UNIVERSITETET I OSLO
Institutt for informatikk

Formell modellering og analyse av sikkerhetsaspekter ved web-baserte offentlige registre

Masteroppgave

Line Borgund

April 2005

Forord

Denne oppgaven er skrevet som en del av en master-grad i tidsrommet januar 2003 til april 2005. Arbeidet er utført ved forskningsgruppen Presis modellering og analyse (PMA), Instutitt for Informatikk, Universitetet i Oslo.

En stor takk rettes til mine veiledere Einar B. Johnsen og Olaf Owe som har gitt konstruktive tilbakemeldinger, fagelige innspill, og de har vært imøtekommende og hjelpsomme gjennom hele perioden.

Jeg ønsker også å takke Pål-Rune Stenersen for hjelp med tekniske problemer og moralsk støtte.

Oslo fredag, 29. april 2005

.....
Line Borgund

Innholdsfortegnelse

1 Innledning

Denne oppgaven har basis i et samarbeid mellom Brønnøysundregistrene og forskningsgruppen "Presis Modellering og Analyse" (PMA) ved institutt for Informatikk. Oppgaven inngår som et element i PMAs nyeste satsningsområde - sikkerhet og sårbarhet [1]. Oppgavene innen dette området går ut på å utvikle abstraksjoner av konkrete systemer og sikkerhetsscenarier som tillater høynivå analyse ved hjelp av modelleringsspråket Maude. Det er også ønskelig å knytte nærmere forbindelser til næringsliv.

1.1 Bakgrunn

Oppgaven knytter seg til Altinn, som er en webportal for innrapportering av data fra både enkeltpersoner, bedrifter og etater til offentlige registre. Altinn brukes for eksempel ved innlevering av selvangivelse over Internett. Sikkerhet i et slikt system er viktig fordi man for eksempel må forsikre seg om at ingen kan endre andres selvangivelse.

Oppgaven baserer seg på observasjoner foretatt på den eksisterende implementasjonen av webportalen, samt dokumentasjon av denne fra Brønnøysundregistrene. Det er ønskelig å opprette en formell modell for systemet slik at vi får en abstrakt fremstilling av viktige sikkerhetsaspekter som knytter seg til overføring av data mellom portalen og brukerne. For å gjøre dette på en mest mulig representativ måte, kreves en forståelse av Altinns oppbygging, samt hvilke sikkerhetstrusler som er gjeldende for systemet. Deretter kan denne kunnskapen formaliseres i en modell som vil kunne gi en dypere forståelse av systemet ved at man kan simulere hendelsesforløp og undersøke ulike handlingsmønstre. Modellen vil da kunne påvise feil som kan være vanskelig å oppdage i den implementerte portalen. Modellen kan også forsterke tillitten til de modellerte mekanismene, forutsatt at modellen representerer virkeligheten på en tilfredsstillende måte, og det ikke blir oppdaget noen svakheter ved analysen av modellen.

Som modelleringsverktøy benyttes et høynivå programmeringsspråk, Maude, som gir mulighet til å lage en formell modell av et systems handlingsmønstre uten å involvere implementeringsdetaljer. Maude er

basert på omskrivings- og likhetslogikk, og inneholder simulerings-, analyse- og modellsjekkingsverktøy.

1.1.1 Formell modellering

Distribuerte systemer blir mer og mer utbredt og de er gjerne store og komplekse. Dette medfører at det er vanskelig å holde oversikten over systemene, og faren for feil i det overordnede designvalget vokser i takt med systemets kompleksitet.

En modell egner seg spesielt godt for å forstå en slik problemstilling fordi vi kan abstrahere bort detaljer som ikke er relatert til systemets spesifikasjon. Det vil si at detaljer som er vesentlig for en implementasjon, for eksempel tidseffektive algoritmer, minnehåndtering og prosesskontroll kan utelates. Vi kan i en modell konsentrere oss om hvordan systemet kvalitativt er ment å fungere.

En modell vil bestå av vesentlig mindre kode enn en implementert løsning. Den er derfor lettere å lese og forstå enn kode skrevet i et implementeringsspråk, som for eksempel C eller Java.

Det er også mulig å analysere modellen ved å foreta forskjellige kjøringer og tester, hvor man fra en gitt initialtilstand kan få en oversikt over ulike hendelsesforløp i modellen. Det er mulig å bevise egenskaper ved modellen og det er mulig å påvise feil eller svakheter i modellen. Selv om vi ikke kan garantere fravær av feil i det endelige systemet, vil formell modellering og analyse understøtte de kvalitative designvalgene i det endelige systemet.

1.1.2 Altinn

Altinn er en webportal som skal sørge for enklere kommunikasjon mellom det offentlige og brukere. Brukerne er enkeltpersoner registrert i folkeregisteret og næringsdrivende som er registrert i enhetsregisteret.

Etatene tilbyr skjemaene sine gjennom portalen slik at brukerne vil ha tilgang til å lese dem, fylle de ut og levere dem. Følgende etater er per i dag knyttet til Altinn:

- Skatteetaten
- Statistisk sentralbyrå
- Brønnøysundregistrene
- Lånekassen
- Konkurransetilsynet
- Kredittilsynet
- Fiskeri- og kystdepartementet

I tillegg vil Norges Bank, Økokrim og Produktregisteret knyttes til portalen i løpet av året.

Ved lanseringen av systemet i desember 2003 inneholdt Altinn over 80 skjemaer, og rettet seg i første omgang mot næringslivets behov for skjemarapportering. I dag innholder løsningen skjema fra lånekassen for studenter, samt mulighet for å levere selvangivelsen for lønnsmottagere og pensjonister. Flere skjemaer er planlagt å gjøres tilgjengelige gjennom portalen i fremtiden.

Altinn har fire forskjellige sikkerhetsnivåer slik at følsomme data vil være beskyttet på et høyere nivå enn mindre følsomme data.

I den fremtidige løsningen av portalen har man et ønske om tilrettelegging for også å kunne hente inn sensitive data. Datatilsynet ønsker at dataene krypteres før de forlater brukerens nettleser og beholdes kryptert til de ankommer hos etatene. Altinn ønsker kun å kryptere dataene når de sender dem over nett eller lagrer dem i en database, men ellers å operere på den ukrypterte versjonen av dataene. Brønnøysundregistrene ønsker en vurdering av hvorvidt det er mulig å lage et like sikkert system basert på disse prinsippene. Slike krav til sikkerhet krever ikke-triviell analyse. Vår tilnærming er å lage en abstrakt modell og deretter analysere denne med hensyn på ulike sikkerhetsegenskaper.

1.2 Hovedtema for oppgaven

En tidkrevende del av oppgaven har vært å sette seg inn i web-systemer og Altinns oppbygging. Deretter er det utvalgt elementer fra dette som er relevante for sikkerhet og som kan formaliseres og modelleres i Maude.

Hovedtema for oppgaven kan oppsummeres med følgende spørsmål:

- Hvordan er Altinn bygget opp, og hvilke mekanismer har nettstedet for å ivareta sikkerheten?
- Hvordan kan disse sikkerhetsmekanismene formaliseres og presenteres i en modell?
- Hvordan kan en slik modell benyttes for analyse?

Altinns produksjonsmiljø består av en rekke servere og andre enheter, slik en fullstendig modell av all informasjonsflyten vil være for omfattende til å kunne dekkes av denne oppgaven. Det er derfor nødvendig å foreta en avgrensning av problemstillingene.

Det er valgt å fokusere på en bruker som logger seg på systemet med tilgangsnivå 1 eller 2. Dette gjøres ved at det opprettes en SSL forbindelse mellom server og bruker. I denne prosessen autentiserer serveren seg, og det blir foretatt en utveksling av nøkkelinformasjon, slik at den resterende kommunikasjonen kan foregå kryptert. Ingen andre enn de kommuniserende partene skal ha tilgang til å lese kryptert informasjon. Meldingene som blir sendt vil også være integritetssikret slik at de ikke kan endres under overføring uten at mottager oppdager dette. Via SSL forbindelsen vil brukeren nå autentisere seg. Dette gjøres ved hjelp av brukernavn og passord, som er den vanligste formen for brukerautentisering over Internett. Prosedyrene for å logge seg på de forskjellige tilgangsnivåene er ulike. Det laveste nivået som er modellert er tilgangsnivå 1, hvor brukerne seg på med et forhåndsvalgt statisk passord. Ved innlogging på nivå 2 mottar brukeren et engangspassord over SMS og benytter dette til å logge seg på.

Denne modellen krever en forståelse av netteverk. Et nettverk er logisk bygget opp av flere lag, hvor SSL opererer på topp av TCP protokollen. TCP sørger for at sekvensering og sikker overføring av data. På toppen av SSL finner vi applikasjoner som tar seg av interaksjonen med brukerne, for eksempel ved innlogging.

Det er ønskelig å lage en modell som kan gjøres til gjenstand for analyse og modellsjekking for å se hvorvidt systemet kan komme i en tilstand som bryter med hensikten til de sikkerhetsmekanismene vi modellerer.

1.3 Kort oversikt over oppgaven

Kapittel 2 vil gi en kort innføring i hvordan nettverk er logisk bygget opp og hvordan kommunikasjon mellom enheter forgår over Internett.

Dette danner grunnlag for kapittel 3 som gir en forståelse av hvilke trusler som er tilstedeværende i et nettverk og hvilke mekanismer vi har for å beskytte oss mot disse truslene.

Altinn er et omfattende system, og består av mange komponenter og ulik software. Kapittel 4 har tatt utgangpunkt i observasjoner av nettstedet og Altinns dokumentasjon. Dokumentasjon har enkelte steder vært uten tilstrekkelig detaljert nivå, slik at noe av bakgrunnsstoffet stammer fra andre kilder som for eksempel programvare-leverandører. Andre steder har dokumentasjonen vært svært detaljert og det har vært behov for å gjøre forenklinger. Dette kapittelet vil gi en presentasjon av de tekniske løsningene i Altinn på et overordnet nivå.

Modelleringsspråket Maude blir introdusert i kapittel 5 hvor grunnleggende syntaks blir beskrevet. Videre blir funksjonelle- og systemmoduler gjennomgått, samt hvordan reduksjoner av uttrykk og tilstandssøk kan utføres i Maude. Kapittelet innholder en rekke eksempler, slik at forhåndskunnskap om Maude ikke er nødvendig for forståelsen av modellen som presenteres i det neste kapittelet.

Kapittel 6 beskriver modellen som er laget i Maude. Her blir hovedtrekkene i modellen omtalt og utvalgte deler av koden er gjengitt. Fullstendig kode finnes som vedlegg. Modellen vil gjengi lagdeling i nettverk presentert ved SSL og TCP. På toppen av disse protokollene er det modellert to forskjellige hendelsesforløp der brukeren logger seg på systemet med ulike tilgangsnivåer. Til slutt er det laget en modell av autorisasjonsmekanismene ved skjemautveksling hos Altinn, og denne baserer seg på at brukeren allerede har logget seg på.

Analyse av modellen blir foretatt i kapittel 7. Her blir modellen feilsjekket og testet. Det blir innført en bruker som ikke følger regler for normalt hendelsesforløp, og det blir foretatt søk for å gi en innføring i hvordan modellen kan benyttes for å undersøke om det eksisterer tilstander hvor sikkerhetsbrudd kan forekomme. Vi kan bruke Maudes søkeverktøy for å søke gjennom alle tilstander som kan nåes fra et gitt utgangspunkt, og gi oss svar på om modellen oppfyller de sikkerhetskrav som er satt.

Avslutningsvis blir det foretatt en oppsummering av de viktigste elementene i oppgaven, og resultatene vi har kommet frem til blir presentert.

2 Nettverk

Et nettverk [2] er en sammenkobling av datamaskiner og datautstyr som gjør at brukere får mulighet til å kommunisere, samarbeide og dele ressurser.

Nettverk klassifiseres i følgende kategorier:

- **LAN** (Local Area Network) er et lokalnett innenfor et forholdsvis lite område, som for eksempel en bygning.
- **MAN** (Metropolitan Area Network) er et nettverk spredt over et større område, som for eksempel en by.
- **WAN** (Wide Area Network) er et nettverk uten geografisk begrensning.

Forskjellige typer nettverk er bygget opp av ulike hardware- og software-komponenter, og kan ikke alltid kan kommunisere med hverandre direkte. Et *Internetwork* er en samling av slike nettverk som er koblet sammen, ofte ved hjelp av maskiner som kalles gateways og som utfører nødvendig oversetting. Internett hører til denne kategorien og er ett bestemt Internetwork.

Dette kapittelet vil gi en oversikt over hvordan et nettverk er bygget opp på det logiske plan og gir en kort introduksjon til Internett.

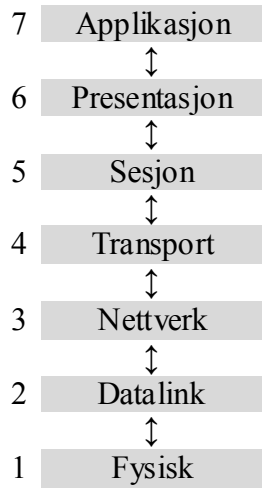
2.1 Kommunikasjonsstandarder

For at to komponenter skal ha mulighet til å kommunisere med hverandre gjøres det bruk av en protokoll. En protokoll er et regelsett som kommunikasjonspartene forventes å følge. Protokoller inneholder regler for hvordan kommunikasjonen skal startes, opprettholdes og avsluttes. Ved bruk av protokoller kan en av kommunikasjonspartene byttes ut eller endres uten at det påvirker den andre parten, så lenge de fremdeles forholder seg til samme protokoll.

2.1.1 OSI-modellen

En standardisert modell for nettverksprotokoller og distribuerte applikasjoner er OSI-modellen [2] som deler inn nettverket i syv forskjellige lag som beskrevet nedenfor. Hvert av lagene kan kommunisere

med laget over og laget under seg, og mange av lagene vil legge til en *header* til dataene. En header inneholder informasjon om hvordan dataene skal behandles, og har fått sitt navn fordi headeren legges til foran datablokken.



Lag 7 – Applikasjon

Dette laget tilbyr tjenester til endebrukerne for spesifikke anvendelser. Eksempler på dette kan være e-post eller filoverføring.

Lag 6 – Presentasjon

Her blir lokal representasjon av data konvertert til normalform. Dette betyr at dataene blir ordnet i en standard byte-rekkefølge og struktur som vil være leselig for alle kommunikasjonspartene.

Lag 5 – Sesjon

Sesjonslaget sørger for synkronisering mellom prosesser.

Lag 4 – Transport

Transportlaget deler data i mindre deler dersom det er nødvendig og sørger for at delene kommer korrekt frem til mottager. Disse delene kalles for pakker. *Transmission Control Protokoll (TCP)* [2-4] er en protokoll som arbeider på dette laget og garanterer pålitelig overføring over nettverket.

Lag 3 – Nettverk

Nettverkslaget ivaretar behovet for ruting og trafikkontroll. *Internetwork Protokoll (IP)* arbeider på dette laget og sørger for å sende dataene fra ruter til ruter gjennom nettverket til dataene når frem til mottager.

Lag 2 – Datalink

Datalinklaget sørger for pukt-til-punkt kommunikasjon og feilkontroll av dataene som sendes over nettverket.

Lag 1 - Fysisk

Det fysiske laget definerer selve overføringsmediet, som for eksempel kabler eller radiobølger.

2.1.2 TCP/IP-modellen

Selv om OSI-modellen ofte blir referert til som standarden er TCP/IP [2, 3] mye brukt i praksis, og Internett er basert på denne. Dette designet har kun fire lag og utelater eller slår sammen enkelte elementer som finnes i OSI-

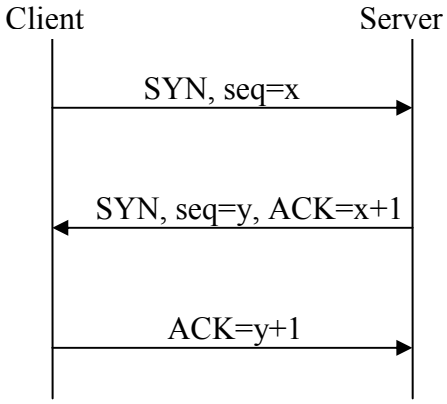
modellen. De fire lagene definert av TCP/IP-modellen er vist til venstre i figuren med de tilsvarende lag i OSI-modellen til høyre.

<table>
<tr><th>OSI</th><th>TCP/IP</th><th>Protokoller</th></tr>
<tr><td>Applikasjon</td><td rowspan="3">Applikasjon</td><td rowspan="3">HTTP</td></tr>
<tr><td>Presentasjon</td></tr>
<tr><td rowspan="2">Sesjon</td></tr>
<tr><td>Transport</td><td rowspan="2">Transport</td><td rowspan="2">TCP</td></tr>
<tr><td rowspan="2">Nettverk</td></tr>
<tr><td>Internetwork</td><td>IP</td></tr>
<tr><td>Datalink</td><td rowspan="2">Nettverksgrensesnitt</td></tr>
<tr><td>Fysisk</td></tr>
</table>

2.1.3 Transmission Control Protokoll

Internett er et upålitelig nettverk i den forstand at man ikke kan være sikker på at data som sendes kommer frem til mottager. TCP-protokollens jobb er å sørge for at alle data kommer frem og gi alle pakker som sendes et sekvensnummer slik at de kan settes sammen igjen i riktig rekkefølge hos mottager.

Når to enheter ønsker å opprette en forbindelse utføres et handshake i tre steg hvor partene blir enige om initielle sekvensnumre. Klienten starter hanshaket ved å sende en spesiell type melding som kalles en SYN, og sammen med denne sendes et sekvensnummer. Serveren svarer med en ny SYN og sender en bekreftelsesmelding (ACK) for pakken den nettopp mottok. Klienten vil tilslutt bekrefte mottak av meldingen med en ACK basert på sekvensnummeret mottatt fra Serveren. Figuren på neste side illustrerer handshaket:



Her er `x` og `y` er tallvariable og `seq` for sekvensnummer.

Når handshaket er ferdig blir alle data som skal sendes utvidet med et sekvensnummer. For hver nye pakke vil sekvensnummeret som inkrementeres med 1. Mottager vil bekrefte alle pakker som mottas med en ACK. Hvis sender ikke mottar en ACK innen en gitt tidsbegrensning vil pakken sendes på nytt.

2.2 Internett

Utviklingen av Internett startet på 1960-tallet med et prosjekt i det amerikanske forsvarsdepartementet som ble kalt ARPANET. ARPANETs hovedformål var å sikre at informasjon alltid skulle være tilgjengelig, selv om enkelte komponenter i nettverket ble satt ut av funksjon.

I 80-årene ble det utviklet nye datanettverk etter modell av ARPANET. Mange av disse ble i 1989 koblet sammen med ARPANET til det som fikk navnet Internett, et nett av nett med ulike eiere. I 1991 ble Internett åpnet for kommersiell bruk.

2.2.1 Word Wide Web

En av de mest populære tjenester som tilbys over Internett er *World Wide Web (www).* Dette er et system av Internettservere som støtter dokumenter formatert på en spesiell måte, og dokumentene kan linkes til andre dokumenter og multimediafiler. For å enkelt aksessere www benyttes en nettleser, som for eksempel ”Internett Explorer”. Dokumenter og andre ressurser har hver sin globale adresse en såkalt *Uniform Resource Locator (URL).* Første del av URL-en angir hvilken protokoll som skal brukes, den andre delen spesifiserer ip-adresse eller domenenavn hvor ressursen er lokalisert.

2.2.2 Hyper Text Transfer Protokoll (HTTP)

Hyper Text Transfer Protokoll [5, 6] er den underliggende protokollen som benyttes av World Wide Web og den opererer på toppen av TCP. HTTP definerer hvordan meldinger formateres og overføres, samt hvilke handlinger webservere og nettlesere skal utføre som svar på forskjellige forespørsler. Et eksempel på dette er når man angir en URL i nettleseren vil det sendes en HTTP-kommando til webserveren som ber den å sende angitte webside.

Alle kommandoer i HTTP er eksekvert uavhengig av hverandre, og protokollen har derfor ingen mulighet til å ha oversikt over tidligere tilstander. En rekke andre teknologier er innført for å løse dette behovet for eksempel JavaScript [7] og cookies.

2.2.3 Cookies

En *cookie* [8] er en tekststreng som kan inkluderes i HTTP meldinger. Cookies brukes som oftest å holde oversikt over tilstander og de deles inn i to kategorier avhenging hvordan de lagres hos klienten. En *session-cookie* lagres i minnet og fjernes når klienten logger ut, mens en *persistent-cookie* plasseres på harddisk og kan leses av serveren ved senere oppkoblinger.

2.2.4 HTML og XML

Hyper Text Markup Language (HTML) [9] er formateringsspråket som benyttes for websider. Formatering blir angitt av tagger og disse blir tolket av nettlesere. Ulike nettlesere og innstillinger vil derfor medføre at samme side gir ulike visninger hos forskjellige klienter.

HTMLs største begrensning ligger i at det er begrenset mengde med tagger, og dagens websider har ofte behov for mer skreddersydde løsninger. *Extensible Markup Language (XML)* er en spesifikasjon utviklet for å gi designere muligheten til å lage egendefinerte tagger, og har avskilt syntaks og semantikk.

3 Datasikkerhet

Det finnes per i dag ingen standardisert definisjon av sikkerhet. Det er imidlertid vanlig å dele sikkerhet inn de følgende tre hovedkategorier:

- **Konfidensialitet** innebærer at ingen skal kunne få tilgang til informasjon de ikke er autorisert for. Et eksempel på konfidensialitetsbrudd kan være at noen leser andres e-post. Dette kan skje ved at e-posten er feilsendt av bruker, ved softwarefeil eller ved at nettverket blir avlyttet.
- **Integritet** betyr at brukeren må kunne stole på elektronisk informasjon og at uautoriserte kilder ikke har mulighet til å utføre endringer uten at dette oppdages. Dataene må også være konsistente og meningsfulle for brukeren. Eksempler på integritetsbrudd kan være at en bruker får uautorisert tilgang til å forandre informasjon eller at en melding blir endret under transmisjon enten ved en feil eller av en angriper.
- **Tilgjengelighet** fordrer at systemet har kapasitet til å utføre forventet arbeid og ikke har uforutsett nedetid. Eksempler på brudd vil være at en tjeneste blir utilgjengelig over uakseptabel lang tid. Hvor lang tid som er akseptabel ventetid vil variere fra system til system.

Dette kapittelet vil først beskrive noen av de vanligste truslene mot sikkerhet i nettverkssystemer. Deretter introduseres kryptografi som er den grunnleggende mekanismen som benyttes for å redusere de fleste av disse truslene. Kryptografi kan enkelt anvendes for konfidensialitet, men kan også benyttes for autentisering, og dette krever et felles rammeverk for brukerne som blir introdusert i kapittel 3.3. Tilslutt beskrives SSL-protokollen som er svært mye brukt av nettsteder fordi den tilbyr både autentisering, meldingsintegritet og konfidensialitet.

3.1 Sikkerhetstrusler

Internett er et stort nettverk bygget opp av mange verter, switcher og rutere. Når man sender en melding over Internett har man ingen kontroll over hvilken sti som blir valgt for dataene og følgelig ingen kontroll over hvem som har tilgang til å lese de dataene som blir sendt. Man må derfor

anta at alt som sendes over Internett kan leses av noen med ondsinnede hensikter.

I dette avsnittet presenteres det et utvalg av vanlige angrepsformer over nettverk.

3.1.1 Website defacement

En av de mest utbredte formene for angrep er *website defacement* som gjør angriper i stand til å bytte ut innholdet på andres websider. Denne typen angrep er populær fordi den er en svært synlig og gjerne blir slått opp i media, samt at det er det relativt lett å gjennomføre [10, 11]. Her følger noen svakheter som kan være tilstede på websteder:

- **Overfylte buffere** [12, 13] er et problem hvor en prosess skriver til et sted utenfor minneplassen den har opprettet, og det kan føre til uønsket oppførsel. Den vanligste feilkilden til dette er at programmere ikke legger til sørger for tilstrekkelig kontroll av brukerinput.
- **Dot-Dot-adresseproblemer** går ut på at man ved å skrive '..' beveger seg oppover i filstrukturen i både Unix og Windows. En angriper kan derfor søke gjennom filstrukturen og få tak i applikasjoner på tjeneren og benytte disse i et videre angrep.
- **URL-en** kan ofte inneholde informasjon om en sesjon. Ved å endre URL kan en angriper endre informasjon som serveren bruker.
- **Server-side include** gjør at websider kan utføre enkelte funksjoner automatisk som for eksempel sende e-post. Problemet med slike funksjoner er at de også kan gi en angriper mulighet til å starte for eksempel en telnetsesjon fra serveren.

3.1.2 Aktiv kode og Cookies

Aktiv eller mobil kode er kode som blir lastet ned til klienten for eksekvering. Det er to hovedtyper av aktiv kode, *JavaScript* og *ActiveX*. Aktiv kode gjør servere sårbare ved at angriper kan endre URL-en og få tilgang til å kjøre programmer på serveren. Andre veien kan uvennlige script bli lastet ned til klienten for å gjøre skade der.

Cookies er ikke direkte aktiv kode, men er tekstfiler som kan lagre alt fra maskindato til tastetrykk. Disse blir lagret på brukerens maskin, ofte uten at brukeren selv kan lese innholdet i dem.

3.1.3 Spoofing

Spoofing er en fellesbetegnelse for at en inntrenger utgir seg for å være noen eller noe annet enn seg selv. Eksempler på spoofing er *maskerade, sesjonskapring* og *Man-in-the-middle.*

- **Maskerade** innebærer at en vert tilsynelatende er en annen, for eksempel ved å ha en webadresse som ligner en annen og dermed lure brukeren til å tro han er et annet sted enn han er. Et eksempel på dette er: www.dinbank.no er ikke det samme som www.din-bank.no.
- **Sesjonskapring** [14] betyr at en angriper overtar en sesjon mellom to maskiner. For eksempel kan en angriper avlytte en TCP-forbindelse og finne ut hvilke sekvensnumre som er de neste. Ved å bruke denne informasjonen sammen med ip-adressen til en av partene i kommunikasjonen vil angriperen overta denne partens plass i forbindelsen.
- **Man-in-the-middle** er en inntrenger som deltar i kommunikasjonen mellom to verter helt fra opprettelsen av en sesjon. Angriperen videreformidler all kommunikasjon mellom partene, som er uvitende om at det er en tredjepart med i sesjonen. Ved å være med fra opprettelsen vil angriperen få tilgang til hemmelige nøkler som partene forhandler seg frem til, og vil dermed få tilgang til kryptert data. Angriperen vil både kunne lese og modifisere meldinger.

3.1.4 Personifisering

Ved Personifisering *(impersonation)* utgir en angriper seg for å være en autorisert bruker og får på denne måten tilgang til systemet. Denne tilgangen kan oppnåes ved at angriperen benytter gyldig autentisering som kan fåes ved gjetning eller avlytting av nettverket. Man kan også benytte seg av svakheter i konfigurasjonen ved å:

- deaktivere autentiseringsmekanismene hos målet
- angripe mål som er uten autentisering
- angripe mål med kjent autentisering, for eksempel innebygde kontoer som Guest i Windows

3.1.5 Trusler mot meldingskonfidensialitet

Meldingskonfidensialitet er truet på grunn av nettverkenes offentlige natur. Det er mange punkter hvor en angriper kan plukke opp meldingen ved å avlytte nettverket. Det kan også forekomme at meldinger leveres til feil mottager. Dette kan skyldes feil i nettverksmaskinvare eller programmer, men i praksis er det oftest menneskelige feil som er årsak til dette. For å løse disse problemene kan man benytte kryptering som vil gjøre meldinger uleselige for inntrengere. Personifisering kan fortsatt utgjøre en trussel for konfidensialitet da uvedkommende vil kunne autentisere seg som en gyldig bruker.

3.1.6 Trusler mot meldingsintegritet

I mange tilfeller vil dataenes integritet være minst like viktig som konfidensialitet. En angriper kan falsifisere en melding slik at dens innhold blir endret, eller meldingen kan gjenbrukes slik at en gammel beskjed sendes og villeder mottageren. En angriper kan for eksempel legge til en eksekverbar fil i en melding fra en troverdig kilde slik at mottager ikke har noen betenkeligheter med å kjøre filen.

I tillegg til angrep kan også støy i transmisjonsmediet føre til at meldinger mister sin integritet. Dette problemet er løst ved kommunikasjons-protokoller som TCP/IP som sikrer mot nesten alle overføringsfeil.

3.1.7 Trusler mot tilgjengelighet

I tillegg til fysiske problemer som for eksempel linjebrudd er det mange forskjellige elektroniske angrep som kan forårsake utilgjengelighet for brukerne.

Forbindelses-flod *(flooding)* er den mest primitive av *DoS- (Denial of Service)* angrep. En angriper sender mer data enn mottager kan håndtere og dermed stopper annen trafikk opp. Forbindelses-flod finnes i flere varianter og noen av de vanligste er som følger:

- **Echo chargen** looper pakker mellom to maskiner.
- **Ping of death** går ut på at angriper sender av gårde en rekke pingpakker som mottager må svare på. Hvis angriper har mer båndbredde enn mottager vil angriper enkelt kunne oversvømme mottageren.
- **Smurf** er en variasjon av ping-angrep. Angriper sender en pingpakke i broadcast-modus, men kamuflerer den slik at den ser ut til å komme fra en annen maskin i nettverket. Alle mottagere vil nå svare maskinen som de tror forespørselen kommer fra og oversvømme denne.

Et annet populært DoS angrep kalles for *SYN-flood*. Dette angrepet utnytter TCP-protokollen ved at angriper sender en rekke SYN-pakker som mottager svarer på og legger i sin kø av mottatte SYN-pakker. Her blir pakkene liggende mens den venter på en ACK fra avsender for å gjøre ferdig oppkoblingen. Angriperen sender aldri ACK, men sender stadig flere SYN-pakker slik at denne køen går full hos mottageren.

En distribuert form for DoS skjer ved at det plasseres trojanere, som inneholder uønsket og skjult kode, hos uvitende verter. Disse vil på kommando utføre et synkronisert DoS angrep. Bruken av trojanere fører også til at det blir vanskeligere å spore hvem som står bak et angrep.

Komplekse angrep kan utføres ved ulike kombinasjoner av truslene ovenfor.

3.1.8 Angrepsverktøy

Portscan avslører hvilke standard porter som er i bruk hos maskinen som skannes og kan gi en god indikasjon på hvilket operativsystem og applikasjoner målet kjører. Programvare for dette kan lastes ned fra Internett og portscanning kan utføres personer med minimale datakunnskaper.

Sosial engineering går ut på at man lurer mennesker til å gi fra seg informasjon som kan benyttes for å bryte sikkerhetsbarrierer. Angriper kan for eksempel utgi seg for å være en ansatt i et firma og be brukerstøtte gi vedkommende et ”nytt passord”.

Rekognosering kan skje ved at søppelbøtter i eller utenfor kontorlokaler gjennomsøkes for informasjon, eller man overhører andres samtaler i for eksempel i en kantine hvor ansatte spiser lunsj og prater sammen uten å tenke på at de avslører sikkerhetsopplysninger.

Tilgjengelig dokumentasjon og nyhetsgrupper er også en trussel, da det gir tilgang og mulighet for mange til å utføre angrep de ellers ikke har forutsetninger for.

3.2 Kryptografi

Ordet kryptografi [15] kommer fra gresk og betyr “hemmelig skrift”, og ordet betegner vitenskapen dedikert til å gjøres data uleselig og deretter leselig igjen. Når man krypterer informasjon blir informasjonen sendt gjennom en algoritme som sammen med en nøkkel produserer en uleselig datamengde som kalles en *ciphertekst*. For å reversere prosessen foretas en dekryptering, ved at cipherteksten og en nøkkel som er relatert til krypteringsnøkkelen blir behandlet av en algoritme som gjenskaper den opprinnelige informasjonen.

Algoritmene antas å være kjente slik at all kryptering er avhengig av at man har nøkler som er hemmeligholdt. Dette er fordi en algoritme gjerne benyttes i lang tid og det er derfor svært usannsynlig at man klarer å holde denne skjult. Nøkler kan enkelt genereres, og dermed kan de enkelt byttes ut med forholdsvis korte tidsintervaller. Man kan også benytte forskjellige nøkler for forskjellige formål, slik at hvis en nøkkel blir kompromittert vil en angriper kun ha tilgang til å lese informasjon i en tidsbegrenset periode, og vil ikke få tilgang til andre deler av systemet.

Det finnes to forskjellige former for kryptering som vil bli omtralt i neste avsnitt. Deretter introduseres digitale signaturer og *Message Authentication Code* som er forskjellige teknikker hvor kryptering blir benyttet for å ivareta meldingsintegritet.

3.2.1 Symmetrisk og asymmetrisk kryptering

Kryptering deles inn i symmetrisk og asymmetrisk kryptering [16]. Symmetrisk kryptering innebærer at meldingsutvekslerne må kjenne til en felles hemmelig nøkkel. Kryptering og dekrypteringsprosessen bruker enten to identiske nøkler eller de bruker to ulike nøkler som kan beregnes på grunnlag av hverandre. Det betyr at selv om ulike nøkler benyttes vil man ved kjennskap til en nøkkel enkelt kunne beregne den andre.

Asymmetrisk kryptering baserer seg på at hver bruker har en privat nøkkel som er hemmelig og en tilhørende offentlig nøkkel som er tilgjengelig for alle. Disse nøklene skal ikke kunne beregnes på grunnlag av hverandre. Hvis man krypterer en melding med den offentlige nøkkelen til A er det kun As private nøkkel som kan dekryptere denne meldingen. Det er matematisk svært vanskelig å generere den private nøkkelen på grunnlag av den offentlige nøkkelen, slik at det er lite sannsynlig at en angriper vil lykkes med dette innen en rimelig tidsperiode.

De to typene for kryptering har hver sine fordeler og ulemper. Symmetrisk kryptering er den eldste formen for kryptering og er velutprøvd, svært rask og har korte nøkler.

Det eneste krypteringssystemet som er bevist sikkert er *one-time-pad* som benytter seg av symmetrisk kryptering. Systemet brukes sjelden i praksis fordi det krever distribusjon av enorme mengder med nøkkelmateriale. One-time-pad krever at nøkkelen må være like lang som dataene som skal krypteres og nøkkelen kan ikke gjenbrukes.

Den største ulempen med symmetrisk kryptering er det at for hver bruker må opprettes en nøkkel for alle andre som brukeren skal kommunisere med. Dette fører til eksponentiell vekst av antall nøkler og symmetrisk kryptering er derfor lite egnet for store nettverk.

Asymmetrisk kryptering gjør det enklere å distribuere nøkler da det kun kreves ett nøkkelpar per bruker. Man har også bare en nøkkel som må holdes hemmelig, noe som reduserer faren for at nøkkelen kompromitteres. I tillegg egner denne formen for kryptering seg svært godt til å produsere *digitale signaturer,* som vil bli beskrevet i seksjon 3.2.2.

Asymmetrisk kryptering har til gjengjeld mye lengre nøkler og er mye mindre tidseffektivt enn symmetrisk kryptering. Det er også en vesentlig nyere teknologi slik faren for uoppdagede svakheter er større.

3.2.2 Digitale signaturer og hash-funksjoner

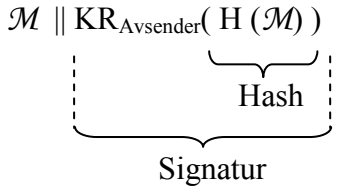
Kryptering er ressurskrevende og noen ganger har man ikke behov for å holde innholdet konfidensielt, men man har behov for dataintegritet. I disse tilfelle benytter man seg av digitale signaturer som legges til en melding for at mottager skal kunne vite med sikkerhet hvem som er opphav til meldingen, og at meldingen ikke har blitt endret underveis. En digital signatur lages ved hjelp av asymmetrisk kryptering og en *hash-funksjon*.

En hash-funksjon H er en funksjon som brukes for å lage en gjengivelse av data med vilkårlig lengde. Denne gjengivelsen kalles en hash og den har en kort og fast lengde. Hash-funksjoner er enveis slik at man ikke kan finne tilbake til de opprinnelige data gitt en hash. Hvis man endrer en bit i den opprinnelige datamengden skal i snitt 50 % av bitene i hashen forandre seg. En hash fungerer som et fingeravtrykk for dataene den er generert fra.

Følgende krav gjelder for en Hash-funksjonen H:

- H kan benyttes på en alle data uansett lengde på datablokken.
- H produserer en output på en fast lengde.
- H(x*)* er enkel å beregne gitt x.
- H må være enveis slik at gitt verdi h vil det være svært vanskelig å finne x slik at $H(x) = h$.
- For en gitt blokk x er det svært vanskelig å finne $y \neq x$ med $H(y) = H(x)$.
- Det er svært vanskelig å finne et par (x, y) slik at $H(x) = H(y)$.

For å lage en signatur tar man meldingen $\mathcal{M}$ og lager en hash av denne med en hash-funksjon H. H antas å være kjent slik at en angriper kan også generere den samme hashen. Det er derfor ikke tilstrekkelig å legge hashen til meldingen, den må krypteres i tillegg. Avsenderen tar sin private nøkkel KR og krypterer hashen, og dette danner en signatur. Denne konkateneres nå med $\mathcal{M}$. Meldingen som sendes har dermed følgende struktur:



|| representerer konkatenering i figuren.

Når mottager får meldingen vil vedkommende dekryptere signaturen med avsenders offentlige nøkkel KU. Deretter tas en hash av $\mathcal{M}$ og denne nye

hashen sammenlignes med den dekrypterte signaturen. Hvis de er like vet man at meldingen ikke er endret underveis og at avsender er den han utgir seg for å være siden ingen andre kjenner hans private nøkkel.

3.2.3 Message Authentication Code (MAC)

En MAC likner en signatur men er basert på symmetrisk kryptering. Også her blir en liten blokk lagt til meldingen som sørger for meldingsintegritet. En melding $\mathcal{M}$ blir kjørt gjennom en krypteringsfunksjon F som i tillegg tar en hemmelig nøkkel K som input. Meldingen som sendes får da følgende struktur:

$$\mathcal{M} \; \| \underbrace{\mathrm{F}(\mathrm{K}, \mathcal{M})}_{\text{MAC}}$$

Når mottager mottar meldingen vil samme operasjon utføres på $\mathcal{M}$ og resultatet sammenlignes med den mottatte MAC-en. Hvis disse er like vet man at ingen kan ha endret innholdet, og man har autentisert avsenderen siden kun to brukere kjenner den hemmelige nøkkelen som er brukt under kryptering.

3.3 Autentisering på Internett

Autentisering er prosessen med å bevise at en enhet er den som den utgir seg for å være, og for å benytte autentisering over Internett må enhetene ha en felles nøkkeldistribusjon.

3.3.1 Public Key Infrastructure (PKI)

Hvem som helst kan opprette et nytt asymmetrisk nøkkelpar, og på grunn av spoofing er det mulig å opprette nøkler i falsk navn. Dette krever et system som er slik at man kan verifisere at den man ønsker å kommunisere med virkelig er den oppgitte eieren av nøkkelparet.

Siden Internett er usikret må det settes opp en infrastruktur som gjør at det kan avgjøres hvilke kilder som er tillitsverdige og de offentlige nøklene må ligge i et register slik at de kan aksesseres globalt. Et rammeverk for dette kalles for *Public Key Infrastructure* (PKI). Det er i hovedsak to slike rammeverk:

- **PGP** [17] som er gratis og er svært utbredt innen e-post.
- **X.509** [2, 18, 19] blir blant annet benyttet i *SSL-protokollen* som blir beskrevet i seksjon 3.4.

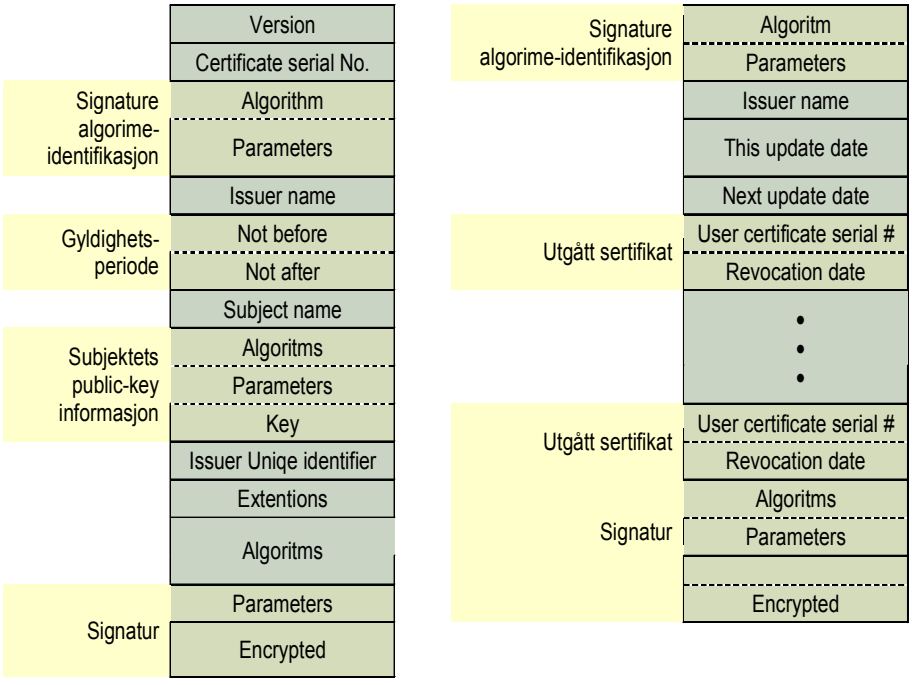
X.509 har flest kommersielle aktører og er *The Telecommunication Standardization Sectors* (ITU-T) anbefaling. Rammeverket er basert på

bruken av asymmetrisk kryptografi, digitale signaturer og sentralt finner vi sertifikater som er assosiert med hver enkelt bruker.

3.3.2 Sertifikater

Et sertifikat inneholder en offentlig nøkkel og informasjon om denne nøkkelen og dens eier. Deretter signeres hele sertifikatet av en sentral og tillitsverdig tredjepart – en *Ceritfication Authority* (CA). CA-en sørger for å validere alle nye sertifikater, ved at hver bruker oppgir sin offentlige nøkkel og identifiserer seg selv. Identifikasjon for høyeste sikkerhetsnivå bør gjøres ved at søkeren møter personlig hos utsteder. Siden dette er lite praktisk å gjennomføre benyttes gjerne andre former for identifikasjon, som f.eks. kredittkort. Etter identifikasjon oppretter CA-en et sertifikat med sin signatur over alle opplysningene i dette, og har dermed verifisert at nøkkelen tilhører den aktuelle brukeren. Siden alle nøkler i sertifikatene er offentlige har ikke CA-en noen hemmeligheter å ivareta bortsett fra sin egen private nøkkel. Et sertifikat kan derfor sendes og lagres på flere ulike destinasjoner.

Generelt sertifikat design:



Sertifikater har en oppgitt gyldighetsperiode. Noen ganger kan det likevel være ønskelig å ugyldiggjøre sertifikatet før tiden. Tilfeller hvor dette er nødvendig vil være når brukerens private nøkkel eller CA-en sertifikat er antatt kompromittert. Sertifikatet kan ikke fornyes når det først er ugyldiggjort, men det må i stedet opprettes et nytt sertifikat hvis det er

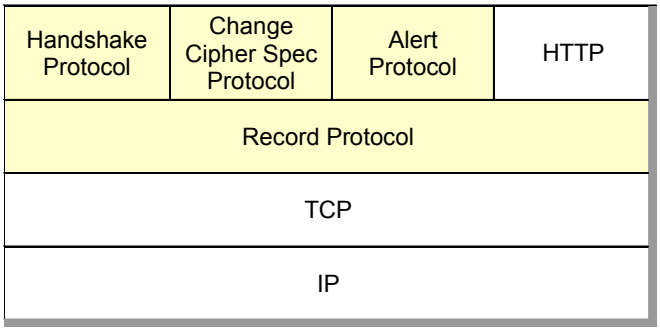
behov for det. De opphevde sertifikatene samles i en *Certificate Revocation List* (CRL) hos CA-en. Hvert sertifikat har en unik id innen CA-en og dette, samt datoen sertifikatet oppheves, lagres i CRL-en. Det er opp til hver enkelt bruker å sjekke at et mottatt sertifikat ikke er opphevet.

En CA kan utstede sertifikater til en annen CA. Dette muliggjør at CA-ene kan samles i en tre-struktur hvor det finnes en rot-CA på toppen. Tillit vil nå flyte nedover i hierarkiet, slik at hvis man har tillitt til roten vil man implisitt stole på alle CA-er nedover i treet. Roten stoler man på hvis man har lagt rotens offentlige nøkkel til sin mengde av tillitsverdige CA-er. I praksis er dette gjort ved at mange nøkler ligger forhåndslagret i nettlesere, og i tillegg kan man legge til egne.

3.4 Secure Sockets Layer (SSL)

SSL ble designet for å tilby en pålitelig og sikker ende-til-ende tjeneste mellom to kommunikasjonsparter og benyttes i dag i stor grad for å sikre kommunikasjon over Internett. SSL 3.0 [16, 20] er den nyeste versjonen av protokollen og videre i denne oppgaven vil det være denne versjonen som er omtalt.

SSL består av fire protokoller som ligger i to lag som på vises i neste figur. Protokollen gjør bruk av TCP og tilbyr en rekke grunnleggende sikkerhetstjenester til protokoller på høyere lag, som HTTP.



To prinsipper i SSL er forbindelse og sesjon:

- **Forbindelse** tilhører transportlaget i OSI-modellen og for SSL er slike forbindelser peer-to-peer-forbindelser og alle forbindelser er assosiert med en sesjon.
- **Sesjon** blir i SSL opprettet av Handshakeprotokollen og definerer et sett med kryptografiske sikkerhetsparametre som kan deles mellom flere forbindelser. Sesjoner er brukt for å unngå de ressurskrevende forhandlingene av nye sikkerhetsparametere for hver forbindelse.

3.4.1 SSL Record Protokollen

Denne protokollen tilbyr konfidensialitet og meldingsintegritet for SSL oppkoblinger. SSL Record protokollen utfører følgende trinn:

- Fragmentering hvor en melding blir delt opp i blokker på 2^{14} bytes eller mindre.
- Komprimering sørger for å redusere blokken i størrelse uten tap av data.
- Legger til MAC slik at eventuelle endringer i meldingen blir oppdaget hos mottager.
- Kryptering sørger for å ivareta konfidensialitet.
- Legger til denne protokollens header.

For både MAC og kryptering benyttes hemmelige nøklene som blir forhandlet frem og beregnet under handshaket.

3.4.2 SSL Alert Protokollen

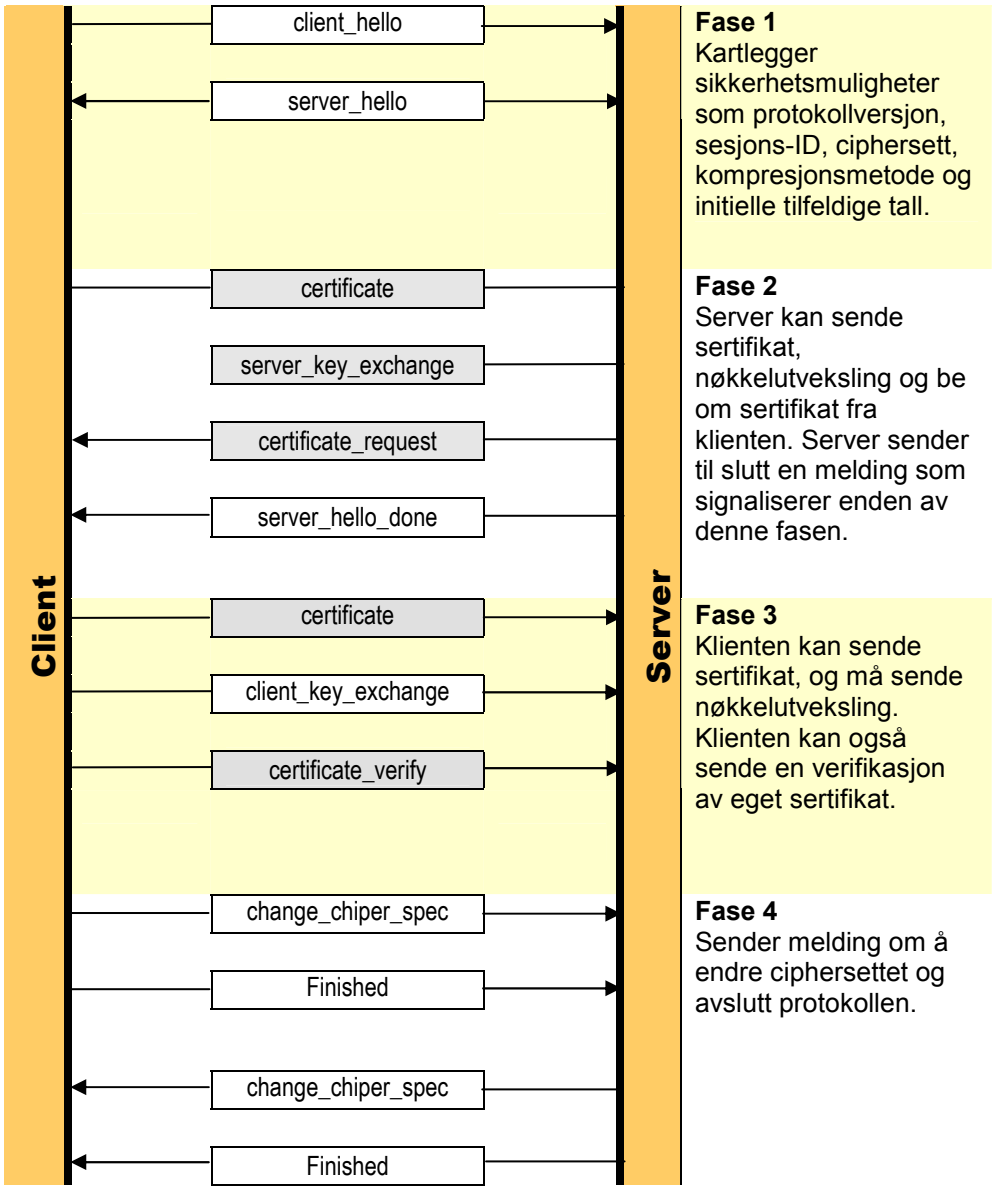
Brukes for å transportere alarmer til kommunikasjonspartner. Hver melding består av to bytes hvor den første brukes til å gradere meldingen, og den andre inneholder kode for å spesifisere alarmen.

3.4.3 SSL Change Cipher Spec Protokollen

Protokollen består av en enkel melding som kun inneholder en byte med verdi 1. Hele hensikten til denne meldingen er å overføre forhandlet ciphersett til gyldig tilstand.

3.4.4 SSL Handshake Protokoll

Denne protokollen tillater samtalepartene å forhandle seg frem til krypterings- og MAC-algoritmer. Protokollen benyttes før applikasjons-data overføres og deles inn i fire faser. Den neste figuren gir en oversikt over disse fasene og hvilke meldinger som blir sendt mellom klient og server.



De grå feltene viser meldinger som er valgfrie, mens hvite meldingene er påkrevd.

Fase 1.

Klienten starter forhandling av sikkerhetsmuligheter ved å sende en *client_hello*-melding. Denne meldingen inneholder en rekke parametere:

- **Tilfeldig verdi**: En tilfeldig struktur generert av klienten, og som består av et 32-bit tidstempel og et 28 bytes tilfeldig nummer. Disse verdiene benyttes for å forhindre reply-angrep.
- **Protokollversjon**: Dette er den høyeste versjonen av protokollen klienten kan benytte.

- **SesjonsID**: En sesjonsidentitet av tilfeldig lengde. En verdi indikerer at klienten ønsker å oppdatere parametrene på en eksisterende forbindelse eller opprette en ny forbindelse på denne sesjonen. En nullverdi angir at klienter ønsker å etablere en ny forbindelse på en ny sesjon.
- **ciphersett**: Dette er en liste som inneholder de kombinasjoner man av kryptografiske algoritmer som er støttet av klienten.
- **Kompresjonsmetode**: En liste av metoder som klienten støtter for komprimering.

Etter at *client_hello* er sendt venter klienten på å motta en *server_hello*-melding, som inneholder de samme parametrene. Serverens tilfeldige verdi genereres uavhengig av klientens verdi. Serveren velger høyeste versjon av SSL, et ciphersett, og en kompresjonsmetode som støttes av begge kommunikasjonsparter. Hvis sesjonsID-feltet fra klienten hadde en verdi vil den samme verdien bli sendt tilbake fra server. Hvis det ble sendt en nullverdi fra klienten vil serveren opprette en ny verdi som vil identifisere den nye sesjonen. Det første elementet i ciphersettet angir hva slags metode som skal benyttes for å utveksle nøkler. Her kan man velge mellom forskjellige asymmetriske krypteringsteknikker med mulighet for autentisering av en eller begge parter.

Fase 2

Hvis serveren trenger å autentisere seg vil den først sende sitt sertifikat for å innlede denne fasen. Meldingen kan inneholde ett sertifikat, eller en kjede av sertifikater i et X.509 hierarki. Meldingen *client_key_exchange* sendes dersom det er nødvendig, og dette avhenger av den første meldingen. Meldingen *certificate_request* sendes dersom serveren krever at klienten skal autentisere seg. Til slutt sendes en *server_hello_done* som angir slutten på denne fasen.

Fase 3

Klienten sender sitt sertifikat dersom serveren har bedt den om å autentisere seg. Deretter må den sende en *key_exhange* melding som inneholder en offentlig verdi som blir brukt i nøkkelutveksling, unntatt hvis sertifikatet inneholder en statisk verdi som benyttes til dette. Til slutt sendes en verifisering av sertifikatet som er en signert hash av de foregående meldingene (unntatt hvis statisk verdi er brukt).

Fase 4

Denne fasen avslutter oppkoblingen av sikker kommunikasjon. Klienten sender en *change_cipher_spec* og tar i bruk ciphersettet partene har forhandlet frem. Denne meldingen er ikke å betrakte som en del av handshake-protokollen, men er sendt ved hjelp av Change Cipher Spec protokollen. Klienten sender deretter en *finished* melding som er den første

meldingen som blir kryptert under det nye settet med algoritmer, nøkler og hemmeligheter. Meldingen verifiserer at nøkkelutveksling og eventuell autentisering var vellykket. Meldingen inneholder blant annet en hash av alle meldingene sendt i handshaket, unntatt Change Cipher Spec som ikke hører direkte til denne protokollen. Som svar på disse meldingene vil serveren sende to tilsvarende meldinger tilbake.

4 Altinn

Altinn [21-29] er en webportal som samler elektroniske skjema fra offentlige etater, og gjør de tilgjengelig for utfylling og levering for enkeltpersoner som er registrert i folkeregistret og for bedrifter som er registrert i enhetsregistret. Altinn arkiverer også skjema som er sendt inn gjennom Altinn og oppbevarer disse i 10 år.

Altinn har 4 tilgangsnivåer og hver bruker er tilordnet roller som gir en videre fininndeling av hvilke skjema brukeren har tilgang til. Dette blir beskrevet nærmere i seksjon 4.4.

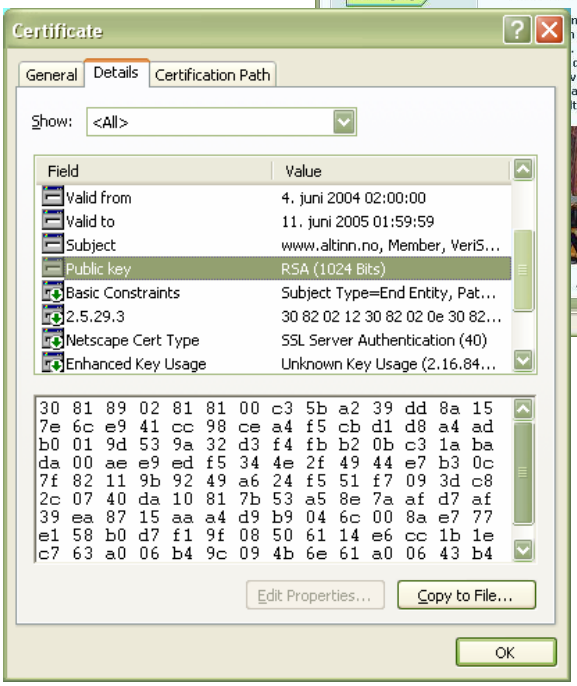
Brukeren har tilgang til å endre informasjon knyttet til seg selv under menyvalget ”min profil”. Informasjon som kan endres her er for eksempel passord og telefonnummer.

Dette kapittelet beskriver Altinns funksjonalitet og tekniske løsning. Noe er basert på egne observasjoner og noe stammer fra intern teknisk dokumentasjon fra Altinn. Det gjøres oppmerksom på at observasjon og testing av den implementerte løsningen er foretatt høsten 2004, samt at dokumentasjon fra Altinn stammer fra dette tidspunktet. Altinn er under stadig oppdatering og dette medfører at implementasjonsdetaljer kan være endret og ikke lenger i overensstemmelse med beskrivelser i dette kapittelet.

4.1 Observasjon og testing

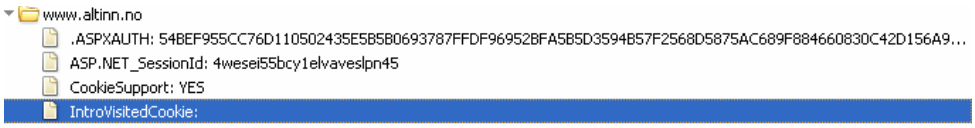
Ved å gå inn på www.Altinn.no kan man gjøre flere sikkerhetstekniske observasjoner som ikke krever kjennskap til Altinns interne dokumentasjon.

Det første som skjer er en oppkobling via SSL. Dette ser man ved at URL-en første del er ”https”.



Videre kan man lese Altinns sertifikat. I nettleseren ”MS Internett Explorer” får man åpnet sertifikatet ved å dobbeltklikke på hengelåsen nederst i høyre hjørne. Ved avlesning av sertifikatet finner man den offentlige nøkkelen til Altinn, og man ser innholdet i alle de andre feltene som beskrevet i avsnitt 3.3.2. Man kan også lese sertifiseringsstien som angir hvilke CA-er som har signert sertifikatet oppover i hierarkiet. Innloggingssiden er kun tilgjengelig gjennom SSL. Forsøk på å få tilgang gjennom HTTP protokollen var ikke vellykket.

Etter en vellykket SSL-oppkobling får man tildelt en session-cookie med en sesjonsID på 24 Byte. Man kan deretter logge seg på med brukernavn og passord og vil etter vellykket pålogging motta en autentiseringscookie, også kalt en autentikator, som inneholder autentiseringsinformasjon på 160 Byte. Cookies kan man enkelt få oversikt over i nettleseren ”Opera” som innholder verktøy for dette.



Sletter man autentikatoren blir man kastet ut fra siden, og må man logge seg inn på nytt og får en ny autentiseringscookie.

4.1.1 Svakhet ved autentikator

Autentikatoren er fremdeles gyldig selv om man har valgt ”logg ut”. Til og med etter at man har mottatt en ny sesjonsID er autentikatoren gyldig. Dette betyr at ved av- og pålogging flere ganger har man altså en rekke cookier som er gyldige. Hvis uvedkommende får tak i en autentikator kan denne benyttes uten problemer helt frem til den løper ut på tid. På en maskin som benyttes av flere brukere kan dette være en trussel for integritet og konfidensialitet.

4.1.2 Svakheter ved passordskifte

Etter innlogging med passord eller med en gyldig cookie aktivert i nettleser kan hente og modifisere en rekke skjema og oppdatere ”Min profil”. Man kan blant annet endre passord uten å bli bedt om å angi det inneværende passordet på nytt. Ved endring av telefonnummer må derimot tilleggspassord oppgis.

Alle som er registrert i folkeregistret skal ha tilgang til å bruke Altinn og dette medfører at en del brukere med dårlige kunnskaper om datasikkerhet vil kunne bruke systemet. Selv om det er laget brukerdokumentasjon vil det være rimelig å anta at mange ikke leser denne, og bruker systemet på en måte som er lite sikker. Dette kan illustreres ved følgende scenarioer:

Scenario 1

En bruker forlater sin arbeidsstasjon uten å låse den, og en forbipasserende kan uten problemer endre passordet. Den opprinnelige brukeren vil nå bli sperret ute på dette tilgangsnivået.

Man har i Altinn i tillegg til innlogging med passord også mulighet til å logge seg på med telefonnummer, og man får da tilsendt et engangspassord via SMS [30], slik at en bruker som har fått passordet sitt uautorisert endret likevel kan logge seg inn og endre passordet sitt tilbake. Det betyr at dette ikke er et stort DoS problem. Selv om integritets og konfidensialitetstrusselen er tilstedeværende, vil ikke en inntrenger få full tilgang.

Scenario 2

En bruker som går fra en datamaskin uten å låse den og legger igjen mobiltelefonen sin. Et par minutter er alt en angriper trenger for å endre telefonnummeret i Altinn. Brukeren vil nå være fullstendig utestengt fra sin egen brukerkonto, og angriper har full tilgang. Dette bryter med alle tre sikkerhetskategoriene.

4.1.3 Svakheter i brukerens omgivelser

Operativsystemer er ikke sikre og trojanere på en brukers maskin vil kunne føre til at passord og nedlastede filer kan lekke ut på nettverket [15].

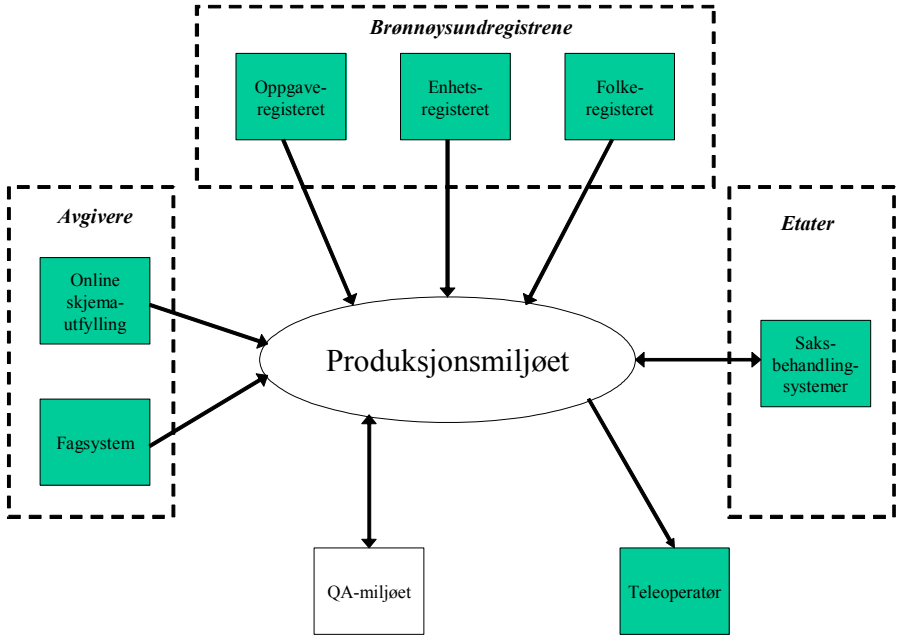
”Internett Explorer” som er en av nettleserne Altinn anbefaler er sårbar ovenfor man-in-the-middle-attacks. Sikkerhetshullet kan utnyttes ved å omdirigere dataflyt fra en sikker side til en vert med et falskt sertifikat. Verten vil da kunne erstatte hele innholdet av målets webside eller fungere som en proxy i mellom brukeren og den virkelige verten, og lese all informasjon som blir sendt i mellom disse [31, 32]. Selv om Microsoft har laget patcher for å løse problemet vil det sannsynligvis være brukere som ikke har installert dette.

4.2 Funksjonalitet

Alle figurer i dette kapittelet er hentet fra Altinns interne dokumentasjon [22-29], men da detaljer ikke alltid går klart frem av dokumentasjonen, er det noen steder foretatt antagelser og fylt ut med informasjon fra andre kilder.

4.2.1 Produksjonsmiljø

Den neste figuren er hentet fra Altinns tekniske spesifikasjon og viser det overordnede produksjonsmiljøet med grensesnitt.



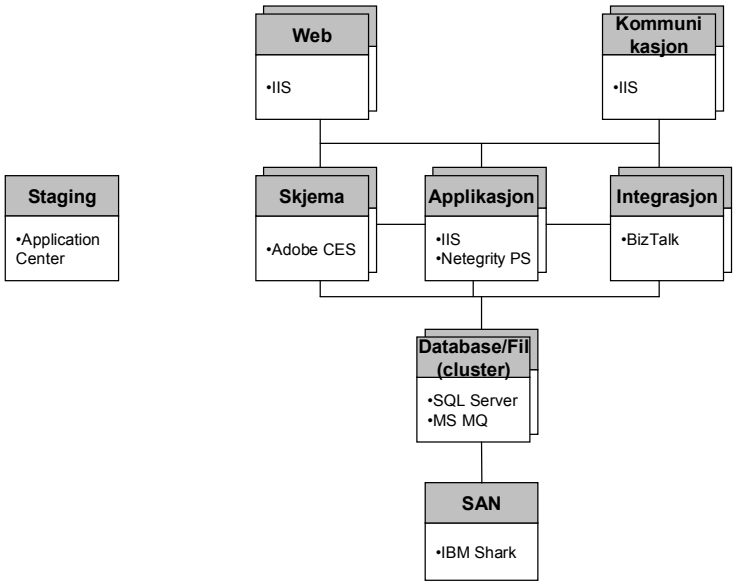
I produksjonsmiljøet skal blant annet følgende kunne gjøres:

- avgivere kan fylle ut skjemaer i en nettleser
- avgivere kan sende inn skjemaene direkte fra fagsystemene sine, som f. eks skatteprogrammer eller lignende.
- etater kan sende inn preutfyllingsdata, det vil si data som legges inn som standard i enkelte felter for den aktuelle etaten.
- etater kan motta data fra skjemaene til sine saksbehandlings-systemer
- skjema-, bruker- og rolle-informasjon kan mottas fra Brønnøysund-registrene
- pinkode kan sendes ut til mobil via SMS.

Det er i figuren på forrige side to former for avgivere:

- **Online skjemautfylling**: utføres av vanlige brukere som logger seg på Altinn.
- **Fagsystem**: består av applikasjoner som har utviklet et grensesnitt mot Altinn. Dette er stort sett økonomi og administrasjons-programvare som er utviklet av kommersielle aktører [21].

Den neste figuren viser de viktigste servere og programvaren som kjøres på hver av disse serverne i produksjonsmiljøet [25].



Figuren forstås slik at det skraverte feltet angir navnet på serveren og det hvite feltet er det angir hvilken programvare som er installert.

Web er webserver portalen i Altinn. En webserver er en datamaskin som har programvare som bruker HTML, og som er oppkoblet mot Internett. En webserver kan ha websider, gi tilgang til innhold, og svare på forespørsler fra nettlesere. En webserver har en IP-adresse og vanligvis et domenenavn. Programvaren som brukes på denne, samt alle andre webservere i figuren er Microsofts Internett Information Server [33].

Kommunikasjon er webserver for mottak av data fra avgiveres fagsystemer.

Skjema generer og ivaretar behandling av skjemaer. Av figuren ser vi at programvaren Adobe CES kjører på denne serveren. Det finnes ingen informasjon om CES på Adobes hjemmesider, eller i dokumentasjonen mottatt fra Altinn. Et annet sted i Altinns dokumentasjonen refereres det til Adobe Form Server som i følge Adobe nå heter Adobe LiveCycleForms [34]. Det er sannsynligvis dette produktet som er installert. LiveCycleForms kan konvertere mellom XML og pdf-format og vil derfor kunne fungere som en oversetter mellom databasen og webserveren.

Applikasjon har en webserver som tilbyr applikasjonstjenester som arbeidsflyt, dataaksess, logging, arkivering. Netegrity SiteMinder [35] brukes for tilgangskontroll. RegisterBC som tar seg av brukerautentisering ved hjelp av sertifikater er også installert her [36].

Integrasjon sørger for mottak og behandling av data fra avgivere, og sending av data til og fra etatssystemer. BizTalk [37, 38] tilbyr følgende tjenester:

- Ruting av datastrømmer mellom kommunikasjonspartnere.
- Tolkning og transformering av dataformater.
- Tracking og beskyttelse av utvekslinger.

Biztalk kan mappe data mellom XML-skjemaer og fungerer sannsynligvis som et bindeledd mellom Database- og Skjema-server.

Database/Fil (cluster) ivaretar behovet for lagring av for eksempel skjemadata, preutfyllingsdata, brukere, roller og arkivdata. SQL serveren [39] innholder brukeroversikt og tilbyr relasjonsdatabase med analyseverktøy.

MSMQ [40] står for Microsoft Message Queuing Center, og tillater flere applikasjoner som kjører på forskjellige tidspunkt å kommunisere over heterogene nettverk. MSMQ garanterer meldingslevering og muliggjør prioriteringsbasert meldingshåndtering.

SAN er en sentral lagringsløsning med stor kapasitet. IBM shark kalles også Enterprise Storage Server (ESS) [41]. ESS tilbyr løsning for sikkerhetskopiering og tilbakeføring av data.

Staging er en server som brukes for å teste nye eller endrede web-sider, før de gjøres aktive for brukere. Application Center [39] gir administratorer mulighet til å gruppere servere i clustere, og til å håndtere applikasjoner og websidenes innhold.

4.3 Autentiseringsmetoder

Altinn har 4 forskjellige autentiseringsmetoder [26] som gir forskjellige tilgangsnivåer:

- Statisk passord – tilgangsnivå 1
- Engangspassord via SMS – tilgangsnivå 2
- Engangspassord via post eller skattekort - tilgangsnivå 1
- Buypass Smartcard – tilgangsnivå 3 og 4

De tre første løsningene blir benyttet av brukere som kobler seg opp til Altinn via SSL og blir beskrevet nærmere nedenfor. Buypass-løsningen var ikke ferdig implementert da oppgaven ble påbegynt og vil derfor ikke bli videre omtalt.

4.3.1 Statisk passord

Løsningen bruker kombinasjon av brukernavn og et statisk passord for å autentisere brukeren. Dette er vanlig innloggingsmetode for brukere og gir tilgang til ressurser som krever innloggingsnivå 1.

Det statiske passordet opprettes når en ny bruker logger inn på Altinn for første gang. Førstegangspassord er sent via post. Senere passordendringer kan gjøres i ”min profil”.

I følge Altinns dokumentasjon [26] gjelder følgende passord-prinsipper:

> Minimum 7 character. Minimum one digit, one upper-case letter and one lower-case letter.
> After four failed login attempts the account is locked for one hour.

Det første kravet viser seg å ikke stemme overens med implementasjonen. Systemet overholder ikke kravet om å kreve både store og små bokstaver. En test av dette tillot passord med kun små bokstaver og tall. Resten av disse kravene er implementert i henhold til spesifikasjon.

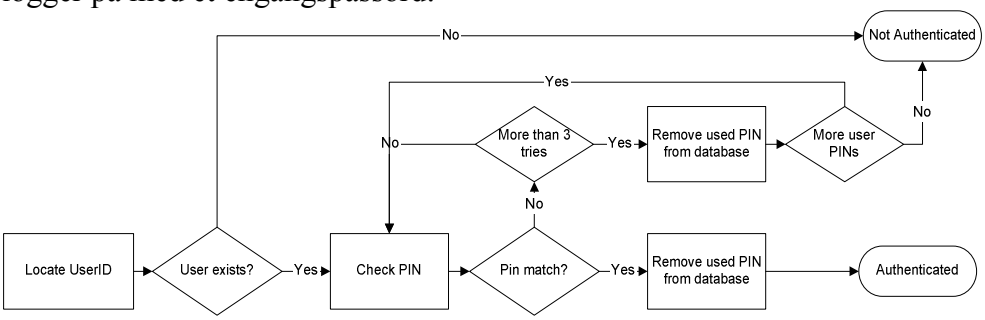
4.3.2 Engangspassord via SMS

Løsningen er basert på at engangspassord sendes til mobiltelefon via SMS. For å benytte seg av dette må bruker på forhånd ha registrert et mobiltelefonnummer.

4.3.3 Engangspassord via post eller skattekort

Løsningen baserer seg på at 30 passord kan genereres og sendes til bruker. Engangspassord kan bestilles til enhver tid av brukerne, og er påkrevd for førstegangspålogging. Når man logger på med et engangspassord mottatt per post vil man bli bedt om å lage seg et statisk passord og legge inn telefonnummer, slik at autentisering ved hjelp av statisk passord eller ved SMS-engangspassord kan gjennomføres ved senere pålogging.

På figuren nedenfor ser man en oversikt over hendelsesforløpet når man logger på med et engangspassord:



Etter tre forsøk vil passordet bli slettet og brukeren må eventuelt prøve andre passord vedkommende har fått utdelt.

4.3.4 Autentiseringsmekanismer

I Altinn blir HTTP forespørsler dirigert til et fåtall av forskjellige URL-er. Hoved URL er www.altinn.no/ega/Login/Login2.aspx. Hvilken ressurs som ønskes tilgang til blir spesifisert videre gjennom HTTP-headervariable og tilstandsinformasjon som er lagret på server.

Spørrings-parameterene er beskrevet nedenfor og rekkefølgen de forekommer i er ikke signifikant bortsett fra at L=X må være første parameter.

Parameter	Beskrivelse
L	(Level) Påkrevd innloggingsnivå. Høyere verdier krever bedre autentisering. Nivå 1 krever statisk passord og nivå 2 krever engangspassord sendt via SMS.
Target	(Function) Det er noe uklart hva denne parameteren betyr, men det er tolket slik at dette er en handling en bruker ønsker å utføre. (FormFilling, User Administration…)
O	(UnitID) Brukt for å endre nåværende organisasjonsnummer, og vil gjelde de brukere som fyller ut skjema på vegne av flere virksomheter.
P/A/fortid	(ProcessID/ArchiveID/FormID) En finkornet forespørsel etter en ressurs for å autorisere brukere på skjema-nivå.

Siden det ikke er beskrevet hvordan informasjonsflyten foregår i detalj antas det her at spørrings-parameterene blir sendt til Siteminder som avgjør om tilstrekkelig autentiseringsinformasjon og autorisasjon foreligger og sender validerte forespørsler videre til database eller skjemaserver.

Etter autentisering blir innloggingsnivå lagt til autentikatoren:

Parameter	Beskrivelse
LoginLevel	Sikkerhetsnivået som brukeren er logget inn med.
UserID	Fødselsnummeret til den autentiserte brukeren.

4.4 Autorisasjon

Funksjonen for å validere brukertilgang til en gitt ressurs vil bruke *Function* og *FormID* for å finne alle roller og autentiseringsnivåer som er kompatible med brukerens tilgangsrettigheter. Rollekravene som har det høyeste matchende nivå i brukerens rolleliste vil avgjøre hvilket autentiseringsnivå som er nødvendig [36].

En *Access Control List (ACL)* er en liste som inneholder brukere eller grupper og forteller hvilke tilganger disse har.

Hver tilstand er definert som en *Security operation* og ACL-sjekk vil bli foretatt før eksekvering av tilstanden for å verifisere autorisasjon.

Den neste tabellen viser et eksempel på en ACL hvor spørringsparametere sendt fra en bruker, blir slått opp for å finne hva slags tilhørende sikkerhets operasjon.

Function/Target	Security operation
FormFilling	Write
PDFViewer	Read
NextStep	Read/Write
Approval	Approval
UserAdministration	Admin

Deretter blir *Form ID* og *Security Operation* og *Role* brukt for å finne autentiseringsnivå. Vi har antatt at *Role* blir hentet ut fra brukerroller som er en ACL knyttet til brukernavn, og er de samme som brukeren ser under ”min profil”.

Det vil være en fil per form navngitt med *External Form ID.*

Security Operation	External Form ID	Role	Authentication level
Read	RF1122.xx.yy.zz	Reader	1
Write	RF1122.xx.yy.zz	Fillout	1
Write	RF1122.xx.yy.zz	CEO	2
Sign	RF1122.xx.yy.zz	Audit	2
…	…	…	…

Gitt tabellen ovenfor og at brukerens security operation er *Read* mot *External Form ID* RF1122.xx.yy.zz og at brukeren har rollen *Audit,* så vil kravet til autentiserings nivå være 1. Dette er nivået som matcher *Read* og *Audit*-rollen har et høyre autentiseringsnivå enn *Reader*.

5 Maude

Maude [42-45] er et eksekverbart modelleringsspråk som gjør det mulig å modellere systemer og hendelser på et abstrakt nivå, slik at implementasjonsdetaljer kan utelates.

Maude deles inn i core-Maude og full-Maude. Core-Maude inneholder språk og verktøy for likhets- og omskrivningslogikk. Full-Maude har i tillegg til dette også språk og verktøy for objektorientering.

Maude innholder analyseverktøy slik at man kan verifisere at et endeligtilstandssystem oppfyller egendefinerte krav til gyldig oppførsel. Dette skjer ved at man har mulighet til å søke gjennom alle mulige tilstander som kan nåes fra et angitt utgangspunkt.

5.1 Syntaks

Maude har en enkel syntaks slik at koden er kompakt og lettleselig, og inneholder forskjellige moduler. Disse modulene tilbyr en samling av sorter med tilhørende operatorer. Verktøy for å redusere eller omskrive uttrykk er innebygd.

Det finnes tre forskjellige typer av moduler:

- **Funksjonellmodul** (fmod) spesifiserer en flertypet likets-spesifikasjon.
- **Systemmodul** (mod) gir mulighet for omskrivningslogiske beregninger.
- **Objekt-orientertmodul** (omod) gir mulighet for objektorientering.

De to første modulene tilhører core-Maude og vil bli beskrevet nærmere. Den siste modulen tilhører full-Maude og blir ikke benyttet i modellen, og vil derfor ikke bli videre omtalt.

5.1.1 Moduler og importering av moduler

Funskjonellemoduler opprettes ved følgende syntaks:

```
fmod MODULNAVN is
BODY
endfm
```

BODY bygges opp av importerte moduler, deklarasjoner av sorter, funksjonssymboler, variable og likheter.

Systemmoduler opprettes tilsvarende:

```
mod MODULNAVN is
BODY
endm
```

BODY for systemmoduler kan inneholde det samme som i en funksjonell modul, men gir i tillegg mulighet for å opprette omskrivningsregler.

En modul kan importere en annen ved hjelp av nøkkelordet *protecting* som henter inn modulen, og den brukes slik at deklarasjoner i den importerte modulen ikke blir modifisert.

Maude har en innebygd standard bibliotek som ligger i filen prelude.Maude, og denne inneholder en rekke forhåndsdefinerte moduler. Disse modulene inneholder en rekke grunnleggende datatyper som foreksempel tall, stringer og boolske verdier, og de kan importeres inn i nye moduler. Boolske verdier blir som standard importert inn i alle nye moduler.

5.1.2 Sorter og variable

En sort gir navn til en samling med verdier som man deklarer i en modul. Sorter kan beskrive hvilke som helst type verdier, inkludert lister og sett. Nøkkelordet *sort*, eller *sorts* for multiple deklarasjoner, brukes for å opprette sorter på følgende måte:

```
sort Element .
sorts List Set .
```

Etter hver deklarasjon må det følge et mellomrom og et punktum for å markere slutten på et deklarasjonen, og denne syntaksen er gjennomgående i Maude. Denne notasjonen gjelder også for å avslutte likheter og regler som blir beskrevet senere.

Maude gir også mulighet for å opprette subsorter, det vil si man lager grupperinger som tilhører en spesiell sort. Subsorter deklareres som følger:

```
subsort Element < Set .
```

Her blir *Element* satt som en subsort av typen *Set*.

En variabel kan inneholde en ubestemt verdi for en sort og fungerer som en midlertidig lagringsplass. Variable deklareres ved å angi et variabelnavn og sorten denne variabelen representerer:

```
var E : Element .
```

Her er variabelen E opprettet og den er av sorten Element.

5.1.3 Operatorer og konstruktører

Operatorer fungerer som bindeledd mellom sorter og defineres ved nøkkelordet op. Både prefiks og infiks notasjon kan benyttes. Eksemplet nedenfor viser den samme operatoren opprettet med først infiks- og deretter prefiks-notasjon.

```
op + : Nat Nat -> Nat .
op _+_ : Nat Nat -> Nat .
```

Operatorene her angir addisjon med to naturlige tall, og resulterer i et naturlig tall.

Ved å legge til attributtet *ctor* etter en operator opprettes en konstruktør:

```
op 0 : -> Nat [ctor] .
op s : Nat -> Nat [ctor] .
```

Maude-konstanter er funksjonsuttrykk uten variable, og de er ofte konstruktører. Uttrykk bygget opp utelukkende av konstruktører kan vanligvis ikke reduseres.

I tillegg til *[ctor]* finnes andre attributter som kan legges til en operator:

- **assoc** brukes for assosiative operatorer
- **comm** brukes for kommutative operatorer
- **id:** ***TERM*** brukes for å angi et identitetselement
- **format** ***(<formateringsvalg>)*** brukes for å formatere utskrift til skjerm

Attributtene kan brukes enkeltvis eller i kombinasjon.

5.2 Funksjonelle moduler

Funksjonelle moduler definerer datatyper med tilhørende operasjoner på grunnlag av likhetsteori. Maudes funksjonelle moduler antas å ha den egenskap at de er terminerende og konfluente.

Terminering betyr at det ikke finnes noen evig løkke, og konfluens betyr at et uttrykk vil reduseres til samme resultat, uavhengig av rekkefølgen likheter blir applisert.

Det medfører at ved å applisere likheter på et uttrykk vil uttrykket reduseres helt til det ikke lenger er mulig å applisere flere likheter, og sluttresultatet kalles en normalform.

Maude vil bruke mønstergjenkjenning når samme variabelen forekommer flere ganger i et uttrykk. Det betyr at variablene med samme navn forsøkes å instansieres til samme verdi.

5.2.1 Likheter

Grunntanken med likheter er at de brukes for å forenkle uttrykk, og de brukes derfor kun fra venstre til høyre. Likheter angis med nøkkelordet *eq* etterfulgt av to uttrykk som er atskilt med et likhetstegn som vist nedenfor:

```
eq 0 + N = N .
eq s(M) + N = s(M + N) .
```

Det er ofte ønskelig at en likhet skal appliseres kun dersom en betingelse er oppfylt. Dette kan gjøres ved å bruke nøkkelordet *ceq:*

```
ceq N - M = 0 if M > N .
```

Denne likheten vil kun appliseres dersom M er større enn N.

Alternativt kan man bruke det boolske uttrykket if_then_else_fi :

```
eq max(M, N) =  if N > M then N else M fi .
```

Dette uttrykket vil alltid appliseres, men gi et ulikt resultat avhengig av verdiene til M og N.

5.2.2 Lister og Set

Lister opprettes ved å sette elementer sammen ved hjelp av en operator:

```
sorts List .
subsort Element < List .
op nil : -> List[ctor] .
op _::_ : List List -> List[ctor assoc id: nil] .
```

Her blir to elementer eller lister koblet sammen med dobbel-kolon som konkateneringsoperator. Gitt at man har listevariabler kalt *L1, L2, L3*, så vil en liste bygges opp slik: *(L1 :: L2 :: L3)*. Listevariablene kan bestå

av null eller flere elementer, og et null element for lister er kalt for nil. Dette elementet er opprettet som et identitetselement slik at:

```
(nil :: L1) = L1
(L1 :: nil) = L1
```

Lister er også opprettet med attributtet assoc slik at listene blir assosiative og dermed er `((L1 :: L2) :: L3)` identisk med `(L1 :: (L2 :: L3))`.

Multisett blir opprettet på liknende vis:

```
sorts Set  .
subsort Element < Set .
op emptySet : -> Set[ctor] .
op _;;_ : Set Set -> Set[ctor assoc comm id: emptySet]
.
```

Her er en multisett sammensatt av to `Set` som bindes sammen med operatoren dobbel-semikolon. Set har på samme måte som lister et identitetselement kalt for `emptySet` og er assosiativ. Mulitsett skiller seg fra lister ved at de er kommutative slik at `(S1 ;; S2)` er ekvivalent med `(S2 ;; S1)`, gitt at `S1` og `S2` er variable av sorten `Set`.

Et sett opprettes på samme måte som multisett, men i en mengde kan ikke samme element forekomme mer enn en gang. Det er derfor nødvendig med en tilleggsligning som blir opprettet slik:

```
eq S1 ;; S1 = S1 .
```

I dette eksemplet vil duplikater fjernet fra et multisett og redusere det til et vanlig sett.

5.2.3 Reduksjoner

Kommandoen `red` vil ta en gitt uttrykk og redusere dette til normalform, det vil si en form hvor uttrykket ikke kan reduseres videre. Maude antas å utføre likhetsreduksjoner i vilkårlig rekkefølge, og resultatet antas å være en unik normalform. Det er derfor opp til brukeren å sørge for at en unik normalform eksisterer. For å sørge for terminering er hovedregelen ved bruk av likeheter av høyresiden skal være et enklere uttrykk enn venstresiden.

5.3 Systemmoduler

Maudes systemmoduler baserer seg på omskrivningslogikk, og består av tilstander og omskrivingsregler. Tilstander er i utgangspunktet statiske, men omskrivningsreglene mapper en tilstand til en annen.

5.3.1 Omskrivningsregler

En omskrivningsregel angir overganger mellom tilstander, og reglene er ikke-reversible. I motsetning til likheter er det ingen krav til konfluens eller terminering for omskrivning. Reglene angis ved nøkkel ordet `rl` med et attributt som angir regelens navn. Gitt at vi har en konstruktør kalt age vil en omskrivningsregel se slik ut:

```
rl [birthday]: age(N)  ->  age(N + 1)  .
```

Her går vi fra tilstand `age(N)` til ny tilstand `age(N + 1).` Regler kan være betingede på samme måte som likheter, med nøkkelord `crl` eller ved bruk av det boolske uttrykket `if_then_else_fi.`

5.3.2 Konfigurasjon og tilstander

For å foreta en analyse av en systemmodul må man ha en *initialtilstand*, det vil si tilstanden til systemet i startfasen. Maude vil benytte en regel på en tilstand for å nå en ny tilstand. Hver tilstandsendring kalles et *steg*, og i hvert steg kan det utføres flere samtidige omskrivninger dersom omskrivningene er disjunkte. En tilstand som ikke lenger kan reduseres eller omskrives kalles en *sluttilstand*.

Konfigurasjonen vil være et bilde av systemets tilstand på et gitt tidspunkt, og i steg null vil konfigurasjonen være initialtilstanden.

Nøkkelordet for en konfigurasjon er `Configuration` og er bygget opp over prelude.maude sin modul for konfigurasjoner:

```
mod CONFIGURATION is
sorts Attribute AttributeSet .
  sorts Oid Cid Object Msg Configuration .
  subsorts Attribute < AttributeSet .
  subsorts Object Msg < Configuration .
  op none : -> AttributeSet .
  op _,_ : AttributeSet AttributeSet ->
AttributeSet [assoc comm id: none
    ctor] .
  op <_:_|_> : Oid Cid AttributeSet -> Object [ctor] .
  op none : -> Configuration .
  op __ : Configuration Configuration -> Configuration
[assoc comm config id:
    none ctor] .
endm
```

Vi ser at `Configuration` er en sort i Maude, og at en gitt tilstand altså er en term av denne sorten. En `Configuration` er bygges vanligvis opp av et multisett av meldinger og objekter. Disse blir listet opp etter hverandre ved

å bruke en blank operator, det vil si at de skilles med mellomrom eller parenteser.

En melding kan foreksempel defineres på følgende måte:

```
op msg_to_from_ : MsgContent Oid Oid -> Msg[ctor] .
```

Her vil meldingen få et innhold og både avsender og mottager av meldingen vil legges til meldingen.

Et objekt bygges opp av en *Object Identifier (Oid),* en *Class Identifier (Cid)* og et *AttributeSet* som består av null eller flere attributter. En konfigurasjon kan være som følger:

```
< A : Client | password: ”myPassword” >
< B : Server |   none >
(msg ”myPassword” to B from A )
```

Her har vi to forskjellige objekter av to forskjellige klasser, kalt `Client` og `Server`. I tillegg har vi en melding som er på vei fra `A` til `B`.

5.3.3 Omskrivninger

Kommandoen `rew` eller `frew` vil ta en gitt initialtilstand og applisere omskrivningsregler til det ikke lenger er mulig å benytte flere regler, og man har da kommet frem til en sluttilstand. Det antas at rekkefølgen på hvilke regler og likheter som blir benyttet er vilkårlig slik at et system som ikke er konfluent vil resultere i forskjellige sluttilstander utfra den samme initialtilstanden. Legg merke til at likheter vil redusere konfigurasjonen til normalform mellom hver regelanvendelse.

5.3.4 Søk

Reduksjoner og omskrivninger vil gi oss én mulig oppførsel av systemet, men i noen tilfeller trenger man en oversikt over alle tilstander som kan nåes. For eksempel vil det i en modell være ønskelig å undersøke om det er mulig å nå en tilstand som medfører et sikkerhetsbrudd. Maude tilbyr kommandoen `search` som vil søke gjennom alle mulige tilstander som et system kan nå fra en gitt initialtilstand så sant vi har en endelig tilstandsmengde.

Søkekommandoen i Maude foretar et bredde-først søk, slik at først vil alle tilstander som kan nåes i ett steg undersøkes, deretter alle tilstander som kan nåes i to steg, osv.

Syntaksen for søk er som følger:

```
search <initialtistand>  PIL:  <resultat>  .
```

PIL varierer i forhold til hvilke tilstander man ønsker å kjøre søket på:

=>1 tilstander som kan nåes i ett steg fra initialtilstanden.
=>* tilstander som kan nåes i null eller flere steg fra initialtilstanden.
=>+ tilstander som kan nåes i ett eller flere steg fra initialtilstanden.
=>! tilstander som ikke kan omskrives videre, det vil si sluttilstander.

Hvis man har en initaltilstand kalt init vil et søk etter alle sluttilstander se slik ut:

```
search init =>! C:Configuration .
```

Det er også mulig å snevre inn søket ved å legge til *such that* etter <*resultat*> og oppgi tilleggskrav som søkeresultatet må tilfredsstille.

```
search init =>! C:Configuration
(msg MSG:MsgContent to A:Oid from B:Oid)
such that MSG contains ”password” .
```

Dette søket vil lete etter en *Configuration* og en melding. I tillegg blir det gitt et krav om at meldingsinnholdet i meldingen skal inneholde tekst-strengen *”password”*. Søket vil derfor returnere alle sluttilstander som innholder en melding med teksten *password* som meldingsinnhold.

Det er også mulig å angi at man kun ønsker å finne et visst antall tilstander som oppfyller søkekravene. Dette angis ved å legge til en klamme med antallet tilstander man ønsker å få returnert:

```
search[1] init =>* C:Configuration
(msg MSG:MsgContent to A:Oid from B:Oid)
such that MSG contains ”password” .
```

Dette søket vil avslutte ved første den finner hvor meldingsinnholdet er lik *”password”*.

Dette betyr at man kan søke etter bestemte tilstander i et ikke-terminerende system med et uendelig tilstandsrom, og ha mulighet til å få et søkeresultat.

6 En modell av Altinn

I dette kapittelet vil vi benytte Maude til å modellere utvalgte hendelsesforløp fra Altinn. Disse utvalgte hendelsesforløpene omhandler brukernes oppkobling, innlogging og skjemautveksling mot portalen. Sikkerhetselementer som inngår ved disse handlingene vil vektlegges i modellen, og andre detaljer vil i størst mulig grad utelates.

Modellens moduler vil bli beskrevet og det blir angitt hvilke forutsetninger, kommentarer, valg og avgrensninger som er foretatt. Alle filene med Maudekode ligger vedlagt. Kapittelet vil først gi en oversikt over modellen og deretter vil hver enkelt modul bli beskrevet.

6.1 Oversikt

Modellen er basert på spesifikasjoner og på observasjoner foretatt på den faktiske implementasjonen. Enkelte steder har det vært nødvendig å foreta egne forutsetninger og gjetninger, da det ikke har foreligget dokumentasjon på et tilstrekkelig detaljert nivå.

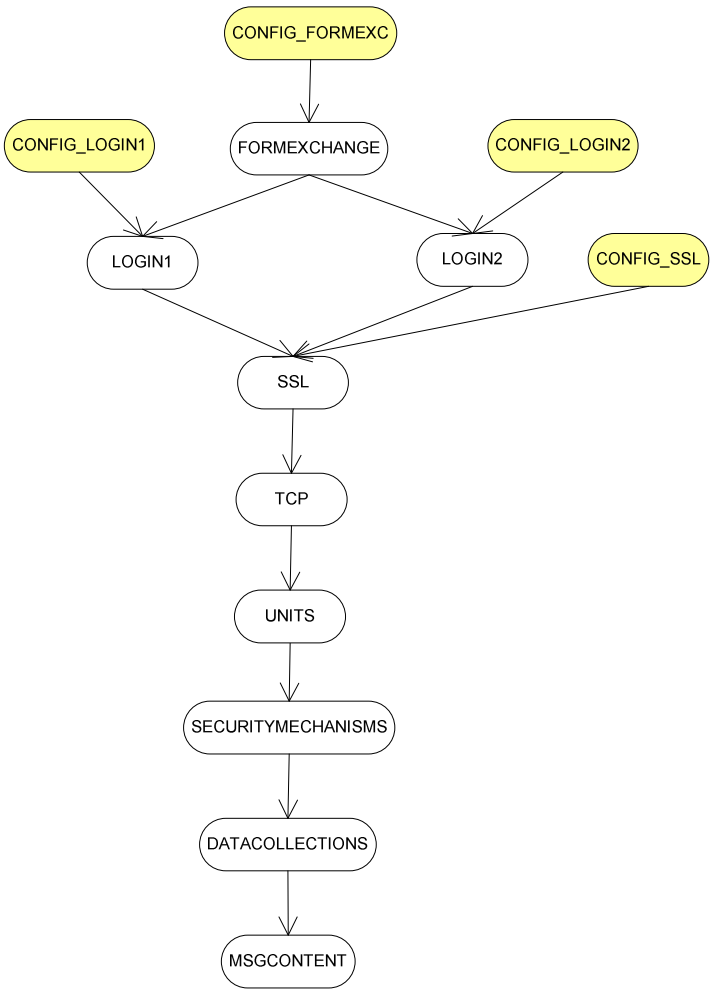
6.1.1 Kodeeksempler

Kapittelet vil inneholde en del eksempler fra koden, og det gjøres oppmerksom på at disse kodeeksemplene enkelte steder forenklet for å bedre lesbarheten. For korrekt og fullstendig kode henvises til vedlagte filer i Maudeformat.

Variabelnavn og modulnavn er konsekvent angitt med store bokstaver, og i Appendiks A ligger en liste over alle variabelnavn som er benyttet, og de samme navnene er gjennomgående i de forskjellige modulene.

6.1.2 Moduloversikt

Modellen består av en rekke forskjellige moduler som bygger på hverandre. Figuren på neste side gir en oversikt over hvordan de forskjellige modulene henger sammen.



Figuren forståes slik at MSGCONTENT er en modul som importeres ved hjelp av nøkkelordet `protecting` inn i modulen DATACOLLECTIONS, osv. I tillegg importerer de nederste modulene i hierarkiet en rekke moduler som finnes ferdigdefinert i prelude.maude, som for eksempel verktøy for tekststrenger, naturlige tall, og konfigurasjonselementer.

Hver av modulene er lagt i en fil med samme navn som modulen bortsett fra de tre nederste modulene som er samlet i filen TOOLS. De forskjellige Maudefilene har kort oppsummert følgende innhold:

- **TOOLS** inneholder en rekke grunnleggende elementer som for eksempel meldingsformatering og kryptografi.

- **UNITS** innholder rammeverk for klasser og alle klasse-attributter for klienter og servere.
- **TCP** modellerer TCP-protokollen.
- **SSL** modellerer SSL-protokollene.
- **LOGIN1** tar for seg pålogging til tilgangsnivå 1.
- **LOGIN2** lager en modell av pålogging for tilgangsnivå 2.
- **FORMEXCHANGE** tar for seg autorisasjon når en klient ønsker tilgang til et skjema i databasen.
- **CONFIG**-modulene inneholder initialtilstander slik at det vil være mulig å foreta søk og reduksjoner på de forskjellige modulene.

6.1.3 Formatering av utskrift

For å lettere skille de forskjellige elementer fra hverandre under søk- og reduseringsresultater er utskriften av klasser farge-formatert på følgende måte:

- **Blå**: attributter som tilhører SSL-protokollen
- **Cyan**: attributter tilhørende TCP-protokollen.
- **Grønn**: sertifikater og CRL.
- **Rød**: De kommuniserende enhetene i systemet.
- **Rosa**: er benyttet på alle meldinger. Det vil si meldinger sendt over Internett, meldinger internt mellom Altinn-servere og SMS.
- **Gul**: brukes på forskjellige steder for å bedre leseligheten ved å fremheve viktige attributter.

Alle attributter er samlet i filen UNITS og gjør det enkelt å finne et attributt og gi endre den, dersom det for eksempel er ønskelig å markere et attributt med en synlig farge i enkelte søk.

6.1.4 Tilleggsmodul for feilhåndtering

Det er opprettet en egen modul som tar for seg av diverse feilhåndtering. Denne modulen kalles ERRORHANDLER, og den blir importert av de andre modulene for å håndtere diverse feil som oppstår når det skjer uventede eller ugyldige handlinger i systemet. Modulen inneholder en logg som fanger noen av feilmeldingene som oppstår, og denne loggen importers inn i CONFIG-filene med å kalle operatoren `initLog`. Modulen blir ikke videre omtalt, og det henvises til den vedlagte Maude-koden for detaljer.

6.2 Gunnleggende elementer

Det første som ble modellert var en rekke grunnleggende elementer som det vil være behov for å benytte i senere moduler. Disse elementene er samlet i filen TOOLS.maude og denne består av de tre følgende moduler:

- **MSGCONTENT** angir meldingsformater.
- **DATACOLLECTIONS** definerer lister og sett, samt operatorer tilhørende disse.
- **SECURITYMECHANISMS** innholder en rekke verktøy for kryptering og autentisering.

TOOLS innholder ingen omskrivningsregler, men en rekke sorter og operatorer, samt noen få likheter. Modulene er likevel opprettet som systemmoduler fordi de importerer moduler fra prelude.maude som ikke er funksjonelle, og dermed krever Maude at også disse modulene er ikke-funksjonelle.

6.2.1 Meldinger og data

For å gjøre analyse og modellering av uventet oppførsel er meldinger delt inn i tre forskjellige formater
Meldinger som sendes over Internett,
Meldinger som sendes internt mellom servere i Altinn. Disse meldingene er beskyttet bak brannmurer og enhetene her har sin ip-adresse gjemt bak en proxy.
Meldinger sendt via SMS.

I modellen blir disse tre meldingstypene representert ved hjelp av ulike operatorer på følgende måte:

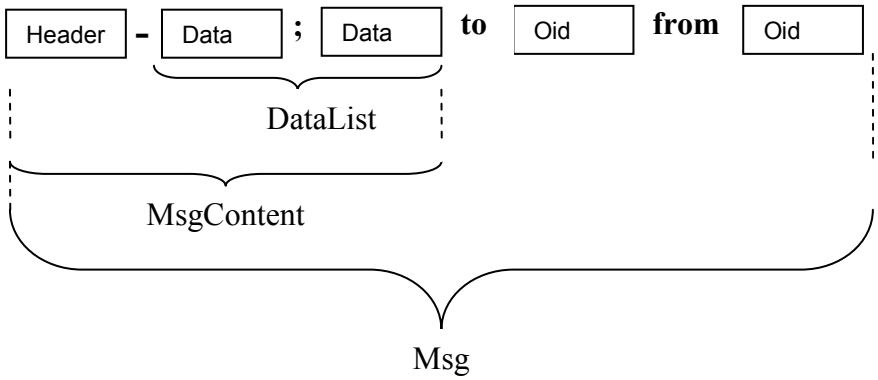
```
op msg_to_from_ : MsgContent Oid Oid -> Msg[ctor] .
op aMsg_to_from_ : MsgContent Oid Oid -> Msg[ctor] .
op sms_to_from : MsgContent Oid Oid -> Msg[ctor] .
```

Meldinger sendt over Internett er den vanligste typen av meldinger i modellen og har derfor fått den øverste operatoren. aMsg står for Altinn-Messages, og meldinger som blir sendt internt mellom Altinns servere vil bruke denne operatoren. Tilslutt har vil meldingstypen for SMS som blir benyttet under påloggingsprosedyren for tilgangsnivå 2.

Vanlige meldinger vil kunne avlyttes og regnes for tilgjengelige for alle. Altinn-meldinger er beskyttet bak brannmurer og kan derfor i teorien ikke avlyttes av andre. Disse meldingene vil i en videre analyse av modellen bli ansett for å være skjulte for alle andre objekter enn Alltinns-servere. SMS blir også betraktet som sikkert i den videre modellen selv om denne protokollen har kjente svakheter [46].

Kroppen til de ulike meldingene er av sort `MsgContent` som bygges opp av en header som angir hva slags melding som er sendt slik at mottager vet hva slags respons som er forventet. Deretter kommer en liste av data som kan bygges opp av null eller flere data-elementer. `DataList` og `Header`

utgjør til sammen et element av typen `MsgContent` og bindes deretter sammen med avsender og mottager som vist på figuren:



I modulen DATACOLLECTIONS er det opprettet operatorer for lister og sett, for å håndtere mengder. Disse er bygget opp på samme måte som beskrevet i kapittel 5.2.2.

6.2.2 Kryptering

Modellen utelater kryptografiske algoritmer fordi modellen er utviklet med tanke på å analysere anvendelsen av kryptering, og ikke selve krypteringsmekanismene. Krypteringsalgoritmene antas derfor å være sikre, slik at ingen kan lese en kryptert melding uten å ha riktig nøkkel.

Vi oppretter sorter for de forskjellige nøkkeltypene: symmetriske, private og offentlige. De symmetriske nøklene vil bli benyttet i regler slik at en melding som er kryptert med en nøkkel, kun kan dekrypteres av de som innehar en identisk nøkkel. Dette kan enkelt gjøres ved mønstergjenkjenning direkte i regler hvor dekryptering skal foretas.

Når det gjelder PK-kryptering vil den private og offentligenøkkelen være ulike og mønstergjenkjenning vil ikke lenger være mulig ved dekryptering. I en implementasjon vil en lokal og kompleks algoritme avgjøre om meldingen lar seg dekryptere. PK-kryptering i modellen er løst ved at de lokale algoritmene er erstattet med en global mengde av nøkkelpar. Denne globale mengden er pakket inn i et objekt slik at det er tilgjengelig for alle andre objekter og kan se slik ut:

```
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" &
"KR") >
```

Objektet inneholder par av asymmetriske nøkler, og brukes for oppslag når en enhet ønsker å dekryptere. Objektet vil brukes senere i regler på en slik

måte at en dekryptering kun kan foretas av en enhet som innehar riktig nøkkel. Det betyr at objektet må ha den private nøkkelen som er i par med den offentlige krypteringsnøkkelen i `keySet`-attributtet. Opprettelsen av nøkler er ikke regelbundet i modellen, men gjøres i CONFIG-filene slik at man kan endre initialtilstanden manuelt avhenging av hva slags søk eller omskrivninger som skal utføres. Objektet vil ikke endres av likheter eller regler, slik at det holder seg konstant under omskrivningsprosessen.

Siden settet for nøkkelpar er et hjelpeobjekt og det er forutsatt at de kryptografiske algoritmene er sikre, impliserer dette at ingen kan lese informasjon ut av dette objektet utover det å avgjøre om to nøkler hører sammen under en dekryptering. Det vil derfor *ikke* eksistere regler eller likheter hvor nøkkelpar blir lest ut av dette objektet. Ønsker man å hente ut en offentlig nøkkel brukes sertifikat-objekter som er ment til dette formålet og blir beskrevet i seksjon 6.2.8.

6.2.3 MAC

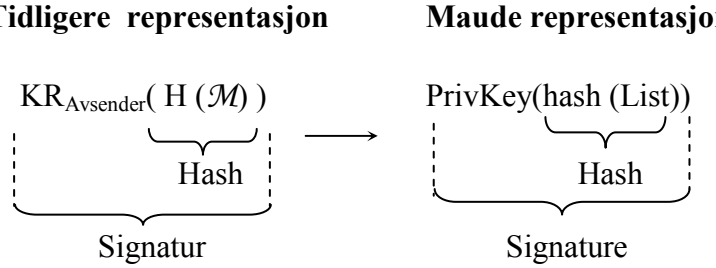
MAC dannes av en melding og en symmetrisk nøkkel, og er i samsvar med hvordan dannelsen av MAC er beskrevet tidligere. Nedenfor sees figuren fra kapittel 3.2.3 til venstre og til høyre vises hvordan denne er overført til Maude:

Tidligere representasjon:		**Maude representasjon:**
F(K, $\mathcal{M}$)	$\longrightarrow$	mac(Msg, SymmKey)

Maude-reprentasjonen blir svært lik den tidligere representasjonen. Funksjonen F nå er representert ved operatoren `mac` som tar en symmetrisk nøkkel og en melding som argumenter. Argumentenes rekkefølge har ingen betydning slik at det ikke har noen praktisk innvirkning at de i Maude-modellen har byttet plass.

6.2.4 Signaturer

Signaturer dannes ved at det tas en hash av en MsgContent og krypterer dette med en privat nøkkel. Selve hash-funksjonen er opprettet som en tom funksjon for å symbolisere bruken av denne funksjonen. To hasher blir i modellen ansett for å være ulike hvis de ikke innholder nøyaktig samme informasjon. Dette skiller seg litt fra virkeligheten hvor kollisjoner kan forekomme, men dette vil skje svært sjelden og i følge spesifikasjons-kravene for hash-funksjoner skal ikke dette forekomme. Modellen har derfor utelatt disse anormale hendelsene. Nedenfor sees figuren fra kapittel 3.2.2 til venstre og Maude-representasjon av den til høyre.



Maude-representasjonen ligner den opprinnelige presentasjonen, men i modellen brukes hash-funksjoner til sertifikatsignaturer og til bekreftelse av meldingene mottatt i SSL-handshaket. Meldinger i modellen er definert som et objekt som overfører data mellom to parter, og disse meldingene ønsker vi ikke å hashe. Det er derfor hensiktsmessig å la hash-funksjonen ta en liste som argument.

6.2.5 Cookies

En *autentiseringsbillett (authentication ticket)* er en samling data som gir serveren nødvendig informasjon om klienten i forbindelse med autentisering. *Autentiserings-cookies* lages ved å kryptere en autentiseringsbillett. Altinns dokumentasjon inneholder ingen informasjon om hvordan en autentiseringsbillett blir laget, men det er nevnt at utviklingsverktøyet "Microsoft .Net" er benyttet under implementasjonen av systemet. Det vil være rimelig å anta at billetter blir laget på den måten som blir anbefalt av "Microsoft .Net HowTo"-dokumentasjon [47]. En billett vil derfor inneholde følgende:

- ObjectID til objektet som skal autentiseres.
- Tidspunktet når den ble opprettet.
- Tidspunkt den vil utgå.
- Tilgangssnivå som representeres av et tall fra 0-4 som tilsvarer nivåene beskrevet tidligere i seksjon 4.3.

En autentiserings-cookie i Maude vil dermed kunne se slik ut:

```
sEncrypt authTicket(CLIENT, T, T + 60, N) with SymmKey
```

Autentiseringsbilletten bygges opp av en objektidentifikator, etterfulgt av tidspunktet billetten ble opprettet. Deretter kommer tidspunktet billetten skal utløpe, og i modellen er dette satt til opprettelsestid addert med 60. Sist i billetten følger et naturlig tall hvor tilgangsnivå blir angitt. Siden mottager ikke skal kunne lese eller endre innholdet i billetten blir informasjonen tilslutt kryptert med en symmetrisk nøkkel som kun webserveren kjenner til. I praksis bør denne nøkkelen skiftes jevnlig, og med dette som utgangspunkt kan vi i modellen anta at denne nøkkelen aldri vil komme på avveie.

6.2.6 Tid

Cookiene utløper på tid og det kreves derfor en klokke-funksjonalitet i modellen. Dette er løst ved å lage et hjelpeobjekt på toppnivå i konfigurasjonen som oppdateres hver gang webserveren gjør en tilstandsendring. Tidsobjektet er opprettet med et attributt som kalles `tick` og som tar et naturlig tall:

```
<Time | tick: 0 >
```

Tidsobjektet vil det være fornuftig å sette til 0 når systemet starter. Deretter oppdateres tiden direkte i de regler hvor det er ønskelig å uttrykke et tidsforbruk ved tilstandsendringen.

```
<Time | tick: N >
=>
<Time | tick N + X >
```

N er et naturlig tall som viser tiden når regelen starter og X er en konstant som angir hvor lang tid det har tatt å kjøre regelen.

Tidsobjektet blir ikke oppdatert når en klient utfører en tilstandsendring, fordi dette vil føre til at ved et stort antall klienter, vil tiden tikke fortere enn ved et lite antall klienter, og dette stemmer ikke overens med virkeligheten. Ser vi bort fra eventuelle forsinkelser som kan forekomme ved stor pågang til serveren, vil tiden en oppkobling tar være like lang uavhengig av klient-antallet. Av samme grunn oppdateres heller ikke tiden når andre servere gjør tilstandsendring. For enkelhets skyld oppdateres tidsobjektet med 1 hver gang, selv om det i praksis vil være slik at enkelte operasjoner vil ta lengre tid enn andre. Det foreligger ingen tall på tidsforbruk i Altinns dokumentasjon. En mer detaljert tidsimplementasjon i modellen ville utelukkende basert seg på antagelser og det anses derfor som lite hensiktsmessig å inkludere.

6.2.7 Passord

For å autentisere objekter ved hjelp av passord må objekter og passord knyttes sammen. Dette er gjort ved å opprette en sort kalt `PWPair` som vil inneholde en objektID og et passord. Dette paret vil deretter kunne inngå som elementer i lister og settmengder som legges i applikasjonsserveren for å kunne autentisere klienter på tilgangsnivå 1. Tilsvarende er det opprettet en sort kalt `OneTimePair` som vil inneholde passord og objektID for klienter som kan logge seg på med tilgangsnivå 2.

6.2.8 Sertifikater og CRL

Ikke alle felter som inngår i et virkelig sertifikat eller CRL som beskrevet i seksjon 3.3.2 har betydning for modellen, og de kan derfor utelates. Følgende attributter er opprettet i modellens sertifikater:

- offentlig-nøkkel
- eier av nøkkelen
- utsteder av sertifikatet
- signatur

Slik blir et sertifikat opprettet i Maude:

```
op <_: Certificate | issuerName:_, subjectName:_,
subjectKu:_, signature:_ > :
Oid Oid Oid PubKey Signature -> Object[ctor].
```

Sertifikatet identifiseres av en unik SertifikatID. Alle felter som er relatert til algoritmer og versjoner er utelatt siden vi i modellen har antatt at kryptografiske algoritmer er sikre, og vi har følgelig ikke implementert noen. I tillegg er tidsangivelser utelatt fordi vi kan endre initialtilstanden å dermed slette sertifikater eller legge de i CRL før kjøring av modellen. Utsteder antas å ha et unikt navn i modellen slik at det ikke kreves et ekstra felt for ID i modellens sertifikat.

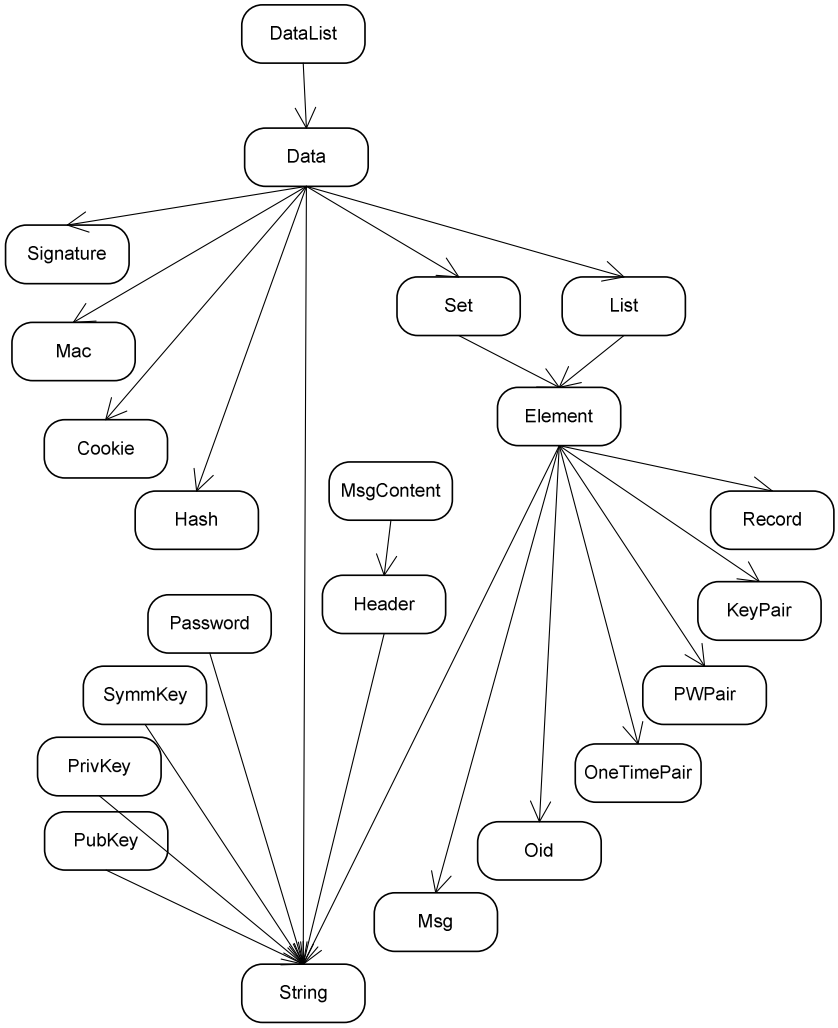
For CRL gjelder de samme argumentene for begrensing av felter og dette objektet vil bli opprettet av følgende operator:

```
op <CRL | list:_ , signature:_ > : List Signature ->
Object[ctor] .
```

CRL-objektet inneholder en liste med utgåtte sertifikatID-er utgått, samt en signatur over disse.

6.2.9 Subsorter

Sortene i TOOLS er ordnet i et subsort-hieraki som vist på figuren på neste side:



Figuren leses ovenfra og ned slik at `Signature` er en subsort av `Data` og i koden vil man blant annet finne følgende:

```
subsorts List Set Signature Hash Mac Cookie < Data .
```

Det gjøres ingen forskjeller på elementer som kan inngå i lister og sett, selv om enkelte elementer i praksis kun vil forekomme i én av dem.

Øverst i subtre-hierakiet finner vi `DataList`, som er den delen i en melding som kan inneholde informasjon. Alt som kan sendes som data i en melding vil være subsorter av `Data`, som igjen er en subsort av `DataList`.

Nøkler og passord inngår ikke direkte i *Data*, men brukes av operatorer og danner andre sorter som for eksempel signaturer og cookies.

Legg også merke til at *Header* er en subsort av *MsgContent* slik at det skal være mulig å sende meldinger som ikke har noen data-del.

6.3 Klasser og objekter

I modulen UNITS opprettes klasser og attributter for å modellere de kommuniserende enheter, det vil si servere og klienter.

```
op <_:_|_> : Oid Unit AttributeSet -> Object[ctor] .
```

Dette objektet skiller seg fra prelude.maude sin definisjon av objekter ved at det er opprettet en *Unit* i stedet for *Cid*. Funksjonsmessig kunne man ha benyttet *Cid*, men vi har valgt å opprette en egen type objekter for å få mulighet til å angi egne utskriftsformater uten å endre i prelude.maude.

Oid gir hvert objekt fra denne klassen en identitet. Det er opprettet fire konstanter av typen *Unit*:

```
ops Client WebServer ApplServer Database : -> Unit
[ctor] .
```

WebServer, *ApplServer* og *Database* er servere som ble beskrevet i seksjon 4.2.1. Disse blir modellert fordi inneholder funksjonalitet som har betydning for sikkerhetsaspektene til hendelsesforløpene vi har valgt å fokusere på. De andre serverene som ble beskrevet i den nevnte seksjonen er utelatt enten fordi de ikke inngår i de utvalgte hendelsesforløpene, eller fordi de ikke har funksjonalitet som påvirker sikkerheten.

AttributeSet vil inneholde en rekke attributter som knytter seg til klassen, og attributtene vil kunne variere for hvert enkelt objekt. Et klientobjekt vil nå kunne se slik ut:

```
< Client 1 : Client | request: L, obtainedContent:
SET, randomGenerator: N, ATTS >
```

Klientens id er her *Client 1* og den har attributtene *request*, *obtainedContent* og *randomGenerator*.

6.3.1 Attributter

Attributter knyttet til server og klientobjekter er samlet i denne modulen for å holde oversikt over hvilke attributter som er tilgjengelig, samt mulighet til gjenbruk i de forskjellige modulene.

Attributtene som ikke er direkte knyttet til spesielle moduler er listet her, mens de øvrige attributtene vil bli omtalt i forbindelse med de modulene der de først benyttes.

- **request**: Siden modellen ikke kan ta input under kjøring må hendelsesforløpet til klienten legges inn som initialverdier i konfigurasjonen. Dette attributtet er derfor opprettet for å angi hvilke operasjoner en klient foretar underveis. Etter hvert som hver enkelt handling i listen utføres fjernes den, slik at listen i sluttilstanden vil være tom, dersom den kun inneholdt lovlige verdier, og kommunikasjonen ikke har stoppet ved en annen feil.
- **obtainedContent**: Et `Set` som tar vare på all informasjonen som er mottatt av klienten. I sluttilstanden skal all informasjon klienten har bedt om å få tilsendt ligge i dette attributtet.
- **randomGenerator**: Naturlig tall som genereres ved bruk av modulen RANDOM, og tar det forrige genererte tallet og tid som input.

De forskjellige attributtene blir ikke koblet direkte til en klasse, men kan brukes av alle klasser. For eksempel vil `randomGenerator` bli brukt av både webserver og klient. Dette har en ulempe ved at det kan bli mindre oversiktelig da noen attributter naturlig hører hjemme i kun en klasse. Det er likevel valgt å lage modellen på denne måten fordi en del attributter hører hjemme i flere klasser, og man slipper da og å opprette like attributter flere ganger. I tillegg kan man enkelt legge til flere attributter da disse inngår i et `AttributeSet`. Ettersom man videreutvikler modellen vil man ikke ha behov for å endre regler, slik det ville være nødvendig å gjøre dersom man benyttet statiske klasse-attributter. Attributter er brukt på tilsvarende måte i full-Maude og det er derfor valgt å videreføre samme notasjon.

6.4 TCP-protokollen

TCP-protokollens hovedoppgaver er å sørge for en forbindelse mellom server og klient og sørge for at pakker ikke blir borte og at pakkene kommer frem i riktig rekkefølge.

Meldinger i Maude vil ikke forsvinne, og det er derfor ikke nødvendig å modellere TCP-handshaking for å ivareta sikker overføring. Maude har derimot ingen kø-ordning for meldinger, og regler kan benyttes i vilkårlig rekkefølge. Dette fører til at meldinger kan komme frem i en annen rekkefølge enn de ble sendt. TCP er derfor modellert som en underliggende modul for å ivareta behovet med sekvensering av meldinger.

6.4.1 Attributter

TCP-oppkoblingene er opprettet som delobjekter i klient og server fordi hver av de kommuniserende partene kan ha flere TCP-forbindelser samtidig. Hver av disse delobjektene inneholder følgende attributter:

- **commPartner**: kommunikasjonspartneren.
- **window**: mellomlagring for innkommende meldinger.
- **seqnrOut**: sekvensnummer for neste melding som sendes.
- **seqnrIn**: sekvensnummer for neste melding som forventes å motta.

Delobjektene blir omsluttet av klammer og de lagres i et attributt kalt *tcpSessions*. Dette er et attributt i objektene for klienter og webservere, og denne skal kunne inneholde mange delobjekter. Vi bruker sorten *Set* slik den ble opprettet i 5.2.2, og har dobbel semikolon som konkateneringsoperator. Et attributt av typen *tcpSessions* kan se slik ut:

```
tcpSessions: (SET* ;; [commPartner: CLIENT, window:
SET, seqnrIn: N, seqnrOut: N])
```

I tillegg til at hver TCP-forbindelse har sitt eget delobjekt må det også være mulig for overliggende protokoller å sende og motta informasjon gjennom TCP-protokollen. Det er derfor opprettet to attributter i klient og webserver som fungerer som et grensesnitt til TCP:

- **tcp**: En liste av meldinger som skal sendes over TCP.
- **tcpReceived**: En liste av meldinger som er mottatt over TCP. Disse meldingene vil være sortert i riktig rekkefølge etter sekvensnumre.

Protokollen består av fire regler, hvor den første erstatter handshaket og initialiserer server og klient med sesjonsobjekter. Deretter har man en regel for å sende meldinger, en for å motta meldinger og tilslutt en som sorterer meldingene i riktig rekkefølge etter sekvensnumrene.

Reglen for å sende meldinger ser slik ut i Maude:

```
rl[tcp_send]:
< A : U | tcp: ((msg MSG to B from A) :: L),
tcpSessions: ([commPartner: B, seqnrOut: N, ATTS* ] ;;
SET), ATTS >
=>
< A : U | tcp: L, tcpSessions: ([commPartner: B,
seqnrOut: (N + 1), ATTS* ] ;;  SET), ATTS >
(msg toString(N) - MSG to B from A ) .
```

Her ser vi at det er lagt en melding i tcp attributtet som er grensesnitt mot neste lag. Det blir foretatt mønstergjenkjenning av mottager *B* mot sesjonsobjektet. Ved bruk av reglen blir meldingen flyttet fra *tcpSessions* til konfigurasjonen, og den blir tillagt en header bestående av

sekvensnummer. Sekvensnummeret til avsender-objektet blir oppdatert med 1.

Mottak av meldinger skjer i to regler. Den første reglen henter en melding ut fra konfigurasjonen og legger den i mottagers window-attributt.

```
rl[tcp_receive]:
(msg toString(N) - MSG to A from B )
< A : U | tcpSessions: ([commPartner: B, window: SET,
seqnrIn: N*, ATTS* ] ;; SET*), ATTS >
=>
< A : U | tcpSessions: ([commPartner: B, window: ((msg
toString(N) - MSG to A from B) ;; SET),
seqnrIn: N*, ATTS*] ;; SET*), ATTS >  .
```

Neste regel vil deretter plukke opp meldingen som inneholder neste forventede sekvensnummer fra *window*, strippe av headeren og legge meldingen i *tcpReceived*.

```
rl[tcp_sort_messages]:
< A : U | tcpSessions: ([window: ((msg toString(N) -
MSG to A from B) ;; SET),seqnrIn: (N), ATTS*] ;;
SET*), tcpReceived: L, ATTS >
=>
 < A : U | tcpSessions: ([window: SET,
seqnrIn: (N + 1 ), ATTS*] ;; SET*), tcpReceived: (L ::
(msg MSG to A from B)), ATTS > .
```

6.5 SSL-protokollen

Filen SSL.maude importerer TCP.maude og modellerer en forenklet versjon av SSL protokollen. Alert-protokollen er utelatt fordi feilaktige tilstander likevel vil oppdages i modellen, fordi ugyldige meldinger ikke vil passe til noen regler, og føre til at protokollen stopper i tilstanden hvor feilen oppstår. SSL modulen inneholder handshaket, cipher-spec og record-protokollen.

Det er i følge SSL-spesifikasjonen flere alternative gjennomkjøringer av protokollen, avhengig av hva slags teknologi og algoritmer de kommuniserende parter benytter, samt hva slags sikkerhetsmekanismer som er nødvendig eller ønskelig. Altinn benytter seg av SSL for å autentisere seg via sertifikater og oppretter en kryptert forbindelse mellom webserver og klient. Det er derfor kun dette scenariet som er inkludert i modellen slik at serveren ikke foretar noen key_exchange, men autentiserer seg med sertifikater. Klienten derimot vil ikke autentisere seg gjennom SSL og har derfor heller ikke noe sertifikat.

6.5.1 Tilfeldige tall

En randomGenerator blir brukt både av klient og server i henhold til SSL-spesifikasjonen. Det er ikke gitt hva slags random-generator som blir brukt i implementasjonen, men det er vanlig å bruke hash-funksjoner, brukerinput og tid som input.

Vi har valgt å benytte en randomgenerator som tar et naturlig tall og gir et kvasi-tilfeldig tall tilbake fordi en slik Maude-modul allerede var tilgjengelig [48]. Denne modulen kalles RANDOM og inneholder en operator `rand` som tar et naturlig tall og returnerer et nytt naturlig tall. Tilfeldigheten i denne modulen skapes ved tre likheter.

Det er opprettet et attributt kalt `randomGenerator` som vil eksisterer i både klient- og webserver-objekter. Dette attributtet vil inneholde det neste tilfeldige tallet som objektet kan bruke. Dette attributtet må oppdateres hver gang et tilfeldig tall blir brukt. Attributtet gjør dette ved å bruke `rand`-opratoreren, og som input til denne adderes modellens nåtid til det forrige tilfeldige tallet som ble generert. Dette er gjort for å simulere virkeligheten der tid ofte brukes som tilleggsinput, for å redusere svakheter i algoritmer og maskinvare.

6.5.2 Sesjoner og tilstandskontroll

SSL holder orden på tilstander, og forventer forskjellig respons fra kommunikasjonspartneren avhengig av hvor langt de har kommet i meldingsutvekslingen. SSL jobber på toppen av TCP, som sørger for sikker overføring, og det kreves derfor ingen mekanismer for å håndtere meldingstap eller rekkefølge av meldingene i denne modulen. Det som må modelleres i er en form for tilstandsindikator. SSL-spesifikasjonen sier ingenting om hvordan dette skal løses, og dermed er det valgt en enkel metode som fungerer slik at når de kommuniserende parter sender den første meldingen til hverandre oppretter de hvert sitt sesjonsobjekt. Objektet blir lagt som et attributt i `WebServer` og `Client` og ser eksempelvis slik ut i Maude:

```
[SID : Session | commPartner: CP, keyMaterial: RANDOM,
handshakeMsgs: MSGL, state: 0, sKey: symmKey(0), mKey:
symmKey(0) ]
```

SID er sesjonsID som blir gitt ved et tilfeldig tall slik at det vil være mulig for to kommuniserende parter å opprette flere sesjoner seg i mellom. Partene sender med SID i alle meldinger i Record-protokollen for å angi hvilken sesjon meldingen tilhører. For enkelhets skyld blir SID satt til det samme tallet som det initielle tilfeldige tallet som serveren genererer, og får samme verdi både hos klient og server. Serveren bestemmer SID for å

unngå kollisjoner. Derfor blir en ny sesjon opprettet hos klienten med SID satt til 0 inntil den mottar en første melding fra server.

6.5.3 Attributter

Til SSL modulen er det opprettet en rekke nye attributter som hører til webservere og klienter. Her gis en oversikt over alle nye attributter som innføres i denne modulen, samt en kort beskrivelse av hvordan de ulike attributtene blir brukt. Attributtene kan kategoriseres i tre deler: sesjonskontroll, grensesnitt og autentiserings-attributter.

Sesjonskontroll

Attributtene nedenfor er knyttet SSL-sesjonskontroll:

- **sessionSet**: mengde av alle SSL-sesjoner hver enhet har gående.
- **commPartner**: inneholder ID-en til kommunikasjonspartneren.
- **keyMaterial**: gjennom handshaket lagres tallverdier fra de initielle tilfeldige verdiene her, slik at de kan brukes til å beregne sesjonsnøkkelen `sKey` og MAC-nøkkelen `mKey`. Når nøklene er beregnet fjernes attributtet.
- **handshakeMsgs**: liste av meldinger som er mottatt og sendt under handshaket. Denne brukes i finished-meldingene for å bekrefte at alle foregående meldinger er sett. Her blir alle meldinger som blir sendt og mottatt i handhaket lagret, frem til siste meldingen hvor det tas en hash av disse for at partene skal bekrefte ovenfor hverandre hvilke meldinger de har mottatt og sendt. Dette er i henhold til spesifikasjonen for SSL. Attributtet blir opprettet i begynnelsen av handshaket og slettet når det har utført sin funksjon på slutten av handshaket.
- **state**: et naturlig tall som brukes for å holde orden på tilstander. Tilstanden blir satt til 0 når sesjonsobjektet opprettes og øker deretter med 1 eller 2, og forventer ulike meldingstyper ettersom verdien her endres. Når handshaket er ferdig og det er på tide for Record-protokollen å overta er partene i state 4.
- **sKey**: Her lagres den symmetriske sesjonsnøkkelen når partene har forhandlet seg frem til den. Nøkkelen genereres på grunnlag av innholdet i keyMaterial.
- **mKey**: her lagres den symmetriske nøkkelen som brukes for å generere MAC. Denne skal være ulik sesjonsnøkkelen, og i følge spesifikasjonen genereres den på grunnlag av de samme tallene. I modellen er dette forenklet ved å ta sesjonsnøkkelen og legge til 1, og vi forutsetter senere at ingen kan dedusere MAC-nøkkelen på grunnlag av sesjonsnøkkelen.

Grensesnitt:

For at lag som ligger høyere opp i referansemodellene skal kunne kommunisere med SSL-protokollen er det opprettet grensesnitt som sender meldinger oppover og nedover i lagstrukturen:

- **ssl**: liste over meldinger som skal sendes over SSL og venter på å bli håndtert av Record protokollen.
- **received**: sortert liste over meldinger som er mottatt, og som er ferdig dekryptert og har fått SSL-headeren fjernet.

Autentisering

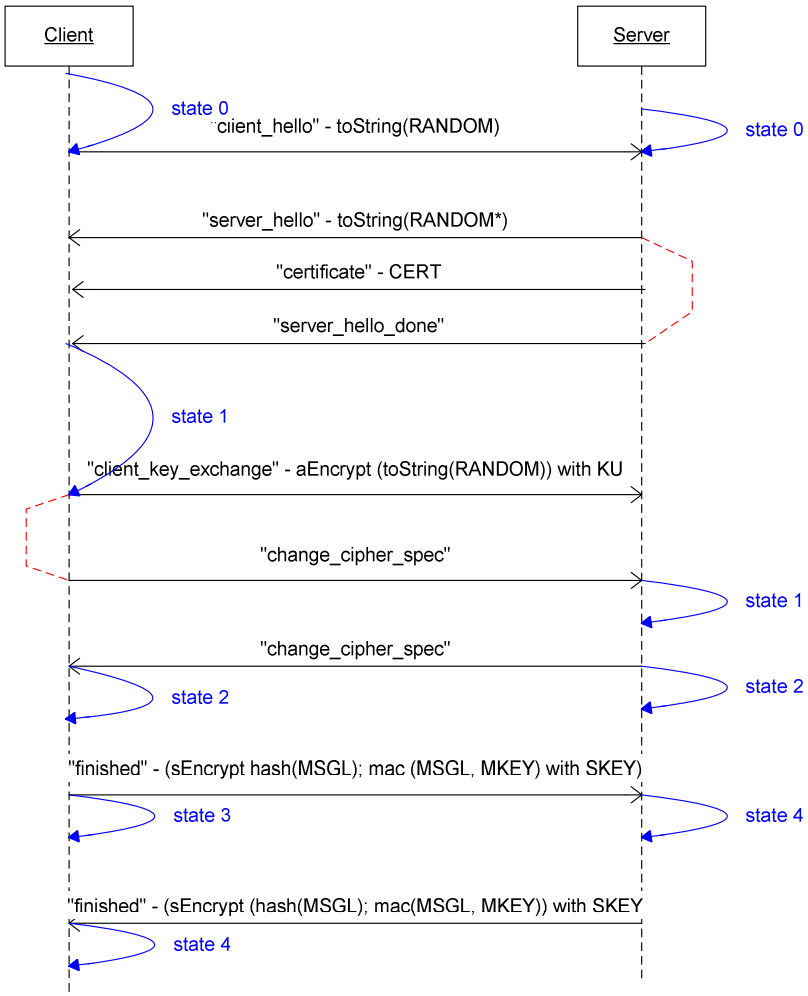
Vi har også to attributter som knytter seg til autentisering:

- **trustedIssuerSet**: mengde av de CA-er som en enhet stoler på. Dette settet blir opprettet i konfigurasjon og endrer seg ikke under kjøring.
- **privKey**: en privatnøkkel som angis av en tekststreng. Nøkler blir opprettet i konfigurasjonen og endres ikke i modellen.

6.5.4 SSL Handshake Protokoll

Handshake protokollens oppgave er å autentisere partene og opprette ett sett med felles nøkler for partene for videre kommunikasjon. Neste figur viser et sekvensdiagram over hvordan meldinger er formatert og i hvilken rekkefølge de blir sendt ved et normalt hendelsesforløp i modellen.

De blå pilene i figuren angir endringer i `state`-attributtet hos de to kommunikasjonspartene og dette attributtet blir oppdatert i samme regel som meldingen de henger sammen med på figuren. De stiplede røde linjene angir at disse meldingene blir sendt i samme regel.



Sekvensdiagram 1: SSL Handshake Protokoll

Hver av meldingene i figuren blir beskrevet nedenfor.

client_hello

Prosessen initialiseres ved at klienten ønsker å starte en SSL forhandling og dette er symbolisert i klienten ved at `request`-attributtet sitt første element er identisk med "startSSL". Det genereres et tilfeldig tall som legges i `client_hello`-meldingen som sendes til server. Maude kode for denne meldingen blir som følger:

```
(msg "client_hello" - toString(RANDOM) to SERVER from
CLIENT)
```

Det tilfeldige tallet i variabelen RANDOM blir pakket inn i operatoren `toString` for å konvertere til `String`, fordi tallverdier vil gi spaltingsfeil (parse error), fordi Maudes predefinerte moduler for strenger og tall innholder operatorer med samme navn. Variabelen kan deretter enkelt leses ut av mottager uten bruk av likninger for å konvertere typen tilbake.

server_hello

Når server mottar client_hello meldingen svarer den med å sende tilbake tre meldinger umiddelbart:

```
(msg "server_hello" - toString(RANDOM*) to CLIENT from
SERVER)
(msg "certificate" - CERT to CLIENT from SERVER)
(msg "server_hello_done" to CLIENT from SERVER)
```

Den første meldingen innholder et nytt tilfeldig tall som serveren genererer. Deretter sender serveren en melding med sertifikatID sin offentlige nøkkel som den vil benytte senere. I en implementert løsning vil serveren sende hele sertifikatet og dens kjede frem til root-CA, for å spare klienten med bryderiet og tidsforbruket med å slå opp dette selv. I modellen er sertifikater laget som egne klasser på toppnivå i konfigurasjonen og de kan derfor slås opp i nulltid og enkelt hentes opp med mønstergjenkjenning av sertifikatID. Derfor har det liten hensikt å sende selve sertifikatene med i meldingene, da dette vil føre til lengre og mindre oversiktlig kode. Den siste meldingen består kun av en header server_hello_done og markerer i følge spesifikasjon slutten på hello-meldingene.

client_key_exhange

Når klienten mottar de tre meldingene som serveren sender fra seg i regelen beskrevet ovenfor, genererer den et nytt tilfeldig tall som krypteres med klientens offentlige nøkkel.

```
(msg "client_key_exchange" - aEncrypt
(toString(RANDOM)) with KU to SERVER from CLIENT)
(msg "change_cipher_spec" to SERVER from CLIENT)
```

Serveren lagrer også den mottatte meldingen fra klienten og meldingene den sender fra seg. Dette er for senere å bekrefte kommunikasjonen med en hash av meldingene og er i henhold til spesifikasjon av SSL-protokollen.

change_chipher_spec

Meldingen `change_cipher_spec` hører til sin egen protokoll, men er sterkt knyttet til handshaket. I modellen har ikke denne forskjellen i protokoller noen betydning siden meldingsformatet er begrenset til et eneste format. Meldingens formål er å trigge partene til å aktivere bruken

av nylig forhandlede algoritmer og nøkkelmateriale. I modellen har protokollen fått regler som genererer sesjonsnøkkel og MAC-nøkkel, samt oppdaterer tilstanden til objektet for klient og server, slik at neste og siste melding i handshaket kan sendes og krypteres.

finish

I modellen er det opprettet tre regler som tar seg av finish-meldingene og avslutter handshaket:

- **client_finish:** Klienten tar en hash av alle meldinger som er utvekslet i handshaket og sender denne hashen med MAC og kryptert med sesjonsnøkkel.
- **server_finish:** tilsvarende client_finish, og oppdaterer serverens tilstand til ferdig.
- **handshake_finish:** fjerner den siste finish meldingen og oppdaterer klientens tilstand for denne sesjonen til ferdig.

Etter vellykket handshake har nå begge parter forhandlet seg frem til gyldig sesjonsnøkkel, MAC-nøkkel og oppdatert sin tilstand til ferdig, det vil si state er satt til 4. Record-protokollen tar nå over og informasjonsoverføring kan starte.

6.5.5 SSL Record Protokoll

Protokollen består av regler for å sende og motta meldinger i SSL. Meldinger som skal sendes legges av høyere lag inn i attributtet som heter `ssl`. Record-protokollen plukker så opp meldinger i rekkefølge herfra og legger til header med sesjonsID og en MAC. Meldingsinnholdet blir også kryptert og meldingen legges i tcp-attributtet, slik at TCP som ligger i laget nedenfor kan overta jobben med å sende meldingen. Alt dette blir foretatt innen en regel som er gjengitt i sin helhet her:

```
rl[record_protocol_send]:
< A : U | sessionSet: (SSET ;; [SID : Session |
commPartner: B, state: 4, sKey: SKEY, mKey: MKEY,
ATTS*]), ssl: ( (msg MSG to B from A) :: L),
tcp: L*,  ATTS >
=>
< A : U | sessionSet: (SSET ;; [SID : Session |
commPartner: B, state: 4, sKey: SKEY, mKey: MKEY,
ATTS*]), ssl: L,
tcp: ((msg toString(SID) - sEncrypt ( (msg MSG to B
from A)  ; mac ((msg MSG to B from A) , MKEY)) with
SKEY to B from A) :: L* ), ATTS > .
```

Variablene i meldingsinnholdet er som følger:

- **SID** : SessjonsID
- **MSG**: Selve innholdet i meldingen

- **MKEY**: Nøkkel for MAC-generering
- **SKEY**: Sesjonsnøkkel

Når meldinger mottas i SSL-format vil reglen for mottak foreta mønstergjenkjenning for å finne riktig sesjonsobjekt, for å validere MAC og for å sjekke om mottager har riktig nøkkel til å dekryptere meldingen. Deretter fjernes all SSL-informasjon fra meldingen, det vil si kryptering, MAC og SSL-header. Hos mottageren legges den mottatte meldingen inn i attributtet received hvor den kan plukkes opp av applikasjoner på høyere nivå, og behandles. Regelen for dette ser slik ut i Maude-kode:

```
rl[record_protocol_receive]:
< A : U | sessionSet: (SSET ;; [SID : Session |
commPartner: B, state: 4, sKey: SKEY, mKey: MKEY,
ATTS*]),
received: L, tcpReceived: ((msg toString(SID) -
sEncrypt (M ; mac (M, MKEY)) with SKEY to A from B) ::
L*), ATTS >
=>
< A : U | sessionSet: (SSET ;; [SID : Session |
commPartner: B, state: 4, sKey: SKEY, mKey: MKEY,
ATTS*]),
received: (L :: M), tcpReceived: L*, ATTS > .
```

De to reglene presentert i denne seksjonen utgjør *hele* SSL Record protokollen, og sørger for all SSL-kommunikasjon mellom partene etter at handshaket er ferdig.

6.5.6 Konfigurasjon

Konfigurasjon for SSL- og TCP-forbindelser er opprettet i filen CONFIG_SSL.maude. Denne filen vil opprette initialtistanden kalt initSSL som gir en konfigurasjon som kan brukes som grunnlag for omskrivninger og søk.

Siden det er ønskelig å endre antall klienter i konfigurasjonen på en enkel måte blir disse opprettet med hjelp av en parameter på initialoperatoren, definert ved følgende likhet:

```
eq initClients(N) =
< Client N : Client |
request: ("startTCP" :: "startSSL"),
ssl: nil , sessionSet: emptySet,  trustedIssuerSet:
Issuer1,  received: nil,
tcp: nil,  tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N) >
initClients(N - 1)  .
```

Variabelen *N* er definert som *NzNat*, det vil si variabelen kan inneholde alle naturlige tall unntatt 0. Det er ingen hensikt å opprette systemet med 0 klienter så denne muligheten er utelatt. Her blir det opprettet *N* klienter med ulike tilfeldige tall, men som forøvrig er like. Attributtene er beskrevet tidligere i seksjon 6.5.3.

Webserveren blir initialisert slik:

```
< WebServer1 : WebServer |
ssl: nil, sessionSet: emptySet , privKey: "KR",
received: nil, randomGenerator: rand(100),
tcp: nil , tcpSessions: emptySet, tcpReceived: nil,
cKey: symmKey(999999)  >
```

Det er mulig å opprette flere webservere i konfigurasjonen, men analysen i denne oppgaven begrenser seg til å se på kommunikasjon mellom en webserver og dens klienter slik at det ikke er laget noen likhet for å opprette webserveren. Servere blir derfor angitt manuelt i konfigurasjonsfilen.

I tillegg inneholder konfigurasjonen sertifikater, CRL, tidsobjekter og mengder for asymmetriske-nøkkelpar. Disse blir opprettet ved operatoren *initPKI* og disse objektene ble beskrevet i seksjon 6.2.

Selve initialtilstanden for SSL opprettes av operatoren *initSSL* som tar antall klienter vi ønsker i konfigurasjonen som parameter.

6.5.7 Omskriving

Ved å kjøre en *rew* commando på *initSSL* får vi ut en tilfeldig sluttilstand av SSL konfigurasjonen, som vist på figuren på neste side:

```
rewrite in CONFIG_SSL : initSSL(1) .
rewrites: 130 in 3ms cpu (2ms real) (43333 rewrites/second)
result Configuration:
<Time | tick: 4 >

<ValidKeyPairs | keySet: ("KU" & "KR") ;; ("KU*" & "KR*") >

<CRL | list: nil, signature: (sign hash(nil) with "KR*") >

<Log | headerErrorSSL: nil, authorizeError: nil >

< WebServer1 : WebServer | randomGenerator: 330237489, tcpSessions:
[commPartner: Client 1, seqnrOut: 16812, window: emptySet, seqnrIn:
    16811], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: Client 1, state: 4, sKey: symmKey(
    1906806022), mKey: symmKey(1906806023)], privKey: "KR",
    received: nil, ssl: nil, cKey: symmKey(999999) >

< Client 1 : Client | request: nil, randomGenerator: 984960465,
    tcpSessions:
[commPartner: WebServer1, seqnrOut: 16811, window: emptySet,
    seqnrIn: 16812], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: WebServer1, state: 4, sKey:
    symmKey(1906806022), mKey: symmKey(1906806023)],
    trustedIssuerSet: Issuer1, received: nil, ssl: nil, password:
    "password1", authCookie: null >

< Cert : Certificate | issuerName: Issuer1,subjectName: WebServer1,
    subjectKu: "KU",signature: sign hash(Cert :: Issuer1 ::
    WebServer1 :: "KU") with "KR*" >

< Cert* : Certificate | issuerName: Issuer1,subjectName: Issuer1,
    subjectKu: "KU*",signature: sign hash(Cert* :: Issuer1 ::
    Issuer1 :: "KU*") with "KR*" >
```

Her ser vi at de symmetriske nøklene er generert og like hos server og klient. Begge er også i `state: 4`, og det betyr at handshaket er ferdig, og partene er klare til å sende data over SSL Record protokollen.

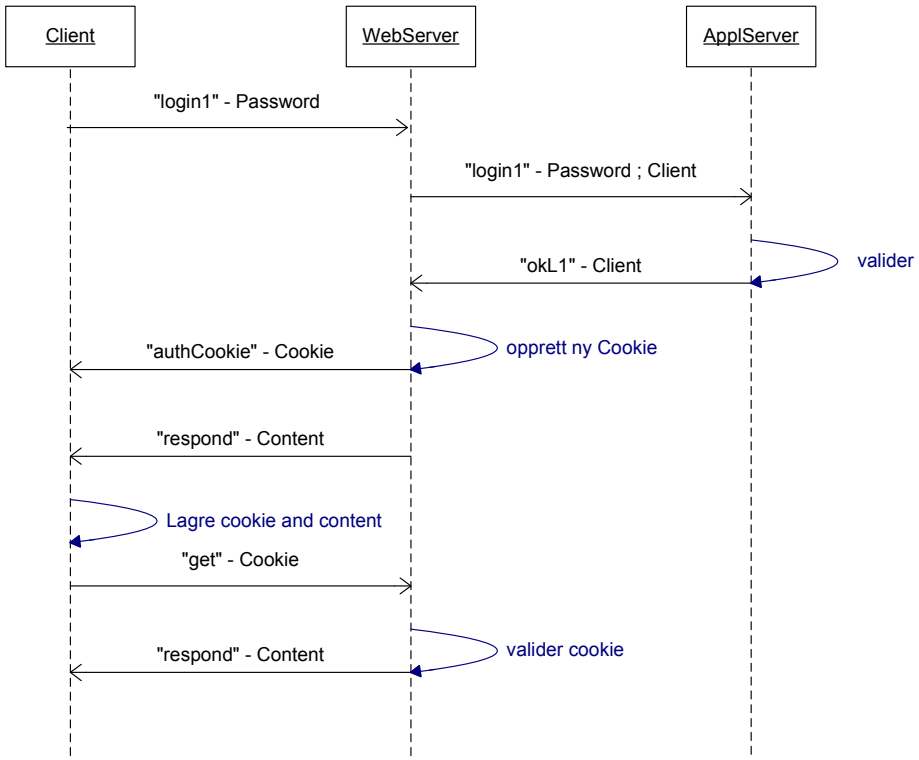
6.6 Login på tilgangsnivå 1

I filen LOGIN1.maude ligger modulen for innlogging til nivå 1 til Altinns webserver. Denne modulen importerer SSL-modulen, og alle meldinger som blir sendt behandles av SSL. Det er antatt av webserveren inneholder informasjon som kan sendes til brukeren på dette nivået i tillegg til informasjon som finnes i databasen. Det er her valgt å modellere innloggingsprosedyren basert på et statisk passord og ikke ved passord sent

via post. Dette fordi passord sendt via post ligner på prosedyren for tilgangsnivå to hvor passord blir sendt over SMS.

6.6.1 Normalt hendelsesforløp

Sekvensdiagrammet nedenfor viser et normalt hendelsesforløp slik det er modellert. De svarte pilene angir meldinger og de blå angir handlinger som foregår i objektene. Pilene for handlinger som er tegnet slik at de enkelte av de ender på samme linje som en melding, og det betyr at disse modellert innen den samme reglen og vil bli utført i ett steg.



Sekvensdiagram 2: Login1

Innlogging blir i implementasjonen trigget av at en klient velger "logg-inn" fra menyen. Login-prosedyren i modellen trigges tilsvarende når første element i request-attributtet er ekvivalent med ”login1”.

Det opprettes så en melding som legges i ssl-attributtet:

```
ssl: (L* :: (msg "login1" - PW to WS from CLIENT))
```

Variabelen `L*` er her listen over meldinger som skal sendes over SSL og den nye meldingen legges bakerst i listen. I datadelen av meldingen ligger

variabelen *PW* som inneholder passordet til klienten. Det forutsettes at innloggingsnavnet er det samme som klientens objektID, som her representeres av variabelen *CLIENT*, og dette trenger derfor ikke å sendes med i datadelen av meldingen. Dette er en rimelig antagelse siden vanlige brukere kun har én konto og identifiserer seg i den implementerte portalen med fødselsnummer.

SSL-modulen vil nå sørge for kryptering og integritetskontroll av meldingen. Webserveren vil starte sin login-prosedyre når den har mottatt meldingen, og denne ligger som første elementet serverens received-attributt. Webserveren sender nå en intern melding til applikasjonsserveren:

```
(aMsg "login1" - PW ; CLIENT  to AS from WS)
```

Applikasjonsserveren svarer ved å sjekke at *CLIENT* og *PW* variabelen er inneholdt i serverens mengde av gyldige par, bestående av påloggingsID og passord. Passord-parene er lagret i et *Set*, fordi hver bruker har sin unike ID og det vil derfor ikke kunne eksistere to like par. Hvis applikasjonsserveren godtar passord-paret vil den sende en intern melding tilbake til webserveren slik:

```
(aMsg "okL1" - CLIENT  to WS from AS)
```

Når webserveren mottar denne meldingen vil den sende en velkomstmelding til klienten, som er det første elementet serveren har i sitt *content*-attributt. Webserveren genererer også en autentiseringscookie ved bruk av funksjonen *authCookie* som ligger i TOOLS-filen. Hvordan cookien er bygget opp er beskrevet tidligere i seksjon 6.2.5. Det antas at webserveren har krypteringsnøkkelen til cookiene for å slippe ressursbruk ved å kontakte applikasjonsserveren hver gang. I modellen har vi også antatt at nøkkelinnholdet ikke kan knekkes. Nedenfor ser man hvordan webserveren oppretter en ny cookie når den aksepterer en forespørsel om tilgang til nivå1:

```
< WS : WebServer | ssl: L* , cKey: CKEY, ATTS >
=>
< WS : WebServer | ssl: (L* :: (msg "authCookie" -
authCookie(CLIENT , T , CKEY) to CLIENT from WS )),
cKey: CKEY, ATTS >
```

Her blir autentiseringscookien sendt over ssl og ikke som en klartekst-melding, og den blir opprettet av operatoren *authCookie* som ble omtalt i seksjon 6.2.5. Det er ønskelig å sende en cookie over SSL for å øke sikkerheten fordi cookies følger et bestemt format og er av den grunn sårbare for kryptoanalyse. Dette stemmer overens med en test vi har kjørt

på Altinns implementasjon hvor det ved hjelp av analyseringsverktøyet "Ethereal" [49] kom frem at cookies ble sendt over SSL.

6.6.2 Attributter og konfigurasjon

Login1 innfører en rekke nye attributter som legges i objektene for klienter og webservere:

- **authCookie**: attributt for klienter hvor de oppbevarer gjeldende cookie.
- **password**: brukes av klienten for autentisering på tilgangsnivå 1. Passordet blir opprettet i initialtilstanden og endres ikke i kjøring av modellen.
- **content**: liste over data som ligger direkte tilgjengelig på webserveren og beskyttes med tilgangsnivå 1.
- **passwordSet**: inneholder gyldige par av klientID og passord som brukes for autentisering på tilgangsnivå 1. Dette attributtet bruker *passwordSet* som ble beskrevet i seksjon 6.2.7.
- **cKey**: Symmetrisk nøkkel for kryptering av cookies. Nøkkelen kjennes kun av webserveren og er uforandret under kjøring. I en implementasjon vil denne være nødvendig å skifte ut jevnlig for å begrense skadeomfanget dersom uvedkommende får tak i nøkkelen. I modellen vil denne være konstant under kjøring.
- **validHeaders**: holder rede på hvilke headere som webserveren tillater, slik at meldinger som mottas må ivareta enkelte formateringskrav.

Konfigurasjonen utvider konfigurasjonen fra SSL med et nytt objekt som representerer applikasjonsserveren:

```
< ApplServer1 : ApplServer |
 passwordSet: initPWSet(N) >
```

Klient og webserver beholder sine gamle attributter og får i tillegg noen nye:

```
< Client N : Client |
password: pw(N), request: ("startTCP" :: "startSSL" ::
"login1" :: "get" ), obtainedContent: emptySet,
gamleAttributter >
```

Nytt i webserveren er at den nå inneholder informasjon som krever tilgangsnivå 1 for å kunne lese:

```
< WebServer1 : WebServer | content: ("Welcome-L1" ::
"Other-L1"), gamleAttributter >
```

6.6.3 Omskriving

Det er opprettet en operator kalt *initL1* som tar antall klienter som parameter og oppretter en initialtilstand for innlogging på tilgangsnivå 1 . Ved å kjøre en rewrite kommando på *initL1(N)* vil man derfor få en sluttilstand fra den gitte konfigurasjonen:

```
rewrite in CONFIG_LOGIN1 : initL1(1) .
rewrites: 230 in 5ms cpu (6ms real) (38339 rewrites/second)
result Configuration:
<Time | tick: 12 >

<ValidKeyPairs | keySet: ("KU" & "KR") ;; ("KU*" & "KR*") >

<CRL | list: nil, signature: (sign hash(nil) with "KR*") >

<Log | headerErrorSSL: nil, authorizeError: nil >

< WebServer1 : WebServer | validHeaders: ("get" ;; "login1"),
    randomGenerator: 330237489, tcpSessions:
[commPartner: Client 1, seqnrOut: 16815, window: emptySet, seqnrIn:
    16813], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: Client 1, state: 4, sKey: symmKey(
    1906806022), mKey: symmKey(1906806023)], privKey: "KR",
    received: nil, ssl: nil, cKey: symmKey(999999), content: (
    "Welcome-L1" :: "Other-L1") >

< ApplServer1 : ApplServer | passwordSet: (Client 1 & "password1") >

< Client 1 : Client | request: nil, randomGenerator: 984960465,
    tcpSessions:
[commPartner: WebServer1, seqnrOut: 16813, window: emptySet,
    seqnrIn: 16815], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: WebServer1, state: 4, sKey:
    symmKey(1906806022), mKey: symmKey(1906806023)],
    trustedIssuerSet: Issuer1, received: nil, ssl: nil,
obtainedContent: ((
        "Other-L1" from WebServer1) ;; (
        "Welcome-L1" from WebServer1)), password: "password1",
    authCookie: (sEncrypt authTicket(Client 1, 6, 66, 1) with
    symmKey(999999)) >

< Cert : Certificate | issuerName: Issuer1,subjectName: WebServer1,
    subjectKu: "KU",signature: sign hash(Cert :: Issuer1 ::
    WebServer1 :: "KU") with "KR*" >

< Cert* : Certificate | issuerName: Issuer1,subjectName: Issuer1,
    subjectKu: "KU*",signature: sign hash(Cert* :: Issuer1 ::
    Issuer1 :: "KU*") with "KR*" >
```

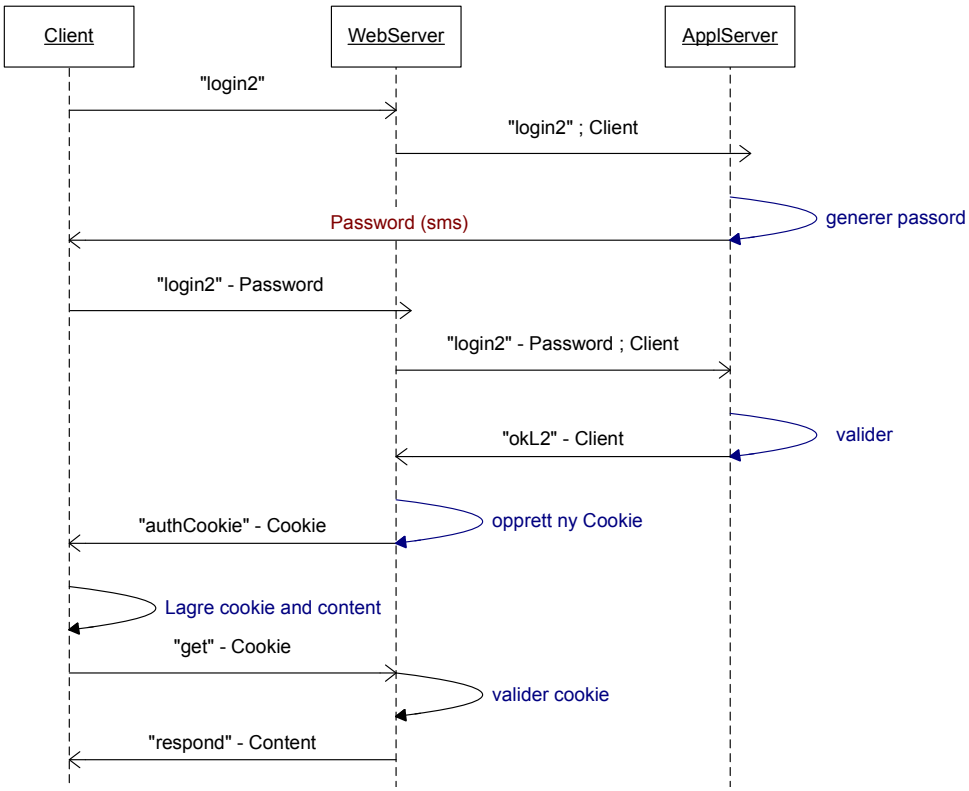
Klientens `obtainedContent`-attributt inneholder nå alle sidene som lå tilgjengelig med tilgangsnivå 1 på webserveren, og det betyr at klienten har kommet til en sluttilstand hvor den har foretatt en vellykket innlogging og hentet ut informasjon tilgjengelig på nivå 1 hos webserveren. Dette er resultatet vi ønsket komme frem til.

6.7 Login på tilgangsnivå 2

Modulen LOGIN2 inneholder regler for innloggingsprosedyren som blir foretatt når en bruker forsøker å logge seg på med et engangspassord som blir mottatt over SMS. En vellykket innlogging vil gi brukeren tilgang til informasjon på tilgangsnivå 2.

6.7.1 Normalt hendelsesforløp

Innloggingsprosedyren for nivå 2 starter på samme måte som innlogging på nivå 1, men for at webserveren skal vite hvilket tilgangsnivå klienten ønsker angis dette i meldingsheaderen. Sekvensdiagrammet nedenfor viser meldingsutveksling og handlinger for modulen LOGIN2. De svarte pilene angir meldinger og de blå handlinger.



Sekvensdiagram 3: Login2

Klienten starter med denne meldingen:

```
(msg "login2" to WS from CLIENT)
```

Når denne mottas hos webserveren vil den sendes videre internt til applikasjonsserveren for videre behandling. For at holde rede på hvem som har sendt forespørselen blir klientID lagt til meldingen. Applikasjonsserveren tar nå i bruk sin passordgenerator for å lage et nytt passord til klienten. Deretter tar den vare på klientID, passord, samt tidspunkt passordet vil utgå og tilslutt antall påloggingsforsøk fra denne klienten etter at passordet ble generert, som initieres til 0. Serveren sender også en melding til klienten over SMS:

```
(sms PW to CLIENT from WS)
```

Når denne er mottatt kan klienten fortsette innloggingen og sender passordet mottatt over SMS i en melding som sendes som vanlig over SSL:

```
(msg "login2" - PW to WS from CLIENT)
```

Denne mottas hos webserveren og videresendes til applikasjonsserveren som den er, med referanse til klienten.

```
(AMsg "login2" - PW ; CLIENT to AS from WS)
```

Applikasjonsserveren mottar denne, og sjekker ved mønstergjenkjenning at klientID og passord er med i settet av engangspassord. Det sjekkes også med betinget regel at gyldighetstidspunktet ikke er oversteget, og at det ikke er registrert mer enn tre påloggingsforsøk for denne klienten.

Er alt i orden signaliserer applikasjonsserveren at tilgang er innvilget ved å sende en ok-melding til webserveren:

```
(aMsg "okL2" - CLIENT to WS from AS)
```

Webserveren danner en ny autentiserings-cookie med tilgangsnivå 2 og sender denne og en velkomstmelding til klienten på samme måte som for LOGIN1. Det er derfor ikke laget noen ny regel for mottak av denne meldingen, da tidligere regel kan gjenbrukes. Dette skjer som før ved at mottatt cookie blir lagret, og meldingsinnhold blir sammen med webserverens ID lagret i klientens mengde av mottatt informasjon.

Tilslutt kan klienten be om å få oversendt informasjon som ligger direkte på webserveren på det nye autorisasjonsnivået. Dette tilsvarer navigeringssider som antas å ligge direkte på webserveren i implementasjonen.

6.7.2 Attributter og konfigurasjon

Denne modulen benytter de fleste attributtene som er opprettet tidligere, og inkluderer også følgende nye:

- **oneTimeSet**: Sett hvor engangspassordene blir lagret hos applikasjonsserveren.
- **content2**: Liste som inneholder all informasjon som er tilgjengelig direkte på webserveren, og som krever pålogging med tilgangsnivå 2.
- **passwordGenerator**: Attributt som tar et naturlig tall som brukes for å generere engangspassord.

Konfigurasjonen for denne modulen vil bli som for Login1, men utvidet med attributtene ovenfor.

6.7.3 Omskriving

Konfigurasjonsfilen for denne modulen inneholder operatoren *initL2* som tar antall klienter vi ønsker i initialtilstanden som attributt. Ved en normal omskrivning med 1 klient i initialtilstanden blir resultatet:

```
rewrite in CONFIG_LOGIN2 : initL2(1) .
rewrites: 231 in 6ms cpu (6ms real) (33004 rewrites/second)
result Configuration:
<Time | tick: 14 >

<ValidKeyPairs | keySet: ("KU" & "KR") ;; ("KU*" & "KR*") >

<CRL | list: nil, signature: (sign hash(nil) with "KR*") >

<Log | headerErrorSSL: nil, authorizeError: nil >

< WebServer1 : WebServer | validHeaders: ("get2" ;; "login2"),
    randomGenerator: 330237489, tcpSessions:
[commPartner: Client 1, seqnrOut: 16815, window: emptySet, seqnrIn:
    16814], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: Client 1, state: 4, sKey: symmKey(
    1906806022), mKey: symmKey(1906806023)], privKey: "KR",
    received: nil, ssl: nil, cKey: symmKey(999999), content2: (
    "Welcome-L2" :: "Other-L2") >

< ApplServer1 : ApplServer | passwordGenerator: 10006, oneTimeSet:
    emptySet >

< Client 1 : Client | request: nil, randomGenerator: 984960465,
    tcpSessions:
[commPartner: WebServer1, seqnrOut: 16814, window: emptySet,
    seqnrIn: 16815], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: WebServer1, state: 4, sKey:
    symmKey(1906806022), mKey: symmKey(1906806023)],
    trustedIssuerSet: Issuer1, received: nil, ssl: nil,
obtainedContent: ((
        "Other-L2" from WebServer1) ;; (
        "Welcome-L2" from WebServer1)), authCookie: (sEncrypt
    authTicket(Client 1, 8, 68, 2) with symmKey(999999)) >

< Cert : Certificate | issuerName: Issuer1,subjectName: WebServer1,
    subjectKu: "KU",signature: sign hash(Cert :: Issuer1 ::
    WebServer1 :: "KU") with "KR*" >

< Cert* : Certificate | issuerName: Issuer1,subjectName: Issuer1,
    subjectKu: "KU*",signature: sign hash(Cert* :: Issuer1 ::
    Issuer1 :: "KU*") with "KR*" >
```

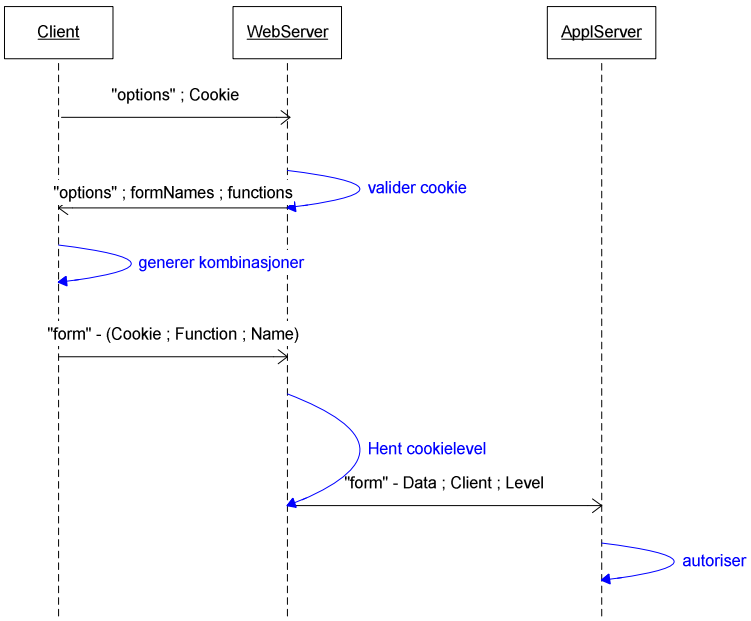
6.8 Skjemautveksling

Filen FORMEXCHANGE.maude inneholder modulen med samme navn og denne tar for seg hele hendelsesforløpet knyttet til skjemautveksling. Modulen baserer seg på at en klient først har logget seg på systemet med

tilgangsnivå 1 eller 2. Når klienten ønsker tilgang til et skjema ber den først om en skjemaoversikt. Deretter vil den forsøke å få tilgang til alle skjema den har mottatt i oversikten. Klienten blir autentisert ved hjelp av en autentiserings-cookie som gjøres ved hjelp av LOGIN1- eller LOGIN2-modulen. Tilslutt kreves autentisering mot databasen, og hvis tilgang blir innvilget vil skjemautvekslingen foregå.

6.8.1 Normalt hendelsesforløp for autentisering

En oversikt over det normale hendelsesforløpet i modellen, frem til databaseautorisasjon er vist i sekvensdiagrammet nedenfor:



Sekvensdiagram 4: skjema-oversikt

Når en klient ønsker å hente et skjema fra databasen vil den først be om å få oversendt en liste over alle skjema som er tilgjengelige og hvilke funksjoner som kan gjøres på disse, for eksempel lese og skrive. Klienten vil sende forespørsel om dette og legge ved sin autentiserings-cookie:

```
(msg "options" - CK to WS from CLIENT)
```

Når webserveren mottar denne meldingen vil den sjekke at cookien er gyldig på minst tilgangsnivå 1, og er den det, vil den svare med samme meldingsheader, samt to vedlagte sett som innholder skjemanavn og funksjoner. Dette representerer valgmulighetene en bruker får ved et besøk på Altinns sider, hvor brukeren kan klikke seg inn på forskjellige skjemaer og utføre forskjellige handlinger. For enklere å kunne søke etter ugyldige tilstander i modellen er det fordelaktig at en klient vil automatisk forsøke alle kombinasjoner av skjemaforespørsler. Vi har derfor opprettet likheter som generer alle mulige kombinasjoner av skjemanavn og funksjoner.

Deretter sender klienten én og én melding til webserveren med forespørsel om et skjema med en gitt funksjon til alle kombinasjonsmulighetene er sendt. Meldingene inneholder også en cookie for autentisering:

```
(msg "form" - CK ; E to WS from CLIENT)
```

Webserveren cookie-autentiserer denne på vanlig måte og sender deretter en intern melding til applikasjonsserveren for videre autentisering. I denne meldingen er cookien fjernet, men selve meldingsinnholdet blir videresendt med informasjon om hvem meldingen i utgangspunktet kom fra, og hvilket tilgangsnivå cookien hadde:

```
(aMsg "form" - FUNCTION ; NAME ; CLIENT ;
toString(LVL) to AS from WS)
```

Siden tilgangsnivå er definert som et naturlig tall, er den pakket inn i en operator som gjør den til en *String* under sending på samme måte som beskrevet i seksjon 6.5.4.

6.8.2 Attributter og konfigurasjon

Autorisasjonsprosedyren er modellert ved at applikasjonsserveren innholder tre tabeller representert ved sett i objektets attributter:

Webserveren og klient vil bli utvidet med følgende attributter:

- **formNames**: innholder alle skjemanavn som ligger lagret i databasen, og som klienten kan be om.
- **functions**: alle funksjoner som kan foretas på skjema, for eksempel
- **combinations**: brukes for å oppbevare alle mulige kombinasjoner av skjema og funksjoner en klient kan be om. Disse kombinasjonene blir opprettet under kjøring og klienten vil prøve alle mulighetene som er opprettet.

Applikasjonsserveren utvides med følgende attributter:

- **acl:** inneholder funksjon og sikkerhetsoperasjon.

- **roleList:** innholder sikkerhetsoperasjon, formNavn, Rolle og autentiseringsnivå.
- **userRoles:** Består av Objekt-id til de forskjellige klientene og deres roller.

Dette tilsvarer tabellene som er angitt i Altinns spesifikasjon og som er beskrevet tidligere i seksjon 4.4.

Database-objektet blir opprettet og vil ha følgende attributter:

- **forms**: består av et skjemanavn og det selve skjemaene som inneholder informasjon.
- **numbering**: brukes til å navngi nye skjema. Attributtet brukes både i generering av initialtilstanden og for å navngi skjemaer som dannes under kjøring.

Database-objektet vil se slik ut:

```
< Database1 : Database |forms: SET, numbering: N >
```

Det er opprettet likheter for automatisk å initialisere skjemanavn, rollelister og funksjoner. Det er også opprettet to konfigurasjonsfiler tilhørende denne modulen:

- CONFIG_FORMEXCH1 som inneholder en initialtilstand hvor klienten først vil logge seg på nivå 1 og deretter be om alle skjema.
- CONFIG_FORMEXCH2 som tilsvarer initialtilstanden ovenfor, men klienten vil her logge seg på med tilgangsnivå 2.

6.8.3 Normalt hendelsesforløp for autorisasjon

Følgende regel er selve hjertet i autorisasjonsprosessen når et skjema fra databasen er forespurt av en klient:

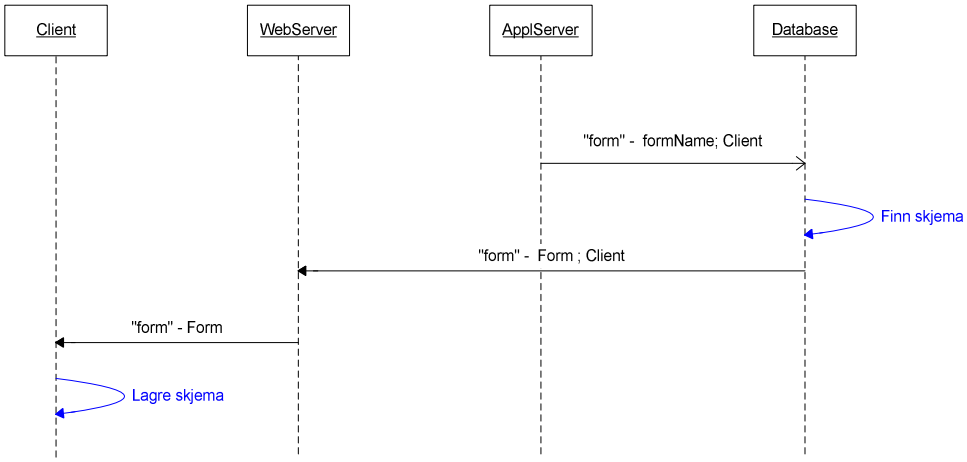
```
crl[authorize_formRequest]:
(aMsg "form" - FUNCTION ; NAME ; CLIENT ;
toString(LVL) to AS from WS)
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET),
roleList: ([ OP, NAME, ROLE, LVL*] ;;  SET*),
userRoles: ([CLIENT, ROLE] ;; S ),  ATTS* >
=>
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET),
roleList: ([ OP, NAME, ROLE, LVL*] ;;  SET*),
userRoles: ([CLIENT, ROLE] ;; S ),  ATTS* >
(if OP == "Read" then
(aMsg "form" -  NAME ; CLIENT to DB from AS )
else  (aMsg "form" - FUNCTION ; NAME ; CLIENT to DB
from AS ) fi)
if LVL* <= LVL  .
```

Denne regelen gjør bruk av mønstergjenkjenning for å binde sammen ACL, rolleliste og brukerroller. Dette representerer tabellene hentet fra Altinns dokumentasjon som er beskrevet i seksjon 4.4.

Når regelen utføres vil tilstandsendringen bestå i at applikasjonsserveren sender en melding videre til databasen, og meldingen vil få et ulikt format avhengig av om brukeren ønsker å lese eller utføre en annen operasjon på skjemaet.

6.8.4 Normalt hendelsesforløp for skjemautveksling

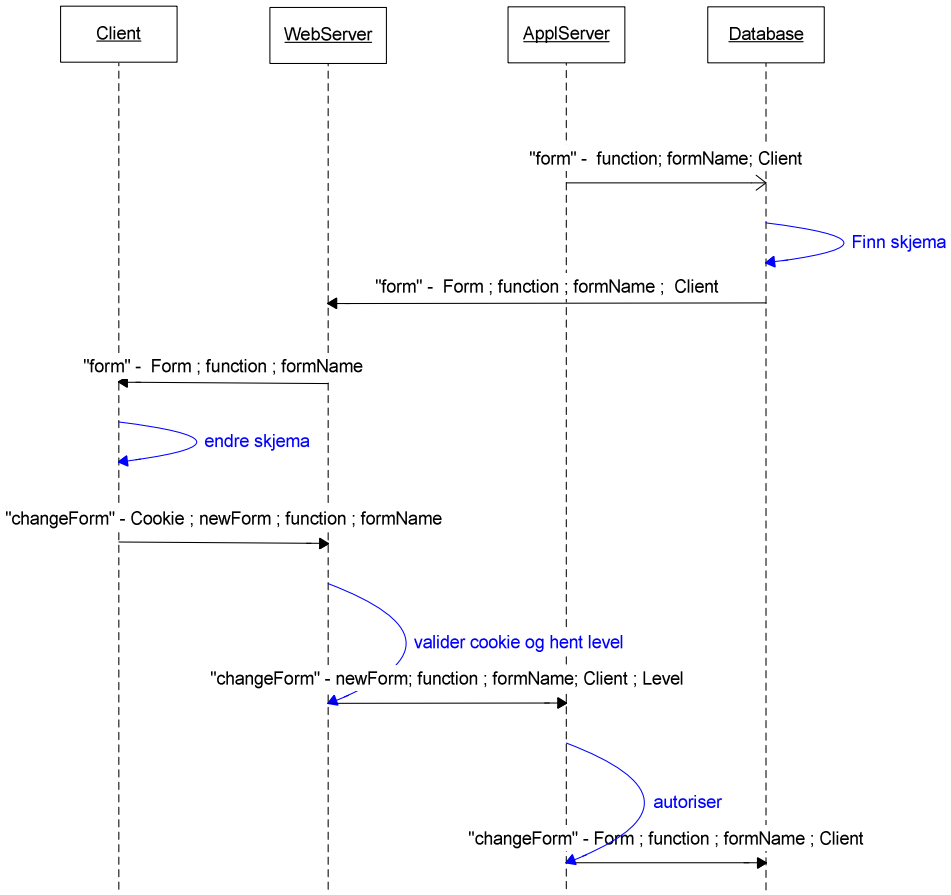
Det er to former for skjemautveksling. Den enkleste er når klienten ber om lesetilgang og alt databasen trenger å gjøre er å sende klienten en kopi. Hendelsesforløpet for dette er vist i seksvensdiagrammet nedenfor:



Sekvensdiagram 5: skjema-utveksling med lesetilgang

Diagrammet er en forlengelse av Sekvensdiagram 4 hvor applikasjons-serveren nettopp har autorisert klienten for database tilgang. Databasen slår opp i sin tabell over skjemanavn og vil ved hjelp av mønstergjenkjenning lokalisere riktig skjema og sende dette til klienten via web-serveren. Tilslutt vil klienten lagre meldingsinnholdet i `obtainedContent`.

Når klienten derimot ønsker å endre i et skjema må databasen også ta imot endringene og lagre disse. Normalt hendelsesforløp for dette er vist i det neste diagrammet:



Sekvensdiagram 6: skjema-utveksling med skrivetilgang

Dette diagrammet er også en fortsettelse av Sekvensdiagram 4 og klientforespørselen er allerede autentisert og autorisert. Meldingene skiller seg fra forespørslene om lesing, ved at her må også funksjonen som klienten ønsker å utføre være med siden det finnes flere måter å endre et skjema. I modellen er det tatt med to former for skjemaendring hvor enten et nytt skjema blir opprettet eller et eksisterende skjema blir overskrevet.

Når databasen mottar forespørselen vil den på samme måte som før finne riktig skjema ved hjelp av mønstergjenkjenning. Deretter sender den skjemaet med funksjon og skjemanavn som parametere til klienten via webserveren.

Klienten mottar skjema og vil endre dette på følgende måte:

```
(FORM + "_" + FUNCTION)
```

Her blir selve skjemaet lagt til en lav bindestrek og deretter funksjonen som er brukt for å endre. Dette er gjort for at man enkelt skal kunne se hvordan et skjema er endret når det blir foretatt omskrivninger og søk i modellen.

Det endrede skjemaet sendes nå tilbake til webserveren, sammen med autentiseringscookien, skjemanavn og funksjon som er benyttet. Webserver autentiserer som vanlig klienten ved hjelp av cookie-validering og henter så ut tilgangsnivået av cookien. Meldingen videresendes til applikasjonsserveren sammen med tilgangsnivået og klientID. Applikasjonsserveren vil ved mottak av denne meldingen autorisere klienten med tabelloppslag på samme måte som tidligere, for deretter å fjerne tilgangsnivået og videresende resten av meldingen.

Når databasen mottar en *changeForm*-melding vil den sjekke hvorvidt funksjonen som er medsendt er identisk med tekststrengen: "FormFilling". Hvis den er identisk vil databasen opprette et nytt skjema som vist i tilstandsendringen hentet fra koden:

```
< DB : Database | forms: ([NAME, FORM] ;; SET),
numbering: N , ATTS  >
=>
< DB : Database | forms: ([ (NAME + "." + string(N,
5)) , FORM*] ;;  [NAME, FORM] ;; SET),  numbering: (N
+ 1) , ATTS  >
```

Skjemaene vil få samme navn som tidligere etterfulgt av punktum og et nummer som inkrementeres hver gang et nytt skjema blir opprettet.

Hvis *changeForm*-meldingen derimot ikke er identisk med ”FormFilling” vil databasen anta at dette skjemaet skal oppdateres, og overskriver det eksisterende skjemaet med samme navn.

6.8.5 Omskriving

Ved innlogging med tilgangsnivå 1 vil sluttilstanden ved et normalt hendelsesforløp bli som følger:

```
rewrites: 428 in 12ms cpu (12ms real) (32928 rewrites/second)
result Configuration:
<Time | tick: 29 >

<ValidKeyPairs | keySet: ("KU" & "KR") ;; ("KU*" & "KR*") >

<CRL | list: nil, signature: (sign hash(nil) with "KR*") >

<Log | headerErrorSSL: nil, authorizeError: nil >
(aMsg "form" - "FormFilling" ; "RF2" ; Client 1 ; toString(1) to
    Appl Server1 from WebServer1
)
(aMsg "form" - "PDFViewer" ; "RF2" ; Client 1 ; toString(1) to
    Appl Server1 from WebServer1
)
(aMsg "form" - "UserAdministration" ; "RF1" ; Client 1 ; toString(1)
    to Appl Server1 from WebServer1
)
(aMsg "form" - "UserAdministration" ; "RF2" ; Client 1 ; toString(1)
    to Appl Server1 from WebServer1
)
(aMsg "form" - "theFormRF1" ; "FormFilling" ; "RF1" ; Client 1 to
    WebServer1 from Database1
)

< WebServer1 : WebServer | validHeaders: ("changeForm" ;; "form" ;;
    "get" ;; "login1" ;; "options"), randomGenerator: 330237489,
    tcpSessions:
[commPartner: Client 1, seqnrOut: 16817, window: emptySet, seqnrIn:
    16820], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: Client 1, state: 4, sKey: symmKey(
    1906806022), mKey: symmKey(1906806023)], privKey: "KR",
    received: nil, ssl: nil, cKey: symmKey(999999), content: (
    "Welcome-L1" :: "Other-L1"), formNames: ("RF1" ;; "RF2"),
    functions: ("FormFilling" ;; "PDFViewer" ;;
    "UserAdministration") >

< Appl Server1 : Appl Server | passwordSet: (Client 1 & "password1"),
    oneTimeSet: emptySet,
acl: (
        ["FormFilling","Write"] ;;
        ["PDFViewer","Read"] ;;
        ["UserAdministration","Admin"]),
roleList: (
        ["Read","RF1","Reader",1] ;;
        ["Read","RF2","Reader",2] ;;
        ["Write","RF1","Fillout",1]),
userRoles: (
        [Client 1,"Fillout"] ;;
        [Client 1,"Reader"]) >
```

```
< Database1 : Database |
forms: (
        ["RF1","theFormRF1"] ;;
        ["RF2","theFormRF2"]), numbering: 1 >

< Client 1 : Client | request: nil, randomGenerator: 984960465,
    tcpSessions:
[commPartner: WebServer1, seqnrOut: 16820, window: emptySet,
    seqnrIn: 16817], tcpReceived: nil, tcp: nil, sessionSet:
[1680700 : Session | commPartner: WebServer1, state: 4, sKey:
    symmKey(1906806022), mKey: symmKey(1906806023)],
    trustedIssuerSet: Issuer1, received: nil, ssl: nil,
obtainedContent: ((
        "Other-L1" from WebServer1) ;; (
        "Welcome-L1" from WebServer1) ;; (
        "theFormRF1" from WebServer1)), password: "password1",
    authCookie: (sEncrypt authTicket(Client 1, 6, 66, 1) with
    symmKey(999999)), formNames: ("RF1" ;; "RF2"), functions: (
    "FormFilling" ;; "PDFViewer" ;; "UserAdministration"),
    combinations: emptySet >

< Cert : Certificate | issuerName: Issuer1,subjectName: WebServer1,
    subjectKu: "KU",signature: sign hash(Cert :: Issuer1 ::
    WebServer1 :: "KU") with "KR*" >

< Cert* : Certificate | issuerName: Issuer1,subjectName: Issuer1,
    subjectKu: "KU*",signature: sign hash(Cert* :: Issuer1 ::
    Issuer1 :: "KU*") with "KR*" >
```

Vi ser her at klienten har mottatt et skjema fra databasen, og det er dette skjemaet klienten har tilgang til med påloggingsnivå 1.

Ved å logge inn med tilgangsnivå 2 vil sluttilstanden bli lik, men klienten vil få tilgang til mer data som blir lagret i klientens `obtainedContent` som vist på neste figur:

```
< Client 2 : Client | request: nil, randomGenerator: 1, tcpSessions:

[commPartner: WebServer1, seqnrOut: 15, window: emptySet, seqnrIn:
    12], tcpReceived: nil, tcp: nil, sessionSet:
[1 : Session | commPartner: WebServer1, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)], trustedIssuerSet: Issuer1, received: nil,
    ssl: nil,
obtainedContent: ((
        "Other-L2" from WebServer1) ;; (
        "Welcome-L2" from WebServer1) ;; (
        "theFormRF1" from WebServer1) ;; (
        "theFormRF2" from WebServer1)), authCookie: (sEncrypt
    authTicket(Client 2, 23, 83, 2) with symmKey(999999)),
    formNames: ("RF1" ;; "RF2"), functions: ("FormFilling" ;;
    "PDFViewer" ;; "UserAdministration"), combinations: emptySet >
```

Her ser vi at klienten har fått tilgang til enda et skjema som er tilgjengelig for tilgangsnivå 2, men ikke for 1. Dette er sluttilstanden vi ønsket å komme frem til.

7 Analyse av modellen

I dette kapittelet skal vi se på noen muligheter for analyse av Maude-modellen fra forrige kapittel, for å sjekke om modellen har de forventede egenskaper. Ved hjelp av Maudes innebygde søkeverktøy vil det være mulig å sjekke alle tilstander som kan nås fra en gitt initialtilstand i en Maude-modell.

7.1 SSL

Hovedformålet med SSL er å danne en sikker kommunikasjonskanal mellom en klient og en server. Under handshaket forhandler partene frem en felles sesjonsnøkkel etter først å ha autentisert hverandre, dersom dette er ønskelig. Hos Altinn og mange andre websteder er det kun webserveren som autentiserer seg under denne oppkoblingen.

For å sannsynliggjøre at vår modell av SSL danner en sikker kommuniksasjonskanal, vil vi gjennomføre følgende søk:

- Finnes det en tilstand hvor SSLs sesjonsnøkler blir sendt i klartekst?
- Finnes det ulike sluttilstander fra SSL-protokollen fra en gitt initialtilstand?
- Er det mulig å autentisere seg med et sertifikat med et utgått sertifikat, det vil si et som er registrert i CRL?
- Er det mulig å autentisere seg med et falskt sertifikat, og for eksempel utgi seg for å være en annen enhet?

7.1.1 Sesjonsnøkler

Modellen er laget slik at en nøkkel ikke kan være et `Data` eller `Header` element og dermed kan de ikke sendes som meldingsinnhold. Det er heller ikke opprettet noen konverteringsmulighet for symmetriske nøkler til tekst. Dette medfører at sikkerhet til nøkler er postulert i modellen. På en annen side er det mulig å konvertere tall til tekst, og derfor gjøres et søk for å se om dette kan føre til et sikkerhetsbrudd.

Vi er interessert i å se om det er mulig å komme i en tilstand hvor sesjonsnøkler blir sendt i klartekst, og det er derfor ikke nødvendig å finne

mer enn én slik tilstand for å bevise et sikkerhetshull. Vi søker derfor ikke etter alle slike tilstander, men ber Maude om å stoppe søket straks en slik tilstand er oppdaget.

For å søke gjennom MsgContent er det for dette søket opprettet en egen modul kalt SEARCH med følgende operatorer:

```
op _contains_ : MsgContent String -> Bool .
op _contains_ : DataList String -> Bool .
```

Videre er det opprettet tre likheter slik at MsgContent blir rekursivt gjennomløpt på denne måten:

```
eq (H - DATALIST) contains SSET = if H == SSET then
true else (DATALIST contains SSET) fi .

eq ( DATA ; DATALIST ) contains SSET = if DATA == SSET
then true else ( DATALIST contains SSET) fi .

eq (DATA contains SSET) =  if DATA == SSET then true
else false fi .
```

Disse tre likhetene sørger for at både header og data-elementer blir sammenlignet med tekststrengen som vi ønsker å finne. Delstrenger vil ikke bli undersøkt, og søk etter *"password"* vil ikke matche med hverken *"word"* eller *(sEncrypt "password" with Key)*. Med andre ord så vil modulen kun returnere med true dersom det finnes et ukryptert element i angitte melding som er identisk med søkeordet.

Søket for å sjekke om vi kan komme i en tilstand hvor nøkkelverdier blir sendt i klartekst blir nå som følger:

```
search [1] in CONFIG_SSL : initSSL(2) =>* C
(msg MSG to A from B)

< B : U:Unit | ATTS*, sessionSet: (SET ;;
[SID:Sid : Session | ATTS, sKey: symmKey(N:Nat)]) > such that MSG
    contains toString(N:Nat) = true .

No solution.
states: 120658  rewrites: 615657 in 37710ms cpu (138706ms real) (
    16325 rewrites/second)
```

Vi har nå vist at det *ikke* er mulig å komme til en tilstand i modellen hvor nøkkelen sendes i klartekst i en melding, og det var dette resultatet vi ønsket å få fra dette søket.

7.1.2 Sluttilstander

Det er interessant å undersøke sluttilstander, for å sjekke at det finnes hendelsesforløp som ikke samsvarer med ønsket handlingsmønster for modellen.

For å finne alle mulige sluttilstander på grunnlag av initaltilstanden til SSL lar vi Maude utføre følgende kommando:

```
search initSSL(N) =>! C:Configuration .
```

Ved søk etter alle mulige sluttilstander med én klient i initialtilstanden får vi én løsning. Det impliserer at det *ikke finnes noen annen løsning* enn den samme som beskrevet tidligere i seksjon 6.5.7.

Oppretter vi to klienter i initialtilstanden vil vi få 88 forskjellige sluttilstander som skiller seg fra hverandre ved at de har forskjellige random-verdier og følgelig også forskjellige nøkkelverdier. Desto flere klienter vi legger i konfigurasjonen desto flere sluttilstander får vi og søkene blir raskt for store til å gi et svar innen rimelig tid. Ved tre klienter i konfigurasjonen blir tilstandsrommet så stort at etter et par timer har søket fremdeles ikke terminert.

7.1.3 Begrensning av tilstandsgenerering

Det er nødvendig å foreta en begrensning av tilstandsgenereringen for videre søking, slik at vi får en sluttilstand også for konfigurasjoner hvor det er mer en klient. Siden generering av tilfeldige tall ikke har betydning for søk som ikke omhandler nøkkelverdier, kan vi modifisere denne før videre søking. Med små endringer i koden kan man sette random-generatoren ut av spill ved å la den returnere en statisk verdi. Vi har her valgt å la generatoren alltid returnere 1. Søk etter alle sluttilstander med to klienter i konfigurasjonen gir oss nå følgende resultat:

```
Maude> search initSSL(2) =>! C:Configuration .
search in CONFIG_SSL : initSSL(2) =>! C .

Solution 1 (state 7421)
states: 7422  rewrites: 41351 in 3098ms cpu (7741ms real) (13345
    rewrites/second)
C -->
<Time | tick: 8 >

<ValidKeyPairs | keySet: ("KU" & "KR") ;; ("KU*" & "KR*") >

<CRL | list: nil, signature: (sign hash(nil) with "KR*") >

<Log | headerErrorSSL: nil, authorizeError: nil >

< WebServer1 : WebServer | randomGenerator: 1, tcpSessions: (
[commPartner: Client 1, seqnrOut: 6, window: emptySet, seqnrIn: 5]
    ;;
[commPartner: Client 2, seqnrOut: 6, window: emptySet, seqnrIn: 5]),
    tcpReceived: nil, tcp: nil, sessionSet: (
[1 : Session | commPartner: Client 1, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)] ;;
[1 : Session | commPartner: Client 2, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)]), privKey: "KR", received: nil, ssl: nil,
    cKey: symmKey(999999) >

< Client 1 : Client | request: nil, randomGenerator: 1, tcpSessions:

[commPartner: WebServer1, seqnrOut: 5, window: emptySet, seqnrIn:
    6], tcpReceived: nil, tcp: nil, sessionSet:
[1 : Session | commPartner: WebServer1, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)], trustedIssuerSet: Issuer1, received: nil,
    ssl: nil, password: "password1", authCookie: null >

< Client 2 : Client | request: nil, randomGenerator: 1, tcpSessions:

[commPartner: WebServer1, seqnrOut: 5, window: emptySet, seqnrIn:
    6], tcpReceived: nil, tcp: nil, sessionSet:
[1 : Session | commPartner: WebServer1, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)], trustedIssuerSet: Issuer1, received: nil,
    ssl: nil, password: "password2", authCookie: null >

< Cert : Certificate | issuerName: Issuer1,subjectName: WebServer1,
    subjectKu: "KU",signature: sign hash(Cert :: Issuer1 ::
    WebServer1 :: "KU") with "KR*" >

< Cert* : Certificate | issuerName: Issuer1,subjectName: Issuer1,
    subjectKu: "KU*",signature: sign hash(Cert* :: Issuer1 ::
    Issuer1 :: "KU*") with "KR*" >

No more solutions.
```

En begrensning som fjerning av random-elementer medfører er at alle klienter får samme nøkler og sesjonsID, men nå returnerer søket kun en sluttilstand slik vi ønsket. Vi vet nå at uansett i hvilken rekkefølge meldinger blir behandlet så vil begge klientene lykkes i å starte en sesjon med serveren.

Selv med random-generatoren satt ut av spill vil tilstandsrommet fort bli stort da meldinger kan sendes og mottas i forskjellige rekkefølger. Søkene som blir foretatt i følgende kapitler er derfor begrenset til to klienter i konfigurasjonen med mindre annet er oppgitt. To klienter vil ofte være tilstrekkelig for å dekke inn kjente sikkerhetsproblemer som for eksempel man-in-the-middle.

7.1.4 Utgått sertifikat

Ved å endre initialtilstanden slik at CRL-en inneholder sertifikatID til webserver skal autentiseringsprosessen feile. For å verifisere dette ble følgende endret i konfigurasjonen:

```
<CRL | list: Cert , signature: (sign hash(Cert) with
"KR*") >
```

Ved en rewrite får vi følgende output for klient og server objektet:

```
< WebServer1 : WebServer | randomGenerator: 1, tcpSessions: [
    commPartner: Client 1, seqnrOut: 4, window: emptySet, seqnrIn:
    2], tcpReceived: nil, tcp: nil, sessionSet: [1 : Session |
    commPartner: Client 1, handshakeMsgs: ((msg "client_hello" -
    toString(1) to WebServer1 from Client 1) :: (msg "server_hello" -
    toString(1) to Client 1 from WebServer1) :: (msg "certificate" -
    Cert to Client 1 from WebServer1) :: msg "server_hello_done" to
    Client 1 from WebServer1), state: 0, keyMaterial: 2], privKey:
    "KR", received: nil, ssl: nil, cKey: symmKey(999999) >


< Client 1 : Client | request: nil, randomGenerator: 1, tcpSessions:
    [commPartner: WebServer1, seqnrOut: 2, window: emptySet, seqnrIn:
    4], tcpReceived: ((msg "server_hello" - toString(1) to Client 1
    from WebServer1) :: (msg "certificate" - Cert to Client 1 from
    WebServer1) :: msg "server_hello_done" to Client 1 from
    WebServer1), tcp: nil, sessionSet: [0 : Session | commPartner:
    WebServer1, handshakeMsgs: (msg "client_hello" - toString(1) to
    WebServer1 from Client 1), state: 0, keyMaterial: 1],
    trustedIssuerSet: Issuer1, received: nil, ssl: nil >
```

Vi ser at det oppstår en feil i handshaket og både klienten og serveren stopper opp i med attributtet state satt til 0.

For å sjekke om dette gjelder for alle tilstander undersøkes det om det finnes en tilstand hvor handshaket har gått videre i protokollen. Det vil si vi lager et søk hvor vi leter etter tilstander hvor handshaket har kommet forbi `state` 0. Søket blir da som følger:

```
search in CONFIG_SSL : initSSL(2) =>* C

< A : U:Unit | ATTS*, sessionSet: (SET ;;
[S:Sid : Session | ATTS, state: N:Nat]) > such that N:Nat > 0 = true
    .

No solution.
states: 761  rewrites: 4839 in 480ms cpu (875ms real) (10061
    rewrites/second)
```

Ut i fra dette ser vi at modellen *ikke* kan komme videre i handshaket når webserveren er registrert i CRL.

7.1.5 Falskt sertifikat

En inntrenger kan forsøke å lage falske sertifikater og dette kan gjøres på forskjellig nivå i sertifikat-hierakiet. I modellen har dette hierarkiet to nivåer slik at man kan forfalske enten serverens sertifikat eller CA-ens sertifikat. Sertifikatet kan forfalskes ved at en angriper bytter ut den offentlige nøkkelen i et allerede eksisterende sertifikat.

Et falskt sertifikat hvor angriper forsøker å utgi seg for å være server kan representeres i initialtilstanden på følgende måte:

```
< Cert : Certificate | issuerName: Issuer1,
subjectName: WebServer1, subjectKu: "adversarys_Key",
  signature: sign hash(Cert :: Issuer1 :: WebServer1
:: "KU") with "KR*" >
```

Her har inntrenger lagt sin nøkkel i feltet for offentlig nøkkel og forøvrig beholdt sertifikatet slik det var.

Ved å gjenbruke søket fra forrige kapittel undersøker vi nå om handshaket kan komme seg forbi tilstand 0:

```
search in CONFIG_SSL : initSSL(2) =>* C

< A : U:Unit | ATTS*, sessionSet: (SET ;;
[S:Sid : Session | ATTS, state: N:Nat]) > such that N:Nat > 0 = true
    .

No solution.
states: 36  rewrites: 149 in 18ms cpu (54ms real) (7843
    rewrites/second)
```

Vi har bekreftet at modellen *ikke* tillater autentisering uten et gyldig sertifikat.

Et falskt sertifikat for CA kan opprettes på tilsvarende måte og gir det samme søkeresultatet, og vi har fått bekreftet at det ikke er mulig for en angriper i modellen å personifisere serveren ved hjelp av et falske sertifikater.

7.2 Login på tilgangsnivå 1

Modulen kalt LOGIN1 inneholder regler som gjenspeiler pålogging med brukernavn og passord for å få tilgang til informasjon tilgjengelig på autorisasjonsnivå 1.

Følgende problemområder vil bli belyst under analysen av denne modulen:

- Finnes det en tilstand hvor en klient med gyldig passord ikke får logget seg på?
- Finnes det en tilstand hvor passord blir sendt i klartekst?

7.2.1 Pålogging

For å sjekke om det finnes ugyldige tilstander i modellen ved innlogging på tilgangsnivå 1, er det først foretatt et søk for å se om det finnes tilstander hvor klienten ikke får logget seg på, og det kan gjøres på følgende vis:

```
search [1] in CONFIG_LOGIN1 : initL1(2) =>! C

< CLIENT : U:Unit | ATTS,
obtainedContent: SET > such that not SET contains (
        "Welcome-L1" from WebServer1) = true .

No solution.
states: 815086  rewrites: 4532209 in 778260ms cpu (793810ms real)
    (5823 rewrites/second)
```

Et søk etter sluttilstander som ikke resulterer i vellykket pålogging gir ingen løsninger og det impliserer at det ikke finnes noen tilstand hvor klienten ikke lykkes i å logge seg på webserveren med tilgangsnivå 1.

7.2.2 Passord

Det er ønskelig å undersøke om passordet blir sent i klartekst da dette er informasjon som ikke må komme på avveie. Søket benytter seg av SEARCH-modulen som ble beskrevet i seksjon 7.1.1.

```
search in CONFIG_LOGIN1 : initL1(2) =>* C
msg MSG to A from B such that MSG contains "password1" = true .

No solution.
states: 815086  rewrites: 4530865 in 4062400ms cpu (4153160ms
    real) (1115 rewrites/second)
```

Søket gir ingen tilstander som oppfyller søkekriteriene. Det betyr at modellen *ikke* lar passord bli sendt i klartekst og dette er hva vi ønsket å bekrefte.

7.3 Login på tilgangsnivå 2

LOGIN2 ligner modulen for innlogging på nivå 1, men noen regler skiller disse ettersom denne modulen benytter seg av engangspassord sendt per SMS og gir tilgang til et høyere nivå. Tilstandene vi ønsker å søke etter her blir de samme som for innlogging på nivå 1:

- Finnes det en tilstand hvor en klient med gyldig passord ikke får logget seg på?
- Finnes det en tilstand hvor passord blir sendt i klartekst?

7.3.1 Pålogging

Det er også ved innlogging på nivå 2 interessant å undersøke om det i modellen finnes en tilstand hvor en klient med gyldig passord ikke får logget seg på. Søket blir tilsvarende som for innlogging på nivå 1, og man søker etter en tilstand hvor klienten ikke har mottatt velkomstmeldingen for det aktuelle tilgangsnivået. Søket blir som følger:

```
search [1] in CONFIG_LOGIN2 : initL2 =>! C

< CLIENT : U:Unit | ATTS,
obtainedContent: SET >
 such that not SET contains (
        "Welcome-L2" from WebServer1) = true .

No solution.
states: 279  rewrites: 909 in 80ms cpu (80ms real) (11362
    rewrites/second)
```

Her ser vi at søket returnerer uten løsning og det betyr at det ikke finnes noen tilstand, hvor en gyldig klient ikke får logget seg på dette tilgangsnivået.

7.3.2 Passord

Vi ønsker å kontrollere at passordet ikke blir sendt i klartekst. Modulen LOGIN2 genererer passordet ved hjelp av en random-generator slik at passordet her ikke er kjent på forhånd. Vi vet at passordet kun blir sendt fra klienten sammen med headeren ”get2” og det er derfor dette vi søker etter. SEARCH modulen fra seksjon 7.1.1 blir brukt får å sjekke meldingsinnhold.

```
search in SEARCH : initL2 =>* C:Configuration
msg MSG to A:Oid from B:Oid such that MSG contains "get2" = true .

No solution.
states: 251  rewrites: 1776 in 80ms cpu (80ms real) (22200
    rewrites/second)
```

Søket gir ingen tilstander. Dermed vet vi at slik at vi vet at passordet alltid blir sendt kryptert. Det betyr at *ingen* uvedkommende kan avlytte meldingen og bruke passordet for å personifisere klienten.

7.4 Skjemautveksling

Etter vellykket autentisering på nivå 1 eller 2 kan brukeren aksessere skjema. Modulen for skjemautveksling sørger for å gi brukeren en oversikt over hvilke skjema som kan hentes og hvilke funksjoner som kan utføres på dem. Deretter blir brukere autorisert ved hjelp av ACL, rolleliste og brukerroller.

Spørsmålet som ønskes besvart i denne seksjonen blir som følger:

- Kan en klient få tilgang til et skjema den ikke er autorisert for?

- Finnes det en tilstand hvor en klient ikke får tilgang til et skjema den er autorisert for?

Disse spørsmålene kan delvis besvares ved å undersøke om det er mulig å få flere sluttilstander. Allerede ved første søk får vi problemer med stor tilstandsgenerering, og søket returnerer ikke noe svar innen rimelig tid. Vi forsøker derfor å redusere antall skjema og funksjoner som kan utføres på dem. Etter å ha endret initaltilstanden vil autoriseringsskjemaene se slik ut:

```
< ApplServer1 : ApplServer | passwordSet: (Client 1 & "passwor
    emptySet,
acl: (
        ["FormFilling","Write"] ;;
        ["PDFViewer","Read"]),
roleList: (
        ["Read","RF1","Reader",1] ;;
        ["Read","RF2","Reader",2]),
userRoles: (
        [Client 1,"Fillout"] ;;
        [Client 1,"Reader"]) >



< Database1 : Database |
forms: (
        ["RF1","theFormRF1"] ;;
        ["RF2","theFormRF2"]), numbering: 1 >
```

Ett nytt søk med den nye initialtilstanden og én klient i konfigurasjonen får i følgende:

```
> search initF1 =>! C:Configuration .
search in CONFIG_FORMEXC1 : initF1 =>! C .

Solution 1 (state 1197008)
states: 1197009  rewrites: 8388143 in 1807740ms cpu (
    3728870ms real) (4640 rewrites/second)
```

Vi ser at det er foretatt svært mange omskrivninger og det tar ca. 30 minutter å få svar på dette søket med tilgjengelig maskinvare. Det returnerer med kun én løsning, og det er dette som er ønskelig. Ved mer enn én klient i konfigurasjonen vil søket ikke gi noe svar innen rimelig tid. Det blir derfor nødvendig å forsøke å redusere tilstandsgenereringen slik at vi kan få resultater når vi søker i disse modulene. Skjemautveksling med tilgangsnivå 2 vil heller ikke resultere i noe svar.

7.4.1 Videre begrensning av tilstandsgenerering

Ytterligere begrensninger i tilstandsgenerering må foretas for å kunne søke videre. Ved å se gjennom koden etter steder hvor det kan foretas en innsnevring ser vi at TCP-protokollen tilstandsendringene i TCP-protokollen ikke trenger å defineres som regler, men vi kan anta at dette laget utfører sin misjon i ett steg. Vi gjør alle reglene i modulen til likheter.

For eksempel vi denne reglen:

```
rl[tcp_sort_messages]:
< A : U | tcpSessions: ([window: ((msg toString(N) -
MSG to A from B) ;; SET),seqnrIn: (N), ATTS*] ;;
SET*), tcpReceived: L, ATTS >
=>
 < A : U | tcpSessions: ([window: SET,
seqnrIn: (N + 1 ), ATTS*] ;; SET*), tcpReceived: (L ::
(msg MSG to A from B)), ATTS > .
```

Den blir endret slik at den ser slik ut:

```
eq
< A : U | tcpSessions: ([window: ((msg toString(N) -
MSG to A from B) ;; SET),seqnrIn: (N), ATTS*] ;;
SET*), tcpReceived: L, ATTS >
=
< A : U | tcpSessions: ([window:  SET, seqnrIn: (N +
1), ATTS*] ;; SET*), tcpReceived: (L :: (msg MSG to A
from B)) , ATTS > .
```

Prosessen med å først motta meldinger og siden sortere dem, er nå slått sammen og utføres steg. For å kontrollere om dette hadde ønsket effekt på tilstandsgenerering ble det utført søk etter alle sluttilstander for SSL-modulen, med 2 klienter i konfigurasjonen:

```
search initSSL(2) =>! C:Configuration .
```

Omskrivninger foretatt med den opprinnelige TCP-modulen importert ble som følger:

```
states: 7422  rewrites: 41351 in 4340ms cpu (6530ms real) (9527
    rewrites/second)
```

Det samme søket ble foretatt på nytt med den modifiserte TCP-modulen og dette ga følgende resultat:

```
states: 150  rewrites: 2271 in 110ms cpu (100ms real) (20645
    rewrites/second)
```

Vi ser det er en markant nedgang i antall omskrivninger, og vil benytte den modifiserte TCP-protokollen for videre søk.

7.4.2 Søk etter sluttilstander

Med tilstandsreduksjonene som beskrevet gjøres et nytt forsøk på søk i skjemautvekslingsmodulene. Det viser seg at det heller ikke nå er mulig å få et søkeresultat når man søker etter sluttilstander med mer enn én klient i konfigurasjonen. Tilstandsbegrensningen fører til at alle søk går raskere, og vi får også resultat når vi gjør søk i skjemautveksling med påloggingsnivå 2. Søkene i denne modulen blir derfor foretatt med kun en klient. Første søket blir da som følger:

```
search initF1(1) =>! C:Configuration.
```

Skjema-utveksling med nivå1 vil nå resultere i en løsning med resultat for klienten som vist nedenfor:

```
< Client 1 : Client | request: nil, randomGenerator: 1, tcpSessions:
[commPartner: WebServer1, seqnrOut: 12, window: emptySet, seqnrIn: 11],
    tcpReceived: nil, tcp: nil, sessionSet:
[1 : Session | commPartner: WebServer1, state: 4, sKey: symmKey(3), mKey:
    symmKey(4)], trustedIssuerSet: Issuer1, received: nil, ssl: nil,
obtainedContent: ((
        "Other-L1" from WebServer1) ;; (
        "Welcome-L1" from WebServer1) ;; (
        "theFormRF1" from WebServer1)), password: "password1", authCookie: (
    sEncrypt authTicket(Client 1, 5, 65, 1) with symmKey(999999)), formNames: (
    "RF1" ;; "RF2"), functions: ("FormFilling" ;; "PDFViewer"), combinations:
    emptySet >
```

Dette viser at det ikke er mulig å finne noen sluttilstand hvor klienten ikke får lastet ned de skjema som den har tilgang til. Det samme søket viser også at det ikke finnes noen løsning der klienten urettmessig har fått tilgang til et skjema. Dette er resultatet vi ønsket å komme frem til ved hjelp av dette søket. Tilstandsgenerering er med den nye versjonen av TCP-modulen også som forventet mye lavere:

```
states: 2428  rewrites: 25754 in 1299ms cpu (302826ms real) (19813
    rewrites/second)
```

Vi utfører samme søket på skjemautveksling hvor klienten logger seg på med tilgangsnivå 2 og får følgende resultat:

```
< Client 1 : Client | request: nil, randomGenerator: 1, tcpSessions:
[commPartner: WebServer1, seqnrOut: 13, window: emptySet, seqnrIn:
    12], tcpReceived: nil, tcp: nil, sessionSet:
[1 : Session | commPartner: WebServer1, state: 4, sKey: symmKey(3),
    mKey: symmKey(4)], trustedIssuerSet: Issuer1, received: nil, ssl:
    nil,
obtainedContent: ((
        "Other-L2" from WebServer1) ;; (
        "Welcome-L2" from WebServer1) ;; (
        "theFormRF1" from WebServer1) ;; (
        "theFormRF2" from WebServer1)), authCookie: (sEncrypt
    authTicket(Client 1, 6, 66, 2) with symmKey(999999)), formNames:
    ("RF1" ;; "RF2"), functions: ("FormFilling" ;; "PDFViewer"),
    combinations: emptySet >
```

Her har vi igjen kun én sluttilstand hvor klienten har mottatt de skjemaene den har tilgang til. Vi har nå vist at modellen oppfyller tilgjengelighetskravet for klienter med de begrensninger som er foretatt i autorisasjonsskjemaet.

Ved søk etter sluttilstander for begge tilgangsnivå, hvor man har mer enn en klient i konfigurasjonen returnerer ikke søkene noe svar innen rimelig tid. Dette betyr at tilstandsrommet blir stort, og hvis modellen ikke inneholder uendelige løkker så vil systemet returnere svar gitt bedre maskinressurser og/eller tid. Det er forventet, at så sant systemet terminerer, vil søkeresultatet gi oss én sluttilstand. Dette forventes fordi søk med én klient gir oss dette resultatet og det er ikke avdekket noen svakheter i de underliggende modulene. Gitt det ønskede resultatet vil en klient *alltid* få tilgang til skjema den er autorisert for og *aldri* få tilgang til skjema den ikke er autorisert for ved de begrensningene som er satt av initialtilstanden.

7.5 En angriper i modellen

For å kunne foreta en analyse av hvorvidt en angriper kan utnytte systemet må man innføre en ny klient som ikke oppfører seg i henhold til forventet oppførsel. Det vil si at denne klienten ikke vil følge normalt hendelsesforløp og kan utføre en rekke operasjoner i tillegg til de en vanlig klient kan utføre. En angriper vil for eksempel kunne avlytte meldinger som sendes over Internett og spoofe andre klienter.

I modulen ADVERSARY opprettes en slik klient. Vi lar inntrengeren snappe opp alle meldinger som sendes over Internett, også den sender og mottar selv. Avlesning av meldingene skjer ved en likhet for å unngå ytterligere tilstandsgenerering.

```
ceq < Adversary : Client | msgIntercepted: SET, ATTS >
(msg MSG to A from B )
=
< Adversary : Client | msgIntercepted: (SET ;; (msg
MSG to A from B )), ATTS >
(msg MSG to A from B )
if not(SET contains (msg MSG to A from B )) .
```

Det er også opprettet en likhet hvor inntrengeren spoofer slik at det ser ut som meldinger den har overhørt ser ut til å komme fra klienten:

```
ceq
< Adversary : Client | msgIntercepted: ((msg MSG to A
from B ) ;; SET), msgSpoofed: SET*, ATTS >
(msg MSG to A from B )
< CLIENT : Client | ATTS* >
=
< Adversary : Client | msgIntercepted: ((msg MSG to A
from B ) ;; SET),msgSpoofed:( (msg MSG to A from B) ;;
SET*), ATTS >
(msg MSG to A from B )
(msg MSG to A from CLIENT )
< CLIENT : Client | ATTS* >
if not(SET* contains (msg MSG to A from B)) and not (
A == Adversary)  .
```

Her ser vi at hver melding som inntrenger har avlyttet blir spoofet en gang. Det er valgt å ikke fjerne innholdet i *msgIntercepted* etter hvert som meldingene spoofes, selv om dette hadde gitt kortere kode. Modulen er laget med tanke på utvidelse slik at *msgIntercepted* senere kan brukes som grunnlag for eksempel modifisering av meldinger.

Vi kan nå utføre et søk med en angriper i systemet, og søke etter tilstander hvor angriperen har fått tak i informasjon den ikke er autorisert for.

```
search[1] initA =>! C:Configuration
< Adversary:Oid : U:Unit | msgIntercepted: SET:Set,
obtainedContent: SET*:Set, ATTS:AttributeSet >
 such that SET:Set contains "Welcome-L1" or SET*:Set
contains "Welcome-L1" or  SET:Set contains "Other-L1"
or SET*:Set  contains "Other-L1"  .
```

Søkeresultatet fra dette søket blir slik:

```
search [1] in ADVERSARY : initA =>! C:Configuration

< Adversary:Oid : U:Unit | ATTS,
obtainedContent: SET*,
msgIntercepted: SET > such that ((SET contains "Other-L1" or SET*
    contains "Other-L1") or SET* contains "Welcome-L1") or SET
    contains "Welcome-L1" = true .

No solution.
states: 14713  rewrites: 7536739 in 43770ms cpu (44990ms real) (172189
    rewrites/second)
```

Her ser vi at Maude har lykkes i å søke gjennom alle mulige tilstander vi kan komme til fra initaltilstanden og ikke funnet noen tilstand hvor en inntrenger urettmessig har fått tak i informasjon på tilgangsnivå 1. Vi har nå bevist at modellen for innlogging med tilgangsnivå 1 ikke er sårbar for avlytting og spoofing, der spoofingen er begrenset til å kun spoofe hver melding én gang, og meldingsinnholdet er uendret.

Vi utfører et tilsvarende søk når en klient logger seg på tilgangsnivå 2, og sjekker om en angriper kan få tak i dokumenter på dette nivået:

```
search [1] in ADVERSARY : initA =>! C:Configuration

< Adversary:Oid : U:Unit | ATTS,
obtainedContent: SET*,
msgIntercepted: SET > such that ((SET contains "Other-L2" or SET*
    contains "Other-L2") or SET* contains "Welcome-L2") or SET
    contains "Welcome-L2" = true .

No solution.
states: 14713  rewrites: 7536739 in 54300ms cpu (55210ms real) (138798
    rewrites/second)
```

Vi ser av søkeresultatet at angriperen ikke vil få tak i informasjon som krever tilgangsnivå 2. Vi foretok også søk for å sjekke om det fantes et hendelsesforløp hvor angriperen får tak i informasjon på tilgangsnivå 1 med utgangspunkt i den gitte initialtilstanden. Dette søket ga heller ingen løsning.

Vi har ved de søkene som er foretatt på modellen ikke avdekket noen feil i modellen, og heller ikke avdekket noen svakheter i sikkerhetsmekanismene som denne modellen representerer. Modellen er *ikke* sårbar for den typen angriper som er opprettet.

8 Konklusjon

Dette kapittelet gir en oppsummering av de viktigste elementene i denne oppgaven og hvilke resultater vi har kommet frem til.

Innledningsvis ble følgende spørsmål stilt:

- Hvordan er Altinn bygget opp, og hvilke mekanismer har nettstedet for å ivareta sikkerheten?
- Hvordan kan disse sikkerhetsmekanismene formaliseres og presenteres i en modell?
- Hvordan kan en slik modell benyttes for analyse?

Disse spørsmålene vil besvares i dette kapittelet.

8.1.1 Altinn

Altinn er et komplekst system slik at det kun har vært utvalgte deler av systemet som har blitt presentert i denne oppgaven. Delene som er utvalgt er de som knytter seg til brukerens hendelsesforløp ved oppkobling til Altinn, innlogging og skjemautveksling.

Implementasjonen av systemet ble undersøkt for å gi komplementerende informasjon til den systemdokumentasjon som vi fikk tilgang til, og for å få en bedre forståelse av systemet. Vi har sett at en SSL-forbindelse blir opprettet mellom klienten og server. Deretter kan brukeren logge seg på systemet med sitt brukernavn og passord. Det er to forskjellige type passord. Statisk passord gir brukeren tilgang til systemet med sikkerhetsnivå 1. Ønsker brukeren tilgang til nivå 2 må vedkommende logge seg på med et engangspassord som mottas over SMS.

Undersøkelsen av Altinn avdekket at dokumentasjon og implementasjon ikke alltid stemmer helt overens. Altinn er under stadig utvikling og endring, og dette er muligens grunnen til avviket. Det ble også observert at autentikatoren ikke blir slettet når brukeren logger seg av systemet, og brukeren ble ikke bedt om å oppgi gammelt passord, når vedkommende ønsker å opprette et nytt. Ellers ble det ikke oppdaget noen svakheter og all kommunikasjon ser ut til å foregå i kryptert form over SSL, og dette stemmer overens med tilgjengelig dokumentasjon. Modellen vi har utviklet er basert på dette.

8.1.2 Maude-modellen

En formell modell kan brukes for å få en forståelse av hvordan et system bygget er opp på et abstrakt nivå. Modellen som er presentert i denne oppgaven er hovedsakelig bygget opp ved en formalisering av regler hentet fra ulike spesifikasjoner, og har fjernet seg fra implementasjonsdetaljer.

Det finnes svært mange sikkerhetsproblemer ved bruk av Internett. Veldig mange kjente angrep utnytter feil i implementasjonen, og en formell modell vil ikke alltid fange opp slike sikkerhetshull. Det er derfor ikke mulig å konkludere med at et system er sikkert ved bruk av en formell modell fordi implementasjonsdetaljer åpner for en rekke nye problemstillinger. Imidlertid vil modellen kunne styrke tillitten til de grunnleggende valg av løsninger, under antagelsen om at disse løsningene er korrekt implementert.

Det er valgt å modellere tre servere i Altinns interne nettverk:

- **Webserver**: serveren tar seg av kommunikasjonen med klienten og inneholder noe data den kan sende hvis brukeren autentiserer seg med en cookie.
- **Applikasjonsserver**: fungerer som et bindeledd mellom webserveren og databasen og sørger for autentisering og autorisasjon av brukere.
- **Database**: inneholder skjema som den sender til applikasjonsserveren dersom den får beskjed om det.

Disse serverene er valgt fordi de ble vurdert til å inneholde den funksjonalitet som er relevant for sikkerhetsanalyse, og som knytter seg til hendelsesforløp hvor enkeltpersoner benytter systemet.

Modellen er basert på lagdeling slik at TCP ligger i bunn med et grensesnitt mot SSL. Disse protokollene tilbyr sikkerhetsmekanismer for overføring av data mellom de kommuniserende partene.

Lagdeling fører til en enorm mengde tilstander og gjør at en detaljert modell blir ressurskrevende å utføre søk på. Det ble nødvendig å forenkle modellen for å få søk som returnerer innen noen timer. Etter et par timer vil minnet på tilgjengelig maskinvare være fullt slik at det har vært lite effektivt å la et søk gå utover denne tiden. Det ble iverksatt tiltak for å redusere tilstandseksplosjonen ved å slå de sammen regler eller opprette likheter der det ellers vil være naturlig for forståelsen av modellen å benytte regler. For en mer realistisk utvidelse av modellen vil vi få et ennå større tilstandsrom. For å begrense tilstandsgenereringen vil det være et naturlig utgangspunkt å kollapse lagdelingen slik at man kun lar det øverste laget angi tilstandsendringer og opprette de andre lag som likheter.

Dette problemet knytter seg til systemets kompleksitet og vil være en utfordring å løse uavhengig av hvilket modellsjekkingsverktøy man velger å benytte.

8.1.3 Resultater

Vi har i denne oppgaven laget et rammeverk for Altinn og delt dette opp i moduler slik at det skal være enkelt å utvide modellen med ytterligere funksjonalitet, eller importere enkelte deler av modellen inn i andre modeller som har fokus på andre sikkerhetsaspekter.

Som det går frem av oppgaven er modellen laget på et abstrakt plan og det er vår forståelse av Altinn på et høyt nivå som er modellert. Dette trenger ikke nødvendigvis å være representativt for den implementerte løsningen av Altinn, men modellen undersøker elementer fra designvalget.

Vi har vist at handlingsmønstre i et forholdsvis komplekst system kan formaliseres og uttrykkes på en kortfattet måte i Maude. Tabellen under viser hvor mange linjer hver modul består av:

ADVERSARY.maude	62
ERRORHANDLER.maude	89
FORMEXCHANGE.maude	189
LOGIN1.maude	111
LOGIN2.maude	126
RANDOM.maude	22
SSL.maude	239
TCP.maude	47
TOOLS.maude	177
UNITS.maude	69
Sum	1131

Disse tallene er basert på kode som inneholder kommentarer og blanke linjer for å bedre lesbarheten, slik at den eksekverbare koden vil være enda kortere enn det som er angitt her. Config-filene er ikke telt med her, fordi de kun inneholder initialtilstander, og er ikke en del av selve modellen.

Modellen kan uttrykke komplekse problemer på en enkel måte. Eksempelvis kan hele autorisasjonsprosedyren som krever oppslag i tre forskjellige tabeller gjøres med én regel i Maude som vist i seksjon 6.8.3.

Et annet eksempel på hvor kompakt Maude-koden er, kan illustreres ved at hele SSL Record protokollen kan uttrykkes ved hjelp av to regler. Den ene reglen tar for seg utgående meldinger og legger til en MAC, krypterer meldingsinnholdet, legger til SSL-header og sender meldingen videre

nedover i lagstrukturen, slik at TCP-protokollen kan overta jobben med å sekvensiere og sende meldingen. Den andre regelen tar for seg innkommende meldinger og dekrypterer, sjekker MAC og fjerner header. Hver av disse to reglene er under ti linjer og det gir et godt inntrykk av hvor kortfattet ting kan uttrykkes i Maude.

Vi mener modellen gir en god representasjon av Altinns sikkerhetsmeksanismer på et abstrakt nivå. Modellen er eksekverbar, den kan utvides og vi kan bruke den i en fornuftig analyse av sikkerheten.

Det er ikke oppdaget noen sikkerhetsfeil i rammeverket med hensyn til de underliggende antagelsene. Dette støtter opp om Altinns designvalg og sikkerhetsvalgene synes å være fornuftige.

8.1.4 Videre arbeid

Det er mange muligheter for å utvide og forbedre modellen. Det mest interessante vil muligens være å forenkle regler slik at tilstandsgenereringen avtar og man kan utvide modulen ADVERSARY med regler for en rekke vanlige sikkerhetsbrudd som for eksempel meldingsmodifikasjon, automatisk generering av nye nøkler, multiple oppkoblinger og reply av meldinger. Deretter kan man kjøre søk etter uønskede tilstander for å undersøke om det finnes svakheter i modellen når man involverer en mer avansert angriper. Ved en begrensning av tilstandsgenerering vil man også kunne kjøre søk med et høyere antall klienter og webservere i konfigurasjonen.

Modellen kan også utvides med mer av Altinns funksjonalitet som for eksempel kommunikasjonen mellom Altinn og fagsystemer, innlogging med Buypass, eller en mer finkornet autorisasjonsmodul. Eksplisitt modellering av tidsforbruk og kø-ordninger for klienter mot Altinn er også et aspekt som kan modelleres.

Altinn har gjennomgått endringer siden denne oppgaven ble påbegynt så det vil sannsynligvis være interessant å sette seg inn i endringene og la en ny versjon av modellen gjenspeile dette.

9 Bibliografi

[1] PMA, "Forskning ved Presis modellering og analyse (PMA)," http://www.ifi.uio.no/forskning/grupper/pma/research/.

[2] Andrew S. Tanenbaum, *Computer Networks*, fourth ed. Upper Saddle River, N.J.: Prentice-Hall PTR, 2003.

[3] Gisle Hannemyr, "Nettverksarkitektur," http://www.ifi.uio.no/inint/emne04a.html.

[4] Linuxguiden, "Transmission Control Protocol," http://www.linuxguiden.no/index.php/TCP.

[5] "Webopedia," Jupitermedia Corporation, http://www.webopedia.com/.

[6] R. Fielding T. Berners-Lee, H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0," vol. 2004: IETF, 1996, http://www.ietf.org/rfc/rfc1945.txt?number=1945.

[7] Arman Danesh, *Internett med Java og JavaScript*: Prentice Hall, 1996.

[8] Microsoft, "Description of Cookies," 2003, http://support.microsoft.com/kb/260971/EN-US/

[9] Lars Marius Garshol, "An introduction to XML," http://www.garshol.priv.no/download/text/xml-intro/index-en.html.

[10] S. Pfleeger C. Pfleeger, *Security in Computing*, third ed. N.J.: Prentice Hall PTR, 2002.

[11] Emil Sit Kevin Fu, Kendra Smith, Nick Feamster, "Dos and Don`ts of Client Authentication on the Web," in *10th USENIX Security Symposium*, vol. 2004. Whashington, D.C.: USENIX Association, 2001, http://cookies.lcs.mit.edu/pubs/webauth.html.

[12] Sandeep Grover, "Buffer Overflow Attacks and Their Countermeasures," 2003, http://www.linuxjournal.com/article.php?sid=6701.

[13] Jon Erickson, *Hacking - The art of exploitation*: No Starch Press, 2003.

[14] Imperva, "Session Hijacking," in *Application Defense Center*, http://www.imperva.com/application_defense_center/glossary/session_hijacking.html.

[15] Matt Bishop, *Computer security: art and science*: Addison-Wesly, 2003.

[16] William Stallings, *Network Security Essentials: Applications and Standards*, second ed. Upper Saddle River, N.J.: Prentice Hall, 2003.

[17] The International PGP Home Page, "Overview of PGP," http://www.pgpi.org/doc/overview/.

[18] Doris Baker H.X. Mel, *Cryptography decrypted*. Upper Saddle River, N.J.: Addison-Wesley, 2000.

[19] Prof.: M. Van Droogenbroeck, "Introduction to PKI - Public Key Infrastructure," 2004, 2002, http://www.k-binder.be/Papers/PKI_V11.pdf.

[20] Philip Karlton Alan O. Freier, Paul C. Kocher, "The SSL Protocol Version 3.0," vol. 2004: Netscape, 1996, http://wp.netscape.com/eng/ssl3/draft302.txt.

[21] Altinn, "Sende skjema fra fagsystemer," https://www.altinn.no/cms/1044/altinn/Tjenester/Sende+fra+fagsystem.htm.

[22] Altinn, "Datamodell Altinn rapporteringsløsning v.4.0."

[23] Altinn, "Vedlegg 13 - Functional design Register v.4.0," 2003.

[24] Altinn, "Vedlegg 10 - Funksjonelt design Sikkerhetsløsning v.5.0," 2003.

[25] Altinn, "Teknisk spesifikasjon v.4.0," 2003.

[26] Altinn, "Security solution - Technical design v.4.0," 2003.

[27] Altinn, "Overordnet spesifikasjon v.4.0," 2003.

[28] Altinn, "Overordnet design Portal v.4.0," 2003.

[29] Altinn, "Funksjonell spesifikasjon v.4.0," 2003.

[30] DevX, "Short Message Service (SMS)," http://www.devx.com/wireless/Article/22266.

[31] Mike Benham, "IE SSL Vulnerability," 2002, http://www.thoughtcrime.org/ie-ssl-chain.txt.

[32] David Legard, "Severe Security Flaw Found in IE."

[33] Microsoft, "Internet Information Services," http://www.microsoft.com/windowsserver2003/iis/default.mspx.

[34] Adobe, "Adobe LiveCycle Forms (formerly Form Server)."

[35] Netegrity, "SiteMinder 6.0 Product Overview."

[36] Accenture, "Security solution - Technical design - 06.08.03-VERSJON 4.0."

[37] Techmetrix Research, "BizTalk Server 2000: An Overview," http://intranetjournal.com/articles/200106/bt_06_27_01a.html.

[38] Microsoft, "BizTalk Server 2004: Building the Smart, Connected Enterprise," http://www.microsoft.com/biztalk/.

[39] Microsoft, "Microsoft SQL Server," http://www.microsoft.com/sql/default.mspx.

[40] Microsoft, "Microsoft Message Queuing (MSMQ) Center," http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp.

[41] IBM, "Enterprise Storage Server family," http://www-1.ibm.com/servers/storage/disk/ess/.

[42] UIUC SRI International, "The Maude website," 2004, http://maude.cs.uiuc.edu/.

[43] Theodore McCombs, "Maude 2.0 Primer: version 1.0," vol. 2004, 2003, http://maude.cs.uiuc.edu/primer/.

[44] Francisco Duràn Manuel Clavel, Steven Eker, Patrick Lincoln, Narciso Martì-Oilet, Josè Meseguer, Carolyn Talcott, "Maude Manual (version 2.1)," vol. 2004, 2004, http://maude.cs.uiuc.edu/maude2-manual/.

[45] Peter Csaba Ølveczky, "Formal Modeling and Analysis of Distributed Systems in Maude," 2003.

[46] Tech FAQ, "What is SMS (Short Message Service)?" http://corky.net/2600/wireless-networks/sms-short-message-service.shtml.

[47] Microsoft, "Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetHT04.asp.

[48] Einar Broch Johnsen, Olaf Owe og Eyvind W. Axelsen, "A Run-Time Environment for Concurrent Objects with Asynchronous Method Calls," *Electronic Notes in Theoretical Computer Science 117: 375-392.*, 2005.

[49] Ethereal, "Powerful Multi-Platform Analysis," http://www.ethereal.com/.

UNIVERSITETET I OSLO
Institutt for Informatikk

Appendix

til Masteroppgave

Formell modellering og analyse av sikkerhetsaspekter ved web-baserte offentlige registre

Line Borgund
April 2005

Innholdsfortegnelse

Appendix A

Vi vil her liste opp de variable vi har brukt i modellen. Disse er brukt gjennomgående.
var A : Aid .
var ATT ATT* : Attribute .
var ATTS ATTS* : AttributeSet .
var CID : Cid .
var C : Configuration .
var CK CK* : Cookie .
var DATA : Data .
var DATALIST : DataList .
var E : Element .
var NAME FUNCTION OP ROLE FORM FORM* NAME* FUNCTION* OP* ROLE* E E* : Element .
var H : Header .
var L : List .
var L L* LI MSGL MSGL* CL LL CRL :List .
var MAC : Mac .
var M : Msg .
var MSG MSG* : MsgContent .
var N T* TI N LVL LVL* RANDOM RANDOM*: Nat .
var N : NzNat . ***N er brukt som NzNat i noen av konfig-filene
var WS AS CLIENT SERVER A B CERT CERT* ISSUER DB KEYPAIRS
var OTP : OneTimePair .
var PW PW* : Password .
var KR KR* : PrivKey .
var KU KU* : PubKey .
var REC : Record .
var PWS SET : Set .
var SET SET* SSET TSET KSET KSET* : Set .
var SSET TSET KSET KSET* OSET SET : Set .
var SID : Sid .
var S S* FUNCTION NAME OP ROLE KU KR KU* KR* : String .
var SKEY CKEY MKEY: SymmKey .
var U : Unit .

Appendix B

Her vil all Maude kode som tilhører modellen.

TOOLS.maude

```
mod MSGCONTENT is
pr STRING .
pr CONFIGURATION .
sort MsgContent .
sorts Header Data DataList .
subsort String <  Data Header .
subsort Header < MsgContent .
subsort Data < DataList .
op emptyDataList :  -> DataList[ctor] .
op _;_ : DataList DataList -> DataList[ctor assoc id: emptyDataList] .
op _-_ : Header DataList -> MsgContent[ctor ] .
op msg_to_from_ : MsgContent Oid Oid -> Msg[ctor  format(m m m m m m mo)] .
*** Internet messages
op aMsg_to_from_ : MsgContent Oid Oid -> Msg[ctor format(m m m m m m mon) ] .
***Altinn internal messages
op sms_to_from_ : MsgContent Oid Oid -> Msg[ctor format(m m m m m m mon )] .
endm

mod DATACOLLECTIONS is
pr CONFIGURATION .
sorts List Set Element .
subsort Msg < Element < List Set .
op nil : -> List[ctor] . ***Ordinary list
op _::_ : List List -> List[ctor assoc id: nil] .
op emptySet : -> Set[ctor] .
op _;;_ : Set Set -> Set[ctor assoc comm id: emptySet] .
op _contains_ : Set Element -> Bool .
op _contains_ : List Element -> Bool .
vars E E* : Element .
```

```
var SET : Set .
var L : List .
eq E ;; E = E . ***removes duplicate entries from set
eq emptySet contains E = false .
eq E ;; SET contains E* = if E == E* then true else SET contains E* fi .
eq nil contains E = false .
eq E :: L contains E* = if E == E* then true else L contains E* fi .
endm

mod SECURITYMECHANISMS is
pr DATACOLLECTIONS .
pr MSGCONTENT .
pr CONVERSION .

sorts SymmKey PrivKey PubKey .  ***Symmetric- , private- and public-keys
sort Hash .         ***One-way function used to reduce messagecontent to a thumbprint
sort Signature Mac . ***Used to ensure a message is from the claimed sender
sort KeyPair .    ***A pair of asymmetric keys (publicKey & privateKey)
sort Cookie .
sort Password .
sort PWPair OneTimePair . ***Pair of Oid and password static and randomgenerated
sort Sid  . ***SessionID : unike for each session between a spesific server and client
(numberformat)
sort Record .  ***Database records
sort Ticket .

subsorts String < PrivKey PubKey SymmKey Password .
subsort KeyPair Oid String PWPair Record OneTimePair < Element .
subsort List Set Signature Hash Mac Cookie  < Data .
subsort Nat < Sid .

***Databse records
op [_,_] : String String -> Record[ctor format(nto o o o o o )] .
op [_,_] : Oid String -> Record[ctor format(nto o o o o o )] .
op [_,_,_,_] : String String String Nat -> Record[ctor format(nto o o o o o o o o o)] .

***Cookie MsgContent
op _from_ : DataList Oid -> Element[format(nto o o o)] .

***type conversion
op toString : Nat -> String .
op hash : List -> Hash .
```

```
op pw : Nat -> String[ctor] .

***Timer
op <Time | tick:_> : Nat -> Object[ctor format(nb b b o b bo)] .

*** passwordpair
op _&_ : Oid Password -> PWPair .
op (_&_&_&_) : Oid Password Nat Nat -> PWPair .


***_____DataIntegrity mechanisms_____
***signature used to authenticate message sender
op sign_with_ : Hash PrivKey -> Signature[ctor] .

***Message Authentication Code (MAC) used to authenticate message sender
op mac : Msg SymmKey -> Mac[ctor] .  *** Mac of handshake-history-list and SSL-Record-Protocol
op validMac : Msg  Mac SymmKey -> Bool .

***Construct a asymmetric keypair
op (_&_) : PubKey PrivKey -> KeyPair[ctor] .


***_____Encipherment_____

*** symmetric and asymmetric
op sEncrypt_with_ : DataList SymmKey -> DataList[ctor] . ***Symmetric encr.
op sDecrypt_with_ : DataList SymmKey -> DataList .

op sEncrypt_with_ : Ticket SymmKey -> Cookie[ctor] . ***Symmetric encr. of tickets
op sDecrypt_with_ : Ticket SymmKey -> Ticket .

op aEncrypt_with_ : SymmKey PubKey -> DataList[ctor] . ***Asymmetric encr. Used to encrypt a sessionKey
op aDecrypt_with_ : MsgContent PrivKey -> DataList .

***List of all Asymmetric keypairs
***The advanced computations done inside a local machine is here replaced by
*** a list of compliant Asymmetric Key pairs
op <ValidKeyPairs | keySet:_ > : Set -> Object [ctor format(nb b b o b bo)] .

***_____Certificates_____
```

***Certificates are identified by an unique certificateId.
***issuerName is the name of the Certificate Authority (CA) holding the certificate
***subjectName is the name of the public keyholder this sertificate is issued for
***subjectKu is the subjects Public Key (KU)
***signature is the whole certificate signed by issuer
op <_: Certificate | issuerName:_, subjectName:_, subjectKu:_, signature:_ > :
Oid Oid Oid PubKey Signature -> Object[ctor format(ng g g g g g g g g g g g g g g g g g g)] .

***Revoke Certificate List(CRL) contains list of all certificates that are revoked and signature
op <CRL | list:_ , signature:_ > : List Signature -> Object[ctor format(ng g g g g g g g go)] .
***op list:_ : List -> Attribute [ctor] .
***op signature:_ : Signature -> Attribute [ctor] .

***_____

***Authentication-ticket based on .Net HowTo documentation from Microsoft. + altinndocs.
***The attributes represents: client timeNow timeExpires level
op authTicket : Oid Nat Nat Nat -> Ticket[ctor] .
op authCookie : Oid Nat SymmKey Nat -> Cookie[ctor] .
op null : -> Cookie .
op validCookie1 : Cookie Oid Nat SymmKey -> Bool .
op validCookie2 : Cookie Oid Nat SymmKey -> Bool .
op getLevel : Cookie -> Nat .

var MSG MSG* : MsgContent .
var SKEY CKEY : SymmKey .
var MAC : Mac .
var CID O CLIENT : Oid .
var E : Element .
var KR* : PrivKey .
var T T* TI : Nat .
var H : Header .
var LVL : Nat .

eq authCookie(CLIENT , T , CKEY, LVL) = sEncrypt authTicket(CLIENT, T, T + 60 , LVL) with CKEY .

***validate client, time, encryption and level.

```
eq validCookie1(null, CLIENT , T*, CKEY) = false .
eq validCookie1((sEncrypt authTicket(CLIENT, T, TI , LVL) with CKEY) , CLIENT,
T*, CKEY ) =  if TI > T* and LVL >= 1 then true else false fi .
eq validCookie2(null, CLIENT , T*, CKEY) = false .
eq validCookie2((sEncrypt authTicket(CLIENT, T, TI , LVL) with CKEY) , CLIENT,
T*, CKEY ) =  if TI > T* and LVL >= 2 then true else false fi .
eq getLevel(sEncrypt authTicket(CLIENT, T, TI , LVL) with CKEY) = LVL
 .
eq getLevel(null) = 0 .

***Check Message Authentication code (MAC)
eq validMac(MSG , mac(MSG, SKEY) , SKEY) = true .


endm
```

UNITS.maude

```
in TOOLS .

***All communication-units with attributes
mod UNITS is
pr SECURITYMECHANISMS .

sort Unit .

***Classes
op <_:_|_> : Oid Unit AttributeSet -> Object[ctor format(nr r r r r o r ro)] .
op [_] : AttributeSet -> Element[ctor format(nc c c c )] .

***SessionElements constructor
op [_: Session |_] : Sid AttributeSet -> Element[ctor format(nb b b b b b b )] .

***Units
ops Client WebServer ApplServer Database : -> Unit[ctor] .

***General Attributes
op request:_ : List -> Attribute [format( o o o) ] . ***Used in configuration to give the
client`s actions in advance.
op commPartner:_ : Oid -> Attribute [format (o o o)] .
op randomGenerator:_ : Nat -> Attribute [format( o o o) ] .
```

***TCP attributes
op seqnrOut:_ : Nat -> Attribute [format(c c c)]. ***Sequense number
op window:_ : Set -> Attribute [format(c c c)]. ***Received messages
op tcpSessions:_ : Set -> Attribute [format(c c c)] .
op seqnrIn:_ : Nat -> Attribute [format(c c c)].
op tcpReceived:_ : List -> Attribute[format(!c oc c)] . ***Interface
op tcp:_ : List -> Attribute[format(!c oc c)] . ***Interface

***SSL attributes
op handshakeMsgs:_ : List -> Attribute [format(b b b)]. ***List of messages seen in handshake
op sessionSet:_ : Set -> Attribute[format(b b b)] .
op trustedIssuerSet:_ : Set -> Attribute[format(b b b)] .
op privKey:_ : String -> Attribute[format(b b b)] . ***Private Key

op state:_ : Nat -> Attribute[format(b b b)] .
op sKey:_ : SymmKey -> Attribute[format(b b b)] . ***sessionKey
op mKey:_ : SymmKey -> Attribute[format(b b b)]. ***MACKey
op received:_ : List -> Attribute[format(!b ob b)] . ***Contains the last received msg. in decrypted form
op keyMaterial:_ : Nat -> Attribute [format(b b b)]. ***Storing of numbers used to calculate keys

op ssl:_ : List -> Attribute [format(!b ob b)] . ***contains messages to be handled of ssl Record Protocol

***LOGIN Attributes
op validHeaders:_ : Set -> Attribute [format(o o o)] . ***used for errorhandling in webservers
op cKey:_ : SymmKey -> Attribute [format(o o o)] . ***CookieKey
op content:_ : List -> Attribute [format(o o o)] . ***The information contained by the unit
op obtainedContent:_ : Set -> Attribute[format(no! o o)]. ***Set of information collected by a unit
op passwordSet:_ : Set -> Attribute [format(o o o)] . ***to store Password Pairs for authentication
op password:_ : Password -> Attribute [format(o o o)] . *** the units personal password.
op passwordGenerator:_ : Nat -> Attribute [format(o o o)] . ***Used to generate onetime-passwords

```
op oneTimeSet:_ : Set -> Attribute [format( o o o) ] . *** Used to store one-time password in server
op content2:_ : List -> Attribute [format( o o o) ] . ***Webservers content at level2
op authCookie:_ : Cookie -> Attribute [format( o o o) ] . ***Client stores authCookie here

***FORMEXCHANGE Attributes
op formNames:_ : Set -> Attribute .
op functions:_ : Set -> Attribute .
op acl:_ : Set -> Attribute [format(n! ! o)] .
op roleList:_ : Set -> Attribute [format(n! ! o)] .
op userRoles:_ : Set -> Attribute [format(n! ! o) ] .
op forms:_ : Set -> Attribute [format(n! ! o)].
op numbering:_ : Nat -> Attribute .
op combinations:_ : Set -> Attribute .


endm
```

TCP.maude (v.1)

```
***TPC v.1
in UNITS
in RANDOM
mod TCP is
pr UNITS .
pr RANDOM .
var U U* : Unit .
var ATTS ATTS* : AttributeSet .
var N N* : Nat .
var SET SET* : Set .
var L : List .
var MSG : MsgContent .
var CLIENT SERVER A B : Oid .


***Handshake not requered in Maude since no messages get lost.
***Makes a set with sequence numbers to control order.
rl[tcp_init]:
< CLIENT : Client | request: ("startTCP" :: L), tcpSessions: SET*, randomGenerator: N,  ATTS >
< SERVER : WebServer | tcpSessions: SET, ATTS* >
```

=>
< CLIENT : Client | request: L, tcpSessions: (SET* ;; [commPartner: SERVER, window: emptySet, seqnrIn: N, seqnrOut: N]),
randomGenerator: rand(N), ATTS >
< SERVER : WebServer | tcpSessions: (SET ;; [commPartner: CLIENT, window: emptySet, seqnrIn: N, seqnrOut: N]) , ATTS* > .

rl[tcp_send]:
< A : U | tcp: ((msg MSG to B from A) :: L), tcpSessions: ([commPartner: B, seqnrOut: N, ATTS*] ;; SET), ATTS >
=>
< A : U | tcp: L, tcpSessions: ([commPartner: B, seqnrOut: (N + 1), ATTS*] ;; SET), ATTS >
(msg toString(N) - MSG to B from A) .

***No window size and accepting all messages.
rl[tcp_receive]:
(msg toString(N) - MSG to A from B)
< A : U | tcpSessions: ([commPartner: B, window: SET, seqnrIn: N*, ATTS*] ;; SET*), ATTS >
=>
< A : U | tcpSessions: ([commPartner: B, window: ((msg toString(N) - MSG to A from B) ;; SET),
seqnrIn: N*, ATTS*] ;; SET*), ATTS > .

***when next expected sequense-number occor, then remove tcp header and put in list available for upper-layers
rl[tcp_sort_messages]:
< A : U | tcpSessions: ([window: ((msg toString(N) - MSG to A from B) ;; SET),seqnrIn: (N), ATTS*] ;; SET*),
tcpReceived: L, ATTS >
=>
< A : U | tcpSessions: ([window: SET,
seqnrIn: (N + 1), ATTS*] ;; SET*), tcpReceived: (L :: (msg MSG to A from B)) , ATTS > .
endm

TCP.maude (v.2)

***TCP v.2
in UNITS
in RANDOM

```
mod TCP is
pr UNITS .
pr RANDOM .
var U U* : Unit .
var ATTS ATTS* : AttributeSet .
var N N* : Nat .
var SET SET* : Set .
var L : List .
var MSG : MsgContent .
var CLIENT SERVER A B : Oid .


***Handshake not requered in Maude since no messages get lost.
***Makes a set with sequence numbers to control order.
eq
< CLIENT : Client | request: ("startTCP" :: L), tcpSessions: SET*, randomGenerator: N,  ATTS >
< SERVER : WebServer | tcpSessions: SET, ATTS* >
=
< CLIENT : Client | request:  L, tcpSessions: (SET* ;;  [commPartner: SERVER, window: emptySet, seqnrIn: N, seqnrOut: N] ),
randomGenerator: rand(N), ATTS >
< SERVER : WebServer | tcpSessions: (SET ;;  [commPartner: CLIENT, window: emptySet, seqnrIn: N, seqnrOut: N]) , ATTS* > .

eq
< A : U | tcp: ((msg MSG to B from A) :: L), tcpSessions: ([commPartner: B, seqnrOut: N, ATTS* ] ;; SET), ATTS >
=
< A : U | tcp: L, tcpSessions: ([commPartner: B, seqnrOut: (N + 1), ATTS* ] ;;  SET), ATTS >
(msg toString(N) - MSG to B from A ) .


***No window size and accepting all messages.
eq
(msg toString(N) - MSG to A from B )
< A : U | tcpSessions: ([commPartner: B, window: SET, seqnrIn: N*, ATTS* ] ;; SET*), ATTS >
=
< A : U | tcpSessions: ([commPartner: B,  window: ((msg toString(N) - MSG to A from B) ;; SET),
```

seqnrIn: N*, ATTS*] ;; SET*), ATTS >
.

***when next expected sequense-number occor, then remove tcp header and put in list available for upper-layers
eq
< A : U | tcpSessions: ([window: ((msg toString(N) - MSG to A from B) ;; SET),seqnrIn: (N), ATTS*] ;; SET*), tcpReceived: L, ATTS >
=
< A : U | tcpSessions: ([window: SET, seqnrIn: (N + 1), ATTS*] ;; SET*), tcpReceived: (L :: (msg MSG to A from B)) , ATTS > .

endm

SSL.maude

in TCP
mod SSL is
pr TCP .

***Generate mac key that is based on the same information as the sessionKey
op newKey : SymmKey -> SymmKey .

***Makes a new session Object with parameters: SessionId, commPartner, SessionPendingState and SessionKey material
***Sessionstate = 0-1 means the session is still negotiated
***Sessionstate = 2 means the session is valid
op newSession : Sid Oid Nat List -> Element[ctor] .

***Making the Symmeteric SessionKey from a Nat that`s made by randomgenerator.
op symmKey : Nat -> SymmKey[ctor] .

var CLIENT SERVER CERT CERT* ISSUER KEYPAIRS CP A B : Oid .
var SSET TSET KSET KSET* OSET SET : Set . ***SessionSet, trustedIssuerSet, KeySets, obtainedContent
var RANDOM RANDOM* N T : Nat .
var KU KU* : PubKey .
var KR KR* : PrivKey .
var SID S : Sid .

```
var SKEY MKEY : SymmKey .
var MSG C : MsgContent .
var L L*  CRL MSGL MSGL* CL : List .
var E : Element .
var ATTS ATTS* : AttributeSet .
var DATA : Data .
var U : Unit .
var M : Msg .

eq newSession(SID, CP, RANDOM, MSGL)=
 [SID : Session | commPartner: CP, keyMaterial: RANDOM, handshakeMsgs: MSGL,
state: 0 ] .

***_____ Handshake protocol_____
***Authentication of server and negotion of cryptographic keys.

*** variable options are chosen as follows :
***server authenticate with certificate in phase 2 and this implies no
server_key_exchange.
***server does not ask for client certificate and this implies: no certificate messages
from client i phase 3
***All messages sent during handshake is copied into the session handshakeMsgs List.
This list is hashed and sent in the finished messages



***sending Client_hello with clients initial random
***state 0
***Start of SSL handshake phase 1
rl[client_hello] :
< CLIENT : Client | sessionSet: SSET,request: ("startSSL" :: L*), randomGenerator:
RANDOM, tcp: L,ATTS >
< SERVER : WebServer | ATTS* >
<Time | tick: T >
 =>
< CLIENT : Client | sessionSet: (SSET ;;(newSession(0 , SERVER, RANDOM,(msg
"client_hello" - toString(RANDOM)to SERVER from CLIENT)))), request: L*,
randomGenerator: rand(RANDOM + T), tcp: (L :: (msg "client_hello" -
toString(RANDOM)to SERVER from CLIENT)),  ATTS >
< SERVER : WebServer | ATTS* >
 <Time | tick: T >  .
```

***WebServer is responding by sending a server_hello message with the sessionID and server initial random
***WebServer is sending it`s certificate with a valid signature
***State 0
***Finishing phase 1 and 2
rl [server_hello]:
< SERVER : WebServer | sessionSet: SSET, randomGenerator: RANDOM*,
tcpReceived: ((msg "client_hello" - toString(RANDOM)to SERVER from CLIENT) :: L*), tcp: L, ATTS >
< CERT : Certificate | issuerName: ISSUER, subjectName: SERVER, subjectKu: KU,
signature: sign hash(CERT :: ISSUER :: SERVER :: KU)with KR* >
<Time | tick: T >
=>
< SERVER : WebServer | sessionSet: (SSET ;; newSession(RANDOM*, CLIENT,(RANDOM* + RANDOM),
((msg "client_hello" - toString(RANDOM)to SERVER from CLIENT) ::
(msg "server_hello" - toString(RANDOM*)to CLIENT from SERVER) ::
(msg "certificate" - CERT to CLIENT from SERVER) ::
(msg "server_hello_done" to CLIENT from SERVER))
)),randomGenerator: rand(RANDOM* + T),
tcp: (L ::
(msg "server_hello" - toString(RANDOM*)to CLIENT from SERVER) :: *** init Random
(msg "certificate" - CERT to CLIENT from SERVER) ::
(msg "server_hello_done" to CLIENT from SERVER)),
tcpReceived: L*, ATTS >
< CERT : Certificate | issuerName: ISSUER, subjectName: SERVER, subjectKu: KU,
signature: sign hash(CERT :: ISSUER :: SERVER :: KU)with KR* >
<Time | tick: T + 1 >
.

***Client verifies that certificate received is issued by a trused issuer and that it's not in CRL
***Verifies that the issuer-certificate has a valid signature.
***Client sends keyinformation: a randomkey is generated and sendt encrypted with the servers public key.
***The finish message is the first messages to be encrypted with the negotiated key.
***State 0->1
***phase3 and starting phase 4
crl[client_key_exhange]:

```
< CLIENT : Client |
tcpReceived: ((msg "server_hello" - toString(RANDOM*)to CLIENT from SERVER)
::
(msg "certificate" - CERT to CLIENT from SERVER) ::
(msg "server_hello_done" to CLIENT from SERVER) :: L*),tcp: L,
sessionSet: (SSET ;; [S : Session | commPartner: SERVER, state: 0, handshakeMsgs:
MSGL, keyMaterial: N, ATTS*]),
trustedIssuerSet: TSET,randomGenerator: RANDOM,
 ATTS >
<ValidKeyPairs | keySet: KSET* >
< CERT : Certificate | issuerName: ISSUER, subjectName: SERVER, subjectKu: KU,
 signature: sign hash(CERT :: ISSUER :: SERVER :: KU)with KR* > ***Senders
certificate
< CERT* : Certificate | issuerName: ISSUER, subjectName: ISSUER, subjectKu:
KU*,
          signature: (sign hash(CERT* :: ISSUER :: ISSUER :: KU*)with KR*)>
<CRL | list: CL, signature: (sign hash(CRL)with KR*)> ***issuers certificate
<Time | tick: T >
=>
< CLIENT : Client | sessionSet:
(SSET ;; [RANDOM* : Session | commPartner: SERVER, state: 1, keyMaterial: (N +
RANDOM + RANDOM*), handshakeMsgs:
(MSGL ::
((msg "server_hello" - toString(RANDOM*)to CLIENT from SERVER) ::
(msg "certificate" - CERT to CLIENT from SERVER) ::
(msg "server_hello_done" to CLIENT from SERVER) ::
(msg "client_key_exchange" - aEncrypt toString(RANDOM)with KU to SERVER
from CLIENT))), ATTS*]), ***end sessionAttribute
trustedIssuerSet: TSET,randomGenerator: rand(RANDOM + T),
tcpReceived: L* ,
tcp: (L ::
     (msg "client_key_exchange" - aEncrypt(toString(RANDOM))with KU to SERVER
from CLIENT) ::
     (msg "change_cipher_spec" to SERVER from CLIENT)),
ATTS > ***end Client
<ValidKeyPairs | keySet: KSET* >
< CERT : Certificate | issuerName: ISSUER, subjectName: SERVER, subjectKu: KU,
 signature: sign hash(CERT :: ISSUER :: SERVER :: KU)with KR* > ***Senders
certificate
< CERT* : Certificate | issuerName: ISSUER, subjectName: ISSUER, subjectKu:
KU*,
```

signature: sign hash(CERT* :: ISSUER :: ISSUER :: KU*)with KR* > ***issuers certificate
<CRL | list: CL , signature: (sign hash(CRL)with KR*)>
<Time | tick: T >
if(TSET contains ISSUER and KSET* contains(KU* & KR*)and(not CL contains CERT)and(not CL contains CERT*)).

***If KU and KR is a valid key pair belonging to SERVER then:
***Compute new session key
***State 0->1
crl[server_change_cipher_spec]:
< SERVER : WebServer | sessionSet: (SSET ;;
[SID : Session | commPartner: CLIENT, state: 0, keyMaterial: N, handshakeMsgs: MSGL, ATTS*]),privKey: KR,
tcpReceived: ((msg "client_key_exchange" - aEncrypt toString(RANDOM)with KU to SERVER from CLIENT) :: L*), tcp: L, ATTS >
<ValidKeyPairs | keySet: KSET* >
<Time | tick: T >
=>
< SERVER : WebServer | sessionSet: (SSET ;;
[SID : Session | commPartner: CLIENT, state: 1, keyMaterial: (N + RANDOM), handshakeMsgs: (MSGL ::
(msg "client_key_exchange" - aEncrypt toString(RANDOM)with KU to SERVER from CLIENT))
, ATTS*]),privKey: KR,
tcpReceived: L*, tcp: (L :: (msg "change_cipher_spec" to CLIENT from SERVER)), ATTS >
<ValidKeyPairs | keySet: KSET* >
<Time | tick: T + 1 >
if KSET* contains(KU & KR).

***_____Change Cipher Spec Protocol_____
***eats messages and updates state

rl[client_change_cipher]:
< CLIENT : Client | sessionSet: (SSET ;; [SID : Session | commPartner: SERVER, state: 1, keyMaterial: N, ATTS*]), tcpReceived: ((msg "change_cipher_spec" to CLIENT from SERVER) :: L*), ATTS >
=>

< CLIENT : Client | sessionSet: (SSET ;; [SID : Session | commPartner: SERVER, state: 2, sKey: symmKey(N),
mKey: symmKey(N + 1), ATTS*]), tcpReceived: L*, ATTS > .

rl[server_change_cipher]:
< SERVER : WebServer | sessionSet: (SSET ;; [SID : Session | commPartner: CLIENT, state: 1, keyMaterial: N, ATTS*]),
tcpReceived: ((msg "change_cipher_spec" to SERVER from CLIENT) :: L*), ATTS >
<Time | tick: T >
=>
< SERVER : WebServer | sessionSet: (SSET ;; [SID : Session | commPartner: CLIENT, state: 2, sKey: symmKey(N),
mKey: symmKey(N + 1), ATTS*]), tcpReceived: L*, ATTS >
<Time | tick: T + 1 > .

***_____ Handshake protocol - continued_____

***After sending key_exch. send the finished msg as the first ecrypted msgs.
***State 2->3
rl[client_finish]:
< CLIENT : Client | sessionSet: (SSET ;; [S : Session | commPartner: SERVER, state: 2,
        sKey: SKEY, mKey: MKEY, handshakeMsgs: MSGL, ATTS*]), tcp: L, ATTS >
=>
< CLIENT : Client | sessionSet: (SSET ;; [S : Session | commPartner: SERVER, state: 3,
        sKey: SKEY, mKey: MKEY, handshakeMsgs: MSGL, ATTS*]), tcp: (L :: (msg "finished" -(sEncrypt hash(MSGL); mac((msg "finished" - hash(MSGL)to SERVER from CLIENT), MKEY)with SKEY)to SERVER from CLIENT)), ATTS >
.

***Decrypt Msg if matching Key exists.
***Validate hash of all handshakeMsgs.
***State 2->4 WebServer hasn`t state 3.
rl[server_finish] :

< SERVER : WebServer | sessionSet: (SSET ;; [SID : Session | commPartner: CLIENT, state: 2, sKey: SKEY, handshakeMsgs: MSGL,
mKey: MKEY, ATTS*]),
tcpReceived: ((msg "finished" -(sEncrypt hash(MSGL); mac((msg "finished" - hash(MSGL)to SERVER from CLIENT), MKEY)with SKEY)to SERVER from CLIENT) :: L*), tcp: L, ATTS >
<Time | tick: T >
=>
< SERVER : WebServer | sessionSet: (SSET ;; [SID : Session | commPartner: CLIENT, state: 4, sKey: SKEY,
mKey: MKEY, ATTS*]), tcpReceived: L*,
tcp: (L :: (msg "finished" -(sEncrypt hash(MSGL); mac((msg "finished" - hash(MSGL)to CLIENT from SERVER), MKEY)with SKEY)to CLIENT from SERVER)), ATTS >
<Time | tick: T + 1 > .

***Decrypt Msg if matching Key exists.
***Validate hash of all handshakeMsgs.
***Eat msgs.
rl[handshake_finish]:
< CLIENT : Client | sessionSet: (SSET ;; [S : Session | commPartner: SERVER, state: 3, sKey: SKEY, handshakeMsgs: MSGL,
mKey: MKEY, ATTS*]), tcpReceived: ((msg "finished" -(sEncrypt hash(MSGL); mac((msg "finished" - hash(MSGL)to CLIENT from SERVER), MKEY)with SKEY)to CLIENT from SERVER) :: L*), ATTS >
=>
< CLIENT : Client | sessionSet: (SSET ;; [S : Session | commPartner: SERVER, state: 4, sKey: SKEY,
mKey: MKEY, ATTS*]), tcpReceived: L*, ATTS > .

eq newKey(symmKey(N))= symmKey(N + 1).

***_____Record Protocol_____
***Add SSL header add mac & encrypt communication

***ssl-encrypt the plaintext messages
*** Add ssl-header(sessionID)+ mac + encrypt
rl[record_protocol_send]:

< A : U | sessionSet: (SSET ;; [SID : Session | commPartner: B, state: 4, sKey: SKEY, mKey: MKEY, ATTS*]),
ssl: ((msg MSG to B from A) :: L), tcp: L*, ATTS >
=>
< A : U | sessionSet: (SSET ;; [SID : Session | commPartner: B, state: 4, sKey: SKEY, mKey: MKEY, ATTS*]),
ssl: L, tcp: ((msg toString(SID)- sEncrypt((msg MSG to B from A) ; mac((msg MSG to B from A), MKEY))with SKEY to B from A) :: L*), ATTS > .

*** Receive ssl-msg
rl[record_protocol_receive]:
< A : U | sessionSet: (SSET ;; [SID : Session | commPartner: B, state: 4, sKey: SKEY, mKey: MKEY, ATTS*]),
received: L, tcpReceived: ((msg toString(SID)- sEncrypt(M ; mac(M, MKEY))with SKEY to A from B) :: L*), ATTS >

=>
< A : U | sessionSet: (SSET ;; [SID : Session | commPartner: B, state: 4, sKey: SKEY, mKey: MKEY, ATTS*]),
received: (L :: M), tcpReceived: L*, ATTS > .
endm

LOGIN1.maude

```
in SSL

mod LOGIN1 is
pr SSL .

var ATTS ATTS* : AttributeSet .
var PW : Password .
var SET : Set .
var WS AS CLIENT : Oid . ***WS=Webserver, AS=ApplServer
var T T* TI TI* N N* : Nat .
var SID : Sid .
var SKEY CKEY : SymmKey .
var L L* LI : List .
var E : Element .
var CK CK* : Cookie .
var DATA : Data .
var H : Header .
```

```
op newest : Cookie Cookie -> Cookie [comm] .
eq newest(null, CK) = CK .
eq newest(CK, null) = CK .
eq newest((sEncrypt authTicket(CLIENT, T, T*, N) with CKEY), (sEncrypt
authTicket(CLIENT, TI, TI*, N*) with CKEY)) =
 if (N >= N*) then (sEncrypt authTicket(CLIENT, T, T*, N) with CKEY) else
(sEncrypt authTicket(CLIENT, T, T*, N*) with CKEY) fi .

***Client sends formLogin level 1 request to server.
rl[login1_request]:
< WS : WebServer | ATTS >
< CLIENT : Client | password: PW, request: ("login1" :: L), ssl: L*, ATTS* >
=>
< CLIENT : Client | password: PW, request: (L), ssl: (L* :: (msg "login1" - PW to WS
from CLIENT)), ATTS* >
< WS : WebServer | ATTS >
 .

**********Authentication**********

***Webserver receives loginrequest and forwards request to Applicationsserver for
validation
rl[login1_receive]:
< WS : WebServer | received: ((msg "login1" - PW to WS from CLIENT) :: L ), ATTS
>
< AS : ApplServer | ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T + 1 >
(aMsg "login1" - PW ; CLIENT to AS from WS) .


***ApplServer checks Level1 login information and sends Authenication cookie back
to WS
crl[login1_validation]:
(aMsg "login1" - PW ; CLIENT to AS from WS)
< AS : ApplServer | passwordSet: SET, ATTS >
=>
< AS : ApplServer | passwordSet: SET, ATTS >
```

(aMsg "okL1" - CLIENT to WS from AS)
if (SET contains (CLIENT & PW)) .

***Webserver sends general welcome information
***WS forwards the cookie over ssl
rl[login1_sucess]:
(aMsg "okL1" - CLIENT to WS from AS)
< WS : WebServer | content: (E :: L), ssl: L* , cKey: CKEY, ATTS >
<Time | tick: T >
=>
< WS : WebServer | content: (E :: L), ssl: (L* :: (msg "authCookie" - authCookie(CLIENT , T , CKEY, 1) to CLIENT from WS) ::
(msg "respond" - E to CLIENT from WS)), cKey: CKEY, ATTS >
<Time | tick: T + 1 > .

**********Cookie authentication **********

***Client saves cookie
rl[save_cookie] :
< CLIENT : Client | received: ((msg "authCookie" - CK to CLIENT from WS) :: L), authCookie: CK*, ATTS* >
=>
< CLIENT : Client | received: L, authCookie: newest (CK*, CK), ATTS* > .

***Client wish to recieve information and sends get-message (cookie-authentication)
***Only if client got a cookie
crl[get]:
< WS : WebServer | ATTS >
< CLIENT : Client | request: ("get" :: L), authCookie: CK, ssl: L*, ATTS* >
=>
< WS : WebServer | ATTS >
< CLIENT : Client | request: (L), authCookie: CK, ssl: (L* ::(msg "get" - CK to WS from CLIENT)), ATTS* >
if CK =/= null .

***Webserver responds
*** Server sends all content other than welcome.
crl[respond]:
< WS : WebServer | received: ((msg "get" - CK to WS from CLIENT) :: L), content: (E :: L*), ssl: LI, cKey: CKEY, ATTS >
<Time | tick: T >
=>

```
< WS : WebServer | received: L, content: ( E :: L* ), ssl:( LI :: (msg "respond" - L* to CLIENT from WS)),cKey: CKEY, ATTS >
<Time | tick: T + 1 >
if validCookie1(CK, CLIENT, T, CKEY) ***parameters: cookie, client, time, cookie-key
 .

***Client saves info
rl[save]:
< CLIENT : Client | received: ((msg "respond" - DATA to CLIENT from WS) :: L), obtainedContent: SET, ATTS* >

=>
< CLIENT : Client | received: L, obtainedContent: ((DATA from WS) ;; SET), ATTS* >
 .
endm
```

LOGIN2.maude

```
in SSL
in LOGIN1 .
mod LOGIN2 is
pr SSL .
pr LOGIN1 .


var CLIENT WS AS : Oid .
var L L* LI : List .
var ATTS ATTS* : AttributeSet .
var T T* N : Nat .
var PW : Password .
var SET : Set .
var OTP : OneTimePair .
var E : Element .
var CKEY : SymmKey .
var CK : Cookie .
eq pw(N) = "password" + string(N, 5) .


***Client sends formLogin level 2 request to server (through ssl)
rl[login2_request]:
```

< WS : WebServer | ATTS >
< CLIENT : Client | request: ("login2" :: L), ssl: L*,ATTS* >
<Time | tick: T >
=>
< CLIENT : Client | request: (L), ssl: (L* :: (msg "login2" to WS from CLIENT)), ATTS* >
< WS : WebServer | ATTS >
<Time | tick: T > .

***Webserver receives loginrequest and forwards request to Applicationsserver
rl[login2_receive]:
< WS : WebServer | received: ((msg "login2" to WS from CLIENT) :: L), ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T + 1 >
(aMsg "login2" ; CLIENT to AS from WS) .

***ApplServer checks Level1 login information and sends Authenication cookie back to WS
rl[login2_password_generate]:
(aMsg "login2" ; CLIENT to AS from WS)
< AS : ApplServer | passwordGenerator: N, oneTimeSet: SET, ATTS >
<Time | tick: T >
=>
< AS : ApplServer | passwordGenerator: (N + T),oneTimeSet: (SET ;; (CLIENT & pw(N) & (T + 60) & 0)), ATTS >
<Time | tick: T >
(sms pw(N) to CLIENT from WS) .

***Client receives onetime-password and sends it through SSL
rl[login2_password_send]:
(sms PW to CLIENT from WS)
< CLIENT : Client | ssl: L, ATTS* >
< WS : WebServer | ATTS >

=>

< CLIENT : Client | ssl: (L :: (msg "login2" - PW to WS from CLIENT)), ATTS* >
< WS : WebServer | ATTS > .

***Webserver receives one-time-password
rl[login2_password_receive]:
< WS : WebServer | received: ((msg "login2" - PW to WS from CLIENT) :: L), ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T + 1 >
(aMsg "login2" - PW ; CLIENT to AS from WS) .

***ApplServer validates one-time-password
crl[login2_password_receive]:
(aMsg "login2" - PW ; CLIENT to AS from WS)
< WS : WebServer | ATTS >
< AS : ApplServer | oneTimeSet: ((CLIENT & PW & T* & N) ;; SET), ATTS* >
<Time | tick: T >
=>
< WS : WebServer | ATTS >
< AS : ApplServer | oneTimeSet: SET, ATTS* >
<Time | tick: T >
(aMsg "okL2" - CLIENT to WS from AS)
if T <= T* and N < 3 .

***Webserver sends general welcome information
***WS forwards the (new) cookie over ssl
rl[login2_sucess]:
(aMsg "okL2" - CLIENT to WS from AS)
< WS : WebServer | content2: (E :: L), ssl: L*, cKey: CKEY, ATTS >
<Time | tick: T >
=>
< WS : WebServer | content2: (E :: L), ssl: (L* :: (msg "authCookie" - authCookie(CLIENT, T, CKEY, 2) to CLIENT from WS) ::
(msg "respond" - E to CLIENT from WS)), cKey: CKEY, ATTS >
<Time | tick: T + 1 > .

***Client wish to recieve information on Level 2 and sends get-message with cookie after receiveing welcome-message for Level2
crl[get2]:
< WS : WebServer | ATTS >
< CLIENT : Client | request: ("get2" :: L), authCookie: CK, ssl: L*, obtainedContent: (("Welcome-L2" from WS) ;; SET), ATTS* >
=>
< WS : WebServer | ATTS >
< CLIENT : Client | request: L, authCookie: CK, ssl: (L* :: (msg "get2" - CK to WS from CLIENT)),
obtainedContent: (("Welcome-L2" from WS) ;; SET), ATTS* >
if CK =/= null .

***Webserver responds
*** Server sends all content other than welcome.
crl[respond2]:
< WS : WebServer | received: ((msg "get2" - CK to WS from CLIENT) :: L), content2: (E :: L*), ssl: LI, cKey: CKEY, ATTS >
<Time | tick: T >
=>
< WS : WebServer | received: L, content2: (E :: L*), ssl: (LI :: (msg "respond" - L* to CLIENT from WS)),cKey: CKEY, ATTS >
<Time | tick: T + 1 >
if validCookie2(CK, CLIENT, T, CKEY) .

endm

FORMEXCHANGE.maude

in SSL
mod FORMEXCHANGE is
pr SSL .

op combi : Set Set -> Set .
op combi : Set Set Set -> Set .

var CLIENT WS AS DB : Oid .
var ATTS ATTS* : AttributeSet .
var SET SET* SSET SSET* : Set .

```
var CK : Cookie .
var T N LVL LVL* : Nat .
var L L* : List .
var CKEY : SymmKey .
var NAME FUNCTION OP ROLE FORM FORM* : Element .
var NAME* FUNCTION* OP* ROLE* : Element .
var MSG : MsgContent .
var H : Header .
var E : Element .
var DATA : Data .
var DATALIST : DataList .

***Client sends request for valid options on webserver
crl[getOptions]:
< CLIENT : Client | request: ("options" :: L), authCookie: CK, ssl: L*, ATTS >
< WS : WebServer | ATTS* >
=>
< CLIENT : Client | request: L, authCookie: CK, ssl: (L* :: (msg "options" - CK to WS from CLIENT)), ATTS >
< WS : WebServer | ATTS* >
if CK =/= null .

***Webserver responds by sending sets for valid functions and formnames
crl[respond_options]:
< WS : WebServer | received:((msg "options" - CK to WS from CLIENT) :: L), formNames: SET, functions: SET*, ssl: L*, cKey: CKEY, ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, formNames: SET, functions: SET*, ssl: (L* :: (msg "options" - SET ; SET* to CLIENT from WS )), cKey: CKEY, ATTS* >
<Time | tick: T + 1 >
if validCookie1(CK, CLIENT, T, CKEY) .

***Client saves and generate all combinations of forms and functions
*** if client have saved formNames and functions earlier, thoose will be overwritten
*** combinations:combi(SET, SET*)-------------------------------------------
rl[save_options]:
< CLIENT : Client | received: ((msg "options" - SET ; SET* to CLIENT from WS ) :: L), formNames: SSET, functions: SSET*, combinations: emptySet, ATTS >
=>
< CLIENT : Client | received: L, formNames: SET, functions: SET*, combinations: combi(SET, SET*), ATTS > .
```

```
op _&_ : String String -> Element .

***Client requests a form with all functions received, then next form with all functions...
***do this as long there is combinations left to try
rl[request_forms]:
< CLIENT : Client | ssl: L, authCookie: CK, request: ("form" :: L*), combinations: ((FUNCTION & NAME) ;; SET), ATTS >
< WS : WebServer | ATTS* >
=>
< CLIENT : Client | ssl: (L :: (msg "form" - (CK ; FUNCTION ; NAME) to WS from CLIENT)),authCookie: CK,
 request: (if SET == emptySet then L* else ("form" :: L*) fi), combinations: SET, ATTS >
< WS : WebServer | ATTS* > .

***Webserver receive a requst and checks for valid cookie
*** then forwards request to Appl-server for further validation
crl[receive_formRequest] :
< WS : WebServer | received: ((msg "form" - CK ; DATALIST to WS from CLIENT) :: L), cKey: CKEY, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, cKey: CKEY, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T + 1 >
(aMsg "form" - DATALIST ; CLIENT ; toString(getLevel(CK)) to AS from WS)
if (validCookie1(CK, CLIENT, T, CKEY)) .

***Applserver authorization check
crl[authorize_formRequest]:
(aMsg "form" - FUNCTION ; NAME ; CLIENT ; toString(LVL) to AS from WS)
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET), roleList: ([ OP, NAME, ROLE, LVL*] ;; SET*),
userRoles: ([CLIENT, ROLE] ;; SSET ), ATTS* >
< DB : Database | ATTS >
=>
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET), roleList: ([ OP, NAME, ROLE, LVL*] ;; SET*),
userRoles: ([CLIENT, ROLE] ;; SSET ), ATTS* >
```

```
< DB : Database | ATTS >
 (if OP == "Read" then (aMsg "form" - NAME ; CLIENT to DB from AS )
         else (aMsg "form" - FUNCTION ; NAME ; CLIENT to DB from AS ) fi)
if LVL* <= LVL .

*** Database send form - read only
rl[send_form]:
(aMsg "form" - NAME ; CLIENT to DB from AS )
< DB : Database | forms: ([NAME, FORM] ;; SET), ATTS >
< WS : WebServer | ATTS* >
=>
(aMsg "form" - FORM ; CLIENT to WS from DB )
< WS : WebServer | ATTS* >
< DB : Database | forms: ([NAME, FORM] ;; SET), ATTS > .

***Database send form - writeable
rl[send_form]:
(aMsg "form" - FUNCTION ; NAME ; CLIENT to DB from AS )
< WS : WebServer | ATTS* >
< DB : Database | forms: ([NAME, FORM] ;; SET), ATTS >
=>
(aMsg "form" - FORM ; FUNCTION ; NAME ; CLIENT to WS from DB )
< WS : WebServer | ATTS* >
< DB : Database | forms: ([NAME, FORM] ;; SET), ATTS > .

***Webserver forwards forms to client
rl[forward_form]:
< WS : WebServer | ssl: L, ATTS >
(aMsg "form" - DATA ; CLIENT to WS from DB )
<Time | tick: T >
=>
< WS : WebServer | ssl: (L :: (msg "form" - DATA to CLIENT from WS) ), ATTS >
<Time | tick: T + 1 > .

***Client receive and save read-only form
rl[receive_form]:
< CLIENT : Client | received: ((msg "form" - FORM to CLIENT from WS) :: L),
obtainedContent: SET, ATTS >
< WS : WebServer | ATTS* >
=>
< WS : WebServer | ATTS* >
```

< CLIENT : Client | received: L, obtainedContent: (SET ;; (FORM from WS)), ATTS > .

***Client recieve, save and write form
***sends cookie ; updated form ; function ; formName
rl[receive_form_and_change]:
< CLIENT : Client | received: ((msg "form" - FORM ; FUNCTION ; NAME to CLIENT from WS) :: L),
obtainedContent: SET, ssl: L*, authCookie: CK, ATTS >
< WS : WebServer | ATTS* >
=>
< WS : WebServer | ATTS* >
< CLIENT : Client | received: L, obtainedContent: (SET ;; ((FORM ; FUNCTION ; NAME)from WS)),
ssl: ((msg "changeForm" - CK ; (FORM + "_" + FUNCTION); FUNCTION ; NAME to WS from CLIENT) :: L*), authCookie: CK, ATTS >
.

***Webserver receives Write requst, check for valid cookie and forward request to Appl-server for further validation
crl[receive_changeFormRequest]:
< WS : WebServer | received: ((msg "changeForm" - CK ; DATA to WS from CLIENT) :: L), cKey: CKEY, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, cKey: CKEY, ATTS >
< AS : ApplServer | ATTS* >
<Time | tick: T + 1 >
(aMsg "changeForm" - DATA ; CLIENT ; toString(getLevel(CK)) to AS from WS)
if (validCookie1(CK, CLIENT, T, CKEY)) .

***Validation by Applserver
crl[validate_formChange]:
(aMsg "changeForm" - FORM* ; FUNCTION ; NAME ; CLIENT ; toString(LVL) to AS from WS)
< AS : ApplServer | acl: ([FUNCTION, OP] ;; SET), roleList: ([OP, NAME, ROLE, LVL*] ;; SET*), userRoles: ([CLIENT, ROLE] ;; SSET), ATTS* >
< DB : Database | ATTS >
=>
< DB : Database | ATTS >

```
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET), roleList: ([ OP, NAME, ROLE,
LVL*] ;; SET*), userRoles: ([CLIENT, ROLE] ;; SSET ), ATTS* >
(aMsg "changeForm" - FORM* ; FUNCTION ; NAME ; CLIENT to DB from AS )
if LVL >= LVL* .


***Database update
*** if function == Formfilling then make a new copy of the form ...
crl[update_form]:
(aMsg "changeForm" - FORM* ; FUNCTION ; NAME ; CLIENT to DB from AS )
< DB : Database | forms: ([NAME, FORM] ;; SET), numbering: N, ATTS >
=>
< DB : Database | forms: ([ (NAME + "." + string(N, 5)), FORM*] ;; [NAME, FORM]
;; SET), numbering: (N + 1), ATTS >
if FUNCTION == "FormFilling" .

***...else change the form
crl[update_form]:
(aMsg "changeForm" - FORM* ; FUNCTION ; NAME ; CLIENT to DB from AS )
< DB : Database | forms: ([NAME, FORM] ;; SET), ATTS >
=>
< DB : Database | forms: ([NAME, FORM*];; SET), ATTS >
if FUNCTION == "FormFilling" .

***Take a name and combine with all functions.
eq combi((NAME ;; SET), (FUNCTION ;; SET*)) = (FUNCTION & NAME) ;;
combi((NAME ;; SET), SET*, FUNCTION) .
eq combi((NAME ;; SET), (FUNCTION ;; SET*), SSET) = (( FUNCTION & NAME)
;; combi((NAME ;; SET), SET*, (FUNCTION ;; SSET))) .
***next name and combine with all functions
eq combi((NAME ;; SET), emptySet, SSET) = combi(SET, SSET) .
***finished
eq combi(emptySet, SET) = emptySet .
endm
```

ADVERSARY.maude

```
in CONFIG_LOGIN1 .
in SEARCH .

mod ADVERSARY is
pr CONFIG_LOGIN1 .
```

```
pr SEARCH .

var T : Nat .
var ATTS ATTS* ATTSW : AttributeSet .
var L : List .
var SET SET* : Set .
var MSG : MsgContent .
var A B CLIENT SERVER : Oid .
var M : Msg .
var H : Header .
var DATA : Data .
var N : Nat .

op msgIntercepted:_ : Set -> Attribute [format(ny o o )] .
op keySet:_ : Set -> Attribute .
op Adversary : -> Oid .
op msgSpoofed:_ : Set -> Attribute [format(ny o o )] .
op msgUnderstood:_ : Set -> Attribute [format(ny o o )] .

op initA : -> Configuration .

ceq <Time | tick: T > = <Time | tick: 0 > if T > 0 .

eq initA =
initL1(1)
< Adversary : Client |
request: ("startTCP" :: "startSSL"),
ssl: nil, sessionSet: emptySet, trustedIssuerSet: Issuer1, received: nil,
tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: 0, msgIntercepted: emptySet,
 msgSpoofed: emptySet, msgUnderstood: emptySet > .

***intercept all messages
ceq < Adversary : Client | msgIntercepted: SET, ATTS >
(msg MSG to A from B )
=
< Adversary : Client | msgIntercepted: (SET ;; (msg MSG to A from B )), ATTS >
(msg MSG to A from B )
if not(SET contains (msg MSG to A from B )) .

***spoof client
ceq
```

< Adversary : Client | msgIntercepted: ((msg MSG to A from B) ;; SET), msgSpoofed: SET*, ATTS >
(msg MSG to A from B)
< CLIENT : Client | ATTS* >
=
< Adversary : Client | msgIntercepted: ((msg MSG to A from B) ;; SET),msgSpoofed:((msg MSG to A from B) ;; SET*), ATTS >
(msg MSG to A from B)
(msg MSG to A from CLIENT)
< CLIENT : Client | ATTS* >
if not(SET* contains (msg MSG to A from B)) and not (A == Adversary) .

endm

ERRORHANDLER.maude

```
mod ERRORHANDLER is
inc FORMEXCHANGE .

var WS CLIENT SERVER AS A B CERT CERT* ISSUER : Oid .
var SET SET*  SSET TSET KSET KSET* : Set .
var L L* LI MSGL CL LL :List .
var H : Header .
var DATA : Data .
var ATTS ATTS* ATTSL : AttributeSet .
var CK : Cookie .
var CKEY : SymmKey . ***cookie-key
var E : Element .
var T T* N LVL LVL* : Nat .
var MSG : MsgContent .
var PW PW* : Password .
var FUNCTION NAME OP ROLE KU KR KU* KR* : String .
var RANDOM* : Nat .

***Log class
op <Log |_> : AttributeSet -> Object[ctor format(nb b o b bo)] .

***Log attributes
op headerErrorSSL:_ : List -> Attribute .
op authorizeError:_ : List -> Attribute .

***Webserver has received a message with invalid header
```

```
crl[error_header_ssl1] :
< WS : WebServer | received: ((msg H - DATA to WS from CLIENT ) :: L), validHeaders: SET, ATTS >
<Log | headerErrorSSL: L*, ATTS* >
<Time | tick: T >
=>
< WS : WebServer | received: L, validHeaders: SET, ATTS >
<Time | tick: T + 1 >
<Log | headerErrorSSL: ( L* :: (msg H - DATA to WS from CLIENT)), ATTS* >
if ( not (SET contains H ) ) .

***Webserver has received a message with invalid header
crl[error_header_ssl2] :
< WS : WebServer | received: ((msg H to WS from CLIENT ) :: L), validHeaders: SET, ATTS >
<Log | headerErrorSSL: L*, ATTS* >
<Time | tick: T  >
=>
< WS : WebServer | received: L, validHeaders: SET, ATTS >
<Log | headerErrorSSL: ( L* :: (msg H to WS from CLIENT)), ATTS* >
<Time | tick: T  >
if ( not (SET contains H ) ) .

*****FormExchange not authorized
crl[not_authorized] :
(aMsg "form" - FUNCTION ; NAME ; CLIENT ; toString(LVL) to AS from WS)
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET), roleList: ([ OP, NAME, ROLE, LVL*] ;; SET*),
userRoles: ([CLIENT, ROLE] ;; SET ), ATTS* >
<Log | authorizeError: L*, ATTS >
=>
< AS : ApplServer | acl: ([FUNCTION, OP ] ;; SET), roleList: ([ OP, NAME, ROLE, LVL*] ;; SET*),
userRoles: ([CLIENT, ROLE] ;; SET ), ATTS* >
<Log | authorizeError: (L* :: (aMsg "form" - FUNCTION ; NAME ; CLIENT ; toString(LVL) to AS from WS) ), ATTS >
if LVL* > LVL .

*****LOGIN2
***Remove one-time passwords if expired or to many trials
crl[expired_one_time_password]:
<Time | tick: T >
```

```
< AS : ApplServer | oneTimeSet: ((CLIENT & PW & T* & N ) ;; SET), ATTS* >
=>
< AS : ApplServer | oneTimeSet: SET, ATTS* >
<Time | tick: T >
if T > T* or N >= 3 .

***Invalid password
crl[error_login2_password_receive]:
(aMsg "login2" - PW ; CLIENT to AS from WS)
< AS : ApplServer | oneTimeSet: ((CLIENT & PW* & T* & N ) ;; SET), ATTS* >
=>
< AS : ApplServer | oneTimeSet:((CLIENT & PW* & T* & ( N + 1 )) ;; SET), ATTS*
>
if PW =/= PW* .

op initLog : -> Configuration .
eq initLog =
<Log | headerErrorSSL: nil, authorizeError: nil > .
endm
```

RANDOM.maude

```
***(
Implementasjon fra Numerical Recipes in C, side 278
***)
fmod RANDOM is

   protecting NAT .

   op rand : Nat -> Nat .

   op seed : -> Nat .
   eq seed = 1 . *** Kan være hvilket som helst positivt oddetall

   ops a m R : -> Nat .
   eq a = 16807 . *** 7^5
   eq m = 2147483647 . *** 2 ^ 31 - 1
   eq R = 8 .
   var N : Nat .
   eq rand(N) = ( a * N ) rem m  .

endfm
```

Appendix C

I dette kapittelet finnes CONFIG-filene som er beskrevet i oppgaven, og disse inneholder forskjellige initialtilstander.

CONFIG_SSL

```
in SSL
in FORMEXCHANGE
in ERRORHANDLER


mod CONFIG_SSL is
pr SSL .
pr ERRORHANDLER .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var ATT ATT* : Attribute .

op Client_ : Nat -> Oid .
op numClients : -> Nat .
op pw : Nat -> String[ctor] .
eq pw(N) = "password" + string(N, 5) .
```

```
***Initialize:
op initClients : Nat -> Configuration .
ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1  : -> Oid .


op Client1 : -> Oid .

op initSSL : Nat  -> Configuration .
eq initSSL(N) =
initClients(N)
initLog
initPKI

<Time | tick: 0 >

***Server
< WebServer1 : WebServer |
ssl: nil, sessionSet: emptySet , privKey: "KR",  received: nil, randomGenerator:
rand(100),
tcp: nil , tcpSessions: emptySet, tcpReceived: nil,
cKey: symmKey(999999)  >  .


***Clients
eq initClients(N) =
< Client N : Client |  password: pw(N),
request: ("startTCP" :: "startSSL" ),
ssl: nil , sessionSet: emptySet,  trustedIssuerSet: Issuer1,  received: nil,
tcp: nil, authCookie: null , tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N) >
initClients(N - 1)  .
eq initClients(0) = none .


op initPKI : -> Configuration .
***PKI structure

eq initPKI =
***WebServer certificate:
```

< Cert : Certificate | issuerName: Issuer1, subjectName: WebServer1, subjectKu: "KU",
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >

***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") > .

endm

CONFIG_LOGIN1

in FORMEXCHANGE
in ERRORHANDLER
in LOGIN1
in CONFIG_SSL

mod CONFIG_LOGIN1 is
pr SSL .
pr LOGIN1 .
pr ERRORHANDLER .
pr CONFIG_SSL .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .

```
op Client_ : Nat -> Oid .
op pw : Nat -> String[ctor] .
eq pw(N) = "password" + string(N, 5) .

ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1  : -> Oid .
op numClients : -> Nat .

***Initialize:
***-------------------------------------------------------------------------------
op initClients : Nat -> Configuration .
op initPWSet : Nat -> Set .
op initL1 : Nat -> Configuration .

eq initL1(N)  =
initClients(N)
initLog
initPKI

<Time | tick: 0 >
***Servers
< WebServer1 : WebServer | sessionSet: emptySet , privKey: "KR",   received: nil,
content: ("Welcome-L1" :: "Other-L1"),
randomGenerator: rand(100), validHeaders: ("get" ;; "login1" ),
 cKey: symmKey(999999), ssl: nil, tcp: nil, tcpSessions: emptySet, tcpReceived: nil  >

< ApplServer1 : ApplServer | passwordSet: initPWSet(N) >
  .

***Clients
eq initClients(N) =
< Client N : Client |  password: pw(N),
request: ("startTCP" :: "startSSL" :: "login1" :: "get" ), obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet,  trustedIssuerSet: Issuer1,  received: nil,
tcp: nil, authCookie: null , tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N) >  initClients(N - 1)  .

eq initClients(0) = none .
eq initPWSet(0) = emptySet .
eq initPWSet(N) = (Client N  & pw(N) ) ;; initPWSet(N - 1) .

endm
```

CONFIG_LOGIN2

```
in LOGIN2
in FORMEXCHANGE
in ERRORHANDLER


mod CONFIG_LOGIN2 is
pr LOGIN2 .
pr ERRORHANDLER .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .

op Client_ : Nat -> Oid .



***_________________initL2 - CONFIGURATION_____________
ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1  : -> Oid .
op initClients : Nat -> Configuration .

op initL2 : Nat -> Configuration .
eq initL2(N) =
initLog
initClients(N)

***Servers
< WebServer1 : WebServer |
content2: ("Welcome-L2" :: "Other-L2"),  validHeaders: ("login2" ;; "get2" ),
cKey: symmKey(999999),  randomGenerator: rand(100),
ssl: nil, sessionSet: emptySet , privKey: "KR",  received: nil,
tcp: nil, tcpSessions: emptySet, tcpReceived: nil >
```

```
< ApplServer1 : ApplServer |  passwordGenerator: 10000, oneTimeSet: emptySet  >

***WebServer certificate:
< Cert : Certificate | issuerName: Issuer1, subjectName: WebServer1, subjectKu: "KU",
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >

***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") >
<Time | tick: 0 > .


***Clients
eq initClients(N) =
< Client N : Client |
request: ("startTCP" :: "startSSL" :: "login2" :: "get2" ), authCookie: null ,
obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet, trustedIssuerSet: Issuer1, received: nil,
tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N) >
initClients(N - 1)  .

eq initClients(0) = none .


endm
```

CONFIG_FORMEXC1 (v.1)

```
***CONFIG_FORMEXC1 v.1
in LOGIN1
in FORMEXCHANGE
in ERRORHANDLER
```

```
in CONFIG_SSL

mod CONFIG_FORMEXC1 is
pr LOGIN1 .
pr ERRORHANDLER .
pr FORMEXCHANGE .
pr CONFIG_SSL .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .
var REC : Record .
var LVL : Nat .
var ATT ATT* : Attribute .
var L : List .

op Client_ : Nat -> Oid .
op _contains_ : AttributeSet Attribute -> Bool .
eq  ATT, ATTS contains ATT* = if ATT == ATT* then true else ATTS contains
ATT*  fi  .
eq none contains ATT = false .

op pw : Nat -> String[ctor] .
eq pw(N) = "password" + string(N, 5) .

***_____InitFormexc - CONFIGURATION_____

ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1 Database1 : -> Oid .
ops initAcl, initRoleList  : -> Set .
ops initFunctions initFormNames initForms : Set -> Set .
op initClients : Nat -> Configuration .
op initPWSet : Nat -> Set .
op initUserRoles : Nat -> Set .

op initF1 : Nat -> Configuration .
eq initF1(N) =
```

```
initClients(N)
initLog
initPKI
<Time | tick: 0 >

***WebServer
< WebServer1 : WebServer | sessionSet: emptySet , privKey: "KR",  received: nil,
content: ("Welcome-L1" :: "Other-L1"),
randomGenerator: rand(100), validHeaders: ("get" ;; "login1"  ;; "options" ;; "form" ;;
"changeForm"),
 cKey: symmKey(999999), ssl: nil, tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
functions: initFunctions(initAcl) , formNames: initFormNames(initRoleList) >

***ApplicationServer
< ApplServer1 : ApplServer | passwordSet: initPWSet(N), oneTimeSet: emptySet,
acl: initAcl, roleList: initRoleList, userRoles: initUserRoles(N)  >

***Database
< Database1 : Database | forms: initForms(initFormNames(initRoleList)), numbering: 1
>    .

***Clients
eq initClients(N) =
< Client N : Client |  password: pw(N),
request:  ("startTCP"  ::  "startSSL"  ::  "login1"  ::  "get"  ::   "options"  ::  "form"  ),
obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet,  trustedIssuerSet: Issuer1,  received: nil,
tcp: nil, authCookie: null , tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N),
functions: emptySet, formNames: emptySet, combinations: emptySet >
initClients(N - 1)  .

eq initClients(0) = none .
eq initPWSet(0) = emptySet .
eq initPWSet(N) = (Client N  & pw(N) ) ;; initPWSet(N - 1) .



eq  initAcl  =  ["FormFilling",  "Write"]  ;;  ["PDFViewer",  "Read"]  ;;
["UserAdministration", "Admin"] .
eq  initRoleList  =  ["Read",  "RF1",  "Reader",  1]  ;;  ["Write",  "RF1",  "Fillout",  1]  ;;
["Read", "RF2", "Reader", 2] .
```

```
eq initFormNames(emptySet) = emptySet .
eq initFormNames([S, NAME, S*, LVL] ;; SET ) = NAME ;; initFormNames(SET) .

eq initFunctions(emptySet) = emptySet .
eq initFunctions([FUNCTION , S] ;; SET ) = FUNCTION ;; initFunctions(SET) .

eq initForms(emptySet) = emptySet .
eq initForms(NAME ;; SET ) =  [NAME , ("theForm" + NAME) ] ;; initForms(SET) .

eq initUserRoles(N) =
([Client N , "Reader"] ;; [Client N , "Fillout"] ;; initUserRoles(N - 1)) .
eq initUserRoles(0) = emptySet .

endm
```

CONFIG_FORMEXC1 (v.2)

```
in LOGIN1
in FORMEXCHANGE
in ERRORHANDLER

mod CONFIG_FORMEXC1 is
pr LOGIN1 .
pr ERRORHANDLER .
pr FORMEXCHANGE .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .
var REC : Record .
var LVL : Nat .
var ATT ATT* : Attribute .
var L : List .

op Client_ : Nat -> Oid .
```

```
op _contains_ : AttributeSet Attribute -> Bool .
eq  ATT, ATTS contains ATT* = if ATT == ATT* then true else ATTS contains ATT*  fi  .
eq none contains ATT = false .

op pw : Nat -> String[ctor] .
eq pw(N) = "password" + string(N, 5) .
op numClients : -> Nat .



***_____InitFormexc - CONFIGURATION_____

ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1 Database1 : -> Oid .
ops initAcl, initRoleList  : -> Set .
ops initFunctions initFormNames initForms : Set -> Set .
op initClients : Nat -> Configuration .
op initPWSet : Nat -> Set .
op initUserRoles : Nat -> Set .

op initF1 : Nat -> Configuration .
eq initF1(N) =
initClients(N)
initLog

***WebServer
< WebServer1 : WebServer | sessionSet: emptySet , privKey: "KR",  received: nil,
content: ("Welcome-L1" :: "Other-L1"),
randomGenerator: rand(100), validHeaders: ("get" ;; "login1"  ;; "options" ;; "form" ;;
"changeForm"),
 cKey: symmKey(999999), ssl: nil, tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
functions: initFunctions(initAcl) , formNames: initFormNames(initRoleList) >


***ApplicationServer
< ApplServer1 : ApplServer | passwordSet: initPWSet(N), oneTimeSet: emptySet,
acl: initAcl, roleList: initRoleList, userRoles: initUserRoles(N)  >

***WebServer certificate:
< Cert : Certificate | issuerName: Issuer1, subjectName: WebServer1, subjectKu:
"KU",
```

```
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Database
< Database1 : Database | forms: initForms(initFormNames(initRoleList)), numbering: 1
>

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >

***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") >
<Time | tick: 0 >
 .

***Clients
eq initClients(N) =
< Client N : Client |  password: pw(N),
request:  ("startTCP"  ::  "startSSL"  ::  "login1"  ::  "get"  ::    "options"  ::  "form"  ),
obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet,  trustedIssuerSet: Issuer1,  received: nil,
tcp: nil, authCookie: null , tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N),
functions: emptySet, formNames: emptySet, combinations: emptySet >
initClients(N - 1)  .

eq initClients(0) = none .
eq initPWSet(0) = emptySet .
eq initPWSet(N) = (Client N  & pw(N) ) ;; initPWSet(N - 1) .



eq initAcl =  ["PDFViewer", "Read"] ;; ["FormFilling", "Write"] . ***rek!
eq initRoleList = ["Read", "RF1", "Reader", 1] ;;  ["Read", "RF2", "Reader", 2] .

eq initFormNames(emptySet) = emptySet .
eq initFormNames([S, NAME, S*, LVL] ;; SET ) = NAME ;; initFormNames(SET) .

eq initFunctions(emptySet) = emptySet .
```

```
eq initFunctions([FUNCTION , S] ;; SET ) = FUNCTION ;; initFunctions(SET) .

eq initForms(emptySet) = emptySet .
eq initForms(NAME ;; SET ) =  [NAME , ("theForm" + NAME) ] ;; initForms(SET) .


eq initUserRoles(N) =
([Client N , "Reader"] ;;  [Client N , "Fillout"]  ;; initUserRoles(N - 1)) . ***ok
eq initUserRoles(0) = emptySet .

endm
```

CONFIG_FORMEXC2 (v.1)

```
***CONFIG_FORMEXC v.1
in LOGIN2
in FORMEXCHANGE
in ERRORHANDLER


mod CONFIG_FORMEXC is
pr LOGIN2 .
pr ERRORHANDLER .
pr FORMEXCHANGE .


var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .
var REC : Record .
var LVL : Nat .
var ATT ATT* : Attribute .

op Client_ : Nat -> Oid .

op _contains_ : AttributeSet Attribute -> Bool .
```

```
eq  ATT, ATTS contains ATT* = if ATT == ATT* then true else ATTS contains ATT*  fi  .
eq none contains ATT = false .

op Client_ : Nat -> Oid .
op numClients : -> Nat .

***change this value to the number of clients wanted in configuration:
eq numClients = 2 .


***_____initFormexc2 - CONFIGURATION_____

ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1 Database1  : -> Oid .
ops initClients  : Nat -> Configuration .
ops initAcl, initRoleList  : -> Set .
ops initFunctions initFormNames initForms : Set -> Set .
op initUserRoles : Nat -> Set .

op initFormexc2 : -> Configuration .
eq initFormexc2 =

initClients(numClients)
initLog

***Servers
< WebServer1 : WebServer |
content2:  ("Welcome-L2"  ::  "Other-L2"),    validHeaders:  ("login2"  ;;  "get2"  ;;
"options" ;; "form" ;; "changeForm" ),
cKey: symmKey(999999),  randomGenerator: rand(100),
ssl: nil, sessionSet: emptySet , privKey: "KR",  received: nil,
tcp:  nil,  tcpSessions:  emptySet,  tcpReceived:  nil,  formNames:
initFormNames(initRoleList), functions: initFunctions(initAcl) >

< ApplServer1  :  ApplServer  |  passwordGenerator:  10000,  oneTimeSet:  emptySet,
oneTimeSet: emptySet,
acl: initAcl, roleList: initRoleList,  userRoles: initUserRoles(numClients) >

***WebServer certificate:
< Cert  :  Certificate  |  issuerName:  Issuer1,  subjectName:  WebServer1,  subjectKu:
"KU",
```

```
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Database
< Database1 : Database | forms: initForms(initFormNames(initRoleList)), numbering: 1
>

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >

***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") >
<Time | tick: 0 > .

***Clients
eq initClients(N) =
< Client N : Client |
request: ("startTCP" :: "startSSL" :: "login2" :: "get2"    :: "options" :: "form"),
authCookie: null , obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet, trustedIssuerSet: Issuer1, received: nil,
tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N),
functions: emptySet, formNames: emptySet, combinations: emptySet >
initClients(N - 1)  .

eq initClients(0) = none .

eq initAcl = ["FormFilling", "Write"] ;; ["PDFViewer", "Read"] ;;
["UserAdministration", "Admin"] .
eq initRoleList = ["Read", "RF1", "Reader", 1] ;; ["Write", "RF1", "Fillout", 1] ;;
["Read", "RF2", "Reader", 2] .

eq initFormNames(emptySet) = emptySet .
eq initFormNames([S, NAME, S*, LVL] ;; SET ) = NAME ;; initFormNames(SET) .

eq initFunctions(emptySet) = emptySet .
eq initFunctions([FUNCTION , S] ;; SET ) = FUNCTION ;; initFunctions(SET) .

eq initForms(emptySet) = emptySet .
```

```
eq initForms(NAME ;; SET ) =  [NAME , ("theForm" + NAME) ] ;; initForms(SET) .

eq initUserRoles(N) =
([Client N , "Reader"] ;; [Client N , "Fillout"] ;; initUserRoles(N - 1)) .
eq initUserRoles(0) = emptySet .

endm
```

CONFIG_FORMEXC2 (v.2)

```
***CONFIG_FORMEXC v.2
in LOGIN2
in FORMEXCHANGE
in ERRORHANDLER

mod CONFIG_FORMEXC is
pr LOGIN2 .
pr ERRORHANDLER .
pr FORMEXCHANGE .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var S S* NAME FUNCTION : String .
var REC : Record .
var LVL : Nat .
var ATT ATT* : Attribute .

op Client_ : Nat -> Oid .

op _contains_ : AttributeSet Attribute -> Bool .
eq  ATT, ATTS contains ATT* = if ATT == ATT* then true else ATTS contains
ATT*  fi  .
eq none contains ATT = false .

op Client_ : Nat -> Oid .

***_____initFormexc2 - CONFIGURATION_____
```

```
ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1 Database1  : -> Oid .
ops initClients  : Nat -> Configuration .
ops initAcl, initRoleList  : -> Set .
ops initFunctions initFormNames initForms : Set -> Set .
op initUserRoles : Nat -> Set .

op initF2 : Nat -> Configuration .
eq initF2(N) =

initClients(N)
initLog

***Servers
< WebServer1 : WebServer |
content2: ("Welcome-L2" :: "Other-L2"),  validHeaders: ("login2" ;; "get2" ;;
"options" ;; "form" ;; "changeForm" ),
cKey: symmKey(999999),  randomGenerator: rand(100),
ssl: nil, sessionSet: emptySet , privKey: "KR",  received: nil,
tcp:  nil,  tcpSessions:  emptySet,  tcpReceived:  nil,  formNames:
initFormNames(initRoleList), functions: initFunctions(initAcl) >

< ApplServer1 : ApplServer |  passwordGenerator: 10000, oneTimeSet: emptySet,
oneTimeSet: emptySet,
acl: initAcl, roleList: initRoleList,  userRoles: initUserRoles(N) >

***WebServer certificate:
< Cert : Certificate | issuerName: Issuer1, subjectName: WebServer1, subjectKu:
"KU",
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Database
< Database1 : Database | forms: initForms(initFormNames(initRoleList)), numbering: 1
>

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >
```

```
***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") >
<Time | tick: 0 > .

***Clients
eq initClients(N) =
< Client N : Client |
request: ("startTCP" :: "startSSL" :: "login2" :: "get2" :: "options" :: "form"),
authCookie: null , obtainedContent: emptySet,
ssl: nil , sessionSet: emptySet, trustedIssuerSet: Issuer1, received: nil,
tcp: nil, tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N),
functions: emptySet, formNames: emptySet, combinations: emptySet >
initClients(N - 1) .

eq initClients(0) = none .


eq initAcl = ["FormFilling", "Write"] ;; ["PDFViewer", "Read"] .
eq initRoleList = ["Read", "RF1", "Reader", 1] ;; ["Read", "RF2", "Reader", 2] .

eq initFormNames(emptySet) = emptySet .
eq initFormNames([S, NAME, S*, LVL] ;; SET ) = NAME ;; initFormNames(SET) .

eq initFunctions(emptySet) = emptySet .
eq initFunctions([FUNCTION , S] ;; SET ) = FUNCTION ;; initFunctions(SET) .

eq initForms(emptySet) = emptySet .
eq initForms(NAME ;; SET ) = [NAME , ("theForm" + NAME) ] ;; initForms(SET) .

eq initUserRoles(N) =
([Client N , "Reader"] ;; initUserRoles(N - 1)) .
eq initUserRoles(0) = emptySet .

endm
```

CONFIG_ADVERSARY

```
in SSL
in FORMEXCHANGE
in ERRORHANDLER
```

```
mod CONFIG_SSL is
pr SSL .
pr ERRORHANDLER .

var CLIENT SERVER A B : Oid .
var PWS SET : Set .
var ATTS ATTS* : AttributeSet .
var N : NzNat .
var C : Configuration  .
var MSG : MsgContent .
var PW : Password .
var ATT ATT* : Attribute .

op Client_ : Nat -> Oid .
op numClients : -> Nat .
op pw : Nat -> String[ctor] .
eq pw(N) = "password" + string(N, 5) .

***change this value to the number of clients wanted in configuration:
eq numClients =  1 .


***Initialize:
*op initClients : Nat -> Configuration .
ops Cert Cert* Issuer1 : -> Oid .
ops WebServer1 ApplServer1  : -> Oid .

op Client1 : -> Oid .

op initSSL : -> Configuration .
eq initSSL =
initClients(numClients)
****Servers
< WebServer1 : WebServer |
ssl: nil, sessionSet: emptySet , privKey: "KR",  received: nil, randomGenerator: rand(100),
tcp: nil , tcpSessions: emptySet, tcpReceived: nil,
cKey: symmKey(999999)  >

***WebServer certificate:
```

```
< Cert : Certificate | issuerName: Issuer1, subjectName: WebServer1, subjectKu:
"KU",
  signature: sign hash(Cert :: Issuer1 :: WebServer1 :: "KU") with "KR*" >

***Root certificate:
< Cert* : Certificate | issuerName: Issuer1, subjectName: Issuer1, subjectKu: "KU*",
  signature: (sign hash(Cert* :: Issuer1 :: Issuer1 :: "KU*") with "KR*") >

***Certificate revocation list
<CRL | list: nil , signature: (sign hash(nil) with "KR*") >

***Help objects
<ValidKeyPairs | keySet: ("KU*" & "KR*") ;; ("KU" & "KR") >
<Time | tick: 0 >

***Error Log
<Log | headerErrorSSL: nil, headerErrorPlain: nil, cookieError: nil, validationError: nil
>  .

***Clients
eq initClients(N) =
< Client N : Client |
request: ("startTCP" :: "startSSL" ),
ssl: nil , sessionSet: emptySet,  trustedIssuerSet: Issuer1,  received: nil,
tcp: nil,  tcpSessions: emptySet, tcpReceived: nil,
randomGenerator: rand(N) >
initClients(N - 1)  .

eq initClients(0) = none .

endm
```

Appendix D

Her er all Maude-kode som er knyttet til søk vedlagt.

SEARCH.maude

```
in UNITS
mod SEARCH is
pr UNITS .
op _contains_ : MsgContent String -> Bool .
op _contains_ : DataList String -> Bool .

var M : Msg .
var MSG : MsgContent .
var S  : String .
var DATA : Data .
var H : Header .
var DATALIST : DataList .

***Search for a String in MsgContent
eq (H - DATALIST) contains S = if H == S then true else (DATALIST contains S) fi .
eq ( DATA ; DATALIST ) contains S = if DATA == S then true else ( DATALIST contains S) fi .
eq (DATA contains S) =  if DATA == S then true else false fi .
endm
```

Søk i SSL

```
in SEARCH
in CONFIG_SSL

*** Is there a state where keys are sent as plaintext?
search[1] initSSL(2) =>*
< B:Oid : U:Unit | sessionSet: ([ SID:Sid : Session | sKey: symmKey(N:Nat), ATTS:AttributeSet] ;; SET:Set),  ATTS*:AttributeSet >
```

(msg MSG:MsgContent to A:Oid from B:Oid)
C:Configuration
such that (MSG:MsgContent contains toString(N:Nat)) .

***Is i possible for SSLhandshake to go furhter than state 0?
search initSSL(2) =>* < A:Oid : U:Unit | sessionSet: ([S:Sid : Session | state: N:Nat , ATTS:AttributeSet] ;; SET:Set), ATTS*:AttributeSet > C:Configuration
such that N:Nat > 0 .

Søk LOGIN1

in SEARCH
in CONFIG_LOGIN1

***Is there a state where clients doesnt succed login
search[1] initL1(2) =>! C:Configuration
< CLIENT:Oid : U:Unit | obtainedContent: SET:Set, ATTS >
such that not(SET contains ("Welcome-L1" from WebServer1)) .

***Is there a state where password is sent as plaintext?
search initL1(2) =>*
(msg MSG:MsgContent to A:Oid from B:Oid)
C:Configuration
such that MSG contains "password1" .

Søk i LOGIN2

in SEARCH .

***Is there a state where clients doesnt succeed login
search[1] initL2 =>! C:Configuration
< CLIENT:Oid : U:Unit | obtainedContent: SET:Set, ATTS >
such that not(SET contains ("Welcome-L2" from WebServer1))

.

search initL2 =>*
(msg MSG:MsgContent to A:Oid from B:Oid)
C:Configuration
such that MSG contains "get2"

.

Søk i Adversary

***Is there a state where the Adversary gets content from webserver that requers password?

***Search login 1
search[1]
initA
=>!
C:Configuration
< Adversary:Oid : U:Unit | msgIntercepted: SET:Set, obtainedContent: SET*:Set, ATTS:AttributeSet >
such that SET:Set contains "Welcome-L1" or SET*:Set contains "Welcome-L1" or SET:Set contains "Other-L1" or SET*:Set contains "Other-L1" .

***Search Login 2
search[1]
initA
=>!
C:Configuration
< Adversary:Oid : U:Unit | msgIntercepted: SET:Set, obtainedContent: SET*:Set, ATTS:AttributeSet >
such that SET:Set contains "Welcome-L2" or SET*:Set contains "Welcome-L2" or SET:Set contains "Other-L2" or SET*:Set contains "Other-L2" .

***Check that all ssl-connections can be established
***Used to check that its no error in the ADVERSARY module
***
search [1]
initA
=>!
C:Configuration
< A:Oid : U:Unit | sessionSet:([N*:Nat : Session | state: N:Nat, ATTSA:AttributeSet] ;; SETA:Set), ATTSA*:AttributeSet >
< B:Oid : U*:Unit | sessionSet: ([M*:Nat : Session | state: M:Nat, ATTSB:AttributeSet] ;; SETB:Set), ATTSB*:AttributeSet >
<

C:Oid : UC:Unit | sessionSet: ([O*:Nat : Session | state: O:Nat , ATTSC:AttributeSet] ;; SETC:Set), ATTSC*:AttributeSet >
such that N:Nat == 4 and M:Nat == 4 and O:Nat == 4 .