Norsk Regnesentral
ANVENDT DATAFORSKNING
Norwegian Computing Center/Applied Research and Development

NOTAT/NOTE

Web-skjemaløsninger for digital signatur for SLN

GEM/02/2000

Anund Lie, Gjertrud Pedersen

Oslo
april 2000

Norsk Regnesentral
ANVENDT DATAFORSKNING

NR-notat/NR Note

Tittel/Title:

Web-skjemaløsninger for digital signatur for SLN

Dato/Date: 14. april
År/Year: 2000
Notat nr: GEM/02/2000
Note no:

Forfatter/Author:

Anund Lie, Gjertrud Pedersen

Sammendrag/Abstract:

I Skattedirektoratets SLN-prosjekt er det ønskelig med en innrapporteringsløsning basert på elektroniske skjema via Web. De vanlige Web-leserne har etter hvert rimelig god støtte for kryptografi for å sikre kommunikasjonskanalen mot Web-tjeneren, men ingen innebygde mekanismer for digital signering av innholdet. Dette kan realiseres ved utvidelser av Web-leserne, men slike løsninger er ikke helt ukompliserte for brukerne å installere og konfigurere. Derfor er det ikke helt opplagt at løsningen vil oppfattes som enklere i bruk enn den EDI-baserte løsningen som finnes for SLN pr. dato (evt. en forbedret utgave av denne).

Emneord: digital signatur, elektronisk skjema, Web
Indexing terms: digital signature, electronic form, Web

Målgruppe/Target group:

Tilgjengelighet/Availability: åpen

Prosjektdata/Project data:

Prosjektnr/Project no: 434 000

Antall sider/No of pages: 33

Satsningsfelt:
Research field:

Norsk Regnesentral/Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, N-0314 Oslo, Norway
Telefon 22 8525 00, telefax 22 69 76 60, telex 76518 ncc n

Sammendrag

Det er ønskelig med en enkel, Web-basert innrapporteringsløsning for signerte ligningsskjemaer i Skattedirektoratets SLN-prosjekt (System for ligning av næringsdrivende). Det er allerede utviklet en EDI-løsning, som imidlertid anses som for tungvint i bruk for enkeltpersoner og mindre bedrifter, bl.a. fordi den krever at en del programvare installeres og vedlikeholdes lokalt hos den som skal signere og sende inn skjemaene. Hovedkomplikasjonen i forbindelse med en Web-løsning er behovet for en kvalifisert digital signatur på skjemaet. Ingen av de vanlige Web-leserne har noen (brukbar) innebygget mulighet for dette, og dette medfører at en eller annen form for utvidelse (programvare, utstyr, drivere) uansett må installeres lokalt på klientmaskinen for å få til en signaturfunksjon som tilfredsstiller kravene.

Web-løsningen forutsetter aksess til en skjemadatabase som ligger hos Skattedirektoratet. Siden signaturløsningen allerede krever smartkort og utstedelse av sertifikater til brukerne, kan aksesskontroll mot denne databasen håndteres på en sikker måte ved bruk av SSL/TLS med klientautentisering.

Konklusjonen er at det er mulig å realisere en Web-løsning som er integrert med digital signering av skjema. Denne løsningen krever likevel betydelig mer av brukeren enn en tradisjonell Web-løsning, og en del av dette er ikke til å komme forbi når det først er krav om kvalifisert digital signatur:

- Smartkortstøtte må installeres lokalt.
- Programvare for signaturgenerering må enten installeres lokalt, eller sikkerhetssystemet i Web-leseren må konfigureres slik at dette kan lastes ned over nettet.
- Brukeren må få utstedt og installere sertifikat og nøkler.

Løsningen vil neppe være en tradisjonell HTML-skjema-løsning med digital signatur tilføyd, men i stedet en ny skjema-innholdstype som håndteres i et eget hjelpeprogram der digital signatur er tett integrert.

Hvor brukbar løsningen vil bli oppfattet som, avhenger i første rekke av kvaliteten på implementasjonen av alle disse detaljene. Det er ikke opplagt at den vil være enklere i bruk enn den EDI-løsningen som allerede finnes, eller en forbedret versjon av denne der installasjon, konfigurasjon og vedlikehold er mer strømlinjeformet.

1. Innledning: Sikkerhetskrav

1.1. Digital signatur: Krav til løsningen

En digital signatur, slik begrepet vanligvis forstås, realiserer disse to sikkerhetstjenestene:

- *Autentisering*: At innholdet av dokumentet virkelig stammer fra den som angivelig har signert det, dvs. beskyttelse mot en utenforstående bevisst har forfalsket det.
- *Ikke-benekting*: At den som har signert dokumentet ikke senere kan benekte det, dvs. beskyttelse mot forfalskning (bedrageri) fra denne parten selv.

Ikke-benekting kan oppfattes som en sterkere form for autentisitet. Legg merke til at for tilfellet autentisering er det ikke noen fundamental mistillit mellom partene. Krav om ikke-benekting er alltid begrunnet i en viss mistillit mellom partene selv.

I praksis er grensen mellom disse to tjenestene litt flytende. I virkelige tilfeller vil graden av ikke-benektbarhet avhenge av hvilken motivasjon underskriveren har for svik, og hvilke sanksjonsmuligheter en tredjepart (f.eks. via domstolene) kan gjøre gjeldende. Det er altså noen tekniske forutsetninger som må være tilstede for å gi ikke-benekting, men **styrken** vil avhenge av andre forhold (lovverk, avtaleforhold mellom partene o.l.)

For at en digital signatur skal kunne erstatte en håndskrevet signatur, vil lovverket stille krav om en såkalt *kvalifisert* digital signatur (egentlig: at signaturen kan bekreftes ved et *kvalifisert sertifikat*). Dette betyr i realiteten at behandlingen av nøkler, signaturgenereringen, sertifiseringstjenesten og sertifikatutsteders ansvar tilfredsstiller visse krav. Et kvalifisert sertifikat gir ikke-benekting, nærmest pr. definisjon. (Men et kvalifisert sertifikat er ikke **nødvendig** for å oppnå en viss grad av ikke-benektbarhet.)

For ligningsoppgaver gjelder krav om signatur. I det som følger vil vi ta det for gitt at det kreves en kvalifisert digital signatur for å oppfylle dette kravet i SLN. (Det kan likevel være verd å vurdere om andre løsninger, uten kvalifisert signatur på ligningsoppgaven, ville kunne være akseptable: Selv om ligningslovens krav tilsier signatur, må vi anta at signaturen er et middel til å forplikte oppgavegiver, og at andre løsninger med samme juridiske effekt vil være gode nok.)

Vi starter derfor med å sammenligne kravene til en digital signaturløsning som gir ikke-benekting og en ren autentiseringsløsning.

1.2. Forutsetninger for autentisitet

1.2.1. Grunnleggende sikkerhetskrav

Den mest grunnleggende forutsetningen for autentisitet er full kontroll over det utstyret og den programvaren som på noen måte kan påvirke genererings- og verifikasjonsprosessen for den digitale signaturen. Det er stor oppmerksomhet på styrken av kryptografiske algoritmer og muligheten for forfalskning av nøkler. Dette er nødvendige ledd i kjeden (se nedenfor). Men andre ledd er like viktige og, dessverre, lettere å overse.

- Den som utsteder dokumentet forholder seg aldri like direkte til dokumentet som til et papirdokument. Dokumentet produseres og presenteres av en eller annen applikasjon, selv i det tilfellet hvor applikasjonen gir illusjonen av direkte redigering (WYSIWYG). Utstederen er avhengig av at programvaren presenterer for signering noe som er en korrekt representasjon av utstederens intensjoner.

- Nøkkelen som benyttes ved signaturgenerering må holdes hemmelig, både ved generering, lagring og bruk. All håndtering og bruk av nøkkelen innebærer en viss risiko for lekkasje av informasjon eller for misbruk.

- Ved signaturverifisering kreves tillit til verifikasjonsnøkkelen, dvs. til nøkkelens integritet og til at koblingen mellom nøkkel og partsidentitet er korrekt.

- Mottakeren er på sin side avhengig av programvare for presentasjon og viderebehandling av det mottatte dokumentet.

Alle disse stegene er i prinsippet like viktige. **Den digitale signaturen vil aldri kunne gi større tillit enn den tilliten en har til hvert enkelt av disse stegene**. Truslene er:

- *Tilfeldige feil i programvare og utstyr* (altså feil som ikke bevisst er introdusert). I mange tilfeller vil slike feil føre til at autentisering feiler, og dermed oppdages uten at noen vesentlig skade er skjedd. Men slike feil kan også føre til at hemmelige nøkler kompromitteres, at ugyldige verifikasjonsnøkler aksepteres som ekte eller falske dokumenter

aksepteres som autentiske. **Spesielt er det en risiko for at kjente feil kan utnyttes av utenforstående angripere**.

- *Bevisste angrep* (trojanske hester, virus) kan modifisere eller rekonfigurere programmer, databaser, kataloger osv. på en måte som i verste fall **kompromitterer hvilket som helst av stegene ovenfor**. Tilliten til en signatur avhenger altså av hvor godt beskyttet mot slike angrep avsenders og mottakers systemer er og hvor effektivt de er i stand til å begrense konsekvensene.

- *Brukerfeil* kan føre til at hemmelige nøkler kommer på avveier eller brukes til å signere dokumenter uten skikkelig kontroll, til at ugyldige sertifikater aksepteres og brukes ved signaturverifikasjon o.l., til at sikkerhetspolicy i Web-leser eller epost-klient konfigureres for løst m.m.

Populære plattformer, som Microsoft Windows (95, 98, men også NT), er svært dårlig beskyttet: **Ett eneste angrep av makrovirus, ondsinnede Active-X-kontroller eller lignende er i prinsippet nok til å ødelegge tilliten til digitale signaturer utført eller verifisert på den aktuelle maskinen.** Brukes programvareløsninger for nøkkeloppbevaring, er også tilliten til nøklene ødelagt. (Spesielt gjelder det hemmelighold av signeringsnøkler, men også for verifikasjonsnøkler siden angrepet kan ha merket forfalskede nøkler som gyldige.) Gode smartkortløsninger hindrer riktignok at hemmelige nøkler kommer på avveier, men kan ikke hindre at de samme nøklene blir misbrukt til å signere forfalskede dokumenter (dvs. dokumenter som er konstruert av angriperen, og som undertegneren blir lurt til å signere).

Plattformer med flernivå sikkerhet (og en løsning med nøkler i smartkort er egentlig et eksempel på det) gir større muligheter til å begrense konsekvensene av et angrep, gitt at mekanismene blir korrekt brukt.

1.2.2. Krav til kryptografiske løsninger

Kryptografiske algoritmer, protokoller og nøkler som brukes ved signaturgenerering, og -verifikasjon, nøkkelgenerering, -utveksling og sertifisering må være (tilstrekkelig) sikre. Ved å velge anerkjente løsninger kan vi forutsette at dette kravet er oppfylt. Det er også en rent teknisk problemstilling som kan vurderes nokså uavhengig av de andre forholdene vi tar opp, og hvor analysene i den generelle faglitteraturen på området gjelder, slik at det ikke er noen grunn til å gå dypere inn på dette her. Vi vil bare påpeke:

- Det er knapt noe teknisk problem å velge sterke nok løsninger — de er ikke så mye mer ressurskrevende i bruk at det har noen praktisk betydning. (Men politiske reguleringer av krypto gjør dem mindre kommersielt tilgjengelig.)

- Selv om medieoppslagene rundt "knekking av krypto" fokuserer på disse faktorene, er de andre forholdene vi diskuterer i praksis vel så stor risiko. (Det er ingen vits i å bryte kryptonøkler hvis andre svakheter i systemene kan gi angriperne adgang til den samme informasjonen ukryptert…)

1.2.3. Krav til sertifiseringstjeneste

En sertifiseringstjeneste er ikke absolutt nødvendig når digital signatur bare benyttes for å verifisere autentisitet, så lenge partene kjenner hverandre på forhånd og er i stand til å utveksle nøkler på en sikker måte. I de tilfellene der det bare er krav om autentisitet, har partene i utgangspunktet ikke noen grunn til mistillit til hverandre, men bare til å frykte angrep fra utenforstående.

Det er likevel gode praktiske grunner til å bruke en sertifiseringstjeneste. I praksis kan en ikke forutsette at partene kjenner hverandre. Når en etablert PKI-infrastruktur finnes, er dette en god mekanisme for å utveksle nøkler.

Det mest fundamentale kravet til en sertifiseringstjeneste er de grunnleggende sikkerhetskravene:

- Bindingen mellom identitet og nøkkel gjennom sertifikatet er korrekt.
- Mottakere av sertifikater har en effektiv måte å verifisere at sertifikatet er gyldig (gjennom katalogtjeneste og/eller tilbakekallingslister).
- Hvis private nøkler genereres hos sertifiseringsautoriteten, at de holdes hemmelig.
- At sertifiseringsautoritetens interne systemer opereres sikkert.

Når partene ikke kjenner hverandre, har sertifiseringsautoriteten et annet (ikke-teknisk) ansvar. Det er sertifiseringsautoriteten (evt. indirekte via en registreringsautoritet) som går god for den parten som blir sertifisert:

- At sertifikat bare utstedes til parter som har krav på de rettighetene eller den tilliten som sertifikatet impliserer.
- At parten selv opererer sine systemer for signaturgenerering på en betryggende måte.

Nøyaktig hvordan sertifiseringsautoriteten utøver dette ansvaret er definert i sertifiseringspolicy og -praksis. I praksis vil sertifiseringsautoriteten gi en garanti overfor dem som mottar og stoler på sertifikater. Den som vil ha utstedt et sertifikat, må på sin side inngå en avtale med sertifiseringsautoriteten, som gir sertifiseringsautoriteten en sanksjonsmulighet hvis sertifikateieren misligholder sin del av avtalen (dvs. et regresskrav). Sertifiseringen er f.eks. ikke noe verd dersom den som har fått et sertifikat ved uaktsomhet lar sin private nøkkel komme på avveie, så sertifiseringsautoriteten må ha betingelser som legger ansvaret på sertifikateieren i dette tilfellet.

1.3. Forutsetninger for ikke-benekting

For å oppnå ikke-benekting må alle forutsetninger for autentisitet først være oppfylt. Dersom det er tvil om at et signert dokument er autentisk, vil den som angivelig signerte dokumentet uten videre kunne påberope seg dette og påstå at et omstridt dokument må være forfalsket.

Dette er et mønster vi finner igjen i flere punkter nedenfor: Enhver uønsket mulighet i systemet gjør det umulig for en tredjepart å avgjøre hva som faktisk skjedde, dvs. hvilken av partene som snakker sant. Den ene parten kan direkte utnytte hullet til sin fordel, f.eks. forfalske et dokument som utgir seg for å komme fra den andre. Men siden denne muligheten ikke kan utelukkes, kan den andre parten på sin side påstå svik, selv når det ikke er tilfelle: Den som har sendt et dokument kan etterpå påstå det må være forfalsket av mottakeren, og løpe fra ansvaret.

I avsnittene nedenfor tar vi for oss de punktene der ikke-benekting stiller sterkere eller andre krav enn autentisering.

1.3.1. Generelle sikkerhetskrav

Sikkerhetskravene generelt er ikke vesensforskjellige for ikke-benekting og autentisitet, men de vil i alminnelighet være sterkere for ikke-benekting. Grunnen er at systemet skal produsere bevis som er holdbare overfor en tredjepart (i verste fall, en domstol), når en må regne med at den ene eller andre parten kan komme til å bestride dem.

1.3.2. Oversikt og kontroll

Ikke-benekting krever at underskriveren der og da får presentert dokumentet som skal signeres og utløser signaturgenereringen. Dokumentet må være til stede og presenteres lokalt på den samme maskinen som signaturen faktisk beregnes. (Jfr. neste avsnitt).

Det er et opplagt krav at selve dokumentpresentasjonen på alle måter er korrekt og får frem dokumentets betydning. Men dokumentet må også presenteres i korrekt sammenheng, slik at det er klart hva konsekvensene av å signere er. (F.eks.: selv om dokumentet er det samme, er det en vesensforskjell mellom en kontrasignatur og en signatur som forplikter en person til innholdet — eller en forpliktende signatur og en signatur som bare tjener til å beskytte innholdet under overføring og lagring[1].)

1.3.3. Programvare- og utstyrsplattform

Underskriveren er alltid i siste instans den som er ansvarlig for integriteten til all programvare og utstyr som brukes til signeringen, altså at signaturløsningen virkelig signerer det (og bare det) som det var hensikten å signere. Siden det i praksis aldri er mulig for underskriveren selv å kontrollere at alle ledd i kjeden gjør det de er forutsatt, er det et tillitsforhold mellom brukeren og leverandører (av programvare og utstyr), driftspersonell (som installerer programvare og utstyr), nettverksoperatører (som setter opp brannvegger og forhindrer angrep) og andre som kan ha mulighet til å påvirke de komponentene som korrekt signaturgenerering avhenger av.

For ikke-benekting er dette forholdet spesielt viktig. I prinsippet delegerer underskriveren signaturmyndigheten sin til programvaren som genererer signaturen. **Spesielt betyr det at signaturen ikke kan genereres av programvare som leveres av den andre parten, eller på utstyr (f.eks. tjenere) som den andre parten opererer.** Dette ville undergrave hele hensikten med ikke-benekting, siden mottakeren nå kan anklages for selv å være skyld i feil i dokumentet som ble signert eller signaturen selv — i verste fall bevisst forfalskning.

1.3.4. Kryptografisk løsning

For autentisitet er symmetrisk krypto tilstrekkelig (dvs. en felles, hemmelig nøkkel). Ikke-benekting krever asymmetrisk krypto, ellers ville mottaker kunne fabrikere forfalskede, signerte dokumenter (og den som angivelig har signert vil kunne vise til denne muligheten og benekte signaturen).

Den litt tilfeldige betegnelsen “sterk kryptografi” viser til kryptosystemer (algoritmer, nøkkellengder) der det ikke er noen kjente, praktisk gjennomførbare angrep på selve kryptosystemet. For ikke-benekting og spesielt kvalifiserte signaturer er alt som ikke faller inn under dette begrepet uaktuelt. Spesielt er “eksport-grad”-kryptosystemene i Netscape og Internet Explorer for svake.

1.3.5. Nøkkel-bruksområder

Nøkler for bruksområdene autentisering og ikke-benekting må normalt holdes strengt adskilt. For ikke-benektingsnøkler vil en normalt kreve spesielt sterke sikkerhetsforanstaltninger:

- Underskriveren skal bekrefte (f.eks. ved PIN-kode) hver enkelt signatur.
- Separat autentiseringsmekanisme til nøkkelmediet for å aktivere nøkkelen (f.eks. en egen PIN-kode til smartkort, slik at underskriveren ikke er i tvil om hvilken nøkkel som brukes når kortet inneholder flere).

[1] Det bør nok være forskjellige nøkler for såpass forskjellige funksjoner, det siste er f.eks. forskjellen mellom key usage non-repudiation og key usage authentication - som riktignok forskjellige X.509-profiler ikke er helt samstemt om.

- Nøkkelen må bare brukes på tiltrodde utstyrsplattformer.

For autentiseringsnøkler er dette ofte overflødig, og i verste fall i konflikt med måten de brukes på.

En annen grunn er: De fleste autentiseringsprotokoller opererer ved at den som krever autentisering av den andre parten sender en tilfeldig utfordring, som den som skal autentiseres krypterer med sin private nøkkel. Digital signatur utføres på sin side ved å kryptere en hash-kode beregnet over dokumentet. Hvis samme nøkkel er gyldig både for autentisering og ikke-benekting, kunne motparten sende noe som angivelig var en tilfeldig utfordring, men i virkeligheten var hash-koden for et forfalsket dokument, og dermed oppnå en bindende signatur[2].

1.3.6. Sertifiseringstjeneste

En form for nøytral tredjepart er essensiell for å oppnå ikke-benektbarhet, mens sertifiseringsautoriteten bare er en praktisk ordning ved autentisering. I praksis er grunnlaget for ikke-benekting at den som signerer og siden løper fra ansvaret risikerer sanksjoner (f.eks. erstatningskrav eller straff) eller andre ubehagelige konsekvenser (f.eks. tap av tillit, ugyldighet av andre signaturer), i ytterste konsekvens ved at saken blir brakt inn for en domstol. Den som har fått utstedt et sertifikat (sertifikateier), har inngått en avtale med sertifiseringsautoriteten. Denne avtalen stiller som nevnt krav til sertifikateier, og er grunnlag for å stille sertifikateieren økonomisk ansvarlig ved mislighold. For et sertifikat som er gyldig for ikke-benekting vil bl.a. avtalen (ved å henvise til sertifiseringspolicy og -praksis) stille krav om forsvarlig håndtering av private nøkler og til å melde fra ved mistanke om at privat nøkkel er kommet på avveie. Hvis sertifikateieren da i etterkant benekter en signatur ved å påstå at privat nøkkel må være kompromittert, er det samtidig en innrømmelse av ikke å ha oppfylt sin del av avtalen.

Hvis saken bringes frem for en domstol, har sertifiseringsautoriteten rollen som nøytral tredjepart som frembringer bevismateriale.

1.3.7. Tidsavhengighet

Av tekniske årsaker, og på grunn av muligheten for tilbakekalling av sertifikater (i tilfelle den private nøkkelen blir kompromittert) er gyldigheten av sertifikater, og dermed digitale signaturer, alltid tidsbegrenset. For autentisering er dette sjelden noe problem, fordi mottaker av en signatur vanligvis verifiserer signaturen umiddelbart. Ved ikke-benektbarhet er problemet større, fordi signaturen må kunne verifiseres av en tredjepart når det oppstår en tvist, og dette kan bli nødvendig å gjøre på et tidspunkt da sertifikatet ikke lenger er gyldig.

Dette problemet kan løses ved tidsstempling og logging hos tredjepart. (Tidsstemplet er i prinsippet bare en ny signatur, utført av tredjeparten, som dekker dokumentet eller den opprinnelige signaturen pluss en angivelse av tidspunktet, altså en attest fra tredjeparten om at dokumentet er sett og logget på et visst tidspunkt. Loggingen er det primære; tredjepartens grunnlag for å kunne bekrefte signaturen på et senere tidspunkt.) Tredjeparten kan dermed bevitne at dokumentet eller signaturen forelå på et tidspunkt da sertifikatet signaturen viser til var gyldig.

1.3.8. Entiteten som signeres

Entiteten (dokumentet) som signeres må en veldefinert betydning, som også en tredjepart kan etterprøve. Hvis ikke er ikke-benekting meningsløst, signaturen bekrefter bare at

[2] Praktisk brukte protokoller for autentisering og signatur vil vanligvis føye til noe til utfordringen hhv. hash-koden før krypteringssteget, og da er ikke dette noe reelt problem. Risikoen for å komme i skade for å bruke nøkkelen i utstyr/protokoller som ikke tar denne forholdsregelen tilsier likevel forsiktighet. Det finnes også angrep på privat nøkkel i noen kryptosystemer dersom en krypterer ukjente data (dvs. valgt av angriperen).

partene har utvekslet en viss bitstreng, men de er ikke enige om hva denne bitstrengen betyr.

- Syntaks og semantikk (data- eller dokumentformatet) for det som blir signert må være definert (i en standard eller spesifikasjon som begge parter slutter seg til).

- Betydningen av det som er signert kan ikke avhenge av noe utenfor dokumentet selv (som ikke er omfattet av signaturen eller avtaler som partene har inngått). Det er nettopp for å understreke at det som er signert må være et sluttet hele at vi bruker ordet dokument.

Dette betyr at ikke-benekting vanskelig kan gjøres gjeldende for transportlags- eller nettverkslags-sikkerhet (f.eks. SSL/TLS[3]). For det første: På dette nivået oppfattes innholdet i pakkene eller datastrømmene bare som en bitsekvens (selv om det kan være mulig å rekonstruere protokollen på applikasjonsnivå). Dessuten vil hver enkelt pakke normalt signeres hver for seg, mens betydningen av innholdet i den avhenger av hva som tidligere har vært utvekslet (og ikke nødvendigvis bare i denne sesjonen).

2. Hvorfor Web-løsning?

Formålet med en Web-løsning er å få på plass en tjeneste som er enklest mulig å ta i bruk for sporadiske brukere, altså enklere enn den EDI-baserte pilotløsningen som nå er utviklet. (Problemet med EDI-løsningen antas å være at den er for tung og komplisert å installere og ta i bruk, ikke med EDI som løsningskonseptet i seg selv.)

For å oppnå dette, må tjenesten klare seg med vanlige Web-lesere som klient, og fortrinnsvis med bruk av automatisk installerbar programvare der det trengs spesialisert funksjonalitet (dvs. plugins, Java-applets eller Active-X-kontroller).

Det er ikke mulig å unngå at **noen** komponenter må installeres lokalt av brukeren selv:

- Smartkortleser og drivere: Smartkort er nødvendig som nøkkelmedium for å tilfredsstille kravene til kvalifisert signatur. Dette vil imidlertid dreie seg om generelle løsninger som vi må anta brukeren har en egeninteresse av å installere for å kunne bruke smartkort i andre sammenhenger, og som på sikt vil være en standard del av brukernes plattform. Installasjonen er ikke spesielt komplisert: Typiske lesere på PC-plattform kobles til en COM-port, og drivere installeres på vanlig måte for Windows.

- Smartkort med nøkler og sertifikater, må skaffes direkte fra sertifiseringsautoritet. På samme måte som for leser og driver vil vi anta at det er mulig å legge til rette for en løsning (valg av sertifiseringsautoritet, policies, navnestruktur) som gjør nøklene og sertifikatene generelt nyttige for brukerne.

Antakelig vil smartkort, -leser, driver, nøkler og sertifikater kunne leveres som en pakke fra aktuelle sertifiseringsautoriteter, og kunne brukes til alle typer kryptografiske tjenester: Signering og kryptering av e-post, Web-klient-autentisering, smartkortbasert innlogging, virtuelle private nett m.m. (Signert/kryptert e-post og Web-klient-autentisering ved bruk av smartkort støttes av dagens Web-lesere. Windows 2000 har

[3] SSL/TLS har andre egenskaper som gjør dem uegnet til ikke-benekting, bl.a. at MAC-er ("digitale signaturer" over innholdet i pakkene) beregnes med symmetriske nøkler som er kjent av begge parter. (Offentlige nøkkelsystemer er i alle tilfeller uegnet til å signere så små enheter, siden krypterings- og dekrypteringsoperasjonene er tidkrevende, så dette problemet må en vente å finne med de fleste praktisk brukbare protokollene i denne kategorien.)

integrert støtte for sterk kryptografi og smartkort.) Smartkortleseren vil dessuten kunne brukes til andre smartkortbaserte tjenester, f.eks. betalingstjenester. Derfor er vi ikke spesielt bekymret for å kreve at brukeren anskaffer og installerer dette. Vi kommer tilbake til smartkortløsningene i et senere kapittel.

3. Mekanismer for Web-leser-utvidelser

Web-lesernes funksjonalitet må tilpasses og utvides, både for å kunne presentere dokumentet lokalt før signering og for å kunne generere selve signaturen. Det finnes forskjellige mekanismer for å gjøre dette gjennom nedlastet kode, men mange av disse har iboende svakheter eller restriksjoner som gjør at de ikke uten videre kan brukes.

Et felles problem for alle løsninger er sikkerhetsrisikoen (virus og trojanske hester) ved å laste ned kode over nettet og kjøre den. En løsning på dette er å hindre nedlastet kode i å utføre sikkerhetskritiske funksjoner, altså behandle koden som ikke-tiltrodd. Denne modellen brukes i JavaScript og Java applets. Dette impliserer imidlertid at koden ikke kan utføre kritiske operasjoner som bl.a. signaturgenerering, som vi har behov for. (Selv om sikkerhetsmodellen i JavaScript og Java i teorien er trygg, er det dessuten mange eksempler på feil i implementasjonene som åpner mer eller mindre alvorlige hull i "sandkassen".)

Den andre løsningen er å autentisere koden slik at den (til en viss grad) kan behandles som tiltrodd, enten ved at koden som lastes ned er digitalt signert, eller ved at den lastes ned via en trygg kanal (SSL). Men dette er ikke problemfritt: Kode som skal brukes til å generere en kvalifisert signatur bør prinsipielt også være autentisert gjennom en kvalifisert signatur. Det er også verd å merke seg at de etablerte oppleggene for kodesignering vanligvis ikke innebærer noe krav om kvalitetskontroll av koden. (Det går i prinsippet an å sette opp sertifiseringsautoriteter som i sertifiseringspolicy krever en definert grad av kvalitetskontroll for signert kode, men vi kjenner ikke til at noen slike er operative. En slik sertifiseringsautoritet bør kunne utstede kvalifiserte sertifikater.)

Disse to typene mekanismer bør kombineres, slik at autentisert kode ikke får fulle rettigheter, men at rettighetene graderes etter hvilken tillit en har til ansvarlig utgiver.

Installasjon av programvare fra nettet uten å bruke noen av disse sikkerhetsmekanismene er (dessverre) vanlig praksis, men **bør i det hele tatt ikke forekomme på en maskin som skal kunne generere kvalifiserte signaturer**. Det er i alle fall uakseptabelt å gjøre det med programvare som er direkte involvert i sikkerhetskritiske funksjoner som signaturgenereringen.

3.1. HTML-scripting

Det finnes en håndfull forskjellige scriptspråk som kan brukes for scripting i HTML-sider. Netscapes *JavaScript* og Microsofts *JScript* er noenlunde kompatible, og nærmer seg en felles standard *ECMAScript* (se f.eks. http://www.cetus-links.org/oo_javascript.html). I Internet Explorer finnes også *VBScript*, som er en delmengde av Visual Basic.

Alle script-språkene har en begrenset grad av aksess til elementene i HTML-dokumentet (bl.a. felter i skjema), og til Web-leser-vinduet (bl.a. mulighet for å åpne dialogbokser eller åpne andre Web-sider). Grensesnittet mot HTML-dokumentet og Web-leseren betegnes ofte som DOM (Document Object Model), som strengt tatt ikke er en del av scriptspråket selv, og ikke er like standardisert[4] som språket.

Script er **ikke-tiltrodd** kode, siden brukeren kan motta og utføre script fra ukjente og potensielt ondsinnede kilder. Derfor er det svært begrenset hvilke funksjoner som kan

[4] Den generelle dokumentmodellen som utvikles i W3C under navnet DOM, og gjelder for XML-dokumenter generelt, er **ikke** den samme som dokumentmodellen i JavaScript og JScript.

utføres fra script. Generelle signatur- eller kryptofunksjoner, eller aksess til smartkort/-leser bryter med sikkerhetsmodellen for script, og ville vært et alvorlig sikkerhetshull (på linje med muligheten for å lese og modifisere lokale filer.)

Netscapes JavaScript (i Communicator 4.x) har en utvidelse for å kunne generere digital signatur på en begrenset måte. Funksjonen `crypto.signText` signerer en tekst (generert av scriptet). Hver gang funksjonen kalles, presenteres teksten i en dialogboks for brukeren, som eksplisitt må godkjenne signaturgenereringen. Denne mekanismen håndterer bare korte tekster som kan presenteres i dialogboksen, og er ikke praktisk brukbar for å signere et stort skjema.

Den har imidlertid noen nøkkelegenskaper som gjør den rimelig trygg:

- Signeringsmekanismen er en innebygd del av den lokale Web-leseren (som i seg selv tiltrodd).
- Mekanismen kobler presentasjon av det som skal signeres uløselig sammen med at brukeren bekrefter signaturen, uten at scriptet kan påvirke dette på noen måte.
- Scriptet håndterer ikke PIN-koder eller tilsvarende.

Alle andre mekanismer for signaturgenerering fra script må ha disse egenskapene for å være akseptable. Såvidt vi vet er dette den eneste ferdige signaturgenereringsmekanismen som finnes i noen av de aktuelle Web-leserne.

Teksten som signeres med `crypto.signText` kan formateres med HTML-oppmerking (lagt inn av scriptet). Vi har ikke undersøkt nøyaktig hva som er mulig, men dette kan være en sikkerhetsmessig svakhet dersom formateringen kan utnyttes til å villede brukeren om hva som faktisk signeres.

3.2. Script-språk med utvidelser

Mekanismene som i tilfelle må brukes for å få til signaturgenerering finnes i DOM, og er altså til en stor del forskjellige mellom forskjellige Web-lesere. Det å hente ut innhold fra et HTML-skjema ser ut til å være mulig å gjøre på en felles måte, mens selve signaturgenereringen må gjøres på helt forskjellige måter for forskjellige Web-lesere.

Det finnes noen muligheter til å installere utvidelser (dvs. nye objekter/funksjoner i objektmodellen) som kan gi anledning til å aktivere en digital signatur-mekanisme. I praksis ville dette måtte skje gjennom Active-X-kontroller (VBScript, ActiveX-objekt i JScript) eller via Java (Netscapes LiveConnect, Microsofts Java/Active-X-bro), altså plattformspesifikke løsninger. Merk at implementasjonen av utvidelsen må være tiltrodd, m.a.o. lokalt installert av brukeren selv eller som signert kode via nettet. (Se underkapitlene om signert Java og Active-X.)

For ikke-tiltrodde script vil selve signaturmekanismen i sin helhet måtte legges i en utvidelse, f.eks. en Active-X-kontroll merket “safe for scripting”. Selve mekanismen måtte være utformet tilsvarende `crypto.signText` ovenfor. Dette vil kunne være en akseptabel løsning, men en må være oppmerksom på risikoen for at scriptet aktiverer signaturfunksjonen på noe som ikke skulle vært signert.

3.3. Signerte script

JavaScript 1.2 og høyere (Netscape) tilbyr også *signerte* script. Sikkerhetsmodellen er avledet av Java, men har vesentlige forskjeller pga. fundamentale forskjeller[5] i språkene.

[5] Java har klasse-basert arv og ett av fundamentene for tillit er at ikke hvem som helst kan introdusere nye klasser eller overstyre systemklasser. JavaScript har forekomst-basert arv og trenger andre mekanismer for å hindre at trojanske hester omgår sikkerhetssystemet.

Stort sett de samme sikkerhetsbetraktningene som for signert Java-kode gjelder (se nedenfor.)

JavaScript (dvs. JavaScripts DOM) inneholder ikke noen mekanismer som kan brukes til signaturgenerering (bortsett fra `crypto.signText`), og signaturgenereringen vil stadig måtte basere seg på egenutviklede utvidelser (f.eks. Java-objekter via LiveConnect). Fordelen med signerte script vil i tilfelle være større frihet i hvordan disse utvidelsene blir utformet, f.eks. at de kunne bruke Active-X-kontroller som ikke generelt er "safe for scripting".

3.4. Java-applet

På samme måte som script er usignerte Java-applets **ikke-tiltrodd** kode. Sikkerhetsmodellen for Java blokkerer for signaturgenerering, aksess til smartkort etc. Det finnes heller ikke noen standard mekanisme i Java tilsvarende Netscapes `crypto.signText`.

Applets kan brukes for dokumentpresentasjon. Men presentasjon av et dokument for at underskriveren skal kunne kontrollere det før signering er en sensitiv operasjon som **ikke** bør utføres av ikke-tiltrodd kode. Vinduer som presenteres av en applet er identifisert som sådan, nettopp på grunn av appletens mulighet til å presentere et brukergrensesnitt som gir seg ut for noe annet enn det er. Ingen sikkerhetsbevisst bruker må gjøre noen sikkerhetskritiske aksjoner bare på grunnlag av informasjon presentert i slike vinduer.

3.5. Signert Java-applet

Signerte applets har alle de muligheter som trengs for denne anvendelsen. Avhengig av hvem som har signert en applet, kan brukeren selv tildele rettigheter, dvs. på kontrollert vis åpne veier ut av Java-"sandkassen".

Brukeren må altså:

- (Importere sertifikatet til sertifiseringsautoriteten som har sertifisert applet-utgiveren.)
- (Importere sertifikatet til applet-utgiveren.)
- Åpne Java "privilege manager"-dialogen for applet-utgiverens sertifikat og krysse av for de navngitte rettighetene som kreves, f.eks. det som gir adgang til smartkort og signaturfunksjoner.
- Alternativt ber appleten selv om de privilegiene den trenger, og brukeren må bekrefte at dette er godtatt.

Alle disse operasjonene bør[6] bekreftes manuelt, slik at brukeren er klar over hva han gjør og konsekvensene av det.

Når først koden er signert er det sikkerhetsmessig nesten irrelevant hvilken tjener den faktisk lastes ned fra. Skatteetaten kan f.eks. operere en Web-tjener for mottaket som leverer ut appleter som er signerte av en uavhengig tredjepart.

Noen av plattformene (Communicator 4.x) kan, som et alternativ til å signere koden, håndtere kode som er lastet ned via en SSL-forbindelse (`https:`) som tiltrodd: Privilegiene tildeles da på basis av tjenerens SSL-sertifikat (altså: som om koden var signert med dette sertifikatet).

[6] Såvidt vi vet finnes det heller ikke noen automatisk måte å importere et sertifikat og tildele privilegier i noen av de aktuelle plattformene, slik at brukeren faktisk **tvinges** til å gjøre dette manuelt.

Problemet med signerte Java-applets er at det er svært varierende støtte i ulike Java-plattformer, og Java-koden må tilpasses plattformen den skal kjøre på.

- JVM 1.1 utfører signerte applets som om de er lokale (dvs. full tillit, samme modell som for Active-X-kodesignering, med de samme problemene, se nedenfor.)
- Microsoft Java svarer noenlunde til JVM 1.1.
- Netscape Communicator 4.x har en fleksibel mekanisme for å be om og tildele rettigheter, basert på opphavet til koden. Systemet av rettigheter er dessverre ikke integrert med kryptografi-støtten i Java.
- JVM 1.2 (Java 2) har en mekanisme som i prinsippene er ganske lik Communicators, men forskjellig i detaljene.

For å få en uniform plattform, er det mulig at signerte applets må kjøres i Suns Java-omgivelse, som kan installeres som en plugin i både Netscape og Internet Explorer. Dette er en ekstra komplikasjon.

Java-løsningen kan generere signaturer på to ulike måter, og reguleringen av privilegier blir litt forskjellig for de to tilfellene:

- Ved å benytte kryptografistøtte i Java Cryptography Extension. I JVM 1.2 styres adgangen til signaturgenerering ved å tildele *SecurityPermission(getSignerPrivateKey)*-tillatelsen. (Dette gir generell adgang til å bruke — ikke lese — alle private nøkler som er tilgjengelig for JCE på maskinen.)
- Ved å kalle metoder definert i eksterne (egenutviklede) DLL-er gjennom JNI. Disse DLL-ene går utenom Java-maskinens sikkerhetsmekanismer, og har dermed de samme potensielle sikkerhetsproblemene som plugins eller Active-X-kontroller. De bør altså installeres lokalt, ikke over nettet. Adgang til å laste og bruke en bestemt DLL med JNI-metoder tildeles gjennom *RuntimePermission(loadLibrary.{libraryName})*-tillatelsen. (Hvis det ikke er kontroll med hvor dette biblioteket stammer fra, er dette i praksis det samme som å gi alle tillatelser.)

3.6. Netscape-plugin

En Netscape-plugin består av maskinkode (altså: en DLL eller delt bibliotek) og kan dermed nå alle lokale ressurser med de samme rettigheter som den brukeren som kjører Web-leseren. En plugin vil altså kunne generere signaturer m.v. Plugins kan installeres over nettet på en måte som er ganske enkel for brukeren, men dette er, som all annen programvareinstallasjon over nettet, **en stor sikkerhetsrisiko**. Dermed vil vi ikke anbefale at løsningen baseres på plugins, med mindre disse kan distribueres på en sikker måte. (Dvs. installeres lokalt fra diskett eller CD-ROM, evt. i signert form.) I såfall er løsningen sikkerhetsmessig tilsvarende Active-X (se nedenfor).

Siden plugin består av maskinkode, må de finnes i separate versjoner for hver enkelt plattform som Netscape kjører på. Enkelte plugins finnes bare for noen av plattformene.

3.7. Active-X-kontroller

En Active-X-kontroll er et Microsoft COM-objekt som kan integreres i en Web-side. (Dessuten i andre applikasjoner som opptrer som "Active-X containers", bl.a. det meste av Office-suiten.) Active-X er en Microsoft-spesifikk mekanisme, og støttes heller ikke av Netscape på Windows. Active-X-kontroller består av maskinkode, i form av en DLL, og har alle muligheter og rettigheter som ligger til den prosessen (f.eks. Internet Explorer)

som kontrollen kjører inne i. Dette er spesielt problematisk på énbruker-operativsystemene Windows 95 og Windows 98, hvor kontrollen dermed får adgang til absolutt alt på maskinen; men også for Windows NT fordi typiske NT-installasjoner er uforsiktige med å tildele rettigheter som normalt bare burde være tilgjengelige for administrator.

Active-X-kontroller kan være enten lokalt installerte eller lastes ned automatisk over nettet. På grunn av sikkerhetsrisikoen ved nedlastet kode, er nedlastede Active-X-kontroller i praksis alltid signerte. (Brukeren har mulighet til å overstyre dette og akseptere ikke-signerte kontroller, selv om de ikke stammer fra lokal maskin.) Men det er svært få muligheter til å gradere tillit, ettersom kontrollen selv består av maskinkode.

Bruk av kontrollen styres av to definerte sikkerhetsegenskaper ved hver enkelt kontroll:

- "Safe for untrusted data", dvs. kan kontrollen gjøre noe farlig dersom den startes opp på ukjent og ikke-tiltrodd data. En kontroll som presenterer et rasterbilde er typisk "safe", mens f.eks. en som presenterer PostScript-kode eller et Word-dokument (med makroer) ikke er det.

- "Safe for scripting", dvs. kan kontrollen på noen måte provoseres til å gjøre noe farlig dersom den styres (mottar metodekall) fra ikke-tiltrodde script.

Active-X-kontroller kan utføre alle operasjoner som er aktuelle for vår anvendelse, og signerte eller lokalt installerte Active-X-kontroller kan være en akseptabel løsning for signaturgenereringen. Dette krever imidlertid at brukeren konfigurerer Internet Explorer til å akseptere kontroller. Selv om standardinnstillingene åpner for det, vil vi nok anbefale IE-brukere at de **ikke** aksepterer Active-X-kontroller fra ukjente kilder, og ikke tillater scripting av Active-X-kontroller fra HTML-sider med ukjent kilde. **Spesielt** gjelder dette når maskinen skal brukes til signaturgenerering. Dermed må brukeren legge SLN-mottakstjeneren som "trusted site" for å kunne motta og bruke Active-X-kontroller derfra.

4. Forkastede løsningsalternativer

Vi vil først gå gjennom noen løsningsmønstre som kan synes plausible i utgangspunktet, men som av forskjellige grunner må forkastes. (Først og fremst fordi de ikke gir reell ikke-benektbarhet, dvs. signaturfunksjonen oppfyller ikke kravene til en kvalifisert signatur.) Likefullt er det instruktivt å se på hvorfor de ikke fungerer.

4.1. Tradisjonell Web-løsning, sikret med SSL

Dette er antakelig den enkleste løsningen å realisere, og i hvert fall den som brukere, utviklere, driftspersonell og tjenesteleverandører er mest kjent med. Ligningsskjemaene realiseres som HTML-skjemaer, og mottas av CGI-programmer eller andre slags transaksjonsprogrammer, avhengig av hva som er tilgjengelig og praktisk på den valgte implementasjonsplattformen. Sikkerheten ivaretas ved å bruke SSL på sesjonen mellom Web-leseren og Web-tjeneren.

Denne løsningen har flere alvorlige ulemper, og er derfor uaktuell:

- SSL gir ikke noen form for ikke-benekting. (Det kan hende det ville være mulig å logge alle data som utveksles over sesjonen, og at dette kunne gi tilstrekkelig sporbarhet i tilfelle tvister, men det er høyst tvilsomt.)

- Det er helt uaktuelt å installere kvalifiserte sertifikater som klientautentiseringssertifikater for SSL. Dette vil være i strid med

sertifiseringspolicy. Brukeren vil altså misligholde avtalen med sertifiseringsautoriteten og utsette seg for sanksjoner (ved siden av risikoen for misbruk av nøklene).

- Formatet for HTML-skjema er ikke spesielt velegnet for å bli signert over. I det minste kreves en del omtanke i design og spesifikasjon av detaljene i skjemainnholdet for at betydningen skal være veldefinert og entydig, og brukerne må også være innforstått med dette.

- Hvis Web-tjeneren selv opereres av Skatteetaten, som er part i saken, er det uheldig at presentasjonen av skjemaet styres av HTML-koden som kommer fra Web-tjeneren.

Dersom en tredjepart skal kunne dømme i en tvist, må HTML-koden som definerer (presentasjonen av) skjemaet kunne knyttes til resultatet av utfyllingen slik at det ikke er tvil om hva brukeren så på skjermen. Dette er ikke praktisk gjennomførbart med standard verktøy. Stilark vil forverre problemet, ettersom presentasjonen av skjemaet både avhenger av stilark og oppsett lokalt hos brukeren, og av stilark som følger skjemaet fra Web-tjeneren. I prinsippet må alt dette logges, ellers kan en ikke utelukke at f.eks. at tekst på skjemaet ble usynlig pga. uheldige kombinasjoner av farger, størrelser på skjemaelementer o.l.

Det gjør ikke noen vesentlig forskjell om andre innholdsformater enn HTML-skjema blir brukt.

4.2. Tradisjonell Web-løsning med uavhengig tredjepart

Denne løsningen er den samme som den ovenfor, bortsett fra at Web-tjeneren kjører hos en uavhengig tjenesteleverandør. Hovedproblemet består, at SSL ikke gir kvalifisert signatur.

Denne løsningen gir imidlertid en viss grad av ikke-benekting likevel, på grunn av at den går gjennom en uavhengig tredjepart som kan loggføre all trafikk. Spesielt vil tredjeparten være i stand til å koble skjemadefinisjonen (dvs. eksakt den HTML-koden som opprinnelig ble sendt til klienten) med det utfylte skjemaet som klienten returnerte.

4.3. Signaturgenerering i klient

Selve signaturgenereringen **må** utføres i klienten, ute hos underskriveren. (Hvis ikke ville det vært som om den oppgavepliktige skulle gitt ligningsfunksjonæren blankofullmakt til å signere ligningsoppgaven.) En nøkkel som kan generere en ikke-benektbar/kvalifisert signatur må eieren aldri slippe utenfor sin kontroll, hvis ikke må den umiddelbart betraktes som kompromittert. Siden løsningen for SLN tar sikte på å kunne signere dokumenter som allerede befinner seg på Web-tjeneren, kunne vi tenke oss en variant av signaturgenereringen der hash-koden ble generert på Web-tjeneren, overført til klienten, signert der og så returnert til Web-tjeneren.

Denne løsningen må vi også forkaste:

- Brukeren har ingen garanti for at det som Web-tjeneren presenterer som dokumentet som skal signeres samsvarer med hash-koden som oversendes. (Web-klienten ser bare den representasjonen av dokumentet som Web-tjeneren velger å generere, f.eks. HTML-siden som er resultatet av å presentere et XML-dokument via XSL og kan ikke selv kontrollere hash-koden.)

- Hash-koden er bare beskyttet av SSL-autentiseringsnøkkelen/sertifikatet under overføring, og som nevnt ovenfor er dette vesentlig svakere enn et kvalifisert sertifikat.

(Denne løsningen kan vi sammenligne med å få lest opp kontrakten av motparten over telefon, og deretter få oversendt den siste siden for å undertegne…)

Løsningen ville være noe mer spiselig dersom tjeneren ble operert av en uavhengig tredjepart, men uansett vil løsningen langt fra oppfylle kravene til kvalifisert signatur.

5. Løsningsalternativer

5.1. Krav til akseptable løsninger

Følgende sikkerhetsrelaterte krav må oppfylles:

1. Signaturen må oppfylle kravene til kvalifisert signatur, og sertifikatet må dermed tilfredsstille kravene til kvalifisert sertifikat. (I så fall vil sertifiseringspolicy antakelig stille krav som er ekvivalente til kravene nedenfor.)

2. Signeringsnøkkel må være under eierens kontroll til en hver tid. Signeringen må foregå inne i et smartkort som besittes av eieren (underskriveren), og hvor PIN-kode eller lignende eksplisitt må gis inn for hver signatur som genereres.

3. Originaldokumentet som skal signeres, ikke bare en transformert presentasjonsform av det, må være til stede på klientmaskinen der signeringen foregår. Underskriveren må være i stand til å lese dokumentet på skjerm, eller på annen måte forsikre seg om at dokumentet som blir signert er det rette.

4. Programvare som brukes for presentasjon av dokumentet og generering av signatur opererer på vegne av underskriveren, og må stamme fra en uavhengig tredjepart. (Det er ikke akseptabelt at den kontrolleres av den parten som krever signatur, i dette tilfellet Skatteetaten. F.eks. vil **ikke** plugins, applets eller annen programvare nedlastet fra Skatteetatens tjenere kunne brukes.)

5. Nedlasting og installasjon av ekstra programvare (plugins etc.) må ikke skje på en måte som gir risiko for kompromittering av klientmaskinen (virus, trojanske hester).

Punkt 3 betyr at for å få generert en signatur på et dokument, må originaldokumentet lastes ned til klienten. Formatet på originaldokumentet må være et som kan presenteres for underskriveren på klienten. Det betyr at det enten må være i HTML, XML, PDF eller andre formater som en kan anta at underskriveren har eller kan skaffe presentasjonsmoduler for i sin Web-leser, eller at det må utvikles og legges til rette for at underskriveren kan få skaffet applets, plugins eller lignende fra en uavhengig tredjepart for å håndtere dokumentformatet.

Det er ikke nødvendig å returnere hele det signerte dokumentet fra klienten, såfremt dokumentet ikke er endret. Det er tilstrekkelig å returnere selve signaturen, som kan kombineres eller assosieres med resten av dokumentet på tjeneren.

Web-leserne inneholder ikke noen direkte støtte for å generere signaturer på Web-sider eller Web-skjema. (En praktisk funksjon, som altså ikke finnes, ville være HTTP "file upload" eller "form submit" med automatisk S/MIME-signatur.) Funksjonene for S/MIME-signering som finnes både i Netscape Messenger og Microsoft Outlook gjelder epost-meldinger. Dette innebærer ett av to:

- Hvis ned- og opplasting av skjema/signatur skal skje interaktivt, ved HTTP, må signaturgenereringen utføres i en eller annen form for ekstern programvare, som applets, plugins etc.

- Alternativt overføres meldinger til epost-klienten for signering, og signerte meldinger returneres pr. epost (i så fall antakelig tilbake til EDI-mottaket).

5.2. Valg av protokoll/sikkerhetssyntaks

S/MIME (PKCS#7 hhv. CMS innkapslet i MIME) er den sikkerhetssyntaksen som allerede er valgt i SLN. Denne (dvs. S/MIME versjon 3) blir Internett-standard, og brukes i andre standarder/profiler som EDIINT. Dessuten er implementasjoner av S/MIME integrert i mange utbredte produkter. Det er derfor liten grunn til å velge noe annet for Web-løsningen. M.a.o., den MIME-innholdsdelen som inneholder det utfylte skjemaet sikres ved hjelp av S/MIME. (Det kan være aktuelt å følge EDIINT-rekommendasjonen for EDI over HTTP. Denne definerer bl.a. bruk av kvitteringsmeldinger, som kan være praktisk, men dette er ikke strengt nødvendig.)

Protokollen S-HTTP sikrer utvekslingen på HTTP-nivå med kryptering og/eller digital signatur. (Denne protokollen er også basert på CMS-formatet.) Denne typen protokoll er ikke spesielt velegnet for vårt formål siden den signerer over HTTP-transaksjonen, ikke bare selve skjemaet. Det fantes en Netscape-plugin for denne protokollen, men såvidt vi kan se er denne ikke lenger tilgjengelig, og vi kjenner ikke til andre implementasjoner.

SSL/TLS og andre mekanismer som gir sikkerhet på transportlag eller lavere er ikke egnet, siden de ikke gir god nok støtte for ikke-benektbarhet.

5.3. Løsningsmønstre

Nedenfor har vi gjennomgått noen typer av løsninger som kan være aktuelle. Merk at vi ikke har gjennomført noen fullstendig evaluering av de produktene som er nevnt, og det at det er nevnt nedenfor betyr ikke at vi går god for sikkerhet og brukbarhet for hvert enkelt produkt. Produktene er tatt med rett og slett som eksempler på hva som finnes. Vi regner heller ikke med at vi har fått med **alle** aktuelle produkter.

5.3.1. Signaturgenerering knyttet til submit-aksjon

HTML-scripting kan brukes til å aktivere et script umiddelbart før "submit" av skjema (`onsubmit`). Dette scriptet kan i prinsippet lese innholdet i skjemaet og generere en signerbar representasjon av dette. (Det kan også i og for seg generere en EDIFACT-melding eller tilsvarende standardisert form av skjemaet.)

Det er en par poeng med denne løsningen som ikke er helt opplagte:

- Det lar seg ikke gjøre å sende det signerte skjemaet i script-funksjonen selv, så signaturen må legges i et skjult felt, som så implisitt blir sendt av den innebygde submit-funksjonen i Web-leseren. (`onsubmit`-aksjonen er grei fordi den kjøres før submit-funksjonen samler innholdet i skjemaet og utfører HTTP POST/GET.) Signaturgenereringen kunne også gjøres i en annen script-aksjon, f.eks. `onclick` på en egen "Signer"-knapp. I dette tilfellet ville brukeren måtte klikke først "Signer" og så "Send".

- For å unngå å sende dobbelt opp (både skjemafeltene selv + den konverterte og signerte versjonen) kan signaturfeltene og submit-knappen legges på et **annet** skjema i samme HTML-side. (Det skjemaet som inneholder de opprinnelige feltene har da ikke noen egen submit-knapp.)

Denne løsningen har flere problemer:

- Mulighetene for signaturgenerering fra script ble diskutert ovenfor, men er generelt svært begrenset.

- Den representasjonen som signeres blir generert av scriptet, og for å få rimelig sikkerhet må brukeren finne seg i å kontrollere at den er korrekt før signeringen faktisk blir utført.

5.3.2. Signaturgenerering av proxy

Denne løsningen går ut på at Web-klienten ikke sender skjemaet direkte til mottakstjeneren, men via en lokal HTTP-tjener (proxy) som tilføyer signaturen og sender den videre. Proxies er en vanlig teknikk ved brannvegger, og Web-klientene kan settes opp til å sende all HTTP-trafikk via en proxy (for sikkerhetskontroll av innholdet). Dersom proxyen kjører på en separat maskin, som er det vanlige i brannveggløsninger, er dette ikke noen hensiktsmessig løsning.

Det finnes imidlertid noen produkter (bl.a. Algorithmic Research PrivateWire, http://www.arx.com) med signering av Web-skjema, som faller i denne kategorien. Ut fra produktbeskrivelsene ser dette ser egentlig ut til å være VPN/brannvegg/transportlagssikkerhetsløsninger som installerer seg i TCP-stakken (lokalt på samme maskin) og fanger opp og "filtrerer" HTTP POST-operasjoner, hvor de bl.a. kan føye til signatur på innholdet. Disse filtrene på transportlaget har i prinsippet samme funksjon som proxies, selv om de ikke er realisert som separate tjenerprosesser på en annen maskin.

En slik løsning har to uheldige sider i forhold til vår anvendelse:

- Det er uheldig å skille dokumentpresentasjonen (i Web-leseren) og signaturgenereringen (i et eget produkt), når de to delene henger sammen på en såpass løs måte. (Det blir for lett å lure brukeren til å signere noe annet enn det som vises i Web-leseren.)

- Det er en ganske vidløftig og gjennomgripende ting for sluttbrukere å installere denne typen produkter — spesielt når det er for én eneste anvendelses skyld.

- Det er neppe mulig å installere mer enn ett produkt av denne typen på en enkelt maskin. Brukeren får med andre ord et problem hvis han/hun skal samarbeide med flere parter som krever signerte skjema og baserer seg på forskjellige produkter.

5.3.3. Signaturgenerering i epost-brukeragent

Denne typen løsning innebærer at Web-løsningen bare brukes for å navigere i skjemadatabasen hos SKD. SSL-klientautentisering er tilstrekkelig for å implementere aksesskontroll. Dokumenter som skal signeres lastes ned, og signeres med S/MIME i epost-klienten. Poenget med denne løsningen er at S/MIME-støtte i epost-klienter er nokså utbredt.

Denne løsningen har flere varianter:

- Mest direkte ville være å sette submit-aksjonen til en `mailto`-URI, slik at skjemaet blir sendt via epost i stedet for HTTP. Dette strander imidlertid på at Web-leserne oppfører seg forskjellig. (Alt annet enn HTTP som protokoll for submit-aksjonen er eksplisitt udefinert i HTTP 4.0-spesifikasjonen.) Ingen av dem vi har prøvd har akkurat den ønskede oppførselen. Communicator sender innholdet i skjemaet som innholdet i meldingen (som ønsket), men uten å åpne epost-klienten, så brukeren får ikke noen sjanse til å signere. Internet Explorer åpner epost-klienten, så meldingen kan signeres, men får ikke med innholdet i skjemaet.

- Epost-klienten aktiveres fra script. Dette krever plattformspesifikke utvidelser og/eller tiltrodde script. Problemene er for det første at det å aktivere en epost-klienten i seg selv er plattform- og klient-avhengig, og for det andre bør scriptet aktivere den epost-klienten brukeren prefererer (ikke bare en bestemt klient på hver plattform). På Windows finnes muligens Active-X-kontroller som kan brukes fra script og gir adgang til epost via MAPI.

- Brukerens epostadresse legges inn i skjemaet (eller gjøres tilgjengelig for tjeneren på en annen måte). Transaksjonsprogrammet sender skjemaet pr. epost til brukeren, og brukeren må signere og sende tilbake. (Evt. plukke ut vedlegget som er det faktiske skjemaet og kjøre dette gjennom en frittstående signeringsapplikasjon.) Denne løsningen krever mest manuelle operasjoner, men krever ikke noe annet enn at epost-klienten har signeringsmuligheter, evt. håndterer binære vedlegg på en rimelig måte.

5.3.4. Signaturgenerering i hjelpeprogram

Skjemapresentasjon og signering skjer inne i et lokalt hjelpeprogram, og er egentlig temmelig uavhengig av Web-leseren. Hjelpeprogrammet kan realiseres som signert Java-applet, Active-X-kontroll, Netscape-plugin eller lignende, som diskutert i kapittel 3.

En løsning for SLN kan utvikles fra bunnen av, men det finnes også en del ferdige løsninger for elektroniske skjema som det er mulig å starte fra, bl.a.:

- UWI InternetForms (http://www.uwi.com) Netscape-plugin og Active-X-kontroll, bruker CryptoAPI på Windows for digital signering. (Kan laste ned demo-eksemplar, som også tillater digital signering.)

- JetForm (http://www.jetform.com) Netscape-plugin og Active-X-kontroll, signering v.hj.a. kryptoløsninger fra Entrust og RSA.

- Informed (http://www.shana.com) Signering v.hj.a. kryptoløsning fra Entrust.

- Adobe Acrobat (http://www.adobe.com) har også støtte for Web-integrerte skjema med signatur (også her kryptoløsning fra Entrust).

Merk: Det kan være eksportrestriksjoner på de versjonene av produktene som inkluderer digital signatur.

Grunnleggende i denne løsningen er at skjemadefinisjon og utfylt skjema er nye innholdstyper (MIME-typer), slik at Web-leseren aktiverer hjelpeprogrammet når Web-tjeneren leverer et skjema. I praksis må det defineres proprietære innholdstyper, selv om mange av løsningene vil være basert på standarder som XML. Produktene ovenfor har hver sine MIME-typer og er ikke uten videre interoperable.

Merk også at måten signaturen er representert (sikkerhetssyntaksen) varierer. Ingen av de produktene som er nevnt ovenfor benytter såvidt vi kan se S/MIME som signaturformat, men fletter i stedet signaturen inn i selve skjemaformatet på en format-avhengig måte. Det betyr at for hvert slikt format som aksepteres, må signaturverifisering skreddersys på tjenersiden.

Hjelpeprogrammet som installeres i Web-leseren presenterer skjema og genererer signatur. Dette programmet er i noen tilfeller (f.eks. Acrobat Reader) gratis tilgjengelig. For andre av produktene over må Skatteetaten antakelig kjøpe rettigheter til å distribuere kopier av hjelpeprogrammet. (Detaljer om prising og lisensbetingelser er stort sett ikke tilgjengelig gjennom Web-sidene, og må antakelig forhandles med leverandøren for hvert enkelt tilfelle.) Verktøyene som brukes for å utforme skjema(maler) og annen støtteprogramvare på tjenersiden er stort sett proprietære (og antakelig relativt kostbare.)

De ferdige løsningene over styrer altså i stor grad skjemaformat og sikkerhetssyntaks. For å få frihet til å velge dette selv, er alternativet et egenutviklet hjelpeprogram, f.eks. i Java. I såfall kan programmet lastes ned som signert applet.

6. Adgangskontroll

6.1. Brukerautentisering

Adgangskontroll til dokumentarkivet baseres på en eller annen form for brukerautentisering. Brukerautentisering betyr i utgangspunktet bare at transaksjonsprogrammene (CGI-programmene) kjenner brukeridentiteten for den som sendte HTTP-forespørselen (GET, POST). Det finnes forskjellige mekanismer for dette:

- Basis HTTP-autentisering med brukernavn/passord (som sendes over som en del av HTTP-protokollen).
- Andre HTTP-autentiseringsmekanismer (mindre brukt, ikke utbredt støtte).
- Autentisering med brukernavn/passord som sendes over som skjemafelter.
- "Cookies" brukes ofte for å referere til en sesjon. Autentisering (innlogging) skjer bare i første HTTP-forespørsel i sesjonen, og dette setter en cookie i klienten. De etterfølgende forespørslene knyttes til den første innloggingen ved å sende cookien.
- En av de ovennevnte mekanismene, men med SSL-kryptering for å gjøre det vanskeligere å plukke opp passord, cookies etc.
- SSL-klientautentisering.

Disse mekanismene gir varierende grad av sikkerhet. Mekanismer som baserer seg på at brukernavn, passord, sesjonsidentiteter osv. sendes over en ukryptert forbindelse er neppe sikre nok. Brukes en kryptert forbindelse bedres sikkerheten. Kryptografisk autentisering (challenge/response) er bedre, og SSL-klientautentisering er den mest tilgjengelige varianten. Dette krever imidlertid en infrastruktur for nøkkeladministrasjon (PKI, Public Key Infrastructure), altså sertifiseringsautoriteter. Siden det allerede er nødvendig å etablere en PKI for signeringssertifikatet, er det nærliggende å bruke SSL-klientautentisering.

Brukeridentitet kan ta forskjellige former. Mekanismer med brukernavn/passord baseres gjerne på brukerdatabaser som helt og holdent kontrolleres av den samme organisasjonen som driver Web-tjeneren. Brukes kryptografiske mekanismer, vil brukeridentiteten knyttes til et sertifikat, og typisk vil det være subjektets (eierens) navn eller ett av de alternative navnene som tjener som brukeridentitet.

Når transaksjonsprogrammet har fått tak i brukeridentiteten, må det slå opp i en brukerdatabase og finne ut hvilke rettigheter den gir, evt. kontrollere aksesskontroll-lister på de enkelte objektene (lagrede skjemaer osv.)

For SLN kan det være praktisk å bruke epostadressen i sertifikatet (dvs. alternativt navn: RFC822-adresse) som brukeridentitet, spesielt dersom skjemaene selv utveksles pr. epost i stedet for via HTTP.

6.2. Adgangskontroll-regler

Reglene for adgangskontroll kan gjøres ganske enkle. Vi foreslår følgende:

- Som brukeridentitet brukes enten epost-adresse eller subjekt-navn fra sertifikatene.

- Registrering av skjema krever ingen spesielle rettigheter.

- Skjema uten signatur kan evt. tillates registrert, men da med eventuelle andre autentiseringsdata, som f.eks. klientsertifikat.

- Hvert registrert skjema har en aksesskontroll-liste assosiert med seg. Alle brukeridentiteter som er registrert i denne listen har adgang til skjemaet.

- Etter registrering av skjema inneholder aksesskontroll-listen i utgangspunktet identiteten til den som har signert skjemaet (hvis signert) og/eller den som utførte registreringen (fra SSL-klientsertifikatet eller avsenders epostadresse).

- Den som registrerer skjemaet kan også føye til andre brukeridentiteter som skal ha adgang, f.eks. revisors identitet. (Dette kan skje idet skjemaet registreres, eller senere.)

- Evt. kan brukerrettigheter implisitt tilføyes fra andre databaser hos Skatteetaten.

Skjemadatabasen må tillate registrering av flere skjema under samme foretaksnummer og skjematype, men bør gi advarsel dersom det er flere registreringer. Hvis flere forsøker å legge inn "samme" skjema flere ganger er det umulig å vite hvilken registrering som er den gyldige. (Ugyldig skjema avvises senere i kjeden ved at det ikke er signert av rette vedkommende. Vi antar imidlertid at Web-tjener-mottaket ikke skal behøve å ha fullstendig og oppdatert informasjon om hvem som skal kunne signere for hvert enkelt foretak, se nedenfor.) Merk at den ene som legger inn et skjema ikke dermed får noen rettigheter i forhold til eventuelle andre skjema under samme foretaksnummer.

Alternativt må det på forhånd registreres hvem som skal sende inn skjema for hvert enkelt foretak, dvs. en fullstendig adgangskontrolldatabase må settes opp før registreringen starter. Vi tror det er verdt å forsøke å unngå dette. Reglene som er skissert ovenfor bør gjøre det mulig å desentralisere administrasjonen av adgangsrettigheter.

Slik reglene er formulert ovenfor har en bruker enten ingen eller alle rettigheter til et skjema. Dette kan evt. graderes, ved å innføre flere ulike rettigheter som kan tildeles uavhengig, f.eks.:

- Leseadgang, gir mulighet til å se innholdet i skjemaet.

- Adgang til å signere, dvs. tilføye signatur.

- Eierskap, gir mulighet til å slette eller overskrive, og til å tildele rettigheter til andre brukere.

Den som registrerer skjemaet opprinnelig får alle rettigheter.

Uautentiserte oppslag via HTTP gir selvfølgelig ingen direkte rettigheter, men hvis en epostadresse er gitt leseadgang, kan en be om at skjemaet sendes pr. epost til denne adressen.

Håndtering av multiple signaturer kan skje på to forskjellige måter:

- Den som signerer på nytt må eksplisitt ha rettigheten til det, og effekten av å tilføye en signatur er i prinsippet å erstatte dokumentet med et nytt.

- Den som signerer på nytt legger inn et nytt dokument, og må altså bare ha leseadgang til det opprinnelige. (Det nye dokumentet består evt. bare av selve signaturen + en referanse til det opprinnelige dokumentet, tilsvarende bruken av multipart/signed i S/MIME.)

Dersom en har brukerautentisering med ulike mekanismer, må en ta hensyn til at disse har ulik styrke. Hvis skjemaet er registrert via en SSL-forbindelse med klientautentisering, er det bare den som kan presentere et gyldig sertifikat med samme brukeridentitet og tilsvarende nøkkellengde for autentiseringsnøkkelen som har adgang. Hvis et skjema er registrert pr. epost og uten signatur, vil epost-avsenderadressen være registrert som "eier", og det er rimelig å stille svakere krav.

6.3. Klientautentisering i SSL/TLS

SSL og TLS med klientautentisering støttes av de nyere versjonene av de aktuelle Web-leserne, inkludert de versjonene som gir "sterk" kryptografi. For å kunne bruke klientautentisering, må brukeren bruke en sertifiseringsautoritet som utsteder slike sertifikater. Brukeren må selv laste inn sitt autentiseringssertifikat i Web-leseren, og indikere at dette sertifikatet skal brukes for klientautentisering.

Klientautentisering aktiveres for en enkelt SSL-sesjon dersom SSL-tjeneren krever det. Brukeren må gjøre følgende:

- Velge hvilket sertifikat som skal sendes til tjeneren. (Hvis brukeren har flere sertifikater, og ikke ett av dem allerede er valgt som standard autentiseringssertifikat.) Dette bestemmer hvilken identitet tjeneren kjenner denne brukeren under.

- Taste PIN-kode for å aktivere den private nøkkelen som hører til sertifikatet. (Både for Netscape og Internet Explorer gjelder: Det er mulig å velge om PIN-koden skal tastes hver gang nøkkelen brukes, eller bare første gang etter at Web-leseren er startet opp. For den innebygde nøkkeldatabasen er PIN-koden er knyttet til nøkkeldatabasen, ikke til hver enkelt nøkkel.)

SSL-tjeneren er konfigurert med hvilke sertifiseringsautoriteter den aksepterer for klientautentisering, og setter først opp sesjonen dersom klientsertifikatet den mottar er gyldig og autentiseringen ellers er korrekt. Når SSL-sesjonen er etablert, brukes den normalt for flere påfølgende HTTP-oppslag fra samme klient (IP-adresse), slik at det ikke er nødvendig å gjenta autentiseringen[7].

Når Web-tjeneren kommer så langt at HTTP-oppslaget behandles, f.eks. CGI-programmet kjører, er altså autentiseringen fullført og godkjent. Klientsertifikatet, eller i det minste et utvalg av informasjonen fra det, er tilgjengelig. Nøyaktig hvordan SSL-tjeneren konfigureres og hvordan autentiseringsinformasjonen er tilgjengelig varierer fra produkt til produkt.

Web-tjeneren Apache kan f.eks. kombineres med SSL-implementasjonen OpenSSL. Her blir klientsertifikat, subjektnavn (DN), utsteders sertifikat, utsteders navn og andre parametre gjort tilgjengelig for CGI-program gjennom CGI-variable på vanlig måte. (Dvs. som environment-variable, på samme måte som "&"-parametre gitt gjennom URL-strengen.) Dessuten kan klientautentisering konfigureres slik at den integreres med basis HTTP-autentisering. Klientens navn fra sertifikatet kontrolleres mot den samme brukerdatabasen, og aksesskontroll og CGI-programmer utføres på samme måte som om brukeren hadde brukt basis-autentisering og oppgitt brukernavn og passord. (Modifikasjonene til Apache for å støtte SSL, samt pekere til rette versjoner av Apache og OpenSSL finnes på https://www.apache-ssl.org)

[7] Dette forutsetter at klienten beholder sesjonsnøklene som ble generert og utvekslet da sesjonen ble satt opp, så det er altså ikke noen risiko for at sesjonen skal bli "overtatt" av en annen klient.

Merk at klientautentisering i prinsippet er autentisering av klientmaskinen, ikke nødvendigvis brukeren som sitter ved den. Men slik SSL-støtten i Web-klientene er utformet, fungerer klientautentisering i praksis også som brukerautentisering: Nøkkeldatabasen er beskyttet med et personlig passord/PIN-kode, og er evt. fysisk plassert på et personlig smartkort. Kun én bruker har adgang til maskinen om gangen. Klienten bør nok konfigureres slik at den ber om passord for nøkkeldatabasen for hver autentisering, ikke bare første gang.

SSLs persistente sesjoner kan være et lite problem på maskiner som flere brukere har adgang til, siden det ikke finnes noen eksplisitt mekanisme å avslutte sesjoner (slette sesjonsnøklene). Det er et spørsmål om sesjonsnøklene kan bli liggende igjen i minne noe sted slik at de er tilgjengelig for andre programmer på samme maskin, muligens også etter at klientprogrammet som åpnet sesjonen har blitt avsluttet.

6.4. Brannvegg

Maskinen med skjemadatabasen og Web-tjeneren må plasseres utenfor Skatteetatens sentrale systemer. Her vil vi bare diskutere hvordan skjemadatabasen og Web-tjeneren selv beskyttes. For overføring av mottatte skjema videre til interne systemer kan samme løsning som for EDI-mottaket brukes, og Web-løsningen vil neppe kreve noen endringer på dette punktet.

Web-tjenesten for SLN har flere logiske funksjoner:

- Distribusjon av tomme skjema, informasjonsmateriale m.m.
- Distribusjon av programvare for nedlasting (applets, script, plugins, Active-X-kontroller osv.)
- Mottak av utfylte skjema, oppslag og transaksjoner mot skjemadatabase.

Disse funksjonene kan skilles mellom flere maskiner som beskyttes på forskjellige måter. De to første kan plasseres et helt annet sted enn mottak og skjemadatabasen, for å skille det administrative ansvaret for disse funksjonene og redusere behovet for administrativ aksess til maskinen med skjemadatabasen mest mulig.

Den første funksjonen er en ordinær, relativt ukritisk Web-tjeneste, som stort sett leverer ut statiske dokumenter, ikke har noen sikkerhetskritiske transaksjonsprogrammer og ikke trenger ekstern adgang til andre tjenester enn HTTP/HTTPS. Den eneste aktuelle sikkerhetstrusselen er uautorisert endring eller overskriving av dokumenter på tjeneren. Det bør være enkelt å unngå at angrep på maskinen som kjører denne tjenesten kan tjene som springbrett til andre systemer.

For programvaredistribusjon bør det være mulig å bruke SSL/TLS, evt. er programvaren som distribueres signert. Uten signert kode bør denne funksjonen i alle tilfeller håndteres av en uavhengig tredjepart, som diskutert i kapittel 1.3.3. Utover dette er denne funksjonen også en temmelig uproblematisk Web-tjeneste, som den forrige, men det er spesielt viktig å sikre programvaren på denne tjeneren mot uautoriserte endringer.

Det mest sikkerhetskritiske i løsningen er transaksjonsprogrammene (CGI-programmene) i skjemadatabasen. Disse regulerer adgangen til skjemaene, og er altså ansvarlig for å implementere aksesskontrollreglene vi diskuterte ovenfor. Ved angrep på tjeneren som har denne funksjonen er det risiko for at skjemadatabasen blir kompromittert eller tilgjengelig utenfra. Velger SLN å bruke klientautentisering, kan aksess til denne maskinen begrenses til autentiserte Web-klienter: Brannveggen slipper bare HTTPS-trafikk gjennom, og HTTP-tjeneren konfigureres til bare å akseptere autentiserte klienter med gyldige sertifikater (dvs. der CA-sertifikatene er installert og merket som tiltrodde).

Felles for alle de tre funksjonene er at det kreves svært begrenset nettverkstilgang utenfra, i prinsippet bare HTTP/HTTPS, slik at brannvegg-konfigurasjonen kan være tilsvarende restriktiv.

7. Skjemaløsninger for XML

XML som sådan støtter ikke Web-skjema, og det finnes heller ikke noen standard (eller opplagte kanditater til standard) for skjemabeskrivelsesspråk. Inntil videre må derfor hver enkelt anvendelse definere sin egen DTD, og selv utvikle hjelpeprogrammer for å håndtere disse. En mulig måte å gjøre dette på, er ved å konvertere en skjemabeskrivelse i XML til HTML-skjema ved hjelp av XSL. (Det utfylte skjemaet blir i tilfelle returnert med den vanlige mekanismen for form-submit, altså ikke i XML, med mindre en bruker script for å lage en egendefinert submit-aksjon.)

Det foreligger imidlertid et forslag til standard for skjemabeskrivelsesspråk for XML, kalt XFDL (Extensible Forms Definition Language). XFDL er et XML-basert språk utviklet for å løse problemene knyttet til det å digitalt representere komplekse skjema/dokument ("forms") samt gi mulighet for signering av dokumenter.

XFDL-spesifikasjonen ble sendt inn til W3C[8] i september 1998, men har fremdeles ingen offisiell status, og det er ingen garantier for at XFDL faktisk vil bli en W3C-standard. Man kan altså ikke forvente at Web-lesere vil støtte XFDL med det første.

Dermed vil bruk av XFDL måtte foregå i et hjelpeprogram og ikke i selve Web-leseren. Det er så langt ikke mange leverandører som har produkter som støtter XFDL[9]. Vi har sett på et produkt levert av UWI Unisoft (http://www.uwi.com/), leverandøren som sendte spesifikasjonen til W3C. Demo-versjonen man kan laste ned fra nettet ser ut til å fungere som ønsket (tillater blant annet digital signering) samt at den har et meget greit og oversiktlig brukergrensesnitt[10]. Om man er villig til å binde seg til en leverandør, vil dette sannsynligvis være et godt alternativ.

Betalingsmodellen som UWI Unisoft benytter seg av, er at plug-ins kan distribueres gratis mens skjemaene blir lisensierte.

XFDL-spesifikasjonen: http://www.w3.org/TR/NOTE-XFDL

8. Smartkort- og kryptografiløsninger

Smartkort- og kryptografiløsninger, sertifiseringstjenester og applikasjonsprogramvare stammer gjerne fra forskjellige leverandører, og interoperabilitet mellom dem er ikke alltid gitt. Applikasjonsprogramvaren støtter typisk ett eller to ulike grensesnitt mot kryptografiske funksjoner, ofte ett proprietært og ett standardisert. Et annet produkt, som realiserer dette grensesnittet må velges, og dette må kunne håndtere alle sertifikater og nøkler som skal brukes. Formatene på sertifikater, smartkort som sertifiseringsautoriteten kan håndtere osv. setter ytterligere begrensninger her. Dessuten er det store variasjoner mellom produktene mhp. hvordan de tolker og implementerer standardene.

Interoperabilitet i smartkortløsninger krever tilpasning på to nivåer.

- Smartkortleser: Leserne tilkobles på ulike måter, og har ganske forskjellige kommandosett.

- De applikasjonsrettede tjenestene som smartkortet tilbyr, for eksempel kryptografitjenestene, må gjøres tilgjengelig for applikasjonsprogrammer på en standardisert måte. (Det finnes ikke

[8] W3C er organisasjonen som står for standardiseringer på Web. Se http://www.w3.org/ for mer informasjon.
[9] UWI Unisoft påstår at det finnes mange leverandører som støtter XFDL. Vi har ikke undersøkt hvilke produkter disse leverandørene har samt hvor godt produktene fungerer.
[10] Det digitale skjemaet ser ut som et helt vanlig papir-skjema, noe som vil lette overgangen fra papir-skjema til digital-skjema.

tilstrekkelig detaljerte standarder for hvordan disse tjenestene er realisert i smartkortene selv.)

I praksis betyr dette gjerne at det må installeres 2 sett med drivere. En driver for hver enkelt (type) smartkortleser, på samme måte som for andre ytre enheter knyttet til maskinen. Dessuten finnes ett eller flere sett med drivere for hver enkelt type smartkort (kanskje til og med for hver enkelt sertifiseringsautoritet som tilbyr sertifikater og nøkler på en bestemt korttype). I tillegg finnes gjerne en infrastruktur som registrerer hvilke kort som befinner seg i leserne, mottar forespørsler om spesifikke tjenester fra applikasjonene og på bakgrunn av dette aktiverer de rette driverne.

Det er spesielt to industristandarder som er aktuelle:

- PKCS#11, som gjør det mulig å "plugge inn" tredjeparts kryptoløsninger i Netscape (og nylig også med sterk krypto).

- PC/SC, som gir en generell smartkortstøtte til applikasjonsprogrammer på Windows, og CryptoAPI, som kan utstyres med drivere for å bruke kryptografiske funksjoner fra tredjeparts kryptoløsninger. Disse blir tilgjengelig for alle applikasjoner under Windows som bruker kryptografiske sikkerhetsfunksjoner, inkludert Internet Explorer.

Med andre ord, for begge de to mest aktuelle Web-leserne: Netscape og Internet Explorer, finnes det kurante smartkortløsninger. Ved at SLN/SKDs Web-sider henviser til aktuelle leverandører, burde det være greit gjennomførbart for avgiverne å anskaffe og installerere det som trengs.

En skal være oppmerksom på at det er store variasjonsmuligheter i hvordan disse standardene implementeres. Selv om applikasjonen forventer PKCS#11 (hhv. CryptoAPI) og krypto-modul tilbyr det samme grensesnittet, er de ikke nødvendigvis interoperable. Basis-funksjonene som hhv. Netscape og Internet Explorer bruker for signering og signaturverifikasjon støttes sannsynligvis av de fleste.

Vi har ikke gjort noe fullstendig produktsøk og –evaluering, men noen gjennomgående inntrykk er at:

- For praktisk talt alle kortlesere finnes PC/SC-drivere for Windows.

- De aller fleste leverandører av kryptografi-smartkort tilbyr CryptoAPI-støtte for disse. PKCS#11-støtte er også utbredt, men ikke i like stor grad som CryptoAPI.

- Kryptografiske applikasjoner, f.eks. tredjeparts epost/fil-krypto, sertifikatutstedelse og andre sertifiseringsautoritet-verktøy bruker først og fremst PKCS#11. Det er stort sett bare Microsofts egen programvare som bruker CryptoAPI.[11]

For andre Web-lesere er det ikke kjent hva slags smartkortstøtte som finnes. Opera Software og Telenor Conax har annonsert at Opera skal støtte Conax' smartkortbaserte PostSec-løsning. Det er ikke kjent om denne baserer seg på noen form for standardisert grensesnitt mellom Web-leser og smartkortløsningen. I så fall ville dette rent praktisk bety at Opera kan brukes mot andre smartkortløsninger også.

8.1. PKCS#11

Netscape støtter pluggbare kryptografimoduler gjennom grensesnittet PKCS#11, både på PC og UNIX-plattform. Nøkkelkarakteristikker ved PKCS#11:

[11] Det er antakelig flere grunner til dette: CryptoAPI støttes bare på Windows, mens mange av de andre applikasjonene også finnes UNIX-varianter. Kunder utenfor USA har tidligere ikke uten videre hatt sterk kryptografi tilgjengelig gjennom CryptoAPI. Dessuten har tredjepartsleverandører har antakelig ønsket å unngå å konkurrere direkte med Microsofts tilsvarende produkter.

- PKCS#11 er et API, som for Netscapes vedkommende er realisert gjennom et dynamisk lenket/delt bibliotek (DLL på Windows, .so på UNIX). API-et er formulert i C/C++, (men en Java-språkbinding er visstnok også under utarbeidelse.)

- Støtter bare kryptografiske tjenester, dvs. lagring av sertifikater og nøkler, (de)kryptering, hash-beregning og signering/verifikasjon, ikke andre smartkortbaserte tjenester.

For å støtte en ny kombinasjon av smartkort og leser på en gitt plattform, må leverandøren produsere sin egen PKCS#11-modul tilpasset denne kombinasjonen. (PKCS#11-modulen er ikke nødvendigvis monolittisk. Den kan godt være basert på standardiserte grensesnitt mot kortleser eller lignende, for den saks skyld PC/SC Resource Manager. Men Netscape forholder seg ikke til dette.)

Etter å ha koblet til leseren, installerer brukeren PKCS#11-modulen i Netscape. (Dette gjøres ved å fortelle Netscape filnavnet for PKCS#11-biblioteket gjennom "Create New Security Module"-dialogen. Evt. gjøres det helt automatisk av et installasjonsprogram: InstallShield eller lignende.)

PKCS#11 definerer et standardisert grensesnitt mot et bredt utvalg av kryptografiske funksjoner. En praktisk implementasjon av PKCS#11 vil bare tilby en delmengde av disse og noen kombinasjoner av parametre og variasjoner. (I noen tilfeller er det aktuelt å tilby nøyaktig de funksjonene som støttes i smartkortet. I andre tilfeller kan det være aktuelt å tilby smartkortets funksjoner og komplettere med funksjoner som er realisert i programvare internt i PKCS#11-modulen.)

Netscape er, for visse kombinasjoner av operasjoner, i stand til å kombinere funksjoner fra forskjellige moduler, men stiller noen krav om hvilke funksjoner som må finnes og hvordan disse er realisert for å kunne bruke en PKCS#11-modul til et bestemt formål (som f.eks. vår signaturanvendelse).

PKCS#11 er utviklet av RSA Data Security, og er uavhengig av Netscape. Andre leverandører som bruker PKCS#11:

- Baltimore Technologies (http://www.baltimore.com) tilbyr både kryptografimoduler med PKCS#11-grensesnitt (selv, eller i samarbeid med flere smartkort-leverandører), og applikasjoner som bygger på PKCS#11: MailSecure, som gir S/MIME-støtte i flere populære epost-klienter. FormSecure for digital signering av Web-skjema. Utviklerverktøy for utvikling av kryptografi- og PKI-løsninger, bl.a. X/Secure for digital signering i XML-dokumenter.

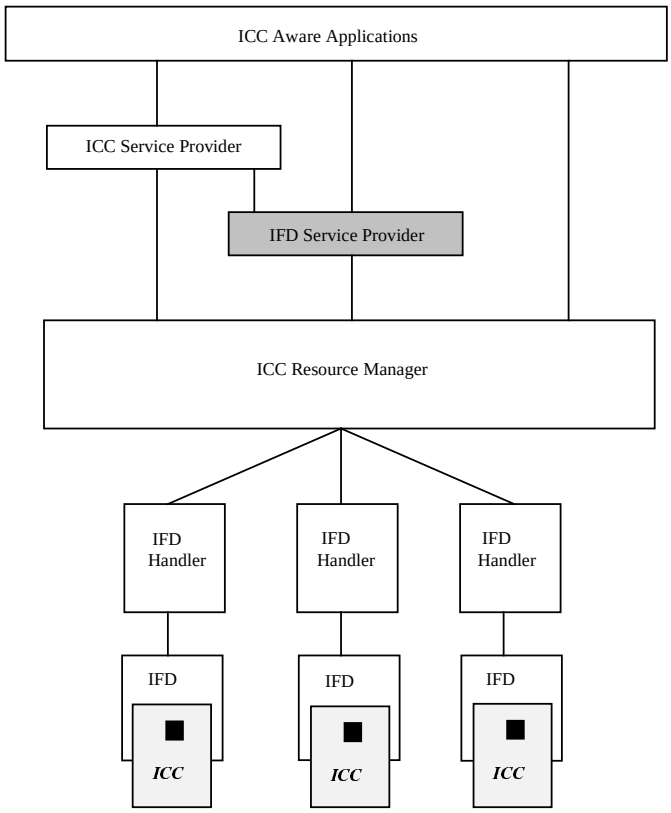
- Entrust (http://www.entrust.com) tilbyr også en omfattende serie med PKI-applikasjoner med PKCS#11 mot kryptomodulene: CA-verktøy, epost- og Web-sikkerhet, utviklingsverktøy. (Entrust leverer ikke selv kryptomoduler.)

- Algorithmic Research (http://www.arx.com) leverer smartkort, -lesere, utviklingsverktøy m.m. med PKCS#11-grensesnitt.

- Mange av leverandørene av smartkort leverer PKCS#11-drivere: ActivCard (http://www.activcard.com), GemPlus (http://www.gemplus.com/), Siemens (http://www.siemens.com/pc/solution/us/sec.htm), Schlumberger (http://www.1.slb.com/smartcards/),

The Open Group har spesifisert arkitekturen CDSA, med et API mot kryptografi-moduler som minner litt om PKCS#11 (men ikke er direkte kompatibelt). CDSA dekker sikkerhetstjenester i bredere forstand enn både PC/SC og PKCS#11, og ikke smartkort spesielt, eller andre smartkortanvendelser. Støtte for CDSA ser ikke ut til å være særlig utbredt, og i hvert fall ikke i vanlige sluttbrukerverktøy, så det er ikke noen grunn til å gå nærmere inn på det her.

8.2. PC/SC og CryptoAPI



Figur 1 Oversikt over PC/SC

PC/SC er en mer overordnet arkitektur enn PKCS#11, spesifikt rettet mot smartkort, men alle aktuelle smartkortanvendelser, ikke bare kryptografi. Microsofts CryptoAPI er basert på PC/SC, og er noenlunde sammenlignbart med PKCS#11[12]. Dette API-et brukes av alle krypto-anvendelser i Windows, f.eks. SSL-sesjoner i Internet Explorer, kodesignering og kryptering og signering av e-post i MS Outlook. Smartkort og lesere som kan integreres i PC/SC, hhv. CryptoAPI, kan dermed brukes fra et stort utvalg av applikasjoner.

PC/SC definerer flere sett med grensesnitt. En "smart card service provider" (SSP[13]) er en pluggbar programvaremodul som tilbyr en bestemt tjeneste (et bestemt grensesnitt) mot visse typer smartkort. En SSP for kryptotjenester kalles CSP ("crypto service provider"), og kryptografi-tjenestene i CryptoAPI baserer seg på slike CSP-er. Den som ønsker å gjøre krypto-tjenester i en bestemt type smartkort tilgjengelig for bruk gjennom CryptoAPI må altså lage en CSP for dette kortet. Alle SSP-er er COM-objekter, og installeres på samme måte som andre COM-klasser, i tillegg til at de må registreres i PC/SC slik at PC/SC gjenkjenner korttypen. (Dette gjøres av automatiske installasjonsprogrammer, og skulle være enkelt å installere for sluttbrukeren.)

Det lavere nivået i PC/SC kalles "resource manager" (RM). RM tilbyr forskjellige funksjoner slik at applikasjonsprogrammene (f.eks. via CryptoAPI) kan finne aktive

[12]CryptoAPI tilbyr noe mer høynivå operasjoner rettet mot spesifikke sikkerhetssyntakser og –protokoller, f.eks. SSL og PKCS#7, mens PKCS#11 først og fremst tilbyr de generelle kryptografiske primitivene.

[13] PC/SC-dokumentene bruker den noe mer nøytrale termen ICC (Integrated Circuit Card) for smartkort, og IFD (Interface Device) for smartkort-leser. Microsoft bruker "smartcard" og "smartcard reader". Bortsett fra figuren, som er hentet fra PC/SC-spesifikasjonen, bruker vi Microsofts terminologi her.

smartkort (eller la brukeren sette inn/velge kort), sette opp sesjoner mot og sende kommandoer til et valgt kort og laste og bruke en SSP som kan betjene kortet. RM styrer også reservasjon og deling av kort/kortleser mellom flere applikasjoner. Alt dette kan gjøres uavhengig av type kortleser og, til en viss grad, tekniske detaljer ved kortet. SSP-ene kommuniserer med kortet gjennom RM, slik at de ikke er avhengig av kortleseren.

RM tilpasses i sin tur til de fysiske kortleserne som finnes gjennom enhetsdriveren for kortleseren. Disse driverne installeres og håndteres på vanlig måte for enhetsdrivere under Windows. (Selve den fysiske tilkoblingen av kortleseren kan også skje på flere måter, bl.a. gjennom en vanlig serieport, mellom tastaturport og tastaturet eller integrert i tastaturet.)

PC/SC og CryptoAPI på Windows er gjennomgående basert på COM, og kan trolig brukes fra alle språk/omgivelser som har en noenlunde brukbar COM-integrasjon: Dette omfatter i hvert fall Visual C++, Visual Basic (inkludert Visual Basic-makroer i andre applikasjoner som f.eks. Office) og Microsoft Java.

De aller fleste kortleserne som er tilgjengelig har PC/SC-drivere, og mange av leverandørene av smartkort/smartkortapplikasjoner eller annen krypto-maskinvare tilbyr CSP-er for sine kort:

- ActivCard (http://www.activcard.com)
- Bull (http://www.smartcard.bull.com/)
- GemPlus (http://www.gemplus.com/)
- ID2 (http://www.id2tech.com)
- Intel (http://developer.intel.com/design/security/)
- Schlumberger (http://www.1.slb.com/smartcards/)

8.3. Open Card Framework

Open Card Framework (OCF) er et rammeverk for smartkort-anvendelser i Java. OCF er "ferskere" enn de andre industristandardene ovenfor, og støttes foreløpig ikke av noen standard-applikasjoner. OCF er heller ikke inkludert i Java-implementasjonene i hhv. Netscape og Internet Explorer, slik at OCF-bibliotekene må installeres separat. OCF 1.2 kjører under både Suns (1.1, 1.2) og Microsofts (4.x) Java-implementasjon.

Mens OCF gir generell adgang til smartkort, gir API-et Java Cryptography Extension (JCE) adgang til kryptografiske tjenester fra Java. Arkitekturen svarer til CryptoAPI og PKCS#11 i det at nye kryptotjenester installeres i form av tjenestemoduler (i dette tilfellet, Java-klasser). OCF og JCE har hver sine typer av tjenestemoduler, hhv. *CardServices* og *Providers*. (For en smartkortbasert kryptoløsning må en altså bygge en *Provider* for de krypto-operasjonene som trengs oppå en *CardService* som styrer adgangen til selve kortet.)

Formålet med å bruke OCF og/eller JCE i vår sammenheng vil være å realisere signeringsfunksjonen i en Java applet. I motsetning til PKCS#11 og CryptoAPI brukes OCF og JCE **ikke** av Web-leseren til å realisere sine egne kryptografiske operasjoner (i forbindelse med SSL etc.).

OCF har en arkitektur som i grove trekk svarer til PC/SC, men den er mer Java-spesifikk. På samme måte som PC/SC støtter den ulike smartkortbaserte tjenester, ikke bare krypto. En del generelle tjenester er predefinert i OCF (f.eks. aksess til "filsystemet" som de fleste smartkort inneholder), men det er også mulig å definere sine egne CardServices. Mekanismen for å velge kort og søke etter kort som støtter spesifikke tjenester er en del mer fleksibel enn PC/SC.

En av de standard-driverne for kortlesere i OCF går direkte mot PC/SC resource manager. Det betyr at alle kortlesere som støttes av PC/SC på en gitt Windows-installasjon også er tilgjengelige for OCF.

8.4. Kryptografiske løsninger og eksportrestriksjoner

Amerikanske eksportrestriksjoner har inntil nylig gjort det umulig å inkludere sterk krypto i amerikanske produkter for internasjonal bruk i et massemarked. Dette har rammet både Netscapes og Microsofts Web-lesere og epost-klienter, og gjort det generelt vanskelig å inkludere sterk krypto som infrastruktur i vanlige operativsystem-plattformer. Også andre land har restriksjoner på bruk eller eksport av krypto.

Nå er de amerikanske eksportrestriksjonene vesentlig lettet. Netscape Communicator i siste versjon er tilgjengelig internasjonalt med sterk krypto. For Microsoft Internet Explorer og Outlook finnes også en 128-bits-oppgradering. (Trolig er denne oppgraderingen rett og slett en oppgradering av standard-CSP-ene for CryptoAPI, slik at andre programmer som benytter krypto også får glede av dem.) Med disse oppgraderingene tilbyr Communicator, Internet Explorer og Outlook de samme kryptomulighetene som i den amerikanske ("domestic") versjonen, dvs. sterk innholdskryptering (128 biters RC4, opp til 168 biter Triple DES) og sterk digital signatur/autentisering (1024 biters RSA). Windows 2000 gir sterk kryptografi i utgangspunktet (uten noen oppgraderinger).

Utvikling av CSP-er til CryptoAPI er underlagt restriksjoner på grunn av de amerikanske eksportreglene. Utviklingsomgivelsen (SDK) for CSP-utvikling, altså biblioteker, headerfiler osv. er ikke fritt tilgjengelig, og CSP-er som blir utviklet av tredjeparter må signeres av Microsoft. CSP-er som ikke oppfyller eksportkontrollbetingelsene blir ikke signert, og vil da ikke kunne brukes under CryptoAPI. Det er uklart om dette blir endret nå. (For Netscape var det ingen tilsvarende restriksjoner, siden begrensningene på krypto-styrken lå i Netscape selv, ikke PKCS#11-modulen.)

Eksportreglene har imidlertid tillatt eksport og signering av CSP-er som tilbyr sterk kryptografi bare for digital signatur. Microsofts innebygde CSP i eksportversjonen av CryptoAPI gjør ikke det (rimeligvis fordi denne tilbyr både signering og kryptering med de samme nøklene), men det finnes smartkortbaserte CSP-er som tilbyr signering med 1024-biters RSA-nøkler. (F.eks. ID2. Microsofts egen fortegnelse over CSP-leverandører ser ikke ut til å være komplett.) En slik CSP vil være tilstrekkelig for SLN, som neppe har behov for sterk innholdskryptering. Vi antar det ganske raskt vil bli smarkort-CSP-er tilgjengelig med generelt sterk krypto.

JCE 1.2 fra Sun har ikke vært tilgjengelig utenfor USA, men det finnes uavhengige leverandører i andre land. JCE 1.2.1 finnes i en eksporterbar versjon, men det er trolig fordi den ikke inkluderer tjenestemoduler som gir sterk krypto, ikke nødvendigvis på grunn av lettelsene i eksportrestriksjonene. Såvidt vi vet er det imidlertid ikke noen restriksjoner på utvikling av tjenestemoduler for JCE.

Eksportrestriksjoner er også grunnen til at SSL-støtten til Apache distribueres som et sett med modifikasjoner til standardutgaven. (Apache, med modifikasjonene for å kunne bruke OpenSSL, men selv uten at OpenSSL med selve krypto-funksjonene var inkludert i distribusjonen, ville ikke kunne eksporteres fra USA.)

9. Evaluering og konklusjoner

Formålet med en Web-basert innrapporteringsløsning er å forbedre den EDI-baserte pilotløsningen. For å kunne ha noe grunnlag for å sammenligne, har vi stilt opp det vi anser som aktuelle evalueringskriterier nedenfor. Løsninger som er vesentlig dårligere enn den eksisterende løsningen på ett eller flere av dem er uinteressante. Kriteriene er valgt ut dels etter hvor det er rimelig å forvente at Web-baserte løsninger kan ha fortrinn overfor pilotløsningen, dels hvor vi ser for oss problemer med Web-løsninger.

1. Brukervennlighet, enkelhet, opplæringsbehov.
2. Enkelhet i installasjon hos bruker.
3. Lokalt vedlikeholdsbehov.
4. Behov for ekstern infrastruktur (tredjepartstjenester etc.)
5. Behov for tredjeparts programvare/utstyr.
6. Underskriverens grad av kontroll over signaturgenereringen[14].
7. Grad av sikkerhet.
8. Utviklingskostnader.
9. Plattformuavhengighet.

Noen av disse punktene er faktisk til en viss grad i motstrid, spesielt 1 og 2 mot 6 og 7.

Vår hovedkonklusjon er at det først og fremst er sikkerhetskravene (ikke-benektbarhet, kvalifisert signatur) som legger begrensningene, og at det er underordnet om løsningen bruker EDI-teknikker eller er Web-basert. Noen flere løsninger blir aktuelle dersom disse kravene ikke er helt absolutte, men så lenge vi fastholder dem er det meget begrenset hva vi kan oppnå på punkt 2 uten å gå på akkord med 6 og 7:

- Underskriveren **må** sørge for å bli sertifisert av en sertifiseringsautoritet som tilbyr kvalifiserte sertifikater.
- Underskriveren **må** installere smartkortleser, drivere og et akseptabelt signaturgenereringsprodukt lokalt.
- Installeres programvare over nettet, må det skje på en måte som gjør at underskriveren beholder kontroll og oversikt over hva som er installert og hvem som er ansvarlig for de modulene som faktisk er involvert i signaturgenereringen.

Med andre ord, det som kanskje er hovedankepunktet mot pilotløsningen, er det ikke mulig å gjøre noe med. Underskriveren må installere og ta ansvar for **noe** programvare og utstyr lokalt hos seg selv. Det beste vi kan oppnå på punkt 2 ovenfor, er altså en strømlinjeformet installasjonsprosess. Men dette er spørsmål om implementasjonskvalitet. Web-løsningene gjør det ikke noe prinsipielt enklere å oppnå dette, fordi alle enkle løsninger for automatisk nedlasting og installasjon av programvare blir diskvalifisert av sikkerhetskravene.

Web-løsningene vil kanskje oppfattes som mer brukervennlige fordi de nås gjennom et relativt velkjent brukergrensesnitt, dvs. en Web-leser. Men dette er også relativt. Brukervennligheten i Web-grensesnittet avhenger mest av **innholdet** og **utformingen** av Web-sidene. Dessuten vil selve signaturgenereringsapplikasjonen, enten den er en plugin, Java-applet, Active-X-kontroll eller noe annet, rent faktisk være en separat applikasjon med sitt eget brukergrensesnitt. I prinsippet er det altså liten forskjell på en EDI-basert løsning med et frittstående applikasjonsprogram og en Web-basert løsning her: Det mest merkbare er at den siste kan startes opp fra Web-leseren og under kontroll av eksterne Web-sider.

Det at EDI-løsningen er basert på EDIFACT har liten betydning, bortsett fra at det er noe større utvalg av hyllevare-verktøy for HTML og XML, slik at det kan være raskere og

[14] Dette er strengt tatt et underpunkt under neste punkt, men er spesielt vesentlig i denne sammenhengen.

mindre ressurskrevende å utvikle løsningen basert på disse formatene. Hovedforskjellen mellom en EDI-løsning og en enkel Web-skjema-løsning (HTML-skjema + CGI eller lignende) er at skjemadefininisjon, syntaks og semantikk, må være så mye mer rigid definert. Dette er nok hovedgrunnen til at EDIFACT oppfattes som mer komplisert og "tungt" enn Web-skjema. Vi ser ikke noen grunn til å bytte fra EDIFACT til XML, HTML-skjema eller noe annet hvis motivasjonen bare er å unngå EDIFACT.

Web-løsningen er forskjellig fra EDI-løsningen på ett vesentlig punkt: Web-løsningen har ikke noen dokumentbase som må vedlikeholdes lokalt. (Bortsett fra muligheten for opplasting av skjema fra lokal fil.)

9.1. Kommentarer til oppdragsbeskrivelsen

Vi har noen kommentarer til "Oppdragsbeskrivelse Hovedleveranse 3C, Avgiverløsning, alternative innrapporteringskanaler", spesielt underkapittel 1.3.1 Brukerkonsept:

Hvis man forutsetter at brukeren er i stand til å produsere et utfylt skjema i en fil i et spesifisert format, kreves ikke noe annet for å sende inn denne enn en vanlig epost-klient med S/MIME-støtte: Filen hentes inn som vedlegg, og hele meldingen signeres. Dette er i prinsippet bare en variant av løsningen vi har skissert i 5.3.3. Figur 1 antyder HTTP-opplasting. Det eneste dette gir i tillegg er muligheten for umiddelbar respons, og, ved bruk av SSL, kryptering av innholdet under oversending uten at det trengs noe spesiell programvare installert på klientsiden.

Signeringsdialogen i figur 2 er lite aktuell: Skjemaene må hentes til klienten, presenteres og i hvert fall i prinsippet inspiseres av avgiveren før signatur genereres **lokalt**. En løsning hvor brukeren krysser av for et antall skjema og signerer disse i en operasjon, uten å se hvert enkelt skjema, er ikke tilfredsstillende. Vi vil i stedet foreslå at brukeren kan navigere i skjemadatabasen og selektere skjemaene som skal hentes opp for signering. Når et skjema vises på skjermen, kan det signeres i én operasjon, og signaturen kan umiddelbart returneres til skjemadatabasen.