Sikkerhedsarkitektur for Enhanced Healthcare Messaging Infrastructure (EHMI) services

2025-01-09, 0.98

medcom

SUNDHEDSDATA-
STYRELSEN

Id og proces	Id og titel	Initialer	Version	Dato
SOP-4.1 - Udarbejdelse og ændring af MedCom standard	Sikkerhedsarkitektur – EMHI services	CHG	0.98	2025-01-09

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
0.3	CHG	2024-06-18	Opdateret med beskrivelse for brugerscenarier mm.
0.4	CHG	2024-06-26	Omstrukturering af indhold
0.5	CHG	2024-08-21	Efter review fra ASHA
0.98	CHG	2025-01-09	Efter review fra OVI, opdateret til FAPI 2.0 RC

Indhold

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)

EHMI sikkerhedsarkitekturen skal tilvejebringe en ensartet og standardiseret måde hvorpå parterne der indgår i EHMI meddelelsesforsendelser kan benytte tjenesterne i EHMI infrastrukturen.

1.1 Publikum

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

Projektledere og andre projektdeltagere kan eventuel orientere sig om sikkerhedsemnerne i baggrundsafsnittet (afsnit 2).

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.

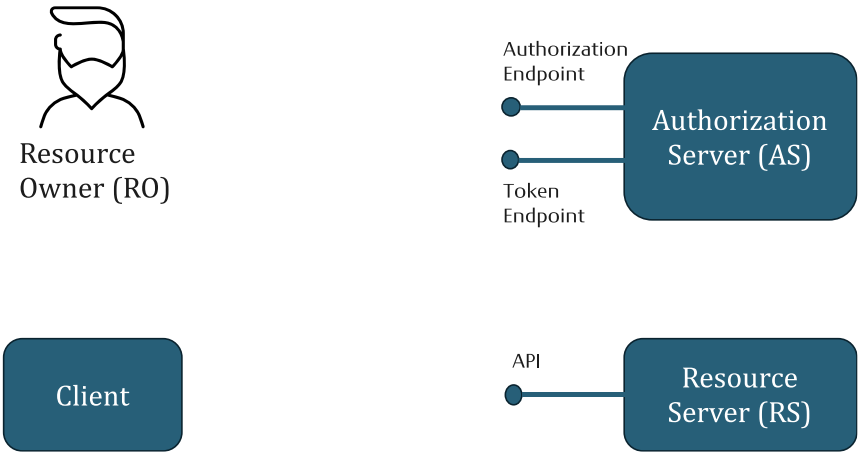
2. Baggrund

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. Sikkerhedsarkitekturen baserer sig endvidere på [JTP-H] som er sundhedsvæsnets profilering af JSON Web Tokens [JWT].

2.1 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'.

Selvom det primære formål med OAuth er at definere protokoller for autorisation af klienter, indgår der også elementer af autentifikation. I de fleste flows skal selve klienten autentificere sig som en del af kaldende til Authorization Server og OAuth definerer således forskellige mekanismer til klient-autentifikation. Ligeledes vil Authorization Serveren afkræve autentifikation af brugeren i de flows hvor brugeren indgår. OAuth forholder sig dog ikke til hvordan brugerautentifikation tilvejebringes.

2.1.1 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.

2.1.2 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 af enhver som er i besiddelse af 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-constrained tokens, baseret på binding af tokenet til transportlaget (via [OAuth-MTLS]) eller til applikationslaget (via [OAuth-DPOP]).

2.1.3 Refresh tokens

Access tokens udstedes til en klient efter brugeren har autoriseret klienten og udstedes med begrænset levetid, typisk kort, for at minimere risiko for misbrug. For at undgå at brugeren skal autorisere klienten på ny ved senere kald til Ressource Serveren, kan en Authorization Server sammen med access tokenet udstede et såkaldt *refresh token* med længere levetid. Klienten kan senere benytte refresh tokenet til at få udstedt et nyt access token ved Authorization Serveren, uden at brugeren skal involveres.

2.1.4 Scopes og claims

I OAuth dækker begrebet *scope* over de privilegier, som brugeren kan autorisere klienten til. Scope kan dække over udlevering af bruger-attributter, adgang til en Ressource Server og over de operationer der kan foretages ved Ressource Server.

Under token-udstedelse foretager Authorization Server typisk en mapning af scopes som klienten er autoriseret til (og har requestet i kaldet til Authorization Server) til konkrete *claims*, som er de attributter, der indgår i tokenet.

2.2 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). Eksempelvis udsteder en Authorization Server typisk både et identity token og et access token til en mobil app. App'en vil efterfølgende ofte anvende oplysningerne fra identity tokenet til visning i dens brugergrænsefladen (fx brugerens navn), men benytte access tokenet til at kalde en eller flere backend services.

3. Sikkerhedsmodel

Udgangspunktet for sikkerhedsmodellen for EHMI services er OAuth-sikkerhedsprofileringen [FAPI], se appendiks 4.1 Udgangspunkt for anvendelse af OAuth 2.0 til sundhedsområdet for en gennemgang af rationalet for valget af FAPI profilen.

FAPI 2.0 sikkerhedsprofilen er i skrivende stund i public review med henblik på blive løftet til en '*OpenID Final Specification*'.

Sikkerhedsmodellen er baseret på høringsversionen af FAPI 2.0 og bliver opdateret til 'final' udgaven når denne foreligger.

I FAPI sikkerhedsprofilen er det påkrævet at benytte sender-constrained tokens (se afsnit 2.1.2), 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 4.2 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 af tokens tager derimod i EHMI afsæt i det danske sundhedsvæsnets profilering af [JWT] tokens, se [JTP-H].

Sikkerhedsmodellen baserer sig på [NSIS] for de anvendelser, som involverer en menneskelig brugers adgang til EHMI services[1]. Adgang til EHMI services som udstiller følsomme persondata forudsætter NSIS-sikringsniveau 'Betydelig'.

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.

Den konkrete anvendelse af den generelle sikkerhedsmodel i de tre EHMI services EDS, EAS og EER er beskrevet i appendiks 7.

3.1 Klienttyper

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

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

[1] NSIS regulerer kun menneskelig brugeres adgang til tjenester, men forholder sig ikke til systemers autentifikation i system-til-system interaktioner. For adgang til ikke følsomme data i EHMI-services som kan foregå som systemkald anvendes NIST-sikringsniveauerne som angivet i [JTP-H].

servicen som tilgår postkasseregisteret eller et fagsystem som tilgår forsendelsesstatus-servicen.

2. En *fuld klient med brugerdelegering* (eller bare *brugerklient* i det følgende*)* er 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 2.1.1), som [FAPI] sikkerhedsmodellen tager udgangspunkt i.

EHMI services understøtter 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[2], 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 beskrevet i ovenstående 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 vil det være oplagt at etablere en

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

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, 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.

3.3.1 Metadata for klienter

Følgende elementer skal angives i klientmetadata-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`	Et array med en angivelse af hvilke OAuth flows klienten anvender. Sættes henholdsvis til den faste værdi `["client_credentials"]` for systemklienter og til `["authorization_code", "refresh_token"]` for brugerklienter.
`client_name`	Et sigende navn for klienten, som letter administrations-opgaven 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`. Se også afsnit 2.1.4 Scopes og claims.
`contacts`	Et array med kontaktoplysninger (typisk e-mailadresser eller telefonnumre) til organisationen, som er ansvarlig for driften af klienten.

Metadata element	Beskrivelse
`tls_client_auth_subject_dn`	Subject Distinguished Name fra OCES systemcertifikatet som anvendes som TLS-klientcertifikat, fx `subject=CN=Korsbæk Kommune test systemcertifikat, serialNumber=UI:DK-O:G:9b996be1-b439-45ab-b239-0c95d8e02aee, O=Korsbæk Kommune, organizationIdentifier=NTRDK-11111111, C=DK` (Der anbefales at OCES systemcertifikater ikke udstedes med et certifikatspecifikt UUID[3], 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).
`redirect_uris`	(Skal kun angives for brugerklienter, men ikke for systemklienter) Et array med en angivelse af hvilke URI'er Authorization Server må redirecte brugerens browser til efter kaldet til Authorization Serverens Authorization Endpoint (se også afsnit 3.4.2 Brugerkalds-scenariet (borgere/fagpersoner)).

[3] 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.crs",
  "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"
}
```

3.4 Integrations-flows og protokoller

I både systemkalds- og brugerkalds-scenarierne foregår klient autentifikation via OAuth `tls_client_auth` autentifikationsmetoden som defineret i [OAuth-MTLS]. Se i øvrigt gennemgangen under appendiks 4.2 Token-binding via applikations- og/eller transportlaget.

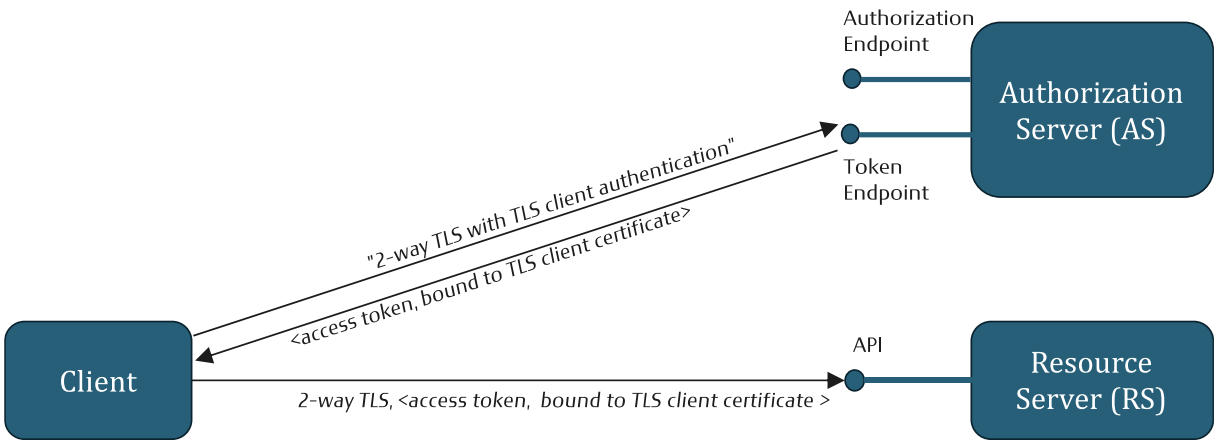
OBS: Der findes en lang række forskellige OAuth kodebiblioteker[4], som implementer de standard-flows og mekanismer, som er defineret i de forskellige OAuth-relaterede specifikationer, som de her beskrevne flows baserer sig på. I en praktisk implementering kan der med fordel tages udgangspunkt i standard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

I det følgende gennemgås først trin for trin integrations-flows for anvendelsesscenarierne 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.

[4] Se fx https://oauth.net/code/ og særligt det FAPI 2.0 certificerede https://github.com/panva/oauth4webapi



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 tidligere 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 faste værdi `client_credentials`.
`scope`	Ønskede scope(s). Se beskrivelsen i Appendiks: Anvendelse af sikkerhedsmodellen i EHMI services for de konkrete scopes der skal angives for de enkelte EHMI services. Authorization Server benytter `scope` værdien til at fastlægge audience/aftager og indhold af access tokenet som den udsteder. Se også afsnit 2.1.4 Scopes og claims.
`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. Bemærk, at `client_credentials` akkreditivet (klientcertifikatet) anvendes på transportniveau som en del [OAuth-MTLS] og derfor ikke indgår i nogen form som kaldsparameter.

Eksempel kald:

```
POST /token HTTP/1.1
Host: authorization.sundhedsdatastyrelsen.dk
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 92

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

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 faste værdi `Bearer` i EHMI. **OBS**: Bemærk at tokenet ikke er et bearer token, men er sender-constrained til TLS-klientcertifikatet. (I OAuth familien af specifikationer er der ikke defineret en særskilt `token_type` for TLS-bundne tokens, men der foreskrives at benytte værdien `Bearer`).
`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.

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 2: Kald EHMI service
Klienten tilgår EHMI servicen 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>
```

Bemærk igen at selvom der skal angives 'Bearer', så er tokenet ikke et bearer token, men er sender-constrained til TLS-klientcertifikatet.

Eksempel kald til en EHMI service:

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

{
...
}
```

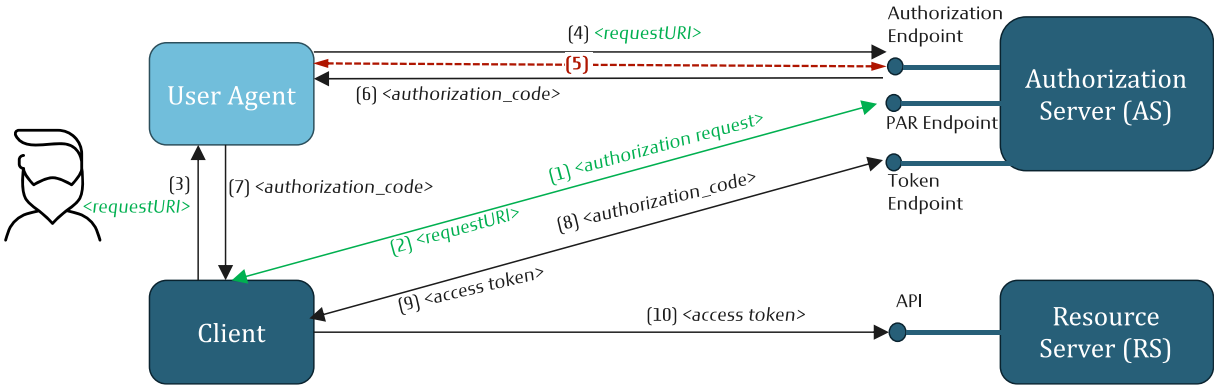
3.4.2 Brugerkalds-scenariet (borgere/fagpersoner)

Brugerkalds-scenariet tager udgangspunkt i OAuth Code Flowet (se afsnit 5.1 Klassisk OAuth Code Flow), hvor autorisationskoden beskyttes med mekanismen fra [OAuth-PKCE] (se afsnit 5.2 OAuth Code Flow med PKCE).

I modsætning til det klassiske OAuth Code Flow sendes request parametre ikke via browseren til Authorization Server, men overføres via [OAuth-PAR] mekanismen direkte fra klienten til Authorization Serveren i et forudgående kald.

Som i systemkalds-scenariet benytter klienten også i brugerkalds-scenariet sit OCES TLS-klientcertifikat som autentifikations-mekanisme i kaldene til Authorization Serveren (til PAR Endpointet og til Token Endpointet) og til kald til EHMI services.

Det samlede flow er illustreret i nedenstående Figur 3, som i det følgende gennemgås trin for trin.



Figur 3: OAuth Code Flow med Pushed Authorization Request (PAR)

Skridt 1 & 2: Kald af PAR Endpoint hos Authorization Server
Klienten danner en tilfældig PKCE engangsnøgle, tilgår Authorization Serveren via 2-vejs TLS med TLS-klientcertifikatet som er blevet registreret i Authorization Server og laver et HTTP POST kald til PAR Endpoint med angivelse af følgende kaldsparametre:

Parameter	Beskrivelse
`response_type`	Sættes til den faste værdi `code`.
`client_id`	Sættes til den client_id som blev tildelt klienten under indrullering ved Authorization Server.
`redirect_uri`	Sættes til den URI som Authorization Serveren skal redirecte brugerens browser til efter det efterfølgende kald til Authorization Endpoint. Den angivne `redirect_uri` skal være blevet registreret i Authorization Server under indrullering af klienten.
`scope`	Ønskede scope(s). Se beskrivelser af sikkerhedsmodellen for de enkelte EHMI services i appendiks 7 for de konkrete scopes der skal angives. For at klienten får et OIDC identity token med oplysninger om brugeren inkluderes scopet `openid`.
`state`	Sættes til en tilfældigt generet streng, som klienten skal benytte til at matche redirect-callback kaldet fra Authorization Serveren

Parameter	Beskrivelse
	(skridt 7 i ovenstående Figur 3) med requestet. Mekanismen beskytter desuden mod CSRF[5]-angreb.
`code_challenge`	Sættes til hash-værdien (beregnet med SHA256 algoritmen) i base64url-encoded form af den tilfældige PKCE engangsnøgle som klienten har genereret.
`code_challenge_method`	Sætte til den faste værdi `S256` (som er SHA256 algoritmen).

Eksempel kald:

```
POST /authorize/par HTTP/1.1
Host: authorization.sundhedsdatastyrelsen.dk
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 274

response_type=code&client_id=0ba284d1-8974-4241-bce1-0498bc2d48ea&
redirect_uri= https%3A%2F%2Fehmi-trackntrace-portal.dk%2Fcallback&
scope=EDS%20openid&state= UYAvv-myWe8HYAvv-mH_yy2irpl&code_challenge=
hfvQEUKr592yejsy286NmFkHjDlEH4dyIJwDgqLTGJI&code_challenge_method=S256
```

Retursvar
Svaret fra PAR Endpoint består af en JSON struktur med følgende værdier:

Returværdi	Beskrivelse
`request_uri`	En engangs-URI, som benyttes som parameter i det efterfølgende browser-kald til Authorization Endpoint.
`expires_in`	Gyldighed af ovenstående `request_uri` i sekunder.

[5] https://en.wikipedia.org/wiki/Cross-site_request_forgery

Eksempel response fra Authorization Server:

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8

{
  "request_uri":
     "urn:ietf:params:oauth:request_uri:6esc_11ACC5bwc014ltc14eY22c",
  "expires_in": 60
}
```

Skridt 3 til 7: Direct/Redirect via Browser til Authorization Endpoint ved Authorization Server
Klienten starter browseren (eller redirecter browseren for browser-baserede klienter) med et HTTP GET kald til Authorization Endpoint ved Authorization Server med følgende kaldsparametre.

Parameter	Beskrivelse
`client_id`	Sættes til den client_id som blev tildelt klienten under indrullering ved Authorization Server.
`request_uri`	Sættes til den engangs-URI som klienten har modtaget fra kaldet til Authorization Serverens PAR Endpoint.

Eksempel browser-kald til Authorization Endpointet:

```
GET https:// authorization.sundhedsdatastyrelsen.dk/authorize?
client_id=0ba284d1-8974-4241-bce1-0498bc2d48ea&request_uri=
urn%3Aietf%3Aparams%3Aoauth%3Arequest_uri%3A6esc_11ACC5bwc014ltc14
eY22c
```

Authorization Serveren sørger for at brugeren autentificerer sig (se 5.3 Brugerautentifikation i brugerklienter), og at brugeren autoriserer klienten. Derefter redirecter Authorization Serveren brugerens browser til callback `redirect_uri` (som klienten har angivet i kaldet til PAR Endpointet) med følgende parametre:

Parameter	Beskrivelse
`code`	Engangskoden som klienten skal benytte til det efterfølgende kald til Token Endpointet.
`state`	Værdien som klienten angav i PAR-requestet, som skal benyttes til at matche callback'et til requestet.

Eksempel browser-redirect til klientens registrerede redirect_uri:

```
HTTP/1.1 302 Found
Location: https://ehmi-trackntrace-portal.dk/callback?
code=QSOBeZQbYS6plxlYWxSbIA&state= UYAvv-myWe8HYAvv-mH_yy2irpl
```

Skridt 8 og 9: Kald af Token Endpoint ved 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 faste værdi `authorization_code`.
`code`	Sættes til engangskoden som klienten har modtaget i callback'et fra Authorization Endpointet.
`redirect_uri`	Sættes til den samme `redirect_uri` værdi som i det foregående kald til PAR Endpointet[6].
`client_id`	Sættes til den client_id som blev tildelt klienten under indrullering ved Authorization Server.
`code_verifier`	Sættes til PKCE engangsnøglen som klienten har genereret forud for kaldet til PAR endpointet.

[6] Formålet med at lade klienten medsende `redirect_uri` i kaldet til Token Endpointet er at beskytte mod "Authorization Code Redirection URI Manipulation" angrebet. PAR mekanismen beskytter i sig selv mod denne type angreb, men en række Authorization Server produkter forventer, at `redirect_uri` altid angives i kaldet til Token Endpointet for authorization code flow'et.

Eksempel kald:

```
POST /token HTTP/1.1
Host: authorization.sundhedsdatastyrelsen.dk
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 267

grant_type=authorization_code&code=QSOBeZQbYS6plxlYWxSbIA&redirect_uri=
https%3A%2F%2Fehmi-trackntrace-portal.dk%2Fcallback&client_id=0ba284d1-
8974-4241-bce1-0498bc2d48ea&code_verifier= 9HumtLsQIHF0-
d9jIvOMurRBV5tKcP1bLAAN3mTIiLuyDkXvZpCUfGLA3lC_V4jBMbcM3AaPhBGOk8oy
```

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`[7] i EHMI.
`expires_in`	Access tokenets gyldighed i sekunder.
`id_token` (Optionel returværdi)	Identity tokenet med oplysninger til klienten om brugeren. Inkluderes såfremt klienten også har angivet scope `openid` i requestet.
`refresh_token`	Et refresh token som klienten kan benytte til at få udstedt et nyt access token, når dette er udløbet uden at skulle involvere brugeren på ny.

[7] 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.

Returværdi	Beskrivelse
`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.

Eksempel response fra Authorization Server:

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

{
  "access_token":"eyX ... Dpg",
  "token_type":"Bearer",
  "expires_in":300,
  "id_token":"eyJ ... sVA"
  "refresh_token":"MwibmFtZSI6NTY3ODkwIiIkpvjM0a"
}
```

Skridt 10: Kald af Resource Server
Kaldet til Resource Server foretages på præcis samme måde som i systemkalds-scenariet beskrevet tidligere, det vil sige med 2-vejs TLS og access tokenet i en `Authorization` HTTP header.

For eksempel:

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

{
...
}
```

Eventuelt senere skridt: Kald af Token Endpoint med refresh token
Når access tokenet er udløbet og brugeren fortsat har en aktiv session (i henhold til EHMI infrastrukturens timeout politik), kan klienten få udstedt et nyt access token fra Authorization Serveren uden at skulle involvere brugeren.

Et nyt access tokenet fås ved kald til Token Endpointet via 2-vejs TLS med TLS-klientcertifikatet som er blevet registreret i Authorization Server (refresh tokenet er ligeledes bundet til klientens certifikat) med følgende parametre:

Parameter	Beskrivelse
`grant_type`	Sættes til den faste værdi `refresh_token`.
`refresh_token`	Sættes til refresh tokenet som klienten har modtaget fra Authorization Server.
`scope` (optionel parameter)	Sættes til de ønskede scope(s). De ønskede scope(s) må ikke være større end scope fra det oprindelige access token. Hvis parameteren ikke angives sætter Authorization Server samme scope som for det oprindelige access token.

Klienten URL-encoder parameterværdierne, sætter dem sammen og laver et HTTP POST kald til Token Endpointet.

Eksempel kald:

```
POST /token HTTP/1.1
Host: authorization.sundhedsdatastyrelsen.dk
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 68

grant_type=refresh_token&refresh_token=MwibmFtZSI6NTY3ODkwIiIkpvjM0a
```

Retursvar

Svaret fra Token Endpoint består af en JSON struktur med et nyt access token, samt information om tokenet. Bemærk, at der i EHMI services ikke opereres med refresh token rotation, det vil sige at klienten ikke får udstedt et nyt refresh token, men kan benytte det samme refresh token så længe dette er gyldigt (hvorefter brugeren skal autorisere klienten på ny).

Følgende værdier returneres:

Returværdi	Beskrivelse
`access_token`	Access tokenet som klienten har requestet. Benyttes til efterfølgende kald til en EHMI service.

Returværdi	Beskrivelse
	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`[8] i EHMI.
`expires_in`	Access tokenets gyldighed i sekunder.
`id_token` (Optionel returværdi)	Identity tokenet med oplysninger til klienten om brugeren. Inkluderes såfremt klienten også har angivet scope `openid` i (det oprindelig) request.
`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.

3.4.3 Omvekslings-scenariet

På sigt (efter EHMI produktionspiloten) vil sikkerhedsarkitekturen også understøtte brugsscenarier, hvor brugere tilgår EHMI services fra portaler eller apps fra andre parter. Brugerautentifikation og autorisation af klienten (portalen, app'en) vil i disse anvendelsesscenarier givetvis foregå hos en anden part end ved EHMI infrastrukturens Authorization Server.

Adgang til EHMI services vil i disse situationer kunne realiseres via token-omveksling hos EHMI infrastrukturens Authorization Server, hvor denne truster tokens udstedt af tredje parten, som har autentificeret brugeren og autoriseret klienten.

Denne token-omveksling vil blive baseret på [OAuth-TKEX].

3.4.4 Delegerings-scenariet

På den længere bane kunne sikkerhedsarkitekturen udvides med anvendelsesscenarier der involverer delegering af rettigheder, eksempelvis brugen af fuldmagt eller til at understøtte den sundhedsfaglig medhjælp.

3.5 Token-indhold

JWT access tokens og identity tokens som Authorization Server udsteder følger [JTP-H] profilen og indeholder følgende claims:

[8] 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.

Claim	Beskrivelse
`iss`	Udstederen (issuer) af tokenet i form af en URI, dvs. en URI for Authorization Serveren i EHMI infrastrukturen
`jti`	Et unikt ID for JWT'en
`sub`	Et unik ID for (system-)brugeren på formen angivet i [JTP-H]. I EHMI anvendes der ikke subjekt ID'er af typen 'transient'
`aud`	En URI som identificerer aftageren af tokenet. For access tokens er det et ID for servicen som tilgås fx `https://eds.ehmi.dk` og for identity tokens er det et ID for brugerklienten fx `https://trackntrace.ehmi.dk`
`exp`	Udløbstidspunktet for tokenet, angivet som antal sekunder efter 1970-01-01T00:00:00Z UTC ("Seconds Since the Epoch")
`iat`	Udstedelsestidspunktet for tokenet, angivet som antal sekunder efter 1970-01-01T00:00:00Z UTC ("Seconds Since the Epoch")
`auth_time`	Autentifikationstidspunktet for (system-)brugeren, angivet som antal sekunder efter 1970-01-01T00:00:00Z UTC ("Seconds Since the Epoch")

`acr`	(System-)Brugerens autentifikationsniveau angivet som specificeret i [JTP-H].
	
	Angives som NSIS sikringsniveau for menneskelige brugere, fx
	
	`https://data.gov.dk/concept/core/nsis/loa/Substantial`
	
	Og som NIST sikringsniveau for systembrugere, fx
	
	`urn:dk:healthcare:loa:3`

`iss_policy`	En URI som identificerer issuance policy'en for Authorization Serveren. Kan benyttes til at skelne mellem forskellige policies som ligger til grund for token-udstedelsen. I EHMI infrastrukturen anvendes der kun een issuance policy.
`name`	(Kun ved brugerklienter) Brugerens fulde navn
`cpr`	(Kun ved brugerklienter) Brugerens CPR nummer

Claim	Beskrivelse
`cvr`	(Kun ved systemklienter eller ved brugerklienter når brugeren er medarbejder) Organisationens CVR nummer
`org_name`	(Kun ved systemklienter eller ved brugerklienter når brugeren er medarbejder) Organisationens navn hørende til CVR nummer
`priv`	(Kun ved brugerklienter når brugeren er medarbejder) OIO-BPP struktur med brugerens adgangsgivende roller. Optionel. Anvendes ikke i identity tokens.
`scope`[9]	En angivelse af de scopes som klienten er blevet autoriseret til. Anvendes ikke i identity tokens.
`cnf`	Et element, som indeholder et thumbprint af det X509 certifikat som tokenet er sender-constrained til, se [OAuth-MTLS]. Anvendes ikke i identity tokens.

[9] Claimet er ikke en del af [JTP-H], men påkrævet i [SMART].

Eksempel access token for en systemklient:


```
{
  "iss": "https://auth.sundhedsdatastyrelsen.dk",
  "jti": "18331e42-62f8-4eec-a9c5-cf3b271d30d0",
  "sub": "urn:dk:healthcare:eid:uuid:persistent:system:1456468e-abff-
44ef-86fb-ee9e4745c063",
  "aud": "https://eds.ehmi.dk",
  "exp": 1718873129,
  "iat": 1718872529,
  "auth_time": 1718872321,
  "acr": "urn:dk:healthcare:loa:3",
  "iss_policy": "urn:dk:ehmi:policy:fapi-strict",
  "cvr": "55133018",
  "org_name": "Aarhus Kommune",
  "scope": "EDS system/AuditEvent.crs SOR:193071000016008
GLN:5790000160921"
  "cnf": {
    "x5t#S256": "e7mOweqDr2dXm9+ZQ1LPZM+NYXQSzqvvnIx6qmeSupE"
  }
}
```


3.6 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.

3.6.1 Krav til alle aktører

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].

Klienter, Authorization Server og EHMI services (Ressource Servere) skal desuden overholde kravene i afsnit *5.4 Cryptography and Secrets* i [FAPI].

3.6.2 Yderligere krav til 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 proprietær 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 (tokenlevetid og scopes) i svaret fra Token Endpoint.

3.6.3 Yderligere krav til Authorization Servere

Authorization Servere (i EHMI infrastrukturen er der kun én) skal følge kravene i afsnit *5.3.1 Requirements for Authorization Servers* i [FAPI], men kan se bort fra krav som vedrører DPoP eller `private_key_jwt`.

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

3.6.4 Yderligere krav til Resource Servere (EHMI services)

Resource Servere (EHMI services EDS, EAS og EER) skal følge kravene i afsnit *5.3.3 Requirements for Resource Servers* i [FAPI], men kan se bort fra krav som vedrører DPoP eller `private_key_jwt`.

Alle EHMI services skal basere adgangsstyring på oplysningerne fra access tokenet i kaldet til servicen.

4. Appendiks: Arkitekturbeslutninger

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

4.1 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 basisspecifikationer 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 omfattende FAPI

	open-source 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 er den mest stringente profilering. I modsætning til [iGOV-OAuth] findes der direkte værktøjsunderstøttelse af [FAPI] i form af FAPI-certificerede produkter og kodebiblioteker, samt en conformance-testsuite, som kan benyttes til verifikation af FAPI-baserede løsninger. 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.

4.2 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, hvilket vurderes som en teknisk overkommelig opgave i praksis. OAuth-MTLS har desuden 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.

4.3 Selv-indeholdt token eller token reference

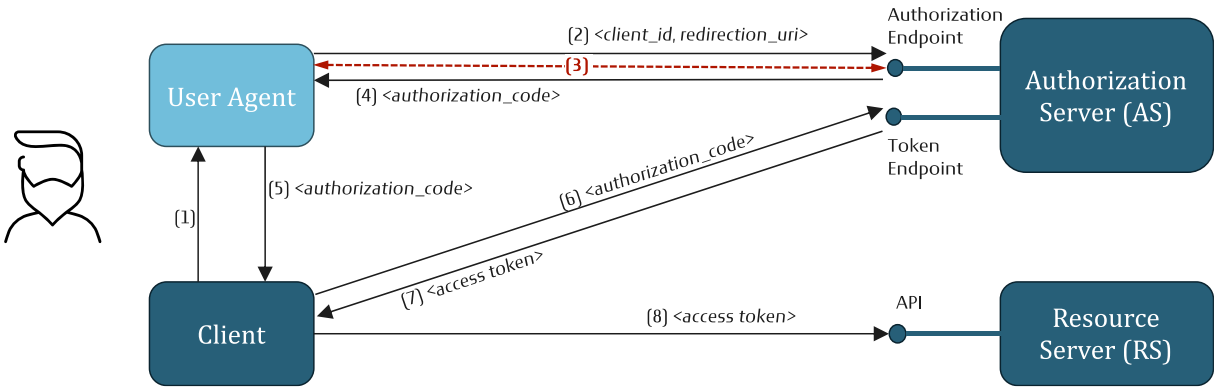
Problemstilling	Hvordan bør tokens overføres – i selv-indeholdt form eller som token reference?
Antagelse	I OAuth kan tokens overføres direkte eller som reference.
Muligheder	• Tokens overføres i selv-indeholdt form (som JWT’er) • Tokens overføres som reference og aftagerne (ressource serverne) benytter 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.

5. Appendiks: Eksempler og uddybninger

5.1 Klassisk OAuth Code Flow

Et standard OAuth Code Flow fra en klient med brugerdelegering består af trinene illustreret i nedenstående Figur 4.



Figur 4: Klassisk OAuth Code Flow

De enkelte trin er:

- Trin (1) og (2): Klienten directer browseren (et HTTP GET kald) til Authorization Serverens Authorization Endpoint med dens `client_id` og en `redirection_uri` som parametre.
- Trin (3): Brugeren autentificerer sig på en ikke nærmere bestemt vis (OAuth forholder sig ikke til autentifikation) og autoriserer klienten til må tilgå ressourcer.
- Trin (4) og (5): Authorization Serveren redirecter brugerens browser tilbage til klienten (til den medsendte `redirection_uri`) med en `authorization_code`.
- Trin (6) og (7): Klienten veksler `authorization_code` til et access token.
- Trin (8): Klienten benytter access tokenet til at tilgå ressource serverens API.

5.2 OAuth Code Flow med PKCE

I ovenstående OAuth Code Flow kan en angriber, som kan opsnappe en `authorization_code` selv veksle denne til et access token.

I OAuth kan tokens bindes til en klient (se afsnit 2.1.2 Bearer tokens og sender-constrained tokens), således at risikoen for misbrug mindskes. Og `authorization_code` kan ligeledes bindes til klienten gennem mekanismen defineret i [OAuth-PKCE].

I flowet med PKCE (som står for 'Proof Key of Code Exchange') generer klienten en tilfældig nøgle og medsender hash-værdien af nøglen (og angiver hash-algoritmen) i trin (1) og (2) i ovenstående Figur 4 til Authorization Serveren.

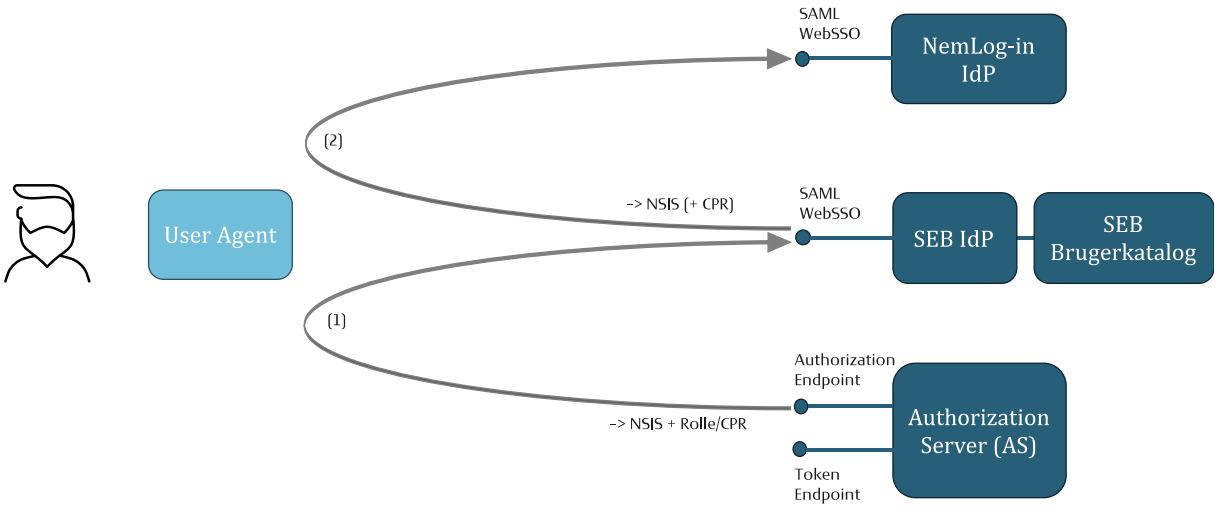
Når klienten i trin (6) veksler `authorization_code` til et access token, medsender klienten den tilfældige nøgle, som den havde genereret. Authorization Serveren validerer at hash-værdien modtaget i trin (2) matcher den tilfældig nøgle – ellers afbrydes flowet med en fejl.

5.3 Brugerautentifikation i brugerklienter

Authorization Serveren varetager i EHMI sikkerhedsarkitekturen ikke selv autentifikation af brugere men lader brugerautentifikation foregå via SEB IdP, som viderestiller til brugerens egen IdP.

Nogle brugere (borgere, private aktører) vil benytte NemLog-in IdP til autentifikation med MitID, andre brugere deres egen organisations lokale IdP (kommuner, regioner).

I nedenstående Figur 5 er der illustreret et autentifikationsflow hvor NemLog-in IdP benyttes. Her redirecter Authorization Server i (1) brugerens browser til SEB IdP, som i (2) redirecter browseren videre til NemLog-in IdP. Brugeren autentificerer sig med MitID i NemLog-in IdP hvorefter der returneres et SAML token til SEB IdP med brugerattributter, herunder NSIS autentifikationsniveau. SEB IdP returnerer efterfølgende et tilsvarende SAML token til Authorization Server med relevante brugerattributter. Hvis brugeren optræder som borger returneres det blandt andet brugerens CPR-nummer. Og for brugere som agerer som medarbejdere returneres EHMI-relaterede roller som medarbejderen måtte være blevet tildelt i SEB Brugerkataloget. Efter endt autentifikationsflow uddrager Authorization Server relevante attributter fra det SEB-udstedte SAML token og lader dem indgå som claims i det access token (og eventuel også identity token) som den selv udsteder til klienten (ikke vist i Figur 5).



Figur 5: Brugerautentifikation i klienter med brugerdelegering

6. Appendiks: Relevante uddrag af FAPI

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

6.1 FAPI afsnit *5.2.1 Requirements for all endpoints*

To protect against network attacks, clients, authorization servers, and resource servers

1. *shall only offer TLS protected endpoints and shall establish connections to other servers using TLS;*
2. *shall set up TLS connections using TLS version 1.2 or later;*
3. *shall follow the recommendations for Secure Use of Transport Layer Security in [BCP195];*
4. *should use DNSSEC to protect against DNS spoofing attacks that can lead to the issuance of rogue domain-validated TLS certificates; and*
5. *shall perform a TLS server certificate check, as per [RFC9525].*

***NOTE 1**: Even if an endpoint uses only organization validated (OV) or extended validation (EV) TLS certificates, an attacker using rogue domain-validated certificates is able to impersonate the endpoint and conduct man-in-the-middle attacks. CAA records [RFC8659] help to mitigate this risk.*

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

For server-to-server communication endpoints that are not used by web browsers, the following requirements apply:

1. *When using TLS 1.2, servers shall only permit the cipher suites recommended in [BCP195];*
2. *When using TLS 1.2, clients should only permit the cipher suites recommended in [BCP195].*

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

For endpoints that are used by web browsers, the following additional requirements apply:

1. *Servers 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, servers shall only use cipher suites allowed in [BCP195].*
3. *Servers shall not support CORS [CORS.Protocol] for the authorization endpoint, as clients must perform an HTTP redirect rather than access this endpoint directly.*

***NOTE 1**: When using TLS1.2 endpoints used by web browsers can use any cipher suite allowed in [BCP195], whereas endpoints not used by web browsers can only use cipher suites recommended by [BCP195].*

***NOTE 2**: New versions of [BCP195] will be published by the IETF periodically. At a minimum, implementors are expected to become compliant with newly issued versions of BCP195 within 12 months, or sooner.*

6.4 FAPI afsnit *5.3.2. Requirements for authorization servers*

5.3.2.1. General requirements

Authorization servers

1. *shall distribute discovery metadata (such as the authorization endpoint) via the metadata document as specified in [OIDD] and [RFC8414];*
2. *shall reject requests using the resource owner password credentials grant;*
3. *shall only support confidential clients as defined in [RFC6749];*
4. *shall only issue sender-constrained access tokens;*
5. *shall use one of the following methods for sender-constrained access tokens:*
    - *MTLS as described in [RFC8705],*
    - ~~*DPoP as described in [RFC9449];*~~
6. *shall authenticate clients using one of the following methods:*
    - *MTLS as specified in Section 2 of [RFC8705], or*
    - ~~*private_key_jwt as specified in Section 9 of [OIDC];*~~
7. *shall not expose open redirectors (see Section 4.11 of [I-D.ietf-oauth-security-topics]);*
8. *shall only accept its issuer identifier value (as defined in [RFC8414]) as a string in the aud claim received in client authentication assertions;*
9. *shall not use refresh token rotation except in extraordinary circumstances (see Note 1 below);*
10. ~~*if using DPoP, may use the server provided nonce mechanism (as defined in Section 8 of [RFC9449]);*~~
11. *shall issue authorization codes with a maximum lifetime of 60 seconds;*
12. ~~*if using DPoP, shall support "Authorization Code Binding to DPoP Key" (as required by Section 10.1 of [RFC9449]);*~~
13. *to accommodate clock offsets, shall accept JWTs with an iat or nbf timestamp between 0 and 10 seconds in the future but shall reject JWTs with an iat or nbf timestamp greater than 60 seconds in the future. See Note 3 for further details and rationale; and*
14. *should restrict the privileges associated with an access token to the minimum required for the particular application or use case.*

***NOTE 1**: The use of refresh token rotation does not provide security benefits when used with confidential clients and sender-constrained access tokens. This specification prohibits the use of refresh token rotation for security reasons as it causes user experience degradation and operational issues whenever the client fails to store or receive the new refresh token and has no option to retry.*

However, as refresh token rotation may be required from time to time for infrastructure migration or similar extraordinary circumstances, this specification allows it, provided that authorization servers offer clients the time-limited option to retry with the old refresh token in case of failure. Implementers need to consider a secure mechanism for clients to recover from a loss of a new refresh token on issue. The details of this mechanism are outside the scope of this specification.

***NOTE 2**: This document is structured to support a variety of grants to be used with the general requirements above. For example the client credentials grant or the IBA grant [CIBA]. Implementers should note that as of the time of writing only the authorization code flow and CIBA flows have been through a detailed security analysis [FAPI2SEC].*

***NOTE 3**: Clock skew is a cause of many interoperability issues. Even a few hundred milliseconds of clock skew can cause JWTs to be rejected for being "issued in the future". The DPoP specification [RFC9449] suggests that JWTs are accepted in the reasonably near future (on the order of seconds or minutes). This document goes further by requiring authorization servers to accept JWTs that have timestamps up to 10 seconds in the future. 10 seconds was chosen as a value that does not affect security while greatly increasing interoperability. Implementers are free to accept JWTs with a timestamp of up to 60 seconds in the future. Some ecosystems have found that the value of 30 seconds is needed to fully eliminate clock skew issues. To prevent implementations switching off iat and nbf checks completely this document imposes a maximum timestamp in the future of 60 seconds.*

5.3.2.2. Authorization endpoint flows

For flows that use the authorization endpoint, authorization servers

1. *shall require the value of response_type described in [RFC6749] to be code;*
2. *shall support client-authenticated pushed authorization requests according to [RFC9126];*
3. *shall reject authorization requests sent without [RFC9126];*
4. *shall reject pushed authorization requests without client authentication;*
5. *shall require PKCE [RFC7636] with S256 as the code challenge method;*
6. *shall require the redirect_uri parameter in pushed authorization requests;*
7. *shall return an iss parameter in the authorization response according to [RFC9207];*
8. *shall not transmit authorization responses over unencrypted network connections, and, to this end, shall not allow redirect URIs that use the "http" scheme except for native clients that use loopback interface Redirection as described in Section 7.3 of [RFC8252];*
9. *shall reject an authorization code (Section 1.3.1 of [RFC6749]) if it has been previously used;*
10. *shall not use the HTTP 307 status code when redirecting a request that contains user credentials to avoid forwarding the credentials to a third party accidentally (see Section 4.12 of [I-D.ietf-oauth-security-topics]);*
11. *should use the HTTP 303 status code when redirecting the user agent using status codes;*
12. *shall issue pushed authorization requests request_uri with expires_in values of less than 600 seconds;*
13. *should provide end-users with all necessary information to make an informed decision about whether to consent to the authorization request, including the identity of the client and the scope of the authorization; and*
14. *if supporting [OIDC], shall support nonce parameter values up to 64 characters in length, may reject nonce values longer than 64 characters.*

***NOTE 1**: If replay identification of the authorization code is not possible, it is desirable to set the validity period of the authorization code to one minute or a suitable short period of time. The validity period may act as a cache control indicator of when to clear the authorization code cache if one is used.*

***NOTE 2**: The request_uri expires_in time must be sufficient for the user's device to receive the link and the user to complete the process of opening the link. In many cases (poor network connection or where the user has to manually select the browser to be used) this can easily take over 30 seconds.*

***NOTE 3**: It is recommended that authorization servers that enforce one-time use of request_uri values ensure the enforcement takes place at the point of authorization, not at the point of loading an authorization page. This prevents user software that preloads urls from invalidating the request_uri.*

***NOTE 4**: In this document the state parameter is not used for CSRF protection, but may be used to by the client for application state. In circumstances where clients encode application state in a JWT the length of the state parameter value could be in excess of 1000 characters.*

***NOTE 5**: The use of OAuth 2.0 Rich Authorization Requests (RAR) [RFC9396] is recommended when the scope parameter is not expressive enough to convey the authorization that a client may want to obtain*

5.3.2.3. Returning authenticated user's identifier

If it is desired to provide the authenticated user's identifier to the client in the token response, the authorization server shall support OpenID Connect [OIDC].

6.5 FAPI afsnit *5.3.3.1 General requirements*

Clients

1. *shall support sender-constrained access tokens using one or both of the following methods:*
    - *MTLS as described in [RFC8705],*
    - ~~*DPoP as described in [RFC9449];*~~
2. *shall support client authentication using one or both of the following methods:*
    - *MTLS as specified in Section 2 of [RFC8705],*
    - ~~*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]* ~~*or Section 7.1 of DPoP [RFC9449]*~~*;*
4. *shall not expose open redirectors (see Section 4.11 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 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 [RFC8705];*
8. ~~*if using DPoP, shall support the server provided nonce mechanism (as defined in Section 8 of [RFC9449]);*~~
9. *shall only use authorization server metadata (such as the authorization endpoint) retrieved from the metadata document as specified in [OIDD] 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;*
12. *shall initiate an authorization process only with the end-user's explicit or implicit consent and protect initiation of an authorization process against cross-site request forgery, thereby enabling the end-user to be aware of the context in which a flow was started; and*
13. *should request authorization with the least privileges necessary for the specific application or use case.*

~~***NOTE 1**: This profile may be used by confidential clients on a user-controlled device where the system clock may not be accurate, causing private_key_jwt client authentication to fail. In such circumstances a client should consider using the HTTP date header returned from the server to synchronize its own clock when generating client assertions.*~~

~~***NOTE 2**: Although authorization servers are required to support "Authorization Code Binding to DPoP Key" (as defined by Section 10.1 of [RFC9449]), clients are not required to use it.*~~

6.6 FAPI afsnit 5.3.3.2 Authorization code flow

For the Authorization Code flow, Clients

1. *shall use the authorization code grant described in [RFC6749];*
2. *shall use pushed authorization requests according to [RFC9126];*
3. *shall use PKCE [RFC7636] with S256 as the code challenge method;*
4. *shall generate the PKCE challenge specifically for each authorization request and securely bind the challenge to the client and the user agent in which the flow was started;*
5. *shall check the iss parameter in the authorization response according to [RFC9207] to prevent mix-up attacks;*
6. *shall only send client_id and request_uri request parameters to the authorization endpoint (all other authorization request parameters are sent in the pushed authorization request according to [RFC9126]);*
7. *if using [OIDC], should not use nonce parameter values longer than 64 characters.*

***NOTE 1**: The recommended restrictions on the nonce parameter value length is to aid interoperability.*

6.7 FAPI afsnit *5.3.4. Requirements for resource servers*

The FAPI 2.0 endpoints are OAuth 2.0 protected resource endpoints that return protected information for the resource owner associated with the submitted access token.

Resource servers with the FAPI endpoints

1. *shall accept access tokens in the HTTP header as in Section 2.1 of OAuth 2.0 Bearer Token Usage [RFC6750]* ~~*or Section 7.1 of DPoP [RFC9449]*~~*;*
2. *shall not accept access tokens in the query parameters stated in Section 2.3 of OAuth 2.0 Bearer Token Usage [RFC6750];*
3. *shall verify the validity, integrity, expiration and revocation status of access tokens;*
4. *shall verify that the authorization represented by the access token is sufficient for the requested resource access and otherwise return errors as in Section 3.1 of [RFC6750]; and*
5. *shall support and verify sender-constrained access tokens using one or both of the following methods:*
    - *MTLS as described in [RFC8705],*
    - ~~*DPoP as described in [RFC9449].*~~

6.8 FAPI afsnit *5.4. Cryptography and secrets*

The following requirements apply to cryptographic operations and secrets:

1. *Authorization servers, clients, and resource servers when creating or processing JWTs shall*
    1. *adhere to [RFC8725];*
    2. *use PS256, ES256, or EdDSA (using the Ed25519 variant) algorithms; and*
    3. *not use or accept the none algorithm.*

2. *RSA keys shall have a minimum length of 2048 bits.*
3. *Elliptic curve keys shall have a minimum length of 224 bits.*
4. *Credentials not intended for handling by end-users (e.g., access tokens, refresh tokens, authorization codes, etc.) shall be created with at least 128 bits of entropy such that an attacker correctly guessing the value is computationally infeasible. Cf. Section 10.10 of [RFC6749].*

***Note**: As of the time of writing there isn't a registered fully-specified algorithm describing "EdDSA using the Ed25519 variant". If such algorithm is registered in the future, it is also allowed to be used for this profile.*

7. Appendiks: Anvendelse af sikkerhedsmodellen i EHMI services

I det følgende beskrives den konkrete anvendelse af sikkerhedsmodellen i de tre EHMI services, herunder parametre til whitelisting af klienter, anvendte integrations-flows og brug af OAuth scopes.

> Indholdet af dette appendiks indarbejdes på et senere tidspunkt i de enkelte EHMI services dokumentation og udgår af dette dokument.

7.1 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 meddelelses-håndteringer i forsendelsesstatusservicen 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.

Forsendelsesstatus indeholder personfølsomme oplysninger (i form at behandlingsstedet som indgår som afsender eller modtager af en meddelelse), og brugeradgange forudsætter derfor et NSIS-niveau 'Betydelig'.

7.1.1 EDS usecases

Der er to overordnende usecases for anvendelsen af forsendelsesstatusservicen EDS.

1. Stationerne i en EHMI forsendelse foretager hver især en *registrering af forsendelsesstatus* i EDS. Registrering sker på systemniveau, og de enkelte stationer kan oprette forsendelsesstatus for de organisationskontekster (kombinationer af GLN numre og SOR koder) som de er whitelistet til (se nedenstående).

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

   Søgning og opslag kan enten foregå:

   a. På systemniveau for stationerne på deres eget device_id. (Derved kan der eksempelvis fra fagsystemer etableres funktionalitet hvor brugere kan fremsøge status for meddelelser der er blevet sendt fra deres organisation.)
   b. På borgerniveau på eget CPR

c. På superbruger/leverandør(supporter)-niveau på CVR nummer for brugerens organisation og hvor brugeren får adgang via en særlig rettighed (som tildeles gennem SEB brugerkataloget)

7.1.2 Indrullering/whitelisting af systemklienter i EDS (til registrering, søgning og opslag)

Stationer som registrerer forsendelsesstatus og kan søge og læse egne registreringer indrulleres som systemklienter.

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

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

Under indrullering angives følgende som scope element:

```
EDS system/AuditEvent.crs
```

(Ovenstående `system/AuditEvent.crs` 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 klienter beskrevne metadata elementer skal følgende metadata elementer angives for systemklienter.

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

Bemærk, at det er hensigten, at Authorization Server efter produktionspiloten i stedet laver opslag på en stations organisationskontekster i EER postkasseregisteret og den eksplicit whitelisting dermed bortfalder.

Eksempel metadata dokument for en EDS systemklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": [
    "client_credentials"
  ],
  "client_name": "Lægesystem XYZ - Frederiksbjerg Lægehus",
  "scope": "EDS system/AuditEvent.crs",
  "contacts": [
    "døgnsupport@lægesystem-xyz.dk",
    "+45 1234 5678"
  ],
  "tls_client_auth_subject_dn": "subject=CN=Lægesystem XYZ's
systemcertifikat, serialNumber=UI:DK-O:G:a262681f-2e94-45c5-aaea-
aad4e9bc5768, O=Leverandør af Lægesystem XYZ,
organizationIdentifier=NTRDK-12345678, C=DK",
  "ehmi:eer:device_id": "c4b8d3ea-b187-426b-be77-bffd9f593d84",
  "ehmi:org_context": [
    {
      "name": "Frederiksbjerg Lægehus",
      "sor": "1216891000016007",
      "gln": "5790000135912"
    }
  ]
}
```

7.1.3 Indrullering/whitelisting af brugerklienter (til søgning og opslag)

Brugerklienter som anvendes af borgere eller superbrugere/supportere til at søge og læse forsendelsesstatus-registreringer indrulleres alene med de i afsnit 3.3 Indrullering af klienter beskrevne elementer.

Under indrullering angives følgende scope element:

```
EDS user/AuditEvent.rs
```

Metadata for en EDS brugerklient til søgning og opslag

For EDS brugerklienter skal der kun angives de i afsnit 3.3.1 Metadata for klienter beskrevne metadata.

Eksempel metadata dokument for en EDS brugerklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "client_name": "Lægesystem XYZ - Frederiksbjerg Lægehus",
  "scope": "EDS user/AuditEvent.rs",
  "contacts": [
    "døgnsupport@lægesystem-xyz.dk",
    "+45 1234 5678"
  ],
  "tls_client_auth_subject_dn": "subject=CN=Lægesystem XYZ's
systemcertifikat, serialNumber=UI:DK-O:G:a262681f-2e94-45c5-aaea-
aad4e9bc5768, O=Leverandør af Lægesystem XYZ,
organizationIdentifier=NTRDK-12345678, C=DK",
  "redirect_uris": [
    "https://www.lægesystem-xyz.dk/lps-system/frederiksbjerg-lægehus"
  ]
}
```

7.1.4 Kald til Token Endpoint

I tilgangen til EDS opereres der for *registreringer* med organisations-specifikke tokens, det vil sige at de enkelte stationers systemklienter 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 at kunne tilgå EDS angives følgende scopes:

scope	Beskrivelse
`EDS`	En angivelse af det er for EDS, at klienten ønsker et access token.
`system/AuditEvent.crs`	(kun for systemklienter) En angivelse af at tokenet skal kunne registrere/læse/fremsøge forsendelsesstatus ressourcer (som er profileringer af FHIR's AuditEvent ressource).
`user/AuditEvent.rs`	(kun for brugerklienter) En angivelse af at tokenet skal kunne læse og fremsøge forsendelsesstatus ressourcer (som er profileringer af FHIR's AuditEvent ressource).
`SOR:<XXXXX>`	(kun for systemklienter og kun ved registreringer) En angivelse af organisationens SOR kode, hvor `<XXXXX>` sættes til selve koden.

scope	Beskrivelse
`GLN:<YYYYY>`	(kun for systemklienter og kun ved registreringer) En angivelse af organisationens GLN lokationsnummer, hvor `<YYYYY>` sættes til selve lokationsnummeret.

Eksempel på en samlet scope som indgår i kaldet for en systemklient:

```
EDS system/AuditEvent.crs 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.sundhedsdatastyrelsen.dk
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 156

grant_type=client_credentials&scope=EDS%20system%2FAuditEvent.crs%20SOR%3
A306861000016006%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.

For systemklienter, der anmoder om et token til at foretage *registreringer*, mapper Authorization Serveren 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.

For access tokens udstedt til systemklienter, der laver *registreringer* i EDS indlejrer Authorization Server device_id og den organisatoriske kontekst som yderligere claims i tokenet på følgende form:

Claim	Beskrivelse
`ehmi:eer:device_id`	En angivelse af device_id som stationen er registreret med i EER.
`ehmi:org_context`	Den aktuelle organisationskontekst for stationen, angivet som JSON objekt bestående af `name` (organisationsnavn), `sor` (SOR kode) og `gln` (lokationsnummer).

7.1.5 Kald til EDS

Kald til EDS foretages som beskrevet i den generelle sikkerhedsmodel som REST-kald over tovejs TLS, med access token (som er sender-constrained) i en HTTP header.

Eksempel på en systemklients 'create' kald til EDS med AuditEvent[10] ressourcen angivet som JSON objekt:

```
POST /base/AuditEvent HTTP/1.1
Host: ehmi.medcom.dk
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 både er gyldigt og udstedt til EDS og validerer 'sender-constrained' egenskaben, det vil sige validerer, at det af klientens anvendte TLS-klientcertifikat matcher certifikatet, som blev indlejret i access tokenet.

Ved *registrering* af en forsendelsesstatus tjekker servicen desuden, at tokenet indeholder de nødvendige scopes til at klienten må foretage registreringer i EDS. 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.

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

Ved *søgning og læsning* afgrænser EDS servicen tilgangen:

1. *For systemklienter som repræsenter en station i et meddelelses-flow*: Til registreringer foretaget af stationens eget device_id.

   EDS begrænser således adgangen til forsendelsesstatusregistreringer hvori der indgår samme device_id, som fremgår af access tokenets `ehmi:eer:device_id` claim.

2. *For borgere som tilgår EDS via en brugerklient*: Til registreringer foretaget på borgerens eget CPR nummer.

   EDS begrænser således adgangen til forsendelsesstatusregistreringer hvori der indgår samme patient CPR nummer, som fremgår af access tokenets `cpr` claim.

3. *For superbrugere/supportere som tilgår EDS via en brugerklient*: Til registreringer vedrørende egen organisation på CVR niveau.

   EDS validerer først, at den adgangsgivende rolle[11] som tildeles superbrugere/supporter fremgår af access tokenets rolle-struktur i `priv` claim'et. EDS begrænser derefter adgangen til forsendelsesstatusregistreringer, hvori der indgår samme organisations CVR-nummer, som fremgår af access tokenets `cvr` claim.

7.2 EHMI Addressing Service (EAS)

Igennem Sundhedsadresseringsservicen (EAS) kan afsendersystemer fremsøge potentielle meddelelsesmodtagere ud fra forskellige søgekriterier.

I kommunikationen med EAS kan der indgå *personhenførbare* oplysninger i form af borgernes egen læge (som kan slås op på baggrund af en borgers CPR nummer), men der indgår ikke *personfølsomme* data. Det er derfor blevet vurderet, at adgang til EAS kan foregå på systemniveau og ikke kræver, at den autentificerede identitet af den menneskelige bruger bag kaldet overføres til EAS.

7.2.1 EAS usecases

EAS er ikke selv databærende, men aggregerer adresseoplysninger ud fra en række forskellige autoritative kilder (herunder postkasseregisteret EER). EAS udstiller således alene en snitflade til at kunne foretage søgninger og opslag.

7.2.2 Indrullering/whitelisting af systemklienter i EAS (til søgning og opslag)

Afsendersystemer indrulleres som systemklienter med de i afsnit 3.3 Indrullering af klienter beskrevne elementer, hvor der angives følgende som scope element:

```
EAS system/Organization.rs
```

[11] Den konkrete navngivning fastlægges senere.

Metadata for en EAS systemklient
Der skal ikke angives yderligere metadata end de i afsnit 3.3.1 Metadata for klienter beskrevne.

Eksempel metadata dokument for en EAS systemklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": [
    "client_credentials"
  ],
  "client_name": "Aarhus Kommunes EOJ",
  "scope": "EAS system/Organization.rs",
  "contacts": [
    "eoj@aarhus-kommune.dk",
  ],
  "tls_client_auth_subject_dn": "subject=CN=EOJ leverandør XYZ's
systemcertifikat, serialNumber=UI:DK-O:G: d6eef4ae-5c37-4206-be4c-
5fac2cbca29d, O=EOJ leverandør XYZ, organizationIdentifier=NTRDK-
56781234, C=DK"
}
```

7.2.3 Kald til Token Endpoint

For at få udstedt et access token til at kunne tilgå EAS angives følgende scopes:

scope	Beskrivelse
`EAS`	En angivelse af det er for EAS, at klienten ønsker et access token.
`system/Organization.rs`	En angivelse af at tokenet skal kunne læse/fremsøge sundhedsadresserings ressourcer (som er profileringer af FHIR's Organization ressource).

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.

7.2.4 Kald til EAS

Kald til EAS foretages som beskrevet i den generelle sikkerhedsmodel som REST-kald over tovejs TLS, med access tokenet (som er sender-constrained) i en HTTP header.

Eksempel på en systemklients kald af EAS operationen som returnerer 'egen læge' for en patient (som har sygesikringsgruppe 1):

```
POST /base/$getSikrGrp1_getReceivingOrganizationByPatientId HTTP/1.1
Host: eas.sundhedsdatastyrlsen.dk
Accept: application/fhir+json
Content-Type: application/fhir+json
Content-Length: 9191
Authorization: Bearer eyJhb ... Dhi6g

{
  "parameter": [
    {
      "name": "target",
      "resource": {
        "resourceType": "Patient",
        // Patient resource
      }
    }
  ]
}
```

EAS adgangskontrol
Sundhedsadresseringsservicen tjekker, at access tokenet er gyldigt og validerer 'sender-constrained' egenskaben, det vil sige validerer, at det af klientens anvendte TLS-klientcertifikat matcher certifikatet som blev indlejret i access tokenet.

Servicen validerer desuden, at tokenet er udstedt til EAS som aftager af tokenet og indeholder de nødvendige scopes til at klienten må foretage opslag i EAS.

7.3 EHMI Endpoint Register (EER)

I Postkasseregisteret (EER) administrerer parterne i EHMI meddelelsesinfrastrukturen Endpoint-adresser for deres forskellige organisatoriske enheder, som skal kunne modtage meddelelser.

Sundhedsadresseringsservicen EAS forventes at være eneste aftager af EER-data, som er en af de autoritative kilder EAS benytter til at udstille et samlet søgeinterface til anvenderne af EHMI.

7.3.1 EER usecases

Der er to overordnende usecases for anvendelsen af postkasseregisteret EER.

1. EER udstiller en snitflade til *søgning og opslag* i organisationers endpoints. I produktionspiloten (og givetvis også i det lange løb) er der udelukkende Sundhedsadresseringsservicen, der benytter denne snitflade.

2. EER stiller desuden en grænseflade til rådighed, hvor organisationers endpoints kan opsættes og administreres. Opsætning og administration foregår på superbruger-niveau for brugerens organisation. Superbrugere kan få adgang via en særlig administrator-rettighed (som tildeles gennem SEB brugerkataloget).

7.3.2 Indrullering/whitelisting af systemklienter i EER (til søgning og opslag)

Sundhedsadresseringsservicen, som indtil videre er eneste klient, der foretager søgninger og opslag i EER, indrulleres som systemklient med de i afsnit 3.3 Indrullering af klienter beskrevne elementer, hvor der angives følgende som scope element:

```
EER system/Endpoint.rs system/Organization.rs
```

Metadata for en EER systemklient

Der skal ikke angives yderligere metadata end de i afsnit 3.3.1 Metadata for klienter beskrevne.

Eksempel metadata dokument for en EER systemklient (dvs. Sundhedsadresseringsservicen):

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": [
    "client_credentials"
  ],
  "client_name": "Sundhedsadresseringsservice (EAS)",
  "scope": " EER system/Endpoint.rs system/Organization.rs",
  "contacts": [
    "eas-support@ehmi.dk"
  ],
  "tls_client_auth_subject_dn": "subject=CN=Systemleverandør ABC's
systemcertifikat, serialNumber=UI:DK-O:G:7000b95d-b9bc-415d-88fe-
5561859e7399, O= Systemleverandør ABC, organizationIdentifier=NTRDK-
34567812, C=DK"
}
```

7.3.3 Indrullering/whitelisting af brugerklienter (til administration)

Brugerklienter som anvendes af superbrugere til at administrere indgange i Postkasseregisteret indrulleres alene med de i afsnit 3.3 Indrullering af klienter beskrevne elementer.

Under indrullering angives følgende scope element:

```
EER user/Endpoint.cruds user/Organization.cruds
```

Metadata for en EER brugerklient til administration af Postkasseregisterindgange

For EER brugerklienter skal der kun angives de i afsnit 3.3.1 Metadata for klienter beskrevne metadata.

Eksempel metadata dokument for en EER brugerklient:

```
{
  "token_endpoint_auth_method": "tls_client_auth",
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "client_name": "Postkasseregister web-admin",
  "scope": "EER user/Endpoint.cruds user/Organization.cruds",
  "contacts": [
    "eer-support@ehmi.dk"
  ],
  "tls_client_auth_subject_dn": " subject=CN=Systemleverandør XYZ's
systemcertifikat, serialNumber=UI:DK-O:G: c91eada9-90a7-4187-94a3-
f880df10348a, O= Systemleverandør XYZ, organizationIdentifier=NTRDK-
67812345, C=DK",
  "redirect_uris": [
    "https://eer.ehmi.dk/web-admin"
  ]
}
```

7.3.4 Kald til Token Endpoint

For at få udstedt et access token til at kunne tilgå EER angives følgende scopes:

scope	Beskrivelse
EER	En angivelse af det er for EER, at klienten ønsker et access token.
system/Endpoint.rs	(kun for systemklienter) En angivelse af at tokenet skal kunne benyttes til at læse og fremsøge Postkasseregister ressourcer (som er FHIR bundles bestående af profileringer af FHIR's Endpoint og Organization ressourcer).
system/Organization.rs	
user/Endpoint.cruds	(kun for brugerklienter) En angivelse af at tokenet skal kunne benyttes til at oprette/læse/opdatere/slette/fremsøge Postkasseregister ressourcer (som er FHIR bundles bestående af profileringer af FHIR's Endpoint og Organization ressourcer).
user/Organization.cruds	

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.

7.3.5 Kald til EER

Kald til EER foretages som beskrevet i den generelle sikkerhedsmodel som REST-kald over tovejs TLS, med access tokenet (som er sender-constrained) i en HTTP header.

Eksempel på en systemklients …. <TODO>

```
POST /base/XXXXXX HTTP/1.1
Host: eer.ehmi.dk
Accept: application/fhir+json
Content-Type: application/fhir+json
Content-Length: 6112
Authorization: Bearer eyJhb ... Dhi6g

{
  "parameter": [
    {
      "name": "YYYY",
      "resource": {
        "resourceType": "ZZZZ",
        // ZZZZ resource
      }
    }
  ]
}
```

EER adgangskontrol

Postkasseregisteret tjekker at access tokenet er gyldigt og validerer 'sender-constrained' egenskaben, det vil sige validerer, at det af klientens anvendte TLS-klientcertifikat matcher certifikatet, som blev indlejret i access tokenet. Postkasseregisteret verificerer desuden, at tokenet er udstedt til EER som aftager af tokenet.

Ved *søgning og opslag* af postkasseregisterindgange tjekker servicen, at tokenet indeholder de nødvendige scopes til at klienten må foretage søgninger op opslag i EER.

Ved *administration* af postkasseregisterindgange afgrænser EER tilgangen til registreringer vedrørende egen organisation på CVR niveau. Konkret validerer EER først at brugeren har fået tildelt superbruger-rollen, dvs. at den adgangsgivende rolle[12] fremgår af access tokenets rolle-struktur i `priv` claim'et. EER begrænser derefter adgangen til postkasseindgange, hvori der indgår samme organisations CVR-nummer, som fremgår af access tokenets `cvr` claim.

[12] Den konkrete navngivning fastlægges senere.

8. Referencer

- [BCP195] ”BCP 195”, https://www.rfc-editor.org/info/bcp195
- [CORS.Protocol] ”CORS Protocol”, https://fetch.spec.whatwg.org/#http-cors-protocol
- [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-security-profile-2_0-04.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
- [NSIS] “National Standard for Identiteters Sikringsniveauer”, https://digst.dk/it-loesninger/standarder/nsis/
- [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
- [OAuth-TKEX] ”OAuth 2.0 Token Exchange”, https://www.rfc-editor.org/rfc/rfc8693
- [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/
- [RFC6797] “HTTP Strict Transport Security (HSTS)”, https://datatracker.ietf.org/doc/html/rfc6797

- [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
- [RFC8725] “JSON Web Token Best Current Practices”, https://datatracker.ietf.org/doc/html/rfc8725
- [RFC9207] “OAuth 2.0 Authorization Server Issuer Identification”, https://datatracker.ietf.org/doc/html/rfc9207
- [RFC9525] “Service Identity in TLS”, https://datatracker.ietf.org/doc/html/rfc9525
- [PSD 2] ”PSD 2 Direktiv”, https://www.finanstilsynet.dk/Lovgivning/Ny_EU_lovsamling/PSD-2
- [SMART] ”SMART App Launch 2.2.0”, 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