Block Cipher Jarl

Jevant Jedidia Augustine - 13520133
David Karel Halomoan - 13520154
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520133@std.stei.itb.ac.id, 13520154@std.stei.itb.ac.id

I. Pendahuluan

Dalam era digital yang semakin maju, keamanan data menjadi salah satu hal yang sangat penting. Oleh karena itu, teknologi kriptografi digunakan untuk melindungi data dan informasi yang sensitif dari pihak yang tidak berwenang. Salah satu teknik kriptografi yang banyak digunakan adalah *block cipher*. *Block cipher* adalah jenis algoritma kriptografi yang mengubah *plaintext* menjadi *encrypted text* dengan cara membagi pesan menjadi blok-blok dengan ukuran tertentu dan mengenkripsi setiap blok secara independen maupun dependen dengan blok lainnya. Dalam makalah ini, penulis akan membahas sebuah *cipher block* "baru" dengan menggunakan teknik *playfair cipher* dan beberapa teknik lainnya.

A. AES

AES (*Advanced Encryption Standard*) merupakan standard algoritma baru sebagai pengganti DES (*Data Encryption Standard*) karena DES sudah dianggap tidak aman. AES dihasilkan dari lomba yang diadakan oleh NIST (*National Institute of Standards and Technology*). Dari perlombaan tersebut, terdapat 5 finalis, *Rijndael*, *Serpent*, *Twofish*, *RC6*, dan *MARS*. Algoritma *Rijndael* yang meraih juara satu pada perlombaan tersebut ditetapkan menjadi algoritma AES pada bulan November 2001. Secara de-fakto, terdapat 2 varian AES, yaitu AES-128 dan AES-256 dengan panjang kunci 128 bit dan 256 bit secara berurutan. [3]

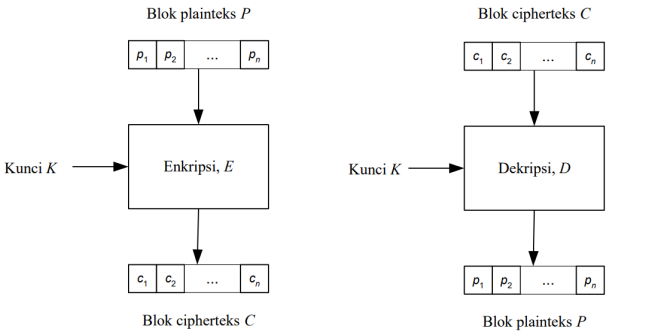
B. Playfair Cipher

Playfair cipher merupakan sebuah *polygram cipher* yang ditemukan oleh Sir Charles Wheatstone yang kemudian dipromosikan oleh Baron Lyon Playfair pada tahun 1854. *Playfair cipher* mengenkripsi *plainteks* per-bigram untuk menyembunyikan frekuensi huruf dalam teks. *Playfair cipher* akan memasukan kunci ke dalam sebuah matriks kemudian menggunakan matriks tersebut untuk mengenkripsikan *plaintext* berdasarkan pasangan bigramnya. [2]

II. Dasar Teori

A. Block Cipher

Block cipher merupakan teknik kriptografi dimana *plaintext* dibagi menjadi blok-blok bit dengan panjang yang sama. Ukuran blok yang biasanya digunakan adalah 64 bit, 128 bit, dan 256 bit. Sebaliknya, dekripsi yang dilakukan terhadap *ciphertext* akan dilakukan per-blok sama seperti enkripsi *plaintext*. Panjang kunci yang dimasukkan tidak harus sama dengan panjang *plaintext*.



Gambar 2.1 *Block cipher*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/13-Block-Cipher-Bagian1-2023.pdf

Terdapat 5 mode operasi pada *cipher* blok:

- *Electronic Code Book*
- *Cipher Book Chaining*
- *Cipher Feedback*
- *Output Feedback*
- *Counter Mode* [1]

B. Prinsip *Diffusion* dan *Confusion* Shannon

Dalam *Communication Theory of Secrecy Systems*, Claude Shannon memperkenalkan prinsip *diffusion* dan *confusion*, yaitu 2 prinsip yang dapat diterapkan dalam merancang algoritma kriptografi sehingga serangan berbasis statistik lebih sulit untuk dilakukan.

Confusion merupakan prinsip yang menyembunyikan hubungan statistik antara *plaintext*, *ciphertext*, dan kunci. Prinsip ini dapat direalisasikan dengan menggunakan teknik substitusi yang *non-linear* yang biasanya dilakukan dengan menggunakan *lookup table*. *Confusion* juga dapat diterapkan dengan menggunakan *one-time pad*. [4]

Diffusion merupakan prinsip dimana perubahan satu bit pada *plaintext* maupun kunci akan berpengaruh ke sebanyak mungkin bit pada *ciphertext*. Hal tersebut berarti perubahan kecil pada *plaintext* atau kunci akan memiliki perubahan besar yang tidak terprediksi pada *ciphertext*. *Diffusion* dapat direalisasikan dengan menggunakan teknik permutasi atau transposisi secara berulang-ulang. [4]

C. *AES*

AES merupakan algoritma kriptografi yang menjadi standar algoritma enkripsi kunci simetris. Algoritma yang digunakan untuk AES itu sendiri adalah algoritma *Rijndael*. Secara de-fakto, terdapat 2 varian AES, yaitu AES-128 dan AES-256. Untuk AES-128, garis besar algoritmanya adalah:

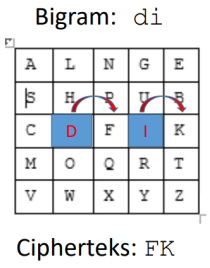
- *AddRoundKey*: melakukan proses XOR antara *plaintext* dengan kunci.
- Putaran sebanyak N-1 kali dengan setiap putaran melakukan:
  - *SubBytes*: substitusi *byte* dengan menggunakan *S-box*
  - *ShiftRows*: pergeseran baris-baris *array state* dengan *wrapping*
  - *MixColumns*: mengacak data di masing-masing kolom *array state*
  - *AddRoundKey*: melakukan proses XOR antara *state* sekarang dengan *round key*
- *FinalRound*: proses untuk putaran terakhir
  - *SubBytes*
  - *ShiftRows*
  - *AddRoundKey*

Algoritma *Rijndael* memiliki $2^{128} = 3{,}4 \times 10^{38}$ kemungkinan kunci. Apabila diasumsikan bahwa sebuah komputer yang sangat cepat dapat mencoba 1 juta kunci setiap detik, maka dibutuhkan waktu sekitar $5{,}4 \times 10^{24}$ tahun untuk mencoba semua kunci. Alhasil, algoritma *Rijndael* dapat dibilang tahan terhadap *brute force attack*. [3]

D. *Playfair Cipher*

Playfair cipher merupakan teknik kriptografi klasik dimana proses enkripsi dilakukan terhadap bigram untuk menyembunyikan statistik antara *plaintext* dan *ciphertext*. *Playfair cipher* menggunakan sebuah matriks yang diisi berdasarkan kunci masukan. Pada algoritma *playfair* standard (huruf abjad), maka matriks yang akan dibangun adalah sebesar 5x5 yang berisikan semua abjad kecuali huruf "j". Setelah matriks diisi dengan huruf berdasarkan kunci, maka *plainteks* akan dipetakan berdasarkan peraturan berikut:

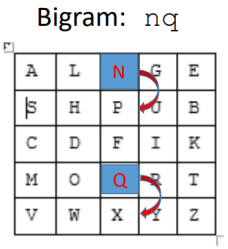
- Apabila huruf pada sebuah pasangan berada pada baris yang sama, maka huruf tersebut akan disubstitusi dengan huruf yang ada di sebelah kanan huruf tersebut (bersifat siklik).



Gambar 2.2 Aturan pertama *playfair cipher*

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/04-Kriptografi-Klasik-Bagian3-(2023).pdf

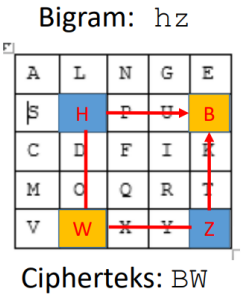
- Apabila huruf pada sebuah pasangan berada pada kolom yang sama, maka huruf tersebut akan disubstitusi dengan huruf yang ada di bawah huruf tersebut (bersifat siklik).



Gambar 2.3 Aturan kedua *playfair cipher*

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/04-Kriptografi-Klasik-Bagian3-(2023).pdf

- Jika kedua huruf tidak berada pada baris atau kolom yang sama:
  - Huruf pertama diganti dengan perpotongan baris huruf pertama dan kolom huruf kedua
  - Huruf kedua diganti dengan huruf pada titik sudut keempat dari persegi panjang yang dibentuk dari ketiga huruf.



Gambar 2.4 Aturan ketiga *playfair cipher*

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/04-Kriptografi-Klasik-Bagian3-(2023).pdf

Terdapat 26 x 26 = 677 bigram pada abjad standard, sehingga identifikasi bigram individual lebih sulit. Akan tetapi, *playfair cipher* dapat dipecahkan dengan analisis frekuensi bigram. Kriptanalisis dapat dilakukan apabila terdapat tabel frekuensi bigram bahasa *plaintext* dan *ciphertext* yang cukup panjang. [2]

E. *Jaringan Feistel*

Merupakan struktur *enchipering* pada setiap putaran. Algoritma yang menggunakan jaringan feistal akan membagi *plaintext* menjadi dua bagian dan pada masing-masing bagian dilakukan proses transformasi menjadi upa-bagian pada putaran selanjutnya. Struktur ini bersifat *reversible* sehingga dapat dilakukan dari atas ke bawah (enkripsi) maupun bawah ke atas (dekripsi). [4]

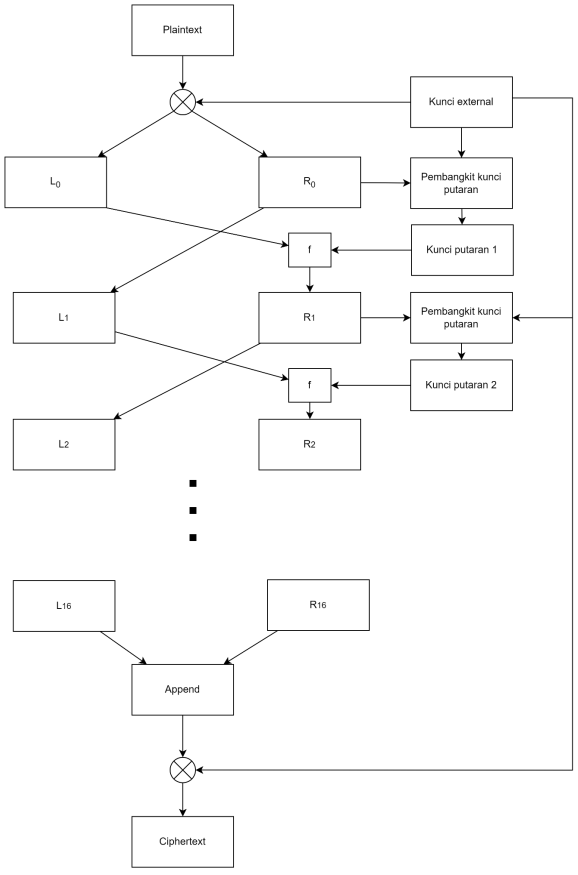
III. RANCANGAN *BLOCK CIPHER*

Rancangan *block cipher* yang dimuat pada makalah ini memiliki ukuran blok pesan sebesar 128 bit, panjang kunci 128 bit, serta jumlah putaran sebanyak 16 kali.

A. Gambaran Umum

Alur kerja *block cipher* Jarl adalah sebagai berikut:

1. Algoritma akan menerima sebuah pasangan blok *plaintext* dan kunci eksternal.
2. *Plaintext* akan di-XOR-kan dengan kunci eksternal kemudian dibagi menjadi 2 bagian (L_0 dan R_0) yang memiliki panjang 64 bit untuk menjadi basis jaringan feistel.
3. Pembangkit dari kunci putaran akan menerima masukan kunci eksternal dan blok R_x untuk membangkitkan kunci putaran pada putaran *x+1*.
4. Blok R_x akan dijadikan blok L_{x+1} pada putaran yang selanjutnya tanpa dilakukan perubahan apapun.
5. Untuk mendapatkan blok R_{x+1}, akan digunakan fungsi putaran yang menerima masukan berupa L_x dan kunci putaran.
6. Langkah 3 sampai 5 akan diulang sebanyak 16 kali putaran.
7. Pada akhir putaran, blok L_x dan R_x akan digabung kemudian di-XOR-kan dengan kunci eksternal untuk menghasilkan blok *ciphertext* sepanjang 128 bit.



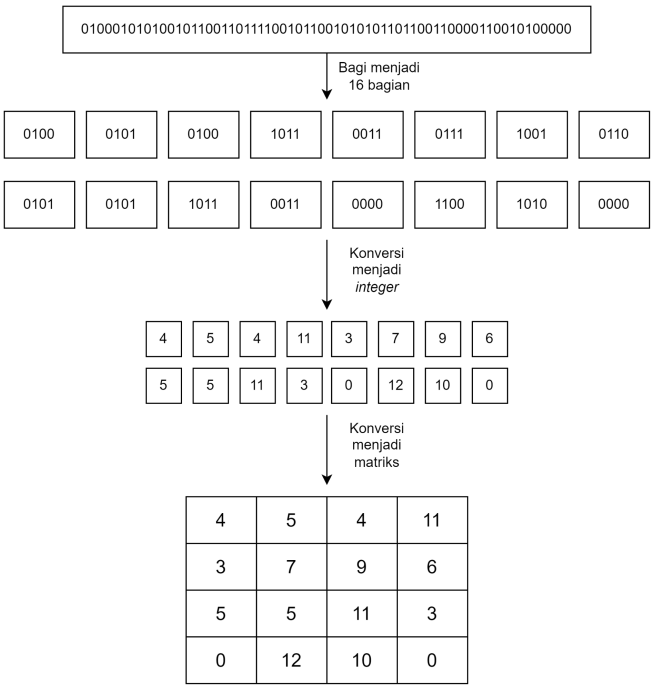
Gambar 3.1 Gambaran umum *block cipher* Jarl

Karena rancangan yang dibuat merupakan jaringan feistal, maka proses dekripsi dan enkripsi menggunakan alur yang serupa. Proses enkripsi dilakukan dari arah atas ke bawah dan proses dekripsi dilakukan dari arah bawah ke atas.

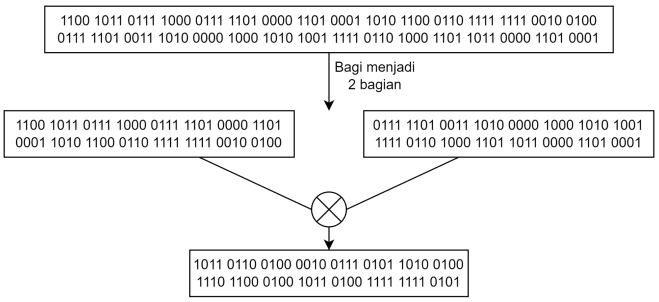
B. Pembangkit Kunci Putaran

Fungsi untuk membangkitkan kunci putaran akan menggunakan perkalian matriks antara R_x dan kunci eksternal dan transpose dari matriks hasil perkalian tersebut. Untuk dapat melakukan perkalian, blok R_x dan kunci perlu dikonversi terlebih dahulu menjadi matriks.

Untuk melakukan konversi tersebut, maka blok R_x yang berukuran 64 bit akan dibagi menjadi 16 bagian dengan setiap bagian memiliki panjang 4 bit. Bit-bit pada setiap bagian kemudian akan dikonversi menjadi *integer* dengan kemungkinan nilai 0 sampai 15. Nilai-nilai hasil konversi tersebut kemudian akan dimasukan ke sebuah matriks sesuai dengan urutan bagiannya masing-masing. Konversi dari bit ke matriks untuk kunci eksternal dilakukan dengan menggunakan proses yang sama, akan tetapi kunci akan dibagi menjadi 2 bagian 64 bit yang kemudian akan di-XOR-kan satu sama dengan yang lain terlebih dahulu sebelum dikonversi menjadi matriks.

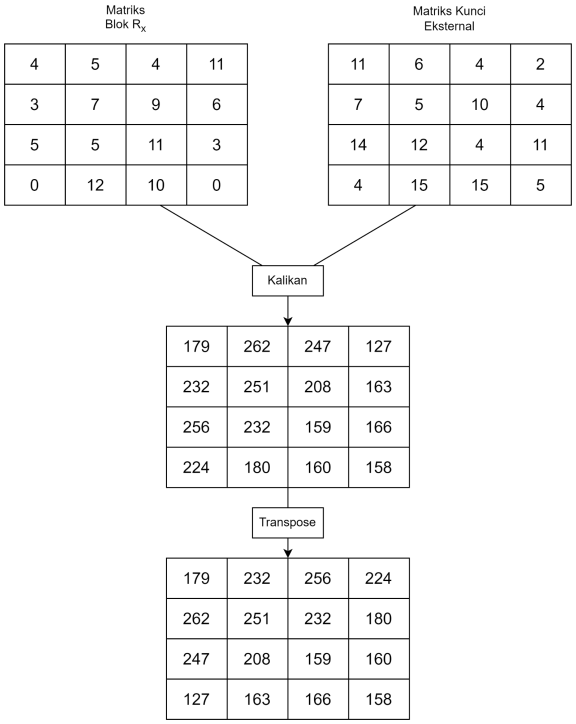


Gambar 3.2 Konversi blok 64 bit menjadi matriks



Gambar 3.3 Konversi kunci 128 bit menjadi 64 bit

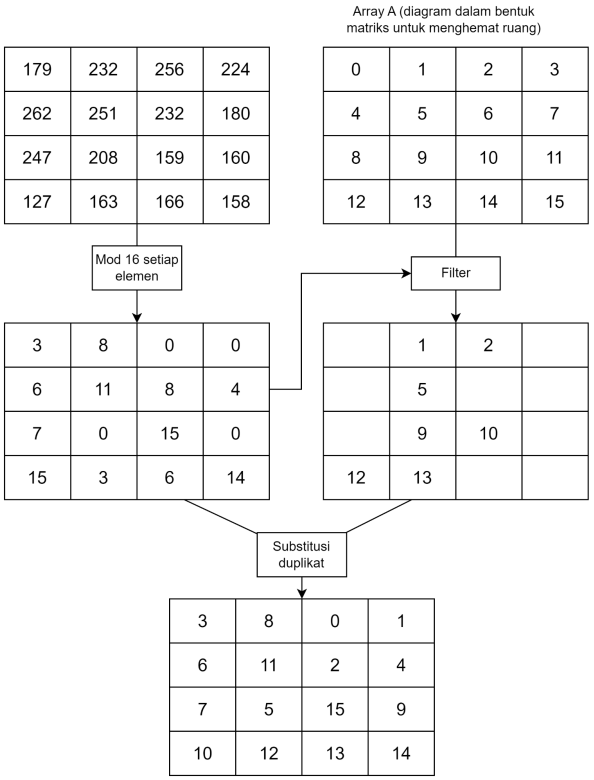
Setelah kunci eksternal dan blok R_x dikonversi menjadi matriks, kedua matriks tersebut dikalikan (R x Kunci) dan matriks hasil perkaliannya di-*transpose*.



Gambar 3.4 Perkalian dan *transpose* matriks R_x dan kunci

Kunci putaran untuk *block cipher* Jarl berbentuk matriks *playfair*, maka dari itu, matriks yang didapat dari hasil operasi perkalian dan *transpose* perlu diubah terlebih dahulu menjadi matriks *playfair*. Hal tersebut dilakukan dengan langkah sebagai berikut:

1. Modulokan setiap elemen pada matriks dengan 16 agar setiap elemen pada matriks memiliki *range* 0 sampai 15.
2. Inisialisasi sebuah *array* A dengan isi 0 sampai 15.
3. Iterasi elemen *array* A kemudian periksa apakah elemen pada indeks tertentu berada pada matriks, bila ada, hapus dari *array*.
4. Iterasi matriks dan periksa apakah elemen pada indeks tertentu merupakan duplikat atau tidak. Bila iya dan elemen duplikat sudah muncul terlebih dahulu daripada elemen sekarang, maka ganti nilai duplikat tersebut dengan elemen pertama *array* A kemudian hapus elemen pertama pada *array* A.



Gambar 3.5 Normalisasi matriks hasil perkalian dan *transpose*

Matriks *playfair* hasil normalisasi menjadi luaran dari pembangkit kunci. Tidak ada perbedaan antara pembangkitan kunci untuk enkripsi maupun dekripsi.

C. Fungsi Putaran

Fungsi putaran digunakan untuk menghasilkan blok R_{x+1}. Fungsi putaran akan menerima masukan berupa blok L_x dan kunci putaran hasil pembangkit kunci. Fungsi putaran *block cipher* Jarl dibagi menjadi 2 tahap, *playfair mapping* dan *shifting & rotating*.

1. Playfair mapping

Sebelum dilakukan *mapping*, masukan blok L_x akan dikelompokan per 8 bit yang selanjutnya dapat diubah menjadi sepasang 4 bit yang kemudian dikonversi menjadi sepasang *integer*. Setelah itu, tergantung pada apakah proses merupakan enkripsi atau dekripsi, akan dilakukan *mapping* terhadap setiap pasangan dengan kunci putaran berdasarkan aturan:

Enkripsi

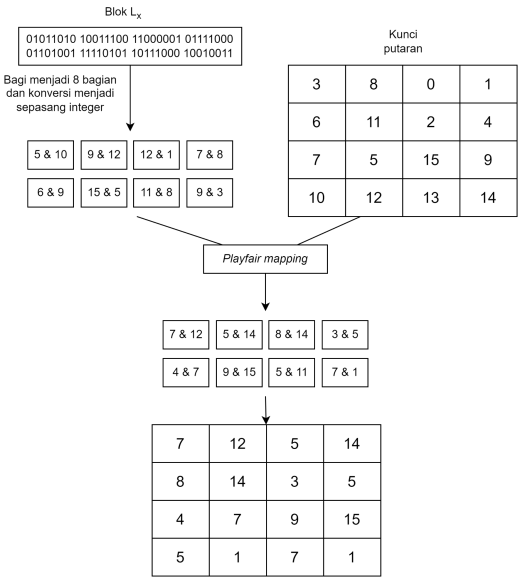
- Apabila angka pada sebuah pasangan berada pada baris yang sama, maka angka tersebut akan disubstitusi dengan angka yang ada di sebelah kanan angka tersebut (bersifat siklik).
- Apabila angka pada sebuah pasangan berada pada kolom yang sama, maka angka tersebut akan disubstitusi dengan angka yang ada di bawah angka tersebut (bersifat siklik).

- Jika kedua angka tidak berada pada baris atau kolom yang sama:
    - Angka pertama diganti dengan perpotongan baris angka pertama dan kolom angka kedua
    - Angka kedua diganti dengan angka pada titik sudut keempat dari persegi panjang yang dibentuk dari ketiga angka.
- Apabila kedua angka sama, maka substitusi kedua angka dengan angka yang berada di sebelah kanan angka pada matriks.

Dekripsi

- Apabila angka pada sebuah pasangan berada pada baris yang sama, maka angka tersebut akan disubstitusi dengan angka yang ada di sebelah kiri angka tersebut (bersifat siklik).
- Apabila angka pada sebuah pasangan berada pada kolom yang sama, maka angka tersebut akan disubstitusi dengan angka yang ada di atas angka tersebut (bersifat siklik).
- Jika kedua angka tidak berada pada baris atau kolom yang sama:
    - Angka pertama diganti dengan perpotongan baris angka pertama dan kolom angka kedua
    - Angka kedua diganti dengan angka pada titik sudut keempat dari persegi panjang yang dibentuk dari ketiga angka.
- Apabila kedua angka sama, maka substitusi kedua angka dengan angka yang berada di sebelah kiri angka pada matriks.

Angka-angka yang didapat dari hasil *mapping* kemudian akan dikonversi menjadi sebuah matriks untuk dilakukan *shifting & rotating*.

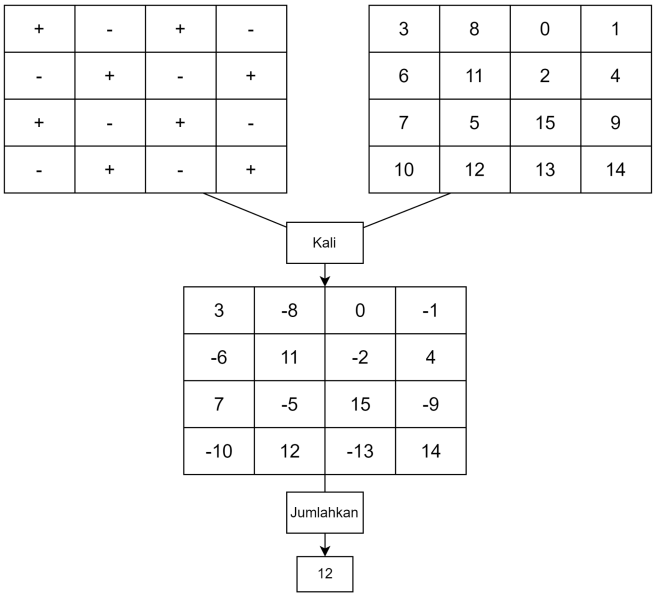


Gambar 3.6 *Playfair mapping*

2. *Shifting & rotating*

Sebelum dilakukan *shifting & rotating*, dilakukan terlebih dahulu penjumlahan dari setiap elemen pada kunci putaran. Elemen pada kunci putaran akan dikali dengan +1 atau -1 berdasarkan posisinya terlebih dahulu sebelum dilakukan penjumlahan. Aturan perkalian +1 atau -1 adalah sebagai berikut:

- Apabila indeks baris 0 atau 2, indeks kolom 0 dan 2 akan dikalikan dengan +1 dan indeks kolom 1 dan 3 akan dikalikan dengan -1.
- Apabila indeks baris 1 atau 3, indeks kolom 1 dan 3 akan dikalikan dengan +1 dan indeks kolom 0 dan 2 akan dikalikan dengan -1.

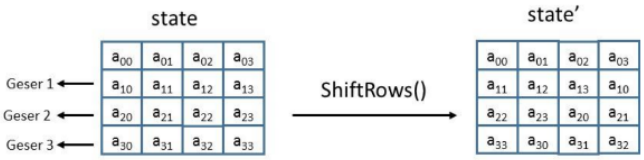


Gambar 3.7 Penjumlahan elemen kunci putaran

Proses *shifting & rotating* akan dilakukan berdasarkan apakah nilai penjumlahan lebih besar sama dengan 0 atau lebih kecil dari 0. Secara umum, akan dilakukan 2 proses pada tahap *shifting & rotating*, yaitu:

- *AES Row Shifting*

  Diterapkan proses *shifting* pada setiap baris sebagaimana algoritma *Rijndael* melakukan *row shifting* pada matriks. Baris kedua matriks akan digeser sebanyak satu elemen ke kiri, baris ketiga matriks digeser sebanyak dua elemen ke kiri, dan baris keempat matriks digeser sebanyak tiga elemen ke kiri. Pergeseran baris bersifat siklik.



Gambar 3.8 *Row shifting* pada algoritma *Rijndael*

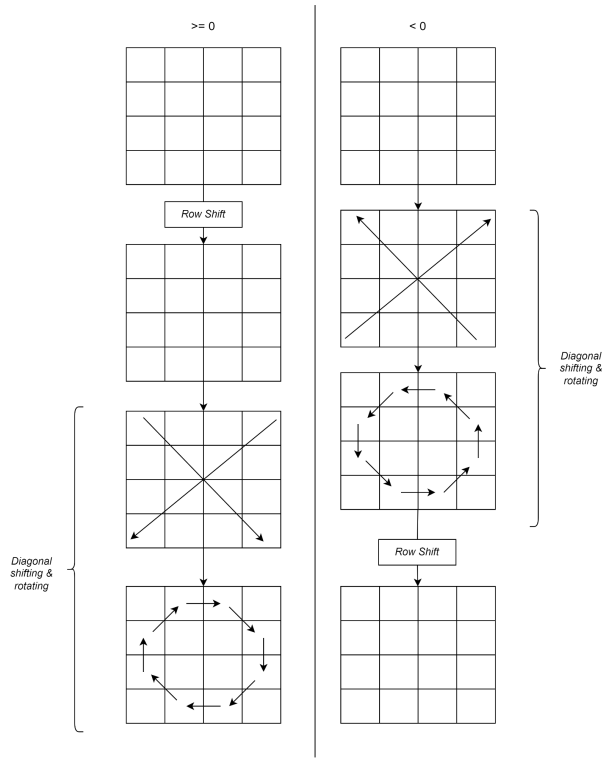
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/17-Beberapa-block-cipher-bagian3-2023.pdf

- *Diagonal shifting & rotating*

Dilakukan pergeseran secara diagonal sebanyak satu elemen terhadap matriks dengan arah yang ditentukan oleh nilai penjumlahan elemen kunci putaran. Apabila nilai total kunci putaran lebih besar sama dengan 0, maka pergeseran diagonal akan dilakukan dengan arah dari pojok kiri atas ke pojok kanan bawah dan pojok kanan atas ke pojok kiri bawah. Apabila nilai total kunci putaran lebih kecil dari 0, maka pergeseran diagonal akan dilakukan dengan arah dari pojok kanan bawah ke pojok kiri atas dan pojok kiri bawah ke pojok kanan atas. Pergeseran diagonal ini bersifat siklik.

Setelah dilakukan *diagonal shifting*, akan dilakukan rotasi sebesar satu elemen terhadap seluruh elemen yang tidak dipengaruhi oleh pergeseran diagonal (elemen yang tidak terletak pada diagonal matriks). Arah rotasi adalah searah jarum jam untuk nilai total kunci putaran yang lebih besar sama dengan 0, dan berlawanan arah jarum jam untuk nilai total kunci putaran yang lebih kecil dari 0.

Urutan dilakukannya kedua proses tersebut tergantung dengan nilai dari penjumlahan semua elemen pada kunci putaran. Apabila nilai tersebut lebih besar sama dengan 0, maka *AES row shifting* akan dilakukan terlebih dahulu sebelum dilakukannya *diagonal shifting & rotating*. Untuk nilai yang lebih kecil dari 0, maka *AES row shifting* akan dilakukan sesudah dilakukannya *diagonal shifting & rotating*.



Gambar 3.9 *Shifting & rotating* pada proses enkripsi

Untuk proses dekripsi, urutan proses pada tahap *shifting & rotating* serta arah putaran rotasi dan pergeseran diagonal merupakan kebalikan dari proses enkripsi bila dilihat dari nilai total kunci putaran, yaitu:

<u>>= 0</u>

- *Diagonal shifting & rotating* kemudian *row shifting*
- Arah putaran rotasi berlawanan arah jarum jam
- Arah pergeseran diagonal dari pojok kanan bawah ke pojok kiri atas dan pojok kiri bawah ke pojok kanan atas.

<u>< 0</u>

- *Row shifting* kemudian *diagonal shifting & rotating*
- Arah putaran rotasi searah jarum jam
- Arah pergeseran diagonal dari pojok kiri atas ke pojok kanan bawah dan pojok kanan atas ke pojok kiri bawah

IV. Eksperimen dan Pembahasan Hasil

A. Waktu Enkripsi dan Dekripsi

- Small

```
(venv) D:\Tugas-2-IF4020-Kriptografi>python -u "d:\Tugas-2-IF4020-Kriptografi\main.py"
Encryption Elapsed Time = 0.0009469985961914062 s

Decryption Elapsed Time = 0.0010001659393310547 s
```

Gambar 4.1 Eksperimen waktu untuk teks kecil

- Medium

```
(venv) D:\Tugas-2-IF4020-Kriptografi>python -u "d:\Tugas-2-IF4020-Kriptografi\main.py"
Encryption Elapsed Time = 0.023159027099609375 s

Decryption Elapsed Time = 0.023232698440551758 s
```

Gambar 4.2 Eksperimen waktu untuk teks medium

- Large

```
(venv) D:\Tugas-2-IF4020-Kriptografi>python -u "d:\Tugas-2-IF4020-Kriptografi\main.py"
Encryption Elapsed Time = 0.09986376762390137 s

Decryption Elapsed Time = 0.09796357154846191 s
```

Gambar 4.3 Eksperimen waktu untuk teks besar

- Very Large

```
(venv) D:\Tugas-2-IF4020-Kriptografi>python -u "d:\Tugas-2-IF4020-Kriptografi\main.py"
Encryption Elapsed Time = 1.234889030456543 s

Decryption Elapsed Time = 1.21048903465271 s
```

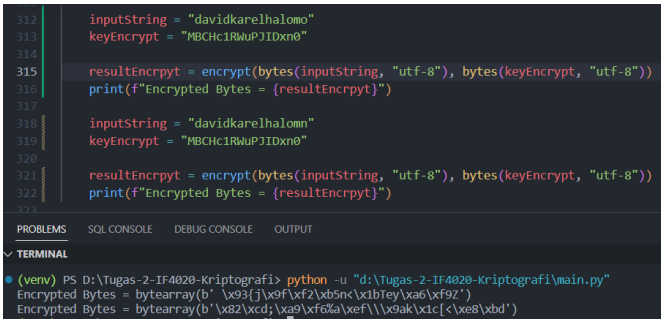
Gambar 4.4 Eksperimen waktu untuk teks sangat besar

Semua *test case* untuk algoritma Jarl dapat dilihat di [5]. Berdasarkan eksperimen yang dilakukan dengan beberapa ukuran *plaintext*, dapat diketahui bahwa waktu yang dibutuhkan untuk menjalankan algoritma Jarl relatif cukup singkat (dapat dijalankan dalam hitungan detik, bahkan milidetik untuk kasus yang kecil) sehingga dapat disimpulkan bahwa algoritma *block cipher* yang kami rancang cukup efisien dalam segi waktu.

B. Analisis Efek *Avalanche*

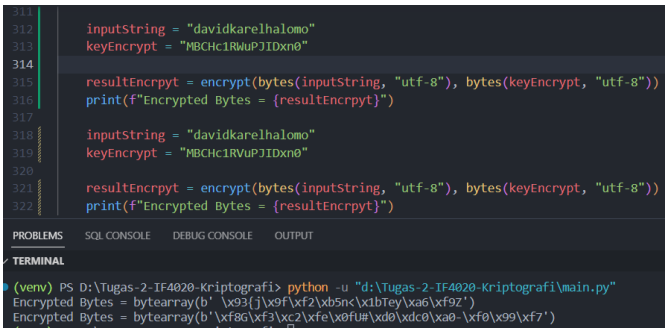
Untuk melakukan analisis efek *avalanche* pada algoritma, akan digunakan 2 *string*, "davidkarelhalomo" dan "davidkarelhalomn". Kedua string memiliki perbedaan 1 bit yaitu pada bit paling kanan sehingga dapat diketahui

perbedaan pada *ciphertext* yang didapat apabila dilakukan pergantian 1 bit pada *plaintext*.



Gambar 4.5 Eksperimen efek *avalanche* pada *plaintext*

Percobaan yang serupa juga dilakukan terhadap kunci. Dengan mengubah huruf W menjadi V, maka terjadi sebuah perubahan sebesar 1 bit.



Gambar 4.6 Eksperimen efek *avalanche* pada kunci

Dapat dilihat bahwa dengan digantinya salah satu bit baik pada kunci maupun *plaintext*, *ciphertext* yang didapat memiliki perbedaan yang sangat jauh. Hal tersebut dikarenakan *block cipher* yang dirancang menganut konsep *chaining* dimana blok yang didapatkan pada putaran *x* akan menjadi masukan bagi pembangkit kunci putaran *x+1* yang akan digunakan untuk menghasilkan blok pada putaran *x+1*. Jumlah putaran yang banyak (16) juga menjadi faktor bagi perbedaan yang jauh antar *testcase*.

C. Analisis Ruang Kunci

Panjang kunci untuk algoritma *block cipher* Jarl adalah 128 bit sehingga terdapat $2^{128} = 3{,}4 \times 10^{38}$ kemungkinan kunci. Apabila diasumsikan terdapat sebuah mesin yang dapat mencoba 1 juta kunci setiap detiknya, maka dibutuhkan waktu sekitar $5{,}4 \times 10^{24}$ tahun untuk mencoba semua kemungkinan kunci. Berdasarkan fakta tersebut, maka dapat ditarik kesimpulan bahwa ruang kunci algoritma Jarl sudah cukup baik untuk menghadapi *brute force attack*.

D. Analisis Keamanan Lainnya

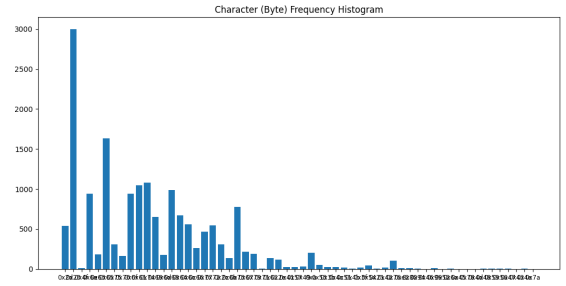
Bila meninjau prinsip Shannon dalam merancang kriptografi untuk menyembunyikan hubungan statistik antara *plaintext* dan *ciphertext*, maka:

- *Diffusion*

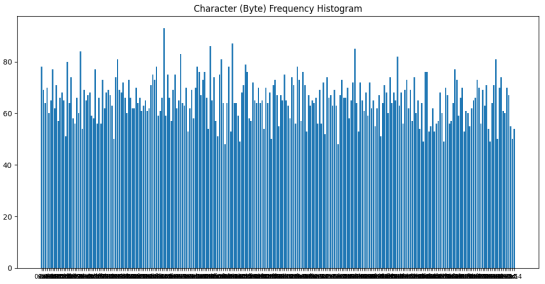
Penerapan prinsip *diffusion* terdapat pada tahap *shifting & rotating* yang melakukan permutasi dan transposisi bit masukan. Permutasi dan transposisi yang dilakukan pada tahap tersebut cukup rumit dan selain itu, tahap ini dilakukan sebanyak 16 kali, 1 kali setiap putaran. Sehingga dapat dikatakan bahwa algoritma Jarl sudah memenuhi prinsip *diffusion* dengan baik.

- *Confusion*

Penerapan prinsip *confusion* pada algoritma Jarl dapat ditemukan pada tahap *playfair mapping* dimana bit masukan disubstitusikan dengan bit yang lain berdasarkan kunci putaran. Kunci putaran juga berbeda untuk setiap putaran sehingga pada setiap putaran, pemetaan dan substitusi sebuah pasangan bit tidak selalu sama. Pada *playfair cipher* klasik, akan terlihat hubungan frekuensi bigram *plaintext* dan *ciphertext* karena menggunakan matriks *playfair* yang sama untuk seluruh *plaintext* maupun *ciphertext*. Algoritma Jarl mengatasi hal tersebut dengan menggunakan matriks yang berbeda pada setiap putaran sehingga masalah pada *playfair cipher* klasik dapat dimitigasi. Hal-hal tersebut menambahkan kompleksitas substitusi pada algoritma Jarl sehingga dapat dikatakan bahwa prinsip *confusion* sudah terpenuhi dengan baik.



Gambar 4.7 Histogram Frekuensi Karakter (byte) pada Teks Sangat Besar (Sebelum Enkripsi)



Gambar 4.8 Histogram Frekuensi Karakter (byte) pada Enkripsi Teks Sangat Besar

Dari histogram pada gambar 4.7, terlihat frekuensi persebaran karakter (*byte*) pada teks sangat besar (sebelum dienkripsi) tidak merata. Dari histogram pada gambar 4.8, terlihat frekuensi persebaran pada *ciphertext* (hasil enkripsi) teks sangat besar cukup tersebar dengan merata. Algoritma terbukti berhasil menyebarkan dan meratakan frekuensi karakter (*byte*) pada *ciphertext* untuk menyembunyikan hubungan statistik antara *plaintext* dengan *ciphertext* sehingga

mempersulit analisis frekuensi. Hal ini membuktikan algoritma sudah menerapkan prinsip *confusion* dan *diffusion* yang cukup baik.

V. Kesimpulan dan Saran

Algoritma Jarl merupakan sebuah algoritma *block cipher* menerima kunci berupa *string* dengan panjang 128 bit saat dikonversi ke tipe data *bytes* dan membagi *plaintext* atau *ciphertext* menjadi blok berukuran 128 bit. Algoritma ini menggunakan jaringan Feistel, melakukan perulangan enkripsi sebanyak 16 putaran, dan melakukan substitusi dan permutasi dengan cara yang cukup kompleks dan berbeda untuk setiap kunci dan/atau *plaintext* yang berbeda. Waktu yang dibutuhkan algoritma untuk melakukan enkripsi maupun dekripsi cukup *feasible* dan algoritma tahan akan serangan *brute force*. Algoritma juga sudah menerapkan prinsip *diffusion* dan *confusion* dengan baik sehingga tahan akan serangan analisis frekuensi.

Proses substitusi algoritma Jarl menggunakan cara yang terinspirasi dari *playfair cipher*, dengan perbedaan apabila terdapat sebuah bigram dengan angka yang sama, maka angka akan dienkripsi menjadi hal yang sama juga. Implementasi yang dilakukan oleh penulis pada [5] hanya dapat menerima masukan *plaintext* yang berkelipatan 16 karakter. Penulis menyarankan kedepannya untuk mengimplementasikan program yang dapat menerima *plaintext* dengan ukuran apa saja. Saran lainnya adalah melakukan berbagai jenis serangan terhadap algoritma Jarl untuk mengetahui kelemahan algoritma Jarl sehingga algoritma dapat ditingkatkan lagi kekuatannya berdasarkan *insight* yang didapat dari penyerangan.

Daftar Referensi

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/13-Block-Cipher-Bagian1-2023.pdf

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/04-Kriptografi-Klasik-Bagian3-(2023).pdf

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/17-Beberapa-block-cipher-bagian3-2023.pdf.

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/14-Prancangan-block-cipher-2023.pdf

[5] https://github.com/davidkarelhp/Tugas-2-IF4020-Kriptografi