Міністерство освіти і науки України

Національний технічний університет «ХПІ»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

Кафедра комп’ютерної інженерії та програмування

ЗВІТ

з лабораторної роботи № 3

з дисципліни «Сучасні технології безпечного програмування»

«**АСИМЕТРИЧНЕ ШИФРУВАННЯ. АЛГОРИТМ RSA**»

Виконав:

студент гр. КН-Н922б

Кулик Д.І.

Перевірив:

Бульба С. С.

Харків – 2022

Мета роботи: Дослідити і реалізувати механізм асиметричного алгоритму шифрування RSA.

Індивідуальне завдання

Розробити додаток обміну таємними посиланнями між двома клієнтами за допомогою алгоритму шифрування RSA

- Реалізувати алгоритм генерації ключів (public / private keys) для алгоритму RSA. Створити ключі заданої довжини (напр. 1024 біт)
- Реалізувати та продемонструвати роботу алгоритму шифрування та дешифрування повідомлення RSA
- Підтвердити роботу реалізованого алгоритму шляхом порівняння результату кодування з існуючим алгоритмом (наприклад, використовуючи утиліту openssl або вбудовані системи шифрування обраної мови програмування)

Хід роботи

Алгоритм RSA є асиметричним алгоритмом шифрування. Асиметричний насправді означає, що він працює з двома різними ключами, тобто відкритим ключем і закритим ключем. Як видно з назви, відкритий ключ надається кожному, а закритий ключ залишається закритим.

Алгоритм створення ключів можна представити як:

- Select two large prime numbers, p and **q**.
- Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
- Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n),
  gcd (e,d(n)) =1
- If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C = m^e \bmod n$
  Here, **m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  $D_e \bmod \{(p - 1) \times (q - 1)\} = 1$
  Or
  $D_e \bmod \varphi(n) = 1$
- The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

Рисунок 1 – Алгоритм створення ключів

Важливі фрагменти програми

```
def gen_pq(self, bits):
    """
    Функція, що генерує пару ключів (p, q), що є дуже великими простими числами
    """
    assert bits >= 512, 'довжина ключа має бути не меншою ніж 512 бітів'
    l = bits >> 1

    while True:
        p = prime.randprime_bits(l)
        if prime.is_probable_prime(p, None, l // 8):
            break

    while True:
        q = prime.randprime_bits(bits - l)
        if p != q and prime.is_probable_prime(q, None, l // 8):
            break

    self.p = p
    self.q = q
    # перемножуємо N = p * q, де N - модуль для шифрування та дешифрування
    self.N = p * q
    # для розрахованого раніше N необхідна функція Ейлера
    self.phi = (p - 1) * (q - 1)
```

Рисунок 2 – Функція для генерації пар ключів

```
def encrypt_data(self, data):
    """
    Функція, що шифрує усе повідомлення по всім блокам
    """
    bs = self.key.block_size - 1
    data_stream = (data[i:i + bs] for i in range(0, len(data), bs))
    return b''.join(self.encrypt_block(block) for block in data_stream)

def decrypt_data(self, data):
    """
    Функція, що дешифрує усе повідомлення по всім блокам
    """
    useCRT = self.key._can_crt
    bs = self.key.block_size
    data_stream = (data[i:i + bs] for i in range(0, len(data), bs))
    return b''.join(self.decrypt_block(block, useCRT)[:bs - 1] for block in data_stream).rstrip(b'\x00')
```

Рисунок 3 – Функції шифрування та дешифрування даних

```
def my_rsa(text, bits):
    key = RSAKey(bits=bits)
    public_key = (key.e, key.N)
    private_key = (key.d, key.N)

    print(f'Розроблений RSA: public_key            : {public_key}')
    print(f'Розроблений RSA: private_key           : {private_key}')
    cipher = RSA(key)

    encrypted = cipher.encrypt_data(text.encode())

    print(f'Розроблений RSA: Заданий текст         : {text}')
    print(f'Розроблений RSA: Зашифрований текст    : {encrypted.hex()}')

    decrypted = cipher.decrypt_data(encrypted)
    print(f'Розроблений RSA: Розшифрований текст   : {decrypted.decode()}')
```

Рисунок 4 – Виклик власної реалізації алгоритму

```
def rsa_lib(text, bits):
    public_key, private_key = rsa.newkeys(bits)
    print(f'RSA_LIB: public_key                    : {public_key}')
    print(f'RSA_LIB: private_key                   : {private_key}')

    encrypted = rsa.encrypt(text.encode(), public_key)

    print(f'RSA_LIB: Заданий текст                 : {text}')
    print(f'RSA_LIB: Зашифрований текст            : {encrypted.hex()}')

    decrypted = rsa.decrypt(encrypted, private_key)

    print(f'RSA_LIB: Розшифрований текст           : {decrypted.decode()}')
```

Рисунок 5 – Виклик реалізації алгоритму з бібліотеки RSA

Результати роботи програми

```
C:\Users\Daniil\PycharmProjects\stbp\Scripts\python.exe C:/Users/Daniil/PycharmProjects/stbp/LABS/kulyk03/main.py
Розроблений RSA: public_key            : (65537,
 4640698912356550619970067581548666383770059148821021128278935824699276150285015331053408928968103243142909605575129420845215875556760397845569439663712516457
 0111090472522310548297942458861020273614369315483585734676889259005532756623558669348977)
Розроблений RSA: private_key           :
 (1955428238471796776942390043249865300331266054208958274828307868823267938570894279064343616177947892936683839021578634307957892541616771999105849769037660098
 8823214019510296219732845124428358006726603636650330179919312548984777311059454640320193,
 4640698912356550619970067581548666383770059148821021128278935824699276150285015331053408928968103243142909605575129420845215875556760397845569439663712516457
 0111090472522310548297942458861020273614369315483585734676889259005532756623558669348977)
Розроблений RSA: Заданий текст         : Daniil Kulyk
Розроблений RSA: Зашифрований текст    :
 b7eed5d4ecc3f8ce48d9829fc2d4e0da2b6b61ee83ce695594472fb2db3eb43b5f1f3dab58d5331b5d2da92dcaff67feb2bf5ebcc20ece8851a9843031af0ec12927531776a9b102daa57df16bc354
 804fcaff073a2c88404e8516723f46b6fc01
Розроблений RSA: Розшифрований текст   : Daniil Kulyk


RSA_LIB: public_key                    : PublicKey
 (945905240153767136233268144391498168545332554834696502169191041752704418286319106131940549327543590808569732258646983326134076597104512208286145936820974099
 619215266082435876482884938883119714929173168245291482241037796409756655659832939336307, 65537)
RSA_LIB: private_key                   : PrivateKey
 (945905240153767136233268144391498168545332554834696502169191041752704418286319106131940549327543590808569732258646983326134076597104512208286145936820974099
 619215266082435876482884938883119714929173168245291482241037796409756655659832939336307, 65537,
 3808041847539098848341936115724749024047492885966497171556820795215428440738246074154312747525506217845385631919700628359052363846072516304289433467305492901
 2500112646094273247901845054219877191531808493859890881993367196167652879638027588 3393,
 5413282509027371524754643408979416969062069447695245035837952764127412827321436558894154732208404531474447117347354146019933316187579515152328896159411726925
 1747378302492699804866266907115069155042575115956456900869829589073744073621937364304736988656676744149452572424542763431271477495300584676955287)
RSA_LIB: Заданий текст                 : Daniil Kulyk
RSA_LIB: Зашифрований текст            :
 407b94ab5b23ce5cb693fe408e9e389a084a779d8d6fdb22ee063dabdda55b53b78c149464e6a927d852456a595abc817b426cc3a1a3676a96ff9b8cb6b6df4e7b0e97dc1c9aa643be9821a0c02ae1
 7b79a5ff6e72919b6e3add95896342d816
RSA_LIB: Розшифрований текст           : Daniil Kulyk
```

Рисунок 6 – Результат виконання програми

При порівнянні можемо побачити що результат виконання реалізацій алгоритму однаковий.

Висновки: в результаті виконання лабораторної роботи було досліджено і реалізовано механізм асиметричного алгоритму шифрування RSA. В результаті порівняння власної реалізації алгоритму з вже реалізованими була виявлена ідентичність роботи, що доводить коректність першого.