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 №4

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

Виконали: студенти гр. ФБ-12 Головко М. С. і Марчук І. С.

Київ – 2023

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 A, 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 A 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 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. Написали функцію генерації випадкового простого числа `generate_random_number(bits)`, як аргумент вона приймає кількість бітів потрібного числа. В ній на початку перевіряється, чи число парне чи непарне (очевидно, що якщо число парне, воно ділиться на 2 і тому воно складене, не підходить), а потім число перевіряється на простоту за допомогою імовірнісного тесту Міллера-Рабіна, який реалізован у функції miller_rabin(p, iteration=500). Також ми використовували вже написану нами в минулій лабі функцію пошуку НСД gcd(a, b). Тест Міллера-Рабіна реалізован за допомогою теоретичних відомостей, наданих в методичних вказівках. Для

генерації випадкового числа використовували функцію з бібліотеки random – `random.randint()`.

```
PS C:\Users\Igorm> & C:/Python311/python.exe "d:/uni year 3/crypto labs/crypto-23-24/cp4/fb-12_Marchuk_Holovko/c
Random prime: 99793130836274818984114343279722358433646308519748858818280672990649889162107
Random prime: 11127929568753624946869457206266182542540484322555021137982789946470499566787 1
Random prime: 11357427265861368464736377553459099676712651770261280158978573818362452071284 9
Random prime: 78178374225192832808254403414253929254636435230298336345672208431767410557199
Random prime: 10130553359041381894952126255588867863700944440477784922953952350391815343959 3
```

2. Реалізували генерацію пар чисел p, q і p1, q1 у функції `generate_pairs()`, в якій використовується функція `generate_random_number(256)`. При цьому була збережена умова, що pq ≤ p1q1. Таким чином отримали p, q – прості числа для побудови ключів абонента A, і p1, q1 – для абонента B.

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/User/Desktop/university/3 year/crypto labs done/lab4/lab4.py"

A pair:  61675600153025843653586391834302732060027072912650561353349963004051504431139 95714629425256737506080711696228363934363037412542969719864457478139600910919
B pair:  78945749752592380591030632779034924985864479333955764860550620128580414216447 84271327546527674933352250254046545389523045605363003224016728150930332148561
PS C:\Users\User> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/User/Desktop/university/3 year/crypto labs done/lab4/lab4.py"

A pair:  67051468070797828695563450257000444633028819243574745920878109561896939109779 87912892139010312706127910481208772049215037296457222739069066121294625747319
B pair:  70150219341642018304838369849258537820845482216895685369463887932787500794129 98862037744214684172470585350733082762179272819616260227668495006600164220197
PS C:\Users\User>
```

Також додали умову, що якщо число не виявилося простим, треба його просто вивести у консоль з приміткою ‘this value didn’t work out’:

```
this value didnt work out: 947410874245047688342944994236802588222093750467223030906495381078743942226 53
this value didnt work out: 685630800998452972158045860439341941461847050202628512476203054481221101271 09
this value didnt work out: 106653961520710734312880972348942849034236656576115301088306906910275574942605
this value didnt work out: 110907807170583270323694637523114330578254466927734170471790553009025449941243
this value didnt work out: 965571539004361896679508477764571498711901550212401398552408321135438892191 41
this value didnt work out: 112728742431750506146027008890329277861517239839171929326944443147536599514819
this value didnt work out: 111747641638497171903692040396891200513962832382728429370961028839903127617967
this value didnt work out: 111082701374959542410662939174595396532440452383663000595244355741962117358453
this value didnt work out: 654971653858416899258023005651830957531845691533510232197857491257061315277 59
this value didnt work out: 931533878991547052154789408995332742242159987195829726139999709725861951736 83
this value didnt work out: 913757837493959585520098859931607928063454744986647443714381178375694843162 7
this value didnt work out: 736594859752947199393855685509152315216116764821005462694165881652718839297 27
this value didnt work out: 100770751397906434708285355197015881130555363077469081817333963190341835255147
this value didnt work out: 924806860601266210614352391335135001912741400880794257565209207903111648027 27
this worked out: 793242132143897340147670167967458464046097841387644424879974587642273665869 79
```

3. Генерація ключів RSA виконується за наступним алгоритмом:

Абонент A генерує два великих випадкових простих числа p і q, обчислює добуток $n = pq$ і функцію Ойлера $\varphi(n) = (p-1)(q-1)$. Потім A обирає випадкове число e, $2 \le e \le \varphi(n)-1$ таке, що $\gcd(e, \varphi(n)) = 1$, і знаходить для e обернений за $\bmod \varphi(n)$ елемент d: $ed \equiv 1 (\bmod \varphi(n))$.

Зауваження: в сучасних криптографічних системах майже завжди обирається $e = 2^{16} + 1$.

Пара чисел n і e – це відкритий ключ A, а d – секретний (чи особистий) ключ A. При цьому числа p і q абонент A також тримає в таємниці. Не знаючи p і q, не можна обчислити $\varphi(n)$ і знайти секретний ключ d по відкритому e і n.

За цим алгоритмом ми створили функцію generate_rsa_keys(p, q, p1, q1), яка приймає як аргументи згенеровані раніше числа p, q для абонента A і p1, q1 для абонента B. У функції

шукається число n, функція Ойлера від цього числа n, генерується число e, і знаходиться обернений до e елемент d. Ці операції проводяться і над числами для абонента A, і над числами для абонента B.

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/User/Desktop/university/3 year/crypto labs done/lab4/lab4.py"

Public Key A: (75568722511062939622951657912260328358342802547992100419765548131925824315773870436572412168369398892931555554262388127344506275486715270956582747373452853, 368597658147519866863978586494482687377713711508800151269426489605001064411779583807616376904709840870233029870306855199536880699748114353425962174335834 9)
Private Key A: (27309047972074534318939140584203373110892451001577000743783022388835286693173198629890354987638869541119501829332015514677095068437028218299620548903469 49, 68758360946344910424790636964447132729955959602408897916828186774725114171333, 10990477590068274835680870804166074696188214421048211809075374895283774631544 1)
Public Key B: (93958303929923822908167959953498251271629742366962603226030337621883594680433427855317288648360902045172080920732975782187569890436896528061950673606581879, 30703199998691737762375684559082041721750342569643355527421523225340585194484725217051447417832692027394342279884463261474883878107998410707989777059058221)
Private Key B: (48923244481209295915689258446297430030089498035528449614074271230986320516823487937606180878556380522853276912807768937806588142374073547180433360862282 61, 10560691643365854647321708318162803069728076746157918284125206172802388786063 1, 88969839384476022984984629332032905301708684335809771062077249588984000981677)
PS C:\Users\User>
```

4. Шифрування і розшифрування повідомлення реалізовано відповідно у функціях encrypt(message, public key) і decrypt(ciphertext, private key). Для зашифрування повідомлення використовується відкритий ключ e:

$$C = M^e \bmod n \ ,$$

Для розшифрування використовується секретний ключ d:

$$M = C^d \bmod n \ .$$

У функції шифрування, якщо повідомлення M ≥ n, то виводиться помилка «Message is too large for the given public key».

```
Public Key A: (59105642778899683538187118688550561429043036821543985842236711224220799497836593218896918128044963135515879141017737527049231207550624741989120843945412 11, 18546067676087151509570330737915272188751414898700619592776612809502494919356755600091154064646035997044324627331356333387857480278423074127232059064080 9)
Private Key A: (17457263836144380241568044444574510689217901341411349225491881493967913872457741567017168432379792956936321788445559648094672540755394726634321835859066 81, 93352986957072568358093947075466243266182548916719040354246889951105460633003, 63314142059620243956939361793680780647347446789908717914373233881795370006737)
Public Key B: (62455574603490576769140319512410888034280346256905860486836024477448126462776423117584985540497087420691875421133046345703996222077064283041358238663454 31, 32824309930709416376488987307575313298873182685368100201774896605920575980053002722353201567022664962330729430804544520380785086982519397166138766727413 77)
Private Key B: (53004570470231781611954151767175525367239062762530414377482684698026142320770056232673994656773026182540190353697306498388702940906027317581608870116404 25, 66305316323948850853330658639777342131434969234724733518804200458049698195867, 94193916967910182613193249607917499530282067928291086213525532890385934949493)
Message to encrypt:  123123123123
Ciphertext A: 24682799755555961824150785540310189992425458275420512145215025831426497877165992586278890945428965909759189652593325679728858111482687388287725973015190790
d: 17457263836144380241568044444574510689217901341411349225491881493967913872457741567017168432379792956936321788445559648094672540755394726634321835859066 81
Decrypted Message A: 123123123123
```

```
Public Key B: (10508101063015903600322318656474678261586955809516338160128576302766640769967176385062531301267847233034390721002792546772750248309735800149201066469972753, 55657774437204071214590465364733411835295359635206769900631102467058348267511188245648876791199794153211750999471852262309235342907658840076832105073341 27)
Private Key B: (35943718695058561706693561751082055268251260638417444099070605773689325373622039408266449932587674101696484125911425459098775105159618743072546263102063, 96595699550283713669793348210741830334903360804989993304309479730113898247301, 10878435698419288471508588648321826109416487099721628699509616741347783252585 3)
Message to encrypt:  319939997168755089470608595740313696391741123559425939597505325980400699869157665138623234847172966674989288625973117725375332676006068244582842493805485094
Traceback (most recent call last):
  File "c:\Users\User\Desktop\university\3 year\crypto labs done\lab4\lab4.py", line 132, in <module>
    ciphertext_A = encrypt(message_to_encrypt, public_key_A)
  File "c:\Users\User\Desktop\university\3 year\crypto labs done\lab4\lab4.py", line 118, in encrypt
    raise ValueError("Message is too large for the given public key")
ValueError: Message is too large for the given public key
```

Також треба реалізувати функцію електронного підпису – в нас це функція `sign(m, private_key)`, яка повертає підпис S:

Цифровий підпис у системі RSA. Щоб підписати відкрите повідомлення M, абонент A зашифровує його на своєму секретному ключі $S = M^d \bmod n$ і додає S до повідомлення M. Підписане повідомлення (M, S) може передаватися як у відкритому, так і в зашифрованому виді. Перевіряється підпис за допомогою відкритого ключа: якщо виконується рівність $M = S^e \bmod n$, то повідомлення M не спотворене і підпис вірний. На

Перевіряється підпис за допомогою функції `verify(s, m, public_key)`: якщо підпис вірний, функція повертає True:

```
print("verification result:",verify(sign(message_to_encrypt,private_key_A),message_to_encrypt,public_key_A))
```

```
Decrypted Message A: 123123123123
verification result: True
```

Також перевірили цю функцію на ключах абонента B:

```
138   print("Decrypted Message A:", decrypted_message_A)
139   sign_val=sign(message_to_encrypt,private_key_A)
140   print("verification result A:",verify(sign_val,message_to_encrypt,public_key_A))
141   sign_val_b=sign(message_to_encrypt,private_key_B)
142   print("verification result B:",verify(sign_val_b,message_to_encrypt,public_key_B))
```

```
verification result A: True
verification result B: True
```

5. Протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника реалізований у функціях `send_key()`, яка приймає як аргументи відкриті ключі A і B і секретний ключ A, а також `recv_key()`, яка приймає значення k1, S1 (які були створені у попередній функції), відкриті ключі A і B і секретний ключ B. Створення протоколу здійснюється за алгоритмом, наданому у теоретичних відомостях:

Нехай абонент A має відкритий ключ (e, n), секретний d і відкритий ключ (e_1, n_1) абонента B, якому він має намір передати секретне значення $0 < k < n$ для створення спільного ключа симетричної чи асиметричної криптосистеми. Обов'язковою умовою роботи протоколу є $n_1 \geq n$; якщо $n_1 < n$, абонент А повинен згенерувати собі іншу пару ключів.

Абонент A формує повідомлення (k_1, S_1) і відправляє його B, де

$$k_1 = k^{e_1} \bmod n_1, \quad S_1 = S^{e_1} \bmod n_1, \quad S = k^d \bmod n.$$

Абонент *B* за допомогою свого секретного ключа d_1 знаходить (конфіденційність):

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1,$$

і за допомогою відкритого ключа *e* абонента *A* перевіряє підпис *A* (автентифікація):

$$k = S^e \bmod n.$$

Перевірили це на практиці (абонент A – відправник, абонент B – одержувач):

```
155
156   k_1,S_1=send_key(public_key_B,private_key_A,public_key_A)
157
158   print("perevirka pidpeesu:",recv_key(k_1,S_1,public_key_B,private_key_B,public_key_A))
```

```
Verification result B: True
k value: 27467293522210213584098391681972183167277275483007831509814391473783766378414695112769481204852154478156111697835653342846709785593287967432158685328247 74
perevirka pidpeesu: True
PS C:\Users\Igorm>
```

```
Verification result B: True
k value: 4563081741836616156406512118577723769615020063403028876029295774135396695602801989547548228965850388542969106248170362637847597127270386257285432889266066
perevirka pidpeesu: True
PS C:\Users\Igorm>
```

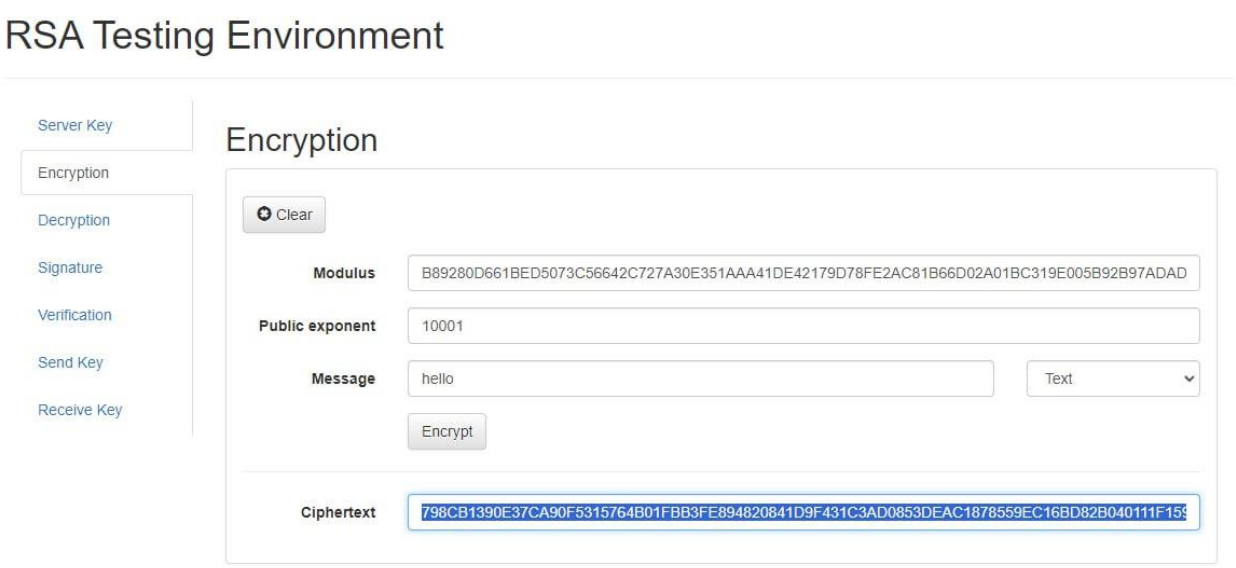
Висновки: у ході лабораторної роботи було розглянуто методи генерації параметрів для асиметричних криптосистем, було побудовано криптосистему RSA та алгоритм електронного підпису. Ми навчилися генерувати RSA ключі і використовувати їх для шифрування і розшифрування тексту, а також розробили протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника. RSA є основним компонентом цифрових сертифікатів, які використовуються для автентифікації особи, веб-сайтів і організацій в Інтернеті, тому важливо розуміти, як цей алгоритм працює на практиці.

UPD: Додамо ще перевірку наших функцій на сервері.

Візьмемо ключі з результатів генерації ключів:

```
Public Key A: (96668346016459206364035186175313317390567226835872305618888042305040693948174702958005116909562045894678126271363
Private Key A: (74559280321650059711144095561129401215472056732832953971398641415310389515727533216541185823468571322849311398813984774208961216538581446749048347038265884241719, 93184623842486028385840877052847584655817366271980728559916062146831293390457
```

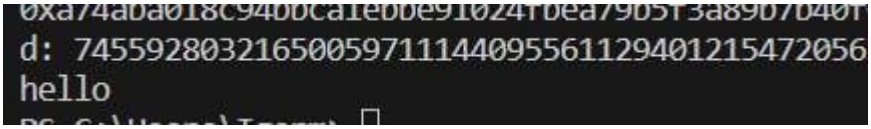
Закинемо в Encrypt на сервері наш модуль n з Public Key A, але в hex значенні (сайт приймає тільки такі) і зашифруємо, наприклад, повідомлення ‘hello’, отримуємо шифртекст:



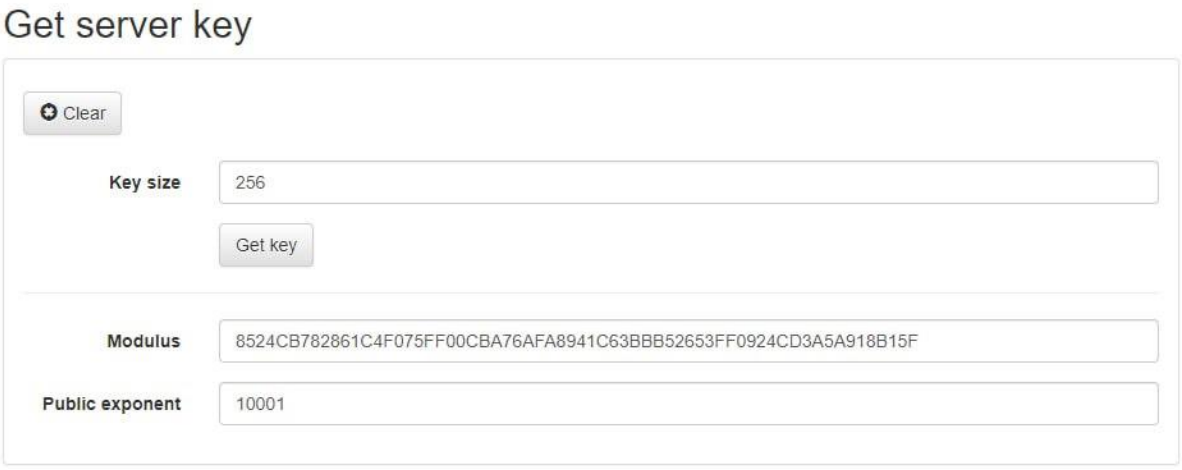
Спробуємо розшифрувати цей шифртекст за допомогою приватного ключа A (адже для шифрування ми використовували відкритий шифр A), для цього нам також знадобилося написати функцію encode і decode, тому що сайт шифрує і розшифровує тільки або текст, або байти, а не числа, як ми робили:

```
print(decode(decrypt(int(0x798CB1390E37CA90F5315764B01FBB3FE894820841D9F431C3AD0853DEAC1878559EC16BD82B040111F1595FB687D0C63580863E5F81DEC5810019111822C480), (745592803
```

Отримуємо повідомлення ‘hello’, тобто наша функція розшифрування працює:



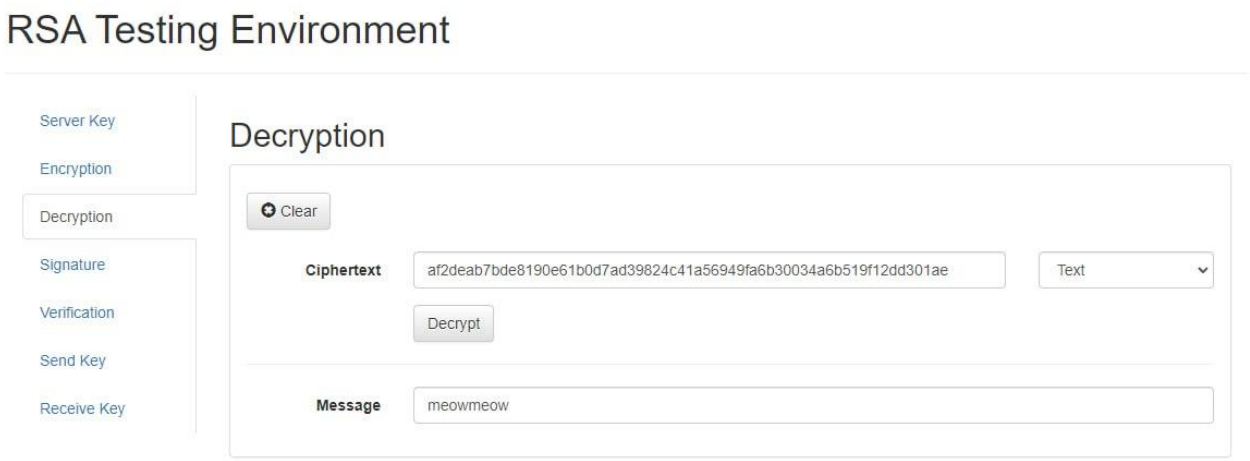
Тепер перевіримо функцію шифрування. Зашифруємо у нашому коді повідомлення ‘meowmeow’ за допомогою відкритого ключа сервера:



```
print(hex(encrypt(encode('meowmeow'), (int(0x8524CB782861C4F075FF00CBA76AFA8941C63BBB52653FF0924CD3A5A918B15F), 65537))))
```

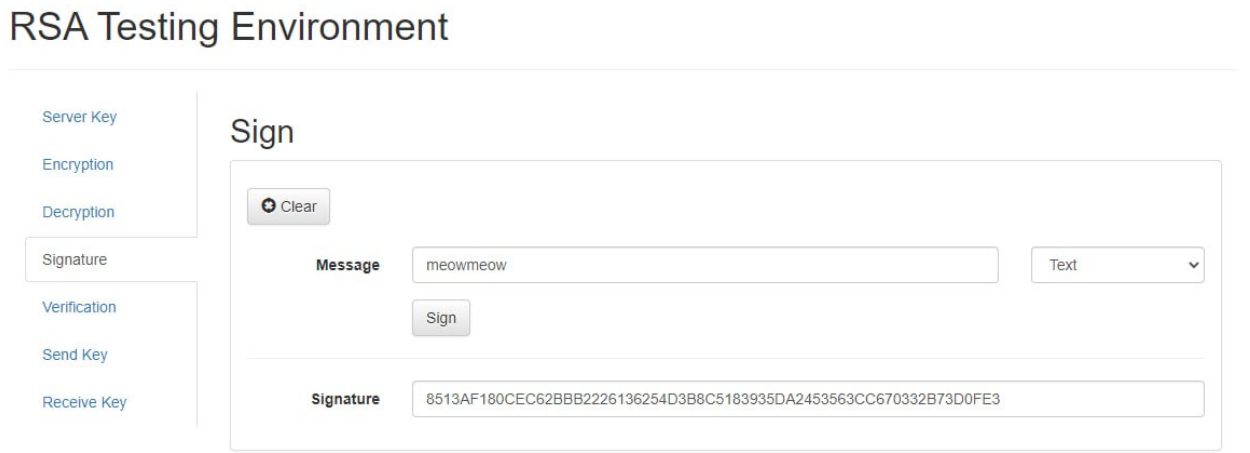
```
hello
0xaf2deab7bde8190e61b0d7ad39824c41a56949fa6b30034a6b519f12dd301ae
PS C:\Users\Igorm>
```

Отримали шифртекст, і тепер можемо розшифрувати цей шифртекст на сайті функцією Decrypt (тільки сайт знає свій приватний ключ):



Як бачимо, вийшло повідомлення ‘meowmeow’, функція шифрування працює правильно.

Перевіримо ще цифровий підпис. Підпишемо на сайті повідомлення ‘meowmeow’ (сайт робить це своїм приватним ключем):



Отримали сигнатуру, кидаємо її в нашу функцію `verify()`, використовуючи відкритий ключ сервера:

```
print(verify(int(0x8513AF180CEC62BBB2226136254D3B8C5183935DA2453563CC670332B73D0FE3), encode('meowmeow'), (int(0x8524CB782861C4F075FF00CBA76AFA8941C63BBB5265
```

Бачимо, що функція підтвердження електронного підпису працює правильно, повертає true:

```
0xaf2deab7bde8190e61b0d7ad39824c41a56949fa6b30034a6b519f12dd3
True
```

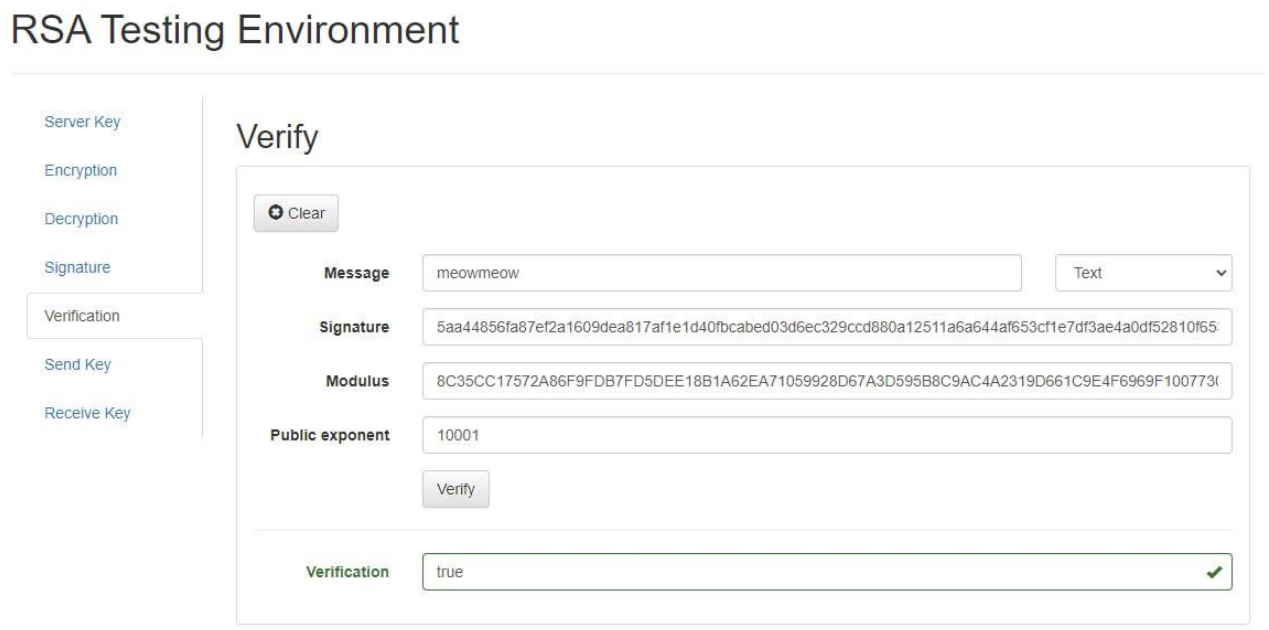
Далі підписуємо повідомлення ‘meowmeow’ за допомогою приватного ключа A:

```
Public Key A: (734340117113545459980601768039409140802128727672497739268370336644582399034738473863422573089392278071872434
Private Key A: (705016710694482206112264266674391918141964684760571246086422350453593001621445262228536631811870053412432694
873668702112362654019953432551622087544014493883, 969738446472617281819534067777315912474982938387713574920688221657317444
```

```
print(hex(sign(encode('meowmeow'), private_key_A)))
```

```
True
0x5aa44856fa87ef2a1609dea817af1e1d40fbcabed03d6ec329ccd880a12511a6a644af653cf1e7df3ae4a0df52810f6533b852d74b2a77ba5380925256d0f6cb
```

На сайті вводимо сигнатуру, яка вийшла в нас, повідомлення і відкритий ключ (n, e) абонента A:



Бачимо, що підпис підтверджений, тому функція `sign()` в нас працює правильно.