МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ "КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

Виконали:

Студенти групи ФБ-03

Митрофанова М.М. та Мец Є.В.

Київ – 2022

Тема: «Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Мета: Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

ХІД РОБОТИ

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.

```
mytroffanova
def generate_prime_number(bits):
    mytroffanova
    def get_random_number(bits):...

    mytroffanova
    def test_miller_rabin(p):...

    num = get_random_number(bits)
    while not test_miller_rabin(num):
        num = get_random_number(bits)

    return num
```

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p1, q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p і q1 – абонента B.

```
mytroffanova
def get_pair(bits):
    pair = (generate_prime_number(bits), generate_prime_number(bits))
    return pair
```

```
s_numbers = get_pair(256)
r_numbers = get_pair(256)

while s_numbers[0] * s_numbers[1] > r_numbers[0] * r_numbers[1]:
    s_numbers = get_pair(256)
    r_numbers = get_pair(256)
```

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p,q) та відкритий ключ (n,e) . За допомогою

цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n) , (,) 1 n1 e та секретні d і d1

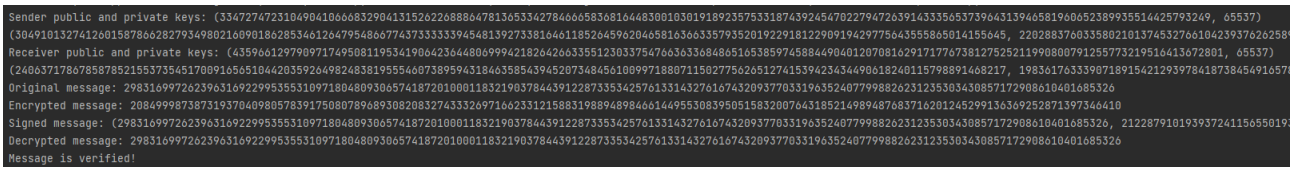
```
mytroffanova
def generate_keys(pair):
    n = pair[0] * pair[1]
    f = (pair[0] - 1) * (pair[1] - 1)
    e = 2**16 + 1
    d = pow(e, -1, f)
    open_key = (n, e)
    secret_key = (d, pair[0], pair[1])
    return open_key, secret_key
```

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання.

```
mytroffanova
def encrypt(message, key):
    encrypted_message = pow(message, key[0][1], key[0][0])
    return encrypted_message


mytroffanova
def sign(message, key):
    signed_message = (message, pow(message, key[1][0], key[0][0]))
    return signed_message


mytroffanova
def decrypt(encrypted, key):
    decrypted_message = pow(encrypted, key[1][0], key[0][0])
    return decrypted_message


mytroffanova
def verify(signed, message, key):
    if message == pow(signed[1], key[0][1], key[0][0]):
        print('Message is verified!')
    else:
        print('Fake sign!')
```

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

```
s_keys = generate_keys(s_numbers)
r_keys = generate_keys(r_numbers)
print(f'Sender public and private keys: {s_keys[0]}\n{s_keys[1]}')
print(f'Receiver public and private keys: {r_keys[0]}\n{r_keys[1]}')
msg = random.randint(0, r_numbers[0] * r_numbers[1])
encrypted = encrypt(msg, r_keys)
signed = sign(msg, s_keys)
decrypted = decrypt(encrypted, r_keys)
print(f'Original message: {msg}')
print(f'Encrypted message: {encrypted}')
print(f'Signed message: {signed}')
print(f'Decrypted message: {decrypted}')
print(verify(signed, msg, s_keys))
```



Sender public and private keys:
(3347274723104904106668329041315262268886478136533427846665836816448300103019189
23575331874392454702279472639143335653739643139465819606523899355144257932 49,
65537)

(3049101327412601587866282793498021609018628534612647954866774373333339454813927
33816461185264596204658163663357935201922918122909194297756435558650141556 45,
22028837603358021013745327661042393762625899284585441312364061924189482541734 7,
15194967539252520086073212040640838619133289423895751018830398075206738957306 7)

Receiver public and private keys:
(4359661297909717495081195341906423644806999421826426633551230337547663633684865
16538597458844904012070816291717767381275252119908007912557732195164136728 01,
65537)

(2406371786785878521553735451700916565104420359264982483819555460738959431846358
54394520734845610099718807115027756265127415394234344906182401157988914682 17,
19836176333907189154212939784187384549165780114270171342200692952606726987 1983,
21978335060762097849885965022203676559614723990195360812416542459494989157 1247)

Original message:
298316997262396316922995355310971804809306574187201000118321903784439122873353425
76133143276167432093770331963524077998826231235303430857172908610401685326

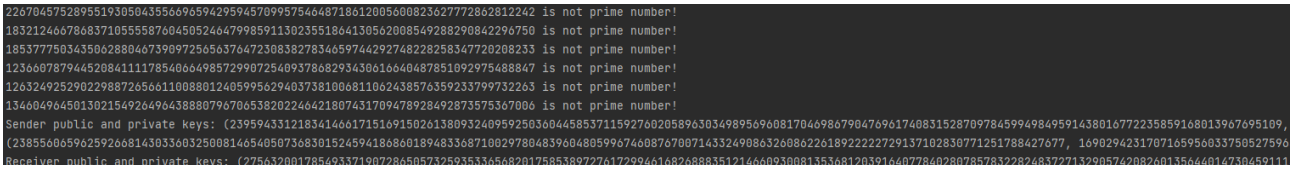
Encrypted message:
208499987387319370409805783917508078968930820832743332697166233121588319889489846
61449553083950515832007643185214989487683716201245299136369252871397346410

Signed message:
(298316997262396316922995355310971804809306574187201000118321903784439122873353425
76133143276167432093770331963524077998826231235303430857172908610401685326,
21228791019393724115655019349072034087588445068052655961813378771125393725704075
26604956088334178497135688501813056208250531518576455012117631523148286371 4)

Decrypted message:
298316997262396316922995355310971804809306574187201000118321903784439122873353425
76133143276167432093770331963524077998826231235303430857172908610401685326

Message is verified!

Декілька кандидатів, які не пройшли провірку:




```
1560354608790388108350417802427020875992787720238302920703631320056142908390 75
is not prime number!

1449982012038555289021284908224937588567328687630946476477739573894443492286 03
is not prime number!

2267045752895519305043556696594295945709957546487186120056008236277728628122 42
is not prime number!

1832124667868371055558760450524647998591130235518641305620085492882908422967 50
is not prime number!

1853777503435062880467390972565637647230838278346597442927482282583477202082 33
is not prime number!

1236607879445208411117854066498572990725409378682934306166404878510929754888 47
is not prime number!

1263249252902298872656611008801240599562940373810068110624385763592337997322 63
is not prime number!

1346049645013021549264964388807967065382022464218074317094789284928735753670 06
is not prime number!
```

Опис кроків протоколу конфіденційного розсилання ключів з підтвердженням справжності:

0 < msg < n

```
msg = random.randint(0, r_numbers[0] * r_numbers[1])
```

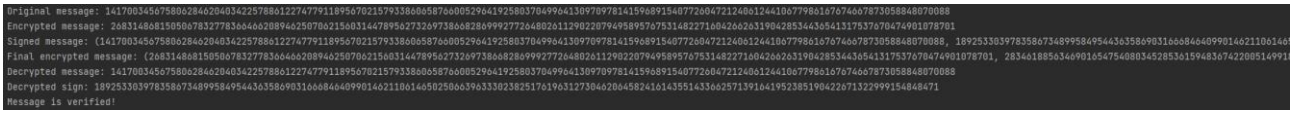
Шифрування та підпис повідомлення:

```
encrypted = encrypt(msg, r_keys)
signed = sign(msg, s_keys)
```

Шифруємо підпис та отримуємо фінальне повідомлення: (k1, s1).

Отримувач за допомогою секретного ключа розшифровує повідомлення і підпис та проводить автентифікацію.

```
signed = sign(msg, s_keys)
s1 = encrypt(signed[1], r_keys)
final_message = (encrypted, s1)
decrypted = decrypt(final_message[0], r_keys)
decrypted_sign = decrypt(final_message[1], r_keys)
print(f'Original message: {msg}')
print(f'Encrypted message: {encrypted}')
print(f'Signed message: {signed}')
print(f'Decrypted message: {decrypted}')
print(verify(decrypted_sign, decrypted, s_keys))
```




```
Signed message:
(14170034567580628462040342257886122747791189567021579338606587660052964192580370499641309709781415968915407726047212406124410677986167674667873058848070088,
18925330397835867348995849544363586903166684640990146211061465025066396333023825176196312730462064582416143551433662571391641952385190422671322999154848471)

Final encrypted message:
(26831486815050678327783664662089462507062156031447895627326973866828699927726480261129022079495895767531482271604266263190428534436541317537670474901078701,
28346188563469016547540803452853615948367422005149918863875227777987826107163403161422201819685753956813780757071881766762539202943125310466211365075724287)

Decrypted message:
14170034567580628462040342257886122747791189567021579338606587660052964192580370499641309709781415968915407726047212406124410677986167674667873058848070088

Decrypted sign:
18925330397835867348995849544363586903166684640990146211061465025066396333023825176196312730462064582416143551433662571391641952385190422671322999154848471

Message is verified!
```

Висновок

В ході лабораторної роботи ми провели тести перевірки чисел на простоту і методи генерації ключів для асиметричної криптосистеми типу RSA. Ознайомились з системою захисту інформації на основі криптосхеми RSA. Реалізували протокол конфіденційного розсилання ключів з підтвердженням справжності за допомогою системи RSA. Було вибране повідомлення розміром [0, n], проведена перевірка pq ≤ p1q1. За допомогою протоколу ми зашифрували та підписали довільне повідомлення, після чого розшифрували та провели автентифікацію.