Synososim Block Cipher

Winston Wijaya[1], **Vincent Chuardi**[2].

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

E-mail: 13517018@std.stei.itb.ac.id, 13517103@std.stei.itb.ac.id

Abstraksi. Kriptografi adalah ilmu yang mempelajari cara menjaga kerahasiaan dan keamanan dari sebuah pesan. Kriptografi modern beroperasi pada tingkat bit pesan. Tetapi operasi yang dilakukan tidak hanya terhadap tiap bit, operasi bisa juga dilakukan pada kumpulan blok bit secara sekaligus. Standar *block cipher* yang terkenal adalah DES (*Data Encryption Standard*). Saat ini, DES menggunakan algoritma DEA (*Data Encryption Algorithm*). *Block cipher* Synososim (*Simple Yet Not So Simple*) adalah *block cipher* varian dari *block cipher* DES.

Keywords: Kriptografi, Bit, *Block cipher*, Synososim, DES, DEA.

1. Pendahuluan

a. Latar Belakang

Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan, integritas data, serta otentikasi (Menez, 1996). Sejarah kriptografi paling tua yang dikenali terdapat pada zaman Mesir Kuno. Kriptografi mengalami perkembangan dari zaman kuno sampai sekarang. Pada zaman sekarang, banyak informasi yang disimpan secara digital. Informasi-informasi tersebut ada yang bersifat publik, tetapi mayoritas informasi-informasi yang disimpan pada sebuah perusahaan bersifat privat. Kemajuan teknologi menyebabkan data-data yang bersifat sensitif dapat dicuri oleh pihak yang tidak bertanggung jawab. Pencegahan pencurian data bisa dilakukan pada berbagai teknologi, tetapi alangkah baiknya keamanan data sensitif tetap dilindungi. Salah satu caranya adalah dengan mengenkripsi data-data sensitif, sehingga hanya pihak berwenang yang dapat melihat data sensitif tersebut.

b. *Review Block Cipher Data Encryption Standard* (DES)

Data Encryption Standard (DES) merupakan salah satu contoh algoritma *block cipher* yang menggunakan kunci simetris dalam proses enkripsi maupun dekripsi. Algoritma ini menerapkan beberapa hal, diantaranya struktur Feistel, prinsip *diffusion*, prinsip *confusion*, substitusi dan transposisi menggunakan s-box dan d-box, dan melakukan sejumlah putaran (*iterated cipher*) menggunakan 16 putaran dengan setiap putarannya memanfaatkan kunci putaran (*round key*) yang dibangkitkan dengan melakukan pergeseran (*shifting*) dan permutasi pada kunci awal yang diberikan dengan algoritma tertentu. DES awalnya terlihat menggunakan 56 bit kunci untuk enkripsinya, namun sebenarnya kunci yang

digunakan adalah 64 bit, tetapi manghapus bit kunci setiap kelipatan 8 sehingga seolah-olah memanfaatkan 56 bit saja. Dengan mengetahui prinsip DES yang memanfaatkan kunci sepanjang 56 bit tersebut, kriptanalisis dapat kebingungan dan ragu untuk mendekripsi secara *brute force* pesan yang dienkripsi karena ada sekitar 2^{56} kombinasi kunci yang mungkin.

Tahapan-tahapan yang dilakukan untuk enkripsi algoritma ini kurang lebih sebagai berikut.

1. Kunci berukuran 64 bit ditransformasikan menjadi kunci berukuran 56 bit dengan membuang bit kunci yang berada di kelipatan 8.
2. Sebanyak 48 bit (*subkey*) dari 56 bit (*key* hasil transformasi) akan digunakan sebagai kunci putar (*round key*) untuk enkripsi setiap iterasi dengan melakukan permutasi pada key hasil transformasi dan tabel *parity* bit yang sudah ditentukan.
3. *Plaintext* berukuran 64 bit yang ingin dienkripsi mula-mula dipermutasikan terlebih dahulu dengan tabel permutasi yang telah ditentukan dan kemudian dibagi menjadi 2 partisi yakni partisi pertama dengan bit-bit dari karakter pertama hingga setengah panjang bit total dan partisi kedua adalah bit sisanya.
4. Partisi kedua dari *plaintext* diperluas menjadi 48 bit dengan menggunakan permutasi tabel d-box dan di xor dengan *round key* pada iterasi tersebut, kemudian dilakukan penyusunan ulang memanfaatkan tabel permutasi awal untuk dikembalikan ke ukuran semula.
5. Partisi pertama kemudian di xor dengan hasil penyusunan ulang dari partisi kedua kemudian kedua partisi tersebut digabung dan diatur kembali menjadi *ciphertext* pada iterasi tersebut.
6. Ulangi langkah ke-4 dan ke-5 hingga sudah dilakukan iterasi sebanyak 16 kali dan *ciphertext* pada iterasi terakhir merupakan *ciphertext* akhir yang diperoleh.
7. Proses dekripsi memanfaatkan langkah yang sama dengan enkripsi namun dengan membalik urutan *round key* yang digunakan pada proses enkripsi.

Untuk penerapan struktur Feistel, penerapan tersebut dapat dilihat pada proses enkripsi dan dekripsi yang dilakukan dimana proses enkripsi dan dekripsi dilakukan dengan algoritma yang sama, namun membalikkan *round key* pada prosesnya. Prinsip *confusion* dan *diffusion* diterapkan pada setiap iterasi dengan melakukan substitusi dan transposisi pada *key* dan *plaintext* selama proses enkripsi dan dekripsi.

c. Gagasan Block Cipher Synososim

Algoritma *block cipher* DES sudah menjadi sebuah algoritma kriptografi yang terkenal, sehingga banyak pihak yang mencoba untuk memecahkan *block cipher* DES dan berhasil. Meskipun *resource* komputasi yang diperlukan untuk memecahkan algoritma DES tidak sedikit, algoritma *block cipher* DES sudah dianggap tidak aman lagi. Algoritma DES juga memiliki tabel-tabel yang sama/tidak berubah/ statik tidak bergantung pada kunci. Oleh karena itu, kami menawarkan gagasan block cipher baru yang bernama *Synososim* (*Simple Yet Not So Simple*). Algoritma *block cipher Synososim* kami mengubah bagian-bagian tabel DES yang statik menjadi tabel p*seudo-random* dengan memanfaatkan *key* awal yang diberikan. Selain itu, angka putaran yang dilakukan juga bergantung terhadap *key* yang diberikan dengan minimal putaran sebanyak 16 kali.

2. Studi Pustaka

a. Block Cipher

Terdapat banyak jenis teknik kriptografi modern yang dapat digunakan, salah satunya adalah *block cipher*. *Block cipher* adalah teknik melakukan enkripsi dalam blok-blok *bit* sebuah pesan. Ukuran blok bit yang digunakan tidak terikat ketentuan apapun, dapat digunakan ukuran blok bit 4, 8, 64, dst. Mode-mode operasi *block cipher* adalah *Electronic Code Book* (ECB), *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB), dan *Counter Mode*. Berikut *review* ringkas terhadap blok cipher sejenis.

1. *Electronic Code Book* (ECB)
   Pada ECB, setiap blok *plaintext* P_i dienkripsi secara individu dan independen terhadap blok lainnya dan kemudian menghasilkan *ciphertext* C_i. Proses enkripsi pada ECB memanfaatkan fungsi enkripsi E tertentu, bentuk paling sederhananya adalah melakukan XOR dengan bit *key* yang digunakan kemudian setiap bit digeser 1 posisi ke kiri. Untuk melakukan dekripsi, maka dilakukan langkah sebaliknya, yakni setiap blok bit pada *ciphertext* C_i dimasukkan ke dalam fungsi dekripsi D, misal terhadap fungsi sederhana untuk enkripsi sebelumnya, maka *ciphertext* digeser 1 posisi ke kanan kemudian di XOR dengan bit *key* yang digunakan.

2. *Cipher Block Chaining* (CBC)
   Berbeda dengan ECB, enkripsi suatu blok *plaintext* bergantung pada hasil enkripsi dari blok sebelumnya kecuali blok pertama.
   Proses enkripsi pada blok pertama dilakukan dengan melakukan operasi XOR antara bit-bit *plaintext* dengan blok *Initialize Vector* (IV) yang dimasukkan oleh pengguna atau dibangkitkan oleh program. Setelah itu blok hasil operasi XOR dienkripsi dengan fungsi enkripsi E, misal yang paling sederhana seperti di ECB, dengan melakukan operasi XOR terhadap bit-bit kunci kemudian setiap bitnya digeser 1 posisi ke kiri. Untuk enkripsi pada blok-blok lain dilakukan dengan cara yang sama kecuali untuk *Initialize Vector* (IV) digantikan dengan *ciphertext* hasil enkripsi blok sebelumnya.
   Proses dekripsi dilakukan dengan langkah sebaliknya. Pada blok pertama, bit-bit *ciphertext* dan bit-bit kunci dimasukkan ke dalam suatu fungsi dekripsi D, misal berkaitan dengan fungsi enkripsi sebelumnya, maka dilakukan pergeseran sebanyak 1 posisi untuk setiap bit kemudian hasil pergeseran di XOR dengan bit-bit kunci, selanjutnya bit-bit hasil dari fungsi dekripsi di XOR dengan nilai *Initialize Vector*. Nilai pada blok sebelum dekripsi akan digunakan sebagai nilai *Initialize Vector* pada blok berikutnya dengan mengulangi langkah yang sama, dimulai dengan memasukkan bit-bit *ciphertext* dan kunci ke dalam fungsi dekripsi dan kemudian di XOR dengan *Initialize Vector*.

3. *Cipher Feedback* (CFB)
   Pada *Cipher Feedback* (CFB), enkripsi dilakukan pada unit-unit kecil (n) yang ukurannya lebih kecil dibandingkan dengan ukuran dari 1 blok, misal bila blok m berukuran 64 bit (8 karakter), maka ukuran unitnya dapat bervariasi mulai dari 1 bit, 2 bit, 4 bit, dan ukuran lainnya selama ukuran unit lebih kecil atau sama dengan ukuran blok. CFB memerlukan sebuah *queue* seukuran blok untuk melakukan enkripsi dan dekripsi terhadap unit-unit *plaintext* dan *ciphertext*.

Secara formal, CFB n-bit dapat dijelaskan sebagai berikut[2].
Enkripsi:

$$C_i = P_i \oplus MSB_m(E_k(X_i)$$

$$X_{i+1} = LSB_{m-n}(X_i) \,||\, Ci$$

Dekripsi:

$$P_i = C_i \oplus MSB_m(D_k(X_i)$$

$$X_{i+1} = LSB_{m-n}(X_i) \,||\, Ci$$

Keterangan:
X_i = isi antrian dengan X_1 adalah IV
E = fungsi enkripsi dengan algoritma *block cipher*
D = fungsi dekripsi dengan algoritma *block cipher*
K = kunci
m = panjang blok enkripsi
n = panjang unit enkripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

Misal CFB dengan ukuran 8 bit (CFB 8-bit) dengan blok berukuran 64 bit.
Proses enkripsi dimulai dengan membagi blok *plaintext* menjadi unit-unit dengan ukuran 8 bit dan dimasukkan ke dalam *queue*. Kunci yang digunakan (IV pada enkripsi dan dekripsi untuk unit pertama) dimasukkan ke dalam fungsi enkripsi E, kemudian MSB hasil fungsi enkripsi di XOR dengan unit *plaintext* dan di *concat* untuk menghasilkan *ciphertext* dari unit tersebut. Langkah tersebut diulang hingga semua unit *plaintext* sudah berubah menjadi *ciphertext*. *Ciphertext* yang diperoleh di akhir perulangan kemudian di *pass* sebagai *Initial Vector* untuk enkripsi pada blok berikutnya.
Proses dekripsi kurang lebih dilakukan seperti saat enkripsi, yang menjadi pembeda adalah fungsi enkripsi digantikan dengan fungsi dekripsi yang sesuai terhadap fungsi enkripsi dan seperti pada CBC, *ciphertext* sebelumnya akan digunakan sebagai *Initial Vector* untuk blok berikutnya.

4. *Output Feedback* (OFB)
   Output Feedback (OFB) merupakan mode yang proses enkripsi dan enkripsinya mirip dengan CFB, namun pada OFB, bit-bit hasil enkripsi disalin menjadi elemen paling kanan di antrian[3].

5. *Counter Mode*
   Pada *Counter Mode* tidak terjadi proses berantai seperti yang terjadi pada CBC, FCB, dan OFB. Enkripsi setiap blok *plaintext* dilakukan secara individu dan independen dengan blok kunci yang digunakan, namun nilai *counter* yang digunakan untuk enkripsi akan berhubungan dengan nilai *counter* pada blok sebelumnya (nilai *counter* dinaikkan sebesar 1 untuk blok berikutnya) kecuali nilai inisialisasi awal *counter* di blok pertama.

Proses enkripsi dilakukan dengan mengenkripsi blok kunci dengan *counter* menggunakan suatu fungsi enkripsi E, misal fungsi sederhana yang dapat digunakan yakni seperti pada ECB, melakukan XOR untuk setiap bit pada kedua blok kemudian digeser 1 posisi ke kiri untuk setiap bitnya. Selanjutnya nilai hasil enkripsi sebelumnya di XOR dengan blok-blok pada *plaintext*.
Proses dekripsi dilakukan dengan melakukan langkah kebalikannya. Pertama blok *counter* dan blok kunci dimasukkan ke dalam fungsi dekripsi D, kemudian hasil dari fungsi tersebut di XOR dengan blok *ciphertext* untuk memperoleh *plaintext* semula.

b. Konsep Matematika

Dalam pembangunan, konsep matematika yang digunakan adalah *logical shift*. *Logical shift* merupakan operasi *bitwise* yang dilakukan pada suatu bilangan yang secara matematis dapat digunakan untuk menghitung nilai bilangan tersebut dikalikan atau dibagi dengan 2 pangkat jumlah pergeseran bit yang dilakukan. Terdapat 2 jenis *Logical shift*, yakni *Left Logical Shift* dan *Right Logical Shift*. *Left logical shift* melakukan pergeseran setiap bit ke kiri sebanyak *n* kali dengan membuang beberapa bit awal, termasuk *Most Significant Bit* (MSB) dan menambahkan bit 0 sebanyak *n* sebagai *n-1* bit tambahan dan *Least Significant Bit* (LSB). Dalam operasi matematis dengan melibatkan bilangan, maka *left logical shift* mengalikan bilangan yang dimasukkan dengan 2 pangkat jumlah pergeseran bit, sebaliknya *right logical shift* menggeser semua bit ke kanan sebanyak n kali dengan membuang *n* bit paling kanan dan menambahkan *n* bit 0 di kiri dari bit yang tersisa dan secara matematis bisa dikatakan membagi suatu bilangan dengan 2 pangkat pergeseran shift dengan pembulatan ke bawah.

Shifting yang diterapkan pada pada algoritma yang kami ciptakan tidak melakukan pergeseran yang menghapus *n* bit awal ataupun akhir tetapi melakukan *shifting* dengan memutar semua bit yang ada seperti melakukan sebuah putaran. Misal dilakukan *left shifting* sebanyak 1 kali, maka semua bit selain MSB akan bergeser 1 posisi ke kiri sebanyak 1 kali sedangkan MSB akan berpindah posisi ke LSB dan berlaku pula sebaliknya.

3. **Proposed Block Cipher**

a. Kami menggunakan kodingan DES Geeksforgeeks[3] sebagai referensi. Algoritma DES GFG sudah menerapkan struktur *Feistel* dalam algoritmanya, prinsip *diffusion* dan *confusion* Shanon, substitusi dengan kotak-S, transposisi, dan *iterated cipher* sebanyak 16 kali. Kami mengubah bagian substitusi dengan kotak-S menjadi lebih dinamis dan fleksibel dengan memanfaatkan kunci. *Iterated cipher* menjadi fleksibel dengan memanfaatkan kunci. Kami juga mengubah fungsi-F pada *Feistel network*.

b. Variabel-variabel yang digunakan adalah variabel tabel-tabel permutasi, tabel ekspansi D, tabel s-box, tabel *parity bit drop*, tabel *bit shift*, tabel kompresi kunci, dan variabel *list round key*.
Variabel-variabel tabel permutasi yang digunakan, yaitu *Initial Permutation* plainteks (initial_perm), *Straight Permutation Table* (per), dan *Final Permutation Table* (final_perm). Nilai dari *list* permutasi menunjukan posisi setelah ditransposisi.
Tabel Ekspansi D berfungsi untuk melebarkan bit dengan ukuran 32 bit menjadi 48 bit.
Tabel S-Box digunakan untuk melakukan substitusi pada jaringan *Feistel*.
Tabel *parity bit drop* digunakan untuk mereduksi kunci 64 bit menjadi 56 bit.
Tabel *bit shift* digunakan untuk operator bit geser kiri pada saat men-*generate* kunci-kunci putaran.
Tabel kompresi kunci digunakan untuk mereduksi kunci 56 bit menjadi 48 bit.
List round key adalah variabel untuk menyimpan kunci-kunci putaran yang di-*generate*.

c. Algoritma enkripsi dan dekripsi sama karena memakai struktur jaringan *Feistel*.
Input: plainteks dalam bentuk *string*, *list* kunci iterasi *binary*, dan list kunci iterasi *hex*.
Output: *ciphertext*
Proses:
   i. Plainteks diubah ke bentuk *binary*
   ii. Plainteks di transposisi dan dibagi 2 menjadi *left* dan *right* plainteks
   iii. Iterasi proses enkripsi sebanyak jumlah iterasi yang didapatkan di awal. Proses pada jaringan *Feistel*:
      1. *Right* plainteks di ekspansi dari 32 bit menjadi 48 bit dan ditransposisi
      2. *Right* plainteks di XOR dengan kunci putaran yang bersesuaian
      3. *Right* plainteks di-*shift* ke kiri sebanyak *n*; dimana *n* adalah bilangan yang dibangkitkan random menggunakan *seed* berupa *key* dengan *range* 1 sampai dengan 63.
      4. *Right* plainteks di substitusi dengan tabel S-box
      5. *Right* plainteks di transposisi kembali menjadi 32 bit
      6. *Left* plainteks di XOR dengan *right* plainteks
      7. *Left* menjadi *right* plainteks dan *right* menjadi *left* plainteks
   iv. Konkatenasi *left* plainteks dan *right* plainteks.
   v. *Ciphertext* di transposisi.

d. Jumlah putaran ditentukan secara random dengan *seed* kunci di awal. Jumlah putaran dibangkitkan dengan menggunakan fungsi *pseudo-random* dengan nilai minimal 16.

e. Pembangkitan kunci putaran sebanyak jumlah putaran:
   i. Kunci diubah dari *string* menjadi bentuk *hexadecimal* dan selanjutnya menjadi bentuk *binary*.
   ii. Transposisi dan reduksi kunci *binary* dari 64 bit menjadi 56 bit.
   iii. Tabel *bit shift* dibangkitkan dengan jumlah elemen berjumlah jumlah putaran. Kemungkinan nilai *shift* tabel adalah 1 atau 2.
   iv. Kunci dibagi menjadi 2 bagian 28 bit, *left* dan *right*. Masing-masing bagian di-*shift* ke kiri sesuai dengan nilai tabel *bit shift*.
   v. Kunci di konkatenasi dan dikompresi dari 56 bit menjadi 48 bit

f. Kotak S

Kotak S yang digunakan pada algoritma yang dibangkitkan menggunakan *library random* dengan *seed* "KRIPTO2020". Berikut adalah S-Box yang dihasilkan.

```
S-Box = [
  [
    [2, 1, 6, 10, 5, 8, 4, 9, 14, 3, 7, 0, 12, 13, 15, 11],
    [3, 11, 7, 9, 4, 10, 12, 13, 6, 14, 2, 1, 0, 8, 5, 15],
    [7, 0, 6, 4, 5, 11, 2, 3, 15, 10, 13, 9, 1, 8, 12, 14],
    [11, 2, 1, 3, 0, 15, 14, 6, 8, 5, 10, 13, 12, 7, 4, 9]
  ],
  [
    [11, 6, 3, 2, 8, 0, 14, 12, 1, 4, 13, 7, 9, 10, 15, 5],
    [15, 9, 6, 10, 2, 0, 11, 12, 5, 14, 1, 13, 3, 7, 4, 8],
    [12, 13, 2, 15, 10, 1, 9, 11, 14, 6, 0, 3, 4, 7, 5, 8],
    [5, 9, 8, 13, 15, 1, 0, 3, 4, 12, 7, 14, 10, 11, 6, 2]
  ],
  [
    [0, 7, 9, 4, 10, 6, 15, 1, 5, 8, 2, 14, 12, 11, 3, 13],
    [4, 11, 12, 7, 5, 14, 13, 9, 6, 1, 3, 2, 15, 10, 0, 8],
    [15, 1, 0, 13, 11, 8, 5, 12, 3, 10, 9, 7, 2, 6, 14, 4],
    [12, 2, 6, 3, 4, 15, 8, 9, 14, 11, 5, 7, 0, 1, 13, 10]
  ],
  [
    [6, 3, 10, 13, 11, 12, 8, 14, 2, 9, 4, 5, 15, 7, 1, 0],
    [0, 12, 5, 6, 2, 15, 1, 7, 13, 11, 14, 8, 3, 10, 9, 4],
    [6, 13, 9, 10, 4, 11, 8, 1, 0, 14, 5, 12, 15, 7, 3, 2],
    [13, 4, 1, 10, 6, 14, 5, 15, 8, 7, 11, 2, 12, 3, 9, 0]
  ],
  [
    [6, 5, 11, 2, 10, 15, 13, 4, 3, 12, 1, 7, 8, 0, 9, 14],
    [9, 13, 2, 14, 0, 6, 4, 7, 15, 12, 5, 11, 1, 10, 3, 8],
    [1, 4, 2, 0, 7, 11, 3, 10, 15, 8, 6, 9, 12, 5, 14, 13],
    [3, 1, 13, 9, 11, 7, 10, 4, 2, 14, 8, 5, 15, 6, 0, 12]
  ],
```

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

g. Operasi permutasi.

Konsep permutasi yang dilakukan pada blok bit dapat dilihat pada pseudocode dibawah.

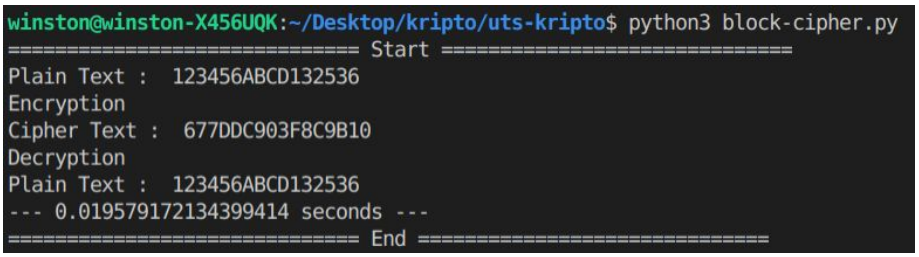
```
perm = [2, 1, 5, 3, 4]
string = ‘12345’
print(permute(string, perm, 5)) # menghasilkan string ‘21534’
```

h. Tabel yang di random

- Tabel parity kunci
- Tabel shift kunci putaran
- Tabel kompresi kunci

4. Eksperimen dan Analisis Hasil

a. *Screenshot* Program



b. Hasil Pengujian

i. Menggunakan CBC

1. Original

Ukuran pesan: 16 *byte hex*
Pesan : `ABCDEF0123456789`
Ukuran kunci : 16 *byte hex*
Kunci : `ACE0246897531EDB`
Waktu enkripsi : 0.003437042236328125 seconds
Waktu dekripsi : 0.002682209014892578 seconds
Hasil:

```
Plain Text : ABCDEF0123456789

Encryption
Cipher Text : B265E5661217D234

Decryption
Plain Text : ABCDEF0123456789
```

2. Mengubah 1 *byte* kunci

Ukuran pesan: 16 *byte hex*
Pesan : `ABCDEF0123456789`
Ukuran kunci : 16 *byte hex*
Kunci : `ACE0246897531EDB → ACE0246897531EDC`
Waktu enkripsi : 0.0021851062774658203 seconds
Waktu dekripsi : 0.012449077606201172
Hasil:

```
Plain Text : ABCDEF0123456789

Encryption
Cipher Text : B265E5661217D234

Decryption
Plain Text : D6F2B61A61887650
```

3. Mengubah 1 *byte ciphertext*
   Ukuran pesan: 16 *byte hex*
   Pesan : `ABCDEF0123456789`
   Ukuran kunci : 16 *byte* hex
   Kunci : `ACE0246897531EDB`
   Waktu enkripsi : 0.003977298736572266 seconds
   Waktu dekripsi : 0.001979827880859375 seconds
   Hasil:

   Plain Text : ABCDEF0123456789

   Encryption
   Cipher Text : B265E5661217D234 → B265E5661217D233

   Decryption
   Plain Text : 4A32EE035654D19D

ii. Menggunakan ECB

1. Original
   Ukuran pesan: 16 *byte hex*
   Pesan : `ABCDEF0123456789`
   Ukuran kunci : 16 *byte hex*
   Kunci : `ACE0246897531EDB`
   Waktu enkripsi : 0.009460687637329102 seconds
   Waktu dekripsi : 0.007559776306152344 seconds
   Hasil:

   Plain Text : ABCDEF0123456789

   Encryption
   Cipher Text : 80B98A3CC0BB47B0

   Decryption
   Plain Text : ABCDEF0123456789

2. Mengubah 1 *byte* kunci
   Ukuran pesan: 16 *byte hex*
   Pesan : `ABCDEF0123456789`
   Ukuran kunci : 16 *byte hex*
   Kunci : `ACE0246897531EDB → ACE0246897531EDC`
   Waktu enkripsi : 0.007675647735595703 seconds
   Waktu dekripsi : 0.008378267288208008 seconds
   Hasil:

Plain Text : ABCDEF0123456789

Encryption
Cipher Text : 80B98A3CC0BB47B0

Decryption
Plain Text : 098828080B626B6F

3. Mengubah 1 *byte ciphertext*
   Ukuran pesan: 16 *byte hex*
   Pesan : `ABCDEF0123456789`
   Ukuran kunci : 16 *byte hex*
   Kunci : `ACE0246897531EDB`
   Waktu enkripsi : 0.006811380386352539 seconds
   Waktu dekripsi : 0.006911754608154297 seconds
   Hasil:

Plain Text : ABCDEF0123456789

Encryption
Cipher Text : 80B98A3CC0BB47B0 → 80B98A3CC0BB47B1

Decryption
Plain Text : 6A497ACAC4EBA126

iii. Menggunakan Mode *Counter*

1. Original
   Ukuran pesan: 16 *byte hex*
   Pesan : `ABCDEF0123456789`
   Ukuran kunci : 16 *byte hex*
   Kunci : `ACE0246897531EDB`
   Waktu enkripsi : 0.009602069854736328 seconds
   Waktu dekripsi : 0.017072439193725586 seconds
   Hasil:

Plain Text : ABCDEF0123456789

Encryption
Cipher Text : F692EF1007327DBC

Decryption
Plain Text : ABCDEF0123456789

2. Mengubah 1 *byte* kunci
   Ukuran pesan: 16 *byte hex*

Pesan : ABCDEF0123456789
Ukuran kunci : 16 *byte hex*
Kunci : ACE0246897531EDB → ACE0246897531EDB
Waktu enkripsi : 0.016776561737060547 seconds
Waktu dekripsi : 0.016249418258666992 seconds
Hasil:

Plain Text : ABCDEF0123456789 Encryption Cipher Text : F692EF1007327DBC Decryption Plain Text : D6F2B61A61887650

3. Mengubah 1 *byte ciphertext*
   Ukuran pesan: 16 *byte hex*
   Pesan : ABCDEF0123456789
   Ukuran kunci : 16 *byte hex*
   Kunci : ACE0246897531EDB:
   Waktu enkripsi : 0.003549814224243164 seconds
   Waktu dekripsi : 0.0019371509552001953 seconds
   Hasil:

Plain Text : ABCDEF0123456789 Encryption Cipher Text : F692EF1007327DBC → F692EF1007327DBB Decryption Plain Text : 0D30C87511B874D3

c. Analisis Hasil Implementasi *Block Cipher* yang Dibuat

Waktu eksekusi untuk algoritma DES referensi lebih cepat dari *block cipher* yang dibuat, karena DES tidak memiliki komputasi untuk membuat random tabel-tabel yang ada.

Mode komputasi paling berat terdapat pada mode *Counter*, karena pada pembangkitan kunci terdapat tambahan variabel yang bertambah tiap putaran. Pada mode CBC dan EBC tidak terdapat variabel tersebut.

Mode komputasi paling ringan terdapat pada mode ECB, karena tidak terdapat perubahan apapun pada bit-bit kunci.

5. Kesimpulan dan Saran Pengembangan (Future Works)

Menurut teori, algoritma *block cipher Synososim* adalah algoritma yang lebih aman daripada algoritma DES. Karena jumlah putaran enkripsi kemungkinan lebih banyak, terdapat operasi bit tambahan terhadap proses penentuan *ciphertext*, dan memiliki jumlah putaran yang berbeda sesuai kunci.

Kelemahan dari tambahan ini adalah waktu enkripsi yang diperlukan menjadi lebih lama dan masih tidak efisien, sehingga algoritma *Synososim* belum bisa digunakan untuk enkripsi secara *real-time*. Selain itu, ketika jumlah putaran yang digunakan dan di-*generate* secara *pseudo-random* tiba-tiba berjumlah ratusan, pasti performa akan menjadi sangat lambat.

Saran pengembangan selanjutnya adalah meningkatkan efisiensi dari algoritma *block cipher Synososim*.

6. References

[1] Pengantar Kriptografi (2020)
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Pengantar-Kriptografi-(2020).pdf

[2] Kriptografi modern (Bagian 3: Block Cipher)
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Kripto-modern-2020-bagian3.pdf

[3] Algoritma DES
https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/

[4] Data Encryption Standard (DES)
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Review-beberapa-block-cipher-dan-stream-cipher-2020-bagian1.pdf

Ucapan Terima Kasih

Kami ucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT. telah membimbing kami ilmu-ilmu kriptografi *block cipher*. Berbekal pengetahuan yang kami dapatkan, kami berhasil menyusun laporan *block cipher* berjudul '*Synososim Block Cipher*' ini. Kami juga mengucapkan terima kasih untuk penyedia referensi-referensi yang kami gunakan. Tanpa ilmu-ilmu tambahan yang didapatkan dari referensi yang kami gunakan, jurnal ini tidak akan mencapai tahap ini.