BAB IV

PERANCANGAN

4.1 Perancangan Sistem

Tahap perancangan sistem merupakan tahap implementasi dari hasil analisis ke dalam suatu bahasa pemrograman tertentu serta penerapan perangkat lunak yang dibangun pada lingkungan yang sesungguhnya. Dalam hal ini, perancangan akan menggunakan bahasa pemrograman Java untuk dapat mengimplementasikan Algoritma RSA yang dapat digunakan untuk sistem pengamanan pesan.

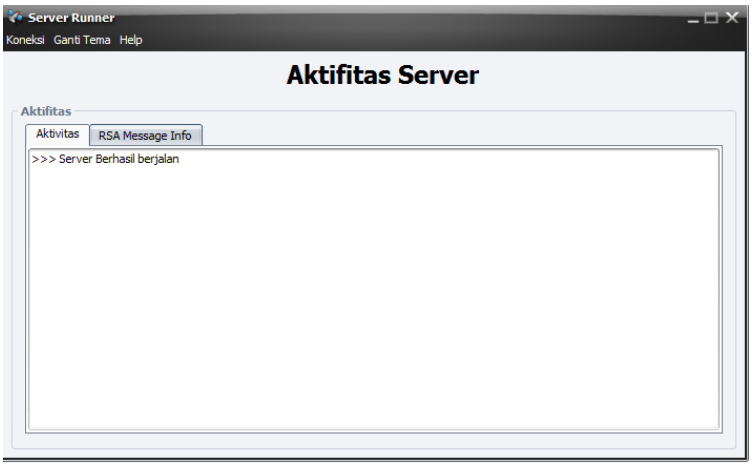
4.1.1 Tujuan Perancangan Sistem

Tujuan dari implementasi algoritma RSA ini ke dalam aplikasi *instant messenger* adalah untuk menyediakan fasilitas pengamanan pesan sehingga dapat menjaga setiap privasi diantara sesama penggunanya.

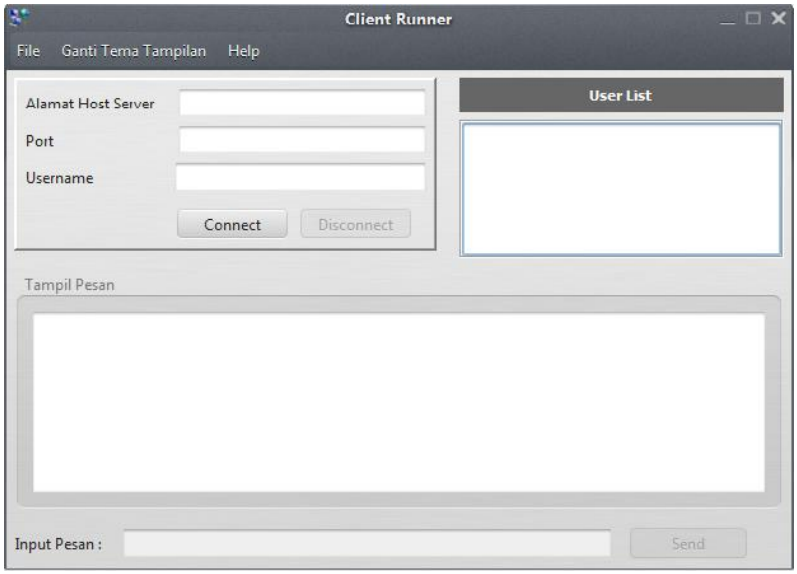
4.1.2 Gambaran Umum Sistem

Sistem terdiri dari *Client* dan *Server*. Disamping dapat mengirimkan pesan, *client* juga dapat melakukan enkripsi dan dekripsi pesan. Sedangkan server hanya sebagai pihak penyambung antara sesama *client*. Sehingga pesan yang dikirimkan oleh *client* tidak bisa terbaca karena proses dekripsi dilakukan

disisi *client* dan hanya terbaca berupa *ciphertext* hasil dari proses enkripsi. Berikut adalah tampilan *server* dan *client*.



Gambar 4.1 Tampilan Server



Gambar 4.2 Tampilan Client

4.2 Implementasi Algoritma RSA pada Instant Messenger

Algoritma RSA yang sebelumnya di analisis kemudian diimplementasikan pada aplikasi *instant messenger*. Dalam hal ini,

kebutuhan untuk proses implementasi menggunakan bahasa pemrograman Java karena beberapa kemudahan diantaranya sudah disediakannya kebutuhan untuk kriptografi diantaranya *libary* dan kelas-kelas khusus untuk menangani proses kriptografi tersebut khususnya algoritma RSA yang memerlukan bilangan yang mempunyai presisi bilangan yang sangat besar yang tidak dapat ditangani dengan tipe data biasa.

4.2.1 Penggunaan Kelas *BigInteger*

Jika presisi bilangan bulat (*Long*) dan *floating point* sudah tidak memadai, dapat menggunakan kelas *BigInteger* untuk dapat melakukan operasi aritmatika untuk bilangan bulat yang sangat besar sehingga kelas ini sering dipakai dalam aplikasi kriptografi yang melibatkan ratusan bahkan ribuan bit.

Pada algoritma RSA, terdapat beberapa besaran yang digunakan. Besaran-besaran tersebut merupakan variabel atau objek dari kelas *BigInteger*. Karena dalam kelas *BigInteger* sudah mempunyai metode-metode khusus dalam kriptografi maka dapat mempermudah dalam proses perancangan.

Adapun besaran-besaran tersebut diantaranya :

```
private BigInteger p;
private BigInteger q;
private BigInteger n;
private BigInteger r;
private BigInteger d;
private BigInteger e;
```

4.2.2 Pengembangan Cara Kerja Umum Algoritma RSA

Terdapat empat fungsi penting dalam algoritma RSA yang sudah diimplementasikan yaitu fungsi *generatePrimeNumber, generatePublicPrivateKeys, encrypt* dan *decrypt*.

4.2.2.1 Fungsi *generatePrimeNumber*

Disamping panjang deret bit algoritma RSA ditentukan oleh *p* dan *q*, tapi jika menggunakan kelas *BigInteger* maka panjang deret juga dapat ditentukan oleh panjang jumlah bit pada setiap angka yang diberikan untuk besaran *p* dan *q*. Operasi pada kelas *BigInteger* yang dapat melebihi operasi 927 digit, namun perancangan ini menggunakan jumlah panjang digit dari kelas *BigInteger* yaitu 8 bit.

Disamping itu, salah satu parameter dalam menentukan nilai *p* dan *q* haruslah acak, maka digunakan kelas *SecureRandom* yang dapat mengacak nilai bulat untuk diisi dalam variabel *p* dan *q*.

Adapun potongan program pada fungsi *generatePrimeNumber* yaitu :

```
public void generatePrimeNumbers() {
    p = new BigInteger(primeSize, 10,
rdm);
    do {
        q = new BigInteger(primeSize,
10, rdm);
```

```
    } while (q.compareTo(p) == 0);
}
```

Variabel *p* dan *q* sama-sama memiliki tiga parameter yang dikelola dalam kelas Integer. Adapun penjelasannya yaitu :

1. *primeSize,* ukuran panjang bit yang digunakan. Dalam hal ini, menggunakan panjang 8 bit.
2. *10*, paramater kedua ini menentukan derajat kepastian dalam menghasilkan bilangan prima. Nilai 0 berarti tingkat kepastian nya yaitu 0, nilai 1 menghasilkan peluang 0.5 sedangkan 10 memberi peluang 0.999.
3. rdm, adalah variabel atau objek dari kelas *SecureRandom* yang tugasnya yaitu mengacak nilai bulat.

Ketiga paramater tersebut saling terintegrasi, pertama mengacak nilai bulat dan menentukan panjang bit terus menentukan derajat kepastian bahwa bilangan bulat yang dihasilkan adalah bilangan prima.

Pada proses penetuan nilai *q*, terdapat *looping* yang selalu berputar selama nilai dari variabel *p* dan variabel *q* adalah sama sehingga menghasilkan nilai 0. Bernila positif jika objek pemanggil (*q*) lebih besar dan bernilai negatif jika objek yang dilewatkan (*p*) lebih besar.

4.2.2.2 Fungsi *generatePublicPrivateKeys*

Fungsi ini menghasilkan (*generate*) kunci publik dan kunci privat yang digunakan untuk enkripsi dan dekripsi pesan nantinya. Proses ini memberikan nilai pada variabel *n, r, e* dan *d*.

```
public void generatePublicPrivateKeys()
{
    // N = p * q
    N = p.multiply(q);
    // r = ( p - 1 ) * ( q - 1 )
    r =
p.subtract(BigInteger.valueOf(1));
    r =
r.multiply(q.subtract(BigInteger.valueO
f(1)));
    // Choose E, coprime to and less
than r
    do {
        E = new BigInteger(2 *
primeSize, rdm);
    } while ((E.compareTo(r) != -1)
||(E.gcd(r).compareTo(BigInteger.valueO
f(1)) != 0));
    // Compute D, the inverse of E mod
r
    D = E.modInverse(r);
}
```

Penjelasan fungsi di atas yaitu :

1. Melakukan proses perhitungan nilai *N* dengan mengkalikan variabel *p* dan variabel *q*.
2. Melakukan proses pencarian nilai *r* (*Φ(n)*). Pertama dengan mengurangi nilai p dengan bilangan *BigInteger*

1 kemudian disimpan di variabel *r* selanjutnya perkalian antara variabel *r* dengan variabel *q* yang sudah dikurangi dengan bilangan *BigInteger* 1.

3. Melakukan proses pencarian nilai *e* yang relatif prima dan kurang dari nilai *r*. Dalam prosesnya terdapat perulangan selama nilai *e* dibandingkan dengan nilai *r* (*compareTo*) menghasilkan nilai -1 atau nilai *gcd e* terhadap *r* dibandingkan dengan nilai *BigInteger* 1 adalah tidak sama dengan 0.
4. Setelah itu, mencari nilai *d* dengan menggunakan metode *inverseMod* atau pembalikan modulus dari variabel *E* terhadap variabel *r* yaitu melakuan pengembalian nilai *BigInteger* untuk di simpan ke varibel *D*.

4.2.2.3 Fungsi *encrypt*

Fungsi *encrypt* melakukan proses enkripsi pesan dengan nilai inputan berupa kelas *String*. Berikut adalah kode fungsi beserta penjelasannya.

```
public   BigInteger[]   encrypt(String
message) {
    byte[] temp = new byte[1];
    // menampung String message ke
array temp
    char[] digits =
message.toCharArray();
```

```
    // variabel untuk menampung dari
var temp
    BigInteger[] bigDigits = new
BigInteger[digits.length];
    for (int i = 0; i <
bigDigits.length; i++) {
        temp[0] = (byte) digits[i];
        bigDigits[i] = new
BigInteger(temp);
    }
    // variabel untuk menampung hasil
enkripsi
    BigInteger[] encrypted = new
BigInteger[bigDigits.length];
    for (int i = 0; i <
bigDigits.length; i++) {
        encrypted[i] =
bigDigits[i].modPow(E, N);
    }
    return (encrypted);
}
```

1. Mendeklarasikan variabel *temp* sebagai variabel array penampung.
2. Mendeklarasikan variabel *array char digits* yang nilainya dari variabel *message* yang di konversi ke bentuk array karakter.
3. Mendeklarasikan variabel *array BigInteger* dengan panjang *array* sesuai dengan panjang dari variabel *digits*.
4. Melakukan proses perulangan sebanyak panjang dari variabel *bigDigits*. Variabel *temp* manampung digits yang dikonversi ke bentuk integer. Selanjutnya nilai variabel temp disimpan di dalam variabel *bigDigits*.

5. Sesudah variabel *bigDigits* memperoleh semua nilai, kemudian dideklarasikan sebuah variabel *BigInteger* yaitu *enrypted* dengan panjang *array* sebanyak nilai yang ada dalam variabel *bigDigits*.
6. Proses enkripsi dilakukan dengan menggunakan metode *modPow* dan kunci publik (*E,N*). Dan dari proses enkripsi mengembalikan nilai array *BigInteger encypted*.

4.2.2.4 Fungsi *decrypt*

Sesudah melakukan proses enkripsi, maka tahap terakhir yaitu melakukan proses dekripsi. Proses dekripsi adalah pembalikan dari proses enkripsi yang sudah dilakukan di atas. Adapun kode dan penjelasannya yaitu :

```
public String decrypt(BigInteger[] 
encrypted, BigInteger D, BigInteger 
N) {
    int i;
    // menampung nilai array encrypted
    BigInteger[] decrypted = new 
BigInteger[encrypted.length];
    for (i = 0; i < decrypted.length; 
i++) {
        decrypted[i] = 
encrypted[i].modPow(D, N);
    }
    char[] charArray = new 
char[decrypted.length];
    for (i = 0; i < charArray.length; 
i++) {
        charArray[i] = (char) 
(decrypted[i].intValue());
```

```
        }
        return (new String(charArray));
    }
```

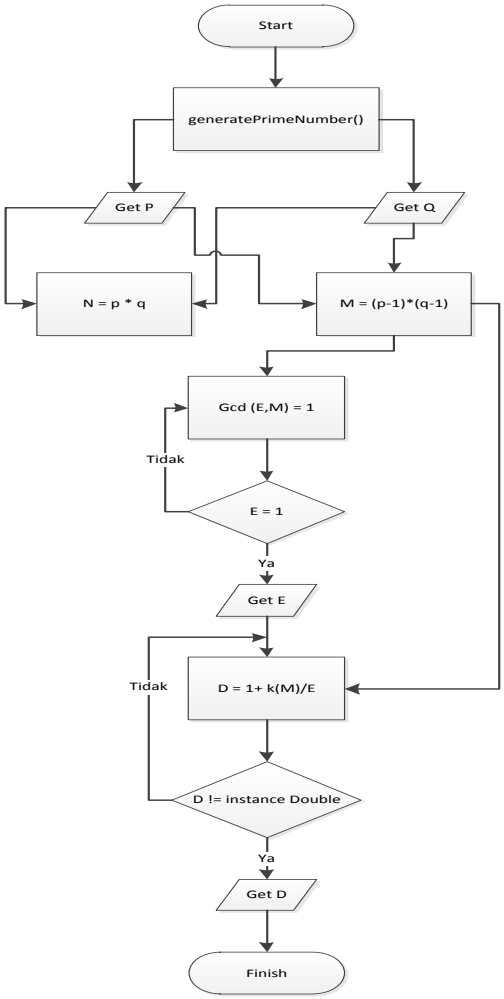
1. Mendeklarasikan variabel *i* dan variabel array *BigInteger decrypted* yang panjang arraynya adalah sama panjang dengan variabel *encrypted*.
2. Melakukan proses dekripsi setiap nilai array yang ada dalam variabel *encrypted* dengan menggunakan metode *modPow* dan kunci privat (*D*,*N*) dan hasilnya ditampung dalam variabel *decrypted*.
3. Variabel *decrypted* masih berupa objek *BigInteger* sehingga diperlukan suatu variabel yang dapat mengkonversi setiap nilai *decrypted* menjadi bentuk *char* yaitu dengan variabel *array charArray*.
4. Setiap nilai *charArray* adalah hasil dari konversi setiap nilai *decrypted*.
5. Fungsi ini mengembalikan nilai dari variabel *charArray* yang dirubah (*casting*) ke bentuk *String*.

4.3 Perancangan Algoritma untuk Proses *GenerateKey*, Enkripsi dan Dekripsi

Untuk memudahkan dalam membaca alur logika program, maka digunakan perancangan prosedur yang memberikan gambaran visual.

Terdapat tiga perancangan prosedur yaitu *flowchart* pembangkitan kunci, *flowchart* enkripsi dan *flowchart* dekripsi.

4.3.1 *Flowchart* Pembangkitan Kunci



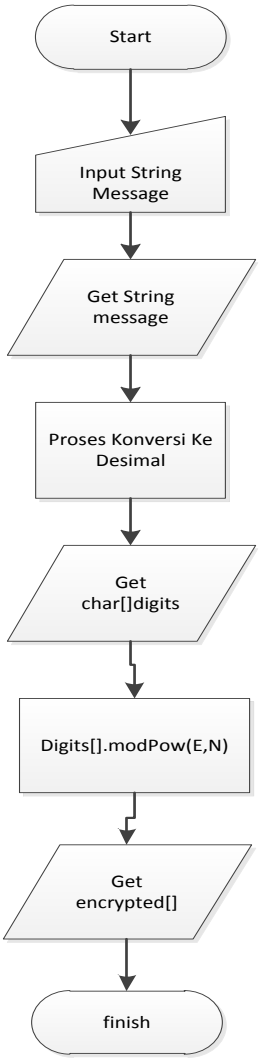
Gambar 4.3 Proses Pembangkitan Kunci

Berikut adalah penjelasan dari *flowchart* gambar 4.3 yaitu :

1. Membangkitkan variabel (besaran) *P* dan *Q* secara acak dan termasuk pada bilangan prima.

2. Setelah nilai *P* dan *Q* maka dapat dihitung nilai *N* dengan cara mengkalikan *P* dan *Q*. Disamping itu, juga dapat menghtung nilai *M* yaitu dengan cara mengkalikan (*P*-1) dengan (*Q*-1).
3. Setelah nilai *M* ditemukan, maka selanjutnya adalah mencari nilai E. Nilai E ini adalah nilai yang prima terhadap nilai *M* sehingga hasilnya adalah 1.
4. Selanjutnya mencari nilai *D* dengan nilai *D* yang bertipe *integer* atau bilangan bulat dengan rumus yang sudah ada. Ada konstanta *K* (dengan nilai 1, 2, 3, .. dan seterusya) yang dicobakan terus menerus sehingga nilai *D* tidak mempunyai desimal atau bilangan dibelakang koma.

4.3.2 *Flowchart* Enkripsi



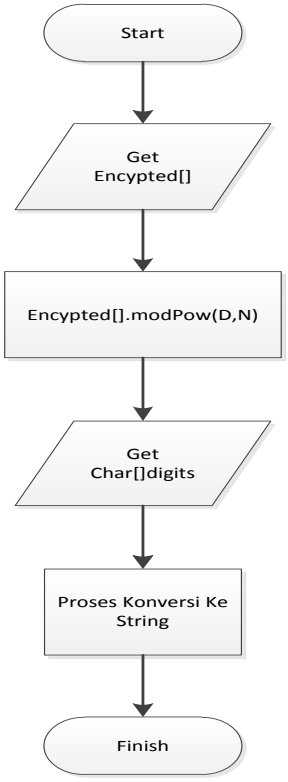
Gambar 4.4 Proses Enkripsi

Berikut adalah penjelasan dari *flowchart* gambar 4.4 yaitu :

1. Pertama kali sistem mendapatkan inputan berupa *String*.
2. Dari hasil masukan tersebut, kemudian dirubah (konversi) ke bentuk desimal ASCII sehingga dapat dilakukan proses enkripsi dan disimpan dengan variabel yang berbentuk *array*.

3. Sesudah nilai *String* dirubah ke bentuk desimal kemudian dilakukan proser enkripsi dengan menggunakan metode *modPow()* yang menggunakan kunci publik (*E*,*N*) dan hasilnya disimpan dalam setiap elemen pada variabel *array encrypted*.

4.3.3 *Flowchart* Dekripsi



Gambar 4.5 Proses Dekripsi

Berikut adalah penjelasan dari *flowchart* gambar 4.5 yaitu :

1. Dari hasil proses enkripsi yang disimpan pada variabel array *encrypted* maka dilakukan proses dekripsi. Masih menggunakan metode *modPow()* tapi dengan parameter kunci

privat (*D,N*). Hasilnya disimpan pada variabel array *digits* dengan tipe *Char*.

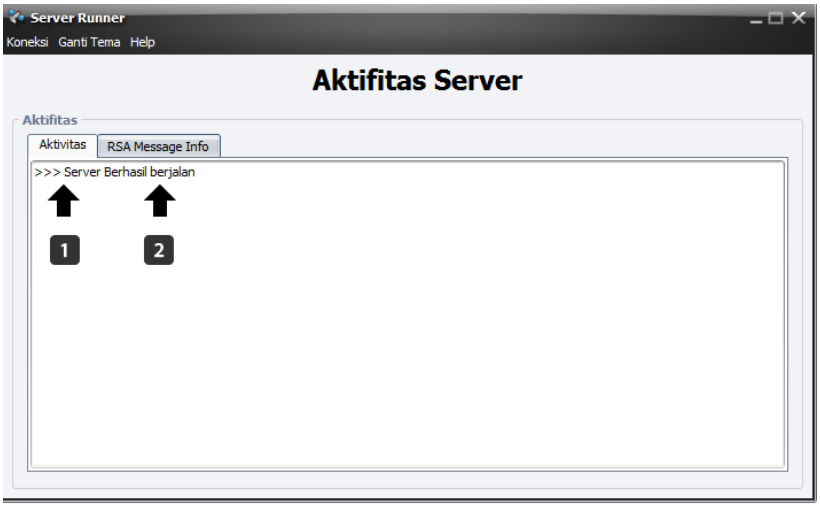
2. Supaya dapat dibaca oleh penerima pesan tersebut (Nilai *String* yang dikirim) maka nilai desimal tersebut dirubah kebentuk *String*.

4.4 Perancangan AntarMuka (*UI / User Interface*)

Perancangan antarmuka berisi pemaparan *user interface* yang digunakan dalam perancangan aplikasi. Dengan perancangan antarmuka ini diharapkan dapat memudahkan pengguna dalam mengoperasikan aplikasi di banding dengan menggunakan baris perintah (*command*) yang mengharuskan pengguna untuk menghapal setiap perintah untuk dapat mengeksekusi suatu program. Perancangan antarmuka terbagi menjadi dua, yaitu perancangan antarmuka *server* dan perancangan antarmuka *client*.

4.4.1 Perancangan Antarmuka Server

Server bertugas untuk mendengarkan (*listening*) permintaan dari *client* atau sebagai jembatan perantara yang menghubungkan dua *client* atau lebih. Adapun perancangan antarmuka untuk *server* yaitu :



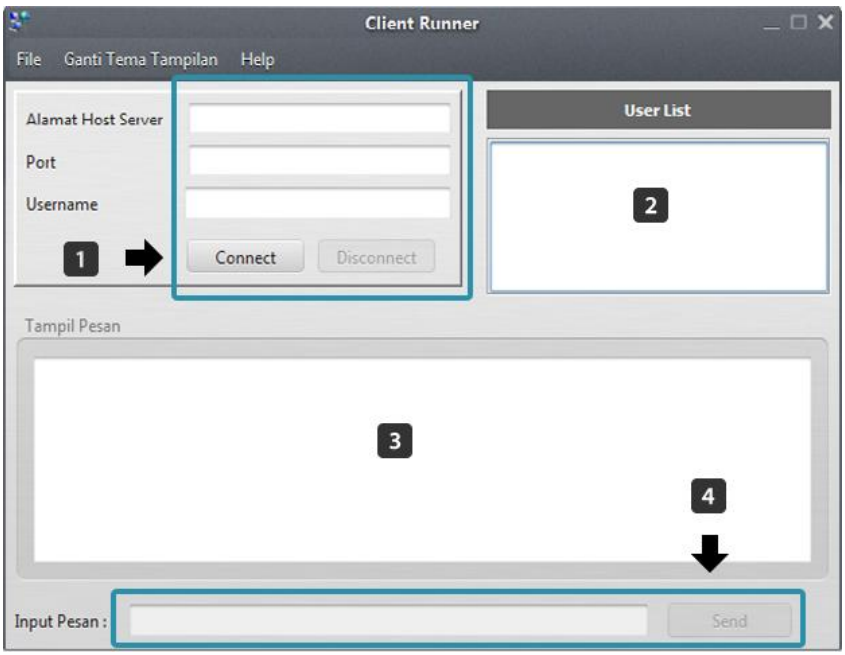
Gambar 4.6 Perancangan Antarmuka Server

Berikut adalah penjelasan dari antarmuka *server* gambar 4.6 :

1. Pada nomor 1, adalah teks area aktifitas status *client*. Baik itu yang yang memberi info *client* terhubung, *client* meninggalkan percakapan atau *server* terlah *off*.
2. Pada nomor 2, adalah teks area yang memberikan besaran-besaran (*p,q,n,e,r,d, ciphertext* dan *plaintext*) algoritma RSA yang digunakan pada saat mengirim dan menerima pesan.

4.4.2 Perancangan Antarmuka Client

Disamping kebutuhan perancangan antarmuka untuk *server*, maka perlu untuk melakukan perancangan di sisi *client*. Sehingga memudahkan *client* dalam mengoperasikan aplikasi tersebut.



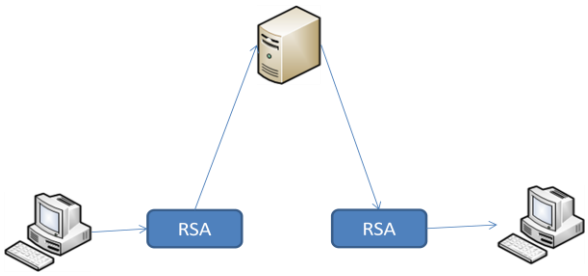
Gambar 4.7 Perancangan Antarmuka *Client*

Berikut adalah penjelasan dari gambar 4.7 :

1. Pada nomor 1, adalah tempat untuk mengisi alamat *IP Address server*, nilai *port* yang dibuka oleh *server* dan username yang akan digunakan untuk menampung daftar *client* yang terhubung.
2. Pada nomor 2, adalah daftar *client* yang telah terhubung ke *server*.
3. Pada nomor 3, adalah teks area tempat *client* membaca pesan yang dikirim dan pesan yang diterima.
4. Pada nomor 4, adalah tempat untuk mengisi pesan yang akan dikirim. Pesan diketik pada *textfield* kemudian untuk mengirimnya tekan "*Enter*" atau klik tombol "*Send*".

4.5 Gambaran Umum Sistem dalam Jaringan

Sistem yang digunakan diasumsikan menggunakan konsep *client-server* dengan 1 server dan minimal 2 komputer client. Berikut adalah gambaran sistem :



Gambar 4.8 Gambaran Sistem dalam Jaringan

Client menggunakan aplikasi *instant messenger* yang telah dibangun sehingga proses mengirim dan menerima pesan harus melewati RSA (proses enkripsi dan dekripsi pesan).

4.6 Distribusi Kunci

Selain pesan yang dienkripsi untuk menjaga keamanan, kunci juga harus diatur sedemikian rupa sehingga kunci yang dilakukan untuk mendekripsi dapat diamankan juga.

`java.rmi.Remote` adalah sebuah interface yang mengenkapsulasi setiap kunci yang digunakan. Kunci dikirimkan secara *by pass* artinya kunci dikirimkan langsung secara serentak. Dengan kemampuan RMI yang dapat mengunduh *bytecode* dari suatu objek *Class*, jika *class* tersebut tidak terdefinisikan di VM-nya penerima. Selain itu,

pesan yang dikirim dibentuk menjadi *stub object* ke VM Server dan dari VM server dibentuk *skeleton object*. Mekanisme ini menggunakan proses yang disebut *Marshalling* sehingga proses distribusi dipercayakan pada VM Java yang sudah *include* keamanan RSA untuk proses RMI nya.