Modifikasi Vigenère Cipher dengan Metode Penyisipan Kunci pada Plaintext

Kevin Leonardo Handoyo/13509019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509019@std.stei.itb.ac.id

***Abstrak* — Kriptografi merupakan metode penyembunyian pesan. Penyembunyian pesan merupakan hal yang sangat penting saat kita hanya ingin penerima pesan saja yang mengetahui isi pesan yang kita sampaikan, Ada banyak sekali metode kriptografi yang telah digunakan, baik yang klasik maupun yang modern. Kriptografi klasik adalah kriptografi yang sudah tidak aman lagi karena dapat dipecahkan dengan metode-metode tertentu. Salah satu algoritma kriptografi klasik adalah Vigenère Cipher. Algoritma ini dapat dipecahkan dengan menggunakan metode Kasiski/Kerckhoff dan analisis frekuensi. Karena itulah, pada makalah ini akan dibahas sebuah modifikasi dari Vigenère Cipher, yaitu dengan cara menyisipkan huruf-huruf dari kata kunci yang diberikan ke dalam plaintext. Dengan metode ini, diharapkan tingkat keamanan dari Vigenère Cipher ini bertambah sehingga dapat lebih berguna dalam penggunaannya untuk penyembunyian pesan yang tidak terlalu penting. Dalam makalah ini akan dibahas bagaimana modifikasi ini dapat mengatasi serangan-serangan seperti metode Kasiski/Kerckhoff dan analisis frekuensi.**

***Kata Kunci*— Vigenère Cipher, metode Kasiski, analisis frekuensi.**

I. PENDAHULUAN

Dalam perkembangan ilmu kriptografi, telah banyak sekali algoritma yang diciptakan untuk menyembunyikan pesan. Algoritma-algoritma tersebut saat ini dapat dikelompokkan menjadi 2 kelompok besarm yaitu algoritma kriptografi klasik dan modern. Suatu algoritma kriptografi tergolong kriptografi klasik apabila ia telah berhasil dipecahkan dengan menggunakan metode-metode tertentu. Biasanya, algoritma kriptografi klasik mengandalkan substitusi, baik itu substitusi abjad tunggal maupun beberapa abjad. Beberapa contoh algoritma kriptografi klasik yaitu Caesar's Cipher, Vigenère Cipher, Playfair Cipher, Hill Cipher, dll.

Algortima-algoritma kriptografi yang menggunakan metode substitusi tersebut memiliki sebuah kelemahan umum, yaitu rentan terhadap serangan yang menggunakan analisis frekuensi, baik itu menghitung huruf, bigram, ataupun trigram yang paling sering muncul dalam suatu bahasa. Meskipun begitu, untuk beberapa algoritma kriptografi klasik, tidak cukup dengan menggunakan analisis frekuensi saja, melainkan harus digabung dengan teknik-teknik lain, seperti metode Kasiski yang dilanjutkan dengan metode Kerckhoff dalam memecahkan Vigenère Cipher.

Karena Vigenère Cipher telah dengan mudah dapat dipecahkan, maka penulis ingin mencoba memperbaiki algoritma kriptografi ini dengan melakukan sebuah modifikasi, yaitu dengan menyisipkan huruf-huruf pada kunci ke dalam plaintext. Dengan metode ini, diharapkan algoritma ini menjadi lebih aman terhadap serangan kriptanalis, khususnya serangan menggunakan analisis frekuensi dan metode Kasiski.

Sebenarnya, cara paling aman untuk menggunakan Vigenère Cipher adalah dengan menggunakan *one-time pad*, yaitu dengan menggunakan kunci yang panjangnya sama dengan panjang plaintext, contohnya dengan menggunakan *autokey*. Tetapi, disini penulis hanya ingin menguji modifikasi ini dan membandingkan tingkat keamanannya dengan Vigenère Cipher biasa(tanpa *autokey*) dalam menghadapi serangan yang menggunakan analisis frekuensi dan metode Kasiski.

Batasan berikutnya yaitu program yang sudah saya buat merupakan program untuk enkripsi menggunakan 256 karakter ASCII, dimana semua karakter termasuk spasi juga dienkripsi.

II. DASAR TEORI

A. Vigenère Cipher

Vigenère Cipher adalah sebuah metode enkripsi yang berbentuk substitusi polyalphabet. Metode ini merupakan perkembangan dari Caesar Cipher, dimana substitusi tidak hanya digeser sekian huruf untuk semua plaintext, melainkan menggunakan banyak tabel pergeseran yang bergantung pada huruf yang terdapat di kata kunci.

Vigenère Cipher sebenarnya pertama kali dikemukakan oleh Giovan Battista Bellaso pada tahun 1553 dalam bukunya, yaitu *La cifra del. Sign. Giovan Battista Bellaso*, namun penemuan metode ini diberikan kepada orang yang salah pada abad ke-19, yaitu kepada Blaise de Vigenère, sehingga nama metode ini dikenal sebagai Vigenère Cipher. Peran Vigenère sendiri dalam *cipher* ini adalah tentang penggunaan *autokey*.

Pada masa kejayaannya, *cipher* ini memiliki reputasi sebagai *cipher* yang tidak dapat dipecahkan dan seringkali

ditinggikan akan hal tersebut. Namun reputasi ini sebenarnya tidaklah benar. Banyak orang yang telah berhasil memecahkan *cipher* ini. Charles Babbage telah berhasil memecahkannya sekitar tahun 1854, namun ia tidak mempublikasikan karyanya tersebut. Selain itu, banyak juga kriptanalis lain yang kadang-kadang berhasil memecahkan *cipher* ini dari sejak abad ke-16. Barulah pada tahun 1863, Friedrich Kasiski memecahkan *cipher* ini sepenuhnya dan mempublikasikan karyanya tersebut. Metode yang dipublikasikannya tersebut dikenal sebagai metode Kasiski.

Metode enkripsi yang digunakan pada Vigenère Cipher adalah dengan menggunakan *tabula recta*/Tabel Vigenère, yaitu tabel pergeseran huruf-huruf pada *plaintext* terhadap huruf-huruf pada kunci.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Gambar 1. *Tabula recta*/Tabel Vigenère

Sebagai contoh, untuk *plaintext* TES SATU DUA TIGA denga kata kunci ALOHA, dengan mengacu pada tabel akan menghasilkan *ciphertext* sebagai berikut:

plaintext : TES SATU DUA TIGA
kunci : ALO HAAL OHA ALOH
ciphertext : TPG ZATF RBA TTUH

Selain menggunakan tabel Vigenère, cara enkripsi Vigenère Cipher dapat juga menggunakan rumus sebagai berikut:

$$C_i \equiv (P_i + K_i) \mod 26$$

dimana C_i adalah huruf ke-i pada *ciphertext*, P_i adalah huruf ke-i pada *plaintext*, dan K_i adalah huruf ke-i pada kunci. Setiap huruf tersebut diwakilkan dengan bilangan 0-25 yang masing-masing merupakan enumerasi dari abjad A-Z.

Proses dekripsi pada Vigenère Cipher juga dapat ditemukan dengan menggunakan rumus yang merupakan kebalikan dari rumus enkripsi, yaitu:

$$P_i \equiv (C_i - K_i) \mod 26$$

Perlu diperhatikan bahwa angka 26 disini disebabkan karena domain huruf yang bisa dienkripsi hanya abjad biasa, yaitu 26 huruf. Apabila domain karakter yang dienkripsi misalnya merupakan karakter ASCII, maka angka 26 diganti dengan 256.

Ada banyak sekali varian pada Vigenère Cipher ini. Beberapa diantaranya adalah *autokey* Vigenère Cipher dan Full Vigenère Cipher. Pada *autokey* Vigenère Cipher, kata kunci tidak mengalami pengulangan, melainkan apabila kata kunci telah habis, akan dilanjutkan dengan huruf-huruf pada *plaintext*. Namun, metode inipun telah berhasil dipecahkan oleh Babbage. Tetapi, apabila kunci memiliki panjang yang sama dengan *plaintext*, serta kunci random(bukan mengikuti *plaintext*), maka *chiper* tersebut tergolong *one-time pad* dan tidak dapat dipecahkan. Full Vigenère Cipher hanya dapat menggunakan tabel, tidak bisa dengan rumus. Hal ini karena Full Vigenère Cipher menggunakan tabel pergeseran yang acak, tidak teratur seperti tabel Vigenère standar. Kedua varian ini memiliki tingkat keamanan yang jauh lebih kuat dibandingkan Vigenère Cipher biasa.

B. Kriptanalisis Vigenère Cipher

Metode analisis frekuensi adalah metode yang paling umum digunakan untuk memecahkan kriptografi klasik. Metode ini pertama kali dikemukakan oleh seorang ahli *polymath* dari Arab yang bernama Al-Kindi. Ia menghitung kemunculan karakter dalam bahasa Arab yang terdapat pada Al-Qur'an. Teknik ini menjadi cikal bakal kriptanalis untuk memecahkan pesan yang tersembunyi dalam algoritma kriptografi klasik. Analisis frekuensi didasarkan pada fakta bahwa pada setiap bahasa, terdapat huruf-huruf dan kombinasi huruf-huruf tertentu yang muncul mengikuti pola distribusi frekuensi tertentu. Hal ini secara garis besar terbukti benar untuk hampir semua sampel bacaan pada bahasa tersebut. Contohnya dalam bahasa inggris, huruf-huruf yang paling sering muncul adalah E, T, A, O, I, N, sedangkan yang paling jarang adalah Z, Q, dan X. Demikian juga bigram dan trigram yang sering muncul selalu TH, ER, HE, THE, ON, dsb.

Sebagai salah satu jenis *polyalphabetic cipher*, Vigenère Cipher sebenarnya aman dari analisis frekuensi yang sederhana. Namun, kelemahan utama dari Vigenère Cipher adalah kuncinya yang selalu berulang. Apabila seorang kriptanalis mampu mengetahui panjang kunci yang digunakan, maka *ciphertext* yang ada dapat diperlakukan sebagai Caesar Cipher yang terpisah-pisah pengerjaannya sesuai dengan huruf-huruf pada kunci. Dalam penentuan panjang kunci ini, ada 2 metode yang dapat digunakan, yaitu metode Kasiski dan metode Friedman.

Sebelumnya, kriptanalisis terhadap Vigenère Cipher menggunakan pengetahuan tentang *plaintext* atau penggunaan kata yang umum untuk kunci. Namun, metode Kasiski tidak bergantung pada hal-hal tersebut. Metode Kasiski memanfaatkan kelemahan Vigenère Cipher dimana kunci berulang. Dasar pemikirannya adalah, suatu kata yang diulang-ulang penggunaannya,

misalnya *the*, memiliki kemungkinan bertemu dengan huruf-huruf yang sama pada kunci di beberapa kemunculannya, seperti pada contoh berikut:

```
Key:        ABCDABCDABCDABCDABCDABCDABCD
Plaintext:  CRYPTOISSHORTFORCRYPTOGRAPHY
Ciphertext: CSASTPKVSIQUTGQUCSASTPIUAQJB
```

Dari contoh diatas dapat terlihat bahwa jarak antara pengulangan CSASTP pada *ciphertext* adalah 16. Dari data tersebut, dapat ditebak bahwa panjang kunci merupakan faktor dari 16, yaitu 16, 8, 4, 2, atau 1. Karena 1 atau 2 dirasa terlalu singkat, kriptanalis dapat mencoba panjang kunci mulai dari 4, 8, dan 16. Agar lebih akurat, dibutuhkan *plaintext* yang lebih panjang. Semakin panjang suatu *plaintext*, maka kemungkinan terjadinya pengulangan seperti diatas akan menjadi lebih besar, kemudian dapat dihitung jaraknya seperti metode diatas, dicari faktor-faktornya, lalu kemudian disesuaikan angka mana saja dari pengulangan-pengulangan yang saling beririsan. Contohnya adalah *ciphertext* dengan 2 macam pengulangan sebagai berikut:

```
Ciphertext:
VHVSSPQUCEMRVBVBBBVHVSURQGIBDUGRNICJQUCERVU
AXSSR
```

Metode Friedman, disebut juga sebagai Tes Kappa, ditemukan pada tahun 1920an oleh William F. Friedman. Prinsipnya adalah dengan menggunakan tabel kemunculan. Dengan mengetahui probabilitas bahwa setiap dipilih dua sumber acak yang berbahasa sama (sekitar 0,067 untuk Bahasa Inggris) dan probabilitas kemunculan untuk pilihan acak seragam dari alfabet (0,0385 untuk Bahasa Inggris), panjang kunci dapat diperkirakan sebagai:

$$\frac{\kappa_p - \kappa_r}{\kappa_o - \kappa_r}$$

dimana

$$\kappa_o = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)}$$

dan c adalah ukuran dari alfabet (26 untuk Bahasa Inggris), N adalah panjang dari *plainteks*, dan n1 sampai nc huruf-huruf pada cipherteks, dalam *integer*.

Setelah mendapat panjang kunci dengan menggunakan metode Kasiski seperti di atas, *ciphertext* dapat ditulis ulang dan dikelompokkan menjadi kolom-kolom sebanyak panjang kunci yang masing-masing merupakan korespondensi terhadap setiap huruf pada kunci. Setiap kolom tersebut dapat dianggap sebagai *plaintext* yang dienkripsi dengan Caesar Cipher. Selanjutnya huruf pada kunci yang merupakan pergeseran pada Caesar Cipher dapat diketahui dengan menggunakan metode analisis frekuensi biasa untuk memecahkan Caesar Cipher.

Metode Kerckhoff merupakan pengembangan dari metode Kasiski, yaitu setiap kali suatu huruf dari kunci yang didapat dari pemecahan Caesar Cipher setiap kolom ditemukan, kriptanalis dapat langsung mendekripsi sebagian *ciphertext* agar dapat langsung dilakukan tebakan apabila ada kata yang sudah cukup jelas untuk ditebak pada *plaintext.*

Metode diatas tidak dapat digunakan apabila tabel Vigenère merupakan tabel acak (Full Vigenère Cipher), namun panjang kunci masih dapat ditentukan dengan menggunakan metode Kasiski biasa.

III. METODE

Seperti yang telah dijelaskan di atas, Vigenère Cipher dapat dengan mudah dipecahkan dengan metode Kasiski, yaitu dengan memanfaatkan pengulangan yang mungkin terjadi untuk menentukan panjang kunci, yang kemudian dilanjutkan dengan melakukan analisis frekuensi. Dari fakta-fakta tersebut, dapat dibuat sebuah hipotesis bahwa apabila pengulangan dapat dikacaukan, serta frekuensi kemunculan huruf-huruf dapat dimanipulasi, tingkat keamanan Vigenère Cipher akan menjadi lebih kuat.

Untuk mengacaukan pengulangan serta memanipulasi kemunculan huruf-huruf, saya memilih suatu cara yaitu dengan melakukan penyisipan huruf-huruf yang ada pada kunci ke dalam plaintext, baru kemudian dienkripsi seperti Vigenère Cipher biasa.

Pertimbangan saya melakukan hal tersebut ada dua, masing-masing untuk mengatasi masalah yang telah disebutkan diatas. Pertama, penyisipan kunci dapat mengacaukan struktur kata pada *plaintext*. Misalnya dalam bahasa inggris, trigram *the* merupakan trigram yang paling banyak muncul. Apabila terjadi penyisipan kunci pada plaintext, kemunculan kata *the* dan pengulangannya akan lebih jarang terjadi, karena kata *the* tersebut dapat berubah menjadi *txhe*, *thse*, *tbhxe*, dan sebagainya, tergantung pada penyisipan setiap berapa karakter yang akan ditentukan nanti. Kedua, penyisipan huruf kunci ke dalam *plaintext* tentunya dapat mengacaukan frekuensi kemunculan huruf-huruf. Misalnya dalam bahasa inggris, huruf yang paling sering muncul adalah E. Namun, kita dapat memanipulasinya, khususnya dalam *plaintext* panjang, dengan menggunakan huruf pancingan misalnya A, agar frekuensi yang paling sering muncul menjadi A bukan E. Misalnya kunci yang digunakan adalah HAVANA, ada 3 huruf A dalam kunci tersebut yang akan disisipkan ke dalam *plaintext*, dan huruf-huruf pada kunci tersebut akan terus menerus diulang untuk dimasukkan ke dalam *plaintext* sehingga bukan tidak mungkin huruf A menjadi huruf yang paling sering muncul pada *plaintext* hasil sisipan, bukan lagi huruf E seperti pada umumnya. Selain itu, dengan penyisipan juga akan mempersulit untuk melakukan metode Kerckhoff dimana kriptanalis harus menebak suatu kata yang dikenali. Hal ini disebabkan karena susunan kata-kata sudah dibuat kacau.

Ketika ingin menyisipkan huruf-huruf kunci ke dalam *plaintext*, timbul masalah baru. Tentunya kita tidak mungkin menggunakan suatu angka yang tetap pada

setiap kali enkripsi, karena hal itu berarti jika algoritmanya diketahui, maka metode ini tidak akan berguna. Karena itulah, diperlukan suatu pembangkit bilangan acak untuk menentukan setiap berapa huruf sekali pada *plaintext* untuk dilakukan penyisipan. Berikut adalah potongan program bagian pembangkit bilangan acak yang telah saya buat:

```
private int generatePengulangan(String key)
{
    int result = 0;
    for (int i = 0; i < key.length(); i++)
    {
        if (i%2 == 0) result += key.charAt(i);
        else result -= key.charAt(i);
    }
    return Math.abs(result)%10+1;
}
```

Gambar 2. Potongan program pembangkit bilangan acak

Cara kerja dari pembangkit bilangan acak ini adalah nilai ASCII dari setiap karakter nomor genap pada kunci dijumlahkan, nilai ASCII dari setiap karakter nomor ganjil dikurangkan, kemudian total dari penjumlahan dan pengurangan tersebut di mod 10. Angka 10 saya pilih karena menurut saya apabila penyisipan dilakukan pada setiap lebih dari 10 karakter sekali akan menyebabkan penyisipan kurang efektif, khususnya pada *plaintext* yang singkat.

Contoh dari hasil penghitungan pembangkit bilangan acak ini adalah sebagai berikut:

Kunci : HAVANA
Penghitungan: (H-A+V-A+N-A)%10 = 2

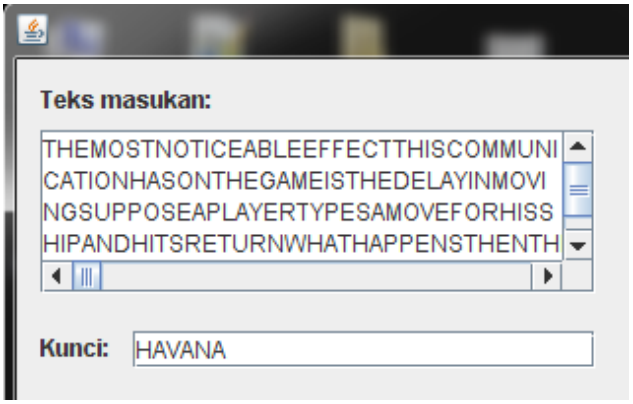
Penghitungan yang ditampilkan diatas dilakukan dengan penjumlahan dan pengurangan kode ASCII dari huruf-huruf kuncinya, karena program yang dibuat merupakan enkripsi dan dekripsi 256 karakter ASCII.

Bagian berikutnya dari program yang dibuat adalah bagian penyisipan huruf-huruf pada kunci ke dalam *plaintext*. Potongan programnya adalah sebagai berikut:

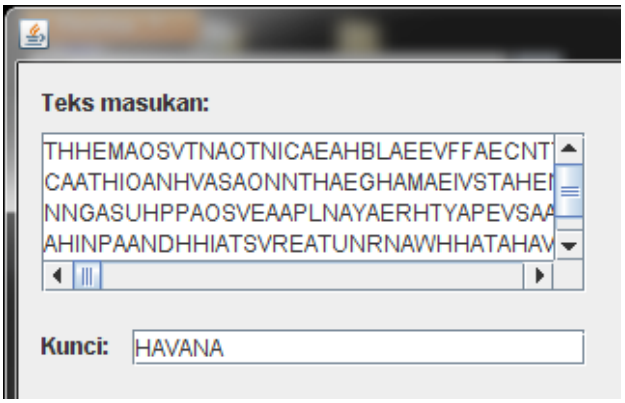
```
private String insertKey(String plainteks, String
key, int delta)
{
    String result = plainteks;
    int j = 0;
    for (int i = 0; (i + delta) < result.length(); i
    += delta)
    {
        result = result.substring(0, i+delta) + key.
        charAt((j++)%key.length()) + result.
        substring(i+delta,result.length());
        i++;
    }
    return result;
}
```

Gambar 3. Potongan program penyisipan

Hasil dari penyisipan oleh program adalah sebagai berikut:



Gambar 4. Program dengan teks sebelum penyisipan



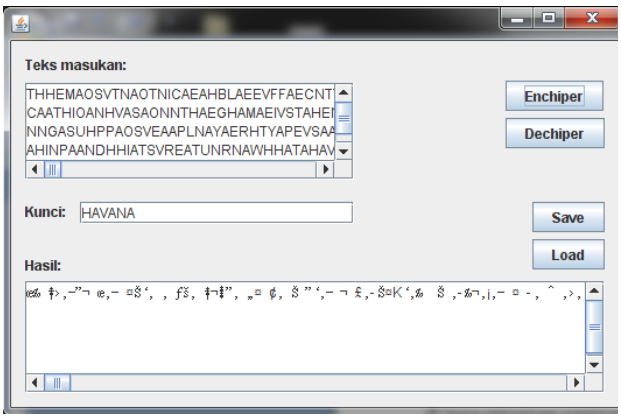
Gambar 5. Program dengan teks setelah penyisipan

Setelah *plaintext* disisipkan dengan kunci, dilakukan enkripsi seperti biasa. Potongan program untuk proses enkripsi:

```
private void jButton1ActionPerformed(java.awt.event.
ActionEvent evt) {
//GEN-FIRST:event_jButton1ActionPerformed
    String hasil=new String();
    String inputan=jTextArea1.getText();
    String konci=jTextArea2.getText();
    inputan = insertKey(inputan, konci,
    generatePengulangan(konci));
    int j=0;
    for (int i=0;i<inputan.length();i++) {
        if (j==konci.length()) {j=0;}
        int upil=(inputan.charAt(i) + konci.charAt(j
        ))%256;
        hasil += (char) upil;
        j++;
    }
    jTextArea3.setText(hasil);
    jTextArea1.setText(inputan);
}//GEN-LAST:event_jButton1ActionPerformed
```

Gambar 6. Potongan program enkripsi

Berikut adalah tampilan program setelah enkripsi:



Gambar 7. Tampilan program setelah enkripsi

Untuk proses dekripsi, pertama-tama dilakukan pembuangan karakter-karakter yang disisipkan dari dalam *ciphertext*. Setelah itu, barulah dilakukan dekripsi seperti biasa. Potongan program untuk pembuangan karakter hasil penyisipan:

```
private String removekey(String cipherteks, int
delta)
{
    String result = cipherteks;
    for (int i = delta; i < result.length(); i+=
    delta)
    {
        if ((i + 1) >= result.length()) result =
        result.substring(0,i+1);
        result = result.substring(0, i) + result.
        substring(i + 1,result.length());
    }
    return result;
}
```

Gambar 8. Potongan program pembuangan sisipan

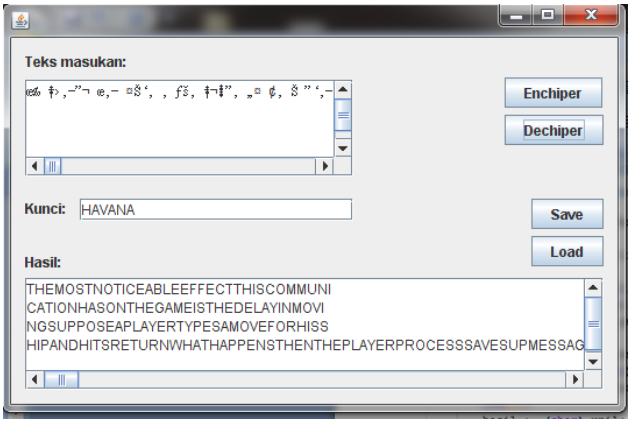
Potongan program untuk proses dekripsi adalah sebagai berikut:

```
private void jButton2ActionPerformed(java.awt.event.
ActionEvent evt) {
//GEN-FIRST:event_jButton2ActionPerformed

    String hasil=new String();
    String inputan=jTextArea1.getText();
    String konci=jTextArea2.getText();
    int j=0;
    for (int i=0;i<inputan.length();i++) {
        if (j==konci.length()) {j=0;}
        int upil=(inputan.charAt(i) - konci.charAt(j
        ))%256;
        hasil += (char) upil;
        j++;
    }
    hasil = removekey(hasil, generatePengulangan(
    konci));
    jTextArea3.setText(hasil);
    jTextArea1.setText(inputan);
}//GEN-LAST:event_jButton2ActionPerformed
```

Gambar 9. Potongan program dekripsi

Tampilan program untuk proses dekripsi adalah sebagai berikut:



Gambar 10. Tampilan program setelah dekripsi

IV. ANALISIS

Dikarenakan tidak adanya *tools* untuk menghitung bigram dan trigram dalam karakter ASCII, maka analisis dilakukan dengan menganalisis hasil setelah penyisipan. Hasil setelah dekripsi hanya merupakan asumsi.

```
TLHAEU TPMYERRAAMHI DL AOUFT MEEGRYAPHT SL
ATUHTEMREER AAHR    EL ANUOT MMEORRAEH  FLAAMUOTUMSE
RAANHC ILEANUTT MSEIRTAEHS  LWAIUTTHMIENR AEHG YLPATU,T
MOERR AFHO RL ATUHTAMTE RMAAHT TLEARU TEMLESREAWHH
ELRAEU TIMNE RTAHHE  LWAOURTLMDE,R ATHH ALNA UTTHMEE
RGARHE ALTA UPTYMREARMAIHD SL AAUTT MGEIRZAAH.
LTAHUETYM EARRAEH,  LWAIUTTHMOEURTA HQ ULEASUTTIMOENR,A
HT HLEA UITCMOENR AMHO SLTA UATSMSEORCAIHA TLEADU
TWMIETRHA HT HLEA UETGMYEPRTA.H  TLHAEUYT MHEARVAEH
BLEAEUNT MBEORTAHH  TLHAEU TMMAEIRNA HD
ELSATUITNMAETRIAOHN  LFAOURT MTEORUARHI SLTASU,T
MAENRDA HA  LSAOUUTRMCEER AOHF  LIAMUATGMIENRAATHI VLEA
UTTHMOEURGAHHT  LTAOU TTMHEER AWHO RLLADU TFMOERR AOHV
ELRA UTTHMREEREA HT HLOAUUSTAMNEDR AYHE ALRASU.T
MHEORWAEHV  ELRA,U TTMHEERRAEH  ALRAEU TAMCETRUAAHL
LLYA UOTVMEERR AOHN EL AHUUTNMDERREADH
PLYARUATMMIEDRSA HI NL AEUGTYMPETR,A HM ALNAYU TOMFE
RWAHHI CLHA UATRMEE RRAEHL ALTAIUVTEMLEYR AUHN
KLNAOUWTNM ETROA HA NLYAOUNTEM EWRHAOH  ILSA UNTOMTE
RAANH  ALNACUITEMNETR AEHG YLPATU TEMNETRHAUHS
ILAASUTT.M EARLALH  BLUATU TAM EVREARHY  LFAEUWT
MAERREA HG RLOAUUPTEMDE RAARHO ULNADU TAMNEDR ANHE ALRA
UTTHMEE RCAIHT YL AOUFT MCEARIARHO ,L AJUUTSMTE RSAOHU
TLHA UOTFM ETRHAEH  NLIALUET MDEERLATHA .L
AOUTTHMEERRWAIHS EL,A UOTNMLEYR AOHN EL ARUOTYMAELR
APHY RLAAMUITDM EIRSA HK NLOAWUNT MIENR ASHO
ULTAHUETRMNE REAGHY PLTA U(TAMTE RAABHY DLOASU)T,M
ETRHAAHT  LBAEUITNMGE RTAHHE  LOANUET MBEURIALHT  LBAYU
TAMHEMROASHE ,L AFUOTUMNEDREARH  OLFA UTTHMEE R1A8HT HL
ADUYTNMAESRIAYH  ALNADU TEMGEYRPATH' SL ANUETWM
EKRIANHG DLOAMU.TIMTE RMAAHY  LHAAUVTEM EARLASHO
LBAEUETNM ETRHAEH  LLAASUTT MREORYAAHL  LPAYURTAMMEIRDA
HB ULIALUTT MIENR AEHG YLPATU.T
```

Gambar 11. Tampilan hasil penyisipan soal 2 tucil 2

Ketika program ini dicoba dengan menggunakan contoh soal nomor 2 pada tugas kecil 2, metode Vigenère

Cipher dengan penyisipan ini terbukti lebih aman dibandingkan dengan Vigenère Cipher biasa. Dari hasil penyisipan saja, bila ditinjau dengan analisis frekuensi sederhana dengan menggunakan fasilitas *word count* dari notepad++, telah terjadi penyimpangan yang jauh dibandingkan sebelumnya. Selain itu, kata-kata yang ada juga menjadi sangat kacau sehingga menyusahkan untuk menebak kunci secara keseluruhan walaupun sebagian huruf dari kunci telah diketahui.

Apabila *plaintext* hasil penyisipan telah menjadi cukup aman, maka setelah dilakukan enkripsi yang sebenarnya, semestinya tingkat keamanan dari Vigenère Cipher modifikasi ini akan menjadi jauh lebih tinggi dibandingkan dengan Vigenère Cipher biasa.

V. KESIMPULAN

Metode enkripsi pesan dengan menggunakan Vigenère Cipher masih dapat diperbaiki. Salah satunya dengan melakukan modifikasi yaitu menyisipkan huruf dari kata kunci ke dalam *plaintext* sebelum dilakukan enkripsi Vigenère Cipher seperti biasa. Dengan melakukan penyisipan tersebut, metode Kasiski/Kerckhoff dan analisis frekuensi menjadi kurang efektif sehingga keamanan metode modifikasi ini menjadi lebih tinggi dibandingkan Vigenère Cipher biasa.

REFERENSI

http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher (19/3/2012)
http://math.ucsd.edu/~crypto/java/EARLYCIPHERS/Vigenère.html (19/3/2012)
http://en.wikipedia.org/wiki/Frequency_analysis (19/3/2012)

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, 19 Maret 2011

ttd

Kevin Leonardo Handoyo/13509019