XML:n turvallisuus web-palveluissa

Tuukka Nissinen

24.5.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu –tutkielma

Tiivistelmä

Web-palveluiden käyttö on yleistynyt valtavasti viime vuosien aikana. Tämän myötä web-palveluiden yleisesti käyttämälle XML-tiedosto- ja siirtomuodon turvallisuudelle on ryhdytty miettimään erilaisia ratkaisumalleja. Tässä tutkielmassa esitellään muutamia World Wide Web Consortiumin ja OASIS-organisaation julkaisemia suosituksia, joilla web-palveluiden, erityisesti niiden viestikerroksen, turvallisuutta saadaan lisättyä.

ACM-luokat (ACM Computing Classification System, 1998 version): E.3, H.3.5, I.7.2, K.6.5

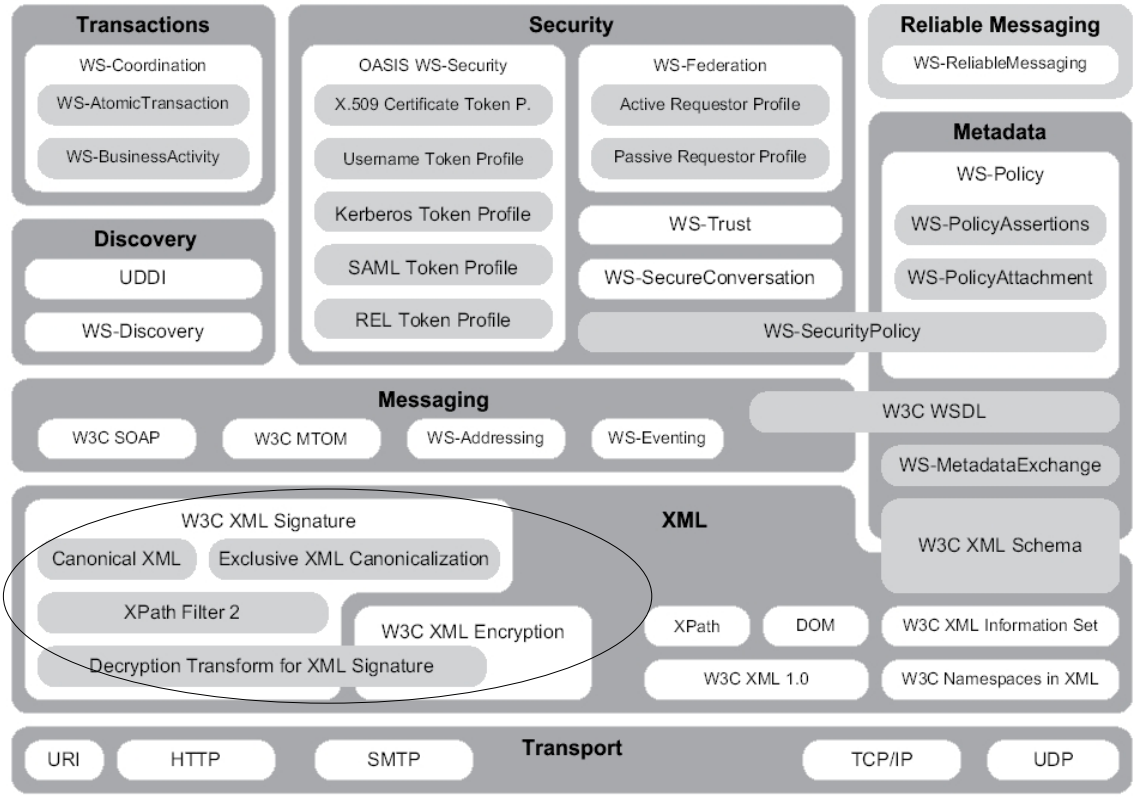
Avainsanat: Web-palvelu, XML, digitaalinen allekirjoitus, salaus

Sisällysluettelo

1. Johdanto

Web-palveluiden määrä on kasvanut lähes räjähdysmäisesti kuluvalla vuosikymmenellä. Näin ollen niiden turvallisuuteenkin on alettu kiinnittää entistä enemmän huomiota samalla kun niiden on huomattu soveltuvan myös tärkeän yritysdatan jakelukanavaksi. Tämä data oli aiemmin saatavissa ainoastaan yritysten omissa verkoissa ja yleensä vain asiakaskohtaisesti räätälöityjen sovellusten kautta. Saavutetuilla eduilla on myös hintansa: tärkeä ja arkaluonteinen tieto voi joutua entistä helpommin vääriin käsiin. Tässä tutkielmassa esitellään muutama ehdotus riskien pienentämiseksi.

Kuvassa 1 on kuvattu web-palveluiden standardit kaaviomuodossa. Tässä tutkielmassa käydään läpi sitä, kuinka XML-tasolla voidaan toteuttaa turvallisia web-palveluita. Tutkielmassa esitellään pintapuolisesti myös osa web-palvelustandardien turvallisuustasolta (Security) löytyvää OASIS-organisaation WS-Securityä, lähinnä sen yhteneväisyydet ja lisäykset XML-tason suojauksiin verrattuna. Myös käytännön esimerkkinä toimiva web-palvelu käyttää hyväkseen WS-Security -standardia.



Kuva 1. Web-palveluspesifikaatiot (Geuer-Pollmann & Claessens, 2005)

Luvussa 2 annetaan pohjatietoa aihepiiriin kuvaamalla web-palveluita, tietoturvaa ja julkisen avaimen salausmenetelmiä sekä näiden keskeisiä käsitteitä.

Luvussa 3 käydään läpi World Wide Web Consortiumin (W3C) XML Signature-, XML Encryption- sekä XML Key Management -spesifikaatiot. Läpikäynti lähtee liikkeelle syntaksista ja päätyy turvallisuusnäkökulmiin. Luvussa esitellään myös osa OASIS-organisaation WS-Security -standardista.

Luvun 4 sisältö käsittelee erilaisia web-palveluiden riskejä, niistä selviämistä W3C:n suositusten ja WS-Securityn avulla sekä yleensäkin niiden soveltamista web-palveluissa. Lisäksi luvussa kuvataan testatun esimerkin avulla web-palvelun toteutusta hyödyntäen ja soveltaen W3C:n suosituksia ja WS-Securityä.

Luvussa 5 tehdään yhteenveto käsitellyistä asioista.

2. Keskeisiä käsitteitä

Luvussa kuvataan aihepiirin keskeisiä käsitteitä, kuten XML, web-palvelu, tietoturva sekä digitaalinen allekirjoitus.

2.1. XML

XML (eXtensible Markup Language) on World Wide Web Consortiumin (W3C) kehittämä metakieli; kieli, jolla voidaan kuvata toisia kieliä. XML on myös nimensä mukaisesti merkkauskieli, jolla kuvataan tekstin rakenne, ja jolla erotetaan tekstin looginen rakenne varsinaisesta sisällöstä. XML on SGML:n (Standard Generalized Markup Language) yksinkertaistettu ja helppokäyttöinen osajoukko (Holzner, 2001).

XML:n suunnittelutavoitteet (Bray & al., 2006):

1. XML:n tulee olla käytettävissä suoraan Internetin kautta
2. XML:n tulee tukea erilaisia sovelluksia.
3. XML:n tulee olla yhteensopiva SGML:n kanssa.
4. XML-dokumentteja käsitteleviä ohjelmia on oltava helppo kirjoittaa.
5. Valinnaisten ominaisuuksien määrä XML:ssä on pidettävä minimissään, mieluiten nollassa
6. XML-dokumenttien tulisi olla ihmisten luettavissa ja kohtuullisen selkeitä.
7. XML-suunnitelma tulisi valmistaa nopeasti.
8. XML-suunnitelman tulisi olla muodollinen ja tiivis.
9. XML-dokumenttien tulisi olla helposti luotavia.
10. XML-merkkauksen suppeus ei ole tärkeää.

Kuten edellä olevasta listasta voidaan päätellä, XML:n tavoite on olla yksinkertainen, joustava ja helppokäyttöinen (McLaughlin, 2002). XML:n tärkeimpinä etuina pidetäänkin sen siirrettävyyttä ja yhteiskäytettävyyttä. XML:n siirrettävyydellä tarkoitetaan sitä, että koska se on yksinkertaista tekstiä, se voidaan siirtää ongelmitta alustalta toiselle. XML:n yhteiskäytettävyys tulee taas siitä, että se ei rajaa paljoakaan dokumentin ele-

menttejä ja sisältöä, vaan sallii käyttäjän tehdä siitä tarkoituksenmukaisen ja julkaista mallin sitten muidenkin käytettäväksi.

2.1.1. XML-dokumentin rakenne

XML-dokumentin tulisi alkaa XML-esittelyllä, joka on samalla prosessointiohje. Esittelyssä määritetään käytettävä XML-versio. Lisäksi esittelyosiossa voidaan kertoa esimerkiksi dokumentissa käytettävä merkkikoodaus tai onko dokumentti itsenäinen vai tarvitaanko käsittelyyn dokumenttityyppimäärittely (DTD). Esittelyosio ei ole tiukasti pakollinen, mutta se lisää koodin siirrettävyyttä, joten sen käyttöä suositellaan (North & Hermans, 2000). XML-esittely voi olla esimerkiksi seuraavanlainen:

```
<? xml version=”1.0” encoding=”UTF8” standalone=”yes”?>
```

XML-dokumentissa esittelyn ja varsinaisen sisällön väliin voidaan sijoittaa mm. DOCTYPE-määrityksiä tai prosessointikomentoja, kuten stylesheet-määrityksiä, esimerkiksi

```
<! DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Transitional//EN”
“http://www.w3c.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
```

tai

```
<?xml-stylesheet href=”perustyyli.xsl” type=”text/xsl”?>
```

Varsinainen dokumentin sisältöosuus alkaa juurielementillä (esim. <Juuri>). Juurielementti voi sisältää attribuutteja, toisia elementtejä tai tekstisisältöä. Juurielementin sulkeva tagi (esim. </Juuri>) täytyy myös olla dokumentin viimeisenä eli dokumentin datasisältö on kokonaisuudessaan juurielementin sisällä. Kaikki elementit voivat olla mielivaltaisesti nimettyjä kuitenkin niin, että nimi alkaa alaviivalla tai kirjaimella. Elementtien täytyy olla kulmasulkeiden sisällä. Lisäksi elementtien nimet ovat merkkiherkkiä, eli iso ja pieni kirjain on eri asia. Kaikki elementit täytyy myös sulkea, tosin tyhjä elementti voidaan avata ja sulkea samalla tagilla (esim. <tyhja />). Elementti voi sisältää myös attribuutteja (esim. <elementti nimi=”norsu”> eli elementillä on attribuutti nimeltä nimi ja sen arvo on norsu) (McLaughlin, 2002).

2.1.2. Hyvinmuodostettu XML-dokumentti

W3C:n XML 1.1. -määrityksessä (Bray & al., 2006) kerrotaan syntaksisäännöt, joita dokumentin tulee noudattaa ollakseen *hyvinmuodostettu* (well-formed). Sääntöjä on varsin paljon, joten tässä yhteydessä läpikäydään vain yleisimmät rajoitteet.

Kuvassa 2 on kuvattu hyvinmuodostettu XML-dokumentti. Hyvinmuodostetun XML-dokumentin täytyy sisältää täsmälleen yksi juurielementti, joka sisältää kaikki muut mahdolliset elementit. Toiseksi elementtien täytyy alkaa ja päättyä ja vieläpä saman elementin sisällä. Kolmanneksi elementtien mahdollisilla attribuuteilla täytyy olla arvo ja sen tulee olla lainausmerkeissä (McLaughlin, 2002).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <henkilo>
      <etunimi>Dumbo</etunimi>
      <sukunimi>Korvanen</sukunimi>
   </henkilo>
```

Kuva 2. Esimerkki hyvinmuodostetusta XML-dokumentista

2.1.3. Kanonisointi

Kanoninen XML (Boyer, 2001) on XML:n seuralaisstandardi. Kyseessä on periaatteessa tiukka XML-syntaksi. Kaksi XML-dokumenttia voi olla tietosisällöltään samanlaisia, mutta eri tavoin järjesteltyjä: eroja voi olla esimerkiksi rakenteessa, attribuuttien järjestyksessä tai merkkikoodauksessa. Kun XML-dokumentit kanonisoidaan, niitä voidaan vertailla keskenään tavu kerrallaan. Kanonisen XML:n syntaksissa loogisesti samat dokumentit ovat tavuvertailussa identtisiä (Holzner, 2001).

Kanonisoinnin yhteydessä tehtävät toimenpiteet (Boyer, 2001):

- otetaan käyttöön UTF-8-koodaus
- normalisoidaan rivinvaihdot
- normalisoidaan attribuuttien arvot
- korvataan merkki- ja entiteettiviittaukset
- korvataan CDATA-osiot niiden sisällöllä
- poistetaan XML:n esittelyosio ja DOCTYPE-määritykset

- muutetaan tyhjät elementit aloitus-lopetus-tagipareiksi.
- normalisoidaan tyhjämerkit elementeistä
- säilytetään elementtien sisällöissä olevat tyhjämerkit (lukuun ottamatta rivinvaihtojen normalisoinnissa poistetuttuja merkkejä)
- muutetaan attribuuttiarvojen rajamerkit lainausmerkeiksi
- korvataan attribuuttiarvojen ja elementtien sisällön erikoismerkit merkkiviittauksiksi
- poistetaan kaikista elementeistä turhat nimiavaruusmääritykset
- lisätään oletusattribuutit jokaiseen elementtiin
- asetetaan jokaisen elementin nimiavaruusmääritykset sekä attribuutit sanastonmukaiseen järjestykseen

Lisäksi kommentit voidaan joko jättää kanoniseen dokumenttiin tai poistaa ne kanonisoinnin yhteydessä. Taulukossa 1 on esimerkki XML-dokumentista ennen ja jälkeen kanonisoinnin.

Taulukko 1. Esimerkki kanonisoidusta dokumentista.

Dokumentti	Kanonisoitu muoto ilman kommentteja
<?xml version="1.0"?> <?xml-stylesheet href="doc.xsl" type="text/xsl" ?> <!DOCTYPE doc SYSTEM "doc.dtd"> <doc>Hello, Dumbo! <!-- Comment 1 --></doc> <?pi-without-data ?> <!-- Comment 2 --> <!-- Comment 3 -->	<?xml-stylesheet href="doc.xsl" type="text/xsl" ?> <doc>Hello, Dumbo!</doc> <?pi-without-data?>

Kanonisesta XML:stä julkaistaan piakkoin versio 1.1, sillä se on jo saavuttanut ”ehdotettu suositus” -vaiheen tammikuussa 2008.

Vielä kanonista XML:ää tiukempi syntaksi löytyy Exclusive XML Canonicalization-spesifikaatiosta eli poissulkevasta XML-kanonisoinnista (Boyer & al., 2002). Tässä spesifikaatiossa esiteltyä mallia voidaan käyttää sellaisissa tapauksissa, joissa on tarve erottaa dokumentin osa sitä kapseloivasta dokumentista. Esimerkiksi joissain tapauksissa voi olla tarve tietosisällön kattavalle digitaaliselle allekirjoitukselle, joka ei rikkoudu, vaikka alidokumentti poistettaisiin alkuperäisestä viestistä ja lisätään eri yhteyteen. Tämä vaatimus toteutuu käytettäessä poissulkevaa XML-kanonisointia.

Poissulkevaa kanonisointia voidaan käyttää XML-allekirjoituksen ja XML-salauksen Transform- tai CanonicalizationMethod-algoritmina (ks. kohdat 3.1.1. ja 3.2.1.).

2.1.4. Transformaatiot

Yksi syy XML:n suosioon siirrettävän datan muotona on sen muokattavuus eli transformoitavuus. Edellisessä kohdassa esitelty kanonisointi on eräs transformointimuoto; muita transformaatioita ovat esimerkiksi base64-koodaus, XPath-suodatus, XSLT-transformaatio sekä kapseloitu (enveloped) transformaatio (Dournaee, 2002).

W3C on julkaissut kolme eri suositusta XML:n transformoinnista, mutta ne liittyvät varsin läheisesti toisiinsa, joten ne käsitellään tässä samassa yhteydessä. Suositukset ovat XSL (Berglund, 2006), XSLT (Berglund & al., 2007) sekä XPath (Clark, 1999). XSL kostuu kahdesta osasta: se on kieli XML-dokumenttien transformointiin ja lisäksi se on XML-sanasto XML-dokumenttien muotoilun määrittelyä varten. XSLT taas on kieli, joka määrittää dokumentin konvertoinnin formaatista toiseen. XPathin avulla voidaan viitata suureen määrään elementtien ja attribuuttien nimiä ja arvoja määrittämällä polku viitattuun elementtiin suhteessa prosessoitavaan elementtiin (McLaughlin, 2002).

2.2. Web-palvelu

Web-palvelu-termin (eng. web service, toinen suositeltu termi on WWW-sovelluspalvelu) määritelmiä on lähes yhtä paljon kuin on määrittelijöitä. Termille ei siis ole vielä löytynyt vakiintunutta määritelmää. W3C:n web-palvelusanasto tarjoaa kuitenkin seuraavanlaisen määritelmän: ”Web-palvelu on verkon yli tapahtuvaa koneiden välistä yh-

teistoimintaa tukemaan suunniteltu ohjelmisto. Web-palvelulla on tietokoneen käsiteltävässä muodossa kuvattu rajapinta, yleensä WSDL-tiedosto. Toiset ohjelmistot ovat yhteydessä web-palveluun sen kuvauksessa esitetyllä tavalla, tyypillisesti käyttäen HTTP-protokollaa ja SOAP-viestejä."

Yksi tärkeimmistä web-palvelun ominaisuuksista on siis yhteentoimivuus (interoperability). Lisäksi sovellusten tulee voida kommunikoida keskenään. Tämä vaatii luonnollisesti yhteisen kielen ja tähän tarkoitukseen XML sopii erinomaisesti. Suurin osa web-palveluista käyttää SOAP:ia vuorovaikutuksessaan asiakasohjelmiin (McLaughlin, 2002).

2.2.1. WSDL

Web Services Description Language (WSDL) on web-palveluiden XML-pohjainen kuvauskieli (Christensen & al., 2001). Yleisimmin käytössä oleva versio on 1.1, mutta kesäkuussa 2007 on julkaistu myös uusi versio 2.0 (Chinnici et al., 2007). WSDL-tiedosto on tarkoituksella pidetty mahdollisimman yksinkertaisena, eli siihen kerätään vain palvelun kutsumisen kannalta oleelliset tiedot, kuten sanomien rakenne, sanomanvälitysmalli, siirtoprotokolla sekä palvelun sijainti. Kuvaustiedostoa voidaan kuitenkin tarvittaessa laajentaa käyttämällä laajennuksia

WSDL-tiedosto voidaan jakaa kahteen osaan, palvelun rajapinnan määritykseen sekä toteutuksen määritykseen. Juurielementtinä on *definitions* ja sen lapsielementteinä ovat

- *types* (määrittelee sanomissa käytettävät tietotyypit)
- *message* (kuvaa yksittäisen sanoman rakenteen abstraktilla tasolla)
- *operation* (kuvaa palvelun tarjoaman toiminnon abstraktilla tasolla)
- *portType* (kokoaa palvelun tarjoamat toiminnot yhteen, WSDL 2.0 -määrityksessä vaihtui nimelle *interface*)
- *binding* (kuvaa palvelun kutsumiseen käytettävät protokollat, kuten SOAP, HTTP GET/POST, MIME)
- *port* (kertoo palvelun sijainnin, esim. URI-viittauksella, WSDL 2.0-määrityksessä vaihtui nimelle *endpoint*)
- *service* (kokoaa yhteen palvelun portit)

Liitteessä 1 on esimerkki WSDL-tiedostosta. Kyseinen tiedosto kuvaa web-palvelun, jota käsitellään kohdassa 4.4.

2.2.2. UDDI

Universal Description, Discovery, and Integration (UDDI, http://www.uddi.org/) on protokolla, joka määrittää standarditavan julkaista ja hakea web-palveluita. UDDI sisältää XML-pohjaisen, hajautetun ja alustariippumattoman rekisterin, johon voidaan lisätä palveluita yksinkertaisen rekisteröintiprosessin kautta. Tämän jälkeen rekisteriä voidaan käyttää palvelun paikallistamiseen monella yksityiskohtaisuuden tasolla ja käyttöönottoon sekä palvelun metadatan ylläpitämiseen (OASIS Open, 2004).

UDDI-rekisteri koostuu kolmesta osiosta: valkoisista, keltaisista ja vihreistä sivuista. Valkoisilla sivuilla kerrotaan web-palvelun tarjoajan yhteystiedot, kuten nimi ja yhteystiedot sekä palvelun kuvaus ja tunnistelistaus. Keltaisilla sivuilla web-palvelut luokitellaan esimerkiksi sijaintinsa tai tyyppinsä mukaan käyttäen yleisiä taksonomioita. Vihreillä sivuilla kerrotaan yksityiskohtaisesti, kuinka palveluun saadaan yhteys (Stencil Group, 2002).

UDDI on OASIS-organisaation (http://www.oasis-open.org/) hyväksymä standardi, joka on versiossa 3.0 (julkaistu 2004) (OASIS Open, 2004).

2.2.3. SOAP

Simple Object Access Protocol (SOAP) on yksinkertainen ja kevyt XML-pohjainen protokolla rakenteisten viestien siirtoon hajautetuissa ympäristöissä. SOAP:in suunnittelun päämääriä ovat yksinkertaisuus sekä laajennettavuus. Yksinkertaisuudella haetaan helppoa hallittavuutta ja laajennettavuudella pyritään säilyttämään käyttökelpoisuus myös tulevaisuuden kehityksessä. SOAP on nykyisin W3C:n suositus, viimeisin versio on julkaistu 27.4.2007 ja se on versionumeroltaan 1.2 (Gudgin & al., 2007).

SOAP-suositus muodostuu kolmesta pääkomponentista: sanomarakenteesta, prosessointimallista sekä sidosten yleisrakenteesta varsinaisille siirtoprotokollille.

SOAP-viesti on XML-dokumentti, joka koostuu yleensä kolmesta pääelementistä: kuorielementistä, otsikkoelementistä sekä runkoelementistä. Kuorielementti toimii viestin kehyksenä, otsikkoelementti tarjoaa mekanismin viestin laajennuksille ja runkoelementti sisältää varsinaisen siirrettävän tietorakenteen. Kuvassa 3 on esimerkki SOAP-kirjekuoresta.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
 envelope">
  <env:Header>
    <n:alertcontrol
xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
     <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
     </m:alert>
  </env:Body>
</env:Envelope>
```

Kuva 3. Esimerkki SOAP-kirjekuoresta (Gudgin & al., 2007)

2.3. Tietoturva

Tietoturvalla pyritään toteuttamaan neljä periaatetta: luottamuksellisuus (confidentiality), eheys (integrity), kiistämättömyys (non-repudiation) sekä oikeellisuus (authentication).

Järvisen (2003) mukaan *luottamuksellisuus* tarkoittaa sitä, että tietoihin pääsee käsiksi vain siihen oikeutetut henkilöt. Luottamuksen säilyttämisen motiivina ovat sekä palvelun tarjoajan oma etu, mutta yhä useammin myös laki; esimerkiksi henkilötietolaki suojaa henkilörekistereitä. Luottamuksellisuuden turvaamisessa salausmenetelmillä on varsin keskeinen rooli. Salauksella voidaan varmistaa tietojen luottamuksellisuus myös haavoittuvimmissa tilanteissa, kuten siirron ja säilytyksen aikana.

Eheys merkitsee sitä, että valtuuttamattomat henkilöt tai prosessit eivät pääse muuttamaan tietoja. Salaustekniikoiden avulla ei varsinaisesti voida estää eheyden rikkoutumista, mutta niiden avulla voidaan havaita mahdolliset muutokset. Tiedosta laskettavat tiivisteet paljastavat muutokset ja salauksella varmistetaan tiivisteiden turvallisuus (Krutz & Vines, 2003).

Ruohonen (2002) mainitsee *kiistämättömyyden* sitovan kohteen tapahtumaan eli sillä taataan se, ettei lähettäjä eikä vastaanottaja pysty kieltämään siirtämäänsä tietoa. Sanoman vastaanottaja voi varmistua lähettäjän olevan se, joka se sanoo olevansa ja vastaavasti lähettäjä voi varmistua vastaanottajasta. Esimerkiksi kauppiaan on lisättävä elektroniseen sopimukseen digitaalinen allekirjoitus, jolla voidaan jälkeenpäin varmistua allekirjoittajasta.

Oikeellisuuden tehtävänä on varmistaa eri osapuolten henkilöllisyys, jos kyseessä on ihmiset, tai aitous, jos kyseessä on ohjelmat tai koneet. Kyseessä on tietoturvan kriittisin osa-alue, sillä ilman tätä aiemmin mainitut kolme periaatetta menettävät merkityksensä. Esimerkiksi luottamuksellisuuden pettäessä ulkopuolinen henkilö saa haltuunsa hänelle kuulumatonta tietoa, mutta jos oikeellisuus pettää, niin sama henkilö pääsee lukemisen lisäksi tuottamaan ja muokkaamaan tietoa toisen nimissä. Tietotekniikassa oikeellisuus todennetaan useimmiten salasanalla (Kerttula, 2000).

2.4. PKI ja digitaalinen allekirjoitus

Epäsymmetrisiä salausalgoritmeja kutsutaan yleensä julkisen avaimen salausmenetelmäksi (PKI, Public Key Infrastructure). Julkisen avaimen menetelmässä käytetään kahta avainta, joista toinen on julkinen ja toinen ainoastaan omistajan hallussa oleva salainen avain. Lähettäessään viestin vastaanottajalle, lähettäjä käyttää salaukseen vastaanottajan julkista avainta. Vastaanottaja käyttää omaa salaista avaintaan vastaavasti purkaessaan salatun viestin. Digitaalisessa allekirjoituksessa avaimia käytetään juuri päinvastoin; salaisella avaimella salataan allekirjoitettava tieto ja viestin avaaja käyttää julkista avainta (Pajukoski, 2004).

Digitaalisen allekirjoituksen luominen aloitetaan laskemalla yksisuuntaisen funktion avulla tiiviste allekirjoitettavasta datasta. Laskentaan käytetään yleisesti SHA-1 (Secure Hash Algoritm) tai MD-5 (Message Digest version 5) -algoritmeja. Allekirjoittaja salaa tiivisteluvun omalla salaisella avaimellaan, jonka käyttö on yleensä suojattu vähintäänkin salasanalla. Näin ollen kukaan muu kuin avaimen haltija ei voi toimia allekirjoittajana. Salauksen tuloksena saadaan digitaalinen allekirjoitus, joka liitetään allekirjoitettavan datan perään ja viesti voidaan lähettää vastaanottajalleen (Rinne, 2002).

Vastaanottaja voi tarkistaa allekirjoituksen avaamalla lähettäjän julkisella avaimella tiivisteen, laskemalla itse uuden tiivisteluvun ja vertaamalla näitä kahta tiivistettä keskenään. Jos tiivisteet täsmäävät, dataa ei ole muutettu allekirjoituksen jälkeen. Digitaalinen allekirjoitus varmistaa siis allekirjoitetun datan eheyden kokonaisuudessaan, ei esimerkiksi viimeistä sivua, kuten käsin allekirjoitetuissa sopimuksissa yleensä. Tässä yhteydessä täytyy kuitenkin ottaa huomioon, että digitaalinen allekirjoitus suojaa tiivisteen avulla datan eheyden, mutta ei siis salaa kohteena olevaa dataa. Digitaalinen allekirjoitus vahvistaa kuitenkin datan alkuperän eli jos tiivistettä suojaava salaus aukeaa lähettäjän julkisella avaimella, voidaan olla varmoja, että allekirjoittajan salaista avainta on käytetty allekirjoituksen luomiseen. Mikään ei myöskään estä salaamasta jo allekirjoitettua dataa, jolloin saadaan varmistettua myös datan luottamuksellisuus (Rinne, 2002).

Digitaalinen allekirjoitus on saanut juridisen aseman useiden maiden lainsäädännössä, erityisesti julkishallinnon puolella. Suomessa sähköistä ja digitaalista allekirjoitusta käsittelevä laki hyväksyttiin vuonna 2002. Laki perustuu EU-direktiiviin sähköisistä allekirjoituksista (Rinne, 2002; Perttula, 2002).

Laki sähköisistä allekirjoituksista astui voimaan helmikuun alussa vuonna 2003 (FINLEX, 2003).

3. W3C:n suosituksia ja WS-Security

Luvussa kuvataan kolme World Wide Web Consortiumin XML:n turvallisuuteen liittyvää suositusta; XML-allekirjoitus, XML-salaus sekä XML-avainten hallinta. Lisäksi luvussa esitellään osa OASIS-organisaation WS-Security-standardista, näkökulmana lähinnä yhteneväisyydet ja lisäykset esiteltyihin W3C:n.suosituksiin verrattuna.

3.1. XML-allekirjoitus

Vuonna 1999 W3C ja IETF (Internet Engineering Task Force) yhdistivät voimansa luodakseen määrityksen, joka tukisi dokumenttien digitaalista allekirjoitusta XML-muodossa (XML Signature). W3C:n asiantuntijuus XML:n saralla yhdistettynä IETF:n asiantuntemukseen kryptografian rintamalla sai aikaan suosituksen, jossa esiteltiin XML-muotoisen allekirjoituksen luominen sekä esitystapa. Suosituksen viralliseksi syntymäajaksi saatiin lopulta helmikuun 12. päivä vuonna 2002 (Galbraith & al., 2002).

XML Signature -spesifikaatiota suunniteltaessa laajennettavuudelle ja joustavuudelle annettiin suuri merkitys. Niinpä XML-allekirjoitusta voidaan käyttää allekirjoittamaan mitä tahansa digitaalista sisältöä, pääpainon ollessa kuitenkin XML-dokumenttien allekirjoittamisessa. XML-allekirjoitus on itsessään hyvin muodostettu XML-dokumentti, joten sitä voidaan käsitellä ja lukea normaalisti lukuunottamatta itse allekirjoitusosiota sekä tiivistearvoa, vaikka käytössä ei olisikaan tarvittavia varmennusmahdollisuuksia. Kaikki allekirjoituksen käsittelyyn tarvittava tieto, jopa varmennusinformaatio, voidaan upottaa allekirjoitukseen (Dournaee, 2002).

Perinteiseen digitaaliseen allekirjoitukseen, kuten esimerkiksi PKCS (Public Key Cryptography Standards), verrattuna XML-allekirjoituksella on useita kehittyneitä ominaisuuksia. Kun perinteistä digitaalista allekirjoitusta voidaan käyttää ainoastaan koko dokumentin allekirjoittamiseen, XML-allekirjoitus soveltuu tämän lisäksi myös dokumentin tietyn osan allekirjoittamiseen. Tällaisessa tapauksessa muu osa dokumentista on edelleen muokattavissa ilman allekirjoituksen rikkoutumista. Näin ollen dokumentin osiot voidaan myös allekirjoittaa eri aikaan ja eri henkilöiden toimesta; allekirjoitusten määrää ei ole rajoitettu. Kyseistä ominaisuutta voi hyödyntää esimerkiksi täydennettä-

vissä työryhmädokumenteissa ja muistioissa (Dournaee, 2002).

XML-allekirjoitus on siis vain tapa luoda yhteys avaimen ja viitatun datan välille. Suositus ei ota kantaa käytettäviin avaimiin, vaan kaikki sen tukemat algoritmit ja avaimet ovat käytettävissä. Suosituksessa ei myöskään oteta kantaa avaimien ja niiden omistajien välisen yhteyden varmistamiseen (ks. kohta 3.3.) eikä myöskään viitattavan ja allekirjoitettavan datan sisältöön. XML-allekirjoitus on tärkeä osa XML:n turvallisuutta ajatellen, mutta näin ollen se ei yksistään riitä ratkaisemaan palveluiden tietoturva- ja luottamusongelmia (Bartel & al., 2002).

3.1.1. Syntaksi

XML-allekirjoitukset esitetään Signature-elementin avulla. Kyseisellä elementillä voi olla ID-attribuutti, joka yksilöi allekirjoituksen useamman allekirjoituksen tapauksessa. Kuvassa 4 esitellään allekirjoituksen rakenne (? tarkoittaa nolla tai yksi ilmentymää, + yksi tai useampi ilmentymää ja * nolla tai useampi ilmentymää) (Bartel & al., 2002):

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

Kuva 4. XML-allekirjoituksen rakenne (Bartel & al., 2002)

Taulukko 2. XML-allekirjoituksen osat (Bartel & al., 2002)

Elementti	**Kuvaus**
Signature	Allekirjoituksen juurielementti, joka identifioi allekirjoituksen. Elementin valinnaista Id-attribuuttia voidaan käyttää yksilöimiseen, jos allekirjoituksia on useampia.

SignedInfo	Sisältää tiedon kaikista allekirjoitettavista kohteista. Elementin valinnaista Id-attribuuttia voidaan käyttää, jos on tarvetta viitata toisista allekirjoituksista tai kohteista.
CanonicalizationMethod	Määrää SignedInfo-elementin kanonisoimiseen käytettävän algoritmin pakollisesssa Algorithm-attribuutissaan.
SignatureMethod	Määrää pakollisessa Algorithm-attribuutissaan salausalgoritmin, jota käytetään kanonisoidun SignedInfo-arvon muuntamiseen SignatureValue-elementin arvoksi.
Reference	Yksilöi allekirjoitettavan kohteen valinnaisen URI-attribuuttiviittauksen avulla.
Transforms	Sisältää listan allekirjoitettavalle datalle ennen hajautusta tehdyt toimenpiteet (esimerkiksi kanonisointi, tiivistys, skeeman validointi jne.).
DigestMethod	Määrittää käytettävän algoritmin, jota käytetään tiivisteen laskemiseen allekirjoitettavasta datasta.
DigestValue	Sisältää lasketun tiivisteen.
SignatureValue	Sisältää allekirjoituksen arvon. Valinnainen Id-attribuutti.
KeyInfo	Osoittaa allekirjoituksen varmentamiseen käytettävän avaimen ja sen tiedot (esim. nimi ja sertifikaatti). Valinnainen elementti, sillä kaikissa tapauksissa avaimeen liittyvää informaatiota ei haluta tässä yhteydessä kertoa. Jos tämä elementti puuttuu, täytyy avaimen tiedot kertoa muulla tapaa sovelluksessa.
Object	Voi sisältää mitä tahansa sisällytetyksi haluttua dataa. Valinnainen elementti.

Edellä esitetyssä taulukossa 2 mainittujen osien lisäksi XML-allekirjoitukseen kuuluu muita, harvemmin käytettäviä elementtejä. Ne ovat yleensä KeyInfo-elementin lapsielementtejä ja liittyvät käytettävään avaimeen. Elementeistä mainittakoon KeyName, jolla voidaan antaa lisäinformaatiota avaimeen liittyen, sekä KeyValue, joka voi sisältää yksittäisen julkisen avaimen käytettäväksi allekirjoituksen varmistamiseen. Käytettävään avainpariin liittyviä elementtejä ovat esimerkiksi Pretty Good Privacyyn liittyvä PGPData-elementti tai DSA-salaukseen liittyvä DSAKeyValue (Galbraith & al., 2002).

Kuvassa 5 on kuvattu XML-allekirjoituksen eri tyypit. Niitä on kolmea eri tyyppiä: ulkoinen (detached), kapseloiva (enveloping) sekä kapseloitu (enveloped) allekirjoitus. *Ulkoisella allekirjoituksella* tarkoitetaan sitä, että allekirjoitus on joko kokonaan eri dokumentissa tai vaihtoehtoisesti samassa dokumentissa, mutta erillään allekirjoitettavasta datasta. *Kapseloivassa allekirjoitustyypissä* allekirjoitettava tieto sijoitetaan Signature-elementin sisälle. *Kapseloidussa allekirjoituksessa* taas allekirjoitus sijoitetaan allekir-

joitettavan tiedon sisälle (Galbraith & al., 2002).

```
<!-- Kapseloitu allekirjoitus-->
<Dokumentti> ...
<Signature> ...</Signature>
</Dokumentti>


<!-- Kapseloiva allekirjoitus-->
<Signature>...
<Dokumentti>...</Dokumentti>
</Signature>


<!-- Ulkoinen allekirjoitus-->
<Signature></Signature>
```

Kuva 5. Allekirjoitustyypit (Dournaee, 2002).

3.1.2. Käsittelysäännöt

XML Signature -spesifikaatio (Bartel & al., 2002) kuvaa joukon allekirjoituksen luomisen ja myöhäisemmän allekirjoituksen varmennuksen aikana tehtäviä toimenpiteitä.

XML-allekirjoituksen luomista kutsutaan ytimen tuottamiseksi, joka koostuu kahdesta eri operaatiosta; viitteiden tuottamisesta sekä allekirjoituksen tuottamisesta. XML-allekirjoituksen varmistus, ytimen varmistaminen, on jaettu niin ikään kahteen osaan; viitteiden varmistamiseen sekä allekirjoituksen varmistamiseen (Dournaee, 2002).

Allekirjoituksen luonnin pakolliset toimenpiteet ovat Reference- ja SignatureValue-elementtien luonnit. Reference-elementti luodaan seuraavasti jokaiselle dataobjektille:

1. Tee dataobjektille kaikki halutut muokkaustoimenpiteet. Toimenpiteitä voi olla useampia.
2. Laske muokatusta dataobjektista tiivistearvo.
3. Luo Reference-elementti asettamalla edellisessä kohdassa luotu tiivistearvo DigestValue-elementtiin sekä tiivistearvon laskemiseen käytetty algoritmi DigestMethod-elementtiin. Lisäksi jos kohdassa yksi suoritettiin muokkaustoimenpiteitä, niin lisää ne Transforms-elementtiin suoritusjärjestyksessään.

Allekirjoitus luodaan seuraavilla operaatioilla:

1. Luo SignedInfo-elementti ja sille lapsielementeiksi SignatureMethod, CanonicalizationMethod sekä kaikki Reference-elementit.
2. Kanonisoi SignedInfo-elementti CanonicalizationMethod-elementissä mainitulla algoritmilla ja laske sitten SignedInfo-elementissä mainitulla algoritmilla SignedInfo-elementistä arvo ja aseta se SignatureValue-elementtiin.
3. Luo Signature-elementti ja sille lapsielementeiksi aiemmin luotu SignedInfo-elementti ja juuri luotu SignatureValue-elementti sekä tarpeen mukaan valinnainen KeyInfo-elementti ja valinnaiset Object-elementit attribuutteineen.

Allekirjoituksen varmentaminen tapahtuu vastaavalla tavalla kahdessa vaiheessa: viitteiden (Reference) varmentaminen sekä allekirjoituksen (Signature) varmentaminen. Ensimmäinen varmennus kertoo, ovatko kaikki viitteet säilyneet ehyinä ja koskemattomina ja jälkimmäinen varmennus kertoo, onko allekirjoitus aito ja sitä kautta myös kiistämätön.

Viitteiden varmennus tapahtuu seuraavalla tavalla:

1. Kanonisoi SignedInfo-elementti CanonicalizationMethod-elementissä kerrotun algoritmin mukaisesti. Tässä vaiheessa täytyy varmistaa, ettei kanonisoinnilla ole sivuvaikutuksia, jotka voisivat vaarantaa toiminnan.
2. Hae tiivistettävä data esimerkiksi URI-viittauksen perusteella ja muokkaa sitä Transforms-elementissä kerrotuilla tavoilla.
3. Laske datasta tiiviste Digestmethod-elementissä ilmoitettavalla algoritmilla.
4. Vertaa saatua tiivistearvoa DigestValue-elementistä löytyvään arvoon. Jos arvot eivät täsmää, varmennus epäonnistuu.
5. Toista kohdat 2-4 jokaisen Reference-elementin osalta.

Allekirjoitus varmennetaan seuraavasti:

1. Hae avainta koskeva tieto KeyInfo-elementistä tai tarvittaessa ulkoisesta lähteestä.

2. Käytä edellisessä kohdassa hankittua avaintietoa salatun tiivistearvon laskemiseen SignatureMethod-elementin arvosta CanonicalizationMethod-elementin tiedon avulla.
3. Vertaa saatua arvoa ja aiemmin saatua KeyInfo-elementin arvoa. Jos arvot eivät täsmää, varmennus epäonnistuu.

3.1.3. XML-allekirjoituksen turvallisuus

XML-allekirjoituksella voidaan varmistua siitä, että allekirjoitettu data on siinä muodossa, missä se oli lähettäjältä lähtiessään, ja että allekirjoitus on tehty allekirjoittajan salaisella avaimella. XML Signature -suosituksessa (Bartel & al., 2002) käydään läpi muitakin huomioitavia turvallisuusseikkoja kyseessä olevaa tekniikkaa käytettäessä. Kaikki mainittavat seikat liittyvät olennaisesti dokumentin transformointiin.

Vain allekirjoitettu on turvallista. Tällä tarkoitetaan sitä, että se osa dokumentista, johon allekirjoitus kohdistuu kaikkien muokkaustoimenpiteiden jälkeen, on luotettavaa. Muu osa voi muuttua allekirjoituksen jälkeen ja allekirjoitus pysyy muutoksista huolimatta validina. Lisäksi kannattaa muistaa, että allekirjoitus takaa ainoastaan allekirjoitetun osion eheyden, ei sen luottamuksellisuutta.

Vain nähtävissä oleva tulee allekirjoittaa. Alkuperäinen data voi erota transformoidusta datasta ja tästä syystä allekirjoittajan olisi hyvä nähdä se lopullisessa muodossaan ennen allekirjoitusta. Tämä voi vaatia dokumentista esimerkiksi kuvaruutukaappauksen, joka taas voi hankaloittaa allekirjoitetun datan käsittelyä myöhemmissä vaiheissa.

Täytyy tietää, mitä allekirjoitetaan. Dokumentissa voi olla viittauksia toisiin dokumentteihin tai esimerkiksi tyylitiedostoihin. Tällaisessa tapauksessa myös viitatut tiedostot tulisi sisällyttää allekirjoitettuun dataan.

Edellä mainittujen lisäksi XML-allekirjoituksen turvallisuuteen vaikuttavat normaalit tietoturvaseikat, kuten turvallisuusmalli, käytettävät algoritmit ja avainten pituudet, yms. aina yrityksen henkilöstön valveutuneisuuteen ja yleiseen tietoturvapolitiikkaan saakka.

3.2. XML-salaus

W3C:n 10.12.2002 julkaistussa XML Encryption Syntax and Processing -suosituksessa (Imamura & al., 2002) kuvataan menetelmä datan salaukseen ja XML-muotoisen tuloksen esittämiseen. Salattava data voi olla mitä tahansa, mutta pääpaino suosituksessa on XML-dokumenteissa ja sen elementeissä. Salauksen tuloksena saadaan XML-salaus -elementti (EncryptedData), joka sisältää salatun datan lapsielementissään tai viittaa siihen URI-viittauksella.

XML-salausta suunniteltiin XML-allekirjoituksen suositusta hyväksikäyttäen ja näin ollen ne ovatkin erityisen hyvin toisensa huomioonottavia.

Kokonaisen dokumentin salaaminen onnistuu ”perinteisilläkin” salausmenetelmillä, mutta XML-salaus mahdollistaa myös dokumentin osittaisen salaamisen. Salatun dokumentin käsittely on aina hitaampaa kuin salaamattoman, joten kun pystytään salaamaan vain todella salausta vaativat osiot, dokumenttien käsittelykin nopeutuu. Usein on tarvetta myös jättää osa dokumentista selkokielisiksi, jotta ns. julkinen tieto on kaikkien asianomaisten luettavissa.

Dokumentteja käsittelee monesti useampi taho, joilla voi olla erilaiset intressit salata tietoa. Tämäkään ei muodostu ongelmaksi XML-salauksen tapauksessa, sillä saman dokumentin eri osiot voivat olla eri tahojen salaamia. Näin mahdollistetaan myös se, että eri osapuolet näkevät vain ne osiot, jotka heidän kuuluukin nähdä.

Salatun XML-dokumentin säilyttäminen on myös huomioitava etu; kyseessä ei siis ole ainoastaan siirrettäviin tiedostoihin käytettävä tekniikka.

Supersalaukseksi kutsutaan menetelmää, jossa jo kertaalleen salattu data salataan uudelleen. Vaikka XML-salaus ei sallikaan EncryptedData-elementin olevan toisen EncryptedData-elementin lapsi- tai vanhempielementti, niin varsinainen salattava data voi olla mitä tahansa, esimerkiksi EncryptedData- tai EncryptedKey-elementti. Tällaisessa tapauksessa elementti on kuitenkin salattava kokonaan, sillä pelkän EncryptedData-elementin sisällön salaaminen ei ole suosituksen mukaan sallittua (Galbraith & al., 2002).

3.2.1. Syntaksi

XML-salaus esitetään EncryptedData-elementin avulla. Seuraavaksi esitellään XML-salauksen perusmuoto. Kuvan 6 esimerkissä ds-etuliitteellä viitataan XML-allekirjoituksen nimiavaruuteen. Lisäksi esimerkissä ? tarkoittaa nolla tai yksi ilmentymää, + yksi tai useampi ilmentymää ja * nolla tai useampi ilmentymää. Taulukossa 3 on esitelty XML-salauksen yleisimmin käytetyt osat (Imamura & al., 2002).

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

Kuva 6. XML-salaus -rakenne (Imamura & al., 2002)

Taulukko 3. XML-salauksen osat (Imamura & al., 2002)

Elementti	**Kuvaus**
EncryptedData	Salauksen juurielementti, joka identifioi salatun osion. Elementti korvaa salattavan datan dokumentissa. Valinnaista Id-attribuuttia voidaan käyttää yksilöimään salattu osio. Valinnaista Type-attribuuttia voidaan käyttää kertomaan onko salattu data elementti vai elementin sisältö. Valinnaista MimeType-attribuuttia voidaan käyttää kertomaan, minkä tyyppistä dataa on salattu.
EncryptionMethod	Kertoo datan salauksessa käytetyn algoritmin Algorithm-attribuutissaan. Elementti on valinnainen, joten sen puuttuessa salausalgoritmi täytyy olla selvillä muuta kautta tai salauksen purkaminen epäonnistuu.
ds:KeyInfo	Osoittaa salauksen avaamiseen käytettävän avaimen ja sen tiedot. XML-allekirjoituksesta periytyvä elementti.
EncryptedKey	Sisältää salauksessa käytetyn avaimen salatussa muodossa. Valinnaista ReferenceList-attribuuttia voidaan käyttää osoittamaan salatun avaimen salaamiseen käytettyihin dataan ja avaimeen. Valinnaista CarriedKeyName-attribuuttia voidaan käyttää kertomaan käyttäjälle avaimen selkokielisen nimen ja arvon yhteys. Valinnaista Recipient-attribuuttia käytetään kertomaan salatun datan käyttötarkoitus. Valinnaista Type-attribuuttia voidaan käyttää antamaan lisätietoa avaimen tyypistä.

AgreementMethod	Ilmoittaa salauksessa käytettyjen avainten ja menetelmien hankkimistavan, mikäli käytössä on ns. jaetun salaisuuden menetelmä.
CipherData	Sisältää salatun datan joko CipherValue-elementissään tai vaihtoehtoisesti URI-viittauksena CipherReference-elementissään.
CipherValue	Sisältää salatun datan.
CipherReference	Osoittaa URI-referenssin avulla salatun datan sijainnin.
EncryptionProperties	Sisältää mahdollisia lisätietoja.

3.2.2. Käsittelysäännöt

XML Encryption -spesifikaatio (Imamura & al., 2002) kuvaa joukon salauksen ja salauksen purkamisen aikana tehtäviä toimenpiteitä.

Salaaminen tapahtuu seuraavassa kuvatulla prosessilla:

1. Valitse salauksessa käytettävä algoritmi.
2. Hanki ja tarvittaessa esittele käytettävä avain.
    a. Jos avaimen tunnistamiseen tarvitsee antaa lisätietoja, luo sopiva ds:KeyInfo-elementti.
    b. Jos itse avain salataan, luo EncryptedKey-elementti rekursiivisesti käyttäen tätä samaa prosessia.
3. Salaa haluttu data.
    a. Jos salattava data on XML-elementti tai sen sisältö, hanki oktetit (8-bittiset tavut) serialisoimalla data UTF-8 -muotoon.
    b. Jos data ei ole valmiiksi oktetteina, se täytyy serialisoida.
    c. Salaa oktetit käyttämällä aiemmin valittuja algoritmia ja avainta
    d. Ilmoita tarvittaessa salatun datan tyyppi purkamista varten
4. Luo EncryptedData- tai EncryptedKey-elementti ja sen tarvittava rakenne.
    a. Jos salattavat oktetit on tarkoitus tallentaa CipherData-elementtiin, ne täytyy muuntaa base64-muotoon ja lisätä CipherValue-elementin sisällöksi.
    b. Jos salattavat oktetit tallennetaan tämän rakenteen ulkopuolelle, ilmoita datan URI ja tarvittaessa tehtävät muokkaustoimenpiteet CipherReference-elementissä.

5. Käsittele EncryptedData-elementti. Jos salattu data on XML-elementti tai sen sisältö, korvataan alkuperäinen data EncryptedData-elementillä; muussa tapauksessa tehdään EncryptedData-elementistä uuden XML-dokumentin juurielementti

Salauksen purkaminen tapahtuu seuraavaksi kuvatulla tavalla:

1. Hae käsiteltävästä EncryptedData- tai EncryptedKey-elementistä algoritmi, parametrit sekä mahdollinen tieto ds:KeyInfo-elementistä.
2. Hanki ds:KeyInfo-elementin perusteella käytetty salausavain. Jos avain on salattu, suorita salauksen purkaminen rekursiivisesti. Avain voi myös olla paikallisesta varastosta hankittavissa.
3. Jos CipherValue-elementti on olemassa, pura CipherData-elementin salattu datasisältö.
    a. Haetaan CipherValue-elementin sisältämä arvo ja muutetaan se base64-muodosta oktettidataksi.
    b. Haetaan salattu oktettidata URI-viittauksen perusteella ja suoritetaan mahdollisesti tarvittavat muokkaukset.
    c. Pura oktettidata käyttämällä aiemmin hankittuja algoritmeja ja avaimia.
4. Käsittele salattu data
    a. Jos kyseessä on XML-elementti tai elementin sisältö, EncryptedData-elementti korvataan UTF-8-muodossa olevalla puretulla rakenteella.
    b. Jos kyseessä on EncryptedKey-elementti, palautetaan puretut tiedot sovellukselle salatun datan purkamisen jatkamista varten

3.2.3. Salausalgoritmit

XML Encryption -spesifikaatio (Imamura & al., 2002) sisältää luettelon salausalgoritmeista, joita suosituksen mukaisen sovelluksen tulisi tukea. Algoritmit on jaettu käyttötarkoituksensa mukaisiin kategorioihin. Kategoriat ovat lohkosalaus, avaimen kuljetus (key transport), avaimesta sopiminen, symmetrisen avaimen kääre (symmetric key wrap), sanoman tiiviste, autentikointi, kanonisointi sekä koodaus.

Suosituksen mukaisia lohkosalausalgoritmeja ovat TRIPLEDES sekä AES. Ensin mainittu TRIPLEDES on parannettu versio Data Encryption Standard (DES) lohkosalausalgoritmista; hieman hitaampi, mutta paljon turvallisempi. Advanced Encryption Standard (AES) -algoritmia pidetään yleisesti TRIPLEDESiä tehokkaampana ja turvallisempana. Algoritmissa voidaan käyttää eripituisia avaimia ja suosituksen mukaisen sovelluksen tulisi tukea vähintään 128- ja 256-bittisiä versioita.

Avainkuljetinalgoritmeja ovat julkisen avaimen järjestelmät, jotka on tarkoitettu avainten salaamiseen. Tällaisia algoritmeja ovat RSA-v1.5 sekä RSA-OAEP.

Avainsopimusalgoritmeilla tarkoitetaan sellaisia algoritmeja, joiden avulla käyttäjät voivat sopia yhteisestä salaisesta avaimesta käyttäen omia julkisia avaimiaan. XML Encryption -spesifikaatio ei pakota kategorian toteuttamiseen, Diffie-Hellman-algoritmi kuitenkin mainitaan mahdollisuutena suosituksessa.

Symmetrisen avaimen järjestelmät ovat jaetun salaisen avaimen järjestelmiä, jotka on tarkoitettu symmetristen avainten salaamiseen ja purkamiseen. Kategoriaan kuuluvat mm. TRIPLEDES KeyWrap- sekä AES KeyWrap –algoritmit.

Tiivistealgoritmeja käytetään laskemaan tiivistearvoja erilaisesta datasta; samaa tekniikkaa käytetään myös digitaalisissa allekirjoituksissa. Suosituksessa mainitaan neljä eri algoritmia, joista SHA1 on pakollinen, SHA256 suositeltava ja SHA512 sekä RIPEMD-160 valinnaisia.

Autentikointiin ja kanononisointiin suositus ehdottaa ”omia tuotteita” eli W3C:n XML-allekirjoitus-, Kanoninen XML sekä Poissulkeva kanoninen XML -suosituksia.

Koodaukseen on vain yksi vaihtoehto, base64-koodausalgoritmi (Galbraith & al., 2002).

3.2.4. Suhde XML-allekirjoitukseen

Usein on tarpeen sekä allekirjoittaa että salata XML-dokumentti tai sen osia. Toimenpiteet voidaan suorittaa eri järjestyksissä ja useampaan otteeseen eli saman dokumentin osiota voidaan esimerkiksi ensin salata, sitten allekirjoittaa ja lopuksi salata koko dokumentti. Jotta allekirjoitus säilyisi ehjänä, täytyy ensin purkaa salaus vain niiltä osin, jotka on tehty allekirjoituksen jälkeen. Allekirjoittamisen jälkeen salatun osion tunnistamiseen W3C (Hughes & al., 2002) tarjoaakin avuksi Decryption Transform for XML Signature (XML-allekirjoituksen salauksen muunto) -suositusta. Kyseisen suosituksen esittelemä mekanismi on riippuvainen sekä XML Signature- että XML Encryption- spesifikaatioista, sillä se käyttää kummankin nimiavaruutta hyväkseen. Suosituksessa esiteltävän mekanismin päätarkoituksena on siis ratkaista ongelma kertomalla sovellukselle missä järjestyksessä salaus-allekirjoitus-yhdistelmiä on suoritettu.

Suosituksessa esitellään uusi dcrpt:Except –elementti, jolla on kaksi attribuuttia: valinnainen Id ja pakollinen URI. Id-parametri yksilöi elementin ja URI-parametrillä viitataan niihin EncryptedData-elementteihin, jotka on salattu ennen allekirjoitusta.

3.2.5. XML-salauksen turvallisuus

XML-allekirjoituksen ja XML-salauksen käyttö samaan dokumenttiin aiheuttaa potentiaalisia turvallisuusongelmia. Ensimmäinen jo XML-allekirjoituksen yhteydessä osittain ilmitullut ”tiedä mitä allekirjoitat”. Eli jos data on salattua, sitä ei välttämättä kannata allekirjoittaa, ellei voi olla varma sen sisällöstä. Toinen mahdollinen ongelma seuraa siitä, että kun XML-allekirjoitettua dataa salataan, tiiviste jää näkyviin Reference-elementin sisälle selkokielisenä. Tämä voi altistaa tiedon ns. pelkkätekstin arvaushyökkäyksille (Imamura & al., 2002).

Käytettävien salausalgoritmien sekä avaimien valinnassa kannattaa olla suunnitelmallinen. Esimerkiksi symmetristä avainta käytettäessä on tärkeää huomioida vastaanottajien määrä ja avaimen saatavuus. Symmetrisellä avaimella ei ole suositeltavaa salata kuin sellaista dataa, jonka tiedetään olevan sallittua kaikille symmetrisen avaimen haltijoille. Toinen algoritmeihin liittyvä ongelma on se, että osa algoritmeista tuottaa samaa avainta ja samaa dataa käytettäessä samanlaisen salauksen. Pitkällä aikavälillä hyökkääjä voi

kerätä julkisen avaimen ja salatun datan muodostamia pareja ja näiden kautta pyrkiä selvittämään salaisen avaimen. Satunnaistetut algoritmit ratkaisevat tämän ongelman kuitenkin varsin kattavasti (Galbraith & al., 2002).

XML-salaus sisältää myös itsessään muutaman huomionarvoisen turvallisuusongelman. Suositus sallii rekursiivisen käsittelyn, jonka avulla hyökkääjä voi aiheuttaa ikisilmukan ja toteuttaa näin palvelunestohyökkäyksen. XML-salauksessa ei myöskään huomioida EncryptedData- eikä EncryptedKey-elementtien eheyttä, joten hyökkääjä voi kyetessään muuttaa tai jopa tuhota em. rakenteita siten, ettei hyökkäystä välttämättä edes huomata. Hyökkäyksen voi tosin estää esimerkiksi käyttämällä XML-allekirjoitusta. Kolmantena seikkana esiin on otettu salauksen sisään piilotettavat tiedostot, jotka näin pääsevät palomuurien ohi huomaamatta. Riskitiedostoja voivat olla esimerkiksi virukset (Galbraith.& al., 2002).

3.3. XML-avaintenhallinta

Sekä XML-allekirjoituksen ja XML-salauksen käyttämisessä on oleellisena osana PKI eli julkisen avaimen järjestelmä. Kumpikaan suositus ei kuitenkaan ota kantaa avaintenhallintaan, kuten esimerkiksi avainten hankkimiseen tai avainparien luomiseen. Helpottaakseen julkisten avainten sekä digitaalisten sertifikaattien hallintaa Microsoft, VeriSign ja WebMethods alkoivat yhteistyössä kehittää avointa XKMS-spesifikaatiota (XML Key Management Specification) (Ford & al., 2001), joka julkaistiinkin 30.3.2001. Myöhemmin vastuu suosituksen kehittämisestä siirtyi W3C:lle ja suosituksen toinen versio julkaistiin 28.6.2005 (Hallam-Baker & Mysore, 2005).

XML Key Management -spesifikaatio (Hallam-Baker & Mysore, 2005) määrittää XML-allekirjoituksen ja XML-salauksen kanssa yhteensopivat protokollat avainten levittämiselle sekä rekisteröinnille. Suosituksen päämääränä on mahdollistaa luotettavien XML-pohjaisten web-palveluiden kehittäminen julkisten salausavainten käsittelyyn ja hallintaan. Tällaisten web-palveluiden luominen helpottaa julkisten avainten käyttämistä web-palveluissa ulkoistamalla julkisten avainten käsittelyn toiselle web-palvelulle. Näin itse sovelluksessa voidaan keskittyä bisneslogiikkaan ja toisaalta sovelluksista saadaan kevyempiä ja yksinkertaisempia.

XML Key Management -spesifikaatio jakaantuu kahteen osaan; avaintiedon käsittelyyn keskittyvään X-KISS:iin (XML Key Information Service Specification) sekä avainten ja niiden käyttötarkoitukseen rekisteröintiin keskittyvään X-KRSS:iin (XML Key Registration Service Specification).

XKMS on suunniteltu toteutettavaksi standardien XML-työkalujen avulla. Viestien muoto on XML:ää ja suositus sallii SOAP:in käytön XKMS-asiakkaan (client) ja XKMS-palvelun (service) välillä. XKMS-palvelut voidaan myös kuvata WSDL:llä (Christensen & al., 2001). Suositus ei kuitenkaan sido toteutusta näihin tekniikoihin, vaan sallii vapaan toteutuksen.

Kuten XML allekirjoitus -suosituksessa mainitaan, allekirjoittajan on mahdollista sisällyttää allekirjoituksensa KeyInfo-elementtiin tietoa käyttämästään julkisesta avaimesta. Näiden allekirjoittajan antamien ”vihjeiden” perusteella varmistajan tulisi pystyä päättelemään oikea julkinen avain. KeyInfo-elementti voidaan kiinnittää allekirjoitukseen kryptografisesti tai jättää kiinnittämättä. Jälkimmäisessä tapauksessa elementtiä voidaan muuttaa särkemättä allekirjoitusta itseään.

XKMS perustuu palvelusta riippumatta asiakkaan ja palvelun välisiin pyyntö-vastausviestipareihin (ks. kuvat 7. ja 8.), jotka ovat yleensä SOAP-viestejä. Palveluita voivat olla esimerkiksi avaintietojen hakeminen tai uuden avaimen rekisteröinti. Viestien käsittelyä koskien XKMS tukee kahta erilaista mallia: synkronista sekä asynkronista käsittelyä. Ensin mainitussa mallissa palvelu palauttaa yhden kattavan vastauksen; jälkimmäisessä palvelu ei täytä pyyntöä kokonaan, vaan palauttaa osittaisen vastauksen, kertoo viestissä käsittelyn jatkuvan edelleen ja palauttaa myöhemmin täydentävän vastauksen.

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I045c66f6c525a9bf3842ecd3466cd422"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyValue</RespondWith>
  <QueryKeyBinding>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
        MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
        OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
        ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
        zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
        9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
        ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
        gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
        oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
        jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
        Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
        QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
        Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
  </QueryKeyBinding>
</LocateRequest>
```

Kuva 7. Pyyntöesimerkki (Hallam-Baker & Mysore, 2005)

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I04cd4f17d0656413d744f55488369264"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I045c66f6c525a9bf3842ecd3466cd422"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <UnverifiedKeyBinding Id="I012f61e1d7b7b9944fe8d954bcb2d946">
    <ds:KeyInfo>
      <ds:KeyValue>
       <ds:RSAKeyValue>
        <ds:Modulus>
        0nIsmR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8
        HBupui8LgGthO6U9D0CNT5mbmhIAErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZ
        drdcDrkNnG7SzbuJx+GDNiHKVDQggPBLc1XagW20RMvok=
        </ds:Modulus>
        <ds:Exponent>AQAB</ds:Exponent>
       </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633" Identifier="a@a.com" />
  </UnverifiedKeyBinding>
</LocateResult>
```

Kuva 8. Vastausesimerkki (Hallam-Baker & Mysore, 2005)

3.3.1. X-KISS

XKMS:n X-KISS-osio käsittelee esimerkiksi XML-allekirjoitukseen tai XML-salaukseen liittyvän julkisen salausavaimen informaation käsittelyä. X-KISS-protokolla tarjoaa suoran käsittelytuen KeyInfo-elementille ja sen lapsille. X-KISS-palvelun asiakas voi antaa vaikka kaikki KeyInfo-elementtiä koskevat tehtävät palvelun huoleksi. X-KISS sisältää kaksi merkittävää toimintoa: paikantaminen ja kelpuutus (Galbraith & al., 2002).

Paikantamiseen tarkoitettua Locate-palvelua käytetään XKMS-palveluun rekisteröidyn julkisen avaimen paikantamiseen ja noutamiseen. Locate-palvelu siis selvittää KeyInfo-elementin ja tarjoaa sitten asiakkaalle haettavat tiedot. Tiedot voivat olla esimerkiksi avaimen arvo, avaimen nimi tai vaikka X.509-sertifikaatti. KeyInfo-elementin sisältöä ei kuitenkaan ole käytettävän julkisen avaimen suhteen rajattu, mutta yksinkertaistetut sisältötyypit eivät välttämättä tue kaikkia mahdollisia PKI-järjestelmiä. Tämä lisäisikin toteutettavan järjestelmän monimutkaisuutta, joten pelkästään tämän toiminnon ulkoistaminen yksinkertaistaa sovellusta. Locate-palvelu siis selvittää avaimen, joko käyttämällä paikallista dataa tai välittämällä pyynnön eteenpäin toiselle palvelimelle. Palvelu ei kuitenkaan tarkista KeyInfo-elementin sisältämien tietojen sidosten (eng. bindings) kelpoisuutta vaan palauttaa tiedot, joiden se uskoo olevan luotettavia.

Pelkkä avaimen selvittäminen ei useinkaan riitä luottamuksen syntymiseen, usein se täytyy myös kelpuuttaa. Validate-palvelu tarjoaa kaikki samat toiminnot kuin Locate-palvelukin, mutta näiden lisäksi se myös suorittaa avaimen kelpuutuksen. Asiakas voi saada palvelulta väitteen, jolla varmistetaan sidos julkisen avaimen ja esimerkiksi sen omistajan nimen välillä. Lisäksi Validate-palvelu vakuuttaa, että kaikki palautettavat tiedot pitävät paikkansa ja ne kaikki ovat sidoksissa toisiinsa. Eli jos jotain KeyInfo-elementin osaa ei saada kytkettyä avaimeen, sitä ei Validate-palvelun kautta enää palauteta asiakkaalle.

3.3.2. X-KRSS

XKMS:n X-KRSS-osio kuvaa joukon palveluita, jotka tarjoavat tukea julkisen avaimen rekisteröintiin ja myöhempään hallintaan liittyen. X-KRSS-määrityksen päätarkoitus on tarjota ratkaisumalli XML-pohjaiseen julkisten avaimien hallitsemiseen. Aiemmin kehi-

tetyt sertifikaattien yms. hallintaan tarkoitetut järjestelmät on koettu liian monimutkaisiksi käytettäviksi varsin kevyissä XML-pohjaisissa sovelluksissa. X-KRSS-määritys sisältää sertifikaatin koko elämänkaaren kattavat neljä toimintoa. Toiminnot ovat rekisteröinti (Register), uudelleenjulkaisu (Reissue), peruuttaminen (Revoke) sekä palauttaminen (Recover). Toteutettava palvelu voi sisältää kaikki toiminnot, toisaalta sen ei välttämättä tarvitse tarjota palveluista ensimmäistäkään (Galbraith & al., 2002).

Rekisteröintitoiminnolla sidotaan haluttu informaatio tiettyyn julkiseen avaimeen. Tarvittaessa myös avainparin luonti voidaan suorittaa palvelun toimesta, mutta yleisempää on, että avaimet tulevat asiakkaalta ja salainen avain pysyy palvelun tietämättömissä.

Uudelleenjulkaisutoiminnolla avainsidos otetaan uudelleen käyttöön. Toiminta on samanlaista kuin uuden avainparin rekisteröinnissäkin, mutta tässä tapauksessa uusitaan vaan aiemmin rekisteröidyn avaimen tiedot. Pääsyy toiminnon käyttämiseen on sertifikaattitietojen päivitys, esimerkiksi voimassaoloajan jatkaminen.

Peruutustoiminnolla aiemmin rekisteröity avain peruutetaan eli poistetaan käytöstä. Peruutuspyynnön tulee sisältää tarpeeksi informaatiota avaimen tunnistamista varten. Jos käytöstä poistettavaan avaimeen on liitetty esimerkiksi sertifikaatti, sekin tulee perua avaimen perumisen yhteydessä. Yleisin syy avaimen peruuttamiseen on salaisen avaimen paljastuminen.

Palautustoiminnolla palautetaan avaimeen sidottu yksityinen avain. Jos avainpari on luotu palvelun toimesta tai asiakas on antanut salaisen avaimensa palvelun haltuun, niin palvelu voi asiakkaan pyynnöstä toimittaa salaisen avaimen asiakkaalle. Tällaisessa tapauksessa on suositeltavaa käyttää jotain ennalta sovittua koodia, joka varmistaa asiakkaan olevan se, keneksi se itseään väittää.

3.3.3. XML-avaintenhallinnan turvallisuus

XML-avaintenhallinta tuo mukanaan muutamia tekniikkariippuvaisia ongelmia. Identiteettivarkaudet ovat ehkä näistä yleisimpiä. Varas rekisteröi hankkimansa julkisen avaimen palveluun ilman että hänen hallussaan olisi avaimen salainen pari. Tämän jälkeen

allekirjoituksen todennus onnistuu edelleenkin, mutta salatut dokumentit jäävät aukeamatta. Ongelmalta vältytään käyttämällä ns. hallussapidon todistamiskäytäntöä (POP, Proof-of-Possession) rekisteröinnin yhteydessä, jolloin rekisteröijä joutuu todistamaan pitävänsä hallussaan myös julkista avainta vastaavaa salaista avainta kuitenkin paljastamatta itse avainta. Yleisin tapa todistamiseen on salata salaisella avaimella jotain, jonka palvelu voi avata käyttämällä julkista avainta (Hirsch & Just, 2003).

Toinen identiteettivarkaustapa on luoda uusi avainpari ja rekisteröidä se palveluun jonkun toisen nimissä. Tällaisessa tapauksessa varkaalla on hallussaan myös salainen avain, mutta kyseessä on väärennös. Ongelman ratkaisuksi käytetään yleensä ns. jaettua salaisuutta rekisteröinnin yhteydessä, eli käytetään esimerkiksi puhelimitse sovittua koodia avainten rekisteröintiin (Ford & al., 2001).

X-KRSS -palvelua voidaan siis käyttää avainparin luomiseen ja tästä seuraa salaisen avaimen jääminen myös palvelun haltuun. Palvelua voidaan yleensä pitää luotettavana tahona, mutta luonnollisesti avaimen säilyttäminen useammassa paikassa lisää riskejä. Jos avainta käytetään pääasiassa allekirjoitukseen, on suositeltavaa että asiakas luo itse avainparinsa. Näin salaisen avaimen tuhoutuessakin sillä allekirjoitetut dokumentit ovat todennettavissa. Jos avainta käytetään pääasiassa salaamiseen, avaimen turvaaminen voi olla perusteltua, sillä salattuja dokumentteja ei muuten saa avattua salaisen avaimen kadottua tai tuhouduttua. Tällaisessa tapauksessa on kuitenkin suositeltavaa, että kun salainen avain palautetaan palvelusta asiakkaan käyttöön, se myös samalla perutaan. Näin aiemmin salatut dokumentit saadaan avattua, mutta mahdollisen avaimen paljastumisen takia uusia salauksia ei enää kyseisellä avainparilla saa tehtyä (Ford & al., 2001).

Lisäksi XKMS sisältää muutamia muita riskikohtia, joista mainittakoon esimerkkinä palvelunestohyökkäykset lähettämällä suuren määrän yhtäaikaisia pyyntöjä, jotka saattavat jumittaa palvelun (Hallam-Baker & Mysore, 2005).

3.4. WS-Security

OASIS (Organization for the Advancement of Structured Information Standards) on vuonna 1993 perustettu voittoa tavoittelematon organisaatio, joka tekee avoimia stan-

dardeja ja edistää niiden käyttöä. Sen työhön osallistuu yli 5000 henkilöä yli 600 organisaatiosta. OASIS tekee läheistä yhteistyötä virallisten standardointi-organisaatioiden kanssa. Organisaation perustajayrityksinä olivat mm. IBM, Microsoft ja VeriSign (OASIS, 2006)

OASIS-organisaation standardit, samoin kuin W3C:n suositukset, tarjoavat ohjeistusta ja periaatteita asioiden toteutukseen, mutta eivät siis valmiita toteutuksia. Tästä syystä ne ovat varsin joustavia, mutta näin ollen valitettavan usein myös tehottomia (Damiani & al., 2002).

OASIS julkaisi vuonna 2004 ensimmäisen version Web Service Security -standardistaan. Sen tarkoitus on tarjota web-palveluiden viestikerrokselle yleinen menetelmä tunnistamisen, viestin eheyden ja luottamuksellisuuden toteuttamiseksi. Versio 1.0 koostui SOAP Message Security, Username Token Profile, X.509 Token Profile, SAML Token Profile sekä REL Token Profile -osioista. Vuonna 2006 julkaistu versio 1.1. koostui samojen osioiden päivitetyistä versioista sekä uusista Kerberos Token Profile ja SOAP with Attachments (SWA) Profile -osioista (OASIS, 2006).

Tämän tutkielman kannalta oleellisin osa WS-Securityä on SOAP Message Security -osio, joka WS-Securityn versiossa 1.1. tunnetaan myös nimellä Core Specification. Se esittelee XML-allekirjoituksen sekä XML-salauksen, esittelyn pohjautuessa erittäin suuressa määrin kohdissa 3.1. ja 3.2. esiteltyihin W3C:n suosituksiin. Näiden lisäksi kyseisessä osiossa esitellään turvallisuusotsikko (Security Header), turvallisuusavain (Security Token) sekä turvallisuusaikaleima (Security Timestamp) -mallit. Turvallisuusotsikko voi sisältää turvallisuusavaimia ja tällaisesta rakenteesta löytyvän tiedon avulla viestin vastaanottaja voi päätellä, kuinka viestin aitous on tarkistettavissa. Rakenne voi sisältää esimerkiksi sertifikaatin, tai avaimen XML-allekirjoitukseen tai XML-salaukseen. Turvallisuusaikaleimoja voidaan käyttää kertomaan esimerkiksi milloin viesti on luotu tai milloin se vanhentuu (OASIS, 2006).

WS-Security-toteutukset tukevat nykyisin laajalti ns. turvallisuuskonfiguraatiotiedostoja (Security configuration files). Niitä käyttäessä toteuttajan ei välttämättä tarvitse toteuttaa ohjelmallisesti turvallisuustoimia, vaan riittää, että kyseisessä tiedostossa kerrotaan mitä

toimenpiteitä suoritetaan millekin XML:n osalle ja mitä tiedostoa käytetään suojauksen toteuttamiseen. Yksi esimerkki tällaisesta turvallisuuskonfiguraatiotiedostosta löytyy kuvasta 12 (Sun, 2005).

WS-Securitylle on nykyisin jo olemassa useita eri toteutuksia. Esimerkkeinä mainittakoon Apachen WSS4J, Microsoftin Web Services Enhancements (WSE) sekä Sunin XML and Web Services Security (XWS-Security). Kohdassa 4.4 kuvattu esimerkkisovellus käyttää sekä WSE:tä että XWS-Securityä.

4. Web-palvelut ja niiden suojaaminen

Luvussa kuvataan web-palveluiden yleistä toimintaa sekä niiden suojausmalleja ja turvallisuushaasteita. Lisäksi kuvataan yksinkertaisen web-palvelun toimintaa.

4.1. Yleistä

King (2003) kuvaa web-palveluita talousihmisten hyvin ymmärtävällä ja hyväksymällä sanalla: edullinen. Web-palveluiden avulla sovellusten uudelleenkäytettävyys lisääntyy, integraatiokustannukset usein pienentyvät ja myös kalliiden toiminnallisuuksien tuottamisen kustannukset saadaan pienenemään. Käytännössä olemassa olevien toimintojen käyttöä saadaan laajennettua ja sitä kautta saadaan lisätuloja.

Web-palveluita voidaan käyttää pohtimatta sen kummemmin laitteistoalustaa, ohjelmointikieliä tai verkkotopologioita (Yu & al., 2005). Avoimien standardien ja internetprotokollien käyttö tekee web-palveluista erityisen sopivia bisnesprosessien yhdistämiseen yli yritysrajojen. Tällainen sovellusten avoimuus kuitenkin tekee niistä suuria tietoturvariskejä, varsinkin jos sovellukset ovat ns. bisneskriittisiä (Kearney, 2005).

Web-palveluiden suorittamat operaatiot voivat olla mitä tahansa yksinkertaisista tietopyynnöistä aina monimutkaisten bisnesprosessien luomiseen ja toteuttamiseen asti (Wang & al., 2004). Myös web-palvelun rajapinnan takana oleva sovellus voi olla riski. Sieltä voi löytyä sovelluspaketti, sisäisesti kehitetty sovellus tai vaikka työpöytäsovellus. Näissä voi itsessään olla isojakin tietoturvariskejä, jotka sitten vain paljastuvat webpalveluita käytettäessä (Yu & al, 2005).

Kun edellä mainittuihin ominaisuuksiin lisätään vielä web-palveluiden yksi odotetuimmista eduista, suora sovellukselta sovellukselle tapahtuva kommunikointi sen sijaan, että loppukäyttäjänä olisi ihminen, niin hälytyskellojen pitäisi alkaa soida tietoturvan puolesta (Hondo & al., 2002). Asiakkaat kuitenkin lähettävät, tai ainakin heidän pitäisi lähettää, luottamuksellista dataa kuten luottokortti- tai terveystietojaan, ainoastaan turvallisia väyliä pitkin, ja tällaiseksi web-palvelutkin täytyisi saada (Komathy & al., 2003).

Aina internetiin kytkeydyttäessä on mahdollisuus joutua hyökkäyksen kohteeksi. Internetin kehityksessä on ollut kolme isompaa aaltoa: sähköposti, www ja nyt web-palvelut. Kahdella ensin mainitulla on ollut ja on edelleen omat tietoturvaongelmansa; niinpä web-palvelut seuraavat tässäkin suhteessa edeltäjiensä jalanjäljissä. Turvallisuustaso onkin ollut pitkään esteenä web-palveluiden reaalimaailman käytölle, mutta nyt tilanne on jo muuttunut (Chou & Yurov, 2004; Rowan, 2005).

Treesen (2002) mukaan pääongelmat web-palveluiden maailmassa ovat seuraavat:

1. Luottamuksellisuus: vain lähettäjä ja vastaanottajaa kykenevät lukemaan viestin.
2. Autentikointi: viestin lähettäjän ja vastaanottajan tunnistaminen.
3. Eheys: viesti saapuu vastaanottajalle koskemattomana.
4. Kiistämättömyys: lähettäjä ei voi kieltää jälkeenpäin lähettäneensä viestiä.
5. Tunnistaminen: vain oikeat ihmiset voivat lukea viestin.
6. Avaintenhallinta: avainten luonti, varastointi, käyttö ja tuhoaminen.

Edellä mainittujen ongelmien ratkaisumalleja käydään läpi seuraavaksi.

4.2. SSL/TLS web-palveluissa

Kearneyn (2005) mukaan ylivoimaisesti yleisin tietoturvaratkaisu web-palveluiden käytössä on *SSL* (Security Socket Layer), jonka uusimmasta versiosta käytetään myös TLS (Transport Layer Security) -nimeä. Kyseinen protokolla käyttää julkisen avaimen tekniikkaa toisen tai molempien päätepisteiden autentikointiin ja sopii symmetrisestä avaimesta, jota käytetään siirtopakettien salaamiseen. Pisteiden välinen liikenne tapahtuu siis salattuna, ja näin estetään kolmansien osapuolien väliintulo.

SSL tarjoaa alkuunpanevalle isäntäkoneelle (host), yleensä siis asiakkaalle (client), varmistuksen kohteena olevan isäntäkoneen (yleensä palvelin) identiteetistä. Myös asiakkaan identiteetti voidaan saattaa isäntäkoneen tietoon käyttämällä asiakaspään sertifikaatteja. Näin taataan luottamuksellisuus ja eheys asiakkaan ja palvelimen välillä siirron aikana. Kannattaa kuitenkin huomioida, että SSL-protokolla ei tarkista millään tasolla digitaalisten sertifikaattien identifioimien olioiden luotettavuutta. Se yksinkertaisesti tarkistaa, että isäntäkoneen luottama varmenneviranomainen on myöntänyt kyseisen serti-

fikaatin (Kearney, 2005).

Viesti on turvassa liikkuessaan SSL-tunnelissa, mutta tiedostoksi tai tietokantaan tallennettuna ei asia enää olekaan näin (Bhargavan & al., 2004). Myös Yu ja muut (2005) toteavat, että SSL ei välitä siitä, mitä viestille tehdään yhteyden eri päissä. Lisäksi jos välissä tarvitaan SOAP-viestin käsittelyä, täytyy SSL-yhteys katkaista. Toki yhteys voidaan saman tien avata uudelleen, mutta tällöinkin päätepisteet (end-point) ovat menettäneet yhteyden hetkeksi. Sama ongelma ilmenee myös käytettäessä palomuuria, eli koska viestit ovat SSL:n salaamia, sovellustason palomuuri ei voi tutkia viestin sisältöä ja yhteys täytyy jälleen katkaista hetkeksi tai vaihtoehtoisesti jättää viestit tutkimatta. Edellä mainituista seikoista johtuen SSL ei tarjoa yleisesti end-to-end –ratkaisua, joka turvaisi datan aina palvelupyynnöstä palomuurin, sovelluspalvelimien ja yritysverkon sisäisten sovellusten kautta arkaluoteisen datan pysyväissäilytyspaikkaan back-office-järjestelmissä (Beznosov & al., 2005). SSL tyytyy ns. point-to-point –ratkaisuun, eli turvaa datan siirron aikana pisteestä toiseen (Kearney, 2005). Jos yksikin päätepiste tai välipalvelin on epäilyttävä, siirtokerroksen turvallisuus ei tarjoa suojaa epäilyksen alaisen koneen suhteen (MacPhee & O'Neill, 2005).

Simon ja muut (2001) toteavatkin SSL:n tarjoavan selaimen ja palvelimen välille hyvän suojatun yhteyden, eli tosiasiassa SSL suojaa dataa vain silloin, kun se on epätodennäköisimmillään joutua hyökkäyksen kohteeksi. Jos olisit krakkeri, yrittäisitkö saada haltuusi esimerkiksi yhden henkilön luottokorttitiedot ip-paketteja haistelemalla siirron aikana vai murtautumalla tietokantaan, joka sisältää tuhansien henkilöiden luottokorttitiedot? Viestien luottamuksellisuuden suojaamisen lisäksi tärkeätä on autentikointi (kuka ne lähetti?), datan eheys (onko muokattu siirron aikana?) sekä kiistämättömyys (voiko lähettäjä kiistää lähettämänsä tiedon?); toisin sanoen juuri ne ominaisuudet, jotka SSL jättää huomiotta.

Yhteenvetona SSL on hyvä teknologia suojaamaan kahden koneen välistä liikennettä, jopa yksinkertaisille web-palveluille, mutta se ei kuitenkaan tarjoa riittävää suojaa useimmille web-palveluille (Yu & al., 2005).

4.3. XML-allekirjoitus ja XML-salaus web-palveluissa

Web-palveluiden tärkeää ja arkaluonteistakin dataa sisältävät viestit siirtyvät pelkkätekstimuodossa palvelimelta toiselle. Näin ollen monimutkaisempien palveluiden, esimerkiksi jos palvelulla on useampia osapuolia tai web-palveluita on useita, on tietoturvaa haettava aina viestitasolta saakka sen sijaan, että turvauduttaisiin pelkkään turvalliseen siirtokerrokseen (Kleiner & Roscoe, 2006). Tarkoituksena on suojata viestien tai niiden osien luottamuksellisuus, eheys sekä kiistämättömyys. Käytännössä on siis tärkeää toimittaa viestin sisältö muuttumattomana halutulle vastaanottajalle ja joissakin tapauksissa myös varmistua siitä, että vaikka viesti päätyisikin väärään osoitteeseen, sitä ei pystyttäisi lukemaan. Tällaisessa tapauksessa esiin astuvat XML-allekirjoitus ja XML-salaus (Sun & Li, 2005; Kearney, 2005), joita käsiteltiin kohdissa 3.1. ja 3.2.

Monimutkaisten isoja XML-tiedostoja tuottavien ollessa kyseessä koko dokumentin salaaminen ei ole järkevää eikä käytännöllistä, vaan tällaisissa tapauksissa XML-salauksen osittainen salausmahdollisuus tulee tarpeeseen (Curphey, 2005). Viestin salaaminen vastaanottajan julkisella avaimella saa aikaan sen, ettei viestiä saa auki kukaan muu kuin salaisen avaimen haltija eli haluttu vastaanottaja. Näin varmistetaan viestin luottamuksellisuus sekä vastaanottajan identiteetti, erityisesti sertifikaatteja käytettäessä. Näillä toimenpiteillä viestin vastaanottaja saa todisteketjun viestin linkittymisestä tiettyyn henkilöön tai yritykseen. Viestin allekirjoittamisella varmistetaan viestin eheys, eli se, ettei viesti ole muuttunut matkalla (Kearney, 2005).

McIntoshin ja Austelin (2005) mukaan XML-allekirjoitusta käytettäessä kannattaa toteutustapa miettiä tarkkaan, sillä sen ajattelematon käyttö voi aiheuttaa mm. allekirjoituksen ja sen elementtien kopioinnin särkymättä toiseen dokumenttiin tai pahimmassa tapauksessa alkuperäisen allekirjoitetun dokumentin muokkaamisen huomaamatta. Tällaiset ongelmat voidaan kuitenkin välttää määräämällä allekirjoitettavien elementtien sijainti allekirjoittamisen yhteydessä. Kuitenkin kaikki allekirjoituksen tai salauksen ulkopuolella oleva tieto on käyttäjän muokattavissa (Kleiner & Roscoe, 2006).

4.3.1. Haasteet

Viegan ja Epsteinin (2006) mukaan suurin vaara XML-allekirjoitusta ja XML-salausta käytettäessä tulee palvelunestohyökkäyksien (DoS, Denial-of-Service) muodossa. Julkisen avaimen järjestelmä on tietokoneen kannalta laskennallisesti varsin työläs. Käytettävästä algoritmista, sen toteutuksesta ja käyttäjän valitseman avaimen pituudesta riippuen julkisen avaimen salaus voi olla jopa tuhansia kertoja hitaampaa kuin salaamattoman viestin käsittely. Näin ollen suuri määrä yhtäaikaisia viestejä voi tukkia palvelimen ja estää sen suunnitellun toiminnan.

Myös Juric ja muut (2006) ovat vertailleet suojaamattomia ja suojattuja Javalla toteutettuja web-palveluita niiden suorituskyvyn osalta. Havainnot olivat samansuuntaisia kuin Viegalla ja Epsteinilla (2006): WS-securityn (lähinnä XML-allekirjoitus, XML-salaus, turva-avaimet (security token) ja aikaleimat (timestamp)) avulla suojattu web-palvelu oli jopa 1000 kertaa hitaampi kuin suojaamaton palvelu. Suurimmat hitauden aiheuttajat olivat tutkimuksen mukaan toistuvat varmennukset sekä avainten hakemiset varastosta. Tutkijat epäilivät osasyyksi hitauteen sitä, että testissä oli käytössä ensimmäinen WS-securityä tukeva Java Web Service Developer Pack ja näin tutkijat arvelivat tehokkuuden paranevan huomattavasti tulevissa versioissa. Yhteenvetona tutkimuksessa todettiin, että WS-securityä tulisi käyttää vain sellaisissa web-palveluissa, joissa turvallisuudelle on annettu huomattavasti tehokkuutta korkeampi prioriteetti.

Lisää haasteita turvallisten web-palveluiden suunnittelijoille tuo avaintenhallinta. Monelle voisi tulla mieleen, että turvallisuus saavutetaan jakamalla kaikille osapuolille avaimet ja tämän jälkeen allekirjoittamalla tai salaamalla kaikki viestit kokonaisuudessaan. Tämä ei kuitenkaan ole järkevä tapa toimia, sillä menetelmä olisi todella raskas ja avaintenhallinta luottamussuhteineen käytännössä mahdotonta. Kunnollinen avaintenhallinta vaatii siis työtä, mutta se on oleellinen osa, jos aiotaan toteuttaa turvallinen järjestelmä. Monesti lähinnä suunnittelullisena ongelmana on, että web-palvelussa tarkistetaan XML-allekirjoituksen oikeellisuus, mutta jätetään tarkastamatta onko allekirjoituksen omistaja ns. luotettu henkilö (Viega & Epstein, 2006).

Myös WS-Securityn turvallisuusavaimet ja aikaleimat tuovat omat haasteensa. Turvallisuusavaimet eivät saa joutua vääriin käsiin ja aikaleimoja ei saa pystyä uudelleenkäyttämään tai väärentämään. Kumpikin ongelma on kuitenkin varsin hyvin ratkaistavissa käyttämällä XML-allekirjoitusta ja XML-salausta.

4.4. Web-palveluesimerkki

Käsiteltävä web-palveluesimerkki on otettu Microsoft Developer Network -sivustolta (MSDN, 2005). Kyseisessä web-palvelussa käytetään WS-Securityä Microsoftin WSE 2.0- sekä Sunin XWS-Security -toteutuksien muodossa. Esimerkin tarkoituksena on kuvata, kuinka asiakas lähettää luottokorttitietonsa web-palvelulle, joka sitten tiedot saatuaan palauttaa kuittauksen seurantanumeron muodossa.

Luottokortista tarvittavat tiedot on kuvattu XSD-muodossa (XML Schema Definition). Tiedoille annetaan nimi (name), tyyppi (type) sekä vähimmäis- (minOccurs) ja enimmäisesiintymiskertamärät (maxOccurs). XSD-tiedosto näkyy kuvassa 9.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="ComplexMessages" targetNamespace="http://schemas.Wsig.
samples.microsoft.com/ComplexMessages.xsd"
elementFormDefault="qualified" xmlns= "http://schemas.wsig.Samples.
microsoft.com/ComplexMessages.xsd" xmlns:mstns="http://schemas.wsig.
samples.microsoft.com/ComplexMessages.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="Order">
  <xs:sequence>
   <xs:element minOccurs="1" maxOccurs="1" name="Id" type="xs:long" />
   <xs:element minOccurs="0" maxOccurs="1" name="CreditCardNum"
type="xs:string" />
   <xs:element minOccurs="0" maxOccurs="1" name="CreditCardExpM"
type="xs:int" />
   <xs:element minOccurs="0" maxOccurs="1" name="CreditCardExpY"
type="xs:int" />
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Kuva 9. Luottokorttitiedot kuvaava XSD-tiedosto.

Esimerkin salaamaton ja allekirjoittamaton tieto näkyvät kuvissa 10 (asiakkaalta web-palvelulle) ja 11 (web-palvelun vastaus asiakkaalle).

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
<soap:Header>
<wsa:Action/>
<wsa:MessageID>uuid:e718caa2-978b-4008-921c-c9d7f4a1092f
</wsa:MessageID>
<wsa:ReplyTo>
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/
role/anonymous</wsa:Address>
</wsa:ReplyTo>
<wsa:To>http://localhost:8080/OrderService/OrderService</wsa:To>
<wsse:Security>
<wsu:Timestamp wsu:Id="Timestamp-8b963093-1f3a-46df-b3e0-6bcb27b470a5">
<wsu:Created>2008-05-24T09:19:38Z</wsu:Created>
<wsu:Expires>2008-05-24T09:24:38Z</wsu:Expires>
</wsu:Timestamp>
</wsse:Security>
</soap:Header>
<soap:Body>
<submitOrder xmlns="http://wss.samples.microsoft.com">
<OrderImpl_1 xmlns="">
<creditCardExpM>10</creditCardExpM>
<creditCardExpY>5</creditCardExpY>
<creditCardNum>4426-1234-5678-9012</creditCardNum>
<id>348922</id>
</OrderImpl_1>
</submitOrder>
</soap:Body>
</soap:Envelope>
```

Kuva 10. Viesti asiakkaalta web-palvelulle

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="http://wss.samples.microsoft.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Body>
<ns0:submitOrderResponse>
<result>1610506393</result>
</ns0:submitOrderResponse>
</env:Body>
</env:Envelope>
```

Kuva 11. Vastausviesti web-palvelulta asiakkaalle

Tiedon allekirjoittaminen ja salaaminen voidaan nykyisin toteuttaa ohjelmallisesti tai käyttämällä ns. turvallisuuskonfiguraatiotiedostoja. Tässä esimerkissä tiedon turvaaminen toteutetaan kyseisten tiedostojen avulla. Turvallisuuskonfiguraatiotiedostot ovat

XML-muotoisia, joten niitä voi käsitellä manuaalisesti tai esimerkiksi WSE 2.0:n tapauksessa Configuration Toolia hyväksikäyttämällä. Kuvassa 12 on JWSDP:n tässä esimerkissä käyttämä turvallisuuskonfiguraatiotiedosto nimeltä wsse.xml. Jos siirrettävä tieto halutaan allekirjoittaa, otetaan käyttöön xwss:Sign-kohta; jos siirrettävä tieto halutaan salata, otetaan käyttöön xwss:Encrypt-kohta. Oheisessa esimerkkitapauksessa tieto salataan.

```
<xwss:JAXRPCSecurity
xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
<xwss:Service>
    <xwss:SecurityConfiguration dumpMessages="true">
      <!--xwss:Sign>
          <xwss:X509Token certificateAlias="s1as"/>
      </xwss:Sign-->
      <xwss:Encrypt>
          <xwss:X509Token certificateAlias="wse2client"/>
      </xwss:Encrypt>
     </xwss:SecurityConfiguration>
 </xwss:Service>
<xwss:SecurityEnvironmentHandler>
    com.sun.xml.wss.sample.SecurityEnvironmentHandler
</xwss:SecurityEnvironmentHandler>
</xwss:JAXRPCSecurity>
```

Kuva 12. JWSDP:n turvallisuuskonfiguraatiotiedosto

Liitteessä 2 on web-palvelun pääasiassa toteuttavat OrderServiceIF.java- sekä OrderService.java-tiedostot. Salaus ja allekirjoitus on siis toteutettu turvallisuuskonfiguraatiotiedostoilla, joten kooditasolla ei ole mitään tavallisuudesta poikkeavaa. Samoin liitteestä 3 löytyvä asiakaspään C#-koodi on perusmallia noudattelevaa.

Liitteessä 4 on varsinainen web-palvelun turvallisen toiminnan ohjaava Security-EnvironmentHandler.java-tiedosto, joka varsinaisesti käsittelee allekirjoituksen ja salauksen. Käytettävän tiedoston nimi on annettu turvallisuuskonfiguraatiotiedoston kohdassa xwss:SecurityEnvironmentHandler.

Liitteessä 5 on esimerkki asiakkaan lähettämästä salatusta pyynnöstä. Viestin koko body-osa on salattu. Liitteessä 6 on allekirjoitettu web-palvelun lähettämä vastausviesti. BinarySecurity-Tokenista löytyy julkinen X509-sertifikaatti ja SignatureValue-kohdasta löytyy viestistä laskettu tiivistearvo.

5. Yhteenveto

Web-palveluiden suosio on kasvanut vuosi vuodelta ja näin ollen niiden turvallisuuteenkin on alettu kiinnittää entistä enemmän huomiota. Web-palvelut käyttävät kommunikaatioon yleensä XML-viestejä, jotka ovat ns. pelkkätekstimuotoisia. Näin ollen vääriin käsiin päätyessään viestit ovat helposti luettavissa.

Tällä hetkellä yleisimmin käytetty suojaamismalli on SSL:n käyttö, mutta kyseisessä mallissa ongelmaksi muodostuvat turvattomat välipalvelimet sekä viestien suojaaminen päätepisteissä ns. varastointivaiheessa. Ongelmalta vältyttäisiin käyttämällä esimerkiksi XML-allekirjoitettua ja XML-salattua dokumenttia, joka voidaan tallentaa tietokantaan tai levylle tietoturvan vaarantumatta.

Jos web-palvelun sisältö on arkaluonteista tai muuten tärkeää, kannattaa ottaa käyttöön XML-allekirjoitus ja XML-salaus sekä mahdollisesti XML-avaintenhallinta, joilla saadaan turvattua palvelun viesteille luottamuksellisuus, varmennus, eheys, kiistämättömyys sekä avaintenhallinta. XML-allekirjoitus ja XML-salaus mahdollistavat viestien osittaisen allekirjoittamisen ja salauksen, joka osaltaan keventää viestien normaalikäsittelyä. Kun tähän vielä lisätään WS-Securityn myötä tuleva turvallisuusaikaleimojen käyttö, ei viestiä mahdollisesti väärinkäyttävällä taholle jää paljoa aikaa murtaa viestin salausta.

Suurimpana ongelmana edellä mainittujen suositusten käyttämisessä on niiden laskentatehollinen raskaus eli tietokone joutuu tekemään paljon toimintoja viestien avaamisessa. Tätä kautta suurimmaksi riskiksi muodostuu DoS- eli palvelunestohyökkäykset, sillä jo kohtalaisen pienellä määrällä viestejä huonosti konfiguroitu palvelin saadaan vastaamattomaan tilaan.

XML-allekirjoitusta ja XML-salausta suositellaan käytettäväksi varsinkin sellaisissa web-palveluissa, joissa viestien turvallisuus sekä siirron aikana että varastoituna, on web-palvelun tehokkuutta tärkeämpää. Ja kuten kaikessa sovelluskehityksessä, panostusta testaamiseen ei kannata koskaan väheksyä.

Viiteluettelo

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E. (2002) *XML-Signature Syntax and Processing. W3C Recommendation 12 February 2002*. WWW-sivusto, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (21.5.2008).

Berglund, A. (2006) *Extensible Stylesheet Language (XSL) Version 1.1. W3C Recommendation 05 December 2006*. WWW-sivusto, http://www.w3.org/TR/2006/REC-xsl11-20061205/ (21.5.2008).

Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J. (2007) *XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007*. WWW-sivusto, http://www.w3.org/TR/2007/REC-xpath20-20070123/ (21.5.2008).

Beznosov, K., Flinn, D.J., Kawamoto, S., Hartman, B. (2005) Introduction to Web services and their security. *Information Security Technical report 10*, 2-14.

Bhargavan, K., Fournet, C., Gordon, A.D. (2004) A Semantics for Web Services Authentication. *POPL'04*. ACM, Venice.

Boyer, J. (2001) *Canonical XML Version 1.0. W3C Recommendation 15 March 2001*. WWW-sivusto, http://www.w3.org/TR/2001/REC-xml-c14n-20010315/ (21.5.2008).

Boyer, J., Eastlake 3rd, d.E., Reagle, J. (2002) *Exclusive XML Canonicalization. Version 1.0. W3C Recommendation 18 July 2002*. WWW-sivusto, http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/ (21.5.2008).

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., Cowan, J (2006) *Extensible Markup Language (XML) 1.1 (Second Edition) W3C Recommendation 16 August 2006, edited in place 29 September 2006*. WWW-sivusto, http://www.w3.org/TR/2006/REC-xml11-20060816 (21.5.2008).

Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S. (2007) *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language W3C Recommendation 26 June 2007.* WWW-sivusto, http://www.w3.org/TR/2007/REC-wsdl20-20070626/ (23.5.2008).

Chou, D.C., Yurov, K. (2004) Security Development in Web Services environment. *Computer Standards & Interfaces 27* (2004), 233-240.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001) *Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001*. WWW-sivusto, http://www.w3.org/TR/wsdl (21.5.2008).

Clark, J. (1999) *XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November 1999*. WWW-sivusto, http://www.w3.org/TR/1999/REC-xslt-19991116 (21.5.2008).

Curphey, M. (2005) Web services: Developers dream or hackers heaven? *Information Security Technical Report* (2005) 10, 228-235.

Damiani, E., De Capitani di Vimercati, S., Paraboschi S., Samarati, P. (2002). Securing SOAP e-services. *International Journal of Information Security Volume 1 Number 2* (2002), 100-115.

Dournaee, B. (2002) *XML Security*. McGraw-Hill, New York.

FINLEX (2003) *Laki sähköisistä allekirjoituksista 24.1.2003/14.* http://www.finlex.fi/fi/laki/ajantasa/2003/20030014 (21.5.2008).

Ford, W., Hallam-Baker, P., Fox, B., Dillaway, B., LaMacchia, B., Epstein, J., Lapp, J. (2001) *XML Key Management Specification (XKMS) W3C Note 30 March 2001*. WWW-sivusto, http://www.w3.org/TR/2001/NOTE-xkms-20010330/ (21.5.2008).

Galbraith, B., Hankison, W., Hiotis, A., Janakiraman, M., Prasad, D.V., Trivedi, R.,

Whitney, D. (2002) *Professional Web Services Security*. WROX Press, Birmingham.

Geuer-Pollmann, C., Claessens, J. (2005) Web services and web service security standards. *Information Security Technical Report* (2005) 10, 15-24.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., Nielsen, H.F. (2007) *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation 27 April 2007*. WWW-sivusto, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (21.5.2008).

Hallam-Baker, P., Mysore, S.H. (2005) *XML Key Management Specification (XKMS 2.0). Version 2.0. W3C Recommendation 28 June 2005*. WWW-sivusto, http://www.w3.org/TR/2005/REC-xkms2-20050628/ (21.5.2008).

Hirsch, M., Just, M. (2003) *XML Key Management (XKMS 2.0) Requirements W3C Note 05 May 2003*. WWW-sivusto, http://www.w3.org/TR/2003/NOTE-xkms2-req-20030505 (21.5.2008).

Holzner, S. (2001) *Inside XML*. Edita, Helsinki.

Hondo, M., Nagaratnam, N., Nadalin, A. (2002) Securing Web Services. *IBM Systems Journal, Volume 41, Number 2*, 2002.

Hughes M., Imamura T., Maruyama H. (2002) *Decryption Transform for XML Signature. W3C Recommendation 10 December 2002*. WWW-sivusto, http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210 (21.5.2008).

Imamura, T., Dillaway, B., Simon, E. (2002) *XML Encryption Syntax and Processing. W3C Recommendation 10 December 2002*. WWW-sivusto, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/ (21.5.2008).

Juric, M.B., Rozman, I., Brumen, B., Colnaric, M., Hericko, M. (2005) Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL. *The Journal of Sys-*

tems and Software 79 (2006) 689-700.

Järvinen, P. (2003) *Salausmenetelmät*. Docendo, Jyväskylä.

Kearney, P. (2005) Message level security for web services. *Information Security Technical report 10*, 41-50.

Kerttula, E. (2000) *Tietoverkkojen tietoturva*. 3. painos. Edita, Helsinki. Liikenneministeriö.

King, S. (2003) Threats and Solutions to Web Services Security. *Network Security, September 2003*, 8-11.

Kleiner, E., Roscoe, A.W. (2006) On the Relationship Between Web Services Security and Traditional Protocols. *Electronic Notes in Theoretical Computer Science 155* (2006) 583-603.

Komathy, K., Ramachandran, V., Vivekanandan, P. (2003) Security for XML messaging services – a component-based approach. *Journal of Network and Computer Applications 26* (2003) 197-211.

Krutz, R.L., Vines, D.V., suom. Suominen, E (2003) *Tietoturvasertifikaatti CISSP*. IT Press, Helsinki.

MacPhee, A., O'Neill, M. (2005) Notes from the field: Implementing a security solution for Web Services. *Information Security Technical report 10*, 25-32.

McIntosh, M., Austel, P. (2005) XML Signature Element Wrapping Attacks and Countermeasures. *Proceedings of the 2005 workshop on Secure web services (SWS'05).*

McLaughlin, B. (2002) *Java & XML – tehokäyttäjän opas*. Gummerus, Jyväskylä.

MSDN (2005) *WS-Security Interoperability Using WSE 2.0 and Sun JWSDP 1.5.*

WWW-sivusto, http://msdn.microsoft.com/en-us/library/ms998284.aspx (24.5.2008).

North, S., Hermans, P. (2000) *XML*. Edita, Helsinki.

OASIS Open (2004) *UDDI Executive Overview: Enabling Service-Oriented Architecture*. Www-sivusto, http://uddi.org/pubs/uddi-exec-wp.pdf (21.5.2008).

OASIS (2006) OASIS Web Services Security (WSS) TC. WWW-sivusto, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss (21.5.2008).

Pajukoski, M. (2004) *Sähköinen asiointi sosiaali- ja terveydenhuollossa*. STAKES. Raportteja 283. Helsinki.

Perttula, J. (2002) *Tietoverkkojen oikeudelliset kysymykset Euroopan unionin jäsenvaltioiden viranomaisten välisessä sähköisessä viestinnässä*. KELA. Sosiaali- ja terveysturvan tutkimuksia 64. Helsinki.

Rinne, T. (2002) *Älykortit – tekniikka, sovellusalueet ja käyttöönotto*. Gummerus, Jyväskylä.

Rowan, L. (2005) Security in a Web services world. *Network Security June 2005*, 7-10.

Ruohonen, M. (2002) *Tietoturva*. WS Bookwell, Porvoo.

Simon, E., Madsen, P., Adams, C. (2001) *An Introduction to XML Digital Signatures*. WWW-sivusto, http://www.xml.com/pub/a/2001/08/08/xmldsig.html (24.5.2008).

Stencil Group (2002) *The Evolution of UDDI - UDDI.org White Paper*. WWW-sivusto, http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf (21.5.2008).

Sun (2005) *The JavaTrademarked Web Services Tutorial For Java Web Services Developer's Pack, v1.6.* WWW-sivusto, http://java.sun.com/webservices/docs/1.6/tutorial/doc/ (24.5.2008).

Sun, L., Li, Y. (2005) XML Undeniable signatures. Proceedings of the International Conference on Computational Intelligence for Modelling, *Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06) - Volume 01*, 981 – 985.

Treese, W. (2002) XML, web services, and XML – Putting it together. *NetWorker Volume 6, Issue 3* (September 2002), 9-12.

Viega, J., Epstein, J. (2006) Why Applying Standards to Web Services Is Not Enough. *IEEE Security & Privacy, July/August 2006*, 25-31.

Wang, H., Huang, J.Z., Qu, Y., Xie, J (2004) Web services: problems and future directions. *Web Semantics: Science, Services and Agents on the World Wide Web 1*, 309-320.

Yu, W.D., Supthaweesuk, P., Aravind, D. (2005) Trustworthy Web Services Based on Testing. *Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05).*

Liite 1: OrderService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="OrderService" targetNamespace="http://wss.samples.microsoft.com" xmlns:tns=
"http://wss.samples.microsoft.com" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema targetNamespace="http://wss.samples.microsoft.com" xmlns:tns=
"http://wss.samples.microsoft.com" xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsdl=
"http://schemas.xmlsoap.org/wsdl/" xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="submitOrder">
        <sequence>
          <element name="OrderImpl_1" type="tns:OrderImpl" nillable="true"/>
        </sequence>
      </complexType>
      <complexType name="OrderImpl">
        <sequence>
          <element name="creditCardExpM" type="int"/>
          <element name="creditCardExpY" type="int"/>
          <element name="creditCardNum" type="string" nillable="true"/>
          <element name="id" type="long"/></sequence>
      </complexType>
      <complexType name="submitOrderResponse">
        <sequence>
          <element name="result" type="int"/>
         </sequence>
      </complexType>
      <element name="submitOrder" type="tns:submitOrder"/>
      <element name="submitOrderResponse" type="tns:submitOrderResponse"/></schema></types>
  <message name="OrderServiceIF_submitOrder">
    <part name="parameters" element="tns:submitOrder"/>
  </message>
  <message name="OrderServiceIF_submitOrderResponse">
    <part name="result" element="tns:submitOrderResponse"/>
  </message>
  <portType name="OrderServiceIF">
    <operation name="submitOrder">
      <input message="tns:OrderServiceIF_submitOrder"/>
      <output message="tns:OrderServiceIF_submitOrderResponse"/>
   </operation>
  </portType>
  <binding name="OrderServiceIFBinding" type="tns:OrderServiceIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="submitOrder">
      <soap:operation soapAction=""/>
      <input><soap:body use="literal"/></input>
      <output><soap:body use="literal"/></output>
    </operation>
  </binding>
  <service name="OrderService">
    <port name="OrderServiceIFPort" binding="tns:OrderServiceIFBinding">
      <soap:address location="http://localhost:8080/OrderService/OrderService" />
    </port>
  </service>
</definitions>
```

Liite 2: OrderServiceIF.java ja OrderServiceImpl.java (web-palvelu)

```
package com.microsoft.samples.wss;

public interface OrderServiceIF extends java.rmi.Remote
{
        public int submitOrder (com.microsoft.samples.wsig.schemas.complexmessages.impl.OrderImpl
o) throws java.rmi.RemoteException;
}
```

```
package com.microsoft.samples.wss;

import java.util.Random;
import java.text.*;
import com.microsoft.samples.wsig.schemas.complexmessages.impl.*;

public class OrderServiceImpl implements OrderServiceIF
{
   public int submitOrder(OrderImpl o)
   {
     // display the details
     System.out.println("\n\nCredit Card information has been received by the Sun JWSDP 1.4 Web
Service");
     System.out.println("Order: "+o.getId());
     NumberFormat fmt = NumberFormat.getInstance();
     fmt.setMinimumIntegerDigits(2);

     System.out.println("Credit Card Information: "+o.getCreditCardNum()+" Expiry:
"+fmt.format(o.getCreditCardExpM())+"/"+fmt.format(o.getCreditCardExpY())+"\n\n");

     // generate a receipt number
     Random generator = new Random();
     return Math.abs(generator.nextInt());
   }
}
```

Liite 3: Main.cs (client)

```
using System;

using Microsoft.Web.Services2.Security;
using Microsoft.Web.Services2.Security.Tokens;
using Microsoft.Web.Services2.Security.X509;

namespace Client
{
    class Class1
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Creating order...");
            // create a reference to the service
            OrderService.OrderService os = new OrderService.OrderService();
            OrderService.submitOrder submit = new OrderService.submitOrder();
            OrderService.OrderImpl order = new OrderService.OrderImpl();

            // create the payment information for an order
            order.id = 348922;
            order.creditCardNum = "4426-1234-5678-9012";
            order.creditCardExpM = 10;
            order.creditCardExpY = 05;
            submit.OrderImpl_1 = order;

            try
            {
                Console.WriteLine("Sending payment information to Sun JWSDP 1.5 Web
Service...");
                OrderService.submitOrderResponse result = os.submitOrder(submit);
                Console.WriteLine("Submit complete.  The tracking number for this order is:
"+result.result.ToString());

                Console.WriteLine("\nPress Enter to continue.");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }

            Console.In.ReadLine();

        }
    }
}
```

Liite 4: SecurityEnvironmentHandler.java

```
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */

package com.sun.xml.wss.sample;

import java.io.IOException;
import java.io.FileInputStream;

import java.util.Enumeration;
import java.util.Properties;
import java.util.Arrays;

import java.math.BigInteger;

import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.CertPathBuilder;
import java.security.cert.Certificate;
import java.security.cert.CertificateExpiredException;
import java.security.cert.CertificateNotYetValidException;
import java.security.cert.PKIXBuilderParameters;
import java.security.cert.PKIXCertPathBuilderResult;
import java.security.cert.X509CertSelector;
import java.security.cert.X509Certificate;

import javax.crypto.SecretKey;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import com.sun.xml.wss.impl.callback.CertificateValidationCallback;
import com.sun.xml.wss.impl.callback.DecryptionKeyCallback;
import com.sun.xml.wss.impl.callback.EncryptionKeyCallback;
import com.sun.xml.wss.impl.callback.PasswordCallback;
import com.sun.xml.wss.impl.callback.PasswordValidationCallback;
import com.sun.xml.wss.impl.callback.SignatureKeyCallback;
import com.sun.xml.wss.impl.callback.SignatureVerificationKeyCallback;
import com.sun.xml.wss.impl.callback.UsernameCallback;
import com.sun.org.apache.xml.security.utils.RFC2253Parser;


/**
 * A sample implementation of a CallbackHandler.
 */
public  class SecurityEnvironmentHandler implements CallbackHandler {

    private String keyStoreURL;
    private String keyStorePassword;
    private String keyStoreType;

    private String trustStoreURL;
    private String trustStorePassword;
```

```
    private String trustStoreType;

    private String symmKeyStoreURL;
    private String symmKeyStorePassword;
    private String symmKeyStoreType;

    private KeyStore keyStore;
    private KeyStore trustStore;
    private KeyStore symmKeyStore;

    private static final String fileSeparator = System.getProperty("file.separator");

    private static final UnsupportedCallbackException unsupported =
        new UnsupportedCallbackException(null, "Unsupported Callback Type Encountered");


    public SecurityEnvironmentHandler() throws Exception {

        Properties properties = new Properties();

        String containerType = System.getProperty("jwsdp.container.type");

        String home = null;
        if (containerType != null) {

            if ("appserver".equals(containerType)) {
                home = System.getProperty("catalina.home") + fileSeparator + ".." + fileSeparator + "..";
            } else if ("webserver".equals(containerType)) {
                home = System.getProperty("catalina.home") + fileSeparator + "..";
            } else if ("tomcat".equals(containerType)) {
                home = System.getProperty("catalina.home");
            }

            String serverPropsFile = home + fileSeparator + "xws-security" + fileSeparator + "etc" +
fileSeparator + "server-security-env.properties";
            properties.load(new FileInputStream(serverPropsFile));

        } else {
            home = System.getProperty("jwsdp.home");
            // we are on the client side
            String clientPropsFile = home + fileSeparator + "xws-security" + fileSeparator + "etc" +
fileSeparator + "client-security-env.properties";
            properties.load(new FileInputStream(clientPropsFile));
        }

        this.keyStoreURL = home + properties.getProperty("keystore.url");
        this.keyStoreType = properties.getProperty("keystore.type");
        this.keyStorePassword = properties.getProperty("keystore.password");

        this.trustStoreURL = home + properties.getProperty("truststore.url");
        this.trustStoreType = properties.getProperty("truststore.type");
        this.trustStorePassword = properties.getProperty("truststore.password");

        this.symmKeyStoreURL = home + properties.getProperty("symmetrickeystore.url");
        this.symmKeyStoreType = properties.getProperty("symmetrickeystore.type");
        this.symmKeyStorePassword = properties.getProperty("symmetrickeystore.password");
    }
```

```
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {

        for (int i=0; i < callbacks.length; i++) {

            if (callbacks[i] instanceof PasswordValidationCallback) {
                PasswordValidationCallback cb = (PasswordValidationCallback) callbacks[i];
                if (cb.getRequest() instanceof PasswordValidationCallback.PlainTextPasswordRequest) {
                    cb.setValidator(new PlainTextPasswordValidator());

                } else if (cb.getRequest() instanceof PasswordValidationCallback.DigestPasswordRequest) {
                    PasswordValidationCallback.DigestPasswordRequest request =
                        (PasswordValidationCallback.DigestPasswordRequest) cb.getRequest();
                    String username = request.getUsername();
                    if ("Ron".equals(username)) {
                        request.setPassword("noR");
                        cb.setValidator(new PasswordValidationCallback.DigestPasswordValidator());
                    }
                } else {
                    throw unsupported;
                }

            } else if (callbacks[i] instanceof SignatureVerificationKeyCallback) {
                SignatureVerificationKeyCallback cb = (SignatureVerificationKeyCallback)callbacks[i];

                if (cb.getRequest() instanceof
SignatureVerificationKeyCallback.X509SubjectKeyIdentifierBasedRequest) {
                    // subject keyid request
                    SignatureVerificationKeyCallback.X509SubjectKeyIdentifierBasedRequest request =
                        (SignatureVerificationKeyCallback.X509SubjectKeyIdentifierBasedRequest)
cb.getRequest();
                    if (trustStore == null)
                        initTrustStore();
                    X509Certificate cert =
                        getCertificateFromTrustStore(
                            request.getSubjectKeyIdentifier());
                    request.setX509Certificate(cert);

                } else if (cb.getRequest() instanceof
SignatureVerificationKeyCallback.X509IssuerSerialBasedRequest) {
                    // issuer serial request
                    SignatureVerificationKeyCallback.X509IssuerSerialBasedRequest request =
                        (SignatureVerificationKeyCallback.X509IssuerSerialBasedRequest) cb.getRequest();
                    if (trustStore == null)
                        initTrustStore();
                    X509Certificate cert =
                        getCertificateFromTrustStore(
                            request.getIssuerName(),
                            request.getSerialNumber());
                    request.setX509Certificate(cert);

                } else  {
                    throw unsupported;
                }

            } else if (callbacks[i] instanceof SignatureKeyCallback) {
                SignatureKeyCallback cb = (SignatureKeyCallback)callbacks[i];

                if (cb.getRequest() instanceof SignatureKeyCallback.DefaultPrivKeyCertRequest) {
```

```
            // default priv key cert req
            SignatureKeyCallback.DefaultPrivKeyCertRequest request =
               (SignatureKeyCallback.DefaultPrivKeyCertRequest) cb.getRequest();
            if (keyStore == null)
               initKeyStore();
            getDefaultPrivKeyCert(request);

         } else if (cb.getRequest() instanceof SignatureKeyCallback.AliasPrivKeyCertRequest) {
            SignatureKeyCallback.AliasPrivKeyCertRequest request =
               (SignatureKeyCallback.AliasPrivKeyCertRequest) cb.getRequest();
            String alias = request.getAlias();
            if (keyStore == null)
               initKeyStore();
            try {
               X509Certificate cert =
                  (X509Certificate) keyStore.getCertificate(alias);
               request.setX509Certificate(cert);
               // Assuming key passwords same as the keystore password
               PrivateKey privKey =
                  (PrivateKey) keyStore.getKey(alias, keyStorePassword.toCharArray());
               request.setPrivateKey(privKey);
            } catch (Exception e) {
               throw new IOException(e.getMessage());
            }

         } else {
             throw unsupported;
         }

      } else if (callbacks[i] instanceof DecryptionKeyCallback) {
         DecryptionKeyCallback cb = (DecryptionKeyCallback)callbacks[i];

         if (cb.getRequest() instanceof
DecryptionKeyCallback.X509SubjectKeyIdentifierBasedRequest) {
            DecryptionKeyCallback.X509SubjectKeyIdentifierBasedRequest request =
               (DecryptionKeyCallback.X509SubjectKeyIdentifierBasedRequest) cb.getRequest();
            byte[] ski = request.getSubjectKeyIdentifier();
            if (keyStore == null)
               initKeyStore();
            PrivateKey privKey = getPrivateKey(ski);
            request.setPrivateKey(privKey);

         } else if (cb.getRequest() instanceof DecryptionKeyCallback.X509IssuerSerialBasedRequest) {
            DecryptionKeyCallback.X509IssuerSerialBasedRequest request =
               (DecryptionKeyCallback.X509IssuerSerialBasedRequest) cb.getRequest();
            String issuerName = request.getIssuerName();

            BigInteger serialNumber = request.getSerialNumber();
            if (keyStore == null)
               initKeyStore();
            PrivateKey privKey = getPrivateKey(issuerName, serialNumber);
            request.setPrivateKey(privKey);

         } else if (cb.getRequest() instanceof DecryptionKeyCallback.X509CertificateBasedRequest) {
            DecryptionKeyCallback.X509CertificateBasedRequest request =
               (DecryptionKeyCallback.X509CertificateBasedRequest) cb.getRequest();
            X509Certificate cert = request.getX509Certificate();
            if (keyStore == null)
```

```
            initKeyStore();
          PrivateKey privKey = getPrivateKey(cert);
          request.setPrivateKey(privKey);

        } else if (cb.getRequest() instanceof DecryptionKeyCallback.AliasSymmetricKeyRequest) {
          DecryptionKeyCallback.AliasSymmetricKeyRequest request =
            (DecryptionKeyCallback.AliasSymmetricKeyRequest) cb.getRequest();
          if (symmKeyStore == null)
            initSymmKeyStore();
          String alias = request.getAlias();
          try {
            // Assuming key password same as key store password
            SecretKey symmKey =
              (SecretKey) symmKeyStore.getKey(alias, symmKeyStorePassword.toCharArray());
            request.setSymmetricKey(symmKey);
          } catch (Exception e) {
            throw new IOException(e.getMessage());
          }

        } else  {
          throw unsupported;
        }

      } else if (callbacks[i] instanceof EncryptionKeyCallback) {
        EncryptionKeyCallback cb = (EncryptionKeyCallback)callbacks[i];

        if (cb.getRequest() instanceof EncryptionKeyCallback.AliasX509CertificateRequest) {
          EncryptionKeyCallback.AliasX509CertificateRequest request =
            (EncryptionKeyCallback.AliasX509CertificateRequest) cb.getRequest();
          if (trustStore == null)
            initTrustStore();
          String alias = request.getAlias();
          try {
            X509Certificate cert =
              (X509Certificate) trustStore.getCertificate(alias);
            request.setX509Certificate(cert);
          } catch (Exception e) {
            throw new IOException(e.getMessage());
          }

        } else if (cb.getRequest() instanceof EncryptionKeyCallback.AliasSymmetricKeyRequest) {
          EncryptionKeyCallback.AliasSymmetricKeyRequest request =
            (EncryptionKeyCallback.AliasSymmetricKeyRequest) cb.getRequest();
          if (symmKeyStore == null)
            initSymmKeyStore();
          String alias = request.getAlias();
          try {
            // Assuming key password same as key store password
            SecretKey symmKey =
              (SecretKey) symmKeyStore.getKey(alias, symmKeyStorePassword.toCharArray());
            request.setSymmetricKey(symmKey);
          } catch (Exception e) {
            throw new IOException(e.getMessage());
          }

        } else {
          throw unsupported;
        }
```

```
            } else if (callbacks[i] instanceof CertificateValidationCallback) {
                CertificateValidationCallback cb = (CertificateValidationCallback)callbacks[i];
                cb.setValidator(new X509CertificateValidatorImpl());

            } else {
                throw unsupported;
            }
        }
    }

    private void initTrustStore() throws IOException {
        try {
            trustStore = KeyStore.getInstance(trustStoreType);
            trustStore.load(new FileInputStream(trustStoreURL), trustStorePassword.toCharArray());
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
    }

    private void initKeyStore() throws IOException {
        try {
            keyStore = KeyStore.getInstance(keyStoreType);
            keyStore.load(new FileInputStream(keyStoreURL), keyStorePassword.toCharArray());
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
    }

    private void initSymmKeyStore() throws IOException {
        try {
            symmKeyStore = KeyStore.getInstance(symmKeyStoreType);
            symmKeyStore.load(new FileInputStream(symmKeyStoreURL),
symmKeyStorePassword.toCharArray());
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
    }

    private X509Certificate getCertificateFromTrustStore(byte[] ski)
        throws IOException {

        try {
            Enumeration aliases = trustStore.aliases();
            while (aliases.hasMoreElements()) {
                String alias = (String) aliases.nextElement();
                Certificate cert = trustStore.getCertificate(alias);
                if (cert == null || !"X.509".equals(cert.getType())) {
                    continue;
                }
                X509Certificate x509Cert = (X509Certificate) cert;
                byte[] keyId = getSubjectKeyIdentifier(x509Cert);
                if (keyId == null) {
                    // Cert does not contain a key identifier
                    continue;
                }
                if (Arrays.equals(ski, keyId)) {
                    return x509Cert;
```

```
            }
        }
    } catch (Exception e) {
        throw new IOException(e.getMessage());
    }
    return null;
}

private X509Certificate getCertificateFromTrustStore(
    String issuerName,
    BigInteger serialNumber)
    throws IOException {

    try {
        Enumeration aliases = trustStore.aliases();
        while (aliases.hasMoreElements()) {
            String alias = (String) aliases.nextElement();
            Certificate cert = trustStore.getCertificate(alias);
            if (cert == null || !"X.509".equals(cert.getType())) {
                continue;
            }
            X509Certificate x509Cert = (X509Certificate) cert;
            String thisIssuerName =
                RFC2253Parser.normalize(x509Cert.getIssuerDN().getName());
            BigInteger thisSerialNumber = x509Cert.getSerialNumber();
            if (thisIssuerName.equals(issuerName) &&
                thisSerialNumber.equals(serialNumber)) {
                return x509Cert;
            }
        }
    } catch (Exception e) {
        throw new IOException(e.getMessage());
    }
    return null;
}

public PrivateKey getPrivateKey(byte[] ski) throws IOException {

    try {
        Enumeration aliases = keyStore.aliases();
        while (aliases.hasMoreElements()) {
            String alias = (String) aliases.nextElement();
            if (!keyStore.isKeyEntry(alias))
                continue;
            Certificate cert = keyStore.getCertificate(alias);
            if (cert == null || !"X.509".equals(cert.getType())) {
                continue;
            }
            X509Certificate x509Cert = (X509Certificate) cert;
            byte[] keyId = getSubjectKeyIdentifier(x509Cert);
            if (keyId == null) {
                // Cert does not contain a key identifier
                continue;
            }
            if (Arrays.equals(ski, keyId)) {
                // Asuumed key password same as the keystore password
                return (PrivateKey) keyStore.getKey(alias, keyStorePassword.toCharArray());
            }
```

```
            }
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
        return null;
    }

    public PrivateKey getPrivateKey(
        String issuerName,
        BigInteger serialNumber)
        throws IOException {

        try {
            Enumeration aliases = keyStore.aliases();
            while (aliases.hasMoreElements()) {
                String alias = (String) aliases.nextElement();
                if (!keyStore.isKeyEntry(alias))
                    continue;
                Certificate cert = keyStore.getCertificate(alias);
                if (cert == null || !"X.509".equals(cert.getType())) {
                    continue;
                }
                X509Certificate x509Cert = (X509Certificate) cert;
                String thisIssuerName =
                    RFC2253Parser.normalize(x509Cert.getIssuerDN().getName());
                BigInteger thisSerialNumber = x509Cert.getSerialNumber();
                if (thisIssuerName.equals(issuerName) &&
                    thisSerialNumber.equals(serialNumber)) {
                    return (PrivateKey) keyStore.getKey(alias, keyStorePassword.toCharArray());
                }
            }
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
        return null;
    }

    public PrivateKey getPrivateKey(X509Certificate certificate)
        throws IOException {

        try {
            Enumeration aliases = keyStore.aliases();
            while (aliases.hasMoreElements()) {
                String alias = (String) aliases.nextElement();
                if (!keyStore.isKeyEntry(alias))
                    continue;
                Certificate cert = keyStore.getCertificate(alias);
                if (cert != null && cert.equals(certificate))
                    return (PrivateKey)
keyStore.getKey(alias, keyStorePassword.toCharArray());
            }
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
        return null;
    }

    private void getDefaultPrivKeyCert(
```

```
        SignatureKeyCallback.DefaultPrivKeyCertRequest request)
        throws IOException {

        String uniqueAlias = null;
        try {
            Enumeration aliases = keyStore.aliases();
            while (aliases.hasMoreElements()) {
                String currentAlias = (String) aliases.nextElement();
                if (keyStore.isKeyEntry(currentAlias)) {
                    Certificate thisCertificate = keyStore.getCertificate(currentAlias);
                    if (thisCertificate != null) {
                        if (thisCertificate instanceof X509Certificate) {
                            if (uniqueAlias == null) {
                                uniqueAlias = currentAlias;
                            } else {
                                // Not unique!
                                uniqueAlias = null;
                                break;
                            }
                        }
                    }
                }
            }
            if (uniqueAlias != null) {
                request.setX509Certificate(
                    (X509Certificate) keyStore.getCertificate(uniqueAlias));
                request.setPrivateKey(
                    (PrivateKey) keyStore.getKey(uniqueAlias, keyStorePassword.toCharArray()));
            }
        } catch (Exception e) {
            throw new IOException(e.getMessage());
        }
    }

    private static byte[] getSubjectKeyIdentifier(X509Certificate cert) {
        String SUBJECT_KEY_IDENTIFIER_OID = "2.5.29.14";
        byte[] subjectKeyIdentifier =
            cert.getExtensionValue(SUBJECT_KEY_IDENTIFIER_OID);
        if (subjectKeyIdentifier == null)
            return null;
        byte[] dest = new byte[subjectKeyIdentifier.length - 4];
        System.arraycopy(
            subjectKeyIdentifier, 4, dest, 0, subjectKeyIdentifier.length - 4);
        return dest;
    }


    private class PlainTextPasswordValidator implements PasswordValidationCallback.PasswordValidator
{

        public boolean validate(PasswordValidationCallback.Request request)
            throws PasswordValidationCallback.PasswordValidationException {

            PasswordValidationCallback.PlainTextPasswordRequest plainTextRequest =
                (PasswordValidationCallback.PlainTextPasswordRequest) request;
            if ("Ron".equals(plainTextRequest.getUsername()) &&
                "noR".equals(plainTextRequest.getPassword())) {
                return true;
```

```
            }
            return false;
        }
    }


    private class X509CertificateValidatorImpl implements
CertificateValidationCallback.CertificateValidator {

        public boolean validate(X509Certificate certificate)
            throws CertificateValidationCallback.CertificateValidationException {

            if (isSelfCert(certificate)) {
                return true;
            }

            try {
                certificate.checkValidity();
            } catch (CertificateExpiredException e) {
                e.printStackTrace();
                throw new CertificateValidationCallback.CertificateValidationException("X509Certificate
Expired", e);
            } catch (CertificateNotYetValidException e) {
                e.printStackTrace();
                throw new CertificateValidationCallback.CertificateValidationException("X509Certificate not
yet valid", e);
            }

            X509CertSelector certSelector = new X509CertSelector();
            certSelector.setCertificate(certificate);

            PKIXBuilderParameters parameters;
            CertPathBuilder builder;
            try {
                if (trustStore == null)
                    initTrustStore();
                parameters = new PKIXBuilderParameters(trustStore, certSelector);
                parameters.setRevocationEnabled(false);
                builder = CertPathBuilder.getInstance("PKIX");
            } catch (Exception e) {
                e.printStackTrace();
                throw new CertificateValidationCallback.CertificateValidationException(e.getMessage(), e);
            }

            try {
                PKIXCertPathBuilderResult result =
                    (PKIXCertPathBuilderResult) builder.build(parameters);
            } catch (Exception e) {
                e.printStackTrace();
                return false;
            }
            return true;
        }

        private boolean isSelfCert(X509Certificate cert)
            throws CertificateValidationCallback.CertificateValidationException {
            try {
                if (keyStore == null)
```

```
                initKeyStore();
                Enumeration aliases = keyStore.aliases();
                while (aliases.hasMoreElements()) {
                    String alias = (String) aliases.nextElement();
                    if (keyStore.isKeyEntry(alias)) {
                        X509Certificate x509Cert =
                            (X509Certificate) keyStore.getCertificate(alias);
                        if (x509Cert != null) {
                            if (x509Cert.equals(cert))
                                return true;
                        }
                    }
                }
                return false;
            } catch (Exception e) {
                e.printStackTrace();
                throw new CertificateValidationCallback.CertificateValidationException(e.getMessage(), e);
            }
        }
    }
}
```

Liite 5: Asiakkaan salattu pyyntöviesti

<soap:Body>
<xenc:EncryptedData Id="EncryptedContent-f002323e-aeff-3e80-b550-6f55f162eff1"
Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
<xenc:CipherData>
<xenc:CipherValue>BhAADV9Qwxuot7V8qCgYA4/gWfLaBM182Gaz9/a0k32hJUtZRJmlFjVQNLs1X
Ua7/vnnRRip74AisqNm084eSD2UfstZPawY1+pi5mp3spir/sF9ujbo8yw9SITsub14VWzXsgTuaAJHTcR
v8fmZgjti2uGZUaxWukndh5B/oYIsUJPRu6xnX/rfGpG/ag7V5fY6VqD0QADgYDZUZwdGQmXoBUE
jEyJpcjSjg3MIgGIMyKbhhbmLAYmeIqsCgYCCARLuzDK75TgR1rcgPYRt1cSPo7g/bBHsQAIUPDj9j
XGqK1EilS3HLf8l5HOOAQf98XBgcHqjn</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</soap:Body>

Liite 6: Web-palvelun allekirjoitettu vastausviesti


```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc=
"http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="http://wss.samples.microsoft.com" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header>
    <wsse:Security env:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="Id3937690203981766074">MIIDWTCCAsKgAwIBAgIBATANBgkqhkiG9w0BAQQFADB0
MQswCQYDVQQGEwJOQTELMAkGA1UECBMCTkExCzAJBgNVBAcTAk5BMQswCQYDVQQK
EwJOQTELMAkGA1UECxMCTkExHjAcBgNVBAMTFWNlcnRpZmljYXRlLWF1dGhvcml0eTERM
A8GCSqGSIb3DQEJARYCTkEwHhcNMDQwNDA5MjAxNTUwWhcNMDUwNDA5MjAxNTUwWjB
yMQswCQYDVQQGEwJOQTELMAkGA1UECBMCTkExCzAJBgNVBAcTAk5BMQswCQYDVQQK
EwJOQTELMAkGA1UECxMCTkExHDAaBgNVBAMTE3h3cy1zZWN1cml0eS1zZXJ2ZXIxETAPBg
kqhkiG9w0BCQEWAk5BMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDTEAYXDMEpu
R5m6mVMNPBTvSbiFJJfykCJuANuIsYMqCGbajlJYxkPSFjsUbxhSinSTCYGlHfqqesxKyk8dPcX/LAu
jCOwwd1tDdql2sQZ6WZwf21wfSv65TczldTTlbADzttoJbzsICl7LH85900XUv25mmouS96Cw5CSdgL/
8wIDAQABo4H8MIH5MAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9wZW5TU0wgR2
VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBS1RijUqFm1EOJR/UtU0NSilNzOajCBngYD
VR0jBIGWMIGTgBS1BYo8LSYEn16yMWhvreilyanXfqF4pHYwdDELMAkGA1UEBhMCTkExCzAJ
BgNVBAgTAk5BMQswCQYDVQQHEwJOQTELMAkGA1UEChMCTkExCzAJBgNVBAsTAk5BMR
4wHAYDVQQDExVjZXJ0aWZpY2F0ZS1hdXRob3JpdHkxETAPBgkqhkiG9w0BCQEWAk5BggEAM
A0GCSqGSIb3DQEBBAUAA4GBAEjzXe4rgJmzrbDKYqe5MLSe8dDwkTWZhlN9OEd3WdVDZAS0t
XSC22sWahPBN9RcAkOSCNyX42vCNxjHwWIbtxCB8ODXBbvJNPzsEA2dVK04VlixzhJsdAZB5BE
eG35nhL3Z0TSo57EclJKCmATyMn0RNWxJFJyHAfj09r4zf</wsse:BinarySecurityToken>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
        <ds:Reference URI="#Id4963882732514175667">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>w/aCLl9IAygdqSus1/TSl6/4ZX0=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#Id4859776246913402640">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>9I0Tb+mmeLwgQAiWhIdTn2/q30o=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
giP3FNF2GCL0DsMgz7ATVj+5VbPtPfmPzWgmef1xpII6ErPxJtf7wlBsIxFc3TcxKKyPolB8wBWp
Vtj67lM8aqTjZtDAGJy8Ftkv3xBb/WC9HsLluwOnTXIG1fDTgyU7iNXcjEVzGNV+n84qshn8xexL
691BOF6KmLAPFs0fZK4=
</ds:SignatureValue>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#Id3937690203981766074" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
        </wsse:SecurityTokenReference>
```

```
        </ds:KeyInfo>
       </ds:Signature>
       <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="Id4859776246913402640">
        <wsu:Created>2008-05-24T10:14:44Z</wsu:Created>
        <wsu:Expires>2008-05-24T10:19:44Z</wsu:Expires>
       </wsu:Timestamp>
     </wsse:Security>
   </env:Header>
   <env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="Id4963882732514175667">
     <ns0:submitOrderResponse>
       <result>1441227696</result>
     </ns0:submitOrderResponse>
   </env:Body>
 </env:Envelope>
```