МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Фізико-технічний інститут

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Варіант №11

Виконали:

ФБ-32 Пінькас Б. О.

ФБ-32 Драчук О. І.

Київ 2025

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 $pq \leq p_1 q_1$; *p* і *q* – прості числа для побудови ключів абонента *A*, p_1 і q_1 – абонента В.

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 $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою

http://asymcryptwebservice.appspot.com/?section=rsa.

Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Хід роботи

Функціонал криптосистеми RSA реалізовано в файлі *RSA.py*, реалізовано взаємодію абонентів A, B через криптосистему RSA у скрипті *lab4.py*, функції арифметики з модулем винесено в окремий файл *math_mod.py*, кожну операцію перевірено із взаємодією з тестовим середовищем в скрипті *test_site.py*, для генерації простих чисел для ключів, що використовувались в *test_site.py*, написано скрипт *gen_primes_128_256bit.py*

Спочатку, ми згенерували пари простих чисел розміру 256 біт для абонентів A і B. Генератором псевдовипадкових чисел ми будемо генерувати число розміром 256 біт, і додавати до цього числа 2, доки не натрапимо на просте число. Таким чином, ми маємо знайти 4 простих числа.

Для перевірки на простоту ми використовували тест пробних ділень, перед цим згенерувавши прості числа в діапазоні від 2 до 1000. Якщо число пройшло перевірку, тобто не поділилось ні на одне просте в нашому діапазоні, далі воно проходить 15 ітерацій тесту Міллера-Рабіна. Така кількість операцій дозволяє отримати похибку $e = 4^{-15} \approx 10^{-9}$, що надає нам можливість визначити чи є число простим з дуже високою точністю, достатньою в межах лабораторної роботи.

```
Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418295
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418297
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418299
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418301
Число не є псевдопростим за основою 52080445672404857254139589764539822766224323198678908826587275789866765191277

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418303
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418305
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418307
Число не є псевдопростим за основою 11128238920620391890086915759908788029065585693543315590662344855367187220488
```

```
Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418345
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418347
Число не пройшло тест пробних ділень

Згенеровано число: 85118714922376686876690589359602778083694121122248847194791182892292958418349

Число 85118714922376686876690589359602778083694121122248847194791182892292958418349 прошло тест Міллера-Рабіна з 15 ітераціями
Похибка: 9.3e-10

 ----------------------------------------------------------------------------------------------------
```

І так 4 простих числа

```
Згенеровано число: 109489276012612362728828094736276370790343389609574255556605693378273622475021
Число не пройшло тест пробних ділень

Згенеровано число: 109489276012612362728828094736276370790343389609574255556605693378273622475023

Число 109489276012612362728828094736276370790343389609574255556605693378273622475023 прошло тест Міллера-Рабіна з 15 ітераціями
Похибка: 9.3e-10

 ----------------------------------------------------------------------------------------------------

p_A: 85118714922376686876690589359602778083694121122248847194791182892292958418349
q_A: 958252814360426474525834469185259626301334603609625656407491500600012716508367

p_B: 952951018548287510253156121063595232231157137637774086905478893582110802307 07
q_B: 109489276012612362728828094736276370790343389609574255556605693378273622475023
```

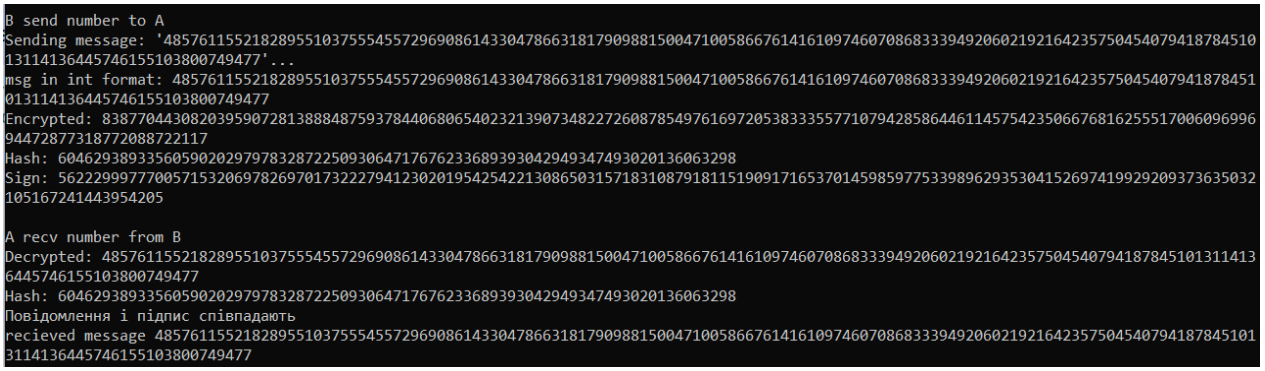
Далі, за допомогою визначених двох простих, ми будуємо пару ключів pk і sk
pk (public key) містить 2 значення: n, e
sk (secret key) містить 3 значення: d, p, q

Ці значення визначаються з наступних формул:
$n=pq$
$\varphi(n) = (p-1)(q-1)$. Обирається випадкове число e,
$2 \leq e \leq \varphi(n) - 1$ таке, що $gcd(e, \varphi(n)) = 1$
d: $ed \equiv 1(mod\varphi(n)) \Rightarrow d \equiv e^{-1} mod(\varphi(n))$

```
-----------------------------------------------------------------------------------------------

n_A: 81565248129110290103227395490864864612334169221955835583888356396692071562233368773615145784667817768312028511093533459795324028211113291164821181448260
83
e_A: 65537
d_A: 52711185345868869749027112642089195583047577232059683911140631375697686907780234561517075668360889398488403003872467388301658208907137110693018927863290
65

n_B: 10433791709633353444619783219515525911491871682866041099345261661582594794411025504515481178130157531115825650331878602674514480772160420958634475185131261
e_B: 65537
d_B: 59784510471970289844265391986091372664043634196711105995485491541533728995790948576011668378540363459417143012065049036633274589966028200329542294977805
45

-----------------------------------------------------------------------------------------------
```

Далі, з використанням створених ключів, ми шифруємо і розшифровуємо повідомлення з підтвердженням цифрового підпису, підписуємо не відкрите повідомлення, а хеш m = H(M):

```
-----------------------------------------------------------------------------------------------

A send number to B
Sending message: '485761155218289551037555455729690861433047866318179098815004710058667614161097460708683339492060219216423575045407941878451013
13114136445746155103800749477'...
msg in int format: 485761155218289551037555455729690861433047866318179098815004710058667614161097460708683339492060219216423575045407941878451013114136445746155103800749477
Encrypted: 838770443082039590728138884875937844068065402321390734822726087854976169720538333557710794285864461145754235066768162555170060969969447287731877208872211
7
Hash: 60462938933560590202979783287225093064717676233689393042949347493020136063298
Sign: 562229997770057153206978269701732227941230201954254221308650315718310879181151909171653701459859775339896293530415269741992920937363503210516724144395420
5

A send text to B
Sending message: 'I'm A'...
msg in int format: 314194075713
Encrypted: 780787426193179152907431636300313237335024455627734475308550164658006954128810351527888711308509288143099806677981262533947764449273339316215952947544013
8
Hash: 76807678959117770766090225879073126004192828600045539836684313845932811375757
Sign: 556686928780740759070768054444711732525232598397709501132391187406728467645868982661497153568221987086156527175865971740162525039421326282346334151914600
1

B recv number from A
Decrypted: 485761155218289551037555455729690861433047866318179098815004710058667614161097460708683339492060219216423575045407941878451013114136445746155103800749477
Hash: 60462938933560590202979783287225093064717676233689393042949347493020136063298
Повідомлення і підпис співпадають
recieved message 485761155218289551037555455729690861433047866318179098815004710058667614161097460708683339492060219216423575045407941878451013114136445746155103800749477

nB recv text from A
Decrypted: I'm A
Hash: 76807678959117770766090225879073126004192828600045539836684313845932811375757
Повідомлення і підпис співпадають
recieved message I'm A
```



Для хешування *H(M)* використано хеш-функцію sha-256.

Протокол обміну повідомленнями send/recv реалізовано наступним чином:

1. Абонент А шифрує ВТ публічним ключем абонента В.
2. Абонент А знаходить хеш ВТ і підписує хеш ВТ своїм секретним ключем
3. Абонент А пакетом надсилає ШТ, цифровий підпис, тип повідомлення (щоб абонент В знав як інтерпретувати отримане повідомлення)
4. Абонент В розпаковує отриманий пакет
5. Абонент В розшифровує ШТ своїм секретним ключем
6. Абонент В знаходить хеш розшифрованого повідомлення
7. Абонент В перевіряє цифровий підпис порівнюючи підписаний хеш з хешем розшифрованого повідомлення

Реалізовані процедури:

Зашифрування:

$C = M^e mod\ n$, де C - криптограма, M - відкрите повідомлення

Розшифрування:

$M = C^d mod\ n$.

Цифорвий підпис:

$S = m^d mod\ n$, де $m = H(M)$

Перевірка цифрового підпису:

Для підтвердження цифрового підпису має виконуватись рівність:

$m = S^e mod\ n$, де $m = H(M)$

Потім, з використанням реалізованих функцій, ми організовуємо **протокол конфіденційного розсилання ключів з підтвердженням справжності**

Протокол реалізується наступним чином:

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

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

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

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

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

$$k = S^e mod\, n.$$

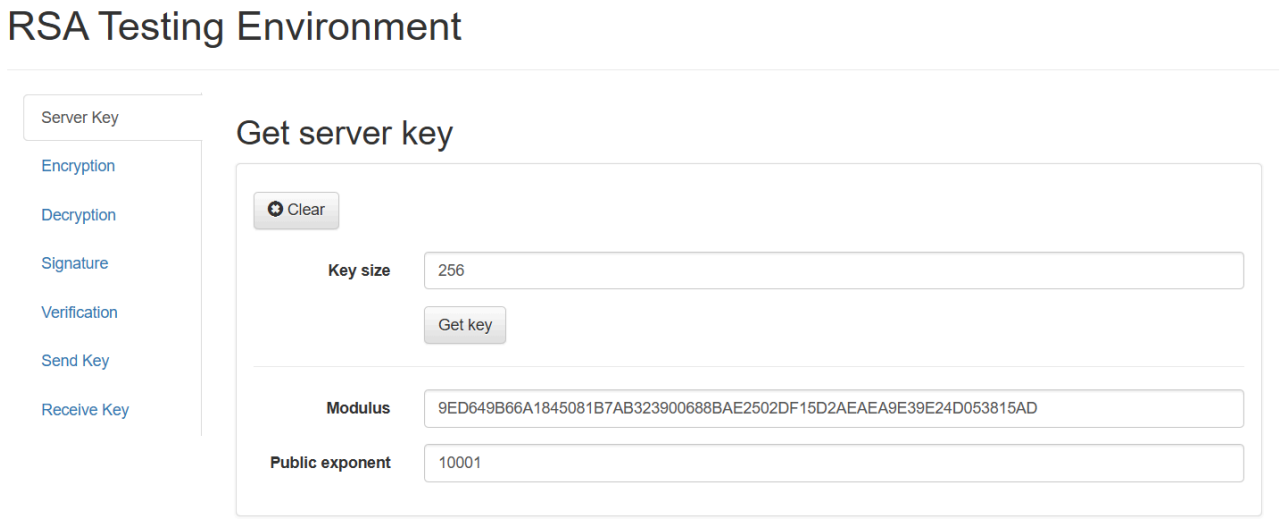
```
A send key to B
Created key: 693558604587889260231114374398538457138133358030534948616442871911630183100657085938873212261095337397852012860378058796337536644
603975478757535687357134 6
msg in int format: 693558604587889260231114374398538457138133358030534948616442871911630183100657085938873212261095337397852012860378058796337
536644603975478757535687357134 6
Encrypted: 310275108869631030698142599206775028142818785100627087509086878455597931007551522878681822829502720819980177671626185938614627815950535185748957916853471 3
Hash: 113907040497693736169165512823090777261667053420650808869693321690786804114193
Sign: 143194415303530800276913781198517412689623355570688991779962249209954087552316829094893249383059415268531441285429920679011345672158796391780452799609213 8
msg in int format: 143194415303530800276913781198517412689623355570688991779962249209954087552316829094893249383059415268531441285429920679011
345672158796391780452799609213 8
Encrypted: 747822732270206873711273737171422961220188846879791813862606225963940064217324389681606316843996737414278038370355774541621526539289592448310907122271263 2
sender saved key: 693558604587889260231114374398538457138133358030534948616442871911630183100657085938873212261095337397852012860378058796337536644603975478757535687357134 6

Decrypted: 693558604587889260231114374398538457138133358030534948616442871911630183100657085938873212261095337397852012860378058796337536644603975478757535687357134 6
Decrypted: 143194415303530800276913781198517412689623355570688991779962249209954087552316829094893249383059415268531441285429920679011345672158796391780452799609213 8
Hash: 113907040497693736169165512823090777261667053420650808869693321690786804114193
Повідомлення і підпис співпадають
reciever recieved key 693558604587889260231114374398538457138133358030534948616442871911630183100657085938873212261095337397852012860378058796337536644603975478757535687357134 6
```

Важливий н’юанс, має виконуватись співвідношення n_A ≤ n_B, інакше абонент А перегенеровує пару ключів

```
B send key to A
Перегенерація пари ключів відправника для коректної роботи протоколу конфіденційного розсилання ключів

Згенеровано число: 9446163539143195843144498020021320218884778755830404817956973218636026503279 9
Число не є псевдопростим за основою 8430736589004474763266464228186590389162452201693592644171111228318029992760 6

Згенеровано число: 9446163539143195843144498020021320218884778755830404817956973218636026503280 1
Число не пройшло тест пробних ділень
```

```
Згенеровано число: 954780054507481362619215281909623816265193837032036863377333842675080732245 73

Число 954780054507481362619215281909623816265193837032036863377333842675080732245 73 прошло тест Міллера-Рабіна з 15 ітераціями
Похибка: 9.3e-10

 ----------------------------------------------------------------------------------------------------

Created key: 399185872495333319878056658671083324518944358568270185390187699826404018539262834911511430950801412598851503818029115240416764809
57250684646692893251545 54
msg in int format: 399185872495333319878056658671083324518944358568270185390187699826404018539262834911511430950801412598851503818029115240416
76480957250684646692893251545 54
Encrypted: 441750020434119170928923934224045648934964648520510953484204121418063574014180659426754214071411085009372714371480551766139858490078
342048895923049865777 03
Hash: 130735416205316701068874820677853348116419942105904873009891728454242904671 87
Sign: 402207409887437306031899471874411895128290771325670342676939133747087920310619194740777858987081481678462784320441680391017535193223368
173872795553272587
msg in int format: 402207409887437306031899471874411895128290771325670342676939133747087920310619194740777858987081481678462784320441680391017
535193223368817387279555327 2587
Encrypted: 378839698286341674213854213709441896271980378465246876790627973180418289820243020485676125996361096951599318665735812463688375994 10
610265501167081110037 74
sender saved key: 399185872495333319878056658671083324518944358568270185390187699826404018539262834911511430950801412598851503818029115240416 7
648095725068464669289325154554

Decrypted: 399185872495333319878056658671083324518944358568270185390187699826404018539262834911511430950801412598851503818029115240416764809 57
250684646692893251545 54
Decrypted: 402207409887437306031899471874411895128290771325670342676939133747087920310619194740777858987081481678462784320441680391017535193 22
336881738727955532725 87
Hash: 130735416205316701068874820677853348116419942105904873009891728454242904671 87
Повідомлення і підпис співпадають
reciever recieved key 399185872495333319878056658671083324518944358568270185390187699826404018539262834911511430950801412598851503818029115240
41676480957250684646692893251545 54
```

Перевірка кожної операції шляхом взаємодії з тестовим середовищем:
Створимо публічний ключ серверу:



Щоб ключі клієнта не перегенеровувались кожного разу, згенеруємо прості числа довжини 256 біт за допомогою *gen_primes_128_256bit.py*

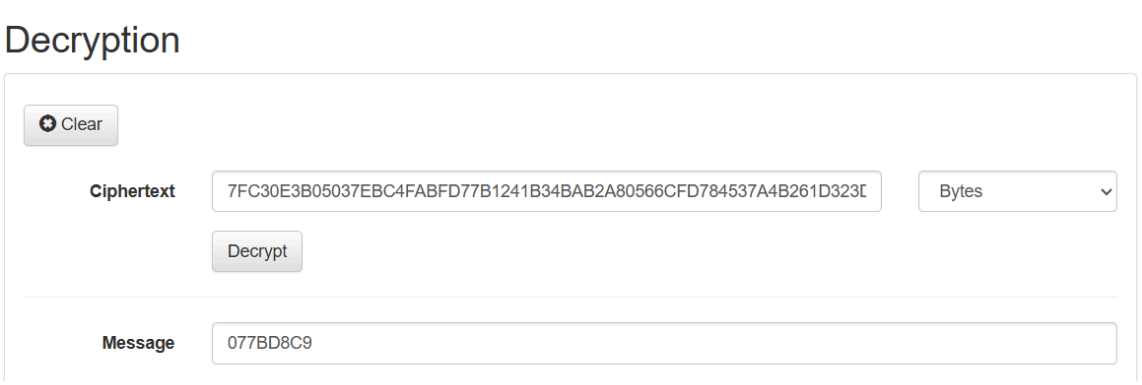
```python
# В якості клієнта будемо використовувати абонента C з pk_C, sk_C
print("CLIENT\n")
# Для фіксованого значення, обрав p, q, згенеровані gen_primes_128_256bit.py
p_C = 899528033027439732426848200911004523967695215855100837440878432288083864385013
q_C = 113185952448982343257461846858511666938310140333639326275953780654565069484579
small_primes = GenSmallPrimes()
if miller_rabin_test(p_C, 15, small_primes):
    print("q_C просте")
if miller_rabin_test(q_C, 15, small_primes):
    print("p_C просте")

pk_C, sk_C = GenKeyPair(p_C, q_C)
```

Публічні ключі сервера і клієнта в десятковому і шістнадцятковому представленні:

```
 ---------------------------------------------------------------------------------------------

SERVER

n_s hex: 9ED649B66A1845081B7AB323900688BAE2502DF15D2AEAEA9E39E24D053815AD
e_s hex: 10001

n_s: 71844044093385359615915615497416937140209825701522898686751755302431710778797
e_s: 65537

 ---------------------------------------------------------------------------------------------

CLIENT

q_C просте
p_C просте
n_C: 10181393717277041233179350702014760892843692697973384129514762638857245511157148065416429769646052803675755936918654650635041509841857280799067830722214527
e_C: 65537

n_C hex: C2659DA77B71F1CDC0B826CF80E05F5A49176029DD2FD7C38073089B9CB705EA7B3E564246ACD85F5C614CAE101E853D40D5A9529251995D644AFB8C25A3BA7F
e_A hex: 10001
```

Перевіримо операцію шифрування:

```
ШИФРУВАННЯ
msg hex: 77BD8C9
msg in int format: 125556937
Encrypted: 577883651633203239404647107954420319210152195374662251565562217256415386823 49
Encrypted hex: 7FC30E3B05037EBC4FABFD77B1241B34BAB2A80566CFD784537A4B261D323DED
```



Розшифрування:


```
ДЕШИФРУВАННЯ
Cipher hex: 5D027A8CB9FDBE4CC6DD97DB47F1CB86F3C2CB8DB569F714B020DF79B20B8E6C2C4BE2C8F9830642C6E46C30B5DFD28A37B084FA562A553E971BAB0A7DBED0F0
Cipher: 4871312336139004571642632345487345680545474010489037559780263507287581225655933511665780849048039823406166666632191209830930178370241081967409873136046320
Decrypted: I'm site
```

Перевірки цифрового підпису:

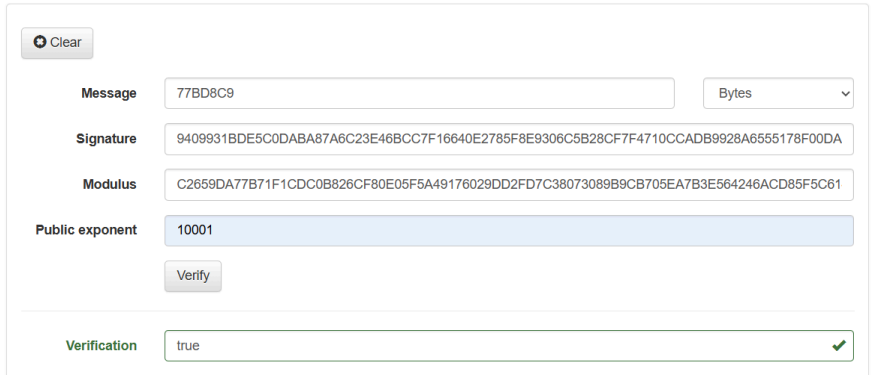
Sign


```
ПЕРЕВІРКА ПІДПИСУ
Sign hex: 34C616A3CDCDC0A7AB4A924050FE2772A2BA92D0244EDF2A6725FB2FF740BA5D
msg hex: 1234F
Sign: 23870260099720569915825432259600753299423708347047875504845729319400794798685
msg: 74575
Повідомлення і підпис співпадають
Повідомлення hex: 1234F
```

Створення цифрового підпису:

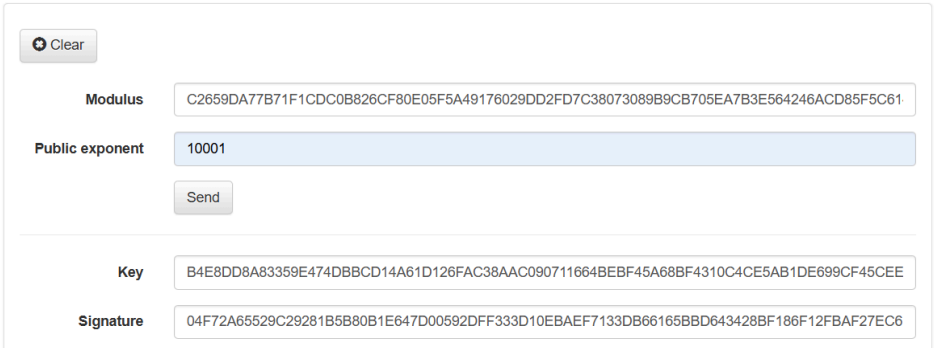
```
ПІДПИС
Sign: 7753347806355188080051208899098400115413264092226369684451014055228866093379055150845839639823796414006009818466948031503831881833864099058277289218498489
msg: 125556937
msg hex: 77BD8C9
Sign hex: 9409931BDE5C0DABA87A6C23E46BCC7F16640E2785F8E9306C5B28CF7F4710CCADB9928A6555178F00DA0723E55FB03515B67B939A5121ACF23659FAE3035FB9
```

Verify



Отримання ключа:

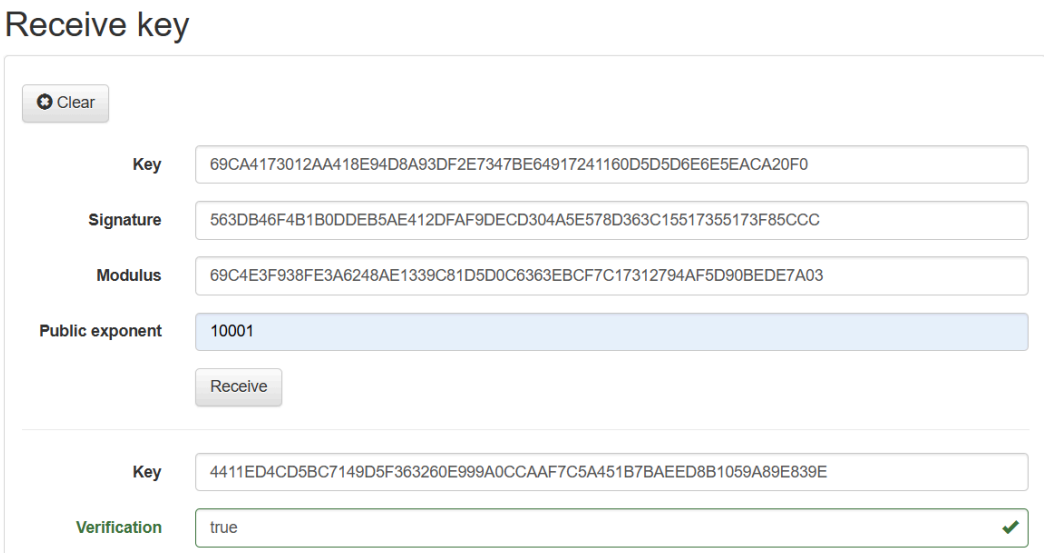
Send key




```
ОТРИМАТИ КЛЮЧ
k1 hex: B4E8DD8A83359E474DBBCD14A61D126FAC38AAC090711664BEBF45A68BF4310C4CE5AB1DE699CF45CEE8B918C387FA5FDFC8B26A89B54CBBF9805FE9ED16A2A1
s1 hex: 04F72A65529C29281B5B80B1E647D00592DFF333D10EBAEF7133DB66165BBD643428BF186F12FBAF27EC6E31AFBC3DC79A6ACF5CEC3936DBF39972CF3EFC3B36
Decrypted: 17318181518674080718
Decrypted: 21617540142335544930049271412742872347278022970069931048047909316237644572548
Повідомлення і підпис співпадають
Отримано ключ 17318181518674080718
```

Відправлення ключа:
Для наступної задачі створимо ще одного абонента з довжиною простих чисел - 128 біт (згенеруємо нові p, q за допомогою gen_primes_128_256bit.py), так як сайт працює з довжиною простих чисел значно менше за 256 біт, внаслідок чого перегенерація ключів не дасть жодного результату

```
ВІДПРАВИТИ КЛЮЧ
n_D: 47840724541389047378460163765150132463267694378875689830441929041849549224451
e_D: 65537

n_D hex: 69C4E3F938FE3A6248AE1339C81D5D0C6363EBCF7C17312794AF5D90BEDE7A03
e_D hex: 10001

Created key: 307889478891125218971254674572602768105003089875098150454036519089284987298886
msg in int format: 307889478891125218971254674572602768105003089875098150454036519089284987298886
Encrypted: 29906377451473416441547076808729219102415695793178949116115835202951180536l6
Sign: 154761518167698412890131149356240602688434501839379228116751013314525622100l9
msg in int format: 154761518167698412890131149356240602688434501839379228116751013314525622100l9
Encrypted: 24379954977449027824051799697364109913287100186104703294935753947160327200l6
Saved key: 307889478891125218971254674572602768105003089875098150454036519089284987298886
key hex: 69CA4173012AA418E94D8A93DF2E7347BE64917241160D5D5D6E6E5EACA20F0
sign hex: 563DB46F4B1B0DDEB5AE412DFAF9DECD304A5E578D363C15517355173F85CCC
```



Ура, все працює (як і очікувалось, при тому, що ми перевіряли взаємодію абонентів A, B локально. Тож, нам треба було помучитись з сайтом, який використовує представлення Hex замість десяткового, а також має менші довжини ключів, через що ключі локально доводилось перегенеровувати з меншими простими для конфіденційного розсилання ключів від клієнта до сервера)

Висновки:

У ході даного комп’ютерного практикуму, було побудовано криптосистему RSA з наступними високорівневими процедурами: GenKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), RecvKey().

Для генерації пари ключів спочатку обирається пара простих чисел фіксованого розміру бітів, у нашому випадку ми брали прості числа розміром 256 бітів. Для знаходження простого числа такого розміру, ми спочатку генеруємо випадкове число генератором псевдовипадкових чисел в діапазоні 256-бітних чисел. Далі, ми перевіряємо число на простоту, спочатку число проходить тест пробних ділень, де ми намагаємось поділити наше число на прості числа від 2 до 1000. Якщо тест провалено - число складене, якщо ні - застосовуємо до нього ймовірнісний тест Міллера-Рабіна з 15 ітераціями. Така кількість ітерацій дозволяє знаходити прості числа з точністю $e = 4^{-15} \approx 10^{-9}$.

Потім, на основі пари простих чисел, процедурою GenKeyPair() генерується пара ключів: публічний і секретний ключі

Процедури Encrypt()/Decrypt() дозволяють зашифровувати/ розшифровувати повідомлення.

Sign()/Verify() дозволяють підписати повідомлення і перевірити цифровий підпис.

SendKey()/RecvKey() дозволяють конфіденційно обмінюватись ключами з підтвердженням справжності.

Для стандартизації обміну зашифрованими повідомленнями з використанням цифрового підпису, було реалізовано процедури send(), recv().

Коректність реалізації високорівневих процедур було перевірено за допомогою тестового середовища, складність взаємодії з яким була обумовлена тим, що тестове середовище працює з шістнадцятковими числами, у той час як наша система працює з десятковими, через що для взаємодії доводилось конвертувати шістнадцяткові числа з сервера в десяткові на нашому клієнті, і навпаки.

Отже, побудована криптосистема підходить для демонстрації роботи RSA. Для практичного застосування, потрібно допрацьовувати та удосконалювати реалізовані процедури. Даний комп’ютерний практикум дозволив практично ознайомитись з механізмом роботи криптосистеми RSA.