КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

Мета роботи

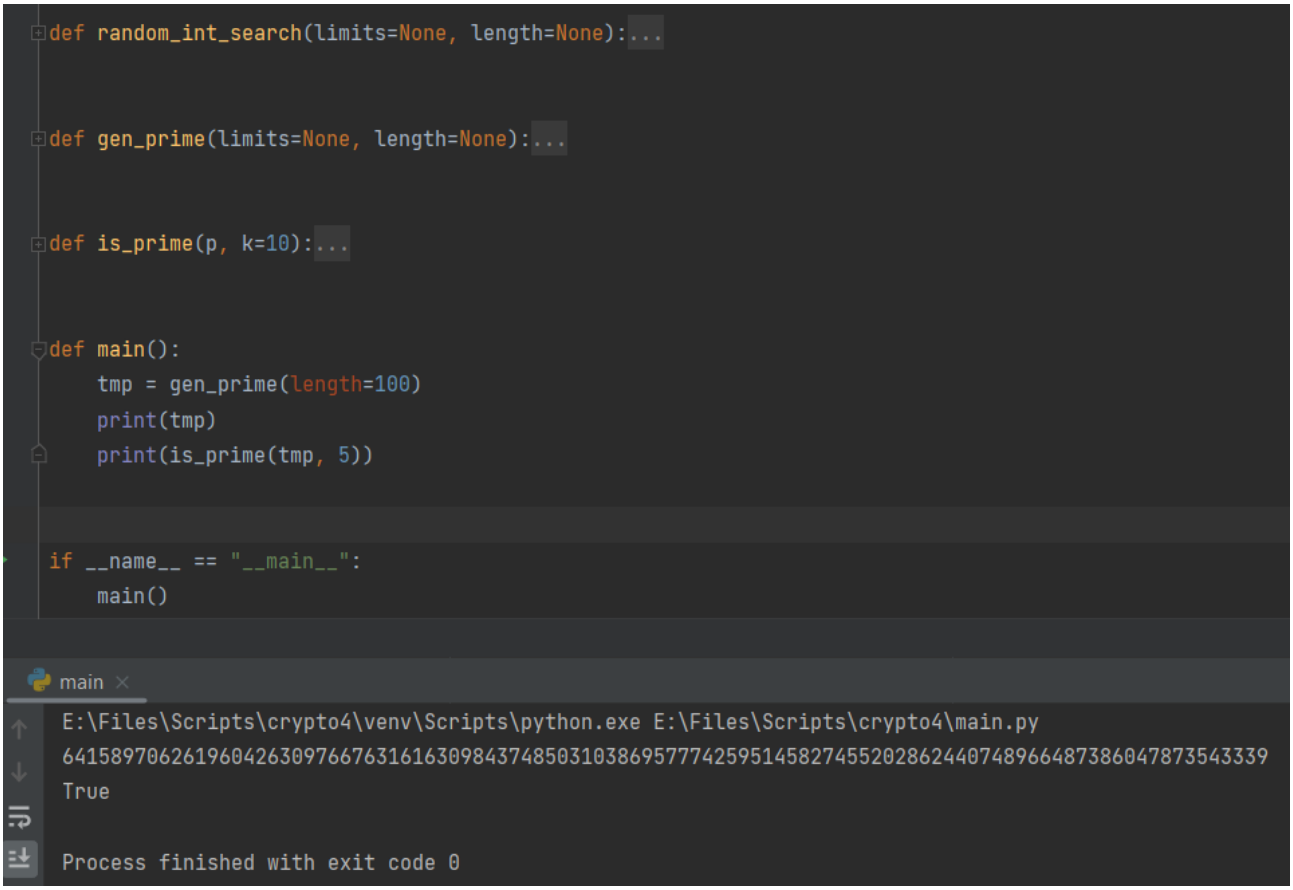
Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу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

Виконали: Бондаренко Олексій, Кригін Дмитро. ФБ-03

Варіант: 2

Порядок виконання 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.



Функція random_int_search(limits=None, length=None) генерує випадкове число або в межах limits або довжини length

Функція is_prime(p, k=10) перевіряє чи є число простим, за k ітерацій

Функція gen_prime(limits=None, length=None) генерує просте число заданої довжини (в бітах) або в заданих межах, використовуючи раніше описані функції.

Код у lab5.py

Все працює правильно!

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.

```python
def main():
    p = gen_prime(length=256)
    q = gen_prime(length=256)
    p1 = gen_prime(length=256)
    q1 = gen_prime(length=256)

    print(f"p = {p}")
    print(f"q = {q}")
    print(f"p1 = {p1}")
    print(f"q1 = {q1}")

    print(p * q <= p1 * q1)
```

OUTPUT:

```
Not prime: 819236211333330749654533525959655826682566539553020922941096746238413925087 32
Not prime: 294050801668423223927108555795676556410969784790095369624966040294823811791 63
Not prime: 789498663700875791018219608534976428059150139382923319684827283464179756693 74
Not prime: 193613935553644899070210845989525332059667390297556888422310982528790394736 45
Not prime: 171827393941911382894505489727891504423481405368212988706869454653629868751 64
p = 525429680973750793469575398782987553044672299481128968436056089288170954215 01
q = 200027281863980888092482061673078009701808561154237689993148676432875723750 41
p1 = 136598391740276971897810921444091232736074667959653480688906567244099924629 47
q1 = 106049897360078569880421646765904286126855568556028684579622345977201440324293
True
```

Повний вивід у файлі task2_output.txt

p = 525429680973750793469575398782987553044672299481128968436056089288170954215 01

q = 200027281863980888092482061673078009701808561154237689993148676432875723750 41

p1 = 136598391740276971897810921444091232736074667959653480688906567244099924629 47

q1 = 106049897360078569880421646765904286126855568556028684579622345977201440324293

Готово!

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 .

```python
def key_gen(p: int, q: int):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = pow(2, 16) + 1
    d = pow(e, -1, phi)
    private_key = (d, p, q)
    public_key = (n, e)
    return public_key, private_key
```

```python
def main():
    p = 525429680973750793469575398782987553044672299481128968436056089288170954215 01
    q = 200027281863980888092482061673078009701808561154237689993148676432875723750 41
    p1 = 136598391740276971897810921444091232736074667959653480688906567244099924629 47
    q1 = 106049897360078569880421646765904286126855568556028684579622345977201440324293

    a_public_key, a_private_key = key_gen(p, q)
    b_public_key, b_private_key = key_gen(p1, q1)

    print(f"A public key: {a_public_key}")
    print(f"A private key: {a_private_key}")
    print(f"B public key: {b_public_key}")
    print(f"B private key: {b_private_key}")
```

OUTPUT:

A public key:
(105100270895838005987162458636280187504368864370648790857178564061609721598237308743198130922736465999942880504641858775578665275454207942918716954715654 1, 65537)

A private key:
(944406206121107187174267541860405609370627648929231929075063801759036346631638294441540319186517324121972878817836895298075809183567961162694373723934 73,
525429680973750793469575398782987553044672299481128968436056089288170954215 01,
200027281863980888092482061673078009701808561154237689993148676432875723750 41)

B public key:
(144862454236081771670392754507866871225033648167525795725699299861661330961319466785448371352726162505673569594343397639603731949755127702504472946647147 1, 65537)

B private key:
(670433968511618813271080860731817152314951185218918307239602890596769737611947091790523361426505119708286337147561995418510865719760833354619361781584089,
136598391740276971897810921444091232736074667959653480688906567244099924629 47,
106049897360078569880421646765904286126855568556028684579622345977201440324293)

Вивід у файлі task3_output.txt

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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.

```python
def encrypt(m: int, public_key: tuple):
    return pow(m, public_key[1], public_key[0])


def decrypt(c: int, private_key: tuple):
    return pow(c, private_key[0], private_key[1] * private_key[2])


def digital_signature(m: int, private_key: tuple):
    return pow(m, private_key[0], private_key[1] * private_key[2])


def sign(m, private_key):
    return m, digital_signature(m, private_key)


def signature_verification(m: int, s: int, public_key: tuple):
    return True if m == pow(s, public_key[1], public_key[0]) else False
```

За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A і B, перевірити правильність розшифрування.

```python
def main():
    a_public_key = (10510027089583800598716245863628018750436886437064879085717856400
    a_private_key = (94440620612110718717426754186040560937062764892923192907506380l
    b_public_key = (14486245423608177167039275450786687122503364816752579572569929980
    b_private_key = (67043396851161881327108086073181715231495118521891830723960289O

    m = randint(10000, 100000000)
    print(f"Plaintext: {m}")

    a_enc = encrypt(m, a_public_key)
    b_enc = encrypt(m, b_public_key)

    print(f"A cryptogram: {a_enc}")
    print(f"B cryptogram: {b_enc}")

    a_dec = decrypt(a_enc, a_private_key)
    b_dec = decrypt(b_enc, b_private_key)

    print(f"A decrypted: {a_dec}")
    print(f"B decrypted: {b_dec}")
```

OUTPUT:

Plaintext: 93144998

A cryptogram:
125435842164605021414015629800943542238516696610272632951967406374113712893582055507652633351627765221411075649848548572828371112937448130125454889110 29

B cryptogram:
691522380101483013893531719620921263047270889240593443031518048518185741637104253493025291101681042961143231257098675250327663870762733548945905306800626

A decrypted: 93144998

B decrypted: 93144998

Скласти для A і B повідомлення з цифровим підписом і перевірити його.

```
def main():
    a_public_key = (1051002708958380059871624586362801875043688643706487908571785640616097215982373 0
    a_private_key = (9444062061211071871742675418604056093706276489292319290750638017590363466316382
    b_public_key = (1448624542360817716703927545078668712250336481675257957256992998616613309613194 6
    b_private_key = (6704339685116188132710808607318171523149511852189183072396028905967697376119470

    m = randint(10000, 100000000)
    print(f"Message: {m}")

    a_signed = sign(m, a_private_key)
    b_signed = sign(m, b_private_key)

    print(f"A signed message: {a_signed}")
    print(f"B signed message: {b_signed}")

    print(f"A signature verify: {signature_verification(a_signed[0], a_signed[1], a_public_key)}")
    print(f"B signature verify: {signature_verification(b_signed[0], b_signed[1], b_public_key)}")
```

OUTPUT:

Message: 14785427

A signed message: (14785427, 609870299619705128880326943157727013746555234043085854905076961021907728429463475633287993725169995527557697532268764686321855441501175539431379499810848)

B signed message: (14785427, 627947742081096348663903911831364638223766056324074719186105598218130075077186739903840321456221777938086888591143869478532861276340085055427035076664325)

A signature verify: True

B signature verify: True

SUCCESS!

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 $0 < k < n$.

Логіка програми:

```
def main():
    a_public_key, a_private_key = key_gen(
        gen_prime(length=256),
        gen_prime(length=256)
    )

    b_public_key, b_private_key = key_gen(
        gen_prime(length=512),
        gen_prime(length=512)
    )

    print("GENERATED KEYS:")
    print(f"a_public_key = {a_public_key}")
    print(f"a_private_key = {a_private_key}")
    print(f"b_public_key = {b_public_key}")
    print(f"b_private_key = {b_private_key}")

    # A side
    k = randint(100000, 10000000000000)
    k1 = encrypt(k, b_public_key)
    s = sign(k, a_private_key)
    s1 = encrypt(s[1], b_public_key)
    message = (k1, s1)
    print(f"GENERATED MESSAGE: {k}")
    print(f"SIGNATURE: {s}")
    print(f"ENCRYPTED MESSAGE: \n{message}")
```

```
    # B side
    dec_k = decrypt(message[0], b_private_key)
    dec_s = decrypt(message[1], b_private_key)
    print(f"DECRYPTED MESSAGE: {dec_k}")
    print(f"DECRYPTED SIGNATURE: {dec_s}")

    verification = signature_verification(dec_k, dec_s, a_public_key)
    print(f"VERIFICATION: {verification}")


if __name__ == "__main__":
    main()
```

OUTPUT:

GENERATED KEYS:

a_public_key =
(814004899825668555806615596961866526845614345352934668376294404955911315937355471381080714991055142895469651201639569107881688210419273681368671802086847 1, 65537)

a_private_key =
(662201065569152067577547195508215418447196680081354225010026949665447179596245157061568986648792457476287807965902570977456709294529593182187237324727347 3,
715248310282732354434255354723298213862515013287358274376516914686502285164 71,
113807315322967831084504286996090539498458774790902154915837559429927015712001)

b_public_key =
(223874700818828440797490706304318683781201362287317607231414957192606456235759792675113663373224388776156048873631358084644656364079649158977126116471995321205395494350276023291603475464961003586513895767495265995883855829343817031530522998794081162442107260177434384965971139596015395910525003169097442116 9, 65537)

b_private_key =
(213513958526939787655925036343517936601617558759273383316178044224890885157390940710824487167249297283440769012885155037957636165531446221717599078138541072874924832950737958996959245382535929860396549066274384942299037194394278336482622748516044323255529389594818984320012048537870668559542919392242411715 3,
414128718522198375262768245046929084297836125590381226194587188579917378810379233114669082662777910283979173989676354498361783156319727683643760439595523 1,
540592069098024109099732541461757836359426830603434490220364398682483111953868094132082872535820680188907904079145602019851431670563859201334357698564399)

GENERATED MESSAGE: 5618015384826

SIGNATURE: (5618015384826,
391451964883046557398188322833345367790009912120439598994449085860384865384992709045405625958808693781482690162488729921263326643641419339279256884283217 1)

ENCRYPTED MESSAGE:

(193088587972009160318823745386706592812878955059855808844326601735102644884547750225172452782403539264388959295252178084997408903371795924978863782456718540975166039266930723644456085284399582985939460655712005523062013837457468281256792605757644894886162053120127900010343802237839693501739340855906098141 1,
664098744770149713338202326569175649755336259985099632747794425755953322528755379343719730043259546294253858320321532052087785402445433492387888386314882959965827350559947004462843058602248651608501407693036561832393401044649195215816291471996130346084410140381902277771763331064534265721954736394930966010)

DECRYPTED MESSAGE: 5618015384826

DECRYPTED SIGNATURE:
391451964883046557398188322833345367790009912120439598994449085860384865384992709045405625958808693781482690162488729921263326643641419339279256884283217 1

VERIFICATION: True

Також вивід у файлі task5_output.txt

Отже, все працює правильно!

Опис кроків протоколу:

Обидва абоненти A і B генерують пари відкритих та закритих ключів (Обов'язковою умовою роботи протоколу є n1 ≥ n).
Абонент A формує повідомлення (k1, S1) і відправляє його B, де:

$$k_1 = k^{e_1} \bmod n_1, \quad S_1 = S^{e_1} \bmod n_1, \quad S = k^d \bmod n.$$

Абонент B за допомогою свого секретного ключа d1 розшифровують повідомлення, тобто знайти:

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1,$$

І за допомогою відкритого ключа e абонента A перевіряє підпис A:

$$k = S^e \bmod n.$$

Висновок:

Отже, під час виконання даної лабораторної роботи, ми ознайомилися з різними методами тестування чисел на простоту і методами генерації ключив для асиметричної криптосистеми RSA. Практично ознайомилися з системою захисту інформації на основі криптосхеми RSA. Організували за допомогою цієї системи засекреченого зв'язку й електронного підпису, вивчили протокол розсилання ключів. Реалізували на мові Python3 протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника.