BAB III

METODOLOGI PENELITIAN

3.1 Analisis Masalah

Rentan nya keamanan pesan pada aplikasi *instant messenger* khususnya pada kasus *eavesdroppingo* atau penyadapan yang menempati urutan pertama dalam kasus penyerangan sehingga diperlukan suatu mekanisme enkripsi data yang dapat mengamankan pesan diantara sesama penggunanya.

Algoritma RSA yang merupakan salah satu algoritma kriptografi menawarkan mekanisme pengamanan data dengan menggunakan kunci yang berbeda, yaitu kunci publik (untuk pengirim) dan kunci privat (untuk penerima).

Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan besar menjadi bilangan faktor-faktor prima. Pemfaktoran dilakukan untuk memperoleh kunci privat. Selama pemfaktoran bilangan besar menjadi faktor-faktor prima belum ditemukan, maka selama itu pula keamanan algoritma RSA tetap terjamin.

3.2 Analisis Aplikasi *Instant Messenger*

3.2.1 Algoritma RSA

3.2.1.1 Perumusan Algoritma RSA

Rumus algoritma RSA didasarkan pada teorema Euler yang menyatakan bahwa :

1. $a^{\Phi(n)}=1 \pmod n$ (1)

   yang dalam hal ini :

   1. *a* harus relatif prima terhadap *n*
   2. $\Phi(n) = n(1-1/p1)(1-1/p2) \ldots (1-1/pr)$, yang dalam hal ini *p1*, *p2*, ... *pr* adalah faktor prima dari *n*.

2. $\Phi(n)$ adalah fungsi yang menentukan berapa banyak dari bilangan-bilangan 1,2,3, ..., *n* yang relatif prima terhadap *n*.
3. Berdasarkan sifat $a^k=b^k \pmod n$ untuk *k* bilangan bulat ≥ 1, maka persamaan (1) dapat ditulis menjadi :

   $a^{k\,\Phi(n)=}1^k \pmod n$

   atau

   $a^{k\,\Phi(n)=}1 \pmod n$ (2)

4. Bila *a* diganti dengan *m* , maka persamaan (2) menjadi

   $m^{k\,\Phi(n)}=1 \pmod n$ (3)

5. Berdasarkan sifat $ac = bc \pmod n$, maka bila persamaan (3) dikali *m* hasil nya menjadi :

   $m^{k\,\Phi(n)+1}=m \pmod n$ (4)

   yang dalam hal ini *m* relatif terhadap *n*.

6. Misalkan *e* dan *d* dipilih sedemikian sehingga

   $e.d = 1 \pmod{\Phi(n)}$ (5)

   atau

   $e.d = k\,\Phi(n)+1$ (6)

7. Sulihkan persamaan (6) ke dalam persamaan (4) menjadi :

   $m^{e.d} = m(mod\ n)$ (7)

8. Persamaan (7) dapat ditulis kembali menjadi :

   $(m^{e})^{d} = m(mod\ n)$ (8)

   Yang artinya perpangkatan *m* dengan *e* diikuti dengan perpangkatan dengan *d* menghasilkan *m* semula.

9. Berdasarkan persamaan (8), maka enkripsi dan dekripsi dirumuskan sebagai berikut :

   $E_e(m) = c = m^e\ mod\ n$ (9)

   $D_d(c) = m = c^d\ mod\ n$ (10)

10. Karena *e.d = d.e* maka enkripsi diikuti dengan dekripsi sama dengan dekripsi diikuti dengan enkripsi :

    $D_d(E_e(m)) = E_e(D_d(m)) = m^d\ mod\ n$ (11)

11. Oleh karena $m^d mod\ n = (m+kn)^d\ mod\ n$ untuk sembarang bilangan bulat *k*, maka tiap plainteks *m,m+n,m+2n,...* menghasilkan cipherteks yang sama. Dengan kata lain, transformasinya dari banyak ke satu.

    Agar transformasinya satu ke satu, maka *m* harus dibatasi dalam himpunan {0,1,2,..., *n*-1} sehingga enkripsi dan dekripsi tetap benar seperti pada persamaan (9) dan persamaan (10).

3.2.1.2 Pseudocode Algoritma RSA

Sebelum dapat diterapkan dalam bahasa pemrograman, algoritma terlebih dahulu dibentuk ke dalam *pseudocode* nya. Hal ini untuk mempermudah dalam proses implementasi ke bahasa pemrograman lain karena *pseudocode* adalah tahapan alur logika yang bebas dari sintaks bahasa pemrograman. Berikut adalah *pseudocode* algoritma RSA :

```
get p                                                          (1)
get q                                                          (2)
calculate n =p-q                                               (3)
calculate m=(p-1)(q-1)                                         (4)
//get e value
get e
do
    if (e==0)
        break;
    r=e mod m
    e=m
    e=r
    if(r==1)
        return false;
while (true);                                                  (5)
//get d value
get i=1;
do {
d=1+k(m)/e;
i++;
}while(d != instanceof double); // end d if d have  int value  (6)
// enkripsi
get str; // message
int[]temp; // for ciphertext storeage
char[]arrChar=str.toCharArray();
for(char c : arrChar) {
    temp[i]=(int)c^e mod n;
}                                                              (7)
//dekripsi
int[]temp2; // for ciphertext to plaintext
for (int i=0;i<temp.length){
    temp2[i]=temp[i]^d mod n;
```

} (8)

Keterangan dari *pseudocode* Algoritma RSA di atas :

1. Mendapatkan nilai *p* prima secara acak.
2. Mendapatkan nilai *q* prima secara acak.
3. Mendapatkan nilai *n* sesuai dengan nilai *p* dan nilai *q*.
4. Mendapatkan nilai *m* sesuai dengan nilai *p* dan nilai *q*.
5. Mendapatkan nilai *e* yang relatif prima terhadap m dan kondisi berakhir jika variabel *r* bernilai 1.
6. Sesudah mendapatkan nilai *e* kemudian melakukan suatu proses untuk mendapatkan nilai *d*. Kondisi berakhir ketika nilai *d* bernilai bulat atau *int*.
7. Kunci publik dan kunci privat sudah ditemukan dengan pasangan nilai d,e dan n.

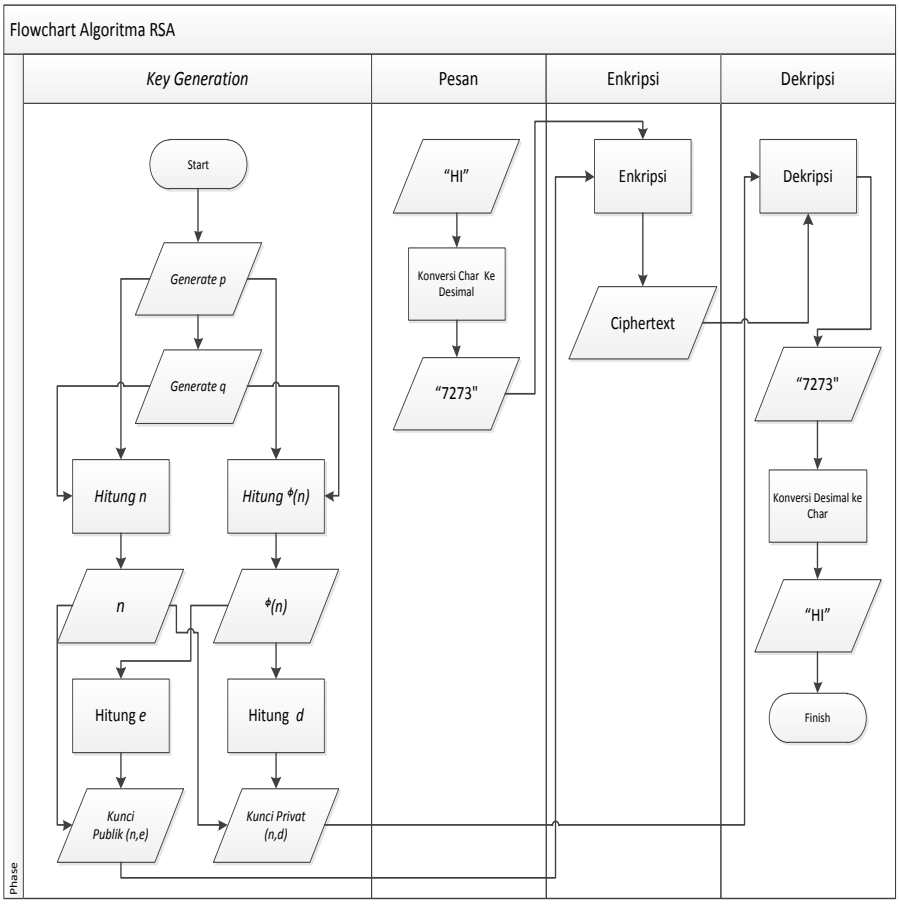
   Kunci publik (n,e)

   Kunci privat (n,d)
8. Melakukan proses enkripsi. Pesan yang diinputkan kemudian dirubah ke bentuk ASCII desimal di tampung di variabel array *c*. Kemudian dari setiap nilai dari variabel temp tersebut di lakukan proses enkripsi dan hasilnya disimpan di variabel array *temp*.
9. Nilai yang ada di variabel *temp* kemudian dilakukan proses dekripsi dan hasilnya disimpan di variabel array *temp2*.

Variabel *temp2* inilah bentuk *plaintext* yang didapatkan kembali setelah dilakukan proses dekripsi.

3.2.1.3 Flowchart Algoritma RSA

Untuk dapat memahami alur algoritma RSA tidak hanya melalui notasi algoritmik (*pseudocode*) tetapi juga dapat juga menggunakan pendekatan visual seperti *Flowchart* atau Diagram Alur. Adapun *flowchart* algoritma RSA yaitu :



Gambar 3.1

Flowchart Algoritma RSA

Keterangan :

1. Diawali dengan mendapatkan nilai prima p secara acak
2. Kemudian mendapatkan nilai q prima secara acak
3. Mencari nilai n dengan melakukan proses perkalian antara nilai p dengan q
4. Sesudah nilai n ditemukan kemudian langkah selanjutnya yaitu mencari nilai *Φ(n)*

5. Mendapatkan nilai prima e relatif prima terhadap nilai *Φ(n)* dengan hasil =1.
6. Dengan menggunakan rumus, kemudian mencari nilai bulat d. Pada proses mencari niali d, pasti banyak menghasilkan nilai yang desimal sehingga perlu di cobakan terus menerus sampai nilai d tersebut benar-benar bulat dan tidak ada bilangan di belakang koma.
7. Kunci publik dan kunci privat sudah didapat .

   Kunci publik (n,e)

   Kunci privat (n,d)
8. Asumsikan pesan yang diinputkan untuk dienkripsi yaitu "HI"
9. Kemudian dari bentuk karakter tersebut diubah ke bentuk desimal sehingga menghasilkan bilangan desimal 7273
10. Dari bentuk desimal kemudian dilakukan proses enkripsi dengan menggunakan persamaan enkripsi dengan menggunakan kunci publik yang sudah dibentuk tadi. Hasilnya berupa *ciphertext* yang nantinya tidak bisa dipahami oleh pihak ketiga.
11. *Ciphertext* tersebut kemudian didekripsi dengan menggunakan persamaan dekripsi dan kunci privat yang telah dibentuk sebelumnya hasilnya yaitu berupa plaintext yang berbentuk desimal.

12. Untuk dapat dimengerti oleh pengguna *plaintext* desimal dirubah kebentuk *plaintext* karakter.

3.2.1.4 Cara Kerja Algoritma RSA

Karena algoritma RSA adalah salah satu dari jenis algoritma asimetris, maka terdapat tiga proses penting dalam menjalankan algoritma ini. Adapun proses nya yaitu:

3.2.1.4.1 Proses Pembentukan Kunci (*Key Generation*)

Ada beberapa tahapan dalam proses pembentukan kunci yaitu:

1. Dapatkan bilangan prima *p* dan *q* secara acak dan terpisah untuk tiap-tiap *p* dan *q*. Semakin besar bilangan yang diberikan maka semakin banyak faktorial sehingga semakin sulit data dapat dipecahkan dalam waktu yang singkat.
2. Sesudah nilai *p* dan *q* didapatkan, maka menentukan modulus yang nantinya berperan dalam pembentukan kunci publik dan kunci privat. Nilai *n* didapatkan dari hasil perkalian antara *p* dan *q* dan di simpan dalam variabel bernama *n*. Sebaiknya $p \neq$ q sebab jika $p = q$ maka $n = p^2$ dan *p* dapat diperoleh dengan menarik akar

pangkat dua dari *n*. Adapun persamaannya adalah sebagai berikut :

$$n = p*q$$

3. Proses selanjutnya yaitu mencari nilai untuk disimpan di *Φ(n)*. Dengan persamaannya yaitu :

$$\Phi(n) = (p-1)(q-1)$$

4. Pilih sebuah angka kecil *e* yang relatif prima terhadap *Φ(n)* untuk menghasilkan nilai 1 dengan syarat nilai $1 < e < \Phi(n)$ (pembagi ini dinyatakan gcd atau *Greatest Common Divisor*). Adapun persamaan nya yaitu

$$gcd\ (e, \Phi(n))=1$$

5. Pembentukan kunci privat dengan menggunakan persaman yaitu :

   $e.d = 1 + (mod(m))$

   $d = 1+(mod(m))/e$

   sehingga d dapat dihitung dengan rumus :

$$d = \frac{1 + k\Phi(n)}{e}$$

Variabel *k* di atas adalah sembarang nilai bulat (1, 2, 3 .. dan seterusnya) yang dicobakan terus sehingga dihasilkan nilai *d* yang bulat.

6. Dari beberapa variabel yang digunakan dalam proses diatas terdapat dua sifat yaitu nilai dari variabel **p**, **q**, **m**, d bersifat rahasia dan nilai dari variabel **n**, **e** boleh dipublikasikan.
7. Kunci publik adalah pasangan (*e*,*n*)
8. Kunci privat adalah pasangan (*d*,*n*)

3.2.1.4.2 Proses Enkripsi

Dalam melakukan proses enkripsi pesan dengan menggunakan kunci publik (*e*,*n*), *plaintext* disusun menjadi blok-blok m1, m2, ..., dan seterusnya dan setiap blok m1 dienkripsi menjadi blok c1 dengan persamaan :

$$c_i = m_i{}^{e} \bmod n$$

dalam hal ini berlaku :

c = ciphertext

m = pesan (*plaintext*)

e = kunci publik

n = modulus

i = indeks blok yang akan dienkripsi

3.2.1.4.3 Proses Dekripsi

Setelah melakukan proses enkripsi, maka dilakukan proses dekripsi dengan menggunakan kunci privat (*d*,*n*) dan menggunakan persamaan :

$$m_i = c_i^{\,d} \bmod n$$

dalam hal ini berlaku :

c = ciphertext

m = pesan (*plaintext*)

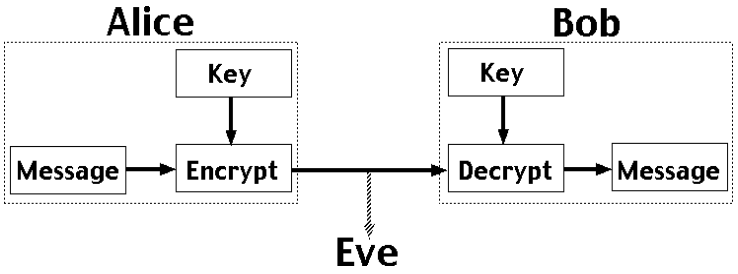
d = kunci privat

n = modulus

i = nilai-nilai blok yang akan didekripsi

3.2.1.5 Contoh Kasus Algoritma RSA

Sebagai contoh Alice dan Bob ingin berkomunikasi dengan privasi tinggi, pada kenyataannya ada pihak ketiga yang ingin mengakses komunikasi mereka, yaitu Eve. Karena menggunakan kunci yang berbeda pada saat enkripsi dan dekripsi, maka Alice dan Bob tetap dapat berkomuniasi tanpa terganggu oleh pihak ke tiga yaitu Eve.



Gambar 3.2

Contoh Kasus Algoritma RSA

3.2.1.5.1 Menggunakan Modulus Dua Digit

Menggunakan modulus dua digit untuk mengirimkan pesan yang akan dikirimkan oleh Alice ke Bob adalah "PESAN INI".

1. Dicari nilai *p* dan *q* secara acak dengan syarat $p \neq q$. Asumsikan kali ini dimisalkan nilai $p = 3$ dan nilai $q = 7$.
2. Sesudah dicari nilai *p* dan *q* maka tahap selanjutanya yaitu menghitung nilai *n* yang berguna sebagai modulus RSA sebagai pasangan kunci publik dan kunci privat.

   $n=p*q$

   $n=3*7$

   $n=21$
3. Mencari nilai $\Phi(n)$ yang berguna untuk mencari nilai *d* untuk kunci privat.

$\Phi(n)=(p-1)(q-1)$

$\Phi(n)=(3-1)(7-1)$

$\Phi(n)= 12$

4. Mencari nilai *e* (*enciphering*) yang relatif prima terhadap nilai $\Phi(n)$ sehingga $gcd(e, \Phi(n))=1$ dengan syarat $1<e<\Phi(n)$. Adapun langkah nya yaitu :

   $e=2 => gcd(e,12) = 2$

   $e=3 => gcd(e,12) = 3$

   $e=5 => gcd(e,12) = 1$

   karena nilai $e=5$ relatif prima terhadap nilai $\Phi(n)$ yaitu 12 dan memenuhi syarat maka asumsikan nilai $e=5$.

5. Sesudah nilai *e* didapat, maka selanjutanya mencari nilai *d* (*deciphering*) dengan menggunakan persamaan :

   $d = 1+k(\Phi(n))/e$

   $d = 1+k(12)/5$

   Variabel *k* adalah konstanta bilangan bulat yang dicobakan sampai didapat nilai *d* dengan nilai integer (bilangan bulat).

Tabel 3.1

Nilai *d* (modulus 2 digit)

k	***Φ(n)***	***e***		***d***
1	12	5		2.6
2	12	5		5

Dengan nilai *k=2* maka didapat nilai *d* yang bulat yaitu bernilai 5 sehingga nilai *d=5*.

6. Dari langkah-langkah yang sudah dilakukan maka didapat kunci RSA yaitu :

   kunci public (*n.e*) (21,5)

   kunci privat (*n,d*) (21,5)

7. Alice mengirim pesan ke Bob dengan pesan "PESAN INI". Alice mempunyai kunci publik yaitu (21,5) dan Bob mempunyai kunci privat yaitu (21,5). Sebelum pesan dapat dikirimkan maka perlu dikonversi ke bentuk ASCII dan dienkripsi.

Tabel 3.2

Konversi Char ke Desimal

(modulus 2 digit)

Plainteks	**Kode ASCII**
P	80
E	69

Plainteks	Kode ASCII
S	83
A	65
N	78
<spasi>	32
I	73
N	78
I	73

Sehingga dari hasil diatas bila dijadikan suatu deret dari ASCII menjadi :

M = 806983657832737873

8. Pecah nilai diatas *M* menjadi blok-blok dengan digit kurang dari jumlah digit *n*. Hal ini untuk menjaga konsistensi data pada saat dekripsi nanti. Karena nilai *n=21* maka jumlah digit untuk setiap blok yaitu 1 digit. Jumlah digit M di atas adalah sebanyak 18 digit. Oleh karena itu maka akan terdapat 18 blok yang akan melewati proses enkripsi.
9. Selanjutnya blok-blok plainteks tersebut dienkripsi dengan menggunakan persamaan :

$$c_i = m_i^{\,e} \bmod n$$

Sehingga didapat hasil enkripsi dalam tabel sebagai berikut :

Tabel 3.3

Enkripsi (modulus 2 digit)

Blok Ke-	m	e	n	c
1	8	5	21	8
2	0	5	21	0
3	6	5	21	6
4	9	5	21	18
5	8	5	21	8
6	3	5	21	12
7	6	5	21	6
8	5	5	21	17
9	7	5	21	7
10	8	5	21	8
11	3	5	21	12
12	2	5	21	11
13	7	5	21	7
14	3	5	21	12
15	7	5	21	7
16	8	5	21	8
17	7	5	21	7
18	3	5	21	12

Sesuai dengan ketentuan dari data sebelumnya yaitu nilai modulus n=21 (2 digit) maka nilai maksimal untuk setiap blok nilai *c* adalah sebanyak 2 digit sehingga untuk menjaga konsistensi nilai *c* yang memiliki nilai tidak sama dengan 2 digit

maka diberi digit tambahan nol (0) diawal nilai *c* tersebut.

Hasil dari tabel diatas ditambah dengan tambahan 1 digit nol untuk yang kurang dari dua digit yaitu

C = 080006188120617070812110712070807l2

Hasil C inilah yang merupakan hasil enkripsi pesan yang berisi "PESAN INI" sehingga pihak ketiga tidak dapat membaca pesan tersebut.

10. Pesan yang dikirim oleh Alice yang sudah dienkripsi kemudian diteruskan ke Bob. Bob yang sudah mempunyai kunci privat kemudian berusaha mendekripsi pesan tersebut.
11. Dari *ciphertext* tersebut kemudian dikembalikan lagi menjadi bentuk *plaintext*. Proses dekripsi ini berlangsung selama ditemukan akhir dari panjang blok *ciphertext* yang dihasilkan pada saat enkripsi. Panjang maksimal *ciphertext* setiap blok maksimal adalah 2 digit maka pada proses dekripsi *ciphertext* juga akan dipecah menjadi 2 digit sehingga akan menghasilkan sebagai berikut :

$m_i = c_i^{\,d} \bmod n$

Tabel 3.4

Dekripsi (modulus 2 digit)

Blok Ke-	c	d	n	M
1	8	5	21	8
2	0	5	21	0
3	6	5	21	9
4	18	5	21	8
5	8	5	21	3
6	12	5	21	6
7	6	5	21	5
8	17	5	21	7
9	7	5	21	8
10	8	5	21	3
11	12	5	21	2
12	11	5	21	7
13	7	5	21	3
14	12	5	21	7
15	7	5	21	3
16	8	5	21	8
17	7	5	21	7
18	12	5	21	3

Dari tabel 3.4 diatas diperoleh kembali nilai *M* = 806983657832737873. Jika diperhatikan maka nilai M sebelum dienkripsi dan nilai M sesudah didekripsi adalah sama.

Selanjutnya mengubah nilai M yang berbentuk kode ASCII ke bentuk *plaintext* yang mudah

dibaca. Berikut adalah hasil konversi dari nilai *M* yang berupa desimal ke bentuk *char*.

Tabel 3.5

Konversi Desimal ke Char

(modulus 2 digit)

Kode ASCII	Plainteks
80	P
69	E
83	S
65	A
78	N
32	<spasi>
73	I
78	N
73	I

Setelah proses ini maka Bob akan mengerti pesan yang telah di kirim sebelumnya oleh Alice yaitu “PESAN INI”.

3.2.1.5.2 Menggunakan Modulus Tiga Digit

Menggunakan modulus tiga digit untuk mengirimkan pesan yang akan dikirimkan oleh Alice ke Bob adalah “Aku”.

1. Dicari nilai *p* dan *q* secara acak dengan syarat $p \neq q$. Asumsikan kali ini dimisalkan nilai $p = 11$ dan nilai $q = 13$.
2. Sesudah dicari nilai *p* dan *q* maka tahap selanjutanya yaitu menghitung nilai *n* yang berguna sebagai modulus RSA sebagai pasangan kunci publik dan kunci privat.

   $n=p*q$

   $n=11*13$

   $n=143$

3. Mencari nilai $\Phi(n)$ yang berguna untuk mencari nilai *d* untuk kunci privat.

   $\Phi(n)=(p-1)(q-1)$

   $\Phi(n)=(11-1)(13-1)$

   $\Phi(n)= 120$

4. Mencari nilai *e* (*enciphering*) yang relatif prima terhadap nilai $\Phi(n)$ sehingga $gcd(e, \Phi(n))=1$ dengan syarat $1<e< \Phi(n)$. Adapun langkah nya yaitu :

   $e=2 => gcd(e,12) = 2$

   $e=3 => gcd(e,12) = 3$

   $e=7 => gcd(e,12) = 1$

karena nilai *e=7* relatif prima terhadap nilai *Φ(n)* yaitu 120 dan memenuhi syarat maka asumsikan nilai *e=7*.

5. Sesudah nilai *e* didapat, maka selanjutnya mencari nilai *d* (*deciphering*) dengan menggunakan persamaan :

   $d = 1+k(\Phi(n))/e$

   $d = 1+k(120)/7$

   Variabel *k* adalah konstanta bilangan bulat yang dicobakan sampai didapat nilai *d* dengan nilai integer (bilangan bulat).

Tabel 3.6

Nilai *d* (modulus 3 digit)

k	***Φ(n)***	***e***		***d***
1	120	7		17,2857142858
2	120	7		34,4285714285
3	120	7		51,5714285714
4	120	7		68,7142857142
5	120	7		85,8571428571
6	120	7		103

Dengan nilai *k=6* maka didapat nilai *d* yang bulat yaitu bernilai 103 sehingga nilai *d=103*.

6. Dari langkah-langkah yang sudah dilakukan maka didapat kunci RSA yaitu :

kunci public (*n.e*) (143,7)

kunci privat (*n,d*) (143,103)

7. Alice mengirim pesan ke Bob dengan pesan "Aku". Alice mempunyai kunci publik yaitu (143,7) dan Bob mempunyai kunci privat yaitu (143,103). Sebelum pesan dapat dikirimkan maka perlu dikonversi ke bentuk ASCII dan dienkripsi.

Tabel 3.7

Konversi Char ke Desimal

(modulus 3 digit)

Plainteks	Kode ASCII
A	65
k	107
u	117

Sehingga dari hasil diatas bila dijadikan suatu deret dari ASCII menjadi :

M = 65107117

8. Pecah nilai diatas *M* menjadi blok-blok dengan digit kurang dari jumlah digit *n*. Hal ini untuk menjaga konsistensi data pada saat dekripsi nanti. Karena nilai *n=143* maka jumlah digit untuk setiap blok yaitu kurang dari 3 digit dan asumsikan menggunakan 2 digit untuk setiap blok nya. Jumlah

digit M di atas adalah sebanyak 8 digit. Oleh karena itu maka akan terdapat 4 blok yang akan melewati proses enkripsi.

9. Selanjutnya blok-blok plainteks tersebut dienkripsi dengan menggunakan persamaan :

   $c_i = m_i^{\,e} \bmod n$

   Sehingga didapat hasil enkripsi dalam tabel sebagai berikut :

Tabel 3.8

Enkripsi (modulus 3 digit)

Blok Ke-	m	e	n	c
1	65	7	143	65
2	10	7	143	10
3	71	7	143	124
4	17	7	143	30

Sesuai dengan ketentuan dari data sebelumnya yaitu nilai modulus *n*=143 (3 digit) maka nilai maksimal untuk setiap blok nilai *c* adalah sebanyak 3 digit sehingga untuk menjaga konsistensi nilai *c* yang memiliki nilai tidak sama dengan 3 digit maka diberi digit tambahan nol (0) diawal nilai *c* tersebut.

Hasil dari tabel diatas ditambah dengan tambahan 1 digit nol untuk yang kurang dari dua digit yaitu

C = 065010124030

Hasil C inilah yang merupakan hasil enkripsi pesan yang berisi “Aku” sehingga pihak ketiga tidak dapat membaca pesan tersebut.

10. Pesan yang dikirim oleh Alice yang sudah dienkripsi kemudian diteruskan ke Bob. Bob yang sudah mempunyai kunci privat kemudian berusaha mendekripsi pesan tersebut.
11. Dari *ciphertext* tersebut kemudian dikembalikan lagi menjadi bentuk *plaintext*. Proses dekripsi ini berlangsung selama ditemukan akhir dari panjang blok *ciphertext* yang dihasilkan pada saat enkripsi. Panjang maksimal *ciphertext* setiap blok maksimal adalah 3 digit maka pada proses dekripsi *ciphertext* juga akan dipecah menjadi 3 digit sehingga akan menghasilkan sebagai berikut :

$m_i = c_i^{\,d} \bmod n$

Tabel 3.9

Dekripsi (modulus 3 digit)

Blok Ke-	c	d	n	M
1	065	103	143	65
2	010	103	143	10
3	124	103	143	71
4	030	103	143	17

Dari tabel diatas diperoleh kembali nilai *M* = 65107117. Jika diperhatikan maka nilai M sebelum dienkripsi dan nilai M sesudah didekripsi adalah sama.

Selanjutnya mengubah nilai M yang berbentuk kode ASCII ke bentuk *plaintext* yang mudah dibaca. Berikut adalah hasil konversi dari nilai *M* yang berupa desimal ke bentuk *char*.

Tabel 3.10

Konversi Desimal ke Char

(modulus 3 digit)

Kode ASCII	**Plainteks**
65	A
107	k
117	u

Setelah proses ini maka Bob akan mengerti pesan yang telah di kirim sebelumnya oleh Alice yaitu “Aku”.

3.2.1.5.3 Menggunakan Modulus Empat Digit

Menggunakan modulus tiga digit untuk mengirimkan pesan yang akan dikirimkan oleh Alice ke Bob adalah “Lg dmn :)”.

1. Dicari nilai *p* dan *q* secara acak dengan syarat $p \neq q$. Asumsikan kali ini dimisalkan nilai $p = 47$ dan nilai $q = 71$.
2. Sesudah dicari nilai *p* dan *q* maka tahap selanjutanya yaitu menghitung nilai *n* yang berguna sebagai modulus RSA sebagai pasangan kunci publik dan kunci privat.

   $n=p*q$

   $n=47*71$

   $n=3337$
3. Mencari nilai $\Phi(n)$ yang berguna untuk mencari nilai *d* untuk kunci privat.

   $\Phi(n)=(p-1)(q-1)$

   $\Phi(n)=(47-1)(71-1)$

   $\Phi(n)= 3220$
4. Mencari nilai *e* (*enciphering*) yang relatif prima terhadap nilai $\Phi(n)$ sehingga $gcd(e, \Phi(n))=1$ dengan syarat $1<e<\Phi(n)$. Asumsikan kita

menggunakan nilai *e=79* karena relatif prima terhadap *Φ(n)*.

5. Sesudah nilai *e* didapat, maka selanjutanya mencari nilai *d* (*deciphering*) dengan menggunakan persamaan :

   $d = 1+k(\Phi(n))/e$

   $d = 1+k(3220)/79$

Tabel 3.11

Nilai *d* (modulus 4 digit)

k	***Φ(n)***	***e***		***d***
1	3220	79		40,77215189874
2	3220	79		81,53164556920
3	3220	79		122,2911392405
4	3220	79		163,0506329113
5	3220	79		203,8101265822
6	3220	79		244,5692025316
7	3220	79		285,3291139240
8	3220	79		326,0886075943
9	3220	79		366,8481102652
10	3220	79		407,6075945936
11	3220	79		448,3670886075
12	3220	79		489,1265822784
13	3220	79		529,8860759493
14	3220	79		570,6455696202
15	3220	79		611,4050632911
16	3220	79		652,1645569620
17	3220	79		692,9240506329

k	*Φ(n)*	*e*		*d*
18	3220	79		733,6835443037
19	3220	79		744,4430379746
20	3220	79		815,2025316455
21	3220	79		855,9620263164
22	3220	79		896,7215189873
23	3220	79		937,4810126582
24	3220	79		978,2405063291
25	3220	79		1019

Dengan nilai *k=25* maka didapat nilai *d* yang bulat yaitu bernilai 1019 sehingga nilai *d=1019*

6. Alice mengirim pesan ke Bob dengan pesan “Lg dmn :)”. Alice mempunyai kunci publik yaitu (79,3337) dan Bob mempunyai kunci privat yaitu (1019,3337). Sebelum pesan dapat dikirimkan maka perlu dikonversi ke bentuk ASCII dan dienkripsi.

Tabel 3.12

Konversi Char ke Desimal

(modulus 4 digit)

Plainteks	**Kode ASCII**
L	76
g	103
<spasi>	32
d	100

Plainteks	Kode ASCII
m	109
n	110
<spasi>	32
:	58
)	41

Sehingga dari hasil diatas bila dijadikan suatu deret dari ASCII menjadi :

M = 7610332100109110325841

7. Pecah nilai diatas *M* menjadi blok-blok dengan digit kurang dari jumlah digit *n*. Hal ini untuk menjaga konsistensi data pada saat dekripsi. Karena nilai *n=3337* maka jumlah digit untuk setiap blok yaitu kurang dari 4 digit dan asumsikan menggunakan 2 digit untuk setiap blok nya. Jumlah digit M di atas adalah sebanyak 22 digit. Oleh karena itu maka akan terdapat 11 blok yang akan melewati proses enkripsi.
8. Selanjutnya blok-blok plainteks tersebut dienkripsi dengan menggunakan persamaan :

   $c_i = m_i^{\,e} \bmod n$

   Sehingga didapat hasil enkripsi dalam tabel sebagai berikut :

Tabel 3.13

Enkripsi (modulus 4 digit)

Blok Ke-	m	e	n	c
1	76	79	3337	1903
2	10	79	3337	3269
3	33	79	3337	1260
4	21	79	3337	1249
5	00	79	3337	00
6	10	79	3337	3269
7	91	79	3337	1252
8	10	79	3337	3269
9	32	79	3337	1379
10	58	79	3337	1697
11	41	79	3337	857

Sesuai dengan ketentuan dari data sebelumnya yaitu nilai modulus n=3337 (4 digit) maka nilai maksimal untuk setiap blok nilai *c* adalah sebanyak 4 digit sehingga untuk menjaga konsistensi nilai *c* yang memiliki nilai tidak sama dengan 4 digit maka diberi digit tambahan nol (0) diawal nilai *c* tersebut.

Hasil dari tabel diatas ditambah dengan tambahan 1 atau lebih digit nol untuk yang kurang dari empat digit yaitu:

C=190332691260124900003269125232691379160970857

Hasil C inilah yang merupakan hasil enkripsi pesan yang berisi “Lg dmn :)” sehingga pihak ketiga tidak dapat membaca pesan tersebut.

8. Pesan yang dikirim oleh Alice yang sudah dienkripsi kemudian diteruskan ke Bob. Bob yang sudah mempunyai kunci privat kemudian berusaha mendekripsi pesan tersebut.
9. Dari *ciphertext* tersebut kemudian dikembalikan lagi menjadi bentuk *plaintext*. Proses dekripsi ini berlangsung selama ditemukan akhir dari panjang blok *ciphertext* yang dihasilkan pada saat enkripsi. Panjang maksimal *ciphertext* setiap blok maksimal adalah 4 digit maka pada proses dekripsi *ciphertext* juga akan dipecah menjadi 4 digit sehingga akan menghasilkan sebagai berikut :

$m_i = c_i^{\,d} \bmod n$

Tabel 3.14

Dekripsi (modulus 4 digit)

Blok Ke-	c	d	n	M
1	1903	1019	3337	76
2	3269	1019	3337	10
3	1260	1019	3337	3

Blok Ke-	c	d	n	M
4	1249	1019	3337	21
5	0000	1019	3337	00
6	3269	1019	3337	10
7	1252	1019	3337	91
8	3269	1019	3337	10
9	1379	1019	3337	32
10	1697	1019	3337	58
11	0857	1019	3337	41

Dari tabel 3.15 diatas diperoleh kembali nilai *M* = 7610332100109110325841. Jika diperhatikan maka nilai M sebelum dienkripsi dan nilai M sesudah didekripsi adalah sama.

Selanjutnya mengubah nilai M yang berbentuk kode ASCII ke bentuk *plaintext* yang mudah dibaca. Berikut adalah hasil konversi dari nilai *M* yang berupa desimal ke bentuk *char*.

Tabel 3.15

Konversi Desimal ke Char

(modulus 4 digit)

Kode ASCII	Plainteks
76	L
103	g
32	<spasi>

Kode ASCII	Plainteks
100	d
109	m
110	n
32	<spasi>
58	:
41	)

Setelah proses ini maka Bob akan mengerti pesan yang telah di kirim sebelumnya oleh Alice yaitu "Lg dmn :)".

3.2.2 Analisis Sintaksis Java

Java yang merupakan alat bantu dalam penelitian ini, maka perlu untuk diketahui struktur dasar baik itu pernyataan, tipe data, logika dan kelas-kelas yang terkait dalam pengembangan algoritma RSA.

3.2.2.1 Pernyataan atau *Statement*

```
package <OPTIONAL>
import <OPTIONAL>
<CLASS MODIFIER> class <NAMA KELAS> {

}
```

Penjelasan :

1. *package* : adalah sebuah tempat menyimpan kelas yang akan kita buat. *Package* ini sifatnya *optional* sehingga kita bsa menggunakannya ataupun tidak. Tapi sangat disarankan ketika menulis program disertakan dengan

package sehingga program yang ditulis menjadi lebih sistematis dan terstruktur.

2. *import* : mengambil kelas dari *package* lain. Sifatnya *optional* sesuai kebutuhan kelas (*atribute,method*) yang akan ditulis.
3. *access modifier* : sebuah sifat dari kelas. Mengatur sebuah kelas apakah tersedia untuk kelas lain atau tidak. *Access modifier* terbagi menjadi terbagi menjadi empat macam yaitu : *default*, *public*, *abstract* dan *final*.
4. *nama kelas* : nama kelas yang akan ditulis. Sesuai dengan yang direkomendasikan oleh Java, setiap penulisan nama kelas maka harus hurup kapital setiap kata dan tidak boleh ada spasi. Misalnya nama kelas *tessegitiga* menjadi *TesSegitiga*.
5. Tanda kurawal awal dan akhir : dalam Java, setiap pendeklarasian kelas dan *method* sering di awali dengan kurawal awal dan kurawal akhir ({ }). Tapi hal ini tidak berlaku untuk *interface* dan kelas yang bersifat *abstract* yang *method* nya tidak boleh ada bodi *method* ({ }).

Java adalah bahasa pemrograman yang *case sensitive*, oleh karena itu jika kita membuat kekeliruan dalam membuat hurup kapital dan hurup kecil maka hal tersebut akan menyebabkan program gagal untuk di kompilasi.

3.2.2.2 Variabel Java

Di Java, variabel harus dideklarasikan sebelum digunakan untuk menyediakan memori yang digunakan untuk menampung tipe nilai dan nama dari variabel tersebut.

Berikut ini adalah contoh deklarasi variabel :

```
double salary;
long distance;
int day;
```

Titik koma selalu mengakhiri masing-masing deklarasi. Dalam Java, titik koma (;) sangat diperlukan karena merupakan akhir dari sebuah deklarasi. Selain bentuk deklarasi diatas juga dapat memakai deklarasi baris tunggal, misalnya :

```
int i,j;
double a,b;
```

Tetapi bentuk pendeklarasian variabel di atas sangat tidak disarankan karena kode tidak akan mudah dipelihara dan dibaca.

3.2.2.3 Tipe Data

Adapun tipe data dari Java diantaranya :

1. Tipe Data Integer

Tabel 3.16

Tipe Data Integer

Tipe	Ukuran	Range
byte	8 bit	-128 s/d 127
short	16 bit	-32768 s/d 32767

Tipe	Ukuran	Range
`int`	32 bit	-2147483648 s/d 2147483647
`long`	64 bit	–9,223,372,036,854,775,808 s/d 9,223,372,036,854,775,807
`char`	16 bit	0 s/d 65535

2. Tipe Data *Floating-Point*

Tabel 3.17

Tipe Data *Floating Point*

Tipe	Ukuran	Range
`float`	32 bits	3.4e-038 s/d 3.4e+038
`double`	64 bits	1.7e-308 s/d 1.7e+308

3. Tipe Data Boolean

   Tipe data boolean merepresentasikan nilai benar dan salah. Java mendefinisikan nilai benar dan salah dengan *keyword true* dan *false*.

4. Tipe Data *Character*

   Tipe data *character* untuk menyimpan karakter *unicode* tunggal. Karena karakter *unicode* disusun dari nilai 16 bit, karena tipe data *char* adalah 16 bi *unsigned integer*.

   Unicode mendefinisikan sebuah kumpual karakter yang bisa direpresenasikan oleh semua karakter yang dipahami oleh manusia. ASCII standar 8 bit adalah sebuah *subset* dari *unicode* dan mempunyai range dari 0 sampai 127.

3.2.2.4 Struktur Kontrol

Struktur kontrol pemilihan adalah pernyataan dari Java yang mengijinkan *user* untuk memilih dan mengeksekusi blok kode spesifik dan mengabaikan blok kode yang lain. Adapun tipe dari struktur kontrol tersebut yaitu :

1. Pernyataan *if*

   Bentuk dari pernyataan *if* akan menentukan sebuah pernyataan yang mengeksekusi jika dan hanya jika pernyataan tersebut bernilai benar atau *true*. Adapun bentuk dari pernyataan *if*,

   ```
   if (boolean expresion)
       Statement;
   ```

   atau

   ```
   if (boolean expresion) {
       Statement1;
       Statement2;
       Statement3;
   }
   ```

   Di mana, *boolean expresion* adalah sebuah pernyataan logika benar atau salah atau variabel dengan tipe *boolean*.

2. Pernyataan *if-else*

   Pernyataan *if-else* digunakan apabila kita ingin mengeksekusi beberapa pernyataan dengan nilai benar dan pernyataan lain dengan kondisi salah. Adapun bentuk dari pernyataan *if-else*,

   ```
   if (boolean expresion){
       Statement;
   ```

```
}else{
        Statement;
}
```

atau

```
 if (boolean expresion) {
        Statement1;
        Statement2;
       Statement3;
}else{
        Statement1;
        Statement2;
       Statement3;
}
```

3. Pernyataan *if-else-if*

Pernyataan pada bagian kondisi *else* dari blok *if-else* dapat menjadi struktur *if-else* yang lain. Kondisi struktur ini mengijinkan kita untuk membuat seleksi persyaratan yang lebih kompleks. Adapun bentuk dari pernyataan *if-else-if*,

```
if (boolean expresion1) {
        Statement1;
}else if (boolean expresion2) {
        Statement2;
}else{
        Statement3;
}
```

Jika *boolean expresion1* bernilai benar maka *statement1* akan dijalankan . Jika *boolean expresion2* bernilai benar maka *statement2* akan dijalankan dan melewati *statement3*.

4. Pernyataan *switch*

Disamping dengan menggunakan bentuk *if-else*, Java juga menyediakan bentuk struktur kontrol dengan

menggunakan kata kunci ***switch***. *Switch* mengkonstruksikan cabang untuk beberapa kondisi dari nilai. Adapun bentuk dari pernyataan ini,

```
switch (switch expression){
    case case_selector1 :
            Statement1;
            Statement2;
            ...
            Break;
    case case_selector2:
            Statement1;
            Statement2;
            ...
            Break;
    default :
            Statement1;
            Statement2;
            ...
            Break;
}
```

Ketika pernyataan *switch* ditemukan pada potongan kode program, Java pertama kali akan memeriksa *switch expression* dan menuju ke *case* yang akan menyamakan nilai yang dimiliki oleh *switch expression*. Selanjutanya program akan mengeksekusi pernyataan pada dari kode setelah *case* yang ditemukan sampai menemui pernyataan *break*. Selanjutnya akan mengabaikan pernyataan yang lainnya hingga akhir dari struktur dari pernyataan *switch*.

3.2.2.5 Kelas *Java.Math*

Sesuai dengan namanya pada paket Java ini berisi *method-method* yang banyak digunakan untuk operasi matematika (eksponen, logaritma, trigonometri dan lain-lain). Dengan kelas yang bersifat *final* dan semua variabel maupun *method* yang bersifat *static* membuat kelas ini tidak bisa diturunkan dan tidak bisa dibuat objek. Sehingga pada saat penggunaanya, Kelas *Java.Math* harus diasosiakan langsung dengan *method* yang akan digunakan.

Adapun *method* yang sering digunakan diantaranya :

1. Metode untuk operasi trigonometri

   Trigonometri digunakan untuk menghitung sesuatu yang berhubungan dengan sudut. Semua metode ini mempunyai tipe *double* sehingga hasil dari operasi masing-masing *method* mempunyai satu buah parameter dan akan mengembalikan nilai dengan tipe *double*.

Tabel 3.18

Method Trigonometri

Method	Keterangan
`static double sin(double a)`	Melakukan operasi trigonometri *sinus* dengan *a* inputan dan mengembalikan nilai *double*
`static double cos(double a)`	Melakukan operasi trigonometri *cosinus* dengan *a* inputan dan

	mengembalikan nilai *double*
`static double tan(double a)`	Melakukan operasi trigonometri *tangen* dengan *a* inputan dan mengembalikan nilai *double*
`static double asin(double a)`	Melakukan operasi trigonometri a*sinus* dengan *a* inputan dan mengembalikan nilai *double*
`static double acos(double a)`	Melakukan operasi trigonometri a*cosinus* dengan *a* inputan dan mengembalikan nilai *double*
`static double atan(double a)`	Melakukan operasi trigonometri *atangen* dengan *a* inputan dan mengembalikan nilai *double*

contoh :

```
double sinHasil=Math.sin(Math.PI/2);
```

2. Metode *exp()*, *log()* dan *pow()*

Tabel 3.19

Method *exp()*, *log()* dan *pow()*

Method	Keterangan
`static double exp(double a)`	Menghitung pangkat dari bilangan *a* yang diinputkan dan mengembalikan nilai dengan tipe *double*
`static double log(double a)`	Mencari nilai *log* dari bilangan *a* yang diinputkan dan mengembalikan nilai dengan tipe *double*

`static double pow(double a, double b)`	Mempunyai dua inputan yaitu *a* dan *b*. Menghitung nilail a^b dan mengembalikan nilai dengan tipe *double*

Contoh :

```
double d1=12.0;
double d2=Math.pow(d1,3.0);
```

3. Metode *ceil(), floor(), round()* dan *rint()*

Tabel 3.20

Method *ceil(), floor(), round()* dan *rint()*

Method	Keterangan
`static double ceil (double a)`	Mengirim *ceiling* yaitu bilangan utuh terkecil yang lebih besar atau sama dengan *a*
`static double floor (double a)`	Mengirim *floor* yaitu bilangan utuh terbesar yang kurang atau sama dengan *a*
`static int round (float a)` `static long round (double a)`	Kedua metode ini membulatkan nilai *float* dan *double* menjadi bilangan bulat terdekat dengan hasil yang bertipe *integer* dan *long*
`static double rint (double a)`	Sama dengan *method round*, tetapi tipe yang dikirim adalah *double*

4. Metode *random()*

   Metode *Math.random()* membangkitkan angka *pseudo-random* acak dari nilai 0.0 sampai nilai 1.0. Untuk menghasilkan angka acak dengan tipe yang berbeda java menyediakan kelas khusus yaitu *Random*. Kelas ini berada pada *packace java.util.Random*.

3.2.2.6 RMI (*Remote Method Invocation*)

RMI atau kepanjangan dari *Remote Method Invocation* adalah sebuah fasilitas yang dimiliki Java untuk dapat memanggil objek dari sisi yang berbeda yang dalam hal ini yaitu *client* dan *server*.

Dengan kata lain, RMI merupakan jembatan penghubung antara satu aplikasi dengan aplikasi lainnya. RMI yang dapat diimplementasikan dalam dua komputer misalnya yang tergabung dalam *client server* juga dapat diimplementasikan dalam satu komputer. Kelas, *interface* dan *exceptions* dalam paket java.rmi diantaranya :

1. *Classes*

Tabel 3.21

Class RMI

Class	**Keterangan**
`RemoteObject`	Menyediakan semantik *remote* dari objek dengan mengimplementasi *method* untuk *hashCode*, *equals* dan *toString*.
`RemoteServer`	Menyediakan *framework* untuk mendukung sebuah wilayah yang lebar dari semantik *remote reference*.
`UnicastRemoteObject`	Digunakan untuk membuat sebuah *remote* dengan JRMP dan berisi sebuah *stub* untuk berkomunikasi dengan objek
`RMISecurityManager`	Kelas yang berfungsi untuk mengatur keamana RMI.
`RMIClassLoader`	*RMIClassLoader* terdiri dari *static method* untuk mendukung RMI.
`LogStream`	Menyediakan mekanisme untuk *logging error*.
`LocateRegistry`	Digunakan untuk

	mengisi sebuah referensi menuju registri *remote object*.

2. *Interface*

Tabel 3.22

Interface RMI

Interface	**Keterangan**
`Remote`	Mengidentifikasi *method* yang akan dipanggil dari mesin virtual yang berbeda.
`Unreferenced`	Untuk menerima sebuah pemberitahuan ketika tidak ada *client* yang merefensi.
`LoaderHandler`	Digunakan secara internal oleh RMI.
`Registry`	Menyediakan *method* untuk penyimpanan dan penyeleksian referensi *remote object*.
`RegistryHandler`	Digunakan secara internal oleh RMI.

3. *Exceptions*

Tabel 3.23

Exception RMI

Exceptions	**Keterangan**
`NoSuchObjectException`	Dilemparkan ketika sesuatu berusaha untuk memanggil sebuah *method* dari sebuah *object* yang sudah tidak ada di mesin virtual.
`NotBoundException`	Dilemparkan ketika sesuatu berusaha untuk memilih nama registri.
`RemoteException`	Melempar eksepsi ketika sesuatu terjadi ketika eksepsi dari pemanggilan sebuah *remote method*.
`UnknowsHostExceptions`	Terjadi ketika selama pembuatan koneksi untuk *remote* untuk pemanggilan *method*.

3.2.2.7 *Port* dan *Socket*

Java menyediakan sekumpulan kelas yang berada dalam paket *java.net*, untuk membangun aplikasi yang berkaitan dengan jaringan. Kelas, *interface* dan *exceptions* dalam paket *java.net* diantaranya :

1. *Classes*

Tabel 3.24

Class Java.Net

Class	**Keterangan**
`ContentHandler`	Kelas abstrak *ContentHandler* adalah *superclass* yang membaca *object* dari kelas *URLConnection*
`DatagramPacket`	Digunakan untuk mengimplementasi pengantar layanan paket.
`DatagramSocket`	Kelas ini merepresentasikan sebuah *socket* untuk mengirim dan menerima paket *datagram*
`DatagramSocketImpl`	Kelas dasar untuk implementasi *datagram* dan *multicast socket*
`HttpURLConnection`	Kelas yang mendukung koneksi *URL* dengan dukungan fitur *HTTP*
`InetAddress`	Kelas ini merepresentasikan sebuah alamat *Internet Protocol* (IP)

`MulticastSocket`	Kelas yang digunakan untuk menerima dan mengirim IP *multicast socket*
`ServerSocket`	Menunggu *request* yang datang dari *client*, melakukan operasi dan mengirim kembali ke *client* tersebut
`Socket`	Sebuah *socket* adalah poin akhir untuk komunikasi antara dua mesin
`SocketImpl`	Kelas abstrak *SocketImpl* adalah *superclass* yang diimplementasi oleh *socket.* Digunakan membuat *socket* antara *client* dan *server*
`URL`	Kelas yang merepresentasikan sebuah *Uniform Resource Locator*
`URLConnection`	*Superclass* dari semua kelas yang merepresentasikan komunikasi antara aplikasi dan *URL*
`URLEncoder`	Kelas utilitas untuk menyandikan bentuk HTML
`URLStreamHandler`	Adalah sebuah *superclass* yang umum digunakan untuk menangani *stream protocol*

2. *Interface*

Tabel 3.25

Interface Java.Net

Interface	**Keterangan**
`FileNameMap`	Menyediakan mekanisme antra nama *file* dan tipe MIME
`SocketImplFactory`	Mendefinisikan sebuah bangunan untuk implementasi *socket*
`URLStreamHandlerFactory`	Mendefinisikan sebuah *stream protocol handler*

3. *Exception*

Tabel 3.26

Exception Java.Net

Exception	**Keterangan**
`BindException`	Menangani ekseksi yang terjadi karena berusaha untuk *bind* sebuah *socket* ke IP dan nomor port
`ConnectException`	Menangani ekseksi yang terjadi karena berusaha menyambungkan sebuah *socket* ke alamat remot dan port

`MalformedURLException`	Mengindikasikan bahwa URL yang dikirim adalah cacat atau buruk
`NoRouteToHostException`	Menangani ekseksi yang terjadi karena berusaha untuk *bind* sebuah *socket* ke IP dan nomor port
`ProtocolException`	Mengindikasikan *error* yang mandasari sebuah *protocol*
`SocketException`	Mengindikasikan *error* yang mandasari sebuah *protocol*
`UnknownHostException`	Mengindikasikan bahwa alamat IP tidak dapat ditentukan
`UnknownServiceException`	Mengindikasikan bahwa eksepsi layanan yang tidak diketahui terjadi

3.2.2.8 Kelas *BigInteger*

Tipe data integer 32 bit hanya dapat menampung bilangan dari rentang -2^31+1 sampai rentang -2^31+1 begitu juga dengan integer 64 bit hanya menampung bilangan dari rentang -2^63+1 sampai rentang 2^63+1.

Operasi algoritma RSA memerlukan tipe data yang dapat menampung bilangan yang sangat besar. Oleh karena itu, diperlukan suatu tipe data yang dapat menampung hasil dari operasi algoritma RSA tersebut.

Java sudah menyediakan kelas bernama *BigInteger* yang mampu melakukan operasi matematika (penjumlahan, pengurangan, perkalian, pembagian, modulo dan lain-lain) baik terhadap bilangan bulat maupun bilangan desimal berdasarkan *memory* komputer yang dipakai sehingga kelas BigInteger ini memiliki daya tampung yang dinamis tapi harus melihat juga batasan yang dimiliki oleh CPU dan memori komputer.

3.3 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan *non fungsional* dilakukan untuk menghasilkan spesifikasi kebutuhan *non fungsional*. Spesifikasi kebutuhan *non fungsional* adalah spesifikasi yang rinci tentang hal-hal yang akan dilakukan oleh program aplikasi ketika diimplementasikan. Analisis ini diperlukan untuk menentukan keluaran yang akan dihasilkan sistem, masukan yang diperlukan sistem, lingkup proses yang digunakan untuk mengolah *input* menjadi *output*.

3.3.1 Analisis Perangkat Keras

Analisis kebutuhan perangkat keras (*hardware*) yang diperlukan untuk proses implementasi algoritma RSA pada aplikasi *instant messager* dengan spesifikasi diantaranya :

1. Processor : Processor dengan kecepatan 2.00Ghz
2. Hardisk Drive : 80 GB
3. Memory : 1 GB
4. Monitor : 15,6 “
5. Mouse dan Keyboard
6. Kabel LAN (*Local Area Network*) sepanjang tiga meter.

3.3.2 Analisis Perangkat Lunak

Spesifikasi perangkat lunak yang dibutuhkan untuk mendukung analisis dan pembangunan adalah sebagai berikut :

1. Sistem Operasi
2. NetBeans IDE sebagai perangkat lunak untuk membangun prgram
3. JDK (*Java Development Kit*) 7
4. Windows Calculator (*Scientific Mode*)
5. Advanced Installer 9
6. Microsoft Office 2007