MERANCANG ALGORITMA KRIPTOGRAFI KUNCI SIMETRI DENGAN MEMPERLUAS ALGORITMA VIGÈNERE DAN ANALISIS METODE KASISKI TERHADAP ALGORITMA TERSEBUT

Ridwan – NIM : 13503072

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13072@students.if.itb.ac.id

Abstrak

Sebelum adanya komputer, kriptografi dilakukan berbasis karakter dengan hanya menggunakan kertas dan pena. Algoritma kriptografi yang digunakan saat itu termasuk dalam sistem kriptografi kunci simetri dan digunakan jauh sebelum ditemukannya sistem kunci publik. Algoritma kriptografi yang berbasis karakter biasanya termasuk dalam salah satu dari *cipher* substitusi, *cipher* transposisi, atau super enkripsi yang merupakan gabungan dari *cipher* substitusi dan *cipher* transposisi.

Salah satu algoritma kriptografi kunci simetri berbasis karakter yang cukup lama digunakan adalah algoritma kriptografi Vigènere *Cipher*. Algoritma ini dapat dikatakan baik karena termasuk *cipher* abjad-majemuk yang merupakan salah satu bagian dari *cipher* substitusi. Namun, saat ini algoritma ini sudah tidak digunakan lagi karena pada tahun 1863 Freidrich Kasiski telah menemukan cara memecahkan Vigènere *Cipher*.

Makalah ini membahas mengenai perancangan sebuah algoritma kriptografi kunci simetri dengan memperluas algoritma kriptografi Vigènere. Dalam makalah ini, algoritma kriptografi yang dirancang akan diujikan dengan metode yang biasanya digunakan untuk melakukan kriptanalis terhadap *ciphertext* hasil enkripsi dari algoritma kriptografi Vigènere, yaitu metode Kasiski.

Penulis merasa algoritma kriptografi Vigènere *Cipher* masih dapat dikembangkan sehingga semakin sulit untuk dipecahkan, meskipun dengan metode Kasiski. Oleh karena itu, penulis akan mencoba membuat sebuah algoritma yang dikembangkan berdasarkan Vigènere *Cipher*, dimana algoritma baru ini diusahakan tidak dapat dipecahkan dengan metode Kasiski. Algoritma yang akan dirancang dapat dilihat seperti membuat sebuah algritma kriptografi modern karena menggunakan bilangan biner. Namun, algoritma yang dirancang masih berupa algoritma kunci simetri, dan termasuk dalam *cipher* substitusi. Oleh karenanya, penulis menyebut hasil rancangan algoritma tersebut dengan nama "*Vigènere Biner*".

Kata Kunci: Kriptografi, Algoritma, Kunci Simetri, *Cipher* Substitusi, *Cipher* Abjad-Majemuk, Vigènere *Cipher*, Metode Kasiski, *Vigènere Biner Cipher*

1. Pendahuluan

Kemajuan di bidang telekomunikasi dan komputer telah memungkinkan seseorang untuk melakukan transaksi bisnis dan bertukar informasi secara on-line melalui internet. Kemajuan ini menimbulkan permasalah sendiri daam hal keamanan data yang saling dipertukarkan melalui jaringan. Salah satunya adalah apabila ada sebuah informasi yang sangat sensitif, kemingkinan informasi tersebut untuk diakses oleh pihak yang tidak berhak (*unauthorized persons*) sangat tinggi. Misalnya, informasi mengenai nomor kartu kredit anda, bila informasi ini jatuh kepada orang-orang yang jahat maka anda harus bersiap-siap terhadap melonjaknya tagihan kartu kredit anda.

Kriprografi atau penyandian termasuk metode pengamanan informasi yang tangguh. Disebut tangguh karena, sekali data tersebut disandikan dengan algoritma sandi yang baik, data tersebut akan tetap aman kendati setiap orang dapat mengaksesnya secara bebas. Dan idealnya selama algoritma sandi tersebut tetap terjaga,

data yang disandikan akan tetap aman. Tentu saja sampai saat ini tidak ada algoritma kriptografi yang 100% aman.

Konsep dari kriptografi adalah mengacak data (*plaintext*) dengan suatu metode tertentu (algoritma sandi) dan dengan menggunakan sebuah kunci rahasia maka akan menjadi kumpulan karakter yang tidak bermakna (*ciphertext*).

Misalnya teks asli: pengalaman adalah guru yang terbaik. Setelah disandikan dengan algoritma sandi xyz dan dengan kunci pqr menjadi teks sandi: 5(tjg$gjbvBGd1297j?òè156a8U8A7£+Y. Proses ini disebut enkripsi dan proses sebaliknya disebut dekripsi.

Dalam prakteknya kriptografi digunakan untuk melindungi kerahasiaan data dan menjamin integritas data. Kriptografi biasanya hanya diterapkan pada data-data yang dinilai penting dan sensitif, yang perlu dilindungi dari akses pihak-pihak yang tidak diinginkan dan dari potensi ancaman pencurian oleh pihak-pihak yang memperoleh akses terhadapnya. Secara prinsip, keamanan data yang disandi sangat tergantung dari terjaganya kerahasiaan kunci dan algoritma sandinya.

Dapat dikatakan bahwa kriptografi hanya mengubah masalah keamanan. Yaitu mengubah dari melindungi data rahasia (besar,kompleks dan banyak) menjadi melindungi algoritma sandi dan kunci (satu hal).

Biasanya data-data dilindungi baik kerahasiaannya maupun integritasnya. Tetapi terkadang data tertentu tidak perlu dirahasiakan namun perlu dijaga integritasnya. Kriptografi dapat juga digunakan untuk menjamin integritas data yaitu agar data tidak dimanipulasi oleh pihak-pihak yang tidak diinginkan. Kriptografi tidak menjamin keamanan 100 %, sebab tidak ada pengamanan yang sempurna.

Pada dasarnya, tujuan dari kriptografi adalah :

- **Convidentiality**
  Yaitu memberikan kerahasiaan pesan dan menyimpan data dengan menyembunyikan informasi lewat teknik-teknik enkripsi.

- **Massage Integrity**
  Yaitu memberikan jaminan untuk tiap bagian bahwa pesan tidak akan mengalami perubahan dari saat data dibuat/dikirim sampai dengan saat data tersebut dibuka.

- **Non-repudiation**
  Yaitu memberikan cara untuk membuktikan bahwa suatu dokumen datang dari seseorang apabila ia mencoba menyangkal memiliki dokumen tersebut.

- **Authentication**
  Yaitu mengidentifikasikan keaslian suatu pesan dan memberikan jaminan keotentikannya.

Perkembangan teknologi pengamanan selalu diimbangi dengan teknologi untuk membongkar keamanan yang diterapkan. Selain itu tingkat kesadaran individu yang bersentuhan dengan data-data yang diamankan tersebut, sangat menentukan lambat atau cepatnya sebuah pengamanan terbongkar.

Perkembangan kriptografi memang sangat pesat. Para ahli kriptografi (cryptographers) terus menerus menciptakan algoritma-algoritma kriptografi yang baru. Hal ini dikarenakan pula semakin banyak orang-orang ahli yang mampu memecahkan kode-kode ciphertext ke dalam plaintext. Orang-orang ahli semacam ini sering disebut sebagai cryptoanalysis. Ketika suatu algoritma kripografi sudah dapat dipecahkan, maka diperlukan algoritma-algoritma baru yang lebih handal agar keamanan data tetap terjaga.
Hal ini menyebabkan kriptografi tak akan pernah berhenti berkembang.

Algoritma-algoritma kriptografi dapat dibedakan menjadi dua macam, yaitu : simetrik dan asimetrik. Algoritma simetrik adalah algoritma yang menggunakan satu kunci untuk proses enkripsi dan dekripsi data. Sedangkan algoritma asimetrik (model enkripsi kunci publik) menggunakan kunci yang berbeda dalam proses enkripsi dan dekripsi pesan.

Pada awalnya, semua algoritma kriptografi merupakan algoritma kunci simetrik. Pada algoritma kunci simetrik, kunci untuk membuat pesan yang disandikan sama dengan kunci untuk membuka pesan yang disandikan itu. Jadi pengirim pesan dan penerima pesan harus memiliki kunci yang sama persis. Siapapun yang memiliki kunci tersebut termasuk pihak-pihak yang tidak diinginkan dapat membuat dan membongkar rahasia ciphertext.

Masalah yang didapat dengan menggunakan algoritma kunci simetrik bukan lagi cara untuk mengirimkan ciphertext, melainkan bagaimana cara mengirimkan kunci untuk membuka ciphertext tersebut kepada pihak yang diinginkan. Dengan kata lain ada masalah *pendistribusian kunci rahasia* . Contoh algoritma kunci simetris yang terkenal adalah Vigènere cipher, DES (data encryption standart) TripleDES, IDEA, Blowfish, Twofish, AES (advanced encryption standard) dan RC-4.

2. Algoritma Vigènere dan Analisis Metode Kasiski

Algoritma Vigenere *cipher* merupakan salah satu contoh algoritma kriptografi yang bagian dari *Cipher* Subtitusi, khususnya subtitusi abjad majemuk. Oleh karena itu, makalah ini akan membahas mengenai Cipher subtitusi dan jenis-jenis algoritma yang termasuk dalam Cipher Subtitusi terlebih dahulu.

2.1 *Cipher* Subtitusi

Algoritma kriptografi klasik terbagi atas dua macam, yaitu *Cipher* Subtitusi dan Cipher Transposisi. Inti dari metode *Cipher* Subtitusi adalah mengganti setiap karakter dalam *plaintext* dengan karakter lainnya.

Salah satu contoh paling sederhana dari *Cipher* Subtitusi adalah algoritma kriptografi *Caesar Cipher*. Algoritma *Caesar Cipher* yang digunakan oleh Julius Caesar pada jaman peperangan adalah dengan mengganti setiap huruf pada pesan yang akan dikirimkan dengan huruf ketiga berikutnya dalam susunan abjad.

Tabel subtitusi (Pergeseran tiga huruf):

p	a	b	c	d	e	f	g	h	i	j	k	l	m
c	d	e	f	g	h	i	j	k	l	m	n	o	p

p	n	o	p	q	r	s	t	u	v	w	x	y	z
c	q	r	s	t	u	v	w	x	y	z	a	b	c

Dengan mengikuti tabel tersebut, maka pesan

```
AWASI ASTERIX DAN OBELIX
```

Disamarkan (enkripsi) menjadi

```
DZDVL DVWHULA GDQ REHOLA
```

Penerima pesan dapt men-dekripsi cipherteks tersebut dengan menggunakan tabel subtitusi.

Dengan mengkodekan setiap huruf abjad dengan *integer* sebagai berikut: A=0, B=1, C=2, ..., Z=25, maka secara matematis *Caesar Cipher* melakukan proses enkripsi sesuai dengan aturan:

$c_i = E(p_i) = (p_i + 3) \bmod 26$

dan melakukan proses dekripsi sesuai dengan aturan:

$p_i = D(c_i) = (c_i - 3) \bmod 26$

Pergeseran sebanyak 3 huruf yang digunakan oleh Julius Caesar mudah ditebak kuncinya, yaitu 3. Kita dapat mengglobalisasikan algoritma tersebut dengan jumlah pergeseran yang launnya. Karena ada 26 huruf abjad, maka pergeseran huruf dapat dilakukan sebanyak 0 sampai 25 huruf. Secara umum, untuk pergeseran sejauh *k* huruf (dalam hal ini *k* adalah kunci enkripsi dan dekripsi yang memiliki nilai antara 0 sampai 25), maka aturan fungsi enkripsi menjadi:

$c_i = E(p_i) = (p_i + k) \bmod 26$

dan fungsi dekripsi menjadi:

$p_i = D(c_i) = (c_i - k) \bmod 26$

Untuk lebih luasnya lagi, kita dapat melakukan enkripsi terhadap karakter ASCII. Karakter ASCII terdiri dari 256 karakter, sehingga pergeseran huruf bisa sejauh 0 – 255 huruf dimana jumlah pergeseran adalah kunci dari fungsi enkripsi dan fungsi dekripsi. Dengan begitu, fungsi enkripsi menjadi:

$c_i = E(p_i) = (p_i + k) \bmod 256$

dan fungsi dekripsi menjadi:

$p_i = D(c_i) = (c_i - k) \bmod 256$

Kelemahan dari Caesar *Cipher* adalah bahwa Caesar *Cipher* mudah dipecahkan dengan metode *exhaustive key search* karena jumlah kuncinya terbatas. Misalkan seorang kriptanalis menemukan potongan cipherteks `XMZVH`. Misalkan kriptanalis tersebut mengetahui bahwa plaionteks disusun dalam Bahasa Inggris dan algoritma kriptografi yang digunakan adalah *caesar cipher* yang tidak menggunakan karakter ASCII. Maka, kriptanalis tersebut dapat

melakukan *exhaustive search* terhadap semua kemungkinan kunci.

Kriptanalis tersebut dapat mencoba mendekripsi mulai dari kunci k = 25 sampai dengan k = 1. Dari percobaan deksripsi tersebut, akan ada hasil yang memiliki makna, maka kemungkinan besar kunci yang digunakan adalah kunci untuk melakukan deksripsi yang menghasilkan pesan yang bermakna.

Key	Hasil	Key	Hasil
0	XMZVH	13	KZMIU
25	YNAWI	12	LANJV
24	ZOBXJ	11	MBOKW
23	APCYK	10	NCPLX
22	BQDZL	9	ODQMY
21	**CREAM**	8	PERNZ
20	DSFEN	7	QFSOA
19	ETGCO	6	RGTPB
18	FUHDP	5	SHUQC
17	GVIEQ	4	TIVRD
16	HWJFR	3	UJWSE
15	IXKGS	2	VKXTF
14	JYLHT	1	WLYUG

Dari tabel tersebut, kata dalam Bahasa Inggris yang potensial menjadi plainteks adalah CREAM dengan menggunakan K = 21. Kunci ini dapat digunakan untuk menggunakan cipherteks lainnya.

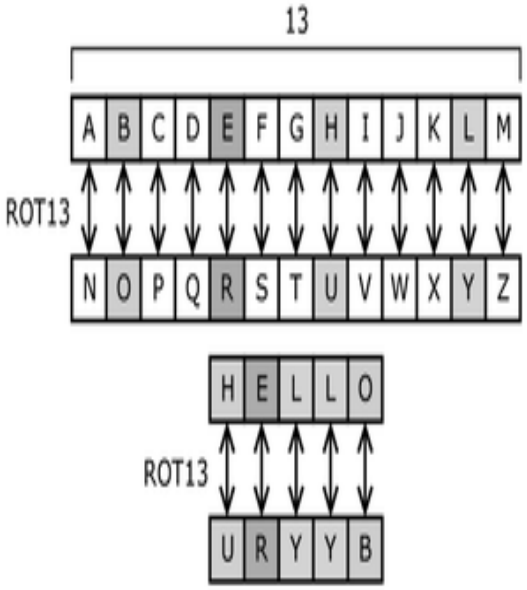
Contoh algoritma kriptografi yang termasuk dalam *Cipher* Subtitusi masih banyak, diantaranya adalah Vigènere *Cipher* yang akan dibahas dalam makalah ini.

2.1.1 *Cipher* Abjad Tunggal

Cipher abjad tunggal adalah salah satu jenis dari Cipher Subtitusi. Pada *cipher* abjad tunggal, satu karakter pada plainteks akan selalu disubtitusi dengan tepat satu huruf selama menggunakan kunci enkripsi yang sama. Jadi dapat disebutkan bahwa fungsi enkripsi dari cipher subtitusi abjad tunggal merupakan fungsi satu ke satu. Jika plainteks terdiri dari huruf-huruf abjad, maka jumlah kemungkinan susunan huruf cipherteks yang dapat dibuat adalah sebanyak:

```
26! = 403.291.461.126.605.635.584.000.000
```

Algoritma enkripsi Caesar *Cipher* yang telah dijelaskan pada bagian sebelumnya merupakan salah satu contoh penerapan cipher abjad tunggal. Salah satu penerapan algoritma enkripsi Caesar Cipher adalah ROT13. ROT13 merupakan program enkripsi sederhana yang ditemukan pada sistem UNIX. ROT13 menggunakan algoritma enkripsi Caesar *Cipher* dengan kunci pergeserannya adalah 13 (k = 13). Dengan demikian dapat dilihat bahwa melakukan enkripsi dua kali pada plainteks dengan algoritma enkripsi ROT13 menghasilkan pesan semula. Sistem enkripsi pada ROT13 dapat dilihat sebagai berikut:



Pada algoritma enkripsi subtitusi abjad tunggal bisa juga dengan cara membuat table subtitusi sendiri. Misalkan dengan membuat table subtitusi secara acak:

```
p: abcdefghijklmnopqrstuvwxyz
c: qwertyuioplkjhgfdsazxcvbnm
```

Tabel subtitusi juga dapat dibangkitkan dari sebuah kalimat yang berperan sebagai kunci dari proses enkripsi dekripsi. Misal dipilih kalimat “itb jalan ganesha” sebagai kunci, maka:

- buang huruf-huruf berulang dan spasi

  ```
  itbjalngesh
  ```

- sambung dengan huruf lainnya

  ```
  itbjalngeshcdfkmopqruvwxyz
  ```

- sehingga table subtitusi menjadi:

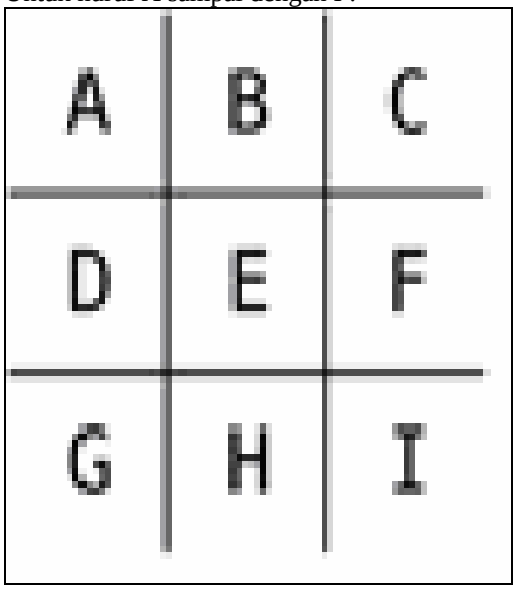
```
p:abcdefghijklmnopqrstuvwxyz
c:itbjalngeshcdfkmopqruvwxyz
```

Contoh algoritma enkripsi subtitusi cipher abjad tunggal lainnya adalah Atbash *cipher*. Pada algoritma enkripsi Atbash cipher, huruf pertama pada susunan alphabet akan diganti dengan huruf terakhir, lalu huruf kedua dengan satu huruf sebelum terakhir. Jadi, huruf A akan diubah menjadi huuf Z, lalu huruf B akan diubah menjadi huruf Y. Sistem subtitusi pada algoritma enkripsi Atbash cipher adalah:

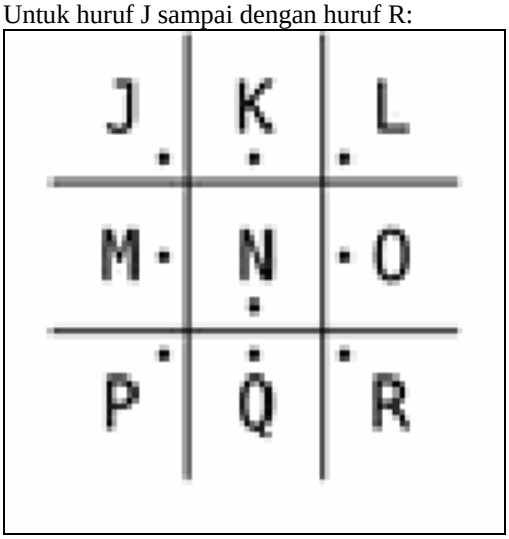
Plainteks: `abcdefghjklmnopqrstuvwxyz`
Cipherteks:`zyxwvutsrqponmlkjhgfedcba`

Contoh lainnya adalah algoritma enkripsi pigpen cipher. Dalam algoritma enkripsi ini, digunakan simbol-simbol untuk mengganti karakter-karakter dalam plainteks. Pigpen cipher adalah algoritma enkripsi subtitusi abjad tunggal sederhana. Kadang algoritma ini disebut Masonic cipher atau freemason's cipher. Tabel enkripsi yang digunakan dalam pigpen cipher adalah:

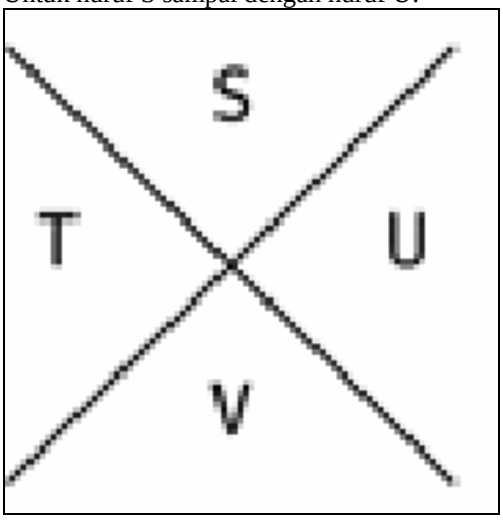
Untuk huruf A sampai dengan I :



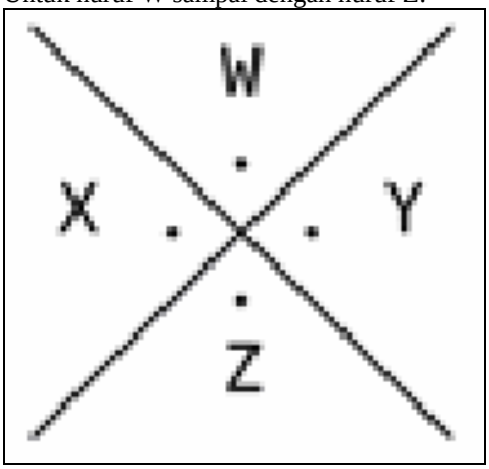
Untuk huruf J sampai dengan huruf R:



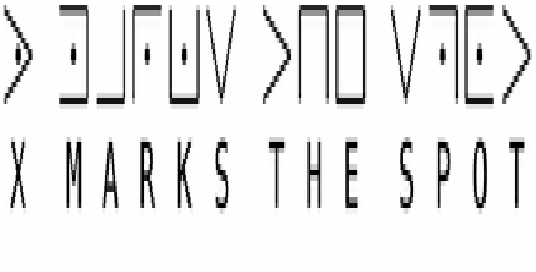
Untuk huruf S sampai dengan huruf U:



Untuk huruf W sampai dengan huruf Z:



Sehingga contoh enkripsi yang menggunakan pigpen cipher adalah:



Pada algoritma enkripsi abjad tunggal, walaupun jumlah susunan huruf yang dapat dibentuk sangat besar (26!, mendekati $2^{88.4}$, atau sekitar 88 bit), algoritma enripsi dengan subtitusi abjad tunggal ini tidak begitu kokoh, dan dapat dipecahkan dengan mudah. Penggantian satu karakter ke tepat satu karakter lainnya akan memudahkan proses kriptanalisis. Contohnya adalah kata *see* dalam bahasa inggris. Hasil enkripsi dari kata tersebut akan membentuk pola ABB pada cipherteks. Hal ini sangat memudahan kriptanlis karena sedikit dari kata dalam bahasa inggris yang berpola seperti itu, sehingga semua huruf pada cipherteks yang bersesuaian dengan huuf s dan e pada plainteks dapat dipecahkan.

2.1.2 *Cipher* Substitusi Homofonik

Chipper subtitusi homophonic merupakan algoritma enkripsi yang lebih kuat daripada chipper subtitusi abjad tunggal. Pada dasarnya chipper subtitusi homofonik sama seperti chipper subtitusi abad tunggal, hanya saja dalam cipher subtitusi homofonik setiap karakter dalam plainteks dapat dipetakan ke dalam salah satu dari karakter chiperteks yang mungkin.

Misalnya huruf A dapat berkoresponden dengan 7, 9, atau 16, huruf B dapat berkoresponden dengan 5, 10, atau 23, dan seterusnya. Dapat dilihat dari penjelasan tersebut bahwa algortima ankripsi dalam cipher subtitusi homofonik merupakan fungsi yang memetakan satu ke banyak.

Pada tahun 1988 Gunther memperkenalkan perluasan yang penting dari cipher subtitusi homofonik, yaitu yang disebut dengan *variable-length homophonic subtitution*.

2.1.3 *Cipher* Abjad Majemuk

Berbeda dengan cipher subtitusi abjad tunggal, cipher subtitusi abjad majemuk memetakan satu karakter pada plainteks ke dalam lebih dari satu karakter dalam cipherteks. Cipher subtitusi abjad majemuk merupakan cipher subtitusi ganda yang melibatkan penggunaan kunci yang berbeda.

Cipher subtitusi abjad majemuk dibuat dari sejumlah cipher abjad tunggal, masing-masing memiliki kunci yang berbeda. Kebanyakan cipher subtitusi abjad majemuk adalaj cipher subtitusi periodik yang didasarkan pada periode *m*.

Misalkan plainteks P adalah:

$P = p_1 p_2 \ldots p_m p_{m+1} \ldots p_{2m} \ldots$

Maka cipherteks hasil enkripsi adalah:

$E_k(P) = f_1(p_1) f_2(p_2) \ldots f_m(p_m) f_{m+1}(p_{m+1}) \ldots f_{2m}(p_{2m})$

Yang dalam hal ini p_i adalah huruf-huruf dalam plainteks.

Contoh lainnya dari cipher subtitusi abjad majemuk dapat dilihat dalam tabel berikut:

KEY:
Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ _ Ciphertext: PQOWIEURYTLAKSJDHFGMZNX_BC V (Position 1) LP_MKONJIBHUVGYCFTXDRZSEAW Q (Position 2) GFTYHBVCDRUJNXSEIKM_ZAWOLQ P (Position 3)
ENCRYPTION:
Position: 12312312312 Plaintext: HOW_ARE_YOU **Ciphertext RYWVLKIQLJR**

Jumlah kemungkinan kunci untuk cipher subtitusi abjad majemuk yang menggunakan ukuran alfabet 26, dan ukuran blok pemisah pesan B adalah $26!^{B}$. jumlah ini jauh lebih besar bila dibandingkan dengan cipher subtitusi biasa, terutama saat periode B bernilai besar.

Cipher abjad majemuk pertama kali ditemukan oleh Leon Battista Alberti pada tahun 1568. Alberti menggunakan Caesar cipher untuk melakukan enkripsi sebah pesan, namun kapanpun dia mau dia akan menggantinya ke kunci yang lain. Alberti menandai pergantian kuncinya dengan membuat huruf kapital setiap huruf pertama pada cipherteks. Alberti juga membuat sebuah alat pengkodean yang mengimplementasikan sebuah cipher ekivalen dengan yang dipublikasikan oleh Johannes Trithemius.

Johannes Trithemius menemukan sebuah kunci progresif cipher abjad majemuk. Tidak seperti cipher milik Alberti, yang merubah kunci dalam interval yang dibangun secara random, Trithrmius mengganti kunci dalam setiap huruf dalam pesan yang akan dienkripsi. Ia memulai dengan sebuah *tabula recta*, sebuah kotak dengan 26 huruf alfabet di dalamnya. Setiap alfabet digeser satu huruf ke kiri bila dibandingkan dengan huruf-huruf diatasnya, dan memulai kembali dari A setelah melewati Z.

Ide dari Trithemius adalah untuk merubah huruf pertama dalam pesan dengan pergeseran pertama pada *tabula recta*. Jadi A menjadi B, B menjadi C, dan begitu seterusnya. Huruf kedua dari pesannya diubah sesuai dengan pregeseran huruf kedua pada *tabula recta*. Dapat dilihat algoritma ini sama saja dengan algoritma enkripsi pada Caesar Chiper namun setiap huruf menggunakan kunci pergeseran yang selalu bertambah satu. Cipherteks ini mudah dipecahkan karena terlalu terstruktur.

Contoh lain dari cipher subtitusi abjad majemuk adalah Bifid Cipher. Cara kerja dari Bifid Cipher adalah:

1. Buat sebuah persegi berisi huruf-huruf yang dibangun secara acak dengan nomor-nomor disampingnya.

	1	2	3	4	5
1	B	G	W	K	Z
2	Q	P	N	D	S
3	I	O	A	X	E
4	F	C	L	U	M
5	T	H	Y	V	R

2. Kemudian ubah pesan sesuai dengan koordinat pada persegi yang telah dibuat. Misalkan pesan yang akan dienkripsi adalah flee at once:

F	L	E	E	A	T	O	N	C	E
4	4	3	3	3	5	3	2	4	3
1	3	5	5	3	1	2	3	2	5

3. Kemudian urutkan angka-angka koordinat tersebut sesuai dengan baris.

4 4 3 3 3 5 3 2 4 3 1 3 5 5 3 1 2 3 2 5

4. Kemudian bagi dalam kelompok dimana setiap kelompok terdiri dari sepasang angka yang merepresentasikan koordinat pada persegi sebelumnya. Lalu kembalikan sesuai dengan koordinat yang dibuat:

44	33	35	32	43	13	55	31	23	25
U	A	E	O	L	W	R	I	N	S

Dengan demikian cipherteks yang didapat adalah UAEOLWRINS

2.1.4 *Cipher* Substitusi Poligram

Pada algoritma enkripsi cipher subtitusi poligram, pesan yang akan dienkripsikan di kelompokan dalam beberapa blok kemudian dienkripsi. Misalnya ABBA diganti dengan RTTQ, POO diganti dengan WAA, dan lain-lain.

Salah satu contoh dari cipher subtitusi poligram adalah *playfair cipher*. *Playfair cipher* ditemukan pada tahun 1854, dan digunakan oleh negara Inggris selama Perang Dunia I.

Playfair Cipher ditemukan pertama kali oleh Sir Charles Wheatstone namun dipromosikan oleh Baron Lyon Playfair pada tahun 1854. Cipher ini mengenkripsikan pasangan huruf (bigram), bukan huruf tunggal seperti cipher klasik pada umumnya. Tujuannya adalah untuk membuat analisis frekuensi menjadi sangat sulit sebab frekuensi kemunculan huruf-huruf di dalam cipherteks menjadi datar.

Kunci kriptografi dari playfair cipher adalah 25 huruf yang disusun dalam sebuah bujursangkar 5x5 dengan menghilangkan huruf j dari abjad.

Contoh kunci:

S	T	A	N	D
E	R	C	H	B
K	F	G	I	L
M	O	P	Q	U
V	W	X	Y	Z

Jumlah kemungkinan kunci yang dapat dibentuk adalah sebanyak:

25!=15.511.210.043.330.985.984.000.000

Pada saat melakukan enkripsi, untuk mempermudah proses pengenkripsian, susunan kunci di dalam bujur sangkar diperluas dengan menambahkan kolom keenam dan baris keenam.

S	T	A	N	D	S
E	R	C	H	B	E
K	F	G	I	L	K
M	O	P	Q	U	M
V	W	X	Y	Z	V
S	T	A	N	D	

Baris ke 6 sama dengan baris pertama, dan kolom ke enam sama dengan kolom pertama.

Pesan yang akan dienkripsi diatur terlebih dahulu sebagai berikut:

1. Ganti huruf J (bila ada) dengan huruf I
2. Tulis pesan dalam pasangan huruf (bigram)
3. Jangan sampai ada pasangan huruf yang sama. Jika ada sisipkan Z ditengahnya.
4. Jika jumlah huruf ganjil, tambahkan huruf Z di akhir.

Contoh:
Plainteks: GOOD BROOMS SWEAP CLEAN

Karena tidak ada huruf J, maka langsung tulis pesan dalam pasangan huruf:

GO OD BR OZ OM SZ SW EZ EP CL EA NZ

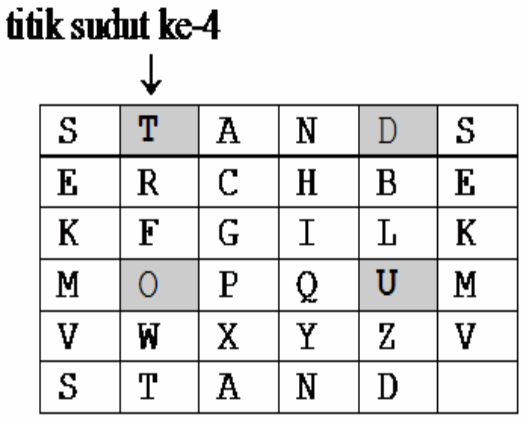
Algoritma enkripsi nya adalah:
1. Jika terdapat dua huruf pada baris kunci yang sama, maka tiap huruf diganti dengan huruf dikanannya.
2. Jika terdapat dua huruf pada kolom yang sama, maka tiap huruf diganti dengan huruf dibawahnya.
3. Jika dua huruf tidak terdapat pada baris yang sama dan kolom yang sama, maka huruf pertama diganti dengan perpotongan baris huruf pertama dengan kolom huruf kedua. Huruf kedua diganti dengan perpotongan baris huruf kedua dengan kolom huruf pertama. Atau bisa disebut dengan titik sudut keempat dari persegi panjang yang dibentuk dari 3 huruf yang digunakan sampai sejauh ini.

Jika menggunakan kunci yang telah dibuat sebelumnya, maka cipherteks dari pesan tersebuat adalah:

FP UT EC UW PO DV TV BV CM BG CS DY

Enkripsi OD menjadi UT ditunjukan pada bujursangkar berikut:

S	T	A	N	D	S
E	R	C	H	B	E
K	F	G	I	L	K
M	O	P	Q	**U**	M
V	W	X	Y	Z	V
S	T	A	N	D	



S	**T**	A	N	D	S
E	R	C	H	B	E
K	F	G	I	L	K
M	O	P	Q	**U**	M
V	W	X	Y	Z	V
S	T	A	N	D	

Pembuatan kunci dapat dengam memilih sebuah kalimat yang mudah diingat, misalnya:

JALAN GANESHA SEPULUH

Lalu buang huruf yang berulang dan huruf J:

ALNGESHPU

Lalu tambahkan huruf-huruf yang belum ada kecuali J:

ALNGESHPUBCDFIKMOQRTVWXYZ

Masukkan ke dalam bujursangkar:

A	L	N	G	E
S	H	P	U	B
C	D	F	I	K
M	O	Q	R	T
V	W	X	Y	Z

Karena ada 26 huruf abjad, maka terdapat 26x26 = 677 bigram, sehingga identifikasi bigram individual lebih sulit.

Sayangnya ukuran poligram di dalam Playfair Cipher tidak cukup besar, hanya dua huruf, sehingga playfair cipher tidak aman. Meskipun playfair cipher sulit dipecahkan dengan analisis frekuensi relatif huruf-huruf, namun ia dapat dipecahkan dengan analisis frekuensi pasangan huruf.

2.2 Algoritma Vigènere

Algoritma enkripsi ini dinamakan Vigènere Cipher atas nama Blaise de Vigènere, meskipun Giovan Batista Belaso telah menemukan algoritma enkripsi ini terlebih dahulu.

Vigènere Cipher adalah sebuah algoritma enkripsi yang menggunakan konsep Caesar Cipher dengan kunci yang berbeda-beda. Vigènere cipher adalah contoh sederhana dari algoritma enkripsi subtitusi abjad majemuk (Polyalphabetic Subtitution Cipher).

Vigènere Cipher diciptakan berkali-kali, namun pada awalnya diciptakan oleh Giovan Batisya Belaso. Algoritma enkripsi ini terkenal karena selain mudah dimengerti dan diimplementasikan, algoritma ini juga bagi kriptanalis pemula tampak seperti tidak dapat dipecahkan.

Vigènere Cipher mendapatkan reputasi sebagai algoritma enkripsi yang kuat. Seorang ilmuwan matematika, Charles Lutwidge Dodgson, menyebut Vigènere Cipher sebagai *unbreakable cipher* pada sebuah majalah anak-anak tahun 1868. Pada tahun 1917, ilmuwan amerika menggambarkan Vigènere Cipher dengan "*impossible of translation*". Reputasi ini lepas setelah Kasiski dengan tuntas memecahkan Vigènere Cipher pada abad ke 19, dan beberapa

kriptanalis yang memiliki kemampuan tinggi berhasil memecahkan cipher tersebut pada abad ke 16.

Vigènere Cipher digunakan oleh tentara Konfiderasi pada Perang Sipil Amerika. Perang sipil terjadi setelah Vigènere Cipher berhasil dipecahkan.

2.2.1 Metode Penerapan Algoritma Vigènere

Vigènere Cipher menggunakan Bujursangkar Vigènere untuk melakukan enkripsi. Setiap baris di dalam bujursangkar menyatakan huruf-huruf cipherteks yang diperoleh dengan Caesar Cipher. Bentuk bujur sangkar Vigènere adalah:

Plainteks

Kunci	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N

KUNCI

Jika panjang kunci lebih pendek daripada panjang plainteks, maka kunci diulang secara periodik. Bila panjang kunci adalah m, maka periodenya dikatakan m.

Contoh:

```
Kunci: sony
Plainteks: this plaintext
Kunci    : sony sonysonys
```

Maka dapat dilihat pada tabel Vigènere bahwa untuk huruf pertama, T yang berkorespondensi dengan huruf S pada kunci akan menghasilkan huruf L.

Plainteks

Kunci	A	B	C	D	E	F	G	H	I	J	K	L	M	N
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M

O	P	Q	R	S	T	U	V	W	X	Y	Z
O	P	Q	R	S	T	U	V	W	X	Y	Z
P	Q	R	S	T	U	V	W	X	Y	Z	A
Q	R	S	T	U	V	W	X	Y	Z	A	B
R	S	T	U	V	W	X	Y	Z	A	B	C
S	T	U	V	W	X	Y	Z	A	B	C	D
T	U	V	W	X	Y	Z	A	B	C	D	E
U	V	W	X	Y	Z	A	B	C	D	E	F
V	W	X	Y	Z	A	B	C	D	E	F	G
W	X	Y	Z	A	B	C	D	E	F	G	H
X	Y	Z	A	B	C	D	E	F	G	H	I
Y	Z	A	B	C	D	E	F	G	H	I	J
Z	A	B	C	D	E	F	G	H	I	J	K
A	B	C	D	E	F	G	H	I	J	K	L
B	C	D	E	F	G	H	I	J	K	L	M
C	D	E	F	G	H	I	J	K	L	M	N
D	E	F	G	H	I	J	K	L	M	N	O
E	F	G	H	I	J	K	L	M	N	O	P
F	G	H	I	J	K	L	M	N	O	P	Q
G	H	I	J	K	L	M	N	O	P	Q	R
H	I	J	K	L	M	N	O	P	Q	R	S
I	J	K	L	M	N	O	P	Q	R	S	T
J	K	L	M	N	O	P	Q	R	S	T	U
K	L	M	N	O	P	Q	R	S	T	U	V
L	M	N	O	P	Q	R	S	T	U	V	W
M	N	O	P	Q	R	S	T	U	V	W	X
N	O	P	Q	R	S	T	U	V	W	X	Y

O	P	Q	R	S	T	U	V	W	X	Y	Z
O	P	Q	R	S	T	U	V	W	X	Y	Z
P	Q	R	S	T	U	V	W	X	Y	Z	A
Q	R	S	T	U	V	W	X	Y	Z	A	B
R	S	T	U	V	W	X	Y	Z	A	B	C
S	T	U	V	W	X	Y	Z	A	B	C	D
T	U	V	W	X	Y	Z	A	B	C	D	E
U	V	W	X	Y	Z	A	B	C	D	E	F
V	W	X	Y	Z	A	B	C	D	E	F	G
W	X	Y	Z	A	B	C	D	E	F	G	H
X	Y	Z	A	B	C	D	E	F	G	H	I
Y	Z	A	B	C	D	E	F	G	H	I	J
Z	A	B	C	D	E	F	G	H	I	J	K
A	B	C	D	E	F	G	H	I	J	K	L
B	C	D	E	F	G	H	I	J	K	L	M
C	D	E	F	G	H	I	J	K	L	M	N
D	E	F	G	H	I	J	K	L	M	N	O
E	F	G	H	I	J	K	L	M	N	O	P
F	G	H	I	J	K	L	M	N	O	P	Q
G	H	I	J	K	L	M	N	O	P	Q	R
H	I	J	K	L	M	N	O	P	Q	R	S
I	J	K	L	M	N	O	P	Q	R	S	T
J	K	L	M	N	O	P	Q	R	S	T	U
K	L	M	N	O	P	Q	R	S	T	U	V
L	M	N	O	P	Q	R	S	T	U	V	W
M	N	O	P	Q	R	S	T	U	V	W	X
N	O	P	Q	R	S	T	U	V	W	X	Y

Hasil enkripsi seluruhnya adalah:

Plainteks : `THIS PLAINTEXT`
Kunci : `sony sonysonys`
Cipherteks : **`LVVQ HZNGFHRVL`**

Pada dasarnya, setiap enkripsi huruf adalah Caesar cipher dengan kunci yang berbeda-beda.

c('T') = ('T' + 's') mod 26 = L
c('H') = ('H' + 'o') ,od 26 = V, dst

2.2.2 Kelebihan Algoritma Vigènere

Pada algoritma Vigènere Cipher, huruf yang sama pada plainteks tidak selalu menjadi huruf yang sama pula dalam cipherteks. Contoh huruf plainteks T dapat dienkripsi menjadi L atau H, dan huruf cipherteks V dapat merepresentasikan H, I, dan X dalam plainteks.

.

Vigènere Cipher dapat mencegah frekuensi huruf di dalam cipherteks yang mempunyai pola tertentu yang sama seperti pada cipher abjad tunggal. Jika periode kunci diketahui dan tidak terlalu panjang, maka kunci dapat ditentukan dengan menulis program komputer untuk melakukan *exhaustive key search*.

Contoh: Diberikan cipherteks sbb:

TGCSZ GEUAA EFWGQ AHQMC

dan diperoleh informasi bahwa panjang kunci adalah *p* huruf dan plainteks ditulis dalam Bahasa Inggris, maka *running* program dengan mencoba semua kemungkinan kunci yang panjangnya tiga huruf, lalu periksa apakah hasil dekripsi dengan kunci tersebut menyatakan kata yang berarti. Cara ini membutuhkan usaha percobaan sebanyak 26^p kali.

2.2.3 Kekurangan Algoritma Vigènere

Seperti telah disebutkan sebelumnya, algoritma Vigènere Cipher sangat tangguh dan bahkan sempat diberi julukan algoritma yang tidak dapat dipecahkan. Kelemahan dari algoritma Vigènere ditemukan oleh Fridrich Kasiski pada tahun 1863. Penjelasan mengenai metode kasiski akan dijelaskan pada bagian berikut.

2.3 Metode Kasiski

Metode Kasiski membantu menemukan panjang kunci Vigènere Cipher. Metode Kasiski memanfaatkan keutungan bahasa inggris tidak hanya mengandung perulangan huruf, tetapi juga perulangan pasangan huruf atau tripel huruf seperti TH, THE, dsb. Perulangan huruf ini memungkinkan menghasilkan kriptogram yang berulang.

Contoh:
Plainteks:
`CRYPTO IS SHORT FOR CRYPTOGRAPHY`
Kunci :
`abcdab cd abcda bcd abcdabcdabcd`
Cipherteks:
`CSASTP` `KV SIQUT GQU` **`CSASTP`**`IUAQJB`

Pada contoh tersebut, CRYPTO dienkripsi menjadi kriptogram yang sama, yaitu CSATP. Hal ini dikarenakan jarak antara dua buah string yang berulang di dalam plainteks merupakan kelipatan dari panjang junci, maka string yang sama tersebut akan muncul menjadi kriptogram yangsama pula di dalam cipherteks. Tujuan dari metode Kasiski adalah mencari dua atau lebih kriptogram yang berulang untuk menentukan panjang kunci.

Langkah-langkah metode Kasiski:
1. Temukan semua kriptogram yang berulang di dalam cipherteks (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 tersebut . Nilai tersebut mungkin adalah panjang kunci. Hal ini karena *string* yang berulang dapat muncul bertindihan (*coincidence*)

Setelah panjang kunci diketahui, maka langkah berikutnya adalah menentukan kata kunci. Kata kuci dapat ditentukan dengan menggunakan *exhaustive key search*. Jika panjang kunci adalah p, maka jumlah kunci yang harus dicoba adlah 26^p. Namun akan lebih efesien bila menggunakan teknik analisi frekuensi.

Langkah-langkah untuk melakukan analisis frekuensi adalah:

1. Misalkan panjang kunci yang sudah berhasil dideduksi adalah *n*. Setiap huruf kelipatan ke-*n* pasti dienkripsi dengan huruf kunci yang sama. Kelompokkan setiap huruf ke-*n* bersama-sama sehingga kriptanalis memiliki *n* buah "pesan", masing-masing dienkripsi dengan substitusi alfabet-tunggal (dalam hal ini *Caesar cipher*).

2. Tiap-tiap pesan dari hasil langkah 1 dapat dipecahkan dengan teknik analisis frekuensi.

3. Dari hasil langkah 3 kriptanalis dapat menyusun huruf-huruf kunci. Atau, kriptanalis dapat menerka kata yang membantu untuk memecahkan cipherteks

3. Perancangan *Vigènere Biner*

Secara umum algoritma *Vigènere Biner* yang dirancang memiliki algoritma yang sama dengan algoritma Vigènere biasa. Yang membuatnya berbeda adalah bahwa pada algoritma yang dirancang ini, pesan terlebih dahulu diubah menjadi bentuk biner, kemudian proses enkripsi dilakukan terhadap pesan yang berbentuk biner.

Karena pesan yang akan dienkripsi berbentuk biner, maka tabel Vigènere yang akan digunakan juga berbeda dengan tabel Vigènere biasa. Tabel Vigènere yang digunakan dalam Vigènere Biner dapat dilihat pada Lampiran dari makalah ini.

3.1 Algoritma Enkripsi *Vigènere Biner*

Algoritma enkripsi pada Vigènere Biner adalah:

1. Buang spasi pada pesan yang akan dienkripsi
2. Tambahkan satu karakter pada bagian belakang pesan yang akan dienkripsi. Karakter ini digunakan untuk memudahkan saat melakukan dekripsi. Sebaiknya gunakan karakter yang tidak merubah arti dari pesan.
3. Ubah masing-masing huruf pada pesan yang akan dienkripsi ke dalam bentuk biner dengan lebar lima bit sesuai urutan alfabet dimana huruf A adalah 00001. Tabel urutan untuk merubah karakter dalam pesan menjadi bentuk biner dapat dilihat sebagai berikut:

a	b	c	d	e	f	g
00001	00010	00011	00100	00101	00110	00111

h	i	j	k	l	m	n
01000	01001	01010	01011	01100	01101	01110

o	p	q	r	s	t	u
01111	10000	10001	10010	10011	10100	10101

v	w	x	y	z
10110	10111	11000	11001	11010

:	.	"	,	-
11011	11100	11101	11110	11111

4. Pada masing-masing 5 bit angka biner yang merepresentasikan satu huruf, buang bit 0 yang berada didepan.
5. Sisipkan kumpulan bit 00000 diantara masing-masing angka biner yang merepresentasikan huruf pada pesan.
6. Tambahkan kumpulan bit 00000 di bagian paling belakang dari pesan.
7. Kelompokan kembali kumpulan angka biner yang ada ke dalam masing-masing memiliki lebar lima bit. Lakukan dengan merampatkan ke depan.
8. Ubah tiap karakter dalam kunci juga menjadi bentuk biner yang memiliki lebar 5 bit sesuai urutan alfabet huruf.
9. Cocokan dengan tabel Vigènere Biner dengan cara yang sama saat mencocokan pada tabel Vigènere biasa.
10. Ubah kembali bentuk biner hasil enkripsi ke dalam bentuk karakter alfabet biasa.

Contoh:
Plainteks: satu tiga lima
Kunci:informatika

1. Buang spasi dari plainteks

```
satutigalima

p: satutigalima
k: informatikai
```

2. Tambahkan satu karakter pada bagian belakang pesan

```
satutigalimaf

p: satutigalimaf
k: informatikain
```

4. Ubah ke dalam bentuk biner

10011	00001	10100	10101	10100	01001	00111
s	a	t	u	t	i	g

00001	01100	01001	01101	00001	00110
a	l	i	m	a	f

5. Hilangkan bit 0 yang berada di depan

10011	1	10100	10101	10100	1001

111	1	1100	1001	1101	1	110

6. Sisipkan dengan 00000

10011	00000	1	00000	10100	00000

10101	00000	10100	00000	1001

00000	111	00000	1	00000

1100	00000	1001	00000	1101

00000	1	00000	110	00000

7. Rampatkan ke depan

10011	00000	10000	01010	00000

01010	10000	01010	00000	01001

00000	11100	00010	00001	10000

00010	01000	00110	10000	01000	00110

8. Ubah tiap karakter kunci ke dalam bentuk biner

01001	01110	00110	01111	10010

01101	00001	10100	01001	01011

00001	01001	01110	00110	01111

10010	01101	00001	10100	01001	01011

9. Cocokan dengan tabel

11100	01110	10110	11001	10010

11110	10001	11110	01001	10100

00001	00101	10000	00111	11111

10100	10101	00111	00100	10001	10001

10. Kembalikan ke bentuk alfabetik

```
p: satutigalimaf
k: informatikain
c: .NVYR”Q”ITAEPG-TUGDQQ
```

Dapat dilihat bahwa pada hasil dari enkripsi dengan menggunakan algoritma enkripsi Vigènere Biner, jumlah karakter pada cipherteks tidak sama dengan jumlah karakter pada plainteks. Jumlah karakter pada cipher teks hampir dua kali dari jumlah karakter pada plainteks. Hal ini akan meninngkatkan keamanan pada cipherteks hasil enkripsi dengan menggunakan Vigènere Biner.

3.2 Algoritma Dekripsi *Vigènere Biner*

Algoritma dekripsi dari Vigènere Biner adalah:

1. Ubah bentuk cipherteks ke dalam bentuk biner dengan lebar masing-masing 5 bit sesuai dengan urutan alfabet dimana A = 00001.
2. Ubah bentuk kunci ke dalam bentuk biner seperti saat merubah cipherteks.
3. Cocokan dengan tabel Vigènere Biner. Lakukan seperti saat melakukan dekripsi pada algoritma Vigènere biasa.
4. Cari kelompok bit 0 yang terdiri dari lima bit. Pisahkan kelompok biner sebelum kelompok bit 0. Apabila ada kelompok bit 0 yang terdiri lebih dari 5 bit, maka ambil lima bit paling belakang, lalu pisahkan sebelum dengan kelompok bit sebelumnya.
5. Hilangkan kelompok biner yang terdiri dari 00000.
6. Tambahkan bit 0 di depan masing-masing kelompok hingga masing-masing kelompok terdiri dari 5 bit.
7. Kembalikan ke dalam bentuk alfabet sesuai dengan tabel pengubahan yang telah diberikan sebelumnya.
8. Buang satu huruf terakhir

Contoh dekripsi:

```
Cipherteks:.NVYR”Q”ITAEPG-TUGDQQ
Kunci     : informatika
```

1. Ubah bentuk cipherteks ke dalam bentuk biner.

11100	01110	10110	11001	10010

11110	10001	11110	01001	10100

00001	00101	10000	00111	11111

10100	10101	00111	00100	10001	10001

2.Ubah kunci ke dalam bentuk biner.

01001	01110	00110	01111	10010

01101	00001	10100	01001	01011

00001	01001	01110	00110	01111

10010	01101	00001	10100	01001	01011

3. Cocokan dengan tabel

10011	00000	10000	01010	00000

01010	10000	01010	00000	01001

00000	11100	00010	00001	10000

00010	01000	00110	10000	01000	00110

4. Cari kelompok bit 0 yang terdiri dari lima bit. Pisahkan kelompok biner sebelum kelompok bit 0. Apabila ada kelompok bit 0 yang terdiri lebih dari 5 bit, maka ambil lima bit paling belakang, lalu pisahkan sebelum dengan kelompok bit sebelumnya.

10011	00000	1	00000	10100	00000

10101	00000	10100	00000	1001

00000	111	00000	1	00000

1100	00000	1001	00000	1101

00000	1	00000	110

5.Hilangkan kelompok biner yang terdiri dari 00000.

10011	1	10100

10101	10100	1001

111	1

1100	1001	1101

1	110

6.Tambahkan bit 0 di depan masing-masing kelompok hingga masing-masing kelompok terdiri dari 5 bit

10011	00001	10100

10101	10100	01001

00111	00001

01100	01001	01101

00001	00110

7. Ubah kembali ke dalam bentuk alfabet

s	a	t

u	t	i

g	a

l	i	m

a	f

`Plainteks: satutigalimaf`

8. Buang satu huruf terakhir

`Plainteks: satutigalima`

Pada contoh ini, kebetulan huruf terakhir tidak menjadi masalah.

4. Pengujian Metode Kasiski Terhadap *Vigènere Biner*

Metode Kasiski membantu menemukan panjang kunci dengan bergantung pada perulangan kata, bigram, trigram, maupun susunan huruf lainnya dalam pesan. Dengan menggunakan algoritma Vigènere Biner, kemungkinan untuk adanya perulangan menjadi kecil sekali.

Hal ini disebabkan karena satu huruf pada plainteks tidak diubah menjadi satu blok

kelompok bilangan biner, melainkan satu atau lebih blok bilangan biner. Dan pencocokan terhadap tabel juga bukan terhadap huruf, melainkan terhadap blok kelompok bilangan biner yang telah diolah sebelumnya. Sedangkan satu huruf pada kunci tetap menjadi satu blok kelompok bilangan biner. Hal ini menyebabkan perulangan dengan korespondensi huruf pada kunci menjadi kecil sekali.

Karena itu, dengan metode Kasiski pun sangat sulit untuk dapat menemukan panjang kunci yang digunakan untuk melakukan enkripsi.

5. Kesimpulan

Algoritma Vigènere biasa merupakan algoritma yang sangat kuat sebelum dipecahkan oleh Kasiski. Oleh karena itu, dengan menemukan sebuah algoritma berdasarkan pada Vigènere yang diolah sedemikian rupa hingga menyulitkan menemukan panjang kunci dengan menggunakan metode Kasiski akan membuat algoritma tersebut menjadi sebuah algoritma enkripsi yang kuat.

Algoritma Vigènere Biner merupakan perluasan dari algoritma Vigènere biasa. Dengan algoritma Vigènere Biner, pencarian panjang kunci dengan metode Kasiski menjadi sulit. Oleh karena itu dapat dikatakan algoritma enkripsi Vigènere Biner adalah algoritma enkripsi yang kuat.

Kekurangan yang dimiliki oleh algoritma Vigènere Biner adalah bertambah besarnya hasil enkripsi pesan menjadi hampir dua kali lipat. Hal ini akan menghabiskan memori komputer dalam penyimpanan cipherteks.

Namun, bila plainteks yang akan dienkripsi tidak terlalu panjang, maka bertambahnya besar cipherteks merupakan suatu keuntungan. Karena dengan bertambah besarnya hasil enkripsi, maka orang yang melihat cipherteks tidak akan mengetahui panjang sebenarnya dari plainteks yang dienkripsi.

Karena keuntungan tersebut, maka algoritma ini akan semakin kuat bila digunakan untuk melakukan enkripsi terhadap pesan yang tidak terlalu panjang. Walaupun untuk pesan yang panjang juga bisa dan mendapatkan keuntungan serupa, kerugian cipherteks yang memakan memori komputer menjadi kekurangan algoritma ini.

DAFTAR PUSTAKA

[1] Munir, Rinaldi, *Diktat Kuliah IF5054 Kriptografi Program Studi Teknik Informatika Institut Teknologi Bandung*, 2006.

[2] *http://en.wikipedia.org*
Tanggal akses : 24 september 2006 pukul 11:40 .

[3] Mengenal Teknik Kriptografi. Sriwijaya Post. Minggu, 26 maret 2006 hal 7 bagian hitech.

[4] *http://www.simonsingh.net*
Tanggal akses 10 Oktober 2006 pukul 15:08

[5] Hidayat, Taufik. Sistem Kriptografi Idea. S2-TI-ITB.

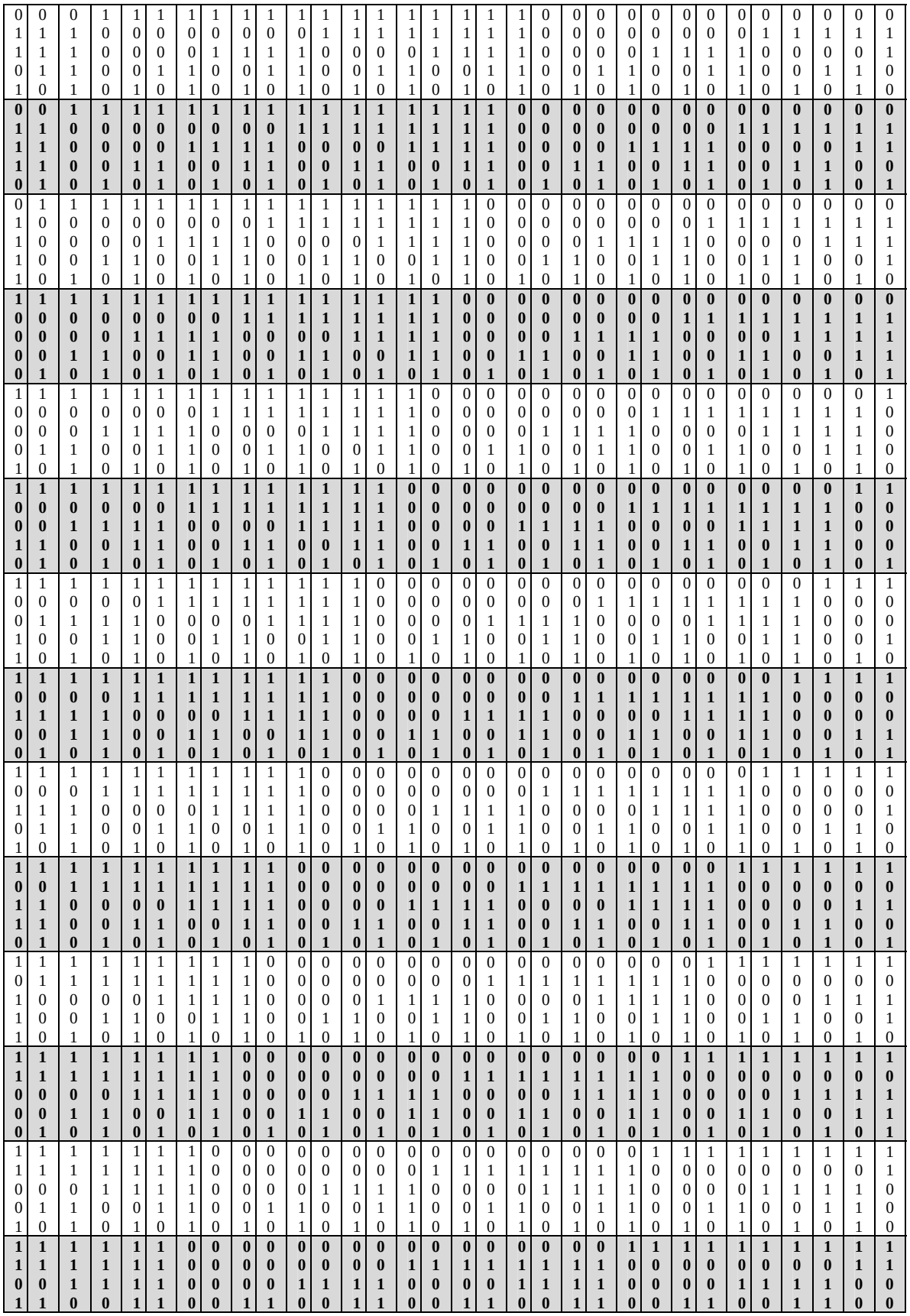
[6] *http://hadiwibowo.wordpress.com/*
Tanggal akses: 24 september 2006 pukul 12:05.

[7]http://www.aegeanparkpress.com/books_by_author.html#N.
Tanggal akses: 11 Oktober 2006 pukul 17:00.

LAMPIRAN

Tabel vigenere biner :

00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111
00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000
00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001
00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010
00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011
00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100
00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101
00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110
01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111
01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111	01000
01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001
01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010
01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011



0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	1	0
0	1	1	1	1	0	0	0	0	1	1	0	1	1	1	0	0	1	0	1	1	1	1	0	0	1	0	0	1	1
1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1	1	0	0	1	0	1	0	1	1	0	1	0	1	0
1	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
1	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
1	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	0	1	1	0	1
1	1	1	1	0	0	0	0	1	1	0	1	1	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	1
1	1	1	1	0	0	0	1	1	0	1	1	0	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1	0	0	1	1
1	0	0	1	0	1	0	1	1	0	0	1	1	0	1	1	1	0	0	0	1	0	1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1	0	0	1	1
1	0	0	1	0	1	0	1	1	0	0	1	1	0	1	1	1	0	0	0	1	0	1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0