BAB III

ANALISIS DAN PERANCANGAN

III.1 Analisis Masalah

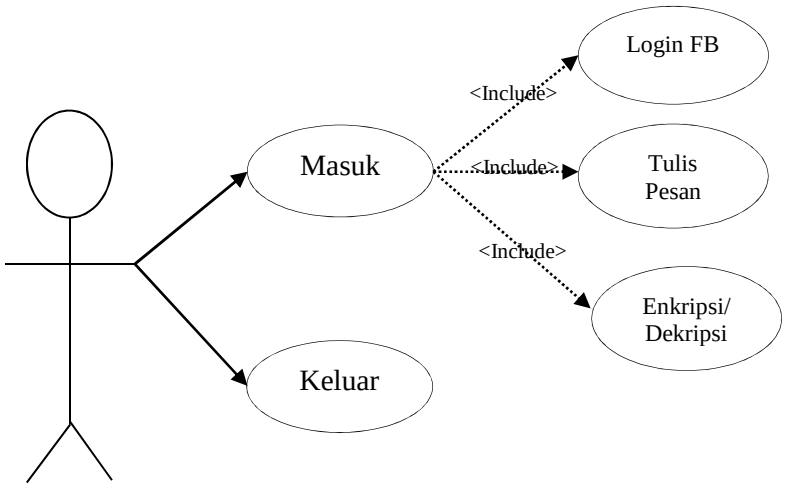
Adapun masalah yang penulis temukan yaitu tentang keamanan pesan berupa teks (*string*) pada *facebook messenger*, dimana kerahasia pesan tersebut tidak terjamin dan pesan masih dapat dibaca oleh pihak-pihak yang tidak bertanggung jawab. Maka dari pada itu menulis mencoba menerapkan kriptografi kedalam *facebook messenger* dengan menggunakan algoritma *Gronsfeld cipher* dan *RC4* yang nantinya akan diaplikasikan untuk merubah teks asli (*plaintext*) ke dalam bentuk pesan terenkripsi (*ciphertext*).

III.2. Perancangan Sistem

Perancangan sistem yang diusulkan meliputi perancangan diagram dan perancangan antarmuka program. Perancangan diagram terdiri dari perancangan *use case diagram*, perancangan *activity diagram* dan *squence diagram*. Perancangan antarmuka program terdiri dari *form* Masuk (Tampilan awal aplikasi) dan *form* halaman utama, didalam *form* halaman utama terdapat fungsi enkripsi dan dekripsi dan *facebook messeger* sebagai media pengirim pesan. Perancangan diagram yang diusulkan dimaksudkan untuk menjelaskan alur kegiatan dari proses yang terjadi pada sistem yang diusulkan, sedangkan perancangan antarmuka program dimaksudkan untuk menampung masukan user dan sebagai tempat interaksi user dengan sistem.

III.2.1. *Use Case Diagram*

Secara garis besar, proses perancangan sistem yang akan dirancang digambarkan dengan *UseCase Diagram* sebagai berikut :

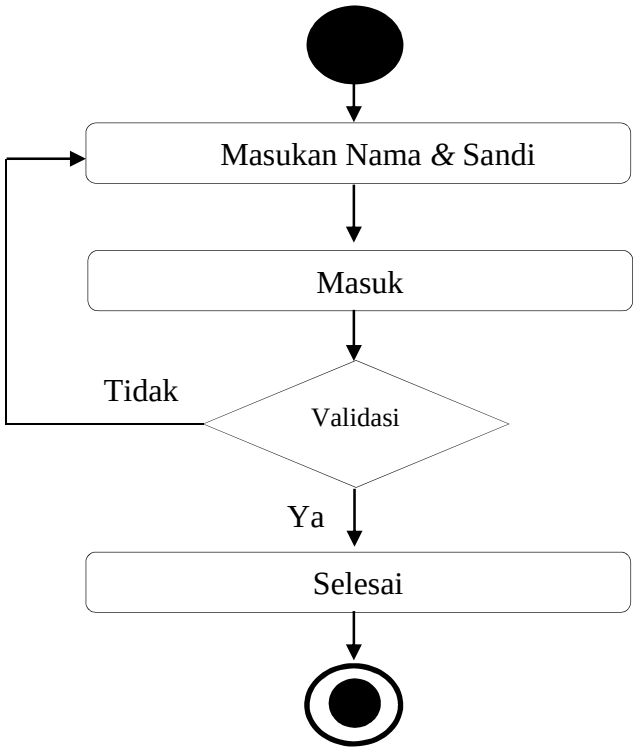


Gambar: III.1 *UseCase Diagram* Sistem

III.2.2. *Activity Diagram*

III.2.2.1. *Activity Diagram* Masuk

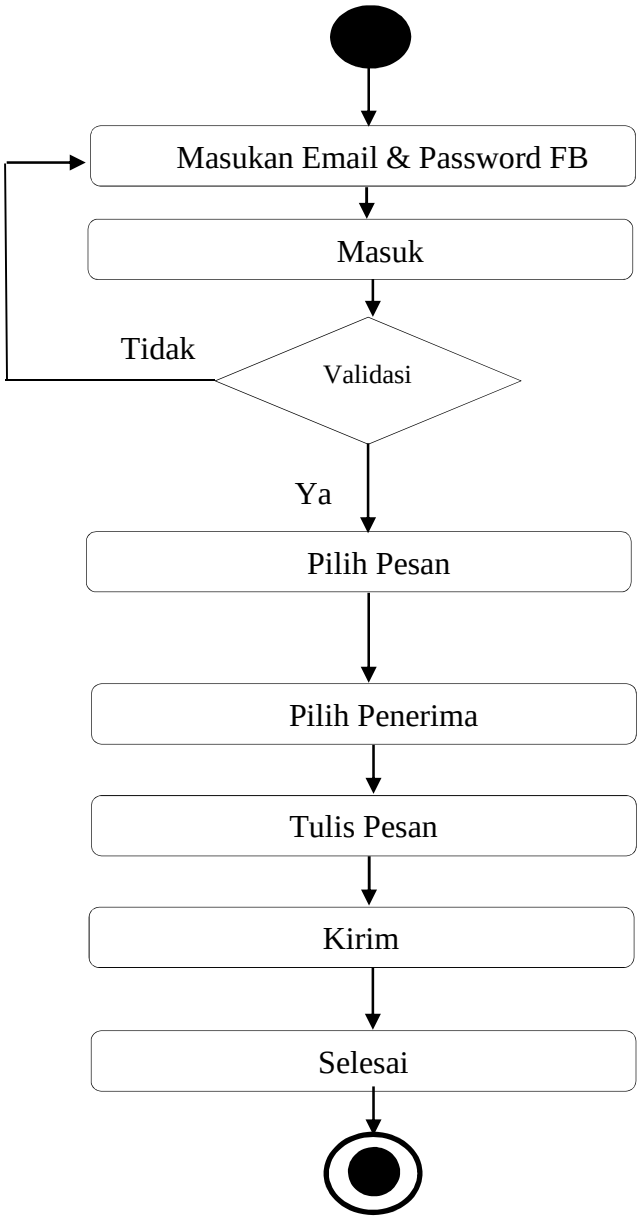
Berikut ini merupakan *Activity Diagram* Masuk yang menjelaskan proses Masuk dari *Usecase Diagram* diatas:



Gambar : III.2. *Activity Diagram* Masuk Aplikasi

III.2.2.2. *Activity Diagram Login Fb* dan Tulis Pesan

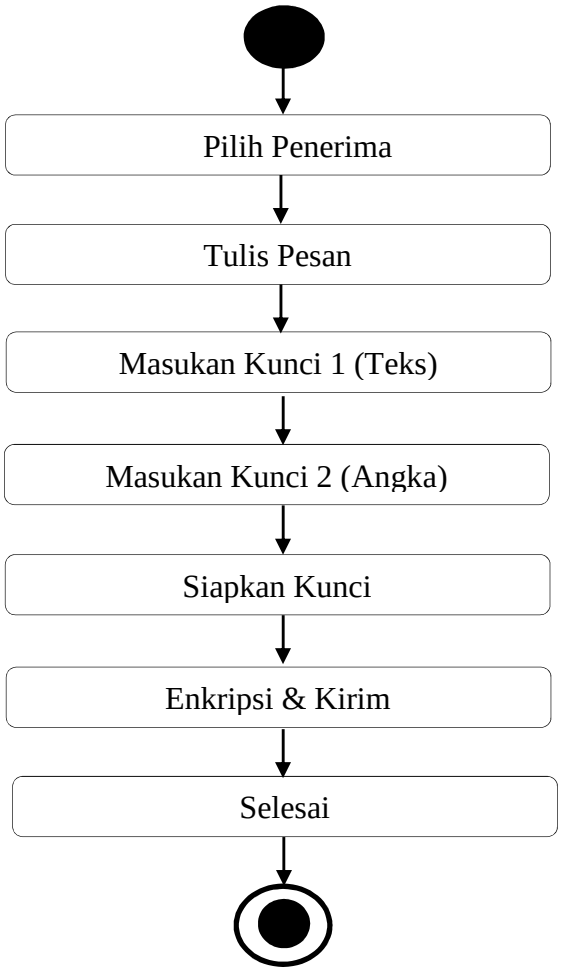
Berikut merupakan *Activity Diagram* tulis pesan yang menjelaskan proses menulis pesan dari *Usecase Diagram* sebelumnya :



Gambar: III.3. *Activity Diagram* Tulis Pesan

III.2.2.3. *Activity Diagram* Enkripsi Pesan

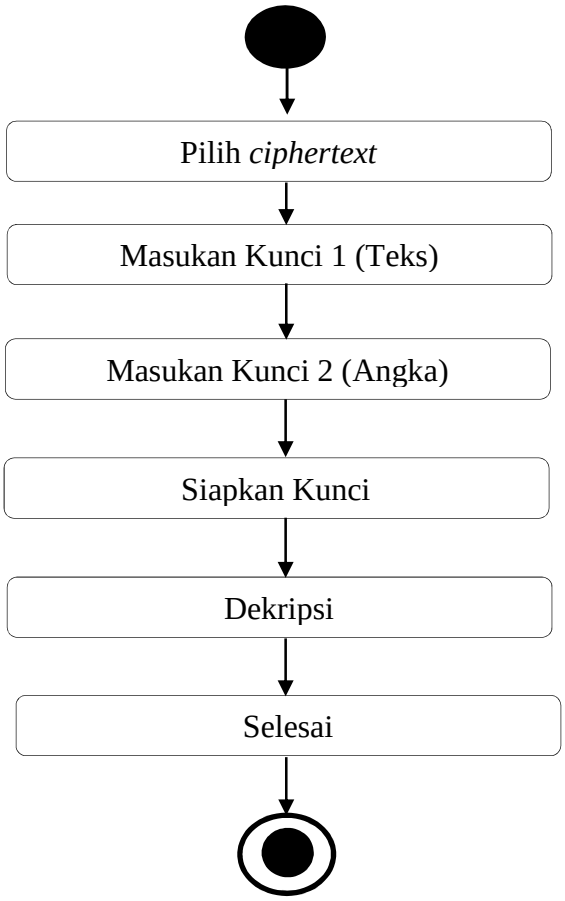
Berikut merupakan *Activity Diagram* Enkripsi pesan yang menjelaskan proses pengenkripsian pesan dari *Usecase Diagram* sebelumnya :



Gambar: III.4. *Activity Diagram* Enkripsi Pesan

III.2.2.4. *Activity Diagram* Dekrispsi Pesan

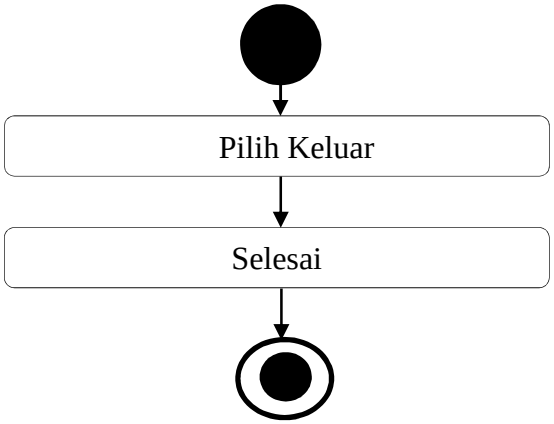
Berikut merupakan *Activity Diagram* Dekripsi pesan yang menjelaskan proses pendekripsisan pesan dari *Usecase Diagram* sebelumnya :



Gambar: III.5. *Activity Diagram* Dekripsi Pesan

III.2.2.5. *Activity Diagram* Keluar

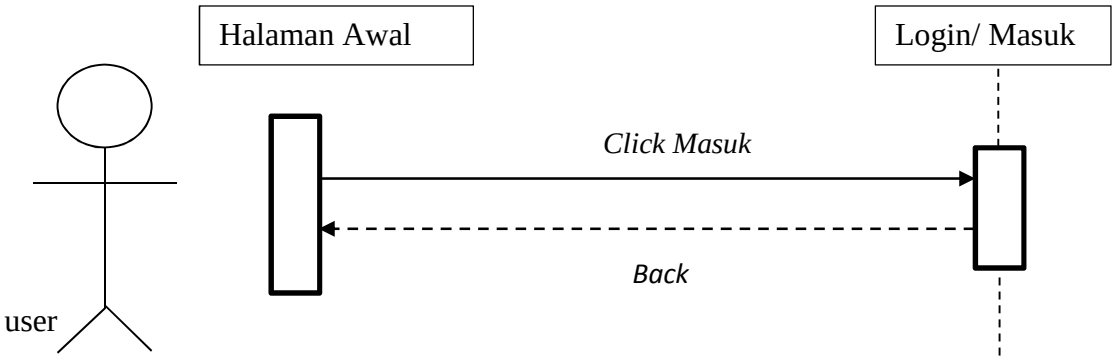
Adapun *Activity Diagram* yang menjelaskan proses keluar adalah sebagai berikut :



Gambar:III.6 *Activity Diagram* Keluar

III.2.3 *Squence Diagram*

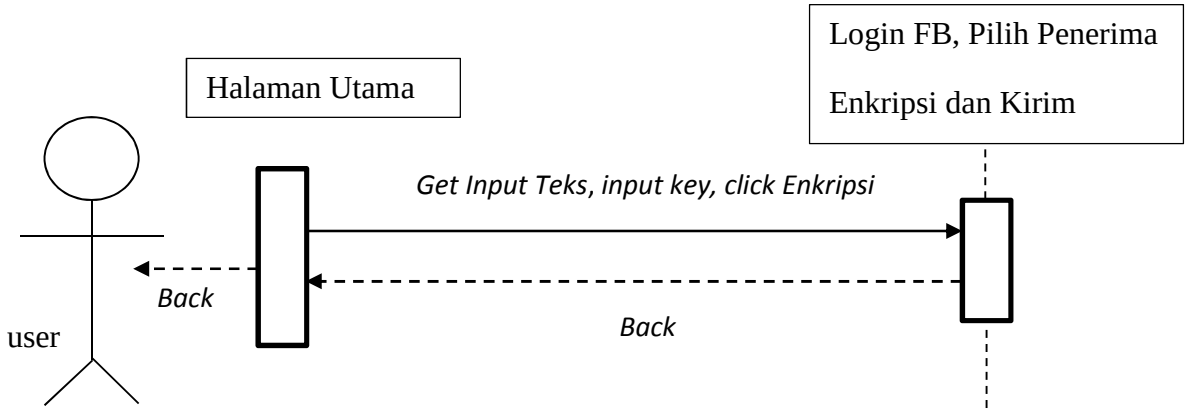
III.2.3.1 *Squence Diagram* Masuk



Gambar:III.7 *Squence Diagram* Masuk

Dari gambar III.7 diatas menunjukan bahwa seorang user yang ingin masuk kedalam aplikasi harus terlebih dahulu berada di halaman awal, kemudian mengklik tombol masuk untuk masuk ataupun login.

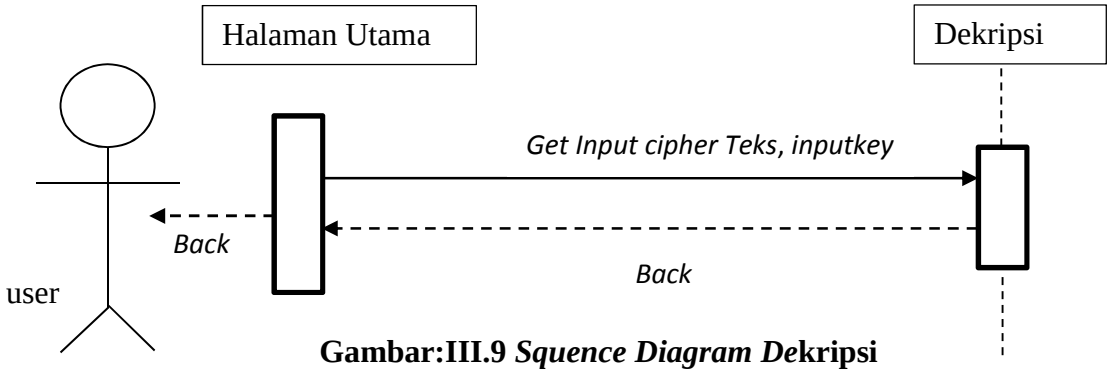
III.2.3.2 *Squence Diagram* Enkripsi



Gambar:III.8 *Squence Diagram* Enkripsi

Dari gambar III.8 diatas, setelah *user* berada dihalaman utama , maka user dapat melakukan login FB, memilih penerima dan menulis pesan dengan fungsi enkripsi teks yaitu dengan menginputkan teks dan kunci untuk proses pengenkripsian dan pengiriman pesan.

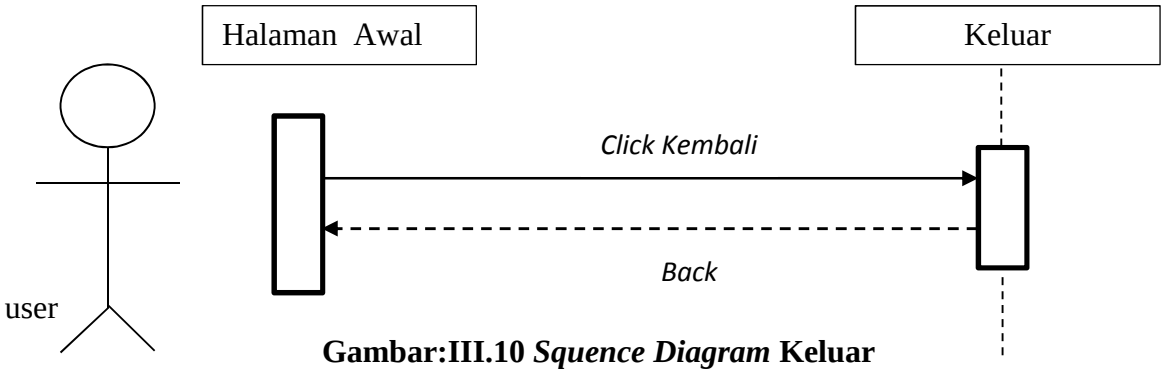
III.2.3.3 *Squence Diagram* Dekripsi



Gambar:III.9 *Squence Diagram De*kripsi

Dari gambar III.9 ditas, user yang berada di halaman utama dapat mendeskripsikan *cipherteks* yang diterima dengan *key* yang sebelumya sudah diketahui bersama.

III.2.3.4 *Squence Diagram* Keluar



Gambar:III.10 *Squence Diagram* Keluar

Dari gambar III.10 diatas, untuk keluar dari aplikasi seorang user harus kembali kehalaman awal dan mengklik tombol keluar.

III.3. Algortima *Gronsfeld Cipher*

Algoritma *gronsfeld cipher* merupakan salah satu algortima subtitusi simetris yang identik dengan algoritma subtitusi simetris lainnya yang cukup terkenal, yaitu *vigenere cipher*. Diciptakan oleh seorang kriptografer abad ke-17 di Jerman, bernama Gaspar Schot yang mempelajari cipher ini selama perjalannya dari Mainz ke Frankfurt, Jerman.

Gronsfeld identik dengan *vigenere cipher*, tetapi perbedaan nya adalah pada kunci *cipher* ini yang menggunakan angka 0-9 dan hanya berlaku untuk huruf kapital saja. Kelebihan *gronsfeld cipher* adalah kuncinya yaitu bukan sebuah kata tetapi kumpulan angka, tetapi kelemahannya cipher ini hanya mempunyai 10 kemungkinan, tidak seperti *vigenere* yang memiliki 26

kemungkinan karena menggunakan alfabet. *Gronsfeld* sendiri umumnya lebih banyak digunakan di Jerman dan beberapa negara lain di Eropa.

III.3.1. Enkripsi *Gronsfeld Cipher*

Adapun penerapan algoritma gronsfeld cipher ini adalah sebagai berikut:

Plainteks : D O D I

Binner *Plaintext* : 01000100 01001111 01000100 01001001

Desimal *Plaintext* : 68,79,68,73

Kunci : 1 2 3 4 +

Desimal Enkripsi : 69 81 71 77

Ciphertext : E Q G M

Binner *Ciphertext* : 0100 0101 0101 0001 0100 0111 0100 1101

Dalam melakukan proses enkripsi teks, tentukan *plaintext* yang akan dienkripsi kemudian ubah dalam bentuk kapital/huruf besar. Kemudian tentukan pula kuncinya, yaitu dalam bentuk angka. Apabila panjang kunci tidak sama dengan panjang *plaintext*, maka kunci yang ada diulang secara priodik sehingga jumlah karakter kuncinya sama dengan jumlah *plaintext*nya.

Ubah bentuk *plaintext* tersebut kedalam bentuk desimal, ditambahkan dengan kunci. Hasil dari pejumlahan tersebut diubah kembali ke bentuk karakter.

III.3.2. Dekripsi *Gronsfeld Cipher*

Dekripsi adalah proses sebaliknya, dimana *ciphertext*nya diubah menjadi nilai desimal, kemudian dikurangi kunci kemudian dikembalikan lagi ke dalam bentuk karakter.

Ciphertext	: E Q G M
Binner *Ciphertext*	: 0100 0101 01010001 01000111 01001101
Desimal Enkripsi	: 69 81 71 77
Kunci	: 1 2 3 4
Desimal *Plaintext*	: 68 79 68 73
Plainteks	: D O D I
Binner *Plaintext*	: 01000100 01001111 01000100 01001001

III.4. Algoritma *RC4*

III.4.1 Algoritma *RC4* (Enkripsi)

Berikut adalah implementasi algoritma RC4 dengan mode 4 *byte* (untuk lebih menyederhanakan dalam perhitungan manual) serta untuk kebutuhan sistem yang sangat terbatas. S-Box dengan panjang 4 byte, dengan S[0]=0, S[1]=1, S[2]=2 dan S[3]=3 sehingga array S menjadi:

0 1 2 3

Inisialisasi 4 byte kunci array, K. Misalkan kunci Ulang kunci sampai memenuhi seluruh adalah 2 5 7 3, sehingga array K berisi 2 5 7 3 dan mencoba untuk mengenkripsikan kata HALO.

Inisialisasi i dan j dengan 0 kemudian dilakukan KSA agar tercipta state-array yang acak. Penjelasan iterasi lebih lanjut dapat dijelaskan sebagai berikut:

Iterasi 1

i = 0

j = (0 + S[0] + K [0 mod 4]) mod 4

= (0 + 0 + 2) mod 4 = 2

Swap (S[0],S[2])

Hasil Array S

2 1 0 3

Iterasi 2

i = 1

j = (2 + S[1] + K [1 mod 4]) mod 4

= (2 + 1 + 5) mod 4 =0

Swap (S[1],S[0])

Hasil Array S

1 2 0 3

Iterasi 3

i = 2

j = (0 + S[2] + K [2 mod 4]) mod 4

= (0 + 0 + 7) mod 4 = 3

Swap (S[2],S[3])

Hasil

1 2 3 0

Iterasi 4

i = 3

j = (3 + S[3] + K [3 mod 4]) mod 4

= (3 + 0 + 3) mod 4 = 2

Swap (S[3],S[2])

Hasil Array S

1 2 0 3

Setelah melakukan KSA, akan dilakukan PRGA. PRGA akan dilakukan sebanyak 4 kali dikarenakan plainteks yang akan dienkripsi berjumlah 4 karakter. Hal ini disebakan karena dibutuhkan 1 kunci dan 1 kali pengoperasian XOR untuk tiap tiap karakter pada plainteks. Berikut adalah tahapan penghasilan kunci enkripsi dengan PRGA.

Array S

1 2 0 3

Inisialisasi

i = 0

j = 0

Iterasi 1

i = (0 + 1) mod 4 = 1

j = (0 + S[1]) mod 4 = (0+ 2) mod

4 = 2

swap (S[1],S[2])

1 0 2 3

K1 = S[(S[1]+S[2]) mod 4] = S[2

mod 4] = 2

K1 = 00000010

Iterasi 2

i = (1 + 1) mod 4 = 2

j = (2 + S[2]) mod 4 = (2+ 2) mod

4 = 0

swap (S[2],S[0])

2 0 1 3

K2 = S[(S[2]+S[0]) mod 4] = S[3

mod 4] = 3

K2 = 00000011

Iterasi 3

i = (2 + 1) mod 4 = 3

j = (0 + S[3]) mod 4 = (0+ 3) mod

4 = 3

swap (S[3],S[3])

1 0 2 3

K3 = S[(S[3]+S[3]) mod 4] = S[6

mod 4] = 2

K3 = 00000010

Iterasi 4

i = (3 + 1) mod 4 = 0

j = (3 + S[0]) mod 4 = (3+ 1) mod

4 = 0

swap (S[0],S[0])

1 0 2 3

K1 = S[(S[0]+S[0]) mod 4] = S[2

mod 4] = 2

K4 = 00000010

Setelah menemukan kunci untuk tiap karakter, makadilakukan operasi XOR antara karakter pada plaintext dengan kunci yang dihasilkan. Berikut adalah tabel ASCII untuk tiap-tiap karakter pada plaintks yang digunakan.

Huruf	Kode ASCII (Binary 8 bit)
H	01001000
A	01000001
L	01001100
O	01001111

Berikut adalah proses pengXORan dari plainteks dengan key yang telah didapat:

H A L O : 01001000 01000001 01001100 01001111

Key : 00000010 00000011 00000010 00000010

Cipherteks : 01001010 01000010 01001110 01001101

III.3.2 Algoritma *RC4* (Dekripsi)

Proses dekripsi *ciphertext* menggunakan algoritma RC4 ini sama untuk proses *key-schedule*-nya. Untuk mendapatkan *plaintext*, *ciphertext* yang diperoleh di XORkan dengan *pseudo random byte* yang didapat sebelumnya. Maka hasilnya adalah plainteks atau teks asli.

Pesan dikirim dalam bentuk cipherteks sehingga setelah sampai di penerima pesan dapat kembali diubah menjadi plainteks dengan meng-XOR-kan dengan kunci yang sama. Pemrosesan pesan setelah sampai pada penerima dapat dilihat pada dibawah ini.

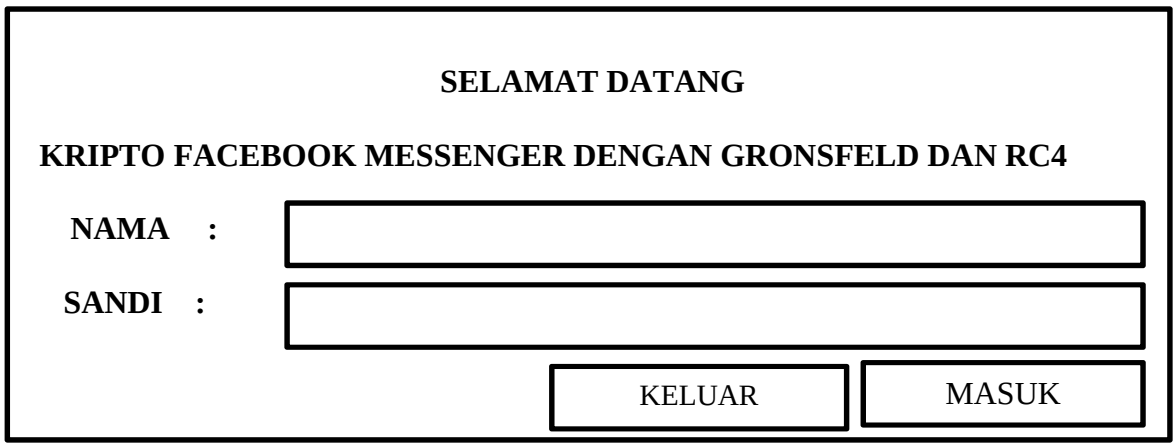
Proses XOR *pseudo random byte* dengan cipherteks pada dekripsi yaitu:

Cipherteks	: 01001010	01000010	01001110	01001101
pseudo random byte	**:** 00000010	00000011	00000010	00000010
Plainteks	: 01001000	01000001	01001100	01001111
	H	A	L	O

III.4. Perancangan *Interface*

III.4.1 Perancangan *form* Masuk (Tampilan Awal)

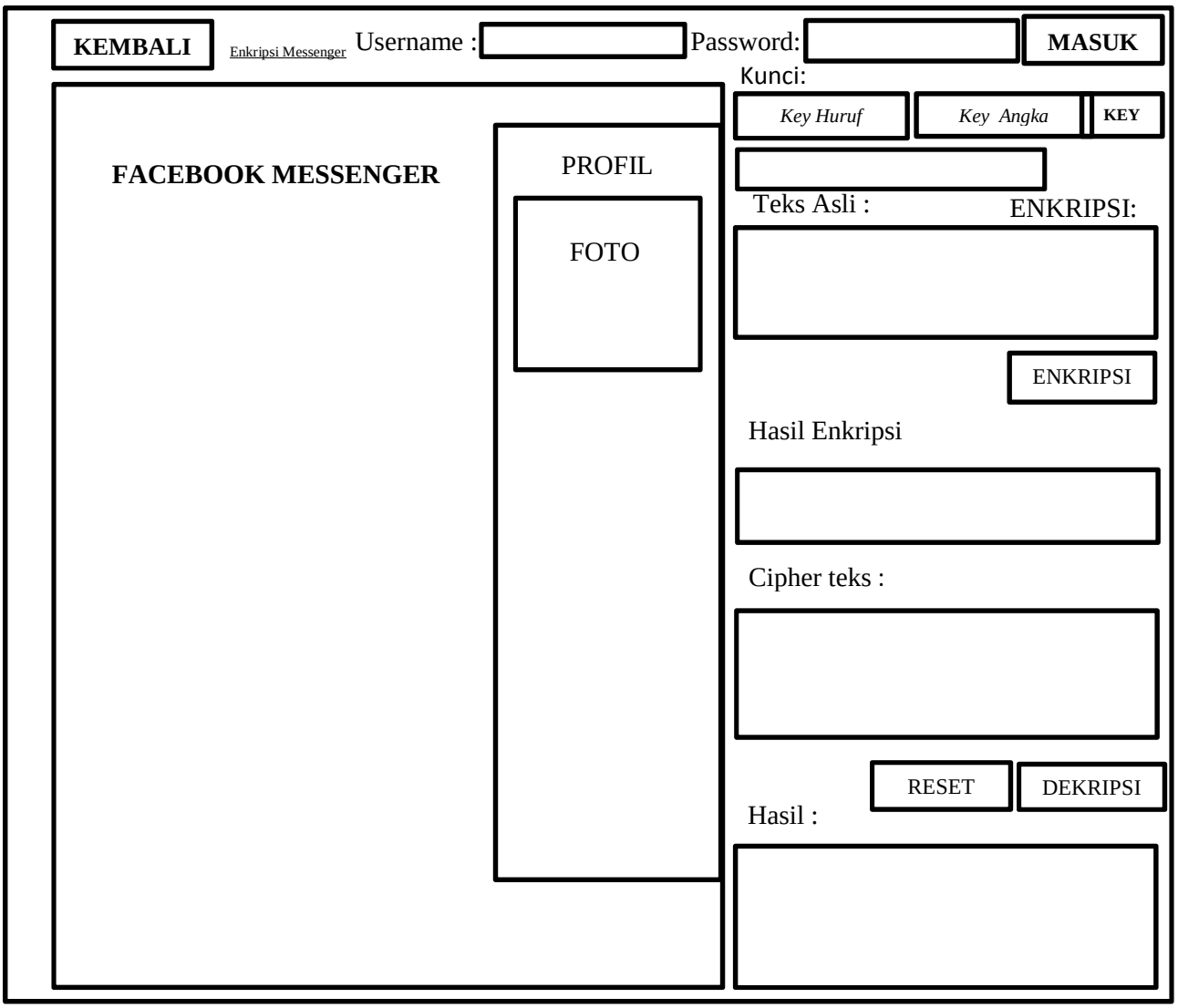
Berikut adalah perancangan *interface* untuk halaman utama perancangan keamanan *messenger* pada media sosial facebook:



Gambar:III.11 Tampilan awal Aplikasi.

III.4.2 Perancangan Halaman Utama

Berikut ada perancangan tampilan dari halaman utama Aplikasi keamanana *messenger* pada media sosial *facebook* dengan metode *gronsfeld chipher* dan RC4:



Gambar:III.12 Tampilan *Form* Utama