Simo K. Kivelä, 25.1.2005

RSA-salakirjoitus

Ron Rivest, Adi Shamir ja Leonard Adleman esittivät vuonna 1978 salakirjoitusmenettelyn, jossa tietylle henkilölle osoitetut viestit voidaan salakirjoittaa hänen ilmoittamallaan *julkisella avaimella* ja hän yksin voi lukea ne vain omassa tiedossaan olevan *salaisen avaimen* avulla. Menettely sai nimekseen *RSA-salakirjoitus* esittäjien sukunimien alkukirjaimien mukaan.

Seuraava dokumentti esittää, miten menetelmä toimii. Lukija voi sen avulla luoda itselleen julkisen ja salaisen avaimen. Esimerkki on kuitenkin vain periaatteellinen: Jotta saataisiin todella käyttökelpoinen salausmenettely, olisi käytettävä huomattavasti suurempia alkukuja kuin esimerkissä.

RSA-salausta pidetään käytännössä luotettavana. Sen murtaminen edellyttäisi nopeata algoritmia luvun tekijöihin jakamiseen. Ainakaan toistaiseksi ei kuitenkaan ole osoitettu, että tällaista algortimia ei ole, mutta sellaista ei myöskään tunneta.

RSA-salausta voidaan käyttää myös viestin lähettäjän varmistamiseen. Jos lähettäjä salaa viestin omalla salaisella avaimellaan ja vastaanottaja voi avata sen vastaavalla julkisella avaimella, viesti on todella peräisin ilmoitetulta lähettäjältä, koska vain hän tietää salaisen avaimen.

Apufunktioita

Seuraavassa määritellään eräät apufunktiot, joita tarvitaan viestien käsittelyssä. RSA-algoritmi edellyttää viestin olevan numeerisessa muodossa, jolloin kirjaimet on muutettava numeroiksi. Yksinkertaisuuden vuoksi oletetaan, että viestissä on vain pieniä kirjaimia. Isoja kirjaimia tai esimerkiksi numeroita ei siis saa olla lainkaan, ei myöskään välilyöntejä.

Funktio `toCode` muuntaa viestin kirjaimet a, b, c, ..., ö numeroiksi 1, 2, 3, ..., 29:

```
In[1]:= toCode[c_] := (ToCharacterCode[c] - 96) /. {133 → 27, 132 → 28, 150 → 29}
```

Vastaava käänteisfunktio on `fromCode`:

```
In[2]:= fromCode[c_] := FromCharacterCode[(c /. {27 → 133, 28 → 132, 29 → 150}) + 96]
```

Viesti on annettava lainausmerkeissä olevana merkkijonona, jolloin saadaan lukuarvojen muodostama (aaltosulkuihin suljettu) lista. Esimerkiksi:

```
In[3]:= toCode["viesti"]
Out[3]= {22, 9, 5, 19, 20, 9}

In[4]:= fromCode[{22, 9, 5, 19, 20, 9}]
Out[4]= viesti
```

Tulos on merkkijono, vaikka näkyvissä ei olekaan lainausmerkkejä:

```
In[5]:= Head[%]
Out[5]= String
```

Yksittäisten kirjainmerkkien salaaminen ei ole riittävää, koska tällainen viesti voidaan helposti murtaa esimerkiksi merkkien frekvenssitarkastelulla. Tämän takia merkit ryhmitetään vakiopituisiksi jaksoiksi. Tarvittava funktio on

`toGroups`, joka saa argumentikseen numerokoodimuodossa olevan viestin ja jaksoon tulevien merkkien lukumäärän. Käänteisfunktio on `fromGroups`.

```
In[6]:= toGroups[c_, m_] := Map[FromDigits[#, 100] &, Partition[c, m, m, {1, 1}]]

In[7]:= fromGroups[c_] := Flatten[IntegerDigits[c, 100]]
```

Esimerkiksi:

```
In[8]:= toGroups[{18, 19, 1, 19, 1, 12, 1, 11, 9, 18, 10, 15, 9, 20, 21, 19,
          20, 1, 11, 15, 19, 11, 5, 22, 1, 1, 18, 20, 9, 11, 11, 5, 12, 9}, 5]
Out[8]= {1 819 011 901, 1 201 110 918, 1 015 092 021,
         1 920 011 115, 1 911 052 201, 118 200 911, 1 105 120 918}

In[9]:= fromGroups[{1 819 011 901, 1 201 110 918, 1 015 092 021,
          1 920 011 115, 1 911 052 201, 118 200 911, 1 105 120 918}]
Out[9]= {18, 19, 1, 19, 1, 12, 1, 11, 9, 18, 10, 15, 9, 20, 21, 19, 20,
         1, 11, 15, 19, 11, 5, 22, 1, 1, 18, 20, 9, 11, 11, 5, 12, 9, 18}
```

Jos ryhmitettävän listan alkioiden määrä ei ole jaollinen halutulla jakson pituudella, listaa täydennetään ottamalla lisäelementtejä tarpeellinen määrä listan alusta (esimerkissä näitä on yksi). Käänteisfunktio ei poista näitä ylimääräisiä elementtejä (jolloin se ei oikeastaan ole käänteisfunktio).

Varsinainen RSA-salaus tapahtuu tämän jälkeen kullekin jaksolle erikseen salausavainta käyttämällä. Tämä muodostuu kahdesta luvusta, e ja n. Kunkin jakson luku v muunnetaan kaavalla $c = v^e \pmod n$, missä c on salattu viesti (sen osa).

Salauksen purkuun käytetään toista avainta, joka myös muodostuu kahdesta luvusta, d ja n. Purku tapahtuu kaavalla $v = c^d \pmod n$.

Potensseja koskeva modulaariaritmetiikka on Mathematicassa toteutettu tehokkaasti funktiolla `PowerMod`, siis `c = PowerMod[v,e,n]` ja `v = PowerMod[c,d,n]`.

Salauksen tuloksena syntyy lista kokonaislukuja, joiden numeroiden määrä riippuu luvusta n. Kokonaisluvut muunnetaan yhtä pitkiksi merkkijonoiksi, jolloin alkuun tarvittaessa lisätään nollia, ja merkkijonot yhdistetään yhdeksi jonoksi. Tarvittava funktio on `toOneString` ja sen käänteisfunktio `fromOneString`.

```
In[10]:= toOneString[c_, n_] :=
          StringJoin[Map[ToString, Flatten[Map[IntegerDigits[#, 10, n] &, c]]]]

In[11]:= fromOneString[c_, n_] :=
          Module[{s}, s = StringJoin[c, Table["0", {Mod[-StringLength[c], n]}]];
           Table[ToExpression[StringTake[s, {k * n + 1, (k + 1) * n}]],
            {k, 0, StringLength[s] / n - 1}]]
```

Esimerkiksi:

```
In[12]:= toOneString[{123, 1234, 12 345, 123 456}, 6]
Out[12]= 000123001234012345123456

In[13]:= fromOneString["000123001234012345123456", 6]
Out[13]= {123, 1234, 12 345, 123 456, 100 000}
```

Yhdistämällä edellä muodostetut funktiot saadaan salakirjoitusta ja sen purkamista varten seuraavat funktiot:

```
In[14]:= encrypt[v_, e_, n_, m_] := toOneString[
          PowerMod[toGroups[toCode[v], m], e, n], Length[IntegerDigits[n]]]
```

```
In[15]:= decrypt[c_, d_, n_] := fromCode[
           fromGroups[PowerMod[fromOneString[c, Length[IntegerDigits[n]]], d, n]]]
```

Merkkien ryhmittelyssä käytettävä jaksonpituus *m* ei saa olla liian suuri, mikä ilmenee RSA-algoritmin pätevyyden todistuksesta.

Avaimien muodostaminen

Salausavain muodostuu kahdesta luvusta, *e* ja *n*. Näihin liittyy purkuavain, joka myös muodostuu kahdesta luvusta, *d* ja *n*. Oleellista on, että lukujen *e* ja *n* perusteella ei voida (ainakaan helposti) laskea vastaavaa lukua *d*. Tällöin salausavain voi olla julkinen, mutta purkuavain salainen.

Avaimien muodostamista varten valitaan kaksi alkulukua. Jotta saadaan vaikeasti murrettava salausjärjestelmä, näiden tulee olla suuria. Esimerkkiin valitaan kuitenkin melko pienet:

```
In[16]:= p = 7919; q = 17 389;
```

Nämä ovat todella alkulukuja:

```
In[17]:= {PrimeQ[p], PrimeQ[q]}
Out[17]= {True, True}
```

Luvut voidaan myös antaa alkuluvun järjestysluvun perusteella: `Prime[k]` antaa *k*:nnen alkuluvun.

Luku *n* on näiden tulo. Oleellista on, että isojen lukujen tapauksessa tekijöiden selvittäminen on vaikeaa: Jos vain *n* tunnetaan, ei ole helppoa löytää tekijöitä *p* ja *q*. Tähän perustuu salauksen pitävyys. Luku *n* on julkinen ja jos se pystytään jakamaan tekijöihin, löydetään myös salainen avain. Isojen lukujen tapauksessa tekijöihin jakoon ei kuitenkaan tunneta nopeaa algoritmia, vaan tarvittava tietokoneaika on helposti miljoonia vuosia riittävän isoilla luvuilla *p* ja *q*.

```
In[18]:= n = p q
Out[18]= 137 703 491
```

Funktio `Timing` antaa laskutoimitukseen, tässä tapauksessa tekijöihin jakoon (`FactorInteger`) kuluneen ajan:

```
In[19]:= Timing[FactorInteger[n]]
Out[19]= {0., {{7919, 1}, {17 389, 1}}}
```

Luku *e* valitaan siten, että sillä ei ole yhteisiä tekijöitä luvun $(p-1)(q-1)$ kanssa, ts. näiden suurimman yhteisen tekijän (syt = gcd, greatest common divisor) tulee olla $= 1$. Tällaisia lukuja on helpsoti löydettävissä:

```
In[20]:= e = 13; GCD[e, (p - 1) (q - 1)]
Out[20]= 1
```

Luvun *d* tulee toteuttaa yhtälö $d\,e \equiv 1 \pmod{(p-1)(q-1)}$ eli $d\,e - 1 = k\,(p-1)(q-1)$ jollakin kokonaisluvulla *k*. Luku *d* on siten muotoa $[k\,(p-1)(q-1) + 1]/e$ oleva kokonaisluku. Tällainen voidaan löytää Eukleideen algoritmilla, mutta myös kokeilemalla. Suraavassa kokeillaan kokonaislukuarvoja luvulle *k* nollasta alkaen arvoon 10 *e* saakka (vähempikin riittäisi) ja poimitaan tuloksista ne, jotka ovat kokonaislukuja.

```
In[21]:= dlist = Select[Table[(k (p - 1) (q - 1) + 1) / e, {k, 0, 10 e}], IntegerQ]
Out[21]= {116 496 925, 254 175 109, 391 853 293, 529 531 477, 667 209 661,
  804 887 845, 942 566 029, 1 080 244 213, 1 217 922 397, 1 355 600 581}
```

Valitaan ensimmäinen luvuista luvuksi d:

```
In[22]:= d = dlist[[1]]
Out[22]= 116 496 925
```

Viestin salakirjoitus ja avaus

Viestin alkioiden ryhmityksessä käytetty jaksonpituus vaikuttaa siihen, miten suuriin lukuihin varsinainen salausalgoritmi kohdistetaan. Jotta algoritmi toimisi, tulee lukujen olla $< n$. Jos siis käytetään suhteellisen pieniä alkulukuja, ei voida käyttää kovin pitkiä jaksoja.

Kun ryhmien jaksonpituudeksi valitaan

```
In[23]:= m = 4
Out[23]= 4
```

viestin salakirjoitus ja salatun viestin avaus onnistuu:

```
In[24]:= v = "abcdefghijklmnopqrstuvwxyzääö";
```

```
In[25]:= c = encrypt[v, e, n, m]
Out[25]= 0182383190928249831051782961270296210477094631219558140243261010692001 33
```

```
In[26]:= vv = decrypt[c, d, n]
Out[26]= abcdefghijklmnopqrstuvwxyzääöabc
```

Harjoitustehtäviä

1) Tutki, miten ryhmityksen jaksonpituuden kasvattaminen vaikuttaa salaus- ja purkualgoritmien toimintaan. Tutki myös tapausta $p = 3$, $q = 5$, $m = 1$. Selitä saamasi tulokset.

2) Tutki, mitä tapahtuu, jos algoritmia yritetään soveltaa siten, että p ja q eivät olekaan alkulukuja.

3) Tutki, minkä takia luku e on valittava siten, että sillä ei ole yhteisiä tekijöitä luvun $(p - 1)(q - 1)$ kanssa (ts. suurin yhteinen tekijä $= 1$).