

Лабораторна робота №2

Симетричне шифрування

Мета: Дослідити принципи роботи симетричного шифрування на прикладі алгоритму AES.

Завдання: Реалізувати алгоритм симетричного шифрування AES (будь-якої версії - 128 або 256). Довести коректність роботи реалізованого алгоритму шляхом порівняння результатів з існуючими реалізаціями.

Хід роботи:

Код програми:

```
from copy import copy    # It is used to copy the array

ROUND = 10              # Number of round
WORD_LENGTH = 4         # Length of matrix

# Most commonly used S_BOX
S_BOX = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
)

# Most commonly used INVERSE_S_BOX
INVERSE_S_BOX = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
```

```

    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
)

# Most commonly used R_CON
R_CON = (
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
    0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
    0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
)

# It takes a string and converts it to a hex string array
def translate_string_into_hex_str(string):
    hex_arr = []
    for i in range(len(string)):
        char_val = string[i].encode('utf8')
        hex_val = char_val.hex()
        hex_arr.append(hex_val)
    return hex_arr

# It takes a hex string array and converts it to a string
def translate_hex_into_str(hex_str):
    text = ""
    for i in range(WORD_LENGTH * WORD_LENGTH):
        text = text + (chr(int(hex_str[i], 16)))

    return text

# It shifts left each item of the array
def left_shift(arr):
    temp = arr[0]
    for i in range(len(arr) - 1):
        arr[i] = arr[i + 1]
    arr[len(arr) - 1] = temp
    return arr

# It shifts right each item of the array
def right_sift(arr):
    temp = arr[WORD_LENGTH - 1]
    i = len(arr) - 1
    while i > 0:
        arr[i] = arr[i - 1]
        i = i - 1
    arr[0] = temp
    return arr

# It takes main key value as a hex string array and generate all round keys
# It return 11 keys as an array
# First key is the main key and the others are generated keys
def find_all_round_keys(main_key_val):
    w = [[0 for x in range(WORD_LENGTH)] for y in range(44)]

    # If the length of the key is smaller than 16 bytes then make it 16 bytes with adding '01'
    if len(main_key_val) < 16:
        for i in range(16 - len(main_key_val)):
            main_key_val.append('01')

    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):

```

```

        w[i][j] = int(main_key_val[(i * WORD_LENGTH) + j], 16) # Converts hex string array to integer

round_num = 0
for word_num in range(WORD_LENGTH, WORD_LENGTH * (ROUND + 1)):

    if word_num % WORD_LENGTH == 0:
        round_num = round_num + 1
        temp_w = [0, 0, 0, 0]
        for i in range(4):
            temp_w[i] = w[word_num - 1][i]
        temp_w = left_shift(temp_w)
        s_box_arr = []

        for i in range(WORD_LENGTH):
            s_box_index = int(temp_w[i])
            s_box_arr.append(S_BOX[s_box_index])

        round_cont_arr = [0, 0, 0, 0]
        round_cont_arr[0] = round_cont_arr[0] + R_CON[round_num]
        for i in range(WORD_LENGTH):
            s_box_arr[i] = s_box_arr[i] ^ round_cont_arr[i]

        for i in range(WORD_LENGTH):
            int_val = int(w[word_num - 4][i])
            w[word_num][i] = int_val ^ s_box_arr[i]

    else:
        for i in range(WORD_LENGTH):
            int_val_minus_1 = int(w[word_num - 1][i])
            int_val_minus_4 = int(w[word_num - 4][i])
            w[word_num][i] = int_val_minus_1 ^ int_val_minus_4

temp_round_keys_hex = []
for i in range(WORD_LENGTH * (ROUND + 1)):
    for j in range(WORD_LENGTH):
        temp_round_keys_hex.append("{:02x}".format(w[i][j]))

round_keys_in_hex = [temp_round_keys_hex[i * (WORD_LENGTH * WORD_LENGTH):(i + 1) * (WORD_LENGTH *
WORD_LENGTH)]]
        for i in range(
            (len(temp_round_keys_hex) + (WORD_LENGTH * WORD_LENGTH) - 1) // (WORD_LENGTH * WORD_LENGTH))]

return round_keys_in_hex

# It takes hex string array and generates 4x4 matrix
def generate_4x4_matrix(hex_string):
    hex_string_arr = [hex_string[i:i + WORD_LENGTH] for i in range(0, len(hex_string), WORD_LENGTH)]
    matrix_4x4 = [['' for x in range(WORD_LENGTH)] for y in range(WORD_LENGTH)]
    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            matrix_4x4[i][j] = hex_string_arr[j][i]

    return matrix_4x4

def make_int_arr_to_hex(int_arr):
    hex_arr = ['' for x in range(len(int_arr))]
    for i in range(len(int_arr)):
        hex_arr[i] = "{:02x}".format(int_arr[i])

    return hex_arr

# It takes two hex string 4x4 matrix
# It converts matrix to the integer version of the matrix and XOR them

```

```

# It converts XOR results to the hex string 4x4 matrix
def add_round_key(m1_hex, m2_hex):
    matrix_4x4 = [['' for x in range(WORD_LENGTH)] for y in range(WORD_LENGTH)]

    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            val_1 = int(m1_hex[i][j], 16)
            val_2 = int(m2_hex[i][j], 16)
            res = val_1 ^ val_2
            matrix_4x4[i][j] = "{:02x}".format(res)

    return matrix_4x4

def substitute_bytes(matrix, normal_or_inv):
    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            int_val = int(matrix[i][j], 16)
            if normal_or_inv:
                s_box_val_int = S_BOX[int_val]
            else:
                s_box_val_int = INVERSE_S_BOX[int_val]
            s_box_val_hex = "{:02x}".format(s_box_val_int)
            matrix[i][j] = s_box_val_hex

def shift_row(matrix):
    matrix[1] = left_shift(matrix[1])

    matrix[2] = left_shift(matrix[2])
    matrix[2] = left_shift(matrix[2])

    matrix[3] = left_shift(matrix[3])
    matrix[3] = left_shift(matrix[3])
    matrix[3] = left_shift(matrix[3])

def shift_row_inv(matrix):
    matrix[1] = right_sift(matrix[1])
    matrix[2] = right_sift(matrix[2])
    matrix[2] = right_sift(matrix[2])
    matrix[3] = right_sift(matrix[3])
    matrix[3] = right_sift(matrix[3])
    matrix[3] = right_sift(matrix[3])

# It takes a hex string array and converts it to integer array
def make_matrix_int(matrix):
    int_matrix = [[0 for x in range(WORD_LENGTH)] for y in range(WORD_LENGTH)]
    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            int_matrix[i][j] = int(matrix[i][j], 16)
    return int_matrix

# It is Galois Multiplication
def galois_multiplication(a, b):
    p = 0
    hi_bit_set = 0
    for i in range(8):
        if b & 1 == 1:
            p ^= a
            hi_bit_set = a & 0x80
            a <<= 1
        if hi_bit_set == 0x80:
            a ^= 0x1b

```

```

    b >= 1
    return p % 256

```

```

def mix_column(column):
    temp = copy(column)
    column[0] = galois_multiplication(temp[0], 2) ^ galois_multiplication(temp[3], 1) ^ \
        galois_multiplication(temp[2], 1) ^ galois_multiplication(temp[1], 3)
    column[1] = galois_multiplication(temp[1], 2) ^ galois_multiplication(temp[0], 1) ^ \
        galois_multiplication(temp[3], 1) ^ galois_multiplication(temp[2], 3)
    column[2] = galois_multiplication(temp[2], 2) ^ galois_multiplication(temp[1], 1) ^ \
        galois_multiplication(temp[0], 1) ^ galois_multiplication(temp[3], 3)
    column[3] = galois_multiplication(temp[3], 2) ^ galois_multiplication(temp[2], 1) ^ \
        galois_multiplication(temp[1], 1) ^ galois_multiplication(temp[0], 3)

def mix_column_inv(column):
    temp = copy(column)
    column[0] = galois_multiplication(temp[0], 14) ^ galois_multiplication(temp[3], 9) ^ \
        galois_multiplication(temp[2], 13) ^ galois_multiplication(temp[1], 11)
    column[1] = galois_multiplication(temp[1], 14) ^ galois_multiplication(temp[0], 9) ^ \
        galois_multiplication(temp[3], 13) ^ galois_multiplication(temp[2], 11)
    column[2] = galois_multiplication(temp[2], 14) ^ galois_multiplication(temp[1], 9) ^ \
        galois_multiplication(temp[0], 13) ^ galois_multiplication(temp[3], 11)
    column[3] = galois_multiplication(temp[3], 14) ^ galois_multiplication(temp[2], 9) ^ \
        galois_multiplication(temp[1], 13) ^ galois_multiplication(temp[0], 11)

def mix_columns(matrix, normal_or_inv):
    int_matrix = make_matrix_int(matrix)
    col_based = [[0 for x in range(WORD_LENGTH)] for y in range(WORD_LENGTH)]

    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            col_based[i][j] = int_matrix[j][i]

    if normal_or_inv:
        for i in range(WORD_LENGTH):
            mix_column(col_based[i])
    else:
        for i in range(WORD_LENGTH):
            mix_column_inv(col_based[i])

    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            matrix[i][j] = "{:02x}".format(col_based[j][i])
            int_matrix[i][j] = col_based[j][i]

def encrypt(text_hex, all_round_keys):
    state_matrix = generate_4x4_matrix(text_hex)
    round_N_matrix = generate_4x4_matrix(all_round_keys[0])
    current_matrix = add_round_key(state_matrix, round_N_matrix)

    for i in range(1, ROUND):
        substitute_bytes(current_matrix, True)
        shift_row(current_matrix)
        mix_columns(current_matrix, True)
        current_matrix = add_round_key(current_matrix, generate_4x4_matrix(all_round_keys[i]))

    substitute_bytes(current_matrix, True)
    shift_row(current_matrix)
    cipher_text_matrix = add_round_key(current_matrix, generate_4x4_matrix(all_round_keys[10]))

    cipher_text = []
    for i in range(WORD_LENGTH):

```

```

        for j in range(WORD_LENGTH):
            cipher_text.append(cipher_text_matrix[j][i])

    return cipher_text

def decryption(cipher_text, all_round_keys):
    state_matrix = generate_4x4_matrix(cipher_text)
    round_N_matrix = generate_4x4_matrix(all_round_keys[10])

    current_matrix = add_round_key(state_matrix, round_N_matrix)
    shift_row_inv(current_matrix)
    substitute_bytes(current_matrix, False)

    i = 9
    while i > 0:
        current_matrix = add_round_key(current_matrix, generate_4x4_matrix(all_round_keys[i]))
        mix_columns(current_matrix, False)
        shift_row_inv(current_matrix)
        substitute_bytes(current_matrix, False)
        i = i - 1

    plain_text_matrix = add_round_key(current_matrix, generate_4x4_matrix(all_round_keys[0]))

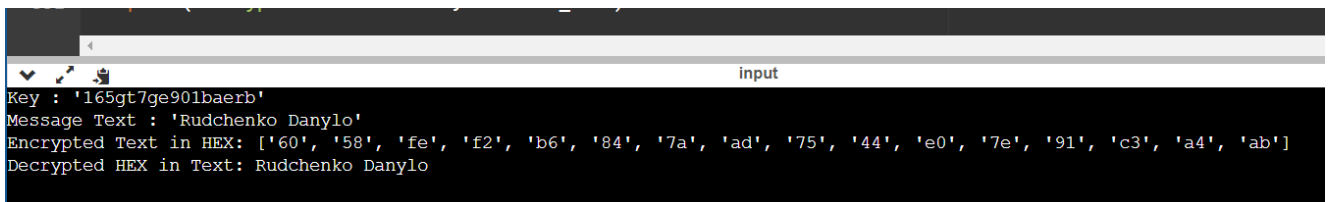
    plain_text = []
    for i in range(WORD_LENGTH):
        for j in range(WORD_LENGTH):
            plain_text.append(plain_text_matrix[j][i])

    return plain_text

KEYS = ["165gt7ge901baerb"]
TEXTS = ["Rudchenko Danylo"]

for i in range(1):
    main_key = translate_string_into_hex_str(KEYS[i])
    main_text = translate_string_into_hex_str(TEXTS[i])
    round_keys = find_all_round_keys(main_key)
    encrypt_text = encrypt(main_text, round_keys)
    encrypt_text_str = translate_hex_into_str(encrypt_text)
    decrypt_text = decryption(encrypt_text, round_keys)
    resolved_text = translate_hex_into_str(decrypt_text)
    print("Key : '{}'".format(KEYS[i]))
    print("Message Text : '{}'".format(TEXTS[i]))
    print("Encrypted Text in HEX:", encrypt_text)
    print("Decrypted HEX in Text:", resolved_text)

```



```

Key : '165gt7ge901baerb'
Message Text : 'Rudchenko Danylo'
Encrypted Text in HEX: ['60', '58', 'fe', 'f2', 'b6', '84', '7a', 'ad', '75', '44', 'e0', '7e', '91', 'c3', 'a4', 'ab']
Decrypted HEX in Text: Rudchenko Danylo

```

Рис. 1 – Результат роботи програми

Enter Secret Key -

As AES is a symmetric encryption and decryption key size dropdown
So if key size is 128 then 16 characters i.e. 16*8=

Output Text Format

Specify if output format

6058fef2b6847aad7544e07e91c3a4ab0206aa763cc12de11f0516c3c7ec0864

Рис. 2 – Перевірка результатів шифрування (успішно)

AES Decryption

Enter Encrypted Text to Decrypt -

Input Text Format

Select Mode

Key Size in Bits

Enter Initialization Vector -

Enter Secret Key -

Rudchenko Danylo

Рис. 3 – Перевірка результатів дешифрування (успішно)

Висновок: в результаті виконання лабораторної роботи було досліджено принципи роботи симетричного шифрування на прикладі алгоритму AES.