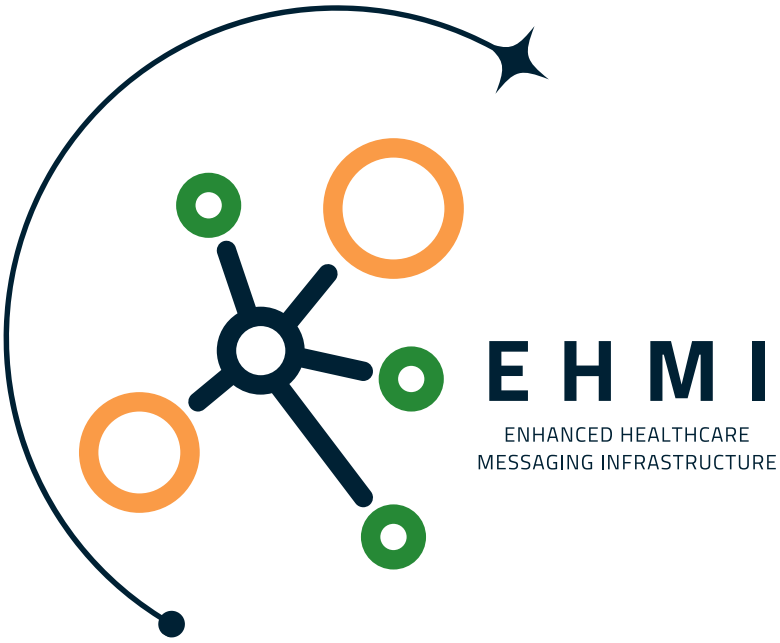


Sikkerhedsarkitektur for Enhanced Healthcare Messaging Infrastructure (EHMI) services

2024-04-29, 0.2.2

medcom

SUNDHEDSDATA-
STYRELSEN

Revisionshistorik

Version	Initialer	Dato	Beskrivelse
0.1	CHG	2024-02-23	Første udgave
0.1.1	CHG	2024-04-08	Efter review fra ASHA
0.2	CHG	2024-04-19	Opdateret til at basere sig på FAPI 2.0
0.2.1	CHG	2024-04-29	Flyttet indhold til ny dokumentskabelon
0.2.2	CHG	2024-05-06	Efter review fra ASHA og OVI

Indhold

Revisionshistorik 2

1. Introduktion 4

1.1 Publikum 4
1.2 Formål 4
1.3 Baggrund 4
1.3.1 OAuth2 4
1.3.2 Confidential clients og public clients 5
1.3.3 Bearer tokens og sender-constrained tokens 5
1.3.4 OpenID Connect 6

2. Overordnede Usecases 7

3. Sikkerhedsmodel 8

3.1 Klienttyper 8
3.2 Klientautentifikation 9
3.3 Indrullering af klienter 9
3.3.1 Metadata for systemklienter 10
3.3.2 Metadata for fulde klienter med brugerdelegering 11
3.4 Integrationsflows og protokoller 11
3.4.1 Systemkalds-scenarie 11
3.4.2 Brugerkald (borgere/fagpersoner) 14
3.4.3 Token-indhold 14
3.5 Krav til aktørerne 14
3.5.1 Regler for netværksforbindelser 15
3.5.2 Regler for klienter 15
3.5.3 Regler for Authorization Servere 15

3.5.4 Regler for Resource Servere (EHMI services) 15

4. EHMI Delivery Status (EDS) 16

4.1 Usecases 16
4.2 Specificering af sikkerhedsmodel til EDS 16
4.2.1 Indrullering/whitelisting af systemklienter til registrering i EDS 16
4.2.2 Indrullering/whitelisting af klienter til søgning og opslag 17
4.2.3 Kald til Token Endpoint 18
4.2.4 Kald til EDS 19

5. EHMI Addressing Service (EAS) 20

6. EHMI Endpoint Register (EER) 21

7. Appendiks: Arkitekturbeslutninger 22

7.1 Udgangspunkt for anvendelse af OAuth 2.0 til sundhedsområdet 22
7.2 Token-binding via applikations- og/eller transportlaget 23
7.3 Selv-indeholdt token eller token reference 24
7.4 Indhold af token til EDS: device_id, SOR kode og GLN 25

8. Appendiks: Relevante uddrag af FAPI 26

8.1 FAPI afsnit 5.2.1 Requirements for all endpoints 26
8.2 FAPI afsnit 5.2.2 Requirements for endpoints not used by web browsers 26
8.3 FAPI afsnit 5.2.3 Requirements for endpoints used by web browsers 26
8.4 FAPI afsnit 5.3.2.1 General Requirements 27
8.5 FAPI afsnit 5.3.2.2 Authorization Code Flow 27

9. Referencer 29

1. Introduktion

I dette dokument beskrives sikkerhedsarkitekturen for services i [EHMI] (Enhanced Healthcare Messaging Infrastructure), herunder anvendte sikkerhedsprotokoller, akkreditiver og token-formater.

Under EHMI services forstås de services i meddelelsesinfrastrukturen som ikke direkte indgår i selve punkt til punkt meddelelseskommunikationen, herunder:

- EHMI Delivery Status (EDS) (På dansk: Forsendelsesstatusservice)
- EHMI Addressing Service (EAS) (På dansk: Sundhedsadresseringsservice)
- EHMI Endpoint Register (EER) (På dansk: Postkasseregister)

Denne udgave af dokumentet har alene til formål at give parterne som skal tilgå EHMI Delivery Status (EDS) et grundlag til estimering af deres integrationsopgave.

Alle afsnit markeret med <*TBD*> bliver udfyldt senere.

1.1 Publikum

Dokumentet har en teknisk karakter og henvender sig primært til arkitekter og software-udviklere.

Projektledere og andre projektdeltagere kan med fordel også orientere sig i relevante dele af dokumentet.

1.2 Formål

Dokumentet skal i forbindelse med pilotafprøvningen af kommunale prøvesvar kunne danne grundlag for realisering af sikkerhedsmekanismerne i både udviklingen af EHMI services, etablering af en Authorization Server og hos parterne som skal anvende disse services.

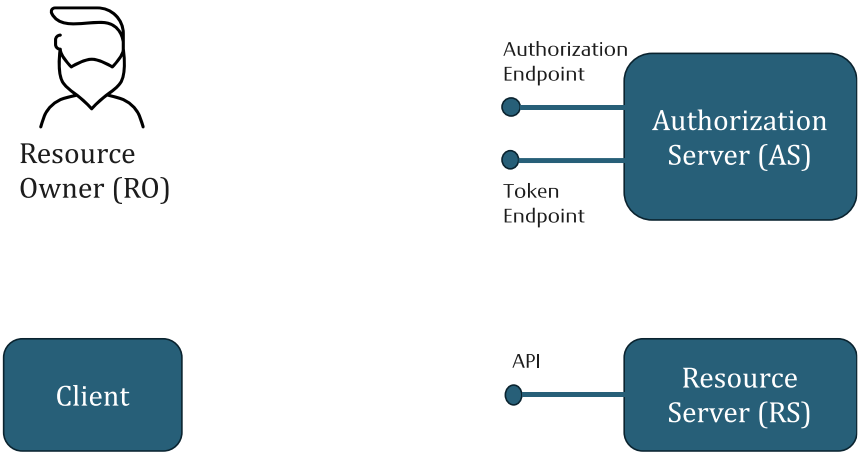
1.3 Baggrund

Sikkerhedsarkitekturen tager afsæt i [OAuth2] specifikationen som er en åben standard for adgangsdelegering til HTTP-baserede tjenester, herunder tjenester som udstilles som REST-fulde services, og baserer sig på [JTP-H] som er sundhedsvæsnets profilering af JSON Web Tokens [JWT].

1.3.1 OAuth2

Den generelle OAuth2 model (herefter blot kaldt *OAuth*) fastsætter protokoller for hvordan en bruger (en *Resource Owner)* autoriserer en *client* (fx en mobil app eller en webapplikation) til at må tilgå beskyttede resurser hos en *Resource Server*. Adgang til beskyttede resurser gives på baggrund af et *access token*, som en *Authorization Server* udsteder til klienten efter brugergodkendelse.

I nedenstående Figur 1 vises de fire aktører som indgår i OAuth modellen – *Resource Owner* er her afbilledet som en menneskelig bruger, men kan i OAuth også være en systembruger.



Figur 1: Aktørerne i OAuth

Bemærk i øvrigt, at 'Auth' i OAuth er en forkortelse for 'Authorization' og ikke 'Authentication'.

1.3.2 Confidential clients og public clients

Både mobile apps, IoT-devices, webapplikationer (single-page eller med backend) og server-processer kan optræde som *clients* i OAuth forstand, men der skelnes mellem *confidential clients* og *public clients.*

En *confidential client* er en klient som kan beskytte en hemmelighed (et *client secret* i form af en shared key eller et PKI nøglepar) som kan anvendes til autentifikation af klienten i Authorization Server, hvorimod en *public client* er en klient som ikke kan beskytte statiske hemmeligheder. En mobil app som distribueres via en app-store eller en webapplikation som afvikles direkte i browseren (fx en SPA (single-page application)) er som udgangspunkt *public clients,* idet en angriber vil kunne ekstrahere hemmeligheden fra den downloadede app eller fra HTML5/JavaScript-klienten i en ren browserbaseret applikation.

Under de rette forudsætninger kan en public client på runtime blive indrulleret ved Authorization Server med instans-specifikke akkreditiver (eksempelvis via [OAuth-DCR] dynamisk klient registreringsprotokollen) og derefter optræde som confidential client.

1.3.3 Bearer tokens og sender-constrained tokens

I OAuth skelnes der mellem to typer af access tokens. *Bearer tokens* er ikke bundet til en bestemt klient, men kan i princippet benyttes til at kalde ressource serveren som tokenet giver adgang til af enhver som er i besiddelse tokenet (eller 'bærer' tokenet).

En højere grad af sikkerhed og beskyttelse kan opnås gennem *sender-constrained tokens*, som er kryptografisk bundet til den klient de er udstedt til og som dermed ikke kan benyttes af andre (ondsindede) parter ('sender-constrained' semantikken svarer i øvrigt til 'holder-of-key' konceptet i SAML standarden). Til OAuth er der defineret to mekanismer til at opnå sender-constained tokens, baseret på binding af tokenet til transportlaget (via [OAuth-MTLS]) eller til applikationslaget (via [OAuth-DPOP]).

1.3.4 OpenID Connect

Standarden OpenID Connect [OIDC] udvider OAuth fundamentet med et autentifikations- og identitets-lag. OIDC udvider OAuth med en brugerautentifikationsprotokol og med et *identity token*, som er et signeret JSON Web Token [JWT] med attributter om den autentificerede bruger. Hvor OAuth access tokens er møntet på eksterne API'er, udstedes OIDC identity tokens i stedet til klienten (når denne har behov for brugeroplysninger).

2. Overordnede Usecases

<TBD>

3. Sikkerhedsmodel

Udgangspunktet for sikkerhedsmodellen for EHMI services er OAuth-sikkerhedsprofileringen [FAPI], se appendiks 7.1 Udgangspunkt for anvendelse af OAuth 2.0 til sundhedsområdet for en gennemgang af rationalet for valget af FAPI profilen.

I FAPI sikkerhedsprofilen er det påkrævet at benytte sender-constrained tokens (se afsnit 1.3.3), og FAPI tillader såvel [OAuth-MTLS] som [OAuth-DPOP] mekanismerne til at realisere sender-constrained tokens. I EHMI sikkerhedsmodellen anvendes alene OAuth-MTLS, hvor tokens bindes til klienten i transportlaget. Se rationalet for valget i appendiks 7.2 Token-binding via applikations- og/eller transportlaget.

Hvor FAPI profilen fastlægger sikkerhedsprotokoller og valideringsregler for aktørerne, der indgår i OAuth flows, forholder den sig ikke til indholdet af de tokens som indgår i de forskellige flows. Indhold at tokens tager derimod i EHMI afsæt i det danske sundhedsvæsnets profilering af [JWT] tokens, se [JTP-H].

I dette kapitel præsenteres og gennemgås de elementer af profilerne som er relevante for at understøtte de overordnede brugsscenarier som relaterer sig til EHMI services. Læseren forventes således ikke at have nærlæst FAPI og JTP-H profilerne.

3.1 Klienttyper

I EHMI sikkerhedsmodellen skelnes der mellem to typer af OAuth klienter.

1. En *systemklient* som via system-til-system-integration tilgår en EHMI service. Selvom klienten måtte foretage opslag på baggrund af en brugerhandling, er systemklienten defineret ved at brugerens identitet ikke er relevant i den givne kontekst og ikke kommunikeres til EHMI servicen. Eksempler på systemklienter er sundhedsadresseringsservicen som tilgår postkasseregisteret eller et fagsystem som tilgår forsendelsesstatusservicen

2. En *fuld klient med brugerdelegering* et en klient som foretager kald for en autentificeret bruger, hvis identitet kommunikeres til EHMI servicen som en del af tokenet der indgår i kaldet. Klienten er defineret som 'fuld' i OAuth forstand, idet den både kan autentificere sig selv og brugeren (via et webbrowser-baseret flow). Et eksempel på en fuld klient med brugerdelegering i en EHMI kontekst er backenden til en webapplikation som tilbyder søgninger i forsendelsesstatusservicen.

Begge klient-typer er således *confidential clients* i OAuth forstand (se afsnit 1.3.2), som [FAPI] sikkerhedsmodellen tager udgangspunkt i.

EHMI services tillader ikke direkte tilgange fra mobile/native apps uden backend.

Denne klienttype (som via fx [OAuth-DCR] dynamisk klient registreringsprotokollen først vil skulle indrulleres for at kunne optræde som confidential client) indgår således ikke i de efterfølgende beskrivelser.

3.2 Klientautentifikation

Autentifikation af klienter til EHMI services er baseret på OCES systemcertifikater[1], hvor hver klient skal have sit eget nøglepar.

OCES systemcertifikaterne kan enten være udstedt af den organisation som anvender klienten eller af leverandøren som tilbyder løsningen (potentiel til flere organisationer). Netop ved multi-tenant systemer vil det som regel være mest hensigtsmæssigt at benytte et systemcertifikat udstedt af leverandøren.

I EHMI sikkerhedsmodellen benyttes OCES systemcertifikater alene til autentifikation af klienter hos Authorization Server og ikke til at autorisere adgange baseret på certifikatoplysninger (som fx CVR nummer), for ikke at skabe unødvendige bindinger mellem certifikater og rettigheder.

Autentifikation af klienterne foregår ved, at OCES3 systemcertifikaterne benyttes af klienterne som TLS-klientcertifikater i kommunikationen med såvel Authorization Server som EHMI services.

3.3 Indrullering af klienter

I EHMI infrastrukturen understøttes kun *confidential clients*, som statisk registreres/indrulleres i Authorization Server.

Til pilotafprøvningen hvor der indgår et begrænset antal klienter registreres disse manuelt via Authorization Serverens administrationssnitflade. På sigt er det muligt at etablere en selvbetjeningsløsning hvor klienter selv kan anmode om indrullering og efter godkendt anmodning eksempelvis kunne benytte mekanismerne i [OAuth-DCR] til registrering i Authorization Server.

For at blive registreret danner klienten et simpelt metadata dokument som beskrevet nedenunder (metadata som er baseret på [OAuth-DCR]), der benyttes til konfiguration i Authorization Server.

Hver klient instans skal registreres separat og skal anvende sit eget OCES nøglepar og sit eget `client_name`. Ved registrering tildeles klienten et `client_id`, som denne skal benytte ved efterfølgende requests til henholdsvis Token Endpoint og Authorization Endpoint (for klienter med brugerdelegering) hos Authorization Server.

[1] I OCES3 anvendes begreberne 'organisationscertifikat' og 'systemcertifikat' for hvad der i OCES2 henholdsvis blev kaldt virksomhedscertifikater (VOCES) og funktionscertifikater (FOCES).

3.3.1 Metadata for systemklienter

Følgende elementer skal angives i klient metadata dokumentet:

Metadata element	Beskrivelse
`token_endpoint_auth_method`	Hvordan klienten autentificerer sig ved Authorization Serverens Token Endpoint. Sættes til den faste værdi `tls_client_auth` dvs. autentifikation på transportlaget via et (OCES) TLS-klientcertifikat.
`grant_types`	Hvilke OAuth flows klienten anvender. Sættes til den faste værdi `client_credentials`.
`client_name`	Et sigende navn for klienten, som letter administrationsopgaven i Authorization Server. Fx `EHMI Access Point for Region Midtjylland`.
`scope`	En liste af OAuth scope værdier (adskilt med blank space) som klienten ønsker at kunne lade indgå i access tokens. Fx `EAS EDS`.
`contacts`	Et array med kontaktoplysninger (typisk e-mailadresser) til organisationen, som er ansvarlig for driften af klienten.
`tls_client_auth_subject_dn`	Subject Distinguished Name fra OCES systemcertifikatet som anvendes som TLS-klientcertifikat, fx `subject=CN=Region Midtjylland test systemcertifikat, serialNumber=UI:DK-O:G: 23b375ea-bfc8-4b84-9fec-c0accc70a865, O= Region Midtjylland, organizationIdentifier=NTRDK-29190925, C=DK` (Der anbefales at OCES systemcertifikater ikke udstedes med et certifikatspecifikt UUID[2], idet Subject Distinguished Name derved videreføres i uforandret form ved certifikatfornyelse og klientens registrering i Authorization Server således ikke skal ajourføres i forbindelse med certifikatfornyelse).

[2] Se MitID Erhvervsadministrationen, under Indstillinger -> Certifikater -> UUID i certifikater

Eksempel metadata dokument for en systemklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": "client_credentials",
  "client_name": "EOJ Systemet i Korsbæk Kommune",
  "scope": "EDS system/AuditEvent.c",
  "contacts": [
    "døgnsupport@korsbæk.dk",
    "+45 1234 5678"
  ],
  "tls_client_auth_subject_dn": "subject=CN=Korsbæk EOJ systemcertifikat,
serialNumber=UI:DK-O:G:9b996be1-b439-45ab-b239-0c95d8e02aee, O=Korsbæk
Kommune, organizationIdentifier=NTRDK-11111111, C=DK"
}
```

3.3.2 Metadata for fulde klienter med brugerdelegering

<TBD>

3.4 Integrationsflows og protokoller

I både systemkalds- og brugerkalds-scenarierne foregår klient autentifikation via OAuth `client_credential` mekanismen og med `tls_client_auth` autentifikationsmetoden som defineret i [OAuth-MTLS]. Se i øvrigt gennemgangen under appendiks 7.3 Token-binding via applikations- og/eller transportlaget.

OBS: Der findes en lang række forskellige OAuth kodebiblioteker[3] og JWT kodebiblioteker[4], som implementer de standardflows og mekanismer, som er defineret i de forskellige OAuth-relaterede specifikationer, som de her beskrevne flows baserer sig på. I en praktisk implementering kan der med fordel tages udgangspunkt i standard kodebibliotekerne.

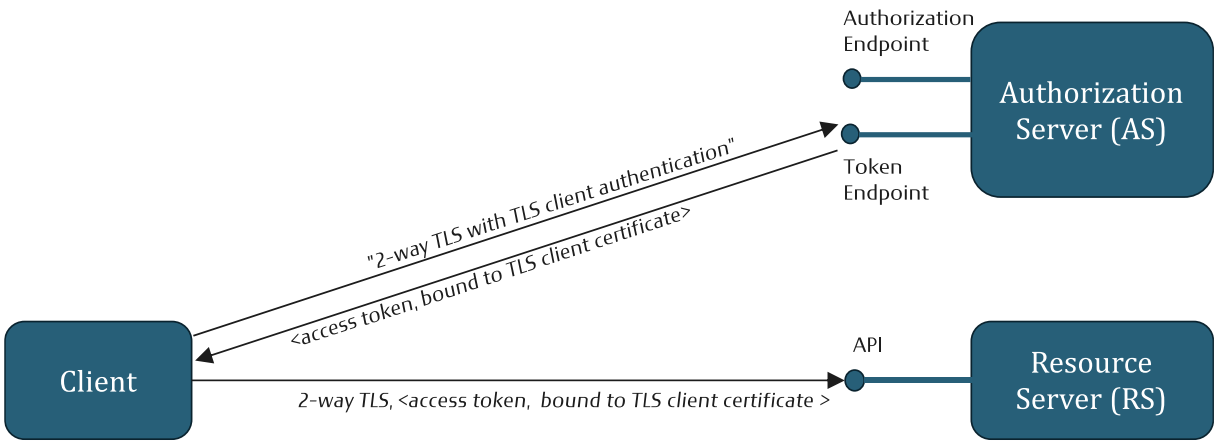
I det følgende gennemgås først trin for trin integrationsflows for de to anvendelsesscenarier systemkald og brugerkald.

3.4.1 Systemkalds-scenarie

I systemkalds-scenariet benytter klienten sit OCES TLS-klientcertifikat som autentifikations-mekanisme i kaldet til Authorization Serverens Token Endpoint og til kald til EHMI services. Authorization Serveren validerer TLS-klientcertifikatet samt requestet og returner et access token til klienten som denne efterfølgende benytter til at tilgå en EHMI service (som optræder som Ressource Server), se nedenstående figur.

[3] Se fx https://oauth.net/code/

[4] Se fx https://openid.net/developers/jwt-jws-jwe-jwk-and-jwa-implementations/



Figur 2: Systemkalds-scenarie

I det følgende gennemgås de enkelte skridt i processen.

Skridt 1: Kald af Token Endpoint hos Authorization Server
Klienten tilgår Authorization Serveren via 2-vejs TLS med TLS-klientcertifikatet som er blevet registreret i Authorization Server og laver et HTTP POST kald til Token Endpoint med angivelse af følgende kaldsparametre:

Parameter	Beskrivelse
`grant_type`	Sættes til den fast værdi `client_credentials`.
`Scope`	Ønskede scope(s). Se nedenstående beskrivelser af sikkerhedsmodellen for de enkelte EHMI services for de konkrete scopes der skal angives. (Authorization Server benytter desuden `scope` værdien til at fastlægge en eventuel audience/aftager for access tokenet som den udsteder).
`client_id`	Sættes til den client_id som blev tildelt klienten under indrullering ved Authorization Server.

Klienten URL-encoder alle parameterværdier, sætter dem sammen og laver POST kaldet til Token Endpoint.

Eksempel kald:

```
POST /token HTTP/1.1
Host: authorization.sundhedsdatastyrelsen.dk
User-Agent: curl/8.4.0
Accept: */*
Proxy-Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 263

grant_type=client_credentials&scope=EDS%20EAS&client_id=0ba284d1-8974-
4241-bce1-0498bc2d48ea
```

Skridt 2: Retursvar
Svaret fra Token Endpoint består af en JSON struktur med et access token, samt information om tokenet.

Følgende værdier returneres:

Returværdi	Beskrivelse
`access_token`	Access tokenet som klienten har requestet. Benyttes til efterfølgende kald til en EHMI service. Access tokenet er kryptografisk bundet til klientens TLS-klientcertifikat. Klienten skal således benytte samme klientcertifikat ved kald til EHMI services.
`token_type`	Typen af tokenet. Har altid den fast værdi `Bearer`[5] i EHMI.
`expires_in`	Tokenets gyldighed i sekunder.
`scope` (Optionel returværdi)	Ønskede scope(s). Inkluderes altid hvis scopet er mindre end requestet. Et fravær af returværdien betyder, at access tokenet indeholder det fulde scope som klienten anmodede om.

[5] Tokenet er sender-constrained til TLS-klientcertifikatet, men i OAuth familien af specifikationer er der ikke defineret en særskilt `token_type` til TLS-bundne tokens.

Eksempel response fra Authorization Server:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "access_token":"ey ... J9.ey ... n0.OG ... pg",
  "token_type":"Bearer",
  "expires_in":300,
  "scope":"EDS"
}
```

Skridt 3: Kald EHMI service

Klienten tilgår EHMI via 2-vejs TLS med samme TLS-klientcertifikat som ved kaldet til Authorization Server. I REST-kaldet til EHMI servicen inkluderes access tokenet i en `Authorization` HTTP header som angivet:

```
Authorization: Bearer <access token>
```

Eksempel kald til en EHMI service:

```
POST /base/AuditEvent HTTP/1.1
Host: ehmi.medcom.dk
User-Agent: curl/8.4.0
Accept: application/fhir+json
Content-Type: application/fhir+json
Content-Length: 11996
Authorization: Bearer eyJhb ... Dhi6g

{
...
}
```

3.4.2 Brugerkald (borgere/fagpersoner)

<TBD>

3.4.3 Token-indhold

<TBD>

3.5 Krav til aktørerne

I det følgende beskrives de krav og regler som aktører der tilgår/udstiller EHMI services eller indgår som infrastrukturkomponenter skal overholde. De fleste regler er baseret på kravene fra [FAPI] og de afsnit fra FAPI som der refereres til i nedenstående er direkte gengivet i Appendiks: Relevante uddrag af FAPI.

3.5.1 Regler for netværksforbindelser

Såvel klienter, Authorization Server og EHMI services (Ressource Servere) skal følge kravene om brugen af TLS i afsnit *5.2.1 Requirements for all endpoints* og *5.2.2 Requirements for endpoints not used by web browsers* i [FAPI].

Authorization Serverens Authorization Endpoint skal desuden opfylde kravene i *5.2.3 Requirements for endpoints used by web browsers* i [FAPI].

3.5.2 Regler for klienter

Alle klienter skal følge kravene i afsnit *5.3.2.1 General Requirements* i [FAPI], men kan se bort fra krav som vedrører DPoP eller `private_key_jwt`. Fulde klienter med brugerdelegering (se afsnit 3.1) skal desuden leve op til kravene i *5.3.2.2 Authorization Code Flow* i [FAPI].

Klienterne skal behandle alle access tokens som de modtager fra Authorization Server som såkaldte *opaque* tokens. Med opaque menes at tokens skal behandles som værende i et proprietært format, som er møntet på EHMI services. Klienten må ikke tolke på format og indhold af et access token, men kan i stedet benytte relevante meta-informationer (token levetid og scopes) i svaret fra Token Endpoint.

3.5.3 Regler for Authorization Servere

<TBD>

3.5.4 Regler for Resource Servere (EHMI services)

<TBD>

4. EHMI Delivery Status (EDS)

I [EHMI] er der følgende *stationer* som indgår i punkt-til-punkt meddelelsesforsendelser: Fagsystemer, message-service-handlere og access-points.

Alle stationer der indgår i en EHMI meddelelsesforsendelse skal registrere deres meddelelseshåndteringer i forsendelsesstatusstatusservicen EDS, som beskrevet i FHIR implementation guiden på https://build.fhir.org/ig/medcomdk/dk-ehmi-eds/.

Stationerne oprettes i EHMI Endpoint registeret (EER) og tildeles i forbindelse med oprettelsen et unikt *device_id.*

Som det fremgår af FHIR implementation guiden realiseres forsendelsesstatus som en profilering af FHIR `AuditEvent` ressourcen.

4.1 Usecases

Der er to overordnende usecases for anvendelsen af forsendelsesstatusservicen EDS.

Stationerne i en EHMI forsendelse foretager hver især en *registrering af forsendelsesstatus* i EDS.

EDS stiller en grænseflade til *søgning og opslag* til rådighed, som kan benyttes til track'n'trace af meddelelsesforsendelser eller til fejlsøgning.

4.2 Specificering af sikkerhedsmodel til EDS

Udgangspunkt er ovenstående generelle Sikkerhedsmodel med følgende udvidelser.

4.2.1 Indrullering/whitelisting af systemklienter til registrering i EDS

Udover de i afsnit 3.3 Indrullering af klienter beskrevne elementer skal der under indrullering af systemklienter der foretager registreringer i EDS angives følgende:

- Det *device_id* som stationen er registreret med i EER
- En liste af *SOR-organisationer* som stationen sender/modtager meddelelser for i form af SOR kode og GLN lokationsnummer

Under indrullering angives som scope element:

```
EDS system/AuditEvent.c
```

(Ovenstående `system/AuditEvent.c` syntaks er baseret på definitionen af scopes for FHIR ressourcer i [SMART].)

Metadata for en EDS systemklient

Udover de i afsnit 3.3.1 Metadata for systemklienter beskrevne metadata elementer skal ovenstående elementer angives på følgende form for systemklienter der skal indrulleres til at måtte foretage registreringer i EDS.

Metadata element	Beskrivelse
`ehmi:eer:device_id`	En angivelse af device_id som stationen er registreret med i EER.
`ehmi:org_context`	Array af JSON objekter bestående af `name` (organisationsnavn), `sor` (SOR kode) og `gln` (lokationsnummer) som stationen sender/modtager meddelelser for.

Eksempel metadata dokument for en EDS systemklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": "client_credentials",
  "client_name": "Apotekssystemet for Aarhus Åbyhøj Apoteket",
  "scope": "EDS system/AuditEvent.c",
  "contacts": [
    "døgnsupport@aarhus-aabyhoej-apoteket.dk",
    "+45 1234 5678"
  ],
  "tls_client_auth_subject_dn": "subject=CN=Apoteksleverandør Apo123's
systemcertifikat, serialNumber=UI:DK-O:G:a262681f-2e94-45c5-aaea-
aad4e9bc5768, O=Apoteksleverandør Apo123, organizationIdentifier=NTRDK-
12345678, C=DK",
  "ehmi:eer:device_id": "c4b8d3ea-b187-426b-be77-bffd9f593d84",
  "ehmi:org_context": [
    {
      "name": "Aarhus Åbyhøj Apotek",
      "sor": "306861000016006",
      "gln": "5790000173372"
    },
    {
      "name": "Bruun's Apotek",
      "sor": "625961000016008",
      "gln": "5790002275296"
    }
  ]
}
```

4.2.2 Indrullering/whitelisting af klienter til søgning og opslag

<TBD>

4.2.3 Kald til Token Endpoint

I tilgangen til EDS opereres der med organisations-specifikke tokens, dvs. klienter som optræder i flere organisatoriske kontekster skal trække et særskilt token hos Authorization Server for hver SOR/GLN kontekst.

For at få udstedt et access token til EDS angives følgende scopes:

scope	Beskrivelse
`EDS`	En angivelse af det er for EDS at klienter ønsker et access token.
`system/AuditEvent.c`	En angivelse at tokenet skal kunne registrere/oprette forsendelsesstatus ressourcer (som er profileringer af FHIR's AuditEvent ressource).
`SOR:<XXXXX>`	En angivelse af organisationens SOR kode, hvor `<XXXXX>` sættes til selve koden.
`GLN:<YYYYY>`	En angivelse af organisationens GLN lokationsnummer, hvor `<YYYYY>` sættes til selve lokationsnummeret.

Eksempel på en samlet scope som indgår i kaldet:

```
EDS system/AuditEvent.c SOR:306861000016006 GLN:5790000173372
```

Eksemplet på et kald til Token Endpoint med ovenstående eksempel scope (bemærk at kaldet foretages via 2-vejs TLS):

```
POST /token HTTP/1.1
Host: authorization.sunhedsdatastyrelsen.dk
User-Agent: curl/8.4.0
Accept: */*
Proxy-Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 973

grant_type=client_credentials&scope=EDS%20system%2FAuditEvent.c%20SOR%3A306861000016006%20GLN%3A5790000173372&client_id=0ba284d1-8974-4241-bce1-0498bc2d48ea
```

Valideringer af kaldet hos Authorization Server

Kaldet til Token Endpoint valideres hos Authorization Server, som validerer klientens TLS-klientcertifikat og tjekker at klienten er indrulleret/whitelistet med de angivne scopes. Authorization Serveren mapper således `client_id` fra kaldet til det registrerede device_id og validerer at klienten

er whitelistet til såvel EDS servicen, den angivne 'create' operation for `AuditEvent` ressourcen og den angivne organisatoriske kontekst i form af SOR kode og GLN lokationsnummer.

4.2.4 Kald til EDS

For at registrere en forsendelsesstatus opretter klienten lokal et `AuditEvent` FHIR objekt (som overholder `EdsPatientDeliveryStatus` eller `EdsBasicDeliveryStatus` profilen) og kalder forsendelsesstatusservicen med et HTTP POST kald, som har 'create' semantikken. I POST kaldet inkluderes access tokenet som beskrevet i ovenstående og det nyoprettede `AuditEvent` FHIR objekt placeres i HTTP body-delen.

Eksempel på kald til EDS med AuditEvent ressourcen angivet som JSON objekt (bemærk at kaldet foretages via 2-vejs TLS med samme klientcertifikat som ved kaldet til Authorization Server):

```
POST /base/AuditEvent HTTP/1.1
Host: ehmi.medcom.dk
User-Agent: curl/8.4.0
Accept: application/fhir+json
Content-Type: application/fhir+json
Content-Length: 11996
Authorization: Bearer eyJhb ... Dhi6g

{
  "resourceType" : "AuditEvent",
  "id" : " cbcb0de9-105e-470a-8754-ffad3b581ed4",
  "meta" : {
    "profile" : [
       "http://medcomehmi.dk/ig/dk-ehmi-
eds/StructureDefinition/EdsPatientDeliveryStatus"
    ]
  },

...

}
```

EDS adgangskontrol

Forsendelsesstatusservicen tjekker at access tokenet er gyldigt og validerer 'sender-contrained' egenskabet, det vil sige validerer, at det af klientens anvendte TLS-klientcertifikat matcher certifikatet som blev indlejret i access tokenet. Forsendelsesstatusservicen tjekker desuden at tokenet er udstedt til EDS som aftager af tokenet og indeholder de nødvendige scopes til at klienten må foretage registreringer i EDS (her "`EDS system/AuditEvent.c`").

EDS tjekker endvidere hvorvidt SOR koden, GLN lokationsnummeret og device_id som den kan uddrage af access tokenet, matcher oplysningerne i den medsendte AuditEvent ressource.

5. EHMI Addressing Service (EAS)

<TBD>

6. EHMI Endpoint Register (EER)

<TBD>

7. Appendiks: Arkitekturbeslutninger

I dette kapitel fastholdes begrundelserne for væsentlige arkitekturvalg som ligger til grund from udformning af denne sikkerhedsarkitektur.

7.1 Udgangspunkt for anvendelse af OAuth 2.0 til sundhedsområdet

Problemstilling	Hvilken OAuth 2.0 sikkerhedsprofil bør anvendelsen af OAuth 2.0 til det danske sundhedsvæsen basere sig på?
Antagelse	Til det generelle [OAuth] framework findes der en lang række yderligere specifikationer (typisk i form af [IETF-RFC]'er) som fastlægger token-formater, protokol-flows, klienttype-specifikke udvidelser, sikkerheds-tiltag mod kendte angreb mm. At basere en dansk profilering af OAuth til sundhedsområdet alene på de mange basis specifikationer vil være en kompleks opgave og vanskeligt at overskue for anvenderne. I stedet bør profileringen basere sig på en allerede eksisterende sikkerhedsprofilering af OAuth som bygger på 'best practice' på området.
Muligheder	Følgende sikkerhedsprofileringer er identificeret som mulige kandidater: - [FAPI] - [HEART] - [iGOV-OAuth] - [OIO-OIDC] - [SMART]
Analyse	[FAPI] specifikationen er forankret i OpenID Foundation og har sin oprindelse i bankverdenen og [PSD 2] EU-direktivet, men er i version 2.0 blevet generaliseret til at være bredt anvendelig i alle områder med høje sikkerhedskrav. Specifikationen fremstår operationel med klare formulerede krav til OAuth/OIDC aktørerne. Udgangspunkt for FAPI sikkerhedsspecifikationen er en angrebs-model og til specifikationen er der udformet et formelt bevis for, at FAPI kan modstå angrebene fra modellen. Specifikationen gør brug af nyere OAuth teknologier som [OAuth-DPOP] og [OAuth-PAR], hvoraf førstnævnte på nuværende tidspunkt har begrænset tool-support (men er heller ikke påkrævet i FAPI, hvor [OAuth-MTLS] kan benyttes i stedet). Det norske Helsenett har baseret deres HelseID løsning på FAPI 2.0. Erfaringerne fra Norge har vist, at FAPI 2.0 kan implementeres i praksis, men Norges tilvalg af [OAuth-DPOP] har krævet en del støtte til anvenderne. OpenID Foundation stiller desuden en FAPI conformance-

	testsuite til rådighed og står for certificering af produkter og tools som lever op til specifikationen. [HEART] som ligeledes er forankret under OpenID Foundation er en decideret sikkerhedsprofilering af OAuth 2.0 til sundhedsområdet, som er skabt til at understøtte andre profiler under HEART initiativet. Det ser ikke ud til at profilen er blevet vedligeholdt siden 2018 og der peges i [HEART] fortsat på flows som jf. 'best practice' på området ikke længere bør anvendes. I [iGOV-OAuth] fra OpenID Foundation er sikkerhedsprofileringen fra [HEART] blevet ajourført og generaliseret. I [iGOV-OAuth] tages der afsæt i velafprøvede og bredt anvendte specifikationer, samt at der åbnes op for nyere teknologier (som [OAuth-DPOP]). Både Holland og Italien har taget udgangspunkt i specifikationen i deres nationale OAuth profileringer og det engelske NHS har ligeledes baseret deres føderation på [iGOV-OAuth]. Specifikationen suppleres med en profilering af [OIDC] ('identitets-laget' ovenpå [OAuth]), se [iGOV-OIDC]. [OIO-OIDC] er Digitaliseringsstyrelsen bud på en OpenID Connect profilering. Specifikationen har ikke været anvendt i praksis og profileringen går netop kun på 'identitets-laget' OpenID Connect og angiver ikke hvordan rene system-til-system integrationer kan realiseres. [SMART] er HL7's profilering af OAuth i forhold til adgangsstyring af FHIR-snitflader. Specifikationen tager i høj grad afsæt i FHIR terminologien og fremstår operationel. Udover en general angivelse af hvordan OAuth bør anvendes i en FHIR kontekst, defineres en model for OAuth 'scopes' for adgangsstyring til FHIR ressourcer. I modsætning til de andre fire analyserede specifikationer fremstår SMART mindre stringent (fx mangler der delvis referencer til underliggende standarder/versioner) og har nogle steder mest karakter af en udviklervejledning. Derudover definerer SMART sin egen metadata-discovery-mekanisme, i stedet for at basere sig på standard mekanismerne.
Beslutning	Anvendelsen af OAuth 2.0 til sundhedsområdet tager udgangspunkt i [FAPI], som den mest stringente profilering. I anvendelsen tillades der derudover, at der i FHIR-baserede applikationer må benyttes mekanismerne fra SMART ('scope' definitioner, SMART-specifik metadata-discovery-mekanisme) for at være compliant med SMART specifikation.

7.2 Token-binding via applikations- og/eller transportlaget

Problemstilling	Hvilke(n) kryptografisk token-bindings mekanisme(r) bør anvendelsen af OAuth 2.0 til det danske sundhedsvæsen basere sig på?

Antagelse	Kryptografisk token-binding til klienten (’sender constraining’) er påkrævet i FAPI specifikationen.
Muligheder	FAPI specifikationen tillader følgende to mekanismer: • [OAuth-DPOP] med token-binding i applikationslaget • [OAuth-MTLS] med token-binding i transportlaget
Analyse	[OAuth-DPOP] med binding i applikationslaget giver umiddelbart den største deployment fleksibilitet, men har en betydelig iboende kompleksitet, særligt hvis der skal opnås høj beskyttelse mod replay-angreb. Som relativ ny teknologi er der på nuværende tidspunkt kun begrænset tool-support for OAuth-DPOP. [OAuth-MTLS] med binding til transportlaget bygger derimod på 2-vejs TLS-protokollen som har en bred produktunderstøttelse. OAuth-MTLS bør dermed være noget nemmere at anvende i praksis, på nær i mobile apps, som fx vil skulle selvudstede et certifikat for at have et instans-specifikt akkreditiv som kan registreres i Authorization Server i forbindelse med indrullering af klienten/app-instansen. Desuden har OAuth-MTLS den fordel, at det samtidig kan anvendes til klient-autentifikation, hvor der ved OAuth-DPOP skal anvendes en særskilt mekanisme (`private_key_jwt` mekanismen som defineret i [OIDC]).
Beslutning	Anvendelsen af FAPI på sundhedsområdet baserer sig i første omgang alene på OAuth-MTLS.

7.3 Selv-indeholdt token eller token reference

Problemstilling	Hvordan bør tokens overføres – i selv-indeholdt form eller som token reference?
Antagelse	I OAuth kan tokens overføres direkte eller som reference.
Muligheder	• Tokens overføres i selv-indeholdt form (som JWT’er) • Tokens overføres som reference og aftagerne (ressource serverne) benytter Authorization Serverens Tokeninspection Endpoint til at hente tokenet.
Analyse	’By-value’ overførsel i selv-indeholdt form er mere robust, idet aftagerne ikke skal integrere til Authorization Server, men har den ulempe, at tokenet er synligt for klienten. Derimod giver ’by-reference’ overførslen udover at beskytte tokenindholdet fra at være synligt for klienten også mulighed for at minimere dataoverførsel, idet referencen typisk vil fylde mindre end et signeret JWT.

	I [IDWS] profileringen af [WS-Trust], som tillader både token-referencer og selv-indeholdte tokens, er der valgt modellen med selv-indeholdte tokens for at få en enklere og mere robust anvendelse.
Beslutning	Tokens overføres 'by-value' i selv-indeholdt form.

7.4 Indhold af token til EDS: device_id, SOR kode og GLN

<TBD>

8. Appendiks: Relevante uddrag af FAPI

I dette dokument refereres der til konkrete afsnit af [FAPI] sikkerhedsprofilen, som er her gengivet for god ordens skyld. De dele af afsnittene som ikke finder anvendelse i EHMI services er markeret med ~~streg igennem~~.

8.1 FAPI afsnit *5.2.1 Requirements for all endpoints*

TLS connections shall be protected against network attackers. To this end, clients, authorization servers, and resource servers:

1. *shall only offer TLS protected endpoints and shall establish connections to other servers using TLS. TLS connections shall be set up to use TLS version 1.2 or later.*
2. *when using TLS 1.2, follow the recommendations for Secure Use of Transport Layer Security in [RFC7525].*
3. *should use DNSSEC to protect against DNS spoofing attacks that can lead to the issuance of rogue domain-validated TLS certificates.*
4. *shall perform a TLS server certificate check, as per [RFC6125].*

***NOTE**: Even if an endpoint uses only organization validated (OV) or extended validation (EV) TLS certificates, rogue domain-validated certificates can be used to impersonate the endpoints and conduct man-in-the-middle attacks. CAA records [RFC8659] can help to mitigate this risk*.

8.2 FAPI afsnit *5.2.2 Requirements for endpoints not used by web browsers*

1. *when using TLS 1.2, the server shall only permit the cipher suites listed in Section 5.2.2.1*
2. *when using TLS 1.2, the client should only permit the cipher suites listed in Section 5.2.2.1*
3. *When using the* `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256` *or* `TLS_DHE_RSA_WITH_AES_256_GCM_SHA384` *cipher suites, key lengths of at least 2048 bits are required.*

5.2.2.1. TLS 1.2 permitted cipher suites

- `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

8.3 FAPI afsnit *5.2.3 Requirements for endpoints used by web browsers*

Endpoints for the use by web browsers

1. *shall use methods to ensure that connections cannot be downgraded using TLS Stripping attacks. A preloaded [preload] HTTP Strict Transport Security policy [RFC6797] can be used for this purpose. Some top-level domains, like .bank and .insurance, have set such a policy and therefore protect all second-level domains below them.*
2. *when using TLS 1.2, shall only use cipher suites allowed in [RFC7525]*

8.4 FAPI afsnit *5.3.2.1 General Requirements*

Clients

1. *shall support sender-constrained access tokens using one of the following methods:*
    - *MTLS as described in [OAuth-MTLS]*
    - ~~*DPoP as described in [I-D.ietf-oauth-dpop]*~~
2. *shall support client authentication using one of the following methods:*
    - *MTLS as specified in section 2 of [OAuth-MTLS]*
    - ~~`private_key_jwt` *as specified in section 9 of [OIDC]*~~
3. *shall send access tokens in the HTTP header as in Section 2.1 of OAuth 2.0 Bearer Token Usage [RFC6750]*
4. *shall not expose open redirectors (see section 4.10 of [I-D.ietf-oauth-security-topics])*
5. ~~*if using* `private_key_jwt`*, shall use the Authorization Server's issuer identifier value (as defined in [RFC8414]) in the aud claim sent in client authentication assertions. The issuer identifier value shall be sent as a string not as an item in an array.*~~
6. *shall support refresh tokens and their rotation*
7. *if using MTLS client authentication or MTLS sender-constrained access tokens, shall support the* `mtls_endpoint_aliases` *metadata defined in [OAuth-MTLS]*
8. ~~*if using DPoP, shall support the server provided nonce mechanism (as defined in section 8 of [I-D.ietf-oauth-dpop])*~~
9. *shall only use authorization server metadata (such as the authorization endpoint) retrieved from the metadata document as specified in [OIDC] and [RFC8414]*
10. *shall ensure that the issuer URL used as the basis for retrieving the authorization server metadata is obtained from an authoritative source and using a secure channel, such that it cannot be modified by an attacker*
11. *shall ensure that this issuer URL and the issuer value in the obtained metadata match*

~~***NOTE:***~~

~~*This profile may be used by Confidential Clients on a user-controlled device where the system clock may not be accurate, this may cause* `private_key_jwt` *client authentication to fail. In such circumstances a Client should consider using the HTTP Date header returned from the server to synchronise it's own clock when generating client assertions.*~~

~~***NOTE:***~~

~~*Although Authorization Servers are required to support "Authorization Code Binding to DPoP Key" (as defined by section 10.1 of [I-D.ietf-oauth-dpop]), clients are not required to use it.*~~

8.5 FAPI afsnit *5.3.2.2 Authorization Code Flow*

For the Authorization Code flow, Clients

1. *shall use the authorization code grant described in [OAuth]*
2. *shall use pushed authorization requests according to [OAuth-PAR]*
3. *shall use PKCE [OAuth-PKCE] with* `S256` *as the code challenge method*

4. *shall check the* `iss` *parameter in the authorization response according to [RFC9207] to prevent Mix-Up attacks*

9. Referencer

- [EHMI] "Ny infrastruktur (EHMI)", https://medcom.dk/projekter/kommunale-proevesvar-paa-ny-infrastruktur/kommunale-proevesvar-ny-infrastruktur-ehmi/
- [FAPI] "FAPI 2.0 Security Profile", https://openid.net/specs/fapi-2_0-security-profile-ID2.html
- [HEART] "Health Relationship Trust Profile for OAuth 2.0", https://openid.net/specs/openid-heart-oauth2-1_0.html
- [IETF-RFC] "The Internet Engineering Task Force (IETF) - RFCs", https://www.ietf.org/standards/rfcs/
- [IDWS] "OIO Identity Based Web Services 1.2 (OIO IDWS)", https://digst.dk/it-loesninger/standarder/oio-identity-based-web-services-12-oio-idws/
- [I-D.ietf-oauth-security-topics], "OAuth 2.0 Security Best Current Practice", https://www.ietf.org/archive/id/draft-ietf-oauth-security-topics-21.txt
- [iGOV-OAuth] "International Government Assurance Profile (iGov) for OAuth 2.0", https://openid.net/specs/openid-igov-oauth2-1_0.html
- [iGOV-OIDC] "International Government Assurance Profile (iGov) for OpenID Connect 1.0", https://openid.net/specs/openid-igov-openid-connect-1_0.html
- [JTP-H] "JWT Token Profile for Healthcare (JTP-H)"
- [JWKS] "JSON Web Key (JWK)", https://datatracker.ietf.org/doc/html/rfc7517
- [JWT] "JSON Web Token (JWT)", https://datatracker.ietf.org/doc/html/rfc7519
- [OAuth] "The OAuth 2.0 Authorization Framework", https://datatracker.ietf.org/doc/html/rfc6749
- [OAuth-DCR] "OAuth 2.0 Dynamic Client Registration Protocol", https://datatracker.ietf.org/doc/html/rfc7591
- [OAuth-DPOP] "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", https://datatracker.ietf.org/doc/html/rfc9449
- [OAuth-MTLS] "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", https://datatracker.ietf.org/doc/html/rfc8705
- [OAuth-PAR] "OAuth 2.0 Pushed Authorization Requests", https://datatracker.ietf.org/doc/html/rfc9126
- [OAuth-PKCE] "Proof Key for Code Exchange by OAuth Public Clients", https://datatracker.ietf.org/doc/html/rfc7636
- [OCES] "OCES (Offentlige certifikater til Elektronisk Service)", https://certifikat.gov.dk/politikker.for.tillidstjenester/ og https://mitid-erhverv.dk/avanceret/certifikater/
- [OIDC] "OpenID Connect Core 1.0", https://openid.net/specs/openid-connect-core-1_0.html
- [OIO-OIDC] "OIO Open ID Connect Profiles Version 0.91", https://digst.dk/media/24669/oio-oidc-profiles-v091.pdf
- [preload] "HSTS Preload List Submission", https://hstspreload.org/
- [RFC6125] "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", https://datatracker.ietf.org/doc/html/rfc6125
- [RFC6797] "HTTP Strict Transport Security (HSTS)", https://datatracker.ietf.org/doc/html/rfc6797
- [RFC7525] "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", https://datatracker.ietf.org/doc/html/rfc7525

- [RFC8414] “OAuth 2.0 Authorization Server Metadata”, https://datatracker.ietf.org/doc/html/rfc8414
- [RFC8659] “DNS Certification Authority Authorization (CAA) Resource Record”, https://datatracker.ietf.org/doc/html/rfc8659
- [RFC9207] “OAuth 2.0 Authorization Server Issuer Identification”, https://datatracker.ietf.org/doc/html/rfc9207
- [PSD 2] ”PSD 2 Direktiv”, https://www.finanstilsynet.dk/Lovgivning/Ny_EU_lovsamling/PSD-2
- [SMART] ”SMART App Launch 2.1.0”, http://hl7.org/fhir/smart-app-launch/index.html
- [WS-Trust] “WS-Trust 1.4”, https://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html