Universitetet i Sørøst-Norge

Hjemmeeksamen
Emnekode: *RFMA310*
Emnenavn: *Diskret matematikk*

Kandidatnr:*22*

Monday 6 January, 2020

Contents

List of Figures

Listings

Abstrakt

I denne raporten skal jeg skrive et Python script for RSA kryptograf og tester programmet med ulike input og gir en gjennomgang av det relevant matematikk som jeg bruker i programmet.

0.1 Introduksjon

RSA Kryptografi er beskrevet i 1977 av Ron Rivest, Adi Shamir og Leonard Adleman og det er grunnlaget for kryptosystem. Dette RSA algoritmen er brukt mye for å sikre sensitive informasjon/ data som vi sender over internett og det er også mye brukt for spesifikk mål. RSA algoritmen bruker to forskjellige nøkkler for å kryptere og dekryptere meldingen.

På eksemplet under kan vi se at Alice sender en melding til Bob, så for hun sender meldingen til Bob over internett så krypter hun meldingen med sin offentlig nøkkel(public key) og når Bob mottar meldingen så han dekrypterer den med sin privat nøkkel (pivate key) så kan han lese meldingen. Dersom Bob svarer på meldingen så krypterer han meldingen med sin public key og Alice dekrypterer meldingen med sin private key. Dette er også kjent som en asymmetrisk kryptograf og de to forskjellige nøkkeler er matematisk linket og den offentlige nøkkelen er delt med alle mens privat nøkkelen er holdt hemmelig. Denne algoritmen gir metoder som forsikrer konfidensialitet, integritet, autentisitet.

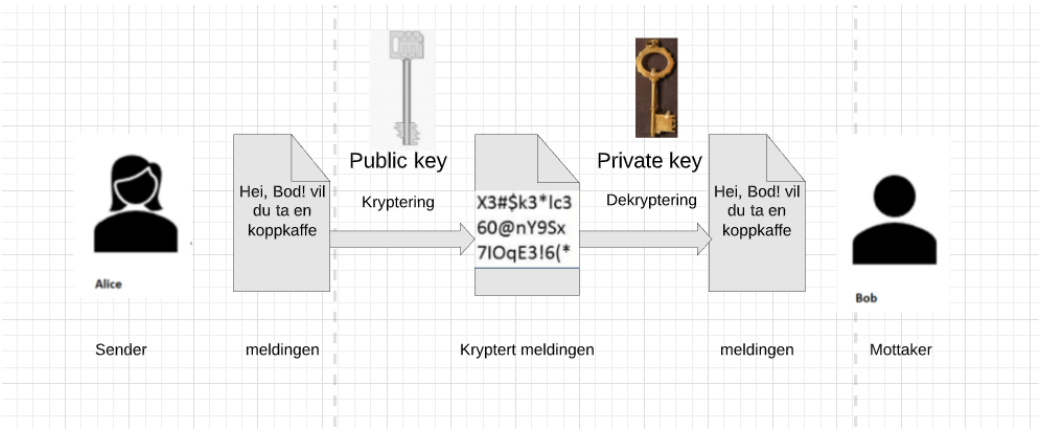

Figure 1: RSA public key kryptografi

RSA er en sikker metode for å sikre informasjon over internett og det er på grunn av at RSA henter sin sikkerhet fra produktet av to primtall som ganges med hverandre. Men for å få de opprinnelige primtallene tilbake fra summen er det svært vanskelig og tar alt for langt tid og det er fordi at vi bruker super datamaskiner. Det er sikkert på grunn av helltall faktorisering problem.

Produsering av nøkkeler er det komplekse delen i RSA kryptosystemet. De to primtallene er genereres ved bruk av Rabin Miller algoritmen og modulo n regnes ut ved å multiplisere de to primtallene. Dette tallet er brukt både i den offentlige og private nøkkelen.

Den offentlige nøkkelen består av modulo n og eksponenten e som skal være et primtall men trenger ikke være så stor og det fordi at den offentlige nøkkelen er delt med alle. Mens den private nøkkelen består av n og eksponenten d som er beregnet ved å bruke av Extended Euclidean algoritmen.

0.2 Relevant matematikk

I RSA kryptosystemet hver enkelte har en krypterings nøkkel (e, n) hvor n = p * q og modulo er produktet av to primtallene p og q. Mens den eksponenten er forholdsvis primtall av (p - 1)*(q - 1).
La oss si at primtallene p og q har følgende verdi:
p = 53 og q = 59

Regner ut n = p * q
n = 53 * 59 = 3127

Nå bruker jeg Euler totient funksjon
$\phi(n) = (p-1) * (q-1) => (53-1) * (59-1) = 3016$
Et primtall er regnet ut fra i rekkevident
[3, $\phi(n)$)som har en største felles divisor av 1 med $\phi(n)$
Nå trenger jeg å finne en eksponent e og det kan vi finne slik:
gcd(e, $\phi(n)$) = 1 så e = 3
Regnet ut den modulær inversen av e så d

$$d = \frac{2 * (\phi(n)) + 1}{e} => \frac{2 * (3016) + 1}{3} = 2011 \tag{1}$$

public key (e, n) og private key(d, n)
public key(3, 3127) og private key (2011, 3127)

Hvordan dette virker når vi krypterer og dekrypterer i virkelighen.
For eksempel hvis skal kryptere: "God jul" og dekryptere det.

Formell for kryptere teksten: "God jul"
kryptering => kryptert_tekst = ren_tekst ^{e} mod n

Tegn	G	o	d	space	j	u	l
ASCII verdi	71	111	100	32	106	117	108

Figure 2: ASCII representasjon av meldingen

Vi trenger å konvertere teksen til tall eller bruke ASCII representasjon av teksten. Men jeg vil bruke ASCII representasjon: Siden jeg bruker ASCII og jeg vil legge den krypterte teksten i en array:
$71^3 \bmod 3127 = 1433$
$111^3 \bmod 3127 = 1132$
$100^3 \bmod 3127 = 2487$
$32^3 \bmod 3127 = 1498$
$106^3 \bmod 3127 = 2756$
$117^3 \bmod 3127 = 589$
$108^3 \bmod 3127 = 2658$

Når teksten er kryptert så vil den se slik ut i en array:
[1433, 1132, 2487, 1498, 2756, 589, 2658]

Nå skal jeg dekryptere meldingen:
Formellen er: kryptert_tekst^d mod n
$1433^{2011} \bmod 3127 = 71$
$1132^{2011} \bmod 3127 = 111$
$2487^{2011} \bmod 3127 = 100$
$1498^{2011} \bmod 3127 = 32$
$2756^{2011} \bmod 3127 = 106$
$589^{2011} \bmod 3127 = 117$
$2658^{2011} \bmod 3127 = 108$

Når teksten er ferdig dekryptert så skal jeg legge dem i rekkefølge og starter fra toppen. '7111110032106117108'. Nå skal jeg bruke ASCII tabellen og det gir oss det originale meldingen: "God jul".

0.3 Test av programmet med ulike input

Programmet tar input fra brukeren og brukeren taster inn meldinger som de ønsker å kryptere og dekryptere med RSA algoritmen. På eksempler under kan vi se meldinger som skal krypteres i lilla farge og den krypterte meldingen ser vi i et array som inneholder en haug med tall. Hvert element i arrayet representerer et tegn i meldingen. Tilslutt så dekrypterer jeg den krypterte meldingen og får tilbake den originale teksten og det ser vi i siste linje på eksempler og har lilla farget.

I programmet så bruker primtall mellom 20^{12} og 99^8 og det er ganske stort primtall men i virkeligheten så bruker vi super datamaskiner for å regne stor primtall. Desto større primtall bruker vi for kryptering, desto vanskeligere blir det for å faktorisere de primtallene.

```
PS C:\Users\m_rah\Desktop\RSA> python .\hjemmeeksamenRSA.py
Skriv en melding som du vil kryptere:
'Hjemmeeksamen i diskret matematikk 2019'
Dette er din krypterte meldingen:
[47394150187379922999676 4, 460533365819803487667584,
   18277627125755644624050 9, 35832222836568997712650 5,
   35832222836568997712650 5, 18277627125755644624050 9,
   18277627125755644624050 9, 18735149689519072276102 3,
   40952340395456814846653 6, 43203332456248380043430 9,
   35832222836568997712650 5, 18277627125755644624050 9,
   16928942925143798606314 6, 46554902725279267015750,
   43957100250626906458595 1, 46554902725279267015750,
   28872219010068455901449 8, 43957100250626906458595 1,
   40952340395456814846653 6, 18735149689519072276102 3,
   54878782645148261783991, 18277627125755644624050 9,
   19181172699398349014090 3, 46554902725279267015750,
   35832222836568997712650 5, 43203332456248380043430 9,
   19181172699398349014090 3, 18277627125755644624050 9,
   35832222836568997712650 5, 43203332456248380043430 9,
   19181172699398349014090 3, 43957100250626906458595 1,
   18735149689519072276102 3, 18735149689519072276102 3,
   46554902725279267015750, 15245966841330554753645 9,
   16955074083215508605521 6, 87218218639311241036388,
   43652518261974006510 52]
Dette er din opprinnelige meldingen:
'Hjemmeeksamen i diskret matematikk 2019'
```

Listing 1: RSA Test 1

```
PS C:\Users\m_rah\Desktop\RSA> python .\hjemmeeksamenRSA.py
Skriv en melding som du vil kryptere:
'RSA alogritmen er vanskelig    bli hacket og det tar kjempe
    langt tid'
Dette er din krypterte meldingen:
[234668793010339851857794, 266785401315790066424781,
    29965092866174454284146 4, 313448927389616199478126,
    55412545002158698958959 1, 860179097477269007797 57,
    23898682676259336324447 8, 310652902403386876766709,
    671525319867331187771 73, 478398213872160429576361,
    15788080164391065671434 7, 216224909147294690233149,
    38871981962808795326995 3, 314821920065544347866930,
    31344892738961619947812 6, 38871981962808795326995 3,
    671525319867331187771 73, 31344892738961619947812 6,
    14543318344850712576850 7, 55412545002158698958959 1,
    31482192006554434786693 0, 35033424260079759647504 5,
    38362667551511035733419 0, 38871981962808795326995 3,
    860179097477269007797 57, 478398213872160429576361,
    31065290240338687676670 9, 31344892738961619947812 6,
    39416696645765727154617 7, 31344892738961619947812 6,
    34314337149361794767564 2, 860179097477269007797 57,
    478398213872160429576361, 31344892738961619947812 6,
    19994284784903532832060 4, 55412545002158698958959 1,
    31110019221217745379655 4, 38362667551511035733419 0,
    38871981962808795326995 3, 15788080164391065671434 7,
    31344892738961619947812 6, 23898682676259336324447 8,
    31065290240338687676670 9, 31344892738961619947812 6,
    90058856211762530786131, 38871981962808795326995 3,
    15788080164391065671434 7, 31344892738961619947812 6,
    15788080164391065671434 7, 55412545002158698958959 1,
    671525319867331187771 73, 31344892738961619947812 6,
    38362667551511035733419 0, 24005920446125678676115 4,
    38871981962808795326995 3, 21622490914729469023314 9,
    13838342863996001587145 4, 38871981962808795326995 3,
    31344892738961619947812 6, 860179097477269007797 57,
    55412545002158698958959 1, 31482192006554434786693 0,
    31065290240338687676670 9, 15788080164391065671434 7,
    31344892738961619947812 6, 15788080164391065671434 7,
    47839821387216042957636 1, 90058856211762530786131]
Dette er din opprinelige meldingen:
'RSA alogritmen er vanskelig    bli hacket og det tar kjempe
    langt tid'
```

Listing 2: RSA Test 2

```
PS C:\Users\m_rah\Desktop\RSA> python .\hjemmeeksamenRSA.py
Skriv en melding som du vil kryptere:
'RSA bruker offentlig n kkel for    kryptere meldingen og
   bruker privat n kkelen til    dekryptere meldingen'
Dette er din krypterte meldingen:
[541754897567054421175079, 463655399151135136250817,
   343953090251349479168912, 440066033480001892327253,
   666379678380562598240990, 465123141698956140778496,
   319957671777848161976620, 212715723186630735136353,
   545514246089805753188895, 465123141698956140778496,
   440066033480001892327253, 163206982738165291366163,
   202623161552800543274407, 202623161552800543274407,
   545514246089805753188895, 559964796854116313900506,
   241810624949797897912893, 265921852745746237646054,
   178994120892916299179134, 156818223464100070840026,
   440066033480001892327253, 559964796854116313900506,
   633643181480746071375509, 212715723186630735136353,
   212715723186630735136353, 545514246089805753188895,
   265921852745746237646054, 440066033480001892327253,
   202623161552800543274407, 163206982738165291366163,
   465123141698956140778496, 440066033480001892327253,
   525678310108893351742461, 440066033480001892327253,
   212715723186630735136353, 465123141698956140778496,
   627048288924663141312468, 586324429350473189650746,
   241810624949797897912893, 545514246089805753188895,
   465123141698956140778496, 545514246089805753188895,
   440066033480001892327253, 658865799502488669119187,
   545514246089805753188895, 265921852745746237646054,
   371670479556028081680039, 178994120892916299179134,
   559964796854116313900506, 156818223464100070840026,
   545514246089805753188895, 559964796854116313900506,
   440066033480001892327253, 163206982738165291366163,
   156818223464100070840026, 440066033480001892327253,
   666379678380562598240990, 465123141698956140778496,
   319957671777848161976620, 212715723186630735136353,
   545514246089805753188895, 465123141698956140778496,
   440066033480001892327253, 586324429350473189650746,
   465123141698956140778496, 178994120892916299179134,
   164416913488359746127459, 242028214500189741691665,
   241810624949797897912893, 440066033480001892327253,
   559964796854116313900506, 633643181480746071375509,
   212715723186630735136353, 212715723186630735136353,
   545514246089805753188895, 265921852745746237646054,
   545514246089805753188895, 559964796854116313900506,
```

```
   4400660334800018923272 53, 2418106249497978979128 93,
   1789941208929162991791 34, 2659218527457462376460 54,
   4400660334800018923272 53, 5256783101088933517424 61,
   4400660334800018923272 53, 3716704795560280816800 39,
   5455142460898057531888 95, 2127157231866307351363 53,
   4651231416989561407784 96, 6270482889246631413124 68,
   5863244293504731896507 46, 2418106249497978979128 93,
   5455142460898057531888 95, 4651231416989561407784 96,
   5455142460898057531888 95, 4400660334800018923272 53,
   6588657995024886691191 87, 5455142460898057531888 95,
   2659218527457462376460 54, 3716704795560280816800 39,
   1789941208929162991791 34, 5599647968541163139005 06,
   1568182234641000708400 26, 5455142460898057531888 95,
   5599647968541163139005 06]
Dette er din opprinelige meldingen:
'RSA bruker offentlig n kkel for    kryptere meldingen og
   bruker privat n kkelen til    dekryptere meldingen'
```

Listing 3: RSA Test 3

0.4 Konklusjon

RSA kryptosystemet bruker to forskjellige nøkkeler for å kryptere og dekryptere informasjon/data som brukerne sender over internett. Matematikken er enkelt, men det er vanskelig å faktorisere to store primtall og RSA henter sin sikkerhet fra de to primtallene. RSA nøkkel bits lengden er på 1024 eller 2048 og dersom man bruker 2048 bits lengde på nøkkelen og det gir bedre sikkerhet.

```
# Importerer noen pakker som jeg trenger videre i programmet
# ------------------------------------------------------------
# Steg 1       Importerer pakker
#-------------------------------------------------------------

import math
import random

#-------------------------------------------------------------
# Steg 2    Sjekker primtallet
#-------------------------------------------------------------
def primtall(tall):
    #sjekker om tallet er mindre enn 2 og primtall kan ikke v re mindre enn 2
    if (tall <=2 ):
        return False
    # sjekker om tallet partall og et partall kan ikke v re primtall
    if tall%2==0:
        return False

    # N   har jeg sjekket om tall er mindre 3
    # N   finner primtallet og det gj r jeg ved    tallet opp til kvadrat av (N)
    #  ker  med 2
    for i in range(3, math.floor(math.sqrt(tall)),2):
        if (tall %i ==0):
            return False

    return True

#-------------------------------------------------------------
# Steg 3   Extended Euclids Algoritme  (EEA)
#-------------------------------------------------------------
# Bruker EEA for    finne modul r  invers i O(log (m)), alts  linear
# Finner verdien til d som er modul r invers av e i RSA
# funksjonen tar to helltall og finner modul r inversen

def m_invers(a,m):
    mod = m
    y = 0
    x = 1
    if (m ==1):
```

```
41          return 0;
42      while (a >1):
43          #kvotient = k
44          k = a // m
45          temp = m
46          # r = remainder
47          # dette like som gcd
48          m = a % m
49          a = temp
50          temp = y
51          # oppdaterer x og y verdiene
52          y = x - k * y
53          x = temp
54          # sjekker om x st rre en 0 og at det er positivt
55      if x <0:
56          x =x+mod
57
58      return x
59
60  # ------------------------------------------------------------
61  # steg 4 gcd algoritmen
62  #-------------------------------------------------------------
63  #This is a simple code to compute gcd numbers
64  # Verifiserer om (e, phi) er komprimer med gcd
65
66  def gcd(x,y):
67      while (y!=0):
68          x, y = y, x%y
69      return  x
70
71  #-------------------------------------------------------------
72  # Steg 5 lage primtall
73  #-------------------------------------------------------------
74  x = pow(20, 12) # => 20^12
75  y = pow(99, 8)  # => 99^8
76
77  def produsere_primtall (start =x, end=y ):
78      tall = random.randint(x,y) # primtallet skal v re mellom
        x og y verdi og det er tilfeldig
79      # sjekker om det er primtall eller ikke
80      while (not primtall(tall)):
81          # hvis det er ikke primtall  s   genererer primtall
       p   nytt
82          tall = random.randint(x,y)
83      # returnerer primtallet
```

```
84      return tall
85
86  #------------------------------------------------------------
87  # Steg 6 : Produserer RSA n kkeler
88  #------------------------------------------------------------
89
90  def RSA_nokkel():
91       # produserer den f rste primtallet
92       primtall_1 = produsere_primtall()
93       # produserer den andre primtallet
94       primtall_2 = produsere_primtall()
95       # Ganger primtall_1 med primtall_2 for    f   n.
96       #    gj re det motsatte for    f   primtall_1 og
     primtall_2 fra n  s   tar den litt tid for n er en
     eksponential
97       n = primtall_1 * primtall_2
98       # Reginer ut phi og bruker Euler sin phi funksjonen
99       phi = (primtall_1-1)*(primtall_2-1)
100      # verdien til e skal  v re mellom 1 og phi
101      e = random.randrange(1,phi)
102      # dobbel sjekker om gcd(e, phi) =1 og at de er komprimer
     hvis de ikke er komprimer  s   kan jeg ikke finne d
103      while gcd(e,phi)!=1:
104          # produserer e  p   nytt hvis det gcd er ikke 1
105          e = random.randrange(1,phi)
106      # d  er modul r invers av e
107      d =m_invers(e,phi)
108      # returnerer  b de privat og offentlige  n kkler
109      return ((d,n),(e,n))
110
111 #------------------------------------------------------------
112 # Steg 7 : krypterer meldingen
113 #------------------------------------------------------------
114 # Funksjonen tar to paramter som er offentlige  n kkelen og
     meldingen og krypterer meldingen ved    bruke den
     offentlige  n kkelen
115 # offentlig_n kkel = on
116 # meldingen = m
117 def krypter(on, m):
118     # her trenger vi e og n for    krypterer meldingen og de
     er offentlig
119     e, n = on
120     # Jeg bruker ascii verdiene for tegn og omformingen
     lagres i en array, arrayet er tomt i starten
121     krypterte_meldingen = []
```

```
122     # vurderer alle bokstavene i en rekkef lge
123     for i in m:
124         temp = ord(i)
125         krypterte_meldingen.append(pow(temp,e,n)) # Her temp
    er grunntallet og e er eksponenten og n er modulo
126     return krypterte_meldingen
127 
128 #------------------------------------------------------------
129 # Steg 8 : Dekrypterer meldingen
130 #------------------------------------------------------------
131 
132 # dekrypterer meldingen ved    bruke privat n kkelen
133 # funksjonen tar to parameter som p-n kkelen og krypterte
    meldingen
134 # privat n kkelen = pn
135 # krypterte_meldingen = kn
136 def dekrypterer(pn, kn):
137     # vi trenger d og n for dekryptere meldingen og det er
    privat
138     d , n = pn
139     m = ''
140     # vurderer alle bokstavene i en rekkef lge
141     for i in kn:
142         temp = pow(i,d,n)
143         m = m + str (chr(temp))
144     return m
145 #------------------------------------------------------------
146 # Siste steg: printer ut resultater
147 #------------------------------------------------------------
148 # Hoved funksjon
149 # privat n kkelen = pn
150 # offentlig_n kkel = on
151 # meldingen = m
152 # krypterte_meldingen = kn
153 
154 if __name__ == "__main__":
155     pn,on = RSA_nokkel()
156     m = input("Skriv en melding som du vil kryptere: ")
157     kn = krypter(on, m)
158     print("Dette er din krypterte meldingen: %s" %kn)
159     opprinnelige_meldingen = dekrypterer(pn, kn)
160     print("Dette er din opprinnelige meldingen: %s" %
    opprinnelige_meldingen)
```

Listing 4: RSA Kryptografi script