Metode Penyerangan Terhadap Algoritma RSA dengan Membuat Algoritma RSA Menjadi Algoritma Kriptografi Kunci Simetri

Nathaniel Angelius Tjahyadi / 18219070
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nthnlius@gmail.com

***Abstrak*— Keamanan kriptografi kunci asimetri RSA sering disebut-sebut dengan kriptografi dengan keamanan berbanding eksponensial dengan panjang kunci. Selain itu, algoritma RSA memiliki dua buah kunci sehingga mempersulit untuk melakukan *cracking*. Kedua hal ini menyebabkan tingginya tingkat keamanan algoritma RSA. Dua hal ini menjadi dasar dari keamanan yang ditawarkan dalam algoritma RSA.**

Keywords—component; formatting; style; styling; insert (key words)

***Abstract*—This electronic document is a "live" template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. **CRITICAL: Do Not Use Symbols, Special Characters, or Math in Paper Title or Abstract.* (*Abstract*)**

Keywords—component; formatting; style; styling; insert (key words)

I. PENDAHULUAN

Informasi merupakan aset berharga dalam kehidupan sehari-hari terutama dalam hal bersaing. Dalam persaingan tak jarang ditemukan keinginan saling sadap sehingga perlu ada metode untuk pengamanan informasi. Hal tersebut adalah kriptografi.

Kriptografi adalah salah satu teknik untuk menyembunyikan maksud dari sebuah pesan menjadi sebuah rangkaian huruf-huruf atau angka yang tidak masuk akal. Kriptografi yang sederhana seperti *Caesar Cipher* sudah menjadi hal yang umum diketahui oleh banyak orang. Seiring berjalannya waktu, metode-metode kriptografi klasik menjadi mudah dipecahkan karena berkembangnya pola pikir manusia. Oleh karena itu diciptakan lah sebuah algoritma bernama RSA yang didasarkan dengan teori yang dimiliki oleh Diffie-Helman yang menjelaskan mengenai sistem kriptografi dengan kunci yang tidak simetrik[1].

RSA adalah suatu kriptosistem asimetrik atau kriptosistem kunci publik. RSA ditemukan oleh Don Rivest, Adi Shamir, dan Leonard Adleman dan dipatenkan pada tanggal 20 September 1983 dan dipublikasikan pada 6 September 2000 [2]. Kesulitan dalam memecahkan algoritma RSA terletak pada kesulitan memfaktorkan satu buah bilangan bulat yang meningkat seiring bertambahnya panjang digit dari angka tersebut. Selain kesulitan dalam memfaktorkan satu buah bilangan bulat, algoritma RSA sulit dipecahkan karena memiliki dua buah kunci yang digunakan untuk mengenkripsi dan mendekripsi pesan. Algoritma kriptografi yang memiliki dua buah kunci sering disebut dengan kriptografi kunci publik.

II. DASAR TEORI

A. *Algoritma Kunci Publik*

Algoritma kunci publik adalah algoritma kriptografi yang menggunakan dua buah kunci untuk melakukan enkripsi dan dekripsi. Dua kunci tersebut adalah kunci publik dan kunci privat. Kunci publik umumnya digunakan untuk mengenkripsi pesan dan mengautentikasi tanda tangan digital atau sertifikat digital, sedangkan kunci privat umumnya digunakan untuk mendekripsi pesan dan membubuhkan tanda tangan digital atau memberikan sertifikat digital.

Teori algoritma kunci publik pertama kali dikenalkan oleh Whitfield Diffie dan Martin E. Hellman di IEEE pada tahun 1976. Keduanya adalah seorang ilmuwan dari Stanford University. Penemuan ini menyebabkan perubahan besar pada dunia kriptografi karena teori ini memiliki dua keuntungan yang sangat besar dibandingkan dengan kriptografi kunci simetri yaitu :

1. Setiap orang hanya perlu memiliki dua buah kunci (kunci publik dan kunci privat). Kunci public orang lain dapat dilihat dari repositori publik.
2. Tidak perlu mengirim kunci privat.

B. *Algoritma Kunci Simetri*

Algoritma kunci simetri adalah algoritma kriptografi yang menggunakan satu buah kunci untuk melakukan enkripsi dan dekripsi. Algoritma kunci simetri lebih sering digunakan dalam pengiriman pesan karena memiliki nilai komputasi yang lebih

rendah, namun kunci akan dikomunikasikan menggunakan algoritma kunci publik. Salah satu algoritma kunci simetri yang sangat sering digunakan adalah Data Encryption Algorithm (DEA) berbasis *Data Encryption Standard* (DES) dan *Advanced Encryption Algorithm* berbasis *Advanced Encryption Standard* (AES) milik Rijndael.

Algoritma kunci simetri berpangku operasi utama berupa operasi-operasi sederhana yang berjalan secara *bit-wise* seperti XOR, AND, dan lain sejenisnya.

C. Dictionary

Dictionary adalah salah satu tipe data yang sering dikenal juga dengan *assosiative array*. Sebuah *dictionary* berisi kumpulan pasangan kunci dan nilai. Setiap kunci akan memetakan ke nilai pasangannya [3]. Tipe data ini merupakan terapan dari penggunaan hash-map.

D. Algoritma RSA

Algoritma RSA merupakan salah satu algoritma kriptografi kunci publik yang keamanannya bergantung dengan kesulitan dalam memfaktorkan sebuah bilangan bulat menjadi dua buah bilangan prima. Berikut adalah komponen penting dalam algoritma RSA .

1. p dan q bilangan prima (rahasia)
2. n = p*q (tidak rahasia)
3. $\Phi(n) = (p-1)*(q-1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia)
   Syarat : $\gcd(e, \Phi(n)) = 1$
5. d (kunci dekripsi) (rahasia)
   Syarat : $e*d \equiv 1 \bmod n$
6. M (plainteks) (rahasia)
7. C (Cipherteks) (tidak rahasia)

Dari komponen-komponen RSA di atas, dapat dirumuskan sebuah pasangan kunci privat dan public sebagai berikut :

- Kunci publik : (e, n)
- Kunci privat : (d, n)

Cara pengenkripsian dapat dilakukan dengan langkah-langkah berikut :

1. Sebuah plainteks M dapat dipecah menjadi (m_1, m_2, m_3, m_4, ….)
2. Hitunglah nilai cipherteks dari setiap karakter dengan persamaan berikut :

$$c_n = m_n^e \bmod n$$

3. Gabungkan kembali setiap karakter dari cipherteks (c_1, c_2, c_3, c_4, ….) menjadi sebuah cipherteks utuh C.

Cara mendekripsi dapat dilakukan dengan 1angkah-langkah berikut :

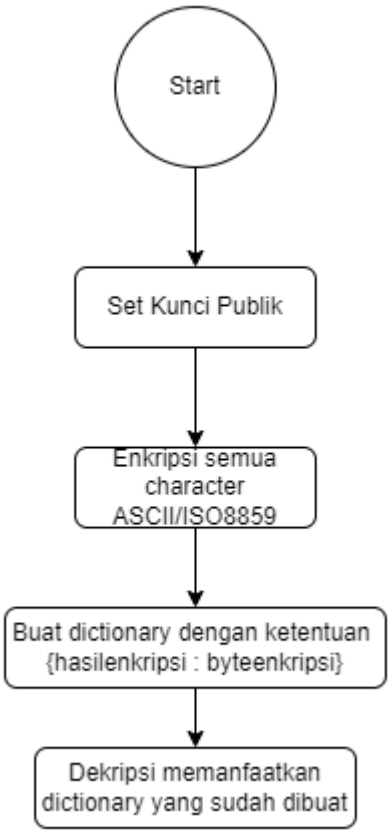
1. Sebuah cipherteks C dapat dipecah menjadi (c_1, c_2, c_3, c_4, ….)
2. Hitunglah nilai cipherteks dari setiap karakter dengan persamaan berikut :

$$m_n = c_n^d \bmod n$$

3. Gabungkan kembali setiap karakter dari plainteks (m_1, m_2, m_3, m_4, ….) menjadi sebuah plainteks utuh M.

III. Skema Penyerangan

Seperti yang sudah dijelaskan sebelumnya, algoritma RSA adalah algoritma kunci publik yang memiliki dua buah kunci yaitu kunci publik dan kunci privat. Penyerangan dilakukan dengan mengubah algoritma RSA menjadi algoritma kunci simetri dengan kunci yang digunakan adalah kunci publik. Metode penyerangan ini akan memanfaatkan fitur *hash-map* yang telah disediakan dalam berbagai Bahasa pemrogramman. Dengan memanfaatkan *hash-map*, metode penyerangan tidak harus melakukan *brute force* terhadap algoritma RSA. Untuk membantu memahami metode penyerangan, berikut adalah diagram alir yang menggambarkan skema penyerangan.



Lalu pada poin-poin berikut ini akan dijelaskan setiap tahapan proses yang ada pada diagram alir di atas .

A. Set Kunci Publik

Seperti yang sudah dijelaskan sebelumnya, penyerangan ini akan mengeksploitasi kunci publik dan *hash-map*. Kunci

publik sendiri adalah kunci yang digunakan untuk melakukan enkripsi pesan pada algoritma RSA. Berikut adalah contoh potongan kode yang akan digunakan.

```
class reverseRSA :
    def __init__(self):
        with open ("RSA.pub", 'r') as
jsonfile :
            pubkey = json.load(jsonfile)
            self.e = jsonfile['e']
            self.n = jsonfile['n'] ...
```

B. Enkripsi Semua Karakter ASCII atau ISO8859

Pada dasarnya tidak diharuskan ASCII atau ISO8859 karena yang seharusnya dienkripsi adalah byte-byte yang mungkin muncul yaitu byte dengan nilai 0 hingga 255. Namun untuk mempermudah algoritma digunakan. Berikut adalah contoh potongan kode yang akan digunakan dengan mengenkripsi semua karakter ASCII yang *printable*

```
...#masih dalam bagian __init__
        self.tupple={0 : 1}
        charascii = string.printable
        charasc = bytearray(charascii,
encoding = "ASCII")
        euy = self.encrypt(charasc)
...
```

Pada tabel ASCII, karakter dengan representasi angka 0 hingga 32 tidak dapat dicetak, maka kode “self.tupple = {0 : 1} tidak perlu dipikirkan karena tidak akan mungkin karakter yang dapat dicetak akan mengandung karakter ASCII dengan orde byte 1.

C. Buat *Dictionary* atau *HashMap*

Seperti sudah dijelaskan sebelumnya, metode ini mengeksploitasi penggunaan HashMap yang tersedia pada banyak bahasa pemrogramman. Pada Bahasa Pemrogramman Python , fitur *HashMap* dinamakan dengan *Dictionary*. Tujuan penggunaan *HashMap* ini untuk menurunkan kompleksitas waktu karena tidak perlu melakukan *sequential search* pada suatu *list* untuk mencari hasil enkripsi pesan. Berikut contoh potongan kode yang digunakan

```
...#masih dalam pemanggilan Konstruktor
        for i in range (len(charascii)):
            self.tupple[euy[i]] =
charascii[i]
...
```

D. Dekripsi dengan memanfaatkan *dictionary* atau *HashMap* yang telah dibuat.

Seperti yang sudah dijelaskan sebelumnya, *HashMap* mempermudah pencarian pasangan antara byte asli dengan hasil enkripsi. Dengan menggunakan hasil enkripsi sebagai kunci, dan byte asli sebagai *value* dari kuncinya, pendekripsian tidak lagi harus melakukan *sequential search*. Berikut adalah contoh kode algoritma untuk mendekripsi suatu cipherteks dengan menggunakan fitur *HashMap* pada Bahasa pemrogramman Python.

```
def decrypt(self, ciphermsg : bytearray):
        ret = []
        for i in range(len(ciphermsg)):

ret.append(self.tupple[int(ciphermsg[i])])
        return ret
```

IV. Pengujian Penyerangan

Dalam melakukan pengujian, perlu ada penggunaan algoritma RSA asli sebagai pembanding kebenaran jawaban. Oleh karena itu, pada bagian Appendix nanti akan ditambahkan kode algoritma RSA yang pendekripsian menggunakan kunci privat, dan aka nada juga kode algoritma RSA yang pendekripsiannya menggunakan eksploitasi *HashMap*. Berikut adalah contoh kode yang digunakan untuk menjadi pengujian.

```
revrsa = reverseRSA()
plainteks = "Ini adalah contoh kalimat"
msg = bytearray(plainteks, encoding =
"ASCII")
cipherteks = revrsa.encrypt(msg)
result = revrsa.decrypt(cipherteks)
print (result)
```

Keluaran yang diharapkan adalah isi dari variabel “result” akan sama dengan isi dari variabel “plainteks”. Berikut adalah hasil keluaran dari potongan kode tersebut.

```
PS D:\tes\kripto\makalah> py .\reversersa.py
['I', 'n', 'i', ' ', 'a', 'd', 'a', 'l', 'a', 'h', ' ', 'c',
'o', 'n', 't', 'o', 'h', ' ', 'k', 'a', 'l', 'i', 'm', 'a', '
t']
```

References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] W. Diffie and M.E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 22:644–654, 1976.)

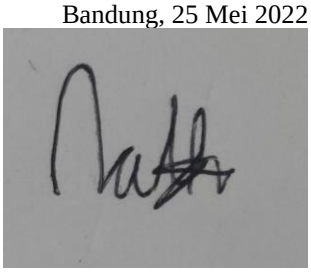
[2] RSA Cryptography : History and uses. (May 26) Telsy https://www.telsy.com/rsa-encryption-cryptography-history-and-uses

[3] Mealus, P. (n.d.). *What is a dictionary in python?* Real Python. Retrieved May 25, 2022, from https://realpython.com/lessons/dictionary-python

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, 25 Mei 2022



Nathaniel Angelius Tjahyadi (18219070)

A. APPENDIX

Kode File reversersa.py

```
import json
from math import gcd
import sympy
import string
class reverseRSA :
    def __init__(self):
        with open ("RSA.pub", 'r') as
jsonfile :
            pubkey = json.load(jsonfile)
            self.e = pubkey['e']
            self.n = pubkey['n']
        self.tupple={0 : 1}
        charascii = string.printable
        charasc = bytearray(charascii,
encoding = "ASCII")
        euy = self.encrypt(charasc)
        for i in range (len(charascii)):
            self.tupple[euy[i]] =
charascii[i]
        # print (self.tupple)

    def generateE(self):
        randnum = sympy.randprime(2**255,
2**256-1)
        while (gcd(self.tot, randnum)!= 1):
            randnum =
sympy.randprime(2**255, 2**256-1)
        self.e = randnum
    def encrypt(self, message:bytearray):
        ciphertext = []

        for byt in message :
            ciphermsg = pow(byt, self.e,
self.n)
            ciphertext.append(ciphermsg)
        return ciphertext
    def decrypt(self, ciphermsg :
bytearray):
        ret = []
        for i in range(len(ciphermsg)):

ret.append(self.tupple[int(ciphermsg[i])])
        return ret

revrsa = reverseRSA()
plainteks = "Ini adalah contoh kalimat"
msg = bytearray(plainteks, encoding =
"ASCII")
cipherteks = revrsa.encrypt(msg)
result = revrsa.decrypt(cipherteks)
print (result)
```

Kode file rsa.py yang digunakan sebagai pembanding

```
from math import ceil, sqrt, gcd
import sympy
import json
def isPrime(rndint) -> bool:
    if rndint>=10 :
        for i in range (2,
ceil(sqrt(rndint))+1):
            if (rndint%i == 0):
                return False
```

```
        return True
    elif rndint >0 and rndint<10 :
        for i in range (2, rndint):
            if (rndint%i == 0):
                return False
        return True
    else :
        return isPrime((rndint)*(-1))
class RSA:
    def __init__(self):
        try :
            with open("RSA.pub", "r")as r:
                pubkey = json.load(r)
                self.e = pubkey['e']
                self.n = pubkey['n']
            with open("RSA.pri", "r")as r:
                prikey = json.load(r)
                self.d = prikey['d']
        except FileNotFoundError :
            p = sympy.randprime(2**63,
2**64-1)
            q = sympy.randprime(2**63,
2**64-1)
            while p==q :
                q = sympy.randprime(2**63,
2**64-1)
            self.n = p*q
            self.tot = (p-1)*(q-1)
            self.generateE()
            self.generateD()
            tup = {'d':self.d, 'n':self.n}
            with open("RSA.pri" , "w")as f:
                json.dump(tup, f)
            tup = {'e':self.e, 'n':self.n}
            with open ("RSA.pub", "w")as f:
                json.dump(tup, f)

    def isEallowed(self, e)-> bool:
        if (isPrime(self.p) and
isPrime(self.q)):
            return gcd(e, self.tot) == 1
    def generateE(self)->None:
        randnum = sympy.randprime(2**63,
2**64-1)
        while (gcd(randnum, self.tot)!=1):
            randnum =
```

```
sympy.randprime(2**63, 2**64-1)
        self.e = randnum
    ''' rumus : d = (1+k*tot)/e'''
    def setE(self,num):
        if (self.isEallowed(num)):
            self.e=num
            print ("E has been set")
        else :
            print("You have chosen the
wrong E")
    def genKey (self):
        p = sympy.randprime(2**63, 2**64-1)
        q = sympy.randprime(2**63, 2**64-1)
        while p==q :
            q = sympy.randprime(2**63,
2**64-1)
        self.n = p*q
        self.tot = (p-1)*(q-1)
        self.generateE()
        self.generateD()
        tup = {'d':self.d, 'n':self.n}
        with open("RSA-NN.pri" , "w")as f:
            json.dump(tup, f)
        tup = {'e':self.e, 'n':self.n}
        with open ("RSA-NN.pub", "w")as f:
            json.dump(tup, f)
    def generateD(self)-> int:
        d = pow(self.e, -1, self.tot)
        print("d has been set to : ", d)
        self.d = d
        return d
    def encrypt(self, message:bytearray):
        ciphertext = []

        for byt in message :
            ciphermsg = pow(byt, self.e,
self.n)
            ciphertext.append(ciphermsg)
    def decrypt (self, ciphermsg :
bytearray):
        plaintext = []
        plaintxt = ''
        for i in range (0, len(ciphermsg),
16):
            print ("i :", i)
            ciphertext =
```

```
bytearray(ciphermsg)
            text9 =
int.from_bytes(ciphertext[i:i+16],
byteorder="big", signed=False)
            plainmsg = pow(text9, self.d,
self.n)
            plaintext.append(plainmsg)
            plaintxt+=chr(plainmsg)
        # print (plaintxt)

        return (plaintxt)

def nextPrime(x):
    nextprime = x + 1
    while (not isPrime(nextprime)):
        print(nextprime, "is not a prime
number")
        nextprime +=1
    print(nextprime, "is prime number")
    return nextprime

def isEven(x : int)->bool :
    return x%2==0
def exp2(a:int , b:int)-> int :
    if b == 0:
        return 1
    else :
        x = exp2(a, b//2)
        if (isEven(b)):
            return x*x
        else :
            return x*x*a
```

Berikut adalah kunci publik dan kunci privat yang digunakan

- Kunci Publik

```
{"e":
71586788970420994835263766948279608417230120155468462850536399742000647450363,
"n": 83529802439348652543304697830860333654301138814169000049287756954815612438966607542330752426554386077205749183743847965251896553709024471972290290838775 01}
```

- Kunci Privat

```
{"d": 74335037690011426456846691846691300978610164372437528479632551037472063951862133004515721321544943153283940026026760494828978874964671224869563162679765 27,
 "n": 83529802439348652543304697830860333654301138814169000049287756954815612438966607542330752426554386077205749183743847965251896553709024471972290290838775 01}
```