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:

студенти групи ФБ-93

Бурячок А.А.

Данілін Д.Д.

Перевірила:

Селюх П.В.

Київ - 2021

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.

Завдання:

- 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.
- За допомогою цієї функції згенерувати дві пари простих чисел $p_0,\ q_0$ і $p_1,\ q_1$ довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p_0 q_0 \leq p_1 q_1$, де $p_0,\ q_0$ - прості числа для побудови ключів абонента А, $p_1,\ q_1$ - абонента В.
- Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p, q) та відкритий ключ (e, n) . За допомогою цієї функції побудувати схеми RSA для абонентів А і В – тобто, створити та зберегти для подальшого використання відкриті ключі $(e_0,\ n_0)$, $(e_1,\ n_1)$ та секретні d_0 і d_1.
- Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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 і B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.
- 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:

Частина 1

Пишемо функцію, пошуку випадкового простого числа заданої довжини з використанням генератора псевдовипадкових чисел та тестів Міллера-Рабіна. У функції пошуку випадкового простого числа ми використовували постулат Бертрана, який пришвидшує пошук.

Частина 2

За допомогою цієї функції генеруємо дві пари простих чисел $p_0,\ q_0$ та $p_1,\ q_1$ довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p_0 q_0 \leq p_1 q_1$, де $p_0,\ q_0$ - прості числа для побудови ключів абонента A, $p_1,\ q_1$ - абонента B.

Частина 3

Пишемо функцію генерації ключових пар для RSA. Після генерування функція повертає секретний ключ (d, p, q) та відкритий ключ (e, n).

Частина 4

Реалізуємо функції шифрування, розшифрування та створення повідомлення з цифровим підписом. На вхід ці функції приймають лише ті змінні, які необхідні.

Перевірка

Перевірка викликається у функції check().

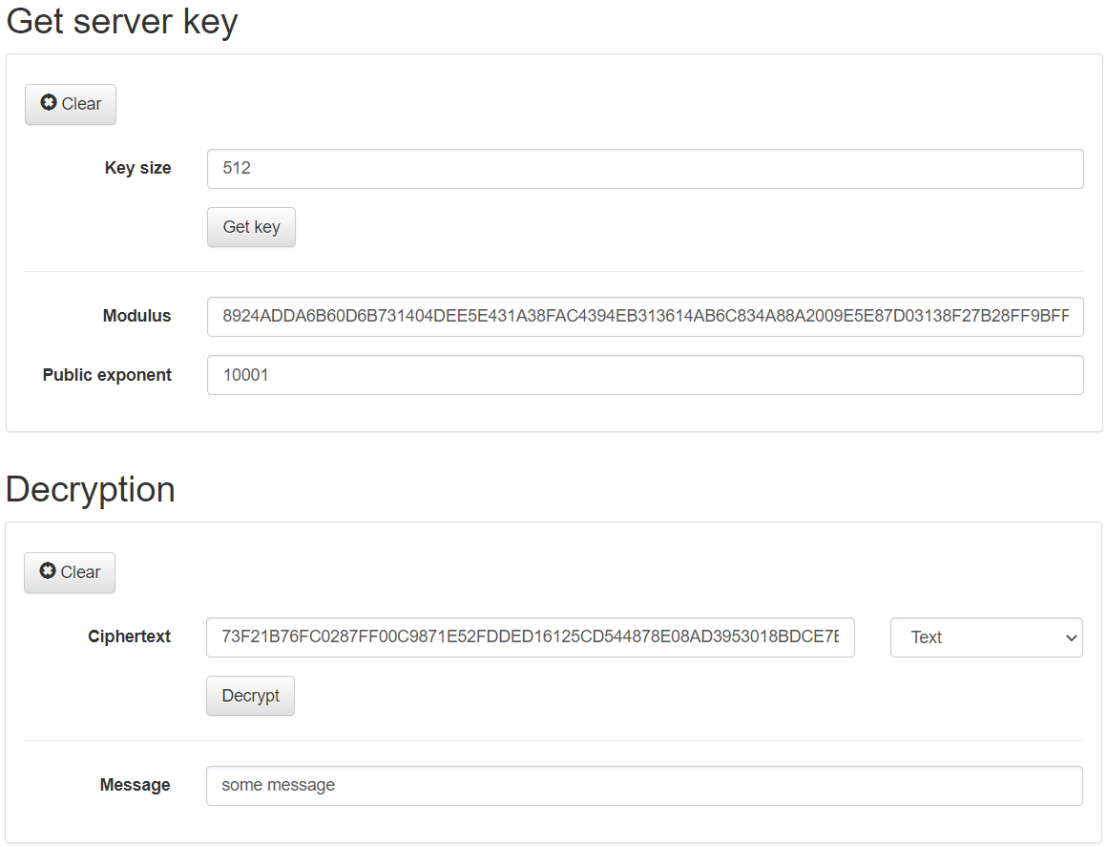
```
modulus =
"8924ADDA6B60D6B731404DEE5E431A38FAC4394EB313614AB6C834A88A2009E5E87D03138F27B28FF9BF
FA69CC06D3D2EC2C513375F2725BB978C55463C95EBD"

exponent = "10001"

n =
892365618632182408148813550464368816254855714944641228781467382616148976120823450325
1945575864814519792032321401499497921783851153183411834517642465402893

e =
282353002967861010349883537442387667494849954917040575974770142880896120506087488972
1846963083436577371845653879481281075390143374156453411628257844977759

d =
496144146083182501237322652562767802579451301083075832780633648023002762060657612857
3641064597210030770586440798368714061641497820932899289200128431851999

'''n, e, d = generate_keys()

while n < hex_to_int(modulus):

    n, e, d = generate_keys()'''

print(f"dec n = {n}")

print(f"dec e = {e}")

print(f"dec d = {d}")

print(f"hex n = {int_to_hex(n)}")

print(f"hex e = {int_to_hex(e)}")

print(f"hex d = {int_to_hex(d)}")

m = encode("some message")

def part1():

    #генерируем ключи на сервере, шифруем у себя, расшифровуем на сервере
```

```
    print("\nCheck part1")

    c = encrypt(m, hex_to_int(exponent), hex_to_int(modulus))

    print(f"hex cipher = {int_to_hex(c)}")

def part2():

    #генерируем ключи у себя, шифруем на сервере, расшифровуем у себя

    print("\nCheck part2")

    c=
"1A82B9887A65073DF4E09FB4D84085C4FECFE185A7965CA7DB6E941E654D1BF33BA10BCF165F6C64493B
B6C89675F41A7658349B262A30C2D61CEB4A8F5BDF03"

    dm = decrypt(hex_to_int(c), d, n)

    print(f"decrypted message = {decode(dm)}")

def part3():

    #генерируем цифровую подпись у себя, проверяем на сервере

    print("\nCheck part3")

    s = sign(m, d, n)

    print(f"hex signature = {int_to_hex(s)}")

def part4():

    #генерируем цифровую подпись на сервере, проверяем у себя

    print("\nCheck part4")

    s =
"2C205280A13C9E4F36356DF777163E41527EA8DF0D270BFFAA62B4C82102DAA10C30D019FC9808AC806F
2007242030CD2C2BD92B00C286940E222E8A3907952C"

    print(verify(m, hex_to_int(s), hex_to_int(exponent), hex_to_int(modulus)))

    part1()

    part2()

    part3()

    part4()
```

Результат виводу наведено нижче.

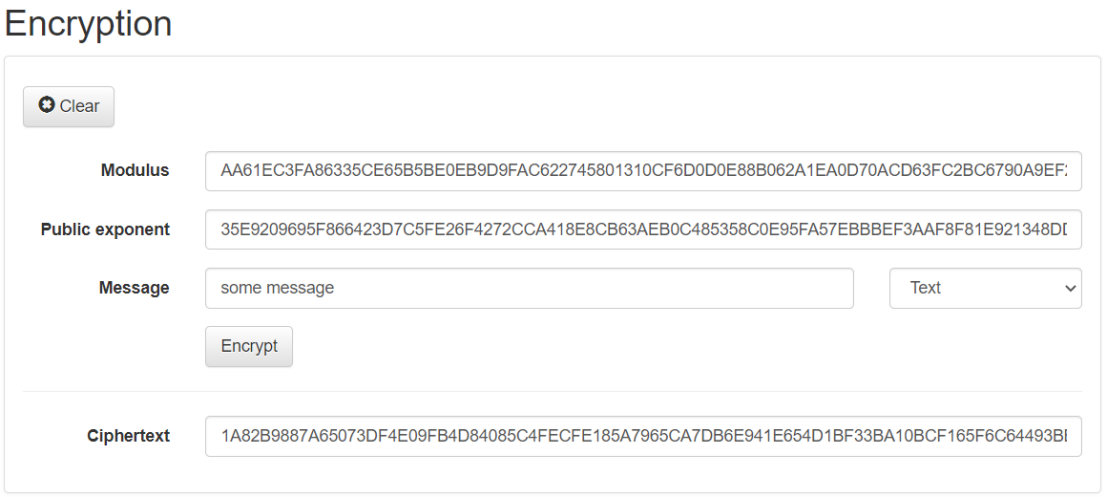
Частина 1

Генеруємо ключі на сервері, шифруємо у себе, розшифровуємо на сервері.



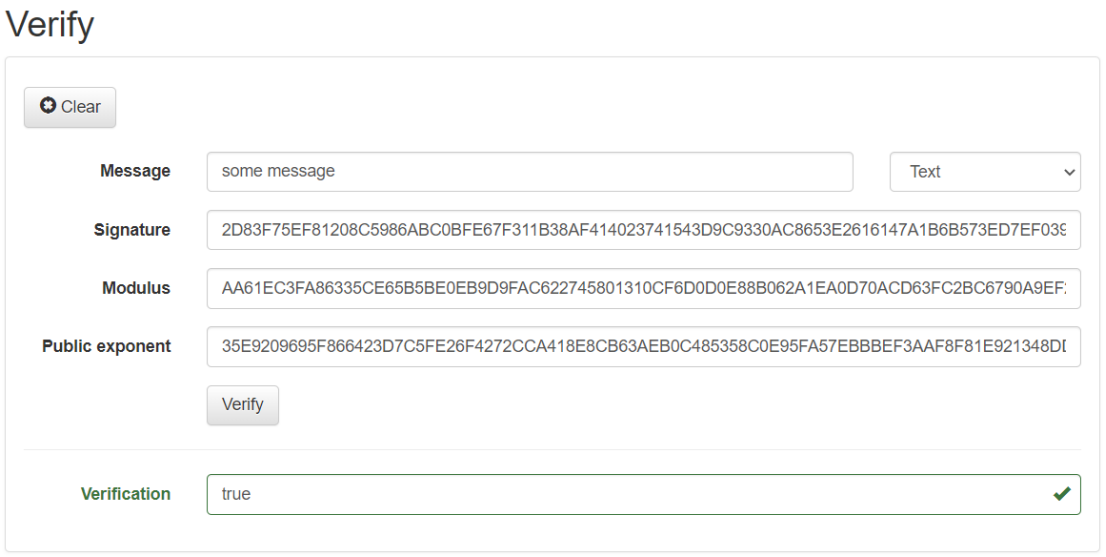
Частина 2

Генеруємо ключі у себе, шифруємо на сервері, розшифровуємо у себе.



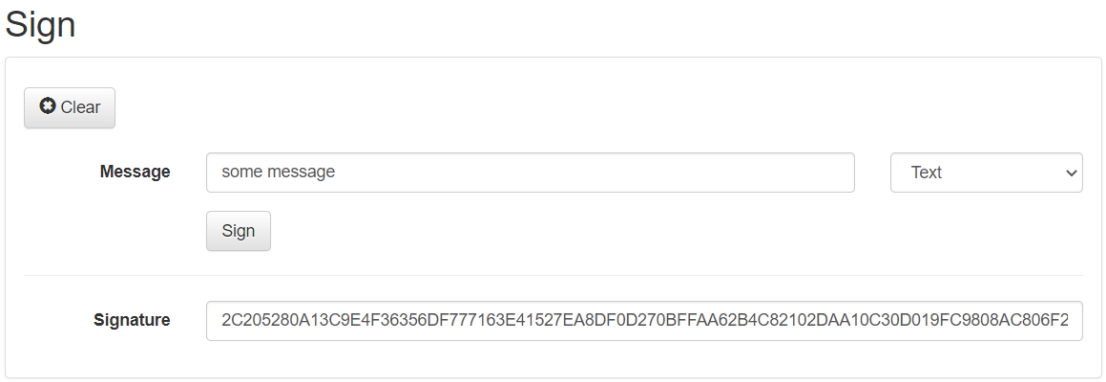
Частина 3

Генеруємо цифровий підпис у себе, перевіряємо на сервері.



Частина 4

Генеруємо цифровий підпис на сервері, перевіряємо у себе.



Демонстрація

Демонстрація викликається у функції demo().

```
sender = Abonent()

receiver = Abonent()

receiver.generate_key_receiver()

sender.generate_key_sender(receiver.n)


print("sender:")

sender.print_keys_dec()

sender.print_keys_hex()


print("receiver")

receiver.print_keys_dec()

receiver.print_keys_hex()

msg = encode("hello world")

cipher = sender.encrypt(msg, receiver.e, receiver.n)

print(decode(receiver.decrypt(cipher)))

k = encode("hello this is a private message")

ks = sender.send_key(k, receiver.e, receiver.n)

res = receiver.receive_key(ks, sender.e, sender.n)
```

```
print(decode(res))
```

Результат виводу наведено нижче.

```
sender:
dec n = 671264516307322499929091770660219899892724841296432087427157986132603024223450983647366988456704098921284922272151648852079184909185797315311908921412985
dec e = 261228917589445570781629394013654322940327945732837036595559458263799762846711633485202848661784483615152025506236533197004269983982156416367468937588575
dec d = 392700777475020346369772334860711266098876442748467559891753642665347385239570065459661045609144716861326627407709968208775356725425714501929630144404018131
hex n = 802AB9E0CAD061FC175D7DC0D1B60D75C1D994B06B443DFD329E3D07B13F64EBFD9FD8424F45E2EFFD448038588285217EAD9D1378778FFC0F71C34AF8A486C1
hex e = 31E09A7FCE5C2354405D1F3505C347CBD531FD8530B207611C085059F316C2E58C9921B53C54793B46C82ABEF7E400DFB961AD3A2A63800F5769A43DA568D9BD
hex d = 4AFAD06CE075D957F1FADAA4FE08E81AF7B78B5A5E5607EC51FA5476892A01B90C3BF2684290C955FA2DCD2F946D0E1885ECA6AE3310853C75983104452A3B95
receiver
dec n = 701778803129036283745707222237958864803595002454376238431541952071010268244620954581336035656243498982465883837921136452065370569384490965916888821467153
dec e = 525390131457110191036043721256520728947650748701199255143015226248533036781484954096033894763953811451324047605117702108041586171900495847412955239226229
dec d = 667394229874405543103846442005710923465966835378126481581115070947378505571596542879823747955281037492859735760469581041157595060663136273340518816939257
hex n = 85FE3BBD48B1646E9D677264634DD15E5DD7624E1D2018B4BE6470DF28DC8E374AE1C9863B1558B445A40EB11DDCA347DA1E395ACB202557CE07EEF4D79A50B3
hex e = 645088E0E7F5A3DA116CBE815D6E226F604CAD92D0C2625F23DEFCE6C0FB2AF4BFB32BD86FDD59292E52CA8FEC4E6448030BF0F6CCBC2834A518A2A1F5931293
hex d = 7F6D8CE8BA13F2F4FA1A4A114AB77D09112B978509A58D1A72CC3989DEE53E11A2E6B21EE8ABF3BCDE39107286CCC9230DC6B89BCD8120E0E036F5C0430BE1BB
hello world
hello this is a private message
```

Висновки: Під час виконання комп’ютерного практикуму ми ознайомилися з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA та практично ознайомилися з системою захисту інформації на основі криптосхеми RSA, організації з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчили протокол розсилання ключів.