TUGAS KRIPTOGRAFI

ENKRIPSI DENGAN ALGORITMA DES

DISUSUN OLEH:

NI KADEK EVI DIANASARI (2008561021)

DOSEN PENGAMPU:

Ida Ayu Gde Suwiprabayanti Putra, S.Kom., M.T.

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS UDAYANA

2023

Tugas Kriptografi – Pertemuan 6,7

Lakukan Enkripsi dengan algoritma DES (Data Encryption Standard) dengan ketentuan sebagai berikut :
1. Plaintext : DOMISILI
2. Kunci : CAPSLOCK

Bentuk Pengumpulan :
1. Dikumpulkan dalam bentuk PDF
2. Alur yang dilakukan adalah perubahan dari Plaintext menjadi Ciphertext
3. Terdapat penjelasan langkah per langkah perubahan/perhitungan

Langkah 1

Mengubah plaintext dan kunci menjadi bilangan biner.

1. TO HEXA
   - DOMISILI = 44 4f 4d 49 53 49 4c 49
   - CAPSLOCK = 43 41 50 53 4c 4f 43 4b
2. TO BINARY
   - DOMISILI = 01000100 01001111 01001101 01001001 01010011 01001001 01001100 01001001
   - CAPSLOCK = 01000011 01000001 01010000 01010011 01001100 01001111 01000011 01001011

PLAIN	HEXA	BINER
D	44	01000100
O	4f	01001111
M	4d	01001101
I	49	01001001
S	53	01010011
I	49	01001001
L	4c	01001100
I	49	01001001

KUNCI	HEXA	BINER
C	43	01000011
A	41	01000001
P	50	01010000
S	53	01010011
L	4c	01001100
O	4f	01001111
C	43	01000011
K	4b	01001011

Langkah 2

Initial Permutation (IP) pada plaintext.

Plaintext (X)

1	2	3	4	5	6	7	8
0	1	0	0	0	1	0	0

9	10	11	12	13	14	15	16
0	1	0	0	1	1	1	1

17	18	19	20	21	22	23	24
0	1	0	0	1	1	0	1

25	26	27	28	29	30	31	32
0	1	0	0	1	0	0	1

33	34	35	36	37	38	39	40
0	1	0	1	0	0	1	1

41	42	43	44	45	46	47	48
0	1	0	0	1	0	0	1

49	50	51	52	53	54	55	56
0	1	0	0	1	1	0	0

57	58	59	60	61	62	63	64
0	1	0	0	1	0	0	1

Tabel IP

1	2	3	4	5	6	7	8
58	50	42	34	26	18	10	2

9	10	11	12	13	14	15	16
60	52	44	36	28	20	12	4

17	18	19	20	21	22	23	24
62	54	46	38	30	22	14	6

25	26	27	28	29	30	31	32
64	56	48	40	32	24	16	8

33	34	35	36	37	38	39	40
57	49	41	33	25	17	9	1

41	42	43	44	45	46	47	48
59	51	43	35	27	19	11	3

49	50	51	52	53	54	55	56
61	53	45	37	29	21	13	5

57	58	59	60	61	62	63	64
63	55	47	39	31	23	15	7

IP(X)

1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1

9	10	11	12	13	14	15	16
0	0	0	1	0	0	0	0

17	18	19	20	21	22	23	24
0	1	0	0	0	1	1	1

25	26	27	28	29	30	31	32
1	0	1	1	1	1	1	0

33	34	35	36	37	38	39	40
0	0	0	0	0	0	0	0

41	42	43	44	45	46	47	48
0	0	0	0	0	0	0	0

49	50	51	52	53	54	55	56
1	1	1	0	1	1	1	0

57	58	59	60	61	62	63	64
0	0	0	1	0	0	1	0

Hasil: IP(X) = 11111111 00010000 01000111 10111110 00000000 00000000 11101110 00010010 Selanjutnya bit pada IP(X) dipecah menjadi 2 :

L_0 : 11111111 00010000 01000111 10111110
R_0 : 00000000 00000000 11101110 00010010

Hasil IP(X) juga bisa didapat dengan code berikut :

```
# Tabel IP
ip_table = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]

# Fungsi untuk melakukan permutasi IP pada plaintext
def initial_permutation(plaintext64):
    ip = ""
    for index in ip_table:
        ip += plaintext64[index - 1]
    return ip

# Masukkan Plaintext
plaintext64 =
"0100010001001111010011010100100101010011010010010100110001001001"
ip_plaintext = initial_permutation(plaintext64)
print("IP(plaintext) =", ip_plaintext)
```

Langkah 3

Generate Kunci menggunakan tabel permutasi kompresi PC-1 Kompresi 64 bit menjadi 56 bit dengan membuang 1 bit (parity bit) tiap blok kunci.

Kunci

1	2	3	4	5	6	7	8
0	1	0	0	0	0	1	1

9	10	11	12	13	14	15	16
0	1	0	0	0	0	0	1

17	18	19	20	21	22	23	24
0	1	0	1	0	0	0	0

25	26	27	28	29	30	31	32
0	1	0	1	0	0	1	1

33	34	35	36	37	38	39	40
0	1	0	0	1	1	0	0

41	42	43	44	45	46	47	48
0	1	0	0	1	1	1	1

49	50	51	52	53	54	55	56
0	1	0	0	0	0	1	1

57	58	59	60	61	62	63	64
0	1	0	0	1	0	1	1

Tabel PC-1

1	2	3	4	5	6	7
57	49	41	33	25	17	9

8	9	10	11	12	13	14
1	58	50	42	34	26	18

15	16	17	18	19	20	21
10	2	59	51	43	35	27

22	23	24	25	26	27	28
19	11	3	60	52	44	36

29	30	31	32	33	34	35
63	55	47	39	31	23	15

36	37	38	39	40	41	42
7	62	54	46	38	30	22

43	44	45	46	47	48	49
14	6	61	53	45	37	29

50	51	52	53	54	55	56
21	13	5	28	20	12	4

OUTPUT

1	2	3	4	5	6	7
0	0	0	0	0	0	0

8	9	10	11	12	13	14
0	1	1	1	1	1	1

15	16	17	18	19	20	21
1	1	0	0	0	0	0

22	23	24	25	26	27	28
0	0	0	0	0	0	0

29	30	31	32	33	34	35
1	1	1	0	1	0	0

36	37	38	39	40	41	42
1	0	0	1	1	0	0

43	44	45	46	47	48	49
0	0	1	0	1	1	0

50	51	52	53	54	55	56
0	0	0	1	1	0	0

Hasil: C_0D_0 = 0000000 0111111 1100000 0000000 1110100 1001100 0010110 0001100
Selanjutnya bit pada C0D0 dipecah menjadi 2 :

C_0 : 0000000 0111111 1100000 0000000
D_0 : 1110100 1001100 0010110 0001100

Hasil: C_0D_0 juga bisa di dapat dengan code berikut :

```
# Tabel PC-1
pc1_table = [
    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
]

# Fungsi untuk melakukan permutasi PC-1
def permute_pc1(key64):
    c0d0 = ""
    for index in pc1_table:
        c0d0 += key64[index - 1]
    return c0d0

# masukkan kunci awal
key64 = "0100001101000001010100000101001101001100010011110100001101001011"
c0d0 = permute_pc1(key64)
print("C0D0 =", c0d0)
```

Langkah 4

Left Shift Operztion Lakukan pergeseran pada C_0 dan D_0 menggunakan tabel pergeseran bit 16 putaran.

Iterasi	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Perputaran bit	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

C_0D_0 = 0000000 0111111 1100000 0000000 1110100 1001100 0010110 0001100

C_0 : 0000000 0111111 1100000 0000000	D_0 : 1110100 1001100 0010110 0001100
C_1 : 0000000 1111111 1000000 0000000	D_1 : 1101001 0011000 0101100 0011001
C_2 : 0000001 1111111 0000000 0000000	D_2 : 1010010 0110000 1011000 0110011
C_3 : 0000111 1111100 0000000 0000000	D_3 : 1001001 1000010 1100001 1001110
C_4 : 0011111 1110000 0000000 0000000	D_4 : 0100110 0001011 0000110 0111010
C_5 : 1111111 1000000 0000000 0000000	D_5 : 0011000 0101100 0011001 1101001
C_6 : 1111110 0000000 0000000 0000011	D_6 : 1100001 0110000 1100111 0100100
C_7 : 1111000 0000000 0000000 0001111	D_7 : 0000101 1000011 0011101 0010011
C_8 : 1100000 0000000 0000000 0111111	D_8 : 0010110 0001100 1110100 1001100
C_9 : 1000000 0000000 0000000 1111111	D_9 : 0101100 0011001 1101001 0011000
C_{10} : 0000000 0000000 0000011 1111110	D_{10} : 0110000 1100111 0100100 1100001
C_{11} : 0000000 0000000 0001111 1111000	D_{11} : 1000011 0011101 0010011 0000101
C_{12} : 0000000 0000000 0111111 1100000	D_{12} : 0001100 1110100 1001100 0010110
C_{13} : 0000000 0000001 1111111 0000000	D_{13} : 0110011 1010010 0110000 1011000
C_{14} : 0000000 0000111 1111100 0000000	D_{14} : 1001110 1001001 1000010 1100001
C_{15} : 0000000 0011111 1110000 0000000	D_{15} : 0111010 0100110 0001011 0000110
C_{16} : 0000000 0111111 1100000 0000000	D_{16} : 1110100 1001100 0010110 0001100

Hasil: C_iD_i juga bisa di dapat dengan code berikut :

```python
# Tabel pergeseran bit 16 putaran
shift_table = [
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
]

# Fungsi untuk melakukan Left Shift pada C0 dan D0 untuk 16 putaran
def left_shift_16_rounds(c0d0):
    c0d0_shifted = c0d0
    c0, d0 = c0d0[:28], c0d0[28:]

    for round_number in range(1, 17):
        shift_amount = shift_table[round_number - 1]

        # Pergeseran bit sesuai dengan shift_amount
        c0 = c0[shift_amount:] + c0[:shift_amount]
        d0 = d0[shift_amount:] + d0[:shift_amount]

        c0d0_shifted = c0 + d0
        print(f"C0D0 Shifted (Round {round_number}) =", c0d0_shifted)

    return c0d0_shifted

# Contoh penggunaan
c0d0 = "00000000111111110000000000001110100100110000101100001100"
final_c0d0 = left_shift_16_rounds(c0d0)
```

Setiap hasil putaran digabungkan kembali menjadi CiDi dan diinput kedalam tabel Permutation Compression 2 (PC-2) dan terjadi kompresi data CiDi 56 bit menjadi CiDi 48 bit.

Tabel PC-2

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

Berikut hasil outputnya :

C_1D_1 = 0000000 1111111 1000000 0000000 1101001 0011000 0101100 0011001
K_1 = 101000 001001 001001 000010 010010 100000 101101 100011

C_2D_2 = 0000001 1111111 0000000 0000000 1010010 0110000 1011000 0110011
K_2 = 101000 000001 001001 010010 011101 001100 000110 100010

C_3D_3 = 0000111 1111100 0000000 0000000 1001001 1000010 1100001 1001110
K_3 = 001001 000101 001001 010000 100001 000000 110001 001111

C_4D_4 = 0011111 1110000 0000000 0000000 0100110 0001011 0000110 0111010
K_4 = 000001 100101 000101 010000 110011 101011 001011 010000

C_5D_5 = 1111111 1000000 0000000 0000000 0011000 0101100 0011001 1101001
K_5 = 000011 100100 000101 010001 001100 011100 011101 101001

C_6D_6 = 1111110 0000000 0000000 0000011 1100001 0110000 1100111 0100100
K_6 = 000011 110100 000100 001001 000110 101001 110000 000010

C_7D_7 = 1111000 0000000 0000000 0001111 0000101 1000011 0011101 0010011
K_7 = 000010 110000 000110 001001 110011 000110 010100 110100

C_8D_8 = 1100000 0000000 0000000 0111111 0010110 0001100 1110100 1001100
K_8 = 000110 010000 100010 001001 001010 010110 101011 001000

C_9D_9 = 1000000 0000000 0000000 1111111 0101100 0011001 1101001 0011000
K_9 = 000110 010000 100010 001000 010000 100010 111001 110001

$C_{10}D_{10}$ = 0000000 0000000 0000011 1111110 0110000 1100111 0100100 1100001
K_{10} = 000100 000010 100010 001100 101110 111000 100100 011100

$C_{11}D_{11}$ = 0000000 0000000 0001111 1111000 1000011 0011101 0010011 0000101
K_{11} = 000100 000010 110000 000100 000000 010101 011110 010010

$C_{12}D_{12}$ = 0000000 0000000 0111111 1100000 0001100 1110100 1001100 0010110
K_{12} = 010000 000010 110000 100100 010111 010010 000000 100101

$C_{13}D_{13}$ = 0000000 0000001 1111111 0000000 0110011 1010010 0110000 1011000
K_{13} = 110000 001010 010000 100100 111000 100100 100011 001100

$C_{14}D_{14}$ = 0000000 0000111 1111100 0000000 1001110 1001001 1000010 1100001
K_{14} = 110000 001000 011000 100010 000000 001011 001110 011111

$C_{15}D_{15}$ = 0000000 0011111 1110000 0000000 0111010 0100110 0001011 0000110
K_{15} = 111000 001001 001000 100010 101101 110001 010010 100001

$C_{16}D_{16}$ = 0000000 0111111 1100000 0000000 1110100 1001100 0010110 0001100
K_{16} = 101000 001001 001000 100010 001010 110111 001001 000110

Hasil: K_i juga bisa di dapat dengan code berikut :

```
# Tabel PC-2
pc2_table = [
    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
]

# Fungsi untuk melakukan permutasi PC-2
def permute_pc2(cndn):
    k = ""
    for index in pc2_table:
        k += cndn[index - 1]
    return k

# Contoh penggunaan untuk C1D1 (Input C1D1-C16D16)
c1d1 = "000000011111111000000000000011010010011000010110000011001"
k1 = permute_pc2(c1d1)
print("K1 =", k1)
```

Langkah 5

Pada langkah ini, kita akan meng-ekspansi data Ri-1 32 bit menjadi Ri 48 Bit sebanyak 16 kali putaran dengan nilai perputaran 1<= i <= 16 menggunakan tabel ekspansi (E).

Tabel Ekspansi(E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Hasil E(Ri-1) kemudian di XOR dengan Ki dan menghasilkan Vektor Matris Ai

Berikut hasil outputnya :

- Iterasi 1
  $E(R_0)$ = 000000 000000 000000 000001 011101 011100 000010 100100
  K_1 = 101000 001001 001001 000010 010010 100000 101101 100011
  A_1 = 101000 001001 001001 000011 001111 111100 101111 000111

- Iterasi 2
  $E(R_1)$ = 010000 000011 110101 010001 010100 000111 111101 010001
  K_2 = 101000 000001 001001 010010 011101 001100 000110 100010
  A_2 = 111000 000010 111100 000011 001001 001011 111011 110011

- Iterasi 3
  E(R2) = 000011 110000 000000 001011 110101 010010 101111 111000
  K_3 = 001001 000101 001001 010000 100001 000000 110001 001111
  A_3 = 001010 110101 001001 011011 010100 010010 011110 110111

- Iterasi 4
  $E(R_3)$ = 011110 101110 101101 010000 001000 000010 100110 101101
  K_4 = 000001 100101 000101 010000 110011 101011 001011 010000
  A_4 = 011111 001011 101000 000000 111011 101001 101101 111101

- Iterasi 5
  $E(R_4)$ = 111001 010001 011000 000111 110011 111000 001011 111011
  K_5 = 000011 100100 000101 010001 001100 011100 011101 101001
  A_5 = 111010 110101 011101 010110 111111 100100 010110 010010

- Iterasi 6
  $E(R_5)$ = 000001 010011 111011 110010 100111 111101 010101 010100
  K_6 = 000011 110100 000100 001001 000110 101001 110000 000010
  A_6 = 000010 100111 111111 111011 100001 010100 100101 010110

- Iterasi 7
  $E(R_6)$ = 001100 000111 110110 100000 001101 011011 111111 111000
  K_7 = 000010 110000 000110 001001 110011 000110 010100 110100
  A_7 = 001110 110111 110000 101001 111110 011101 101011 001100

- Iterasi 8
  $E(R_7)$ = 000000 001000 000100 001110 100010 101110 101111 111100
  K_8 = 000110 010000 100010 001001 001010 010110 101011 001000
  A_8 = 000110 011000 100110 000111 101000 111000 000100 110100

- Iterasi 9
  $E(R_8)$ = 011101 011101 010111 110100 001011 110000 001101 011101
  K_9 = 000110 010000 100010 001000 010000 100010 111001 110001
  A_9 = 011011 001101 110101 111100 011011 010010 110100 101100

- Iterasi 10
  $E(R_9)$ = 000111 111010 100111 111001 010110 100100 000010 100000
  K_{10} = 000100 000010 100010 001100 101110 111000 100100 011100
  A_{10} = 000011 111000 000101 110101 111000 011100 100110 111100

- Iterasi 11
  $E(R_{10})$ = 101101 011000 000100 001101 010100 000101 011000 001010
  K_{11} = 000100 000010 110000 000100 000000 010101 011110 010010
  A_{11} = 101001 011010 110100 001001 010100 010000 000110 011000

- Iterasi 12
  $E(R_{11})$ = 100011 111010 101101 011000 000101 011101 010110 101010
  K_{12} = 010000 000010 110000 100100 010111 010010 000000 100101
  A_{12} = 110011 111000 011101 111100 010010 001111 010110 001111

- Iterasi 13
  $E(R_{12})$ = 101001 010101 010111 110111 111000 001011 110101 010110
  K_{13} = 110000 001010 010000 100100 111000 100100 100011 001100
  A_{13} = 011001 011111 000111 010011 000000 101111 010110 011010

- Iterasi 14
  $E(R_{13})$ = 110001 010111 110011 111000 001111 110110 100100 001111
  K_{14} = 110000 001000 011000 100010 000000 001011 001110 011111
  A_{14} = 000001 011111 101011 011010 001111 111101 101010 010000

- Iterasi 15
  $E(R_{14})$ = 101110 101001 010111 110011 110010 100001 011111 110110
  K_{15} = 111000 001001 001000 100010 101101 110001 010010 100001
  A_{15} = 010110 100000 011111 010001 011111 010000 001101 010111

- Iterasi 16
  $E(R_{15})$ = 110000 001010 100011 110101 011100 000111 110100 001111
  K_{16} = 101000 001001 001000 100010 001010 110111 001001 000110
  A_{16} = 011000 000011 101011 010111 010110 110000 111101 001001

Hasil E(Ri-1) dan Vektor Matris Ai bisa dihasilkan juga dengan code berikut :

```
# Tabel Ekspansi
expansion_table = [
    32,  1,  2,  3,  4,  5,  4,  5,
     6,  7,  8,  9,  8,  9, 10, 11,
    12, 13, 12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21, 20, 21,
    22, 23, 24, 25, 24, 25, 26, 27,
    28, 29, 28, 29, 30, 31, 32,  1
]

# Fungsi untuk melakukan Ekspansi
def expansion(data32):
    expanded_data = ""
    for index in expansion_table:
        expanded_data += data32[index - 1]
    return expanded_data

# Fungsi untuk melakukan XOR antara dua bitstring
def xor(bitstring1, bitstring2):
    result = ""
    for b1, b2 in zip(bitstring1, bitstring2):
        result += '1' if b1 != b2 else '0'
    return result

# Input Data R1-R16
Ri = "00000000000000001110111000010010"

# Input Kunci K1-K16
keys = "101000001001001001000010010010100000101101100011"

# Lakukan Ekspansi pada Ri
expanded_R = expansion(Ri)


# XOR dengan kunci Ki
Ai = xor(expanded_R, keys)


# Tampilkan hasil pada setiap iterasi
print(f"Ekspansi dari Ri = {expanded_R}")
print(f"Vektor Matriks Ai = {Ai}")
print()
```

Langkah 6

Setiap vektor Ai disubstitusikan ke 8 buah S-box (substitution box), dimana blok ke 1 disubstisusikan ke S1, blok ke-2 disubstitusikan ke S2, dst-nya, menghasilkan output vektor Bi 32 bit.

S1 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
01	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
10	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
11	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
01	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
10	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
11	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
01	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
11	3	15	0	6	10	1	13	18	9	4	5	11	12	7	2	14

S5 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
01	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	16
10	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
01	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
10	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
11	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
01	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
10	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
11	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
01	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
10	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
11	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Kita ambil Contoh S1, kemudian konversi setiap angka di dalam tabel S1 yang berwarna putih menjadi biner, sehingga menjadi bentuk seperti dibawah :

S1 :

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

Kemudian kita ambil sampel blok bit pertama dari A_1 yaitu **1**0100**0**
Kita pisahkan blok menjadi 2 yaitu :

- Bit pertama dan terakhir yaitu 1 dan 0 digabungkan menjadi 10
- Bit kedua hingga ke lima 0100

Kemudian dibandingkan dengan memeriksa perpotongan antara keduanya didapatkan nilai 1101 (warna merah) dan seterusnya untuk blok kedua hingga blok kedelapan kita bandingkan denganS2 hingga S8.

B_1 = 1101 1111 0011 1000 0001 1011 0111 1000
B_2 = 0011 0001 1110 1000 0100 1100 0010 1100
B_3 = 1111 0111 0011 1010 0011 1101 0001 0000
B_4 = 1000 0010 1000 0111 0100 1001 1010 0110
B_5 = 1010 0111 1111 0101 0011 1111 0111 1001
B_6 = 0100 0001 1100 0111 1011 0011 1101 1110
B_7 = 1000 1100 1011 1010 1110 0011 0100 1011
B_8 = 0001 1100 1001 0101 1010 0001 0010 1010
B_9 = 0101 1000 1110 1000 1001 1101 0110 1110
B_{10} = 1111 1001 0000 0101 0110 0101 1101 0101
B_{11} = 0100 0000 0010 0110 0011 0000 1110 0101
B_{12} = 1011 1001 1111 1000 0101 0101 0111 0100
B_{13} = 1001 0101 1001 0111 0010 1010 0111 0000
B_{14} = 0000 0101 1001 1100 0001 1000 0011 1010
B_{15} = 1100 0000 0001 0100 0110 0000 0001 1011
B_{16} = 0101 1101 1001 1100 1111 0111 0011 1010

Hasil: B_i juga bisa di dapat dengan code berikut :

```
# Tabel S-boxes S1 hingga S8 dalam DES
s_boxes = [
    # S1
    [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    # S2
    [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    # S3
    [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    # S4
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
    ],
    # S5
    [
        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
    ],
    # S6
    [
        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
```

```
    ],
    # S7
    [
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
    ],
    # S8
    [
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
]

# Input vektor Ai A1-A16
Ai = "101000001001001001000011001111111100101111000111"

# Bagi vektor Ai menjadi 8 blok 6-bit
blocks = [Ai[i:i+6] for i in range(0, len(Ai), 6)]

# Inisialisasi output vektor Bi
Bi = ""

# Proses substitusi S-box untuk setiap blok
for i, block in enumerate(blocks):
    s_box = s_boxes[i]
    row = int(block[0] + block[5], 2)
    col = int(block[1:5], 2)
    output = s_box[row][col]
    Bi += format(output, '04b')

# Output vektor Bi
print("Vektor Bi:", Bi)
```

Langkah 7

Setelah didapat vektor Bi, lalu permutasikan bit vektor Bi dengan tabel P-Box, lalu kelompokkan menjadi 4 blok dimana tiap-tiap blok memiliki 32 bit data.

Tabel P-Box

Input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Output	16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
Input	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Output	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Sehingga hasil yang didapat adalah sebagai berikut :

$P(B_1)$ = 01111110 10111000 11100100 01010110
$P(B_2)$ = 00011000 00000101 01000111 01101110
$P(B_3)$ = 01110110 11000000 11100010 11011110
$P(B_4)$ = 11010000 11000110 00110101 00100001
$P(B_5)$ = 11111110 10110001 01111111 10011100
$P(B_6)$ = 10101011 01110011 11110001 10100001
$P(B_7)$ = 00001101 11111110 00101001 11010100
$P(B_8)$ = 10001101 00001010 00110101 10010010
$P(B_9)$ = 00111001 00011011 10100101 01101110
$P(B_{10})$ = 10000010 10011100 11111010 10101011
$P(B_{11})$ = 00100000 01010000 10011100 10100101
$P(B_{12})$ = 00100110 10011101 01100111 01101110
$P(B_{13})$ = 10010110 11110000 01010101 10010010
$P(B_{14})$ = 00111110 00000010 01010101 01010000
$P(B_{15})$ = 00001110 10000110 10011000 10000000
$P(B_{16})$ = 00101111 00101110 11110101 11011010

Hasil: P(B_i) juga bisa di dapat dengan code berikut :

```
# kemudian B1 dipermutasi menggunakan matriks permutasi dan menjadi P(B1)

# Vektor Bi (B1-B16)
Bi = "11011111001110000001101101111000"

# Tabel P-Box
p_box = [
    16, 7, 20, 21, 29, 12, 28, 17,
    1, 15, 23, 26, 5, 18, 31, 10,
    2, 8, 24, 14, 32, 27, 3, 9,
    19, 13, 30, 6, 22, 11, 4, 25
]

# Buat vektor hasil setelah permutasi P-Box
B_permuted = ""
for position in p_box:
    B_permuted += Bi[position - 1]

# Output vektor Bi setelah permutasi P-Box
print("Vektor Bi setelah permutasi P-Box:", B_permuted)
```

Hasil P(Bi) kemudian di XOR kan dengan Li-1 untuk mendapatkan nilai Ri. Sedangkan nilai Li sendiri diperoleh dari Nilai Ri-1 untuk nilai 1<= i<=16.

L0 : 11111111 00010000 01000111 10111110
R0 : 00000000 00000000 11101110 00010010

- Iterasi 1
  P(B1) = 01111110 10111000 11100100 01010110
  L(1)-1 = 11111111 00010000 01000111 10111110
  R1 = 10000001 10101000 10100011 11101000

- Iterasi 2
  P(B2) = 00011000 00000101 01000111 01101110
  L(2)-1 = 00000000 00000000 11101110 00010010
  R2 = 00011000 00000101 10101001 01111100

- Iterasi 3
  P(B3) = 01110110 11000000 11100010 11011110
  L(3)-1 = 10000001 10101000 10100011 11101000
  R3 = 11110111 01101000 01000001 00110110

- Iterasi 4
  P(B4) = 11010000 11000110 00110101 00100001
  L(4)-1 = 00011000 00000101 10101001 01111100
  R4 = 11001000 11000011 10011100 01011101

- Iterasi 5
  P(B5) = 11111110 10110001 01111111 10011100
  L(5)-1 = 11110111 01101000 01000001 00110110
  R5 = 00001001 11011001 00111110 10101010

- Iterasi 6
  P(B6) = 10101011 01110011 11110001 10100001
  L(6)-1 = 11001000 11000011 10011100 01011101
  R6 = 01100011 10110000 0110110 111111100

- Iterasi 7
  P(B7) = 00001101 11111110 00101001 11010100
  L(7)-1 = 00001001 11011001 00111110 10101010
  R7 = 00000100 00100111 00010111 01111110

- Iterasi 8
  P(B8) = 10001101 00001010 00110101 10010010
  L(8)-1 = 01100011 10110000 0110110 111111100
  R8 = 11101110 10111010 01011000 01101110

- Iterasi 9
  P(B9) = 00111001 00011011 10100101 01101110
  L(9)-1 = 00000100 00100111 00010111 01111110
  R9 = 00111101 00111100 10110010 00010000

- Iterasi 10
  P(B10) = 10000010 10011100 11111010 10101011
  L(10)-1 = 11101110 10111010 01011000 01101110
  R10 = 01101100 00100110 10100010 11000101

- Iterasi 11
  P(B11) = 00100000 01010000 10011100 10100101
  L(11)-1 = 00111101 00111100 10110010 00010000
  R11 = 00011101 01101100 00101110 10110101

- Iterasi 12
  P(B12) = 00100110 10011101 01100111 01101110
  L(12)-1 = 01101100 00100110 10100010 11000101
  R12 = 01001010 10111011 11000101 10101011

- Iterasi 13
  P(B13) = 10010110 11110000 01010101 10010010
  L(13)-1 = 00011101 01101100 00101110 10110101
  R13 = 10001011 10011100 01111011 00100111

- Iterasi 14
  P(B14) = 00111110 00000010 01010101 01010000
  L(14)-1 = 01001010 10111011 11000101 10101011
  R14 = 01110100 10111001 10010000 11111011

- Iterasi 15
  P(B15) = 00001110 10000110 10011000 10000000
  L(15)-1 = 10001011 10011100 01111011 00100111
  R15 = 10000101 00011010 11100011 10100111

- Iterasi 16
  P(B16) = 00101111 00101110 11110101 11011010
  L(16)-1 = 01110100 10111001 10010000 11111011
  R16 = 01011011 10010111 01100101 00100001

- L16 nilainya sama dengan R15
  L16 = 10000101 00011010 11100011 10100111

Nilai R_i juga bisa didapatkan dengan code berikut :

```
# Vektor P(Bi) (P(B1)-P(B16))
P_Bi = "01111110101110001110010001010110"

# Vektor Li-1 (L1-L16)
Li_1 = "11111111000100000100011110111110"


# Fungsi XOR antara dua vektor bit
def xor(bitstring1, bitstring2):
    result = ""
    for b1, b2 in zip(bitstring1, bitstring2):
        result += '1' if b1 != b2 else '0'
    return result


# XOR P(Bi) dengan Li untuk mendapatkan Ri
R = xor(P_Bi, Li_1)

print(f"Print Ri = {R}")
print()
```

Langkah 8

Gabungkan R_{16} dengan L_{16} lalu permutasikan untuk terakhir kali dengan tabel Inverse Initial Permutation (IP^{-1}).

Tabel IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Sehingga Input :

$R_{16}L_{16}$ = 01011011 10010111 01100101 00100001 10000101 00011010 11100011 10100111

Menghasilkan Output :

Cipher (dalam biner) = **11011111 01111010 10010110 01100000 01110000 00001111 01001100 10011010**

Atau

Cipher (dalam hexa) = **df 7a 96 60 70 0f 4c 9a**

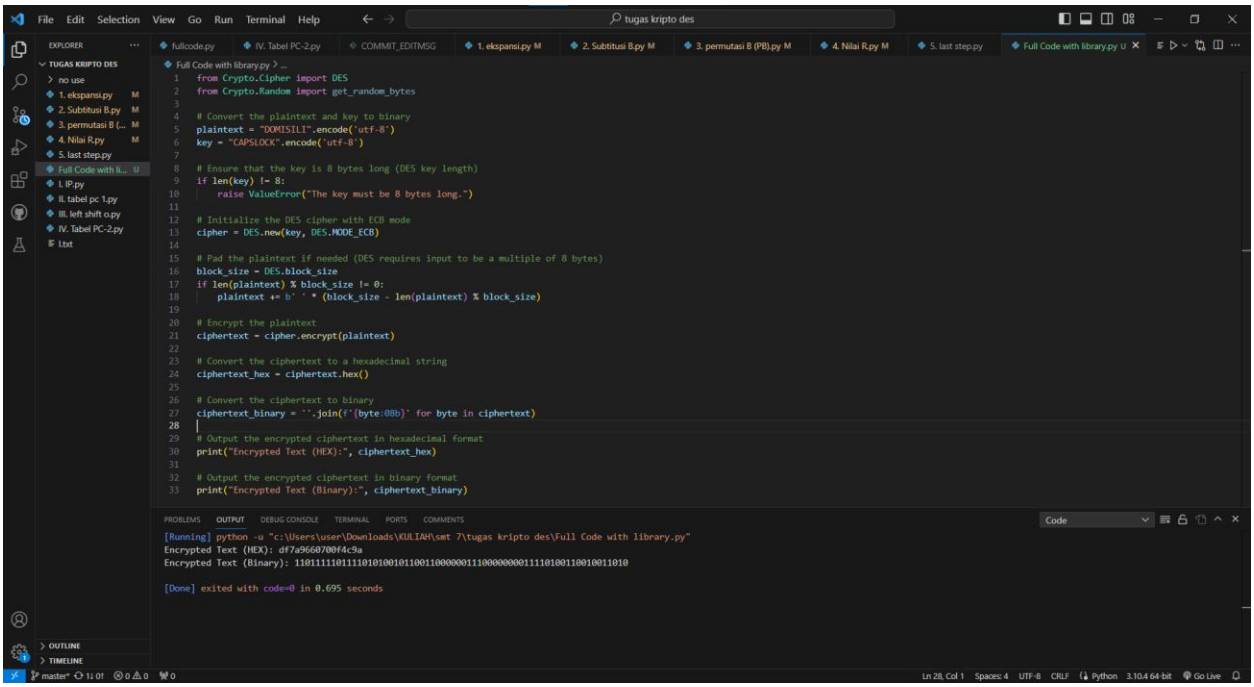
Langkah 8 yaitu Gabungkan R_{16} dengan L_{16} lalu permutasikan untuk terakhir kali dengan tabel Inverse Initial Permutation (IP^{-1}) juga bisa dihasilkan dengan code berikut :

```
# Vektor R16 dan L16
R16 = "01011011100101110110010100100001"
L16 = "10000101000110101110001110100111"

# Tabel Inverse Initial Permutation (IP-1)
ip1_table = [
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41, 9, 49, 17, 57, 25
]

# Gabungkan R16 dan L16
merged = R16 + L16

# Permutasi akhir dengan tabel IP-1
final_permuted = ""
for position in ip1_table:
    final_permuted += merged[position - 1]

# Konversi vektor hasil akhir ke format heksadesimal
cipher_hex = hex(int(final_permuted, 2))[2:]  # Mengabaikan "0x" di awal
heksadesimal

# Output vektor hasil akhir dan cipher dalam format heksadesimal
print("Hasil Akhir (IP-1):", final_permuted)
print("Cipher (HEX):", cipher_hex)
```

Kemudian Untuk memastikan bahwa perhitungan manual hasil enkripsi dari teks DOMISILI dengan kunci CAPSLOCK sudah benar yaitu :

- Cipher (dalam biner) = **11011111 01111010 10010110 01100000 01110000 00001111 01001100 10011010**
- Cipher (dalam hexa) = **df 7a 96 60 70 0f 4c 9a**

Maka saya mencoba untuk melakukan pengecekan dengan library Crypto.Cipher import DES yang telah tersedia pada python. Setelah dicoba hasilnya adalah sama sebagai berikut :



Kode lengkap dapat diakses pada github saya sebagai berikut : https://github.com/cipEpic/DES-Encrypt