Tietoliikenteen salaaminen Java-sovelluksen ja tietokannan välillä

Miika Päivinen

13.12.2005

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Sähköisen kanssakäymisen määrän lisääntyessä jatkuvasti, kasvaa myös sellaisen tietoliikenteen määrä, jonka täytyy säilyä vain niiden osapuolien hallussa, joille se kuuluu. Jo varsin yksinkertaisissa sovelluksissa vaaditaan käytettäväksi salattua tietoliikennettä.

Tässä tutkielmassa perehdytään kirjallisuuden avulla salauksen perusteisiin, yleisimpiin salausmenetelmiin ja niiden turvallisuuteen. Tarkemmin analysoidaan Java-ohjelmointikielen ja Oracle 9iR2 -tietokannan tarjoamia salaustoimintoja. Esimerkin avulla esitetään, mitä toimenpiteitä vaatii Oracle JDBC Thin –ajuria käyttävän asiakas-sovelluksen ja tietokannan välisen tietoliikenteen salaus. Salauksen onnistuminen todennetaan kuuntelemalla tietoliikennettä ja tallentamalla se lokitiedostoon. Lokitiedostoa tutkimalla nähdään mitä eroa on salaamattoman ja salatun tietoliikenteen välillä.

Esimerkin tuloksista voitiin todeta, että ilman tietoliikenteen salaamista, tietoliikennettä kuuntelemalla voidaan saada selville kaikki SQL-lauseet, joita asiakassovellus generoi tietokannalle. Tietoliikenteen salauksen ollessa päällä, ei kuuntelun tuloksena syntyneestä lokitiedostosta ollut nähtävissä selkokielisiä SQL-lauseita. Salauksen käyttöönotto vaatii tietokannan erillisen turvallisuuslisäosan käyttöönottoa.

ACM-luokat (ACM Computing Classification System, 1998 version): E.3, D.3, H.2.7

Avainsanat: salausmenetelmät, salaus, tietoliikenne, Java, Oracle

1 JOHDANTO

Salauksesta puhuttaessa sekoitetaan hyvin usein kaksi toisistaan poikkeavaa käsitettä: salaus ja tietoturva. Tietoturva käsitteenä on kuitenkin huomattavasti salausta laajempi kokonaisuus. Järvisen (2003) mukaan tietoturvan avulla suojataan tiedot erityyppisiltä uhkilta, joita ovat esim. luvaton käyttö tai kiintolevyn fyysinen rikkoutuminen. Salaus on siis vain osa tietoturvaa ja sen avulla voidaan kohentaa tiettyjen tietoturvan osa-alueiden laatua. Tietoturva-käsitteeseen kuuluu mm. seuraavat osa-alueet: luottamuksellisuus, eheys, saatavuus, pääsynvalvonta, kiistämättömyys ja todennus.

Luottamuksellisuudella (confidentiality) käsitetään ne toimenpiteet ja säännöt joilla varmistetaan, että tiettyyn tietoon pääsevät käsiksi vain ne ihmiset, joilla on siihen oikeus. Esim. tuotekehitystiedot ovat tietoja, joihin vain osa yrityksen henkilökunnasta pääsee käsiksi ja ulkopuoliset eivät ollenkaan. Salausmenetelmien avulla voidaan huomattavasti parantaa tietojen luottamuksellisuutta. Yleensä tietojen luottamuksellisuuden rikkoontuminen on todennäköisintä siirron (tietoverkot) ja säilytyksen aikana. Salatun tiedon vuotaminen ulkopuolisten käsiin tietoliikenneyhteyden aikana tai esim. tietokoneen fyysisen varkauden yhteydessä on helppoa estää käyttämällä salausta (Järvinen, 2003).

Eheydellä (integrity) tarkoitetaan, että tietojen oikeudettomat muutokset on estetty. Eheys voi rikkoontua mm. mikäli tiedonsiirron aikana kadotetaan osa sanoman sisällöstä. Eheyden saavuttamiseksi tulee olla keinot, joilla voidaan todeta mikäli eheyttä on rikottu (Menezes & al., 1997).

Saatavuus (availability) tarkoittaa Järvisen (2003) mukaan tietoturvan kannalta sitä, että määritelty tieto on saatavissa, vaikka erinäisiä esteitä (sähkökatkos, levyrikko, jne.) ilmenisikin. Saatavuutta voidaan parantaa esim. huolehtimalla asianmukaisesta varmuuskopioinnista. Saatavuus on ainoa tietoturvan osa-alue, johon salausmenetelmien avulla ei voida vaikuttaa parantavasti, sillä esim. salasanalla suojatun tietolähteen salasanan unohtaminen itse asiassa heikentää tietojen saatavuutta.

Pääsynvalvonnan (access control) avulla varmistetaan, että vain sallituilla käyttäjillä ja ohjelmilla on oikeus päästä käyttämään järjestelmän tietoja tai toimintoja (Menezes & al., 1997).

Kiistämättömyys (non-repudiation) tarkoittaa, että myöhemmin voidaan todentaa kaikki oleelliset tapahtumat tietojärjestelmässä. Voidaan esimerkiksi kiistattomasti osoittaa, kuka on käynyt lukemassa tietyn asiakkaan asiakastietoja (Menezes & al., 1997).

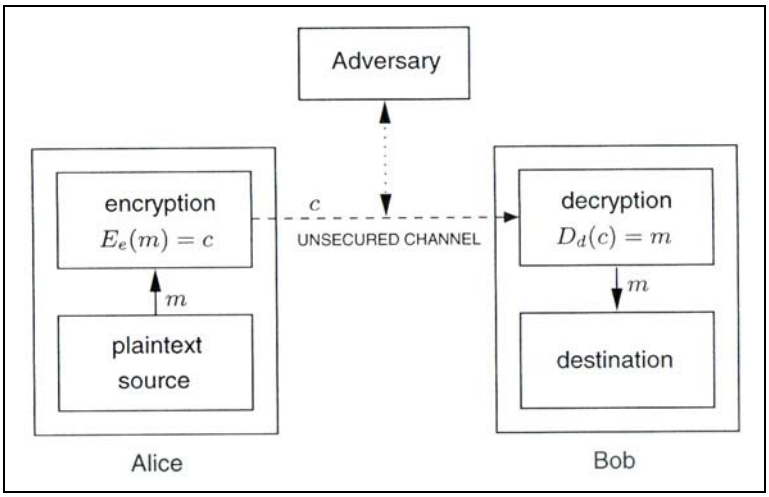
Todennuksen (authentication) avulla voidaan varmistaa, onko tietojärjestelmässä esiintyvä ihminen tai laite se kuka väittää olevansa. Koneiden yhteydessä puhutaan aitoudesta. Todennuksella on erittäin tärkeä rooli tietoturvan toteuttamisessa: Jos luottamuksellisuus rikkoutuu, se tarkoittaa, että ulkopuolinen pääsee näkemään hänelle kuulumatonta tietoa, mutta jos todennus pettää, se tarkoittaa, että ulkopuolinen pääsee pahimmassa tapauksessa lukemaan ja tuottamaan tietoa toisen nimissä (Järvinen, 2003).

Tämän tutkielman tarkoituksena on esitellä ne salaukseen liittyvät asiat, joiden avulla voidaan salata asiakaskoneella suoritettavan Java-ohjelman ja erillisellä palvelimella sijaitsevan Oracle-tietokannan välinen tietoliikenne. Tietoliikenteen salauksen toteuttamiseksi käyttäjän täytyy tuntea perusteet salausmenetelmistä, Java-ohjelmointikielestä sekä käytettävästä tietokannasta. Lisäksi tulee tietää käytettävien välineiden tarjoamat vaihtoehdot salauksen toteuttamiseksi.

Tutkielman luvussa 2 käsitellään salausta yleisesti, salausmenetelmiä ja salaukseen käytettyjä algoritmeja. Tutkielman luvussa 3 perehdytään luvun 2 teorian avulla yksittäisten tuotteiden tarjoamiin vaihtoehtoihin salauksen toteuttamisessa. Tuotteet, joiden osalta asiaa käsitellään, ovat Java-ohjelmointikieli (versio 1.4) ja Oracle-tietokanta (versio 9i Release 2). Luvussa 4 esitetään käytännön esimerkin avulla kuinka tietoliikenne voidaan salata luvun 3 esittelemässä ympäristössä toimittaessa. Esimerkin yhteydessä näytetään mitä toimenpiteitä salauksen käyttöönotto vaatii sovellukselta ja sen käyttämältä tietokantapalvelimelta. Salauksen toimivuus todennetaan kuuntelemalla asiakaskoneen tietoliikennettä salaamattomana ja salattuna.

2 SALAUSMENETELMIÄ

Tiedon salauksessa (encryption) *selväkielinen teksti* (plaintext) muutetaan matemaattisen algoritmin ja *avaimen* (key) avulla salattuun muotoon, *salatekstiksi* (ciphertext). Salatekstin palauttaminen takaisin ymmärrettävään muotoon tapahtuu purkamalla salaus (decryption) (Käpylä, 2000). Menezes & al. (1997) kuvaavat salauksen periaatteen kuvassa 1. Alice salaa selvätekstin (*m*) käyttämällä salausalgoritmia, jolloin saadaan tuloksena salateksti *c*. Alice lähettää salatekstin Bobille, eivätkä ulkopuoliset tahot pääse selville viestin sisällöstä, vaikka kaappaisivatkin viestiliikenteen. Bob purkaa saamansa salatekstin ja saa käyttöönsä selvätekstin. Riippuen käytetystä salausmenetelmästä, Alicella ja Bobilla tulee olla joko yhteinen salainen avain tai sitten Alicen tulee käyttää Bobin julkista avainta salaukseen ja Bobin omaa yksityistä avaintaan salatekstin purkamiseen.



Kuva 1 Tiedon salaamisen periaate (Menezes & al., 1997).

Käpylän (2000) ja Järvisen (2003) mukaan salaus ei saa koskaan perustua salaiseen salausalgoritmiin (algoritmin toimintaperiaatetta ei ole julkistettu), koska tällöin ei voida olla varmoja esim. algoritmin sisältämistä salaovista. Salaovien avulla ne tuntevan henkilön on helppoa murtaa salattu viesti ilman avainta.

Vaikka algoritmi olisi osapuolten itsensä kehittämä salainen algoritmi, jolloin tarkoituksellisen salaoven mahdollisuus voidaan sulkea pois, on olemassa muitakin syitä salaisien algoritmien käytön välttämiseen. Koska salausalgoritmit ovat usein matemaattisesti monimutkaisia, on niiden turvallisuuden analysoiminen hankalaa. Jos

algoritmin toimintaperiaatteen tuntevat vain harvat ihmiset, sen turvallisuuttakin analysoi kovin pieni joukko ihmisiä. Vakuuttuakseen algoritmin toimivuudesta sen kehittäjien pitäisi antaa se laajalle joukolle analysoitavaksi. Vanha, pitkään ja laajalti puolueettomasti analysoitu algoritmi on siksi huomattavasti turvallisempi vaihtoehto. Useat algoritmien kehittäjät tarjoavatkin julkisesti rahapalkinnon sille joka kykenee murtamaan heidän salausalgoritminsa (RSA Laboratories, 2005).

Koska algoritmin tulee edellä mainittujen syiden johdosta olla julkinen, tulee salauksen käytännössä perustua jonkinlaiseen salassa pidettävään avaimeen. Tämä tuo Menezesin & al. (1997) mukaan myös sellaisen edun, että mikäli kolmas osapuoli saa selvitettyä salauksen, riittää kun vaihdetaan vain avainta, kun taas salaisen algoritmin paljastuttua pitäisi kehittää kokonaan uusi algoritmi.

RSA Laboratoriesin (2000) mukaan salaamiseen ja purkamiseen käytetään joko molempiin samaa avainta (symmetrinen salaus) tai salaamiseen ja purkamiseen eri avainta (epäsymmetrinen salaus, julkisen avaimen salausmenetelmä). Käytännön sovelluksissa kuitenkin hyödynnetään yleensä molemmista tekniikoista niiden parhaat puolet käyttämällä niitä yhdessä (Burke & al., 2000).

Tutkielman seuraavissa kohdissa käsitellään näitä kahta salausmenetelmää (mm. toimintaperiaate, erilaiset toteutustavat, turvallisuus, käyttökohteet). Kun ko. asiat on käyty läpi, voidaan siirtyä seuraaviin lukuihin, joissa puhutaan tietoliikenteen, Java-sovelluksen ja Oracle-tietokannan tietojen salauksesta.

Edellä mainittujen menetelmien lisäksi mainittakoon tässä yhteydessä vielä kolmas salausmenetelmä: *kvanttisalaus.* Kvanttisalauksessa salaus perustuu fotonien tilojen vertailuun. Teoriassa kvanttisalausta käyttämällä voidaan osapuolien välinen avain vaihtaa siten, ettei ulkopuolisen tahon ole sitä mahdollista selvittää ilman, etteivät salauksen viralliset osapuolet saa sitä välittömästi selville (Hurwitz, 2000). Kvanttisalausta käyttämällä on vuonna 2004 onnistuttu salaamaan pankkitapahtuman tietoliikenne (Quantum Technologies, 2005). Kvanttisalus eroaa toteutustavaltaan oleellisesti muista salausmenetelmistä ja on vielä tekniikkana uusi eikä laajemmalti käytössä. Tästä johtuen sitä ei käsitellä tässä tutkielmassa enempää, vaan keskitytään yleisesti käytössä oleviin menetelmiin.

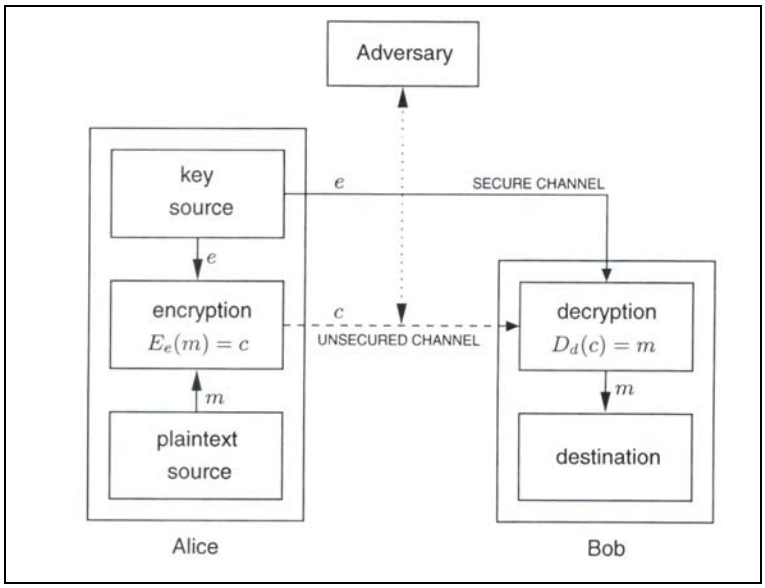
2.1 Symmetrinen salaus

Seuraavissa alakohdissa esitellään aluksi symmetrisen salauksen toimintaperiaatteet. Toimintaperiaatteiden läpikäymisen jälkeen tutustutaan salaukseen käytettyihin *salaimiin,* lisäksi käsitellään avaimen pituuden merkitystä ja sen vaihtamista osapuolien välillä.

2.1.1 Yleisperiaate

Symmetrinen salaus perustuu tietokoneella tapahtuvaan bittien sekoittamiseen (Järvinen, 2003). Käytännössä salaus tapahtuu syöttämällä selväkielinen teksti ja salainen avain *salaimelle,* joka sekoittaa selkokielisen tekstin salatekstiksi. Salaista avainta tuntemattomat henkilöt eivät tämän jälkeen pysty purkamaan salausta (Burke & al., 2000).

Menezes & al. (1997) antavat symmetriselle salaukselle seuraavanlaisen määritelmän: Sanotaan että salausmenetelmä on *symmetrinen*, jos jokainen salaus-/purkuavainpari (*e*,*d*) ovat sellaisia, joista on helppo laskea *d*, vaikka tiedetään vain *e*, ja vastaavasti *e* voidaan helposti johtaa *d*:n avulla. Edelleen Menezesin & al. (1997) mukaan käytännössä kaikissa käytössä olevissa symmetrisissä salausmenetelmissä salaus- ja purkuavain ovat samat (*e*=*d*), eli molemmilla osapuolilla on käytössään sama salainen avain.



Kuva 2 Symmetrisen salauksen periaate (Menezes & al., 1997).

Kuvassa 2 esitetään symmetrisen salauksen periaate Menezesin & al. (1997) näkemyksen mukaan. Alice toimittaa purkamiseen tarvittavan avaimen Bobille ja salaa sitten viestin.

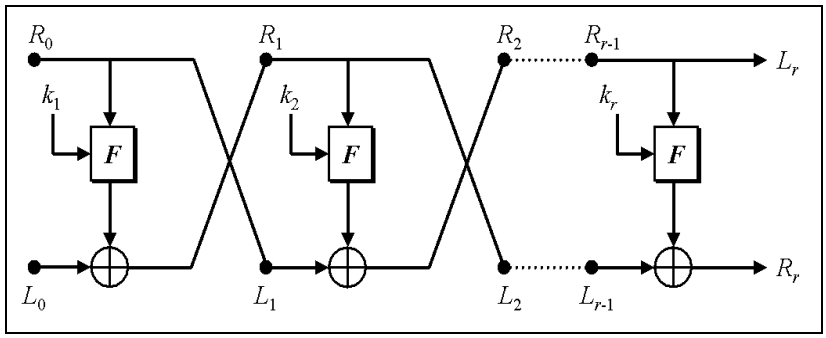
Kuten kuvasta huomataan, täytyy salausavain toimittaa vastaanottavalle osapuolelle ennen salatun viestin lähettämistä. Vaikka salateksti voidaankin lähettää avoimen kanavan ylitse, niin avain täytyy toimittaa jonkin turvallisen kanavan kautta. RSA Laboratories (2000) mainitseekin avaimen turvallisen vaihtamisen symmetrisen salauksen suurimmaksi haasteeksi. Avaimen vaihdosta aiheutuvaa ongelmaa ei ole esimerkiksi salattaessa omia tiedostoja kiintolevylle (Käpylä, 2000).

Symmetrisessä salauksessa käytetyt salaimet jaetaan lohko- ja jonosalaimiin toimintaperiaatteensa mukaan. Lohkosalaimilla salataan tietyn kokoinen lohko selvätekstiä kerrallaan, kun taas jonosalaimilla salataan bitti tai tavu kerrallaan.

2.1.2 Lohkosalaimet

Lohkosalain (block cipher) on Menezesin & al. (1997) määritelmän mukaan funktio, joka liittää selväkielisen tekstin *n*-bittiset lohkot salatekstin *n*-bittisiin lohkoihin (*n* ilmaisee lohkon koon). Funktiolle annetaan lisäksi parametrina salainen avain. Käytännössä tämä tarkoittaa sitä, että salain ottaa salattavasta tekstistä määrätynmittaisen osan ja tekee siitä samanmittaisen salatekstin. Salateksti puretaan yleensä ajamalla salainfunktio läpi toisinpäin avain ja salateksti syötteenä. Järvisen (2003) mukaan lohkosalaimien lohkon koko on yleensä 64 tai 128 bittiä.

Iteroiva lohkosalain on salaintyyppi, jossa selväteksti salataan luomalla alkuperäisestä avaimesta useita *aliavaimia* ja toistamalla samaa salausfunktiota antamalla sille joka kierroksella syötteenä edellisen kierroksen salateksti ja aliavain (RSA Laboratories, 2000). Eräs tällainen salaintyyppi on *Feistelin salain* (kuva 3*).*



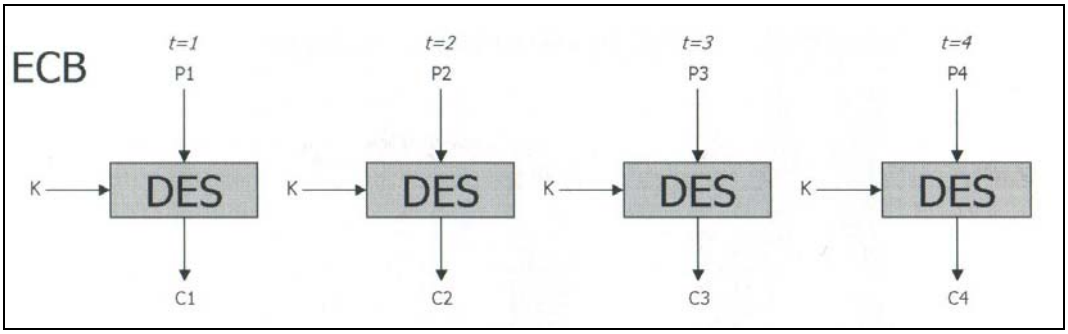
Kuva 3 Feistelin salain (RSA Laboratories, 2000).

Feistelin salaimessa luodaan aluksi salaisesta avaimesta aliavaimet (k_1, k_2 ja k_3). Tämän jälkeen selväkielinen teksti jaetaan kahtia (R_0 ja L_0). Toinen puoli annetaan aliavaimen kanssa syötteenä salausfunktiolle. Tulos ja käyttämätön selvätekstin puolikas käsitellään XOR-operaatiolla. Tämän jälkeen aloitetaan uusi iteraatio, kuitenkin siten että XOR-operaation tuottamaa tietoa (R_1) käytetään seuraavan iteraation funktion syötteenä uuden aliavaimen (k_2) kanssa ja edellisen kierroksen syötettä (R_0) käytetään seuraavaan XOR-operaatioon (RSA Laboratories, 2000).

Tällä tyylillä salatun tiedon purkaminen tapahtuu käänteisessä järjestyksessä. Kun vastaan-ottaja tietää salaisen avaimen, hän kykenee luomaan aliavaimet. Tämän jälkeen hän syöttää salatekstin ja aliavaimet käänteisessä järjestyksessä salaimeen (RSA Laboratories, 2000).

Salattaessa selvätekstiä, jonka pituus ylittää lohkosalaimen lohkonpituuden, käytetään lohkosalaimia erilaisissa *moodeissa*. Salaimesta riippumattomia standardoituja moodeja on kaikkiaan seitsemän kappaletta (Järvinen, 2003). Seuraavaksi esitellään neljä yleisimmin käytettyä moodia: ECB (electronic codebook), CBC (cipher-block chaining), CFB (cipher feedback) ja OFB (output feedback).

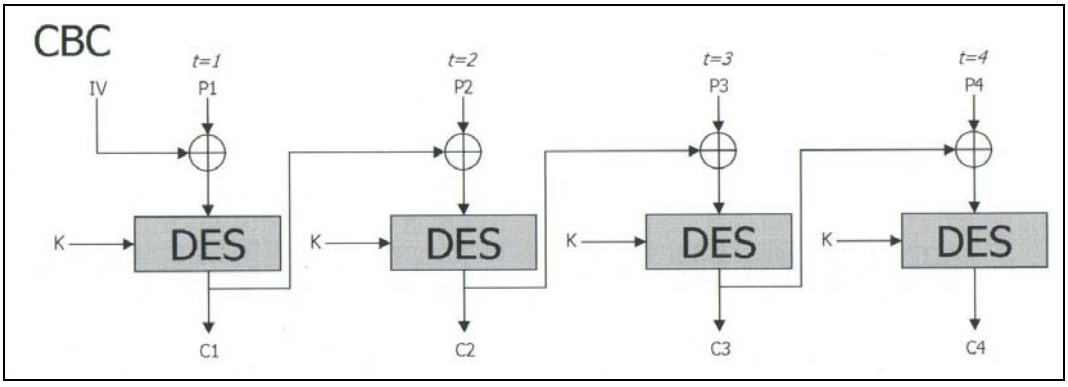
ECB-moodi jakaa selvätekstin määritellyn lohkokoon mukaisiin jaksoihin ja jokainen salataan samalla avaimella (Järvinen, 2003). Menezesin & al. (1997) mukaan ECB-moodia ei tule käyttää lohkon koon ylittävien selvätekstien salaamiseen. Perusteluina on mainittu mm. ketjuttomuus: salatekstin lohko ei riipu millään lailla edellisestä lohkosta, ts. vastaan-ottaja ei välttämättä huomaa, vaikka yksi lohko olisi siirron aikana poistettu tai vaihdettu. ECB-moodi ei myöskään häivytä selvätekstin toistoa salatekstistä, jos samat selvätekstin osuudet (esim. samat sanat) sattuvat useamman lohkon alkuun, ne näkyvät samanlaisena salatekstinä. Tämä mahdollistaa salauksen murtamisen. ECB-moodia voidaan kuitenkin käyttää esim. uuden salaisen avaimen välittämiseen, mikäli avaimen pituus ei ylitä lohkon kokoa. ECB-moodin toimintaperiaate on esitetty kuvassa 4.



Kuva 4 ECB-moodin toimintaperiaate (Järvinen, 2003).

CBC-moodi on eniten käytetty lohkosalainmoodi (Järvinen 2003). Moodin käyttöä varten tarvitaan Menezesin & al. (1997) mukaan *n*-bitin mittaisia selväkielitekstilohkoja, *k*-bittiä pitkä salausavain *K* ja *n*-bittiä pitkä *alustusvektori* (IV, initialization vector).

Salaus tapahtuu kuvan 5 esittämällä tavalla (*t* = ajanhetki): ensimmäinen selvätekstilohko ja alustusvektori yhdistetään XOR-operaatiolla, jonka jälkeen tulos annetaan salausavaimen kanssa syötteenä salausfunktiolle. Tuloksena saadaan ensimmäinen salatekstilohko, joka vastaavasti yhdistetään seuraavan selvätekstilohkon kanssa XOR-operaatiolla ennen salaamista. Jokaisen lohkon salaamisessa käytetään samaa salausavainta. Salauksen purkaminen tapahtuu Järvisen (2003) mukaan siten, että vastaanottaja antaa purkufunktiolle syötteenä salausavaimen ja ensimmäisen saamansa salatekstilohkon. Saatu tulos yhdistetään alustusvektorin XOR-operaatiolla, jolloin saadaan ensimmäinen selvätekstilohko. Seuraava salatekstilohko puretaan ja yhdistetään XOR-operaatiolla edellisen salatekstilohkon kanssa.

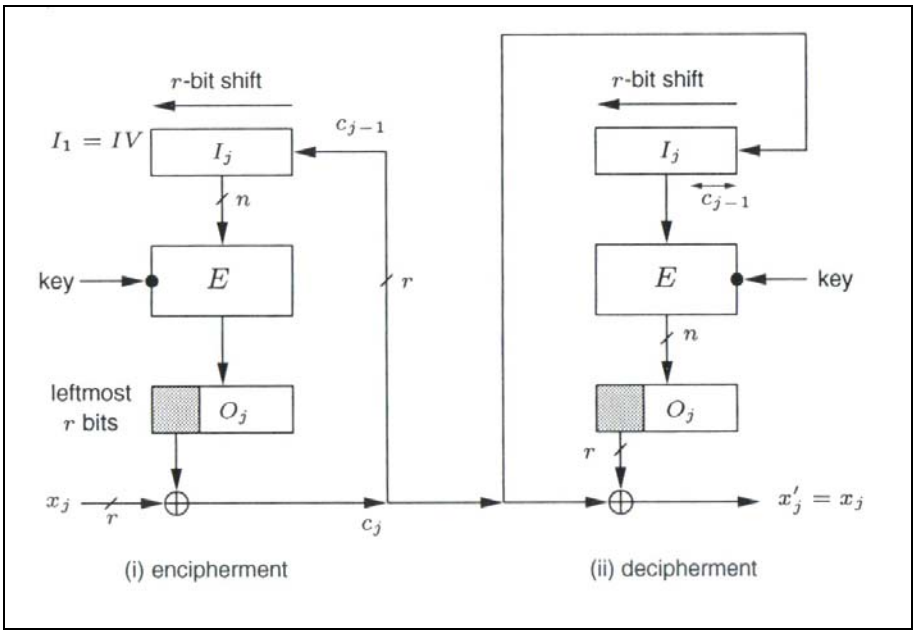


Kuva 5 CBC-moodin toimintaperiaate (Järvinen, 2003).

RSA Laboratoriesin (2000) mukaan tällaisen ketjuttamisen johdosta salatekstistä ei ilmene selvätekstissä mahdollisesti olleet toistumat. Ne katoavat koska yksittäistä salatekstilohkoa vastaava selväkielinen lohko on ensin yhdistetty XOR-operaatiolla edellisen salatekstilohkon kanssa. Ulkopuoliset eivät voi myöskään siirron aikana manipuloida salatekstilohkoja, koska jokainen lohko on riippuvainen kaikista edellisistä lohkoista.

Alustusvektori täytyy vaihtaa osapuolien kesken ennen salausta. RSA Laboratoriesin (2000) mielestä sitä ei ole välttämätöntä siirron ajaksi salata, mutta Menezes & al. (1997) ja Järvinen (2003) suosittavat sen vaihtamista salattuna varmuuden vuoksi. Järvisen mukaan alustusvektorin salaamiseen voidaan käyttää ECB-moodia, koska vektori sopii yhteen lohkoon. RSA Laboratories (2000) suosittelee vaihtamaan alustusvektorin joka kerta, jos samalla salausavaimella salataan useampia viestejä.

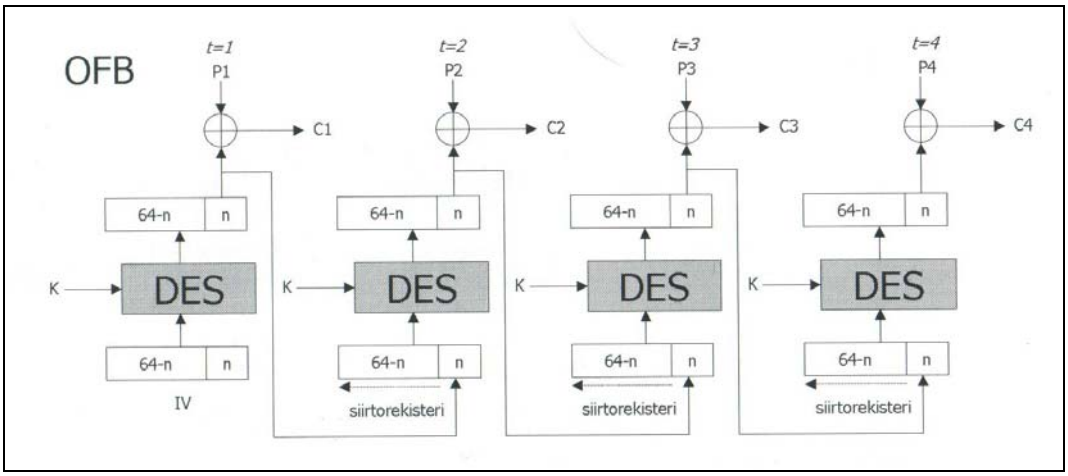
CFB-moodi mahdollistaa lohkosalaimen muuttamisen *jonosalaimeksi*, jolloin voidaan salata ja lähettää esim. vain yksi tavu kerrallaan (Järvinen, 2003). Moodin toimintaperiaate näkyy kuvassa 6. CFB-moodissa ensimmäinen salatekstilohko saadaan antamalla ensin salausfunktiolle syötteenä alustusvektori ja salausavain. Tuloksena saatu bittijono yhdistetään selvätekstilohkon kanssa XOR-operaatiolla. Saatua salatekstilohkoa käytetään seuraavassa salausfunktion syötteessä (RSA Laboratories, 2000).



Kuva 6 CFB-moodin toimintaperiaate (Menezes & al. 1997).

Koska salaukseen käytettävät salaimet tarvitsevat ja palauttavat lohkon koon mukaisen tuloksen (esim. 64 tai 128 bittiä), käytetään tuloksen käsittelyyn *siirtorekisteriä* (feedback shift register). Tällöin salaimen palauttamasta tuloksesta annetaankin XOR-operaatioon vain *r* bittiä (vasemmalta) ja vastaavasti selvätekstistä otetaan myös vain *r* bittiä. Saatu *r*-bitin salateksti syötetään takaisin siirtorekisteriin oikealta, jolloin siirtorekisterissä olevat ”ylimääräiset” bitit putoavat pois vasemmalta (kuva 6). Mikäli *r*:ksi valitaan vaikkapa 8 bittiä, pystytään selvätekstiä salaamaan (ja lähettämään) tavu kerrallaan. Tämä on hyödyllistä esim. kirjoitettaessa tekstiä/komentoja salatun yhteyden yli.

OFB-moodi mahdollistaa myös lohkosalaimen käytön jonosalaimena. OFB-moodi on hyvin samankaltainen kuin CFB-moodi, mutta OFB-moodissa edellinen selvätekstilohko ei vaikuta siirtorekisteriin vietäviin arvoihin, koska bitit otetaan siirtorekisteriin ennen salausfunktion tuloksen ja selvätekstilohkon yhdistämistä XOR-operaatiolla (RSA Laboratories, 2000). OFB-moodin toimintaperiaate on esitetty kuvassa 7.



Kuva 7 OFB-moodin toimintaperiaate (Järvinen, 2003).

Menezesin & al. (1997) mukaan OFB-moodia käytetään usein salauksessa, jossa tiedonsiirron yhteydessä mahdollisesti sattunut virhe ei saa levitä koko sanoman salaukseen. Järvinen (2003) mainitsee satelliittiyhteyksien salaamisen yhdeksi tällaiseksi käyttökohteeksi.

Koska OFB-moodissa selväteksti ei vaikuta lainkaan salausfunktion tuottamiin arvoihin (avainjono voidaan laskea jopa ennen varsinaista salausta, jos tiedetään alustusvektori ja

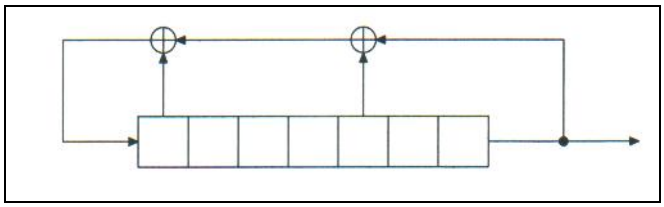
salausavain) niin sekä Menezesin & al. (1997) että Järvisen (2003) mukaan täytyy alustusvektori ehdottomasti vaihtaa, mikäli samaa salausavainta käytetään uudelleen. Vaarana on, että ulkopuolinen osapuoli pystyy tekemään ns. *bitinkääntöhyökkäyksen*, jossa hän yhdistää XOR-operaatiolla kaksi (eri salauskerroilla kaapattua) salatekstiä ja mikäli hän tämän jälkeen arvaa tai tietää toisen selvätekstin, pystyy hän laskemaan toisenkin (Järvinen, 2003).

2.1.3 Jonosalaimet

Jonosalaimet ovat salaimia, joilla voidaan salata esim. yhden bitin tai tavun mittaisia selvätekstin osia kerrallaan (RSA Laboratories, 2000).

Jonosalaimien toimintaperiaate noudattelee Robshawn (1995) mukaan *kertakäyttölehtiö-salausta* (one-time-pad, Vernam-salain). Kyseisessä salauksessa luodaan täysin satunnainen koko selvätekstin mittainen avain ja salataan selväteksti merkki kerrallaan tämän avulla. Shannon (1949) on Robshawn mukaan osoittanut kyseisen menetelmän murtamattomaksi. Murtamattomuuden aiheuttaa se, ettei murtajalla ole yksinkertaisesti mitään mitä salatekstistä voisi analysoida, koska se on salattu täysin satunnaisella koko selvätekstin mittaisella avaimella. Käytännössä tällaista kertakäyttölehtiö-salausta ei avaimen pituuden aiheuttamista luomis- ja hallinnointiongelmista johtuen käytetä missään.

Jonosalaimessa koko selvätekstin mittainen salausavain korvataan satunnaisen kaltaisella *avainvirralla* (keystream), joka tuotetaan *avaingeneraattorilla* (keystream generator). Avainvirta alustetaan salaisella avaimella. Salauksessa avainvirran tuottama bitti ja selvätekstin bitti yhdistetään (yleensä XOR-operaatiolla). Tämän jälkeen avainvirta tuottaa uuden bitin. Avaingeneraattori toteutetaan useimmissa tapauksissa *lineaarisen siirtorekisterin* (linear feedback shift register) avulla. Siirtorekisterin karkea toimintaperiaate on esitetty kuvassa 8. Lineaarisessa siirtorekisterissä seuraavaksi salaukseen käytettävä bitti otetaan ulos toisesta päästä ja toiseen päähän syötetään ”uusi” tilalle. ”Uusi” on itse asiassa ulosotettu bitti hieman muutettuna, esim. yhdistettynä XOR-operaatiolla jonkin rekisterissä jo olevan bitin kanssa. Avainvirrasta ei saada täysin satunnaista, mutta satunnaisuus on riittävää tietokoneille (Menezes & al., 1997). Siirtorekisterin virran- ja muistinkulutus ovat hyvin vähäistä, siksi siirtorekisteriin perustuvat jonosalaimet ovat hyvin suosittuja mm. mikropiireihin toteutetuissa salauksissa (Järvinen, 2003).

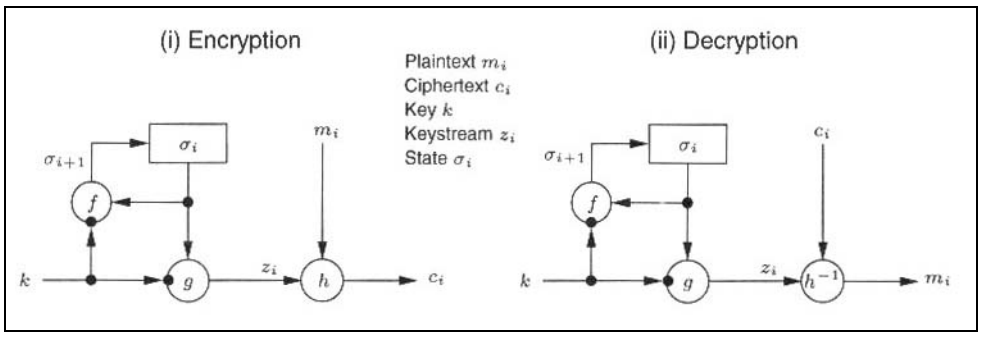


Kuva 8 Siirtorekisterin toimintaperiaate (RSA Laboratories, 2000).

Robshawn (1995) mukaan jonosalaimet luokitellaan sen mukaan käytetäänkö avainvirran seuraavan arvon luomiseen hyväksi selvä- tai salatekstiä vai ei mitään. Jos selvä- tai salatekstiä ei käytetä, puhutaan *synkronisesta* (synchronous) salaimesta. Jos avainvirran seuraavaan tilaan vaikuttaa selvä- tai salateksti, puhutaan *itse-synkronoivasta* (self-synchronizing) tai *epäsynkronisesta* (asynchronous) salaimesta.

Synkronisessa salaimessa yksittäisen bitin salaamiseen ei vaikuta millään lailla muu osa sala- tai selvätekstistä. Tällöin yksittäisen bitin korruptoituminen siirron aikana ei vaikuta purkamiseen kuin juuri kyseisen bitin osalta. Tämä toisaalta myös heikentää virheiden huomaamista purkamisen yhteydessä sekä antaa ulkopuoliselle mahdollisuuden tehdä harkittuja muutoksia salatekstiin, koska hän tietää niiden päätyvän selvätekstiin (Robshaw, 1995). Synkronisen jonosalaimen toimintaperiaate on esitelty kuvassa 9. Kuvan merkkien selitys:

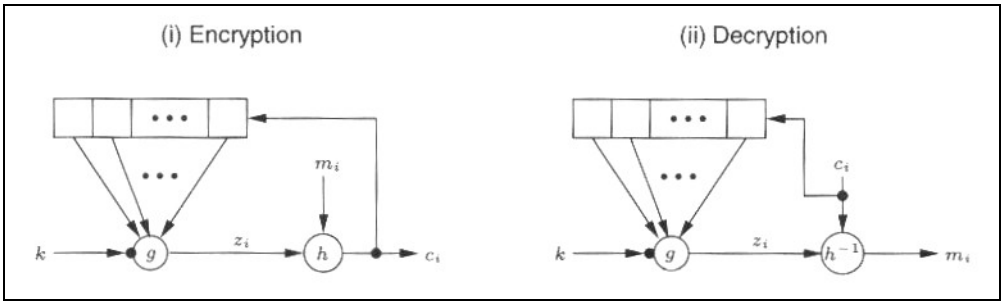
- f = seuraavan tilan asettava funktio. Esimerkiksi rekisterin bittien ja ulostulleen bitin XOR-operaatioita.
- g = avainvirran tuottava funktio. Yksinkertaisimmillaan ulostulevan bitin ja selvätekstin XOR-operaatio, voidaan tehdä monimutkaisemminkin.



Kuva 9 Synkronisen jonosalaimen toimintaperiaate (Menezes & al., 1997).

Menezes & al. (1997) painottavat lisäksi, että synkronisessa salauksessa tulee molempien osapuolien olla ”synkronoituja”, eli purkamisen alkaessa tulee vastaanottajalla olla tiedossa salausavain ja avaingeneraattorin alkutila (sekä tietenkin salateksti). Jos siirron aikana hukataan bittejä, epäonnistuu koko lopputekstin purkaminen, koska salateksti ja avainvirta eivät ole enää synkronoidussa tilassa. Robshawn mukaan tätä riskiä voidaan pienentää lisäämällä salaukseen mukaan *merkkikohtia* (marker points), joiden avulla synkronoitu tila saadaan palautettua.

Epäsynkronisessa salauksessa avainvirran seuraava ulostuleva bitti muodostetaan edellisien salattujen bittien avulla (Menezes & al., 1997). Tällaisessa salauksessa siirron aikana kadonneet tai lisätyt bitit eivät sotke koko loppuviestiä, vaan salain synkronoi itsensä taas vastaanotettuaan riittävän määrän oikeellisia bittejä (Robshaw, 1995). Epäsynkronisen salauksen toimintaperiaate on esitetty kuvassa 10. Kuvassa käytetyt merkinnät ovat vastaavuuksiltaan samoja kuin kuvassa 9.



Kuva 10 Epäsynkronisen jonosalaimen toimintaperiaate (Menezes & al., 1997).

Lohkosalaimista poiketen jonosalaimella salatun selvätekstin toistuvuudet eivät näy salatekstissä, koska kaikki tiedot salataan merkki kerrallaan ja saatu tulos riippuu avaimesta, selvätekstistä ja salausprosessin eli avainvirran vaiheesta. Jonosalain on lisäksi lohkosalainta huomattavasti nopeampi (RSA Laboratories, 2000). Menezes & al. (1997) mainitsevat jonosalaimien käyttökohteiksi mm. tilanteet, joissa tiedonsiirron virheet ovat todennäköisiä (jonosalalaimissa virheet eivät siirry kuin muutaman bitin verran), sekä tilanteet, jolloin on tarvetta pystyä salaamaan ”reaaliaikaisesti” merkki tai tavu kerrallaan. Lohkosalaimissahan täytyy odottaa lohkon täyttymistä (esim. 8 merkkiä) tai täyttää lohkon loppuosa merkityksettömällä datalla, mikä kuluttaa turhaan resursseja.

2.1.4 Avaimen pituus

Järvisen (2003) määritelmän mukaan avaimen arvaaminen on ulkopuoliselle henkilölle ainoa keino avata salattu viesti, ellei hän tiedä avainta, eikä salaimessa itsessään ole muuta oikotietä. Menetelmää, jossa avain koetetaan arvata käymällä järjestelmällisesti läpi kaikki mahdolliset avaimet siten, että jokaista kokeillaan vuorollaan, kutsutaan *brute force* – menetelmäksi. Kaikkien mahdollisten avainten määrä saadaan laskemalla 2^L, missä L = avaimen pituus. Esimerkiksi 10 bittiä pitkällä avaimella olisi eri vaihtoehtoja 2^{10} = 1024 kappaletta. Avaimen pituutta kasvatettaessa vaihtoehtojen määrä kasvaa eksponentiaalisesti, jokainen avaimen pituuteen lisätty bitti kaksinkertaistaa avaimien määrän.

Järvinen (2003) kuvaa tätä kasvua (ei välttämättä kovin tieteellisesti mutta kuitenkin havainnollisesti): Jos 40-bittisen salauksen avainten läpikäymiseen tarvittava laskentateho (aika) vastaa lusikallista hiekkaa, niin määrällisesti 128 bitin salauksen murtamiseen tarvittaisiin hiekkaa noin kolmen maapallon verran. Oikeaa avainta etsittäessä ei tilastomatematiikan lakien mukaan tarvitse käydä kaikkia avaimia läpi, oikea avain löytyy keskimäärin kun puolet avaimista on kokeiltu. Puoliväli saadaan kaavasta 2^{L-1}, missä L on siis avaimen pituus.

Käytettävän avaimen pituutta määritettäessä tulee RSA Laboratoriesin (2000) mukaan ottaa huomioon se tosiseikka, että tietokoneiden laskentatehon kasvaessa muuttuu myös suositeltava avaimen pituuskin suuremmaksi. Tosin ellei mitään mullistavaa tapahdu, ovat nykyisen suosituksen pituiset avaimet turvallisia pitkälle tulevaisuuteen.

Lenstra (2004) määrittelee lohkosalainten turvallisen avaimen pituuden sen avulla kuinka kauan sitä hänen Mooren lakiin perustuvien laskelmiensa mukaan voidaan pitää turvallisena. Lenstran mukaan 90-bittinen avain on turvallinen noin vuoteen 2033 saakka. Salaamalla tiedot 128-bittisillä avaimilla, saavutetaan Lenstran näkemyksen mukaan riittävä turvallisuustaso useiksi vuosisadoiksi. Lenstra huomauttaa lisäksi, että vaikka 128-bittisen avaimen käyttäminen tuntuu tuhlailulta, ei avaimen pituus symmetrisessä salauksessa yleensä juurikaan vaikuta salaimen tehokkuuteen. Reunaehtona tietenkin tällaisessa tulevaisuuden ennustamisessa on se, että tietokoneiden laskentakapasiteetti kasvaa odotetun mukaisesti, eikä mitään radikaaleja uusia murtamiskeinoja keksitä.

ECRYPT (2004) on lähestynyt avaimenpituutta siltä kannalta millaisilla resursseilla ulkopuolisen hyökkääjän oletetaan salausta murtavan. Seuraavat luvut ovat ehdottomia minimilukuja, eli ne kelpaavat vain tiedon lyhytaikaiseen (kuukausia) salaamiseen. Jos asialla on tavallinen hakkeri ”nollabudjetilla”, riittää symmetrisessä salauksessa 51-bittinen salaus. Jos hyökkääjän oletetaan käyttävän zombie-koneiden verkkoa apunaan, on avaimen pituutta nostettava 59 bittiin. Jos oletuksena on, että jonkin valtion tiedustelupalvelu (lähteen mukaan heillä on suurin laskentateho käytettävissään) yrittää murtaa käytetyn salauksen, saavutetaan muutaman kuukauden suoja käyttämällä 81-bittistä avainta. Pitkäaikaiseen säilytykseen ECRYPT suosittaa 128- ja 256-bittisiä salauksia.

Suomen valtio määrittelee omassa tietoturvallisuussuosituksessaan (Valtionhallinnon tietoturvallisuuden johtoryhmä, 2001) symmetrisen salauksen tasoksi korkeimman turvaluokan tiedoille 256 bittiä ja alemman turvaluokan tiedoille sekä muuten yleisesti käytettäväksi (mikäli salausmenetelmänä käytetään symmetristä salausta) 128 bittiä.

Avaimen pituutta mietittäessä on mielessä pidettävä myös Lenstran (2004) huomautus, että useimmiten ulkopuolinen hyökkääjä ei koeta etsiä avainta brute force –tekniikalla, vaan käyttää muita menetelmiä (protokollien heikkoudet, ohjelmien puutteelliset toteutukset, käyttäjien sosiaalinen hakkerointi).

2.1.5 Salaimia

Tässä kohdassa esitellään muutamia yleisesti käytössä olevia lohko- ja jonosalaimia. Pääasiassa keskitytään salaimen turvallisuuteen ja mainitaan muutamia tunnuslukuja, joilla salainta voi verrata muihin. Varsinaisia salaimien toiminta-algoritmeja ei tässä käydä läpi, niiden toimintaperiaatteiden tarkkoja kuvauksia löytyy useista tämän tutkielman viiteteoksista (mm. Menezes & al. (1997)).

DES (Data Encryption Standard) on IBM:n ja NSA:n yhdessä 1970-luvulla kehittämä lohkosalainalgoritmi, joka käyttää 56-bittistä avainta ja jonka lohkokoko on 64 bittiä. Salaimen toiminnan määrittelevä algoritmi *Data Encryption Algorithm* (DEA) on ollut aiemmin Yhdysvaltojen salausstandardi (RSA Laboratories, 2000). Salaimen rakenteesta

on voitu päätellä, että sen kehittäjät ovat mm. tunteneet differentiaalisen kryptoanalyysin[1] jo salainta kehitettäessä (Järvinen 2003).

DES on Stallingsin (2000) mukaan yksi eniten tutkituista salaimista, eikä siitä ole löydetty (ainakaan julkistettuja) heikkouksia. Salaimen kiinteänmittainen ja liian lyhyt avain kuitenkin aiheuttaa sen, että DES:in murtaminen kaikki avaimet kokeilemalla on liian helppoa. Näin ollen sitä ei voida enää käyttää todellista salausta vaativiin toimintoihin. Yhdysvaltain hallitus ei salli enää sen käyttöä virallisessa toiminnassa, korvaavina salaimina tulee käyttää joko TDEA- tai AES-salainta (Barker, 2004).

TDEA (Triple Data Encryption algorithm) eli *3DES* (Stallings, 2000) on DES-salaimesta kehitetty lohkosalain, jota pidetään turvallisena ja soveltuvana nykykäyttöön. Salain käyttää 168-bittistä avainta ja sen toiminta perustuu DES-algoritmin kolminkertaiseen suorittamiseen (DES:in avain = 56 bittiä, 3x56 = 168). Avain jaetaan siis kolmeen osaan (*K1*, *K2* ja *K3*) ja itse salaus tapahtuu salaamalla (*E*) ja purkamalla (*D*) selväteksti (*p*) vuoronperään avaimilla (purkaminen tarkoittaa tässä algoritmin suorittamista purku-suuntaan, ei salauksen poistamista). Salaimen toimintamalli on esitetty kaavassa (1). Toi-mintamalli etenee siten, että selvätekstilohko salataan ensin *K1*-avaimella, jonka jälkeen saatu salateksti ”puretaan” käyttämällä avainta *K2*, ja lopuksi lohko salataan uudelleen käyttämällä avainta *K3* (Barker, 2004).

$$c = E_{K3}(D_{K2}\, E_{K1}(p)) \qquad (1)$$

Barkerin (2004) mukaan avain voidaan muodostaa joko siten, että kaikki kolme avainta ovat erilaisia tai siten, että *K1 = K3*. Mikäli käytetään jälkimmäistä vaihtoehtoa, ei salaimelle taata turvallisuutta kuin seuraavaksi viideksi vuodeksi. Järvinen (2003) mainitsee salaimen huonoiksi puoliksi sen hitauden (lähes kolme kertaa hitaampi kuin DES) sekä lohkokoon, joka on edelleenkin vain 64 bittiä.

AES (Advanced Encryption Standard) lohkosalain kehitettiin NIST:in (National Institute of Standards and Technology) julkistaman kilpailun pohjalta. Kilpailu oli kaikille avoin ja

[1] *Differentiaalinen kryptoanalyysi* on murtamismenetelmä, joka keksittiin julkisesti vuonna 1990. Menetelmä sai paljon huomiota aikaan, kun sen osoitettiin olevan tehokas murtoväline useita tunnettuja salaimia vastaan (Järvinen, 2003).

siihen sai lähettää ehdotuksia uudeksi standardisalaimeksi 15.6.1998 saakka. Ehdokkaiden joukosta karsittiin huonot (hitaat, tietoturvattomat, vaikeasti toteutettavat) salaimet pois, ja jäljelle jääneet (MARS, RC6, Rijndael, Serpent, Twofish) analysoitiin tarkemmin kansainvälisen yhteistyön avulla. Tutkimusten jälkeen asiantuntijat päätyivät valitsemaan Rijndael-salaimen uudeksi standardiksi vuoden 2001 lopussa (Järvinen 2003).

Järvisen (2003) mukaan AES:n lohkon koko on 128 bittiä ja avaimenpituutena voidaan käyttää mitä tahansa 32:n monikertaa, mutta NIST:in standardoimia avainpituuksia ovat kuitenkin vain 128, 192 ja 256. Avaimenpituuden ja riittävän lohkokoon lisäksi Järvinen (2003) listaa muita AES-salaimen hyviä puolia: nopea, kykenevä rinnakkaiseen toimintaan ja kestää kaikki nykyisin tunnetut lineaarisen ja differentiaalisen kryptoanalyysin menetelmät.

Sekä Järvinen (2003) että Stallings (2000) mainitsevat, että uuteen (vaikkakin huolella testattuun ja tutkittuun) salausalgoritmiin tulee aina suhtautua epäillen, sillä salaus-algoritmin todistaminen täysin murtamattomaksi on käytännössä mahdotonta. Niinpä voi hyvinkin olla, että joku tutkija löytää algoritmista lähivuosina virheen tai esim. keksii uuden kryptoanalyyttisen menetelmän, jolle AES on herkkä. Yleisesti ottaen algoritmia siis kuitenkin pidetään turvallisena ja sen käyttöä suositellaan.

Blowfish-lohkosalain on Bruce Schneierin vuonna 1993 kehittämä salain. Blowfish on vähän tila vievä, helposti toteutettavissa oleva ja nopea salain. Avaimen pituus on säädettävissä, pisin mahdollinen avain voi olla 448-bittinen. Salaimen teknisestä toteu-tuksesta johtuen se ei ole kovin sovelias toimintoihin, joissa salaista avainta joudutaan vaihtamaan usein (Stallings, 2000). Blowfish on patentoimaton ja sen lähdekoodi on saatavissa ilmaiseksi (Järvinen 2003).

Twofish on lohkosalain, joka on AES-kilpailuun kehitetty uudempi versio Blowfish-salaimesta. Salain oli kilpailun kolmas eikä siitä ole löydetty vakavia puutteita (Järvinen, 2003). Salaimen lohkokoko on 128 bittiä ja avaimen pituuden voi valita: 128-, 192- tai 256-bittiä (Schneier, 2005).

IDEA (Internal Data Encryption Algorithm) on vuonna 1991 julkaistu ja patentoitu lohkosalain, joka käyttää 128-bittistä avainta. IDEA:sta kaavailtiin aiemmin DES:in

korvaajaa sen avainpituuden takia ja sen toiminta on siksi tarkasti analysoitu. IDEA:n on todettu olevan erittäin vastustuskykyinen kryptoanalyyseille (Stallings, 2000). Järvisen (2003) mukaan salainta pidetään turvallisena, mutta sen huonona puolena on hitaus.

CAST on lohkosalain, jonka avainpituus on joko 128 (CAST-128) tai 256 bittiä (CAST-256). Salainta käytetään salaustoimintoja tarjoavan PGP-ohjelman oletussalaimena versiosta 7.0 alkaen (Järvinen, 2003).

RC4 on RSA:n kehittämä jonosalain. RC4 on turvallinen käytettäessä riittävän pitkää avainta (Järvinen, 2003).

RC5 on RSA:n patentoima lohkosalain vuodelta 1994. Salauslohkon kooksi on mahdollista valita 32, 64 tai 128 bittiä. Avaimen pituudeksi on mahdollista valita 8-2048 bittiä.

SEAL (Software-optimized Encryption Algorithm) on IBM:n patentoima jonosalain, joka on erittäin nopea ja sitä käytetään siksi mm. salatuissa levyasemissa (Järvinen, 2003). Nimensä mukaisesti salain on optimoitu toteutettavaksi ohjelmallisesti (Menezes & al., 1997).

Taulukkoon 1 on koottu edellä mainittujen salaimien tärkeimmät ominaisuudet sekä tieto siitä onko salaimen käyttö suositeltavaa. Turvallisuutta analysoitaessa täytyy muistaa, että turvallinenkin salain voi muuttua turvattomaksi, mikäli käytettävä avain on liian lyhyt.

Taulukko 1 Yhteenveto symmetrisen salauksen salaimista.

Salain (Lohko/Jono)	**Avain (bittiä)**	**Lohkokoko (bittiä)**	**Turvallinen**
DES (L)	56	64	Ei
TDEA (L)	168	64	Kyllä
AES (L)	Kaikki 32:n monikerrat, standardoitu: 128/192/ 256	128	Kyllä
Blowfish (L)	Säädettävissä, max. 448	64	Kyllä
Twofish (L)	128/192/ 256	128	Kyllä
IDEA (L)	128	64	Kyllä
CAST (L)	128/256	64/128	Kyllä
RC4 (J)	1-2048	-	Kyllä
RC5 (L)	0-2040	(32)/64/128	Kyllä
SEAL (J)	160	-	Kyllä

Symmetrinen salaus on siis nopeaa ja tehokasta jo pienilläkin (128 bittiä) avaimen pituuksilla. Lisäksi siihen on tarjolla useita turvallisiksi todettuja salaimia. Mihin siis enää tarvitaan epäsymmetristä salausta, jota käsitellään seuraavassa kohdassa? Symmetrisen salauksen ongelmana on käytettävän avaimen välittäminen osapuolien kesken. Ennen salausta osapuolten pitää kyetä jotenkin kertomaan toisilleen käytettävä salausavain. Tämän tulee tapahtua turvallista kanavaa pitkin. Jos on olemassa tällainen turvallinen ja helppo tapa vaihtaa salainen avain, voidaan Järvisen (2003) mukaan yhtä hyvin kyseistä kanavaa käyttää itse salattavaksi halutun viestinkin välittämiseen. Usein tätä turvallista kanavaa ei kuitenkaan ole jolloin epäsymmetrinen salaus tarjoaa ratkaisun salauksen toteuttamiseen.

2.2 Epäsymmetrinen salaus

Epäsymmetrisen salauksen avulla osapuolet voivat salata kahdenkeskiset viestinsä ilman että heidän tarvitsee välittää toisilleen salainen avain salatun kanavan kautta. Epäsymmetrisellä salauksella on mahdollista välittää esim. symmetrisen salauksen avain salattuna toiselle osapuolelle ilman osapuolien välistä yhteistä salaisuutta. Epäsymmetristä salausta kutsutaan usein myös *julkisen avaimen menetelmäksi*.

Tässä kohdassa esitellään epäsymmetrisen salauksen toimintaperiaate, sekä kuinka sitä hyödyntäen voidaan luoda digitaalisia allekirjoituksia. Myös salauksen osapuolien tunnistamiseksi tarvittavien sertifikaattien eli varmenteiden toimintaa käsitellään hieman. Lopuksi esitellään varsinaisia algoritmeja, joita on kehitetty epäsymmetristä salausta varten. Aivan ensimmäiseksi tarkastellaan kuitenkin tiivistefunktioita, sillä käsite täytyy olla tuttu, kun perehdytään tarkemmin epäsymmetriseen salaukseen.

2.2.1 Tiivistefunktiot

Tiivistefunktio (hash function) on yksisuuntainen funktio, joka ottaa syötteenä vaihtelevanmittaisen bittimäärän ja muodostaa siitä määritellyn mittaisen yksilöllisen bittijonon, ”sormenjäljen” eli *tiivisteen*. Tiivisteitä käytetään mm. viestien/tiedostojen eheyden varmistamiseen ja digitaaliseen allekirjoitukseen (Stallings, 2000).

Tiivistefunktio siis muodostaa syötetiedon avulla tiivisteen, joka on erilainen jokaiselle hiemankin erilaiselle syötteelle. Funktiot pyritään rakentamaan siten, että pienikin muutos syötteessä vaikuttaa paljon tiivisteeseen. Koska funktio on yksisuuntainen, voi toinen osapuoli tiivisteen ja syötetekstin avulla ainoastaan varmistaa, onko tiiviste luotu juuri siitä syötetekstistä, josta sen väitetään olevan (Järvinen, 2003).

RSA Laboratoriesin (2000) mukaan salauksen yhteydessä käytettävältä tiivistefunktiolta vaaditaan seuraavia ominaisuuksia: syötteen vapaa pituus, tuloksen pituuden säätämismahdollisuus, laskennan helppous, funktion yksisuuntaisuus ja törmäyksettömyys. Yksisuuntaisuus merkitsee sitä, ettei tiivisteestä voida järkevässä ajassa laskea funktiolle syötteenä annettua viestiä. Törmäyksettömyys tarkoittaa, ettei löydy kahta erilaista syötettä, jotka tuottavat saman tiivisteen. Tiivisteen etuna salauksessa on, että se voidaan esittää julkisesti ilman pelkoa alkuperäisen viestin paljastumisesta.

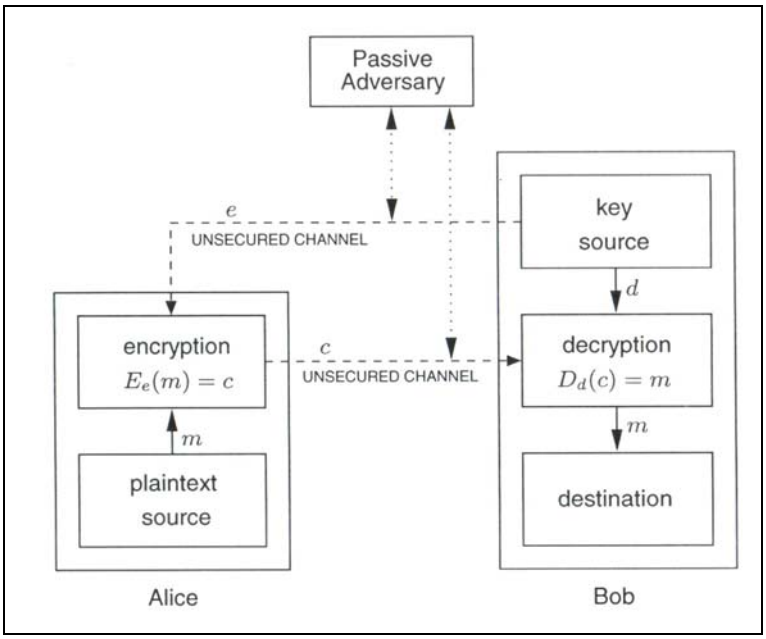
Järvinen (2003) mainitsee nykyisin käytössä oleviksi tiivistefunktioiksi seuraavat: *MD5*, *RIPEMD-160*, *SHA-1*, *SHA-256* ja *SHA-512*. Kolme ensimmäistä tuottavat 160-bittisen tiivisteen ja kaksi jälkimmäistä nimensä mukaisesti joko 256- tai 512-bittisen. Yhdysvaltojen hallinnon käyttämässä standardissa (NIST, 2002) luokitellaan hyväksytyiksi tiivistefunktioiksi vain erilaiset SHA-tiivistefunktion variaatiot: *SHA-1*, *SHA-224*, *SHA-256*, *SHA-384* ja *SHA-512*.

SHA-1 tiivistefunktio on todettu syksyllä 2005 oletettua heikommaksi, kun on tieteellisesti osoitettu, että törmäyksiä voidaan aiheuttaa kohtuullisen pienellä laskennalla (Szydlo, 2005).

2.2.2 Epäsymmetrisen salauksen periaate

RSA Laboratoriesin (2000) mukaan epäsymmetrisessä salauksessa jokaisella osapuolella on hallussaan kaksi avainta: *julkinen avain* (public key) ja *yksityinen avain* (private key). Avaimet ovat matemaattisesti sidoksissa toisiinsa, mikä mahdollistaa sen, että julkisella avaimella salattu viesti voidaan purkaa ainoastaan ko. julkiseen avaimeen liittyvän yksityisen avaimen avulla. Julkinen avain on nimensä mukaan julkinen ja kaikkien tiedossa, vastaanottajalle lähetettävät viestit salataan käyttäen hänen julkista avaintaan. Vastaanottaja purkaa salauksen käyttämällä omaa yksityistä avaintaan, jota ei koskaan

paljasteta muille osapuolille. Kuvassa 11 Menezes & al. (1997) havainnollistavat kuinka epäsymmetrinen salaus toimii periaatteellisella tasolla. Alice saa Bobin julkisen avaimen *e* avointa kanavaa pitkin itselleen ja muodostaa sen avulla salatekstin *c*. Tämän jälkeen Alice lähettää salatekstin avoimen kanavan kautta Bobille, joka purkaa salauksen omalla yksityisellä avaimellaan *d*.



Kuva 11 Epäsymmetrinen salaus (Menezes & al., 1997).

Ulkopuolisen tahon voi siis olla mahdollista saada tietoonsa sekä salateksti että avain, jolla salateksti on muodostettu. Salauksessa käytetään Menezesin & al. (1997) mukaan *yksisuuntaisia salaluukullisia funktioita* (trapdoor one-way function). Salaluukullinen yksisuuntainen funktio on kuten tiivistefunktio, mutta tässä tapauksessa on kuitenkin olemassa ”salaluukku”, jonka avulla päästään takaisin alkutilaan (selvätekstiin). Epäsymmetrisessä salauksessa salaluukkuna toimii vastaanottajan yksityinen avain (Järvinen, 2003).

Koska julkisen ja yksityisen avaimen välillä on matemaattinen yhteys, on ulkopuolisen osapuolen periaatteessa mahdollista laskea julkisen avaimen avulla vastaava yksityinen avain ja purkaa koko salateksti. Avainten välinen matemaattinen yhteys (epäsymmetrisen salauksen teho) perustuu sellaisiin matemaattisiin ongelmiin, joiden ratkaiseminen nykymittapuun mukaan on ylivoimaista kohtuullisessa ajassa. Matemaattisena ongelmana

epäsymmetrisessä salauksessa käytetään mm. suurten lukujen tekijöihin jakoa ja diskreettiä logaritmia (Järvinen, 2003).

Mitä pidempiä avaimia salauksessa käytetään, sitä hitaammin toimivat tekijöiden jakamiseen ja diskreetin logaritmin laskemiseen keksityt algoritmit. Algoritmeista tällä hetkellä tehokkaimmaksi uskotun mukaan määritellään myöskin salauksessa käytetyn avaimen pituus (RSA Laboratories, 2000). Mielenkiintoiseksi asian tekee se, että RSA Laboratoriesin (2000) mukaan voidaan vain todeta, että ei ole tiedossa algoritmeja, joilla epäsymmetrisessä salauksessa käytetyt matemaattiset ongelmat voitaisiin ratkaista nopeammin, mutta ei ole toisaalta myöskään voitu tieteellisesti osoittaa, ettei sellaisia ratkaisumenetelmiä ole olemassa.

2.2.3 Digitaalinen allekirjoitus

Varsinaisen salauksen lisäksi epäsymmetristä salausta voidaan käyttää myös digitaalisten allekirjoitusten tekemiseen.

RSA Laboratoriesin (2000) mukaan *digitaalinen allekirjoitus* perustuu siihen, että epäsymmetrinen salaus voidaan tehdä myös ”väärinpäin” eli salaamalla haluttu bittijono yksityisellä avaimella. Tällöin salaus voidaan purkaa ainoastaan kyseiseen yksityiseen avaimeen liittyvällä julkisella avaimella. Tämä ei tietenkään salaa mitään (koska julkinen avain on kaikkien tiedossa), mutta se osoittaa, että allekirjoituksen on tehnyt taho, joka omistaa tietyn yksityisen avaimen. Käytännössä allekirjoitus tehdään siten, että lähettäjä ajaa lähetettävän tekstin tiivistefunktion läpi ja salaa saadun tiivisteen omalla yksityisellä avaimellaan. Tämän jälkeen lähettäjä lähettää vastaanottajalle sekä salatun viestin että allekirjoituksen. Vastaanottaja purkaa omalla yksityisellä avaimellaan varsinaisen salatekstin selvätekstiksi ja laskee siitä tiivisteen. Varmistaakseen viestin aitouden vastaanottaja purkaa mukana tulleen allekirjoituksen lähettäjän julkisella avaimella ja vertaa allekirjoituksessa olevaa tiivistettä laskemaansa tiivisteeseen. Mikäli tiivisteet eivät täsmää, ei viestin allekirjoittajaa voida todentaa, tai viesti on muuttunut matkalla.

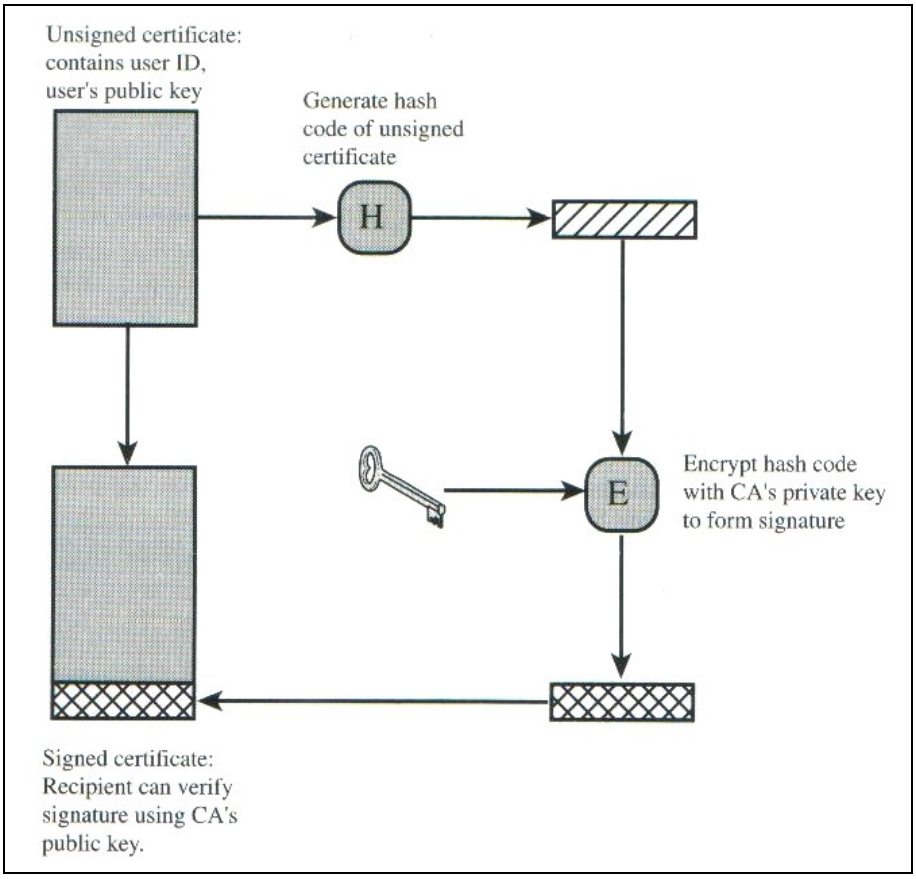
Mistä sitten tiedetään, mikä on varmuudella kenenkin julkinen avain? Järvisen (2003) mukaan täytyisi jollain luotettavalla tavalla pystyä toteamaan, että tietty julkinen avain kuuluu juuri sille taholle, jolle sen väitetään kuuluvan. Jos nimittäin ensimmäisen kerran

julkisia avaimia vaihdettaessa ei pystytä varmistumaan, ettei sanomaliikennettä kuuntele kukaan ulkopuolinen, voi pahimmassa tapauksessa käydä niin, että osapuolien välissä onkin ulkopuolinen taho, jonka kautta kaikki (salaamaton) liikenne kulkee. Tällöin ulkopuolinen taho voi kaapata osapuolten toisilleen lähettämät julkiset avaimet ja korvata ne omillaan. Tällaisessa tapauksessa viestiliikenteen osapuolet siis salaavat lähettämänsä viestit julkisilla avaimilla, joiden yksityiset avaimet ovat itse asiassa tämän ulkopuolisen tahon hallinnassa. Vaihdettuaan avaimet haluamikseen ulkopuolinen taho kaappaa sala-tutkin viestit, avaa ne omilla avaimillaan ja salaa tämän jälkeen oikealla (vastaanottajan) julkisella avaimella ja lähettää eteenpäin viestin alkuperäiselle vastaanottajalle. Tällaista hyökkäystä kutsutaan *mies-keskellä* –hyökkäykseksi (man-in-the-middle attack). RSA Laboratoriesin (2000) mukaan tällainen tilanne voidaan estää käyttämällä sertifikaatteja, joita käsitellään seuraavassa kohdassa.

2.2.4 Sertifikaatit

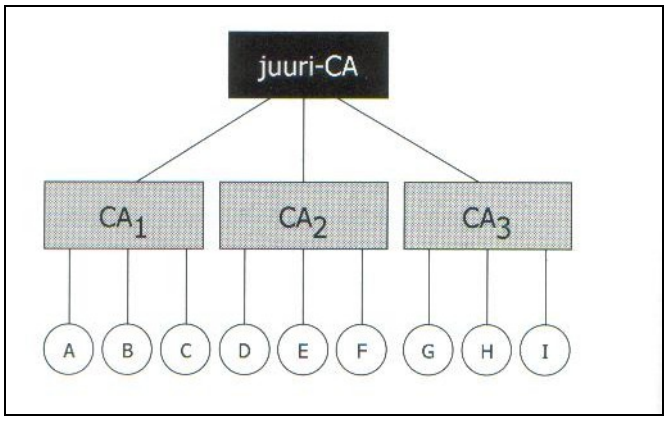
Sertifikaatti eli *varmenne* on digitaalinen dokumentti, jonka avulla voidaan varmasti todeta tietyn julkisen avaimen kuuluvan juuri tietylle henkilölle/tietokoneelle (RSA Laboratories, 2000). Sertifikaatti sisältää tietoja omistajasta, mm. julkisen avaimen, nimen ja voimassaoloajan. Sertifikaatti on digitaalisesti allekirjoitettu *varmenteiden myöntäjän* (Certification Authority, CA) toimesta (Järvinen 2003). CA:t ovat julkisia, yleisesti tunnettuja ja luotettuja tahoja, joiden julkiset avaimet ovat yleisessä tiedossa ja toimitettu turvallisia reittejä käyttäen kaikille osapuolille (RSA Laboratories, 2000). Esimerkiksi uudemmissa Windows-käyttöjärjestelmissä on tällaisia *juuri-varmenteita* valmiina asennettuna.

Kuvassa 12 Stallings (2000) kuvaa varmenteen luomisprosessia: käyttäjä muodostaa sertifikaattitiedoston (tai vaihtoehtoisesti CA luo senkin) ja CA allekirjoittaa sen omalla yksityisellä avaimellaan. Ennen allekirjoitusta sertifikaatin hakijan täytyy todentaa itsensä CA:lle, tämä saattaa myöntäjästä riippuen vaatia vaikkapa fyysisen käynnin paikan päällä (Järvinen, 2003).



Kuva 12 Sertifikaatin muodostaminen (Stallings, 2000).

Osaltaan hakijoiden luotettavan tunnistamisen helpottamiseksi sertifikaattien myöntäjien organisaatio on yleensä hierarkkinen (kuva 13) (Järvinen 2003). Tällöin ylin taso antaa valituille alemman tason yksiköille oikeuden varmentaa sertifikaatteja itsenäisesti. Ylin taso siis varmentaa alemman tason, joka varmentaa varsinaiset käyttäjien sertifikaatit.



Kuva 13 Varmenteiden myöntäjien hierarkkinen organisaatiomalli (Järvinen, 2003).

Kun käyttäjällä on voimassaoleva sertifikaatti hallussaan, hän salaa viestinsä, kuten normaalistikin julkisen avaimen menetelmissä. Viestin mukana hän lähettää sertifikaatin, joka on CA:n allekirjoittama todistus lähettäjän julkisesta avaimesta. Mukana voi toisinaan mennä useitakin sertifikaatteja, joista ensimmäinen varmentaa lähettäjän, seuraava lähettäjän varmentaneen sertifikaatin ja seuraava taas tämän sertifikaatin jne. Polkua seurataan ylöspäin niin kauan, että löytyy taho, johon vastaanottaja voi luottaa täysin (jonka julkisesta avaimesta hän on varma). Vastaanottaja purkaa sertifikaatin allekirjoituksen CA:n julkisella avaimella ja varmistuttuaan näin lähettäjän henkilöllisyydestä purkaa myös itse viestin oman yksityisen avaimensa avulla. Vastaanottajalla tulee siis olla hallussaan CA:n sertifikaatti tai ainakin sellaisen CA:n sertifikaatti, joka on allekirjoittanut jonkun sertifikaatin mukana tulleessa sertifikaattipolussa (Järvinen, 2003).

2.2.5 Algoritmeja

Epäsymmetrisen salauksen algoritmeja on tarjolla vähemmän kuin symmetriseen salaukseen, lisäksi osa algoritmeista soveltuu vain tutkimuskäyttöön. Seuraavissa kohdissa käydään läpi muutamia sellaisia algoritmeja, joita käytetään todellisissa sovelluksissa. Algoritmit ja niiden yleistiedot on esitetty taulukossa 2. Taulukossa esitetyt avaimen pituudet ovat suosituksia (NIST, 2005), jotka pätevät tiedolle, jonka täytyy säilyä salattuna vuoteen 2010 saakka.

Taulukko 2 Epäsymmetrisiä algoritmeja, avainpituudet NIST:in (2005) suosituksia.

Algoritmi	**Päätoiminnot**	**Suositeltava avaimen pituus bitteinä**
Diffie-Hellman	Avaimen vaihtaminen	1024
RSA	Salaus, digitaalinen allekirjoitus	1024
ElGamal	Salaus, digitaalinen allekirjoitus	1024
DSA	Digitaalinen allekirjoitus	1024
Elliptiset käyrät	Salaus, digitaalinen allekirjoitus	160

2.2.5.1 Diffie-Hellman

Ensimmäinen julkisen avaimen periaatteella toimiva algoritmi oli Diffien ja Hellmanin vuonna 1976 julkaisema *Diffie-Hellmanin avaintenvaihtoprotokolla*. Nimensä mukaisesti

sen avulla osapuolet voivat vaihtaa salausavaimen turvallisesti turvattoman kanavan ylitse, mutta eivät suorittaa varsinaista salausta (Stallings, 2000).

RSA Laboratoriesin (2000) mukaan algoritmin turvallisuus perustuu diskreetin logaritmin laskemisen vaikeuteen (hitauteen). Mikäli jonain päivänä keksitään erittäin tehokas tapa diskreetin logaritmin laskemiseksi, merkitsee se Diffie-Hellman algoritmin (kuten monen muunkin) käytön loppumista.

Algoritmin toimintaperiaate on melko yksinkertainen, kaikki tarvittava tietoliikenne voidaan toteuttaa salaamattoman yhteyden ylitse (RSA Laboratories, 2000):

1. Osapuolet sopivat yhteisestä suuresta alkuluvusta *p*.
2. Osapuolet sopivat yhteisestä luvusta *g*.
3. Osapuoli A generoi satunnaisen yksityisen luvun *a*.
4. Osapuoli B generoi satunnaisen yksityisen luvun *b*.
5. Osapuoli A luo julkisen lukunsa, $PA = g^a \bmod p$.
6. Osapuoli B luo julkisen lukunsa, $PB = g^b \bmod p$.
7. Osapuolet vaihtavat julkiset lukunsa keskenään.
8. Osapuoli A laskee yhteisen avaimen $k = PB^a \bmod p$ (eli oikeastaan $(g^b)^a \bmod p$)
9. Osapuoli B laskee yhteisen avaimen $k = PA^b \bmod p$ (eli oikeastaan $(g^a)^b \bmod p$)

Lukujen täytyy täyttää lisäksi seuraavat ehdot:

- *p*:n pitää olla alkuluku
- *g*:n tulee täyttää ehto: jokaiselle luvulle *n* välillä [1, *p*-1] on olemassa *g*:n potenssi *k* siten että: $n = g^k \bmod p$
- $1 < a$ ja $b < p-2$

Askeleet 8 ja 9 ovat totta koska: $g^{ab} = g^{ba}$. Mikäli joku ulkopuolinen kuuntelee tietoliikennettä, hän saa haltuunsa luvut *p*, *g* , *PA* ja *PB*. Ellei julkisen luvun laskennassa olisi käytetty moduulia, ulkopuolisen olisi helppoa laskea tavallisen logaritmin avulla mihin potensseihin luku *g* on korotettu, että on saatu *PA* ja *PB*. Moduulin käyttö laskutoi-

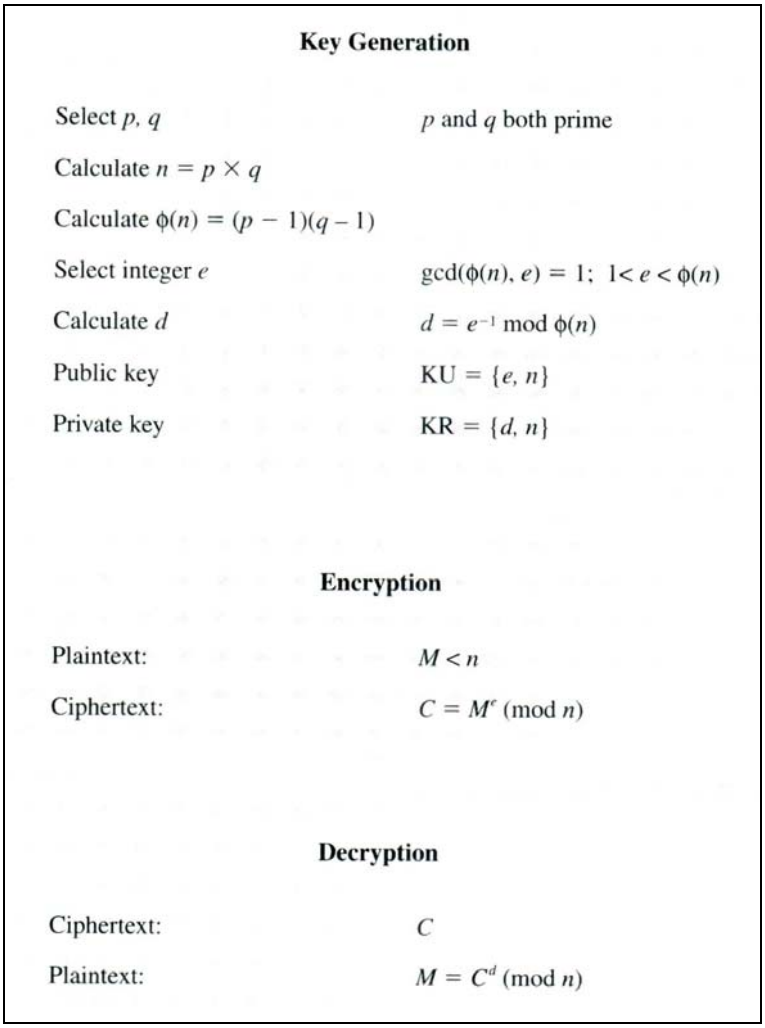
mituksissa aiheuttaa kuitenkin sen, että potenssin selvittämiseen pitää käyttää diskreettiä logaritmia (Järvinen, 2003).

Algoritmin heikkoutena on, ettei se todenna osapuolia, mikä mahdollistaa mies-keskellä –hyökkäyksen käyttämisen algoritmia vastaan (RSA Laboratories, 2000).

2.2.5.2 RSA

RSA on julkisen avaimen salausmenetelmä, jonka kehittivät Ronald Rivest, Adi Shamir ja Leonard Adleman vuonna 1977. RSA:lla voidaan sekä salata että luoda digitaalisia allekirjoituksia. Sen turvallisuus perustuu suurten lukujen tekijöihin jakamisen vaikeuteen (RSA Laboratories, 2000).

Stallingsin (2000) mukaan iso osa julkisen avaimen salausmenetelmiä käyttävistä järjestelmistä käyttää RSA:ta. Kuvassa 14 käydään vaiheittain läpi yksinkertaistetusti RSA-algoritmin toiminta. Ensin valitaan kaksi suurta alkulukua (p, q), jotka kerrotaan keskenään, näin saadaan *modulus* n. Modulusta käytetään sekä salaukseen että purkamiseen. Tämän jälkeen valitaan luku e, josta tulee julkisen avaimen eksponentti. Eksponentti tulee valita siten, että sillä ja tulolla $(p-1)(q-1)$ ei ole muita yhteisiä tekijöitä kuin 1. Julkisen eksponentin selvittyä lasketaan vielä yksityisen avaimen eksponentti d, kyseisen luvun täytyy olla e:n käänteisluku modulo $(p-1)(q-1)$. Järvisen (2003) mukaan modulaariaritmetiikassa luvut ovat käänteislukuja keskenään mikäli niiden tulo annetun moduluksen suhteen on yksi. Edelleen Stallings (2000) esittää, että näiden lukujen määrittelyn jälkeen voidaan todeta julkisen avaimen (e, n) ja yksityisen avaimen (d, n) olevan valmiita. Kuten kuvassa 14 esitetään, salaus tapahtuu korottamalla salateksti (numeerisessa muodossa) potenssiin e ja laskemalla tuloksesta moduuli moduluksen (n) suhteen. Purkaminen tapahtuu vastaavasti korottamalla salateksti yksityisen avaimen eksponenttiin (e) ja laskemalla tuloksen moduuli moduluksen (n) suhteen.



Key Generation

Select p, q	p and q both prime
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Kuva 14 RSA-algoritmi (Stallings, 2000).

RSA-algoritmin murtamiseksi voidaan käyttää brute force –menetelmää, mutta se ei ole järkevää, sillä yleisesti RSA:ssa käytetyt avaimet ovat vähintäänkin 512-bittisiä. Toisaalta julkisen avaimen menetelmän periaatteesta johtuen voidaan keskittyä salauksen perustana olevan matemaattisen ongelman ratkaisemiseen (Järvinen, 2003). RSA Laboratoriesin (2000) mukaan RSA-algoritmi voidaan periaatteessa murtaa selvittämällä lausekkeesta $C = M^e \pmod{n}$ diskreetin logaritmin avulla selväteksti (jolloin varsinainen yksityinen avain kuitenkin säilyy tuntemattomana hyökkääjälle). Tämä ei kuitenkaan nykyisen tietämyksen mukaan ole mahdollista järkevässä ajassa.

Toinen (ja todennäköisempi) uhka salauksen murtamiseksi on RSA Laboratoriesin (2000) mukaan moduluksen tekijöihin jako. Julkisesti esitetyistä luvuista *n* ja *e* voidaan teoriassa laskea tietyn avainparin yksityisen avaimen eksponentti *d* jakamalla modulus *n* tekijöihinsä

(*p* ja *q*). Tällöin hyökkääjä saa tietoonsa myös yksityisen avaimen ja voi näin ollen purkaa toiselle henkilölle osoitetut salatekstit sekä tehdä digitaalisia allekirjoituksia tämän nimissä. Nykyisen yleisesti hyväksytyn tietämyksen mukaan suuren moduluksen tekijöihin jakaminen ei ole mahdollista järkevässä ajassa.

Moduluksen tekijöihin jakamisen estämiseksi, täytyy käytetyt alkuluvut *p* ja *q* olla riittävän pitkiä/suuria (mitä pidempi modulus, sitä kauemmin sen tekijöihin jako kestää). Moduluksen pituus määräytyy *p*:n ja *q*:n mukaan. Jos ne ovat esim. 512-bittisiä, on modulus vastaavasti 1024-bittinen, mikä kymmenjärjestelmässä tarkoittaa 309:n numeron pituista lukua (Järvinen, 2003). Lenstra (2004) on artikkelissaan määritellyt seuraavanlaiset vähimmäisvaatimukset RSA-algoritmissa käytettävälle modulukselle: Jos tiedon pitää pysyä salattuna vuoteen 2010 saakka, tulee moduluksen olla vähintään 1112-bittinen (NIST:in (2005) mukaan jo 1024 bittiä riittää). Käyttämällä 4096-bittistä modulusta tieto säilyy salaisena vuoteen 2060 asti, ja mikäli tiedon halutaan säilyvän vuoteen 2100 saakka salattuna, tulisi avaimen olla 8192 bittinen. Eli epäsymmetrisessä salauksessa laskentatehon kasvaessa avaimen pituutta täytyy kasvattaa nopeammin kuin symmetrisessä salauksessa.

Johtuen mm. suurilla avainpituuksilla tapahtuvan laskennan vaikeudesta, on esim. RSA huomattavasti hitaampi salauksessa kuin useimmat symmetriset lohkosalaimet (ohjelmallisesti toteutettuna jopa noin 100 kertaa hitaampaa). Tämän takia RSA:ta käytetäänkin yleensä siten, että itse viesti salataan symmetrisellä salaimella ja RSA:n avulla salataan vain symmetrisen salaimen salainen avain. Vastaanottajalle lähetetään (mahdollisesti digitaalisella allekirjoituksella varustettu) salattu viesti ja salattu avain. Vastaanottaja purkaa ensin omalla yksityisellä avaimellaan salaisen avaimen selkokieliseksi ja purkaa tämän jälkeen itse viestin salaisen avaimen avulla (RSA Laboratories, 2000).

2.2.5.3 ElGamal

ElGamal algoritmi perustuu Diffie-Hellmanin avaimenvaihtoalgoritmiin, eli sen salauksen teho on riippuvainen diskreetin logaritmin laskemisen vaikeudesta. ElGamal-algoritmilla voidaan sekä salata että allekirjoittaa viestejä, algoritmi on julkaistu vuonna 1985 (RSA Laboratories, 2000).

RSA Laboratoriesin (2000) mukaan ElGamal ja RSA ovat samoilla avainpituuksilla yhtä turvallisia, joten ElGamalia käytettäessä voidaan valita avainpituudeksi kulloinkin suositeltu RSA:n pituus. Vaikkakin ElGamal ja RSA ovat yhtä turvallisia samalla avainpituudella, on ElGamalissa muutamia ongelmia: ElGamalin toimintaan tarvitaan satunnaisuutta (mikä on vaikeaa toteuttaa tietokoneilla), lisäksi algoritmi itsessään on hidas erityisesti digitaalisen allekirjoituksen tekemisessä (RSA Laboratories, 2000).

2.2.5.4 DSA

DSA eli *Digital Signature Algorithm* on USA:n standardoimisviranomaisten julkaisema algoritmi. Algoritmi julkaistiin digitaalista allekirjoitusta koskevan standardin yhteydessä, joka on nimeltään DSS *Digital Signature Standard* (RSA Laboratories, 2000).

RSA Laboratoriesin (2000) mukaan DSA:lla voidaan ainoastaan tehdä digitaalisia allekirjoituksia, ei suorittaa salausta. Koska algoritmi on suunniteltu nimenomaan allekirjoitukseen, on sillä allekirjoituksen muodostaminen hitaampaa kuin allekirjoituksen tarkastaminen (toisin kuin RSA:ssa). Tämä johtuu siitä, että useimmiten asiakirja tms. allekirjoitetaan kerran ja sen jälkeen allekirjoituksen oikeellisuus joudutaan mahdollisesti tarkastamaan usein.

Algoritmin turvallisuus perustuu diskreetin logaritmin ongelmaan, eikä siitä ole toistaiseksi löytynyt vakavia puutteita. Algoritmi onkin levinnyt laajaan käyttöön. Jonkin verran kritiikkiä on aiheuttanut se, että koko standardi on USA:n viranomaisten tekemä ja varsinkin se, että NSA (National Security Agency) on ollut mukana standardin kehittämisessä (RSA Laboratories, 2000).

2.2.5.5 Elliptiset käyrät

Tietokoneiden laskentatehon kasvaessa on RSA-salausalgoritmia käyttävissä järjestelmissä jouduttu avaimen pituutta kasvattamaan vähän kerrallaan. Koska laskutoimitukset ovat vaikeita ja luvut suuria, vie RSA-salauksen käyttö aikaa ja laskentatehoa. Viime aikoina onkin suosiotaan kasvattanut julkisen avaimen salausmenetelmä, joka perustuu matemaattisiin *elliptisiin käyriin* (elliptic curves), koska siinä voidaan käyttää huomattavasti lyhyempää avainta kuin RSA:ssa (Stallings, 2000). Elliptisiin käyriin

perustuvissa salausmenetelmissä käytettävän kulloinkin turvalliseksi oletetun avaimenpituuden voi karkeasti arvioida kertomalla kyseisellä ajanhetkellä turvalliseksi arvioidun symmetristen salaimien avainpituuden kahdella (SSH Communications Security, 2005). RSA Laboratoriesin (2000) mukaan 160-bittinen avain elliptisen käyrän menetelmässä vastaa suurin piirtein 1024-bittistä avainta RSA:ssa.

Elliptisiä käyriä käytettiin salauksessa ensimmäisen kerran 1980-luvulla, ja niiden tehokkuus perustuu elliptisen käyrän diskreetin logaritmin ongelmaan (RSA Laboratories, 2000). RSA Laboratoriesin mukaan lyhyemmät avainpituudet mahdollistuvat sen takia, että elliptisen käyrän diskreetin logaritmin ratkaisemiseksi tiedetyt algoritmit ovat huomattavasti hitaampia kuin normaalit diskreetin logaritmin laskemiseen tai tekijöihin jakamiseen tarkoitetut algoritmit. Johtuen menetelmän nuoresta iästä ja tuntemattomuudesta, sen käyttö ei ole kuitenkaan yleistynyt kuin vasta viime vuosina. Menetelmästä ei ole toistaiseksi löydetty vakavia puutteita. Mikäli laajasti käytössä olevasta RSA:sta joudutaan joskus luopumaan, korvataan se Järvisen (2003) mukaan todennäköisesti elliptisiin käyriin perustuvalla salausalgoritmilla.

3 JAVAN JA ORACLEN SALAUSTOIMINNOT

Luvussa 2 käsiteltiin salausmenetelmien teoreettista taustaa. Seuraavaksi esitellään muutamia todellisia tuotteita ja niiden tarjoamia vaihtoehtoja salauksen toteuttamiseksi. Tämän tutkielman yhtenä tavoitteena on toteuttaa tietoliikenteen salaus Java-sovelluksen ja sen tietovarastonaan käyttämän Oracle-tietokannan välille. Tämän vuoksi seuraavissa kohdissa keskitytään esittelemään näiden kahden tuotteen tietoliikenteen salaus-ominaisuuksia.

3.1 Java-kielen mahdollistama salaus

Toteuttaakseen salausoperaatioita Java-ohjelmassa, ohjelmoijan ei tarvitse osata ohjelmallisesti toteuttaa luvussa 2 esiteltyjä matemaattisia ja monimutkaisia algoritmeja. Weissin (2004) mukaan onkin oleellisempaa ymmärtää millaista salausta pitää missäkin tilanteessa käyttää, Javan valmiit kirjastot hoitavat itsenäisesti varsinaisen salauksen toteutuksen.

Weissin (2004) mukaan Javassa salaus tapahtuu käyttämällä *salauskirjastoja*. Salauskirjasto sisältää sellaisia luokkia, joiden avulla voidaan suorittaa kryptografisia toimenpiteitä. Tällaisiksi toimenpiteiksi Weiss mainitsee mm. tiivisteiden luomisen, viestien autentikoinnin tiivisteiden avulla, digitaaliset allekirjoitukset, kryptografisesti turvallisten satunnaislukujen luomisen, salaisten avaimien generoimisen ja säilytyksen, avaimen vaihtamisen osapuolien kesken sekä salauksen ja salauksen purkamisen.

Aiemmista USA:n vientirajoituksista johtuen Java sisältää kaksi erillistä salauskirjastoa: *Java Cryptography Architecture* (JCA) ja *Java Cryptography Extensions* (JCE). Edistyneempien salausmenetelmien käytön mahdollistava Java Cryptography Extension on vapautunut vientisäännöstelystä vasta Javan versioon 1.4 (Weiss, 2004).

Java-järjestelmällä on mahdollista suorittaa kryptografisia toimenpiteitä vasta, kun siitä löytyy salauskirjastojen lisäksi ainakin yksi *toimittaja* (provider). Toimittaja on taho, joka on toteuttanut salauskirjastojen määrittelemät toiminnot, eli koodannut matemaattiset algoritmit tietokoneen ymmärtämään muotoon. Oletuksena Javassa tulee mukana vain Sunin toteuttamia toimittajia, mutta siihen voidaan liittää muita, joko maksullisia

(kaupallisia) tai maksuttomia (vapaan lähdekoodin) toimittajia (Weiss, 2004). Salauskirjastojen ja toimittajien ideologia toimii Javassa siten, että ellei ohjelmoija määrittele tarkasti kenen toimittajan toteuttamana hän haluaa esimerkiksi digitaalisen allekirjoituksen, Java valitsee sen itse hakemalla toimittajalistauksesta ensimmäisen sellaisen asennetun toimittajan, joka pystyy tarjoamaan digitaalisen allekirjoituksen toteutuksen. Toisena ääripäänä ohjelmoija voi määritellä tarkasti, minkä toimittajan toteutuksella operaatio tehdään (Weiss, 2004). Ohessa on kahden koodirivin esimerkki molemmista (BC = Bouncy Castle, ilmainen toimittaja):

```
MessageDigest md1 = MessageDigest.getInstance(”SHA-1“);
MessageDigest md2 = MessageDigest.getInstance(”SHA-1“, ”BC”);
```

JCA tarjoaa taulukon 3 mukaiset kryptografiset luokat.

Taulukko 3 JCA:n tarjoamat toiminnot (Weiss, 2004).

Luokka (Engine)	**Toiminto**
MessageDigest	Tuottaa tiivisteen viestistä.
Signature	Tuottaa digitaalisen allekirjoituksen dokumentista.
KeyPairGenerator	Tuottaa avainparin, jota tarvitaan mm. digitaalisen allekirjoituksen luomiseen.
KeyFactory	Muodostaa avaimen saamiensa parametrien perusteella.
KeyStore	Säilyttää ja hallinnoi salaisia avaimia ja avainpareja.
SecureRandom	Luo kryptografisesti turvallisia satunnaislukuja.
AlgorithmParameters	Sisältää tietyn algoritmin tarvitseman parametrijoukon.
AlgorithmParameterGenerator	Muodostaa tietyn algoritmin tarvitseman parametrijoukon.
CertificateFactory	Luo sertifikaatteja.
CertPathBuilder	Luo tunnistusketjuja sertifikaattien välille.
CertStore	Säilyttää ja hallinnoi sertifikaatteja.

Kuten taulukosta 3 voidaan huomata, JCA:n avulla voidaan toteuttaa esim. digitaalisia allekirjoituksia ja hallinnoida niissä vaadittavia avaimia. JCA ei kuitenkaan tarjoa toteutusta varsinaisiin salausoperaatioihin. Tähän tarvitaan JCE:tä, joka on siis tullut mukaan vasta Javan version 1.4 myötä. Tämän salauskirjaston avulla Javalla voidaan suorittaa edistyneempiä salausoperaatioita. Taulukossa 4 on esitetty tärkeimmät JCE:n luokat.

Taulukko 4 JCE:n tarjoamat toiminnot (Weiss, 2004).

Luokka (Engine)	Toiminto
Cipher	Salaaminen ja purkaminen.
KeyGenerator	Tuottaa avaimia salaimille.
SecretKeyFactory	Muodostaa avaimen saamiensa parametrien perusteella.
KeyAgreement	Muodostaa dynaamisesti useamman tahon kesken yhteisen salaisuuden.
Mac	Luo tiivisteen avulla dokumentin yksilöivän koodin (*Message Authentication Code*).

Taulukoissa 3 ja 4 mainittujen kirjastoluokkien käyttö on hyvin yksinkertaista. Kuten aiemmin mainittiin, oleellisempaa onkin tietää, milloin mitäkin menetelmää pitää käyttää, ja kuinka se tehdään turvallisesti. Weiss (2004) havainnollistaa salauskirjastojen käyttöä seuraavalla koodiesimerkillä (kuva 15). Esimerkistä on jätetty pois mm. poikkeustenkäsittely selvyyden takia.

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

...

        KeyGenerator kg = KeyGenerator.getInstance("DES");
        SecretKey key = kg.generateKey();
        SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "DES");
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE, keySpec);
        String plainText = "This is a secret message";
        byte[] cipherText = cipher.doFinal(plainText.getBytes());
```

Kuva 15 Pelkistetty esimerkki merkkijonon salauksesta Javalla (Weiss, 2004).

Aluksi kuvan 15 koodissa luodaan KeyGenerator-ilmentymä *kg*, jonka avulla saadaan tehtyä salauksessa tarvittava salainen avain *key*. KeyGenerator-luokalle täytyy jo sen luontivaiheessa kertoa millainen avain siltä tullaan pyytämään. Tämän jälkeen avaimen avulla luodaan *avainspesifikaatio* (keyspecification), jonka avulla voidaan mm. varmistaa, ettei Keygenerator ole luonut heikkoa avainta[2]. Seuraavaksi luodaan itse salain *cipher*, joka alustetaan salaus-moodiin. Varsinainen tiedon salaus tapahtuu doFinal() –metodilla, jolle annetaan parametrina salattava teksti (Weiss, 2004).

[2] *Heikot avaimet* (weak keys) ovat avaimia, joilla suoritettu salaus on helppo murtaa. Heikkoja avaimia on ajan mittaan löydetty lähes kaikista yleisistä salaimista. Yleisessä käytössä olevien salaimien heikkojen avainten määrä on kuitenkin niin pieni osa avainavaruutta, että ne on helppo valikoida pois avainta luotaessa (RSA Laboratories, 2000).

Kuvan 15 esimerkissä suoritettiin merkkijonon salaus symmetrisellä salaimella käyttäen vanhentunutta DES-algoritmia, mutta algoritmi olisi myös voinut olla vaikkapa AES tai Blowfish. Salausmenetelmäkin olisi yhtälailla voinut olla epäsymmetrinen. Vaikka esimerkissä salattiin vain yksi merkkijono, taulukoiden 3 ja 4 salauskirjastojen avulla voidaan avaintenvaihtoprotokollia, epäsymmetristä ja symmetristä salausta käyttämällä salata myös tietoverkon tietoliikenne.

Tietoliikenteen salauksessa kryptografisten operaatioiden määrä kasvaa, jolloin kasvaa myös vaadittavien koodirivien määrä. Java-sovelluksen tietoliikenteen salaus voidaan kuitenkin toteuttaa vielä yksinkertaisemmin, ilman salauskirjastojen suoraa käyttöä. Tällöin hyödynnetään tietokannan ja Java-sovelluksen välistä tietoliikennerajapintaa. Tähän ratkaisuun perehdytään seuraavissa kohdissa Oracle-tietokannan osalta.

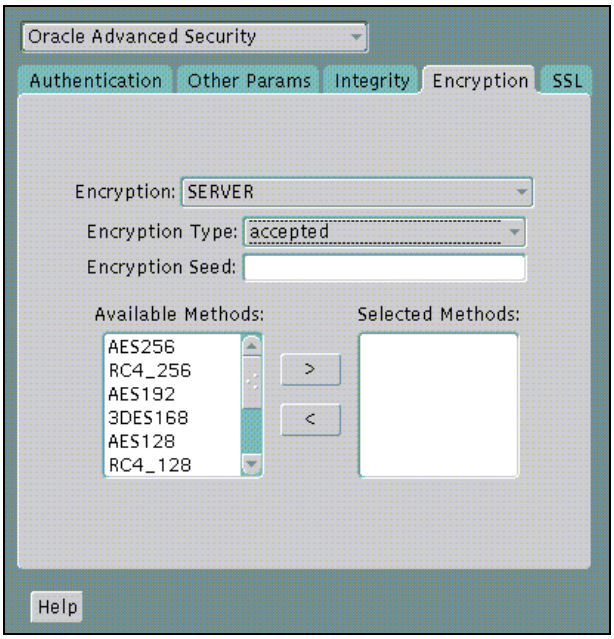
3.2 Oracle-tietokannan mahdollistama salaus

Oracle-tietokannan tarjoamilla työkaluilla voidaan suorittaa tiedonsalaustoimenpiteitä ainakin kahdella tasolla: salaamalla itse tietokannan tauluissa olevat rivitiedot, sekä salaamalla tietoliikenne tietokannan ja sitä käyttävän asiakasohjelman välillä. Tietokannan rivitiedon salaaminen voi tulla kyseeseen, mikäli halutaan varmistaa, ettei kukaan ulkopuolinen pääse tutkimaan tietokannan dataa (Oracle, 2002c).

3.2.1 Oraclen turvallisuuslisäosa

Oracle-tietokannan tietoliikenteen salaaminen mahdollistuu maksullisen turvallisuuslisäosan *OAS* (Oracle Advanced Security) avulla (Oracle, 2002c). Lisäosaa käyttämällä tietokannan tietoliikenne voidaan salata ja varmistaa sen siirronaikainen eheys. Lisäksi sen avulla voidaan todentaa sekä asiakas että tietokantapalvelin, tarvittaessa käyttämällä kolmansien osapuolien autentikointiratkaisuja.

Palvelin ilmaisee asiakasohjelmille SQLNET.ORA-tiedostolla missä moodissa se on salauksen ja eheyden suhteen ja mitä algoritmeja se suostuu käyttämään. Eli SQLNET.ORA-tiedosto määrittelee onko turvallisuuslisäosa käytössä, ja kuinka laajasti. Tiedoston sisältöä voidaan muokata mm. Oraclen *Net Manager* -ohjelmalla. Kuvassa 16 on näkymä Net Manager -ohjelman Encryption-välilehdeltä.



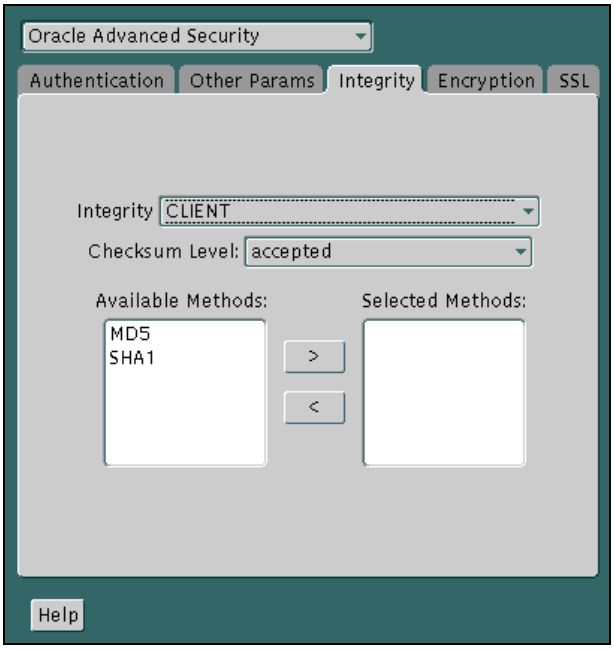
Kuva 16 OAS:n salaustoimintojen konfigurointi-ikkuna (Oracle, 2002a).

Vaihtoehto *Encryption* kuvassa 16 ilmaisee että ollaan asettamassa palvelimen salausasetuksia. Kohdassa *Encryption type* määritellään palvelimelle asetettava moodi, millä tasolla se salausta vaatii. Oletuksena on aina vaihtoehto ACCEPTED, mutta käyttäjä voi tarvittaessa muuttaa arvoa (REJECTED/ACCEPTED/REQUESTED/REQUIRED).

Kohdassa *Encryption seed* määritellään 10-70 merkkiä pitkä siemenluku, jota käytetään yhtenä siemenenä salauksen satunnaislukuja generoitaessa. Siemenlukua voidaan vaihtaa milloin vain, ja se jää selväkielisenä näkyviin tiedostoon SQLNET.ORA.

Kohdissa *Available Methods* ja *Selected Methods* näkyy tarjolla olevat ja valitut salausalgoritmit. Vain valittuja algoritmeja voidaan käyttää salauksessa. Kuvassa 16 ei ole vielä yhtään algoritmia valittuna.

Kuvassa 17 on näkymä samasta Net Manager –ohjelmasta. *Integrity*-välilehdellä voidaan asettaa tiedon eheyden asetukset päälle, tiivistefunktioita on valittavana kaksi. Kuvassa on Integrity-kohdassa valittuna ”Client”, mutta palvelimen asetuksia määritettäessä asetetaan vaihtoehdoksi ”Server”.



Kuva 17 OAS:n eheyden konfigurointi-ikkuna (Oracle, 2002a).

Oraclen (2002a) mukaan OAS:n salausmenetelmiä käytettäessä määritellään sekä palvelimelle että asiakkaalle millaisen tason tietoliikenteen salaukseen ja tietoliikennepakettien eheyden tarkastukseen kumpikin haluaa. Vaihtoehdot ovat: REJECTED, ACCEPTED, REQUESTED ja REQUIRED. Taulukossa 5 on kuvattu vaihtoehtoiset kombinaatiot konfiguraatiolle.

Taulukko 5 OAS:n salauksen tasot (Oracle, 2002a).

		Client			
		REJECTED	ACCEPTED	REQUESTED	REQUIRED
Server	REJECTED	OFF	OFF	OFF	Connection fails
	ACCEPTED	OFF	OFF[1]	ON	ON
	REQUESTED	OFF	ON	ON	ON
	REQUIRED	Connection fails	ON	ON	ON

[1] This value defaults to OFF. Cryptography and data integrity are not enabled until the user changes this parameter using Oracle Net Manager or by modifying the `sqlnet.ora` file.

REJECTED ei salli koskaan käyttää salausta, mutta jos toisella osapuolella on samaan aikaan asetettu vaatimus salauksen ehdottomasta käytöstä (REQUIRED), yhteys epäonnistuu. ACCEPTED sallii salauksen käytön, jos toinen osapuoli niin haluaa. Mikäli

molemmilla osapuolilla on asetettuna arvoksi ACCEPTED, ei salausta aseteta päälle. Yhteys ei voi epäonnistua, mikäli toisen osapuolen asetus on ACCEPTED. REQUESTED on muuten sama kuin ACCEPTED, mutta se salaa tietoliikenteen myös mikäli vastapuolen arvoksi on asetettu ACCEPTED. REQUIRED vaatii ehdottomasti salauksen, ja mikäli vastapuoli ei sitä kykene tai halua (REJECTED) tarjota, yhteys epäonnistuu (Oracle, 2002a). Vaikka selityksessä puhutaan pelkästään salauksesta, samat säännöt koskevat myös eheyden tarkastamista

Seuraavassa on lueteltu kaikki OAS:n tukemat salausalgoritmit, suluissa vaihtoehtoiset avainpituudet (Oracle, 2002a):

- RC4 (256/128/56/40)
- AES (256/192/128)
- 3DES (168/112)
- DES (56/40)

DES-salain toimii CBC-moodissa. Eheyden varmistamiseksi voi tiivistefunktiona olla joko MD-5 tai SHA-1 (Oracle, 2002a).

3.2.2 OJDBC-ajurin salausominaisuudet

JDBC (Java Database Connectivity) on standardi rajapinta, jonka avulla Java-sovellukset kommunikoivat relaatiotietokantojen kanssa. JDBC on alun perin Sun Microsystemsin kehittämä rajapinta, johon löytyy tuki useasta käytössä olevasta relaatiotietokannasta. Koska JDBC on vain rajapinta, ovat useat tietokantatoimittajat toteuttaneet oman versionsa JDBC-rajapinnasta, jolloin itse toteutus noudattaa JDBC-standardia, mutta tarjoaa myös joitakin tietokantakohtaisia laajennuksia (esim. tuen tietokantakohtaisten tietotyyppien käytölle). Näin on menetellyt myös Oracle luodessaan OJDBC-ajurin (Oracle JDBC driver) (Oracle 2002b). OJDBC-ajurin avulla asiakaskoneilta voidaan luoda tietokantayhteyksiä kahdella toisistaan poikkeavalla tavalla, voidaan käyttää ns. Thin-ajuria tai OCI-ajuria.

Oraclen (2002b) mukaan *Thin-ajuri* on tarkoitettu applettien ja yksinkertaisten Java-sovelluksien käyttöön, sillä se ei vaadi ajurikirjaston lisäksi mitään erillisiä asennuksia

asiakaskoneelle. Thin-ajurin tarjoamat toiminnot saadaan siis käyttöön kopioimalla ajurikirjasto (ojdbc14.jar) asiakaskoneelle. Ajurikirjasto on toteutettu puhtaasti Javalla, joten sen tehokkuus ei ole paras mahdollinen.

OCI-ajuri (Oracle Call Interface) vaatii toimiakseen erillisen Oraclen asiakasohjelman asennuksen asiakaskoneelle. Toisaalta OCI-ajuri on osittain toteutettu C-kielellä, jolloin sitä suositellaan käytettäväksi, mikäli sovellukselta vaaditaan suurta tehokkuutta. Koska OCI-ajuri vaatii asennuksen asiakaskoneelle, sitä ei voida käyttää applettien kanssa (Oracle, 2002b).

Mikäli tietokantapalvelimen OAS-turvallisuuslisäosa on käytössä, voidaan OJDBC-ajurin avulla salata tietoliikenne jollakin valittavana olevista salausalgoritmeista. Lisäksi voidaan laskea tietoliikennepaketeista tarkistussummat, jolloin voidaan varmistua pakettien eheydestä (muuttumattomuudesta).

OJDBC:n Thin-ajuri ei kuitenkaan tue kaikkia OAS:n tukemia algoritmeja, Oraclen (2002a) mukaan Thin-ajurin kanssa on mahdollista käyttää seuraavia salausalgoritmeja:

- RC4 (256/128/56/40)
- DES (56/40)

Eheyden varmistamiseen Thin-ajurin kanssa voidaan käyttää MD5-tiivistefunktiota.

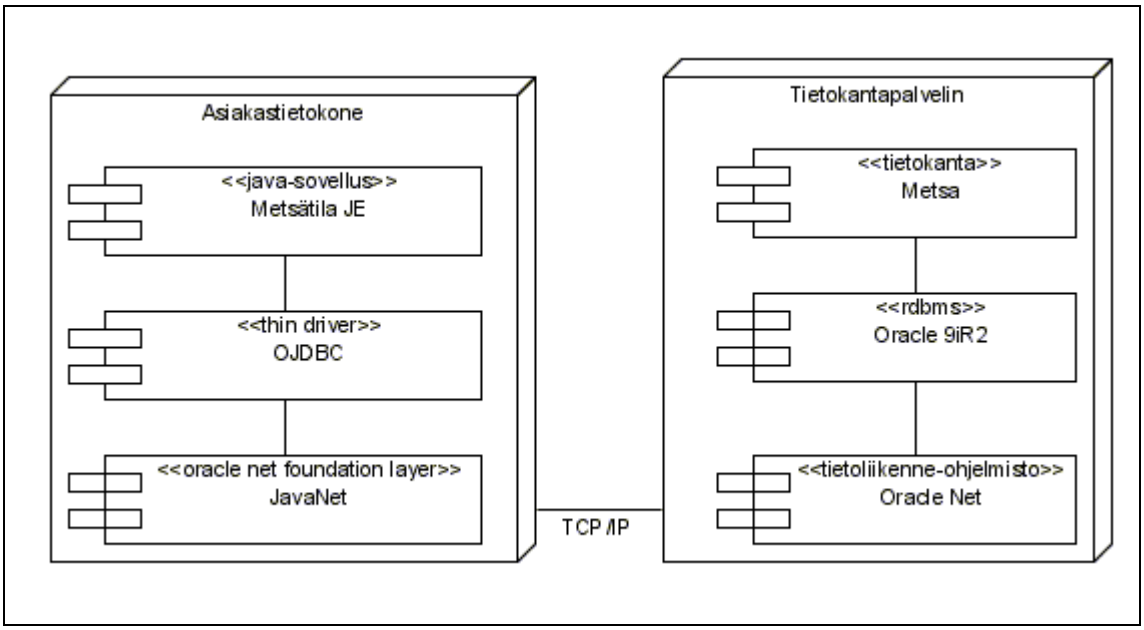
OCI-ajuria käytettäessä voidaan hyödyntää täysimittaisesti OAS:n tarjoamia toimintoja. Esimerkiksi käyttäjän ja palvelimen molemminpuolinen autentikointi voidaan suorittaa kolmansien osapuolien tuotteilla (mm. RADIUS, CyberSafe, Kerberos). Myös laajempi valikoima salaus- ja eheysalgoritmeja on käytettävissä (Oracle, 2002a). OCI-ajurin käyttö siis mahdollistaisi useampien huomattavasti turvallisempien algoritmien käytön, mutta kuten aiemmin todettiin, se vaatii asiakaskoneilta Oraclen ohjelmiston erillisasennuksen.

4 ESIMERKKISOVELLUS

Tutkielman viimeisessä luvussa sovelletaan tutkielmassa esiteltyjä salausmenetelmiä ja työkaluja käytännössä. Esimerkin tavoitteena on kuvata, kuinka saadaan salattua aiemmin tietojenkäsittelytieteen opintojen yhteydessä harjoitustyönä valmistuneen *Metsätila Java Edition 1.0* –sovelluksen (Päivinen & Ronkainen, 2005) tietoliikenne. Koska sovellus käyttää OJDBC Thin-ajuria, toteutetaan myös tietoliikenteen salaus niillä välineillä ja metodeilla, joita kyseinen yhteystyyppi tarjoaa.

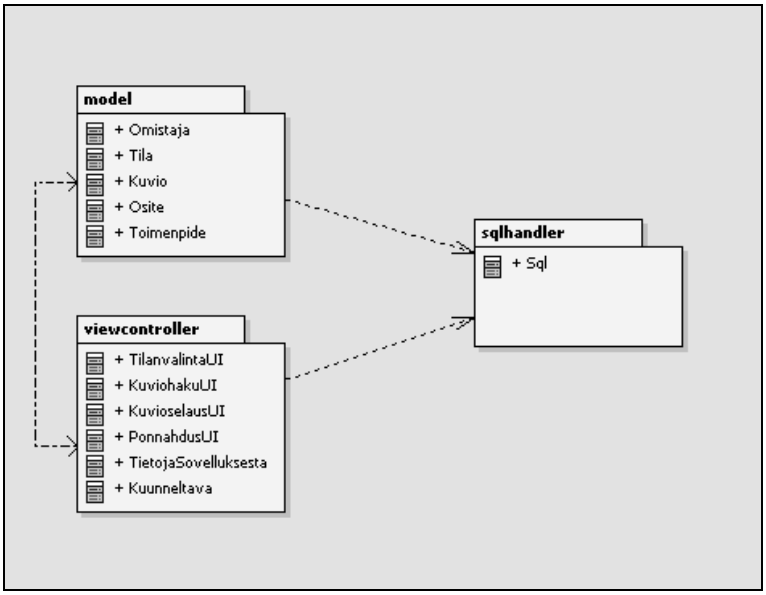
4.1 Metsätila Java Edition 1.0

Metsätila Java Edition, lyhyemmin Metsätila JE, on metsätilojen puustotietoja käsittelevä Java-sovellus. Käyttäjällä voi olla useita erillisiä metsätiloja järjestelmän tietokannassa, joiden tietoja hän pääsee lukemaan ja muuttamaan ohjelman käyttöliittymän kautta. Jokaisella käyttäjällä on sovellukseen oma henkilökohtainen käyttäjätunnus ja salasana. Ohjelma toimii itsenäisenä sovelluksena asiakaskoneella ja käyttää tietovarastonaan erillisellä palvelimella sijaitsevaa Oracle 9iR2-tietokantaa. Järjestelmän komponentti-rakenne on esitetty kuvassa 18.



Kuva 18 Metsätila Java Edition 1.0 –sovelluksen käyttämät komponentit.

Sovelluksen luokat on jaettu pakkauksiin, joita on kolme: *model*, *viewcontroller* ja *sqlhandler*. Sovelluksen sisäinen rakenne pakkauksien osalta on esitetty kuvassa 19. Kuten nimistä voidaan päätellä, on sovelluksessa pyritty toteuttamaan MVC-suunnittelumallin (Model-View-Controller) mukaista rakennetta. Liiketoimintaan eli metsätalouteen liittyvä toimintalogiikka on sijoitettu model-pakkaukseen ja käyttöliittymän ja käyttäjän komentojen kontrollointi viewcontroller-pakkaukseen. Sovelluksen tietoliikenteen tietokannan kanssa hoitaa kokonaan sqlhandler-pakkaus, ja tarkemmin sanottuna sen ainoa luokka nimeltään *Sql*.



Kuva 19 Metsätila Java Editionin pakkaukset luokkineen.

Seuraavassa kohdassa esitetään, kuinka salaamaton yhteys muodostetaan ja mitä pitää lisätä, että salaus ja tiedon eheyden varmistaminen saadaan käyttöön asiakaskoneella.

4.2 Tietoliikenteen salauksen käyttöönotto ohjelmassa

Tietokantaliikennettä hoitavan Sql-luokan konstruktoria käytetään yhteyden ominaisuuksien määrittelyyn ja sitä kutsutaan ensimmäisen yhteyden luomiseen sisäänkirjautumisen yhteydessä. Ilman salausta tai eheyden tarkastamista konstruktori on kuvan 20 mukainen.

```
package mtje.sqlhandler;

import java.sql.*;
import java.util.*;
import oracle.jdbc.pool.*;
import mtje.model.*;

public class Sql{
  OracleDataSource ods;
  Connection conn;
  private static String tilanne;

  public Sql() throws SQLException{
    ods = new OracleDataSource();
    ods.setURL("jdbc:oracle:thin:Username/Password@//193.167.42.127:1521:/metsa.cs.joensuu.fi");
    Connection conn = ods.getConnection();
  }
   ...
```

Kuva 20 Sql-luokan konstruktori ilman salausta.

Luokan alussa esitellään kaksi yhteyden kannalta merkityksellistä oliomuuttujaa, *ods* ja *conn*. OracleDataSource on Oraclen laajennus normaalista DataSource-luokasta, jonka ilmentymillä voidaan joustavasti määritellä tietokantoja ja muita tietoresursseja sovelluksien käyttöön (Oracle, 2005). Connection-luokka hoitaa varsinaisen yksittäisen yhteyden luomisen tietokantaan kunkin kyselyn yhteydessä.

Konstruktorissa luodaan ja alustetaan sekä OracleDataSource- että Connection-luokkien ilmentymät. OracleDataSource-luokan ilmentymälle annetaan parametrina tietokantayhteyden vaatimat tiedot, jonka jälkeen sen avulla alustetaan Connection-luokan olio *conn*. Konstruktori ei tee tämän lisäksi mitään muuta. Varsinaiset tietokantaoperaatiot suoritetaan Sql-luokan muissa metodeissa, joita sovelluksen muut oliot kutsuvat.

Tietoliikenteen salauksen päälle asettaminen ja eheyden varmistaminen tapahtuu käyttämällä Properties-luokan ilmentymää, jolle määritellään salaukseen liittyviä ominaisuuksia. Seuraavassa koodiesimerkissä (kuva 21) on samainen Sql-luokan konstruktori, mutta nyt otetaan salaus ja tiedon eheyden tarkastaminen käyttöön.

```
package mtje.sqlhandler;

import java.sql.*;
import java.util.*;
import oracle.jdbc.pool.*;
import mtje.model.*;

public class Sql{
  OracleDataSource ods;
  Connection conn;
  private static String tilanne;

  public Sql() throws SQLException{

    Properties prop = new Properties();
    prop.put("oracle.net.encryption_client", "REQUIRED");
    prop.put("oracle.net.encryption_types_client", "( RC4_256 )");
    prop.put("oracle.net.crypto_checksum_client", "REQUESTED");
    prop.put("oracle.net.crypto_checksum_types_client", "( MD5 )");

    ods = new OracleDataSource();
    ods.setConnectionProperties(prop);
    ods.setURL("jdbc:oracle:thin:Username/Password@//193.167.42.127:1521:/metsa.cs.joensuu.fi");
    Connection conn = ods.getConnection();
  }
```

Kuva 21 Sql-luokan konstruktori, salaus ja tiedon eheyden tarkastaminen otetaan käyttöön.

Properties-luokan ilmentymälle kerrotaan, että salausta pitää käyttää (REQUIRED) eikä yhteyttä siis muodosteta, ellei palvelin tarjoa tietoliikenteen salausta. Salauksessa käytetään RC4-salainta, joka on jonosalain ja siksi nopea ja tehokas tietoliikenteen salaamiseen. Lisäksi se on itse asiassa ainoa varteenotettava salain jota Oracle JDBC Thin-ajuri tukee. Avain on 256-bittinen. Yhteyttä ei myöskään muodosteta mikäli palvelin ei tue RC4-salaimen käyttöä.

Eheyden tarkastusta käytetään (REQUESTED) vain, mikäli palvelin tarjoaa kyseistä palvelua. Mikäli tietoliikenteen eheyden tarkastaminen tulee tietokantayhteydessä käyttöön, tiivistefunktiona käytetään MD5-funktiota.

Properties-luokan ilmentymälle voitaisiin samalla lailla syöttää myös muita yhteyteen liittyviä parametreja (käyttäjätunnus/salasana, URL, jne.), mutta tässä tapauksessa ne syötetään seuraavaksi luotavan OracleDataSource-luokan ilmentymän omien metodien avulla, kuten salaamattomassakin esimerkissä (kuva 20). Seuraavaksi OracleDataSource-luokan ilmentymä *ods* saa yhteyden salaukseen liittyvät määrittelyt itselleen *prop*-oliolta,

kun sen setConnectionProperties–metodia kutsutaan. Tämän jälkeen asetetaan vielä yhteyden loput tiedot, kuten salaamattomassakin esimerkissä, ja luodaan yhteysolio.

Ellei tietokantapalvelimessa ole otettu OAS-lisäosaa käyttöön, ei yhteyden muodostaminen onnistu, koska nyt asiakassovellus vaatii tietoliikenteen salaamista. Samoin vaikka OAS olisikin käytössä, mutta tukea asiakassovelluksen pyytämille algoritmeille ei löydy, ei yhteyden luominen onnistu. Yhteys ei myöskään onnistu vaikka OAS on päällä, mutta tilassa, jossa se kieltäytyy salaamasta mitään (REJECTED). Seuraavassa kohdassa käydään läpi, kuinka tietokantapalvelimen OAS asetetaan sellaiseen tilaan, että yllä oleva esimerkkikoodi toimii ja tietoliikenne salataan.

4.3 Oraclen turvallisuuslisäosan konfiguroiminen

Ennen kuin tietoliikenteen salaus ja eheyden tarkastus voidaan asiakassovelluksessa ottaa käyttöön, täytyy tietokantapalvelimen OAS-turvallisuuslisäosa konfiguroida käyttökuntoon. Konfigurointi tapahtuu kuvien 16 ja 17 mukaisesti Oracle Net Manager -ohjelmalla. Tällöin määritellään vaadittu salauksen ja tiedon eheyden taso, sekä sallitut algoritmit. Tässä tapauksessa vaadituksi tasoksi sekä salaukseen että tiedon eheyteen asetetaan ACCEPTED. Palvelin tarjoaa salauksen ja tiedon eheyden tarkastuksen, mikäli asiakas niitä pyytää. Salauksessa sallituiksi algoritmeiksi asetetaan AES-256 ja RC4-256. Eheyden tarkastamiseksi tarjotaan molemmat käytettävissä olevat algoritmit, eli MD5 ja SHA-1.

Kun Oracle Net manager –ohjelmalla tehdyt asetukset on tallennettu, päivittyvät tiedot tehdyistä valinnoista tiedostoon SQLNET.ORA. Alla olevassa esimerkissä (kuva 22) on esimerkkisovelluksen käyttämän SQLNET.ORA-tiedoston rivit konfiguroimisen jälkeen. Ensimmäisellä rivillä ilmaistaan ne salausalgoritmit, joita palvelin tukee, seuraavalla rivillä esitetään ne tiivistefunktiot, joita palvelimen puolesta voidaan käyttää ja viimeisellä rivillä on kyseisen hetken siemenluku. Koska tietokannan vaatimaksi salaustasoksi on asetettu oletusarvo ACCEPTED, kyseistä riviä ei erikseen esitetä tiedostossa.

```
# SQLNET.ORA Network Configuration
# Generated by Oracle configuration tools.

SQLNET.ENCRYPTION_TYPES_SERVER= (AES256, RC4_256)
SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER= (MD5, SHA1)
SQLNET.CRYPTO_SEED = Siemenluku11112222333
```

Kuva 22 SQLNET.ORA-tiedoston salausta koskevat rivit.

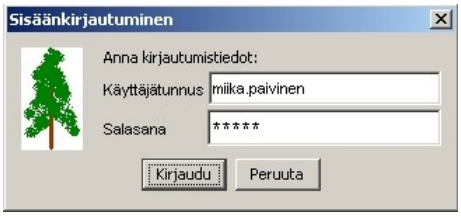
4.4 Salauksen todentaminen tietoliikennettä kuuntelemalla

Salauksen onnistumisen todentamiseksi, tarkastellaan Metsätila JE -sovelluksen tuottamaa tietoliikennettä ilman salausta ja salattuna. Käytön aikana tapahtuvaa tietoliikennettä kuunnellaan ja kirjataan lokitiedostoon.

Tietoliikenteen kuuntelu suoritetaan *Ethereal*-ohjelmalla (versio 0.10.13*)*. Ethereal-ohjelma on tarkoitettu mm. tietoliikenteen analysointiin, ongelmien selvittämiseen, sovelluskehityksen tukemiseen ja opetustarkoituksiin (Ethereal.com, 2005). Tässä tapauksessa Ethereal asennetaan asiakaskoneelle ja asetetaan kuuntelemaan koneen tietoliikennettä, ohjelma tuottaa kuuntelemastaan tietoliikenteestä lokin tekstitiedostoon. Järjestelmän kannalta Ethereal-ohjelma sijoittuu Kuvan 18 komponenttikaaviossa asiakaskoneelle, Metsätila JE:n ja tietoliikenneväylän väliin.

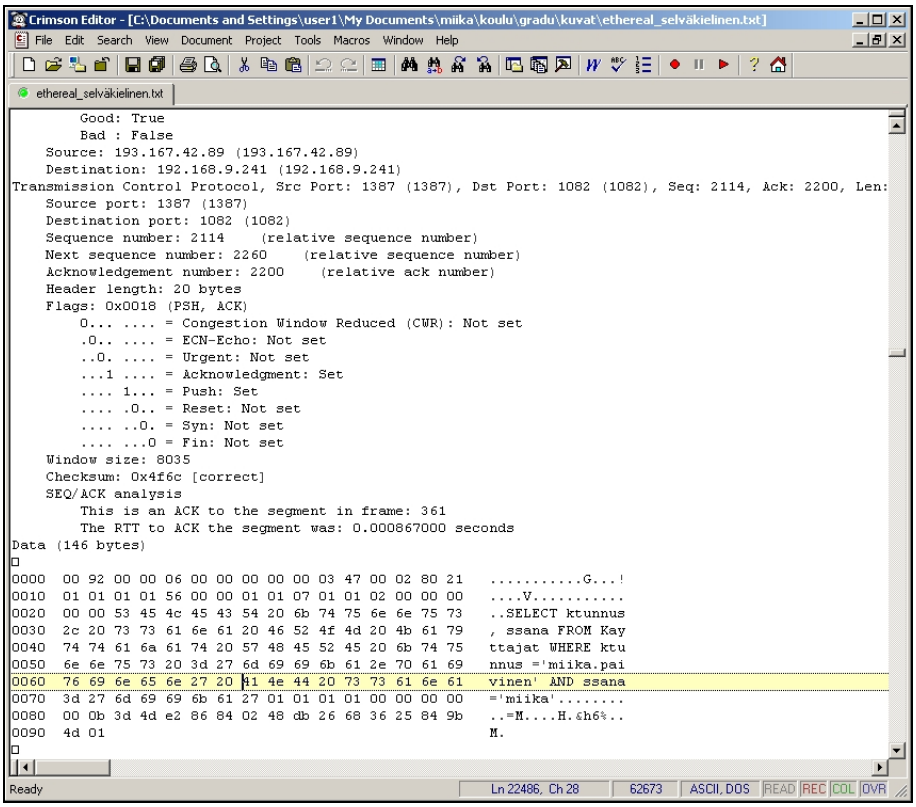
4.4.1 Salaamaton tietoliikenne

Metsätila JE -sovelluksen käyttäminen aloitetaan aina syöttämällä käyttäjätunnus ja salasana (kuva 23). Tämän jälkeen sovellus muodostaa yhteyden tietokantaan ja tarkastaa yksinkertaisella SELECT-lauseella löytyykö annettu käyttäjätunnus-salasana -pari tietokannan taulusta.



Kuva 23 Sisäänkirjautuminen Metsätila Java Editioniin.

Kun Metsätila JE –sovelluksessa käytetään kuvan 20 mukaista Sql-luokan konfiguraatiota (ei salaustoimintoja) ja käyttäjä suorittaa ohjelmaan kirjautumisen, generoi tietoliikennettä kuunteleva Ethereal-ohjelma muodostuvasta tietoliikenteestä loki-tiedoston (kuva 24).



Kuva 24 Tietoliikennelokin osa salaamattomasta tietoliikenteestä.

Kuvan alareunassa oleva säännöllisen muotoinen numeromatriisi sisältää heksadesimaalimuodossa kulkeneen datan. Sen oikealla puolella on heksadesimaaliarvoista ASCII-merkeiksi muutettuna kulkeneen datan sisältö. Ne merkit, joille ei löydy vastinetta ASCII-koodistosta, esitetään pisteinä. Esimerkiksi sana SELECT muodostuu heksadesimaaliarvoista 53, 45, 4C, 45, 43 ja 54, mikä voidaan kymmenjärjestelmän ja ASCII-koodien (Lähteinen & al., 2005) avulla muuntaa kirjaimiksi (taulukko 6).

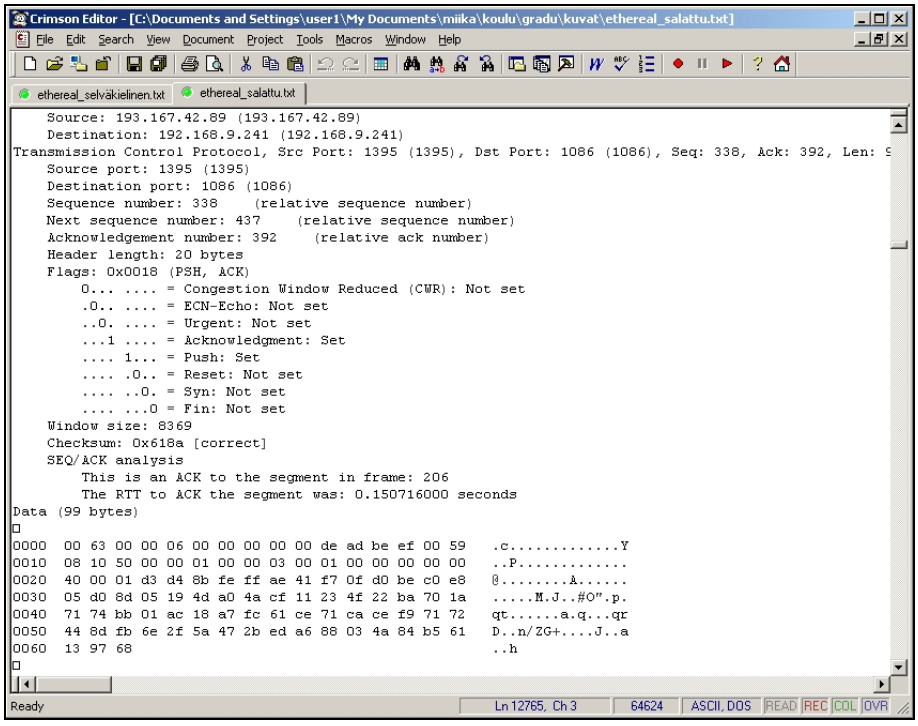
Taulukko 6 Heksadesimaalilukujen muuntaminen ASCII-merkeiksi.

Merkkijärjestelmä	**Arvot**					
Heksadesimaalijärjestelmä	53	45	4C	45	43	54
Kymmenjärjestelmä	83	69	76	69	67	84
ASCII-koodisto	S	E	L	E	C	T

Etherealin muodostamasta lokista voidaan todeta, että mm. käyttäjän käyttäjätunnus ja salasana järjestelmään kulkevat tietoliikenteessä selväkielisenä.

4.4.2 Salattu tietoliikenne

Kun Metsätila JE –sovelluksessa käytetään Sql-luokan konstruktorina kuvan 21 mukaista konfiguraatiota, tulee tietoliikenteen salaus käyttöön. Ohjelman sisäänkirjauksen yhteydessä kaapatusta tietoliikennelokista ei ole enää löydettävissä selkokielisiä SQL-lauseita. Kuva 25 havainnollistaa tilannetta.



Kuva 25 Tietoliikennelokin osa salatusta tietoliikenteestä.

5 YHTEENVETO

Tutkielmassa perehdyttiin kirjallisuuden avulla käytössä oleviin yleisimpiin salausmenetelmiin ja niiden teoreettiseen taustaan. Salausta kokeiltiin Java-perustaisen esimerkkisovelluksen ja Oraclen tarjoaman tietokanta-ajuri OJDBC:n avulla.

Vaikkakin useat salausmenetelmät ovat teoreettisesti vaikeita ja usein monimutkaisia toteuttaa, tutkielman teon aikana kävi ilmi, että ohjelmistotuotteissa on tarjolla hyvin pitkälle vietyjä valmiita toteutuksia. Tällöin salauksen käyttöönottamiseksi ei tarvitse osata toteuttaa itse salausmenetelmää, vaan usein riittää kun tietää millaista salausta tarvitsee. Esimerkkisovelluksen OJDBC:n Thin-ajurin salausominaisuudet toimivat juuri tällä periaatteella: salauksen saa käyttöön muutamalla koodirivillä, mutta täytyy tietää millaista salausta haluaa ja mitkä ovat käyttökelpoisia algoritmeja.

OJDBC:n Thin-ajurille tarjoamat vanhahkot ja osittain jopa tietoturvattomat algoritmit aiheuttivat hieman hämmennystä tutkielman teon aikana. Oraclen ohjekirjoissa algoritmien mukanaoloa perusteltiin sillä, että niitä mahdollisesti käytetään vielä jossakin vanhemmissa sovelluksissa, mutta ei toisaalta mainittu mitään siitä, että osa niistä on murrettavissa kotikoneen laskentateholla.

Yhtenä mielenkiintoisena jatkotutkimuskohteena voisikin olla tilanne, jossa tietoliikenne salattaisiin käyttämällä 40-bittistä DES-salausta ja se koetettaisiin murtaa, esimerkiksi käyttämällä yliopiston tietokoneita simuloimaan kaapattuja yksityisiä koneita. Toinen tutkielman teon aikana esiin tullut lisätutkimusta kaipaava aihe olisi varmastikin kvanttisalaus. Kvanttisalaus poikkeaa huomattavasti tekniseltä toteutukseltaan muista perinteisistä salausmenetelmistä ja kykenee ainakin teoriassa tarjoamaan niitä paremman suojan.

VIITELUETTELO

Barker, W. C. (2004) *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. NIST Special Publication 800-67 Version 1, National Institute of Standards and Technology, Gaithersburg (Saatavana myös: http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf, 18.11.2005).

Burke, J., McDonald, J., Austin T. (2000) Architectural Support for Fast Symmetric-Key Cryptography. *ACM SIGOPS Operating Systems Review*, 2000 (34), 178-189.

ECRYPT (2004) *D.SPA.10 ECRYPT Yearly Report on Algorithms and Keysizes*. Katholieke Universiteit Leuven (Saatavana myös: http://www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf, 21.09.2005).

Ethereal.com (2005) *Ethereal: A Network Protocol Analyzer*. WWW-sivusto, http://www.ethereal.com/ (16.11.2005).

Hurwitz M. V. (2000) Quantum ”encryption”. *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*, ACM Press, New York, 303-313.

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

Käpylä, T. (2000) Salaustekniikoilla lisää turvallisuutta, *Systeemityölehti*, 2000 (4), 26-32.

Lenstra, A. K. (2004) *Key Lengths* (*Contribution to The Handbook of Information Security)*. Lucent Technologies and Technische Universiteit Eindhoven (Saatavana myös: http://cm.bell-labs.com/who/akl/key_lengths.pdf, 21.09.2005).

Lähteinen, O., Pietikäinen, V., Kosonen, H (2002) *Uusi PC-tekniikan käsikirja*, WS Bookwell Oy, Juva.

McGraw G., Viega J. (2002) *Building secure software*. Addison-Wesley, USA.

Menezes, A. J., Oorschot, P. C., Vanstone, S. A. (1997) *Handbook of Applied cryptography*, CRC press LLC.

NIST (2005) *Recommendation for Key Management*. NIST Special Publication 800-57, (Saatavana myös: http://csrc.nist.gov/CryptoToolkit/kms/SP800-57Part1August2005.pdf, 11.12.2005).

NIST (2002) *Secure Hash Signature Standard (SHS) (FIPS PUB 180-2)*. Computer Security Standard, Cryptography, (Saatavana myös: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf, 29.05.2005).

Oracle (2002a) *Oracle Advanced Security Administrator's Guide, Release 2 (9.2)*. Oracle Corporation, http://www.oracle.com/pls/db92/db92.homepage (09.11.2005).

Oracle (2002b) *Oracle9i JDBC Developer's Guide and Reference, Release 2 (9.2)*. Oracle Corporation, http://www.oracle.com/pls/db92/db92.homepage (09.11.2005).

Oracle (2002c) *Oracle9i Security Overview, Release 2 (9.2)*. Oracle Corporation, http://www.oracle.com/pls/db92/db92.homepage (09.11.2005).

Oracle (2005) *Oracle Database JDBC Developer's Guide and Reference, 10g Release 2 (10.2)*. Oracle Corporation, http://www.oracle.com/technology/documentation/database10gr2.html (15.11.2005).

Päivinen M., Ronkainen A. (2005) *Metsätila Java Edition 1.0*. Tietojenkäsittelytieteen laudatur-harjoitustyö, Joensuun yliopisto, Tietojenkäsittelytieteen laitos, Joensuu.

Quantum Technologies (2005) *Quantum Cryptography "live": World Premiere: Bank Transfer via Quantum Cryptography Based on Entangled Photons.* http://www.quantenkryptographie.at/Quantum%20Cryptography%2021%20April%2004.pdf (02.11.2005).

RSA Laboratories (2000) *RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1*. ftp://ftp.rsasecurity.com/pub/labsfaq/rsalabs_faq41.pdf (18.09.2005).

RSA Laboratories (2005) *RSA Security - The New RSA Factoring Challenge*. http://www.rsasecurity.com/rsalabs/node.asp?id=2092 (18.09.2005).

Robshaw, M.J.B (1995) *RSA Laboratories Technical Report TR-701: Stream Ciphers version 2.0*. ftp://ftp.rsasecurity.com/pub/pdfs/tr701.pdf (21.09.2005).

Shannon, C.E. (1949) *Communication theory of secrecy systems*. Bell System Technical Journal, 1949 (28), 657-715.

Schneier, B. (2005) *Twofish*. http://www.schneier.com/twofish-brief.html, (29.09.2005).

Szydlo, M. (2005) *SHA1 Collisions can be Found in 2^63 Operations*. http://www.rsasecurity.com/rsalabs/node.asp?id=2927 (29.09.2005).

Valtionhallinnon tietoturvallisuuden johtoryhmä (2001) *Salauskäytäntöjä Koskeva Valtionhallinnon Tietoturvallisuussuositus: Salausmenetelmien käyttö valtion tietohallinnossa*. http://www.vm.fi/tiedostot/pdf/fi/3370.pdf (21.09.2005).

SSH Communications Security (2005) *SSH: Support: Cryptography A-Z.* WWW-sivusto, http://www.ssh.fi/support/cryptography/algorithms/asymmetric.html (12.05.2005).

Weiss, J. (2004) *Java Cryptography Extensions: Practical Guide for Programmers*. Elsevier, San Francisco.