# POTENSSIINKOROTUSALGORITMIT JA NIIDEN VERTAILU RSA:SSA

Mirka Räzm

Pro gradu -tutkielma
Kesäkuu 2012

MATEMATIIKAN JA TILASTOTIETEEN LAITOS
TURUN YLIOPISTO

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

RÄZM, MIRKA: Potenssiinkorotusalgoritmit ja niiden vertailu RSA:ssa
Pro gradu -tutkielma, 44 s.
Matematiikka
Kesäkuu 2012

---

Tässä tutkielmassa tutustutaan potenssiinkorotusalgoritmeihin ja vertaillaan niitä RSA:ssa. Kun viestejä salataan RSA menetelmää käyttäen, suoritetaan paljon potenssiin korotuksia $x^e \pmod{n}$. Potensiin korotuksessa käytettävät luvut ovat suuria, joten tietokoneen muistia tarvitaan paljon. Tämän vuoksi tarvitaan menetelmiä, joilla potenssiin korotukset voidaan suorittaa vaiheittain.

Ensin tutkielmassa esitetään RSA:n toiminta periaate. Pääpaino tutkielmassa on potenssiinkorotusalgoritmeissa. Potenssiinkorotusmenetelmät on esitetty algoritmeina ja jokaisen algoritmin yhteydessä on esimerkkejä niiden toiminnasta. Algoritmit on jaettu kolmeen ryhmään. Ensimmäisenä on esitelty yleisiä potenssiinkorotusalgoritmeja, jotka sopivat useimpiin tilanteisiin. Kun eksponentti pysyy vakiona, kuten RSA:ssa, potenssiin korotus yhteenlaskuketjumenetelmällä on hyvä valinta. Tämä nopeuttaa laskentaa verrattuna yleisiin potenssiinkorotusalgoritmeihin. Viimeiseksi on esitettynä potenssiinkorotusalgoritmeja tilanteisiin, joissa kanta pysyy vakiona. Nämä eivät nopeuta laskentaa RSA:ssa, mutta ne otettu mukaan työhön, koska potenssiin korotuksen pystyy näilläkin algoritmeilla suorittamaan ja algoritmit ovat tehokkaita silloin, kun suoritetaan toistuvia potenssiin korotuksia kannan pysyessä vakiona.

Tutkielman lopussa on esimerkki, joka vertailee algoritmien tehokkuutta, kun salataan sama viesti kullakin menetelmällä. Lopuksi on vielä pohdittu algoritmien tehokkuutta salattaessa viestiä RSA menetelmällä todellisen suuruisilla luvuilla.

Asiasanat: kryptografia, potenssiin korotus, RSA, algoritmi.

# Sisältö

# 1 Johdanto

RSA kryptosysteemiä käytettäessä suoritetaan paljon potenssiin korotuksia $x^e \pmod{n}$, jossa luvut $e$ ja $n$ ovat suuria. Nykytekniikalla lukujen täytyy olla suuria, koska muuten ulkopuolinen voisi murtaa viestin liian helposti. Tietokoneen muisti ei riitä käsittelemään potenssiin korotusta, jossa ensin lasketaan $x^e$ ja sitten lasketaan modulo $n$. Tarvitaan menetelmiä, joilla potenssiin korotus voidaan suorittaa vaiheittain. Yksinkertaisin menetelmä olisi suorittaa $(e-1)$ kertolaskua, jossa jokaisen kertolaskun jälkeen tulosta lasketaan modulo $n$. Käytännössä luku $e$ on niin suuri, että tähän kuluu liian paljon aikaa. Tarvitaan menetelmiä, jotka mahdollistavat potenssiin korotuksen lyhyemmässä ajassa ja vähemmällä työllä.

Työn toinen luku esittelee RSA protokollaa. Tässä esitetään lyhyesti kuinka avaimet luodaan ja kuinka viestien salaaminen ja purkaminen käytännössä toimii. RSA menetelmää voidaan käyttää myös digitaalisessa allekirjoituksessa ja tämä periaate esitetään myös. Luku kaksi pohjautuu lähteisiin [1], [3], [4], [5] ja [6].

Pääpaino tässä työssä on esitellä potenssiinkorotusalgoritmeja, jotka vähentävät ja nopeuttavat laskentaa. Kolmas luku, jossa nämä algoritmit esitellään, pohjautuu kirjaan Handbook of Applied Cryptography [2]. Algoritmien yhteydessä on esimerkkejä, joiden tarkoitus on selventää algoritmin toimintaa. Näissä esimerkeissä luvut eivät ole kovin suuria, suurimmaksi osaksi siitä syystä, että laskuvaiheet paperille kirjoitettuna tarvitsevat paljon tilaa.

Ensin luvussa esitetään yleisiä potenssiinkorotusalgoritmeja. Algoritmit toimivat hyvin tilanteissa, joissa yleisesti tarvitaan potenssiin korotusta. Potenssiin korotus käyttäen yhteenlaskuketjumenetelmää on suunnattu tilanteeseen, jossa eksponentti $e$ pysyy vakiona, kuten RSA:ssa. Viimeiset algoritmit ovat käteviä tilanteissa, joissa kantaluku $x$ pysyy samana. RSA ei ole tälläinen protokolla, mutta nämä algoritmit on otettu työhön mukaan, koska näilläkin potenssiin korotus pystytään suorittamaan. Näiden algoritmien työläs esilaskenta hidastaa tehokasta RSA:n käyttöä.

Algoritmien esittelyjen jälkeen on pohdittu niiden laskennallista tehokkuutta. Työssä oletetaan, että kertolasku ja neliöiminen vaativat suunnilleen

yhtä paljon työtä ja näiden laskutoimitusten määrän laskeminen antaa viitteitä algoritmin tehokkuudesta.

Viimeisessä, neljännessä, luvussa on kokoava esimerkki kaikista esitetyistä algoritmeista. Tässä luodaan RSA avaimet ja salataan sama viesti kaikilla esitellyillä algoritmeilla. Nämäkään lukuarvot eivät ole todellisuuteen verrattaessa riittävän suuria, mutta ovat hieman suuremmat, kuin kolmannen luvun esimerkeissä. Luvun lopussa on yhteenveto kuinka paljon työtä tarvittiin viestin salaamiseen kullakin menetelmällä. Tämä on vain suuntaa antava taulukko, joka pätee vain näihin lukuarvoihin. Parhain käytettävä menetelmä on aina valittava käyttötarkoituksen mukaan. Esimerkiksi viimeiset kolme algorimia ovat erittäin nopeita ja tehokkaita, mutta algoritmi ei ole käytännöllinen RSA menetelmässä. Esilaskenta näissä on hyvin työläs ja vie pohjan RSA menetelmää käytettäessä, koska kertaalleen esilaskettuja lukuja ei voi uudelleenkäyttää. Neljänneksi viimeisin algoritmi, potenssiin korotus yhteenlaskuketjumenetelmällä, on kätevä RSA protokollassa, kunhan löytää eksponentille $e$ hyvän yhteenlaskuketjun. RSA:ssa eksponentti $e$ pysyy samana, joten yhteenlaskuketjua ei joka kerta tarvitse muodostaa uudelleen, vaan voi hyödyntää samaa ketjua.

Lopuksi on pohdittu algorimien tehokkuutta todellisen suuruisilla luvuilla. Esitetyt luvut ovat karkeita arvioita pohjautuen yleiseen tilanteeseen, jossa oletetaan, että luvuissa $e$ ja $x$ kummassakin on 1000 bittiä.

# 2 RSA

RSA on yksi tunnetuimmista julkisen avaimen kryptosysteemeistä. Julkisen avaimen kryptosysteemissä on sekä julkinen avain että salainen avain ja salaista avainta ei pitäisi pystyä selvittämään julkisen avaimen avulla. RSA sopii viestien salaamiseen sekä digitaaliseen allekirjoittamiseen.

Olkoon $n = pq$, jossa $p$ ja $q$ ovat suuria satunnaisia alkulukuja. RSA:n julkinen avain on pari $(n, e)$, jossa $\text{syt}(e, \varphi(n)) = 1$. Tässä $\varphi(n) = (p-1)(q-1)$. Salainen avain on pari $(n, d)$, jossa $d$ on $e$:n käänteisluku modulo $\varphi(n)$, siis $ed = 1 \pmod{\varphi(n)}$. Kryptausfunktio on

$$E(x) = E_{n,e}(x) = x^e \pmod{n}$$

ja dekryptausfunktio on

$$D(x) = D_{n,d}(x) = x^d \pmod{n}.$$

Eulerin lauseen mukaan $x^{\varphi(n)} = 1 \pmod{n}$ ja siten

$$D(E(x)) = x^{ed} = x^{1+k\varphi(n)} = x \cdot x^{k\varphi(n)} = x \pmod{n}.$$

RSA allekirjoitus toimii vastaavalla periaatteella. Allekirjoittaja allekirjoittaa viestin $m$ omilla salaisilla RSA avaimillaan $(n, d)$:

$$s = m^d \pmod{n}.$$

Allekirjoitus varmistetaan allekirjoittajan julkisilla RSA avaimilla $(n, e)$ vertaamalla onko allekirjoitettu viesti sama kuin alkuperäinen:

$$m = s^e \pmod{n}.$$

Seuraavan esimerkin avulla selvennetään, miten viestien salaaminen ja purku yllä esitetyllä RSA kryptosysteemillä toimii.

**Esimerkki 1.** *Liisa ja Pekka haluavat käyttää RSA menetelmää salaamaan viestinsä. Pekka valitsee alkuluvut* $p = 11$ *ja* $q = 17$*. Hän laskee* $n = pq = 11 \cdot 17 = 187$ *ja* $\varphi(n) = 10 \cdot 16 = 160$*. Sitten Pekka valitsee luvun* $e = 7$ *ja*

*varmistaa, että* $\text{syt}(e, \varphi(n)) = 1$ *ja laskee* $d = e^{-1} \pmod{\varphi(n)} = 23$. *Pekan julkinen avain on* $(n, e) = (187, 7)$, *jonka hän julkaisee kaikille. Salaisen avaimen* $(n, d) = (187, 23)$ *hän pitää itsellään. Pekka ei tarvitse enää lukuja* $p$ *ja* $q$, *mutta hänen on pidettävä huoli, ettei kukaan saa niitä selville. Liisa haluaa lähettää Pekalle viestin* $x = 6$. *Hän käyttää Pekan julkista avainta ja laskee* $E(x) = x^e \pmod{n} = 6^7 = 184 \pmod{187}$. *Liisa lähettää Pekalle salatun viestin* $y = 184$. *Pekka käyttää salaista avaintaan avatakseen viestin* $x = D(y) = 184^{23} = \cdots = 6 \pmod{187}$.

Vastaavalla tavalla kuka tahansa voi tehdä itselleen RSA avaimet ja julkaista julkisen avaimensa. Kaikki saavat nähdä julkiset avaimet ja voivat lähettää niiden julkaisijalle salattuja viestejä.

Alkuluvut yllä olevassa esimerkissä ovat aivan liian pieniä. Kuka tahansa voi löytää luvun $n = 187$ alkutekijät ja siten pystyy laskemaan salatun avaimen julkisen avaimen avulla. Jotta saataisiin parempi turvallisuus, lukujen täytyy olla paljon suurempia.

Yleisimmät salausmenetelmät perustuvat suuren luvun, joka on kahden alkuluvun tulo, tekijöihin jaon vaikeuteen. Myös RSA on tällainen salausmenetelmä. Jos luku on liian pieni, tekijöiden löytäminen on helppoa. Jos luku on riittävän iso, niin laskennallisesti kestää hyvin kauan löytää luvun alkutekijät. Tekniikka kehittyy koko ajan ja aina suurempia lukuja voidaan jakaa tekijöihin. Suuremmat alkuluvut lisäävät turvallisuutta, mutta hidastavat ja vaikeuttavat viestien salaamista.

Esimerkiksi vuonna 1991 RSA Data Security antoi joukon haasteita, joiden tarkoitus oli selvittää tekijöihin jaon vaikeutta. Listassa oli 41 lukua, jotka olivat kahden suunnilleen samanpituisen alkuluvun tuloja. Vuoteen 1999 mennessä luvuista viisi ensimmäistä oli onnistuttu selvittämään. Nämä luvut olivat suuruudeltaan sadasta numerosta 140 numeroon, eli 330-463 bittiä pitkiä. Samana vuonna oli myös onnistuttu jakamaan tekijöihin avaimen pituutena suosittu 512 bittiä (155 numeroa) pitkä luku. Vuonna 2009 onnistuttiin jakamaan 768 bittiä (232 numeroa) pitkä luku tekijöihin.

Laskettaessa potenssiin korotusta $x^e \pmod{n}$ ensin ei lasketa $x^e$ ja sitten suoriteta jakolaskua saadaksemme jakojäännöksen. Tarvittava tila luvun $x^e$ laskemiseksi on valtava. Oletetaan, että luvuissa $x$ ja $e$ on kummassakin 256

bittiä. Silloin tarvitaan

$$log_2(x^e) = e \cdot log_2 x \approx 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

bittiä luvun $x^e$ tallentamiseksi, joka vie liikaa muistia. Potenssiin korotus täytyy jakaa pienempiin osiin ja jokaisessa laskun välivaiheessa täytyy laskea tuloksen modulo $n$, jotta luvut pysyvät käsiteltävissä olevissa suuruuksissa.

# 3 Potenssiin korotus

Potenssiin korotuksessa $x^e$ kanta $x$ on äärellisen joukon $G$ alkio ja eksponentti $e$ on positiivinen kokonaisluku. Potenssiin korotusta $x^e$ merkitään tässä työssä $x^e \pmod n$. Potenssiin korotuksessa on tärkeää, että on olemassa tehokas tapa kertoa kaksi lukua modulo $n$. Laskettaessa potenssiin korotusta $x^e \pmod n$ olisi yksinkertaisinta suorittaa $e-1$ kertolaskua. Kryptografisissa sovelluksissa luvun $n$ suuruus ylittää luvun $2^{160}$ ja voi ylittää luvun $2^{1024}$. Usein $e$ valitaan niin suureksi, että ei ole järkevää laskea $e-1$ kertolaskua. On olemassa kaksi tapaa vähentää aikaa laskettaessa potenssiin korotusta $x^e$. Toinen on lukujen kertolaskuun käytettävän ajan vähentäminen ja toinen on vähentää kertolaskujen määrää potenssiin korotuksessa $x^e$. Parhaimmassa tapauksessa on olemassa tapa, joka sisältää molemmat.

## 3.1 Menetelmiä yleiseen potenssiin korottamiseen

Edellä esitetty tapa laskea potenssiin korottaminen suorittamalla $e-1$ kertolaskua ei ole käytännöllinen, mutta sitä voidaan muokata. Jos esimerkiksi pitäisi laskea $x^{15} \pmod n$, niin yllä olevaa tapaa käyttämällä saadaan

$$\begin{aligned} x^2 &= x \cdot x \pmod n \\ x^3 &= x^2 \cdot x \pmod n \\ x^4 &= x^3 \cdot x \pmod n \\ &\dots \\ x^{15} &= x^{14} \cdot x \pmod n. \end{aligned}$$

Edellä ollutta laskua voidaan huomattavasti yksinkertaistaa. Kaikkia $x$:n potensseja ei tarvitse laskea saavuttaaksemme $x^{15} \pmod n$:

$$\begin{aligned} x^2 &= x \cdot x \pmod n \\ x^3 &= x^2 \cdot x \pmod n \\ x^6 &= x^3 \cdot x^3 \pmod n \\ x^7 &= x^6 \cdot x \pmod n \\ x^{14} &= x^7 \cdot x^7 \pmod n \\ x^{15} &= x^{14} \cdot x \pmod n. \end{aligned}$$

Tällä menetelmällä tarvittiin vain kuusi kertolaskua neljäntoista sijaan.

Samalla tavalla voidaan laskea $x^e$ millä tahansa $e$:n arvolla. Menetelmää kutsutaan *kerro ja neliöi* -menetelmäksi. Tämä luku esittelee yleiskäyttöisiä potenssiinkorotusalgoritmeja, joita voidaan sanoa toistetuiksi kerro ja neliöi -algoritmeiksi.

### 3.1.1 Perusbinäärinen ja $2^k$-kantainen potenssiin korottaminen

Alla olevassa algoritmissa (3.1) potenssiin korotuksen $x^e \pmod{n}$ laskemiseksi käytetään apumuuttujia $A$ ja $S$. Kohdassa 1 muuttujaan $A$ talletetaan luku 1 ja muuttujaan $S$ talletetaan $x$. Kohdassa 2 muuttujaan $A$ talletetaan $A \cdot S \pmod{n}$ vain, jos luku $e$ on pariton. Tämän jälkeen luku $e$ saa uuden arvon, joka on $e$ jaettuna kahdella ja pyöristettynä alaspäin lähimpään kokonaislukuun. Niin kauan kuin $e$ on erisuuri kuin nolla, muuttujaan $S$ talletetaan $S \cdot S \pmod{n}$ ja kierros aloitetaan alusta. Kohtaa 2 jatketaan niin kauan kunnes $e = 0$. Tämän jälkeen palautetaan muuttuja $A$, jossa on laskun $x^e \pmod{n}$ tulos.

---

**Algoritmi 3.1.** Binäärinen oikealta vasemmalle etenevä potenssiin korotus
SYÖTE: $x$, $n$ ja kokonaisluku $e \geq 1$.
TULOSTE: $x^e \pmod{n}$.

1. $A \leftarrow 1$, $S \leftarrow x$.
2. Niin kauan kuin $e \neq 0$, niin
   - 2.1 Jos $e$ on pariton, niin $A \leftarrow A \cdot S \pmod{n}$.
   - 2.2 $e \leftarrow \lfloor e/2 \rfloor$.
   - 2.3 Jos $e \neq 0$ niin $S \leftarrow S \cdot S \pmod{n}$.
3. Palauta $(A)$.

---

**Esimerkki 2.** *Taulukossa 1 näkyy lukujen $A$, $e$ ja $S$ saamat arvot algoritmin jokaisella kierroksella, kun yllä olevalla algoritmilla lasketaan $x^{283}$. Ensimmäisessä sarakkeesa on muuttujien saamat arvot algoritmin ensimmäisessä kohdassa. Algoritmin kohtaa 2 käydään läpi niin kauan, kunnes $e = 0$. Lopuissa sarakkeissa on muuttujien saamat arvot näillä algoritmin toisen kohdan kierroksilla.*

| $A$ | 1 | $x$ | $x^3$ | $x^3$ | $x^{11}$ | $x^{27}$ | $x^{27}$ | $x^{27}$ | $x^{27}$ | $x^{283}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $e$ | 283 | 141 | 70 | 35 | 17 | 8 | 4 | 2 | 1 | 0 |
| $S$ | $x$ | $x^2$ | $x^4$ | $x^8$ | $x^{16}$ | $x^{32}$ | $x^{64}$ | $x^{128}$ | $x^{256}$ | – |

Taulukko 1

*Kun $e = 0$, algoritmin kolmas kohta palauttaa potenssiin korotuksen tuloksen $A = x^{283}$.*

**Esimerkki 3.** *Käytetään binääristä oikealta vasemmalle etenevää menetelmää RSA:ssa. Käytetään esimerkissä 1 olevia arvoja $n = 187$, $e = 7$ ja $d = 23$. Kryptataan viesti $x = 123$. Alla olevassa taulukossa 2 on näkyvillä algoritmin laskuvaiheet laskussa $y = 123^7 \pmod{187}$.*

| $A$ | 1 | 123 | 30 | 183 |
|---|---|---|---|---|
| $e$ | 7 | 3 | 1 | 0 |
| $S$ | 123 | 169 | 137 | – |

Taulukko 2

*Algoritmi palauttaa viimeiseksi muuttujaan $A$ talletetun luvun. Salattu viesti on $y = 183 \pmod{187}$. Käytetään samaa algoritmia salatun viestin selvittämiseksi. Taulukossa 3 on esillä algoritmin laskuvaiheet, kun lasketaan $y^d = 183^{23} \pmod{n}$.*

| $A$ | 1 | 183 | 123 | 72 | 72 | 123 |
|---|---|---|---|---|---|---|
| $d$ | 23 | 11 | 5 | 2 | 1 | 0 |
| $S$ | 183 | 16 | 69 | 86 | 103 | – |

Taulukko 3

*Tästä saimme alkuperäisen viestin* 123.

Olkoon luvun $e$ binääriesityksen pituus $t + 1$ bittiä ja olkoon $\mathrm{wt}(e)$ ykkösten lukumäärä binääriesityksessä. Algoritmi 3.1 suorittaa $t$ neliöintiä ja

$\mathrm{wt}(e)-1$ kertolaskua. Jos $e$ valitaan satunnaisesti väliltä $0 \leq e < n$, niin voidaan odottaa noin $\lfloor log_2\ n \rfloor$ neliöintiä ja $\frac{1}{2}(\lfloor log_2\ n \rfloor + 1)$ kertolaskua. Laskua $1 \cdot x$ ei lasketa kertolaskuksi, eikä laskua $1 \cdot 1$ lasketa neliöinniksi. Jos neliöinti on suunnilleen yhtä arvokas kuin kertolasku, niin odotettu määrä työtä on noin $\frac{3}{2}\lfloor log_2\ n \rfloor$ kertolaskua.

Algoritmissa 3.2 tarvitsemme eksponentin $e$ binääriesityksen $e = (e_t e_{t-1} \cdots e_1 e_0)_2$ ja käsittelemme sen merkitsevimmästä bitistä $e_t$ vähiten merkitsevään bittiin $e_0$. Muuttujaan $A$ kerääntyy potenssiin korotuksen tulos. Muuttujaan $A$ asetetaan aluksi luku 1. Seuraavaksi eksponentti käsitellään bitistä $e_t$ bittiin $e_0$. Joka kierroksella muuttujan $A$ arvo neliöidään ja muuttuja kerrotaan luvulla $x$ mikäli bitti $e_i$ on yksi.

---

**Algoritmi 3.2.** Binäärinen vasemmalta oikealle etenevä potenssiin korotus
SYÖTE: $x$, $n$ ja positiivinen kokonaisluku $e = (e_t e_{t-1} \cdots e_1 e_0)_2$.
TULOSTE: $x^e \pmod n$.

1. $A \leftarrow 1$.
2. Kun $i$ käy luvusta $t$ nollaan, niin
   2.1 $A \leftarrow A \cdot A \pmod n$.
   2.2 Jos $e_i = 1$, niin $A \leftarrow A \cdot x \pmod n$.
3. Palauta $(A)$.

---

**Esimerkki 4.** *Taulukossa 4 on esitettynä yllä olevan algoritmin laskutoimitusten laskuvaiheet, kun lasketaan* $x^{283}$. *Koska* $283 = (100011011)_2$, *niin* $t = 8$.

| $i$ | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| $e_i$ | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| $A$ | $x$ | $x^2$ | $x^4$ | $x^8$ | $x^{17}$ | $x^{35}$ | $x^{70}$ | $x^{141}$ | $x^{283}$ |

Taulukko 4

*Algoritmi palauttaa muuttujaan A tallennetun potenssiin korotuksen tuloksen* $x^{283}$.

**Esimerkki 5.** *Olkoon jälleen* $e = 7$, $d = 23$, $n = 187$ *ja* $x = 123$, *kuten esimerkissä 3. Salataan viesti* $x = 123$ *käyttäen yllä olevaa algoritmia. Nyt* $e = 7 = (111)_2$ *ja* $t = 2$. *Taulukossa 5 on esitettynä algoritmin suorittamat laskuvaiheet.*

| $i$ | 2 | 1 | 0 |
|---|---|---|---|
| $e_i$ | 1 | 1 | 1 |
| $A$ | 123 | 30 | 183 |

Taulukko 5

*Potenssiin korotuksesta saatiin* $y = 183$, *niin kuin kuuluukin. Dekryptataan viesti* $y = 183$. *Nyt avain* $d = 23 = (10111)_2$ *ja* $t = 4$. *Taulukossa 6 on suoritetut laskuvaiheet.*

| $i$ | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| $e_i$ | 1 | 0 | 1 | 1 | 1 |
| $A$ | 183 | 16 | 98 | 106 | 123 |

Taulukko 6

*Algoritmi palauttaa alkuperäisen viestin* $x = 123$.

Olkoon luvun $e$ binääriesityksen bittien lukumäärä $t + 1$ ja $\mathrm{wt}(e)$ ykkösten lukumäärä binääriesityksessä. Algoritmi 3.2 suorittaa $t + 1$ neliöintiä ja $\mathrm{wt}(e) - 1$ kertolaskua. Kertolaskujen ja neliöintien määrä on sama kuin algoritmissa 3.1, mutta tässä algoritmissa kerrotaan aina määrätyllä luvulla $x$. Jos luvulla $x$ on erityinen rakenne, niin kertolasku voi olla huomattavasti helpompaa kuin kahden satunnaisen luvun kertominen.

Algoritmia 3.3 voidaan sanoa potenssiin korotuksen *ikkunamenetelmäksi*. Tämä on yleistys algoritmista 3.2 ja se käsittelee eksponentista useamman kuin yhden bitin kerrallaan.

---

**Algoritmi 3.3.** $2^k$-kantainen vasemmalta oikealle etenevä potenssiin korotus
SYÖTE: $x$, $n$ ja $e = (e_t e_{t-1} \cdots e_1 e_0)_b$, missä $b = 2^k$ jollain $k \geq 1$.
TULOSTE: $x^e \pmod{n}$.

1. Esilaskenta.
   - 1.1 $x_0 \leftarrow 1$.
   - 1.2 Kun $i$ käy 1:stä lukuun $(2^k - 1)$, laske
     $x_i \leftarrow x_{i-1} \cdot x \pmod{n}$ (Siis $x_i = x^i$).
2. $A \leftarrow 1$.
3. Kun $i$ käy $t$:stä nollaan, niin
   - 3.1 $A \leftarrow A^{2^k} \pmod{n}$.
   - 3.2 $A \leftarrow A \cdot x_{e_i} \pmod{n}$.
4. Palauta $(A)$.

---

Algoritmi 3.3 saa syötteenä luvut $x, n$ ja $e$ sekä luvun $k$. Luku $e$ esitetään $2^k$-kantaisena. Esilaskennassa lasketaan $x_i = x^i \pmod{n}$ kaikilla $i$:n arvoilla yhdestä lukuun $(2^k - 1)$ saakka ja $x_0 = 1$. Muuttujaan $A$ talletetaan aluksi luku 1. Kun $i$ käy luvusta $t$ nollaan, muuttujaan $A$ talletetaan joka kierroksella ensin $A^{2^k} \pmod{n}$ ja tämän jälkeen $A \cdot x_{e_i} \pmod{n}$. Lopulta muuttujaan $A$ on saatu potenssiin korotuksen tulos.

**Esimerkki 6.** *Lasketaan $x^e$, kun $e = 250$. Valitaan $k = 2$, joten $e = (3322)_4$.*
*Kohta 1 Esilaskenta ($i$ käy yhdestä lukuun $(2^2 - 1) = 3$):*

$x_0 = 1$
$x_1 = x_0 \cdot x = x$
$x_2 = x_1 \cdot x = x^2$
$x_3 = x_2 \cdot x = x^3$.

*Kohdassa 2 muuttujaan $A$ talletetaan luku 1.*
*Kohdassa 3 $i$ käy kolmesta nollaan:*

$i = 3$ $A = 1^{2^2} = 1$
$A = 1 \cdot x_3 = x^3$
$i = 2$ $A = (x^3)^4 = x^{12}$
$A = x^{12} \cdot x^3 = x^{15}$

$i = 1 \quad A = (x^{15})^4 = x^{60}$

$\qquad\quad A = x^{60} \cdot x^2 = x^{62}$

$i = 0 \quad A = (x^{62})^4 = x^{248}$

$\qquad\quad A = x^{248} \cdot x^2 = x^{250}.$

*Nyt* $A = x^{250}$*, jonka algoritmi palauttaa laskutoimituksen tuloksena.*

*Lasketaan sama laskutoimitus, mutta valitaan nyt* $k = 3$ *ja siten* $e = (372)_8$*.*

1. *Esilaskenta* $((2^3 - 1) = 7)$*:*
$x_0 = 1$, $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$, $x_4 = x^4$, $x_5 = x^5$, $x_6 = x^6$, $x_7 = x^7$.
2. $A = 1$.
3. *i käy luvusta* 2 *nollaan:*

$i = 2 \quad A = 1^8 = 1$

$\qquad\quad A = 1 \cdot x_3 = x^3$

$i = 1 \quad A = (x^3)^8 = x^{24}$

$\qquad\quad A = x^{24} \cdot x_7 = x^{24} \cdot x^7 = x^{31}$

$i = 0 \quad A = (x^{31})^8 = x^{248}$

$\qquad\quad A = x^{248} \cdot x_2 = x^{248} \cdot x^2 = x^{250}.$

4. *Palauta* $A = x^{250}$*.*

*Luvun* $k$ *ollessa suurempi esilaskennan määrä kasvaa, mutta kohdan* 3 *laskukierroksia tarvitaan vähemmän.*

**Esimerkki 7.** *Lasketaan* $x^{11749}$ *käyttäen algoritmia 3.3 ja olkoon* $k = 3$*. Tähän esimerkkiin tullaan viittaamaan algoritmin 3.5 esimerkkien yhteydessä. Nyt* $e = (26745)_{2^3}$*.*

*Esilaskennasta saadaan* $x_0 = 1$, $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$, $x_4 = x^4$, $x_5 = x^5$, $x_6 = x^6$ *ja* $x_7 = x^7$*.*

*Taulukossa 7 on esitetty kohdassa* 3 *suoritettavat laskuvaiheet. Kohdassa* 3.1 *muuttujaan* $A$ *talletettaan* $A^{2^3} = A^8$ *ja kohdassa* 3.2 *muuttujaan* $A$ *talletetaan* $A \cdot x_{e_i}$*.*

*Algoritmi palauttaa potenssiin korotuksen tuloksen* $A = x^{11749}$*.*

| $i$ | 3.1 | 3.2 |
|---|---|---|
| 4 | 1 | $x^2$ |
| 3 | $x^{16}$ | $x^{16} \cdot x^6 = x^{22}$ |
| 2 | $x^{176}$ | $x^{176} \cdot x^7 = x^{183}$ |
| 1 | $x^{1464}$ | $x^{1464} \cdot x^4 = x^{1468}$ |
| 0 | $x^{11744}$ | $x^{11744} \cdot x^5 = x^{11749}$ |

Taulukko 7

Seuraavassa algoritmissa algoritmia 3.3 muokataan hieman, jotta saadaan vähennettyä esilaskennan määrää. Algoritmissa 3.4 käytetään seuraavaa merkintätapaa: jos $e_i \neq 0$, kun $0 \leq i \leq t$, niin merkitään $e_i = 2^{h_i} u_i$, jossa $u_i$ on pariton. Jos $e_i = 0$, niin olkoon $h_i = 0$ ja $u_i = 0$. Esimerkiksi olkoon $e = 250 = (3322)_4$. Edellä esitetystä merkintätavasta saadaan $e_3 = e_2 = 3 = 2^0 \cdot 3$, joten $h_3 = h_2 = 0$, $u_3 = u_2 = 3$ ja $e_1 = e_0 = 2 = 2^1 \cdot 1$, joten $h_1 = h_0 = 1$ ja $u_1 = u_0 = 1$.

Algoritmi saa syötteenä luvut $x, n, e$ ja $k$. Luku $e$ esitetään $2^k$-kantaisena lukuna. Esilaskennassa lasketaan aina luvut $x_0, x_1$ ja $x_2$. Kun luku $i$ käy yhdestä lukuun $(2^{k-1}-1)$, niin lasketaan vielä luvut $x_{2i+1} = x_{2i-1} \cdot x_2 \pmod{n}$. Käytännössä laskemme vain joka toisen $x$:n potenssin luvusta $x_3 = x^3$ lähtien. Esilaskennan jälkeen muuttujaan $A$ talletetaan luku 1. Kun luku $i$ käy luvusta $t$ nollaan, talletetaan joka kierroksella muuttujaan $A = (A^{2^{k-h_i}} \cdot x_{u_i})^{2^{h_i}} \pmod{n}$.

---

**Algoritmi 3.4.** Muokattu $2^k$-kantainen vasemmalta oikealle etenevä potenssiin korotus
SYÖTE: $x$, $n$ ja $e = (e_t e_{t-1} \cdots e_1 e_0)_b$, missä $b = 2^k$ jollain $k \geq 1$.
TULOSTE: $x^e \pmod{n}$.

1. Esilaskenta.
   1.1 $x_0 \leftarrow 1$, $x_1 \leftarrow x$, $x_2 \leftarrow x^2 \pmod{n}$.

1.2 Kun $i$ käy 1:stä lukuun $(2^{k-1}-1)$, niin
$x_{2i+1} \leftarrow x_{2i-1} \cdot x_2 \pmod{n}$.

2. $A \leftarrow 1$ .
3. Kun $i$ saa arvot $t$:stä nollaan, niin $A \leftarrow (A^{2^{k-h_i}} \cdot x_{u_i})^{2^{h_i}} \pmod{n}$.
4. Palauta $(A)$.

---

**Esimerkki 8.** *Lasketaan yllä esitettyä algoritmia käyttäen $x^e$, jossa $e = 250$, kuten esimerkissä 6. Valitaan $k = 2$, jolloin saadaan $e = (3322)_{2^2}$.*

1. *Esilaskenta.*
   1.1 $x_0 = 1$, $x_1 = x$ *ja* $x_2 = x^2 \pmod{n}$.
   1.2 $i = 1 : x_3 = x_1 \cdot x_2 = x^3 \pmod{n}$.
2. $A = 1$.
3. *Kuten ennen algoritmia 3.4 esitettiin, niin merkintätavasta* $e_i = 2^{h_i}u_i$ *saadaan* $e_3 = e_2 = 3 = 2^0 \cdot 3$, *joten* $h_3 = h_2 = 0$, $u_3 = u_2 = 3$ *ja* $e_1 = e_0 = 2 = 2^1 \cdot 1$, *joten* $h_1 = h_0 = 1$ *ja* $u_1 = u_0 = 1$.
*Nyt $i$ käy luvusta 3 lukuun 0:*
   $i = 3 : A = (A^{2^{2-h_3}} \cdot x_{u_3})^{2^{h_3}} = (1 \cdot x_3)^{2^0} = x^3$
   $i = 2 : A = ((x^3)^{2^2} \cdot x^3)^{2^0} = x^{12} \cdot x^3 = x^{15}$
   $i = 1 : A = ((x^{15})^{2^1} \cdot x)^2 = x^{62}$
   $i = 0 : A = ((x^{62})^{2^1} \cdot x)^2 = x^{250}$.

*Algoritmi palauttaa potenssiin korotuksen tuloksen $A = e^{250}$.*

*Valitaan nyt $k = 3$ ja saadaan $e = (372)_{2^3}$. Lasketaan sama lasku tällä uudella $k$:n arvolla.*

1. *Esilaskenta.*
   1.1 $x_0 = 1$, $x_1 = x$ *ja* $x_2 = x^2 \pmod{n}$.
   1.2 *$i$ käy yhdestä lukuun* $(2^{3-1}-1) = 3$.
   $i = 1 : x_3 = x_1 \cdot x_2 = x^3$
   $i = 2 : x_5 = x_3 \cdot x_2 = x^5$

$i = 3 : x_7 = x_5 \cdot x_2 = x^7$

2. $A = 1$.

3. *Merkintätavasta* $e_i = 2^{h_i} u_i$ *saadaan* $e_2 = 3 = 2^0 \cdot 3$, $e_1 = 7 = 2^0 \cdot 7$ *ja* $e_0 = 2 = 2^1 \cdot 1$, *joten* $h_2 = 0$, $u_2 = 3$, $h_1 = 0$, $u_1 = 7$, $h_0 = 1$ *ja* $u_1 = 1$.
*Nyt* $i$ *käy luvusta* 2 *nollaan:*

$i = 2 : A = (1^{2^3} \cdot x^3)^{2^0} = x^3$

$i = 1 : A = ((x^3)^{2^3} \cdot x^7)^{2^0} = x^{31}$

$i = 0 : A = ((x^{31})^{2^2} \cdot x)^2 = x^{250}$.

*Luvun* $k$ *ollessa suurempi esilaskennan määrä kasvaa, mutta kohdassa* 3 *suoritettavien laskutoimitusten määrä vähenee.*

### 3.1.2 Potenssiin korotus ikkunamenetelmällä

Kuten edellinen algoritmi, niin myös seuraava algoritmi vähentää tarvittavan esilaskennan määrää verrattuna algoritmiin 3.3. Lisäksi tämä algoritmi vähentää suoritettavien kertolaskujen määrää. Luvun $k$ sanotaan olevan *ikkunan koko*.

Algoritmi saa syötteenä luvut $x, n$, ikkunan koon $k$ ja luvun $e$ binäärisessä muodossa. Esilaskennassa lasketaan luvut $x_1 = x$ ja $x_2 = x^2 \pmod n$ sekä $x_{2i+1} = x_{2i-1} \cdot x_2 \pmod n$, kun $i$ käy läpi luvut yhdestä lukuun $(2^{k-1} - 1)$. Eli esilaskettavat luvut tässäkin ovat $x_3 = x^3, x_5 = x^5, \ldots, x_{2^k-1} = x^{2^k-1}$. Muuttujaan $A$ talletetaan luku 1 ja $i = t$. Kohtaa 3 kierretään niin kauan kuin $i \geq 0$. Joka kierroksella tehdään seuraavaa: jos $e_i = 0$, niin silloin neliöidään muuttujan $A$ arvo ja vähennetään $i$:n arvoa yhdellä. Muuten, siis jos $e_i = 1$, etsitään pisin korkeintaan $k$ pituinen bittijono $e_i e_{i-1} \cdots e_l$, jossa viimeinen bitti on 1. Lasketaan $A = A^{2^{i-l+1}} \cdot x_{(e_i e_{i-1} \cdots e_l)_2} \pmod n$. Luku $x_{(e_i e_{i-1} \cdots e_l)_2}$ on jokin esilasketuista luvuista. Tämän jälkeen luku $i = l - 1$. Lopulta suoritettujen kierrosten jälkeen muuttujaan $A$ on tallentunut potenssiin korotuksen $x^e \pmod n$ tulos.

---

**Algoritmi 3.5.** Potenssiin korotus ikkunamenetelmällä
SYÖTE: $x$, $n$ ja $e = (e_t e_{t-1} \cdots e_1 e_0)_2$, jossa $e_t = 1$ ja kokonaisluku $k \geq 1$.

TULOSTE: $x^e \pmod n$.

1. Esilaskenta.
   - 1.1 $x_1 \leftarrow x$, $x_2 \leftarrow x^2 \pmod n$.
   - 1.2 Kun $i$ käy luvusta 1 lukuun $(2^{k-1} - 1)$, niin
     $x_{2i+1} \leftarrow x_{2i-1} \cdot x_2 \pmod n$.
2. $A \leftarrow 1$, $i \leftarrow t$.
3. Niin kauan kuin $i \geq 0$, niin
   - 3.1 Jos $e_i = 0$, niin $A \leftarrow A^2 \pmod n$, $i \leftarrow i - 1$.
   - 3.2 Muuten $(e_i \neq 0)$ etsi pisin sellainen bittijono $e_i e_{i-1} \cdots e_l$,
     että $i - l + 1 \leq k$ ja $e_l = 1$.
     $A \leftarrow A^{2^{i-l+1}} \cdot x_{(e_i e_{i-1} \cdots e_l)_2} \pmod n$, $i \leftarrow l - 1$ .
4. Palauta $(A)$.

---

**Esimerkki 9.** *Lasketaan algoritmia 3.5 käyttäen potenssiin korotus* $x^{11749}$, *jossa* $e = 11749 = (10110111100101)_2$ *ja* $k = 3$.

*Esilaskennasta saadaan*
1.1*:* $x_1 = x$ *ja* $x_2 = x^2 \pmod n$.
1.2*:*

$i = 1 : x_3 = x_1 \cdot x_2 = x^3 \pmod n$
$i = 2 : x_5 = x_3 \cdot x_2 = x^5 \pmod n$
$i = 3 : x_7 = x_5 \cdot x_2 = x^7 \pmod n$.

*Nyt* $A = 1$ *ja* $i = t = 13$. *Taulukossa 8 on esitetty kohdan* 3 *laskuvaiheet.*

*Neljäs kohta palauttaa muuttujaan A talletetun potenssiin korotuksen tuloksen* $x^{11749} \pmod n$.

*Tällä menetelmällä tarvitaan kolme kertolaskua silloin, kun i saa arvot* $10, 7$ *ja* 2. *Algoritmilla 3.3 tarvittavien kertolaskujen määrä oli neljä, kun käytettiin samoja arvoja luvuille k ja e. Tämä esitettiin esimerkissä 7.*

| $i$ | Pisin bittijono | $A$ |
|---|---|---|
| 13 | 101 | $x^5$ |
| 10 | 101 | $(x^5)^{2^3} \cdot x^5 = x^{45}$ |
| 7 | 111 | $(x^{45})^{2^3} \cdot x^7 = x^{367}$ |
| 4 | - | $(x^{367})^2 = x^{734}$ |
| 3 | - | $(x^{734})^2 = x^{1468}$ |
| 2 | 101 | $(x^{1468})^{2^3} \cdot x^5 = x^{11749}$ |

Taulukko 8

### 3.1.3 Potenssiinkorotusalgoritmien vertailua

Olkoon luvun $e$ pituus $t+1$. Olkoon luvusta $e$ muodostettujen $k$ bitin pituisten sanojen lukumäärä $l+1$ ($l = \lceil (t+1)/k \rceil - 1 = \lfloor t/k \rfloor$). Taulukossa 9 on yhteenveto tarvittavista kertolaskuista ja neliöinneistä algoritmeissa 3.1, 3.2, 3.3 ja 3.4. Algoritmin 3.5 kertolaskujen ja neliöintien lukumäärän selvittäminen on paljon vaikeampaa, vaikka tämä algoritmi on näistä suositeltavin.

| | Esilaskenta | | | Kertolasku | |
|---|---|---|---|---|---|
| | | | | huonoin | keskiverto |
| Algoritmi | neliöinti | kertolasku | neliöinti | tilanne | tilanne |
| 3.1 | 0 | 0 | $t$ | $t$ | $t/2$ |
| 3.2 | 0 | 0 | $t$ | $t$ | $t/2$ |
| 3.3 | 1 | $2^k - 3$ | $lk$ | $l-1$ | $l(2^k-1)/2^k$ |
| 3.4 | 1 | $2^{k-1} - 1$ | $lk + h_l$ | $l-1$ | $l(2^k-1)/2^k$ |

Taulukko 9

Algoritmissa 3.3 tarvitaan $lk$ neliöintiä. Huomataan, että $lk = \lfloor t/k \rfloor k = t - s$, missä $s$ on pienin mahdollinen $t \pmod k$. Tästä seuraa, että $t - (k-1) \leq lk \leq t$, joten algoritmi 3.3 voi parhaimmillaan säästää $k-1$ neliöintiä verrattuna algoritmeihin 3.1 ja 3.2. Algoritmissa 3.3 luvun $k$ optimaalinen arvo on riippuvainen luvusta $t$.

Algoritmissa 3.4 tarvitaan $lk + h_l$ neliöintiä, jossa $0 \leq h_l \leq s$, missä $s$ on pienin mahdollinen $t \pmod{k}$. Koska $t - (k-1) \leq lk \leq lk + h_l \leq lk + s = t$ tai $t - (k-1) \leq lk + h_l \leq t$, niin tarvittavilla neliöinneillä on samat rajat kuin algoritmilla 3.3.

### 3.1.4 Montgomeryn potenssiin korotus

Alla olevaa algoritmia 3.6 eli Montgomeryn kertolaskua tarvitaan Montgomeryn potenssiin korotuksessa ja tämän vuoksi se on otettu mukaan tähän työhön.

Algoritmi saa syötteenä luvut $n, x$ ja $y$ $b$-kantaisina. Luku $b$ saadaan syötteenä annetusta luvusta $R = b^l$. Syötteenä annetaan myös luku $n' = -n^{-1} \pmod{b}$. Muuttujaan $A$ talletetaan aluksi luku 0. Muuttujaan $A$ talletettavat luvut ovat myös $b$-kantaisessa muodossa, $A = (a_l a_{l-1} \cdots a_1 a_0)_b$. Algoritmin toisessa kohdassa $i$ käy nollasta lukuun $(l-1)$. Jokaisella kierroksella lasketaan $u_i = (a_0 + x_i y_0)n' \pmod{b}$ ja muuttujaan $A$ talletetaan arvo $(A + x_i y + u_i n)/b$. Kolmannessa kohdassa varmistetaan, että $A < n$. Jos $A \geq n$, niin suoritetaan vielä yksi laskutoimitus $A = A - n$. Nyt muuttujaan $A$ on talletettuna Montgomeryn kertolaskun $xyR^{-1} \pmod{n}$ tulos.

---

**Algoritmi 3.6.** Montgomeryn kertolasku
SYÖTE: kokonaisluvut $n = (n_{l-1} \cdots n_1 n_0)_b$, $x = (x_{l-1} \cdots x_1 x_0)_b$, $y = (y_{l-1} \cdots y_1 y_0)_b$, joissa $0 \leq x, y < n$, $R = b^l$, jossa $\text{syt}(n, b) = 1$ ja $n' = -n^{-1} \pmod{b}$.
TULOSTE: $xyR^{-1} \pmod{n}$.

1. $A \leftarrow 0$. (Huom. $A = (a_l a_{l-1} \cdots a_1 a_0)_b$.)
2. Kun $i$ käy nollasta lukuun $(l-1)$, niin
   2.1 $u_i \leftarrow (a_0 + x_i y_0)n' \pmod{b}$.
   2.2 $A \leftarrow (A + x_i y + u_i n)/b$.
3. Jos $A \geq n$, niin $A \leftarrow A - n$.
4. Palauta $(A)$.

---

**Esimerkki 10.** *Alla on esitetty algoritmin 3.6 laskuvaiheet, kun käytetään syötteen arvoja* $n = 72639$, $R = 10^5$, $x = 5792$ *ja* $y = 1229$. *Tässä* $l = 5$, $b = 10$ *ja* $n' = -n^{-1} \pmod{10} = 1$ .

$$
\begin{array}{lll}
A = 0 & & \\
i = 0: & u_0 = & (0 + 2 \cdot 9) = 8 \pmod{10} \\
 & A = & (0 + 2 \cdot 1229 + 8 \cdot 72639)/10 = 58357 \\
i = 1: & u_1 = & (7 + 9 \cdot 9) = 8 \pmod{10} \\
 & A = & (58357 + 9 \cdot 1229 + 8 \cdot 72639)/10 = 65053 \\
i = 2: & u_2 = & (3 + 7 \cdot 9) = 6 \pmod{10} \\
 & A = & (65053 + 7 \cdot 1229 + 6 \cdot 72639)/10 = 50949 \\
i = 3: & u_3 = & (9 + 5 \cdot 9) = 4 \pmod{10} \\
 & A = & (50949 + 5 \cdot 1229 + 4 \cdot 72639)/10 = 34765 \\
i = 4: & u_4 = & (5 + 0 \cdot 9) = 5 \pmod{10} \\
 & A = & (34765 + 0 \cdot 1229 + 5 \cdot 72639)/10 = 39796
\end{array}
$$

*Algoritmi palauttaa Montgomeryn kertolaskun tuloksen* $xyR^{-1} \pmod{n} = 39796$ .

Yllä olevaa algoritmia 3.6 ja algoritmia 3.2 yhdistämällä saadaan algoritmi 3.7 Montgomeryn potenssiin korotus. Syötteen $n'$ määritelmä vaatii, että $\text{syt}(n, R) = 1$. Algoritmissa 3.7 tarvitaan algoritmin 3.6 Montgomeryn kertolaskua. Algoritmi 3.7 käyttää merkintää $\text{Mont}(u, v)$. Tällöin lasketaan kertolasku käyttäen algoritmia 3.6, jossa $\text{Mont}(u, v) = uvR^{-1} \pmod{n}$ ja $0 \leq u, v < n$.

Algoritmi 3.7 saa syötteenä luvun $n$ $b$-kantaisena, luvun $e$ binäärisenä, luvut $R = b^l$, $n'$ ja kokonaisluvun $x$. Ensimmäisessä kohdassa lasketaan $\widetilde{x} = \text{Mont}(x, R^2 \pmod{n})$ käyttäen Montgomeryn kertolaskua ja muuttujaan $A$ talletetaan $R \pmod{n}$. Toisessa kohdassa $i$ saa arvot luvusta $t$ nollaan. Jokaisella kierroksella muuttujaan $A$ ensin talletetaan $\text{Mont}(A, A)$ käyttäen jälleen Montgomeryn kertolaskua. Jos $e_i = 1$, niin muuttuja $A$ saa vielä uuden arvon Montgomeryn kertolaskusta $\text{Mont}(A, \widetilde{x})$. Kun toisen kohdan kierrokset on käyty läpi, muuttuja $A$ saa vielä kerran uuden arvon $A = \text{Mont}(A, 1)$. Nyt muuttujassa $A$ on potenssiin korotuksen tulos, jonka algoritmi palauttaa.

**Algoritmi 3.7.** Montgomeryn potenssiin korotus
SYÖTE: $n = (n_{l-1} \cdots n_0)_b$, $R = b^l$, $n' = -n^{-1} \pmod b$, $e = (e_t \cdots e_0)_2$, jossa $e_t = 1$ ja kokonaisluku $x$, $1 \leq x < n$.
TULOSTE: $x^e \pmod n$.

1. $\widetilde{x} \leftarrow \text{Mont}(x, R^2 \pmod n)$, $A \leftarrow R \pmod n$.
   ($R \pmod n$ ja $R^2 \pmod n$ voidaan antaa syötteinäkin.)
2. Kun $i$ käy $t$:stä nollaan, niin
   2.1 $A \leftarrow \text{Mont}(A, A)$.
   2.2 Jos $e_i = 1$, niin $A \leftarrow \text{Mont}(A, \widetilde{x})$.
3. $A \leftarrow \text{Mont}(A, 1)$ .
4. Palauta $(A)$.

**Esimerkki 11.** *Tutustutaan Montgomeryn potenssiinkorotusalgoritmiin laskemalla* $123^7 (\text{mod } 851)$.

*Olkoon* $R = b^l = 10^3$. *Algoritmi saa syötteenä luvut* $n = 851$, $n' = 9$, $e = (111)_2$ *ja* $x = 123$. *Luku* $n' = 9$, *koska* $nn' = -1 \pmod b$.

*Algoritmin ensimmäisessä kohdassa muuttujaan* $A$ *talletetaan* $R \pmod n = 149$ *ja lasketaan* $\widetilde{x}$ *käyttäen Montgomeryn kertolaskua. Alla on esitetty Montgomeryn kertolaskun laskuvaiheet, kun* $\widetilde{x} = \text{Mont}(x, R^2 \pmod n) = \text{Mont}(123, 75)$.

$$
\begin{array}{lll}
A = 0 & & \\
i = 0: & u_0 = & (0 + 3 \cdot 5) \cdot 9 = 5 \pmod{10} \\
 & A = & (0 + 3 \cdot 75 + 5 \cdot 851)/10 = 448 \\
i = 1: & u_1 = & (8 + 2 \cdot 5) \cdot 9 = 2 \pmod{10} \\
 & A = & (448 + 2 \cdot 75 + 2 \cdot 851)/10 = 230 \\
i = 2: & u_2 = & (0 + 1 \cdot 5) \cdot 9 = 5 \pmod{10} \\
 & A = & (230 + 1 \cdot 74 + 5 \cdot 851)/10 = 456
\end{array}
$$

*Tästä saadaan* $\widetilde{x} = 456$ *ja yllä saatiin* $A = 149$.

*Algoritmin toisessa vaiheessa muuttuja* $i$ *käy luvusta* $t = 2$ *nollaan. Montgomeryn kertolaskut on laskettu yllä olevalla tavalla ja tässä on annettuna*

*vain niistä saadut tulokset.*

$i = 2$

2.1 $A = \text{Mont}(149, 149) = 149$

2.2 $e_2 = 1$

$A = \text{Mont}(149, 456) = 456$

$i = 1$

2.1 $A = \text{Mont}(456, 456) = 773$

2.2 $e_1 = 1$

$A = \text{Mont}(773, 456) = 618$

$i = 0$

2.1 $A = \text{Mont}(618, 618) = 787$

2.2 $e_0 = 1$

$A = \text{Mont}(787, 456) = 638$

*Algoritmin kolmannessa vaiheessa lasketaan vielä yksi Montgomeryn kertolasku:* $A = \text{Mont}(638, 1) = 564$*, joka on potenssiin korotuksen tulos.*

## 3.2 Potenssiin korotus, kun eksponentti pysyy vakiona

RSA menetelmässä useiden eri lukujen eksponentti on sama. Tässä luvussa käsitellään algoritmia, joka kehittää edellisen luvun kerro ja neliöi -algoritmeja vähentämällä tarvittavien kertolaskujen määrää, kun eksponentti pysyy vakiona.

### 3.2.1 Yhteenlaskuketjut

Yhteenlaskuketjujen tarkoitus on minimoida kertolaskujen määrää potenssiin korotuksessa.

**Määritelmä 3.8.** Positiivisen kokonaisluvun $e$ *yhteenlaskuketju* $V$ on positiivisten kokonaislukujen jono $u_0, u_1, \ldots, u_s$. Yhteenlaskuketjuun liittyy myös jono $w_1, \ldots, w_s$ pareja $w_i = (i_1, i_2)$, $0 \leq i_1, i_2 < i$. Yhteenlaskuketjuilla on seuraavat ominaisuudet

(i) $u_0 = 1$ ja $u_s = e$ ja

(ii) $u_i = u_{i_1} + u_{i_2}$ kaikilla $u_i$, $1 \leq i \leq s$.

Yhteenlaskuketjun pituus on $s$.

**Esimerkki 12.** *Luvun* 31 *yhteenlaskuketju on* $V = (1, 2, 3, 6, 12, 24, 30, 31)$ *ja ketjuun liittyvä jono pareja on* $(0, 0), (0, 1), (2, 2), (3, 3), (4, 4), (2, 5), (0, 6)$*, koska*

$$1, 1{+}1 = 2, 1{+}2 = 3, 3{+}3 = 6, 6{+}6 = 12, 12{+}12 = 24, 6{+}24 = 30, 30{+}1 = 31.$$

*Tämän yhteenlaskuketjun pituus on* 7*.*

Algoritmi 3.9 saa syötteenä luvut $x$ ja $n$ sekä luvun $e$ yhteenlaskuketjun $V$ ja siihen liittyvän jonon pareja $w_i$. Ensin muuttujaan $x_0$ talletetaan luku $x$. Tämän jälkeen $i$ käy yhdestä lukuun $s$ tallentaen joka kierroksella uuteen muuttujaan $x_i$ tulon $x_{i_1} \cdot x_{i_2} \pmod n$. Luvut $i_1$ ja $i_2$ saadaan kullakin kierroksella yhteenlaskuketjuun liittyvästä jonosta $w_i$. Algoritmi palauttaa viimeisen muuttujan $x_s$, jossa on potenssiin korotuksen $x^e \pmod n$ tulos.

---

**Algoritmi 3.9.** Potenssiin korotus yhteenlaskumenetelmällä
SYÖTE: $x, n$, positiivisen kokonaisluvun $e$ yhteenlaskuketju $V = (u_0, u_1, \ldots, u_s)$ ja siihen liittyvä jono $w_1, \ldots, w_s$, jossa $w_i = (i_1, i_2)$.
TULOSTE: $x^e \pmod n$.

1. $x_0 \leftarrow x$.
2. Kun $i$ käy yhdestä lukuun $s$, niin $x_i \leftarrow x_{i_1} \cdot x_{i_2} \pmod n$.
3. Palauta $x_s$.

---

**Esimerkki 13.** *Lasketaan* $17^{11} \pmod{91}$ *käyttäen algoritmia 3.9. Tässä* $x = 17$ *ja* $e = 11$*. Luvun* 11 *yhteenlaskuketju on* $(1, 2, 3, 6, 9, 11)$ *ja siihen liittyvä jono pareja on* $(0, 0), (0, 1), (2, 2), (2, 3), (1, 4)$*. Taulukossa 10 on algoritmin laskuvaiheet luvuilla* $x$ *ja* $x = 17$*.*

Kun positiiviselle kokonaisluvulle $e$ on annettu yhteenlaskuketju $V$, niin algoritmilla 3.9 laskettaessa laskun $x^e$, $\forall x < n, x \neq 1$ laskemiseen tarvitaan tasan $s$ kertolaskua.

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $w_i$ | – | $(0,0)$ | $(0,1)$ | $(2,2)$ | $(2,3)$ | $(1,4)$ |
| $x_i$ | $x$ | $x_0 \cdot x_0 = x^2$ | $x_0 \cdot x_1 = x^3$ | $x_2 \cdot x_2 = x^6$ | $x_2 \cdot x_3 = x^9$ | $x_1 \cdot x_4 = x^{11}$ |
| $x_i$ | 17 | 16 | 90 | 1 | 90 | 75 |

Taulukko 10

Jos eksponentti $e$ on annettu binäärisessä muodossa, on suhteellisen helppoa muodostaa yhteenlaskuketju. Tosin tällä tavoin muodostettu yhteenlaskuketju ei yleensä ole lyhyin mahdollinen.

Olkoon $l$ lyhyin mahdollinen pituus positiivisen kokonaisluvun $e$ yhteenlaskuketjulle. Silloin $l \geq (log_2\ e + log_2\ wt(e) - 2, 13)$, jossa $wt(e)$ on ykkösten lukumäärä luvun $e$ binääriesityksessä. Yläraja $(\lfloor log_2\ e \rfloor + wt(e) - 1)$ on saatu muodostamalla luvun $e$ yhteenlaskuketju luvun binäärisestä muodosta.

## 3.3 Potenssiin korotus, kun kanta pysyy vakiona

Seuraavat algoritmit esittävät potenssiinkorotusmenetelmiä tilanteisiin, joissa kanta $x$ pysyy samana ja eksponentti $e$ vaihtuu. Kun kanta pysyy vakiona, niin esilaskenta voidaan suorittaa vain kerran useampaa potenssiin korotusta varten. Nämä menetelmät ovat käytännöllisempiä esimerkisi Diffie-Hellman -avaimenvaihtoprotokollassa kuin RSA:ssa.

Jokaisessa tämän luvun algoritmissa kuvattu joukko $\{b_0, b_1, \ldots, b_t\}$ on sellainen joukko kokonaislukuja, että eksponentti $e$ voidaan kirjoittaa muodossa $e = \sum_{i=0}^{t} e_i b_i$, missä $0 \leq e_i < h$ ja $h$ on positiivinen kokonaisluku. Esimerkiksi, jos $e$ on jokin $(t+1)$-numeroinen $b$-kantainen kokonaisluku, jossa $b \geq 2$, niin $b_i = b^i$ ja $h = b$ ovat mahdollisia valintoja.

Algoritmit 3.10 ja 3.11 vaativat potenssien $x^{b_0}, x^{b_1}, \ldots, x^{b_t}$ laskemista etukäteen. Näiden laskemiseen käytetään esimerkiksi jotain luvussa 3.1 esitetyistä algoritmeista. Algoritmin 3.13 esilaskenta on monimutkaisempi ja on esitetty algoritmissa 3.12.

### 3.3.1 Ikkunointimenetelmä

Algoritmin 3.10 syötteenä ovat esilasketut potenssit $x_i = x^{b_i}$, $0 \leq i \leq t$ ja positiiviset kokonaisluvut $h$ ja $e = \sum_{i=0}^{t} e_i b_i$, missä $0 \leq e_i < h$, $0 \leq i \leq t$. Algoritmin perustana on havainto, että $x^e = \prod_{i=0}^{t} x_i^{e_i} = \prod_{j=1}^{h-1} (\prod_{e_i=j} x_i)^j$.

---

**Algoritmi 3.10.** Potenssiin korotus ikkunointimenetelmällä
SYÖTE: $\{x^{b_0}, x^{b_1}, \ldots, x^{b_t}\}$, $e = \sum_{i=0}^{t} e_i b_i$, $h$ ja $n$.
TULOSTE: $x^e \pmod n$.

1. $A \leftarrow 1$, $B \leftarrow 1$.
2. Kun $j$ käy luvusta $(h-1)$ yhteen, niin
   2.1 Kaikilla $i$, joilla $e_i = j$, niin $B \leftarrow B \cdot x^{b_i} \pmod n$.
   2.2 $A \leftarrow A \cdot B \pmod n$.
3. Palauta $(A)$.

---

**Esimerkki 14.** *Esilasketaan alkiot* $x^1, x^4, x^{16}, x^{64}, x^{256}$. *Olkoon* $e = 862 = (31132)_4$, *kun lasketaan laskua* $x^e$. *Tässä* $t = 4$, $h = 4$ *ja* $b_i = 4^i$, *kun* $0 \leq i \leq 4$. *Taulukossa 11 on algoritmin 3.10 muutujien A ja B arvot algoritmin toisen vaiheen jokaisen iteraation jälkeen.*

| $j$ | - | 3 | 2 | 1 |
|---|---|---|---|---|
| $B$ | 1 | $x^4 x^{256} = x^{260}$ | $x^{260} x = x^{261}$ | $x^{261} x^{16} x^{64} = x^{341}$ |
| $A$ | 1 | $x^{260}$ | $x^{260} x^{261} = x^{521}$ | $x^{521} x^{341} = x^{862}$ |

Taulukko 11

*Ensimmäisellä kierroksella, kun* $j = 3$, *kohdassa* 2.1 *i:n arvoilla* 1 *ja* 4 *saadaan* $e_i = 3$. *Täten* $B = x^4 x^{256}$. *Kun* $j = 2$, *niin luvussa* $e = (31132)_4$ *vain* $e_0 = 2$, *joten tällöin* $B = x^{260} x$. *Viimeisellä kierroksella muuttuja B kerrotaan luvuilla* $x^{16}$ *ja* $x^{64}$. *Jokaisella kierroksella muuttuja A kerrotaan saadulla B:n arvolla ja näin potenssiin korotuksen tulos kerääntyy muuttujaan A.*

Oletetaan, että $t+h \geq 2$. Ainoastaan ne kertolaskut, joissa tulon molemmat tekijät ovat erisuuria kuin 1, otetaan huomioon. Kohta 2.2 suoritetaan $h-1$ kertaa, mutta ainakin ensimmäisessä kertolaskussa toinen tulon tekijöistä on 1. Koska $B$ on aluksi 1, niin kohdassa 2.1 suoritetaan korkeintaan $t$ kertolaskua. Täten algoritmissa 3.10 laskun $x^e$ suorittamiseen tarvitaan korkeintaan $t+h-2$ kertolaskua. Muistia algorimissa tarvitaan esilaskettujen alkioiden $x_i$ tallentamiseen, joita on $t+1$ kappaletta.

### 3.3.2 Euklidinen menetelmä

Olkoon $\{v_0, v_1, \ldots, v_t\}$ joukko kokonaislukuja, jossa $t \geq 2$. Määritellään $M$ olemaan sellainen kokonaisluku välillä $[0,t]$, että $v_M \geq v_i$ kaikilla $0 \leq i \leq t$. Määritellään $N$ olemaan sellainen kokonaisluku välillä $[0,t]$, että $M \neq N$ ja $v_N \geq v_i$ kaikilla $0 \leq i \leq t$ ja $i \neq M$.

---

**Algoritmi 3.11.** Potenssiin korotus euklidisella menetelmällä
SYÖTE: $\{x^{b_0}, x^{b_1}, \ldots, x^{b_t}\}$, $n$ ja $e = \sum_{i=0}^{t} e_i b_i$.
TULOSTE: $x^e \pmod n$.

1. Kun $i$ käy nollasta lukuun $t$, niin $x_i \leftarrow x^{b_i}$, $v_i \leftarrow e_i$.
2. Määritä joukolle $\{v_0, v_1, \ldots, v_t\}$ kokonaisluvut $M$ ja $N$.
3. Niin kauan kuin $v_N \neq 0$, niin
   3.1 $q \leftarrow \lfloor v_M / v_N \rfloor$, $x_N \leftarrow (x_M)^q \cdot x_N \pmod n$, $v_M \leftarrow v_M \pmod{v_N}$.
   3.2 Määritä joukolle $\{v_0, v_1, \ldots, v_t\}$ kokonaisluvut $M$ ja $N$.
4. Palauta $(x_M^{v_M}) \pmod n$.

---

**Esimerkki 15.** *Lasketaan $x^e$ käyttäen yllä olevaa algoritmia. Olkoon $e = 862$, kuten edellisessä esimerkissä. Valitaan $b_0 = 1, b_1 = 16, b_2 = 256$. Tästä saadaan $e = (3,5,14)_{16}$. Esilasketaan $x^1, x^{16}$ ja $x^{256}$. Taulukossa 12 on esitetty algoritmilla 3.11 saadut tulokset jokaisen kierroksen jälkeen.*

*Ensimmäisellä rivillä on arvot, jotka on saatu algoritmin kohdista 1 ja 2. Lukujen $v_i$ arvot on saatu luvusta $e = (3,5,14)_{16}$ ja luvut $x_i$ ovat esilasketut syötteenä annetut luvut $x^{b_i}$. Joukon $\{v_0, v_1, v_2\}$ luvuille määritellään luvut $M$*

| $v_0$ | $v_1$ | $v_2$ | $M$ | $N$ | $q$ | $x_0$ | $x_1$ | $x_2$ |
|---|---|---|---|---|---|---|---|---|
| 14 | 5 | 3 | 0 | 1 | | $x$ | $x^{16}$ | $x^{256}$ |
| 4 | 5 | 3 | 1 | 0 | 2 | $x$ | $x^{18}$ | $x^{256}$ |
| 4 | 1 | 3 | 0 | 2 | 1 | $x^{19}$ | $x^{18}$ | $x^{256}$ |
| 1 | 1 | 3 | 2 | 1 | 1 | $x^{19}$ | $x^{18}$ | $x^{275}$ |
| 1 | 1 | 0 | 0 | 1 | 3 | $x^{19}$ | $x^{843}$ | $x^{275}$ |
| 0 | 1 | 0 | 1 | 0 | 1 | $x^{19}$ | $x^{862}$ | $x^{275}$ |

Taulukko 12

*ja $N$. Tässä $M = 0$, koska $v_0 = 14$ ja täten on suurin luvuista $v_i$. Koska $M \neq N$, niin $N = 1$. Algoritmin kolmatta kohtaa käydään läpi niin kauan kuin $v_N \neq 0$. Toisella rivillä on ensin määritelty algoritmin kolmannen kohdan mukaan $q = \lfloor v_M / v_N \rfloor = \lfloor 14/5 \rfloor = 2$. Tämän arvon avulla on laskettu uusi $x_N = x_1 = (x_M)^q \cdot x_N = (x_0)^2 \cdot x_1 = x^2 \cdot x^{16} = x^{18}$ ja $v_M = v_M \pmod{v_N} = 14 = 4 \pmod 5$. Tämän jälkeen määritellään uudet $M$ ja $N$, jotka ovat nyt $M = 1$ ja $N = 0$. Kun on tarkastettu, että $v_N = v_0 = 4 \neq 0$, niin voidaan siirtyä seuraavalle kierrokselle ja käydä kohta 3 uudelleen läpi. Vastaavalla tavalla käydään seuraavat rivit, kunnes viimeiseltä riviltä huomataan, että $v_N = v_0 = 0$. Kierrosta ei enää jatketa, vaan siirrytään algoritmin neljänteen kohtaan ja palautetaan potenssiin korotuksen tulos $x_M^{v_M} = x_1^{v_1} = x^{862}$.*

### 3.3.3 Kampamenetelmä

Algoritmi 3.13 laskee $x^e \pmod n$, jossa $e = (e_t e_{t-1} \cdots e_1 e_0)_2$ ja $t \geq 1$. Valitaan sellainen kokonaisluku $h$, että $1 \leq h \leq t+1$, ja lasketaan $a = \lceil (t+1)/h \rceil$. Valitaan sellainen kokonaisluku $v$, että $1 \leq v \leq a$, ja lasketaan $b = \lceil a/v \rceil$. Selvästi $ah \geq t+1$. Olkoon $X = R_{h-1}||R_{h-2}|| \cdots ||R_0$ luvusta $e$ muodostettu bittijono, johon on tarvittaessa lisätty vasemmalle puolelle nollia, jotta bittijonon X pituus on $ah$ ja jokaisen $R_i$ bittijonon pituus on $a$. Muodostetaan $h \times a$ taulukko $EA$ (exponent array), jossa taulukon rivillä $i$ on bittijono $R_i$, kun $0 \leq i \leq h-1$.

**Esimerkki 16.** *Muodostetaan taulukko $EA$ luvusta $e = 11749 =$*

$(10110111100101)_2$. *Valitaan* $h = 5$ *ja lasketaan* $a = \lceil (t+1)/h \rceil = \lceil 14/5 \rceil = 3$. *Muodostetaan* $X = R_{h-1}||R_{h-2}||\cdots||R_0$ *luvusta e, lisäämällä yksi nolla binääriesityksen vasemmalle puolelle, jotta pituudeksi saadaan* $ah = 15$. *Saadaan* $X = R_4||R_3||R_2||R_1||R_0 = 010||110||111||100||101$. *Alla olevassa taulukossa 13 on saaduista arvoista muodostettu taulukko* $EA$, *jossa rivillä* $i$ *on* $R_i$.

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 0 |

Taulukko 13

Algoritmissa 3.13 tarvitaan lukua $I_{j,k}$. Kun on valittu sellainen kokonaisluku $v$, että $1 \leq v \leq a$, ja siitä on saatu $b = \lceil a/v \rceil$, niin $0 \leq k \leq b$ ja $0 \leq j < v$. Olkoon $I_{j,k}$ sellainen kokonaisluku, jonka binääriesitys on taulukon $EA$ sarake $(jb + k)$. Taulukossa $EA$ sarake 0 on oikealla ja sarakkeen vähiten merkitsevät bitit ovat ylimmäisinä.

Algoritmissa 3.13 tarvittavan esilaskennan suorittaa algoritmi 3.12. Algoritmi 3.12 saa syötteenä luvun $x$ ja edellä määritellyt parametrit $h, v, a$ ja $b$. Tulosteena saadaan taulukko, jossa on esilaskettu tietyt tarvittavat potenssiin korotukset. Algoritmin alla olevassa esimerkissä on algoritmin toimintaa esitetty tarkemmin.

---

**Algoritmi 3.12.** Esilaskenta algoritmille 3.13
SYÖTE: $x$, $n$ ja yllä määritellyt parametrit $h$, $v$, $a$ ja $b$.
TULOSTE: $\{G[j][i] : 1 \leq i < 2^h, 0 \leq j < v\}$.

1. Kun $i$ käy nollasta lukuun $(h-1)$, niin $x_i \leftarrow x^{2^{ia}} \pmod{n}$.
2. Kun $i$ käy yhdestä lukuun $(2^h-1)$, missä $i = (i_{h-1}\cdots i_0)_2$, niin
   2.1 $G[0][i] \leftarrow \prod_{j=0}^{h-1} x_j^{i_j} \pmod{n}$.
   2.2 Kun $j$ käy yhdestä lukuun $(v-1)$, niin
   $G[j][i] \leftarrow (G[0][i])^{2^{jb}} \pmod{n}$.
3. Palauta($\{G[j][i] : 1 \leq i < 2^h, 0 \leq j < v\}$) .

---

**Esimerkki 17.** *Olkoon syötteenä annettu $x$ ja olkoon $t = 9$. Valitaan $h = 3$, josta saadaan $a = 4$. Valitaan vielä $v = 2$, josta saadaan $b = 2$.*

*Algoritmin ensimmäisessä kohdassa $i$ käy nollasta lukuun $(h-1) = 2$ ja laskee $x_i = x^{2^{ia}}$. Saadaan $x_0 = x$, $x_1 = x^{16}$ ja $x_2 = x^{256}$. Alla olevassa taulukossa on algoritmin toisen kohdan tulokset, kun $i$ käy yhdestä lukuun $(2^h-1) = 7$. Ensimmäisellä rivillä on kohdan 2.1 tulokset. Kohdasta 2.2 tulee tässä esimerkissä vain yksi rivi, koska $(v-1) = 1$. Tämä rivi on taulukon toinen rivi. Luvun $v$ ollessa suurempi, taulukkoon tulisi useampi rivi.*

*Ensimmäisellä kierroksella, kun $i = 1 = (001)_2$, niin $G[0][1] = \prod_{j=0}^{2} x_j^{i_j} = x_0^1 \cdot x_1^0 \cdot x_2^0 = x_0 = x$. Seuraavaksi on laskettu kohta $G[1][1] = (G[0][1])^{2^{jb}} = (x_0)^{2^2} = x_0^4 = x^4$. Vastaavalla tavalla on laskettu algoritmin kohdan 2 loput kierrokset. Algoritmi palauttaa taulukossa 14 olevat arvot.*

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $G[0][i]$ | $x$ | $x^{16}$ | $x^{17}$ | $x^{256}$ | $x^{257}$ | $x^{272}$ | $x^{273}$ |
| $G[1][i]$ | $x^4$ | $x^{64}$ | $x^{68}$ | $x^{1024}$ | $x^{1028}$ | $x^{1088}$ | $x^{1092}$ |

Taulukko 14

---

**Algoritmi 3.13.** Potenssiin korotus kampamenetelmällä
SYÖTE: $x$, $e$, $n$ ja algoritmissa 3.12 saadut $v$ ja $b$ sekä laskettu $\{G[j][i] : 1 \leq i < 2^h, 0 \leq j < v\}$.

TULOSTE: $x^e \pmod n$.

1. $A \leftarrow 1$.
2. Kun $k$ käy luvusta $(b-1)$ nollaan, niin
   2.1 $A \leftarrow A \cdot A \pmod n$.
   2.2 Kun $j$ käy luvusta $(v-1)$ nollaan, niin
   $A \leftarrow G[j][I_{j,k}] \cdot A \pmod n$.
3. Palauta $(A)$.

---

**Esimerkki 18.** *Lasketaan* $x^{729} \pmod n$ *käyttäen algoritmia 3.13. Eksponentti* $e = 729 = (1011011001)_2$, *joten* $t = 9$. *Valitaan* $h = 3$, *josta saadaan* $a = 4$ *ja valitaan* $v = 2$, *josta saadaan* $b = 2$. *Saadut luvut ovat samat kuin edellisessä esimerkissä, joten voimme käyttää esimerkissä 17 saatua taulukkoa esilaskennan tuloksena:*

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $G[0][i]$ | $x$ | $x^{16}$ | $x^{17}$ | $x^{256}$ | $x^{257}$ | $x^{272}$ | $x^{273}$ |
| $G[1][i]$ | $x^4$ | $x^{64}$ | $x^{68}$ | $x^{1024}$ | $x^{1028}$ | $x^{1088}$ | $x^{1092}$ |

*Algoritmia varten tarvitsemme vielä taulukon EA, joka on muodostettu luvusta e. Taulukko 15 on taulukko EA.*

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |

Taulukko 15

*Yllä annetuilla arvoilla pystymme suorittamaan algoritmin 3.13. Ensin muuttujaan A talletetaan luku* 1. *Taulukossa 16 on esitettynä algoritmin kohdassa* 2 *muuttujaan A tallettuvat tulokset. Muistetaan, että* $I_{j,k}$ *on kokonaisluku, jonka binääriesitys on taulukossa EA sarake* $(jb + k) = (j \cdot 2 + k)$.

*Nyt algoritmi palauttaa muuttujan* $A = x^{729}$, *joka on potenssiin korotuksen tulos.*

| $k$ | $j$ | $A$ |
|---|---|---|
| 1 | | $1$ |
| | 1 | $G[1][I_{1,1}] \cdot 1 = G[1][3] = x^{68}$ |
| | 0 | $G[0][4] \cdot x^{68} = x^{256} \cdot x^{68} = x^{324}$ |
| 0 | | $x^{324} \cdot x^{324} = x^{648}$ |
| | 1 | $G[1][2] \cdot x^{648} = x^{712}$ |
| | 0 | $G[0][3] \cdot x^{712} = x^{729}$ |

Taulukko 16

Algoritmi 3.13 vaatii korkeintaan yhden kertolaskun jokaista taulukon $EA$ saraketta kohti. Oikeanpuoleisin sarake kerrotaan muuttujan $A$ alkuarvolla 1. Algoritmissa myös neliöidään muuttuja $A$ jokaisella luvulla $k$, kun $0 \leq k \leq b-1$, paitsi silloin kun $k = b-1$. Tällöin muuttujaan $A$ on tallennettu arvo 1. Kun kertolaskujen määrästä vähennetään 1, niin epätriviaaleja kertolaskuja tarvitaan korkeintaan $a + b - 2$.

Algoritmi tarvitsee tallennustilaa $v(2^h - 1)$:lle esilasketulle alkiolle. Jos neliöinti on merkittävästi helpompaa verrattuna alkioiden kertomiseen, niin jonkin verran tilaa voidaan säästää tallentamalla vain $2^h - 1$ alkiota (esimerkiksi ne, jotka on laskettu algoritmin 3.12 vaiheessa 2.1).

Lukujen $h$ ja $v$ valinta voidaan tehdä perustuen tarvittavaan muistitilaan tai käytettävissä olevaan aikaan.

# 4 Potenssiinkorotusalgoritmit RSA:ssa

Luodaan RSA salasanat ja salataan viesti $x = 280565$ käyttäen vuorollaan jokaista edellisessä luvussa esitettyä potenssiinkorotusalgoritmia. Tästä näkee, kuinka paljon työtä saman potenssiin korotuksen laskeminen kullakin algoritmilla vaatii. Tässä ei enää selitetä algoritmien toimintatapaa, vaan lähinnä laskutoimitukset ovat esillä.

Valitaan alkuluvut $p$ ja $q$ suhteellisen suuriksi, mutta sellaisiksi, joita vielä pystytään käsittelemään hyvin tässä työssä. Todellisuudessa lukujen on oltava vieläkin paljon suurempia. Olkoon $p = 863$ ja $q = 587$. Näistä saadaan $n = 506581$. Valitaan luku $e = 4381$ ja tarkastetaan, että $\text{syt}(e, \varphi(n)) = 1$. Määritellään myös salainen avain $d = 243169$ siten, että $ed = 1 \pmod{\varphi(n)}$. Muistetaan, että $\varphi(n) = (p-1)(q-1)$.

Lasketaan potenssiin korotus $280565^{4381} \pmod{506581}$ käyttäen edellisessä luvussa esitettyjä menetelmiä.

## 4.1 Binäärinen oikealta vasemmalle etenevä potenssiin korotus

Algoritmi 3.1 saa syötteenä luvut $x = 280565$, $n = 506581$ ja $e = 4381$. Algoritmin ensimmäisessä kohdassa muuttujat $A$, $S$ ja $e$ saavat arvot $A = 1$, $S = 280565$ ja $e = 4381$. Taulukossa 17 on esitettynä algoritmin toisessa kohdassa muuttujien $A$, $S$ ja $e$ saamat arvot.

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.2 Binäärinen vasemmalta oikealle etenevä potenssiin korotus

Algoritmi 3.2 saa syötteenä luvut $x = 280565$ ja $n = 506581$ sekä eksponentin binäärisessä muodossa $e = (1000100011101)_2$. Nyt muuttujia on vain yksi, johon kertyy potenssiin korotuksen tulos. Taulukossa 18 on algoritmin suorittamien laskujen tulokset.

| $A = A \cdot S$ | $e = \lfloor e/2 \rfloor$ | $S = S \cdot S$ |
|---|---|---|
| 280565 | 2190 | 110797 |
| | 1095 | 504417 |
| 247959 | 547 | 123667 |
| 491142 | 273 | 235080 |
| 116021 | 136 | 461529 |
| | 68 | 319218 |
| | 34 | 350212 |
| | 17 | 119034 |
| 32492 | 8 | 22586 |
| | 4 | 329 |
| | 2 | 108241 |
| | 1 | 415294 |
| 441132 | 0 | |

Taulukko 17

| $i$ | $A = A \cdot A$ | $e_i$ | $A = A \cdot x$ |
|---|---|---|---|
| 12 | 1 | $e_{12} = 1$ | 280565 |
| 11 | 110797 | $e_{11} = 0$ | |
| 10 | 504417 | $e_{10} = 0$ | |
| 9 | 123667 | $e_9 = 0$ | |
| 8 | 353080 | $e_8 = 1$ | 482231 |
| 7 | 222730 | $e_7 = 0$ | |
| 6 | 188732 | $e_6 = 0$ | |
| 5 | 31390 | $e_5 = 0$ | |
| 4 | 32055 | $e_4 = 1$ | 178582 |
| 3 | 230450 | $e_3 = 1$ | 258058 |
| 2 | 312847 | $e_2 = 1$ | 148428 |
| 1 | 170075 | $e_1 = 0$ | |
| 0 | 237106 | $e_0 = 1$ | 441132 |

Taulukko 18

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.3 $2^k$-kantainen vasemmalta oikealle etenevä potenssiin korotus

Algoritmin 3.3 syötteenä ovat luvut $x = 280565$ ja $n = 506581$. Valitaan $k = 2$, josta saadaan $b = 2^2 = 4$. Eksponentti $e = (1010131)_4$. Alla on algoritmin suorittamat laskut.

1. Esilaskenta
   $x_0 = 1$
   $x_1 = x_0 \cdot x = 280565$
   $x_2 = x_1 \cdot x = 110797$
   $x_3 = x_2 \cdot x = 430402$
2. $A = 1$
3.

| $i$ | $A = A^4$ | $A = A \cdot x_{e_i}$ |
|---|---|---|
| 6 | 1 | 280565 |
| 5 | 504417 | 504417 |
| 4 | 353080 | 482231 |
| 3 | 188732 | 188732 |
| 2 | 32055 | 178582 |
| 1 | 289946 | 148428 |
| 0 | 237106 | 441132 |

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.4 Muokattu $2^k$-kantainen vasemmalta oikealle etenevä potenssiin korotus

Algoritmi 3.4 saa syötteenä luvut $x = 280565$ ja $n = 506581$. Valitaan $k = 2$, josta saadaan $b = 2^2 = 4$. Eksponentti $e = (1010131)_4$. Al-

goritmia varten tarvitaan luvut $h_i$ ja $u_i$, jotka saadaan määritelmästä $e_i = 2^{h_i}u_i$. Nyt $e_6 = e_4 = e_2 = e_0 = 1$, joten $h_6 = h_4 = h_2 = h_0 = 0$ ja $u_6 = u_4 = u_2 = u_0 = 1$. Luvut $e_5 = e_3 = 0$, joten $h_5 = h_3 = 0$, $u_5 = u_3 = 0$ ja luku $e_1 = 3$, joten $h_1 = 0$ ja $u_1 = 3$. Algoritmin muuttuja $A = (A^{2^{k-h_i}} \cdot x_{u_i})^{2^{h_i}} = A^4 \cdot x_{u_i}$, koska $k = 2$ ja $h_i = 0$ kaikilla $i$. Alla on esitettynä algoritmissa suoritetut laskut.

1. Esilaskenta
   $x_0 = 1$
   $x_1 = 280565$
   $x_2 = 110797$
   $x_3 = x_1 \cdot x_2 = 430402$
2. $A = 1$
3.

| $i$ | $u_i$ | $A = A^4 \cdot x_{u_i}$ |
|---|---|---|
| 6 | 1 | 280565 |
| 5 | 0 | $280565^4 \cdot 1 = 504417$ |
| 4 | 1 | $504417^4 \cdot 280565 = 482231$ |
| 3 | 0 | $482231^4 \cdot 1 = 188732$ |
| 2 | 1 | $188732^4 \cdot 280565 = 178582$ |
| 1 | 3 | $178582^4 \cdot 430402 = 148428$ |
| 0 | 1 | $148428^4 \cdot 280565 = 441132$ |

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.5 Potenssiin korotus ikkunamenetelmällä

Algoritmi 3.5 saa syötteenä luvut $e = (1000100011101)_2$, $x = 280565$ ja $n = 506581$. Valitaan ikkunan koko $k = 3$.

1. Esilaskenta
   $x_1 = 280565$
   $x_2 = x^2 = 110797$

$x_3 = x_1 \cdot x_2 = 430402$

$x_5 = x_3 \cdot x_2 = 247959$

$x_7 = x_5 \cdot x_2 = 212531$

2. $A = 1$ ja $i = 12$

3.

| $i$ | Pisin bittijono | $A$ |
|---|---|---|
| 12 | 1 | $1 \cdot x_1 = 280565$ |
| 11 | – | $280565^2 = 110797$ |
| 10 | – | $110797^2 = 504417$ |
| 9 | – | $504417^2 = 123667$ |
| 8 | 1 | $123667^2 \cdot x_1 = 482231$ |
| 7 | – | $482231^2 = 222730$ |
| 6 | – | $222730^2 = 188732$ |
| 5 | – | $188732^2 = 31390$ |
| 4 | 111 | $31390^8 \cdot x_7 = 148428$ |
| 1 | – | $148428^2 = 170075$ |
| 0 | 1 | $170075^2 \cdot x_1 = 441132$ |

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.6 Montgomeryn potenssiin korotus

Algoritmi 3.7 saa syötteenä luvut $n = 506581$, $R = 10^6$, $e = (1000100011101)_2$, $x = 280565$ ja $n' = 9$.

Ensin ratkaistaan $\widetilde{x} = \text{Mont}(x, R^2 (\text{mod } n)) = \text{Mont}(280565, 494123)$ käyttäen Montgomeryn kertolaskua.

$A = 0$

$i = 0:$ $u_0 = (0 + 5 \cdot 3) \cdot 9 = 5 \pmod{10}$

$A = (0 + 5 \cdot 494123 + 5 \cdot 506581)/10 = 500352$

$i = 1:$ $u_1 = (2 + 6 \cdot 3) \cdot 9 = 0 \pmod{10}$

$A = (500352 + 6 \cdot 494123 + 0)/10 = 346509$

$i = 2:$ $u_2 = (9 + 5 \cdot 3) \cdot 9 = 6 \pmod{10}$

$A = (346509 + 5 \cdot 494123 + 6 \cdot 506581)/10 = 585661$

$$
\begin{aligned}
i = 3: \quad u_3 &= (1 + 0 \cdot 3) \cdot 9 = 9 \pmod{10} \\
A &= (585661 + 0 + 9 \cdot 506581)/10 = 514489 \\
i = 4: \quad u_4 &= (9 + 8 \cdot 3) \cdot 9 = 7 \pmod{10} \\
A &= (514489 + 8 \cdot 494123 + 7 \cdot 506581)/10 = 801354 \\
i = 5: \quad u_5 &= (4 + 2 \cdot 3) \cdot 9 = 0 \pmod{10} \\
A &= (801354 + 2 \cdot 494123 + 0)/10 = 178960
\end{aligned}
$$

1. $\widetilde{x} = 178960$ ja $A = R(\text{mod } n) = 493419$.

2. Taulukossa 19 on jälleen esitettynä vain Mongomeryn kertolaskujen tulokset. Kukin lasku vaatii työtä suunnilleen saman verran, kuin yllä olevan $\widetilde{x}$:n ratkaisu.

| $i$ | $A = \text{Mont}(A, A)$ | $e_i$ | $A = \text{Mont}(A, \widetilde{x})$ |
|---|---|---|---|
| 12 | 493419 | 1 | 178960 |
| 11 | 136585 | 0 | – |
| 10 | 114032 | 0 | – |
| 9 | 446280 | 0 | – |
| 8 | 135134 | 1 | 335508 |
| 7 | 11987 | 0 | – |
| 6 | 182640 | 0 | – |
| 5 | 214916 | 0 | – |
| 4 | 74063 | 1 | 39556 |
| 3 | 224128 | 1 | 66209 |
| 2 | 304735 | 1 | 273581 |
| 1 | 54289 | 0 | – |
| 0 | 256369 | 1 | 252038 |

Taulukko 19

3. $\text{Mont}(A, 1) = 441132$.

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

## 4.7 Potenssiin korotus yhteenlaskuketjumenetelmällä

Algoritmi 3.9 saa syötteenä luvut $x = 280565$ ja $n = 506581$ sekä luvun $e = 4381$ yhteenlaskuketjun $V = (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 4352, 4368, 4376, 4380, 4381)$ ja siihen liittyvän jonon $w_1, \ldots, w_{17} =$ (0,0), (1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (7,7), (8,8), (9,9), (10,10), (11,11), (12,8), (13,4) ,(14,3), (15,2), (16,0). Tämä on vain yksi mahdollisista luvun $e$ yhteenlaskuketjuista. Eri pituisella yhteenlaskuketjulla laskuvaiheita luonnollisesti on eri määrä.

Taulukossa 20 on yhteenlaskuketjun ja siihen liittyvän jonon avulla algoritmin suorittama potenssiin korotus. Tässä $x_0 = 280565$.

| $i$ | $w_i = (i_1, i_2)$ | $x_i = x_{i_1} \cdot x_{i_2} \pmod n$ |
|---|---|---|
| 1 | $(0, 0)$ | 110797 |
| 2 | $(1, 1)$ | 504417 |
| 3 | $(2, 2)$ | 123667 |
| 4 | $(3, 3)$ | 353080 |
| 5 | $(4, 4)$ | 461529 |
| 6 | $(5, 5)$ | 319218 |
| 7 | $(6, 6)$ | 350212 |
| 8 | $(7, 7)$ | 119034 |
| 9 | $(8, 8)$ | 22586 |
| 10 | $(9, 9)$ | 329 |
| 11 | $(10, 10)$ | 10824 |
| 12 | $(11, 11)$ | 415294 |
| 13 | $(12, 8)$ | 412273 |
| 14 | $(13, 4)$ | 313652 |
| 15 | $(14, 3)$ | 1295 |
| 16 | $(15, 2)$ | 237106 |
| 17 | $(16, 0)$ | 441132 |

Taulukko 20

Algoritmi palauttaa luvun $x_{17} = 441132$, joka on potenssiin korotuksen

$280565^{4381} \pmod{506581}$ tulos.

## 4.8 Potenssiin korotus ikkunointimenetelmällä

Algoritmin 3.10 tarvitsemaa esilaskentaa varten $e = (1010131)_4$. Kun eksponentti kirjoitetaan muodossa $e = \sum_{i=0}^{t} e_i b_i$, voidaan valita $b_i = 4^i$ ja $h = 4$ ja saadaan algoritmissa tarvittava joukko $\{b_0, \ldots, b_6\} = \{1, 4, 16, 64, 256, 1024, 4096\}$.

Esilasketaan algoritmin syötteenä tulevat luvut $\{x^{b_o}, \ldots, x^{b_t}\} = \{x^1, x^4, x^{16}, x^{64}, x^{256}, x^{1024}, x^{4096}\}$ käyttämällä jotain luvussa 3.1 esitetyistä potenssiinkorotusalgoritmeista.

$x^1 = 280565$
$x^4 = 504417$
$x^{16} = 353080$
$x^{64} = 319218$
$x^{256} = 119034$
$x^{1024} = 329$
$x^{4096} = 415294$

1. $A = 1$ ja $B = 1$.
2.

| $j$ | $e_i = j$ | $B = B \cdot x^{b_i}$ | $A = A \cdot B$ |
|---|---|---|---|
| 3 | $e_1$ | $1 \cdot x^4 = 504417$ | 504417 |
| 2 | $-$ | | 123667 |
| 1 | $e_0$ | 247959 | |
| | $e_2$ | 8976 | |
| | $e_4$ | 69855 | |
| | $e_6$ | 494824 | 441132 |

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

Esilaskentaa ei tässä esimerkissä ole erikseen laskettu. Käytännössä esilaskenta on työläs ja vaatii paljon laskutoimituksia.

## 4.9 Potenssiin korotus euklidisella menetelmällä

Edellisessä algoritmissa on esilaskentana laskettu tässäkin algoritmissa tarvittavat potenssiin korotukset $\{x^{b_0}, \ldots, x^{b_n}\} = \{280565, 504417, 353080, 319218, 119034, 329, 415294\}$. Taulukossa 21 ovat algoritmin 3.11 suorittamat laskuvaiheet. Ensimmäisessä sarakkeessa ovat algoritmin kohdat 1 ja 2 ja siitä eteenpäin kohdan 3 suorittamat kierrokset.

| | | | | | |
|---|---|---|---|---|---|
| $v_0$ | 1 | 1 | 0 | 0 | 0 |
| $v_1$ | 3 | 0 | 0 | 0 | 0 |
| $v_2$ | 1 | 1 | 1 | 0 | 0 |
| $v_3$ | 0 | 0 | 0 | 0 | 0 |
| $v_4$ | 1 | 1 | 1 | 1 | 0 |
| $v_5$ | 0 | 0 | 0 | 0 | 0 |
| $v_6$ | 1 | 1 | 1 | 1 | 1 |
| $M$ | 1 | 0 | 2 | 4 | 6 |
| $N$ | 0 | 2 | 4 | 6 | 0 |
| $q$ | | 3 | 1 | 1 | 1 |
| $x_0$ | 280565 | 491142 | 491142 | 491142 | 491142 |
| $x_1$ | 504417 | 504417 | 504417 | 504417 | 504417 |
| $x_2$ | 353080 | 353080 | 116021 | 116021 | 116021 |
| $x_3$ | 319218 | 319218 | 319218 | 319218 | 319218 |
| $x_4$ | 119034 | 119034 | 119034 | 32492 | 32492 |
| $x_5$ | 329 | 329 | 329 | 329 | 329 |
| $x_6$ | 415294 | 415294 | 415294 | 415294 | 441132 |

Taulukko 21

Algoritmi palauttaa luvun $x_6^{v_6} = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

Ilman tarvittavaa esilaskentaa tämäkin menetelmä olisi nopea. RSA:ssa joudutaan kuitenkin jokaista salattavaa lohkoa kohden suorittamaan uusi esilaskenta.

## 4.10 Potenssiin korotus kampamenetelmällä

Algoritmi 3.13 saa syötteenä luvut $x = 280565$, $e = 4381 = (1000100011101)_2$ ja $n = 506581$ sekä esilasketun taulukon $G[j][k]$. Valitaan luvut $h = 7$ ja $v = 1$, joista saadaan $a = 2$ ja $b = 2$.

Taulukossa 22 on eksponentista $e$ muodostettu taulukko $EA$.

| | |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 0 | 1 |
| 0 | 0 |
| 0 | 1 |
| 0 | 0 |
| 0 | 1 |

Taulukko 22

Esilaskenta-algorimista saadaan

$x_0 = x = 280565$
$x_1 = x^4 = 504417$
$x_2 = x^{16} = 353080$
$x_3 = x^{64} = 319218$
$x_4 = x^{256} = 119034$
$x_5 = x^{1024} = 329$
$x_6 = x^{4096} = 415294.$

Taulukossa 23 ovat algorimin suorituksen kannalta vain oleellisimmat esilaskennat, koska esilaskettavia lukuja olisi $(2^h - 1) = 127$ kappaletta.

| $i$ | 1 | 2 | $\cdots$ | 87 | $\cdots$ |
|---|---|---|---|---|---|
| $G[0][i]$ | 280565 | 504417 | | 494824 | |

Taulukko 23

Taulukossa 24 ovat kampamenetelmällä suoritetut laskutoimitukset.

| $k$ | $j$ | $A$ |
|---|---|---|
| 1 | | 1 |
| | 0 | $G[0][2] \cdot A = 504417$ |
| 0 | | $A \cdot A = 123667$ |
| | 0 | $G[0][87] \cdot A = 441132$ |

Taulukko 24

Algoritmi palauttaa muuttujan $A = 441132$, joka on potenssiin korotuksen $280565^{4381} \pmod{506581}$ tulos.

Tässäkin, kuten kahdessa edellisessä algorimissa, algoritmi olisi nopea ilman esilaskentaa. RSA:ssa tarvittava esilaskenta joudutaan kuitenkin joka kerta tekemään uudelleen, koska luku $x$ vaihtuu joka kerta.

## 4.11 Yhteenveto

Jos neliöinti on suunnilleen yhtä arvokas kuin kertolasku, niin taulukossa 25 on esitettynä kunkin algoritmin vaatima työmäärä yllä suoritetuissa laskuissa.

Kuten jo kohdissa 4.8, 4.9 ja 4.10 tuli esille, niin näissä algoritmeissa esilaskenta vaatii paljon työtä. Esilaskentaa ei ole esimerkeissä laskettu, mutta taulukossa 25 on esillä tarvittavien laskutoimitusten määrä, jotka tarvitaan esilaskettavien laskujen suorittamiseen. RSA menetelmällä nämä algoritmit ovat tehottomia, koska esilaskennan vaatima työ hidastaa laskemista. Nämä viimeiset algoritmit toimivat kuitenkin loistavasti tilanteissa, joissa samalla kantaluvulla $x$ on useita eri eksponentteja $e$.

Montgomeryn potenssiin korotuksessa suoritetaan 21 Montgomeryn kertolaskua, joista jokainen koostuu noin kahdestakymmenestäneljästä yksinkertaisesta kertolaskusta. Nämä kertolaskut ovat huomattavasti vähemmän työläitä kuin vertailussa käytetyt modulaariset kertolaskut ja neliöinnit.

| Algoritmi | Esilaskenta | Laskutoimitusten lukumäärä |
|---|---|---|
| 3.1 | | 17 |
| 3.2 | | 17 |
| 3.3 | 2 | 16 |
| 3.4 | 2 | 16 |
| 3.5 | 4 | 17 |
| 3.7 | | 21 |
| 3.9 | | 17 |
| 3.10 | 12 | 6 |
| 3.11 | 12 | 6 |
| 3.13 | 132 | 3 |

Taulukko 25

## 4.12 Arvio laskutoimistusten määrästä todellisen suuruisilla luvuilla

Nykyään RSA:ssa luku $n$ on suuruudeltaan vähintään noin 1000 bittiä ja $1 < x < n$. Luvun $e$ valinta voi vaikuttaa huomattavasti laskutehokkuuteen. Oletetaan, että luvuissa $x$ ja $e$ molemmissa olisi kyseiset 1000 bittiä. Arvioidaan laskutoimitusten määrää kullakin algoritmilla tässä tilanteessa.

Algoritmeissa 3.1 ja 3.2 tarvittava laskutoimitusten määrä on sama. Kun luvun $e$ pituus on $t + 1 = 1000$, niin huonoimmassa tilanteessa tarvitaan $t = 999$ neliöintiä ja $t = 999$ kertolaskua, näin ollen $2t = 1998$ laskutoimitusta. Keskivertotilanteessa työmäärä olisi $3t/2 \approx 1499$ laskutoimitusta.

Algoritmeissa 3.3 ja 3.4 tarvittavien kertolaskujen ylärajat ovat samat. Näissä luvun $k$ valinta vaikuttaa tarvittavien laskutoimitusten määrään. Jos esimerkiksi valitaan luku $k = 10$, niin $l = \lfloor t/k \rfloor = 99$. Algoritmi 3.3 tarvitsee tällöin $lk = 990$ neliöintä ja korkeintaan 98 kertolaskua. Tarvittavia laskutoimituksia olisi 1088. Esilaskentaan tämä algoritmi tarvitsee $(1+2^k-3) = 1022$ laskutoimitusta.

Algoritmi 3.4 tarvitsee esilaskentaan 512 laskutoimitusta, kun luku $k =$

10. Koska $0 \leq h_l \leq s$, missä $s$ on pienin mahdollinen $t \pmod{k}$, niin valitaan $h_l$ olemaan suurin mahdollinen, eli $h_l = s = 9$. Näin ollen tarvittavien neliöintien määrä olisi 999 ja kertolaskujen määrä olisi 98. Yhteensä tarvittavia laskutoimituksia on 1097.

Algoritmin 3.5 tarvitsemat laskutoimitukset on hankalampi selvittää. Olkoon edelleen $k = 10$. Esilaskentaan tarvitaan 512 laskutoimitusta. Algoritmin suorituksessa tarvittavien laskutoimitusten määrä riippuu luvun $e$ ominaisuuksista. Jos joka kierroksella pystytään muodostamaan pisin mahdollinen bittijono, niin algoritmin suorittamiseksi laskutoimituksia tarvittaisiin 1100.

Algoritmi 3.7 vaatii keskimäärin $3t/2 + 2 = 1500$ Montgomeryn kertolaskua. Tämä on riippuvainen luvun $e$ ykkösten lukumäärästä. Huonoimmassa tapauksessa luvussa $e$ on 1000 ykköstä, jolloin Montgomeryn kertolaskuja suoritettaisiin 2002 kappaletta.

Algoritmia 3.9 varten luvusta $e$ muodostetaan yhteenlaskuketju. Algoritmin suorittamiseen tarvitaan yhteenlaskuketjun pituuden $s$ verran kertolaskuja. Huonoimmassa mahdollisessa tilanteessa tarvittavien kertolaskujen määrä olisi melkein 2000. Tällöin yhteenlaskuketju olisi muodostettu binäärisestä esitysmuodosta ja siinä kaikki luvut olisivat ykkösiä. Keskivertotilanteessa luvussa $e$ olisi 500 ykköstä. Tällöin tarvittavia kertolaskuja olisi 1500, kun yhteenlaskuketju on muodostettu luvun binäärisestä muodosta. Kuitenkaan binäärinen esitystapa ei aina ole edullisin tapa muodostaa yhteenlaskuketju. Alaraja lyhyimmälle mahdolliselle yhteenlaskuketjulle saadaan lausekkeesta $log_2 e + log_2 wt(e) - 2,13$. Jos luvussa $e$ olisi 500 ykköstä, niin algoritmin suorittaminen vaatisi noin 1000 kertolaskua.

Algoritmissa 3.10 esilaskenta vaatii taas runsaasti työtä, mutta algoritmin suorittamiseen tarvittavien laskutoimitusten määrä riippuu luvun $h$ valinnasta. Luvut $h$ ja $b$ voidaan valita samoiksi. Algoritmin suorittamiseen tarvitaan korkeintaan $t + h - 2$ kertolaskua. Jos luku $h = 10$ ja siten myös $b = 10$, niin luku $e$ olisi kymmenkantainen ja silloin $t$ olisi noin 300. Tarvittavien laskutoimitusten määrä olisi 308. Esilaskettavia alkoita olisi 300 ja esilaskentaan tarvittaisiin 1196 neliöintiä.

Algoritmin 3.11 tarvitsemien laskutoimitusten arvioiminen on hankalaa.

Jos kanta $b$ on sama kuin algoritmissa 3.10 niin laskennalliset vaatimukset eivät ole paljoa suuremmat kuin algoritmissa 3.10. Suuremmasta $h$:n arvosta on tässä algoritmissa enemmän etua kuin algoritmissa 3.10. Oletetaan, että tämän algoritmin laskutoimitusten määrä on sama kuin algoritmissa 3.10. Myös esilaskenta vaatii saman verran laskutoimituksia kuin algoritmi 3.10.

Algoritmin 3.13 vaatimien laskutoimitusten määrään vaikuttaa lukujen $h$ ja $v$ valinta ja niistä määritellyt $a$ ja $b$. Algoritmin suorittamiseksi tarvitaan korkeintaan $a + b - 2$ laskutoimitusta. Jos valitaan $h = 10$ ja $v = 10$, niin saadaan $a = 100$ ja $b = 10$. Näin ollen algoritmi suorittaa korkeintaan 108 laskutoimitusta. Muistitilaa tarvitaan $v(2^h - 1) = 10230$ esilasketulle alkiolle. Esilaskenta-algoritmin ensimmäisessä kohdassa tarvitaan 900 laskutoimitusta. Kohdassa 2.1 tarvitaan 1013 kertolaskua. Kohdassa 2.2 laskettavat luvut vaativat 10 neliöintiä. Joka rivillä on 1023 alkiota ja rivejä on 9. Näin ollen kohta 2.2 vaatii 92070 neliöintiä. Kaikkiaan esilaskentaan tarvitaan 93983 laskutoimitusta.

Taulukossa 26 on esitettynä yllä saadut tulokset.

| Algoritmi | Esilaskenta | Laskutoimitusten lukumäärä |
|---|---|---|
| 3.1 | | 1499 |
| 3.2 | | 1499 |
| 3.3 | 1022 | 1088 |
| 3.4 | 512 | 1097 |
| 3.5 | 512 | 1100 |
| 3.7 | | 1500 |
| 3.9 | | 1000 |
| 3.10 | 1196 | 308 |
| 3.11 | 1196 | 308 |
| 3.13 | 93983 | 108 |

Taulukko 26

# Kirjallisuutta


[1] Çetin Kaya Koç: *High-Speed RSA Implementation*, RSA Laboratories, RSA Data Security, Inc, 1994. ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf

[2] A.Menezes, P. van Oorschot and S. Vanstone: *Handbook of Applied Cryptography*, CRC Press, Inc, 1997.

[3] A. Renvall: *Kryptografia I*, Matematiikan laitos, Turun yliopisto, 2005.

[4] *The RSA Factoring Challenge*, http://www.rsa.com/rsalabs/node.asp?id=2092, 8.2.2012.

[5] B. Schneier: *Applied Cryptography*, John Wiley & Sons, 1996.

[6] H. van Tilborg: *Encyclopedia of Cryptography and Security*, Springer Science + Business Media, Inc., 2005.