DIGITAL SUNDHED

SAMMENHÆNGENDE DIGITAL SUNDHED I DANMARK

SOSIGW

- Arkitektur og design for SOSIGW 1.0

Indeks

Version	**Dato**	**Ansvarlig**	**Kommentarer**
2	15-10-2009	jre	Formattering

Revisionshistorik

Version	Dato	Ændring	Ansvarlig
1	09/10/09	Initiel version af dokument	jre
2	15/10/09	Formattering	jre
3	06/12/10	Bootstrap snitflade	bbg

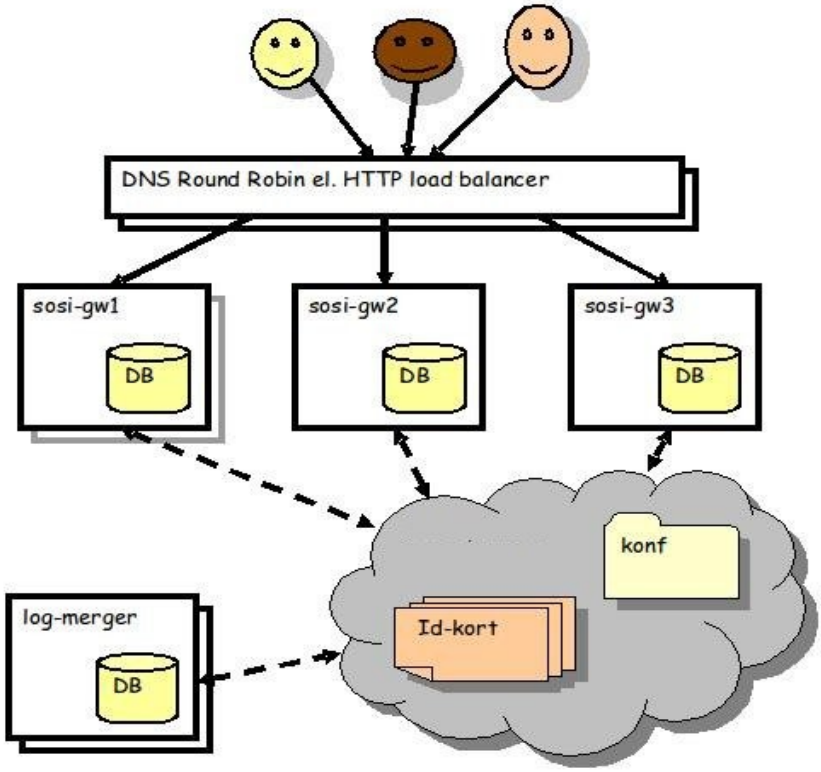
Arkitektur af SOSI-GW

Den grundlæggende arkitektur er baseret på et cluster af servlet containere. Der er tale om en symmetrisk løsning, hvor alle servere i clusteret er identiske i opsætning og konfiguration. Clustering funktionaliteten opnås på applikationsniveau, der er derfor ikke krav om clustering support i den benyttede servlet container. Den nedre grænse for cluster-størrelse vil være en enkelt server, den øvre grænse i princippet ubegrænset. Der understøttes rullende opdatering, load balancering og fail-over ved en clusterstørrelse på 2 eller flere.

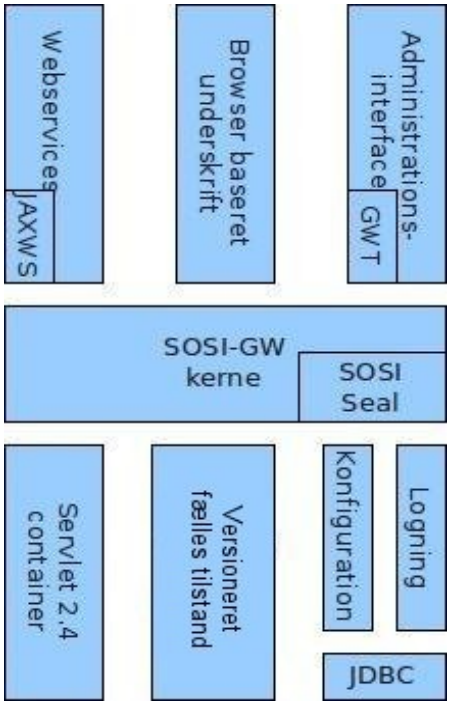
Hver server vil være bestykket med en servlet container (fx Tomcat) og en lokal PostgreSQL DBMS. Den lokale database anvendes udelukkende til at opsamle logentries samt lokal konfiguration. En knude i nettet kræver at begge komponenter (servlet container og DBMS) er kørende for at være funktionelle.

Til opsamling og søgning på audit logs opstilles en log-merger server, som ligeledes er bestykket med en lokal PostgreSQL DBMS.

Figuren nedenfor skitserer SOSI-GW løsningen bestående af et antal gateway servere (sosi-gw1, .., sosi-gw3) og en central database til opsamling og fletning af audit logs (log-merger).



Intern arkitektur



Figuren ovenfor viser den interne arkitektur i SOSIGW. Løsningen består af en kerne, der anvender SOSISeal til at håndtere idkort. Idkort distribueres over en fælles cache, der drives af multicasting på lokalnettet. Konfiguration og logning gemmes i en lokal JDBCdatabase. Ovenpå kernen ligger tre snitflader, nemlig webservices (JAX-WS), http til browserbaseret signering samt GWT (Google Web Toolkit) til administration.

Tilstandsdeling imellem clustermedlemmer

De servere, der indgår i et cluster kommunikerer indbyrdes via et distribueret, delt hukommelseområde, realiseret via UDP multicast på lokalnettet. Herigennem overføres løbende opdateringer i form af id-kort oprettelse, opdatering og nedlæggelse.

Udover id-kort, bruges clusteret også til at synkronisere konfigurationsændringer, der er foretaget igennem administrationskonsollen.

Multicast benyttes endvidere som discovery mekanisme, dvs. som måden hvorpå clustermedlemmer fastlægger hvilke øvrige medlemmer clusteret indeholder på et givet tidspunkt. Det betyder, at der ikke vil være behov for nogen manuel konfiguration i forbindelse med at en server tilføjes eller fjernes.

Versionsstyring af delte data

Som led i understøttelsen af rullende opgraderinger, påhæftes alle delte datatyper, dvs id-kort og dynamisk konfiguration, et versionsnummer. Software-opgraderinger, der involverer modifikationer af de delte datatyper, må understøtte både det nye og det gamle format, og fortsætte med at udgive data i det forrige format indtil det detekteres at der ikke længere er cluster-medlemmer, der kræver dette.

Der understøttes kun det aktuelle og det forrige format, ved en forskel på mere end et versionsnummer af beskedsformatet, kan der ikke kommunikeres. Ved opstart checker servere om beskeder afsendt af andre servere er i et af de to understøttede formater, og afbryder ellers opstarten.

Konfiguration

Den statiske, filbaserede konfiguration af SOSI-GW applikationen holdes på et minimum for at undgå omstændig kopiering af konfiguration, og for lettere at kunne honorere kravet om rullende opdateringer mm. Al anden konfiguration lægges i den lokale database på de enkelte servere og udveksles via multicast.

Den dynamiske konfiguration persisteres i den lokale database på alle servere i SOSI-GW clusteret. Konfigurationsdata gemmes altid som et komplet hele, der er forsynet med et tidsstempel, der identificerer og sammenbinder sammenhørende konfigurationsdata. Det er ikke påtænkt, at administrationskonsollen skal betjene mere end én samtidig bruger, men der laves ikke nogen explicit sikring af, at dette overholdes. Derimod håndteres evt. konflikter ved at lade den yngste konfiguration vinde. Den statiske konfiguration replikeres ikke automatisk. I stedet bør konfigurationsfilen synkroniseres via andre mekanismer, fx rsync.

En vilkårlig server i clusteret kan servicere administrationskonsollen. Når administratoren markerer at konfigurationsændringer skal gemmes, distribueres de nye konfigurationsdata, og alle servere persisterer konfigurationen i den lokale database. Når en server startes, sammenlignes tidsstemplet på de persisterede konfigurationsdata med den version, der kan aflæses via netværket, og den nyeste af de to benyttes.

En konsekvens af den anvendte strategi vil være, at det er ukompliceret at pakke løsningen som fx et VMware image, hvilket letter opsætningen af yderligere SOSI-GW servere.

Administrationskonsol

Administration af et SOSI-GW cluster foretages via en webbaseret administrationskonsol. Grænsefladen er baseret på GWT (Google Web Toolkit). Administrationskonsollen indeholder flg. funktioner:

- Bootstrap mode, hvor gateway funktionalitet er slået fra, men hvor adgangsbegrænsning er baseret på filbaseret brugernavn/password. Dette mode benyttes kun under installationsprocessen.
- Opsætning af white lists, hvor tilladte klienter sættes op og identificeres.
- Opsætning af service positivliste, hvor services som må kaldes på vegne af klienter sættes op og identificeres.
- Id-kort revokering, hvilket giver mulighed for at fremsøge et id-kort på baggrund af et bruger id, og efterfølgende revokere dette, dvs. fjerne id-kortet fra den delte cache.
- Søgning i audit logs. Administrationskonsollen giver mulighed for at foretage simple søgninger i auditloggen, fx baseret på bruger-id og/eller tidsrum.

Logning

Auditlogning foretages lokalt på de enkelte servere i clusteret. Persistering af log elementer foretages i en asynkron afkoblingstråd, således at gateway funktionaliteten kan afvikles optimalt. Audit logs persisteres lokalt i de decentrale databaser, der er installeret på alle SOSI-GW servere.

Med et konfigurerbart interval sender hver enkelt server opsamlede logs til den centrale log-merger database server. Dette foregår via jdbc adgang til log-mergerens database. Når data er committet på log-merger databasen, slettes disse lokalt. Dette betyder, at selvom log-mergeren ikke er dupleret, mistes der ikke auditlogs hvis log-mergeren er ude af drift. I det øjeblik log-mergeren er reetableret, vil akkumulerede auditlogs automatisk blive overført. De data som er overført til log-mergeren bør sikres med passende disk-spejlings- og backup-strategier.

Søgning i auditlogs foregår på log-mergerens database via administrationskonsollen eller som alternativ via et generisk SQL værktøj (PgAdmin, DbVisualizer el. lign.) direkte på databasen.

I en minimal ikke-redundant konfiguration kan log-merger og SOSI-GW server køre på den samme host, (evt. under samme DBMS).

Brugen af en central log mens der logges direkte til lokale logs gør, at den centrale log kan undværes i kortere eller længere perioder. Dermed er der ikke nogen faste afhængigheder mellem serverne, og SOSI-GW går ikke ned hvis den centrale log forsvinder. Konsekvensen er, at den centrale log ikke viser realtidsoplysninger, da der kan gå et stykke tid fra en hændelse sker til den optræder i den centrale log. Det betyder også, at skulle en af serverne stoppe helt med at fungere, så kan logbeskeder gå tabt. Det anbefales dog, at der også på de enkelte servere køres med spejlede diske, og derfor skal det gå meget galt før noget går helt tabt.

Ud over den beskrevne logning, som fungerer som audit-logning, bliver der også internt foretaget logning via log4j. Denne logning sker kun lokalt på de enkelte maskiner, og er beregnet til at overvåge og debugge systemet.

Til auditloggen logges følgende:

- Proxyrequests: Tidspunkt, NameID, Systemid, ip for afsender, endpoint, IDCardID (kortets) unikke id
- Service requests: Tidspunkt, NameID, Systemid, ip for afsender, operation, status (OK/ERR)
- URL requests: Tidspunkt, NameID, Systemid, ip for afsender, status (OK/ERR)

Desuden logges alle ugyldige requests, både Webservicerequests og requests der stammer fra den browserbaserede signering.

Samtidighed

Metoderne til at oprette, signere og fjerne id-kort implementerer ikke nogen form for låsning. Det betyder, at hvis to klienter på samme tid bestiller et nyt id-kort for den samme bruger, så vil kun en af dem kunne signere det endelige id-kort, nemlig den klient, der bestilte sidst. Det antages at der ikke foretages samtidige kald fra den samme bruger, og at problemet derfor ikke er relevant.

Load balancing

For at distribuere load til de servere, der indgår i et SOSI-GW cluster, er der behov for at opsætte en loadbalancing mekanisme. Dette er ikke en del af selve SOSI-GW, men det kunne fx bestå i en DNS round-robin løsning, en tcp load balancer (fx Linux Virtual Server) eller en http load balancer. Der er ikke behov for understøttelse af session affinity eller lign., da alle servere har adgang til alle cachede id-kort via det distribuerede netværk. Dette gælder for alle funktioner, også administrationskonsol og signeringsapplet.

Der vil være behov for at enten den anvendte load balancer eller (i tilfældet DNS round-robin) klienterne forsøger igen i tilfælde af at en server ikke kan nås.

I tilfælde, hvor det ønskes at kommunikation imellem klient og SOSI-GW foregår over SSL (fx den genererede URL til signeringsapplet), skal der enten opsættes en SSL terminering foran SOSI-GW serverne eller også skal det være muligt for de individuelle SOSI-GW servere at kunne uddele en URL, der rammer en specifik server.

Afvikling på en enkelt maskine

Det er muligt at afvikle SOSI-GW på en enkelt maskine uændret, da systemet selv vil detektere at der kun er et medlem af clusteret. Det vil også være muligt at fravælge distribuerings-funktionaliteten via den statiske konfiguration, da det i dette tilfælde vil påføre et unødvendigt (om end minimalt) overhead. Det vil fortsat være muligt at køre log-mergerens database på samme eller en anden maskine.

Anvendte teknologier og komponenter

I SOSI-GW er følgende teknologier og komponenter anvendt:

- SOSI Seal til udstedelse af ID-kort
- PostgreSQL til opbevaring af konfiguration og auditlogs
- JAX-WS-RI til implementering af web services
- Google Web Toolkit til implementering af administrationskonsol
- OpenSign applet til signering af ID-kort via browser
- Apache JMeter til performancetest

Webservice API

SOSIGW's primære funktionalitet stilles til rådighed gennem Webservices. Funktionaliteten er delt op i to dele: Services til at håndtere idkort (oprette, signere og nedlægge) og en proxyservice.
Proxyservicen er et generisk endpoint, der står for at videresende requests fra en klient til et andet endpoint mens der også tilføjes et signeret idkort. Når proxyservicen anvendes, foretages der ikke nogen validering af SOAPbody, og ligeledes checkes headeren kun for felter, der er nødvendige for at SOSIGW kan fungere.
Alle services bygger på den allerede definerede form for idkort fra DGWS, hvilket betyder, at data sendes som SAMLassertions. Ved samtlige kald til SOSIGW, checkes det om kalderen, identificeret ved en ipadresse og en shared secret, har adgang til at benytte SOSIGW. Der benyttes altså kun standarder, der allerede er i anvendelse (WSAddressing, SAML m.v.).

Oversigt over metoder tilgængelige som webservices.

- proxy
- requestIdCardDigestForSigning
- signIdCard
- getValidIdCard
- logout

Proxyservice

Proxy tager et generisk SOAPrequest og tilføjer IDkort til headerne. For at gøre dette, skal klienten medsende sessionsidentifikation i form af et brugerid. Brugerid'et transporteres i en SAMLassertion i stil med DGWS' idkort, hvor det eneste benyttede felt er <NameID>. Når requestet er modtaget, og brugerens idkort er vedhæftet, sendes requestet direkte videre til det endelige endpoint. SOSIGW understøtter opgradering af niveau 1 til 4. Niveau 1 er defineret i DGWS, og betyder at der kommer et usigneret idkort med brugernavn. Til requests hvor der allerede eksisterer et signeret idkort, ignoreres alt andet indhold end <NameID>feltet, og kortet erstattes med det, der opbevares af serveren. I de tilfælde, hvor der ikke eksisterer et kort i forvejen fejler operationen.

Servicen tager ikke argumenter direkte, da SOAPbody indeholder det, der skal sendes videre til den endelige aftager. Til gengæld kræves det, at requests er udstyret med en header, der indeholder følgende:

- Unikt brugerid i form af en SAMLassertion med <saml:NameID>
- WSAddressingheaders i henhold til http://www.w3.org/Submission/wsaddressing/
- Et optionelt idkort. Dette fjernes inden dispatch og erstattes evt. af det cachede kort.

Returværdier:

- “200” (OK) på HTTP niveau, og som indhold det uændrede svar fra den ønskede service.
- “500” (Error) på HTTP niveau, hvilket kan skyldes:
- Den ønskede service returnerede denne fejl.
    - Den ønskede service er ikke konfigureret på listen over kendte endpoints.
    - Manglende eller ugyldige argumenter i SOAPheaderne, f.eks. manglende WSaddressing.
    - Der findes ikke et gyldigt signeret Idkort i SOSIGW.

I fejltilfældene sendes en beskrivelse af fejlen i <fault> elementet, dels som kodede værdier, og dels som læselig tekst i henhold til DGWS.

Hvordan NameID tilføjes til SOAPbeskeden er op til klienten selv, men det vil formentligt være smartest at skubbe ansvaret for det ned i den webservicestak, der anvendes. Det vil kunne holde selve klientapi'et fri for at skulle håndtere SOAPheaders, og i stedet lade stakken håndtere det internt. Det samme vil gøre sig gældende i forhold til WSAddressingheaders. Disse er ikke en del af standardprofilerne, og skal ligeledes tilføjes eksplicit.

WSAddressing er nødvendigt for at identificere det endelige endpoint for den forespurgte service. Nogle services understøtter dog ikke WSAddressing endnu, og da nogle webservicestakke sætter mustUnderstand=”1” på WSAddressingheaders, vil disse kald umiddelbart fejle når de når til endpointet. Derfor understøtter SOSIGW en filtreringsmekanisme, så WSAddressingheaders automatisk kan fjernes inden et request sendes videre. Hvorvidt headers skal fjernes konfigureres pr. endpoint. WSAddressing understøttes i øvrigt kun til at finde endpoint – det forudsættes at svaret sendes tilbage på samme kanal som requestet kom. SOSIGW understøtter derfor ikke at der svares på en ny service, som klienten selv udstiller. Endelig understøtter SOSIGW heller ikke, at svar kommer tilbage fra endpoint på andre kanaler.

Håndtering af implicit Idkortoprettelse ved proxy-requests

Hvis der er et tilstrækkeligt udfyldt idkort i et proxy-requestet, mens der ikke findes et signeret idkort i SOSIGW, bliver der oprettet et nyt usigneret idkort. Fra det fremsendte idkort benyttes data fra afsnittene “Obligatoriske system og organisationsoplysninger” og “Eventuelle brugeroplysninger”, jævnfør “Den Gode Webservice”. Digest af idkortet samt signingurl sendes retur til klienten i en ny SOAPheader. Det er op til klienten at inspicere fejlbeskeder og headers for at afgøre om den skal præsentere brugeren for en signeringsdialog, enten indbygget i klienten, eller gennem start af en browser mod den returnerede url.

Det skal bemærkes, at denne implicitte oprettelse af idkort har den konsekvens, at den etablerede kontrakt mellem klienten og den oprindelige serviceudbyder bliver brudt, da der pludselig kan opstå tilfælde, som ikke var planlagt oprindeligt. Konkret betyder det, at en klient genereret ud fra en WSDL-fil ikke vil være forberedt på alle fejlmulighederne, og at de nye situationer derfor bliver svære at håndtere.

Iflg. DGWS er de eneste tilladte returkoder fra en service 200 (OK) eller 500 (fejl). I det aktuelle tilfælde med implicit login, skal der hæftes noget mere på svarene for at skelne mellem en “normal” servicefejl og det tilfælde, hvor det ikke er muligt at kalde servicen pga. manglende idkort. Især ved autogenererede klientstubbe bliver det problematisk, da de genererede klasser ikke er beregnet på at håndtere disse tilfælde.

Endelig har implicit oprettelse den svaghed, at hvis der ikke er oprettet et idkort inden et request, så skal det fulde request alligevel sendes til SOSIGW, som straks derefter vil afvise det. Især ved store requests, fx hvis der sendes billeder, vil dette være et stort spild.

requestIdCardDigestForSigning

Denne benyttes til eksplicit oprettelse af et Idkort, klar til signering. Servicen kaldes af en klient for at etablere en session for en bruger, så der kan sendes signerede requests til andre services.

Servicen kræver et partielt idkort, som SOSIGW skal opbevare i signeret tilstand. I SOAPbody sendes derfor en SAMLassertion med samme indhold som for implicit oprettelse. Servicen returnerer den digestværdi som klienten skal signere sammen med en URL, der kan anvendes af en browser. Der skelnes ikke mellem URLrequests og WSrequests.

Usignerede idkort fjernes automatisk fra serverne hvis der ikke er modtaget en signeret digest inden et vist interval. Dette interval er konfigurerbart.

signIdCard

Denne metode benyttes til at tilføje et signeret digest til det cachede idkort i SOSIGW. Argumenter til dette kald er NameID, SignatureValue og brugerens offentlige nøgle. SOSIGW lader herefter STS'en signere idkortet, checker at signaturen er gyldig, og gemmer det i sin cache.

getValidIdCard

Denne metode henter idkortet for en specifik bruger. Argumentet er NameID for brugeren, og returværdien er brugerens idkort eller en fault.

Metoden er et supplement til de metoder, der er angivet i kravspecifikationen, men er nødvendig for at håndtere to tilfælde pænt:

- Når et idkort er ved at blive signeret af en bruger gennem browserbaseret signering, er der ikke noget feedback der fortæller klienten hvornår brugeren er færdig med signeringen. Klienten har derfor to muligheder: Poll serveren periodisk via denne metode eller lad brugeren manuelt indikere når processen er færdig.

Metoden erstatter [Krav 10a], som havde samme effekt, bare med et blokerende kald til serveren. Det blokerende kald har den ulempe, at det optager serverressourcer i længere tid, og kan derfor være en hindring for høj skalabilitet. Til gengæld bliver en del af ansvaret flyttet til klienten, der aktivt skal checke for idkort, men dette er vurderet som den bedre løsning.

- Det andet tilfælde er det, hvor en webservice kræver et andet timeoutniveau (i DGWStermer). Nogle services kræver fx, at idkort er underskrevet for hvert request, og denne situation skal håndteres i klienten. Klienten er derfor nødt til at kunne se, om idkortet er for gammelt til at kunne bruges, og det kan ses ved at benytte getValidIdCard, der returnerer brugerens idkort.

Et tredje tilfælde er det, hvor en klient selv står for at kontakte en service, men ikke vil håndtere opbevaringen af idkort og kommunikationen med STS. Her kan klienten få det underskrevne idkort fra SOSIGW og selv vedhæfte det i en besked der skal sendes.

Hvis der ikke eksisterer noget idkort for brugeren, returneres der en fault. Denne fault angiver status for brugerens idkort: Enten kan en bruger være i gang med signering, og der kan derfor med god mening prøves igen lidt senere, eller også er der slet ikke oprettet noget idkort, og det giver derfor ikke mening at prøve igen, før et nyt idkort er bestilt.

Metoden kan i øvrigt konfigureres fra serveren, så selve idkortets signerede digest ikke sendes med ud, da det i visse tilfælde kan betragtes som et brud på sikkerheden.

logout

Denne metode fjerner det signerede idkort for en bestemt bruger. Som argument kræves NameID.

Tilgang til services til at håndtere IDKort

Der er udstillet to snitflader til at tilgå IDKort, begge som webservices. Den ene service udstiller den fulde funktionalitet, og kan passende beskyttes, så den kun kan tilgåes fra bestemte fagsystemer.

Der anden webservice udstiller kun håndtagene requestIdCardForSigning og SignIDCard. Denne webservice er lavet, så disse håndtag kan udstilles til at bredere publikum, eksempelvis gennem SignOn Biblioteket.

HTTP API

En del af SOSIGW's funktionalitet sker gennem et normalt HTTP API for at understøtte, at normale browsere kan komme i kontakt med systemet for at signere digests. Det sker ved at klienten får svar på et request om at oprette et nyt idkort,

enten eksplicit eller implicit, og bruger den url, der kommer ud af det, til at sende til en browser. For at understøtte proxy/nat mellem browserne og SOSIGW, er det muligt at konfigurere hostadressen for SOSIGW og dermed gøre det muligt at sende requests gennem en proxy på det lokale netværk.

Browseren åbner adressen og bliver præsenteret for en Javaapplet, der kan bruges til at signere digestværdien fra det nye idkort. Adressen er en engangsadresse, der er identificeret af en lang og tilfældigt genereret tekststreng, og som kun er gyldig i max 30 sekunder. Serveren genererer herefter en ny URL, som bruges af appletten til at sende svaret. Denne URL er gyldig i 60 sekunder, hvilket betyder at brugeren skal have signeret inden dette tidsrum. Sker det ikke, vil der fremkomme en fejlmeddelelse når brugeren prøver at signere. På siden med appletten vil tiden være angivet. Gyldighedsperioderne kan konfigureres i administrationsinterfacet.

Den URLbaserede signering baseres på OpenSign, der er en Javaapplet, som implementerer browserbaseret signering. Resultatet af denne applet er en XMLrepræsentation af signaturen, struktureret i forhold til W3C's XML Signature skema.