VAU-Protokoll erklären

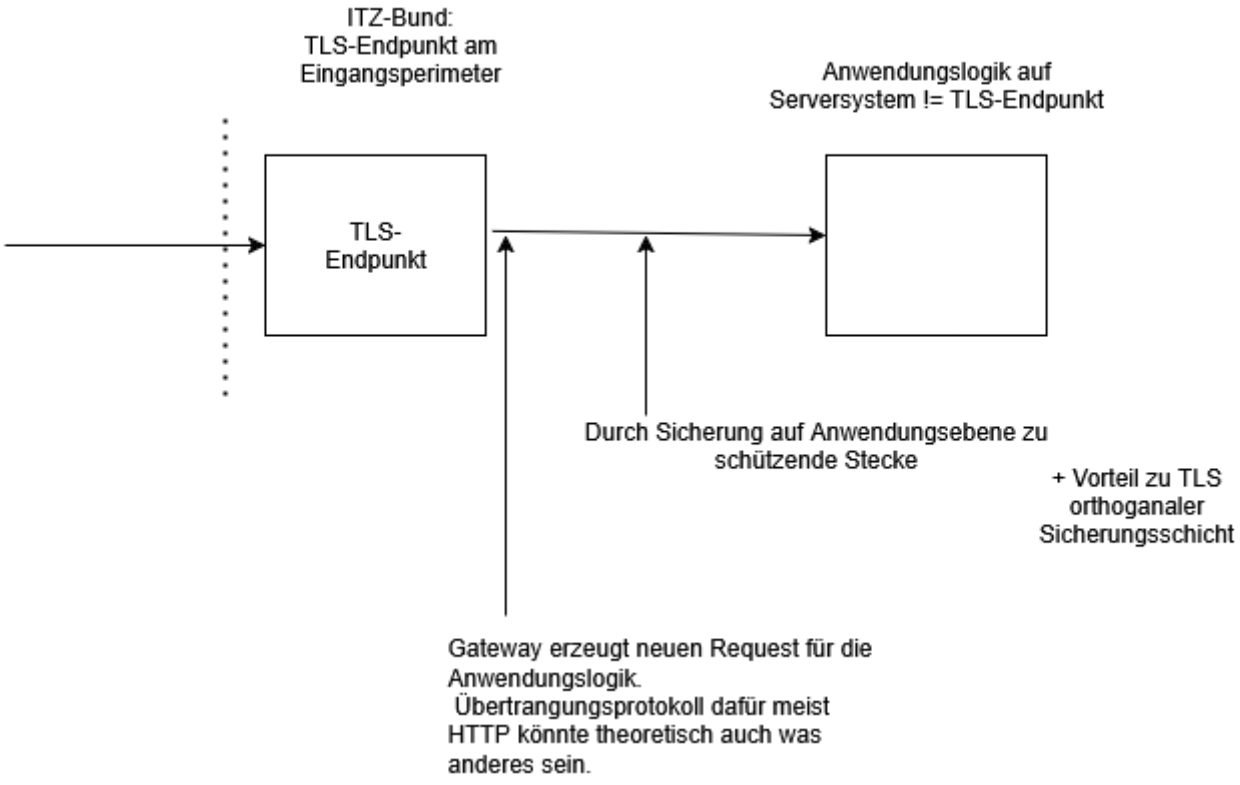
Entwicklungsunterstützung für das RKI

- Verhältnis Transportsicherung und Sicherung auf der Anwendungsschicht
- Historische Entwicklung VAU-Protokoll
- Beispiel TLS-Endpunkt / VAU-Protokoll
- Authentisierung Aktensystem als Client der VST
- Lieferung einer Submission AS->VST
- Beispiel-Implementierungen

Verhältnis Transportsicherung und Sicherung auf der Anwendungsschicht

Vorgegebene Architektur:

TLS-Endpunkt liegt nach ITZ-Bund-Vorgabe vor der "eigentlichen" Anwendung.



Historische Entwicklung VAU-Protokoll

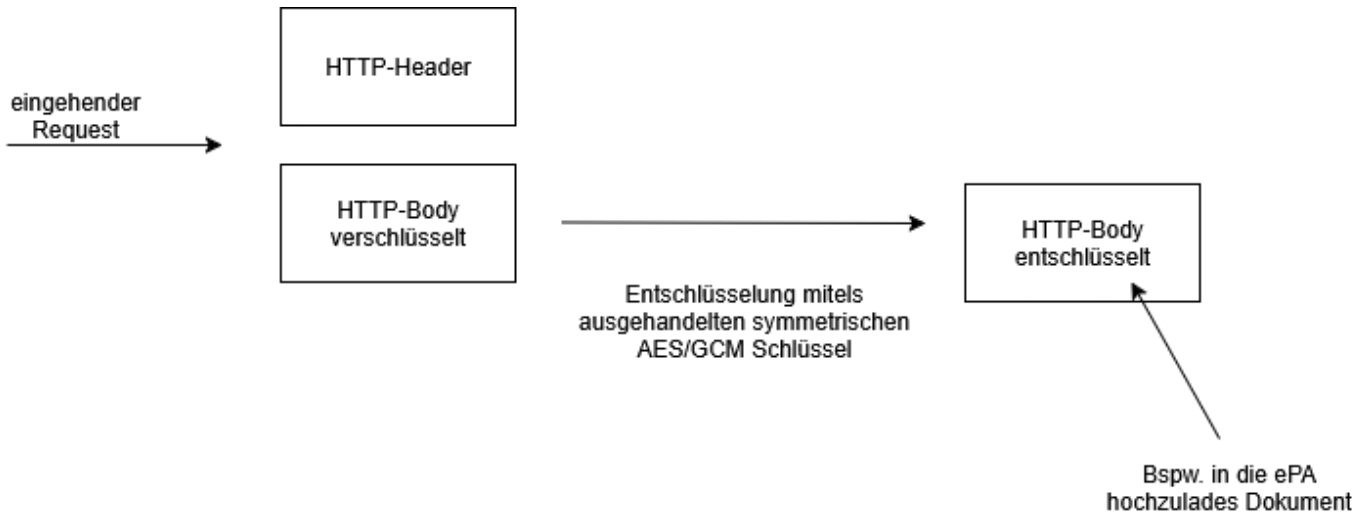
VAU-Protokoll Version 1.0

Im Jahr 2018 für erste ePA-Version (= Version 1.0) spezifiziert als zusätzliche Sicherungsschicht da TLS vor den VAU-Instanzen (Anwendungslogik) terminiert.

Für die Sicherung (authentisierte symmetrische Verschlüsselung mittels AES/GCM) muss mindestens ein symmetrischer Schlüssel frisch erzeugt werden. (Forward-Secrecy). -> Schlüsselaushandlung in einer Verbindungsinitialisierung mit dem Client.

(Hinweis: mit asymmetrischen kryptographischen Verfahren können nur sehr kleine Datenmengen verschlüsselt werden. -> man muss irgendwie zu symmetrischen Schlüsseln kommen.)

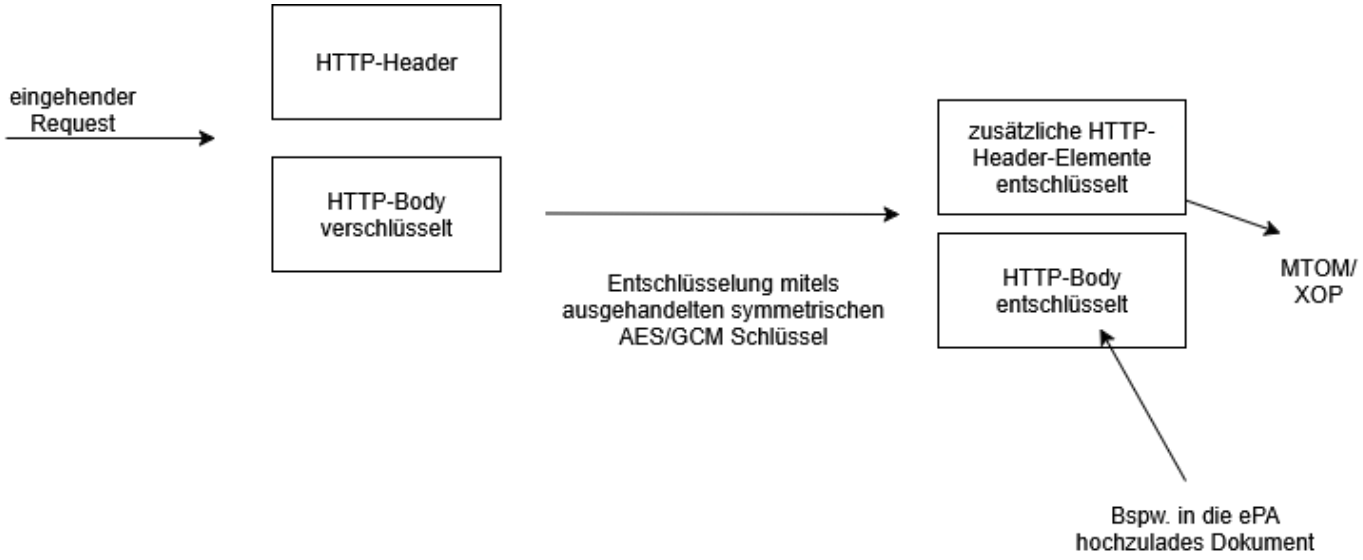
Nur die "Dokumente" (in SOAP-Container eingebettete Objekte) im HTTP-Body wurde verschlüsselt. Da 1-zu-1 Beziehung wurde der HTTP-Body = SOAP-Container = medizinisches Dokument verschlüsselt.



VAU-Protokoll Version 1.1

Im Jahr 2019 Erkenntnis: bei bestimmen SOAP-Kodierungen („SOAP Message Transmission Optimization Mechanism (MTOM)“/ „XML-binary Optimized Packaging (XOP)“-Kodierung) findet eine (aus Sicht VAU-Protokoll 1.0) ungünstige Verknüpfung zwischen HTTP-Body und HTTP-Header statt. Deshalb mussten nun auch Teile des HTTP-Headers in die Sicherung durch das VAU-Protokoll inkludiert

werden.

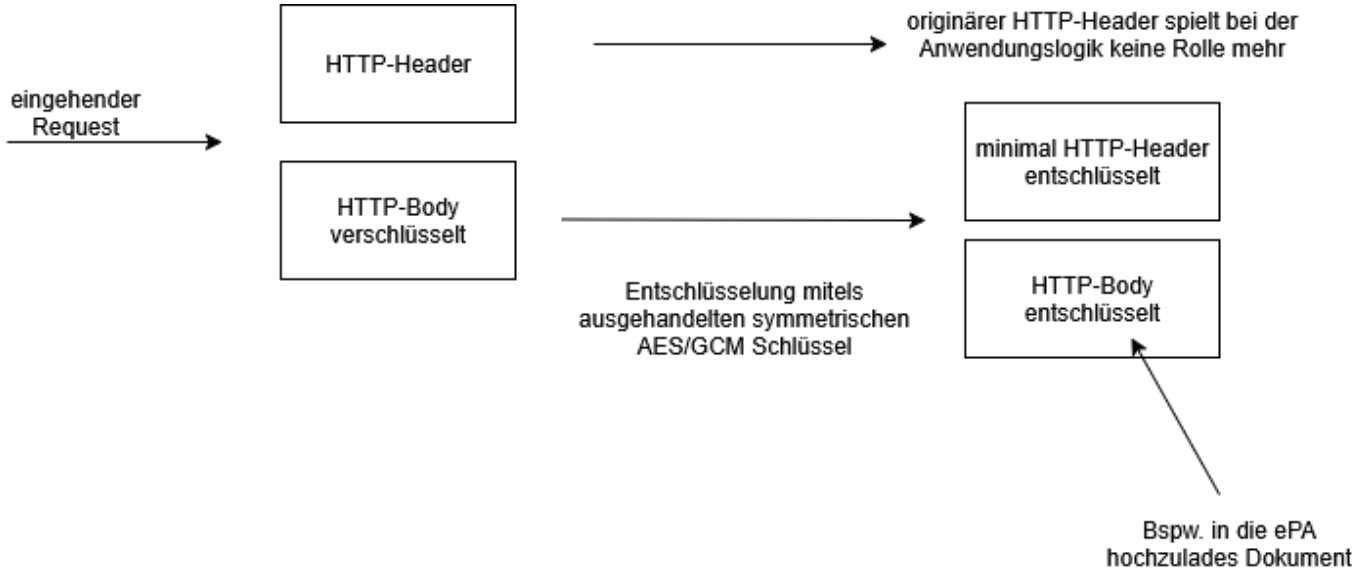


VAU-Protokoll Version für ePA für alle

Im Jahr 2023, Notwendigkeit der Änderung:

- Änderung der Authentisierungsmethode (OAuth2/OIDC)
- PQC-Sicherheit mit einbringen

Mit den Erfahrungen der Notwendigkeit der Anpassung von Version 1.0 nach 1.1 wird nun komplett ein minimaler HTTP-Header mit inkludiert. Es gibt also eine vollständigen HTTP-konformen inneren HTTP-Request / -Response.



Beispiel TLS-Endpunkt / VAU-Protokoll

https://tipkg.de:3000/

> location ~ ^/VAU {
>
> proxy_pass http://127.0.0.1:8081;
> }

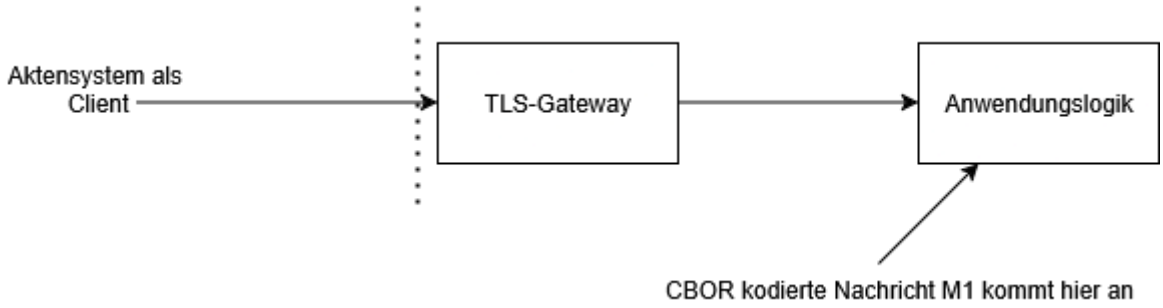
Client erzeugt Nachicht M1:

> { "MessageType": "M1",
> "ECDH_PK": { "crv": "P-256",
> "x": "(hexdump L=32)
> 4eb5890be8ecfee4e169dbf3bc3330fa19ccba9b7558b38216afed8ca833c536",
> "y": "(hexdump L=32)
> 0f278790e24a5512c0096008dde38ddab137a76142a3a543ccacf56d1242206d"
> },
> "Kyber768_PK": "(hexdump L=1184)
> c6d1809ba8063c4361a49019b68a31f440406bb8185391b355d08184c99fdb7c8
> 1d102ad46a3072d7794f9e1bb4132255ed86bd817024d9a76a54c25aa33b720db
> 9ac2f2ac0b948a634381d959a07f1060d7a8cb7940bd99866648b86cf55a3f9635
> 49c80540925a95954784a0ea86db4429c8906671a08c2c9c6287ca03c899afdd9
> 49d8e835cae8046b49508195051a518112be5ce63e89edb628c2a28134e8ca2ee
> b9345eca8f43085e47b4783ca10493f905a9384ed936497ff229f71c05ad26607b8
> 9170bac3a6209051af74f6498b06b46c62f0a376f5b2effa197940bcd696bc13d85
> 6ad17c5742535b81ac39fc295ae070833fd5c0138a3c5e560ea0f4b705e462dafa
> 0744ca65eeca08e6b21c5e002a7ee96765c55a22a8223980a25a1874b82555009
> db530357a4aba578eb343ea670724870b930b7be6e01b1305011971580c458f0b
> 4b11faf80c6cc68a5c075f1fa5ce53170169f84391ab94079a6793241428a71765a
> 7b7baf4cded0bbd9d1c2be10b718605a9df449780171ab7d70cb95a63aec83a4e
> a57348e22f19c74eaa3b6ec2c1c4130a373a38144fa869db7170faf602d1588031

```
1ba5c0344a1734980a4749bde95b9b8bbcd1518dec014d9d915d52989b040933
75c536373aafaecc7b2b63971cd22d10f9cbdc3c36e2db86ead6b21f9016de4849
add9b00de90b428856dad3b10a9b08d3054b0372a4ace203b2facd12139afdbc0
11ef06a89d9725ae9147ef33d3224538a4088980ca1043b966d26380a05c7ace7
1ff948b707da9937aabec39951bc247b54275d7322415897457846b3c8b429167
66677f833b9cb7e0976b75a0b3b56eb0d5c48cb27a20cac958a0183bfc912ae50a
ba838518af8b46177656d5b14afa03489336840d6b95a6c9a5d40812b02b12aa2
406edf9c40b3d08991254e1d971b074972412196ec387336e73c6d279a1f86b11
72a152c5b66f9d97fd243295e8b0a90ac2e9e257179cbb782db82db37559b2c49
38dc3115f12483a4a91c040be7
2948c6174da68c58d303311bc54c6274271f30621c2465b83894d757b33e120ba
cdb0071e6098d317389a7a945b5477337480185b4c42a10555a2085422281476
73d914a68f8b3972889205368da38ab43d0014ebaba6ac08924516ceebc0a53bc
cd72fc360177a090962598287d26d023a553413dac9618e7bf78922622c637ee2
c8716912012e12dd2fb9577e503c72548e89722b902c1
f1dbca735b9c8f1758c78609b534276ec1c23aea0655eaa3f583441eba0b14a095
4d1234b6db6f5fc63bbbf92ee55a520f469441784dcdb3875296c0348178c2bc13
e3c319978b241250b713
447df30107988c37a69b604a45a54c4b76a8a559e158ac62a17c2ae21de5a306e
7c24776c0251e9ca603aa1c61842dcc2b53684c33aef9b9fe7a694e691238827d7
6f9c24b0aa7273532fcda
5d7f70337dbbb62d81bacbdb22076ba7b8957727baa0bbba096ae09ddc1c64581
b56a9dc7ed2a3802f17c5c91950ead16246b6720a01680703b389ba10fcd1023f6
48cafc70a44e6470a1bbe
7074c3e9168ca75187ffc808935b973873276ed62e9ca4532868abe73a3a20cbb1
ba85837c08c00a0accfdd6017a4f3a29335950b3a2fb985ef2b34e545abd54da60
996b6350d9"
          }
```

Das wird mittels CBOR kodiert und per POST mit Pfadname "/VAU" an die VST geschickt:

https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#A_24428



Anwendungslogik erzeugt die Anwort-Nachricht "M2"

```
 {
 "MessageType": "M2",
 "ECDH_ct": {
  "crv": "P-256",
     "x": "(hexdump L=32)
204a1caba37874744a22906da570a2fd6aa04859d3176de481f4cfd928602f2b",
     "y": "(hexdump L=32)
d16d5326db4313626357fa1dc720b11f101ba582017888f334155f6780fcb175"
   },
"Kyber768_ct": "(hexdump L=1088)
c9c818ea529e86aa5cb69ae60b66825f6039aea437f635e068c26789010d38a64f
ba3730d788a708e4f25733fec4b04370
d8f23c2ba7bd7f40624a2e75954e287678922117f...8105321c1436f8e262c9a",
            "AEAD_ct": "(hexdump L=2010)
dbb4c502f1cb3cf7f47a4642520d....9c4509fce19b749a239c5a03455113fc5aadd
9b3c9885665ac768043d3c10f55578a4a9673d074c0"
          }
```

Client beantwortet diese Nachricht mit Nachricht "M3":

```
{
            "MessageType": "M3",
            "AEAD_ct": "(hexdump L=1233)
d352f2d7a65689883b826f8120f...cbe970e776e5ed9e776c43b45c81cfe5759c9a
699d06af0651235e1a",
            "AEAD_ct_key_confirmation": "(hexdump L=60)
248b899b95689b672661da83ceb0db15889bb365a7536e686924786f5a8b29b8
decbf4a087b784355982b9b
2eab3385cd58eadb7204d4a4b0a6d14a3"
          }
```

Anwendungslogik antwortet mit Nachricht "M4":

```
{
            "MessageType": "M4",
            "AEAD_ct_key_confirmation": "(hexdump L=60)
b2652fd91639b49d5efbbe62f2c915896dfd0fbba43a2e843f73ebec7c85e6c91cd
d1f5bbbb63fc8f1e64c8
f907a97ba93b1bb72790bd69f1dfd2e2e"
          }
```

Dann haben Client und Server (Anwendungslogik in der VST) die beiden AES/GCM-Schlüssel K2_c2s_AppData und K2_s2c_AppData gemeinsam erzeugt.

Im Rahmen des Verbindungsaufbau (mit der Nachricht M2) legt die VST die VAU-Connection-ID (VAU-CID) fest. ("Im Response-Header MUSS mit der HTTP-Header-Variable "VAU-CID" die ID aufgeführt werden.").

Authentisierung Aktensystem als Client der VST

Nun sind die Schlüssel K2_c2s_AppData und K2_s2c_AppData erfolgreich erzeugt.

Das AS erzeugt die Daten

```
GET /epa/authz/v1/freshness HTTP/1.1
Host: www.vst...
```

Das wird mittels K2_c2s_AppData und AES/GCM verschlüsselt (https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#A_24632).

Die Anwendungslogik in der VST entschlüsselt den Request und erzeugt eine Nonce. Antwort dann mit K2_s2c_AppData verschlüsselt. Entschlüsselte Anwort dann bspw.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Date: Fri, 21 Jun 2024 14:18:33 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Content-Length: xxx

xxxx korrigieren
YSBiYXNlNjQgZW5jb2RlZCBjaGFsbGVuZ2UgKGhlYWx0aCByZWNvcmQgc3lzdGVtIHNwZWNpZmljIGNvbnRlbnQp
```

Dann erzeugt das Aktensystem einen JWT-Body

```
{
   "type" : "ePA-Authentisierung über PKI",
   "iat" : ...zeit...,
   "challenge" :
"YSBiYXNlNjQgZW5jb2RlZCBjaGFsbGVuZ2UgKGhlYWx0aCByZWNvcmQgc3lzdGVtIHNwZWNpZmljIGNvbnRlbnQp",
   "sub": "9-ePA-AS"
```

```
}
```

Das signiert das AS mittels seine VAUAUT Schlüsselmaterials und erzeugt damit einen JWT ganz klassisch in compact form. Dann erzeugt das AS eine Zeichenkette

```
POST /epa/authz/v1/send_authorization_request_bearertoken HTTP/1.1
Host: www.vst...
Content-Type: application/json; charset=UTF-8
Content-Length: xxy

34893028409332sljdaslkjdlaksjdlasjaljdas.e07radasdlkjasldjsaldjalsdjas.sajdlasue908ejdlskajdlaksjd
```

Das wird mittel K2_c2s_Data verschlüsselt und an die Anwendungslogik gesendet. Die Anwendungslogik entschlüsselt das mittels K2_c2s_Data. Extrahiert das JWT und prüft das JWT (insbesondere die Signatur) wie in

https://gemspec.gematik.de/docs/gemSpec/gemSpec_Krypt/latest/#A_24658-01 definiert.

Lieferung einer Submission AS->VST

1. Das AS (als Client der VST) und die VST haben jetzt Schlüssel K2_c2s_AppData und K2_s2c_AppData ausgehandelt.
2. Das AS hat sich als Client authentisiert.

Das AS erstellt eine Submission-Datastrukur (pre_submission)

```
[
  {
    "type": "as->vst",
    "created_at": Unixzeit (POSIX time),
    "submission_id": Submission-ID,
    "call_back_url": url_aktensystem
   },
   [LP1, AN1],
   [LP2, AN2],
   [LPx, ANx]
]
```

Das wird per CBOR kodiert. Das AS erzeugt eine Datenstruktur

```
PUT /api/epa/v1/submissions/87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7
HTTP/1.1
Host: xyz
Content-Type: application/cbor
Date: Fri, 21 Jun 2024 14:18:33 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Content-Length: xyz

... binäre Daten (CBOR) ...
```

Das wird per K2_c2s_AppData verschlüsselt und per

```
POST /VAU/CID-xyz HTTP/1.1
Host: xyz
Content-Type: application/octet-stream
Date: Fri, 21 Jun 2024 14:18:33 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Content-Length: xyz

... binäre Daten (VAU-Protokoll A_24628-01) ...
```

an die Applikationslogik (über den TLS-Kanal gesendet).

Beispiel-Implementierungen

öffentlich verfügbare Beispiel-Implementierungen des VAU-Protokolls

- von fbeta:

  https://github.com/fbeta-GmbH/ePA3-Service-OpenSource

  (haben meine Implementierung (https://bitbucket.org/andreas_hallof/vau-protokoll/) als Basis verwendet)
- gematik-Java-Implementierung aus dem Test-Bereich:

  https://github.com/gematik/lib-vau
- gematik-Beispiel-Code ist C#

  https://github.com/gematik/lib-vau-csharp
- gematik-Beispiel-Code in Go:

  https://github.com/gematik/zero-lab/tree/vau/pkg/libvau