КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3

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

ФБ-12 Юрченко Вікторія

Варіант 10

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 p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – 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) , (e_1,n_1) та секретні d і d_1.

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

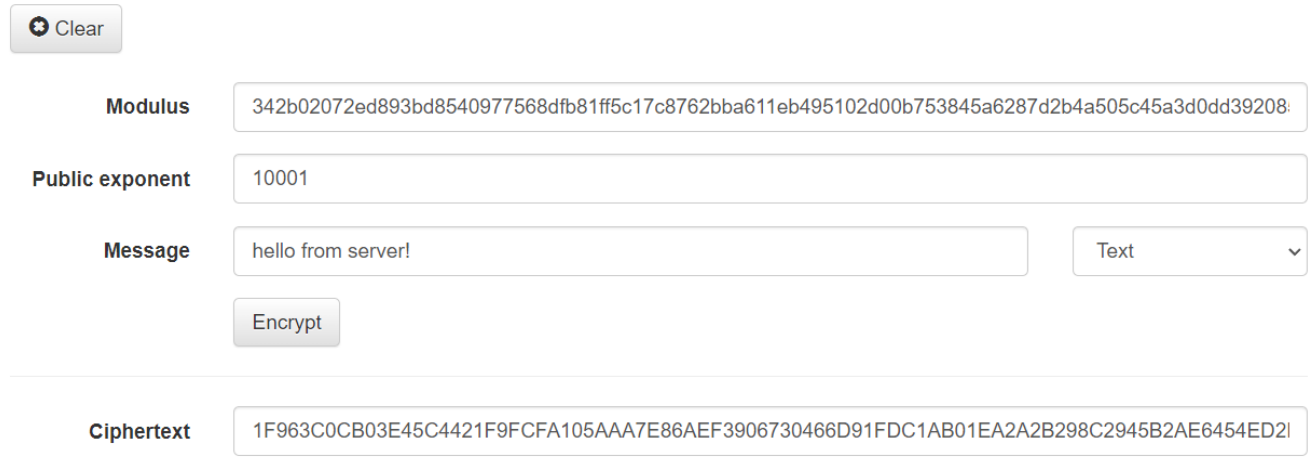
Функція generate_prime генерує просте число для заданої довжини у бітах. Для перевірки числа на простоту використовується тест Міллера-Рабіна з попереднім пробним ділення на перші чотири прості числа. Функція generate_prime_pairs генерує дві пари простих чисел p, q і p_1, q_1 такі що $pq \leq p_1q_1$, $p \neq q$, $p_1 \neq q_1$; p і q – прості числа для побудови ключів абонента A, p_1 і q_1 – абонента B. Функція generate_key_pair генерує ключові пари для RSA та повертає відкритий (n, e) і секретний (p, q, d) ключі. Шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису реалізовані у функціях encrypt, decrypt, sign, verify відповідно. Вони використовують схему Горнера для швидкого піднесення до степеня по модулю (mod_exp). Конфіденційне розсилання ключів реалізовано у функціях send_key і receive_key. Також додано функції encoding, decoding для роботи з рядками.

Тестування (файл test.py):

```
Keys for A:
Public key
n: 65378409573449426812874629324179994330037572135065120251723385553279142656285003365019498302957574226038543511111447591704625857330415552648477428979350 1
e: 65537
Private key
p: 802741583127293490055019094356210313857596474386997021179508331530426921977 07
q: 814439054206590763445825472816251614542713386532392485702418842759964114034 3
d: 157557807162833246423019347169705483993562936097352413027102118105572543618601049009118008539996456084772338752082555939544270545528832215305184248901497

Keys for B:
Public key
n1: 34954498402027531677016592063206077802151482029074256643959213511464720783678420027652428661646364872249839918362268038632231417385012729680903727240768 61
e1: 65537
Private key
p1: 41140650786689098018315699599089430583175534519721412745520991244214026140103
q1: 84963406590877088042146722366900304497621840990810304612783595859441805319387
d1: 13504553146151595313518472176638813036154775546274015872332381496105813971385715939893975829753810053559426860124970151863662249681148047577182915257193 93

Test for A:
rand message: 650438475466834324789776649910316169204873369795616935957586768959328943496738243957239622431843465684823944829758001913090521059247130565602416390145759
cipher message: 402037380794724327785200224516630417957143164776170100692988606071833947660315932700753694513231369815922015471059880878965305823592433174010553856933460
decrypted: 650438475466834324789776649910316169204873369795616935957586768959328943496738243957239622431843465684823944829758001913090521059247130565602416390145759
Encryption and decryption for A passed
signature: 594178504801679109165868989970826911416110111846387462657297817986869017146976081224110585521002284668873031579496318403543354666438381971871269745217916
verification: True
Signature verification for A passed
```

```
Test for B:
rand message: 155931674686302708580270764384982927842531153778286576352070472753184113669847410824133912893597002124318809994914186346294575928426870477837076499122716
cipher message: 760672471731432261847657666901853064017370875230527707958577705490632818143345618674104758577003479135871583579998970777545138975818666650739544099458555
decrypted: 155931674686302708580270764384982927842531153778286576352070472753184113669847410824133912893597002124318809994914186346294575928426870477837076499122716
Encryption and decryption for B passed
signature: 13729982243670452077283838806946767836224418187240709708781506379892761794501022996807592754607601931529956786765256208726718797621057999728024874624380
verification: True
Signature verification for B passed

Key exchange:
message: 57572988650587490461184855484525035767979467940297256846991168590282791633449142889494842555442169180264741011556788158105585474425062035169419434187581
k1: 18652961917472814439785071538104372647297550959436448761694667310641894854296194423124022132372115545561683118382723859042089379578591658075983448923084
S1: 15110918175587810321532987600693673380951181750593195281833490440089078497372943269693656128385589948557035443559378533707169441087920762219622902109594080
k: 57572988650587490461184855484525035767979467940297256846991168590282791633449142889494842555442169180264741011556788158105585474425062035169419434187581
S: 47321982706902838934999037092379294476204702505556616443203415996519648338591326668772743846413003458621930988094790803127565347481021836719718456467424
Key exchange test passed

Key exchange:
message: 448378203247
k1: 33581457969698274786479167071912390926778281515392641622387814629342557651153585749328194453922000518927018300937686830726501622341647508006707406274911400
S1: 30418495229306427738296102951472363877478675900369487752382079068113676667009575591082072908167650072655188496940285328402956724610390589391269076061426600
k: hello
S: 20094479146854239486926012003572422502759103631987901685140052698181542981603656963031327472672550676305288950644924365964805704563061923568529216903997 5
Key exchange test passed
PS E:\CryptoLabsGit\crypto-23-24\cp4\yurchenko_fb-12_cp4>
```

Ділюся з сервісом своїм відкритим ключем для отримання тексту для розшифрування:



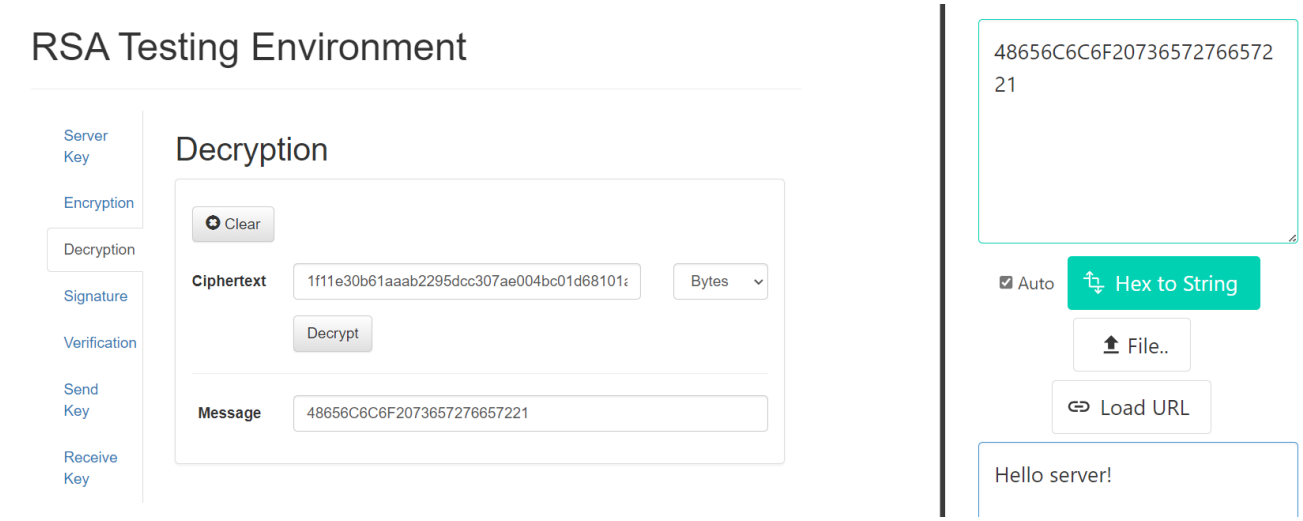
Розшифровую за допомогою свого секретного ключа:

```
>>> from lab4 import *
>>> p,q,_,_ = generate_prime_pairs()
>>> my_public_key, my_private_key = generate_key_pair(p,q)
>>> print(f"My public key:\nn: {hex(my_public_key[0])[2:]}\ne: {hex(my_public_key[1])[2:]}")
My public key:
n: 342b02072ed893bd8540977568dfb81ff5c17c8762bba611eb495102d00b753845a6287d2b4a505c45a3d0dd3920850e26969081559652b7a2a5
f37c2744b9cb
e: 10001
>>> decoding(decrypt(0x1F963C0CB03E45C4421F9FCFA105AAA7E86AEF3906730466D91FDC1AB01EA2A2B298C2945B2AE6454ED2DBA72055D905
A893DE6743AC8406CDD73FFFD825AC7C, my_private_key))
'hello from server!'
>>>
```

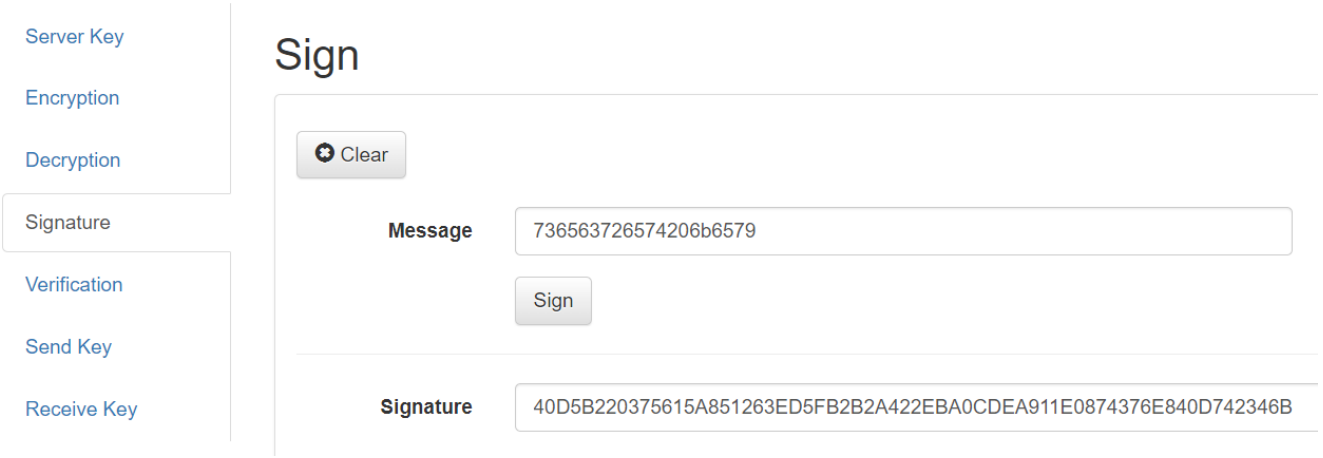
Отримую відкритий ключ серверу та шифрую повідомлення для нього:

```
>>> server_public_key = (0x839F5CF04D30F0145826F8175B6045544CDDAE87F076E28362B54246E28581A5, 0x10001)
>>> message_for_encr = encoding("Hello server!")
>>> encrypt(message_for_encr, server_public_key)
14053301709448126608850902637970353429157923880321674521095277226848090558 34
>>> hex(encrypt(message_for_encr, server_public_key))[2:]
'1f11e30b61aaab2295dcc307ae004bc01d68101adadd2ef930b1caa1bf098e5a'
>>>
```

Розшифровую на сервері:



Отримую підпис для повідомлення:



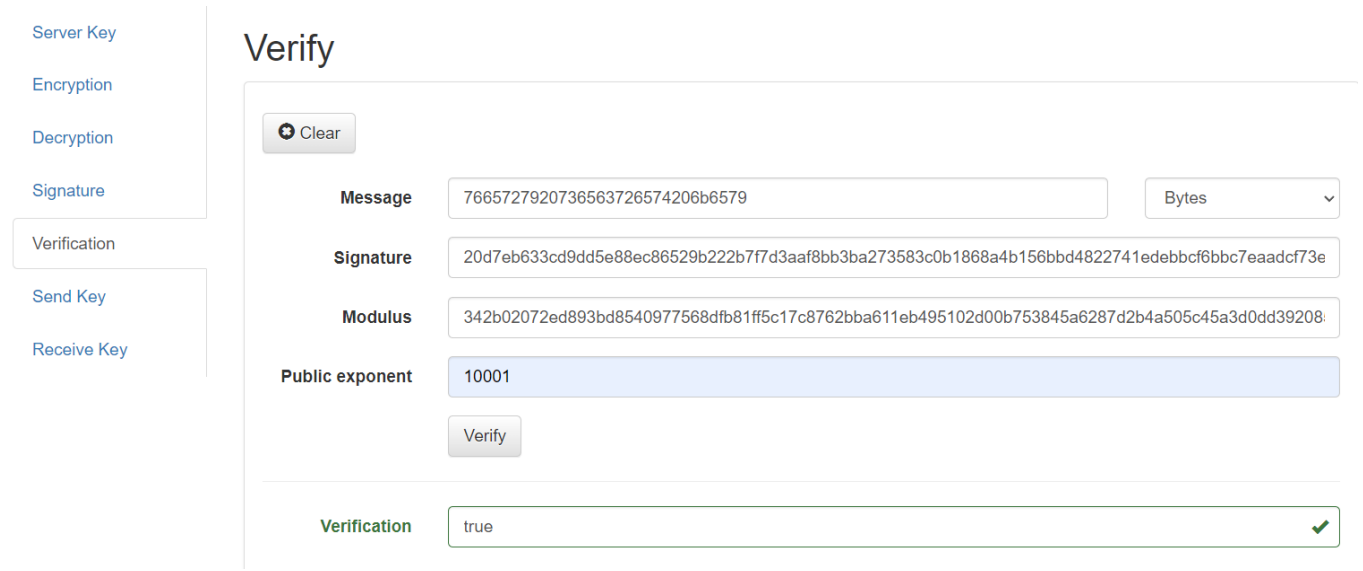
Перевіряю його:

```
>>> message_for_sign = encoding("secret key")
>>> print(hex(message_for_sign)[2:])
736563726574206b6579
>>> verify(0x40D5B220375615A851263ED5FB2B2A422EBA0CDEA911E0874376E840D742346B, message_for_sign, server_public_key)
True
>>>
```

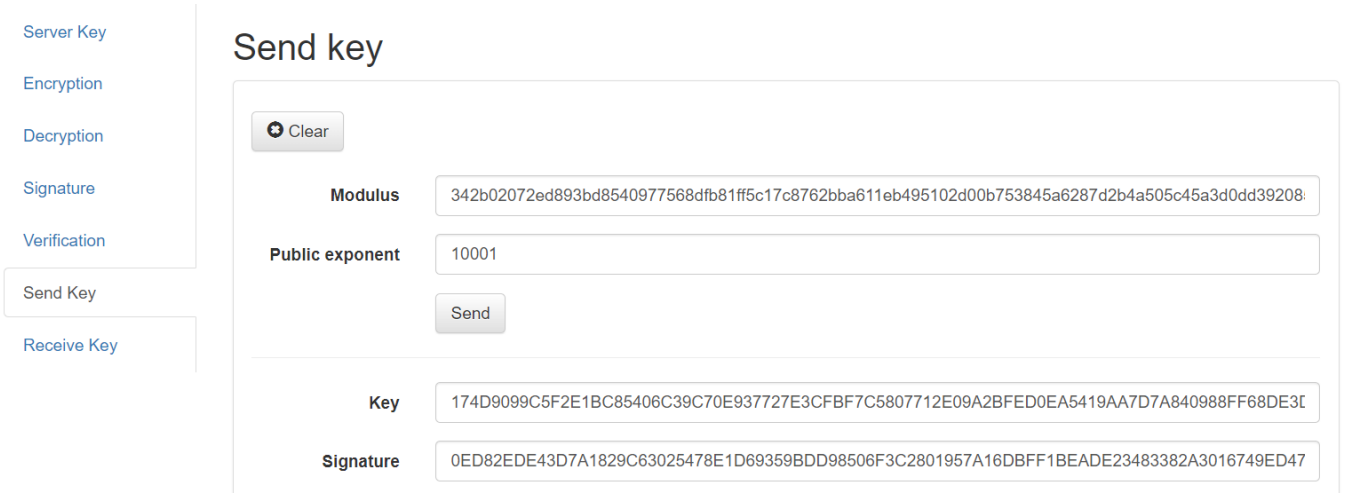
Підписую повідомлення за допомогою свого секретного ключа:

```
>>> message_for_ver = encoding("very secret key")
>>> hex(message_for_ver)[2:]
'766572792073656372657420657579'
>>> sign(message_for_ver, my_private_key)
172015029172436014229238879381662488467695785580116811591520621673224751761953042526475331297664928482989243884586958347444096689308943358041196708935262 8
>>> hex(sign(message_for_ver, my_private_key))[2:]
'20d7eb633cd9dd5e88ec86529b222b7f7d3aaf8bb3ba273583c0b1868a4b156bbd4822741edebbcf6bbc7eaadcf73ebb6610f3d95fd31c72a7cde51474831fb4'
>>>
```

Перевіряю підпис на сервері:



Відправляю собі зашифровані повідомлення та підпис з серверу:



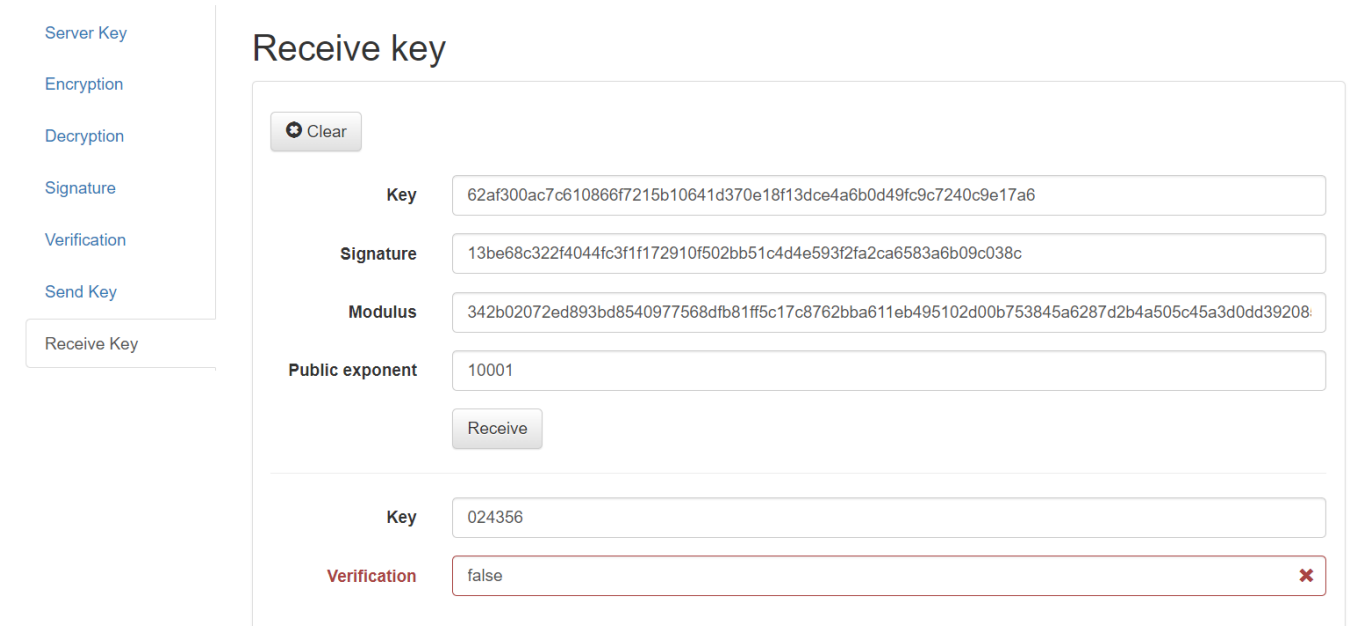
Отримую розшифровані повідомлення та підпис:

```
>>> receive_key(server_public_key, my_private_key, 0x174D9099C5F2E1BC85406C39C70E937727E3CFBF7C5807712E09A2BFED0EA5419A
A7D7A840988FF68DE3D38CEEE5AA1FC75F17349C2184540E8F01F4F6A466A4, 0x0ED82EDE43D7A1829C63025478E1D69359BDD98506F3C2801957A
16DBFF1BEADE23483382A3016749ED47CFA272F888A02C01B778192DB354D03B9D9503A2196)
(16788081948980847801, 49853606877418473379347818473472072909682526889778950182916879360118944247650)
>>>
```

Тут вже намагаюся відправити зашифровані повідомлення та підпис серверу:

```
>>> k1, S1 = send_key(my_private_key, server_public_key, 0x24356)
>>> hex(k1)[2:]
'62af300ac7c610866f7215b10641d370e18f13dce4a6b0d49fc9c7240c9e17a6'
>>> hex(S1)[2:]
'13be68c322f4044fc3f1f172910f502bb51c4d4e593f2fa2ca6583a6b09c038c'
```

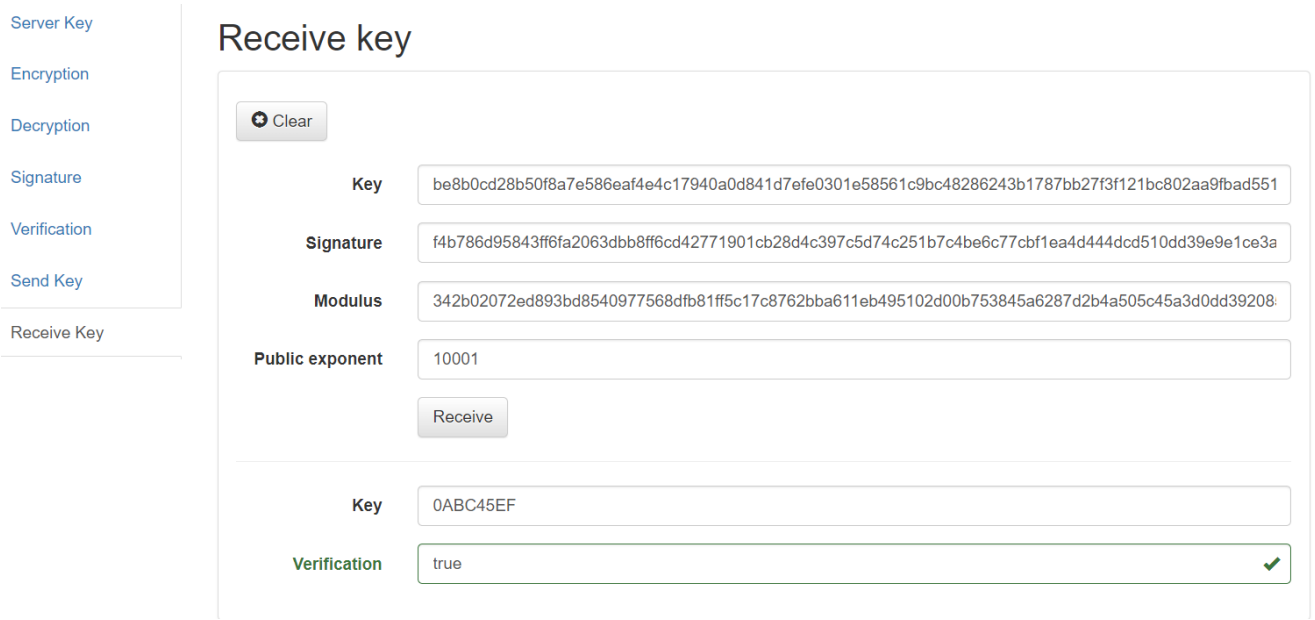
Але на першій сторінці я вказала довжину ключа для серверу 256, а у мене таку довжину має кожне просте число, тому і довжина модулю набагато більша за 256. А як відомо, обов'язковою умовою роботи протоколу є $n_1 \geq n$, тому і перевірка не була пройдена, хоча повідомлення було розшифровано правильно:



Отримую відкритий ключ сервера правильної довжини та відправляю зашифровані повідомлення та підпис серверу:

```
>>> server_public_key = (0x27D7BB3EDE9CC8790483B83E382CB6A501219011B73FF94424F44611811E037FF48CAE901A2ACE23DB0CAE4F4AAB
272231B9A7BB5A403182F236DAB2A143EBA9, 0x10001)
>>> k1, S1 = send_key(my_private_key, server_public_key, 0xabc45ef)
>>> hex(k1)[2:]
'be8b0cd28b50f8a7e586eaf4e4c17940a0d841d7efe0301e58561c9bc48286243b1787bb27f3f121bc802aa9fbad551cc98ab45830fedc5bcb1126
c8c9f5f2b'
>>> hex(S1)[2:]
'f4b786d95843ff6fa2063dbb8ff6cd42771901cb28d4c397c5d74c251b7c4be6c77cbf1ea4d444dcd510dd39e9e1ce3a0ef324e9f14cd5548f2c40
2ecd0af87'
>>>
```

Повідомлення отримано та перевірка пройдена:



Висновок:

Під час виконання комп’ютерного практикуму, я навчилась будувати криптосистему RSA, використовувати її для генерації ключів, шифрування, розшифрування,

підпису повідомлень, перевірки підпису та конфіденційного розсилання ключів з підтвердженням справжності. Також впевнилася, що при розсиланні ключів по відкритому каналу обов’язково потрібно обирати такі n_1 та n що $n_1 \geq n$.