НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ» ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали:
студентки групи ФБ-23
Сівашенко Анна
Тарасенко Ангеліна

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.

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 p1 , q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p1q1$; p і q – прості числа для побудови ключів абонента A, p1 і q1 – 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), (e1, n1) та секретні d і d1 .

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 написали функцію для пошуку випадкового простого числа заданої довжини.

```
def generate_prime(bit_length):
    while True:
        candidate = random.getrandbits(bit_length) | (1 << (bit_length - 1))
| 1
        if is_prime(candidate):
            return candidate
```

Тест Міллера-Рабіна із попередніми пробними діленнями:

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

    for p in [3, 5, 7, 11, 13, 17, 19, 23, 29, 31]:
        if n % p == 0 and n != p:
            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
```

У даному фрагменті коду спочатку виконуються базові перевірки: числа ≤ 1 не є простими; числа 2 та 3, тобто ≤ 3 є простими і якщо n парне (крім 2), воно складене. Далі виконується пробне ділення: перевіряється, чи n ділиться без залишку на одне з невеликих простих чисел (3, 5, 7, ... 31), і якщо n ділиться на будь-яке з них і $n \neq p$, це означає, що n складене. Тест Міллера-Рабіна використовує властивості степенів модулів для перевірки простоти, тож треба представити $n-1$ у вигляді $2^r \times d$, де d — непарне, а r — кількість множників 2. Далі виконується основна частина тесту: k разів обирається випадкове число тест вважається успішним, і алгоритм переходить до наступної кроку - перевірки через послідовність квадратів. У результаті якщо після k ітерацій не було виявлено, що n є складеним, то n вважається ймовірно простим.

Наступним кроком ми за допомогою функції *generate_prime_pairs* генеруємо дві пари простих чисел p, q для абонента A та p_1 , q_1 для абонента B, що задовольняють умову

$\cdot q \leq p_1 \cdot q_1$ та мають довжину щонайменше 256 біт:

```python
def generate_prime_pairs(bit_length):
    p = generate_prime(bit_length)
    q = generate_prime(bit_length)
    while True:
        p1 = generate_prime(bit_length)
        q1 = generate_prime(bit_length)
        if p * q <= p1 * q1:
            break
```

```
    return (p, q), (p1, q1)
```

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

Спочтаку функція обраховує значення n та φ (значення функції Ейлера, яке враховує кількість чисел, що взаємно прості з n). Для шифрування число m (повідомлення) підноситься до степеня e за модулем n:

$$\bmod n,$$

де e - публічна експонента. Тож ми генеруємо e, щоб виконувалась умова $\gcd(e, \varphi) \neq 1$. Далі генеруємо приватну експоненту – d, що використовується для дешифрування повідомлення. Вона є оберненою до e за модулем φ.

$$\cdot d \equiv 1 \bmod \varphi$$

```
def generate_rsa_keys(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    while gcd(e, phi) != 1:
        e = random.randint(2, phi - 1)

    d = pow(e, -1, phi)
    return (n, e), (d, p, q)
```

За допомогою функції *generate_rsa_keys* написали функцію *generate_rsa_schemes*, що генерує дві незалежні схеми ключів RSA для абонентів A і B.

```
def generate_rsa_schemes(bit_length=256):
    (p, q), (p1, q1) = generate_prime_pairs(bit_length)
    public_key_A, private_key_A = generate_rsa_keys(p, q)
    public_key_B, private_key_B = generate_rsa_keys(p1, q1)

    return {
        "A": {"public_key": public_key_A, "private_key": private_key_A},
        "B": {"public_key": public_key_B, "private_key": private_key_B}
    }
```

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.

Далі написали функції шифрування та розшифрування.
Шифрування, що приймає на вхід повідомлення та публічний ключ і здійснюється за формулою:

$$\bmod n$$

```
def encrypt(message, public_key):
    n, e = public_key
    assert message < n, "Повідомлення перевищує модуль n, шифрування
неможливе!"
    return pow(message, e, n)
```

Розшифрування приймає на вхід зашифрований текст та приватний ключ. Здійснюється за формулою:

$$m = c^d \bmod n$$

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

Програма для створення цифрового підпису:

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

На віхд приймає повідомлення та приватний ключ, а на виході отримуємо підпис повідомлення. Обраховується він за формулою:

Програма для перевірки цифрового підпису приймає на вхід цифровий підпис, повідомлення та публічний ключ. Перевірка здійснюється за формулою:

Повертається значення True, якщо підпис вірний, і False в іншому випадку.

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

Функція для вибору випадкового повідомлення M:

```
def generate_message(bit_length):
    return random.getrandbits(bit_length - 1)
```

Функція, що надсилає ключ. Спочатку шифруємо ключ K публічним ключем отримувача, а потім створюємо цифровий підпис для K.

```
def send_key(sender_private_key, receiver_public_key, K):
    encrypted_key = encrypt(K, receiver_public_key)
    signature = sign(K, sender_private_key)
    return encrypted_key, signature
```

Відповідно, функція, що отримує ключ. Спочатку ключ K розшифровується, а потім перевіряється на відповідність.

```
def receive_key(sender_public_key, receiver_private_key, encrypted_key,
signature):
    K = decrypt(encrypted_key, receiver_private_key)
    is_valid = verify(signature, K, sender_public_key)
    return K, is_valid
```

Реалізували клас *User*, що має атрибути: ім'я користувача, публічний ключ, приватний ключ. У ньому є метод *send_key*, де відправник шифрує ключ за допомогою публічного ключа отримувача і створює підпис цього ключа за допомогою свого приватного ключа. Цей метод повертає зашифрований ключ та підпис. Та *receive_key*: отримувач розшифровує отриманий зашифрований ключ за допомогою свого приватного ключа. Потім перевіряється підпис відправника, щоб впевнитися, що повідомлення не було змінене. На виході повертається розшифрований ключ та інформація про те, чи є підпис дійсним.

```
class User:
    def __init__(self, name, public_key, private_key):
        self.name = name
        self.public_key = public_key
        self.private_key = private_key

    def send_key(self, receiver, key):
        print(f"{self.name} надсилає ключ {receiver.name}.")
        encrypted_key = encrypt(key, receiver.public_key)
        signature = sign(key, self.private_key)
        return encrypted_key, signature

    def receive_key(self, sender, encrypted_key, signature):
        print(f"{self.name} отримує ключ від {sender.name}.")
        key = decrypt(encrypted_key, self.private_key)
        is_valid = verify(signature, key, sender.public_key)
        return key, is_valid
```

У основному блоці *main* для кожного користувача (A і B) генеруються пари ключів (публічний та приватний) за допомогою функції *generate_rsa_keys*. Далі створюються два об'єкти класу User для користувачів A і B, де кожному користувачу передаються його публічний та приватний ключі. Два випадкових повідомлення M1 і M2 генеруються за допомогою функції *generate_message*(256), де 256 біт – довжина повідомлення. Потім відбувається обмін повідолменнями:

- Спочатку користувач A надсилає повідомлення M1 користувачу B за допомогою методу *send_key*, який шифрує повідомлення і створює підпис. Потім користувач B розшифровує повідомлення за допомогою методу *receive_key* і перевіряє підпис.
- Потім користувач B надсилає повідомлення M2 користувачу A, і користувач A розшифровує його та перевіряє підпис.

В кінці перевіряється правильність розшифрованих повідомлень за допомогою assert. Якщо підпис не дійсний або розшифрування не вдалося, виводиться повідомлення про помилку. І виводяться публічні та приватні ключі обох користувачів.

```
if __name__ == "__main__":
    rsa_keys = {
        "A": generate_rsa_keys(generate_prime(256), generate_prime(256)),
        "B": generate_rsa_keys(generate_prime(256), generate_prime(256)),
```

```
    }

    user_A = User("User A", rsa_keys["A"][0], rsa_keys["A"][1], None, True)
    user_B = User("User B", rsa_keys["B"][0], rsa_keys["B"][1], None, True)

    M1 = generate_message(256)
    M2 = generate_message(256)

    print(f"\nВідкрите повідомлення M1: {M1}")
    print(f"Відкрите повідомлення M2: {M2}\n")

    encrypted_M1, signature_A = user_A.send_key(user_B, M1)
    decrypted_M1, valid_signature_A = user_B.receive_key(user_A,
encrypted_M1, signature_A)

    encrypted_M2, signature_B = user_B.send_key(user_A, M2)
    decrypted_M2, valid_signature_B = user_A.receive_key(user_B,
encrypted_M2, signature_B)

    print(f"\nРозшифроване M1 для B: {decrypted_M1}, підпис {'валідний' if
valid_signature_A else 'невалідний'}")
    print(f"Розшифроване M2 для A: {decrypted_M2}, підпис {'валідний' if
valid_signature_B else 'невалідний'}")
    assert M1 == decrypted_M1, "Розшифрування M1 некоректне!"
    assert M2 == decrypted_M2, "Розшифрування M2 некоректне!"

    print("\nКлючі користувачів:")
    print(f"User A:\n  Публічний ключ: {rsa_keys['A'][0]}\n  Приватний ключ:
{rsa_keys['A'][1]}")
    print(f"\nUser B:\n  Публічний ключ: {rsa_keys['B'][0]}\n  Приватний
ключ: {rsa_keys['B'][1]}")

    print("\nПротокол успішно завершено!")
```

Результат виконання:

```
Відкрите повідомлення M1: 4521707499439710126090754226896100078351597623566366586755875960101789247411
Відкрите повідомлення M2: 1078784835133382854923781320162308451208724885270255393492302607629045982837 7

User A надсилає ключ User B.
User B отримує ключ від User A.
User B надсилає ключ User A.
User A отримує ключ від User B.

Розшифроване M1 для B: 4521707499439710126090754226896100078351597623566366586755875960101789247411, підпис валідний
Розшифроване M2 для A: 1078784835133382854923781320162308451208724885270255393492302607629045982837 7, підпис валідний

Ключі користувачів:
User A:
  Публічний ключ: (80950162187387815953027359377394349829759797776593866196678886818625877481290132388412056015574332348581517947822560546553045442359546208149485533192472 53, 65537)
  Приватний ключ: (75215209366293752607590338616766887995077484868210429042839375222140228929392085654775818470498802564826472820724258716833391172668236498569455917506622 49, 723053216382176352438794838222555050019342652711472937165274668949108772244487, 1119560225351392000820481009812099562230759709479875743530873902165554846460 19)

User B:
  Публічний ключ: (10989147202136479190374576513236010007858129952106412932639030324884370082313202919987431040173343467196194606886236850357864056495322037313890621727755119, 65537)
  Приватний ключ: (41969933696915646754516631845567745013761538169465113707364531338305962000136444237094169554200203187463260556566034609094718227317775993231852906246769 7, 1102802371209360917081508741008050924114234591191306675831789269747647595625 67, 9964747527778257297084849636584695695682851184700947246484612788292082498645 7)

Протокол успішно завершено!
```

Висновки.

У процесі роботи було вивчено принципи роботи криптосистеми RSA: методи генерації ключів, перевірки простоти чисел та виконання операцій шифрування і розшифрування. Розглянуто алгоритм електронного підпису, а також його застосування для забезпечення автентичності та цілісності даних. Реалізували протокол обміну ключами, який включає шифрування ключа за допомогою публічного ключа отримувача, створення цифрового підпису відправником і його перевірку отримувачем.