НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра Інформаційної Безпеки

Лабораторна робота №4 дисципліни

”КРИПТОГРАФІЯ”

Підготував:

студент групи ФБ-06

Жак Костянтин

Київ 2022

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

Мета роботи: 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.

Порядок 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.

```
'''
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.
'''
def prime_test(num):
    if num % 2 == 0 or num % 3 == 0 or num % 5 == 0 or num % 7 == 0 or num % 11 == 0 or num % 13 == 0:
        return False

    d = num - 1
    s = 0
    while d % 2 == 0:
        d = d // 2
        s += 1

    x = randint(2, num - 2)

    if gcd(x, num) > 1:
        return False

    if pow(x, d, num) == 1 or pow(x, d, num) == -1:
        return True

    for i in range(1, s - 1):
        x = (x ** 2) % num
        if x == - 1:
            return True
        if x == 1:
            return False
    return False
```

```
def choose_number(h, l=0):
    while True:
        x = randint(l, h)
        if miller_rabin_test(x):
            return x
```

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і 1 1 p , q 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.

```
a_nums = (choose_number(2 ** 258, 2 ** 256), choose_number(2 ** 258, 2 ** 256))
b_nums = (choose_number(2 ** 258, 2 ** 256), choose_number(2 ** 258, 2 ** 256))

if a_nums[0] * a_nums[1] > b_nums[0] * b_nums[1]:
    a_nums, b_nums = b_nums, a_nums
```

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 .

```
def make_keys(p, q):
    e = 2 ** 16 + 1
    n = p * q
    f = (p - 1) * (q - 1)
    d = pow(e, -1, f)
    return (n, e), (d, p, q)
```

```
a_open, a_secret = make_keys(a_nums[0], a_nums[1])
b_open, b_secret = make_keys(b_nums[0], b_nums[1])
```

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. За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A и B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

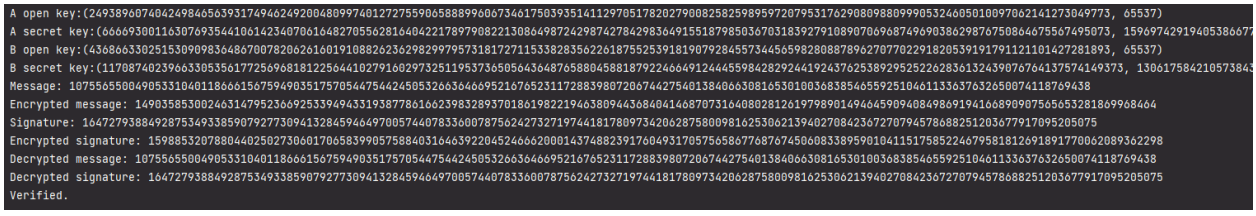
```
def encrypt(message, open):
    return pow(message, open[1], open[0])


def decrypt(message, secret):
    return pow(message, secret[0], secret[1] * secret[2])


def sign(message, secret):
    return pow(message, secret[0], secret[1] * secret[2])


def verify(message, signed, open):
    if message == pow(signed, open[1], open[0]):
        return True
    return False
```

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 $0 < k < n$.

```python
message = randint(0, a_nums[0] * a_nums[1])

print(f"Message: {message}")

encrypted_message = encrypt(message, b_open)
print(f"Encrypted message: {encrypted_message}")

signed = sign(message, a_secret)
print(f"Signature: {signed}")

encrypted_sign = encrypt(signed, b_open)
print(f"Encrypted signature: {encrypted_sign}")

decrypted_message = decrypt(encrypted_message, b_secret)
print(f"Decrypted message: {decrypted_message}")

decrypted_sign = decrypt(encrypted_sign, b_secret)
print(f"Decrypted signature: {decrypted_sign}")


if verify(decrypted_message, decrypted_sign, a_open):
    print("Verified.")
else:
    print("Error.")
```

Вибираємо повідомлення від 0 до n = pq. Запускаємо протокол роботи.


A open key:(249389607404249846563931749462492004809974012727559065888996067346175039351411297051782027900825825989597207953176290809880999053246050100970621412730497 73, 65537)
A secret key:(666693001163076935441061423407061648270556281640422178979082213086498724298742784298364915518798503670318392791089070696874969038629876750864675567495073, 15969742919405386677
B open key:(43686633025153090983648670078206261601910882623629829977573181727115338283562261875525391819079284557344565982808878962707702291820539191791121101427281893, 65537)
B secret key:(11708740239663305356177256968181225644102791602973251195373650564364876588045881879224664912444559842829244192437625389295252262836132439076764137574149373, 13061758421057384
Message: 10755655004905331040118666156759490351757054475442450532663646695216765231172883980720674427540138406630816530100368385465592510461133637632650074118769438
Encrypted message: 14903585300246314795236692533949433193877861662398328937018619822194638094436840414687073164080281261979890149464590940849869194166890907565653281869968464
Signature: 16472793888492875349338590792773094132845946497005744078336007875624273271974418178097342062875800981625306213940270842367270794578688251203677917095205075
Encrypted signature: 15988532078804402502730601706583990575884031646392204524666200014374882391760493170575658677087674506083389590104115175852246795818126918917700620893622 98
Decrypted message: 10755655004905331040118666156759490351757054475442450532663646695216765231172883980720674427540138406630816530100368385465592510461133637632650074118769438
Decrypted signature: 16472793884928753493385907927730941328459464970057440783360078756242732719744181780973420628758009816253062139402708423672707945786882512036779170952050 75
Verified.

```
A open
key:(24938960740424984656393174946249200480997401272755906588899606734617503935141129705178202790082582598959720795317629080988099905324605010097062141273049773, 65537)
A secret
key:(66669300116307693544106142340706164827055628164042217897908221308649872429874278429836491551879850367031839279108907069687496903862987675086467556749 5073,
15969742919405386677226016525673911124044829428177030035315083543765634398555 1,
15616382095995290787795577981993043606647158588728691655423790236749710547712 3)
```

B open
key:(436866330251530909836486700782062616019108826236298299795731817271153382
835622618755253918190792845573445659828088789627077022918205391917911211014427
281893, 65537)
B secret
key:(117087402396633053561772569681812256441027916029732511953736505643648765
880458818792246649124445598428292441924376253892952522628361324390767641375 74
149373,
130617584210573843191988391216880670284601171783486495102656304068248460502 96
3,
334462111584640233011531691739273514361219592113080902344496786575125429231 11
1)
Message:
107556550049053310401186661567594903517570544754424505326636466952167652311 72
883980720674427540138406630816530100368385465592510461133637632650074118769 43
8
Encrypted message:
149035853002463147952366925339494331938778616623983289370186198221946380944 36
840414687073164080281261979890149464590940849869194166890907565653281869968 46
4
Signature:
164727938849287534933859079277309413284594649700574407833600787562427327197 44
181780973420628758009816253062139402708423672707945786882512036779170952050 75
Encrypted signature:
159885320788044025027306017065839905758840316463922045246662000143748823917 60
493170575658677687674506083389590104115175852246795818126918917700620893622 98
Decrypted message:
107556550049053310401186661567594903517570544754424505326636466952167652311 72
883980720674427540138406630816530100368385465592510461133637632650074118769 43
8
Decrypted signature:
164727938849287534933859079277309413284594649700574407833600787562427327197 44
181780973420628758009816253062139402708423672707945786882512036779170952050 75
Verified.

```
251474549524242660169553523054287960943341244480002908800504236696988337265944 not prime
366276646689594808977967595754736057789747160841241252116634503409621559690698 not prime
343692618175006772824743723798412610180775230042737045103501494050461270655154 not prime
366083273306070864161757916797096759272022880822186694149401185090866117723378 not prime
346386643608103352914432883706718967643430865176930211993867479850030250555876 not prime
220477575329821555125687107771027979325886622651676692428614157308420665553568 not prime
299450282462871002641616094068022652761527277561324478708376312274895357361461 not prime
230099917870948497408057838304629141041625360149113330871412422929445913141998 not prime
263700788745771521751949135844103526217325821882481084404359489423838769851647 not prime
267805946646132660090793992538690476594799392889268523409354287915852880149485 not prime
326457720214222006590021133195512130095609093791794758243308444499356690409163 not prime
254238548721111175074301682765233218892860444371160637255013309610502585797791 not prime
235274834650167994914685872044565799075228509643034740959637406402092960314390 not prime
414011016872179091521910728327383820946286993827742291407664826321331301452677 not prime
317876957759814568289762592689861187667244935414879261277060314721641934879405 not prime
321618643191633073441273391182399845358476157903541829955974773903537388401953 not prime
```

Кандидати які не пройшли перевірку:
366276646689594808977967595754736057789747160841241252116634503409621559690698 not prime

```
343692618175006772824743723798412610180775230042737045103501494050461270655154 not prime
366083273306070864161757916797096759272022880822186694149401185090866117723378 not prime
346386643608103352914432883706718967643430865176930211993867479850030250555876 not prime
220477575329821555125687107771027979325886622651676692428614157308420665553568 not prime
299450282462871002641616094068022652761527277561324478708376312274895357361461 not prime
230099917870948497408057838304629141041625360149113330871412422929445913141998 not prime
263700788745771521751949135844103526217325821882481084404359489423838769851647 not prime
267805946646132660090793992538690476594799392889268523409354287915852880149485 not prime
326457720214222006590021133195512130095609093791794758243308444499356690409163 not prime
254238548721111175074301682765233218892860444371160637255013309610502585797791 not prime
235274834650167994914685872044565799075228509643034740959637406402092960314390 not prime
414011016872179091521910728327383820946286993827742291407664826321331301452677 not prime
317876957759814568289762592689861187667244935414879261277060314721641934879405 not prime
321618643191633073441273391182399845358476157903541829955974773903537388401953 not prime
```

Висновок

В ході комп’ютерного практикуму було проведене 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. Був реалізований протокол конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Програма перевірена для ключа довжиною $0 < k < pq$.