Wonderful Journey Block Cipher

Joshua Christo Randiny, **Willy Santoso**.

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung (ITB), Jalan Ganesha 10, Bandung 40132
E-mail: 13517063@std.stei.itb.ac.id, 13517066@std.stei.itb.ac.id

Abstrak - *Block cipher* adalah salah satu teknik kriptografi yang melakukan enkripsi dengan ukuran blok tertentu. Untuk membangun algoritma *block cipher* yang aman diperlukan beberapa prinsip yang perlu dipenuhi. *Wonderful Journey Block Cipher* merupakan salah satu algoritma *block cipher* baru dan aman yang berbasis jaringan Feistel dan menerapkan prinsip-prinsip *confusion* dan *diffusion* pada proses enkripsinya. Keunikan dari *Wonderful Journey Block Cipher* adalah algoritma enkripsi dengan konsep perjalanan dan pengalaman pada tiap perhentiannya.

Keywords: *block cipher*, kriptografi, aman, enkripsi, jaringan Feistel, *Wonderful Journey Block Cipher*.

1. Pendahuluan

Dunia modern yang ada sekarang tidak mungkin tercipta tanpa teknologi enkripsi. Teknologi enkripsi mendasari berbagai fasilitas yang dapat manusia nikmati sekarang ini. Untuk melakukan enkripsi dan dekripsi tersebut dibutuhkan sebuah algoritma yang baik. Sekarang ini ada banyak algoritma untuk melakukan enkripsi tersebut, salah satu kelas dari teknologi tersebut adalah *block cipher*. Namun, beberapa algoritma *block cipher* yang sudah ada masih memiliki tingkat keamanan yang kurang baik, begitu juga dengan hasil enkripsinya masih dapat dipecahkan oleh kriptanalis dengan berbagai cara. Maka dari itu, dibutuhkan sebuah rancangan algoritma baru yang lebih baik dan aman. Terdapat beberapa prinsip-prinsip yang perlu dipenuhi dalam merancang suatu algoritma *block cipher*. *Wonderful Journey Block Cipher* (WJBC) dirancang untuk membentuk algoritma *block cipher* yang baik dan aman. Dengan penggunaan banyak operasi blok seperti *shit* dan XOR menjadi keunikan pada WJBC serta menambah kompleksitas algoritmanya. Untuk lebih jelasnya akan dijelaskan pada bagian berikut.

2. Studi Pustaka

Pada bagian akan dijelaskan mengenai teori-teori dasar pada *block cipher*, mulai dari definisi singkat, prinsip-prinsip dalam perancangan *block cipher*, dan mode operasi pada *block cipher*.

2.1. Block Cipher

Block cipher adalah metode enkripsi yang dilakukan untuk setiap blok bit pada plainteks dengan suatu kunci blok dengan ukuran tertentu. Blok yang dimaksud adalah kumpulan bit yang akan dijadikan sebagai satu kesatuan untuk melakukan operasi enkripsinya. Tujuan untuk pengumpulan ini adalah untuk meningkatkan keamanan karena perubahan 1 bit dalam 1 blok dapat mempengaruhi hasil dalam 1 blok atau lebih. Berbeda dengan stream cipher yang setiap perubahan biasanya tidak disebarkan ke tempat-tempat lain.

2.2. Prinsip-prinsip perancangan block cipher

Terdapat beberapa prinsip yang harus diperhatikan dalam merancang suatu *block cipher*. Prinsip-prinsip tersebut diharapkan dapat memberikan arahan agar hasil algoritma *block cipher* menjadi lebih kompleks dan lebih aman. Semakin kompleks dari algoritma *block* cipher, maka akan semakin sulit bagi kriptanalis untuk memecahkan cipherteks yang dihasilkan. Terdapat dua prinsip dalam perancangan algoritma kriptografi untuk *block cipher*, yaitu :

- *Diffusion* (Penyebaran)

  Prinsip diffusion menyangkut dengan ketergantungan antara bit-bit plainteks dan cipherteks. Setiap perubahan bit pada plainteks idealnya mengubah minimal 50 persen dari cipherteks. Sesuai dengan namanya, prinsip ini menyebarkan pengaruh perubahan. Dalam *block cipher* prinsip ini biasa diaplikasikan dalam 1 bloknya. Dengan menerapkan prinsip ini maka kriptanalis akan kesulitan dalam memecahkan algoritma enkripsi. Teknik yang paling umum digunakan untuk menerapkan prinsip *diffusion* adalah dengan menggunakan operasi permutasi atau transposisi. Dengan operasi permutasi yang sulit maka kompleksitas dari algoritma enkripsi tersebut akan bertambah. Namun, dengan semakin banyaknya operasi permutasi akan menambah kinerja dari komputer untuk melakukan perhitungan.

- *Confusion* (Membingungkan)

  Prinsip *confusion* menyangkut hubungan antara bit-bit plainteks dan cipherteks. Tujuan dari prinsip ini adalah menyulitkan kriptanalis untuk mencari pola statistik dari cipherteks. Hal ini dilakukan dengan meminimalisir keterhubungan antara plainteks, cipherteks, dan kunci enkripsi yang digunakan. Dengan prinsip *confusion,* perhitungan statistik hasil analisis dari cipherteks idealnya mempunyai nilai yang kurang lebih sama untuk tiap karakternya. Teknik yang paling umum digunakan untuk menerapkan prinsip *confusion* pada perancangan algoritma adalah dengan menggunakan operasi substitusi. Operasi substitusi bisa diterapkan pada banyak hal, seperti substitusi untuk setiap posisi *bit*, *byte*, atau *hexadecimal* dari plainteks, cipherteks, maupun kunci enkripsi. Semakin kompleks operasi substitusinya akan semakin sulit bagi pihak ketiga untuk melakukan kriptanalisis.

2.3. Iterated Cipher

Sesuai dengan namanya, *Iterated cipher* merupakan teknik enkripsi yang dilakukan berulang-ulang. Biasanya satuan perulangan untuk *iterated cipher* disebut dengan putaran. *Iterated cipher* atau cipher berulang menghasilkan algoritma enkripsi blok dengan prinsip *confusion dan diffusion* sekaligus. Pada setiap putaran, dibutuhkan kunci putaran yang berbeda. Secara matematis, fungsi enkripsi yang dilakukan adalah :

$$C_i = f(C_{i-1}, K_i)$$

Pada persamaan di atas, C_i adalah hasil proses satu putaran, i adalah nilai jumlah putaran, K_i adalah kunci pada putaran ke-i, dan f adalah fungsi transformasi yang dilakukan terhadap blok. Pada iterated cipher, plainteks didefinisikan dengan C_0, sedangkan cipherteks hasil enkripsi yang dihasilkan setelah *n* iterasi didefinisikan dengan C_n.

2.4. Jaringan Feistel

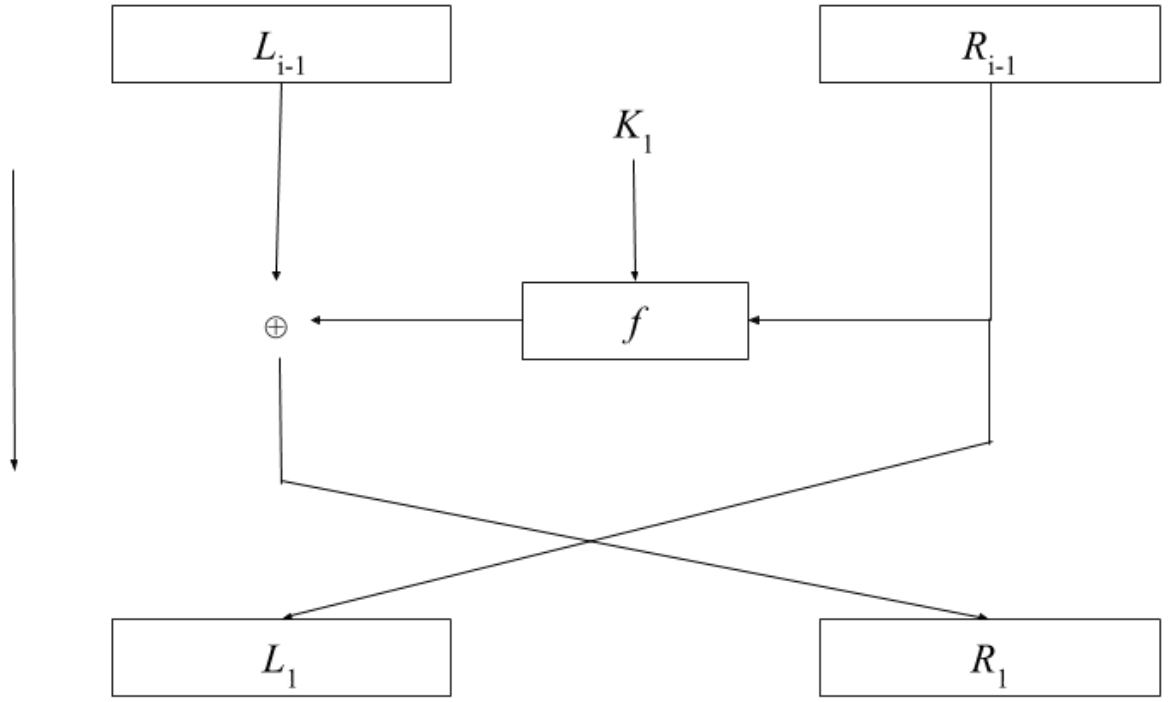
Jaringan Feistel merupakan suatu struktur operasi yang dilakukan pada algoritma enkripsi blok yang unik dengan sekaligus menerapkan prinsip *diffusion* dan *iterated* cipher. Jaringan Feistel pertama kali dikemukakan oleh Host Feistel pada tahun 1973. Kebanyakan dari algoritma enkripsi blok yang sudah ada menerapkan prinsip dari jaringan Feistel ini. Salah satu keuntungan dari penerapan jaringan Feistel untuk merancang algoritma enkripsi blok adalah fungsi transformasi pada jaringan Feistel bersifat *reversible* sehingga tidak diperlukan untuk membuat algoritma dekripsi baru untuk mengembalikan pesan dari cipherteks menjadi plainteks. Algoritma dekripsi dapat menggunakan fungsi transformasi yang dilakukan untuk proses enkripsi. Berikut adalah model jaringan Feistel yang umum dipakai :

a. Blok plainteks yang digunakan sebagai *input* dibagi menjadi dua bagian, kiri (L) dan kanan (R), dengan masing-masing bagian berukuran setengah dari *input* blok sehingga perlu diperhatikan bahwa ukuran dari *input* blok harus bernilai genap.
b. Mendefinisikan *iterated cipher* dengan hasil putaran ke-*i* akan bergantung pada hasil putaran sebelumnya.

Secara matematis, terdapat dua fungsi persamaan yang dilakukan, yaitu :

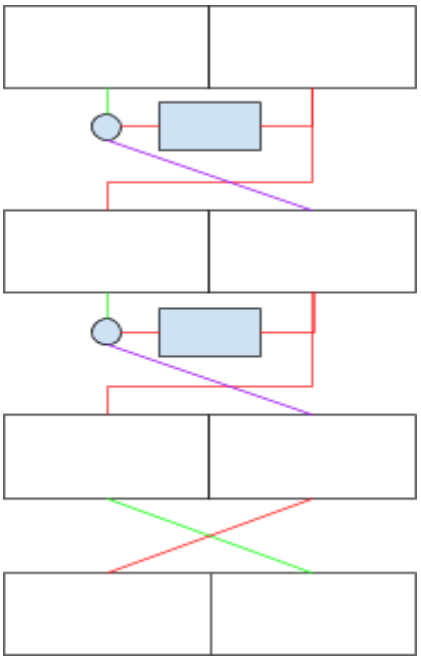
$$L_i = R_{i-1}$$

$$R_{i-1} = L_{i-1} \oplus f(R_{i-1}, K_i)$$



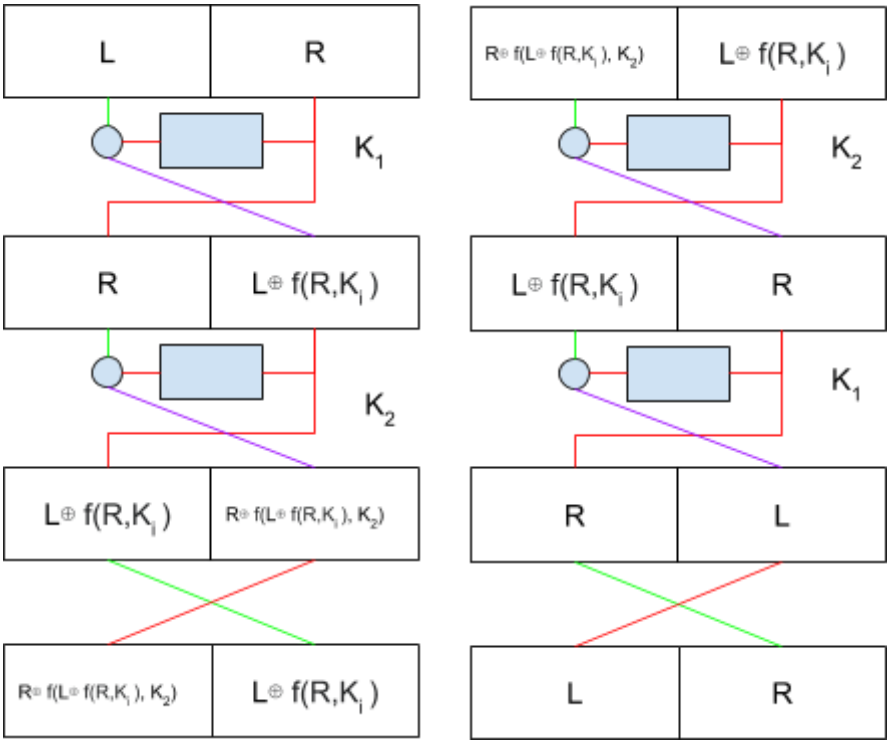
Gambar 2.1. Jaringan Feistel pada enkripsi putaran ke-i

Pada persamaan diatas, *i* adalah nilai jumlah putaran, *K* adalah kunci internal pada putaran ke-*i,* dan *f* adalah fungsi transformasi yang dilakukan terhadap blok. Kunci internal yang digunakan pada setiap putaran didapatkan pembangkitan kunci menggunakan kunci eksternal-nya. Pada jaringan Feistel, plainteks didefinisikan dari penggabungan antara bagian *L* dan *R* awal, dinyatakan dengan (L_0, R_0), sedangkan cipherteks didefinisikan dari penggabungan antara bagian R akhir dengan L akhir, dinyatakan dengan (L_f, R_f). Perlu diperhatikan juga, agar sifat *reversible* dari jaringan ini berefek, pada tahap terakhir perlu ada penanganan khusus yaitu dibalik. Jaringan akan menjadi seperti gambar berikut.



Gambar 2.2. Jaringan Feistel penuh

Jika dilakukan perhitungan matematis maka dapat dibuktikan bahwa jaringan tersebut *reversible*. Berikut adalah hasil enkripsi dan dekripsi kembali dari sebuah input LR. Jangan lupa untuk membalik urutan kunci saat melakukan dekripsi



Gambar 2.3. Contoh operasi perhitungan pada jaringan Feistel

2.5. Kotak-S

Kotak-S adalah suatu komponen yang menerima input *m bit* lalu mengeluarkan *m bit*, sehingga matriks kotak-S yang terbentuk akan berukuran *m x n*. Kotak-S biasa diimplementasikan dengan matriks yang digunakan sebagai metode operasi substitusi. Matriks tersebut digunakan untuk pemetaan posisi dari setiap bit atau satuan bloknya. Operasi yang dilakukan dengan kotak-s cukup sederhana karena hanya melakukan operasi *look-up* terhadap matriks yang sudah didefinisikan sebelumnya. Masukan dari kotak-S akan menyatakan posisi baris dan kolom dari matriks, kemudian keluarannya didapatkan dari nilai elemen yang ada pada matriks dengan posisi tersebut.

Contoh proses *look-up* yang dilakukan pada algoritma *Data Encryption Standard* (DES). Kotak-S didefinisikan dengan matriks berukuran *6 x 4*, yang artinya kotak-S menerima masukan sebanyak 6 *bit* dan kemudian akan mengembalikan nilai 4 *bit* keluaran. Contoh matriks isi kotak-S yang digunakan adalah sebagai berikut :

Tabel 2.1. Contoh kotak-S dengan ukuran *6x4*

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
8	4	15	5	3	7	13	6	2	10	12	14	0	9	11	1
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8

Baris memiliki indeks antara *0* hingga *3*, sedangkan kolom memiliki indeks antara *0* hingga *15*. Masukan untuk kotak-S adalah data berukuran *6 bit*, yang secara struktur adalah *b1b2b3b4b5b6*. Kemudian indeks baris dari matriks dinyatakan dengan *string bit* dari *b1b6* dan indeks kolom dinyatakan dengan *string bit* dari *b2b3b4b5*.

Sebagai contoh, masukan terhadap kotak-S adalah *011101*. Indeks baris didapatkan dari *b1b6* menghasilkan *string bit 01*, yang bernilai *1*, sedangkan indeks kolom didapatkan dari *b2b3b4b5* menghasilkan *string bit 1110*, yang bernilai *14*. Maka keluaran dari kotak-S adalah nilai elemen dari matriks pada indeks *(1,14),* yaitu *2* menjadi *0010*.

2.6. Kotak-P

Kotak-P atau yang biasa disebut dengan matriks permutasi adalah matriks berisi nilai yang akan menyatakan indeks dari kumpulan blok yang digunakan pada *block cipher*. Kotak-P mirip dengan kotak-S tetapi pemetaan pada kotak-P langsung langsung berdasarkan indeks terurut, sedangkan kotak-S menggunakan bagian-bagian *bit* tertentu. Ukuran *bit* yang masuk ke kotak-P akan berukuran sama dengan keluaran *bit*-nya. Kotak-P digunakan dengan tujuan untuk menerapkan prinsip *diffusion* atau penyebaran pada algoritma *block cipher*. Pada umumnya, elemen pada kotak-P biasanya dibangkitkan dengan metode acak atau *random*.

Contoh proses *look-up* yang dilakukan pada algoritma *Data Encryption Standard* (DES). Kotak-P didefinisikan dengan matriks berukuran *2x8*. Contoh matriks isi kotak-P yang digunakan adalah sebagai berikut :

Tabel 2.2. Contoh kotak-P dengan ukuran 2x8

14	0	5	12	3	15	7	2
6	10	13	1	8	11	4	9

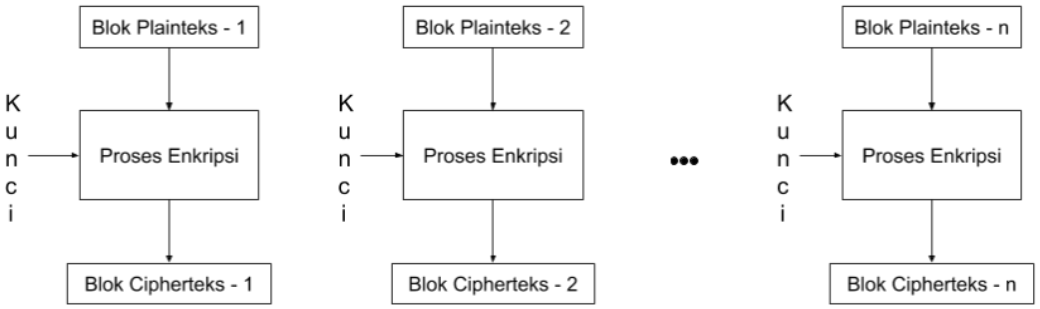
Setiap nilai pada matriks menentukan posisi bit hasil pemetaannya. Maka makna dari kotak-P di atas adalah *bit* ke-14 akan berpindah ke-0, *bit* ke 0 akan berpindah pada *bit* ke-1, dan seterusnya. Misalkan masukan kotak-P berupa blok pesan sebesar 16-*bit* adalah sebagai berikut *01100101 01100101*. Maka hasil permutasi dengan matriks kotak-P tersebut adalah *01000111 01110001*.

2.7. Mode Operasi Block Cipher

Mode operasi pada *block cipher* menyatakan bagaimana *block cipher* melakukan proses operasi terhadap blok plainteks atau blok cipherteks-nya. Terdapat setidaknya tiga buat *mode* operasi pada *block cipher* yang juga digunakan untuk pengujian yang dilakukan, yaitu *Electronic Code Block* (ECB), *Cipher Block Chaining* (CBC), dan *mode Counter*.

2.7.1. Electronic Code Block (ECB)

Pada mode *Electronic Code Block* atau biasanya disingkat menjadi ECB, *block cipher* melakukan operasi enkripsi dan dekripsi secara individual untuk setiap bloknya sehingga setiap proses blok bersifat independen dengan blok lainnya.



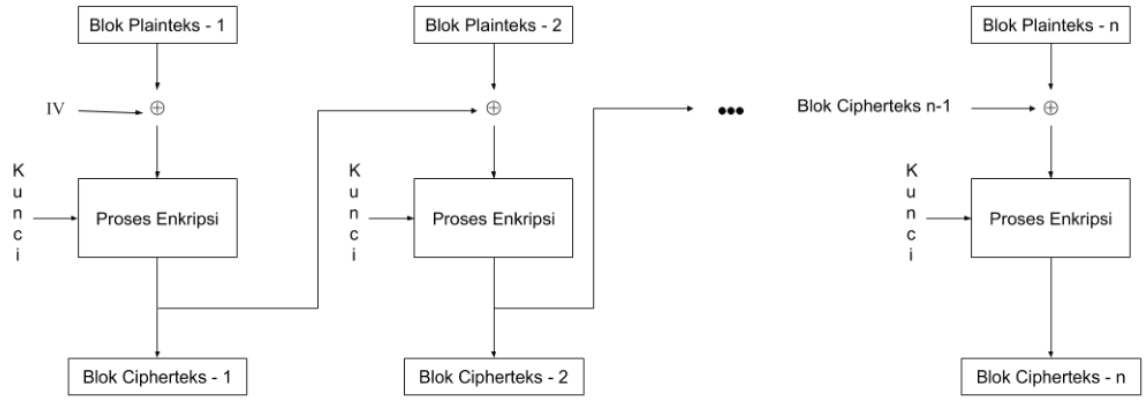
Gambar 2.4 Skema enkripsi *block cipher* dengan mode ECB

Terdapat beberapa keuntungan dari penggunaan mode ECB untuk proses *block cipher*. Karena operasi enkripsi dan dekripsi dilakukan secara individual untuk setiap bloknya, maka untuk proses tersebut tidak diharuskan untuk dilakukan secara sekuensial. Proses enkripsi maupun dekripsi dapat dilakukan secara paralel sehingga dapat mempercepat waktu eksekusi dari algoritma *block cipher* tersebut. Keunggulan lainnya dari penggunaan mode ECB adalah jika terjadi kesalahan operasi atau perhitungan pada proses enkripsi satu blok tidak akan mempengaruhi blok-blok lainnya karena sesuai dengan sifat independen dari suatu blok dengan blok-blok lainnya.

Namun, terdapat beberapa kelemahan pula yang muncul jika suatu *block cipher* menggunakan *mode* ECB. Setiap blok pada plainteks mungkin saja memiliki bagian yang sama karena terdapat pengulangan seperti kata pada pesan tersebut. Dengan menggunakan mode ECB, bagian perulangan tersebut dapat menghasilkan blok-blok cipherteks yang sama, sehingga kriptanalis mungkin dapat menemukan pola-pola berulang yang akan membantunya untuk memecahkan plainteks dari cipherteksnya. Serangan yang paling mudah digunakan jika diketahui bahwa struktur pada cipherteks membentuk pola berulang adalah dengan menggunakan *known-plaintext attack*. Tetapi, penggunaan ukuran blok yang besar untuk *block cipher* dapat mengurangi kemungkinan untuk menghasilkan pola blok yang berulang.

2.7.2. Cipher Block Chaining (CBC)

Pada mode *Cipher Block Chaining* atau biasanya disingkat menjadi CBC, sebelum *block cipher* melakukan enkripsi suatu blok, blok tersebut akan dioperasikan dengan hasil enkripsi blok sebelumnya. Dengan demikian, proses enkripsi suatu blok akan mempengaruhi dengan blok-blok setelahnya seperti membentuk rantai penghubung. Proses operasi yang dilakukan adalah melakukan operasi XOR antara blok plainteks dengan blok hasil enkripsi dari blok sebelumnya. Namun, karena blok pertama tidak memiliki hasil enkripsi blok sebelumnya, maka dibutuhkan blok khusus yang akan dilakukan operasi XOR dengan blok plainteks pertama. Blok khusus tersebut dinamakan dengan *initialization vector* (IV), dan biasanya adalah blok yang dibangkitkan secara acak dan bersifat publik.

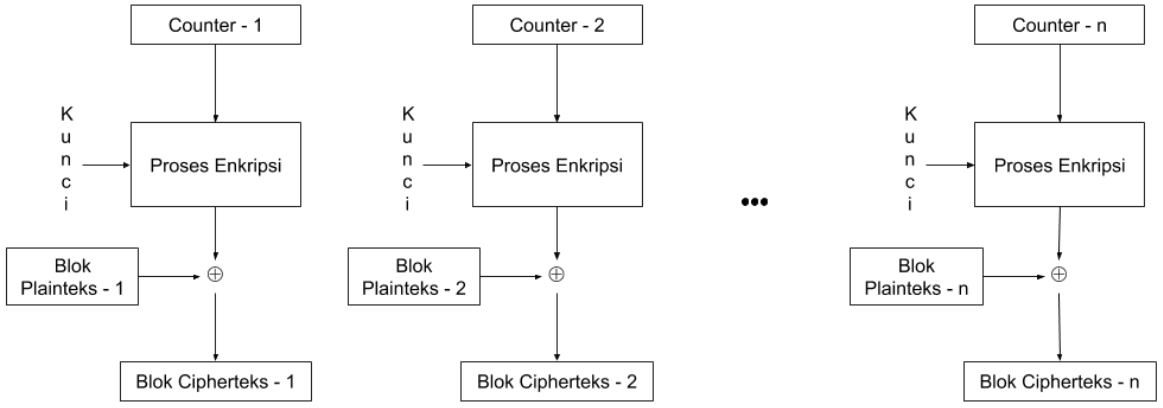


Gambar 2.5. Skema enkripsi *block cipher* dengan mode CBC

Dengan menggunakan mode CBC, walaupun terdapat blok-blok yang sama pada plainteks akan menghasilkan blok cipherteks yang berbeda. Hal ini yang menjadi keunggulan serta menyelesaikan persoalan pada kelemahan *block cipher* jika menggunakan mode ECB. Terdapat beberapa kelemahan pada mode CBC, yaitu jika terjadi kesalahan pada proses enkripsi satu blok, sudah dipastikan kesalahan tersebut akan berdampak pada blok-blok setelahnya. Selain itu, pihak ketiga bisa saja menambahkan blok cipherteks di bagian akhir tanpa terdeteksi, yang akan berdampak bertambahnya blok plainteks di bagian akhir. Hal ini mengakibatkan pesan yang didapatkan oleh penerima sudah termanipulasi, dengan demikian pengirim pesan sebaiknya memberikan catatan atau penanda untuk awal dan akhir dari blok teksnya agar dapat mendeteksi jika ada penambahan blok oleh pihak ketiga.

2.7.3. Mode Counter

Mode *counter* pertama kali dikemukakan oleh Diffie dan Hellman pada tahun 1979. Mode *counter* melakukan operasi enkripsi dan dekripsi pada sebuah blok plainteks, dilakukan secara independen dan tidak bertanggung pada blok-blok lainnya sehingga Mode *counter* tidak membentuk rantai blok seperti mode CBC. Dari mekanisme tersebut, menggambarkan bahwa mode *counter* mirip dengan mode ECB, namun terdapat modifikasi yang membedakan keduanya ialah proses enkripsinya dan juga terdapat blok tambahan baru yang dinamakan blok *counter*. Blok *counter* memiliki ukuran blok yang sama dengan plainteks yang berisi suatu nilai tertentu. Pada proses enkripsi blok plainteks pertama, akan dilakukan inisiasi awal *block counter* dengan nilai konstanta tertentu dan untuk setiap proses enkripsi blok-blok plainteks berikutnya nilai dari blok *counter* tersebut akan bertambah secara inkremental. Dengan demikian, walaupun proses enkripsi setiap blok dilakukan secara independen, tetapi bergantung dengan nilai blok *counter*-nya. Akibat dari nilai blok *counter* yang berbeda-beda, blok plainteks yang sama akan menghasilkan blok cipherteks yang berbeda.



Gambar 2.6 Skema enkripsi *block cipher* dengan mode counter

Proses yang dilakukan pada mode counter berbeda dengan mode ECB. Pada mode *counter*, untuk proses pada setiap bloknya, blok counter akan dilakukan enkripsi menggunakan fungsi enkripsi dan kunci tertentu dan kemudian dilakukan operasi XOR dengan blok plainteks untuk menghasilkan blok cipherteks-nya. Mode counter sudah banyak digunakan dalam aplikasi keamanan tertentu seperti keamanan pada jaringan ATM dan IP.

2.8. *Padding*

Karena *block cipher* membutuhkan *input* dengan ukuran yang tetap maka jika *input* ukurannya kurang perlu dilakukan proses *padding* atau penambahan data yang biasa dilakukan di akhir. Metode *padding* beragam tergantung algoritma yang digunakan. Salah satu standar yang ada adalah *PKCS#7*. Dalam standar tersebut *padding* dilakukan dengan mengisi x *byte* di akhir dengan x. Misalkan untuk data yang kurang 6 *byte* pada boks berukuran 16 *byte* maka 6 *byte* di akhir yang kosong akan diisi dengan angka 6. Untuk mengurangi keambiguan maka jika *input* habis pas pada batas blok tetap akan dikorbankan satu blok tambahan yang akan diisi dengan *padding* saja.

3. Rancangan Algoritma

Hasil rancangan algoritma *block cipher* yang dibuat dinamakan *Wonderful Journey* atau bisa disingkat menjadi WJBC (*Wonderful Journey Block Cipher)*. WJBC menggunakan struktur jaringan Feistel pada algoritma perancangan *block cipher* sehingga tidak memerlukan algoritma dekripsi yang berbeda dengan enkripsi. Ukuran blok yang digunakan pada WJBC adalah 128 bit dan beroperasi dalam *byte.* WJBC menyediakan tiga opsi mode *block cipher,* yaitu *Electronic Code Block* (ECB), *Cipher Block Chaining* (CBC), dan *counter*. Untuk *padding* digunakan skema seperti yang terdapat pada *PKCS#7*.

3.1. Ide

Nama *Wonderful Journey* diambil dari ide perjalanan (*journey*) yang biasa kalau dalam cerita-cerita akan berlangsung dengan pengalaman unik di tiap perhentiannya. Algoritma yang dibuat juga kurang lebih dibuat seperti itu. Berjalan direpresentasikan dengan operasi pergeseran ke kiri sebanyak *1 byte* dan pengalaman yang dimaksud adalah sebuah operasi yang akan dilaksanakan di tiap antara pergeseran.

3.2. Key Scheduling

Untuk membuat kunci pada algoritma ini digunakan kunci sebelumnya yang digeser sebanyak x *byte* ke kanan, dengan x adalah nilai *byte* pertama dari kunci sebelumnya. Kemudian akan diambil 16 *byte* pertama (128 bit) untuk dijadikan *round key*. Pada awalnya nilai “kunci sebelumnya” adalah nilai kunci masukkan. Kunci masukkan pertama harus berukuran minimal 16 *byte* agar algoritma dapat bekerja.

3.3. Perjalanan

Berikut adalah proses yang dilakukan pada fungsi transformasi pada jaringan Feistel. Setidaknya ada sembilan operasi yang dilakukan pada blok, hal ini dikarenakan agar hasil perancangan semakin kompleks. Dengan algoritma yang kompleks maka akan semakin sulit bagi kriptanalis untuk memecahkan algoritma dekripsi tersebut. Sembilan fungsi transformasi yang dilakukan secara terurut adalah :

1) Berjalan (Pergeseran ke kiri seluruh blok sebanyak 1 *byte*)
2) Pengalaman 1 (Operasi XOR dengan 8 *byte* pertama kunci)
3) Berjalan (Pergeseran ke kiri seluruh blok sebanyak 1 *byte*)
4) Pengalaman 2 (Operasi substitusi dengan Kotak-S)
5) Berjalan (Pergeseran ke kiri seluruh blok sebanyak 1 *byte*)
6) Pengalaman 3 (Operasi XOR dengan 8 *byte* terakhir kunci)
7) Berjalan (Pergeseran ke kiri seluruh blok sebanyak 1 *byte*)
8) Pengalaman 4 (Operasi XOR dengan *byte* berikutnya)
9) Berjalan (Pergeseran ke kiri seluruh blok sebanyak 1 *byte*)

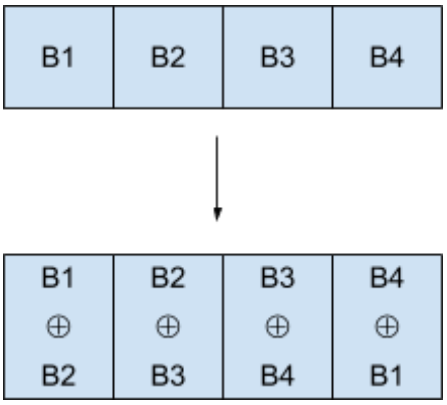
Perjalanan di atas akan dilaksanakan sebanyak 16 *round* pada sebuah jaringan Feistel. Karena sifat jaringan Feistel maka *input* dari perjalanan di atas hanyalah setengah dari ukuran blok yaitu 64 bit. Karena panjangnya yang hanya 64 bit sebuah kunci 128 bit dapat dipecah dua dan digunakan untuk operasi XOR di dua tempat yaitu pada tahap 2 dan 6.

Untuk Kotak-S yang akan digunakan pada algoritma ini adalah kotak-S yang sama dengan algoritma AES sebagai berikut

Tabel 3.1 Kotak-S AES

0x63	0x7C	0x77	0x7B	0xF2	0x6B	0x6F	0xC5	0x30	0x01	0x67	0x2B	0xFE	0xD7	0xAB	0x76
0xCA	0x82	0xC9	0x7D	0xFA	0x59	0x47	0xF0	0xAD	0xD4	0xA2	0xAF	0x9C	0xA4	0x72	0xC0
0xB7	0xFD	0x93	0x26	0x36	0x3F	0xF7	0xCC	0x34	0xA5	0xE5	0xF1	0x71	0xD8	0x31	0x15
0x04	0xC7	0x23	0xC3	0x18	0x96	0x05	0x9A	0x07	0x12	0x80	0xE2	0xEB	0x27	0xB2	0x75
0x09	0x83	0x2C	0x1A	0x1B	0x6E	0x5A	0xA0	0x52	0x3B	0xD6	0xB3	0x29	0xE3	0x2F	0x84
0x53	0xD1	0x00	0xED	0x20	0xFC	0xB1	0x5B	0x6A	0xCB	0xBE	0x39	0x4A	0x4C	0x58	0xCD
0xD0	0xEF	0xAA	0xFB	0x43	0x4D	0x33	0x85	0x45	0xF9	0x02	0x7F	0x50	0x3C	0x9F	0x18
0x51	0xA3	0x40	0x8F	0x92	0x9D	0x38	0xF5	0xBC	0xB6	0xDA	0x21	0x10	0xFF	0xF3	0xD2
0xCD	0x0C	0x13	0xEC	0x5F	0x97	0x44	0x17	0xC4	0xA7	0x7E	0x3D	0x64	0x5D	0x19	0x73
0x60	0x81	0x4F	0xDC	0x22	0x2A	0x90	0x88	0x46	0xEE	0xB8	0x14	0xDE	0x5E	0x0B	0xDB
0xE0	0x32	0x3A	0x0A	0x49	0x06	0x24	0x5C	0xC2	0xD3	0xAC	0x62	0x91	0x95	0xE4	0x79
0xE7	0xC8	0x37	0x6D	0x8D	0xD5	0x4E	0xA9	0x6C	0x56	0xF4	0xEA	0x65	0x7A	0xAE	0x08
0xBA	0x78	0x25	0x2E	0x1C	0xA6	0xB4	0xC6	0xE8	0xDD	0x74	0x1F	0x4B	0xBD	0x8B	0x8A
0x70	0x3E	0xB5	0x66	0x48	0x03	0xF6	0x0E	0x61	0x35	0x57	0xB9	0x86	0xC1	0x1D	0x9E
0xE1	0xF8	0x98	0x11	0x69	0xD9	0x8E	0x94	0x9B	0x1E	0x87	0xE9	0xCE	0x55	0x28	0xDF
0x8C	0xA1	0x89	0x0D	0xBF	0xE6	0x42	0x68	0x41	0x99	0x2D	0x0F	0xB0	0x54	0xBB	0x16

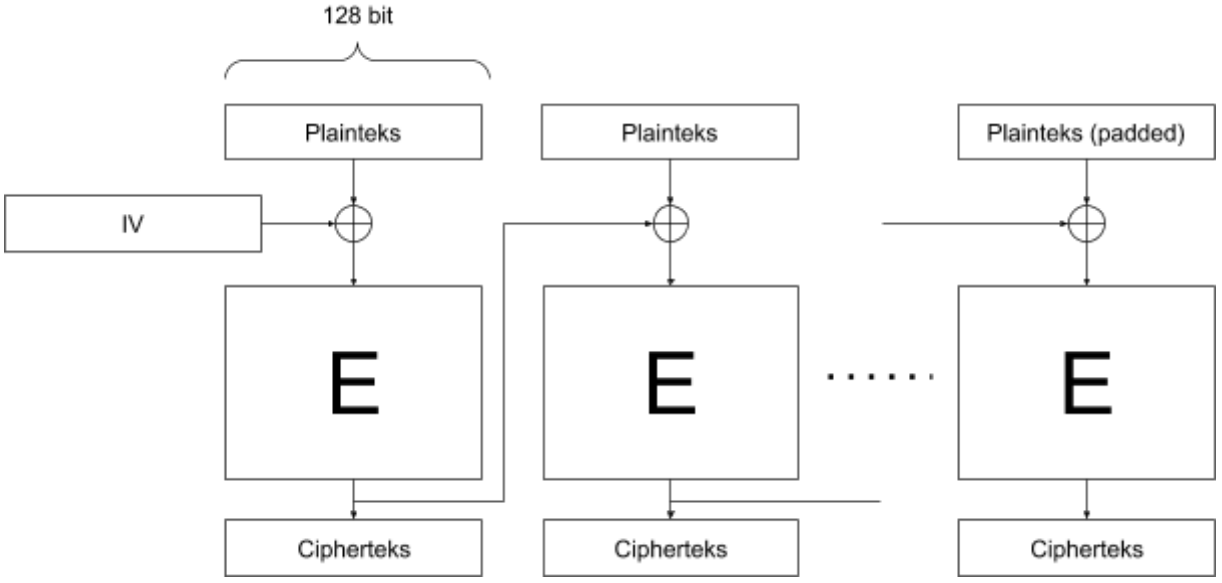
Untuk tahapan 8 yang dimaksud adalah mengisi *byte* ke-*i* dengan hasil XOR *byte* ke-*i* dengan *i+1*. Jika i+*1* melebihi blok maka akan di-modulus dengan ukuran blok. Jika digambarkan kurang lebih seperti berikut



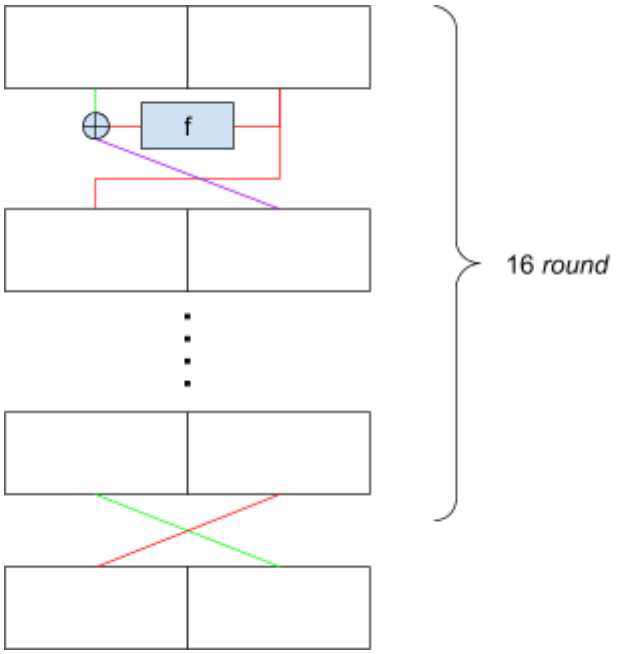
Gambar 3.1. Penjelasan tahap 8 dari algoritma

3.4. Proses Penuh

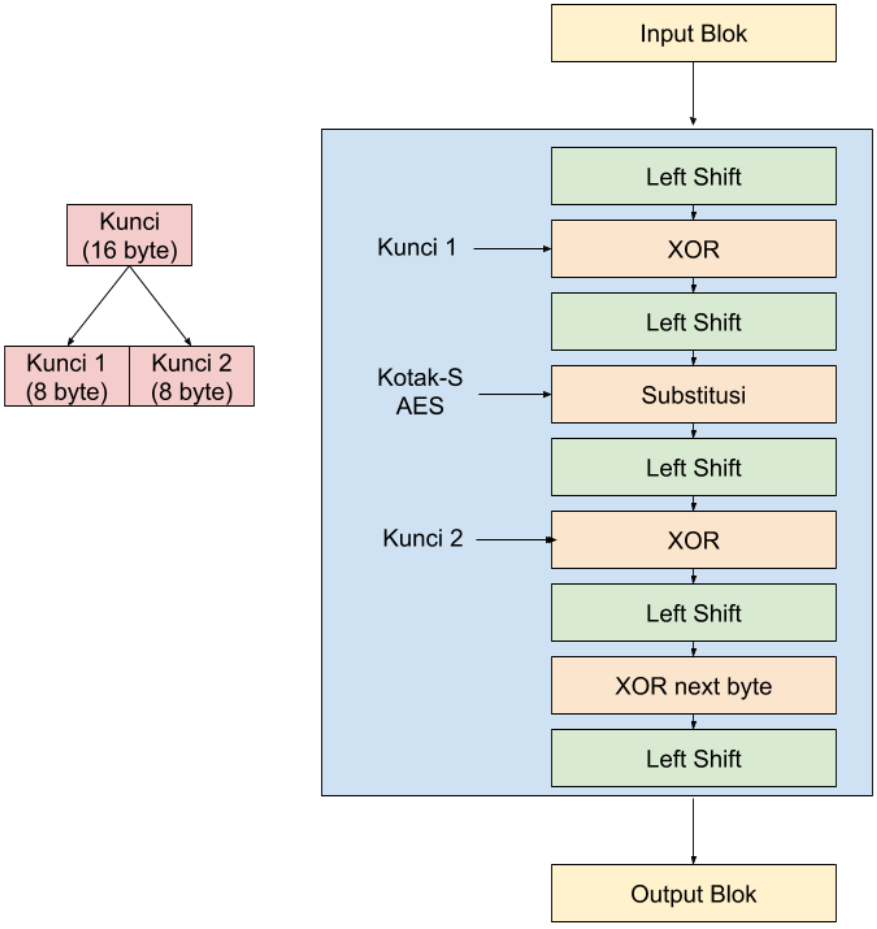
Jika digambarkan maka operasi penuh dari algoritma ini adalah sebagai berikut (asumsi mode CBC)



Gambar 3.2. Tahapan pemrosesan CBC



Gambar 3.3. Tahapan jaringan Feistel (fungsi E pada gambar 3.2.)



Gambar 3.4. Tahapan perjalanan pada tiap *round* (fungsi f pada gambar 3.3.)

4. Eksperimen dan Hasil Analisis

Algoritma diimplementasikan menggunakan bahasa C++. Untuk menguji keamanan dan performa dari *block cipher* baru yang dibuat dilakukan beberapa eksperimen sebagai berikut :

4.1. Pengujian Kecepatan

Untuk menguji performa dari implementasi *block cipher* yang dibuat dilakukan percobaan dengan 3 ukuran *file* yaitu 1 KB, 1 MB, dan 10 MB

Tabel 4.1. Ukuran *file* yang diuji

Ukuran	Ukuran (byte)
1KB	958
1MB	1103239
10 MB	10259179

Pertama, akan diuji performa dari masing-masing mode yang didukung (ECB, CBC, dan Counter) dengan ukuran *file* 1MB. Percobaan dilakukan sebanyak 5 kali untuk tiap mode.

Tabel 4.2. Waktu enkripsi dan dekripsi untuk *file* 1MB

Mode	Waktu rata-rata (ms)
Enkripsi ECB	354.2
Dekripsi ECB	341.2
Enkripsi CBC	735.6
Dekripsi CBC	775.2
Enkripsi Counter	836.2
Dekripsi Counter	826.2

Dari hasil pengujian di atas didapati bahwa mode tercepat adalah mode ECB. Hal tersebut wajar karena pada mode tersebut tidak dilakukan pemrosesan tambahan selain membagi ke dalam blok lalu langsung diberikan kepada fungsi enkripsi. Untuk mode lain diperlukan manipulasi tambahan seperti XOR sehingga wajar jika lebih lambat. Setelah itu dapat dilakukan pengujian performa untuk ukuran *file* yang berbeda. Untuk percobaan ini dilakukan dengan mode CBC dan dilakukan sebanyak 5 kali.

Tabel 4.3. Waktu enkripsi dan dekripsi untuk *file* 1MB

Ukuran	Waktu rata-rata (ms)
Enkripsi 1KB	1.726
Dekripsi 1KB	1.707
Enkripsi 1MB	735.6
Dekripsi 1MB	775.2
Enkripsi 10MB	4060
Dekripsi 10MB	4787

Dari hasil di atas didapati bahwa waktu yang dibutuhkan untuk enkripsi dan dekripsi cukup cepat mengingat implementasi dibuat tanpa konsiderasi khusus optimasi.

4.2. Pengujian Keamanan

4.2.1. Prinsip Diffusion

Untuk menguji prinsip *diffusion* dilakukan percobaan dengan pertama mengenkripsi secara normal kemudian dilakukan pengubahan 1 bit pada plainteks dan kemudian dienkripsi kembali. Setelah itu dibandingkan *output* dari kedua enkripsi tersebut. Jika prinsip *diffusion* terpenuhi maka hasil cipherteks seharusnya berbeda jauh.

Tabel 4.4. Persentase kesamaan *byte* cipherteks setelah diubah 1 bit pada plainteks

Mode	Persentase Perbedaan
ECB	0.001%
CBC	99.61%
Counter	0.0001%

Jika dilihat sekilas mungkin hasil tersebut terlihat kurang baik. Namun yang perlu diingat adalah bahwa percobaan tersebut dilakukan pada *file* berukuran 1MB. Pada mode ECB dan Counter, tiap blok tidak akan mempengaruhi blok lainnya, berbeda dengan mode CBC yang saling mempengaruhi. Dengan pengertian tersebut maka hasil di atas sangatlah wajar. Namun tetap butuh dibuktikan prinsip *diffusion* untuk algoritma *cipher* ini sendiri. Untuk itu perlu dilakukan percobaan yang sama untuk *file* berukuran 1 blok.

Tabel 4.5. Persentase kesamaan *byte* cipherteks setelah diubah 1 bit pada plainteks berukuran 1 blok

Mode	Persentase Perbedaan
ECB	100 %
CBC	100 %
Counter	6.25 %

Dengan melihat hasil di atas, dapat dibuktikan bahwa hasil algoritma sudah baik secara *diffusion*. Untuk mode *counter* hasilnya tidak terlalu baik karena metode dari mode itu sendiri yang hanya menggunakan plainteks untuk di XOR di akhir dengan nilai *counter* yang sama. Selain dari itu, jika terjadi perubahan kunci hasilnya juga akan berubah secara drastis. Untuk menguji hal tersebut dijalankan eksperimen yang sama tetapi dengan perubahan pada 1 bit kuncinya.

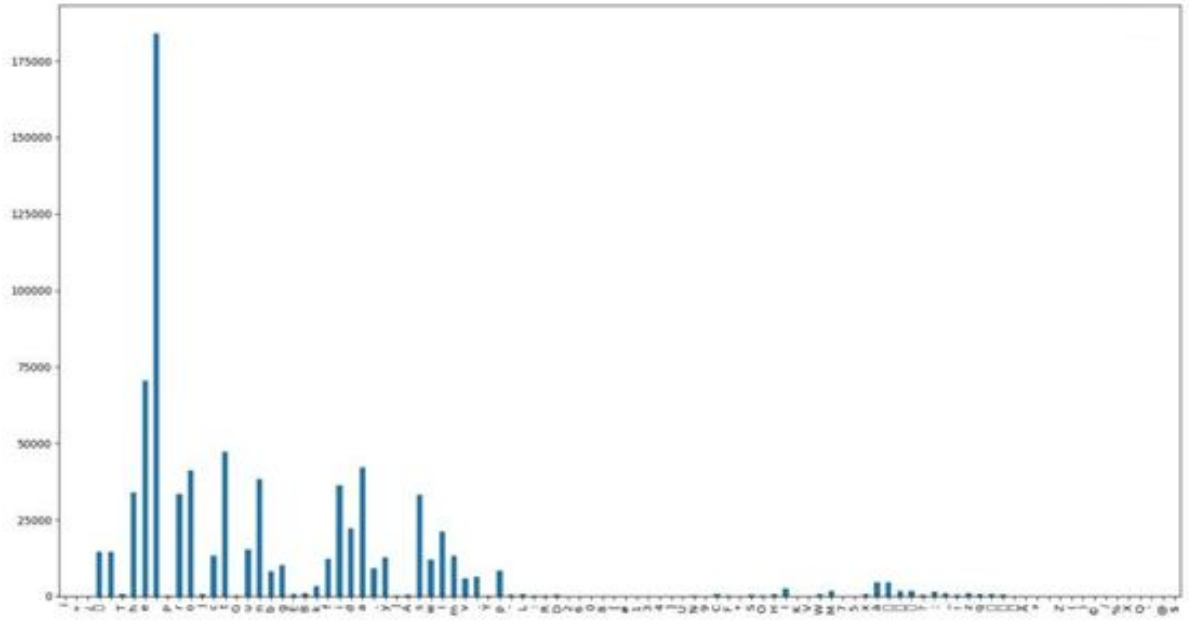
Tabel 4.6. Persentase kesamaan *byte* cipherteks setelah diubah 1 bit pada kunci

Mode	Persentase Perbedaan
ECB	99.644 %
CBC	99.606 %
Counter	99.512 %

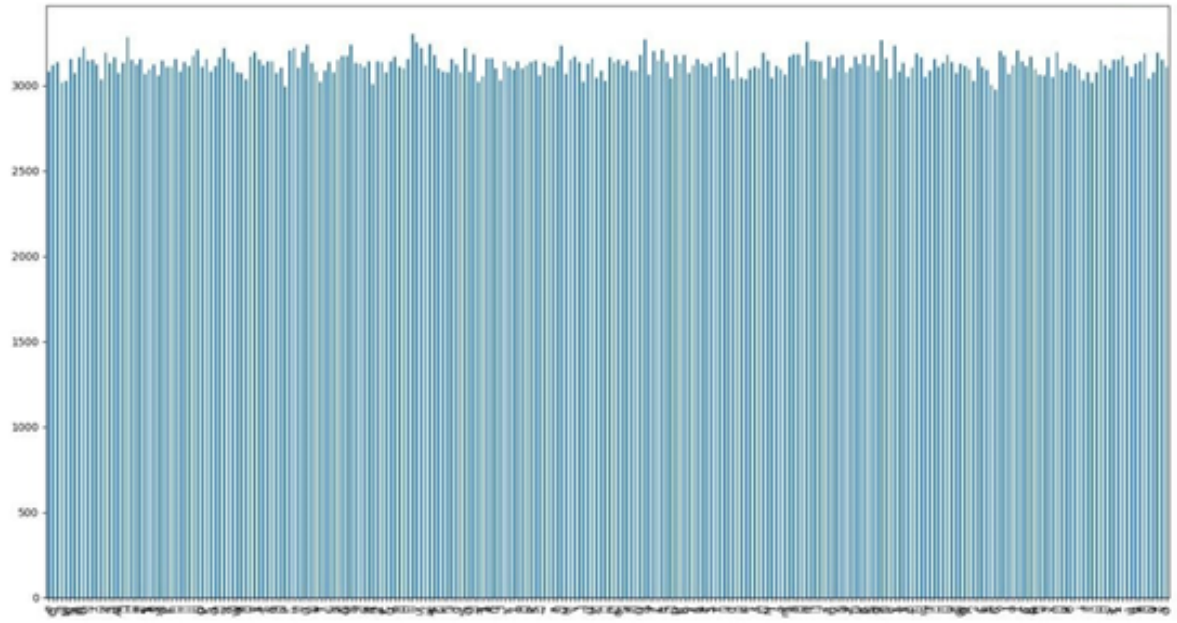
Jika melihat tabel di atas sudah terbukti bahwa perubahan pada kunci akan langsung mengubah seluruh hasil cipherteks. Dengan percobaan ini terbuktilah sudah prinsip *diffusion*.

4.2.2. Prinsip Confusion

Untuk menguji prinsip *confusion* ini akan dilakukan uji frekuensi kemunculan *byte* atau karakter pada plainteks dan cipherteks. Untuk percobaan ini akan digunakan teks *Pride and Prejudice* yang tersedia dalam domain publik.

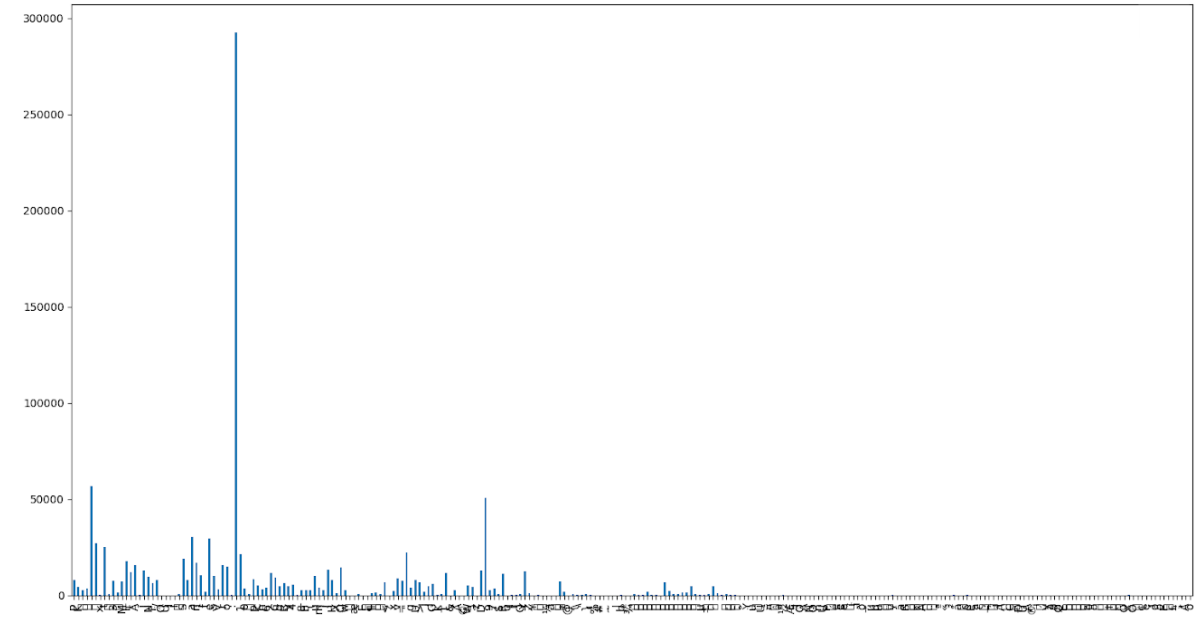


Gambar 4.1 Grafik frekuensi kemunculan karakter pada plainteks

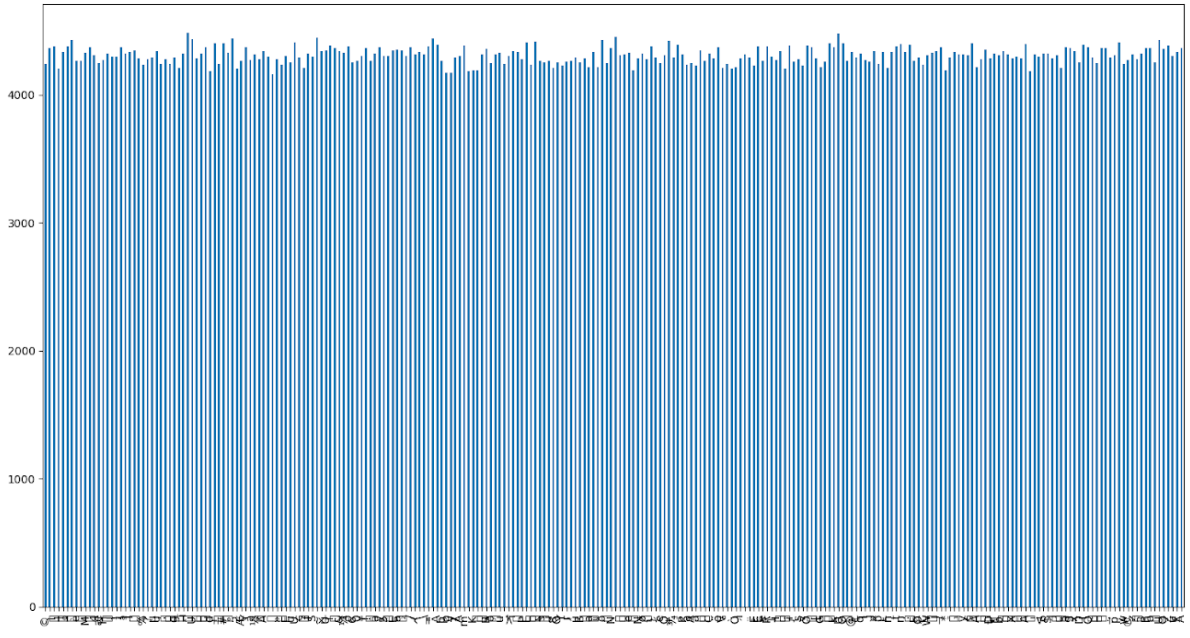


Gambar 4.2 Grafik frekuensi kemunculan karakter pada cipherteks

Selain itu eksperimen juga dapat dilakukan dengan *file* yang bukan berupa teks. Untuk contoh ini akan digunakan sebuah *executable* Java berukuran sekitar 1MB.



Gambar 4.3 Grafik frekuensi kemunculan karakter pada plainteks



Gambar 4.4 Grafik frekuensi kemunculan karakter pada cipherteks

Dari semua grafik di atas dapat terlihat dengan jelas bahwa prinsip *confusion* sudah terpenuhi karena saat dienkripsi, distribusi untuk tiap karakter atau *byte* menjadi sama sehingga akan hampir mustahil untuk melakukan analisis frekuensi.

4.2.3. Analisis Kemungkinan Bruteforce

Oleh karena algoritma yang digunakan untuk meregenerasi *round key* yaitu MD5 maka otomatis panjang kunci efektif dari sistem adalah sepanjang hasil *digest* MD5 yaitu sepanjang 128 bit. Untuk melakukan *brute force* untuk kunci sepanjang 128 bit tidaklah mudah. Oleh karena itu berarti akan ada 2^{128} kemungkinan kunci. Dengan asumsi ada komputer yang memiliki kemampuan 1 milyar pengujian per detik maka dibutuhkan $3.4 * 10^{29}$ detik atau sekitar 10^{22} tahun. Data yang disimpan di dalamnya tidak akan sebanding dengan usaha untuk memecahkan enkripsi selama itu sehingga *block cipher* ini relatif aman.

5. Kesimpulan & Saran Pengembangan

5.1. Kesimpulan

Penulis berhasil membangun sebuah algoritma *block cipher* yang baik dan diberi nama *Wonderful Journey Block Cipher* (WJBC). Perancangan algoritma WJBC menggunakan prinsip-prinsip *confusion* dan *diffusion* pada fungsi transformasi enkripsi yang dilakukan. Dengan beberapa pengujian dapat dikatakan bahwa algoritma WJBC memberikan hasil enkripsi yang kompleks dan aman.

5.2. Saran Pengembangan

Untuk pengembangan berikutnya disarankan beberapa hal berikut :

- Melakukan analisis keamanan yang lebih mendetail dengan mencoba mencari kelemahan secara matematis
- Meningkatkan performa implementasinya
- Menggunakan ukuran kunci yang lebih panjang

6. Referensi

[1] E. Errata, "Cryptographic Message Syntax (CMS)", Vigil Security, September 2009

[2] M. Alfred, O. Paul van, dan V. Scott, "*Handbook of Applied Cryptography*", 1996, isbn : 0-8493-8523-7

[3] National Institute of Standards and Technology, Data Encryption Standard (DES). (U.S.: U.S. Department of Commerce)

[4] National Institute of Standards and Technology, Advanced Encryption Standard (AES). (U.S.: U.S. Department of Commerce)

[5] R. Munir, "Kriptografi", Edisi Kedua. Bandung: Informatika Bandung, 2019, isbn: 978-623-7131-05-2.

7. *Acknowledgments*

Puji syukur penulis panjatkan ke hadirat Tuhan yang Maha Esa, karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada seluruh pihak yang telah membantu dalam proses desain, perancangan hingga implementasi dari Algoritma *Wonderful Journey Block Cipher*, khususnya teruntuk kepada Bapak Dr. Ir. Rinaldi Munir, MT. sebagai dosen pengajar mata kuliah IF4020-Kriptografi yang telah memberikan wawasan dan pengetahuan di bidang kriptografi serta orang tua dan teman-teman yang telah memberikan dukungan baik secara langsung maupun tidak langsung. Penulis berharap hasil makalah dapat memberikan manfaat berupa wawasan maupun solusi mengenai algoritma *block cipher* kepada pembaca.