**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**Кафедра систем штучного інтелекту**



**Лабораторна робота № 4**

з дисципліни «Технології захисту інформації»

**Виконав:**

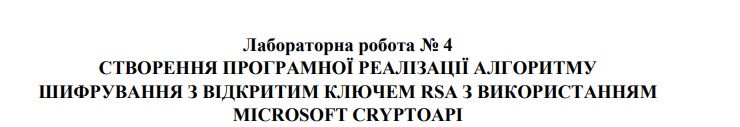
Студент групи КН-314

Ляшеник Остап

**Викладач:**

Яковина В. С.

Львів – 2023р.



**Варіант 7**

**Мета роботи**: ознайомитись з методами і засобами криптографії з

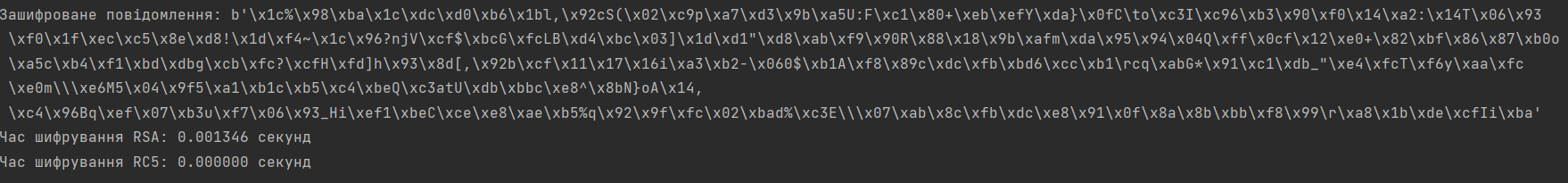
відкритим ключем, навчитись створювати програмні засоби з використанням криптографічних інтерфейсів.

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()  
)  
  
# 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.