МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО
КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ЗВІТ

ЗВІТ З ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ТЕХНОЛОГІЇ ЗАХИСТУ ІНФОРМАЦІЇ»

Виконав студент групи КН-23-1
Полинько Ігор Миколайович
Перевірив: ассистент кафедри АІС Андреєв П. І.

Кременчук 2025

ЛАБОРАТОРНА РОБОТА № 6

Тема: Шифрування алгоритмом DES

Мета: навчитися створювати програми шифрування (дешифрування) алгоритмом DES.

Порядок виконання роботи:

Реалізувати алгоритм DES (шифрування та дешифрування):

7−8 Робочий режим зворотний зв'язок за виходом OFB (Output Feed Back).

Варіант: 15

Скрипт програми:

```
from math import ceil
import os

def bytes_to_bits(b: bytes) -> list:
    bits = []
    for byte in b:
        bits.extend([ (byte >> i) & 1 for i in range(7, -1, -1) ])
    return bits

def bits_to_bytes(bits: list) -> bytes:
    if len(bits) % 8 != 0:
        bits = bits + [0] * (8 - (len(bits) % 8))
    out = bytearray()
    for i in range(0, len(bits), 8):
        byte = 0
        for bit in bits[i:i+8]:
            byte = (byte << 1) | bit
        out.append(byte)
    return bytes(out)

def string_to_bits(s: str) -> list:
    return bytes_to_bits(s.encode('utf-8'))

def bits_to_string(bits: list) -> str:
    b = bits_to_bytes(bits)
    return b.decode('utf-8', errors='replace')

def xor_bits(a: list, b: list) -> list:
    n = min(len(a), len(b))
    return [ (a[i] ^ b[i]) for i in range(n) ]

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
]
```

```
FP = [
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
]

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
]

P = [
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
]

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
]
]

PC1 = [
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
]

PC2 = [
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
]

ROTATIONS = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

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

def left_rotate(l: list, n: int) -> list:
    return l[n:] + l[:n]

def generate_round_keys(key_bytes: bytes) -> list:
    key_bits = bytes_to_bits(key_bytes)
    key56 = permute(key_bits, PC1)
    C = key56[:28]
    D = key56[28:]
    round_keys = []
    for r in range(16):
        C = left_rotate(C, ROTATIONS[r])
        D = left_rotate(D, ROTATIONS[r])
        CD = C + D
        round_key = permute(CD, PC2)
        round_keys.append(round_key)
```

```
    return round_keys

def sbox_substitution(bits48: list) -> list:
    out = []
    for i in range(8):
        block6 = bits48[i*6:(i+1)*6]
        row = (block6[0] << 1) | block6[5]  # 1-й и 6-й биты
        col = (block6[1] << 3) | (block6[2] << 2) | (block6[3] << 1) |
block6[4]
        val = S_BOXES[i][row*16 + col]
        out.extend([ (val >> 3) & 1, (val >> 2) & 1, (val >> 1) & 1, val & 1
])
    return out

def feistel(R: list, round_key: list) -> list:
    expanded = permute(R, E)
    xored = [ expanded[i] ^ round_key[i] for i in range(48) ]
    sboxed = sbox_substitution(xored)
    permuted = permute(sboxed, P)
    return permuted

def des_encrypt_block_bits(block_bits: list, key_bytes: bytes) -> list:
    if len(block_bits) != 64:
        block_bits = block_bits + [0] * (64 - len(block_bits))
    round_keys = generate_round_keys(key_bytes)
    permuted = permute(block_bits, IP)
    L = permuted[:32]
    R = permuted[32:]
    for rk in round_keys:
        f_out = feistel(R, rk)
        newR = [ L[i] ^ f_out[i] for i in range(32) ]
        L = R
        R = newR
    preoutput = R + L
    cipher_bits = permute(preoutput, FP)
    return cipher_bits

def des_decrypt_block_bits(block_bits: list, key_bytes: bytes) -> list:
    if len(block_bits) != 64:
        block_bits = block_bits + [0] * (64 - len(block_bits))
    round_keys = generate_round_keys(key_bytes)[::-1]
    permuted = permute(block_bits, IP)
    L = permuted[:32]
    R = permuted[32:]
    for rk in round_keys:
        f_out = feistel(R, rk)
        newR = [ L[i] ^ f_out[i] for i in range(32) ]
        L = R
        R = newR
    preoutput = R + L
    plain_bits = permute(preoutput, FP)
    return plain_bits

def simple_des_block_encrypt(block_bits: list, key: bytes) -> list:
    return des_encrypt_block_bits(block_bits[:64], key)

def simple_des_block_decrypt(block_bits: list, key: bytes) -> list:
    return des_decrypt_block_bits(block_bits[:64], key)

def des_ofb_encrypt(plaintext: str, key: bytes, iv: bytes) -> (bytes, bytes):
    pt_bits = string_to_bits(plaintext)
    B = 64
    prev = bytes_to_bits(iv)
    cipher_bits = []
```

```
    for i in range(0, len(pt_bits), B):
        block = pt_bits[i:i+B]
        keystream = simple_des_block_encrypt(prev, key)
        ks_slice = keystream[:len(block)]
        cipher_block = xor_bits(block, ks_slice)
        cipher_bits.extend(cipher_block)
        prev = keystream

    cipher_bytes = bits_to_bytes(cipher_bits)
    return iv, cipher_bytes

def des_ofb_decrypt(cipher_bytes: bytes, key: bytes, iv: bytes) -> str:
    cipher_bits = bytes_to_bits(cipher_bytes)
    B = 64
    prev = bytes_to_bits(iv)
    plain_bits = []

    for i in range(0, len(cipher_bits), B):
        block = cipher_bits[i:i+B]
        keystream = simple_des_block_encrypt(prev, key)
        ks_slice = keystream[:len(block)]
        plain_block = xor_bits(block, ks_slice)
        plain_bits.extend(plain_block)
        prev = keystream

    return bits_to_string(plain_bits)

if __name__ == "__main__":
    key = b'12345678'
    iv = os.urandom(8)

    plaintext = "Це тестовий текст для шифрування DES OFB — перевірка
кирилиці та довжини"

    print("Текст для шифрування:", plaintext)
    iv_out, ciphertext = des_ofb_encrypt(plaintext, key, iv)
    print("IV:", iv_out.hex())
    print("Зашифрований текст:", ciphertext.hex())

    recovered = des_ofb_decrypt(ciphertext, key, iv_out)
    print("Дешифрований текст:", recovered)
```

Результат:

```
Текст для шифрування: Це тестовий текст для шифрування DES OFB — перевірка кирилиці та довжини
IV: 73387b36ab6c798f
Зашифрований текст: 8192f935aa7d9d03c2ad89b39937741a407e390d22d15fd9b06b1afefb16c384f2e0dbebccddcd5b95
Дешифрований текст: Це тестовий текст для шифрування DES OFB — перевірка кирилиці та довжини
```

Рисунок 6.1 – Результат роботи програми

Висновок: на цій лабораторній роботі ми навчилися створювати програми шифрування (дешифрування) алгоритмом DES.

Контрольні питання:

1. Структура алгоритму DES

Алгоритм DES є блочним симетричним шифром.

– Довжина блока: 64 біти, ключа — 56 біт.

– Процес складається з:

1) Початкової перестановки (IP);

2) 16 раундів за схемою Фейстеля:

– поділ блока на півблоки L і R;

– у кожному раунді:

L_i = R_{i-1},

R_i = L_{i-1} XOR F(R_{i-1}, K_i);

3) Кінцевої перестановки (IP^{-1}).

Розшифрування виконується аналогічно, але підключі використовуються у зворотному порядку.

2. Функція шифрування F

Функція F(R, K) виконує:

1) Розширення (E): 32 біти → 48 біт;

2) XOR з підключем K;

3) S-перетворення: 8 S-блоків по 6→4 біти;

4) P-перестановка: перестановка 32 біт за таблицею.

Результат — 32 біти.

3. Алгоритм обчислення підключів

1) Ключ 64 біти → PC-1 → 56 біт (видалення бітів парності).

2) Поділ на C і D (по 28 біт).

3) Для кожного з 16 раундів:

– циклічний зсув вліво на 1 або 2 біти;

– PC-2 – формування підключу K_i (48 біт).

У розшифруванні ключі застосовуються у зворотному порядку.

4. Зв’язок IP і IP^{-1}

Матриця IP^{-1} є оберненою до IP.

Застосування IP, а потім IP^{-1} повертає початковий порядок бітів: $IP^{-1}(IP(\text{дані})) = \text{дані}$.

5. Режим ECB (Electronic Code Book)

Кожен блок шифрується незалежно одним ключем:

C_i = E_K(P_i)

Недолік — однакові блоки породжують однакові шифроблоки.

6. Режим CBC (Cipher Block Chaining)

Кожен блок перед шифруванням XOR’иться з попереднім шифроблоком:

C_i = E_K(P_i XOR C_{i-1}), C_0 = IV.

Підвищує стійкість за рахунок залежності від попередніх блоків.

7. Режим CFB (Cipher Feedback)

Шифрується вхідний регістр (IV), результат XOR’иться з відкритим текстом: C_i = P_i XOR E_K(ShiftReg)

Регістр оновлюється частиною шифротексту.

Підходить для потокових даних.

8. Режим OFB (Output Feedback)

Вихід шифру подається назад на вхід: O_i = E_K(O_{i-1}), C_i = P_i XOR O_i.

Помилки передачі не накопичуються, бо зворотного зв’язку по шифротексту немає.