Міністерство освіти і науки України

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

Фізико-технічний інститут

Криптографія

Лабораторна робота №4

Виконали студенти групи ФБ-13

Дмитрів Анастасія

Лагно Костянтин

**Київ 2023**

**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 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 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.

**Хід роботи:**

1. Написали функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тест перевірки на простоту. В якості теста перевірки на простоту, вибрали тест Міллера-Рабіна. Оскільки даний тест використовує пошук НСД, з попередньої лабораторної взяли функції gcd та inverse\_mod.

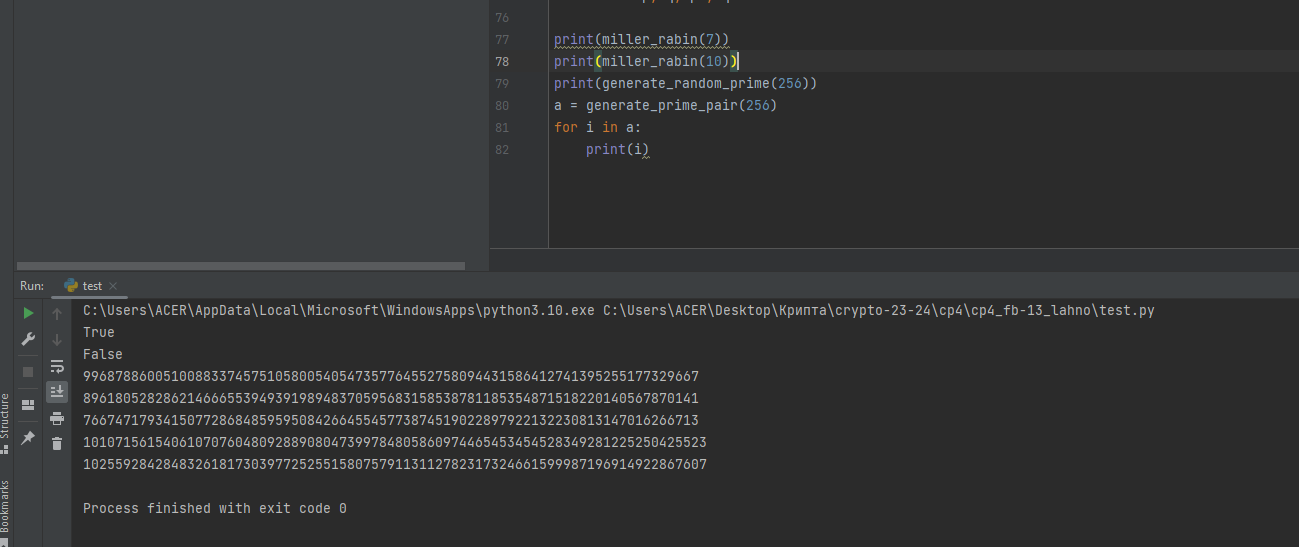
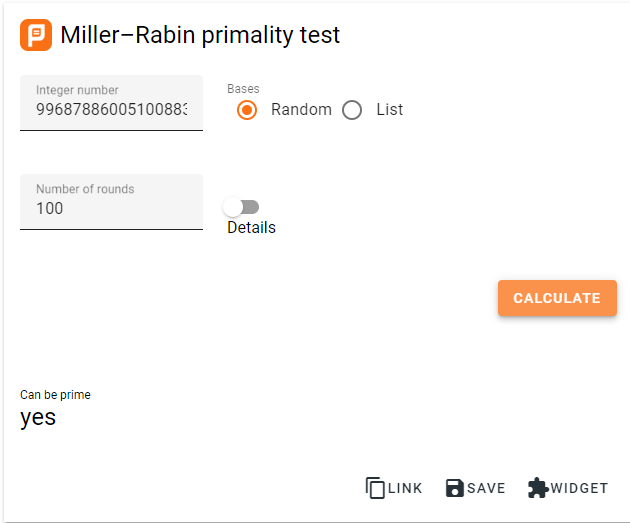
Код функцій:

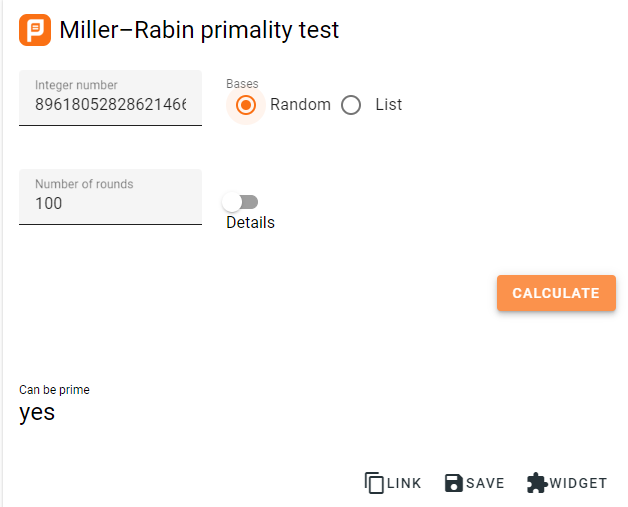
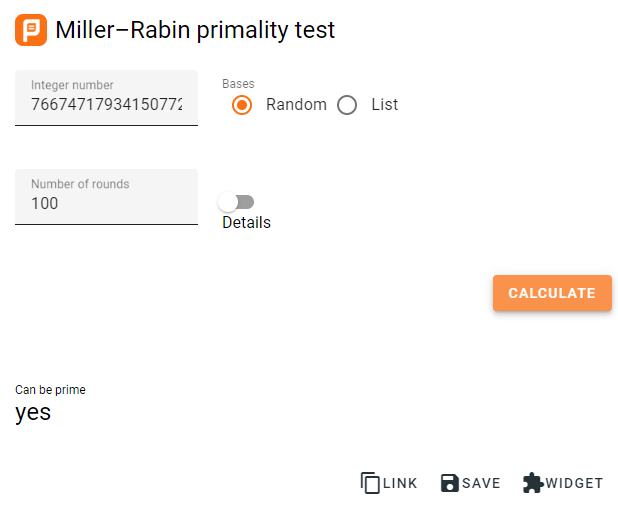
def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return a  
  
  
#a^-1  
def inverse\_mod(a, mod, v = None):  
 if v is None:  
 v = [0, 1]  
 if a == 0 or gcd(a, mod) != 1:  
 return None  
 else:  
 d = mod % a  
 q = mod // a  
 v.append((v[len(v) - 2] - q \* v[len(v) - 1]))  
 if d != 0:  
 mod, a = a, d  
 return inverse\_mod(a, mod, v)  
 else:  
 return v[len(v) - 2]  
  
  
def miller\_rabin(n):  
 k = 100  
 if n <= 1:  
 return False  
 if n <= 3:  
 return True  
 # Розклад n - 1 = 2^s \* d, де d непарне  
 s, d = 0, n - 1  
 while d % 2 == 0:  
 s += 1  
 d //= 2  
  
 for \_ in range(k):  
 x = random.randint(1, n)  
 if gcd(n, x) != 1:  
 return False  
 xd = pow(x, d, n)  
 if xd == 1 or xd == n - 1:  
 return True  
 for i in range(0, s-1):  
 w = (2\*\*i)\*d  
 if pow(x, w, n) == n-1:  
 return True  
 return False

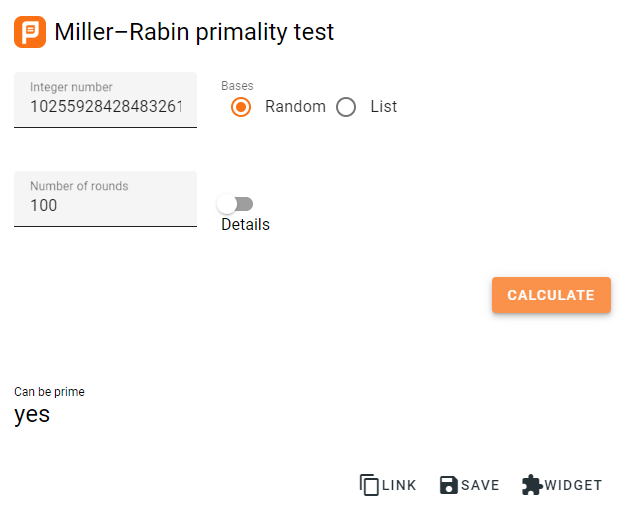
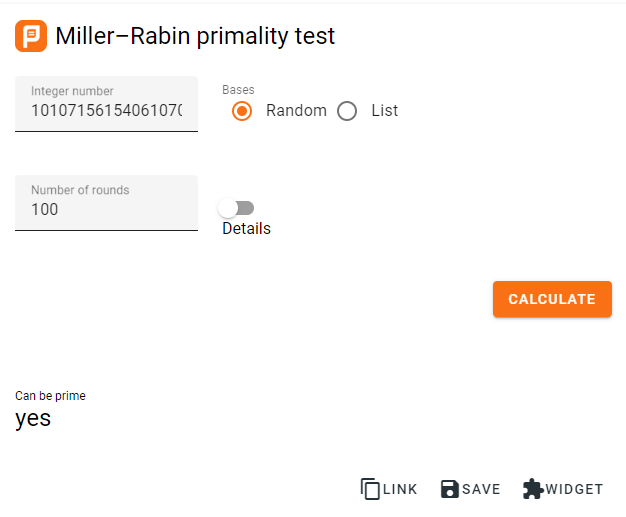
Перевірка роботи буде в наступному пункті, оскільки вони пов’язані.

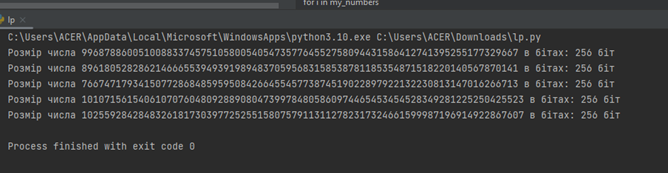
2. З допомогою даних функцій, а також функцій generate\_random\_prime і generate\_prime\_pair згенерували дві пари чисел довжиною 256 біт. Зразу також додали перевірку, якщо пара для абонента А більше, ніж для В, то вони міняються місцями, замість того щоб заново генеруватись.  
Код функцій:

def generate\_random\_prime(bits):  
 while True:  
 number = random.randint(2\*\*(bits-1), 2\*\*bits - 1)  
 if miller\_rabin(number):  
 return number  
  
  
def generate\_prime\_pair(bits):  
 p = generate\_random\_prime(bits)  
 q = generate\_random\_prime(bits)  
 p1 = generate\_random\_prime(bits)  
 q1 = generate\_random\_prime(bits)  
  
 if p\*q > p1\*q1:  
 p, p1 = p1, p  
 q, q1 = q1, q  
  
 return p, q, p1, q1

Перевірка роботи:  
Якщо число є простим, функція тесту виводить значення True, в іншому випадку – False. Функції для генерації числа та пари чисел мають виводити прості числа розміром 256 байт, якщо ж числа не прості – функція циклічно генерує нове число, поки всі не будуть простими.  
  
Бачимо, що коли в функцію тесту передається число 7, яке є простим, результатом є True, а при 10, яке не є простим, False.  
  
Перевірю на окремому сайті, чи є отримані числа простими:  


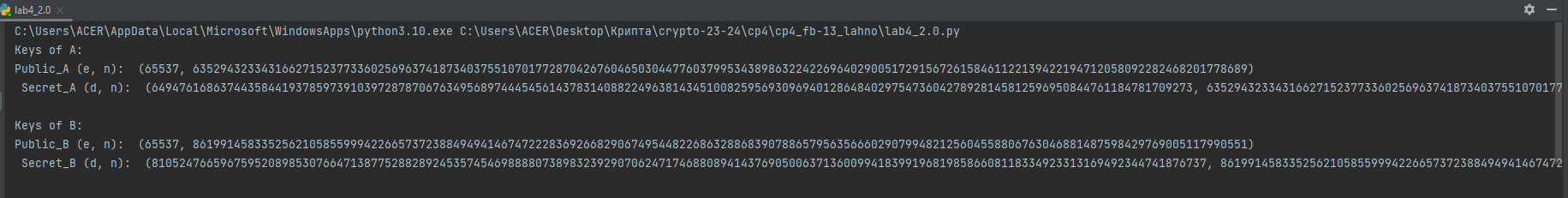


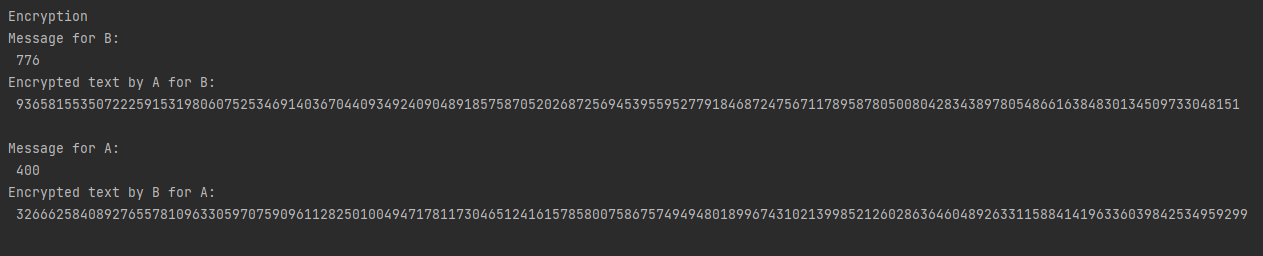
І окремим скриптом перевірю числа на розмір:  
  
  
Як бачимо, функції працюють коректно.

3 – 4 – 5.

Тепер переходимо до основного завдання лабораторної.  
Код основних функцій:

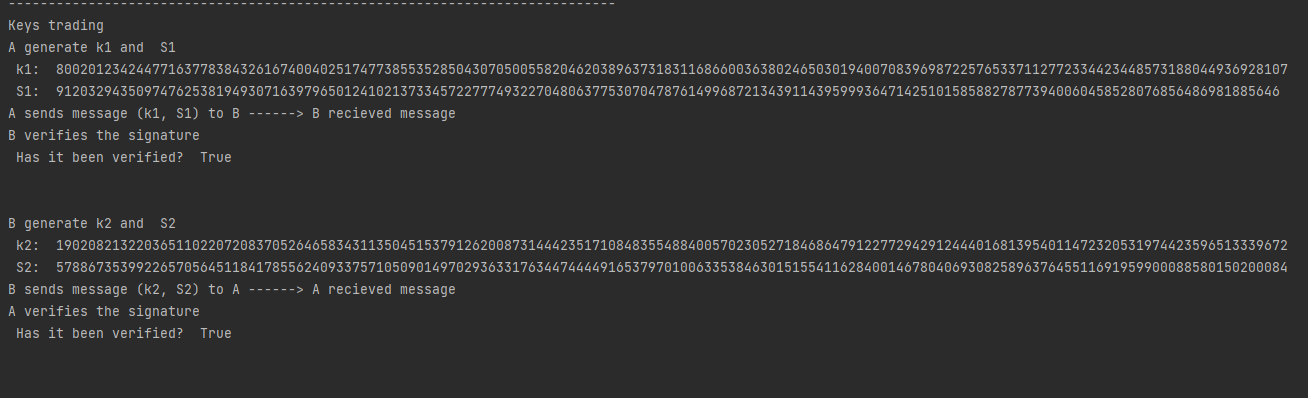
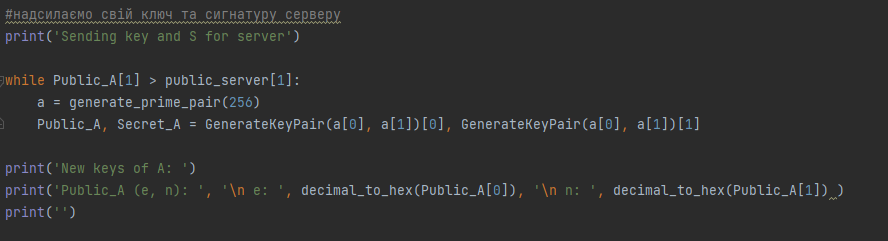
def GenerateKeyPair(p, q):  
 n = p\*q  
 f = (p-1)\*(q-1) #ойлера  
 e = (2\*\*16) + 1  
 if gcd(e, f) != 1:  
 return false  
 else:  
 m = inverse\_mod(e, f)  
 if m < 0:  
 d = m + f  
 else:  
 d = m  
 public\_k = (e, n)  
 secret\_k = (d, n)  
 return public\_k, secret\_k  
  
  
def Encrypt(key\_p, M): #tuple  
 e, n = key\_p[0], key\_p[1]  
 C = pow(M, e, n)  
 return C  
  
  
def Decrypt(key\_s, C):  
 d, n = key\_s[0], key\_s[1]  
 M = pow(C, d, n)  
 return M  
  
  
def Sign(key\_s, k):  
 d, n = key\_s[0], key\_s[1]  
 S = pow(k, d, n)  
 return S  
  
  
def Verify(k, S, key\_p):  
 e, n = key\_p[0], key\_p[1]  
 return k == pow(S, e, n)  
  
  
def SendKey(key\_s, key\_p):  
 e1, n1 = key\_p[0], key\_p[1]  
 k = random.randint(1, 1000)  
 k1 = pow(k, e1, n1)  
 S = Sign(key\_s, k)  
 S1 = pow(S, e1, n1)  
 return k1, S1, k  
  
  
def ReceiveKey(key\_s, k1, S1, key\_p):  
 d1, n1 = key\_s[0], key\_s[1]  
 k = pow(k1, d1, n1)  
 S = pow(S1, d1, n1)  
 check = Verify(k, S, key\_p)  
 return check

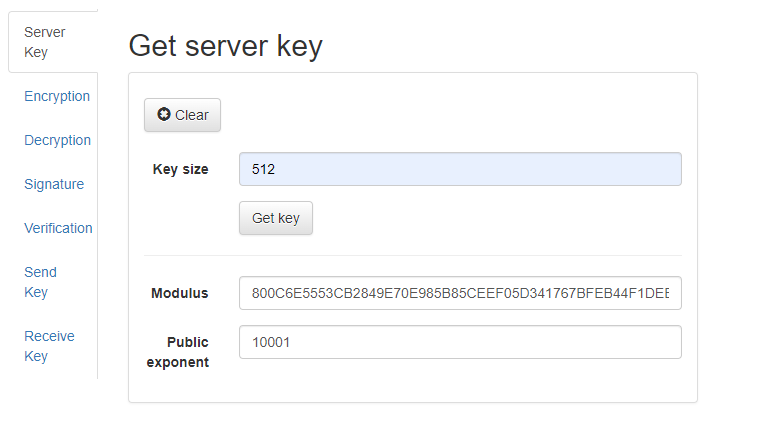
Генеруємо дві пари ключів для кожного з абонентів:  


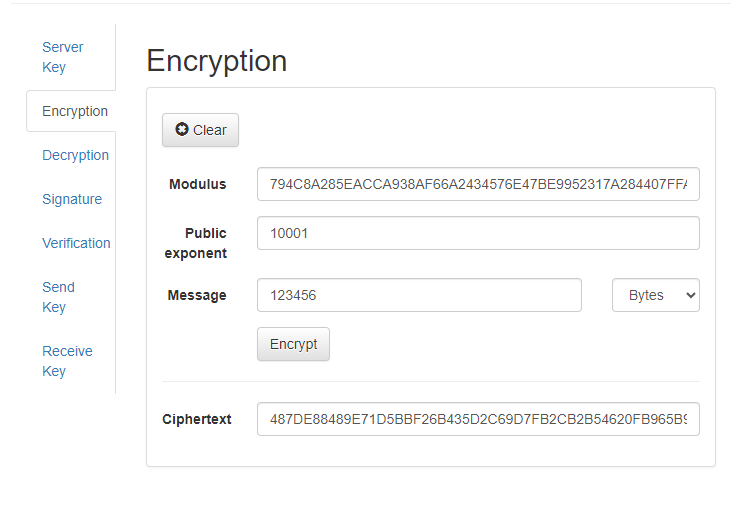
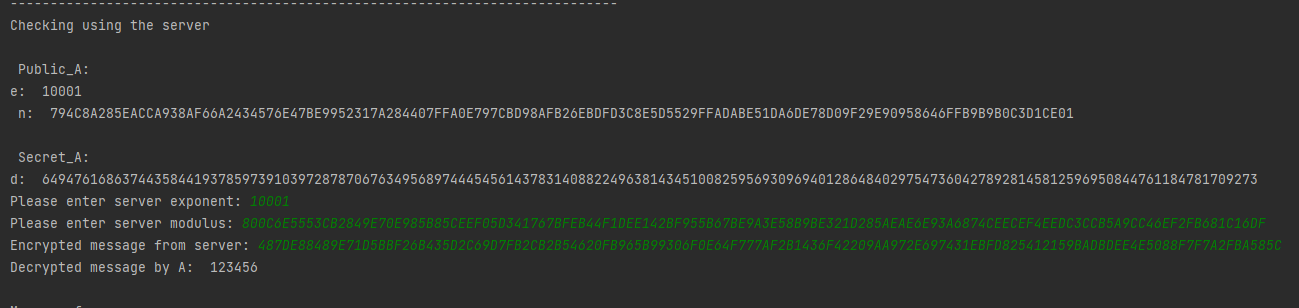
Шифрування повідомлень:  


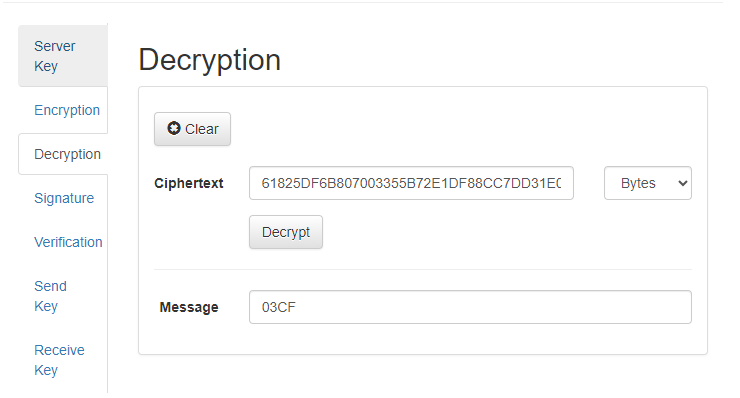
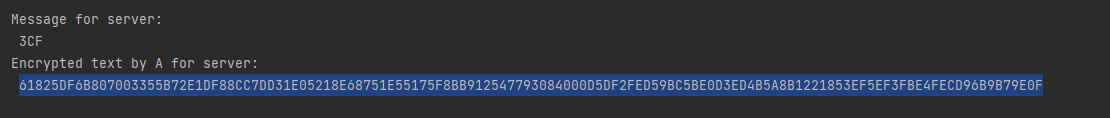
Розшифрування повідомлень:  

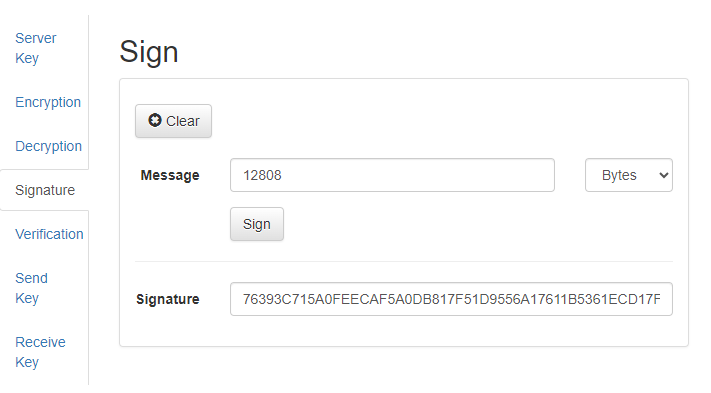
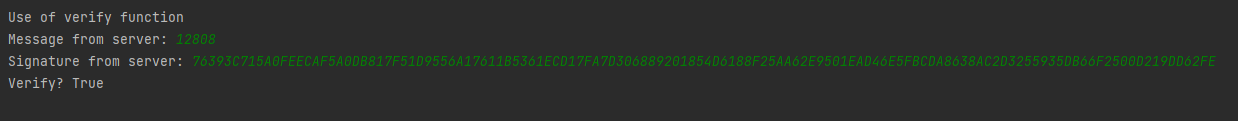

Обмін ключами (тут одразу і використовується створення цифрового підпису):

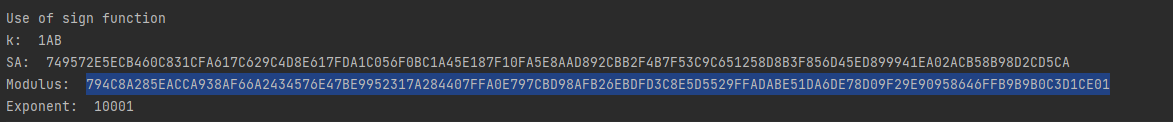
  
  
Тепер перевіримо роботу нашої програми з сервісом з методички. Основний момент, над яким ми довго ламали голову – в нас працювали всі функції і завдання, окрім отримання ключа сайтом. Як виявилось, на сайті потрібно при генерації ключів задавати вдвічі більше значення розміру, ніж яке ми задаємо при створенні ключів у нас, так як якщо задати на сайті 256 біт, і у нас 256, то виходить, що у сайту значення n буде 256 біт, а у нас, так як ми перемножуємо числа розміром 256 байтів, набагато більше. На роботу шифрування, дешифрування і перевірки цифрового підпису це ніяк не впливає, оскільки там допускається різниця між розмірами ключів в абонентів в обидві сторони, а от при розсиланні ключів важливо, щоб розмір n1 у абонента, який отримує ключ, був більшим, ніж розмір n у того, хто надсилає. Додатково ще була додана перевірка, що якщо публічний ключ нашого абонента більше, ніж на сайті, то пари ключів генеруються заново, поки публічний ключ сайту не буде більшим:  
  


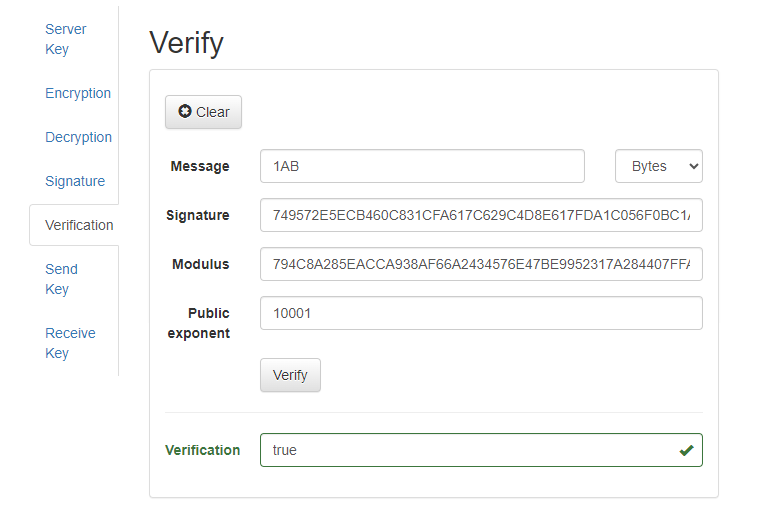
Створення пари і розшифрування повідомлення, отриманого від сайту:  
  


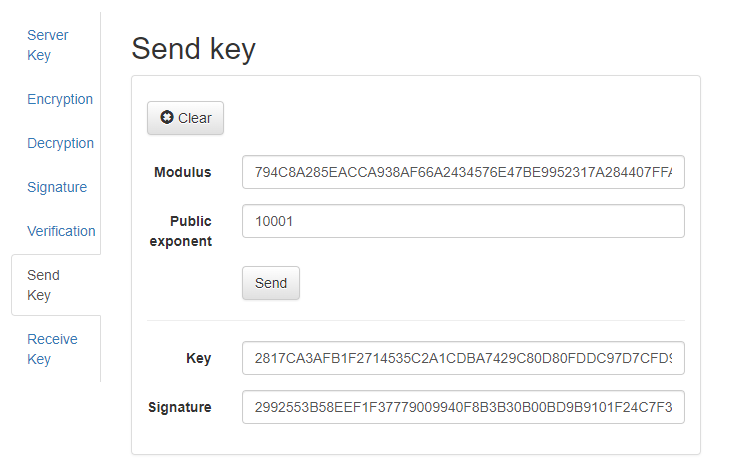
  


Шифрування повідомлення для сервера:  


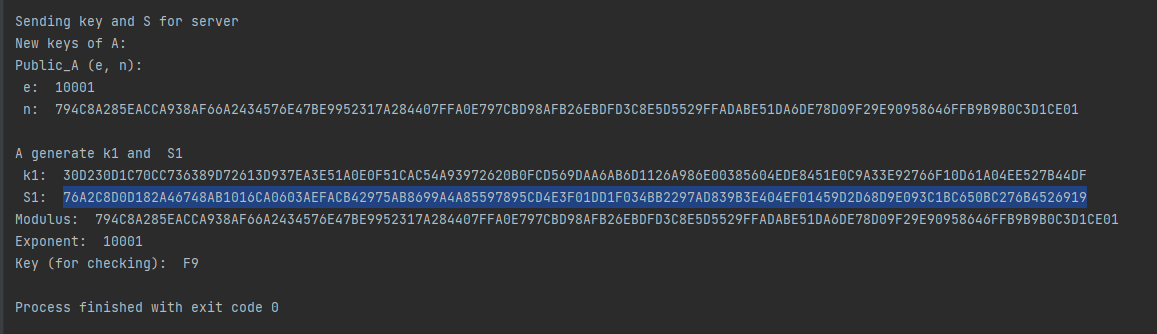
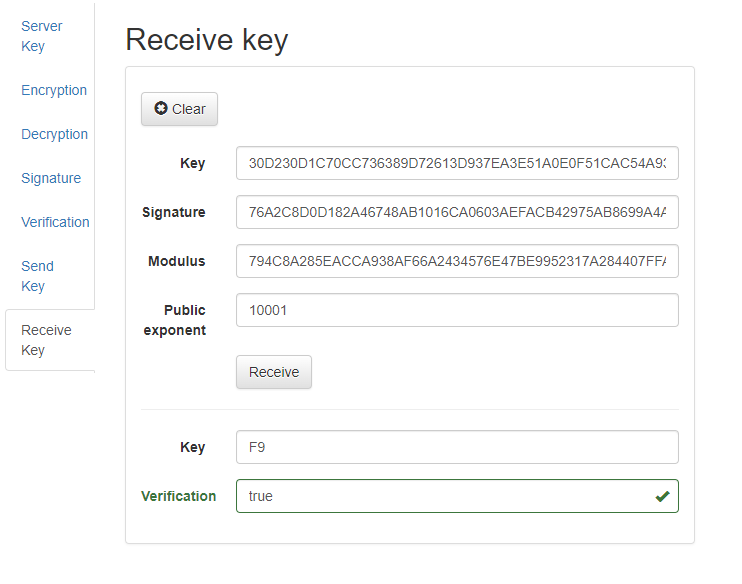
Створення цифрового підпису сайтом:  
  


Створення цифрового підпису нами та перевірка на сайті:  




Отримання ключа від сайту:  




Надсилання ключа на сайт:  
  
Бачимо, що модуль (відкритий ключ) не змінився, отже виконується умова розсилання ключів, в іншому випадку модуль би змінився на новий.  
  
Верифікація пройдена, отже все працює правильно.

Висновки:

В ході виконання даної лабораторної роботи, ми ознайомились з криптосистемою RSA, алгоритмом цифрового підпису та з методами генерації параметрів для асиметричних криптосистем. Під час виконання роботи набули практичних навичок роботи з даними алгоритмами, розібрали принципи та умови даної криптосистеми, а також отримали додаткові знання по алгоритму пошуку простих чисел.