КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

***ФБ-13 Владислав Садохін та Данило Розумовський***

**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.

**Код**

import random  
import hashlib  
import string  
  
alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"  
  
  
def ext\_gcd(a, b):  
 if a == 0:  
 return (b, 0, 1)  
 else:  
 g, x, y = ext\_gcd(b % a, a)  
 return (g, y - (b // a) \* x, x)  
  
  
def mod\_inverse(a, m):  
 g, x, y = ext\_gcd(a, m)  
 if g != 1:  
 return None # Оберненого елемента не існує  
 else:  
 return (x % m + m) % m  
  
  
def trial\_division(n):  
 if n < 2:  
 return False  
 for i in range(2, 100):  
 if n % i == 0:  
 return False  
 return True  
  
  
def is\_prime\_miller\_rabin(n, k=10):  
 if n <= 1:  
 return False  
 if n <= 3:  
 return True  
  
 def find\_s\_d(n):  
 s = 0  
 while n % 2 == 0:  
 s += 1  
 n //= 2  
 return s, n  
  
 s, d = find\_s\_d(n - 1)  
  
 for i in range(k):  
 a = random.randint(1, n - 1)  
 x = pow(a, d, n)  
 if x == 1 or x == n - 1:  
 continue  
 else:  
 for j in range(s - 1):  
 x = pow(x, 2, n)  
 if x == n - 1:  
 break  
 if x == 1:  
 return False  
 if x == n - 1:  
 continue  
 return False  
  
 return True  
  
  
def generate\_random\_prime\_number(min\_value, max\_value):  
 while True:  
 n = random.randint(min\_value, max\_value)  
 if trial\_division(n):  
 if is\_prime\_miller\_rabin(n):  
 return n  
 else:  
 print(f"Кандидат що не пройшов: {n}")  
  
  
def decimal\_to\_binary(decimal\_number):  
 binary\_representation = bin(decimal\_number)[2:]  
 return binary\_representation

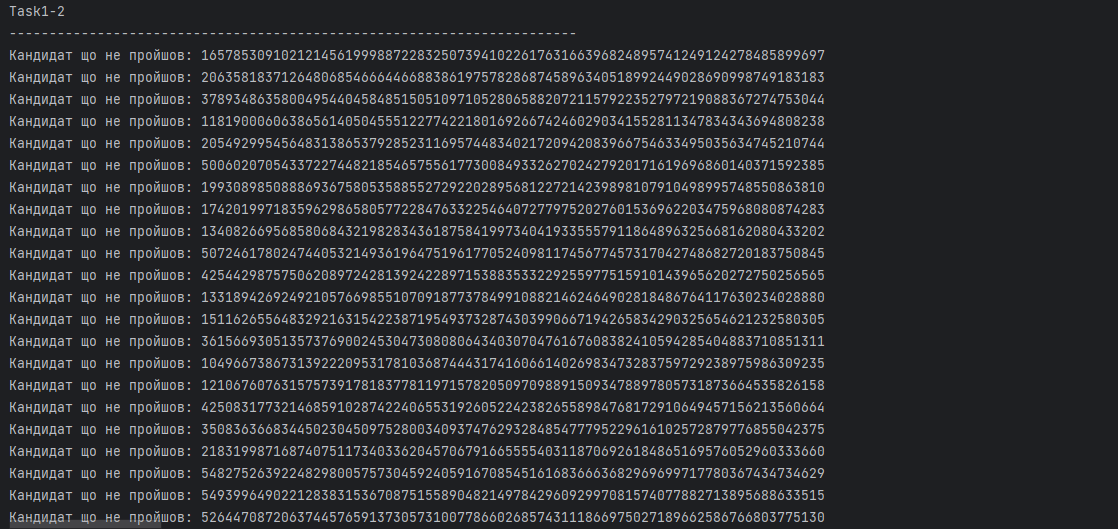
2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента А, 1 p і q1 – абонента B.

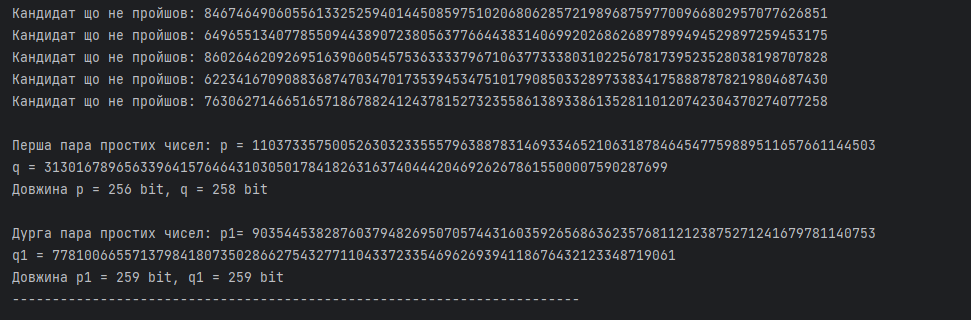
**Код**

print("Task1-2")  
print("-----------------------------------------------------------------------")  
p1 = generate\_random\_prime\_number(100000000000000000000000000000000000000000000000000000000000000000000000000000, 555555555555555555555555555555555555555555555555555555555555555555555555555555)  
q1 = generate\_random\_prime\_number(100000000000000000000000000000000000000000000000000000000000000000000000000000, 555555555555555555555555555555555555555555555555555555555555555555555555555555)  
  
p2 = generate\_random\_prime\_number(555555555555555555555555555555555555555555555555555555555555555555555555555555, 999999999999999999999999999999999999999999999999999999999999999999999999999999)  
q2 = generate\_random\_prime\_number(555555555555555555555555555555555555555555555555555555555555555555555555555555, 999999999999999999999999999999999999999999999999999999999999999999999999999999)  
  
print(f"Перша пара простих чисел: p = {p1}\nq = {q1}")  
print(f"Довжина p = {len(decimal\_to\_binary(p1))} bit, q = {len(decimal\_to\_binary(q1))} bit\n")  
print(f"Дурга пара простих чисел: p1= {p2}\nq1 = {q2}")  
print(f"Довжина p1 = {len(decimal\_to\_binary(p2))} bit, q1 = {len(decimal\_to\_binary(q2))} bit")  
print("-----------------------------------------------------------------------\n")

**Результати:**

Так як чисел що не пройшли перевірку дуже багато, то всі не влізли в скріншоти тому ми тільки початок та кінець цих чисел вставили в протокол, також в кінці числа які є прості та їхня довжина, яка має бути мінімум 256 біт





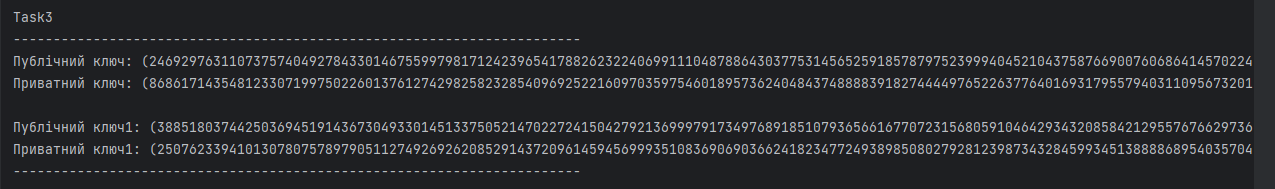
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p,q) та відкритий ключ (n,e) . За допомогою цієї функції побудувати схеми RSA для абонентів А і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n) , ( , ) 1 n1 e та секретні d і d1 .

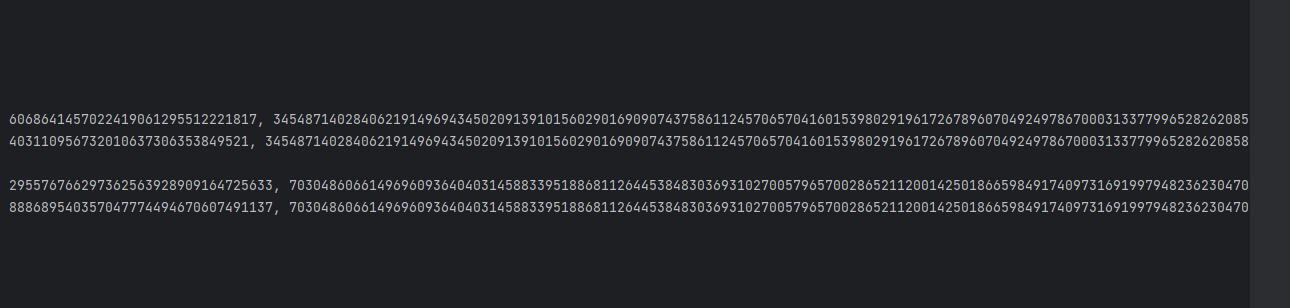
**Код**

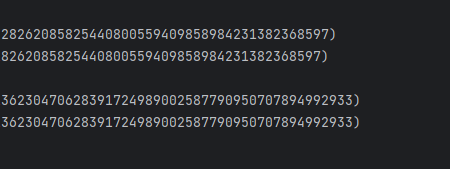
def GenerateKeyPair(p, q):  
 n = p\*q  
 totient = (p-1)\*(q-1)  
 e = random.randint(2, totient - 1)  
 (g, x, y) = ext\_gcd(e, totient)  
 while g > 1:  
 e = random.randint(2, totient-1)  
 (g, x, y) = ext\_gcd(e, totient)  
 d = mod\_inverse(e, totient)  
 public\_key = (e, n)  
 private\_key = (d, n)  
 return public\_key, private\_key

print("Task3")  
print("-----------------------------------------------------------------------")  
key1 = GenerateKeyPair(p1, q1)  
key2 = GenerateKeyPair(p2, q2)  
  
e1, d1 = key1  
e2, d2 = key2  
  
print(f"Публічний ключ: {e1}\nПриватний ключ: {d1}\n")  
print(f"Публічний ключ1: {e2}\nПриватний ключ1: {d2}")  
print("-----------------------------------------------------------------------\n")

**Результати:**







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.

**Код**

def ConvertToInt\_en(text, hash=False):  
 num = 0  
 if not hash:  
 for char in text:  
 num = num \* 50000 + ord(char)  
 else:  
 for char in text:  
 num = num \* 256 + ord(char)  
 return num  
  
def ConvertToText\_en(number):  
 text = ""  
 while number > 0:  
 char\_code = number % len(alphabet)  
 text = alphabet[char\_code] + text  
 number //= len(alphabet)  
 return text  
  
def ConvertToInt\_dec(text):  
 num = 0  
 for char in text:  
 if char in alphabet:  
 num = num \* len(alphabet) + alphabet.index(char)  
 return num  
  
  
def ConvertToText\_dec(num, hash=False):  
 text = ""  
 if not hash:  
 while num > 0:  
 char\_code = num % 50000  
 text = chr(char\_code) + text  
 num //= 50000  
 else:  
 while num > 0:  
 char\_code = num % 256  
 text = chr(char\_code) + text  
 num //= 256  
 return text  
  
  
def GenerateKeyPair(p, q):  
 n = p\*q  
 totient = (p-1)\*(q-1)  
 e = random.randint(2, totient - 1)  
 (g, x, y) = ext\_gcd(e, totient)  
 while g > 1:  
 e = random.randint(2, totient-1)  
 (g, x, y) = ext\_gcd(e, totient)  
 d = mod\_inverse(e, totient)  
 public\_key = (e, n)  
 private\_key = (d, n)  
 return public\_key, private\_key  
  
  
def split\_blocks(text, max\_block\_size):  
 blocks = []  
  
 for i in range(0, len(text), 32):  
 block = text[i:i + 32]  
 blocks.append(block)  
  
 for i in range(len(blocks)):  
 while len(decimal\_to\_binary(ConvertToInt\_en(blocks[i]))) > max\_block\_size:  
 half\_size = len(blocks[i]) // 2  
 block1 = blocks[i][:half\_size]  
 block2 = blocks[i][half\_size:]  
 blocks[i] = block1  
 blocks.insert(i + 1, block2)  
  
 return blocks  
  
  
def Encrypt(input\_text, key, hash=False):  
 e, n = key  
 if hash:  
 text\_in\_num = ConvertToInt\_en(input\_text, hash)  
 number\_encrypted = pow(text\_in\_num, e, n)  
 text\_encrypted = ConvertToText\_en(number\_encrypted)  
 return text\_encrypted  
  
 mes\_len\_in\_num = len(decimal\_to\_binary(ConvertToInt\_en(input\_text)))  
  
 if mes\_len\_in\_num >= len(decimal\_to\_binary(n)):  
 blocks = split\_blocks(input\_text, len(decimal\_to\_binary(n)) - 1)  
 enc\_text = []  
 for block in blocks:  
 text\_in\_num = ConvertToInt\_en(block)  
 number\_encrypted = pow(text\_in\_num, e, n)  
 text\_encrypted = ConvertToText\_en(number\_encrypted)  
 enc\_text.append(text\_encrypted)  
 return enc\_text  
 text\_in\_num = ConvertToInt\_en(input\_text)  
 number\_encrypted = pow(text\_in\_num, e, n)  
 text\_encrypted = ConvertToText\_en(number\_encrypted)  
 return text\_encrypted  
  
  
def Decrypt(encrypted\_text, key, hash=False):  
 d, n = key  
 if isinstance(encrypted\_text, list):  
 dec\_text = ""  
 for block in encrypted\_text:  
 number\_encrypted = ConvertToInt\_dec(block)  
 number\_decrypted = pow(number\_encrypted, d, n)  
 text\_decrypted = ConvertToText\_dec(number\_decrypted, hash)  
 dec\_text += text\_decrypted  
 return dec\_text  
 number\_encrypted = ConvertToInt\_dec(encrypted\_text)  
 number\_decrypted = pow(number\_encrypted, d, n)  
 text\_decrypted = ConvertToText\_dec(number\_decrypted, hash)  
 return text\_decrypted  
  
  
def Sign(input\_text, my\_private\_key, public\_key):  
 sha256\_hash = hashlib.sha256()  
 sha256\_hash.update(input\_text.encode('utf-8'))  
 sha256\_hash\_value = sha256\_hash.hexdigest()  
 print(sha256\_hash\_value)  
 hash\_encrypted\_with\_prv = Encrypt(sha256\_hash\_value, my\_private\_key, True)  
 print(f"Signature: {hash\_encrypted\_with\_prv}")  
  
 hash\_encrypted\_with\_pbl = Encrypt(hash\_encrypted\_with\_prv, public\_key)  
 print(f"Signature encrypted with public key: {hash\_encrypted\_with\_pbl}")  
 return hash\_encrypted\_with\_pbl  
  
  
def Verify(input\_text, sign, public\_key, my\_private\_key):  
 hash\_decrypted\_with\_prv = Decrypt(sign, my\_private\_key)  
 print(f"Signature decrypted with private key: {hash\_decrypted\_with\_prv}")  
 hash\_decrypted\_with\_pbl = Decrypt(hash\_decrypted\_with\_prv, public\_key, True)  
 print(f"Hash decrypted with public key(signature is verified): {hash\_decrypted\_with\_pbl}")  
  
 sha256\_hash\_cal = hashlib.sha256()  
 sha256\_hash\_cal.update(input\_text.encode('utf-8'))  
 sha256\_hash\_value = sha256\_hash\_cal.hexdigest()  
 print("Hash calculated from received message: ", sha256\_hash\_value)  
  
 if hash\_decrypted\_with\_pbl == sha256\_hash\_value:  
 return True  
 else:  
 return False

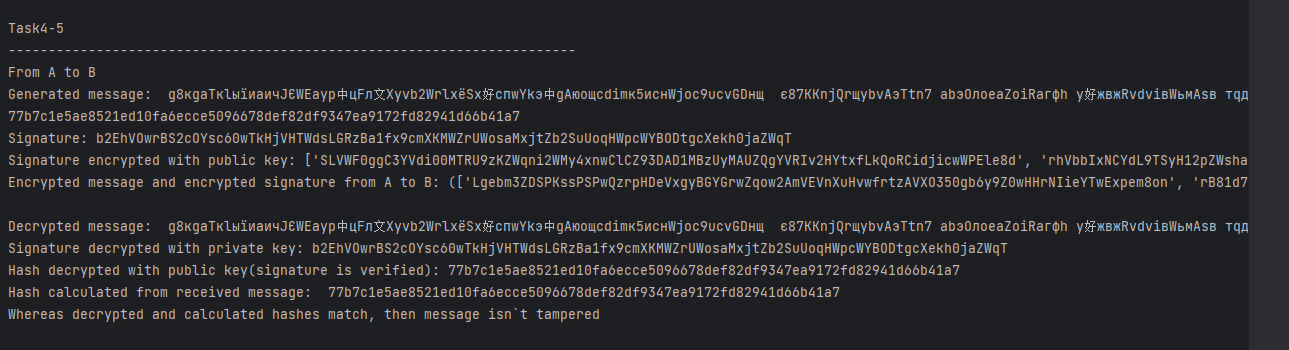
5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді 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.

**Код**

def create\_message\_for\_abonent(public\_key, length, my\_private\_key):  
 characters = (string.ascii\_letters + string.digits + ' ' + 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя' + 'ґєіїҐЄІЇ' + '你好' + '中文' + ' ')  
 random\_message = ''.join(random.choice(characters) for i in range(length))  
 print("Generated message: ", random\_message)  
 encrypted\_message = Encrypt(random\_message, public\_key)  
 signed\_message = Sign(random\_message, my\_private\_key, public\_key)  
 message = (encrypted\_message, signed\_message)  
 return message  
  
  
def receive\_message\_from\_abonent(message, my\_private\_key, public\_key):  
 enc\_mes, signed\_mes = message  
 decrypted\_message = Decrypt(enc\_mes, my\_private\_key)  
 print("Decrypted message: ", decrypted\_message)  
 ver\_mes = Verify(decrypted\_message, signed\_mes, public\_key, my\_private\_key)  
 if ver\_mes:  
 print("Whereas decrypted and calculated hashes match, then message isn`t tampered\n")  
 else:  
 print("Message is tampered by someone\n")

Опис кроків протоколу конфіденційного розсилання ключів з підтвердженням справжності:  
Спочатку генерується повідомлення, потім воно зашифровується з публічним ключем отримувача, потім повідомлення(не зашифроване) шифрується з моїм приватним ключем(підписується) потім зашифроване моїм приватним ключем повідомлення, шифрується публічним ключем отримувача, щоб тільки він міг подивитися цей підпис. Далі зашифроване повідомлення та зашифрований підпис повертаються функцією як кортеж (зашифроване повідомлення, ашифрований підпис).

**Результати:**



Так як зашифровані повідомлення та підпис не влязли в скріншот, то ми написали їх нижче, спочатку від A до B, потім від B до A, інші дані видно на скріншоті, тому ми їх не переписували

**Encrypted message and encrypted signature from A to B:** (['Lgebm3ZDSPKssPSPwQzrpHDeVxgyBGYGrwZqow2AmVEVnXuHvwfrtzAVXO350gb6y9Z0wHHrNIieYTwExpem8on', 'rB81d7sd7tXvLluIald4GrkEeVUX71IQuONLTYQe6MuvE710YsLY8GY4CbMCnJLbBzlOXpgmHsxwhuw8squOwMn', 'MkSVrHUsD6eoVWxqa1YBqW26vHTsyX0IlDn8oK3o1SZqfr5w3YlJglL6nkbhEcAN21utoRSENQpp39gZF2GmmoZ', 'jeGcSikkV3J9ZlKzzjts1O0yiFzoyP0vbntxjG5TF5rrVjZd7nIZFW8Ai7JusEPuYYT2AnBU9xvUTKVVZ5UaVmA', 'JpZOscHDWy56ZCXJNPH5LLFrNRPgqn9QCKP2b3QikrHeHfaNqJMQNNnLUoDNpMMUfSbELHTlS2SI8hnJZIW0yMT', 'Mx5BpCJS3nEdKdM93jdNfiesF4n6DHVWNaoG2K0WoaFzRepR6zyV2FyAeZoLcqAtK98f1yDMfC8vr3kzzDCqmsY', 'pBxLAlSPtgmAG6aGRqwTXgeaxa4uFRPlBzu4F1e3KxpK5FNkmXyXw5jLGG9PYa5AFyJsm6GJ5QtRxyUZOkeCC3b', 'w3kdkBbfmJBJQlGFnkwSMciTQt5XPGgG9ugX1XL3YajlDibn6kc76iv0NHQltidR9sVKYVHpo56ExwAQXnQgKeZ', 'nloFHwaG9aJIyw4RRKE03eL0o06Daz11kG6Gy0ySpG9SPuTGyxNLY2ISkgVH0ETugj6pyyuZhHkJBeuEwqLtsyq', 'uKIzGMAyiMzmtBsK4fA93dgdssjf0hpyfAGd9LG0mXscoB8O6DO4nlvfxCbTA6l4EY9AmtJPTXwoJ1m1oSvHgjp'], ['SLVWF0ggC3YVdi00MTRU9zKZWqni2WMy4xnwClCZ93DAD1MBzUyMAUZQgYVRIv2HYtxfLkQoRCidjicwWPEle8d', 'rhVbbIxNCYdL9TSyH12pZWshaZOjUb3BWzPySIS0Az091zznd8OnosMNPNBvMbgCZJxTx4sffRLNJieon4HzoMV', 'LqscU7qGwVd8lARAIlFKlHnlKlS40BnjgT0P9iwlGfUUddWqMwu5tkpy6TpVFKkD12wZxqVGQcIKX2ZhFTlnXZa'])

****

**Encrypted message and encrypted signature from B to A:** (['sAcylMJoyFefC6AwKpJSSrnKJ7gCjSzAPexCKFS1eXI6KuP23jcvdfFSWncXh8thrL4A3IfLVlr6k1mvPr2KvN', 'JsiUxa03vwgsxSfPilULMm8Ws4RirUs7YnQrjmD3xm7fcWOV9T4BnoQb13CAd4ayd58vaol7aiCI92DHzEFUuH', 'btMpVpMw9LqPu3Cxm6yn0IwGE58S9SHwHsB72fwAMHCdzaAZvjgx5tMVnLzsC93Dmp6v1AJ8fdBkKqzqr3QgOGS', 'u7pphiczy05VQvFe7Ay7BD6RFUFRdoUM2ZNuIpsgFjrkcMKyaDG3UOkez4y02AGizEm11lBepgU75r2agnsQkB', 'bPwgXHZJehfscovtkZMzyeSj81yL6OBMVW4ogxdeDPe6U2492G5rpBKWnFbP1Z3RtAaUUP2QcsLEpMKCmYQXgIc', 'DiyYRieSUQ68JFK3ht1JfG2xgu7Z4uBiCduFhz9Dqe96WOb263tYJGPwr8Yf8eKcNZJWXODGRrI1udgnOpqwcY', '3LrsXarD73DXSJ653xH0xZ37uNhDRFbTerxH13FiXCK5jIjOMYSXtw7qShEuQzYpX0J2nVHy2KIT6T6MHIHxY8', 'bHl24eWHRhf6N2cZIe5th0eDEaGQqQvRnDudhgrmpzQxmnSFN8bKTqkWo8Cn2urphNefgeD5kQ7RHyRYm2sPD53', 'cqSzF4HcEpgc2Qlj2n76LR1lEMSjYd4r5Pdd0t0zI1iDmkySCeljoSWI2AoUVtvX3YWK1ASAlz2WsE22s7z7T8X', '3ZXVup6tbwBO3th44QsbMnJBHTPqK7U2YPSzGfBiwtlslly0vRvqm2dif3oqDnkxqvSMwB7Ss9LNm2pewvsnwt'], ['cmm4ccLUXkLgaXWe83YcLbwRp2E907RtUPrPfXOpvuxapxgEFJ1kln0vBW0ZeGEEUB6cdKexRU93jvUYIkewqBq', 'UZXV7BItcVveh6MlqyaXlpRZAjqSQQWLFHM2rt2WBsPKjV5zKejGKO2Nhxh6zjMb921tmxY3nySmSpDBuTnoPS', 'Prl3bnth6swNm0NPI83AeFwO32y0WklO2lpkwR1m6Rba9vUjwn84E7R8A9KVDIei9i5mtW281g5vZOxPSa2259'])

**Висновки:**У ході виконання даної роботи було досягнуто поставленої мети, яка передбачала ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Практичне засвоєння цих тестів та методів генерації ключів дозволило глибше зрозуміти принципи роботи криптосистеми RSA, що є однією з ключових технологій у сфері інформаційної безпеки. Також, робота надає можливість вивчити систему захисту інформації на основі криптосхеми RSA. Встановлення концепції захисту інформації з використанням даної системи, організація засекреченого зв'язку та електронного підпису, є важливим етапом для забезпечення конфіденційності та цілісності обмінюваної інформації.