Vaasan yliopisto
UNIVERSITY OF VAASA

Ville Pursiainen

Symmetristen lohkosalausalgoritmien suoritustehokkuuksien analysointi

Tapaukset AES-128 ja 3DES

Tekniikan ja innovaatiojohtamisen
akateeminen yksikkö
Tietojärjestelmätieteen Pro Gradu
Kauppatieteen maisteri

Vaasa 2025

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen akateeminen yksikkö
Tekijä: Ville Pursiainen
Tutkielman nimi: Symmetristen lohkosalausalgoritmien suoritustehokkuuksien analysointi: Tapaukset AES-128 ja 3DES
Tutkinto: Kauppatieteen maisteri
Oppiaine: Tietojärjestelmätiede
Työn ohjaaja: Tomi Pasanen
Valmistumisvuosi: 2025 **Sivumäärä:** 72

TIIVISTELMÄ:
Moderni informaatioyhteiskunta edellyttää informaation tehokasta salaamista. On tärkeää varmistaa, että yksityinen viestintä tai henkilötiedot pysyvät yksityisenä. Tämä toteutetaan käytännössä salausalgoritmeilla. Koska näitä algoritmeja on erilaisia, niiden suoritustehokkuutta on syytä tarkastella osana niiden vastuullista implementaatiota.

Tutkimuksen tavoite oli kehittää analyysimenetelmä, joka vastaa tähän tarpeeseen. Aiemmissa tutkimuksissa salausalgoritmien suoritustehokkuutta tutkittiin pääsääntöisesti perinteisillä aikamittauksilla. Tämä tarkoittaa, että eri algoritmien salaamiseen käyttämä aika mitattiin erimittaisilla syötteillä. Tällainen analyysimenetelmä on kuitenkin altis ulkopuolisille vaikutteille. Esimerkiksi analysoijan kyky implementoida algoritmi oikein tai käytetyn tietokoneen laskentateho voivat vaikuttaa mittaustuloksiin.

Tässä tutkimuksessa aihetta lähestyttiin tavanomaisen aikamittausten sijasta algoritmianalyysin teorian kautta. Näin ollen, analyysin tulos ei ole yhtä riippuvainen ulkopuolisista tekijöistä. Analyysimenetelmä on myös suunniteltu yksinkertaiseksi, jotta eri analysoijien tulokset algoritmin suoritustehokkuudelle eivät poikkea merkittävästi toisistaan. Tutkimuksen rajaamiseksi ehdotettu malli koskee vain lohkosalausalgoritmeja, kuten AES-128 ja 3DES.

Tutkimus on suunnittelutieteellinen tutkimus ja ehdotettu analyysimenetelmä on sen tuote, eli artefakti. Tutkimuksen aineisto, eli tutkimustyypin termein tietämyskanta, tukeutuu vahvasti algoritmianalyysin sekä lohkosalausalgoritmien teoriaan. Aineisto kerättiin eri tietokannoista sekä useista alan teoksista.

Tutkimuksessa huomattiin, että lohkosalausalgoritmien rakenteiden samankaltaisuuksia voidaan hyödyntää niiden analysoinnissa. Ne salaavat syötteensä lohkoissa, joiden koko on algoritmille vakio. Näiden lohkojen sisältä löytyy jonkinlainen kierrosrakenne, jota algoritmi toistaa aina uudelleen. Algoritmin suoritustehokkuutta voidaan analysoida tutkimalla ensin sen pienimpiä rakenteita, eli kierroksia ja kierroksen funktioita. Tästä voidaan sitten ekstrapoloida ylöspäin, johtaen lopulta koko algoritmin analyysiin. Tutkimukseen liittyy myös esimerkkitapaus, jossa tätä menetelmää hyödynnetään AES-128- ja 3DES lohkosalausalgoritmeihin.

Tutkimus tukee aiempaa tutkimusta kehittämällä uuden työkalun algoritmianalyysin tueksi. Kyseinen työkalu on suhteellisen helppokäyttöinen, eikä se vaadi syvällistä perehtymistä salauksen teoriaan tai vaadi monipuolista matemaattista osaamista. Ehdotetun analyysimallin tulokset ovat vertailukelpoisia toisiinsa, jolloin eri algoritmien suoritustehokkuuksien vertailu on myös mahdollista, vastaten modernin informaatioyhteiskunnan tarpeisiin.

AVAINSANAT: Algoritmianalyysi, AES-128, 3DES, salausalgoritmi, lohkosalaus, suoritustehokkuus

Sisällys

1 Johdanto 6
1.1 Tutkimuksen tavoite 7
2 Algoritmianalyysi ja lohkosalausalgoritmit 8
2.1 Algoritmianalyysi 8
2.1.1 Vakioaikainen toiminto 8
2.1.2 Syötteen vaikutus tehokkuuteen 9
2.1.3 Asymptoottisuus, ja kasvun voimakkuus arvioinnin kohteena 10
2.1.4 Asymptoottisen ylärajan notaatio 12
2.2 Symmetrinen lohkosalaus 14
2.3 AES-128 15
2.3.1 Tavujen substituutio 16
2.3.2 Rivien siirto 17
2.3.3 Kolumnien sekoitus 17
2.3.4 Lisää kierrosavain 18
2.3.5 Avainalgoritmi 18
2.4 3DES 19
2.4.1 Feistel-salaus 19
2.4.2 Aloituspermutaatio 20
2.4.3 Kierrosavainten muodostaminen 21
2.4.4 Salauskierros 21
2.4.5 Lopetuspermutaatio 22
2.5 Aiempaa tutkimusta 22
3 Suunnittelutieteellinen tutkimus 24
3.1 Suunnittelutieteellisen tutkimuksen teoriaa 24
3.2 Tutkimusaineisto 26
4 Artefakti: lohkosalausalgoritmin analysointi 28
4.1 Taustatietoa arviointimallille 28

4.2 Analyysin suorittaminen 30
4.2.1 Analyysistrategia 30
4.2.2 Analyysin aloittaminen - Ongelmanmäärittely 32
4.2.3 Algoritmin osien eristäminen 33
4.2.4 Vakioaikaisten toimintojen laskeminen 33
4.3 Lohkon analyysin ekstrapolointi laajemmalle algoritmille 38
4.4 Analyysin vaiheet 40
5 Esimerkkitapaukset AES-128 ja 3DES 41
5.1 Analysoitavat ohjelmakoodit 41
5.2 Ehdotettu malli 47
5.3 Ongelmanmäärittely 47
5.4 AES-128 analyysi 48
5.4.1 AES-128 osien erittely 48
5.4.2 AES-128 osien analysointi 49
5.4.3 AES-128 analyysien yhdistäminen 53
5.4.4 AES-128 ekstrapolointi kokonaissyötteen suhteen 54
5.4.5 AES-128 ylärajan muodostaminen 54
5.5 Analyysi 3DES algoritmille 55
5.5.1 3DES osien analysointi 56
5.5.2 3DES analyysien yhdistäminen 61
5.5.3 3DES ekstrapolointi kokonaissyötteen suhteen 62
5.5.4 3DES ylärajan muodostaminen 62
5.6 AES-128 ja 3DES suoritustehokkuuden vertailu 63
6 Diskussio tuloksista 64
6.1 Tutkimuksen rajoitteet ja luotettavuus 64
6.2 Tulosten vertailu aiempaan tutkimukseen 65
6.3 Mahdollisia jatkotutkimusaiheita 66
Lähteet 67
7 Liitteet 70

Kuvat

Kuva 1 - Termien kasvu, n→∞ 11
Kuva 2 - Kasvun voimakkuuksia eri suuruusluokilla (Alsuwaiyel, 2016, s.23) 13
Kuva 3 - AES-algoritmin toiminta. (Xilinx, 2020) 16
Kuva 4 - AES S-box (St. Denis & Johnson, 2007, s.148) 17
Kuva 5 - MDS · kolumni a (St. Denis & Johnson, 2007, s. 151) 18
Kuva 6 - Rcon-taulukko (St. Denis & Johnson, 2007, s.155) 19
Kuva 7 - 3DES-algoritmi, joka toistetaan kolmesti (Oppliger, 2005, s. 243) 20
Kuva 8 - Suunnittelutieteellinen tutkimus (Vom Brocke ja muut, 2020) 24
Kuva 9 - Analyysiprosessi (Elo ja Kyngäs, 2008, s. 109) 28
Kuva 10 - Symmetrisen lohkosalausalgoritmin yleinen rakenne 29
Kuva 11 - Algoritmin hajottaminen osiin 31
Kuva 12 - Lisäysjärjestäminen (Cormen ja muut, 2009, s.26) 34
Kuva 13 - AES-128. Texas Instruments, 2011. 43
Kuva 14 - 3DES (Bhandarkar, 2018) 46
Kuva 15 - galois_mul2. Texas Insrument, 2011. 50
Kuva 16 - SubBytes ja AddRoundKey 51
Kuva 17 - MixColumn 52
Kuva 18 - Aloituspermutaatio & lopetuspermutaatio. (Bhandarkar, 2018). 70
Kuva 19 - Laajennusfunktio & salauskierroksen permutaatio. (Bhandarkar, 2018). 70
Kuva 20 - Sboxit. (Bhandarkar, 2018). 71
Kuva 21 - Avainalgoritmin taulukot. (Bhandarkar, 2018). 72

Taulukot

Taulukko 1 - Analyysiesimerkki (Cormen ja muut, 2009, s.26) 34
Taulukko 2 - Analyysin vaiheet 40
Taulukko 3 - Ehdotettu analyysimalli 47

1 Johdanto

Nykyaikaisessa informaatioyhteiskunnassa tietotekniset laitteet ja tietoverkot ovat läsnä lähes jokaisen elämässä. Ihmisten elämää koskevia tietoja käsitellään erilaisissa tietokannoissa, ja erilaiset laitteet tietokoneista älypuhelimiin ja tabletteihin ovat arkipäiväistyneet. Näiden myötä myös salausalgoritmit ovat tulleet osaksi ihmisten jokapäiväistä elämää, vaikka ne toimivatkin usein näkymättömästi (Google Cloud, 2024).

Tietotekniset laitteemme suorittavat jatkuvasti salausalgoritmeja (Google Cloud, 2024). Tämä tarkoittaa loogisesti sitä, että suoritustehokkuus on tärkeä ominaisuus salausalgoritmille, sillä laitteen käyttökokemus voi mahdollisesti vaarantua, jos laite pysähtyy ajoittain suorittaakseen käyttäjälle näkymättömiä algoritmeja. On siis olemassa perusteltu tarve perehtyä salausalgoritmien suoritustehokkuuteen, ja kehittää jokin systemaattinen tapa arvioida tätä tehokkuutta. Tämä tutkimus vastaa tähän tarpeeseen.

Tämä tutkimus on tyypiltään suunnittelutieteellinen tutkimus. Tämä tarkoittaa sitä, että tutkimuksen tavoite on tuottaa artefakti, joka vastaa tutkimuksen taustalla olevaan käytännön ongelmaan. Tutkimuksen taustalla oleva ongelma on tarve salausalgoritmien suoritustehon analyysimenetelmälle ja artefakti on siis juuri tutkimuksessa ehdotettu analyysimenetelmä. Kehitettyä analyysimenetelmää sitten sovelletaan AES-128-algoritmiin ja 3DES-algoritmiin testitapauksena. Ehdotettu analyysimenetelmä rajataan koskemaan vain lohkosalausalgoritmeja, sillä tutkimuksen laajuus on rajallinen. Tehokkuusarvio tuotetaan nojaten asymptoottisen analyysin teoriaan, jossa sekuntien sijasta algoritmia arvioidaan abstraktista vakioaikaisten toimenpiteiden näkökulmasta. Tämä näkökulman vaihto erottaa tutkimuksen merkittävästi aiemmista tutkimuksista.

Tämä eroaa aiemmista tutkimuksista, kuten Pandan (2016) analyysistä siten, että Pandan tutkimus käsittelee salausalgoritmien analyysiä tavanomaisesti ajan mittaamisen kautta, eli hän mittaa montako millisekuntia algoritmilla kestää salata dataa. Tämä teema toistuu Durranin ja Hayan (2022) artikkelissa, jonka analyysimetodi on perinteinen aikamittaus.

1.1 Tutkimuksen tavoite

Tutkimuksen tavoite on tuottaa suunnittelutieteellinen artefakti, jonka avulla aiheeseen pinnallisesti perehtynyt henkilö, ilman syvällisiä matemaattisia taitoja, voi tuottaa kelvollisen analyysin symmetrisestä lohkosalausalgoritmista. Artefakti on muodoltaan dokumentti, joka sisältää toimintamallin tätä prosessia varten. Tämän tutkimusraportin luku 4 on kyseinen dokumentti, eli lukua voidaan hyödyntää itsenäisenä opasteena analyysiprosessissa. Tutkimus sisältää myös esimerkkitapaukset artefaktin soveltamisesta sekä AES-128 että 3DES salausalgoritmiin.

Tutkimus hyödyntää kahta tutkimuskysymystä. Nämä kysymykset ovat:

1) Miten symmetrisiä lohkosalausalgoritmeja voidaan analysoida tehokkaasti, mutta yksinkertaisesti?
2) Miten kehitettyä artefaktia hyödynnetään käytännössä?

Tutkimusraportin rakenne koostuu useista osista. Ensin, luvussa kaksi käsitellään hyödyllistä teoriaa, johon artefakti perustuu. Tässä teoriassa on kolme osaa, ensimmäinen käsittelee algoritmianalyysiä ja toinen symmetristä salausta, eritoten AES-128 ja 3DES algoritmeja. Kolmas osa käsittelee hieman aiempaa tutkimusta algoritmien suoritustehoanalyysistä. Luku kolme käsittelee tutkimusmenetelmää, eli suunnittelutieteellistä tutkimusta, sekä tutkimusaineiston keräämistä. Luku neljä sisältää itse artefaktin, eli kehitetyn menetelmän algoritmianalyysille. Luku viisi puolestaan käsittelee artefaktin soveltamista AES-128 ja 3DES algoritmeihin. Lopuksi luku kuusi sisältää lyhyen diskussion tutkimuksen tuloksista.

2 Algoritmianalyysi ja lohkosalausalgoritmit

Tässä luvussa käsitellään tutkimuksen kannalta oleellista algoritmianalyysin ja lohkosalauksen teoriaa, sekä esitellään tärkeitä käsitteitä. Luku 2.1 alalukuineen käsittelee algoritmianalyysiä, joka toimii teoriapohjana tutkimukselle. Luku 2.2 puolestaan käsittelee lohkosalauksen teoriaa yleisesti, jotta lukija ymmärtää mitä suoritetussa algoritmianalyysissä arvioidaan. Tällöin tutkimuksen tuottaman artefaktin yleinen käyttökonteksti on määritelty tarpeeksi tarkasti. Luvut 2.3 ja 2.4 käsittelevät AES-128-algoritmia ja 3DES-algoritmia, jotta lukija ymmärtää luvussa 5 käsiteltyjen testitapauksien sisällön. Lopuksi luku 2.5 esittelee lyhyesti salausalgoritmien suoritustehokkuusanalyysiin liittyvää aiempaa tutkimusta.

2.1 Algoritmianalyysi

Cormen ja muut (2009, s.29) määrittelee algoritmianalyysin ennustukseksi algoritmin resurssitarpeista. Nämä resurssit voidaan määritellä monella tavalla, kuten tietokoneelta vaadittuna muistikapasiteettina tai tiedonsiirron tietoverkolta vaatimana kaistanleveytenä. Cormenin ja muiden mukaan vaatimuksella kuitenkin tyypillisesti tarkoitetaan aikavaatimusta: Kuinka tehokkaasti algoritmi käyttää aikaa? Tämä aikatehokkuus on juuri se vaatimusmäärittely, jota tämä tutkimuskin suoritustehokkuudesta käyttää.

2.1.1 Vakioaikainen toiminto

Algoritmin aikavaatimuksen tutkiminen kuitenkin asettaa oman ongelmansa ajan mittaamiselle: Mitä yksikköä käyttää? Puhtaasti ajan mittaaminen sekunneissa tai sen osissa ei ole aina toimiva mittausmetodi algoritmille. Mietitään tapausta, jossa sama algoritmi suoritetaan kahdella erilaisella tietokoneella, A ja B. Oletetaan, että A:n laskennallinen tehokkuus on kaksinkertainen B:hen verrattuna. On selvää, että A suorittaa algoritmin nopeammin, sillä se kykenee suorittamaan enemmän laskennallisia

toimenpiteitä samassa ajassa kuin B. Yhdellä algoritmilla on siis useampi eri suoritusaika, riippuen algoritmia suorittavasta laitteesta (Alsuwaiyel, 2016, s. 21).

Cormen ja muut (2009, s.25) tarjoavat vastauksen tähän ongelmaan. Tavallisen ajan käsitteen sijasta tehokkuutta tulee mitata primitiivisten laskennallisten toimitusten, kuten aritmeettisten toimitusten, datan tallentamisen tai datan siirron lukumääränä. Näitä tietokoneelle triviaaleja toimintoja voidaan kutsua vakioaikaisiksi toiminnoiksi (eng. constant time instruction), joiden suorittamiseen kuluu tietokoneelta hyvin pieni aika. Tämän ajan oletetaan olevan toiminnosta riippumatta vakio. Algoritmeja voidaan siten verrata toisiinsa niiden vakioaikaisten toimintojen määrien perusteella. Kun analyysin tulos on sekuntien sijasta abstrakti lukumäärä vakioaikaisia toimintoja, tuloksesta tulee vertailukelpoinen: Aiemmassa esimerkissä esitellyt A ja B saavat subjektiivisten aikamittausten sijasta saman absoluuttisen tuloksen, sillä tulos ei enää riipu tietokoneen tehosta.

2.1.2 Syötteen vaikutus tehokkuuteen

Joidenkin algoritmien suoritustehokkuus riippuu sille annetun syötteen koosta. Esimerkiksi, jos algoritmin tehtävä on asettaa joukko lukuja suuruusjärjestykseen, syötteeksi annetun joukon koko luonnollisesti vaikuttaa algoritmin suoritusaikaan. Enemmän lukuja voi tarkoittaa enemmän työtä (Cormen ja muut, 2009, s.25). Myös salausalgoritmin analyysissä tämä on syytä huomioida, sillä syötteen (salattavan tekstin) koko voi vaihdella.

On myös huomattavaa, että syötteen eri ominaisuudet voivat vaikuttaa algoritmin toimintaan. Esimerkiksi lukuja järjestävä algoritmi voi vaihdella suoritustehokkuudeltaan riippuen siitä, ovatko syötteen luvut jo valmiiksi järjestyksessä tai esimerkiksi täysin käänteisessä järjestyksessä. Algoritmien analysoinnissa on yleensä syytä olettaa, että syöte kuvastaa pahinta mahdollista tapausta. Tällöin analyysi kuvastaa suoritustehokkuuden ylärajaa, jolloin algoritmi ei voi ikinä suoriutua huonommin kuin on arvioitu (Cormen ja muut, 2009, s.27).

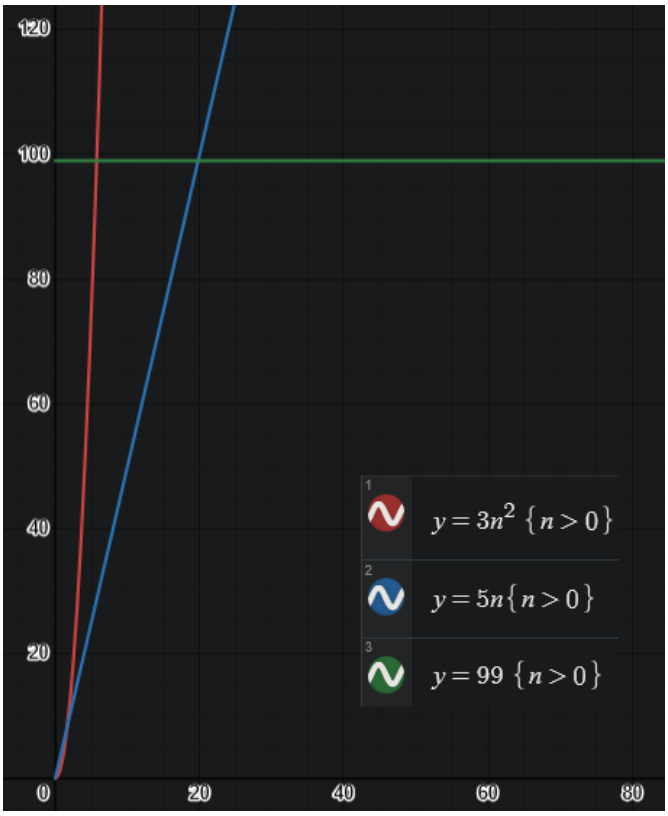
Tyypillisesti algoritmin suoritustehokkuutta tulkitaan siis pahimman tapauksen funktiona, joka kuvaa vakioaikaisten toimintojen määrää syötteen suhteen (Cormen ja muut, s.26 & s.43). Syötteen koko merkitään muuttujalla n ja pahimman tapauksen funktion nimi on T(n).

2.1.3 Asymptoottisuus, ja kasvun voimakkuus arvioinnin kohteena

Syötteen koolla on merkitystä algoritmin suoritustehokkuuteen, kuten luvussa 2.1.2 todettiin. Tästä seuraa loogisesti seuraava jatkokysymys: Mikä arvo tulee valita syötteen kooksi, eli arvoksi n? Tähän kysymykseen voidaan vastata asymptoottisuuden käsitteellä.

Cormenin ja muiden (2009, s.43) mukaan algoritmin syötteen koosta n muodostuu nopeasti algoritmin suoritustehokkuutta dominoiva tekijä, kun se kasvaa yhä suuremmaksi. Näin ollen, sen sijaan että laskettaisiin yksittäisten vakioaikaisten toimintojen määrää jollain vakio n-arvolla, suurempi hyöty saadaan arvioimalla niiden määrää, kun n kasvaa kasvamistaan. Tätä tehokkuusarviota, jossa n kasvaa rajattomasti, kutsutaan asymptoottiseksi tehokkuudeksi. Tällöin siis tutkitaan, miten funktio T(n) käyttäytyy, kun n lähestyy ääretöntä. Asymptoottisen tehokkuuden käsite suo tietyn mahdollisuuden yksinkertaistaa algoritmianalyysiprosessia, kuten seuraavasta esimerkistä käy ilmi.

Otetaan esimerkkinä algoritmi A, ja oletetaan sille pahimman tapauksen tehokkuus: $T(n) = 3n^2 + 5n + 99$. Jos funktion eri termit esitetään graafisesti, saadaan kuvan 1 mukaiset tulokset.



Kuva 1 - Termien kasvu, n→∞

Tärkein havainto kuvasta 1 on termien kasvun voimakkuus. Kun syötteen koko n kasvaa asymptoottisesti, eli rajatta, suurin termi $3n^2$ kasvaa selkeästi voimakkaimmin. Sen sijaan termi 99 ei riipu ollenkaan syötteen koosta. On siis selvää, että tapauksessa, jossa n kasvaa rajattomasti, termien 5n ja 99 vaikutus koko funktion suoritustehokkuuteen on lopulta minimaalinen verrattuna termiin $3n^2$. Näin ollen, Cormenin ja muiden (2009, s.28) mukaan algoritmianalyysissä huomio kiinnitetään aina muuttujan n suurimpaan termiin, sillä muut termit muuttuvat merkityksettömiksi sen rinnalla. Algoritmianalyysissä ei siis pyritä täydellisen yksityiskohtaiseen arvioon, vaan arvioon siitä kuinka voimakkaasti funktio T(n) kasvaa kun n kasvaa. Tähän riittää käytännössä suurimman termin arviointi, jolloin T(n) pelkistyy muotoon $T(n) = 3n^2$. Myös Alsuwaiyel (2016, s. 22) tukee tätä ajattelutapaa yksinkertaistetusta analyysistä.

2.1.4 Asymptoottisen ylärajan notaatio

Kuten alaluvussa 2.1.2 todettiin, algoritmianalyysissä on syytä tutkia algoritmin suoritustehokkuuden pahinta tapausta, eli sen ylärajaa. Asymptoottisen tehokkuuden ylärajaa merkitään O-notaatiolla. Asymptoottinen yläraja tarkoittaa, että jotain funktiota f(n) voidaan rajoittaa jostain syötteen koon n lähtöarvosta n0 alkaen jollain toisella funktiolla g(n), kertomalla g(n) vakiokertoimella c (Cormen ja muut, 2009, s.47). Eli on olemassa jokin n0 ja c, joilla voidaan määritellä g(n):

$$0 \leq f(n) \leq c \cdot g(n), n0 \leq n$$

Jos ehto pitää paikkaansa, sanotaan että funktion f(n) suoritusaika on O(g(n)), eli:

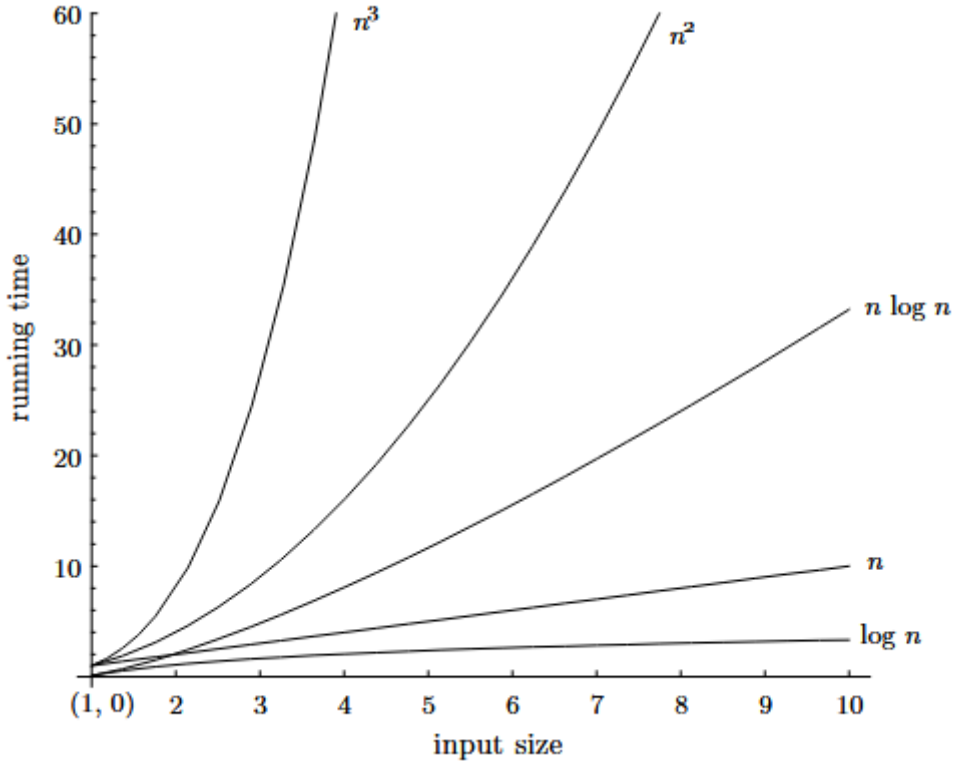
$$f(n) = O(g(n))$$

Otetaan esimerkiksi jälleen funktio T(n) = $3n^2 + 5n + 99$, joka aiemmin pelkistyi muotoon T(n) = $3n^2$. Tämä vastaa ylärajan määritelmässä funktiota f(n). Asymptoottinen yläraja löytyy jollain funktiolla g(n), jonka kertomalla jollain vakiolla c, on aina suurempi tai yhtä suuri kuin T(n) = $3n^2$, kun n >= n0. Intuitiivisesti g(n):ksi voidaan valikoida suuri määrä erilaisia funktioita. On esimerkiksi selvää, että:

$$[\mathrm{n}^2 \leq \mathrm{cn}^2] \vee [\mathrm{n}^2 \leq \mathrm{cn}^3] \vee [\mathrm{n}^2 \leq \mathrm{cn}^4], \mathrm{kun}\ 0 < \mathrm{n0}\ \wedge\ \mathrm{c} \geq 1$$

Kunhan c ja n0 valitaan huolellisesti, epäyhtälöiden paikkansapitävyys eivät vaarannu.

On huomattavaa, että asymptoottinen O-notaatio ei itsessään ota kantaa siihen, kuinka väljä tai tiukka yläraja on. Kuva 2 esittää funktioiden suuruusluokkia:



Kuva 2 - Kasvun voimakkuuksia eri suuruusluokilla (Alsuwaiyel, 2016, s.23)

Kuvasta kaksi nähdään, että n-funktiota voidaan selvästi rajoittaa ylhäältäpäin isomman suuruusluokan funktioilla. Lineaarinen f(n) kyllä rajoittuu sekä n^2 sekä n^3 funktioilla, mutta asymptoottinen analyysi tuskin saavuttaa tavoitettaan, jos ylärajaksi valitaan jokin koomisen iso raja, vaikka se teoriassa olisikin käypä vastaus. On siis syytä pyrkiä kohti asymptoottisesti tiukkaa rajaa, jossa g(n) rajoittaa f(n) funktiota vain vakiokertoimen avulla, n-muuttujan kokoluokan kasvattamisen sijaan (Cormen ja muut, 2009, s.47).

Palataksemme T(n) = $3n^2$ tapaukseen, kaikki ehdotetut ylärajavaihtoehdot toimivat, mutta g(n) = n^2 on rajana tiukempi verrattuna g(n) = n^3 tai g(n) = n^4. Itseasiassa, valittaessa ylärajaksi nimenomaan g(n) = n^2, on kyseessä tiukka yläraja, sillä valitsemalla arvot c ja n0 tarkasti, rajoittaa se funktiota T(n) vakiokertoimen tarkkuudella. Tällöin:

$$3n^2 \leq cn^2, kun\ [0 < n0] \wedge [\mathrm{c} \geq 3]$$

Jolloin:

$$T(n) = O(n^2)$$

Kyseessä on tällöin Cormenin ja muiden (2009, s.47) määritelmällä asymptoottisesti tiukka raja. Se on tehokkuusarviona hyödyllinen, sillä se mallintaa algoritmin tehokkuutta suhteellisen tarkasti, ainakin tarkemmin kuin suuremman eksponentin ylärajat.

2.2 Symmetrinen lohkosalaus

Salauksella tarkoitetaan tiedon saattamista muotoon, jonka vain tarkoitettu vastaanottaja pystyy purkamaan sen alkuperäiseen, ymmärrettävään muotoon (Kikani ja muut, 2022, s. 3247 & Chandra ja muut, s.1228). Näin varmistetaan tiedon turvallisuustavoitteet, nimellisesti tiedon luottamuksellisuus, eheys, aitous ja kiistämättömyys (Chandra ja muut, s.1228 & St. Denis & Thompson, 2007, s. 4). Salauksessa käytetään salausalgoritmia, joka ottaa salattavan tekstin ja salausavaimen, tuottaen salatekstin. Symmetrisessä salauksessa sekä salaaminen että salauksen purkaminen suoritetaan samalla avaimella (Aumasson, 2018, s.1).

Lohkosalausalgoritmi on salausalgoritmiperhe, johon kuuluvat algoritmit salaavat dataa lohkoissa. Salattava syöte siis jaetaan vakiokokoisiin osiin, lohkoihin, ja nämä osat salataan erikseen (Aumasson, 2018, s.53 & 55). Tämän ominaisuuden lisäksi lohkosalausalgoritmien toimintatapa perustuu tyypillisesti yhteen kahdesta arkkitehtuurista: Joko substituutio-permutaatioverkkoon tai Feistel-salaukseen. Esimerkiksi AES-algoritmit perustuvat substituutio-permutaatioon, kun DES (Data encryption standard) -pohjaiset algoritmit, kuten 3DES, puolestaan edustavat Feistel-salausta (Aumasson, 2018, s.55–56). Substituutio-permutaatioverkkoa käsitellään lähemmin alaluvussa 2.4 ja Feistel-salausta luvussa 2.5.

Lohkosalausalgoritmit ovat rakenteeltaan kierrospohjaisia. Tämä tarkoittaa sitä, että salaus suoritetaan yhden suuren salaustoiminnon sijaan toistamalla yhtä suhteellisen kevyttä salaustoimintoa useita kertoja samalle salattavalle datalle. Nämä

salauskierrokset hyödyntävät pääasiallisesta salausavaimesta johdettuja joka salauskierrokselle uniikkeja kierrosavaimia. Kierrosavaimet muodostetaan erityisellä avainalgoritmilla (eng. key schedule algorithm), joka on osa itse lohkosalausalgoritmia. On huomattavaa, että yksittäisen salauskierroksen tuottaman salauksen voimakkuus on suhteellisen heikko. Kierrospohjaisen rakenteen tuottaman salauksen vahvuus perustuu pääasiallisesti salauskierrosten määrään, ei niinkään yksittäisen salauskierroksen vahvuuteen. (Aumasson, 2018, s.55–56).

Tiivistetysti, lohkosalausalgoritmit ovat rakenteeltaan suhteellisen säännöllisiä. Ne käsittelevät salattavaa dataa vakiokokoisissa lohkoissa. Lohkon salaaminen tapahtuu kierrospohjaisesti, jossa salaaminen tapahtuu hyödyntämällä yksinkertaista salauskierrosta useita kertoja peräkkäin. Jokaisella kierroksella käytetään uutta kierrosavainta, joka johdetaan pääasiallisesta salausavaimesta käyttämällä avainalgoritmia. Salauskierrokset perustuvat yleensä joko substituutio-permutaatio verkkoon tai Feistel-salaukseen.

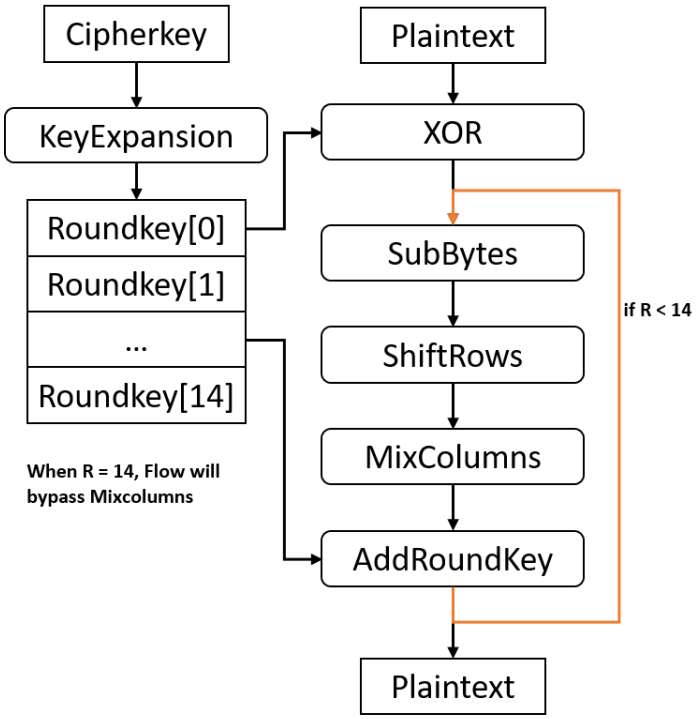
2.3 AES-128

Koska AES-128-algoritmi on lohkosalausalgoritmi, se sisältää lohkosalaukselle ominaiset piirteet, joita käsiteltiin alaluvussa 2.2. Se salaa dataa 128-bitin kokoisissa lohkoissa, joita se käsittelee 16:a tavusta koostuvana kaksiulotteisen matriisina (Aumasson, s.59). AES-128 hyödyntää substituutio-permutaatioverkkoa, joka ilmenee sen salauskierrosten rakenteessa (St. Denis & Johnson, 2007, s.142).

Salauskierros AES-128-algoritmissa on nelivaiheinen. Nämä vaihteet ovat (St. Denis & Johnson, 2007, s.142):

1. Tavujen substituutio (eng. SubBytes).
2. Rivien siirto (eng. ShiftRows).
3. Kolumnien sekoitus (eng. MixColumns).
4. Lisää kierrosavain (eng. AddRoundKey).

Kierroksen lisäksi algoritmiin kuuluu myös avainalgoritmi. Algoritmin toimintaa on kuvattu kuvassa 3. On huomattavaa, että 128-bittisessä versiossa kierrosmäärä R=10, ei 14 ja kierrosavaimia on vain 11 (Xilinix, 2020).



Kuva 3 - AES-algoritmin toiminta. (Xilinx, 2020)

2.3.1 Tavujen substituutio

Tavujen substituutiovaiheessa käsiteltävän lohkon kaikki 16 tavua korvataan joillain toisilla tavuilla. Tämän vaiheen tarkoitus on hajottaa algoritmin lineaarisuutta (St. Denis & Johnson, 2007 s. S.146). Substituutioon käytettävät arvot voidaan määritellä matemaattisesti, mutta käytännössä ajan säästämiseksi AES-128 implementaatiot käyttävät valmiiksi määriteltyä substituutiotaulukkoa (S-box, eng. substitution box). Tällöin korvattava tavu vaihdetaan vastaavaan tavuun S-boxista. Kuva 4 esittää AES-algoritmin substituutiotaulukkoa (St. Denis & Johnson, 2007, s.148).

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
Ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
Bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
Cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
Dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e

Kuva 4 - AES S-box (St. Denis & Johnson, 2007, s.148)

2.3.2 Rivien siirto

Rivien siirrossa lohkon kaksiulotteisen matriisin rivien arvoja siirretään 0,1,2 tai 3 paikkaa vasemmalle. Tämä voidaan toteuttaa uudelleennimeämällä tavuja, jolloin niitä ei tarvitse kirjaimellisesti siirtää. Tämä vaikuttaa tehokkuuteen positiivisesti. Prosessina rivien siirto on täysin lineaarinen (St. Denis & Johnson, 2007, s.150).

2.3.3 Kolumnien sekoitus

Kolumnien sekoittaminen on tärkeä osa AES-algoritmeja, joka varmistaa sen, että pienetkin muutokset salattavaan tekstiin vaikuttaa suuresti tuotettuun salatekstiin. Tämä suoritetaan yleensä matriisikertolaskulla, jossa lohkon kolumnit kerrotaan annetulla vakiomatriisilla. Tämä kertolasku on kuvattu kuvassa 5 jossa vasemmalla on vakiomatriisi MDS (maximum distance separable) ja oikealla yksi kolumneista, nimeltään a (St. Denis & Johnson, 2007, 151–152).

The AES MDS Matrix

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Kuva 5 - MDS · kolumni a (St. Denis & Johnson, 2007, s. 151)

Toisaalta kertolaskun sijasta voidaan jälleen käyttää hyväksi valmiiksi laskettuja taulukoita. Koska kolumnin a arvot a0-a3 ovat kaikki 8-bittisiä muuttujia, on mahdollista luoda taulukko (usein nimeltään Xtime), joka sisältää kaikki mahdolliset MDS-matriisin ja kahdeksan-bittisen muuttujan tulot. Tällöin säästetään laskennallista tehoa, koska kertolaskua ei enää tarvita (St. Denis & Johnson 2007, s.153–154). On siis huomioitava, miten kolumnien sekoitus on toteutettu luotettavan tehokkuusarvion saavuttamiseksi.

2.3.4 Lisää kierrosavain

Kierrosavaimen lisääminen salattavaan lohkoon on yksinkertainen XOR (eksklusiivinen disjunktio, eng. Exclusive or**)** toimenpide lohkon ja kierrosavaimen välillä. Tämä on se vaihe, jossa salausavaimesta johdettu kierrosavain vaikuttaa lohkoon. Avainalgoritmi muodostaa vaaditun määrän kierrosavaimia tätä toimenpidettä varten (St. Denis & Johnson, 2007, s.146).

2.3.5 Avainalgoritmi

Avainalgoritmin tehtävä on muuttaa annettu salausavain useammaksi kierrosavaimeksi. Kierrosavain on kooltaan 128-bittinen, riippumatta itse salausavaimen koosta, ja niitä vaaditaan yksi per kierros, sekä yksi ylimääräinen niiden ulkopuolelle. Nämä 128-bitiset kierrosavaimet ajatellaan koostuvan neljästä 32-bittisesta sanasta, joita avainalgoritmi

manipuloi. Algoritmi suorittaa sanan jokaiselle tavulle samanlaisen tavun substituution kuin salauskierros suorittaa salattavan lohkon tavuille. Lisäksi avainalgoritmi kiertää sanan tavujen järjestyksen RotWord-funktiolla. Substituution ja rotaation lisäksi joka kierrosavaimen merkitsevimmälle tavulle suoritetaan XOR operaatio Rcon-nimisen vakio taulukon seuraavan käyttämättömän arvon kanssa (St. Denis & Johnson, 2007, s.155–156). Kuva 6 esittää Rcon-taulukkoa.

```
static const unsigned char Rcon[10] = {
0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,  0x1B, 0x36 };
```

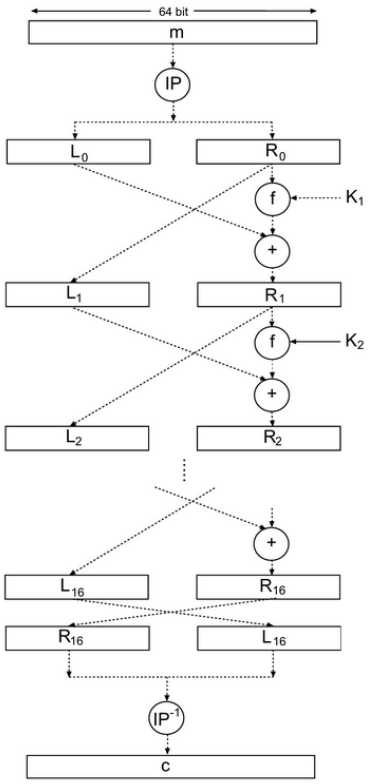
Kuva 6 - Rcon-taulukko (St. Denis & Johnson, 2007, s.155)

2.4 3DES

3DES-algoritmi kuuluu AES-128-algoritmin tavoin lohkosalausalgoritmien perheeseen. 3DES-algoritmi kehitettiin vastaamaan vanhentuneen DES-algoritmin turvallisuuspuutteisiin. Kyseessä on DES-algoritmi, joka yksinkertaisesti suoritetaan kolmesti; kahdesti salausmoodissa, kerran salauksen purkumoodissa. Joka suorituskerralla käytetään eri salausavainta, jolloin salauksen purkaminen ei toimi käänteisoperaationa toiselle salaustoiminnoista (Oppliger, 2005, s.252 & 254). Tässä luvussa selkeyden vuoksi 3DES tarkoittaa yhtä näistä kolmesta DES-algoritmin suorituksesta, joka on havainnollistettu kuvassa 7.

2.4.1 Feistel-salaus

AES-128:n ja 3DES:n suurin ero on niiden sisäisessä arkkitehtuurissa. Siinä missä AES-128 kierrättää bittisubstituutioita ja permutaatioita koko salattavalle 128-bittiselle lohkolle, Feistel-salauksessa salattava lohko, 3DES:n tapauksessa 64-bittinen lohko, jaetaan kahtia. Bittisubstituutiot ja permutaatiot tehdään sitten vain toiselle näistä lohkoista (Oppliger, 2005, s. 241–242). Tätä kuvataan kuvassa 7:



IP	Aloituspermutaatio
m	salattava viesti
L_i & R_i	Lohkon puolikkaat kierroksella i
K_i	Kierroksen i kierrosavain
f	Substituutio-permutaatiofunktio
IP^{-1}	Lopetuspermutaatio
c	Salateksti

Kuva 7 - 3DES-algoritmi, joka toistetaan kolmesti (Oppliger, 2005, s. 243)

Koska algoritmi suorittaa bittien substituutiot ja permutaatiot pääosin vain toiselle salattavan lohkon puolikkaalle, tärkeä osa Feistel-salausta on puolikkaiden vaihto, jossa kuvan 7 lohko L saa arvon R ja lohko R saa arvon L.

2.4.2 Aloituspermutaatio

Feistel-salauksessa salauskierros käsittelee kahta pientä lohkoa, yhden ison lohkon sijaan. Aloituspermutaation tehtävä on saattaa 64-bittinen salattava lohko tarvittavaan muotoon. Tämä suoritetaan jakamalla salattava lohko kahteen osaan, 32-bittisiksi L- ja R-lohkoiksi. Samalla niiden bittejä myös siirrellään eri järjestykseen, joka määräytyy esilasketusta taulukosta (Oppliger, 2005, 241–242).

2.4.3 Kierrosavainten muodostaminen

3DES sisältää kuusitoista salauskierrosta. Se tarvitsee yhden kierrosavaimen per salauskierros, eli yhteensä kuusitoista avainta. Tämä avainalgoritmi on irrallinen 3DES:n kierrosrakenteesta. 3DES nimellisesti käyttää 64-bittistä salausavainta, mutta käytännössä tästä avaimesta käytetään vain 56 bittiä kierrosavainten muodostamiseen (Oppliger, 2005, s.241 & 244).

Avainalgoritmi ottaa 64-bittisen salausavaimen ja typistää sen kahdeksi 28-bittiseksi avaimenpuolikkaaksi. Näiden avaimenpuolikkaiden biteille sitten suoritetaan erilaisia permutaatio- ja siirtotoimintoja. Kun nämä puolikkaat jälleen yhdistetään yhdeksi kierosavaimeksi, tämä yhteensä 56-bittinen kokonaisuus typistetään vielä uudelleen 48-bittiseksi. Näitä puolikkaita muokataan ja yhdistetään yhtensä kuusitoista kertaa, jotta 3DES saa tarvittavan määrän kierrosavaimia. Typistämistoimenpiteissä avainalgoritmi hyödyntää esilaskettuja taulukoita (Oppliger, 2005, s. 244–246).

2.4.4 Salauskierros

3DES salauskierros ottaa aloituspermutaatiossa muodostetut kaksi 32-bittistä lohkoa ja suorittaa niille samankaltaisia toimenpiteitä, joita myös AES-128 salauskierros suorittaa. Tämä tarkoittaa bittien substituutiota, permutaatiota ja kierrosavaimen käyttämistä (Oppliger, 2005, s. 242–243).

Ensimmäinen toimenpide salauskierroksessa on kuitenkin lohkon R_i laajentamisella 32-bitistä 48-bittiseksi. Tässä käytetään hyväksi laajennusfunktiota, jossa esilasketun taulukon mukaisesti joitain lohkon R_i bittejä toistetaan uudelleen (Oppliger, 2005, s.242). Laajennuksen jälkeen 3DES lisää kierrosavaimen laajennettuun R_i lohkoon.

Laajennuksen ja kierrosavaimen lisäämisen jälkeen 3DES suorittaa bittien substituution. AES-128:n tapaan tässä käytetään hyväksi esilaskettuja S-box-taulukoita. Kuitenkin 3DES:n tapauksessa näitä taulukoita on yhteensä kahdeksan kappaletta yhden sijaan ja

ne ovat typistäviä S-boxeja. Tämä tarkoittaa, että 3DES:n tapauksessa 6-bittinen syöte palauttaa S-boxista 4-bittisen tuloksen. Näin ollen, bittien substituutio palauttaa R_i-lohkon takaisin 32-bittiseksi (Oppliger, 2005, s.243–244).

Salauskierroksen lopuksi R_i-lohkon bitit järjestetään uudelleen vielä kerran. Kyseessä on salauskierroksen permutaatiovaihe. Se on aloituspermutaation kaltainen toimenpide, mutta siinä käytetään eri esilaskettua taulukkoa, eikä se jaa lohkoa puoliksi. Permutaation jälkeen R_i-lohkoa muokataan vielä lisäämällä siihen L_i-lohko XOR-toiminnolla. Lopuksi L-lohkon ja R-lohkon arvot vaihdetaan keskenään, jotta seuraavalla kierroksella tässä luvussa kuvatut toimenpiteet tehdään myös vanhalle L-lohkolle (Oppliger, 2005, s.243).

2.4.5 Lopetuspermutaatio

Kun jokainen salauskierros on suoritettu, 3DES-algoritmi suorittaa lopetuspermutaation. Sitä varten salauskierrosten kaksi lohkonpuolikasta yhdistetään jälleen 64-bittiseksi lohkoksi. Itse lopetuspermutaatio vastaa läheisesti aloituspermutaatiota, lohkon bitit järjestellään uudelleen esilasketun taulukon mukaan, mutta se ei enää jaa lohkoa kahteen osaan (Oppliger, 2005, s.244).

3DES-algoritmista on huomattavaa se, että se käyttää hyvin paljon erinäisiä esilaskettuja taulukoita. Lähes joka vaihe aloituspermutaatiosta lähtien käyttää omaa taulukkoansa. Nämä taulukot ovat listattuna liitteisiin.

2.5 Aiempaa tutkimusta

Aiemmat tutkimukset pyrkivät tyypillisesti mittaamaan algoritmin suoritusaikaa millisekunneissa. Toisin sanoen, suoritustehoa mitataan kellottamalla algoritmeja erikokoisilla syötteillä. Näin toimitaan ainakin Pandan (2016, s.283), Durranin & Hayan (2022) ja Radhakrishan, Jadon & Honnavallin (2024) artikkeleissa. Nämä artikkelit käsittelevät algoritmin tehokkuusanalyysiä, mutta ne eroavat huomattavasti tästä

tutkimuksesta. Ne ovat enemmänkin kiinnostuneita algoritmien suoritustehosta millisekunneissa mitattuna. Tällainen mittaustekniikka riippuu varmasti ulkoisista tekijöistä, sillä algoritmien suorittaminen nopeammalla tietokoneella vaikuttaisi varmasti mittaustulokseen. Näin ollen, kahden artikkelin tuloksien vertailu voi osoittautua hankalaksi. Tämä tutkimus lähestyy aihetta abstraktimmasta näkökulmasta, jolloin kyseinen ongelma tuskin vaikuttaa sen tuloksiin.

Rosyidin ja Sarin (2016) artikkelissa puolestaan hyödynnetään algoritmin suoritustehon analyysissä hieman samankaltaista metodologiaa, kuin tässä tutkimuksessa. Artikkelin käyttökonteksti ei kuitenkaan ole sama, vaan se käsittelee autonomisen IoT-järjestelmän analysointia.

Käyttökontekstin huomioiden, IoT (internet of things) vaikuttaa olevan suosittu algoritmianalyysin konteksti. Rosyidin ja Sarinin (2016) lisäksi IoT:n aihepiiriin liittyy myös esimerkiksi Iftikharin ja muiden (2021) artikkeli, sekä Radhakrishan, Jadon & Honnavallin (2024) artikkeli. Salausalgoritmien suoritustehokkuus on selkeästi tärkeä aihe IoT:hen liittyen. Onhan kyse verkosta, joka koostuu pääosin pienistä, haavoittuvista laitteista kuten sensoreista (IBM, 2025).

On havaittavissa, että algoritmianalyysiin liittyvät artikkelit ovat tyypillisesti itsenäisiä analyysejä eri algoritmeista. Ne eivät niinkään käsittele itse analyysimetodia, kuten tämä tutkimus. Tällainen näkökulmanvaihto erottaa tämän tutkimuksen aiemmista tutkimuksista.

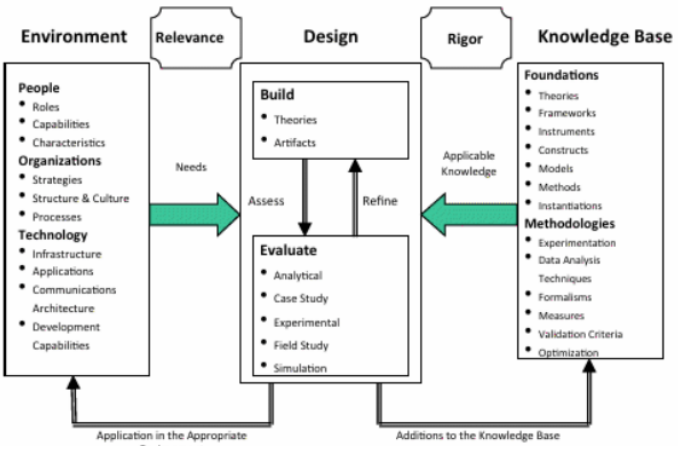
3 Suunnittelutieteellinen tutkimus

Tämän tutkimuksen tavoitteena on tuottaa ohjaava dokumentti lohkosalausalgoritmien suoritustehoanalyysiä varten. Tavoite on selkeä, mutta se ei ole niinkään numeraaliseen dataan perustuva tavoite, eikä myöskään arvopohjainen selitys millekään yksittäiselle ilmiölle. Näin ollen Vilkan (2021, sanasto) määritelmät määrällisestä ja laadullisesta tutkimuksesta eivät täysin kuvaa tutkimuksen tavoitetta.

Tämä tutkimus on tyypiltään suunnittelutieteellinen tutkimus. Vom Brocke ja muut (2020, s.1–4) määrittelevät suunnittelutieteellisen tutkimuksen tutkimukseksi, jonka tavoitteena on tuottaa uusi artefakti ratkaisuksi käytännölliseen ongelmaan käyttämällä iteratiivista prosessia. Tämä luku käsittelee kyseistä tutkimusmenetelmää. Luku 3.1 esittelee tutkimusmenetelmän rakennetta ja teoriaa. Luku 3.2 käsittelee tutkimusaineistoa ja sen keräämistä.

3.1 Suunnittelutieteellisen tutkimuksen teoriaa

Vom Brocke ja muut (2020, s. 3) määrittelee suunnittelutieteelliselle tutkimukselle kolme tärkeää osaa: Ympäristön, suunnittelun ja tietämyskannan (eng. knowledge base). Tätä havainnollistaa kuva 8.



Kuva 8 - Suunnittelutieteellinen tutkimus (Vom Brocke ja muut, 2020)

Ympäristö muodostuu ihmisistä, organisaatiosta ja teknologiaympäristöstä. Tämä laajempi konteksti määrittelee tarpeen, johon tutkimus vastaa (Vom Brocke ja muut, 2020, s.2). Tämän tutkimuksen tapauksessa tarve määräytyy pitkälti teknologisen kontekstin myötä, sillä tutkimusta ei tehdä ulkopuolisten organisaatioiden tai henkilöiden suorasta toiveesta. Tämä teknologiaympäristöön perustuva tarve esitettiin luvussa 1, jossa tietoteknisten laitteiden levinneisyys ja tarve datan suojaamiselle muodostavan syyn salauksen kehitykselle.

Vom Brocken (2020, s. 2;) ja muiden, sekä Gregorin ja Hevnerin (2013, s. 350) mukaan tietämyskanta on kokoelma relevanttia teoriaa ja metodologiaa. Tässä tutkimuksessa tietämyskantana käytetään kirjallisuutta symmetrisestä lohkopohjaisesta salauksesta sekä algoritmianalyysistä. Tutkimuksen artefakti perustuu tähän tietämyskantaan.

Keskeisin osa suunnittelutieteellistä tutkimusta on itse suunnitteluprosessi. Suunnitteluprosessi on iteratiivinen prosessi, jossa tunnistetaan ongelma, asetetaan tavoite, suunnitellaan artefakti, suoritetaan artefaktidemonstraatio, artefaktin arviointi ja lopulta julkaisu. Tätä prosessia toistetaan tarpeen mukaan (Vom Brocke ja muut, 2020, s.3–5; Hevner, 2007, s.4). On huomattavaa, että uusi iteraatio voidaan aloittaa jo ennen julkaisuvaihetta, jos se arvioidaan tarpeelliseksi. Tämän tutkimuksen tapauksessa suunnitteluvaiheen tulos esitellään luvussa neljä.

Koska itse tutkimus seurasi edellä mainittua sykliä, tutkimuksen raportti vastaa rakenteeltaan suunnittelutieteellistä suunnitteluprosessia. Luku yksi vastasi ongelmanmäärittelyvaihetta ja tavoitteenasetusta. Luku neljä puolestaan kuvaa artefaktin suunnittelua. Luku viisi, jossa kehitettyä artefaktia sovelletaan AES-128- ja 3DES-algoritmiin, vastaa puolestaan artefaktidemonstraatiota. Luku kaksi, joka käsittelee tutkimuksen teoriapohjaa, vastaa puolestaan suunnittelutieteellisen tutkimuksen taustalla vaikuttavaa tietämyskantaa, sillä luvussa käsitellään relevanttia teoriaa ja aiempaa metodologiaa. Suunnittelutieteellisen tutkimusmenetelmän heikkous tämän

tutkimuksen kontekstissa on se, että rajallinen aikataulu sekä laajuus rajoittavat suunnittelun iteraatioita sekä artefaktin testausprosessia.

Suunnittelutieteellinen tutkimus on menetelmänä kuitenkin hyödyllinen tämän tutkimuksen kontekstissa. Hevnerin (2007, s.5) mukaan suunnittelutieteellisen tutkimuksen etu on sen pragmaattisuus. Tutkimustyyppinä se pyrkii vastaamaan käytännöllisellä tavalla käytännölliseen ongelmaan, joten sen kontribuutio tieteelle on tyypillisesti hyvin konkreettinen. Tämä on linjassa myös tämän tutkimuksen tavoitteiden kanssa tuottaa käytännöllinen suoritustehon analyysimenetelmä.

Huomioiden tutkimuksen tavoitteen, Gregorin ja Hevnerin (2013, s. 345) kuvauksen mukaisesti tutkimuksen kontribuutio on kehitystyyppinen (eng. improvement). Artefaktin käyttöympäristö on suhteellisen tunnettu, mutta tutkimus kuvaa uuden tavan lähestyä valittua tutkimusongelmaa. Gregorin ja Hevnerin (2013, s. 346) mukaan suunnittelutieteelliset tutkimukset tietotekniikan kuuluvatkin tyypillisesti tähän kastiin.

3.2 Tutkimusaineisto

Tutkimuksen aineisto koostuu algoritmianalyysiä ja salausalgoritmeja koskevista alan julkaisuista ja teoksista, jonka pohjalta tutkimuksen artefakti muodostettiin. Julkaisuja etsittiin pääosin Elsevierin Science Direct tietokannasta, IEEE Xploresta, Google Scholarista sekä tiedekirjasto Tritonian Finna-hakukoneesta.

Hakua myös rajattiin eri tavoilla. Artikkeleita ja teoksia etsittiin lähtökohtaisesti aikavälilta 2010–2025. Jotkin aineistoon kuuluvat teokset tai artikkelit saattoivat myös olla ennestään tuttuja, kuten Cormenin ja muiden Introduction to algorithms (2009), jolloin ajallinen rajaus ohitettiin.

Artikkeleille suoritettiin taustatutkimusta, jossa pyrittiin selvittämään niiden luotettavuus ja kelpoisuus lähteeksi. Taustatutkimuksessa artikkeleita julkaisseita tahoja taustoitettiin Julkaisufoorumi.fi -verkkosivun avulla. Julkaisufoorumi on yksinkertainen

tapa arvioida lähteen luotettavuutta sen asettamien tasoluokitusten avulla (Julkaisufoorumi, 2024).

Aineistoa hyödynnettiin tutkimuksessa nimenomaan ohjaamaan suunnittelutieteellisen artefaktin luomista. Koska kyseessä on toimintamalli suoritustehon analyysille, aineistosta kerättiin tietoa yleisesti analyysiprosessien suorittamisesta sekä siitä, miten suoritustehoa voidaan analysoida. Tämän lisäksi lohkosalausalgoritmeista kerättiin tietoa, jotta ehdotettu analyysimenetelmä lähestyisi analyysiprosessia järjestelmällisesti. Toisin sanoen, aineistosta selvitettiin, miten suoritustehoa voidaan mitata algoritmeissa, miten analyysejä tyypillisesti suoritetaan aihepiiristä riippumatta ja miten tätä voidaan soveltaa nimenomaan lohkosalausalgoritmeihin tavalla, joka takaa käyttökelpoisen lopputuloksen.

4 Artefakti: lohkosalausalgoritmin analysointi

Tässä luvussa käsitellään symmetrisen salausalgoritmin suoritustehokkuuden arviointia. Luku on suunnittelutieteellisen tutkimuksen artefakti, jota hyödyntämällä lukija voi muodostaa analyysejä eri lohkosalausalgoritmien suoritustehokkuuksista. Alaluku 4.1 kuvaa tarvittavaa taustatietoa analyysin suorittamisesta yleisesti ja taustoittaa käsiteltävien algoritmien rakennetta. Alaluku 4.2 käsittelee itse analyysistrategiaa ja opettaa, miten muodostetaan analyysi lohkosalausalgoritmin lohkolle. Alaluku 4.3 käsittelee lohkon analyysin ekstrapolointia koko algoritmille. Lopuksi alaluku 4.4 sisältää yhteenvedon ehdotetusta analyysimenetelmästä.

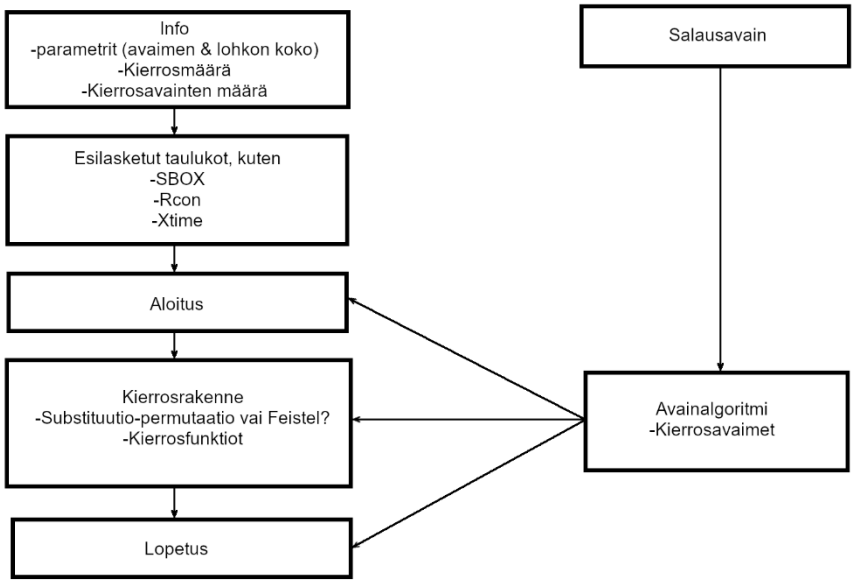
4.1 Taustatietoa arviointimallille

Elo ja Kyngäs (2008, s. 109) määrittelevät yhdeksi analyysimenetelmäksi deduktiivisen analyysin. Vaikka he eivät käytäkään termiä algoritmien kontekstissa, on heidän määritelmänsä kuitenkin käyttökelpoinen. Deduktiivinen analyysi operoi ottamalla ilmiötä ympäröivää laajaa teoriaa ja hyödyntämällä sitä selittämään yksilöllisempää ilmiötä. Käytännössä prosessia kuvaillaan kolmivaiheiseksi; analyysi alkaa valmistelulla, johon liittyy ongelmanmäärittely. Sen jälkeen analysoitava data kategorisoidaan jollain määrätyllä tavalla. Lopuksi tästä kategorisoidusta joukkiosta johdetaan yhteyksiä tai päätelmiä, joita analyysi vaatii. Tämä analyysimalli toimii ideologisena pohjana tutkimuksen artefaktille. Kuva 9 edustaa prosessia visuaalisesti.

Kuva 9 - Analyysiprosessi (Elo ja Kyngäs, 2008, s. 109)

Toinen ideologinen pohja, johon analyysimenetelmä nojaa, on algoritmien kontekstissa tunnettu hajota ja hallitse -malli (eng. Divide-and-conquer). Kyseisessä mallissa algoritmi jakaa ongelman aina pienempiin ja pienempiin osiin, jotka on helpompi ratkaista kuin itse alkuperäinen ongelma. Lopuksi ratkaisut näihin pienempiin ongelmiin yhdistetään muodostamaan ratkaisu alkuperäiseen ongelmaan (Cormen ja muut, 2009, s.65). Ehdotettu salausalgoritmin arviointimalli perustuu selkeästi myös tähän ajatukseen.

Analyysin kannalta on myös tärkeää ymmärtää perusteet lohkosalausalgoritmin rakenteesta. Kuten luvussa 2 esitettiin, symmetrisillä lohkosalausalgoritmeilla on tiettyjä säännöllisyyksiä. Kyseessä on algoritmi, joka salaa viestin algoritmille ominaisissa lohkoissa, käyttämällä hyväkseen kevyitä salauskierroksia toistuvasti samalle lohkolle (Aumasson, 2018, s.53 & 55, s. 55–56). Tätä kaavamaisuutta kuvataan kuvassa 10.



Kuva 10 - Symmetrisen lohkosalausalgoritmin yleinen rakenne

Kuva 10 havainnollistaa algoritmin rakennetta tarkemmin. Ensin, algoritmista voidaan kerätä alustavaa informaatiota. Esimerkiksi, algoritmi ottaa tietynkokoisen lohkon salattavaksi, sekä salausavaimen avainalgoritmia varten. Lisäksi algoritmi suorittaa tietyn vakiomäärän salauskierroksia lohkolle, sekä tuottaa kierrosavaimia tätä toimintoa varten.

Lohkolla, salausavaimella ja kierrosavaimella on biteissä mitattava koko ja kierroksilla sekä kierrosavaimilla on jokin suoritettava lukumäärä.

Lohkosalausalgoritmissa saattaa myös olla joitain esimääriteltyjä taulukoita salaamista varten. Näin on esimerkiksi AES-algoritmeilla, joilla salauskierrosten funktioita pyritään korvaamaan näillä taulukoilla algoritmin suoritustehon optimointia varten. On kuitenkin huomattavaa, että näiden taulukoiden säilöminen vaatii tietokoneilta muistia, joten niiden käyttö on tasapainottelua suoritustehokkuuden ja muistin käytön välillä. Kaikki algoritmin implementaatiot eivät siis välttämättä käytä samoja määriä taulukoita. Esimerkiksi AES-algoritmeissa näitä taulukoita voidaan tyypillisesti korvata myös ylimääräisillä funktioilla, jotka taulukosta etsinnän sijaan laskevat vaaditut tulokset algoritmin suorituksen aikana (St. Denis & Johnson, 2007, s. 166).

Aloitus ja lopetusvaiheet kuvaavat yksinkertaisesti rakenteita, jotka kuuluvat algoritmiin, mutta toimivat algoritmin kierrosrakenteen ja avainalgoritmin ulkopuolella. Mitä tämä tarkoittaa, riippuu täysin käsiteltävästä algoritmista.

4.2 Analyysin suorittaminen

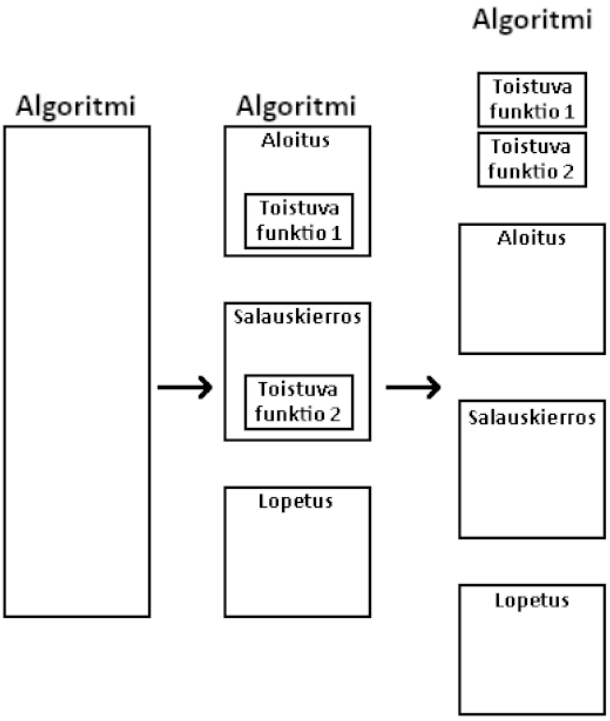
Tämä luku esittelee ehdotetun arviointiprosessin, ja siihen kuuluvat päätökset, joita analysoijan on tehtävä. Luku 4.2.1 esittelee analyysistrategian korkealla tasolla. Luku 4.2.2 puolestaan käsittelee analyysin ongelmanmäärittelyä. Luku 4.2.3 käsittelee analyysin alkuvaiheita ja luku 4.2.4 selittää, miten algoritmin vakioaikaisia toimintoja voidaan arvioida järjestelmällisesti.

4.2.1 Analyysistrategia

Algoritmin eri komponentit eivät välttämättä ole koodin ajon kannalta kronologisessa järjestyksessä, riippuen tavasta, jolla analysoitava koodi tai pseudokoodi on jäsennelty. Näin ollen, jotta analyysi voidaan suorittaa tehokkaasti, on syytä pohtia miten algoritmin rakenne vaikuttaa analyysiin. Koska lohkosalausalgoritmit ovat rakenteeltaan

suhteellisen kaavamaisia, voi olla hyödyllistä tutkia eri kuvan 10 määrittelemiä algoritmin osia, edeten yksitellen osasta osaan. Näiden osuuksia ovat kuvan 10 mukaisesti esimerkiksi aloitus, kierrosrakenne ja lopetus.

Myös eri funktiot, joita algoritmi hyödyntää useaan kertaan koodin ajon aikana on saatettu eristää laajemmista rakenteista. Näitä funktioita sitten kutsutaan koodissa tarpeen mukaan. St. Denisin ja Johnsonin (2007, s. 157) mukaan niin kutsuttu Xtime-funktio AES-algoritmin sisällä on esimerkki tällaisesta toistuvasti kutsuttavasta funktiosta. Tällaisten osuuksien ja funktioiden arviointi voi myös olla hyvä lähtökohta, jos laajempi koodi on rakennettu hyvin modulaariseksi. Tällainen lähestymistapa hajottaa algoritmia pienempiin osiin arviointia varten, joista voidaan sittemmin koota laajempia kokonaisuuksia. Kuva 11 esittää tätä lähestymistapaa.



Kuva 11 - Algoritmin hajottaminen osiin

Kuvassa 11 algoritmi hajotetaan ensin osiin. Tämän jälkeen näiden osien sisältä etsitään toistuvia funktioita, kuten Xtime-funktio AES-algoritmeissa. Analysoimalla nämä funktiot etukäteen, säästetään työtä eri osien sisällä, joissa näitä funktioita sittemmin käytetään. Kun eri osien analyysit on suoritettu, ne voidaan laskea lopulta yhteen. Tästä toimintatavasta ilmenee hajota ja hallitse -ajattelun vaikutus analyysiin, ja kuvaa korkealla tasolla tutkimuksen ehdottamaa analyysimallia. Se peilaa myös deduktiivisen analyysin kategorisointivaihetta, jossa algoritmin hajottaminen osiin vastaa analysoitavan datan kategorisointia.

4.2.2 Analyysin aloittaminen - Ongelmanmäärittely

Kun analyysistrategia on esitelty, on vuorossa ehdotetun analyysiprosessin tarkempi tarkastelu. Tässä voidaan hyödyntää Elon ja Kyngäksen (2008, s.109) deduktiivisen analyysin toimintakaavaa määrittelemään analyysin eri työvaiheita. Deduktiivinen analyysi koostuu muutamasta eri vaiheesta, alkaen valmistelulla. Valmisteluvaiheessa määritellään itse analyysin kohde, eli mitä tutkimuksessa mitataan ja miten tarkasti.

Tutkimuksen analyysi kohdistuu vakioaikaisiin toimintoihin. Kuitenkin ennen itse algoritmianalyysin aloittamista, on syytä miettiä tarkoin, mitä analyysiltä vaaditaan. Analyysin vaadittu tarkkuus on yksi merkittävä mittaukseen vaikuttava tekijä, joka riippuu suoritettavan analyysin käyttökohteesta. Kuten luvussa 2 todettiin, analyysissä yleensä tutkitaan asymptoottisesti kasvavan syötteen n vaikutusta suoritusaikaan. Tämä mahdollistaa arvion pelkistämisen, sillä kasvuun vaikuttavat pienemmät tekijät voidaan jättää huomiotta (Cormen ja muut, 2009, s.28. & Alsuwaiyel, 2016, s. 22). Tämä tietysti helpottaa arvioprosessia, mutta analyysin tarkkuusvaatimus ei kuitenkaan välttämättä vastaa aina tätä oletusta. Mietitään tilannetta, jossa tavoitteena on tunnistaa useamman ehdokkaan joukosta suoritusteholtaan paras algoritmi. Onko tällöin hyväksyttävää käyttää pelkistettyjä arvioita, ilman nyanssia? Analyysin vaadittu tarkkuus on syytä määritellä ennalta, ennen itse käytännön analyysin aloittamista, jotta analyysin tulos vastaa oikeasti tehtävänantoon.

Valmisteluun kuuluu myös analysoitavan datan ymmärtäminen. On syytä ymmärtää, mitä analysoidaan, miksi ja miten analysoinnin kohde toimii (Elo & Kyngäs, 2008, s. 109). Tällöin harkittavana on esimerkiksi lohkosalausalgoritmin rakenne, sillä tutkimuksen esittämä analyysimenetelmä perustuu lohkosalausalgoritmien rakenteellisiin samankaltaisuuksiin, joten on syytä varmistua, että arvioitava algoritmi sopii kyseiseen muottiin.

4.2.3 Algoritmin osien eristäminen

Deduktiivisen analyysin näkökulmasta valmisteluvaihe on suoritettu ongelmanmäärittelyn yhteydessä. Seuraava vaihe on analysoitavan datan kategorisointi (Elo & Kyngäs, 2008, s.109). Ehdotetun analyysistrategian kategorisointivaihe perustuu hajota ja hallitse -malliin. Näin ollen, ensimmäinen vaihe itse analyysissä on eristää kuvan 11 mukaisesti algoritmin eri vaiheet, kuten aloituksen, kierrosrakenteen, lopetuksen ja avainalgoritmin. Nämä osat arvioidaan omina osuuksinaan, joiden analyysit yhdistetään myöhemmin. Lohkosalausalgoritmi voi käyttää hyväkseen erilaisia esilaskettuja taulukoita tai toistuvia funktioita (St. Denis & Johnson, 2007, s. 165–166), joten niiden tunnistaminen ja eristäminen hajottaa alkuperäisen ongelman vielä pienempiin osiin. Algoritmin osien eristämisen jälkeen voidaan siirtyä hajota ja hallitse -strategian seuraavaan vaiheeseen, osien analysointiin.

4.2.4 Vakioaikaisten toimintojen laskeminen

Itse analyysi aloitetaan algoritmin osien erittelyn jälkeen. Kuten aiemmin esitettiin, analyysin tavoitteena on laskea vakioaikaisten toimintojen määriä, joita algoritmin koodissa esiintyy. Oletetaan, että tällaiset toiminnot todellakin vievät saman ajan arvion yksinkertaistamiseksi. Tässä luvussa vakioaikaisia toimintoja voidaan merkitä merkillä Ci, jossa i on toiminnolle annettu järjestysnumero. Joitain vakioaikaisia toimintoja voidaan myös toistaa koodissa useita kertoja silmukoiden muodossa. Tällöin, silmukassa määritellään jollain loogisella testillä kierrosmäärä tj, kun j esittää testin sisältämää muuttujaa, jolla kierrosmäärä määritellään (Cormen ja muut, 2009, s.25).

Cormen ja muut (2009, s.26) antavat esimerkin arviointiprosessista lisäysjärjestämisen muodossa. Kyseessä on siis algoritmi, joka järjestää annetun luvuista koostuvan taulukon A suuruusjärjestykseen. Taulukko koostuu n-määrästä lukuja. Esimerkin pseudokoodi on toistettu kuvassa 12.

```
1    for j = 2 to A.length
2        key = A[j]
3        //insert A[j] into sorted sequence A[1...j-1]
4        i = j-1
5        while i>0 and A[i] > key
6            A[i+1] = A[i]
7            i = i-1
8        A[i+1] = key
```

Kuva 12 - Lisäysjärjestäminen (Cormen ja muut, 2009, s.26)

Cormenin ja muiden analyysi kuvan 12 algoritmista kuvataan taulukossa 1:

Rivinumero	Kustannus	Määrä
1	C1	n
2	C2	n-1
3	0 (kommentti)	0
4	C4	n-1
5	C5	$\sum_{j=2}^{n} \mathrm{tj}$
6	C6	$\sum_{j=2}^{n} (\mathrm{tj} - 1)$
7	C7	$\sum_{j=2}^{n} (\mathrm{tj} - 1)$
8	C8	n-1

Taulukko 1 - Analyysiesimerkki (Cormen ja muut, 2009, s.26)

Taulukossa 1 rivinumero vastaa kuvan 12 pseudokoodin rivinumeroita. Kustannus vastaa rivissä esiintyvää vakiokustannusta. Määrä puolestaan kuvaa, kuinka monesti tämä rivin vakiokustannus esiintyy, kun se toistetaan n kertaa. Tämän algoritmin analysoinnissa tehdään pahimman tapauksen oletus.

Taulukon 1 ensimmäisellä rivillä tämä tarkoittaa, että for-silmukassa tapahtuva j-muuttujan vertailu, joka kestää vakioajan C1, tehdään yhteensä n kertaa. Tämä n siis ilmoittaa taulukon A sisältämien lukujen määrän, eli algoritmin syötteen koon. Näin ollen, rivin n aikavaatimus on siis $C1 \cdot n$. Syötteen koon vaikutus algoritmin suoritusaikaan nähdään välittömästi rivissä 1, sillä syötteen n koon kasvaessa tapahtuu myös C1 operaatioita suhteessa kasvava määrä.

Rivit 5,6 ja 7 käyttäytyvät hieman eri tavalla kuin muut rivit. Rivillä 1 on silmukka, jossa tehdään yksi vakiotoiminto per syötteen A luku. Riveillä 5,6 ja 7 on silmukka, joka tapahtuu jokaiselle syötteen luvulle erikseen. Kyseessä on siis määrällisesti n-1 (koska j = 2) silmukan summa, eli silmukoita silmukan sisällä.

Nyt, kun pseudokoodin eri rivit on analysoitu, ne voidaan koota kokonaiseksi funktioksi. Cormen ja muut (2009, s.26) esittävät tätä pahimman tapauksen funktiota T(n):

$$T(n) = C1n + C2(n-1) + C4(n-1) + C5\sum_{j=2}^{n} \mathrm{tj} + C6\sum_{j=2}^{n}(\mathrm{tj}-1) + C7\sum_{j=2}^{n}(\mathrm{tj}-1) + C8(n-1)$$

On huomattavaa, että rivien 5,6, ja 7 määrät muodostavat aritmeettisen sarjan, jotka ovat ratkaistavissa seuraavanlaisesti (Cormen ja muut, 2009, s.27 & s.1146):

1) $\sum_{j=2}^{n} \mathrm{tj} = \frac{\mathrm{n(n+1)}}{2} - 1$
2) $\sum_{j=2}^{n}(\mathrm{tj}-1) = \frac{\mathrm{n(n-1)}}{2}$

Nämä voidaan sijoittaa takaisin yhtälöön T(n) korvaamaan summafunktiot:

$$T(n) = C1n + C2(n-1) + C4(n-1) + C5\left(\frac{\mathrm{n(n+1)}}{2} - 1\right) + \mathrm{C6}\left(\frac{\mathrm{n(n-1)}}{2}\right) + \mathrm{C7}\frac{\mathrm{n(n-1)}}{2} + C8(n-1)$$

Josta kertomalla termit auki saadaan:

$$T(n) = C1n + C2n - C2 + C4n - C4 + + \frac{\mathrm{C5n^2}}{2} + \frac{\mathrm{C5n}}{2} - \mathrm{C5} + \frac{\mathrm{C6n^2}}{2} - \frac{\mathrm{C6n}}{2} + \frac{\mathrm{C7n^2}}{2} - \frac{\mathrm{C7n}}{2} + \mathrm{C8n} - \mathrm{C8}$$

Kuten alaluvussa 2.1.3 todettiin, tavoitteena on arvioida funktion kasvun voimakkuutta, joten edellä mainittua funktiota on syytä sieventää muuttujan n mukaan:

$$T(n) = \frac{\mathrm{C5n^2}}{2} + \frac{\mathrm{C6n^2}}{2} + \frac{\mathrm{C7n^2}}{2} + \mathrm{C1n} + \mathrm{C2n} + \mathrm{C4n} + \frac{\mathrm{C5n}}{2} - \frac{\mathrm{C6n}}{2} - \frac{\mathrm{C7n}}{2} + \mathrm{C8n} - \mathrm{C2} - \mathrm{C4} - \mathrm{C5} - \mathrm{C8}$$

Eli:

$$T(n) = \left(\frac{\mathrm{C5}}{2} + \frac{\mathrm{C6}}{2} + \frac{\mathrm{C7}}{2}\right)\mathrm{n^2} + \left(\mathrm{C1} + \mathrm{C2} + \mathrm{C4} + \frac{\mathrm{C5}}{2} - \frac{\mathrm{C6}}{2} - \frac{\mathrm{C7}}{2} + \mathrm{C8}\right)\mathrm{n} - (\mathrm{C2} + \mathrm{C4} + \mathrm{C5} + \mathrm{C8})$$

On nähtävissä, että algoritmin suoritusnopeus on ilmaistavissa funktiossa, joka on muotoa $an^2 + bn + c$, eli kyseessä on toisen asteen yhtälö. T(n) on algoritmin tarkka suoritustehoanalyysi. Seuraavaksi, yhtälö voidaan muuntaa yksinkertaiseen vertailukelpoiseen muotoon hyödyntämällä asymptoottista notaatiota.

Funktio T(n) on rajattavissa jollain toisen asteen funktiolla g(n). Tämä on havaittavissa siitä, että funktion T(n) muuttujan n suurin potenssi on 2. Alaluvun 2.1.4 mukaisesti tämä tarkoittaa sitä, että on olemassa jokin toisen asteen funktio g(n), jolla voidaan ylhäältä päin rajoittaa funktiota T(n) käyttämällä jotain kerrointa c siten, että jostain syötteen koon n lähtöarvosta n0 lähtien:

$$\mathrm{c} \cdot \mathrm{g(n)} \geq T(n)$$

Koska asymptoottisessa notaatiossa vain T(n)-funktion isoimmalla termillä on merkitystä (Cormen ja muut, 2009, s.47), riittää siis, että

$$c \cdot g(n) \geq \left(\frac{C5}{2} + \frac{C6}{2} + \frac{C7}{2}\right) n^2$$

alkaen jostain lähtöarvoa n0. Koska C5, C6 ja C7 ovat joitain vakioita, tämän vakioista koostuvan kertoimen arvo itsessään on jokin vakio, jota voidaan tässä tapauksessa merkitä termillä a. Tällöin funktion T(n) johtava termi on muotoa an^2, jossa a on jokin vakio. Tällöin,

$$\mathrm{c} \cdot \mathrm{g(n)} \geq an^2$$

kun c ≥ a, jostain sopivasta syötteen koosta n0 alkaen.

On huomattavaa, että C1-C8 kuvaavat jotain askelmäärää, jonka kyseinen rivi ohjelmasta suorittaa (Cormen ja muut, 2009, s26). Se, mitä tämä tarkoittaa on hieman häilyvää. Käytännössä, asymptoottisessa analyysissä ei arvioida tehokkuutta yksittäisen toiminnon tarkkuudella, joten yksittäisen kustannuksen arvolla ei ole suurta merkitystä (Cormen ja muut, 2009, s.28). Ei siis ole juurikaan väliä, edustaako jokin C-arvo vaikkapa yhtä, kahta tai useampaa perustoimintoa. Jotta funktiolle T(n) voidaan selvittää rajaava funktio g(n), voidaan kunkin kustannuksen C1-C8 olevan vakiota edustava arvo 1. Sijoitetaan tämä funktioon T(n). Saadaan:

$$T(n) = an^2 = \left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2}\right) n^2 = 1.5n^2$$

On selvää, että funktio T(n) rajautuu n^2-kokoluokan funktiolla g(n), mutta nyt on myös helppo määritellä arvot c ja n0. Jos $g(n)=n^2$, voidaan valita esimerkiksi c = 3, jolloin:

$$c \cdot g(n) \geq T(n) => cn^2 \geq an^2 => 3n^2 \geq 1.5n^2, kun\ n0 \geq 1$$

Tästä voidaan todeta, että tämän algoritmin suoritustehokkuutta voidaan rajata ylhäältä termillä n^2, joten $T(n) = O(n^2)$.

Analyysiprosessi matalalla tasolla tapahtuu siis asettamalla algoritmin toiminnoille kustannuksia ja laskemalla, kuinka monesti nämä toiminnot suoritetaan. Tämän jälkeen muodostetaan funktio T(n), ja sievennetään se mahdollisimman pitkälle. Lopuksi valitaan suurin termi ja määritellään asymptoottinen yläraja sen perusteella.

4.3 Lohkon analyysin ekstrapolointi laajemmalle algoritmille

Luvun 4.2 analyysiesimerkissä on kyse lisäysjärjestämisestä. Tällaisessa algoritmissa syötteen n koko voi olla periaatteessa ääretön. Lohkosalausalgoritmin tapauksessa tapaus on hieman erilainen. Lohkosalausalgoritmissa ehdotettu analyysimenetelmä kohdistuu yhteen vakiomittaiseen lohkoon. Luvun 2 mukaisesti lohkon kokoa merkittiin vakiolla b, joka on ominainen kullekin lohkosalausalgoritmille. AES-algoritmien tapauksessa b = 128 bittiä (Aumasson, s.59). Tällöin, salatakseen koko syötteen, vaadittu lohkojen lukumäärä on loogisesti $\frac{n}{\mathrm{b}}$, jossa n on salattavan tekstin, eli syötteen, kokonaispituus. Tämä voi vaikuttaa asymptoottisen notaation hyödyntämiseen, sillä oletus äärettömyydestä ei välttämättä päde yksittäisen lohkon kohdalla, sillä lohkon sisäisen syötteen b arvo on vakio.

Oletetaan esimerkiksi, että lohkon suoritustehokkuus T(b) on jokin vakio x. Vastauksen x on oltava vakioarvo, koska tässä esimerkkitapauksessa funktio T(b) on vain kokoelma

määrättyjä laskutoimenpiteitä vakioarvolla b. Toisin sanoen funktion laskutoimenpiteissä ei siis ole äärettömyyttä tavoittelevaa muuttujaa n, jolloin vastauksessakaan ei voi olla kyseistä muuttujaa. Salausalgoritmin suoritustehokkuus koostuu joukosta näitä lohkoja, joita tarvitaan yhteensä $\frac{n}{\mathrm{b}}$ kappaletta. Tällöin lopullinen analyysi T(n), jonka syöte onkin vakion sijasta äärettömyyttä lähentelevä muuttuja n, on muotoa:

$$T(n) = \frac{n}{\mathrm{b}} T(b)$$

Tästä voidaan loogisesti päätellä, että lohkosalausalgoritmin suoritustehokkuus lähentelisi muotoa T(n) = O(n). Tämä ei kuitenkaan ole aina täysin paikkansapitävää. Esimerkiksi 3DES-algoritmi on salausalgoritmi, jossa DES-lohkosalaus yksinkertaisesti toistetaan kolme kertaa samalle salattavalle tekstille (Dibas & Sabri, 2021, s.345). Tämä vaikuttaa suoritustehokkuuteen (Dibas & Sabri, 2021, s.348). On siis huomattavaa, että asymptoottisessa näkökulmasta merkitykselliset suoritustehoerot eivät välttämättä ilmene lohkon sisällä, vaan siinä, miten algoritmi näitä lohkoja käyttää.

Tämä havainto vaikuttaa myös ehdotettuun analyysimenetelmään. Jos algoritmianalyysi suoritetaan vain yhdelle lohkolle, ei sen tulos heijasta koko algoritmin toimintaa, koska lohkoja voi tarvita useita jollekin syötteen n koolle. Samalla, asymptoottisuuden olettamusta ei voi hyödyntää kunnolla, sillä yksittäisen lohkon koko ei lähesty ääretöntä. Täten, lohkon analyysi on syytä ekstrapoloida heijastamaan koko algoritmin suoritustehokkuutta, jolloin myös oletus asymptoottisuudesta voi pitää paikkaansa.

Kuten mainittua,

$$T(n) = \frac{n}{\mathrm{b}} T(b)$$

kuvaa kohtuullisesti tätä ekstrapolaatiota tapauksessa, jossa lohkoja yksinkertaisesti käytetään tarpeellinen määrä kattamaan syötteen koko.

4.4 Analyysin vaiheet

Luku neljä on käsitellyt lohkosalausalgoritmien asymptoottista analyysiä, jolle on muodostettu yksinkertainen analyysimalli. Taulukko 2 esittää yhteenvetoa esitetystä analyysimenetelmästä-

Nro.	Vaihe	Tavoite
1	Ongelmanmäärittely	Määritellään analyysin tavoite ja vaadittu tarkkuus
2	Osien erittely	Hajotetaan algoritmi osiin analyysiä varten
3	Osien analysointi	Analysoidaan algoritmin eri osat
4	Analyysien yhdistäminen	Yhdistetään erilliset analyysit yhteen
5	Ekstrapolointi	Jatketaan lohkon analyysi edustamaan koko algoritmin analyysiä
6	Ylärajan muodostaminen	Muodostetaan asymptoottinen yläraja

Taulukko 2 - Analyysin vaiheet

Kyseessä on kuusivaiheinen prosessi. Analyysimenetelmä on tarkoituksellisesti yksinkertainen, sillä tutkimuksen tavoite on tuottaa helposti lähestyttävä malli analyysin suorittamiseen. Tätä mallia hyödynnetään luvussa 5, jossa sen käyttöä sovelletaan kahteen erilaiseen lohkosalausalgoritmiin.

5 Esimerkkitapaukset AES-128 ja 3DES

Tässä luvussa tuotetaan asymptoottinen tehokkuusarvio AES-128 ja 3DES lohkosalausalgoritmeista, hyödyntäen tutkimuksen artefaktia, eli luvussa 4 määriteltyä toimintamallia. Luvun tavoitteena on tuottaa käyttökelpoinen analyysi metodilla, jonka hyödyntämiseen ei vaadita syvällistä perehtymistä algoritmeihin. AES-128 ja 3DES valittiin vertailun kohteiksi, koska ne edustavat kahta erilaista lohkosalausalgoritmin arkkitehtuuria, substituutio-permutaatio verkkoa ja Feistel-salausta (Aumasson, 2018, s.55–56), joten niiden analysointi voi vakuuttaa ehdotetun analyysimallin yleistettävyydestä eri lohkosalausalgoritmeihin.

Luku 5.1 esittelee AES-128 ja 3DES ohjelmakoodit, johon analyysit kohdistuvat. Luku 5.2 kertaa lyhyesti ehdotetun analyysimenetelmän sisällön. Alaluku 5.3 käsittää ehdotetun mallin mukaisesti molempien analyysien ongelmanmärittelyn. Ensimmäinen analyysi tehdään AES-128 algoritmille, joka aloitetaan luvussa 5.4. Luku 5.5 sisältää vastaavanlaisen analyysin kokonaisuudessaan 3DES algoritmille. Luku 5.6 sisältää lyhyen vertailun AES-128 ja 3DES algoritmien analyysien tuloksista.

5.1 Analysoitavat ohjelmakoodit

Eri lohkosalausalgoritmeille on olemassa useita eri implementaatioita. Eri tahot ovat luoneet implementaatioita eri ohjelmointikielillä ja erinäisiin tarkoituksiin. Eri implementaatiot saattavat myös tehdä eri päätöksiä siitä, mitä arvoja ohjelmakoodissa implementoidaan taulukkoina ja mitkä lasketaan ajon aikana. Tähän tutkimukseen AES-128-ohjelmakoodiksi valikoitui Texas Instrumentsin AES-128 ohjelmakoodi, joka on saatavilla heidän verkkosivuillaan (Texas Instruments, 2011). Koodia on muokattu poistamalla vahtiajastin (eng. watch-dog timer), sillä kyseinen ajastin ei kuulu itse AES-

128 algoritmiin, vaikka kyseinen funktio olisikin suotava lisäys käytännön implementaatiossa. Kuva 13 esittää analysoitua AES-128 koodia:

```
1   /* --COPYRIGHT--,BSD
2    * Copyright (c) 2011, Texas Instruments Incorporated
3    * All rights reserved.
4    *
5    * TI_aes_128_encr_only.c
6    *
7    *  Created on: Nov 3, 2011
8    *      Author: Eric Peeters
9    */
10
11
12  // forward sbox
13  const unsigned char sbox[256] =   {
14  //0     1    2      3     4    5    6     7      8    9     A      B    C     D     E     F
15  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0
16  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1
17  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2
18  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3
19  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4
20  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5
21  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6
22  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7
23  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8
24  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9
25  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A
26  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B
27  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C
28  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D
29  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E
30  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F
31
32  // multiply by 2 in the galois field
33  unsigned char galois_mul2(unsigned char value)
34  {
35    signed char temp;
36    // cast to signed value
37    temp = (signed char) value;
38    // if MSB is 1, then this will signed extend and fill the temp variable with 1's
39    temp = temp >> 7;
40    // AND with the reduction variable
41    temp = temp & 0x1b;
42    // finally shift and reduce the value
43    return ((value << 1)^temp);
44  }
45
46  // aes encryption function
47  // It manipulates the state and computes the key schedule on the fly
48  void aes_encrypt(unsigned char *state, unsigned char *key)
49  {
50    unsigned char buf1, buf2, buf3, buf4, round, i;
51    unsigned char rcon;
52
53    // Rcon initial value. All subsequent values are computed.
54    rcon = 0x01;
55
56    // main AES data loop
57    for (round = 0; round < 10; round++){
58      //add key + sbox
59      for (i = 0; i <16; i++){
60        state[i]=sbox[state[i] ^ key[i]];
61      }
62      //shift rows
63      buf1 = state[1];
64      state[1] = state[5];
65      state[5] = state[9];
66      state[9] = state[13];
67      state[13] = buf1;
68
69      buf1 = state[2];
70      buf2 = state[6];
71      state[2] = state[10];
72      state[6] = state[14];
73      state[10] = buf1;
74      state[14] = buf2;
75
76      buf1 = state[15];
77      state[15] = state[11];
78      state[11] = state[7];
79      state[7] = state[3];
80      state[3] = buf1;
```

```
82      //process mixcolumn for all rounds but the last one
83      if (round < 9) {
84        for (i=0; i <4; i++){
85          // compute the current index
86          buf4 = (i << 2);
87      buf1 = state[buf4] ^ state[buf4+1] ^ state[buf4+2] ^ state[buf4+3];
88      buf2 = state[buf4];
89      buf3 = state[buf4]^state[buf4+1]; buf3=galois_mul2(buf3); state[buf4] = state[buf4] ^ buf3 ^ buf1;
90      buf3 = state[buf4+1]^state[buf4+2]; buf3=galois_mul2(buf3); state[buf4+1] = state[buf4+1] ^ buf3 ^ buf1;
91      buf3 = state[buf4+2]^state[buf4+3]; buf3=galois_mul2(buf3); state[buf4+2] = state[buf4+2] ^ buf3 ^ buf1;
92      buf3 = state[buf4+3]^buf2;     buf3=galois_mul2(buf3); state[buf4+3] = state[buf4+3] ^ buf3 ^ buf1;
93      }
94      }
95
96      //key schedule
97      // compute the 16 next round key bytes
98      key[0] = sbox[key[13]]^key[0]^rcon;
99      key[1] = sbox[key[14]]^key[1];
100     key[2] = sbox[key[15]]^key[2];
101     key[3] = sbox[key[12]]^key[3];
102     for (i=4; i<16; i++) {
103     key[i] = key[i] ^ key[i-4];
104     }
105     // compute the next Rcon value
106     rcon = galois_mul2(rcon);
107   }
108
109   // process last AddRoundKey
110   for (i = 0; i <16; i++){
111     state[i]=state[i] ^ key[i];
112   }
113 }
114
```

Kuva 13 - AES-128. Texas Instruments, 2011.

3DES algoritmin ohjelmakoodiksi valittiin Sanat Bhandarkarin versio, jonka hän on julkaissut vuonna 2018. 3DES algoritmin toiminnasta on huomattavaa, että kyseinen ohjelma ajetaan kolme kertaa. Ensimmäisellä ja kolmannella ajokerralla salausmoodissa, toisella kerralla salauksen purkumoodissa. Salaus ja purkuoperaatiot eivät kumoa toisiaan, sillä 3DES käyttää joka ajokerralla eri salausavainta (Oppliger, 2005, s.254). 3DES algoritmin ohjelmakoodi esitetään kuvassa 14:

```
6   uint64_t des(uint64_t input, uint64_t key, char mode) {
7
8     int i, j;
9
10    /* 8 bits */
11    char row, column;
12
13    /* 28 bits */
14    uint32_t C                    = 0;
15    uint32_t D                    = 0;
```

```
16
17      /* 32 bits */
18      uint32_t L                  = 0;
19      uint32_t R                  = 0;
20      uint32_t s_output           = 0;
21      uint32_t f_function_res     = 0;
22      uint32_t temp               = 0;
23
24      /* 48 bits */
25      uint64_t sub_key[16]        = {0};
26      uint64_t s_input            = 0;
27
28      /* 56 bits */
29      uint64_t permuted_choice_1  = 0;
30      uint64_t permuted_choice_2  = 0;
31
32      /* 64 bits */
33      uint64_t init_perm_res      = 0;
34      uint64_t inv_init_perm_res  = 0;
35      uint64_t pre_output         = 0;
36
37      /* initial permutation */
38      for (i = 0; i < 64; i++) {
39
40          init_perm_res <<= 1;
41          init_perm_res |= (input >> (64-IP[i])) & LB64_MASK;
42
43      }
44
45      L = (uint32_t) (init_perm_res >> 32) & L64_MASK;
46      R = (uint32_t) init_perm_res & L64_MASK;
47
48      /* initial key schedule calculation */
49      for (i = 0; i < 56; i++) {
50
51          permuted_choice_1 <<= 1;
52          permuted_choice_1 |= (key >> (64-PC1[i])) & LB64_MASK;
53
54      }
55
56      C = (uint32_t) ((permuted_choice_1 >> 28) & 0x000000000fffffff);
57      D = (uint32_t) (permuted_choice_1 & 0x000000000fffffff);
58
59      /* Calculation of the 16 keys */
60      for (i = 0; i< 16; i++) {
61
62          /* key schedule */
63          // shifting Ci and Di
64          for (j = 0; j < iteration_shift[i]; j++) {
65
66              C = 0x0fffffff & (C << 1) | 0x00000001 & (C >> 27);
67              D = 0x0fffffff & (D << 1) | 0x00000001 & (D >> 27);
68
69          }
```

```
70
71          permuted_choice_2 = 0;
72          permuted_choice_2 = (((uint64_t) C) << 28) | (uint64_t) D ;
73
74          sub_key[i] = 0;
75
76          for (j = 0; j < 48; j++) {
77
78              sub_key[i] <<= 1;
79              sub_key[i] |= (permuted_choice_2 >> (56-PC2[j])) & LB64_MASK;
80
81          }
82
83      }
84
85      //kierros alkaa?
86      for (i = 0; i < 16; i++) {
87
88          /* f(R,k) function */
89          s_input = 0;
90
91          //expansion function
92          for (j = 0; j< 48; j++) {
93
94              s_input <<= 1;
95              s_input |= (uint64_t) ((R >> (32-E[j])) & LB32_MASK);
96
97          }
98
99          /*
100          * Encryption/Decryption
101          * XORing expanded Ri with Ki
102          */
103         if (mode == 'd') {
104             // decryption
105             s_input = s_input ^ sub_key[15-i];
106
107         } else {
108             // encryption
109             s_input = s_input ^ sub_key[i];
110
111         }
112
113         /* S-Box Tables */
114         for (j = 0; j < 8; j++) {
115             // 00 00 RCCC CR00 00 00 00 00 00 s_input
116             // 00 00 1000 0100 00 00 00 00 00 row mask
117             // 00 00 0111 1000 00 00 00 00 00 column mask
118
119             row = (char) ((s_input & (0x0000840000000000 >> 6*j)) >> 42-6*j);
120             row = (row >> 4) | row & 0x01;
121
122             column = (char) ((s_input & (0x0000780000000000 >> 6*j)) >> 43-6*j);
123
124             s_output <<= 4;
125             s_output |= (uint32_t) (S[j][16*row + column] & 0x0f);
126
127         }
```

```
128
129             f_function_res = 0;
130
131             for (j = 0; j < 32; j++) {
132
133                 f_function_res <<= 1;
134                 f_function_res |= (s_output >> (32 - P[j])) & LB32_MASK;
135
136             }
137
138             temp = R;
139             R = L ^ f_function_res;
140             L = temp;
141
142         }
143
144         pre_output = (((uint64_t) R) << 32) | (uint64_t) L;
145
146         /* inverse initial permutation */
147         for (i = 0; i < 64; i++) {
148
149             inv_init_perm_res <<= 1;
150             inv_init_perm_res |= (pre_output >> (64-PI[i])) & LB64_MASK;
151
152         }
153
154         return inv_init_perm_res;
155
156     }
157 }
158
```

Kuva 14 - 3DES (Bhandarkar, 2018)

Koodista on huomattavaa, että tämä 3DES toteutus hyödyntää hyvin monesti erinäisiä esilaskettuja taulukoita. Nämä taulukot ovat listattuna tutkimusraportin liitteisiin.

5.2 Ehdotettu malli

Luvussa neljä esiteltiin ehdotettu malli lohkosalausalgoritmien suoritustehoanalyysille, perustuen algoritmianalyysin teoriaan ja lohkosalausalgoritmien tyypilliseen rakenteeseen. Kyseinen malli on esitetty uudelleen taulukossa 3:

Nro.	Vaihe	Tavoite
1	Ongelmanmäärittely	Määritellään analyysin tavoite ja vaadittu tarkkuus
2	Osien erittely	Hajotetaan algoritmi osiin analyysiä varten
3	Osien analysointi	Analysoidaan algoritmin eri osat
4	Analyysien yhdistäminen	Yhdistetään erilliset analyysit yhteen
5	Ekstrapolointi	Jatketaan lohkon analyysi edustamaan koko algoritmin analyysiä
6	Ylärajan muodostaminen	Muodostetaan asymptoottinen yläraja

Taulukko 3 - Ehdotettu analyysimalli

Kyseessä on siis kuusivaiheinen prosessi, joka alkaa ongelmanmäärittelyllä. Ongelmanmäärittelyssä esimerkiksi pohditaan, kuinka tarkka analyysi tarvitaan. Seuraavaksi erotellaan algoritmin eri osat, eri hajotetaan se analyysiä varten. Osittelun jälkeen erotellut osat ja erilliset funktiot analysoidaan erikseen. Tämän jälkeen nämä osittaiset analyysit yhdistetään yhtä lohkoa edustavaksi analyysiksi. Sitten, tästä lohkon suoritustehokkuudesta ekstrapoloidaan arvio koko algoritmin suoritustehokkuudelle. Lopuksi tämä tulos sievennetään, ja muodostetaan arvio suoritustehon funktion ylärajaksi. Tätä toimintaketjua kuvataan AES-128 algoritmille luvusta 5.3 eteenpäin.

5.3 Ongelmanmäärittely

Ensimmäinen askel analyysissä on ongelmanmäärittely. Tähän liittyi kaksi pääkohtaa, analyysin tavoite ja vaadittu tarkkuus. Tavoite lienee selvä; tuottaa analyysit AES-128 ja 3DES algoritmien suoritustehokkuudesta. Tavallisesti analyysin tarkkuudeksi riittää

asymptoottisen ylärajan määritelmä. Toisin sanoen, analyysin tulos ilmaistaan muodossa $T(n) = O(n^x)$. Tässä tapauksessa kuitenkin vertaillaan kahta samantyyppistä algoritmia, joten niiden asymptoottiset ylärajat eivät välttämättä ilmaise niiden tehokkuuseroa tarpeeksi tarkasti. Tästä syystä niiden vertailussa hyödynnetään myös niiden tarkkoja suoritustehokkuuden arvioita.

5.4 AES-128 analyysi

Analyysiesimerkit aloitetaan AES-128 algoritmista. Ehdotetun analyysimallin mukaisesti ongelmanmäärittely on suoritettu, joten itse analyysiprosessi voi alkaa osien erittelyllä.

5.4.1 AES-128 osien erittely

Ensimmäinen vaihe analyysissä on algoritmin osien erittely. Ensimmäinen ohjelmakoodissa määritelty asia on rivillä 13 määritelty AES-algoritmin S-box. Kyseessä on esilaskettu vastaustaulukko, jota hyödynnetään eri osissa algoritmia. Tämä nopeuttaa bittien substituutiota, koska näitä substituutioarvoja ei tarvitse laskea ajon aikana, vaan oikeat arvot vain katsotaan valmiista taulukosta.

Seuraava ohjelmakoodin erillinen osa on rivillä 33 määritelty galois_mul2 -funktio. Kyseinen funktio liittyy bittiaritmetiikkaan, sillä siinä kerrotaan annettu arvo kahdella. Näin triviaalille toimenpiteelle on luotu oma funktio siksi, että toiminto tapahtuu niin kutsutussa Galoisin kunnassa $GF(p^2)$, joka on tietynlainen matemaattinen äärellinen kunta (St. Denis & Johnson, 2007, s.144). Sen toimintaa ei tarvitse ymmärtää suoritustehon analysoinnin kannalta tätä pidemmälle, sen tehtävä on vain suorittaa määrätty laskutoimitus tiettyjen matemaattisten sääntöjen sisällä. Tätä funktiota hyödynnetään muualla algoritmissa.

Tässä AES-128 toteutuksessa loput toiminnot tapahtuvat riviltä 48 alkavassa koodisegmentissä. Tästä segmentissä on havaittavissa, että tässä AES-128-toteutuksessa on kolme osaa: Aloitus, kierrosrakenne ja lopetus. Aloituksessa initialisoidaan muuttujia

ja lopetuksessa, rivillä 110, suoritetaan viimeisen kierrosavaimen lisäys salattavaan lohkoon. Itse salauskierroksessa on muutama huomattava asia. Rcon-taulukkoa, jota hyödynnetään avainalgoritmissa, ei ole määritelty ennakkoon. Sen ensimmäinen arvo on annettu valmiiksi ja loput lasketaan koodin suorituksen aikana. Lisäksi rivillä 60 suoritetaan tavujen substituutio ja kierrosavaimen lisääminen yhdessä funktiossa. Rivillä 63 suoritetaan rivien siirto ja rivillä 83 suoritetaan kolumnien sekoitus. Riviltä 98 eteenpäin suoritetaan avainalgoritmia.

Rakenteesta on siis selkeästi havaittavissa kaksi funktiota, joita hyödynnetään toistuvasti kierrosrakenteen sisällä: sbox ja galois_mul2. Näiden lisäksi algoritmi koostuu aloituksesta, kierrosrakenteesta ja lopetuksesta.

5.4.2 AES-128 osien analysointi

Algoritmin eri osat on siis eristetty. Seuraava vaihe analyysiprosessissa on ottaa nämä eri osat ja analysoida niiden sisältö. Sbox-taulukko ja galois_mul2 ovat hyviä lähtökohtia. Sbox on vain yksinkertainen taulukko. Sen koko ei ikinä muutu, eikä se riipu syötteen n koosta. Täten, sen initiaatio on vakioaikainen toiminto. Sitä myös hyödynnetään yksinkertaisella tavalla: jollekin arvolle x valitaan vastakappale y taulukosta valitsemalla y = sbox[x], eli alkuperäistä arvoa käytetään indeksiarvona. Tähän toimintoon kuluva aika on tietysti vakio, sillä kyseessä on käytännössä vain yksinkertainen datan lukutoiminto. S-boxin käyttöön kuluu siis vakiomittainen aika O(1).

Funktio galois_mul2 on myös vakioaikainen funktio. Funktio ottaa syötteenä arvon, jonka se lukee tilapäismuuttujaan temp. Tämän muuttujan bittejä se siirtää oikealle yhteensä seitsemän kertaa ja kutistaa vakioarvolla. Funktio palauttaa syötearvon, jonka bitit on siirretty kerran vasemmalle ja jota on kutistettu loogisella AND-operaatiolla tilapäismuuttujan temp avulla. Kuva 15 on ote funktiosta, jotta selitystä on helpompi seurata.

```
32  // multiply by 2 in the galois field
33  unsigned char galois_mul2(unsigned char value)
34  {
35      signed char temp;
36      // cast to signed value
37      temp = (signed char) value;
38      // if MSB is 1, then this will signed extend and fill the temp variable with 1's
39      temp = temp >> 7;
40      // AND with the reduction variable
41      temp = temp & 0x1b;
42      // finally shift and reduce the value
43      return ((value << 1)^temp);
44  }
45
```

Kuva 15 - galois_mul2. Texas Insrument, 2011.

Funktio luo muuttujan temp ja lukee syötteen temp-muuttujaan. Tämä kuvaa kahta vakioaikaista toimintoa O(1). Funktiossa suoritetaan myös vakioaikainen bittien siirto O(1) yhteensä kahdeksan kertaa. Myös looginen operaatio AND, myös vakioaikainen O(1), suoritetaan kahdesti. Lopputulos myös kirjoitetaan return-toiminnolla, joka sekin on vakioaikainen O(1) toiminto. Yhteensä funktio siis tekee 13 O(1) toimintoa. Tätä arvoa käytetään analyysissä myöhemmin.

AES-128:n toiminta alkaa riviltä 48. Ensimmäisenä on aloitusvaihe, jossa initialisoidaan seitsemän muuttujaa ja kirjoitetaan yhdelle muuttujalle arvo. Yhteensä tämä vastaa siis kahdeksaa O(1) toimintoa, eli:

$$8 \cdot O(1)$$

Aloituksen jälkeen seuraava vaihe on salauskierros, joita AES-128:n tapauksessa on kymmenen (St. Denis & Johnson, 2007, s.159). Tämä on nähtävissä koodissa riviltä 57 alkavasta silmukasta, jota toistetaan kymmenen kertaa. Täten tästä kappaleesta eteenpäin kaikki osa-analyysit tapahtuvat tämän silmukan sisällä, kunnes toisin mainitaan. Tämä kymmenkerroin jätetään erikseen mainitsematta selkeyden vuoksi ja se huomioidaan myöhemmin.

Ensimmäisenä silmukan sisällä aloitetaan toinen silmukka, joka tehtävänä on hoitaa SubBytes ja AddRoundKey -funktiot, eli tavujen korvaamis- ja kierrosavaimen lisäysfunktiot. Sisempää silmukkaa toistetaan 16 kertaa ja sen sisällä on yksi O(1) toiminto. On huomattavaa, että todellisuudessa kyseisessä toiminnossa kirjoitetaan arvo muuttujaan, käytetään eksklusiivista tai (XOR) -toimintoa ja haetaan korvaava arvo s-boxista, eli silmukassa on yhteensä kolme toimintoa yhdessä koodirivissä. Korvaavan arvon haku on aiemmin määritelty sbox-funktio, joka toimi ajassa O(1). Kyseessä on siis kolme O(1) toimintoa, joten ne sulautetaan tässä yhteydessä yhteen, sillä vakioaikaisten toimintojen vaikutus algoritmin suoritustehoon on käytännössä merkityksetön (Cormet ja muut, 2009, s.28). Näin ollen, analyysin helpottamiseksi samalla koodirivillä tapahtuvat vakioaikaiset toiminnot O(1) luetaan tässä analyysissä yhdeksi O(1) toiminnoksi. Täten, kuvassa 16 esitetyn sisemmän silmukan suoritustehokkuus on:

$$16 \cdot O(1)$$

```
for (i = 0; i <16; i++){
  state[i]=sbox[state[i] ^ key[i]];
}
```

Kuva 16 - SubBytes ja AddRoundKey

Tämän jälkeen on vuorossa kuusitoista vakioaikaista O(1) riviä, liittyen rivien siirtämiseen, eli ShiftRows -toimintoon. Kyseessä on koodin rivit 62–80. Koska koodirivit ovat käytännössä vain datan siirtelyä edestakaisin, sen tehokkuus on:

$$16 \cdot O(1)$$

Tämän jälkeen riviltä 111 alkaen koodi toteuttaa kolumnien sekoituksen, eli MixColumns-toiminnon. Toiminto koostuu uudesta silmukasta, jota toistetaan neljä kertaa. Silmukan sisällä on seitsemän riviä, joista kolme on tavallisia O(1)-toimintoja ja neljä puolestaan sisältää aiemmin määriteltyjä galois_mul2 -toimintoja. Kuten aiemmin määriteltiin, galois_mul2-funktio on suoritustehokkuudeltaan $13 \cdot O(1)$. Analyysin

yksinkertaistamiseksi valitaan riveille, joilla tämä funktio esiintyy, suoritustehokkuus $13 \cdot O(1)$, sillä se on rivin suoritustehollisesti raskain komponentti. Näin ollen, kyseessä on:

$$4(4 \cdot [13 \cdot O(1)] + 3 \cdot O(1) = 220 \cdot O(1)$$

Tässä koodissa on kuitenkin huomattavissa yksi tärkeä huomautus: MixColumn-toimintoa ei suoriteta viimeisellä salauskierroksella. Tämä on havaittavissa if-ehtolausekkeesta kuvassa 17.

```
//process mixcolumn for all rounds but the last one
if (round < 9) {
  for (i=0; i <4; i++){
    // compute the current index
    buf4 = (i << 2);
buf1 = state[buf4] ^ state[buf4+1] ^ state[buf4+2] ^ state[buf4+3];
buf2 = state[buf4];
buf3 = state[buf4]^state[buf4+1]; buf3=galois_mul2(buf3); state[buf4] = state[buf4] ^ buf3 ^ buf1;
buf3 = state[buf4+1]^state[buf4+2]; buf3=galois_mul2(buf3); state[buf4+1] = state[buf4+1] ^ buf3 ^ buf1;
buf3 = state[buf4+2]^state[buf4+3]; buf3=galois_mul2(buf3); state[buf4+2] = state[buf4+2] ^ buf3 ^ buf1;
buf3 = state[buf4+3]^buf2;     buf3=galois_mul2(buf3); state[buf4+3] = state[buf4+3] ^ buf3 ^ buf1;
}
}
```

Kuva 17 - MixColumn

Tämä on otettava huomioon analyysissä, sillä tämä koodi sijaitsee silmukan sisällä, jota toistetaan kymmenen kertaa. Kun salauskierrosten määrä on kymmenen, on MixColumns-toimintojen määrä yhdeksän. Tämä voidaan ottaa huomioon lisäämällä lopuksi $-220 \cdot O(1)$ salauskierrosten kokonaissuoritustehokkuuteen.

Seuraava vaihe salauskierroksessa on avainalgoritmi, eli KeySchedule, alkaen riviltä 98. Avainalgoritmi alkaa neljällä O(1) -toiminnolla, jotka hyödyntävät vakioaikaista sbox-funktiota. Tämän jälkeen alkaa uusi silmukka, jota toistetaan 12 kertaa. Silmukka toistaa yhtä O(1)-toimintoa, joten avainalgoritmin kokonaissuoritustehokkuus on $4 \cdot O(1) + 12 \cdot O(1) = 16 \cdot O(1)$. Tämä ei kuitenkaan riitä, sillä avainalgoritmin yhteydessä lasketaan vielä galois_mul2 -funktiolla seuraava rcon-arvo. Tälle funktiolle määriteltiin aiemmin suoritustehokkuudeksi $13 \cdot O(1)$, joten avainalgoritmin kokonaissuoritusteho on:

$$16 \cdot O(1) + 13 \cdot O(1) = 29 \cdot O(1)$$

Salauskierroksen suoritustehoanalyysi on siis:

$$tavujen\ korvaus\ ja\ avaimen\ lisäys + rivien\ siirto + kolumnien\ sekoitus + avainalgoritmi$$

Eli:

$$16 \cdot O(1) + 16 \cdot O(1) + 220 \cdot O(1) + 29 \cdot O(1) = 281 \cdot O(1)$$

On kuitenkin muistettava, että tämä salauskierros toistuu kymmenen kertaa, paitsi kolumnien sekoitus, joka toistuu vain yhdeksän kertaa. Salauskierrosten suoritustehokkuus on siis:

$$(10 \cdot 281 - 220) \cdot O(1) = 2590$$

AES-128:n viimeinen osa on lopetus. Kyseisessä osiossa lisätään viimeinen kierrosavain salattavaan lohkoon. Tämä tehdään silmukalla, joka toistuu kuusitoista kertaa. Itse lisäys on yhden O(1) toiminnon arvoinen, joten lopetus on:

$$16 \cdot O(1)$$

5.4.3 AES-128 analyysien yhdistäminen

AES-algoritmin eri komponenttien suoritustehokkuus on täten määritelty. Yhdistämällä nämä komponentit saadaan suoritustehokkuudeksi:

$$T(n) = aloitus + salauskierros + lopetus$$

Eli:

$$T(n) = 8 \cdot \mathrm{O}(1) + 2590 \cdot \mathrm{O}(1) + 16 \cdot \mathrm{O}(1) = 2614$$

O(1) poistettiin yhtälöstä selkeyden vuoksi, mutta kyseessä on yhä vakioaikainen kokonaisuus.

5.4.4 AES-128 ekstrapolointi kokonaissyötteen suhteen

AES-128-algoritmin suoritustehokkuus on siis täten määritelty, mutta tämä analyysi kuvaa vain yhtä lohkoa koko lohkosalausalgoritmissa. Nämä lohkot salaavat vain 128-bittiä dataa (Aumasson, s.59), joten n-bittisen syötteen salaamiseksi näitä lohkoja tarvitaan $b = \frac{\mathrm{n}}{128}$ kappaletta. Täten, jos lohkon suoritustehokkuus on $T(n) = 2614$ja näitä lohkoja tarvitaan b kappaletta, koko algoritmin suoritustehokkuudeksi voidaan ekstrapoloida:

$$T(n) = b \cdot 2614 = \frac{\mathrm{n}}{128} \cdot 2614 = \frac{2614}{128} n \approx 21n$$

Kyseessä on siis yksinkertainen arvio AES-128-algoritmin kokonaissuoritustehosta, joka on johdettu yhdestä algoritmin salauslohkosta.

5.4.5 AES-128 ylärajan muodostaminen

Viimeinen vaihe analyysissä on muodostaa suoritustehokkuudelle selkeä yläraja, hyödyntäen asymptoottista notaatiota ja ylärajan määritelmää. Asymptoottisen ylärajan määritelmä on (Cormen ja muut, 2009, s.47):

$$0 \leq f(n) \leq c * g(n), n0 \leq n$$

Eli funktiota $f(n)$ voidaan rajoittaa ylhäältäpäin jollain funktiolla $g(n)$, jos rajaava funktio on vakiolla c kerrottuna aina suurempi tai yhtä suuri kuin $f(n)$, alkaen jostain pisteestä $n0$.

Kun funktiomme $f(n) = T(n) = \frac{2614}{128}n$, niin tämän ylärajan muodostaminen on suhteellisen yksinkertaista. Voimme valita funktion $g(n) = n$ ja arvot $c = 2614, n0 = 1$. On ilmeistä, että tällöin asymptoottisen ylärajan ehto täyttyy, eli:

$$0 \leq \frac{2614}{128}n \leq 2614 \cdot n, kun\ n0 = 1$$

On myös ilmeistä, että kyseessä on tiukka yläraja, sillä Cormenin ja muiden (2009, s.47) mukaisesti suoritustehokkuutta rajaava funktio $g(n)$ on aina suurempi vakiokertoimen verran, jolloin sen suurin termi n on samaa kokoluokkaa funktion $f(n)$ suurin termi. Koska asymptoottinen yläraja suoritusteholle löytyy funktiosta $g(n) = 2614n$, on AES-128-algoritmin suoritusteho lineaarinen, eli:

$$T(n) = O(n).$$

Kyseinen yläraja on tiukka, sillä se täyttää Cormenin ja muiden (2009, s.47) määritelmän tiukalle ylärajalle, eli g(n) rajoittaa funktiota vakioarvolla.

5.5 Analyysi 3DES algoritmille

Analyysin ongelmanmäärittely toteutettiin yhtäaikaisesti AES-128 algoritmin kanssa, joten 3DES analyysi voidaan aloittaa heti osien erittelyllä. Tässä 3DES toteutuksessa ei ole juurikaan galois_mul_2-funktion tapaisia toistuvia funktioita. Se tosin hyödyntää esilaskettuja taulukkoja huomattavasti AES-128 toteutusta enemmän. Tästä esimerkkinä on sen kahdeksan uniikkia S-boxia, verrattuna AES-128:n yhteen S-boxiin. Tämä on hyvä tiedostaa, mutta se ei aiheuta erittelyvaiheessa erityisiä toimenpiteitä. Muita osia 3DES funktiossa on aloituspermutaatio (eng. initial permutation), avainalgoritmi,

salauskierros ja lopetuspermutaatio (eng. inverse initial permutation). Osien erittelyn jälkeen on aika aloittaa niiden analysointi.

5.5.1 3DES osien analysointi

3DES algoritmin ensimmäinen vaihe on aloituspermutaatio. Ennen aloituspermutaatiota algoritmi kuitenkin luo 18 eri muuttujaa. Muuttujan luominen on vakioaikainen toiminto O(1), joten kokonaisuudessaan aloitus on $18 \cdot O(1)$ arvoinen. Itse aloituspermutaatio, jonka tehtävä on jakaa salattava 64-bittinen lohko kahteen 32-bittiseen lohkoon ja siirrellä bittien järjestystä (Oppliger, 2005, 241–242), on 64 kertaa toistuva silmukka. Silmukassa init_perm_res-muuttujan sisältöä siirretään yksi bitti oikealle, ja salattavan lohkon bitti esilasketun IP-taulukon osoittamasta paikasta siirretään init_perm_res-muuttujan oikeimmanpuoliseksi bitiksi. Silmukan bittisiirrot huolehtivat, että bitit kulkeutuvat init_perm_res-muuttujassa kohti vasenta laitaa, tehden tilaa oikealla seuraavalle bitille. Bittisiirto rivillä 40 on yksi vakioaikainen toiminto $O(1)$. Bitin asetus init_perm_res-muuttujaan on puolestaan toteutettu OR-bittitoiminnolla, bittisiirrolla, AND-bittitoiminnolla, yhdellä miinuslaskulla sekä yhdellä datan lukutoiminnolla esilasketusta IP-taulukosta. Nämä toiminnot voidaan typistää yhdeksi $O(1)$ toiminnoksi, sillä kyse on yhdellä koodirivillä tapahtuvista vakiotoiminnoista. Silmukassa tehdään siis kahdella rivillä yhteensä $1 \cdot O(1) + 1 \cdot O(1)$ toimintoa. Kertoen sen silmukan koolla saadaan tulokseksi $64 \cdot 2 \cdot O(1) = 128 \cdot O(1)$.

Lopuksi 64-bittinen init_perm_res jaetaan kahteen eri 32-bittiseen muuttujaan. Tämä jako toteutetaan riveillä 45 ja 46. Riveillä kirjoitetaan dataa muuttujiin kahdesti, siirretään bittejä kerran ja käytetään AND-bittitoimintoa kahdesti ja muutetaan muuttujan tyyppiä (eng. type cast/type conversion) kahdesti, johtaen tulokseen $2 \cdot O(1)$. Aloituksen kokonaisarvio on siis:

$$18 \cdot O(1) + 128 \cdot O(1) + 2 \cdot O(1) = 148 \cdot O(1)$$

Seuraavana analysoidaan riviltä 49 alkava avainalgoritmi. Avainalgoritmi alkaa typistämällä 64-bittinen salausavain 56-bittiseksi. Typistys tehdään lähes identtisesti aloituspermutaation nähden. 56-kierroksinen silmukka siirtää salausavaimesta halutun bitin avaimen oikeaan laitaan ja siirtää sen permuted_choice_1-muuttujan oikeanpuolimmaiseksi bitiksi, joka puolestaan bittisiirroilla siirtyy vasemmalle. Salausavaimesta haluttu bitti määräytyy esilasketun PC1-taulukon mukaan. Myös tämän toiminnon tulos, typistetty salausavain, jaetaan kahteen osaan. Kokonaisuudessaan salausavaimen typistämiseen kuluu vakioaikaisiatoimintoja:

$$56 \cdot 2 \cdot O(1) + 2 \cdot O(1) = 114 \cdot O(1).$$

Avainalgoritmi kuitenkin jatkuu vielä, sillä typistämisen jälkeen algoritmi muodostaa 16 kappaletta kierrosavaimia. Riviltä 60 alkaa silmukka, jota toistetaan 16 kertaa. Tämä pääsilmukka sisältää ensin alemman tason silmukan, jota toistetaan yksi tai kaksi kertaa per pääsilmukan kierros. Tässä alasilmukassa aiemmin typistetyn salausavaimen kahden puoliskon bittejä siirretään vasemmalle samalla huolehtien, että vasen bitti siirtyy oikeimpaan positioon. Tämä alasilmukka toistuu 28 kertaa, joka määräytyy esilasketun iteration_shift-taulukosta. Alasilmukka sisältää kaksi $O(1)$ toimintoa, joten sen tulos on:

$$28 \cdot 2 \cdot O(1) = 56 \cdot O(1).$$

Alasilmukan jälkeen ohjelmakoodissa esiintyy kolme erillistä koodiriviä, alkaen riviltä 71. Niissä alasilmukassa käsitellyt avaimenpuolikkaat asetetaan yhteen muuttujaan, sekä lopulliset kierrosavaimet sisältävän taulukon käsiteltävä indeksi asetetaan nollaan. Näissä riveissä esiintyy kirjoittamistoimintoja, bittioperaatioita sekä tyyppimuunnoksia. Niiden toimintojen kokonaissumma on kolme vakioaikaista toimintoa, eli:

$$3 \cdot O(1)$$

Avainalgoritmin viimeinen vaihe on itse kierrosavaimen luonti, rivillä 76. Tämä tehdään hyödyntämällä jälleen alasilmukkaa, tällä kertaa sitä iteroidaan 48 kertaa. Tämän alasilmukan sisältö on hyvin samanlainen, kuin aiemmissa silmukoissa; siinä aiemman vaiheen permuted_choice_2 muuttujan määrätty bitti siirretään oikeaan laitaan, kirjoitetaan sub_key-taulukkoon ja siirretään taulukon indeksin bittejä vasemmalle. Permuted_choice_2-muuttujan määrätty bitti valikoituu esilasketusta PC2-taulukosta. Alasilmukan sisältö $2 \cdot O(1)$, joten alasilmukka on kokonaisuudessaan:

$$48 \cdot 2 \cdot O(1) = 96 \cdot O(1)$$

On muistettava, että avainalgoritmi toistetaan 16 kertaa, jotta se tuottaa tarvittavat 16 kierrosavainta. Näin ollen, alkuperäisen salausavaimen typistystä lukuun ottamatta, muut askeleet täytyy kertoa luvulla 16, eli pääsilmukan kierrosmäärällä. Koko avainalgoritmin suoritustehokkuus on siis salausavaimen typistys summattuna loppuun avainalgoritmiin kertaa kuusitoista:

$$114 \cdot O(1) + 16 \cdot (56 + 3 + 96) \cdot O(1) = 2594 \cdot O(1)$$

Seuraavaksi analyysi kohdistuu itse 3DES algoritmin salauskierrokseen. 3DES algoritmin salauskierrosta toistetaan 16 kertaa, joka ilmenee koodirivillä 86. Kyseessä on kierroksen pääsilmukka. Ensimmäinen rivi pääsilmukan sisällä on yksittäinen $O(1)$ toiminto, jossa asetetaan muuttuja nollaksi. Tämän jälkeen alkaa ensimmäinen alasilmukka, jota toistetaan 48 kertaa. Kun aloituspermutaatiossa salattava 64-bittinen lohko jaettiin kahteen osaan, salauskierroksen alasilmukka ottaa toisen näistä lohkoista ja laajentaa sen 32 bitistä 48 bittiin, hyödyntäen esilaskettua E-taulukkoa. Tämä tunnetaan laajennusfunktiona (eng. expansion function). Itse laajennus tapahtuu tutulla tavalla, siirretään 32-bittisen muuttujan tietty bitti ensin oikealle, kirjoitetaan se erilliseen muuttujaan ja siirretään uuden muuttujan bittejä yksi vasemmalle. Alasilmukassa esiintyy bittisiirtoja, kirjoittamista, bittioperaatioita, taulukon arvon lukua ja tyyppimuunnoksia yhteensä kahdella rivillä:

$$48 \cdot 2 \cdot O(1) = 96 \cdot O(1)$$

Seuraava toiminto salauskierroksen sisällä, rivillä 103, on moodin valinta; halutaanko suorittaa salaus vai salauksen purkutoiminto. Kun 3DES ajetaan, ohjelmakoodi, jota analysoidaan, käynnistetään kolme kertaa, kahdesti salausmoodissa, kerran purkumoodissa. Tässä analyysissä oletetaan kuitenkin selkeyden vuoksi, että kyseessä on aina salausmoodi. Tämä ero ei aiheuta merkittävää muutosta algoritmin analyysin tulokselle, sillä molemmissa tapauksissa kyseessä on vakioaikainen toiminto. Ehtolauseen ehdon tarkistus on yksi vakioaikainen toiminto, jonka lisäksi moodin valinnassa kirjoitetaan muuttujaan, tehdään bittioperaatio ja luetaan arvo taulukosta, jotka myös ovat vakioaikaisia toimintoja. Kyseessä on siis:

$$2 \cdot O(1)$$

Moodin valinnan jälkeen rivillä 114 algoritmi hyödyntää S-boxeja. Kun AES-128 algoritmilla oli yksi S-box, 3DES algoritmilla niitä on kahdeksan (Oppliger, 2005, s.246). Ne tekevät bittien substituution, kuten AES-128:n tapauksessa, mutta tässä ne myös supistavat kuuden bitin pätkän neljään bittiin. Näin ollen, aiemmin laajennusfunktiolla 32 bitistä 48 bittiin laajennettu lohkon puolikas supistuu takaisin 32-bittiseksi. Lohkon puolikas täten käydään läpi pienissä kuuden bitin joukoissa, joissa ne kutistuvat neljään bittiin ja niiden bittien sisältö muuttuu S-boxin mukaan.

3DES S-boksit substituoivat bittejä siis kuuden joukoissa. Tällaisen kuuden bitin joukosta laitimmaiset bitit ilmaisevat S-boksin rivinumeron 0-3 ja neljä keskimmäistä kolumninumeron. Nämä kaksi koordinaattia siis selvitetään ensin. Rivinumeron selvittämisessä käytetään bittioperaatioita, bittisiirtoja, aritmeettisia operaatioita sekä tyyppimuunnoksia. Näiden summa on:

$$2 \cdot O(1)$$

ja kolumninumeron löytämiseen käytettiin vakiotoimintoja:

$$1 \cdot O(1)$$

Kun rivi- ja kolumninumerot ovat selvillä, toteutetaan bittien substituutio. Käytännössä substituoitavasta bittijoukosta korvataan kuusi bittiä rivi- ja kolumninumeron perusteella löytyvästä S-boxin indeksin arvosta. Ohjelmakoodissa tämä tapahtuu riveillä 124 ja 125. Vakioaikaisia toimintoja tähän kuluu kaksi kappaletta:

$$2 \cdot O(1)$$

Bittien substituutio kaikkinensa toistetaan kahdeksan kertaa, joka tarkoittaa kahdeksaa rivinumeroa, kolumninumeroa sekä itse substituutiotoimintoa. Yhteensä tämä tarkoittaa suoritustehon kannalta:

$$8(2 + 1 + 2) \cdot O(1) = 40 \cdot O(1)$$

Substituution jälkeen, alkaen riviltä 129, toteutetaan permutaatiovaihe. Tämä vaihe vastaa läheisesti aloituspermutaatiota, mutta se hyödyntää erillistä esilaskettua taulukkoa P, eikä se jaa syötettyä lohkoa enää kahteen osaan. Kyseessä on 32 kertaa toistuva silmukka ja yksi silmukan ulkopuolinen vakioaikainen toiminto. Silmukan sisältö on kaksi koodiriviä, joiden sisältö on vakio $O(1)$. Niiden summa on:

$$(32 \cdot 2 + 1) \cdot O(1) = 65 \cdot O(1)$$

Salauskierroksen viimeinen toiminto on aloituspermutaatiossa luotujen kahden lohkonpuolikkaan paikkojen vaihto. Koska Feistel-salauksessa salauskierroksen toimenpiteet koskevat pääosin vain toista näistä kahdesta lohkosta, vaihdetaan niiden paikkaa joka salauskierroksen päätteeksi, jotta molemmat puoliskot päätyvät salauksen piiriin. Tämä riviltä 138 alkava toimenpide on arvoltaan:

$$3 \cdot O(1)$$

Salauskierroksen sisältö on analysoitu, mutta on muistettava, että kierroksia on algoritmissa yhteensä 16 kappaletta. Näin ollen, salauskierroksen kokonaisanalyysi on:

$$16 \cdot (laajennusfunktio\ +\ moodivalinta\ + bittisubtituutio + permutaatio + lohkojen\ puoltenvaihto)$$

Eli:

$$16(96 + 4 + 40 + 65 + 3) \cdot O(1) = 3296 \cdot O(1)$$

Salauskierroksen jälkeen 3DES algoritmissa on jäljellä vain aloituspermutaation käänteinen toimenpide, lopetuspermutaatio. Alkaen riviltä 140, tässä toiminnossa hyödynnetään esilaskettua PI-taulukkoa. Lopetuspermutaatio koostuu 64-kierroksisesta silmukasta sekä $1 \cdot O(1)$ kokoisesta rivistä, jossa salauskierroksesta saapuvat salatut lohkonpuolikkaat asetetaan yhteen muuttujaan. Itse kierros muistuttaa toiminnaltaan aloituspermutaatiota, pois lukien syötteen kahtiajaon. Se sisältää $2 \cdot O(1)$ toimintoa. Suoritusteholtaan se on yhteensä:

$$(1 + 64 \cdot 2) \cdot O(1) = 129 \cdot O(1)$$

5.5.2 3DES analyysien yhdistäminen

Koska 3DES algoritmin kaikki osat on nyt analysoitu, tutkimuksen ehdottaman analyysimallin seuraava vaihe on näiden osa-analyysien yhdistäminen. Tämä yhdistelmä on muotoa:

$$T(n) = Aloituspermutaatio + avainalgoritmi + salauskierros + lopetuspermutaatio$$

Eli:

$$T(n) = (148 + 2594 + 3296 + 129) \cdot O(1) = 6167$$

5.5.3 3DES ekstrapolointi kokonaissyötteen suhteen

Seuraava analyysiprosessin vaihe on ekstrapolointi lohkon suoritustehosta koko algoritmin suoritustehoksi. Tämän tekemiseksi täytyy määritellä AES-128 algoritmin tavoin vaadittu lohkojen lukumäärä b. Lohkokoko 3DES algoritmilla on 64 bittiä, joten n-kokoisen syötteen salaamiseen tarvitaan $b = \frac{n}{64}$ lohkoa (Oppliger, 2005, s.241). Lisäksi 3DES algoritmi perustuu siihen, että DES algoritmi suoritetaan kolmesti samalle salattavalle tekstille. (Oppliger, 2005, s.254). Näin ollen, lohkoja tarvitaankin $b = \frac{3n}{64}$ kappaletta. Koko algoritmin suoritustehokkuus on siis lohkojen lukumäärä kerrottuna lohkon suoritustehokkuudella, eli:

$$T(n) = \frac{3n}{64} \cdot 6167 = \frac{18501}{64} n \approx 290n$$

5.5.4 3DES ylärajan muodostaminen

Viimeinen vaihe ehdotetussa analyysimallissa on asymptoottisen ylärajan muodostaminen. Kertaukseksi, asymptoottisen ylärajan määritelmä oli Cormenin ja muiden (2009, s.47) mukaan:

$$0 \leq f(n) \leq c \cdot g(n), n0 \leq n$$

Funktio $f(n) = T(n) = \frac{18501}{64} n$, joka on siis 3DES algoritmin tarkka suoritustehokkuuden analyysi. Kuten AES-128 tapauksessa, on hyvin helppoa määrittää tiukka yläraja tälle funktiolle. Jos valitaan $g(n) = n$ ja $c = 18501$, on selvää, että

$$0 \leq \frac{18501}{64} n \leq 18501 \cdot n, kun\ n0 = 1$$

Joten on selvää, että 3DES algoritmin asymptoottinen yläraja on $\mathrm{T(n) = O(n)}$. Kyseessä on myös tiukka raja, sillä g(n) rajoittaa suoritustehokkuutta vakioarvon avulla (Cormen & muut, 2009, s.47).

5.6 AES-128 ja 3DES suoritustehokkuuden vertailu

AES-128 ja 3DES algoritmien suoritustehokkuuksista on tuotettu arviot, joten niiden suoritustehokkuuksia on mahdollista vertailla keskenään. Koska molempien algoritmien suoritustehokkuuden yläraja on muotoa $\mathrm{T(n) = O(n)}$, on syytä tarkastella niiden tehokkuutta hieman syvällisemmin. Tässä voidaan käyttää hyväksi niiden tarkkoja suoritustehokkuusarvioita, eli:

AES-128: $T(n) = \frac{2614}{128} n \approx 21n$

3DES: $T(n) = \frac{18501}{64} n \approx 290n$

On välittömästi nähtävissä, että siinä missä AES-128 suorittaa noin 21 vakioaikaista toimintoa per yksi syötteen bitti, 3DES tekee niitä noin 290. Ero on suuri, joka tukee aiemman tutkimuksen löydöksiä. Esimerkiksi Panda (2016, s.282) ja Iftikhari ja muut (2021) päätyvät perinteisellä aikamittauksella samankaltaiseen tulokseen. Tällä perusteella aiempi tutkimus tukee ehdotetun analyysimenetelmän lopputuloksia.

6 Diskussio tuloksista

Tutkimuksen tavoite oli tuottaa yksinkertainen menetelmä, jolla aiheeseen pinnallisesti perehtynyt henkilö voisi arvioida lohkosalausalgoritmien suoritustehokkuutta. Kyseinen menetelmä on suunnittelutieteellisen tutkimuksen tulos, eli tutkimuksen artefakti. Tätä kehitettyä artefaktia sitten hyödynnettiin kahteen esimerkkitapaukseen, eli AES-128- ja 3DES-salausalgoritmiin. Tutkimus sisälsi kaksi tutkimuskysymystä:

1) Miten symmetrisiä lohkosalausalgoritmeja voidaan analysoida tehokkaasti, mutta yksinkertaisesti
2) Miten kehitettyä artefaktia hyödynnetään käytännössä?

Tutkimus saavutti tavoitteensa hyvin. Kehitetty analyysimenetelmä on toimiva, eikä sen käyttö vaadi syvällistä matemaattista osaamista tai perehtymistä salausalgoritmeihin. Ohjelmakoodin lukeminen todennäköisesti vaatii kohtalaista ohjelmointitaitoa, sillä arviointimallissa tarkastellaan ohjelmakoodia. Analyysin tulos on vertailukelpoinen muiden vastaavien analyysien kanssa, sillä Cormenin ja muiden (2009, s.12–14) mukaisesti tulos ohittaa ulkoisten tekijöiden, kuten ohjelmointitaidon tai tietokoneen suoritustehon, vaikutukset analyysin tulokseen.

6.1 Tutkimuksen rajoitteet ja luotettavuus

Tutkimuksella on omat rajoittavat tekijänsä. Koska se on sekä laajuudeltaan että aikataulultaan rajoitettu, on haasteellista kehittää artefakti, joka on todellakin lopullinen versio itsestään. Tämä realiteetti vaikuttaa tutkimukseen monella tavalla.

Esimerkiksi, kehitettyä analyysimenetelmää sovelletaan vain kahteen lohkosalausalgoritmiin, joten sen toimivuutta voi olla vaikea yleistää. Tämä tarkoittaa, että suunnittelutieteellisen tutkimuksen mukainen artefaktin jatkuva testaaminen ei välttämättä ole toteutunut vielä täydellisesti. Tämän ongelman minimoimiseksi testatut algoritmit olivat arkkitehtuuriltaan eri tyyppisiä, perustuen substituutio-permutaatio-

verkkoon ja Feistel-salaukseen. Näin analyysimallin yleistettävyyttä pyrittiin osoittamaan lohkosalausalgoritmien kontekstissa, sillä malli toimi molemmilla arkkitehtuureilla.

Laajuudesta, aikataulusta ja testauksesta johtuen ei myöskään ole takeita siitä, että artefakti on paras mahdollinen analyysimenetelmä suoritustehokkuudelle, jonka aineiston pohjalta olisi voinut kehittää. Testauksen avulla tämä iteraatioprosessi voisi jatkua myös pidemmälle.

Edellä mainituista syistä kehitetty artefakti on syytä nähdä täydellisen vastuksen sijasta tähän käyttötarkoitukseen nähden tarpeeksi hyvänä. Artefakti vastaa asetettuun tutkimusongelmaan, sekä saavuttaa tutkimuksen tavoitteen. Artefakti vastaa Cormenin ja muiden (2009, s.12) vaatimukseen ulkopuolisten tekijöiden eliminoinnista, eli tekijät kuten tietokoneen nopeus tai käyttäjän ohjelmointitaito eivät merkittävästi vaikuta analyysin sisältöön.

6.2 Tulosten vertailu aiempaan tutkimukseen

Algoritmianalyysin aiempi tutkimus on pitkälti yksittäisiä analyysejä joistain algoritmeista. Esimerkkejä tästä ovat esimerkiksi Pandan (2016, s.283), Durranin & Hayan (2022) ja Radhakrishan, Jadon & Honnavallin (2024) artikkelit. Tämä tutkimus eroaa aiemmasta käsittelemällä pikemminkin tapaa, jolla algoritmeja voi analysoida. Näin ollen, tämä tutkimus eroaa merkittävästi aiemmasta tutkimuksesta.

Tuotettu artefakti, eli tapa, jolla lohkosalausalgoritmeja voi analysoida, tukee kuitenkin tuloksillaan aiempaa tutkimusta. AES-128- ja 3DES-algoritmien analyysien tulokset, jotka tuotettiin artefaktin avulla, vastaavat aiempien tutkimusten eri metodologialla tuottamia tuloksia. Tämän tutkimuksen analyysien mukaan AES-128 on huomattavasti nopeampi salausalgoritmi kuin 3DES. Tämä sama raportoitiin esimerkiksi Iftikharin ja muiden (2021) ja Pandan (2016) artikkeleissa. Näin ollen, analyysimenetelmien eroavaisuuksista huolimatta analyysit tuottivat samankaltaiset tulokset.

Tämä tukee sekä tämän tutkimuksen, että aiemman tutkimuksen luotettavuutta. Jos tämän tutkimuksen tuottaman analyysimenetelmän ja aiemman tutkimuksen menetelmän tuloksissa olisi ollut havaittavissa eroja, olisi tämä kyseenalaistanut kehitetyn artefaktin luotettavuutta.

6.3 Mahdollisia jatkotutkimusaiheita

Tutkimuksen rajoitteista syntyy loogisia jatkotutkimusaiheita. Ensiksi mallia voitaisiin testata laajempaan joukkoon erilaisia lohkosalausalgoritmeja sen toimivuuden toteamiseksi. Tällöin suunnittelutieteellisen tutkimuksen testausiteraatiot toteutuisivat täydellisemmin, tukien tutkimuksen tuloksia.

Lisäksi artefaktia voitaisiin kehittää pitemmälle. Useampi suunnitteluiteraatio ehkä tuottaisi paremman version, joka vastaisi tutkimuksen tavoitteisiin vielä paremmin. Tutkimuksen rajoitteet asettavat pakotteensa suunnittelulle, joten mahdollisuudet parannukselle ovat todennäköisesti olemassa.

On myös todettava, että suunniteltu artefakti on vain yhden tutkimuksen näkemys siitä, miten lohkosalausalgoritmien suoritustehokkuutta voidaan arvioida. Kenties toinen tutkimus suunnittelisi samankaltaisen aineiston pohjalta erilaisen artefaktin. Tämä luo mahdollisuuden hyvään diskussioon, jossa useampi tutkimus tuo esiin oman näkemyksensä artefaktista. Näiden ideoiden synteesi mahdollisesti kehittäisi artefaktia pidemmälle.

Lähteet

Alsuwaiyel, M. (2016). Algorithms. Design techniques and analysis. World Scientific Publishing. ISBN: 9789814723640

Aumasson, J. (2018). Serious cryptography, A practical introduction to modern encryption. William Pollock. ISBN-13: 978-1-59327-826-7.

Bhandarkar, S. (2018). 3DES-Implementation-in-C. Github.com. Noudettu 7.11.2025 osoitteesta https://github.com/sanatb97/3DES-Implementation-in-C/tree/master.

Chandra, S., Mandal, B., Alam, Sk. S., & Bhattacharyya, S. (2015). Content Based Double Encryption Algorithm Using Symmetric Key Cryptography. *Procedia Computer Science*, *57*, 1228–1234. https://doi.org/10.1016/j.procs.2015.07.420

Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2009). Introduction to algorithms. Third edition. MIT Press. ISBN 978-0-262-03384-8.

Dibas, H., & Sabri, K. E. (2021). A comprehensive performance empirical study of the symmetric algorithms: AES, 3DES, Blowfish and Twofish. *2021 International Conference on Information Technology (ICIT)*, 344–349. https://doi.org/10.1109/ICIT52682.2021.9491644

Durrani, O., K. & Hayan, S., A. (2022). Performance measurement of popular sorting algorithms implemented using Java and Python. Proc. of the international conference on electrical, computer, communications and mechatronics engineering. DOI:10.1109/ICECCME55909.2022.9988424

Elo, S. & Kyngäs, H. (2008). The qualitative content analysis process. Journal of Advanced Nursing. https://doi.org/10.1111/j.1365-2648.2007.04569.x

Google Cloud. (2024). *What is encryption and how does it work?* Noudettu 4 marraskuuta 2024 osoitteesta https://cloud.google.com/learn/what-is-encryption

Gregor, S., & Hevner, A. (2013). Positioning and presenting design science research for maximum impact. MIS Quarterly. DOI: 10.25300/MISQ/2013/37.2.01

Hevner, A., R. (2007). A three cycle view of design science research. Scandinavian journal of information systems. http://aisel.aisnet.org/sjis/vol19/iss2/4

IBM. (2025). What is the Internet of Things. Noudettu 18.11.2025 osoitteesta https://www.ibm.com/think/topics/internet-of-things.

Iftikhar, U., Asrar, K., Waqas, M. & Ali, S. (2021). Evaluating the performance parameters of cryptographic algorithms for IoT-based devices. Engineering, technology & applied sciences research. https://doi.org/10.48084/etasr

Julkaisufoorumi. (2024). Julkaisufoorumi-luokituksen käyttöohje 2024. Noudettu 4.10.2025 osoitteesta https://julkaisufoorumi.fi/fi/kayttoohje.

Kikani, R. J., Verma, K., Navalakhe, R., Shrivastava, G., & Shrivastava, V. (2022). Cryptography: Recent research trends of encrypting mathematics. *Materials Today: Proceedings*, *56*, 3247–3253. https://doi.org/10.1016/j.matpr.2021.09.378

Oppliger, R. (2005). Contemporary cryptography. Artech House Inc. ISBN: 1-58053-642-5

Panda, M. (2016). Performance analysis of encryption algorithms for security. *International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES). DOI:* 10.1109/SCOPES.2016.7955835

Rosyidi, L., & Sari, R., F. (2016). A practical approach for complexity analysis of autonomic internet of things protocol algorithm. 19th international symposium on wireless personal multimedia communications. **ISBN:**978-4-9040-2009-8

Radhakrishan, I., Jadon, S. & Honnavalli, P. (2024). Efficiency and security of lightweight cryptographic algorithms for resource-constrained IoT devices. Sensors. https://doi.org/10.3390/s24124008

St. Denis, T., & Johnson, S. (2007). *Cryptography for developers*. Syngress Publishing, Inc. ISBN-13: 978-1-59749-104-4

Texas Instruments. (2011). AES-128 Software codec. Noudettu 18.6.2025 osoitteesta https://www.ti.com/tool/AES-128.

Vilkka, H. (2021). Tutki ja kehitä. PS-kustannus. ISBN 978-952-370-173-1.

Xilinx. (2020). AES Encryption algorithms. Noudettu 18.11.2025 osoitteesta https://xilinx.github.io/Vitis_Libraries/security/2020.1/guide_L1/internals/aes.html.

vom Brocke, J., Hevner, A., & Maedche, A. (2020). *Design Science Research. Cases*. Springer International Publishing AG. http://ebookcentral.proquest.com/lib/tritonia-ebooks/detail.action?docID=6356690

7 Liitteet

Liitteinä 3DES algoritmin käyttämät esilasketut taulukot (Bhandarkar, 2018).

```
/* Initial Permutation Table */
static char IP[] = {
    58, 50, 42, 34, 26, 18, 10,  2,
    60, 52, 44, 36, 28, 20, 12,  4,
    62, 54, 46, 38, 30, 22, 14,  6,
    64, 56, 48, 40, 32, 24, 16,  8,
    57, 49, 41, 33, 25, 17,  9,  1,
    59, 51, 43, 35, 27, 19, 11,  3,
    61, 53, 45, 37, 29, 21, 13,  5,
    63, 55, 47, 39, 31, 23, 15,  7
};

/* Inverse Initial Permutation Table */
static char PI[] = {
    40,  8, 48, 16, 56, 24, 64, 32,
    39,  7, 47, 15, 55, 23, 63, 31,
    38,  6, 46, 14, 54, 22, 62, 30,
    37,  5, 45, 13, 53, 21, 61, 29,
    36,  4, 44, 12, 52, 20, 60, 28,
    35,  3, 43, 11, 51, 19, 59, 27,
    34,  2, 42, 10, 50, 18, 58, 26,
    33,  1, 41,  9, 49, 17, 57, 25
};
```

Kuva 18 - Aloituspermutaatio & lopetuspermutaatio. (Bhandarkar, 2018).

```
/*Expansion table */
static char E[] = {
    32,  1,  2,  3,  4,  5,
     4,  5,  6,  7,  8,  9,
     8,  9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32,  1
};

/* Post S-Box permutation */
static char P[] = {
    16,  7, 20, 21,
    29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2,  8, 24, 14,
    32, 27,  3,  9,
    19, 13, 30,  6,
    22, 11,  4, 25
};
```

Kuva 19 - Laajennusfunktio & salauskierroksen permutaatio. (Bhandarkar, 2018).

```
/* The S-Box tables */
static char S[8][64] = {{
    /* S1 */
    14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
     0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
     4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
    15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13
},{
    /* S2 */
    15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
     3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
     0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
    13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9
},{
    /* S3 */
    10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
    13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
    13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
     1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12
},{
    /* S4 */
     7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
    13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
    10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
     3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14
},{
    /* S5 */
     2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
    14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
     4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
    11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3
},{
    /* S6 */
    12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
    10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
     9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
     4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13
},{
    /* S7 */
     4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
    13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
     1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
     6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12
},{
    /* S8 */
    13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
     1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
     7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
     2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
}};
```

Kuva 20 - Sboxit. (Bhandarkar, 2018).

```
/* Permuted Choice 1 Table */
static char PC1[] = {
    57, 49, 41, 33, 25, 17,  9,
     1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27,
    19, 11,  3, 60, 52, 44, 36,

    63, 55, 47, 39, 31, 23, 15,
     7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29,
    21, 13,  5, 28, 20, 12,  4
};

/* Permuted Choice 2 Table */
static char PC2[] = {
    14, 17, 11, 24,  1,  5,
     3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
};

/* Iteration Shift Array */
static char iteration_shift[] = {
 /* 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16 */
    1,  1,  2,  2,  2,  2,  2,  2,  1,  2,  2,  2,  2,  2,  2,  1
};
```

Kuva 21 - Avainalgoritmin taulukot. (Bhandarkar, 2018).