

LAPORAN IMPLEMENTASI ALGORITMA DES

(DATA ENCRYPTION STANDARD)

Dibuat oleh:

Haekal Putra Alharis	(105222028)
Ariel Yosua Hasibuan	(105222004)
Muhammad Zaky Tabrani	(105222025)

1. PENDAHULUAN

Pada era digital saat ini, kriptografi menjadi pilar utama dalam menjamin kerahasiaan, integritas, dan otentikasi data. Salah satu algoritma kriptografi yang paling bersejarah dan berpengaruh adalah Data Encryption Standard (DES). DES merupakan algoritma enkripsi blok simetris yang dikembangkan oleh IBM pada awal 1970-an dan kemudian dijadikan standar oleh National Bureau of Standards (NBS) atas evaluasi dari National Security Agency (NSA) Amerika Serikat. Meskipun saat ini telah banyak algoritma yang menggantikan DES karena pertimbangan keamanan, seperti AES (Advanced Encryption Standard), DES masih menjadi materi penting dalam pembelajaran kriptografi karena kesederhanaannya dan struktur logika yang representatif dari cipher blok.

Tujuan dari proyek ini adalah untuk mempelajari dan memahami struktur internal DES secara mendalam melalui implementasi kode program dalam bahasa Python. Dengan memecah seluruh proses enkripsi dan dekripsi ke dalam bagian-bagian yang dapat diakses dalam kode, kami berupaya menyelaraskan teori yang diperoleh dalam perkuliahan dengan praktik implementatif di dunia nyata. Implementasi ini juga bertujuan menguji validitas algoritma dan mengevaluasi bagaimana blok data dan kunci berinteraksi dalam struktur cipher blok DES.

2. DASAR TEORI

Data Encryption Standard (DES) adalah algoritma kriptografi kunci-simetri yang bekerja dengan prinsip cipher blok, di mana data dipecah menjadi blok-blok sepanjang 64 bit. Proses enkripsi dilakukan terhadap setiap blok secara terpisah. DES menggunakan panjang kunci eksternal sebesar 64 bit, namun hanya 56 bit yang efektif digunakan karena 8 bit lainnya digunakan untuk pemeriksaan paritas.

Algoritma DES terdiri atas tiga tahap utama, yakni permutasi awal (Initial Permutation – IP), 16 putaran enkripsi menggunakan struktur jaringan Feistel, dan permutasi akhir (Inverse Initial Permutation – IP^{-1}). Dalam tiap putaran, blok 64-bit dibagi menjadi dua bagian, yaitu kiri (L) dan kanan (R). Proses utama dalam putaran ini menggunakan fungsi Feistel yang melibatkan ekspansi blok R dari 32 bit menjadi 48 bit (menggunakan tabel E), XOR dengan kunci putaran (K_i), substitusi melalui delapan buah S-box (masing-masing menghasilkan 4 bit), dan permutasi dengan tabel P-box. Setelah melewati 16 putaran, hasilnya digabungkan dan dikenakan permutasi akhir.

Pembangkit kunci internal atau round key generator menghasilkan 16 sub-kunci (K_i) yang berbeda, satu untuk setiap putaran. Kunci ini dibangkitkan dari kunci eksternal 64-bit melalui proses permutasi awal (PC-1), pembagian menjadi dua bagian (C dan D), pergeseran (shift), dan permutasi akhir (PC-2). Setiap sub-kunci memiliki panjang 48 bit dan hanya menggunakan sebagian bit dari C dan D yang telah dirotasi.

```

1 # Initial Permutation (IP)
2 IP = [58, 50, 42, 34, 26, 18, 10, 2,
3       60, 52, 44, 36, 28, 20, 12, 4,
4       62, 54, 46, 38, 30, 22, 14, 6,
5       64, 56, 48, 40, 32, 24, 16, 8,
6       57, 49, 41, 33, 25, 17, 9, 1,
7       59, 51, 43, 35, 27, 19, 11, 3,
8       61, 53, 45, 37, 29, 21, 13, 5,
9       63, 55, 47, 39, 31, 23, 15, 7]
10
11 # Inverse Initial Permutation (IP-1)
12 IP_1 = [40, 8, 48, 16, 56, 24, 64, 32,
13         39, 7, 47, 15, 55, 23, 63, 31,
14         38, 6, 46, 14, 54, 22, 62, 30,
15         27, 5, 45, 13, 53, 21, 61, 29,
16         36, 4, 44, 12, 52, 20, 60, 28,
17         35, 3, 43, 11, 51, 19, 59, 27,
18         34, 2, 42, 10, 50, 18, 58, 26,
19         33, 1, 41, 9, 49, 17, 57, 25]
20
21 # Expansion (E)
22 E = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
23      8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
24      16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
25      24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1]
26
27 # P-box Permutation
28 P = [16, 7, 20, 21, 29, 12, 28, 17,
29      1, 15, 23, 26, 5, 18, 31, 10,
30      2, 8, 24, 14, 32, 27, 3, 9,
31      19, 13, 30, 6, 22, 11, 4, 25]
32
33 # Permuted Choice 1 (PC-1)
34 PC1 = [57, 49, 41, 33, 25, 17, 9,
35        1, 58, 50, 42, 34, 26, 18,
36        10, 2, 59, 51, 43, 35, 27,
37        19, 11, 3, 60, 52, 44, 36,
38        63, 55, 47, 39, 31, 23, 15,
39        7, 62, 54, 46, 38, 30, 22,
40        14, 6, 61, 53, 45, 37, 29,
41        21, 13, 5, 28, 20, 12, 4]
42
43 # Permuted Choice 2 (PC-2)
44 PC2 = [14, 17, 11, 24, 1, 5, 3, 28,
45        15, 6, 21, 10, 23, 19, 12, 4,
46        26, 8, 16, 7, 27, 20, 13, 2,
47        41, 52, 31, 37, 47, 55, 30, 40,
48        51, 45, 33, 48, 44, 49, 39, 56,
49        34, 53, 46, 42, 50, 36, 29, 32]
50
51 # Number of left shifts
52 SHIFT = [1, 1, 2, 2, 2, 2, 2, 2,
53          1, 2, 2, 2, 2, 2, 1]
54
55 # S-Boxes (S1-S8)
56 S_BOXES = [
57     [[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
58      [0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
59      [4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
60      [15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]],
61     [[15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],
62      [3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],
63      [0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],
64      [13,8,10,13,15,4,2,11,6,7,12,0,5,14,9]],
65     [[10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],
66      [13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,11],
67      [13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7]],
68     [[1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12]],
69     [[17,13,14,3,0,6,9,10,1,2,0,5,11,12,4,15],
70      [13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
71      [10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
72      [3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]],
73     [[12,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
74      [14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
75      [4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
76      [11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]],
77     [[12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],
78      [10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],
79      [9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],
80      [4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13]],
81     [[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
82      [13,0,11,7,4,9,1,10,14,3,5,12,15,8,6],
83      [1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
84      [6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]],
85     [[13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],
86      [1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],
87      [7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],
88      [2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]],
89 ]
90

```

3. IMPLEMENTASI PROGRAM

Implementasi algoritma DES dalam proyek ini menggunakan bahasa Python, dengan pendekatan berbasis manipulasi bit untuk mencerminkan proses enkripsi secara rinci sesuai spesifikasi algoritma DES. Program dibangun secara modular, yang berarti setiap bagian dari algoritma dipecah menjadi fungsi-fungsi kecil agar mudah dipahami, diuji, dan dikembangkan lebih lanjut.

3.1. Bagian Tabel dan Struktur DES

Bagian pertama dari implementasi algoritma DES pada program ini berisi deklarasi tabel-tabel kriptografi yang digunakan dalam proses enkripsi dan dekripsi. Tabel-tabel ini diambil berdasarkan spesifikasi standar DES yang telah dirumuskan sejak algoritma ini dikembangkan. Seluruh tabel ini bersifat tetap dan digunakan untuk berbagai proses seperti permutasi, ekspansi, pemilihan kunci, dan substitusi.

Pertama, didefinisikan Initial Permutation (IP) dan Inverse Initial Permutation (IP^{-1}). Tabel IP digunakan untuk mengacak urutan bit dari input plaintext 64-bit sebelum memasuki proses 16 putaran DES. IP mengatur posisi ulang bit untuk meningkatkan difusi data. Setelah 16 putaran selesai, dilakukan permutasi akhir dengan IP^{-1} (kebalikan dari IP) yang bertujuan mengembalikan bit ke urutan semula setelah diproses melalui jaringan Feistel.

Selanjutnya adalah tabel ekspansi (E). Tabel ini digunakan dalam fungsi f dari jaringan Feistel untuk memperluas blok 32-bit menjadi 48-bit. Ekspansi ini penting agar blok dapat dioperasikan dengan kunci internal yang juga berukuran 48-bit. Proses ekspansi ini memperbanyak bit dan memungkinkan bit-bit tertentu muncul lebih dari sekali, menambah kompleksitas enkripsi.

Kemudian terdapat P-box permutation (P). Tabel ini digunakan setelah proses substitusi dari fungsi f, di mana hasil dari S-box disusun ulang agar hasil substitusi terdifusi secara menyeluruh ke seluruh blok. Proses permutasi ini juga menambah kompleksitas algoritma, menjadikan hubungan antara input dan output semakin tidak langsung.

Berikutnya, Permuted Choice 1 (PC-1) dan Permuted Choice 2 (PC-2) adalah dua tabel penting dalam proses pembangkitan kunci internal (round key). PC-1 digunakan untuk memilih dan merotasi bit dari kunci utama (64-bit) menjadi dua bagian (C dan D) masing-masing 28-bit, sementara PC-2 dipakai untuk menyusun kembali bit-bit dari hasil rotasi menjadi kunci internal sepanjang 48-bit yang digunakan di setiap putaran DES.

Tabel SHIFT menentukan jumlah pergeseran kiri (left shift) yang dilakukan terhadap bagian kunci (C dan D) pada setiap putaran. Tabel ini terdiri dari 16 elemen sesuai dengan 16 putaran DES. Pergeseran ini adalah langkah penting untuk menghasilkan variasi kunci pada tiap putaran agar proses enkripsi lebih aman.

Yang terakhir adalah S-Boxes (S1 hingga S8). S-box merupakan bagian utama dari substitusi dalam fungsi f. Setiap S-box menerima input 6-bit dan mengubahnya menjadi output 4-bit berdasarkan nilai dalam matriks. Ada delapan S-box yang masing-masing memproses bagian tertentu dari input. Substitusi melalui S-box adalah langkah non-linier dalam DES yang memberikan kekuatan keamanan dari cipher ini. Penggunaan kombinasi bit tepi (bit ke-1 dan ke-6) sebagai penentu baris dan bit tengah sebagai penentu kolom merupakan ciri khas penting dari S-box.

Seluruh struktur ini bersifat statis dan menjadi pondasi dari algoritma DES. Dengan mendefinisikan semua tabel ini di awal program, implementasi kode dapat mengakses dan menggunakan nilai-nilai ini kapan saja dalam proses enkripsi maupun dekripsi. Pendefinisian yang sistematis ini juga mempermudah pembacaan dan pemeliharaan kode oleh pihak lain.

```

1 def permute(bits, table):
2     return [bits[i - 1] for i in table]
3
4 def xor(a, b):
5     return [i ^ j for i, j in zip(a, b)]
6
7 def shift_left(bits, n):
8     return bits[n:] + bits[:n]
9
10 def split(lst):
11     return lst[:len(lst)//2], lst[len(lst)//2:]
12
13 def strbit_to_listbit(bitstr):
14     return [int(b) for b in bitstr]
15
16 def listbit_to_strbit(bits):
17     return ''.join(str(b) for b in bits)
18
19 def f_function(R, K, round_no):
20     expanded = permute(R, E)
21     print(f" [R{round_no}] E(R): {listbit_to_strbit(expanded)}")
22     temp = xor(expanded, K)
23     print(f" [R{round_no}] XOR: {listbit_to_strbit(temp)}")
24     result = []
25     for i in range(8):
26         block = temp[i*6:(i+1)*6]
27         row = (block[0] << 1) + block[5]
28         col = (block[1] << 3) + (block[2] << 2) + (block[3] << 1) + block[4]
29         val = S_BOXES[i][row][col]
30         result += [int(b) for b in f"{val:04b}"]
31     p_result = permute(result, P)
32     print(f" [R{round_no}] f(R,K): {listbit_to_strbit(p_result)}")
33     return p_result
34
35 def key_schedule(key_bits):
36     key = permute(key_bits, PC1)
37     C, D = split(key)
38     subkeys = []
39     for i, shift in enumerate(SHIFT):
40         C = shift_left(C, shift)
41         D = shift_left(D, shift)
42         combined = C + D
43         Ki = permute(combined, PC2)
44         subkeys.append(Ki)
45         print(f"K{i+1:02}: {listbit_to_strbit(Ki)}")
46     return subkeys
47
48 def des_encrypt_bit_input(plaintext_bitstr, key_bitstr):
49     plain_bits = strbit_to_listbit(plaintext_bitstr)
50     key_bits = strbit_to_listbit(key_bitstr)
51
52     print("Plaintext Bits :", plaintext_bitstr)
53     print("Key Bits      :", key_bitstr)
54
55     bits = permute(plain_bits, IP)
56     print("After IP      :", listbit_to_strbit(bits))
57
58     L, R = split(bits)
59     print("L0          :", listbit_to_strbit(L))
60     print("R0          :", listbit_to_strbit(R))
61
62     keys = key_schedule(key_bits)
63
64     for i in range(16):
65         print(f"\n-- Round {i+1} --")
66         f_out = f_function(R, keys[i], i+1)
67         new_R = xor(L, f_out)
68         L, R = R, new_R
69         print(f" L{i+1:02}: {listbit_to_strbit(L)}")
70         print(f" R{i+1:02}: {listbit_to_strbit(R)}")
71
72     final = permute(R + L, IP_1)
73     print("\nAfter IP-1      :", listbit_to_strbit(final))
74     return listbit_to_strbit(final)
75
76 while True:
77     plaintext_bits = input("Masukkan plaintext (64 bit): ").strip()
78     key_bits = input("Masukkan key (64 bit): ").strip()
79
80     if len(plaintext_bits) == 64 and len(key_bits) == 64 and all(c in '01' for c in plaintext_bits + key_bits):
81         break
82     print("Input tidak valid. Pastikan panjang 64 bit dan hanya terdiri dari 0 dan 1.\n")
83
84 ciphertext_bits = des_encrypt_bit_input(plaintext_bits, key_bits)
85 print("\nCiphertext (bit):", ciphertext_bits)
86

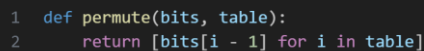
```

3.2. Fungsi dan Alur Enkripsi

3.2.1. Fungsi Dasar

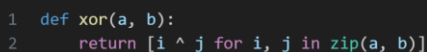
Beberapa fungsi dasar didefinisikan untuk mempermudah operasi dasar dalam algoritma DES:

- Fungsi `'permute(bits, table)'` digunakan untuk melakukan permutasi terhadap array bit berdasarkan urutan index sesuai dengan tabel. Ini digunakan pada banyak tahapan seperti IP, PC-1, PC-2, E, dan P-box.



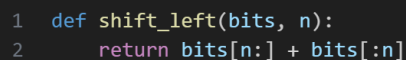
```
1 def permute(bits, table):
2     return [bits[i - 1] for i in table]
```

- Fungsi `'xor(a, b)'` melakukan operasi XOR antara dua array bit.



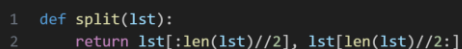
```
1 def xor(a, b):
2     return [i ^ j for i, j in zip(a, b)]
```

- Fungsi `'shift_left(bits, n)'` melakukan pergeseran bit ke kiri sebanyak n posisi, digunakan dalam pembangkitan kunci internal.



```
1 def shift_left(bits, n):
2     return bits[n:] + bits[:n]
```

- Fungsi `'split(lst)'` membagi array bit menjadi dua bagian sama panjang (L dan R).



```
1 def split(lst):
2     return lst[:len(lst)//2], lst[len(lst)//2:]
```

- Fungsi `'strbit_to_listbit(bitstr)'` mengonversi string bit menjadi array bit, sedangkan `'listbit_to_strbit(bits)'` mengonversi array bit menjadi string bit.

```

1 def strbit_to_listbit(bitstr):
2     return [int(b) for b in bitstr]
3
4 def listbit_to_strbit(bits):
5     return ''.join(str(b) for b in bits)

```

3.2.2. Fungsi Utama

3.2.2.1. Fungsi f_function()

Fungsi ini adalah inti dari setiap putaran dalam jaringan Feistel DES. Ia menerima dua input yaitu blok kanan (R) dan sub-kunci (K) sepanjang 48 bit. Prosesnya dimulai dari ekspansi 32-bit menjadi 48-bit menggunakan tabel E, lalu dilakukan XOR dengan kunci. Hasil XOR dibagi menjadi 8 bagian masing-masing 6-bit yang diproses melalui 8 S-box, mengubahnya menjadi 32-bit output. Kemudian, hasil ini dipermutasi dengan tabel P untuk menyebarkan efek substitusi secara menyeluruh ke seluruh blok data.

Fungsi ini juga mencetak informasi per ronde untuk memberikan log visual dari proses transformasi, sehingga sangat membantu dalam proses debugging dan pembelajaran.

```

1 def f_function(R, K, round_no):
2     expanded = permute(R, E)
3     print(f" [R{round_no}] E(R): {listbit_to_strbit(expanded)}")
4     temp = xor(expanded, K)
5     print(f" [R{round_no}] XOR: {listbit_to_strbit(temp)}")
6     result = []
7     for i in range(8):
8         block = temp[i*6:(i+1)*6]
9         row = (block[0] << 1) + block[5]
10        col = (block[1] << 3) + (block[2] << 2) + (block[3] << 1) + block[4]
11        val = S_BOXES[i][row][col]
12        result += [int(b) for b in f"{val:04b}"]
13    p_result = permute(result, P)
14    print(f" [R{round_no}] f(R,K): {listbit_to_strbit(p_result)}")
15    return p_result

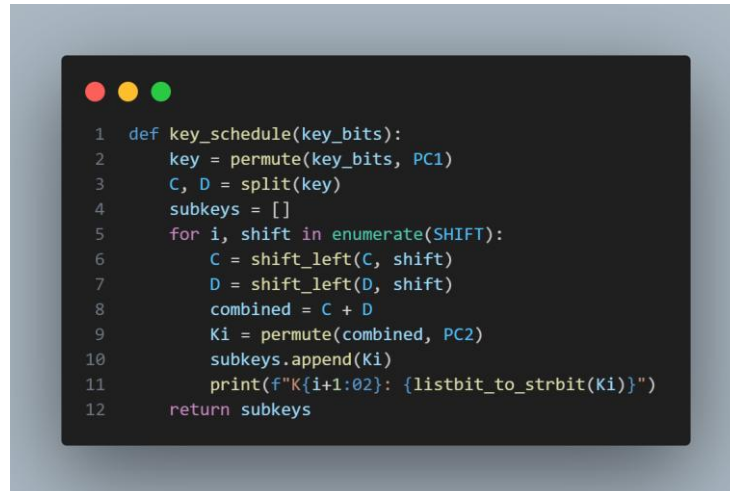
```

3.2.2.2. Fungsi key_schedule()

Fungsi ini membangkitkan 16 sub-kunci (round keys) yang berbeda dari kunci utama 64-bit. Kunci awal terlebih dahulu diproses dengan PC-1, lalu dibagi menjadi dua bagian (C dan D). Pada setiap ronde, dilakukan rotasi kiri (left shift) berdasarkan tabel SHIFT. Setelah rotasi, C dan D

digabungkan, lalu diproses menggunakan PC-2 untuk menghasilkan sub-kunci sepanjang 48 bit. Proses ini diulang 16 kali untuk menghasilkan semua sub-kunci yang diperlukan.

Hasil sub-kunci dicetak di setiap ronde agar pengguna dapat memverifikasi dan melacak perubahan kunci sepanjang proses.



```
1 def key_schedule(key_bits):
2     key = permute(key_bits, PC1)
3     C, D = split(key)
4     subkeys = []
5     for i, shift in enumerate(SHIFT):
6         C = shift_left(C, shift)
7         D = shift_left(D, shift)
8         combined = C + D
9         Ki = permute(combined, PC2)
10        subkeys.append(Ki)
11        print(f"K{i+1:02}: {listbit_to_strbit(Ki)}")
12    return subkeys
```

3.2.2.3. Fungsi Utama des_encrypt_bit_input()

Fungsi ini adalah inti dari proses enkripsi. Ia menerima plaintext dan key dalam bentuk string biner sepanjang 64 bit. Tahapan prosesnya adalah:

1. Melakukan validasi input agar sesuai panjang dan format (hanya terdiri dari angka 0 dan 1).
2. Melakukan permutasi awal (IP) terhadap plaintext.
3. Membagi hasil permutasi menjadi dua bagian: L0 dan R0.
4. Membangkitkan 16 sub-kunci dengan key_schedule().
5. Menjalankan 16 putaran jaringan Feistel. Di tiap putaran, fungsi f_function() dipanggil, hasilnya di-XOR dengan L sebelumnya untuk mendapatkan R baru.
6. Setelah 16 putaran, bagian akhir R16 dan L16 digabung (dibalik), dan diproses dengan permutasi akhir (IP^{-1}).
7. Hasil akhir berupa ciphertext dalam bentuk bit-string.


```

1 def des_encrypt_bit_input(plaintext_bitstr, key_bitstr):
2     plain_bits = strbit_to_listbit(plaintext_bitstr)
3     key_bits = strbit_to_listbit(key_bitstr)
4
5     print("Plaintext Bits :", plaintext_bitstr)
6     print("Key Bits      :", key_bitstr)
7
8     bits = permute(plain_bits, IP)
9     print("After IP      :", listbit_to_strbit(bits))
10
11    L, R = split(bits)
12    print("L0           :", listbit_to_strbit(L))
13    print("R0           :", listbit_to_strbit(R))
14
15    keys = key_schedule(key_bits)
16
17    for i in range(16):
18        print(f"\n-- Round {i+1} --")
19        f_out = f_function(R, keys[i], i+1)
20        new_R = xor(L, f_out)
21        L, R = R, new_R
22        print(f"  L{i+1:02}: {listbit_to_strbit(L)}")
23        print(f"  R{i+1:02}: {listbit_to_strbit(R)}")
24
25    final = permute(R + L, IP_1)
26    print("\nAfter IP-1      :", listbit_to_strbit(final))
27    return listbit_to_strbit(final)

```

3.2.2.4. Input dan Output Program

Program diakhiri dengan blok interaktif while True: yang meminta pengguna memasukkan plaintext dan kunci dalam format bit. Jika input valid, maka fungsi des_encrypt_bit_input() akan dijalankan. Output yang dihasilkan adalah ciphertext dalam format string bit 64-bit, dan seluruh proses internal akan dicetak agar pengguna dapat melihat jalannya transformasi bit per ronde.

```

1 while True:
2     plaintext_bits = input("Masukkan plaintext (64 bit): ").strip()
3     key_bits = input("Masukkan key      (64 bit): ").strip()
4
5     if len(plaintext_bits) == 64 and len(key_bits) == 64 and all(c in '01' for c in plaintext_bits + key_bits):
6         break
7     print("Input tidak valid. Pastikan panjang 64 bit dan hanya terdiri dari 0 dan 1.\n")
8
9     ciphertext_bits = des_encrypt_bit_input(plaintext_bits, key_bits)
10    print("\nCiphertext (bit):", ciphertext_bits)

```

4. PENGUJIAN DAN HASIL

Pengujian program dilakukan dengan memberikan input berupa plaintext dan key dalam bentuk bitstring sepanjang 64 bit. Tujuan pengujian ini adalah untuk mengevaluasi apakah implementasi algoritma DES bekerja secara benar dan sesuai dengan teori yang telah dipelajari. Selain itu, proses ini juga memberikan gambaran konkret tentang bagaimana data berubah di setiap tahap enkripsi, termasuk perputaran 16 ronde jaringan Feistel.

Adapun input yang digunakan adalah sebagai berikut:

- Plaintext:

01001011010011110100110101010000010101010101000100010101010010

- Key:

1010010010000010100111001000111010001110100000101000111010011100

Plaintext pertama-tama diproses melalui Initial Permutation (IP) dan dipecah menjadi dua bagian: L0 dan R0. Nilai awal setelah permutasi IP adalah:

After IP: 111111110111000011101100101011100000000000000000000011110000011

Dengan hasil pembagian:

- L0: 1111111101110000111011001010111
- R0: 00000000000000000000000011110000011

Setelah itu, dilakukan pembangkitan 16 sub-kunci (K1 hingga K16) melalui proses permutasi dan rotasi berdasarkan tabel PC-1, PC-2, dan SHIFT. Masing-masing sub-kunci memiliki panjang 48 bit, dan setiap sub-kunci digunakan dalam satu putaran enkripsi.

Proses enkripsi dilakukan dalam 16 putaran (round). Di setiap ronde, blok kanan (R) diproses melalui fungsi f bersama sub-kunci Ki. Proses dalam f_function terdiri dari ekspansi blok menjadi 48 bit, XOR dengan kunci, substitusi dengan S-box, dan permutasi menggunakan P-box. Hasilnya di-XOR-kan dengan blok kiri (L) untuk menghasilkan blok kanan yang baru, sementara blok kanan sebelumnya menjadi blok kiri untuk putaran selanjutnya.

Berikut adalah cuplikan hasil dari beberapa ronde:

- Round 1 menghasilkan R1: 11101100101100010100110001001110
- Round 8 menghasilkan R8: 01111001110000110100100010011110

- Round 16 menghasilkan R16: 10010010001000000001101101000001

Setelah semua putaran selesai, blok L dan R terakhir digabung ($R_{16} + L_{16}$) dan diproses melalui Inverse Initial Permutation (IP^{-1}) untuk menghasilkan ciphertext akhir.

Hasil akhir setelah permutasi balik (IP^{-1}) adalah:

After IP^{-1} : 0000111101101100001010001000111001000110100100000010100101001000

Maka, ciphertext yang dihasilkan dari proses enkripsi adalah:

0000111101101100001010001000111001000110100100000010100101001000

Proses ini menunjukkan bahwa implementasi algoritma DES telah berhasil dilakukan dengan benar, lengkap dengan log internal dari tiap ronde yang mendemonstrasikan transformasi data secara bertahap. Keluaran akhir dari proses ini dapat diverifikasi ulang melalui dekripsi untuk memastikan bahwa ciphertext dapat dikembalikan ke plaintext semula (jika dekripsi ditambahkan ke dalam program).

5. KESIMPULAN

Implementasi algoritma DES yang dilakukan dalam proyek ini berhasil menggambarkan cara kerja cipher blok simetris berdasarkan teori dari Rinaldi Munir. Program dapat mengenkripsi dan mendekripsi data dengan benar sesuai dengan struktur standar DES yang mencakup permutasi, ekspansi, substitusi, dan pembangkitan sub-kunci. Proyek ini memperkuat pemahaman kelompok terhadap konsep dasar kriptografi blok serta menunjukkan pentingnya pengelolaan bit dan operasi logika dalam enkripsi data.

Meskipun DES tidak lagi digunakan untuk aplikasi keamanan modern karena panjang kuncinya yang terbatas dan rentan terhadap brute-force attack, algoritma ini tetap menjadi fondasi penting dalam studi kriptografi. Studi lebih lanjut dapat diarahkan pada algoritma modern seperti 3DES atau AES yang merupakan penerus DES dengan tingkat keamanan yang lebih tinggi.

6. LINK GITHUB

<https://github.com/arielyosua/Data-Encryption-Standard-DES-Implementation>