu vermitteln, Verfahren festlegen und dokumentieren, verwendet diese Spezifikation standardisierte~~**Schlüssel-IDs** (z.B. PrK.AuthS.Sig für den privaten Signaturschlüssel des Authorization Servers).

~~5.3.1 Nomenklatur für Schlüssel-IDs~~

~~Jeder Schlüssel erhält eine eindeutige ID nach folgendem Schema:~~ **[Typ].[Komponente].[Zweck]**

~~Typ:~~

- ~~• PrK: Privater Schlüssel (Private Key)~~
- ~~• PuK: Öffentlicher Schlüssel (Public Key)~~
- ~~• C: Zertifikat (Public Key inkl. Identitätsbindung, Certificate)~~
- ~~• SymK: Symmetrischer Schlüssel (Symmetric Key)~~

~~Komponente oder Schlüssel:~~

- ~~• AuthS: Authorization Server (PDP)~~
- ~~• PEP: Policy Enforcement Point (HTTP Proxy)~~
- ~~• DB: Datenbank (PDP-DB)~~
- ~~• Client: ZETA-Client (App / Primärsystem)~~
- ~~• K8s: Kubernetes / Infrastruktur~~
- ~~• Sys: Gematik / Zentrales System~~
- ~~• AK: Attestation Keys~~
- ~~• EK: TPM Endorsement Keys~~
- ~~• DPoP: DPoP Keys~~
- ~~• SM(C)-B: SM(C)-B Keys~~
- ~~• TI-RootCA: TI-Root-CA Signer Keys~~
- ~~• TI-TSL: TI-TSL Signer Keys~~
- ~~• TI-KompCA: TI-Komponenten-PKI-CA Keys~~

- TI FedMaster: Federation Master Signer Keys
nde IPv6-Adresspräfixe minimiert. [≤]
- **A_28828** -ZETA-IK: ZETA-OCI-Container-Image-Signer-Keys
- ZETA-DK: ZETA-OCI-Data-Image-Signer-Keys
- ZETA-PAK: ZETA-OPA-Bundle-Autor-Signer-Keys
- ZETA-PFK: ZETA-OPA-Bundle-Freigeber-Signer-Keys

Zweck:

- TLS: Verschlüsselung des externen Traffics
- mTLS: Interne Komponenten Authentifizierung
- ASL: Application Security Layer
- Sig: Signatur (Token, Policies, Entity Statements)
- Enc: Payload-/Daten-Verschlüsselung (JWE, DB At Rest)

Übersicht: **Guard - Weiterleitung Client-IP**

5.3.2 der ZETA Guard Schlüssel (Anbieter-Seite)

Diese Tabelle fasst alle Schlüssel zusammen, die vom Betreiber des TI 2.0-Dienstes (ZETA-Guard) verwaltet werden müssen. Bei Verarbeitung von Daten mit Schutzbedarf "sehr hoch" greifen strenge VAU- und HSM-Pflichten.

Hinweis: Schlüssel mit dem Vermerk "Die Schlüsselverwaltung muss dokumentiert werden", werden vom Anbieter des TI 2.0-Dienstes verwaltet. Die Schlüssel ohne Vermerk verwaltet ZETA-Guard automatisch.

MUSS alle in seiner Hoheit Betriebsmodi und Schlüsselschutz:

Der ZETA-Guard unterscheidet enen Netzwei primäre Betriebsmodi:

1. **Regulärer Betrieb (ohne VAU):** Schlüssel werden in sicheren Software-Keystores (z.rkkomponenten (wie z. B. KMS v2) der jeweiligen Container-Umgebung abgelegt.
2. **Betrieb mit Schutzbedarf "sehr hoch" (mit VAU):** SoLoad bald sensible Daten verarbeitet werden, muss der ZETA-Guard in einer Vertrauenswürdigem Ausführungsumgebung (VAU) betrieben werden (siehe A_25608-01). In diesem Fall **MÜSSEN** asymmetrische private Schlüssel der TI- und Internet-Identitäten zwingend in einem Hardware Security Module (HSM) verbleiben. Schlüssel, die in den Arbeitsspeicher der VAU geladen werden müssen (z.B. Token-Signatur- oder Datenbank-ancer, Reverse Proxy, DDoS-Schutz, WAF, CDN oder API Gateway) zwiSchlüssel), müssen durch HSM-generierte Key-Encryption-Keys (KEK) "gewrappt" (Sealing) übergeben werden.

Tabelle 1: Tab-en Client und ZETA-Guard-Keys

Schlüssel-ID	Bezeichnung & Zweck	Speicherung / Betrieb OHNE VAU	Speicherung / Betrieb MIT VAU (Schutzbedarf "sehr hoch")
PrK.AuthS.Sig PuK.AuthS.Sig	AuthS-Token-Signaturschlüssel Zur Signatur von Access- und Refresh-Token sowie	PrK: Sicherer-Software-Keystore des AuthS:	PrK: Muss durch HSM-geschützt sein. (Darf in den AuthS geladen werden, wenn z.B.

	des Entity Statements. Der PuK wird im JWKS-bereitgestellt. Die Schlüsselverwaltung muss dokumentiert werden.		mittels KEK aus dem HSM gesichert).
PrK.AuthS.TLS C.AuthS.TLS	AuthS Internet-Identität (TLS) Terminierung der externen TLS-Verbindung am Authorization Server. Die Schlüsselverwaltung muss dokumentiert werden.	Sicherer Software-Keystore.	PrK: Verbleibt zwingend im HSM (via HSM-Proxy), falls kein ASL-Tunnel zum AuthS existiert.
SymK.DB.Enc	Datenbank-Verschlüsselung Verschlüsselung der Session-, Nutzer- und Client-Daten At Rest im Authorization Server.	Sicherer Software-Keystore (z.B. K8s-Secrets).	Muss durch HSM geschützt sein. Übergabe an AuthS darf nur an attestierte Instanzen erfolgen.
PrK.PEP.TLS C.PEP.TLS	PEP Internet-Identität (TLS) Terminierung der externen TLS-Verbindung am HTTP-Proxy. Die Schlüsselverwaltung muss dokumentiert werden.	Sicherer Software-Keystore.	PrK: Verbleibt zwingend im HSM, falls kein ASL-Tunnel zum HTTP-Proxy existiert.
PrK.PEP.Sig C.PEP.Sig	PEP Signatur-Identität Zertifikat aus der Komponenten-PKI. Dient als Identität des PEP (für Signatur des ASL-Master-Schlüssels). Die Schlüsselverwaltung muss dokumentiert werden.	Sicherer Software-Keystore.	Privater Schlüssel verbleibt zwingend im HSM (Zugriff via HSM-Proxy).
PrK.PEP.ASL PuK.PEP.ASL	PEP ASL-Identität Semi-statische Schlüsselpaare für den ZETA/ASL-Kanal (maximal 1-Monat gültig).	Im RAM des PEP.	Wird im PEP generiert, aber vom PrK.PEP.Sig im HSM beglaubigt.
SymK.PEP.ASL	PEP ASL Session-Key Abgeleiteter kurzlebiger Schlüssel für den eigentlichen ASL-Traffic.	Im RAM des PEP.	Wird im PEP generiert.
PrK.ZG.IDP PuK.ZG.IDP	IDP für die ZETA-Guard-Workload Identity Zur Signatur von Subject-Token für die Kommunikation mit gematik-Diensten (SIEM, Telemetrie) und für die Dienst-zu-Dienst-	Innerhalb der Kubernetes-Infrastruktur des Anbieters.	Wenn langlebiger Schlüssel, dann wird der private Schlüssel im HSM gespeichert.

	<p>Kommunikation: Initial: Kubernetes IDP oder langlebiger Schlüssel Zukünftig: AuthS als IDP des ZETA Guard mit PrK.AuthS.Sig und PuK.AuthS.Sig Die Schlüsselverwaltung muss dokumentiert werden.</p>		
PrK.Ingress.TLS C.Ingress.TLS	<p>Ingress-TLS TLS-Terminierung am vorgeschalteten Ingress (falls genutzt). Die Schlüsselverwaltung muss dokumentiert werden.</p>	Sicherer Software-Keystore.	Verbleibt zwingend im HSM, falls kein ASL-Tunnel zum AuthS und zum HTTP-Proxy existiert.
PrK.K8s.mTLS C.K8s.mTLS	<p>Interne Mesh-Identitäten Zur Absicherung der microservice-internen Kommunikation (z. B. AuthS <-> PE).</p>	K8s-Service-Mesh (z.B. Istio).	Verwaltung innerhalb der VAU (Zugriff strikt limitiert).
PuK.Client.Sig	<p>Öffentlicher Client Instance-Key Wird bei der initialen Client-Registrierung an den AuthS gesendet. Dient der Validierung der "Client-Assertion" bei zukünftigen Logins.</p>	Gespeichert in der PDP-Datenbank.	Gespeichert in der PDP-Datenbank. Muss vor unautorisierter Manipulation (Austausch durch Angreifer) geschützt sein (Integrität).

Mainline_OPB1/ML-188082A_28826 -HSM Proxy, Übergabe PDP-Datenbank-Schlüssel an ZETA Guard Authorization Server

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Übergabe der PDP-Datenbank-Schlüssel (SymK.DB.Enc) ausschließlich an einen attestierten und freigegebenen ZETA Guard Auth_ orization-Server erfolgt. [<=]

Mainline_OPB1/ML-188732A_28863 -HSM Proxy, Verwendung AuthS-Schlüssel nur durch ZETA Guard Authorization Server

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Verwendung der AuthS-Schlüssel (PrK.AuthS.Sig, PrK.AuthS.TLS) im HSM ausschließlich durch einen attestierten und freigegebenen ZETA Guard Authorization-Server möglich ist. [<=]

Mainline_OPB1/ML-188733A_28864 -HSM Proxy, ginale IP-Adresse des Clients über die gesamte Verwendung PEP-Schlüssel nur durch ZETA Guard HTTP Proxy

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Verwendung der PEP-Schlüssel (PrK.PEP.TLS, PrK.PEP.Sig) im HSM ausschließlich durch einen arbeitungskette hinweg mittestierten und freigegebenen ZETA Guard HTTP-Proxy möglich ist. [<=]

Mainline_OPB1/ML-188734A_28865 -HSM Proxy, Verwendung Ingress-Schlüssel nur durch ZETA Guard Ingress

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS dies standardisierter HSM-Proxy sicherstellen, dass die Verwendung der Ingress-Schlüssel (PrK.Ingress.TLS) im HSM ausschließlich durch einen attestierten und freigegebenen ZETA Guard Ingress möglich ist. [<=]

5.3.3 ZETA Client Schlüssel (Nutzer-Seite)

Diese Schlüssel verbleiben in der Verfügungsgewalt TTP-Header (vorzugsweise Forwarded gemäß RFC 7239, alt des Endnutzers (Smartphone / Primärsystem) bzw. der Institution.

Tabelle 2: ernaTab-ZETA-Client-Keys

Schlüssel-ID	Bezeichnung & Zweck	Speicherung / Verwendung
PrK.Client.Sig PuK.Client.Sig	Client Instance Key Pair Langlebiges ECC-Schlüsselpaar zur eindeutigen Identifikation der Client-Installation. Dient der Signatur der Client-Assertion.	PrK: Zwingend in Hardware (TPM, Secure Enclave, TEE) generiert und gespeichert. Erhält strikt das Attribut sign (kein Export möglich). PuK: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen.
PrK.DPoP.Sig PuK.DPoP.Sig	DPoP-Schlüsselpaar Kurzlebiger (Session-basierter) Schlüssel zum Signieren von Anfragen als Beweis für den Besitz des Access Token (Proof of Possession).	PrK: Wird für die Dauer der Session sicher lokal gehalten (flüchtig im RAM oder durch native OS-Sicherheitsmechanismen / TPM-Wrapping geschützt). Eine Verschlüsselung des DPoP-Keys durch den PrK.Client.Sig ist aus Gründen der Schlüsseltrennung nicht zulässig. PuK: Wird im HTTP-Header gesendet.
PrK.AK.Sig PuK.AK.Sig	Plattform Attestation Key Zur Signatur des Client-Zustands.	PrK: Wird zur Signatur des Client-Zustands verwendet. Hardwaregebunden (TPM / Secure Enclave). PuK: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen und zur Prüfung der Systemintegrität gesendet (inkl. Zertifikatskette zur Validierung gegen Root-CA des Herstellers).
PrK.EK.Sig PuK.EK.Sig C.EK.Sig	TPM 2.0 Endorsement Keys Wird bei der TPM Attestation verwendet.	PrK: Hardwaregebunden (TPM / Secure Enclave). PuK: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen und zur Bindung des AK an den EK verwendet. C: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen und zur Bindung des AK an den EK verwendet.
PrK.SM(C)-	SMC-B Institutionsidentität	PrK: Verbleibt hardwaregebunden

B.Sig C.SM(C)-B.Sig	Ausgestellt von der SMC-B-CA. Zur Signatur des subject_token beim Token-Exchange, um die Identität der Institution (z.B. Praxis) nachzuweisen. Das Subject Token enthält ein Binding des PuK.Client.Sig und des PuK.DPoP.Sig.	auf der Smartcard (SMC-B) oder im HSM-B. C: Wird übermittelt und durch AuthS gegen TSL validiert.
------------------------	---	--

5.3.4 gematik veiv X-Forwaltete Schlüssel (TI)

Diese Zertifikate und Schlüssel spannen den Vertrauensraum der TI 2.0 auf. Die privaten Schlüssel liegen hochsicher bei rded-For oder gematik (bzw. den zugelassenen Trust-Centern). Die X-Real-IP an ZETA Guards und Clients nutzen die öffentlichen Teile/Zertifikate zur Validierung. Diese Schlüssel dienen der Sicherstellung der Supply-Chain-Security und der Integrität von Regelwerken.

Tabelle 3: Tab-Gematik-Keys weitergeleitet wird.[<=]

Schlüssel-ID	Bezeichnung & Zweck	Speicherung / Verteilung
PrK.TI-RootCA.Sig C.TI-RootCA.Sig	TI-Root-CA Oberster Vertrauensanker der TI. Alle Zertifikate im System leiten sich hiervon ab.	C: Lokal in Truststores hinterlegt. Wird mit dem ZETA-Guard-Provisioning OCI Image in den ZETA-Guard geladen. PrK: Offline/Hochsicher bei der gematik.
PrK.TI-TLSig C.TI-TLS.Sig	TSL-Signer Zertifikat zur Validierung der Signatur der Trust-Service-Status-List (TSL), welche die aktuell gültigen CAs definiert.	C: Im ZETA-Guard (über lokales Artifact-Registry) zur TSL-Verifikation. PrK: Bei der gematik.
PrK.TI-KompCA.Sig C.TI-KompCA.Sig	Komponenten-PKI-CA Sub-CA der Root-CA. Stellt die Zertifikate für die TI-Dienste (PEP, AuthS) aus.	C: Über die TSL als vertrauenswürdig verteilt.
PrK.TI-SMCB-CA.Sig C.TI-SMCB-CA.Sig	SMC-B-CA Sub-CA der Root-CA. Stellt die Institutionszertifikate für Leistungserbringer aus.	C: Über die TSL als vertrauenswürdig verteilt (zur Prüfung im ZETA-Guard).
PrK.TI-FedMaster.Sig PuK.TI-FedMaster.Sig	Federation-Master-Signer Zur Signatur der Entity-Statements (Trustmarks) in der OIDC-Föderation.	PuK: Im ZETA-Guard hinterlegt, um die Föderations-Zugehörigkeit anderer AuthS zu prüfen. PrK: Bei der gematik.
PrK.ZETA-IK.Sig C.TI-ZETA-IK.Sig	Signaturzertifikat Ausführbare-ZETA-Images Signatur der ausführbaren ZETA-OCI-Container (PEP, AuthS, OPA,	C: Im Admission-Controller validiert (Signaturkette bis zur Root-CA). PrK: In gematik CI/CD-Pipeline.

	etc.-).	
PrK.ZETA-DK.Sig C.TI-ZETA-DK.Sig	Signaturzertifikat-Daten-Images Signatur von Datencontainern- (z.B. Konfigurationsdaten,- Provisioning-Daten).	C: Im Admission Controller / durch ZETA Guard validiert. PrK: In gematik CI/CD-Pipeline.
PrK.ZETA-PAK.Sig C.TI-ZETA-PAK.Sig	OPA Policy-Autor Signatur Signaturzertifikat des Policy- Erstellers für OPA-Bundles.	C: Im ZETA Guard (OPA Engine) zur Signaturprüfung konfiguriert. PrK: Beim Datenbearbeiter.
PrK.ZETA-PFK.Sig C.TI-ZETA-PFK.Sig	OPA Policy-Freigeber Signatur Zweitsignatur (4-Augen-Prinzip)- für OPA-Bundles.	C: Im ZETA Guard (OPA Engine)- konfiguriert. PrK: Beim Freigeber.
PrK.K8s.IDP C.K8s.IDP	Kubernetes-IDP (Workload- Identity) Zur Signatur von Subject Token- für die Kommunikation mit- gematik-Diensten (SIEM,- Telemetrie).	PrK: Innerhalb der Kubernetes- Infrastruktur des Anbieters. C: Innerhalb des gematik-Workload- Identity Pools

5.4 Sicherheits- und Datenschutzanforderungen an Logging und Monitoring

Hinweis: Die Anforderungen dieses Abschnitts könnten sich noch ändern, falls sich bei der Umsetzung des Zero Trust herausstellt, dass weitere Protokollierungen auf Seiten des Anbieters notwendig werden.

A_25744 -ZETA Guard - Datenschutzkonformes Logging und Monitoring

ZETA Guard MUSS die für den Betrieb des Zero Trust erforderlichen Logging- und Monitoring-Informationen in solcher Art und Weise erheben und verarbeiten, dass mit technischen Mitteln ausgeschlossen ist, dass dem Anbieter eines TI 2.0-Dienstes vertrauliche oder zur Profilbildung geeignete Daten zur Kenntnis gelangen. [<=]

Hinweis: Der Telemetriedaten Service im ZETA Guard muss die vom PEP und PDP gesammelten Telemetriedaten so verändert an das Monitoring System des Anbieters weitergeben, dass eine Profilbildung nicht mehr möglich ist.

A_25745 -ZETA Guard - Keine medizinischen Informationen in Logging und Monitoring

ZETA Guard MUSS sicherstellen, dass in für den Betrieb erstellten Protokollen keine personenbezogenen medizinischen Informationen enthalten sind (u. a. medizinische Daten von Versicherten oder Informationen, aus denen sich ableiten lässt, bei welchen Leistungserbringerinstitutionen ein Versicherter in Behandlung ist). [<=]

A_25746 -ZETA Guard - Keine sicherheitsrelevanten Daten in Logging und Monitoring

ZETA Guard MUSS sicherstellen, dass in für den Betrieb erstellten Protokollen keine sicherheitsrelevanten Daten enthalten sind. [<=]

Hinweis: Sicherheitsrelevante Daten sind zum Beispiel, Kryptoschlüssel, Access/Refresh Token usw.

A_25747-01 -ZETA Guard - Löschfristen Protokolle

Der Anbieter eines TI2.0 Dienstes MUSS sicherstellen, dass die zum Zwecke der Fehleranalyse erhobenen Protokolle des ZETA Guards nach Behebung des Fehlers unverzüglich gelöscht werden.

[<=]

5.5 Sicherheits- und Datenschutz-Anforderungen an das Security Monitoring

Um eine lückenlose Nachvollziehbarkeit komplexer Systeminteraktionen zu gewährleisten, ist die nahtlose Verknüpfung aller anfallenden Telemetriedaten entscheidend. Jede einzelne Anfrage löst eine Vielzahl von Prozessen, Logs und Metriken aus, die erst durch eine konsistente Korrelation ihren vollen diagnostischen Wert entfalten. Ziel ist es, die gesamte Verarbeitungskette eines Requests über alle ZETA-Komponenten hinweg als zusammenhängendes Ereignis sichtbar zu machen, sodass die kausalen Zusammenhänge zwischen den verschiedenen Datenpunkten jederzeit transparent und ohne Informationsverlust nachverfolgbar bleiben.

A_25484-03 -Security Monitoring - Security KPIs

ZETA Guard MUSS einmal täglich mittels dem Telemetriedaten-Service die folgende Sicherheits-KPIs automatisiert über die von der gematik angebotene Schnittstelle an das TI SIEM-System als OTel-Metric übermitteln:

- Anzahl versuchter Zugriffe von nicht registrierten Clients (hier muss die KPIs zwischen Resource Server APIs und Client-Registrierung APIs unterscheiden)
- Anzahl von Zugriffen von Botnetzen
- Anzahl von Zugriffen aus jedem Land gezählt plus weitere Zugriffe, die separat in Versicherte und LE ausgewiesen werden
- Anzahl fehlerhafter Clientfreischaltungen plus weitere breakdown in Versicherte und LE
- Anzahl von Impossible travel Zugriffen (inkl. Land- und Ortsdaten) plus weitere breakdown in Versicherte und LE
- Anzahl von Zugriffen über TOR Netzwerke plus weitere breakdown in Versicherte und LE
- Anzahl von Zugriffen über VPNs plus weitere breakdown in Versicherte und LE
- Anzahl erkannte Angriffe in Kategorie (siehe A_25404-*) plus weitere breakdown in Versicherte und LE
- Anzahl fehlerhafte Authorization Codes vom IDP.

[<=]

Hinweis: Security KPIs beinhalten anonyme Daten und sind nicht auf individuelle Nutzer zurückzuführen.

Hinweis: Impossible Travel ist eine Methode zur Anomalieerkennung in der Cybersicherheit, die potenzielle Kompromittierungen identifiziert, indem sie Nutzeranmeldeaktivitäten analysiert und mit geografischen Standorten korreliert. Dabei werden Fälle markiert, in denen auf das Nutzerkonto innerhalb eines verdächtig kurzen Zeitraums aus zwei verschiedenen Ländern zugegriffen wird.

*Hinweis: Falls der Anbieter aufgrund einer Netzwerk-Sicherheitsrichtlinie Anfragen mit bestimmten Kommunikationsmerkmalen (z. B. Zugriffe aus einem **W**TOR-Netzwerk oder Botnetz) am Netzwerk-Perimeter blockiert, hat diese Richtlinie Vorrang. Es wird nicht erwartet, dass entsprechende Requests dennoch an den ZETA Guard weitergeleitet, da diese Anfragen bereits durch den Schutzmechanismus am Netzwerk-Perimeter abgefangen werden.*

Hinweis: Forbidden Countries beinhaltet eine Liste von IP-Ranges der Länder-ASN.

Hier ein Beispiel Metric für die Anzahl von Zugriffen von Botnetzen:

```
{
  "resourceMetrics": [
    {
      "resource": {
        "attributes": [
          {
            "key": "service.name",
            "value": {
              "stringValue": "zeta-guard"
            }
          },
          {
            "key": "service.version",
            "value": {
              "stringValue": "1.0.0"
            }
          }
        ]
      },
      "scopeMetrics": [
        {
          "scope": {
            "name": "zeta-guard-kpis",
            "version": "1.0.0"
          },
          "metrics": [
            {
              "name": "zeta_guard_kpi_botnet_accesses_24h",
              "description": "Number of botnet accesses detected by ZETA Guard in the last 24 hours",
              "unit": "1",
              "sum": {
                "isMonotonic": true,
                "aggregationTemporality": "AGGREGATION_TEMPORALITY_DELTA",
                "dataPoints": [
                  {
                    "attributes": [
                      {
                        "key": "date",
                        "value": {
                          "stringValue": "2026-03-09"
                        }
                      }
                    ]
                  }
                ],
                "startTimeUnixNano": "1762128000000000000",
                "timeUnixNano": "1762214400000000000",
                "asInt": "123"
              }
            }
          ]
        }
      ]
    }
  ]
}
```


		existierende Routen zu erreichen (z.B. für Brute-Force-Angriffe oder Enumeration).
http_fqdn	https://opentelemetry.io/docs/specs/semconv/registry/attributes/server/#server-address	Identifikation des Zielservers bei Multi-Host-Umgebungen und Erkennung von Anfragen an nicht autorisierte oder verdächtige Domänen.
http_status	https://opentelemetry.io/docs/specs/semconv/registry/attributes/http/#http-response-status-code	Erkennung von Fehlern, Ausfällen oder potenziellen Angriffsversuchen (z.B. viele 401/403-Fehler bei Brute-Force-Angriffen, viele 5xx-Fehler bei Denial-of-Service-Angriffen).
client_id	https://opentelemetry.io/docs/specs/semconv/registry/attributes/app/#app-installation-id	Identifikationsmerkmal des anfragenden Clients

[<=]

A_28793 -Telemetriedaten für Policy Entscheidungen

Der PDP MUSS über den Telemetriedaten-Service die folgende Daten zu jeder Policy-Entscheidung automatisiert über die von der gematik bereitgestellte Schnittstelle an das TI-SIEM-System als Teil eines OTel-Traces übermitteln.

Tabelle 5: Telemetriedaten Policy Entscheidungen

Daten	Open Telemetry Attribute	Begründung und Zweck
produkt_id	nicht Verfügbar	Möglichkeit abweichende und verdächtige Produktversionen zu erkennen

produkt_version	https://opentelemetry.io/docs/specs/semconv/registry/attributes/app/#app-build-id	Möglichkeit abweichende und verdächtige Produktversionen zu erkennen
os	https://opentelemetry.io/docs/specs/semconv/registry/attributes/os/#os-name	Möglichkeit abweichende und verdächtige Betriebssystemversionen zu erkennen
os_version	https://opentelemetry.io/docs/specs/semconv/registry/attributes/os/#os-version	Möglichkeit abweichende und verdächtige Betriebssystemversionen zu erkennen
device_integrity	nicht Verfügbar	Indikator ob das Gerät gerootet/kompromittiert ist
<p>Optional: Gerätmodel für Android und iOS Clients</p>		
device_model		Möglichkeit abweichende und verdächtige Geräte zu erkennen.
<p>Optional: Falls die Policyentscheidung fehlgeschlagen hat</p>		
Ergebnis der Auswertung der Policy als getrennte OTEL Log	nicht Verfügbar	Verständnis über die Ablehnungsgrund von Policies plus Angriffe

		oder Manipulationen zu erkennen.
Optional: Falls ein Simulationspolicy existiert		
Ergebnis der Auswertung der Simulationspolicy als getrennte OTEL Log	nicht Verfügbar	Möglichkeit häufige Violators von Simulation Policy zu erkennen und verstehen.

[<=]

A_28867 -Ergebnis der Policy Entscheidung als OTel-Log

Der PDP MUSS über den Telemetriedaten-Service das Ergebnis einer Policy-Entscheidung automatisiert über die von der gematik bereitgestellte Schnittstelle an das TI-SIEM-System als ein OTel-Log übermitteln.[<=]

Hier ein Beispiel eines OTel-Logs für die Policyentscheidung:

```
{
  "resourceLogs": [
    {
      "resource": {
        "attributes": [
          {
            "key": "service.name",
            "value": {
              "stringValue": "openpolicyagent"
            }
          },
          {
            "key": "service.version",
            "value": {
              "stringValue": "1.14.0"
            }
          },
          {
            "key": "opa.instance.id",
            "value": {
              "stringValue": "4794056b-0001-48e2-8acf-3a6fb0287384"
            }
          },
          {
            "key": "opa.bundle.authz",
            "value": {
              "stringValue": ""
            }
          }
        ]
      }
    }
  ]
}
```

```
]
},
"scopeLogs": [
  {
    "scope": {
      "name": "openpolicyagent.org/decision_logs",
      "version": "1.14.0"
    },
    "logRecords": [
      {
        "timeUnixNano": "1741170585787524800",
        "observedTimeUnixNano": "1741170585787524800",
        "severityNumber": 9,
        "severityText": "INFO",
        "body": {
          "stringValue": "Decision Log"
        },
        "attributes": [
          {
            "key": "opa.decision_id",
            "value": {
              "stringValue": "d88b7796-8e90-441e-a572-a2f6ea179c18"
            }
          },
          {
            "key": "opa.path",
            "value": {
              "stringValue": "policies/zeta/authz/decision"
            }
          },
          {
            "key": "opa.result.allow",
            "value": {
              "boolValue": false
            }
          },
          {
            "key": "opa.result.reasons",
            "value": {
              "arrayValue": {
                "values": [
                  {
                    "stringValue": "User profession is not allowed"
                  },
                  {
                    "stringValue": "TelematikID is blocked"
                  }
                ]
              }
            }
          }
        ]
      }
    ]
  },
  {
    "traceId": "",
    "spanId": "",
    "flags": 0
  }
]
```

```

    ]
  }
]
}

```

Hinweis: Im "opa.path" Attribut soll der Path der Simulation-Policy oder der Policy eingetragen werden.

A_28795-01 -Telemetriedaten Angriffserkennung

ZETA Guard MUSS über den Telemetriedaten-Service die folgende Daten zu jedem erkannten Angriff (A_25404-*) automatisiert über die von der gematik bereitgestellte Schnittstelle an das TI-SIEM-System als Teil eines OTel-Traces übermitteln

Tabelle 6: Telemetriedaten Angriffserkennung

Daten	Open Telemetry Attribute	Datenschutzbeurteilung
Angriffstyp (OWASP CAPEC Klassifizierung) [CAPEC OWASP]	nicht Verfügbar	Verständnis über die Angriffsklassifizierung
Angriffsdaten	nicht Verfügbar	Verständnis über den aktuellen Angriff

[<=]

A_28796 -ZETA Guard - Korrelation der Telemetrie des Security Monitorings

ZETA Guard MUSS sicherstellen, dass alle während der Anfrageverarbeitung anfallenden Telemetriedaten des Security Monitorings (A_28783-, A_28793- und A_28795-*) mittels einer durchgängigen Korrelations-ID logisch miteinander verknüpft werden.[<=]

Hinweis: Dies gewährleistet eine lückenlose und verlustfreie Nachverfolgung der gesamten Verarbeitungskette eines Requests über alle Systemkomponenten hinweg, analog zur Funktionsweise eines Spans in OpenTelemetry.

A_25485-02 -Security Monitoring - Sicherheitsmeldung bei Aktualisierung von PIP-Daten oder PAP-Policies

ZETA Guard MUSS bei der erfolgreichen Aktualisierung der PIP-Daten und PAP-Policies eine Sicherheitsmeldung automatisiert über die von der gematik angebotene Schnittstelle an das TI SIEM-System übermitteln.[<=]

A_25606-02 -Security Monitoring - Fehlermeldung bei Aktualisierung von PIP-Daten oder PAP-Policies

ZETA Guard MUSS bei folgenden Fehlern während der Aktualisierung der PIP-Daten und PAP-Policies eine Fehlermeldung automatisiert über die von der gematik angebotene Schnittstelle an das TI SIEM-System übermitteln:

- Policy Download Fehler
- Fehler bei der Integritätsprüfung der Policy-Signatur

[<=]

89775A_28960-01 -Security Monitoring - Keine Weitergabe sicherheitsrelevanter Telemetriedaten

ZETA Guard DARF sicherheitsrelevante Telemetriedaten (A_25840-*, A_288783-*, A_28793-*, A_28867-*, A_28795-*) NICHT an den Betreiber weitergeben.[<=]

Hinweis: Betriebliche Telemetriedaten dürfen gemäß A_27260-* an den Betreiber übermittelt werden.

A_28964 -Security Monitoring - Löschung sicherheitsrelevanter Telemetriedaten

Nach erfolgreicher Übermittlung der sicherheitsrelevanten Telemetriedaten an die gematik MUSS ZETA Guard alle relevanten Logs unmittelbar löschen.

[<=]

5.6 Sicherheits- und Datenschutz-Anforderungen an die Protokollierung von Administrationsaktivitäten

Administratoren haben weitreichende Zugriffsrechte in Kubernetes, die es ihnen ermöglichen, die Sicherheitskonfigurationen von TI 2.0-Diensten direkt zu ändern. Ohne Kontrolle und Protokollierung dieser Aktivitäten besteht das Risiko, dass unautorisierte oder fehlerhafte Änderungen vorgenommen werden, die die Sicherheit der TI 2.0-Diensten gefährden.

A_28749 -ZETA Guard - Tamper-Proof Protokollierung von Administrationsaktivitäten

Der Anbieter des TI 2.0-Dienstes MUSS eine revisionssichere Protokollierung (Tamper-Proof) aller TI 2.0-Dienst relevanten administrativen Vorgänge auf dem ZETA Kubernetes Cluster führen.[<=]

Hinweis: Diese Anforderung umfasst alle Administrationsaktivitäten z.B. Anwendung, Plattform und Cluster Administration.

A_28750-01 -ZETA Guard - Löschfristen Auditeinträge des Admin-Audit-Logs

Der Anbieter des TI 2.0-Dienstes MUSS sicherstellen, dass die Löschung eines Admin-Auditeintrags den gesetzlichen Vorgaben entspricht und frühestens nach 6 Monaten erfolgt.[<=]

A_28751 -ZETA Guard - Kontrolle des Admin-Audit-Logs

Der Anbieter des TI 2.0-Dienstes MUSS das Admin-Audit-Log mindestens alle 3 Monate im Vieraugenprinzip kontrollieren. Diese Rollen DÜRFEN NICHT an der Administration des ZETA Clusters teilnehmen. Bei der Kontrolle ist insbesondere auf ungewöhnliche, nicht nachvollziehbare oder maliziöse Administratoraktivitäten zu achten.[<=]

Hinweis: Die in A_28751-* und A_28749-* definierten Anforderungen dürfen durch Automatisierung bzw. unterstützendes Tooling anstelle manueller Kontrollen umgesetzt werden, sofern das Vier-Augen-Prinzip weiterhin gewährleistet ist.

A_28752 -ZETA Guard - Sicherheitsmeldung bei Aktualisierung der Konfiguration des ZETA Guard

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Anbieter des TI 2.0-Dienstes sicherstellen, dass

- jede Konfigurationsänderung am ZETA Guard eine automatisierte Sicherheitsmeldung an das SIEM-System des Anbieters auslöst und
- auf jede dieser Meldungen eine Reaktion gemäß dem vordefinierten Incident-Management-Prozess erfolgt

[<=]

A_28753 -ZETA Guard - Incident-Management-Prozess bei Konfigurationsänderungen von ZETA Guard

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Anbieter des TI 2.0-Dienstes einen Incident-Management-Prozess etablieren, der bei jeder automatisierten SIEM-Meldung über eine Konfigurationsänderung am ZETA Cluster ausgelöst wird. Dieser Prozess MUSS eine unverzügliche Analyse und Bewertung der Änderung sicherstellen, um

festzustellen, ob sie autorisiert war und ein Sicherheitsrisiko darstellt. Bei unautorisierten oder schädlichen Änderungen müssen definierte Korrekturmaßnahmen eingeleitet werden. [≤]

5.7 Sicherheits- und Datenschutz-Anforderungen an die Verarbeitung von Daten mit dem Schutzbedarf "sehr hoch"

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter keine Kenntnis über die in dem ZETA Guard verarbeiteten Daten erlangen darf, gibt es Sonderanforderungen, um die Daten während der Verarbeitung zu schützen.

ZETA Guard muss in einer VAU laufen können. Daher muss der ZETA Guard die Speicherung und Verwendung der Privatschlüssel von Komponenten-Identitäten in einem HSM unterstützen. Für die Kubernetes etcd Verschlüsselung unterstützt ZETA Guard KMS v2.

A_25608-01 -ZETA Guard - Verarbeitung von Daten mit Schutzbedarf "sehr hoch"

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0-Dienstes keine Kenntnis über die in dem ZETA Guard verarbeiteten Daten erlangen darf, MUSS der Anbieter den ZETA Guard in einer VAU umsetzen. [≤]

A_25763-01 -Zero Trust-Komponenten - Private Schlüssel der Komponenten-Identitäten in einem HSM

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0-Dienstes keine Kenntnis über die in dem ZETA Guard verarbeiteten Daten erlangen darf, MUSS der Anbieter die privaten Schlüssel der Identitäten des ZETA Guards in einem HSM speichern. [≤]

A_25764 -Zero Trust-Komponenten - Sicherer Betrieb und Nutzung eines HSMs

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0-Dienstes keine Kenntnis über die in dem ZETA Guard verarbeiteten Daten erlangen darf, MUSS der Anbieter beim Einsatz eines HSMs sicherstellen, dass die auf dem HSM verarbeiteten privaten Schlüssel, Konfigurationen und eingesetzte Software nicht unautorisiert ausgelesen, unautorisiert verändert, unautorisiert ersetzt oder in anderer Weise unautorisiert benutzt werden können. [≤]

A_25765 -Zero Trust-Komponenten - Einsatz zertifizierter HSM

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0-Dienstes keine Kenntnis über die in dem ZETA Guard verarbeiteten Daten erlangen darf, MUSS der Anbieter beim Einsatz eines HSMs sicherstellen, dass dessen Eignung durch eine erfolgreiche Evaluierung nachgewiesen wurde. Als Evaluierungsschemata kommen dabei Common Criteria oder Federal Information Processing Standard (FIPS) in Frage. Die Prüftiefe MUSS mindestens:

1. FIPS 140-2 Level 3 oder
2. FIPS 140-3 Level 3 oder
3. Common Criteria EAL 4+ (mit AVA_VAN.5)

entsprechen. [≤]

A_26065-01 -Nur zugelassene Images in Produktion

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0-Dienstes keine Kenntnis über die in dem ZETA Guard verarbeiteten

Daten erlangen darf, MUSS der Anbieter eine VAU verwenden, die mit technischen Mitteln sicherstellt, dass nur von gematik signierte und für den Einsatz in der PU vorgesehene Produktimages in der PU laufen können. [≤]

Das ZETA Guard-Image ist unabhängig von einer spezifischen VAU-Architektur. Eine VAU ist allerdings in der Regel auf eine bestimmte Architektur (wie z. B. SGX, TDX, SEV usw.) ausgelegt. Da ZETA Guard aus mehreren Komponenten besteht, kann der Hersteller des TI 2.0-Dienstes flexibel entscheiden, wie ZETA Guard auf seiner VAU-Plattform implementiert wird. Dies kann beispielsweise die Nutzung von Confidential Containern, virtuellen Maschinen (VMs) oder die vollständige Ausführung von ZETA Guard innerhalb einer VAU umfassen. Entsprechend dieser Entscheidung muss der Hersteller das ZETA Guard-Image für die jeweils eingesetzte VAU-Architektur umwandeln.

A_28756 -Umsetzung ZETA Guard in Sicherheits- und Datenschutzkonzept

Falls der ZETA Guard in einer VAU umgesetzt wird, muss der Hersteller des TI 2.0-Dienstes die Umsetzung von ZETA auf seiner VAU-Plattform in seinem Sicherheits- und Datenschutzkonzept beschreiben. [≤]

A_28405 -ZETA Guard - Umwandlung für Ziel-VAU-Architektur

Falls der ZETA Guard in einer VAU umgesetzt wird, muss der Hersteller des TI2.0-Dienstes sicherstellen:

- dass das ZETA Guard-Image in einer manipulationssicheren Build-Pipeline für die Ziel-VAU-Architektur erstellt und umgewandelt wird, und
- dass der Build-Log sämtliche VAU-spezifischen Ergänzungen am ZETA Guard-Image protokolliert und für die gematik auditierbar ist.

[≤]

A_28962 -ZETA GUARD - Nachweis der Umsetzung von Anforderungen in dem Build-Pipeline

Falls der ZETA Guard innerhalb einer VAU umgesetzt wird, können die Sicherheits- und Datenschutzanforderungen an den ZETA Guard über die Build-Pipeline erfüllt werden. In diesem Fall MUSS der Hersteller des TI-2.0-Dienstes die Umsetzung dieser Anforderungen im Sicherheits- und Datenschutzkonzept nachvollziehbar beschreiben, sodass ein Gutachter die konforme Umsetzung eindeutig prüfen kann. [≤]

A_28744 -ZETA Guard - Ressourcenserver-Zugriff nur von freigegebenen Komponenten

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Hersteller des TI 2.0-Dienstes sicherstellen, dass der Zugriff auf den Ressourcenserver ausschließlich durch attestierte ZETA-Komponenten erfolgt. Die Attestierungswerte dieser Komponenten müssen vorab eindeutig geprüft und explizit für den Zugriff freigegeben worden sein.

[≤]

A_28745 -ZETA Guard - Zugriff nur von freigegebenen Ressourcenservern

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Hersteller des TI 2.0-Dienstes sicherstellen, dass der Zugriff auf die ZETA-Komponenten ausschließlich durch einen attestierten Ressourcenserver erfolgt. Die Attestierungswerte dieses Ressourcenservers müssen zuvor eindeutig geprüft und explizit für diesen Zugriff freigegeben worden sein.

[≤]

A_28757 -ZETA Guard - Zugriff nur zwischen attestierten Komponenten

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Hersteller des TI 2.0-Dienstes sicherstellen, dass der Zugriff zwischen ZETA-Komponenten ausschließlich durch attestierte ZETA-Komponenten erfolgt. Die Attestierungswerte dieser Komponenten müssen vorab eindeutig geprüft und explizit für den Zugriff freigegeben worden sein.

[≤]

A_28809 -ZETA Guard - Abgesicherte Kommunikation zwischen ZETA-Komponenten in einer VAU

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und der Anbieter des TI 2.0 Dienstes keine Kenntnis über die im ZETA Guard verarbeiteten Daten erlangen darf, MUSS der Anbieter sicherstellen, dass:

- die Kommunikation zwischen den ZETA Guard Komponenten durch mTLS abgesichert ist oder
- sich die Komponenten innerhalb einer gemeinsamen VAU befinden, sodass die unverschlüsselte Kommunikation den geschützten Bereich der VAU zu keinem Zeitpunkt verlässt.

[<=]

5.8 Sicherheits- und Datenschutz Anforderungen an dem ZETA Client in FdVs

A_25802 -ZETA Client - Einhaltung der BSI [TR-03161-1]

Der ZETA Client eines FdVs MUSS die BSI [TR-03161-1] erfüllen, sofern sie für den ZETA Client anwendbar ist. [<=]

Hinweis: Nicht anwendbar können zum Beispiel sein: O.Paid .. Die Anwendbarkeit ist zwischen Hersteller des ZETA Clients und dem Gutachter zu klären. Der Gutachter gibt sein Votum über die Erfüllung der BSI [TR-03161-1] in Form der Bewertung der Erfüllung der A_25802 ab, wobei die A_25802 als „umgesetzt“ bewertet werden kann, wobei die anwendbaren Abschnitte der BSI [TR-03161-1] aus Sicht des Gutachters erfüllt sind.

5.9 Schlüssel Management und Verwaltung in ZETA

Teil der Sicherheitsarchitektur von ZETA ist ein robustes Schlüsselmanagement. Um Entwicklern, Betreibern und Auditoren ein klares Verständnis der kryptografischen Architektur zu vermitteln, verwendet diese Spezifikation standardisierte Schlüssel-IDs (z.B. PrK.AuthS.Sig für den privaten Signaturschlüssel des Authorization Servers).

5.9.1 Nomenklatur für Schlüssel-IDs

Jeder Schlüssel erhält eine eindeutige ID nach folgendem Schema: [Typ].[Komponente].[Zweck]

Typ:

- PrK: Privater Schlüssel (Private Key)
- PuK: Öffentlicher Schlüssel (Public Key)
- C: Zertifikat (Public Key inkl. Identitätsbindung, Certificate)
- SymK: Symmetrischer Schlüssel (Symmetric Key)

Komponente oder Schlüssel:

- AuthS: Authorization Server (PDP)
- PEP: Policy Enforcement Point (HTTP Proxy)
- DB: Datenbank (PDP DB)
- Client: ZETA Client (App / Primärsystem)

- K8s: Kubernetes / Infrastruktur
- Sys: Gematik / Zentrales System
- AK: Attestation Keys
- EK: TPM Endorsement Keys
- DPoP: DPoP Keys
- SM(C)-B: SM(C)-B Keys
- TI-RootCA: TI Root CA Signer Keys
- TI-TSL: TI TSL Signer Keys
- TI-KompCA: TI Komponenten PKI CA Keys
- TI-FedMaster: Federation Master Signer Keys
- ZETA-IK: ZETA OCI Container Image Signer Keys
- ZETA-DK: ZETA OCI Data Image Signer Keys
- ZETA-PAK: ZETA OPA Bundle Autor Signer Keys
- ZETA-PFK: ZETA OPA Bundle Freigeber Signer Keys

Zweck:

- TLS: Verschlüsselung des externen Traffics
- mTLS: Interne Komponenten Authentifizierung
- ASL: Application Security Layer
- Sig: Signatur (Token, Policies, Entity Statements)
- Enc: Payload-/Daten-Verschlüsselung (JWE, DB At-Rest)

5.9.2 Übersicht der ZETA Guard Schlüssel (Anbieter-Seite)

Diese Tabelle fasst alle Schlüssel zusammen, die vom Betreiber des TI 2.0-Dienstes (ZETA Guard) verwaltet werden müssen. Bei Verarbeitung von Daten mit Schutzbedarf "sehr hoch" greifen strenge VAU- und HSM-Pflichten.

Hinweis: Schlüssel mit dem Vermerk "Die Schlüsselverwaltung muss dokumentiert werden", werden vom Anbieter des TI 2.0 Dienstes verwaltet. Die Schlüssel ohne Vermerk verwaltet ZETA Guard automatisch.

Betriebsmodi und Schlüsselschutz:

Der ZETA Guard unterscheidet zwei primäre Betriebsmodi:

11. Regulärer Betrieb (ohne VAU):Schlüssel werden in sicheren Software-Keystores (z.B. KMS v2) der jeweiligen Container-Umgebung abgelegt.

12. Betrieb mit Schutzbedarf "sehr hoch" (mit VAU):Sobald sensible Daten verarbeitet werden, muss der ZETA Guard in einer Vertrauenswürdigem Ausführungsumgebung (VAU) betrieben werden (siehe A_25608-01). In diesem Fall **MÜSSEN** asymmetrische private Schlüssel der TI- und Internet-Identitäten zwingend in einem Hardware Security Module (HSM) verbleiben. Schlüssel, die in den Arbeitsspeicher der VAU geladen werden müssen (z.B. Token-Signatur- oder Datenbank-Schlüssel), müssen durch HSM-generierte Key-Encryption-Keys (KEK) "gewrappt" (Sealing) übergeben werden.

Tabelle 7: Tab-ZETA-Guard-Keys

<u>Schlüssel-ID</u>	<u>Bezeichnung & Zweck</u>	<u>Speicherung / Betrieb OHNE VAU</u>	<u>Speicherung / Betrieb MIT VAU (Schutzbedarf "sehr hoch")</u>
<u>PrK.AuthS.Sig</u> <u>PuK.AuthS.Sig</u>	<u>AuthS Token-Signaturschlüssel</u> <u>Zur Signatur von Access- und Refresh-Token sowie des Entity Statements. Der PuK wird im JWKS bereitgestellt.</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>PrK: Sicherer Software-Keystore des AuthS.</u>	<u>PrK: Muss durch HSM geschützt sein. (Darf in den AuthS geladen werden, wenn z.B. mittels KEK aus dem HSM gesichert).</u>
<u>PrK.AuthS.TLS</u> <u>C.AuthS.TLS</u>	<u>AuthS Internet-Identität (TLS)</u> <u>Terminierung der externen TLS-Verbindung am Authorization Server.</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>Sicherer Software-Keystore.</u>	<u>PrK: Verbleibt zwingend im HSM (via HSM-Proxy), falls kein ASL Tunnel zum AuthS existiert.</u>
<u>SymK.DB.Enc</u>	<u>Datenbank-Verschlüsselung</u> <u>Verschlüsselung der Session-, Nutzer- und Client-Daten At-Rest im Authorization Server.</u>	<u>Sicherer Software-Keystore (z.B. K8s Secrets).</u>	<u>Muss durch HSM geschützt sein. Übergabe an AuthS darf nur an attestierte Instanzen erfolgen.</u>
<u>PrK.PEP.TLS</u> <u>C.PEP.TLS</u>	<u>PEP Internet-Identität (TLS)</u> <u>Terminierung der externen TLS-Verbindung am HTTP Proxy.</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>Sicherer Software-Keystore.</u>	<u>PrK: Verbleibt zwingend im HSM, falls kein ASL Tunnel zum HTTP Proxy existiert.</u>
<u>PrK.PEP.Sig</u> <u>C.PEP.Sig</u>	<u>PEP Signatur-Identität</u> <u>Zertifikat aus der Komponenten-PKI. Dient als Identität des PEP (für Signatur des ASL Master-Schlüssels).</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>Sicherer Software-Keystore.</u>	<u>Privater Schlüssel verbleibt zwingend im HSM (Zugriff via HSM-Proxy).</u>
<u>PrK.PEP.ASL</u> <u>PuK.PEP.ASL</u>	<u>PEP ASL-Identität</u> <u>Semi-statische Schlüsselpaare für den ZETA/ASL-Kanal (maximal 1 Monat gültig).</u>	<u>Im RAM des PEP.</u>	<u>Wird im PEP generiert, aber vom PrK.PEP.Sig im HSM beglaubigt.</u>
<u>SymK.PEP.ASL</u>	<u>PEP ASL Session-Key</u>	<u>Im RAM des</u>	<u>Wird im PEP generiert.</u>

	<u>Abgeleiteter kurzlebiger Schlüssel für den eigentlichen ASL-Traffic.</u>	<u>PEP.</u>	
<u>PrK.ZG.IDP</u> <u>PuK.ZG.IDP</u>	<u>IDP für die ZETA Guard Workload Identity</u> <u>Zur Signatur von Subject Token für die Kommunikation mit gematik-Diensten (SIEM, Telemetrie) und für die Dienst-zu-Dienst Kommunikation.</u> <u>Initial: Kubernetes IDP oder langlebiger Schlüssel</u> <u>Zukünftig: AuthS als IDP des ZETA Guard</u> <u>mit PrK.AuthS.Sig und PuK.AuthS.Sig</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>Innerhalb der Kubernetes-Infrastruktur des Anbieters.</u>	<u>Wenn langlebiger Schlüssel, dann wird der private Schlüssel im HSM gespeichert.</u>
<u>PrK.Ingress.TLS</u> <u>C.Ingress.TLS</u>	<u>Ingress TLS</u> <u>TLS-Terminierung am vorgeschalteten Ingress (falls genutzt).</u> <u>Die Schlüsselverwaltung muss dokumentiert werden.</u>	<u>Sicherer Software-Keystore.</u>	<u>Verbleibt zwingend im HSM, falls kein ASL Tunnel zum AuthS und zum HTTP Proxy existiert.</u>
<u>PrK.K8s.mTLS</u> <u>C.K8s.mTLS</u>	<u>Interne Mesh-Identitäten</u> <u>Zur Absicherung der microservice-internen Kommunikation (z. B. AuthS <-> PE).</u>	<u>K8s Service Mesh (z.B. Istio).</u>	<u>Verwaltung innerhalb der VAU (Zugriff strikt limitiert).</u>
<u>PuK.Client.Sig</u>	<u>Öffentlicher Client Instance Key</u> <u>Wird bei der initialen Client-Registrierung an den AuthS gesendet. Dient der Validierung der "Client Assertion" bei zukünftigen Logins.</u>	<u>Gespeichert in der PDP Datenbank.</u>	<u>Gespeichert in der PDP Datenbank. Muss vor unautorisierter Manipulation (Austausch durch Angreifer) geschützt sein (Integrität).</u>

A_28826 -HSM Proxy, Übergabe PDP-Datenbank-Schlüssel an ZETA Guard Authorization Server

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Übergabe der PDP-Datenbank-Schlüssel (SymK.DB.Enc) ausschließlich an einen attestierten und freigegebenen ZETA Guard Authorization Server erfolgt. [<=]

A_28863 -HSM Proxy, Verwendung AuthS-Schlüssel nur durch ZETA Guard Authorization Server

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Verwendung der AuthS-Schlüssel (PrK.AuthS.Sig, PrK.AuthS.TLS) im HSM

ausschließlich durch einen attestierten und freigegebenen ZETA Guard Authorization Server möglich ist. [<=]

A_2886A_25 -HSM Proxy, Verwendung PEP-Schlüssel nur durch ZETA Guard HTTP Proxy

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Verwendung der PEP-Schlüssel (PrK.PEP.TLS, PrK.PEP.Sig) im HSM ausschließlich durch einen attestierten und freigegebenen ZETA Guard HTTP Proxy möglich ist. [<=]

A_28865 -HSM Proxy, Verwendung Ingress-Schlüssel nur durch ZETA Guard Ingress

Falls ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM-Proxy sicherstellen, dass die Verwendung der Ingress-Schlüssel (PrK.Ingress.TLS) im HSM ausschließlich durch einen attestierten und freigegebenen ZETA Guard Ingress möglich ist. [<=]

5.9.3 ZETA Client Schlüssel (Nutzer-Seite)

Diese Schlüssel verbleiben in der Verfügungsgewalt des Endnutzers (Smartphone / Primärsystem) bzw. der Institution.

Tabelle 8: Tab-ZETA-Client-Keys

<u>Schlüssel-ID</u>	<u>Bezeichnung & Zweck</u>	<u>Speicherung / Verwendung</u>
<u>PrK.Client.Sig PuK.Client.Sig</u>	<u>Client Instance Key Pair Langlebiges ECC-Schlüsselpaar zur eindeutigen Identifikation der Client-Installation. Dient der Signatur der Client Assertion.</u>	<u>PrK: Zwingend in Hardware (TPM, Secure Enclave, TEE) generiert und gespeichert. Erhält strikt das Attribut sign (kein Export möglich). PuK: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen.</u>
<u>PrK.DPoP.Sig PuK.DPoP.Sig</u>	<u>DPoP Schlüsselpaar Kurzlebiger (Session-basierter) Schlüssel zum Signieren von Anfragen als Beweis für den Besitz des Access Token (Proof of Possession).</u>	<u>PrK: Wird für die Dauer der Session sicher lokal gehalten (flüchtig im RAM oder durch native OS-Sicherheitsmechanismen / TPM-Wrapping geschützt). Eine Verschlüsselung des DPoP-Keys durch den PrK.Client.Sig ist aus Gründen der Schlüsseltrennung nicht zulässig. PuK: Wird im HTTP-Header gesendet.</u>
<u>PrK.AK.Sig PuK.AK.Sig</u>	<u>Plattform Attestation Key Zur Signatur des Client-Zustands.</u>	<u>PrK: Wird zur Signatur des Client-Zustands verwendet. Hardware-gebunden (TPM / Secure Enclave). PuK: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen und zur Prüfung der Systemintegrität gesendet (inkl. Zertifikatskette zur Validierung gegen Root-CA des Herstellers).</u>
<u>PrK.EK.Enc PuK.EK.Enc C.EK.Enc</u>	<u>TPM 2.0 Endorsement Keys Wird bei der TPM Attestation verwendet.</u>	<u>PrK: Hardware-gebunden (TPM / Secure Enclave). PuK: Wird bei der Registrierung</u>

		(DCR) an den ZETA Guard übertragen und zur Bindung des AK an den EK verwendet. C: Wird bei der Registrierung (DCR) an den ZETA Guard übertragen und zur Bindung des AK an den EK verwendet.
PrK.SM(C)-B.Sig C.SM(C)-B.Sig	SMC-B Institutionsidentität Ausgestellt von der SMC-B CA. Zur Signatur des subject_token beim Token Exchange, um die Identität der Institution (z.B. Praxis) nachzuweisen. Das Subject Token enthält ein Binding des PuK.Client.Sig und des PuK.DPoP.Sig.	PrK: Verbleibt hardwaregebunden auf der Smartcard (SMC-B) oder im HSM-B. C: Wird übermittelt und durch AuthS gegen TSL validiert.

5.9.4 gematik verwaltete Schlüssel (TI)

Diese Zertifikate und Schlüssel spannen den Vertrauensraum der TI 2.0 auf. Die privaten Schlüssel liegen hochsicher bei der gematik (bzw. den zugelassenen Trust Centern). Die ZETA Guards und Clients nutzen die öffentlichen Teile/Zertifikate zur Validierung. Diese Schlüssel dienen der Sicherstellung der Supply-Chain-Security und der Integrität von Regelwerken.

Tabelle 9: Tab-Gematik-Keys

Schlüssel-ID	Bezeichnung & Zweck	Speicherung / Verteilung
PrK.TI-RootCA.Sig C.TI-RootCA.Sig	TI Root CA Oberster Vertrauensanker der TI. Alle Zertifikate im System leiten sich hiervon ab.	C: Lokal in Truststores hinterlegt. Wird mit dem ZETA Guard Provisioning OCI Image in den ZETA Guard geladen. PrK: Offline/Hochsicher bei der gematik.
PrK.TI-TLSig C.TI-TSL.Sig	TSL Signer Zertifikat zur Validierung der Signatur der Trust-Service Status List (TSL), welche die aktuell gültigen CAs definiert.	C: Im ZETA Guard (über lokales Artifact Registry) zur TSL-Verifikation. PrK: Bei der gematik.
PrK.TI-KompCA.Sig C.TI-KompCA.Sig	Komponenten PKI CA Sub-CA der Root CA. Stellt die Zertifikate für die TI-Dienste (PEP, AuthS) aus.	C: Über die TSL als vertrauenswürdig verteilt.
PrK.TI-SMCB-CA.Sig C.TI-SMCB-CA.Sig	SMC-B CA Sub-CA der Root CA. Stellt die Institutionszertifikate für Leistungserbringer aus.	C: Über die TSL als vertrauenswürdig verteilt (zur Prüfung im ZETA Guard).

PrK.TI-FedMaster.Sig PuK.TI-FedMaster.Sig	<u>Federation Master Signer Zur Signatur der Entity Statements (Trustmarks) in der OIDC-Föderation.</u>	<u>PuK: Im ZETA Guard hinterlegt, um die Föderations-Zugehörigkeit anderer AuthS zu prüfen. PrK: Bei der gematik.</u>
PrK.ZETA- IK.Sig C.TI-ZETA- IK.Sig	<u>Signaturzertifikat Ausführbare ZETA Images Signatur der ausführbaren ZETA OCI Container (PEP, AuthS, OPA, etc.).</u>	<u>C: Im Admission Controller validiert (Signaturkette bis zur Root CA). PrK: In gematik CI/CD-Pipeline.</u>
PrK.ZETA- DK.Sig C.TI-ZETA- DK.Sig	<u>Signaturzertifikat Daten-Images Signatur von Datencontainern (z.B. Konfigurationsdaten, Provisioning-Daten).</u>	<u>C: Im Admission Controller / durch ZETA Guard validiert. PrK: In gematik CI/CD-Pipeline.</u>
PrK.ZETA- PAK.Sig C.TI-ZETA- PAK.Sig	<u>OPA Policy-Autor Signatur Signaturzertifikat des Policy- Erstellers für OPA Bundles.</u>	<u>C: Im ZETA Guard (OPA Engine) zur Signaturprüfung konfiguriert. PrK: Beim Datenbearbeiter.</u>
PrK.ZETA- PFK.Sig C.TI-ZETA- PFK.Sig	<u>OPA Policy-Freigeber Signatur Zweitsignatur (4-Augen-Prinzip) für OPA Bundles.</u>	<u>C: Im ZETA Guard (OPA Engine) konfiguriert. PrK: Beim Freigeber.</u>
PrK.K8s.IDP C.K8s.IDP	<u>Kubernetes IDP (Workload Identity) Zur Signatur von Subject Token für die Kommunikation mit gematik-Diensten (SIEM, Telemetrie).</u>	<u>PrK: Innerhalb der Kubernetes- Infrastruktur des Anbieters. C: Innerhalb des gematik Workload Identity Pools</u>

5.10 ZETA Abläufe

Dieses Kapitel spezifiziert die dynamischen Kommunikations- und Prozessabläufe innerhalb der ZETA-Architektur. Um die Interaktionen zwischen dem ZETA Client, dem ZETA Attestation Service (ZAS), dem ZETA Guard (Authorization Server) und dem Resource Server nachvollziehbar darzustellen, sind die Unterkapitel chronologisch anhand des Lebenszyklus einer Client-Instanz strukturiert.

Die Beschreibung folgt dem Weg von stationären und mobilen Clients von der initialen Bereitstellung (Installation und Start) über die netzwerktechnische Ermittlung der benötigten Endpunkte (Service Discovery) bis hin zur initialen Schlüsselbindung und Registrierung (Dynamic Client Registration). Darauf aufbauend wird detailliert dargestellt, wie der Client unter fortlaufendem Nachweis seiner Systemintegrität (Attestierung) Access Tokens bezieht und diese für den sicheren Zugriff auf geschützte Fach-dienste (Resource Server) nutzt. Den Abschluss bilden die Prozesse zur Session-Erneuerung sowie die Besonderheiten bei der Dienst-zu-Dienst-Kommunikation.

5.10.1 Abläufe für stationäre Clients

Dieses Unterkapitel beschreibt die spezifischen ZETA-Abläufe für stationäre Endgeräte (Primärsysteme). Im Zentrum steht dabei der Nachweis der Systemintegrität, welcher primär hardwaregestützt über ein Trusted Platform Module (TPM) oder eine Secure Enclave erfolgt. Für Systeme ohne entsprechende Hardware-Voraussetzungen wird eine Software-Attestierung (SW Attestation) als Fallback unterstützt. Auf Basis dieser Attestierung erfolgt die Autorisierung beim ZETA Guard nach OAuth Token Exchange Grant, um die sichere Ausstellung und den Austausch von Access Tokens für den Zugriff auf geschützte Ressourcen zu regeln.

Abbildung 3: Attestierungsablauf nach Betriebssystem gibt einen vereinfachten Überblick über die ganzheitlichen Prozesspfad über die verschiedenen Betriebssystemvarianten und dient als Leitfaden für die folgenden Detailbeschreibungen.

Überblick Attestierungsabläufe nach Betriebssystem

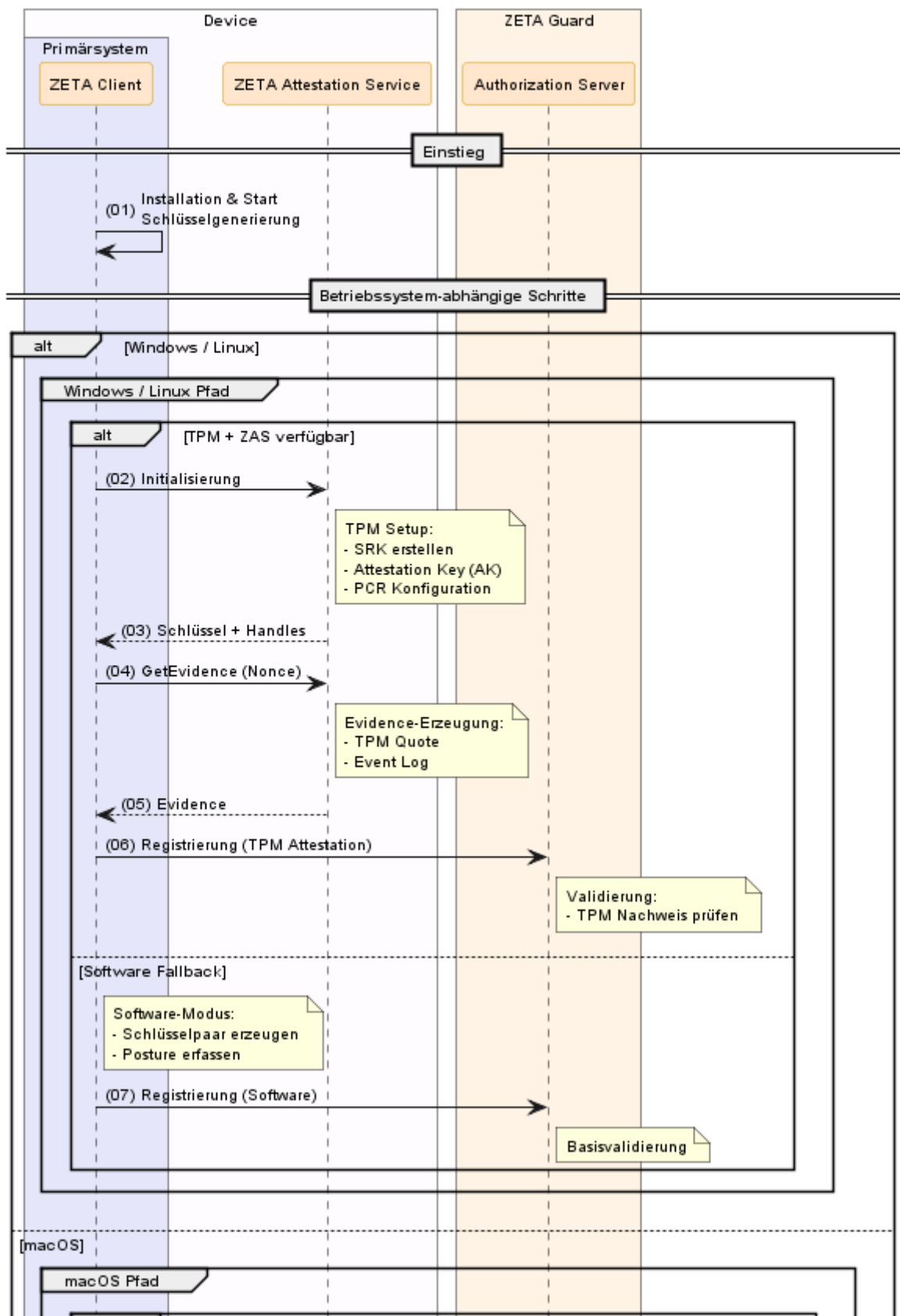


Abbildung 3 : Abb-Attestierungsablauf-nach-Betriebssystem

Der Ablauf kann wie folgt zusammengefasst werden.

3. Initialisierung

- Installation des Clients und Generierung eines Client-Schlüsselpaares
- Aufbau einer initialen System-Baseline (Messwerte, Schlüsselmaterial)

4. Hardware-abhängiger Attestierungspfad

- Windows / Linux
 - Primär: TPM-basierte Attestierung durch den ZETA Attestation Service (ZAS)
 - Fallback: Software-basierte Attestierung ohne TPM
- macOS
 - Primär: Secure Enclave + Apple App Attest / DeviceCheck
 - Fallback: Software-basierte Attestierung

5. Vorbereitung der Registrierung

- Bereitstellung der attestierungsrelevanten Schlüssel und Nachweise
- Plattform-spezifische Attestierungsobjekte:
 - TPM: EK-, AK- und Client-Schlüssel + Zertifikate
 - macOS: Apple Attestation Object
 - Software: ausschließlich Client-Schlüssel

6. Client-Registrierung beim ZETA Guard

- Durchführung eines Dynamic Client Registration (DCR)
- Ablauf abhängig vom Attestierungstyp:
 - TPM (Windows/Linux)
 - Secure Enclave (macOS)
 - Software (Fallback)

7. Laufende Attestierung und Authentifizierung

- Regelmäßige Erhebung von Zustandsdaten (Evidence)
- Token-Abruf über OAuth Token Exchange
- optionales Attestation Token bei hardwaregestützter Vertrauensbildung

5.10.1.1 Client Installation und Schlüsselgenerierung

Nach der Installation werden beim ersten Start des Primärsystems die Schlüssel generiert, die vom ZETA Client als Identitäts-Nachweis gegenüber ZETA Guard Instanzen verwendet werden.

Das Ergebnis des Prozesses ist eine initiale Baseline des Clientsystems, die – abhängig vom gewählten Attestierungsmodus – aus hardware- oder software-gebundenen Schlüsseln sowie initial erhobenen Zustands- und Messinformationen besteht. Diese Baseline bildet die Grundlage für alle weiteren Schritte des Attestierungslebenszyklus.

5.10.1.1.1 Schlüsselgenerierung auf Windows und Linux Systemen

Die Installation des Primärsystems (inkl. ZETA Client) erfolgt in der Regel ohne Admin Rechte. Um Zugriff auf die TPM Funktionen zu erhalten, ist jedoch eine Komponente erforderlich, die unter Admin-Rechten ausgeführt wird. Dafür wird der ZETA Attestation Service (ZAS) verwendet. Zwischen ZAS und ZETA Client muss eine Vertrauensbeziehung bestehen, damit nicht beliebige Clients vom ZAS Attestation-Daten anfragen können und damit der ZETA Client nicht mit einem kompromittierten ZAS kommuniziert.

Das folgende Sequenzdiagramm beschreibt den Prozess der Schlüsselgenerierung und Attestierung auf Windows und Linux Betriebssystemen. Der Ablauf gliedert sich in einen primären, hardwaregestützten Pfad über das TPM und einen Software-Attestation Fallback-Pfad.

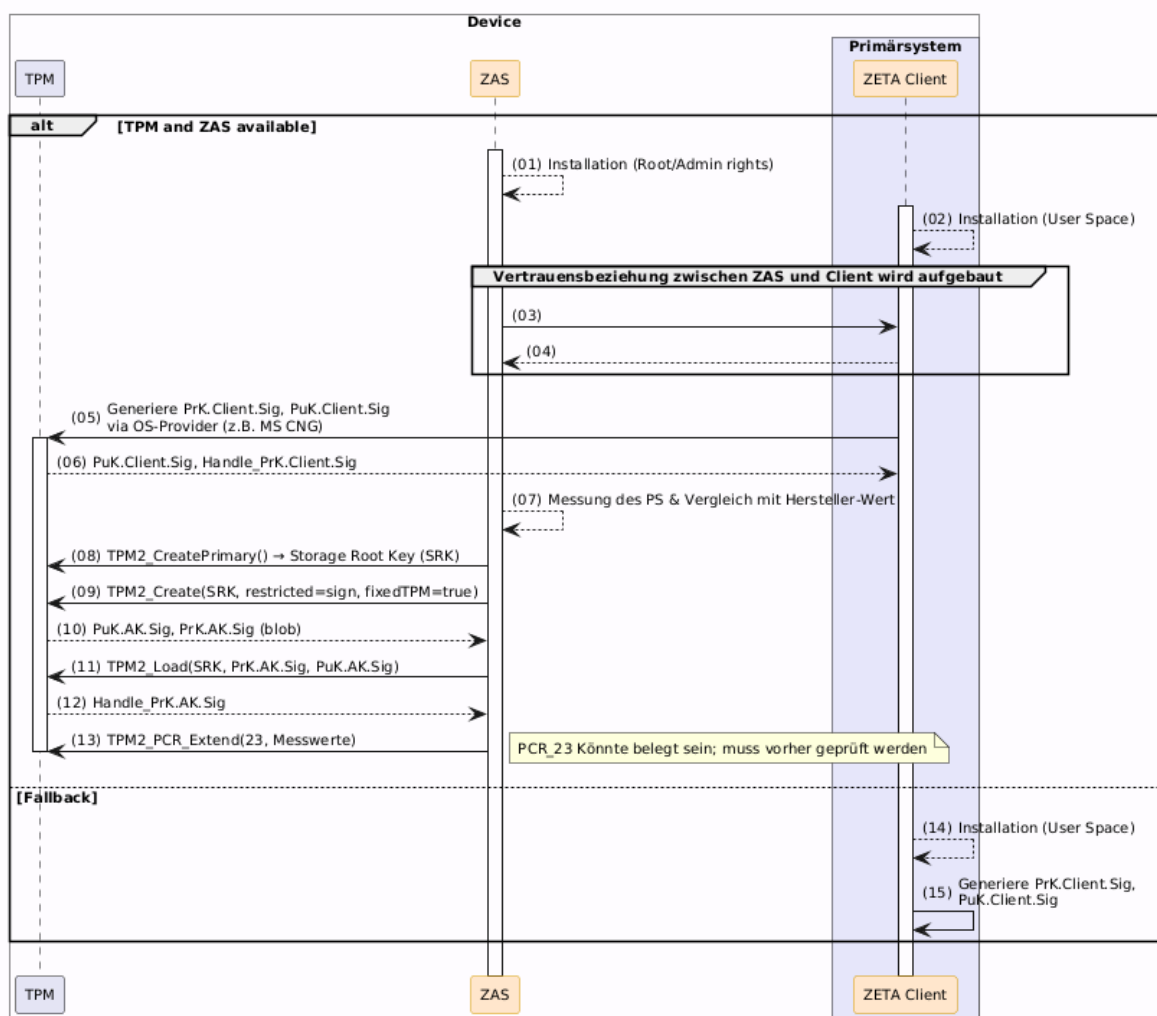


Abbildung 4 : Abb-ZETA-Schlüsselgenerierung-Windows-und-Linux

Die Schritte (01)-(13) werden nur durchgeführt, wenn auf dem Clientsystem ein TPM vorhanden ist und das Primärsystem ein ZETA Attestation Service beinhaltet.

- (01) Privilegierte Installation: Der ZETA Attestation Service wird mit administrativen Rechten installiert, damit Messungen des Primärsystems in ein TPM Register geschrieben werden können.
- (02) Die Installation des Primärsystems (mit Ausnahme des ZAS) erfolgt ohne Admin Rechte.

- (03)-(04) Um eine exklusive und sichere Vertrauensbeziehung zwischen dem ZAS (läuft privilegiert als Root/Admin) und dem Primärsystem (läuft im User Space) herzustellen, müssen Betriebssystem-Mechanismen (IPC-Sicherheit) mit kryptographischer Bindung kombiniert werden. Da beide Komponenten vom selben Hersteller stammen, können zudem Code-Signatur-Prüfungen genutzt werden.
 - (05) Erzeugung eines Client-Signaturschlüsselpaares: Auf Windows- oder Linux-Systemen wird ein Client-Signaturschlüsselpaar über einen Betriebssystem- oder TPM-Provider erzeugt.
 - (06) Übergabe von Schlüsselmaterial: Die öffentlichen Schlüsselanteile sowie zugehörige Schlüssel-Handles werden dem ZETA Client bereitgestellt.
 - (07) Messung des Primärsystems: Der Zustand des Primärsystems wird gemessen. Die Messwerte KÖNNEN optional mit herstellerseitig bereitgestellten Referenzwerten verglichen werden.
 - (08) Erzeugung einer Storage Root Key (SRK): Eine Storage Root Key wird innerhalb des TPM als Vertrauensanker erzeugt.
 - (09) Erzeugung des Attestierungsschlüssels (AK): Ein Attestierungsschlüssel wird erzeugt und fest an das TPM gebunden.
 - (10) Bereitstellung der öffentlichen AK-Anteile: Die öffentlichen Anteile des Attestierungsschlüssels werden exportiert.
 - (11) Laden des Attestierungsschlüssels: Der Attestierungsschlüssel wird in das TPM geladen und aktiviert.
 - (12) Übergabe des AK-Handles: Der ZETA Client erhält einen Handle zur Referenzierung des Attestierungsschlüssels.
 - (13) Erweiterung von PCR-Registern: Die ermittelten Messwerte werden in geeignete Platform Configuration Registers (PCRs) erweitert. Vor der Erweiterung prüft der ZAS, ob die jeweiligen PCR-Register bereits belegt sind. Nur wenn ein PCR frei ist, kann der ZAS das TPM verwenden. Wenn kein PCR frei ist, muss die Fallback Alternative ohne TPM Nutzung verwendet werden. In den weiteren Schritten ist dann nur die Software-Attestation nutzbar.
- Die Schritte (14)-(15) stellen eine Fallback Alternative zu den TPM-spezifischen Schritten dar.
- (14) Die Installation des Primärsystems erfolgt ohne Admin Rechte.
 - (15) Der ZETA Client erzeugt das Client-Signaturschlüsselpaar lokal im Software-Kontext.

5.10.1.1.2 Schlüsselgenerierung auf macOS Systemen

Siehe 5.3.2.1.2- Apple Secure Enclave

5.10.1.2 Client Start mit TPM und ZAS

Wenn unter Windows oder Linux ein TPM 2.0 und der ZAS verfügbar sind, dann wird nach dem Booten und bei jedem Start des Primärsystems eine Messung durch den ZAS durchgeführt. Die Messung wird im weiteren Verlauf bei der Remote Attestierung im ZETA Guard mit einem vom Hersteller vorgegebenen Wert verglichen.

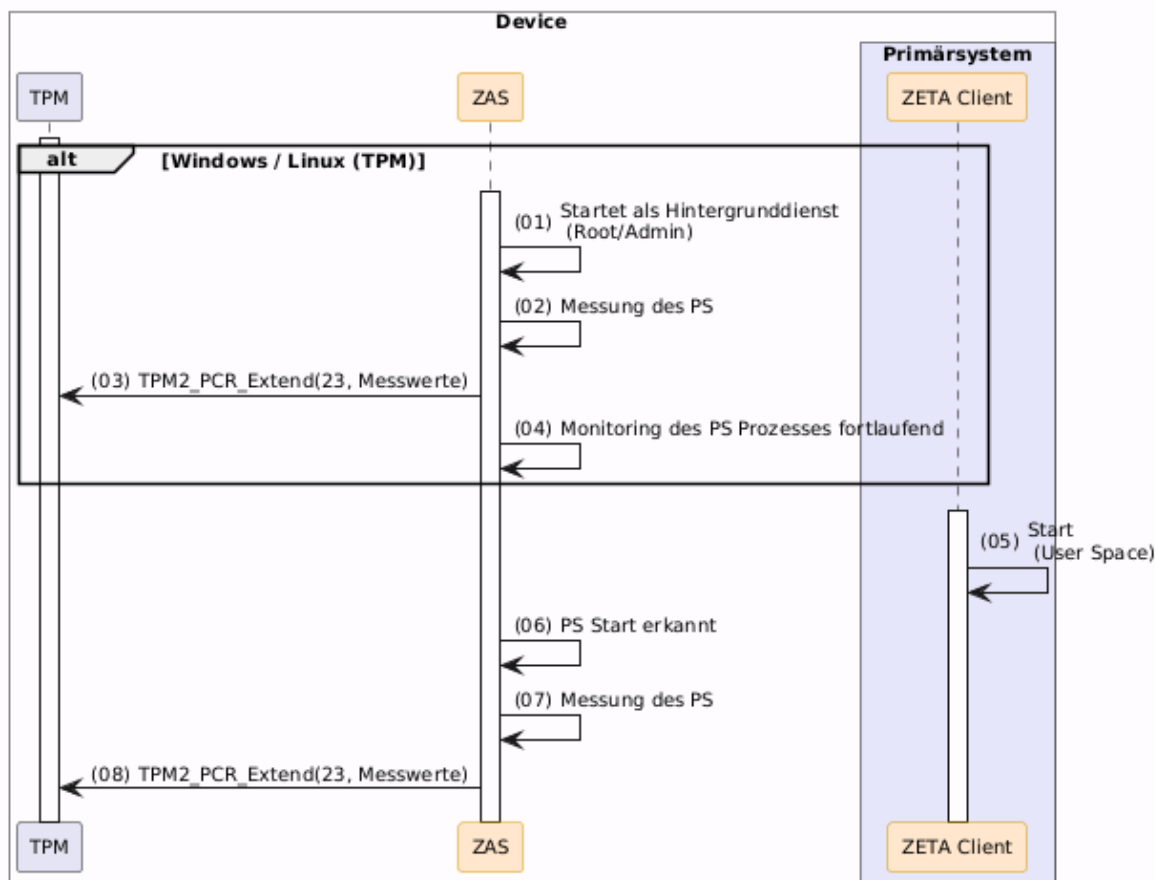


Abbildung 5 Abb-ZETA-Client-Start-mit-TPM-und-ZAS

- (01) Der ZAS wird während des Starts des Betriebssystems als Service gestartet.
- (02)-(03) Der ZAS führt eine Messung der unveränderlichen Teile des Primärsystems durch und trägt sie in ein TPM PCR ein.
- (04) Durch ein kontinuierliches Monitoring prüft der ZAS, ob das Primärsystem gestartet wurde.
- (05)-(08) Wenn ein Start des Primärsystems erkannt wurde, führt der ZAS eine Messung der unveränderlichen Teile des Primärsystems durch und trägt sie in ein TPM PCR ein.

5.10.1.3 Service Discovery

Das folgende Sequenzdiagramm beschreibt den sogenannten Discovery-Prozess (Metadaten-Abruf) eines ZETA Clients in der TI 2.0 Architektur. Bevor der Client Zugriffstokens anfordern kann, muss er dynamisch herausfinden, welcher Autorisierungsserver für eine bestimmte Ressource zuständig ist, welche Konfiguration für den Zugriff verwendet werden muss (z. B. ZETA/ASL Nutzung) und welche Endpunkte und Konfigurationen der TI 2.0 Dienst nutzt.

Dieser Prozess basiert auf etablierten IETF/OAuth-Standards (RFCs) und teilt sich in zwei aufeinanderfolgende Abfragen auf.

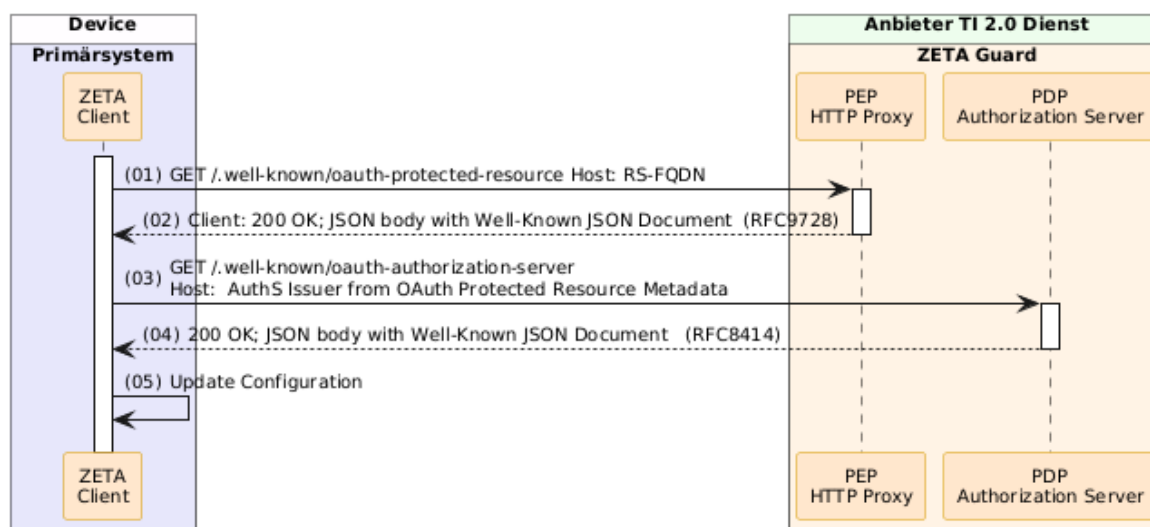


Abbildung 6 Abb-ZETA-Service-Discovery

(01) Der ZETA Client sendet einen HTTP GET Request an den PEP HTTP Proxy. Der Proxy agiert hier als Stellvertreter für die eigentliche Fachanwendung (Resource Server / RS-FQDN). Der Pfad `/.well-known/oauth-protected-resource` ist standardisiert und dient exakt diesem Zweck.

(02) Antwort des Proxys: Der PEP Proxy antwortet mit 200 OK und liefert ein JSON-Dokument zurück. Gemäß RFC 9728 (OAuth 2.0 Protected Resource Metadata) enthält dieses Dokument Metadaten über die Ressource. Die für den Client wichtigste Information darin ist der Issuer (die URL) des Autorisierungsservers (PDP), der diese spezielle Ressource schützt. Das Schema für das OAuth Protected Resource Well-known JSON Dokument ist in `[opr-well-known.yaml]` festgelegt.

Nachdem der Client nun weiß, mit wem er für die Authentifizierung sprechen muss, fragt er diesen Server nach seinen Fähigkeiten und Endpunkten.

(03) Der ZETA Client sendet nun einen GET Request an den in Schritt (02) ermittelten PDP Authorization Server. Er fragt den standardisierten Pfad `/.well-known/oauth-authorization-server` ab.

(04) Der Authorization Server antwortet mit 200 OK und einem weiteren JSON-Dokument. Nach RFC 8414 (OAuth 2.0 Authorization Server Metadata) enthält dieses Dokument die vollständige Konfiguration des Servers. Dazu gehören z.B. die URLs für den Token-Endpoint (um Tokens anzufordern), den Registration-Endpoint (DCR), unterstützte Verschlüsselungsalgorithmen, Scopes und die URL zu den öffentlichen Schlüsseln (JWKS) des Servers. Über das Feld `api_versions_supported` gibt der Authorization Server zudem bekannt, welche Vertragsversionen der Token-Ausstellung er unterstützt; hieran erkennt der ZETA Client, ob er den Parameter `resource` (Contract v2) verwenden kann oder den Parameter `audience` (Contract v1, Legacy) verwenden muss (siehe 5.4.2.5). Das Schema für das OAuth Authorization Server Well-known JSON Dokument ist in `[as-well-known.yaml]` festgelegt.

(05) Update Configuration: Der ZETA Client verarbeitet die empfangenen JSON-Dokumente aus Schritt (02) und (04) lokal. Er aktualisiert seine interne Konfiguration mit den abgerufenen Endpunkten und Parametern.

A_29374 -ZETA, Ablauf Service Discovery

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-Service-Discovery* unterstützen. [**<=**]

5.10.1.4 Vorbereitung der Client-Registrierung beim ZETA Guard

Wenn ein ZAS und TPM oder eine Secure Enclave verwendet werden, dann müssen Schlüssel für die Attestierung beim Authorization Server bekannt gemacht werden, um die Attestierung dort verifizieren zu können.

5.10.1.4.1 ZAS und TPM

Der folgende Ablauf zeigt die erforderlichen Schritte, um alle benötigten Schlüssel für die Verifikation per TPM Attestation einzusammeln.

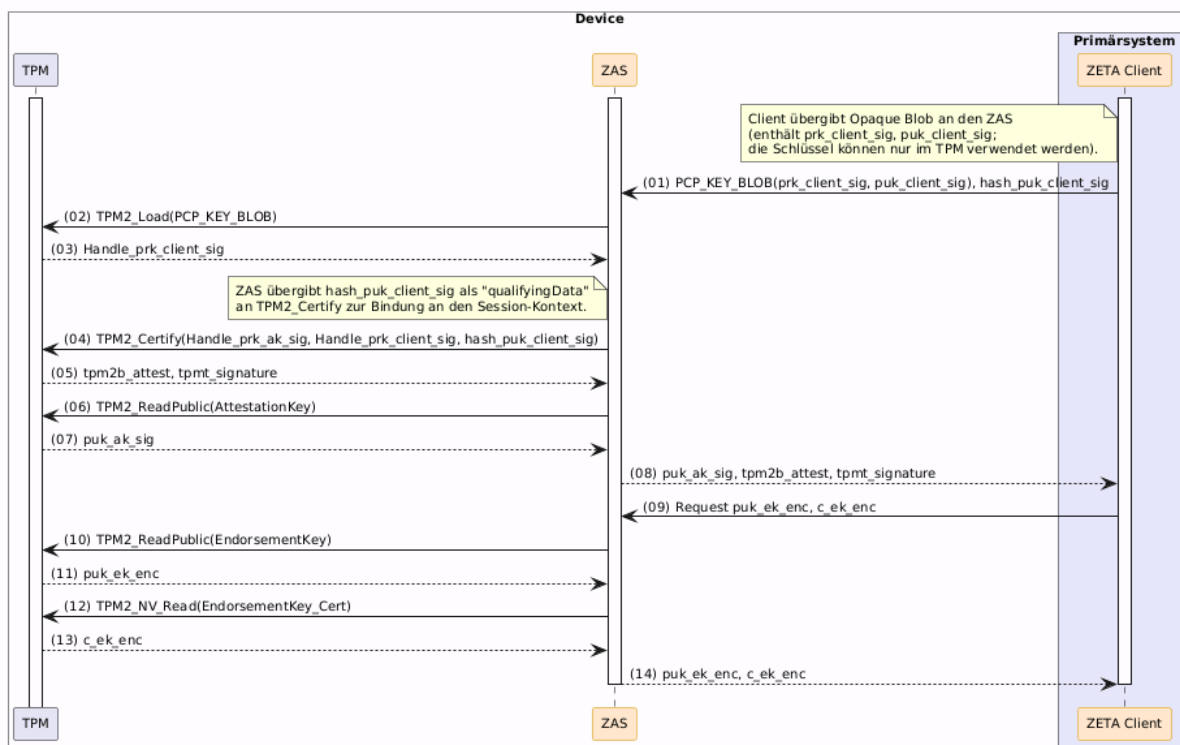


Abbildung 7 Abb-ZETA-TPM-Attestation-Key

(01)-(03) Das TPM hat nur wenig Speicher. Daher werden anwendungsspezifische Schlüssel außerhalb des TPM als PCP_KEY_BLOB gespeichert. Die Schlüssel können dennoch nur im TPM verwendet werden. Vor der Verwendung müssen die Schlüssel in das TPM geladen werden.

(04)-(05) Es wird ein TPM2 Attest erzeugt, das das Schlüsselpaar PrK.Client.Sig und PuK.Client.Sig als TPM-Schlüssel bestätigt.

(06)-(14) Aus dem TPM werden der PuK.AK.Sig und der Puk.EK.Enc Schlüssel sowie das C.EK.Enc Zertifikat ausgelesen und an den ZETA Client übergeben.

5.10.1.4.2 Secure Enclave

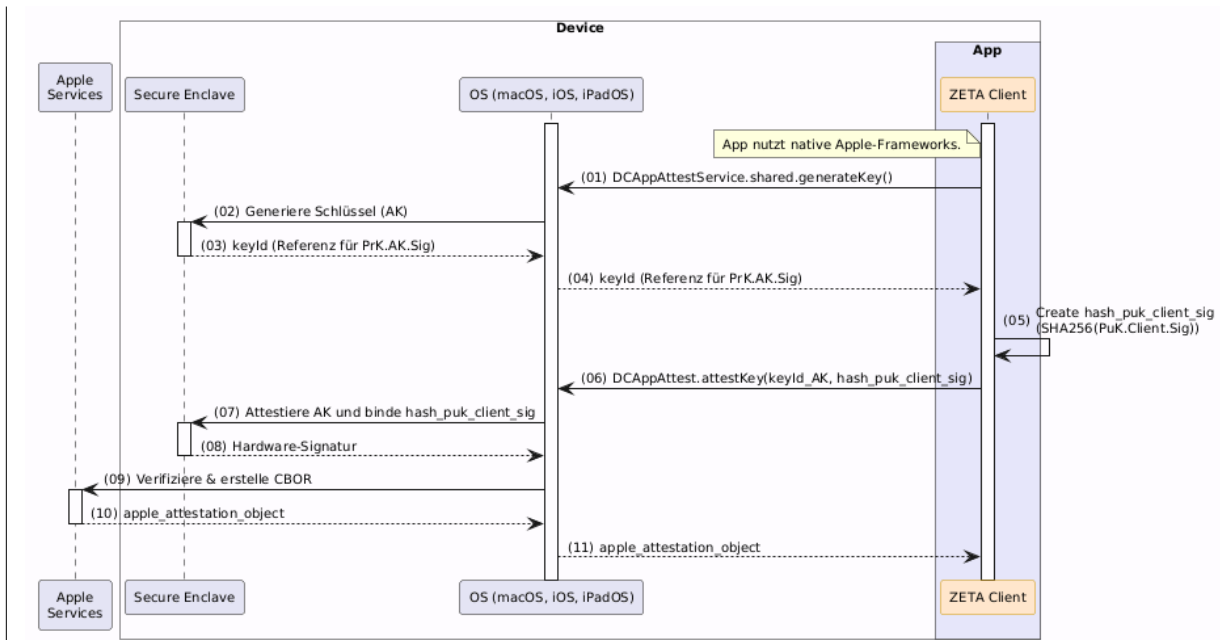


Abbildung 8 Abb-ZETA-SE-Attestation-Key

(01)-(11) Der Ablauf zeigt, wie ein Apple Attestation Object erzeugt wird, das den Hash des PuK.Client.Sig enthält.

5.10.1.5 Client-Registrierung

Das folgende Sequenzdiagramm beschreibt den Registrierungsprozess eines ZETA Clients bei einem ZETA Guard Authorization Server. Der Ablauf unterscheidet sich grundlegend danach, ob bereits ein ZETA Guard Attestation Token vorhanden ist oder nicht. Dies wird durch den äußeren alt-Block dargestellt. Für eine bessere Übersichtlichkeit sind nicht alle erforderlichen Parameter des DCR Requests dargestellt (siehe [dcr-request.yaml] für detaillierte Informationen).

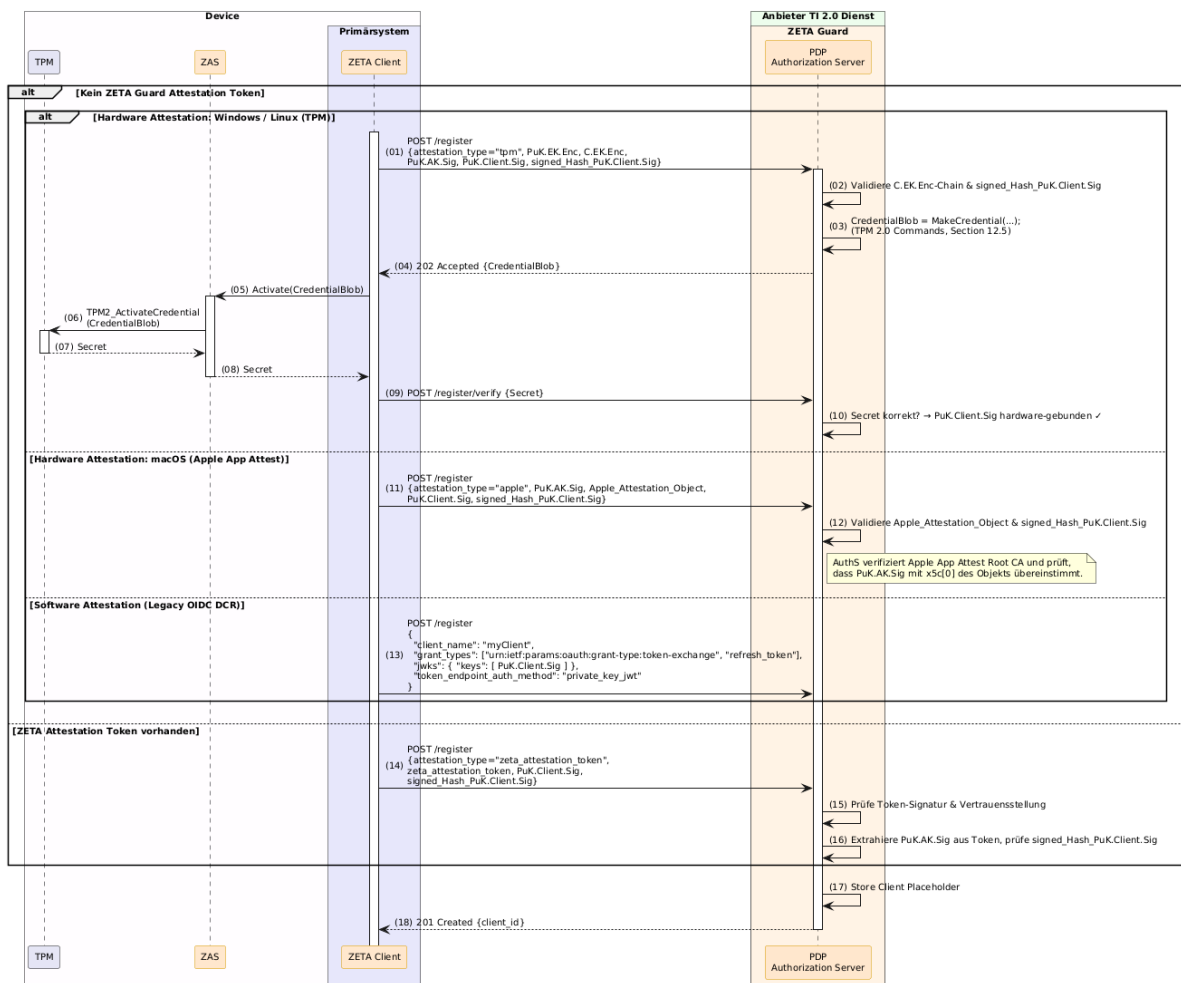


Abbildung 9 Abb-ZETA-DCR-für-stationäre-Clients

Wenn der Client noch kein Token besitzt, muss er seine Vertrauenswürdigkeit durch eine Attestierung nachweisen. Hierfür gibt es drei Unterfälle (innerer alt-Block), abhängig von der genutzten Hardware/Software.

(01)-(10) Hardware-Attestation: Windows / Linux (TPM)

Der ZETA Client sendet einen Request an den PDP Authorization Server mit folgenden Parametern:

Der Authorization Server validiert die Zertifikatskette von `signed_Hash_PuK.Client.Sig` sowie `C.EK.Enc`. Anschließend erzeugt er ein `CredentialBlob` gemäß dem TPM 2.0 Befehl (vgl. TCG TPM Library Part 3, Section 12.5) und antwortet mit `202 Accepted` gemäß [dcr-response-202.yaml].

Der ZETA Client leitet das `CredentialBlob` an den ZAS weiter. Der ZAS führt `TPM2_ActivateCredential` aus und erhält das `Secret`. Das `Secret` wird an den Authorization Server via `POST /register/verify` übermittelt. Nach erfolgreicher Prüfung gelten `C.EK.Enc` und `PuK.AK.Sig` hardware-gebunden.

(11)-(12) Hardware-Attestation: macOS (Secure Enclave)

Der ZETA Client sendet insbesondere mit:

Der Authorization Server validiert das `Apple_Attestation_Object` und `signed_Hash_PuK.Client.Sig`. Dabei wird sichergestellt, dass `Apple_Attestation_Object.x5c[0]` identisch mit dem Schlüssel aus `Apple_Attestation_Object.jwks` des Attestation-Objekts ist und die Apple Root CA-Kette gültig ist.

Der Wert von signed Hash PuK.Client.Sig enthält die Signatur des Hashes von PuK.Client.Sig mit dem *eigenen privaten Client-Schlüssel* (PrK.Client.Sig). Der Hash PuK.Client.Sig ist im Apple Attestation Object enthalten.

(13)-(14) Software-Attestation (Legacy OIDC DCR)

Im Software-Attestation-Pfad sendet der ZETA Client einen Standard-OIDC-DCR-Request:

```
{ "client_name": "<Name des Clients>", "grant_types": [ "urn:ietf:params:oauth:grant-type:token-exchange", "refresh_token" ], "jwks": { "keys": [ PuK.Client.Sig ] }, "token_endpoint_auth_method": "private_key_jwt" }
```

Der Authorization Server legt den Client in der PDP DB an. Erst wenn beim Token Request die Policy Engine dem Authorization Server eine allow: true Decision übergibt, wird der Client als unterstützter Client akzeptiert.

Vorbedingung: ZETA Guard Attestation Token vorhanden

Verfügt der ZETA Client über ein gültiges ZETA Guard Attestation Token aus einer vorherigen Authentifizierung, kann die erneute Registrierung beschleunigt erfolgen.

Beispiel

```
{
  "attestation_type": "zeta_attestation_token",
  "client_name": "ZETA Client App",
  "token_endpoint_auth_method": "private_key_jwt",
  "grant_types": [
    "urn:ietf:params:oauth:grant-type:token-exchange",
    "refresh_token"
  ],
  "redirect_uris": [
    "https://app.example-hersteller.de/cb/zeta-client/oidc",
    "https://app.example-hersteller.de/cb/zeta-client/app"
  ],
  "jwks": {
    "keys": [
      {
        "kty": "EC",
        "crv": "P-256",
        "x": "11qYAYKxCrfVS_7TyWQHOG7hcvPapiMlrwlaaPcHURo",
        "y": "eZXwxvO1hvCY0KucrPfKo7yAyMT6Ajc3N7OkAB6VYy8"
      }
    ]
  },
  "zeta_attestation_token": "eyJ...",
  "signed_hash_puk_client_sig": "MEQ..."
}
```

Der Authorization Server prüft die Token-Signatur, die Vertrauensstellung sowie anhand des im Token enthaltenen .

Abschluss der Registrierung

Nach erfolgreichem Durchlauf eines der obigen Pfade antwortet der Authorization Server mit .

A 29375 -ZETA, Ablauf Dynamic Client Registration

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-DCR-für-stationäre-Clients* unterstützen. [**<=>**]

Hinweis: Ein ZETA Client muss nur die für das eigene Betriebssystem passende Variante des Dynamic Client Registration Flows implementieren.

5.10.1.6 Authentifizierung

Die Authentifizierung für Primärsysteme erfolgt per OAuth Token Exchange mit einem SM(C)-B signiertem Subject Token. Eine erfolgreiche Authentifizierung führt auch zu einem Abschluss der Client-Registrierung, da die Clients grundsätzlich einem Nutzer zugeordnet werden.

ZETA Guard prüft nicht nur, ob der Nutzer für den Zugriff berechtigt ist, sondern auch ob der Client Zugriff erhalten darf. Daher muss der ZETA Client neben dem Subject Token auch ein Client Assertion JWT erstellen und mit dem Schlüssel PrK.Client.Sig signieren. Das Client Assertion JWT dient zur Authentifizierung des Clients gegenüber dem ZETA Guard Authorization Server und enthält im `client_statement` die Attestation Daten des Clients (siehe `[client-assertion-jwt.yaml]`).

5.10.1.6.1 Client Statement mit ZAS und TPM Attestation

Das folgende Sequenzdiagramm beschreibt den Prozess zur Erhebung von Integritäts- und Zustandsdaten (Evidence/Posture) eines Gerätes durch einen ZETA Client mit ZAS und TPM. Dieser Prozess dient dazu, dem ZETA Guard kryptografisch abzusichern, in welchem Sicherheitszustand sich der Client aktuell befindet.

Im Schema `[client-statement.yaml]` sind die Daten, die bei der Attestierung verwendet werden, festgelegt.

Der Ablauf beginnt mit der Einholung einer kryptografischen Herausforderung (Nonce) und teilt sich danach auf, je nachdem, welche Hardware-Sicherheitsfeatures das Gerät bietet.

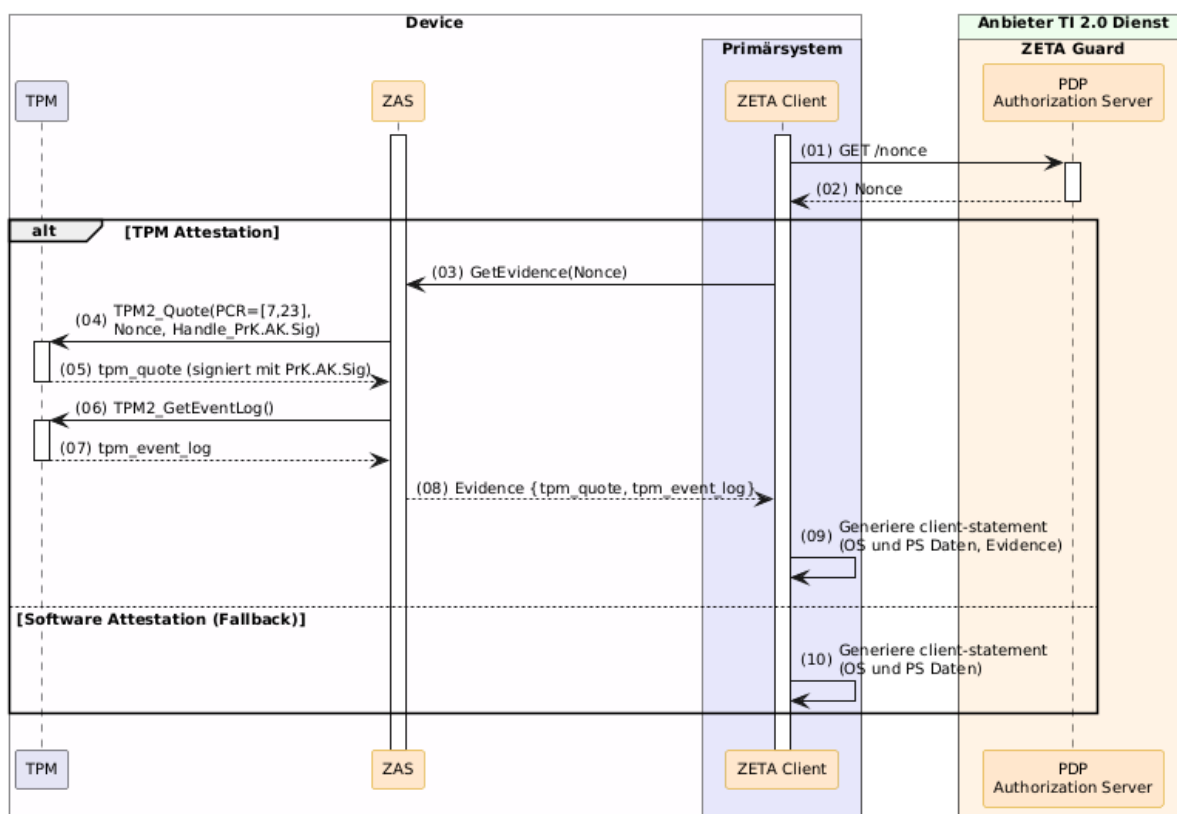


Abbildung 10 Abb-ZETA-Client-Statement-mit-TPM-Attestation

(01) Der ZETA Client sendet einen Request (GET /nonce) an den PDP Authorization Server (ZETA Guard), um eine Nonce (eine einmalige Zufallszahl) anzufordern. Dies schützt den späteren Prozess vor Replay-Attacken.

(02) Der Server generiert die Nonce und sendet sie an den Client zurück.

(03) Der ZETA Client beauftragt den ZAS mit der Erhebung der Nachweise (GetEvidence) und übergibt dabei die erhaltene Nonce.

(04)-(05) Der ZAS fordert vom TPM ein sogenanntes Quote an (TPM2_Quote). Dabei fordert er die Signierung bestimmter Platform Configuration Registers (hier PCR=[7,23], welche typischerweise Boot-State und Application-State repräsentieren) an. In diesen Aufruf fließen die Nonce (zur Sicherstellung der Aktualität) und das Handle des privaten Attestation Keys (Handle_PrK.AK.Sig) ein.

(06)-(07) Zusätzlich ruft der ZAS das TCG Event Log (TPM2_GetEventLog()) vom System ab. Das Log enthält die detaillierte Historie, wie die Werte in den PCRs zustande gekommen sind.

(08) Der ZAS bündelt das Quote und das Event Log zu einer Evidence und sendet diese an den ZETA Client zurück.

(09) Der ZETA Client generiert das client_statement, in dem Daten über das Primärsystem, das Betriebssystem und die Evidence Daten enthalten sind.

(10) Software-Attestation: Der ZETA Client sammelt die OS- und Primärsystem-Daten selbst und packt sie in ein client_statement. In diesem Fall findet keine kryptografische Signatur durch ein TPM statt.

5.10.1.6.2 Client Statement mit Apple AppAttest

Das folgende Sequenzdiagramm beschreibt detailliert den Prozess zur Erhebung von Integritäts- und Zustandsdaten (Evidence) auf einem macOS-Gerät durch einen ZETA Client. Es zeigt den korrigierten, architektonisch sauberen Ablauf unter Verwendung der Apple App Attest (bzw. DeviceCheck) API in Verbindung mit der Secure Enclave.

Im Schema [client-statement.yaml] sind die Daten, die bei der Attestierung verwendet werden, festgelegt.

Der Ablauf ist in eine initiale Phase und eine Fallunterscheidung (Attestierungs-Methode) gegliedert.

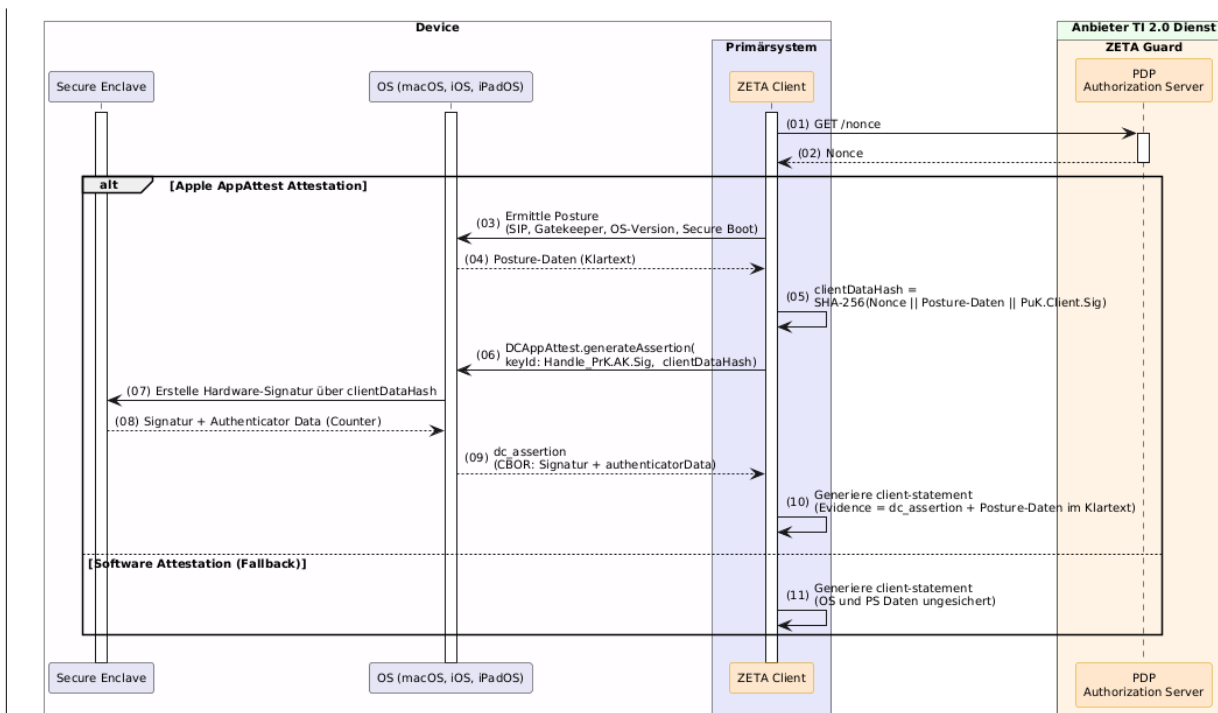


Abbildung 11 Abb-ZETA-Client-Statement-mit-Apple-AppAttest

Vorbereitungsphase: Nonce-Bezug

Um Replay-Angriffe zu verhindern und die Frische des Integritätsnachweises zu garantieren, benötigt der Client eine kryptografische Challenge vom Server. (01) GET /nonce: Der ZETA Client fragt eine neue Nonce beim ZETA Guard (Authorization Server) an.

(02) Nonce: Der ZETA Guard generiert eine zufällige Nonce und liefert diese zurück.

Primärer Pfad: Apple AppAttest Attestation (Hardwaregestützt)

Dieser Pfad wird durchlaufen, wenn das Gerät über eine Secure Enclave verfügt und das App Attest Framework bei der initialen Registrierung (DCR) erfolgreich eingerichtet wurde.

(03) Ermittle Posture: Der ZETA Client fragt über native Apple-APIs sicherheitsrelevante Systemparameter ab. Dazu gehören beispielsweise der Status der System Integrity Protection (SIP), Gatekeeper-Einstellungen, die exakte OS-Version und der Status des Secure Boots.

(04) Posture-Daten (Klartext): Das Betriebssystem übergibt diese Systemdaten im Klartext an den ZETA Client.

(05) Hash-Berechnung (clientDataHash): Dies ist der zentrale kryptografische Sicherheitsanker. Der ZETA Client berechnet einen SHA-256 Hash aus der Verkettung der serverseitigen Nonce, den ermittelten Posture-Daten und seinem öffentlichen Client-Schlüssel (PuK.Client.Sig).

(Architektur-Hinweis: Dadurch werden die Frische des Requests, der Systemzustand und die Identität des Clients untrennbar aneinander gebunden).

(06) Assertion anfordern: Der Client ruft die API DCAppAttest.generateAssertion auf. Er übergibt das Handle seines hardwaregebundenen Attestation Keys (Handle_PrK.AK.Sig) sowie den soeben berechneten clientDataHash.

(07) Hardware-Signatur: Das Betriebssystem leitet den Hash an die Secure Enclave weiter. Die Secure Enclave signiert diesen Hash mit dem privaten Attestation Key.

(08) Rückgabe der Signatur & Metadaten: Die Secure Enclave gibt die Signatur sowie zusätzliche Authenticator-Daten (insbesondere einen monoton steigenden Counter zum

Schutz vor Klon-Angriffen) an das OS zurück.
(09) dc_assertion (CBOR): Das OS verpackt die Signatur und die Authenticator-Daten in ein standardisiertes CBOR-Objekt (dc_assertion) und reicht es an den ZETA Client weiter.
(10) Generiere client_statement: Der ZETA Client baut das finale client_statement (Evidence) für den Request an den ZETA Guard zusammen. Es besteht aus der kryptografischen dc_assertion und den Posture-Daten im Klartext.
(Der ZETA Guard kann später den Hash aus der ihm bekannten Nonce, den Klartext-Posture-Daten und dem registrierten Client-Key selbst berechnen und die Hardware-Signatur in der dc_assertion validieren).

Alternativer Pfad: Software Attestation (Fallback)

Dieser Pfad dient als Rückfallebene, falls keine Hardware-Unterstützung vorliegt.
(11) Generiere client_statement (Software): Der ZETA Client sammelt die OS- und Primärsystem-Daten (PS-Daten) und fügt sie ungesichert in das Statement ein. Da hierbei keine Signatur durch eine Secure Enclave stattfindet, wird die Policy Engine des ZETA Guards diesen Request mit einem entsprechend niedrigen Trust-Score bewerten.

5.10.1.6.3 Token Exchange mit Attestation

Das folgende Sequenzdiagramm beschreibt den zentralen OAuth Token Exchange Authentifizierungs- und Autorisierungsprozess eines ZETA Clients am ZETA Guard Authorization Server. Der Prozess verknüpft die Nutzer-/Institutions-Authentifizierung (mittels SM(C)-B) mit der Client-Authentifizierung (Client Assertion & Evidence) und bindet die resultierenden Tokens kryptografisch an die aktuelle Sitzung (DPoP).

Das Subject Token zur Authentifizierung der Institution ist nach Schema [subject-token-smb.yaml] festgelegt. Die Client Assertion zur Authentifizierung des Clients ist in [client-assertion-jwt.yaml] festgelegt.

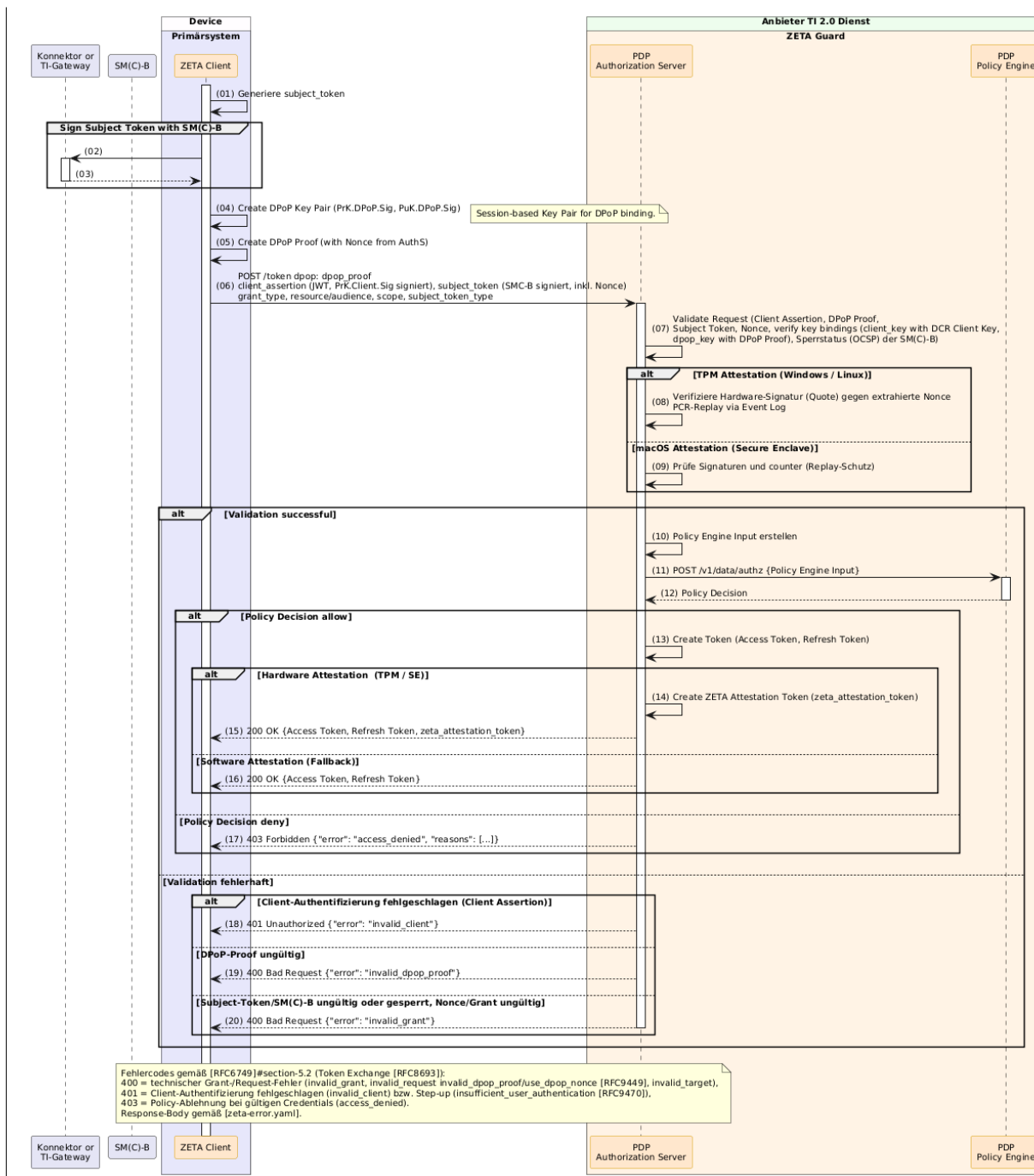


Abbildung 12 Abb-ZETA-Token-Exchange-mit-Attestation

Das Diagramm ist in mehrere logische Phasen und (verschachtelte) Fallunterscheidungen (alt-Blöcke) unterteilt.

Phase 1: Vorbereitung der Tokens und Schlüssel (Schritte 01 - 05)

(01 - 03) Subject Token Erstellung: Der ZETA Client generiert zunächst ein subject_token. Um die Identität des Nutzers bzw. der Institution (z.B. einer Arztpraxis in der Telematikinfrastruktur) nachzuweisen, wird dieses Token an die lokale Smartcard/Sicherheitsmodul (SM(C)-B) gesendet, dort signiert und an den Client zurückgegeben.

(04) DPoP Key Pair: Der Client generiert ein temporäres, sitzungsbasiertes Schlüsselpaar (PrK.DPoP.Sig, PuK.DPoP.Sig). Dieses dient dem Demonstrating Proof-of-Possession

(DPoP) – einem Mechanismus, der verhindert, dass gestohlene Access Tokens von Angreifern genutzt werden können.

(05) DPoP Proof: Der Client erstellt mit dem privaten DPoP-Schlüssel und einer vom Server stammenden Nonce einen DPoP Proof.

Phase 2: Der Token-Request (Schritt 06)

(06) POST /token: Der ZETA Client sendet die eigentliche Anfrage an den PDP Authorization Server. Dieser Request enthält ein umfassendes Sicherheitspaket:

- Im Body: den Parameter resource [RFC8707] (Endpunkt-URL der Zielressource aus [opr-well-known.yaml]) sowie scope. Aus Gründen der Abwärtskompatibilität kann stattdessen der Parameter audience verwendet werden (siehe 5.4.2.5).
- Im Header (dpop): Den in (05) erstellten dpop_proof.
- client_assertion: Ein JWT, signiert mit dem privaten Client-Schlüssel (PrK.Client.Sig), um den Client selbst zu authentifizieren.
- subject_token: Das von der SMC-B signierte Token inkl. Nonce aus Schritt (03).
- [Evidence]: Optional/Bedingt mitgeschickte Integritäts- und Zustandsdaten des Geräts (wie in den vorherigen Diagrammen ermittelt).

Phase 3: Validierung durch den ZETA Guard (Schritte 07 - 09)

(07) Basis-Validierung: Der Authorization Server prüft die Client Assertion, den DPoP Proof, das Subject Token, die Nonces und gleicht ab, ob die Schlüssel zu der vorherigen Registrierung (DCR) passen.

Hardware-Attestierungsprüfung (Innerer alt-Block): Falls Evidence mitgeliefert wurde, wird diese hardware-spezifisch geprüft:

(08) [TPM Attestation (Windows / Linux)]: Die TPM-Hardware-Signatur (Quote) wird verifiziert und die Systemzustände (PCRs) werden anhand des Event Logs nachvollzogen.

(09) [macOS Attestation (Secure Enclave)]: Die Apple App Attest Signaturen werden geprüft und der Counter ausgelesen, um Replay-Attacken (Wiederholungsangriffe) zu verhindern.

Phase 4: Policy-Entscheidung und Response (Schritte 10 - 18)

Ab hier spaltet sich der Ablauf je nach Erfolg der Validierung auf (äußerer alt-Block):

Fall A: [Validation successful] (Schritte 10 - 16)

Wenn die technische und kryptografische Validierung (07-09) erfolgreich war, muss inhaltlich über den Zugriff entschieden werden.

(10 - 11): Der Authorization Server bereitet die validierten Daten auf (Policy Engine Input) und sendet sie via POST /v1/data/authz an die PDP Policy Engine.

(12): Die Policy Engine evaluiert die Zugriffsregeln und liefert eine Entscheidung (Policy Decision) zurück.

Unterfall A1: [Policy Decision allow] (Schritte 13 - 16)

Die Policy Engine erlaubt den Zugriff.

(13): Der Server erstellt die Standard-OIDC-Tokens (Access Token, Refresh Token). Diese sind durch den Request an den DPoP-Schlüssel gebunden. Der aud-Claim des Access Tokens wird dabei aus der Policy Engine Decision übernommen (Contract v2, ver: 2); bei einem Legacy-Request über audience wird der Wert verbatim übernommen (ver: 1).

Unterscheidung nach Attestierungs-Level (Innerster alt-Block):

(14 - 15) [Hardware Attestation (TPM / SE)]: Wenn das Gerät seinen Zustand erfolgreich durch Hardware (TPM/Secure Enclave) bewiesen hat, generiert der Server zusätzlich ein ZETA Attestation Token (zeta_attestation_token). Er antwortet mit 200 OK und liefert Access Token, Refresh Token und das ZETA attestation Token an den Client.

(16) [Software Attestation (Fallback)]: Wenn das Gerät nur auf Software-Basis attestiert wurde, erhält der Client zwar Zugriff (200 OK), bekommt aber nur das Access Token und das Refresh Token (kein Attestation Token).

Unterfall A2: [Policy Decision deny] (Schritt 17)

(17): Wenn die Policy Engine den Zugriff verweigert (z.B. weil der Gerätezustand nicht den Sicherheitsrichtlinien entspricht, obwohl die Signaturen gültig sind), antwortet der Server dem Client mit 403 Forbidden (error: access_denied).

Fall B: [Validation fehlerhaft] (Schritt 18)

(18)-(20): Wenn bereits die technische Prüfung in Schritt (07) bis (09) fehlschlägt (z.B. ungültige Signatur der SMC-B, DPoP-Proof falsch, Attestation manipuliert), bricht der Server sofort ab und antwortet dem ZETA Client mit

- 400 Bad Request (invalid_grant bei ungültiger/gesperrter SMC-B bzw. abgelaufenem Grant, invalid_dpop_proof bei fehlerhaftem DPoP-Proof); Client-Authentifizierungsfehler
- 401 (invalid_client).

Die Policy Engine wird in diesem Fall nicht befragt.

A_29376 -ZETA, Ablauf Token Exchange mit Attestation

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-Token-Exchange-mit-Attestation* unterstützen. [<=]

5.10.1.6.4 Token Request mit grant_type=refresh_token

Das folgende Sequenzdiagramm beschreibt den Refresh-Token-Prozess (Token Renewal) eines ZETA Clients am ZETA Guard Authorization Server. Es zeigt, wie ein Client, der bereits eine Sitzung etabliert hat, mit einem gültigen Refresh Token neue Zugriffs-Token (Access Tokens) anfordert, ohne den vollständigen (und zeitaufwendigen) Hardware-Attestierungs- und Smartcard-Signaturprozess erneut durchlaufen zu müssen.

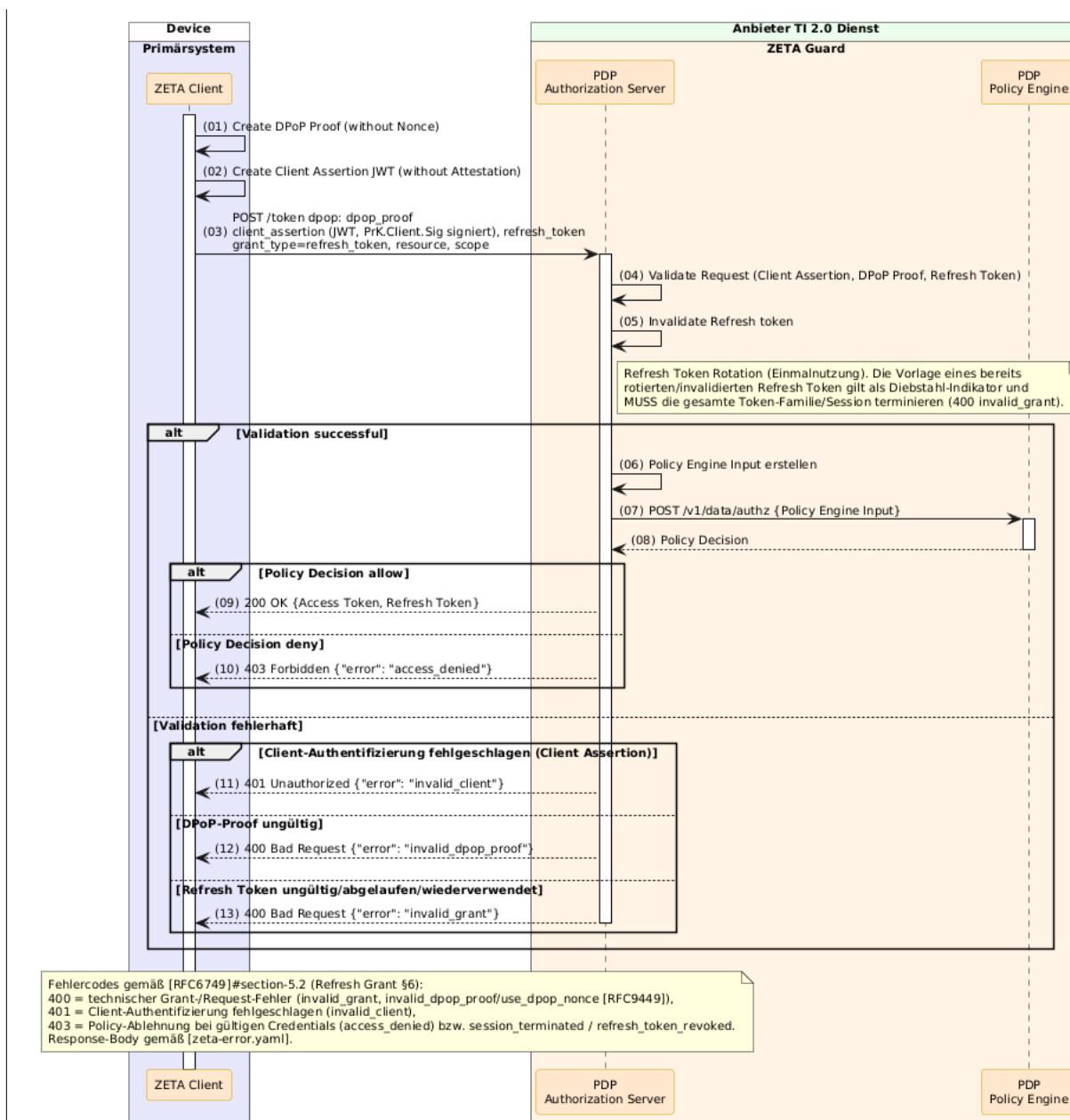


Abbildung 13 Abb-ZETA-Token-Request-mit-Refresh-Token

Das Diagramm gliedert sich in die Vorbereitung der Anfrage, die Validierung durch den Server und die finale Autorisierungsentscheidung.

Phase 1: Vorbereitung und Token-Request

(01) DPoP Proof: Der ZETA Client erstellt einen DPoP Proof (Demonstrating Proof-of-Possession). Bemerkenswert ist hier der Hinweis (without Nonce): Um einen zusätzlichen Netzwerk-Roundtrip zu sparen, wird beim Refresh oft zunächst auf eine frische Nonce des Servers verzichtet.

(02) Client Assertion: Der Client erstellt ein JWT (JSON Web Token) zur eigenen Authentifizierung (Client Assertion). Hier geschieht dies explizit (without Attestation), da der Gerätezustand bei einem bloßen Refresh nicht zwingend neu aufwendig hardware-attestiert wird.

(03) POST /token: Der Client sendet die Anfrage an den PDP Authorization Server. Dieser

Request enthält im Header den `dpop_proof` und in der Payload die signierte `client_assertion` sowie das bisherige `refresh_token`.

Phase 2: Validierung durch den ZETA Guard

(04) Basis-Validierung: Der Server prüft die kryptografische Gültigkeit der Client Assertion, des DPoP Proofs und verifiziert, ob das eingereichte Refresh Token gültig und dem Client zugeordnet ist.

(05) Token Invalidation: Der Server invalidiert das eingereichte Refresh Token. Dies implementiert das Konzept der Refresh Token Rotation – ein Refresh Token kann nur exakt einmal benutzt werden. Bei einem erneuten Versuch würde dies als Diebstahl-Indikator gewertet.

Phase 3: Policy-Entscheidung und Response

Der weitere Ablauf spaltet sich auf (äußerer alt-Block), je nachdem, ob die technische Validierung in Phase 2 erfolgreich war.

Fall A: [Validation successful]

Wenn die Token gültig sind und die SMC-B nicht gesperrt ist, wird die inhaltliche Autorisierung geprüft.

(06 - 07): Der Authorization Server bereitet die Daten auf (Policy Engine Input) und befragt via POST `/v1/data/authz` die PDP Policy Engine.

(08): Die Policy Engine liefert ihre Entscheidung (Policy Decision) zurück.

Unterfall A1: [Policy Decision allow]

(09): Die Policy Engine erlaubt die Verlängerung der Sitzung. Der Server antwortet dem Client mit 200 OK und stellt ein brandneues Paar aus Access Token und einem neuen Refresh Token aus.

Unterfall A2: [Policy Decision deny]

(10): Verweigert die Policy Engine den Zugriff (z.B. weil sich übergeordnete Zugriffsregeln geändert haben), antwortet der Server mit 403 (`access_denied`). Der Client muss sich dann ggf. komplett neu authentifizieren.

Fall B: [Validation fehlerhaft]

(11)-(13): Schlägt bereits die technische Prüfung (Schritte 04-06) fehl – beispielsweise weil das Refresh Token abgelaufen ist, der DPoP-Proof falsch ist oder die SMC-B zwischenzeitlich gesperrt wurde – bricht der Server den Vorgang ab. Die Policy Engine wird nicht befragt und der Client erhält sofort ein

- 400 (`invalid_grant` / `invalid_dpop_proof`); reuse eines rotierten Refresh Token
- 400 `invalid_grant` und Terminierung der Token-Familie. Client-Auth-Fehler
- 401 `invalid_client`.

Die Sitzung ist damit beendet.

A 29377 -ZETA, Ablauf Token Exchange mit Refresh Token

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-Token-Request-mit-Refresh-Token* unterstützen. [`<=`]

5.10.1.7 Zugriff auf den Resource Server

Voraussetzung für den Zugriff auf eine geschützte Ressource (Resource Server) durch einen Client in der ZETA-Architektur ist, dass der vorgeschaltete Authentifizierungs- und Autorisierungsprozess bereits erfolgreich war und der ZETA Client über ein gültiges Access Token verfügt.

Durch die Auswertung des OAuth Protected Resource Well-known JSON Dokuments kann der ZETA Client erkennen, ob eine ZETA/ASL Verbindung aufgebaut werden muss und welche Token benötigt werden.

5.10.1.7.1 Zugriff auf den Resource Server mit ZETA/ASL

Das folgende Diagramm zeigt, wie eine Anfrage vom Fach-Client über den ZETA Client zum Resource Server geleitet wird.

Das Hauptmerkmal dieses Diagramms ist die Nutzung des ZETA/ASL (Application Security Layer), welcher eine zusätzliche Verschlüsselungsschicht auf Applikationsebene darstellt. Das Diagramm zeigt in einem großen alt-Block zwei unterschiedliche Architektur-Ansätze, je nachdem, wo dieser verschlüsselte Tunnel endet (terminiert). Je nach Einstellung im OAuth Protected Resource Well-known JSON Dokument wird der Tunnel entweder am ZETA Guard HTTP Proxy oder am Resource Server terminiert.

Vorbedingung: Der ZETA Client hat bereits erfolgreich den Authentifizierungsprozess durchlaufen und besitzt ein gültiges Access Token für den Resource Server und ggf. für den ASL Tunnel.

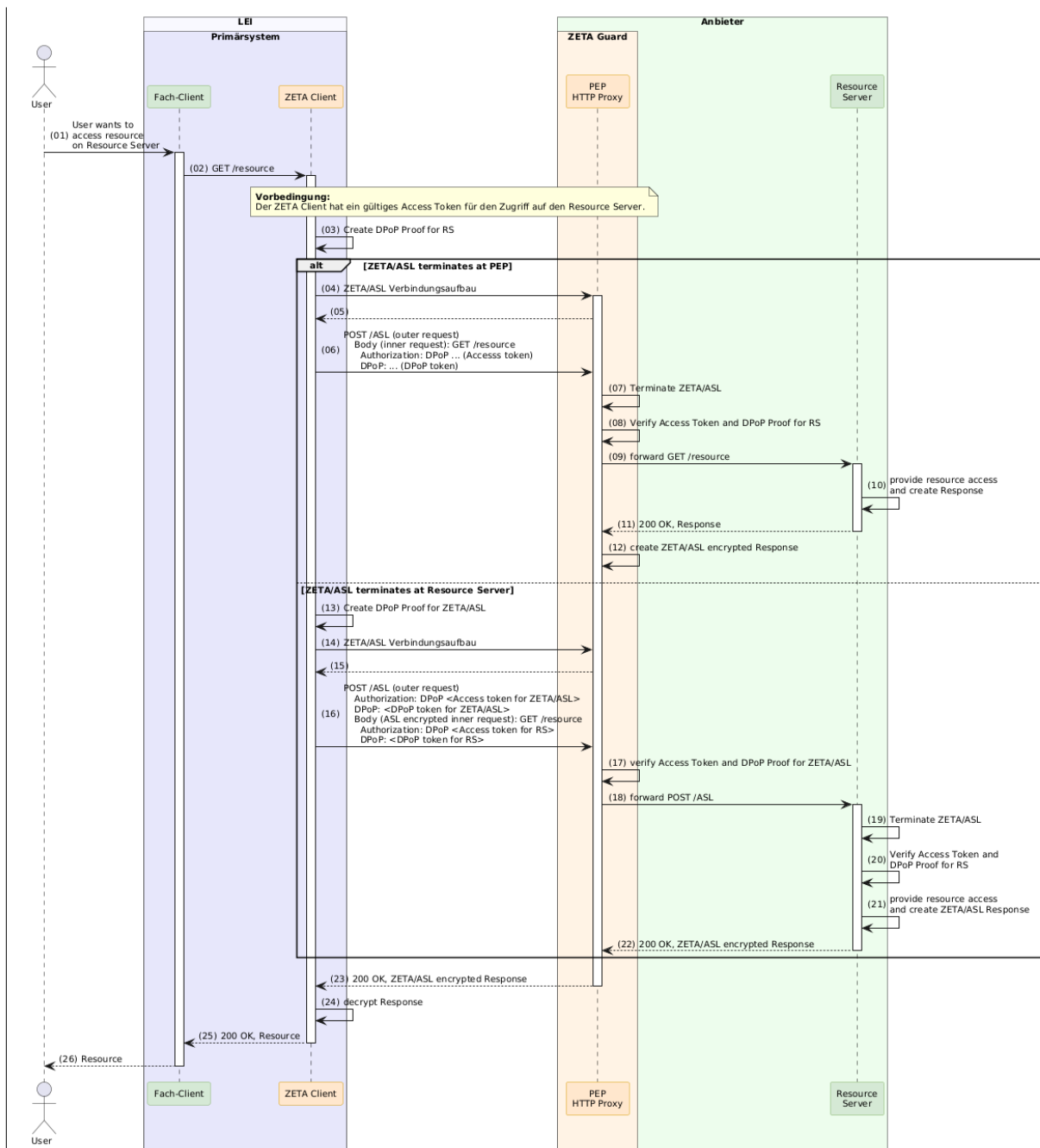


Abbildung 14 Abb-ZETA-Zugriff-auf-RS-mit-ASL

(01)-(02): Der Nutzer (User) möchte auf eine Ressource zugreifen. Das Primärsystem (Fach-Client) sendet eine normale, unverschlüsselte HTTP-Anfrage (GET /resource) an den lokalen ZETA Client.

(03): Der ZETA Client erstellt einen kryptografischen DPoP-Proof (Demonstrating Proof-of-Possession), der an die URL des Resource Servers gebunden ist, um das Access Token vor Diebstahl/Replay zu schützen.

Fall A: [ZETA/ASL terminates at PEP]

In diesem Szenario dient ASL als sicherer Tunnel zwischen dem lokalen Client und dem Netzübergang (PEP) des Anbieters. Im internen Netz des Anbieters (vom PEP zum Resource Server) fließen die Daten nur mit TLS verschlüsselt.

(04)-(05): Es wird ein ZETA/ASL-Verbindungsaufbau zwischen dem ZETA Client und dem PEP HTTP Proxy initiiert.

(06): Der ZETA Client verpackt die eigentliche Anfrage. Er sendet einen "äußeren" Request (POST /ASL) an den PEP. Im verschlüsselten Body (inner request) liegt der eigentliche GET /resource-Aufruf, zusammen mit dem Access Token und dem in (03) generierten DPoP-Proof.

(07): Der PEP terminiert den ASL-Tunnel. Das bedeutet, er entschlüsselt den Body.

(08): Der PEP verifiziert nun das entpackte Access Token und den DPoP-Proof.

(09): Ist die Prüfung erfolgreich, leitet der PEP den ursprünglichen Klartext-Request (GET /resource) an den eigentlichen Resource Server weiter.

(10 - 11): Der Resource Server verarbeitet die Anfrage und sendet die Response an den PEP zurück.

(12): Der PEP verschlüsselt diese Antwort wieder mit ASL, um sie für den sicheren Rückweg über das Internet vorzubereiten.

Fall B: [ZETA/ASL terminates at Resource Server]

In diesem hochsicheren Szenario wird der Payload Ende-zu-Ende verschlüsselt. Der PEP agiert nur als Router und kann den Inhalt der Fachnachricht nicht mitlesen.

(13): Der ZETA Client muss hier einen zweiten DPoP-Proof generieren. Einer ist für den Resource Server (für die Fachdaten), der neue ist für den PEP (um überhaupt den ASL-Endpunkt nutzen zu dürfen).

(14)-(15): Ein ZETA/ASL-Verbindungsaufbau findet logisch zwischen dem ZETA Client und dem Resource Server statt.

(16): Der Request (POST /ASL) ist komplexer verschachtelt:

Auf der äußeren Ebene (für den PEP sichtbar) liegen ein Access Token und ein DPoP-Proof, die nur für den Zugang zum ASL-Endpunkt berechtigen.

Der verschlüsselte Body enthält den eigentlichen Request (GET /resource) samt eigenen Autorisierungsdaten (Access Token und DPoP-Proof für den RS).

(17)-(18): Der PEP prüft nur die äußeren Header. Da er den Body nicht entschlüsseln kann, leitet er den POST /ASL Request ungeschoren an den Resource Server weiter.

(19): Erst der Resource Server terminiert den ASL-Tunnel und entschlüsselt die Nachricht.

(20): Der Resource Server prüft nun die inneren Header (Access Token und DPoP-Proof für die Ressource).

(21): Er verarbeitet die Anfrage, erzeugt die Antwort und verschlüsselt diese direkt wieder in eine ASL-Response.

(22)-(23): Unabhängig davon, ob der PEP oder der Resource Server die ASL-Verschlüsselung der Antwort übernommen hat, empfängt der ZETA Client nun die verschlüsselte ASL-Response.

(24): Der ZETA Client entschlüsselt die Antwort lokal.

(25): Er reicht die Response im Klartext an den Fach-Client weiter.

(26): Der Fach-Client stellt dem User die angeforderten Daten dar.

A_29380 -ZETA, Ablauf Zugriff auf den RS mit ZETA/ASL

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-Zugriff-auf-RS-mit-ASL* unterstützen. [<=]

5.10.1.7.2 Zugriff auf den Resource Server ohne ZETA/ASL

Das folgende Sequenzdiagramm beschreibt den direkten Zugriff auf eine geschützte Ressource (Resource Server) durch einen Client in der ZETA-Architektur.

Er zeigt das Routing über den PEP, bei dem die Absicherung primär über tokenbasierte HTTP-Header (Authorization, DPoP) erfolgt, ohne dass ein zusätzlicher ASL-Verschlüsselungstunnel aufgebaut wird.

Vorbedingung: Der ZETA Client hat im Vorfeld bereits erfolgreich einen Authentifizierungsprozess durchlaufen und verfügt über ein gültiges Access Token für den entsprechenden Resource Server.

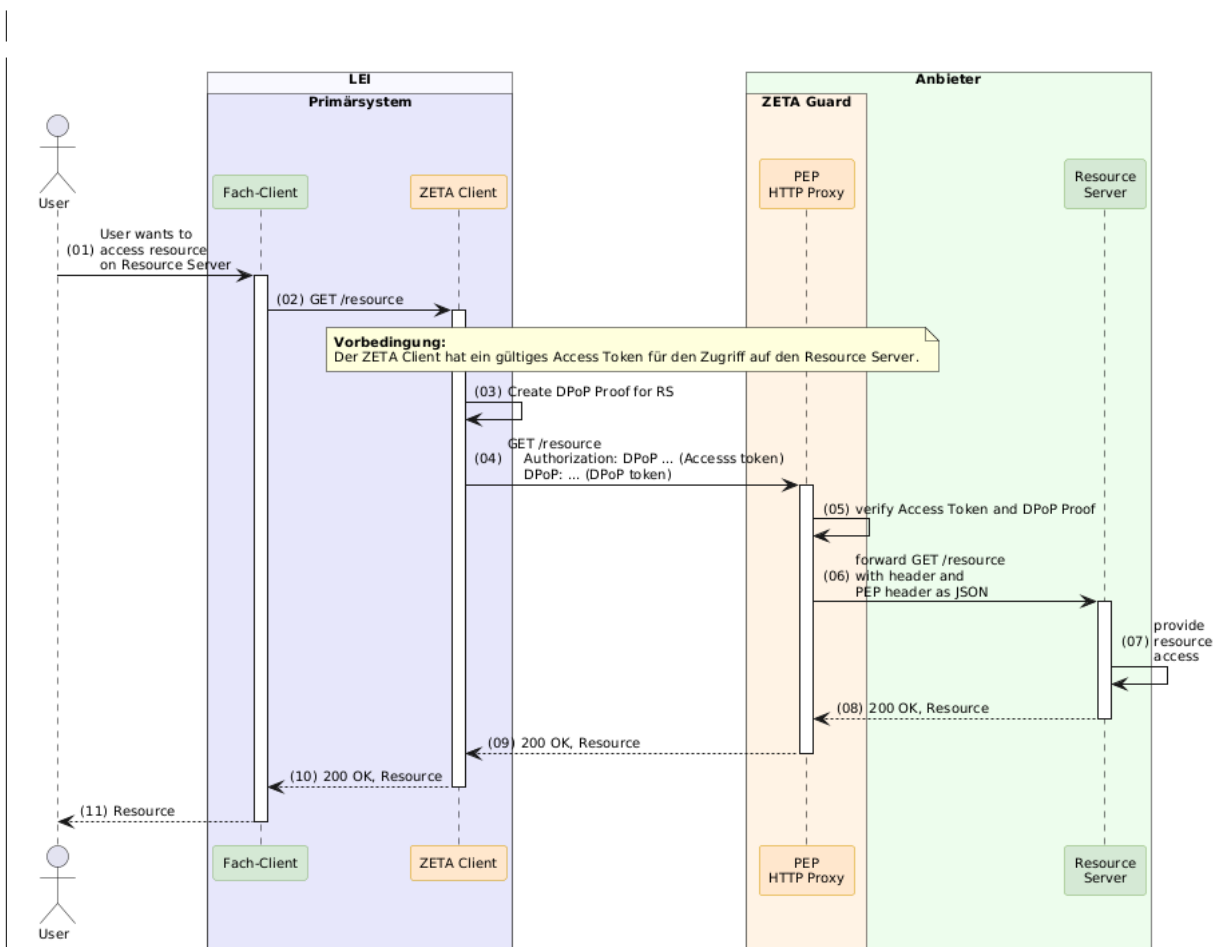


Abbildung 15 Abb-ZETA-Zugriff-auf-RS-ohne-ASL

(01): Der Endnutzer (User) möchte auf eine bestimmte Ressource zugreifen und interagiert dafür mit seiner Fachanwendung (Fach-Client, z. B. ein Praxisverwaltungssystem).

(02): Der Fach-Client sendet eine reguläre, ungesicherte HTTP-Anfrage (GET /resource) an den lokal installierten ZETA Client. Der Fach-Client selbst muss sich nicht um die komplexe ZETA-Kryptografie kümmern.

(03): Der ZETA Client bereitet die Anfrage vor. Er erstellt einen DPoP Proof (Demonstrating Proof-of-Possession). Dies ist eine kryptografische Signatur, die beweist, dass der Client im Besitz des privaten Schlüssels ist, an den das Access Token gebunden wurde. Dieser Proof wird speziell für die Ziel-URL des Resource Servers (RS) generiert (Schutz vor Replay-Attacks).

(04): Der ZETA Client leitet den GET /resource Request an den PEP HTTP Proxy (den ZETA Guard des Anbieters) weiter. Dabei fügt er zwei Sicherheits-Header hinzu:

- Authorization: DPoP ...: Enthält das eigentliche Access Token.
- DPoP: ...: Enthält den in Schritt 03 generierten DPoP-Token (Proof).

(05): Der PEP HTTP Proxy empfängt die Anfrage und verifiziert die Gültigkeit des Access Tokens und des DPoP Proofs.

(06): Verläuft die Prüfung erfolgreich, leitet der PEP den Request in das interne Netz an den eigentlichen Resource Server weiter. Um dem Resource Server mitzuteilen, wer anfragt, fügt der PEP verifizierte Kontextdaten (Identität, Berechtigungen) in Form von ZETA headers as JSON an die Anfrage an.

(07): Der Resource Server verarbeitet die nun vollständig autorisierte Anfrage und stellt

die angeforderten Daten bereit (provide resource access).
(08): Der Resource Server antwortet dem PEP mit einer Response.
(09): Der PEP HTTP Proxy leitet diese Antwort transparent an den ZETA Client zurück.
(10): Der ZETA Client reicht die Response an den Fach-Client durch.
(11): Der Fach-Client nimmt die Daten entgegen und präsentiert die Ressource dem User.

A 29381 -ZETA, Ablauf Zugriff auf den RS ohne ZETA/ASL

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-Zugriff-auf-RS-ohne-ASL* unterstützen. [\leq]

5.10.2 Abläufe für mobile Clients

In diesem Unterkapitel werden die Kommunikations- und Attestierungsprozesse für mobile ZETA Clients (z. B. auf Basis von Android oder iOS) detailliert. Aufgrund der Architektur mobiler Betriebssysteme werden für den Integritätsnachweis plattformspezifische Mechanismen wie die Android Key and ID Attestation oder die Secure Enclave von Apple genutzt. Als Fallback-Lösung ist eine Software-Attestierung vorgesehen, falls hardwarebasierte Nachweise nicht erbracht werden können. Im Unterschied zu stationären Systemen erfordert das mobile Umfeld in der Regel eine interaktive Nutzerauthentisierung. Daher erfolgt die Autorisierung hier standardisiert über OpenID Connect (OIDC) in Verbindung mit dem OAuth Authorization Code Flow.

Im Unterschied zu stationären Clients basieren mobile Clients auf plattformintegrierten Sicherheitsmechanismen und erfordern in der Regel eine interaktive Nutzerauthentisierung. Während bei stationären Clients TPM/ZAS im Fokus steht, erfolgt der Integritätsnachweis bei mobilen Clients über plattformabhängige Attestierungsverfahren. Die Autorisierung erfolgt über den OIDC Authorization Code Flow in Kombination mit OAuth 2.0.

Der Lebenszyklus eines mobilen Clients gliedert sich analog zu stationären Clients in folgende Phasen:

4. Installation und Initialisierung
5. Schlüsselgenerierung
6. Service Discovery
7. Key Preparation und Attestation
8. Dynamic Client Registration (DCR)
9. Authentifizierung (OIDC Flow)
10. Token Exchange und Zugriff
11. Session-Erneuerung

Hardware-Attestation SOLL bevorzugt eingesetzt werden und Software-Attestation dient als Fallback-Mechanismus.

5.10.2.1 Initialisierung und Schlüsselgenerierung

Nach der Installation der mobilen Anwendung wird beim ersten Start ein initialer Satz kryptografischer Schlüssel erzeugt, der als langlebige Identität der Client-Instanz dient. Im Gegensatz zu stationären Clients erfolgt die Schlüsselgenerierung vollständig innerhalb der durch das mobile Betriebssystem bereitgestellten Sicherheitsmechanismen.

5.10.2.1.1 Android TEE oder Strongbox

Das Sequenzdiagramm zeigt den grundlegenden Prozess der kryptografischen Schlüsselgenerierung und Hardware-Attestierung auf einem Android-Gerät. Es veranschaulicht das Zusammenspiel zwischen der App (mit ZETA Client), dem Betriebssystem (Android OS) und dem isolierten Hardware-Sicherheitsmodul (TEE / StrongBox). Es nutzt ein Zwei-Schlüssel-Modell (beide in der Hardware verankert) und integriert die externe Google Play Integrity API.

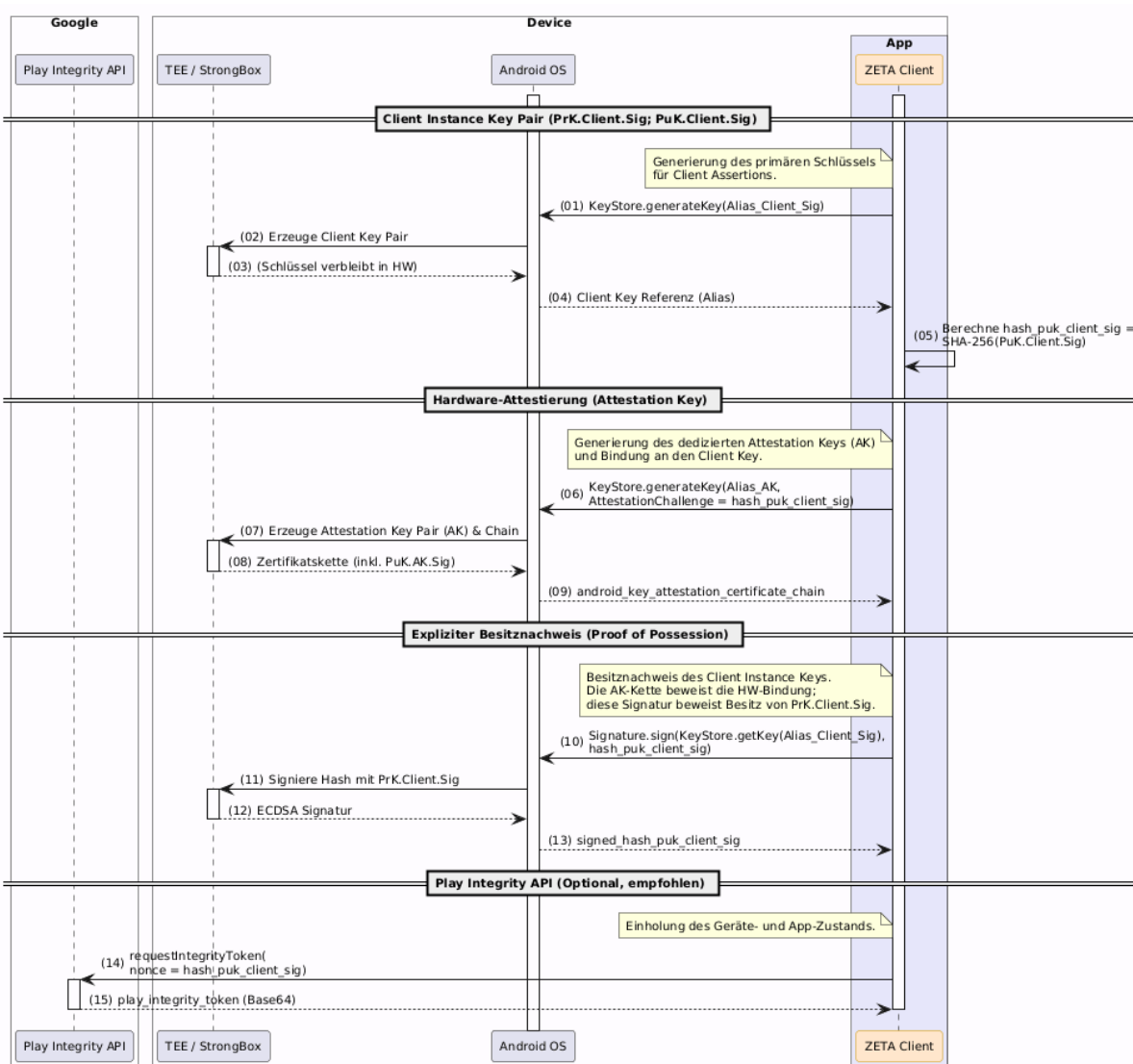


Abbildung 16 Abb-ZETA-Schlüsselgenerierung-Android

1. Generierung des Client Instance Key Pairs

In dieser Phase erzeugt die App den primären kryptografischen Schlüssel, der später für die fortlaufende Authentisierung (Client Assertions) genutzt wird.

(01) Schlüsselgenerierung anfordern: Der ZETA Client ruft die native Android-API `KeyStore.generateKey` auf, um ein asymmetrisches Schlüsselpaar (Alias: `Alias_Client_Sig`) zu erzeugen.

(02) Erzeugung in der Hardware: Das Android OS leitet den Befehl an das Hardware-Sicherheitsmodul (TEE oder StrongBox) weiter.

(03) Schlüssel verbleibt in HW: Die StrongBox erzeugt das Schlüsselpaar (`PrK.Client.Sig`, `PuK.Client.Sig`). Der private Schlüssel (`PrK.Client.Sig`) ist hardwaregebunden und kann

nicht in den Arbeitsspeicher des Betriebssystems oder der App ausgelesen werden.
(04) Rückgabe der Referenz: Das Android OS gibt dem ZETA Client lediglich eine Referenz (den Alias) auf diesen Schlüssel zurück.

(05) Hash-Berechnung: Der ZETA Client berechnet den SHA-256 Hash des öffentlichen Schlüssels (hash_puk_client_sig = SHA-256(PuK.Client.Sig)). Dieser Hash dient im weiteren Verlauf als zentrales kryptografisches Bindeglied (Key-Binding).

2. Hardware-Attestierung durch den Attestation Key

Um dem ZETA Guard (AuthS) später beweisen zu können, dass der Client-Schlüssel tatsächlich in einem sicheren Hardware-Modul liegt, wird ein dedizierter Attestierungsschlüssel (AK) erzeugt.

(06) AK anfordern mit Challenge: Der ZETA Client fordert die Generierung eines weiteren Schlüssels an (Alias_AK). Das entscheidende Sicherheitsmerkmal: Der zuvor berechnete Hash des Client-Schlüssels (hash_puk_client_sig) wird als AttestationChallenge an das OS übergeben.

(07) AK-Generierung & Zertifikatserstellung: Die StrongBox erzeugt das Attestation Key Pair. Gleichzeitig erstellt die Hardware ein X.509-Attestierungszertifikat und schreibt die übergebene AttestationChallenge unveränderlich in die ASN.1-Erweiterung (KeyDescription) des Zertifikats.

(08) & (09) Rückgabe der Zertifikatskette: Die Hardware liefert die Zertifikatskette (android_key_attestation_certificate_chain), die bis zur Google Root CA reicht, über das OS an den ZETA Client zurück. (Diese Kette beweist später dem ZETA Guard die Hardware-Bindung).

3. Expliziter Besitznachweis / Proof of Possession

Der Client muss beweisen, dass er den soeben erzeugten privaten Client-Schlüssel auch tatsächlich physisch kontrolliert.

(10) Signatur anfordern: Der ZETA Client ruft Signature.sign auf und nutzt dabei den Alias des Client Instance Keys (Alias_Client_Sig), um den Hash (hash_puk_client_sig) zu signieren.

(11) Signatur in der HW: Das Android OS instruiert die StrongBox, den Hashwert mit dem privaten Schlüssel (PrK.Client.Sig) zu signieren.

(12) & (13) Rückgabe der Signatur: Die erzeugte ECDSA-Signatur (signed_hash_puk_client_sig) wird an den Client zurückgeliefert.

4. Play Integrity API - Optional, empfohlen

Zur zusätzlichen Absicherung (Software- und Geräteintegrität) wird ein serverseitiges Urteil von Google eingeholt.

(14) Integrity Token anfordern: Der ZETA Client ruft die Google Play Integrity API auf (requestIntegrityToken). Auch hier übergibt er den Hash des Client-Schlüssels (hash_puk_client_sig) als nonce. Dadurch wird die Bewertung kryptografisch untrennbar an diesen spezifischen ZETA Client gebunden (Schutz vor Token-Injection).

(15) Rückgabe des Tokens: Die Play Integrity API liefert das signierte Base64-Token (play_integrity_token) an den Client zurück.

Der ZETA Client verfügt nun über alle notwendigen kryptografischen Artefakte, um die Dynamic Client Registration (DCR) am ZETA Guard durchzuführen).

A_29781 -ZETA Client, Android Ablauf zur Schlüsselgenerierung

Der ZETA Client MUSS den Ablauf gemäß Abbildung *Abb-ZETA-Schlüsselgenerierung-Android* unterstützen.【<=】

5.10.2.1.2 Apple Secure Enclave

Das Sequenzdiagramm beschreibt die initiale Generierung des Client Key Pair durch eine ZETA Client App auf einem Apple-Gerät. Der Vorgang ist für mobile und stationäre Clients identisch (iOS, iPadOS, macOS).

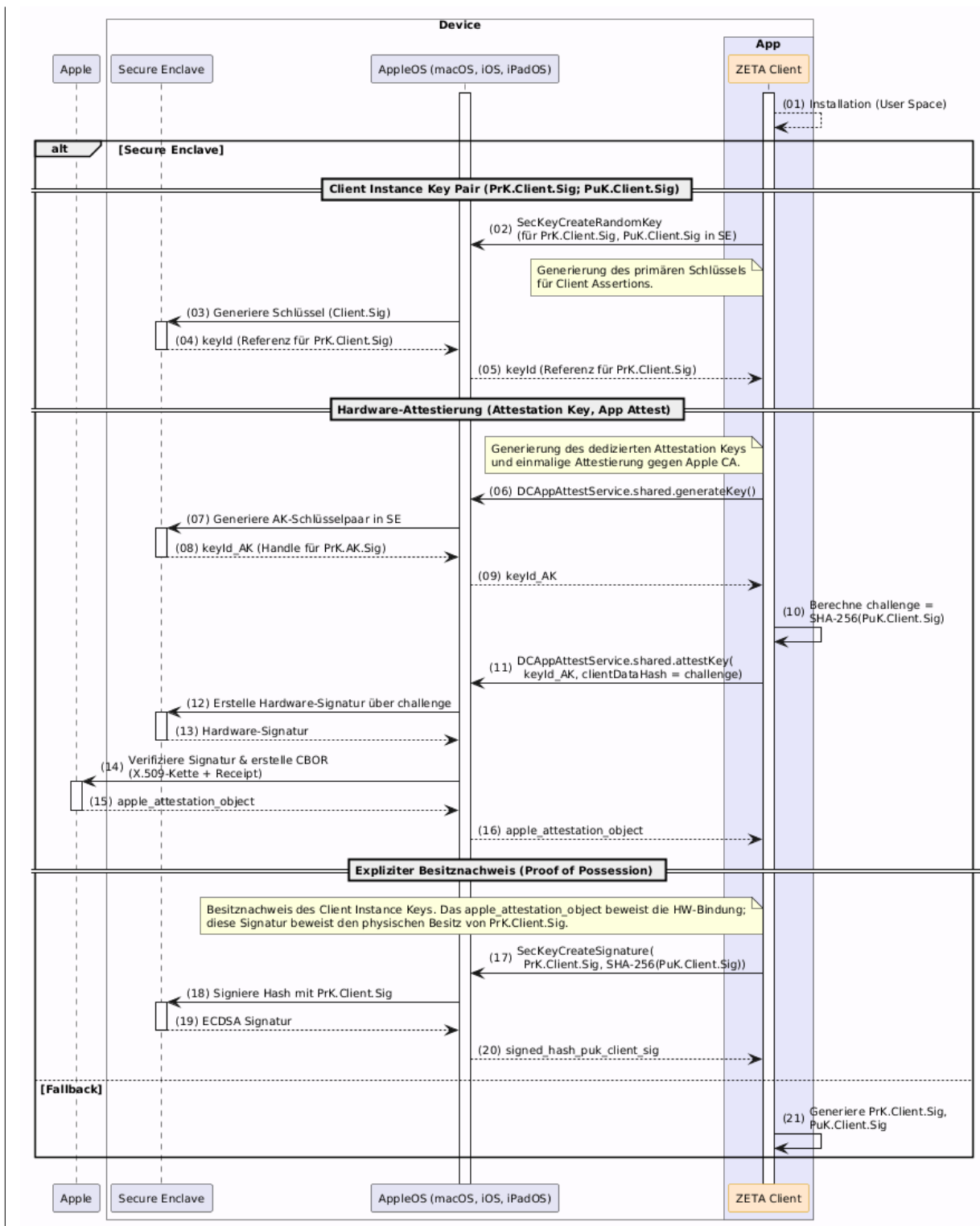


Abbildung 17 Abb-ZETA-Schlüsselgenerierung-Apple

1. Generierung des Client Instance Key Pairs

In dieser Phase erzeugt die App den primären kryptografischen Schlüssel, der zukünftig für die Authentisierung gegenüber dem ZETA Guard (Client Assertions) genutzt wird. (01) Installation: Der ZETA Client wird im User Space des Betriebssystems installiert und gestartet.

(02) Schlüsselgenerierung anfordern: Der Client ruft die native Apple-API SecKeyCreateRandomKey auf, um ein asymmetrisches Schlüsselpaar zu erzeugen. Die Parameter erzwingen die Speicherung zwingend in der hardwareisolierten Secure Enclave (SE).

(03) Erzeugung in der SE: Das Betriebssystem leitet den Befehl an die Secure Enclave weiter, welche das Schlüsselpaar (PrK.Client.Sig, PuK.Client.Sig) erzeugt. Der private Schlüssel verlässt die Secure Enclave niemals.

(04) & (05) Rückgabe der Referenz: Die Secure Enclave und das Betriebssystem geben lediglich einen Handle bzw. eine Referenz (keyId) für diesen Schlüssel an den ZETA Client zurück.

2. Hardware-Attestierung durch den Attestation Key

Um dem ZETA Guard (AuthS) beim späteren DCR-Prozess beweisen zu können, dass der Client-Schlüssel tatsächlich in der Secure Enclave dieses spezifischen Apple-Geräts liegt, wird das App Attest Framework genutzt.

(06) AK-Schlüsselpaar anfordern: Der ZETA Client fordert über DCAppAttest.shared.generateKey() die Generierung eines dedizierten Attestierungsschlüssels (AK) an.

(07) bis (09) AK in der SE: Die Secure Enclave erzeugt den Attestation Key und gibt dessen Referenz (keyId_AK) an den Client zurück.

(10) Hash-Berechnung (Challenge): Der ZETA Client berechnet den SHA-256 Hash seines öffentlichen Client-Schlüssels (challenge = SHA-256(PuK.Client.Sig)). Dies ist das entscheidende Key-Binding.

(11) Attestierung initiieren: Der Client ruft DCAppAttest.shared.attestKey auf und übergibt die Referenz des Attestation Keys (keyId_AK) sowie den Hash als clientDataHash (Challenge).

(12) bis (16) Erstellung des Attestation Objects: Die Secure Enclave signiert die Challenge mit dem Attestation Key. Das Betriebssystem kommuniziert im Hintergrund mit den Apple Services, um ein standardisiertes, von Apple signiertes CBOR-Objekt zu erstellen. Dieses finale apple_attestation_object (welches eine X.509-Zertifikatskette, das Apple-Receipt und die eingebettete Challenge enthält) wird dem Client übergeben. (Dieses Objekt beweist später dem ZETA Guard die Hardware-Bindung).

3. Expliziter Besitznachweis / Proof of Possession

Obwohl das Attestation Object beweist, dass der Schlüssel sicher in der Hardware liegt, muss der ZETA Client zusätzlich beweisen, dass genau diese App-Instanz autorisiert ist, den Schlüssel zu verwenden (Besitznachweis).

(17) Signatur anfordern: Der ZETA Client ruft die Standard-Krypto-API SecKeyCreateSignature auf. Er nutzt die Referenz des primären Client Instance Keys (PrK.Client.Sig), um den Hash seines eigenen Public Keys zu signieren.

(18) Signatur in der HW: Die Secure Enclave wird angewiesen, den Hash mit dem privaten Client-Schlüssel zu signieren.

(19) & (20) Rückgabe der Signatur: Die erzeugte ECDSA-Signatur (signed_hash_puk_client_sig) wird an den ZETA Client zurückgeliefert.

4. Software Fallback

(21) Fallback: Für den Fall, dass das Apple-Gerät keine Secure Enclave besitzt oder App Attest nicht verfügbar ist (z. B. in bestimmten Simulator-Umgebungen oder bei veralteter Hardware), generiert der ZETA Client das Schlüsselpaar rein in Software. Es existiert dann keine kryptografische Hardware-Bindung, was bei der Registrierung (DCR) am ZETA Guard zu einer Abwertung des Trust-Scores durch die Policy Engine führt.

A_29782 -ZETA Client, Ablauf Apple OS Schlüsselgenerierung

Der ZETA Client für Apple Betriebssysteme MUSS den Ablauf gemäß Abbildung *Abb-ZETA-Schlüsselgenerierung-Apple* unterstützen. [<=]

5.10.2.2 Client Registrierung und Authentifizierung

Das Sequenzdiagramm zeigt den Prozess der Dynamic Client Registration (DCR) für mobile ZETA Clients. Es vereint die kryptografische Hardware-Attestierung mit der nutzerzentrierten E-Mail-Verifikation (TOFU - Trust On First Use) und die Generierung eines Wiederherstellungscode (Recovery Code / Faktor F3) für den Fall eines Geräteverlusts.

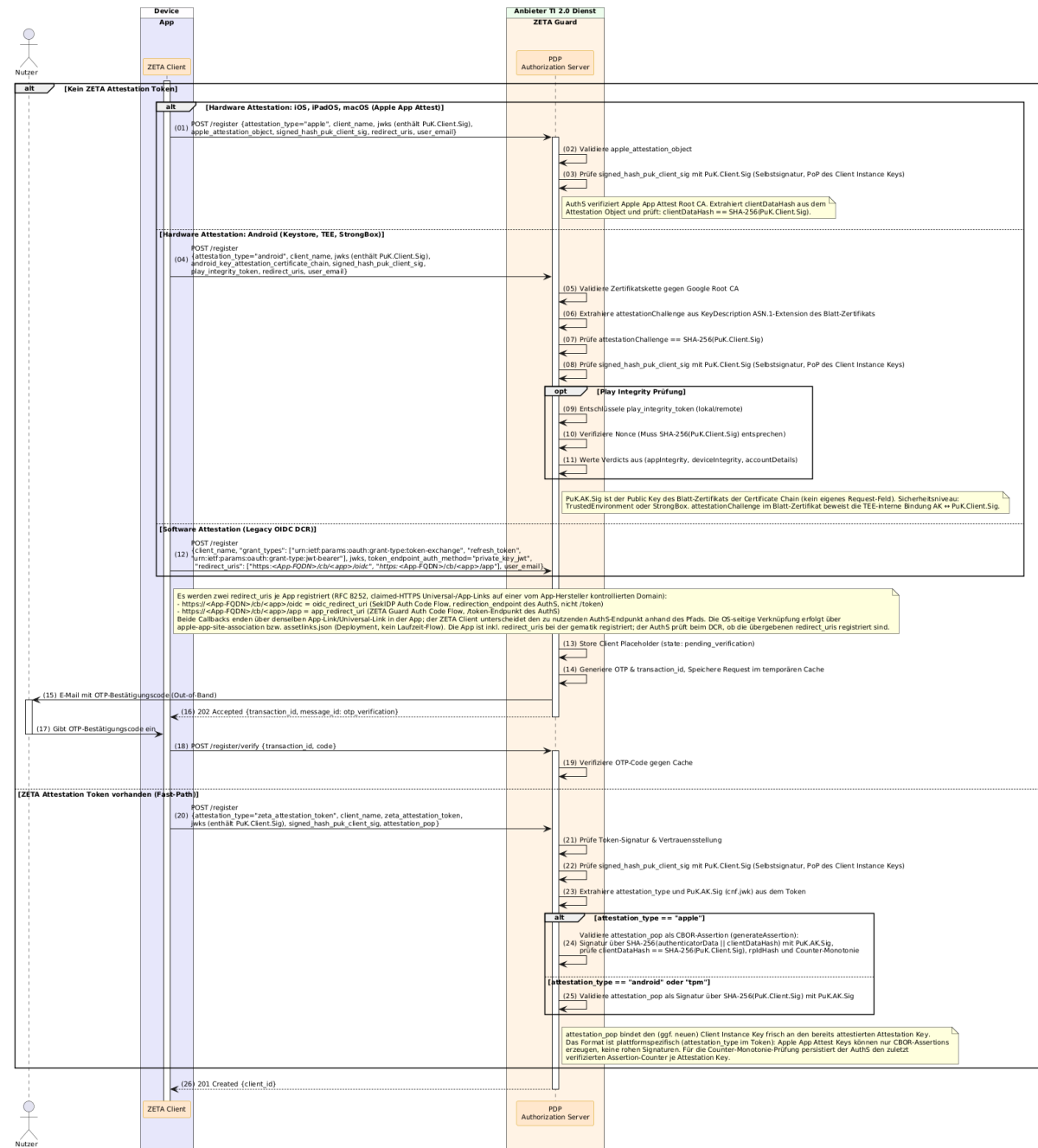


Abbildung 18 Abb-ZETA-DCR-für-mobile-Clients

Der Gesamtprozess teilt sich in zwei Hauptpfade auf:

- Pfad 1: Kein ZETA Attestation Token vorhanden (Erstregistrierung mit Hardware- oder Software-Attestierung sowie einer E-Mail-Verifizierung mittels OTP).
- Pfad 2: ZETA Attestation Token vorhanden (Fast-Path) (Schnellere Registrierung unter Wiederverwendung eines bereits ausgestellten Attestierungstokens).

Hauptpfad 1: [Kein ZETA Attestation Token]

Falls noch kein Token vorhanden ist, muss sich der Client erst registrieren. Hierbei gibt es drei verschiedene Attestierungsmethoden (Sub-Pfade):

[Hardware Attestation: iOS, iPadOS, macOS (Apple App Attest)]

(01) POST /register: Der ZETA Client sendet eine Registrierungsanfrage an den Authorization Server. Übermittelt werden unter anderem: attestation_type="apple", client_name, jwks (mit dem öffentlichen Schlüssel PuK.Client.Sig), das apple attestation object, die Signatur über den Hash des Client-Signaturschlüssels

(signed_hash_puk_client_sig), die Redirect-URLs und die E-Mail-Adresse des Nutzers.

(02) Validiere apple_attestation_object: Der Authorization Server prüft das empfangene Apple-Attestierungsobjekt (Verifikation der Apple App Attest Root CA, Abgleich des extrahierten clientDataHash mit dem SHA-256-Hash des öffentlichen Client-Schlüssels PuK.Client.Sig).

(03) Prüfe signed_hash_puk_client_sig mit PuK.Client.Sig: Der Server verifiziert die Signatur (Selbstsignatur bzw. Proof-of-Possession des Client Instance Keys).

[Hardware Attestation: Android (Keystore, TEE, StrongBox)]

(04) POST /register: Der ZETA Client sendet eine Registrierungsanfrage mit attestation_type="android", client_name, jwks (mit PuK.Client.Sig), der android key attestation certificate chain, der Signatur über den Hash des Client-Signaturschlüssels, dem play integrity token, den Redirect-URLs und der Nutzer-E-Mail.

(05) Validiere Zertifikatskette gegen Google Root CA: Der Server überprüft die Vertrauenskette der Android-Schlüsselattestierung.

(06) Extrahiere attestationChallenge aus KeyDescription ASN.1-Extension des Blatt-Zertifikats: Der Server liest die Challenge aus den Zertifikatsdetails aus.

(07) Prüfe attestationChallenge == SHA-256(PuK.Client.Sig): Der Server stellt sicher, dass die Challenge dem Hash des übermittelten öffentlichen Schlüssels entspricht.

(08) Prüfe signed_hash_puk_client_sig mit PuK.Client.Sig: Verifikation des Proof-of-Possession (PoP) des Client Instance Keys.

Optionaler Block: [Play Integrity Prüfung]

(09) Entschlüssele play_integrity_token (lokal/remote): Der Server entschlüsselt den Play-Integrity-Token von Google.

(10) Verifiziere Nonce: Der Server prüft, ob die Nonce im Token dem SHA-256-Hash von PuK.Client.Sig entspricht.

(11) Werte Verdicts aus: Auswertung der Integritätsurteile (appIntegrity, deviceIntegrity, accountDetails).

[Software Attestation (Legacy OIDC DCR)]

(12) POST /register: Der Client sendet eine klassische Registrierungsanfrage ohne Hardware-Attestierung mit Angaben zu client_name, unterstützten grant_types (Token-Exchange, Refresh Token, JWT-Bearer), jwks, der Authentifizierungsmethode (private_key_jwt), den Redirect-URLs und der Nutzer-E-Mail.

(13) Store Client Placeholder (state: pending_verification): Der Server legt einen vorläufigen Client-Eintrag an.

(14) Generiere OTP & transaction_id, Speichere Request im temporären Cache: Erstellung eines Einmalpassworts (OTP) und Speicherung des Vorgangs.

(15) E-Mail mit OTP-Bestätigungscode (Out-of-Band): Der Server sendet das OTP direkt an die E-Mail-Adresse des Nutzers.

(16) 202 Accepted (transaction_id, message_id: otp_verification): Der Server meldet dem Client, dass die Registrierung läuft und auf die Verifizierung wartet.

(17) Gibt OTP-Bestätigungscode ein: Der Nutzer liest die E-Mail und gibt das OTP in der

App (ZETA Client) ein.

(18) POST /register/verify (transaction_id, code): Der Client sendet das eingegebene OTP zusammen mit der Transaktions-ID an den Server.

(19) Verifiziere OTP-Code gegen Cache: Der Server prüft die Gültigkeit des Codes.

Hauptpfad 2: [ZETA Attestation Token vorhanden (Fast-Path)]

Wenn der Client bereits über ein gültiges ZETA-Attestierungstoken verfügt, kann der Prozess stark verkürzt werden:

(20) POST /register: Der Client sendet eine Anfrage mit attestation_type="zeta_attestation_token", client_name, dem ZETA Attestation Token selbst (zeta_attestation_token), dem jwks (mit PuK.Client.Sig) und zur Absicherung die Selbstsignatur des SHA-256-Hashes von puk_client_sig (signed_hash_puk_client_sig) und dem Besitznachweis des Attestation Keys (attestation_pop).

(21) Prüfe Token-Signatur & Vertrauensstellung: Der Server verifiziert das eingereichte ZETA Attestation Token.

(22) Prüfe signed_hash_puk_client_sig mit PuK.Client.Sig: Prüfung der Selbstsignatur (PoP) des neuen Client Instance Keys.

(23) Extrahiere attestation_type und PuK.AK.Sig (cnf.jwk) aus dem Token: Der Server liest den Attestierungstyp und den öffentlichen Attestierungsschlüssel (PuK.AK.Sig) aus dem Token aus.

Bedingte Verifizierung (abhängig vom extrahierten Typ):

Bei attestation_type == "apple":

(24) Validiere attestation_pop als CBOR-Assertion (generateAssertion): Validierung der Signatur über die Authentifikatordaten und den Client-Daten-Hash unter Verwendung von PuK.AK.Sig. Zudem Prüfung des rpldHash und der Monotonie des Counters.

Bei attestation_type == "android" oder "tpm":

(25) Validiere attestation_pop als Signatur über SHA-256(PuK.Client.Sig) mit PuK.AK.Sig: Direkte Signaturprüfung zur Bindung des (neuen) Client-Schlüssels an den bereits bekannten Attestierungsschlüssel.

Abschluss der Registrierung

Egal welcher Pfad erfolgreich durchlaufen wurde, der Prozess endet mit der Bestätigung:

(26) 201 Created (client_id): Der Authorization Server sendet die Antwort 201 Created inklusive der zugewiesenen client_id an den ZETA Client zurück. Damit ist die Client-Registrierung abgeschlossen.

A_29658 -ZETA, Ablauf Client Registrierung für mobile Clients

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-DCR-für-mobile-Clients* unterstützen. [<=]

5.10.2.3 Plattformabhängige Attestierung

Die Attestierungsdaten werden in der Client Assertion (siehe [client-assertion-jwt]) im Attribut client_statement (siehe [client-statement.yaml]) eingetragen.

5.10.2.3.1 Android

Auf Android-Geräten erfolgt die Attestierung über plattformspezifische Sicherheitsmechanismen, die eine Bindung zwischen Anwendung, Schlüsselmaterial und Gerät herstellen.

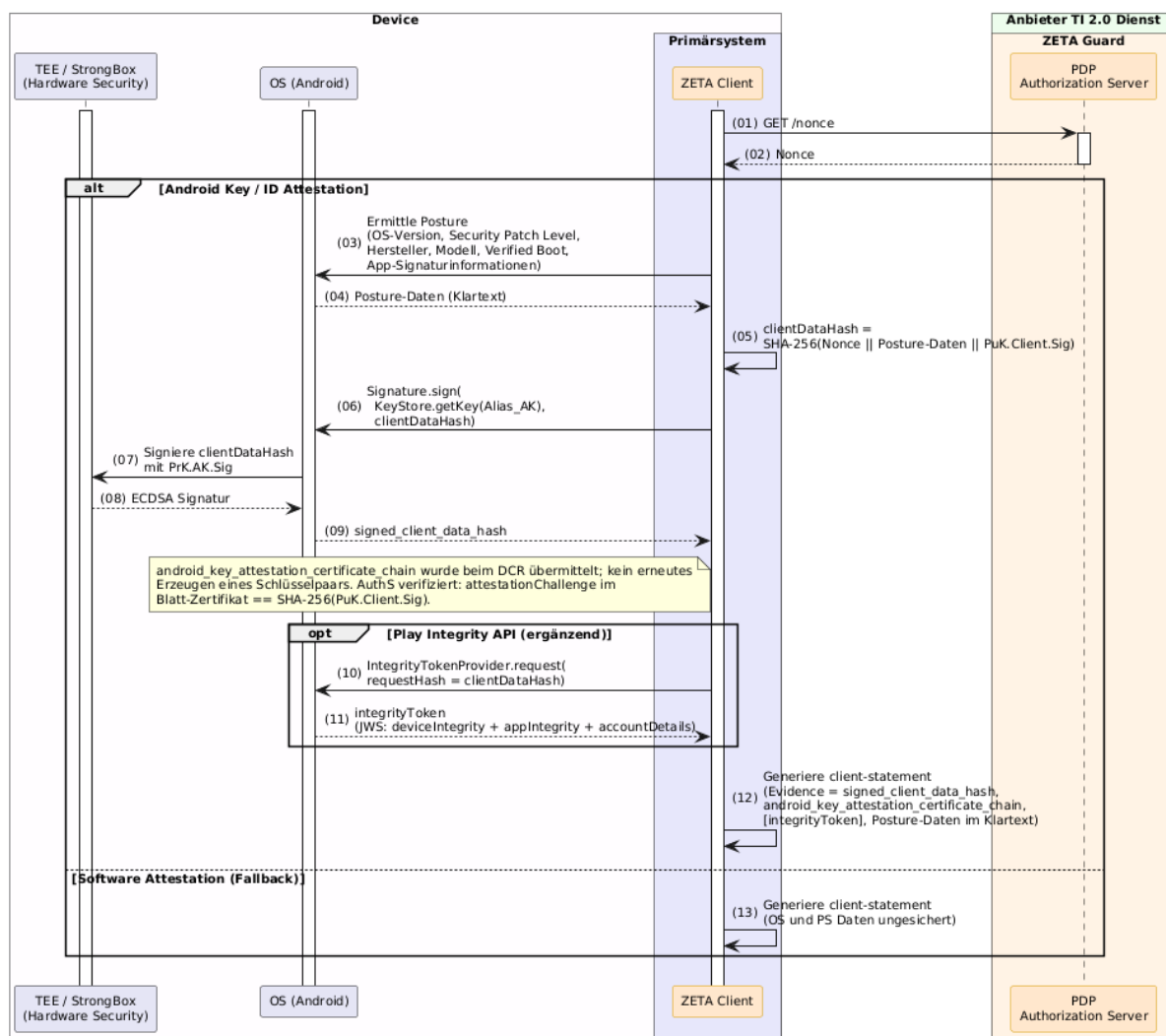


Abbildung 19 Abb-ZETA-Client-Statement-mit-Android-Attestation

Vorbereitungsphase: Nonce-Bezug

Um Replay-Angriffe zu verhindern und die Frische des Nachweises sicherzustellen, benötigt der Client eine kryptografische Challenge vom Server.

(01) GET /nonce: Der ZETA Client fragt eine neue Nonce beim ZETA Guard (Authorization Server) an.

(02) Nonce: Der ZETA Guard generiert eine zufällige Nonce und liefert diese zurück.

Primärer Pfad: Android Key / ID Attestation (Hardwaregestützt)

Dieser Pfad wird durchlaufen, wenn das Gerät über eine Trusted Execution Environment (TEE) oder StrongBox verfügt und die Hardware-Attestierung bei der Registrierung erfolgreich war.

(03) Ermittle Posture: Der ZETA Client fragt über native Android-APIs die aktuellen, sicherheitsrelevanten Systemparameter ab (z. B. OS-Version, Security Patch Level, Hersteller, Modell, Verified Boot, App-Signaturinformationen).

(04) Posture-Daten (Klartext): Das Android OS übergibt diese Systemdaten im Klartext an den Client.

(05) Hash-Berechnung (clientDataHash): Dies ist der zentrale kryptografische Sicherheitsanker. Der ZETA Client berechnet einen SHA-256 Hash aus der Verkettung der serverseitigen Nonce, den ermittelten Posture-Daten und seinem öffentlichen Client-Schlüssel (PuK.Client.Sig).

(Dadurch werden die Frische des Requests, der Systemzustand und die Identität des Clients untrennbar aneinander gebunden).

(06) Signatur anfordern: Der Client ruft Signature.sign auf. Hierbei referenziert er den dedizierten Attestation Key (Alias_AK), der beim DCR-Prozess in der Hardware erzeugt wurde, und fordert die Signatur des clientDataHash an.

(07) Signatur in der HW: Das Android OS instruiert die TEE / StrongBox, den Hash mit dem privaten Attestation Key (PrK.AK.Sig) zu signieren.

(08) & (09) Rückgabe der Signatur: Die TEE liefert die erzeugte ECDSA-Signatur (signed_client_data_hash) an den ZETA Client zurück.

(Erklärung gemäß Notiz: Da der ZETA Guard bereits beim DCR verifiziert hat, dass der Attestation Key kryptografisch an den Client-Schlüssel gebunden ist [attestationChallenge == SHA-256(PuK.Client.Sig)], beweist diese Laufzeit-Signatur dem Server, dass die Posture-Daten vom selben physischen Gerät stammen).

Optionaler Block: Play Integrity API (ergänzend)

Da die Posture-Daten im Laufzeit-Flow durch Software-APIs ermittelt wurden, kann (und sollte) dieser Status durch eine serverseitige Google-Prüfung verifiziert werden.

(10) Request an Play Integrity: Der ZETA Client ruft den IntegrityTokenProvider auf. Entscheidend: Er übergibt den exakt selben clientDataHash als requestHash.

(11) integrityToken (JWS): Das System liefert das von Google signierte Token zurück, welches unabhängige Bewertungen (Verdicts) zur Geräte- und App-Integrität enthält und kryptografisch an den aktuellen Request gebunden ist.

Zusammenbau des Statements

(12) Generiere client-statement: Der ZETA Client baut das finale client-statement (Evidence) für den ZETA Guard zusammen. Es besteht aus der Hardware-Signatur (signed_client_data_hash), der beim DCR ausgetauschten android_key_attestation_certificate_chain, dem optionalen integrityToken und den Posture-Daten im Klartext.

Alternativer Pfad: Software Attestation (Fallback)

(13) Generiere client-statement (Software): Falls keine TEE/StrongBox verfügbar ist, sammelt der ZETA Client die OS- und Primärsystem-Daten (PS-Daten) und fügt sie ungesichert in das Statement ein. Die Policy Engine des ZETA Guards wird diesen Request mit einem deutlich geringeren Vertrauensniveau (Trust-Score) bewerten.

A_29809 -ZETA Client, Ablauf Android Attestierung

Der ZETA Client MUSS den Ablauf gemäß Abbildung *Abb-ZETA-Client-Statement-mit-Android-Attestation* unterstützen.

[<=]

5.10.2.3.2 Apple

Für Apple-basierte Geräte erfolgt die Attestation durch die Secure Enclave in Kombination mit Apple App Attest.

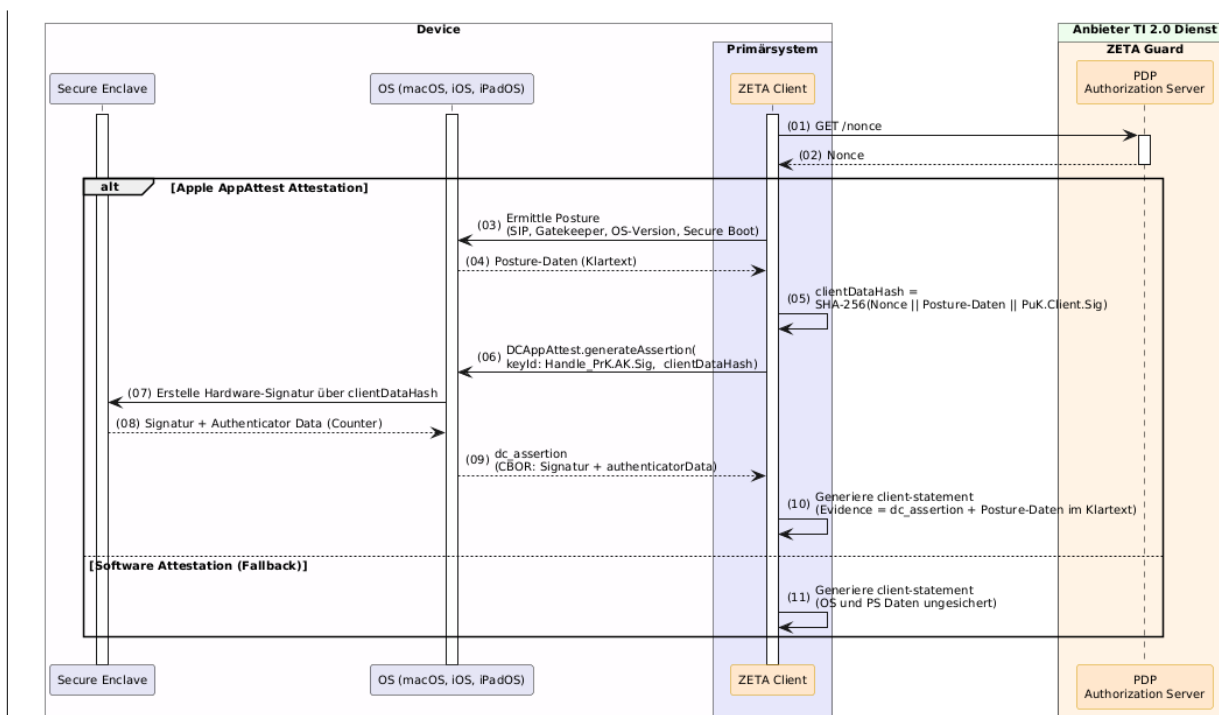


Abbildung 20 Abb-ZETA-Client-Statement-mit-Apple-AppAttest

Vorbereitungsphase: Nonce-Bezug

Um Replay-Angriffe zu verhindern und die Frische des Integritätsnachweises zu garantieren, benötigt der Client eine kryptografische Challenge vom Server.
 (01) GET /nonce: Der ZETA Client fragt eine neue Nonce beim ZETA Guard (Authorization Server) an.
 (02) Nonce: Der ZETA Guard generiert eine zufällige Nonce und liefert diese zurück.

Primärer Pfad: Apple AppAttest Attestation (Hardwaregestützt)

Dieser Pfad wird durchlaufen, wenn das Gerät über eine Secure Enclave verfügt und das App Attest Framework bei der initialen Registrierung (DCR) erfolgreich eingerichtet wurde.
 (03) Ermittle Posture: Der ZETA Client fragt über native Apple-APIs sicherheitsrelevante Systemparameter ab. Dazu gehören beispielsweise der Status der System Integrity Protection (SIP), Gatekeeper-Einstellungen, die exakte OS-Version und der Status des Secure Boots.
 (04) Posture-Daten (Klartext): Das Betriebssystem übergibt diese Systemdaten im Klartext an den ZETA Client.
 (05) Hash-Berechnung (clientDataHash): Dies ist der zentrale kryptografische Sicherheitsanker. Der ZETA Client berechnet einen SHA-256 Hash aus der Verkettung der serverseitigen Nonce, den ermittelten Posture-Daten und seinem öffentlichen Client-Schlüssel (PuK.Client.Sig).
 (Architektur-Hinweis: Dadurch werden die Frische des Requests, der Systemzustand und die Identität des Clients untrennbar aneinander gebunden).
 (06) Assertion anfordern: Der Client ruft die API DCAppAttest.generateAssertion auf. Er übergibt das Handle seines hardwaregebundenen Attestation Keys (Handle_PrK.AK.Sig) sowie den soeben berechneten clientDataHash.
 (07) Hardware-Signatur: Das Betriebssystem leitet den Hash an die Secure Enclave weiter. Die Secure Enclave signiert diesen Hash mit dem privaten Attestation Key.
 (08) Rückgabe der Signatur & Metadaten: Die Secure Enclave gibt die Signatur sowie zusätzliche Authenticator-Daten (insbesondere einen monoton steigenden Counter zum Schutz vor Klon-Angriffen) an das OS zurück.
 (09) dc_assertion (CBOR): Das OS verpackt die Signatur und die Authenticator-Daten in

ein standardisiertes CBOR-Objekt (dc_assertion) und reicht es an den ZETA Client weiter.
(10) Generiere client_statement: Der ZETA Client baut das finale client_statement (Evidence) für den Request an den ZETA Guard zusammen. Es besteht aus der kryptografischen dc_assertion und den Posture-Daten im Klartext. (Der ZETA Guard kann später den Hash aus der ihm bekannten Nonce, den Klartext-Posture-Daten und dem registrierten Client-Key selbst berechnen und die Hardware-Signatur in der dc_assertion validieren).

Alternativer Pfad: Software Attestation (Fallback)

Dieser Pfad dient als Rückfallebene, falls keine Hardware-Unterstützung vorliegt.
(11) Generiere client_statement (Software): Der ZETA Client sammelt die OS- und Primärsystem-Daten (PS-Daten) und fügt sie ungesichert in das Statement ein. Da hierbei keine Signatur durch eine Secure Enclave stattfindet, wird die Policy Engine des ZETA Guards diesen Request mit einem entsprechend niedrigen Trust-Score bewerten.

A_29810 -ZETA Client, Ablauf Apple OS Attestierung

Der ZETA Client MUSS den Ablauf gemäß Abbildung *Abb-ZETA-Client-Statement-mit-Apple-AppAttest* unterstützen.

[<=]

5.10.2.3.3 Software Attestierung (Fallback)

Falls keine hardwaregestützte Attestierung möglich ist, wird eine softwarebasierte Attestierung durchgeführt.
Dies betrifft insbesondere:

- ältere Geräte
- Emulatoren oder spezielle Umgebungen
- Geräte ohne verfügbare Attestation-APIs oder Hardware

Der Ablauf ist wie folgt:

(01) Generierung des Client-Schlüssels
Das Schlüsselpaar wird im Software-Kontext erzeugt.
(02) Erhebung von Basisinformationen
Der ZETA Client sammelt einfache Kontextdaten:

- Betriebssystem und Version
- Architektur
- Produkt- und Versionsinformationen

Eine hardwarebasierte Vertrauensbindung besteht nicht. Der resultierende Sicherheitsgrad ist daher geringer. Die finale Zugriffserlaubnis erfolgt unter zusätzlicher Policy-Prüfung.

5.10.2.4 Service Discovery

Die Service Discovery ist identisch für stationäre und mobile Clients. Siehe 5.3.1.3- Service Discovery.

5.10.2.5 Authentifizierung

Mobile ZETA Clients authentisieren den Nutzer interaktiv über einen sektoralen Identity Provider (IDP) der TI-Föderation. Hierzu wird ein OpenID Connect (OIDC) Authorization Code Flow mit Pushed Authorization Request (PAR, RFC 9126) und Proof Key for Code Exchange (PKCE, RFC 7636) ausgeführt.

Auf dem mobilen Endgerät wirken zwei Komponenten zusammen:

- Der **Fach-Client** enthält die fachliche Logik für die Arbeit mit dem Resource Server. Er wählt bzw. lässt den Nutzer den zuständigen sektoralen IDP wählen (idp_iss), erstellt den vollständigen fachlichen HTTPS-Request inklusive aller Header und übergibt ihn an den ZETA Client.
- Der **ZETA Client** kümmert sich um Client-Registrierung, Session-, Schlüssel- und Token-Verwaltung sowie die Authentifizierung. Er verarbeitet den Request, beschafft bei Bedarf ein gültiges Access Token und gibt die Response an den Fach-Client zurück.

Diese Aufgabenteilung entspricht dem Zusammenspiel von Fach-Client und ZETA Client beim Zugriff auf den Resource Server (siehe Kapitel 5.3.1.7).

Der PDP Authorization Server des ZETA Guard übernimmt zwei Rollen:

- Gegenüber dem mobilen ZETA Client agiert er als der für den Fachdienst zuständige Authorization Server (äußerer Flow, code_challenge_app).
- Gegenüber dem sektoralen IDP agiert er als OpenID-Connect-Relying-Party (Fachdienst, innerer Flow, code_challenge_as). Die Client-Authentifizierung am IDP erfolgt per mTLS (self_signed_tls_client_auth), die Vertrauensbeziehung über OpenID Federation 1.0.

Das vom PDP Authorization Server ausgestellte Access Token ist über DPoP an die Sitzung gebunden.

5.10.2.5.1 Voraussetzungen

- **Service Discovery abgeschlossen:** Der Client kennt die Endpunkte des PDP Authorization Server (authorization_endpoint, token_endpoint); siehe Kapitel 5.3.1.3-Service Discovery.
- **Dynamic Client Registration abgeschlossen:** Der Client besitzt eine client_id sowie das Schlüsselpaar PrK.Client.Sig / PuK.Client.Sig.
- **App-Link / Universal-Link** für ZETA Client und Authenticator-Modul sind im Betriebssystem registriert. Sektoraler IDP wird vom Fach-Client ausgewählt und als idp_iss an den ZETA Client übergeben.

5.10.2.5.2 Übersicht

Der Ablauf besteht aus einem äußeren Authorization-Code-Flow (ZETA Client ↔ PDP Authorization Server) und einem inneren Authorization-Code-Flow (PDP Authorization Server ↔ sektoraler IDP) und ist in drei Teilabläufe (A), (B) und (C) zerlegt:

- **(A) Authorization Request mit PAR:** Der ZETA Client startet den Authorization Request; der PDP Authorization Server stellt einen Pushed Authorization Request (PAR) am IDP und erhält eine request_uri.
- **(B) Nutzerauthentisierung:** Der ZETA Client öffnet das Authenticator-Modul; der Nutzer authentisiert sich (eGK+PIN / eID) und gibt den Consent frei; der IDP stellt einen AUTHORIZATION_CODE (IDP) aus.
- **(C) Token-Bezug und Ausstellung der ZETA Token:** Der PDP Authorization Server löst den Code am IDP ein, entscheidet über die Policy Engine und stellt dem ZETA Client ein DPoP-gebundenes Access Token und ein Refresh Token aus.

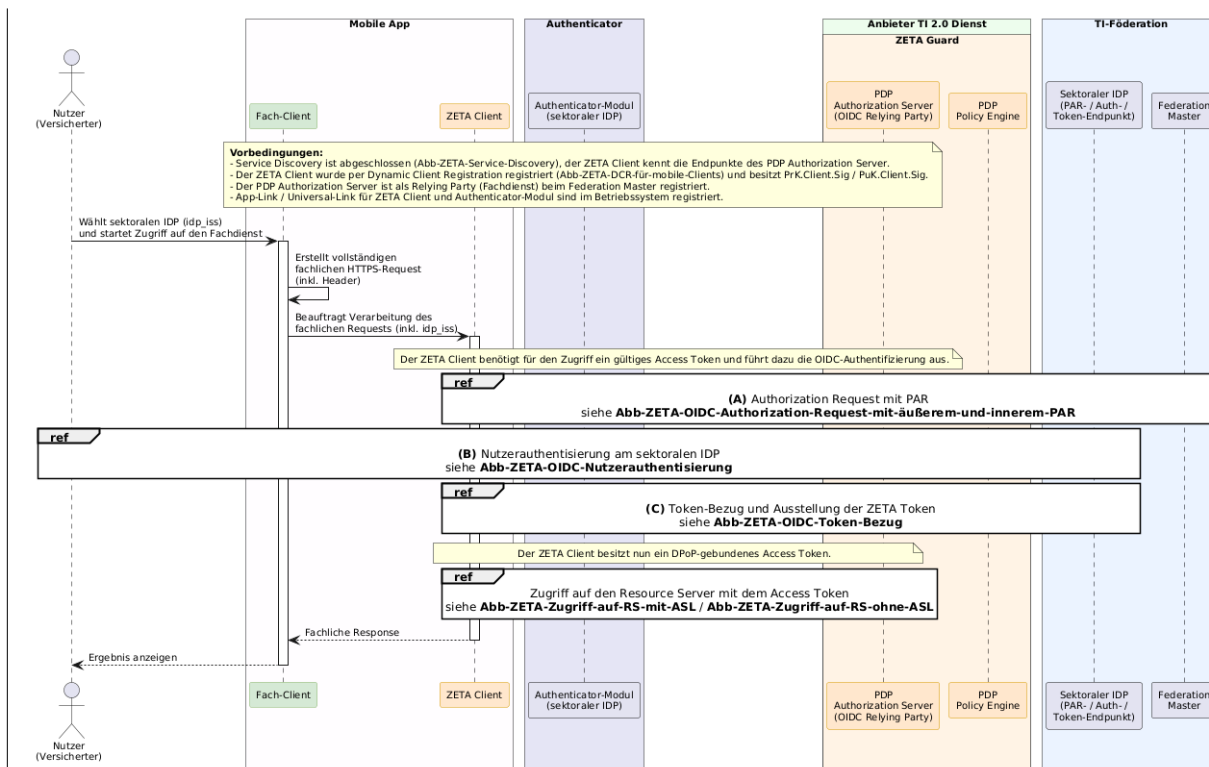


Abbildung 21 Abb-ZETA-OIDC-Authentifizierung-mobiler-Clients

A_29659 -ZETA, Ablauf OIDC Authentifizierung für mobile Clients

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-OIDC-Authentifizierung-mobiler-Clients* unterstützen. [**<=**]

5.10.2.5.3 Teilablauf (A): Authorization Request mit äußerem und innerem PAR

Das folgende Sequenzdiagramm beschreibt die initiale Phase der Nutzerauthentisierung für mobile Clients (Apps) innerhalb der ZETA-Architektur. Im Kern zeigt es den Aufbau eines OIDC Authorization Requests, erweitert um Sicherheitsmechanismen wie PKCE (Proof Key for Code Exchange), PAR (Pushed Authorization Requests) und die dynamische Vertrauensbildung über die TI-Föderation (OpenID Federation).

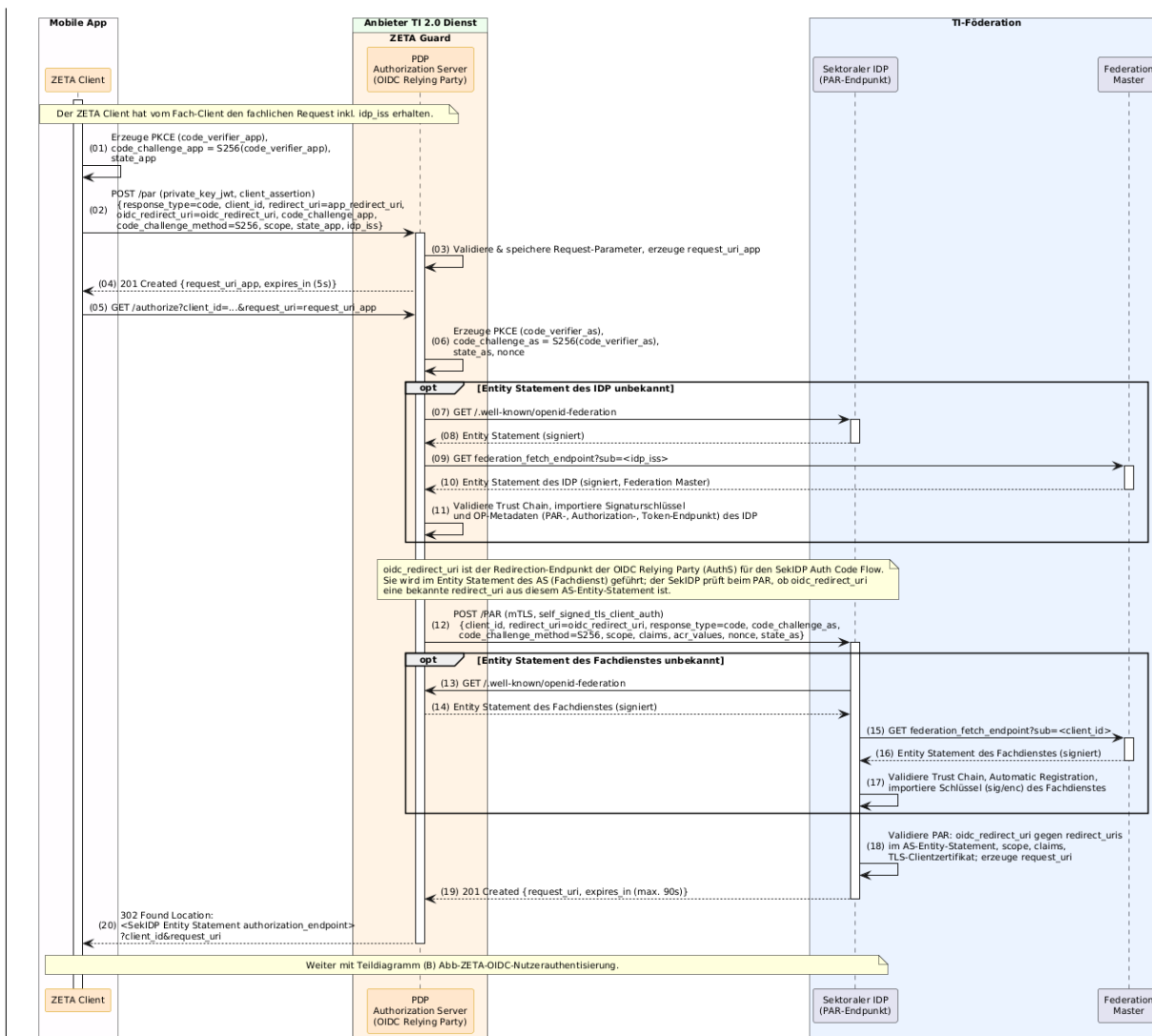


Abbildung 22 Abb-ZETA-OIDC-Authorization-Request-mit-äußerem-und-innerem-PAR

Ausgangssituation: Der ZETA Client hat vom aufrufenden Fach-Client (der eigentlichen Fach-App) den Auftrag erhalten, einen Request abzusichern. Dabei wurde auch der Identifier des gewünschten Identity Providers (idp_iss) übergeben.

1. PAR und Initiierung durch den ZETA Client

- (01) PKCE & State Generierung (Client): Der ZETA Client generiert einen zufälligen code_verifier_app, leitet daraus über SHA-256 die code_challenge_app ab und erstellt einen state_app zur Verhinderung von CSRF-Angriffen.
- (02) POST /par am ZETA Guard: Der Client initiiert den Autorisierungsablauf über Pushed Authorization Requests (RFC 9126) und sendet einen POST /par an den ZETA Guard. Er authentisiert sich dabei über eine Client Assertion (private_key_jwt). Im Body überträgt er die Autorisierungsparameter (u. a. response_type=code, app_redirect_uri, die eigene PKCE-Challenge, scope, state_app und idp_iss).
- (03) Validierung & request_uri Erzeugung: Der ZETA Guard validiert die Client Assertion sowie die Request-Parameter, speichert diese zwischen und generiert eine eindeutige request_uri_app.
- (04) 201 Created (request_uri_app): Der Guard antwortet mit HTTP 201 und übergibt die request_uri_app mit einer sehr kurzen Gültigkeitsdauer von 5 Sekunden.
- (05) GET /authorize: Der ZETA Client ruft nun den Authorization-Endpoint des ZETA Guards auf (GET /authorize), wobei lediglich die eigene client_id und die zuvor erhaltene

| sichere request_uri_app übergeben werden.

2. Vorbereitung durch den ZETA Guard (als Relying Party)

(06) PKCE & Nonce Generierung (Guard): Da der Guard als Vermittler (Relying Party) gegenüber dem sektoralen IDP auftritt, generiert er einen eigenen, zweiten PKCE-Satz (code_verifier_as, code_challenge_as), einen eigenen state_as sowie einen nonce für das spätere ID-Token.

3. Dynamischer Vertrauensaufbau: IDP (OpenID Federation)

(Dieser optionale Block wird nur ausgeführt, wenn dem ZETA Guard das Entity Statement des angeforderten IDP noch nicht bekannt ist oder dieses abgelaufen ist).

(07) & (08) Fetch IDP Entity Statement: Der Guard ruft das signierte Entity Statement des sektoralen IDPs über dessen .well-known/openid-federation Endpunkt ab.

(09) & (10) Fetch from Federation Master: Um die Vertrauenswürdigkeit des IDP zu überprüfen, fragt der Guard beim zentralen Federation Master das Entity Statement für diesen IDP (Subjekt = idp_iss) an.

(11) Validierung & Metadaten-Import: Der Guard validiert die erhaltene Trust Chain, importiert den Signaturschlüssel des IDP sowie dessen OP-Metadaten (PAR-, Authorization- und Token-Endpunkt).

4. Pushed Authorization Request (PAR) am sektoralen IDP

(12) POST /PAR am SekIDP: Der Guard sendet die Autorisierungsdaten per mTLS (unter Nutzung seiner self_signed_tls_client_auth) an den PAR-Endpunkt des sektoralen IDP. Im Payload übergibt er seine eigene client_id, die oidc_redirect_uri (der eigene Redirection-Endpunkt für den SekIDP Auth Code Flow), seine PKCE-Daten (code_challenge_as), scope, claims, acr_values, nonce und state_as.

5. Dynamischer Vertrauensaufbau: ZETA Guard (OpenID Federation)

(Dieser optionale Block auf Seiten des IDP wird nur ausgeführt, wenn dem IDP das Entity Statement des anfragenden Fachdienstes/Guards noch nicht bekannt ist).

(13) & (14) Fetch Guard Entity Statement: Der IDP ruft das Entity Statement des anfragenden ZETA Guards ab.

(15) & (16) Fetch from Federation Master: Der IDP verifiziert den Guard beim Federation Master (Subjekt = client_id).

(17) Automatic Registration: Der IDP validiert die Trust Chain, führt die Automatic Registration durch und importiert die Schlüssel (Signatur und Verschlüsselung) des Fachdienstes.

6. PAR-Antwort am IDP und Umleitung an den Authenticator

(18) PAR-Validierung (SekIDP): Der sektorale IDP validiert den PAR-Request (u. a. Prüfung der oidc_redirect_uri gegen die im AS-Entity-Statement hinterlegten redirect_uris, Scopes, Claims und das TLS-Clientzertifikat) und erzeugt eine eindeutige request_uri.

(19) 201 Created (SekIDP): Der IDP antwortet dem Guard mit HTTP 201 und übergibt die generierte request_uri (Gültigkeit max. 90 Sekunden).

(20) 302 Found (HTTP-Redirect an App): Der ZETA Guard beantwortet nun den Authorization Request des ZETA Clients (aus Schritt 05) mit einem HTTP 302 Redirect. Die Location-URL verweist auf den Authorization-Endpunkt des sektoralen IDP (aus dessen Entity Statement), parametrisiert mit der client_id des Guards und der neuen request_uri. (Dieser Redirect löst auf dem mobilen Endgerät den App-Wechsel in das Authenticator-Modul aus).

A_29660 -ZETA, Ablauf Authorization Request mit PAR

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-OIDC-Authorization-Request-mit-äußerem-und-innerem-PAR* unterstützen.

[<=]

5.10.2.5.4 Teilablauf (B): Nutzerauthentisierung am sektoralen IDP

Der ZETA Client delegiert die interaktive Nutzerauthentisierung an das Authenticator-Modul des sektoralen IDP. Im Zentrum dieses Ablaufs steht der Wechsel aus der aufrufenden App (ZETA Client) in eine dedizierte Authenticator-App (z. B. die App der Krankenkasse), in welcher der Versicherte sich sicher authentisiert (z. B. via eGK und PIN oder eID) und dem Datenzugriff zustimmt. Eine In-App-Authentifizierung ist ebenfalls möglich, wenn der Herausgeber der Fach-App (z. B. der ePA-App) identisch mit dem Anbieter des Authenticator-Moduls (z. B. der Krankenkasse als sektoraler IDP) ist.

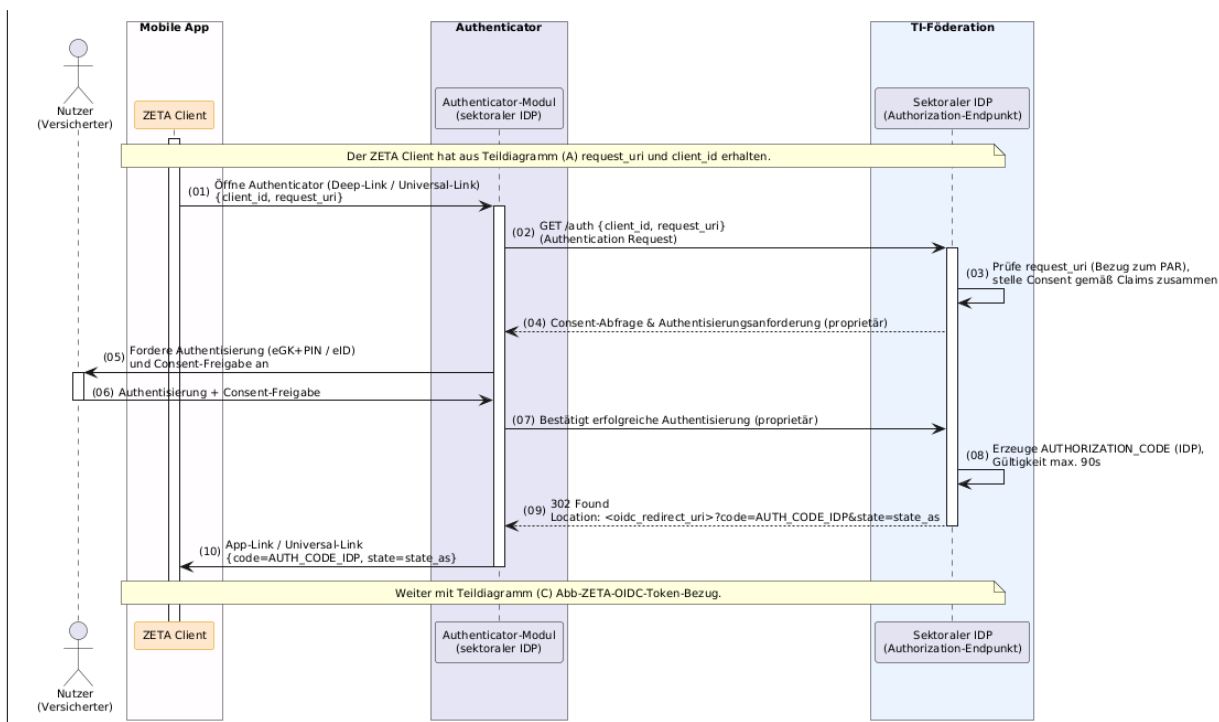


Abbildung 23 Abb-ZETA-OIDC-Nutzerauthentisierung

- (01) Der ZETA Client öffnet das Authenticator-Modul des sektoralen IDP per Deep-Link bzw. Universal-Link und übergibt `client_id` und `request_uri`.
- (02) Das Authenticator-Modul übermittelt den Authentication Request (GET /auth mit `client_id` und `request_uri`) an den Authorization-Endpunkt des sektoralen IDP.
- (03) Der IDP prüft die `request_uri` (Bezug zum zuvor gestellten PAR) und stellt die Consent-Abfrage gemäß den angeforderten claims zusammen.
- (04) Der IDP fordert über das Authenticator-Modul die Nutzerauthentisierung und die Consent-Freigabe an (proprietäres Protokoll des sektoralen IDP).
- (05)-(06) Der Nutzer authentisiert sich (z. B. eGK+PIN oder eID) und gibt den Consent frei.
- (07) Das Authenticator-Modul bestätigt dem IDP die erfolgreiche Authentisierung.
- (08) Der IDP erzeugt einen AUTHORIZATION_CODE (IDP) mit einer Gültigkeit von max. 90 Sekunden.
- (09) Der IDP antwortet mit einem 302 Found. Die Location verweist auf die `oidc_redirect_uri` des Fachdienstes und enthält `code=AUTH_CODE_IDP` und `state=state_as`.
- (10) Das Authenticator-Modul ruft den ZETA Client per App-Link bzw. Universal-Link auf und übergibt `code=AUTH_CODE_IDP` und `state=state_as`.

A_29671 -ZETA, Ablauf Nutzerauthentisierung am sektoralen IDP

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-OIDC-Nutzerauthentisierung* unterstützen. [<=]

5.10.2.5.5 Teilablauf (C): Token-Bezug und Ausstellung der ZETA Token

Das folgende Sequenzdiagramm (Teildiagramm C) spezifiziert den finalen Abschnitt der Nutzerauthentisierung: die Auflösung des sektoralen IDP-Codes im Backend (Innerer Flow), die dynamische Policy-Evaluation und die anschließende Ausstellung des DPoP-gebundenen ZETA Access Tokens (Äußerer Flow).

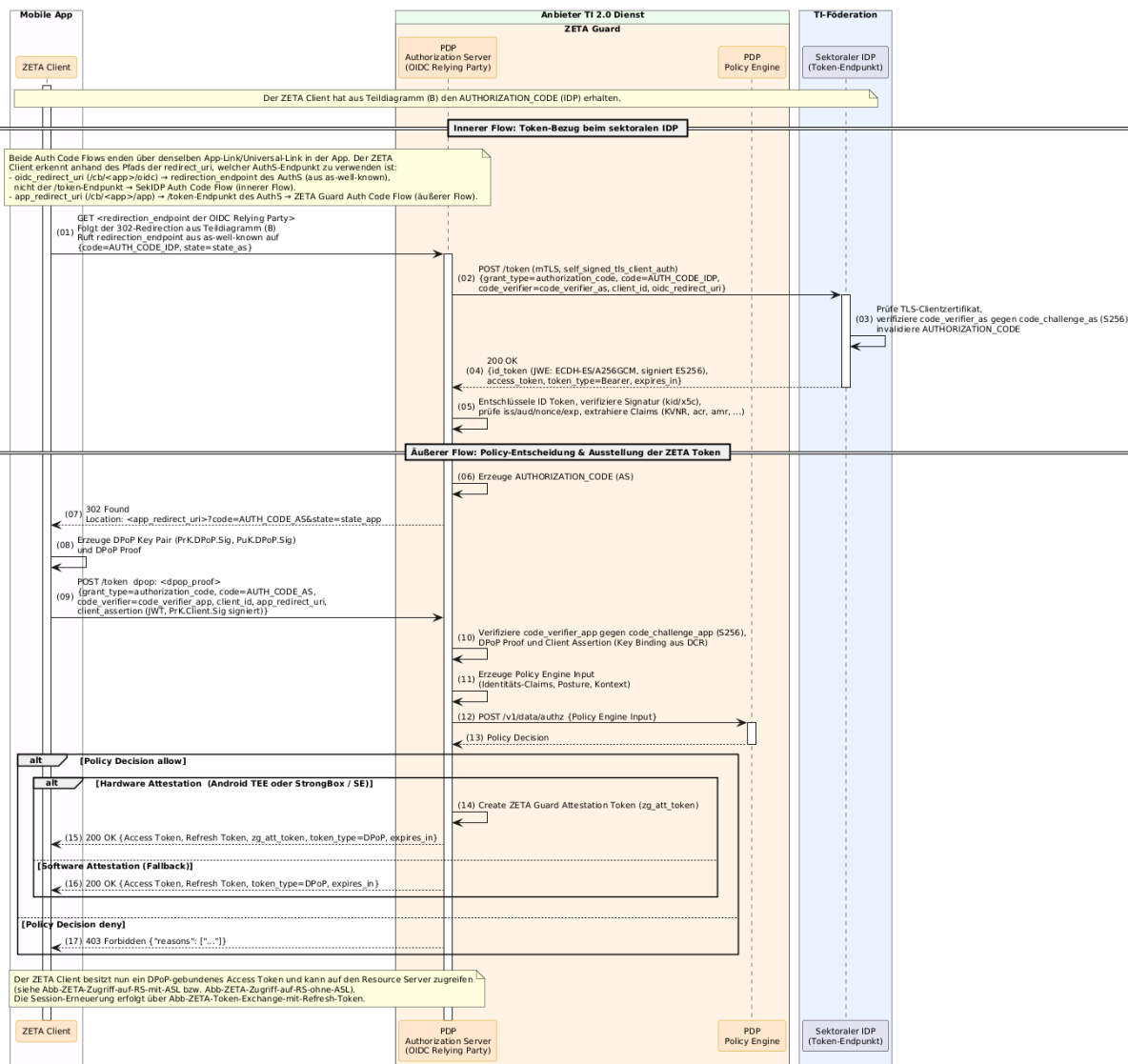


Abbildung 24 Abb-ZETA-OIDC-Token-Bezug

Vorab-Definition: Routing-Logik der Redirection URIs

Da sowohl der innere Auth Code Flow (SekIDP) als auch der äußere Auth Code Flow (ZETA Guard) über App-Links/Universal-Links auf dem mobilen Endgerät terminieren, nutzt der ZETA Client eine strikte Pfad-Unterscheidung, um den korrekten Folge-Endpunkt anzusteuern:

oidc_redirect_uri (Schema: /cb/<app>/oidc): Signalisiert dem ZETA Client den Abschluss des inneren Flows. Der Client erkennt am Pfad-Suffix /oidc, dass der empfangene Code an den Redirection-Endpunkt des ZETA Guards (aus dessen as-well-

known) weitergeleitet werden muss - und ausdrücklich nicht an dessen /token-Endpunkt. **redirect_uri (Schema: /cb/<app>/app):** Signalisiert dem ZETA Client den Abschluss des äußeren Flows. Der Client erkennt am Pfad-Suffix /app, dass der empfangene Code nun am /token-Endpunkt des ZETA Guards eingetauscht werden muss.

Ablaufbeschreibung: OIDC Token-Bezug (Teildiagramm C)

Ausgangssituation: Der ZETA Client hat am Ende von Teildiagramm B den AUTHORIZATION_CODE (AUTH_CODE_IDP) des sektoralen IDP erhalten.

1. Innerer Flow: Token-Bezug beim sektoralen IDP

(01) Code-Übergabe an den Guard: Folgend der 302-Redirection aus Teildiagramm B ruft der ZETA Client den Redirection-Endpunkt des ZETA Guards auf (GET <oidc_redirect_uri>) und überträgt den AUTH_CODE_IDP sowie den state_as.

(02) POST /token am SekIDP: Der ZETA Guard (als Relying Party) ruft per mTLS (unter Nutzung seiner self_signed_tls_client_auth) den Token-Endpunkt des sektoralen IDP auf. Er überträgt den AUTH_CODE_IDP, seinen eigenen code_verifier_as, seine client_id und die oidc_redirect_uri.

(03) Validierung durch den SekIDP: Der sektorale IDP prüft das TLS-Clientzertifikat des Guards, verifiziert den code_verifier_as gegen die code_challenge_as (S256) und invalidiert den einmalig gültigen Authorization Code.

(04) 200 OK (Token-Ausstellung SekIDP): Der SekIDP antwortet mit HTTP 200 und übermittelt ein verschlüsseltes ID-Token (id_token als JWE: ECDH-ES/A256GCM, signiert mit ES256), ein kurzlebigen Access Token (token_type=Bearer) sowie die Gültigkeitsdauer (expires_in).

(05) Token-Entschlüsselung & Extraktion: Der ZETA Guard entschlüsselt das ID-Token, verifiziert die Signatur über die Zertifikatskette (kid/x5c), prüft iss, aud, nonce sowie exp und extrahiert die im Token enthaltenen fachlichen Identitäts-Claims (u. a. die Krankenversichertennummer (KVNR), acr und amr).

2. Äußerer Flow: Policy-Entscheidung (Policy Engine)

(06) Erzeuge AUTHORIZATION_CODE (AS): Bei positiver Policy-Entscheidung generiert der ZETA Guard einen neuen Authorization Code (AUTH_CODE_AS) für den äußeren Flow.

(07) 302 Found (Redirect an App): Der Guard beantwortet den Request des Clients aus Schritt (01) mit einem HTTP 302 Redirect auf die <app_redirect_uri>, parametrisiert mit dem AUTH_CODE_AS und dem ursprünglichen state_app.

(08) DPoP-Schlüsselpaar generieren: Der ZETA Client fängt den Aufruf der app_redirect_uri ab. Er erzeugt im Arbeitsspeicher ein neues, asymmetrisches DPoP-Schlüsselpaar (PrK.DPoP.Sig, PuK.DPoP.Sig) sowie einen frischen DPoP Proof.

(09) POST /token am ZETA Guard: Der Client ruft den Token-Endpunkt des ZETA Guards auf (POST /token), setzt den HTTP-Header DPoP und authentisiert sich via Client Assertion (signiert mit PrK.Client.Sig). Im Body überträgt er den AUTH_CODE_AS, seinen code_verifier_app, seine client_id und die app_redirect_uri.

(10) Validierung durch den ZETA Guard: Der Guard verifiziert den code_verifier_app gegen die code_challenge_app (S256), validiert den DPoP Proof und verifiziert die Client Assertion (Prüfung der Key-Bindung aus der Dynamic Client Registration).

(11) Erzeuge Policy Engine Input: Der ZETA Guard aggregiert die extrahierten Identitäts-Claims des Versicherten, die im Vorfeld erfassten Geräte-Integritätsdaten (Posture) sowie Kontextmetadaten zu einem Bewertungs-Payload.

(12) POST /v1/data/authz: Der Guard ruft die PDP Policy Engine (OPA) auf und übergibt diesen Input.

(13) Policy Decision: Die Policy Engine evaluiert die unternehmensweiten Zugriffsrichtlinien und gibt ihre Entscheidung (allow oder deny) an den Authorization Server zurück.

3. Ausstellung des ZETA Tokens (Fall: [Policy Decision allow])

(14)-(16) Ausstellung ZETA Token: Der Guard antwortet mit HTTP 200 und stellt das finale, DPoP-gebundene Access Token (token_type=DPoP), ein Refresh Token und wenn

HW-Attestation verwendet wurde zusätzlich ein ZETA Guard Attestation Token (zeta_attestation_token) aus.

4. Fehlerpfad (Fall: [Policy Decision deny])

(17) 403 Forbidden: Entscheidet die Policy Engine auf Ablehnung, bricht der ZETA Guard den Prozess ab und beantwortet den Request des Clients aus Schritt (01) direkt mit HTTP 403 Forbidden unter Angabe der spezifischen Ablehnungsgründe (reasons).

A_29672 -ZETA, Ablauf Token-Bezug und Ausstellung der ZETA Token

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-OIDC-Token-Bezug* unterstützen.

[<=]

A_29842 -Authorization Server, ZETA Attestation Token

Der Authorization Server MUSS, zusätzlich zur Ausstellung von Access und Refresh Token, ein ZETA Attestation Token gemäß [zeta-attestation-token.yaml] ausstellen, wenn bei der Client Registrierung eine Hardware basierte Attestation verwendet und eine erfolgreiche Nutzer-Authentifizierung durchgeführt wurde.[<=]

5.10.2.5.6 Authentifizierung für eine andere Resource am gleichen ZETA Guard

Die Gültigkeitsdauer der Nutzer-Authentifizierung wird bei ZETA durch die Session des Authorization Servers abgebildet. Innerhalb einer gültigen Session kann der ZETA Client das Refresh Token einsetzen, um Zugriff auf andere durch den ZETA Guard geschützte Ressourcen (z. B. Notification Service Endpunkt oder Client Management Endpunkt) zu beantragen.

A_29843 -PDP Authorization Server, Zugriff auf andere Resource

Der Authorization Server MUSS Token Requests mit `grant_type=refresh_token` und Angabe der Parameter `resource` und `scope` verarbeiten, auch wenn das Refresh Token ursprünglich für eine andere `resource` und einen anderen `scope` beantragt wurde.[<=]

A_29952 -PDP Authorization Server, Parameter audience anstatt resource im Token Request

Der Authorization Server MUSS Token Requests mit dem Parameter `audience` anstatt `resource` unterstützen und die `audience` als `resource` Parameter interpretieren.

[<=]

A_29844 -PDP Authorization Server, Abfrage der Policy Engine bei Einsatz des Refresh Tokens

Der Authorization Server MUSS Token Requests mit `grant_type=refresh_token` und nach erfolgreicher Verifikation des Refresh Token, über die Policy Engine abfragen, ob neue Token ausgestellt werden dürfen.[<=]

A_29848 -PDP Authorization Server, Prüfung Session-Status bei Verwendung des Refresh Token

Der Authorization Server MUSS bei Token Requests mit `grant_type=refresh_token` vor der Ausstellung eines neuen Token-Sets prüfen, ob die referenzierte Session terminiert wurde. Ist die Session terminiert, MUSS der Authorization Server den Request ablehnen.

[<=]

5.10.3 Authentifizierung ohne Nutzer-Identität

Das folgende Sequenzdiagramm spezifiziert den Prozess zur Autorisierung von Client-Instanzen bei fachlichen Aufrufen, die keinen unmittelbaren Nutzerkontext erfordern (z. B. Abfrage des Verzeichnisdienstes). Der Nachweis der Berechtigung stützt sich ausschließlich auf die kryptografische Identität und die Geräteintegrität (Posture) des anfragenden ZETA Clients.

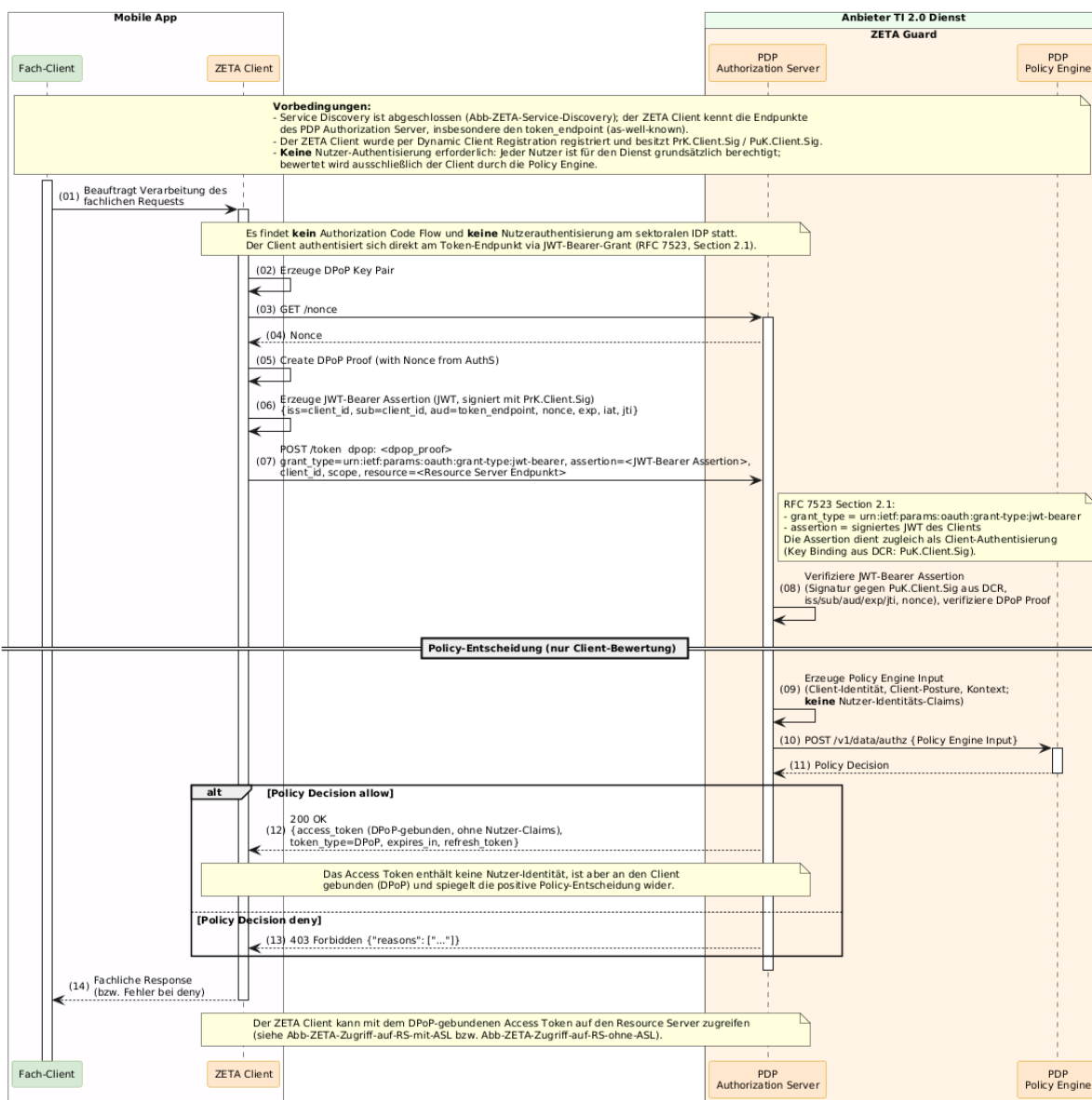


Abbildung 25 Abb-ZETA-OAuth-Client-Authentifizierung-ohne-Nutzer

Vorbedingungen:

Die Service Discovery ist abgeschlossen; der ZETA Client kennt die Endpunkte des ZETA Guards (insbesondere den token_endpoint).
 Der ZETA Client wurde über die Dynamic Client Registration (DCR) registriert und besitzt ein gerätegebundenes Signaturschlüsselpaar (PrK.Client.Sig / PuK.Client.Sig).
 Für den aufgerufenen Fachdienst ist keine Nutzerauthentisierung erforderlich; bewertet wird ausschließlich der Client durch die Policy Engine.

1. Initiierung und Vorbereitung der Client-Assertion (Schritte 01 bis 06)

- (01) Beauftragung: Der Fach-Client beauftragt den ZETA Client mit der Absicherung eines maschinellen Requests.
- (02) DPoP-Schlüsselpaar erzeugen: Der ZETA Client generiert im Arbeitsspeicher ein flüchtiges DPoP-Schlüsselpaar (PrK.DPoP.Sig, PuK.DPoP.Sig).
- (03) & (04) Nonce-Bezug: Der ZETA Client ruft proaktiv eine serverseitige Nonce beim ZETA Guard ab (GET /nonce), um Replay-Angriffe auf den nachfolgenden DPoP-Proof zu verhindern.

(05) DPoP Proof erstellen: Der Client erstellt den DPoP Proof unter Einbindung der erhaltenen Nonce.

(06) JWT-Bearer Assertion erzeugen: Der Client erstellt ein selbst-signiertes JWT gemäß RFC 7523 (Section 2.1) inklusive der Nonce und signiert dieses mit seinem privaten Instanz-Schlüssel (PrK.Client.Sig). Die Claims iss und sub entsprechen der eigenen client_id, der Claim aud der URL des Token-Endpunkts des ZETA Guards.

2. Token Request am ZETA Guard (Schritte 07 und 08)

(07) POST /token (JWT-Bearer-Grant): Der ZETA Client ruft den Token-Endpunkt des Guards auf (POST /token). Er setzt den HTTP-Header DPoP und beantragt über den Grant-Typ urn:ietf:params:oauth:grant-type:jwt-bearer ein Access Token. Im Body überträgt er die erzeugte assertion, seine client_id, den angeforderten scope sowie die Ziel-audience (im Bild noch resource).

(08) Validierung am AuthS: Der Authorization Server verifiziert die Signatur der JWT-Bearer Assertion gegen den bei der DCR hinterlegten öffentlichen Schlüssel (PuK.Client.Sig). Er prüft die zeitlichen Claims (iat, exp), schützt über den Claim jti vor Replays und validiert den DPoP Proof.

3. Reine Client-Policy-Entscheidung (Schritte 09 bis 11)

(09) Erzeuge Policy Engine Input: Da kein Nutzer involviert ist, aggregiert der Guard ausschließlich die Client-Identität, die übermittelten Posture-Daten des Geräts sowie Kontextmetadaten *umgesetzt* einem Bewertungs-Input. Nutzer-Identitäts-Claims werden nicht erstellt.

(10) & (11) Policy-Abfrage: Der Guard ruft die PDP Policy Engine auf (POST /v1/data/authz). Diese bewertet den Gerätezustand und gibt ihre Entscheidung (allow oder deny) zurück.

4. Token-Ausstellung & Rückgabe (Schritte 12 bis 14)

(12) 200 OK (Erfolgspfad): Bei positiver Policy-Entscheidung stellt der ZETA Guard ein DPoP-gebundenes Access Token (token_type=DPoP) und ein Refresh Token aus. Das Access Token enthält ausschließlich Client-Claims und keine Nutzer-Identität.

(13) 403 Forbidden (Fehlerpfad): Entscheidet die Policy Engine auf Ablehnung, abweist der Guard den Request unter Angabe der Ablehnungsgründe (reasons).

(14) Fachliche Rückgabe: Der ZETA Client quittiert dem aufrufenden Fach-Client den Vorgang (entweder durch Bereitstellung des abgesicherten Kanals/Tokens oder durch Weitergabe des Fehlers).

A 29733 -ZETA, Ablauf OAuth Client Authentifizierung ohne Nutzer

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung *Abb-ZETA-OAuth-Client-Authentifizierung-ohne-Nutzer* unterstützen. [<=]

5.10.4 Dienst-zu-Dienst Kommunikation

Für die sichere Maschine-zu-Maschine Interaktion zwischen Backends wird die Workload Identity Federation zur Authentifizierung verwendet. Ein Backend-Dienst authentifiziert sich mit einem Subject Token am token_endpoint des Ziel-Dienstes und erhält nach erfolgreicher Prüfung ein Access Token für den Zugriff auf den Ziel-Dienst.

Wenn eine ZETA Guard geschützte Workload (z. B. der Resource Server) auf einen externen Dienst zugreift, der eine Authentifizierung gemäß Workload Identity Federation verlangt, dann stellt der ZETA Guard Authorization Server als OpenID Provider die erforderlichen Subject Token aus. Der ZETA Guard erstellt regelmäßig einen cronjob Pod, der ein Subject Token vom Authorization Server bezieht und dieses per Token Exchange beim Zieldienst in ein Access Token tauscht. Das Access Token wird der Workload als Secret bereitgestellt und jeweils vor Ablauf der Gültigkeit durch erneuten Start des cronjob Pods und Token Exchange erneuert. Dadurch hat die Workload permanent Zugriff auf ein gültiges Access Token und kann mit dem externen Ziel-Dienst kommunizieren.

Wenn ein ZETA Guard geschützter Resource Server als Ziel-Dienst genutzt wird, dann übernimmt der ZETA Guard Authorization Server die Rolle eines Secure Token Service in der Workload Identity Federation und tauscht beim Token Exchange das Subject Token eines anfragenden Dienstes gegen ein Access Token für den Zugriff auf den Resource Server.

A_28431 -ZETA Guard, Ablauf Dienst-zu-Dienst Kommunikation

Der ZETA Guard MUSS den Ablauf gemäß Abbildung Abb-ZETA-Dienst-zu-Dienst-Kommunikation unterstützen. [<=]

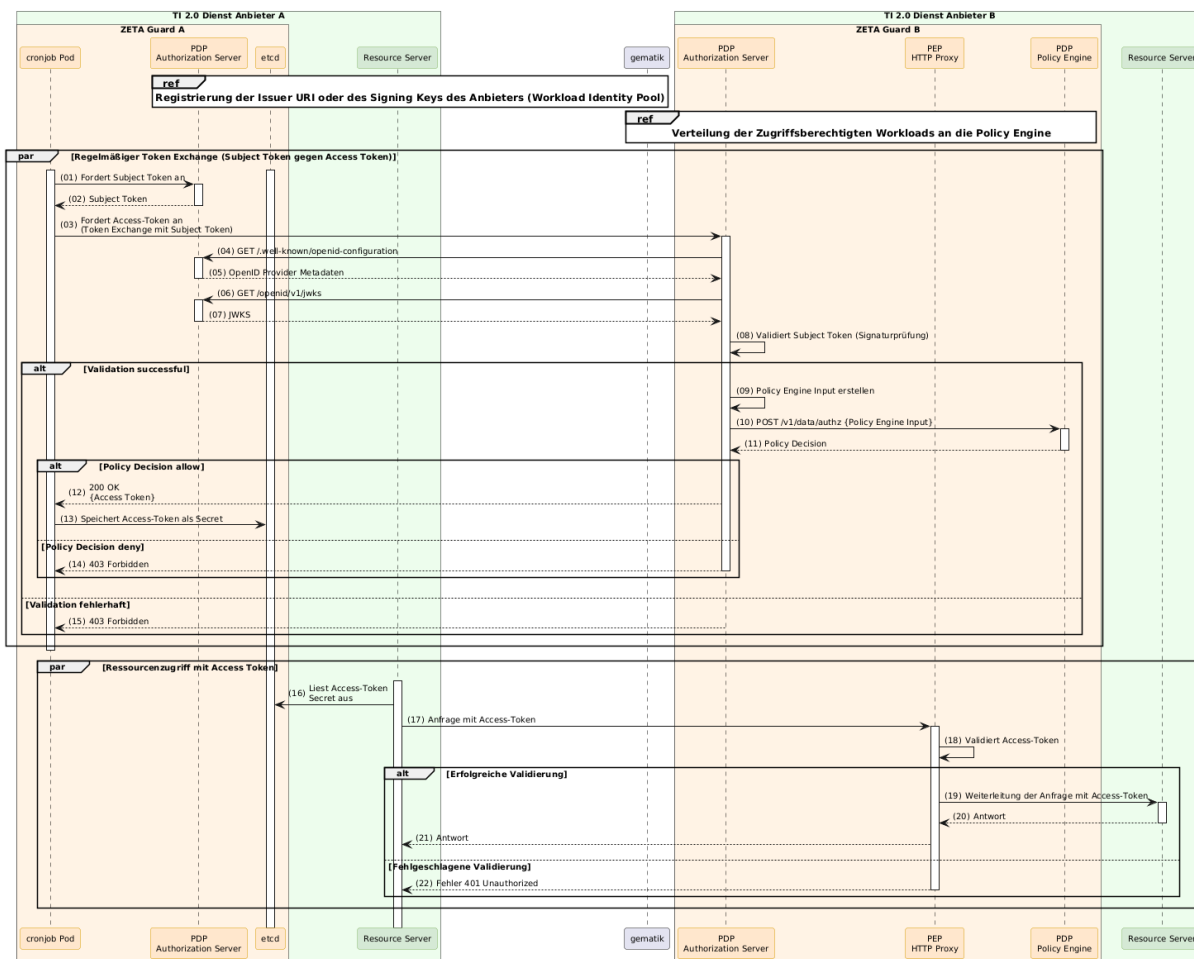


Abbildung 26 : Abb-ZETA-Dienst-zu-Dienst-Kommunikation

Voraussetzung für die Dienst-zu-Dienst-Kommunikation ist:

- Die Registrierung der Issuer URI oder des Signing Keys des Anbieters im Workload Identity Pool (unter Einbindung der gematik).
- Die Verteilung der zugriffsberechtigten Workloads an die Policy Engine von Anbieter B.

Regelmäßiger Token Exchange (Subject Token gegen Access Token)

Dieser parallele Block (par) beschreibt, wie Anbieter A ein gültiges Access-Token von Anbieter B erhält:

Lokaler Token-Abruf (01-02): Der cronjob Pod fordert ein lokales Subject Token vom eigenen PDP Authorization Server (Anbieter A) an und erhält dieses.

Token-Austausch-Anfrage (03): Der cronjob Pod sendet dieses Subject Token an den PDP

Authorization Server von Anbieter B, um es gegen ein Access-Token einzutauschen.
Metadaten- & Schlüsselabruf (04-07): Zur Verifizierung des Tokens ruft der
Autorisierungsserver von Anbieter B die OpenID-Konfiguration und die Signaturschlüssel
(JWS) direkt vom Autorisierungsserver von Anbieter A ab.
Validierung (08): Der Server von Anbieter B validiert die Signatur des Subject Tokens.

Falls die Validierung erfolgreich ist wird ein Input für die Policy Engine erstellt (09).
Dieser wird per POST-Request an die PDP Policy Engine übermittelt (10), welche eine
Entscheidung zurückgibt (11).
Entscheidung „allow“ bewertet wer(12-13): Anbieter B antwortet mit 200 OK und liefert
das Access Token aus. Der cronjob Pod von Anbieter A speichert dieses Token als Secret
im lokalen etcd (oder in einem anderen sicheren Speicher).
Entscheidung „deny“ (14): Es wird ein 403 Forbid~~en~~ *kann, wenn* zurückgegeben.
Falls die Validierung fehlschlägt (15): Es wird direkt ein 403 Forbiden an den cronjob Pod
zurückgesendet.

Ressourcenzugriff mit Access Token

Der zweite parallele Block (par) beschreibt den eigentlichen Zugriff auf die geschützte
Ressource:

Token auslesen (16): Der Resource Server von Anbieter A liest das zuvor im etcd
gespeicherte Access-Token aus.

Anfrage senden (17): Der Resource Server von Anbieter A stellt eine Anfrage inklusive
des Access-Tokens an den PEP HTTP Proxy von Anbieter B.

Token-Prüfung (18): Der Proxy validiert das Access-Token.

Abzweigung (alt-Block):

Erfolgreiche Validierung (19-21): Die Anfrage wird an den eigentlichen Resource Server
von Anbieter B weitergeleitet. Dieser antwortet, und die Antwort wird über den Proxy *an*
die anwendbaren Abschnitte der BSI- Resource Server von Anbieter A zurückgegeben.

Fehlgeschlagene Validierung (22): Der Proxy blockiert die Anfrage und liefert einen Fehler
401 Unauthorized zurück.

5.10.4.1 Ausgehende Verbindungen der Dienst-zu-Dienst **Kommunikation mit ZG Client**

5.10.4.1.1 Motivation und Abgrenzung

Die Dienst-zu-Dienst-Kommunikation funktioniert auf Basis eines regelmäßig laufenden
Cronjob-Pods, der ein Subject Token bezieht, per Token Exchange gegen ein Access
Token tauscht und dieses als Secret für die Workload bereitstellt. Dieses Muster eignet
sich für Zielsysteme, die selbst Workload Identity Federation (WIF) gemäß RFC 8693
anbieten, erfordert jedoch, dass jede aufrufende Workload den Lebenszyklus des Secrets
(Ablage, Rotation, Aktualisierung) kennt und im Fehlerfall behandelt.

Um Resource Server und andere ZETA Guard Komponenten (z. B. den Telemetriedaten-
Service) vollständig von dieser Aufgabe zu entlasten, wird eine neue Komponente ZG
Client (ZETA Guard Client) eingeführt. Der ZG Client übernimmt für ausgehende Aufrufe
an externe Dienste - diese müssen kein TI-Dienst sein - alle notwendigen Schritte der
Authentifizierung mittels Workload Identity Federation. Die aufrufende Fachkomponente
sendet ausschließlich den fachlichen Request an den ZG Client; dieser reichert den
Request um ein gültiges Access Token an und leitet ihn an den Zieldienst weiter. Die
Fachkomponente muss weder Token-Beschaffung noch -Erneuerung noch
Schlüsselmateriale verwalten.

Der ZG Client ist damit das Pendant zum ZETA Client (Kapitel 5.4) auf der Seite
ausgehender Server-zu-Server-Kommunikation. Während der ZETA Client die Absicherung
von Anfragen eines Clientsystems gegenüber einem ZETA Guard übernimmt, übernimmt
der ZG Client die Absicherung ausgehender Anfragen einer serverseitigen Workload
gegenüber einem beliebigen externen Dienst.

5.10.4.1.2 Architektur

Der ZG Client wird als eigenständiger Prozess (z. B. Sidecar-Container im selben Pod oder lokal erreichbarer Proxy) neben der Fachkomponente betrieben. Er stellt einen lokalen Endpunkt (Egress-Proxy) bereit, an den die Fachkomponente ihre ausgehenden Requests adressiert. Der ZG Client:

- ermittelt anhand der Ziel-Adresse des Requests die zuständige Konfiguration (Zieldienst, erforderlicher Scope/Audience, Token-Endpunkt des Zieldienstes, Subject-Token-Quelle),
- bezieht bei Bedarf ein Subject Token vom lokalen ZETA Guard Authorization Server und tauscht das Subject Token per Token Exchange (RFC 8693) beim Zieldienst gegen ein Access Token,
- cached das Access Token bis kurz vor Ablauf der Gültigkeit und erneuert es danach automatisch im Hintergrund,
- fügt das Access Token dem fachlichen Request hinzu und leitet diesen an den Zieldienst weiter,
- gibt die Antwort des Zieldienstes unverändert an die Fachkomponente zurück.

Im Unterschied zum Cronjob-Pod-Muster erfolgt die Token-Beschaffung beim ZG Client requestgetrieben bzw. proaktiv im Hintergrund innerhalb desselben Prozesses, ohne Ablage des Access Tokens als Kubernetes Secret.

5.10.4.1.3 Konfiguration über die Policy Engine

Die Konfiguration des ZG Client (zulässige Zieldienste, Scopes/Audiences, Token-Endpunkte, Caching-Parameter) erfolgt nicht statisch, sondern wird zur Laufzeit über die Policy Engine (PAP/PDP) bereitgestellt. Dadurch lassen sich Berechtigungen für ausgehende Verbindungen zentral pflegen, ohne Deployments der Fachkomponenten anzupassen, und Änderungen wirken sich ohne Neustart des ZG Client aus.

5.10.4.1.4 Ablauf

Das Sequenzdiagramm beschreibt die maschinelle Dienst-zu-Dienst-Kommunikation über System- und Anbietergrenzen hinweg (hier: von Anbieter A zu Anbieter B) mittels ZG Client.

Es visualisiert das Konzept der Workload Identity Federation. Dabei agiert der ZETA Guard des aufrufenden Anbieters (A) als lokaler Identity Provider (IdP) und stellt ein Subject Token aus. Dieses Token wird über einen OAuth 2.0 Token Exchange beim Secure Token Service (STS) des fremden Anbieters (B) gegen ein dort gültiges Access Token eingetauscht. Den Prozess steuert der ZETA Guard Client (ZG Client), der typischerweise als Egress-Proxy (Sidecar) oder SDK für die Fachkomponente fungiert.

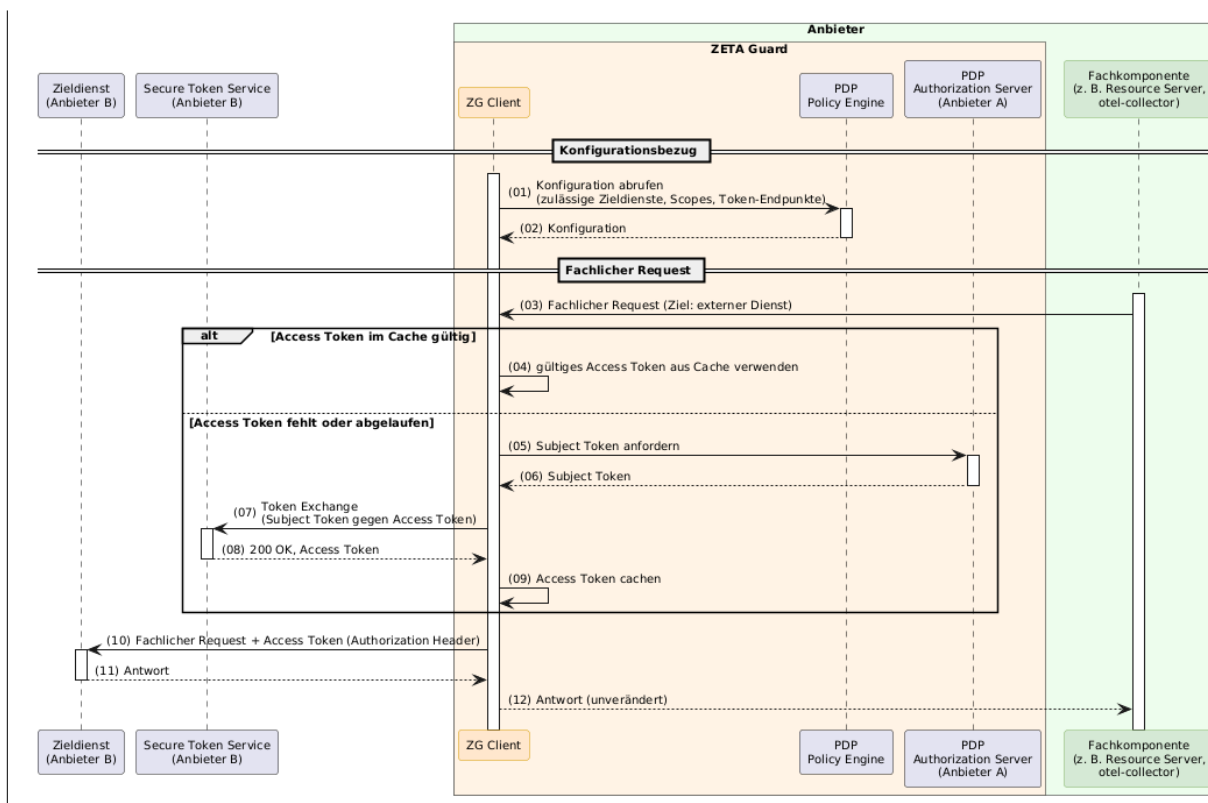


Abbildung 27 Abb-ZETA-Dienst-zu-Dienst-Kommunikation-mit-ZG-Client

1. Konfigurationsbezug

Bevor der ZG Client externe Aufrufe absichern kann, muss er die geltenden Föderationsrichtlinien kennen. Die Konfiguration wird regelmäßig aktualisiert.
 (01) Konfiguration abrufen: Der ZG Client fragt die lokale PDP Policy Engine (von Anbieter A) nach den aktuellen Konfigurationsdaten ab.
 (02) Konfiguration: Die Policy Engine liefert eine Liste der zulässigen Zieldienste, die erlaubten Scopes sowie die Endpunkt-URLs der fremden Secure Token Services (STS), denen vertraut wird.

2. Initiierung des fachlichen Requests

(03) Fachlicher Request: Eine interne Fachkomponente (z. B. ein Resource Server oder ein Telemetrie-Dienst wie der otel-collector) initiiert einen Aufruf, der an einen externen Dienst (Zieldienst bei Anbieter B) gerichtet ist. Der ZG Client fängt diesen Request ab bzw. nimmt ihn entgegen, um die Autorisierungsschicht transparent hinzuzufügen.

3. Token-Management und Token Exchange

Der ZG Client prüft nun, ob er bereits über ein gültiges Access Token für das angeforderte externe Zielsystem verfügt.

Pfad A: Access Token im Cache gültig

(04) Token aus Cache verwenden: Liegt ein gültiges Access Token für den Zieldienst im lokalen Cache des ZG Clients vor, wird dieses direkt für den Aufruf verwendet (Weiter mit Schritt 10).

Pfad B: Access Token fehlt oder ist abgelaufen (Workload Identity Federation)

(05) Subject Token anfordern: Der ZG Client ruft den lokalen PDP Authorization Server (ZETA Guard von Anbieter A) auf und fordert ein Identitätstoken für den eigenen Dienst an.

(06) Subject Token: Der lokale AuthS stellt ein signiertes Subject Token aus. Dieses Token bestätigt die Identität der lokalen Fachkomponente gegenüber der Außenwelt.

(07) Token Exchange: Der ZG Client ruft den externen Secure Token Service (STS) von Anbieter B auf. Er nutzt den OAuth 2.0 Token Exchange (RFC 8693) und reicht das

Subject Token ein, um im Gegenzug ein lokales Access Token von Anbieter B zu erhalten.
(08) 200 OK (Access Token): Der fremde STS validiert das Subject Token (basierend auf der Föderations-Vertrauensstellung), akzeptiert es und stellt ein neues Access Token aus, das für den Zugriff auf den Zieldienst bei Anbieter B berechtigt.

(09) Access Token cachen: Der ZG Client speichert das neu erhaltene Access Token in seinem lokalen Cache, um Folgeaufrufe zu beschleunigen.

4. Ausführung des fachlichen Requests

(10) Fachlicher Request + Access Token: Der ZG Client fügt das (entweder aus dem Cache stammende oder frisch eingetauschte) Access Token als Authorization: DPoP <Token> Header in den ursprünglichen HTTP-Request ein und leitet diesen an den externen Zieldienst (Anbieter B) weiter.

(11) Antwort: Der externe Zieldienst verarbeitet die Anfrage und sendet die fachliche HTTP-Antwort zurück an den ZG Client.

(12) Antwort (unverändert): Der ZG Client reicht die empfangene Antwort transparent und unverändert an die aufrufende interne Fachkomponente zurück.

5.10.4.1.5 Anforderungen

A_29878 -ZETA Guard, Unterstützung des Ablaufs der Dienst-zu Dienst Kommunikation mit ZG Client

Der ZETA Guard MUSS für ausgehende Dienst-zu-Dienst-Kommunikation die Bereitstellung einer ZG Client-Komponente gemäß Abbildung *Abb-ZETA-Dienst-zu-Dienst-Kommunikation-mit-ZG-Client* unterstützen. [FR-03161-1]<=]

A_29879 -ZG Client, Transparenz für Fachkomponente

Der ZG Client MUSS fachliche Requests einer aufrufenden Fachkomponente entgegennehmen, ohne dass diese Token-Beschaffung, Token-Erneuerung oder Schlüsselmaterial selbst verwalten muss. [<=]

A_29880 -ZG Client, Authentifizierung mittels Workload Identity Federation

Der ZG Client MUSS sich gegenüber externen Diensten mittels Workload Identity Federation gemäß RFC 8693 (Token Exchange) authentifizieren. [<=]

Hinweis: Der externe Zieldienst muss nicht zwingend ein ZETA Guard geschützter TI-Dienst sein. Es werden jedoch nur externe Dienste unterstützt, zu denen alle TI 2.0 Fachdienste des gleichen Typs eine Dienst-zu-Dienst Kommunikation benötigen.

A_29881 -ZG Client, Konfiguration über die Policy Engine

Der ZG Client MUSS seine Konfiguration (zulässige Zieldienste, Scopes/Audiences, Token-Endpunkte sowie Caching-Parameter) zur Laufzeit von der Policy Engine beziehen und für die in der Response der Policy Engine angegebene Gültigkeitsdauer cachen. Nach Ablauf der Gültigkeitsdauer MUSS der ZG Client die aktuelle Konfiguration von der Policy Engine beziehen. Eine ausschließlich statische, lokale Konfiguration ist nicht zulässig. [<=]

A_29882 -ZG Client, Aktualisierung der Konfiguration

Der ZG Client MUSS Änderungen der von der Policy Engine bereitgestellten Konfiguration zeitnah übernehmen, ohne dass hierfür ein Neustart des ZG Client erforderlich ist. [<=]

A_29883 -ZG Client, Caching und Erneuerung von Access Tokens

Der ZG Client MUSS bezogene Access Tokens bis zu deren Ablauf zwischenspeichern und VOR Ablauf der Gültigkeit automatisiert erneuern, sodass für die Fachkomponente durchgehend ein gültiges Access Token zur Verfügung steht. [<=]

A_29884 -ZG Client, Fehlerbehandlung

Lehnt der Secure Token Service des Zieldienstes den Token Exchange ab oder schlägt die Validierung des Subject Tokens fehl, so MUSS der ZG Client den Fehler unverändert (inkl. HTTP-Statuscode) an die aufrufende Fachkomponente weiterleiten. [≤]

A_29885 -ZG Client, Geltungsbereich der Berechtigung

Der ZG Client MUSS Access Tokens ausschließlich für die in der Policy-Engine-Konfiguration als zulässig hinterlegten Zieldienste und Scopes beziehen und verwenden. [≤]

A_29886 -ZG Client, Erreichbarkeit

Der ZG Client MUSS für die ihm zugeordnete(n) Fachkomponente(n) als lokal erreichbarer Egress-Endpunkt bereitgestellt werden. [≤]

5.10.5 Token Revocation

Das Sequenzdiagramm spezifiziert den Prozess der Token Revocation (Token-Widerruf), welcher bei einer nutzerinitiierten Abmeldung (Logout) durchlaufen wird.

Das Diagramm setzt die Vorgaben aus RFC 7009 (OAuth 2.0 Token Revocation) im Kontext der ZETA-Architektur um. Ziel ist es, aus einem reinen lokalen Logout der App eine echte, serverseitige Terminierung der Sitzung (Grant-Terminierung) beim ZETA Guard zu machen.

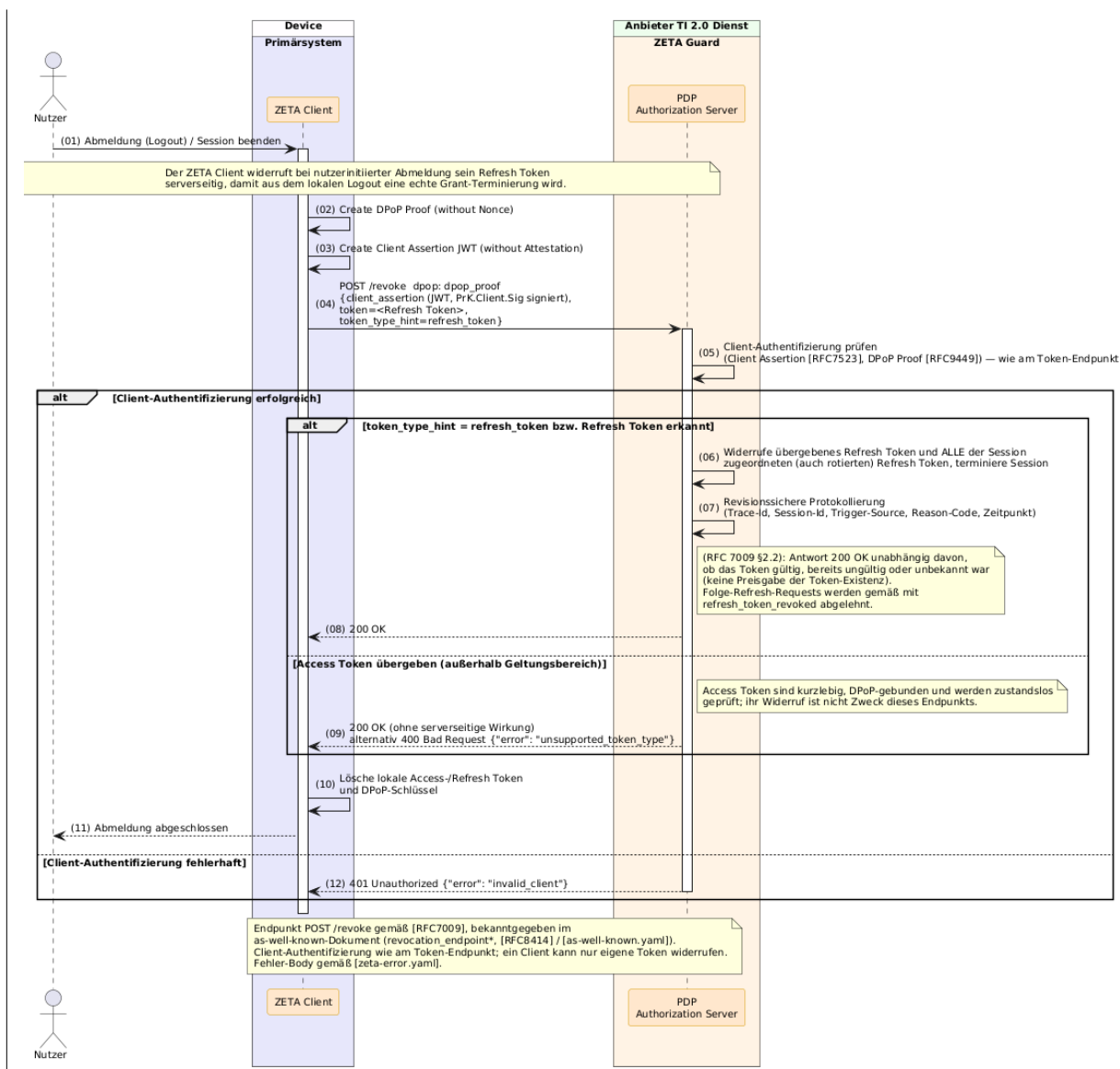


Abbildung 28 Abb-ZETA-Token-Revocation

Ausgangssituation: Der Nutzer ist am ZETA Client angemeldet, die App besitzt ein gültiges Refresh Token und ein zugehöriges DPoP-Schlüsselpaar.

1. Initiierung und Request-Vorbereitung

(01) Abmeldung: Der Nutzer initiiert in der Benutzeroberfläche des Primärsystems (ZETA Client) die Abmeldung. Gemäß Anforderung A_30002 *aus Sicht des Gutachters erfüllt sind*.

Gist der Client nun verpflichtet, das Refresh Token serverseitig zu widerrufen.

(02) DPoP Proof: Der Client erstellt einen DPoP Proof für den /revoke-Endpoint. (Da dieser Endpoint in der Regel keine Nonce erfordert, wird der Proof ohne Nonce generiert).

(03) Client Assertion: Der Client generiert ein JWT (gemäß RFC 7523) zur eigenen Authentisierung und signiert dieses mit seinem privaten Instanzschlüssel (PrK.Client.Sig). Hardware-Attestierungsdaten (Evidence) sind für diesen Endpoint nicht erforderlich.

(04) POST /revoke: Der Client sendet den Widerrufs-Request an den Token Revocation Endpoint des ZETA Guards. Er übergibt den DPoP Proof im Header sowie die client_assertion, das zu widerrufende token (das Refresh Token) und den Hinweis token_type_hint=refresh_token im Body.

2. Authentisierung am ZETA Guard

(05) Client-Authentifizierung prüfen: Der ZETA Guard prüft die Identität und Berechtigung des aufrufenden Clients. Analog zum Token-Endpunkt werden die Client Assertion und der DPoP Proof kryptografisch validiert. (Ein Client darf ausschließlich seine eigenen Tokens widerrufen).

Der weitere Ablauf verzweigt sich anhand des Ergebnisses der Client-Authentifizierung (A_29997).

5.11-3. Erfolgreiche Clientsystem und ZETA Client

Ein Clientsystem ist eine Softwarekomponente **Authentifizierung (Hauptpfad)**

Wenn die Authentifizierung des Clients erfolgreich war, prüft der Server, welcher Token-Typ widerrufen werden soll.

Pfad A: Refresh Token erkannt (Der Standardfall)

(06) Token und Session terminieren: Der ZETA Guard identifiziert das übergebene Refresh Token. Gemäß A_29998 widerruft er nicht nur dieses spezifische Token, sondern alle der aktuellen Session zugeordneten (auch bereits rotierten) Refresh Tokens. Die Sitzung wird serverseitig endgültig beendet.

(07) Protokollierung: Der ZETA Guard protokolliert den Vorgang revisions sicher (inkl. Trace-Id, Session-Id, Trigger-Source und Reason-Code) (A_29857).

(08) 200 OK: Der ZETA Guard antwortet mit HTTP 200 OK.

(Wichtiger Hinweis gem. RFC 7009 §2.2 / A_29999: Diese Erfolgsmeldung wird immer gesendet, unabhängig davon, ob das Token gültig, bereits abgelaufen oder dem Server völlig unbekannt war. Dies verhindert das Ausspähen von Token-Zuständen durch Angreifer).

Pfad B: Access Token übergeben (Außerhalb des Geltungsbereichs)

(09) Umgang mit Access Tokens: ZETA Access Tokens sind extrem kurzlebig, zustandslos und DPoP-gebunden. Ein serverseitiger Widerruf (Revocation List) ist für sie nicht vorgesehen (A_30000). Sendet der Client versehentlich ein Access Token, antwortet der Server entweder mit 200 OK (ohne serverseitige Wirkung) oder weist den Request mit 400 Bad Request ("unsupported_token_type") ab.

Abschluss auf Client-Seite

(10) Lokale Bereinigung: Unabhängig von der Antwort des Servers (solange es kein 401/403 ist), löscht der ZETA Client nach dem Aufruf das Refresh Token, etwaige Access Tokens sowie die zugehörigen flüchtigen DPoP-Schlüssel aus seinem lokalen Speicher (A_30002).

(11) Abmeldung abgeschlossen: Der ZETA Client meldet dem Nutzer den erfolgreichen Logout.

4. Fehlerhafte Client-Authentifizierung (Fehlerpfad)

(12) 401 Unauthorized: Schlägt die Signaturprüfung der Client Assertion oder des DPoP Proofs fehl (Schritt 05), bricht der ZETA Guard ab und antwortet mit 401 Unauthorized (Fehlercode invalid_client).

A_30001 -ZETA, Ablauf Token Revocation

Der ZETA Guard und der ZETA Client MÜSSEN den Ablauf gemäß Abbildung Abb-ZETA-Token-Revocation unterstützen. [<=]

5.12 Clientsystem und ZETA Client

Ein Clientsystem ist eine Softwarekomponente in der Verwendung eines Nutzers zum Ausführen fachlicher Anwendungsfälle z.B. als Primärsystem (PVS, AVS, LIS, KIS etc.) oder

als Frontend des Versicherten (ePA-App, E-Rezept-App, TI-Messenger etc.). Dieses Clientsystem wird dem Nutzer durch einen Hersteller zur Verfügung gestellt.

Ein ZETA Client ist ein Teil des Clientsystems, der die Kommunikation mit dem PDP und dem PEP eines Dienstes übernimmt.

Mobile iOS und Android Apps werden unterstützt und setzen ebenfalls einen ZETA Client um. Anforderungen, die nur für diese Clientsysteme und ZETA Clients gelten werden, verwenden die Bezeichnung "mobiler Client" oder "mobiles Clientsystem" und "mobiler ZETA Client".

5.12.1 Hersteller

A_25335 -Hersteller Clientsystem - Hinweise und Maßnahmen sicherer Betrieb

Der Hersteller eines Clientsystems MUSS den Nutzer über Maßnahmen zum sicheren Betrieb seines Clientsystems vor der Inbetriebnahme informieren und während des Betriebs stets zum Abruf durch den Nutzer bereithalten. [\leq]

A_25336 -Hersteller Clientsystem - Regelmäßige Updates

Der Hersteller eines Clientsystems MUSS, solange das Produkt nicht abgekündigt ist, dem Nutzer regelmäßig (z. B. quartalsweise) Updates für das Clientsystem bereitstellen, um das Clientsystem dauerhaft auf dem Stand der Technik zu halten und Sicherheitslücken zu schließen. [\leq]

A_25337 -Hersteller Clientsystem - Registrierung für product_id

Der Hersteller eines Clientsystems MUSS sich über einen organisatorischen Prozess bei der gematik für die Nutzung von Diensten, für welche Token abgerufen werden sollen, registrieren. Der Hersteller eines Clientsystems bekommt dabei eine "product_id" zugewiesen, die in jeder Instanz des Clientsystems verwendet werden MUSS. [\leq]

5.12.2 Verbindungsaufbau

Mainline_OPB1/ML-1888A_29723 -Hersteller Clientsystem, Meldung von Client-Metadaten

Der Hersteller eines Clientsystems MUSS über einen Prozess der gematik die folgenden Client-Metadaten an die gematik melden:

- fortlaufend im Betrieb alle Änderungen von aktiv unterstützten Versionen seines Clients (Claim product_version)
- bei Nutzung von TPM Attestation, die kryptografischen Hashes der unveränderlichen Bestandteile (Immutable Code / Binärdateien) jeder gemeldeten Produktversion des Clients
- bei mobilen Applikationen:
 - die oidc_redirect_uri nach dem Schema /cb/<app>/oidc (für die Weiterleitung an den Redirection-Endpunkt des ZETA Guards),
 - die app_redirect_uri nach dem Schema /cb/<app>/app (für die Weiterleitung an den Token-Endpunkt des ZETA Guards).

[\leq]

5.12.3 Verbindungsaufbau

Bevor ein ZETA Client auf geschützte Fachdienste zugreifen kann, muss der netzwerktechnische Verbindungsaufbau sowie die korrekte Adressierung und Autorisierung über Token-Strukturen sichergestellt werden. Die nachfolgenden

Definitionen bilden die Grundlage für die Ausgestaltung der Token-Requests und den Einsatz des ZETA/ASL-Kanals.

5.12.3.1 Definition der Endpunkte

Für den Verbindungsaufbau und die Anforderung von Token werden die folgenden Endpunkte unterschieden:

Resource Server Endpunkt: Der finale Endpunkt des aufgerufenen Fachdienstes. Die URL dieses Endpunkts MUSS mit dem Parameter `resource` der OAuth Protected Resource Metadata ([`opr-well-known.yaml`]) des jeweiligen Dienstes übereinstimmen.

ZETA/ASL Endpunkt: Der fest definierte Endpunkt für die Terminierung des ZETA Application Security Layer (ASL) Protokolls. Die URL dieses Endpunkts MUSS nach dem Schema `https://<hostname>/ASL` gebildet werden, wobei `<hostname>` exakt dem Hostnamen des ermittelten Resource Server Endpunkts entspricht.

ZETA Guard Token Endpunkt: Der Autorisierungsendpunkt des ZETA Guards (Authorization Server), an den der Token Exchange Request gesendet wird.

5.12.3.2 Adressierung und Gültigkeitsbereich (Audience und Scope)

Um sicherzustellen, dass Tokens nur für den vorgesehenen Empfänger gültig sind, muss bei der Token-Anforderung strikt zwischen Parametern des HTTP-Requests und den Claims innerhalb der JWT-Strukturen unterschieden werden.

Claim aud im Subject Token: Das Subject Token wird an den ZETA Guard gesendet, damit dieser es auswertet. Folglich MUSS der Claim `aud` (Audience) innerhalb des Subject Tokens exakt die URL des ZETA Guard Token Endpunkts enthalten.

Parameter resource im Token Request (HTTP Body): Dieser Parameter gemäß [RFC8707] enthält die absolute URL des aufgerufenen Endpunkts am Resource Server und wird der OAuth Protected Resource Metadata ([`opr-well-known.yaml`], Feld `resource`) entnommen. Der ZETA Guard ermittelt aus dem angeforderten `resource` über die Policy Engine den logischen Bezeichner der Zielressource und stellt das Access Token mit diesem Wert im Claim `aud` aus (siehe 5.4.2.4). Die Ressourcen-URL wird NICHT in den `aud`-Claim übernommen; sie wird beim späteren Aufruf der Ressource über den Claim `htu` des DPoP Proofs [RFC9449] an den konkreten Request gebunden.

Parameter audience im Token Request (HTTP Body): Aus Gründen der Abwärtskompatibilität DARF ein ZETA Client anstelle von `resource` den Parameter `audience` (logischer Bezeichner) senden. In diesem Fall wird der angegebene Wert unverändert als `aud`-Claim in das Access Token übernommen (Legacy-Verhalten, siehe 5.4.2.5). Der Parameter `audience` ist als veraltet gekennzeichnet; neue ZETA Clients SOLLEN ausschließlich `resource` verwenden.

Parameter scope im Token Request (HTTP Body): Dieser Parameter MUSS die spezifischen Berechtigungen (Scopes) anfordern, die für den Zugriff auf den unter `resource` definierten Endpunkt erforderlich sind. Die unterstützten Scopes werden der OAuth Protected Resource Metadata ([`opr-well-known.yaml`], Feld `scopes_supported`) entnommen.

5.12.3.3 Zusammenhang der Token Exchange Komponenten

Stationäre ZETA Clients nutzen für den Bezug eines Access Tokens den OAuth 2.0 Token Exchange Grant [RFC 8693]. Ein vollständiger, valider Token Request an den ZETA Guard (POST /token) setzt sich aus vier ineinandergreifenden kryptografischen Komponenten zusammen:

Der Token Request (HTTP Body): Die Klammer um alle Parameter. Er definiert über `grant_type=urn:ietf:params:oauth:grant-type:token-exchange` die Art der Anfrage, benennt über `resource` [RFC8707] die Endpunkt-URL der Zielressource und überscope die angeforderten Berechtigungen und bindet das Subject Token sowie die Client Assertion ein. Aus der angegebenen `resource` leitet der ZETA Guard über die Policy Engine den logischen `aud-Claim` ab; die `resource-URL` selbst wird nicht in den `aud-Claim` übernommen, sondern bestimmt den späteren Ziel-Endpunkt (Bezug zum DPoP-Claim `htu`, siehe 5.4.2.4).

Das Subject Token: Ein JWT, welches den Kontext der Anfrage (z.B. den Nutzer oder die Session) repräsentiert. Es ist (wie oben beschrieben) über den `aud-Claim` an den ZETA Guard Token Endpunkt gebunden.

Die Client Assertion: Ein JWT (`client_assertion_type=...jwt-bearer`), welches die Authentisierung und Integrität (Attestierung) des ZETA Clients gegenüber dem ZETA Guard nachweist. Es enthält das `client_statement` mit den plattformspezifischen Integritätsnachweisen (z. B. Apple App Attest oder das Software-Fallback).

Der DPoP Proof für den Request: Ein über den HTTP-Header (DPoP) übergebenes JWT. Es beweist, dass der Client im Besitz des privaten DPoP-Schlüssels ist. In diesem DPoP-Proof MUSS der `Claim htm` den Wert `POST` und der `Claim htu` die URL des ZETA Guard Token Endpunkts enthalten (da dieser Request an den Guard gerichtet ist).

Hinweis: Über den Parameter `resource` können in getrennten Token Requests verschiedene Access Tokens für unterschiedliche Zielressourcen bezogen werden. Alle so bezogenen Access Tokens werden über denselben DPoP-Schlüssel an die Client-Instanz gebunden (`Claim jkt`); sie unterscheiden sich im logischen `aud-Claim`, den die Policy Engine je Zielressource vergibt.

5.12.3.4 Resultierende Token für den Aufruf von Fachdiensten

Nach erfolgreicher Validierung des Token Requests stellt der ZETA Guard ein Access Token aus. Der `aud-Claim` dieses Access Tokens enthält den von der Policy Engine festgelegten logischen Bezeichner der Zielressource, der aus dem im Request angegebenen Parameter `resource` abgeleitet wird. Tokens, die auf diesem Weg ausgestellt werden, tragen den `Claim ver` mit dem Wert `2`. Wurde der Token Request im Legacy-Verfahren über den Parameter `audience` gestellt, so entspricht der `aud-Claim` dem verbatim übernommenen `audience`-Wert und das Token trägt `ver` mit dem Wert `1` (siehe 5.4.2.5).

Wenn der ZETA Client anschließend den eigentlichen Request an den Resource Server (oder den ZETA/ASL Endpunkt) durchführt, MUSS er einen neuen DPoP Proof generieren. Für diesen neuen DPoP Proof gilt zwingend: Der `Claim htu` MUSS exakt der URL des aufgerufenen Endpunkts entsprechen gemäß [RFC9449]; wurde im Token Request der Parameter `resource` angegeben, MUSS `htu` mit dieser URL übereinstimmen. Während der `aud-Claim` die Zielressource über ihren logischen Bezeichner benennt, adressiert der `htu-Claim` den konkret aufgerufenen Endpunkt und bindet das DPoP-gebundene Access Token an den tatsächlichen Request. Der `claim htm` MUSS der HTTP Methode des fachlichen Requests entsprechen.

Wenn im OAuth Protected Resource Well-known JSON Dokument angegeben ist `zeta_asl_use: required_passthrough`, dann hat der ZETA Client beim Token Request neben dem Access und Refresh Token für den Zugriff auf den Resource Server auch ein zusätzliches Access Token für den Zugriff auf den /ASL Endpunkt erhalten (Parameter `asl_access_token`).

A_29862 -ZETA Client, zusätzliches DPoP Token bei ZETA/ASL Terminierung am Resource Server

Der ZETA Client MUSS beim durchführen eines fachlichen Requests zum Resource Server ein zusätzliches DPoP Token für den /ASL Endpunkt gemäß Tabelle Tab-ZETA-Token-Ausstellung-ASL-am-Resource-Server erzeugen und im äußeren HTTP Request im DPoP Header verwenden. [<=]

5.12.3.5 Versionierung des Token-Ausstellungsverfahrens

Da bereits produktive ZETA Clients und ZETA Guards im Einsatz sind, MUSS das Verfahren zur Ableitung des aud-Claims versioniert und abwärtskompatibel betrieben werden. Es werden zwei Vertragsversionen (Contract Versions) unterschieden:

- Contract v1 (Legacy): Der ZETA Client sendet den Parameter audience (logischer Bezeichner). Der ZETA Guard übernimmt diesen Wert verbatim in den aud-Claim und stellt das Access Token mit dem Claimver: 1 aus. Fehlt der Claimver, ist das Token wiever: 1 zu behandeln.
- Contract v2: Der ZETA Client sendet den Parameter resource [RFC8707] sowie scope. Der ZETA Guard leitet den logischen aud-Claim aus der Policy Engine Decision ([pdp-decision.yaml], Feld aud) ab und stellt das Access Token mit dem Claimver: 2 aus.

Der ZETA Guard Authorization Server MUSS die von ihm unterstützten Vertragsversionen in seinem OAuth Authorization Server Well-known JSON Dokument ([as-well-known.yaml], Feld api_versions_supported) bekannt geben. Der ZETA Client MUSS dieses Feld vor dem Token Request auswerten und das Verfahren entsprechend wählen.

Unterstützt der ZETA Guard Contract v2 und wird der Parameter resource für eine nicht unterstützte Zielressource angegeben, so MUSS der Authorization Server den Token Request mit dem Fehlerinvalid_target gemäß [RFC8707] ablehnen.

Der PEP HTTP Proxy MUSS während der Übergangsphase sowohl Access Tokens mitver: 1 als auch mit ver: 2 akzeptieren und die jeweils zutreffende Regel zum Audience-Matching anwenden.

5.12.3.6 Anforderungen

A_25338-01 -ZETA Client - Authentifizierung beim Token Exchange

Der ZETA Client MUSS sich gegenüber dem ZETA Guard Authorization Server beim Token Exchange mit einem JWT Bearer Token gemäß [RFC7523] und gemäß [client-assertion-jwt.yaml] authentifizieren. Dabei MUSS die von der gematik für das Clientsystem ausgestellte product_id nach [A_25337] und die Versionskennzeichnung des Clients nach folgendem Schema verwendet werden:

- <product_id> gemäß Registrierung bei der gematik mit Länge maximal 20 Zeichen, Zeichenvorrat [0-9a-zA-Z\],
- <product_version> gemäß Produktidentifikation mit Länge 1-20 Zeichen, Zeichenvorrat [0-9a-zA-Z\].

[<=]

A_25339 -ZETA Client - Exponential Backoff

Der ZETA Client SOLL bei Server-seitigen Fehlermeldungen, die auf eine Überlastung des Zielsystems schließen lassen (z. B. HTTP-status 5xx, 429 - too many requests etc.), erneute Verbindungsversuche nach dem Prinzip des Exponential Backoffs [ExpBack] durchführen. [<=]

Hinweis: Die Parameter für das Verfahren des Exponential Backoffs werden vom Hersteller des ZETA Clients festgelegt.

A_27378-01 -ZETA Client - TLS

Der ZETA Client MUSS mindestens TLS 1.2 oder höher unterstützen.

[<=]

88025A_25340-012 -ZETA Client- Zertifikatsprüfung

Der ZETA Client MUSS alle Zertifikate, die es aktiv verwendet (bspw. TLS-Verbindungsaufbau), auf Integrität und Authentizität prüfen. Falls ein Zertifikat ungültig ist, so MUSS der ZETA Client die von dem Zertifikat und den darin enthaltenen Attributen (bspw. öffentliche Schlüssel) abhängenden Arbeitsabläufe ablehnen.

Der ZETA Client MUSS alle öffentlichen Schlüssel, die es verwenden will, auf eine positiv verlaufene Zertifikatsprüfung zurückführen können.

Die Zertifikatsprüfung MUSS folgende Prüfungen enthalten:

- Der ZETA Client MUSS den ~~Common Name (CN) oder~~ Subject Alternative Name (SAN, Extension dNSName) des Zertifikats ~~m~~(oder falls nicht vorhanden den Common Name (CN)) m mit dem erwarteten Hostnamen vergleichen.
- Der Client MUSS die Gültigkeitsdauer des Zertifikats (Nicht vor und Nicht nach) überprüfen.
- Vertrauenswürdigkeit der Zertifikatskette: Sicherstellen, dass die Kette zu einer vertrauenswürdigen Root-CA führt.
- Revocation-Prüfung für das End-Entity-Zertifikat (Server-Zertifikat): Der Client MUSS den Widerrufsstatus prüfen. Hierbei SOLL primär eine vom Server bereitgestellte OCSP-Response (OCSP Stapling) ausgewertet werden. Fehlt diese, MUSS die Prüfung über gecachte CRLs oder OCSP-Responses erfolgen.
- Revocation-Prüfung für Sub-CA-Zertifikate: Der Client MUSS den Widerrufsstatus der Sub-CAs in der Kette prüfen. Um Lastspitzen und Latenzen zu vermeiden, SOLLEN hierfür vom Betriebssystem bereitgestellte Mechanismen (z. B. CRLite, lokale Trust-Stores) oder serverseitiges Multi-Stapling (TLS 1.3) verwendet werden. Falls Live-Abfragen (OCSP/CRL) für Sub-CAs notwendig sind, MÜSSEN deren Antworten zwingend entsprechend ihrer Gültigkeitsdauer (Feld nextUpdate) lokal gecacht werden.

Der ZETA Client MUSS die Root-Zertifikate, die von den Mitgliedern des [CAB-Forum] ausgestellt werden, als Vertrauensanker standardmäßig unterstützen. Dies impliziert:

- Eine Standard-Liste von [CAB-Forum] Root-Zertifikaten ist im ZETA Client enthalten und wird regelmäßig aktualisiert oder der ZETA Client verwendet den Truststore des Betriebssystems.

[<=]

88757A_27379-012 -ZETA Client - TLS OCSP Stapling Unterstützung

Der ZETA Client MUSS beim TLS Verbindungsaufbau OCSP Stapling erkennen und nutzen, wenn es vom Server bereitgestellt wird.

Der ZETA Client MUSS die Gültigkeit der OCSP-Antwort (Signatur, Signierer) prüfen.

Der ZETA Client MUSS die Gültigkeitsdauer der OCSP-Antwort prüfen.

Der ZETA Client MUSS überprüfen, ob die OCSP-Antwort zum angefragten Zertifikat passt. OCSP-Antworten MÜSSEN im Cache für die Zeit bis NextUpdate der OCSP Response gespeichert werden, um unnötige Abfragen zu vermeiden. Die Dauer der Speicherung im Cache MUSS konfigurierbar sein und mindestens 1 Stunde betragen.

~~WennNextUpdate nicht in der nn OCSP Response enthalten ist, dann MUSS die OCSP-Response bisThisUpdate + 24 Stunden im Cache gespeichert werden.~~

~~Wenn OCSP StapStapling nicht angeboten wird, MUSS der ZETA Client entweder die CRL laden oder den OCSP Responder des Zertifikats direkt abfragen.~~

Wenn weder eine OCSP Response noch eine CRL verfügbar sind, MUSS der ZETA Client den Verbindungsaufbau abbrechen.

[<=]

88403A_26681-012 -ZETA Client - Umsetzen eines ZETA/ASL-Kanals

Der ZETA Client MUSS einen ZETA/ASL-Kanal (Client-Seite) umsetzen können.

Der ZETA/ASL Kanal muss aufgebaut werden, wenn in [opr-well-known.yaml] der claimzeta_asl_use den Wert required oder required_passthrough hat. Der innere Request MUSS ein Access Token und ein DPoP Token enthalten. Das Access Token claim MUSS die URI des Resource Server Endpunktes enthalten [RFC9449]. Das Access Token und das DPoP Token MUSS den logischen Bezeichner (logische Audience) des Resource Server Endpunktes enthalten, den die Policy Engine gemäß 5.4.2.4 vergibt; das zugehörige Token htu trägt ver : 2 (Contract v2). Bei Legacy-Clients (Contract v1) entspricht deraud claim MÜSS dem verbatim aus dem Parameter audience übernommenen Wert und das Token trägt ver : 1 (siehe 5.4.2.5). die URI des Resource Server Endpunktes enthält wird nicht in dem claim übernommen, sondern ausschließlich über den htu claim des DPoP Proofs gebunden.

Bei zeta_asl_use: required_passthrough gilt:

Im äußeren HTTP Request MÜSSEN zusätzlich ein Access Token und ein DPoP Token vorhanden sein. Das aud htu claim im Access DPoP Token und das htu im äußeren HTTP Request MUSS die URI des ZETA/ASL Endpunktes enthalten. Das aud claim im DPoP Access Token im äußeren HTTP Request MÜSSEN die URI des logischen Bezeichner des ZETA/ASL Endpunktes enthalten.

[<=]

Hinweis: Ob ein ZETA/ASL Kanal zu verwenden ist, wird im OAuth Protected Resource Well-known Dokument des TI 2.0 Dienstes festgelegt (claim zeta_asl_use). Die Anforderungen für den ZETA/ASL-Kanal sind in [gemSpec_Krypt#8] zu finden.

A_28426 -ZETA Client, Service Discovery

Der ZETA Client MUSS die Well-known und JWKS JSON-Dokumente regelmäßig einmal alle 24 Stunden neu laden, wenn im HTTP-Response-Header kein Cache-Control-Header vom ZETA Guard eingefügt wurde. Der ZETA Client MUSS die Service Discovery erneut durchführen, wenn beim Kommunikationsaufbau zu den geschützten Ressourcen der HTTP-Statuscodes 404 (Not Found) empfangen wird.[<=]

A_28425-01 -ZETA Client, Service Discovery - if-none-match und etag

Der ZETA Client MUSS bei der Abfrage der Well-known JSON-Dokumente die HTTP-Header if-none-match und etag verwenden.[<=]

A_29691 -ZETA Guard - Versionierung der Token Ausstellung

Der ZETA Guard Authorization Server MUSS im OAuth Authorization Server Well-known JSON Dokument ([as-well-known.yaml]) über das Feld api_versions_supported die unterstützten Vertragsversionen der Token-Ausstellung bekannt geben.

[<=]

A_29692 -ZETA Client-Registrierer - Versionierung der Token Ausstellung

Der ZETA Client MUSS das Feld api_versions_supported auswerten und

5.12.4 bei Unterstützung

Offener Punkt: Details in diesem Kapitel werden im Rahmen der Implementierung zwischen gematik und dem Zero Trust Hersteller festgelegt und in einer Folgeversion veröffentlicht. Die Client-Registrierung für mobile Apps (ZETA Stufe 2) wird dienstübergreifend ermöglichen, dass im Falle von Big Apps (Unterstützung mehrerer TI-Anwendungen in einem Client) der Nutzer nur einmalig aktiv werden muss, um sein Client mittels 2-Faktor zu bestätigen.

- von Contract v2 den Parameter resource [RFC8707] und scope verwenden.

- andernfalls den Parameter audience (Contract v1) verwenden

[<=]

A_29693 -ZETA Guard - Versionsabhängige Policy Engine Decision

Der ZETA Guard MUSS bei einem Token Request mit resource denaud-Claim aus der Policy Engine Decision ableiten und das Access Token mitver : 2 ausstellen. Bei einem Token Request mit audience MUSS der ZETA Guard denaud-Claim verbatim übernehmen und das Access Token mitver : 1 ausstellen.

[<=]

~~81~~A_29694 -ZETA Guard - Fehlermeldung bei fehlerhaftem Parameter im Token Request

Der ZETA Guard MUSS einen Token Request mitresource für eine nicht unterstützte Zielressource mit dem Fehlerinvalid target gemäß [RFC8707] ablehnen.

[<=]

5.12.5 Client-Registrierung

A_28465 -ZETA Client, Registrierung mit mehreren ZETA Guard

Der ZETA Client MUSS sich einmalig am Authorization Server pro ZETA Guard registrieren.

[<=]

Hinweis: Dabei ist es unerheblich, ob der TI 2.0-Dienst einen oder mehrere Ressource Server bereitstellt. Der PDP im ZETA Guard kann für mehrere Resource Server eines TI 2.0-Fachdienste zuständig sein. Diese Realisierung wird durch die Verwendung des API-Katalog in der Service Discovery ermöglicht.

A_28524 -ZETA Client, Konfigurationsdaten pro ZETA Guard Registrierung

Der ZETA Client MUSS die Registrierungs- und Konfigurationsdaten inklusive der client-id pro ZETA Guard Registrierung verwalten.[<=]

Hinweis: Grundsätzlich kann ZETA Guard den Zugriff auf mehrere Ressourcen kontrollieren.

A_25432-01 -ZETA Client - Ablauf Client-Registrierung

Der mobile ZETA Client (oder ein stationärer Client) für Versicherte MUSS, sofern er an Schnittstellen der Telematikinfrastuktur wegen einer ungültigen/fehlenden Client-Registrierung abgewiesen wird, eine Registrierung am Authorization Server durchführen, indem er

- kryptografische Client-Credentials lokal generiert,
- den/die Nutzer:in mittels OpenID Connect authentifiziert,
- die generierten Credentials sowie die Clientintegrität attestiert und
- eine zusätzliche Nutzerbestätigung mittels One-Time-Passwort (gemäß [TR-03107-1]) über einen zweiten Kommunikationsweg (E-Mailbestätigung) startet.

[<=]

Hinweis: Für die Client-Registrierung muss das Vertrauensniveau hoch, nicht erreicht werden.

A_25766 -ZETA Client - Client Credentials in TI Qualität

Der ZETA Client MUSS die Client-Credentials im Form von kryptografischen Schlüsseln gemäß der Festlegungen in [gemSpec_Krypt] (Verfahren, Algorithmen, Schlüssellängen etc.) unterstützen.[<=]

A_25769 -ZETA Client - Client Credentials sicher generieren und schützen

Der ZETA Client auf mobilem Gerät mit Apple- oder Android-basierter Betriebsumgebung MUSS die Generierung der Client-Credentials derart generieren und speichern, dass ein Kopieren und Exportieren der Schlüssel verhindert wird. [<=]

Hinweis: Eine Speicherung der Schlüssel in einem Hardware-Modul ist gegenüber einer Software-Lösung (z. B. Android TEE) zu bevorzugen.

~~Mainline_OPB1/ML-153922A_25770-ZETA Client - Client Credentials Rotation~~
~~Der ZETA Client MUSS seine Client-Credentials regelmäßig rotieren (erneuern und neu registrieren), wobei die Häufigkeit der Rotation durch die gematik nach einer Auswertung der initialen Nutzererfahrung festgelegt wird. [<=]~~

~~HinweisHinweis: Das Client Instance Key Pair soll initial 2 Jahre gültig sein. Der private Client Instance Key PrK.Client.Sig soll im TPM2 Modul verwendet werden (oder Secure Enclave bei macOS und iOS sowie TEE bei Android).~~

A_25767 -ZETA Client - Clientkey in JWT

Das ZETA Client MUSS Private Key JWT [RFC7521] und [RFC7523] sowie DPoP [RFC9449] zur Authentifizierung unterstützen. [<=]

A_25434 -ZETA Client - Client-Registrierung mit bestätigten Umgebungseigenschaften Android

Der ZETA Client für eine Google-Android basierte Betriebsumgebung MUSS seine Client-Credentials, App-Integrität und -Authentizität sowie OS-/FW und HW-Eigenschaften über Key and ID Attestation gegenüber PDP Client-Registrierung bestätigen.. [<=]

A_25768 -ZETA Client - Client-Registrierung mit bestätigten Umgebungseigenschaften Apple

Der ZETA Client für eine Apple-basierte Betriebsumgebung (iOS, macOS) MUSS die Client-Credentials, App Integrität und Authentizität über DCAppAttest gegenüber dem PDP Authorization Server bestätigen. Eigenschaften der Laufzeitumgebung MÜSSEN durch das Clientsystem über einen geprüften Prozess bestätigt werden. [<=]

A_25758 -ZETA Client - Erfassung Kontaktinformation für Offband-Verifikation

Der mobile ZETA Client MUSS vom Nutzer eine strukturell valide Kontaktinformation (E-Mailadresse) abfragen und für eine Offband-Verifikation (Trust on First Use) an den Endpunkt des Authorization Servers übertragen. [<=]

A_25732 -ZETA Client - Unterstützung des Nutzers bei der Client-Registrierung

Der mobile ZETA Client MUSS den Nutzer bei der Client-Registrierung und -Verwaltung geeignet unterstützen (z. B. mittels Guide, Tutorial o. ä.). [<=]

A_25733 -ZETA Client - Clientverwaltung und manuelle De-Registrierung

Der ZETA Client MUSS dem Nutzer eine Übersicht aller beim Client-Registrierungsdienst registrierten Clients darstellen und die Möglichkeit zur De-Registrierung einzelner Clients anbieten. [<=]

A_25734 -ZETA Client - Zugriffsprotokoll Client-Registrierung

Der ZETA Client MUSS dem Nutzer einen Einblick in das Zugriffsprotokoll der Schnittstellen des Client-Registrierungsdienstes für genutzte Clients dieses Nutzers geben. [<=]

A_25735-01 -mobiler Client- Push-Benachrichtigung

Das Clientsystem MUSS dem Nutzer die Möglichkeit geben, Push-Benachrichtigungen für Aktivitäten über registrierte Clients und Neuregistrierungen für diesen Nutzer zu aktivieren, zu ändern und zu deaktivieren. [<=]

5.12.6 Nutzerauthentifizierung

A_25761 -ZETA Client - Nutzerauthentifizierung mittels etablierter Standards

Der ZETA Client MUSS die Mechanismen OAuth Authorization Code Flow mit OpenID Connect und OpenID Federation (Auswahl des zuständigen sektoralen IDP) oder OAuth2 Token Exchange mit `private_key_jwt` Client Authentifizierung unterstützen. [<=]

Hinweis: Perspektivisch sollen ZETA Clients auch OpenID for Verifiable Credentials (OIDC4VC) unterstützen. OAuth2 Token Exchange wird für stationäre Clients mit SM(C)-B Authentisierung verwendet. OAuth Authorization Code Flow, OpenID Connect und OpenID Federation wird grundsätzlich von mobilen Clients verwendet, kann aber auch von stationären Clients verwendet werden.

A_25762-01 -ZETA Client - Nutzerauthentifizierung - Unterstützung etablierter Identitäten und Dienste

Der ZETA Client MUSS zur Authentifizierung des Nutzers mindestens eines der folgenden Verfahren unterstützen:

- Authentifizierung des Nutzers gegenüber einem Sektoralen IDP der IDP Föderation gemäß [gemSpec_IDP_Sek] (GesundheitsID)
- Authentifizierung des Nutzers mittels SM(C)-B signiertem Subject Token.

[<=]

5.12.7 Session Management

53946A_25781-01 -ZETA Clients - OAuth2 Autorisierung

Der ZETA Client MUSS die Rolle eines OAuth2 Clients [RFC6749] übernehmen und eine Autorisierung vom Authorization Server einholen. Dabei MUSS [beim Authorization-Code-Flow der PKCE Flow \[RFC7636\]](#) verwendet werden.

[<=]

A_25782 -ZETA Client - OAuth2 Session Management

Der ZETA Client MUSS

- die vom Autorisation Server ausgestellten Access- und Refresh Token gemäß [RFC6749#1.5] sowie die DPoP Schlüssel gemäß [RFC9449] bis zur nächsten Aufforderung zur Autorisierung oder Authentifizierung als User-Session sicher aufbewahren,
- nach Bedarf abgelaufene Access Token über Refresh Token erneuern und
- eine Refresh Token-Rotation gemäß [RFC6749#10.4] unterstützen.

[<=]

MainlineHinweis: Die Session des ZETA Clients zum ZETA Guard Authorization Server wird über den `claim sid` im Access Token abgebildet.

A_25783-01 -ZETA Client - Anweisungen aus HTTP Response Status Codes und Header folgen

Der ZETA Client MUSS die HTTP Response Status Codes und HTTP Header entsprechend der Vorgaben der Resource Server und Zero Trust-APIs auswerten und den Anweisungen daraus folgen und insbesondere

- eine Step-Up- oder erneute Authentifizierung des Nutzers,
- eine Re-Autorisierung und erneute Attestation der Client-Instanz und
- eine Anzeige der Warnungen aufgrund der Policy-Entscheidungen

umsetzen. [<=]

5.12.8 Liste der HTTP-Statuscodes

DA_29849 -ZETA Client, Reaktion auf terminierte Session

Der ZETA Client MUSS bei einer Ablehnung aufgrund terminierter Session eine erneute Authentifizierung des Nutzers einleiten.

Der ZETA Client DARF keinen weiteren Refresh-Request mit demselben Session-Kontext durchführen.

[<=]

A_30002 -ZETA Client - Serverseitiger Token-Widerruf bei Logout

Der ZETA Client MUSS bei einer nutzerinitiierten Abmeldung bzw. Beendigung der Session sein Refresh Token über POST /revoke [RFC7009] am Authorization Server widerrufen, damit die clientseitige Abmeldung zu einer echten serverseitigen Grant-Terminierung wird, und MUSS lokal gespeicherte Access-/Refresh Token sowie DPoP-Schlüssel anschließend löschen.[<=]

5.12.9 Fehlerbehandlung

5.12.9.1 Liste der HTTP-Statuscodes

Der folgende Abschnitt enthält die HTTP-Statuscodes, die ZETA Clients von Zero Trust-Komponenten erhalten können, basierend auf den spezifischen Schritten wie Authentifizierung, Client-Registrierung und HTTP Proxy.

Die Fehlercodes sollen dem Client die Möglichkeit geben, angemessen auf die jeweilige Fehlersituation zu reagieren.

Dabei wird unterschieden zwischen den folgenden Ergebnisklassen, die über verschiedene Statuscodes angezeigt werden:

1. **Erfolgreiche Aktion:** dies sind die Statuscodes
 - a. 2xx
 - b. 3xx
2. **Abschließend nicht erfolgreiche Aktion:** dies sind insb. die Statuscodes:
 - a. 400 Bad Request: sollte ein Request mit 400 abgelehnt werden, kann der Client nicht entscheiden was anders aufzurufen wäre und würde den Request daher nur wiederholen können. Daher wird hier abgebrochen.
 - b. Alle weiteren 4xx Statuscodes die nicht in den anderen Klassen gelistet sind (insb. nicht 401 und 403, siehe nächsten Punkt).
3. **Step-Up Authentifizierung:** Eine Step-Up-Authentifizierung wird grundsätzlich nur einmal automatisch durchgeführt, bevor abgebrochen wird. Dies dient auf der einen Seite der erneuten Prüfung gegen die OPA-Regeln z.B. im Falle von Netzwerkwechseln, und auf der anderen Seite dem Vermeiden einer Überlastung durch nur einmalige Wiederholung. Hier wird unterschieden zwischen Requests gegen den PDP, bei denen die OPA Regeln bereits abgefragt werden – diese erlauben keinen Retry bei Forbidden, und anderen Requests, bei denen nach dem 403 Forbidden die OPA Regeln neu abgefragt werden sollen:
 - a. Bei einem 401 Unauthorized Statuscode bei einem Request gegen den PDP im Rahmen der Client-Registrierung, der Authentifizierung oder des Token Refreshes kann der Client einen Request nach einer erneuten Authentifizierung einmalig wiederholen. Die angenommenen Fehlersituationen hier sind z.B. während des Requests ablaufende Zertifikate/Token (time-of-check to time-of-use Situation), oder die Invalidierung eines Access Token durch Impossible-Travel Detektion oder ähnliches. Die Authentifizierung soll maximal einmal wiederholt werden, bevor ein

401/403 Status nach einem wiederholten Request als abschließend nicht erfolgreich gewertet wird.

- b. Bei ~~einem 401 Unauthorized sowie bei einem 403 Forbidden beim Zm~~ Zugriff auf eine Ressource über den PEP HTTP Proxy sowiebzw. andere PDP-APIs (z.B. Client-Management in Stufe 2) kann/wird unterschieden:
 - (i) Step-up (401 mit error=insufficient_user_authentication, [RFC9470]): der Client statt einer vollständig neuen Authentifizierung führt einmalig eine Authentisierung auf dem geforderten Niveau (acr_values aus dem WWW-Authenticate-Header) durch und wiederholt den Request. Schlägt auch dieser fehl, wird abgebrochen.
 - (ii) Ungültiges/abgelaufenes Token (401 mit error=invalid_token) oder erneute Policy-Prüfung (403): Der Client DARF einmalig einen Token-Refresh unter Nutzung eines vorhandenen Refresh-Token versuchen. Falls das (dies löst ein weitere erneute OPA-Prüfung aus, z. B. nach Netzwerkwechsel/Impossible-Travel). Wird erneut 401/403 Statuscode erhalten-wird, wird dann einmalig eine volle Authentifizierung durchgeführt, bevor der ursprüngliche Request mit neuem Access Token wiederholt werden kann.

4. Temporäre Fehlersituation:

- a. Bei 5xx Statuscode kann der Client wie in ZT_HTTP_Statuscodes den Request ggf. nach Wartezeit wiederholen
- b. 429 Too Many Requests – auch hier kann der Client – nach Wartezeit – den Request wiederholen.

A_27007 -ZETA Client - HTTP Statuscodes

Der ZETA Client MUSS die HTTP Statuscodes gemäß Tabelle "ZT_HTTP_Statuscodes" unterstützen.[<=]

Tabelle 10: ZT_HTTP_Statuscodes

Endpunkte	HTTP Statuscodes
Authentifizierung mit <u>Client-Assertion-JWTSM(C)-B signiertem Subject Token</u>	<p>200 OK: Authentifizierung erfolgreich. Der Client kann mit der gewünschten Operation fortfahren, z.B. Zugriff auf geschützte Ressourcen.</p> <p>400 Bad Request: Fehlerhafte oder ungültige JWT-Assertion (z.B. falsches Format oder fehlende Claims).</p> <p>401 Unauthorized: Authentifizierung fehlgeschlagen, z.B. aufgrund einer ungültigen Signatur oder eines abgelaufenen JWT. Der Client <u>(error Code initiiert eine einmalige Step-Up Authentifizierung, siehe oben zu Retry der valid client oder insufficient_user Authentifizierungcation).</u></p> <p>403 Forbidden: Der Client hat keine Berechtigung, auf die angeforderte Ressource zuzugreifen, obwohl er authentifiziert ist. Der Client darf keine weiteren Versuche unternehmen und soll den Nutzer entsprechend informieren <u>(error Code access denied, session terminated oder refresh token revoked).</u></p>

<p>Authentifizierung mit OIDC und Authorization Code Flow</p>	<p>200 OK: Erfolgreiche Authentifizierung und Autorisierung. Der Client kann mit der gewünschten Operation fortfahren, z.B. Zugriff auf geschützte Ressourcen.</p> <p>302 Found: Der Client wird zum Autorisierungs-Endpunkt des Identitätsproviders umgeleitet. Der Client sollte der Weiterleitungs-URL folgen, um den nächsten Schritt im Autorisierungscode-Austausch abzuschließen.</p> <p>400 Bad Request: Fehlerhafte Anfrage, z.B. fehlende oder ungültige Parameter (z.B. falscher "redirect_uri", "client_id").</p> <p>401 Unauthorized: Authentifizierung fehlgeschlagen, z.B. ungültiger Code oder Token. Der Client (error Code initiiert eine einmalige Step-Up Authentifizierung, siehe oben zu Retry der valid client oder insufficient user Authentifizierungscation).</p> <p>403 Forbidden: Autorisierung fehlgeschlagen, z.B. fehlende Zugriffsrechte. Der Client hat möglicherweise keine Berechtigung, die angeforderte Ressource zu nutzen. Der Client sollte den Nutzer darüber informieren und keine weiteren Anfragen stellen (<u>error Codeaccess denied, session terminated oder refresh token revoked</u>).</p>
<p>Token-Refresh</p>	<p>200 OK: Authentifizierung erfolgreich. Der Client kann mit der gewünschten Operation fortfahren, z.B. Zugriff auf geschützte Ressourcen.</p> <p>400 Bad Request: Fehlerhafte oder ungültige JWT-Assertion (z.B. falsches Format oder fehlende Claims).</p> <p>401 Unauthorized: Authentifizierung fehlgeschlagen, z.B. aufgrund einer ungültigen Signatur oder eines abgelaufenen JWT. Der Client initiiert eine einmalige Step-Up Authentifizierung, siehe oben zu Retry der Authentifizierung (<u>error Code invalid client oder insufficient user authentication</u>).</p> <p>403 Forbidden: Der Client hat keine Berechtigung, auf die angeforderte referenzierte Session wurde serverseitig terminiert oder das zugehörige Refresh Token wurde entzogen. Der Authorization Server liefert im Response-Body einen standardisierte Ressource-zuzugreifen, obwohl ern Fehlercode (session terminated oder refresh token revoked). Der Client MUSS eine erneute authentifiziert ist. Der Client dung durchführen und Darf keinen weiteren Versuche unternehmen und soll Refresh-Request mit demselben Session-Kontext senden Nutzer entsprechend info(error Codeaccess denied, session termierennated oder refresh token revoked).</p>
<p>Client-Registrierung</p>	<p>201 Created: Client erfolgreich registriert. Der Client sollte die Registrierungsdaten sicher speichern und mit der weiteren Interaktion fortfahren.</p> <p>400 Bad Request: Fehlerhafte Anfrage, z.B. ungültige oder unvollständige Clientdaten.</p> <p>401 Unauthorized: Fehlende oder ungültige Authentifizierung bei der Registrierung. Der Client muss sicherstellen, dass er korrekt authentifiziert ist. Der Client initiiert eine einmalige Step-Up Authentifizierung, siehe oben zu Retry der Authentifizierung.</p> <p>403 Forbidden: Zugriff verweigert, z.B. wenn der Client nicht berechtigt ist, sich zu registrieren. Der Client sollte prüfen, ob er die Berechtigung zur Registrierung hat. Wenn nicht, sollte er keine weiteren Versuche unternehmen und den Nutzer informieren.</p>

	<p>409 Conflict: Konflikt bei der Registrierung, z.B. ein Client mit der gleichen ID existiert bereits. Der Client darf keine weiteren Registrierungsversuche senden und soll den Nutzer über die das Serverproblem informieren.</p>
<p>HTTP Proxy</p>	<p>200 OK: Anfrage erfolgreich durch den Proxy weitergeleitet. Der Client kann die angeforderte Ressource wie gewohnt verwenden.</p> <p>301 Moved Permanently: Permanente Weiterleitung der Anfrage durch den Proxy. Der Client sollte die neue URL speichern und zukünftige Anfragen an diese Adresse senden.</p> <p>302 Found: Temporäre Weiterleitung durch den Proxy. Der Client sollte der Weiterleitung folgen, um die angeforderte Ressource zu erhalten, aber die ursprüngliche URL für zukünftige Anfragen beibehalten.</p> <p>400 Bad Request: Ungültige Anfrage an den Proxy. Der Client sollte die Anfrage überprüfen und sicherstellen, dass sie korrekt formatiert und vollständig ist, bevor er sie erneut sendet.</p> <p>401 Unauthorized: <u>Das Access/DPoP-Token ist Fehlende-, ungültig oder ungültig abgelaufen (error=invalid_token) oder das nachgewiesene Authentifizierung. Der Client initiiert eine „sniveau reicht nicht aus (error=insufficient_user_authentication, Step-Up“ A gemäß [RFC9470], signalisiert im WWW-Authentifizierung, d.h. cate-Header). Im ersten Fall führt ein der Client einmalig eine erneute Authentifizierung/Token-Refresh durch und probiert den Request erneut.; im zweiten Falls auch dieser Request fehlschlägt wird abgebrochen. Siehe dazu auch oben zu Retry initiiert er das Step-up-Verfahren gemäß A_29859 und wiederholt der Authentifizierung Request einmalig.</u></p> <p>403 Forbidden: <u>Zugriff auf die angeforderte Ressource Endgültige Ablehnung durch den Proxy verweigert. Der Client initiiert eine „Step-Up“— der Zugriff ist mit dem vorliegenden, korrekt Authentifizierung, d.h. führt einmalig eine neue Authentifizierung dursierten Token nicht gestattet (z. B. aud/acr/htu-Mismatch und probiert nach A_29676, Policy den Request erneut. Falls auchy, session_terminated, refresh_token_revoked). dieser Reque ist fehlschlägt wird abgebrochenkein Step-up-Fall. Der Client darf keine weiteren Anfragen an diese Ressource senden malig einen Token-Refresh versuchen (erneute OPA-Prüfung, z. B. nach Netz-/IP-Wechsel); bleibt die Antwort 403, bricht er ab und sollinformiert den Nutzer über die fehlende Berechtigung informier. Bei session_terminated/refresh_token_revoked MUSS er eine vollständige Neu-Authentifizierung durchführen.</u></p> <p>404 Not Found: Die angeforderte Ressource wurde nicht gefunden.</p> <p>405 Method Not Allowed: Die verwendete HTTP Methode wird nicht unterstützt für den Endpunkt.</p> <p>502 Bad Gateway: Der Proxy hat eine ungültige Antwort vom Upstream-Server erhalten. Der Client sollte die Anfrage später erneut senden, da der Fehler auf einem Problem des Upstream-Servers beruhen könnte. Gegebenenfalls kann der Nutzer informiert werden.</p> <p>504 Gateway Timeout: Der Proxy hat auf eine Antwort vom Upstream-Server gewartet, diese aber nicht innerhalb des Timeouts erhalten. Der Client sollte die Anfrage nach einer angemessenen Wartezeit erneut versuchen und den Nutzer darüber informieren, dass der Server nicht rechtzeitig geantwortet hat.</p>

Alle Endpunkte	<p>429 Too Many Requests: Zu viele Anfragen innerhalb eines bestimmten Zeitraums (Rate Limiting). Der Client sollte die Anzahl der Anfragen reduzieren und eine geeignete Wartezeit (Retry-After Header beachten) einhalten, bevor er die Anfrage erneut sendet.</p> <p>500 Internal Server Error: Allgemeiner Serverfehler. Der Client sollte den Vorgang möglicherweise zu einem späteren Zeitpunkt wiederholen oder den Nutzer auf ein Problem auf dem Server hinweisen.</p>
-----------------------	---

5.12.9.2 Behandlung von PEP-Fehlern mittels HTTP-Header

5.12.10 Für die korrekte Fehlerbehandlung im ZETA-Attestation-Service

Der ZETA-Attestation-Service stellt einen gRPC-Dienst zur Verfügung. Client (SDK) ist es erforderlich, Fehler, die direkt durch den Policy Enforcement Point (PEP) verursacht werden, von fachlichen Fehlern des dahinterliegenden Resource Servers (Fachdienstes) zu unterscheiden.

Step-up-Bedarf (unzureichendes Sicherheitsniveau) signalisiert der PEP mit 401 (Unauthorized) gemäß [RFC9470]; 403 (Forbidden) kennzeichnet eine endgültige Ablehnung. Beide Codes können sowohl vom PEP (Header zeta-error-origin: pep) als auch vom Fachdienst stammen und werden über diesen Header unterschieden.

Während fachliche Fehler des Resource Servers unverändert an den Fachclient durchgereicht werden müssen, erfordern Fehler des PEP eine dedizierte Behandlung durch das ZETA-SDK (z. B. das Initiieren eines Step-Up-Verfahrens oder ein kontrollierter Retry).

Um diese Unterscheidung ohne komplexe Payload-Analysen (insb. im Kontext von JSON- vs. CBOR-Sicherheitsprüfungen in ASL) zu ermöglichen, signalisiert das PEP eigene Fehler über einen spezifischen HTTP-Response-Header.

A_29852 -ZETA Client, Auswertung des PEP-Fehler-Headers

Empfängt der ZETA Client eine HTTP-Fehlerantwort (Statuscodes 401 oder 403), MUSS es prüfen, ob der HTTP-Header zeta-error-origin: pep in der Antwort enthalten ist. Ist der HTTP-Header zeta-error-origin: pep in der Fehlerantwort vorhanden, MUSS der ZETA Client die für den spezifischen Fehlerfall definierte interne Fehlerbehandlung einleiten. [<=]

A_29853 -ZETA Client, Weiterleitung von Fachdienst-Fehlern

Ist der HTTP-Header zeta-error-origin in einer Fehlerantwort (auch bei den Statuscodes 401 oder 403) nicht enthalten, MUSS der ZETA Client den Fehler unverändert und transparent an den aufrufenden Fach-Client durchreichen. [<=]

A_29859 -ZETA Client, Behandlung von Step-Up-Fehlermeldungen

Empfängt der ZETA Client eine HTTP-Fehlerantwort mit HTTP-Statuscode 401 und zeta-error-origin: pep, bei der das Step-up über den WWW-Authenticate-Header error="insufficient user authentication" gemäß [RFC9470] oder error="insufficient scope" signalisiert wird, MUSS der ZETA Client den Fall als Step-up erkennen und dabei das geforderte Niveau acr values und oder scopes sowie ggf. max_age aus dem WWW-Authenticate-Header übernehmen und folgende Schritte durchführen:

- Fehlererkennung: Den Fehler als dedizierten Step-Up-Fehler identifizieren (und nicht als regulären Verbindungs- oder Token-Fehler).

- Initiierung des Upgrades: Die Prozedur zum Erlangen eines neuen Access Tokens mit erforderlichem Sicherheitsniveau (z. B. durch erneute Authentifizierung mit den geforderten Parametern) einleiten.
- Fachclient-Interaktion: Kann das SDK den Step-Up-Prozess nicht vollautomatisch im Hintergrund durchführen (z. B. weil eine Benutzerinteraktion für die Multi-Faktor-Authentifizierung nötig ist), MUSS der ZETA Client dem aufrufenden Fach-Client eine spezifische Exception bzw. ein definiertes Fehlerobjekt zurückgeben, der-eas den Step-Up-Bedarf signalisiert.
- Retry-Option: Nach erfolgreicher Durchführung des stationep-Ups MUSS der ZETA Client die Möglichkeit bieten, den ursprünglichen Request mit dem neuen, höherwertigen Access Token zu wiederholen.

[<=]

Hinweis: Nach RFC 6750 §3.1 ist insufficient_scope klassisch mit 403 assoziiert. Die bewusste Behandlung als 401-Step-up ist hier korrekt, weil der fehlende Scope in ZETA über eine (Step-up-)Authentisierung erlangbar ist.

5.12.11 ZETA Attestation Service

Der ZETA Attestation Service stellt einen Dienst zur Verfügung, der es stationären Clients (Primärsysteme auf Basis von Windows oder Linux) ermöglicht, TPM-signierte Attestierungsinformationen für den Client abzurufen. Diese Informationen basieren auf Integritätsmessungen, die in ausgewählten Platform Configuration Registers (PCRs) des Trusted Platform Module (TPM) gespeichert sind. Der ZETA Guard Authorization Server verwendet diese Attestierungsdaten, um die Integrität und Authentizität der Softwareumgebung des Clients zu verifizieren, bevor Zugriff auf geschützte Ressourcen gewährt wird.

Der ZETA Attestation Service wird vom Hersteller des stationären Clients bereitgestellt und es muss eine manipulationsgeschützte Vertrauensbeziehung zwischen stationären Client und ZETA Attestation Service bestehen, um zu gewährleisten, dass die Attestation über die vorgesehenen Software-Komponenten erfolgt.

Der ZETA Attestation Service muss gemäß [API-ZETA-Attestation-Service] implementiert werden.

Hinweis: Während der Installation oder bei Updates des stationären Clients muss auch ein Update des ZETA Attestation Service erfolgen um eine neue Baseline für die Integrität des stationären Clients zu setzen. Die Baseline besteht aus einem Hash über alle unveränderlichen Komponenten des stationären Clients, inkl. ZETA Attestation Service.

Hinweis: Der ZETA Attestation Service muss bei jedem Start des Clients die Messung über die Integrität des Clients durchführen und in das PCR schreiben.

Hinweis: Der ZETA Attestation Service ist nicht für mobile Clients vorgesehen. Mobile Clients verwenden eine andere Attestierungsmethode, die auf den jeweiligen Plattformen basiert (z.B. Android und iOS).

Hinweis: Wenn die Messung des Clients von der Baseline abweicht, dann soll der Zeta Attestation Service automatisch den Support des Herstellers informieren.

5.13 Client- Management

5.14 Client Registrierungsdaten

Dieses Kapitel beschreibt die Datenstrukturen für die Kommunikation zwischen ZETA Client und ZETA Guard bei der Client-Registrierung.

Die Client-Registrierung erfolgt über ein vom Client erzeugtes und signiertes Client Assertion JWT, das Informationen über den Client sowie über die aktuelle Gerätesituation (Posture) enthält.

Die übermittelten Informationen werden vom Authorization Server geprüft und anschließend in der Client-Registry gespeichert. Der Authorization Server stellt diese Informationen dem Policy Decision Point für Autorisierungsentscheidungen zur Verfügung.

Die Client-Registrierung besteht aus drei logischen Datenstrukturen:

- Client Assertion JWT – kryptographisch signierte Identifikation der Client-Instanz gemäß [client-assertion-jwt.yaml]
- Client Statement – statische Eigenschaften des Clientsystems gemäß [client-statement.yaml]
- Posture Informationen – Eigenschaften der Laufzeitumgebung des Clients gemäß [posture.yaml] sowie den referenzierten-plattformabhängigen Informationen gemäß [posture-apple.yaml], [posture-android.yaml], [posture-tpm.yaml] und [posture-software.yaml].

5.14.1 Client Assertion JWT

Die konkrete Installation eines Clients wird durch ein **Client Assertion JWT** identifiziert. Das JWT wird vom Client signiert und beim Authorization Server übermittelt.

Tabelle 11: Tab-Client-Assertion-JWT

Claim	Beschreibung
iss	Aussteller des JWT. Muss die vom AuthS vergebene die OAuth 2.0 client_id des Clients sein.
sub	Subject. Muss die OAuth 2.0 client_id des Clients sein.
aud	Zielgruppe. Muss die Token-Endpoint-URL des Authorization Servers enthalten.
exp	Ablaufzeitpunkt des JWT (Sekunden seit Epoch).
jti	JWT-ID – eindeutiger Bezeichner des Tokens.
client_statement	Optionales Client Statement. Nur im Registrierungs-Ablauf vorhanden. Schema [client-statement.yaml]

5.14.2 Client Statement

Das Client Statement (Schema: [client-statement.yaml]) enthält Informationen über die Client-Instanz und wird als Claim `client_statement` in den Payload des Client Assertion JWT eingebettet.

Tabelle 12: Tab-Client-Statement

Claim	Beschreibung
sub	Name / Bezeichnung des Clients.
platform	Plattform der Client-Instanz. Erlaubte Werte: android, apple, windows, linux, other.
posture_type	Art der übermittelten Posture. Erlaubte Werte: android, apple, software, tpm.
posture	Posture der Client-Instanz. Struktur abhängig vom posture_type (s. Tabellen 3-610-13). Schema: posture.yaml.
attestation_timestamp	Zeitstempel der Attestierung.

5.14.3 Posture Informationen

Das Posture-Objekt dient dem Nachweis der Integrität der Laufzeitumgebung des Clients. Diese Informationen können sich im Laufe der Zeit ändern, beispielsweise durch Updates des Betriebssystems oder der Clientsoftware.

5.14.3.1 TPM Posture

Die TPM-Posture gilt für stationäre Clients (Windows oder Linux) mit TPM-basierter Attestierung.

Tabelle 13: Tab-Posture-TPM

Claim	Beschreibung
product_id	gematik-Produktbezeichner.
product_version	Produktversion.
os	Name des Betriebssystems.
os_version	Version des Betriebssystems.
arch	Hardware-Architektur.
tpm_attestation_key	Öffentlicher Schlüssel des im TPM residenten Attestierungsschlüssels (PEM oder base64-DER).
tpm_quote	TPM Quote der Client-Instanz.

tpm_event_log	TPM Event Log der Client-Instanz.
tpm_ek_certificate_chain	Endorsement-Key-Zertifikatskette des TPM-Herstellers (PEM oder base64-DER).
platform_product_id	Plattform-spezifischer Produktbezeichner (Schema: product-id-windows.yaml oder product-id-linux.yaml).

5.14.3.2 Software Posture

Die Software-Posture gilt für generische Software-Clients ohne TPM-Attestierung (Windows oder Linux).

Tabelle 14: Tab-Posture-Software

Claim	Beschreibung
product_id	gematik-Produktbezeichner.
product_version	Produktversion.
os	Name des Betriebssystems.
os_version	Version des Betriebssystems.
arch	Hardware-Architektur.
public_key	Öffentlicher selbst-signierter Signierungsschlüssel (PEM oder base64-DER).
attestation_challenge	Attestierungs-Challenge der Client-Instanz; dient zur Überprüfung des öffentlichen Client-Instanz-Schlüssels und des Nonce vom AS.
platform_product_id	Plattform-spezifischer Produktbezeichner (Schema: product-id-windows.yaml oder product-id-linux.yaml).

5.14.3.3 Apple Posture

Die Apple-Posture enthält die dekodierten Inhalte des Attestation- bzw. Assertion-Objekts einer iOS-App (Apple App Attest).

Tabelle 15: Tab-Posture-Apple

Claim	Beschreibung
product_id	gematik-Produktbezeichner.
product_version	Produktversion.
system_version	Betriebssystemversion.
system_name	Name des Betriebssystems auf dem Gerät.
device_model	Gerätemodell.

key_id	Schlüsselbezeichner des kryptographischen Schlüssels auf dem Gerät (base64-encodiert).
platform_product_id	Plattform-spezifischer Produktbezeichner für Apple-Apps (Schema: product-id-apple.yaml).
fmt	Format des Attestation-Statements. MUSS den Wert apple-appattest haben. Optional, bei initialer Attestation verwendet.
attStmt	Attestation-Statement: x5c (array of base64-Strings, X.509-Zertifikatskette) und receipt (base64, Apple-Receipt für Device Risk Assessment). Beide Unterfelder sind Pflicht, wenn attStmt vorhanden. Optional, bei initialer Attestation verwendet
authData	Strukturierte Authenticator-Daten der Initial-Attestation Optional, bei initialer Attestation verwendet.
signature	Kryptographische Signatur des privaten Geräteschlüssels. Optional, bei nachfolgender Attestation.
assertionAuthenticatorData	Vereinfachte Authenticator-Daten für Folge-Assertions: rpIdHash (byte, SHA-256 der App-ID) und counter (integer, pro Assertion inkrementiert). Beide Unterfelder sind Pflicht, wenn das Objekt vorhanden ist. Optional, bei nachfolgender Attestation.
client_data_json	JSON-String mit den zu signierenden Request-Daten einschließlich eines server-seitig bereitgestellten Challenge. Optional, bei nachfolgender Attestation.

5.14.3.4 Android Posture

Die Android-Posture enthält plattformspezifische Geräte- und Sicherheitseigenschaften gemäß den Android-APIs (vgl. android.os.Build).

Tabelle 16: Tab-Posture-Android

Claim	Beschreibung
product_id	gematik-Produktbezeichner.
product_version	Produktversion.
build	Android-Build-Informationen; (Build.VERSION.SDK_INT, Build.VERSION.SECURITY_PATCH, Build.MANUFACTURER, Build.PRODUCT, Build.MODEL, Build.BOARD).
key_attestation_certificate_chain	Zertifikatskette aus der Android Key Attestation.
platform_product_id	Plattform-spezifischer Produktbezeichner für Android-Apps

	(Schema: product-id-android.yaml).
ro	Systemeigenschaften (ro.crypto.state, ro.product.first_api_level).
packageManager	Informationen des Package Managers (packageManager.feature_verified_boot, packageManager.mainline_patch_level).
keyguardManager	Keyguard-Zustand (isDeviceSecure: boolean).
biometricManager	Biometrische Fähigkeiten (deviceCredential: boolean, biometricStrong: boolean).
devicePolicyManager	Richtlinien des Device Policy Managers (passwordComplexity: integer; erlaubte Werte: 0, 1, 2, 3).
play_integrity_token	Von der Google Play Integrity API ausgestelltes und base64- enkodiertes Token. Wird vom AuthS validiert, um die meets_*_integrity-Claims zu generieren.

5.15 ZETA Guard

5.15.1 Die Software-~~d~~Vertrauensmodell, Faktoren und Geltungsbereich

Die eigentliche **Client-Registrierung** (dynamische Registrierung nach [RFC7591], Client Assertion JWT, Client Statement, Posture/Attestierung sowie der TOFU-Ablauf mit E-Mail-OTP) ist bereits in den vorangehenden Kapiteln normativ definiert (insb. 5.3.2, 5.5.1-5.5.3 sowie A_29658, A_25432-01). Dieses Kapitel definiert **kein** eigenes Registrierungsverfahren, sondern setzt darauf das **Client Management** auf: die faktor-gebundene Verwaltung *bestehender* Registrierungen über ihren Lebenszyklus hinweg (Folgerregistrierung, E-Mail-Bindung, Schlüssel-Rollover, Recovery, Client-Verwaltung und außerordentliche Löschung).

Grundlage dieses Managements ist ein Trust-On-First-Use-Verfahren (TOFU) nach [RFC7591]/[RFC7592] mit dem Faktorsatz:

- **F1 - verifizierte E-Mail:** auf Identitätsebene gebundener Nachweis, einmalig bei Erstnutzung gepinnt.
- **F2 - Instanzschlüssel (PoP):** je Client-Instanz gebundenes asymmetrisches ZETA-Guards wird im AuftraSchlüsselpaar.

Schutzgut sind **bestehende** Client-Registrierungen gegenüber Übernahme durch einen kompromittierten sektoralen IDP. Die Grenze des Verfahrens (First-Use-Capture vor Erstnutzung) ist akzeptiert.

A_29892 -Authorization Server - Bereitstellung der TOFU-Erweiterungsschnittstellen

Der Authorization Server MUSS zusätzlich zur bereits definierten Client-Registration-API (POST /register, POST /register/verify; definiert durch A_29658) die darauf aufbauenden TOFU-Erweiterungen gemäß Annex [zeta-guard-client-management] (RFC 7592-Registrierungsverwaltung/register/{client_id}, Schlüssel-Rollover, E-Mail-Änderung, Recovery, Client-Verwaltung,

clientseitiges Veto) sowie die privilegierte, IDP-unabhängige Operator-Schnittstelle für die außerordentliche Löschung gemäß Annex zeta-guard-admin-oob.yaml bereitstellen. Die Bereitstellung der gematik-entwickelt und alle Komponenten als signierte OCI-Images- und als signiertes Helm-Chart in ein Basis-Registrierung selbst richtet sich unverändert nach A_29658. [<=]

A_29893 -Authorization Server - Geltungsbereich des TOFU-Faktormodells (mobile Clients)

Der Authorization Server MUSS das TOFU-Faktormodell dieses Kapitels ausschließlich auf mobile Client-Registrierungen anwenden, die im TOFU-Verfahren an eine TI-Identität (KVNR oder Telematik ID) gebunden werden. Für stationäre Clients mit Authentisierung per SMC-B Token Exchange (vgl. A_25762-01) sowie für Clients ohne Nutzer-Identität (vgl. A_29733) gelten die faktorgebundenen Anforderungen NICHT; deren Absicherung erfolgt gemäß den dort genannten Verfahren. Der Schlüssel-Rollover (A_29921 ff.) gilt clientübergreifend. [<=]

A_29894 -Authorization Server - Identitätslose Clients: reduziertes Vertrauensmodell

Der Authorization Server MUSS einen ohne TI-Identität registrierten Client (vgl. A_29733) ausschließlich über den Instanzschlüssel (F2, Proof-of-Possession via JWT-Bearer Assertion nach [RFC7523]) und die plattformabhängige Attestierung (Posture) absichern. Für solche Clients DARF NICHT eine identitätsgebundene E-Mail (F1) erzeugt oder verlangt werden; und es findet KEINE identitätsbezogene außerordentliche Löschung (A_29940 ff.) statt. [<=]

A_29895 -Authorization Server - Identitätslose Clients: Schlüssel-Rollover anwendbar

Der Authorization Server MUSS den Schlüssel-Rollover (A_29921 ff.) auch für identitätslose Clients zulassen, da er nur den Besitz des bisherigen Instanzschlüssels voraussetzt. Bei Verlust des Instanzschlüssels eines identitätslosen Clients existiert kein Recovery-Pfad; der Client MUSS sich neu registrieren. [<=]

A_29896 -Authorization Server - IDP-Authentisierung notwendig, aber nicht hinreichend

Der Authorization Server MUSS sicherstellen, dass eine erfolgreiche Authentisierung des Nutzers gegenüber dem sektoralen IDP allein NICHT ausreicht, um eine bestehende Client-Registrierung zu übernehmen, zu verändern oder wiederherzustellen. Jede solche Operation MUSS zusätzlich den Besitz bzw. die Kontrolle mindestens eines bei der Erstnutzung gebundenen Faktors aus {F1, F2} nachweisen (vgl. A_25649). [<=]

A_29897 -Authorization Server - Keine Faktoränderung allein aus IDP

Der gematik-Arte-Authentisierung Server DARF NICHT eine Änderung oder Neubindung eines Faktors allein auf Basis einer IDP-Authentisierung zulassen. [<=]

A_29898 -ZETA Guard - Eigenständige Vertrauensregistry je Fachdienst

Jeder ZETA Guard MUSS eine eigenständige, von anderen Guards unabhängige Vertrauensregistry führen. Der ZETA Guard DARF NICHT TOFU-Vertrauenszustände (gepinnte E-Mail, Instanzschlüssel, Tombstone-/Karenzzeit-Zustände) mit anderen Guards teilen oder von diesen beziehen. [<=]

A_29899 -ZETA Guard - Keine fact-Registry bereitgestellt, sodass die Anbieter von TI-2.0-Dienstübergreifende Korrelation

Der ZETA Guard DARF NICHT Identitätsdatensätze fachdienstübergreifend verknüpfen oder zur Korrelation eines Nutzers über mehrere Fachdienste hinweg verwenden. [<=]

Hinweis (Per-Guard-Modell): Dieselbe Identität ist je ZETA Guard ein eigenständiger Datensatz. Die gebundene E-Mail KANN zwischen Fachdiensten abweichen. Es existiert kein fachdienstübergreifendes Token; jeder Guard stellt eigene, DPoP-/audience-gebundene Token aus. Die fachdienstübergreifende Authentisierung ergibt sich

ausschließlich aus dem SSO der sektoralen IDPs, NICHT aus geteiltem ZETA-Zustand.

A_29900 -Authorization Server - Bindung an die TI-Identität

Ergänzend zu A_25650 (Bindung des registrierten Clients an eine TI-Identität) MUSS der Authorization Server diese Identität (KVNR oder Telematik-ID) als Ankerpunkt des TOFU-Identitätsdatensatzes führen, an den Faktor F1 auf Identitätsebene gebunden werden (vgl. A_29901). Die Ableitung der Identität aus der Nutzerauthentisierung regelt A_29903. [<=]

A_29901 -Authorization Server - Trennung von Identitäts- und Client-Ebene

Der Authorization Server MUSS die gebundene E-Mail (F1) auf Identitätsebene führen sowie den Instanzschlüssel (F2) und die berechtigten Audiences je Client-Instanz auf Client-Ebene führen (Datenhaltung vgl. A_26585-02). [<=]

A_29902 -Authorization Server - Verwaltung nur durch eingebuchten Client

Der Authorization Server MUSS für identitäts- und clientbezogene Verwaltungsoperationen (Änderung der E-Mail, Schlüssel-Rollover, Löschung von Clients) den Nachweis eines an diesem Fachdienst eingebuchten, gültigen Clients verlangen. [<=]

A_29903 -Authorization Server - Ableitung der Identitätsbindung aus der Nutzerauthentisierung

Der Authorization Server MUSS die Bindung einer Registrierung an die TI-Identität aus der OIDC-Nutzerauthentisierung ableiten (Statusmodell `pending_user_binding` gemäß A_29658 / [dcr-response-202.yaml]). Ein `realm-/mandantenweites Initial Access Token` ohne Identitätsbezug DARF NICHT als hinreichend für die Bindung der Identität verwendet werden. [<=]

5.15.2 Erstregistrierung (First Use)

A_29905 -Authorization Server - Freischaltung erst nach E-Mail-Verifikation

Klarstellend zum Statusmodell nach A_29658 (`pending_verification/pending_user_binding`) DARF der Authorization Server den vollen Berechtigungsumfang (Scope/Audiences) NICHT freigeben, bevor die E-Mail-Verifikation erfolgreich abgeschlossen ist. [<=]

A_29907 -Authorization Server - Instanzschlüssel im Besitz der Client-Instanz

Der Authorization Server MUSS den Instanzschlüssel (F2) als öffentlichen Schlüssel der Client-Instanz führen. Der zugehörige private Schlüssel DARF NICHT an den Authorization Server übertragen werden (vgl. A_25769). [<=]

Hinweis: Erstregistrierung und E-Mail-OTP-TOFU nutzen die **bereits definierte** Client-Registration-API `POST /register` (Body [dcr-request.yaml]) → `202 Accepted` ([dcr-response-202.yaml], `challenge_type: email_otp, transaction_id`) → `POST /register/verify` (OTP) → `201 Created` mit `client_id` (Status `pending_user_binding`); vgl. A_29658, A_25432-01.

5.15.3 Folgeregistrierung und E-Mail-Bindung

A_29909 -Authorization Server - Eine E-Mail je Identität; Verwaltung nur durch registrierten ihren spezifischen ZETA-Guard dClient

Der Authorization Server MUSS einer Identität genau eine verifizierte E-Mail-Adresse zuordnen, gebunden bei der Erstnutzung (TOFU). Der Authorization Server DARF NICHT das Hinzufügen einer weiteren E-Mail-Adresse nach der Erstnutzung zulassen. Nach erfolgreicher Erstregistrierung MUSS der Authorization Server sicherstellen, dass die Anwendungsfälle *Client löschen*, *Client umbenennen* und *E-Mail-Adresse aktualisieren* nur durch einen registrierten, gültigen (mobilen) Client durchgeführt werden. [<=]

A_30005 -Authorization Server - Beschränkung der Metadatenaktualisierung über RFC 7592

Der Authorization Server MUSS die Metadatenaktualisierung über PUT /register/{client_id} mit Content-Type: application/json (autorisiert allein durch das Registration Access Token nach [RFC7592]) ausschließlich auf die Umbenennung des Clients (client_name) beschränken. Der Authorization Server DARF NICHT über diesen Pfad eine Änderung des Instanzschlüssels (F2), der gebundenen E-Mail (F1), der berechtigten Audiences oder von DCR-Parametern (jwks, redirect_uris, grant_types) zulassen; entsprechende Felder MUSS er ablehnen. Der Schlüssel-Rollover erfolgt ausschließlich über Content-Type: application/zeta-rollover+jws (A_29921 ff.), die E-Mail-Änderung über POST /zeta/identity/email (A_29911). [<=]

A_29910 -Authorization Server - Verknüpfung weiterer Clients mit bestehender Identität

Der Authorization Server MUSS bei jeder weiteren Registrierung einer bereits bekannten Identität die gebundene E-Mail erneut verifizieren, DARF NICHT eine abweichende E-Mail-Adresse zulassen und MUSS den neuen Client mit dem bestehenden Identitätsdatensatz verknüpfen sowie seinen eigenen Instanzschlüssel (F2) binden. [<=]

A_29911 -Authorization Server - E-Mail-Änderung mit Step-up und überlebendem Faktor

Der Authorization Server MUSS für eine Änderung der gebundenen E-Mail eine ernetes-Umgebung konfigurieren und betreiben können.

ute (Step-up-)Authentisierung gegenüber dem IDP UND den Nachweis eines überlebenden Faktors (Besitz des Instanzschlüssels F2, Nachweis der Kontrolle über die bisherige E-Mail F1) verlangen sowie die neue Adresse verifizieren. Für die Step-up-Authentisierung MUSS der Authorization Server den Mechanismus nach [RFC9470] verwenden: Bei fehlender oder nicht ausreichender Nutzerauthentisierung MUSS er die Anfrage mit 401 und einem WWW-Authenticate-Header (error="insufficient_user_authentication", geforderte acr_values, optional max_age) ablehnen. Der ZETA Client MUSS daraufhin einmalig den regulären Authorization Code Flow (PAR → GET /authorize → Nutzerauthentisierung am sektoralen IDP → POST /token) mit den geforderten acr_values durchlaufen. Als Nachweis der Step-up-Authentisierung MUSS der Authorization Server das dabei von ihm selbst ausgestellte, DPoP-gebundene Access Token akzeptieren und dessen Signatur, die gemäß A_29994 aus dem ID-Token des sektoralen IDP übernommenen Claims acr/amr, ein aktuelles auth_time sowie die DPoP-Bindung ([RFC9449]) prüfen. [<=]

A_29912 -Authorization Server - Identitätsweite Wirkung der E-Mail-Änderung

Der Authorization Server MUSS eine erfolgreiche E-Mail-Änderung identitätsweit für alle Clients dieser Identität an diesem Fachdienst wirksam machen. [<=]

Hinweis: Die Folgeregistrierung (A_29910 und A_29911) erfolgt über die definierte API POST /register +POST /register/verify (erneute OTP-Verifikation der gebundenen E-Mail; A_29658). Die identitätsweite E-Mail-Änderung (vgl. A_29911 und A_29912) erfolgt über die Erweiterung POST /zeta/identity/email (Annex [zeta-guard-client-management]). Der Step-up-Nachweis wird dabei als vom Authorization Server ausgestelltes, DPoP-gebundenes Access Token im Feld idp_step_up übergeben; der zugehörige DPoP-Proof ([RFC9449], gebunden an Methode und Ziel-URI) im HTTP-Header DPoP. Die Verifikation der neuen Adresse erfolgt über POST /zeta/identity/email/verify (Annex [zeta-guard-client-management]); Challenge und Verify-Payload entsprechen dem definierten E-Mail-OTP-Mechanismus der Registrierung ([dcr-response-202.yaml], [verify-request.yaml], verify_type: email_otp). Erst nach erfolgreicher Verifikation wird die Änderung identitätsweit wirksam (A_29912).

Die Registrierung eines neuen Clients an einem weiteren ZETA Guard soll komfortabel und reibungsarm sein. Statt einer erneuten E-Mail-OTP-Erstnutzung KANN eine bereits

bestehende, gültige und faktorgestützte Registrierung derselben Identität an einem anderen Guard als Grundlage dienen. Der Pfad ist ausschließlich clientvermittelt (der Client trägt seinen ZETA Attestation Token vom Quell- zum Ziel-Guard); die Guards tauschen keinen Zustand direkt aus (vgl. A_29898 und A_29899). Da der Pfad über den Instanzschlüssel (F2) verankert ist — der aus einer IDP-Kompromittierung nicht ableitbar ist —, ist er mindestens so stark wie die E-Mail-OTP-Erstnutzung. E-Mail (F1) entwickelt sich anschließend je Guard eigenständig; eine Divergenz zwischen Fachdiensten ist ausdrücklich zulässig.

A_29913 -Authorization Server - Fachdienstübergreifende Fast-Path-Registrierung (Guard-Vouching)

Der Authorization Server MUSS als Alternative zur E-Mail-OTP-Erstnutzung eine fachdienstübergreifende Fast-Path-Registrierung zulassen (attestation_type=zeta attestation_token), bei der ein Client eine bestehende, gültige und faktorgestützte Registrierung derselben Identität an einem anderen ZETA Guard nachweist. Der Ziel-Guard MUSS in diesem Fall die Identitätsbindung und die verifizierte E-Mail (F1) aus einer geprüften, signierten ZETA Attestation Token des Quell-Guards übernehmen und DARF für diesen Pfad KEINE erneute E-Mail-OTP-Verifikation verlangen. Die Einlösung erfolgt als zusätzliches Credential auf der definierten API POST /register (vgl. A_29658).[<=]

A_29914 -Authorization Server - ZETA Attestation Token als Cross-Guard-Credential: Inhalt und Vertrauensanker

Der Authorization Server MUSS für die fachdienstübergreifende Fast-Path-Registrierung (A_29913) einen ZETA Attestation Token (attestation_type=zeta attestation_token) als Credential akzeptieren, der von einem von der gematik zugelassenen ZETA Guard signiert ist und für diesen Anwendungsfall mindestens enthält: das Identitätssubjekt (KVNR/Telematik-ID), die verifizierte E-Mail nebst Verifikationsstatus (F1), die Bestätigung des attestierten Schlüssels (cnf nach [RFC7800]/[RFC7638]), eine eindeutige Kennung (jti), eine Gültigkeitsdauer (exp) sowie eine Einschränkung des Empfängerkreises (aud) auf ZETA Guards.[<=]

A_29915 -Authorization Server - Prüfung des ZETA Attestation Token durch den Ziel-Guard

Der Authorization Server MUSS einen vorgelegten ZETA Attestation Token vollständig prüfen: (a) Signatur gegen das Zertifikat eines von der gematik zugelassenen ZETA Guards (TI-Vertrauensraum/Trust-Liste), (b) Empfängerkreis (aud) sowie (c) den DCR-Besitznachweis (signed_hash_puk_client_sig), der den neu vorgelegten Instanzschlüssel (F2) an den im Token bestätigten (attestierten) Schlüssel bindet. Schlägt eine dieser Prüfungen fehl, MUSS der Ziel-Guard die Fast-Path-Registrierung ablehnen. Der ZETA Attestation Token ist mehrfach und an mehreren Guards einlösbar. Da jede Registrierung an einen frischen, nur mit dem attestierten Schlüssel signierbaren Instanzschlüssel gebunden ist, ist eine Wiedereinspielung ohne den zugehörigen privaten Schlüssel nutzlos; der Besitznachweis ist NICHT challenge-gebunden. Der Schutz gegen missbräuchliche Einlösung ergibt sich aus der Pflicht-Benachrichtigung nach A_29920 (akzeptiertes Restrisiko).[<=]

A_29916 -Authorization Server - Faktorverankerung statt IDP-Ableitung

Der Authorization Server MUSS sicherstellen, dass der Fast-Path durch den Besitz des Instanzschlüssels (F2) autorisiert wird, der aus einer IDP-Kompromittierung nicht ableitbar ist. Eine erfolgreiche IDP-Authentisierung allein DARF NICHT ausreichen, um den Fast-Path (zeta_attestation_token) einzulösen (vgl. A_29896).[<=]

A_29918 -ZETA Guard - Eigenständige Weiterentwicklung nach Fast-Path-Registrierung (akzeptierte Divergenz)

Der ZETA Guard MUSS E-Mail (F1) nach einer Fast-Path-Registrierung eigenständig führen. Eine spätere E-Mail-Änderung (A_29911) an einem Guard DARF NICHT automatisch auf andere Guards übertragen werden; eine Divergenz der E-Mails zwischen den Guards ist zulässig und erwartbar.[<=]

A_29919 -ZETA Guard - Keine persistente fachdienstübergreifende Korrelation

Der ZETA Guard MUSS die die im ZETA Attestation Token offengelegte Herkunft (Quell-Guard) ausschließlich transient zur Prüfung verwenden und DARF NICHT eine fachdienstübergreifende Korrelation über das nach A_29899 zulässige Maß hinaus persistieren; insbesondere DARF NICHT die Identität des Quell-Guards als verknüpfbares Attribut der neuen Registrierung gespeichert werden. [\leq]

A_29920 -Authorization Server - Pflicht-Benachrichtigung bei Cross-Guard-Onboarding

Der Authorization Server MUSS bei jeder erfolgreichen fachdienstübergreifenden Fast-Path-Registrierung (vgl. A_29913 und A_29915) die aus dem ZETA Attestation Token übernommene E-Mail (F1) unverzüglich über den Notification Service benachrichtigen (vgl. A_25652, A_25735-01, A_25750). Die Benachrichtigung MUSS den neu registrierten Client erkennbar machen und auf die Widerrufsmöglichkeiten (Löschung nach A_29934, OOB-Löschung nach A_29940 ff.) hinweisen. [\leq]

Hinweis: Da jeder zugelassene Guard einen ZETA Attestation Token für den Cross-Guard-Fast-Path ausstellen kann, ist die Benachrichtigung des Identitätsinhabers die maßgebliche Erkennungs- und Widerspruchsmöglichkeit gegen ein missbräuchliches Vouching eines kompromittierten Quell-Guards.

5.15.4 Schlüssel-Rollover (Proof-of-Possession)**A_29921 -Authorization Server - Schlüssel-Rollover mit Proof-of-Possession**

Der ZETA Client MUSS seinen Instanzschlüssel regelmäßig wechseln (Intervall durch gematik festgelegt). Der Wechsel MUSS als In-Place-Rollover der bestehenden Registrierung erfolgen und durch eine Signatur mit dem bisherigen (aktiven) Instanzschlüssel autorisiert werden (Proof-of-Possession). Der Authorization Server DARF NICHT einen Schlüsselwechsel allein auf Basis des Registration Access Token nach [RFC7592] zulassen. [\leq]

Hinweis: Der Client Instance Key soll initial 2 Jahre gültig sein.

A_29922 -Authorization Server - Verschachtelte Signatur (Autorisierung und Anti-Substitution)

Der Authorization Server MUSS für den Rollover eine verschachtelte JWS-Struktur verlangen, bei der die äußere Signatur mit dem bisherigen Schlüssel (Autorisierung) und die innere Signatur mit dem neuen Schlüssel (Nachweis der Kontrolle über den neuen Schlüssel) erfolgt. [\leq]

A_29923 -Authorization Server - Replay- und Fehlleitungsschutz

Der Authorization Server MUSS den Rollover gegen Wiedereinspielung und Fehlleitung absichern durch (a) einen serverseitig ausgestellten Einmal-Nonce, (b) eine Bindung an die Identität des Guard (Audience) sowie (c) eine Bindung an HTTP-Methode und Ziel-URL der Anfrage. [\leq]

A_29924 -Authorization Server - Overlap-Fenster: Ressourcenzugriff vs. Verwaltung

Der Authorization Server MUSS nach erfolgreichem Rollover den bisherigen Schlüssel für einen begrenzten Übergangszeitraum ausschließlich für den Ressourcenzugriff weiter akzeptieren. Der bisherige Schlüssel DARF NICHT nach dem Rollover weitere Verwaltungsoperationen (insbesondere einen erneuten Rollover) autorisieren. [\leq]

A_29925 -Authorization Server - Invalidierung des bisherigen Schlüssels

Der Authorization Server MUSS den bisherigen Schlüssel nach Ablauf des Übergangszeitraums vollständig invalidieren und aus dem Schlüsselmaterial des Clients entfernen. [\leq]

A_29926 -Authorization Server - Keine Kopplung von Rollover und Faktoränderung

Der Authorization Server DARF NICHT einen Schlüssel-Rollover mit einer Änderung von E-Mail (F1) in einer untrennbaren Operation zusammenfassen. [<=]

A_29927 -Authorization Server - Abgrenzung zum Schlüsselverlust

Der In-Place-Rollover setzt den Besitz des bisherigen Instanzschlüssels voraus. Bei Verlust DARF der Rollover-Pfad NICHT verwendet werden; verfügt der Nutzer noch über ein weiteres eingebuchtes Gerät (F2) oder die Kontrolle über die gebundene E-Mail (F1), erfolgt die (Neu-)Einbuchung als Folgeregistrierung mit erneuter E-Mail-Verifikation (A_29910). Sind alle Faktoren {F1, F2} verloren, ist keine Selbst-Wiederherstellung möglich; die Identität kann nur über die außerordentliche Löschung (OOB, A_29940 ff.) mit anschließender Erstnutzung neu etabliert werden. [<=]

87646A_28798 -ZETA Guard - AusführA_29933 -ZETA Client - Ersatzgerät bei E-Mail-Verlust

Der ZETA Client SOLL einen Nutzer, der den Zugriff auf die gebundene E-Mail verloren hat und über ein weiteres registriertes Gerät verfügt, anleiten, zunächst von diesem Gerät die E-Mail zu ändern (A_29911) und erst danach das Ersatzgerät zu registrieren (vgl. Nutzerunterstützung A_25732). [<=]

5.15.5 Verwaltung von Clients/Geräten

A_29934 -Authorization Server - Löschung anderer Clients nur auf Client-Ebene

Der Authorization Server MUSS einem eingebuchten Client die Löschung eines anderen Clients derselben Identität erlauben und DARF NICHT bei dieser Löschung identitätsgebundenen Faktor (F1) verändern oder entfernen. Die clientseitige Darstellung und Auslösung erfolgt gemäß A_25733; die Höchstzahl registrierbarer Clients bleibt nach A_25748-02 begrenzt. [<=]

A_29935 -Authorization Server - Benachrichtigung bei Client-Löschung

Der Authorization Server MUSS bei der Löschung eines Clients die gebundene E-Mail (F1) der Identität benachrichtigen (vgl. Notification Service A_25652, Push A_25735-01, Nutzerinformation A_25750). [<=]

A_29936 -Authorization Server - Erhöhte Anforderung bei Löschung des letzten/aller Clients

Der Authorization Server MUSS für die Löschung des letzten verbleibenden Clients einer Identität eine Step-up-Authentisierung verlangen ODER die Löschung mit einer Einspruchsfrist (Veto-Fenster) ausführen. Das Veto-Fenster wird analog A_29942 ausgestaltet (geplante Löschung mit Benachrichtigung, Abbruch durch einen überlebenden Faktor gemäß A_29943); die Einlegung des Vetos erfolgt über POST /zeta/deletions/{deletion_id}/veto. Eine Sammellöschung mehrerer Clients ist KEINE eigene Serveroperation; sie erfolgt clientseitig als wiederholte Einzel-Löschung (DELETE /zeta/clients/{target_client_id}). Die erhöhte Anforderung greift, sobald dabei der letzte verbleibende Client gelöscht würde. Die Step-up-Authentisierung erfolgt nach demselben Mechanismus wie bei der E-Mail-Änderung (A_29911, [RFC9470]): 401 mit error="insufficient_user_authentication" und geforderten acr_values, einmaliger regulärer Authorization Code Flow, Nachweis durch das vom Authorization Server ausgestellte, DPoP-gebundene Access Token. [<=]

A_29937 -Authorization Server - Fachdienstbezogener Geltungsbereich der Löschung

In Anwendung des Per-Guard-Prinzips (vgl. A_29898) MUSS der Authorization Server Client-Löschungen ausschließlich auf den eigenen Fachdienst beziehen; eine fachdienstübergreifende Löschung DARF NICHT serverseitig erfolgen. [<=]

A_29938 -Authorization Server - Fortbestand der Identität nach Löschung des letzten Clients

Der Authorization Server MUSS den Identitätsdatensatz (gebundene E-Mail F1) nach Löschung des letzten Clients erhalten (unbeschadet der Inaktivitäts-Löschfristen nach A_28808). Eine erneute Einbuchung MUSS als Folgeregistrierung mit erneuter Verifikation der bestehenden E-Mail behandelt werden. [<=]

A_29939 -Authorization Server - Autorisierungsprüfung bei Client-übergreifender Löschung

Der Authorization Server MUSS bei der Löschung eines anderen Clients (DELETE /zeta/clients/{target_client_id}) prüfen, dass der Ziel-Client tatsächlich zur Identität des aufrufenden Registration Access Token gehört, und einen nicht zur Identität gehörenden Ziel-Client ablehnen (403). Der Besitz einer Objektreferenz (target_client_id) allein DARF NICHT als Autorisierung gelten (Schutz gegen Broken Object Level Authorization). [<=]

Genutzte Schnittstellen (Annex [zeta-guard-client-management]): GET /zeta/clients , DELETE /zeta/clients/{target_client_id}, DELETE /register/{client_id} (Selbst-Deregistrierung, RFC 7592).

5.15.6 Außerordentliche Löschung (Out-of-Band) und Protokollierung

A_29940 -Authorization Server - OOB-Löschung als Notfallpfad

Der Authorization Server MUSS einen außerordentlichen Löschpfad bereitstellen, wenn ein Nutzer alle online verfügbaren Faktoren (F1, F2) verloren hat und eine Selbst-Recovery nicht mehr möglich ist. Der Prozess DARF NICHT Zugriff gewähren, ein Token ausstellen oder einen Faktor neu binden. [<=]

A_29941 -Authorization Server - IDP-unabhängiger, hochassuranter Identitätsnachweis

Der Authorization Server MUSS für die OOB-Löschung einen vom sektoralen IDP unabhängigen Identitätsnachweis auf hohem Vertrauensniveau verlangen. Ein Nachweis auf niedrigerem Niveau DARF NICHT akzeptiert werden. [<=]

A_29942 -Authorization Server - Verzögerte Ausführung mit Veto-Fenster

Der Authorization Server MUSS eine OOB-Löschung mit einer Einspruchsfrist planen und DARF sie NICHT sofort ausführen. Der Authorization Server MUSS während der Frist alle überlebenden, gebundenen Kanäle benachrichtigen (vgl. A_25750). Die Dauer der Einspruchsfrist MUSS über Policy definiert und dokumentiert sein und MUSS mindestens so bemessen sein, dass die Benachrichtigung zugestellt und ein überlebender Faktor rechtzeitig ein Veto einlegen kann. [<=]

A_29943 -Authorization Server - Veto durch überlebenden Faktor

Der Authorization Server MUSS während der Einspruchsfrist den Abbruch der geplanten Löschung durch den Proof-of-Possession eines noch eingebuchten Clients (F2) zulassen. Ein Veto allein durch Kontrolle der gebundenen E-Mail (F1) ist NICHT vorgesehen. Verfügt der Nutzer über keinen eingebuchten Client mehr, MUSS er zunächst ein Gerät neu registrieren (Folgeregistrierung mit erneuter Verifikation der gebundenen E-Mail F1, A_29910) und kann anschließend mit diesem Client das Veto einlegen. [<=]

A_29944 -Authorization Server - Wirkung der Ausführung: Tombstone, kein Zugriff

Der Authorization Server MUSS bei Ausführung der OOB-Löschung den Identitätsdatensatz in einen Tombstone-Zustand überführen und alle zugehörigen Client-Registrierungen entfernen. Die OOB-Löschung DARF NICHT Zugriff gewähren oder einen Faktor neu binden. [<=]

A_29945 -Authorization Server - Zweite, unabhängige OOB-Bestätigung für Wiederregistrierung

Der Authorization Server MUSS nach einem Tombstone die Erstnutzung der betroffenen Identität sperren und diese Sperre erst mit dem Erfassen einer zweiten, unabhängigen

OOB-Bestätigung aufheben. Diese Bestätigung MUSS durch einen anderen Operator erfolgen als die Ausführung der Löschung (Funktionstrennung). Diese Funktionstrennung ist ein zweiter, eigenständiger Vier-Augen-Schritt in einer späteren Lebenszyklus-Phase (Wiederregistrierung nach dem Tombstone) und NICHT identisch mit dem Vier-Augen-Prinzip bei der Auslösung der Löschung (A_29946). Die Sperre ist ereignisgesteuert: Sie endet ausschließlich mit dieser Bestätigung und DARF NICHT allein durch Zeitablauf aufgehoben werden (bleibt ohne Bestätigung unbegrenzt bestehen). [<=]

A_29946 -Authorization Server - Funktionstrennung bei der Löschung (Vier-Augen-Prinzip)

Der Authorization Server SOLL die Auslösung einer OOB-Löschung einem Vier-Augen-Prinzip unterwerfen, bei dem die Freigabe der Löschung durch einen anderen Operator erfolgt als deren Initiierung. Diese Funktionstrennung wirkt vor der Ausführung (Tombstone, A_29944) und ist ein von der zweiten, unabhängigen Bestätigung bei der Wiederregistrierung (A_29945) getrennter, früherer Vier-Augen-Schritt: A_29943 sichert dasVeto der Löschung, A_29945 die spätere Wiederregistrierung nach dem Tombstone. [<=]

A_29947 -Authorization Server - Vorrang der Selbst-Recovery

Der Authorization Server SOLL vor einer OOB-Löschung prüfen, ob noch ein überlebender Faktor vorliegt, und in diesem Fall auf die Selbst-Recovery verweisen. [<=]

A_29948 -Authorization Server - Revisions sichere Protokollierung

Der Authorization Server MUSS alle sicherheitsrelevanten Lebenszyklus-Ereignisse (Erst-/Folgeregistrierung, Bindung/Verifikation von Faktoren, Rollover, Recovery, Client-Löschung, OOB-Planung/Veto/Ausführung, Wiederregistrierungs-Bestätigung) revisions sicher protokollieren, jeweils mit Zeitstempel und — bei operatorgetriebenen Aktionen — der Operatoridentität und Begründung (erweitert A_25738, A_25749). [<=]

A_29949 -ZETA Guard - Absicherung des OOB-Administrationskanals

Der ZETA Guard MUSS den Administrationskanal für OOB-Operationen über gegenseitige TLS-Authentisierung **in einem Ku**(mTLS nach [RFC8705]) absichern und für die Operatorauthentisierung eine vom sektoralen IDP unabhängige Identitätsquelle verwenden. [<=]

Hinweis: Genutzte Schnittstellen Operator-Pfad — Annex zeta-guard-admin-oob.yaml (POST /deletions , POST /deletions/{deletion_id}/approve, POST /identities/{identity_ref}/reregister-confirm , mTLS + IDP-unabhängige Operator-Auth nach [RFC8705]). Clientseitiges Veto — Annex [zeta-guard-client-management]POST /zeta/deletions/{deletion_id}/veto.

5.16 ZETA Guard

Die Software des ZETA Guards wird im Auftrag der gematik entwickelt und alle Komponenten als signierte OCI Images und als signiertes Helm Chart in einer gematik Artifact Registry **bernetes-Cluster** bereitgestellt, sodass die Anbieter von TI 2.0 Diensten ihren spezifischen ZETA Guard darauf aufbauend in ihrer Kubernetes Umgebung konfigurieren und betreiben können.

A_28798 -ZETA Guard - Ausführung in einem Kubernetes Cluster

Der Anbieter eines TI 2.0-Dienstes MUSS ZETA Guard grundsätzlich in einem Kubernetes Cluster betreiben.

Wenn z. B. in einer VAU die Nutzung von Kubernetes nicht sinnvoll möglich ist, dann

MUSS der Anbieter des TI 2.0-Dienstes nachweisen, dass seine Lösung ohne Kubernetes eine hinreichende Verfügbarkeit, Skalierbarkeit und Betriebsstabilität ermöglicht. [\leq]

A_26519 -ZETA Guard, Unterstützung von Service-Mesh Lösungen

Die Komponenten des ZETA Guard MÜSSEN es ermöglichen, dass Service-Mesh Lösungen zur Verwaltung der Komponenten eingesetzt werden können. [\leq]

A_26521 -ZETA Guard, Unterstützung von Canary Releases

Die Komponenten des ZETA Guard MÜSSEN Canary Releases unterstützen. [\leq]

A_28851 -ZETA Guard - Architektur der TLS-Terminierung

Der Anbieter des TI 2.0-Dienstes MUSS sicherstellen, dass alle von außen eingehenden Verbindungen zum PEP HTTP Proxy und zum PDP Authorization Server über TLS abgesichert sind.

Die TLS-Terminierung KANN dabei flexibel erfolgen:

- durch ein dem ZETA Guard vorgeschaltetes System (z. B. CDN, WAF oder externer Ingress),
- durch den ZETA-internen Ingress / API-Gateway,
- oder direkt an den Komponenten PEP HTTP Proxy und PDP Authorization Server.

[\leq]

Hinweis: Die TLS-Terminierung für den PDP Authorization Server und den PEP HTTP Proxy kann über eine gemeinsame TLS-Verbindung (gleicher FQDN, gleicher Port) abgebildet werden.

886A_29868 -ZETA Guard - Dual-Stack bei TLS-Terminierung

Der Anbieter des TI 2.0-Dienstes MUSS sicherstellen, dass die gemäß A_28851 vorgesehenen TLS-Terminierungspunkte über IPv4 und IPv6 erreichbar sind. Die Sicherheitsmechanismen (z. B. Zertifikatsprüfung, Rate-Limiting, Protokollierung) MÜSSEN für beide IP-Versionen gleichwertig umgesetzt werden. [\leq]

A_28852 -ZETA Guard - TLS-Terminierung und ASL bei Einsatz einer VAU

Falls der ZETA Guard Daten mit dem Schutzbedarf „sehr hoch“ verarbeitet und in einer VAU (Vertrauenswürdige Ausführungsumgebung) betrieben wird, MUSS der Anbieter des TI 2.0-Dienstes sicherstellen, dass die TLS-Verbindung des Clients erst innerhalb der VAU terminiert wird (z. B. am PEP und PDP oder Ingress innerhalb der VAU).

Wird die TLS-Verbindung stattdessen an einer Komponente außerhalb der VAU terminiert (z. B. durch ein externes CDN oder eine WAF), MUSS zwingend der ZETA/ASL-Kanal (Application Security Layer) für PEP und PDP Endpunkte verwendet werden, um die Vertraulichkeit und Integrität der Daten vom Client bis in die VAU sicherzustellen.

[\leq]

A_25666-01 -ZETA Guard - TLS-Terminierung

Die Komponente Ingress MUSS TLS für von außen eingehende Verbindungen terminieren können.

[\leq]

Hinweis: TLS-Verbindungen von ZETA Clients können dort terminiert werden, wo es aus Sicht des Anbieters des TI 2.0-Dienstes am sinnvollsten ist (z. B. an einem CDN). Um dennoch eine verschlüsselte Verbindung bis zum ZETA Guard zu haben, kann ZETA/ASL verwendet werden.

A_26964-01 -ZETA Guard - OCSP Stapling

Komponenten innerhalb des ZETA Guards (inkl. Ingress), die von außen kommende TLS-Verbindungen terminieren, SOLLEN OCSP-Stapling [RFC6066] verwenden.

[\leq]

A_26639 -ZETA Guard - Unterstützung WebSocket

Die Komponente Ingress MUSS das WebSocket-Protokoll unterstützen. [\leq]

A_26640 -ZETA Guard - HTTP Protokoll-Versionen

Die Komponente Ingress MUSS das HTTP Protokoll in den Versionen HTTP/1.1, HTTP/2 und SOLL HTTP/3 unterstützen. [≤]

A_25652 -ZETA Guard - Notification Service

Der ZETA Guard Notification Service MUSS Push-Notifications über die von App-Anbietern bereitgestellten Push-Gateways unterstützen, um die Notifications an bestimmte oder alle registrierte Clients eines Anwenders verschicken zu können.

Der Notification Service MUSS gemäß [gemF_PushNotification] die Push Konfigurationen von ZETA Clients registrieren und verwalten können und MUSS die vom Resource Server empfangenen Notification Events an die dafür registrierten Push Gateways der Clients weiterleiten. [≤]

A_25737 -ZETA Guard - Push Notification

Der ZETA Guard MUSS eine Push Benachrichtigung an alle registrierten Clients des Nutzers, für die eine Push Notification aktiviert ist, verschicken, sobald sich Änderungen an der Liste der registrierten Clients dieses Nutzers ergibt. [≤]

A_26988 -Telemetriedaten Service - Fehlermeldungen

Alle Komponenten des ZETA Guard MÜSSEN ihre Fehlermeldungen so bereitstellen, dass der Telemetriedaten Service die Fehlermeldungen sammeln und an einen Monitoring Service weiterleiten kann. [≤]

A_26661 -ZETA Guard - HTTP Statuscodes

Der ZETA Guard MUSS die HTTP Statuscodes gemäß Tabelle "ZT_HTTP_Statuscodes" unterstützen. [≤]

A_26662 -ZETA Guard, HTTP Fehlerdetails

Der ZETA Guard MUSS bei Beantwortung eines Requests mit einem HTTP Fehler ein JSON Object ergänzen, das den Fehler beschreibt. Das JSON Object MUSS gemäß [zeta-error.yaml] aufgebaut sein. [≤]

Beispiel für eine sinnvolle Ergänzung der Fehlerdetails:

Bei der Authentifizierung mit Client Assertion JWT fehlt im JWT eine Angabe product_version oder die product_version ist nicht in der Policy Engine bekannt, dann antwortet der Authorization Server mit HTTP Status 400 Bad Request sowie einer Beschreibung, dass die product_version fehlt oder nicht bekannt ist.

66329A_27264802-03 -ZETA Guard, Telemetriedaten

Alle JWT Prüfung

Der ZETA Guard Komponenten mit Ausnahme der Datenbanken MÜSSEN OpenTelemetry-konforme Traces, Metriken, MUSS bei der Prüfung von JWT die Prüfschritte gemäß <https://www.rfc-editor.org/rfc/rfc7519.html#section-7.2> und <https://www.rfc-editor.org/rfc/rfc7515.html#section-5.2> erzeugen und durchführen.

[≤]

87693A_27802-029046 -ZETA Guard, JWT spezifische Prüfungen für TI-Zertifikate

Der ZETA Guard MUSS bei der Prüfung von JWT die Signaturen mit TI-Zertifikaten zusätzlich folgende Prüfungen durchführen:

- Integrität der Nachricht: Hash-Überprüfung: Sicherstellen, dass die Nachricht nicht manipuliert wurde. Prüfschritte gemäß <https://www.rfc-editor.org/rfc/rfc7519.html#section-7.2>
- Ablaufdatum des Zertifikats: Sicherstellen, dass das Zertifikat noch gültig ist.
- Widerrufsstatus des Zertifikats (OCSP): Prüfen, ob das Zertifikat nicht widerrufen wurde. Die OCSP Response wird für eine konfigurierbare Dauer zwischengespeichert, um zu häufige OCSP Anfragen und <https://www.rfc-editor.org/rfc/rfc7515.html#section-5.2> zu verhindern. Die durchführen. auer soll der Gültigkeitsdauer des Refresh Token entsprechen.

- Vertrauenswürdigkeit der Zertifikatskette: Sicherstellen, dass die Kette zu einer vertrauenswürdigen Root-CA führt.

[<=]

A_28963 -ZETA Guard, DPoP Prüfung

Der ZETA Guard MUSS bei der Prüfung von JWT die Prüfschritte gemäß [\[RFC9449\]#name-checking-dpop-proofs](#) durchführen.

[<=]

A_26668-02 -ZETA Guard - Rate Limit

Die Komponenten des ZETA Guard MÜSSEN für ihre durch ZETA Clients erreichbaren Endpunkte ein Rate Limit konfigurierbar einstellen können. Wenn ein Rate Limit konfiguriert ist, dann muss der Client über folgende Response Header informiert werden:

- ((das erlaubte Limit),
- (verbleibende Anfragen) und
- (Zeitpunkt, an dem das Limit zurückgesetzt wird)).

[<=]

Hinweis: Es gibt einen RFC Draft, der Rate Limits neu spezifiziert (<https://datatracker.ietf.org/doc/draft-ietf-httpapi-ratelimit-headers/>). Es ist geplant, dass nach Veröffentlichung des RFC ZETA Guard angepasst wird, um die neuen Rate Limit Festlegungen zu unterstützen.

~~88037A~~ 27864-012 -ZETA Guard - Egress

Der ZETA Guard MUSS eine Egress Funktion implementieren, die durchsetzt, dass nur erlaubte ZETA Guard Verbindungen, zu Endpunkten außerhalb des Kubernetes Clusters, aufgebaut werden.

Zu den erlaubten ZETA Guard Verbindungen gehören mindestens:

- Telemetriedaten-Empfänger der gematik
- SIEM der gematik
- Sektorale IDPs (wenn Versicherte zur Nutzergruppe gehören)
- Federation Master der TI
- ZETA Artifact Registry
- DNS Resolver
- OCSP Responder oder CRL (TLS TSPs nach CAB Forum)
- SMC-B TSP OCSP Responder
- OCSP Responder des TSP der Komponenten PKI der TI
- Notification Services der App Hersteller
- Mail Server des TI 2.0 Dienst Anbieters (für TOFU OTP Codes; wenn Versicherte zur Nutzergruppe gehören)

- Weitere erPoPP Service (Download des JWKS, wenn PoPP Token durch den TI 2.0 Dienst verwendet werden)

Weitere erlaubte Verbindungen zu Endpunkten des Anbieters, zu Endpunkten von Clientsystem Notification Services oder für Dienst-zu-Dienst Kommunikation können hinzugefügt werden.[<=]

8A_29869 -ZETA Guard - Dual-Stack in Egress- und Zielauflösung

Die Egress-Funktion MUSS Verbindungen zu erlaubten Zielen über IPv4 und IPv6 unterstützen. Bei FQDN-basierten Zielen MÜSSEN A- und AAAA-Auflösungen verarbeitet werden können[<=]

A_28435 -ZETA Guard, Ingress - Unterstützung Forwarded-Header

Die Komponente Ingress MUSS in jeder empfangene HTTP-Anfrage für die nachgelagerten Komponenten PDP Authorization Server und PEP HTTP Proxy den Forwarded-Header gemäß [RFC 7239] in der weitergeleiteten Anfrage hinzufügen oder aktualisieren. [≤]

8769A_29870 -ZETA Guard - Verarbeitung von IPv6 in Forwarded-Headern

Die Komponenten des ZETA Guard MÜSSEN IPv6-Adressen in Forwarded-Headern gemäß RFC 7239 korrekt verarbeiten und an nachgelagerte Komponenten konsistent weiterreichen. [≤]

A_28526-01 -ZETA Guard - Bereitstellung der lokalen Artifact Registry

Der Anbieter eines TI 2.0 Dienstes MUSS für seinen ZETA Guard eine lokale Artifact Registry bereitstellen, die alle benötigten Container Images aus der gematik Artifact Registry enthält.

Die lokale Artifact Registry MUSS so konfiguriert werden, dass

- OCI Container Images für die ZETA Guard Komponenten in der benötigten Version vorhanden sind,
- das ZETA Guard Provisioning Container Image alle 60 Minuten aktualisiert wird.

[≤]

Hinweis: Die Verfügbarkeit von ZETA Guard soll durch die lokale Bereitstellung der Artifact Registry erhöht werden. Die Policies und Daten werden von der Policy Engine direkt aus der ZETA Artifact Registry geladen und aktualisiert.

5.16.1 Klassifizierung der ZETA Guard Komponenten

Der ZETA Guard ist ein logisches Bundle das in der Laufzeitumgebung des TI-2.0-Diensteanbieters (in der Regel ein Kubernetes-Cluster) betrieben wird. Die Komponenten des ZETA Guard werden nach ihrem Funktionsumfang, ihrer Austauschbarkeit und der Verantwortung für die Bereitstellung klassifiziert. Diese Klassifizierung bestimmt die Möglichkeiten bei der Integration von ZETA Guard in die individuelle Laufzeitumgebung des TI-2.0-Diensteanbieters und die Sicherheitsvorgaben z. B. beim Betrieb in einer VAU.

5.16.1.1 Unveränderliche Kernkomponenten

Diese Komponenten bilden den funktionalen Kern des ZETA Guard. Sie werden von der gematik als signierte OCI-Images in der ZETA Artifact Registry bereitgestellt.

A_28789 -ZETA Guard - Integrität der Kernkomponenten

Der Anbieter eines TI 2.0-Dienstes MUSS die folgenden Komponenten zwingend als unveränderte Images der gematik beziehen und deren Signatur vor dem Deployment validieren:

- PEP HTTP Proxy
- PDP Authorization Server
- PDP Policy Engine (OPA)
- Telemetriedaten-Service
- Notification Service (nur für mobile Clients)

Wenn z. B. in einer VAU-Umsetzung die Images der gematik nicht unverändert übernommen werden können, dann MUSS per Attestation nachgewiesen werden, dass das verwendete Image auf dem unveränderten Source-Code der gematik basiert.

[≤]

5.16.1.2 Austauschbare Kernkomponenten

Diese Komponenten sind für den Betrieb des ZETA Guard notwendig. Der Anbieter des TI 2.0-Dienstes kann wählen, ob die von gematik bereitgestellten Images verwendet werden oder ob eigene Lösungen eingesetzt werden.

A_28790 -ZETA Guard - Kernkomponenten, Verwendung von eigenen Lösungen

Der Anbieter eines TI 2.0-Dienstes MUSS bei Verwendung eigener Lösungen der folgenden Infrastruktur-Komponenten sicherstellen, dass die Anforderungen an die Komponenten erfüllt werden und dass die Schnittstellen zur Kernkomponenten-Schicht kompatibel sind.

Folgende ZETA Guard Komponenten können durch eigene Lösungen ersetzt werden:

- PDP Datenbank
- HSM Proxy
- Local Artifact Registry Cache
- Gateway oder Ingress und Egress

[<=]

Hinweis: Bei der Verwendung von eigenen Lösungen für ZETA Guard Kernkomponenten muss die Umsetzung und Einhaltung der zu den Komponenten gehörenden Anforderungen gemäß Anbietertypsteckbrief nachgewiesen werden. Im ZETA Guard Produkthandbuch ist beschrieben, was hinsichtlich Kompatibilität zu den ZETA Guard Kernkomponenten zu beachten ist.

A_28432 -ZETA Guard, Komponenten Ingress optional

Der Ingress MUSS als optionale Komponente im ZETA Guard angeboten werden. [<=]

Es ist möglich auf die Komponente Ingress im ZETA Guard zu verzichten. Dadurch wird der Hersteller des TI 2.0-Dienstes in die Lage versetzt seinen eigenen externen Ingress zu verwenden.

A_28433 -ZETA Guard, Bereitstellung externer Ingress

Der Hersteller des TI 2.0-Dienstes MUSS entweder den Kubernetes Ingress des ZETA Guard oder einen eigenen Ingress verwenden. [<=]

5.16.1.3 Hilfskomponenten und Funktionen

Diese Komponenten und Funktionen bieten zusätzliche Funktionen zur Verbesserung der Sicherheit und zur Vereinfachung des Betriebs. Sie können durch eigene Lösungen umgesetzt werden. ZETA Guard enthält keine fertigen Images, sondern ausschließlich unterstützende Konfigurationen (z. B. Policies für einen Admission Controller).

A_28792 -ZETA Guard - Hilfskomponenten und Funktionen

Der Anbieter eines TI 2.0-Dienstes MUSS bei Verwendung eigener Lösungen der Hilfskomponenten sicherstellen, dass die Anforderungen an die Komponenten erfüllt werden und dass die Schnittstellen zur Kernkomponenten-Schicht kompatibel sind.

Die folgenden Komponenten zählen zu den Hilfskomponenten:

- Admission Controller
- Service Mesh

[<=]

Hinweis: Bei der Verwendung von eigenen Lösungen für ZETA Guard Hilfskomponenten und Funktionen muss die Umsetzung und Einhaltung der zu den Komponenten

gehörenden Anforderungen gemäß Anbietertypsteckbrief nachgewiesen werden. Für die Hilfskomponenten und Funktionen werden keine Images durch die gematik bereitgestellt.

5.16.2 Kommunikation mit Diensten der gematik

ZETA Guard Instanzen benötigen Zugriff auf Dienste der gematik, um PIP und PAP OCI Container Images zu laden, um Telemetriedaten an den gematik Empfänger zu senden und um Security Events an das gematik SIEM zu senden. Für den Zugriff auf diese Dienste ist ein gültiges Access Token erforderlich.

Um ein gültiges Access Token zu erhalten wird Workload Identity Federation eingesetzt. Voraussetzung dafür ist, dass

- der Kubernetes Cluster, in dem ZETA Guard ausgeführt wird, seine OpenID Konfiguration und seine JWKS URI (`/.well-known/openid-configuration` und bei Kubernetes IDP/openid/v1/jwks) öffentlich erreichbar bereitstellt (alternativ kann ein Schlüssel (JWK) im Workload Identity Federation Pool der gematik registriert werden),
- ein ServiceAccount innerhalb des Clusters existiert, der die benötigten Secrets (Access Token) für die Workloads bereitstellt und
- die Issuer URI des Kubernetes Cluster bei der gematik registriert ist.

Siehe auch Kapitel [5.16.3.8- Prozesse zur Inbetriebnahme eines ZETA Guard](#).

Hinweis: Der Kubernetes IDP wird vom kube-apiserver umgesetzt. Der API-Server ist die Instanz, die die Identitätsnachweise (Tokens) signiert und die notwendigen Metadaten für externe Dienste bereitstellt. Damit dies funktioniert, muss er mit bestimmten Flags konfiguriert sein (`--service-account-issuer`, `--service-account-signing-key-file`, `--service-account-jwks-uri`).

Für die Workload Identity Federation (WIF) nutzt Kubernetes die TokenRequest API. Diese API ermöglicht es, kurzlebige, zielgruppenspezifische (Audience-bound) JSON Web Tokens (JWT) zu erstellen. Ein externer Dienst (gematik Telemetriedaten Empfänger) akzeptiert nur Tokens,

- *deren Issuer im gematik WIF Pool registriert sind (per Onboarding Prozess für den TI 2.0 Dienst Anbieter) und*
- *die eine für ihn definierte aud (Audience) enthalten.*

Damit die gematik Dienste die Token validieren können, muss entweder der Token Signatur-Schlüssel bei der gematik registriert sein oder es werden die standardisierten Endpunkte des api-servers öffentlich bereitgestellt:

- */.well-known/openid-configuration: Das Discovery-Dokument, das die unterstützten Funktionen und die URL zum Schlüsselsatz enthält.*
- */openid/v1/jwks: Der JSON Web Key Set (JWKS), der die öffentlichen Schlüssel zur Verifizierung der Signatur enthält.*

5.16.3 Deployment Szenarien

5.16.3.1 Geo-Redundanz

Der Betrieb von Kubernetes in einer Multi-Cluster-Umgebung bietet Unternehmen erhebliche Vorteile in Bezug auf Skalierbarkeit, Ausfallsicherheit und Flexibilität, bringt jedoch auch eine erhöhte Komplexität in Verwaltung, Netzwerk und Sicherheit mit sich und es ergeben sich Anforderungen an Anbieter von TI 2.0-Diensten.

A_28438-01 -ZETA Guard, geo-redundanter Betrieb

Der Anbieter eines TI 2.0-Dienstes MUSS beim geo-redundanten Betrieb des ZETA Guard in einer Kubernetes Multi-Cluster-Umgebung folgende Bedingung erfüllen:

- Die Authorization Server Instanzen müssen über einen globalen Load Balancer als ein logischer Authorization Server bereitgestellt werden.

[<=]

A_28464 -ZETA Guard, genau ein Authorization Server

Die Komponente PEP HTTP Proxy MUSS das OAuth Protected Resource Well-known so bereitstellen, dass für ZETA Clients der ZETA Guard Authorization Server als genau eine logische Komponente bereitgestellt wird (nur ein `Eintragauthorization_servers` im OAuth Protected Resource Well-known).[<=]

5.16.4 LaufzeitüProvisioning Image für den ZETA Guard

5.16.5 Das Provisioning Image dient der sicheren Bereitstellung und Verteilung von kryptografischen Trust Anchors (Vertrauensankern) und Konfigurationsdaten, die der ZETA Guard zur Verifikation von Signaturen, Attestierungen und Zertifikaten benötigt. Es handelt sich um ein rein datenführendes OCI Image, welches von der gematik signiert berwachung

Um-eitgestellt wird.

Pfad für die Produktivumgebung:

europe-west3-docker.pkg.dev/gematik-pt-zeta-prod/zeta-provisioning/zeta-guard-provisioning:latest

Pfad für die Testumgebung:

europe-west3-docker.pkg.dev/gematik-pt-zeta-test/zeta-provisioning/zeta-guard-provisioning:latest

5.16.5.1 Struktur und Inhalt

Das Provisioning Image enthält folgende Daten:

```

├─
├─ .manifest
├─ .revision
├─ ECC-RSA_TSL-test.xml
├─ TrustedTpm.cab
├─ android-roots
├─ '-- roots.json
├─ apple-roots
├─ '-- apple-root.pem
├─ federation-master
├─ '-- federation-master.yaml
├─ federation-master.yaml
├─ policy-engine-bundle-keys
├─ |-- ca
├─ | |-- GEM.KOMP-CA8.pem
├─ | |-- GEM.RCA7.pem
├─ |-- signers
├─ '-- ZETA PIPPAP Policies 03.06.2026 16 31.pem
├─ roots.json

```

```
|-- ti-roots
|   |-- roots.json
|-- trusted-tpm
|   |-- TrustedTpm.cab
|-- tsl
|   |-- ECC_PU_TSL_10322.xml
```

Das Provisioning Image wird als OCI-konformes Image in der oben dargestellten Verzeichnisstruktur bereitgestellt.

Das Provisioning Image enthält im Root-Verzeichnis die Integrität und Dateien `.manifest` und `.revision`.

.manifest: Diese Datei enthält die SHA-256-Prüfsummen aller im Image enthaltenen Dateien im Format:

<SHA-256-Hex-Hash> <Relativer-Pfad>.

.revision: Diese Datei enthält die eindeutige Version des Bereitstellungsstands als Git-Commit-Hash oder als Vertrauensstring.

Hinweis: Die Dateien im Root-Verzeichnis (außer `.manifest` und `.revision`), namentlich:

- `ECC-RSA_TSL-test.xml`
- `TrustedTpm.cab`
- `federation-master.yaml`
- `roots.json`

werden aus Gründen der Abwärtskompatibilität weit der ZETA mitgeführt.

A 29737 -ZETA Guard, Verwendung der Daten aus den Unterverzeichnissen des Provisioning Images

ZETA Guard Microservice Komponenten MÜSSEN die in den Unterverzeichnissen des Provisioning Images strukturierten Dateien verwenden. [≤]

A 29738 -ZETA Guard, Dynamische Dateinamenerkennung in Unterverzeichnissen des innerhalb einer Kubernetes-Infrastruktur Provisioning Images

Die Komponenten des ZETA Guard DÜRFEN NICHT auf fest vorgegebene (hardcodierte) Dateinamen für die Nutzdaten in den Unterverzeichnissen des Provisioning Images vertrauen, da sich diese im Rahmen von Aktualisierungen ändern können. [≤]

Hinweis: Insbesondere bei Dateien mit versions- oder laufzeitabhängigen Namensbestandteilen – wie der Trust Service Status List (TSL) im Verzeichnis `/tsl/` (z. B. `ECC_PU_TSL_10322.xml`) oder den Richtlinien-Signaturen – MÜSSEN die ZETA Guard Komponenten die zu ladenden Dateien dynamisch ermitteln. Dies kann beispielsweise durch das Auslesen des Verzeichnisinhalts (Directory Listing), das Filtern nach Dateiendungen (z. B. `.xml`, `*.pem`, `*.json`) oder die Auswertung der in der `.manifest`-Datei gelisteten Pfade gelöst werden.*

5.16.5.2 Integritätsschutz und Verifikation

Das von der gematik bereitgestellte Provisioning Image ist kryptografisch signiert. Das für die Verifikation genutzte Signaturzertifikat der gematik wird im offiziellen GitHub-Repository unter:

<https://github.com/gematik/zeta/tree/main/zeta-guard-prv-signing-key>

in den umgebungsspezifischen Verzeichnissen (`prod` und `test`) veröffentlicht.

A_29739 -ZETA Guard, Verifikation des Provisioning Images

Der ZETA Guard MUSS vor dem Laden und Verarbeiten der Inhalte des Provisioning Images folgende Prüfschritte erfolgreich durchführen:

- Verifikation der Image-Signatur: Die Signatur des OCI-Images MUSS gegen das für die jeweilige Umgebung (prod/test) gültige Signaturzertifikat der gematik erfolgreich validiert werden.

[<=]

5.16.5.3 Verarbeitung durch ZETA Guard Komponenten**A_29740 -PDP Authorization Server, Verwendung Trust Anchor des Provisioning Images**

Der ZETA Guard Authorization Server MUSS die im Provisioning Image enthaltenen Trust Anchor für folgende Validierungsschritte nutzen:

- TI-Zertifikate (TSL): Die Trust Service Status List (TSL) unter /tsl/ sowie die TI-Roots unter /ti-roots/roots.json MÜSSEN zur Validierung von SMC-B-Zertifikatsketten herangezogen werden.
- Plattform-Attestierung:
 - TPM-Attestierung: Zur Verifikation von hardwarebasierten Attestierungsdaten der Clients MÜSSEN die TPM-Stammzertifikate aus /trusted-tpm/TrustedTpm.cab verwendet werden.
 - Android-Attestierung: Zur Verifikation von Android-Clients MÜSSEN die Trust Anchors aus /android-roots/roots.json verwendet werden.
 - Apple-Attestierung: Zur Verifikation von iOS-/macOS-Clients MÜSSEN die Apple-Wurzelzertifikate aus /apple-roots/apple-root.pem verwendet werden

[<=]

A_29741 -PDP Policy Engine, Verwendung der Signer-Keys des Provisioning Images

Die ZETA Guard Policy Engine MUSS die im Verzeichnis /policy-engine-bundle-keys/ bereitgestellten Schlüssel nutzen, um die Signaturen geladener Policy Bundles zu verifizieren:

Unter /policy-engine-bundle-keys/ca/ abgelegte CA-Zertifikate dienen als Vertrauensanker für die ausstellenden Instanzen.

Unter /policy-engine-bundle-keys/signers/ abgelegte Zertifikate definieren die explizit autorisierten Signaturschlüssel für die Freigabe von OPA-Policies.

[<=]

A_29742 -HTTP Proxy - Verwendung von Trust Anchors und TSL beim ZETA/ASL-Verbindungsaufbau

Der ZETA Guard HTTP-Proxy MUSS für die Etablierung des ZETA/ASL-Kanals die im Provisioning Image bereitgestellten TI-Root-Zertifikate (/ti-roots/roots.json) sowie die Trust Service Status List (/tsl/* .xml) verwenden.

Diese Daten MÜSSEN genutzt werden, um die aktuellen Status- und Sperrinformationen der beteiligten Zertifikate im Zuge des Handshakes zu ermitteln.

[<=]

5.16.5.4 Lifecycle und Update-Regeln

Das Provisioning Image wird durch die gematik neu erstellt und publiziert, sobald sich enthaltene Vertrauensanker oder Konfigurationsdateien ändern.

Bei regulären TSL-Updates erfolgt die Bereitstellung zyklisch (in der Regel alle 14 Tage). Bei ad-hoc Änderungen (z. B. Sperrung oder Austausch von CA-/Signatur Schlüsseln) erfolgt die Bereitstellung unverzüglich.

A_29743 -ZETA Guard, Automatische und unterbrechungsfreie Aktualisierung der Provisioning Daten

Der ZETA Guard MUSS einen Mechanismus implementieren, um das Provisioning Image im laufenden Betrieb zu aktualisieren:

- **Erkennung neuer Versionen:** Der ZETA Guard MUSS zyklisch auf das Vorhandensein eines neuen Images (identifiziert über das Tag latest) in der Registry prüfen.
- **Unterbrechungsfreies Laden (Hot-Reload):** Nach erfolgreicher Signaturprüfung MÜSSEN die neuen Trust Anchor und Konfigurationen unterbrechungsfrei (Zero-Downtime / Hot-Reload) in die aktiven Konfigurationen der betroffenen Komponenten eingespielt werden. Laufende Validierungsprozesse und bestehende Sitzungen DÜRFEN hierbei nicht gestört werden

[<=]

A_29950 -Anbieter TI 2.0 Dienst, Betriebsanforderung beim Root-CA-Wechsel

Der Anbieter des TI 2.0 Dienstes MUSS sicherstellen, dass Provisioning-Updates im Root-CA-Wechselzeitraum ohne Betriebsunterbrechung in die produktiven ZETA Guard Instanzen übernommen werden.

Hinweis: Beim Wechsel auf eine neue TI-Root-CA wird durch die gematik ein Provisioning Image bereitgestellt, das für einen definierten Übergangszeitraum sowohl die bisherige als auch die neue TI-Root-CA sowie die zugehörigen TSL-Informationen enthält. Nach Ende des Übergangszeitraums wird durch die gematik ein aktualisiertes Provisioning Image bereitgestellt, in dem die bisherige TI-Root-CA nicht mehr als gültiger Vertrauensanker enthalten ist. [**<=**]

5.16.6 Laufzeitüberwachung

Um die Integrität und Vertraulichkeit der ZETA Guard Microservices innerhalb einer Kubernetes-Infrastruktur zu gewährleisten, werden verschiedene sich ergänzende Sicherheitsmechanismen eingesetzt. Diese dienen der Härtung der Laufzeitumgebung, der Isolation von Workloads und der Erkennung von Anomalien. Diese werden nachfolgend beschrieben. Die Mechanismen sind in aktuell eingesetzte und geplante Verfahren unterteilt; darüber hinaus werden weitere empfohlene Maßnahmen zur Vertiefung der Cluster-Sicherheit aufgeführt.

5.16.6.1 Aktuell eingesetzte Verfahren

5.16.6.1.1 Pod Security Standards (PSS)

Kubernetes Pod Security Standards definieren drei Sicherheitsstufen für Pods (Privileged, Baseline, Restricted). Im ZETA_Guard-Cluster soll die Stufe Restricted für alle ZETA_Guard_Namespaces per Namespace-Label durchgesetzt werden. Damit werden unter anderem folgende Eigenschaften erzwungen:

- Ausführung als Nicht-Root-Benutzer ()
- Keine privilegierten Container ()
- Keine Fähigkeiten über die Allowlist hinaus (,)
- Nur schreibgeschützte Root-Filesysteme ()
- Ausschluss von Host-Netzwerk, Host-PID und Host-IPC

5.16.6.1.2 Admission Controller

Admission Controller sind Kubernetes-Webhooks, die API-Server-Anfragen vor der persistenten Speicherung validieren oder mutieren. Im ZETA_Guard_Cluster sollen folgende Admission Controller eingesetzt werden:

Validating Admission Webhooks prüfen eingehende Ressourcen gegen Richtlinien und lehnen nicht-konforme Anfragen ab. Typische Prüfungen umfassen:

- Sicherstellung, dass nur Images mit der gematik Signatur produktiv eingesetzt werden. Bei jedem Deployment wird geprüft, ob das Image eine gültige gematik-Signatur trägt.
- Ablehnung von Deployments ohne definierte und

Als Policy-Engine für Admission Control sollen Kyverno oder OPA Gatekeeper eingesetzt werden, um Richtlinien deklarativ als Kubernetes-Custom-Resources zu verwalten.

5.16.6.1.3 Network Policies

Kubernetes Network Policies steuern den zulässigen Netzwerkverkehr zwischen Pods auf Basis von Label-Selektoren, Namespaces und Ports. Im ZETA_Guard_Cluster gilt das Default-Deny-Prinzip: Jeder Namespace verfügt über eine Network Policy, die eingehenden und ausgehenden Verkehr standardmäßig verbietet. Explizite Freigaben erfolgen nur für notwendige Kommunikationsbeziehungen, insbesondere:

- Eingehender Verkehr zum PEP (HTTP Proxy) ausschließlich vom Ingress Controller
- Kommunikation zwischen PEP, PDP Authorization Server und Policy Engine (OPA) nur innerhalb dedizierter ZETA_Guard_Namespaces
- Ausgehender Verkehr zu gematik-Diensten (PIP/PAP Artifact Registry, Telemetrie-Service) nur über definierte Egress-Regeln
- Vollständige Isolation zwischen ZETA_Guard_Namespace und Resource-Server-Namespace auf Netzwerkebene

Hinweis: Native Kubernetes Network Policies sind Layer-3/4-Konstrukte und operieren auf IP-Adressen und Ports. Für eine weitergehende Layer-7-Kontrolle wird auf das Service Mesh (s. u.) verwiesen.

5.16.6.1.4 Service Mesh

Ein Service Mesh (z. B. Istio, Linkerd oder Cilium) ergänzt Kubernetes um transparente mTLS-Verschlüsselung, feingranulare Traffic-Kontrolle und Observability zwischen Microservices. Im ZETA- Das Service Mesh SOLL Kommunikationspfade zwischen ZETA_Guard-Componenten absichern und überwachen. Im ZETA Guard Cluster übernimmt das Service Mesh folgende Aufgaben:

Mutual TLS (mTLS):

Alle Kommunikationsverbindungen zwischen ZETA_Guard-Komponenten werden durch das Service Mesh mit mTLS verschlüsselt und wechselseitig authentifiziert. Zertifikate werden automatisch rotiert.

Autorisierungsrichtlinien:

Mit AuthorizationPolicy-Ressourcen (Istio) oder Server-Ressourcen (Linkerd) wird die dienstspezifische Kommunikation auf Layer 7 eingeschränkt. So darf z. B. nur der PEP den Resource Server aufrufen.

Traffic-Observability:

Das Service Mesh liefert metrische Daten (Latenz, Fehlerrate, Throughput) und Traces für jede Dienstkommunikation, die in das Monitoring einfließen.

5.16.6.1.5 Ingress und Egress / Gateway

Ingress Controller:

Externer eingehender Verkehr zum ZETA Guard wird über einen Ingress Controller (z. B. NGINX oder Envoy-basiert) terminiert. Der Ingress Controller übernimmt TLS-Terminierung, Rate Limiting und ggf. WAF-Funktionen (Web Application Firewall). Es wird ausschließlich HTTPS mit TLS 1.2 oder höher akzeptiert und die Ingress-Kommunikation wird für IPv4 und IPv6 bereitgestellt.

Egress Gateway:

Ausgehender Verkehr aus dem ZETA_Guard_Cluster (z. B. zu PIP/PAP-Registry, Telemetriedaten-Service, Identity Providern) wird über ein dediziertes Egress Gateway geleitet. Durch Egress-Policies auf dem Gateway wird sichergestellt, dass kein unkontrollierter Datenabfluss stattfindet. Zulässige externe Ziele werden in einer Allowlist gepflegt.

Hinweis: Am Anfang wird ZETA Guard nur Network-Policies als Egress Funktion verwenden.

5.16.6.2 Geplante Verfahren

5.16.6.2.1 Pod-Überwachung gemäß Cilium Tetragon

Cilium Tetragon ist ein eBPF-basiertes Security-Observability- und Enforcement-Framework für Kubernetes. Im Unterschied zu netzwerkorientierten Schutzmechanismen operiert Tetragon im Linux-Kernel und ermöglicht **Laufzeit-Sicherheitsüberwachung** auf Systemaufruf-Ebene (syscall-Ebene) innerhalb von Pods und Containern.

Hinweis: Da Tetragon tief in den Linux-Kernel (via eBPF) eingreift und clusterweite Ressourcen erstellt, benötigt das Service-Konto, das die Installation durchführt, cluster-admin-Rechte. Daher wird Tetragon nicht durch ZETA Guard, sondern durch den Anbieter installiert. Für den laufenden Betrieb und das Erstellen von Regeln reicht ein dediziertes RBAC-Profil für die Cilium-CRDs aus.

Im ZETA_Guard_Cluster ist der Einsatz von Cilium Tetragon geplant für:

Process Execution Monitoring:

Jede Prozessausführung innerhalb eines ZETA_Guard_Pods wird erfasst. Unerwartete Prozesse (z. B. Shell-Aufruf in einem Applikations-Container, Ausführung von Netzwerktools) lösen Sicherheitswarnungen aus und können direkt blockiert werden (Enforcement Mode). Dies erkennt insbesondere Post-Exploitation-Aktivitäten nach einer Kompromittierung.

File Integrity Monitoring (FIM):

Schreibzugriffe auf kritische Dateisystempfade innerhalb von Pods (z. B. Konfigurationsdateien, Zertifikate, Binary-Pfade) werden überwacht. Unzulässiger Zugriff auf Dateien wird unterbunden (Enforcement Mode) und gemeldet.

Network Observability auf Systemaufruf-Ebene:

Netzwerkverbindungen werden auf Basis der tatsächlichen Systemaufrufe (, , etc.) beobachtet, nicht nur auf Paketebene. Dies ermöglicht die Attribution jeder Netzwerkverbindung auf den auslösenden Prozess und Container.

TracingPolicy und Enforcement:

Über -Custom-Resources können deklarative Sicherheitsregeln definiert werden, die Tetragon im Kernel durchsetzt. Im Enforcement Mode werden unzulässige Aktionen (z. B. eines nicht-erlaubten Binaries, Verbindung zu einer nicht-erlaubten IP) direkt durch den Kernel blockiert, noch bevor eine Reaktion auf Anwendungsebene erfolgen kann.

5.16.6.2.2 RBAC-Härtung (Role-Based Access Control)

Kubernetes RBAC steuert, welche Subjekte (ServiceAccounts, User, Groups) welche API-Ressourcen lesen oder verändern dürfen. Für den ZETA-Guard-Cluster gilt:

- Jeder ZETA-Guard-Microservice erhält einen **dedizierten ServiceAccount** mit minimalen Berechtigungen (Least Privilege).
- Die Verwendung des default-ServiceAccounts in Pods wird durch eine Admission Policy verboten.
- ClusterAdmin- und ClusterRole-Bindings werden auf das absolute Minimum reduziert und regelmäßig überprüft.
- Der Kubernetes API-Server-Zugriff vom Produktivsystem wird auf dedizierte Operator-Identitäten beschränkt; interaktive Zugriffe erfordern MFA und werden auditiert.

5.17 Policy Enforcement Points

Der Policy Enforcement Point (PEP) stellt die zentrale Sicherheitskomponente einer Zero Trust-Architektur dar, da in dieser alle Zugriffsentscheidungen durchgesetzt (engl.: enforce) werden.

5.17.1 PEP HTTP Proxy

Die Komponente HTTP Proxy ist die "letzte" vor das Resource Backend geschaltete Zero Trust-Komponente und prüft das Access Token im Authorization Header des Requests. Ist das Access Token gültig, wird der Zugriff gewährt. Zudem wird der Request um zusätzliche HTTP-Header angereichert, um ein Tracing zu ermöglichen. Wenn ein PoPP Token im popp Header vorhanden ist, wird es geprüft.

A_26666-01 -PEP HTTP Proxy - TLS Terminierung

Die Komponente PEP HTTP Proxy MUSS TLS für eingehende Verbindungen terminieren können.[<=]

A_26195 -PEP HTTP Proxy - Unterstützung Websocket

Die Komponente HTTP Proxy MUSS das WebSocket-Protokoll unterstützen.[<=]

A_26641 -PEP HTTP Proxy - HTTP Protokoll-Versionen

Die Komponente HTTP Proxy MUSS das HTTP Protokoll in den Versionen HTTP/1.1, HTTP/2 und SOLL HTTP/3 unterstützen.[<=]

A_25667 -PEP HTTP Proxy - Verifikation Access Token Binding

Die Komponente HTTP Proxy MUSS das Access Token Binding über den Mechanismus OAuth 2.0 Demonstrating Proof of Possession (DPoP) gemäß [RFC9449] verifizieren; d. h. der Claim "jkt" im Access Token MUSS eindeutig der Angabe im DPoP-Token entsprechen.[<=]

53288A_30003 -PEP HTTP Proxy, keine Mehrfachnutzung der DPoP Proof Token
Der HTTP Proxy MUSS anhand des claims jti im DPoP Proof durchsetzen, dass schon verwendete DPoP Proofs nicht noch einmal genutzt werden können.[<=]

A_25668-01 -PEP HTTP Proxy - Access Token Validierung

Die Komponente HTTP Proxy MUSS das übergebene Access Token validieren. Insbesondere MÜSSEN

- die Signatur des Authorization Servers gültig,
- die Angaben zur zeitlichen Gültigkeit (Felder:iat, exp) valide,

- die Angabe `Claim` für das Resource Backend korrekt eingetragen und, d.h. exakt übereinstimmend mit der für den aufgerufenen Request-Pfad konfigurierten logischen Audience, und
 - die Angabe `scope` und `aud` passend zur Request url
 - sein, `aud` passend zur Request url
- sein. Die Bindung des Tokens an die konkret aufgerufene URL erfolgt nicht über `aud` sondern über den DPoP-Claim `htu` (siehe A_29676).
Der HTTP Proxy MUSS dabei sowohl Access Tokens mit `ver: 1` (Legacy; `aud` kann ein verbatim übernommener Wert sein) als auch mit `ver: 2` (von der Policy Engine vergebener logischer `aud`) verarbeiten.

Die Signatur des Access Token ist gültig, wenn sie mathematisch gültig ist und die Signatur vom Authorization Server im gleichen ZETA Guard erstellt wurde (default Einstellung) oder von einem Authorization Server, der in der Konfiguration des HTTP Proxy angegeben und im Entity Statement des Federation Master aufgeführt ist. Es MUSS möglich sein, mehrere Authorization Server in die Konfiguration des HTTP Proxy einzutragen.

Wenn das Access Token ungültig ist, dann MUSS der Request mit HTTP Code 401 Unauthorized beantwortet werden. [\leq]

Hinweis: Jeder Authorization Server, der zur Föderation des Federation Masters gehört, veröffentlicht ein eigenes Entity Statement, in dem die Schlüssel enthalten sind, mit denen die Signatur der Access Token geprüft werden kann.

Hinweis: Einige TI 2.0-Dienste benötigen ein PoPP Token. Diese werden im Request Header PoPP übertragen und vom HTTP Proxy geprüft.

61406A_26477-01 -PEP HTTP Proxy - PoPP Token Validierung

Die Komponente HTTP Proxy MUSS so konfiguriert werden können, dass pro Endpunkt des Resource Servers der Request Header PoPP verlangt und das PoPP Token validiert wird. Zusätzlich MUSS die Konfiguration pro Endpunkt unterstützen, dass die Dauer der Gültigkeit des PoPP Token in Sekunden seit Ausstellung und nach Ausstellungszeitpunkt und Prüfzeitpunkt innerhalb des gleichen Quartals eingestellt werden kann. Bei einem Prüfzeitpunkt innerhalb des gleichen Quartals MUSS eine grace-period von 5 Minuten beim Quartalswechsel akzeptiert werden.

Wenn ein Request für einen Endpunkt des Resource Servers empfangen wird, der kein PoPP Header enthält und das Vorhandensein des PoPP Headers verlangt wird, dann MUSS der HTTP Proxy den Request mit HTTP Code 400 Bad Request beantworten.

Insbesondere MUSS die Signatur des PoPP Servers gültig sein. Die Signatur des PoPP Token ist gültig, wenn sie mathematisch gültig ist und die Signatur vom PoPP Server erstellt wurde. Der HTTP Proxy MUSS ein vorhandenes und noch gültiges JWKS des PoPP Servers verwenden oder das JWKS des PoPP Servers herunterladen, um anhand der im JWKS enthaltenen Schlüssel die Signatur des PoPP Token zu prüfen.

Der `claimactorId` des PoPP Token MUSS mit dem Attribut `sub` aus den zum Access Token gehörenden Nutzer-Daten übereinstimmen.

Vor Ablauf der Gültigkeit MUSS der HTTP Proxy ein neues JWKS des PoPP Servers herunterladen.

Wenn die Signatur des PoPP Token ungültig ist oder eine der anderen Prüfungen nicht erfolgreich war, dann MUSS der Request mit HTTP Code 403 Forbidden beantwortet werden.

[\leq]

Hinweis: Wenn ein ZETA/ASL-Kanal am HTTP Proxy terminiert wird, dann wird das PoPP Token durch den ZETA/ASL-Kanal geschützt transportiert.

87874A_28525-012 -PEP HTTP Proxy - Step-up-Bedingung

Die Komponente HTTP Proxy MUSS dem Client ~~eine SStep-up-Bedarf mit HTTP-Statuscode 401 (Unauthorized) und einem WWW-Authenticate-Header gemäß [RFC9470] signalisieren, wenn eine der folgenden, durch eine erneute (Step-up-)Authentisierung behebaren Bedingungen vorliegt:~~

~~das im gültigen Access Token (Claim acr) nachgewiesene Authentifizierung mit HTTP-Statuscode erreicht nicht das für die aufgerufene Route konfigurierte Mindest-acr, ode 401 signalisieren, wenn~~

- ~~• der~~
- ~~• der im gültigen Access Token (Claim scope) enthaltene Scope deckt den für die aufgerufene Route konfigurierten erforderlichen Scope im Access-Token nicht enicht ab, dieser ist aber über eine erneute (Step-up-)Authentisierung erlangbar.~~
- ~~• Der WWW-Authenticate-Header MUSS den zutreffenden Fehlercode enthalten ist oder~~
- ~~• die Audience im Access-Token nicht zur Request URL passt.~~

~~:~~

- ~~• bei unzureichendem Niveau error="insufficient_user_authentication" mit den Parametern acr_values (und ggf. max_age);~~
- ~~• bei unzureichendem Scope error="insufficient_scope" mit dem Parameter scope.~~

~~Sind beide Bedingungen erfüllt, MUSS error="insufficient_user_authentication" verwendet werden (das höhere Niveau schließt die Scope-Erlangung ein).~~

~~Die angefragte URL MUSS existieren (~~unterstützte und einer konfigurierten Route (mit hinterlegter logischer Audience);~~ e, erforderlichem acr und erforderlichem Scope) zugeordnet sein.~~

~~[<=]~~

~~*MainlineHinweis: Ein Nicht-Übereinstimmen des aud-Claims oder des DPoP-htu-Claims ist KEIN Step-up-Fall und wird gemäß A_29676 mit 403 (Forbidden) abgewiesen.*~~

A_26493-01 -PEP HTTP Proxy - Umgang mit JWKS des Popp Servers

Die Komponente HTTP Proxy MUSS, falls der Header Parameter "kid" im PoPP Token JWT nicht zu einem Key im JWKS passt, ein neues JWKS des PoPP Servers herunterladen. [~~<=~~]

A_26480 -PEP HTTP Proxy - Umsetzen eines ZETA/ASL-Kanals

Die Komponente HTTP Proxy MUSS einen ZETA/ASL-Kanal (Server-Seite) umsetzen können. Die Verwendung des ZETA/ASL-Kanals MUSS durch Konfiguration ein- und ausschaltbar sein. In der Default-Einstellung ist der ZETA/ASL-Kanal ausgeschaltet. [~~<=~~]

Hinweis: Ob ein ZETA/ASL Kanal zu verwenden ist, wird in der Spezifikation des TI 2.0-Dienstes festgelegt. Die Anforderungen für den ZETA/ASL-Kanal sind in [gemSpec_Krypt#8] zu finden.

Wenn der ASL Kanal am HTTP Proxy terminiert, dann enthält der äußere Request kein Access und kein DPoP Token. Der HTTP Proxy terminiert den Kanal und prüft das Access und das DPoP Token im inneren Request.

Wenn der Resource Server den ASL Kanal terminiert, dann enthält der äußere Request ein Access Token und ein DPoP Token passend zum ASL Endpunkt am HTTP Proxy. Der HTTP Proxy leitet den Request nach erfolgreicher Token-Prüfung an den Resource Server weiter. Der Resource Server terminiert den ASL Kanal und findet im inneren Request ein Access und ein DPoP Token passend zum Endpunkt des Resource Servers. Es wird empfohlen, dass der Resource Server das Access und das DPoP Token prüft.

A_26492-02 -PEP HTTP Proxy - Weiterleitung von Client-Daten

Die Komponente HTTP Proxy MUSS so konfiguriert werden können, dass pro Endpunkt des Resource Servers die Weiterleitung der Client-Daten durch den HTTP Proxy ein- und ausgeschaltet werden kann. Die default-Einstellung ist keine Weiterleitung der Client-Daten.

[<=]

A_25669-01 -PEP HTTP Proxy - Zusätzliche HTTP-Header

Die Komponente HTTP Proxy MUSS die HTTP Requests mit allen HTTP-Headern an das Resource Backend weiterleiten und dabei die folgenden zusätzlichen HTTP-Header einsetzen.

Tabelle 17: PEP HTTP Proxy - Zusätzliche HTTP-Header

HTTP-Header	Format	Schema	Größe (max. geschätzt)
zeta-user-info	Base64-URL kodierte JSON Struktur des User-Info Inhalts	[zeta-user-info.yaml]	250 Byte
zeta-popp-token-content	Base64-URL kodierte JSON Struktur des PoPP Token Inhalts. Der PoPP Header enthält das PoPP Token. Optional: Wird nur gesetzt, wenn im Request ein PoPP Header enthalten ist.	PoPP Token Payload	450 Byte
zeta-client-data	Base64-URL kodierte JSON Struktur der Client-Daten Optional: Wird nur gesetzt, wenn die Konfiguration des HTTP Proxy für die URL des Requests die Weiterleitung der Client-Daten festlegt.	[client-data.yaml]	250 Byte

Gleichnamige HTTP-Header aus dem ursprünglichen HTTP-Request MÜSSEN entfernt bzw. überschrieben werden.[<=]

Hinweis: Die Schema-Dateien sind in [GitHub ZETA Schemas] festgelegt.

6196A_26560-01 -PEP HTTP Proxy - Weiterleitungskonfiguration

Der PEP HTTP Proxy MUSS es ermö**g**lichen, dass für jede dieser statisch konfigurierten Routen zwingend eine erwartete logische Audience zu hinterlegen. Diese logische Audience entspricht dem aud-Wert, den die Policy Engine für die Weiterleitung von Request-URLs an die URLszugehörige Zielressource vergibt. Das Routing an die internen Resource Server DARF NICHT rein dynamisch auf Basis des ausgelesenen aud-Claims des Access Tokens ohne vordefinierte Pfad-Zuordnung erfolgen.[<=]

A_29675 -Anbieter TI 2.0 Dienst - Konfiguration der internen Endpunkte am PEP

Der Anbieter des TI 2.0 Dienstes MUSS das statische Routing des PEP HTTP Proxys so konfigurieren, dass externe Aufrufe der Clients ausschließlich auf die für den jeweiligen Dienst vorgesehenen, internen Resource Servern Endpunkte des Anbieters geleitet werden (Whitelist-Prinzip). Der Anbieter MUSS dabei für jeden konfigurierten Pfad oder virtuellen Host die exakte

logische Audience festlegen, deren Vorhanden-können sein im aud-Claim des Access Tokens durch den PEP für den Zugriff erzwungen wird (Strict Audience Enforcement). Diese logische Audience MUSS mit dem Wert übereinstimmen, den die Policy Engine des ZETA Guard für die betreffende Zielressource im aud-Claim vergibt. [<=]

66355A_272659676 -PEP HTTP Proxy - unveränderte Sicherheitsprüfungen vor Weiterleitung

Zusätzlich zur Tokenvalidierung MUSS der PEP HTTP Proxy vor der Weiterleitung von Host Header und Request Zeile

Der PEP HTTP Proxy MUSS an einen internen Resource Server die folgenden Übereinstimmungen prüfen:

- Strict Audience-Matching (nur wenn im PEP konfiguriert): Der aud-Claim des Access Tokens MUSS exakt mit der für den aufgerufenen Resource-Pfad konfigurierten logischen Audience übereinstimmen. Der PEP MUSS dabei Tokens mit ver: 1 und ver: 2 unterstützen.
- Strict acr-Matching (nur wenn im PEP konfiguriert): Der acr-Claim des Access Tokens MUSS mindestens dem geforderten Niveau gemäß LoA-Ordnung mit der für den aufgerufenen Resource-Pfad konfigurierten acr entsprechen.
- DPoP Target URI-Abgleich: Die im DPoP-Proof enthaltene Target URI (Claim htu) MUSS den Host Header und die exakt der extern aufgerufenen URL des PEP HTTP Proxys entsprechen.

Sind diese Übereinstimmungen nicht gegeben, MUSS der PEP HTTP Proxy den Request Zeile unverändert mit dem HTTP-Fehlercode 403 Forbidden abweisen und DARF ihn NICHT an den internen Resource Server weiterleiten.

[<=]

1963A_265617265 -PEP HTTP Proxy - Caching unveränderte Weiterleitung von Host Header und Request Zeile

Der PEP HTTP Proxy MUSS so konfiguriert werden können, dass Caching von Inhalten den Host Header und die Request Zeile unverändert an den Response möglich ist source Server weiterleiten. [<=]

A_26589-01 -PEP HTTP Proxy - Nutzer-Daten

Die Komponente HTTP Proxy MUSS für jeden Request mit gültigem Access Token die Nutzer-Daten aus dem Access Token als neuen HTTP Header zeta-user-info in den Request eintragen, bevor der Request an den Resource Server weitergeleitet wird. Folgende claims das Access Token gehören zu den Nutzer-Daten und werden gemäß Tabelle in den zeta-user-info Header übernommen.

Tabelle 18: zeta-user-info-Header

Access Token claim	zeta-user-info Header claim
identifizier	identifizier
profession_oid	professionOID
common_name	commonName
organization_name	organizationName

[<=]

Beispiel für den zeta-user-info Header:

```
{"commonName":"Arztpraxis Walter","identifizier":"1-234567890123","organizationName":"Arztpraxis Walter","professionOID":"1.2.276.0.76.4.50"}
```

A_26590-02 -PEP HTTP Proxy - Client-Daten

Wenn die Request-Weiterleitung mit Client-Daten konfiguriert wurde und der Request ein gültiges Access Token hat, dann MUSS die Komponente HTTP Proxy die Client-Daten aus dem Access Token als neuen HTTP Header `zeta-client-data` in den Request eintragen, bevor der Request an den Resource Server weitergeleitet wird.

Folgende claims das Access Token gehören zu den Client-Daten:

- `client_id`
- `product_id`
- `product_version`

[<=]

A_26974-01 -PEP HTTP Proxy - Fehler vom Resource Server

Die Komponente HTTP Proxy MUSS die Response vom Resource Server als Fehler des HTTP Proxy werten, wenn der Resource Server den Response Header `zeta-cause: Proxy` gesetzt hat (der Resource Server hat einen Fehler im Request festgestellt und vermutet die Ursache beim HTTP Proxy) und DARF diese Response nicht an den Client weiterleiten. Der entsprechende Request des Clients MUSS in diesem Fall mit HTTP 500 beantwortet werden.[<=]

A_27266 -PEP HTTP Proxy - Protected Resource Metadata Well-known

Die Komponente HTTP Proxy MUSS gemäß [RFC9728] und [opr-well-known.yaml] ein Well-known JSON Dokument wie folgt bereitstellen, damit Clients die notwendigen Informationen zur Interaktion mit dem Resource Server finden können.

GET `/.well-known/oauth-protected-resource`

Host: `<FQDN des Resource Servers>`

Das Well-known JSON Dokument MUSS mit dem Schema [opr-well-known.yaml] validiert werden können.

[<=]

Hinweis: Es ist geplant, die Protected Resource Metadata Well-known JSON Dokumente in einer folgenden ZETA Ausbaustufe durch den Federation Master bereitzustellen. Der Federation Master signiert die Protected Resource Metadata Dokumente und stellt so sicher, dass alle Services zur Föderation und zur gleichen Umgebung gehören (dev, prod, ref, etc.). Dadurch verbessert sich für ZETA Clients die Authentizität der TI 2.0-Services.

A_28439 -PEP HTTP Proxy - Unterstützung Forwarded-Header

Die Komponente PEP HTTP Proxy MUSS in jeder empfangene HTTP-Anfrage den Forwarded-Header gemäß [RFC 7239] in der weitergeleiteten Anfragen aktualisieren.[<=]

Sicherheits- und Datenschutz-Anforderungen an den A_29850 -PEP HTTP Proxy, Kennzeichnung von PEP-Fehlern im HTTP-Header

Tritt im PEP HTTP Proxy ein Fehler auf, der zu den HTTP-Statuscodes 401 oder 403 führt, MUSS der HTTP Proxy der HTTP-Antwort an den ZETA-Client den folgenden HTTP-Header hinzufügen:

Header-Name: `zeta-error-origin`

Header-Wert: pep[<=]

A_29851 -PEP HTTP Proxy, Ausschluss von Nicht-PEP-Fehlern

Fehler, die vom nachgelagerten Resource Server (Fachdienst) generiert und durch das PEP lediglich durchgereicht werden, sowie PEP-Fehler mit anderen HTTP-Statuscodes (wie z. B. 404), DÜRFEN den Header `zeta-error-origin` NICHT enthalten.[<=]

A_29860 -PEP HTTP Proxy, Erzeugung der Step-Up-Fehlermeldung

Stellt der PEP HTTP Proxy fest, dass für einen Request ein höheres Authentifizierungsniveau erforderlich ist, als das im gültigen Access Token des ZETA-Clients hinterlegte Niveau aufweist (Step-Up-Bedarf), MUSS er den Request abweisen und eine Fehlerantwort mit folgenden Parametern erzeugen:

- HTTP-Statuscode: 401 Unauthorized
- HTTP-Response-Header: WWW-Authenticate mit error=insufficient user authentication, acr values, ggf. max age (gemäß [RFC9470]);zeta-error-origin: pep
- Response-Body (Content-Type: application/json): Ein JSON-Objekt entsprechend dem Schema [zeta-error.yaml] mit folgendem Inhalt:
 - error: MUSS auf den Wert insufficient user authentication gesetzt werden.
 - error_description: MUSS eine für Entwickler verständliche Beschreibung des Fehlers enthalten (z. B. eine Erklärung, dass das aktuelle Sicherheitsniveau unzureichend ist).
 - error_uri: KANN eine URL enthalten, die zusätzliche Dokumentation zur Fehlerbehebung bereitstellt.

[<=]

5.17.2 Sicherheits- und Datenschutz-Anforderungen an den PEP**A_25445 -PEP - Zugriffsentscheidung nur über PDP**

Der PEP MUSS sicherstellen, dass Zugriffe auf den Resource Server nur durch eine positive Zugriffsentscheidung vom PDP möglich sind.[<=]

Hinweis: Die positive Zugriffsentscheidung auf den Resource Server ist gegeben, wenn der Client im Request Authorization Header ein gültiges Access Token mit passendem scope - ausgestellt von einem Authorization Server, zu dem eine Vertrauensbeziehung besteht - vorweisen kann. Durch das gültige Access Token ist sichergestellt, dass der Zugriff von dem PDP freigegeben wird.

5.18 Policy Decision Point

Der Policy Decision Point (PDP) setzt sich aus den Komponenten

- Authorization Server
- Policy Engine und
- PDP Datenbank

zusammen.

5.18.1 Policy Engine

Der PDP implementiert die Policy Engine als [Open Policy Agent] (OPA). Die Policies und die zugehörigen Daten erhält die Policy Engine per OCI Protokoll von der ZETA Artifact Registry. Aus den Input-Daten vom Authorization Server, den Daten vom PIP und den Policies vom PAP ermittelt die Policy Engine eine Entscheidung und gibt diese zurück an den Authorization Server.

Neben der OPA Instanz, die die Entscheidung für den Authorization Server trifft (aktive Instanz), ob eine Kommunikation zulässig ist, implementiert die Policy Engine noch eine zweite OPA Instanz, die mit einem zweiten OPA Bundle vom PIP und PAP Service arbeitet, aber die getroffenen Entscheidungen nicht an den Authorization Server zurückgibt. Diese Instanz wird Simulations-Instanz genannt und dient dazu in produktiven Umgebungen die weiterentwickelte Policies und Daten zu evaluieren.

A_25739-02 -PDP Policy Engine, OPA Instanzen

Der PDP MUSS als Policy Engine zwei Open Policy Agent (OPA) Instanzen bereitstellen, wobei eine Instanz die Entscheidung für den Authorization Server trifft (aktive Instanz), und eine Instanz eine Entscheidung trifft, diese aber nicht an den Authorization Server sendet (Simulations-Instanz).

Die OPA Instanzen MÜSSEN so konfiguriert sein, dass nur signierte OPA Bundles mit gematik Signatur akzeptiert werden. Das Polling Intervall zur Aktualisierung des OPA Bundles über die lokale Artifact Registry MUSS 60 Sekunden sein. [≤]

A_27401 -PDP Policy Engine - Decision Eigenschaften

Die PDP Policy Engine MUSS Decision-Anfragen mit einem JSON Objekt nach dem Schema [pdp-decision.yaml] beantworten. [≤]

A_25490-03 -PDP - Sicherheitsmeldung bei Änderungen und Aktualisierung

Der PDP MUSS sicherstellen, dass bei Aktualisierung und Änderungen der Policies oder PIP-Daten eine Sicherheitsmeldung inklusive der OPA Bundle revision an das Security Monitoring automatisiert übermittelt wird. [≤]

PA_29699 -PDP Authorization Policy Engine, Verwendung der korrekten Policies und Daten

Der Anbieter des TI 2.0 Dienstes MUSS die PDP Policy Engine so konfigurieren, dass das korrekte, dem TI2-0 Dienst zugewiesene OPA Bundle Image aus der ZETA Artifact Registry geladen wird. [≤]

Hinweis: Die Policy Engine muss das OPA Bundle Image nicht direkt aus der ZETA Artifact Registry laden. Ein lokaler Artifact Registry Cache kann verwendet werden.

5.18.2 PDP Authorization Server

A_25760 -PDP Authorization Server - OAuth2 Schnittstellen

Der PDP Authorization Server MUSS eine OAuth2 Schnittstelle gemäß [RFC6749] und [RFC7636] implementieren.

Der Authorization Server MUSS am Token Endpunkt REFRESH_TOKEN entsprechend [RFC6749] ausstellen können. [≤]

A_26669-01 -PDP Authorization Server - TLS Terminierung

Die Komponente PDP Authorization Server MUSS eingehende TLS Verbindungen von außerhalb des ZETA Guards terminieren können. [≤]

A_25659 -PDP Authorization Server - Check Client-Registrierung

Der PDP Authorization Server MUSS die Client Instanzen über den Mechanismus JSON Web Token Client Authentication gemäß [RFC7523] mit DPoP gemäß [RFC9449] authentifizieren und Anfragen, die den Mechanismus nicht verwenden, ablehnen. [≤]

A_25660 -PDP Authorization Server - Session Management mittels Access Token und Refresh Token

Die Komponente Authorization Server MUSS ein Session Management mittels OAuth2 und Ausgabe, Verwaltung und Entzug von Access und Refresh Token gemäß [RFC6749#1.5] unterstützen. [≤]

88744A_27867-019854 -PDP Authorization Server - Aktive Session-Management in der P-Termination

Die Komponente Authorization Server MUSS eine aktive Session-Termination anhand einer Session-Id unterstützen.

Für die Session-Termination MÜSSEN die Eingaben Trace-Id, Session-Id, Reason-Code und Trigger-Source verarbeitet werden. [\leq]

A_29855 -PDP DatenbankAuthorization Server - Entzug der Refresh Token nach Session-Termination

Die Komponente Authorization Server SOLL die Session-Daten gemäß [sMUSS bei erfolgreicher Session-Termination alle der Session zugeordneten Refresh Token unverzüglich ungültig machen. Die Ungültigmachung MUSS auch rotierte Refresh Token derselben Session umfassen. [\leq]

A_29856 -PDP Authorization Server - Verhalten bei Token Requests nach Session-Termination

Die Komponente Authorization Server MUSS vor der Ausstellung eines neuen Token-Sets bei Token Requests mit grant_type=refresh_token prüfen, ob die referenzierte Session terminiert wurde. Ist die Session terminiert, MUSS der Request mit einem standardisierten Fehler gemäß [zeta-Kapitel 5.4.6 abgelehnt werden. [\leq]

A_29857 -PDP Authorization Server - Protokollierung der Session-Termination

Die Komponente Authorization Server MUSS jede Session-Termination revisionsicher protokollieren.

Das Protokoll MUSS mindestens Trace-Id, Session-Id, Trigger-Source, Reason-Code und Zeitpunkt enthalten. [\leq]

A_26944 -PDP Authorization Server - Access Token Inhalt

Die Komponente Authorization Server MUSS Access Token mit Attributen gemäß [access-token.yaml] ausstellen. [\leq]

53294A_25661-01 -PDP Authorization Server - Umsetzung der Policy Decision

Die Komponente Authorization Server MUSS die positive Zugriffs-Entscheidung eines PDP mittels der Ausgabestellung eines Access- und eines Refresh-Token (200 OK) umsetzen bzw. eine Zugriffsverweigerung. Fehler- und Ablehnungsfälle MUSS der Authorization Server gemäß [RFC6749]#section-5.2 (für Token Exchange [RFC8693] und Refresh) mit einem dem zutreffenden HTTP-Statuscode 403 quittieren.

Die Claims im Access und im error-Code beantworten, wobei der Response-Body dem Schema [zeta-error.yaml] entspricht:

- 400 bei fehlerhaftem Request oder ungültigem/abgelaufenem/wiederverwendetem Grant (invalid_request, invalid_grant, invalid_scope, invalid_target) sowie DPoP-Fehlern (invalid_dpop_proof, use_dpop_nonce gemäß [RFC9449]);
- 401 bei fehlgeschlagener Client-Authentifizierung (invalid_client) sowie bei Step-up-Bedarf (insufficient_user_authentication gemäß [RFC9470], siehe A_28525-02);
- 403 ausschließlich bei einer negativen Policy-Entscheidung trotz technisch gültiger Credentials (access_denied) sowie bei session_terminated / Refresh-Token-MÜSS_revoked.

Die Claims im Access TokEN MUSS zur Entscheidung der Policy Engine fpassen. [\leq]

Hinweis: RFC 6749 §5.2 definiert für die angefragten Token-Endpoint keinen Fehlercode für eine Policy-basierte Ablehnung bei gültigen Claims; ZETA verwendet hierfür bewusst 403 access_denied, um technische Grant-Fehler (400/401) von der autorisierungsbezogenen Ablehnung zu unterscheiden.

A_28837 -PDP Authorization Server, Policy Prüfung bei Ausstellung Token-Set

Die Komponente Authorization Server MUSS vor der Ausstellung des neuen Token-Sets (Access Token und Refresh Token) eine Autorisierungsanfrage an die Policy Engine stellen. [≤]

82690A_28527-01 -PDP Authorization Server - Gültigkeitsdauer Access und Refresh Token

Die Komponente Authorization Server MUSS beim Ausstellen von Access und Refresh Token die Gültigkeitsdauer aus der Policy Decision übernehmen. [≤]

Hinweis: Die Gültigkeitsdauer des Refresh Token wird aktuell fest konfiguriert und nicht über die Policy Engine Decision gesetzt, da der verwendete Authorization Server diese Funktion noch neingehalten werden.

- Access Token: maximal 3600 Sekunden
- Refresh Token (Versicht unterstützt, erte, mobile Clients): maximal 1 Tag

[≤]

A_25662 -PDP Authorization Server - Refresh Token Rotation

Die Komponente Authorization Server MUSS eine Refresh Token Rotation gemäß [RFC6749#10.4] erzwingen und MUSS sicherstellen, dass ein Refresh Token nur einmal gegen ein Access Token und ein Refresh Token getauscht werden kann.

Die Komponente Authorization Server MUSS erzwingen, dass der Nutzer eine Authentisierung durchführen muss, wenn seit der letzten Authentisierung die Zeit der Gültigkeitsdauer des Refresh Token abgelaufen ist. [≤]

A_25663 -PDP Authorization Server - Token-Binding an Client-Registrierung

Die Komponente Authorization Server MUSS auszugebende Access Token und Refresh Token über den Mechanismus OAuth 2.0 Demonstrating Proof of Possession (DPoP) gemäß [RFC9449] an die registrierte Client-Instanz binden, indem im Token-Binding-Claim die Angabe der Clientidentifikation als "jkt" eindeutig referenziert wird. [≤]

53305A_25665-01 -PDP Authorization Server - Plugin-Schnittstelle Application Authorization Backend

Die Komponente Authorization Server MUSS eine Plug-In Schnittstelle (per Webhook) zu einem anwendungsspezifischen Authorization Backend implementieren und dabei die folgenden Signale und Informationen aus der erhaltenen Zugriffsanfrage weiterreichen.

Tabelle 19: PDP Authorization Server - Plugin-Schnittstelle Application Authorization Backend

Operation	Operation Kennung	Input	Output
Benachrichtigung über die Ablehnung des Zugriffs durch PDP	notifyAccessDenied	Trace-Id Subject-Information Client-Information PDP-Decision	-
Anwendungsspezifische Autorisierung	authorizeAccess	Trace-Id Session-Id Authorization-Scopes Authorization-Details Subject-Information	Zugriff erlauben Ja/Nein Zusätzliche Authorization Scopes Zusätzliche Authorization Details

		Client-Information PDP-Decision	Zusätzliche Claims
Benachrichtigung über abgelaufene oder terminierte Sessions	notifySessionTermination	Trace-Id Session-Id	-
<u>Aktive Beendigung einer Session durch Anwendung</u>	<u>terminateSession</u>	<u>Trace-Id, Session-Id, Reason-Code, Trigger-Source</u>	<u>Session-Termination akzeptiert Ja/Nein, Fehlercode</u>

[<=]

62412A_265869858 -PDP Authorization Server - Idempotenz der Session- und Nutzer-Termination

Die Operation terminateSession MUSS idempotent ausgeführt werden. Wiederholte Aufrufe mit derselben Session-Id DÜRFEN keinen fachlichen Fehler verursachen.

[<=]

A_26586-01 -PDP Authorization Server - Session-Daten

Die Komponente Authorization Server MUSS nach jeder vollständigen und erfolgreichen Authentifizierung Session-Daten in der Datenbank des PDP speichern.

Die Zu den Session-Daten gehören die Client-, User, und NutzerRequest-Daten gemäß [policy-engine-input.yaml].

Die Session-Daten MÜSSEN bei Anfragen des Authorization-Servers an die Policy Engine im Request mit übergeben werden.[<=]

88101A_26972-023 -PDP Authorization Server - Nutzer-Daten aus der SM(C)-B

Die Komponente Authorization Server MUSS im Falle der SM(C)-B Authentifizierung die folgenden Daten aus dem SM(C)-B Zertifikat auslesen und als Nutzer-Daten gemäß [zeta-user-info.yaml] in der PDP Datenbank speichern:

Tabelle 20: SM(C)-B_Nutzer-Daten

SM(C)-B Daten (C.HCI.AUT gemäß [gemSpec_PKI])	Nutzer-Daten gemäß [zeta-user-info.yaml]	Beschreibung
Extension Admission,registrationNumber	identifizier	Telematik-ID Eindeutige ID der Organisation des Gesundheitswesens
subject,commonName	common_Name	Wird aus dem Zertifikat übernommen. „Kurzname“ der Institution, so wie sie sich auf dem Anschriftenfeld findet.
Extension Admission,professionOID	profession_OID	OID der Institution gemäß [gemSpec_OID#GS-A_4443-*]
subject,organizationName	organization_Nam e	Wird übernommen, wenn im Zertifikat vorhanden. Name der Organisation/Einrichtung des

		Gesundheitswesens
--	--	-------------------

[<=]

88745A_26973-012 -PDP Authorization Server - Nutzer-Daten aus id_token

Die Komponente Authorization Server MUSS im Falle der OIDC Authentifizierung für Versicherte alle Claims aus dem id_token gemäß [gemSpec_IDP_Sek] auslesen und als Nutzer-Daten gemäß [zeta-user-info.yaml] in der PDP Datenbank speichern. Dabei gilt das folgende Mapping für die Pflicht-Nutzer-Daten.

Tabelle 21: id_token_Nutzer-Daten

id_token claims (gemäß [gemSpec_IDP_Sek])	Nutzer-Daten gemäß [zeta-user-info.yaml]	Beschreibung
urn:telematik:claims:id	identifizier	für Versicherte der unveränderliche Anteil der KVNR
urn:telematik:claims:profession	profession_OID	OID für Versicherte
urn:telematik:claims:organization	organization_name	ID oder Name der attributsbestätigenden Stelle (IK-Nummer der Kasse)
urn:telematik:claims:id	common_Name	für Versicherte der unveränderliche Anteil der KVNR

[<=]

A_28144 -PDP Authorisation Server - Länge der Nonce

Der ZETA Guard MUSS am Endpunkt GET /nonce einen 128 Bit langen base64url kodierten Zufallswert ausgeben. [<=]

A_28440 -PDP Authorization Server - Auswertung Forwarded-Header

Die Komponenten PDP Authorization Server MUSS HTTP-Anfragen mit einem vorhandenen Forwarded-Header auswerten, um die Client-IP Adresse zu ermitteln. Bei der Auswertung ist die Semantik gemäß [RFC 7239] und die Reihenfolge der Parameter zu beachten. [<=]

A_25644-01 -PDP Client-Registrierung mit Attestation

Die Komponente Authorization Server MUSS die Clients bei der Registrierung über folgende Mechanismen attestieren:

- Android Key and ID Attestation (für Google-Android Clients)
- Apple DCAAppAttest
- TPM Attestation für Windows und Linux Clients
- Software Attestation

[<=]

A_25645-01 -PDP Authentifizierung mittels TI-Smartcard

Die Komponente Authorization Server MUSS die Authentifizierung per Token Exchange eines Subject Token gemäß [subject-token-smb.yaml] mit Signatur einer SM(C)-B unterstützen. [<=]

A_25649 -PDP Authorization Server - Regelmäßige Wiederholung der Attestation

Die Komponente Authorization Server SOLL die Client-Attestierung für jede neue Session verlangen. [<=]

Hinweis: Eine Session des Authorization Servers ist gültig vom Zeitpunkt t_1 = (Ausstellungszeitpunkt des ersten Refresh Token nach vollständiger Authentifizierung) bis zum Zeitpunkt $t_2 = t_1 +$ (Gültigkeitsdauer des ersten Refresh Token). Bei mobilen Clients werden für die Attestation Services des mobilen Betriebssystems verwendet, die ein Rate Limit haben können. In diesem Fall kann die Attestation eventuell nicht für jede neue Session wiederholt werden oder es werden bei Folge-Attestations nur lokale Verfahren angewendet.

A_25650 -PDP Client-Registrierung - TI-Identität in Attestation

Die Komponente Authorization Server MUSS den registrierten Client, wenn möglich, an eine TI-Identität (mit KVNR oder TelematikID binden. [\leq]

Hinweis: In [ZETA-Stufe 2 iKapitel 5.3.3- Authentifizierung ohne Nutzer-Identität](#) ist eine Autorisierung vorgesehen, bei der keine Nutzer-Identität durch ZETA Guard festgestellt wird ([eGK in Fernversorgung](#)). In diesem Fall erfolgt eine Client-Registrierung ohne Bindung an eine TI-Identität.

A_25651 -PDP Client-Registrierung - Offband Nutzer Verification

Die Komponente Authorization Server MUSS einen Offband Prozess (per E-Mail) für die Kommunikation mit diesem Nutzer unterstützen (Trust on First Use), wobei der Nutzer seine E-Mail Adresse eigenverantwortlich vergibt. [\leq]

Hinweis: TOFU wird nur für mobile Clients genutzt.

~~53894A_25752-01 -PDP Client-Registrierung Policy Engine - Nutzer Client über Hintergrund zuder Ablehnung der Client-Registrierung informieren~~

Falls ein Client ~~ab~~ [beim Token Request](#) nicht die geforderten Parameter der ~~Client~~ Policy unterstützt bzw. das geforderte Niveau nicht erreicht, MUSS ~~die Komponente er~~ Authorization Server ~~den Nutzer nutzerfreundlich darüber informieren, welche Clienteigenschaften zu der Ablehnung geführt habe von der Policy Engine in der Decision-Response aufgeführten Gründe (Attribut reasons) in der Response an den Client übergeben.~~ [\leq]

A_25738 -PDP Client-Registrierung - Telemetrie

Die Komponente Authorization Server MUSS in den Telemetriedaten zu jeder versuchten Client-Registrierung folgende Parameter ohne einen Nutzerbezug protokollieren:

- Clientparameter (Betriebssystem(-version), Patchlevel, Geolocation etc.) gemäß Clientattestierung
- verwendeter Faktor für Offband-Verifikation (E-Mail)
- Zeitstempel Registrierung, Zeitpunkt Offband-Bestätigung
- verwendeter Faktor der Nutzerauthentifizierung (SmartCard, Digitale Identität)
- Status/Ergebnis des Registrierungsversuchs

[\leq]

~~53898A_25754 -PDP Client-Registrierung - Notfall-Recovery-Prozess für Nutzer~~

~~Die Komponente Authorization Server MUSS dem Nutzer einen Notfall-Recovery Prozess anbieten, falls der Nutzer sein letztes Gerät verloren und keinen Zugriff mehr auf seine registrierte E-Mail-Adresse hat.~~ [\leq]

Mainline_OPB1/ML-188851A-A_26585-012 -PDP Client-Registrierung - Client-Daten

Die Komponente Authorization Server MUSS die Client-Daten gemäß [[epolicy-engine-client-statementdata](#).yaml] in der PDP Datenbank verwalten.

Die Client-Daten MÜSSEN bei Anfragen des Authorization-Servers an die Policy Engine im Request mit übergeben werden. [\leq]

5.18.2.1 PDP Relying Party

A_25655 -PDP - Relying Party

Der PDP Authorization Server MUSS in der TI-Föderation als Relying Party registriert sein. [\leq]

A_25656 -PDP - Entity Statement

Der PDP Authorization Server MUSS die Redirect-URLs aller zulässigen Clients als erlaubte Redirect-URLs im Entity Statement ausweisen. [\leq]

A_25657 -PDP - Authentication über sektoralen IDP

Der PDP Authorization Server MUSS die Nutzer über sektorale IDPs authentifizieren können. [\leq]

5.18.2.2 Ablauf für den Zugriff auf ein Claims amr und acr bei Token Resource Server

5.18.2.3 Um auf einen durch ZETA-Guard geschützten Resource Sequest

~~A_29820 -Authorization Server zugreifen zu können, müssen abhängig vom Zusta, Attribute amr und des ZETA-Clients verschiedene Teilabläufe ausgeführt werden, oder könnenacr bei Token überspruExchangen werden. Die ZETA-API besteht aus mehreren Endpunkten, die verschiedene Funktionen bereitstellen. Diese Endpunkte sind in verschiedene Unter-Abläufe aufgeteilt:~~

~~Konfigur mit SMC-B signiertem Subject-Token~~

- ~~• **Der PDP Authorization und Discovery:** DServer ZETA-Client muss die Konfiguration des ZETA-Guards erbeim Token Exchange mitteln, um die richtigen Endpunkte zu erreichen.~~
- ~~• **Client-Registrierung:** Jeder ZETA-Client muss sich einmalig beim ZETA-Guard registrieren, um eine `client_id` zu erhalten und seinen öffent SMC-B signiertem Subject-Token im Request an die Polichen Schlüssel (PuK.Client.Sig) zu hinterlegen.~~
- ~~• **Authentifizierung und Autorisierung:** Der Client muss sich authentifizieren und die Integrität seiner Plattform nachweisen. Zusätzlich muss sich der Nutzer oder beim Primärsystemy Engine gemäß [policy-engine-input.yaml] die Organisation authentifizieren, um ein Access Token für den Zugriff auf geschützte Ressourcen zu erhalten.~~

~~Der Gesamtprozess beginnt damit, dass ein **Nutzer** auf einen Endpunkt eines Resource-Servers zugreifen möchte. Dieser Zugriff wird über das Primärsystem vom **ZETA-Client** im Auftrag des Nutzers ausgeführt; siehe folgende Abbildung.~~

~~Attributeamr und acr wie folgt verwenden:~~

~~amr: urn:telematik:auth:sc~~

~~acr: gematik-ehealth-loa-substantial~~

~~[\leq]~~

~~76008A_27797 -ZETA, Ablauf Zugriff auf geschützte Ressource~~

~~Der ZETA-Guard und der ZETA-Client MÜSSEN den Ablauf gemäß Abbildung-Abb_ZETA_Zugriff_auf_geschützte_Ressource unterstützen.~~

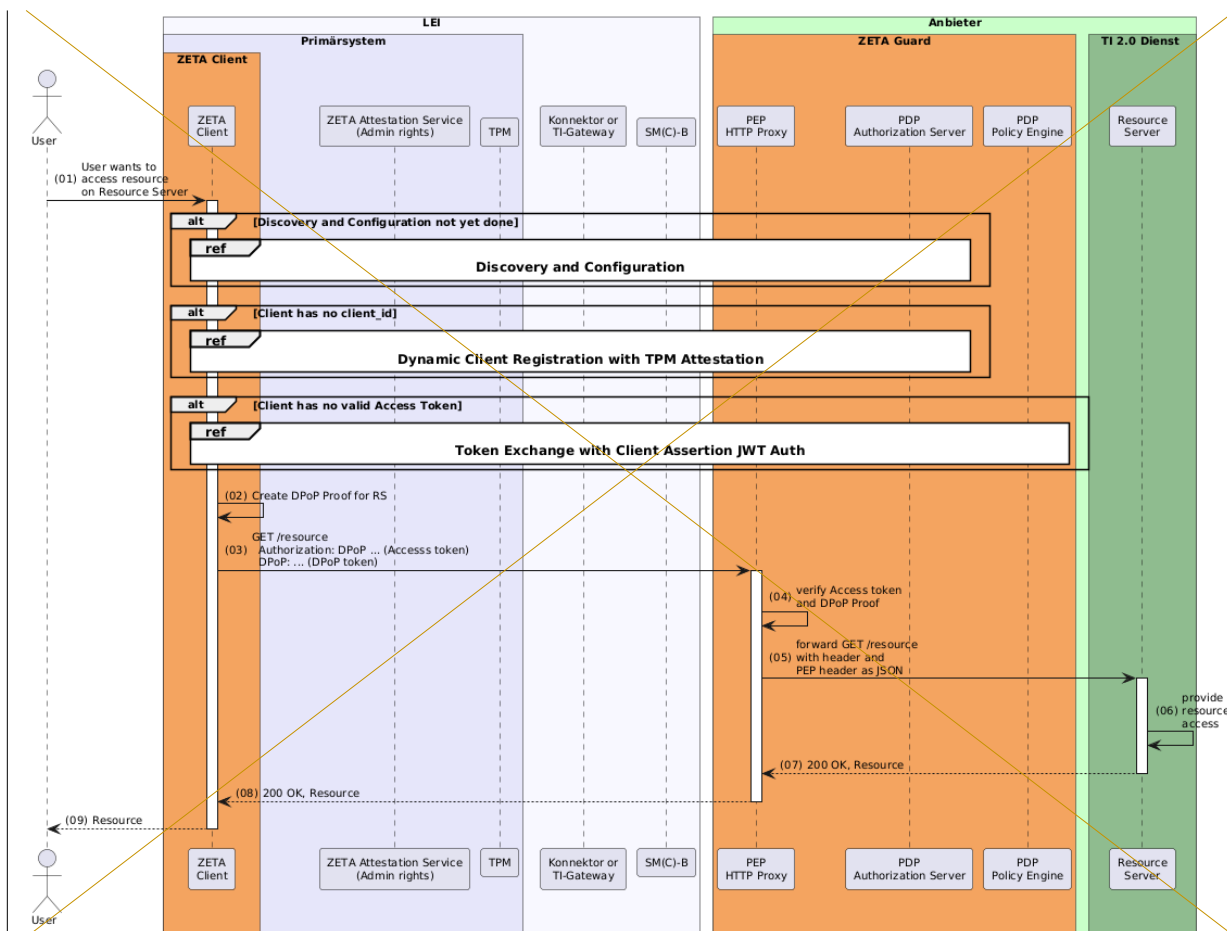


Abbildung9994 -Authorization 29: Abb_ZETA_Zugriff_auf_geschützte_Ressource

[<=>]

5.18.2.4 Service Discovery

Der ZETA-Guard ermöglicht ZETA Clients die Ermittlung der bereitgestellten-Endpunkerver, Attribute durch Abfrage von Well-known JSON-Dokumenten.

Mainline_OPB1/ML-176009A_27798 –ZETA Guard, Service Discovery

Der ZETA-Guard MUSS die Well-known JSON-Dokumamr und acr bei OIDC Authente für die Service Discovery gemäß Abbildifizierung Abb_Service_Discovery bereitstellen. Das Protected-Resource Well-known JSON-Dokument MUSS mit dem Schema [opr-well-known.yaml] validiert werden können.

Das Amit Sek IDP

Der Authorization Server Well-known JSON-Dokument MMUSS mitbei dem Schema [as-well-known.yaml] validrOIDC Authentifiziert werden können.

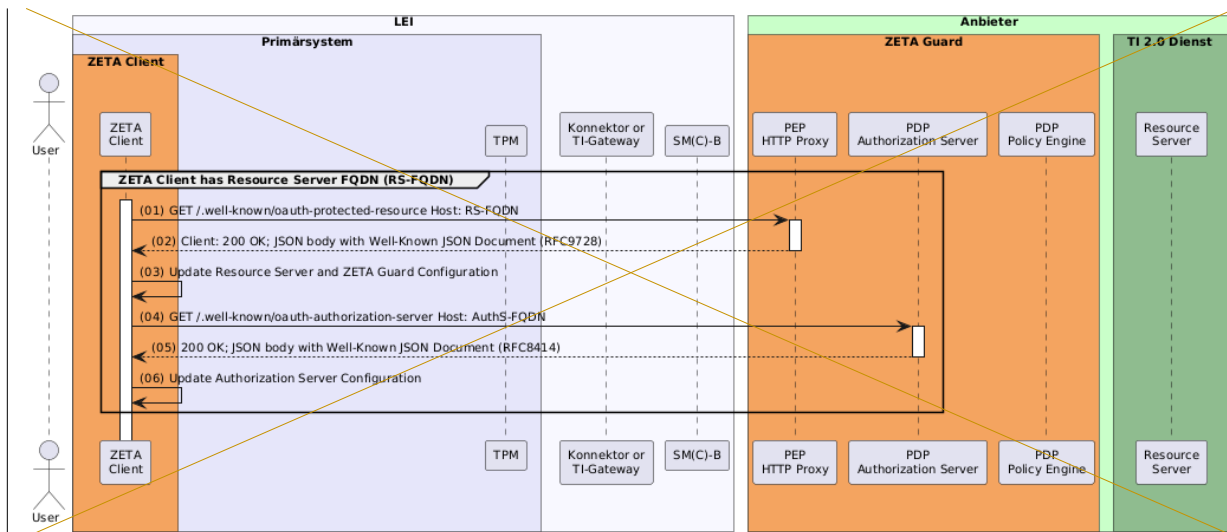


Abbildung 30: Abb_Service_Discovery

[<=]

88102A_28420-01-ZETA Guard, Service Discovery -- Bereitstellung etag Header

Der ZETA Guard MUSS für jedes Well-known JSON-Dokument einen eindeutigen Attribute amr und acr aus dem ID Token etag-Header generieren und diesen gemäß [RFC 9110] im Request an der HTTP Antwort bei jeder Anfrage bereitstellen. Bei jeder inhaltlichen Policy Engine gemäß [policy-engine-input.yaml] verwenden. [=>]

88102A_28421-01-ZETA Guard, A_30004 -Authorization Service Discovery -- Unterstützung if-none-match-Header

Der ZETA Guard MUSS Client-Anfragen, die den if-none-match Header enthalten, korrekt verarbeiten, Attribute amr und acr berechnen. Die nachfolgende Fallunterscheidung ist zu beachten:

- Keine Änderung Authentifizierung des Well-known JSON-Dokuments:

Wenn der vom ZETA Client im if-none-match Header gesendete ETag mit dem aktuellen ETag des Dokuments auf dem ohne Nutzer-Identität

Der Authorization Server übereinstimmt, MUSS bei der ZETA Guard mit einem HTTP-Statuscode 304 (Not Modified) antworten und darf keine Nachricht Authentifizierung ohne Nutzer-Identität senden.

- Änderung des Well-known JSON-Dokuments:

Wenn der vom ZETA Client im if-none-match Header gesendete ETag nicht mit dem aktuellen ETag des Dokuments auf dem Server übereinstimmt (oder wenn der Header fehlt), MUSS der ZETA Guard mit einem HTTP-Statuscode 200 OK antworten und Attribute amr berechnen.

acr: gematik-ehealth-loa-low

das vollständige, aktuelle Well-known JSON-Dokument im Nachrichtentext sowie den aktuellen ETag-Header Attribute amr wird nicht verwendet.

t. [=>]

80784A_28422-ZETA Guard, 9993 -Authorization Service Discovery -- Cache-Control-Header Direktiver, Attribute acr im Access Token

Der ZETA-Guard MUSS im HTTP-Response-Header `PDP Authorization Server` zur Anfrage der Well-Known- und JWKS-JSON-Dokumente einen Cache-Control-Header einfügen, der die Direktiven `max-age` und `public` setzt.

Die Dauer der Direktive `max-age` MUSS konfigurierbar sein (Default: 86400 s). [\leq]

Die Verwendung der Kombination `max-age` und `public` MUSS das nachgewiesene Authentifizierungsniveau als Claim `acr` (Wertebereich gemäß `gematik`-Direktiven `public` und `max-age` im Cache-Control-Header) sorgt für effizientes Caching mit garantierter Aktualität (LoA) in das Access Token übernehmen, damit der Inhalt der Inhalte. In Kapitel 5.16.7 Betriebliche Schnittstellendefinition sind die Well-known- und JWKS-Endpunkte gelistet.

5.18.2.5 Client-Registrierung für stationäre Clients

Jeder PEP die Step-up-Bedingung auswerten kann. [\leq]

Hinweis: Die bei ZETA Client muss sich am ZETA-Guard registrieren, über den er auf geschützte Ressourcen zugreifen möchte. Dieser Prozess `zulässig gematik-LoA Werte` findet einmalig pro ZETA-Guard-Instanz statt. Der gesamte Prozess ist zweistufig, um die administrative Einrichtung von der technischen Inbetriebnahme zu trennen:

– **Initiale Registrierung:** Der Client erzeugt ein langlebiges kryptographisches Schlüsselpaar (Client Instance Key Pair; PrK.Client.Sig und PuK.Client.Sig), sendet `gematik-ehealth-loa-low`.

5.18.2.6 Token-Ausstellung

Wenn den öffentlichen Teil ZETA/ASL Kanal an den Authorization Resource Server und erhält im Gegenzug eine `client_id`. Der Client ist terminiert, danach im System bekannt, aber sein Status ist `pending attestation`, d.h. er benötigt die noch nicht für den Zugriff auf Ressourcen freigeschaltet.

– **Aktivierung (Erster Token Exchange)** Der Client wird aktiviert, iPEP HTTP Proxy im äußeren POST /ASL Request ein passendes er zum ersten Mal einens Access Token Exchange mit und einer erfolgreichen Attestierung DPoP Token. Die Ausstellung durchführt. Damit beweist er nicht nur den Besitz der privaten zusätzlichen Schlüssel (PrK.Client.Sig, PrK.DPoP.Sig, PrK.Client.SMCB), sondern (bei der TPM-Attestierung) auch die Integrität der Plattform, auf der er läuft. Nach erfolgreicher Prüfung wird sein Status im ZETA-Guard auf `active` gesetzt.

Mainline OPB1/ML-176019A_27799 – ZETA-Guard, Client-Registrierung für stationäre Clients

Der ZETA-Guard MUSS die Access Tokens folgt dem versionierten Token-Ausstellungsverfahren (Contract v1/v2, siehe 5.4.2.5): Der Claim benennt die Client-Registrierung für stationäre Clients gemäß Abbildung `Abb_Client_Rez` Zielressource über ihren `logistrierung` bereitstellen.

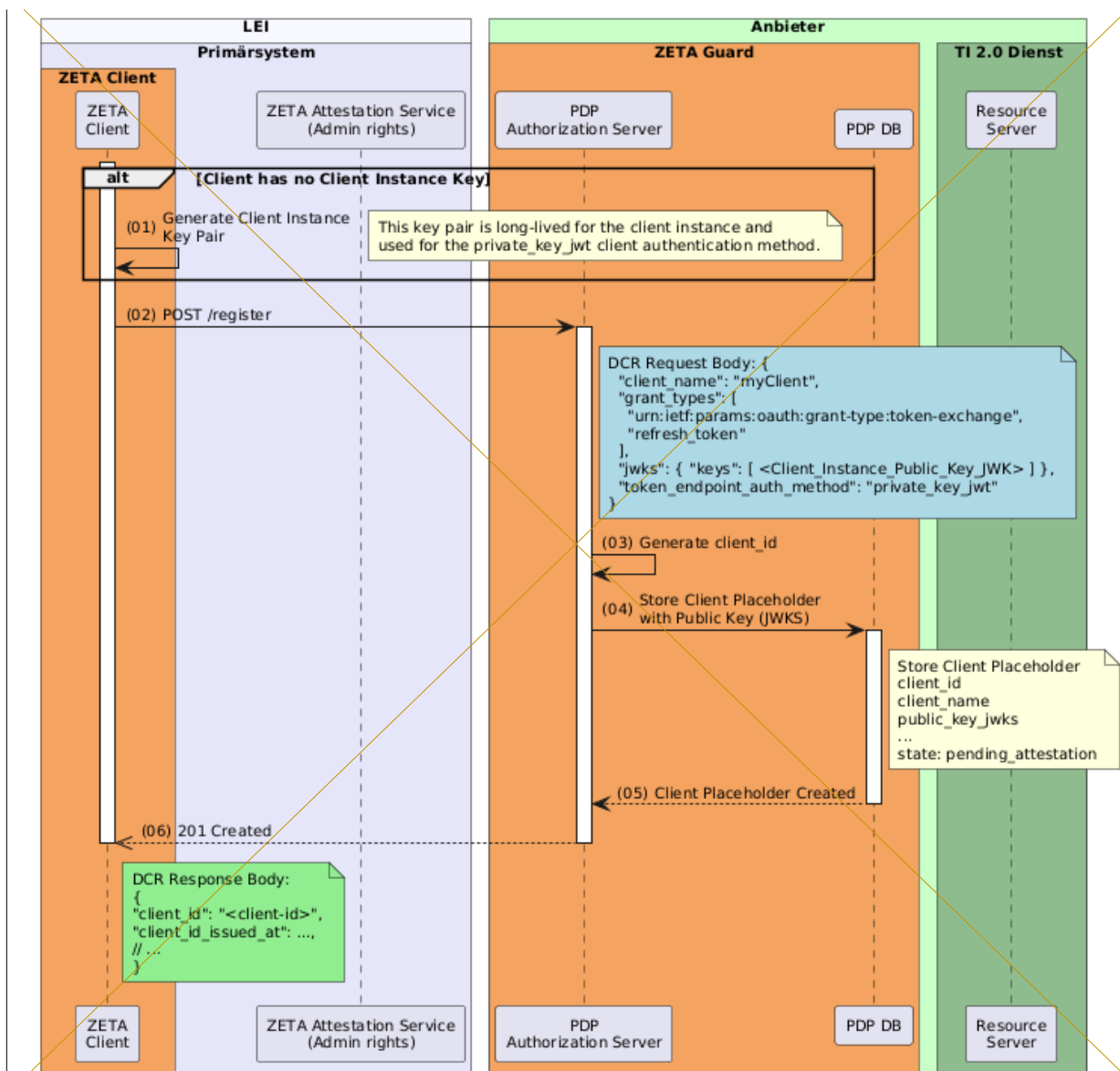


Abbildung 31: Abb_Client_Registrierung

{<=>}

Fürhner, während die initiale Registrierung sendet der ZETA-Client-URL ausschließlich eine Anfrage anlässlich über den Dynamic-Client-Registration (DCR)-Endpoint. Diese Anfrage enthält aPoP-Claim und gebunden wird.

Tabelle notwendigen Metadaten, um den Client für die private_key_jwt-Authentifizierung-ZETA-Token-Ausstellungsmethode vorzubereiten:
 -client_name: Ein für Mensch-ASL-am-Resource-Server

Token	claim	Wert
Access Token	aud	<logischer Bezeichner des ZETA/ASL Endpunkts> (von der Policy Engine vergeben, siehe 5.4.2.4)
	scope	asl

	<u>ver</u>	<u>2 (Contract v2) bzw. 1 (Contract v1, Legacy)</u>
<u>DPoP Token</u>	<u>htm</u>	<u>POST</u>
	<u>htu</u>	<u>https://<FQDN>/ASL</u>

hen lesbarer Name für den Client: Der <FQDN> ent-
 -token_endpoint_auth_method: Die geplante Authentifizierungsmethode, hier private_key_jwt.
 -grant_types: Die erlaubten Grant Types (z.B. urn:ietf:params:spricht dem <FQDN> aus dem Attribut resource des oAuth:grant-type:token-exchange, ref_Protected_refresh_token).
 -jwks: Eiource Well-known JSON Web Key Set, das den öffentlichen ClientDokuments.
Diese Endpunkt-URL https://<FQDN>/ASL wird nicht Instance Key (PuK.Client.Sig) enthält.
Dieser S den aud-Claim übernommen, sondern ausschüssel wird vom Authorization Server verwendet, um die Signaturießlich über den DPoP-Claim htu an den konkreten Request gebunden (RFC9449).
der Client Assertions (Client Authentifizierung) zu überprüfen.

5.18.2.7 Authentifizierung und Autorisierung für stationäre Clients

Nach erfolg aud-Claim des Access Tokens enthält den logischen Bezeicher Registrierung besitztner der ZETA Client eine client_id/ASL Endpunkts, die aen den Schlüssel PuK.Client.Sig gebunden ist. Die Policy Engine aus dem auf eingegebenen Fachdienst zugreifen zu können, benötigt der Client ein Access Token vom Authorization Server (AuthS). Stationäre ZETA Clients resource-Wert ableitet (Contract v2, ver: 2); bzw bei einem Legacy-Request (verwenden dafür den Token Exchange Flow, während mobile ZETA Clients den Authorization Code Flow mit OpenID Connect nutzen.

Die Authentifizierung und: 1) wird der Wert verbatim aus audience übernommen.

A_29861 -PDP Auth orisierung für stzationäre Clients unterscheidet zwei Hauptfälle:

1. **Server, Token-Austausch mit Attestierung:** Hier wird die Identität der Institution (mittels subject_token von bei ZETA/ASL Terminierung am Resource Server

der SM(C)-B) nachgewiesen und die Integrität des Clients durch eine Attestierung überprüft. Dieser aufwändigere Prozess wird zu Beginn eAuthorization Server MUSS, wenn:

- die Policy Engine neuen Session (oder zur Re-Attestier die Token-Ausstellung) durchgeführt, um sicherzustellen, dass erlaubt hat und
 - der ZETA Client und das Primärsystem vertrauenswürdig sind.
2. **Token-Erneuerung (Refresh Token)** Hier wird ein vorhandenes Refresh Token genutzt, um ein neue/ASL Kanal am Resource Server terminiert,

ein zusätzliches Access Token zu erhalten. Dieser Prozess ist performanter und verzichtet für den Zugriff auf eine erneute Attestierung.

Diese Trennung schafft eine Balance zwischen höchsten /ASL Endpunkt ausster Sicherheit beim initialen Zugriff und Effizienz bei der Erneuerung bestehender Sitzungen.

Die folgende Abbildlln gemäß Tab-ZETA-Token-Ausstellung zeigt den Ablauf des Token-Austauschs mit Client Assertion JWT Authentifizierung und DPoP.

Hinweis:-ASL-am-Resource-Server.

Das zusätzliche Access Token wird in Der TPM-Attestation Flow mussResponse aufgrund

einer Sicherheitslücke im Design überarbeitet werden und wird in ein den Token Request im Parameter Folgeversion nachgereicht: asl_access_token gesendet. [<=]

88742A_27800-02-ZETA Guard,A_29951 -PDP Authentifizierung und Autorisierung für stationäre Clients

Der ZETA Guard MUSS die Client-Registrierung für stationäre Clients gemäß Abbildung Abb_Authentifizierung und Autorisierung bereit **Server, Token Ausstellen**.

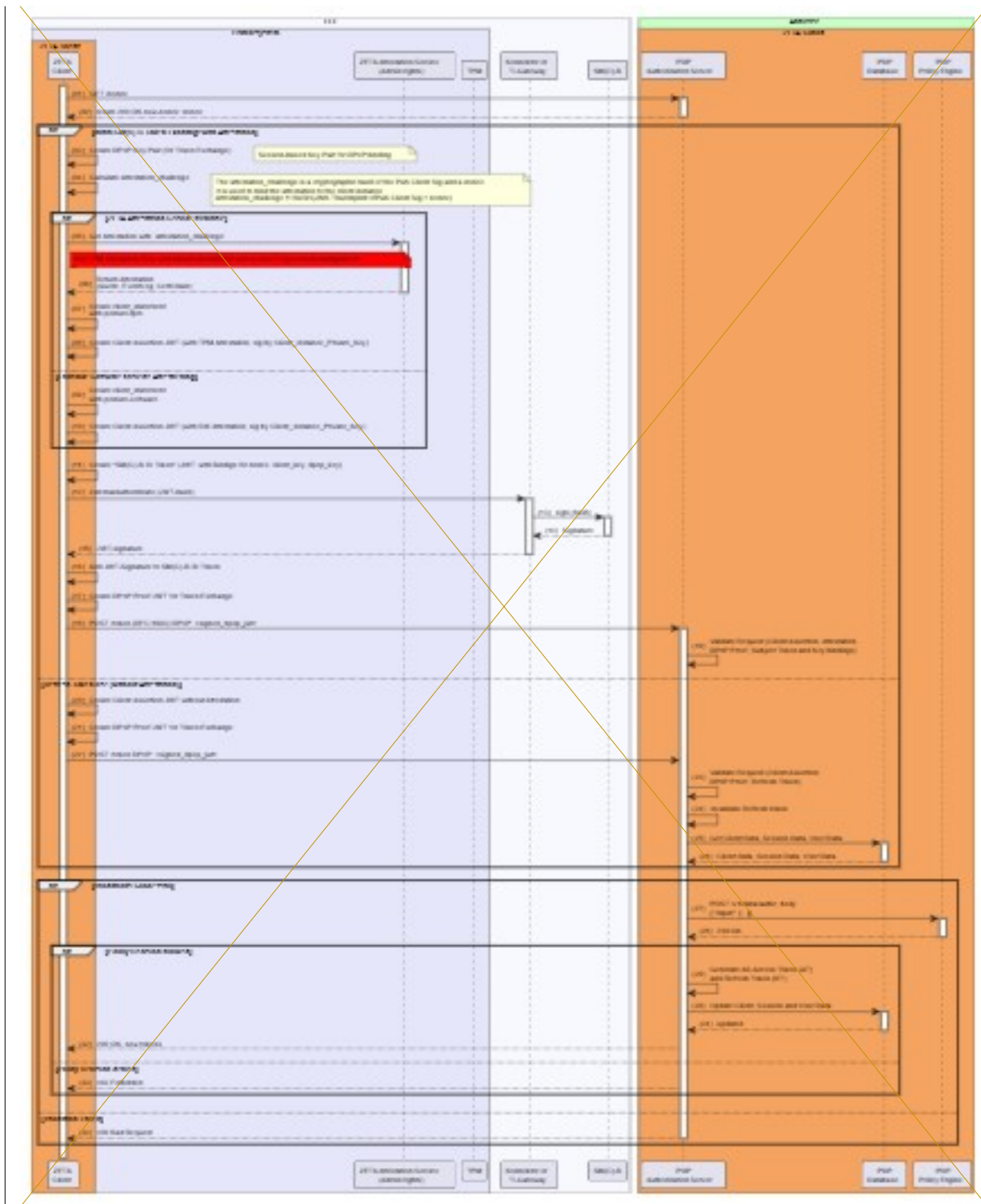


Abbildung 32: Abb_Auth

Identifizierung und Autorisierung

[<=]

5.18.2.7.1 Pfad A: Token-Austausch mit Attestierung

Dieser Pfad wird beschrieben, wenn der Client keine bestehende Session (d.h. kein gültiges Refresh Token) hat.

1. Vorbereitung:

— Der Client fordert eine frische, einmalig gültige nonce vom Authorization Server MUSS die Token Response nach dem Schema [token-response.yaml] erstellen. [<=]

A_29953 -PDP Authorization Server an (GET /nonce).

—, **Abfrage** Der Client erzeugt ein temporäres, nur für diese Session gültiges DPoP-Schlüsselpaar (PrK.DPoP.Sig, PuK.DPoP.Sig).

2. Integritätsprüfung und kryptografische Bindung:

— Um zu beweisen, dass die Attestierung vor der Token Ausstellung für genau diesen Client und diese Transaktion

Der Authorization Server MUSS vor der Client-Attestation eine attestation_challenge-Token Ausstellung Diese bindet den Zustand Decision des TPMs an den Thumbprint des öffentlichen Client Instance Key (PuK.Client.Sig) und die nonce des AuthS: attestation_challenge = HASH(Thumbprint(PuK.Client.Sig) + nonce).

— Der Client fordert beim ZETA Attestation Service eine TPM Quote an, die diese attestation_challenge als qualifyingData enthält. Das TPM signiert somit eine Aussage, die mit der Identität des Clients verbunden ist.

3. Policy Engine abfragen und den Request Body gemäß [policy-engine-input.yaml] Erstellen des Client Statement:

Wenn Die Attestierungsartefakte (TPM Quote, Event Log, Zertifikatskette) werden in einer client_statement-Struktur Response der Policy Engine gepackt. Im Falle des Falls [pdp-decision.yaml] allbacks (Software-Attestierung) ow: true enthält diese Struktur andere, softwarebasierte Evidenz.

4. Erstellen der Client Assertion, dann MUSS der Authorization (mit Attestierung)

Für Server die Authentifizierung am Token-Endpoint erstellt der Client eine Client Assertion. Dieses JWT, mit den Gültigkeitsdauern aus dem privaten Client Instance Key (PrK.Client.Sig) signiert, dient als "Umschlag":

— Es authentifiziert den Client gegenüber dem AuthS (iss und sub sind die client_id).

— Es er Response erstellen.

Wenn die Response allow: false enthält die client_statement-Struktur als Beweis für die Clientintegrität.

—// Client Assertion für, dann MUSS der Authorization Server initialen der Token-Austausch (Beispiel TPM)

```
{
  "iss": "cid-win-tpm-abede",
  "sub": "cid-win-tpm-abede",
  "aud": {
    "https://auth-server.zeta.gematik.de/ab/token"
  },
  "exp": 1678888310,
  "jti": "unique-jwt-id-win-tpm-789",
  "client_statement": {
    "sub": "cid-win-tpm-abede",
    "platform": "windows",
    "posture_type": "tpm",
    "posture": {
```

```

———"platform_product_id": {
———"package_family_name": "Primärsystem_123456789"
———}
———"product_id": "Primärsystem-win-v3",
———"product_version": "3.5.0",
———"os": "Windows 11 Pro",
———"os_version": "10.0.22621",
———"arch": "x86_64",
———"tpm_attestation_key": "base64 encoded tpm attestation key...",
———"tpm_quote": "base64 encoded tpm quote.l...",
———"tpm_event_log": "base64 encoded tpm event log...",
———"tpm_ek_certificate_chain": {
———"base64 encoded ek cert 1...",
———"base64 encoded ek cert 2..."
———}
———}
———"attestat-response.yaml] die reasons aus der Response der Policy Engine
übernehmen.
[<=]

```

```

Token Revocation_timestamp": 1678888010
—}
}

```

5.18.2.8 5. Authentisierung der Institution (SM(C)-B Token) Parallel dazu erstellt der Client das subject_token. Dies ist ein vom ZETA Client erzeugtes JWT, dessen Hash vom Konnektor mittels der SM(C)-B signiert wird und die Identität der Institution (z.B. Praxis) belegt. Die Audience (aud) dieses Token ist (RFC 7009)

A_29996 -PDP Authorization Server, Revocation-Endpunkt

der Token Endpoint des Authorization Servers. Das subject_token enthält ein Binding des Client Instance Keys (PuK.Client.Sig) und des DPoP Keys (PuK.DPoP.Sig) um Manipulationen durch einen Man in the middle zu verhindern.

// Beispiel für die Payload eines Subject_T MUSS einen Endpunkt POST /revoken

```

{
"jti": "unique-smcb-token-id-abc123",
"nonce": "bfb76c3e-8b4a-4f91-9d2a-3c7e5f8a1b20",
"iss": "cid-win-tpm-abcde",
"sub": "1-2-SMC-B-Testkarte-883110000129068",
"aud": {
———"https://auth-server.zeta.gematik.de/token"
}
"exp": 1678888370,
"iat": 1678888070,
"client_key": {
———"jkt": "NzbLsXh8uDCcd-6MnwXF4W-7noWXFZAfHkxZsRGC9Xs"
}
"dpop_key": {
———"jkt": "0Zc0CORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"
}
}
}

```

6. Token Request: Der gemäß [RFC7009] bereitstellen, über den ein Client sendet eine

eine POST-Anfrage an `gaten /token` Endpoint, die alle Teile kombiniert: `grant_type=token-exchange`, das `subject_token=Refresh Token`, die `client_assertion` (mit wider-eingebetteten Attestierung) und rufen kann den DPoP-Proof.

```


**Token Exchange Request Body: **
grant_type=urn:ietf:MUSS den params:oauth:grant_type:eter token-exchange
&subject_token=<SM(C) B signed Subject Token>
&subject_token_type=urn:ietf: enthalten und KANN den params:oauth:eter
token_type:jwt
&client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
bearer
&cli_hint (refresh_token) ent_assertion=<Client_Assertion_JWT>
&audience=<Resource Server Endpoint>
&scope=<Resource Server scopes>


```

7. Validierung durch den AuthS: Der AuthS führt eine umfassende Prüfung durch: Validierung der Client Assehalten.

Die URL des Revocation (Signatur gegen den bei der DCR hinterlegten Public Key PuK.Client.Sig), des DPoP Proofs, des Subject Token und insbesondere der **eingebetteten** Endpunkts MUSS im `[as-well-known.yaml]` im **Attestierung** (Prüfung der Quote, der `attestribut revocation_challenge` und der PCR-Werte `endpoint` angegen die Sicherheits-Policy) und der Key Bindings (Client Instance Key und DPoP Key).

Hinweis: Für die Signaturprüfung der Quote muss `derben` sein. [`<=`]

A 29997 -PDP Authorization Server die vertrauenswürdigen TPM Stamm- und Zwisc- **Client-Authensignaturzertifikate** installieren, die zum Signieren des Endorsement Key in den TPMs **verwzierung am Revocation**-endet werden. Eine Liste **punkt** der CAs wird hier bereitgestellt: `[TrustedTPM_RootCA]`.

Beispiel Policy Engine Response Body mit `"allow": true`:

```

{
  "result": {
    {
      "expressAuthorizations": {
        {
          "value": {
            "allow": true,
            "ttl": {
              "access_token": 300,
              "refresh_token": 86400
            }
          },
        },
        "text": "data.policies.zeta.authz.decision",
        "location": {
          "row": 1,
          "col": 1
        }
      }
    }
  }
}

```

Beispiel Policy Engine Response Body mit `"allow": false`:

```

{
  "result": {
    {
      "expressions": {

```

```


    {
      "value": {
        "allow": false,
        "reasons": [
          "User profession is not allowed",
          "One or more requested audiences are not allowed"
        ]
      },
      "text": "data.policies.zeta. Server MUSS Requests an POST /revoke in derselben Weise wie am Token-Endpoint authz.decision",
      "location": {
        "row": 1,
        "col": 1
      }
    }
  ]
}


```

Beispiel Token Response Body:

```

{
  "access_token": "eyJ...",
  "expires_in": 300,
  "refresh_expires_in": 86400,
  "refresh_token": "eyJ...",
  "token_type": "DPoP",
  "not_before_policy": 0,
  "session_state": "4c1a0db3-e67d-41db-aa46-b81b4c071de9",
  "scope": "vsdservice"
}

```

5.18.2.8.1 Pfad B: Token-Erneuerung via Refresh-Token

Dieser effiziente Pfad wird genutzt, wenn ein gültiges Refresh-Token vorhanden ist.
 1. **Erste** identifizieren (private_key_jwt Client Assertion [RFC7523] mit DPoP [RFC9449] und MUSS sicherstellen der Client, d**Assertion (ohne Attestierung)**) Der ein Client erstellt eine einfache `client_assertion`. Sie beweist durch ihre Signatur mit dem Client-Instance Key (`PrK.Client.Sig`) die Identität ausschließlich eigene Token widerrufen kann.
Schlägt diese Clients. Diese Assertion enthält keine Attest-Authentifizierungsdaten.

```


// Client Assertion für Refresh-Token-Nutzung
{
  "iss": "<client_id>",
  "sub": "<client_id>",
  "aud": "<AuthS-Token-Endpoint-URL>",
  "exp": "...",
  "jti": "..."
}


```

2. **Token Request:** fehl, MUSS Der Client sendet eine POST-Anfrage an den `/token-Endpoint` mit `grant_type=refresh_token`, dem Refresh-Token 401 (Unauthorized) und der einfachen `client_assertion`.

3. error=**in Validierung durch den AuthS:** Der AuthS validiert das Refresh-Token, die Signatur der Client Assertion und den DPoP-Proof. Die Prüfung einer TPM-Attestierung entfällt. Bei Erfolg wird das alte Refresh-Token invalidiert (`Rotat_client` gemäß `[zeta-error.yaml]`) antworten. **[<=]**

A_29998 -PDP Authorization)

5.18.2.8.2 Gemeinsame nachfolgende Schritte

Nach erfolgreicher Validierung **Server - Wirkung** in einem der **bedes Widen** Pfade fragt der AuthS **rrufs** bei der Policy Engine (PE) an, ob der Zugriff gewährt werden soll. Ist die Entscheidung positiv, stellt der AuthS ein neues Access Token (gebunden an den DPoP-Schlüssel PuK.DPoP.Sig) und ein neues Refresh Token aus.

5.18.2.9 Ablauf der **Aur Client-Authentifizierung bei Dienst-zu-Dienst-Kommunikation**

Um externen Diensten Zugriff auf den Resource **MUSS** der Authorization Server zu ermöglichen, wird (in ZETA Stufe 2) Workload Identity Federation verwendet. Jeder Kubernetes Cluster, der ZETA Guard ausführt, stellt einen Identity Provider (IDP) bereit, der Subject **das** im Parameter token übergebene Refresh Token für die im Cluster ausgeführten Workloads ausstellen kann.

sowie alle Der ZETA Guard Authorization **gehörigen** Server dient als Secure Token Service (STS) und kann per Toks **ion zugeordneten** Exchange ein Subject Tok- auch rotierten gegen ein Access- Refresh Token austauschen. Er überprüft dabei die **unverzüglich un** Gültigkeit des Subject Token, insbesondere ob der issuer uri des ausstellenden IDP registriert ist. Die Policy Engine (OPA) enthält **machen (analog A_29855) und** die Liste der registrierten issuer uri und wei **Session** terer Attribute, die zur Autorisierung herangezogen werden **minieren**.

Der folgende Ablauf beschreibt die Schritte, die für eine erf **Nachfolgreiche** Dienst-zu-Dienst-Kommunikation notwendig sind:

- **Registrierung des Issuers:** Die issuer URI des externen IDP (z. B. Kubernetes IDP, in dem die Client Workload läuft), oder ein Schlüssel (mit dem Subject **Tende Refresh Requests MÜSSEN** mit refresh_token für den T_revoken Exchange ausgestellt **abgelehnt** werden), wird bei **. Der Wider-ruf MUSS gematikäß A_29857** registriert. Eine Policy zur Überprüfung und die issuer URI werden in die PIP und PAP Daten des Ziel-ZETA Guard aufgenommen.

Subject visionssicher proToken-Anforderung: Die Collien **t Workload**, oder eine andere Workload im externen Kubernetes Cluster, fordert zunächst ein Subject Token von seinem **erden. [<=]**

- **A_29999 -PDP an.** Dieses Subject Token enthält Inform **uthorizationen** über die Identität d **Server** Workload.

Token Exchange-Anfrage: an ZETA Guard: Die externe **tWorkload** sendet das **tverhaltene** Subject **ohne** Token an den ZETA Guard **Enumeration**

- **Der** Authorization Server (STS) und fordert im Austausch **MUSS auf** ein Access Token an. Dieser Prozess wird als "Token Exchange" bezeichnet.

- **Validierung des Subject Token und Autorisierungsprüfung durch OPA:** Der ZETA Guard **Authorizen** erfolgreich client-authentifizierten Revocation Server validiert die Signatur **Request** und die Standard Claims des Subject Token. Anschließend fragt er die Policy Engi **abhängig davon, ob das übergebene (OPA) an, um zu überprüfen, ob der Aussteller (issuer URI) des Subject Token vertrauenswürdigToken gültig, bereits ungültig oder unbekannt ist und ob die im Token enthaltenen Attribute den definierten Richtlinien für den Zugriff, mit 200 OK antworten (RFC 7009 §2.2), um Rückschlüsse auf den angefragten Resource Server entsprechen. Die Policy Engine enthält hierfür eine Liste der zuvor registrierten Issuer.**

~~**Ausstellung des initialen Access-Token:** Nach erfolgreicher Überprüfung durch OPA stellt der ZETA-Guard die Existenz von Token zu verhindern. [\leq]~~

- ~~• **A 30000 -PDP** Authorization Server ein Access-Token aus. Dieses Token ist für den Zugriff auf den spezifischen Resource Server vorgesehen.~~

~~**Ausstellung innerhalb des Geltungsbereichs**~~

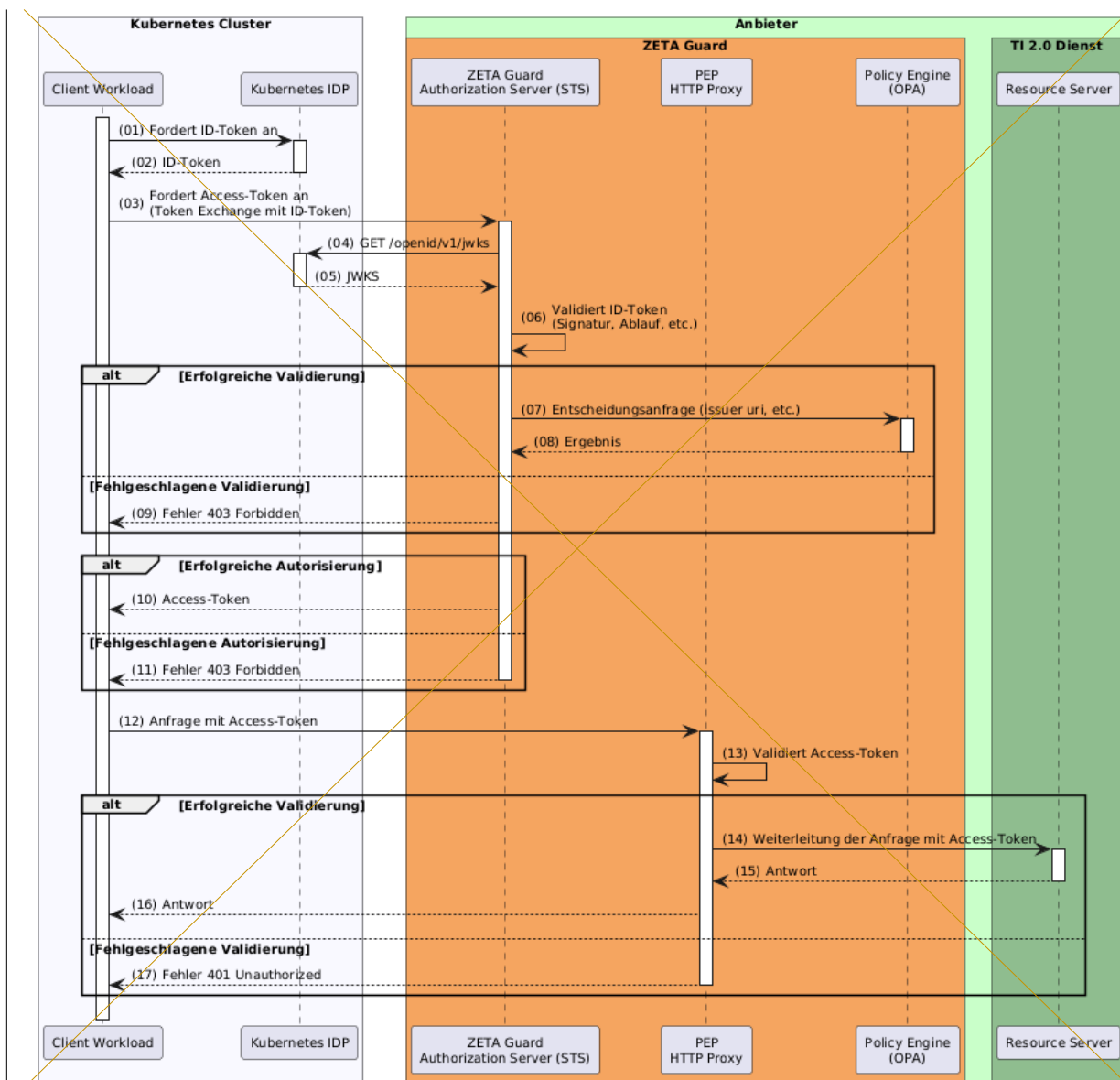
- ~~• **Wird ein Access Token-Set:** Nach erfolgreicher Überprüfung des Refresh-Token durch OPA stelltgeben, MUSS der ZETA-Guard AuAuthorization Server ein neues Token-Set aus. Dieses Token-Set ist für den Zugriff auf den spezifischen Resource Server vorgesehen.~~

- ~~• **Zugriff auf den Resource Server:** Die Client mit 200 OK ohne serverseitige Wirkung antWorkload im externen Dienst verwendet das erhten und KANN altene Access-Token, um Anfragen an den Resource Server zu stellen.~~

~~**Validierung des Access-rnativ error=unsupported Token durch den PEP HTTP Proxy:** Der PEP HTTP Proxy validiert bei jeder Anfrage das mitgesendete _type (400) gemäß RFC 7009 §2.2.1 zurückgeben. [\leq]~~

- ~~• **Hinweis:** Access-Token durch Überprüfung der Signatur des ZETA-Guard Authorization Servers sowie der aud und scope-claims.~~

- ~~• **Zugriffsentscheidung sind kurzlebig, DPoP-gebunden und Antwort:** Bei gültigem Access-Token gewährt der PEP HTTP Proxy den werden vom Resource Server Zugriff und verarbeitet die Anfrage. Andernfalls wird der Zugriff verweigert und eine entsprechende Fehlermeldung zurückgegeben.~~



Abstandslos geprüft; ihr Wildung 33: Abb_ZdETA-Guard-Dienst-zu-Dienst-Kommunikation

Mainline_OPB1/ML-180834A_28431-ZETA-Guard, Ablauf Dienst-zu-Dienst-Kommunikation

Der ZETA-Guard MUSS den Ablauf gemäß Abbildung Abb_ZETA-Guard-Dienst-zu-Dienst-Kommunikation unterstützen. [=<=] nicht Zweck dieses Endpunkts.

5.18.3 PDP Datenbank

Die Datenbank speichert Session-, Nutzer- und Client-Daten. Die Struktur und Verwaltung der Daten wird vom Authorization Server bestimmt.

A_26587-01 -PDP Datenbank - Kompatibilität

Die Komponente PDP Datenbank MUSS kompatibel zum Authorization Server sein. [=<=]

Hinweis: Die vom ZETA Guard Authorization Server Keycloak unterstützten Datenbanken werden im ZETA Guard Produkthandbuch aufgelistet (siehe auch <https://www.keycloak.org/server/db>).

5.18.4 Sicherheits- und Datenschutzerfordernungen an den PDP

A_28832 -PDP Policy Engine - Integritätsprüfung der Bundle-Signaturen

Die PDP Policy Engine MUSS sicherstellen, dass beim Import des Bundles mit den Policies und Daten die folgenden Signaturen erfolgreich validiert werden, bevor Bundle in die Policy Engine übernommen wird:

- Autor-Signatur
- Freigabe-Signatur

[<=]

A_25449 -PDP- Nutzeridentität nur von einem zugelassenem IDP

Der PDP MUSS sicherstellen, dass nur Nutzeridentitäten von einem zugelassenen IDP akzeptiert werden.[<=]

A_26281-01 -PDP Authorization Server - Schlüssel für Token-Signatur

Der PDP Authorization Server MUSS für die Signatur von Token asymmetrische Schlüsselpaare (PrK.AuthS.Sig, PuK.AuthS.Sig) verwenden. Die Gültigkeitsdauer (Rotationsintervall) dieser Schlüssel MUSS in den ZETA Guard-Manifest-Dateien konfigurierbar sein.

[<=]

~~53893A_25751 -PDP Client-Registrierung – Anwendungsfälle nur für registrierte mobile Clients~~

~~Nach der erfolgreichen Registrierung des ersten mobilen Clients (Gerät/App-Kombination), MUSS die Komponente Authorization Server sicherstellen, dass die folgenden Anwendungsfälle nur von einem registrierten mobilen Client durchgeführt werden können:~~

- ~~• Client löschen~~
- ~~• Client umbenennen~~
- ~~• E-Mail-Adresse hinzufügen~~
- ~~• E-Mail-Adresse aktualisieren~~

~~[<=]~~

~~Mainline_OPB1/ML-18A_25748-02 -PDP Client-Registrierung - Maximale Anzahl von Clients~~

~~Die Komponente Authorization Server MUSS sicherstellen, dass ein Nutzer maximal 256 Clients registrieren kann (konfigurierbares Limit).~~

~~Um eine Überlastung durch fehlerhafte Clients zu verhindern, MUSS der Authorization Server das Limit für die maximale Anzahl an Client-Registrierungen pro User durchsetzen. Wird dieses Limit erreicht, MUSS der Authorization Server bei der Anlage einer neuen Registrierung automatisch die am längsten inaktive Client-Registrierung dieses Users löschen, um Platz für die neue Registrierung zu schaffen.[<=]~~

A_28808 -PDP Authorization Server - Löschrufen

Der Authorization Server (PDP) MUSS konfigurierbare Inaktivitäts-Löschrufen für User- und Client-Daten durchsetzen und diese Daten nach Ablauf der Fristen löschen (Default-Wert: 1 Jahr).

Der Authorization Server MUSS Session-Daten automatisch löschen, wenn die Session expired ist.[<=]

A_25749 -PDP Client-Registrierung - Nutzer Protokollierung

Die Komponente Authorization Server MUSS ein Nutzerprotokoll führen und die folgenden Anwendungsfälle für den Nutzer protokollieren:

- Client hinzufügen

- Client löschen
- Client umbenennen
- E-Mail-Adresse hinzufügen
- E-Mail-Adresse aktualisieren

[<=]

A_25750 -PDP Client-Registrierung - Nutzer über sicherheitsrelevante Ereignisse informieren

Die Komponente Authorization Server MUSS sicherstellen, dass der Nutzer bei folgenden Anwendungsfällen informiert wird:

- Client hinzufügen
- Client löschen
- Client umbenennen
- E-Mail-Adresse aktualisieren

[<=]

Hinweis: Der Nutzer kann z.B. durch eine geeignete E-Mail oder App-Notifikation über die sicherheitsrelevanten Ereignisse informiert werden.

5.19 Policies und Daten

Die Policies und Daten werden in der ZETA Artifact Registry als OPA Bundles im OCI Format für die PDP Policy Engines der ZETA Guard Instanzen bereitgestellt. Die Bundles werden von der gematik in einem CI Prozess mit Quality Gates entwickelt und in die ZETA Artifact Registry deployed. Für jeden Fachdienst-Typ werden spezifische OPA Bundles erstellt. Falls es erforderlich ist, können auch pro ZETA Guard Instanz spezifische OPA Bundles erstellt werden.

Es gibt umgebungsspezifische Repositories in der ZETA Artifact Registry, sodass für Test und Entwicklung andere OPA Bundles verwendet werden können als für die Produktionsumgebung.

Unter dem label "latest" werden Bundles für die aktive Policy Engine bereitgestellt. Unter dem label "latest-sim" werden Bundles für die Simulations-Policy Engine bereitgestellt.

Im folgenden werden die Anforderungen an den Prozess zur Erstellung und Verteilung der OPA Bundles, dem ~~m-dem-n~~ Policies und Den Daten des CI/CD-Prozess, festgelegt.

A_25464-02 -Policies und Daten CI/CD-Prozess - Tamper-Proof Protokollierung von Administrationsaktivitäten

Der Policies und Daten CI/CD-Prozess MUSS ein "Tamper-Proof" Audit-Log von allen administrativen Vorgängen umsetzen. [<=]

Hinweis: Die Tamper-Proof Protokollierung von Adminiistrationsaktivitäten betrifft insbesondere Änderungen an der ZETA Artifact Registry.

A_25777-02 -Policies und Daten CI/CD-Prozess - Löschfristen Auditeinträge des Admin Audit-Logs

Der Policies und Daten CI/CD-Prozess MUSS sicherstellen, dass die Löschung eines Auditeintrags den gesetzlichen Vorgaben entspricht und frühestens nach 12 Monaten erfolgt. [<=]

A_25778-01 -Policies und Daten CI/CD-Prozess - Kontrolle des Audit-Logs

Der Policies und Daten CI/CD-Prozess MUSS so umgesetzt werden, dass das Audit-Log mindestens alle 3 Monate im Vieraugenprinzip kontrolliert wird. Die Rolle des Prüfers DARF NICHT an der Administration der Infrastruktur zur Bereitstellung der Policies und Daten teilnehmen. Bei der Kontrolle ist insbesondere auf ungewöhnliche, nicht nachvollziehbare oder malizöse Administratoraktivitäten zu achten. [≤]

A_25465-01 -Policies und Daten CI/CD-Prozess - Änderungen nur durch berechnigte Nutzer

Der Policies und Daten CI/CD-Prozess MUSS sicherstellen, dass nur berechnigte Nutzer Änderungen von Policies oder PIP-Daten durchführen können. [≤]

A_25466-01 -ZETA Artifact Registry - Sicherheitsmeldung bei Aktualisierung von Policies oder PIP-Daten

Die ZETA Artifact Registry MUSS sicherstellen, dass bei der Aktualisierung der Policies oder der PIP-Daten eine Sicherheitsmeldung automatisiert über die von der gematik angebotene Schnittstelle an das TI SIEM-System übermittelt wird. [≤]

A_25467-01 -Policies und Daten CI/CD-Prozess - Änderungen nur unter 4 Augen

Der Policies und Daten CI/CD-Prozess MUSS sicherstellen, dass Änderungen in Policies oder PIP-Daten nur im Vieraugenprinzip durchgeführt werden können. [≤]

5.20 Telemetriedaten-Service

Der Telemetriedaten-Service ist eine Implementierung des OpenTelemetry-Frameworks der die Telemetriedaten aller ZETA Guard-Komponenten einsammelt (Collector) und für die Weitergabe (Exporter) an externe Systeme (Monitoring des Anbieters, gematik Telemetriedaten Schnittstelle und gematik SIEM) aufbereitet.

Zusätzlich nimmt der Telemetriedaten-Service Fehlermeldungen (Traces), Bestandsdaten (Metriken) und Selbstauskunftsdaten (Logs) des Resource Servers entgegen und leitet sie an die gematik Telemetriedaten Schnittstelle weiter.

A_27260 -ZETA Guard, Telemetriedaten-Service, Weitergabe der Daten ohne Profilbildung

Der Telemetriedaten-Service im ZETA Guard MUSS die von den Komponenten des ZETA Guard gesammelten Telemetriedaten so verändert an das Monitoring System des Anbieters und an die gematik Telemetriedaten Schnittstelle sowie SIEM der gematik weitergeben, dass eine Profilbildung nicht mehr möglich ist.

[≤]

A_27261 -ZETA Guard, Telemetriedaten-Service, Protokoll

Der Telemetriedaten-Service im ZETA Guard MUSS zur Weitergabe der Daten das OpenTelemetry Protokoll (OTLP) über gRPC oder über HTTP/JSON verwenden. [≤]

A_27492-02 -ZETA Guard, OpenTelemetry Unterstützung

Die ZETA Guard Komponenten HTTP Proxy, Authorization Server, Policy Engine und Notification Service MÜSSEN für alle durchgeführten Operationen Telemetriedaten in Form von Traces an den Telemetriedaten-Service senden. [≤]

87630A_27727-012 -ZETA Guard, Telemetriedaten-Service, Lieferung

Der Telemetriedaten-Service im ZETA Guard SOLL Telemetriedaten asynchron an den gematik Telemetriedaten-Service liefern. Dafür MUSS der Telemetriedaten-Service des ZETA Guard für die Telemetriedatenlieferung von Traces an den gematik Telemetriedaten-Service als den OTLP-ExportProcessorType den Typen mit eingeschaltetem Batch-ing verwenden (dies ist auch der Standardwert bei OpenTelemetry).

[≤]

A_27725-01 -ZETA Guard, Telemetriedaten Service, Status Codes

Der Telemetriedaten Service im ZETA Guard MUSS im Parameter `http.response.status_code` einen HTTP-Statuscode gemäß `Tab_gemSpec_ZETA_Telemetriedaten_HTTP_Statuscodes` übermitteln.

Tabelle 23: Tab_gemSpec_ZETA_Telemetriedaten_HTTP_Statuscodes

HTTP-Statuscodes	Name der Statuscodegruppe	Beschreibung
1xx	INFORMATIONAL	Der Server hat die Anfrage erhalten und befindet sich in der Bearbeitung.
2xx	SUCCESSFUL	Die Operation wurde erfolgreich durchgeführt.
3xx	REDIRECTION	Der Client muss zusätzliche Maßnahmen ergreifen, um die Anfrage abzuschließen.
4xx	CLIENT_ERROR	Ein Client-seitiger Fehler verhindert die erfolgreiche Durchführung der Operation.
5xx	SERVER_ERROR	Ein Server-seitiger Fehler verhindert die erfolgreiche Durchführung der Operation.

[<=]

Hinweis: Es sind vom Anbieter, anstatt der Status Code Klassen (first digit of status code), die konkreten 3-stelligen HTTP-Statuscodes gemäß [RFC9110] zu verwenden.

76224A_27494-012 -Telemetriedaten-Service, Custom Collector für Selbstauskunft

Der Telemetriedaten-Service MUSS einen ~~Custom Collector (oder einen OTEL Collector mit einem Custom Processor)~~ für die Selbstauskunft des Resource Servers ~~implementierbereitstellen~~. Die Selbstauskunft des Resource Servers erfolgt für jede eigenständige Instanz als OTLP Log Record.

Hinweis: Der OTLP Log Record für die Selbstauskunft ist ~~wie folgt definiert:~~

Body: ~~"Selbstauskunft".~~

Attributes:

- ~~• Name des Produkts~~
- ~~• Aktuelle Produktversion~~
- ~~• Version des in gemSpec_Perf (Tab_gemSpec_Perf_Telemetriedaten) Produkt-Typs~~
- ~~• Konfigurationsversion~~
- ~~• Name des Pods/der Instanz des Resource Servers~~
- ~~• Der Zeitpunkt der Erstellung des Log Records (wird automatisch gesetzt).~~

~~ct_info) definiert. [<=]~~

Beispiel: Selbstauskunft OTLP-Log-Nachricht (JSON) vom Resource Server an den Telemetriedaten-Service. Die Log-Nachricht wird unverändert an die gematik Telemetriedaten-Schnittstelle weitergeleitet.

```
{
  "resourceLogs": [
    {
      "scopeLogs": [
        {
          "logRecords": [
            {
              "timestamp": "1678886400000000000",
              "body": { "stringValue": "Selbstauskunft" },
              "attributes": [
                { "key": "product_name", "value": { "stringValue": "Backend-Service" } },
                { "key": "product_version", "value": { "stringValue": "1.2.0" } },
                { "key": "producttype_version", "value": { "stringValue": "1.2.3" } },
                { "key": "configuration_version", "value": { "stringValue": "cfg-rev-45" } },
                { "key": "pod_name", "value": { "stringValue": "my-app-pod-1" } }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

5.21 HSM Proxy

Der HSM Proxy dient als Abstraktionsschicht zwischen den ZETA Guard Kernkomponenten (insbesondere dem AuthS und dem HTTP Proxy) und den physischen oder cloudbasierten Hardware Security Modulen (HSM). Er stellt sicher, dass kryptographisches Schlüsselmaterial (Private Keys) die sichere Umgebung des HSM niemals verlässt. Die Kommunikation mit dem HSM Proxy erfolgt zustandslos über das gRPC-Protokoll.

Neben der Erzeugung von Signaturen ermöglicht der HSM Proxy ein sicheres Schlüsselmanagement durch Key Wrapping. Hierbei werden Schlüssel innerhalb der sicheren HSM-Umgebung durch Key Encryption Keys (KEK) verschlüsselt, sodass sie außerhalb des HSM gespeichert werden können, ohne die Vertraulichkeit zu gefährden.

Hinweis: Aktuell wird der HSM Proxy immer durch den Anbieter/Hersteller des TI 2.0 Dienstes bereitgestellt.

A_28829 -HSM Proxy, Schnittstelle zu den Komponenten

Der ZETA Guard HSM Proxy MUSS die Schnittstelle zu den Komponenten gemäß [hsm-proxy.proto] umsetzen. [≤]

A_28830 -HSM Proxy, Attribute Based Access Control

Der ZETA Guard HSM Proxy MUSS den Zugriff auf die Operationen der Schnittstelle nach berechtigter Komponente und key_id einschränken können.

Die berechtigten Komponenten müssen mit mTLS authentifiziert werden können. [≤]

A_28831 -HSM Proxy, key_id

Der ZETA Guard HSM Proxy MUSS die key_id nach dem Schema[Komponente]-[Zweck]-[Algorithmus]-[Version/Datum] umsetzen.[<=]

Beispiel für eine key_id: zeta-guard-keycloak-jwt-es256-v1 -> Darf nur von Keycloak genutzt werden.

5.21.1 Sicherheits- und Datenschutzerfordernungen an den HSM Proxy**A_28746 -ZETA Guard - HSM Proxy in einer VAU**

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der Hersteller des TI 2.0-Dienstes sicherstellen dass der HSM-Proxy in einer VAU läuft.[<=]

A_28748 -HSM-Proxy - Zugriff auf HSM Proxy durch attestierte ZETA-Komponenten

Falls der ZETA Guard in einer VAU umgesetzt wird, MUSS der HSM Proxy sicherstellen, dass den Zugriff auf den ZETA relevanten Schlüssel ausschließlich durch attestierte ZETA-Komponenten erfolgen kann.[<=]

A_28747 -HSM-Proxy - Zugriff auf ZETA-Schlüssel über HSM-Proxy

Falls der HSM-Proxy in einer VAU umgesetzt wird, MUSS der Anbieter des TI 2.0-Dienstes sicherstellen, dass Zugriffe auf die ZETA-relevanten Schlüssel oder die Schnittstellen zur Nutzung der Schlüssel im HSM ausschließlich durch einen HSM-Proxy erfolgen können, der entweder

- gegenüber der HSM attestiert ist oder
- gekoppelt (paired) ist, d. h., dass der Proxy im Rahmen einer dokumentierten Schlüsselzeremonie eindeutig mit dem HSM(-Cluster) bzw. der HSM(-Cluster)-Partition verbunden wird; die dabei erzeugten und zugewiesenen kryptographischen Zugriffscredentials sind für das HSM Proxy exklusiv).

[<=]

A_28814 -HSM-Proxy - Sealing von Credentials

Der HSM-Proxy MUSS im Falle eines Pairings mit dem HSM seine lokal auf dem Server gehaltenen Zugriffs-Credentials mittels Sealing schützen und nach einem Neustart des Systems damit nur für die korrekt gestartete Verarbeitungskontexte des HSM-Proxy wieder verfügbar machen.[<=]

A_28797-01 -HSM-Proxy - Isolation HSM Proxy

Falls das HSM-Proxy in einer VAU umgesetzt wird, MUSS die VAU mittels eines wirksamen technischen Separationsmechanismus gewährleisten, dass Verarbeitungen in anderen Verarbeitungskontexten weder direkt noch indirekt die Integrität oder Sicherheit der Verarbeitungen einem HSM-Proxy beeinträchtigen können.[<=]

A_28816 -HSM-Proxy - Minimale Trusted-Computing-Base

Der HSM-Proxy MUSS als gehärteter Software-Stack umgesetzt sein und insbesondere dem Prinzip der minimalen Trusted-Computing-Base folgen, um die Härting bzw. die notwendige Robustheit des Ergebnisses der Begutachtung zu erreichen. Die Trusted-Computing-Base DARF NICHT mehr Komponenten enthalten als für die Bereitstellung der HSM-Proxy-Funktionalität erforderlich.[<=]

A_28817 -HSM-Proxy - Keine Persistierung von Request/Response Daten

Der HSM-Proxy DARF Request- oder Response-Daten NICHT persistieren bzw. cachen. [<=]

A_28819 -HSM-Proxy - Integrität und Authentizität der Konfiguration und Rollback-Schutz

Der HSM-Proxy MUSS mit technischen Mitteln (wie Sealing, Signaturen und monotonen Versionszählern) sicherstellen, dass die Integrität und Authentizität geladener Konfigurationsdaten sichergestellt sind und dass ein Downgrade auf eine nicht mehr autorisierte Version der Konfiguration abgewehrt wird. [<=]

A_28820 -HSM-Proxy - Eingabe Validierung von Operationen

Der HSM-Proxy MUSS sicherstellen, dass alle Daten und Parameter, die über eine API kommuniziert werden, sicherheitstechnisch validiert werden. [<=]

A_28821 -HSM-Proxy - Schutzmaßnahmen gegen die OWASP Top 10 Risiken

Der HSM-Proxy MUSS geeignete technische Maßnahmen zum Schutz vor den Risiken in der aktuellen Version der [OWASP-Top-10-Risiken] umsetzen. [<=]

A_28822 -HSM-Proxy - Datenschutzkonformes Logging und Monitoring

Der HSM-Proxy MUSS die für den Betrieb des Zero Trust erforderlichen Logging- und Monitoring-Informationen in solcher Art und Weise erheben und verarbeiten, dass mit technischen Mitteln ausgeschlossen ist, dass dem Anbieter eines TI 2.0-Dienstes vertrauliche oder zur Profilbildung geeignete Daten zur Kenntnis gelangen. [<=]

5.22 Betrieb

~~5.23 Die Komponenten des ZETA Guard werden in einer Kubernetes (KNotification Service~~

Der ZETA Notification Service ist die zentrale Fassade vor den Push Gateways und übernimmt innerhalb der Telematikinfrastruktur die Aufgabe, Benachrichtigungen eines Fachdienstes (Resource Server) an die zugehörigen Endgeräte der Versicherten zuzustellen. Er baut auf dem Push-Notification-Konzept der gematik [gemF_PushNotification].

Fachlich ordnet sich der Notification Service als Bindeglied zwischen dem Fachdienst und der plattformspezifischen Push-Infrastruktur der Betriebssystemhersteller (Apple APNs, Google FCM) ein. Er entkoppelt den Fachdienst von den Details der Push-Zustellung, verwaltet die Push-Konfigurationen und Pusher der Clients und stellt sicher, dass Nachrichteninhalte und Nutzeridentifikatoren geschützt verarbeitet werden.

Der Dienst wird in zwei Betriebsvarianten betrieben: mit VAU, bei der Notification Service und Resource Server gemeinsam in einer Vertrauenswürdigem Ausführungsumgebung laufen und HSM-Anbindung sowie At-rest-Verschlüsselung und Pseudonymisierung aktiv sind, sowie ohne VAU, bei der diese HSM-gestützten Schutzmaßnahmen entfallen. Die Verschlüsselung der Nachrichten ist ein pro ZETA-Guard-Instanz zentral konfigurierbares Feature; ist es nicht aktiv, werden Nachrichten unverschlüsselt weitergegeben.

5.23.1 Ablauf des Push-Versands

Das folgende Architekturbild zeigt die am Push-Versand beteiligten Komponenten und ihr Zusammenspiel entlang der Zustellkette vom Fachdienst bis zur FdV App. Es umfasst die optionale VAU mit Resource Server, Notification Service und dessen Datenhaltung, die HSM-Anbindung über den ZETA HSM Proxy, das Push Gateway sowie die plattformspezifische Push-Infrastruktur von Apple und Google. Die eingeklammerten Marker (0) bis (7) sowie (T) und (H) kennzeichnen die einzelnen Ablaufschritte, die im Anschluss an das Bild erläutert werden.

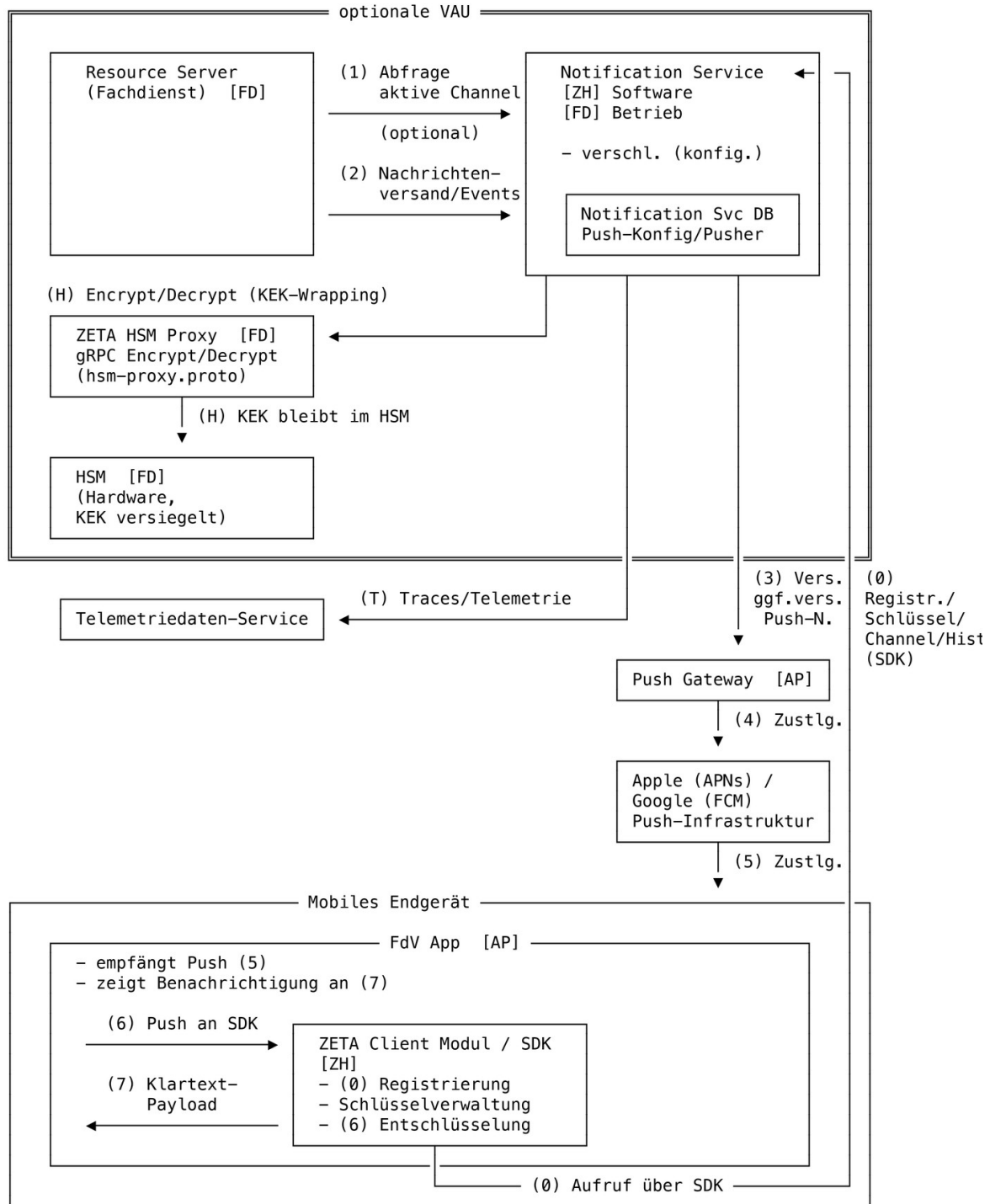


Abbildung 34 - Ablauf Notification Versand

- (0) Registrierung über das SDK: Das ZETA Client Modul (SDK) in der FdV App registriert die Pusher, verwaltet die Schlüssel und konfiguriert die Channel beim Notification Service. Optional kann der Client über das Feature Nachrichten-Historie bereits zugestellte oder ältere Benachrichtigungen abrufen, sofern der Betreiber dieses Feature aktiviert hat. Die Authentisierung erfolgt mit einem dienstspezifischen, DPoP-gebundenen ZETA-Zugriffstoken für den Notification Service.

- (1) Optionale Channel-Abfrage: Der Resource Server (Fachdienst) fragt beim Notification Service die aktiven Channel ab. Die Kommunikation erfolgt innerhalb der VAU über einen technischen Nutzer.
- (2) Nachrichtenversand vom Fachdienst: Der Resource Server übergibt die Notification-Events an den Notification Service. Dieser verschlüsselt die Nachrichten nur dann, wenn das zentral pro ZETA-Guard-Instanz konfigurierte Verschlüsselungs-Feature aktiv ist; andernfalls werden sie unverschlüsselt weitergegeben. Die Authentisierung erfolgt innerhalb der VAU über einen technischen Nutzer.
- (3) Versand an das Push Gateway: Der Notification Service leitet die Push-Nachricht an das Push Gateway weiter, verschlüsselt bei aktivem Verschlüsselungs-Feature, sonst unverschlüsselt. Die Verbindung ist über mTLS abgesichert.
- (4) Zustellung an die Push-Infrastruktur: Das Push Gateway übergibt die Nachricht an die plattformspezifische Push-Infrastruktur von Apple (APNs) beziehungsweise Google (FCM).
- (5) Zustellung an das Endgerät: Die Push-Infrastruktur stellt die Push-Nachricht an das mobile Endgerät zu.
- (6) Verarbeitung im SDK: Die FdV App reicht die empfangene Push-Nachricht an das integrierte ZETA Client Modul (SDK) weiter, das sie verarbeitet und entschlüsselt.
- (7) Anzeige in der App: Das SDK übergibt den Klartext-Payload an die FdV App, die die Benachrichtigung anzeigt.
- (T) Telemetrie: Der Notification Service liefert für alle Operationen Traces und Telemetriedaten an den Telemetriedaten-Service.
- (H) HSM-Zugriff: Der Notification Service nutzt in der Betriebsvariante mit VAU den ZETA HSM Proxy für das KEK-Wrapping (Encrypt/Decrypt). Data Encryption Key und Pseudonymisierungs-HMAC-Schlüssel werden in der VAU erzeugt und per HSM-KEK versiegelt persistiert; der KEK verlässt das HSM nicht.

5.23.2 Authentisierung

Der Notification Service besitzt drei authentisierungspflichtige Schnittstellen mit unterschiedlichen Vertrauensmodellen:

- ZETA Client Modul / SDK → Notification Service: nutzergebundener Zugriff, authentisiert über ein dienstspezifisches, DPoP-gebundenes ZETA-Zugriffstoken des ZETA Guard.
- Resource Server → Notification Service: Service-to-Service-Zugriff innerhalb der (optionalen) VAU, authentisiert über einen technischen Nutzer per mTLS Terminierung innerhalb der VAU-Vertrauensgrenze).
- Notification Service → Push Gateway: mTLS mit EV-Zertifikat in der Rolle des Fachdienstes gemäß [gemF_PushNotification]. Normativ: A_2998s2.

5.23.3 Rollen der Komponenten

Die Rollen der beteiligten Komponenten:

- **ZetaClient Modul / SDK:** integrierter Bestandteil der FdV App; übernimmt die Registrierung der Pusher inkl. Schlüsselverwaltung und Channel-Konfiguration, sowie die Verarbeitung/Entschlüsselung der von der App weitergereichten Nachrichten und gibt den Klartext-Payload zur Anzeige an die FdV App zurück.

Notification Service: Fassade vor den Push Gateways; registriert/verwaltet Push-Konfigurationen, beantwortet die optionale Channel-Abfrage, nimmt Resource-Server-Events entgegen, verschlüsselt die Nachrichten (sofern konfiguriert) ~~Umgebung des TI-2.0 Dienst-Anbieters betrieben.~~

- Anfordernd leitet sie an die Push Gateways weiter. Teil der optionalen VAU (gemeinsam mit dem Fachdienst). Übernimmt die Push-Verarbeitungsrolle des Fachdienstes aus [gemF_PushNotification] (Pusher-/Schlüssel-/Channel-Verwaltung, Verschlüsselung, Versand).
- **Notification Service DB:** persistiert Push-Konfigurationen/Pusher. In Variante (A) liegen alle Daten verschlüsselt at-rest (Envelope-Verschlüsselung mit HSM-versiegeltem Schlüssel); die Nutzeridentifikatoren (KVNR/Telematik-ID) werden ausschließlich pseudonymisiert (schlüsselgebundener MAC) gespeichert und indiziert.
- **ZETA HSM Proxy:** einheitliche gRPC-Schnittstelle (vgl. Kap. 5.11) zum HSM des Anbieters. Der Notification Service nutzt ausschließlich die bestehende Schnittstelle mit Encrypt/Decrypt (KEK-Wrapping). Sowohl der Data Encryption Key (DEK) als auch der Pseudonymisierungsschlüssel werden in der VAU erzeugt und per HSM-KEK (Encrypt/Decrypt) versiegelt gespeichert; der KEK verlässt das HSM nicht. Die eigentliche Ver-/Entschlüsselung sowie die Pseudonymbildung erfolgen mit dem entwrappten Schlüssel innerhalb der VAU. Zugriff nur durch attestierte/gepaarte VAU-Instanzen.
- **Resource Server**(Fachdienst): fragt optional die aktiven Channel ab und erzeugt bzw. versendet die Notification-Events; liegt ebenfalls in der VAU (falls genutzt).
- **Push Gateway und Apple/Google Push-Infrastruktur:** plattformspezifische Zustellkette (4)/(5) zum Endgerät.
- **FdV App:** empfängt die Push-Nachricht (5), reicht sie zur Entschlüsselung an das integrierte ZETA Client Modul / SDK weiter und zeigt die Benachrichtigung an.

5.23.4 Anforderungen

A_29957 -Notification Service - Betriebsvarianten und Geltungsbereich des HSM-Schutzes

Der Notification Service MUSS in einer der beiden folgenden Betriebsvarianten betrieben werden:

- Variante (A) - mit VAU: ZETA Guard und Notification Service werden gemeinsam in einer VAU betrieben. In dieser Variante MUSS der Notification Service die HSM-Anbindung sowie die At-rest-Verschlüsselung und Pseudonymisierung gemäß A_29958 - A_29963 umsetzen.
- Variante (B) - ohne VAU: Betrieb ohne VAU-Umgebung. In dieser Variante kommen die HSM-Anbindung und die HSM-gestützte Verschlüsselung/Pseudonymisierung nach A_29958 - A_29963 NICHT zur Anwendung; der Schutz der persistierten Daten MUSS durch die allgemeinen betrieblichen und organisatorischen Maßnahmen des Betreibers sichergestellt werden.

Die Anforderungen A_29958 - A_29963 sind ausschließlich auf Variante (A) anzuwenden. Unabhängig von der gewählten Variante bleiben die Anforderungen zu Authentisierung, Autorisierung, Transportschutz sowie der Schutz personenbezogener Daten gegenüber dem Push Gateway (A_29984) uneingeschränkt gültig.

[<=]

A_29958 -Notification Service - Schlüssel ausschließlich aus dem VAU-HSM

Der Notification Service MUSS sämtliches Schlüsselmaterial zur Verschlüsselung und Pseudonymisierung der persistierten Daten durch ein VAU-HSM schützen und hierfür

ausschließlich die Schnittstelle des ZETA Guard HSM Proxy (A_28829) in ihrer bestehenden Form (Encrypt/Decrypt) verwenden. Der KEK DARF das HSM NICHT im Klartext verlassen. Der DEK und der Pseudonymisierungsschlüssel werden in der VAU erzeugt und ausschließlich HSM-KEK-versiegelt persistiert; im Klartext DÜRFEN sie nur innerhalb der VAU vorliegen. Entwrappte Schlüssel DÜRFEN ausschließlich im flüchtigen Speicher derVAU-Instanz vorgehalten (gecacht) werden, DÜRFEN NICHT persistiert werden und MÜSSEN bei Beendigung der VAU-Instanz sowie bei Schlüsselrotation (A_29963) sicher verworfen werden.[<=]

A_29959 -Notification Service - Verschlüsselung von Pusher- und Statusdaten

Der Notification Service MUSS Pusher-Konfigurationen, Channel-/Geräte-Zuordnungen, den Nachrichten-/Versandstatus sowie das Push-Schlüsselmaterial (initial shared secret iss, shared-secret-Jahr-Monat, AES/GCM-Schlüssel-Jahr-Monat) vor der Persistierung at-rest verschlüsseln. Das Push-Schlüsselmaterial besitzt gemäß [gemF_PushNotification] Kap. 5.5 (Schutzbedarf des iss: sehr hoch) den höchsten Schutzbedarf. Die Verschlüsselung MUSS als Envelope-Verschlüsselung umgesetzt werden: Ein in der VAU verwendeter Data Encryption Key (DEK) wird durch einen im HSM gehaltenen Key Encryption Key (KEK) über HsmProxyService.Encrypt/Decrypt gewrappt/entwrappt (Sealing gemäß Kap. 5.15); der KEK DARF das HSM NICHT verlassen. Die eingesetzten kryptographischen Verfahren, Schlüssellängen und Nutzungsgrenzen (einschließlich Nonce-/IV-Handhabung und Mengengrenzen der symmetrischen Verschlüsselung) MÜSSEN den Vorgaben aus [gemSpec_Krypt] entsprechen. Die Massen-Ver-/Entschlüsselung mit dem entwrappten DEK DARF ausschließlich innerhalb der VAU erfolgen; Erzeugung, Cache-Haltung und sicheres Verwerfen des DEK richten sich nach A_29958.[<=]

A_29960 -Notification Service - Pseudonymisierung von KVNR und Telematik-ID

Der Notification Service MUSS Nutzeridentifikatoren (KVNR, Telematik-ID) nicht im Klartext, sondern ausschließlich als Pseudonym persistieren und indizieren. Das Pseudonym MUSS als schlüsselgebundener MAC über den Identifikator gebildet werden; das eingesetzte MAC-Verfahren und die Schlüssellänge MÜSSEN [gemSpec_Krypt] entsprechen. Der zugehörige Pseudonymisierungsschlüssel MUSS in der VAU erzeugt und ausschließlich HSM-versiegelt (per HsmProxyService.Encrypt/Decrypt) persistiert werden; die eigentliche MAC-Berechnung MUSS mit dem entwrappten Schlüssel innerhalb der VAU erfolgen (Erzeugung, Cache-Haltung und sicheres Verwerfen gemäß A_29958). Die Pseudonymbildung MUSS deterministisch sein (gleicher Identifikator → gleiches Pseudonym), damit die für (1) und (2) benötigten Lookups (Channel-/Pusher-Abfrage je Nutzer) ohne Klartext-Identifikator möglich sind. Eine Rotation des Pseudonymisierungsschlüssels verändert alle abgeleiteten Pseudonyme und ist daher nur unter den in A_29963 (Punkt 3) genannten Bedingungen (vollständige Re-Pseudonymisierung oder versioniertes Vorhalten der vorherigen Schlüsselgeneration) zulässig, damit der deterministische Lookup nicht bricht.
[<=]

A_29961 -Notification Service - Domänentrennungen an Hersteller einer ZETA-Kompo der Pseudonyme

Der Notification Service MUSS für KVNR und Telematik-ID unterschiedliche Ableitungsdomänen verwenden - entweder über ein in der VAU eingebundenes Domänen-/Kontext-Label (kvnr / tid) bei der schlüsselgebundenen MAC-Berechnung oder über getrennte Pseudonymisierungsschlüssel -, sodass Pseudonyme nicht über verschiedene Identifikortypen oder über andere Dienste hinweg verkettet werden können (Domain-Separation).
[<=]

A_29962 -Notification Service - Zugriff auf HSM nur durch attestierte VAU-Instanz

5.23.5 Der Notification Service MUSS sicherstellen, dass die HSM-Proxy-Operationen zur Envelope-Verschlüsselung (Encrypt/Decrypt) sowie das HSM-Wrapping/Entwrapping des Pseudonymisierungsschlüssels (Encrypt/Decrypt) ausschließlich durch eine attestierte, integrale VAU-Instanz ausgelöst werden können (attestierter bzw. gepairter HSM Proxy gemäß gemSpec_ZETA A_28746-A_28748; Zugriffsbeschränkung nach Komponente und key_id per mTLS/ABAC gemäß A_28830). Weder die DEK noch entschlüsselte Inhalte oder Klartext-Identifikatoren DÜRFEN die VAU-Vertrauensgrenze verlassen. [<=]

76164A_27851-A_29963 -Notification Service - Schlüsselrotation und 4-Augen-Prinzip

Der Betreiber des Notification Service MUSS das Einbringen und Verwalten der HSM-KEK ausschließlich im 4-Augen-Prinzip zulassen (analog A_24612-*, A_27499). Die key_id des KEK MUSS dem Schema [Komponente]-[Zweck]-[Algorithmus]-[Version] gemäß gemSpec_ZETA A_28831 folgen (z. B. notification-service-dek-kek-aesgcm-v1).

Der Notification Service MUSS eine Schlüsselrotation für sämtliches eingesetztes Schlüsselmaterial (HSM-KEK, VAU-seitig erzeugte DEK bzw. Wrapping-Keys sowie den Pseudonymisierungsschlüssel) unterstützen und implementieren. Das konkrete Rotationsverfahren ist nicht vorgegeben und liegt in der Verantwortung des Herstellers/Betreibers; es MUSS jedoch folgende Ergebnisse sicherstellen:

1. Eine Rotation MUSS sowohl regulär (geplant) als auch außerordentlich (z. B. bei Verdacht auf Schlüsselkompromittierung) ohne Betriebsunterbrechung der fachlichen Schnittstelle durchführbar sein.
2. Über eine Rotation hinweg MÜSSEN bereits persistierte Daten weiterhin entschlüsselbar bleiben (kein Klartext-/Datenverlust). Hierzu MUSS je Datensatz die verwendete key_id/Schlüsselgeneration nachvollziehbar bleiben, und die jeweils ältere Schlüsselgeneration MUSS so lange verfügbar gehalten werden, bis der Bestand vollständig auf die neue Generation überführt ist.
3. Eine Rotation des Pseudonymisierungsschlüssels DARF den nach A_29960 geforderten deterministischen Lookup nicht brechen: Sie MUSS entweder mit einer vollständigen Re-Pseudonymisierung des Bestands einhergehen oder die vorherige Schlüsselgeneration für Lookups vorhalten (versioniertes Pseudonym), bis die Re-Pseudonymisierung abgeschlossen ist.

Der entwrappte DEK bzw. Pseudonymisierungsschlüssel MUSS bei einer Rotation gemäß A_29959/A_29960 sicher aus dem VAU-Cache verworfen werden. [<=]

Hinweis zur Abgrenzung: Die hier beschriebene Verschlüsselung/Pseudonymisierung schützt die at-rest in der Notification Service DB gespeicherten Daten. Sie ist unabhängig von der Ende-zu-Ende-Verschlüsselung der eigentlichen Push-Payload zwischen Notification Service und ZETA Client Modul/SDK sowie vom mTLS-Transportschutz gegenüber dem Push Gateway.

A_29964 -Notification Service - Realisierung der RS-Schnittstelle gemäß OpenAPI

Der Notification Service MUSS die Schnittstelle gegenüber dem Resource Server gemäß der OpenAPI-Spezifikation [OpenApi-Notification-Service-RS] (notification-service-rs-endpoint.yaml, Version 1.0.0) umsetzen und dabei mindestens die Operationen

getActiveChannels (GET /users/{userId}/channels) und submitNotification (POST /notifications) anbieten. [<=]

A_29965 -Notification Service - Strikt asynchrone Verarbeitung

Der Notification Service MUSS eine eingehende Notification (POST /notifications) nach erfolgreicher Validierung mit 202 Accepted und einer notification_id bestätigen und die Zustellung an das Push Gateway asynchron durchführen. Die 202-Antwort DARF NICHT als Zustellbestätigung an das Endgerät interpretiert werden. [<=]

A_29966 -Notification Service - Synchrone Vorprüfung der Zustellvoraussetzungen

Ein Nutzer ist dem Notification Service nur bekannt, solange für ihn mindestens ein registriertes Gerät (Pusher) existiert; die Channel-/Pusher-Daten werden ausschließlich geräte-/pushkey-gebunden gehalten (kein eigenständiger Nutzer-Datensatz ohne Gerät). Ist für den adressierten Identifikator kein Gerät registriert, MUSS der Notification Service den Nutzer als unbekannt behandeln und die Operation (POST /notifications bzw. GET /users/{userId}/channels) mit 404 (USER_NOT_FOUND) beantworten; ein terminaler Status no_registered_devices wird nicht verwendet.

Der Notification Service MUSS bei der Annahme einer Notification synchron prüfen, ob der adressierte Channel für den Nutzer aktiv (enabled) ist. Ist der Channel auf keinem registrierten Gerät enabled, MUSS er die Notification nicht an das Push Gateway weiterleiten und den terminalen Status no_active_channel bereits in der 202-Antwort zurückgeben. In diesem terminalen Fall MUSS der Status synchron in der Antwort übermittelt werden; eine Persistierung eines Notification-Datensatzes oder ein nachgelagertes Status-Update DARF dabei NICHT erfolgen - eine ggf. zurückgegebene notification_id wird nur transient erzeugt. Ein (verschlüsselter) Notification-Datensatz DARF erst angelegt werden, wenn tatsächlich ein Versand an das Push Gateway erfolgt (Status accepted). [<=]

Geltungsebene der Channel-Prüfung (Gerät vs. Nutzer): Die Channel-Konfiguration wird gemäß [gemF_PushNotification] A_27190 geräteindividuell pro pushkey geführt; ein Nutzer kann mehrere FdV-Instanzen/Geräte (Pusher) mit je eigener Channel-Auswahl registrieren. Die RS-seitige Schnittstelle GET /users/{userId}/channels bzw. die Vorprüfung beim Versand arbeitet demgegenüber auf Nutzerebene: Der Notification Service MUSS den Channel als für den Nutzer aktiv behandeln, wenn der Channel für mindestens ein registriertes Gerät (pushkey) des Nutzers den Status enabled hat (Aggregation über alle Geräte des Nutzers), und den Versand anschließend an jeden Pusher ausführen, dessen pushkey den adressierten Channel auf enabled gesetzt hat. Die gerätegenaue Auswertung bleibt damit Push-Spec-konform; die Nutzerebene ist nur eine Aggregation/Abstraktion für die RS-Schnittstelle und führt keine zweite, abweichende Channel-Haltung ein.

A_29967 -Notification Service - Nutzeridentifikation über KVNR oder Telematik-ID

Der Notification Service MUSS den Nutzer ausschließlich über die Kombination aus Identifikatorwert (userId/user.value) und deklariertem Typ (id_type ∈ {kvnr, telematik-id}) auflösen und die Werte gemäß den in der OpenAPI definierten Formaten/Pattern validieren. Die interne Persistierung/Indizierung erfolgt ausschließlich pseudonymisiert (siehe A_29960/A_29961). [<=]

A_29968 -Notification Service - Konfigurierbare, transparente Verschlüsselungsentscheidung

Die Verschlüsselung der Nachrichten-Payload MUSS ein pro **ZETA Guard** ~~---~~ **Rückwärtskompatibilität**

Der Hersteller des ZETA-Guard-Instanz konfigurierbares Feature sein, das durch den Fachdienst-Anbieter bzw. ZETA-Betreiber gesetzt wird. Der Notification Service MUSS die Entscheidung, ob und wie eine Notification-Payload verschlüsselt wird, ausschließlich aus

dieser eigenen (Instanz-)Konfiguration (je Fachdienst/Channel) ableiten. Der Resource Server DARF im Request keine Verschlüsselungsanweisung übergeben, und der Notification Service DARF eine solche NICHT auswerten (Transparenz gegenüber dem RS).

Die Konfigurierbarkeit ist durch die Spezifikation des jeweiligen Fachdienstes begrenzt: Schreibt die Fachdienst-Spezifikation die Verschlüsselung zwingend vor (z. B. ePA, E-Rezept), so MUSS das Feature in der Instanz aktiv sein und der Betreiber DARF es NICHT deaktivieren. Nur wo die Fachdienst-Spezifikation die Verschlüsselung als optional zulässt (z. B. TI-M), DARF der Betreiber sie je Instanz aktivieren oder deaktivieren. Bei aktivem Feature gelten die Schlüsselableitungs- und Verschlüsselungspflichten nach A_29970. [<=]

A_29969 -Resource Server - Nutzung der Notification-Service-Schnittstelle

Der Resource Server MUSS zum Versand von Push Notifications und zur optionalen Channel-Abfrage ausschließlich die Schnittstelle [OpenApi-Notification-Service-RS] des Notification Service verwenden und das reference-Feld - sofern genutzt - als unverschlüsselten Identifier ohne personenbezogene Daten (keine KVNR, kein Name, kein Nachrichteninhalte) befüllen. [<=]

A_29970 -Notification Service - Übernahme der Schlüsselableitungs-/Verschlüsselungspflichten

Der Notification Service MUSS, soweit er gemäß Konfiguration verschlüsselt versendet, die Verschlüsselungs- und Schlüsselableitungspflichten aus [gemF_PushNotification] übernehmen, insbesondere:

- initiale und fortlaufende Schlüsselableitung mittels HKDF-SHA256 auf Basis shared-secret-Jahr-Monat mit Ableitungsvektor yyyy-MM (A_27157, A_27158-01, A_27160),
- AES/GCM-Verschlüsselung des Nachrichteninhalts und Kodierung als Base64(IV || Ciphertext || Authentication Code) (A_27161),
- 1024-Byte-Padding zur Größenverschleierung (A_27610),
- Einbetten von time_message_encrypted im Format yyyy-MM (A_27162),
- Löschen überholten Schlüsselmaterials nach jeder Ableitung (A_27405),
- UTF-8-Kodierung des Nachrichteninhalts (A_27680).

[<=]

A_29971 -Notification Service - Übernahme der Versand- und Pusher-Pflichten

Der Notification Service MUSS die Versandpflichten des Fachdienstes aus [gemF_PushNotification] beim Weiterleiten an das Push Gateway übernehmen, insbesondere:

- Exponential Back-off bei Nichtverfügbarkeit/Fehlern des Push Gateways (A_27166),
- Verarbeitung der Push-Gateway-Antwort und Löschen ungültig gewordener Pusher (A_27374),
- Versand ausschließlich an die in der Registrierung hinterlegte URL (A_27652).

Für die Übermittlung MUSS der Notification Service den zur Betriebsart passenden Endpunkt des Push Gateways nutzen:

- unverschlüsselt: POST /notify (Einzelnachricht, Antwort { rejected[] }) oder POST /notify/batch (Stapel, Antwort BatchNotificationResponse),
- verschlüsselt: ausschließlich POST /notifyEncrypted/batch (Stapel); ein Einzel-Endpunkt für verschlüsselte Nachrichten existiert in der Push-Spezifikation nicht - auch eine einzelne verschlüsselte Nachricht MUSS sicher-stdedher als Batch (mit einem Element) gesendet werden.

Der Notification Service MUSS die in der Antwort zurückgemeldeten abgelehnten Pushkeys (rejected je Notification bzw. je Batch-Item) auswerten und den Versand an diese Pushkeys einstellen, dass bei Änderungen an den sowie die zugehörigen Pusher entfernen (Push-Spezifikation RejectedPushKeys, [gemF_PushNotification] A_27374). Dies MUSS auch dann erfolgen, wenn der abgelehnte Pushkey nicht aus der aktuellen, sondern aus einer früheren Notification resultiert. [<=]

A_29972 -Notification Service - Client-Modul-Schnittstelle

Der Notification Service MUSS für das ZETA Client Modul / SDK (Schritt (0)) die Fachdienst-seitigen Schnittstellen aus [gemF_PushNotification] realisieren – mindestens die Endpunkte zur Pusher-Registrierung/-Verwaltung (GET /pushers, POST /pushers/set, A_27104) und zur Channel-Konfiguration (GET /channels, GET/POST /channels/{pushkey}, A_27190) . Die Channel-Konfiguration wird dabei – gemäß Push-Spezifikation – geräteindividuell je pushkey gehalten (GET /channels/{pushkey} liefert die Konfiguration für das Gerät mit diesem pushkey, POST /channels/{pushkey} aktualisiert sie); der Status je Channel ist enabled, disabled oder not_set, wobei not_set wie disabled behandelt wird. Eine nutzerbezogene Sicht (z. B. GET /users/{userId}/channels der RS-Schnittstelle) MUSS der Notification Service als Aggregation über alle pushkey-Konfigurationen des Nutzers ableiten (siehe A_29966) und DARF NICHT als eigenständige, von der geräteindividuellen Konfiguration abweichende Channel-Haltung implementiert werden. [<=]

A_29973 -Notification Service - Umsetzung des Features Nachrichten-Historie

Das Feature Nachrichten-Historie (Push-Historie) ist in [gemF_PushNotification] als optionales Fachdienst-Feature beschrieben. In der ZETA-Notification-Service-Implementierung ist es als Pflichtfunktion umzusetzen: Der Notification Service MUSS das Feature Nachrichten-Historie implementieren. Dazu MUSS er

- die fachdienstseitigen Historien-Endpunkte /history/* für authentifizierte Nutzer (ZETA Client Modul / SDK, Authentisierung gemäß A_29975-A_29978) bereitstellen, über die nach dem Versand auf die referenzierten versendeten Push-Nachrichten zugegriffen werden kann, und
- den beim Versand an das Push Gateway mitgegebenen, PII-freien identifier (Feld reference, siehe A_29969) so verarbeiten/persistieren, dass eine vollständige, datenschutzkonforme Liste der versendeten Notifications je Nutzer rekonstruierbar ist (kein Personenbezug im identifier, keine Klartext-Inhalte im Push Gateway).

Die Persistierung der Historiendaten MUSS den Schutzanforderungen der Pusher und Statusdaten (vgl.. A_29959) genügen. [<=]

A_29974 -Notification Service - Optionale Aktivierung durch den Betreiber

Die Nutzung/Aktivierung des Features Nachrichten-Historie KANN der Betreiber des Notification Service (der ZETA-/FD-Anbieter) je Betrieb bzw. je Fachdienst konfigurieren; sie ist für den Betreiber optional. Ist das Feature deaktiviert, MUSS der Notification Service die Historien-Endpunkte /history/* deaktiviert halten (Beantwortung mit 404/501) und DARF keine Historiendaten persistieren. Die Implementierung des Features im Produkt bleibt davon unberührt gemäß A_29973 verpflichtend. [<=]

A_29975 -Notification Service - Authentisierung des Client-Moduls über ZETA-Zugriffstoken

Der Notification Service MUSS Aufrufe des ZETA Client Moduls / SDK (Registrierung/Verwaltung von Pushern, Schlüsselverwaltung, Channel-Konfiguration) über ein vom ZETA Guard (AuthS) ausgestelltes ZETA-Zugriffstoken (OAuth 2.0 Access Token, at+jwt gemäß [RFC9068], DPoP-gebunden gemäß [gemAPI_ZETA]) authentisieren. Das Token MUSS spezifisch für den Notification Service ausgestellt sein (siehe A_29979). Aufrufe ohne gültiges Token MUSS er mit 401 ablehnen. [<=]

A_29976 -Notification Service - Prüfung des ZETA-Zugriffstoken

Der Notification Service MUSS bei jedem Aufruf durch das Client-Moduls Signatur, Gültigkeitszeitraum exp / nbf / iat, Aussteller iss sowie die Zielgruppe aud - diese MUSS der eigenen Resource-Kennung des Notification Service entsprechen (siehe A_29979) - sowie die für die Operation erforderlichen Scopes/Berechtigungen des Tokens prüfen und das Token bei einer fehlenden oder fehlgeschlagenen Prüfung zurückweisen 401 / 403. Ein Token, dessen aud den Notification Service nicht adressiert, DARF er NICHT akzeptieren. [<=]

A_29977 -Notification Service - Sender-Constrained Token (Token-Binding)

Der Notification Service MUSS sicherstellen, dass das ZETA-Zugriffstoken sender-constrained ist (z. B. via DPoP oder mTLS-gebundenem Token gemäß den ZETA-Vorgaben) und DARF ein Token NICHT akzeptieren, dessen Nachweis der Besitzbindung (Proof-of-Possession) fehlt oder ungültig ist. Damit werden Token-Replay und Bearer-Token-Missbrauch verhindert.

[<=]

A_29978 -Notification Service - Bindung von Operationen an den authentisierten Nutzer

Der Notification Service MUSS Schreib-/Leseoperationen des Client-Moduls (Registrierung, Schlüssel, Channel-Konfiguration eines Pushers) ausschließlich auf die im ZETA-Zugriffstoken authentifizierte Nutzeridentität beziehen und MUSS Operationen auf Pusher/Daten anderer Nutzer mit 403 ablehnen. [<=]

A_29979 -Notification Service - Dienstspezifisches Access Token (Audience/Resource & Discovery)

Der Zugriff des Client-Moduls auf den Notification Service MUSS mit einem dienstspezifischen Access Token erfolgen, das ausschließlich für den Notification Service ausgestellt ist; ein für einen anderen Resource Server (Fachdienst) ausgestelltes Token DARF NICHT verwendet bzw. akzeptiert werden. Hierzu MUSS:

- der Notification Service ein eigenes Protected-Resource-Discovery-Dokument (GET /.well-known/oauth-protected-resource) gemäß [gemAPI_ZETA] bereitstellen, das die eigene resource-Kennung (stabile URI, z. B. https://<fd>/notification-service), die zuständigen authorization_servers, die unterstützten scopes_supported sowiedpop_bound_access_tokens_required: true deklariert,
- der Client das Token beim Token Exchange (POST /token) gezielt für diese Resource anfordern (Resource Indicator resource gemäß RFC 8707), sodass der AuthS die aud des Tokens auf die Resource-Kennung des Notification Service setzt,
- das Token ausschließlich die folgend normativ festgelegten Scopes tragen, die der Notification Service je Operation nach dem Least-Privilege-Prinzip auswertet; eine Operation ohne den ihr zugeordneten Scope MUSS er mit 403 ablehnen.

Scope	erlaubt ausschließlich zu (Endpunkte gemäß A_29972/A_29973)
notification.pusher.read	GET /pushers
notification.pusher.write	POST /pushers/set
notification.channel.read	GET /channels, GET /channels/{pushkey}
notification.channel.write	POST /channels/{pushkey}
notification.history.read	GET /history/* (nur bei aktiviertem Feature, A_29974)

Der Notification Service MUSS die aud-Bindung gemäß A_29976 durchsetzen. Ein Token mit einem nicht hier aufgeführten Scope DARF NICHT zur Autorisierung einer Operation der Client-Modul-Schnittstelle herangezogen werden. Andere als die in Tabelle X aufgeführten Scopes DARF der AuthS für die Resource-Kennung des Notification Service NICHT ausstellen. [\leq]

Hinweis (Umsetzungsvariante PEP): Die in A_29975-A_29977 und A_29979 geforderten Prüfungen des ZETA-Zugriffstokens (Signatur, exp/nbf/iat, iss, aud, Scopes, Sender-Constraining/DPoP) KANN der Betreiber statt im Notification Service selbst durch einen dem Notification Service vorgelagerten, dedizierten Policy Enforcement Point (PEP HTTP Proxy, vgl. Kap. 5.7.1) erbringen; der Ingress routet die Aufrufe des Client-Moduls dann an diesen PEP, der nur erfolgreich autorisierte Anfragen weiterleitet. Die Anforderungen gelten in diesem Fall als durch den vorgelagerten PEP erfüllt. Die objektbezogene Nutzerbindung nach A_29978 verbleibt in jedem Fall beim Notification Service.

A_29980 -Notification Service - mTLS-Authentisierung des Resource Servers

Der Notification Service MUSS alle Aufrufe des Resource Servers (getActiveChannels, submitNotification) über gegenseitiges TLS (mTLS) mit einem technischen Nutzer authentisieren, das Client-Zertifikat des Resource Servers prüfen und Verbindungen mit fehlendem oder ungültigem Zertifikat ablehnen (401). [\leq]

A_29981 -Notification Service - mTLS-Terminierung innerhalb der VAU

Der Betreiber des Notification Service MUSS die für die mTLS-Authentisierung verwendeten Client-/Server-Zertifikate gemeinsam mit dem Betreiber des Resource Servers ausstellen und verwalten sowie eine Sperrung/Rotation kompromittierter Zertifikate ermöglichen. [\leq]

A_29982 -Notification Service - mTLS mit EV-Zertifikat zum Push Gateway

Der Notification Service MUSS die Verbindung zum Push Gateway über mTLS absichern - Schnittstellen rn und dabei - in der Rolle des Fachdienstes gemäß [gemF_PushNotification] - ein Extended-Validation-TLS-Zertifikat gemäß den Anforderungen des CA/Browser Forums verwenden bzw. ein solches des PushGateways prüfen (vgl. A_28267, A_28268). [\leq]

A_29983 -Notification Service - Mandantentrennung / Autorisierung je Fachdienst

Der Notification Service MUSS nach erfolgreicher Authentisierung autorisieren, dass ein Resource Server bzw. ein technischer Nutzer ausschließlich Notifications für die ihm zugeordneten Fachdienste/Channels versenden und ausschließlich die Channel-/Geräteinformationen dieser Fachdienste abfragen darf. Nicht autorisierte Zugriffe MUSS er mit 403 ablehnen. [\leq]

A_29984 -Notification Service - keine Breaking-Changes eingef. personenbezogenen Klartextdaten Richtung Push Gateway

Der Notification Service DARF NICHT personenbezogene Daten (insbesondere KVNR, Telematik-ID, Name) oder unverschlüsselte Nachrichteninhalte an das Push Gateway übermitteln (vgl. A_27436, A_27539). Konsistent zu A_27396 DARF der Notification Service in den an das Push Gateway weitergereichten Registrierungs-/Versanddaten keine KVNR oder Telematik ID führen; die Adressierung erfolgt ausschließlich über pushkey/app_id. [\leq]

A_29985 -Notification Service - Registry zulässiger Channels

Der Notification Service MUSS eine Konfiguration/Registry der je Fachdienst zulässigen Channels führen. Eine Notification für einen dem Fachdienst nicht zugeordneten Channel MUSS er mit 403 und eine Notification für einen dem Nutzer unbekanntem Channel mit 422 (UNKNOWN_CHANNEL, siehe [OpenApi-Notification-Service-RS])

zurückweisen; syntaktisch ungültige

Channel-Angaben werden, bei gleichzeitiger Wahrung der Rückwärtskompatibilität für all, gemäß A_29986 mit 400 abgelehnt. Ein Channel gilt als „dem Nutzer unbekannt“ (422) nur dann, wenn er nicht in der Channel-Registry des Fachdienstes geführt wird, für den Fachdienst des Aufrufers aber grundsätzlich zulässig wäre. Ist der Channel in der Registry geführt, aber auf keinem registrierten Gerät des Nutzers enabled (einschließlich des Status not_set, der wie disabled behandelt wird, vgl. A_29972), antwortet der Notification Service mit 202 und dem terminalen Status no_active_channel (A_29966).[<=]

A_29986 -Notification Service - Eingabevalidierung

Der Notification Service MUSS alle eingehenden Requests gegen das in der OpenAPI definierte Schema validieren (Pflichtfelder, Formate/Pattern für KVNR und Telematik-ID, id_type) und ungültige Requests mit 400 ablehnen.[<=]

A_29987 -Notification Service - Schutz vor Überlast und Missbrauch

Der Notification Service MUSS Maßnahmen gegen Überlast und Missbrauch implementieren (Rate-Limiting mit 429/Retry-After, Begrenzung von Payload- und Stapelgrößen) und bei temporärer Nichtverfügbarkeit mit 503/Retry-After antworten. Überschreitet eine Notification die zulässige Payload-Größe, MUSS er sie mit 413 (PAYLOAD_TOO_LARGE, siehe [OpenApi-Notification-Service-RS]) zurückweisen.[<=]

A_29988 -Notification Service - Datensparsamkeit und Aufbewahrungsfristen

Der Notification Service MUSS Notification-Status- und Verarbeitungsdaten (notification_id, Status) nur für eine kurze, definierte Aufbewahrungsfrist speichern und nach deren Ablauf löschen. Hiervon unberührt bleiben die Historiendaten des Features Nachrichten-Historie (A_29973), sofern dieses durch den Betreiber aktiviert ist; deren Aufbewahrung richtet sich nach der Betreiber-Konfiguration. Bei deaktiviertem Feature DARF der Notification Service keine Inhalts-/Versandhistorie vorhalten.[<=]

A_29989 -Notification Service - Löschung bei Deregistrierung

Der Notification Service MUSS bei Deregistrierung eines Pushers durch das Client-Modul die zugehörige Push-Konfiguration sowie das gespeicherte kryptographische Schlüsselmaterial (shared-secret-Jahr-Monat, AES/GCM-Schlüssel-Jahr-Monat) löschen (analog A_27156)[<=]

A_29990 -Notification Service - Protokollierung und Telemetrie ohne Klartext-PII

Der Notification Service MUSS sicherheitsrelevante Ereignisse sowie Betriebs-/Telemetriedaten protokollieren, dabei aber keine Klartext-Identifikatoren (KVNR, Telematik-ID) und keine entschlüsselten Nachrichteninhalte protokollieren; Identifikatoren sind ausschließlich pseudonymisiert zu protokollieren. Eine etwaige Übermittlung an einen Telemetrie-/Monitoring-Dienst MUSS ebenfalls frei von Klartext-PII erfolgen.[<=]

A_29991 -Notification Service - TLS-/Krypto-Konformität

Der Notification Service MUSS für alle TLS-/mTLS-Verbindungen (RS-Schnittstelle sowie Push Gateway) die in [gemSpec_Krypt] zugelassenen TLS-Versionen, Cipher-Suiten und Schlüssellängen verwenden. Für die Verbindung zum Push Gateway gilt ergänzend A_29982 (EV-Zertifikat).[<=]

A_29992 -Notification Service - Idempotenz des Versands

Der Notification Service SOLL Retransmissionen des Resource Servers idempotent behandeln. Übermittelt der Resource Server einen Idempotency-Key, MUSS der Notification Service innerhalb der Aufbewahrungsfrist nach A_29988 wiederholte Übermittlungen mit demselben Schlüssel erkennen und dieselbe notification_id ohne erneuten Versand zurückgeben. Das reference-Feld ist hierfür NICHT geeignet, da es nicht eindeutig sein muss (A_29969).[<=]

5.24 Betrieb

Die Komponenten des ZETA Guard werden in einer Kubernetes (K8s) Umgebung des TI 2.0 Dienst-Anbieters betrieben.

5.24.1 Anforderungen an Hersteller einer ZETA Komponente

A_27851 -ZETA Guard - Rückwärtskompatibilität

Der Hersteller des ZETA Guard MUSS sicher stellen, dass bei Änderungen an den öffentlichen Schnittstellen keine Breaking-Changes eingeführt werden, bei gleichzeitiger Wahrung der Rückwärtskompatibilität für alle aktiv unterstützten Releases. [<=]

5.24.2 Anforderungen an Hersteller eines TI 2.0-Dienstes

A_27818 -Unterstützung der Wartbarkeit des ZETA Guard-Dienstes

Der Hersteller eines Dienstes der TI2.0 MUSS regelmäßige Updates seines Produktes einplanen, damit die Aktualisierung der ZETA Guard gewährleistet ist. Ein Regelupdate erfolgt maximal einmal pro Quartal. Diese Anforderung gilt über den gesamten Lebenszyklus des Produktes hinweg. [<=]

5.24.3 Anforderungen an Anbieter eines TI 2.0-Dienstes

A_27792 -ZETA Guard - Verbot der Nutzung bestimmter ZETA Guard Versionen

Der Anbieter eines Dienstes der TI2.0 DARF eine zurückgezogene oder ungültige ZETA Guard Version NICHT produktiv einsetzen. [<=]

A_27793 -ZETA Guard - Reguläre Aktualisierung von ZETA Guard

Der Anbieter eines Dienstes der TI2.0 MUSS in der Lage sein, regelmäßig Patchupdates der integrierten ZETA Guard Version, die keine Auswirkung auf das Zusammenspiel mit dem Ressource Server haben, durchzuführen. Ein Regelupdate erfolgt maximal einmal pro Quartal. [<=]

A_27794 -ZETA Guard - Prüfung auf neue ZETA Guard Versionen

Der Anbieter eines Dienstes der TI2.0 MUSS regelmäßig auf neue freigegebene ZETA Guard Versionen prüfen und - wenn vorhanden - Aktualisierungen im Rahmen des Gültigkeitszeitraums einplanen. Ein Regelupdate erfolgt maximal einmal pro Quartal. [<=]

A_27795-01 -ZETA Guard - Gewährleistung der Verbindung zur ZETA Artifact Registry

Der Anbieter eines Dienstes der TI 2.0 MUSS gewährleisten, dass der eingesetzte ZETA Guard jederzeit Aktualisierungen der PIP-Daten und PAP-Policies sowie der Provisioning Daten über die lokale Artifact Registry von der ZETA Artifact Registry abrufen kann. [<=]

Hinweis: Der ausfallsichere Betrieb der lokalen Artifact Registry und die Verbindung zur ZETA Artifact Registry sind vom Anbieter sicherzustellen.

A_27796 -ZETA Guard - Gewährleistung der Verbindung zur Telemetriedatenlieferung der gematik

Der Anbieter eines Dienstes der TI2.0 MUSS gewährleisten, dass der eingesetzte ZETA Guard jederzeit Datenlieferungen an die gematik übermitteln kann. [<=]

Hinweis: Notwendige Freischaltungen sind vom Anbieter zu beauftragen und deren Funktionsfähigkeit sicherzustellen.

A_25773-02 -ZETA Guard - Nutzung der von der gematik bereitgestellten Container Images

Der Anbieter eines Dienstes der TI 2.0 MUSS die von der gematik bereitgestellten Container Images im ZETA Guard verwenden, um den Zugang zum TI 2.0 Dienst zu kontrollieren. [≤]

Hinweis: Ausgenommen sind die austauschbaren ZETA Guard Komponenten.

5.24.4 Anforderungen für nahtlose Aktualisierungen

Es ist durch geeignete Maßnahmen sicherzustellen, dass ein unterbrechungsfreier Betrieb, bzw. die durchgängige Verfügbarkeit zu jeder Zeit gewährleistet ist.

A_25784 -ZETA Guard-Komponenten - Download von Aktualisierungen im Hintergrund

Die Komponenten des ZETA Guard MÜSSEN in der Lage sein, Aktualisierungen im Hintergrund herunterzuladen, ohne den laufenden Betrieb zu beeinträchtigen. [≤]

A_25785 -ZETA Guard-Komponenten - Nahtloser Übergang zu neuen Versionen

Die Komponenten des ZETA Guard MUSS einen Mechanismus bieten, der einen nahtlosen Übergang zu neuen Versionen oder Patches ermöglicht, ohne die Verfügbarkeit für Endnutzer zu unterbrechen. [≤]

A_25786 -ZETA Guard-Komponenten - Abschluss von Transaktionen vor Aktualisierung

Die Komponenten des ZETA Guard MUSS sicherstellen, dass alle aktuellen Transaktionen und Anfragen abgeschlossen oder ordnungsgemäß übernommen werden, bevor ein Update finalisiert wird. [≤]

A_25787 -ZETA Guard-Komponenten - Gewährleistung der Systemintegrität während Aktualisierungen

Die Komponenten des ZETA Guard MUSS während des gesamten Aktualisierungsprozesses die Systemintegrität und Sicherheitsrichtlinien aufrechterhalten. [≤]

A_25788 -ZETA Guard-Komponenten - Unterstützung von Rollbacks

Die Komponenten des ZETA Guard MUSS die Fähigkeit besitzen, zu einer stabilen Vorversion zurückzukehren, sollte eine Aktualisierung fehlerhaft sein oder abgebrochen werden müssen. [≤]

A_25789 -ZETA Guard-Komponenten - Schnelle Rollback-Durchführung

Die Komponenten des ZETA Guard MUSS Rollbacks schnell und ohne manuelle Eingriffe durchführen können. [≤]

5.24.5 Überwachung des Betriebsstatus

Um die Verfügbarkeit des ZETA-Guard sicherzustellen, führt die gematik ein externes Probing durch, das von außen den allgemeinen Erreichbarkeits- und Betriebsstatus überprüft. Dies erfolgt im Rahmen der überwachten TI 2.0-Dienstes, die eine konkrete ZETA Guard Instanz einsetzen.

Dem Anbieter von TI 2.0-Dienstes wird empfohlen, ergänzend dazu geeignete technische Maßnahmen aufzusetzen, um interne Self-Checks der ZETA-Guard Komponenten durchzuführen sowie Mechanismen zur Selbstüberwachung und -reparatur zu etablieren. Dies umfasst insbesondere automatisierte Verfahren zur Prüfung zentraler Funktionspfade, der Erreichbarkeit benötigter Abhängigkeiten sowie der Integrität interner Zustände.

Durch solche Self-Checks können potenzielle Störungen frühzeitig erkannt, lokalisiert und häufig sogar ohne manuelle Eingriffe behoben werden. Dies maximiert die Betriebsstabilität, minimiert Ausfallzeiten und stärkt die Resilienz des Gesamtsystems.

Bei einer Bereitstellung in Kubernetes-Umgebungen stehen hierfür erprobte Mechanismen zur Verfügung. Beispiele umfassen:

- **Liveness-Probes**(z. B.): erkennt Hänger oder Deadlocks und startet Pods automatisch neu.
- **Readiness-Probes**(z. B.): signalisiert, ob eine Komponente aktuell Anfragen zuverlässig bedienen kann; Pods werden erst nach erfolgreichem Probe-Ergebnis in den Service-Traffic aufgenommen.
- **Startup-Probes**: eignen sich für Komponenten mit längerer Initialisierungsphase, um sicherzustellen, dass der Pod nicht fälschlicherweise als „defekt“ betrachtet wird.
- **Self-Healing-Mechanismen**durch Kubernetes (z. B. RestartPolicy, ReplicaSets): ermöglichen automatische Wiederherstellung bei Ausfällen einzelner Komponenten, ohne dass ein manuelles Eingreifen nötig ist.
- **Resource-Überwachung**mittels Kubernetes-Metriken (CPU, RAM, Netzwerk): kann frühzeitig Engpässe und Anomalien im Ressourcenverbrauch sichtbar machen.

A_27385-01 -ZETA Guard-Komponenten - Abbildung von Produkt- und Konfigurationsversionen

Der ZETA Guard MUSS als Subkomponente eines TI 2.0-Fachdienstes folgende Versionsangaben abrufbar vorhalten:

- ZETA-Guard Modulversion gem.Semantic Versioning (Major-Minor-Patch)
- Konfigurationsversion gem. [gemKPT_Betr#A_20219-01] des ZETA-Guard

[<=]

A_27496 -Zeta Guard-Komponenten - Aufbereitung von Client Daten zum Monitoring

Die Komponenten des ZETA Guard MÜSSEN zu jedem Schnittstellenaufruf an den Resource-Server mindestens folgende Informationen aus der PDP Datenbank - und dem Request des Aufrufers verarbeiten und protokollieren, damit eine anschließende Zusammenführung von Monitoring-Daten aus verschiedenen Quellen (siehe [A_27494*]) im Telemetriedaten-Service ermöglicht werden kann.

Diese Daten umfassen mindestens:

- product_id - aus dem Token Self-Assessment des aufrufenden Clientsystems
- product_version - aus demToken Self-Assessment des aufrufenden Clientsystems
- professionOID - aus dem SM-B Zertifikat des aufrufenden Clientsystems

[<=]

5.24.6 Leistungs-Anforderungen

In diesem Kapitel werden die Anforderungen an Performance, Skalierbarkeit und Last an die ZETA-Guard Komponenten zusammengefasst. Die zugrunde liegenden Messungen dienen der Verifikation der Leistungsfähigkeit der Komponenten des ZETA-Guard. Da der ZETA-Guard in verschiedenen Szenarien genutzt werden kann, werden die Messungen einheitlich direkt am Eingang zum PEP bzw. PDP gemessen und durch das Service-Mesh vorgenommen. Die Latenzmessungen müssen über die Differenz zwischen Antwortzeiten beim Eingang in den PEP und dem Ausgang aus dem PEP gebildet werden.

Die geforderten Antwortzeiten bzw. Latenzen werden dabei in Perzentilen vorgegeben. Zur Methode der Perzentile siehe u.a. auch [gemSpec_Perf, Kapitel 2.1] zur Bearbeitungszeit. Dort werden grundsätzlich Mittelwerte der Bearbeitungszeiten sowie 99%-Quantile vorgegeben.

A_26486-01 -PEP HTTP Proxy - Bearbeitungszeiten

Die Komponente PEP HTTP Proxy MUSS die Bearbeitungszeitvorgaben unter Last aus der Tabelle "Tab_gemSpec_ZETA_PEP_HTTP_Proxy: Bearbeitungszeitvorgaben" erfüllen.

Die Bearbeitungszeiten lassen sich in Latenzen und Antwortzeiten unterscheiden.

PEP Latenzen sind die Zeiten, die der PEP benötigt, um die Anfragen an den Fachdienst weiterzuleiten, bzw. die Response eben zurück durchzuleiten. Die Benennung als Latenz erfolgt aufgrund der Tatsache, dass der PEP hier Requests nur durchleitet, die eigentliche Beantwortung erfolgt durch den Fachdienst.

PEP Antwortzeiten beziehen sich dabei dann auf den Austausch der Nachrichten 1-4 des ASL Protokolls, die der PEP selbst beantwortet.

Tabelle 24: Tab_gemSpec_ZETA_PEP_HTTP_Proxy: Bearbeitungszeitvorgaben

Request-Typ	Messung-Typ	Mittelwert	90%	95%	99%
Request an Fachdienst ohne ASL	Latenz	75ms	100ms	150ms	1s
Request an Fachdienst mit ASL	Latenz	75ms	100ms	150ms	1s
Austausch der ASL-Nachrichten 1-4	Antwortzeit	75ms	100ms	150ms	1s
Ausliefern der .well-known (1)	Antwortzeit	7,5ms	10ms	15ms	100ms

(1) dies kann durch Caching mitigiert werden

[<=]

A_26487 -PEP HTTP Proxy -Skalierbarkeit

Die Komponente PEP HTTP Proxy MUSS horizontal skalierbar sein, sodass mit jedem zusätzlichen Pod mindestens 75% der Leistung eines einzigen Pods verfügbar werden.

[<=]

A_26488 -PEP HTTP Proxy - Last

Die Komponente PEP HTTP Proxy MUSS pro Pod mehr als 300 Websocket Verbindungen und mehr als 300 Requests pro Sekunde unterstützen können.[<=]

8862A_26489-012 -PDP Authorization Server - Bearbeitungszeiten

Die Komponente PDP Authorization Server MUSS die Bearbeitungszeitvorgaben unter Last aus der Tabelle "Tab_gemSpec_ZETA_PDP_Auth_Server: Bearbeitungszeitvorgaben" erfüllen.

Für den PDP gibt es verschiedene Typen von Requests, die hier analog zum PEP in Perzentilen dargestellt werden. Alle Typen von Requests sind Antwortzeiten, daher ist hier kein besonderer Messung-Typ aufgeführt.

Tabelle 25: Tab_gemSpec_ZETA_PDP_Auth_Server: Bearbeitungszeitvorgaben

Request-Typ	Mittelwert	90%	95%	99%
/nonce	33ms	50ms	75ms	500ms
/register	75ms	100ms	150ms	1s

/token	75ms	100ms	150ms	1s
/refreshvoke	75ms	100ms	150ms	1s

[<=]

A_26490 -PDP Authorization Server - Skalierbarkeit

Die Komponente PDP Authorization Server MUSS horizontal skalierbar sein, sodass mit jedem zusätzlichen Pod mindestens 75% der Leistung eines einzigen Pods verfügbar werden.[<=]

A_26491 -PDP Authorization Server - Last

Die Komponente PDP Authorization Server MUSS pro Pod über alle Endpunkte zusammen mehr als 300 Requests pro Sekunde unterstützen können.[<=]

Hinweis: Anfragen an abhängige Dienste im Hintergrund, um einen Request vollständig zu bearbeiten (z.B. OCSP), werden bei den Bearbeitungszeiten mit berücksichtigt, jedoch nicht dem abfragenden Dienst zur Last gelegt.

5.24.7 Betriebliche Schnittstellendefinition

Die Komponenten des ZETA Guard stellen Endpunkte zur Verfügung, um die grundlegende Funktionalität, eingebettet in einen Service, zu gewährleisten. Jeder Dienst, der die ZETA Guard_Komponenten betreibt, stellt damit folgende Endpunkte für einen Nutzer zur Verfügung.

A_28436 -ZETA Guard, Endpunkte im Internet

Der Anbieter eines TI 2.0 Dienstes MUSS die ZETA Guard und Kubernetes Cluster Endpunkte gemäß Tab_gemSpec_ZETA_Schnittstellendefinition_ZETA_Guard im Internet bereitstellen.[<=]

Tabelle 26: Tab_gemSpec_ZETA_Schnittstellendefinition_ZETA_Guard

Endpunkt / Anwendungsfall	Beschreibung
GET /.well-known/api-catalog	Abruf des Resource Server API Catalog Well-known JSON Dokuments (https://www.rfc-editor.org/rfc/rfc9727.html)
GET /.well-known/oauth-protected-resource/<resource>	Abruf des Resource Server Well-known JSON Dokuments. Default ist GET /.well-known/oauth-protected-resource/. Wenn mehrere Ressourcen vom Resource Server bereitgestellt werden, dann werden die Well-known Dokumente über den Subpath <resource> bereitgestellt. Beispiel:GET /.well-known/oauth-protected-resource/resource1 (https://www.rfc-editor.org/rfc/rfc9728.html)
GET /.well-known/oauth-authorization-server	Abruf des Autorisierungsserver Well-known JSON Dokuments
GET /.well-known/openid-federation HOST <ZETA Guard Authorization Server>	Abruf des Entity Statement Well-known nachOpenID Federation 1.0

GET /.well-known/openid-configuration HOST <ZETA Guard Authorization Server>	Abruf des OpenID Well-known JSON Dokuments. Der ZETA Guard Authorization Server stellt Subject Token für Workloads im ZETA Guard aus, für den Zugriff auf das SIEM und den Telemetriedaten Empfänger der gematik.
GET /openid/v1/jwks HOST <ZETA Guard Authorization Server>	JWKS des ZETA Guard Authorization Server Enthält die Signatur-Zertifikate des ZETA Guard Authorization Server
GET /openid/v1/jwks HOST <Kubernetes Cluster IDP>	JWKS des Kubernetes Cluster IDP Enthält die Signatur-Zertifikate des IDP <i>Hinweis: Die Bereitstellung erfolgt nicht durch de ZETA Guard, sondern durch den Kubernetes Cluster.</i>
GET /nonce	Nonce abrufen
POST /register	Dynamic Client Registration
GET /authorize	Für die Autorisierung nach OAuth Authorization Code Flow
POST /token	Es werden verschiedene Token Requests unterstützt: - Token Request mit Authorization Code - Token ExchangeRequest mit Subject Token - Token Exchange - Token Request mit Refresh Token
POST /revoke	Widerruf (Revocation) eines Refresh Token gemäß RFC 7009

Hinweis: In ZETA Stufe 1 werden die Endpunkte: GET /.well-known/api-catalog und GET /.well-known/openid-federation noch nicht bereitgestellt.

Zusätzlich wird die ZETA Artifact Registry bereitgestellt, die über die lokale Artifact Registry von ZETA Guard abgefragt wird, um beispielsweise aktualisierte Policy-Informationen abzuholen. Folgende Repositories sind in der ZETA Artifact Registry definiert.

Tabelle 27: Tab_gemSpec_ZETA_ZETA_Artifact_Registry_Repositories

ZETA Artifact Registry Repository	Beschreibung
europa-west3-docker.pkg.dev/gematik-pt-zeta-prod/zeta-policies/{application}:{label}	Abruf der Policy eines Dienstes
europa-west3-docker.pkg.dev/gematik-pt-zeta-prod/zeta-helm	ZETA Guard Helm Charts
europa-west3-docker.pkg.dev/gematik-pt-zeta-prod/zeta-dcr	ZETA Guard Container Images, ZETA Guard Provisioning Container Image und

5.24.8 Prozesse zur Inbetriebnahme eines ZETA Guard

Für den Betrieb eines ZETA Guard ist es erforderlich, dass ein Registrierungsprozess der gematik durchlaufen wird. In diesem Prozess werden Informationen über den Kubernetes Cluster und den Authorization Server des ZETA Guard bereitgestellt, die eine sichere Kommunikation mit Diensten der gematik (Artifact Registry, SIEM und Telemetriedaten Empfänger) und die Integration in den Federation Master der TI ermöglichen.

A_28437-01 -ZETA Guard, Registrierung bei der gematik

Der Anbieter des TI 2.0 Dienstes MUSS den ZETA Guard Authorization Server und den Issuer des Kubernetes Cluster IDP (oder alternativ einen JWK, der für die Signatur von Subject Token im Workload Identity Federation Authorization Flow verwendet wird) bei der gematik registrieren. [≤]

5.25 Anforderungen an Dienste der TI

Dienste der TI (Resource Server), die durch einen ZETA Guard geschützt sind, erhalten nur Requests von Clients oder anderen Diensten, wenn der PDP des ZETA Guard den Zugriff gewährt und ein Access Token ausgestellt hat. Der PEP des ZETA Guards setzt durch, dass nur Requests mit gültigem Access Token zum Resource Server gelangen.

5.26 Sicherheitsleistungen des ZETA Guard für Resource Server

Der ZETA Guard bietet umfassende Sicherheitsleistungen für Resource Server in der TI 2.0, indem er das Zero Trust-Paradigma umsetzt. Seine wichtigsten Sicherheitsleistungen lassen sich wie folgt zusammenfassen:

- **Zugriffskontrolle:** Der Policy Enforcement Point (PEP) fungiert als HTTP Proxy und kontrolliert den gesamten Datenverkehr zwischen Client-Anwendungen und dem Resource Server. Er lässt nur Anfragen mit gültigem Access Token durch, das vom Policy Decision Point (PDP) ausgestellt wurde. Zusätzliche Prüfungen, gesteuert durch Attribute im Access Token, können integriert werden. Optional kann konfiguriert werden, dass im Request Header PoPP ein PoPP Token vorhanden sein muss.
- **Client-Registrierung:** Der ZETA Guard setzt eine sichere Client-Registrierung durch, bei der Clients durch verschiedene Mechanismen attestiert werden (Android Key and ID Attestation, Apple DCAppAttest, TPM Attestation und Software Attestation). Der Client wird in der Regel an eine TI-Identität gebunden (gID, SM(C)-B oder HBA).
- **Autorisierung und Authentifizierung:** Der ZETA Guard OAuth2 Authorization Server steuert die Authentifizierung des Nutzers. Die Entscheidung zur Ausstellung des Access Token, nach erfolgreicher Client-Registrierung und Authentifizierung, wird durch die ZETA Guard Policy Engine basierend auf definierten Richtlinien (Policies) getroffen. Dies beinhaltet die Überprüfung von Nutzer- und Client-Registrierungsdaten inkl. Client-Attestierung. Unterstützte Authentifizierungsverfahren sind Token Exchange mit SM(C)-B signiertem Subject-Token sowie Authentifizierung über sektorale IDPs mit OIDC Flow.

- **Policy-Based-Access-Control:** Der Policy Information Point (PIP) und der Policy Administration Point (PAP) verwalten und liefern die freigegebenen Policies an die ZETA Guard Policy Engine. Die Policies sind maschinenlesbar und definieren die Zugriffsregeln, nach denen die Entscheidung zur Ausstellung von Access und Refresh Token erfolgt. Die Integrität und Authentizität der Policies wird durch Signaturen sichergestellt.
- **Schutz vor Token-Theft:** Durch den Einsatz von DPoP wird verhindert, dass gestohlene Access und Refresh Token durch Angreifer verwendet werden können, um Zugriff auf den Resource Server zu erhalten.
- **TLS Transportverschlüsselung:** Alle Komponenten des ZETA Guards stellen die Vertraulichkeit und Integrität der transportierten Daten sicher, indem sie TLS an allen Endpunkten verwenden und clientseitig mTLS unterstützen.
- **Mehrschichtige Transport-Sicherheit:** ZETA/ASL bietet neben TLS eine zweite Sicherungsschicht beim Transport der Daten vom Client zum ZETA Guard, um Schwachstellen in der TLS-Schicht abzufangen. Dies schützt vor bekannten und zukünftigen Schwachstellen in TLS-Protokoll und -Implementierungen. Der Betrieb der Anwendung kann fortgeführt werden, selbst wenn Schwachstellen im TLS-Protokoll entdeckt werden. Diese Funktion ist optional nutzbar.
Datenverschlüsselung: Die ZETA Guard Daten (ZETA/ASL-Daten, Session-, User- und Client-Daten) werden bei der Persistenz zusätzlich verschlüsselt (SymK.DB.Enc). Der ZETA Guard kann so konfiguriert werden, dass private Schlüssel für die Datenverschlüsselung mit einem Key Encryption Key geschützt werden, der in einem Hardware Security Module (HSM) gespeichert ist.
- **VAU Unterstützung:** Der ZETA Guard kann optional in einer Vertrauenswürdigem Ausführungsumgebung (VAU) ausgeführt werden.
- **Monitoring und Logging:** Der ZETA Guard sammelt Telemetriedaten und sicherheitsrelevante Ereignisse von PEP und PDP, um die Sicherheit kontinuierlich zu überwachen. Dies beinhaltet die Erkennung von Anomalien und potenziellen Bedrohungen. ZETA Guard sendet fachdienstspezifische Telemetriedaten und sicherheitsrelevante Ereignisse an die gematik (TI SIEM und gematik Telemetriedaten-Schnittstelle) sowie an den Anbieter des TI 2.0 Dienstes in einer anonymisierten Form, um eine Profilbildung zu verhindern.
- **Sicherheits- und Produktgutachten:** Für die ZETA Guard Implementierung werden ein Sicherheits- und ein Produktgutachten bereitgestellt.
- **Produkthandbuch sowie Sicherheits- und Datenschutzkonzept:** Im Produkthandbuch sind die Anforderungen an den Betrieb sowie die Konfiguration des ZETA Guard beschrieben. Im Sicherheits- und Datenschutzkonzept sind die Bedrohungen und umgesetzten Maßnahmen gegen Bedrohungen beschrieben. Dadurch wird für den Anbieter des TI 2.0 Dienstes erkennbar, welche zusätzlichen Sicherheitsleistungen zum Schutz des Dienstes und der Daten vom Anbieter erbracht werden müssen.

5.27 Weitere Leistungen des ZETA Guard für Resource Server

Der ZETA Guard übernimmt für Resource Server weitere Leistungen:

- **gematik Telemetriedaten Lieferung:** Der Telemetriedaten-Service sammelt von allen Komponenten des ZETA Guard Metriken, Traces und Logdaten und bereitet diese für den Versand an das Monitoring des TI 2.0 Dienst-Anbieters und an den gematik Telemetriedaten Empfänger in anonymisierter Form auf.

- **SIEM Daten Lieferung:** Der Telemetriedaten Service sammelt vom Monitoring des TI 2.0 Dienst-Anbieters SIEM Daten und leitet sie an das SIEM der gematik weiter.
- **Notification Service:** Der ZETA Guard implementiert eine API um Notification-Konfigurationen für Nutzer und ihre Clients zu speichern und um Notification-Events vom Resource Server entgegenzunehmen und an die Clientsystem Notification Services weiterzuleiten.
- **Protected Resource Metadata:** Über den HTTP Proxy können Clients das OAuth 2.0 Protected Resource Metadata Well-known JSON-Dokument ([RFC9728]) abfragen, um die notwendigen Informationen für die Interaktion mit diesem Resource Server zu erhalten.

6 Beispiele und Referenzimplementierungen

Die gematik stellt API-Spezifikationen und Proof-of-Concept-Implementierungen im Internet zur freien Verfügung.

Das Projekt <https://dsr.gematik.solutions> demonstriert eine Attestation mobiler Anwendungen auf gängigen mobilen Betriebssystemplattformen.

Im GitHub-Projekt [gemAPI_ZETA] werden die Schnittstellenspezifikationen der hier spezifizierten Zero Trust-Komponenten veröffentlicht.

7 Anhang A - Verzeichnisse

7.1 Abkürzungen

Tabelle 28: Im Dokument verwendete Abkürzungen

Kürzel	Erläuterung
API	Application Programming Interface
<u>ASL</u>	<u>Application Security Level</u>
CD	Continuous Delivery
CN	Common Name
<u>CTSDCR</u>	<u>Compatibility Test Suite</u> <u>Dynamic Client Registration</u>
<u>DEK</u>	<u>Data Encryption Key</u>
<u>DPoP</u>	<u>Demonstrating Proof-of-Possession</u>
DSR	Device Security Rating
eGK	elektronische Gesundheitskarte
ePA	elektronische Patientenakte
<u>FBEdV</u>	<u>File-Based-Encryption</u> <u>achdienst der Versicherten</u>
<u>FDE</u>	<u>Full-Disk-Encryption</u>
FIPS	Federal Information Processing Standards
GesundheitsID	Digitale Identität
HSM	Hardware Security Module
IDP	Identity Provider
IDS	Intrusion Detection System
ISMS	Informationssicherheitsmanagementsystem
JWT	JSON Web Token

k8s	Kubernetes
<u>KEK</u>	<u>Key Encryption Keys</u>
<u>KVNR</u>	<u>Krankenversicherungsnummer</u>
mTLS	Mutual Transport Layer Security
OCSP	Online Certificate Status Protocol
OIDC	OpenID Connect, https://de.wikipedia.org/wiki/OpenID_Connect
<u>OPA</u>	<u>Open Policy Agent</u>
OTLP	Open Telemetry Protocol
<u>OTP</u>	<u>One Time Password</u>
PAP	Policy Administration Point
PAR	Pushed Authorization Request
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
<u>PKCE</u>	<u>Proof Key for Code Exchange</u>
<u>PoPP</u>	<u>Proof of Patient Presence</u>
PU	Produktivumgebung
SAN	Subject Alternative Name
SIEM	Security Information and Event Management
SMC-B	Security Module Card Typ B, (Institutionskarte, Praxiskarte)
SPIFFE	Secure Production Identity Framework for Everyone
SPIRE	SPIFFE Runtime Environment
TI	Telematikinfrastruktur
<u>TI-HSMOFU</u>	<u>IT-Service-Management der IT-Trust On First Use</u>
TPM	Trusted Platform Module

VAU	Vertrauenswürdige Ausführungsumgebung
ZAS	ZETA Attestation Service
ZETA	Zero Trust Access

7.2 Glossar

Tabelle 29: Glossar der explizit im Dokument verwendeten Begriffe

Begriff	Erläuterung
Authorization Server	Ein Server, der Zugriffstoken ausgibt, nachdem er die Identität eines Nutzers authentifiziert und die Berechtigungen überprüft hat. In OAuth 2.0-basierten Systemen ist der Authorization Server eine zentrale Komponente zur Verwaltung von Zugriffsrechten.
Google Remote Procedure Call (gRPC)	Framework für die Kommunikation zwischen verteilten Systemen. Die Daten werden in einem effizienten binären Datenformat (Protocol Buffers alias Protobuf) serialisiert und per HTTP/2 übertragen. Es ermöglicht Anwendungen, welche direkt auf Funktionen anderer Anwendungen zuzugreifen, als wären diese lokal verfügbar, unabhängig von der zugrunde liegenden Plattform oder Programmiersprache.
Demonstrating Proof of Possession (DPoP)	Ein DPoP Token ist ein Sicherheitsmechanismus im OAuth 2.0-Protokoll, der den Besitz eines kryptografischen Schlüssels nachweist. Es stellt sicher, dass der Client, der ein Access Token verwendet, auch den zugehörigen privaten Schlüssel besitzt, um Missbrauch durch Dritte zu verhindern.
Identity and Access Management (IAM)	Prozesse und Technologien zur Verwaltung digitaler Identitäten und deren Zugriff auf Unternehmensressourcen.
Management Service	Ein Dienst zur Verwaltung und Orchestrierung von ZETA Guard-Ressourcen, insbesondere in containerisierten Umgebungen wie Kubernetes. Er ermöglicht die Verwaltung von Services, Pods und anderen Ressourcen innerhalb Kubernetes, um die Verfügbarkeit, Skalierbarkeit und Sicherheit der Anwendungen zu gewährleisten.
Open Policy Agent (OPA)	Eine Open Source Policy Engine, die es ermöglicht, Richtlinien als Code zu definieren und durchzusetzen, um Entscheidungslogik zentralisiert und flexibel in verschiedensten Software-Systemen und Anwendungen zu implementieren.
Policy Administration Point (PAP)	Eine Komponente, die Sicherheitsrichtlinien erstellt, verwaltet und verteilt. Der PAP definiert und verwaltet die Richtlinien, die von der PDP Policy Engine bei der Entscheidungsfindung verwendet

	werden.
Policy Decision Point (PDP)	Der PDP trifft die Entscheidung, ob ein Access Token ausgestellt werden darf, basierend auf den definierten Richtlinien und Informationen über den Anfragenden und den Client.
Policy Enforcement Point (PEP)	Ein Punkt in einem Netzwerk, an dem Sicherheitsrichtlinien durchgesetzt werden. Der PEP überwacht und kontrolliert den Zugriff auf Ressourcen basierend auf den Entscheidungen, die vom Policy Decision Point (PDP) getroffen werden.
Policy Information Point (PIP)	Eine Quelle von Attributen oder Kontextinformationen, die für die Entscheidungsfindung des Policy Decision Point (PDP) erforderlich sind. Der PIP stellt die notwendigen Daten zur Verfügung, um Zugriffsanfragen entsprechend den festgelegten Richtlinien zu bewerten.
Security Information and Event Management (SIEM)	Technologien und Prozesse zur Sammlung, Analyse und Korrelation von Sicherheitsdaten aus verschiedenen Quellen, um Sicherheitsvorfälle zu erkennen und darauf zu reagieren.
Telemetriedaten	<p>Telemetriedaten sind Daten, die von entfernten oder verteilten Systemen, Geräten oder Anwendungen gesammelt und an ein zentrales System zur Überwachung, Analyse und Verwaltung übertragen werden. Im Kontext der IT spielen Telemetriedaten eine entscheidende Rolle bei der Überwachung und Sicherung von Netzwerken und Systemen.</p> <p>Merkmale und Arten von Telemetriedaten:</p> <ul style="list-style-type: none"> - System- und Leistungsmetriken: Informationen über die Leistung und den Zustand von Hardware und Software, wie CPU-Auslastung, Speichernutzung, Netzwerkbandbreite und Festplattenkapazität. - Nutzeraktivitätsdaten: Protokolle und Aufzeichnungen über Nutzeraktionen und -verhalten, einschließlich Anmeldungen, Dateizugriffe, Anwendungsnutzung und andere Interaktionen. - Sicherheitsereignisse: Daten über sicherheitsrelevante Vorfälle, wie fehlgeschlagene Anmeldeversuche, erkannte Malware, unerlaubte Zugriffsversuche und andere sicherheitsbezogene Anomalien. - Netzwerkverkehrsdaten: Informationen über den Datenfluss im Netzwerk, einschließlich IP-Adressen, Ports, Protokolle, Datenmengen und Verbindungen zwischen verschiedenen Systemen und Diensten. - Konfigurationsdaten: Details zu den aktuellen Einstellungen und Konfigurationen von Systemen und Anwendungen, einschließlich Softwareversionen, installierte Patches und Sicherheitsrichtlinien.
Telemetriedaten-Service	Ein Dienst, der Telemetriedaten sammelt, verarbeitet und analysiert. Telemetriedaten umfassen Informationen über die Nutzung, Leistung und Zustände von Systemen und Anwendungen. Der Dienst hilft dabei, Einblicke in das Verhalten und die Gesundheit der Infrastruktur zu gewinnen, um proaktive Maßnahmen zur Optimierung und Sicherheit zu ergreifen.

Zero Trust (ZT)	Ein Sicherheitskonzept, das davon ausgeht, dass keine Entität (intern oder extern) automatisch vertraut wird. Alle Zugriffsanfragen werden überprüft, unabhängig von ihrem Ursprung.
Zero Trust/Additional Application Security Layer (ZETA/ASL)	Eine auf HTTP basierende zusätzliche Verschlüsselung der Daten zwischen ZETA Client und ZETA Guard PEP <u>oder zwischen ZETA Client und Resource Server</u> . Die verschlüsselte Verbindung wird auch ZETA/ASL-Kanal genannt.

7.3 Abbildungsverzeichnis

Abbildung 1: NIST Zero Trust-Referenzarchitektur, Quelle [NIST_SP1800-35_FIG1].....	13
Abbildung 2: Abbildung Zero-Trust-Architektur der TI 2.0.....	14
Abbildung 3: Abb_ZETA_Zugriff_auf_geschützte_Ressource.....	79
Abbildung 4: Abb_Service_Discovery.....	80
Abbildung 5: Abb_Client_Registrierung.....	82
Abbildung 6: Abb_Authentifizierung_und_Autorisierung.....	84
Abbildung 7: Abb_ZETA-Guard-Dienst-zu-Dienst-Kommunikation.....	90
Abbildung 1: NIST Zero Trust-Referenzarchitektur, Quelle [NIST_SP1800-35_FIG1].....	20
Abbildung 2: Abb-ZETA-Architektur.....	21
Abbildung 3 : Abb-Attestierungsablauf-nach-Betriebssystem.....	63
Abbildung 4 : Abb-ZETA-Schlüsselgenerierung-Windows-und-Linux.....	64
Abbildung 5 Abb-ZETA-Client-Start-mit-TPM-und-ZAS.....	66
Abbildung 6 Abb-ZETA-Service-Discovery.....	67
Abbildung 7 Abb-ZETA-TPM-Attestation-Key.....	68
Abbildung 8 Abb-ZETA-SE-Attestation-Key.....	69
Abbildung 9 Abb-ZETA-DCR-für-stationäre-Clients.....	70
Abbildung 10 Abb-ZETA-Client-Statement-mit-TPM-Attestation.....	73
Abbildung 11 Abb-ZETA-Client-Statement-mit-Apple-AppAttest.....	74
Abbildung 12 Abb-ZETA-Token-Exchange-mit-Attestation.....	76
Abbildung 13 Abb-ZETA-Token-Request-mit-Refresh-Token.....	79
Abbildung 14 Abb-ZETA-Zugriff-auf-RS-mit-ASL.....	82
Abbildung 15 Abb-ZETA-Zugriff-auf-RS-ohne-ASL.....	84
Abbildung 16 Abb-ZETA-Schlüsselgenerierung-Android.....	86
Abbildung 17 Abb-ZETA-Schlüsselgenerierung-Apple.....	89
Abbildung 18 Abb-ZETA-DCR-für-mobile-Clients.....	91
Abbildung 19 Abb-ZETA-Client-Statement-mit-Android-Attestation.....	94

Abbildung 20 Abb-ZETA-Client-Statement-mit-Apple-AppAttest.....96

Abbildung 21 Abb-ZETA-OIDC-Authentifizierung-mobiler-Clients.....99

Abbildung 22 Abb-ZETA-OIDC-Authorization-Request-mit-äußerem-und-innerem-PAR...100

Abbildung 23 Abb-ZETA-OIDC-Nutzerauthentisierung.....102

Abbildung 24 Abb-ZETA-OIDC-Token-Bezug.....103

Abbildung 25 Abb-ZETA-OAuth-Client-Authentifizierung-ohne-Nutzer.....106

Abbildung 26 : Abb-ZETA-Dienst-zu-Dienst-Kommunikation.....109

Abbildung 27 Abb-ZETA-Dienst-zu-Dienst-Kommunikation-mit-ZG-Client.....112

Abbildung 28 Abb-ZETA-Token-Revocation.....115

Abbildung 29 - Ablauf Notification Versand.....195

7.4 Tabellenverzeichnis

| [Tabelle 1: Tab-ZETA-Guard-Keys.....28](#)

| [Tabelle 2: Tab-ZETA-Client-Keys.....31](#)

| [Tabelle 3: Tab-Gematik-Keys.....32](#)

| [Tabelle 4: Security-Telemetriedaten-PEP-und-PDP.....36](#)

| [Tabelle 5: Telemetriedaten-Policy-Entscheidungen.....37](#)

| [Tabelle 6: Telemetriedaten-Angriffserkennung.....41](#)

| [Tabelle 7: ZT_HTTP_Statuscodes.....52](#)

| [Tabelle 8: Tab-Client-Assertion-JWT.....55](#)

| [Tabelle 9: Tab-Client-Statement.....56](#)

| [Tabelle 10: Tab-Posture-TPM.....56](#)

| [Tabelle 11: Tab-Posture-Software.....57](#)

| [Tabelle 12: Tab-Posture-Apple.....57](#)

| [Tabelle 13: Tab-Posture-Android.....58](#)

| [Tabelle 14: PEP-HTTP-Proxy-Zusätzliche-HTTP-Header.....70](#)

| [Tabelle 15: zeta-user-info-Header.....71](#)

| [Tabelle 16: PDP-Authorization-Server-Plugin-Schnittstelle-Application-Authorization-Backend.....75](#)

| [Tabelle 17: SM\(C\)-B-Nutzer-Daten.....75](#)

| [Tabelle 18: id_token-Nutzer-Daten.....76](#)

| [Tabelle 19: Tab_gemSpec_ZETA_Telemetriedaten_HTTP_Statuscodes.....94](#)

| [Tabelle 20: Tab_gemSpec_ZETA_PEP_HTTP_Proxy: Bearbeitungszeitvorgaben.....100](#)

| [Tabelle 21: Tab_gemSpec_ZETA_PDP_Auth_Server: Bearbeitungszeitvorgaben.....101](#)

| [Tabelle 22: Tab_gemSpec_ZETA_Schnittstellendefinition_ZETA_Guard.....102](#)

| [Tabelle 23: Tab_gemSpec_ZETA_ZETA_Artifact_Registry_Repositories.....103](#)

Tabelle 24: Im Dokument verwendete Abkürzungen.....	107
Tabelle 25: Glossar der explizit im Dokument verwendeten Begriffe.....	108
Tabelle 26: Referenzierte Dokumente der gematik.....	112
Tabelle 27: Weitere Referenzen.....	114
Tabelle 1: Security Telemetriedaten PEP und PDP.....	44
Tabelle 2: Telemetriedaten Policy Entscheidungen.....	46
Tabelle 3: Telemetriedaten Angriffserkennung.....	49
Tabelle 4: Tab-ZETA-Guard-Keys.....	55
Tabelle 5: Tab-ZETA-Client-Keys.....	58
Tabelle 6: Tab-Gematik-Keys.....	59
Tabelle 7: ZT_HTTP_Statuscodes.....	127
Tabelle 8: Tab-Client-Assertion-JWT.....	133
Tabelle 9: Tab-Client-Statement.....	133
Tabelle 10: Tab-Posture-TPM.....	134
Tabelle 11: Tab-Posture-Software.....	134
Tabelle 12: Tab-Posture-Apple.....	135
Tabelle 13: Tab-Posture-Android.....	136
Tabelle 14: PEP HTTP Proxy - Zusätzliche HTTP-Header.....	161
Tabelle 15: zeta-user-info-Header.....	163
Tabelle 16: PDP Authorization Server - Plugin-Schnittstelle Application Authorization Backend.....	168
Tabelle 17: SM(C)-B_Nutzer-Daten.....	169
Tabelle 18: id_token_Nutzer-Daten.....	170
Tabelle 19 Tab-ZETA-Token-Ausstellung-ASL-am-Resource-Server.....	176
Tabelle 20: Tab_gemSpec_ZETA_Telemetriedaten_HTTP_Statuscodes.....	190
Tabelle 21: Tab_gemSpec_ZETA_PEP_HTTP_Proxy: Bearbeitungszeitvorgaben.....	210
Tabelle 22: Tab_gemSpec_ZETA_PDP_Auth_Server: Bearbeitungszeitvorgaben.....	210
Tabelle 23: Tab_gemSpec_ZETA_Schnittstellendefinition_ZETA_Guard.....	211
Tabelle 24: Tab_gemSpec_ZETA_ZETA_Artifact_Registry_Repositories.....	212
Tabelle 25: Im Dokument verwendete Abkürzungen.....	217
Tabelle 26: Glossar der explizit im Dokument verwendeten Begriffe.....	219
Tabelle 27: Referenzierte Dokumente der gematik.....	224
Tabelle 28: Weitere Referenzen.....	227

7.5 Referenzierte Dokumente

7.5.1 Dokumente der gematik

Die nachfolgende Tabelle enthält die Bezeichnung der in dem vorliegenden Dokument referenzierten Dokumente der gematik zur Telematikinfrastruktur.

Tabelle 30: Referenzierte Dokumente der gematik

posture-android.yaml [Quelle]	Herausgeber: Titel
[access-token.yaml]	Schema access-token.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/access-token.yaml
[API-ZETA-Attestation-Service]	API des ZETA Attestation Service https://github.com/gematik/ZETA/blob/main/docs/api/v1/index.md#zeta-attestation-service-endpunkte
[as-well-known.yaml]	Schema für das OAuth Authorization Server Well-known JSON Dokument https://raw.githubusercontent.com/gematik/zeta/main/src/schemas/as-well-known.yaml
[client-assertion-jwt.yaml]	Schema client-assertion-jwt https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/client-assertion-jwt.yaml
[client-data.yaml]	Schema client-data.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/client-data.yaml
[client-statement.yaml]	Schema client-statement.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/client-statement.yaml
[dcr-request.yaml]	Schema dcrrequest.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/dcr-request.yaml
[federation-master.yaml]	Schemafederation-master.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/federation-master.yaml
[gemAPI_ZETA]	gematik: OpenAPI Schnittstellen- und Schemaspezifikation ZETA https://github.com/gematik/zeta
[gemGlossar]	gematik: Glossar der Telematikinfrastruktur

[gemF_PushNotification]	gematik: Feature: Anwendungsübergreifende Push Notification https://gemspec.gematik.de/docs/gemF/gemF_PushNotification/latest/
[gemKPT_Betr]	gematik: Betriebskonzept Online-Produktivbetrieb https://gemspec.gematik.de/docs/gemKPT/gemKPT_Betr/latest/
[gemSpec_DS_Hersteller]	gematik: Spezifikation Datenschutz- u. Sicherheitsanforderungen der TI an Hersteller https://gemspec.gematik.de/docs/gemSpec/gemSpec_DS_Hersteller/latest/
[gemSpec_IDP_Sek]	gematik: Spezifikation Sektoraler Identity Provider https://gemspec.gematik.de/docs/gemSpec/gemSpec_IDP_Sek/latest/
[gemSpec_Krypt]	gematik: Übergreifende Spezifikation Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/
[gemSpec_OID]	Spezifikation Festlegung von OIDs https://gemspec.gematik.de/docs/gemSpec/gemSpec_OID/latest/
[gemSpec_OM]	gematik: Übergreifende Spezifikation Operations und Maintenance https://gemspec.gematik.de/docs/gemSpec/gemSpec_OM/latest/
[gemSpec_Perf]	gematik: Übergreifende Spezifikation Performance und Mengengerüst TI-Plattform https://gemspec.gematik.de/docs/gemSpec/gemSpec_Perf/latest/
[gemSpec_PKI]	Übergreifende Spezifikation Spezifikation PKI https://gemspec.gematik.de/docs/gemSpec/gemSpec_PKI/latest/
[GitHub ZETA Schemas]	Schemas für zusätzliche HTTP-Header https://github.com/gematik/zeta/tree/main/src/schemas
[hsm-proxy.proto]	Protobuf Spezifikation der HSM Proxy Schnittstelle für Komponenten https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/gRPC/hsm-proxy.proto
[ISMS]	Spezifikation Datenschutz- und Sicherheitsanforderungen der TI an Anbieter (Abschnitt 3.3) https://gemspec.gematik.de/docs/gemSpec/gemSpec_DS_Anbieter/latest/#3.3
[JOSEnotification-]	JSON-Object-Signing-and-EncrypOpenApi Spezifikation der

[service-rs-endpoint]	Schnittstelle zwischen ZETA Guard Notification (JOSE)Service und Resource Server https://www.iana.org/assignments/jose/jose.xhtml#raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/openapi/notification-service-rs-endpoint.yaml
[opr-well-known.yaml]	Schema für das OAuth Protected Resource Metadata Well-known JSON Dokument https://raw.githubusercontent.com/gematik/zeta/main/src/schemas/opr-well-known.yaml
[pdp-decision.yaml]	Schema pdp-decision.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/pdp-decision.yaml
[policy-engine-client-data.yaml]	Schema policy-engine-client-data.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/policy-engine-client-data.yaml
[policy-engine-input.yaml]	Schemapolicy-engine-input.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/policy-engine-input.yaml
[posture-android.yaml]	Schemaposture-android.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/posture-android.yaml
[posture-apple.yaml]	Schemaposture-apple.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/posture-apple.yaml
[posture-software.yaml]	Schema posture-software.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/posture-software.yaml
[posture-tpm.yaml]	Schema posture-tpm.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/posture-tpm.yaml
[posture.yaml]	Schema posture.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/posture.yaml
[dcr-response-202.yaml]	Schema dcr-response-202.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/dcr-response-202.yaml
[TrustedTPM_RootCA]	Liste der vertrauenswürdigen TPM Stamm- und SubCA-Zertifikate https://learn.microsoft.com/de-at/windows-server/security/guarded-fabric-shielded-vm/guarded-fabric-install-trusted-tpm-root-certificates

[subject-token-smb.yaml]	Schema subject-token-smb.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/subject-token-smb.yaml
[token-response.yaml]	Schema token-response.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/token-response.yaml
[verify-request.yaml]	Schema verify-request.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/verify-request.yaml
[zeta-attestation-token.yaml]	Schemazeta-attestation-token.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/zeta-attestation-token.yaml
[zeta-user-info.yaml]	Schema zeta-user-info.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/zeta-user-info.yaml
[zeta-error.yaml]	Schema zeta-error.yaml https://raw.githubusercontent.com/gematik/zeta/main/src/schemas/zeta-error.yaml
[zeta-guard-client-management]	OpenAPI Spezifikation der ZETA Guard Client Management API https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/openapi/zeta-guard-client-management.yaml
[zeta-guard-admin-oob]	OpenAPI Spezifikation der ZETA Guard Out Of Band Admin API https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/openapi/zeta-guard-admin-oob.yaml

7.5.2 Weitere Referenzen

Tabelle 31: Weitere Referenzen

[Quelle]	Herausgeber (Erscheinungsdatum) Titel
[Android Platform Security Model]	The Android Platform Security Model (2023) https://research.google/pubs/the-android-platform-security-model/ (Abruf 01/2025)
[Apple Platform Security Guide]	Einführung in die Sicherheit der Apple-Plattformen https://support.apple.com/de-de/guide/security/seccd5016d31/web (Abruf 01/2025)
[BSI-Grundschriftz]	IT-Grundschriftz - Informationssicherheit mit System https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschriftz/it-grundschriftz_node.html (Abruf 01/2025)

	01/2025)
[CAB-Forum]	Certification Authority Browser Forum (CA/Browser Forum) https://cabforum.org/ (Abruf 01/2025)
[CAPEC OWASP]	CAPEC: OWASP Related Patterns CAPEC - CAPEC-659: OWASP Related Patterns (Version 3.9) (mitre.org) (Abruf 01/2025)
[ExpBack]	Exponential Backoff https://en.wikipedia.org/wiki/Exponential_backoff (Abruf 01/2025)
[JOSE]	JSON Object Signing and Encryption (JOSE) https://www.iana.org/assignments/jose/jose.xhtml
[NIST_SP1800-35_FIG1]	National Institute of Standards and Technology (NIST) NIST SP 1800-35 Publication (Figure 1 - General ZTA Reference Architecture) https://pages.nist.gov/zero-trust-architecture/VolumeB/architecture.html (Abruf 01/2025)
[OPA Bundle]	Open Policy Agent, Bundles https://www.openpolicyagent.org/docs/latest/management-bundles/ (Abruf 01/2025)
[Open Policy Agent]	Open Policy Agent https://www.openpolicyagent.org/docs/latest/ (Abruf 01/2025)
[OWASP-Top-10-Risiken]	OWASP Top 10 https://owasp.org/www-project-top-ten/ (Abruf 01/2025)
[OWASP-Top-Ten-Kubernetes]	OWASP Kubernetes Top 10 https://owasp.org/www-project-kubernetes-top-ten/ (Abruf 04/2026)
[RFC2119]	Key words for use in RFCs to Indicate Requirement Levels https://datatracker.ietf.org/doc/html/rfc2119 (Abruf 01/2025)
[RFC2986]	PKCS #10: Certification Request Syntax Specification https://datatracker.ietf.org/doc/html/rfc2986 (Abruf 01/2025)
[RFC6066]	Transport Layer Security (TLS) Extensions: Extension Definitions https://datatracker.ietf.org/doc/html/rfc6066 (Abruf 01/2025)
[RFC6749]	The OAuth 2.0 Authorization Framework https://datatracker.ietf.org/doc/html/rfc6749 (Abruf 01/2025)
[RFC7009]	OAuth 2.0 Token Revocation https://www.rfc-editor.org/info/rfc7009/

[RFC7231]	Hypertext Transfer Protocol (HTTP/1.1) Semantics and Content https://datatracker.ietf.org/doc/html/rfc7231 (Abruf 01/2025)
[RFC7232]	Hypertext Transfer Protocol (HTTP/1.1) Conditional Requests https://datatracker.ietf.org/doc/html/rfc7232 (Abruf 01/2025)
[RFC7239]	Forwarded HTTP Extension https://datatracker.ietf.org/doc/html/rfc7239
[RFC7515]	JSON Web Signature (JWS) https://datatracker.ietf.org/doc/html/rfc7515
[RFC7519]	JSON Web Token (JWT) https://datatracker.ietf.org/doc/html/rfc7519
[RFC7521]	Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants https://datatracker.ietf.org/doc/html/rfc7521 (Abruf 01/2025)
[RFC7523]	JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants https://datatracker.ietf.org/doc/html/rfc7523 (Abruf 01/2025)
[RFC7591]	OAuth 2.0 Dynamic Client Registration Protocol https://datatracker.ietf.org/doc/html/rfc7591
[RFC7592]	OAuth 2.0 Dynamic Client Registration Management Protocol https://datatracker.ietf.org/doc/html/rfc7592
[RFC7636]	Proof Key for Code Exchange by OAuth Public Clients https://datatracker.ietf.org/doc/html/rfc7636 (Abruf 01/2025)
[RFC7638]	JSON Web Key (JWK) Thumbprint https://datatracker.ietf.org/doc/html/rfc7638
[RFC8414]	OAuth 2.0 Authorization Server Metadata https://datatracker.ietf.org/doc/html/rfc8414
[RFC8555]	Automatic Certificate Management Environment (ACME) https://datatracker.ietf.org/doc/html/rfc8555#section-6.5.1 (Abruf 01/2025)
[RFC8693]	OAuth 2.0 Token Exchange https://datatracker.ietf.org/doc/html/rfc8693
[RFC8705]	OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Token https://datatracker.ietf.org/doc/html/rfc8705 (Abruf 01/2025)
[RFC8707]	Resource Indicators for OAuth 2.0 https://www.rfc-editor.org/rfc/rfc8707.html

[RFC9068]	JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens https://www.rfc-editor.org/info/rfc9068/
[RFC9110]	HTTP Semantics https://datatracker.ietf.org/doc/html/rfc9110
[RFC9126]	OAuth 2.0 Pushed Authorization Request https://datatracker.ietf.org/doc/html/rfc9126
[RFC9334]	Remote ATtestation procedureS (RATS) Architecture https://www.rfc-editor.org/rfc/rfc9334.html
[RFC9449]	OAuth 2.0 Demonstrating Proof of Possession (DPoP) https://datatracker.ietf.org/doc/html/rfc9449 (Abruf 01/2025)
[RFC9727-470]	api-catalog: A Well-Known-URI and Link-Rel OAuth 2.0 Step Up Authentication to Help Discovery of APIs Challenge Protocol https://www.rfc-editor.org/doc/html/rfc9727.html 470
[RFC9728]	OAuth 2.0 Protected Resource Metadata https://www.rfc-editor.org/rfc/rfc9728.html
{session.yaml}	Schema session.yaml https://raw.githubusercontent.com/gematik/zeta/refs/heads/main/src/schemas/session.yaml (Abruf 06/2025)
[Shared Signals]	OpenID Shared Signals and Events Framework https://openid.net/specs/openid-sse-framework-1_0.html (Abruf 01/2025)
[SPIFFE und SPIRE]	Universal identity control plane for distributed systems https://spiffe.io/ (Abruf 01/2025)
[TR-03107-1]	BSI TR-03107 Elektronische Identitäten und Vertrauensdienste im E-Government https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03107/TR-03107-1.pdf (Abruf 01/2025)
[TR-03161]	BSI TR-03161 Anforderungen an Anwendungen im Gesundheitswesen https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03161/tr-03161.html (Abruf 01/2025)
[TR-03161-1]	Technische Richtlinie TR-03161: Anforderungen an Anwendungen im Gesundheitswesen Teil 1: Mobile Anwendungen; Version 3.0 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03161/BSI-TR-03161-1.pdf?__blob=publicationFile&v=13 (Abruf 01/2025)
[VerifiedBoot]	Verifizierter Start https://source.android.com/docs/security/features/verifiedboot?hl=de (Abruf 01/2025)

|
|