Penggunaan Pembangkit Aliran Kunci Internal (*Internal Keystream Generator*) pada Algoritma *Vigenere Chiper* dengan Metode Penyusunan-Khusus Kartu Sulap (*Magic Card Vigenere Chiper*)

Muhammad Ghifary
Sekolah Teknik Elektro dan Informatika, Program Studi Informatika ITB
Jl. Ganesha 10, Bandung
email: m_ghifary@students.itb.ac.id, if15023@students.if.itb.ac.id

Abstrak :

Makalah ini membahas tentang pemodifikasian algoritma *vigenere chiper* dengan mengadopsi salah satu teknik atau alat pengacak yang digunakan oleh beberapa algoritma kriptografi modern, yaitu *keystream generator*. Algoritma modifikasi tersebut penulis namakan dengan *Magic Card Vigenere Chiper*. Tujuan dari penggunaan *keystream generator* ini adalah agar percobaan kriptanalisis dengan menggunakan metode analisis frekuensi ataupun dengan menggunakan metode kasiski menjadi sangat sulit untuk dilakukan. Pembuatan *keystream generator* pada algoritma modifikasi ini yaitu dengan menggunakan metode pengacakan dan penyusunan kartu sulap. Prinsip enkripsi dan dekripsi yang dilakukan mirip dengan prinsip algoritma *Stream Chiper*, namun yang akan dienkripsi ataupun didekripsi adalah simbol-simbol berupa karakter-karakter, bukan berupa bit-bit.

***Kata Kunci**: vigenere chiper, kartu sulap, internal keystream generator, polyalphabetic substitution chiper, magic card vigenere chiper, one-time pad*

1. PENDAHULUAN

1.1. *Vigenere Chiper*

Vigenere Chiper merupakan salah satu algoritma kriptografi klasik yang paling populer dan sering digunakan pada zamannya untuk menjaga kerahasiaan pesan. Algoritma ini dirancang pertama kali oleh Giovan Batisto Belaso pada tahun 1553, dan dipublikasikan oleh Blaise de Vigenere pada tahun 1586. Namun demikian, algoritma ini baru dikenal luas 200 tahun kemudian yang oleh penemunya dinamakan dengan *vigenere chiper*.

Vigenere Chiper termasuk dalam kategori *chiper* abjad-majemuk (*polyalphabetic subsitution chiper*) yang berarti satu karakter pesan pada *plaintext* berkorespondensi dengan lebih dari satu karakter pesan pada *chipertext* Misalnya, huruf 'A' pada *plaintext* dapat menjadi huruf 'P' atau 'H' pada *chipertext* yang berkaitan, bergantung pada kunci yang diberikan. Agar dapat terjadi demikian (*chiper* abjad-majemuk), metode enkripsi dilakukan dengan menggunakan suatu tabel khusus yang dinamakan bujursangkar *vigenere*, begitu pula dengan dekripsi.

	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. Bujursangkar Vigenere

Deretan karakter horizontal yang terletak pada bagian atas bujursangkar menyatakan karakter-karakter *plaintext* dan pada bagian sebelah kiri menyatakan karakter-karakter kunci. Sedangkan, karakter-karakter yang terletak pada setiap baris tabel menyatakan *chipertext* yang dihasilkan dari hasil pemetaan antara karakter *plaintext* dengan karakter kunci tertentu.

Algoritma *Vigenere Chiper* juga merupakan salah-satu jenis algoritma kriptografi kunci-simetris, dimana kunci yang sama digunakan untuk melakukan enkripsi dan dekripsi. Secara umum, algoritma *vigenere chiper* dapat dinyatakan dengan persamaan :

- $c_i = E(p_i) = (p_i + k_i) \bmod N$, untuk enkripsi, dan
- $p_i = D(c_i) = (c_i - k_i) \bmod N$, untuk dekripsi.

Ket : jika mengacu pada bujursangkar diatas, maka nilai N = 26

Sebagai contohnya, kita lakukan pengenkripsian terhadap suatu *plaintext* yang bertuliskan "VIGENERE" dengan kunci "pass". Karena panjang(kunci) < panjang(*plaintext*), maka kunci akan dibuat berulang hingga panjang(kunci) = panjang(*plaintext*). Kunci "pass" tersebut akan menjadi "passpass". Gunakan bujursangkar *vigenere* untuk melakukan pengenkripsian. Ambil satu per satu karakter *plaintext* dan karakter kunci yang berkoresponden. Lalu, cari karakter *chipertext* yang

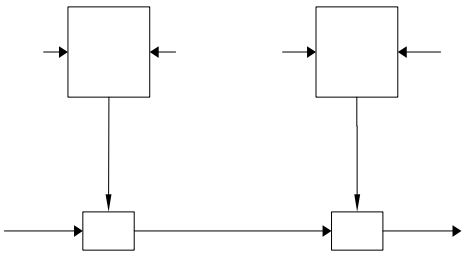
bersesuaian. Misalnya, karakter *plaintext* 'V' dengan karakter kunci 'p' akan memiliki karakter *chipertext* 'K'. Karakter *plaintext* 'I' dengan karakter kunci 'a' akan memiliki karakter *chipertext* 'I'. Dan seterusnya hingga karakter akhir. Hasil *chipertext* yang didapatkan adalah "KIYWCEJW".

Namun demikian, *vigenere chiper* telah dapat dipecahkan (selain dengan metode *exhaustive search*) dengan cara yang lebih mudah, yaitu dengan menggunakan teknik perpaduan antara metode kasiski dengan teknik analisis frekuensi, memanfaatkan perulangan yang terjadi pada *chipertext*. Metode kasiski berguna untuk mencari panjang kunci *vigenere chiper*.

1.2. *Magic Card Vigenere Chiper*

Agar teknik perpaduan tersebut tidak dapat atau sangat sulit untuk digunakan dalam percobaan kriptanalisis, maka dilakukan modifikasi terhadap algoritma *vigenere chiper* dengan menambahkan pembangkit aliran kunci internal (*internal keystream generator*). Algoritma modifikasi ini dinamakan dengan *Magic Card Vigenere Chiper*. Pembangkit aliran kunci yang terdapat pada algoritma ini biasa digunakan pada algoritma *stream chiper*. Namun, metode yang digunakan untuk membangkitkan kunci internal diadopsi dari metode penyusunan-khusus kartu sulap (akan dibahas pada bab selanjutnya). Dengan pengimplementasian pembangkit aliran kunci internal ini menjadikan *chipertext* yang dihasilkan semakin acak dan peluang timbulnya perulangan frase menjadi semakin kecil. Selain itu, mekanisme pengenkripsian dan pendekripsian juga dilakukan dengan cara *streaming* karakter per karakter.

Sebagai ilustrasi, algoritma *Magic Card Vigenere Chiper* dapat dimodelkan dengan diagram berikut.



Gambar 2. Diagram *Magic Card Vigenere Chiper*

U merupakan kunci yang dimasukkan oleh pengguna dan *Z* merupakan umpan(*seed*) yang dalam hal ini berupa susunan awal kartu pada sebuah tumpukan kartu sulap. *U* dan *Z* harus dirahasiakan dan hanya diketahui oleh pihak yang berwenang untuk melakukan enkripsi. *Keystream* yang dihasilkan merupakan hasil operasi yang dilakukan terhadap *U* dan *Z*. *Vig_E* merupakan fungsi enkripsi *vigenere* dengan cara *streaming* karakter per karakter, sedangkan *Vig_D* merupakan fungsi dekripsi *vigenere* dengan cara *streaming* karakter per karakter.

Secara umum, algoritma *Magic Card Vigenre Chiper* dapat dinyatakan dengan persamaan yang tidak jauh berbeda dengan algoritma *vigenere chiper* yang asli, yaitu sebagai berikut.

- $c_i = E(p_i) = (p_i + f(Ui, Zi)) \bmod N$, untuk enkripsi, dan
- $p_i = D(c_i) = (c_i - f(Ui, Zi)) \bmod N$, untuk dekripsi

Ket : $f(Ui, Zi)$ merupakan fungsi yang mengembalikan karakter-karakter *keystream*.

2. METODE PENYUSUNAN-KHUSUS KARTU SULAP

2.1. Sekilas tentang Kartu Sulap

Sebelum membahas tentang metode yang akan digunakan, kita tinjau terlebih dahulu apakah yang dimaksud dengan kartu sulap. Setumpuk kartu sulap berjumlah 54 buah kartu yang terbagi menjadi 4 buah *avatar*(wajik, keriting, hati, sekop) dan ditambah 2 buah *jocker*. Masing-masing *avatar* memiliki 13 buah kartu yang terdiri dari As, 2,3,…,10, Jack, Queen, dan King (jadi, jumlah seluruhnya adalah 13 x 4 + 2 = 54). Kartu tersebut disebut kartu sulap (*magic card*) karena sering dipakai dalam pertunjukan sulap berbasis trik kartu. Selain itu, kartu tersebut juga kadang disebut sebagai kartu permainan (*playing card*) karena digunakan dalam permainan-permainan kartu yang sudah lazim dimainkan dimasyarakat ataupun pada permainan judi.

Kombinasi penyusunan yang dilakukan terhadap setumpuk kartu sulap menjadi ide dasar untuk menghasilkan bilangan yang *pseudo-random*, yang berarti terdapat langkah-langkah tertentu yang telah terdefinisi untuk menghasilkan kembali bilangan acak yang sama. Langkah-langkah terdefinisi tersebut menjadi dasar perancangan algoritma yang digunakan pada *internal keystream generator* agar untuk satu siklus proses enkripsi-dekripsi dapat dihasilkan *keystream* yang sama untuk pengirim(*sender*) dan penerima(*receiver*). Selain itu, banyaknya susunan kartu yang mungkin adalah berjumlah 52! (jika 2 buah *jocker* tidak disertakan), sebuah bilangan yang cukup baik untuk menghasilkan susunan yang *pseudo-random*.

2.2. Aturan-Aturan yang Digunakan

Untuk setiap kartu pada sebuah tumpukan kartu, diberikan nilai berupa angka [1..53] dengan aturan-aturan sebagai berikut.

1. 2 buah *jocker* dibedakan menjadi 1 *jocker* besar (JB) dan 1 *jocker* kecil (JK)
2. Didefinisikan sebuah fungsi ***harga(x)*** yang mengembalikan nilai nominal dari sebuah kartu. (*x* merupakan variabel yang menyatakan sebuah kartu). Nilai dari fungsi ***harga(x)*** tidak bergantung pada *avatar* yang dimiliki oleh *x*.
   Contoh :

- *harga(1S) = harga(1W) = harga(1K) = harga(1H) = 1*, yang berarti harga kartu As Sekop, As Wajik, As Keriting, dan As Hati adalah masing-masing sebesar 1.
- Untuk x = *jack* atau *x = queen* atau *x = king*, maka masing-masing dari harga(x)-nya adalah 11, 12, dan 13.

3. Didefinisikan sebuah fungsi ***nilai(x)*** yang mengembalikan nilai berupa integer [1..53]. Nilai dari fungsi ***nilai(x)*** bergantung pada harga(x) dan *avatar* yang dimiliki oleh kartu x.
   - jika sebuah kartu x adalah kartu keriting, maka *nilai(x) = harga(x)*,
   - jika sebuah kartu x adalah kartu wajik, maka *nilai(x) = harga(x) + 13*,
   - jika sebuah kartu x adalah kartu hati, maka *nilai(x) = harga(x) + 26*,
   - jika sebuah kartu x adalah kartu sekop, maka *nilai(x) = harga(x) + 39*,
   - jika sebuah kartu x adalah kartu joker (JB atau JK), maka *nilai(x) = 53*.

   Contoh : As keriting memiliki nilai 1(*nilai(11K) = 1*), sedangkan As wajik memiliki nilai 14 (*nilai(11W)=14*).

2.3. Langkah-Langkah Pembentukan *Keystream*

Berikut ini langkah-langkah yang digunakan pada *internal keystream generator* untuk menghasilkan *keystream* yang *pseudo-random*. Langkah-langkah ini akan digunakan sebagai dasar perancangan fungsi *f(Ui,Zi)*.

1. Tentukan susunan awal kartu. Susunan ini akan menjadi nilai *Z* untuk digunakan oleh *keystream generator* pada proses enkripsi maupun dekripsi (lihat gambar 2). Susunan awal dapat diperoleh dengan cara memilih secara manual atau mengocok tumpukan kartu.
2. Cari *jocker* JB dan jocker JK, lalu pindahkan JB ke bawah sebanyak m buah kartu dan JK ke bawah sebanyak *n* buah kartu. Kartu JB dan JK berfungsi sebagai pembatas antara 3 buah himpunan kartu(mungkin himpunan kosong). Jika JB terletak di bagian paling akhir tumpukan kartu, maka JB akan ditempatkan *m* posisi setelah kartu yang paling atas. Jika JK terletak di bagian paling bawah tumpukan kartu, maka JK akan ditempatkan *n* posisi setelah kartu yang paling atas.
   Contoh :
   Didefinisikan sebuah tumpukan kartu dengan *n* buah kartu. Untuk penyederhanaan, kita gunakan *n* = 10
   Misalkan susunan awal yang diperoleh dari langkah 1 adalah

4H 2K 7H **JB** 6W 3K **JK** 8H 4K 11H

   Jika diketahui nilai m = 1 dan nilai n = 2, maka susunan kartu setelah dilakukannya langkah ke-2 akan menjadi

4H 2K 7H 6W **JB** 3K 8H 4K **JK** 11H

3. Untuk posisi JB < posisi JK, tukar posisi antara himpunan kartu sebelah kiri JB dengan himpunan kartu sebelah kanan JK.
   Untuk posisi JB > posisi JK, tukar posisi antara himpunan kartu sebelah kanan JB dengan himpunan kartu sebelah kiri JK.
   Contoh :
   Setelah dilakukan langkah ke-3 terhadap susunan kartu hasil dari langkah ke-2, maka akan diperoleh susunan sebagai berikut.

11H **JB** 3K 8H 4K **JK** 4H 2K 7H 6W

4. Lakukan kembali langkah ke-2 dengan m dan n yang sama.
   Contoh :
   Setelah dilakukan langkah ke-4, diperoleh susunan kartu sebagai berikut.

11H 3K **JB** 8H 4K 4H 2K **JK** 7H 6W

5. Untuk posisi JB < posisi JK, tukar posisi antara himpunan kartu di antara JB dan JK dengan himpunan kartu di sebelah kanan JK.
   Untuk posisi JK < posisi JB, tukar posisi antara himpunan kartu di antara JB dan JK dengan himpunan kartu di sebelah kanan JB.
   Contoh :
   Hasil dari pengerjaan langkah ke-5 adalah sebagai berikut.

11H 3K **JB** 7H 6W **JK** 8H 4K 4H 2K

6. Tinjau kartu di posisi akhir. Gunakan nilai dari fungsi *nilai(x_akhir)* untuk memindahkan sebuah himpunan kartu A yang berjumlah sebesar *nilai(x_akhir)* ke bagian depan. Himpunan kartu A adalah himpunan sejumlah *nilai(x_akhir)* kartu terakhir sebelum kartu yang paling akhir. Langkah ini tidak menyebabkan perubahan terhadap posisi kartu pada bagian akhir.
   Contoh :
   Dari hasil pengerjaan langkah ke-5, *nilai(x_akhir)* dimana *x_akhir* = 2K adalah sebesar 2. Maka, himpunan kartu A adalah {4K, 4H}. Hasil dari pengerjaan langkah ke-6 ini adalah sebagai berikut.

4K 4H 11H 3K **JB** 7H 6W **JK** 8H 2K

   Untuk kasus ini, jika *nilai(x_akhir) > n – 1*, dimana *n* adalah total jumlah kartu pada tumpukan, maka hasilnya akan sama dengan hasil yang didapatkan pada langkah ke-5.
7. Tinjau kartu di posisi pertama. Gunakan nilai dari fungsi *nilai(x_pertama)* untuk mencari sebuah kartu *x_stream* yang *nilai(x_stream)*-nya akan digunakan menjadi karakter *cardstream*. *Cardstream* merupakan karakter yang akan digunakan untuk menghasilkan *keystream* dengan melakukan operasi transformasi terhadap *cardstream* dan kunci yang diberikan. Kartu *x_stream* merupakan kartu yang terletak urutan ke-*nilai(x_pertama)* dari depan. Jika *x_stream = JB* atau *x_stream = JK*, maka langkah ini tidak menghasilkan *cardstream* dan *loop* tidak ter-*increment*.

Contoh :

Dari hasil pengerjaan langkah ke-6, *nilai(x_pertama)* di mana *x_pertama* = 4K adalah sebesar 4. Maka, kartu *x_stream* adalah kartu urutan ke-4 dari depan yaitu kartu 3K. Nilai dari fungsi *nilai(3K)* akan menjadi karakter pertama *cardstream*. Untuk kasus ini, jika *nilai(x_pertama)* melebihi total jumlah kartu pada tumpukan, maka perhitungan dilanjutkan dan diulang dari kartu pertama. Misalkan, *x_pertama* = 12H, maka *nilai(x_pertama)* = 12, dimana nilai tersebut lebih besar dari *n* buah kartu yang didefinisikan, yaitu 10. Jika demikian, kartu yang dipilih kartu ke- (12-10) = 2, yaitu kartu 4H.

8. Langkah ini untuk menghasilkan sebuah karakter *keystream*. Lakukan operasi transformasi antara *cardstream* yang dihasilkan pada langkah ke-7 dengan kunci *U* yang diberikan untuk menghasilkan *keystream*. Operasi penghasil *keystream* tersebut dapat dinyatakan dengan suatu fungsi ***T(Ui, CSi)***, dimana *Ui* menyatakan *stream* nilai integer dari *stream* karakter alfabet kunci dan *CSi* menyatakan stream nilai integer dari *cardstream*. Hasil fungsi tersebut didapatkan dari nilai *Ui* ditambahkan dengan angka 1 secara berulang-ulang. Setiap penambahan nilai *Ui* dengan 1 diikuti dengan pengurangan nilai *CSi* dengan 1. Proses penambahan nilai *Ui* berakhir pada saat *CSi* = 0. Korespondensi antara suatu alfabet dengan nilai *integer*-nya adalah sebagai berikut.

   ‘a’=0, b=’1’,c=’2’,....,’y’=24,’z’=25

   Contoh :

   Misalkan karakter pertama kunci *U* adalah ‘p’. Ubah alfabet ‘p’ menjadi *integer*, yaitu 15 ($U_1 = 15$). Ambil nilai *integer* dari *cardstream* karakter pertama yang diperoleh dari langkah ke-7, yaitu CS_1=*nilai(3k)*=3. *Keystream* karakter pertama yang dihasilkan adalah *T(15,3) = 18*, yaitu karakter ‘S’. Langkah-langkah pembentukan *keystream* yang dilakukan oleh fungsi T(15,3) adalah

   - $U_1 = 15 + 1 = 16;\ CS_1 = 3 – 1 = 2$
   - $U_1 = 16 + 1 = 17;\ CS_1 = 2 – 1 = 1$
   - $U_1 = 17 + 1 = \mathbf{\underline{18}};\ CS_1 = 1 – 1 = 0$

   Langkah tersebut berakhir pada $CS_1 = 0$ dan *keystream* yang dipilih merupakan nilai akhir dari $U_1\{18\}$.

   Namun, misalkan karakter pertama kunci *U* adalah ‘y’ dimana nilai integer ‘y’ adalah 24 ($U_1 = 24$), maka T(24,3) = 1, dengan langkah-langkah sebagai berikut

   - $U_1 = 24 + 1 = 25;\ CS_1 = 3 – 1 = 2$
   - $U_1 = 25 + 1 = 0$ (kembali ke alfabet A)*;* $CS_1 = 2 – 1 = 1$
   - $U_1 = 0 + 1 = \mathbf{\underline{1}};\ CS_1 = 1 – 1 = 0$

Tanpa mengubah susunan kartu, ulangi langkah-2 hingga langkah ke-8 ini sebanyak panjang *plaintext/chipertext* yang diberikan. Susunan kartu yang dihasilkan pada langkah ke-6 akan menjadi susunan awal untuk *loop* berikutnya. *Loop* berakhir hingga semua karakter *plaintext/chipertext* telah dienkripsi/didekripsi.

Misalkan, diketahui sebuah *plaintext/chipertext* yang memiliki panjang karakter sebesar 5 (spasi diabaikan), maka *cardstream* dan *keystream* yang dihasilkan juga akan sepanjang 5 karakter. Jika simulasi langkah-langkah pada contoh tersebut (lihat contoh pada langkah ke-6)diteruskan hingga 5 kali, maka *keystream* yang dihasilkan dari T(*Ui*, *CSi*) dengan kunci “pk” adalah sebagai berikut.

Cardstream	: 3K	2K	(JB)	7H	4K	7H
CSi {nilai(cardstream)}	: 3	2	(53)	33	4	33
Kunci	: p	k	-	p	k	p
Ui	: 15	10	-	15	10	15
T(*Ui*,*CSi*)	: 18	12	-	22	14	22
Keystream	: S	M	-	W	O	W

Ket:

- Jocker tidak dijadikan sebagai *keystream*.

Keystream “SMWOW” kemudian akan digunakan untuk proses enkripsi/dekripsi terhadap *plaintext* dengan menggunakan algoritma *Vigenere Chiper* biasa.

Algoritma langkah-langkah penghasil *keystream* tersebut bersifat *reversible*. Untuk masalah keamanan, algoritma langkah-langkah penghasil keystream ini sebaiknya dirahasiakan.

3. STUDI KASUS TERHADAP ALGORITMA *MAGIC CARD VIGENERE CHIPER*

Untuk mengukur tingkat kekuatan algoritma *Magic Card Vigenere Chiper* yang telah dibuat, akan dilakukan sebuah simulasi pengenkripsian suatu *plaintext*, baik dengan teknik algoritma *vigenere* yang asli (sebagai tolok ukur) maupun dengan algortima *Magic Card Vigenere Chiper*.

Diberikan sebuah *plainteks* sebagai berikut:

“THE BEAR WENT OVER THE MOUNTAIN YEAH THE DOG WENT ROUND THE HYDRANT”

Plaintext tersebut akan dienkripsi dengan algoritma vigenere yang asli, menggunakan kata kunci “KUNCI”. *Chipertext* yang dihasilkan adalah sebagai berikut.

“**DBR**DM KLJGV DIIGZ **DBR**OW EHGCQ XSRCP DBRFW QQRPB BIHPL **DBR**JG NLNPB”

Dari *chipertext* di atas, dapat dilihat bahwa terdapat sebuah *substring* tripel huruf yang berulang sejumlah

kali(*substring* yang digarisbawahi). Jika terjadi kondisi seperti ini, *chipertext* akan mudah dipecahkan dengan menggunakan metode kasiski dan analisis frekuensi (dengan tambahan informasi berupa data statistik berbagai tripel huruf yang paling sering muncul). Kondisi tersebut terjadi karena karakteristik bawaan dari algoritma *vigenere chiper* dan juga akibat kunci yang dipilih tidak terlalu panjang.

Sekarang kita akan lakukan pengenkripsian *plaintext* dengan menggunakan algoritma *Magic Card Vigenere Chiper* dengan kunci yang sama, yaitu "KUNCI", serta umpan *Z* (susunan awal kartu berjumlah 54 buah) sebagai berikut :

11H 1K … 12S 3K **JB** 10K 11K …5H 7H **JK** 7K 8K … 2S 3S

maka *chipertext* yang akan dihasilkan adalah sebagai berikut :

"ZLMIZTHDMWLTDIVAVZUYGQFBI IVO…dst"

* Keterangan : tidak dilakukan pengenkripsian hingga akhir, hanya menunjukkan bahwa *chipertext* pada string-string awal yang dihasilkan tidak lagi terdapat tripel-tripel huruf

Dari *chipertext* yang dihasilkan dengan algoritma *Magic Card Vigenere Chiper* dapat dilihat bahwa *chipertext* semakin tidak memiliki pola atau terlihat lebih acak dibandingkan dengan *chipertext* dengan algoritma *vigenere chiper* biasa sehingga *chipertext* akan sangat sulit untuk dipecahkan dengan metode kasiski ataupun analisis frekuensi. Pada *chipertext* hasil algoritma modifikasi tidak terdapat tripel-tripel huruf yang berulang. Hasil enkripsi dari algoritma modifikasi ini memberikan hasil yang mirip dengan algoritma *One-Time Pad*, yaitu algoritma yang termasuk kategori *chiper* yang tidak dapat dipecahkan (*unbreakable chiper*)[1]. Namun, kelebihan yang dimiliki oleh algoritma *Magic Card Vigenere Chiper* ini dibandingkan dengan *One-Time Pad* adalah panjang kunci yang dimasukkan dapat berjumlah jauh lebih sedikit daripada panjang *plaintext*, tidak seperti algoritma *One-Time Pad* yang mengharuskan panjang kunci sama dengan panjang *plaintext* dan memiliki buku kode(*pad*) tersendiri. Oleh karena itu, algoritma ini cukup baik dan mangkus untuk diimplementasikan.

Untuk melakukan dekripsi, susunan awal kartu dan kunci yang digunakan harus sama dengan proses enkripsi untuk dapat meng-*generate keystream* yang sama. Kunci dan susunan awal kartu sulap harus benar-benar dirahasiakan serta harus diinformasikan ke pihak dekriptor yang berwenang dengan cara yang aman.

Adapun kekurangan-kekurangan yang dimiliki oleh algoritma *Magic Card Vigenere Chiper* ini yaitu

- Susunan awal kartu yang digunakan dapat diketahui hanya dari *cardstream* yang di-*generate* dengan memanfaatkan sifat *reversible* dari algoritma pembentuk *internal keystream generator*. Bahkan, jika kunci telah diketahui sebelumnya, susunan awal kartu dapat diketahui dari *keystream* yang telah di-*generate*.
- Adanya kemungkinan dapat dilakukannya *chosen-text attack* dengan memilih beberapa pasangan *plaintext-chipertext* yang diketahui telah dienkripsi/didekripsi dengan susunan awal kartu dan kunci yang sama. Jika diketahui 2 buah *plaintext* A dan B serta 2 buah *chipertext* $A+f(U,Z)$ dan $B+f(U,Z)$, maka akan dihasilkan $[A+f(U,Z)] - [B+f(U,Z)] = A - B$.

Untuk mengoptimalkan penggunaan algoritma *Magic Card Vigenere Chiper*, maka hal-hal yang seharusnya dilakukan adalah

1. Tidak menggunakan kunci dan susunan awal kartu yang sama untuk mengenkripsi *plaintext* yang berbeda.
2. Menggunakan kunci yang cukup panjang untuk menghindari *brute force attack*. [2]

4. KESIMPULAN DAN SARAN

Dari hasil percobaan terhadap *Magic Card Vigenere Chiper* dapat diambil kesimpulan bahwa algoritma ini lebih kuat(*robust*) dibandingkan dengan algoritma *Vigenere Chiper* yang umum. Algoritma modifikasi ini mampu menghasilkan *chipertext* yang tidak berpola khusus sehingga menjadikan percobaan kriptanalisis dengan menggunakan metode kasiski dan teknik analisis frekuensi menjadi sangat sulit untuk dilakukan.

Saran pengembangan yang diusulkan adalah menyimpan informasi susunan awal kartu ke dalam *chipertext* dengan tujuan agar dekriptor cukup diberitahukan kuncinya saja. Penyimpanan informasi susunan awal kartu ke dalam *chipertext* sebaiknya dilakukan secara tersebar di seluruh *chipertext* dengan pola penyebaran yang *pseudo-random*. Untuk itu, perlu dibuat dan ditambahkan algoritma khusus untuk menyimpan informasi susunan awal kartu pada waktu enkripsi serta mengambil informasi susunan tersebut pada waktu dekripsi.

DAFTAR REFERENSI

[1] Munir, Rinaldi. "Diktat Mata Kuliah IF5054 Kriptografi STEI Program Studi Teknik Informatika ITB", 2006

[2] M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, S. Shimomura, E. Thompson, M. Weiner, "Minimal Key Length for Symmetric Chipers to Provide Adequate Commercial Security", http://www.schneier.com

[3] B. Braatz, "Using RC4 as A Card Decker Chiper", 2006, http://www.heptasean.de