МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

КРИПТОГРАФІЯ
КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали роботу:
студент ФБ-23 Гнидюк Данііл
студент ФБ-23 Жушман Ілля

Київ 2024

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.

Варіант: 5

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 *B*.
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 $_1d$.
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 $0<k<n$

Хід роботи

Спочатку ми розробили функцію для генерації випадкового простого числа заданої довжини у бітах. Для перевірки числа на простоту було використано алгоритм Міллера-Рабіна з додатковим попереднім діленням на малі прості числа.

```python
import random

def is_prime_miller_rabin(n, k=5):
    """Перевірка числа n на простоту тестом Міллера-Рабіна."""
    if n < 2:
        return False
    for p in [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]:
        if n == p:
            return True
        if n % p == 0:
            return False

    r, d = 0, n - 1
    while d % 2 == 0:
        r += 1
        d //= 2

    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(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
```

```
Випадкове просте число у діапазоні 64–128 біт:
96215469936290816546118774591585618977
```

Ми спочатку згенерували випадкове число заданої довжини. Щоб гарантовано отримати непарне число потрібної довжини, ми встановили старший і молодший біти через бітові операції. Далі це число перевіряється на простоту у два етапи: спочатку проводиться пробне ділення на невеликі прості числа для швидкого відсіювання очевидно складених чисел. Після цього використовується тест Міллера-Рабіна, який перевіряє число за допомогою випадкових свідків. Якщо число успішно проходить усі перевірки, воно вважається простим і повертається як результат.

Далі ми цю функцію реалізували для генерації двох пар простих чисел

```python
def generate_two_prime_pairs(min_bits=256, max_bits=256):
    """Генерує дві пари простих чисел p,q і p1,q1 з умовою pq <= p1q1."""
    while True:
        p = generate_prime_in_bit_range(min_bits, max_bits)
        q = generate_prime_in_bit_range(min_bits, max_bits)
        pq = p * q

        p1 = generate_prime_in_bit_range(min_bits, max_bits)
        q1 = generate_prime_in_bit_range(min_bits, max_bits)
        p1q1 = p1 * q1

        if pq <= p1q1:
            return (p, q), (p1, q1)
```

Для реалізації завдання ми використали функцію для генерації випадкових простих чисел із заданим діапазоном бітів. Спочатку ми згенерували пару чисел p і q для абонента A. Далі аналогічно згенерували пару p1, q1 для абонента B. Щоб виконати умову pq ≤ p1 q1 , ми перевіряли значення добутків і у разі невідповідності перегенеровували пару p1, q1 У результаті програма повернула дві пари простих чисел.

```
Пара простих чисел для абонента A:
p = 650809009749323617582908260268027490309003666475153880383047962891591727868 21
q = 618068416730010967648166993923115869349617212475964208275738112086606041505 41
Добуток pq = 402244494249390720352106938232111445938197624562409485363342916988778478758724614603914408734371007091715184019012657856808535008429952883574413038482016 1

Пара простих чисел для абонента B:
p1 = 849947571650798714733136105825661810056312567638507745983984103801147545379 99
q1 = 974301847842554501310149349367793210285969358264986421413992261032555883031 69
Добуток p1q1 = 828105489628665179161731763695979624000982279955924756215059044407854110146385973140767671687249040039487796620726828751607018649694635259976510694261883 1
```

Далі написали функцію генерації ключових пар для RSA

```python
def generateKeyPair(bits=256):
    """Генерує пару ключів RSA: відкритий (e, n) та секретний (d, p, q)."""
    p = generate_prime_in_bit_range(bits, bits)
    q = generate_prime_in_bit_range(bits, bits)
    while p == q:
        q = generate_prime_in_bit_range(bits, bits)

    n = p * q
    phi = (p - 1) * (q - 1)

    e = 65537
    if gcd(e, phi) != 1:
        e = random.randrange(2, phi)
        while gcd(e, phi) != 1:
            e = random.randrange(2, phi)

    d = mod_inverse(e, phi)

    return (e, n), (d, p, q)

keypair_A = generateKeyPair()
keypair_B = generateKeyPair()

(e_A, n_A), (d_A, p_A, q_A) = keypair_A
(e_B, n_B), (d_B, p_B, q_B) = keypair_B

print("Ключі для абонента A:")
print(f"Відкритий ключ: (e = {e_A}, n = {n_A})")
print(f"Секретний ключ: (d = {d_A}, p = {p_A}, q = {q_A})")

print("\nКлючі для абонента B:")
print(f"Відкритий ключ: (e = {e_B}, n = {n_B})")
print(f"Секретний ключ: (d = {d_B}, p = {p_B}, q = {q_B})")
```

Цей блок відповідає за генерацію ключової пари RSA для двох абонентів A і B. Функція генерує випадкові великі прості числа p та q, обчислює модуль $n=p \cdot q$, значення $\phi(n)$, а також відкритий ключ e (стандартно 65537) та секретний ключ d — мультиплікативний обернений до e за модулем $\phi(n)$. Генеровані ключі (відкритий: e,n та секретний: d,p,q) повертаються для кожного абонента та виводяться на екран. Це дозволяє використовувати ці ключі для шифрування, розшифрування та цифрового підпису в схемі RSA.

```
Ключі для абонента A:
Відкритий ключ: (e = 65537, n = 637463384485819414094884567286972801115076978935045278483439291701605125308958346
035976006320032585548252355323283021122391611975922839154330663891916754 9)
Секретний ключ: (d = 572702543832347548445804788703933221796155109979348016415557960481847020355909046821911508589004504573002715918328641513403784862935031779092189594884509 3, p = 706383817397894546574313311781026673559405112611819561848259751977721284449 19, q = 902432033103536727072298034660719857592315876793355404077908935449595436657 71)

Ключі для абонента B:
Відкритий ключ: (e = 65537, n = 524046009388738586697800700252705109571387335819406385395079510264843443953294327761793946156376556509246920933627691435600721653438338897873189384100323 3)
Секретний ключ: (d = 198937099158985417374224996896212069838816935896531905064988985395716008393940007905192945809245988420420532301519094246573450762370449862277172354482668 9, p = 635243173048159620474283814061174959856649868036512800388266180848622574028 89, q = 824953390485330165843519491753851621060813006782054675877905353891656743070 97)
```

Далі написали програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів *A* і *B*.

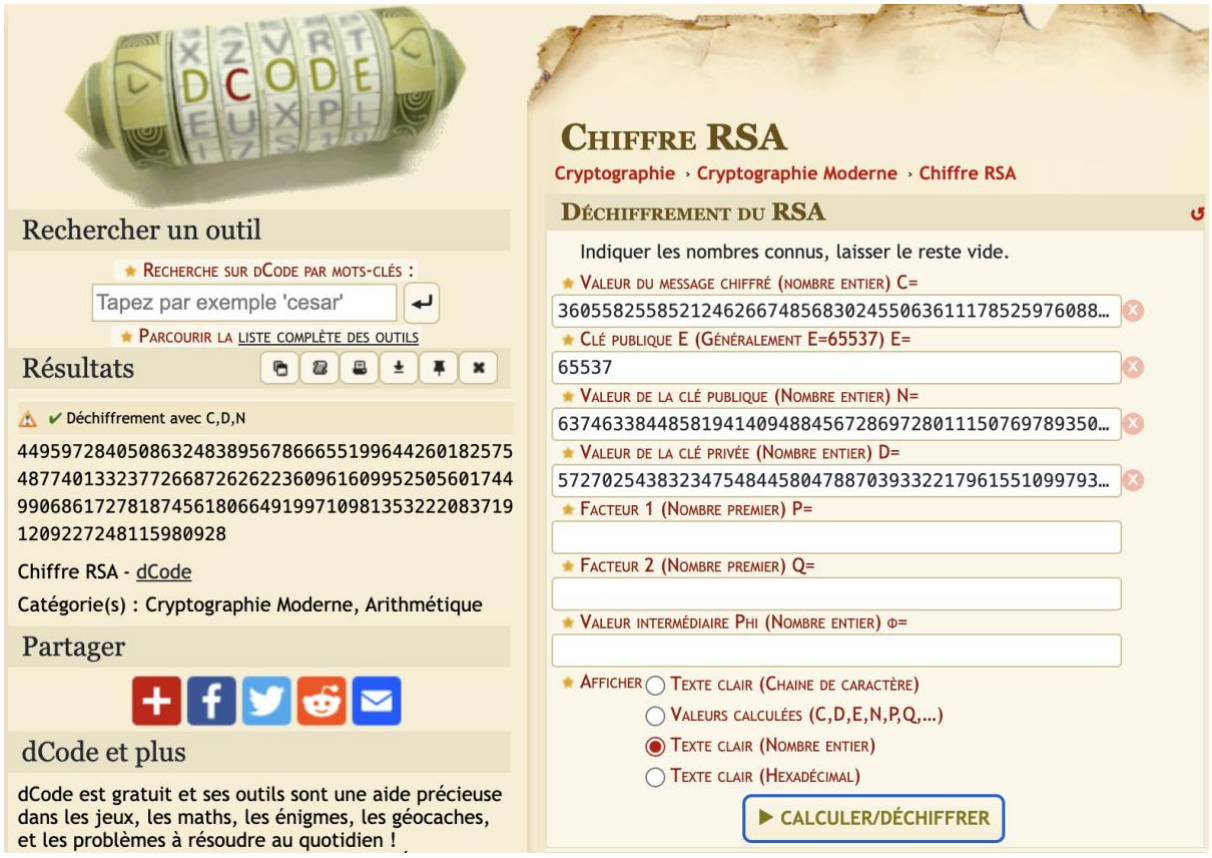
```python
def encrypt(message, e, n):
    """Шифрує повідомлення для відкритого ключа (e, n)."""
    return pow(message, e, n)

def decrypt(ciphertext, d, n):
    """Розшифровує повідомлення за секретним ключем (d, n)."""
    return pow(ciphertext, d, n)

def sign(message, d, n):
    """Створює цифровий підпис повідомлення."""
    return pow(message, d, n)

def verify(signature, e, n, original_message):
    """Перевіряє цифровий підпис."""
    return pow(signature, e, n) == original_message

def generate_random_message(n):
    """Генерує випадкове повідомлення M, яке менше за n."""
    return random.randint(1, n - 1)
```

Виконує шифрування, розшифрування, створення та перевірку цифрового підпису для двох абонентів A і B. Спочатку для кожного абонента генерується випадкове повідомлення M, яке шифрується за допомогою відкритого ключа (e,n), а потім розшифровується секретним ключем (d,n) для перевірки коректності. Далі створюється цифровий підпис повідомлення з використанням секретного ключа абонента та перевіряється його валідність за допомогою відкритого ключа. Всі операції (шифрування, розшифрування, підпис, перевірка підпису) реалізовані як окремі функції, що приймають тільки необхідні ключові дані.

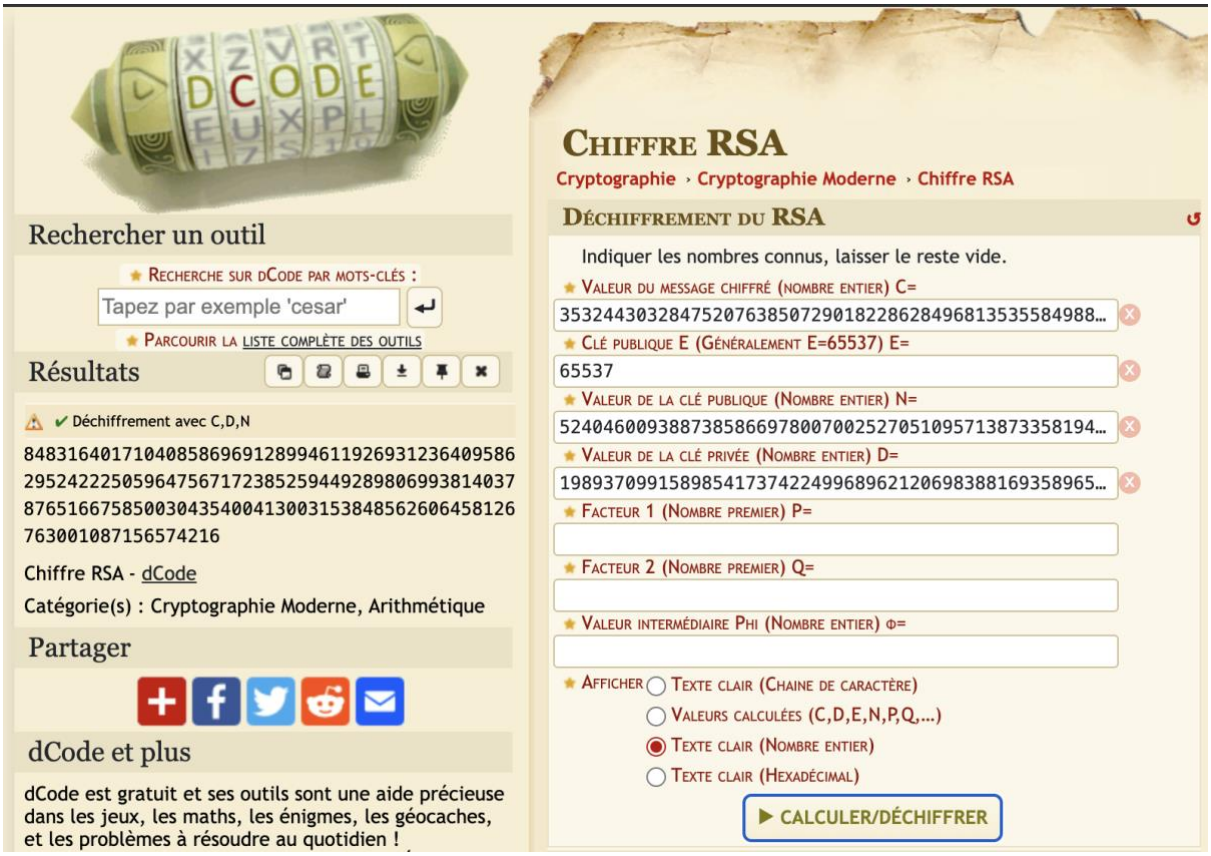
```
Відкрите повідомлення для A: 360558255852124626674856830245506361117852597608858331633672317568208836507790064708
651659707408516435798604829432324998446559875746880051694111134322502 1
Криптограма для A: 341543988898125825981618497355044158584093823968052597496229132063712807631593365600834002401 4
478324568949467767301702769086160539639626127301411887752307
Розшифроване повідомлення для A: 360558255852124626674856830245506361117852597608858331633672317568208836507790064
708651659707408516435798604829432324998446559875746880051694111134322502 1

Відкрите повідомлення для B: 353244303284752076385072901822862849681353558498852229352482835393635545758208002255
643182970572095874144985796339263933686278022691128791293386992851568 8
Криптограма для B: 136609337901634613371214619032107945967994029802854815431384178364588468008976076763584968523 8
771133557101537188782411466591504774995259490889011101488 89
Розшифроване повідомлення для B: 353244303284752076385072901822862849681353558498852229352482835393635545758208002
255643182970572095874144985796339263933686278022691128791293386992851568 8

Цифровий підпис повідомлення для A: 449597284050863248389567866551996442601825754877401332377266872626223609616 0
995250560174499068617278187456180664919971098135322208371912092272481159809 28
Перевірка підпису для A: Валідний

Цифровий підпис повідомлення для B: 848316401710408586969128994611926931236409586295242225059647567172385259449 28
980699381403787651667585003043540041300315384856260645812676300108715657421 6
Перевірка підпису для B: Валідний
```

Для перевірки правильності скористаємося онлайн-ресурсами

Для A:



Для B:



За допомогою раніше написаних на попередніх етапах програм ми організували роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA.

```python
def sendkey(key, sender_private_key, sender_public_key, receiver_public_key):
    """
    Протокол відправника: шифрує ключ для отримувача і створює цифровий підпис.
    """
    encrypted_key = encrypt(key, receiver_public_key[0], receiver_public_key[1])
    signature = sign(key, sender_private_key[0], sender_public_key[1])

    return encrypted_key, signature
```

Відправник A шифрує переданий ключ k за допомогою відкритого ключа отримувача B (e,n). Це забезпечує конфіденційність, оскільки тільки отримувач B зможе розшифрувати ключ своїм секретним ключем. Далі A створює цифровий підпис, підписуючи ключ k своїм секретним ключем d. Підпис гарантує справжність відправника та цілісність переданого ключа. В результаті функція повертає зашифрований ключ та підпис.

```python
def receivekey(encrypted_key, signature, sender_public_key, receiver_private_key,
receiver_public_key):
    """
    Протокол отримувача: розшифровує ключ і перевіряє цифровий підпис відправника.
    """
    decrypted_key = decrypt(encrypted_key, receiver_private_key[0], receiver_public_key[1])
    is_valid = verify(signature, sender_public_key[0], sender_public_key[1], decrypted_key)

    return decrypted_key, is_valid
```

Отримувач B приймає зашифрований ключ та підпис від відправника. Спочатку ключ розшифровується за допомогою секретного ключа B, що дозволяє отримати вихідний ключ k. Після цього підпис перевіряється за допомогою відкритого ключа відправника A. Якщо перевірка успішна, це підтверджує, що ключ дійсно надійшов від відправника A та не був змінений під час передачі. У результаті функція повертає розшифрований ключ та результат перевірки підпису.

```
Випадковий ключ k: 304001060416550595285431835769644747350197052810768540434639300436222444149662944584436801127
160706428760823288416354114443861674791218729786564976915372

Відправник A:
Зашифрований ключ: 54796651926462259926288225941967834414748400191594435956854567557063517067223754187275329680400
924168054027101230247545740844521229383570128484252437339888
Цифровий підпис: 719021518373147355442890088751676706348671955114062891397338752574317543854319374350315014781494
061465035333159692019025529361467263403484834794089548172

Отримувач B:
Розшифрований ключ: 304001060416550595285431835769644747350197052810768540434639300436222444149662944584436801127
716070642876082328841635411444386167479121872978656497691537 2
Перевірка підпису: Валідний

Передача ключа від A до B завершена
```

Висновки:

У процесі виконання лабораторної роботи було реалізовано перевірку чисел на простоту за допомогою тесту Міллера-Рабіна, що дозволило ефективно знаходити великі прості числа для використання в криптографічних задачах. Також було побудовано криптосистему RSA, яка включає як процеси шифрування та розшифрування даних, так і створення та перевірку цифрових підписів для забезпечення цілісності. Було реалізовано протокол конфіденційного обміну ключами з підтвердженням автентичності за допомогою криптосистеми RSA.