Algoritma Scratter untuk Kriptografi pada Teks

Zulhendra Valiant Janir (13510045)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510045@stei.itb.ac.id

***Abstract*—Kriptografi adalah salah satu metode penyembunyian atau penyamaran suatu pesan yang bersifat privat sehingga pesan menjadi tersembunyi maknanya. Kriptografi mengubah suatu pesan menjadi sesuatu yang tidak bisa dibaca oleh pihak yang tidak berkepentingan atau yang tidak berhak untuk membacanya. Fungsi kontrolnya terdapat pada peran kunci pada kriptografi.**

Pada makalah ini, tipe kriptografi yang dibuat adalah tipe kriptografi kunci simetris pada teks dengan harapan algoritma buatan yang serumit mungkin dengan kemungkinan kunci sebanyak-banyaknya sehingga susah dipecahkan dengan metode *brute force* atau pun meminimalkan metode lain yang lebih efisien. Metode yang digunakan antara lain adalah transposisi, Vigenere Cipher, dan *Cipher Block Chaining* (CBC).

***Index Terms*—Kriptografi, kunci, transposisi, Vigenere Cipher, *Cipher Block Chaining*.**

I. PENDAHULUAN

Pada zaman teknologi saat ini, sudah banyak komunikasi yang menggunakan media teknologi pada komunikasi langsung maupun tidak langsung. Semakin banyak kebutuhan dalam berkomunikasi, semakin tinggi permintaan kelengkapan spek atau fitur pada alat komunikasi tersebut. Salah satu fungsi yang dibutuhkan saat ini adalah keamanan agar privasi informasi dapat terpenuhi.

Dalam kegiatan penting seperti transaksi bisnis, mengurus kriminal, penyimpanan barang berharga, dan data rahasia pribadi membutuhkan keamanan agar tidak merugikan pihak yang berhak untuk mengetahui dan memiliki informasi tersebut. Walaupun informasi atau data tersebut tidak ditujukan pada pihak lain, ada kemungkinan data tersebut dapat tersebar dan dilihat dengan bebas tanpa sepengetahuan pemilik akibat kebocoran informasi yang disebabkan lemahnya keamanan, virus, atau serangan dari pihak lain. Ada pun untuk berkomunikasi dengan pihak lain tanpa tersadap pihak ketiga, kedua pihak yang berkomunikasi dapat melakukan perjanjian dalam menggunakan kode atau simbol yang hanya diketahui oleh kedua belah pihak yang disampaikan secara bertatap muka atau metode lainnya yang memiliki kemungkinan kecil untuk diketahui oleh pihak yang tidak berhak untuk mengetahuinya. Namun dengan minimnya penggunaan algoritma rumit dan kemungkinan kunci yang sedikit, dalam kenyataannya metode ini hanya bertahan sebentar karena makna pesan sesungguhnya dapat ditebak dari pola yang ada dengan mempelajari banyak contoh pesan terenkripsi. Kondisi ini dikarenakan sedikitnya simbol atau karakter yang digunakan dan persepsi umum pada masyarakat terhadap representasi simbol.

Dengan mempertimbangkan hal di atas, ada baiknya pemenuhan kebutuhan keamanan dalam berkomunikasi atau menyimpan informasi dengan teknik kriptografi algoritma baru serumit mungkin. Konsep yang diterapkan pada algoritma baru ini adalah mengacak pesan dan mengubahnya menjadi simbol atau karakter dengan jarak kode *unicode* yang tersebar atau berbeda jauh jika kode *unicode* pesan asli berjarak dekat.

Ada pun pemenuhan kebutuhan umum dalam algoritma kriptografi dikelompokkan menjadi 4, yaitu:

1. *Confidentiality*

   Confidentiality mengharuskan pesan asli harus serancu mungkin sehingga susah untuk ditebak. Jika pihak yang tidak berhak memiliki banyak referensi algoritma, maka pihak tersebut dapat dengan mudah menerka algoritma yang digunakan maupun kuncinya. Sebisa mungkin algoritma yang digunakan adalah algoritma yang rumit dengan *range* kunci sebesar mungkin sehingga kriptanalisis membutuhkan pengorbanan berupa waktu dan tenaga yang sebesar mungkin pula.
2. *Authentication*

   Authentication menyediakan informasi tersirat mengenai identifikasi pengirim pesan. Hal ini dibutuhkan karena pada pengiriman pesan dapat terjadi pengakuan pengiriman pesan dari pengirim yang salah sehingga dapat merusak jalur komunikasi serta pembocoran informasi dengan adanya pihak yang seharusnya tidak berhak untuk mengetahuinya. Salah satu solusinya adalah dengan menggunakan kunci yang sama pada pihak tertentu saja sehingga kedua pihak yang berkomunikasi mengetahui pengirim dan

penerima pesan saat itu dari informasi kunci tersebut.

3. *Integrity*

   Jika pihak yang tidak berhak untuk membaca atau menerima suatu pesan enkripsi tidak dapat mengerti arti dalam pesan tersebut, maka ada kemungkinan baginya untuk mengubah bahkan menghancurkan pesan tersebut sehingga arti yang sesungguhnya rusak ketika sampai pada penerima atau bahkan penerima tidak dapat menerima pesan sama sekali. Jika pesan yang terenkripsi diubah dan hasil dekripsinya tidak rusak atau masih bermakna, maka hal ini dapat merugikan penerima karena pesan asli berhasil diubah maknanya. Sebaiknya algoritma enkripsi yang digunakan dapat mendeteksi perubahan sekecil mungkin pada pesan enkripsi yang telah diubah.

4. *Nonrepudiation*

   Demi suatu kepentingan atau manipulasi di luar penerapan kriptografi, pengirim pesan dapat menyangkal bahwa dia telah mengirim pesan tertentu dengan cara mengubah kembali hasil enkripsi pesannya. Cara mengatasinya adalah dengan membuat deteksi perubahan hasil enkripsi dengan menyimpan data pesan asli pada proses enkripsi sehingga nantinya dalam proses dekripsi dapat dilakukan verifikasi.

II. METODE

Pada mulanya, pengguna melakukan input pesan yang akan dikirim kepada penerima pesan. Pesan ini harus terlebih dahulu dienkripsi sebelum sampai kepada penerima. Algoritma yang digunakan adalah algoritma baru yang dinamakan Scratter (*Scramble* and *Scatter*). Algoritma ini terdiri dari tiga tahap, yaitu tahap transposisi, tahap *Vigenere Cipher*, dan CBC.

A. Enkripsi

1. Tahap Transposisi

   Pada tahap ini, pesan akan diacak berkali-kali dengan pola yang sama. Sebelum diacak, pengguna harus menentukan kunci yang akan digunakan. Kunci yang digunakan tidak memiliki batas tertentu, tapi semakin besar panjang kuncinya, semakin banyak pengacakannya. Contoh kasus untuk mengatur kuncinya adalah sebagai berikut.

```
Kunci umum : 20s45
->Panjang kunci umum : 5,
->Generate kunci acak dari “20s45”
->’2’ -> kunci acak = 0
->’0’ -> kunci acak = 3
->’s’ -> kunci acak = 3
->’4’ -> kunci acak = 0
->’5’ -> kunci acak = 3
->kunci acak = 3

Kunci umum : 20s3
->Panjang kunci umum : 4,
->Generate kunci acak dari “20s3”
->’2’ -> kunci acak = 2
->’0’ -> kunci acak = 2
->’s’ -> kunci acak = 1
->’3’ -> kunci acak = 0
->kunci acak = 0
->kunci acak = 2

Kunci umum : saya suka main
->Panjang kunci umum : 14,
->Generate kunci acak dari “saya
suka main”
->’s’ -> kunci acak = 3
->’a’ -> kunci acak = 2
->’y’ -> kunci acak = 11
->’a’ -> kunci acak = 10
->’ ’ -> kunci acak = 0
->’s’ -> kunci acak = 3
->’u’ -> kunci acak = 8
->’k’ -> kunci acak = 3
->’a’ -> kunci acak = 2
->’ ’ -> kunci acak = 6
->’m’ -> kunci acak = 3
->’a’ -> kunci acak = 2
->’i’ -> kunci acak = 9
->’n’ -> kunci acak = 7
->kunci acak = 7
```

Dapat diamati, dari contoh kasus ketiga kunci umum tersebut dapat membuat kunci acak yang sangat jauh berbeda. Ada pun rumus untuk menentukannya melibatkan panjang kunci umum dan karakter yang akan digunakan. Pada program java, rumusnya sebagai berikut.

```
KunciAcak=0;
for(int i=0; i<KunciUmum.length();
i++){
  KunciAcak=(KunciAcak+KunciUmum.c
  harAt(i))%KunciUmum.length();
}
```

Pada rumus di atas, panjang kunci umum akan membatasi kunci acak sehingga makin panjang kunci umum, maka semakin besar kemungkinan kunci acak akan terbentuk. Sedangkan penentuan kunci acak akan diakumulasi dari nomor karakter pada tabel desimal *Unicode*.

Perlu diperhatikan juga, pada algoritma ini juga dapat menghasilkan kunci acak bernilai 0 atau 1, tapi nilai tersebut akan menjadi tidak berarti jika digunakan pada tahap pengulangan berikutnya, sehingga nilai minimal kunci acak dibatasi, yaitu 2.

Setelah mendapatkan kunci acak dari kunci

umum, algoritma akan memproses pesan asli menjadi pesan teracak. Berikut dapat dilihat contohnya.

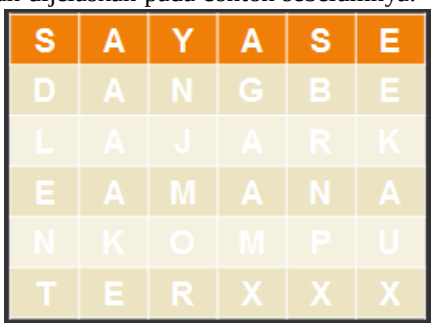
```
Pesan : Aku tak tahan lagi
Kunci umum : 20s45
->kunci acak = 3
->Teracak = “h guataliA kanakt ”
->Teracak = “ utlAknk hgaai aat”


Pesan : Aku tak tahan lagi
Kunci umum : 20s3
->kunci acak = 0
->kunci acak = 2
->Teracak = “k a aa aiAutkthnlg”

Pesan : Aku tak tahan lagi
Kunci umum : saya suka main
->kunci acak = 7
->Teracak = “ hitaank A lktauag”
->Teracak = “tAua aalg nkhkti a”
->Teracak = “lhaagk  ttaniAak u”
->Teracak = “ua aaa nklgti hktA”
->Teracak = “g t akthAua likaan”
->Teracak = “  khu iangtatAalka”
```

Jika proses pertama ditelaah per proses, maka yang terjadi pada algoritma adalah sebagai berikut.

```
Pesan : Aku tak tahan lagi

->kunci acak = 3
->Teracak = “Aku”,“ ta”,“k t”,
“aha”,“n l”,“agi”
->Teracak = “A kana”,“kt h lg”,
“uatali”
->Teracak = “A kanakt h lguatali”
->Teracak = “A kanakt ”,
“h lguatali”
->Teracak = “h lguatali”,
“A kanakt ”
->Teracak = “h guataliA kanakt ”
->kunci acak = 2
...
->Teracak = “ utlAknk hgaai aat”
```

Proses pertama yang dilakukan pada pengacakan adalah membagi pesan menjadi beberapa partisi dengan masing-masing partisi memiliki panjang sebesar nilai kunci acak. Kemudian pesan akan dikelompokkan berdasarkan urutan, untuk kelompok pertama adalah huruf pertama dari kelompok-kelompok pada proses sebelumnya, lalu huruf kedua, sampai huruf ke-n sesuai dengan nilai kunci acak sehingga terkumpul n kelompok. Lalu pesan tersebut disambung dan kemudain dibagi dua sama panjang. Kemudian urutan kelompok pesan pertama ditukar dengan yang kedua. Lalu pada akhirnya disambung kembali dan proses pengacakan selesai untuk putaran pertama. Putaran berikutnya akan mengacak pesan dengan kunci acak - 1 dari kunci acak sebelumnya sampai kunci acak bernilai sama dengan 2.

Berikut adalah gambaran dari metode yang telah dijelaskan pada contoh sebelumnya.



Gambar 1 Teknik Transposisi Vertikal

Pada gambar di atas, pesan yang dimasukkan adalah “SAYASEDANGBELAJARKEAMANAN KOMPUTER” dan akan diacak menjadi “SDLE NTAAAAKEYNJMORAGAAMSBRNPEEKAU” lalu ditukar menjadi “ORAGAAMSBRNPEEKAU SDLENTAAAAKEYNJM” dengan kunci acak bernilai 6. Lalu akan terjadi pengulangan sebanyak KunciAcak-1 kali dengan urutan kunci acak 6, 5, 4, 3, dan kemudian 2.

2. Tahap *Vigenere Cipher*

Pada tahap *Vigenere Cipher*, pesan teracak akan dilanjutkan dengan enkripsi menggunakan kunci umum juga. Berikut adalah contoh enkripsinya.

```
Pesan : Aku tak tahan lagi
Kunci umum : 20s45
->Teracak = “ utlAknk hgaai aat”
->Vig Cip = “R¥ç v  Þ T ™‘Ô U“‘ç”

Pesan : Aku tak tahan lagi
Kunci umum : 20s3
->Teracak = “k a aa aiAutkthnlg”
->Vig Cip = “ PÔS“‘“”›qè§ ¤Û¡ —”

Pesan : Aku tak tahan lagi
Kunci umum : saya suka main
->Teracak = “  khu iangtatAalka”
->Vig Cip = ““ äÉ•“ÞÌÏ‡áÂÝ¯ÔÍäÂ”
```

Vigenere Cipher pada algoritma ini tidak dibatasi, yaitu menggunakan domain *Unicode* yang jenis karakternya sebanyak 65536 karakter. Dengan banyaknya kemungkinan karakter seperti ini, diharapkan dapat memberikan kerancuan hasil enkripsi serancu mungkin.

Ada pun penggunaan algoritma Vigenere Cipher tidak berbeda dari algoritma Vigenere Cipher yang telah ada, yaitu menggeser karakter pada pesan sebesar nilai *Unicode* pada kunci umum yang bersesuaian. Berikut adalah kode program dan contoh penentuan pasangan bersesuaiannya.

```
for
(int i=0;i<Teracak.length();i++)
VigCip+=(char)(
((int)Teracak.charAt(i)+
(int)KunciUmum.charAt
(i%KunciUmum.length())
)%65536);

->Teracak = " utlAknk hgaai aat"
->Vig Key = "20s4520s4520s4520s"
->-----------------------------+
->Vig Cip = "R¥ç v  ÞT ™'Ô U"'ç"
```

Pada contoh di atas, dapat teramati bahwa kunci umum tidak harus sepanjang pesan untuk memenuhi kebutuhan, melainkan kunci umum tersebut yang beradaptasi, berulang sampai sepanjang pesan yang akan dienkripsi.

3. Tahap CBC

Pada tahap ini, hasil dari *Vigenere Cipher* akan diambil bit-bitnya untuk diproses dengan operasi *xor*. Pemrosesan CBC yang digunakan adalah melakukan xor tiap byte sepanjang 8 bit dengan tiap karakter dipisah menjadi 2 byte terurut. Berikut adalah transformasi karakter menjadi byte.

```
->Vig Cip = "R¥ç v  ÞT ™'Ô U"'ç"
->Char1 = "00000000", "01010010"
->Char2 = "00000000", "10100101"
->Char3 = "00000000", "11100111"
->Char4 = "00000000", "10100000"
->Char5 = "00000000", "01110110"
->Char6 = "00000000", "10011101"
->Char7 = "00000000", "10011110"
->Char8 = "00000000", "11011110"
->Char9 = "00000000", "01010100"
->Char10 = "00000000", "10011101"
->Char11 = "00000000", "10011001"
->Char12 = "00000000", "10010001"
->Char13 = "00000000", "11010100"
->Char14 = "00000000", "10011101"
->Char15 = "00000000", "01010101"
->Char16 = "00000000", "10010011"
->Char17 = "00000000", "10010001"
->Char18 = "00000000", "11100111"
```

Dari contoh di atas, sangat terlihat jelas bahwa karakter hanya berkisar di domain ASCII. Dengan adanya pemisahan per 8 bit, diharapkan penyebaran angka '1' lebih tersebar di kedua partisi.

Setelah rangkaian bit telah terbentuk dari pesan hasil enkripsi *Vigenere Cipher*, selanjutnya algoritma akan melakukan xor pada suatu byte terhadap byte sebelumnya yang telah dienkripsi menggunakan CBC secara berurutan. Bagi byte pertama, dibutuhkan *Initialize Vector* (IV) yang berupa byte buatan yang pasti. Dalam algoritma ini, IV bernilai "00000000".

Adapun proses pada enkripsi kunci ada dua tahap, yaitu enkripsi dengan kunci dan *shift left*. Kunci kali ini juga dapat dibentuk dari kunci umum yang panjangnya tidak dibatasi dan tidak ditentukan, tapi jhasil akhir dari kunci CBC harus memiliki panjang tepat 8 bit. Berikut adalah contoh penerapannnya.

```
->Kunci umum = "20s45"
->'2' -> kode Unicode = 50 -> '0'
->'0' -> kode Unicode = 48 -> '0'
->'s' -> kode Unicode =115 -> '1'
->'4' -> kode Unicode = 52 -> '0'
->'5' -> kode Unicode = 53 -> '1'
->Kunci CBC = "00101"
->Kunci CBC = "0010101"
->Kunci CBC = "001010101"
->Kunci CBC = "00101010"
```

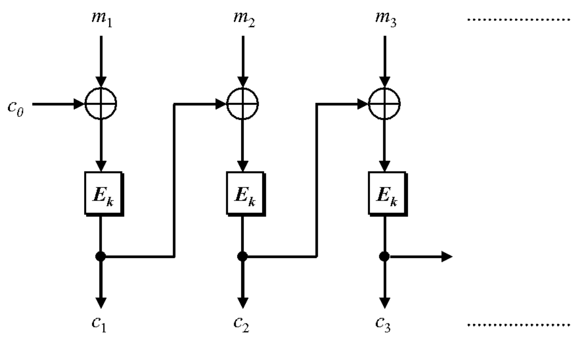
Pada proses pencarian kunci tersebut, tiap karakter kunci umum diekstrak menjadi kode desimal *Unicode* dan dicek apakah dapat dibagi 2 atau tidak, jika dapat maka bit yang diambil adalah 0, atau bit yang diambil adalah 1 jika kode desimal tidak dapat dibagi 2. Jika panjang kunci umum tidak mencukupi spek kunci CBC, maka kunci CBC saat itu ditambah "01" secara terus-menerus sampai kunci CBC saat itu panjangnya melebihi 7 bit. Jika kunci CBC sudah melebihi 7 bit, maka 8 bit pertama pada kunci akan diambil sebagai kunci yang akan digunakan pada tahap CBC dalam algoritma ini. Berikut adalah aturan xor dan pemrosesan kunci terhadap byte yang akan dienkripsi.

```
'1' ⊕ '1' -> '0'
'1' ⊕ '0' -> '1'
'0' ⊕ '1' -> '1'
'0' ⊕ '0' -> '0'

"01010101" ⊕ KunciCBC
="01111101" ⊕ "00101010"
="01010111"
```

Selanjutnya adalah proses *shift left* yaitu menggeser atau memutar bit pada tiap byte sebanyak 1 bit ke kiri. Berikut adalah contohnya.

```
“11011110” -> “10111101”
“11000000” -> “10000001”
“10101010” -> “01010101”
```

Berikut adalah gambar pemrosesan CBC secara keseluruhan.



Gambar 2 Proses Enkripsi CBC

Berikut adalah contoh serta hasil dari proses keseluruhan CBC di atas.

```
->Byte1 = Byte1 ⊕ IV ⊕ KeyNshift
        = “00000000” ⊕ KeyNshift
        = “10101000”
->Byte2 = Byte2 ⊕ Byte1 ⊕
KeyNshift
        = “11111010” ⊕ KeyNshift
        = “10100100”
...
->Byte35 = Byte35 ⊕ Byte34 ⊕
KeyNshift
         = “11101111” ⊕ KeyNshift
         = “11011101”
->Byte36 = Byte36 ⊕ Byte35 ⊕
KeyNshift
         = “11011000” ⊕ KeyNshift
         = “01000110”
```

4. Tahap Pembuatan Cipherteks

   Setelah semua byte telah melakukan CBC, maka langkah terakhir adalah menggabungkan tiap 2 byte dan mengubahnya menjadi karakter. Berikut adalah hasil enkripsi dari algoritma Scatter.

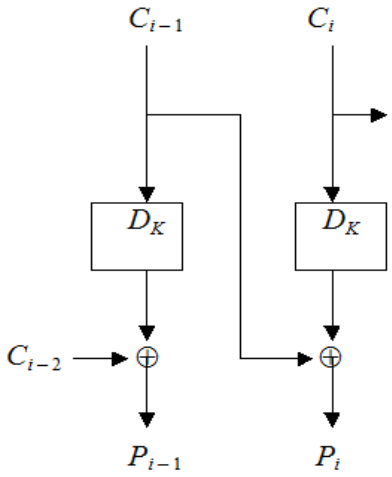
```
->Pesan = “Aku tak tahan lagi”
->Kunci Umum = “20s45”
->Cipherteks =
“咤 •昰瑡詑? •?扻猰•合•躍生?ئ?”

->Pesan = “Aku tak tahan lagj”
->Kunci Umum = “20s45”
->Cipherteks =
“吧猄傿•꾖錡瞜핫•칢黷ஃ⊙•膝镸•”

->Pesan = “Ak tak tahan lagi”
->Kunci Umum = “20s45”
->Cipherteks =
“吧猄偉 •흑鄣瞄 • •곧鳲ॠ᷍ 黄긺 •”

->Pesan = “Aku tak tahan lagi”
->Kunci Umum = “20s3”
->Cipherteks =
“樌军 •墔ﺳ?•?▪쌓↫•뾂葴춭稞懰c”

->Pesan = “Aku tak tahan lagi”
->Kunci Umum = “saya suka main”
->Cipherteks =
“ •젤쯤ɾ酕멲鶌 •막•↠•滲픔词ㅍ릶瀚”
```

Pada kelima contoh tersebut, terlihat hasilnya cukup tersebar dan sangat berbeda, apalagi dengan susahnya mengidentifikasi karakter *Unicode* yang jarang ditemui dan memiliki kode desimal yang sangat tinggi. Namun ketersebarannya hampir tidak pernah di bawah 256 karena pemrosesan byte memisahkan 8 bit awal dengan 8 bit akhir pada karakter sehingga hampir tidak mungkin mendapatkan bit awal bernilai “00000000”. Terdapat pula karakter yang tidak dapat terbaca pada font atau program tertentu dikarenakan jarangmya penggunaan pada karakter tersebut sehingga tidak semua aplikasi dapat menggunakan algoritma yang berdomain *Unicode* ini.

B. Dekripsi

1. Tahap CBC

   Merupakan tahap kebalikan dari enkripsi. Ada pun setelah pengubahan karakter menjadi bit-bit, proses CBC dapat dijelaskan pada gambar berikut.



Gambar 3 Proses Dekripsi CBC

Berikut adalah contoh serta hasil dari proses dekripsi CBC di atas.

```
->Byte1 = Byte1 ⊕ IV ⊕ KeyNshift
        = “00000000”
->Byte2 = Byte2 ⊕ Byte1lama ⊕
KeyNshift
        = “01010010”
...
->Byte35 = Byte35 ⊕ Byte34lama ⊕
KeyNshift
         = “11011101”
->Byte36 = Byte36 ⊕ Byte35lama ⊕
KeyNshift
      = “01000110”
```

2. Tahap *Vigenere Cipher*
   Pada tahap dekripsi kali ini juga hanya membalik proses *Vigenere Cipher* seperti yang telah dijelaskan di proses enkripsi.

```
Kunci umum : 20s45
->Vig Cip = “R¥ç v  ÞT ™‘Ô U“‘ç”
->Teracak = “ utlAknk hgaai aat”

Kunci umum : 20s3
->Vig Cip = “ PÔS“‘“”›qè§ ¤Û¡ —”
->Teracak = “k a aa aiAutkthnlg”

Kunci umum : saya suka main
->Vig Cip = “” äÉ•“ÞÌÏ‡áÂÝ¯ÔÍäÂ”
->Teracak = “  khu iangtatAalka”
```

Ada pun algoritma yang digunakan berubah menjadi sebagai berikut.

```
for(int i=0;i<VigCip.length();i++)
Teracak+=(char)(
((int)Teracak.charAt(i)-
(int)KunciUmum.charAt
(i%KunciUmunm.length())
+65536)%65536);
```

3. Tahap Transposisi
   Tahap dekripsi transposisi juga membalikkan tahap enkripsi transposisi. Jika *generator* kuncinya bernilai 6, maka penyusunan dilakukan dengan kunci 2, 3, 4, 5, dan yang terakhir adalah 6 secara terurut. Ada pun cara untuk mengembalikannya lebih rumit daripada enkripsinya jika panjang pesan bukan kelipatan dari kunci. Berikut adalah contoh pengerjaannya.

```
->Teracak = “ utlAknk hgaai aat”
->kunci acak = 2
->Teracak = “ utlAknk ”,
“hgaai aa”
->Teracak = “hgaai aat”,
“ utlAknk ”
->Teracak = “hgaai aat utlAknk ”
->Teracak = “hgaai aat”,
“ utlAknk  ”
->Teracak = “h ”,“gu”,“at”,“al”,
“iA”,“ k”,“an”,“ak”,“t ”
->Teracak = “h guataliA kanakt ”
->kunci acak = 3
...
->Teracak = “Aku tak tahan lagi”
```

   Pertama-tama, Teracak dibagi dua, jika panjangnya ganjil, maka yang ganjil adalah partisi kiri dengan panjang partisi kanan lebih panjang 1 karakter. Lalu ditukar posisi keduanya dan disambung kembali. Kelompokkan Teracak dengan panjang maksimal sebanyak kunci acak dan dahulukan karakter awal yang memiliki panjang maksimal. Pengelompokkan dilakukan dengan lompat sebanyak kunci acak ke kanan, sebanyak panjang pesan *mod* kunci acak jika panjang pesan tidak habis dibagi kunci acak. Setelah lompat sebanyak panjang pesan mod kunci acak, dilanjutkan loncat sebesar kunci acak-1 ke kanan. Lalu untuk karakter kedua dan seterusnya mengikuti pola sebelumnya. Lalu seluruhnya digabungkan secara berurutan. Jika putaran masih ada, maka dilanjutkan untuk melakukan transposisi dengan kunci acak + 1.
4. Tahap Pembuatan *Plaintext*
   Plaintext adalah hasil terakhir proses dekripsi transposisi.

III. ANALISIS

Dari hasil percobaan dengan menggunakan algortima Scatter, cipherteks yang dihasilkan dapat beragam tergantung kuncinya. Semakin panjang kunci yang digunakan, maka semakin tinggi kemungkinan mendapatkan cipherteks yang semakin tersebar dan teracak sehingga kriptanalis yang ingin menemukan

kuncinya dan bergantung pada posisi karakter akan mengalami kesulitan dalam menemukan kuncinya. Keterkaitan dan kedekatan antar karakter juga tidak dapat dilacak karena adanya algoritma CBC yang berkomplemen satu sama lain.

Namun karena perubahan satu karakter dapat berpengaruh terhadap karakter lainnya, algoritma ini tidak cocok untuk spek yang membutuhkan algoritma dengan *integrity* yang tinggi. Selain itu karena penggunaan Unicode yang sering, algoritma ini menghasilkan cipherteks yang tidak dapat dibaca pada kebanyakan blok teks di suatu aplikasi.

IV. KESIMPULAN

Algoritma Scatter sangat cocok untuk digunakan pada mesin atau alat elektronik yang memiliki spek tinggi sehingga dapat membaca karakter *Unicode* serta terjamin akan *intregity*-nya dalam pengiriman pesan.

Semakin panjang kuncinya, semakin susah untuk menebak kuncinya. Hal ini dapat dilihat dari besarnya kunci dan keragaman cipherteks yang dihasilkan.

Teknik CBC dan transposisi tidak cocok untuk, menjamin integritas pesan karena memilik ketergantungan terhadap karakter lainnya.

Untuk menghindari Unicode, karakter yang digunakan cukup berdomain ASCII.

IV. DAFTAR GAMBAR

REFERENSI

[1] Munir, Runaldi. 2005. *Diktat Kuliah IF5054 Kriptografi*. Departemen Teknik Informatika, Institut Teknologi Bandung.
[2] http://eprints.undip.ac.id/25368/2/ML2F306042.pdf (17.38, 24 Maret 2013)
[3] http://www.slideshare.net/adentya/multi-enkripsi-dengan-teknik-transposisi-dan-scytale (19.20, 24 Maret 2013)
[4] http://www.rsa.com/rsalabsnode.aspid=2171 (20.09, 25 Maret 2013)

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, 26 Maret 2013