

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 3 курсу  
групи ФБ-21  
КАЮН Вероніка  
РУДЮК Олександр

**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.

### Постановка задачі

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 А,  $p_1$  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 А і В – 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 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 А і В, перевірити правильність розшифрування. Скласти для А і В 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

#### Варіант 4

1. Генеруємо випадкове просте число

```
def generate_number(bit_length):
    while True:
        number = random.getrandbits(bit_length)
        number |= (1 << bit_length - 1) | 1
        if miller_rabin(number):
            return number

bit_length = 256
prime_number = generate_number(bit_length)
print(f"Згенероване просте число ({bit_length} bit): {prime_number}")
```

Функція `generate_number` використовує `random.getrandbits(bit_length)`, що генерує випадкове число довжиною `bit_length` біт. `number |= (1 << bit_length - 1) | 1` забезпечує:

- Установлення найстаршого біта (гарантує, що число має необхідну довжину).
- Установлення наймолодшого біта (гарантує, що число непарне).

Число перевіряється на простоту за допомогою функції `miller_rabin`.

```
def miller_rabin(n, k=10):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False

    # Представимо n-1 у вигляді (2^r) * d
    r, d = 0, n - 1
    while d % 2 == 0:
        r += 1
        d //= 2

    # k ітерацій тесту
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n) # a^d % n
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True
```

Числа  $n \leq 1$  не є простими.

Числа  $n=2, 3$  є простими.

Парні числа  $n > 2$  не є простими.

Розклад  $n-1$  у вигляді  $2^r d$  це робиться у циклі `while d % 2 == 0`, який ділить  $n-1$  на 2, збільшуючи  $r$  на 1, поки  $d$  залишається парним.

Основна перевірка ( $k$  ітерацій). Випадково вибирається число  $a$  з діапазону  $[2, n-2]$ . Використовується піднесення до степеня за модулем для обчислення  $x = a^d \bmod n$ . **if  $x == 1$  or  $x == n - 1$ :**  $a$  є свідком простоти числа  $n$ . Рекурсивна перевірка: Якщо  $x^2 \bmod n = n-1$  на кожному кроці,  $n$  вважається простим.

Якщо число проходить всі перевірки, воно вважається простим із високою ймовірністю.

Згенероване просте число (256 біт): 81575641111645250922011054406759458687687729611391087788670614759804575094387

2. За допомогою цієї функції згенерували дві пари простих чисел  $p$  і  $q$  — для побудови ключів абонента А,  $p_1$  і  $q_1$  — абонента В.

Абонент А:  $p = 62134537171254466919512511820159991692932331059326494868784649327292291000337$ ,  $q = 78784354574222019020508201870722725030755245451139870572001567965135149491103$   
Абонент В:  $p_1 = 59008500315852422418732402544458578100965240199090740535226129290215080363447$ ,  $q_1 = 67434160788590422198386831741757576284306511871834137029420064400676972866377$

### 3. Генеруємо ключі RSA

```
def generate_key(p, q):  
    n = p * q  
    phi = (p - 1) * (q - 1)  
    e = 65537  
    d = mod_inverse(e, phi)  
  
    return (n, e), (d, p, q)
```

Обчислюємо модуль  $n=p \cdot q$

Знаходимо функцію Ойлера  $\phi = (p - 1) * (q - 1)$

Визначаємо відкритий ключ:  $n$  і  $e$  (зазвичай 65537, так як це стандартне значення.)

Визначаємо секретний ключ:  $d$  (зворотне до  $e$  за модулем  $\phi$ ). Це реалізується через розширений алгоритм Евкліда `mod_inverse`),  $p$ ,  $q$  – наші згенеровані числа.

Далі реалізували функцію піднесення до степеня за схемою Горнера `mod_exp`

```
def mod_exp(x, alpha, m):  
    y = 1  
    while alpha > 0:  
        if alpha % 2 == 1:  
            y = (y * x) % m  
        x = (x * x) % m  
        alpha //= 2  
    return y
```

$y$  буде містити результат, спочатку  $y=1$ , бо будь-яке число в степені 0 дорівнює 1. Цикл продовжується, поки  $\alpha > 0$  (поки степінь не обнулиться).

Перевірка поточного біта  $\alpha$  - якщо поточний біт  $\alpha$  дорівнює 1 (тобто  $\alpha$  — непарне), множимо поточний результат  $y$  на  $x$  за модулем  $m$ .

Основу  $x$  підносимо до квадрату та беремо за модулем  $m$  (це відповідає переміщенню до наступного біта  $\alpha$ )

Степінь  $\alpha$  ділимо на 2 (тобто зсуваємо його двійкове представлення вправо).

Повертаємо  $y$ .

Відкритий ключ A:  $n = 4895229407885289931250719627411865709319332861859916814970749665052652135224278294245020857551998749713106923958720977276518042356504207821835532351501711$ ,  
Секретний ключ A:  $d = 1420383637011541470202537260400446134678371510766867237644136527925312922523493543032453188982592623871292482723619031401122176252285915211742857973171969$ ,

Відкритий ключ B:  $n = 39791886981927809671440923387723362184255545206305572109930231108709135448464981224950940871221791469861666759731379593518567160268249045439274122526121519$ ,  
Секретний ключ B:  $d = 591441126525411285242696545035343807371154106281798950093581307093604663630445497093304557163465757502804734348284201112767441734297641421658395227679601$ ,

$e = 65537$   
 $p = 62134537171254466919512511820159991692932331059326494868784649327292291000337$ ,  $q = 78784354574222019020508201870722725030755245451139870572001567965135149491103$

$e = 65537$   
 $p = 59008500315852422418732402544458578100965240199090740535226129290215080363447$ ,  $q = 67434160788590422198386831741757576284306511871834137029420064400676972866377$

### 4. Написали програми шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B.

```
# Зашифрування
def encrypt_RSA(M, e, n):
    return mod_exp(M, e, n)

# Розшифрування
def decrypt_RSA(C, d, n):
    return mod_exp(C, d, n)

# Створення цифрового підпису
def sign(M, d, n):
    return mod_exp(M, d, n)

# Перевірка цифрового підпису
def verify_signature(M, S, e, n):
    return M == mod_exp(S, e, n)
```

Зашифрування:  $C = M^e \bmod n$

Розшифрування:  $M = C^d \bmod n$

Створення цифрового підпису:  $S = M^d \bmod n$

Перевірка цифрового підпису:  $M = S^e \bmod n$

Абонент А:

Повідомлення М: 9290792106447832390637629774159542363827332321747663232639773899369099283918230941191331427602498405552768688729281750654008904704003795323600975772280

Зашифроване повідомлення С: 379164558981831785062483172981128678678042039838605310971552247123220264306802628585348378191438424176601578969796071064151558703083191973424031169

Розшифроване повідомлення М: 9290792106447832390637629774159542363827332321747663232639773899369099283918230941191331427602498405552768688729281750654008904704003795323600975772280

Правильність розшифрування для А: True

Цифровий підпис С: 3469901439187506820480383456597639321978670995199838651724958327860061275848058447289420092864341428846925395623433737894013185383012374032354125127

Перевірка підпису М: True

Абонент В:

Повідомлення М: 9290792106447832390637629774159542363827332321747663232639773899369099283918230941191331427602498405552768688729281750654008904704003795323600975772280

Зашифроване повідомлення С: 215815666817934919575079162748871411765236636618613067932439069969810157936747973593732321338334083036062379903936546320324790569019158155384683653269

Розшифроване повідомлення М: 9290792106447832390637629774159542363827332321747663232639773899369099283918230941191331427602498405552768688729281750654008904704003795323600975772280

Правильність розшифрування для В: True

Цифровий підпис С: 3796494902998581325876353523025878158392207460115172157536648732178489612658758995936493615444935746029631143542226191741627835247732589257561745241

Перевірка підпису М: True


Перевірили за допомогою <https://www.dcode.fr/rsa-cipher>

Абонент А:

**Results**

🔍 Decryption using C,D,N

929079210764478323906376297741595423638273323  
 217476632326397738993699092839182309411913314  
 276024984055527686868729281750654008904704003  
 795323600975772280



ДИВОВИЖНИХ СВЯТ  
 YES!SANTALINENT  
 В МАГАЗИН

RSA Cipher - dCode  
 Tag[s] : Modern Cryptography, Arithmetics  
 Share

+ | f | t | r | e

dCode and more

---

### RSA DECODER

Indicate known numbers, leave remaining cells empty.

- \* VALUE OF THE CIPHER MESSAGE (INTEGER) C =  
 379164555898183817850268483173981128678678042039...
- \* PUBLIC KEY E (USUALLY E=65537) E =  
 65537
- \* PUBLIC KEY VALUE (INTEGER) N =  
 4895229407805289931250719627411865709319332681859...
- \* PRIVATE KEY VALUE (INTEGER) D =  
 14203836370115417020253726040046134678371510766...
- \* FACTOR 1 (PRIME NUMBER) P =  
 6213453717125446691951251182015999169293233105932...
- \* FACTOR 2 (PRIME NUMBER) Q =  
 7878435474222019020520810772272503075524545113...
- \* INTERMEDIATE VALUE PHI (INTEGER) Φ =

\* DISPLAY ○ PLAINTEXT AS CHARACTER STRING  
☐ COMPUTED VALUES (C,D,E,N,P,Q,...)  
☒ PLAINTEXT AS INTEGER NUMBER  
☐ PLAINTEXT AS HEXADEXIMAL FORMAT

▶ CALCULATE/DECRYPT

---

### RSA CERTIFICATE READER

- \* CERTIFICAT (STARTING WITH -----BEGIN\_KEY-----)

▶ EXPORT VALUES

Абонент В:



Все працює правильно

5. Реалізували протокол конфіденційного розсилання ключів між двома сторонами

```
def generate_session_key(bit_length):
    return random.getrandbits(bit_length)
```

Генеруємо випадковий ключ  $k$ , який використовується як сеансовий ключ для спільної роботи.

Відправник **A** створює повідомлення для передачі  $k_1$ ,  $s_1$  і відправляє його **B**, де

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

```
# Відправник: створення повідомлення для передачі
def sender_protocol(session_key, recipient_public_key, sender_private_key, sender_public_key):
    encrypted_session_key = encrypt_RSA(session_key, recipient_public_key[1], recipient_public_key[0])
    signature = sign(session_key, sender_private_key[0], sender_public_key[0])
    return encrypted_session_key, signature
```

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

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1,$$

```
# Отримувач: знаходить (конфіденційність):
def recipient_protocol(encrypted_session_key, signature, sender_public_key, recipient_private_key, recipient_public_key):
    session_key = decrypt_RSA(encrypted_session_key, recipient_private_key[0], recipient_public_key[0])
    is_valid = verify_signature(session_key, signature, sender_public_key[1], sender_public_key[0])
    S = mod_exp(session_key, recipient_private_key[0], recipient_public_key[0])
    return session_key, is_valid, S
```

```
Ключ k1: 10916774011550799312414644699893603725611046078698185845218623998006298755787

Відправник (A) відправляє зашифрований ключ k1 та підпис s1.
Зашифрований ключ k1: 2977570074230823852640051443870707573271611928598368481750147533014591519200566575797307546990782536830679234901318527344919456681466172240881721987056528
Цифровий підпис s1: 2567057846588635624032509946794259440743229679471759187200624878641567310880226264760762055476298879402647192662259455419210092975177138279001929108908896

Отримувач (B) прийняв повідомлення.
Розшифрований ключ k: 10916774011550799312414644699893603725611046078698185845218623998006298755787
Підпис дійсний: True
Обчислений S: 3591106944979327643384837905779963146966863073010426266237661933506154946116249953003793104148951931179114633603893579639090387449636651989458010017955163
Ключ k1 переданий успішно: True
```

**Висновок:** під час виконання даної лабораторної роботи ми ознайомились з методами перевірки чисел на простоту та генерації ключів для асиметричної криптосистеми RSA. Ми здобули практичні навички в застосуванні цієї криптосистеми,

організували засекречений зв'язок та електронний підпис, а також ознайомилися з протоколом розсилання ключів у рамках цієї системи.