МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут

Кафедра інформаційної безпеки

Дисципліна «Криптографія»

Комп’ютерний практикум

Робота No 4

Виконав : студент групи ФБ-24 Луняка Артем

Київ – 2024

**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.

**Варіант 10**

***Завдання до виконання***

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 p1 , q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq <= p1q1; p і q – прості числа для побудови ключів абонента А, p1 і q1 – 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 А і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n), (e1,n1) та секретні d і d1.

4. 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.

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. Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

**1. Написання програм для виконання роботи.**

В якості мови програмування виберемо Python. Напишемо декілька службових модулів та основну програму для виконання лабораторної роботи. У службових модулях опишемо декілька класів та функцій, які ми зможемо використовувати і надалі для схожих задач. У цій роботі ми також будемо використовувати модулі, написані в рамках виконання лабораторних робіт №1, №2 та №3: affine\_cipher(функція inverse) та bigram\_affine\_cipher(функція gcd).

**2. Модуль primes.**

Модуль primes призначено для роботи з псевдопростими числами: вибору випадкового числа заданого діапазону та проведення тесту на простоту.

Цей модуль містить тільки функції.

Функція swift\_power виконує «швидке» піднесення до степеня з використанням схеми Горнера так, як це було описано в завданні. Ця функція багаторазово використовується практично у всіх діях шифрування та розшифрування.

Функція miller\_rabin\_primes\_test перевіряє число p на простоту за k основами, реалізує тест Міллера-Рабіна. Коли для піднесення до степеня використовувалась звичайна операція Python(\*\*), ця дія для великих чисел не закінчувалась за прийнятний час. Використання swift\_power вирішило цю проблему.

Функція select\_prime вибирає випадкове непарне число в діапазоні від n0 до n1 та перевіряє, чи є воно псевдопростим. Повертає перше знайдене псевдопросте число.

Функція select\_prime\_multiplied за наявним псевдопростим числом рр знаходить псевдопросте число р=2\*і\*рр+1.

Функція select\_prime\_with\_prime\_divisor знаходить псевдопросте число у діапазоні від n0 до n1, для якого є великий псевдопростий дільник. Використовує select\_prime\_multiplied.

Функція select\_primes\_pair вибирає пару псевдопростих чисел у діапазоні від n0 до n1. Використовує select\_prime\_with\_prime\_divisor.

Функція select\_two\_primes\_pairs знаходить дві пари псевдопростих чисел p, q, p1, q1, такі, що p\*q<=p1\*q1.

Функція oyler\_phi - функція Ойлера.

from random import randint  
  
from bigram\_affine\_cipher import gcd  
  
  
PRIME\_TESTS\_QUANTITY = 10  
  
  
def swift\_power(x, a, m):  
 *"""x\*\*a (mod m)"""* ai\_list = list()  
 while a > 0:  
 ai\_list.append(a % 2)  
 a //= 2  
 y = 1  
 for i in range(len(ai\_list) - 1, -1, -1):  
 y = (y \*\* 2) % m  
 y = y if ai\_list[i] == 0 else (y \* x) % m  
 return y  
  
  
def miller\_rabin\_primes\_test(p, k):  
 if p % 2 == 0:  
 return False  
  
 d = (p - 1) // 2  
 s = 1  
 while d % 2 == 0:  
 d //= 2  
 s += 1  
 for i in range(k):  
 *# print("i=", i)* a = randint(1, p)  
 if gcd(a, p) > 1:  
 return False  
 *# print("a=", a, "d=", d)  
 # u = (a \*\* d) % p* u = swift\_power(a, d, p)  
 *# print(u)* if u != 1:  
 j = 1  
 while u != p - 1 and j < s:  
 u = (u \*\* 2) % p  
 j += 1  
 *# print("j=", j)* if u != p - 1:  
 return False  
 return True  
  
  
def select\_prime(n0, n1):  
 *"""select prime p: n0 <= p <= n1"""* x = randint(n0, n1)  
 m0 = x if x % 2 != 0 else x + 1  
 for i in range((n1 - m0) // 2):  
 p = m0 + 2 \* i  
 if miller\_rabin\_primes\_test(p, PRIME\_TESTS\_QUANTITY):  
 return p  
  
 return None  
  
  
def select\_prime\_multiplied(pp):  
 *"""select prime p = 2ipp + 1"""* result = None  
 i = 0  
 while True:  
 i += 1  
 p = 2 \* i \* pp + 1  
 if miller\_rabin\_primes\_test(p, PRIME\_TESTS\_QUANTITY):  
 result = p  
 break  
  
 return result  
  
  
def select\_prime\_with\_prime\_divisor(n0, n1):  
 *"""select primes p', p: p = 2ip' + 1"""* n02 = n0 // 2  
 n12 = n1 // 2  
 while True:  
 pp = select\_prime(n02, n12)  
 if pp:  
 break  
  
 p = select\_prime\_multiplied(pp)  
 return p  
  
  
def select\_primes\_pair(n0, n1):  
 p = select\_prime\_with\_prime\_divisor(n0, n1)  
 q = select\_prime\_with\_prime\_divisor(n0, n1)  
 return p, q  
  
  
def select\_two\_primes\_pairs(n0, n1):  
 p, q = select\_primes\_pair(n0, n1)  
 p1, q1 = select\_primes\_pair(n0, n1)  
 if p \* q > p1 \* q1:  
 p, p1 = p1, p  
 q, q1 = q1, q  
 return p, q, p1, q1  
  
  
def oyler\_phi(p, q):  
 return (p -1) \* (q - 1)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print(miller\_rabin\_primes\_test(93, 10))  
 print(miller\_rabin\_primes\_test(103, 10))  
 print(miller\_rabin\_primes\_test(2 \*\* 256 - 1, 10))  
 p, q, p1, q1 = select\_two\_primes\_pairs(2 \*\* 256, 2 \*\* 257)  
 print(p, q)  
 print(p1, q1)

**3. Модуль rsa**

Модуль rsa реалізує дії , що необхідні для побудови криптосистеми RSA. У цьому модулі реалізовані функції, які мають бути згідно завдання, а також клас Caller(абонент), який полегшує виконання завдання. Розглянемо спочатку функції.

Функція generate\_key\_pair генерує відкритий та секретний ключі за заданими p та q, або вибирає p i q, якщо вони не задані.

Функція encrypt шифрує повідомлення відкритим ключем.

Функція decrypt розшифровує повідомлення секретним ключем.

Функція sign підписує повідомлення електронним підписом.

Функція verify перевіряє електронний підпис повідомлення.

Функція send\_key готує ключ до відправки відкритим каналом.

Функція receive\_key перевіряє ключ, який був відправлений відкритим каналом.

Клас Caller містить інформацію, що необхідна для криптосистеми RSA одного абонента. Цей клас містить методи, які фактично дублюють функції, описані вище, але використовують для них ключі, що зберігаються в об’єкті класу. Окрім цього, клас містить методи для збереження ключів у текстових файлах, а також читання ключів з текстових файлів.

from affine\_cipher import inverse  
from primes import oyler\_phi, select\_primes\_pair, select\_two\_primes\_pairs, swift\_power  
  
  
E\_STANDARD = 2 \*\* 16 + 1  
KEY\_STORE\_PATH = ".\\"  
  
  
def generate\_key\_pair(p=None, q=None, n0=2 \*\* 256, n1=2 \*\* 257):  
 if p is None or q is None:  
 p, q = select\_primes\_pair(n0, n1)  
 n = p \* q  
 e = E\_STANDARD  
 while True:  
 d = inverse(e, oyler\_phi(p, q))  
 if d is not None:  
 break  
  
 e = e \* 2 + 1  
 return d, e, n  
  
  
def encrypt(message, e, n):  
 encrypted = swift\_power(message, e, n)  
 return encrypted  
  
  
def decrypt(encrypted, d, n):  
 message = swift\_power(encrypted, d, n)  
 return message  
  
  
def sign(message, d, n):  
 signature = swift\_power(message, d, n)  
 return message, signature  
  
  
def verify(message, signature, e, n):  
 m = swift\_power(signature, e, n)  
 return m == message  
  
  
def send\_key(k, e1, n1, d, n):  
 k1 = swift\_power(k, e1, n1)  
 s = swift\_power(k, d, n)  
 s1 = swift\_power(s, e1, n1)  
 return k1, s1  
  
  
def receive\_key(k1, s1, e, n, d1, n1):  
 k = swift\_power(k1, d1, n1)  
 s = swift\_power(s1, d1, n1)  
 kk = swift\_power(s, e, n)  
 return k if kk == k else None  
  
  
def \_make\_hex\_wo\_0x(n):  
 return hex(n)[2:]  
  
  
class Caller:  
  
 def \_\_init\_\_(self, name, p=None, q=None, keystore=KEY\_STORE\_PATH):  
 self.\_name = name  
 self.\_p = p  
 self.\_q = q  
 self.\_keystore = keystore  
 if p is None or q is None:  
 self.\_d = self.\_e = self.\_n = None  
 return  
  
 self.\_d, self.\_e, self.\_n = generate\_key\_pair(self.\_p, self.\_q)  
  
 @property  
 def name(self):  
 return self.\_name  
  
 @classmethod  
 def from\_files(cls, name, keystore=KEY\_STORE\_PATH, filename\_pub="", fileanme\_sec=""):  
 caller = cls(name, keystore=keystore)  
 caller.load\_own\_public\_key(filename\_pub)  
 caller.load\_own\_secret\_key(fileanme\_sec)  
 return caller  
  
 def save\_own\_public\_key(self, filename=""):  
 if not filename:  
 filename = f"{self.\_name}\_pub.txt"  
 with open(self.\_keystore + filename, 'w') as f:  
 print(f"{\_make\_hex\_wo\_0x(self.\_n)}\n{\_make\_hex\_wo\_0x(self.\_e)}\n", file=f)  
  
 def save\_own\_secret\_key(self, filename=""):  
 if not filename:  
 filename = f"{self.\_name}\_sec.txt"  
 with open(self.\_keystore + filename, 'w') as f:  
 print(f"{\_make\_hex\_wo\_0x(self.\_d)}", file=f)  
 print(f"{\_make\_hex\_wo\_0x(self.\_p)}", file=f)  
 print(f"{\_make\_hex\_wo\_0x(self.\_q)}", file=f)  
  
 def load\_own\_public\_key(self, filename=""):  
 if not filename:  
 filename = f"{self.\_name}\_pub.txt"  
 with open(self.\_keystore + filename, 'r') as f:  
 self.\_n = int(f.readline(), 16)  
 self.\_e = int(f.readline(), 16)  
  
 def load\_own\_secret\_key(self, filename=""):  
 if not filename:  
 filename = f"{self.\_name}\_sec.txt"  
 with open(self.\_keystore + filename, 'r') as f:  
 self.\_d = int(f.readline(), 16)  
 self.\_p = int(f.readline(), 16)  
 self.\_q = int(f.readline(), 16)  
  
 def save\_public\_key(self, filename, e, n):  
 with open(self.\_keystore + filename, 'w') as f:  
 print(f"{\_make\_hex\_wo\_0x(n)}\n{\_make\_hex\_wo\_0x(e)}\n", file=f)  
  
 def load\_public\_key(self, filename):  
 with open(self.\_keystore + filename, 'r') as f:  
 n = int(f.readline(), 16)  
 e = int(f.readline(), 16)  
 return e, n  
  
 def get\_public\_key(self):  
 return self.\_e, self.\_n  
  
 def encrypt(self, message, e, n):  
 return encrypt(message, e, n)  
  
 def decrypt(self, encrypted):  
 return decrypt(encrypted, self.\_d, self.\_n)  
  
 def sign(self, message):  
 return sign(message, self.\_d, self.\_n)  
  
 def verify(self, message, signature, e, n):  
 return verify(message, signature, e, n)  
  
 def send\_key(self, k, e, n):  
 return send\_key(k, e, n, self.\_d, self.\_n)  
  
 def receive\_key(self, k1, s1, e, n):  
 return receive\_key(k1, s1, e, n, self.\_d, self.\_n)

**4. Основна програма**

Основна програма для лабораторної роботи міститься у модулі lab4. У цьому модулі, зокрема описано константи для шляхів до файлів, що використовуються у програмі.

У головній програмі ми спочатку знаходимо дві пари псевдопростих чисел. Ці пари псевдопростих чисел використовуємо для створення двох абонентів(об’єктів класу Caller): а та b. Для цих об’єктів зберігаємо їх ключі у файлах та повертаємо їх відкриті ключі, які будемо використовувати у подальшому.

Вибираємо випадкове ціле число в діапазоні від 0 до 2256. Це число буде нашим повідомленням. Шифруємо це повідомлення відкритим ключем абонента А, показуємо зашифроване повідомлення та розшифровуємо його секретним ключем абонента В. Початкове та розшифроване повідомлення мають бути рівними.

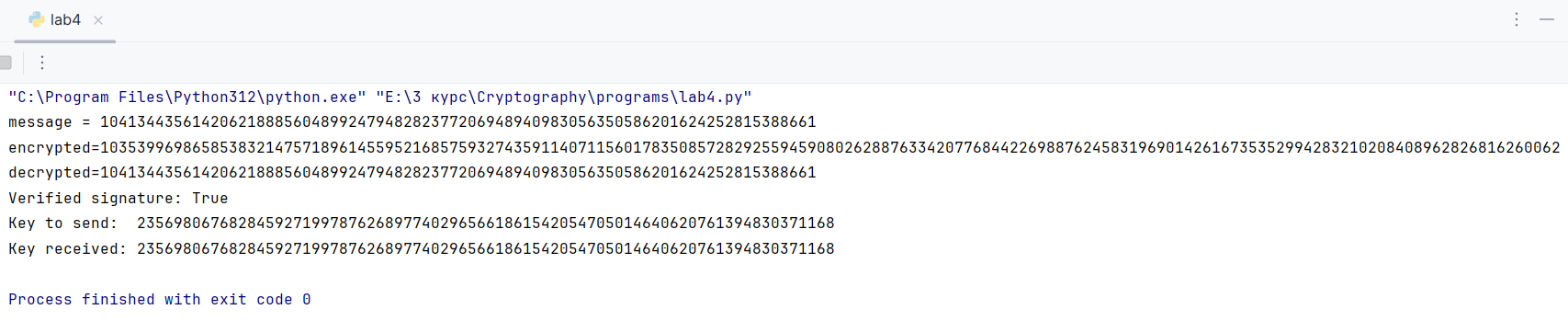
Підписуємо створене раніше повідомлення ключем абонента А. Перевіряємо підпис від особи абонента В, використовуючи відкритий ключ абонента А. Показуємо результат перевірки(має бути True).

Генеруємо ключ К(ще одне випадкове число), готуємо та відправляємо його від особи абонента А за допомогою send\_key. Моделюємо отримання цього ключа абонентом В за допомогою receive\_key. Показуємо розшифрований ключ, який має бути рівним початковому ключу.

from random import randint  
  
from primes import select\_two\_primes\_pairs  
from rsa import Caller  
  
N0 = 2 \*\* 256  
N1 = 2 \*\* 257  
LAB4\_PATH = "..\\lab4\\"  
  
p, q, p1, q1 = select\_two\_primes\_pairs(N0, N1)  
a = Caller("A", p, q, keystore=LAB4\_PATH)  
a.save\_own\_public\_key()  
a.save\_own\_secret\_key()  
  
b = Caller("B", p1, q1, keystore=LAB4\_PATH)  
b.save\_own\_public\_key()  
b.save\_own\_secret\_key()  
  
message = randint(1, N0)  
print(f"message = {message}")  
ae, an = a.get\_public\_key()  
be, bn = b.get\_public\_key()  
  
*# encrypt / decrypt*encrypted = a.encrypt(message, be, bn)  
print(f"encrypted={encrypted}")  
message1 = b.decrypt(encrypted)  
print(f"decrypted={message1}")  
  
*# sign / verify signature*message, signature = a.sign(message)  
print(f"Verified signature: {b.verify(message, signature, ae, an)}")  
  
*# send / receive key*k = randint(1, N0)  
print(f"Key to send: {k}")  
k1, s1 = a.send\_key(k, be, bn)  
kk = b.receive\_key(k1, s1, ae, an)  
print(f"Key received: {kk}")

**5. Запуск програми**

Запустимо програму.



Бачимо, що розшифроване повідомлення дорівнює початковому, перевірка цифрового підпису є успішною та ключ пересланий та отриманий правильно.

Висновок

У цій роботі було виконано 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. Виконання написаних програм демонструє правильність здійсненої реалізації.