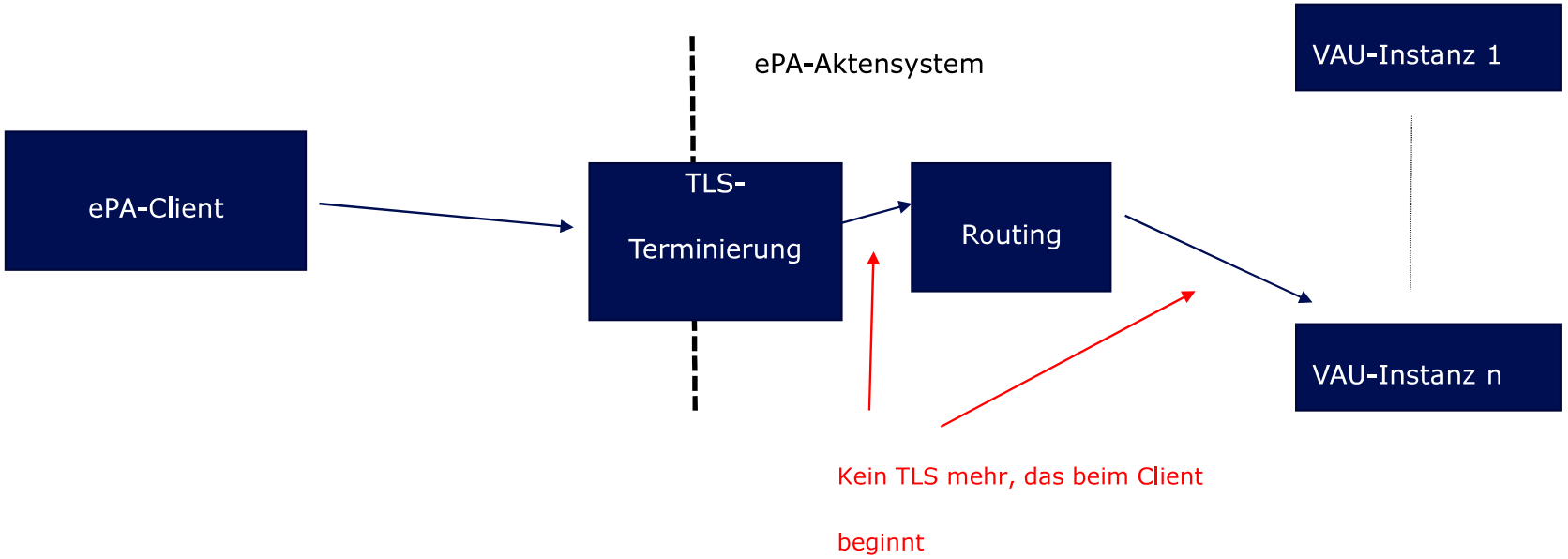
VAU-Protokoll: Ziele

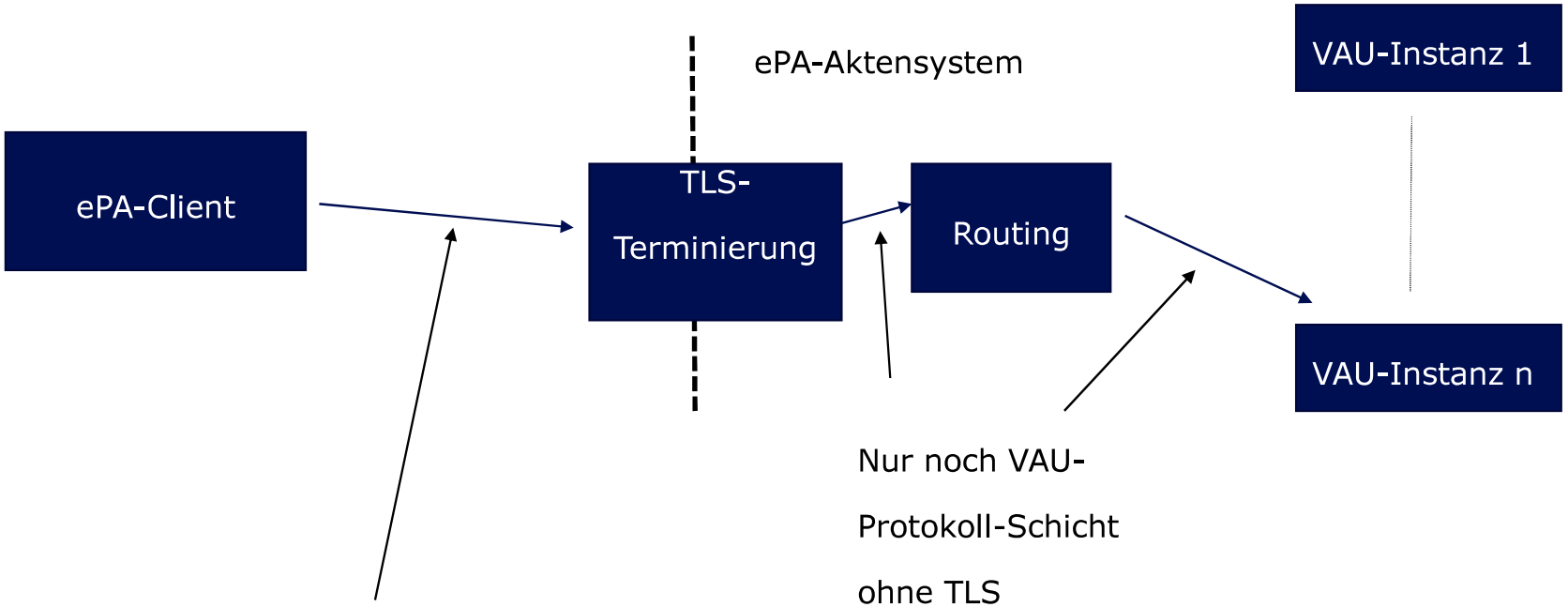
Ziel: E2E verschlüsselten und authentisierten Kanal zwischen Client (bspw. PVS) und VAU zu erhalten.

Problem:

Aus verschiedenen Gründen geht die TLS-Verbindung zum Aktensystem nicht direkt in die VAU (sowohl bei ePA als auch beim E-Rezept).



VAU-Protokoll: VAU-Kanal innerhalb von TLS



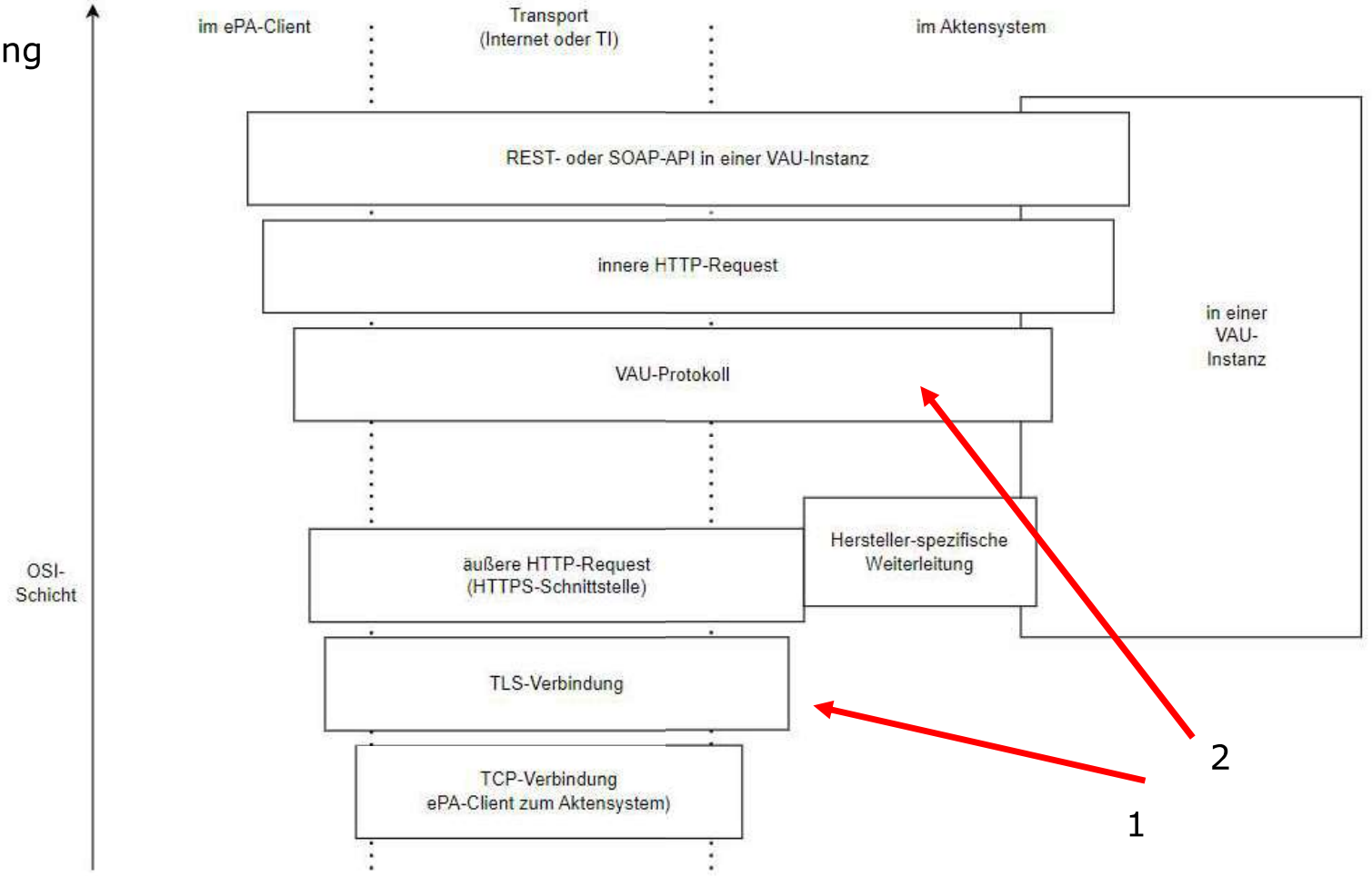
TLS + HTTP + Verschlüsselung

der HTTP-Inhalte über VAU-Protokoll (OSI-Schichtenmodell)

VAU-Protokoll: elektronische Patientenakte (ePA)

Transport-Verschlüsselung

1) TLS
2) VAU-Protokoll



VAU-Protokoll: Historische Entwicklung

VAU-Protokoll muss den strategischen Entscheidungen bei der Nutzerauthentisierungen folgen

(um Ziel beidseitig authentisierten Kanal zu erreichen)

2018	2020	2025
ePA 1.0 seit 2018 bis heute (ePA 2.6) VAU-Protokoll (ePA) unverändert, PKI-basierte Authentisierung über Signaturen der EE.	E-Rezept mit OIDC-Variante	ePA-für-alle VAU-Protokoll mit PKCE-OIDC

VAU-Protokoll: Leistungsmerkmale

- Ende-zu-Ende verschlüsselter und authentisierter Kanal zwischen ePA-Client und ePA-VAU
- Forward-Secrecy
- Quantum computing resistent
  (PQC-sicher, Hybridverfahren aus ECDH und Kyber768)

OAuth2/OIDC/PKCE (2/2)

Ganz allgemein

OIDC (egal ob TI oder

nicht)

1 Transport-Ebene

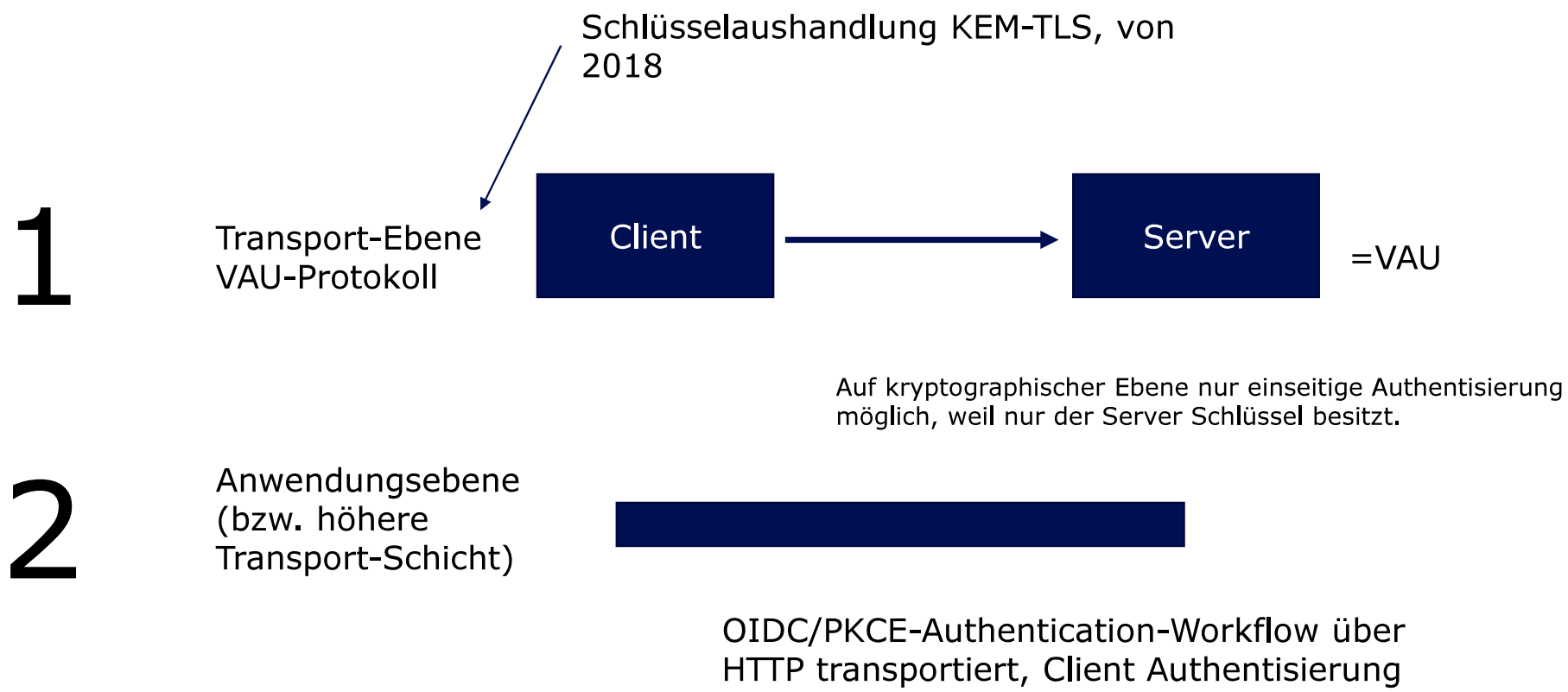
=VAU

Auf kryptographischer Ebene nur einseitige Authentisierung möglich, weil nur der Server Schlüssel besitzt.

2 Anwendungsebene (bzw. höhere Transport-Schicht)

OIDC/PKCE-Authentication-Workflow über HTTP transportiert, Client Authentisierung

VAU-Protokoll



VAU-Protokoll, Hybrid ECDH + Kyber768

1

Transport-Ebene

VAU-Protokoll

Schlüsselaushandlung KEM-TLS, von 2018

ECDH (Kurve P-256)

+

Kyber768

Kyber768X25519 Hybrid bei TLS zu Cloudflare (CDN)

ab Chrome-Version 116 (aktuell am 18.10.2023 ist 118)

https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html
https://bwesterb.github.io/draft-westerbaan-tls-xyber768d00/draft-tls-westerbaan-xyber768d00.html

Messenger Signal, Umstieg auf ein hybrides ECC+Kyber(PQC)-Verfahren

Ankündigung: https://signal.org/blog/pqxdh/ Spec: https://signal.org/docs/specifications/pqxdh/
https://www.heise.de/hintergrund/Signal-Chefin-ueber-Post-Quanten-Kryptographie-Zukunftsplaene-und-Co-9310463.html

VAU-Protokoll, 1

Schlüsselaushandlung KEM-TLS

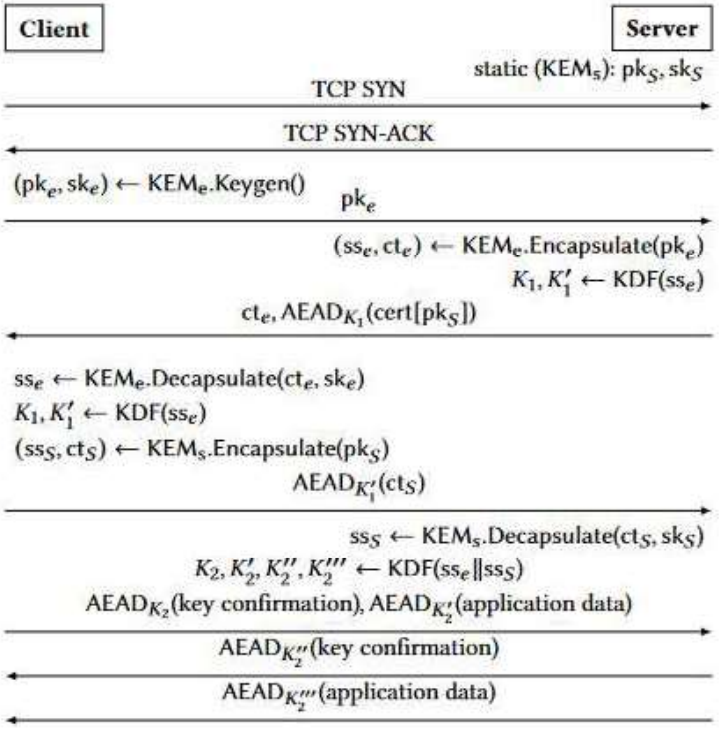


Abbildung 6: KEM-TLS Verbindungsaufbau [KEM-TLS]

- 4 Nachrichten (=> 2 round-trips)
- Bei Erfolg zwei symmetrische Kanalschlüssel ausgehandelt.
- Schlüssel 24 Stunden verwendbar.
- Nach Phase eins nur einseitige Auththentisierung (/VAU-Status A_25143)
- Nach Phase zwei (PKCE-OIDC): beseitige Authentisierung (/VAU-Status A_25143)
- VAU-Kanal unabhängig von äußerer TLS-Verbindung
- Nach zweiten Nachricht erzeugt die VAU eine VAU-Connection-ID (VAU-CID)
- (Nutzerpseudonym später Folie 16)

https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#A_25143

VAU-Protokoll, 1, Details

Spezifikation:
https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#7

Beispiel-Code für Aushandlung der symmetrischen VAU-Kanal-Schlüssel:
https://bitbucket.org/andreas_hallof/vau-protokoll/src/master/minimal/

Beispiel-Client für verschiedene Programmiersprachen (in Arbeit):
https://bitbucket.org/andreas_hallof/vau-protokoll/src/master/

Beispiel-Code für Java:
https://github.com/gematik/lib-vau

VAU-Protokoll, 1, Details

Beispiel-Code für Aushandlung der symmetrischen VAU-Kanal-Schlüssel: https://bitbucket.org/andreas_hallof/vau-protokoll/src/master/minimal/

```
a@t:~/git/vau-protokoll/minimal$ ./minimal.py
ic| 'Beginn Erzeugung von Nachricht 1'
ic| nachricht_1: {
                    "MessageType": "M1",
                    "ECDH_PK": {
                        "crv": "P-256",
                        "x": "(hexdump) 3f8ca628597dfec588399c43df0facb11cf180bc1eaebb35c8e910def051ef25",
                        "y": "(hexdump) db1eaf2e19c789beaf08376b3deb71c89b730912ab40192c37bcb5b4422f73d8"
                    },
                    "Kyber768_PK": "(hexdump) 65a2c1c380c6be1b2ee5e2629680a751b70475c49098a5c68a80.."
                }
ic| 'Beginn Erzeugung von Nachricht 2'
ic| ecdh_shared_secret: (hexdump) 57b201d42c1feae0c0bda5496a5c07aa1da724d6361db357cd499617a4643ba7
ic| Kyber768_shared_secret: (hexdump) afbb824230691fd5a7a2df7b21242a7ba4b26a6b36f2f70480e57a0864d25e23
ic| nachricht_2: {
                    "MessageType": "M2",
                    "ECDH_ct": {
                        "crv": "P-256",
                        "x": "(hexdump) d62b31b357c214028814cd16dddf99c5b2ea6c7d15aeb3a60a369977bd560a25",
                        "y": "(hexdump) 64e0c7b01e85fcbb266e5cf6295846b44b26d5c7ab7dccecfeaccf5094fda183"
                    },
                    "Kyber768_ct": "(hexdump) 7aecbf44d707c25450e4cca0428f1796991cf390f3cbf162476b…",
                    "AEAD_ct": "(hexdump) a5a0b5d1fec2c2ece723f1f7dc7e8de1fe00efef949dd32.."
                }
```

VAU-Protokoll, 1, Details

Beispiel-Code für Aushandlung der symmetrischen VAU-Kanal-Schlüssel: https://bitbucket.org/andreas_hallof/vau-protokoll/src/master/minimal/

```
ic| 'Beginn Erzeugung von Nachricht 3'
ic| ecdh_shared_secret: (hexdump) 57b201d42c1feae0c0bda5496a5c07aa1da724d6361db357cd499617a4643ba7
ic| shared_secret_client: (hexdump) afbb824230691fd5a7a2df7b21242a7ba4b26a6b36f2f70480e57a0864d25e23
ic| 'Schlüsselableitung für die K1-Schlüssel'
ic| ecdh_shared_secret: (hexdump) 5419f03f471b1a6f58af6c55b7acd1758fc61fb12c75ca35aeec9c0977433f50
ic| Kyber768_shared_secret: (hexdump) 64644d490bd1745dac172c03c2695472efb6d7a372c3ab1ee40140db006575ee
ic| 'Schlüsselableitung für die K2-Schlüssel'
ic| nachricht_3: {
                    "MessageType": "M3",
                    "AEAD_ct": "(hexdump) 362e5e5316a8871fef3173bee19ed1a22e7989…",
                    "AEAD_ct_key_confirmation": "(hexdump) d28b94ecb2afd542ec462fe28773bf23554a62fe3b…"
                }
ic| 'Beginn Erzeugung von Nachricht 4'
ic| ecdh_shared_secret: (hexdump) 5419f03f471b1a6f58af6c55b7acd1758fc61fb12c75ca35aeec9c0977433f50
ic| shared_secret_client: (hexdump) 64644d490bd1745dac172c03c2695472efb6d7a372c3ab1ee40140db006575ee
ic| nachricht_4: {
                    "MessageType": "M4",
                    "AEAD_ct_key_confirmation": "(hexdump) ae8a38a544880faf9fab4c37d113a2d05f8672473…"
                }
Time-Total für die Handshake-Phase: 0.19974207878112793
```

VAU-Protokoll, 1, bei etablierten VAU-Kanal

Innerer HTTP-Request:

```
GET /VAU-Status HTTP/1.1
Host: epa.aktensystem.ti
```

AES/GCM-Verschlüsselung mit K2_c2s_app_data

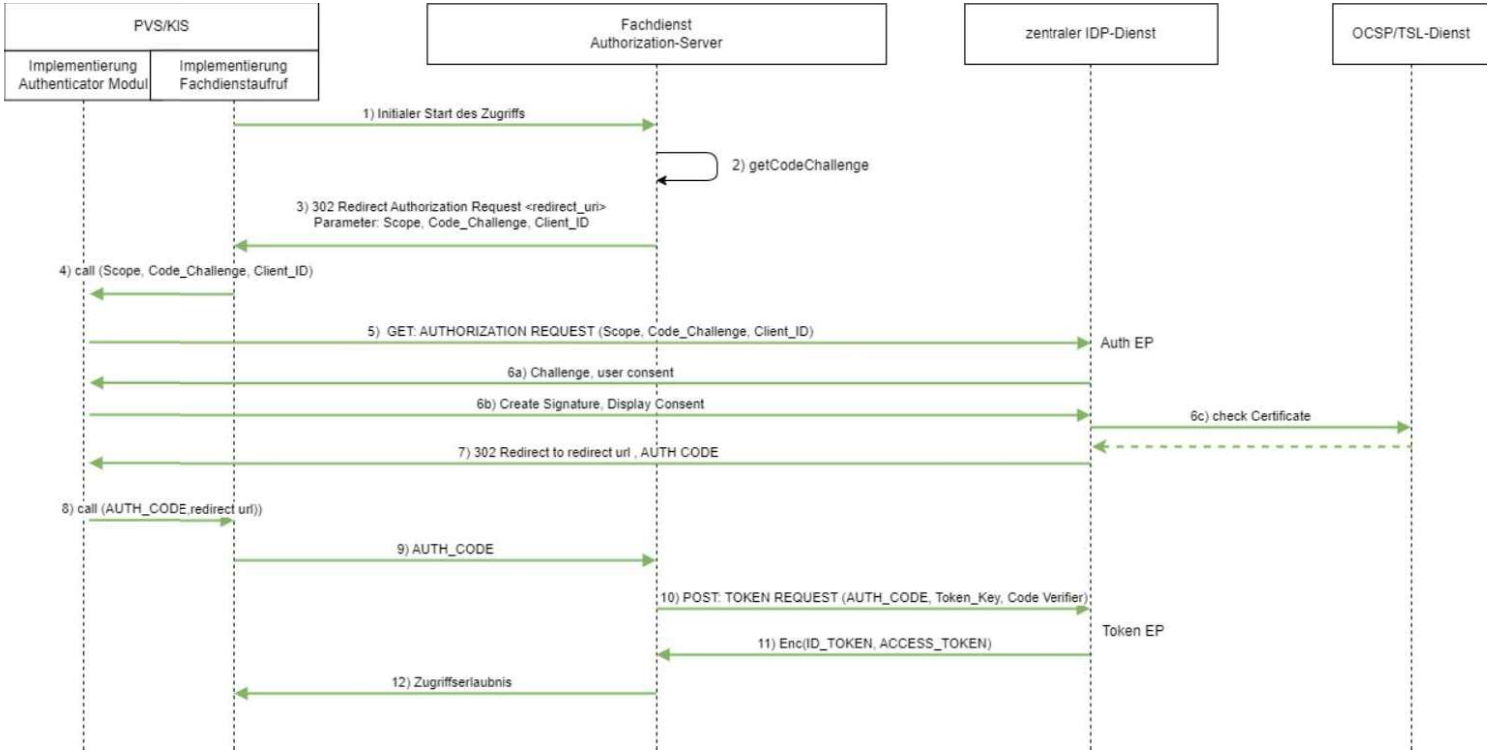
Datenstruktur nach gemSpec_Krypt#7.2

Name	Länge	Beschreibung bzw. Vorgabe des Werts
Version (=0x02)	1 Byte	Versionsnummer, wird auf den Wert 2 gesetzt
PU/nonPU	1 Byte	Wird das Chiffrat in der PU erzeugt, so MUSS der Wert 1 sein. Anderenfalls hat das Byte den Wert 0.
Response/Request	1 Byte	Für eine Nachricht des ePA-Clients an eine VAU-Instanz wird der Wert auf 1 gesetzt. In der Kodierung des Response-Chiffrat wird es auf den Wert 2 gesetzt, was markiert, dass es sich um eine Response handelt.
Request-Counter	8 Byte	Eindeutige Zählernummer für diesen Request, für jeden neuen Request wird vom Client dieser Wert um eins erhöht.
KeyID	32 Byte	KeyID aus dem Handshake (vgl. A_24623-*)
IV	12 Byte (= 96 Bit)	IV für die AES/GCM-Verschlüsselung (32 Bit Zufall + 64 Bit Verschlüsselungszähler, s. o. in A_24628-*)
CT	variabel	eigentliche AES/GCM-Chiffrat, dessen Länge gleich der Länge des Klartextes ist
GMAC-Wert	16 Byte (= 128 Bit)	Authentication-Tag, der während der AES/GCM-Verschlüsselung inkl. der Associated Data (Daten aus der Header-Tabelle, s. o.) berechnet wird.

Äußerer HTTP-Request
POST /VAU/CID-xyz mit Chiffrat im HTTP-POST-Request-Body

VAU-Protokoll, 2, Nutzer-Authentisierung via PKCE-OIDC in VAU-Kanal

Übersicht: https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#7.3



Mehr Informationen unter https://wiki.gematik.de/pages/viewpage.action?pageId=540040527

VAU-Protokoll, 2, Nutzer-Authentisierung, Nutzerpseudonym

Nach erfolgreicher Nutzer-Authentisierung deterministische Erzeugung eines Nutzer-Pseudonyms in der VAU-Instanz (im Aktensystem).

Übergabe im HTTP-Response-Header (VAU-NP: ….) der inneren HTTP-Response bei der letzten PKCE-OIDC-Nachricht an den Client.

Persistente Speicherung im Client.

Bei neuer späterer Neuaushandlung muss der Client diese Nutzerpseudonym im ersten äußeren Request (Nachricht M1) aufführen.

Vergleich 1.x, 2.x vs. 3.x

	1.x, 2.x	3.x
4 Nachricht Handshake / Verbindungsaufbau	+	+
Verbindungsschlüssel nach Verbindungsaufbau	+	+
Krypto	ECDH, ECDSA	ECDH, Kyber768, ECDSA
Beidseitige Authentisierung	+	-/+
Nichtpoduktiv-umgebungen	feste Client-Schlüssel	Client liefert Verbindungsschlüssel im äußeren HTTP-Request-Header
Kodierung im Handshake	JSON	CBOR