**Міністерство освіти і науки України**

**Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"**

**Фізико-технічний інститут**

КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали студенти групи ФБ-23:

Кушнарьов Данііл та Присєвок Оксана

Київ - 2024

**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 криптосистеми 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 реалізувати власноруч, використання готових реалізацій тестів не дозволяється.

Виконання завдання:

#lab4, v\_6

import random

from math import gcd

# Завдання 1: Перевірка числа на простоту

def is\_probable\_prime(n, iterations=5):

"""Перевіряємо, чи є число n імовірно простим за допомогою тесту Міллера-Рабіна."""

if n < 2:

return False

if n in (2, 3):

return True

if n % 2 == 0:

return False

# Розкладання n-1 у вигляді 2^s \* d

s, d = 0, n - 1

while d % 2 == 0:

s += 1

d //= 2

# Локальна функція перевірки свідків складеності

def is\_composite(base):

result = pow(base, d, n)

if result in (1, n - 1):

return False

for \_ in range(s - 1):

result = pow(result, 2, n)

if result == n - 1:

return False

return True

for \_ in range(iterations):

candidate = random.randint(2, n - 2)

if is\_composite(candidate):

return False

return True

def find\_prime(bit\_size):

"""Генеруємо випадкове просте число заданого розміру в бітах."""

while True:

num = random.getrandbits(bit\_size) | (1 << (bit\_size - 1)) | 1

if is\_probable\_prime(num):

return num

# Генерація випадкового простого числа для завдання 1

bit\_length = 256

random\_prime\_number = find\_prime(bit\_length)

print(f"№1: Згенеровано випадкове просте число ({bit\_length} біт): {random\_prime\_number}")

Результат: 

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

Виконання завдання:

# Завдання 2: Генерація пар простих чисел

def generate\_key\_pairs(size=256):

"""Створюємо дві пари простих чисел (p, q) і (p1, q1), дотримуючись умови p\*q <= p1\*q1."""

while True:

p, q = find\_prime(size), find\_prime(size)

p1, q1 = find\_prime(size), find\_prime(size)

if p \* q <= p1 \* q1:

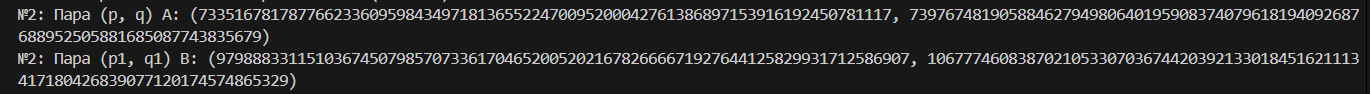
return (p, q), (p1, q1)

key\_pair\_a, key\_pair\_b = generate\_key\_pairs()

print("№2: Пара (p, q) A:", key\_pair\_a)

print("№2: Пара (p1, q1) B:", key\_pair\_b)

Результат:



1. 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), (e1, n1​) та секретні d і d1​.

Виконання завдання:

# Завдання 3: Генерація ключів RSA

def rsa\_key\_generation(prime1, prime2):

"""Генеруємо публічний та приватний ключі RSA на основі двох простих чисел."""

modulus = prime1 \* prime2

phi = (prime1 - 1) \* (prime2 - 1)

# Пошук відкритої експоненти e

public\_exponent = 65537 # Стандартне значення

if gcd(public\_exponent, phi) != 1:

for candidate in range(3, phi, 2):

if gcd(candidate, phi) == 1:

public\_exponent = candidate

break

# Обчислення секретної експоненти d

private\_exponent = pow(public\_exponent, -1, phi)

# Повернення пари ключів

return (public\_exponent, modulus), (private\_exponent, prime1, prime2)

# Генерація RSA-ключів для абонентів A та B

public\_key\_a, private\_key\_a = rsa\_key\_generation(key\_pair\_a[0], key\_pair\_a[1])

public\_key\_b, private\_key\_b = rsa\_key\_generation(key\_pair\_b[0], key\_pair\_b[1])

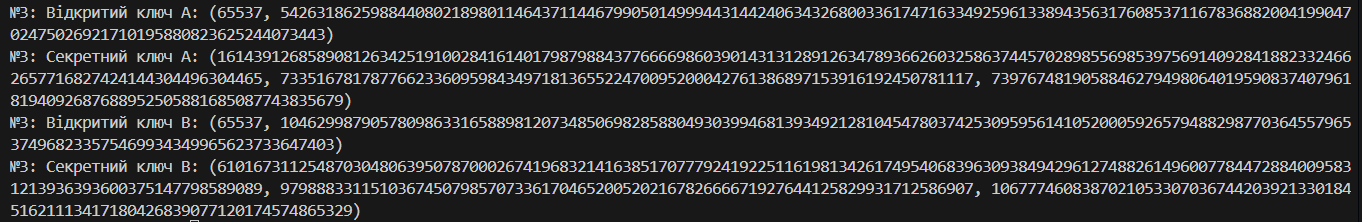
print("№3: Відкритий ключ A:", public\_key\_a)

print("№3: Секретний ключ A:", private\_key\_a)

print("№3: Відкритий ключ B:", public\_key\_b)

print("№3: Секретний ключ B:", private\_key\_b)

Результат:



1. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А 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 А и B, перевірити правильність розшифрування. Скласти для А і B повідомлення з цифровим підписом і перевірити його.

# Функція для перетворення рядка в число

def string\_to\_number(message):

    return int.from\_bytes(message.encode('utf-8'), 'big')

# Функція для перетворення числа в рядок

def number\_to\_string(number):

    byte\_length = (number.bit\_length() + 7) // 8

    return number.to\_bytes(byte\_length, 'big').decode('utf-8', errors='ignore')

# Функція шифрування

def encrypt(message, public\_key):

    e, n = public\_key

    m = string\_to\_number(message)

    cipher = pow(m, e, n)

    return cipher

# Функція дешифрування

def decrypt(cipher, private\_key):

    d, p, q = private\_key

    n = p \* q

    decrypted = pow(cipher, d, n)

    return number\_to\_string(decrypted)

# Функція хешування повідомлення (SHA-256)

def hash\_message(message):

    return int(hashlib.sha256(message.encode('utf-8')).hexdigest(), 16)

# Функція підпису повідомлення

def sign\_message(message, private\_key):

    d, p, q = private\_key

    n = p \* q

    message\_hash = hash\_message(message)

    signature = pow(message\_hash, d, n)

    return signature

# Функція перевірки підпису

def verify\_signature(message, signature, public\_key):

    e, n = public\_key

    message\_hash = hash\_message(message)

    verified\_hash = pow(signature, e, n)

    return message\_hash == verified\_hash

# Приклад шифрування та дешифрування

    print('\n№4')

    message = "Hello, RSA!"

    print(f"Повідомлення для шифрування: {message}")

    encrypted\_message = rsa.encrypt(message, public\_key\_a)

    print(f"Зашифроване повідомлення: {encrypted\_message}")

    decrypted\_message = rsa.decrypt(encrypted\_message, private\_key\_a)

    print(f"Дешифроване повідомлення: {decrypted\_message}")

     # Приклад підпису та перевірки підпису

    signature = rsa.sign\_message(message, private\_key\_a)

    print(f"Підпис повідомлення: {signature}")

    is\_valid = rsa.verify\_signature(decrypted\_message, signature, public\_key\_a)

    print(f"Перевірка підпису: {'Дійсний' if is\_valid else 'Недійсний'}")

     # Приклад перевірки підпису зміненого повідомлення

    is\_valid = rsa.verify\_signature(decrypted\_message + 'a', signature, public\_key\_a)

    print(f"Перевірка підпису пошкодженого повідомлення: {'Дійсний' if is\_valid else 'Недійсний'}")

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

1. 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 rsa

class Person:

    def \_\_init\_\_(self,name, bit\_size=256):

        # Generate key pairs

        prime1, prime2 = rsa.generate\_key\_pairs(size=bit\_size)[0]

        self.name = name

        self.public\_key, self.private\_key = rsa.rsa\_key\_generation(prime1, prime2)

        self.friends\_public\_key = None

    def send\_message(self, message):

        encrypted\_message = rsa.encrypt(message, self.friends\_public\_key)  # Encrypt message

        signature = rsa.sign\_message(message, self.private\_key)  # Create signature

        return encrypted\_message, signature

    def receive\_message(self, encrypted\_message, signature):

        # Decrypt the message

        decrypted\_message = rsa.decrypt(encrypted\_message, self.private\_key)

        # Verify the signature

        is\_valid = rsa.verify\_signature(decrypted\_message, signature, self.friends\_public\_key)

        if not is\_valid:

            return None

        # Return the decrypted message as a string

        return decrypted\_message

    def take\_friends\_key(self, public\_key):

        self.friends\_public\_key = public\_key

    print('\n№5')

    # Create two people (sender and receiver)

    alice = Person('Alice')

    bob = Person('Bob')

    #Exchange public keys

    alice.take\_friends\_key(bob.public\_key)

    bob.take\_friends\_key(alice.public\_key)

    # Alice sends a message to Bob

    message = "Hello, Bob!"

    print(f"Аліса пише '{message}' Бобу")

    encrypted\_message, encrypted\_message\_signature = alice.send\_message(message)

    print(f"Зашифроване повідомлення: {encrypted\_message}")

    print(f"Підпис повідомлення Аліси: {encrypted\_message\_signature}")

    decrypted\_message= bob.receive\_message(encrypted\_message, encrypted\_message\_signature)

    if decrypted\_message:

        print(f"Боб отримав: {decrypted\_message}")

    else:

        print('Повідомлення пошкоджене')

    # Bob sends a message to Alice

    message = "Hello, Alice!"

    print(f"\nБоб пише '{message}' Алісі")

    encrypted\_message, encrypted\_message\_signature = bob.send\_message(message)

    print(f"Зашифроване повідомлення: {encrypted\_message}")

    print(f"Підпис повідомлення Боб: {encrypted\_message\_signature}")

    decrypted\_message= alice.receive\_message(encrypted\_message, encrypted\_message\_signature)

    if decrypted\_message:

        print(f"Аліса отримала: {decrypted\_message}")

    else:

        print('Повідомлення пошкоджене')

**Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис**

Підставив відкритий ключ А конвертований у HEX

**Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис**

**Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис**



**Зображення, що містить знімок екрана, текст

Автоматично згенерований опис**



if \_\_name\_\_ == "\_\_main\_\_":

    message = "Hello, RSA!"

    print(f"\n Повідомлення для шифрування: {message}")

    # Приклад шифрування та дешифрування

    print('Розшифрувати повідомлення з сервера')

    public\_key\_a = (

        65537,

        6178263559633057148267807723786822689853507354271103554805737053903986071238208067502946203776131463132298348011108982858765071921031193101260286905417517

        )

    private\_key\_a = (

        1208370573986855158559237673428742439210556743016113117254374438209580747686487537184350425347702431387234863951015319288419174432654233652304400063766457,

        82401489338079545932457810672827645719787070425033484991970341676096304784619,

        74977571513115179459869276594214486337005102965765164634907665882094565443143

        )

    encrypted\_message = 3943341437669028212230546245688015304020085812894358406729217090350170765294999141260704744258064127544053973186585184940320148081729006037045501390745259

    decrypted\_message = decrypt(encrypted\_message, private\_key\_a)

    print(f"Дешифроване повідомлення: {decrypted\_message}")

     # Приклад підпису та перевірки підпису

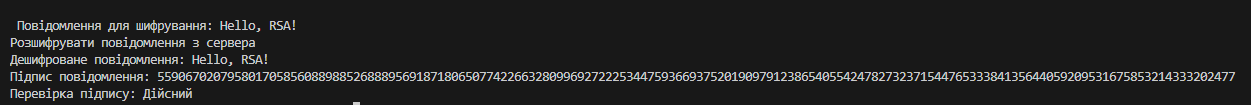
    signature = sign\_message(message, private\_key\_a)

    print(f"Підпис повідомлення: {signature}")

    is\_valid = verify\_signature(decrypted\_message, signature, public\_key\_a)

    print(f"Перевірка підпису: {'Дійсний' if is\_valid else 'Недійсний'}")

Підставив зашифроване повідомлення і ключі з сайту



Підпис дійсний