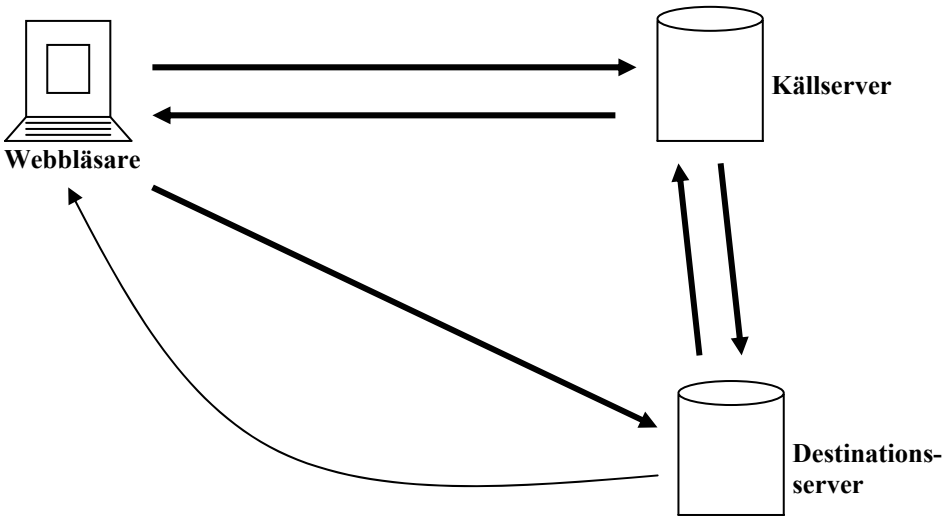
FOI-R--1025--SE
November 2003
ISSN 1650-1942
Användarrapport

Alf Bengtsson, Amund Hunstad, Lars Westerdahl

Identitetsverifiering över systemgränser



Ledningssystem
581 11 Linköping

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--1025--SE

November 2003

ISSN 1650-1942

Användarrapport

Alf Bengtsson, Amund Hunstad, Lars Westerdahl

Identitetsverifiering över systemgränser

Utgivare Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	**Rapportnummer, ISRN** FOI-R--1025--SE	**Klassificering** Användarrapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år November 2003	**Projektnummer** E7083
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 41 Ledning med samband och telekom och IT-system	
Författare/redaktör Alf Bengtsson Amund Hunstad Lars Westerdahl	**Projektledare** Alf Bengtsson	
	Godkänd av Martin Rantzer	
	Uppdragsgivare/kundbeteckning FM	
	Tekniskt och/eller vetenskapligt ansvarig Alf Bengtsson	

Rapportens titel

Identitetsverifiering över systemgränser

Sammanfattning (högst 200 ord)

Visionen om det nätverksbaserade försvaret kräver att ledningssystemet, FMLS, skall kunna samverka med externa system, t ex civila system eller andra nationers system. Vid hopkopplingen av delsystem aktualiseras flera olika säkerhetsfrågor. En av dem är möjligheten att över systemgränser autentisera, d v s verifiera identitet hos, aktörer från olika system. Med aktörer avses inte bara människor, utan också digitala aktörer såsom datorer, mobil programkod, radioutrustning m m.

Huvuddelen av rapporten beskriver Web Services och de delar, för stöd till identitetsverifiering, som standardiserats eller håller på att standardiseras. Eftersom området är under utveckling är rapporten huvudsakligen beskrivande snarare än värderande.

En slutsats är att man har kommit en bit på väg med identitetsverifiering mellan en mänsklig användare och en Web Service. Beträffande identitetsverifiering mellan Web Services är utvecklingen i sin linda.

Nyckelord

Autentisering, identitetsverifiering, webbtjänster, Web Services

Övriga bibliografiska uppgifter	**Språk** Svenska
ISSN 1650-1942	**Antal sidor:** 68 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--1025--SE	User report
	Programme Areas 4. C4ISR	
	Month year November 2003	**Project no.** E7083
	General Research Areas 5. Commissioned Research	
	Subcategories 41 C4I	

Author/s (editor/s)	Project manager
Alf Bengtsson Amund Hunstad Lars Westerdahl	Alf Bengtsson
	Approved by Martin Rantzer
	Sponsoring agency Swedish Defence
	Scientifically and technically responsible Alf Bengtsson

Report title (In translation)

Authentication across system borders

Abstract (not more than 200 words)

The vision of a network based defence demands that its C^4I-system, FMLS, shall co-operate with external systems, e.g. civilian systems or systems from other nations. When subsystems are connected a lot of security issues arise. One is authentication, i. e. verification of identities stated by different parties, across system borders. Parties to authenticate include not only humans, but also digital objects like computers, mobile software code, radio sets etc.

The main part of the report describes Web Services and developing standards for authentication within Web Services. Since this area is developing, the report is descriptive rather than assessing.

One conclusion is that authentication of a human user to a Web Service is under way, but authentication between Web Services is in its infancy.

Keywords

Authentication, Web Services

Further bibliographic information	Language Swedish
ISSN 1650-1942	**Pages** 68 p.
	Price acc. to pricelist

Innehåll

Figurer

1 Sammanfattning

Föreliggande rapport ingår i projektet ”System av system – säkerhetsfrågeställningar vid hopkopplingar”. Grundmotiveringen till projektet är att ledningssystemet i det framtida NBF av flera skäl inte kan bestå av ett enda system, inte ens ett distribuerat system, utan det kommer att bestå av flera samverkande delsystem. Inte minst uppkommer krav att samverka med externa system, till exempel civila system eller andra nationers system. Vid hopkopplingen av delsystemen aktualiseras flera olika säkerhetsfrågor. En av dem är möjligheten att över systemgränser autentisera, det vill säga verifiera identitet hos, aktörer från olika system. Med aktörer avses inte bara människor, utan också digitala aktörer såsom datorer, mobil programkod, radioutrustning med mera.

Huvuddelen av rapporten beskriver Web Services och de delar, för stöd till identitetsverifiering, som standardiserats eller håller på att standardiseras. Eftersom området är under utveckling är rapporten huvudsakligen beskrivande snarare än värderande.

I avsnitt 2 Bakgrund redovisas en förstudie för att avgränsa det vida begreppet ”system av system”. Den i förstudien valda avgränsningen innebär att studera Web Services, det implementationskoncept som växer fram för att via Internet koppla ihop system.

I avsnitt 3 beskrivs fundamenten för Web Services och för kommunikation mellan dem. Speciellt beskrivs de etablerade standarderna för kryptering och för digital signatur.

Avsnitt 4 beskriver XKMS. Denna standard är tänkt för hantering av öppna nycklar i olika varianter av PKI, Public Key Infrastructure. XKMS standardiserar kommunikationen till bakomliggande tjänster för att begära, validera och registrera nycklar. I avsnittet görs vissa referenser till krav från NBF beskrivna i andra sammanhang. Exempelvis medför tanken på bakomliggande tjänster on-line att förmågan till autonomitet minskar.

Avsnitt 5 beskriver SAML. Detta är ett format för säkerhetskritiska meddelanden av typ intyg (assertions). Exempel är ett intyg om att en aktör har autentiserat av en annan och vilken metod, nyckel med mera, som har använts. Sådana intyg skickas mellan olika aktörer, som måste kunna lita på att intygen är giltiga och att de är äkta.

I avsnitt 6 beskrivs tre koncept för identitetsverifiering. Microsoft .NET Passport beskrivs eftersom det är ett koncept som har produktifierats. Det är

dock dåligt ur säkerhetssynpunkt. Dessutom är det så hårt knutet till centraliserade register, att det inte är ett naturligt alternativ över systemgränser. Liberty Alliance är ett koncept som bygger på SAML, och som är mer distribuerat. Standardiseringen inom konceptets första fas har inriktats mot autentisering mellan en mänsklig användare och en Web Service. I en andra fas utvecklas autentisering mellan Web Services. Det tredje konceptet benämns WS-Security. Dess första fas inriktades på kommunikation av säkerhetsviktiga meddelanden, till exempel SAML-intyg, certifikat med flera. I en nyss påbörjad andra fas vill man bygga upp mer funktionalitet, bland annat WS-Federation för identitetshantering mellan system. Detta har stora likheter med Liberty Alliance, och det är naturligt att försöka samordna dessa. Detta i sin tur illustrerar läget inom utvecklingen av Web Services. Det är en teknik där mycket återstår att utveckla.

Slutsatsen är att gällande identitetsverifiering över systemgränser har man kommit en bit på väg med identitetsverifiering mellan en mänsklig användare och en Web Service. Beträffande identitetsverifiering mellan Web Services är utvecklingen i sin linda.

2 Bakgrund

Projektet inleddes med en förstudie, i första hand för att avgränsa det vida begreppet ”system av system”. Förstudien sammanfattas nedan i avsnitt 2.1. Slutsatsen i förstudien blev att vi fokuserar på Web Services, det implementationskoncept som växer fram med målet att via Internet koppla ihop system. I och med att området har ett gigantiskt kommersiellt intresse drivs utvecklingen på många fronter. Under förutsättning att man når fram till målet, gemensamma och användbara standarder, så kommer Web Services att ha inverkan även i militära system.

Web Services är alltså ett framväxande implementationskoncept. Det innehåller inga nya eller speciella säkerhetsfunktioner, till exempel för autentisering. I avsnitt 2.3 Autentisering refereras därför, mycket kortfattat, till de klassiska metoderna för autentisering.

2.1 Förstudie

I förstudien gjorde vi en konkretisering av begreppet ”system av system” enligt följande citat.

> Vår första avgränsning är att ”System av system” innebär att de ingående delsystemen skall vara helt kompletta och helt självförsörjande, och driftsläget i ett system skall inte vara beroende av driftsläget i ett annat system. Vid hopkoppling av system skall ingetdera systemet behöva ändras eller anpassas till det andra systemet. Hopkopplingen skall ske via mäklare (broker, mediator, gateway …) där eventuell anpassning sker. Infrastrukturen inom varje delsystem skall vara fullständig. Till exempel skall PKI-strukturen inom varje del vara komplett.
>
> Nästa steg vad gäller avgränsning rör vilken systemnivå som skall vara i fokus. Vi har diskuterat två alternativa nivåer, koalitionsnivå respektive tjänstenivå. Den högre av dessa nivåer, koalitionsnivån, innefattar bland annat det svåra problemet att hantera (och inte minst beskriva) tilltro, trust, mellan olika system eller organisationer. Tjänstenivån är mer av datorteknisk karaktär, hur man kan koppla ihop system utan att dessa måste förändras eller anpassas.

De båda egenhändigt myntade begreppen koalitionsnivå respektive tjänstenivå utvecklas.

> Koalitionsnivån, den skulle också kunna kallas federationsnivån, innebär att självständiga delsystem skall samverka och alltså utväxla information. Delsystemen är av samma karaktär och hanterar i huvudsak samma typ av information. Däremot skall delsystemen kunna ha olika sätt att beskriva sin respektive information. Till exempel kan det finnas olika sätt att beskriva delsystemens säkerhetspolicies. Den mäklare

> som finns mellan delsystemen skall kunna beskriva information, bland annat policies och andra regler, på ett sätt som kan tolkas och verifieras i alla delsystem.
>
> Med begreppet tjänstenivå menar vi en starkare fokusering på datatekniska metoder att knyta ihop system. Detta ligger helt i linje med den skisserade arkitekturen i FMLS. Man ser hela FMLS som ett antal tjänster som skall kunna anropas av olika aktörer, inbegripet anrop av andra tjänster. Anropen måste på vanligt sätt regleras av säkerhetspolicyn – autentisering, auktorisering, tillgänglighet et cetera. Detta synsätt är också naturligt vid hopknytning av självständiga system. Den nämnda mäklaren kan ses som en typisk tjänst.

Begreppet tjänstenivå utvecklas vidare.

> Tjänstebegreppet är ett ”innebegrepp” som flitigt används nästan synonymt med ett annat innebegrepp, Web Services (WS). Detta har växt fram ur det arbete, med gigantiskt kommersiellt intresse, som bedrivs för att få standarder för anrop till tjänster för e-handel. Det finns en oändlig mängd referenser till WS på Internet. En startpunkt är [OASIS].
>
> En första fråga är vad som är skillnaden mellan WS och det man brukar kalla distribuerade system. Den viktigaste skillnaden mellan WS och tidigare distribuerade system kan skrivas XML. Alla meddelanden som finns i de olika kommunikationsprotokollen inom WS är strukturerade med hjälp av XML. XML ger en mycket mer öppen och enkel (men också ineffektivare) kodning än de andra standarderna. Därmed får man bättre förutsättningar för till exempel plattformsoberoende.

Slutsatsen i förstudien är att vi fokuserar på Web Services, framför allt eftersom vi bedömer att det ligger i linje med arkitekturen inom FMLS. I det följande används förkortningen WS för Web Services.

2.2 Autentisering

Autentisering innebär verifiering av en förfäktad identitet. Trots alla välkända brister är det fortfarande vanligt att mänskliga aktörer autentiseras via någon form av lösenord. Men i system med större säkerhetskrav, till exempel FMLS, och med både mänskliga och digitala aktörer måste autentiseringen ske med bättre metoder. I [BEN01] och [BEN02] värderas tre klasser av metoder utifrån ett antal krav som utlästs ur visionen om NBF.

Alla tre klasserna bygger på att den anropande aktören styrker sin identitet genom att visa på innehav av en hemlighet, autentiseringsnyckeln. Den anropade aktören verifierar, med hjälp av en verifieringsnyckel, att anroparen måste inneha rätt hemlighet. Genom att på ett eller annat sätt knyta ihop verifieringsnycklar och identiteter uppnås identitetsverifiering.

WS innebär som nämnts inte några nya autentiseringsmetoder. Av de värderade tre metodklasserna är en klass, identitetsbaserad autentisering där veri-

fieringsnyckeln är lika med identiteten, inte användbar över systemgränser. I WS blir de två andra klasserna aktuella, biljettbaserad respektive certifikatbaserad autentisering.

Biljettbaserade metoder är utvecklade ur Kerberosmetoden [NEU93]. Principen är att en aktör autentiserar sig mot sin KAS, Kerberos Authentication Server, som måste vara on-line. KAS konstruerar biljetter, med kort giltighet, som skall fungera som passersedel när aktören begär åtkomst av en WS. För en detaljerad metodbeskrivning, biljettutseende, kryptering, nycklar et cetera, hänvisas till nämnda referenser. Det väsentliga för WS är att KAS måste vara on-line samt att det finns WS-standard för biljetterna (se SAML).

Certifikatbaserade metoder baseras på olika former av PKI, Public Key Infrastructure. Certifikat fyller i princip samma funktion som biljetter men har väsentligt längre giltighetstid. Därmed behöver inte utfärdaren CA, Certification Authority, nödvändigtvis vara on-line. Dessutom bygger metoden på öppen-nyckel kryptering. Därför behövs en öppen-nyckel infrastruktur för hopkoppling av verifieringsnycklar och identiteter. För detaljer se till exempel [SCH96] och [X509]. Man har sluppit det oönskade on-line-beroendet på bekostnad av en komplicerad PKI. Det finns WS-standard för överföring av certifikat (se SAML) samt för hantering av öppna nycklar (se XKMS).

3 Web Services

En försvenskad beteckning vore till exempel webbtjänster, eller kanske internettjänster. Begreppet Web Services är dock så allmänt etablerat att det är meningslöst att använda en försvenskad beteckning. De två leverantörsoberoende sammanslutningar som har störst inverkan på utvecklingen av WS är Organization for the Advancement of Structured Information Standards [OASIS], samt World Wide Web Consortium, W3C, med dess undergrupp med inriktningen WS [W3Ca]. Av dessa arbetar W3C med den tekniska grundstandarden, som får samma status som övriga internetstandarder. OASIS inriktar sig mot tillämpningar, bland annat säkerhet i affärstransaktioner. Deras standardförslag är plattforms- och leverantörsoberoende, men har inte samma definitiva status som W3C. Utöver dessa två organisationer finns sammanslutningar med större leverantörsdominans. Deras rekommendationer blir därför mindre definitiva.

3.1 Allmänt

Det finns lite skilda meningar om vad man egentligen lägger i begreppet WS. En definition, direkt citerad från [W3Cb] är

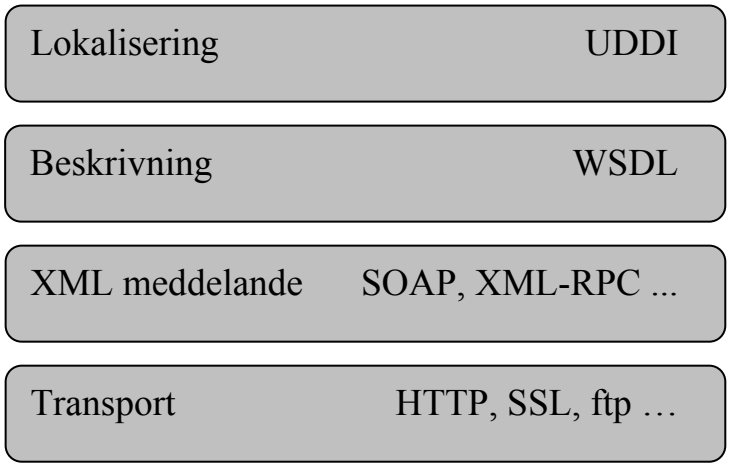
> *Definition: A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.*

En WS är en programvaruagent som skall kunna anropas av en annan WS (också i flera steg), eller annan programvara. Anropet skall ske via ett meddelande som skall vara uppbyggt med hjälp av XML (se SOAP nedan). Meddelanden skall överföras via något standardiserat internetprotokoll.

Vidare skall varje WS ha en, globalt sett, unik identitet, URI, Uniform Resource Identifier. En WS skall kunna katalogiseras och lokaliseras (discovery) via en katalogtjänst (UDDI, Universal Description, Discovery and Integration) som själv är specificerad med hjälp av XML. Metoderna att anropa en WS, samt typerna av data som ingår i anrops- respektive svarsmeddelanden, skall beskrivas med hjälp av XML (se WSDL).

Det som konstituerar WS är alltså

- En enhetlig standard, XML, för att bygga upp data och meddelanden. XML används också för att beskriva metadata, det vill säga beskrivningar av datatyper, objekttyper och -metoder, bindningar et cetera.
- Fyra stycken protokollsnivåer över TCP/IP-nivån enligt figur 3.1.



Figur 3.1: *Protokollsnivåer inom Web Services*.

Enligt definitionen är en WS tänkt att anropas från olika typer av agenter och andra program. De i följande avsnitt beskrivna standarderna har också detta mål. Det bör dock understrykas att man i stort sett alltid har börjat med att utveckla modeller för den vanligaste typen av anrop, nämligen en person som via en webbläsare anropar en WS via http-protokollet. Utvecklingen har inte gått lika långt för andra modeller.

3.2 XML

XML står för Extensible Markup Language. Redan namnet pekar på en viktig egenskap, XML skall vara utbyggbart. Detta sker genom att definitioner och beskrivningar görs i en hierarki, helt i analogi med abstraktionerna förälder-barn vid objektorienterad programmering. En utbyggnad görs genom att införa nya definitioner, eller omdefiniera gamla definitioner, på en barnnivå där man också refererar till definitioner på föräldranivå, som i sin tur kan referera till farföräldern et cetera. De grundläggande definitionerna finns på XML-rotnivå. Vi återkommer till detta senare, vid beskrivning av namnrymder och scheman.

Markup, M i XML, har en beskrivningar i GuruNets [GUR] lexikon *the collection of detailed stylistic instructions written on a manuscript that is to be typeset*. Bakgrunden är alltså att införa märken (”Tags”) i dokument för

att styra hur den uppmärkta dokumentdelen skall presenteras. Detta är precis vad standarden HTML beskriver, för presentation i webbläsare. En annan presentationsstandard, inriktad mot bokförlagsverksamhet, är SGML. Denna är mycket mer omfattande och strukturerad än HTML. Den utvecklades redan i början av 80-talet. Ofta beskrivs XML som ”HTML förbättrad i riktning mot SGML”. Det är riktigt såtillvida att det mesta som skrivits om XML har varit inriktat mot presentation, särskilt i webbläsare. Men XML är mycket mer än så, det är en generell standard för strukturering av data. W3C, som leder utvecklingen av XML, har valt att marknadsföra de 10 viktigaste egenskaperna enligt nedan [W3Cc].

1. XML is for structuring data
   Jämför ovanstående två stycken.
2. XML looks a bit like HTML
   Jämför ovan.
3. XML is text, but isn't meant to be read
   Allt lagras i textformat, inga binärformat används. Det blir därför läsbart för en människa, vilket till exempel underlättar felsökning. Men det primära är att ett enkelt textformat tjänar som minsta gemensamma nämnare mellan olika plattformar, språk, applikationer et cetera.
4. XML is verbose by design
   Jämför punkt 3. XML är ytterst “pratigt”, det kan vara svårt att se skogen för alla träd. En konsekvens är att XML-filer kan bli stora. Detta är en nackdel, speciellt vid datakommunikation. Textformat kan dock lätt komprimeras med effektiva algoritmer.
5. XML is a family of technologies
   Jämför ovan, XML-familjen följer en objektorienterad hierarki.
6. XML is new, but not that new
   Syftar på att XML bygger på mycket erfarenhet från SGML.
7. XML leads HTML to XHTML
   Jämför ovan, XHTML är “en förbättrad HTML”.
8. XML is modular
   Jämför ovan om utbyggbarhet och nedan om namnrymder och scheman.
9. XML is the basis for RDF and the Semantic Web
   RDF, Resource Description Framework, är ett XML-baserat format för metainformation, det vill säga beskrivning av information. Visionen, Semantic Web, är att informationen på webben skall bli sökbar via informationsinnehåll i tillägg till nuvarande sökning via textmatchning.
10. XML is license-free, platform-independent and well-supported
    Oerhört viktigt för generell utveckling av nättjänster, WS.

Efter dessa allmänna ord om XML kan det vara hög tid att beskriva XML. Men först några avgränsningar.

Vårt fokus är WS, vilket innebär kommunikation mellan en beställare av en tjänst samt utförare av tjänsten. Kommunikationen sker via ett meddelande uppbyggt med hjälp av XML. Vidare innebär WS att skapande respektive tolkning av XML-meddelanden sker automatiserat, eventuellt av människa med hjälp av programverktyg. För att understryka detta har vi valt att använda neutrala begrepp, till exempel kedjan

Producent → generator → sändare ➔ mottagare → tolk → konsument

där → betecknar övergång mellan nivåer i OSI-stacken medan ➔ betecknar transport över nätet.

Det finns en oändlig mängd böcker, artiklar och referenser på Internet om XML, framför allt inriktade mot webbpresentation. För vår inriktning, data- och informationsstrukturering i XML-meddelanden som genereras respektive tolkas av programvara, rekommenderas [IBMa]

Ett problem vid beskrivning av XML är ”pratigheten”, punkt 4. Det skönlitterära värdet av en rapport fylld med exempel på fullständiga XML-meddelanden är högst begränsat. Med risk för missförstånd och ofullständighet har vi därför medvetet valt bara skissartade exempel.

Det grundläggande elementet i XML kallas just `element`. Element kan ha ett eller flera underelement, som i sin tur kan ha underelement. Hela XML-meddelandet måste bestå av ett enda rotelement, med barnelement, barnbarnselement et cetera. i en strikt hierarki. Ett barn får bara ha en förälder, det vill säga element får inte överlappa varandra.

Element omgärdas av hakarna <> så att elementet inleds med `<märke ….>` och det avslutas med `</märke>`. Märke är en textsträng, en `tag`. Inuti start-`tag` kan elementet tilldelas `attribute` (punkteringen …. ovan). Attributen är par namn-värde, där namn är en textsträng och värde är en textsträng inom dubbelapostrofer ” ” eller enkelapostrofer ’ ’. Längst ner i hierarkin, i trädets löv, finns elementinnehåll i form av `text`. Ett element utan innehållsdel, det vill säga bestående bara av attribut, kan förkortat skrivas `<märke ….. />`. Ett exempel på ett XML-meddelande (från [IBMa]) är

```
<memories>
     <memory tapeid="1">
          <media mediaid="1" status="vhs" />
          <subdate>2001-05-23</subdate>
          <donor>John Baker</donor>
```

```
          <subject>Fishing off Pier 60</subject>
          <location>
               <description>Outside in the woods</description>
          </location>
     </memory>
     <memory tapeid="2">
          <media mediaid="2" status="vhs"/>
          <subdate>2001-05-18</subdate>
          <donor>Elizabeth Davison</donor>
          <subject>Beach volleyball</subject>
          <location>
               <place>Clearwater beach</place>
          </location>
     </memory>
</memories>
```

Det första en tolk bör göra är att kontrollera att meddelandet följer nämnda regler, till exempel att element inte överlappar varandra. Ett regelriktigt meddelande benämns `well-formed`. En mer omfattande kontroll är att tolken kontrollerar om meddelandet är `valid`. Förutom att vara syntaktiskt riktigt skall då meddelandet följa en struktur som beskrivs i det schema som meddelandet sägs vara en `instance` av. Ett schema är definitioner av alla element som kan ingå i meddelandet, i vilken ordning de måste komma, vilka datatyper som elementinnehållet består av med mera. Ett schema är i sig självt skrivet i XML. Det är därför en instans av ”ett schema för scheman”. Denna hierarki kan bestå av flera nivåer. De grundläggande begreppen finns i en rotnivå, som måste vara inlagd i alla XML-tolkar.

Ovanstående exempel, hämtat från [IBMa], är `well-formed`. Det kan självständigt användas av en XML-tolk åt en lokal applikation ifall man i denna har kontroll över olika elements betydelse och uppbyggnad. I WS-applikationer måste meddelandet emellertid oftast kopplas ihop med maskintolkbart schema, meddelandet skall vara `valid`. Ett förenklat sådant kan se ut som

```
<?xml version="1.0"?>
<memories xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance'     xsi:noNamespaceSchemaLocation='memory.xsd'>
     <memory tapeid="1">
          ........
     </memory>
</memories>
```

Den första raden tillhör egentligen inte meddelandet utan är ett direktiv till tolken. Sådana skall inledas med två tecken `<?` och avslutas med `?>`. I detta fall är innebörden att tolken måsta kunna tolka XML version 1.0. I så fall förstår den också de attribut som lagts till i elementet `memories`.

Det första attributet `xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'` rör begreppet namnrymder (namespaces). Namnrymder är ett sätt att säkerställa att märken, element och attribut får, globalt sett, unika identiteter. Detta är viktigt för WS eftersom samverkande delar skrivs oberoende av varandra av olika organisationer. Namnrymd anges via en sträng följd av : kolon. Början `xmlns:` är den i roten inbyggda rotnamnrymden. Nästa namnrymd i exemplet är `xsi:` som kopplas ihop med strängen `http://www.w3.org/2001/XMLSchema-instance`. Man kan bildligt tänka sig att när det i meddelandet står ett element eller attribut `xsi:something` skall det kopplas ihop (kvalificeras) till den globalt unika strängen `http://www.w3.org/2001/XMLSchema-instance/something`. Det vanligaste är att en namnrymd implicit (det vill säga om man inte kvalificerar ett element/attribut med annan explicit angiven namnrymd) gäller inom det element, och dess underelement, där den anges. Men schemaförfattaren kan, för att till exempel få flexibilitet vid kombination av flera namnrymder, ändra på detta.

Det finns några olika sätt att försäkra sig om att namnrymder blir globalt unika. Ett ofta använt sätt är att, som i detta exempel, utnyttja webbadresser, URL. Dessa måste ju vara globalt unika. Men det innebär alltså inte alls till exempel att tolken skall hämta ett schema på webbadressen. Om inte tolken redan känner till schemat kan man via attribut (det finns några alternativ) i meddelandet tala om var det finns, till exempel `xsi:noNamespaceSchemaLocation='memory.xsd'`. Tillsammans med de komplicerade reglerna för hur man importerar namnrymder mellan olika scheman blir effekten mycket förvirrande. Men en XML-tolk kan lätt bygga upp en entydig hierarki av element.

Schemat lagras i en fil, av konvention med ändelse `.xsd`, medan meddelandet, instansen, kan lagras med ändelsen `.xml`. För en ordentlig genomgång av scheman hänvisas till [KAL02a]. Vi väljer här att bara, nästan utan kommentarer, visa exempelmeddelandet `memories.xml` tillsammans med schemat `memories.xsd`.

```
<?xml version="1.0"?>
<memories xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance'     xsi:noNamespaceSchemaLocation='memory.xsd'>
     <memory tapeid="1">
          <media mediaid="1" status="vhs" />
          <subdate>2001-05-23</subdate>
          <donor>John Baker</donor>
          <subject>Fishing off Pier 60</subject>
          <location>
               <description>Outside in the woods</description>
          </location>
     </memory>
```

```
     <memory tapeid="2">
          <media mediaid="2" status="vhs"/>
          <subdate>2001-05-18</subdate>
          <donor>Elizabeth Davison</donor>
          <subject>Beach volleyball</subject>
          <location>
               <place>Clearwater beach</place>
          </location>
     </memory>
</memories>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="memories">
     <xsd:complexType>
          <xsd:sequence>
               <xsd:element name="memory" minOccurs = "0"
               maxOccurs="unbounded" type="memoryType"/>
          </xsd:sequence>
     </xsd:complexType>
</xsd:element>

<xsd:complexType name="memoryType">
     <xsd:sequence>
          <xsd:element name="media">
               <xsd:complexType>
                    <xsd:attribute name="mediaid"
                    type="xsd:integer"/>
                    <xsd:attribute name="status"
                    type="mediaType" />
               </xsd:complexType>
          </xsd:element>
          <xsd:element name="subdate" type="xsd:date"/>
          <xsd:element name="donor" type="xsd:string"/>
          <xsd:element name="subject">
               <xsd:complexType mixed="true">
                    <xsd:all>
                         <xsd:element name="i" minOccurs="0"
                         maxOccurs="1" type="xsd:string" />
                         <xsd:element name="b" minOccurs="0"
                         maxOccurs="1"  type="xsd:string" />
                    </xsd:all>
               </xsd:complexType>
          </xsd:element>
          <xsd:element name="location" type="locationType" />
     </xsd:sequence>
     <xsd:attribute name="tapeid" type="idNumber" />
     <xsd:attribute name="status" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="locationType">
     <xsd:choice>
          <xsd:element name="description" type="xsd:anyType"
          />
```

```
          <xsd:element name="place" type="xsd:string" />
     </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="idNumber">
     <xsd:restriction  base="xsd:integer">
          <xsd:minInclusive value="1" />
          <xsd:maxInclusive value="100000" />
     </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="mediaType">
     <xsd:restriction  base="xsd:string">
          <xsd:enumeration value="8mm" />
          <xsd:enumeration value="vhs" />
          <xsd:enumeration value="vhsc" />
          <xsd:enumeration value="digital" />
          <xsd:enumeration value="audio" />
     </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

Schemat anger hur man har byggt upp egna strukturer (`complexType`) ur enkla typer (`simpleType`) som är inbyggda i rotschemat (namnrymd `xsd:`). Det finns ett 40-tal inbyggda `simpleType`, exempel är `integer`, `decimal`, `dateTime`, `string`, `boolean`, et cetera. Rotschemat (`xsd:`) består av 30-tal grundelement (`sequence`, `choice`, `enumeration`, med flera) och ungefär lika många grundattribut (`name`, `minOccurs`, `maxOccurs`, med flera) med vars hjälp man bygger upp sitt schema.

Det är inte meningsfullt att gå in på alla detaljer i exemplet. Men rot-elementet skall ha märket `memories`. Det består av en sekvens av element märkta `memory`, av en nydefinierad typ `complexType` som refereras med namnet `memoryType`. Denna nydefinierade typ består i sin tur av en sekvens med element märkta `media`, `subdate`, `donor`, `subject` och `location`, och så vidare. på nästa nivå i hierarkin.

Det viktigaste i ovanstående beskrivning och exemplifiering av XML är två slutsatser.

a) XML-meddelanden är hierarkiskt uppbyggda. De skall alltid kunna beskrivas som ett hierarkiskt träd utgående från ett enda rotelement.
b) XML kan delas upp i två delar. En beskrivningsdel `schema.xsd` och en meddelandedel, `instansen.xml`.

Båda dessa egenskaper återfinns i moderna objektorienterade program-meringsspråk. Det är därför väl lämpat att skriva XML-tolkar i sådana

språk. Det finns bibliotek i många språk för att skriva tolkar, i princip enligt två arkitekturer. Dels DOM, Document Object Model, vilket innebär att hela meddelandet läses in och hela trädet byggs upp innan noderna tolkas. Dels SAX, Simple API for XML, som innebär att man tolkar ”i flykten”, efterhand som meddelandet läses. Den senare arkitekturen är naturlig i tillämpningar av typ XML-brandväggar. Man vill då snabbt hitta vissa attribut eller element för att avgöra om meddelandet bör skickas vidare till mottagande WS eller ej. Jämför också följande beskrivning av headers i SOAP (som numera inte har någon uttydning).

3.3 UDDI och WSDL

I Figur 3.1 anges de fyra nivåer som konstituerar en WS. Den översta nivån är katalogtjänsten UDDI, Universal Description, Discovery and Integration [UDDI]. Denna är i sig själv en WS, det vill säga både innehållet i UDDI och kommunikationen till/från UDDI är beskrivet med hjälp av XML. Det är inte en och endast en UDDI, utan det är naturligen så att det finns en UDDI, eller grupp av UDDI, för varje administrativ eller organisatorisk domän. I alla katalogtjänster är det väsentligt att man har full kontroll över identiteten hos den aktör (konsument eller producent) som anropar katalogen för att hämta respektive lagra data där. Man behöver alltså stark autentisering. I vissa lägen kan det bli aktuellt för en aktör att anropa en UDDI i en främmande domän. Detta är ett typexempel på att man skulle kunna använda SAML, Security Assertion Markup Language, se nedan.

UDDI beskrivs ofta som en katalogtjänst, liknande vita och gula sidorna. Ett företag som har utvecklat en WS registrerar sig i en UDDI, där det i standardiserat och maskintolkbart format publiceras data om företaget och dess WS. Sedan är det tänkt att kunder automatiserat skall kunna söka i UDDI för att hitta den eller de WS som löser deras behov. I en eventuell militär tillämpning, till exempel FMLS, känns det kanske inte naturligt att ha kataloger av typ gula sidorna. Men det finns vissa data som känns väsentliga, till exempel pekare till beskrivningar av WS. Dessa borde kunna användas om man flyttar en WS, tar den off-line et cetera. Nedan ges bara en summarisk beskrivning över vilka huvudelement som finns i UDDI.

`businessEntity` – data om företaget
`businessService` – data om WS. Innehåller bland annat elementet
`bindingTemplate` – som innehåller accessPoint, pekare (URL) till WS samt vilken bindning (till exempel http) som skall ges till SOAP (se nedan).
`tModel` – bland annat pekare till WSDL-beskrivning av WS

Alla dessa element kan signeras med digital signatur för att binda utgivaren och för att garantera äkthet.

Nästa nivå inom WS är alltså beskrivningen WSDL, Web Service Definition Language. Detta är en XML-grammatik för att beskriva en WS. Det bör påpekas att WSDL ännu (oktober -03) inte är stadfäst som W3C-standard, utan fortfarande modifieras. Det senaste utkastet till version 1.2 är [W3Cf], varifrån bland annat nedanstående exempel är hämtat.

Rotelementet i WSDL heter `definitions`. De viktigaste elementen och deras väsentligaste innehåll är:

`definitions` – rotelementet. Där anges namnet på WS och vilka namnrymder som används. Innehåller bland annat följande underelement

`types` – beskriver egendefinierade datatyper som används vid kommunikation med denna WS. Jämför `complexType` i beskrivningen av XML-scheman ovan.

`message` – beskriver vardera ett envägsmeddelande till eller ifrån WS. De olika delarna i meddelandet beskrivs i underelementet `part`.

`interface` – benämndes `portType` i version 1.1. En WS kan ha flera interface, med var sitt attribut `name`, som beskriver var sitt sätt att anropa WS, och vilka meddelanden som används i anropet. Det vanligaste anropssättet är request/response med ett anropsmeddelande till WS följt av ett svarsmeddelande. Men det finns också tre andra sätt – one-way (ett meddelande till WS), solicit-response (WS skickar ett meddelande som följs av ett svar), samt notification (ett meddelande från WS). Elementet operation anger anropssättet, till exempel RPC.

`binding` – beskriver bindningar för kommunikationsprotokoll (vanligtvis SOAP).

`service` – adress till WS, till exempel adressen `endpoint` (som hette `port` i version 1.1).

Ett ofullständigt exempel som är hämtat från [W3Cf]:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TicketAgent"
     targetNamespace="http://airline.wsdl/ticketagent/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://airline.wsdl/ticketagent/"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:xsd1="http://airline/">
     <import location="TicketAgent.xsd"
     namespace="http://airline/"/>
     <message name="listFlightsRequest">
```

```
            <part name="depart" type="xs:dateTime"/>
            <part name="origin" type="xs:string"/>
            <part name="destination" type="xs:string"/>
        </message>
        <message name="listFlightsResponse">
            <part name="result" type="xsd1:ArrayOfString"/>
        </message>
        <message name="reserveFlightRequest">
            <part name="depart" type="xs:dateTime"/>
            <part name="origin" type="xs:string"/>
            <part name="destination" type="xs:string"/>
            <part name="flight" type="xs:string"/>
        </message>
        <message name="reserveFlightResponse">
            <part name="result" type="xs:string"/>
        </message>
        <interface name="TicketAgent">
            <operation name="listFlights"
            parameterOrder="depart origin destination">
                <input message="tns:listFlightsRequest"
                name="listFlightsRequest"/>
                <output message="tns:listFlightsResponse"
                name="listFlightsResponse"/>
            </operation>
            <operation name="reserveFlight"
            parameterOrder="depart origin destination flight">
                <input message="tns:reserveFlightRequest"
                name="reserveFlightRequest"/>
                <output message="tns:reserveFlightResponse"
                name="reserveFlightResponse"/>
            </operation>
        </interface>
</definitions>
```

3.4 SOAP

Låt oss återvända till kedjan

Producent → generator → sändare ➔ mottagare → tolk → konsument.

Enligt Figur 3.1 är de lägsta nivåerna, sändare respektive mottagare, något standardiserat Internetprotokoll. Här finns ett antal alternativ – ftp (tänkt för filöverföringar), smtp (e-mail), http (hypertextläsning), https (säker hyper-text, SSL+http), med flera.

De översta nivåerna, producent respektive konsument, är de program som (eventuellt i samarbete med människor och efter anrop av UDDI) skapar respektive använder de data som överföres.

Det vi valt att kalla generator är den nivå som tar hand om producentdata, väljer till vilken WS det skall skickas, formaterar data i ett XML-format som WS förväntar sig (eventuellt efter att ha tolkat WSDL), och paketerar in allt i ett eller flera paket på det sätt som den valda sändarstandarden kräver.

Det vi kallar tolk tar å andra sidan om hand paket från mottagaren och packar upp paket(en) för vidare befordran till konsumenten. Man kan tänka sig två typer av tolkar. Den kan vara, i princip, en enda modul som klarar av att tolka alla typer av XML-meddelanden som skall användas av konsumenten. Den kan också vara modulariserad. Då är tolkningen uppdelad i två steg, enligt principen för Remote Procedure Call, RPC. I det första steget väljs bara vilken modul (objekt i objektorienterad arkitektur) som skall involveras, sedan sker vidare tolkning och bearbetning i en metod i detta objekt. I vilket fall som helst måste generator och tolk följa gemensamma och standardiserade regler för hur XML-meddelande skall paketeras och hur paket(en) skall läggas in i nästa nivås paket, det vill säga enligt vald sändar/mottagar-standard (oftast http eller https). Det är här som SOAP kommer in i bilden. SOAP är en standard för paketering av XML-meddelanden och bindning av paket(en) till kommunikationsprotokoll. (Det finns också andra sådana standarder – XML-RPC, BEEP, ebXML med flera).

Den än så länge mest implementerade versionen är SOAP V1.1, som anges med namnrymd `xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`. I V1.1 står att SOAP skall uttydas Simple Object Access Protocol. Den senaste standarden är SOAP V1.2 [W3Ce] från juni -03, namnrymder `xmlns:soap-env="http://www.w3.org/2003/05/soap-envelope"`, `xmlns:soap-ncoding="http://www.w3.org/2003/05/soap-encoding"`, `xmlns:soap-rpc="http://www.w3.org/2003/05/soap-rpc"`. I V1.2 påtalar man att SOAP inte skall ha någon uttydning, eftersom protokollet handlar om annat än objektåtkomst. Nedanstående exempel är hämtade från [W3Ce].

Ett SOAP-meddelande skall, enligt reglerna för XML, bestå av ett enda element, ett `Envelope`. Inom detta kan det finnas ett element, `Header`, och det måste finnas ett element, `Body`. Inom såväl `Header` som `Body` får det finnas flera underelement. Speciellt kan det finnas ett element, `Fault`, under `Body`, för signalering i fellägen.

Elementet `Body` innehåller de data som skall skickas från producent till konsument i vår tidigare nämnda kedja. Med konsument menas den slutliga WS som data är avsedd för. Det skall nämligen vara möjligt att data slussas via en serie mellanliggande WS, bland annat för brandvägg eller andra kontroller, loggning med mera. Elementet `Header` skall komma först. Däri läggs bland annat parametrar som rör data i `Body`, till exempel digitala signaturer.

Likaså måste parametrar avsedda för mellanliggande WS ligga i `Header`. Attributet `role` är (olyckligt valt namn) till för tolkning i mellanliggande WS. Attributet `mustUnderstand` är ett sätt att explicit tvinga fram felhantering, vilket inte är standard.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">
     <env:Header>
          <p:oneBlock xmlns:p="http://example.com"
               ...
               ...
          </p:oneBlock>
          <q:anotherBlock xmlns:q="http://example.com"
               env:mustUnderstand="true"
               env:role="http://www.w3.org/2003/05/soap-
               envelope/role/next">
               ...
               ...
          </q:anotherBlock>
     </env:Header>
     <env:Body >
     ...
     ...
     </env:Body>
</env:Envelope>
```

`Envelope` skall nu överföras via kommunikationsprotokoll, vanligast http. Man knyter ihop (begreppet binding [W3Ce]) i princip enligt exempel nedan. SOAP-standarden anger två sätt att binda till http, via `POST` som är Request/Response med indata/resultatdata, respektive via `GET` som bara hämtar resultatdata.

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope" >
     <env:Header>
          ...
     </env:Header>
     <env:Body >
          ...
     </env:Body>
</env:Envelope>
```

Ifall det är fråga om ett objektanrop via RPC skall man via scheman och namnrymder ha specificerat vilka element i Envelope som avser metodnamn, in- och utdata et cetera. I SOAP-bindningen till http rekommenderas att redan i http-headern lägga till eventuella referenser till instans av metodanropet. I ovanstående exempel, ett bokningssystem, skulle första raden kunna se ut som

```
POST /Reservations?bokningsnummer=ATQ123B HTTP/1.1
```

Samma exempel bundet till smtp (e-mail) skulle kunna se ut som:

```
From: a.bengtsson@foi.example.com
To: reservations@travelcompany.example.org
Subject: Travel to CPH
Date: Thu, 29 Nov 2003 13:20:00 EST
Message-Id: <EE492E16A090090276D208424960C@foi.example.com>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">
     <env:Header>
          ...
     </env:Header>
     <env:Body >
          ...
     </env:Body>
</env:Envelope>
```

3.5 Grundläggande säkerhet inom WS

De grundläggande säkerhetsfunktionerna inom WS är, liksom vid all informationshantering, den digitala signaturen och kryptering. Digital signatur säkerställer ett meddelandes originalitet och kan verifierbart och obestridligt knyta ihop ett meddelande med dess utfärdare (den vi kallat producent). Det är därför den digitala signaturen som är användbar när det gäller identitetsverifiering, autentisering. Kryptering kan användas för åtkomstkontroll och för att säkerställa sekretess.

De grundläggande standarderna för WS-säkerhet anger inte tillvägagångssättet för signering respektive kryptering. De beskriver alltså inte algoritmer, nyckellängder, et cetera utan de beskriver hur man i XML-element paketerar dels de data som signerats respektive krypterats, dels anger vilka metoder man har använt. Det är ju väsentligt att detta görs på ett entydigt sätt så att konsumenten kan verifiera signaturen respektive dekryptera meddelandet. Det bör understrykas att WS-standarderna ligger på applikationsnivå, mellan producent-konsument.

De kompletteras ofta av standarderna på lägre nivåer, främst TLS/SSL på transportnivå och IPSEC på nätnivå.

3.5.1 Digital signatur

WS-standarden för digital signatur är XML-Signature Syntax and Processing, [W3Cg]. Det intuitivt naturliga sättet att komplettera ett meddelande med en XML-Signature är till exempel:

```
<?xml version='1.0' ?>
<meddelande>
     <dataelement1> ... </dataelement1>
     <dataelement2> ... </dataelement2>
     <ds:Signature
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          ...
     </ds:Signature>
</meddelande>
```

där det inom elementet `ds:Signature` finns underelement som beskriver vilket, eller vilka, dataelement som har signerats och hur det gått till. Namnrymden `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"` är den som gäller den senaste standarden XML-Signature. Skissen ovan är ett av tre specificerade sätt att paketera signaturen. Det kallas enveloped, vilket innebär att signaturen finns inuti meddelandet enligt ovan. Ett annat sätt kallas enveloping, vilket innebär att meddelandet finns inuti signaturen. Bildligt talat skiftar märkena `<meddelande>` och `<ds:Signature>` plats. Det tredje sättet kallas detached. Detta innebär att signaturen är helt frikopplad från dataelementen. Inuti `ds:Signature` finns då referenser i form av URIer till de aktuella datameddelandena, som i princip kan finnas var som helst.

Nedanstående är en skiss över element som kan eller måste finnas inom `ds:Signature`, hämtad från [W3Cg].

```
<Signature ID?>
     <SignedInfo>
          <CanonicalizationMethod/>
          <SignatureMethod/>
          (<Reference URI? >
               (<Transforms>)?
               <DigestMethod>
               <DigestValue>
          </Reference>)+
     </SignedInfo>
     <SignatureValue>
     (<KeyInfo>)?
     (<Object ID?>)*
</Signature>
```

Kursivering har valts eftersom meddelandet inte är well-formed utan bara en principskiss. Tecknen ?, * respektive + anger att elementet skall förekomma 0 eller 1 gång, 0 eller flera gånger respektive 1 eller flera gånger.

De två obligatoriska elementen i signaturen är enligt ovan *SignatureValue* och *SignedInfo*. I *SignatureValue* läggs den digitala signaturen. Eftersom denna är ett binärt tal måste den kodas till en teckenrepresentation. Detta skall ske via base64-kodning, inga alternativ är tillåtna. Inom *SignedInfo* läggs de underelement som signeras. Observera att det inte bara är data-element som signeras, utan också flera andra element.

CanonicalizationMethod är ett viktigt obligatoriskt element som anger vilken metod som används för kanonisering. Ett och samma XML-meddelande kan vara utformat på olika sätt och likväl vara valid. Blanktecken och ny rad kan läggas till valbart, vissa element kan komma i godtycklig ordning, namnrymder kan anges explicit eller implicit et cetera. Vår generator respektive tolk måste behandla de signerade elementen på exakt samma sätt, annars kan inte signaturen verifieras. Detta kallas kanonisering.

Det obligatoriska *SignatureMethod* anger vilken metod för digital signatur som använts. Metoden är valbar, men för att följa standarden är det ett krav att kunna använda DSS.

Reference pekar på de dataelement som signeras. Elementet finns en eller flera gånger, man kan alltså i en och samma signatur signera olika delar av ett meddelande. Man kan till och med peka på helt externa data, *URI* kan i princip peka vart som helst. *Transforms* anger eventuella transformationer av det aktuella dataelementet, till exempel komprimering. *DigestMethod* anger vilken metod för beräkning av hashvärde som använts. Det är ett krav att kunna använda SHA1. Slutligen finns hashvärdet, base64-kodat, i *DigestValue*. Det är ju detta/dessa värden som, tillsammans med ovan nämnda element, slutligen signeras.

I *KeyInfo* läggs information om den nyckel som konsumenten kan verifiera signaturen med. Om inte elementet anges, måste alltså konsumenten på annat sätt känna till verifieringsnyckeln. De underelement, med data-strukturer, som finns definierade i standarden är *DSAKeyValue*, *RSAKey Value*, *X509Data*, *PGPData*, *SPKIData*, *MgmtData* och *RetrievalMethod*. Namnen torde vara självförklarande, bortsett från de två sista. *MgmtData* kan användas till exempel för att bygga upp en gemensam Diffie-Hellmannyckel. Men standarden rekommenderar andra sätt att göra detta. *RetrievalMethod* används om man behöver peka på externa data, till exempel för att konstruera kedjor av certifikat.

Elementet *Object* är valbart. I fallet Enveloping Signature, när dataelementen finns inkapslade inuti *Signature*, läggs dataelementen i *Object*.

3.5.2 Kryptering

Standarden för kryptering är XML Encryption Syntax and Processing, [W3Ch]. Den är i flera avseenden en ren parallell till XML Signature i det avseendet att den beskriver hur man XML-paketerar krypterade data och hur man via XML anger vilka data som krypterats, vilka krypteringsmetoder som använts et cetera. Det finns också naturliga skillnader. Vid kryptering är det aldrig aktuellt att infoga eller referera klartexten, så till exempel detached kryptering är ju inte tillämplig. Å andra sidan är hantering av symmetriska nycklar viktigare än vid signering. Från [W3Ch] har vi hämtat följande skiss på vad som ingår i XML Encryption.

```
<EncryptedData Id? Type? MimeType? Encoding?>
     <EncryptionMethod/>?
     <ds:KeyInfo>
          <EncryptedKey>?
          <AgreementMethod>?
          <ds:KeyName>?
          <ds:RetrievalMethod>?
          <ds:*>?
     </ds:KeyInfo>?
     <CipherData>
          <CipherValue>?
          <CipherReference URI?>?
     </CipherData>
     <EncryptionProperties>?
</EncryptedData>
```

Samma konventioner gäller som i skissen för signatur. Eftersom kryptering inte är lika väsentlig när det gäller identitetsverifiering väljer vi att inte i detalj utveckla skissen. Man kan notera att avseende nyckelbeskrivningar, *ds:KeyInfo*, så har man hämtat mycket direkt från XML Signature (namnrymden *ds:*). Man lagt till *EncryptedKey*, för att till exempel beskriva distribution av symmetrisk nyckel, och *AgreementMethod*, för att till exempel beskriva konstruktion enligt Diffie-Hellman.

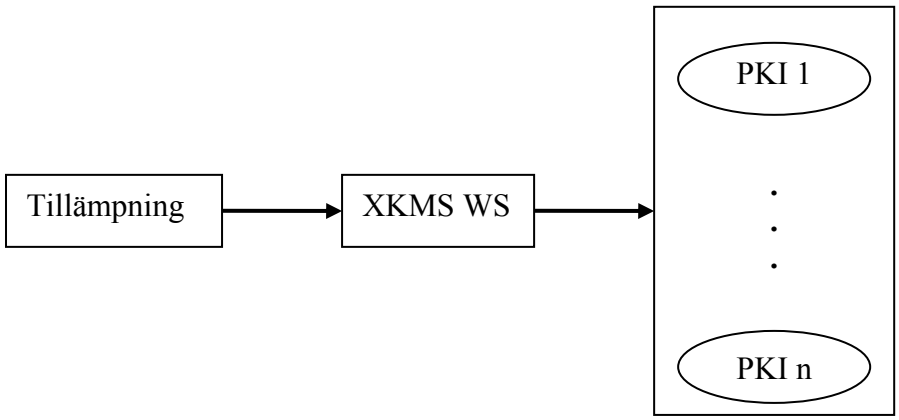
4 XKMS (=XML Key Management Specification)

XKMS utgör ett gränssnitt till en PKI (Public Key Infrastructure) eller en uppsättning av olika PKI-lösningar. PKI:er tenderar att vara synnerligen komplexa. XKMS flyttar komplexiteten från klienten till XKMS-tjänsten. Tanken bakom detta är att underlätta för enskilda klienter och användare, bland annat genom att vidareutveckling av PKI-teknologin inte kräver ständiga uppdateringar av enskilda slutanvändare, och därmed blir man inte hårt bunden till valet av en specifik PKI-lösning.

Detta medför vidare ett behov av centraliserade tjänster och förvaltning av dessa. Man kan redan här observera att detta medför ett beroende av vara uppkopplat mot dessa. Vi återkommer till detta i kapitlets slutdiskussion om XKMS i relation till NBF.

Den vidare beskrivningen av XKMS och dess ingående delprotokoll är till stor grad baserad på material ur [ONE03] och i mindre omfattning material ur [GAL02]. Där värderingar görs, speciellt med avseende på NBF-tillämpning av XKMS, är detta i huvudsak värderingar för vilka denna rapports författare svarar.

Relationen mellan tillämpningsprogram (eller klienter), WS och olika PKI-lösningar kan illustreras med följande figur (vidareutveckling av figur från [ONE03]:



Figur 4.1: *Relationen mellan tillämpningsprogram, WS och PKI-lösningar*.

Förenklad till ett fåtal punktformuleringar är fördelarna med XKMS som en mellannivå mellan aktuell tillämpning och PKI-lösningen enligt [ONE03] att:

- Komplexiteten hos tillämpningen/klienten minskar
- Programmeringen förenklas
- Nya funktioner behöver inte installeras eller uppdateras hos enskilda användare
- PKI-förvaltningen centraliseras

Dessa förenklingar och centraliseringar medför enligt vår bedömning inte att PKI totalt sett blir enklare. Däremot medför centraliseringen att komplexiteten flyttas från de enskilda klienternas många installationer till färre centrala. Dessa centrala installationer torde dock öka i komplexitet.

I tillägg uppstår en betydande trafik av WS-relaterade meddelanden. Denna trafik berör, som framgår av figuren, tillämpning/klient, XKMS-tjänsten och PKI-lösningen. Intentionen, enligt WS-litteraturen i allmänhet och inräknat därmed [ONE03], är att denna trafik skall vara snabb, smidig och underlätta kommunikation och utförande av övriga systemfunktioner. Hur väl detta kan uppfyllas är svårt att bedöma i nuläget, ty utvecklingsarbetet försiggår parallellt i olika WS-projekt och graden av standardisering varierar mellan dessa olika projekt.

4.1 XKMS-protokollet

XKMS-protokollet är ett fråga-/svarsprotokoll som bygger på SOAP. Protokollet består, som beskrivet i [ONO03], av två delar:

- XML Key Information Service Specification (X-KISS)
- XML Key Registration Service Specification (X-KRSS)

I utgångspunkten hanterar XKMS förfrågningar om enskilda nyckelbindningar. Det existerar och vidareutvecklas även funktioner för mängdhantering av kreditivhandlingar. Detta diskuteras kort i [ONE03] och mera detaljerad i [W3Ci][1].

Dessutom är för hela XKMS en ny typ av kreditivhandling, den såkallade nyckelbindningen (key binding association), av vikt.

[1] I äldre WS-litteratur som till exempel [GAL02] – från 2002 – presenteras dessa funktioner som delar av ett separat protokoll, X-BULK. Detta indikerar dynamiken på WS-området.

4.1.1 Nyckelbindningen (Key Binding Association)

En nyckelbindning associerar eller knyter en publik nyckel till specificerade Internettjänster/-adresser och protokoll som skall användas i samband med dessa. XKMS definierar fyra element som används, av XKMS olika protokoll, vid bindning till nyckel eller vid förfrågan om nyckelbindning. Dessa element är:

- Nyckelbindning (`KeyBinding`): Godkänd nyckelbindning från någon betrodd utgivare av nyckelbindningar.
- Ickeverifierad nyckelbindning (`UnverifiedKeyBinding`): Nyckelbindning från ej betrodd källa och som därmed är i behov av verifiering.
- Förfrågan om nyckelbindning (`QueryKeyBinding`)
- Specificering av efterfrågade nyckelbindningsparametrar som skall registreras (`ProtoypeKeyBinding`)

Sättet dessa element används återkommer vi till i diskussionen av protokollen X-KISS och X-KRSS. Ytterligare parametrar, förutom dessa fyra nyckelbindningselement, används vid några av ovan nämnda förfrågningar.

En nyckelbindnings principiella uppbyggnad framgår av följande exempel ur [ONE03]:

```
<KeyBinding>
     <KeyInfo>
     <ds: KeyValue>
          <ds: RSAKeyValue>
               <ds:Modulus>4i0BEhQ8Jc4tjwZYbvtMyYfBrIGOMx34K4C
               do2pAzoGnV679FLmGHWnQy2cSj39hf5D1mIaPyD3j/33Tdf
               glTaaKqp7IPf6ei754fOuI/r1HpX7uqsw+j9LC4Z7GnG3yo
               Y/eBJOZ8TRwMnx+MkwmopXPVLvhMWRyiUOcO3SEkTE=</ds
               :Modulus>
               <ds:Exponent>AQAB</ds:Exponent>
          </ds:RSAKeyValue>
     </ds:KeyValue>
     <UseKeyWith Application=urn:ietf:rfc:2633"
     Identifier="bob@bobcorp.test" />
     </KeyInfo>
</KeyBinding>
```

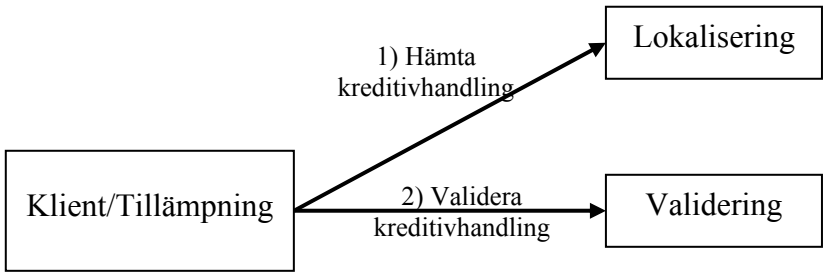
Denna nyckelbindning läsas som att till elementet `KeyInfo` knyts den publika RSA-nyckeln med angivna parametrar (Modulus respektive Exponent). Denna nyckel får användas tillsammans med protokollet S/MIME (vilket beskrivs i RFC 2633) för att skicka e-post till och från adressen bob@bobcorp.test.

Där detta konkreta exempel utgår ifrån en RSA-nyckel, kan detta generellt sett vara ett godtyckligt godkänd slag av kreditivhandling som till exempel ett X.509- eller SPKI-certifikat, en PGP-nyckel eller en ny typ av kreditivhandling. På motsvarande sätt kan andra protokoll och information associeras med nyckeln. Även giltighetstid för nyckelbindningen kan anges.

4.1.2 XML Key Information Service Specification (X-KISS)

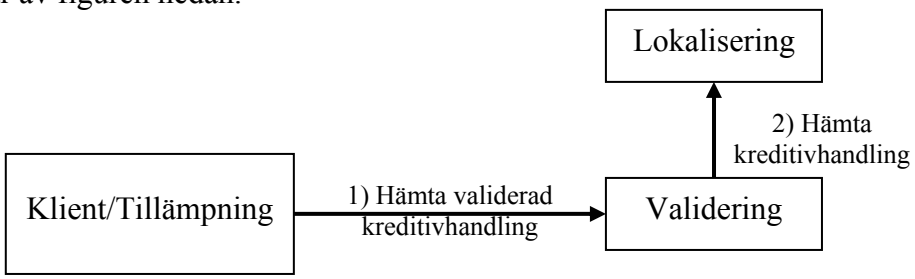
Denna WS stödjer förfrågningar rörande lokalisering och validering av publika nycklar. Förfrågningarna är av typen ”Vilka är kreditivhandlingarna för publika nycklar som jag skall använda för att kommunicera med X med hjälp av protokoll Y?”. En viktig skillnad mellan lokaliseringsvarianten av förfrågningen respektive valideringsvarianten, är att man vid lokalisering kan få åter information som inte är validerad som tillförlitlig. Vid validering returneras endast information som i enlighet med en specificerad säkerhetspolicy bedöms vara tillförlitlig.

Lokaliserings- och valideringstjänster kan användas i kombination med varandra. Inhämtade kreditivhandlingar kan hämtas från osäkra lokaliseringstjänster för att skickas till valideringstjänster för verifiering. I konfigurationen nedan hämtas först kreditivhandlingen från lokaliseringstjänsten, därefter skickas denna till valideringstjänsten för verifiering.



Figur 4.2: *Lokaliserings- och valideringstjänster i kombination*.

Valideringstjänsten kan ta över ansvaret för att hantera lokaliseringsförfrågningen från klienten/tillämpningen. Denna andra tänkbara konfiguration framgår av figuren nedan.



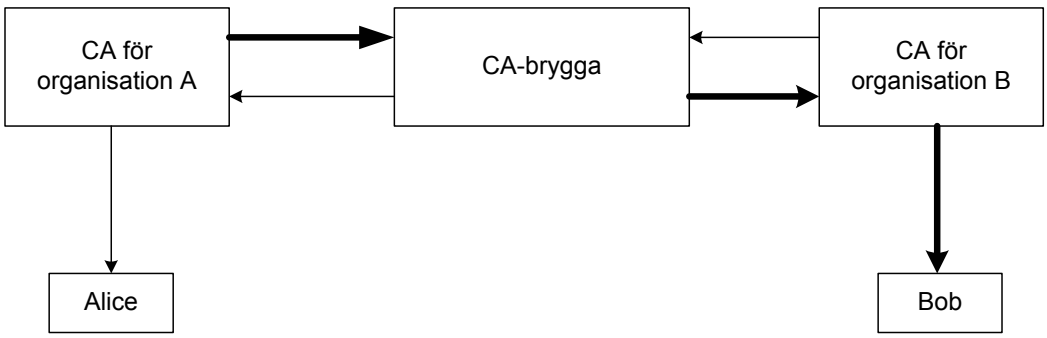
Figur 4.3: *Lokaliserings- och valideringstjänster i kombination (kedjekoppling)*.

Lokalisering

En lokaliseringsförfrågan innehåller en förfrågan om nyckelbindning (`QueryKeyBinding`) och om lokaliseringen lyckas ger den åter en eller flera ickeverifierade nyckelbindningar (`UnverifiedKeyBinding`), ty validering av denna information görs via en separat valideringsförfrågan. Med andra ord bör man vara försiktig med att utan validering förlita sig på en ickeverifierad nyckelbindning.

Bland de lösningar som diskuteras i [ONE03] torde varianten med en certifieringsinstans (Certification authority, CA), vilken agerar som en brygga mellan olika organisationer eller delorganisationer, vara speciellt intressant i försvarssammanhang. Även i andra sammanhang med komplexa organisationer och känslig information som skall hanteras inom dessa, torde lösningen vara intressant. Det huvudsakliga problemet detta kan lösa är att dessa olika organisationer inte behöver besluta att en organisation har bemyndigande över en annan. Varje organisation bibehåller roträttigheter över sin egen PKI och är ur CA-bryggans perspektiv jämställda.

I figuren nedan illustreras förenklat hur två separata organisationer A och B är knutna ihop via en CA-brygga. CA-bryggan är en betrodd jämbördig aktör för organisationerna A respektive B. Alice och Bob avser kommunicera via krypterad e-post. För Alice går certifikatkedjan upp till CA för organisation A, med andra ord ligger roträttigheterna för hennes organisation där. Detta benämns i engelskspråklig litteratur tidvis som ”root of trust”. På motsvarande sätt är Bobs toppnod för tilltro-/certifikatkedjor CA för organisation B. För att kunna skicka krypterad e-post mellan Alice och Bob gäller det därmed att hitta tilltro-/certifikatkedjor mellan organisationerna via CA-bryggan.



Figur 4.4: *Tilltro-/certifikatkedjor mellan organisationer via CA-brygga*.

I fall där flera aktörer än Alice och Bob existerar med mera komplexa organisationstillhörigheter, blir situationen betydligt mera komplex och beräkningsintensiv än vad figuren ovan kan antyda. Till exempel kan man få situ-

ationer där CA-bryggor inom delar av en koalition måsta samsas med bryggor inom andra delar av denna.

XKMS' styrka i detta sammanhang är återigen möjligheterna att dölja komplexiteten från de enskilda tillämpningar och klienter. Därigenom kan lättare olika PKI-lösningar knytas ihop i koalitioner. Koalitionerna kan även ändras över tid. Själva konstruktionen med en CA-brygga är dock inte någon XKMS-konstruktion, däremot har den en potential att underlätta realisering av denna typ av lösningar.

Validering

En valideringsförfrågan innehåller en förfrågan om en specificerad nyckelbindning (`QueryKeyBinding`) och om lokaliseringen lyckas ger den åter en eller flera godkända nyckelbindningar (`KeyBinding`) från någon betrodd utgivare av nyckelbindningar.

Valideringstjänsten är en tjänst till vilken man har byggd upp tilltro, men det bör observeras att detta inte nödvändigtvis medför att tjänsten är tillförlitlig. Man bör med andra ord även här vara medveten och försiktig.

Det existerar behov av autentisering av valideringstjänsten i sig. För närvarande används ett X.509-certifikat för att autentisera valideringstjänsten. Intentionen med XKMS är dock att inte vara bunden av enskilda lösningar. Gissningsvis, enligt [ONE03], torde arbete med åtgärda detta ske inom ramarna för vidare XKMS-specifikation.

4.1.3 XML Key Registration Service (X-KRSS)

Denna WS förvaltar kreditivhandlingar för publika nycklar. Protokollet X-KRSS har via fyra olika tjänster en livscykelshantering av kreditiv för publika nycklar. Dessa fyra tjänster är registrering, återskapning, återutgivning respektive återkallande av kreditiv.

På motsvarande sätt som för X-KISS är förvaltningen av kreditivhandlingar med hjälp av nyckelbindningar (`KeyBindingAssociation`) och deras fyra element (`KeyBinding`, `UnverifiedKeyBinding`, `QueryKeyBinding` samt `ProtoypeKeyBinding`) av vikt.

Registrering är första steget i nyckelbindningens livscykel, vid vilken användaren specificerar tjänster och information denne avser knyta till sin publika nyckel.

Vid en registreringsförfrågan specificeras efterfrågade nyckelbindningsparametrar som skall registreras (`ProtoypeKeyBinding`) tillsammans med

information för att autentisera förfrågan och om nyckelparet är klientgenererad ett äkthetsintyg. Som svar på denna förfrågan ger tjänsten åter en nyckelbindning (`KeyBinding`) och, i fallet av tjänstegenerering av nyckelpar, även en krypterad privat nyckel.

Vid introducering i systemet av aktören Alice, kan detta lösas genom att motta ett engångslösenord direkt från en registreringstjänstansvarig (vid personligt möte utanför systemets ramar eller per e-post om man bedömer att detta säkerhetsmässigt kan fungera tillräckligt väl). Engångslösenordet används endast innan Alice har fått genererad sitt nyckelpar. Nyckelgenerering kan genomföras på klient- eller tjänstesidan. Olika avvägningar och behov kan tala för respektive lösning. Klientbehov av kontroll över nyckelgenereringen kan tala för klientgenerering av nyckel. Om privata nycklar lagras i hårdvara, till exempel i form av smarta kort, kan detta göra klientkontrollerad nyckelgenerering osmidigt och därmed tala för att nyckelgenereringen bör vara tjänstens ansvar.

Återskapning av de privata nycklarna kan behövas av olika skäl. Nyckeln kan ha kommit bort, till exempel genom att ett aktivt kort på vilket nyckeln lagrades har förlorats. Det kan rent av vara så att ägaren har dött eller avsiktligt inte vill ge från sig nyckeln. I allmänhet hanteras återskapning genom att kopior av de privata nycklarna förvaras på säkert sätt hos en särskild aktör (Key escrow agent). Förtroende för denna är naturligt av central vikt i detta sammanhang.

X-KRSS kan med avseende på återskapning endast hantera tjänstegenererade nycklar, inte klientgenererade.

Vid en återskapningsförfrågan specificeras aktuell nyckelbindning (`Key Binding`) tillsammans med information för att autentisera förfrågan, och får åter en krypterad privat nyckel, om återskapning tillåts, tillsammans med uppdaterad nyckelbindning.

Återutgivningstjänsten baserar sig på vad underliggande PKI-lösningar till XKMS erbjuder eller kräver. Livslängden, eller giltigheten, för nyckelbindningen är ej fördefinierad av XKMS. Dock kan kreditivhandlingarna från de underliggande PKI-lösningarna ha definierat detta.

Vid en återutgivningsförfrågan specificeras aktuell nyckelbindning (`Key Binding`) tillsammans med information för att autentisera förfrågan och ett äkthetsintyg för nyckelparet. Som svar ges en uppdaterad nyckelbindning.

Revokering, eller återtagning av nyckelbindningar, kan bli aktuellt av olika skäl som till exempel att en privat nyckel har röjts, att en aktör har fått till-

gång till en kreditivhandling på falska premisser, bytt jobb eller fått ändrade rättigheter inom organisationen.

Vid en revokeringsförfrågan specificeras aktuell nyckelbindning (`Key Binding`) tillsammans med information för att autentisera förfrågan och en såkallad revokeringskod. Som svar ges en uppdaterad nyckelbindning.

Vid revokering är det av stor vikt att säkerställa att den aktör som efterfrågar revokering för specificerad nyckelbindning är auktoriserad för att göra detta. I fallet att aktören är den till vilken nyckelbindningen är gjord, används en särskild revokeringsidentifierare vid registrering av nyckelbindningen. Vid förfrågan om revokering autentiseras revokeringen med hjälp av tillhörande revokeringskod och en envägsfunktion (hashfunktion), som möjliggör beräkning av revokeringsidentifieraren med hjälp av revokeringskoden.

4.2 XKMS i relation till NBF

Om man sätter in WS i ett NBF-sammanhang framträder ytterligare centrala frågeställningar, men även möjligheter.

I koalitioner torde man snabbt hamna i lägen där starkt olika systemlösningar behöver kommunicera och utbyta information. WS-lösningar bör ha betydande fördelar för att realisera kommunikation och informationsutbyte mellan olika systemlösningar. Samtidigt måste man beakta den redan omnämnda trafiken av WS-meddelanden. I exempelvis situationer där autonoma aktörer uppstår i ett NBF-scenario kan mycket tydliga kapacitetsbegränsningar i datatrafiken uppstå. Detta bedömer vi i sådana fall kan göra WS-lösningar tröga och otympliga.

Ovan diskuterade koalitionsrelevanta scenario med CA-bryggor är naturligt nog av särskilt intresse i NBF-sammanhang, genom att uppbyggande av system-av-system underlättas.

Enligt [HAL02] är ett huvudmål med WS att reducera eller rent av att eliminera gränssnittskostnader mellan olika informationssystem. Med gränssnitt avses här inte endast gränssnitt inbyggda i systemen, även manuella gränssnitt ingår. Betydande kostnader är associerade med manuell inmatning av resultat från ett system till ett annat. Övergång från gamla till nya system genererar också betydande kostnader. I NBF-perspektiv är detta viktiga argument, ty ett effektivt NBF kräver snabbhet och säkerhet. Onödiga gränssnittskostnader och tidsförluster måste undvikas.

Detta generella huvudmål tillsammans med den förenkling XKMS avser ge rörande gränssnitten mot olika PKI-lösningar, talar för WS-lösningar med XKMS som en komponent i NBF. Vi ser behov att närmare bedöma i vilken mån denna vision är realistisk med avseende på säkerhetskrav och var insatser bör göras för att bättre närma sig visionen. Inom ramarna av denna studie har dock detta inte låtit sig göras.

Som omnämnt i kapitlets inledning medför XKMS ett beroende av att vara uppkopplat mot centraliserade tjänster och förvaltning av dessa. I litteraturen och debatt kring WS är e-handel det typiskt återkommande exemplet på tillämpning av WS. NBF, och även tänkbara civila tillämpningar inom krishantering, medför situationer där ett sådant beroende åtminstone i vissa lägen har klara begränsningar. [BEN02] diskuterar problem och lösningar rörande autonomitet (avknoppning av enheter) där teknik baserad på digitala certifikat har vissa fördelar i sammanhangen. En snabb bedömning indikerar att WS-konceptet kan försvaga dessa fördelar, men närmare studier krävs för att värdera hur omfattande dessa nackdelar är och hur det kan uppvägas eller hanteras.

Centraliseringen som XKMS implicerar även ytterligare avväganden som behöver studeras närmare. Systemtilltron till de centrala lösningarna är av stor vikt, vilket för militära och andra säkerhetskritiska tillämpningar accentueras ytterligare av deras särskilda behov. Det finns här en balansgång mellan

- lokal kontroll över säkerhetsfunktioner och placering av dessa centralt hos särskild säkerhetskompetens och
- den flexibilitet XKMS erbjuder och höga krav till säkerhet.

[ONE03] diskuterar detta kort med utgångspunkt i principen om säkerhet från slutanvändare till slutanvändare (end-to-end principle). Ur [ONE03]:s perspektiv är fördelarna med centralisering hos särskild säkerhetskompetens tydlig, men beaktar då exempelvis inte explicit militära behov och krav.

5 Security Assertion Markup Language

Security Assertion Markup Language (SAML) är ett XML-baserat ramverk för att transportera säkerhetsinformation. Det som skiljer SAML från tidigare ansträngningar, till exempel Kerberos och digitala certifikat, är att SAML kan användas över systemgränser, beroende på att SAML är oberoende av underliggande autentiseringsmekanism.

I det här kapitlet kommer huvudsakligen SAML att beskrivas främst med avseende på autentisering. Inledningsvis presenteras SAML:s uppbyggnad för att ge en bakgrund till efterföljande delar. Autentisering över systemgränser är syftet med den här rapporten vilket innebär att det här kapitlet i stort begränsas till att beskriva SAML:s stöd för autentisering.

5.1 SAML översikt

Arbetet och standardiseringen av SAML drivs av OASIS vilken är en sammanslutning av flera företag, till exempel Sun Microsystems, Netegrity, RSA Security, IBM och Cisco Systems, bara för att nämna några.

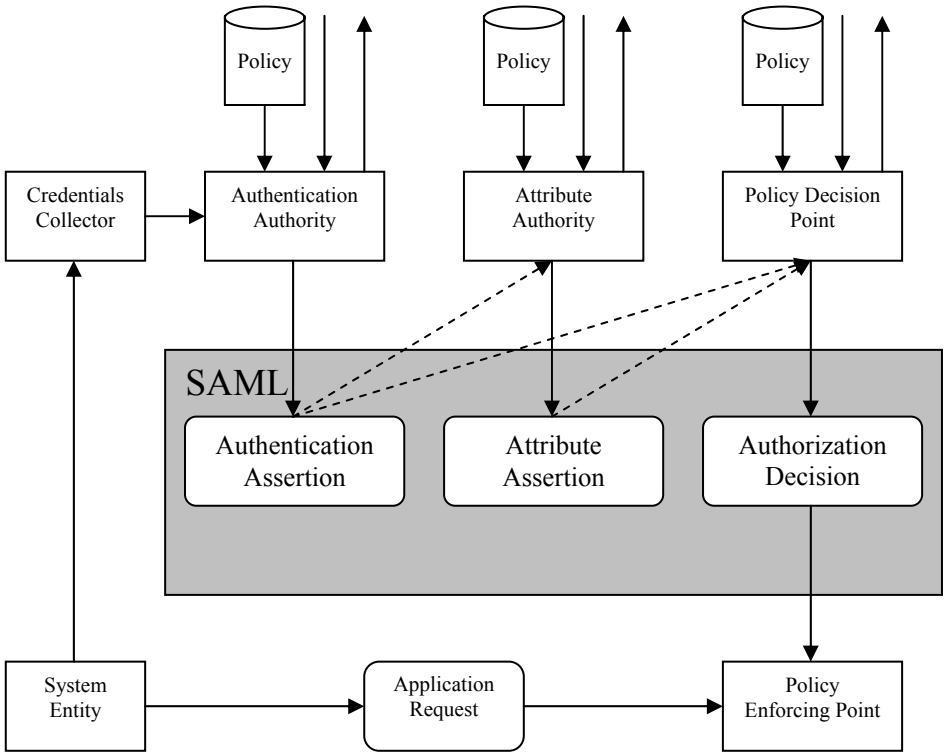
SAML beskrivs i en serie av specifikationer. I denna rapport beskrivs i huvudsak version 1.1 vilken blev en officiell standard i augusti 2003. I korthet består SAML följande specifikationer:

- Assertions and Protocol: Specificerar syntaxen och semantiken för XML-kodade SAML meddelanden [MAL03a].
- Bindings and Profiles: Specificerar bindningar till protokoll samt profiler [MAL03b].
- Conformance Program Specification: Definierar hur implementationer skall överensstämma med SAML protokollet [MAL03c].
- Glossary: Definierar ord och uttryck som används i SAML protokollen [MAL03d].
- Security and Privacy Considerations: Beskriver och analyserar säkerhetsegenskaper i SAML protokollet [MAL03e].

Protokollstacken har utvecklats från ett antal användarfall (use cases) med tillhörande beskrivningar (scenarios). SAML definierar fyra användarfall; Single Sign-On, Authorization Service, Back Office Transaction och User Session [PAL01]. Av dessa är det Single Sign-On som kommer att behandlas i detta kapitel.

5.1.1 SAML modell

En modell av SAML, i SAML specifikationen kallad för SAML domän-modell (Figur 5.1), består av flera sammanlänkade delar vilka samarbetar olika mycket beroende på vilken förfrågan som skall behandlas.



Figur 5.1: *SAML domänmodell* [MAL03a].

I figuren framgår bland annat att SAML inte själv hanterar autentisering utan endast förmedlar resultat. Vidare fattar inte SAML några beslut avseende accessrättigheter. Det är upp till det mottagande systemet att avgöra om intygen (assertions) rättfärdigar tillgång till önskad tjänst.

Två viktiga enheter i modellen är Policy Decision Point (PDP) och Policy Enforcing Point (PEP). Dessa återfinns i den mottagande domänen, det vill säga den domän som äger den tjänst som efterfrågas. PDP är den enhet som fattar beslut avseende auktorisering för sin egen del (om tjänsten kan göra det) eller för den domän den företräder. En PDP konsumerar förfrågningar om auktoriseringsbeslut och producerar intyg om auktoriseringsbeslut. En PEP skickar förfrågningar till PDP och konsumerar svaren, det vill säga intygen om auktoriseringsbeslut.

5.1.2 Assertion

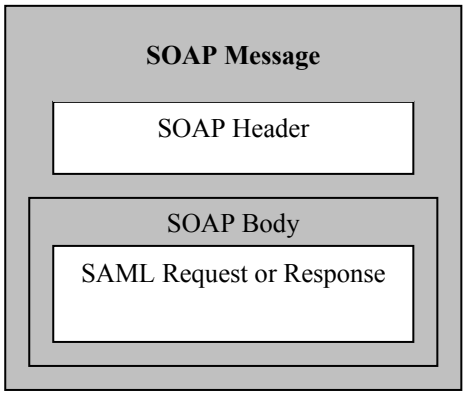
Likt övriga Web Services protokoll bygger SAML på XML-teknik. Det innebär att SAML, som nämndes ovan, inte kan utföra någon egen autentisering utan endast beskriva att och hur autentiseringen har gått till. En vanlig missuppfattning rörande SAML är att det är en ny autentiseringsauktoritet, likt CA-funktionen i ett X.509 system men så är inte fallet [COH03]. SAML är ett protokoll för autentisering med mera, men det ersätter inte existerande autentiseringsmekanismer, utan möjliggör en kommunikation mellan dem. En konsekvens av att använda SAML för att kommunicera säkerhetsinformation mellan två separata säkerhetsdomäner är att man kan bibehålla sina egna säkerhetslösningar, samtidigt som man kan nyttja utomstående resurser. System som använder Kerberos eller PKI bereds här en möjlighet att hantera varandras rutiner utan lokala överenskommelser, något som tidigare varit svårt.

SAML fungerar som en referens mellan två säkerhetsdomäner, det vill säga i och med att SAML själv inte utför autentiseringar förmedlar SAML intyg (assertions) om att autentisering har genomförts samt med vilken metod. Ett intyg är alltså en form av kreditivbrev där det intygas att identiteten av en användare (subject) är verifierad. Detsamma gäller även i de fall då SAML används för att förmedla attribut- och auktoriseringsintyg.

5.1.3 Protokoll och bindningar

Då SAML inte är ett eget transportprotokoll måste det utnyttja ett befintligt sådant. I de standardiserade versionerna av SAML, det vill säga version 1.0 och 1.1 är endast SOAP definierad som bindning till SAML.

Själva SAML-intyget paketeras i SOAP:s meddelandekropp enligt figuren nedan.



Figur 5.2. *SOAP-meddelande med SAMLkropp*.

För att en användare, vare sig det är en människa eller en maskin, skall kunna erhålla ett intyg måste en förfrågan göras samt ett intyg måste genereras. SAML har fem fördefinierade frågeställningar; `Query`, `Subject Query`, `AuthenticationQuery`, `AttributeQuery` och `Authorization Decisionquery`. `AuthenticationQuery` är den frågetyp som används av en PEP hos ägaren till den tjänst som efterfrågas.

Oavsett vilken typ av förfrågan som skickas finns det bara ett svarsformat. Svaret innehåller, om allt stämmer, ett intyg som passar förfrågan samt eventuella felmeddelanden.

Ett tillägg i version 1.1 av SAML är att en utfärdare av ett intyg kan ange `<saml:DoNotCacheCondition>`, vilket innebär att intyget endast får användas vid ett tillfälle och inte sparas för framtida bruk [MIS03]. I version 1.0 fanns endast angivet att HTTP proxies inte fick spara svar innehållande SAML intyg [MAL03b].

5.1.4 SAML intyg

Ett SAML intyg är uppbyggt efter vissa grundkrav på vad ett intyg måste innehålla. Utöver dessa grundkrav finns ett antal tillägg vilka möjliggör ytterligare precisering av vem och vad intyget gäller för. Gemensamt för alla SAML intyg är att de minst skall beskriva följande:

- `xmlns`: XML schema hänvisning.
- `MajorVersion`: Huvudversionen av den använda specifikationen (1 i version 1.1).
- `MinorVersion`: Underversionen av den använda specifikation (1 i version 1.1).
- `AssertionID`: En unik identifierare för det aktuella intyget.
- `Issuer`: Namnet på den SAML auktoritet som utfärdat intyget.
- `IssueInstant`: När intyget utfärdades.

Ett autentiseringsintyg kan se ut enligt följande (delvis ur [GAL02]):

```
<saml: Assertion
     xmlns:”http://www.oasis-
          open.org/committees/security/docs/drafts-sstc-
     schema-  assertion-16.xsd”
     MajorVersion=”1”
     MinorVersion=”1”
     AssertionID=”250.192.2.150.1003876543098”
     Issuer=”Content Portal”
     IssueInstant=”2002-11-03T16:28:22-05:00”>
```

```
     <saml:Conditions
          NotBefore=”2002-11-03T16:28:22-05:00”
          NotOnOrAfter=”2002-11-03T16:38:22-05:00”/>

     <saml:AuthenticationStatement
          AuthentictionMethod=”Password”
          AuthenticationInstant=”2002-11-03T16:28:22-05:00”>

          <saml:Subject>
               <saml:NameIdentifier
                    SecurityDomain=www.contentportal.com
                    Name=”contentportal”/>
          </saml:Subject>

          <saml:AuthenticationLocality
               IPAddress=”250.192.2.150”
               DNSAddress=”secure_server”/>
     </saml:AuthenticationStatement>
</saml: Assertion>
```

I intyget ovan finns villkor (`Conditions`) vilka anger giltighetsperioden för intyget. `AuthenticationStatement` beskriver vilken typ av autentisering som har genomförts (i det här fallet lösenord) samt en kortare beskrivning av användaren så som domänadress och IP adress.

5.2 Use case/Profile SSO

Förmågan att vid ett tillfälle autentisera sig för att sedan, utan att aktivt autentisera sig igen, kunna begära nya resurser kallas Single Sign-On (SSO). I ett lokalt nätverk, med exempelvis Kerberos som autentiseringsmekanism, är detta ett välbekant och lösbart problem. Dock blir detta ett problem när man vill röra sig utanför den egna domänen, när man vill kontakta och utnyttja resurser inom andra säkerhetsdomäner. Det kan mycket väl vara så att man vill utnyttja en resurs i en domän man tidigare aldrig varit i kontakt med. Här uppstår ett problem med flera autentiseringsmekanismer – de har ofta svårt eller omöjligt att samarbeta med andra lösningar. Det finns ett behov av en ”översättare”, en mekanism vilken kan förklara och styrka hur en användare i en domän har blivet autentiserad så att en annan domän kan lita på detta. Single Sign-On var ett viktigt designmål vid skapandet av SAML [MAL03a].

Intressant att notera i de protokoll OASIS producerar, både standardiserade och utkast, är att man hela tiden talar om webbläsare och användare. Det kan nog vara rimligt att anta att människan kommer att vara en stor aktör men tanken med XML och Web Services är att man även skall kunna

genomföra maskin till maskin kommunikation och att de skall kunna förstå varandra.

SAML möjliggör autentisering över systemgränser. Detta sker genom att SAML skickar ett intyg (assertion) till den resurs en användare efterfrågar för att tillstyrka att användaren har autentiserats. Intyget kan förmedlas på två sätt, nämligen:

- SAML artifact, och
- Form POST.

Dessa metoder är HTTP-baserade tekniker vilka stöds av en webbläsare.

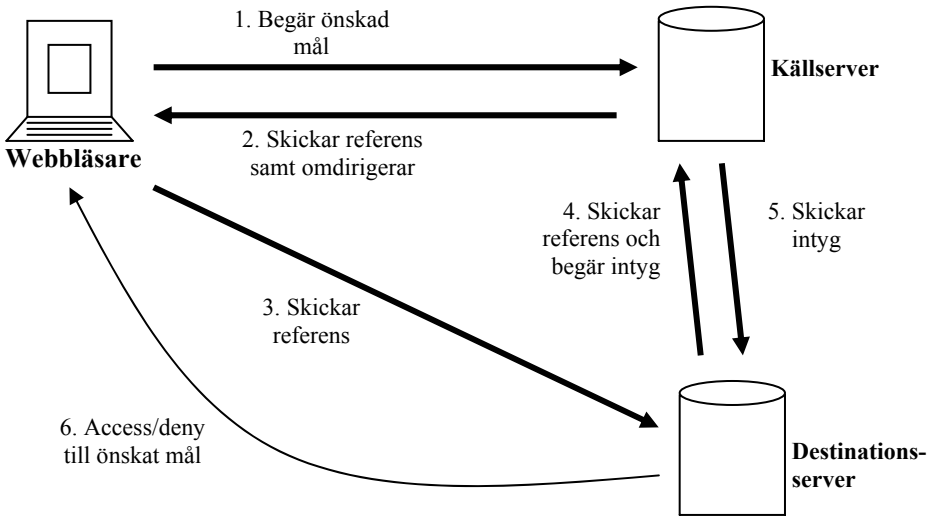
Instanserna i de båda metoderna är desamma, det är tillvägagångssättet som skiljer sig. Principen bygger på att en användare autentiserar sig inom sin egen domän hos sin ”källserver” (source site). Källservern dirigerar om webbläsaren till den server (destination site) som hanterar den önskade resursen. Detta innebär att användaren inte direkt autentiserar sig hos den server som skall tillgodose den önskade resursen. Ur användarens perspektiv märks inte SAML:s inverkan, då det endast är användarens webbläsare som utför arbetet hos användaren. SAML artifact använder sig av en referenssträng vilken bland annat hänvisar till det intyg som gäller just för den aktuella användaren. Form POST är en teknik där intyget skickas med förfrågan om resursen. I följande två kapitel studeras dessa metoder närmare.

5.2.1 Browser/Artifact Profile of SAML (Pull)

I Browser/Artifact profilen läggs stor vikt vid en referens (artifact) vilken skickas mellan de olika enheterna. Själva referensen har en bestämd storlek och skickas med den sträng som innehåller URL-förfrågan till destinationservern.

Förloppet beskrivs i bilden nedan (Figur 5.3). En användare har autentiserat sig lokalt och önskar nu tillgång till en resurs utanför den egna domänen.

1. Användarens webbläsare anropar en transfereringstjänst (hos källservern) inom den egna domänen, det vill säga en tjänst som hanterar förfrågningar om resurser utanför den egna domänen. Förfrågan anger vilken resurs som önskas och var den finns.
2. Transfereringstjänsten tillverkar en referens samt skickar en ny adress till användarens webbläsare vilken dirigerar om läsaren till en mottagartjänst hos destinationsservern.
3. Användarens webbläsare vidarebefordrar referensen till mottagartjänsten.



Figur 5.3: *Browser/Artifact Profilens händelseförlopp*.

4. Mottagartjänsten extraherar den information som behövs för att kunna begära ett intyg om användaren hos användarens källserver.
5. Källservern svarar med att skicka ett intyg. Destinationsservern har här möjlighet att begära mer information avseende användaren om informationen i intyget inte tillgodoser kraven från resursägaren.
6. Destinationsservern meddelar användarens webbläsare om resursförfrågan godkänns eller inte.

Referensen som skickas i steg 2-5 är en referens till det intyg som gäller just för aktuell användare och förfrågan. En referens är definierad som:

```
TypeCode           :=   0x0001
RemainingArtifact  :=   SourceID AssertionHandle
SourceID           :=   20-byte_sequence
AssertionHandle    :=   20-byte_sequence
```

där `SourceID` används av destinationsservern för att avgöra källserverns identitet och var den finns. `AssertionHandle` hjälper källservern att skilja på olika referenser.

Kommunikationen mellan käll- och destinationsservern sker med standardiserade förfrågningar och svar (se kapitel 5.1.3).

I och med att referensen skickas mellan olika enheter, både inom och utanför den egna säkerhetsdomänen, ställer SAML protokollet krav på sekretess och integritetsskydd i steg 1-5. SSL 3.0 eller TLS 1.0 rekommenderas för att

tillgodose dessa krav. I steg 4-5 krävs även ömsesidig autentisering mellan käll- och destinationsserver.

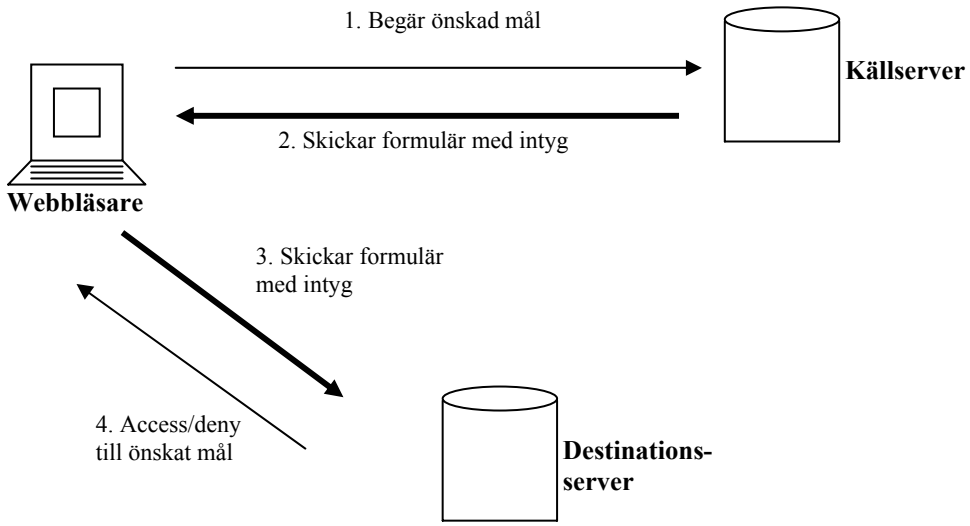
Metoden för Browser/Artifact profilen brukar även benämnas som Pull, det vill säga mottagaren får aktivt deltaga i processen för att erhålla ett intyg (assertion). Källservern antar rollerna Credentials collector, Authentication authority och Attribute Auhtority, medan destinationsservern tar rollerna Policy Decision Point och Policy Enforcing Point.

Denna metod är den vanligast implementerade lösningen i de produkter på marknaden som stödjer SAML [FON02].

5.2.2 Browser/POST Profile of SAML (Push)

Till skillnad från föregående profil använder Browser/POST profilen inte sig av någon referens. Intyget följer med förfrågan om en resurs från användarens webbläsare.

Förloppet beskrivs i bilden nedan (Figur 5.4). En användare har autentiserat sig lokalt och önskar nu en resurs utanför den egna domänen.



Figur 5.4: *Browser/POST Profilens händelseförlopp*.

1. Användarens webbläsare begär tillgång till en resurs via källserverns transföreringstjänst i den egna domänen.
2. Källservern genererar ett formulär med ett SAML svar innehållande ett intyg. SAML svaret är digitalt signerat.

3. Användarens webbläsare skickar SAML svaret till önskad destinationsserver.
4. Destinationsservern svarar genom att godkänna eller förkasta användarens önskan avseende aktuell resurs.

Svaret från källservern i steg 2 är ett givet HTML-dataformulär vilket ”fylls i” av källservern. Metoden POST används för att möjliggöra ”ifyllningen” [W3Cj]. Dataformuläret inkluderas i body-delen av ett SOAP-meddelande. Själva intyget inkluderas i formuläret.

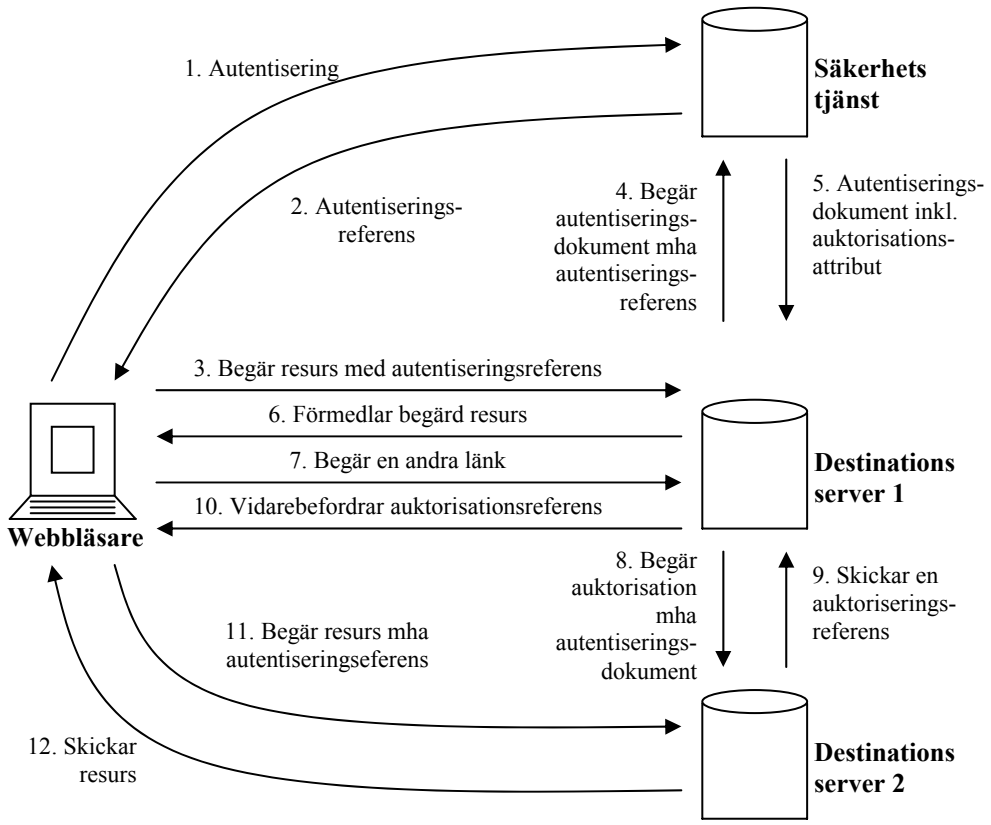
Till skillnad från Browser/artifact profilen skickas här intyget via användarens webbläsare till destinationsservern, vilket även kallas för Push. Således finns det ingen direkt kommunikation mellan utfärdare och konsument av ett intyg. Av denna anledning kräver specifikationen [MAL03b] att SAML svaret från källservern är digitalt signerat. Detta gäller SAML meddelandet som helhet, det krävs dock inte att individuella intyg inom meddelandet är signerade. Vidare rekommenderas SSL 3.0 eller TLS 1.0 för att hantera kravet på sekretess och integritetsskydd i steg 2 och 3.

5.2.3 Tredje part

En användare befinner sig inte alltid inom en säkerhetsdomän, vilken är kontrollerad av användaren själv eller ett företag där användaren är anställd, utan måste utnyttja en självständig tjänst för autentisering. För denna situation finns en tredjepartsprofil (Figur 5.5). Metoden kan även användas om två olika domäner inte har tidigare kunskap om varandra, det vill säga de har inte samarbetat tidigare. Profilen utnyttjar både push och pull förfarandet från tidigare profiler.

De tidigare profilerna i 5.4.3 och 5.4.2 är standardiserade av OASIS i SAML version 1.0 och 1.1, men denna lösning finns endast som ett förslag i [PLA01] vilken gavs ut innan version 1.0.

Den framtida förhoppningen ur SAML:s perspektiv är att två för varandra okända domäner skall kunna utbyta information eller tjänster via en tredjepartstjänst [MAL03f].



Figur 5.5: *SSO tredjepartssäkerhetstjänst scenario händelseförlopp*.

5.3 Säkerhetsdiskussion

SAML bidrar till att beskriva säkerhetsinformation i en Web Services miljö. Det gör SAML till ett säkerhetsprotokoll, men det är viktigt att komma ihåg att SAML i sig själv inte utför säkerhetsåtgärder. Om till exempel den lokala autentiseringen mellan användare och det egna systemet är bristfällig tillför inte SAML en höjning av säkerhetsnivån. I värsta fall hjälper SAML till att sprida osäkerheten till ett annat system.

Tanken är att alla lokala system har sina egna lösningar för att verifiera identiteten på sina användare. SAML intygar att en utomstående användare är verifierad och därmed betrodd i det egna systemet, vilket gör att det mottagande systemet också kan välja att lita på densamme. Fullt utnyttjat kan en användare utnyttja resurser i ett externt system utan att autentisera sig hos detta system. Detta ställer krav på att olika system kan uttrycka sina säkerhetsbehov och förklara dessa för ett annat system. Bara för att en användare är autentiserad i ett system innebär detta inte att samma användare automatiskt kommer att bli godkänd av nästa system.

SAML har flera likheter med digitala certifikat. Ett intyg är utom kontroll för utfärdaren i samma ögonblick som det skickas över ett nät till en mottagare. Det finns ett antal regler för hur intyg skall hanteras men utfärdaren har ingen möjlighet att kontrollera detta. Likt certifikat bör ett SAML intyg ha en så kort giltighetstid som möjligt. I de flesta fall är detta inget problem då ett intyg är en engångsprodukt, vilket måste förnyas om en ny resurs begärs. I de båda standardiserade versionerna av SAML (1.0 och 1.1) sker detta internt mellan autentiseringsservern och den applikation som utfärdar intyg. Tredjepartsprofilen i [PLA01]anger att en resurs kan vidarebefordra förtroende genom att göra nya intyg när en ny resurs begärs, även om denna resurs inte återfinns inom den nuvarande domänen. I detta fall behövs ett intyg med lite längre varaktighet.

I de fall då en HTTP Proxy används tillåter inte specifikationen att SAML intyg sparas i ett cache-minne. Om en efterfrågad tjänst är hemlig, eller i alla fall av känslig karaktär, kan identiets- eller tjänsteförmedlaren ange i SAML intyget att det inte får sparas i något cache-minne ens hos användaren. Funktionen `<DoNotCacheCondition>` beskriver hur cache-hanteringen skall gå till.

En väsentlig fråga att ställa sig är varför man behöver flera profiler för att utföra autentisering? Browser/POST profilen är rättfram och har det minsta antalet kommunikationssteg. Ändå är Browser/artifact profilen den mest implementerade lösningen [FON02]. Svaret ligger troligtvis i behovet av digitala signaturer i Browser/POST profilen. Digitala signaturer kräver hantering av publika nycklar, vilket kan vara komplicerat. Därav blir Browser/artifact profilen enklare att hantera även om den kan tyckas lite mer omständlig. Den blir enklare beroende på att det sker en direkt kommunikation mellan utfärdaren av intyget och mottagaren. Dessa steg (steg 4 och 5 i Figur 5.3) kräver ömsesidig autentisering med exempelvis SSL3.0 eller TLS 1.0.

Intressant att notera i de protokoll OASIS producerat, både standardiserade och utkast, är att man hela tiden talar om webbläsare och mänskliga användare. Det kan nog vara rimligt att anta att människan kommer att vara en stor aktör men tanken med Web Services är även att program och processer skall kunna kommunicera med varandra över systemgränser.

6 Koncept för identitetsverifiering över systemgränser

Det finns flera aktörer på marknaden när det gäller säkerhet inom Web Services. Oftast är det koalitioner av företag som arbetar för att gemensamt hitta en bred standard att arbeta efter. W3C och OASIS har nämnts tidigare i rapporten och senare kommer Liberty Alliance att presenteras. Fördelen med koalitioner är att de lösningar som framkommer accepteras av flera tillverkare, vilket i sin tur medför att de produkter som tas fram smidigare kommer att kunna samarbeta med andra tillverkares produkter.

I det här kapitlet presenteras tre olika lösningar på identitetshantering över systemgränser, nämligen Microsofts .NET Passport, Liberty Alliance och WS-Security. .NET Passport är en egenutvecklad produkt från Microsoft, medan Liberty Alliance och WS-Security är samarbete mellan flera företag. I Liberty Alliance:s fall var det Sun Microsystems som initierade arbetet, men nu är det mer än 150 företag involverade [CRO03].

I grunden syftar alla tre systemen mot samma sak: att hantera webbaserad identifikation och autentisering. Lösningarna möjliggör för olika systemägare att förmedla och dela autentiseringsinformation. De huvudsakliga skillnaderna är att Microsoft .NET Passport är en produkt baserad på centraliserade servrar medan Liberty Alliance är en specifikation baserad på distribuerade servrar [TAS02]. Liberty Alliance och WS-Security är i flera avseenden närbesläktade.

6.1 Microsoft . NET Passport

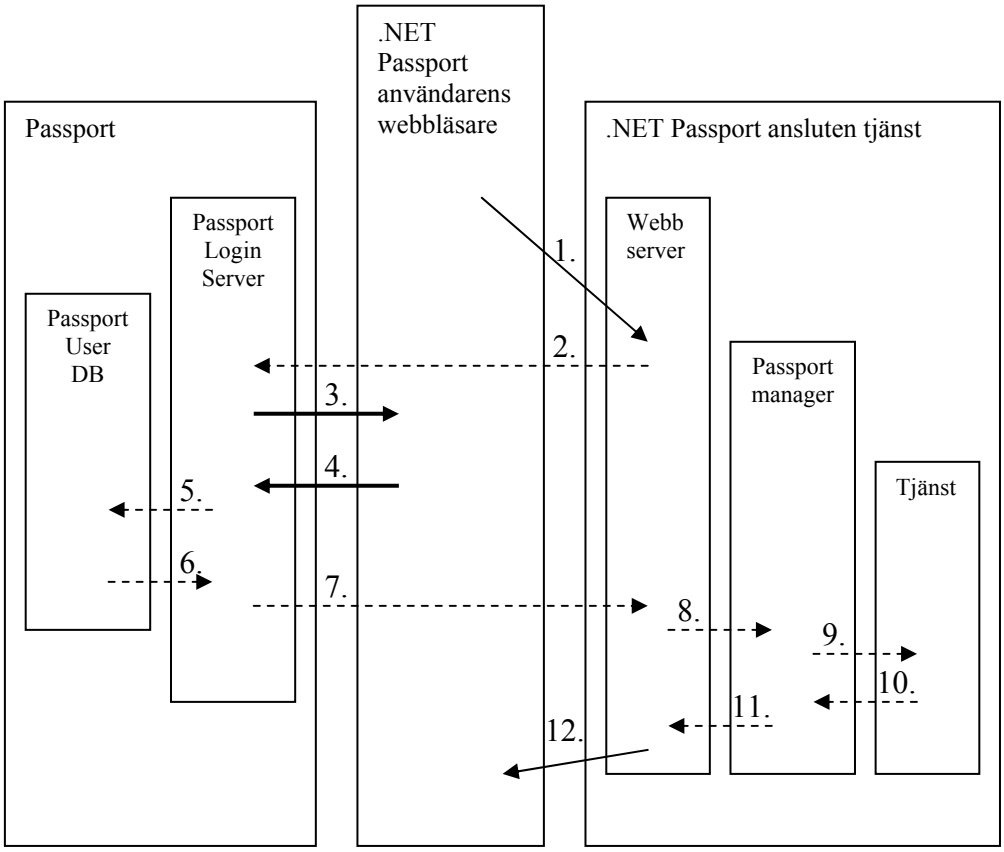
Microsoft .NET Passport [MICRO] är ett centralt reglerat autentiseringssystem helt inriktat på människa-maskin. En användare har en global identitet vilken hanteras av en .NET Passport server. Anslutna tjänster har .NET Passport single sign-in (SSI) implementerat, vilket är ett program för att hantera den information som skickas via användarens webbläsare till .NET Passport servern och tillbaka igen.

.NET Passport erbjuder, utöver autentisering, även förvaring och en förenklad hantering av information avseende en specifik användare i .NET Passport Profile. Det rör sig om vanligt återkommande information såsom namn, adresser, föredragna inställningar et cetera. Om man önskar kan man även bygga på med .NET Passport wallet, en utökad profil där man exempelvis förvarar kortnummer med mera.

Centrala tekniker för .NET Passport är HTTP redirect och cookies. En användare dirigeras om från den tjänst användare vill utnyttja till en .NET Passport server så fort autentisering, profilen eller plånboken (wallet) efterfrågas. Information avseende användaren skickas endast som krypterade cookies mellan .NET Passport och den aktuella tjänsten.

En inloggning med .NET Passport kan se ut enligt följande (Figur 6.1). Det förutsätts här att den efterfrågade tjänsten är ansluten till .NET Passport samt att tjänsten och .NET Passport servern därmed utbytt krypteringsnycklar.

En användare ansluter sig till den server som erbjuder önskad tjänst (1). I det här fallet är användaren inte ännu inloggad någonstans med .NET Passport så användaren utnyttjar länken ”Passport sign in”. Webbservern dirigerar om användaren till .NET Passport Log In server (2). Tillsammans med omdirigeringen skickas även webbtjänstens identifierare samt en returadress med.



Figur 6.1: *Inloggning via .NET Passport*.

.NET Passport Log In servern verifierar att webbtjänsten är en .NET Passport-ansluten tjänst genom att jämföra identifieraren och returadressen med sitt eget register. Användaren presenteras en inloggningssida vilken kräver att användaren identifierar sig. Vanligtvis sker detta genom en mail-adress och ett lösenord (3, 4). Inloggningen sker krypterat via en SSL uppkoppling.

Om autentiseringen är framgångsrik hämtas användarens .NET Passport Unique ID (PUID) tillsammans med den information ur .NET Passport Profile som användaren, i förväg, har angivet som godkänd för spridning vid inloggning. Informationen hämtas ur en databas kontrollerad av .NET Passport (5, 6). .NET Passport tillverkar tre cookies:

- Biljettcookie vilken inkluderar PUID samt en tidsstämpling.
- Profilcookie vilken lagrar information från användarprofilen.
- Cookie för besökta tjänster.

baserade på den upphämtade informationen. Med hjälp av den ursprungliga tjänstens krypteringsnyckel krypteras biljett- och profilcookien och skickas med, via användarens webbläsare, till den angivna returadressen (7). Cookie:n för besökta tjänster lagras i användarens webbläsare.

.NET Passport Manager-funktionen i den ursprungliga tjänsten dekrypterar biljett- och profilcookien vilket medför att användaren nu är autentiserad och den önskade tjänsten kan utföras (8-12).

I det fall när användaren redan har loggat in hos en webbsida med .NET Passport sker steg 2-12 utan användarens aktiva medverkan. .NET Passport server vet att användaren redan har loggat in vid ett annat tillfälle, genom att hämta de biljett- och profilcookie:na som har placerats i användarens webbläsare, och skickar biljett- och profilcookies till den returadress som specificerats vid förfrågan.

Vid utloggning utnyttjas cookie:n för besökta tjänster. .NET Passport servern meddelar alla webtjänster användaren loggat in på att de skall förstöra de biljett- och profilcookies som förmedlats vid inloggningen. .NET servern kan inte göra detta själv då cookie:s endast kan förstöras av användaren och den som har skickat dem.

Stark kritik har riktats mot .NET Passport och dess tidigare versioner. Kritiken har främst handlat om centraliseringen av autentiseringsservrarna, plånboken (wallet) samt cookie betydelsen. Centralisering av autentiseringstjänsten medför minskad kontroll både för användare och tjänsteförmedlare.

Det medför också att systemet skulle drabbas hårdare om en av servrarna (i praktiken finns det mer än en server) skulle utsättas för en lyckad attack.

Plånbokstjänsten bevisades 2001 [SLE01] vara mycket sårbar för en attack. Tjänsten togs bort för att ses över [MCW01] och har sedan dess åter tagits i bruk.

Större delen av alla webbläsare i bruk är Microsoft Internet Explorer. .NET Passport fungerar bäst tillsammans med denna webbläsare. Borttagning av förbrukade cookies fungerar sämre exempelvis med Netscape Navigator [KOR00]. Det kan mycket väl hända att en cookie ligger kvar i användarens webbläsare även efter det att den skall ha förstörts. Främst är detta ett problem om användaren tror sig har loggat ut från en offentlig dator, vilket kan medföra att nästa användare kan komma åt den rättmätiga användarens uppgifter och agera i dennes namn.

Mycket av .NET Passport struktur liknar Kerberos. En väsentlig skillnad är dock förhållandet biljettcookie och Kerberosbiljett. En Kerberosbiljett är en sessionsnyckel knuten till en specifik tjänst. Den kan med andra ord endast användas en gång, även om en angripare skulle lyckas få tag i den. En .NET Passport biljettcookie är giltig så länge användaren är inloggad, vilket innebär att om en angripare kommer över en annan användares biljettcookie kan angriparen fritt utnyttja den på samtliga .NET Passport anslutna system med enkel inloggning.

6.2 Liberty Alliance

Målsättningen med Liberty Alliance (vanligtvis bara kallad Liberty) skiljer sig inte från Microsofts – båda syftar till autentisering över skilda domäner. Liberty är dock en distribuerad lösning där ägandet av autentiseringsservrar inte är givet.

I Libertys specifikation finns det tre olika aktörer; Användare (principal), Identitetsförmedlare (Identification Provider) och Tjänsteförmedlare (Service Provider). Rollerna känns igen från SAML och det beror på att SAML är en viktig komponent. Liberty sätter dock in autentisering i ett sammanhang, till skillnad från den ursprungliga SAML specifikationen. SAML möjliggör autentisering över systemgränser medan Liberty är en komplett autentiseringsspecifikation.

Version 1.0 och 1.1 av specifikationen syftar till att med dagens teknik hantera autentisering via en webbläsare. Tanken med Liberty är att i förlängningen även kunna förmedla autentisering mellan program men inled-

ningsvis har man fokuserat på människa-maskin och då specifikt webbläsare-server.

6.2.1 Identitetsfederation

Single Sign-On i lokala system är ofta skapade baserat på att en användare endast har en identitet. Microsoft har fört denna idé vidare i Passport och skapat en global identitet för varje enskild användare. Liberty har valt att låta användarna behålla sina lokala identiteter, det vill säga en användare kan ha skilda identiteter hos olika tjänster. Orsaken till detta beslut är underhåll av användarinformation. En användare uppger olika mängd information om sig själv beroende på vad tjänsten kräver och erbjuder. Genom att låta användaren själv kontrollera vilken information som förs vidare samt även själv kunna uppdatera information stärks användarens integritet samtidigt som repeterade uppdateringar undviks.

En autentiseringsserver måste dock veta alla identiteter en användare besitter. En tjänsteförmedlare kan också ha intresse av att veta om en användare har öppnat ett konto hos en annan tjänst, om tjänsterna samarbetar. Tjänst X kan exempelvis tillhandahålla en produkt medan tjänst Y har en transporttjänst. Om X och Y samarbetar kan X direkt erbjuda en användare att skicka produkten med Y. Det underlättar givetvis för användaren att automatiskt få produkten levererad utan att själv behöva beställa detta. Hur vet då X att användaren även har ett konto hos Y och hur vet Y vem X avser när de skall leverera produkten? Med en global identitet hade detta inte varit en aktuell frågeställning då användaren alltid är densamme. I ett distribuerat system med flera lokala identiteter måste tjänst X på något sätt kunna förmedla användarens identitet till tjänst Y utan att veta vad användaren har för identitet där.

Lösningen är identitetsfederation. Identitetsservern har en lista på vilka tjänster en användare har registrerat och därmed även vilka identiteter som är aktuella för respektive tjänst. Tjänst X frågar identitetsservern om användaren har någon leverantör registrerad och får en lista med aktuella leverantörer sänd till sig. Tjänst X kan på så sätt föreslå en leverantör av produkten utan att användaren behöver efterfråga detta själv. Vidare kan tjänst X beställa leverans hos tjänst Y i användarens namn genom att be identitetsservern om den identitet användaren använder hos tjänst Y.

Identitetsfederationen hjälper även till vid utloggning. När en användare loggar ut, från en godtycklig tjänsteförmedlare, meddelas identitetsservern och därmed distribueras utloggningen till samtliga tjänsteförmedlare som efterfrågats under sessionen. Liberty kallar detta för global utloggning.

6.2.2 Pseudonymer

En användares identitet skyddas vid transport mellan olika tjänsteproducenter genom pseudonymer. Identitetsservern måste etablera en unik pseudonym för varje kombination av användare och tjänst.

I exemplet ovan är det användarens pseudonym som skickas när X begär användaridentitet för att kunna utnyttja tjänst Y. Pseudonymen är dessutom krypterad med, i det här fallet, Ys publika nyckel, vilket gör att X inte lär känna pseudonymen.

Själva pseudonymen består av en sträng slumpade tecken. Stränginnehållet är meningslöst i sig självt, men fungerar som en referens mellan identitetsservern och tjänsteproducenten.

6.2.3 SAML i Liberty

Mycket av vad Liberty gör känns igen från SAML beskrivningen (kapitel 5). En utvidgning av SAML specifikationen sker genom införandet av autentiseringskontext [MAD03]. Ett normalt SAML intyg kan ange, utöver användaridentitet med mera, vilken metod som används för att autentisera användaren. Liberty utökar intyget genom att även ange begränsningar rörande vilken metod som måste användas och vad som minst måste uppfyllas för att ett intyg skall godkännas.

6.2.4 Liberty i framtiden

Den första fasen i utvecklingen av Liberty Alliance, version 1.0 och 1.1, syftade till Single Sign-On och identitetsfederation. Användaren stod i centrum för utvecklingen.

Fas två riktar mer tydligt in sig på Web Services. Ledordet för utvecklingen är interoperabilitet och arbetet syftar till att beskriva de nödvändiga tekniska komponeterna inom Liberty Identity Web Service Framework (ID-WSF) [TOU03].

Exempel på ny funktionalitet är förmedling av användarattribut baserat på vad användaren tillåter. En utbyggnad av användarinteraktionen möjliggör för en identitetsförmedlare att begära tillstånd av en användare att dela med sig av användarinformation. Syftet med detta är att en tjänsteförmedlare kan på så sätt erbjuda en bättre service genom att ställa specifika frågor avseende användare.

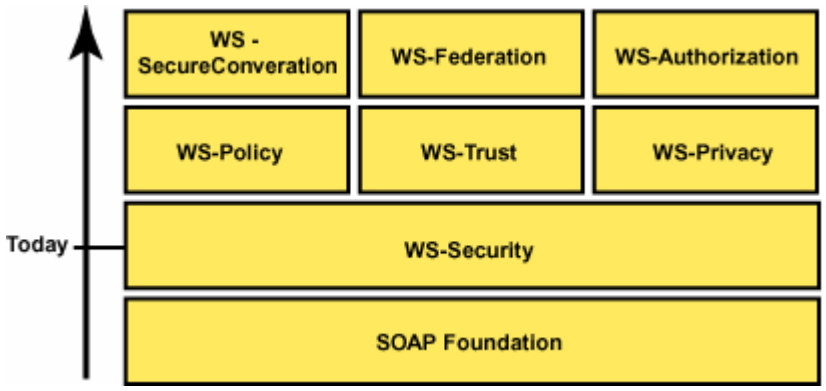
HTTP är idag den dominerade tekniken inom Liberty, och Web Services i allmänhet. Version 2.0 kommer att erbjuda en Liberty-klient som inte kräver

stöd för HTTP. Detta är en stor fördel för resurssvaga komponenter såsom mobiltelefoner [XML].

6.3 WS-Security

IBM, Microsoft, Verisign, BEA och RSA har gemensamt tagit fram en specifikation, WS-Security [KAL02a]. Den handlar om hur man paketerar in krypterade och signerade dataelement, jfr XML Encryption respektive XML Signature ovan, i SOAP-meddelanden. Speciellt finns detaljerade scheman för att paketera Kerberos biljetter, X.509-certifikat och SAML-assertions. Specifikationen har överlämnats till OASIS i syfte att standardiseras till en OASIS-standard.

I tillägg till grundspecifikationen WS-Security har de fem företagen gått vidare med utbyggnader för olika WS-funktioner. Nedanstående figur är hämtad från [IBMb]. Notera att utbyggnaderna är under utveckling och föremål för granskning och revision. Today på tidsaxeln antyder standardiseringsläget april -02.



Figur 6.2: WS-Security med föreslagna utbyggnader.

WS-Policy, hur man beskriver sin WS vad gäller tillåtna attribut, algoritmer, certifikattyper et cetera.

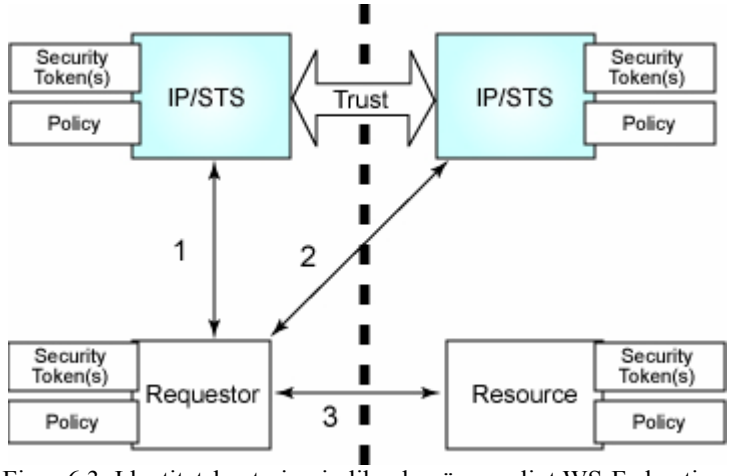
WS-Trust, vilka typer av tilltro man kan bygga upp. Till exempel kan man ange att man inte godkänner direktanrop, utan att alla anrop måste ske via godkända proxy-servrar.

WS-Privacy, hur man beskriver vilka sekretesskrav man har. Dessa är sedan tänkta att ingå i WS-Policy och WS-Trust.

WS-SecureConversation, hur man bygger upp hela säkrade sessioner mellan WS, trots att SOAP i grunden är tillståndslöst requets/response. Kan i viss mån liknas vid ”SSL mellan WS”.

WS-Authorization, hur man beskriver åtkomstkontroll. Detta har stora likheter med XACML, eXtensible Access Control Markup Language.

WS-Federation, [KAL02b]. Här finns några modeller av just det som vår rapport försöker fokusera på, hantering av identiteter över systemgränser. Dels behövs hantering av pseudonymer, eftersom ett visst objekt oftast identifieras på olika sätt inom olika system. Dels behöver man (via WS-Trust, WS-Policy med flera) beskriva hur den som anropar en WS kan erhålla attribut, identiteter med mera som den aktuella WS kan verifiera. Som exempel har figur nedan hämtats från [KAL02b]. En anropare (Requestor) från en domän vill nå en WS (Resource) i en annan domän, och erhåller en pseudonym med attribut genom att identitetstjänster (IP/STS) i de två domänerna har etablerat tilltro till varandra enligt någon viss policy.



Figur 6.3: Identitetshantering i olika domäner enligt WS-Federation.

WS-Federation handlar i mångt och mycket om samma frågeställningar som Liberty Alliance i kap 6.2. Det är därför angeläget att granska skillnader och möjligheterna för ömsesidig anpassning till en enda standard. I [LIBER] presenterar Liberty sin syn på detta. Det är troligt att de sammansmälts till ett koncept, men det är för tidigt att sia om på vilket sätt.

7 Slutord

Huvuddelen av denna rapport har bestått i en beskrivning av grundstenarna inom Web Services. Målet har varit att ge en bild över hur långt utvecklingen har gått inom olika koncept för autentisering över systemgränser.

Slutintrycket är att Web Services är en teknik i stark utveckling, men där också mycket återstår att utveckla. Beträffande autentisering över systemgränser har man kommit en bit på väg med autentisering mellan en mänsklig användare och en Web Service. Beträffande autentisering mellan Web Services är utvecklingen i sin linda.

I rapporten redovisas inte några egna implementationer eller demonstrationer. Inom projektets ram finns dock flera av de beskrivna komponenterna implementerade i två prototyper i två olika systemmiljöer. Vår avsikt är att använda dessa prototyper för att utveckla andra säkerhetsfunktioner utöver autentisering. I första hand tänker vi studera möjligheterna till dataspårning över systemgränser.

Referenser

[BEN01] Bengtsson, A., Hunstad, A. & Westerdahl, L.: *Autentisering i nätverksbaserade system*, Användarrapport FOI-R--0331--SE, December 2001, FOI.

[BEN02] Bengtsson, A., Hunstad, A. & Westerdahl, L.: *Autonomitet vid autentisering i nätverksbaserade system*, Användarrapport FOI-R--0695--SE, December 2002, FOI.

[COH03] Cohen, F.: *Debunking SAML myths and misunderstandings*, IBM, 2003, http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html?Open&ca=daw-se-news, besökt 8 oktober 2003

[CRO03] Crowing, J.; *Driving Federated Identity to promote Web Services Services Liberty – SAML update*, 25 juni 2003, tillgänglig via http://www.drgsf.com/Cowing_LibertySAMLUpdate2.pdf, besökt 6 november 2003

[FON02] Fontana, J.: *SAML gains stream*, NetworkWorldFusion, 05-06-2002, www.nwfusion.com/news/2002/132320_05-06-2002.html, besökt 13 oktober 2003

[GAL02] Galbraith, B., Hankison, W., Hiotis, A., Janakiraman, M., D.V., P., Travedi, R. & Whitney, D.: *Professional Web Services Security*, Wrox Press Ltd, 2002

[GUR] GuruNet, http://www.atomica.com/

[HAL02] Hallam-Baker, P.: *Trust Assertion XML Infrastructure, 1st annual PKI research workshop – Proceedings*, 2002, tillgänglig via http://www.cs.dartmouth.edu/~pki02/Hallam-Baker/paper.pdf, besökt 13 november 2003

[IBMa] En samling tutorials och andra artiklar. Vissa kräver registrering vid läsning. Startsida: ibm.com/developerWorks

[IBMb] *Security in a Web Services World: A Proposed Architecture and Roadmap*, tillgänglig via http://www-106.ibm.com/developerworks/library/ws-secmap/, besökt 13 november 2003

[KAL02a] Kaler, E. (ed.): *Web Services Security (WS-Security) Version 1.0*, 5 April 2002, tillgänglig via http://www-106.ibm.com/developerworks/library/ws-secure/, besökt 13 november 2003

[KAL02b] Kaler, C. & Nadalin, A. (eds.): *Web Services Federation Language (WS-Federation)*, 8 July 2003, tillgänglig via http://www-106.ibm.com/developerworks/library/ws-fed/, besökt 13 november 2003

[KOR00] Kormann, D. P. & Rubin, A. D.; *Risks of the Passport Single Signon Protocol*, Computer Networks, Elsevier Science Press, volume 33, pages 51-58, 2000

[LIBER] *LIBERTY ALLIANCE PROJECT WHITE PAPER - Liberty Alliance & WS-Federation: A Comparative Overview*, tillgänglig via http://projectliberty.org/resources/whitepapers/wsfed-liberty-overview-10-13-03.pdf, besökt 28 oktober 2003

[MCW01] McWilliams, B.; *Stealing MS Passport's Wallet*, 2 november 2001, tillgänglig via http://www.wired.com/news/technology/0,1282,48105,00.html, besökt 22 oktober 2003

[MAD03] Madsen, P.; *The Liberty Alliance*, 1 april 2003, tillgänglig via http://www.xml.com/pub/a/ws/2003/04/01/liberty.html, besökt 28 oktober 2003

[MAL03a] Maler, E., Mishra, P. & Philpott, R. (eds.): *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*, OASIS Committee Specification, 18 July 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf, besökt 22 oktober 2003

[MAL03b] Maler, E., Mishra, P. & Philpott, R. (eds.): *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1*, OASIS Committee Specification, 18 July 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3405/oasis-sstc-saml-bindings-1.1.pdf, besökt 22 oktober 2003

[MAL03c] Maler, E., Mishra, P. & Philpott, R. (eds.): *Conformance Program Specification for the OASIS Security Assertion Markup Language (SAML) V1.1*, OASIS Committee Specification, 18 July 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3402/oasis-sstc-saml-conform-1.1.pdf, besökt 22 oktober 2003

[MAL03d] Maler, E. & Philpott, R. (eds.): *Glossary for the OASIS Security Assertion Markup Language (SAML) V1.1*, OASIS Committee Specification, 18 juli 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3401/oasis-sstc-saml-glossary-1.1.pdf, besökt 22 oktober 2003

[MAL03e] Maler, E. & Philpott, R. (eds.): *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V1.1*, OASIS Committee Specification, 18 juli 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3404/oasis-sstc-saml-sec-consider-1.1.pdf, besökt 22 oktober 2003

[MAL03f] Maler, E.: *SAML basics - A technical introduction to the Security Assertion Markup Language*, SUN Microsystems, Inc., 4 februari 2002, tillgänglig via http://www.oasis-open.org/committees/security/outreach/, besökt 22 oktober 2003

[MICRO] Microsoft: *Microsoft .NET Passport Technical Overview*, 2001, tillgänglig via http://www.microsoft.com/germany/library/resourcesmod/wp_engl_net_passport.doc, besökt 5 november 2003

[MIS03] Mishra, P (ed.).: *Differences between OASIS Security Assertion Markup Language (SAML) V1.1 and 1.0*, OASIS, 21 maj 2003, tillgänglig via http://www.oasis-open.org/committees/download.php/3412/sstc-saml-diff-1.1-draft-01.pdf, besökt 12 november 2003

[NEU93] Neuman, C. &, Kohl, J.: *The Kerberos Network Authentication Service (V5)*, RFC 1510, 1993, tillgänglig via ftp://ftp.isi.edu/in-notes/rfc1510.txt, besökt 15 november 2003

[OASIS] Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/who/

[ONE03] O'Neill, M.; Hallam-Baker, P., Mac Cann, S., Shema, M., Simon, E., Watters, P. A. & White, A.: *Web Services Security*, McGraw-Hill, 2003

[PLA01] Platt, D. & Prodromou, E. (eds.): *OASIS Security Services Use Cases And Requirements*, OASIS Consensus Draft 1, 30 maj 2001, tillgänglig via http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-reqs-01.pdf, besökt 12 november 2003

[SCH96] Schneier, B.: *Applied Cryptography: Protocols Algorithms and Source Code in C*, Sec. Ed., John Wiley & Sons, 1996

[SLE01] Slemko, M.; Microsoft Passport to Trouble, 2001, tillgänglig via http://alive.znep.com/~marcs/passport/, besökt 6 november 2003

[TAS02] Taschek, J.: *Liberty Allince or Passport*, eWeek, 24 juni 2002, tillgänglig via http://www.eweek.com/article2/0,3959,266840,00.asp, besökt 28 oktober 2003

[TOU03] Tourzan, J. (ed.); *Liberty ID-WSF Overview – Draft Version 1.0-06*, 27 juli 2003, tillgänglig via http://www.project-liberty.org/specs/draft-lib-idwsf-overview-v1.0-07.pdf, besökt 6 november 2003

[UDDI] Universal Description, Discovery and Integration protocol, http://www.uddi.org

[W3Ca] Web Services Activity, http://www.w3.org/2002/ws/, besökt 13 november 2003

[W3Cb] *Web Services Architecture Requirements*, Working Draft 19 August 2002, http://www.w3.org/TR/2002/WD-wsa-reqs-20020819, besökt 13 november 2003

[W3Cc] *XML in 10 points*, tillgänglig via http://www.w3.org/XML/1999/XML-in-10-points.html, besökt 13 november 2003

[W3Cd] *XML Schema Part 0: Primer*, W3C Recommendation, 2 maj 2001, tillgänglig via http://www.w3.org/TR/xmlschema-0, besökt 13 november 2003

[W3Ce] *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation, 24 juni 2003, tillgänglig via http://www.w3.org/TR/soap12-part0, besökt 13 november 2003

[W3Cf] *Web Services Description Language (WSDL) Version 1.2 Part 1*: Core Language, W3C Working Draft, 11 juni 2003, tillgänglig via http://www.w3.org/TR/2003/WD-wsdl12-20030611, besökt 13 november 2003

[W3Cg] *XML-Signature Syntax and Processing*, W3C Recommendation, 12 februari 2002, tillgänglig via http://www.w3.org/TR/xmldsig-core/, besökt 13 november 2003

[W3Ch] *XML Encryption Syntax and Processing*, W3C Recommendation, 10 december 2002, tillgänglig via http://www.w3.org/TR/xmlenc-core/, besökt 13 november 2003

[W3Ci] *XML Key Management Specification (XKMS) version 2.0*, W3C Working Draft 18 april 2003, tillgänglig via http://www.w3.org/TR/xkms2/#XKMS_2_0_LC2_Section_5_1, besökt 13 november 2003

[W3Cj] Raggett, D., Le Hors, A. & Jacobs, I. (eds.): *HTML 4.01 Specification*, W3C Recommendation, 24 december 1999, tillgänglig via http://www.w3.org/TR/html401/, besökt 17 oktober 2003

[XML] *Liberty Alliance Releases Phase 2 Specifications for Federated Network Identity*, XML Coverpages, 15 april 2003, tillgänglig via http://xml.coverpages.org/ni2003-04-15-b.html, besökt 5 oktober 2003

[X509] *The Directory-Authentication Framework*, Recommendation X.509, CCITT, Geneve 1989