Pengukuran Kekuatan Kunci pada Vigenere Cipher

Rezan Achmad / 13508104[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if18104@students.if.itb.ac.id, rezanachmad@yahoo.co.id

***Abstrak* — *Vigenere cipher* adalah salah satu algoritma kriptografi klasik dan cukup mudah digunakan. Cipher ini merupakan salah satu jenis cipher subtitusi dan hanya mensubtitusi satu huruf plainteks dengan satu huruf kunci. Telah banyak cara untuk menyerang Vigenere cipher hanya melalui ciphertext-nya saja, tanpa memerlukan kunci. Beberapa metode yang telah ada yaitu analisis frekuensi dan metode kasiski. Metode kasiski memanfaatkan kriptogram yang berulang yang terdapat pada ciphertext. Metode kasiski menghasilkan panjang kunci yang mungkin untuk mendekripsi ciphertext. Jika metode kasiski telah diketahui, tentu bisa diketahui cara mengindarinya. Pada makalah ini dipaparkan bagaimana cara mengukur kekuatan kunci yang digunakan untuk mengenkripsi plaintext. Caranya cukup sederhana, plaintext yang telah dienkripsi akan diperiksa kriptogram berulangnya kemudian akan diberitahu apakah kunci termasuk lemah atau kuat. Untuk mensimulasikan hal ini penulis membuat program sederhana yang bernama *Vigenere Meter*. Hasil yang diperoleh setelah mensimulasikan aplikasi ini yaitu kunci yang lebih panjang belum tentu lebih kuat dari pada kunci yang pendek, hal itu tergantung dari pola tulisan yang terdapat plainteks serta karakter kunci yang dipilih. Pola berulang pada plainteks memang sulit dihindari apalagi untuk teks yang panjang dan ini menjadi kelemahan pada *Vigenere cipher*.**

***Kata kunci* — Cipher; Kasiski; Kekuatan Kunci; Kriptogram; Vigenere; Vigenere Meter**

I. PENGANTAR

Vigenere *Cipher* adalah salah satu algoritma klasik yang cukup mudah untuk digunakan. Enkripsi ini merupakan salah satu enkripsi jenis subtitusi dan hanya mensubtitusi satu huruf plainteks dengan satu huruf kunci. Setiap huruf plainteks dienkripsi dengan kunci yang berbeda. Rumus enkripsi Vigenere Chiper yaitu

$$C_i = (P_i + K_i) \% 26$$

sedangkan rumus untuk dekripsi yaitu

$$P_i = (C_i + K_i) \% 26$$

Berikut contoh penggunaan Vigenere Cipher.

Plainteks : Di suatu senja di musim
Key : Pelangi

D	i	s	u	a	t	U		s	e	
p	e	l	a	n	g	I		p	e	
s	m	d	u	n	z	C		h	i	

n	j	a		d	i		m	u	s	i	m
l	a	n		g	o		p	e	l	a	n
y	j	n		j	q		b	y	d	i	z

Chiper : sm dunzc hiykn jq bydiz

Cara untuk dekripsi Vigenere *Cipher* persis sama dengan cara enkripsinya.

Sekarang telah banyak cara untuk mendekripsi chiper Vigenere tanpa menggunakan kunci. Cara ini biasa disebut dengan *chiper attack*. Cara yang paling sering digunakan yaitu metode frekuensi huruf dan metode kasiski.

Yang menarik adalah metode kasiski. Metode ini membantu menemukan panjang kunci Vigenere chiper. Metode kasiski memanfaatkan keuntungan bahwa bahasa Inggris tidak hanya mengandung perulangan huruf tetapi juga perulangan pasangan huruf atau tripel huruf seperti TH, THE dan sebagainya. Perulangan kelompok huruf ini ada kemungkinan menghasilkan kriptogram yang berulang.

Sebagai contoh :
Plainteks : CRYPTO IS SHORT FOR CRYPTOGRAPHY
Kunci : abcdab cd abcda bcd abcdabcdabcd
Chipertesk : **CSASTP** KV SIQUT GQU **CSASTP**IUAQJB

Pada contoh ini, CRYPTO dienkripsi menjadi kriptogram yang sama yaitu CSATP. Secara intuitif jika jarak antara dua buah string yang berulang di dalam plainteks merupakan kelipatan dari panjang kunci maka string yang sama tersebut akan muncul menjadi kriptogram yang sama pula di dalam chiper teks. Pada contoh, jarak antar dua CRYPTO yang berulang adalah 16. Panjang kunci yang mungkin adalah faktor pembagi dari 16 yaitu 16, 8, 4, 2 dan 1.

Secara lengkap langkah-langkah metode kasiski yaitu :

1. Temukan semua kriptogram yang berulang di dalam chiperteks (pesan yang panjang biasanya mengandung kriptogram yang berulang)
2. Hitung jarak antara kriptogram yang berulang

3. Hitung semua faktor (pembagi) dari jarak tersebut (faktor pembagi menyatakan panjang kunci yang mungkin)
4. Tentukan irisan dari himpunan faktor pembagi tersebut. Nilai yang muncul di dalam irisan menyatakan angka yang muncul pada semua faktor pembagi dari jarak-jarak tersebutt. Nilai tersebut mungkin adalah panjang kunci.

Saat melakukan enkripsi, kita dapat memilih panjang kunci agar menyulitkan kriptanalisis untuk melakukan attack dengan metode kasiski. Jika Anda mengetahui metode kasiski, tentu Anda bisa menghindari metode itu ibarat Anda telah tahu bahwa di depan jalan terdapat lubang tentu Anda bisa menghindarinya.

Makalah ini akan membahas bagaimana memilih panjang kunci untuk menyultkan serangan dari kriptanalis dengan metode kasiski. Dalam makalah ini akan dibahas algoritma program kecil yang menentukan apakah kunci yang dipilih termasuk lemah atau kuat.

II. METODE KASISKI

Sebelum merancang algoritm untuk menentukan lemah atau kuatnya suatu kunci, terlebih dahulu harus diketahui secara detail bagaimana cara kerja metode kasiski. Secara lengkap langkah-langkah metode kasiski yaitu :

1. Temukan semua kriptogram yang berulang di dalam chiperteks (pesan yang panjang biasanya mengandung kriptogram yang berulang
2. Hitung jarak antara kriptogram yang berulang
3. Hitung semua faktor (pembagi) dari jarak tersebut (faktor pembagi menyatakan panjang kunci yang mungkin)
4. Tentukan irisan dari himpunan faktor pembagi tersebut. Nilai yang muncul di dalam irisan menyatakan angka yang muncul pada semua faktor pembagi dari jarak-jarak tersebutt. Nilai tersebut mungkin adalah panjang kunci.

Berikut contoh penggunaan metode kasiski. Misalkan *ciphertext*-nya adalah

LJVBQ STNEZ LQMED **LJV**MA MPKAU FAVAT **LJV**DA YYVNF JQLNP **LJV**HK VTRNF **LJV**CM LKETA **LJV**HU YJVSF KRFTT WEFUX VHZNP

1. Temukan kriptogram yang berulang.
   Salah satu kriptogram yang berulang yaitu LJV.
2. Hitung jarak antar kriptogram yang berulang.
   LJV ke – 1 dengan LJV ke - 2 = 15
   LJV ke – 2 dengan LJV ke - 3 = 15
   LJV ke – 3 dengan LJV ke - 4 = 15
   LJV ke – 4 dengan LJV ke - 5 = 10
   LJV ke – 5 dengan LJV ke - 6 = 10
3. Hitung semua fakto pembagi dari jarak tersebut.
   Faktor pembagi 15 = {3, 5, 15}
   Faktor pembagi 10 = {2, 5, 10}
4. Tentukan irisan dari himpunan faktor permbagi tersebut.
   Irisan dari {3, 5, 15} dan {2, 5, 10} adalah {5}.
   Jadi perkiraan panjang kunci adalah 5 karakter.

Setelah panjang kunci ditemukan, langkah berikutnya yaitu mengelompokkan pesan. Satu pesan terdiri dari 5 karakter.

LJVBQ STNEZ LQMED **LJV**MA MPKAU FAVAT **LJV**DA YYVNF JQLNP **LJV**HK VTRNF **LJV**CM LKETA **LJV**HU YJVSF KRFTT WEFUX VHZNP

Kelompok	Pesan	Huruf paling sering muncul
1	LSLLM FLYHL VLLLY KWV	L
2	JTQJP AJYQJ TJKJJ REH	J
3	VNMVK VVVLV RVEVV FFZ	V
4	BEEMA ADNNH NCTHS TUN	N
5	QZDAU TAFPK FMAUF TXP	A

Triplet yang paling sering muncul adalah THE. Menurut tabel diatas, triplet yang sering muncul adalah LJV. LJV disinyalir sebagai THE. Untuk menentukan kunci, dibuat tabel dibawah ini :

Plainteks	**Cipherteks**	**Kunci**
T	L	S
H	J	C
E	V	R
*	N	*
*	A	*

Hasil dekripsi sementara dengan menggunakan kunci diatas yaitu :

THEBQ ARWEZ TOVED THEMA UNTAU NYEAT THEDA
GWENF ROUNP THEHK DRANF THECM TINTA THEHU
GHESF SPOTT ECOUX DFINP

Perhatikan bagian string MA UNTAU N. MA UNTAU N mungkin adalah MO UNTAI N.

Plainteks	**Cipherteks**	**Kunci**
O	A	M
I	U	M
M	M	A
A	A	A

Jadi, kunci yang utuh adalah SCRAM. Dengan demikian plain teks yang sebenarnya adalah

THE BEAR WENT OVER THE MOUNTAIN YEAH THE DOG
WENT ROUND THE HYDRANT THE CAT INTO THE
HIGHEST SPOT HE COULD FIND

III. ALGORITMA PENGUKURAN KEKUATAN KUNCI

A. Algoritma

Sebelumnya telah dijelaskan mengenai langkah-langkah metode kasiski. Pada bagian ini akan dijelaskan bagaimana mencari string yang berulang. Berikut pseudo code untuk mencari string berulang dengan panjang tertentu.

```
array of <string, array of integer>
SearchRedundantString(text : string, length
: int)
  textLength : integer
  bound, i   :  integer
  sub     :  string
  result  :  array of <string, array of
             integer>
  search  :  array of integer

  textLength := text.Length
  bound = textLength – length;


  for (i := 0, i < bound, i := i + 1)
    sub := text.subString(i, length)
    if (not result.ConainsKey(sub))
      search = BMMatch(text.SubString(i),
      sub)
      if (searchResult.Count > 1)
        UpdateSearch(search, i)
        result.Add(<sub, search>)

  return result
```

Fungsi `SearchRedundantString` menerima dua masukan yaitu `text` serta `length`. Fungsi ini akan mencari semua *substring* sepanjang `length` yang terdapat pada `text`. Semua informasi posisi *substring* akan disimpan di `result` yang berupa pasangan `string` dan `array of integer`.. Pencarian posisi-posisi *substring* tertentu menggunakan algoritma Boyer More (`BMMatch`). Hasil dari `BMMatch` akan ditambahkan ke `result` jika jumlah posisi *substring* yang ditemukan lebih dari satu. Sebelum ditambahkan ke `result`, posisi – posisi pada `search` harus disesuaikan dulu agar posisinya relatif terhadap `text`.

Langkah selanjutnya yaitu mencari *substring* mana yang paling banyak terdapat pada *text*. Sebenarnya, *substring* yang lain bisa digunakan tetapi hanya diambil satu untuk mewakili semua *substring*.

```
string MaxOccurence(result : array of
<string, array of integer>)
  substring : string
  max : int
  count : int

  substring := ””
  max := 0;
  foreach (result as val)
    count := val.Second.Count

    if (max < count)
      max := count;
      substring := val.First

  return substring
```

Algoritma diatas merupakan algoritma pencarian nilai maksimal biasa. Awalnya nilai `max` diinisasi nol dan `substring` dengan string kosong. Setiap iterasi, elemen `result` akan disimpan sementara di `val`. Kemudian diperiksa apakah nilai `max` lebih kecil dari `count`, jika ya nilai `max` sekarang diperbarui serta nilai `substring`-nya.

Setelah semua *substring* yang diperlukan terkumpul saatnya menentukan jarak antara *substring-subtring* tersebut, berikut algoritmnya :

```
array of integer CalculateLength(positions
: array of integer)
  length, i : integer
  result : array of integer

  length = positions.length – 1
  for (i := 0, i < length, i := i + 1)
    result.add(positions[i + 1] –
    positions[i]);

  return result;
```

Langkah selanjutnya yaitu mencari semua faktor pembagi dari jarak tersebut kemudian tentukan irisan dari himpunan faktor pembaginya. Sebelum melangkah kesana, berikut algoritma untuk menghitung faktor pembagi.

```
array of integer CalculateDivisor(number :
integer)
  divisor : array of integer
  div, i : integer

  div = number / 2 + 1
  for (i := 0, i < div, i := i + 1)
    if (number mod i = 0)
      divisor.add(i)

  return divisor
```

Langkah terakhir yaitu mencari irisan dari faktor pembagi.

```
array of integer IntersectDivisor(positions
: array of integer)
  lengths, temp, result : array of integer
  numbers : array of <integer, integer>
  l,t : integer
  ii : <integer, integer>

  lengths := CalculateLength(positions)
  foreach l in lengths
    temp = CalculateDivisor(l)
    foreach t in temp
      if numbers.ContainsKey(t)
        numbers[t]++
      else
        numbers[t] := 1

  // Intersect
  foreach ii in numbers
    if ii.Second >= 2
       result.add(ii.First)

  return result
```

Diawal fungsi `CalculateLength` dipanggil untuk mendapatkan jarak antar *substring*. Setelah itu masing – masing jarak dicari faktor pembaginya dengan memanggil

fungsi `CalculateDivisor`. Kemudian, setiap nilai faktor pembagi dimasukkan ke `numbers`. `numbers` merupakan *array* dari pasangan dua *integer*. *Integer* pertama berfungsi sebagai kunci yaitu bilangan faktor pembagi sedang integer kedua merupakan berapa banyak faktor pemabgi tersebut telah dimasukkan.

Di langkah selanjutnya melakukan *intersect*. Bilangan dengan kemunculan lebih dari satu termasuk ke dalam himpunan *intersect*.

Langkah paling terakhir yaitu menampilkan himpunan *intersect* tadi. Himpunan *intersect* tidak lain adalah kemungkinan panjang kunci.

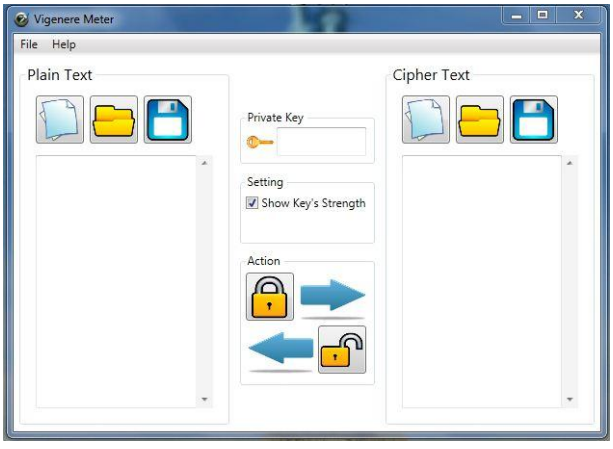
B. Penentuan Panjang Kunci

Penentuan apakah kunci termasuk kuat atau tidak yaitu dengan melihat jumlah himpunan *intersect*-nya. Kunci dianggap kuat jika himpunan *intersect*-nya kosong yaitu tidak bisa ditebak panjang kuncinya. Sebaliknya, kunci dianggap tidak kuat jika himpunan *intersect*-nya tidak kosong. Kunci tergolong sangat lemah jika himpunan *intersect*-nya hanya memiliki satu elemen.

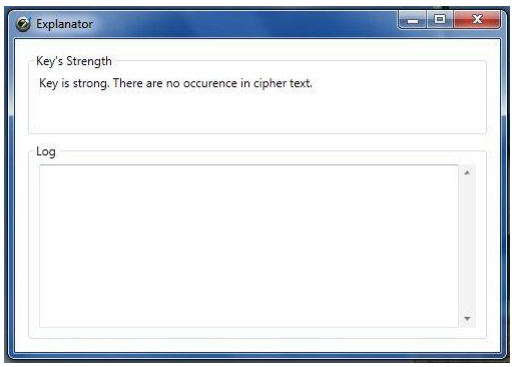
IV. IMPLEMENTASI

Implementasi menggunakan bahasa C# (*Windows Presentation Foundation*) dengan framework .NET. Implementasi pseudo code ke dalam kode C# ditampilkan agak berbeda namu tujuannya tetap sama. Berikut tampilan dari program untuk mengukur kekuatan vigenere cipher. Program ini diberi nama *Vigenere Meter*.

Vigenere cipher yang digunakan pada program ini yaitu Vigenere 26 karakter. Batasan kunci yang bisa diterima maksimal 256 karakter. Selain itu plain teks, kunci serta cipher teks tidak boleh kosong saat enkripsi dilakukan.



Gambar 1 Jendela utama Vigenere Meter



Gambar 2 Jendela Explanation Vigenere Meter

Vigenere Meter terdiri dari dua jendela yaitu jendela utama dan jendela *Explanation*. Window utama memiliki empat bagian yaitu editor *plaintext*, editor *ciphertext*, *private key*, *setting* serta *action*. Jendela *Explanation* merupakan jendela untuk member tahu apakah kunci yan dimasukan termasuk kuat atau tidak. Berikut penjelasan masing – masing elemen yang terdapat pada *Vigenere Meter* :

Jendela Utama

- Editor *plaintext* dan *ciphertext* berfungsi untuk mengelola teks dan file. Editor bisa membuat file baru, bisa membuka file serta menyimpan file.
- *Private key* merupakan box / kotak untuk menempatkan kunci.
- *Setting* merupakan *box* / kotak yang berisi *check box* apakah jendela *Explanation* akan ditampilkan setelah enkripsi.
- Kotak *Action* berisi dua tombol yaitu tombol enkripsi (ditandai dengan gambar gembok terkunci) serta tombol dekripsi (ditandai dengan gambar gembok terbuka). Ketika tombol enkripsi ditekan terlebih dahulu divalidasi apakah editor *plaintext* atau kotak *private key* sedang kosong atau tidak. Jika tidak, enkripsi akan dilakukan dan editor *ciphertext* berisi hasil enkripsi. Begitu saat menekan tombol dekripsi.

Jendela *Explanation*

- Kotak *Key's strength* berisi pernyataan apakah kuci yang dimasukkan termasuk kuat atau belum.
- Kotak *Log* akan berisi proses serta alasan untuk kotak *Key's strength*

Seluruh kode program Vigenere Meter dapat diperoleh melalui http://code.google.com/p/code-room/source/browse/trunk/VigenereMeter/.

V. PENGUJIAN

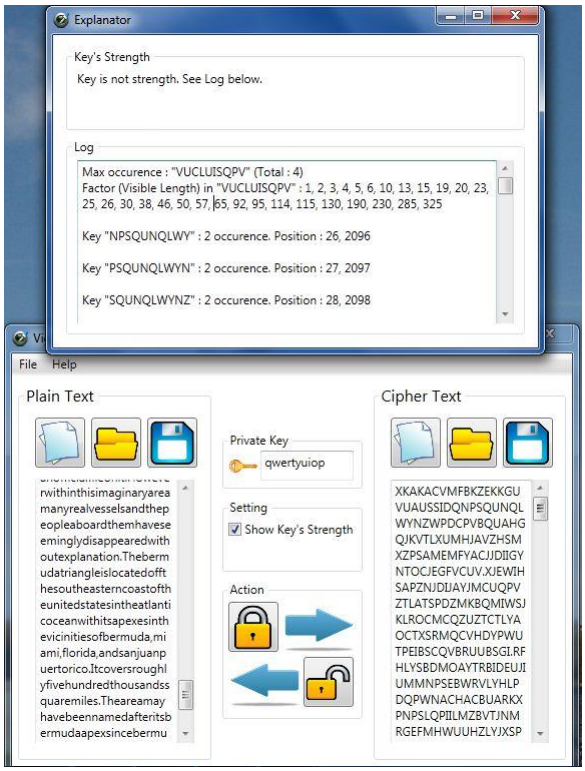
Pada bagian ini akan dilakukan pengujian cara kerja *Vigenere Meter*. Plainteks yang digunakan adalah plainteks dalam bahasa inggris dengan ukuran yang cukup panjang.

> HowthebermudatriangleworksThebermudatriangleisastrangeand mysteriousplaceinthesouthernatlanticocean.Itisroughlytheshapeo fatriangleanditissaidtohavesuckedplanesandboatsintoitsdarkand murkywaters.Nooneknowswhathappenedtotheplanesandboatswh ensomeenterthewatersofthetriangletheydisappearforever.Foroverf ortyyears.Thebermudatrianglehasbeenpopularlyknownforsuppos edlyparanormaldisappearancesofboatsandaircraft.Thisimaginaryt rianglealsoknownasdevilstriangle.Hasitsthreepointsatmiami,puer torico,andbermudaactuallydespiteseveralfactorswhichshouldcont ributetohigherratesofaccidentsintheregion.Thebermudatriangleha sbeenfoundtobenomorestatisticallydangerousthanotherareasofthe openocean.Thepopularlegendofthebermudatrianglebeganwithani neteensixtyfourarticleinthemagazineargosythatdescribedandnam edthetriangle.Furtherarticlesandreportsinsuchmagazinesasnation algeographicandplayboymerelyrepeatedthelegendwithoutadditio nalresearch.Manyofthedisappearancesdiscussedinthesearticlesan dothersdidnotevenoccurintheareaofthetriangle.Youwontfinditona nyofficialmapandyouwontknowwhenyoucrosstheline.Butaccordi ngtosomepeople,thebermudatriangleisaveryrealplacewheredozen ofships,planes,andpeoplehavedisappearedwithnogoodexplanatio ns.Inceamagazinefirstcoinedthephrasebermudatriangleinnineteen sixtyfour.Themysteryhascontinuedtoattractattentionwhenyoudig deeperintomostcasesthoughtheyremuclessmysteriouseither.They wereneverintheareatobeginwiththeywereactuallyfoundortheresar easonableexplanationfortheirdisappearancedoesthismeantheresn othingtotheclaimsof.Somanywhohavehadoddexperiencesinthebe rmudatrianglenotnecessarilyscientistshavedocumenteddeviations fromthenormitheareaandhavefoundsomeinterestingformationson theseafloorwithinthebermudatrianglesboundaries.Soforthosewho liketobelieveinitthereisplentyfuelforthefireinthisarticle.Well,look atthefactssurroundingwhatwedoknowabouttheareaaswellassome ofthemostcommonlyrecitedstorieswellalsoexplorethebizarretheor ieslikealiensandspaceportalsaswellasthemundaneexplanations.M anythinkofthebermudatrianglealsoknownasthedevilstriangleasani maginaryarea.Theusboardofgeographicnamesdoesnotrecognizeth ebermudatriangleanddoesnotmaintainanofficialfileonit.However withinthisimaginaryareamanyrealvesselsandthepeopleaboardthe mhaveseeminglydisappearedwithoutexplanation.Thebermudatria ngleislocatedoffthesoutheasterncoastoftheunitedstatesintheatlant icoceanwithitsapexesinthevicinitiesofbermuda,miami,florida,and sanjuanpuertorico.Itcoversroughlyfivehundredthousandssquarem iles.Theareamayhavebeennamedafteritsbermudaapexsincebermu dawasonceknownastheisleofdevilstreacherousreefsthathaveensna redshipssailingtooclosetoitsshoressurroundbermudaandthereareh undredsofshipwrecksinthewatersthatsurroundit.

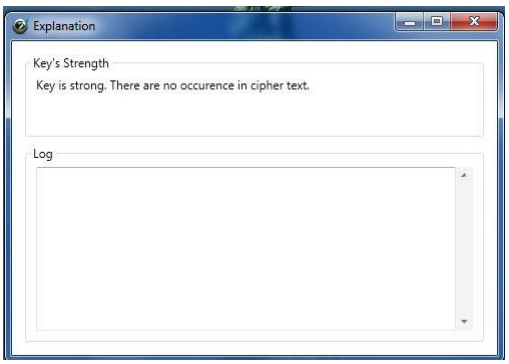
Pengujian pertama dilakukan dengan menggunakan kunci : `qwertyuiop`. Kopi file diatas ke Vigenere Meter kemudian centang *Show Key's Strength* di bagian *setting* setelah itu klik *encrypt*. Hasil eksekusi kurang lebih seperti pada gambar 3 dibawah ini. Jendela *Explanation* member informasi bahwa kunci tidak kuat. Penjelasan lebih lanjut terdapat pada kotak *Log* di jendela yang sama.

Pengujian kedua dilakukan dengan menggunakan kunci yang lumayan panjang yaitu : `qwertyuiopasdfghjkl`. Hasil eksekusi bisa dilihat pada gambar 4. Pada gambar tersebut terlihat jelas bahwa program menyatakan bahwa kunci termasuk kuat. Kotak Log juga terlihat kosong yang berarti bahwa tidak ditemukan string berulang yang berukuran sepanjang panjang kunci.

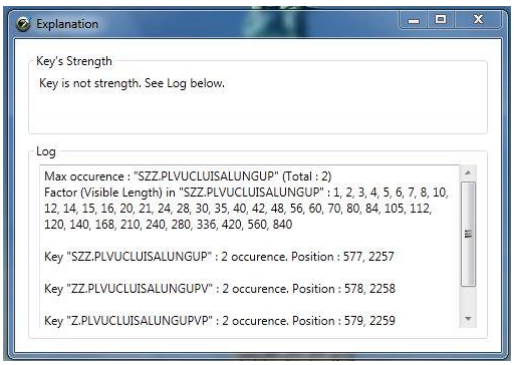
Pengujian ketiga dilakukan dengan menggunakan kunci yang hamper sama dengan pengujian kedua tetapi hanya berbeda satu karakter yaitu : `qwertyuiopasdfghjklm`. Hasil eksekusi terlihat pada gambar 5. Hasil yang diperoleh berbeda dengan pengujian kedua. Pengujian ini menampilkan hasil yang cukup mengejutkan yaitu kunci yang dimasukkan tergolong lemah walaupun terlihat lebih panjang.



Gambar 3 Pengujian dengan kunci `qwertyuiop`



Gambar 4 Pengujian dengan kunci `qwertyuiopasdfghjkl`



Gambar 5 Pengujian dengan kunci qwertyuiopasdfghjklm

VI. ANALISIS

Dari dua pengujian sebelumnya, kunci dengan ukuran yang lebih pendek lebih lemah daripada kunci dengan ukuran yang lebih panjang. Hal ini terjadi karena kunci yang lebih pendek memiliki peluang yang lebih besar untuk menciptakan kriptogram yang berulang.

Kunci yang lebih panjang tidak selalu menjamin bahwa kunci itu lebih kuat. Sebagai contoh kasus pada pengujian kedua dan ketiga. Pengujian kedua menggunakan kunci yang lebih pendek dari pada pengujian ketiga dan hasil menunjukkan bahwa kunci pada pengujian kedua lebih kuat.

Kenapa kunci yang panjang sekalipun belum tentu lebih kuat dari pada kunci yang pendek? Banyak factor yang mempengaruhi. Beberapa diantaranya adalah karakter-karakter yang dipilih untuk menjadi kunci serta pola-pola tulisan berulang yang tersebar di plainteks. Jika karakter yang dipilih menjadi kunci terlalu banyak memilii kesamaan kemungkinan untuk membuat kunci itu menjadi lemah semakin besar. Usahakan karakter di setiap kunci seunik mungkin.

Pola pada plainteks memang susah dicocokkan dengan kunci yang dipilih. Salah satu cara yang paling mudah yaitu dengan *brute force*. Setiap melakukan enkripsi, hasil dekripsi dilihat kembali apakah cukup tahan untuk tidak dijebol. Apalagi jika text yang dienkripsi merupakan teks yang panjang. Kemungkinan untuk mendapat pola yang sama lebih besar lagi. Inilah bagian kelamahan dari Vigener cipher. Vigenere cipher bisa diserang melalui kemunculan kriptogram yang sama.

VII. KESIMPULAN

Beberapa kesimpulan yang dapat ditarik dari pembahasan sebelumnya yaitu :

1. Kekuatan kunci pada Vigenere cipher dapat diketahui dengan memanfatkan metode kasiski.
2. Kunci yang lebih panjang tidak selalu lebih kuat, hal itu tergantung dari pemilihan karakter pada kunci serta pola kata yang tersebar pada plainteks.
3. Kriptogram yang berulang memang sulit dihindari apalagi untuk text yang panjang karena dalam satu text cenderung muncul kata yang sama berulang-ulang kali.

REFERENCES

[1] http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/Kriptanalisis.ppt

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, 23 Maret 2011

Rezan Achmad / 13508104