Laporan Tugas Programming KIJ - E

Playfair Cipher, DES-CBC, DAA

Anggota Kelompok:

1. Khaela Fortunela 05111940000057
2. Fajar Satria 05111940000083
3. Riki Wahyu 05111940000188

A. Playfair Cipher

1. Introduction

Playfair Cipher adalah teknik enkripsi simetris menggunakan substitusi digram. Pada implementasinya, Playfair Cipher terdiri dari proses sebagai berikut:

a. Membuat key menjadi matrix 5x5
b. Padding plaintext dan menjadikan plaintext menjadi array yang terdiri dari 2 huruf
c. Melakukan proses enkripsi

2. Code and Comment

a. Membuat key menjadi matrix 5x5.

```
def make_key_matrix(key):
    key_array = []

    #append key first
    for i in key:
        #special rule for letter J
        if i == "J": i = "I"

        #check if the letter already in array
        if i not in key_array:
            key_array.append(i)

    alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K',
'L', 'M','N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z']
    #then the rest of alphabet
    for i in alphabet:
        if i not in key:
            key_array.append(i)
```

```python
    #make into 5x5 matrix
    key_matrix = []
    while key_array != []:
        key_matrix.append(key_array[:5])
        key_array = key_array[5:]
        #print(key_array)

    #print(key_matrix)
    return key_matrix
```

b. Padding plaintext dan menjadikan plaintext menjadi array yang terdiri dari 2 huruf.

```python
def prepare_text(plaintext):
    array_text=[]
    prev = ""
    for i in plaintext:
        #special rule for letter J
        if i == "J": i = "I"

        #get last element if array not empty
        if len(array_text)>0:
            prev = array_text[-1]

        #append x if current and prev is the same
        if prev == i:
            array_text.append("X")

        #append the rest of text except space character
        if i != " ":
            array_text.append(i)
        else:
            continue

    #if text is odd length, add Z
    if len(array_text)%2:
        array_text.append("Z")

    return array_text
```

```python
def split_text(plaintext):
    array_text = prepare_text(plaintext)

    #split into 2d array [][2]
    two_letter_array = []
    while array_text != []:
        two_letter_array.append(array_text[:2])
        array_text = array_text[2:]
    return two_letter_array
```

c. Melakukan proses enkripsi

```python
def encryption_rules(first_index, second_index):
    #if in the same column, shift down
    if first_index[1] == second_index[1]:
        first_index[0] = add_index(first_index[0])
        second_index[0] = add_index(second_index[0])

    #if in the same row, shift right
    if first_index[0] == second_index[0]:
        first_index[1] = add_index(first_index[1])
        second_index[1] = add_index(second_index[1])

    #else, switch using rectangle, horizontal opposite
    else:
        hold = first_index[1]
        first_index[1] = second_index[1]
        second_index[1] = hold

    return [first_index, second_index]

def start_encrypt(matrix_key, plaintext):
    encrypted_text = ""

    length = len(plaintext)
    for i in range(length):
        #get index in matrix key
        first_index = search(matrix_key, plaintext[i][0])
        second_index = search(matrix_key, plaintext[i][1])
```

```
        #apply encryption rules
        new_first, new_second = encryption_rules(first_index,
second_index)

        #append encrypted letters to text
        encrypted_text += matrix_key[new_first[0]][new_first[1]]
        encrypted_text += matrix_key[new_second[0]][new_second[1]]
    return encrypted_text
```

3. Operasional

```
KEY:  XJVWOIAGJERIGNIESOR
PLAINTEXT:  JDNVGSOERQWOEKFOERMGKW
KEY MATRIX =  [['X', 'I', 'V', 'W', 'O'], ['A', 'G', 'E', 'R', 'N'], ['S',
'B', 'C', 'D', 'F'], ['H', 'K', 'L', 'M', 'P'], ['Q', 'T', 'U', 'Y', 'Z']]
ENCRYPTED TEXT: WBEOABVNAYOXGLPNRNKRMI
```

B. DES - CBC

1. Introduction

DES (Data Encryption Standard) adalah algoritma enkripsi dengan key simetris. DES-CBC (Cipher Block Chaining) merupakan implementasi DES menggunakan mode operasi enkripsi per blok dengan ukuran yang tetap. Pada implementasinya, DES-CBC secara garis besar terdiri dari proses sebagai berikut:

   a. Pembuatan key untuk 16 round
   b. Initial permutation dari plaintext
   c. Proses enkripsi 16 round
   d. Final permutation

2. Code and Comment

   a. Pembuatan key untuk 16 round

      1) Convert key menjadi binary string.

2) Permutasi key dengan menghilangkan parity bits (bit 8, 16, 24, ..) menggunakan tabel parity bit drop.

Table: Parity-bit drop table

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

3) Membagi key menjadi dua bagian.
4) Shift kedua bagian sesuai schedule left shift.

(d) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

5) Gabungkan kedua bagian dan kompresi dari 56 bit menjadi 48 bit menggunakan tabel PC-2.

(c) Permuted Choice Two (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

6) Simpan key ke dalam array.

```
#function to prepare 16 round of key
def prepare_key(key):
    #convert into binary
    key = hex_to_bin(key)

    #permute key into 56 bit by dropping parity bits (bit 8, 16, 24, ...)
    key = permute(key, parity_drop_table, 56)

    #split key into two parts, 28 bit each
```

```python
    left_key = key[0:28]
    right_key = key[28:56]

    #array to hold 16 round of key
    key_round_bin = []

    for i in range(16):
        #shift according to the shift table scheme
        left_key = shift_left(left_key, shift_table[i])
        right_key = shift_left(right_key, shift_table[i])

        #combine both parts
        combined = left_key + right_key

        #compress key from 56 bit to 48 bit using PC-2 table or
compression permutation table
        round_key = permute(combined, key_compression_table, 48)

        #append each round of key
        key_round_bin.append(round_key)

    return key_round_bin
```

b. Initial permutation dari plaintext

Permutasi awal dari plaintext menggunakan tabel Initial Permutation (IP).

(a) Initial Permutation (IP)

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

```python
def encrypt(plaintext, key):
    ...
    plaintext = permute(plaintext, initial_perm_table, 64)
    ...
```

c. Proses enkripsi 16 round

1) Bagi plaintext menjadi 2 bagian, left dan right.
2) Loop untuk 16 round:

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

a) Expand right text dari 32 bit menjadi 48 bit dengan tabel expansion permutation (E).

(c) Expansion Permutation (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

b) Melakukan XOR antara right text dengan key round sekarang.

$$K_n \oplus \mathbf{E}(R_{n-1})$$

c) Menggunakan tabel S-Box, transformasi tiap 6 bit menjadi 4 bit, sehingga di akhir berubah dari 48 bit menjadi 32 bit.

$$K_n \oplus \mathbf{E}(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

d) Lakukan XOR antara left text dan hasil dari transformasi S-Box.
e) Simpan hasil sebagai left text (untuk round selanjutnya).
f) Swap right text dan left text.

```
def encrypt_round(left, right, key_round, i):
    #expand right text from 32 bit to 48 bit using expansion permutation table
    right_expand = permute(right, expansion_table, 48)

    #xor the 48 bit right text with current round key
    xor_r = xor(right_expand, key_round[i])

    #transform each 6 bits into 4 bits, resulting from 48 bits into total of 32 bits
    sbox_str = ""
    for j in range(0, 8):
```

```
        row = bin_to_dec(int(xor_r[j * 6] + xor_r[j * 6 + 5]))
        col = bin_to_dec(int(xor_r[j * 6 + 1] + xor_r[j * 6 + 2] +
xor_r[j * 6 + 3] + xor_r[j * 6 + 4]))
        val = s_box[j][row][col]
        sbox_str += dec_to_bin(val)
    #permutation of s-box output
    sbox_str = permute(sbox_str, permutation_table, 32)

    #xor left text with x-box output
    result = xor(left, sbox_str)
    #save result for next round
    left = result

    #swap left text and right text
    if(i != 15):
        temp = left
        left = right
        right = temp

    return [left, right]

def encrypt(plaintext, key):
    ...
    left_text = plaintext[0:32]
    right_text = plaintext[32:64])

    for i in range(0, 16):
        left_text, right_text = encrypt_round(left_text, right_text,
key_round, i)
    ...
```

d. Final permutation

Dilakukan permutasi final menggunakan tabel inverse initial permutation (IP^-1).

(b) Inverse Initial Permutation (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

```
def encrypt(plaintext, key):
    ...
    ciphertext = permute(round_result, final_permutation_table, 64)
    ...
```

e. Tambahan: Proses Decrypt

Proses decrypt memiliki langkah yang sama dengan enkripsi, perbedaan ada di round key yang digunakan. Pada proses decrypt, urutan round key di reverse.

```
def decrypt(ciphertext, key):
    ...
    key_round = prepare_key(key)[::-1]
    ...
```

3. Operasional

```
PLAINTEXT = 123456ABCDEF1234
KEY = AABBCCDDEEFF1234

INITIAL PERMUTATION: 34C7B63838AA386D
LEFT0: 34C7B638 | RIGHT0: 38AA386D
ROUND 1: 38AA386D 58B10531 365FB3EB96AE
ROUND 2: 58B10531 D9700D8A 7EFB011BF7EA
ROUND 3: D9700D8A 9E1D60E6 0BBD7F3CFD25
ROUND 4: 9E1D60E6 EEE48130 ED64DFEA6CF6
ROUND 5: EEE48130 1DC28E7B 77CFA8EDEB9B
ROUND 6: 1DC28E7B F43C7470 DAB9B3B7565B
ROUND 7: F43C7470 10AB1EF5 BDAE5FDF9366
ROUND 8: 10AB1EF5 4444310C 67768E94EFEC
ROUND 9: 4444310C 5038BFD0 7437ED557CCD
ROUND 10: 5038BFD0 8C3D4C8D D3DC71EAB0FD
ROUND 11: 8C3D4C8D 9B261A08 CDEBF6A3FFAF
ROUND 12: 9B261A08 F52F6550 B6F78F3E1FB3
ROUND 13: F52F6550 B6CDDC21 7B1763DF4977
ROUND 14: B6CDDC21 CC99066E E9D8FD47EBD8
ROUND 15: CC99066E 6DB9049F 95E3DEF1B55D
ROUND 16: 4CF47CE0 6DB9049F F64FDEFF965D

CIPHERTEXT = A202DEE636B5D533
```

C. DAA

1. Introduction

DAA (Data Authentication Algorithm) adalah MAC (Message Authentication Code) berbasis DES yang cukup banyak digunakan. Algoritma ini menggunakan mode CBC dari operasi DES. Dengan menggunakan algoritma enkripsi DES (E) dan secret key (K), nilai dari data authentication code (DAC) dapat dikalkulasikan. Nilai DAC diambil dari O_N, dengan N adalah jumlah bit plaintext dibagi menjadi 64 bit block.

$$\begin{aligned} O_1 &= \mathrm{E}(K, D) \\ O_2 &= \mathrm{E}(K, [D_2 \oplus O_1]) \\ O_3 &= \mathrm{E}(K, [D_3 \oplus O_2]) \\ &\vdots \\ O_N &= \mathrm{E}(K, [D_N \oplus O_{N-1}]) \end{aligned}$$

Proses DAA secara garis besar terdiri dari:

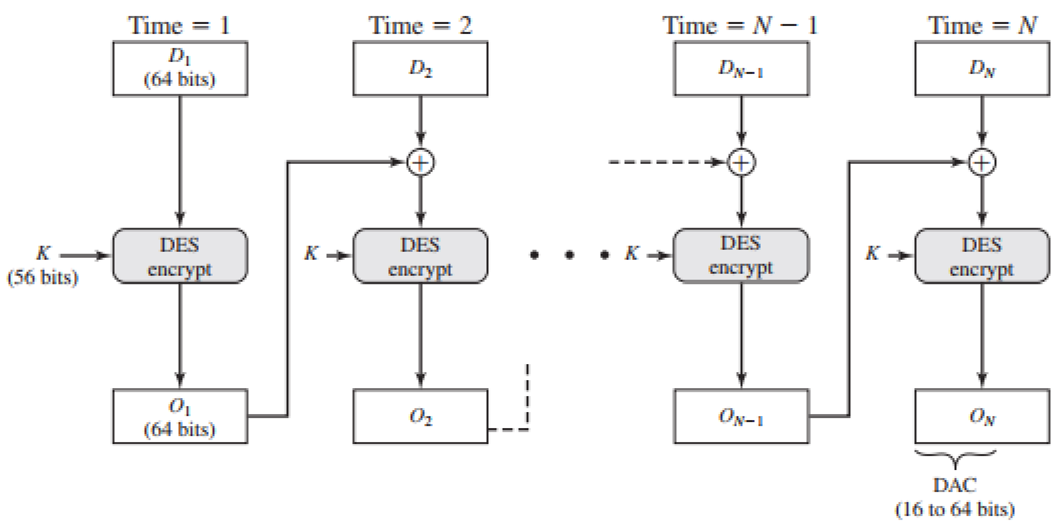

Figure 12.7 Data Authentication Algorithm (FIPS PUB 113)

a. Operasi enkripsi blok pertama (O_1)
b. Untuk N kali:
   1) Proses padding string D_N menjadi 64 bit, apabila masih kurang dari 64 bit.
   2) XOR antara D_N dan O_{N-1}.
   3) Proses enkripsi dari hasil XOR antara D_N dan O_{N-1}.
c. Mengambil DAC dari bit O_N.

2. Code and Comment

```
import descbc as des #import descbc.py as library of function

plaintext = "123456ABCDEF1234123456ABCDEF1234"
```

```python
key = "AABBCCDDEEFF1234"

#take first 64 bit of text (16 hex code)
d1 = plaintext[0:16]
plaintext = plaintext[16:]
result = []
result.append(des.encrypt(d1, key))

length = len(plaintext)
for i in range(0, 16, length):
    #take subsequent 64 bit of text (16 hex code)
    dn = plaintext[i:i+16]
    dn = des.hex_to_bin(dn)
    counter = 64 - len(dn)

    #pad with 0 until it’s 64 bit
    for i in range(0, counter):
        dn += '0'

    #xor the current input plaintext with previous result
    text = des.xor(dn, des.hex_to_bin(result[i]))

    #append to array of result
    result.append(des.encrypt(des.bin_to_hex(text), key))

#get 16 bit of last result (size of dac is 16 ≤ x ≤ 64)
dac = des.hex_to_bin(result[-1])
dac = dac[0:16]
print("Data Authentication Code: " + dac + " | " + des.bin_to_hex(dac))
```

3. Operasional

```
ENCRYPTION

PLAINTEXT = 123456ABCDEF1234
KEY = AABBCCDDEEFF1234

INITIAL PERMUTATION: 34C7B63838AA386D
LEFT0: 34C7B638 | RIGHT0: 38AA386D
ROUND 1: 38AA386D 58B10531 365FB3EB96AE
ROUND 2: 58B10531 D9700D8A 7EFB011BF7EA
ROUND 3: D9700D8A 9E1D60E6 0BBD7F3CFD25
ROUND 4: 9E1D60E6 EEE48130 ED64DFEA6CF6
ROUND 5: EEE48130 1DC28E7B 77CFA8EDEB9B
ROUND 6: 1DC28E7B F43C7470 DAB9B3B7565B
ROUND 7: F43C7470 10AB1EF5 BDAE5FDF9366
ROUND 8: 10AB1EF5 4444310C 67768E94EFEC
ROUND 9: 4444310C 5038BFD0 7437ED557CCD
ROUND 10: 5038BFD0 8C3D4C8D D3DC71EAB0FD
ROUND 11: 8C3D4C8D 9B261A08 CDEBF6A3FFAF
ROUND 12: 9B261A08 F52F6550 B6F78F3E1FB3
ROUND 13: F52F6550 B6CDDC21 7B1763DF4977
ROUND 14: B6CDDC21 CC99066E E9D8FD47EBD8
ROUND 15: CC99066E 6DB9049F 95E3DEF1B55D
ROUND 16: 4CF47CE0 6DB9049F F64FDEFF965D

CIPHERTEXT = A202DEE636B5D533
_________________________________
ENCRYPTION
```

```
PLAINTEXT = B036884DFB5AC707
KEY = AABBCCDDEEFF1234

INITIAL PERMUTATION: 7833CAD855133CF2
LEFT0: 7833CAD8 | RIGHT0: 55133CF2
ROUND 1: 55133CF2 217946E4 365FB3EB96AE
ROUND 2: 217946E4 5810A4E4 7EFB011BF7EA
ROUND 3: 5810A4E4 DC89F4A9 0BBD7F3CFD25
ROUND 4: DC89F4A9 9286068D ED64DFEA6CF6
ROUND 5: 9286068D 149D5EDA 77CFA8EDEB9B
ROUND 6: 149D5EDA A8477BAA DAB9B3B7565B
ROUND 7: A8477BAA 6D100CFA BDAE5FDF9366
ROUND 8: 6D100CFA 1E66E1D2 67768E94EFEC
ROUND 9: 1E66E1D2 6D6CEF98 7437ED557CCD
ROUND 10: 6D6CEF98 3EE0BF3F D3DC71EAB0FD
ROUND 11: 3EE0BF3F D105687C CDEBF6A3FFAF
ROUND 12: D105687C A00B15DA B6F78F3E1FB3
ROUND 13: A00B15DA 228BBAFA 7B1763DF4977
ROUND 14: 228BBAFA 1D411123 E9D8FD47EBD8
ROUND 15: 1D411123 49A9EAC1 95E3DEF1B55D
ROUND 16: 0A5F933E 49A9EAC1 F64FDEFF965D

CIPHERTEXT = B65D11F915299A2E

_________________________________
Data Authentication Code: 1011011001011101 | B65D
```