V Cipher

Putra Hardi Ramadhan[1], **Kevin Sendjaja**[2], **Michael Ray**[3].

[1,2,3] Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung (ITB), Jalan Ganesha 10, Bandung 40132

E-mail: 13516080@std.stei.itb.ac.id, 13517023@std.stei.itb.ac.id, 13517092@std.stei.itb.ac.id

Abstract. Kriptografi adalah salah satu ilmu yang diaplikasikan untuk mengamankan informasi. *Block cipher* adalah salah satu algoritma Kriptografi Modern yang paling banyak digunakan saat ini. Algoritma ini memiliki banyak variasi, dikarenakan konsepnya yang memungkinkan perancang algoritma untuk merancang algoritma serumit mungkin untuk dipecahkan. V *Cipher* yang penulis buat merupakan salah satu variasi *block cipher*. V *Cipher* memiliki konsep yang sama seperti Algoritma DES, tetapi dilengkapi dengan penggunaan *randomizer* untuk mengurangi pola yang berulang selama enkripsi, sekaligus menambahkan proses enkripsi baru berupa pengaplikasian *masking bits*. **Keywords**: *Block Cipher*, *Feistel*, *Confusion*, *Diffusion*, *Randomizer*, *Masking Bits*, *Seed*.

1. Pendahuluan

1.1. Latar Belakang

Algoritma *block cipher* adalah salah satu algoritma kriptografi modern. Algoritma kriptografi *block cipher* bekerja pada suatu data yang berbentuk blok/kelompok data dengan panjang data tertentu (dalam beberapa *byte*), jadi dalam sekali proses enkripsi atau dekripsi data yang masuk mempunyai ukuran yang sama.

Pada algoritma penyandian blok *(block cipher),* plainteks yang masuk akan diproses dengan panjang blok yang tetap yaitu *n*, namun terkadang jika ukuran data ini terlalu panjang maka dilakukan pemecahan dalam bentuk blok yang lebih kecil. Jika dalam pemecahan dihasilkan blok data yang kurang dari jumlah data dalam blok maka akan dilakukan proses *padding* (penambahan beberapa *bit*).

1.2. Beberapa block cipher sejenis

1. *Data Encryption Standard* (DES)

   DES merupakan algoritma *block cipher* dengan ukuran blok 64 *bit* dan ukuran kunci efektif 56 *bit*, sementara 8 *bit* sisanya merupakan *bit* paritas. DES pernah dijadikan sebuah standar dalam metode enkripsi pada tahun 1972, namun kini sudah dianggap tidak aman lagi karena ukuran kuncinya yang sangat pendek.

   Proses enkripsi dengan DES diawali dengan proses permutasi blok plainteks dengan matriks permutasi awal (*initial permutation* atau IP). Hasil dari permutasi awal tersebut kemudian di *enchiper* sebanyak 16 kali atau 16 putaran. Setiap putarannya menggunakan kunci internal yang berbeda. Hasil dari proses *enchiper* kembali dipermutasi dengan matriks permutasi balikan (*inverse initial permutation* atau IP-1) menjadi blok cipherteks.

2. *Triple DES*

Triple DES merupakan variasi dari *Data Encryption Standard* (DES). Metode ini menggunakan kunci 64 *bit* terdiri atas 56 *bit* kunci yang efektif dan 8 *bit* paritas. *Triple* DES digunakan untuk memeriksa kesalahan selama proses transmisi. Ukuran setiap *block triple* DES adalah 8 *byte* dengan menggunakan tiga kunci yang berbeda.

3. AES (Advanced Encryption Standard)

AES merupakan metode *symmetric key encryption* untuk data elektronik. AES pertama kali digunakan oleh pemerintah Amerika Serikat dan sekarang digunakan oleh seluruh dunia menggantikan DES**.** Standar *key* AES terdiri dari 128, 192 dan 256 *bit*. AES didesain untuk menggantikan DES sebagai standar baru dalam enkripsi. Berbeda dengan DES, AES mengenkripsi pesan dalam blok-blok berukuran 128 *bit*.

1.3. Pendekatan Desain Block Cipher Baru

Pendekatan yang dilakukan dalam mendesain *block cipher* yang baru ini adalah dengan menggunakan *randomizer*. Pendekatan ini didasari dengan sifat *randomizer* yang dapat memberikan urutan bilangan berbeda tergantung dari *seed* yang digunakan, sehingga pengaplikasiannya pada algoritma *block cipher* diharapkan dapat memberikan keamanan tambahan sehingga lebih sulit untuk memecahkan *ciphertext* dengan metode seperti *brute force*. Algoritma yang diusulkan ini menggunakan dasar algoritma DES, dan menambahkan penggunaan *randomizer* di beberapa bagian pada algoritma serta penambahan beberapa improvisasi lainnya. Algoritma *block cipher* yang baru ini diberi nama V *Cipher*, yang mengambil inspirasi dari kata *volatile*, yang dapat diartikan memiliki kecenderungan untuk berubah sifat. Dalam kasus ini, sifat tersebut merujuk pada nilai *seed* yang dapat berubah dengan mudah jika kunci eksternal pengguna berubah sedikit saja, dan *seed* ini akan mempengaruhi bagian-bagian dari proses enkripsi lainnya.

2. Studi Pustaka

2.1. Teori Singkat Block Cipher

Block cipher merupakan algoritma pemetaan blok – blok *plaintext* ke blok – blok *ciphertext*. Pada suatu teks sandi sepanjang n-bit, terlebih dahulu kita bagi dalam beberapa blok – blok dengan ukuran panjang yang sama. Dengan kunci yang sama dan dengan algoritma tertentu, blok – blok ini dienkripsi, dan hasil outputnya pun berupa blok – blok sandi yang terenkripsi dan berukuran sama.

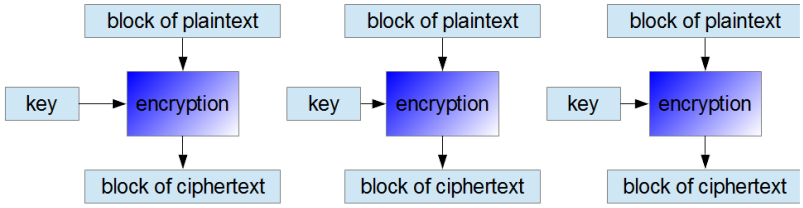
Block cipher memiliki beberapa keuntungan, yaitu mudahnya implementasi algoritma *block cipher* ke dalam *software – software*. *Error Propagation* yang terjadi pun tidak merambat ke *ciphertext* lainnya karena enkripsi masing – masing bloknya independen. Namun, *clock cipher* sangat mudah dianalisis karena blok – blok yang dienkripsi saling independen dan kuncinya sama, maka hal ini memudahkan kriptanalis untuk mengetahui kunci yang digunakan.

Ada 5 method dalam pengenkripsian, yaitu metode *Electronic Code Book*, *Cipher Block Chainin*g, *Cipher Feedback*, *Output Feedback*, dan *Counter*.

1. Electronic Code Book (ECB)

ECB merupakan metode standar dari *Block Cipher*, yaitu setiap blok plainteks dienkripsi dengan kunci yang sama secara satu persatu. Kelemahan utama dari metode ini adalah mudahnya pendeteksian, terutama jika ada blok – blok data yang sama dan dienkripsi dengan kunci yang sama maka akan menghasilkan cipherteks yang sama pula. Hal inilah yang menyebabkan mengapa disebut *Electronic Code Book*, karena seolah kita dapat mengetahui dan membuat

sebuah kamus atau ensiklopedi dari plainteks dan cipherteks dengan kunci yang sama, dan sangat mudah di kriptanalisis.

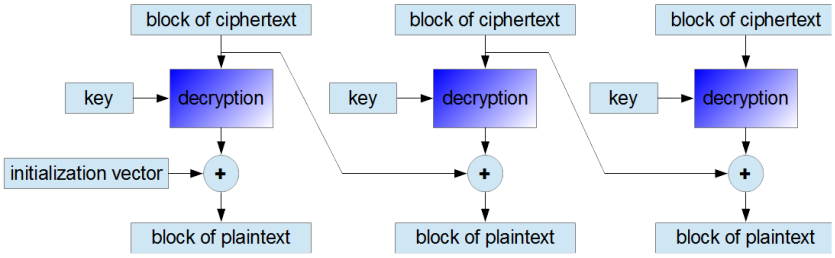


Gambar 1 *Electronic Code Book* [4]

2. Cipher Block Chaining (CBC)

Metode CBC didesain untuk mengatasi kelemahan pada metode ECB. Pada dasarnya, enkripsi yang dilakukan sama dengan ECB, namun terdapat perbedaan disini. Plaintext yang akan dienkripsi terlebih dahulu dilakukan XOR dengan *ciphertext* fase sebelumnya. Untuk fase pertama digunakan *Initialization Vector* (IV) sebagai nilai awal, yang kemudian di XOR dengan blok *plaintext* yang selanjutnya dilakukan enkripsi dengan kunci yang telah disepakati. Selanjutnya, blok *ciphertext* yang dihasilkan, selain dikeluarkan melalui output, blok *ciphertext* tersebut dilakukan XOR lagi dengan blok *plaintext* yang selanjutnya.

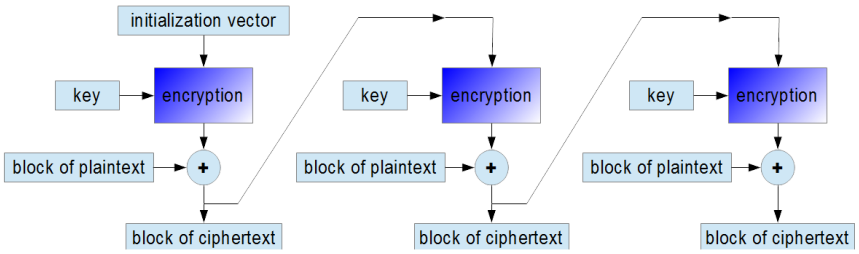
Pada metode dekripsi, yang dilakukan terlebih dahulu adalah mendekripsi blok *ciphertext* yang kemudian dilakukan XOR dengan ciphertext sebelumnya, dimana untuk fase awal digunakan *Initialization Vector* yang sama dengan pada saat enkripsi. Selanjutnya, blok *ciphertext* ini di-XOR dengan blok *ciphertext* selanjutnya setelah dilakukan proses dekripsi.



Gambar 2 *Cipher Block Chaining* [4]

3. Cipher Feedback (CFB)

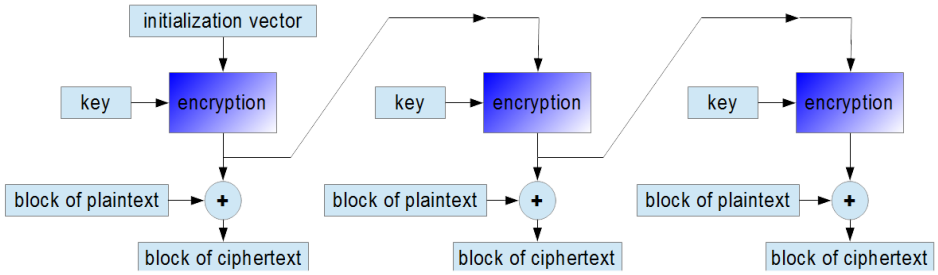
Metode *Cipher Feedback* menggunakan sistem *Shift Register*, dimana yang diproses terlebih dahulu adalah *Initialization Vector* dalam algoritma enkripsi dengan kunci. Setelah diproses, *bit* yang dihasilkan akan melalui proses seleksi *bit*, biasanya *bit – bit* yang paling kiri, untuk selanjutnya dienkripsi dengan *plaintext* untuk menghasilkan *ciphertext*. *Bit* hasil seleksi yang digunakan tergantung besarnya *bit* blok *plaintext* yang diinput. Selanjutnya, setelah mendapatkan blok *ciphertext*, selain di output, blok *ciphertext* tersebut dimasukkan ke IV yang sebelumnya, dan IV digeser sebanyak *bit* blok *ciphertext* sebelumnya, yang selanjutnya IV yang telah digeser bersama blok *ciphertext* yang digabung bersama IV tersebut diproses kembali oleh algoritma enkripsi tersebut.



Gambar 3 *Cipher Feedback* [4]

4. Output Feedback (OFB)

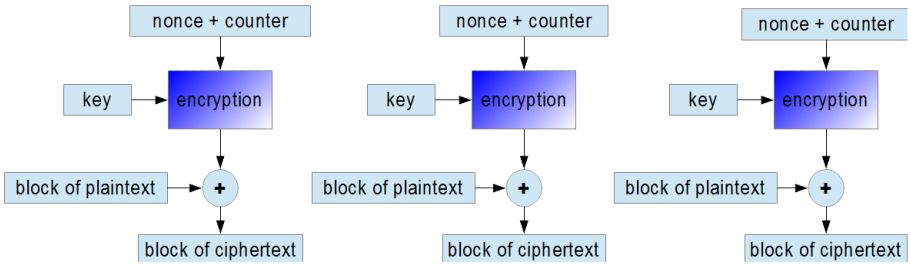
Metode OFB memiliki perbedaan dengan CFB, dimana input yang digunakan dalam proses enkripsi. Kalau dalam CFB, input yang digunakan adalah *ciphertext* yang selanjutnya di-*shift* bersama IV, dalam OFB yang digunakan adalah output *bit* hasil dari proses seleksi yang kemudian di shift bersama IV yang sebelumnya. Hasil seleksi tetap digunakan dalam proses enkripsi *plaintext* untuk mendapatkan *ciphertext*.



Gambar 4 *Output Feedback* [4]

5. Counter Mode

Mode *Counter* memanfaatkan sebuah nilai *counter* yang dienkripsi menggantikan blok-blok *plaintext*. Nilai *counter* harus berbeda dari setiap blok yang dienkripsi. Pada mulanya, untuk enkripsi blok pertama, counter diinisialisasi dengan sebuah nilai. Selanjutnya, untuk enkripsi blok-blok berikutnya counter dinaikkan nilainya satu. Blok *plaintext* akan kemudian di-XORkan dengan hasil enkripsi untuk menghasilkan blok-blok *ciphertext*.



Gambar 5 *Counter Mode* [4]

2.2 *Shannon's Confusion dan Diffusion*

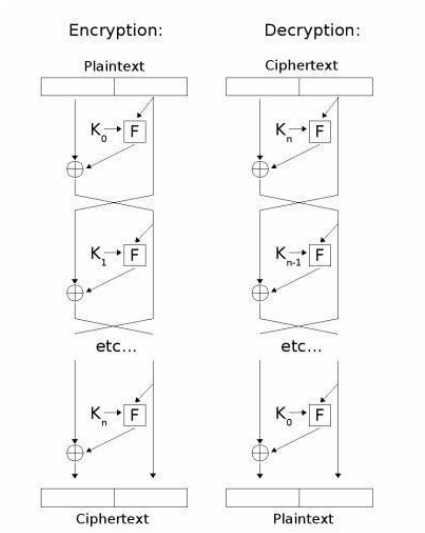
Dalam kriptografi, prinsip *diffusion* dan *confusion* adalah 2 sifat dari *secure cipher* yang diperkenalkan oleh Claude Shannon pada tahun 1945. Prinsip yang dikemukakan pada publikasinya—yang berjudul

Communication Theory of Secrecy Systems—dibuat dengan tujuan untuk mempersulit beberapa metode kriptanalisis seperti metode statistik. Pengertian *confusion* menurut Shannon adalah hubungan antara *ciphertext* dan kunci yang dibuat serumit mungkin sehingga kriptanalis frustasi dalam menemukan hubungan antara keduanya. Sedangkan untuk *diffusion* mengacu pada persebaran pengaruh kunci terhadap bit-bit *plaintext* yang seluas mungkin. Sebagai contoh pada *vigenere cipher*, prinsip *confusion* diterapkan sehingga hubungan antara kunci dan *ciphertext* tidak serta merta terlihat—dengan membuat *ciphertext* yang berasal dari operasi penjumlahan karakter pesan dan karakter kunci.

Diffusion bertujuan agar perubahan bit pada *ciphertext* menampilkan hasil pesan di luar prediksi kriptanalis. Untuk mendapatkan *cipher* dengan keamanan yang tinggi, prinsip *diffusion* dan *confusion* diterapkan secara berulang dalam sebuah blok tunggal dengan kombinasi yang berbeda-beda. Metode paling sederhana untuk memenuhi prinsip *confusion* dan *diffusion* adalah menerapkan jaringan substitusi dan permutasi.

2.3. *Feistel Network*

Feistel network atau jaringan Feistel adalah struktur simetris yang digunakan dalam mengkonstruksi *block cipher*. Model dasar jaringan Feistel membagi blok masukan menjadi beberapa bagian, menggunakan fungsi Feistel kepada upablok-upablok tersebut, kemudian melakukan operasi XOR antara upablok-upablok tersebut. Kunci enkripsi (*round key*) digunakan pada tiap fungsi Feistel yang dikenakan pada upablok. Rangkaian operasi ini dapat dilakukan berulang kali selama beberapa putaran atau *round*. *Round key* pada setiap putaran dibangkitkan secara pseudo-random. Karena sifatnya yang simetris, jaringan Feistel memiliki keunggulan yaitu algoritma yang digunakan untuk proses enkripsi dapat digunakan lagi untuk proses dekripsi. Kelebihan lain dari jaringan Feistel yang membuatnya menjadi menarik untuk dibahas adalah karena fungsi Feistel yang dikenakan pada upablok dapat dirancang sesulit apapun. Perancang fungsi tidak perlu khawatir akan kehilangan properti reversible dari jaringan Feistel. Selain itu perancang fungsi juga tidak perlu menentukan fungsi balikan (f-invers) dari fungsi Feistel tersebut.



Gambar 6 Proses enkripsi dan dekripsi melalui jaringan Feistel

Model dasar jaringan Feistel adalah sebagai berikut :

1. Bagi blok plainteks berukuran n bit menjadi dua bagian, kiri (L) dan kanan (R), yang masing-masing panjangnya n/2.

2. Untuk setiap putaran i = 0,1,2,...,n, lakukan :

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus \mathrm{F}(R_i, K_i)$$

Dalam hal ini, n = jumlah putaran K_i = upa-kunci pada putaran ke-i F = fungsi Feistel

3. Jika blok plainteks adalah (L_0,R_0), maka *cipher block* adalah (R_{n+1},L_{n+1}). Untuk menyederhanakan proses di atas, pada satu putaran ke-i jaringan Feistel,

$$X_{i+1} = (F_{k_i}(msb_{n/2}(X_i)) \oplus lsb_{n/2}(X_i))||msb_{n/2}(X_i)$$

Di mana, X_i adalah blok masukan X_{i+1} adalah blok keluaran F_{ki} adalah fungsi Feistel dengan upa kunci ke-i Msbn/2 adalah n/2 *most significant bit* dari blok masukkan Lsbn/2 adalah n/2 *least significant bi*t dari blok masukkan Salah satu keunggulan dari penggunaan jaringan Feistel dalam algoritma enkripsi adalah fungsi Feistel F tidak perlu memiliki atribut *invertible* (memiliki F-invers) dan dapat dibuat serumit apapun. Selain itu perbedaan dari proses enkripsi dan dekripsinya hanya terletak pada penggunaan upa-kunci yang dibalik.

3. Rancangan Block Cipher

3.1 Struktur Algoritma

Secara struktur, algoritma V *cipher* tidak berbeda jauh dengan algoritma DES, dimana proses enkripsi terdiri dari :

1. *Initial Permutation*
2. *Iterated Cipher* dengan *Feistel Network*
3. *Inverse Permutation*

Tahapan *Initial Permutation* dan *Inverse Permutation* tidak berbeda dari tahapan serupa pada algoritma DES, sementara tahap *Iterated Cipher* mengalami beberapa perubahan yang menggunakan *randomizer*, dan akan dijelaskan pada subbab-subbab selanjutnya.

Pemrosesan *plaintext* dilakukan dalam blok-blok berukuran 64 *bit*. Kunci eksternal yang digunakan berukuran 64 *bit* dengan 8 *bit* paritas layaknya algoritma DES. Akan tetapi, panjang kunci eksternal yang dimasukkan oleh pengguna tidak dibatasi sepanjang 8 karakter saja. Pengguna dapat memasukkan kunci eksternal dengan panjang kurang atau lebih dari 8 karakter, selama tidak 0, layaknya sebuah *password*. Dari kunci eksternal ini, akan dibangkitkan sebuah *seed* yang akan digunakan untuk menginisiasi *randomizer* yang nantinya akan digunakan untuk membangkitkan kunci-kunci internal untuk proses *Iterated Cipher* nantinya. Proses pembangkitan *seed* adalah sebagai berikut:

1. Ambil nilai sebuah *initial seed* berupa *integer value* dari setiap karakter pada kunci eksternal
2. Gunakan *initial seed* tersebut untuk menginisiasi *randomizer*
3. Inisialisasi nilai *seed* dengan *value* 0
4. Gunakan *randomizer* untuk mendapatkan nilai *n* berupa sebuah bilangan antara 0 dan panjang kunci eksternal - 1
5. Tambahkan *integer value* dari karakter pada indeks *n* dari kunci eksternal ke dalam nilai *seed*
6. Ulangi langkah 4 dan 5 hingga (panjang kunci eksternal / 2) kali
7. Simpan nilai *seed*

Sebagai contoh, misalkan kunci eksternal dari pengguna adalah “secret key”, maka *initial seed* didapatkan dengan cara

$$\textit{Initial seed} = \text{int(s)} + \text{int(e)} + \text{int(c)} + \ldots + \text{int(y)}$$

Setelah *randomizer* diinisiasi dengan *initial seed*, akan dibangkitkan 5 buah bilangan random n_1, n_2, …, n_5 antara indeks 0 hingga 9. Lalu *seed* akan didapatkan dengan cara

$$\textit{Seed} = \text{int(kunci\_eksternal}[n_1]) + \text{int(kunci\_eksternal}[n_2]) + \ldots + \text{int(kunci\_eksternal}[n_5])$$

3.2 Pembangkitan Kunci Internal

Proses pembangkitan kunci internal tidak jauh berbeda dengan proses pembangkitan kunci eksternal pada algoritma DES, namun jumlah kunci yang dibangkitkan hanya sejumlah 8 buah, karena proses *Iterated Cipher* nantinya hanya akan dilakukan sebanyak 8 kali putaran. Selain itu, proses pergeseran saat membangkitkan kunci internal kini ditentukan oleh *seed* yang didapat dari kunci eksternal. Proses pembangkitan kunci internal adalah sebagai berikut:

1. Inisialisasi sebuah list *keys* untuk kunci-kunci internal yang akan dibangkitkan
2. Inisialisasi variabel *key* dengan kunci eksternal. Jika kunci eksternal bukan merupakan kelipatan 8, maka tambahkan *padding* dengan karakter *dummy* hingga *key* kelipatan 8
3. Jika panjang *key* lebih dari 8 karakter, maka bagi menjadi blok-blok 8 karakter, lakukan XOR pada masing-masing *bit* pada blok-blok 8 karakter, dan simpan hasilnya ke dalam *key*
4. Lakukan permutasi pada *key* menggunakan matriks permutasi 1 untuk mendapatkan 56 *bit* yang akan dipakai, lalu bagi menjadi 2 blok 28 *bit*
5. Inisiasi *randomizer* dengan *seed* yang didapat dari kunci eksternal
6. Gunakan *randomizer* untuk mendapatkan nilai *n* berupa sebuah bilangan antara 1 dan 27
7. Geser *bit-bit* pada kedua blok 28 *bit* ke kanan sejauh *n* posisi
8. Lakukan permutasi pada kedua blok 28 *bit* dengan matriks transformasi 2 untuk mendapatkan kunci internal sepanjang 48 *bit* dan simpan di list *keys*
9. Ulangi langkah 6 hingga 8 sampai terbentuk 8 buah kunci internal

Proses permutasi menggunakan 2 buah matriks permutasi yang serupa dengan matriks permutasi pada algoritma DES.

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Gambar 7 Kedua matriks permutasi yang digunakan dalam pembangkitan kunci internal

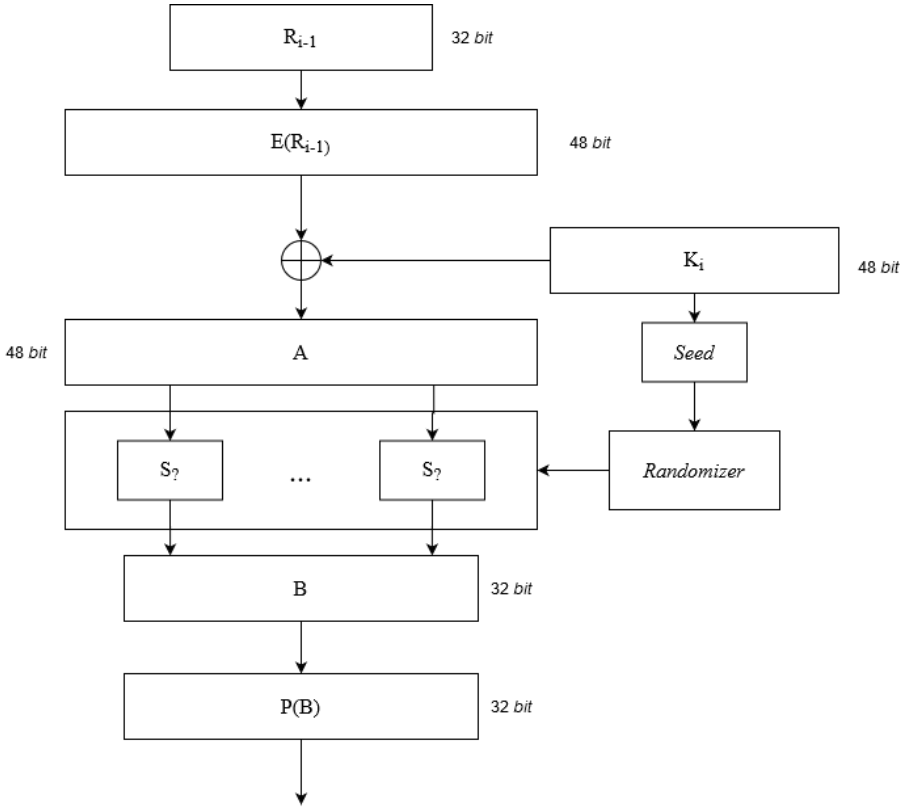
3.3 Iterated Cipher

Proses ini juga tidak jauh berbeda dengan algoritma DES, tetapi putaran hanya dilakukan sebanyak 8 kali. Jumlah *S-Box* yang digunakan untuk proses substitusi lebih sedikit dari algoritma DES biasa, yaitu hanya sebanyak 4 buah. Namun, penentuan penggunaan *S-Box* akan kembali diatur oleh *randomizer*, dimana *seed* yang akan digunakan dibangkitkan dari masing-masing kunci internal. Proses baru yang

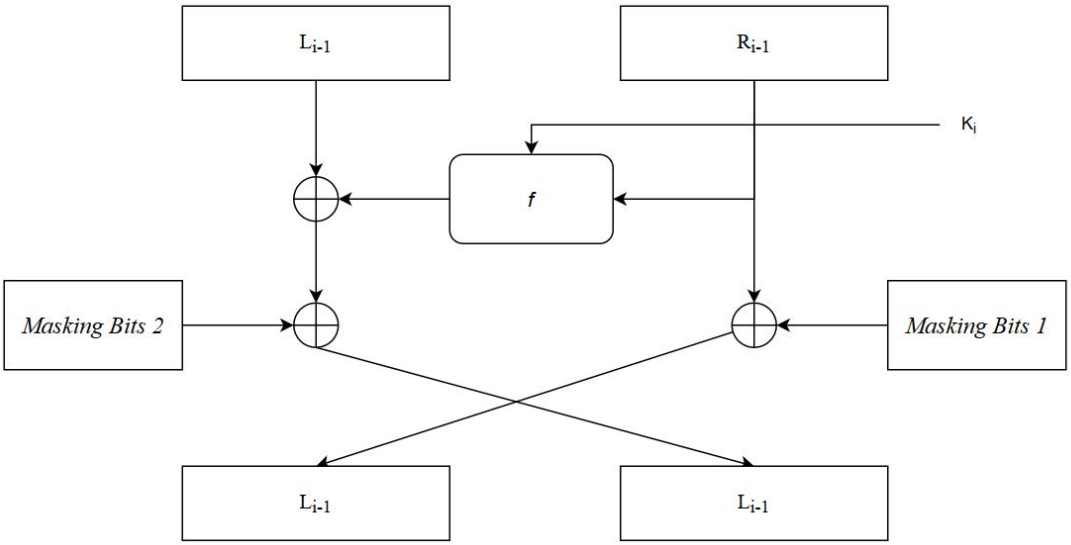
ditambahkan adalah penggunaan *masking bits* pada saat proses enkripsi berulang, dimana kedua blok kode akan di XOR kan dengan *masking bits* yang dibangkitkan oleh *randomizer* yang juga menggunakan *seed* dari kunci internal. Terdapat 2 kali 32 *masking bits* yang akan dibangkitkan, dan 32 *bit* terakhir merupakan hasil kebalikan dari 32 *bit* pertama. Sebagai contoh, apabila *masking bits* pertama adalah 1010, maka *masking bits* kedua adalah 0101. Proses ini ditambahkan sebagai enkripsi terakhir setelah fungsi enkripsi *f* pada jaringan Feistel selesai dilaksanakan. Proses *Iterated Cipher* adalah sebagai berikut:

1. Lakukan permutasi awal untuk mengacak urutan *plaintext*
2. Bagi blok 64 *bit plaintext* menjadi blok L dan R yang masing-masing memiliki panjang 32 *bit*
3. Gunakan matriks ekspansi untuk meng-*expand* blok R menjadi 48 *bit*
4. Lakukan XOR antara blok R dengan kunci internal untuk putaran saat ini
5. Bagi blok R menjadi 8 blok masing-masing 6 bit
6. Inisialisasi *randomizer* dengan *seed* dari kunci internal. Untuk mendapatkan nilai *seed*, pertama nilai masing-masing *bit* pada kunci internal dengan indeks *bit* tersebut dan dijumlahkan untuk mendapatkan *initial seed*. Lalu setelah *randomizer* diinisiasi dengan *initial seed*, bangkitkan angka *random n* dengan nilai antara 0 hingga 47. *Seed* yang memiliki nilai awal 0 akan dijumlahkan dengan nilai *bit* pada indeks *n* dikalikan dengan *n*. Ulangi proses sebanyak 24 kali
7. Gunakan *randomizer* untuk menghasilkan 8 buah angka antara 1 hingga 4. Angka yang dihasilkan akan menentukan indeks *S-Box* yang akan digunakan untuk masing-masing blok 6 *bit*
8. Gunakan *S-Box* untuk mensubstitusi nilai 6 *bit* menjadi 4 *bit*, dan gabungkan semuanya menjadi blok R yang baru sepanjang 32 *bit*
9. Lakukan permutasi untuk mengacak urutan blok R
10. Lakukan XOR antara blok R baru dengan blok L lama untuk mendapatkan blok L baru
11. Dengan *seed* dari kunci internal, bangkitkan 2 blok *masking bits* yang masing-masing berukuran 32 *bit*. Pertama akan dibangkitkan 32 bilangan *random*. Isi *masking bits* pertama dengan 0 apabila bilangan *random* genap, dan 1 apabila bilangan *random* ganjil. Setelah didapat blok *masking bits* pertama, buat blok kedua dengan membalikkan nilai masing-masing *bit* pada *masking bits* pertama
12. Lakukan XOR antara blok R lama dengan *masking bits* pertama, dan antara blok L baru dengan *masking bits* kedua
13. Blok L baru akan menjadi blok R di putaran berikutnya, dan blok R lama akan menjadi blok L baru di putaran selanjutnya
14. Ulangi langkah 3 hingga 13 sampai telah dilalui 8 putaran
15. Gabungkan blok L dan blok R menjadi blok 64 *bit*
16. Lakukan inversi permutasi untuk mengembalikan susunan *ciphertext*

Untuk proses dekripsi, khusus untuk metode ECB dan CBC, pengaplikasian *masking bits* akan dilakukan sebelum proses XOR dengan kunci internal agar proses dekripsi dapat menghasilkan *plaintext* yang benar.



Gambar 8 Diagram fungsi *f* pada algoritma V *cipher*



Gambar 9 Proses satu putaran *enciphering* pada algoritma V *cipher*

4. Eksperimen dan Hasil

4.1 Simulasi Metode ECB, CBC, dan Counter

Kunci
BlockCipher
Plainteks
Setelah rancangan anda sudah selesai, coding-lah menjadi program enkripsi dan dekripsi dalam Bahasa pemrograman yang dipilih (bebas), minimal BahasaC. Block cipher harus dapat dioperasikan minimal dalam tiga mode ECB, CBC, dan mode counter.
Cipherteks Mode ECB (Dalam Hexadecimal)
492008c346abe497a261de8e7308737fcac0b29e2cafa594bc21bcab12dc3326150cf6a7014aa5f348503fc fb78844797e5df2a3de1b3bae03125d95779ac9e6c13cc7ae255270797e02e0f2f62e6e20e579aeca1c1fd0 dd4c28553418744a1b30f04c7aae07722d1f4cfeed96ca9d31a984befdc895167fe2a5cf7445e9897c1dbc8 00d3eac17ebac86e42c3d61beb3e6acb38d7821262c32682ab8536777a9cdb9968de3158d6775c8744645 535a46a6ff77c53ca7f2bde9043693a75ffd3be7c8790d9ad620483bb0fd450c1b595f6c3c819350f7afbed 9a1e8dbf3ae6ab36a1fea352a959870aeac34fcd9bd3224ad64de53e3168e09
Cipherteks Mode CBC (Dalam Hexadecimal)
2e33f7bd6b6d7deb5b39df44671991058375e26beaae51b5ad5296a7301fb99b1f8c8f597e6d75b35422e8 4da8ab97cb482274b288de3fd4d66a202fd77cd2f76937f1455ca32488a78987324cca7cbcd6a25b57b64e cee2fab9f2c8049fa42bc10d8ded010f14e2f69aafdf1f245422f0df60fa7e60aca780eebde65b332eea18701 5bbe333aa0e98f7bbcec1f9a026ea484790d5858325b4d71277a7923d44aecaae30fb749f32f349e90aad55 8402cb3be213ea81c29439eb0882f7ba7ace69f823a3fd2e2f2048b193f3d7e25e53b92f79748a898b6482 824d0852493edfb5b85a8f1743f71a9c1e62e089f216c1124b6629594ed790
Cipherteks Mode Counter (Dalam Hexadecimal)
bf2aa4804862b96fc2099882a2b29108de48978fa7bdd61ac50c9789e3af9305d51b9788effc9506d40198 86eeb097019005938fa9bd92009018848ea4ae9704900d988ab1b5861ad9489280adfc920cdb1a9f91b0b 5d60dd104978ce39e9701d11b97c1b3b99b1bdf0f8480aebd9849c9099886e3b89f19d9049f89e3f4940cd 20985c8effc9b00de019b80affcb408d809858080f2d62bdc07958ae3bf9f19d80d84c1abbd841cc3489280 b3bd8249d4019991a6ae971ad903978fe3b19f07d905978de3b89705d105d695aabb9749dd079284e399 b52b9c48b5a380f0d60dd106d68cacb89349d307838fb7b98447

Tabel 1 Hasil uji coba cipherteks

Dari hasil ketiga mode diatas dapat terlihat bahwa cipherteks dari ketiganya memiliki perbedaan yang signifikan. Pada algoritma V *Cipher* yang didesain, apabila jumlah *byte* data bukanlah kelipatan 8 maka akan ditambahkan *padding.*

4.2 Analisis Confusion dan Diffusion

Prinsip *confusion* dan *diffusion* merupakan prinsip dasar yang dapat mengukur seberapa rumit dan sulitnya suatu algoritma cipher untuk dipecahkan. Untuk membuktikannya dapat digunakan *key* yang berbeda satu karakter dari contoh pada Tabel 1. Dan perbedaan 1 kata pada plainteks yang digunakan. Kemudian dapat dihitung kesamaan *byte*-nya

Kunci	Jumlah kesamaan Byte	Persentase Perubahan Cipherteks
BlockCapher	2/417	99,5%
BlockCepher	0/417	100%
BlockCopher	0/417	100%
BlockCupher	1/417	99,75%
BlockDipher	2/417	99,5%
Rata-rata	1/417	99,75%

Tabel 2 Hasil ujicoba penggantian 1 karakter pada kunci

Posisi penggantian	Jumlah kesamaan Byte	Persentase Perubahan Cipherteks
Posisi Pertama	2/417	99,52%
Posisi Ke-10	8/417	98,02%
Posisi Ke-25	18/417	95,61%
Posisi Ke-35	30/417	92,81%
Posisi Ke-50	39/417	90,65%
Rata-rata	19.4/417	95,34%

Tabel 3 Hasil uji coba penggantian 1 karakter pada plainteks

Dari hasil di atas, dapat dilihat bahwa perubahan karakter pada kunci memberikan persentase perubahan yang relatif besar, sehingga jumlah kesamaan *byte* dapat ditekan hingga 0 pada kasus terbaik. Perubahan pada *plaintext* juga masih memberikan perubahan *byte* yang cukup signifikan, sehingga secara keseluruhan algoritma yang didesain sudah cukup baik.

4.3 Analisis peluang Brute Force Attack

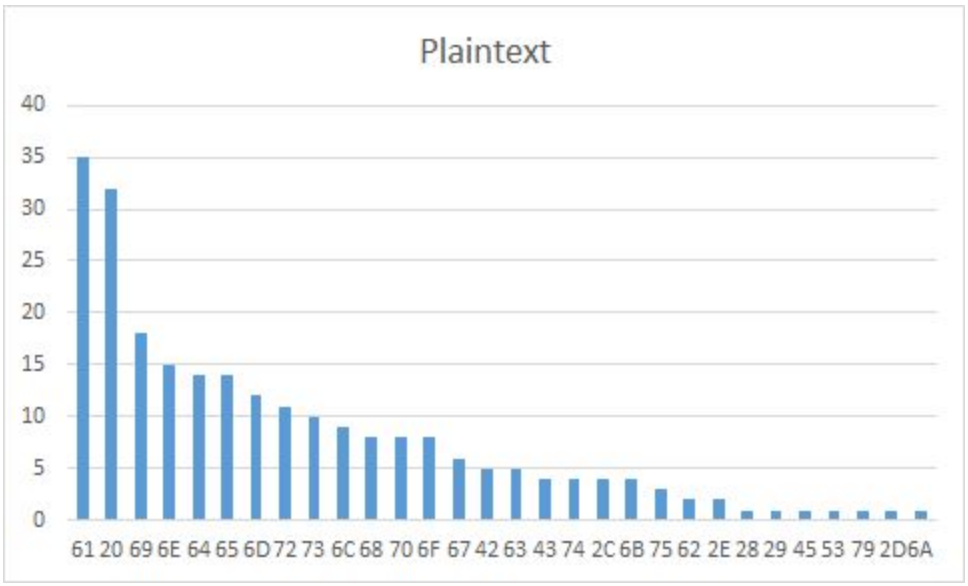
V *cipher* menggunakan 64 *bit* dari kunci eksternal untuk membangkitkan kunci-kunci internal yang digunakan pada proses enkripsi, yang kemudian hanya diambil 56 *bit* saja. Maka dapat disimpulkan bahwa ada 2^{56} kemungkinan kunci. Jika diasumsikan bahwa sebuah komputer dapat mencoba 1 juta kemungkinan kunci dalam 1 detik, maka dibutuhkan kurang lebih 2284 tahun untuk mencoba semua kemungkinan kunci.

Meski demikian, V *cipher* juga menggunakan *seed* yang dibangkitkan dari kunci eksternal yang dimasukkan oleh pengguna, yang panjangnya tidak dibatasi asalkan tidak kosong. Nilai *seed* ini sangat bergantung dari masing-masing karakter yang menyusun kunci eksternal yang asli, sehingga perubahan, penghilangan, atau penambahan 1 karakter saja akan langsung merubah *seed* yang akan dihasilkan. Jika

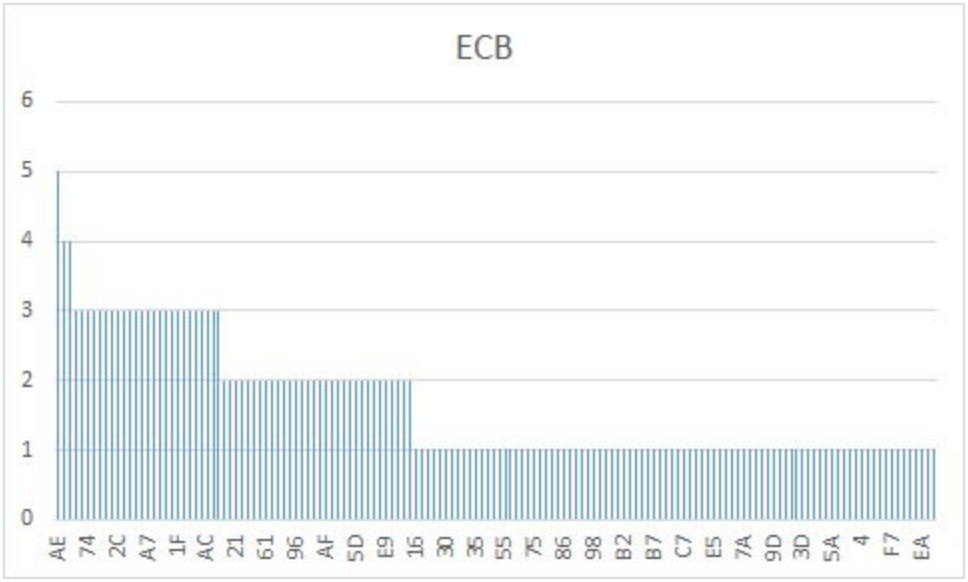
diasumsikan bahwa *seed* yang berbeda akan memberikan urutan bilangan yang selalu berbeda, maka tanpa mengetahui panjang dari kunci eksternal serta urutan karakter pada kunci eksternal yang diberikan pengguna, maka akan sangat sulit untuk memecahkan *ciphertext* dengan metode *Brute Force*, meskipun 56 *bit* yang digunakan untuk pembangkitan kunci internal dapat dipecahkan.

4.4 Analisis Statistik

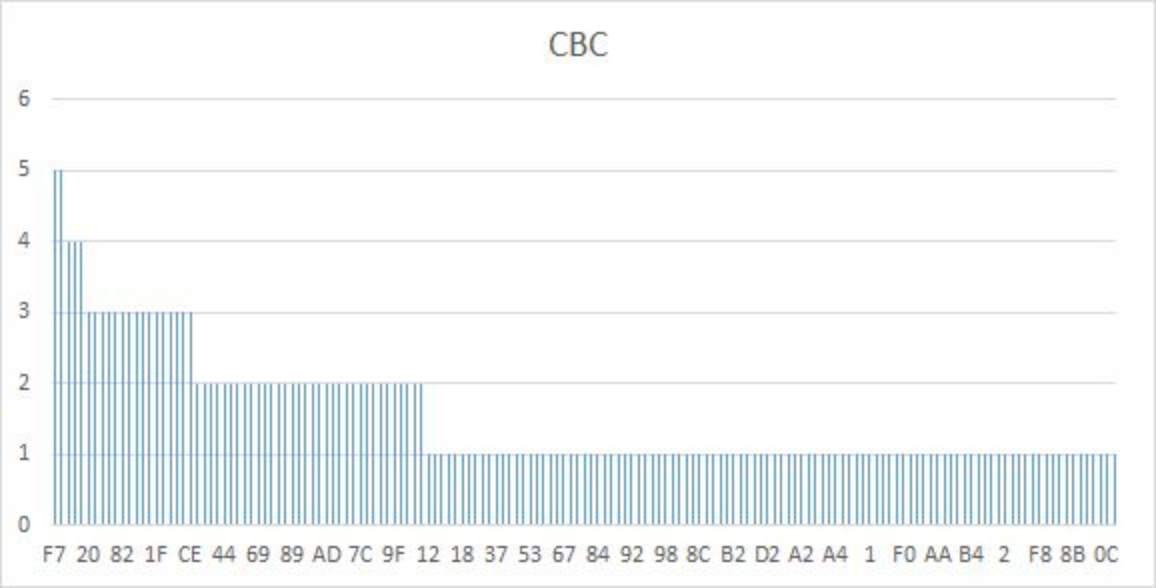
Analisis secara statistik dilakukan dengan membandingkan jumlah frekuensi kemunculan *hexadecimal* pada *plaintext* dan hasil enkripsi dengan mode ECB, CBC, dan Counter.



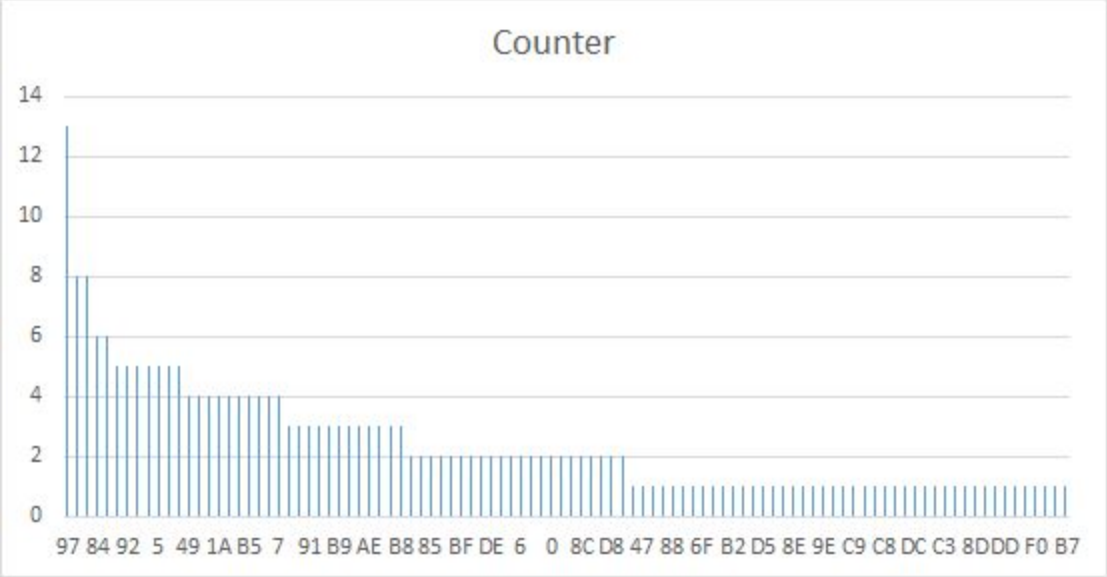
Gambar 10 *Chart diagram* frekuensi kemunculan *hexadecimal* pada *plaintext*



Gambar 11 *Chart diagram* frekuensi kemunculan *hexadecimal* pada hasil enkripsi mode ECB



Gambar 12 *Chart diagram* frekuensi kemunculan *hexadecimal* pada hasil enkripsi mode CBC



Gambar 12 *Chart diagram* frekuensi kemunculan *hexadecimal* pada hasil enkripsi mode Counter

Dari hasil analisis di atas, dapat dilihat bahwa *plaintext* memiliki frekuensi *hexadecimal* berulang yang cukup tinggi. Sementara pada hasil mode ECB dan CBC, frekuensi *hexadecimal* lebih tersebar merata, dengan jumlah kemunculan terbanyak hanya 5 kali. Mode Counter juga tidak jauh berbeda meskipun kemunculan terbanyak lebih tinggi dari 2 mode lainnya, yaitu sebanyak 12 kali. Maka dapat disimpulkan bahwa algoritma V *Cipher* sudah cukup baik dalam meratakan frekuensi *hexadecimal* saat proses enkripsi.

5. Kesimpulan dan Saran

Algoritma V *Cipher* yang didesain sudah memberikan hasil yang baik dalam mengaplikasikan konsep keamanan seperti prinsip *confusion* dan *diffusion*, penangan terhadap serangan *brute force*, dan juga sudah relatif aman jika melihat hasil pada analisis pengulangan *hexadecimal*.

Algoritma V *Cipher* dapat ditingkatkan lebih lanjut dengan menambah pengaruh *randomizer* di bagian-bagian lain pada proses enkripsi. Selain itu, penambahan proses enkripsi baru di antara proses yang sudah ada juga dapat menambahkan kompleksitas dari algoritma saat ini.

6. Referensi

[1] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Kriptografi Modern (Bagian 3). Diakses pada tanggal 21 Oktober 2020

[2] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Kriptografi Modern (Bagian 4). Diakses pada tanggal 21 Oktober 2020

[3] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Data Encryption Standard (DES). Diakses pada tanggal 21 Oktober 2020

[4] Crypto-it.net, Block Ciphers Modes of Operation, http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.html

Acknowledgments

Penulis berterima kasih kepada Tuhan Yang Maha Esa, karena hanya dengan berkat dan bimbingan-Nya Tugas Pengganti UTS ini bisa diselesaikan dengan baik dan tepat waktu. Penulis juga berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. selaku dosen pengampu mata kuliah IF4020 Kriptografi, yang telah memberikan berbagai wawasan terkait kriptografi sehingga algoritma V *Cipher* ini dapat terwujud. Tidak lupa, penulis juga berterima kasih terhadap teman-teman penulis yang telah memberikan berbagai dukungan selama pengerjaan Tugas Pengganti UTS ini.