МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота № 4

з дисципліни «Технології захисту інформації»

Виконав:

Студент групи КН-314

Ляшеник Остап

Викладач:

Яковина В. С.

Львів – 2023р.

Лабораторна робота № 4
СТВОРЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ ШИФРУВАННЯ З ВІДКРИТИМ КЛЮЧЕМ RSA З ВИКОРИСТАННЯМ MICROSOFT CRYPTOAPI
Варіант 7

Мета роботи: ознайомитись з методами і засобами криптографії з відкритим ключем, навчитись створювати програмні засоби з використанням криптографічних інтерфейсів.

Реалізуємо алгоритм RSA:

Код програми:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa,
padding
from cryptography.hazmat.primitives import serialization,
hashes  # Додано імпорт hashes
import time
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 hexstr_to_bytes(hexstr):
    return bytes.fromhex(hexstr)

# Генеруємо пару ключів RSA
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)
```

```python
# Зберігаємо приватний ключ в файл
with open('private_key.pem', 'wb') as f:
    f.write(private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    ))

# Отримуємо публічний ключ з приватного ключа
public_key = private_key.public_key()

# Зберігаємо публічний ключ в файл
with open('public_key.pem', 'wb') as f:
    f.write(public_key.public_bytes(
        encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo
    ))

# Повідомлення, яке потрібно зашифрувати
message = b"Hello, RSA encryption!"

# Виміряємо час шифрування
start_time = time.time()

# Зашифруємо повідомлення з використанням публічного ключа
ciphertext = public_key.encrypt(
    message,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

end_time = time.time()
print(f'{start_time} _ {end_time}')
```

```
encryption_time = end_time - start_time

print("Зашифроване повідомлення:", ciphertext)
print("Час шифрування RSA: {:.6f}
секунд".format(encryption_time))


# Ініціалізація і використання RC5
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)

# Шифрування
plaintext = b"Hello, RSA encryption!"  # Текст для
шифрування




# Виміряємо час шифрування
start_time = time.time()

encrypted = rc5_cbc.encrypt(plaintext)

end_time = time.time()

decrypted = rc5_cbc.decrypt(encrypted)

encryption_time = end_time - start_time
print("Час шифрування RC5: {:.6f}
```

```
секунд".format(encryption_time))
```

Робота програми:

Висновок: Під час виконання цієї роботи я ознайомився з роботою алгоритму асиметричного шифрування RSA, а також загальними методами і засобами криптографії з відкритим ключем.