# Pembuatan *Block Cipher* "Cripopo"

Harith Fakhiri Setiawan (13519161)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13519161@std.stei.itb.ac.id

M. Ibnu Syah Hafizh (13519177)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13519177@std.stei.itb.ac.id

Ryan Kurnia Hidayatullah (NIM: 13519212)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13519212@std.stei.itb.ac.id

***Abstract*—Dalam era digital saat ini, keamanan data sangat penting. Setiap hari, miliaran byte data dipertukarkan di seluruh dunia melalui internet, termasuk data sensitif seperti informasi keuangan, kesehatan, pemerintah, dan militer. Penting untuk memastikan bahwa data tersebut tidak dapat diakses oleh pihak yang tidak berwenang. Oleh karena itu, pengembangan algoritma kriptografi terus dilakukan untuk melindungi data-data tersebut. Pada makalah ini, akan dibahas mengenai Cripopo yaitu algoritma block cipher baru yang diharapkan dapat memberikan keamanan terhadap data penting yang disimpan dalam bentuk digital.**

***Keywords—kriptografi; block cipher; Cripopo***

## I. PENDAHULUAN

Block cipher adalah sebuah algoritma kriptografi yang digunakan untuk mengenkripsi sejumlah besar data dengan ukuran yang sama, yaitu blok. Algoritma ini bekerja dengan memisahkan data menjadi blok-blok yang sama ukurannya, kemudian melakukan operasi enkripsi pada setiap blok secara terpisah. Dalam perancangan block cipher, terdapat beberapa faktor yang harus dipertimbangkan, seperti efisiensi enkripsi dan dekripsi, keamanan, dan kecepatan. Cipher blok kerap digunakan di era digital ini, contoh algoritmanya adalah DES [1] dan AES [2], dengan ukuran blok dan algoritma yang berbeda-beda pula [1][2]. Kedua cipher tersebut juga mengimplementasikan substitusi dengan sebuah mapping khusus (S-Box), yang didefinisikan secara terpisah.

Perancangan block cipher penting dilakukan karena penggunaan algoritma kriptografi yang baik sangat diperlukan dalam menjaga keamanan data. Saat ini, banyak sekali data penting yang disimpan dalam bentuk digital, seperti informasi perbankan, data pribadi, dan dokumen bisnis. Oleh karena itu, diperlukan sebuah algoritma kriptografi yang dapat memberikan keamanan maksimal terhadap data tersebut.

Dalam perancangan block cipher, ada beberapa aspek yang harus diperhatikan. Pertama, efisiensi algoritma harus dipertimbangkan agar dapat diimplementasikan dengan cepat pada berbagai platform. Kedua, keamanan algoritma harus dipertimbangkan agar tidak mudah diretas atau ditembus oleh pihak yang tidak berwenang. Ketiga, ukuran blok harus dipilih dengan hati-hati agar dapat menangani sejumlah besar data dengan baik.

Perancangan block cipher yang baik juga harus mempertimbangkan beberapa teknik enkripsi yang dapat digunakan, seperti teknik penggabungan (substitution-permutation network), teknik enkripsi berbasis kunci publik, dan teknik penggabungan berbasis blok. Setiap teknik memiliki keunggulan dan kelemahan masing-masing, oleh karena itu, pemilihan teknik yang tepat harus dilakukan sesuai dengan kebutuhan dan tingkat keamanan yang diinginkan.

Dalam makalah ini, akan dibahas mengenai perancangan block cipher baru dan beberapa faktor yang harus diperhatikan, seperti efisiensi, keamanan, dan teknik enkripsi yang dapat digunakan. Dengan memahami hal-hal tersebut, diharapkan akan tercipta sebuah algoritma kriptografi yang aman dan efisien dalam melindungi data penting dari akses yang tidak sah.

## II. DASAR TEORI

### *A. Block Cipher*

Block cipher adalah algoritma yang mengenkripsi satu blok bit plaintext x ke blok bit ciphertext y.Transformasi

plain text tersebut dilakukan menggunakan kunci rahasia K, dengan EK(x) = y. Setiap kunci mendefinisikan pemetaan dari blok plaintext ke blok ciphertext. Untuk ukuran blok adalah b bit,jumlah nilai input yang berbeda ke block cipher adalah 2^b. Hal ini berarti bahwa sebuah block cipher dapat dipandang sebagai cipher dari alfabet ukuran 2^b. Walau demikian, hanya sebagian kecil dari semua kemungkinan permutasi pada ukuran alfabet 2^b yang akan sesuai dengan kunci tertentu di block cipher.

.

*B. Confusion & Diffusion*

Claude Shannon mengidentifikasi 2 properti dari secure cipher yaitu confusion dan diffusion[3]. Confusion berarti setiap digit biner atau bit dari ciphertext harus bergantung pada beberapa bagian dari kunci[3]. Confussion bertujuan untuk menyembunyikan hubungan statistik yang ada diantara plainteks, cipherteks, dan kunci. Diffusion berarti bahwa jika satu bit plaintext atau kunci diubah maka perubahan yang terjadi pada ciphertext harus signifikan[3]. Kedua properti ini mengaburkan hubungan antara ciphertext, plaintext, dan kunci.

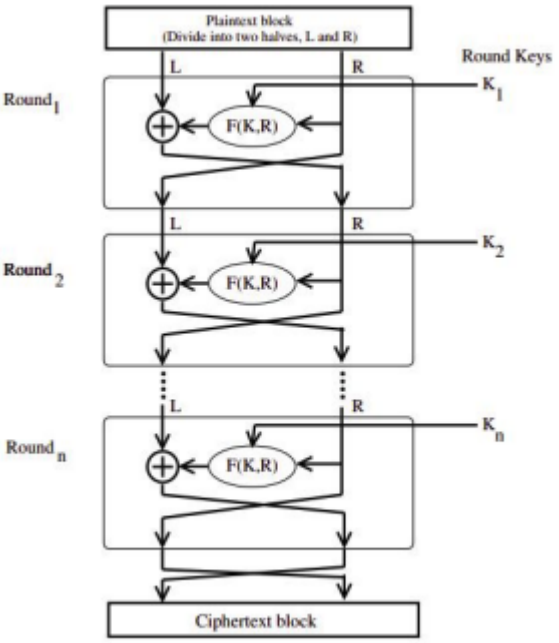
*C. Jaringan Feistel*

Feistel network adalah sistem kriptografi yang menggunakan algoritma yang sama untuk prosesenkripsi dan dekripsinya. Cara kerja Feistel Network dalam kriptografi block cipher adalah dengan membagi plaintext menjadi dua bagian yang sama besar, dan kemudian melakukan serangkaian transformasi pada blok-blok ini secara bergantian pada setiap putaran. Setiap putaran terdiri dari dua tahap yaitu tahap transformasi dan tahap penggabungan. [1]

Pada tahap transformasi, setengah blok plaintext dioperasikan dengan fungsi non-linear yang disebut F function. F function ini mengambil input dari setengah blok plaintext dan kunci putaran yang dihasilkan dari kunci enkripsi utama. F function ini dapat menggunakan operasi-operasi seperti substitusi, permutasi, dan operasi XOR untuk memproses inputnya. Fungsi ini kemudian menghasilkan sebuah output yang sama besar dengan input.

Setelah itu, output dari F function dioperasikan pada setengah blok plaintext yang lain menggunakan operasi XOR. Operasi XOR menghasilkan nilai output yang sama besar dengan inputnya, dan menerapkan prinsip diffusion dengan menyebarkan pengaruh perubahan satu bit dari plaintext atau kunci pada banyak bit dari ciphertext.

Pada tahap penggabungan, blok-blok yang dihasilkan dari tahap transformasi digabungkan sehingga menjadi ciphertext. Setelah satu putaran selesai dilakukan, blok-blok ini akan dibagi lagi menjadi dua bagian yang sama besar untuk dilakukan transformasi pada putaran selanjutnya. Transformasi ini akan dilakukan sebanyak putaran yang diinginkan hingga mencapai ciphertext yang diinginkan. Berikut adalah diagram alir jaringan feistel,




**Gambar 1. Diagram alir jaringan feistel**

## III. RANCANGAN ALGORITMA

Untuk menyelesaikan permasalahan yang diberikan, yaitu pembuatan *block cipher* baru, dibutuhkan rancangan algoritma untuk *block cipher* yang memenuhi persyaratan pada masalah yang diberikan.

*A. ALUR RANCANGAN UMUM*

Pertama-tama, algoritma ini menerapkan prinsip diffusion dan confusion dari Shannon, dimana substitusi dan transposisi dilakukan pada setiap putaran untuk mengacak blok pesan agar sulit dibaca.

Selanjutnya, algoritma ini memiliki fungsi putaran (f) yang terdiri dari jaringan substitusi-permutasi, dimana setiap putaran terdiri dari operasi substitusi dan transposisi. Aturan substitusi dan transposisi dapat menggunakan tabel substitusi S-box dan tabel permutasi, atau menggunakan pergeseran bit atau byte untuk permutasi, dan digunakan kunci putaran yang berbeda pada setiap putaran.

Selanjutnya, algoritma ini menerapkan cipher berulang, yaitu melakukan enciphering terhadap blok pesan berulang kali dengan jumlah putaran sebanyak 16 kali, dan setiap putaran menggunakan kunci putaran yang berbeda. Ukuran blok pesan yang dienkripsi adalah 128 bit, dan panjang kunci yang digunakan juga adalah 128 bit. Jumlah putaran yang dilakukan adalah 16 kali, dan dapat menggunakan jaringan Feistel untuk mempermudah implementasi algoritma.

Pada setiap putaran, algoritma block cipher akan melakukan beberapa operasi, yaitu melakukan perputaran kanan setengah blok pesan sebanyak 1 kali dengan menggunakan kunci putaran yang berbeda pada setiap putaran, melakukan substitusi pada setiap byte blok pesan dengan menggunakan tabel substitusi S-box yang telah ditentukan, melakukan permutasi pada setiap byte blok pesan dengan menggunakan tabel permutasi yang telah ditentukan, melakukan operasi XOR antara setengah blok pesan yang telah diputar dan blok pesan yang telah melalui substitusi dan permutasi, dan melakukan perputaran kiri setengah blok pesan sebanyak 1 kali dengan menggunakan kunci putaran yang berbeda pada setiap putaran.

Setelah 16 putaran, algoritma block cipher akan menghasilkan blok pesan yang telah dienkripsi. Proses ini dapat diulang untuk setiap blok pesan yang ingin dienkripsi dengan menggunakan kunci yang berbeda untuk setiap blok pesan. Dalam proses dekripsi, algoritma block cipher dapat mengembalikan pesan ke bentuk semula dengan menggunakan operasi-invers pada setiap putaran, yaitu melakukan invers permutasi, invers substitusi, dan invers operasi XOR.

### *B. PEMBANGKITAN KUNCI PUTARAN*

Untuk menghasilkan kunci putaran pada setiap putaran dalam algoritma block cipher Cripopo, digunakan fungsi untuk menghasilkan kunci putaran yang membutuhkan dua parameter yaitu kunci utama (kunci yang diinisiasi diawal) dan nomor putaran saat ini.

Pertama-tama, nomor putaran diubah menjadi sebuah list biner dengan panjang yang sama dengan panjang kunci utama. Hal ini dilakukan untuk memastikan bahwa panjang kunci putaran yang dihasilkan selalu sama dan sejajar dengan panjang kunci utama.

Setelah nomor putaran berupa list biner, fungsi xor operation digunakan untuk melakukan operasi XOR antara kunci utama dan nomor putaran. Hasil operasi XOR tersebut akan menjadi kunci putaran pada putaran saat ini.

Setelah itu, kunci putaran yang telah dihasilkan oleh fungsi ini akan digunakan pada setiap putaran dalam algoritma Cripopo. Dengan demikian, di setiap putaran akan dihasilkan kunci putaran yang berbeda-beda untuk memastikan keamanan dan kerahasiaan dari pesan yang dienkripsi.

.

## IV. EKSPERIMEN DAN PEMBAHASAN HASIL

Program algoritma Cripopo dibangun dengan bahasa pyton dengan implementasi dan hasil sebagai berikut.

### *A. IMPLEMENTASI*

a. Tabel substitusi dan permutasi

Pada algoritma Cripopo digunakan tabel substitusi S_BOX dan tabel permutasi PERMUTATION_TABLE sebagai berikut

```
S_BOX = np.array([
    [0x9, 0x4, 0xA, 0xB],
    [0xD, 0x1, 0x8, 0x5],
    [0x6, 0x2, 0x0, 0x3],
    [0xC, 0xE, 0xF, 0x7]
])
```

```
PERMUTATION_TABLE = np.array([
    0, 4, 8, 12,
    1, 5, 9, 13,
    2, 6, 10, 14,
    3, 7, 11, 15
])
```

Ukuran S_BOX dan PERMUTATION_TABLE pada program tersebut hanya 4x4 karena program tersebut merupakan cipher dengan panjang blok 128-bit dan kunci 128-bit, dengan tujuan untuk memperlihatkan jaringan Feistel dengan 16 putaran. Ukuran 4x4 pada S_BOX dan PERMUTATION_TABLE hanya mencakup 16 elemen, yang cukup untuk menghasilkan operasi substitusi dan permutasi yang cukup acak pada blok 128-bit.

b. Fungsi Substitusi

```
def substitution(input_block, str):
    # print("ib" , input_block)
    input_block =
binary_to_hex(input_block)
    # print("hex", input_block)
    S_BOX = np.array([
        [0x9, 0x4, 0xA, 0xB],
        [0xD, 0x1, 0x8, 0x5],
        [0x6, 0x2, 0x0, 0x3],
        [0xC, 0xE, 0xF, 0x7]
    ])
    output_block =
np.zeros_like(input_block)
    if str=="encrypt":
        for i in
range(len(input_block)-4,
len(input_block)):
            row = input_block[i] >> 4
            col = input_block[i] & 0x0F
            # print(row, col)
            output_block[i] =
S_BOX[row%4][col%4]
```

```
    elif str=="decrypt":
        for i in
range(len(input_block)-4,
len(input_block)):
            row = input_block[i] << 4
            col = input_block[i] & 0x0F
            # print(row, col)
            output_block[i] =
S_BOX[row%4][col%4]
    # print("after", input_block)
    return output_block
```

Fungsi ini berfungsi untuk melakukan substitusi pada blok input yang diberikan dengan menggunakan tabel substitusi (S-box) yang telah ditentukan.

Langkah awal yang dilakukan oleh fungsi adalah mengonversi input_block dari bentuk biner ke bentuk heksadesimal dengan memanggil fungsi binary_to_hex(). Kemudian, fungsi ini membuat sebuah array dua dimensi (S_BOX) yang berisi tabel substitusi yang akan digunakan dalam proses substitusi.

Setelah itu, fungsi akan membuat sebuah array kosong (output_block) yang akan diisi dengan hasil substitusi. Setelah selesai melakukan substitusi, fungsi akan mengembalikan output_block yang berisi hasil substitusi berdasarkan parameter *encrypt* atau *decrypt*.

c. Fungsi Permutasi

```
def permutation(input_block, str):
    PERMUTATION_TABLE = np.array([
        0, 4, 8, 12,
        1, 5, 9, 13,
        2, 6, 10, 14,
        3, 7, 11, 15
    ])
    # print("this is input block",
input_block)
    # print(len(input_block))
    output_block =
np.zeros_like(input_block)
    if str=="encrypt":
        for i in range(16):
            #
print(PERMUTATION_TABLE[i])
            output_block[i] =
input_block[PERMUTATION_TABLE[i]]
    elif str=="decrypt":
        for i in range(15,-1):
            #
print(PERMUTATION_TABLE[i])
            output_block[i] =
input_block[PERMUTATION_TABLE[i]]
    return output_block
```

Fungsi permutation menerima dua parameter yaitu input_block dan str. Fungsi ini berfungsi untuk melakukan permutasi pada blok input yang diberikan dengan menggunakan tabel permutasi (permutation table) yang telah ditentukan.

Langkah awal yang dilakukan oleh fungsi adalah membuat sebuah array satu dimensi (PERMUTATION_TABLE) yang berisi tabel permutasi yang akan digunakan dalam proses permutasi.

Perulangan dilakukan sebanyak 16 kali, dimana nilai i akan diambil dari range 0 hingga 15 untuk encrypt dan 15-0 untuk decrypt. Pada setiap iterasi, output_block akan diisi dengan nilai input_block yang berada pada indeks PERMUTATION_TABLE[i].

Setelah itu, fungsi akan membuat sebuah array kosong (output_block) yang akan diisi dengan hasil permutasi. Setelah selesai melakukan permutasi, fungsi akan mengembalikan output_block yang berisi hasil permutasi.

d. Fungsi Pembangkit Kunci Putaran

```
def generate_round_key(master_key,
round_number):
    # round_key = master_key ^
round_number
    # round_key =
round_key.to_bytes(16,
byteorder='big')
    # round_key =
np.frombuffer(round_key,
dtype=np.uint8)
```

```
    round_number = [int(i) for i in
list('{0:0b}'.format(round_number).zfi
ll(len(master_key)))]

    # print("round number" ,
round_number)
    # print("master key", master_key)
    round_key =
xor_operation(master_key,round_number)
    return round_key
```

Untuk menghasilkan kunci putaran pada setiap putaran dalam algoritma cripopo yang telah dirancang, digunakan fungsi generate_round_key yang membutuhkan dua parameter yaitu master_key dan round_number.

Pertama-tama, round_number diubah menjadi sebuah list biner dengan panjang yang sama dengan panjang master_key. Hal ini dilakukan untuk memastikan bahwa panjang kunci putaran yang dihasilkan selalu sama dan sejajar dengan panjang kunci utama.

Setelah round_number berupa list biner, fungsi xor_operation digunakan untuk melakukan operasi XOR antara master_key dan round_number. Hasil operasi XOR tersebut akan menjadi kunci putaran pada putaran saat ini.

e. Fungsi Putaran

```
def round_function(input_block,
round_key, str_type):
    # print("input block" ,
input_block)
    # print("round key", round_key)
    output_block =
substitution(input_block,str_type)
    # print("hasil subs", output_block)
    output_block =
permutation(output_block, str_type)
    # print("hasil permut",
output_block)
    output_block =
xor_operation(output_block, round_key)
    # print("hasil xor", output_block)
    # print()
    return output_block
```

Program ini merupakan sebuah fungsi yang merepresentasikan fungsi putaran. Fungsi ini memiliki tiga parameter masukan yaitu "input_block" (blok input), "round_key" (kunci putaran), dan "str_type" (tipe string). Fungsi ini akan melakukan beberapa operasi pada blok input untuk menghasilkan blok output.

f. Fungsi Jaringan Feistel

```
def feistel_cipher(input_block,
master_key, num_rounds):
    left_block = input_block[:8]
    right_block = input_block[8:]
    for i in range(num_rounds):
        round_key =
generate_round_key(master_key, i)
        # zfill buat nyamain xor aja
        new_right_block =
xor_operation(left_block,
round_function([int(i) for i in
list(''.join(map(str,
right_block)).zfill(len(round_key)))],
round_key, "encrypt"))
        left_block = right_block
        right_block = new_right_block
    output_block =
np.concatenate((right_block,
left_block))
    return output_block
```

Fungsi feistel_cipher menerima tiga parameter yaitu input_block, master_key, dan num_rounds. Fungsi ini menerapkan algoritma kriptografi Feistel Cipher untuk mengenkripsi atau mendekripsi input_block yang diberikan.

Langkah pertama yang dilakukan oleh fungsi adalah membagi input_block menjadi dua bagian dengan panjang masing-masing 8 bit. Bagian pertama akan disebut sebagai left_block, sedangkan bagian kedua disebut sebagai right_block.

Selanjutnya, fungsi akan melakukan iterasi sebanyak num_rounds. Pada setiap iterasi, fungsi akan memanggil fungsi generate_round_key untuk

menghasilkan kunci ronde yang akan digunakan pada iterasi tersebut.

Setelah itu, fungsi akan melakukan operasi XOR antara left_block dengan output dari fungsi round_function, dengan menggunakan right_block dan round_key sebagai input. Untuk memastikan bahwa panjang output dari fungsi round_function sama dengan panjang round_key, fungsi akan mengisi *leading zeroes* pada right_block jika dibutuhkan.

Hasil XOR tersebut kemudian disimpan pada variabel new_right_block. Left_block akan diisi dengan nilai right_block, sedangkan right_block akan diisi dengan nilai new_right_block yang merupakan hasil operasi XOR.

Setelah selesai melakukan iterasi sebanyak num_rounds, left_block dan right_block akan digabungkan kembali dengan urutan right_block kemudian left_block. Hasil penggabungan tersebut akan dihasilkan sebagai output_block dan dikembalikan oleh fungsi.

Secara keseluruhan, fungsi feistel_cipher akan membagi input_block menjadi dua bagian, melakukan iterasi sebanyak num_rounds, dan pada setiap iterasi melakukan operasi XOR antara left_block dan output dari fungsi round_function menggunakan round_key yang dihasilkan oleh fungsi generate_round_key. Hasil XOR tersebut kemudian digunakan untuk memperbarui nilai left_block dan right_block pada setiap iterasi. Akhirnya, left_block dan right_block akan digabungkan kembali untuk menghasilkan output_block yang merupakan hasil enkripsi atau dekripsi dari input_block.

g. Fungsi Enkripsi

```
def encrypt(input_message,
master_key):
    BLOCK_SIZE = 16
    NUM_ROUNDS = 16
    input_message = [int(x) for x in
input_message]
    master_key = [int(x) for x in
master_key]
    num_blocks = len(input_message) //
BLOCK_SIZE
    # print(len(input_message))
    output_message = []
    for i in range(num_blocks):
        input_block =
np.array(input_message[i*BLOCK_SIZE:(i
+1)*BLOCK_SIZE], dtype=np.uint8)
        output_block =
feistel_cipher(input_block,
master_key, NUM_ROUNDS)
        # print(i, " :" , output_block)
        output_message =
np.concatenate((output_message,
output_block))
    # print("enkripsi biner",
output_message)
    output_message =
output_message.astype(int)
    return output_message.tolist()
```

Program di atas merupakan sebuah fungsi Python yang bernama "encrypt" dengan dua parameter input, yaitu "input_message" dan "master_key". Fungsi ini bertujuan untuk mengenkripsi pesan teks yang diberikan menggunakan algoritma kriptografi Feistel Cipher dengan kunci (key) tertentu.

Pada baris pertama, konstanta BLOCK_SIZE diinisialisasi dengan nilai 16, yang menunjukkan ukuran blok enkripsi yang digunakan oleh algoritma Feistel Cipher. Konstanta NUM_ROUNDS diinisialisasi dengan nilai 16 juga, yang merupakan jumlah putaran (rounds) yang dilakukan oleh algoritma Feistel Cipher.

Enkripsi dilakukan setiap blok pesan secara terpisah melalui iterasi. Di setiap iterasi, blok pesan kemudian dienkripsi menggunakan fungsi "feistel_cipher()", yang merupakan implementasi dari algoritma Feistel Cipher. Hasil enkripsi blok ini kemudian diambil sebagai output_block.

h. Fungsi Dekripsi

```
def decrypt(input_message,
master_key):
    BLOCK_SIZE = 16
    NUM_ROUNDS = 16
    input_message = [int(x) for x in
input_message]
    master_key = [int(x) for x in
master_key]
    num_blocks = len(input_message) //
BLOCK_SIZE
```

```
    output_message = []
    # print("ini oy", input_message)
    for i in range(num_blocks):
        input_block =
np.array(input_message[i*BLOCK_SIZE:(i
+1)*BLOCK_SIZE], dtype=np.uint8)
        left_block = input_block[:8]
        right_block = input_block[8:]
        round_keys =
[generate_round_key(master_key, i) for
i in range(NUM_ROUNDS)]
        for i in range(NUM_ROUNDS-1,
-1, -1):
            new_right_block =
left_block
            left_block =
xor_operation(right_block,
round_function([int(i) for i in
list(''.join(map(str,
left_block)).zfill(len(round_keys[i]))
)], round_keys[i], "decrypt"))
            right_block =
new_right_block
        output_block =
np.concatenate((left_block,
right_block))
        # print("outblock nya",
output_block)
        # print("ini output block",
output_block)
        output_message =
np.concatenate((output_message,
output_block))
        output_message =
output_message.astype(int)
    return output_message.tolist()
```

Fungsi decrypt pada program di atas merupakan fungsi untuk melakukan dekripsi sebuah pesan yang sudah dienkripsi oleh algoritma Cripopo.

Pada fungsi encrypt, round key yang digunakan dihitung pada setiap putaran enkripsi dengan menggunakan fungsi generate_round_key. Sedangkan pada fungsi decrypt, round key dihitung sebelum putaran dekripsi dimulai, kemudian round key tersebut diputar kebalikannya dan digunakan pada setiap putaran dekripsi.

Selain itu, pada fungsi encrypt, dilakukan iterasi sebanyak NUM_ROUNDS putaran enkripsi, sedangkan pada fungsi decrypt, dilakukan iterasi sebanyak NUM_ROUNDS-1 putaran dekripsi, karena pada putaran terakhir dilakukan swap antara left_block dan right_block. Iterasi dilakukan secara terbalik yang bertujuan untuk memutar urutan kunci putaran.

*B. Hasil Percobaan*

*a. Teks pendek*

```
message original: Informatikmantab

message binary:
01001001011011100110011001101111011100100110101011000010111010001101001011010110111010110000101101110011010001100001
01100010
key : This is a key.iy

encrypted message in binary :
01101110010010010111011101100110011011010111001001110100011000010110101101101001011000010110110111010001101110011000010
01100001
encrypted char:  nIofmrtakiamtnba

=======

decrypted message in binary
01101110010010010111011101100110011011010111001001110100011000010110101101101001011000010110110111010001101110011000010
01100001
decrypted char:  nIofmrtakiamtnba
```

Enkripsi belum sempurna serta terdapat kesalahan pada dekripsi.

*b. Teks medium*

```
message original: cobalah ungkapkan hatimu wkwkwwk
key : This is a key.iy

encrypted char:  ocabal hnukgpaak nahitumw wkwkkw

=======

decrypted char:  ocabal hnukgpaak nahitumw wkwkkw
```

Enkripsi belum sempurna serta terdapat kesalahan pada dekripsi.

*c. Teks panjang*

```
message original: jika bisa begitu maka bisa begini informatika informatika jayaaa
key : This is a key.iy

encrypted char:  ijakb si aebigutm ka aibasb geni iniofmrtaki aniofmrtaki aajayaa

=======

decrypted char:  ijakb si aebigutm ka aibasb geni iniofmrtaki aniofmrtaki aajayaa
```

Enkripsi belum sempurna serta terdapat kesalahan pada dekripsi.

*d. Teks very large*

```
message original: pada suatu pagi rina berangkat ke itb berangkat dengan gojek sambil bernyanyi jika bisa begitu maka
bisa begini informatika informatika jaya sip
key : This is a key.iy

encrypted char:  apads auutp ga iiranb renakgtak  eti bebargnak tedgnnag jokes maib lebnrayyn iijakb si aebigutm ka
aibasb geni iniofmrtaki aniofmrtaki aajays pi

=======

decrypted char:  apads auutp ga iiranb renakgtak  eti bebargnak tedgnnag jokes maib lebnrayyn iijakb si aebigutm ka
aibasb geni iniofmrtaki aniofmrtaki aajays pi
```

Enkripsi belum sempurna serta terdapat kesalahan pada dekripsi.

e. Mengganti 1 karakter pada key

```
message original: cobalah ungkapkan hatimu wkwkwwk
key : This is a key.ay

encrypted char:  ocabal hnukgpaak nahitumw wkwkkw

=======

decrypted char:  ocabal hnukgpaak nahitumw wkwkkw
```

Tidak terjadi perubahan signifikan apabila dibandingkan dengan poin *teks medium* dimana pesan original sama namun kenya diganti 1 character.

## IV. KESIMPULAN DAN SARAN

Dari percobaan yang telah dilakukan, dapat disimpulkan bahwa block cipher terbukti bisa dibuat menggunakan *confusion & diffusion, enciphering,* serta ketentuan lainnya yang tertera pada spesifikasi. Namun sayangnya, program tidak berhasil diimplementasi dengan baik. Hal ini menyebabkan *block cipher* yang dibuat program masih belum dapat bekerja dengan baik, terutama pada bagian dekripsi, dimana dekripsi selalu menghasilkan output yang sama dengan input.

Penulis menyarankan agar dapat menyisihkan waktu terlebih dahulu untuk mulai mempelajari dan melakukan riset terlebih dahulu, sehingga pembuatan kode dapat dilakukan dalam waktu yang relatif lebih lama atau cukup. Sehingga permasalahan kurang terimplementasikannya kode dengan baik, *bug* pada program, serta permasalahan lainnya yang muncul pada eksperimen ini dapat teratasi dan tidak terulangi di masa depan.sponsor acknowledgments in the unnumbered footnote on the first page.

## TAUTAN PROGRAM

https://github.com/ibnuhafizzh/IF4020_Tugas2

## REFERENCES


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

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

[3] Claude E. Shannon, "Communication Theory of Secrecy Systems", Bell System Technical Journal, vol. 28-4, pages 656–715, 1949.

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

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/14-Prancangan-block-cipher-2023.pdf Diakses pada 1 Maret 2023.