Rancangan Algoritma Kriptografi Boink2 Berbasis Subtitusi Karakter

Tessa Ramsky - NIM : 13504124
Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
Email: if14124@students.if.itb.ac.id

Abstract – *Makalah ini membahas tentang rancangan dan implementasi algoritma kriptografi Boink2. Algoritma ini merupakan hasil rekayasa dari algoritma klasik dengan memanfaatkan nilai indeks dari karakter. Algoritma Boink2 yang akan dirancang ini akan selalu menghasilkan panjang chipertext yang lebih pendek dibandingkan plaintextnya.*

Makalah ini membahas konsep dasar algoritma Boink2, proses enkripsi dan dekripsinya, dan hasil implementasi sederhana algoritma tersebut menggunakan bahasa pemrograman C# dan memanfaatkan kakas Visual Studio 2005. Selain itu, makalah juga membahas tingkat keamanan algoritma ini ditinjau dari aspek kelebihan dan kekurangan algoritma, serangan kriptanalisis yang mungkin, serta arah pengembangan algoritma ini ke depannya.

Kata Kunci*: algoritma klasik, chipper, indeks karakter, level, enkripsi, dekripsi*

1. PENDAHULUAN

1.1. Kriptografi

Algoritma kriptografi (*cipher*) adalah aturan untuk *enciphering* dan *deciphering* atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Berdasarkan sejarah algoritma kriptografi dapat dibedakan menjadi algoritma kriptografi klasik dan algoritma kriptografi modern. Algoritma yang akan dirancang berikut ini termasuk dalam algoritma klasik.

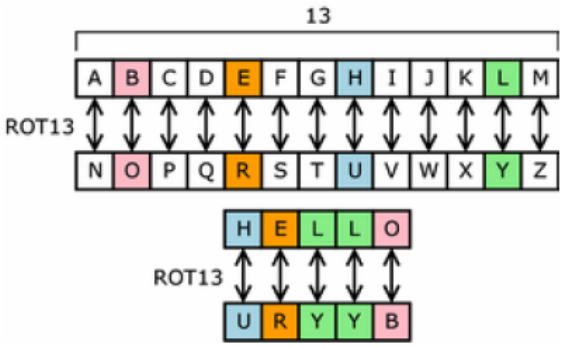
1.2. Algoritma Klasik

Algoritma kriptografi klasik yaitu algoritma kriptografi sebelum masuk era digital, Kriptografi yang dilakukan berbasis karakter. Algoritma kriptografi klasik termasuk dalam sistem kriptografi simetri, karena kunci untuk melakukan enkripsi sama dengan kunci untuk melakukan enkripsi.

1.3. *Chiper* Subtitusi

Algoritma kriptografi substitusi merupakan cipher tertua. Prinsip utama cipher substitusi adalah menukarkan setiap huruf pada *plaintext* dengan huruf lain/simbol.

Salah satu jenis algoritma chipper subtitusi adalah Cipher Substitusi Abjad Tunggal. Algoritma ini mensubstitusi satu karakter pada *plaintext* dengan satu karakter yang bersesuaian. Jadi, fungsi *cipher*ingnya adalah fungsi satu ke satu.



Gambar 1 Tabel Subtitusi Abjad Tunggal

2. RANCANGAN ALGORITMA BOINK2

2.1. Latar Belakang

Saat ini, algoritma kriptografi modern merupakan algoritma yang paling banyak dikembangkan dan digunakan karena dianggap memiliki tingkat keamanan yang tinggi. Namun, algoritma kriptografi klasik tetap masih penting peranannya karena algoritma kriptografi klasik merupakan landasan dasar algoritma kriptografi modern, seperti pada operasi substitusi dan transposisi. Hanya saja algoritma modern berada pada operasi bit per bit, bukan operasi karakter.

Kebanyakan dari algoritma yang ada sekarang memiliki karakteristik yaitu panjang (jumlah karakter) *chipertext* sama dengan panjang *plaintext*, atau kelipatannya seperti pada chipper subtitusi homofonik. Algoritma akan dirancang ini memiliki ciri yang berbeda. Panjang *chipertext* akan selalu lebih pendek dari panjang *plaintext*. Bahkan sebuah *chipertext* dapat hanya terdiri dari satu buah karakter. Hal ini tentunya dapat menimbulkan *diffusion dan confussion.*

Algoritma yang dirancang ini menggunakan prinsip *chiper* subtitusi, hanya saja karakter hasil subtitusinya berasal dari dua karakter yang berdampingan pada *plaintext*. Selain itu, proses enkripsinya dapat

dilakukan berulang-ulang sebanyak n kali, dengan n <panjang *plaintext*.

Cara kerja algoritma ini mirip dengan permainan TRUE LOVE yang sering dimainkan anak-anak Sekolah Dasar, yaitu dengan menjumlahkan dua nilai integer dari karakter yang berdampingan hingga tersisa dua karakter. Nama algoritma 'Boink2' berasal dari cara menjumlahkan dua angka yang berdekatan dengan membuat garis melengkung.

Gambar 2 Permainan TRUE LOVE

Hal penting yang harus diperhatikan pada algoritma ini adalah pendefinisian indeks karakter, kunci, dan level.

2.2. Indeks Karakter

Indeks karakter merupakan nilai integer *n* dari sebuah karakter. Nilai integer ini menunjukkan indeks/urutan karakter tersebut di dalam daftar karakter yang didefinisikan. Indeks karakter ini harus sudah didefinisikan di awal algoritma.

Karakter	A	B	C	D	E	F	G	H	..	..
Indeks	0	1	2	3	4	5	6	7	..	..

Tabel 1Table Indeks Karakter

Kolom setelah karakter 'H' dapat diisi dengan karakter lainnya, seperti angka atau symbol lainnya sesuai kebutuhan algoritma.

Indeks karakter ini digunakan dalam perhitungan subtitusi karakter pada proses enkripsi dan dekripsi. Dalam perancangan ini, indeks karakter yang didefinisikan hanya abjad a hingga z.

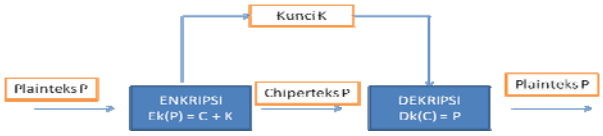
2.3. Level dan Kunci

Level menunjukkan tingkatan enkripsi pada *plaintext*, karena *plaintext* pada algoritma ini dapat dienkripsi berulang-ulang. Level 3 menunjukkan bahwa *plaintext* dienkripsi sebanyak tiga kali. Begitu juga dengan proses dekripsinya.

Yang unik dari algoritma ini adalah kunci bukan merupakan input pada proses enkripsi, tetapi justru menjadi output yang akan digunakan untuk mendekripsi *chipertext*. Input untuk proses enkripsi adalah level, yang juga menunjukkan panjang kunci. Panjang kunci merupakan selisih dari panjang plaintext dan chippertext karena setiap proses enkripsi dilakukan, panjang *chipertext* berkurang satu. Jadi, semakin tinggi levelnya maka semakin pendek *chipertext* nya.. Jadi, persamaannya dapat dibuat sebagai berikut:

$$\mathbf{n_{plaintext} = n_{chippertext} + n_{key}}$$

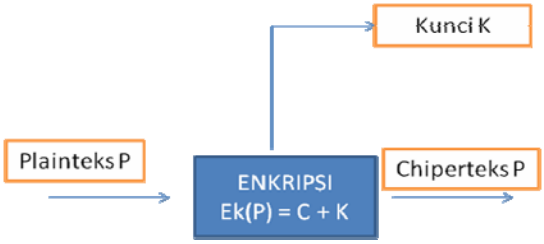
Berikut adalah gambaran proses enkripsi dan dekripsinya:



Gambar 3 Proses Dekripsi dan Dekripsi

2.4. Proses Enkripsi

Proses enkripsi pada algoritma ini sama sekali tidak membutuhkan kunci, tetapi justru menghasilkan kunci yang nantinya diperlukan saat mendekripsi *chipertext* tersebut. Kunci dihasilkan bersamaan saat menghasilkan *chipertext*. Panjang kunci yang dihasilkan pun sama dengan level.



Gambar 4 Proses Enkripsi

Berikut adalah langkah-langkah dalam melakukan enkripsi:

a. Catat karakter pertama pada *plaintext*, kemudian tambahkan (*concat*) pada kunci.
b. Ubah nilai setiap karakter pada *plaintext* menjadi nilai indeks karakternya.
c. Jumlahkan nilai indeks karakter dari setiap dua karakter yang berdampingan pada *plaintext*. Apabila jumlahnya lebih dari nilai maksimum indeks karakter (25), maka hasilnya dimodulo 25.
d. Ubah setiap hasil penjumlahan di atas menjadi karakter sesuai dengan nilai indeks karakternya. Maka diperoleh *chipertext* level 1.
e. Untuk melakukan enkripsi hingga level n, lakukan enkripsi kembali terhadap *chipertext* pada hasil *d* sebagai *plaintext.* Ulangi proses ini sebanyak *n* kali.

Misal *plaintext* yang akan dienkripsi adalah

"BAKAYARO" dengan indeks karakter 25. Maka untuk kasus level 1:

B	A	K	A	Y	A	R	O
1	0	10	0	24	0	17	14

'B' sebagai kunci. Jumlahkan setiap dua indeks karakter yang berdampingan di atas:

B	A	K	A	Y	A	R	O
1	0	10	0	24	0	17	14
	1	10	10	24	24	17	31

Untuk hasil penjumlahan yang lebih dari 25, maka dimodule dengan 25:

$$(31) \bmod 25 = 6$$

Setelah diubah, maka hasilnya:

B	A	K	A	Y	A	R	O
1	0	10	0	24	0	17	14
	1	10	10	24	24	17	6
	B	K	K	Y	Y	R	G

Maka,*chipertext* hasil enkripsi level 1 adalah "BKKYYRG" dengan kunci 'B'.

Untuk kasus level 3, maka proses enkripsinya menjadi:

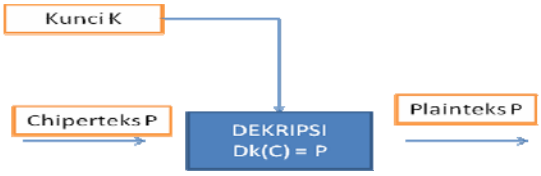
B	A	K	A	Y	A	R	O *(plaintext)*
1	0	10	0	24	0	17	14
	B	K	K	Y	Y	R	G *(Level 1)*
	1	10	10	24	24	17	6
		11	20	34	48	41	23
		11	20	9	23	16	23
		L	U	J	X	Q	X *(Level 2)*
		11	20	9	23	16	23
			31	29	32	39	39
			6	4	7	14	14
			G	E	H	O	O *(Level 3)*

Maka, *chipertext* hasil enkripsi pada level 3 adalah "GHEDD" dengan kunci "BBL".

Dari contoh sederhana di atas kita dapat melihat bahwa *plaintext* yang dienkripsi sebanyak tiga kali akan menghasilkan *chipertext* yang sangat jauh berbeda dari *plaintext*nya, baik dari susunan karakternya maupun panjang teksnya.

2.5. Proses Dekripsi

Proses dekripsi pada algoritma ini sama dengan algoritma kriptografi lainnya, yaitu dengan menerima *chipertext* dan kunci yang diperoleh dari hasil enkripsinya. Perbedaannya terletak pada panjang *plaintext* yang dihasilkan yang lebih panjang dari *chipertext*. Selain itu, *chipertext* didekripsi sebanyak panjang kunci. Pertambahan panjang pada *plaintext* ini sesuai dengan panjang kunci/levelnya.



Gambar 5 Proses Dekripsi

Berikut adalah langkah-langkah dalam melakukan dekripsi:

a. Ubah nilai setiap karakter pada *chipertext* menjadi nilai indeks karakternya.
b. Ambil karakter terakhir dari kunci, kemudian tambahkan pada *plaintext*. Ubah nilainya menjadi nilai indeks karakternya.
c. Lakukan pengurangan nilai indeks karakter pada *chipertext* dengan indeks karakter pada *plaintext*. Apabila nilainya negatif, tambahkan 25. Lakukan operasi ini untuk karakter berikutnya pada *chipertext* dan karakter terakhir pada *plaintext*.
d. Ubah setiap hasil pengurangan di atas menjadi karakter sesuai dengan nilai indeks karakternya. Maka diperoleh *plaintext* level 1.
e. Untuk melakukan dekripsi hingga level n, lakukan dekripsi kembali dengan *plaintext* pada hasil *d* sebagai *chipertext* untuk proses dekripsi *a,b,c,d* di atas sebanyak *n* kali.

Misal *chipertext* yang akan dienkripsi adalah "GHEDD" dengan kunci "BBL". Maka untuk kasus level 1:

G	E	H	O	O
6	4	7	14	14

Ambil karakter terakhir dari kunci 'BBL', yaitu L, kemudian isikan pada sebagai karakter pertama *plaintext*.

	G	E	H	O	O
	6	4	7	14	14
L					
11					

Hitung indeks karakter *plaintext* selanjutnya dengan mengurangi indeks karakter *chipertext* dengan indeks karakter *plaintext* yang terakhir dihitung.

P2 = (C1 – P1 + 25) mod 25
= (6 – 11 + 25) mod 25
= 20

Karakter yang memiliki indeks karakter 20 adalah U.

	G	H	E	O	O
	6	4	7	14	14
L	U				
11	20				

Selanjutnya, untuk mencari karakter *plaintext* dilakukan dengan cara yang sama:

P3 = (C2 – P2 + 25) mod 25
= (4 – 20 + 25) mod 25
= 9

Karakter yang memiliki indeks karakter 9 adalah ‘J’.

G	H	E	O	O
6	4	7	14	14
L	U	J		
11	20	10		

Jika algoritma ini dijalankan untuk menghasilkan *plaintext* level 1, maka hasilnya adalah sebagai berikut:

G	H	E	O	O	
6	4	7	14	14	
L	U	J	X	Q	X
11	20	10	23	16	23

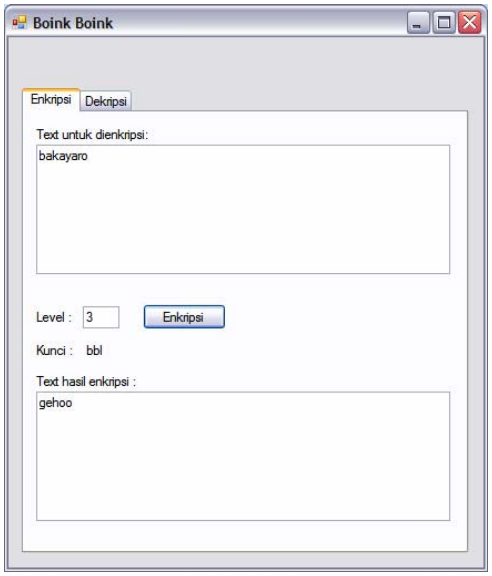
Untuk menghasilkan *plaintext* yang sesungguhnya, maka proses dekripsi perlu dilakukan sebanyak tiga kali, sesuai dengan panjang kuncinya. Proses yang dilakukan sama dengan cara di atas, namun karakter awal *plaintext* berikutnya diisi dengan karakter akhir dari kunci yang belum digunakan. Dalam kasus ini ‘B’.

Hasil akhir dekripsi akhirnya adalah sebagai berikut:

G	H	E	O	O		
6	4	7	14	14		
L	U	J	X	Q	X	
11	20	10	23	16	23	
B	K	K	Y	Y	R	G
1	10	10	24	24	17	6
B	A	K	A	Y	A	R	O
1	0	10	0	24	0	17	14

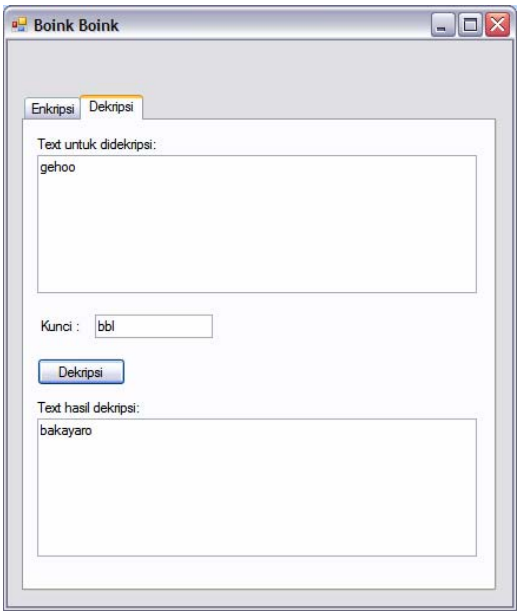
2.6. Implementasi

Rancangan algoritma baru ini diimplementasikan menggunakan bahasa pemrograman C# dan kakas Visual Studio 2005. Berikut adalah antarmuka implementasinya:



Gambar 6 Halaman Enkripsi

Pada halaman enkripsi di atas, terdapat *textarea* untuk menuliskan *plaintext* dan *textbox* untuk mengisi level enkripsi yang diinginkan. Untuk malakukan enkripsi, klik tombol Enkripsi, maka kunci dan *chipertext*nya akan ditampilkan.



Gambar 7 Halaman Dekripsi

Pada halaman dekripsi di atas, terdapat *textarea* untuk menuliskan *chipertext* dan *textbox* untuk mengisi kunci dekripsi yang diinginkan. Untuk malakukan dekripsi, klik tombol dekripsi, maka *plaintext*-nya akan ditampilkan.

Berikut adalah kode enkripsi dan dekripsi yang digunakan dalam implementasi algoritma ini:

```
//menghitung level
for (i = 0; i < levlen; i++)
{lev=(lev*10)+stringToInt(levelstring[i]);}

if (lev >= plainlen)
{//do nothing}
else
{//Inisialisasi Array of String
String[] plainlev = new String[lev + 1];
Char[] key = new Char[lev];
String[] keystr = new String[lev];

plainlev[0] = plaintext;
int len = 0;
String plaintemp;

//ENKRIPSI
for (i = 0; i < lev; i++)
{
   plaintemp = plainlev[i];
   len = plaintemp.Length;
   plainlev[i] = "";
   Char[] plainkar = new Char[len - 1];
   String[] plainstr = new String[len - 1];

   for (j = 0; j < len - 1; j++)
   {
        plainkar[j] = getChar(plaintemp[j],
        plantemp[j + 1]);
       plainstr[j]= plainkar[j].ToString();
      if (j==0)
      {
        key[i] = plaintemp[j];
        keystr[i] = key[i].ToString();
      }
    plainlev[i+ 1]=String.Concat(plainstr);
   }

  keyLabel.Text = String.Concat(keystr);
  chipertext1.Text = plainlev[i];
}
```

```
int chiperlen,lev,i, j;
String chipertext, key;

chipertext = chipertext2.Text.ToString();
chiperlen = chipertext.Length;
key = kunci.Text.ToString();
lev = key.Length;

//Inisialisasi Array of String
String[] chiperlev = new String[lev + 1];
chiperlev[0] = chipertext;
int len = 0;
String chipertemp;

//DEKRIPSI
for (i = 0; i < lev; i++)
{
  chipertemp = chiperlev[i];
  len = chipertemp.Length;
  chiperlev[i] = "";
  Char[] chiperkar = new Char[len + 1];
  String[] chiperstr = new String[len + 1];
  chiperstr[0] = key[lev-i-1].ToString();

  for (j = 0; j < len; j++)
  {
    chiperkar[j+1]=getChar2(chipertemp[j],
    chiperstr[j]);

chiperstr[j+i]=chiperkar[j+1].ToString();
  }
  chiperlev[i+1]=String.Concat(chiperstr);
}
plaintext2.Text = chiperlev[i];
```

2.7.Pengujian

Untuk menguji program kecil yang telah dibuat, digunakan *plaintext* sebagai berikut:

```
When the sun shines
We'll shine together
Told you I'll be here forever
Said I'll always be your friend
Took an oath
I'mma stick it out 'till the end
Now that it's raining more than ever
Know that we still have each other
You can stand under my Umbrella
You can stand under my Umbrella
(Ella ella eh eh eh)
Under my umbrella
(ella ella eh eh eh)
Under my umbrella
(ella ella eh eh eh)
Under my umbrella
(ella ella eh eh eh eh eh eh)
```

Sebelum melakukan enkripsi, karakter-karakter selain kedua-puluh enam abjad dihapus terlebih dahulu, karena indeks karakter yang didefinisikan hanya 26 abjad tersebut. Selanjutnya pilih level yang diinginkan, misalnya 23. Maka hasil enkripsinya adalah sebagai berikut:

```
fvemhvypymqxmdjbsthwbqhblyfjnumojpxqwtjnyubol
sihgmtmgfqifcivwxfvndqtyedbhxyvqkahkumncxvhrm
hgvjfwdfkveysylhmbahvrgjrfttdbvejqojichthseds
qykvyfoddtkngbslhmsraaonoflvrvwyhqqxoelslencq
hwspnrbcrfwcdohfowelsqqaojhhjrhscfojxwrdaxdjl
lubpjpsqkmtpnrhscfojxwrdaxdjllubpjpsqkmtpnrhs
cfojxwrdaxdjllubpjpsqkmtpnrhscfosymogm
```

Hasil kunci untuk dekripsinya adalah sebagai berikut:

`wepsuepsqyjbvkdawucrssv`

Jika*chipertext* tersebut didekripsi dengan kunci tersebut, maka diperoleh *plaintext*nya sebagai berikut:

```
whenthesunshineswellshinetogethertoldyouillbe
hereforeversaidillalwaysbeyourfriendtookanoat
himmastickitouttilltheendnowthatitsrainingmor
ethaneverknowthatwestillhaveeachotheryoucanst
andundermyumbrellayoucanstandundermyumbrellae
llaellaehehehundermyumbrellaellaellaehehehund
ermyumbrellaellaellaehehehundermyumbrellaella
ellaehehehehehehh
```

3. ANALISIS HASIL

3.1. Kelebihan Algoritma Boink2

Algoritma Kriptografi Boink2 memiliki kelebihan sebagai berikut:

a. Algoritma ini sangat mudah dimengerti dan dapat digunakan oleh siapa saja, karena pada dasarnya hanya diperlukan pengetahuan menghitung dan mengingat nilai indeks karakter.
b. Panjang *chipertext* yang berbeda dengan panjang *plaintext*nya, bahkan dapat didefinisikan sependek yang diinginkan.

c. Merupakan prinsip *confussion* dan *diffusion* dari Shannon yang sangat bagus karena menimbulkan kebingungan pada susunan karakter dan panjangnya *chipertext* yang tidak sama dengan teks aslinya.

3.2. Kekurangan Algoritma Boink2

Algoritma Kriptografi Boink2 memiliki kekurangan sebagai berikut:

a. Untuk proses enkripsi level 1 dimana pada *plaintext*nya terdapat banyak karakter berindeks karakter 0 (‘a’), maka pada *chipertext*nya masih dapat ditebak huruf pada *plaintext*nya.

   Plaintext : bakayaro
   Level : 1
   Chipertext: bkkyyrg

   Karakter ‘b’, ‘k’, ‘r’, dan ‘y’ muncul kembali pada *chipertext*, sehingga masih dapat diterka *plaintext*nya.

b. Suatu karakter yang diapit oleh dua karakter yang sama akan memberikan subtitusi karakter yang sama pula,sehingga mudah ditebak polanya.

   Plaintext : bakayaro
   Level : 1
   Chipertext: bkkyyrg

   Karakter ‘k’ dan ‘y’ yang muncul berurutan menunjukkan karakter pada *chipertext*nya diapit oleh dua karakter yang sama. Dalam kasus di atas diapit oleh huruf ‘a’.

c. Setiap karakter pada *chipertext* mempresentasikan penjumlahan 2 karakter. Jika pihak lawan mencoba menerka penjumlahan dari dua karakter tersebut, *chipertext* dapat terbongkar

   Contoh:
   c = a + c
   c = b+b

d. Jika pihak lawan melakukan *Exhausted Search*, untuk setiap karakter kunci, maka pada level tertentu *chipertext* tersebut pasti akan terbongkar. Kemungkinannya adalah sekali dalam 26^n percobaan, dengan n menunjukkan level, karena kriptanalis akan mencoba semua karakter pada awal *plaintext*nya.. Akan tetapi, kemungkinannya sangat kecil apabila proses enkripsinya dilakukan dengan level yang tinggi.

3.4. Saran dan Arah Pengembangan

Setelah mengetahui ide dasar dari algoritma ini, serta hasil analisa terhadap tingkat keamanannya, dapat dikatakan bahwa algoritma ini memiliki tingkat keamanan yang cukup rendah. Oleh karena itu, diperlukan berbagai masukan untuk pengembangan algoritma ini lebih lanjut.

a. Melakukan proses enkripsi pada level yang sangat tinggi (kunci yang panjang), sehingga kriptanalis akan membutuhkan waktu yang cukup lama karena proses perhitungan di dalam algoritmanya cukup banyak dan kemungkinan percobaan *exhausted search*-nya jugasemakin besar..
b. Mungkin dapat dibuat algoritma Boink2 enkripsi yang baru dimana kunci didefinisikan sendiri saat mengenkripsi. Sehingga untuk plainteks yang sama, dapat memiliki dua kunci yang berbeda.
c. Algoritma ini dapat dikembangkan untuk digunakan pada operasi bit, hexa, okta, dan basis bilangan lainnya dan dapat disisipkan pada algoritma modern.
d. Algoritma ini dapat digunakan sebagai algoritma enkripsi pada jaringan feistel sehingga fungsi *diffusion dan confussion*-nya menjadi semakin kuat.
e. Indeks karakter yang digunakan dapat dikembangkan tidak hanya 26 abjad, namun dapat ditambahkan karakter-karakter lainnya bahkan keseluruhan karakter ASCII sehingga kemungkinan serangan lebih sulit lagi karena kemungkinannya bertambah menjadi $1/K^n$, dimana K menunjukkan jumlah karakter yang didefinisikan dan n menunjukkan level.

4. KESIMPULAN

Kesimpulan yang dapat diambil dari studi perancangan algoritma Boink2 ini adalah:

a. Algoritma kriptografi klasik memang terlalu sederhana jika diterapkan sendiri-sendiri. Namun, kekuatannya akan meningkat bila dikombinasikan dengan sesama algoritma klasik maupun dengan algoritma modern.
b. Algoritma Boink2 pada dasarnya adalah algoritma subtitusi klasik yang merekayasa nilai-nilai indeks karakter pada *plaintext*nya
c. Algoritma Boink2 ini memiliki tingkat confussion dan diffusion yang cukup tinggi karena susunan karakter dan panjang *plaintext* aslinya tidak diketahui.
d. Algoritma Boink2 memiliki tingkat keamanan yang masih rendah, sehinggaperlu dikembangkan lagi.

DAFTAR REFERENSI

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF5054 Kriptografi. Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika , Institut Teknologi Bandung.

[2] Microsoft Visual Studio 2005 Documentation