

# **КРИПТОГРАФІЯ КОМП'ЮТЕРНИЙ ПРАКТИКУМ №5**

Вивчення криптосистеми RSA та алгоритму електронного підпису;

ознайомлення з методами генерації параметрів для асиметричних криптосистем

Виконали: ФБ-12 Карабінський Василь та Мосейко Олег

## **Мета та основні завдання 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.

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.

```
def find_p():
    while True:
        p = secrets.randbits(256)
        if miller_rabin(p, k=20):
            return p

def find_q():
    while True:
        q = secrets.randbits(256)
        if miller_rabin(q, k=20):
            return q
```

Написана дві функції `def find_q()` та `def find_p():` відповідно для генерування простих чисел  $p$  та  $q$ , для того, щоб перевіряти простоту чисел використали тест Міллера-Рабіна із попередніми пробними діленнями, код цього тесту реалізований в функції `def miller_rabin(p, k=20):`

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 А,  $p_1$  і  $q_1$  – абонента В

```
def gen_pp_qq1():
    p = find_p()
    q = find_q()
    p1 = find_p()
    q1 = find_q()
    while p*q >= p1*q1:
        return(gen_pp_qq1())
    return p, q, p1, q1
```

За допомогою цієї функції ми в подальшому згенерували дві пари чисел  $p$ ,  $q$  і  $p_1$ ,  $q_1$ , також як можемо бачити перевіряється умова  $pq \leq p_1q_1$

Згенеровані числа:

```
Згенеровані числа p і q для користувача А:  
p=69484180286085756662357980635987651746378420086802761811361930247679791845301  
q=32719116355383950405404217610558744743902793085465731676497639007202372462709  
  
Згенеровані числа p1 і q1 для користувача В:  
p1=106373821323838567366532374965240693171142663508994099660504015019052006799313  
q1=40986111285997201939553024530377876613112288107365586429648252965411227551123
```

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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(n_1, e_1)$  та секретні  $d$  і  $d_1$ .

```
def GenerateKey(p, q):  
    n = p * q  
    fn = (p-1)*(q-1)  
    e = 2**16 + 1  
    d = congruence(e, 1, fn)  
    return p, q, n, e, d
```

функція для генерації ключових пар

```
p, q, n, e, d = GenerateKey(qqq[0], qqk[1])  
p1, q1, n1, e1, d1 = GenerateKey(qqq[2], qqk[3])  
open_key_A = (n, e)  
open_key_B = (n1, e1)  
secret_key_A = (p, q, d)  
secret_key_B = (p1, q1, d1)
```

Можемо бачити результат виконання, так як ключі занадто довгі додаю їх в протокол не скріншотом.

**Відкритий ключ А**

(2273460979638915538933929588553835294085488503135097  
19816624930262974722044340009143154047985718670036585  
2069729149637854794854643889125065093058919380409,  
65537)

**Відкритий ключ В**

(4359849278695629726401464003763707245984019547482780  
76632415150086685401897289516724935547176090508629414  
8817054791054620485826000325781158105230308778499,  
65537)

### Таємний ключ А

(6948418028608575666235798063598765174637842008680276  
1811361930247679791845301,  
32719116355383950405404217610558744743902793085465731  
676497639007202372462709,  
16801971794377355406604647698491693618723871347383681  
40030551977857878856556924076656145895549399740149045  
173300843125851610387417713433545797367763735473)

### Таємний ключ В

(1063738213238385673665323749652406931711426635089940  
99660504015019052006799313,  
40986111285997201939553024530377876613112288107365586  
429648252965411227551123,  
42293272181069420305472187320029554017338914312711565  
56129483065560069033617535771398783951175151540074729  
955501233002562649744738506951900291862104715873)

4. 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 М і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

Викориснані функції `encrypt_rsa(message, public_key)`  
`decrypt_rsa(message, secret_key)`  
`signature(message, secret_key)`  
`verify_sign(sign_mes, message, public_key)`

Ключі для А та В згенеровані за допомогою процедури з минулого пункту

Результат зашифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису для користувачів А та В:  
А:

```
Оригінальне повідомлення користувача А: 123456789
Закодоване повідомлення користувачем А: 29933210321340077084670899625382150720307995108481188192989416585866043117933080608762659545193742303426327508509164481094599486159992496979075729139722
Повідомлення підписане цифровим підписом користувача А 5609570706113784563333798116240142437570045900082924659347247166480577232369731261439175071721266976280598098939081566912178032376914667635814394541077087
розкодоване повідомлення користувача А 123456789
Перевірка цифрового підпису користувача А, цифровий підпис вірний??? == True
```

додам ще і текстом

Оригінальне повідомлення користувача А: 123456789

**Закодоване повідомлення користувачем А:**

299332103213400770846708996253821507203079951004811881929894146585  
866043117933080600762659545419374230342632750850916448109459948615  
9992496979075729139722

**Повідомлення підписане цифровим підписом користувача**

А:5609570706113784563333798116240142437570045900082924659347247166  
480577232369731261439175071721266976280598098939081566912178032376  
914467635814394541077087

**розкодоване повідомлення користувача А:** 123456789

**Перевірка цифрового підпису користувача А, цифровий підпис**  
вірний??? == True

**В:**

```
Оригінальне повідомлення користувача В: 988814
Закодоване повідомлення користувачем В: 482918512390354685642489684396237748361006005161851179193335550905764419615475892414306718949664763861643522527406670073882098919341675281158423917029856
Повідомлення підписане цифровим підписом користувача В 4021112999233468163163733182674465382830364402965705832792849886179038307619104072061558909499693717820862729862190034817536530825797121484285050036116165
розкодоване повідомлення користувача В 988814
Перевірка цифрового підпису користувача В, цифровий підпис вірний??? == True
```

**Оригінальне повідомлення користувача В:** 988814

**Закодоване повідомлення користувачем В:**

482918512390354685642489684396237748361006005161851179193335550905  
764419615475892414306718949664763861643522527406670073882098919341  
675281158423917029856

**Повідомлення підписане цифровим підписом користувача В**

402111299923346816316373318267446538283036440296570583279284988617  
903830761910407206155890949969371782086272986219003481753653082579  
7121484285050036116165

**розкодоване повідомлення користувача В** 988814

**Перевірка цифрового підпису користувача В, цифровий підпис вірний???**  
== True

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().

До минулого коду були додані ще дві функції:

```
def send_key(prvt_key_A, public_key_b, msg):  
    k1 = encrypt_rsa(msg, public_key_b)  
    S = signature(msg, prvt_key_A)  
    S1 = encrypt_rsa(S, public_key_b)  
    return k1, S1  
  
def recieve_key(prvt_key_B, public_A, k1, S1):  
    k = decrypt_rsa(k1, prvt_key_B)  
    S = decrypt_rsa(S1, prvt_key_B)  
    if verify_sign(S, k, public_A) == False:  
        raise TypeError("cant verify signature")  
    else:  
        return k, S
```

Вони нам потрібні для конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA

результат роботи коду для обміну ключами:

**закодований ключ:**

242135716861858688209736554868315927439631873368610947240620288406  
715045752479911679470788956116332263215102636692450254184490369643  
86203308426132192064

**закодований підпис:**

389346642135770302245515198132784674902810908915087894994326215428  
466392524755841685556335935973583811009100771351231580201403834903  
147044066629472499963

**декодований ключ:**

446488608082769666546882890232621543706921098052135428839140088299  
800548375922537884587790133561251245774686585042939247627105801017  
623882821963651522084

**декодований підпис:**

365067103716561975441642202019553074521776686965130103374153546768  
882671038228133181721813086296779598653121600203065264746035865236  
356555250707151140521

# Перевірка реалізованого нами коду на сервері

1. Перевіримо функцію дешифрування, зашифруємо повідомлення нашим відкритим ключем А на сервері і розшифруємо локально за допомогою приватного ключа

```
хекс значення відкритого ключа А 0x23db4bbfe741eb1b1c6be4c69c812305a2e3dd1ef4021d22eed7225ce0dd4cdd8e3105d02e5f6bc291eec2ebcd27cac0fa9ed72fd75f1d042793390462b768d
```

хекс значення відкритого ключа А

```
0x23db4bbfe741eb1b1c6be4c69c812305a2e3dd1ef4021d22eed7225ce0dd4cdd8e3105d02e5f6bc291eec2ebcd27cac0fa9ed72fd75f1d042793390462b768d
```

## Encryption

Modulus

23db4bbfe741eb1b1c6be4c69c812305a2e3dd1ef4021d22eed7225ce0dd4cdd8e3105d02e5f6bc291eec2ebcd27cac0fa9ed72fd75f1d042793390462b768d

Public exponent

10001

Message

Text

Ciphertext

759D45E794DF6B83C6B9E350E328BF54EB62B70E9CA301A6D5012967D6AF906E99311B2543BE558BA8BD6

```
#Перевірка на сайті
#Завдання 1 decrypt
encrypted_hex = "759D45E794DF6B83C6B9E350E328BF54EB62B70E9CA301A6D5012967D6AF906E99311B2543BE558BA8BD613A3E21D2289A9867D54FD5F7707B288C4"
prv_key = (77000727766915613692786033325637612867113956129398433538854106165276803537433, 1524306868158978166934791472082457035704546613)
print("decrypted message:", decrypt_rsa(int(encrypted_hex, 16), prv_key, 'text'))
```

```
decrypted message: lala
```

```
Process finished with exit code 0
```

2. Перевіримо шифрування

Зашифруємо повідомлення локально, використовуючи публічний ключ сервера та розшифруємо на сервері

## Get server key

Key size

256

Get key

Modulus

88CFE8F633B942B446AD21BBB53717A9010A36EBC9C447B8B8BAC8EE1238247B

Public exponent

10001

```
#Завдання 2
msg33 = 'hello lab4'
print("оригінальне повідомлення:", msg33)
serv_pub_key = (int("88CFE8F633B942B446AD21BBB53717A9010A36EBC9C447B8B8BAC8EE1238247B", 16), int("10001", 16))
encr_serv = encrypt_rsa(msg33, serv_pub_key)
print("Зашифроване повідомлення:", encr_serv)
print("Закодоване повідомлення у хексі: ", hex(encr_serv))
```

```
оригінальне повідомлення: hello lab4
Зашифроване повідомлення: 24853908373529849507760405282972064837706387949367123040000380097286120364334
Закодоване повідомлення у хексі: 0x36f2d04a13c2694c800a9ca1060cdd38ff9956742e27a5d89d4364fcc34b592e

Process finished with exit code 0
```

## Decryption

Ciphertext

36f2d04a13c2694c800a9ca1060cdd38ff9956742e27a5d89d4364fcc34b592e

Text

Decrypt

Message

hello lab4

Як бачимо все працює вірно



### 3. Перевіримо підпис

#### Sign

Message

hvast bobra

Text ▼

Signature

053382B2DCE32C2BE22B0A0B6D6B29882055688DDC118266A1E7F4A74EA6775A

Підписуємо повідомлення використовуючи публічний ключ сервера

```
#Завдання 3
server_sign = (int("053382B2DCE32C2BE22B0A0B6D6B29882055688DDC118266A1E7F4A74EA6775A", 16))
print("повідомлення підписане сервером: ", server_sign)
message = "hvast bobra"
serv_pub_key = (int("88CFE8F633B942B446AD21B8B53717A9010A36EBC9C447B8B8BAC8EE1238247B", 16), int("10001", 16))
print("перевірка підпису: ", verify_sign(server_sign, message, serv_pub_key))
```

```
повідомлення підписане сервером: 2352575492377776600227126691795773887721261338146356570423259541924466816858
перевірка підпису: True
```

```
Process finished with exit code 0
```

### 4. Перевіримо верифікацію

```
#Завдання 4
message_to_sign = "server verify"
print("оригінальне повідомлення", message_to_sign)
print(hex(signature(message_to_sign, secret_key_A)))
print(hex(secret_key_A[0]*secret_key_A[1]))
```

```
оригінальне повідомлення server verify
0xb4e96952823ab094d24e05851b61aaf7ae5b311205e80d905374aa73467fa556b6290895eafad0db82ae3d65a5c821c688a947da5e7379ae3562d6f30d0a7
0x32167cb03bdf7cdba502ebb138277372af243df91bf1bbdf02b549c40f20359b2f2e8ae744435fd17a10768b186aaf13025681d42b79afab7eb793c249a397
```

## Verify

✖ Clear

Message

server verify

Text ▼

Signature

b4e96952823ab094d24e05851b61aaf7ae5b311205e80d905374aa73467fa556b6290895eafad0db82ae3d65a5c82

Modulus

32167cb03bdf7cdba502ebb138277372af243df91bf1bbdf02b549c40f20359b2f2e8ae744435fd17a10768b186aaf13

Public exponent

10001

Verify

Verification

true ✓

## Висновок

В ході виконання лабораторної роботи було успішно реалізовано криптоалгоритм RSA, вивчено тести простоти чисел і методи генерації ключів. Практично застосовано систему захисту інформації на основі RSA, включаючи засекречений зв'язок та електронний підпис. Вивчено протокол розсилання ключів, надаючи поглиблене розуміння безпеки комунікації. Лабораторна робота надала практичні навички в галузі асиметричних криптосистем.