

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту



Лабораторна робота № 3

з дисципліни «Технології захисту інформації»

Виконав:

Студент групи КН-314

Ляшеник Остап

Викладач:

Яковина В. С.

Львів – 2023р.

Лабораторна робота № 3
СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ ЗАБЕЗПЕЧЕННЯ
КОНФІДЕНЦІЙНОСТІ ІНФОРМАЦІЇ

Варіант 7

Мета роботи: ознайомитись з методами криптографічного забезпечення конфіденційності інформації, навчитись створювати комплексні програмні продукти для захисту інформації з використанням алгоритмів симетричного шифрування, хешування та генераторів псевдовипадкових чисел.

Реалізуємо алгоритм RB5-CBC-Pad:

Код програми:

```
import struct
import math

# Клас для генерації псевдовипадкових чисел з використанням лінійного
# конгруентного методу
class LinearCongruentialGenerator:
    def __init__(self, seed=None):
        # Якщо насіння не задано, використовуємо визначене за замовчуванням
        if seed is None:
            seed = 7 # Стандартне насіння
        self.state = seed

    def generate(self):
        # Константи для генератора
        a = 10**3 # Множник
        c = 377 # Приріст
        m = 2**23 - 1 # Модуль
        # Обчислення наступного стану генератора
        self.state = (a * self.state + c) % m
        return self.state

# Функція для додавання доповнення до повідомлення перед обчисленням MD5 хешу
def md5_padding(message):
    original_byte_len = len(message)
    # Додавання байта 0x80 до повідомлення
    message += b'\x80'
    # Додавання нульових байтів, щоб довжина повідомлення стала кратною 64 байтам,
    # мінус 8
    message += b'\x00' * ((56 - (original_byte_len + 1) % 64) % 64)
    # Додавання довжини повідомлення у бітах у вигляді 64-бітного числа з
    # маленьким порядком байтів
```

```

message += struct.pack('<Q', original_byte_len * 8)
return message

# Функція для обчислення MD5 хешу
def md5(message):
    def left_rotate(val, r_bits, max_bits):
        # Функція для циклічного зсуву вліво
        return (val << r_bits % max_bits) & (2 ** max_bits - 1) | \
            ((val & (2 ** max_bits - 1)) >> (max_bits - (r_bits % max_bits)))

    # Ініціалізація початкових значень MD-буфера
    a0 = 0x67452301 # A
    b0 = 0xEFCDAB89 # B
    c0 = 0x98BADCFE # C
    d0 = 0x10325476 # D

    # Попередній розрахунок таблиці синусів
    T = [int(2 ** 32 * abs(math.sin(i + 1))) & 0xFFFFFFFF for i in range(64)]

    # Додавання доповнення до повідомлення
    message = md5_padding(message)

    # Константи для кількості зсувів на кожному раунді
    s = [7, 12, 17, 22] * 4 + [5, 9, 14, 20] * 4 + [4, 11, 16, 23] * 4 + [6, 10,
15, 21] * 4

    # Обробка повідомлення блоками по 64 байти
    for i in range(0, len(message), 64):
        chunk = message[i:i + 64]
        A, B, C, D = a0, b0, c0, d0

        # Виконання 64 раундів алгоритму MD5
        for j in range(64):
            if 0 <= j <= 15:
                F = (B & C) | ((~B) & D)
                g = j
            elif 16 <= j <= 31:
                F = (D & B) | ((~D) & C)
                g = (5 * j + 1) % 16
            elif 32 <= j <= 47:
                F = B ^ C ^ D
                g = (3 * j + 5) % 16
            elif 48 <= j <= 63:
                F = C ^ (B | (~D))
                g = (7 * j) % 16

            dTemp = D
            D = C
            C = B
            B = (B + left_rotate((A + F + T[j] + struct.unpack('<I', chunk[4 * g:4
* g + 4])[0]), s[j],
                                32)) & 0xFFFFFFFF
            A = dTemp

        # Оновлення значень буфера
        a0 = (a0 + A) & 0xFFFFFFFF
        b0 = (b0 + B) & 0xFFFFFFFF
        c0 = (c0 + C) & 0xFFFFFFFF
        d0 = (d0 + D) & 0xFFFFFFFF

```

```

        # Форматування результату як 32-символьного шістнадцяткового рядка
        result = struct.pack('<I', a0) + struct.pack('<I', b0) + struct.pack('<I', c0)
+ struct.pack('<I', d0)
        result_hex = result.hex()
        return result_hex

```

Клас RC5 для шифрування та дешифрування

```
class RC5:
```

```

    def __init__(self, w=16, r=16, b=8):
        self.w = w # Бітова довжина слова: 16 біт
        self.r = r # Кількість раундів: 16
        self.b = b # Довжина ключа: 8 байтів
        self.t = 2 * (r + 1) # Розмір таблиці ключів
        self.S = [0] * self.t
        self.modulo = 2 ** self.w
        self.block_size = 2 * (self.w // 8) # Розмір блоку в байтах: 4 байти (2

```

слова)

```

    def expand_key(self, key):
        # Перетворення ключа в масив цілих чисел
        L = [0] * (self.b // (self.w // 8))
        for i in range(self.b - 1, -1, -1):
            L[i // (self.w // 8)] = (L[i // (self.w // 8)] << 8) + key[i]
        # Ініціалізація таблиці ключів
        self.S[0] = 0xB7E15163
        for i in range(1, self.t):
            self.S[i] = (self.S[i - 1] + 0x9E3779B9) % self.modulo

        # Розширення ключа
        i = j = 0
        A = B = 0
        for k in range(3 * max(self.t, len(L))):
            A = self.S[i] = self.left_rotate((self.S[i] + A + B) % self.modulo, 3,
self.w)
            B = L[j] = self.left_rotate((L[j] + A + B) % self.modulo, (A + B) %
self.w, self.w)
            i = (i + 1) % self.t
            j = (j + 1) % len(L)

```

```

    def encrypt_block(self, block):
        # Шифрування блоку даних
        A = int.from_bytes(block[:self.w // 8], byteorder='little')
        B = int.from_bytes(block[self.w // 8:], byteorder='little')
        A = (A + self.S[0]) % self.modulo
        B = (B + self.S[1]) % self.modulo
        for i in range(1, self.r + 1):
            A = (self.left_rotate(A ^ B, B % self.w, self.w) + self.S[2 * i]) %
self.modulo
            B = (self.left_rotate(B ^ A, A % self.w, self.w) + self.S[2 * i + 1])
% self.modulo
        return A.to_bytes(self.w // 8, byteorder='little') + B.to_bytes(self.w //
8, byteorder='little')

```

```

    def decrypt_block(self, block):
        # Дешифрування блоку даних
        A = int.from_bytes(block[:self.w // 8], byteorder='little')
        B = int.from_bytes(block[self.w // 8:], byteorder='little')

```

```

        for i in range(self.r, 0, -1):
            # Виконання обернених операцій раунду шифрування
            B = self.right_rotate((B - self.S[2 * i + 1]) % self.modulo, A %
self.w, self.w) ^ A
            A = self.right_rotate((A - self.S[2 * i]) % self.modulo, B % self.w,
self.w) ^ B

            # Застосування останніх обернених операцій
            B = (B - self.S[1]) % self.modulo
            A = (A - self.S[0]) % self.modulo
            return A.to_bytes(self.w // 8, byteorder='little') + B.to_bytes(self.w //
8, byteorder='little')

    @staticmethod
    def left_rotate(val, r_bits, max_bits):
        # Функція циклічного зсуву вліво
        return ((val << r_bits) | (val >> (max_bits - r_bits))) & (2 ** max_bits -
1)

    @staticmethod
    def right_rotate(val, r_bits, max_bits):
        # Функція циклічного зсуву вправо
        return ((val >> r_bits) | (val << (max_bits - r_bits))) & (2 ** max_bits -
1)

class RC5_CBC:
    def __init__(self, rc5_instance, iv):
        self.rc5 = rc5_instance
        self.iv = iv

    def encrypt(self, plaintext):
        # Додавання до тексту
        pad_len = self.rc5.block_size - (len(plaintext) % self.rc5.block_size)
        padding = bytes([pad_len] * pad_len)
        padded_plaintext = plaintext + padding

        # Шифрування в режимі CBC
        blocks = [padded_plaintext[i:i+self.rc5.block_size] for i in range(0,
len(padded_plaintext), self.rc5.block_size)]
        ciphertext = b''
        previous_block = self.iv
        for block in blocks:
            block_to_encrypt = bytes([_a ^ _b for _a, _b in zip(block,
previous_block)])
            encrypted_block = self.rc5.encrypt_block(block_to_encrypt)
            ciphertext += encrypted_block
            previous_block = encrypted_block

        return ciphertext

    def decrypt(self, ciphertext):
        # Розшифровування в режимі CBC
        blocks = [ciphertext[i:i+self.rc5.block_size] for i in range(0,
len(ciphertext), self.rc5.block_size)]
        decrypted_text = b''
        previous_block = self.iv
        for block in blocks:
            decrypted_block = self.rc5.decrypt_block(block)
            decrypted_text += bytes([_a ^ _b for _a, _b in zip(decrypted_block,

```

```

previous_block]))
    previous_block = block

    # Видалення додавання
    pad_len = decrypted_text[-1]
    return decrypted_text[:-pad_len]

def read_from_file(file_path):
    with open(file_path, 'rb') as file:
        return file.read()

def write_to_file(file_path, data):
    with open(file_path, 'wb') as file:
        file.write(data)

def hexstr_to_bytes(hexstr):
    """ Конвертація шістнадцяткового рядка у байти """
    return bytes.fromhex(hexstr)

lcg = LinearCongruentialGenerator()
iv = lcg.generate().to_bytes(8, 'little')

# Ініціалізація і використання RC5
key = b"PassKey12" # Визначення ключа
md5_key = md5(key) # Генерація MD5 хешу з ключа
md5_key_bytes = hexstr_to_bytes(md5_key) # Конвертація MD5 хешу у байти

rc5 = RC5()
rc5.expand_key(key)
rc5_cbc = RC5_CBC(rc5, iv)

input_file_path = 'input.txt'
output_file_path = 'output.txt'

# Шифрування
plaintext = read_from_file(input_file_path) # Текст для шифрування

encrypted = rc5_cbc.encrypt(plaintext)
decrypted = rc5_cbc.decrypt(encrypted)

# Шифрування
print("Зашифровані дані (у шістнадцятковому форматі):", encrypted) # Виведення
зашифрованих даних

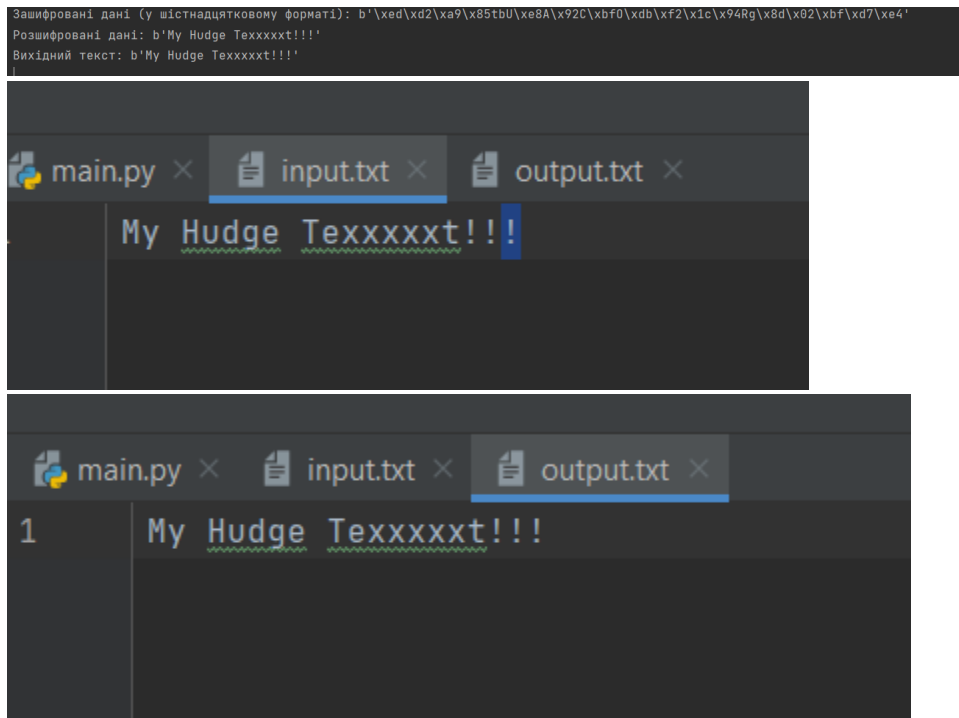
# Дешифрування
print("Розшифровані дані:", decrypted) # Виведення розшифрованих даних
write_to_file(output_file_path, decrypted)

print("Вихідний текст:", plaintext) # Вихідний текст

```

Робота програми:

```
Зашифровані дані (у шістнадцятковому форматі): b'\xed\xda9\x85tbU\xe8A\x92C\xbf0\xdb\xf2\x1c\x94Rg\x8d\x02\xbf\xd7\xe4'  
Розшифровані дані: b'My Hudge Texxxxxt!!!'  
Вихідний текст: b'My Hudge Texxxxxt!!!'
```



The image shows two screenshots of a code editor. The top screenshot displays the results of a decryption process: 'Зашифровані дані (у шістнадцятковому форматі): b'\xed\xda9\x85tbU\xe8A\x92C\xbf0\xdb\xf2\x1c\x94Rg\x8d\x02\xbf\xd7\xe4'', 'Розшифровані дані: b'My Hudge Texxxxxt!!!'', and 'Вихідний текст: b'My Hudge Texxxxxt!!!''. The bottom screenshot shows a code editor with three tabs: 'main.py', 'input.txt', and 'output.txt'. The 'input.txt' tab is active, showing the text 'My Hudge Texxxxxt!!!' with a blue cursor at the end of the line.

Висновок: Під час виконання цієї роботи я ознайомився з роботою шифрувального алгоритму RB5, його різними режимами, такими як CBC-Pad.