PENERAPAN CHIPER TRANSPOSISI-REKURSIF KONKATENASI STRING KODE ASCII DOKUMEN TEKS

Masykur Marhendra S.N. – NIM : 13504063

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if14063@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang penerapan algoritma cipher transposisi rekursif dalam menyandikan data yang disimpan dalam media penyimpanan. Cipher transposisi termasuk dalam cipher klasik yang digunakan sebelum berkembangnya teknologi komputer. Secara sederhana, cipher transposisi adalah operasi permutasi terhadap kata-kata dari sebuah pesan. Pengaturan dan pertukaran yang dilakukan merupakan operasi blok dengan aturan-aturan tertentu menggunakan kunci tertentu pula. Variasi implementasi dari cipher transposisi cukup banyak. Diantaranya makalah ini akan sedikit membahas tentang beberapa variasi implementasi cipher transposisi sebagai perbandingan terhadap penerapan cipher transposisi secara rekursif.

Cipher transposisi rekursif merupakan operasi blok transposisi terhadap kode ASCII dari pesan yang akan dienkripsi. Sebelum dienkripsi, kode ASCII tadi di-*plain*-kan menjadi deretan bilangan sehingga transposisi dapat dilakukan terhadap satuan bilangan-bilangannya. Setelah itu kita ambil bilangan sepanjang kunci (k) dari bilangan yang ke-n untuk ditransposisikan secara rekursif dari bilangan paling luar sampai ke dalam dari blok tersebut. Kemudian dilanjutkan ke bilangan n+1 dengan panjang blok (k+k/2). Apabila panjang blok (k+k/2) melebihi panjang bilangan sisa, maka kita ambil kunci yang dipakai sebelumnya. Setelah selesai ditransposisikan kita kodekan kembali deretan bilangan tadi menjadi karakter ASCII sesuai panjang masing-masing kode ASCII plainteks sebelumnya.

Secara manual teoritis algoritma ini tidak mempunyai permasalahan. Namun ketika mengimplementasikan pada program aplikasi terdapat beberapa kendala. Diantaranya yaitu apabila ditemukan kode seperti 002, maka program hanya akan mengkodekan sebagai karakter ASCII 2. Sehingga ketika pembacaan kembali untuk dekripsi akan menjadikan suatu masalah. Hal ini karena panjang kode untuk pentransposisian tidak sama ketika pengenkripsian. Beberapa solusi telah saya implementasikan untuk mengatasi permasalahan ini. Dan akhirnya, saya menemukan solusi cukup tepat meskipun kurang mangkus.

***Kata kunci**: analisis frekuensi, cipherteks, cipher substitusi, Columnar Transposition, dekripsi, dummy char, eliminasi Gauss-Jordan, enkripsi, key streaming, Known Plainteks Attack, kode ASCII, kriptanalis, kriptanalisis, kunci, matriks augmentasi, matriks identitas, matriks kolom, matrix cypher, permutasi, plainteks, Rail Fence, rekursif , reversible, Route Cipher, transposisi, vektor kolom.*

1. Pendahuluan

Perkembangan dunia digital saat ini membuat lalu lintas pengiriman data elektronik semakin ramai. Hampir setiap orang melakukan transaksi data setiap harinya. Data yang dipertukarkan pun juga bervariasi baik dari jenisnya maupun tingkat kerahasiaannya. Mulai dari data pribadi,data organisasi sampai dengan data negara yang sangat rahasia. Hal inilah yang menuntut adanya pengamanan data tersebut sehingga tidak sampai tersadap oleh pihak ketiga. Sampai saat ini telah banyak ditemukan teknik-teknik dalam pengamanan data, baik teknik klasik maupun modern.

Chiper transposisi merupakan salah satu teknik klasik dalam pengamanan data. Teknik ini secara sederhana melakukan permutasi karakter pada pesan plainteks, yaitu dengan menyusun ulang urutan karakter dalam pesan. Penyusunan ini berdasarkan blok-blok dengan panjang tertentu

Katakanlah kita memilih ukuran blok 5 dan dari masing-masing blok kita spesifikasikan sebagai berikut :
Karakter pertama menjadi karakter keempat,
Karakter kedua menjadi karakter ketiga,
karakter ketiga menjadi pertama,
karakter keempat menjadi karakter kelima, dan
karakter kelima menjadi karakter kedua.
[1]

Atau secara matematis untuk menuliskan permutasi diatas dengan menggunakan notasi matriks :

(1 4 5 2 3)

notasi ini akan lebih bermakna daripada pernyataan sebelumnya.

Misalkan kita mempunyai plainteks

```
THE SKY  FALLING PLEASE ADVISE
```

dengan pemisahan karakter menjadi blok ukuran lima. Jika kita memakai permutasi yang telah didefinisikan pada [1], kita akan mendapat pesan dengan karakter teracak sebagai cipherteks

```
EKHTS ALFYL GLNIP SAAEE IEVDS
```

Berdasarkan contoh diatas, cipher transposisi cenderung lemah terhadap serangan-serangan kriptanalis, terutama analisis frekuensi. Siapapun yang pernah bermain anagram atau bermain puzzle akan dapat dengan mudah memecahkan cipherteks tersebut. Melihat begitu besarnya kelemahan cipher ini, algoritma yang dikembangkan saat ini tidaklah murni seutuhnya, biasanya dikombinasikan dengan algoritma cipher substitusi. Hal ini terkadang dapat memberi kekuatan tersendiri pada cipherteks sehingga sulit untuk dikriptanalisis.

2. Variasi implementasi cipher transposisi

2.1 *Rail Fence*

Pada *Rail Fence* cipher, plainteks dituliskan secara vertikal ke bawah sepanjang n-rails, dan menulis lagi ke kolom baru ketika telah mencapai karakter ke-n. Cipherteks yang dihasilkan adalah urutan karakter yang dibaca secara horizontal. Sebagai contoh,kita mempunyai n=3 dan sebuah pesan

```
WE ARE DISCOVERED FLEE AT ONCE
```

sang cipher menulis :

```
W R I O R F E O E P
E E S V E L A N J D
A D C E D E T C X Q
```

Karakter tambahan di akhir cipherteks sengaja dibubuhkan diantaranya untuk melengkapi cipherteks sehingga melengkapi blok dan atau untuk mengelabui kriptanalis. Pesan tersebut kemudian dibaca

```
WRIOR FEOEP EESVE LANJD ADCED ETCXQ
```

Penulisan pesan menjadi blok-blok standar, biasanya sepanjang 5 karakter, dilakukan
untuk memudahkan pentransmisian pesan pada telegraf. Algoritma Rail fence cipher ini tidak terlalu kuat, kemungkinan kunci-kunci yang dipakai terlalu kecil sehingga kriptanalis dapat mencoba semuanya dengan manual.

Cara termudah dalam mendekripsi sebuah cipherteks Rail Fence adalah dengan melalui metode berikut :

1. Pertama, tuliskan kembali cipherteks menjadi satu deretan string/karakter

```
WRIORFEOEPEESVELANJDADCEDETCXQ
```

2. Kemudian,bagi pesan tersebut menjadi blok dengan ukuran sama dan mempunyai jumlah blok sama dengan ukuran rails

```
WRIORFEOEP|EESVELANJD|ADCEDETCXQ
```

Karena terdapat 30 karakter,dan kita tahu bahwa jumlah rails ada 3,kita membagi blok menjadi 3 dengan panjang masing-masing blok adalah 10

3. Langkah terakhir, tulis karakter pertama dari blok ke-1,blok ke-2,blok ke-3 diikuti dengan karakter kedua dari blok ke-1,blok ke-2,blok ke-3 sampai karakter ke-n.

```
WEAREDISCOVEREDFLEEATONCEJXPDQ
```

4. Sekarang pilah-pilah karakter dari pesan sehingga dapat mudah dibaca, hilangkan karakter dummy, dan kode pun telah berhasil didekripsi

2.2 *Route Cipher*

Pada *Route Cipher*, plainteks dituliskan di dalam grid dengan dimensi tertentu (n), kemudian pembacaan pesan dilakukan dengan pola

tertentu yang didefinisikan didalam kunci. Sebagai contoh, menggunakan plainteks dan grid yang sama seperti pada *Rail Fence*

```
W R I O R F E O E
E E S V E L A N J
A D C E D E T C X
```

Pola kunci dapat didefinisikan sebagai ‘spiral kebelakang,searah jarum jam, dimulai dari karakter paling atas kanan. Yang akan memberikan cipherteks :

```
EJX CTE DEC DAE WRI ORF EON ALE
VSE
```

(Pembagian menjadi blok-blok dengan panjang 3 diharapkan dapat membantu untuk menghindari kesalahan)

Route Cipher mempunyai kemungkinan kunci lebih banyak daripada *Rail Fence*. Bahkan, untuk pesan-pesan dengan panjang tertentu, jumlah kemungkinan kunci yang ada sangat besar untuk dienumerasi bahkan untuk mesin modern. Tetapi, tidak semua kunci mempunyai kekuatan yang sama. Jika salah memilih rute pengenkripsian malah akan meninggalkan potongan plainteks yang berlebihan atau teks akan mudah dipecahkan, dan hal ini akan memberikan kriptanalis petunjuk untuk mencari rute yang dijadikan sebagai kunci.

2.3 *Columnar Transposition*

Pada *Columnar Transposition*, pesan ditulis dalam baris dengan panjang tertentu, kemudian dibaca kembali dari kolom ke kolom. Pembacaan per kolomnya berdasarkan urutan yang acak. Panjang baris dan permutasi kolomnya biasanya didefinisikan oleh sebuah kata kunci. Sebagai contoh kata ZEBRAS mempunyai panjang 6 (sehingga panjang baris adalah 6) dan permutasi didefinisikan dengan urutan alphabet dari kata kunci. Dengan kata ZEBRAS, maka urutannya akan menjadi ‘6 3 2 4 1 5’

Pada cipher *Columnar Transposition* yang umum, semua area kosong diisi dengan dengan null (*dummy char*), sedangkan pada beberapa *Columnar Transposition* yang lain, area kosong tetap dibiarkan kosong.

Sebagai contoh dari *Columnar Transposition*, misalkan kita mempunyai kata kunci ZEBRAS dan pesan

```
WE ARE DISCOVERED FLEE AT ONCE
```

Pada Columnar cipher yang umum, kita tuliskan pesan tersebut ke dalam grid menjadi :

```
6 3 2 4 1 5
W E A R E D
I S C O V E
R E D F L E
E A T O N C
E Q K J E U
```

dengan tambahan 5 buah null (QKJEU) di akhir. Kemudian cipherteks dibaca menjadi sebagai :

```
EVLNE ACDTK ESEAQ ROFOJ DEECU
WIREE
```

Sedangkan pada kasus unik, kolom yang kosong tidak ditambahkan null karakter :

```
6 3 2 4 1 5
W E A R E D
I S C O V E
R E D F L E
E A T O N C
E
```

Cipherteks yang dihasilkan akan menjadi :

```
EVLNA CDTES EAROF ODEEC WIREE
```

Untuk mendapatkan kembali plainteks, penerima pesan harus mencari jumlah kolom dengan membagi panjang pesan dengan panjang kunci. Kemudian dia akan dapat menulis kembali pesan dalam kolom-kolom,. Selanjutnya mengurutkan kembali kolom tersebut dengan melihat kata kunci.

2.4 *Double Transposition*

Columnar Transposition tunggal mudah diserang dengan menerka semua panjang kolom yang mungkin, menuliskan kembali pesan ke dalam kolom-kolom (tapi dengan urutan yang salah, karena kunci masih belum diketahui), dan kemudian mencari anagram yang mungkin. Maka, untuk membuat algoritma ini lebih kuat, transposisi ganda sering digunakan. Yaitu *Columnar Transposition* yang diterapkan dua kali. Kunci yang sama atau berbeda dapat digunakan untuk kedua operasi.

Sebagai contoh,kita dapat memakai hasil Columnar Transposition dari upabab 2.3 dan melakukan enkripsi kedua dengan kata kunci yang berbeda. STRIPE, yang memberikan permutasi '564321'

```
5 6 4 2 3 1
E V L N A C
D T E S E A
R O F O D E
E C W I R E
E
```

Seperti sebelumnya, hasil ini akan dibaca berlawanan arah kolom yang memberikan cipherteks

CAEEN SOIAE DRLEF WEDRE EVTOC

2.5 *Matrix Cipher*

Transposisi dengan memakai matriks, lebih memudahkan kita dalam mendefinisikan permutasi. Matriks transposisi untuk tujuan ini didefinisikan sebagai :

Matriks persegi A adalah matriks transposisi jika setiao kolom dan baris dari A hanya mempunyai satu nilai 1; elemen lainnya adalah 0.

contoh :

$$A = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{vmatrix}$$

Matriks identitas merupakan matriks transposisi juga. Tipe matriks ini akan dipakai pada matriks cipher. Kita mengambil formula :

$$V = AX$$

dimana V dan B adalah matriks kolom dan A adalah matriks transposisi, maka

- karena tiap baris dari A hanya mempunyai satu buah nilai 1, maka nilai masukan V berasal dari X ,dan
- karena tiap kolom dari A juga hanya mempunyai satu buah nilai 1, maka nilai masukan V akan berbeda satu sama lain.

Dengan demikian, elemen dari V adalah permutasi dari elemen X .

Sebagai contoh untuk lebih memahami hal ini, misalkan kita mempunyai A dan B dengan

A = matriks transposisi dan

B = matriks kolom

sebagai berikut :

$$A = \begin{vmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

$$B = \begin{vmatrix} 51 \\ 63 \\ 56 \\ 13 \\ 16 \end{vmatrix}$$

dan kita mengambil formula $V = AX$, dengan X adalah B , maka

$$V = \begin{vmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{vmatrix} \begin{vmatrix} 51 \\ 63 \\ 56 \\ 13 \\ 16 \end{vmatrix} = \begin{vmatrix} 63 \\ 16 \\ 51 \\ 56 \\ 13 \end{vmatrix}$$

Kini kita dapat mendefinisikan matriks pada cipher transposisi . Cipher transposisi memiliki blok kunci sepanjang-n memetakan blok plainteks P (sebagai vektor kolom) ke blok cipherteks C (juga sebagai vektor kolom) dengan transformasi :

$$C = AP$$

Sebagai contoh, misalkan kita mempunyai pesan TRANSFORMASI dengan menggunakan matriks transposisi :

$$\begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

$$A = \begin{vmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

dan index alphabet normal. Pertama, kita kelompokkan plainteks menjadi blok sepanjang 5

TRANS FORMA SIXXX

dan apabila diperlukan, kita dapat menambahkan karakter X sebagai dummy. Blok diatas akan berkoresponden dengan :

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

19 17 0 13 18 5 14 17 12 0 18 8 23 23 23

Jika kita memikirkan bahwa blok-blok diatas menjadi vektor kolom, kita akan dapat memperoleh blok cipherteks dengan melakukan operasi formulasi, yang telah didefinisikan sebelumnya, A dengan setiap vektor kolom plainteks. Sehingga, kita akan mendapatkan setiap blok cipherteksnya untuk blok pertama, blok kedua, dan blok ketiga:

$$\begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} 19 \\ 17 \\ 0 \\ 13 \\ 18 \end{vmatrix} = \begin{vmatrix} 0 \\ 18 \\ 13 \\ 17 \\ 19 \end{vmatrix}$$

$$\begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} 5 \\ 14 \\ 17 \\ 12 \\ 0 \end{vmatrix} = \begin{vmatrix} 17 \\ 0 \\ 12 \\ 14 \\ 5 \end{vmatrix}$$

$$\begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} 18 \\ 8 \\ 23 \\ 23 \\ 23 \end{vmatrix} = \begin{vmatrix} 23 \\ 23 \\ 23 \\ 8 \\ 18 \end{vmatrix}$$

Dengan demikian , cipherteks yang dihasilkan adalah :

0 18 13 17 19 17 0 12 14 5 23 23 23 8 18

ASNRT RAMOF XXXIS

Tentu saja kita harus dapat meyakinkan bahwa transposisi ini bersifat reversible. Sekilas memang tampak rumit, namun yang kita perlukan hanya matriks invers dari A. dengan formulasi dekripsi

$$P = A'C$$

Invers dari Matriks transposisi ini dapat dengan mudah diperoleh dengan menggunakan *eliminasi Gauss-Jordan*. Mengingat bahwa matriks transposisi yang dipakai elemen baris dan kolomnya hanya memiliki satu buah nilai 1,maka matriks ini akan dapat dengan mudah direduksi dengan melakukan pertukaran baris. Oleh karena itu, *A'* selalu ada untuk setiap matriks transposisi *A*.

Sebagai contoh, kita akan mencoba mendekripsikan pesan dari contoh sebelumnya. Matriks tranposisi A didefinisikan sebagai :

$$A = \begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Untuk mencari inverse dari A kita membuat matriks augmentasi dari matriks A menjadi A|I

$$\left|\begin{array}{ccccc:ccccc} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right|$$

kemudian dengan menukar baris-barisnya , kita akan mendapatkan matriks identitas di sisi kiri

$$\left|\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right|$$

maka matriks A' yang kita cari adalah :

$$A' = \left|\begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{array}\right|$$

Kemudian kita akan memakai matriks A' ini untuk mendekripsikan kembali plainteks.

- blok pertama

$$\left|\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right| \left|\begin{array}{c} 0 \\ 18 \\ 13 \\ 17 \\ 19 \end{array}\right| = \left|\begin{array}{c} 19 \\ 17 \\ 0 \\ 13 \\ 18 \end{array}\right|$$

- blok kedua

$$\left|\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right| \left|\begin{array}{c} 17 \\ 0 \\ 12 \\ 14 \\ 5 \end{array}\right| = \left|\begin{array}{c} 5 \\ 14 \\ 17 \\ 12 \\ 0 \end{array}\right|$$

- blok ketiga

$$\left|\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right| \left|\begin{array}{c} 23 \\ 23 \\ 23 \\ 8 \\ 18 \end{array}\right| = \left|\begin{array}{c} 18 \\ 8 \\ 23 \\ 23 \\ 23 \end{array}\right|$$

Produk operasi ini akan menghasilkan blok plainteks

19 17 0 13 18	5 14 17 12 0	18 8 23 23 23
TRANS	FORMA	SIXXX

3. Transposisi Rekursif

3.1 *Overview*

Transposisi rekursif yaitu tranposisi penukaran (swapping) secara terus menerus sampai basis tertentu (dalam hal ini basis=1) pada blok karakter dengan panjang tertentu. Panjang blok untuk setiap operasi akan berbeda-beda bergantung pada panjang kunci dan plainteks yang akan dienkripsi.

Pada algoritma transposisi sebelumnya, transposisi beroperasi pada alphabet/ indeks alphabet, sedangkan transposisi rekursif ini beroperasi pada konkatenasi string dari semua kode ASCII karakter pesan. Algoritma ini juga beroperasi pada blok-blok pesan namun secara mengalir. Maksudnya, bahwa pada sebagian elemen pada blok ke-n akan dioperasikan kembali pada blok ke- (n+1) .

Transposisi rekursif ini juga menggunakan *key streaming*, yaitu kunci yang dipakai pada operasi blok-blok ke-(n+1) dan berikutnya, berbeda satu sama lain. *Streaming* ini dilakukan dengan melakukan operasi terhadap kunci ke n-1 dengan nilai integer dari karakter pertama pada blok ke-n-1. *Streaming* akan berhenti ketika panjang kunci yang dihasilkan lebih panjang dari blok pesan yang belum ditransposisikan. Kemudian, transposisi dilanjutkan dengan menggunakan kunci-kunci sebelumnya sampai blok pesan terakhir.

Karena algoritma ini melakukan transposisi terhadap konkatenasi string kode ASCII, maka untuk mengembalikan kembali kita harus mengetahui panjang kode ASCII sebelumnya. Oleh karena itu, panjang dari kode ASCII juga ditulis ke dalam cipherteks sebagai header.

Algoritma transposisi ini akan dijelaskan lebih mendalam pada upabab **3.3**.

3.2 Lingkungan Pengembangan Aplikasi

Program aplikasi algoritma transposisi ini dikembangkan pada lingkungan sebagai berikut:

- Intel(R) Pentium(R) 4 2,66 GHz Core Speed 2660.8 MHz Bus Speed 532.2 MHz
- Data Cache

- L1 8-ways 16 KB
- L2 8-ways 1024 KB

- Memory Visipro DDR-SDRAM 512 MB 166.3 MHz
- Nvidia Ge-Force FX5200 128 MB
- Windows XP SP2 with Framework 2.0
- Kakas pemrograman Microsoft Visual Studio .NET berbasis C# language

3.3 Algoritma umum

3.3.1 Pemrosesan Kunci

Kunci yang dimasukkan oleh pemakai berupa karakter dengan kode ASCII 1-255. Kunci yang dimasukkan pemakai merupakan generator untuk memperoleh kunci awal untuk melakukan proses enkripsi dan dekripsi.

Secara umum, algoritma dari pemrosesan kunci adalah sebagai berikut :

```
function  GetKeyAfterEncrypt(
              keyWord:String)
{ Men-generate kunci proses dari
keyWord(kunci dari user)
} → integer

Deklarasi
Q : integer = keyWord.Length;
ki: integer = 0;
ka: integer = keyWord.Length - 1;

Algoritma
while (ki < ka) do
{
   Q += ((int)keyWord[ki] –
            (int)keyWord[ka]);
   ki++;
   ka--;
}

if (Q == 0 || Q==1) then
   Q = keyWord.Length;
else
   Q = abs(Q);
{end while}


return
(GetSumValASCII(keyWord)%keyWord.
Length+Q));
```

```
function  GetSumValASCII(
              keyWord:String)
{ Menjumlahkan nilai kode ASCII
dari karakter pada kata kunci
keyWord
} → integer

Deklarasi
m: integer

Algoritma
for(i:=0;i< keyWord.Length;i++)
      m += (byte) keyWord[i];


return m;
```

Pada awal proses, kita mempunyai kunci berupa string `keyWord`. Kemudian kita inisiasi `Q`, kunci yang dipakai untuk proses selanjutnya, dengan nilai `keyWord.Length`; `ki` – index iterasi kiri – dengan nilai 0; `ka` – index iterasi kanan – dengan nilai `keyWord.Length`.

Pada kalang iterasi, kita ambil karakter pertama dan karakter ke `keyWord.Length`, lalu kita ambil selisih dari nilai ASCII-nya, assign ke variabel `Q`. Kemudian kita lanjutkan untuk karakter ke `(ki+1)` dan `(ka-1)` dan seterusnya, sampai semua karakter telah selesai diproses.

Untuk memahami lebih dalam, misalkan kita mempunyai kata kunci `MAKALAH` (`keyWord`) dengan panjang karakter = 7. Maka nilai Q pada awal proses adalah 7. Kemudian kita iterasi dari setiap karakternya

```
M A K A L A H
ki         ka
   77 – 72
```

Kita cari selisih dari `keyWord[ki]` dengan `keyWord[ka]` kemudian assign ke variabel Q. Dalam hal ini

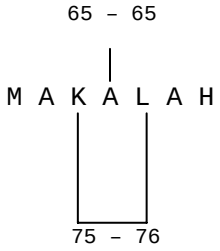
`keyWord[ki] = M`

`keyWord[ka] = H`

selisih nilai ASCII diantaranya adalah 77- 72 = 5, sehingga nilai Q saat ini adalah Q = 7 + 5 = 12. Kemudian `ka--` dan `ki++` menjadi

```
M A K A L A H
  ki     ka
  65 - 65
```

proses dilakukan dengan cara yang sama hingga posisi `ki > ka`.



Nilai akhir dari Q adalah positif. Apabila nilai Q nol atau satu maka nilai Q akan diassign menjadi `Q = keyWord.Length ,` karena apabila nilai kunci nol atau satu maka transposisi tidak akan dapat berjalan.

Pengabsolutan nilai Q di akhir dengan tujuan untuk menghindarkan dengan palindrom dari kata kunci akan menghasilkan nilai kunci yang sama. Apabila hal ini terjadi maka akan terjadi duplikasi kunci dalam pengenkripsian.

Setelah itu kita mencari nilai penjumlahan kode ASCII dari `keyWord`, yaitu

`M + A + K + A + L + A + H`

77+ 65+75+65+76+65+72 = 495

Hasil dari penjumlahan ini kita modulo dengan panjang kunci dan ditambahkan dengan nilai Q sebelumnya, menjadi $Q = 495 \bmod 7 + 11 = 16$

3.3.2 Proses Enkripsi

Enkripsi dilakukan dalam operasi blok karakter. Ukuran blok ini berbeda-beda untuk blok satu dengan lainnya, bergantung pada panjang kunci yang dihasilkan selama proses. Namun pemakaian kunci yang sama sangat memungkinkan terjadi. Pemakaian kunci ini mengikuti grafik fungsi kuadrat dengan gradien positif.

Sebelum memasuki skema algoritma, ada beberapa *class* yang akan saya jelaskan :

- *Class* `CodeProcessing`, yaitu *class* yang menangani proses enkripsi dan dekripsi. Pada *class* ini, terdapat beberapa fungsi statik yang dapat membantu dalam mengkonversi antar tipe.
- *Class* `KeyProcessing`, yaitu *class* yang menangani proses generate awal kunci . Pada *class* ini hanya ada satu *method*

Berikut adalah implementasi dari beberapa fungsi statik dari `CodeProcessing` yang dipakai pada proses enkripsi

```
typedef struct CodeProcessing
{

Merupakan class yang digunakan
untuk menangani proses enkripsi
dan dekripsi. Beberapa fungsi
statiknya dipakai oleh beberapa
class yang lainnya.
}
```

```
function GetLengthBytePlainCode(
code: array of byte) → array of
integer
{ Mencari panjang dari kode ASCII
tiap karakter pada pesan
}

Deklarasi
lengthPerCode : array of integer
[0..(code.Length)];

Algoritma

for (int i:=0;i<code.Length;i++)
{
  lengthPerCode[i] =
       Length (ToString(code[i]));
}

return lengthPerCode;
```

```
function PowInteger (
number: integer, heap : integer)

{Memangkatkan bilangan

} → integer

Algoritma

if (heap == 0) then
    return 1;
else
{

    int m = number;
    for (int i = 1; i < heap; i++)
        m = m * number;
    return m;
}
```

```
function ToByteArray (strCode :
string)
{ Mengubah string menjadi array
of byte[0..stringLength]. Fungsi
ini diimplementasikan dengan
menggunakan fungsi bawaan

} → array of byte
```

```
function ToString(code :array of
byte)
{Mengkonkatenasi elemen array of
byte menjadi string
} → string

Deklarasi
pesan : string = "";
k : integer =0;

Algoritma

for (int i := 0; i < code.Length;
      i++)
{
    for (int j = 0;
    j < code[i].ToString().Length;
    j++)
     {
        pesan +=
               ((string) code[i]).
                 Substring(j, 1);
        k++;
     }
}
return pesan;
```

```
function ToInteger (key: array of
char)
{ Mengambil nilai integer dari
array of char
} → integer

Deklarasi
  number : integer=0;
  pangkat: integer = key.Length-1;

Algoritma

for(int i=0; i<key.Length;i++)
{
    number += ToInteger(key[i])*
        PowInteger(10, pangkat);
   pangkat--;
}

return (number);
```

```
function ToInteger(c : char)
{ Mengambil nilai integer dari
char c
} → integer

Algoritma

    return (c-48);
```

Fungsi statik diatas diimplementasikan menjadi 2 jenis, yaitu yang melibatkan proses interface dan tanpa melibatkan interface. Sedangkan notasi algoritmik diatas adalah yang tidak melibatkan proses interface

Implementasi fungsi `ToString()` saya overload menjadi 2 jenis, yaitu masukan array of byte dan array of integer. Sedangkan fungsi `ToInteger()` saya overload menjadi :

- `ToInteger()` ← array of char, idxAwal (integer), idxAkhir (integer)

Fungsi overload `ToInteger()` ini untuk mencari nilai integer dari array of char dari indeks tertentu saja.

Berikut ini adalah skema utama dari keseluruhan proses enkripsi.

```
procedure EncryptMessage(
              message : string,
              key : integer)
{ Melakukan proses enkripsi pesan

}

Deklarasi
  charCode : array of char of
               message;
  lenCode  : array of char of
               length ASCII code
  handler  : CodeProcessing

Algoritma
if (key >= charCode.Length) then
    writeln("Kunci tidak
              memenuhi syarat");
else
{

charCode =
CodeProcessing.ToString(
CodeProcessing.ToByteArray(
message)).ToCharArray();
```

```
lenCode =
CodeProcessing.ToString(
CodeProcessing.GetLengthBytePlain
Code( CodeProcessing.ToByteArray(
message)).ToCharArray();

  handler.EncryptCode(charCode,key
);

  handler.EncryptCode(lenCode,
key);
}
```

Pertama, proses utama ini akan mengecek apakah nilai kunci valid, yaitu nilai kunci > 1, kemudian jika valid proses dilanjutkan dengan mencari kode ASCII dari setiap karakter pada pesan dan merubahnya menjadi suatu deretan karakter-karakter. Misalkan kita mempunyai pesan:

LUMPUR PORONG

kode ASCII yang bersesuaian adalah

76 85 77 80 85 82 32 80 79 82 79 78 71

(termasuk karakter spasi) maka kode-kode ASCII yang bersesuaian ini dikonkatenasi menjadi suatu string

"7685778085823280798279787 1"

string inilah yang selanjutnya akan ditransposisikan

Kemudian langkah selanjutnya adalah mencatat panjang dari setiap kode ASCII karakter pada pesan. Pencatatan ini dilakukan untuk acuan dalam mengkodekan kembali pesan menjadi plainteks. Dari contoh pesan LUMPUR PORONG dengan kode ASCII

76 85 77 80 85 82 32 80 79 82 79 78 71

maka nilai yang disimpan adalah

2 2 2 2 2 2 2 2 2 2 2 2 2

nilai ini juga akan ditransposisikan dan ditulis ke dalam cipherteks sebagai header. Penulisan dilakukan sebagai byte dengan panjang dua dan satu. Penulisan byte dengan panjang satu hanya dioperasikan pada kode sisa. Kemudian proses dilanjutkan dengan mengenkripsi pesan dan panjang byte tadi.

Skema dari algoritma enkripsi adalah sebagai berikut :

```
procedure SwapCodeDeeply (
arrCode: array of char,
idxAwal: integer,
idxAkhir: integer,
length: integer
)
{
Melakukan pertukaran karakter-
karakter blok pesan
}

Deklarasi
aw : integer idxAwal;
ak : integer idxAkhir;
copy : char;

Algoritma
aw = idxAwal;
ak = idxAkhir;

while (aw < ak) do
{
   copy = arrCode[aw];
   arrCode[aw] = arrCode[ak];
   arrCode[ak] = copy;
   aw++;
   ak--;
}
{end while}
```

```
procedure SwapCodeRecursively (
arrCode : array of char,
idxAwal : integer
key     : integer)
{ Melakukan transposisi dari
level luar dari blok pesan sampai
level terdalam

}

Deklarasi
idxAkhir: integer
length : integer = key;
i : integer = idxAwal;

Algoritma
idxAkhir = idxAwal + key - 1;
length = key;
i = idxAwal;
while (i < idxAkhir)
{
  SwapCodeDeeply(
       arrCode, i,idxAkhir,
       idxAkhir - idxAwal + 1);
  idxAkhir--;
  i++;
}
```

```
procedure EncryptCode (
arrCode : array of char,
key     : integer)
{ Melakukan proses transposisi
rekursif pada pesan }

Deklarasi
index : integer= key mod 2;
newKey: integer= key;
idxKeySimpan : integer= 0;
idxKeyAmbil  : integer= 0;
arrKey : array of integer[0];

Algoritma

if (key <= arrCode.Length) then
{
  while (idxKeyAmbil >= -1) do
  {
    if (newKey<=
        arrCode.Length - index)
    then
    {
        Array.Resize(ref arrKey,
        arrKey.Length + 1);

        arrKey[idxKeySimpan]=
        newKey;

        SwapCodeRecursively(
        arrCode, index, newKey);

        newKey +=
        Math.Abs(newKey –
        ToInteger(arrCode[index]
        ));

        index +=
          arrKey[idxKeySimpan]/2;
        idxKeySimpan++;
        idxKeyAmbil=
        idxKeySimpan - 1;
    }
    else
    {
        if (idxKeyAmbil >= 0) then
            newKey=
             arrKey[idxKeyAmbil];
        else
            break;
        idxKeyAmbil--;
  }
   {end if}
 }
 {end while}
}
```

Penjelasan dari algoritma proses enkripsi diatas adalah sebagai berikut :

1. Inisiasi awal sebelum proses enkripsi berjalan, kita tentukan dulu index awal kode yang akan dienkripsi. Nilai ini didapat dengan memodulo panjang kunci dengan 2. Nilai ini selalu bernilai 0 atau 1. Kita definisikan array of integer untuk menyimpan kunci, karena apabila pada generasi kunci didapatkan lebih panjang dari blok pesan sisa, maka proses akan memakai kunci sebelumnya hingga tidak ada lagi kunci yang sesuai. Variabel `idxKeySimpan` sebagai pointer index penyimpanan kunci dan `idxKeyAmbil` sebagai pointer index pengambilan kunci
2. Kemudian, kita cek apakah panjang kunci yang dipakai lebih pendek dari panjang pesan, jika memenuhi maka pesan akan dienkripsi. Pengecekan ini dilakukan untuk menangani proses pengenkripsian panjang kode ASCII (header cipherteks).
3. Kita iterasi arrray of char dari pesan, hingga tidak ada kunci yang sesuai.
   a. Jika panjang kunci masih kurang dari blok pesan sisa, maka kita simpan kunci tersebut ke dalam array ,dan melakukan operasi `SwapCodeRecusively` dari index sepanjang kunci. Kemudian kita generate kunci berikutnya dengan formulasi kunci yang lama dikurangi dengan nilai integer pertama dari blok pesan. Setelah itu kita naikkan index sebanyak setengah panjang kunci, `idxKeySimpan` kita naikkan,dan `idxKeyAmbil` kita assign menjadi `idxKeySimpan-1`.
   b. Jika panjang kunci lebih besar dari panjang blok pesan, kita ambil kunci yang dipakai sebelumnya dari array. Apabila index pengambilan kunci telah pada ujung pertama maka iterasi berhenti.

Sebagai contoh misalkan kita mempunyai kata kunci

MAKALAH

pesan

LUMPUR PORONG

dari contoh sebelumnya kita dapatkan kunci untuk proses adalah 16, dan hasil string konkatenasi kode ASCII dari pesan adalah

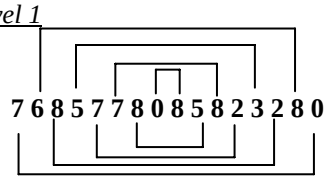
768577808582323280798279 7871

Kita tentukan index awal untuk enkripsi adalah $16 \bmod 2 = 0$. Kita ambil blok dari sring kode sepanjang kunci=16 dari index=0

76857780858232807982797871
k= 16

Kemudian kita pertukarkan karakter-karakter dari blok pesan tadi hingga level paling dalam

- *level 1*

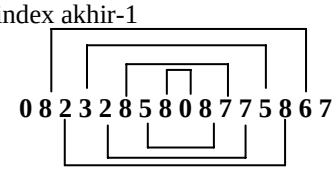


menghasilkan

0 8 2 3 2 8 5 8 0 8 7 7 5 8 6 7

- *level 2*

Untuk level 2 kita mulai dari index awal+1 dan index akhir-1



menghasilkan

0 6 8 5 7 7 8 0 8 5 8 2 3 2 8 7

Untuk level selanjutnya menghasilkan

0 6 2 3 2 8 5 8 0 8 7 7 5 8 8 7 → level 3

0 6 2 5 7 7 8 0 8 5 8 2 3 8 8 7 → level 4

0 6 2 5 2 8 5 8 0 8 7 7 3 8 8 7 → level 5

0 6 2 5 2 7 8 0 8 5 8 7 3 8 8 7 → level 6

0 6 2 5 2 7 8 0 8 5 8 7 3 8 8 7 → level 7

0 6 2 5 2 7 5 8 0 8 8 7 3 8 8 7 → level 8

0 6 2 5 2 7 5 0 8 8 8 7 3 8 8 7 → level 9

Sehingga barisan string kode menjadi

06252750888738877982797871

Setelah itu kita enkripsi blok berikutnya dengan panjang kunci berikutnya = 16+ 0=16, dimana 0 adalah nilai awal dari blok enkripsi yang pertama. Index awal pengenkripsian blok berikutnya = 0+(16 div 2) = 8.

06252750888738877982797871
k= 16

Demikian seterusnya sampai pada blok yang terakhir. Apabila kunci k lebih panjang dari blok sisa, maka proses akan memakai kunci sebelumnya.

Setelah pengenkripsian kode selesai, maka, kode-kode tersebut dikembalikan lagi menjadi bytes untuk dituliskan sebagai cipherteks.

Panjang kode byte untuk cipherteks sama dengan panjang kode byte pada plainteks. Sebagai contoh kita ambil dari string kode yang telah dienkripsi sampai selesai (hasil digenerate dari program) :

062527508897289778837878 71

dengan panjang kode sebelumnya :

2 2 2 2 2 2 2 2 2 2 2 2 2

maka string diatas kita pilah sepanjang deretan panjang kode diatas

06 25 27 50 88 97 28 97 78 83 78 78 71

2 2 2 2 2 2 2 2 2 2 2 2 2

kemudian kita kodekan kembali menjadi karakter sehingga hasil yang diperoleh adalah :

−├←2Xa aNSNNG

keseluruhan cipherteks yang dihasilkan (termasuk pengenkripsian panjang kode dan penulisannya) adalah

┬┬┬┬┬┬┐ −├←2Xa aNSNNG

pemisah antara kode panjang dengan kode pesan adalah NULL .

Pada penulisan cipherteks ini, apabila kita menemukan blok kode ASCII < 10, maka penulisan dilakukan dengan menuliskan ASCII 0 dan bilangannya (misal 07 ditulis 0 dan 7). Hal ini untuk memudahkan dalam pembacaan ketika melakukan proses dekripsi. Sebelum melakukan teknik ini, penulis telah mencoba beberapa cara lain namun masih terdapat kebocoran algoritma. Sehingga penulis berpendapat bahwa teknik ini adalah yang terbaik.

Apabila pada proses penulisan cipherteks kita menemukan blok kode ASCII > 255 , misal 754, maka penulisan kode dibagi menjadi 2 blok, blok dengan panjang 2 dan blok dengan panjang 1. Misal 754, kita tuliskan sebagai 75 dan 4 . Penulisan panjang kode tetap 3.

Berikut adalah statistik dari beberapa berkas teks dengan ukuran berbeda dengan kunci MAKALAH

Nama berkas	Ukuran	Waktu proses
source-1.txt	500 bytes	1 detik
source-2.txt	5000 bytes	6 detik
source-3.txt	10.000 bytes	13 detik
source-4.txt	15.000 bytes	22 detik

Berdasarkan data statistik diatas bahwa semakin panjang file yang dibaca dengan kunci yang sama maka proses enkripsi menjadi semakin lama. Namun, ukuran dalam bytes disini bukan panjang kode sebenarnya yang diproses, karena bergantung pada panjang kode ASCII tiap karakter didalamnya. Sehingga dengan ukuran file yang sama tapi isi berbeda, waktu pemrosesan akan berbeda pula.

3.3.3 Proses Dekripsi

Proses dekripsi merupakan kebalikan dari fungsi enkripsi. Pemrosesan kata kunci juga sama dengan proses enkripsi. Namun, pada pendekripsian pesan, kita mencari dulu semua kunci yang dipakai, karena fungsi dekripsi akan dilakukan dengan urutan terbalik.

Sebelum masuk ke dalam pendekripsian pesan, kita harus memilahkan terlebih dahulu antara kode pesan sebenarnya dengan kode panjang ASCII, mengingat bahwa pada cipherteks terdapat header – panjang kode – yang ditulis.

Setelah pemilahan selesai, yang kita lakukan pertama kali adalah mendekripsi panjang kode – header – untuk mendapatkan kembali panjang kode sebenarnya dari plainteks. Kemudian, kita dekripsikan pesan yang sebenarnya dan menulis hasilnya ke file.

Skema utama dari dekripsi adalah sebagai berikut :

```
procedure DecryptMessage (
byteCode: array of byte,
key: integer)
{ Melakukan proses dekripsi pesan
}

Deklarasi
charCode : array of char;
lenCode  : array of char;
idxPesan : integer =0 ;
handler  : CodeProcessing;
```

```
Algoritma

lenCode = CodeProcessing.
GetLengthByteCipherCode(byteCode,
ref idxPesan).ToCharArray();

charCode = CodeProcessing.
GetCharByteCipherCode(byteCode
,idxPesan);

if(CodeProcessing.GetArrayOfKey
(lenCode, key).Length != 0)
{
    handler.DecryptCode(
    CodeProcessing.
        GetArrayOfKey(lenCode,key),
    lenCode);
}

handler.DecryptCode(
    CodeProcessing.
        GetArrayOfKey(
                charCode,key),
    charCode);

handler.SavePlainteks(
charCode,
CodeProcessing.
 GetArrayOfLengthInt(lenCode),
savePath);
```

Penjelasan dari skema utama dekripsi :

1. Inisiasi awal kita membuat 2 array of char, yaitu charCode untuk menyimpan kode pesan dan lenCode untuk menyimpan panjang kode-kode ASCII serta idxPesan sebagai indexing posisi pembacaan terakhir dalam memilahkan code untuk mendapatkan karakter pertama kode pesan
2. Kemudian kita cek terlebih dahulu apakah panjang kode-kode ASCII pesan terenkripsi atau tidak. Pengecekan ini dilakukan dengan melihat panjang dari array kunci-kunci yang dipakai dalam dekripsi nantinya. Jika kosong, maka panjang kode-kode tersebut tidak di enkripsi. Sebaliknya, kita dekripsikan panjang kode-kode tadi.
3. Lalu langkah selanjutnya yaitu mendekripsikan kode pesan yang sebenarnya. Array kunci ditemukan dengan memakai fungsi statik dari class `CodeProcessing`, yaitu `GetArrayOfKey`

4. Langkah terakhir adalah menuliskannya ke file luar kode pesan sebenarnya yang telah didekripsikan kembali.

Selanjutnya, mari kita lihat lebih dalam skema dari algoritma dekripsi (DecryptCode). Skema dari algoritma ini lebih sederhana dari skema enkripsi dan juga ada beberapa fungsi pada enkripsi dipakai kembali :

```
procedure DecryptCode (
arrKey   : array of key (integer),
charKey  : array of char,
key      : integer)
{ Melakukan proses dekripsi pesan
}

Deklarasi
charCode : array of char;
lenCode  : array of char;
idxPesan : integer =0 ;
handler  : CodeProcessing;
i        : integer
Algoritma
idxAwal = GetIndexAwal(arrKey);
for (i= arrKey.Length-1;
     i >= 0;i--)
{
  SwapCodeReverseRecursively(
       charKey, idxAwal,
       idxAwal + (arrKey[i] - 1),
       arrKey[i]);

  if (i > 0)
     idxAwal =
       Math.Abs(idxAwal-(
                 arrKey[i-1]/2));
  else
     idxAwal = 0;
}
```

```
function GetIndexAwal(
{Mencari index awal mulai
operasi transposisi untuk
dekripsi
}
arrKey :array of key (integer)
){
Algoritma
idx: integer= arrKey[0] mod 2;
i  : integer=0;
while(i< arrKey.Length-1) do
{
   idx += arrKey[i]/2;
   i++;
}
return index;
}
```

```
procedure
SwapCodeReverseRecursively (
arrCode  : array of char,
idxAwal  : integer
idxAkhir : integer
length   : integer)

{ Melakukan transposisi dari
  level terdalam dari blok pesan
  sampai level terluar
}

Deklarasi
aw   : integer;
akh  : integer;

Algoritma
aw = (idxAwal+length/2) - 1;
akh= (idxAkhir-length/2) + 1;
while (aw >= idxAwal)
{
    SwapCodeDeeply(arrCode, aw,
                   akh, length);
    aw--;
    akh++;
}
```

Penjelasan dari algoritma dekripsi adalah sebagai berikut :

1. Inisiasi awal, kita definisikan 2 array of char, charCode dan lenCode masing-masing untuk menyimpan kode pesan dan kode panjang . Kemudian kita cari index awal operasi transposisi. Index awal ini dicari berdasarkan array kunci yang telah diperoleh, yaitu dengan formulasi pada enkripsi dimana Q += key/2, idx dicari dengan formulasi itu juga.
2. Setelah kita mendapatkan index awal transposisi, array kunci, selanjutnya kita traversal pesan secara menurun untuk ditransposisikan balik.
3. Untuk skema algoritma **SwapCodeReverse Recursively (),** transposisi dilakukan dari level terdalam dari pesan, misalkan dari contoh enkripsi kita mempunyai kode

   0 6 2 5 2 7 5 8 0 8 8 7 3 8 8 7

   maka transposisi dimulai pada kode 808, yaitu level ke-9, untuk selanjutnya kode menjadi

   0 6 2 5 2 7 5 0 8 8 8 7 3 8 8 7 → level 9

0 6 2 5 2 7 5 8 0 8 8 7 3 8 8 7 → level 8

0 6 2 5 2 7 8 0 8 5 8 7 3 8 8 7 → level 7

0 6 2 5 2 7 8 0 8 5 8 7 3 8 8 7 → level 6

0 6 2 5 2 8 5 8 0 8 7 7 3 8 8 7 → level 5

0 6 2 5 7 7 8 0 8 5 8 2 3 8 8 7 → level 4

0 6 2 3 2 8 5 8 0 8 7 7 5 8 8 7 → level 3

0 6 8 5 7 7 8 0 8 5 8 2 3 2 8 7 → level 2

7 6 8 5 7 7 8 0 8 5 8 2 3 2 8 0 → level 1

4. Kemudian indexAwal mundur dengan formulasi $abs(idxAwal - arrKey[i])$, jika index masih > 0, pema kaian absolut karena yang kita cari hanya selisihnya.

Berikut adalah statistik dari contoh beberapa berkas teks hasil enkripsi dengan kunci MAKALAH

Nama berkas	Ukuran	Waktu proses
cipherteks-1.txt	891 bytes	1 detik
cipherteks-2.txt	8697 bytes	5 detik
cipherteks-3.txt	17.244 bytes	8 detik
cipherteks-4.txt	26.135 bytes	10 detik

Ukuran dari cipherteks ini lebih besar daripada plainteksnya, karena kita juga menyimpan panjang kode plainteks sebagai header. Rasio peningkatannya adalah 17.82% , 17.39%, 17.24%, 17.42%, jika dirata-ratakan menjadi 17.48% . Meskipun ukuran lebih besar daripada plainteks sebelumnya, namun waktu proses dekripsi lebih cepat daripada enkripsi, hal ini karena pada dekripsi kita tidak perlu melakukan pengecekan tentang kode ASCII ketika menulis ke plainteks, karena selalu benar. Dan juga kita tidak perlu menuliskan panjang kode ASCII tidak seperti pada proses enkripsi.

3.4 Kekuatan algoritma

Kekuatan algoritma ini terletak pada penciptaan kunci dan *keystreaming* . Penciptaan kunci pertama kali dengan memanfaatkan kode ASCII dan panjang kunci dapat menjamin keunikan dari kunci. Meskipun dapat dipecahkan dengan *brute force*, namun akan memerlukan banyak percobaan dan waktu. Semakin panjang kunci, maka keamanan kunci semakin meningkat.

Kita ambil contoh apabila seseorang mencoba memecahkan kunci dari cipherteks dengan panjang kunci=6 dengan usaha pertama adalah panjang kunci =2. Maka untuk mencoba semua kemungkinan (tanpa karakter NULL), terdapat $2.(255^2)+3.(255^3)+4.(255^4)+5.(255^5)+1 =$ 130.050+49.744.125+16.913.002.500+5.391.019 .546.875+1 = 5.407.982.423.551 kali percobaan dengan kemungkinan kata kunci terbaik. Sedangkan untuk panjang kunci n, maka diperlukan percobaan sebanyak $n.\sum_{m=2}^{n} 255^m$ kali.

Kekuatan lainnya adalah pemakaian operasi blok aliran, maka cipherteks tidak akan dapat ditebak hanya dengan mengetahui sebagian plainteksnya. Karena sebagian elemen dari blok dipakai kembali untuk dioperasikan pada blok berikutnya. Kecuali dalam kasus-kasus tertentu dimana panjang plainteks yang diketahui sama dengan panjang kunci yang dipakai.

Selain itu algoritma ini juga tidak dapat diserang dengan analisis frekuensi, karena algoritma ini melibatkan pertukaran string ASCII, tidak menggunakan metode substitusi.

3.5 Kelemahan Algoritma

Pada bab 3.4 telah dibahas bahwa kekuatan algoritma ada pada penciptaan kunci dan *keystreaming*. Namun ada beberapa celah dari algoritma tesebut, yaitu ada kemungkinan bahwa dengan kata kunci yang berbeda dapat menghasilkan kunci yang sama. Tetapi hal ini sangat sulit dan jarang terjadi.

Dan juga penyerangan terhadap panjang kunci daripada kata kunci merupakan kelemahan dari algoritma ini. Kriptanalis hanya mencoba memakai panjang kunci 2 s.d. (panjang kode string ASCII pesan). Karena panjang kunci tidak mungkin melebihi panjang kode string ASCII pesan.

Selain itu, kelemahan dari algoritma ini adalah konsumsi waktu yang cukup lama untuk melakukan proses. Hal ini karena algoritma ini memakai proses pertukaran satu-satu dari string ASCII. Dan juga hasil enkripsi, berkas cipherteks, mempunyai ukuran lebih besar daripada berkas plainteksnya.

4. Serangan serangan Kriptanalis terhadap Transposisi Rekursif

4.1 Analisis Frekuensi

Algoritma ini tidak dapat diserang dengan analisis frekuensi. Alasannya adalah algoritma ini tidak menggunakan substitusi alphabet dalam melakukan proses enkripsi. Sehingga karakter yang sering muncul ada pada cipherteks tidak memberikan petunjuk apa-apa.

4.2 Known Plainteks Attack

Kita ambil dari contoh sebelumnya, misalkan dari cipherteks = −├←2 kita mengetahui bahwa plainteksnya = LUMP. Kita ambil kode ASCII dari masing-masing teks :

cipher = **0 6 2 5 2 7 5 8** 0 8 8 7 3 8 8 7 ...

plain = **7 6 8 5 7 7 8 0**

kita coba ambil index awal transposisi adalah 0. Kita tidak mempunyai petunjuk apa-apa dalam mentransposisi. Namun apabila kita melihat pola selanjutnya dari cipherteks, kita cari angka 7 pertama. yaitu pada indeks ke-6. Maka kemungkinan kunci sementara adalah 7. Kita coba transposisikan :

0 6 2 2 5 7

0 5 2 2 6 7

7 6 2 2 5 0

Dari hasil diatas kita mendapatkan ASCII untuk kode pertama yang bersesuaian. Kemudian kita lanjutkan dengan kunci baru = 6+7 =13,dan index awal = 0+6 div 2 = 3. Apabila pada operasi transposisi kedua ini tidak mendapatkan hasil, maka kunci tadi tidak memenuhi. Kita coba :

7 6 **2 2 5 0 5 8 0 8 8 7 3 8 8** 7 ...

2 2 5 0 5 8 0 8 8 7 3 8 8

2 2 5 0 5 8 0 8 8 7 3 8 8

2 2 5 0 8 8 0 8 5 7 3 8 8

2 2 5 7 5 8 0 8 8 0 3 8 8

2 2 3 0 8 8 0 8 5 7 5 8 8

2 8 5 7 5 8 0 8 8 0 3 2 8

8 2 3 0 8 8 0 8 5 7 5 8 2

kita dapatkan plainteks dari hasil transposisi adalah :

7 6 8 2 3 0 8 8 0 8 5 7 5 8 2 7 ..

karena kode kedua tidak bersesuaian, maka kunci= 6 tidak memenuhi. Dan begitu seterusnya kita cek semua kemungkinan kunci dengan mencari angka 7 berikutnya. Hingga pada index =16, kita dapatkan kunci=16 untuk cipherteks diatas. Dan apabila ditransposisikan akan didapatkan keseluruhan plainteks.

Tapi, bagaimana jika kriptanalis hanya mengetahui plainteks yang bersesuaian dengan cipherteks pada bagian tengah. Metode seperti diatas tidak akan dapat berjalan. Karena kita tidak akan tahu apakah kode pertama dari plainteks merupakan kode ke-n dari kode pertama di cipherteks ataukah kode sebelumnya. Dan kita tidak tahu indeks awal transposisi.

5. Kesimpulan dan Saran

Berdasarkan pembahasan diatas, maka kesimpulan yang dapat diambil :

1. Variasi dari algoritma transposisi sangat banyak. Dan terapannya dapat bermacam-macam.
2. Algoritma transposisi rekursif mempunyai beberapa kekuatan dan juga kelemahan. Algoritma ini akan lebih kuat lagi apabila pada akhir proses enkripsi cipherteks di-XOR-kan dengan kata kunci, sehingga penebakan terhadap kunci sulit untuk dilakukan.
3. Algoritma transposisi jika diterapkan secara stand alone, akan terdapat celah untuk mendapatkan kunci enkripsi dan dekripsi.
4. Tiap proses penukaran dan transposisi membutuhkan cost waktu yang cukup besar.
5. Algoritma pencarian kunci pada algoritma diatas terkadang dapat menghasilkan nilai yang sama pada beberapa kasus. Namun peluang terjadinya hal ini sangat kecil sekali.

Saran dari penulis adalah :

1. Gunakan algoritma ini hanya untuk mengenkripsi berkas-berkas dengan ukuran relatif kecil dan tidak membutuhkan tingkat keamanan cukup tinggi. Karena keamanan dari algoritma ini masih belum diuji secara tepat oleh para ahli kriptografer dan kriptanalis profesional.
2. Untuk mengembangkan algoritma transposisi atau lainnya, usahakan untuk dapat memikirkan *time-cost* dan *algorithm-cost*

DAFTAR PUSTAKA

[1] Banerjee,Rahul, *Introduction to Symmetric Key Cryptography* ,CS & IS Group Birla Institute of Technology and Science

[2] Bishop,David, *Introduction to Cryptography with Java Applet*, Jones and Bartlet Computer Science, 2003

[3] Munir, Rinaldi, *Diktat Kuliah IF5054 Kriptografi*, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2006

[4] *http://en.wikipedia.org/Transposition_cipher*

[5] Schneier, Bruce. (1996). Applied Cryptography 2nd. John Wiley & Sons.

Note : Program dapat didownload di
http://students.if.itb.ac.id/~if14063/rex_cipher/