STUDI MENGENAI APLIKASI TEORI *QUASIGROUP* DALAM KRIPTOGRAFI

Fajar Yuliawan – NIM: 13503022

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13022@students.if.itb.ac.id

Abstrak

Beberapa teori dalam aljabar abstrak sering digunakan dalam teori pengkodean dan kriptografi, salah satunya adalah teori tentang *quasigroup*. *Quasigroup* dapat didefinisikan sebagai himpunan berhingga *Q* bersama-sama dengan operasi biner $*$ pada *G* dimana invers kiri dan kanan selalu ada dan tunggal. Dengan demikian, persamaan dalam operasi biner tersebut selalu memiliki solusi tunggal. Sifat inilah yang digunakan untuk membangun proses enkripsi dengan menggunakan *quasigroup*. Proses enkripsi dilakukan dengan melakukan operasi biner $*$ terhadap anggota-anggota *quasigroup* tersebut dan proses dekripsi dilakukan dengan menggunakan inversnya, bisa menggunakan invers kiri maupun invers kanan.

Dalam teori kombinatorika, *quasigroup* direpresentasikan dengan sebuah *Latin square*, yaitu sebuah matriks berukuran $n \times n$ dimana setiap baris dan kolom adalah permutasi dari anggota-anggota quasigroup. Tabel yang digunakan dalam *Vigenere cipher* termasuk salah bentuk *Latin square*, dengan huruf-huruf alfabet sebagai anggota-anggota quasigroup. Dengan menggunakan *Latin square* secara umum, algoritma kriptografi *Vigenere cipher* dapat dibuat lebih *general*. Hal ini dapat mempersulit proses kriptanalis pada algoritma *Vigenere cipher*.

Enkripsi dan dekripsi dengan menggunakan *quasigroup* dapat bekerja pada mode karakter dan mode bit. Pada mode karakter contohnya adalah *Vigenere cipher* dan bentuk umumnya. Dengan mengubah masukan dari karakter menjadi blok-blok bit, maka fungsi enkripsi dan dekripsi dengan menggunakan *quasigroup* akan bekerja pada mode blok. Dengan demikian, algoritma kriptografi dengan menggunakan *quasigroup* dapat dikombinasikan dengan mode operasi *cipher* blok, yaitu *Electronic Cipher Block* (*ECB*), *Cipher Block Chaining* (*CBC*), *Cipher Feedback* (*CFB*) dan *Output-Feedback* (*OFB*).

Pada makalah ini, akan dijelaskan mengenai aplikasi teori quasigroup tersebut dalam kriptografi. Hal ini mencakup penjelasan mengenai *quasigroup* dan proses enkripsi dan dekripsi dengan menggunakan *quasigroup*.

Keywords: *Algebra-cryptography, cipher* blok, dekripsi, enkripsi, *quasigroup, Latin squar*, *Electronic Code Book, Cipher Block Chaining, Cipher Feedback, Output-Feedback, Vigenere Cipher*

1 Pendahuluan

Dalam aljabar abstrak, teori *quasigroup* telah dikembangkan sangat luas. Banyak diantara teori-teori tersebut yang telah diaplikasikan dalam berbagai bidang kriptologi, steganografi dan teori pengkodean. Salah satunya dalam hal enkripsi dan dekripsi. Enkripsi adalah proses penyandian pesan menjadi bentuk lain yang tidak dimengerti maknanya. Pesan disebut juga plainteks dan bentuk lain dari pesan yang tidak dimengerti maknanya disebut cipherteks. Pesan atau plainteks dienkripsi karena pengirim pesan tidak ingin pesan yang dikirimkannya dibaca atau dimanipulasi oleh orang selain penerima pesan. Setelah pesan dienkripsi dan dikirimkan kepada penerima pesan, cipherteks hasil enkripsi tersebut harus dapat dikembalikan menjadi pesan asli yang dimengerti maknanya. Proses ini disebut sebagai dekripsi. Parameter yang digunakan dalam proses enkripsi dan dekripsi ini adalah sebuah string yang disebut kunci.

Misalkan P menyatakan plainteks, C cipherteks dan K kunci. Fungsi enkripsi dinyatakan sebagai E_K dan fungsi dekripsi sebagai D_K. Kedua fungsi ini memenuhi persamaan

$$C = E(P) \text{ dan } P = D(C)$$

sehingga hubungan berikut juga berlaku

$$D(E(P)) = P$$

yang artinya, proses dekripsi harus mengembalikan plainteks seperti semula. Jika persamaan-persamaan tersebut tidak dipenuhi, maka algoritma enkripsi dan dekripsinya merupakan algoritma yang salah dan tidak dapat digunakan.

Salah aplikasi teori *quasigroup* dalam kriptografi adalah *Vigenere cipher* dan bentuk umumnya. Algoritma *Vigenere cipher* menggunakan sebuah bujursangkar yang berisi karakter-karakter dalam alfabet. Proses enkripsi dan dekripsi dilakukan dengan menggunakan bujursangkar tersebut. Sebeneranya, bujursangkar yang digunakan dalam algoritma *Vigenere cipher* merupakan salah satu bentuk dari *Latin Square* yang merepresentasikan sebuah *quasigroup*. Dengan menggunakan *Latin square* yang lebih umum atau *quasigroup* yang lebih umum, algoritma *Vigenere Cipher* dapat dibuat lebih general. Dengan membuat bentuk *Vigenere cipher* yang lebih umum dengan kriptanalisis akan menjadi lebih sulit. Hal ini berarti, dengan menggunakan dasar yang sederhana seperti pada *Vigenere cipher*, bisa didapatkan keamanan yang cukup tinggi. Hal ini sangat berguna ketika diaplikasikan dalam suatu lingkungan yang membutuhkan memori kecil dan proses yang cepat, misalnya untuk aplikasi *mobile* seperti enkripsi SMS dengan menggunakan enkripsi quasigroup.

Selain *Vigenere cipher*, ada beberapa algoritma enkripsi dan dekripsi lain yang menggunakan teori *quasigroup*. Algoritma tersebut memanfaatkan sifat *quasigroup*, yaitu operasi biner $*$ pada *quasigroup* selalu memiliki invers kanan dan kiri yang tunggal. Dengan demikian, proses enkripsi dapat menggunakan kombinasi dari operasi biner tersebut, kemudian dekripsi dengan menggunakan invers dari operasi tersebut, baik invers kiri maupun invers kanan.

Algoritma kriptografi dengan menggunakan *quasigroup* ini dapat bekerja pada mode karakter dan mode bit. Pada mode karakter contohnya adalah *Vigenere cipher* dan bentuk umumnya. Untuk mengubah algoritma ini agar bekerja pada mode bit, cukup mengubah masukan yang diterima saja. Dengan menggunakan mode bit, maka algoritma ini akan melakukan proses enkripsi dan dekripsi pada sekumpulan blok-blok bit. Dengan demikian, algoritma dengan menggunakan *quasigroup* dapat dikombinasikan dengan mode operasi pada *cipher* blok, yaitu *Electronic Code Book* (*ECB*), *Cipher Block Chaining* (*CBC*), *Cipher Feedback* (*CFB*) dan *Output-Feedback* (*OFB*). Hal ini akan membuat algoritma kriptografi yang dihasilkan menjadi lebih kuat, sehingga proses kriptanalisis menjadi lebih sulit.

2 Landasan Matematika Quasigroup

Ada beberapa landasan matematika yang harus diketahui sebelum membahas mengenai aplikasi *quasigroup* dalam kriptografi. Hal ini antara lain tentang *grupoid, quasigroup* sendiri dan *Latin square*.

2.1 Groupoid

Sebuah *groupoid* adalah sebuah himpunan berhingga Q bersama-sama dengan sebuah operasi biner $*$ pada Q yang memenuhi $a * b \in Q$ untuk semua $a,b \in Q$. Dengan kata lain, Q tertutup terhadap operasi $*$. Sebuah *grupoid* yang memiliki n anggota dapat dinyatakan sebagai sebuah matriks $n \times n$ yang anggotanya merupakan anggota *grupoid* tersebut. Misalnya sebuah *grupoid* yang memiliki anggota {1,2,3,4} dapat dinyatakan dengan matriks suatu matriks M sebagai berikut

$$\begin{pmatrix} 1 & 3 & 2 & 4 \\ 2 & 1 & 3 & 4 \\ 3 & 4 & 1 & 2 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Dari baris pertama matriks di atas, dapat diketahui bahwa *grupoid* di atas memenuhi

$$1 * 1 = 1,\ 1 * 2 = 3,\ 1 * 3 = 2, \text{ dan } 1 * 4 = 4.$$

Selanjutnya dari baris keduanya, dapat diketahui bahwa

$$2 * 1 = 2,\ 2 * 2 = 1,\ 2 * 3 = 3,\ 2 * 4 = 5,$$

dan seterusnya. Dari kolom pertama matriks tersebut, dapat diketahui juga bahwa

$$1 * 1 = 1,\ 2 * 1 = 2,\ 3 * 1 = 3 \text{ dan } 4 * 1 = 1,$$

dan seterusnya.

Secara umum, jika bilangan-bilangan 1,2,...,*n* diasosiasikan dengan setiap anggota *grupoid* a_1, a_2,..., a_n, maka matriks *M* yang merupakan representasi *grupoid* tersebut memiliki anggota

$$M[i,j] = a_i * a_j.$$

Demikian juga sebaliknya. Jika diketahui representasi matriks dari sebuah *grupoid*, maka hasil operasi tiap anggota *grupoid* tersebut juga dapat diketahui.

Satu hal yang perlu diperhatikan adalah *grupoid* tidak memeiliki aksioma apapun selain sifat ketertutupan. Operasi pada *grupoid* tidak harus memiliki sifat asosiatif, komutitatif maupun distributif. Dengan demikian, sembarang matriks *M* berukuran $n \times n$ selalu dapat menyatakan *grupoid* dengan anggota sebanyak *n* dan anggotanya sama dengan anggota matriks *M* atau merupakan superset dari anggota-anggota matriks *M*. Sebagai contoh satu lagi, matrks

$$\begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 2 & 1 & 1 \end{pmatrix}$$

dapat menyatakan sebuah *grupoid* dengan anggota 1,2,3,4 (walaupun di dalam matriks tersebut tidak mengandung angka 3 dan 4).

2.2 Quasigroup

Sebuah *quasigroup Q* adalah *grupoid* yang memiliki invers kiri dan kanan, yaitu untuk setiap $u, v \in Q$ ada $x, y \in Q$ yang tunggal sehingga $x * u = v$ dan $u * y = v$. Hal ini berarti juga bahwa operasi pada *quasigroup* dapat dibalik dan memiliki solusi tunggal. Dengan demikian, kedua operasi invers terhadap operasi pada *quasigroup* dapat didefinisikan, yaitu invers kiri (*left inverse*) dan invers kanan (*right inverse*). Notasi invers kiri adalah \ dan notasi untuk invers kanan adalah /. Karena ketunggalan ini, daapt juga dikatakan bahwa operator \ bersama-sama dengan himpunan berhingga *Q* mendefinisikan sebuah *quasigroup* (*Q*, \) dan untuk aljabar (*Q*, \, $*$). Demikian halnya dengan operator /. Dalam hal ini, berlaku persamaan

$$x * (x \setminus y) = y = x \setminus (x * y).$$

2.3 Latin Squares

Bagaimana representasi matriks dari sebuah *quasigroup*? Pada bagian sebelumnya telah dijelaskan bahwa *quasigroup* adalah *grupoid* dengan satu aksioma tambahan, yaitu adanya invers kiri dan kanan yang tunggal. Dengan demikian, representasi matriks dari sebuah *quasigroup* juga harus memiliki sebuah sifat tambahan, yaitu setiap kolom dan baris pada matriks harus merupakan permutasi dari angota-anggota *quasigroup*. Dengan kata lain, setiap kolom dan baris pada matris harus mengandung semua anggota-anggota *quasigroup*. Dalam kombinatorika, matriks semacam ini disebut juga sebagai *Latin Square*. Contoh sebuah *latin square* yang paling sederhana adalah sebagai berikut

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

Dari matriks tersebut terlihat bahwa setiap kolom dan baris merupakan permutasi dari 1,2,3,4. Kedua matriks pada bagian 2.1 di atas tidak merepresentasikan sebuah *quasigroup*. Matriks pertama pada bagian 2.1 mengandung tiga buah angka 4 dan sebuah angka 2 pada kolom keempat dan matriks kedua pada bagian 2.1 tidak mengandung angka 3 dan 4 sama sekali.

Pehatikan matriks pada contoh tersebut. Pada baris ketiga terlihat bahwa

$$1 * 1 = 3,\ 1 * 2 = 4,\ 1 * 3 = 1 \text{ dan } 1 * 4 = 2.$$

Dengan demikian,

$$1 \setminus 3 = 1,\ 1 \setminus 4 = 2,\ 1 \setminus 1 = 3 \text{ dan } 1 \setminus 2 = 4,$$

dan seterusnya.

Contoh lain sebuah *Latin Squares* yang lebih rumit adalah sebagai berikut

$$\begin{pmatrix} 1 & 4 & 0 & 6 & 2 & 7 & 3 & 5 \\ 2 & 7 & 3 & 5 & 1 & 4 & 0 & 6 \\ 5 & 0 & 4 & 2 & 6 & 3 & 7 & 1 \\ 4 & 1 & 5 & 3 & 7 & 2 & 6 & 0 \\ 7 & 2 & 6 & 0 & 4 & 1 & 5 & 3 \\ 0 & 5 & 1 & 7 & 3 & 6 & 2 & 4 \\ 3 & 6 & 2 & 4 & 0 & 5 & 1 & 7 \\ 6 & 3 & 7 & 1 & 5 & 0 & 4 & 2 \end{pmatrix}$$

3 Aplikasi *Quasigroup* dalam *Kriptografi*

Pada bagian sebelumnya telah dijelaskan mengenai dasar teori *quasigroup*. *Quasigroup* memiliki sifat yang diperlukan dalam proses enkripsi dan dekripsi, yaitu invers yang tunggal. Dengan adanya invers yang tunggal tersebut, plainteks yang telah dienkripsi dengan menggunakan operasi pada *quasigroup* selalu dapat dikembalikan dengan proses dekripsi menggunakan invers dari operasi pada *quasigroup*. Secara matematis, hal ini dapat dinyatakan sebagai

$$D(E(P)) = P,$$

dimana D dan E masing-masing menyatakan fungsi-fungsi dekripsi dan enkripsi. Dengan menggunakan parameter tambahan berupa kunci K, persamaan yang baru dapat dituliskan sebagai

$$D_K(E_K(P)) = P.$$

3.1 Vigenere Cipher

Vigenere Cipher pertama kali dipublikasikan pada tahun 1856. Tetapi algoritma tersebut mulai dikenal luas 200 tahun kemudian dan berhasil dipecahkan oleh Babbage dan Kasiski pada pertengahan Abad 19. *Vigenere Cipher* digunakan oleh tentara Konfederasi (*Confederate Army*) pada Perang Sipil Amerika (*American Civil War*). Perang Sipil terjadi setelah *Vigenere Cipher* berhasil dipecahkan.

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

Gambar 1 Bujursangkar *Vigenere* dengan 26 karakter

Hal ini diilustrasikan oleh kutipan pernyataan Jendral Ulysses S. Grant: "*It would sometimes take too long to make translation of intercepted dispatches for us to receive any benefits from them, but sometimes they gave useful information*"

Walaupun sebenernya *Vigenere Cipher* tidak diturunkan dari teori *quasigroup* (*Vigenere Cipher* sudah ada sebelum ditemukannya aplikasi-aplikasi teori *quasigroup* dalam kriptografi), tetapi *Vigenere Cipher* tetap dapat dipandang sebagai sebuah aplikasi dari teori *quasigroup*.

Vigenere Cipher menggunakan bujursangkar *Vigenere* untuk melakukan enkripsi (perhatikan gambar 1). Kolom paling kiri dari bujursangkar menyatakan huruf-huruf kunci, sedangkan baris paling atas menyatakan huruf-huruf plainteks. Setiap baris di dalam bujursangkar menyatakan huruf-huruf cipherteks yang diperoleh dengan *Caesar Cipher*, yang mana jauh pergeseran huruf plainteks ditentukan oleh nilai desimal oleh huruf kunci tersebut (di sini A = 0, B = 1, C = 2, ..., Z = 25). Sebagai contoh, huruf kunci C (= 2) menyatakan huruf plainteks digeser sejauh 2 huruf ke kanan (dari susunan alfabetnya).

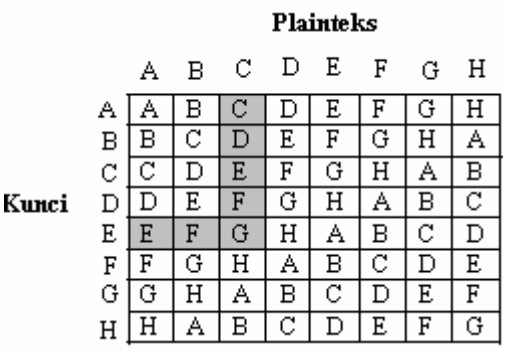
Bujursangkar Vigenere digunakan untuk memperoleh cipherteks dengan menggunakan kunci yang sudah ditentukan. Jika panjang kunci lebih pendek daripada panjang plainteks, maka kunci diulang penggunaannya (sistem periodik). Bila panjang kunci adalah *m*, maka periodenya dikatakan *m*. Sebagai contoh, jika plainteks LIMA JUTA RUPIAH dan kunci adalah UANG, maka penggunaan kunci sebagai periodik adalah sebagai berikut:

Plainteks: `LIMA JUTA RUPIAH`
Kunci: `UANG UANG UANGUA`

Setiap huruf plainteks akan dienkripsi dengan setiap huruf di bawahnya.

Untuk melakukan enkripsi dengan *Vigenere Cipher*, lakukan pada Bujursangkar *Vigenere* sebagai berikut: tarik garis vertikal dari huruf plainteks ke bawah, lalu tarik garis mendatar dari huruf kunci ke kanan. Perpotongan kedua garis tersebut menyatakan huruf cipherteksnya.

Sebagai contoh, akan digunakan tabel yang lebih kecil (untuk menyederhanakan penulisan), misalnya hanya terdiri dari 8 alfabet: A, B, C, D, E, F, G, H. Plainteks yang akan dienkripsi adalah CGEH FGEH HAB dan kunci yang digunakan adalah ECFGC.



Gambar 2 Enkripsi Huruf C dengan kunci E

Plainteks: `CGEH FGEH HAB`
Kunci: `ECFG CECF GCE`

Untuk plainteks tersebut, tarik garis vertikal dari huruf C dan tarik garis mendatar dari huruf E, perpotongannya adalah kotak dengan huruf G (perhatikan Gambar 2). Dengan kata lain, huruf plainteks C dienkripsikan oleh huruf kunci E menghasilkan huruf cipherteks G. Dengan cara yang sama, tarik garis vertikal dari huruf G dan tarik garis mendatar dari huruf C, perpotongannya adalah huruf A. Lakukan langkah ini berulang-ulang sampai akhir plainteks. Hasil enkripsi seluruhnya adalah sebagai berikut:

Plainteks: `CGEH FGEH HAB`
Kunci: `ECFG CECF GCE`
Cipherteks: `GABF HCGE FCF`

Perhatikan bahwa G dapat dienkripsi menjadi huruf A dan C, dan huruf cipherteks F dapat merepresentasikan huruf plainteks H dan B. Hal tersebut merupakan karakteristik dari *cipher* abjad-majemuk. Pada *cipher* substitusi sederhana, setiap huruf cipherteks selalu menggantikan huruf plainteks tertentu, sedangkan pada *cipher* abjad majemuk, setiap huruf cipherteks dapat memiliki kemungkinan banyak huruf plainteks. Jadi dengan menggunakan *Vigenere Cipher*, kita dapat mencegah frekuensi huruf-huruf di dalam cipherteks yang memiliki pola tertentu yang sama sebagaimana yang diperlihatkan pada *cipher* substitusi sederhana (*cipher* abjad-tunggal).

Dekripsi pada *Vigenere Cipher* dilakukan dengan cara yang berkebalikan, yaitu menarik garis mendatar dari huruf kunci sampai ke huruf cipherteks yang dituju, lalu dari huruf cipherteks tarik garis vertikal ke atas sampai ke huruf plainteks.

Untuk memecahkan *Vigenere Cipher*, cukup menentukan kuncinya. Jika periode kunci diketahui dan tidak terlalu panjang, maka kunci dapat ditentukan dengan menulis program komputer untuk melakukan *exhaustive key search*. Salah satu metode yang sering digunakan untuk menentukan panjang kunci pada *Vigenere Cipher* adalah Metode *Kasiski*.

Sekarang perhatikan Bujursangkar *Vigenere* pada gambar 1 dan gambar 2. Perhatikan bahwa pada gambar 1, setiap baris dan kolom pada bujursangkar tersebut merupakan permutasi dari alfabet A, B, C, ..., Z. Demikian juga dengan gambar 2 yang merupakan permutasi dari alfabet A, B, C, D, E, F, G, H. Hal ini merupakan karakteristik dari *Latin Square*. Dan seperti telah dibahas sebelumnya, sebuah *Latin Square* merupakan representasi dari suatu *quasigroup.*

Quasigroup apakah yang direpresentasikan oleh *Vigenere Cipher*? Kita dapat mendefinisikan *quasigroup* berikut untuk merepresentasikan *Vigenere Cipher*: elemen dari *quasigroup* adalah alfabet dan operasinya ditentukan dengan cara yang sama seperti proses enkripsi tiap huruf pada *Vigenere Cipher*, yaitu menarik garis vertikal dari huruf plainteks dan garis huruf mendatar dari huruf kunci, perpotongannya adalah hasil operasi. Sebagai contoh, pada gambar 2, dapat disimpulkan bahwa

$$C * E = G,\ G * C = A,\ E * F = B$$

dan seterusnya. Secara matematis, proses enkripsi dinyatakan sebagai berikut

$$C_i = E_K(P_i) = K_i * P_i$$

dimana P_i, C_i dan K_i berturut-turut menyatakan karakter ke-i pada plainteks, cipherteks dan kunci. Dan seperti penjelasan sebelumnya, jika panjang kunci lebih kecil daripada panjang plainteks, maka kunci diperpanjang (agar sama panjang dengan plainteks) dengan cara mengulang-ulang kunci tersebut. Proses dekripsi juga dapat dinyatakan secara matematis (dalam notasi *quasigroup*) sebagai berikut

$$P_i = D_K(C_i) = K_i \setminus C_i$$

Karena operasi invers kiri pada *quasigroup* bersifat tunggal, maka hasil operasi $K_i \setminus C_i$ juga pasti hanya satu kemungkinan. Tidak mungkin ada dua atau lebih kemungkinan huruf plainteks. Inilah yang dimaksud dengan ketunggalan invers pada *quasigroup* bermanfaat pada proses enkripsi dan dekripsi.
Jika huruf-huruf pada alfabet diganti dengan angka-angka dari 0 sampai 25 (A = 0, B = 1, C = 2, ..., Z = 25), maka operasi pada *quasigroup* yang merepresentasikan *Vigenere Cipher* juga dapat dinyatakan secara matematis, yaitu

$$x * y = x + y \bmod n,$$

dimana n menyatakan banyaknya alfabet yang digunakan, misalnya untuk alfabet latin 26.

Sebagai contoh pada alfabet dengan 8 huruf pada gambar 2, berlaku

$$C * E = 2 * 4 = 2 + 4 \bmod 26 = 6 = \text{G}$$
$$G * C = 6 * 2 = 6 + 2 \bmod 26 = 8 = \text{A}$$

dan seterusnya.
Dengan menggunakan notasi yang sama (huruf-huruf alfabet yang diganti dengan angka), proses pencarian invers kiri juga dapat dinyatakan secara matematis, yaitu

$$x \setminus y = y - x \bmod n,$$

dimana n menyatakan banyaknya alfabet yang digunakan.
Dengan demikian, proses enkripsi dan dekripsi pada *Vigenere Cipher* dapat dituliskan juga sebagai

$$C_i = E_K(P_i) = K_i * P_i = K_i + P_i \bmod 26$$

dan

$$P_i = D_K(C_i) = K_i \setminus C_i = C_i - K_i \bmod 26$$

Sekarang perhatikan bahwa *Vigenere Cipher* merupakan bentuk *quasigroup* paling sederhana. Dengan menggunakan bentuk *quasigroup* lain, *Vigenere Cipher* dapat digeneralisasikan menjadi berbagai macam. Bahkan ada kemungkinan hal ini membuat proses kriptanalis menjadi lebih sulit. Hal ini disebabkan *Vigenere Cipher* memiliki satu karakteristik khusus, yaitu bujursangkar *Vigenere* simetris terhadap diagonal yang ditarik dari kiri atas tabel sampai ke kanan bawah tabel. Dari sudut pandang *quasigroup*, hal ini dinyatakan dengan operasi pada *quasigroup* yang komutatif, yaitu

$$x * y = y * x, \text{ untuk semua } x,y \in Q$$

Jika sifat komutatif pada *quasigroup Vigenere Cipher* ini dihilangkan, ada kemungkinan proses kriptanalis menjadi lebih sulit. Namun hal ini belum diteliti lebih jauh karena terbatasnya waktu yang ada. Untuk menghilangkan sifat komutatif pada *quasigroup Vigenere Cipher* ini, bisa digunakan *quasigroup* lain yang representasi *Latin Square*-nya tidak simetris terhadap

diagonal yang ditarik dari kiri atas matriks sampai ke kanan bawah matriks.

Sebagai contoh, *Vigenere Cipher* diaplikasikan dengan perubahan bujursangkar sebagai berikut (seperti sebelumnya, hanya digunakan alfabet 8 karakter untuk menyederhanakan persoalan)

Kunci \ Plainteks	A	B	C	D	E	F	G	H
A	B	E	A	G	C	H	D	F
B	C	H	D	F	B	E	A	G
C	F	A	E	C	G	D	H	B
D	E	B	F	D	H	C	G	A
E	H	C	G	A	E	B	F	D
F	A	F	B	H	D	G	C	E
G	D	G	C	E	A	F	B	H
H	G	D	H	B	F	A	E	C

Gambar 3 Generalisasi Bujursangkar *Vigenere*

Jika huruf-huruf pada alfabet tersebut digant dengan angka 0 sampai 7, maka didapatkan matriks *Latin Square* yang berkorespondensi dengan Bujursangkar *Vigenere* tersebut, yaitu

$$\begin{pmatrix} 1 & 4 & 0 & 6 & 2 & 7 & 3 & 5 \\ 2 & 7 & 3 & 5 & 1 & 4 & 0 & 6 \\ 5 & 0 & 4 & 2 & 6 & 3 & 7 & 1 \\ 4 & 1 & 5 & 3 & 7 & 2 & 6 & 0 \\ 7 & 2 & 6 & 0 & 4 & 1 & 5 & 3 \\ 0 & 5 & 1 & 7 & 3 & 6 & 2 & 4 \\ 3 & 6 & 2 & 4 & 0 & 5 & 1 & 7 \\ 6 & 3 & 7 & 1 & 5 & 0 & 4 & 2 \end{pmatrix}$$

Perhatikan bahwa Bujursangkar *Vigenere Cipher* pada gambar di atas tidak simetris terhadap diagonal yang ditarik dari sudut kiri atas ke kanan bawah.

Proses enkripsi dan dekripsi dengan menggunakan bujursangkar di atas bisa mengadopsi cara yang digunakan pada Bujursangkar *Vigenere* yang asli, yaitu menarik garis vertikal dari huruf plainteks dan garis horisontal dari huruf kunci untuk mendapatkan sebuah huruf cipherteks.

Kita gunakan contoh yang sama ketika mengenkripsikan pesan plainteks CGEH FGEH HAB dengan kunci ECFGC.

Kunci \ Plainteks	A	B	C	D	E	F	G	H
A	B	E	A	G	C	H	D	F
B	C	H	D	F	B	E	A	G
C	F	A	E	C	G	D	H	B
D	E	B	F	D	H	C	G	A
E	H	C	G	A	E	B	F	D
F	A	F	B	H	D	G	C	E
G	D	G	C	E	A	F	B	H
H	G	D	H	B	F	A	E	C

Gambar 4 Enkripsi huruf C dengan kunci E

Plainteks: `CGEH FGEH HAB`
Kunci: `ECFG CECF GCE`

Untuk plainteks tersebut, tarik garis vertikal dari huruf C dan tarik garis mendatar dari huruf E, perpotongannya adalah kotak dengan huruf G (perhatikan Gambar 4). Dengan kata lain, huruf plainteks C dienkripsikan oleh huruf kunci E menghasilkan huruf cipherteks G. Dengan cara yang sama, tarik garis vertikal dari huruf G dan tarik garis mendatar dari huruf C, perpotongannya adalah huruf H. Lakukan langkah ini berulang-ulang sampai akhir plainteks. Hasil enkripsi seluruhnya adalah sebagai berikut:

Plainteks: `CGEH FGEH HAB`
Kunci: `ECFG CECF GCE`
Cipherteks: `GHDH DFGE HFC`

Proses dekripsi dilakukan dengan cara yang juga dengan versi *Vigenere Cipher* yang asli.

3.2 Algoritma Enkripsi Lain dengan Quasigroup

Selain *Vigenere Cipher*, ada banyak cara lain yang dapat dilakukan untuk melakukan enkripsi dengan menggunakan *quasigroup* ini.

Langkah-langkah yang dilakukan dalam mendefinisikan algoritma baru ini adalah dengan mendefinisikan *quasigroup* yang akan dipergunakan terlebih dahulu. Kemudian mendesain proses enkripsi yang terdiri dari operasi biner $*$ pada *quasigroup* dan proses dekripsi dengan menggunakan invers kiri atau invers kanan.

Sekarang kita definisikan fungsi enkripsi E sebagai berikut

$$E(p_1 p_2 ... p_n) = c_1 c_2 ... c_n ,$$

dimana

$$\begin{cases} c_1 = l * p_1 \\ c_{i+1} = c_i * p_{i+1} \end{cases}$$

untuk i = 1, 2, ..., n. Dalam hal ini, $p_1p_2...p_n$ adalah plainteks dan $c_1c_2...c_n$ adalah cipherteks dengan $p_1, p_2, ..., p_n$, $c_1, c_2, ..., c_n$ menjadi anggota-anggota pada *quasigroup* yang telah didefinisikan sebelumnya. Dalam proses enkripsi tersebut, belum ditambahkan parameter kunci. Sebagai gantinya, digunakan nilai l yang disebut sebagai *leader* pada proses enkripsi tersebut. Karena menggunakan parameter l, fungsi enkripsi juga sering dituliskan sebagai E_l. Perhatikan juga bahwa transformasi E_l tersebut dapat dipandang sebagai pemetaan $Q^+ \rightarrow Q^+$.

Dengan menggunakan algoritma enkripsi seperti itu, proses dekripsi bisa dilakukan secara langsung dengan mengaplikasikan invers kiri pada *quasigroup*, yaitu

$$D_l(c_1c_2...c_n) = p_1p_2...p_n,$$

dimana

$$\begin{cases} p_1 &= l \setminus c_1 \\ p_{i+1} &= c_i \setminus c_{i+1} \end{cases}$$

untuk i = 1, 2, ..., n.

Dalam algoritma tersebut, nilai dari $c_i * p_{i+1}$ maupun $c_i \setminus c_{i+1}$ dapat dicari dengan menelusuri matriks *Latin Square* yang merepresentasikan *quasigroup* yang digunakan. Hal ini sama seperti mencari huruf plainteks dan cipherteks pada *Vigenere Cipher*.

Namun algoritma enkripsi seperti di atas bisa dikatakan sangat lemah atau bahkan tidak berguna sama sekali. Karena jika sudah diketahui *quasigroup* atau *Latin Square* yang digunakan untuk proses enkripsi, maka sebanyak $n - 1$ karakter plainteks bisa diketahui dengan mengaplikasikan invers kiri pada *cipherteks* (dari persamaan $p_{i+1} = c_i \setminus c_{i+1}$). Sedangkan nilai p_1 bisa diketahui dengan sangat mudah menggunakan *exhaustive key search* sebanyak n kali. Satu-satunya cara untuk menangani hal semacam ini adalah dengan menyembunyikan *quasigroup* maupun *Latin Squares* yang digunakan pada proses enkripsi. Dengan kata lain, *quasigroup* dan *Latin Squares* bertindak sebagai *secret key* pada proses enkripsi. Namun hal ini tidak realistis, karena untuk mengenkripsikan huruf-huruf pada alfabet minimal diperlukan *quasigroup* dengan 26 anggota (yaitu huruf-huruf A, B, C, ..., Z) dan matriks *Latin Squares* yang berukuran 26×26. Hal ini hampir mustahil untuk dihafalkan. Alternatif cara yang lain adalah dengan mendefinisikan sebuah fungsi yang memetakan suatu string dengan sebuah *quasigroup*. Dengan demikian, hanya string itu saja yang perlu dihafalkan. Dengan mengetahui string tersebut, *quasigroup* yang digunakan untuk proses enkripsi bisa diketahui sehingga proses dekripsi juga bisa dilakukan. String itulah yang akan bertindak sebagai *secret key* untuk enkripsi dan dekripsi. Namun hal ini bisa dikatakan cukup lemah juga, karena banyaknya *quasigroup* itu terbatas. Jadi, *exhaustive search* pada *quasigroup* yang akan digunakan bisa menjadi salah satu cara yang cukup ampuh dalam kriptanalis.

Untuk mengatasi kelemahan ini, cara yang bisa dilakukan adalah dengan mengulang-ulang enkripsi berkali-kali menggunakan nilai *leader* yang berbeda-beda. Caranya dengan melakukan komposisi fungsi enkripsi.

Misalkan $L = \{l_1, l_2, ..., l_n\}$ adalah himpunan semua *leader* yang akan digunakan dalam proses enkripsi (dalam hal ini $l_i \in Q$, untuk i=1,2,...,n). Kita definisikan fungsi enkripsi E_L sebagai berikut

$$E_L = E_{l_1} \circ E_{l_2} \circ \cdots \circ E_{l_n}$$

Untuk lebih memperkuat proses enkripsi lagi, *quasigroup* atau *Latin Squares* yang digunakan pada setiap tahap enkripsi E_l juga bisa dibuat berbeda. Jadi, *quasigroup* yang digunakan untuk enkripsi dengan *leader* l_1 tidak sama dengan *quasigroup* yang digunakan untuk enkripsi dengan *leader* l_2 dan berbeda lagi dengan *quasigroup* yang digunakan untuk enkripsi dengan *leader* l_3 dan seterusnya.

Proses dekripsi dapat dilakukan dengan mudah juga, yaitu dengan mengkomposisikan juga fungsi-fungsi dekripsi D_l pada setiap leader. Tetapi, urutan *leader* yang digunakan tidak sama seperti pada proses enkripsi, yaitu harus dibalik dari *leader* ke-n sampai *leader* pertama. Fungsi dekripsi D_L dapat dituliskan sebagai berikut

$$D_L = D_{l_n} \circ D_{l_{n-1}} \circ \cdots \circ D_{l_1}$$

Jika menggunakan *quasigroup* yang berbeda pada setiap proses enkripsi dengan *leader* berbeda, maka proses dekripsi juga harus menggunakan *quasigroup* yang berbeda juga pada setiap tahap dekripsi dengan fungsi D_l. *Quasigroup* yang digunakan pada fungsi dekripsi D_l harus sama dengan *quasigroup* yang digunakan pada fungsi enkripsi E_l.

Pada bagian sebelumnya, telah dibuktikan bahwa masing-masing fungsi D_l dengan *leader* berapapun pasti memenuhi

$$D_l(E_l(P)) = P = E_l(D_l(P))$$

karena hal ini merupakan akibat langsung dari teori *quasigroup*. Namun mengenai fungsi E_L dan D_L yang merupakan komposisi dari beberapa fungsi E_l dan D_l, sebelumnya belum dibuktikan bahwa persamaan

$$D_L(E_L(P)) = P = E_L(D_L(P))$$

akan selalu berlaku. Sebenarnya, hal ini sudah dibuktikan pada makalah-makalah lain. Namun bukti mengenai hal ini tidak akan dibahas di sini karena terlalu panjang dan hal tersebut sebenarnya diluar topik persoalan makalah ini mengenai aplikasi teori *quasigroup* dalam kriptografi.

Dengan menggunakan fungsi komposisi E_L dan D_L tersebut, *quasigroup* maupun *Latin Squares* yang digunakan dalam proses enkripsi dan dekripsi tidak perlu disembunyikan. Dalam prakteknya, informasi mengenai hal ini bisa disimpan bersama-sama dengan penyimpanan cipherteks, misalnya ke dalam sebuah berkas. Yang perlu diingat dalam hal ini hanyalah himpunan L yang berisi *leader-leader* yang digunakan dalam proses enkripsi dan dekripsi. Jika *leader-leader* tersebut merupakan karakter-karakter, maka yang perlu diingat hanyalah string yang dibentuk dari karakter-karakter tersebut. Inilah yang digunakan sebagai *secret key* pada proses enkripsi dan dekripsi.

Lebih jauh, karena hubungan

$$D_l(E_l(P)) = P = E_l(D_l(P))$$

selalu berlaku, maka fungsi enkripsi dan dekripsi pada waktu transformasi *enciphering* dapat dipertukarkan. Hal ini menghasilkan persamaan

$$E_L = H_{l_1} \circ H_{l_2} \circ \cdots \circ H_{l_n}$$

dimana $H_l \in \{E_l , D_l\}$. Dengan konstrukti seperti ini, fungsi E_l akan digunakan pada proses dekripsi (pada fungsi D_L) jika fungsi D_l sudah digunakan pada proses enkripsi (pada fungsi E_l) dengan kata lain ketika $H_l = D_l$.

3.3 *Cipher Blok* dengan Quasigroup

Pada bahasan sebelumnya, *quasigroup* digunakan untuk enkripsi dengan mode karakter. *Quasigroup* dapat juga digunakan untuk enkripsi dengan mode bit. Hal ini dilakukan dengan algoritma yang sama seperti sebelumnya, tetapi masukan fungsi E tidak harus berupa karakter melainkan *entity* dari bit-bit dimana panjang bit setiap elemen ditentukan dengan ukuran dari *Latin Square* yang berkorespondensi dengan *quasigroup* yang digunakan.

Jadi, fungsi E yang digunakan tetap

$$E(p_1p_2...p_n) = c_1c_2...c_n ,$$

Dimana

$$\begin{cases} c_1 &= l * p_1 \\ c_{i+1} &= c_i * p_{i+1} \end{cases}$$

untuk i=1,2,...,n. tetapi elemen c_i dan p_i tidak harus berupa karakter, melainkan *entity* dari bit-bit.

Dengan demikian, fungsi dekripsi yang digunakan juga sama,

$$D_l(c_1c_2...c_n) = p_1p_2...p_n ,$$

dimana

$$\begin{cases} p_1 &= l \setminus c_1 \\ p_{i+1} &= c_i \setminus c_{i+1} \end{cases}$$

untuk i = 1, 2, ..., n.

Demikian juga fungsi komposisi yang digunakan

$$E_L = E_{l_1} \circ E_{l_2} \circ \cdots \circ E_{l_n}$$

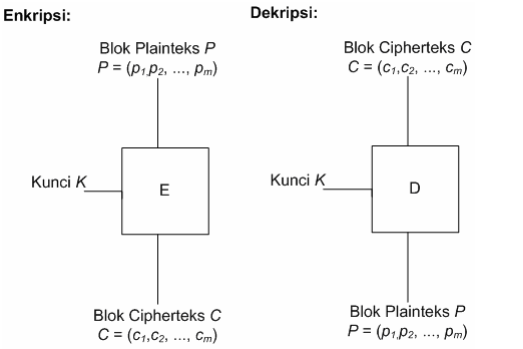
dan

$$D_L = D_{l_n} \circ D_{l_{n-1}} \circ \cdots \circ D_{l_1}$$

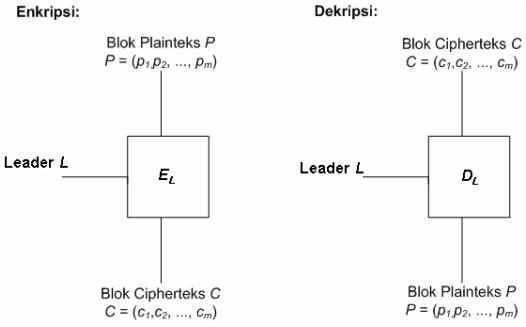
atau bisa juga

$$E_L = H_{l_1} \circ H_{l_2} \circ \cdots \circ H_{l_n}$$

Gambar 5 memperlihatkan enkripsi satu blok plainteks dengan fungsi E menghasilkan satu blok cipherteks dan dekripsi satu blok cipherteks dengan fungsi D menghasilkan satu blok plainteks. Gambar 5.a memperlihatkan algoritma *cipher* blok secara umum dan gambar 5.b memperlihatkan algoritma *cipher* blok dengan menggunakan *quasigroup*.



(a) bentuk umum *cipher* blok



(b) *cipher* blok dengan *quasigroup*

Gambar 5 Skema enkripsi dan dekripsi *cipher* blok

Dengan menggunakan mode bit ini, mode operasi *cipher* blok yang digunakan juga bisa dipilih. Ada empat mode operasi *cipher* blok yang bisa digunakan, yaitu

1. *Electronic Code Book* (*ECB*)
2. *Cipher Block Chaining* (*CBC*)
3. *Cipher Feedback* (*CFB*)
4. *Output Feedback* (*OFB*)

3.3.1 *Electronic Code Block* (*ECB*)

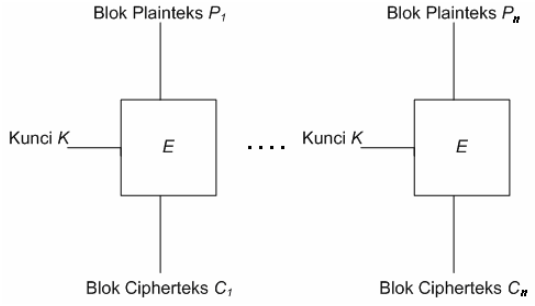
Pada mode ini, setiap blok plainteks P_i dienkripsi secara individual dan independen menjadi sebuah blok cipherteks C_i. Secara matematis, enkripsi pada mode *ECB* ini dinyatakan sebagai

$$C_i = E_K (P_i)$$

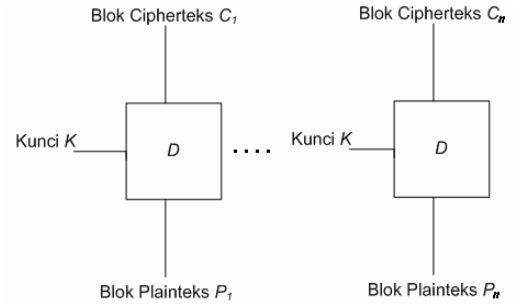
dan dekripsi sebagai

$$P_i = D_K (C_i)$$

yang dalam hal ini, P_i dan C_i berturut-turut menyatakan blok-blok plainteks dan cipherteks ke-*i*. Skema enkripsi dan dekripsi dengan mode *ECB* dapat dilihat pada gambar 6 dan .



(a) Enkripsi



(b) Dekripsi

Gambar 6 Skema enkripsi dan dekripsi dengan mode *ECB*

Sebagai contoh, sebuah plainteks (dalam biner) adalah

```
1100101011011010
```

Selanjutnya, tentukan panjang blok yang ingin digunakan. Panjang blok yang digunakan ini pada umumnya sama dengan panjang kunci. Misalkan kunci yang digunakan (dalam biner) adalah `1101` atau dalam notasi HEX adalah D. Dengan demikian, panjang blok yang digunakan adalah 4. Sekarang bagi plainteks menjadi blok-blok yang berukuran 4-bit.

```
1100 1010 1101 1010
```

atau dalam notasi HEX adalah CADA.

Proses enkripsi dilakukan dengan membaca 4-bit plainteks, kemudian mengenkripsinya dengan fungsi *E* dan kunci *K* sampai semua blok-blok berukuran 4-bit pada plainteks selesai dienkripsi. Sebagai contoh, fungsi *E* yang digunakan adalah dengan meng-XOR-kan blok plainteks P_i dengan kunci *K*, kemudian menggeser blok hasil operasi XOR tersebut secara *wrapping* bit-bit dua posisi ke-kanan. Proses enkripsi untuk setiap blok digambarkan sebagai berikut.

```
                1100 1010 1101 1010
                1101 1101 1101 1101
```

Hasil XOR:	0001	0111	0000	0111
Geser bit:	0100	1101	0000	1101
Dalam HEX	4	D	0	D

Jadi, hasil enkripsi plainteks

1100101011011010 (CADA dalam HEX)

adalah

0100110100001101 (4D0D dalam HEX).

Pada proses dekripsi, blok-blok cipherteks C_i juga didekripsi secara individual dan independen. Sebagai contoh, pada enkripsi di atas, fungsi enkripsi yang digunakan adalah meng-XOR-kan blok plainteks dengan kunci kemudian menggeser secara *wrapping* bit-bit dua posisi ke kanan. Dengan demikian, fungsi dekripsi yang digunakan adalah dengan menggeser blok cipherteks secara *wrapping* bit-bit dua posisi ke kiri kemudian meng-XOR-kan hasil *wrapping* tersebut dengan kunci. Dengan demikian, persamaan

$$P_i = D_K(E_K(P_i))$$

akan selalu terpenuhi.

Pada contoh di atas, cipherteksnya adalah

0100110100001101

atau 4D0D dalam notasi HEX.

Proses dekripsi untuk setiap blok digambarkan sebagai berikut

```
                0100 1101 0000 1101

Geser bit:      0001 0111 0000 0111
                1101 1101 1101 1101
```

Hasil XOR:	1100	1010	1101	1010
Dalam HEX	C	A	D	A

Terlihat bahwa proses dekripsi menghasilkan plainteks yang sama seperti sebelum proses enkripsi dilakukan.

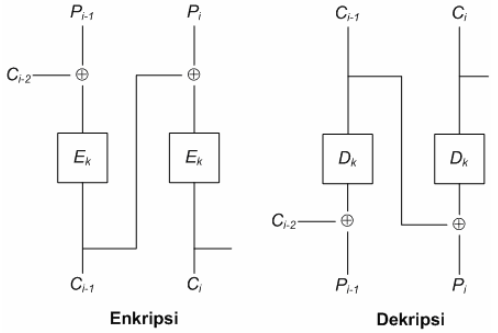
Sekarang perhatikan bahwa setiap blok plainteks dienkripsikan secara individual dan independen dengan sebuah kunci yang tetap. Dengan demikian, satu blok plainteks yang sama akan selalu dienkripsi menjadi sebuah blok cipherteks yang sama pula (atau identik). Pada contoh di atas, blok 1010 muncul dua kali dan selalu dienkripsi menjadi 1101.

Kata "*code book*" di dalam *ECB* (*Electronic Code Book*) muncul dari fakta bahwa karena blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama, maka secara teoritis dimungkinkan membuat buku kode plainteks dan cipherteks yang berkoresponden. Namun semakin besar ukuran blok, semakin besar pula ukuran buku kodenya. Misalkan jika blok berukuran 32-bit, maka buku kode terdiri dari $2^{32} - 1$ buah kode (*entry*), yang berarti cukup besar untuk disimpan. Lagipula, setiap kunci mempunyai buku kode yang berbeda.

Satu isu yang perlu diperhatikan dalam *cipher* blok dengan mode operasi *ECB* ini adalah bahwa plainteks harus dibagi-bagi menjadi blok-blok bit yang ukurannya sama. Ada kemungkinan, panjang plainteks tidak habis dibagi dengan panjang ukuran blok yang ditetapkan. Misalnya panjang ukuran blok ditetapkan 64-bit atau sama dengan panjang kunci. Pada contoh di atas, jika panjang kunci adalah 12-bit, maka blok-blok plainteks tidak dapat dibagi menjadi dua blok dengan ukuran yang sama 12-bit. Untuk mengatasi hal ini, dilakukan *padding* atau penambahan beberapa bit agar blok plainteks habis dibagi oleh ukuran blok yang telah ditetapkan. Dalam contoh di atas, blok plainteks berukuran 16-bit. Agar habis dibagi 12, maka harus ditambahkan *padding bit* sebanyak 8-bit. Dengan demikian, blok plainteks menjadi 24-bit dan dapat dibagi menjadi dua blok berukuran 12-bit. Pada umumnya, *padding bit* dipilih dengan pola teratur, misalnya bit 0 semua, atau bit 1 semua atau bit 0 dan bit 1 berselang-seling. Jika proses enkripsi harus mengandung penambahan *padding bit*, maka proses dekripsi harus mengandung penghilangan *padding bit*. Misalnya pada contoh di atas, plainteks berukuran 16-bit. Kemudian kunci yang digunakan adalah 12-bit. Oleh karena itu, sebelum proses enkripsi dilakukan, ditambahkan *padding* bit sebanyak 8-bit, sehingga menjadi 24-bit. Proses enkripsi 24-bit plainteks tersebut akan menghasilkan cipherteks berukuran 24-bit juga. Dengan melakukan dekripsi cipherteks berukuran 24-bit tersebut, akan dihasilkan plainteks berukuran 24-bit juga. 24-bit plainteks ini adalah plainteks awal yang berukuran 16-bit ditambah dengan *padding bit*. Jadi, agar proses dekripsi mengembalikan plainteks sama seperti semula, maka *padding bit* berukuran 8-bit ini harus dihilangkan.

3.3.2 *Cipher Block Chaining* (*CBC*)

Mode ini merupakan mekanisme umpan-balik (*feedback*) pada sebuah blok. Dalam hal ini, hasil enkripsi blok sebelumnya di-umpan-balikkan ke dalam enkripsi blok yang *current*. Caranya, blok plainteks yang *current* di-XOR-kan terlebih dahulu dengan blok cipherteks hasil enkripsi sebelumnya, selanjutnya hasil peng-XOR-an ini masuk ke dalam fungsi enkripsi menghasilkan satu blok cipherteks. Dekripsi dilakukand dengan memasukkan blok yang cipherteks yang *current* ke dalam fungsi dekripsi, kemudian meng-XOR-kan hasil dekripsi tersebut dengan blok cipherteks sebelumnya. Dalam hal ini, blok cipherteks sebelumnya berfungsi sebagai umpan-maju (*feedforward*) pada akhir proses dekripsi.



Gambar 7 Skema enkripsi dan dekripsi dengan mode *CBC*

Gambar 7 memperlihatkan skema enkripsi dan dekripsi dengan mode CBC.

Secara matematis, enkripsi dengan mode *CBC* dinyatakan sebagai

$$C_i = E_K(P_i \oplus C_{i-1})$$

dan dekripsi sebagai

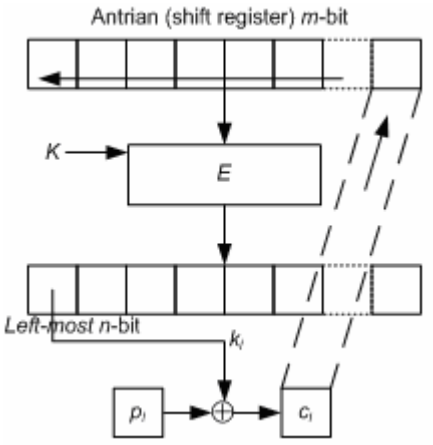
$$P_i = D_K(C_i) \oplus C_{i-1}$$

Dengan demikian, selain plainteks dan kunci, proses enkripsi dan dekripsi membutuhkan satu parameter lagi, yaitu C_0. Dalam hal ini, C_0 adalah *initialization vector* (*IV*). *IV* dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh program. Jadi, untuk menghasilkan blok cipherteks yang pertama (C_1), *IV* digunakan untuk menggantikan blok cipherteks sebelumnya, C_0. Sebaliknya pada dekripsi, blok plainteks yang pertama (P_1) diperoleh dengan cara meng-XOR-kan *IV* dengan hasil dekripsi terhadap blok cipherteks yang pertama.

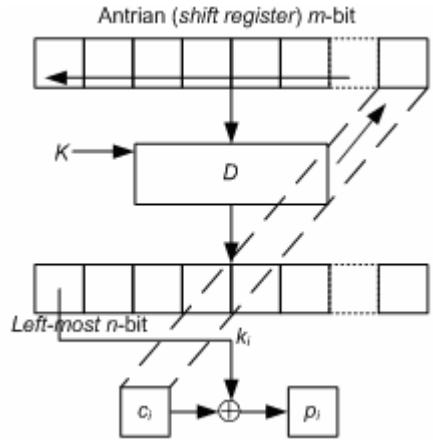
Dengan mode *CBC* ini, setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya saja, tetapi juga pada seluruh blok plainteks sebelumnya. Akibatnya, blok plainteks yang sama berlum tentu menghasil blok cipherteks yang sama pula. Blok plainteks yang sama akan menghasilkan blok cipherteks yang sama jika semua blok-blok plainteks sebelumnya sama dan *IV* nya juga sama. Kalau *IV* dipilih secara acak dari program, maka blok plainteks yang sama hampir tidak mungkin menghasilkan blok cipherteks yang sama pula. Pada mode *CBC*, *IV* tidak perlu dirahasiakan. Misalnya, *IV* bisa dikirimkan atau disimpan bersamaan dengan cipherteks.

3.3.3 *Cipher Feedback* (*CFB*)

Jika mode *CBC* yang diterapkan pada aplikasi komunikasi data, maka enkripsi tidak dapat dilakukan bila blok plainteks yang diterima belum lengkap. Misalnya bila pengiriman pengiriman data dilakukan setiap karakter di-*enter* dari terminal komputer ke *host*.



(a) Enkripsi



(b) Dekripsi

Gambar 8 Skema enkripsi dan dekripsi dengan mode *CFB n*-bit

Pada mode *CFB*, data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. Unit

yang dienkripsikan dapat berupa bit per bit, 2-bit, 3- bit, dan seterusnya. Bila unit yang dienkripsikan satu karakter setiap kalinya, maka mode *CFB*-nya disebut *CFB* 8-bit. Secara umum *CFB* *n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, dimana $n \leq m$ (m = ukuran blok).

Mode *CFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.

Tinjau mode *CFB* *n*-bit yang bekerja pada blok berukuran *m*-bit. Algoritma enkripsi dengan mode *CFB* adalah sebagai berikut (lihat gambar 8):

1. Antrian diisi dengan *IV* (*initialization vector*) seperti pada mode *CBC*.
2. Enkripsikan antrian dengan fungsi *E* dan kunci *K*. *n* bit paling kiri dari hasil enkripsi berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari plainteks menjadi *n*-bit pertama dari cipherteks. Salinan (*copy*) *n*-bit dari cipherteks ini dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan semua $m - n$ bit lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang sudah digunakan.
3. $m - n$ bit plainteks berikutnya dienkripsikan dengan cara yang sama seperti pada langkah sebelumnya (langkah 2).

Algoritma dekripsi dengan mode *CFB* merupakan kebalikan dari algoritma enkripsi. Tetapi dalam algoritma dekripsi, fungsi dekripsi *D* yang digunakan bukan merupakan kebalikan dari fungsi enkripsi *E*, melainkan sama. Jadi, fungsi *D* sama dengan fungsi *E*. Algoritma dekripsi dengan mode *CFB* adalah sebagai berikut:

1. Antrian diisi dengan *IV* (*initialization vector*) sama seperti proses enkripsi.
2. Enkripsikan antrian dengan fungsi *E* dan kunci *K*. *n* bit paling kiri dari hasil enkripsi berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari cipherteks menjadi *n*-bit pertama dari plainteks. Selain di-XOR-kan, *n*-bit cipherteks juga dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan semua $m - n$ bit lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang sudah digunakan.
3. $m–n$ bit cipherteks berikutnya dienkripsikan dengan cara yang sama seperti langkah 2.

Secara matematis, mode *CFB* *n*-bit dapat dinyatakan sebagai:

Proses Enkripsi:

$$C_i = P_i \oplus MSB_m(E_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

Proses Dekripsi:

$$P_i = C_i \oplus MSB_m(D_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

yang dalam hal ini:

X_i = isi antrian dengan X_i adalah *IV*
E = fungsi enkripsi dengan algoritma *cipher* blok
D = fungsi dekripsi dengan algoritma *cipher* blok
K = kunci
m = panjang blok enkripsi/dekripsi
n = panjang unit enkripsi/dekripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

Sama seperti mode *CBC*, *IV* pada mode *CFB* tidak perlu dirahasiakan. *IV* harus unik untuk setiap pesan, sebab *IV* yang sama untuk setiap pesan yang berbeda akan menghasilkan *keystream* k_i yang sama.

3.3.4 *Output-Feedback* (*OFB*)

Secara umum, mode *OFB* hampir sama dengan mode *CFB*. Perbedaannya hanyalah *n*-bit dari hasil enkripsi terhadap antrian disalin menjadi elemen posisi paling kanan antrian.

Seperti pada mode *CFB*, pada mode *OFB* data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. *OFB* *n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, dimana $n \leq m$ (m = ukuran blok). Algoritmanya secara umum sama dengan mode *CFB*. Perbedaannya hanya ketika memasukkan *n*-bit ke dalam antrian.

Secara matematis, mode *OFB* *n*-bit dapat dinyatakan sebagai:

Proses Enkripsi:

$$C_i = P_i \oplus MSB_m(E_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, LSB_n(E_k(X_i))$$

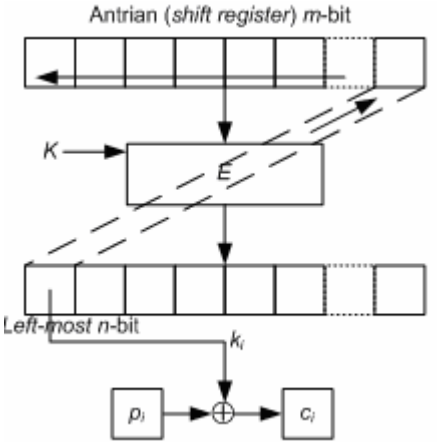
Proses Dekripsi:

$$P_i = C_i \oplus MSB_m(D_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, LSB_n(E_k(X_i))$$

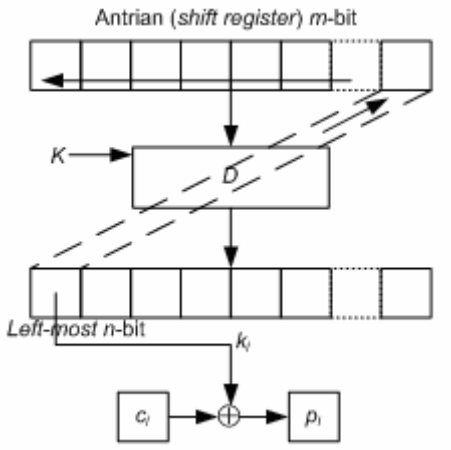
yang dalam hal ini:

X_i = isi antrian dengan X_i adalah *IV*

E = fungsi enkripsi dengan algoritma *cipher* blok
D = fungsi dekripsi dengan algoritma *cipher* blok
K = kunci
m = panjang blok enkripsi/dekripsi
n = panjang unit enkripsi/dekripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte



(a) Enkripsi



(b) Dekripsi

Gambar 9 Skema enkripsi dan dekripsi pada mode *OFB n*-bit

4 Kesimpulan

Kesimpulan yang dapat diambil dari studi mengenai aplikasi teori *quasigroup* dalam kriptografi adalah:

1. *Vigenere cipher* merupakan salah bentuk khusus dari sebuah algoritma kriptografi yang menggunakan teori *quasigroup*. *Vigenere cipher* dapat dibuat lebih umum dengan mengubah bujursangkar *Vigenere* yang digunakan dalam proses enkripsi dan dekripsi. Bujursangkar *Vigenere* dapat digantikan dengan bujursangkar lain yang didapatkan dengan menggunakan *Latin Square*. Dengan menggunakan bentuk umum dari bujursangkar *Vigenere* ini, beberapa sifat yang *vulnerable* pada bujursangkar *Vigenere* yang asli dapat dihilangkan, misalnya kesimetrian bujursangkar terhadap diagonal yang ditarik dari sudut kiri atas ke kanan bawah. Dengan demikian, proses kriptanalis pada bentuk umum algoritma *Vigenere* akan menjadi lebih sulit.

2. Selain *Vigenere cipher*, ada banyak algoritma enkripsi dan dekripsi yang menggunakan teori *quasigroup*. Algoritma-algoritma enkripsi dan dekriptsi ini memanfaatkan sifat khusus *quasigroup*, yaitu operasi biner $*$ pada *quasigroup* selalu memiliki invers kiri dan kanan yang tunggal. Dengan demikian, proses enkripsi dapat menggunakan kombinasi dari operasi biner tersebut dan proses dekripsi menggunakan inversnya, baik invers kiri maupun invers kanan. Operasi biner yang dikombinasikan tersebut dapat membentuk sebuah fungsi enkripsi *E* dan invers dari operasi biner tersebut (baik invers kiri maupun invers kanan) dapat membentuk sebuah fungsi dekripsi *D*. Untuk membuat sebuah algoritma enkripsi yang lebih kuat dengan menggunakan *quasigroup*, fungsi-fungsi tersebut dapat dikomposisikan. Dalam teori *qusigroup*, dapat dibuktikan bahwa komposisi-komposisi fungsi tersebut tetap memenuhi persamaan enkripsi dan dekripsi yang benar
$$D(E(P)) = P$$

3. Algoritma enkripsi dan dekripsi dengan menggunakan *quasigroup* dapat bekerja pada mode karakter dan mode bit. Pada mode bit contohnya adalah *Vigenere cipher* dan bentuk umumnya. Pada mode blok, algoritma ini dapat dikombinasikand dengan mode operasi *cipher* blok, yaitu *Electronic Code Book* (*ECB*), *Cipher Block Chaining* (*CBC*), *Cipher FeedBack* (*CFB*) dan *Output-Feedback* (*OFB*). Dengan menggunakan kombinasi ini, maka proses enkripsi dan dekripsi dengan menggunakan *quasigroup* dapat dibuat lebih kuat, sehingga semakin sukar untuk dipecahkan dengan kriptanalis.

DAFTAR PUSTAKA

[1] Shcherbacov, Victor, *Elements of quasigroup theory and some it's applications in code theory and cryptologi*.

[2] Hassinen, M., Markovski S., *Secure SMS messaging using quasigroup encryption and Java SMS API*.

[3] Gligoriski, D., *Stream cipher based on quasigroup string transformation in Zp*.

[4] Koscielny, Czeslaw, *Generating quasigroups for cryptographic applications*.

[5] Kumar, Gagan, *Term Paper on Classical Cryptography*.

[6] Munir, Rinaldi. 2006. *Diktat Kuliah IF5054, Kriptografi*.