Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4
з дисципліни «Криптографія»

Виконав:

студент 3 курсу

НН ФТІ групи ФБ-25

Черняк Денис

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

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.
Хід 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

```python
def is_prime(n, k=10):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 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)  # 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


def generate_random_prime(bit_length):
    candidates = []
    while True:
        candidate = random.getrandbits(bit_length) | (1 << (bit_length - 1)) | 1
        if is_prime(candidate):
            return candidate, candidates
        candidates.append(candidate)
```

2.За допомогою цієї функції згенерувати дві пари простих чисел p, q і p1, q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq <= p1q1 ; p і q – прості числа для побудови ключів абонента A, p1і q1 – абонента B.

```python
def generate_prime_pairs(bit_length=256):
    while True:
        p, p_candidates = generate_random_prime(bit_length)
        q, q_candidates = generate_random_prime(bit_length)
        if p * q < generate_random_prime(bit_length * 2)[0]:
            return p, q, p_candidates, q_candidates
```

Перша пара:

```
Значення p: 101600032152859747491385239073623143444258246127820096463102378356226155994171
Значення q: 818684748889781872939308505594591020824144197879221879719845758650493703588 37
```

Друга пара:

```
Значення p: 705324970224085484381030214348218888582434147935159212524215971207932953323 47
Значення q: 100639900694751171681754425793889812159290651645633913897836524634610829585557
```

Приклад підбору кандидатів(які не підійшли):

```
Кандидати на p для A, що не пройшли перевірку простоти: [848028177792141972426385377678558972804205745021931871401155847042551264593 31, 1
Кандидати на q для A, що не пройшли перевірку простоти: [113624073754351590900649385128481885521216954654613103524621409571913568906879, 
Кандидати на p для B, що не пройшли перевірку простоти: [785487488966259453573878243933806701074735118804199864511492070266720579141 75, 1
Кандидати на q для B, що не пройшли перевірку простоти: [606820699054022178037731194704729056554474031442720504764799386363435918017 95, 1
```

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), (e1, n1) та секретні d і d1.

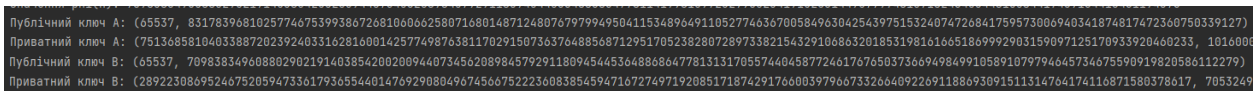
```python
def generate_rsa_keys(bit_length=256):
    p, q, p_candidates, q_candidates = generate_prime_pairs(bit_length)
    n = p * q
    phi_n = (p - 1) * (q - 1)

    print(f"Значення p: {p}")
    print(f"Значення q: {q}")
    print(f"Значення n: {n}")
    print(f"Значення phi(n): {phi_n}")

    e = 65537
    if gcd(e, phi_n) != 1:
        raise ValueError("e і phi_n не взаємно прості!")

    d = pow(e, -1, phi_n)

    return (e, n), (d, p, q), p_candidates, q_candidates
```



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.

```python
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)


def decrypt(ciphertext, private_key):
    d, p, q = private_key
    n = p * q
    return pow(ciphertext, d, n)


def sign(message, private_key):
    d, p, q = private_key
    n = p * q
    return pow(message, d, n)


def verify(signature, message, public_key):
    e, n = public_key
    return pow(signature, e, n) == message
```

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 $0 < k < n$.

```python
public_key_A, private_key_A, p_candidates_A, q_candidates_A = generate_rsa_keys()
public_key_B, private_key_B, p_candidates_B, q_candidates_B = generate_rsa_keys()

message = random.randint(1, public_key_A[1] - 1)
print("Відкрите повідомлення M:", message)

signature = sign(message, private_key_A)
print("Цифровий підпис A:", signature)

encrypted_message = encrypt(message, public_key_B)
encrypted_signature = encrypt(signature, public_key_B)
print("Зашифроване повідомлення:", encrypted_message)
print("Зашифрований підпис:", encrypted_signature)

decrypted_message = decrypt(encrypted_message, private_key_B)
decrypted_signature = decrypt(encrypted_signature, private_key_B)
print("Розшифроване повідомлення B:", decrypted_message)
print("Розшифрований підпис B:", decrypted_signature)

is_valid = verify(decrypted_signature, decrypted_message, public_key_A)
print("Валідність підпису B:", is_valid)
```

```
Відкрите повідомлення M: 271088390310743356143022039216096341557322021334804977101242037151747063083157337467791412318457465103660957216601505149013646514242781124993116821716822 5
Цифровий підпис A: 120909125757492012673613449907012607071191936611290635855895295572074924872246245125877635200775241741825441877266566054311992274347694084307243629761 30
Зашифроване повідомлення: 406940296852572021285425178485723381279783982716218667430676812045180905851122569678396380298921479379622743364566188746070655493075060774305863427417240 9
Зашифрований підпис: 47035339346192408419597138471829794637979041205452663108022373259039226583341782240306670416036549106195820576079449293671647103220096332111333397539041 94
Розшифроване повідомлення B: 271088390310743356143022039216096341557322021334804977101242037151747063083157337467791412318457465103660957216601505149013646514242781124993116821716822 5
Розшифрований підпис B: 120909125757492012673613449907012607071191936611290635855895295572074924872246245125877635200775241741825441877266566054311992274347694084307243629761 30
Валідність підпису B: True
```

Висновок:

Під час виконання роботи я дослідив принцип функціонування криптосистеми RSA та алгоритму цифрового підпису. Написав програми: 1.Для перевірки чисел на простоту за допомогою тесту Міллера–Рабіна та 2.Для створення ключів до системи типу RSA. Також перевірив роботу протоколу конфіденційного розсилання ключів з підтвердженням валідності.