

Порядок і рекомендації щодо 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.

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента A , p_1 і q_1 – абонента B .

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) , (e_1, n_1) та секретні d і d_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 і B , перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

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 $0 < k < n$.

```
import random

def is_prime(n, k=5):
```

```

"""Тест Міллера-Рабіна"""
if n == 2 or n == 3:
    return True
if n <= 1 or n % 2 == 0:
    return False

r, s = 0, n - 1
while s % 2 == 0:
    r += 1
    s //= 2

for i in range(k):
    a = random.randint(2, n - 2)
    x = pow(a, s, n)
    if x == 1 or x == n - 1:
        continue
    for i in range(r - 1):
        x = pow(x, 2, n)
        if x == n - 1:
            break
    else:
        return False
return True

def prime_generate(length):
    while True:
        num = random.getrandbits(length)
        if is_prime(num):
            return num

def prime_generate_pairs(length=256):
    p = prime_generate(length)
    q = prime_generate(length)
    p1 = prime_generate(length)
    q1 = prime_generate(length)
    while p*q > p1*q1 or p==q or p1==q1:
        p = prime_generate(length)
        q = prime_generate(length)
        p1 = prime_generate(length)
        q1 = prime_generate(length)
    return p, q, p1, q1

def generate_key_pair(p, q):
    n = p*q
    phi = (p-1)*(q-1)
    e = 65537
    d = pow(e, -1, phi)
    public_key = (n, e)
    private_key = (p, q, d)
    return public_key, private_key

def encrypt(message, public_key):
    n, e = public_key
    encrypted_text = pow(message, e, n)
    return encrypted_text

def decrypt(encrypted_text, private_key):
    p, q, d = private_key
    original_text = pow(encrypted_text, d, p*q)
    return original_text

def sign(message, private_key):
    signature = decrypt(message, private_key)
    return signature

```

```

def verify(signature, message, public_key):
    decrypted_signature = encrypt(signature, public_key)
    return decrypted_signature == message

def send_key(A_private_key, B_public_key, message):
    key1 = encrypt(message, B_public_key)
    signat = sign(message, A_private_key)
    signat1 = encrypt(signat, B_public_key)
    return key1, signat1

def receive_key(A_public_key, B_private_key, key1, signat1):
    key = decrypt(key1, B_private_key)
    signat = decrypt(signat1, B_private_key)
    if not verify(signat, key, A_public_key):
        raise ValueError("Отриманий ключ невірний")
    return key, signat

def encode(text):
    return int.from_bytes(text.encode("utf-8"), "big")

def decode(integ):
    return integ.to_bytes((integ.bit_length()+7) // 8, "big").decode("utf-8")

p, q, p1, q1 = prime_generate_pairs()
A_public_key, A_private_key = generate_key_pair(p, q)
B_public_key, B_private_key = generate_key_pair(p1, q1)
print(f"Ключі для А: \nПублічний ключ\nn: {A_public_key[0]}\ne: {A_public_key[1]}")
print(f"Приватний ключ\np: {A_private_key[0]}\nq: {A_private_key[1]}\nd: {A_private_key[2]}\n")
print(f"Ключі для В: \nПублічний ключ\nn1: {B_public_key[0]}\ne1: {B_public_key[1]}")
print(f"Приватний ключ\np1: {B_private_key[0]}\nq1: {B_private_key[1]}\nd1: {B_private_key[2]}\n")

message_A = random.randint(1, A_public_key[0] - 1)
encrypted_text_A = encrypt(message_A, A_public_key)
decrypted_message_A = decrypt(encrypted_text_A, A_private_key)
print(f"Тест для А:\nвипадкове повідомлення: {message_A}\nшифроване повідомлення: {encrypted_text_A}\nрозшифроване: {decrypted_message_A}")
assert decrypted_message_A == message_A, "Шифрування та розшифрування для А не пройшли"
print("Шифрування та розшифрування для А пройшли успішно")

signature_A = sign(message_A, A_private_key)
verified_A = verify(signature_A, message_A, A_public_key)
print(f"підпис: {signature_A}\nперевірка: {verified_A}")
assert verified_A, "Перевірка підпису для А не пройшла"
print("Перевірка підпису для А пройшла успішно\n")

message_B = random.randint(1, B_public_key[0] - 1)
encrypted_text_B = encrypt(message_B, B_public_key)
decrypted_message_B = decrypt(encrypted_text_B, B_private_key)
print(f"Тест для В:\nвипадкове повідомлення: {message_B}\nшифроване повідомлення: {encrypted_text_B}\nрозшифроване: {decrypted_message_B}")
assert decrypted_message_B == message_B, "Шифрування та розшифрування для В не пройшли"
print("Шифрування та розшифрування для В пройшли успішно")

signature_B = sign(message_B, B_private_key)
verified_B = verify(signature_B, message_B, B_public_key)

```

```

print(f"підпис: {signature_B}\nперевірка: {verified_B}")
assert verified_B, "Перевірка підпису для В не пройшла"
print("Перевірка підпису для В пройшла успішно\n")

message = random.randint(1, A_public_key[0] - 1)
key1, signat1 = send_key(A_private_key, B_public_key, message)
key, signat= receive_key(A_public_key, B_private_key, key1, signat1)
print(f"Обмін ключами:\nпідпис: {message}\nключ1: {key1}\nпідпис1: {signat1}\nключ: {key}\nпідпис: {signat}")
print("Тест обміну ключами пройшов успішно\n")

message = encode("message")
key1, signat1 = send_key(A_private_key, B_public_key, message)
k, s= receive_key(A_public_key, B_private_key, key1, signat1)
print(f"Обмін ключами:\nпідпис: {message}\nключ1: {key1}\nпідпис1: {signat1}\nключ: {decode(k)}\nпідпис: {s}")
print("Тест обміну ключами пройшов успішно")

"""print (decode (decrypt (int (0xB5365BEA4E56AC2C7E96DA7C166B8796F3E1BBCB0ECCDC8
A002A9977200C0D3DE22E1320264FA9637486D6ED741C209643A59156B5B99DE6F00B39A5BE5E
5B) ,
(4835022568567016675703666136244359194605652339567602571304767314189459746513
,
27682111353061048507642170403125566640443418292324064649347027760001991019107
,
33580891094834047191681774302930838134463348542406185695553162095654443989930
029120861493813958941318625025268559238609723340356875562501347176779034081) )
) )
print (hex (encrypt (encode ("secret" ) ,
(int (0x8C497BAE6CC578B79F7C66C70B6C2E2C7A557D9BC2ACFDADB58E2D224BD821CF) ,
65537) ) ) )
print (verify (int (0x11C13D6C957F563CF389979496B8BA38A23072412881D47B9FB16A4826
FB205F) , encode ("SomeText" ) ,
(int (0x8C497BAE6CC578B79F7C66C70B6C2E2C7A557D9BC2ACFDADB58E2D224BD821CF) ,
65537) ) )
print (hex (sign (encode ("SomeText" ) , A_private_key) ) ) """

```

```


Ключі для А:
Публічний ключ
n: 6872282773534741169074707015784089035472885437991155840147225599934393149750960757183311731128267984207871620671374886474659766097403653628665076771235971
e: 65537
Приватний ключ
p: 99962407781330846722493830899045366042934063316830729290807324343705483340953
q: 60745663377956480852410697596949077601015268084607365071618832525148942218107
d: 3526884718168361728501895456853786290114667015535244931187026297534258132738212061422617495048122812675677635722699828290189179477787575319497737210860913

Ключі для В:
Публічний ключ
n1: 9617927230934687312094763316835272015411512586798371520810491790818445784188301811293226847076756207933855390688485097645956827625875815584745458720710653
e1: 65537
Приватний ключ
p1: 93050771125171204101786899113372783448921609907498450534812456614492152740083
q1: 103362144285690263150127804472544413567979036955280281670732527471350698835791
d1: 1012173942463270883854552786397435203059829151646912183809140162240386434158622294675038047629392726308675795149405092497787761908485871543321213522886653

Тест для А:
випадкове повідомлення: 1426901792002356732609305593837610272019697235529911379831680992191951699606920499521644643668781982318963314572709611546729050469192842896097757760450347
шифроване повідомлення: 5285958033559113445468537627934749742473730078996261716843746007077615973445426162435013589246460416020345336052358026481763606285671681734031214459799749
розшифроване: 1426901792002356732609305593837610272019697235529911379831680992191951699606920499521644643668781982318963314572709611546729050469192842896097757760450347
Шифрування та розшифрування для А пройшли успішно
підпис: 1085787241923863686806530939425821482513053222014859844848226719236491565929366563512082880842503507466623921923851305547724831078456075351142770253552945
перевірка: True
Перевірка підпису для А пройшла успішно


```


Get server key



Key size

256



Modulus

8C497BAE6CC578B79F7C66C70B6C2E2C7A557D9BC2ACFDADB58E2D224BD821CF

Public exponent

10001


зашифруємо текст “secret” за допомогою цього відкритого ключа

```
print(hex(encrypt(encode("secret"), (int(0x8C497BAE6CC578B79F7C66C70B6C2E2C7A557D9BC2ACFDADB58E2D224BD821CF), 65537))))
```

```
0x48c4fef960f943a7437f21c37a29019cd28fe64fa18f5df2aceb4f06ad2799e4
```

перевіримо на сайті


Decryption



Ciphertext

48c4fef960f943a7437f21c37a29019cd28fe64fa18f5df2aceb4f06ad2799e4

Text ▼




Message

secret

перевіримо цифровий підпис


Sign



Message

SomeText

Text ▼



Signature

11C13D6C957F563CF389979496B8BA38A23072412881D47B9FB16A4826FB205F

```
print(verify(int(0x11C13D6C957F563CF389979496B88A38A23072412881D47B9FB16A4826FB205F), encode("SomeText"), int(0x8C497BAE6CC578B79F7C66C7
```

True

Підпишемо повідомлення за допомогою приватного ключа A


```
print(hex(sign(encode("SomeText"), A_private_key)))
```


Ключі для A:

Публічний ключ

n: 25866951971747134999411668764288172133193047084031878804998129818000347681298647927480058039963674777909432057445300047055728245677813971782493321396981
e: 65537

Verify

 Clear

Message	<input type="text" value="SomeText"/>	Text ▾
Signature	<input type="text" value="7df47836673b95a9a4400eae86d28d2d4ca732ee14aba00431db73fe37124648ea074b1703c033e0ff42482e7d95a36d7ee279298fb"/>	
Modulus	<input type="text" value="7E6F5DF67014613B55DEDC201479EC4226BB92B5D38C97072F290517D1B756EC835448DB952AF4D90C6F48DD4DF9B948A1B0C/"/>	
Public exponent	<input type="text" value="10001"/>	
	<input type="button" value="Verify"/>	
Verification	<input type="text" value="true"/> 	

Висновки: у ході виконання практикуму було побудовано криптосистему RSA, розглянуто та запроваджено алгоритм роботи електронного підпису, шифрування, розшифрування повідомлень