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

Виконали:

Студент гр. ФБ-11 Падик Володимир

Та

Студент гр. ФБ-11 Ахунов Михайло

Мета та основні завдання роботи

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключ для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захис інформації на основі криптосхеми RSA, організація з використанням цієї систем засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів

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.**

На цьому етапі було розроблено інтерфейс для генерації простих чисел шляхом генерування випадкових значення та проведення тестів чи є це число простим. Було написано функцію для проведення тесту Міллера-Рабіна попередньо перевіривши пробними діленнями на декілька перших простих чисел.
Ось приклад виконання коду:

Candidate:
207485233626585534865251284731690847536685355897757979266326190998295916384 6
failed the test
Candidate:
103453682096013391350351115890902750869523248630699464414424703048482957764681
failed the test
Candidate:
829632242636293493518826226784697437031445816216321129190687292207921019706 92
failed the test
Candidate:
105859173764934330644976275768237325077229882825385223950405747170349377573967
passed the test
Candidate:
847660164628986413239597728387699654354711032275259517388671705478157268845 72
failed the test
Candidate:
922866999805521702970086513827690491246260673237022362604320427814324012140 60
failed the test
Candidate:
107670865394889930775257991148688467125177750037201207245912979436681900707831
passed the test

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* ≤ *p*1*q*1 *p q* – прості числа для побудови ключів абонента *A* і *p*1 *q*1 – абонента *B*.**

Було створено інтерфейс для перебору простими числами поки не буде досягнуто умови завдання ***pq*** ≤ ***p*1*q*1**

p=17785635007388085002987711792826431870533678094765922345939117727295465420411
,
q=99612590179303655588295003437339023277785003226139407456443581842982818410139
p1=40014900583280666903912106165384059717504473134401009982785693164350297515423
q2=10767086539488993077525799114868846712517775003720120724591297943668190070783
31

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 , B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e_1, n_1) та секретні d , d_1 .**

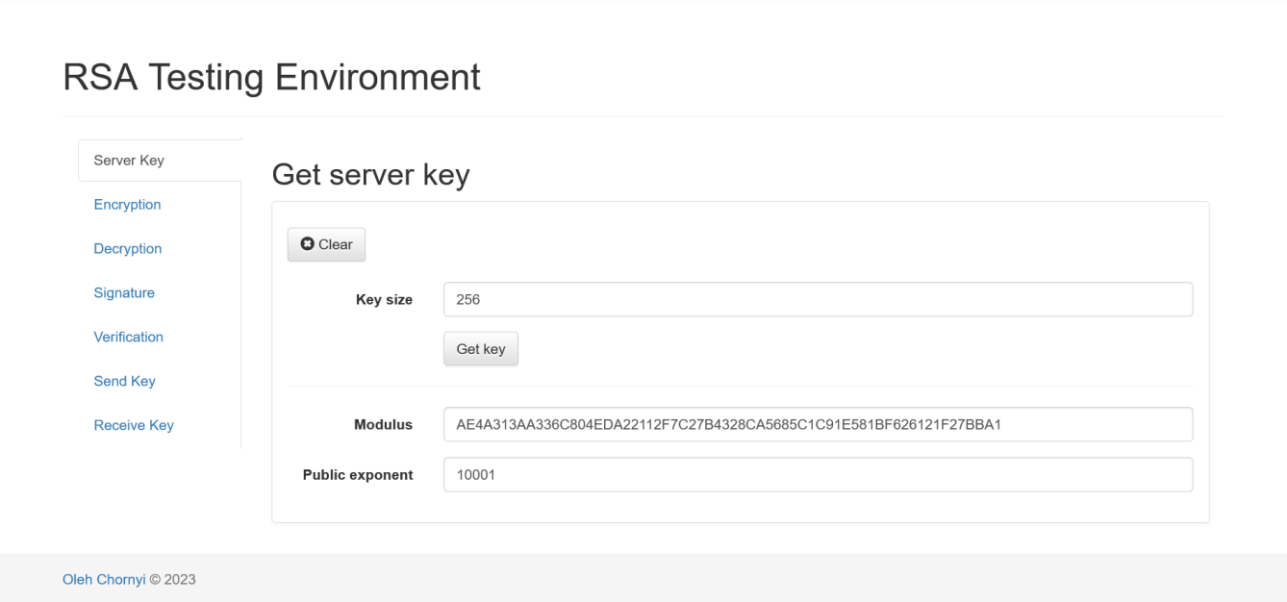
Було розроблено програмний інтерфейс для створення ключових пар для RSA.
Відкриті та секретні ключі було збереженно

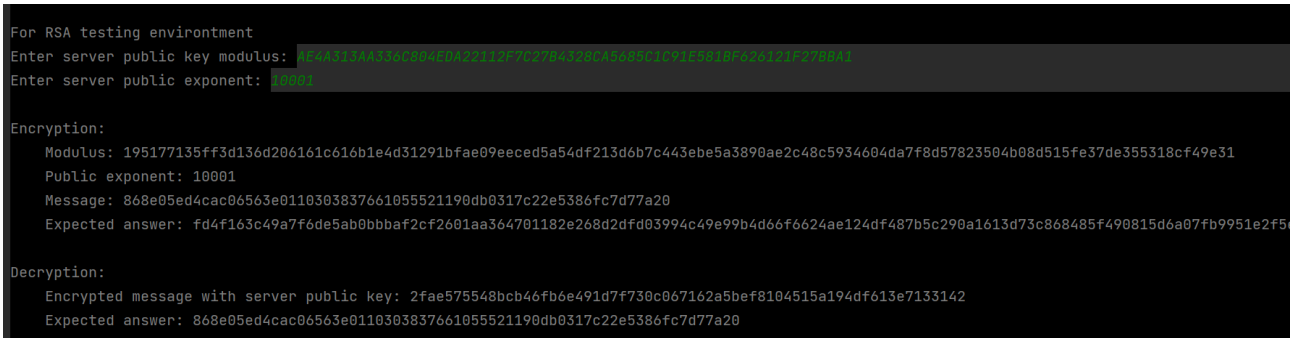
Public A key: (
n=17716731710696256560715995621499976751259828453188705003645512732993108891157
16662694185614941140385120422000629882333279201861061478096065854390259947129,
e = 65537)
Private A key: (
d=59143179633888139682216847308722781232595713398364662415087130563257890400950
73120230214755295728102052395893915319708160648114728718320494921006731079 93,
p=17785635007388085002987711792826431870533678094765922345939117727295465420411
,
q=99612590179303655588295003437339023277785003226139407456443581842982818410139
)

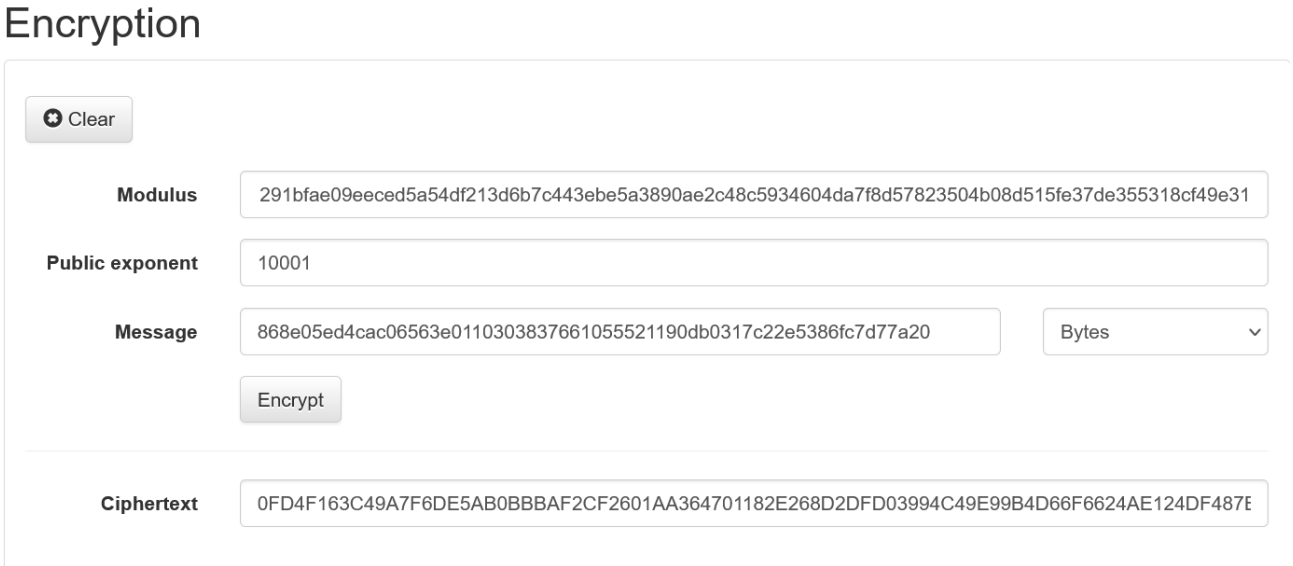
Public B key: (
n=43084389744923152646316215210465553397579149107234305319938276984146775261437
68870844773917270252760246882895603387525656180058065648537932430384339377513,
e= 65537)
Private B key: (
d=27270498426826101562086901130056793659129625452283343047159308571591261903884
78172626739172734054010201632877898915623163831984459223824285155579070133833,
p=
40014900583280666903912106165384059717504473134401009982785693164350297515423,
q=10767086539488993077525799114868846712517775003720120724591297943668190070783
1)

4. **Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A , 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 , B, перевірити правильність розшифрування. Скласти для A , B повідомлення з цифровим підписом і перевірити його.**

На цьому етапі було створено функції **Encrypt**(msg,pubKey) **Decrypt**(msg,pubKey,privKey) **Sign**(Msg, pubKey,privKey) **Verify**(msg,signed,pubKey). Кожна функція ізольована і приймає лише необхідні їй аргументи.
Було теж написано код для демонстрації роботи цих функцій ,
а також було створено зручний інтерфейс для проведення тестів у тестувальному серидовищі. Тести доводять правильність виконання функцій
Ось приклад роботи :







Шифрування вдале

Decryption

Clear

Ciphertext 2fae575548bcb46fb6e491d7f730c067162a5bef8104515a194df613e7133142 Bytes

Decrypt

Message 0868E05ED4CAC06563E011030383766105552119 0DB0317C22E5386FC7D77A20

Дешифрування вдале

Sign

Clear

Message 868e05ed4cac06563e011030383766105552119 0db0317c22e5386fc7d77a20 Bytes

Sign

Signature 2BFBBADED0E3927C7F4E0B29391B27E9130C680657C5A055C6679B3548A0E5E8

Отримую сигнатуру для підпису

```
Signature:
    Message to sign: 868e05ed4cac06563e011030383766105552119 0db0317c22e5386fc7d77a20
    Enter server signature: 2BFBBADED0E3927C7F4E0B29391B27E9130C680657C5A055C6679B3548A0E5E8
    Message verification succeed

Verification:
    Message: 868e05ed4cac06563e011030383766105552119 0db0317c22e5386fc7d77a20
    Signature: ec6bfb85e9d55bc9035e140de2738beaec8744c0b6a4437b0e60d3a9e67ac0f59fd408c9d5b64ab894657ab5fbb6698918d59eb9502806cc94b629e139799de
    Modulus: 195177135ff3d136d206161c616b1e4d31291bfae09eeced5a54df213d6b7c443ebe5a3890ae2c48c5934604da7f8d57823504b08d515fe37de355318cf49e31
    Public exponent: 10001
```

Verify

Clear

Message 868e05ed4cac06563e011030383766105552119 0db0317c22e5386fc7d77a20 Bytes

Signature :8744c0b6a4437b0e60d3a9e67ac0f59fd408c9d5b64ab894657ab5fbb6698918d59eb9502806cc94b629e139799de

Modulus 291bfae09eeced5a54df213d6b7c443ebe5a3890ae2c48c5934604da7f8d57823504b08d515fe37de355318cf49e31

Public exponent 10001

Verify

Verification true ✔

Верифікація успішна

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 $0 < k < n$**

На цьому етапі було створено функції SendKey() та RecieveKey().

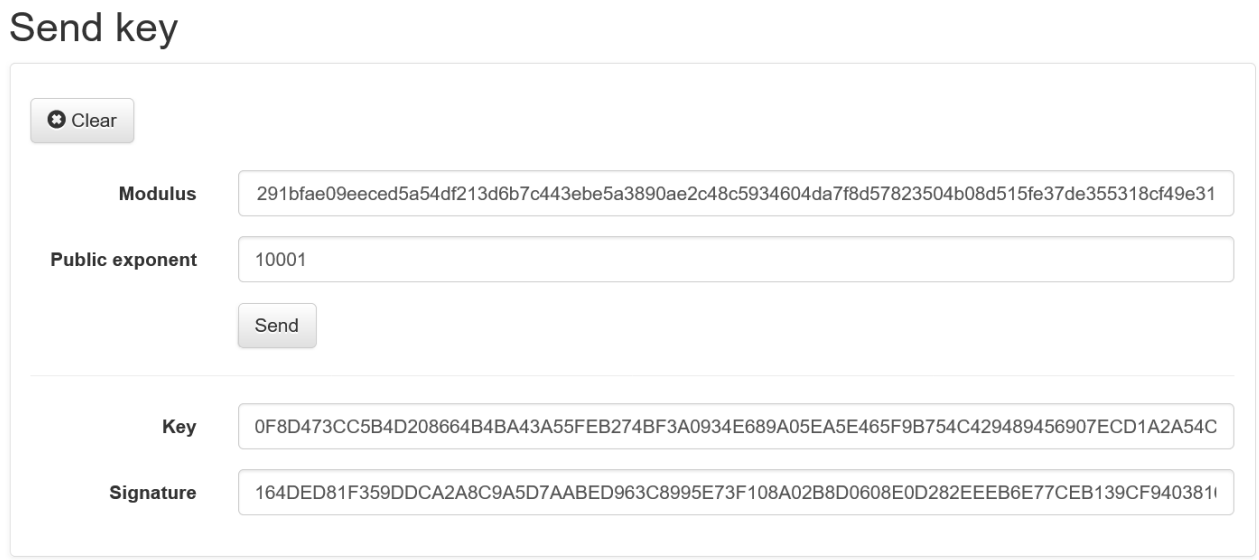
Спочатку генеруємо $0 < k < n$

SendKey підписує ключ приватним ключем надсилаючого

Потім зашифровує оригінал та підпис публічним ключем одержувача і надсилає їх відкритим каналом звязку

RecieveKey отримує повідомлення та розшифровує ключ та підпис приватним ключем одержувача . Потім перевіряє чи отримане значення ключа було підписане саме надсилаючим.

Також було проведено тест на правильність виконання функції у тестувальному середовищі .



Отримуємо ключ

Get server key

Clear

Key size 512

Get key

Modulus B72D08C2379F09ACD1F91D632C2BCBD5F8017AC35C5ABA2CE02129C6639079140F5B1419FB94B376F8AB:

Public exponent 10001

Генеруємо ключ більшої довжини

Receive key

Clear

Key af49c1e55d6739ea04547a6232c36352220ad4a2eb8c85ac355cb5cdec805d52bd8e998eb32fb284216b7a67e2688

Signature ba9bc373726bcfe330204623af5698608379fcb64ba8723a5598d48a4d1364a490fa8a4f9d3a97e0488763ab84de37

Modulus 291bfae09eeced5a54df213d6b7c443ebe5a3890ae2c48c5934604da7f8d57823504b08d515fe37de355318cf49e31

Public exponent 10001

Receive

Key 02F1D1A589DFA73C333FFA667A55A70E81F19B75763A5BF9A13AC0EE03DE1A5F811245B1F1C8A0750C09B

Verification true ✔

Розсилка ключа пройшла вдало

Для коректної перевірки потрібно обирати довжину відкритого ключа сервера таким чином щоб ключ одержувача був не меншим ніж ключ насдсилаючого .

Висновки:

Після виконання лабораторного практикуму було здобуто навички розроблення асиметричних криптосистем типу RSA . Здобуто практичні навички роботи з тестами перевірки чисел на простоту і методами генерації ключів. Збудовано систему захищеного зв’язку.