КРИПТОГРАФІЯ КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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.*

ФБ-11 «Проект Нівечення»

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 $pq \leq p1q1$; p і q – прості числа для побудови ключів абонента A, 1 p 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 A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e,n) , (,) 1 n1 e та секретні d і d1 .

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 и 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 $0 < k < n$.

Хід роботи

1.Реалізував функцію пошук випадкового простого числа(тут для прикладу взяв к-ть ітерацій k = 30:

```
def generate_prime():
    prime = random.getrandbits(256)
    if prime % 2 == 0:
        prime += 1
    while not millerabin_test(prime, 30):
        prime += 2

    return prime
```

Генерується випадкове число, розміром 256 біт, якщо число парне, додається один.Якщо число не проходить тест, воно збільшується на два та цикл повторюється поки число не пройде тест.

Та тест Міллера-Рябіна:

```
def millerabin_test(p, k=30):
    if p == 2 or p == 3: #перевірка на особливі випадки(2 і 3 - прості числа)
        return True
    if p % 2 == 0 or p < 2: #перевірка на парність
        return False
    s, d = 0, p-1
    while d % 2 == 0:
        s += 1
        d //= 2

    for _ in range(k):
        x = random.randint(1, p - 1)
        gcd1 = pow(x, d, p)
        if gcd1 == 1 or gcd1 == p - 1:
            continue

        for _ in range(1, s):
            gcd1 = pow(gcd1, 2, p)
            if gcd1 == p - 1:
                break
        else:
            return False # p - є складеним

    return True # p - псевдопросте
```

Найбільш складним було розуміння роботи самого тексту, так як потрібно було зрозуміти перехід від псевдопростого числа до простого.(Якщо

сильнопросте за всіми основами – просте, якщо хоча б за однією основою ні – складене), k – кількість ітерацій.

2.За допомогою функції генерації простого числа згенерував дві пари простих чисел, з умовою, якщо пара абонента А буде більша за пару В, числа перегенеровувалися.

```
def GenerateKeyPair():
    p, q, p1, q1 = 0, 0, 0, 0
    while True:
        p = generate_prime()
        q = generate_prime()
        p1 = generate_prime()
        q1 = generate_prime()
        if p*q <= p1*q1:
            break
    return p, q, p1, q1
```

3.Функція генерації ключів RSA, виконується перевірка чи е взаємопросте, за розширеним алгоритмом Евкліда, далі вираховується закритий ключ d, по оберненому е(відкритим ключем) за модулем р-ту функції Ойлера.

```
def RSA_Keys(p, q):
    n = p*q
    fo = (p-1)*(q-1) #функція Ойлера
    while True:
        e = random.randint(2, fo-1)
        if ext_gcd(e, fo)[0] == 1:
            d = modInverse(e, fo)
            break

    return (e, n), (d, p, q)
```

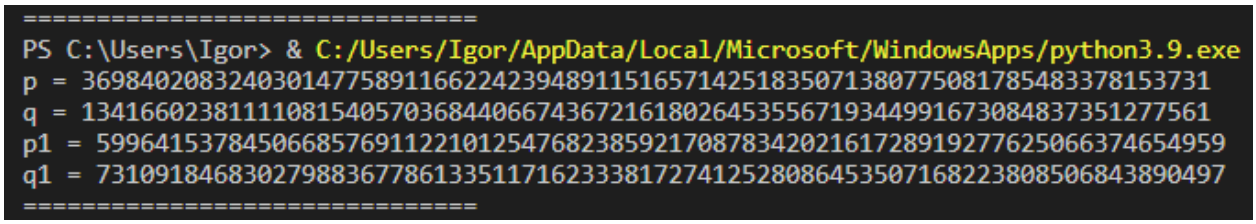
Функція modInverse, що це вираховує:

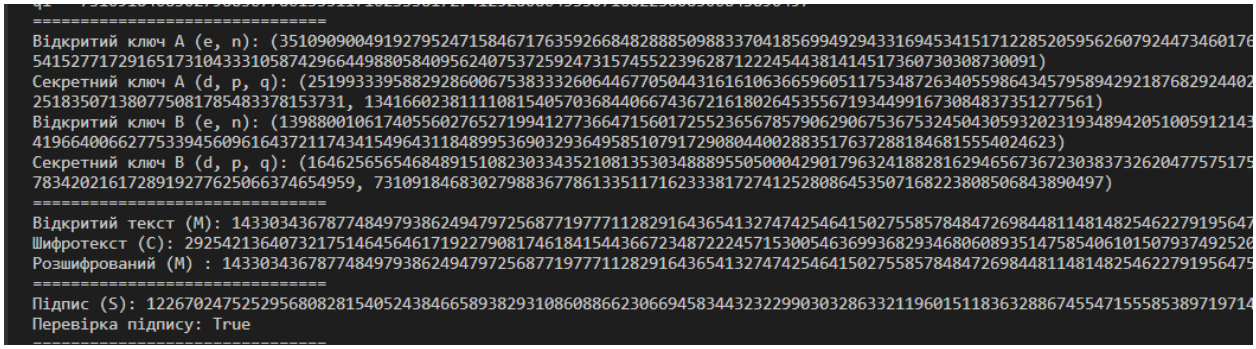
```python
def modInverse(num, mod):
    a, b = num, mod
    x, y, u, v = 0, 1, 1, 0
    while a != 0:
        q, r = divmod(b, a)
        m, n = x - u * q, y - v * q
        b, a, x, y, u, v = a, r, u, v, m, n
    return x % mod
```

4.Реалізовані функції:

```python
def Encrypt(message, e, n):
    encrypted_message = pow(message, e, n)
    return encrypted_message


def Decrypt(encrypted_message, d, p, q):
    decrypted_message = pow(encrypted_message, d, p*q)
    return decrypted_message


def Sign(message, secret_key):
    sign = pow(message, secret_key[0], secret_key[1]*secret_key[2])
    return sign


def Verify(signed_message, sign, public_key):
    return signed_message == pow(sign, public_key[0], public_key[1])


def SendKey(k, secret_key, public_key_B):
    k1 = Encrypt(k, public_key_B[0], public_key_B[1])
    s = Sign(k, secret_key)
    s1 = Encrypt(s, public_key_B[0], public_key_B[1])
    print(f"k1: {k1} \ns: {s}\ns1: {s1}")
    return (k1, s1)


def RecieveKey(k_list, secret_key, public_key_A):
    k = Decrypt(k_list[0], secret_key[0], secret_key[1], secret_key[2])
    s = Decrypt(k_list[1], secret_key[0], secret_key[1], secret_key[2])
    sign_verification = Verify(k, s, public_key_A)
    print(f"Функція отримання ключа: ")
    print(f"k: {k} \ns: {s}\ncheck: {sign_verification}")
    return (k, sign_verification)
```

5.Перевірка роботи





Отриманні ключі:

Відкритий ключ A (e, n):
(351090900491927952471584671763592668482888509883370418569949294331694534151712285205956260792447346017627601815230215005013948379724701394850033623567809,
496199901963080132004349529234736757798994268875415277172916517310433310587429664498805840956240753725924731574552239628712224544381414517360730308730091)

Секретний ключ A (d, p, q):
(251993339588292860067538333260644677050443161610636659605117534872634055986434579589429218768292440210086314739708392280031674466341488995335175355335489,
369840208324030147758911662242394891151657142518350713807750817854833781537 31,
134166023811110815405703684406674367216180264535567193449916730848373512775 61)

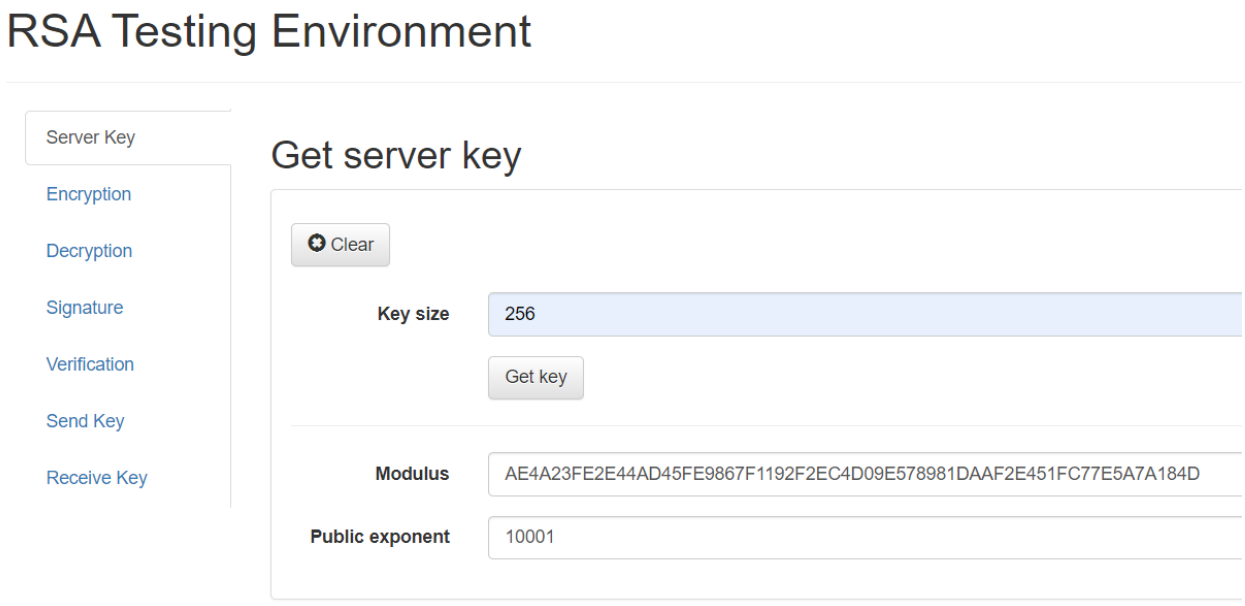
Відкритий ключ B (e, n):
(139880010617405560276527199412773664715601725523656785790629067536753245043059320231934894205100591214399198929997681282664068126633878767741366682679214 3,
438393039339299097273237750984075381569508159041966400662775339456096164372117434154964311848995369032936495851079172908044002883517637288184681555402462 3)

Секретний ключ B (d, p, q):
(16462565654684891510823033435210813530348889550500042901796324188
28162946567367230383732620477575175727471953736645621491855187092
81424426497382809635959 9,
59964153784506685769112210125476823859217087834202161728919277625
066374654959,
73109184683027988367786133511716233381727412528086453507168223808
506843890497)

```
============================
Функція відправки ключа:
k: 362658762228479991813238391156777067149550979196260549267437004313962719804451110689694280787092928245000104720688231916571335104004281559648144
k1: 202403821272071135455719020479421966730938314819036521965976564616458235470729524881009485758706825811639068044190636286030599397041768111373230
s: 599445936179783474349070509936741675377707117543185803917068713213172848187182773193101815832692463715684756033777795319371024508069930175358214
s1: 211076016125128406615277022614324443160867185712932434036057499600831631766546356684135889459355874150817405596493795760230520680175638208554568
Відправлений ключ (k1, S1): (202403821272071135455719020479421966730938314819036521965976564616458235470729524881009485758706825811639068044190636
857129324340360574996008316317665463566841358945935587415081740559649379576023052068017563820855456889504866 51)
Функція отримання ключа:
k: 362658762228479991813238391156777067149550979196260549267437004313962719804451110689694280787092928245000104720688231916571335104004281559648144
s: 599445936179783474349070509936741675377707117543185803917068713213172848187182773193101815832692463715684756033777795319371024508069930175358214
check: True
Отриманий ключ (k, sign_verification): (362658762228479991813238391156777067149550979196260549267437004313962719804451110689694280787092928245000104
============================
```

6.Перевірка шифрування за допомогою онлайн серверу(http://asymcryptwebservice.appspot.com):

Генерую публічний ключ:



Тест – Encryption(перевірка ф-ї дешифрування):

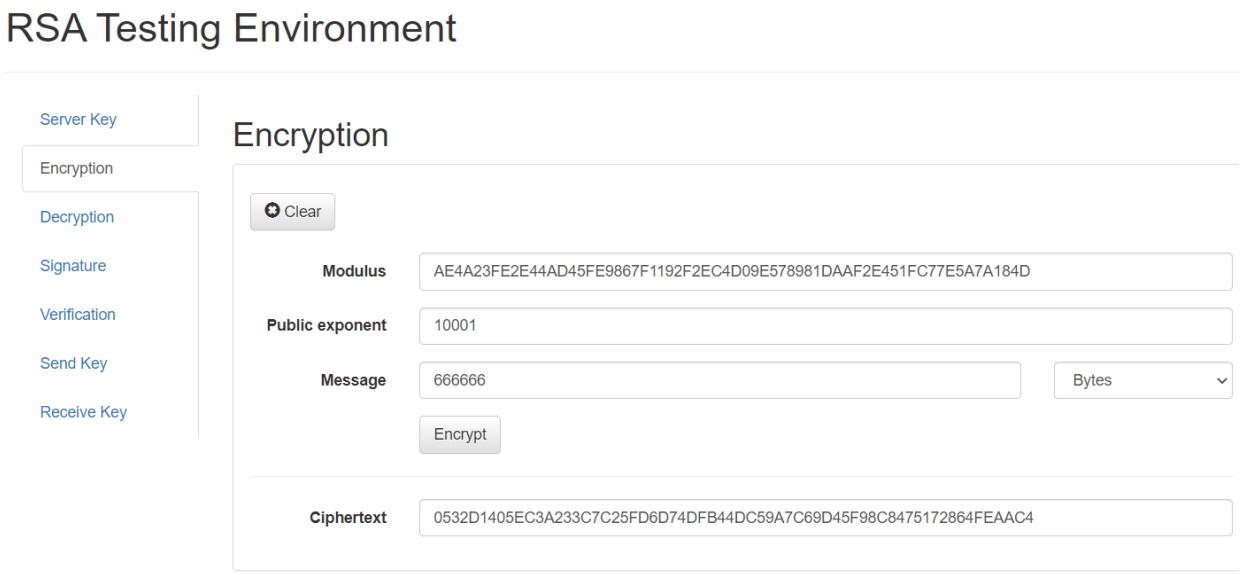
За допомогою алгоритму генерую повідомлення та шифрую його локально за допомогою публічного ключа сервера(e – Public exponent):

```
print(f"Test -- Encryption")
server_key = 'AE4A23FE2E44AD45FE9867F1192F2EC4D09E578981DAAF2E451FC77E5A7A184D'
n = int(server_key, 16)
print(f"n: ", n)
public_e = '10001'
e = int(public_e, 16)
print(f"e: ", e)
M1 = random.randint(0, n-1)
M1_hex = hex(M1)[2:]
print(f"Повідомлення_1: ", M1_hex)
encrypted_text = hex(Encrypt(M1, e, n))[2:]
print(f"Зашифрований текст: ", encrypted_text.upper())
```

Шифрування:

```
================================
Test -- Encryption
n:  78833430750103160577215063062802098332682665854464432204738650415522326911053
e:  65537
Повідомлення_1:  6710886
Зашифрований текст:  532D1405EC3A233C7C25FD6D74DFB44DC59A7C69D45F98C8475172864FEAAC4
```

Тепер шифруємо повідомлення на сервері:



Результат – шифротекст, однаковий, тест успішний.

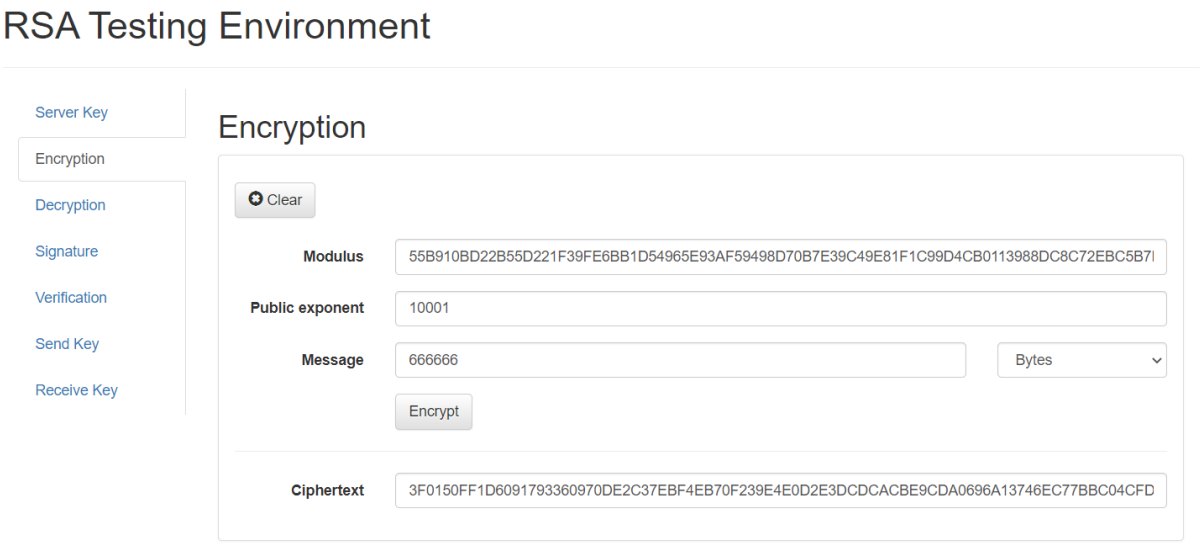
Тест – Decryption:

Для цього тесту була створена функція-дублікат для створення RSA ключів, з єдиним нововведенням, що змінна e в нас вже є.

```
def RSA_Keys_2(p, q):
    n = p*q
    fo = (p-1)*(q-1) #функція Ойлера
    while True:
        e = int('10001',16)
        if ext_gcd(e, fo)[0] == 1:
            d = modInverse(e, fo)
            break

    return (e, n), (d, p, q)
#Генерує пари ключів
p2, q2, p3, q3 = GenerateNumPair()
```

Генеруємо пари ключів A та B, B знадобиться у двох останніх тестах(не знадобився(().Щоб було зручніше працювати з алгоритмом, і він кожного разу не генерував нові ключі, значення p2, q2, p3, q3 були занесені в змінні.

```
================================
Test -- Decryption
p2:  53942798491662374014013154466993223980861401413964560391950983741994658392791
q2:  83230260726133430091891278119226286536741447953693128832336769322857561796643
p3:  83145974407219291949006645963343446718067013046388947315900544481589237159467
q3:  98218213215060430385367454899289751131197639009554022820863489091984936291813
```



Дешифрування:

```
C2 = int('3F0150FF1D6091793360970DE2C37EBF4EB70F239E4E0D2E3DCDCACBE9CDA0696A13746EC77BBC04CFD63DF933CC5EAD496A3EF8B51895F40CF5734C7C3D48F7',16)
d_2 = secret_key2_A[0]
M2 = Decrypt(C2, d_2, p2, q2)
print(f"Розшифроване повідомлення: ", hex(M2).upper()[2:])
print(f"==============================")
```

```
==============================
Test -- Decryption
p2 = 539427984916623740140131544669932239808614014139645603919509837419946583927 91
q2 = 832302607261334300918912781192262865367414479536931288323367693228575617966 43
p3 = 831459744072192919490066459633434467180670130463889473159005444815892371594 67
q3 = 982182132150604303853674548992897511311976390095540228208634890919849362918 13
Відкритий ключ A (hex): (65537, '55B910BD22B55D221F39FE6BB1D54965E93AF59498D70B7E39C49E81F1C99D4CB0113988DC8C72EBC5B7E2B487166C5E29873180C9F012919E59077F8EF4EB65')
Секретний ключ A (d, p, q): (29311632584750743035934604702041911168076390401069375941703687468233601085110900073494776334213430492814978719261414538730084458915132361413964560391950983741994658392791, 832302607261334300918912781192262865367414479536931288323367693228575617966 43)
==============================
Розшифроване повідомлення:  666666
==============================
```

Як можна переконатися, шифротекст з серверу було розшифровано правильно.

Тест – Sign:

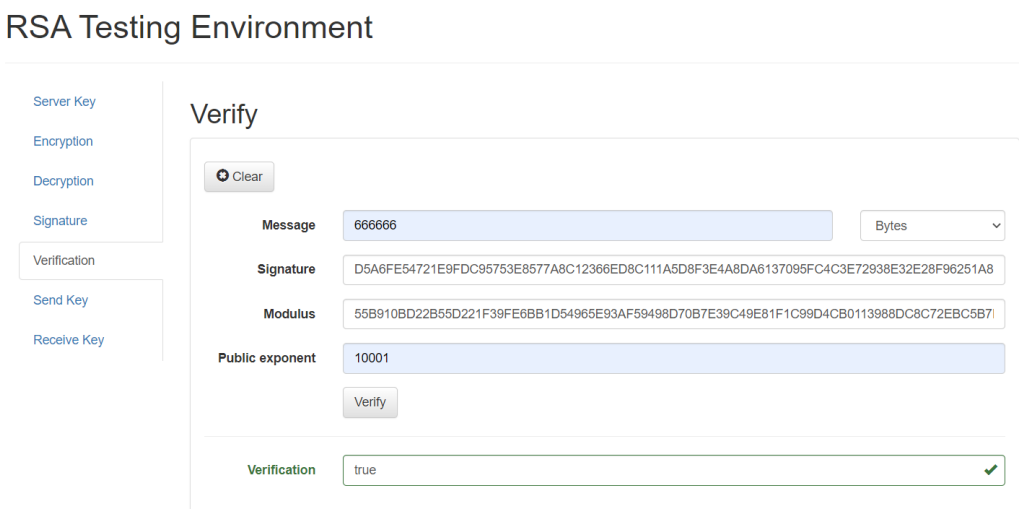
Тепер підпишемо наше повідомлення M1 локально, і перевіримо підпис на сервері:

```
###3
print(f"Test -- Sign")
print("Відкритий ключ A (hex):", (e, n2_hex))
print(f"Секретний ключ A(d, p, q): ", secret_key2_A)
s3 = Sign(M1, secret_key2_A)
print(f"Підпис 2(s3): ", hex(s3).upper()[2:])
```

Значення секретного та публічного ключів було взято з другого тесту – secret_key2_A, public_key2_A(e, n2_hex)

```
==============================
Test -- Sign
Відкритий ключ A (hex): (65537, 'AC9A7438C14A0F913DE34105499D535787A7656BA285ED2032608230BA3B80992A3419580524208752EE5395
Секретний ключ A(d, p, q):  (272784001455904090831712152064138676316769716648942510608616186495422245038783524322071522147938497598848248084631649188233629, 110643810509453761315545838266923295616626296235349237680876597963220162024747)
Підпис 2(s3):  90188906AD497DC6EBD8E9FF7A1C180EB26C68777E9EB72F0BB5D69CF86AE7A9ECF543734121887C320E94C687B0FADCE554874B04
==============================
```

Перевірка підпису:



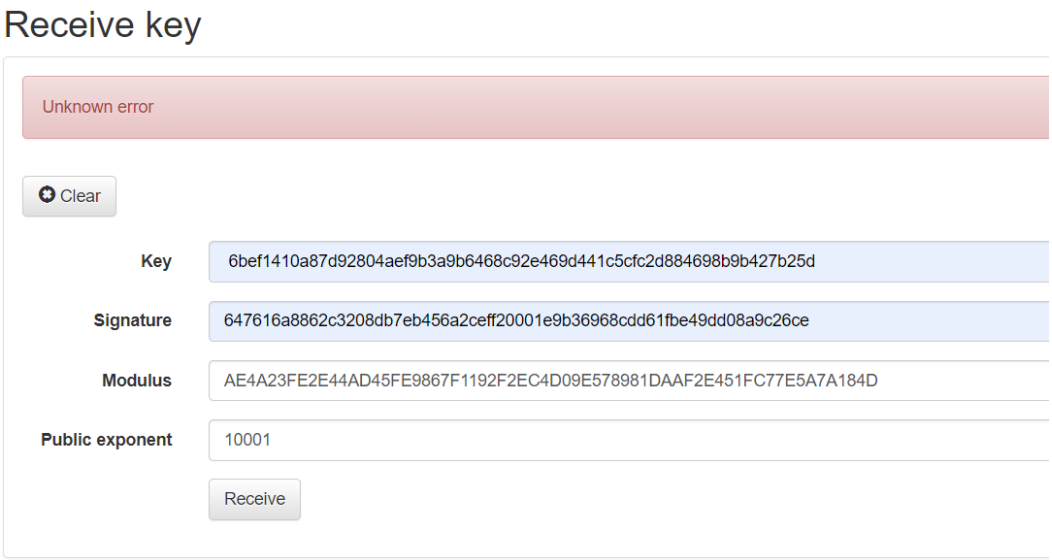
Тест – Verify:
Беремо текст з першого тесту – M1, і публічний ключ A – public_key1_A(e,n)(згенерованийсервером), підписуємо його на сервері і перевіряємо підпис локально.

Верифікація:

```
print(f"Test -- Verify")
server_sign = int("1829319C6C3166C3E57F6159B23236AE46F45E94E6B2FC3E1F96F4A95282F9DC", 16)
server_sign_hex = hex(server_sign)[2:]
print(f"server_sign_hex: ", server_sign_hex)
print(Verify(M1, server_sign, public_key1_A))
print(f"================================")
```

```
================================
Test -- Verify
server_sign_hex:  1829319c6c3166c3e57f6159b23236ae46f45e94e6b2fc3e1f96f4a95282f9dc
True
================================
```

Труднощі: Найбільшою проблемою було реалізувати за допомогою серверу перевірки функцій Send Key та Receive Key, і в мене ці пункти не вийшли, самі тести роботи функцій без серверу є.



Висновок: В ході виконання лабораторної роботи було отримано навички роботи з RSA ключами, шифруванням та дешифруванням за допомогою відкритих та секретних ключей, стало більш зрозуміло роботу алгоритму Міллера-Рабіна.