Kombinasi Algoritma RSA sebagai Algoritma Enkripsi Pesan dengan Henon *Map* sebagai *Chaotic Map* untuk Pembentukan Kunci Publik dan Kunci Privat

Dicky Novanto / 13515134[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]dickynovanto1103@gmail.com

Abstrak—Algoritma RSA terkenal sebagai algoritma yang digunakan untuk mengenkripsi pesan yang cukup kuat. Algoritma RSA terkenal cukup kuat karena memiliki properti yang membuat kriptanalisis tidak tahu cara untuk melakukan dekripsi cipherteks secara efisien. Namun sayangnya, algoritma ini telah dapat dipecahkan oleh kriptanalisis, walaupun proses kriptanalisis membutuhkan waktu yang banyak. Oleh karena itu, pada makalah ini, penulis membuat modifikasi pada algoritma RSA, yaitu memanfaatkan Henon *Map* sebagai *Chaotic Map* untuk membentuk kunci publik dan kunci privat dari algoritma RSA tersebut, sehingga akan menyebabkan algoritma RSA yang digunakan untuk melakukan enkripsi semakin memiliki sifat *confussion* yang tinggi, dan tentunya kriptanalis akan semakin susah untuk melakukan dekripsi terhadap cipherteks yang ada. Selain itu, pada laporan ini, akan ditunjukkan hasil waktu, memori eksekusi program dan analisis tingkat keamanan algoritma yang telah dimodifikasi ini.

Kata kunci—RSA, Henon *Map*, *Chaotic Map*, enkripsi dan dekripsi.

I. PENDAHULUAN

Sejak dahulu, kriptografi memegang peranan penting dalam proses pengamanan informasi. Dengan adanya kriptografi, pesan yang telah terenkripsi dan kemudian pesan tersebut dikirimkan melalui media apapun, pesan tersebut dapat dikatakan aman jika tidak ada seorang kriptanalis yang berusaha untuk mendekripsi pesan tersebut. Pertama-tama, kriptografi dikembangkan dengan algoritma kunci simetri dengan kunci yang digunakan untuk melakukan enkripsi dan dekripsi sama. Sebagai contoh, algoritma kriptografi kunci simetri yang paling terkenal adalah algoritma Caesar Cipher.

Seiring dengan perkembangan teknologi, mulai diciptakan algoritma kunci asimetri, yaitu kunci yang digunakan untuk melakukan proses enkripsi dan dekripsi berbeda, yaitu menggunakan kunci publik sebagai kunci yang digunakan untuk melakukan proses enkripsi dan kunci privat sebagai kunci yang digunakan untuk melakukan proses enkripsi tersebut. Algoritma terkenal yagn telah menggunakan prinsip tersebut adalah algoritma RSA.

Algoritma enkripsi yang dikembangkan saat ini oleh penulis adalah algoritma yang digunakan untuk melakukan proses enkripsi dan dekripsi yang memanfaatkan 2 buah alat yang telah dikembangkan, yaitu menggunakan RSA sebagai algoritma untuk melakukan enkripsi dan dekripsi, dan Henon *Map* sebagai *Chaotic Map* yang digunakan untuk melakukan proses pembentukan kunci publik dan kunci privat dari algoritma RSA untuk setiap byte pesan yang akan dienkripsi atau didekripsi.

II. DASAR TEORI

A. Pengertian Kriptografi

Definisi lama dari kriptografi adalah ilmu dan seni untuk menjaga kerahasiaan pesan dengan cara menyandikannya ke dalam bentuk yang tidak dapat dimengerti lagi lamanya [RIN18]. Sedangkan definisi baru dari kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan [RIN18].

Pada jaman dahulu, sebelum ditemukannya komputer, proses enkripsi dan dekripsi masih menggunakan pensil dan kertas, sehingga proses enkripsi dan dekripsi yang digunakan masih sederhana, misalkan menggunakan kunci enkripsi dan kunci dekripsi yang sama dan membutuhkan operasi yang tidak banyak pula, seperti hanya digunakan proses rotasi, permutasi, dan/atau substitusi. Sedangkan pada jaman setelah komputer ditemukan, prinsip menjaga keamanan pesan dapat dilakukan dengan menggunakan prinsip dan konsep matematika yang rumit karena perhitungan tersebut dapat dilakukan oleh komputer.

B. Prinsip Penyandian Shannon

Proses serangan kriptografi yang dilakukan oleh kriptanalis biasanya menggunakan serangan statistik kemunculan huruf pada suatu bahasa plainteks, atau dengan kata lain menggunakan frekuensi kemunculan suatu huruf. Prinsip penyandian Shannon pada dasarnya ialah prinsip kriptografi yang digunakan untuk membuat kriptanalis tidak mendapatkan suatu petunjuk statistik dari cipherteks yang ada. Prinsip penyandian Shanon terdiri dari:

1. Confussion

Confussion adalah prinsip penyandian Shannon yang bertujuan untuk membuat bingung kriptanalis karena tidak adanya hubungan antara plainteks, cipherteks, dan kunci yang

digunakan untuk proses enkripsi. Prinsip ini umumnya diterapkan pada algoritma yang menerapkan metode substitusi dalam melakukan enkripsi, sebagai contoh prinsip ini diterapkan pada algoritma *One Time Pad*.

2. Diffusion

Diffusion adalah prinsip penyandian Shannon yang menyebarkan pengaruh 1 bit plainteks atau kunci ke sebanyak mungkin cipherteks yang dihasilkan.

C. Algoritma RSA

Algoritma RSA adalah algoritma dengan kunci asimetri, yaitu menggunakan kunci publik dan kunci privat yang berbeda. Kunci publik digunakan untuk melakukan proses enkripsi, dan kunci privat digunakan untuk melakukan proses dekripsi. Keamanan dari algoritma RSA adalah terletak pada susahnya mencari 2 buah faktor prima dari sebuah bilangan yang sangat besar secara efektif dan efisien.

Adapun terdapat property dari algoritma RSA, yaitu sebagai berikut:

1. p dan q bilangan prima → dianjurkan sangat besar, sekitar 100 digit (rahasia)
2. n = p*q (tidak rahasia)
3. phi(n) = (p-1)*(q-1) (rahasia)
4. e sebagai kunci enkripsi (tidak rahasia), syarat: Greatest Common Divisor (e, phi(n)) = 1
5. d sebagai kunci dekripsi (rahasia)
   d dihitung dengan menggunakan cara $d = e^{-1} \bmod (\text{phi}(n))$
6. m sebagai plainteks (rahasia)
7. c sebagai cipherteks (tidak rahasia)

Properti nomor 3 adalah phi(n) melambangkan euler totient function dari n, yang mana artinya adalah jumlah bilangan diantara 1 hingga n yang relative prima terhadap n. Bilangan prima memiliki ciri khusus pada hasil phi(n), yaitu nilainya adalah n-1. Hal ini karena bilangan prima n relatif prima dengan semua bilangan antara 1 hingga n-1 secara inklusif.

Selain itu phi(n) memiliki property khusus pula, yaitu phi(p*q) = phi(p)*phi(q) = (p-1)*(q-1), sehingga property nomor 3 dari algoritma RSA terbukti benar.

Tentu saja properti yang telah disebutkan sebagai property yang rahasia tidak boleh dibuka kepada publik, hal ini karena apabila property yang rahasia tersebut diketahui, maka plainteks yang dienkripsi dapat secara mudah didapatkan. Sebagai contoh, apabila p atau q telah diketahui, maka dapat langsung diketahui nilai n, yang nantinya dapat dihitung nilai phi(n), kemudian nilai d sebagai kunci dekripsi, dan tentunya dapat langsung didapatkan plainteks dengan proses perhitungan komputer.

Proses enkripsi dilakukan dengan rumus sebagai berikut:

$$E(m) = c = m^e \bmod (n)$$

Sedangkan proses dekripsi dapat dilakukan dengan menggunakan rumus sebagai berikut:

$$D(c) = m = c^d \bmod (n)$$

Teori yang melandasi algoritma RSA adalah algoritma RSA adalah teorema euler sebagai berikut:

$$a^{\text{phi}(n)} \equiv 1 \pmod n$$

Kemudian penurunan rumusnya adalah sebagai berikut:

$$a^{\text{phi}(n)} \equiv 1 \pmod n$$
$$a^{\text{phi}(n)*k} \equiv 1^k \pmod n$$
(kemudian a diganti dengan m)
$$m^{\text{phi}(n)*k+1} \equiv m \pmod n \quad (1)$$

Sesuai dengan property 5 algoritma RSA, khususnya pada cara melakukan dekripsi, jelas bahwa e.d = 1 (mod (phi(n))), sehingga bila diturunkan dari rumus 1, dapat terlihat bahwa property 5 dapat diturunkan menjadi persamaan berikut:

$$e.d \equiv 1 \pmod{\text{phi}(n)}$$
$$e.d = k*\text{phi}(n) + 1$$
sehingga
$$m^{\text{phi}(n)*k+1} \equiv m \pmod n$$
$$m^{e.d} \equiv m \pmod{(n)} \quad \text{QED}$$

Dari persamaan diatas, terlihat bahwa cara untuk mengenkripsi adalah dengan memangkatkan m dengan e, dan untuk mengembalikan cipherteks tersebut, dilakukan pemangkatan kembali dengan d.

Berikut adalah implementasi kode program dalam proses enkripsi dalam bahasa python:

```
def encrypt(self, byteElement):
        elemen = byteElement
        bilangan = pow(elemen, self.e,
self.n) # elemen^self.e % self.n

        return bilangan
```

Berikut adalah implementasi kode program dalam proses dekripsi dalam bahasa python:

```
def decrypt(self, byteElement, pangkat,
mod):
        elemen = byteElement

        kar = pow(elemen, pangkat, mod)

        return bytes([kar])
```

Pada proses enkripsi dan dekripsi program, dilakukan operasi pemangkatan modulo antara elemen, pangkat, dan mod sama seperti pada rumus proses enkripsi dan dekripsi yang telah disebutkan sebelumnya.

Pada proses dekripsi, dilakukan operasi modulo inverse antara e dengan phi(n). Berikut adalah kode program modulo inverse yang dapat digunakan:

```
# return modular inverse of a with m
def modular_inverse(a, m):
    g, x, y = extended_gcd(a, m)

    if g != 1:
        raise Exception('Modular inverse
does not exist')
    else:
        return x % m
```

Dengan implementasi extended_gcd adalah sebagai berikut:

```
# return extended gcd of a and b
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        g, y, x = extended_gcd(b % a, a)
        return g, x - ((b // a) * y), y
```

Algoritma extended_gcd merupakan algoritma yang berdasar pada teorema matematika, yaitu extended euclid, yang digunakan untuk melakukan pemecahan masalah *linear diophantine equation*. Linear diophantine equation secara general dapat dideskripsikan sebagai berikut:

$ax + by = c$

Persamaan diatas dapat dicari solusinya apabila gcd(a,b) habis membagi c. Jika diterapkan pada algoritma RSA, tentunya linear diophantine equationnya adalah ex + dy = 1, sehingga dapat dicari nilai d jika x dan y adalah suatu nilai sembarang yang dapat memenuhi persamaan tersebut.

D. *Chaotic Map*

Chaotic map adalah peta yang memiliki sifat *chaos*, atau dengan kata lain memiliki sifat ketidakteraturan [WIK]. Chaotic map dapat diberikan beberapa parameter yang berupa bilangan diskrit ataupun bilangan kontinu. Chaotic map ini biasanya digunakan untuk proses pembuatan bilangan secara acak. Berikut adalah beberapa contoh chaotic map:

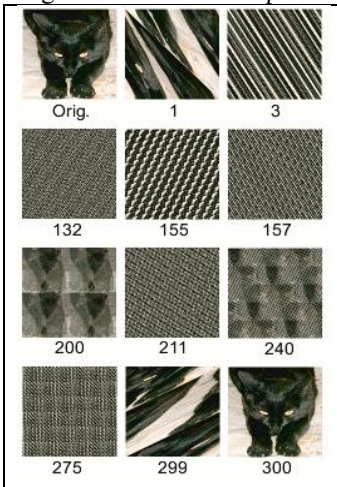
1. Arnold Cat's *Map*

   Arnold Cat's *Map* pada aplikasinya dapat digunakan untuk melakukan proses enkripsi pada gambar yang nantinya setelah dilakukan sebanyak iterasi tertentu, gambar hasil enkripsi pada tahap tersebut akan kembali sama seperti gambar input awal. Arnold Cat's *Map* memiliki persamaan sebagai berikut:

   $T^i(x,y) = T^{i-1}(\mathrm{mod}(2x+y, N), \mathrm{mod}(x+y, N))$

   Dengan T^0 adalah input gambar awal.

   Berikut adalah contoh gambar hasil enkripsi gambar dengan Arnold Cat's *Map*:



   Gambar 1: Arnold's Cat *Map* pada gambar kucing

   Pada gambar tersebut ditunjukkan bahwa setelah dilakukan sebanyak 300 buah iterasi, gambar akan kembali seperti semula. Pada *chaotic map* ini, hasil enkripsi pada 2 buah gambar akan kembali ke bentuk semula dengan beragam jumlah iterasi.

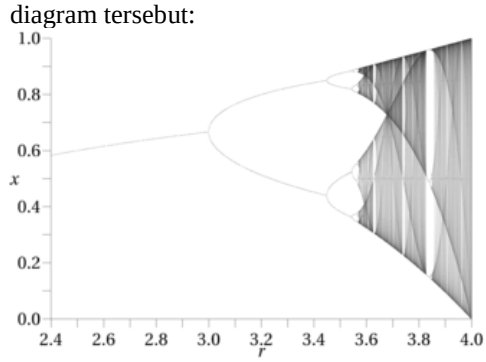
2. *Logistic Map*

   Logistic Map adalah pemetaan fungsi polynomial berderajat 2. Berikut adalah persamaan yang digunakan pada *Logistic Map*.

   $X_{n+1} = rx_n\ (1-x_n) = rx_n - rx^2_n$

   Pada logistic map tersebut digunakan nilai r yang berkisar antara 0 sampai 4. Semakin besar nilai r yang digunakan, maka semakin chaotic pula nilai hasil pemetaan fungsi polynomial tersebut yang dihasilkan.

   Pada logistic map, juga dapat digambarkan bifurcation diagramnya. Berikut adalah gambar bifurcation diagram tersebut:



   Gambar 2: Bifurcation Diagram pada logistic map.

   Pada bifurcation diagram tersebut, dapat terlihat bahwa untuk nilai r yang relative rendah, yaitu dibawah 3.0, nilai x yang dihasilkan setelah beberapa iterasi masih terlihat polanya. Ketika r bernilai 3.0, mulai terjadi bifurcation, yaitu pemecahan 2 buah nilai x , dan begitu pula pada nilai r sekitar 3.45, dan setelah itu, nilai x yang dihasilkan menjadi tidak beraturan dan inilah yang disebut dengan chaotic map.

E. *Henon Map*

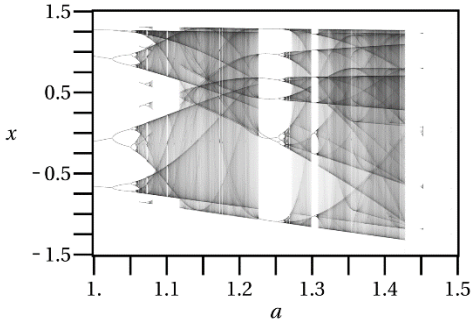
Henon *map* adalah salah satu *chaotic map* yang dapat digunakan untuk proses pembentukan bilangan acak. Pada dasarnya, Henon *map* menggunakan sebuah point (x_n, y_n) dan dapat dipetakan menjadi sebuah point yang baru, yaitu dengan rumus sebagai berikut:

$X_{n+1} = 1 - ax^2_n + y_n$

$Y_{n+2} = bx_n,$

Dengan nilai a dan b adalah sebuah bilangan riil constant sembarang.

Untuk nilai a >=1.0 dan nilai b = 0.3, akan dihasilkan henon map sebagai berikut:



Gambar 3: Gambar henon *map*

Pada gambar tersebut, dapat terlihat bahwa terjadi bifurcation mulai pada nilai a = 1.075.

Namun pada tugas kali ini, penulis akan membuat henon map dengan dekompisisi dimensi 1, yaitu menggunakan sebuah persamaan sebagai berikut:

$$x_n = 1 + b(x_{n-2} - x_{n-3}) + ax^2_{n-2}$$

F. Serangan kriptografi

Serangan kriptografi yang dapat dilakukan adalah sebagai berikut:

1. Serangan *brute force*

Serangan brute force adalah serangan yang dilakukan dengan cara melakukan segala kemungkinan percobaan untuk mendekripsi cipherteks, misalkan dengan cara memasangkan segala kemungkinan ciphertext ke plainteks (substitusi), atau mencoba mencari semua kemungkinan kunci yang ada. Segala kemungkinan ini dapat dilakukan dengan cara membuat program komputer yang dapat melakukan segala percobaan ini.

Kelemahan dari cara ini adalah program yang dijalankan untuk melakukan brute force ini biasanya berlangsung sangat lama karena jumlah kombinasi kunci yang ingin dicari sangatlah banyak, dan belum tentu pula metode brute force yang digunakan tepat.

2. Serangan analisis frekuensi

Serangan analisis frekuensi adalah serangan yang dilakukan dengan memanfaatkan statistik frekuensi kemunculan suatu huruf pada suatu bahasa yang digunakan dalam plainteks. Sebagai contoh, pada Bahasa Inggris, karakter yang paling sering muncul adalah karakter ‘e’, sehingga dilakukan proses perhitungan jumlah karakter yang paling sering muncul pada cipherteks, lalu kriptanalis menduga bahwa karakter yang terenkripsi tersebut dapat diganti dengan karakter ‘e’. Namun serangan ini dapat berjalan apabila teknik enkripsi yang digunakan adalah teknik substitusi.

G. Algoritma Sieve of Eratosthenes

Algoritma Sieve of Eratosthenes adalah algoritma yang dapat digunakan untuk membangkitkan bilangan prima dengan efektif dan efisien. Berikut adalah kode program algoritma Sieve of Eratosthenes dalam bahasa pemrograman C++:

```
void sieve(int n) {
      int i,j;
      isprime[0] = isprime[1] =false;
      for(i=2;i*i<=n;i++){
            if(isprime[i]) {
                  for(j=i*i;j<=n;j+=i){
                  isprime[j]= false;
                  }
            }
      }
}
```

Kemudian, untuk mendapatkan semua semua bilangan prima, cukup melakukan iterasi dari 2 hingga bilangan maksimal, lalu jumlah bilangan prima tersebut dicatat. Berikut adalah kode program pada program utama:

```
int main() {
    memset(isprime,true,sizeof isprime);
      sieve(maxn);
      int cnt = 0;
      ofstream listPrime;
      listPrime.open("listPrime.txt");
      for(int i=2;i<=maxn;i++){
            if(isprime[i]){
                  listPrime<<i<<endl;
                  cnt++;
            }
      }
      listPrime.close();
      return 0;
}
```

Algoritma tersebut pada awalnya menganggap bahwa semua bilangan adalah bilangan prima, kemudian langsung melakukan pernyataan bahwa bilangan 0 dan 1 bukanlah bilangan prima. Kemudian, dari bilangan nomor 2, yang mana merupakan prima, kemudian dari bilangan 4, yang kemudian bertambah terus dengan penambahan 2, dinyatakan sebagai bilangan bukan prima, yaitu bilangan 4,6,8, dan seterusnya. Begitu program dijalankan seterusnya hingga akar kuadrat dari batas atas bilangan saja karena untuk menentukan suatu bilangan prima atau tidak cukup dengan cara melakukan pengecekan hingga akar bilangan tersebut. Kompleksitas dari algoritma ini adalah O(N log (log(N))) dengan N adalah batas atas bilangan yang ingin dicari apakah bilangan tersebut prima.

III. PENJELASAN ALGORITMA

Pada program yang dibuat, dibuat 3 buah algoritma, yaitu algoritma enkripsi dan dekripsi RSA seperti yang telah dijelaskan pada dasar teori. Selain itu, dibuat sebuah algoritma *random number generator* yang dibuat berdasarkan henon *map*, yang memiliki persamaan yang telah disebutkan pada dasar teori pula. Kemudian, dibuat sebuah algoritma untuk melakukan pembentukan daftar bilangan prima dari 2 hingga 50 juta. Daftar bilangan prima ini akan digunakan untuk menentukan nilai p dan q sebagai property dari algoritma RSA.

Secara detail, program akan menerima tiap byte dari file yang digunakan sebagai input, kemudian setiap byte tersebut dilakukan proses enkripsi menggunakan nilai p dan q yang berbeda-beda tergantung dari angka yang dikeluarkan oleh algoritma henon *map*, Sehingga, pada proses pembentukan bilangan acak, dibutuhkan 2 kali panjang daftar byte pesan yang diterima sebagai input (karena terdapat p dan q).

Untuk setiap bilangan bulat yang dibentuk oleh algoritma henon *map*¸ akan dilakukan pemetaan bilangan tersebut menjadi bilangan prima terkait. Sebagai contoh, apabila 2 buah bilangan acak yang dikeluarkan adalah 2 dan 3, maka nilai p dan q pertama yang digunakan untuk melakukan proses enkripsi dengan menggunakan algoritma RSA adalah 3 dan 5 karena 3 adalah bilangan prima ke 2 dan 5 adalah bilangan prima ke 3. Hal ini dilakukan secara terus menerus hingga akhir byte pesan.

Proses pembentukan semua property yang dibutuhkan akan menghasilkan kunci privat dan public, sehingga kunci public yang terdiri dari (e,n) akan disimpan pada sebuah file public.pub yang terdiri dari sejumlah pasang (e,n) sebanyak jumlah byte file input. Begitu pula dengan penyimpanan kunci privat, akan disimpan sejumlah pasang (d,n) sesuai degan jumlah byte file input dan disimpan pada file private.pri.

Proses algoritma dekripsi dilakukan dengan cara mengambil semua kunci privat yang ada pada file private.pri, kemudian dilakukan proses dekripsi sesuai dengan algoritma dekripsi yang telah dijelaskan pada dasar teori.

Berikut adalah pseudocode dari algoritma pembangkit bilangan acak yang menggunakan property dari salah satu *chaotic map*, yaitu Henon *map*.

```
Initiate x with empty array
Input jumlahBilanganPrima
Input x[0] % jumlahBilanganPrima
Input x[1] % jumlahBilanganPrima
Input x[2] % jumlahBilanganPrima
Input a
Input b
Input jumlahElemenArray
Iterate i in range(3, jumlahElemenArray):
   next = 1 + b*(x[i-2] - x[i-3]) +
a*(pow(x[i-2], 2))
   Append next to x

Output x
```

Berikut adalah pseudocode dari algoritma keseluruhan program yang dibuat untuk mengenkripsi pesan:

```
Sieve(5 x 10^7) -> listPrime[1 .. 3001134]

Input(file)

Read(bytes of file)

Henon(countOfBytes)
-> x[1 .. 2*countOfBytes]

Encrypt each byte:
   1. Generate p and q using x[i] and
      x[i+1] -> p = listPrime[x[i]], q =
      listPrime[x[i+1]]
   2. Encrypt -> new number encrypted
   3. Append new number to fileEncrypted
   4. Append (e,n) to public.pub
   5. Append (d,n) to private.pri

Output fileEncrypted
```

Berikut adalah pseudocode dari algoritma untuk melakukan dekripsi pesan:

```
Input(fileEncrypted)

Read(bytes of file)

Accept each (d,n) from private.pri -> list
it in listD -> listD[1..numberOfBytes]

Accept each number from fileEncrypted ->
listEncrypted[1..numberOfBytes]

Decrypt each element:
   1. Decrypted number =
      listEncrypted[i]^listD[i] mod n
   2. Append each byte to fileDecrypted

Output fileDecrypted
```

IV. EKSPERIMEN

Pada eksperimen ini, akan dilakukan percobaan dengan berbagai jenis file input yang akan dilakukan proses enkripsi dan proses dekripsi, kemudian akan dilakukan pemberian waktu enkripsi dan dekripsi, kemudian dilakukan perbandingan dengan algoritma enkripsi RSA standard.

Pada algoritma penulis, digunakan persamaan pada henon map adalah sebagai berikut:

$$x_n = 1 + b(x_{n-2} - x_{n-3}) + ax^2_{n-2}$$

Berdasarkan persamaan tersebut, diperlukan 3 buah variabel awal karena suatu nilai x_n bergantung dari nilai x_{n-3} yang telah dihitung sebelumnya. Maka dari itu, akan dilakukan sebanyak 3 kali percobaan dengan 3 buah nilai variabel yang berbeda.

Pada percobaan pertama, penulis akan melakukan proses enkripsi terhadap sebuah file berextension .txt yang berisi :

```
Hello world
```

Kemudian hasil enkripsi tersebut diletakkan pada file lain. Setelah itu, dilakukan proses dekripsi. Berikut adalah perbandingan waktu enkripsi dan dekripsi yang dilakukan pada file .txt ini. Berikut juga ditunjukkan waktu algoritma enkripsi RSA standard dimana tidak diperlukan berkali-kali melakukan pembentukan nilai p dan q.

Kegiatan	**Waktu (detik)**
Enkripsi	0.004009
Dekripsi	0.011390
Enkripsi RSA standard	0.0010030
Dekripsi RSA standard	0.0010008

Pada percobaan kedua, dilakukan proses enkripsi dan dekripsi pada file berextension .pdf berukuran 9KB. Berikut adalah hasil waktu eksekusinya:

Kegiatan	**Waktu (detik)**
Enkripsi	0.2198851
Dekripsi	0.1415050
Enkripsi RSA standard	0.1939058
Dekripsi RSA standard	0.1372259

Pada percobaan ketiga, dilakukan proses enkripsi pada file berextension .jpg yang berukuran 39 KB. Berikut adalah hasil waktu eksekusinya:

Kegiatan	Waktu (detik)
Enkripsi	0.9944860
Dekripsi	0.6483891
Enkripsi RSA standard	0.9022693
Dekripsi RSA standard	0.6172521

Pada percobaan keempat, dilakukan proses enkripsi dan dekripsi pada sebuah file berekstension .exe berukuran 2648 KB. Berikut adalah hasil waktu eksekusinya:

Kegiatan	Waktu (detik)
Enkripsi	95.662229
Dekripsi	84.549275
Enkripsi RSA standard	90.472948
Dekripsi RSA standard	82.128372

Fakta menarik dari setelah melakukan proses enkripsi ini, ukuran file private.pri menjadi sangat besar, yaitu menjadi 86044 KB. Hal ini disebabkan karena untuk setiap byte yang diproses, dimasukkan nilai private key pada file private.pri.

Dari sisi waktu, dapat disimpulkan bahwa, dengan menggunakan algoritma RSA standard, waktu eksekusi memang lebih cepat dan dari sisi memori yang dihabiskan, jumlah memori yang dihabiskan untuk menyimpan public key dan private key menjadi lebih sedikit dibandingkan dengan algoritma RSA yang termodifikasi ini.

Berikut adalah hasil percobaan dengan menggunakan 3 buah nilai awal yang berbeda untuk proses random number generator dan melakukan enkripsi pada file berextension .pdf. Pertama tama dengan nilai x[0] = 80, x[1] = 47, dan x[2] 35.

Didapatkan hasil enkripsi untuk setiap byte sebagai berikut dalam bentuk integer:

```
19528
4623115
56318883960
53650170043631
116192485993519
67759923661910
28094472260877
18712114543558
435069736364734
90882352262845
319463669746563
261052843542709
196978773372422
46267739961081
188467481577343
871192429153238
23114963342837
272368181657229
243516161087626
….
```

Pada percobaan kedua, nilai awal henon *map* yang digunakan, yaitu x[0] = 96, x[1] = 32, dan x[2] = 20, berikut adalah hasil enkripsi file tersebut:

```
20239
1224339
257058619145
86332781056842
1391594316192410
442552980872444
145560281680426
372532728068975
366569031253138
58003940921552
5030482204438
65644284117086
297067753295325
123254220055120
129587011061550
22230921733433
50076830697995
1031270829377649
1035532861817985
160565896012035
…
```

Pada percobaan ketiga, nilai awal henon map, yaitu x[0] = 39, x[1] = 81, dan x[2] = 32, berikut adalah hasil enkripsi file tersebut:

```
12918
26778471
454635668535
216947843869820
335947060281410
172227386809128
168526225876455
367418161819490
100583123078921
11819415229452
414299941739871
25142991565002
242074518484628
226148471490362
629449021836887
942783423182774
151155221566023
17907693529899
…
```

Dapat dilihat pada hasil enkripsi diatas, bahwa nilai hasil enkripsi yang dihasilkan untuk 3 buah percobaan berbeda total. Hal ini menunjukkan bahwa dengan metode ini, dapat membuat kriptanalis bingung berapa nilai yang diperlukan untuk menentukan nilai awal dari x, dan memerlukan program untuk mencari semua kemungkinan x dari semua bilangan bulat positif, tentu saja jika kriptanalis mengetahui metode yang digunakan untuk memecahkan permasalahan ini.

Tentunya, nilai a dan b yang digunakan dalam persamaan henon map tersebut dapat diubah sesuai keinginan pengenkripsi pesan dengan nilai bilangan bulat positif sembarang, sehingga membuat jumlah kemungkinan hasil enkripsi semakin banyak.

Setelah dilakukan proses perhitungan frekuensi pada hasil enkripsi, ternyata nilai frekuensi yang dihasilkan rata-rata

menghasilkan 1, atau dengan kata lain, tiap angka yang dihasilkan pada hasil enkripsi rata-rata hanya muncul sekali.

V. ANALISIS

Secara umum, algoritma modifikasi RSA ini merupakan sebuah algoritma yang menurut penulis cukup aman dari sisi serangan, namun memiliki kekurangan dari sisi memori, karena data yang digunakan untuk melakukan proses enkripsi dan dekripsi, diperlukan penyimpanan file private.pri dan public.pub yang mana isinya adalah nilai kunci public dan privat yang digunakan untuk melakukan proses dekripsi sebanyak jumlah byte pesan yang diterima. Jika jumlah file byte pesan input sangat besar, algoritma ini menjadi tidak efisien dari sisi memori.

Berikut adalah analisis dari berbagai serangan kriptografi terhadap hasil enkripsi algoritma ini:

A. Serangan brute force

Jika seorang kriptanalis ingin melakukan pemecahan cipherteks ini dengan menggunakan metode brute force, maka metode bruteforce tersebut dapat dilakukan pada bagian RSA atau bagian pembangkit bilangan acaknya.

Pada bagian RSA, kriptanalis akan melakukan pencarian lengkap bilangan p dan q dari 2 hingga 50 juta yang mana sebanyak 3001134 bilangan prima di antara 2 buah bilangan tersebut, dengan asumsi rata-rata jumlah byte pesan yang akan dienkripsi adalah 30 bytes, maka jumlah operasi yang dibutuhkan untuk melakukan pencarian tersebut adalah sebanyak 208238779864687836732658862948314892334145346285236722823440602789492207249587172070525068441208364833628698394960335947269587292780364456305345387371562931459249728804113507199506547173884952576 = 2,08 x 10^{194} operasi, dan apabila dalam 1 detik, komputer dapat melakukan operasi sebanyak 10^8 operasi, maka banyak waktu yang dibutuhkan untuk melakukan komputasi adalah 660320839246219675078192741464722515011876415161202190586759902300520697772663533962852195716667823546514137477677371725233343774671373846731815662644479107874333234411826189749 8 = 6,60 x 10^{177} tahun yang mana tentunya waktu tersebut ialah sangat lama.

Sedangkan apabila penyerangan dilakukan dari sisi menebak 3 nilai pertama dari algoritma pembangkit bilangan acak tersebut, dengan asumsi bahwa 3 nilai pertama berada pada kisaran nilai antara 1 hingga 10^{20}, maka terdapat sebanyak 10^{60} operasi yang dibutuhkan untuk menentukan 3 bilangan pertama, yang bila dilakukan operasi perhitungan menggunakan komputer dengan kecepatan perhitungan yang sama seperti diatas, dibutuhkan waktu sebanyak 3170979198376458650431253170979198376458650 43 = 3,17 x 10^{44} tahun. Kriptanalis akan memilih untuk melakukan perhitungan dengan mencari 3 kemungkinan bilangan awal tersebut karena waktu yang dibutuhkan lebih singkat dibandingkan dengan perhitungan jumlah kombinasi p dan q pada RSA, namun dengan waktu tersebut, waktu tersebut tentulah sangat lama.

B. Serangan analisis frekuensi

Telah disebutkan pada hasil eksperimen, bahwa rata-rata frekuensi jumlah kemunculan setiap angka pada hasil enkripsi adalah 1, sehingga serangan analisis frekuensi tidaklah mungkin dapat dilakukan oleh kriptanalis untuk memecahkan cipherteks ini.

Selain itu, dari hasil analisis frekuensi kemunculan karakter tersebut membuktikan bahwa algoritma ini memiliki prinsip penyandian Shannon, yaitu confussion, karena algoritma ini tidak memiliki hubungan apapun dari kunci public yang dimasukkan pada algoritma RSA dengan hasil enkripsi pada cipherteks yang dihasilkan.

VI. KESIMPULAN DAN SARAN

Algoritma RSA yang digabungkan dengan algoritma pembentuk bilangan acak yang memanfaatkan henon *map* ini merupakan algoritma yang aman dari serangan kriptanalis berdasarkan hasil analisis serangan brute force dan serangan analisis frekuensi. Selain itu, dari sisi waktu, waktu eksekusi algoritma RSA standard memang relatif lebih kecil dibandingkan dengan waktu eksekusi dari algoritma RSA yang telah dimodifikasi, namun selisih waktunya pun tidak terlalu banyak, sehingga baik digunakan untuk melakukan proses enkripsi file / pesan secara lebih aman.

Namun kekurangan dari algoritma yang telah dimodifikasi ini adalah dari sisi memori. Memori yang dibutuhkan dari algoritma ini menjadi sangat besar digunakan untuk menyimpan private key dan public key dalam 2 buah file yang berbeda dan ukuran 2 buah file tersebut sebanding dengan jumlah byte pesan yang menjadi input untuk kemudian dilakukan proses enkripsi.

Penulis memiliki saran bagi

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa, karena atas bantuan, rahmat, dan berkat-Nya, makalah ini dapat selesai pada waktunya. Tak lupa juga, penulis ingin menyampaikan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir selaku dosen mata kuliah IF4020 Kriptografi yang telah membagikan ilmunya kepada penulis. Selain itu, penulis juga ingin menyampaikan terima kasih kepada kedua orang tua penulis yang selalu mendukung penulis dalam bidang akademis yang dijalani.

REFERENSI

[1] Algoritma-Kriptografi-Modern-(2018). 2018. Munir, Rinaldi. ITB
[2] Algoritma-Kriptografi-Klasik-bag2(2018). 2018. Munir, Rinaldi. ITB
[3] Algoritma-RSA-(2018). 2018. Munir, Rinaldi. ITB
[4] Serangan-Terhadap-Kriptografi-(2018). 2018. Munir, Rinaldi. ITB
[5] Halim, Steven, Halim, Felix. Competitive Programming 3. 2013. NUS
[6] https://en.wikipedia.org/wiki/Logistic_map, diakses pada tanggal 18 Mei 2018, pukul 10.56.
[7] https://en.wikipedia.org/wiki/Arnold%27s_cat_map, diakses pada tanggal 18 Mei 2018, pukul 10.44.
[8] https://en.wikipedia.org/wiki/H%C3%A9non_map, diakses pada tanggal 18 Mei 2018, pukul 10.52.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2018

Dicky Novanto / 13515134